Computer Architecture: Dynamic Scheduling: Loop Unrolling and Scoreboard

Berk Sunar and Thomas Eisenbarth

ECE 505

WPI

Outline

- 5 stages of RISC
- Type of hazards
- Static and Dynamic Branch Prediction
- Pipelining with Exceptions
- Pipelining with Floating-Point Operations
- Correlating and Tournament Branch Prediction

- Loop Unrolling

- Dynamic Scheduling: Scoreboard: C.7
- Dynamic Scheduling: Tomasulo: 3.5, 3.6
- Hardware Based Speculation 3.7
- Multiple Issue

Loop Unrolling

- How can we implement this?

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

Direct Code:

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

3 useful instructions
2 control instructions

Loop Unrolling

- Assume:

  Load to Add : 1 stall

  FP Add to Store: 2 stalls

  Add to Branch: 1 stall

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

Can we do better?

```
Loop: L.D F0,0(R1)          1
      stall                 2
      ADD.D F4,F0,F2        3
      stall                 4
      stall                 5
      S.D F4,0(R1)          6
      DADDUI R1,R1,#-8      7
      stall                 8
      BNE R1,R2,Loop        9
```

1st try: 9 clocks per element
3 useful clocks, 2 control clocks, 4 stalls

Loop Unrolling

- 2nd try: schedule to minimize stalls

```
Loop: L.D F0,0(R1)          1
      stall                 2
      ADD.D F4,F0,F2        3
      Stall                 4
      stall                 5
      S.D F4,0(R1)          6
      DADDUI R1,R1,#-8      7
      stall                 8
      BNE R1,R2,Loop        9
```

```
Loop: L.D F0,0(R1)          1
      DADDUI R1,R1,#-8      2
      ADD.D F4,F0,F2        3
      Stall                 4
      stall                 5
      S.D F4,8(R1)          6
      BNE R1,R2,Loop        7
```

Can we do better?

2st try: 7 clocks per element
3 useful clocks, 2 control clocks, 2 stalls

Loop Unrolling

- 3rd try: Unroll to 4 elements per iteration
  - Rename registers
  - Correct loop increment

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

Can we do better?

```
Loop: L.D    F0,0(R1)      1  2
      ADD.D  F4,F0,F2      3  4  5
      S.D    F4,0(R1)      6
      L.D    F6,-8(R1)     7  8
      ADD.D  F8,F6,F2      9  10 11
      S.D    F8,-8(R1)     12
      L.D    F10,-16(R1)   13 14
      ADD.D  F12,F10,F2    15 16 17
      S.D    F12,-16(R1)   18
      L.D    F14,-24(R1)   19 20
      ADD.D  F16,F14,F2    21 22 23
      S.D    F16,-24(R1)   24
      DADDUI R1,R1,#-32    25 26
      BNE    R1,R2,Loop    27
```

3rd try: 27 clocks per 4 elements (avg = 6.75)
12 useful clocks, 2 control clocks, 13 stalls

Loop Unrolling

- 4th try: schedule to remove stalls

```
Loop: L.D    F0,0(R1)      1 2
      ADD.D  F4,F0,F2      3 4 5
      S.D    F4,0(R1)      6
      L.D    F6,-8(R1)     7 8
      ADD.D  F8,F6,F2      9 10 11
      S.D    F8,-8(R1)     12
      L.D    F10,-16(R1)   13 14
      ADD.D  F12,F10,F2    15 16 17
      S.D    F12,-16(R1)   18
      L.D    F14,-24(R1)   19 20
      ADD.D  F16,F14,F2    21 22 23
      S.D    F16,-24(R1)   24
      DADDUI R1,R1,#-32    25 26
      BNE    R1,R2,Loop    27
```

```
Loop: L.D    F0,0(R1)      1
      L.D    F6,-8(R1)     2
      L.D    F10,-16(R1)   3
      L.D    F14,-24(R1)   4
      ADD.D  F4,F0,F2      5
      ADD.D  F8,F6,F2      6
      ADD.D  F12,F10,F2    7
      DADDUI R1,R1,#-32    8
      ADD.D  F16,F14,F2    9
      S.D    F4,32(R1)     10
      S.D    F8,24(R1)     11
      S.D    F12,16(R1)    12
      S.D    F16,8(R1)     13
      BNE    R1,R2,Loop    14
```

4th try: 14 clocks per 4 elements (avg = 3.5)
12 useful clocks, 2 control clocks, 0 stalls

Can we do better?

Loop Unrolling

- How many elements per iteration?
  - More elements → less stalls

    Then, more elements → less control clocks
  - But, longer code (embedded devices)

    And more registers

Loop Unrolling

- Static or Dynamic scheduling?
  - Static Scheduling.

- Who should take this job?
  - Compiler. The compiler needs to do:

  1- Check if the elements are independent

  2- Register renaming

  3- Adjust addresses of loads and stores

  4- Adjust loop increments

  5- Schedule the code to remove stalls

Outline

- 5 stages of RISC
- Type of hazards
- Static and Dynamic Branch Prediction
- Pipelining with Exceptions
- Pipelining with Floating-Point Operations
- Correlating and Tournament Branch Prediction
- Loop Unrolling

- Dynamic Scheduling: Scoreboard: C.7

- Dynamic Scheduling: Tomasulo: 3.5, 3.6
- Hardware Based Speculation 3.7
- Multiple Issue

Dynamic Scheduling: Idea

Idea: Hardware rearranges execution flow to reduce stalls

- Must ensure same data flow and exception behavior
  - Better if: dependencies unknown at compile time
  - Simplifies compiler
  - Allows handling of unpredictable delays
  - Disadvantage: Increase in hardware complexity

Dynamic Scheduling: Why?

