DEPT. OF ELECTRONICS ENGINEERING & INST. OF ELECTRONICS NCTU

Computer Architecture
Lecture 6: Dynamic Scheduling for ILP (Chapter 3)

Chih-Wei Liu 劉志尉
National Chiao Tung University
cwliu@twins.ee.nctu.edu.tw

Dynamic Scheduling

- A technique that uses hardware to overcomes data hazards
  - the hardware rearranges the instruction execution to reduce the stalls while maintaining data flow and exception behavior
  - Handling some cases when dependences are unknown at compiler time (e.g. memory reference)
  - Simplify the compiler
  - (Perhaps most importantly) Allow code compiled with one pipeline run on a different pipeline
  - Will explore hardware speculation
  - But, a cost of significant increase in hardware complexity

Basic for Dynamic Scheduling

- Idea:
  - To maintain IPC≈1 by executing an instruction as early as possible
  - When stalled, other instructions can be issued and executed if they do not depend on any active or stalled instructions
- Dynamic Scheduling implies *out-of-order execution* and *out-of-order completion*

- Advantages:
  - Compiler doesn't need to have knowledge of microarchitecture
  - Handles cases where dependencies are unknown at compile time
- Disadvantage:
  - Substantial increase in hardware complexity
  - Creates the possibility for *WAR* and *WAW* hazards
  - Complicates exceptions (precise vs. *imprecise* exceptions)

OOO (Out-of-Order) Example

- In-order issue, but allow out-of-order execution (and thus out-of-order completion)

Example 1

```
DIV.D   F0, F2, F4
ADD.D   F10, F0, F8      ;  stalled
SUB.D   F12, F8, F14
```

SUB.D has dependence with neither DIV.D nor ADD.D

However, it cannot execute if **out-of-order execution** is not allowed.

Performance limitation due to hazard…

Example 2

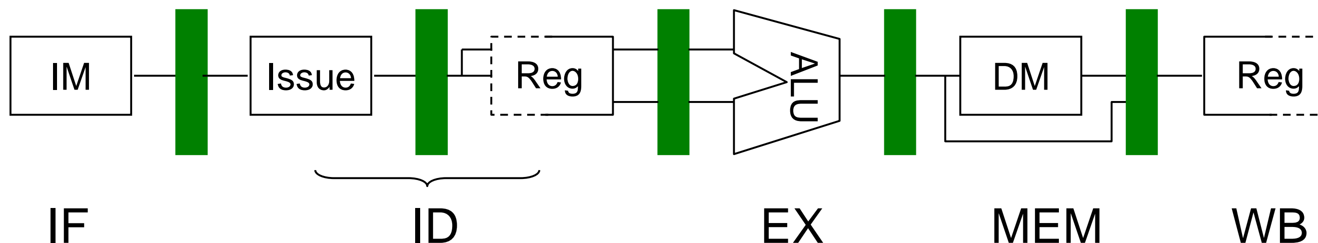
```
DIV.D   F0, F2, F4
ADD.D   F6, F0, F8      ; stalled
SUB.D   F8, F10, F14
MUL.D   F6, F10, F8
```

However, if out-of-order execution is allowed, **WAR** or **WAW** hazards could arise

Eliminating WAR and WAW hazards is essential to out-of-order execution → ***Register Renaming***

Dynamic Scheduling Introduction

- In classic 5-stage pipeline, both structural and data hazards could be checked during ID stage
  - When an instruction could execute without hazards, it was issued from ID knowing that all data hazards had been resolved.
- Let separate the ID stage into two parts
  - ***Issue***: Decode, check for structural hazard in the manner of in-order issue
  - ***Read Operands***: Wait until no data hazards, then read operands
- Out-of-order (OOO) execution
  - It may introduce WAR, WAW hazards, solved by register renaming.



Register Remaining Example

- Before:

```
fdiv.d    f0,f2,f4
fadd.d    f6,f0,f8
fsd       f6,0(x1)
fsub.d    f8,f10,f14
fmul.d    f6,f10,f8
```

Anti-dependence

- After:

```
fdiv.d    f0,f2,f4
fadd.d    S,f0,f8
fsd       S,0(x1)
fsub.d    T,f10,f14
fmul.d    f6,f10,T
```

Only RAW hazards remain

Solving WAR & WAW when Dynamic Scheduling

- Scoreboard (used in CDC6600 first, 1963)
  - Bookkeeping approach
  - Centralized control
  - Stall the instruction and keep track of dependencies between pending instructions
- Tomasulo's approach (used in IBM 360/91 Floating-point Unit, 1966)
  - Register remaining approach by using reservation registers
  - Distributed control

Scoreboard

- The scoreboard takes full responsibility for instruction issue and execution, including hazard detection
- Three parts to the scoreboard
  - Instruction status
    - Indicate the pipeline stage of the instruction
  - Functional unit status
    - 9 fields to indicate the state of the functional unit (FU)
  - Register result status
    - Indicate which FU will write the result to register

Scoreboard Example

Instruction status:

Instruction		j	k	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2				
LD	F2	45+	R3				
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
	FU									

Scoreboard Example: Cycle 1

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1			
LD	F2	45+	R3				
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F6		R2				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status:

Clock
1

	F0	F2	F4	F6	F8	F10	F12	...	F30
FU				Integer					

Scoreboard Example: Cycle 2

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2		
LD	F2	45+	R3				
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F6		R2				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
2	FU				Integer					

- Issue 2nd LD?

Scoreboard Example: Cycle 3

Instruction status:

Instruction		j	k	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	
LD	F2	45+	R3				
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	Yes	Load	F6		R2				No
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
3	*FU*				Integer					

- Issue MULT?

Scoreboard Example: Cycle 4

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3				
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
4	*FU*				Integer					

Scoreboard Example: Cycle 5

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5			
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F2		R3				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
5	*FU*		Integer							

DEPT. OF ELECTRONICS ENGINEERING & INST. OF ELECTRONICS NCTU

Scoreboard Example: Cycle 6

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6		
MULTD	F0	F2	F4	6			
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F2		R3				Yes
	Mult1	Yes	Mult	F0	F2	F4	Integer		No	Yes
	Mult2	No								
	Add	No								
	Divide	No								

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
6	FU	Mult1	Integer							

Scoreboard Example: Cycle 7

Instruction status:

Instruction		j	k	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	
MULTD	F0	F2	F4	6			
SUBD	F8	F6	F2	7			
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	Yes	Load	F2		R3				No
	Mult1	Yes	Mult	F0	F2	F4	Integer		No	Yes
	Mult2	No								
	Add	Yes	Sub	F8	F6	F2		Integer	Yes	No
	Divide	No								

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
7	*FU*	Mult1	Integer			Add				

- Read multiply operands?

Scoreboard Example: Cycle 8a

(First half of clock cycle)

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	
MULTD	F0	F2	F4	6			
SUBD	F8	F6	F2	7			
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	Yes	Load	F2		R3				No
	Mult1	Yes	Mult	F0	F2	F4	Integer		No	Yes
	Mult2	No								
	Add	Yes	Sub	F8	F6	F2		Integer	Yes	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
8	*FU*	Mult1	Integer			Add	Divide			

Dept. of Electronics Engineering & Inst. of Electronics NCTU

Scoreboard Example: Cycle 8b

(Second half of clock cycle)

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6			
SUBD	F8	F6	F2	7			
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
	Add	Yes	Sub	F8	F6	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
8	FU	Mult1				Add	Divide			

Scoreboard Example: Cycle 9

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9		
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
10	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
2	Add	Yes	Sub	F8	F6	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
9	*FU*	Mult1				Add	Divide			

- Read operands for MULT & SUB? Issue ADDD?

Scoreboard Example: Cycle 10

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9		
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	No								
9	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
1	Add	Yes	Sub	F8	F6	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
10	*FU*	Mult1				Add	Divide			

Scoreboard Example: Cycle 11

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	No								
8	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
0	Add	Yes	Sub	F8	F6	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
11	*FU*	Mult1				Add	Divide			

Scoreboard Example: Cycle 12

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	No								
7	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
	Add	No								
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
12	*FU*	Mult1					Divide			

- Read operands for DIVD?

Scoreboard Example: Cycle 13

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13			

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	No								
6	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
13	FU	Mult1			Add		Divide			

DEPT. OF ELECTRONICS ENGINEERING & INST. OF ELECTRONICS NCTU

Scoreboard Example: Cycle 14

Instruction status:

Instruction		j	k	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14		

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
5	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
2	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
14	*FU*	Mult1			Add		Divide			

Scoreboard Example: Cycle 15

Instruction status:

Instruction		j	k	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14		

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
4	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
1	Add	Yes	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
15	*FU*	Mult1			Add		Divide			

Scoreboard Example: Cycle 16

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14	16	

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	No								
3	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
0	Add	Yes	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
16	*FU*	Mult1			Add		Divide			

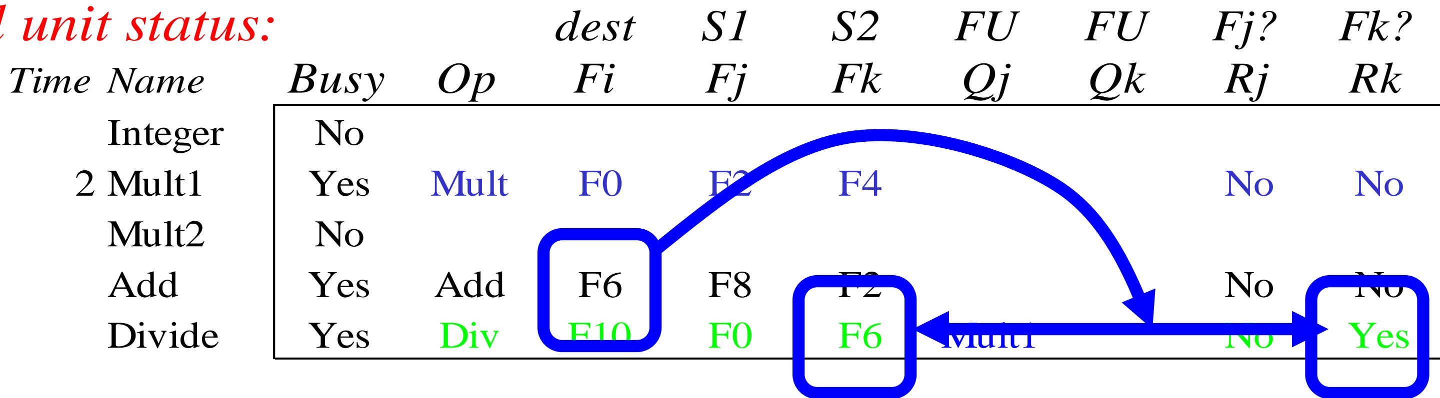
Scoreboard Example: Cycle 17

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14	16	

WAR Hazard!

Functional unit status:



Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	No								
2	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
17	*FU*	Mult1			Add		Divide			

- Why not write result of ADD???

Scoreboard Example: Cycle 18

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14	16	

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	No								
1	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
18	*FU*	Mult1			Add		Divide			

Scoreboard Example: Cycle 19

Instruction status:

Instruction		j	k	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14	16	

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
0	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
19	*FU*	Mult1			Add		Divide			

Scoreboard Example: Cycle 20

Instruction status:

Instruction		j	k	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14	16	

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6			Yes	Yes

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
20	*FU*				Add		Divide			

Scoreboard Example: Cycle 21

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21		
ADDD	F6	F8	F2	13	14	16	

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6			Yes	Yes

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
21	FU				Add		Divide			

- WAR Hazard is now gone...

