

Lecture 3: Introduction to Advanced Pipelining

Professor David A. Patterson
Modifié et corrigé par E.M. Aboulhamid

DAP.F96 1

Review: Evaluating Branch Alternatives

- Two part solution:
 - Determine branch taken or not sooner, AND
 - Compute taken branch address earlier

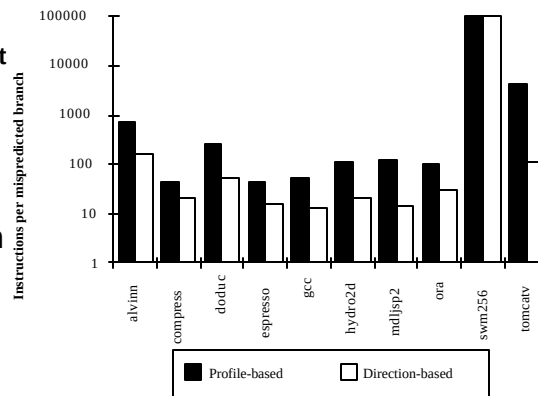
$$\text{Pipeline speedup} = \frac{\text{Pipeline depth}}{1 + \text{Branch frequency} \times \text{Branch penalty}}$$

<i>Scheduling scheme</i>	<i>Branch penalty</i>	<i>CPI</i>	<i>speedup v. unpipelined</i>	<i>speedup v. stall</i>
Stall pipeline	3	1.42	3.5	1.0
Predict taken	1	1.14	4.4	1.26
Predict not taken	1	1.09	4.5	1.29
Delayed branch	0.5	1.07	4.6	1.31

DAP.F96 2

Review: Evaluating Branch Prediction

- Two strategies
 - Backward branch predict taken, forward branch not taken
 - Profile-based prediction: record branch behavior, predict branch based on prior run
- “Instructions between mispredicted branches” a better metric than misprediction



DAP.F96 3

Review: Summary of Pipelining Basics

- Hazards limit performance
 - Structural: need more HW resources
 - Data: need forwarding, compiler scheduling
 - Control: early evaluation & PC, delayed branch, prediction
- Increasing length of pipe increases impact of hazards; pipelining helps instruction bandwidth, not latency
- Interrupts, Instruction Set, FP makes pipelining harder
- Compilers reduce cost of data and control hazards
 - Load delay slots
 - Branch delay slots
 - Branch prediction
- Today: Longer pipelines (R4000) => Better branch prediction, more instruction parallelism?

DAP.F96 4

Opérations multi-cycles

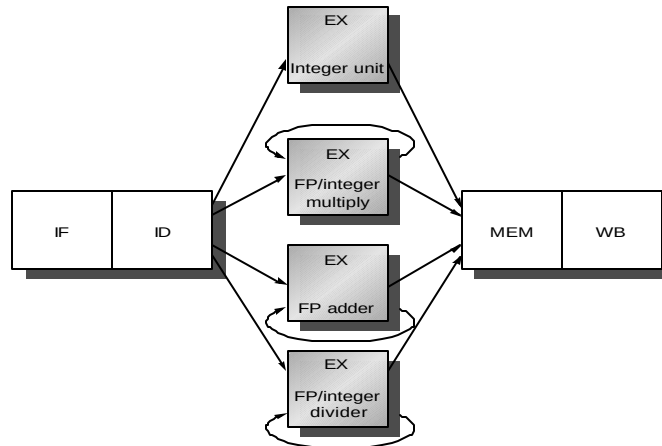


FIGURE 3.42 The DLX pipeline with three additional unpipelined, floating-point, functional units.

DAP.F96 5

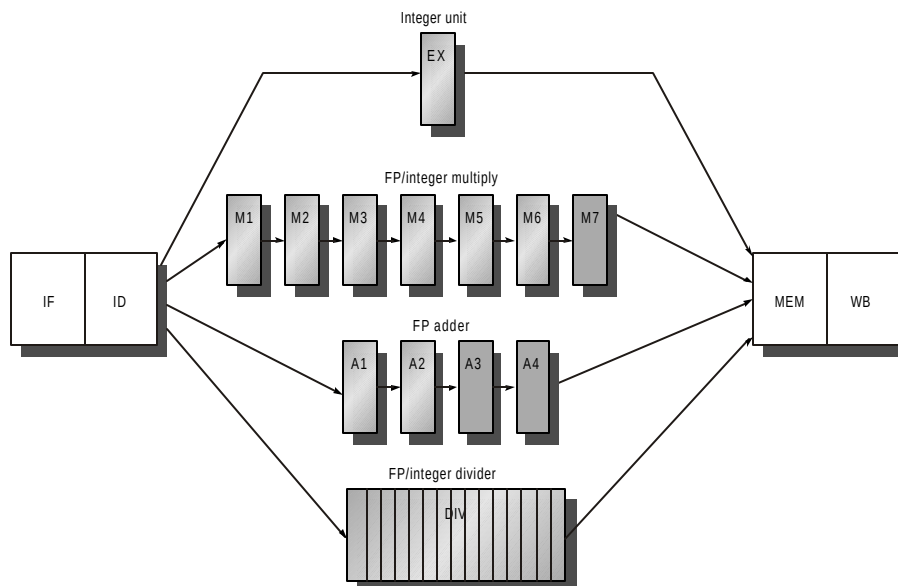


FIGURE 3.44 A pipeline that supports multiple outstanding FP operations.

DAP.F96 6

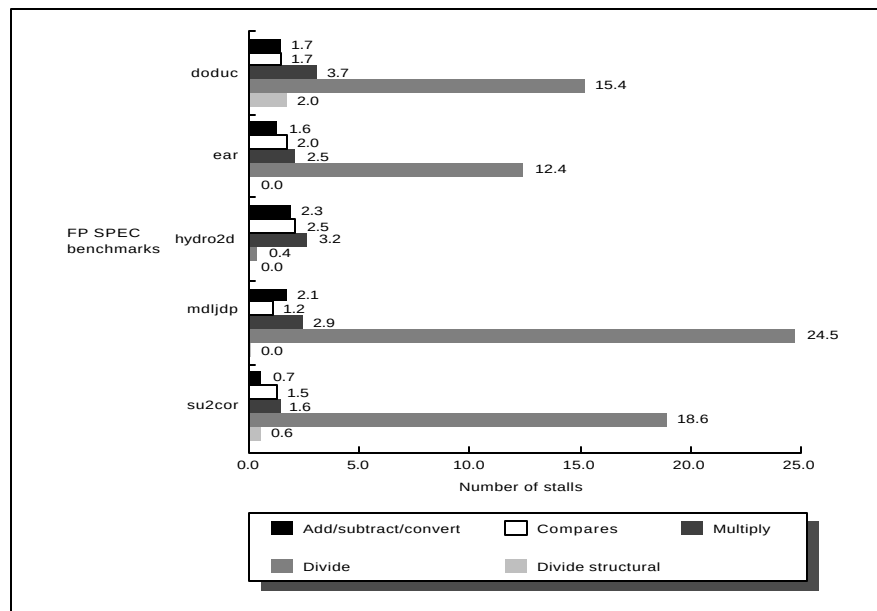


FIGURE 3.48 Stalls per FP operation for each major type of FP operation.