Example:

```
1       DIV.D   F0,F2,F4
2       ADD.D   F4,F0,F2
3       SUB.D   F12,F8,F14
```

What is thee problem here? (Hazards?)

- 2 depends on 1 → 2 stalls pipeline
- 3 is not data-dependent on pipeline

➔ Violating program order can result in speed-up

Issue process becomes two-part:

- Checking for structural hazards
- Waiting for absence of data hazards

Result: *out-of-order execution* and hence *out-of-order completion*

Enabling Out-of Order Execution

ID pipeline stage is split:

1. Issue: decode instruction and check structural hazards (in-order issue)
2. Read operands: wait for absence of data hazards, then read operands

Preceded by instruction fetch (IF)

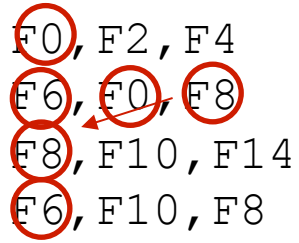
Followed by instruction execution (EX):

- Scheduling makes sense only if multiple instructions can execute in parallel

Hazards in Dynamic Scheduling

Example 2:

```
1       DIV.D   F0,F2,F4
2       ADD.D   F6,F0,F8
3       SUB.D   F8,F10,F14
4       MUL.D   F6,F10,F8
```



Data Dependence
Antidependence
Output Dependence

- 2 depends on 1: **RAW hazard**
- Antidependence between 3 and 2:
  - Early execution of 3 gives **WAR hazard**
- 2 and 4 have output dependence: potential **WAW hazard**

➢ **Scoreboarding** handles these hazards by stalling the later instruction

Scoreboarding

- Allows out-of-order execution when sufficient resources and no data dependencies exist
- Tries to keep 1 IPC when no structural hazard happens, while executing each instruction as early as possible
- Multiple instructions may and will be in EX stage simultaneously→ multiple functional units
- Scoreboard controls instruction execution and keeps track of instruction, register, and functional unit status

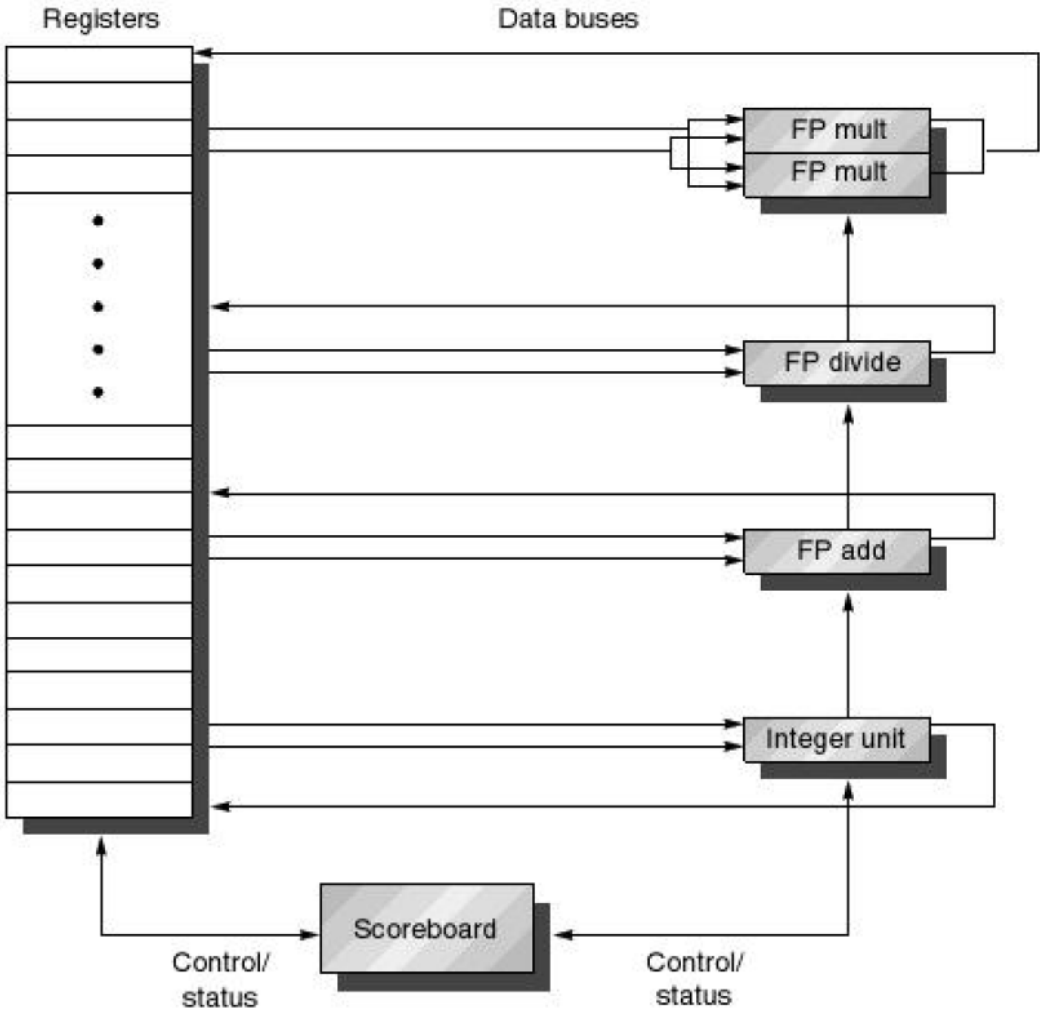
Scoreboarding: Example

Functional units:

- 2 FP multipliers
- 1 FP adder
- 1FP divider
- 1 integer unit

Scoreboard :

- Stores instructions
- tracks data dependencies
- Decides when to execute instruction
- Controls write of result to register



Scoreboard Instruction Execution

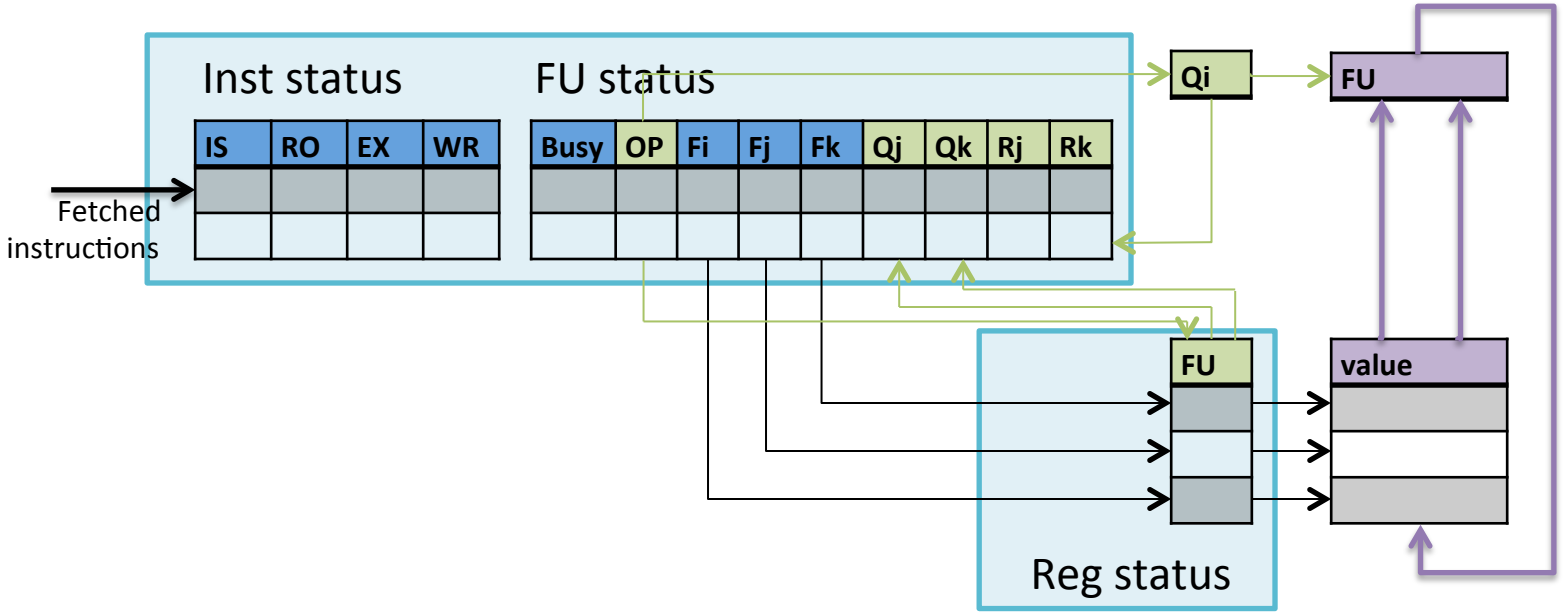
Four steps replacing the **ID, EX and WB** steps:

1. Issue: instruction issued if functional unit is free, no other active instruction has same target register (avoids WAW hazard)
2. Read operands: source operand available if no earlier instruction plans overwriting it (avoids RAW hazards). Once all operands are available, the instruction is executed.
3. Execution: functional unit executes instruction. Notifies Scoreboard when result is ready.
4. Write result: Scoreboard checks that there is no preceding instruction that has not read the target register yet (avoids WAR hazard).