Scoreboard Example: Cycle 22

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21		
ADDD	F6	F8	F2	13	14	16	22

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
39	Divide	Yes	Div	F10	F0	F6			No	No

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
22	*FU*						Divide			

skip a couple of cycles

Scoreboard Example: Cycle 61

Instruction status:

Instruction		j	k	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21	61	
ADDD	F6	F8	F2	13	14	16	22

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
0	Divide	Yes	Div	F10	F0	F6			No	No

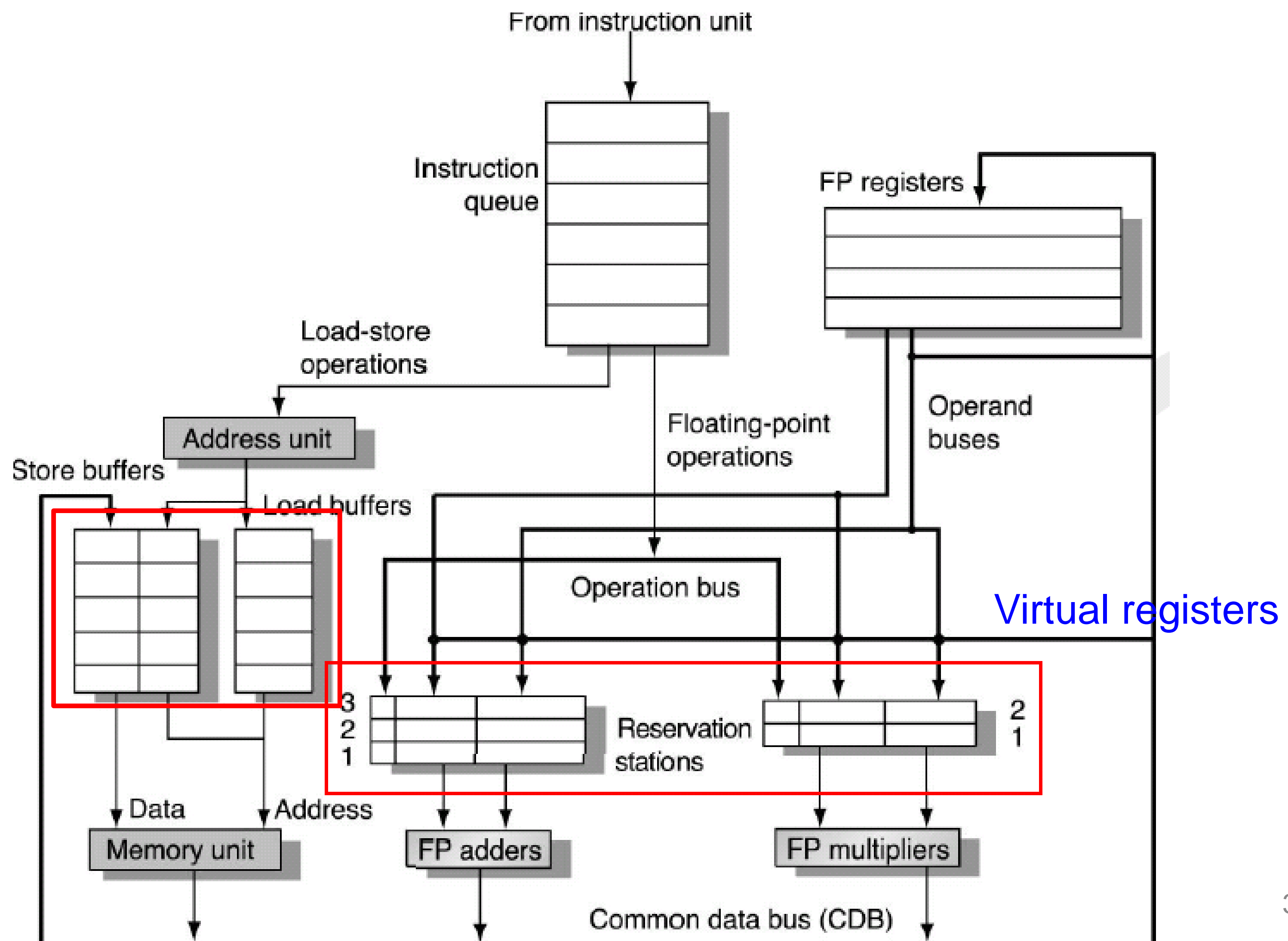
Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
61	*FU*						Divide			

Scoreboard Summary

- In-order issue and out-of-order execution/completion
- Do not issue on structural hazards
- Solution for WAR: wait for WAR hazards
  - Stall write-back until registers have been read (flag check)
  - Read registers only during Read-Operand stage
- Solution for WAW: prevent WAW hazards
  - Detect hazard and stall issue of new instruction until other instruction completes
- No register renaming
- Scoreboard replaces 3-stages, i.e. ID:EX:WB, with Issue(ID1):Read-Operand(ID2):EX:WB

Tomasulo's Algorithm



Tomasulo's Algorithm

- Virtual registers & buffers distributed with Function Units (FUs)
  - FU virtual registers, called "*reservation stations* (RSs)," have pending operands
  - Registers in instruction are renamed by pointers to RSs & buffers
    - Avoids WAR and WAW hazards
    - RSs & buffers are more than registers, so can do optimizations that compiler can't
  - Results to FU from RS, not through registers, over *common data bus* (CDB) that broadcasts to all FUs
  - Load and Store are treated as FUs with RSs as well

Reservation Station Duties

- Each RS holds an instruction that has been issued and is awaiting execution at a FU, and either the operand values or the RS names that will provide the operand values
- RS fetches operands from CDB when they appear
- When all operands are present, enable the associated functional unit to execute
- Only the last output updates the register file
- Since values are not really written to registers
  - No WAW or WAR hazards are possible

Three Stages of Tomasulo's Algorithm

1. Issue
   - Get the next instruction from the head of OP queue
     - The FIFO instruction queue (in-order issue)
   - If no RS is available
     - Structural hazards → stall the pipeline
   - If there is an available RS
     - Issue the instruction
     - If the operands are available in the RFs
       - Fetch the operands and buffer them in the RS
       - To solve WAR hazards (register renaming)
     - If the operand is not available in the RFs
       - some FU is currently computing it
       - Redirect the operand source to that reservation station
       - To solve WAW hazards (register renaming)

Three Stages of Tomasulo's Algorithm

2. Execute
   - If one of operands is not available
     - Monitor (CDB) and wait for it
     - When the operand becomes available, it is placed into the corresponding RS
   - If all operands are available
     - The operation is performed at FU
     - RAW hazards are avoided !
     - Several insts. could become ready at the same clock cycle for the same FU

- Loads and stores require 2-step execution process
  - Effective address (EA) calculation, L/S buffer for memory access
  - L/S are maintained in program order through the EA calculation, which will help to prevent hazards through memory
- To preserve exception behavior
  - No instruction is allowed to initiate execution until all branches that precede it in program order have completed.

Three Stages of Tomasulo's Algorithm

3. Write result
   - When result is available, write it on the CDB
   - When both the address and data values are available, they are sent to the memory unit

Summary for 3-stages of Tomasulo's algorithm

1. Issue—get instruction from the head of Op Queue (FIFO)

   If reservation station free (no structural hazard),
   control issues instr & sends operands (renames registers).

2. Execute—operate on operands (EX)

   When both operands ready then execute;
   if not ready, watch Common Data Bus for result

3. Write result—finish execution (WB)

   Write on Common Data Bus to all awaiting units;
   mark reservation station available

- Normal data bus: data + destination ("go to" bus)
- Common data bus: data + source ("come from" bus)
  - 64 bits of data + 4 bits of Functional Unit source address
  - Write if matches expected Functional Unit (produces result)
  - Does the broadcast

DEPT. OF ELECTRONICS ENGINEERING & INST. OF ELECTRONICS NCTU

Tomasulo's Example

Instruction stream

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2			
LD	F2	45+	R3			
MULTD	F0	F2	F4			
SUBD	F8	F6	F2			
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

3 Load/Buffers

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	No					

FU count down

3 FP Adder R.S.
2 FP Mult R.S.

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
0	*FU*									

Clock cycle counter

Tomasulo's Example Cycle 1

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1		
LD	F2	45+	R3			
MULTD	F0	F2	F4			
SUBD	F8	F6	F2			
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

	Busy	Address
Load1	Yes	34+R2
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	No					

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
1	*FU*				Load1					

Tomasulo's Example Cycle 2

Instruction status:

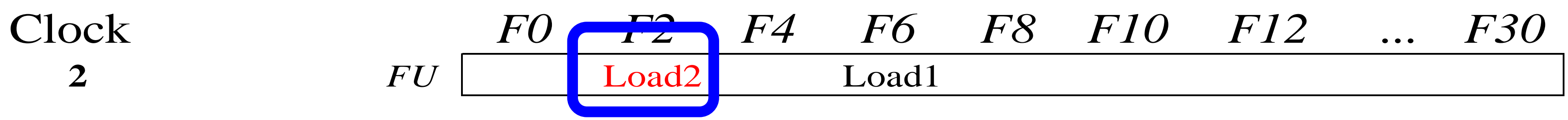
Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1		
LD	F2	45+	R3	2		
MULTD	F0	F2	F4			
SUBD	F8	F6	F2			
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

	Busy	Address
Load1	Yes	34+R2
Load2	Yes	45+R3
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	No					

Register result status:



Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
2	FU		Load2		Load1					

Note: Unlike Scoreboard, can have multiple loads outstanding

Tomasulo's Example Cycle 3

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	
LD	F2	45+	R3	2		
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2			
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

	Busy	Address
Load1	Yes	34+R2
Load2	Yes	45+R3
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	MULTD		R(F4)	Load2	
	Mult2	No					

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
3	FU	Mult1	Load2		Load1					

- Note: registers names are removed ("renamed") in Reservation Stations; MULT issued vs. scoreboard
- Load1 completing; what is waiting for Load1?

Tomasulo's Example Cycle 4

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4		
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

	Busy	Address
Load1	No	
Load2	Yes	45+R3
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	Yes	SUBD	M(A1)			Load2
	Add2	No					
	Add3	No					
	Mult1	Yes	MULTD		R(F4)	Load2	
	Mult2	No					

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
4	FU	Mult1	Load2		M(A1)	Add1				

- Load2 completing; what is waiting for Load2?

DEPT. OF ELECTRONICS ENGINEERING & INST. OF ELECTRONICS NCTU

Tomasulo's Example Cycle 5

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4		
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2			

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
2	Add1	Yes	SUBD	M(A1)	M(A2)		
	Add2	No					
	Add3	No					
10	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
5	*FU*	Mult1	M(A2)		M(A1)	Add1	Mult2			

- Timer starts down for Add1, Mult1

Tomasulo's Example Cycle 6

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4		
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6		

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
1	Add1	Yes	SUBD	M(A1)	M(A2)		
	Add2	Yes	ADDD		M(A2)	Add1	
	Add3	No					
9	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
6	FU	Mult1	M(A2)		Add2	Add1	Mult2			

- Issue ADDD here despite name dependence on F6 vs. scoreboard

Tomasulo's Example Cycle 7

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6		

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
0	Add1	Yes	SUBD	M(A1)	M(A2)		
	Add2	Yes	ADDD		M(A2)	Add1	
	Add3	No					
8	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
7	FU	Mult1	M(A2)		Add2	Add1	Mult2			

- Add1 completing; what is waiting for it?

Tomasulo's Example Cycle 8

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6		

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
2	Add2	Yes	ADDD	(M-M)	M(A2)		
	Add3	No					
7	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
8	*FU*	Mult1	M(A2)		Add2	(M-M)	Mult2			

Tomasulo's Example Cycle 9

Instruction status:

Instruction		j	k	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6		

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
	Add1	No					
1	Add2	Yes	ADDD	(M-M)	M(A2)		
	Add3	No					
6	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
9	*FU*	Mult1	M(A2)		Add2	(M-M)	Mult2			

Tomasulo's Example Cycle 10

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
0	Add2	Yes	ADDD	(M-M)	M(A2)		
	Add3	No					
5	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
10	*FU*	Mult1	M(A2)		Add2	(M-M)	Mult2			

- Add2 (ADDD) completing; what is waiting for it?