DAP.F96 7

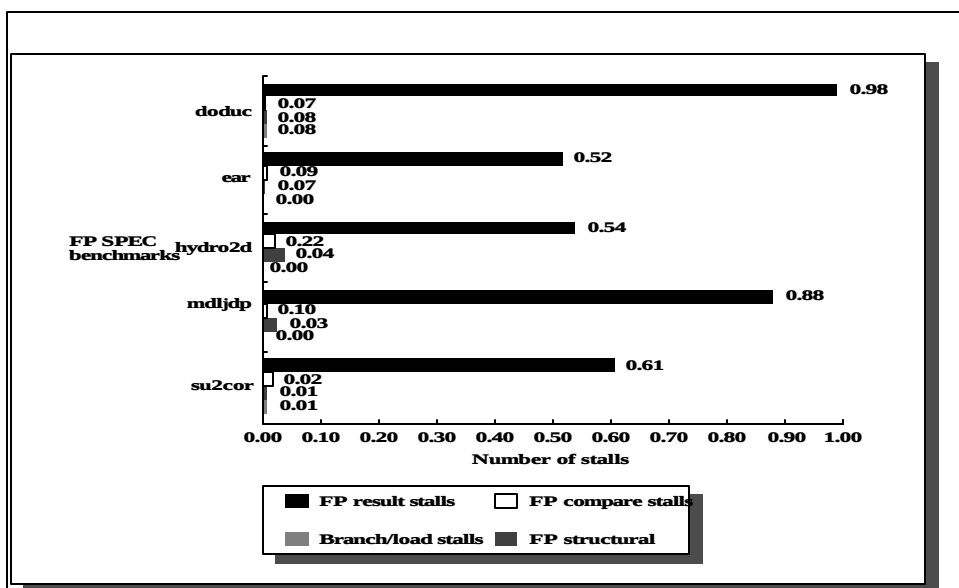


FIGURE 3.49 The stalls occurring for the DLX FP pipeline for the five FP SPEC benchmarks.

DAP.F96 8

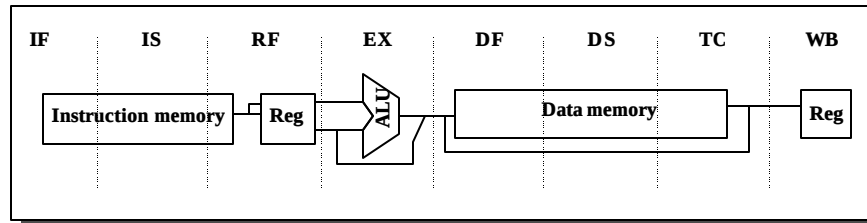


FIGURE 3.50 The eight-stage pipeline structure of the R4000 uses pipelined instruction and data caches.

DAP.F96 9

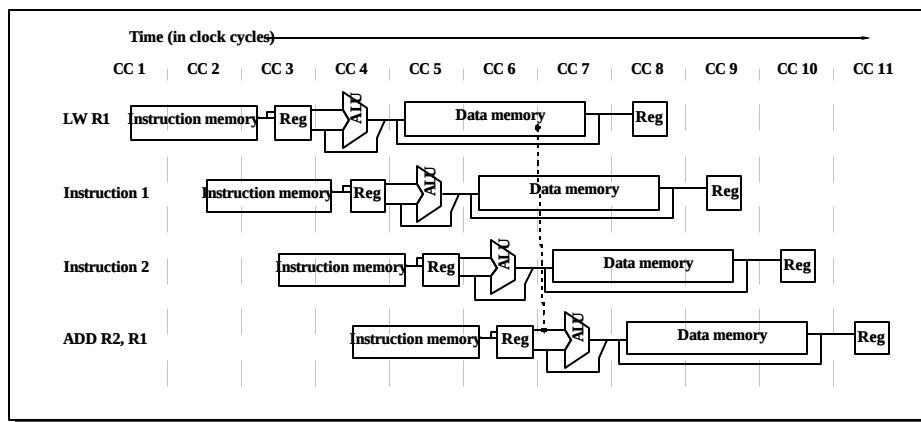
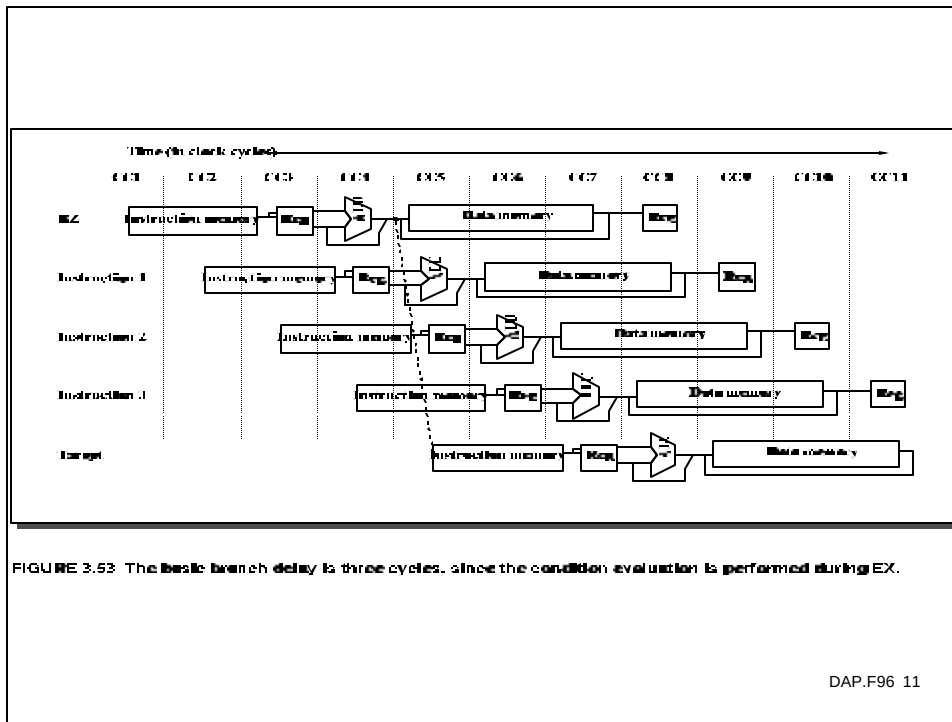


FIGURE 3.51 The structure of the R4000 integer pipeline leads to a two-cycle load delay.