Scoreboard Components

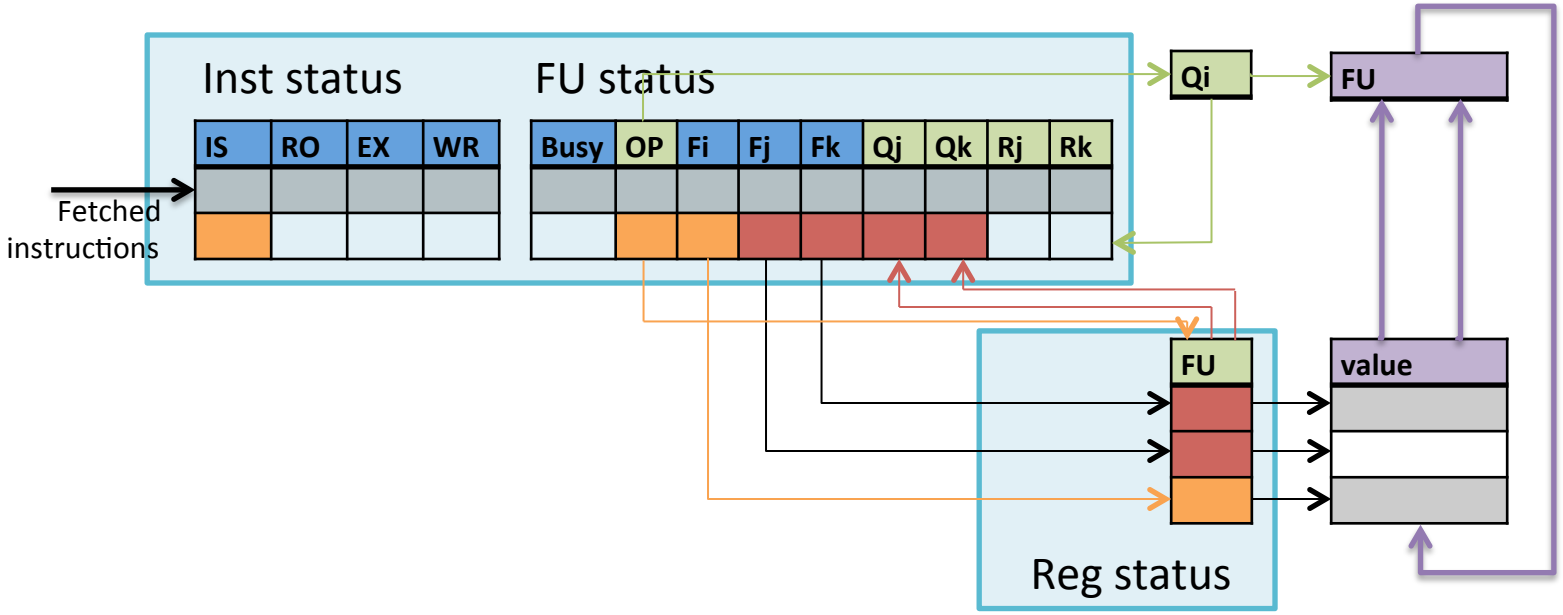
- Instruction Status: indicates which stage the instruction is in. 1 entry per instruction
- Functional unit Status: status for each FU:
  - Busy: FU is busy
  - Op: current operation (e.g. add or subtract)
  - Fi, Fj, Fk: destination and source register numbers
  - Qj,Qk: FUs producing Qj and Qk
  - Rj,Rk: Flags indicating when Fj,Fk are ready and not yet read
- Register Result Status: indicates which FU will write each register. Blank if no active instruction targets register

Simple Scoreboard



- Instruction fields and status bits
- Tags
- Values

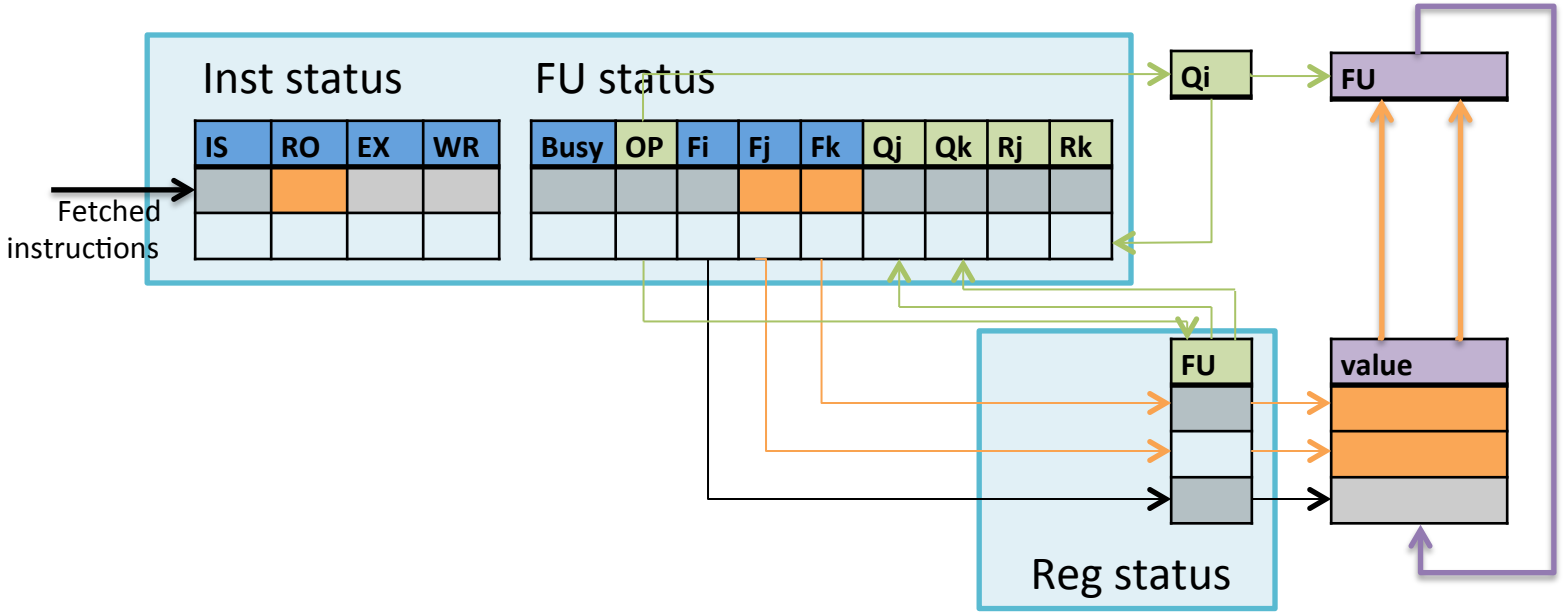
Scoreboard: Issue



stall for WAW and structural hazards, but otherwise:

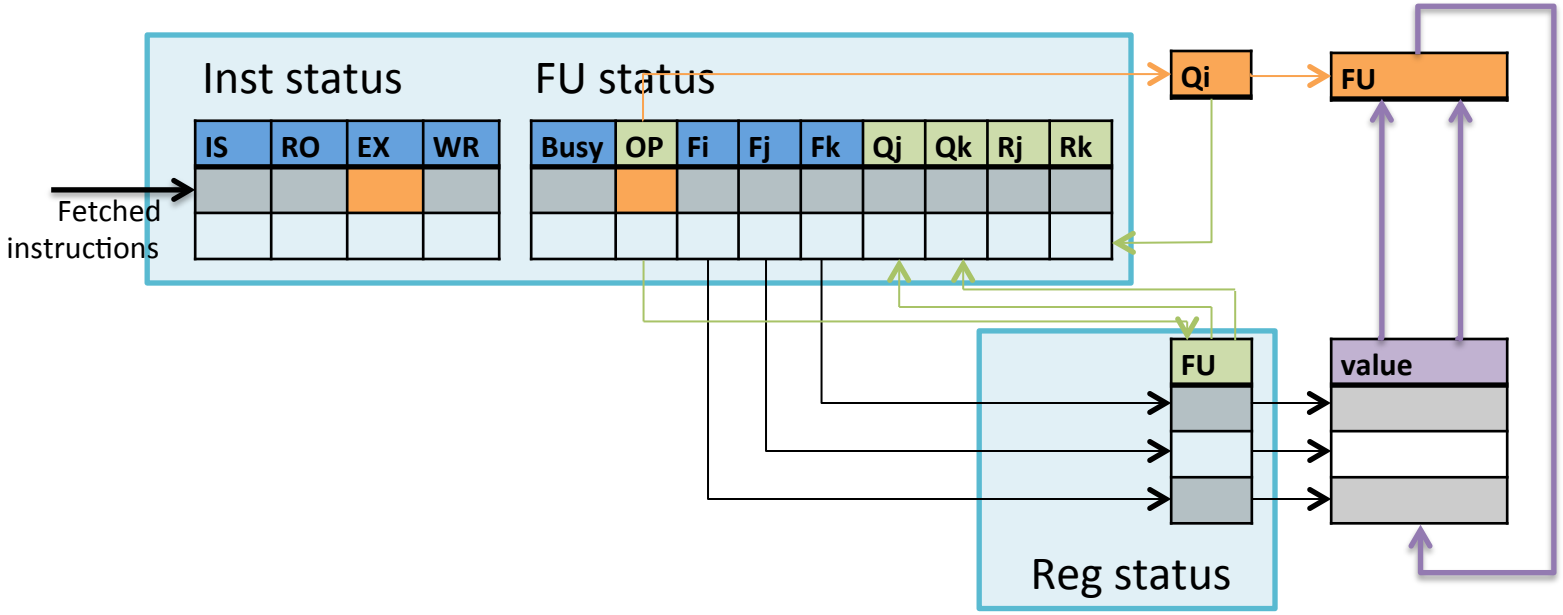
- allocate scoreboard entry
- copy status for input registers
- set status for output register

Scoreboard: Read Operands



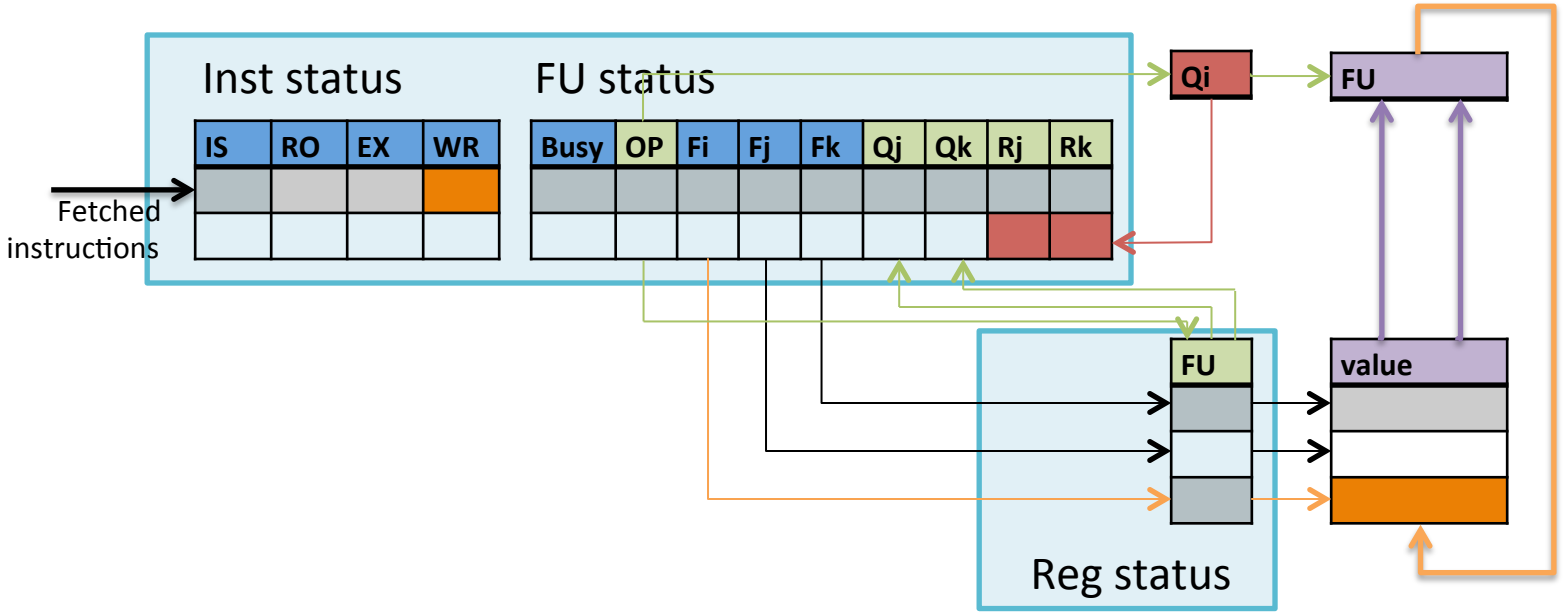
- wait for RAW hazards (Fi or Fj not ready), but otherwise: read registers