Tomasulo's Example Cycle 11

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
4	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
11	FU	Mult1	M(A2)		(M-M+M	(M-M)	Mult2			

- Write result of ADDD here vs. scoreboard?
- All quick instructions complete in this cycle!

Tomasulo's Example Cycle 12

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
3	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
12	FU	Mult1	M(A2)		(M-M+M	(M-M)	Mult2			

Tomasulo's Example Cycle 13

Instruction status:

Instruction	j	k		Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
2	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
13	FU	Mult1	M(A2)		(M-M+M	(M-M)	Mult2			

Tomasulo's Example Cycle 14

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
1	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
14	*FU*	Mult1	M(A2)		(M-M+M	(M-M)	Mult2			

Tomasulo's Example Cycle 15

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3	15	
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
0	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
15	*FU*	Mult1	M(A2)		(M-M+M	(M-M)	Mult2			

- Mult1 (MULTD) completing; what is waiting for it?

DEPT. OF ELECTRONICS ENGINEERING & INST. OF ELECTRONICS NCTU

Tomasulo's Example Cycle 16

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3	15	16
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
40	Mult2	Yes	DIVD	M*F4	M(A1)		

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
16	*FU*	M*F4	M(A2)		(M-M+M	(M-M)	Mult2			

- Now, wait for Mult2 (DIVD) to complete

Tomasulo's Example Cycle 55

Instruction status:

Instruction	j	k	Issue	Exec Comp	Write Result
LD F6	34+	R2	1	3	4
LD F2	45+	R3	2	4	5
MULTD F0	F2	F4	3	15	16
SUBD F8	F6	F2	4	7	8
DIVD F10	F0	F6	5		
ADDD F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
1	Mult2	Yes	DIVD	M*F4	M(A1)		

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
55	*FU*	M*F4	M(A2)		(M-M+M	(M-M)	Mult2			

Tomasulo's Example Cycle 56

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3	15	16
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5	56	
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
0	Mult2	Yes	DIVD	M*F4	M(A1)		

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
56	*FU*	M*F4	M(A2)		(M-M+M	(M-M)	Mult2			

- Mult2 (DIVD) is completing; what is waiting for it?

Tomasulo's Example Cycle 57

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3	15	16
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5	56	57
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	Yes	DIVD	M*F4	M(A1)		

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
56	*FU*	M*F4	M(A2)		(M-M+M	(M-M)	Result			

- Once again: In-order issue, out-of-order execution and completion.

Compare to Scoreboard Cycle 62

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21	61	62
ADDD	F6	F8	F2	13	14	16	22

Issue	Exec Comp	Write Result
1	3	4
2	4	5
3	15	16
4	7	8
5	56	57
6	10	11

- Why take longer on scoreboard/6600?
  - Structural Hazards
  - Lack of forwarding (CDB)

Summary for Tomasulo's Algorithm

- Distribution of the hazard detection logic
  - Distributed RS and CDB
  - If multiple instructions are waiting on a single result, and each already has its other operand, then the instruction can be released simultaneously by the broadcast on CDB
- Elimination of stalls for WAW and WAR
  - Rename register using RSs
  - Store operands into RS as soon as they are available
  - For WAW-hazard, the last write will win

Loop Unrolling in Tomasulo's Algorithm

```
Loop:   fld      f0, 0(x1)
        fmul.d   f4, f0, f2
        fsd      f4, 0(x1)
        addi     x1, x1, 8
        bne      x1, x2, Loop     // branches if x16 != x2
```

- Assume multiplies takes 4 clocks
- Let's assume we have issued all the instructions in two successive iterations of the loop, but none of the floating-point load/stores or operations have completed.

	Reservation stations						
Name	**Busy**	**Op**	**Vj**	**Vk**	**Qj**	**Qk**	**A**
Load1	Yes	Load					Regs[x1] + 0
Load2	Yes	Load					Regs[x1] − 8
Add1	No						
Add2	No						
Add3	No						
Mult1	Yes	MUL		Regs[f2]	Load1		
Mult2	Yes	MUL		Regs[f2]	Load2		
Store1	Yes	Store	Regs[x1]			Mult1	
Store2	Yes	Store	Regs[x1] − 8			Mult2	

	Register status							
Field	**f0**	**f2**	**f4**	**f6**	**f8**	**f10**	**f12**	**...**	**f30**
Qi	Load2		Mult2						

Figure 3.14 Two active iterations of the loop with no instruction yet completed. Entries in the multiplier reservation stations indicate that the outstanding loads are the sources. The store reservation stations indicate that the multiply destination is the source of the value to store.

Greater ILP by Speculation

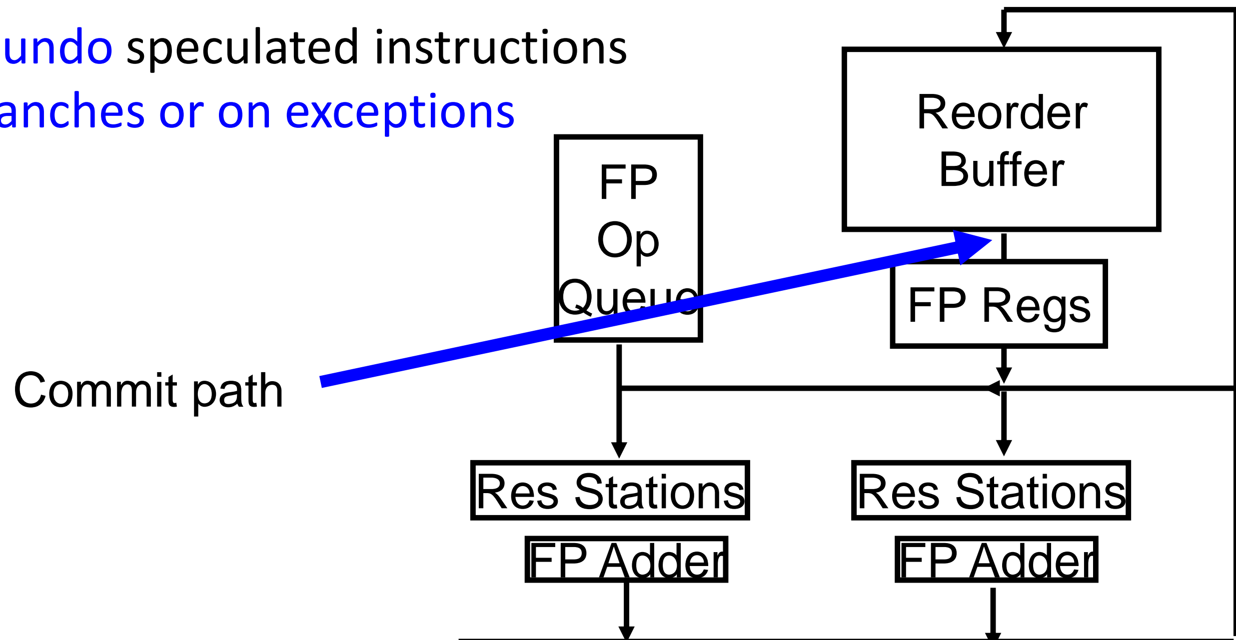
- Essential data flow execution model
  - Operations execute as soon as their operands are available
- Greater ILP
  - Overcome control dependence by hardware speculating on outcome of branches and executing program as if guesses were correct
- Prediction vs Speculation
  - Dynamic scheduling $\Rightarrow$ only fetches and issues instructions
  - Speculation $\Rightarrow$ fetch, issue, and execute instructions as if branch predictions were always correct

Drawbacks of Tomasulo's Algorithm

- Performance limited by Common Data Bus
  - Each CDB must go to multiple functional units $\Rightarrow$high capacitance, high wiring density
  - Number of functional units that can complete per cycle limited to one!
    - Multiple CDBs $\Rightarrow$ more complexity
- Non-precise interrupts!
  - Need way to resynchronize execution with instruction stream (i.e., with issue-order)
  - Easiest way is with reorder buffer (i.e. in-order completion)

Reorder Buffer (ROB) Operation

- Holds instructions in FIFO order, exactly as issued
- When instructions complete, results placed into ROB
  - Supplies operands to other instruction between execution complete & commit ⇒ more registers like RS
  - Tag results with ROB buffer number instead of reservation station
- Instructions commit ⇒ values at head of ROB placed in registers
- As a result, easy to undo speculated instructions on mispredicted branches or on exceptions



Hardware-Based Speculation

3 components of HW-based speculation:

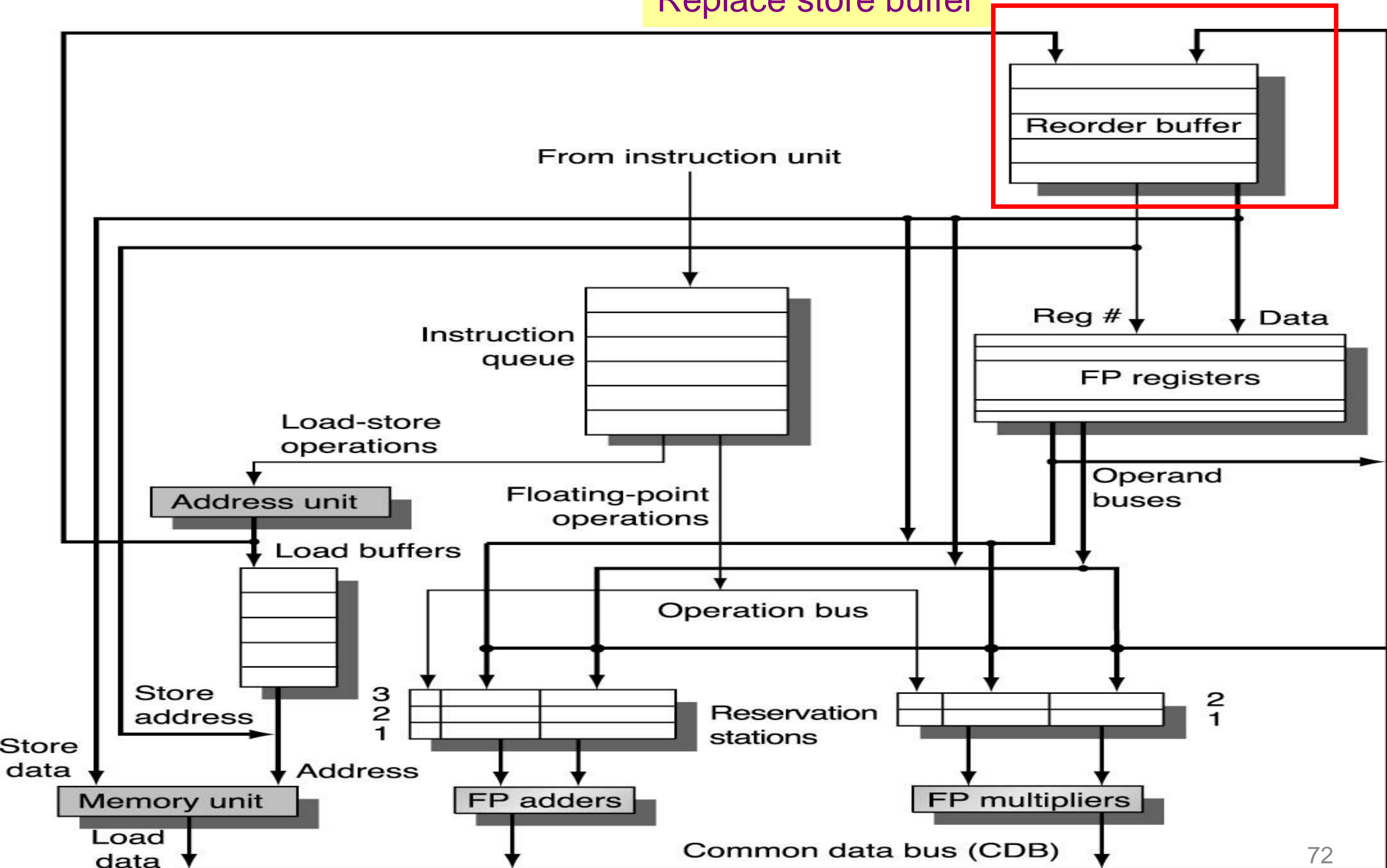
1. Dynamic branch prediction to choose which instructions to execute
2. Dynamic scheduling to deal with scheduling of different combinations of basic blocks
3. Speculation to allow execution of instructions before control dependences are resolved + ability to undo effects of incorrectly speculated sequence