DAP.F96 10

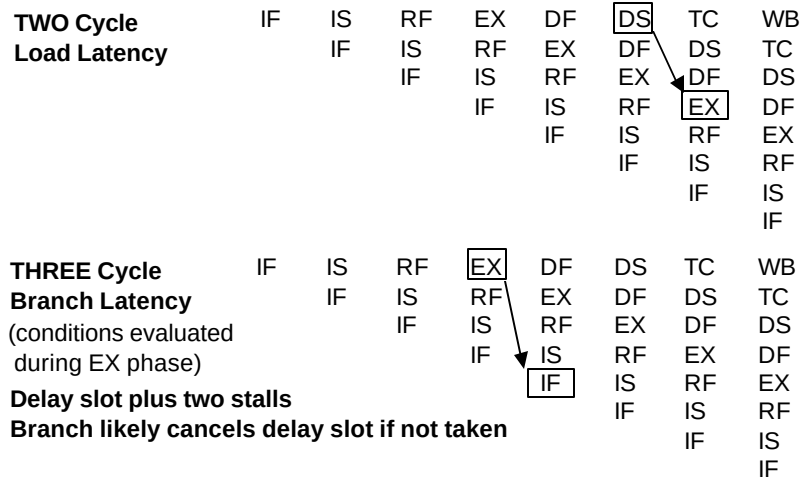


Case Study: MIPS R4000 (200 MHz)

- **8 Stage Pipeline:**
 - IF—first half of fetching of instruction; PC selection happens here as well as initiation of instruction cache access.
 - IS—second half of access to instruction cache.
 - RF—instruction decode and register fetch, hazard checking and also instruction cache hit detection.
 - EX—execution, which includes effective address calculation, ALU operation, and branch target computation and condition evaluation.
 - DF—data fetch, first half of access to data cache.
 - DS—second half of access to data cache.
 - TC—tag check, determine whether the data cache access hit.
 - WB—write back for loads and register-register operations.
- **8 Stages: What is impact on Load delay? Branch delay? Why?**

DAP.F96 12

Case Study: MIPS R4000



DAP.F96 13

MIPS R4000 Floating Point

- FP Adder, FP Multiplier, FP Divider
- Last step of FP Multiplier/Divider uses FP Adder HW
- 8 kinds of stages in FP units:

Stage	Functional unit	Description
A	FP adder	Mantissa ADD stage
D	FP divider	Divide pipeline stage
E	FP multiplier	Exception test stage
M	FP multiplier	First stage of multiplier
N	FP multiplier	Second stage of multiplier
R	FP adder	Rounding stage
S	FP adder	Operand shift stage
U		Unpack FP numbers

DAP.F96 14

MIPS FP Pipe Stages

FP Instr	1	2	3	4	5	6	7	8	...
Add, Subtract	U	S+A	A+R	R+S					
Multiply	U	E+M	M	M	M	N	N+A	R	
Divide	U	A	R	D ²⁸	...	D+A	D+R, D+R, D+A, D+R, A, R		
Square root	U	E	(A+R) ¹⁰⁸	...	A	R			
Negate	U	S							
Absolute value	U	S							
FP compare	U	A	R						

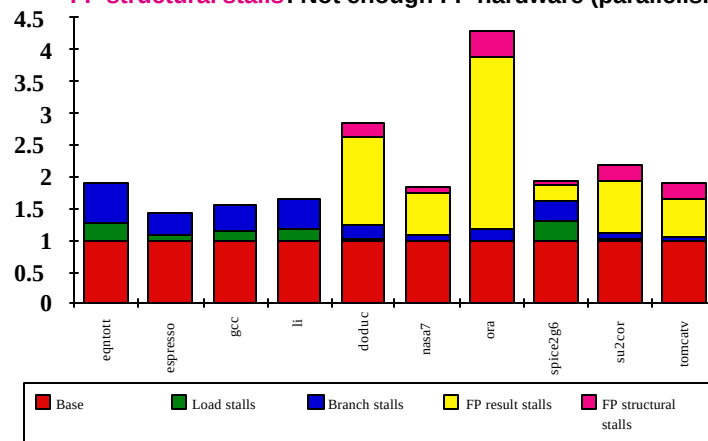
Stages:

<i>M</i>	<i>First stage of multiplier</i>	<i>A</i>	<i>Mantissa ADD stage</i>
<i>N</i>	<i>Second stage of multiplier</i>	<i>D</i>	<i>Divide pipeline stage</i>
<i>R</i>	<i>Rounding stage</i>	<i>E</i>	<i>Exception test stage</i>
<i>S</i>	<i>Operand shift stage</i>		
<i>U</i>	<i>Unpack FP numbers</i>		

DAP.F96 15

R4000 Performance

- Not ideal CPI of 1:
 - Load stalls (1 or 2 clock cycles)
 - Branch stalls (2 cycles + unfilled slots)
 - FP result stalls: RAW data hazard (latency)
 - FP structural stalls: Not enough FP hardware (parallelism)



DAP.F96 16

Advanced Pipelining and Instruction Level Parallelism (ILP)

- ILP: Overlap execution of unrelated instructions
- gcc 17% control transfer
 - 5 instructions + 1 branch
 - Beyond single block to get more instruction level parallelism
- Loop level parallelism one opportunity, SW and HW
- Do examples and then explain nomenclature
- DLX Floating Point as example
 - Measurements suggests R4000 performance FP execution has room for improvement