Scoreboard: Execute



- Execute until finished

Scoreboard: Write Result



wait for WAR hazards, but otherwise:

- write result
- compare tags with waiting instructions
- on match: clear tag (set input to "ready")

Scoreboard Data Structure

Instruction Status

Instruction	Issue	Read ops	Execute complete	Write result
L.D F6, 32(R2)				
L.D F2, 44(R3)				
MUL.D F0, F2, F4				
SUB.D F8, F6, F2				
DIV.D F10,F0, F6				
ADD.D F6, F8, F2				

Register Status

Register	FU
F0	
F2	
F4	
F6	
F8	
F10	

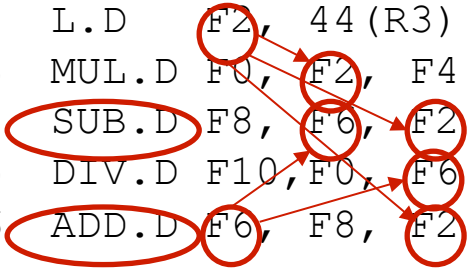
Functional Unit Status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Int									
Mul1									
Mul2									
Add									
Div									

Scoreboard Data Structure: Example

Example Program:

```
1  L.D    F6, 32(R2)
2  L.D    F2, 44(R3)
3  MUL.D  F0, F2, F4
4  SUB.D  F8, F6, F2
5  DIV.D  F10,F0, F6
6  ADD.D  F6, F8, F2
```



Dependence

Antidependence

- EX latencies: Add: 2cc, Mul: 10cc, Div: 40cc
- Which hazards exist?
  - RAW hazards: from 2 to 3, 4, and 6, from 3 to 5, and from 4 to 6
  - WAR hazards: from 4 to 6, and 5 to 6
  - Functional hazard: on Add FU for 4 and 6

Scoreboard Example: Initial State

Instruction Status

Instruction	Issue	Read ops	Execute complete	Write result
`L.D F6, 32(R2)`	ok	ok	ok	ok
`L.D F2, 44(R3)`	ok	ok	ok	
`MUL.D F0, F2, F4`	ok			
`SUB.D F8, F6, F2`	ok			
`DIV.D F10,F0, F6`	ok			
`ADD.D F6, F8, F2`				

Register Status

Register	FU
F0	Mul1
F2	Int
F4	
F6	
F8	Add
F10	Div

Functional Unit Status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Int	Yes	Load	F2	F3				No	
Mul1	Yes	MUL	F0	F2	F4	Int		No	Yes
Mul2	No								
Add	Yes	SUB	F8	F6	F2		Int	Yes	No
Div	Yes	DIV	F10	F0	F6	Mul1		No	Yes

Scoreboard Example: Cycle 1

Instruction Status

Instruction	Issue	Read ops	Execute complete	Write result
L.D F6, 32(R2)	ok	ok	ok	ok
L.D F2, 44(R3)	ok	ok	ok	**cc1**
MUL.D F0, F2, F4	ok			
SUB.D F8, F6, F2	ok			
DIV.D F10,F0, F6	ok			
ADD.D F6, F8, F2				

Register Status

Register	FU
F0	Mul1
F2	~~Int~~
F4	
F6	
F8	Add
F10	Div

Stalled on functional hazard

Functional Unit Status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Int	**No**	~~Load~~	~~F2~~	~~F3~~				~~No~~	
Mul1	Yes	MUL	F0	F2	F4	~~Int~~		**Yes**	Yes
Mul2	No								
Add	Yes	SUB	F8	F6	F2		~~Int~~	Yes	**Yes**
Div	Yes	DIV	F10	F0	F6	Mul1		No	Yes