- Adding ROB to Tomasulo
  - Instruction commit: when an instruction is no longer speculative, allow it to update the register file or memory
  - ROB is also used to pass results among instructions that are speculated

Reorder Buffer (ROB)

- Additional registers, just like reservation stations
  - ROB is a source of operands
  - It holds the results of instruction that have finished execution but not committed
  - Use ROB number instead of RS to indicate the source of operands when execution completes (but not committed)
  - It also uses to pass results among instructions that may be speculated
  - Each (pending) instruction occupies an ROB entry before being committed
  - Instructions in ROB are committed in order
    - Once instruction commits, the result is put into register
  - On misprediction, the corresponding ROB entry will be flushed
  - In case of exceptions: Not recognized until it is ready to commit

The Hardware Speculative Processor



Observations

- For an execution result, separate
  - data forwarding (thru RS) path
  - write-back (thru ROB) path
- Data forwarding path
  - still use RS to buffer operands
  - provide speculative register reads
  - provide out-of-order completion
- Register write-back path
  - use ROB to buffer results
  - when it’s committed, update RF (in order)

Reorder Buffer Entry

Each entry in the ROB contains four fields:

1. Instruction type
   - a branch (has no destination result), a store (has a memory address destination), or a register operation (ALU operation or load, which has register destinations)
2. Destination
   - Register number (for loads and ALU operations) or
     memory address (for stores)
     where the instruction result should be written
3. Value
   - Value of instruction result until the instruction commits
4. Ready
   - Indicates that instruction has completed execution, and the value is ready

DEPT. OF ELECTRONICS ENGINEERING & INST. OF ELECTRONICS NCTU

Four Steps of *Speculative* Tomasulo

1. Issue—get instruction from FP Op Queue

   If reservation station and reorder buffer slot free, issue instr & send operands & reorder buffer no. for destination (this stage sometimes called "dispatch")

2. Execution—operate on operands (EX)

   When both operands ready then execute; if not ready, watch CDB for result; when both in reservation station, execute; checks RAW (sometimes called "issue")

3. Write result—finish execution (WB)

   Write on Common Data Bus to all awaiting FUs & reorder buffer; mark reservation station available.

4. Commit—update register with reorder result

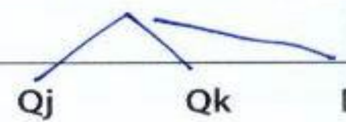
   When instr. at head of reorder buffer & result present, update register with result (or store to memory) and remove instr from reorder buffer. Mispredicted branch flushes reorder buffer (sometimes called "graduation")

Example

- The same example as Tomasulo without speculation.
  - 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

Assume
FP ADD: 2 cycles
MUL: 10 cycles
DIV: 40 cycles

- Modified status tables
  - Qj and Qk fields, and register status fields use ROB (instead of RS)
  - Add Dest field to RS (ROB to put the operation result)
- Show the status tables when MUL.D is ready to go to commit
  - At this time, only two L.D instructions have been committed



Reservation stations								
Name	**Busy**	**Op**	**Vj**	**Vk**	**Qj**	**Qk**	**Dest**	**A**
Load1	no							
Load2	no							
Add1	no							
Add2	no							
Add3	no							
Mult1	no	MUL.D	Mem[45 + Regs[R3]]	Regs[F4]			#3	
Mult2	yes	DIV.D		Mem[34 + Regs[R2]]	#3		#5	

should be removed from ROB

Reorder buffer					
Entry	**Busy**	**Instruction**	**State**	**Destination**	**Value**
1	no	L.D F6,34(R2)	Commit	F6	Mem[34 + Regs[R2]]
2	no	L.D F2,45(R3)	Commit	F2	Mem[45 + Regs[R3]]
3	yes	MUL.D F0,F2,F4	Write result	F0	#2 × Regs[F4]
4	yes	SUB.D F8,F6,F2	Write result	F8	#1 – #2
5	yes	DIV.D F10,F0,F6	Execute	F10	
6	yes	ADD.D F6,F8,F2	Write result	F6	#4 + #2

In-order commit

FP register status										
Field	**F0**	**F1**	**F2**	**F3**	**F4**	**F5**	**F6**	**F7**	**F8**	**F10**
Reorder # (ROB)	3						6		4	5
Busy	yes	no	no	no	no	no	yes	. . .	yes	yes

Precise Exceptions

- Consider the case if MUL.D causes an interrupt…
- Tomasulo without speculation
  - SUB.D and ADD.D have completed
- Tomasulo with speculation
  - No instruction after the earliest uncompleted instruction (MUL.D) is allowed to complete
  - In-order commit

- ROB with in-order instruction commit provides precise exceptions
  - Exceptions are handled in the instruction order

Memory Disambiguation Problem

- Given a load that follows a store in program order. E.g.,
  - sd 0(x2), x5
  - ld x6, 0(x3)
- Question: are the two related?
- Question: can we go ahead and start the load early?
  - We do not know whether 0(x2) $\neq$ 0(x3) in compiler time
  - Hardware-based speculation would be helpful

ROB Avoids Memory Hazards

- WAW and WAR hazards through memory are eliminated with speculation because actual updating of memory occurs in order (i.e. commits in order), when a store is at head of the ROB, and hence, no earlier loads or stores can still be pending
- RAW hazards through memory are maintained by two restrictions:
  1. not allowing a load to initiate the second step of its execution if any active ROB entry occupied by a store has a Destination field that matches the value of the A field of the load, and
  2. maintaining the program order for the computation of an effective address of a load with respect to all earlier stores.
  - These restrictions ensure that any load that accesses a memory location written to by an earlier store cannot perform the memory access until the store has written the data

3.8 Exploiting ILP Using Dynamic Scheduling, Multiple Issue, and Speculation

Multi-issue Superscalar Processor

Advantages of Superscalar over VLIW

- Old codes still run
  - Like those tools you have that came as binaries
  - HW detects whether the instruction pair is a legal dual issue pair
    - If not they are run sequentially
- Little impact on code density
  - Don't need to fill all of the can't issue here slots with NOP's
- Compiler issues are very similar
  - Still need to do instruction scheduling anyway
  - Dynamic issue hardware is there so the compiler does not have to be too conservative

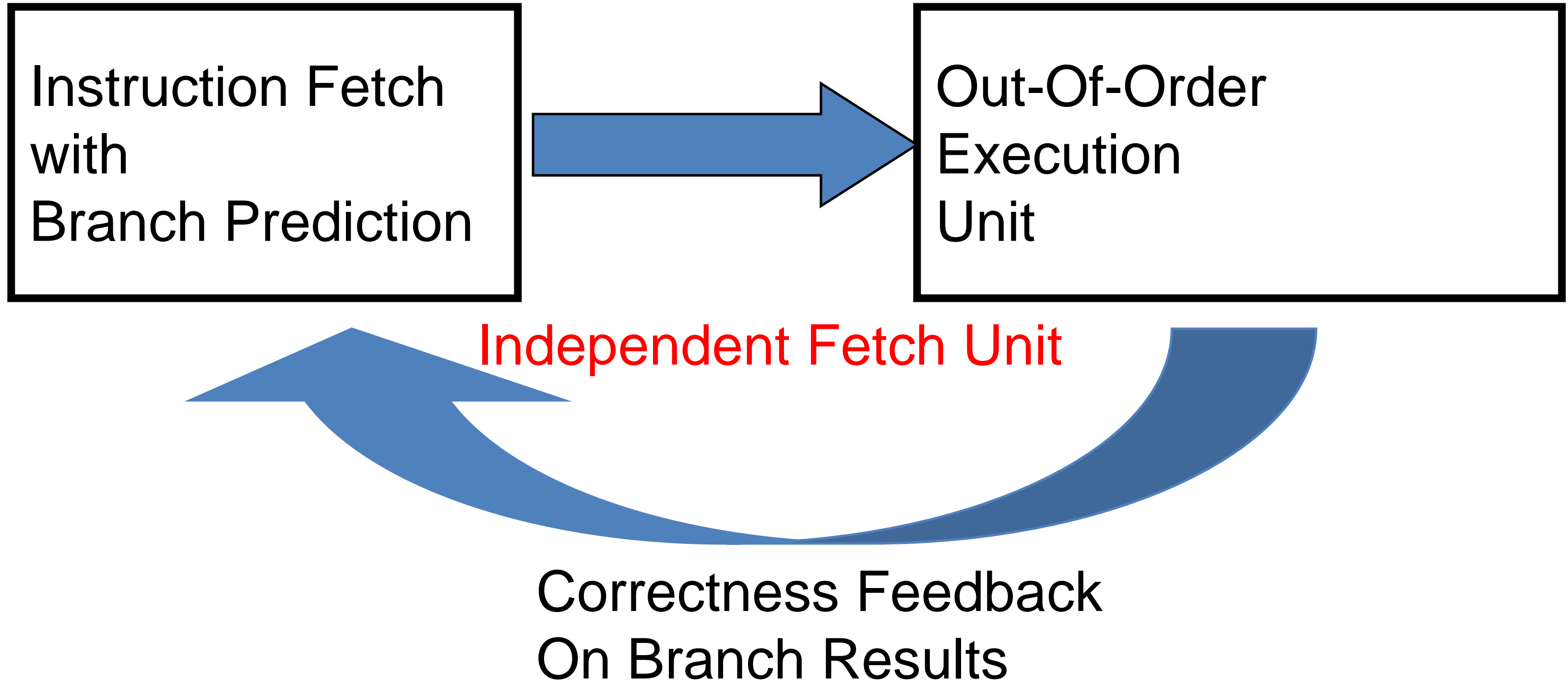
Multiple Issue with Speculation

- To maintain throughput of greater than one instructions per cycle, we must handle multiple instructions commit per clock
- Extend Tomasulo's speculation algorithm to multiple-issue scheme
  - 2 challenges
    - Instruction issue will be the bottleneck
    - Monitor CDB for instruction completion
  - In addition,
    - How to handle multiple instruction commits per clock cycle?

