Instruction-level Parallelism

(Computer Architecture: Chapter 3 & Appendix C)

Yanyan Shen

Department of Computer Science and Engineering

Shanghai Jiao Tong University

Outline

- 3.1 Concepts and Challenges
- 3.2 Basic Compiler Techniques
- 3.3 Reducing Branch Costs with Advanced Branch Prediction
- 3.4 Overcoming Data Hazards with Dynamic Scheduling
- 3.5 Hardware-based Speculation

Background

- "*Instruction Level Parallelism (ILP)*" refers to overlap execution of instructions
  - **Pipelining** become universal technique in 1985
- There are two main approaches to exploit ILP
  - Hardware-based dynamic approaches
    - Used in server and desktop processors
  - Compiler-based static approaches
    - Not as successful outside of scientific applications

The goal of this chapter:

1) to look at the limitation imposed by data and control hazards;

2) Investigate the topic of increasing the ability of the compiler (software) and the processor (hardware) to exploit ILP

Pipeline CPI

When exploiting instruction-level parallelism, the goal is to minimize CPI

Pipeline CPI = Ideal pipeline CPI + Structural Stalls + Data hazard stalls + Control Stalls

Ideal pipeline CPI is a measure of the maximum performance attainable by the implementation

By reducing each of the terms in the right-hand side, we decrease the overall pipeline CPI or improve IPC (instructions per clock)

Major Techniques for Improving Pipeline CPI

Technique	Reduces	Section
Forwarding and bypassing	Potential data hazard stalls	C.2
Delayed branches and simple branch scheduling	Control hazard stalls	C.2
Basic compiler pipeline scheduling	Data hazard stalls	C.2, 3.2
Basic dynamic scheduling (scoreboarding)	Data hazard stalls from true dependences	C.7
Loop unrolling	Control hazard stalls	3.2
Branch prediction	Control stalls	3.3
Dynamic scheduling with renaming	Stalls from data hazards, output dependences, and antidependences	3.4
Hardware speculation	Data hazard and control hazard stalls	3.6
Dynamic memory disambiguation	Data hazard stalls with memory	3.6
Issuing multiple instructions per cycle	Ideal CPI	3.7, 3.8
Compiler dependence analysis, software pipelining, trace scheduling	Ideal CPI, data hazard stalls	H.2, H.3
Hardware support for compiler speculation	Ideal CPI, data hazard stalls, branch hazard stalls	H.4, H.5

What is Instruction-Level Parallelism

- **Basic block:** a straight-line code sequence with no branches in except to the entry and no branches out except at the exit

- **Parallelism with a basic block is limited**
  - The average dynamic branch frequency is often between 15% and 25%
  - Thus, typical size of basic block = 3-6 instructions

- ***Conclusion: Must optimize across branches***

Dependences and Hazards

Dependences:

Determining how one instruction depends on another is critical to determining

1) how much parallelism exists in a program and

2) how that parallelism can be exploited!

Hazards:

When there are hazards, the pipeline may be stopped in order to resolve the hazards

Three Types of Dependences

Data dependences (also called true data dependences) Data flow between instructions

Name dependences (anti-dependence and output dependence)

Control dependence

Data Dependences

- **Dependencies are a property of *programs***
- Pipeline organization determines if a dependence results in an actual hazard being detected and if the hazard causes a stall
- Data dependence conveys:
  - **Possibility** of a hazard
  - Order in which results must be calculated
  - Upper bound on exploitable instruction level parallelism

Name Dependences

- **Two instructions use the same register or memory location but no flow of information**
  - Not a true data dependence, ***but is a problem when reordering instructions***
  - ***Antidependence*:** instruction *j* writes a register or memory location that instruction *i* reads
    - Initial ordering (*i* before *j*) must be preserved
  - ***Output dependence***: instruction *i* and instruction *j* write the same register or memory location
    - Ordering must be preserved
- To resolve, use ***renaming techniques***

Control Dependence

- Ordering of instruction *i* with respect to a branch *instruction*
  - Instruction control *dependent on* a branch cannot be moved before the branch so that its execution is no longer controller by the branch
  - An instruction not *control dependent* on a branch cannot be moved after the branch so that its execution is controlled by the branch

Data Hazards

Read after write (RAW)

- *j* tries to read a source before *i* writes it
- Program order must be preserved!

Write after write (WAW)

- *j* tries to write an operand before it is written by *i*

Write after read (WAR)

- *j* tries to write a destination before it is read by *i*, so *i* incorrectly gets the new value

Outline

- 3.1 Concepts and Challenges
- 3.2 Basic Compiler Techniques
- 3.3 Reducing Branch Costs with Advanced Branch Prediction
- 3.4 Overcoming Data Hazards with Dynamic Scheduling
- 3.5 Hardware-based Speculation

Latencies of FP operations

To avoid a pipeline stall, the execution of a dependent instruction must be separated from the source instruction by *a distance in clock cycles equal to the pipeline latency* of that source instruction

A compiler's ability to perform pipeline scheduling is dependent both on the amount of ILP available in the program and on the latencies of the functional units in the pipeline

Instruction producing result	Instruction using result	Latency in clock cycles
FP ALU op	Another FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1
Load double	Store double	0

Pipeline Stalls

Example:

```
for (i=999; i>=0; i=i-1)
  x[i] = x[i] + s;
```

```
Loop:   L.D      F0,0(R1)
        stall
        ADD.D F4,F0,F2
        stall
        stall
        S.D F4,0(R1)
        DADDUI R1,R1,#-8
        stall (assume integer load latency is 1)
        BNE R1,R2,Loop
```

Pipeline Scheduling

Scheduled code:

```
Loop:L.D    F0,0(R1)
     DADDUI R1,R1,#-8
     ADD.D F4,F0,F2
     stall
     stall
     S.D F4,8(R1)
     BNE R1,R2,Loop
```

Loop Unrolling

Unroll by a factor of 4 (assume # elements is divisible by 4)
Eliminate unnecessary instructions

```
Loop:   L.D F0,0(R1)
        ADD.D F4,F0,F2
        S.D F4,0(R1) ;drop DADDUI & BNE
        L.D F6,-8(R1)
        ADD.D F8,F6,F2
        S.D F8,-8(R1) ;drop DADDUI & BNE
        L.D F10,-16(R1)
        ADD.D F12,F10,F2
        S.D F12,-16(R1) ;drop DADDUI & BNE
        L.D F14,-24(R1)
        ADD.D F16,F14,F2
        S.D F16,-24(R1)
        DADDUI R1,R1,#-32
        BNE R1,R2,Loop
```

Loop Unrolling/Pipeline Scheduling

```
Loop: L.D F0,0(R1)
      L.D F6,-8(R1)
      L.D F10,-16(R1)
      L.D F14,-24(R1)
      ADD.D F4,F0,F2
      ADD.D F8,F6,F2
      ADD.D F12,F10,F2
      ADD.D F16,F14,F2
      S.D F4,0(R1)
      S.D F8,-8(R1)
      DADDUI R1,R1,#-32
      S.D F12,16(R1)
      S.D F16,8(R1)
      BNE R1,R2,Loop
```

Outline

- 3.1 Concepts and Challenges
- 3.2 Basic Compiler Techniques
- 3.3 Reducing Branch Costs with Advanced Branch Prediction
- 3.4 Overcoming Data Hazards with Dynamic Scheduling
- 3.5 Hardware-based Speculation

Control Speculation

- Execute instruction beyond a branch before the branch is resolved → Performance
- Speculative execution

- What if mis-speculated?
  - Recovery mechanism
  - Squash instructions on the incorrect path
- Branch prediction: **Dynamic** vs. **Static**

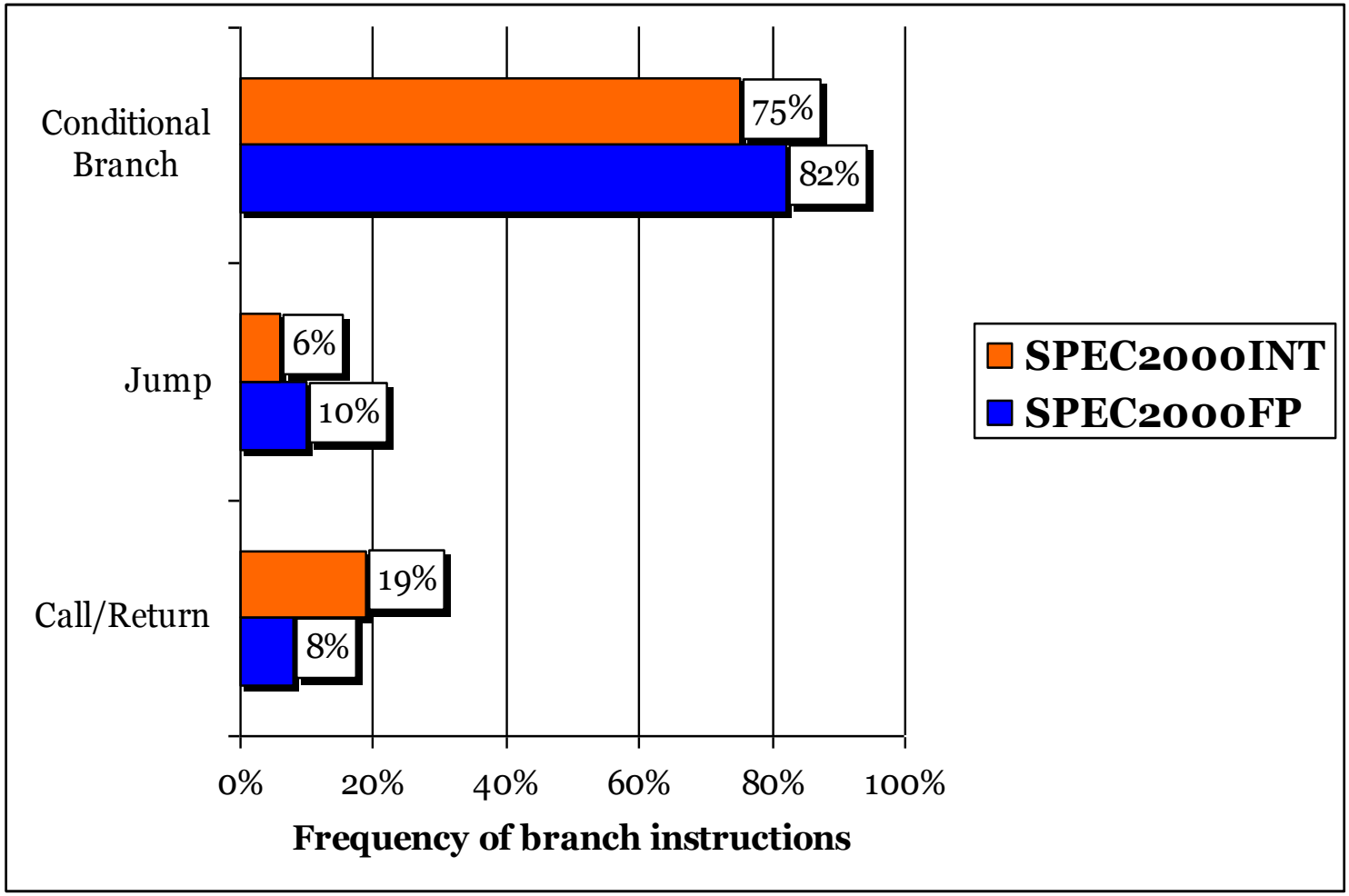
Predict What?