DAP.F96 17

FP Loop: Where are the Hazards?

```

Loop:  LD    F0,0(R1)    ;F0=vector element
        ADDD F4,F0,F2    ;add scalar from F2
        SD    0(R1),F4    ;store result
        SUBI  R1,R1,8    ;decrement pointer 8B (DW)
        BNEZ  R1,Loop    ;branch R1!=zero
        NOP              ;delayed branch slot
  
```

<i>Instruction producing result</i>	<i>Instruction using result</i>	<i>Latency in clock cycles</i>
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
Integer op	Integer op	0

- **Where are the stalls?**

DAP.F96 18

FP Loop Hazards

```

Loop:  LD    F0,0(R1)    ;F0=vector element
        ADDD  F4,F0,F2    ;add scalar in F2
        SD    0(R1),F4    ;store result
        SUBI  R1,R1,8     ;decrement pointer 8B (DW)
        BNEZ  R1,Loop    ;branch R1!=zero
        NOP                ;delayed branch slot
    
```

<i>Instruction producing result</i>	<i>Instruction using result</i>	<i>Latency in clock cycles</i>
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
Integer op	Integer op	0

DAP.F96 19

FP Loop Showing Stalls

```

1 Loop:  LD    F0,0(R1)    ;F0=vector element
2        stall
3        ADDD  F4,F0,F2    ;add scalar in F2
4        stall
5        stall
6 SD    0(R1),F4    ;store result
7 SUBI  R1,R1,8     ;decrement pointer 8B (DW)
8        stall
9 BNEZ  R1,Loop    ;branch R1!=zero
10       stall        ;delayed branch slot
    
```

<i>Instruction producing result</i>	<i>Instruction using result</i>	<i>Latency in clock cycles</i>
FP ALU op	Another FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1

- 10 clocks: Rewrite code to minimize stalls?

DAP.F96 20

Revised FP Loop Minimizing Stalls

```

1 Loop: LD    F0,0(R1)
4      SUBI   R1,R1,8
3      ADDDD  F4,F0,F2
2      stall
5      BNEZ   R1,Loop    ;delayed branch
6 SD     8(R1),F4        ;altered when move past SUBI
    
```

Swap BNEZ and SD by changing address of SD

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

6 clocks: Unroll loop 4 times code to make faster? DAP.F96 21

Unroll Loop Four Times (straightforward way)

```

1 Loop: LD      F0,0(R1)
2 +1  ADDDD    F4,F0,F2
3 +2  SD       0(R1),F4      ;drop SUBI & BNEZ
4      LD      F6,-8(R1)
5      ADDDD    F8,F6,F2
6      SD      -8(R1),F8     ;drop SUBI & BNEZ
7      LD      F10,-16(R1)
8      ADDDD    F12,F10,F2
9      SD      -16(R1),F12   ;drop SUBI & BNEZ
10     LD      F14,-24(R1)
11     ADDDD    F16,F14,F2
12     SD      -24(R1),F16
13     SUBI     R1,R1,#32     ;alter to 4*8
14 +1  BNEZ    R1,LOOP
15     NOP
    
```

Rewrite loop to minimize stalls?

15 + 4 x (1+2) + 1 = 28 clock cycles, or 7 per iteration
Assumes R1 is multiple of 4

DAP.F96 22

Unrolled Loop That Minimizes Stalls

```
1 Loop: LD      F0,0(R1)
2   LD      F6,-8(R1)
3   LD      F10,-16(R1)
4   LD      F14,-24(R1)
5   ADDD    F4,F0,F2
6   ADDD    F8,F6,F2
7   ADDD    F12,F10,F2
8   ADDD    F16,F14,F2
9   SD      0(R1),F4
10  SD      -8(R1),F8
12  SUBI     R1,R1,#32
13  SD      16(R1),F12
13  BNEZ     R1,LOOP
14  SD      8(R1),F16      ; 8-32 = -24
```

- What assumptions made when moved code?

- OK to move store past SUBI even though changes register
- OK to move loads before stores: get right data?
- When is it safe for compiler to do such changes?

14 clock cycles, or 3.5 per iteration

When safe to move instructions?

DAP.F96 23

Compiler Perspectives on Code Movement

- Definitions: compiler concerned about dependencies in **program**, whether or not a HW hazard depends on a given **pipeline**
- Try to schedule to avoid hazards
- (True) **Data dependencies** (RAW if a hazard for HW)
 - Instruction i produces a result used by instruction j, or
 - Instruction j is data dependent on instruction k, and instruction k is data dependent on instruction i.
- If dependent, can't execute in parallel
- Easy to determine for registers (fixed names)
- Hard for memory:
 - Does $100(R4) = 20(R6)$?
 - From different loop iterations, does $20(R6) = 20(R6)$?

DAP.F96 24

Where are the data dependencies?

```
1 Loop: LD      F0,0(R1)
2      ADD D   F4,F0,F2
3      SUB I   R1,R1,8
4      BNEZ   R1,Loop      ;delayed branch
5 SD      8(R1),F4          ;altered when move past SUBI
```

DAP.F96 25

Compiler Perspectives on Code Movement

- Another kind of dependence called **name dependence**: two instructions use same name (register or memory location) but don't exchange data
- **Antidependence** (WAR if a hazard for HW)
 - Instruction j writes a register or memory location that instruction i reads from and instruction i is executed first
- **Output dependence** (WAW if a hazard for HW)
 - Instruction i and instruction j write the same register or memory location; ordering between instructions must be preserved.

DAP.F96 26

Where are the name dependencies?

```

1 Loop: LD      F0,0(R1)
2      ADDD     F4,F0,F2
3      SD       0(R1),F4      ;drop SUBI & BNEZ
4      LD       F0,-8(R1)
2      ADDD     F4,F0,F2      Replace : F0 par F6, F4 par F8
3      SD       -8(R1),F4      ;drop SUBI & BNEZ
7      LD       F0,-16(R1)
8      ADDD     F4,F0,F2      Replace : F0 par F10, F4 par F12
9      SD       -16(R1),F4      ;drop SUBI & BNEZ
10     LD       F0,-24(R1)
11     ADDD     F4,F0,F2      Replace : F0 par F14, F4 par F16
12     SD       -24(R1),F4
13     SUBI     R1,R1,#32      ;alter to 4*8
14     BNEZ     R1,LOOP
15     NOP

```

How can remove them?