Dynamic Scheduling, Multiple Issue, and Speculation

- Two approaches to issue multiple instructions:
  - Assign reservation stations and update pipeline control table in half clock cycles
    - Only supports 2 instructions per clock
    - Cannot extend easily to handle 4 or more instructions per clock
  - Design logic to handle any possible dependencies between instructions
    - Examine all the dependencies among the instructions in the bundle
    - If dependencies exist in bundle, encode them in reservation stations
- Issue logic is the bottleneck in dynamically scheduled superscalars

Multi-issue Superscalar Processor



- Instruction fetch decoupled from execution
- Often issue logic (+ rename) included with Fetch and branch prediction

Example

```
Loop:  ld      x2, 0(x1)          //x2=array element
       addi    x2, x2, 1          //increment x2
       sd      x2, 0(x1)          //store result
       addi    x1, x1, 8          //increment pointer
       bne     x2, x3, Loop       //branch if not last
```

- Assume separate integer FUs:
  - for effective address calculation,
  - ALU operations, and
  - branch condition evaluation
- Assume up to 2 instructions of any type can commit per clock

No Speculation

Iteration number	Instructions	Issues at clock cycle number	Executes at clock cycle number	Memory access at clock cycle number	Write CDB at clock cycle number	Comment
1	ld x2,0(x1)	1	2	3	4	First issue
1	addi x2,x2,1	1	5		6	Wait for ld
1	sd x2,0(x1)	2	3	7		Wait for addi
1	addi x1,x1,8	2	3		4	Execute directly
1	bne x2,x3,Loop	3	7			Wait for addi
2	ld x2,0(x1)	4	8	9	10	Wait for bne
2	addi x2,x2,1	4	11		12	Wait for ld
2	sd x2,0(x1)	5	9	13		Wait for addi
2	addi x1,x1,8	5	8		9	Wait for bne
2	bne x2,x3,Loop	6	13			Wait for addi
3	ld x2,0(x1)	7	14	15	16	Wait for bne
3	addi x2,x2,1	7	17		18	Wait for ld
3	sd x2,0(x1)	8	15	19		Wait for addi
3	addi x1,x1,8	8	14		15	Wait for bne
3	bne x2,x3,Loop	9	19			Wait for addi

Out-of-order executing

Speculation

Iteration number	Instructions	Issues at clock number	Executes at clock number	Read access at clock number	Write CDB at clock number	Commits at clock number	Comment
1	ld x2,0(x1)	1	2	3	4	5	First issue
1	addi x2,x2,1	1	5		6	7	Wait for ld
1	sd x2,0(x1)	2	3			7	Wait for addi
1	addi x1,x1,8	2	3		4	8	Commit in order
1	bne x2,x3,Loop	3	7			8	Wait for addi
2	ld x2,0(x1)	4	5	6	7	9	No execute delay
2	addi x2,x2,1	4	8		9	10	Wait for ld
2	sd x2,0(x1)	5	6			10	Wait for addi
2	addi x1,x1,8	5	6		7	11	Commit in order
2	bne x2,x3,Loop	6	10			11	Wait for addi
3	ld x2,0(x1)	7	8	9	10	12	Earliest possible
3	addi x2,x2,1	7	11		12	13	Wait for ld
3	sd x2,0(x1)	8	9			13	Wait for addi
3	addi x1,x1,8	8	9		10	14	Executes earlier
3	bne x2,x3,Loop	9	13			14	Wait for addi

Out-of-order executing

In-order committing

Comparisons

- Without speculation (Tomasulo only)
  - `ld` following `bne` cannot start execution earlier

  → wait until branch outcome is determined
  - Completion rate is falling behind the issue rate rapidly, stall when a few more iterations are issued
- With speculation
  - `ld` following `bne` can start execution early because it is speculative
  - More complex HW is required
  - Completion rate is almost equal to issue rate

Advanced Techniques for Instruction Delivery and Speculation

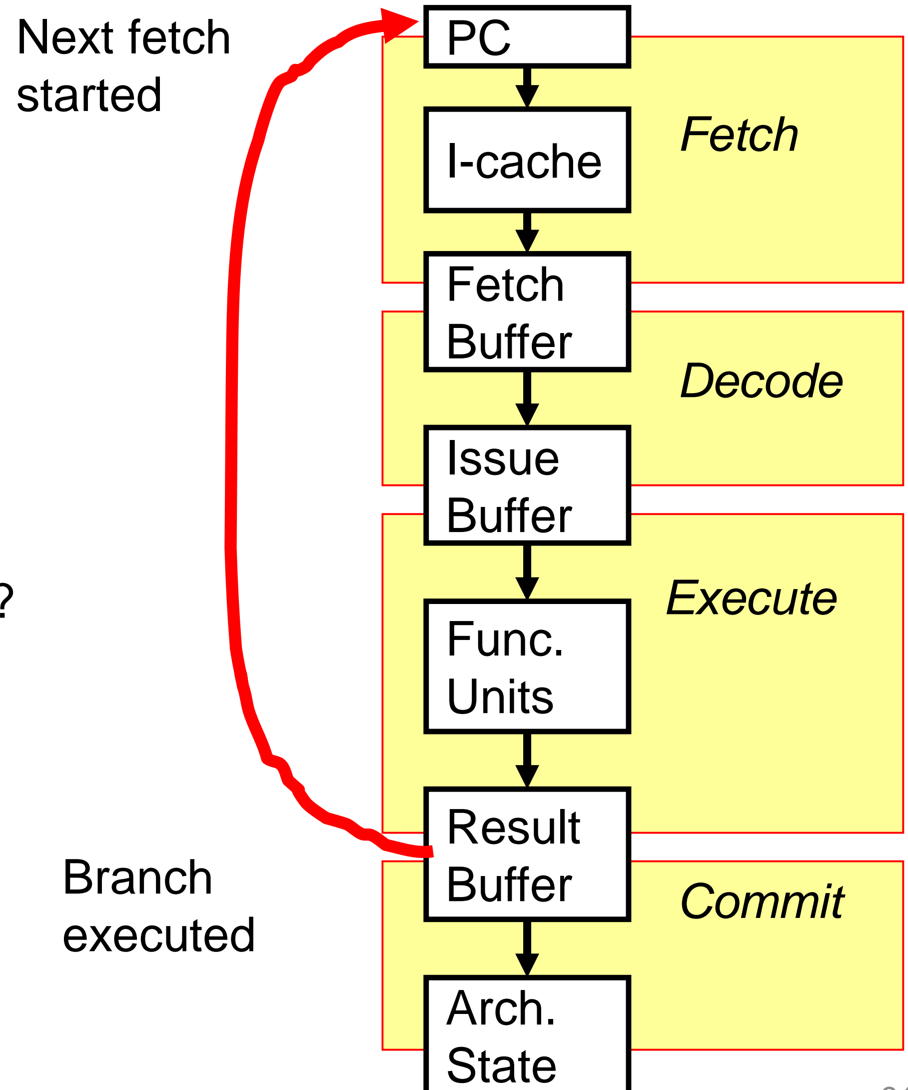
- High performance instruction delivery
  - For a multiple-issue processor, predicting branches well is not enough
    - Predicated execution
    - Branch target buffer (BTB)
  - Deliver a high-bandwidth instruction stream is necessary (e.g. 4~8 instructions/cycle)
    - Increasing instruction fetch bandwidth
      - Branch target buffer (BTB)
    - Predicting procedure returns, indirect jumps, and loop branches
      - Integrated instruction fetch units

Control Flow Penalty

Modern processors may have > 10 pipeline stages between next PC calculation and branch resolution !

How much work is lost if pipeline doesn't follow correct instruction flow?

~ Loop length x pipeline width



Branch and Jump Instruction

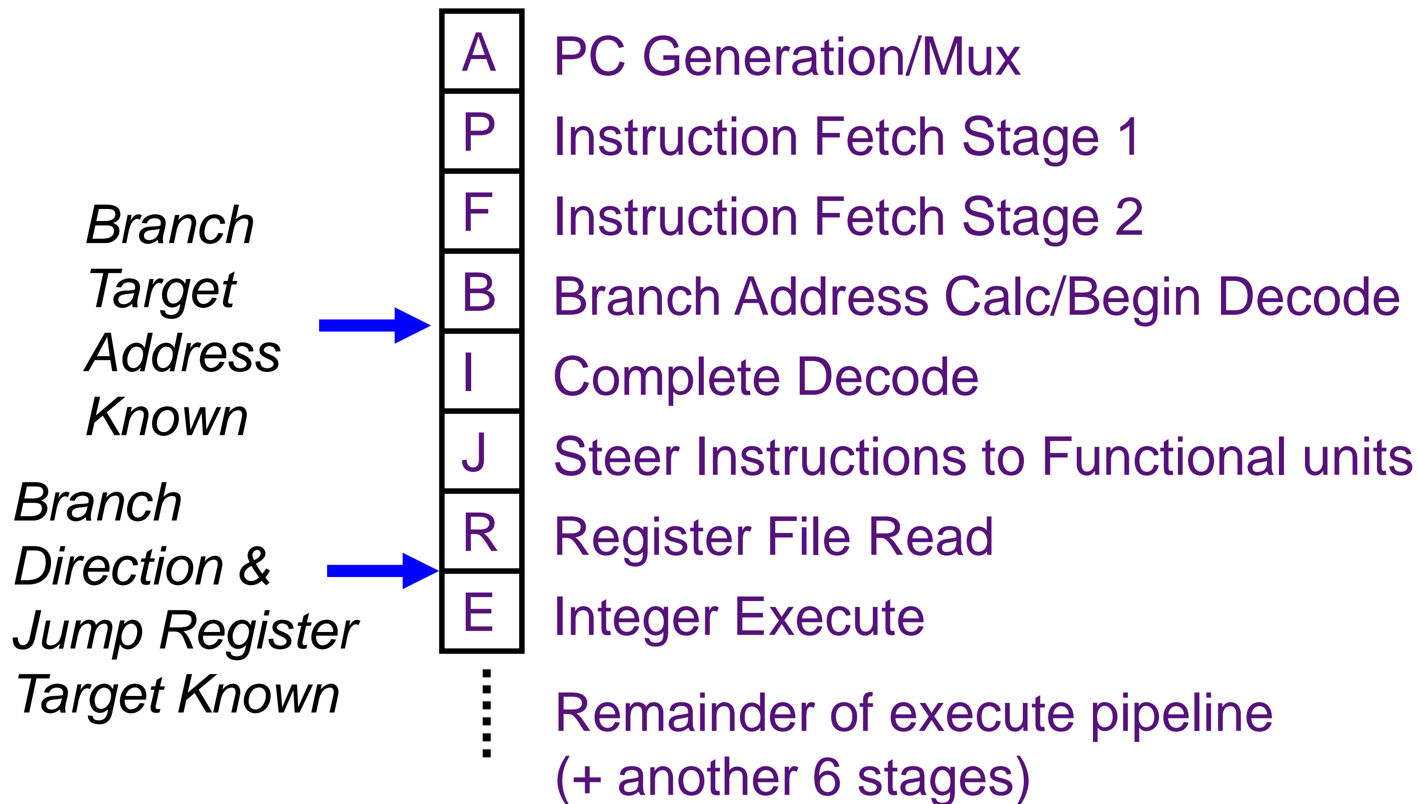
- Each instruction fetch depends on one or two pieces of information from the preceding branch instruction:
  1. Is a taken branch?
  2. If so, what is the target address?
- Example: MIPS branches and jumps

Instruction	*Taken known?*	*Target known?*
J	After Inst. Decode	After Inst. Decode
JR	After Inst. Decode	After Reg. Fetch
BEQZ/BNEZ	After Reg. Fetch*	After Inst. Decode

*Assuming zero detect on register read

Branch Penalties in Modern Pipelines

UltraSPARC-III instruction fetch pipeline stages
(in-order issue, 4-way superscalar, 750MHz, 2000)



Reducing Control Flow Penalty

- Software solutions
  - Loop unrolling: eliminate branches
    - To increase the run length
  - Instruction scheduling: reduce resolution time
    - e.g., delay branch
- Hardware solutions
  - Branch prediction and Speculation
  - Predicated instruction
  - Branch target buffer (BTB)