- **Direction (1-bit)**
  - *Single direction* for unconditional jumps and calls/returns. **No need to predict!**
  - *Binary directions* for conditional branches

- **Target (32-bit or 64-bit addresses)**
  - Some are easy
    - One: Uni-directional jumps
    - Two: Fall through (Not Taken) vs. Taken
  - Many: Function Pointer or Indirect Jump (e.g. jr r31)

Prof. Sean Lee's Slide

Categorizing Branches



Source: H&P using Alpha

Prof. Sean Lee's Slide

Why Branch is Predictable?

```
for (i=0; i<100; i++) {

        ....
}
```

```
    addi   r10, r0, 100
    addi   r1,   r0, r0

L1: … …
    … …
    addi   r1, r1,   1
    bne    r1, r10, L1
    … …
```

Prof. Sean Lee's Slide

Static Branch Prediction

- Uni-directional, always predict taken

- **Static prediction** is used as a fall-back technique in some processors with dynamic branch when there is not any information for dynamic predictors to use

Modified from Prof. Sean Lee's Slide

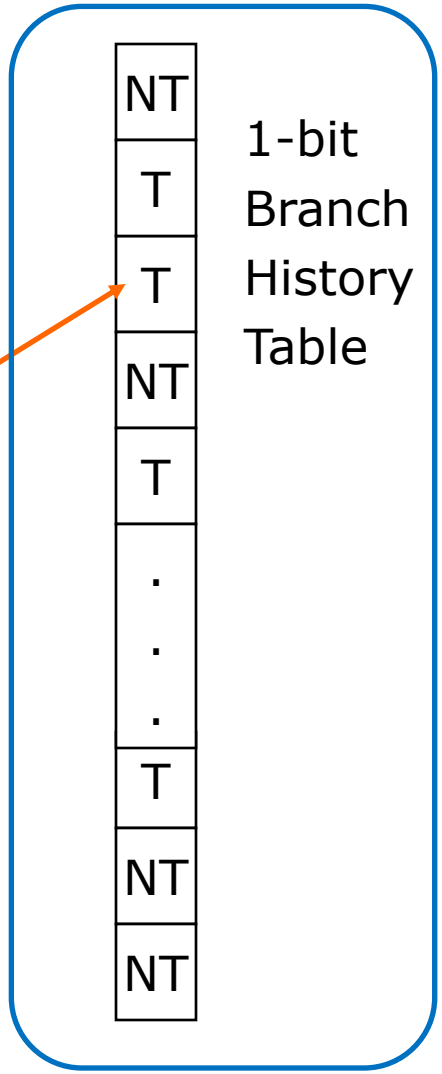
Simplest Dynamic Branch Predictor

- Prediction based on latest outcome
- Index by some bits in the branch PC
  - Aliasing

```
for (i=0; i<100; i++) {
        ....
}
```

```
0x40010100  addi  r10, r0, 100
0x40010104  addi  r1,  r1, r0

0x40010108  L1:
            ... ...
...         ... ...
0x40010A04  addi  r1, r1,  1
0x40010A08  bne   r1, r10, L1
            ... ...
```



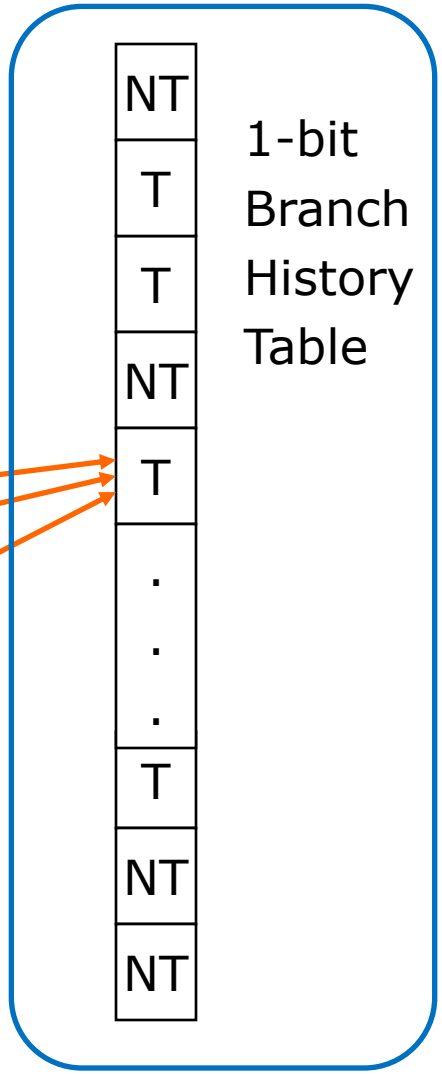
How accurate?

Prof. Sean Lee's Slide

Simplest Dynamic Branch Predictor

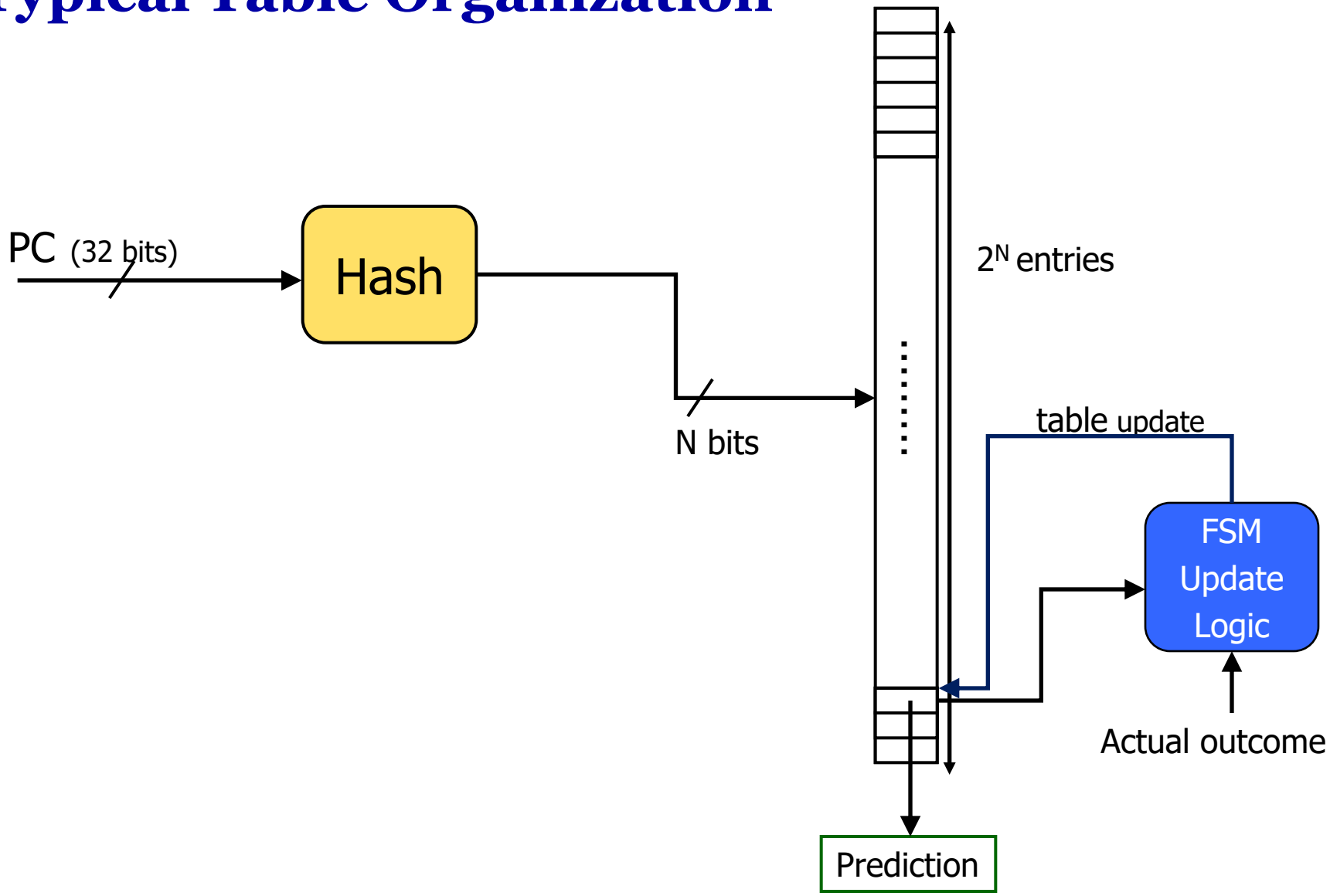
```
for (i=0; i<100; i++) {
  if (a[i] == 0) {
    …
  }
  …
}
```

```
0x40010100   addi   r10, r0, 100
0x40010104   addi   r1,   r1, r0
             L1:
0x40010108   add    r21, r20, r1
0x4001010c   lw     r2, (r21)
0x40010110   beq    r2, r0, L2
             … …
0x40010210   j      L3
             L2:
             … … …
             L3:
0x40010B0c   addi   r1, r1,   1
0x40010B10   bne    r1, r10, L1
```



