Lecture 14

Computer Architecture
(Spring 2020)

Instruction-Level Parallelism

Dr. Duo Liu (刘铎)
Office: Main Building 0626
Email: liuduo@cqu.edu.cn

Dynamic Scheduling & Data Hazards: The Scoreboard Approach

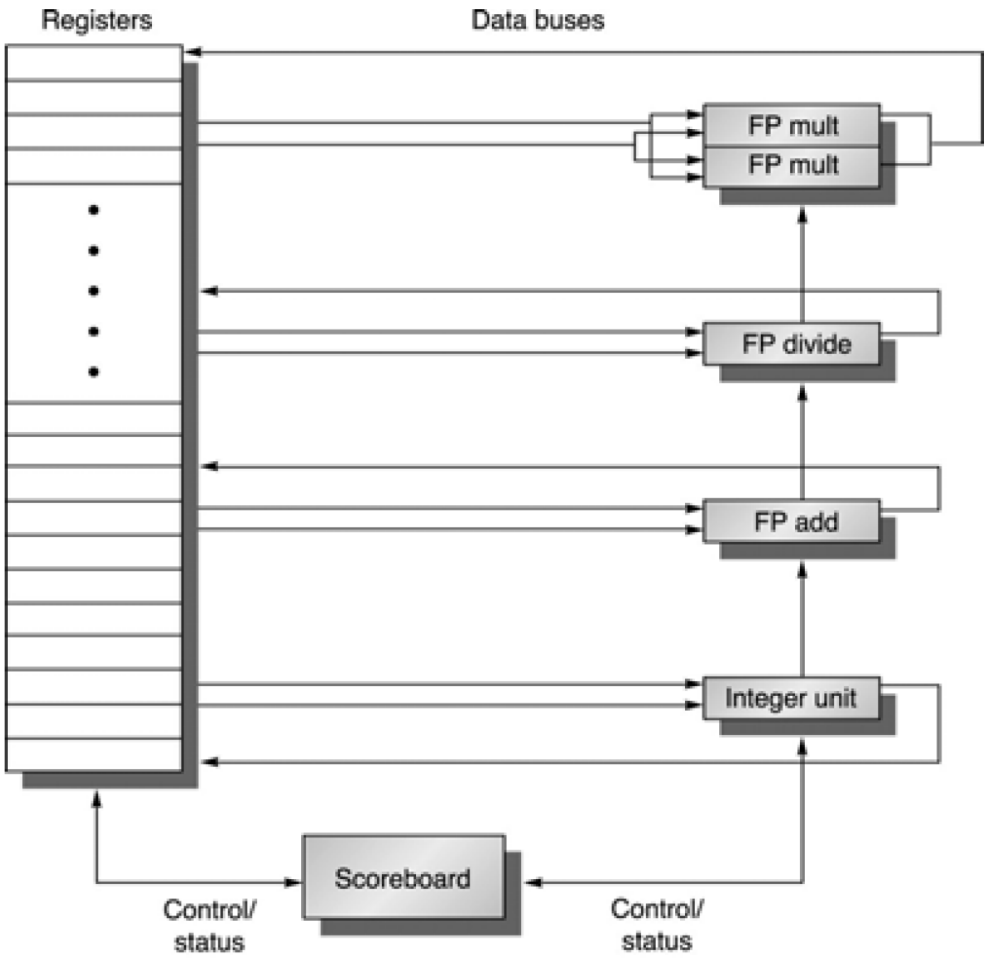
```
DIV.D   F0,  F2,  F4
ADD.D   F6,  F0,  F8
SUB.D   F8,  F8,  F14
MUL.D   F6,  F10, F9
```

- anti-dependency between ADD.D and SUB.D (may lead to WAR hazard)
- output dependency between MUL.D and ADD.D (may lead to WAW hazard)

- Goal: to maintain CPI≈1 with out-of-order execution by
  - issuing an instruction ASAP
  - taking care of issuing/execution, including all hazard detections
    1. checking for structural hazards
    2. waiting for absence of data hazards
- Scoreboard was introduced in 1963 with CDC 6600
  - 7 units for integer operations
  - 5 memory reference units
  - 4 FP units