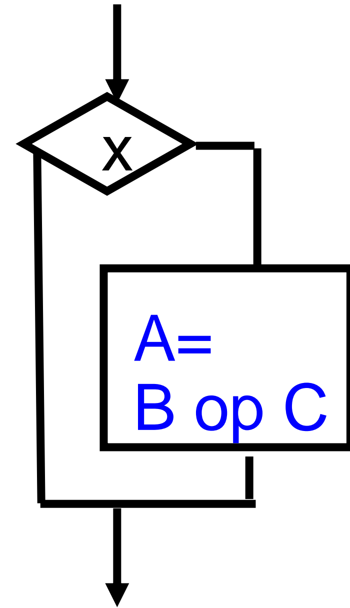
DEPT. OF ELECTRONICS ENGINEERING & INST. OF ELECTRONICS NCTU

Predicated Execution

- Avoid branch prediction by turning branches into conditionally executed instructions:

  if (x) then A = B op C else NOP

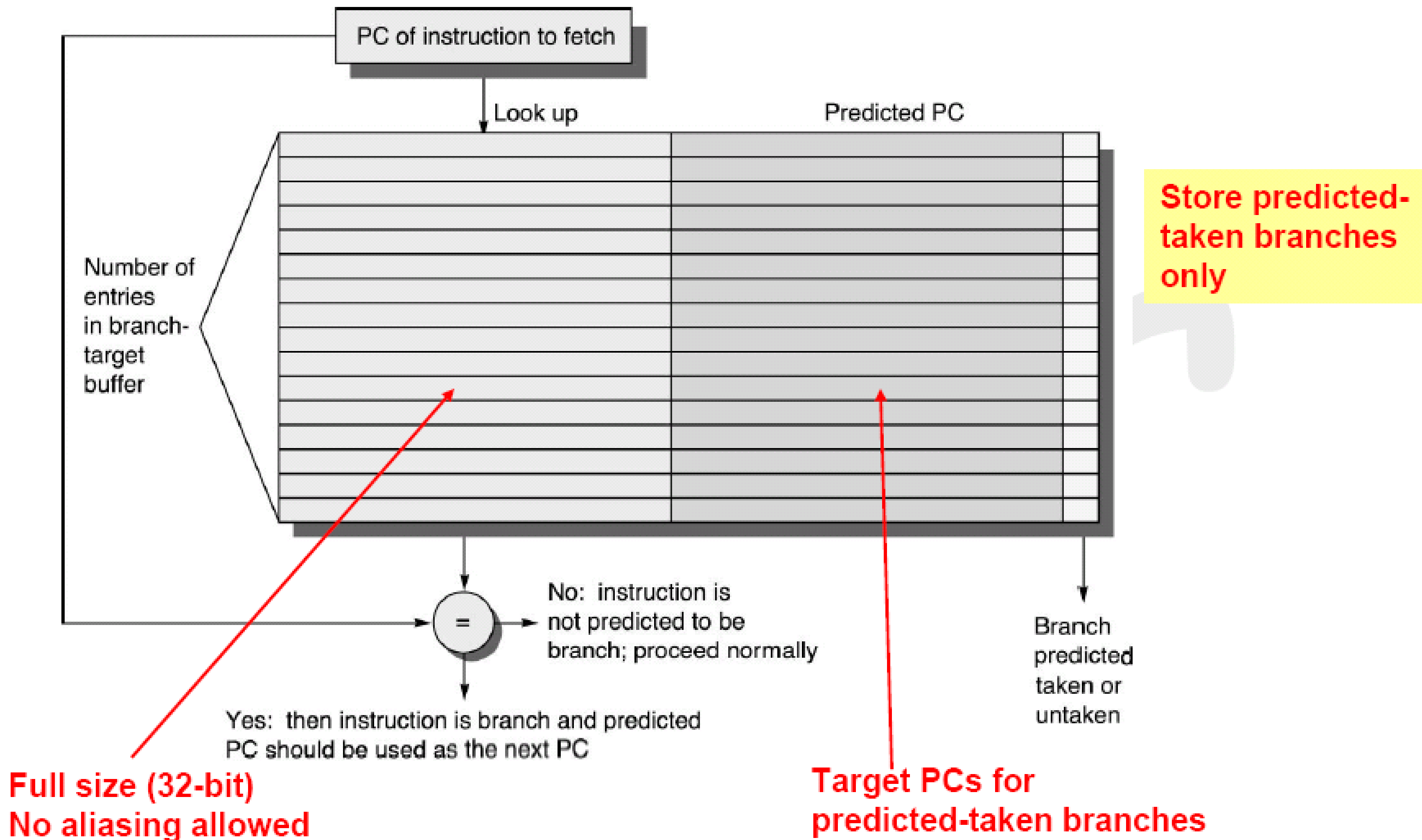
  - If false, then neither store result nor cause exception
  - Expanded ISA with 1-bit condition field
  - This transformation is called "if-conversion"
- Drawbacks to predicated instructions
  - Still takes a clock even if "annulled"
  - Stall if condition evaluated late
  - Complex conditions reduce effectiveness; condition becomes known late in pipeline



Branch Target Buffer/Cache

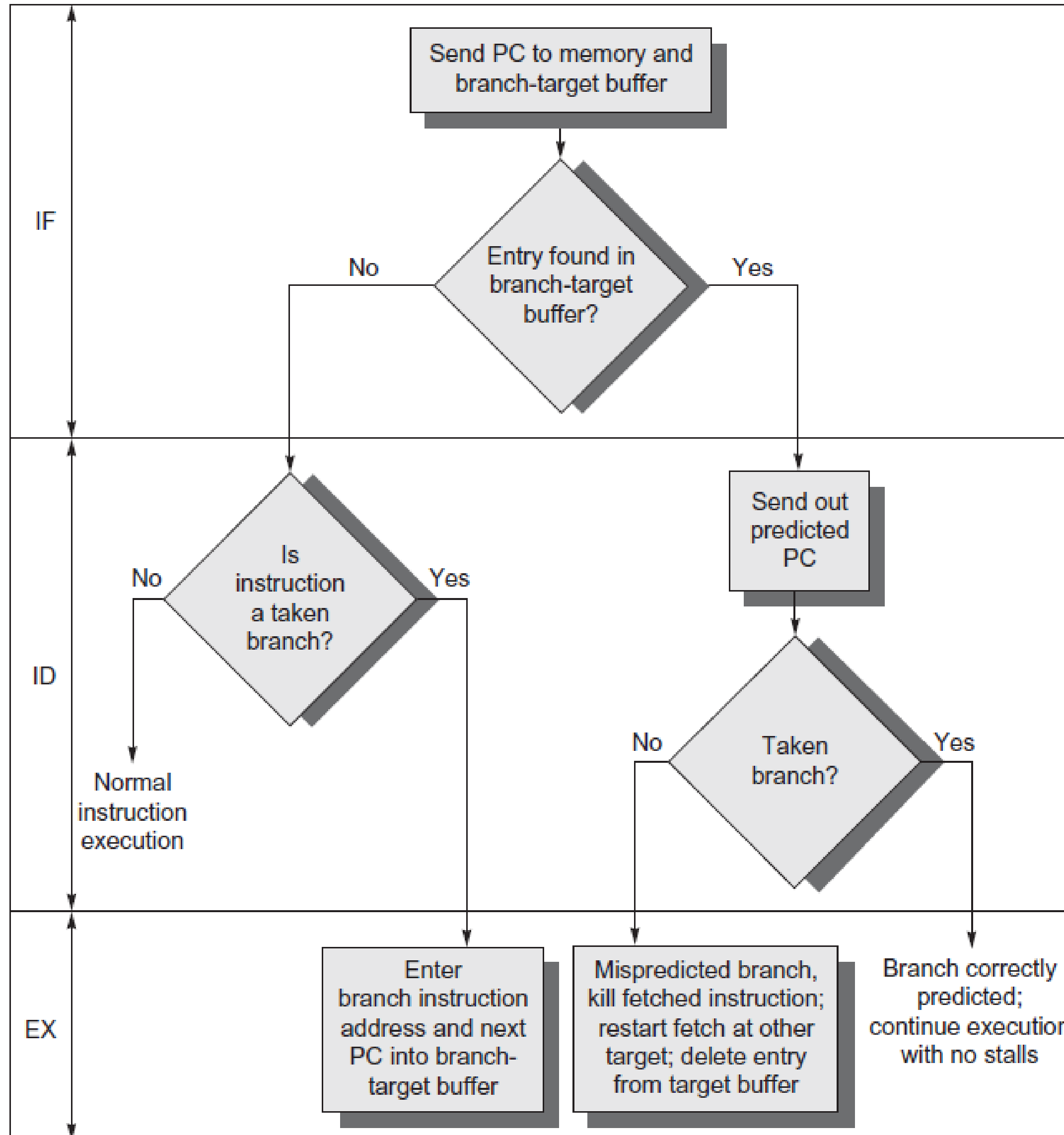
- To reduce the branch penalty from 1 cycle to 0
  - Need to know what the address is at the end of IF
  - But the instruction is not even decoded yet
  - So use the instruction address rather than wait for decode
    - If prediction works then penalty goes to 0!
- BTB Idea -- Cache to store taken branches (no need to store untaken)
  - Access the BTB during IF stage
  - Match tag is instruction address → compare with current PC
  - Data field is the predicted PC
- May want to add predictor field
  - To avoid the mispredict twice on every loop phenomenon
  - Adds complexity since we now have to track untaken branches as well

BTB -- Illustration



Flowchart for BTB

For a simple 5-stage pipeline example



Penalties Using this Approach for 5-Stage Pipeline

Assume we store only taken branches in the buffer

Instruction in buffer	Prediction	Actual Branch	Penalty Cycles
Yes	Taken	Taken	0
Yes	Taken	Not Taken	2
No		Taken	2
No		Not Taken	0

- Predict_wrong = 1 CC to update BTB + 1 CC to restart fetching
- Not found and taken = 2CC to update BTB

- For complex pipeline design, the penalties may be higher

Example

- Given prediction accuracy (for inst. in buffer): 90%
- Given hit rate in buffer (for branches predicted token): 90%
- Determine the total branch penalty=?

Solution

- Probability (branch in buffer, but actually not taken) = percent buffer hit rate × percent incorrect prediction = 90% × 10% = 0.09
- Probability (branch not in buffer, but actually taken) = 10%
- Hence, we have 2 cycles × (0.09+0.1) = 0.38 cycles

Comparing the delay branch with the penalty = 0.5 cycles/branch

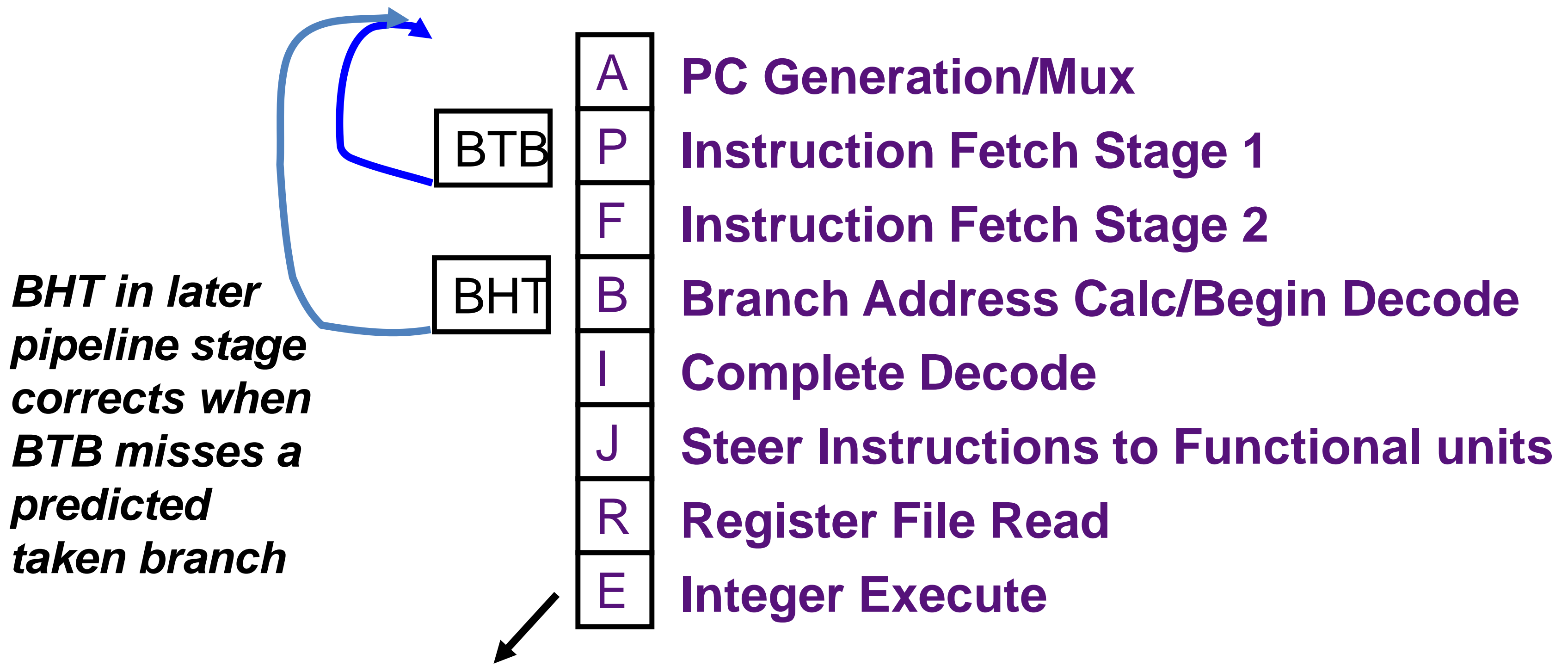
Branch Folding

- Optimization of BTB
  - To store target instructions instead of, or in addition to, the predicted target address
  - Called *Branch folding*
    - It allows the branch-target buffer access to take longer than the time between successive instruction fetches, possibly allowing a larger branch-target buffer.
    - It can be used to obtain 0-cycle unconditional branches and sometimes 0-cycle conditional branches.
    - E.g., Cortex A-53.