Prof. Sean Lee's Slide

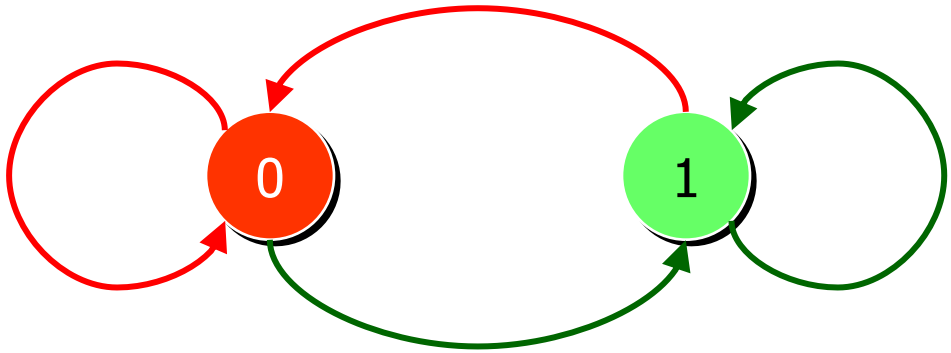
Typical Table Organization



Prof. Sean Lee's Slide

FSM of the Simplest Predictor

- A 2-state machine
- Change mind fast



Actual outcome

If branch taken

If branch not taken

0 Predict **not taken**

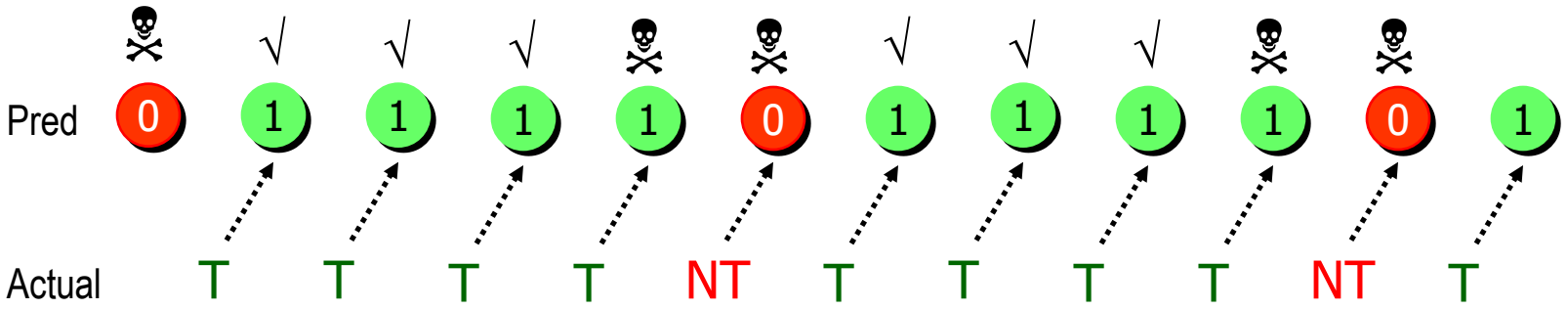
1 Predict **taken**

Prof. Sean Lee's Slide

Example using 1-bit branch history table

```
for (i=0; i<5; i++) {
        ....
}
```

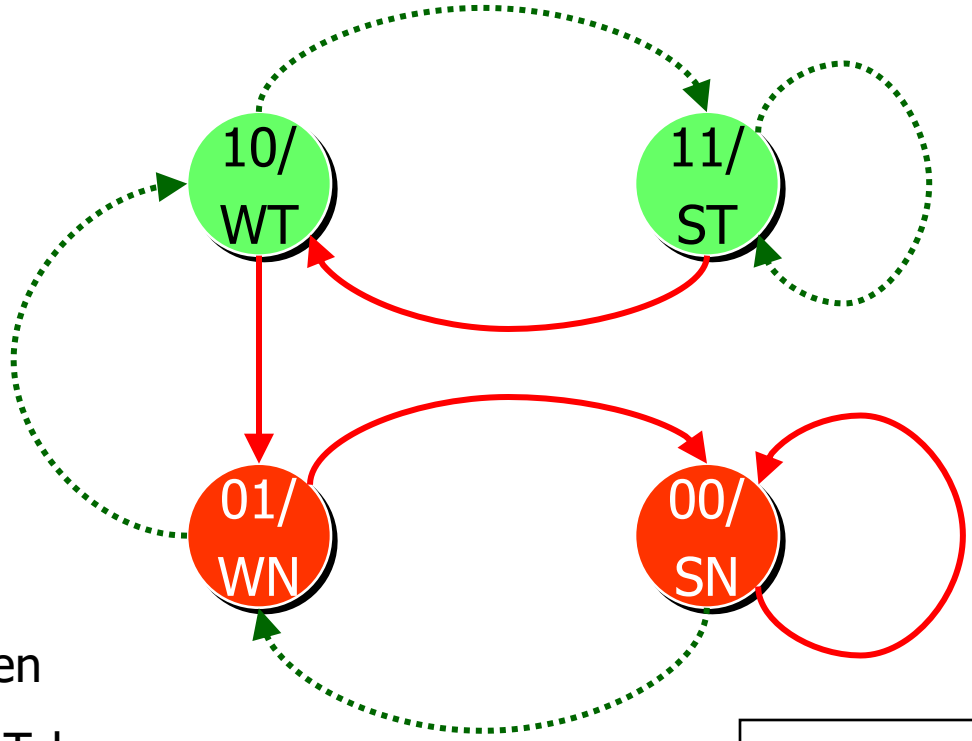
```
 addi   r10, r0, 5
 addi   r1,   r0, 0
L1:
 … …
 addi   r1, r1,   1
 bne    r1, r10, L1
```



Prof. Sean Lee's Slide

2-bit Saturating Up/Down Counter Predictor

MSB: Direction bit
LSB: Hysteresis bit



Taken

Not Taken

Predict Not taken

Predict taken

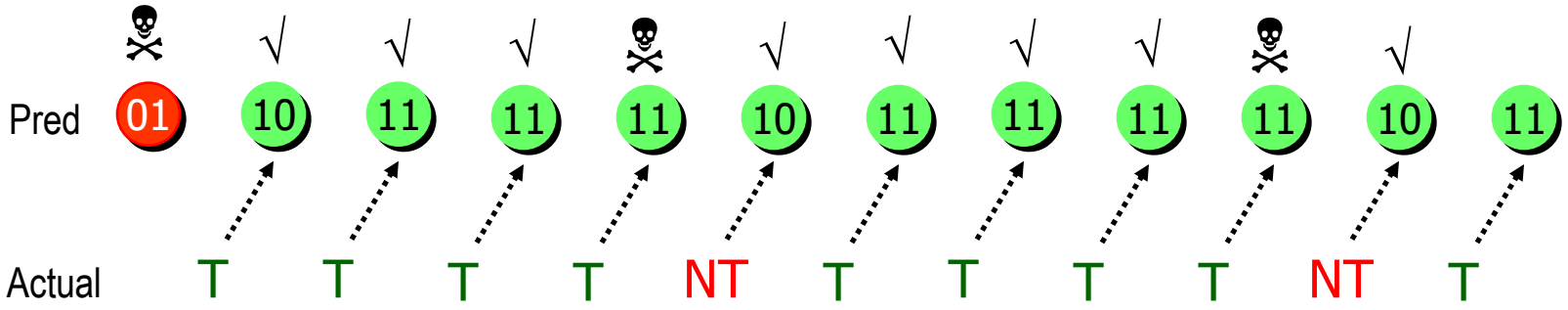
ST: Strongly Taken
WT: Weakly Taken
WN: Weakly Not Taken
SN: Strongly Not Taken

Prof. Sean Lee's Slide

Example using 2-bit up/down counter

```
for (i=0; i<5; i++) {
        ....
}
```

```
 addi   r10, r0, 5
 addi   r1,   r0, 0
L1:
 … …
 addi   r1, r1,   1
 bne    r1, r10, L1
```



Outline

- 3.1 Concepts and Challenges
- 3.2 Basic Compiler Techniques
- 3.3 Reducing Branch Costs with Advanced Branch Prediction
- 3.4 Overcoming Data Hazards with Dynamic Scheduling
- 3.5 Hardware-based Speculation