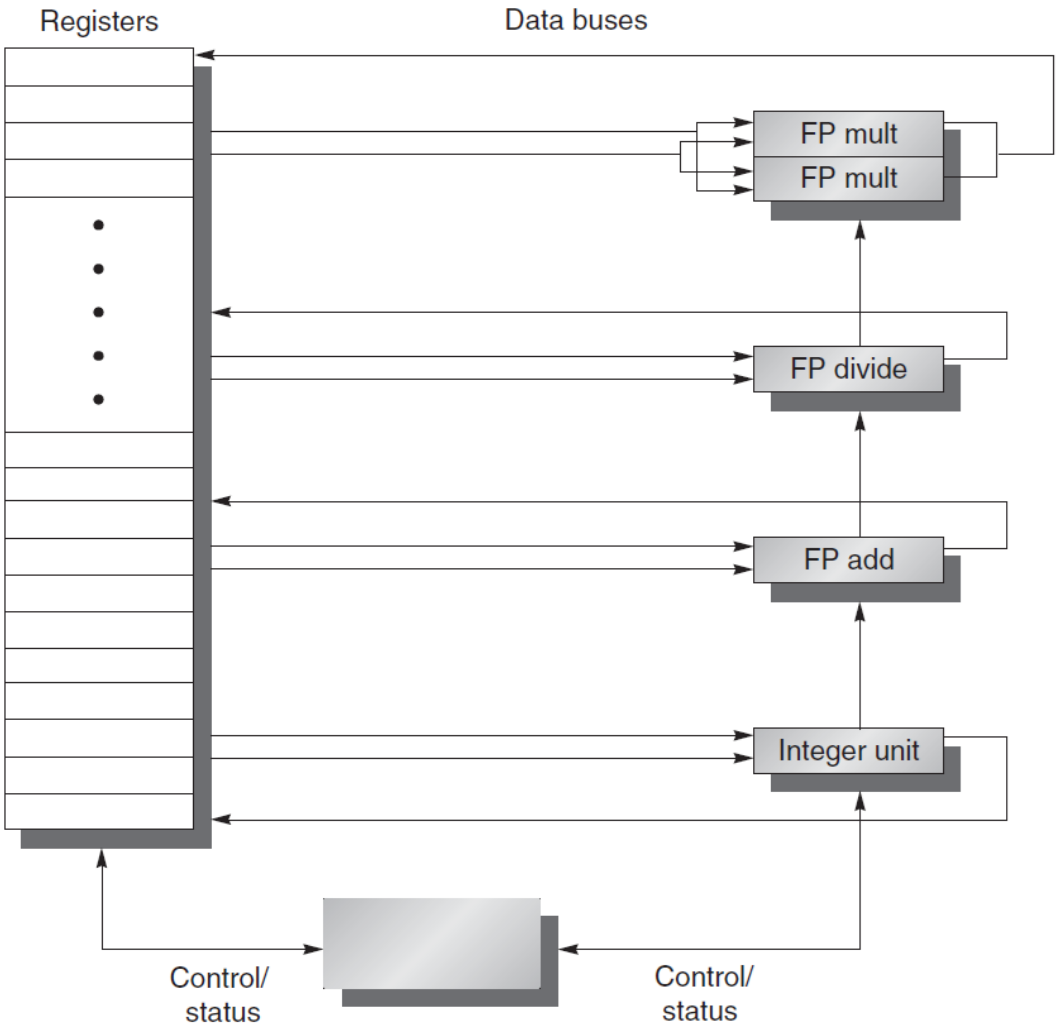
Assumption: Basic Architecture of MIPS Processor with Scoreboard

- Simpler structure than CDC 6600 original one
- For MIPS, focus on FP units (other units have small latencies)
- Assume
  - 1 integer unit
    - integer operations
    - memory references
    - branches
  - 1 FP adder (2 cycles)
  - 2 FP multipliers (10 cycles)
  - 1 FP divider (40 cycles)



Scoreboard

- All instructions go through Scoreboard, and Scoreboard record all the data dependencies.
- It can determine when an instruction can read its operands and begin execution.
  - If an instruction has to be stalled (RAW hazards), scoreboard can also find the earliest time when its operands are available.
- The scoreboard also controls when an instruction can write its result into the destination register.
  - To avoid WAW and WAR