Scoreboard Example: Cycle 2

Instruction Status

Instruction	Issue	Read ops	Execute complete	Write result
L.D F6, 32(R2)	ok	ok	ok	ok
L.D F2, 44(R3)	ok	ok	ok	cc1
MUL.D F0, F2, F4	ok	**cc2**		
SUB.D F8, F6, F2	ok	**cc2**		
DIV.D F10,F0, F6	ok			
ADD.D F6, F8, F2				

Register Status

Register	FU
F0	Mul1
F2	
F4	
F6	
F8	Add
F10	Div

Functional Unit Status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Int	No								
Mul1	Yes	MUL	F0	F2	F4			~~Yes~~	~~Yes~~
Mul2	No								
Add	Yes	SUB	F8	F6	F2			~~Yes~~	~~Yes~~
Div	Yes	DIV	F10	F0	F6	Mul1		No	Yes

Scoreboard Example: Cycle 3

Instruction Status

Instruction	Issue	Read ops	Execute complete	Write result
L.D F6, 32(R2)	ok	ok	ok	ok
L.D F2, 44(R3)	ok	ok	ok	cc1
MUL.D F0, F2, F4	ok	cc2	**cc3+**	
SUB.D F8, F6, F2	ok	cc2	**cc3+**	
DIV.D F10,F0, F6	ok			
ADD.D F6, F8, F2				

Register Status

Register	FU
F0	Mul1
F2	
F4	
F6	
F8	Add
F10	Div

Functional Unit Status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Int	No								
Mul1	Yes	MUL	F0	F2	F4				
Mul2	No								
Add	Yes	SUB	F8	F6	F2				
Div	Yes	DIV	F10	F0	F6	Mul1		No	Yes

Scoreboard Example: Cycle 4

Instruction Status

Instruction	Issue	Read ops	Execute (complete)	Write result
`L.D F6, 32(R2)`	ok	ok	ok	ok
`L.D F2, 44(R3)`	ok	ok	ok	cc1
`MUL.D F0, F2, F4`	ok	cc2	cc3+	
`SUB.D F8, F6, F2`	ok	cc2	cc3 (**cc4**)	
`DIV.D F10,F0, F6`	ok			
`ADD.D F6, F8, F2`				

Register Status

Register	FU
F0	Mul1
F2	
F4	
F6	
F8	Add
F10	Div

Functional Unit Status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Int	No								
Mul1	Yes	MUL	F0	F2	F4			No	No
Mul2	No								
Add	~~Yes~~ No	~~SUB~~	~~F8~~	~~F6~~	~~F2~~			~~No~~	~~No~~
Div	Yes	DIV	F10	F0	F6	Mul1		No	Yes

Scoreboard Example: Cycle 5

Instruction Status

Instruction	Issue	Read ops	Execute (complete)	Write result
L.D F6, 32(R2)	ok	ok	ok	ok
L.D F2, 44(R3)	ok	ok	ok	cc1
MUL.D F0, F2, F4	ok	cc2	cc3+(cc12)	
SUB.D F8, F6, F2	ok	cc2	cc3 (cc4)	**cc5**
DIV.D F10,F0, F6	ok			
ADD.D F6, F8, F2	**cc5**			

Register Status

Register	FU
F0	Mul1
F2	
F4	
F6	Add
F8	~~Add~~
F10	Div

Functional Unit Status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Int	No								
Mul1	Yes	MUL	F0	F2	F4			No	No
Mul2	No								
Add	~~No~~ Yes	ADD	F6	F8	F2			Yes	Yes
Div	Yes	DIV	F10	F0	F6	Mul1		No	Yes

Allocate Add

Scoreboard Example: Cycle 6

Instruction Status

Instruction	Issue	Read ops	Execute (complete)	Write result
L.D F6, 32(R2)	ok	ok	ok	ok
L.D F2, 44(R3)	ok	ok	ok	cc1
MUL.D F0, F2, F4	ok	cc2	cc3+(cc12)	
SUB.D F8, F6, F2	ok	cc2	cc3 (cc4)	cc5
DIV.D F10,F0, F6	ok			
ADD.D F6, F8, F2	cc5	**cc6**		

Register Status

Register	FU
F0	Mul1
F2	
F4	
F6	Add
F8	
F10	Div

Functional Unit Status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Int	No								
Mul1	Yes	MUL	F0	F2	F4			No	No
Mul2	No								
Add	Yes	ADD	F6	F8	F2			~~Yes~~ No	~~Yes~~ No
Div	Yes	DIV	F10	F0	F6	Mul1		No	Yes

Scoreboard Example: Cycle 7

Instruction Status

Instruction	Issue	Read ops	Execute (complete)	Write result
L.D F6, 32(R2)	ok	ok	ok	ok
L.D F2, 44(R3)	ok	ok	ok	cc1
MUL.D F0, F2, F4	ok	cc2	cc3+(cc12)	
SUB.D F8, F6, F2	ok	cc2	cc3 (cc4)	cc5
DIV.D F10,F0, F6	ok			
ADD.D F6, F8, F2	cc5	cc6	**Cc7+(cc8)**	

Register Status

Register	FU
F0	Mul1
F2	
F4	
F6	Add
F8	
F10	Div

Functional Unit Status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Int	No								
Mul1	Yes	MUL	F0	F2	F4			No	No
Mul2	No								
Add	Yes	ADD	F6	F8	F2			No	No
Div	Yes	DIV	F10	F0	F6	Mul1		No	Yes