Motivation

DIV.D ***F4***, F0, F2
ADD.D F10, ***F4***, F8
SUB.D F12, F6, F14

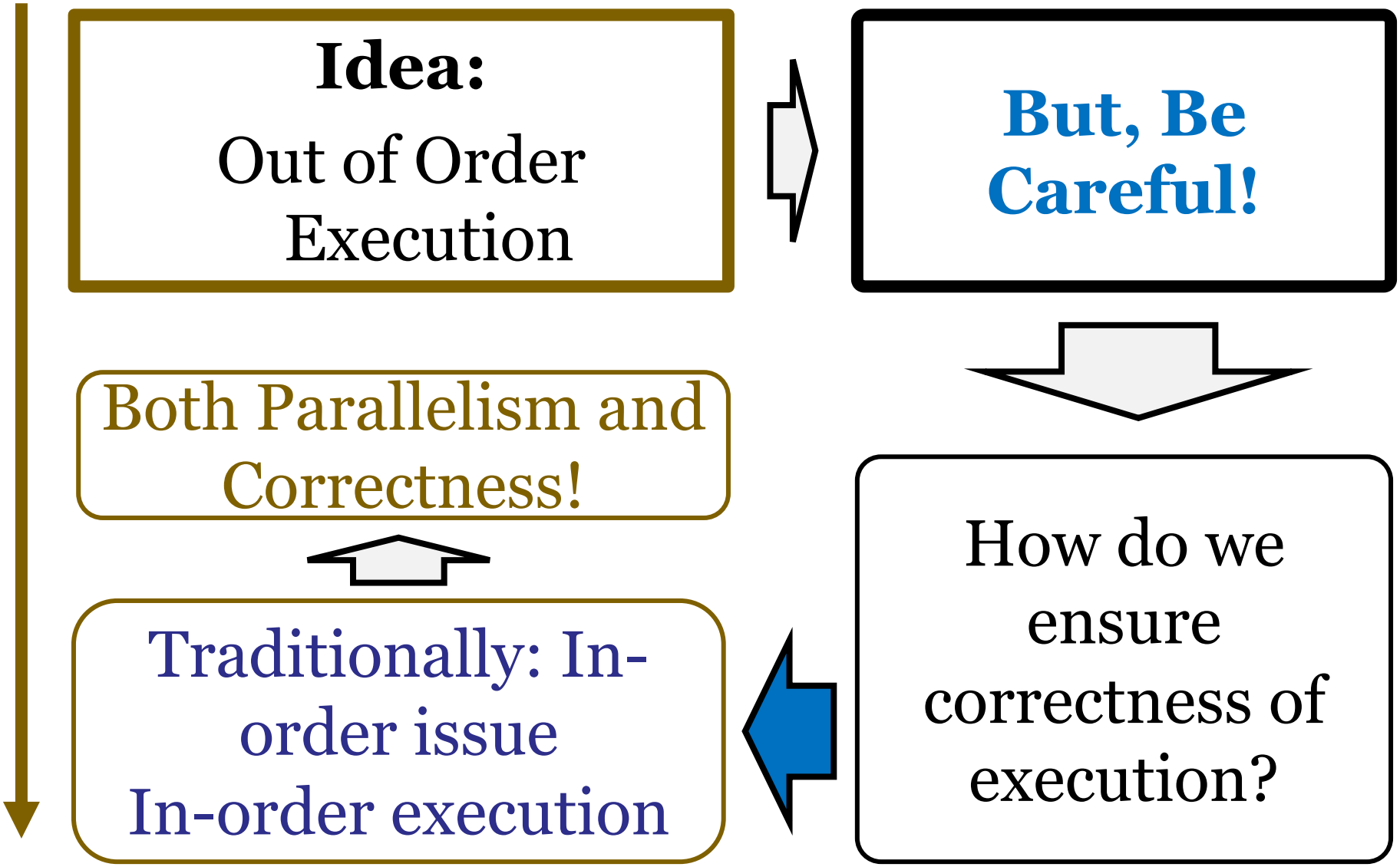
- There is a RAW hazard between DIV.D and ADD.D
- ADD.D should be stalled
- SUB.D is not dependent on previous instructions. Thus, it can be executed. But it is blocked by ADD.D

There is little parallelism among the instructions (if executed in order)

Can we execute instructions out of order? To get more parallelism

Yes! Out of Order Execution

Motivation


Idea:
Out of Order Execution
But, Be Careful!
How do we ensure correctness of execution?
Traditionally: In-order issue
In-order execution
Both Parallelism and Correctness!

Improvement Through Out-of-Order-Execution (hardware)

- The ID stage is divided into *two parts*
  - ***Part 1)*** checking for any structural hazard
  - ***Part 2)*** waiting for the absence of a data hazard

- Thus, ***in-order issue, out-of-order execution***
  - We can use in-order instruction issue (instructions issued in *program order*),
  - But we want an instruction to begin execution as soon as its data operands are available.

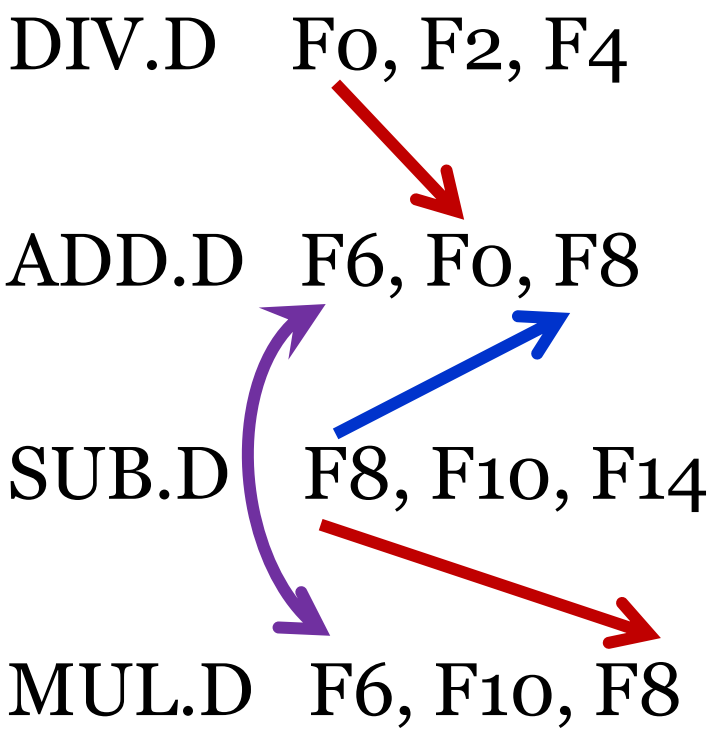
Issues with Out-of-Order Execution

List all WAR and WAW hazards

RAW

WAR

WAW



Dynamic Scheduling

Dynamic Scheduling:
Rearrange order of instructions to reduce stalls while maintaining data flow

- **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
  - Complicates exceptions

Two Algorithms for Dynamic Scheduling

Scoreboard Approach

- Adopted by CDC6600 in 1963

Tomasulo's Approach

- Tracks when operands are available
- Introduces **register renaming** in hardware
  - Minimizes WAW and WAR hazards

Assignment 3

- Appendix C:
  - C.1 a-d
  - C.3
- Online submission. Please check oc.sjtu.edu.cn for more details.

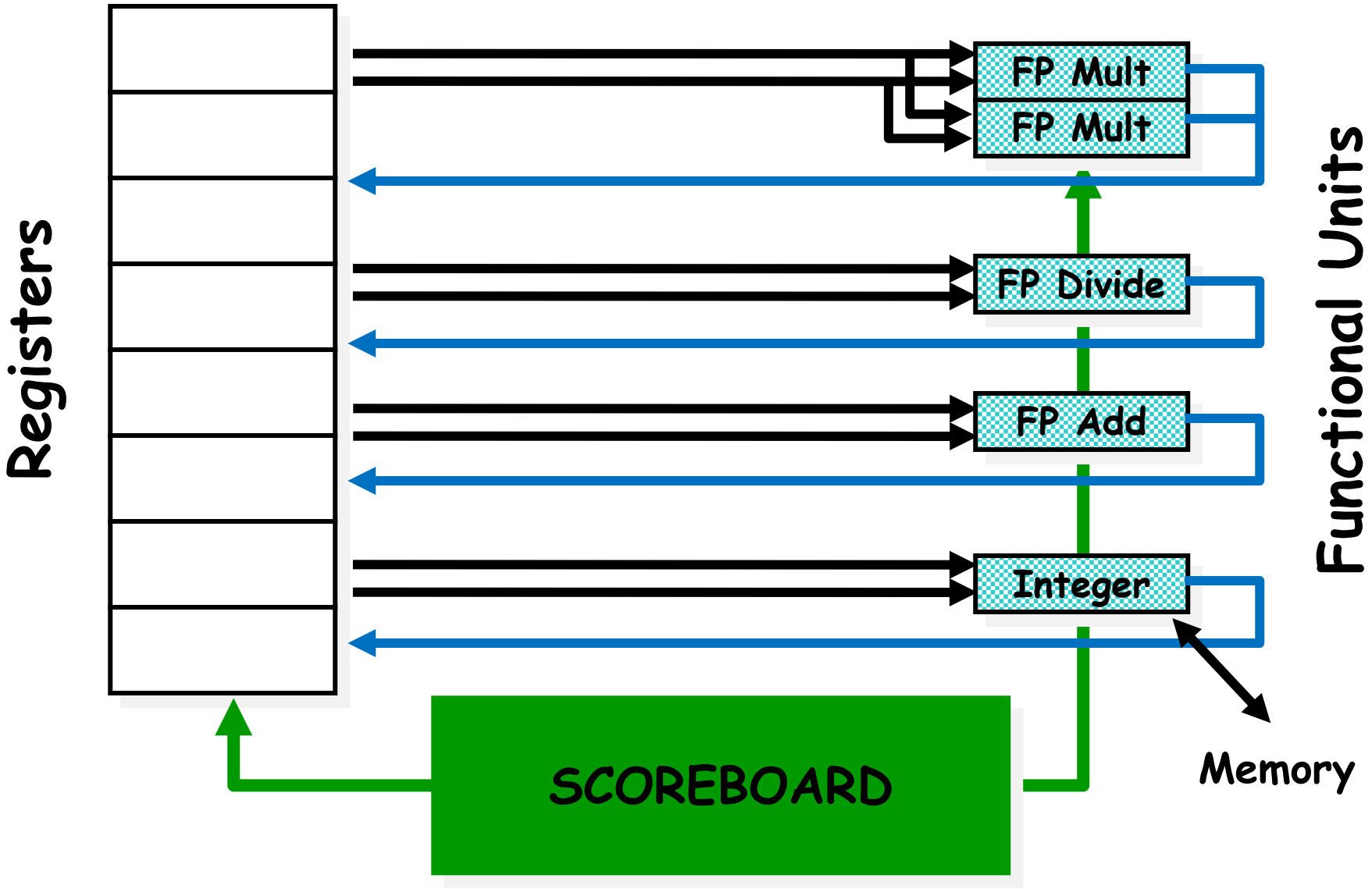
Outline

- 3.1 Concepts and Challenges
- 3.2 Basic Compiler Techniques
- 3.3 Reducing Branch Costs with Advanced Branch Prediction
- 3.4 Overcoming Data Hazards with Dynamic Scheduling
  - Scoreboarding Algorithm
  - Tomasulo's Algorithm
- 3.5 Hardware-based Speculation