DAP.F96 27

Where are the name dependencies?

```

1 Loop: LD      F0,0(R1)
2      ADDD     F4,F0,F2
3      SD       0(R1),F4      ;drop SUBI & BNEZ
4      LD       F6,-8(R1)
5      ADDD     F8,F6,F2
6      SD       -8(R1),F8      ;drop SUBI & BNEZ
7      LD       F10,-16(R1)
8      ADDD     F12,F10,F2
9      SD       -16(R1),F12    ;drop SUBI & BNEZ
10     LD       F14,-24(R1)
11     ADDD     F16,F14,F2
12     SD       -24(R1),F16
13     SUBI     R1,R1,#32      ;alter to 4*8
14     BNEZ     R1,LOOP
15     NOP

```

Called “register renaming”

DAP.F96 28

Compiler Perspectives on Code Movement

- Again Name Dependencies are Hard for Memory Accesses
 - Does $100(R4) = 20(R6)$?
 - From different loop iterations, does $20(R6) = 20(R6)$?
- Our example required compiler to know that if R1 doesn't change then:

$0(R1) \quad -8(R1) \quad -16(R1) \quad -24(R1)$

There were no dependencies between some loads and stores so they could be moved by each other

DAP.F96 29

Compiler Perspectives on Code Movement

- Final kind of dependence called **control dependence**
- Example

if p1 {S1;;}

if p2 {S2;;}

S1 is control dependent on p1 and S2 is control dependent on p2 but not on p1.

DAP.F96 30

Compiler Perspectives on Code Movement

- **Two (obvious) constraints on control dependences:**
 - An instruction that is **control dependent** on a branch cannot be moved **before** the branch so that its execution is no longer controlled by the branch.
 - An instruction that is not **control dependent** on a branch cannot be moved to **after** the branch so that its execution is controlled by the branch.
- **Control dependencies relaxed to get parallelism; get same effect if preserve order of exceptions (address in register checked by branch before use) and data flow (value in register depends on branch)**

DAP.F96 31

Where are the control dependencies?

```
1 Loop: LD      F0,0(R1)
2      ADDD    F4,F0,F2
3      SD      0(R1),F4
4      SUBI    R1,R1,8
5      BEQZ    R1,exit
6      LD      F0,0(R1)
7      ADDD    F4,F0,F2
8      SD      0(R1),F4
9      SUBI    R1,R1,8
10     BEQZ    R1,exit
11     LD      F0,0(R1)
12     ADDD    F4,F0,F2
13     SD      0(R1),F4
14     SUBI    R1,R1,8
15     BEQZ    R1,exit
....
```

DAP.F96 32

When Safe to Unroll Loop?

- Example: Where are data dependencies?
(A,B,C distinct & nonoverlapping)

```
for (i=1; i<=100; i=i+1) {  
    A[i+1] = A[i] + C[i];    /* S1 */  
    B[i+1] = B[i] + A[i+1]; /* S2 */  
}
```

1. S2 uses the value, A[i+1], computed by S1 in the same iteration.
2. S1 uses a value computed by S1 in an earlier iteration, since iteration i computes A[i+1] which is read in iteration i+1. The same is true of S2 for B[i] and B[i+1].

This is a “**loop-carried dependence**”: between iterations

- Implies that iterations are dependent, and can't be executed in parallel
- Not the case for our prior example; each iteration was distinct

DAP.F96 33

HW Schemes: Instruction Parallelism

- Why in HW at run time?
 - Works when can't know real dependence at compile time
 - Compiler simpler
 - Code for one machine runs well on another
- Key idea: Allow instructions behind stall to proceed

```
DIVD  F0, F2, F4  
ADDD  F10, F0, F8  
SUBD  F12, F8, F14
```

- Enables out-of-order execution => out-of-order completion
- ID stage checked both for structuralScoreboard dates to CDC 6600 in 1963

DAP.F96 34

HW Schemes: Instruction Parallelism

- **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 allow instruction to execute whenever 1 & 2 hold, not waiting for prior instructions**
- **CDC 6600: In order issue, out of order execution, out of order commit (also called completion)**

DAP.F96 35

Scoreboard Implications

- **Out-of-order completion => WAR, WAW hazards?**
- **Solutions for WAR**
 - Queue both the operation and copies of its operands
 - Read registers only during Read Operands stage
- **For WAW, must detect hazard: stall until other completes**
- **Need to have multiple instructions in execution phase => multiple execution units or pipelined execution units**
- **Scoreboard keeps track of dependencies, state or operations**
- **Scoreboard replaces ID, EX, WB with 4 stages**

DAP.F96 36

Four Stages of Scoreboard Control

1. Issue—decode instructions & check for structural hazards (ID1)

If a functional unit for the instruction is free and no other active instruction has the same destination register (WAW), the scoreboard issues the instruction to the functional unit and updates its internal data structure. If a structural or WAW hazard exists, then the instruction issue stalls, and no further instructions will issue until these hazards are cleared.

2. Read operands—wait until no data hazards, then read operands (ID2)

A source operand is available if no earlier issued active instruction is going to write it, or if the register containing the operand is being written by a currently active functional unit. When the source operands are available, the scoreboard tells the functional unit to proceed to read the operands from the registers and begin execution. The scoreboard resolves RAW hazards dynamically in this step, and instructions may be sent into execution out of order. DAP.F96 37

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)

Once the scoreboard is aware that the functional unit has completed execution, the scoreboard checks for WAR hazards. If none, it writes results. If WAR, then it stalls the instruction.

Example:

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

CDC 6600 scoreboard would stall SUBD until ADDD reads operands

DAP.F96 38

Three Parts of the Scoreboard

1. Instruction status—which of 4 steps the instruction is in

2. 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

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

DAP.F96 39

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; Result(`D')← FU;
Read operands	Rj and Rk	Rj← No; Rk← No
Execution complete	Functional unit done	
Write result	$\forall f((Fj(f) \neq Fi(FU) \text{ or } Rj(f) = \text{No}) \& (Fk(f) \neq Fi(FU) \text{ or } Rk(f) = \text{No}))$	$\forall f(\text{if } Qj(f) = FU \text{ then } Rj(f) \leftarrow \text{Yes};$ $\forall f(\text{if } Qk(f) = FU \text{ then } Rj(f) \leftarrow \text{Yes};$ $\text{Result}(Fi(FU)) \leftarrow 0; \text{Busy}(FU) \leftarrow \text{No}$