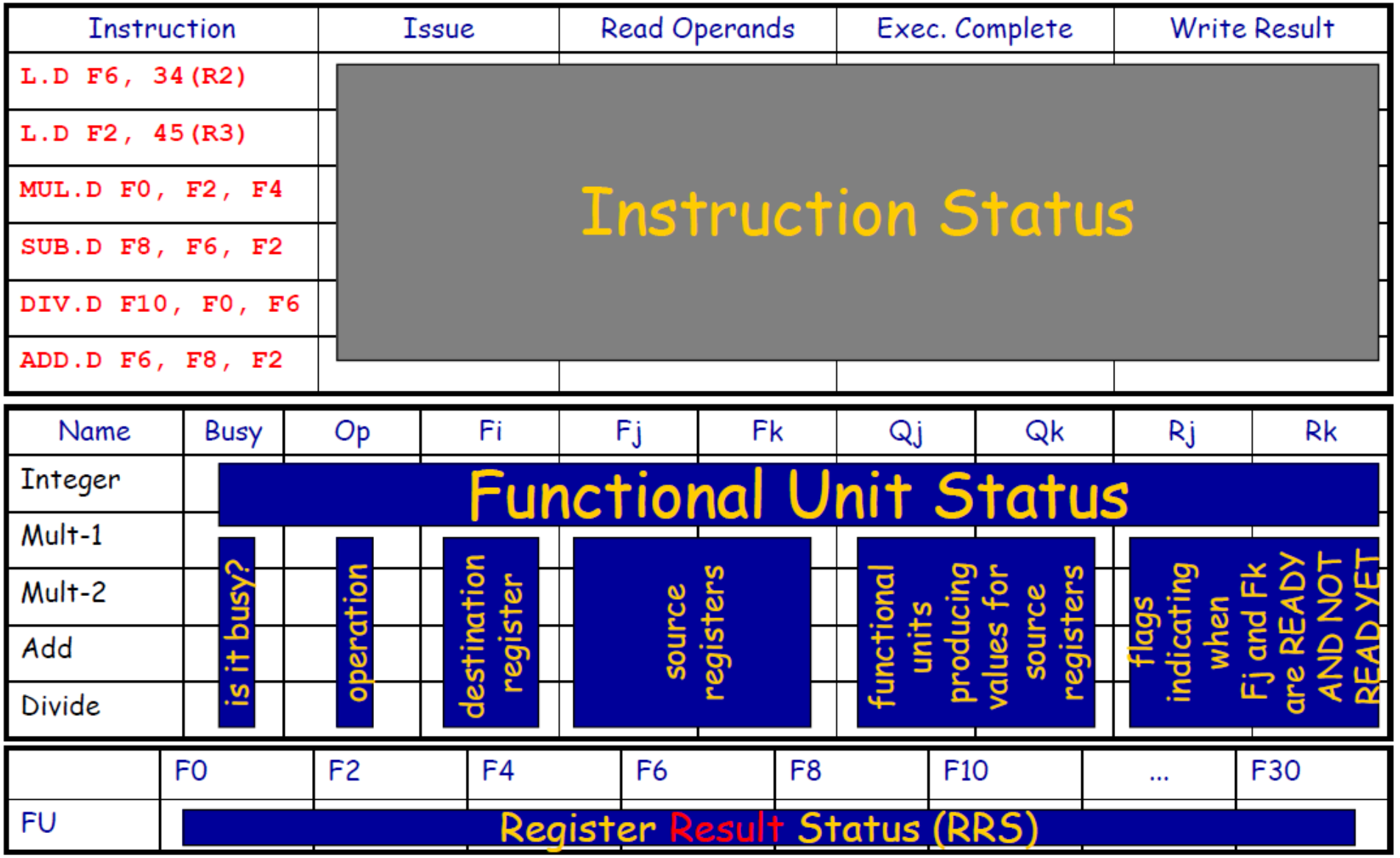


MIPS Pipeline Stages with Scoreboard

- All hazard detection and resolution is centralized in the *scoreboard module*
  - when operands can be read
  - when execution can start
  - when result can be written
- In-order issuing, out-of-order execution and commit
- The ID, EX, and WB stages are replaced by four new stages

1. Issue (ID-1)
   - in-order instruction decoding
   - check/stall for structural hazards
   - check/stall for WAW hazards
2. Read Operands (ID-2)
   - wait for each operand availability
   - resolve RAW hazards dynamically
   - no forwarding, but wait for WB
3. Execution (EX)
   - out-of-order execution
   - inform scoreboard upon completion
4. Write Result (WB)
   - check/stall for WAR hazards
   - write result into register asap
   - no statically-assigned write slot

MIPS Scoreboard Components



Scoreboard: Checking & Bookkeeping per Instruction (op D, S1, S2)

Instruction Status	Wait until	Bookkeeping
1. Issue	not Busy[FU] and not RRS[D]	Busy[FU] ← true; Op[FU] ← op; Fi[FU] ← D; Fj[FU] ← S1; Fk[FU] ← S2; Qj ← RRS[S1]; Qk ← RRS[S2]; Rj ← (Qj==0); Rk ← (Qk==0); RRS[D] ← FU
2. Read Operands	Rj=true and Rk=true	Rj ← false; Rk ← false; Qj ← 0; Qk ← 0
3. Execution Complete	FU is done executing	
4. Write Result	∀f { (Fj[f] ≠ Fi[FU] or Rj[f] = false) & (Fk[f] ≠ Fi[FU] or Rk[f] = false) }	∀f((Qj[f] = FU) ⇒ Rj[f] ← true) ∀f((Qk[f] = FU) ⇒ Rk[f] ← true) RSS[Fi[FU]] ← 0; Busy[FU] ← false

Scoreboard Example: Clock Cycle = 1

Instruction	Issue	Read Operands	Exec. Complete	Write Result
L.D F6, 34(R2)	1			
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				

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	yes	L.D	F6		R2				true
Mult-1	no								
Mult-2	no								
Add	no								
Divide	no								

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

- issue first instruction

Scoreboard Example: Clock Cycle = 2

Instruction	Issue	Read Operands	Exec. Complete	Write Result
L.D F6, 34(R2)	1	2		
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				

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	yes	L.D	F6		R2				false
Mult-1	no								
Mult-2	no								
Add	no								
Divide	no								

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

• read operand; second load instruction cannot be issued (struct. hazard)

Scoreboard Example: Clock Cycle = 3

Instruction	Issue	Read Operands	Exec. Complete	Write Result
L.D F6, 34(R2)	1	2	3	
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				

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	yes	L.D	F6		R2				false
Mult-1	no								
Mult-2	no								
Add	no								
Divide	no								