Scoreboard: a bookkeeping technique

- Out-of-order execution divides ID stage:
  1. Issue—decode instructions, check for structural hazards
  2. Read operands—wait until no data hazards, then read operands
- Scoreboards date back to CDC6600 in 1963
- Instructions execute whenever *not dependent on previous instructions* and *no hazards*.
- CDC 6600: In order issue, out-of-order execution, out-of-order commit (or completion)
  - No forwarding!
  - Imprecise interrupt/exception model for now

Scoreboard Architecture (CDC 6600)



Scoreboard Implications

- **The key question: Out-of-order completion** => WAR, WAW hazards?
- **Solutions for WAR:**
  - Detect hazard and stall writeback until registers have been read
  - Read registers only during Read Operands stage
- **Solution for WAW:**
  - Detect hazard and stall issue of new instruction until other instruction completes
- Need to have multiple instructions in execution phase => multiple execution units or pipelined execution units
- Scoreboard keeps track of dependencies between instructions that have already issued.

Four Stages of Scoreboard Control

- 1) Issue—decode instructions & check for structural hazards (ID1)
  - Instructions issued in program order (for hazard checking)
  - Don't issue if structural hazard
  - Don't issue if instruction is output dependent on any previously issued but uncompleted instruction (**no WAW hazards**)
- 2) Read operands—wait until no data hazards, then read operands (ID2)
  - All real dependencies (**RAW hazards**) resolved in this stage, since we wait for instructions to write back data.
  - No forwarding of data in this model!

Four Stages of Scoreboard Control

- 3) Execution—operate on operands (EX)
  - The functional unit begins execution upon receiving operands. When the result is ready, it notifies the scoreboard that it has completed execution.
- 4) Write result—finish execution (WB)
  - Stall until no **WAR hazards** with previous instructions:

Example:

```
DIVD   F0,F2,F4
ADDD   F10,F0,F8
SUBD   F8,F8,F14
```

CDC 6600 scoreboard would stall SUBD until ADDD reads operands

Three Parts of the Scoreboard

- Instruction status:
  Which of 4 steps the instruction is in
- Functional unit status:—Indicates the state of the functional unit (FU). 9 fields for each functional unit
  - Busy: Indicates whether the unit is busy or not
    Op: Operation to perform in the unit (e.g., + or –)
    Fi: Destination register
    Fj,Fk: Source-register numbers
    Qj,Qk: Functional units producing source registers Fj, Fk
    Rj,Rk: Flags indicating when Fj, Fk are ready
- Register result status—Indicates which functional unit will write each register, if one exists. Blank when no pending instructions will write that 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									

Detailed Scoreboard Pipeline Control

Instruction status	Wait until	Bookkeeping
Issue	Not busy (FU) and not result(D)	Busy(FU)← yes; Op(FU)← op; Fi(FU)← `D'; Fj(FU)← `S1'; Fk(FU)← `S2'; Qj← Result('S1'); Qk← Result(`S2'); Rj← not Qj; Rk← not Qk; ('D')← FU;
Read operands	Rj and Rk	Rj← No; Rk← No
Execution complete	Functional unit done	
Write result	∀f((Fj(f)≠Fi(FU) or Rj(f)=No) & (Fk(f) ≠Fi(FU) or Rk(f)=No))	∀f(if Qj(f)=FU then Rj(f)← Yes); ∀f(if Qk(f)=FU then Rj(f)← Yes); Result(Fi(FU))← Ø; Busy(FU)← No

Result() denotes the content in the register status

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

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									

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							

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 cycl

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			

Scoreboard Example: Cycle 8b (Second half of clock c

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			

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

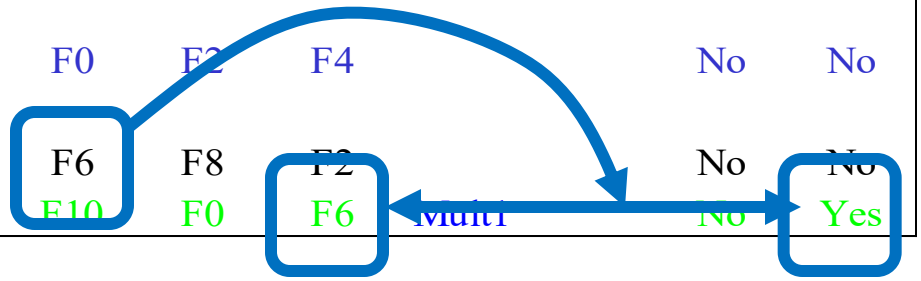
Instruction status:

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			

Faster than light computation (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 Example: 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

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
62	*FU*									

Review: Scoreboard Example: 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

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
62	FU									

- In-order issue; out-of-order execute & commit

Limitations of 6600 scoreboard:

- No forwarding hardware
- Limited to instructions in basic block (small window)
- Small number of functional units (structural hazards), especially integer/load store units
- Do not issue on structural hazards
- Wait for WAR hazards
- Prevent WAW hazards

Outline

- 3.1 Concepts and Challenges
- 3.2 Basic Compiler Techniques
- 3.3 Reducing Branch Costs with Advanced Branch Prediction
- 3.4 Overcoming Data Hazards with Dynamic Scheduling
  - Scoreboarding Algorithm
  - Tomasulo's Algorithm
- 3.5 Hardware-based Speculation

Tomasulo's Algorithm - Overview

- A technique for allowing instructions to *execute out of order* when there are sufficient resources and no data dependences
- Can be extended to handle antidependences and output dependences by effectively renaming the registers dynamically
- Can be extended to handle speculation
  - A technique to reduce the effect of control dependences by predicting the outcome of a branch
- Notes
  - IBM 360/91
  - Invented by Robert Tomasulo

Dynamic Algorithm: Tomasulo's Algorithm

- For IBM 360/91– 3 years after CDC
- Goal: High Performance without special compilers
- Small number of floating point registers (4 in 360) prevented interesting compiler scheduling of operations
  - This led Tomasulo to try to figure out how to get more effective registers — renaming in hardware!
- *Why Study 1966 Computer?*
- The descendants of this have flourished!
  - Alpha 21264, HP 8000, MIPS 10000, Pentium III, PowerPC 604, …

Eliminate WAR and WAW by register renaming

- Original

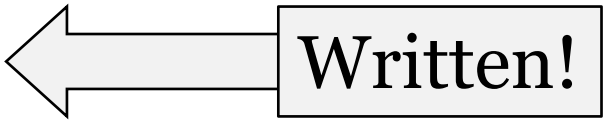
```
DIV.D    F0, F2, F4
ADD.D    F6, F0, F8
S.D      F6, 0(R1)
SUB.D    F8, F10, F14
MUL.D    F6, F10, F8
```