DAP.F96 40

Scoreboard Example

<u>Instruction status</u>				Read	Executic	Write				
Instruction	j	k		Issue	operand complet	Result				
LD	F6	34+	R2							
LD	F2	45+	R3							
MUL1F0	F2	F4								
SUBDF8	F6	F2								
DIVDF10	F0	F6								
ADDDF6	F8	F2								
<u>Functional unit status</u>				dest	S1	S2	FU for j	FU for k	Fj?	Fk?
Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								
<u>Register result status</u>										
Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
	FU									

DAP.F96 41

DAP.F96 41

Scoreboard Example Cycle 1

<u>Instruction status</u>				<i>Read</i>		<i>Execution</i>		<i>Write</i>				
Instruction	<i>j</i>	<i>k</i>		Issue	operand complete	Result						
LD	F6	34+	R2	1								
LD	F2	45+	R3									
MUL1F0	F2	F4										
SUBDF8	F6	F2										
DIVDF10	F0	F6										
ADDDF6	F8	F2										
<u>Functional unit status</u>				<i>dest</i>		<i>S1</i>	<i>S2</i>	<i>FU for j</i>		<i>FU for k</i>	<i>Fj?</i>	<i>Fk?</i>
Time	Name			Busy	Op	<i>Fi</i>	<i>Fj</i>	<i>Fk</i>	<i>Qj</i>	<i>Qk</i>	<i>Rj</i>	<i>Rk</i>
	Integer			Yes	Load	F6		R2				Yes
	Mult1			No								
	Mult2			No								
	Add			No								
	Divide			No								
<u>Register result status</u>												
Clock				<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	...	<i>F30</i>
1	<i>FU</i>			Integer								

DAP.F96 42

Scoreboard Example Cycle 2

<u>Instruction status</u>				Read		Executio		Write		
Instruction	j	k		Issue	operand: complete	Result				
LD	F6	34+	R2	1	2					
LD	F2	45+	R3							
MUL1F0	F2	F4								
SUBDF8	F6	F2								
DIVDF10	F0	F6								
ADDDF6	F8	F2								
<u>Functional unit status</u>										
Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F6		R2				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								
<u>Register result status</u>										
Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
2	FU	Integer								

- Issue 2nd LD?

DAP.F96 43

Scoreboard Example Cycle 3

<u>Instruction status</u>				Read		Execution		Write			
Instruction	<i>j</i>	<i>k</i>		Issue	operand: complete	Result					
LD	F6	34+	R2	1	2	3					
LD	F2	45+	R3								
MUL1F0	F2	F4									
SUBDF8	F6	F2									
DIVDF10	F0	F6									
ADDDF6	F8	F2									
<u>Functional unit status</u>											
Time	Name	Busy	Op	dest <i>Fi</i>	<i>S1</i> <i>Fj</i>	<i>S2</i> <i>Fk</i>	<i>FU for j</i> <i>Qj</i>	<i>FU for k</i> <i>Qk</i>	<i>Fj?</i> <i>Rj</i>	<i>Fk?</i> <i>Rk</i>	
	Integer	Yes	Load	F6		R2				Yes	
	Mult1	No									
	Mult2	No									
	Add	No									
	Divide	No									
<u>Register result status</u>											
Clock		<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	...	<i>F30</i>	
3	FU	Integer									

- Issue MULT?

DAP.F96 44

Scoreboard Example Cycle 4

<u>Instruction status</u>			Read	Execution		Write				
Instruction	j	k	Issue	operands	complete	Result				
LD	F6	34+	R2	1	2	3	4			
LD	F2	45+	R3							
MUL1F0	F2	F4								
SUBDF8	F6	F2								
DIVD	F10	F0	F6							
ADDDF6	F8	F2								
<u>Functional unit status</u>			dest	S1	S2	FU for j	FU for k	Fj?	Fk?	
Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	Yes	Load	F6		R2				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								
<u>Register result status</u>										
Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
4	FU	Integer								

DAP.F96 45

DAP.F96 45

Scoreboard Example Cycle 5

<u>Instruction status</u>				<i>Read Executio: Write</i>							
Instruction	<i>j</i>	<i>k</i>		<i>Issue</i>	<i>operand: complete</i>	<i>Result</i>					
LD	F6	34+	R2	1	2	3	4				
LD	F2	45+	R3	5							
MUL1F0	F2	F4									
SUBDF8	F6	F2									
DIVD	F10	F0	F6								
ADDDF6	F8	F2									
<u>Functional unit status</u>				<i>dest S1 S2 FU for j FU for k Fj? Fk?</i>							
Time	Name			<i>Busy</i>	<i>Op</i>	<i>Fi</i>	<i>Fj</i>	<i>Fk</i>	<i>Qj Qk Rj Rk</i>		
	Integer			Yes	Load	F2		R3	Yes		
	Mult1			No							
	Mult2			No							
	Add			No							
	Divide			No							
<u>Register result status</u>											
Clock				<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	<i>... F30</i>
5		FU		Integer							

DAP.F96 46

DAP.F96 46

Scoreboard Example Cycle 6

<u>Instruction status</u>			Read	Execution	Write					
Instruction	j	k	Issue	operands: complete	Result					
LD F6 34+ R2			1	2	3	4				
LD F2 45+ R3			5	6						
MUL1F0 F2 F4			6							
SUBDF8 F6 F2										
DIVDF10 F0 F6										
ADDDF6 F8 F2										
<u>Functional unit status</u>										
Time	Name	Busy	Op	dest	S1	S2	FU for j	FU for k	Fj?	Fk?
	Integer	Yes	Load	F2		R3				Yes
	Mult1	Yes	Mult	F0	F2	F4	Integer		No	Yes
	Mult2	No								
	Add	No								
	Divide	No								
<u>Register result status</u>										
Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
6	FU	Mult1 Integer								

DAP.F96 47

DAP.F96 47

Scoreboard Example Cycle 7

<u>Instruction status</u>				Read		Executing		Write		
Instruction	j	k		Issue	operand: complete	Result				
LD F6 34+ R2				1	2	3	4			
LD F2 45+ R3				5	6	7				
MUL1F0 F2 F4				6						
SUBDF8 F6 F2				7						
DIVDF10 F0 F6										
ADDDF6 F8 F2										
<u>Functional unit status</u>										
Time	Name	Busy	Op	dest	S1	S2	FU for j	FU for k	Fj?	Fk?
	Integer	Yes	Load	F2		R3				Yes
	Mult1	Yes	Mult	F0	F2	F4	Integer		No	Yes
	Mult2	No								
	Add	Yes	Sub	F8	F6	F2		Integer	Yes	No
	Divide	No								
<u>Register result status</u>										
Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
7	FU	Mult1	Integer			Add				

- Read multiply operands?