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

- execute first load instruction; MUL.D waits too (issuing stage is stalled)

Scoreboard Example: Clock Cycle = 4

Instruction	Issue	Read Operands	Exec. Complete	Write Result
L.D F6, 34(R2)	1	2	3	4
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				

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	no								
Mult-2	no								
Add	no								
Divide	no								

	F0	F2	F4	F6	F8	F10	...	F30
FU								

- first load instruction commits (writing F6)

Scoreboard Example: Clock Cycle = 5

Instruction	Issue	Read Operands	Exec. Complete	Write Result
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	5			
MUL.D F0, F2, F4				
SUB.D F8, F6, F2				
DIV.D F10, F0, F6				
ADD.D F6, F8, F2				

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	yes	L.D	F2		R3				true
Mult-1	no								
Mult-2	no								
Add	no								
Divide	no								

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

- issue second load instruction

Scoreboard Example: Clock Cycle = 6

Instruction	Issue	Read Operands	Exec. Complete	Write Result
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	5	6		
MUL.D F0, F2, F4	6			
SUB.D F8, F6, F2				
DIV.D F10, F0, F6				
ADD.D F6, F8, F2				

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	yes	L.D	F2		R3				false
Mult-1	yes	MUL.D	F0	F2	F4	Integer		false	true
Mult-2	no								
Add	no								
Divide	no								

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult-1	Integer						

- read operands of second load instruction; issue multiply instruction

Scoreboard Example: Clock Cycle = 7

Instruction	Issue	Read Operands	Exec. Complete	Write Result
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	5	6	7	
MUL.D F0, F2, F4	6			
SUB.D F8, F6, F2	7			
DIV.D F10, F0, F6				
ADD.D F6, F8, F2				

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	yes	L.D	F2		R3				false
Mult-1	yes	MUL.D	F0	F2	F4	Integer		false	true
Mult-2	no								
Add	yes	SUB.D	F8	F6	F2		Integer	true	false
Divide	no								

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult-1	Integer			Add			

- execute second load instruction; stall multiply; issue subtraction

Scoreboard Example: Clock Cycle = 8

Instruction	Issue	Read Operands	Exec. Complete	Write Result
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	5	6	7	8
MUL.D F0, F2, F4	6			
SUB.D F8, F6, F2	7			
DIV.D F10, F0, F6	8			
ADD.D F6, F8, F2				

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	yes	MUL.D	F0	F2	F4	Integer		true	true
Mult-2	no								
Add	yes	SUB.D	F8	F6	F2		Integer	true	true
Divide	yes	DIV.D	F10	F0	F6	Mult-1		false	true

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult-1				Add	Divide		

- issue division; second load instruction is done; F2 ready (but not read yet)

Scoreboard Example: Clock Cycle = 9

Instruction	Issue	Read Operands	Exec. Complete	Write Result
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	5	6	7	8
MUL.D F0, F2, F4	6	9		
SUB.D F8, F6, F2	7	9		
DIV.D F10, F0, F6	8			
ADD.D F6, F8, F2				

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	yes	MUL.D	F0	F2	F4			false	false
Mult-2	no								
Add	yes	SUB.D	F8	F6	F2			false	false
Divide	yes	DIV.D	F10	F0	F6	Mult-1		false	true

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult-1				Add	Divide		

- reading operands of multiplication and subtraction; addition not issued

Scoreboard Example: Clock Cycle = 10

Instruction	Issue	Read Operands	Exec. Complete	Write Result
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	5	6	7	8
MUL.D F0, F2, F4	6	9		
SUB.D F8, F6, F2	7	9		
DIV.D F10, F0, F6	8			
ADD.D F6, F8, F2				

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	yes	MUL.D	F0	F2	F4			false	false
Mult-2	no								
Add	yes	SUB.D	F8	F6	F2			false	false
Divide	yes	DIV.D	F10	F0	F6	Mult-1		false	true

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult-1				Add	Divide		

- multiplication (10 cycles) and subtraction (2 cycles) are executing

Scoreboard Example: Clock Cycle = 11

Instruction	Issue	Read Operands	Exec. Complete	Write Result
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	5	6	7	8
MUL.D F0, F2, F4	6	9		
SUB.D F8, F6, F2	7	9	11	
DIV.D F10, F0, F6	8			
ADD.D F6, F8, F2				

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	yes	MUL.D	F0	F2	F4			false	false
Mult-2	no								
Add	yes	SUB.D	F8	F6	F2			false	false
Divide	yes	DIV.D	F10	F0	F6	Mult-1		false	true

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult-1				Add	Divide		

- subtraction execution is completed

Scoreboard Example: Clock Cycle = 12

Instruction	Issue	Read Operands	Exec. Complete	Write Result
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	5	6	7	8
MUL.D F0, F2, F4	6	9		
SUB.D F8, F6, F2	7	9	11	12
DIV.D F10, F0, F6	8			
ADD.D F6, F8, F2				

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	yes	MUL.D	F0	F2	F4			false	false
Mult-2	no								
Add	no								
Divide	yes	DIV.D	F10	F0	F6	Mult-1		false	true