WAR between (F8)ADD.D and SUB.D, between (F6)S.D and MUL.D

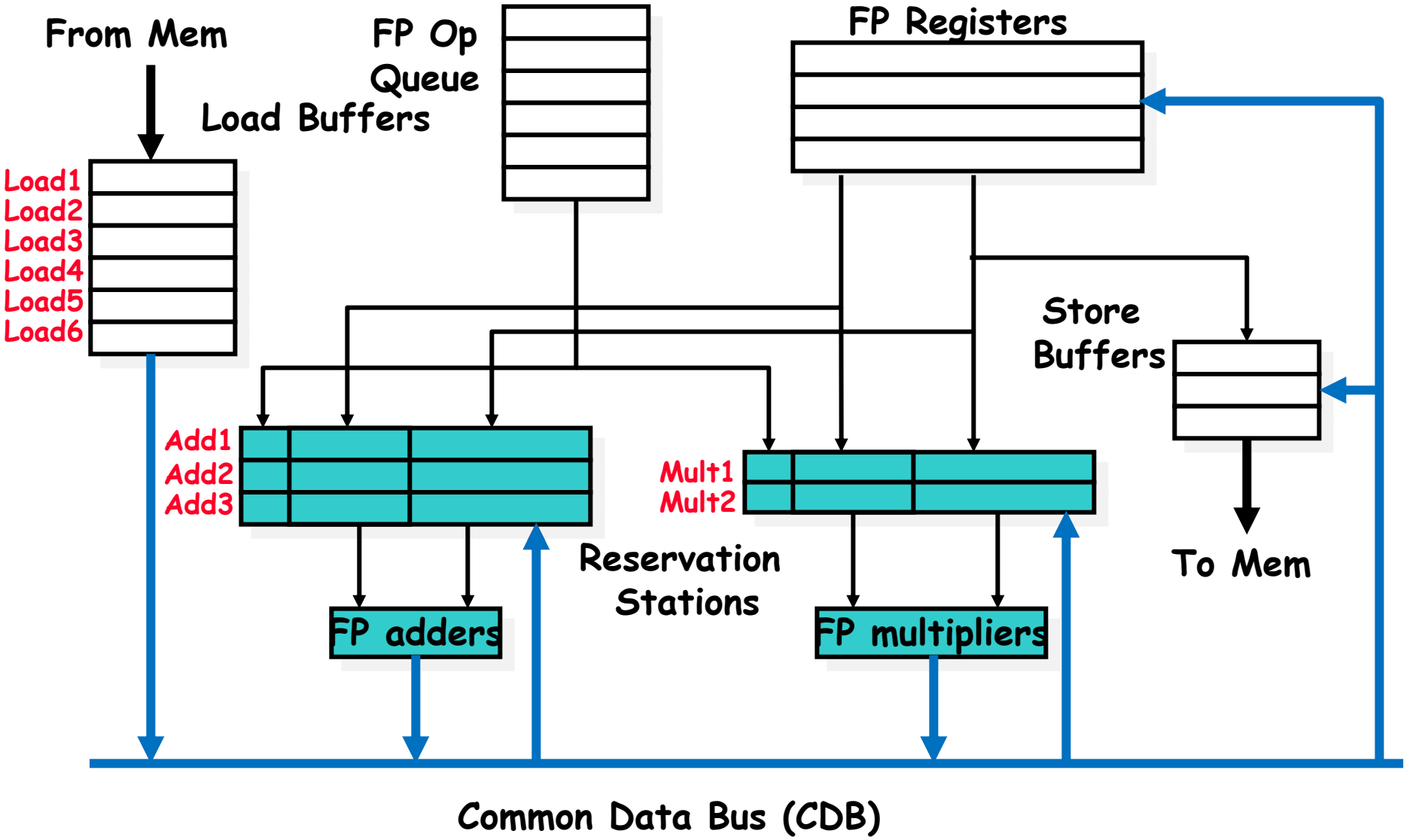
WAW between (F6)ADD.D and MUL.D

- Register renaming

```
DIV.D    F0, F2, F4
ADD.D    S, F0, F8
S.D      S, 0(R1)
SUB.D    T, F10, F14
MUL.D    F6, F10, T
```



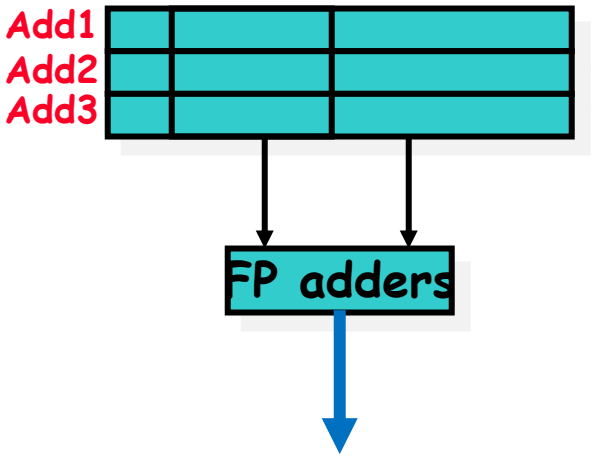
Basic Structure



Basic Idea

WAR

```
DIV.D   F0, F6, F4
ADD.D   F6, F2, F8
```



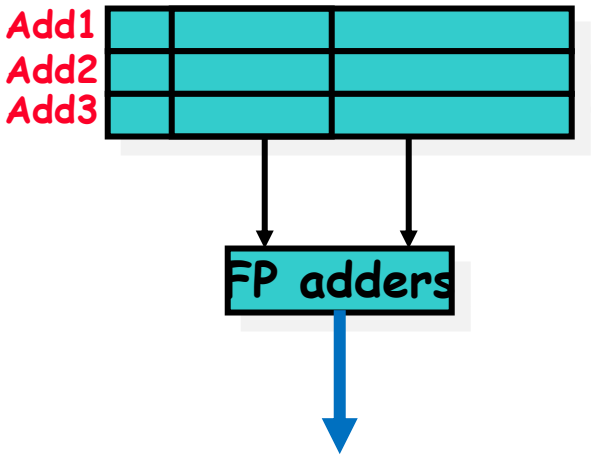
a reservation station fetches and buffers an operand as soon as it is available

eliminating the need to get the operand from a register

Basic Idea (Cont')

RAW

```
DIV.D   F0, F6, F4
ADD.D   F3, F0, F8
```



pending instructions designate the reservation station that will provide their input

Basic Idea (Cont')

WAW

```
DIV.D   F0, F6, F4
ADD.D   F0, F3, F8
```

when successive writes to a register overlap in execution, only the last one is actually used to update the register

Basic Idea (Cont')

Register specifiers => names of reservation stations

- As instructions are issued, the register specifiers for pending operands are renamed to the names of the reservation station, which provides register renaming
- more reservation stations than real registers

Properties of Tomasulo Algorithm

1. Control & buffers distributed with Function Units (FU)
   - Hazard detection and execution control are distributed
   - FU buffers called "reservation stations" which hold pending operands
   - Registers in instructions replaced by values or pointers to reservation stations(RS)
     - form of register renaming to avoid WAR, WAW hazards

2. Bypassing: Results passed directly to FU from RS, not through registers, over Common Data Bus
   - that broadcasts results to all FUs, so allows all units waiting for an operand to be loaded simultaneously

- Load and Stores treated as FUs with RSs as well

What states to be maintained?

Reservation Station Components

Busy: Indicates reservation station or FU is busy

Op: Operation to perform in the unit (e.g., + or –)

Vj, Vk: Value of Source operands

- Store buffers has A field, offset/effective memory address

Qj, Qk: Reservation stations producing source operands (value to be written)

- Store buffers only have Qi for RS producing result register.

Register result status—Indicates which functional unit will write each register, if one exists. Blank when no pending instructions that will write that register.

Three Stages of Tomasulo Algorithm

1. Issue—get instruction from FP Op Queue

   If reservation station free (no *structural hazard*),
   a. control issues instr & b. 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

Tomasulo Example

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	

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
0	FU									

Tomasulo 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

Example speed: 2 clocks for FP +,-; 10 for * ; 40 clks for /

Tomasulo 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 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 6600, can have multiple loads outstanding

Tomasulo 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 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?

Tomasulo 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			

Tomasulo 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 vs. scoreboard?**

Tomasulo 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 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 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 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 completing; what is waiting for it?

Tomasulo 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 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 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 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 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			

Tomasulo 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			

Faster than light computation (skip a couple of cycles)

Tomasulo 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 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 is completing; what is waiting for it?**

Tomasulo 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					
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			

- 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

Tomasulo Algorithm vs. Scoreboard

- Control & buffers distributed with Function Units (FU) vs. centralized in scoreboard;
  - FU buffers called "reservation stations"; have pending operands
- Registers in instructions replaced by values or pointers to reservation stations(RS); called register renaming ;
  - avoids WAR, WAW hazards
  - More reservation stations than registers, so can do optimizations compilers can't
- Results to FU from RS, not through registers, over Common Data Bus that broadcasts results to all FUs
- Load and Stores treated as FUs with RSs as well

Tomasulo v. Scoreboard (IBM 360/91 v. CDC 6600)

Tomasulo	Scoreboard
No issue on structural hazard	same
WAR: renaming avoids	stall completion
WAW: renaming avoids	stall issue
Broadcast results from FU	Write/read registers
Control: reservation stations	central scoreboard

Outline

- 3.1 Concepts and Challenges
- 3.2 Basic Compiler Techniques
- 3.3 Reducing Branch Costs with Advanced Branch Prediction
- 3.4 Overcoming Data Hazards with Dynamic Scheduling
- **3.5 Hardware-based Speculation**

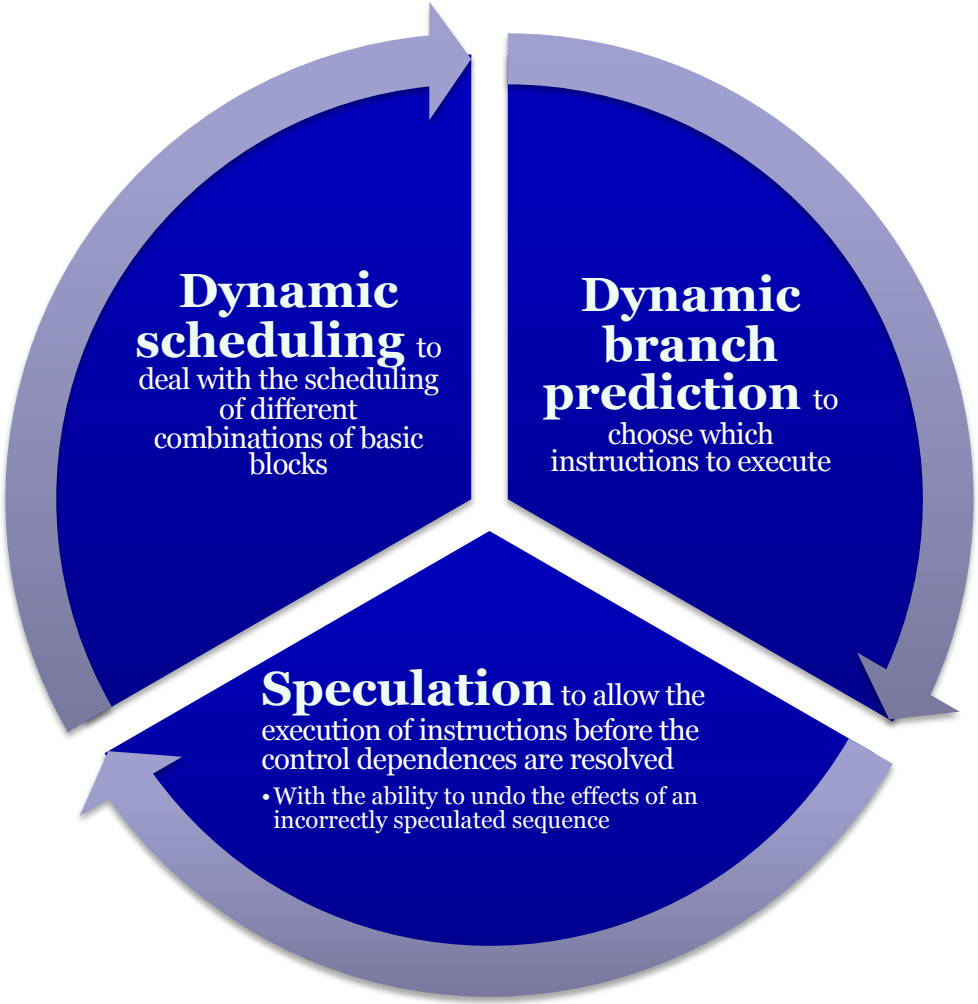
Motivation

- What we have learned (dynamic scheduling) is able to exploit only the parallelism within a basic block
  - The parallelism is too limited!
- As we try to exploit more instruction-level parallelism, *maintaining control dependences* becomes an increasing burden.
- **Thus, exploiting more parallelism requires that we overcome the limitation of control dependences**