Scoreboard Example: Cycle 8

Instruction Status

Instruction	Issue	Read ops	Execute (complete)	Write result
L.D F6, 32(R2)	ok	ok	ok	ok
L.D F2, 44(R3)	ok	ok	ok	cc1
MUL.D F0, F2, F4	ok	cc2	cc3+(cc12)	
SUB.D F8, F6, F2	ok	cc2	cc3 (cc4)	cc5
DIV.D F10,F0, F6	ok			
ADD.D F6, F8, F2	cc5	cc6	cc7+**(cc8)**	

Register Status

Register	FU
F0	Mul1
F2	
F4	
F6	Add
F8	
F10	Div

Stalled on WAR

Functional Unit Status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Int	No								
Mul1	Yes	MUL	F0	F2	F4			No	No
Mul2	No								
Add	~~Yes~~ No	~~ADD~~	~~F6~~	~~F8~~	~~F2~~			~~No~~	~~No~~
Div	Yes	DIV	F10	F0	F6	Mul1		No	Yes

Scoreboard Example: Cycle 12

Instruction Status

Instruction	Issue	Read ops	Execute (complete)	Write result
L.D F6, 32(R2)	ok	ok	ok	ok
L.D F2, 44(R3)	ok	ok	ok	cc1
MUL.D F0, F2, F4	ok	cc2	cc3+(**cc12**)	
SUB.D F8, F6, F2	ok	cc2	cc3 (cc4)	cc5
DIV.D F10,F0, F6	ok			
ADD.D F6, F8, F2	cc5	cc6	cc7+(cc8)	

Register Status

Register	FU
F0	Mul1
F2	
F4	
F6	Add
F8	
F10	Div

Functional Unit Status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Int	No								
Mul1	~~Yes~~ No	~~MUL~~	~~F0~~	~~F2~~	~~F4~~			~~No~~	~~No~~
Mul2	No								
Add	~~Yes~~ No	~~ADD~~	~~F6~~	~~F8~~	~~F2~~			~~No~~	~~No~~
Div	Yes	DIV	F10	F0	F6	Mul1		No	Yes

Scoreboard Example: Cycle 13

Instruction Status

Instruction	Issue	Read ops	Execute (complete)	Write result
L.D F6, 32(R2)	ok	ok	ok	ok
L.D F2, 44(R3)	ok	ok	ok	cc1
MUL.D F0, F2, F4	ok	cc2	cc3+(cc12)	**cc13**
SUB.D F8, F6, F2	ok	cc2	cc3 (cc4)	cc5
DIV.D F10,F0, F6	ok			
ADD.D F6, F8, F2	cc5	cc6	cc7+(cc8)	

Register Status

Register	FU
F0	~~Mul1~~
F2	
F4	
F6	Add
F8	
F10	Div

Functional Unit Status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Int	No								
Mul1	~~Yes~~ No	~~MUL~~	~~F0~~	~~F2~~	~~F4~~			~~No~~	~~No~~
Mul2	No								
Add	~~Yes~~ No	~~ADD~~	~~F6~~	~~F8~~	~~F2~~			~~No~~	~~No~~
Div	Yes	DIV	F10	F0	F6	~~Mul1~~		~~No~~ Yes	Yes

Scoreboard Example: Cycle 14

Instruction Status

Instruction	Issue	Read ops	Execute (complete)	Write result
L.D F6, 32(R2)	ok	ok	ok	ok
L.D F2, 44(R3)	ok	ok	ok	cc1
MUL.D F0, F2, F4	ok	cc2	cc3+(cc12)	cc13
SUB.D F8, F6, F2	ok	cc2	cc3 (cc4)	cc5
DIV.D F10,F0, F6	ok	**cc14**		
ADD.D F6, F8, F2	cc5	cc6	cc7+(cc8)	

Register Status

Register	FU
F0	
F2	
F4	
F6	Add
F8	
F10	Div

Functional Unit Status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Int	No								
Mul1	No								
Mul2	No								
Add	~~Yes~~ No	~~ADD~~	~~F6~~	~~F8~~	~~F2~~			~~No~~	~~No~~
Div	Yes	DIV	F10	F0	F6			~~Yes~~ No	~~Yes~~ No

Scoreboard Example: Cycle 15

Instruction Status

Instruction	Issue	Read ops	Execute (complete)	Write result
L.D F6, 32(R2)	ok	ok	ok	ok
L.D F2, 44(R3)	ok	ok	ok	cc1
MUL.D F0, F2, F4	ok	cc2	cc3+(cc12)	cc13
SUB.D F8, F6, F2	ok	cc2	cc3 (cc4)	cc5
DIV.D F10,F0, F6	ok	cc14	**cc15+**	
ADD.D F6, F8, F2	cc5	cc6	cc7+(cc8)	**cc15**

Register Status

Register	FU
F0	
F2	
F4	
F6	~~Add~~
F8	
F10	Div

Functional Unit Status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Int	No								
Mul1	No								
Mul2	No								
Add	No								
Div	Yes	DIV	F10	F0	F6			No	No

Scoreboard: Summary

+ Cheap in hardware
+ pretty good performance
– no bypassing
  – RAW dependencies handled through registers
– limited scheduling scope
  – WAW/structural hazards force in-order dispatch
  – WAR hazards delay writeback and issue of dependent operations

Problems can be solved with register renaming!