DEPT. OF ELECTRONICS ENGINEERING & INST. OF ELECTRONICS NCTU

Combining BTB and BHT

- BTB entries are considerably more expensive than BHT, but can redirect fetches at earlier stage in pipeline and can accelerate indirect branches (JR)
- BHT can hold many more entries and is more accurate



BTB/BHT only updated after branch resolves in E stage

BTB Remarks

- BTB contains useful information for branch and jump instructions only
  - Do not update BTB for other instructions
  - For all other instructions, the next PC is PC+4
- Keep both the branch PC and target PC in the BTB
  - "Branch folding"
  - 0-cycle unconditional branches
  - Sometimes 0-cycle conditional branches
- Only predicted taken branches and jumps held in BTB
  - More room to store
- Subroutine returns? (jump to return address)
  - BTB can work well if usually return to the same place
  - Return address predictors

Return Address Predictor ?

- Indirect jump – jumps whose destination address varies at run time
  - indirect procedure call, select or case, procedure return
  - SPEC89 benchmarks: 85% of indirect jumps are procedure returns
- Accuracy of BTB for procedure returns are low
  - if procedure is called from many places, and the calls from one place are not clustered in time
- Use a small buffer of return addresses operating as a stack
  - Cache the most recent return addresses
  - Push a return address at a call, and pop one off at a return
  - If the cache is sufficient large (max call depth) → prefect

Subroutine Return Stack

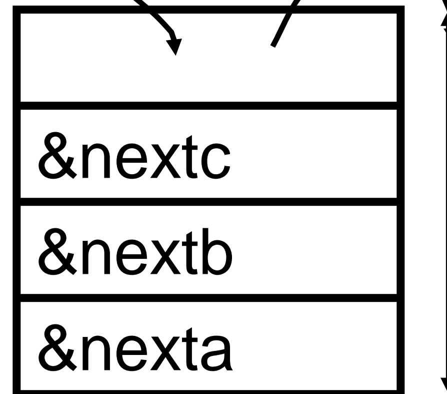
- Small structure to accelerate JR for subroutine returns, typically much more accurate than BTBs.

fa() { fb(); nexta: }

fb() { fc(); nextb: }

fc() { fd(); nextc: }

Push return address when function call executed

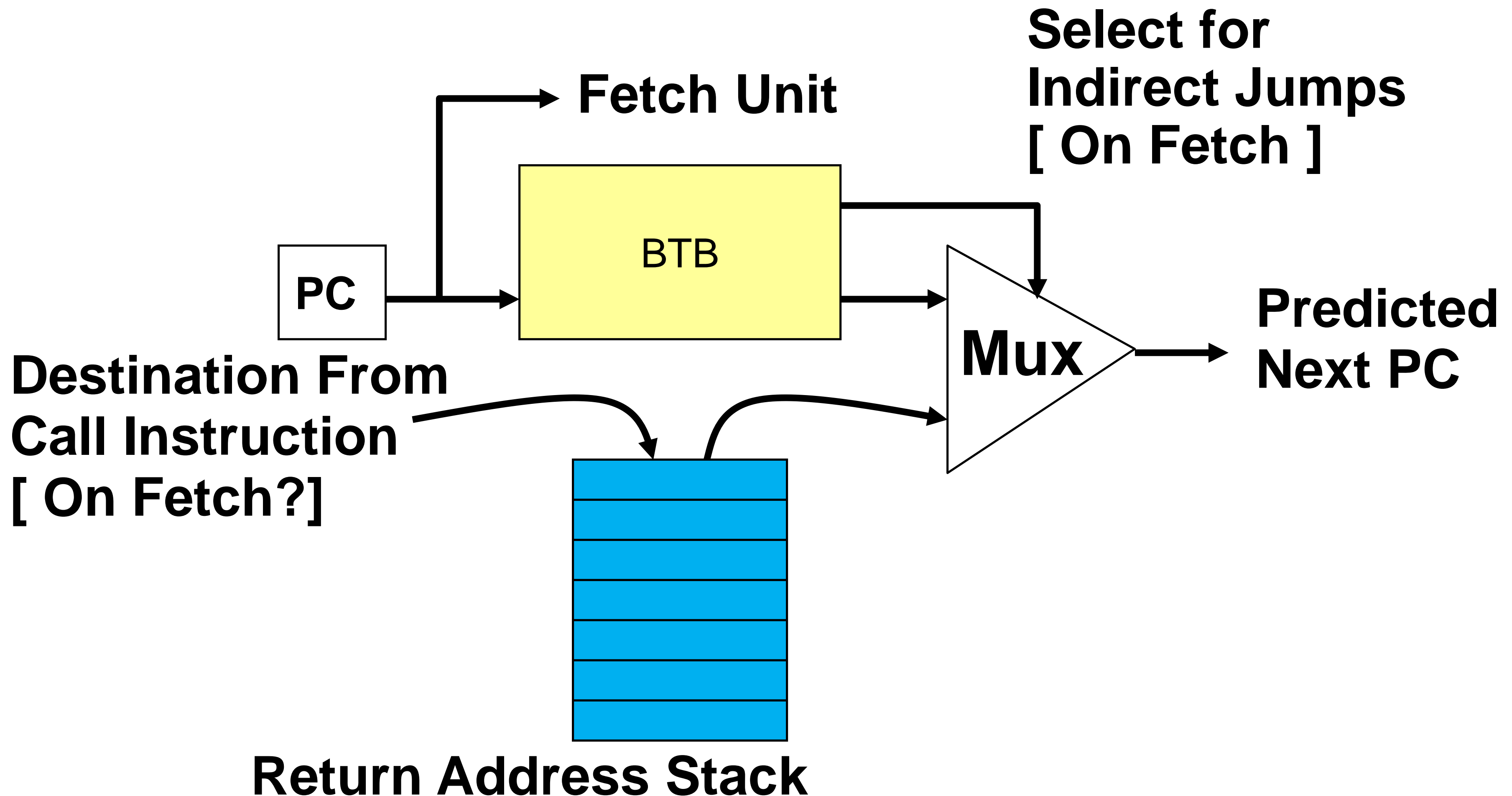


Pop return address when subroutine return decoded

k entries (typically k=8-16)

BTW with Return Addresses Stack

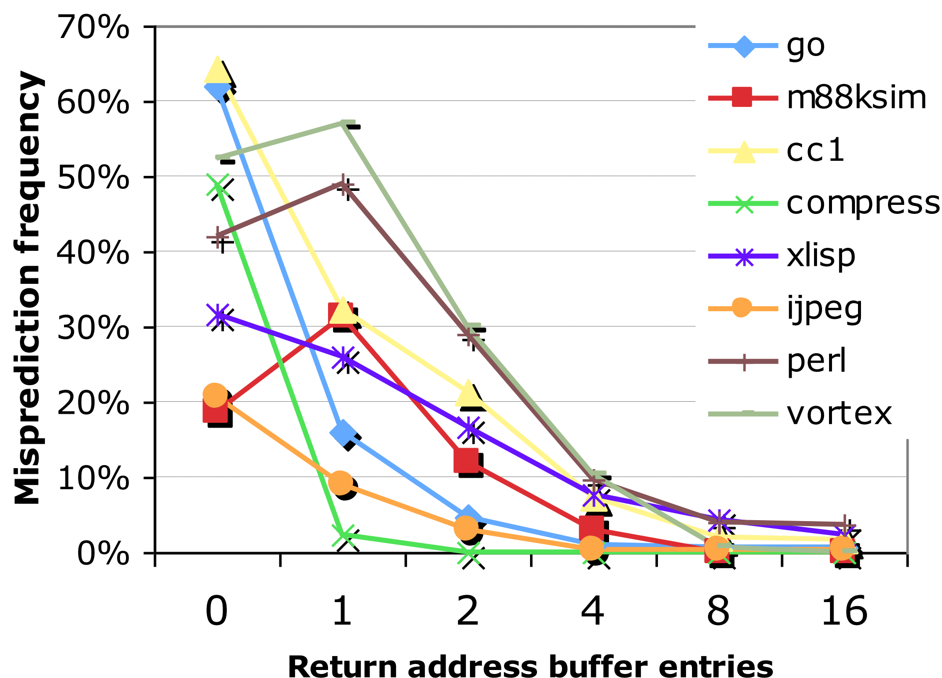
- Register Indirect branch hard to predict address



DEPT. OF ELECTRONICS ENGINEERING & INST. OF ELECTRONICS NCTU

Performance: Return Address Predictor

- Cache most recent return addresses:
  - Call ⇒ Push a return address on stack
  - Return ⇒ Pop an address off stack & predict as new PC
- SPEC95 Benchmarks



Dynamic Branch Prediction Summary

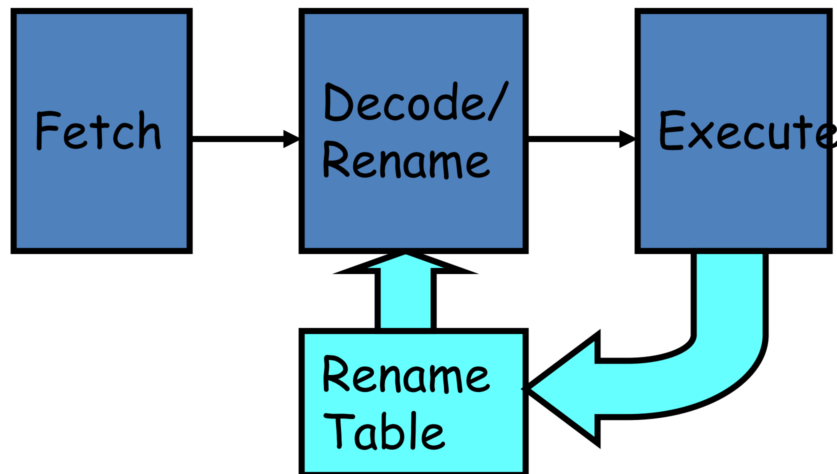
- Branch prediction scheme are limited by
  - Prediction accuracy
  - Mis-prediction penalty
- Branch History Table: 2 bits for loop accuracy
- Correlation: Recently executed branches correlated with next branch
- Tournament predictors take insight to next level, by using multiple predictors
  - usually one based on global information and one based on local information, and combining them with a selector
  - In 2006, tournament predictors using ≈ 30K bits are in processors like the Power5 and Pentium 4
- Branch Target Buffer: include branch address & prediction
- Reduce penalty further by fetching instructions from both the predicted and unpredicted direction
  - Require dual-ported memory, interleaved cache → HW cost
  - Caching addresses or instructions from multiple path in BTB