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult-1					Divide		

- subtraction is done and result is written into F8

Scoreboard Example: Clock Cycle = 13

Instruction	Issue	Read Operands	Exec. Complete	Write Result
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	5	6	7	8
MUL.D F0, F2, F4	6	9		
SUB.D F8, F6, F2	7	9	11	12
DIV.D F10, F0, F6	8			
ADD.D F6, F8, F2	13			

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	yes	MUL.D	F0	F2	F4			false	false
Mult-2	no								
Add	yes	ADD.D	F6	F8	F2			true	true
Divide	yes	DIV.D	F10	F0	F6	Mult-1		false	true

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult-1			Add		Divide		

- addition is issued because FP adder is now available

Scoreboard Example: Clock Cycle = 14

Instruction	Issue	Read Operands	Exec. Complete	Write Result
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	5	6	7	8
MUL.D F0, F2, F4	6	9		
SUB.D F8, F6, F2	7	9	11	12
DIV.D F10, F0, F6	8			
ADD.D F6, F8, F2	13	14		

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	yes	MUL.D	F0	F2	F4			false	false
Mult-2	no								
Add	yes	ADD.D	F6	F8	F2			false	false
Divide	yes	DIV.D	F10	F0	F6	Mult-1		false	true

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult-1			Add		Divide		

- addition operands are read and execution starts (will take two cycles)

Scoreboard Example: Clock Cycle = 15

Instruction	Issue	Read Operands	Exec. Complete	Write Result
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	5	6	7	8
MUL.D F0, F2, F4	6	9		
SUB.D F8, F6, F2	7	9	11	12
DIV.D F10, F0, F6	8			
ADD.D F6, F8, F2	13	14		

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	yes	MUL.D	F0	F2	F4			false	false
Mult-2	no								
Add	yes	ADD.D	F6	F8	F2			false	false
Divide	yes	DIV.D	F10	F0	F6	Mult-1		false	true

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult-1			Add		Divide		

- executing addition (1 more cycle) and multiplication (still 4 more to go)

Scoreboard Example: Clock Cycle = 16

Instruction	Issue	Read Operands	Exec. Complete	Write Result
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	5	6	7	8
MUL.D F0, F2, F4	6	9		
SUB.D F8, F6, F2	7	9	11	12
DIV.D F10, F0, F6	8			
ADD.D F6, F8, F2	13	14	16	

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	yes	MUL.D	F0	F2	F4			false	false
Mult-2	no								
Add	yes	ADD.D	F6	F8	F2			false	false
Divide	yes	DIV.D	F10	F0	F6	Mult-1		false	true

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult-1			Add		Divide		

- addition execution is completed, but multiplication has still 3 more to go

Scoreboard Example: Clock Cycle = 17

Instruction	Issue	Read Operands	Exec. Complete	Write Result
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	5	6	7	8
MUL.D F0, F2, F4	6	9		
SUB.D F8, F6, F2	7	9	11	12
DIV.D F10, F0, F6	8			
ADD.D F6, F8, F2	13	14	16	

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	yes	MUL.D	F0	F2	F4			false	false
Mult-2	no								
Add	yes	ADD.D	F6	F8	F2			false	false
Divide	yes	DIV.D	F10	F0	F6	Mult-1		false	true

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult-1			Add		Divide		

- addition stalls (result is not written) to avoid WAR hazard on F6

Scoreboard Example: Clock Cycle = 18

Instruction	Issue	Read Operands	Exec. Complete	Write Result
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	5	6	7	8
MUL.D F0, F2, F4	6	9		
SUB.D F8, F6, F2	7	9	11	12
DIV.D F10, F0, F6	8			
ADD.D F6, F8, F2	13	14	16	

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	yes	MUL.D	F0	F2	F4			false	false
Mult-2	no								
Add	yes	ADD.D	F6	F8	F2			false	false
Divide	yes	DIV.D	F10	F0	F6	Mult-1		false	true

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult-1			Add		Divide		

- multiplication is completing, division waits for it, addition waits for division

Scoreboard Example: Clock Cycle = 19

Instruction	Issue	Read Operands	Exec. Complete	Write Result
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	5	6	7	8
MUL.D F0, F2, F4	6	9	19	
SUB.D F8, F6, F2	7	9	11	12
DIV.D F10, F0, F6	8			
ADD.D F6, F8, F2	13	14	16	

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	yes	MUL.D	F0	F2	F4			false	false
Mult-2	no								
Add	yes	ADD.D	F6	F8	F2			false	false
Divide	yes	DIV.D	F10	F0	F6	Mult-1		false	true

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult-1			Add		Divide		

- multiplication execution is finally completed

Scoreboard Example: Clock Cycle = 20

Instruction	Issue	Read Operands	Exec. Complete	Write Result
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	5	6	7	8
MUL.D F0, F2, F4	6	9	19	20
SUB.D F8, F6, F2	7	9	11	12
DIV.D F10, F0, F6	8			
ADD.D F6, F8, F2	13	14	16	