Overview of Speculation

- We overcome control dependence by
  - (1) speculating on the outcome of branches and
  - (2) executing the program as if our guesses were correct
- This essentially **enlarges instruction-level parallelism** (previously only limited to a basic block!)
- **Mechanisms** are needed to handle the situation where the speculation is incorrect

Three Key Ideas of Hardware Speculation



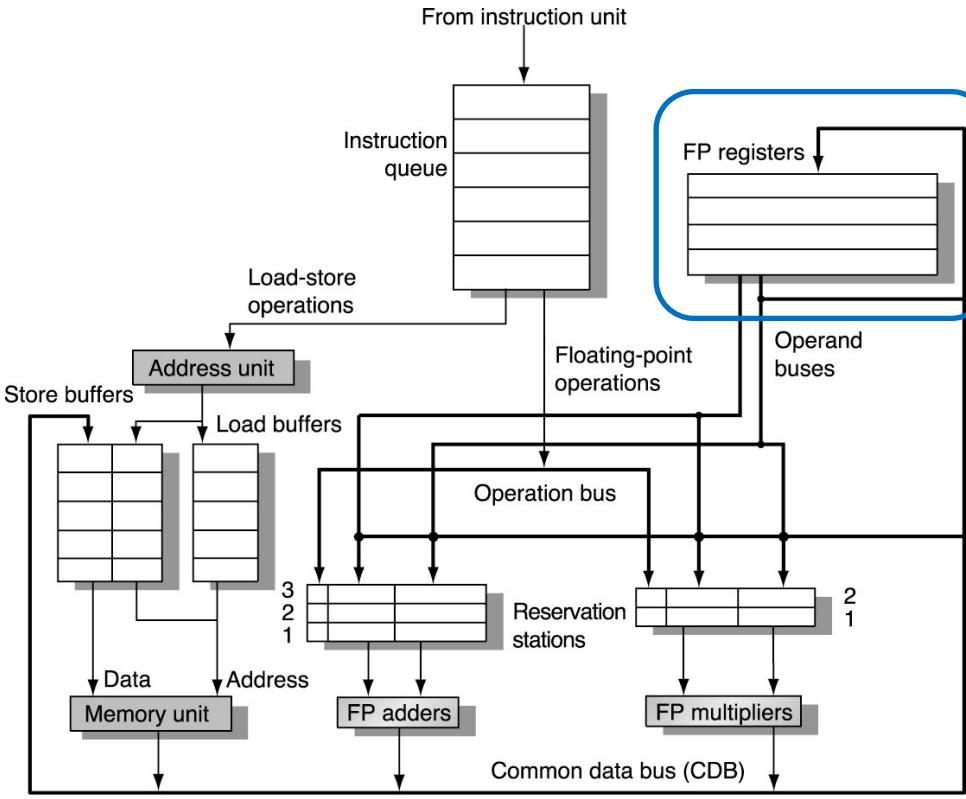
Main Challenge of Speculation

- What if a branch instruction is mispredicted?
  - Mispredicted instructions may have already executed and completed
  - Meaning that, the results of these instructions already written to registers and memory locations
- We must address this challenge when taking advantage of speculations

Key Idea: add buffers to hold speculated results

How to extend Tomasulo's algorithm to support speculation?

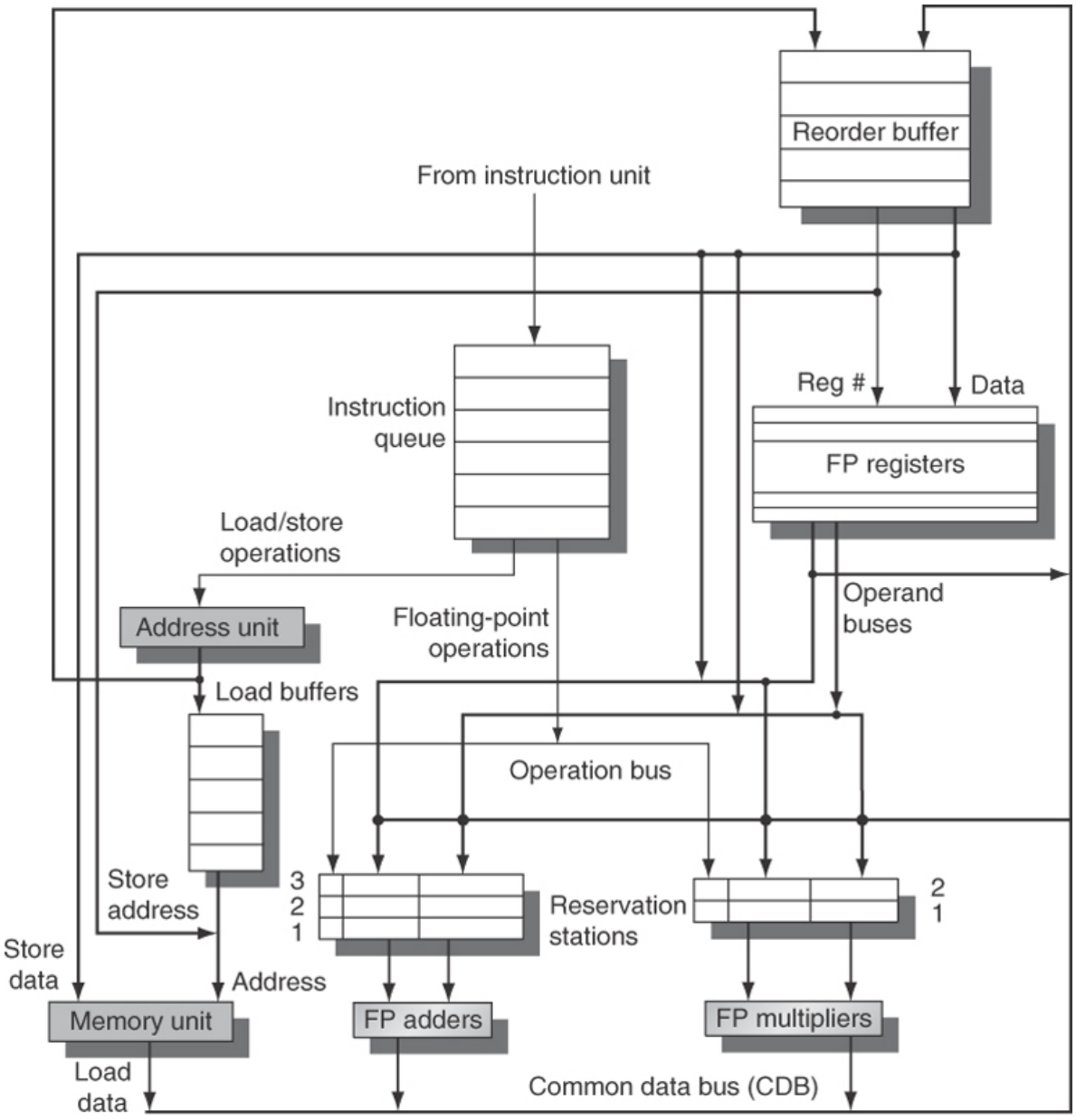
We must separate the bypassing of results among instructions from the actual completion of an instruction



Key Idea behind Implementing Speculation

To allow instructions to **execute out of order**
But to force them to **commit in order**

- Adding this commit phase to the instruction execution sequence requires an additional set of hardware buffers that hold the results of instructions that have finished execution but have not committed.
- This is ***Reorder Buffer***



The basic structure of a FP unit using Tomasulo's algorithm and extended to handle speculation.

Reorder Buffer

- Reorder buffer – holds the result of instruction between completion and commit
- Four fields:
  - Instruction type: branch/store/register
  - Destination field: register number or mem location
  - Value field: output value
  - Ready field: completed execution?
- Modify reservation stations:
  - Operand source is now *reorder buffer instead of functional unit*

Reorder Buffer

- Register values and memory values are not written until an instruction commits
- On ***misprediction***:
  - Speculated entries in ROB are cleared
- Exceptions:
  - Not recognized until it is ready to commit

Four Steps

1) Issue:

- Get an instruction from the instruction queue
- Condition: if there is an empty reservation station and an empty slot in the ROB
- Send operands to reservation station if they are available in either register or ROB
- The # for allocated ROB buffer for result is also sent to reservation station (to receive data on CDB)