More Instruction Fetch Bandwidth

- Consider the fetch unit as a separate autonomous unit, not a pipeline stage
- Functions for the integrated instruction fetch unit
  - Branch prediction
    - The branch predictor becomes part of the instruction fetch unit
  - Prefetch
    - To deliver multiple instructions per cycle
  - Instruction memory access and buffering
    - Fetching multiple instructions may require accessing multiple cache lines
      - Using prefetch may hide the latency for memory access
    - Buffering can provide instructions to the issue stage as needed

Explicit Register Renaming

- Instead of virtual registers from reservation stations and reorder buffer, create a single (physical) register pool
  - Contains visible registers and virtual registers
- Use hardware-based map to rename registers during issue
- Still need a ROB-like queue to update table in order
- Physical register becomes free when not being used



Speculation: Register Renaming vs. ROB

- Alternative to ROB is a larger physical set of registers combined with register renaming
  - An extended set of registers replace function of both ROB and reservation stations
- Instruction issue maps names of architectural registers to physical register numbers in extended register set
  - On issue, allocates a new unused register for the destination (which avoids WAW and WAR hazards)
  - Speculation recovery easy because a physical register holding an instruction destination does not become the architectural register until the instruction commits
- Most Out-of-Order processors today use extended registers with renaming

Speculation Performance

- Speculation will raise the power consumption
- How much to speculate
  - Mis-speculation degrades performance and power relative to no speculation
    - May cause additional misses (cache, TLB)
  - Prevent speculative code from causing higher costing misses (e.g. L2)
- Speculating through multiple branches
  - Complicates speculation recovery
  - No processor can resolve multiple branches per cycle
- Speculation and energy efficiency
  - Note: speculation is only energy efficient when it significantly improves performance

Miss-speculation Performance

SPEC2000

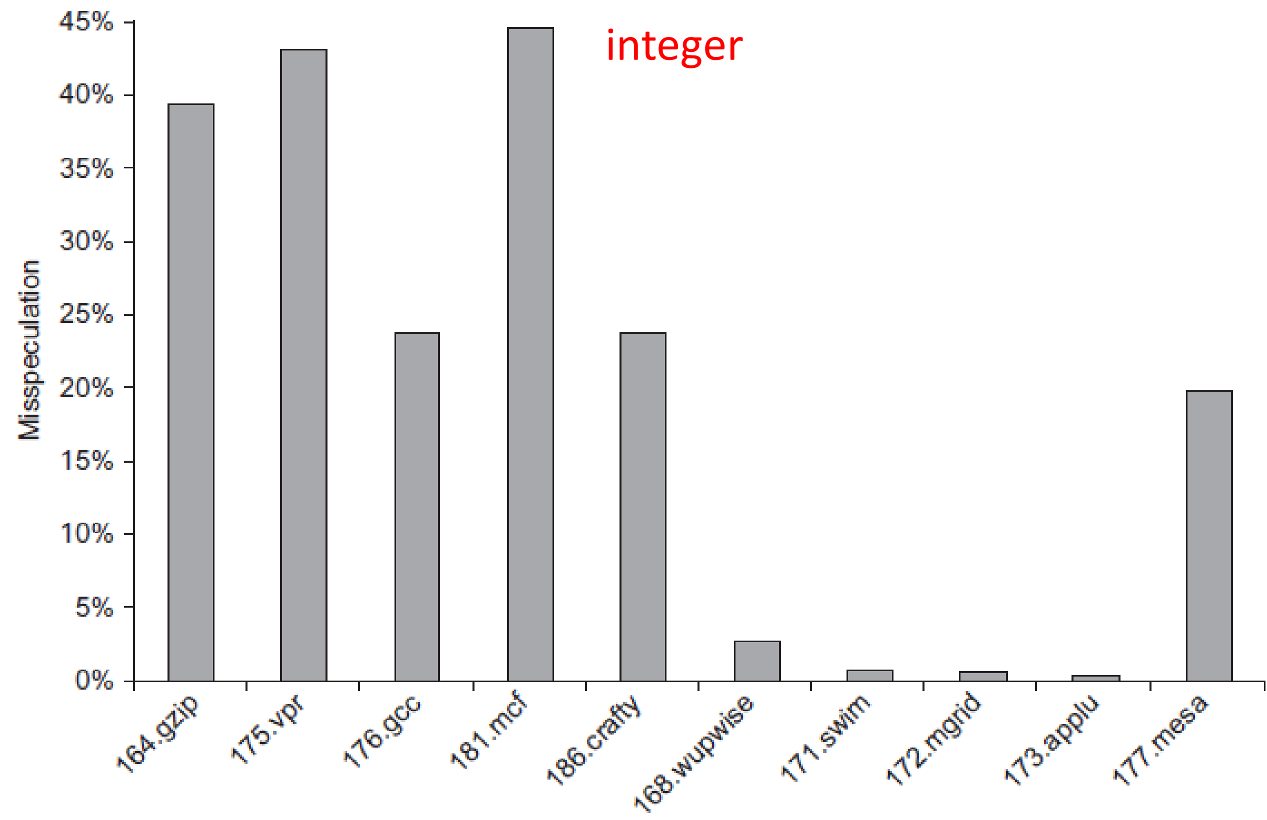


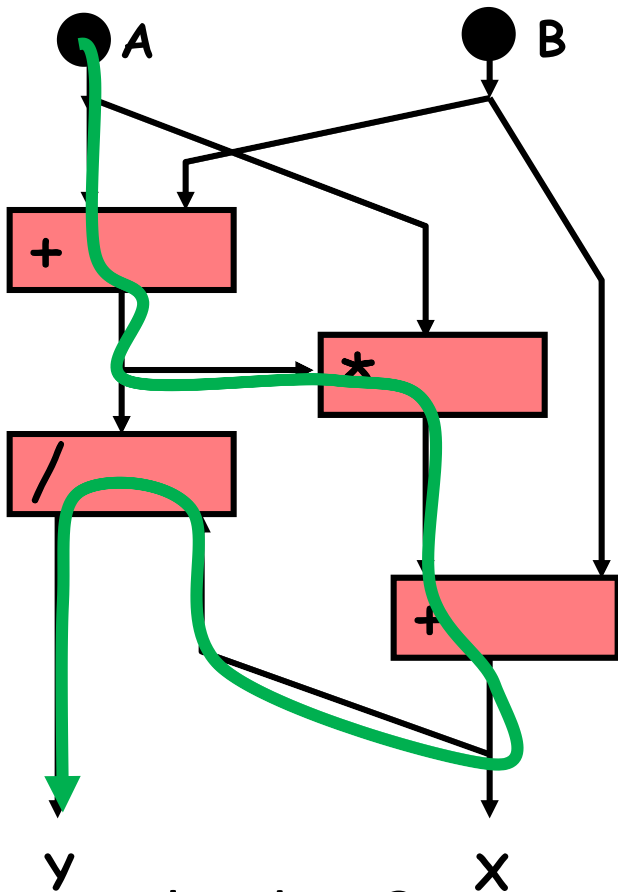
Figure 3.30 The fraction of instructions that are executed as a result of misspeculation is typically much higher for integer programs (the first five) versus FP programs (the last five).

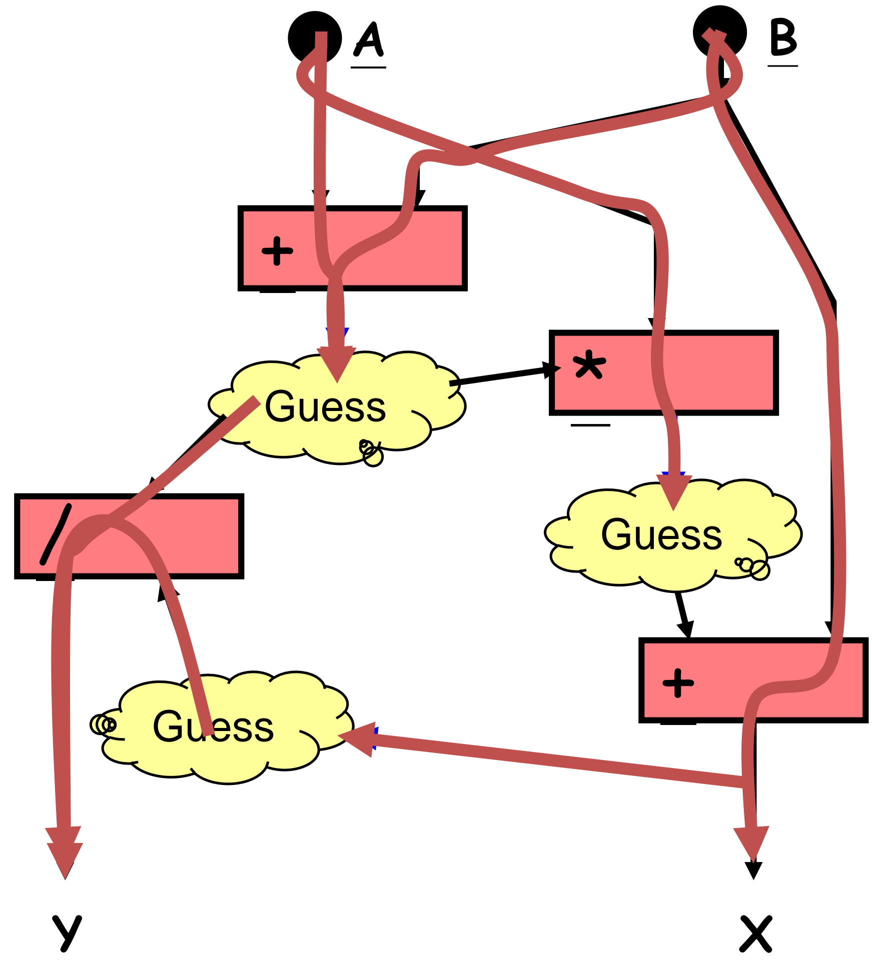
Value Prediction

- Attempts to predict value produced by instruction
  - E.g., Loads a value that changes infrequently
- Value prediction is useful only if it significantly increases ILP
  - Focus of research has been on loads; so-so results, no processor uses value prediction
- Related topic is *address aliasing prediction*
  - RAW for load and store or WAW for 2 stores
- Address alias prediction is both more stable and simpler since need not actually predict the address values, only whether such values conflict
  - Has been used by a few processors

DEPT. OF ELECTRONICS ENGINEERING & INST. OF ELECTRONICS NCTU

Data Value Prediction Example





- Why do it?
  - Can "Break the DataFlow Boundary"
  - Before: Critical path = 4 operations (probably worse)
  - After: Critical path = 1 operation (plus verification)

DEPT. OF ELECTRONICS ENGINEERING & INST. OF ELECTRONICS NCTU

In Conclusion…

- Interest in multiple-issue because wanted to improve performance without affecting uniprocessor programming model
- Taking advantage of ILP is conceptually simple, but design problems are amazingly complex in practice
- Conservative in ideas, just faster clock and bigger
- Processors of Pentium 4, IBM Power 5, and AMD Opteron have the same basic structure and similar sustained issue rates (3 to 4 instructions per clock) as the 1st dynamically scheduled, multiple-issue processors announced in 1995
  - Clocks 10 to 20X faster, caches 4 to 8X bigger, 2 to 4X as many renaming registers, and 2X as many load-store units $\Rightarrow$ performance 8 to 16X
- Peak vs. delivered performance gap increasing