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	no								
Mult-2	no								
Add	yes	ADD.D	F6	F8	F2			false	false
Divide	yes	DIV.D	F10	F0	F6			true	true

	F0	F2	F4	F6	F8	F10	...	F30
FU				Add		Divide		

- multiplication has committed (writing F0)

Scoreboard Example: Clock Cycle = 21

Instruction	Issue	Read Operands	Exec. Complete	Write Result
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	5	6	7	8
MUL.D F0, F2, F4	6	9	19	20
SUB.D F8, F6, F2	7	9	11	12
DIV.D F10, F0, F6	8	21		
ADD.D F6, F8, F2	13	14	16	

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	no								
Mult-2	no								
Add	yes	ADD.D	F6	F8	F2			false	false
Divide	yes	DIV.D	F10	F0	F6			false	false

	F0	F2	F4	F6	F8	F10	...	F30
FU				Add		Divide		

- reading division operands

Scoreboard Example: Clock Cycle = 22

Instruction	Issue	Read Operands	Exec. Complete	Write Result
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	5	6	7	8
MUL.D F0, F2, F4	6	9	19	20
SUB.D F8, F6, F2	7	9	11	12
DIV.D F10, F0, F6	8	21		
ADD.D F6, F8, F2	13	14	16	22

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	no								
Mult-2	no								
Add	no								
Divide	yes	DIV.D	F10	F0	F6			false	false

	F0	F2	F4	F6	F8	F10	...	F30
FU						Divide		

- No more WAR hazards, addition can write result into F6

Scoreboard Example: Clock Cycle = 61

Instruction	Issue	Read Operands	Exec. Complete	Write Result
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	5	6	7	8
MUL.D F0, F2, F4	6	9	19	20
SUB.D F8, F6, F2	7	9	11	12
DIV.D F10, F0, F6	8	21	61	
ADD.D F6, F8, F2	13	14	16	22

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	no								
Mult-2	no								
Add	no								
Divide	yes	DIV.D	F10	F0	F6			false	false

	F0	F2	F4	F6	F8	F10	...	F30
FU						Divide		

- After 40 cycles division execution is completed

Scoreboard Example: Clock Cycle = 62

Instruction	Issue	Read Operands	Exec. Complete	Write Result
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	5	6	7	8
MUL.D F0, F2, F4	6	9	19	20
SUB.D F8, F6, F2	7	9	11	12
DIV.D F10, F0, F6	8	21	61	62
ADD.D F6, F8, F2	13	14	16	22

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	no								
Mult-2	no								
Add	no								
Divide	no								

	F0	F2	F4	F6	F8	F10	...	F30
FU								

- Division commits and the whole program is done

Scoreboard Example: Clock Cycle = 62

Instruction	Issue	Read Operands	Exec. Complete	Write Result
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	5	6	7	8
MUL.D F0, F2, F4	6	9	19	20
SUB.D F8, F6, F2	7	9	11	12
DIV.D F10, F0, F6	8	21	61	62
ADD.D F6, F8, F2	13	14	16	22

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	no								
Mult-2	no								
Add	no								
Divide	no								

	F0	F2	F4	F6	F8	F10	...	F30
FU								

- 62 cycles necessary (in-order issue; out-of-order execution & commit)

Scoreboard: Benefits & Costs

- The CDC 6600 designers measured
  - 1.7x performance improvement for FORTRAN programs
  - 2.5x performance improvement for assembly programs
    - but before breakthroughs in DRAM, SW scheduling, caches…
    - dynamic scheduling now popular with multiple-issue & speculation
- CDC 6600 scoreboard had about as much logic as one of the functional units
    - but four times as many buses as a simple in-order implementation
- In eliminating stalls, a scoreboard is limited by
    - # of scoreboard entries ≥ window size
    - amount of parallelism among the instructions
      - recall that CDC 6600 windows do not extend beyond a basic block
    - # and type of functional units (structural hazards)
    - # of buses ≥ # functional units proceeding in steps 2 & 4
    - presence of anti-dependences and output dependences