DAP.F96 48

Scoreboard Example Cycle 8a

<u>Instruction status</u>			<u>Read Executio. Write</u>									
Instruction	j	k	Issue	operand: complete	Result							
LD F6 34+ R2			1	2	3	4						
LD F2 45+ R3			5	6	7							
MUL1F0 F2 F4			6									
SUBDF8 F6 F2			7									
DIVD F10 F0 F6			8									
ADDDF6 F8 F2												
<u>Functional unit status</u>												
			dest	S1	S2	FU for j		FU for k		Fj?	Fk?	
Time	Name	Busy	Op	Fi	Fj	Fk	Oj	Ok	Rj	Rk		
	Integer	Yes	Load	F2		R3					Yes	
	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	
<u>Register result status</u>												
Clock			F0	F2	F4	F6	F8	F10	F12	...	F30	
8	FU		Mult1	Integer			Add	Divide				

DAP.F96 49

DAP.F96 49

Scoreboard Example Cycle 8b

<u>Instruction status</u>				Read		Executio. Write						
Instruction	j	k		Issue	operands: complete	Result						
LD F6 34+ R2				1	2	3	4					
LD F2 45+ R3				5	6	7	8					
MUL1F0 F2 F4				6								
SUBDF8 F6 F2				7								
DIVD F10 F0 F6				8								
ADDDF6 F8 F2												
<u>Functional unit status</u>												
Time	Name	Busy	Op	dest	S1	S2	FU for j		FU for k		Fj?	Fk?
	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
<u>Register result status</u>												
Clock		F0	F2	F4	F6	F8	F10	F12	...	F30		
8	FU	Mult1				Add	Divide					

DAP.F96 50

Scoreboard Example Cycle 9

<u>Instruction status</u>			Read		Executio: Write						
Instruction	j	k	Issue	operand: complete	Result						
LD	F6	34+ R2	1	2	3	4					
LD	F2	45+ R3	5	6	7	8					
MUL1	F0	F2 F4	6	9							
SUBD	F8	F6 F2	7	9							
DIVD	F10	F0 F6	8								
ADDD	F6	F8 F2									
<u>Functional unit status</u>											
Time	Name	Busy	Op	dest	S1	S2	FU for j		FU for k	Fj?	Fk?
	Integer	No		Fi	Fj	Fk	Oj	Ok	Rj	Rk	
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	
<u>Register result status</u>											
Clock		F0	F2	F4	F6	F8	F10	F12	...	F30	
9	FU	Mult1				Add	Divide				

- Read operands for MULT & SUBD? Issue ADDD? DAP.F96 51

Scoreboard Example Cycle 11

<u>Instruction status</u>				Read Executio: Write									
Instruction	j	k		Issue	operand: complete	Result							
LD	F6	34+	R2	1	2	3	4						
LD	F2	45+	R3	5	6	7	8						
MULTF0	F2	F4		6	9								
SUBDF8	F6	F2		7	9	11							
DIVDF10	F0	F6		8									
ADDDF6	F8	F2											
<u>Functional unit status</u>													
Time	Name			Busy	Op	dest	S1	S2	FU for j		FU for k	Fj?	Fk?
						Fi	Fj	Fk	Qj	Qk	Rj	Rk	
	Integer			No									
8	Mult1			Yes	Mult	F0	F2	F4			Yes	Yes	
	Mult2			No									
0	Add			Yes	Sub	F8	F6	F2			Yes	Yes	
	Divide			Yes	Div	F10	F0	F6	Mult1		No	Yes	
<u>Register result status</u>													
Clock				F0	F2	F4	F6	F8	F10	F12	...	F30	
11	FU			Mult1				Add	Divide				

DAP.F96 52

Scoreboard Example Cycle 12

Instruction status			Read	Executio	Write		
Instruction	j	k	Issue	operands complete	Result		
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MUL1F0	F2	F4		6	9		
SUBDF8	F6	F2		7	9	11	12
DIVDF10	F0	F6		8			
ADDDF6	F8	F2					

Functional unit status			dest	S1	S2	FU for j FU for k Fj?		Fk?		
Time	Name	Busy	Op	Fi	Fj	Fk	Oj	Ok	Rj	Rk
	Integer	No								
7	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
	Add	No								
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status			F0	F2	F4	F6	F8	F10	F12	...	F30
Clock											
12	FU		Mult1					Divide			

- Read operands for DIVD?

DAP.F96 53

Scoreboard Example Cycle 13

<u>Instruction status</u>				Read		Executio		Write	
Instruction	j	k		Issue	operands complete	Result			
LD	F6	34+	R2	1	2	3	4		
LD	F2	45+	R3	5	6	7	8		
MULTF0	F2	F4		6	9				
SUBDF8	F6	F2		7	9	11	12		
DIVDF10	F0	F6		8					
ADDDF6	F8	F2		13					

<u>Functional unit status</u>				dest	S1	S2	FU for j FU for k Fj?		Fk?	
Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	No								
6	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

<u>Register result status</u>				F0	F2	F4	F6	F8	F10	F12	...	F30
Clock												
13		FU		Mult1			Add		Divide			

DAP.F96 54

Scoreboard Example Cycle 14

<u>Instruction status</u>				<u>Read</u>		<u>Executio</u>	<u>Write</u>
Instruction	j	k		Issue	operands complete	Result	
LD F6 34+ R2				1	2	3	4
LD F2 45+ R3				5	6	7	8
MUL1F0 F2 F4				6	9		
SUBDF8 F6 F2				7	9	11	12
DIVD F10 F0 F6				8			
ADDDF6 F8 F2				13	14		

<u>Functional unit status</u>		dest	S1	S2	FU for j		FU for k		Fk?
Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rk
	Integer	No							
5	Mult1	Yes	Mult	F0	F2	F4			Yes Yes
	Mult2	No							
2	Add	Yes	Add	F6	F8	F2			Yes Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No Yes

<u>Register result status</u>		F0	F2	F4	F6	F8	F10	F12	...	F30
Clock										
14	FU	Mult1			Add		Divide			