2) Execute

- If operands not all ready, wait and monitor the CDB
- If ready, execute

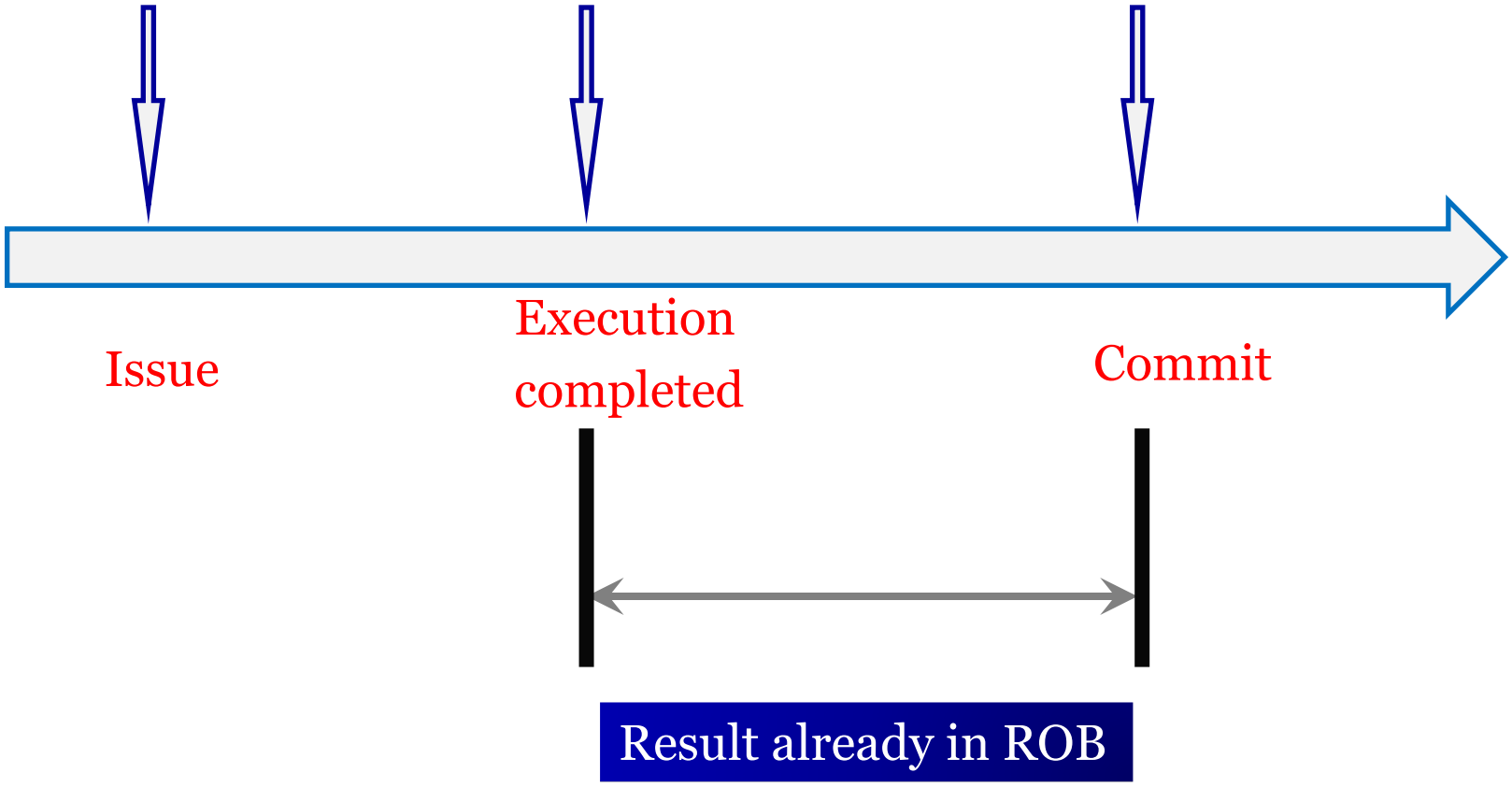
3) Write result

- Write it on CDB (with the ROB tag sent)
  - Send to ROB
  - Send to other RS's waiting for the result

4) Commit

- When an instruction reaches the head of ROB and its results are ready, update register file or mem location
- When a branch with incorrect prediction reaches the head of ROB, the ROB is flushed and execution is restarted!

Instruction Lifecycle



Example – without branch instruction

Program order

Reorder buffer

Entry	Busy	Instruction		State	Destination	Value
1	No	L.D	F6,32(R2)	Commit	F6	Mem[32 + Regs[R2]]
2	No	L.D	F2,44(R3)	Commit	F2	Mem[44 + Regs[R3]]
3	Yes	MUL.D	F0,F2,F4	Write result	F0	#2 × Regs[F4]
4	Yes	SUB.D	F8,F2,F6	Write result	F8	#2 – #1
5	Yes	DIV.D	F10,F0,F6	Execute	F10	
6	Yes	ADD.D	F6,F8,F2	Write result	F6	#4 + #2

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[44 + Regs[R3]]	Regs[F4]			#3	
Mult2	Yes	DIV.D		Mem[32 + Regs[R2]]	#3		#5	

FP register status

Field	F0	F1	F2	F3	F4	F5	F6	F7	F8	F10
Reorder #	3						6		4	5
Busy	Yes	No	No	No	No	No	Yes	. . .	Yes	Yes

Example – with branch instruction

```
Loop:   L.D      F0,0(R1)
        MUL.D    F4,F0,F2
        S.D      F4,0(R1)
        DADDIU   R1,R1,#-8
        BNE      R1,R2,Loop     ;branches if R1¦R2
```

Assume that we have issued all the instructions in the loop twice. Let's also assume that the L.D and MUL.D from the first iteration have committed and all other instructions have completed execution. Normally, the store would wait in the ROB for both the effective address operand (R1 in this example) and the value (F4 in this example). Since we are only considering the floating-point pipeline, assume the effective address for the store is computed by the time the instruction is issued.

Example – with branch instruction

Program order

Reorder buffer

Entry	Busy	Instruction		State	Destination	Value
1	No	L.D	F0,0(R1)	Commit	F0	Mem[0 + Regs[R1]]
2	No	MUL.D	F4,F0,F2	Commit	F4	#1 × Regs[F2]
3	Yes	S.D	F4,0(R1)	Write result	0 + Regs[R1]	#2
4	Yes	DADDIU	R1,R1,#-8	Write result	R1	Regs[R1] – 8
5	Yes	BNE	R1,R2,Loop	Write result		
6	Yes	L.D	F0,0(R1)	Write result	F0	Mem[#4]
7	Yes	MUL.D	F4,F0,F2	Write result	F4	#6 × Regs[F2]
8	Yes	S.D	F4,0(R1)	Write result	0 + #4	#7
9	Yes	DADDIU	R1,R1,#-8	Write result	R1	#4 – 8
10	Yes	BNE	R1,R2,Loop	Write result		

FP register status

Field	F0	F1	F2	F3	F4	F5	F6	F7	F8
Reorder #	6				7				
Busy	Yes	No	No	No	Yes	No	No	...	No

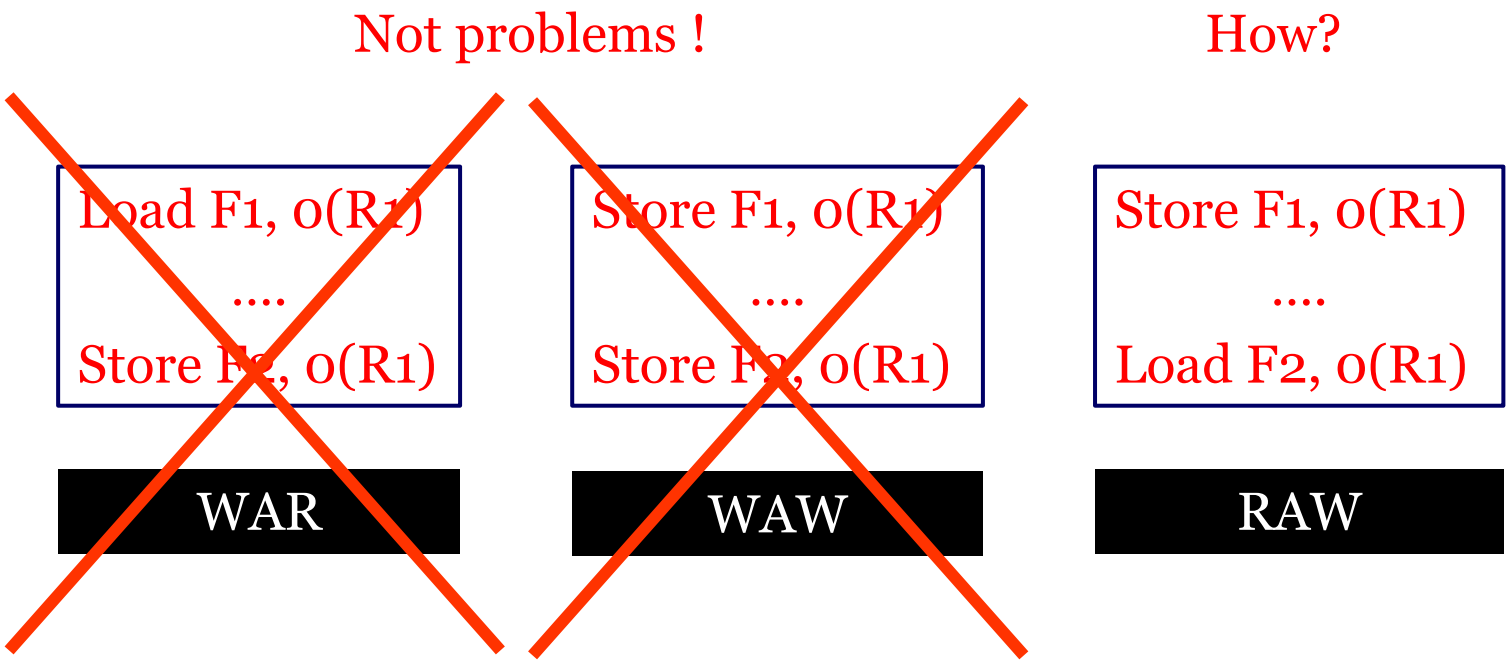
Renaming?

- In Tomasulo's algorithm, the renaming function is via reservation stations?
- In hardware based speculation, how is the renaming function achieved?

Data Hazards through Memory

How to handle them in hardware based speculations?

Data Hazards through Memory



- For RAW, we add two restrictions
  - (1) maintaining effective address calculation of a load with respect to all earlier stores
  - (2) not allowing to execute a load if any stores in ROB has a destination matching the load target