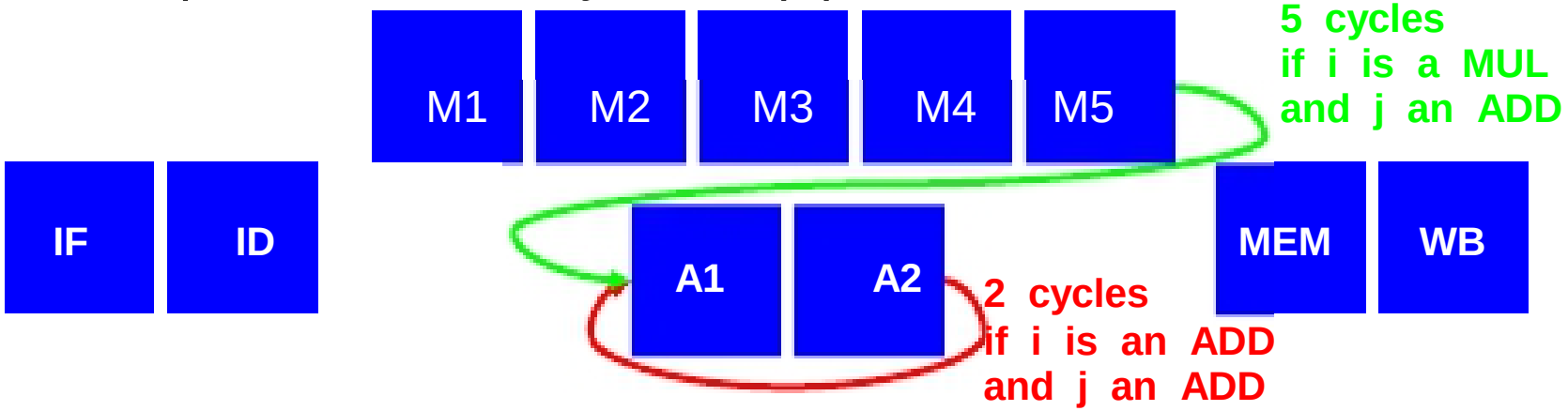
Compiler Technology to Improve ILP

- Loop Unrolling & Pipeline Static Scheduling
- Software Pipelining
- Compiler-Based Branch Prediction

- Applications
  - processors that use static issue
    - VLIW and VLIW-like
      - NXP Semiconductors (founded by Philips) Trimedia
      - Transmeta Crusoe
  - processors that combine dynamic scheduling and static scheduling
    - Itanium Processor Family (based on IA-64 ISA)
      - Explicit Parallel Instruction Computing (EPIC)

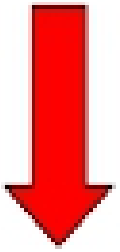
How to Keep a Pipeline Full?

- Exploit Parallelism “in Time”
  - find sequences of independent instructions that can be overlapped in the pipeline
- Scheduling problem for a compiler
  - given “source instruction” i and “dependent instruction” j
  - avoid pipeline stalls by…
  - …separating j from i by a distance in clock cycles that is equal to the latency of the pipeline execution of i



Example: a Simple Parallel Loop

```
for (i=1000; i>0; i=i-1) {
    x[i] = x[i] + s;
}
```



compiled into MIPS assembly language

```
loop:   L.D      F0, 0(R1)      ; R1 = highest address of array element
        ADD.D    F4, F0, F2     ; add scalar s (stored in F2)
        S.D      F4, 0(R1)      ; store result
        DADDUI   R1,R1,#-8      ; decrement pointer by 8 bytes (DW)
        BNE      R1,R2, loop    ; 8(R2) points to x[1], last to process
```

Assumptions for Following Examples: Latencies and Resources of MIPS Pipeline

instruction Producing result	instruction Consuming result	Latency in Clock cycles
FP ALU op	FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1
Load double	Store double	0
Integer op	Integer op	1

- Additional assumption: no structural hazards
  - functional units are fully pipelined (or replicated as many times as the pipeline depth) so that an operation of any type can be issued on every clock cycle

Example: Scheduling the Simple Parallel Loop on the MIPS Pipeline

```
loop:   L.D      F0, 0(R1)     ; R1 = highest address of array element
        ADD.D    F4, F0, F2    ; add scalar s (which is stored in F2)
        S.D      F4, 0(R1)     ; store result
        DADDUI   R1,R1,#-8     ; decrement pointer by 8 bytes (DW)
        BNE      R1,R2, loop; 8(R2) points to x[1], last to process
```

- unscheduled execution

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

- executionTime = 9 cycles

- scheduled execution

```
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
```

- executionTime = 7 cycles

- **NOTE: execution time is expressed in 'clock cycles per loop iteration '**

Scheduled Execution and Critical Path

- The clock cycle count for the loop is determined by the critical path, longest chain of dependent instructions
  - in the example, 6 clock cycles to execute sequentially the chain

```
L.D      F0, 0(R1)
stall
ADD.D    F4, F0, F2
stall
stall
S.D      F4, 8(R1)
```

- A good compiler (capable of some symbolic optimization) optimizes the "filling" of those stalls, for instance by altering and interchanging **S.D** with **DADDUI**

- scheduled execution

```
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
```

- executionTime = 7 cycles

Loop Overhead and Loop Unrolling

- At each iteration of the loop, the only actual work is done on the array

```
L.D     F0,  0(R1)
ADD.D   F4,  F0,  F2
S.D     F4,  8(R1)
```

- Beside the stall cycle, **DADDUI** and **BNE** represent loop overhead cycles
- To reduce loop overhead, before scheduling the loop execution apply loop unrolling
  1. replicate the loop body multiple times
  2. merge unnecessary instructions
  3. eliminate the name dependencies
  4. adjust the control code

• scheduled execution

```
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
```

• executionTime = 7 cycles

Reducing Loop Overhead by Loop Unrolling before Scheduling – Step 1 (replication)

• original loop

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

• unrolling the original loop 4 times (dropping 3 **BNE**)

```
loop:   L.D      F0,   0(R1)
        ADD.D    F4,   F0,  F2
        S.D      F4,   0(R1)
        DADDUI   R1,   R1,  #-8

        L.D      F0,   0(R1)
        ADD.D    F4,   F0,  F2

        S.D      F4,   0(R1)
        DADDUI   R1,   R1,  #-8
        L.D      F0,   0(R1)
        ADD.D    F4,   F0,  F2
        S.D      F4,   0(R1)
        DADDUI   R1,   R1,  #-8

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

Execution Time after Step 1

- unrolling the original loop 4 times (dropping 3 **BNE**)

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

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

- executionTime = 33 cycles

Reducing Loop Overhead by Loop Unrolling before Scheduling – Step 2 (merging)

• unrolling the original loop 4 times (dropping 3 **BNE**)

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

• merging **DADDUI** and adjusting the offsets

```
loop:  L.D     F0,  0(R1)
       ADD.D   F4,  F0,  F2
       S.D     F4,  0(R1)
       L.D     F0,  -8(R1)
       ADD.D   F4,  F0,  F2
       S.D     F4,  -8(R1)
       L.D     F0,  -16(R1)
       ADD.D   F4,  F0,  F2
       S.D     F4,  -16(R1)
       L.D     F0,  -24(R1)
       ADD.D   F4,  F0,  F2
       S.D     F4,  -24(R1)
       DADDUI  R1,  R1,  #-32
       BNE     R1,  R2,  loop
```

Execution Time after Step 2

- merging **DADDUI** and adjusting the offsets

```
loop:   L.D      F0,  0(R1)
        ADD.D    F4,  F0,  F2
        S.D      F4,  0(R1)
        L.D      F0,  -8(R1)
        ADD.D    F4,  F0,  F2
        S.D      F4,  -8(R1)
        L.D      F0,  -16(R1)
        ADD.D    F4,  F0,  F2
        S.D      F4,  -16(R1)
        L.D      F0,  -24(R1)
        ADD.D    F4,  F0,  F2
        S.D      F4,  -24(R1)
        DADDUI   R1,  R1,  #-32
        BNE      R1,  R2,  loop
```

```
loop:   L.D      F0,  0(R1)
        stall
        ADD.D    F4,  F0,  F2
        stall
        stall
        S.D      F4,  0(R1)
        L.D      F0,  -8(R1)
        stall
        ADD.D    F4,  F0,  F2
        stall
        stall
        S.D      F4,  -8(R1)
        L.D      F0,  -16(R1)
        stall
        ADD.D    F4,  F0,  F2
        stall
        stall
        S.D      F4,  -16(R1)
        L.D      F0,  -24(R1)
        stall
        ADD.D    F4,  F0,  F2
        stall
        stall
        S.D      F4,  -24(R1)
        DADDUI   R1,  R1,  #-32
        stall
        BNE      R1,  R2,  loop
```

- executionTime = 27 cycles

Reducing Loop Overhead by Loop Unrolling before Scheduling – Step 3 (renaming)

• merging **DADDUI** and adjusting the offsets

```
loop:   L.D      F0,   0(R1)
        ADD.D    F4,   F0,  F2
        S.D      F4,   0(R1)
        L.D      F0,   -8(R1)
        ADD.D    F4,   F0,  F2
        S.D      F4,   -8(R1)
        L.D      F0,   -16(R1)
        ADD.D    F4,   F0,  F2
        S.D      F4,   -16(R1)
        L.D      F0,   -24(R1)
        ADD.D    F4,   F0,  F2
        S.D      F4,   -24(R1)
        DADDUI   R1,   R1,  #-32
        BNE      R1,   R2,  loop
```

• eliminating name dependences via register renaming

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

Execution Time after Step 3

- eliminating name dependences via register renaming

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

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

- executionTime = 27 cycles

Reducing Loop Overhead by Loop Unrolling before Scheduling – Step 4 (scheduling)

- eliminating name dependences via register renaming

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

- 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
```

Scheduling after Step 4

- 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)
        BNE      R1,   R2,   loop
        S.D      F16,  8(R1)
```

```
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
```

- executionTime = 14 cycles

Final Performance Comparison

• original scheduled execution

```
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
```

• executionTime = 7cycles
• execution Time Per Element = 7 cycles
• e.g., an array of 1000 elements is processed in 7,000 cycles

• scheduled execution after loop unrolling

```
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
```

• executionTime = 14 cycles
• execution Time Per Element = 3.5 cycles
• e.g., an array of 1000 elements is processed in 3,500 cycles

Loop-unrolling speedup = 2

Loop Unrolling & Scheduling: Summary

- How is the final instruction sequence obtained?
- The compiler needs to determine that it is both possible and convenient to…
  - unroll the loop because the iterations are independent (except for the loop maintenance code)
  - eliminate extra loop maintenance code as long as the offsets are properly adjusted
  - use different registers to increase parallelism by removing name dependencies
  - remove stalling cycles by interleaving the instructions (and adjusting the offset or index increment consequently)
    - e.g. the compiler analyzes the memory addresses to determine that loads and stores from different iterations are independent