DAP.F96 55

Scoreboard Example Cycle 15

<u>Instruction status</u>				<u>Read</u>		<u>Executio</u>	<u>Write</u>
Instruction	j	k		Issue	operands complete	Result	
LD F6 34+ R2				1	2	3	4
LD F2 45+ R3				5	6	7	8
MUL1F0 F2 F4				6	9		
SUBDF8 F6 F2				7	9	11	12
DIVD F10 F0 F6				8			
ADDDF6 F8 F2				13	14		

<u>Functional unit status</u>		dest	S1	S2	FU for j		FU for k		Fk?
Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rk
	Integer	No							
4	Mult1	Yes	Mult	F0	F2	F4			Yes Yes
	Mult2	No							
1	Add	Yes	Add	F6	F8	F2			Yes Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No Yes

<u>Register result status</u>		F0	F2	F4	F6	F8	F10	F12	...	F30
Clock										
15	FU	Mult1			Add		Divide			

DAP.F96 56

Scoreboard Example Cycle 16

<u>Instruction status</u>				<u>Read Executio Write</u>			
Instruction	j	k		Issue	operands complete	Result	
LD F6 34+ R2				1	2	3	4
LD F2 45+ R3				5	6	7	8
MUL1F0 F2 F4				6	9		
SUBDF8 F6 F2				7	9	11	12
DIVDF10 F0 F6				8			
ADDDF6 F8 F2				13	14	16	

<u>Functional unit status</u>				<u>dest S1 S2 FU for j FU for k Fj? Fk?</u>						
Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	No								
3	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
0	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

<u>Register result status</u>				<u>F0 F2 F4 F6 F8 F10 F12 ... F30</u>								
Clock												
16	FU			Mult1		Add		Divide				

DAP.F96 57

DAP.F96 57

Scoreboard Example Cycle 17

<u>Instruction status</u>				Read		Executio		Write	
Instruction	j	k		Issue	operands complete	Result			
LD F6 34+ R2				1	2	3	4		
LD F2 45+ R3				5	6	7	8		
MULTF0 F2 F4				6	9				
SUBDF8 F6 F2				7	9	11	12		
DIVDF10 F0 F6				8					
ADDDF6 F8 F2				13	14	16			

<u>Functional unit status</u>				dest	S1	S2	FU for j FU for k Fj?		Fk?	
Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	No								
2	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

<u>Register result status</u>												
Clock				F0	F2	F4	F6	F8	F10	F12	...	F30
17	FU			Mult1			Add		Divide			

- Write result of ADDD?

DAP.F96 58

Scoreboard Example Cycle 18

Instruction status				Read		Execution	Write				
Instruction	j	k		Issue	operand:	complete	Result				
LD	F6	34+	R2	1	2	3	4				
LD	F2	45+	R3	5	6	7	8				
MUL1F0	F2	F4		6	9						
SUBDF8	F6	F2		7	9	11	12				
DIVD	F10	F0	F6	8							
ADDDF6	F8	F2		13	14	16					
Functional unit status				dest		S1	S2	FU for j		FU for k	
Time	Name			Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj
	Integer			No							
1	Mult1			Yes	Mult	F0	F2	F4			Yes
	Mult2			No							
	Add			Yes	Add	F6	F8	F2			Yes
	Divide			Yes	Div	F10	F0	F6	Mult1		No
Register result status											
Clock				F0	F2	F4	F6	F8	F10	F12	...
18	FU			Mult1			Add		Divide		

DAP.F96 59

Scoreboard Example Cycle 19

Instruction status				Read		Execution	Write				
Instruction	j	k		Issue	operand:	complete	Result				
LD	F6	34+	R2	1	2	3	4				
LD	F2	45+	R3	5	6	7	8				
MUL1F0	F2	F4		6	9	19					
SUBDF8	F6	F2		7	9	11	12				
DIVD	F10	F0	F6	8							
ADDDF6	F8	F2		13	14	16					
Functional unit status				dest		S1	S2	FU for j		FU for k	
Time	Name			Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj
	Integer			No							
0	Mult1			Yes	Mult	F0	F2	F4			Yes
	Mult2			No							
	Add			Yes	Add	F6	F8	F2			Yes
	Divide			Yes	Div	F10	F0	F6	Mult1		No
Register result status											
Clock				F0	F2	F4	F6	F8	F10	F12	...
19	FU			Mult1			Add		Divide		

DAP.F96 60

Scoreboard Example Cycle 20

<u>Instruction status</u>				Read		Execute		Write				
Instruction	j	k		Issue	operands	complete	Result					
LD	F6	34+	R2	1	2	3	4					
LD	F2	45+	R3	5	6	7	8					
MUL1F0	F2	F4		6	9	19	20					
SUBDF8	F6	F2		7	9	11	12					
DIVD F10	F0	F6		8								
ADDDF6	F8	F2		13	14	16						
<u>Functional unit status</u>				dest		S1	S2	FU for j FU for k Fj?		Fk?		
Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk		
	Integer	No										
	Mult1	No										
	Mult2	No										
	Add	Yes	Add	F6	F8	F2			Yes	Yes		
	Divide	Yes	Div	F10	F0	F6			Yes	Yes		
<u>Register result status</u>												
Clock				F0	F2	F4	F6	F8	F10	F12	...	F30
20	FU			Add				Divide				

DAP.F96 61

DAP.F96 61

Scoreboard Example Cycle 21

<u>Instruction status</u>				Read		Executio		Write				
Instruction	j	k		Issue	operand:	complete	Result					
LD	F6	34+	R2	1	2	3	4					
LD	F2	45+	R3	5	6	7	8					
MUL1F0	F2	F4		6	9	19	20					
SUBDF8	F6	F2		7	9	11	12					
DIVD F10	F0	F6		8	21							
ADDDF6	F8	F2		13	14	16						
<u>Functional unit status</u>				dest		S1	S2	FU for j	FU for k	Fj?	Fk?	
Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk		
	Integer	No										
	Mult1	No										
	Mult2	No										
	Add	Yes	Add	F6	F8	F2			Yes	Yes		
	Divide	Yes	Div	F10	F0	F6			Yes	Yes		
<u>Register result status</u>												
Clock				F0	F2	F4	F6	F8	F10	F12	...	F30
21	FU			Add				Divide				

DAP.F96 62

Scoreboard Example Cycle 22

Instruction status				Read Executio. Write			
Instruction	j	k		Issue	operands complete	Result	
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTF0	F2	F4		6	9	19	20
SUBDF8	F6	F2		7	9	11	12
DIVDF10	F0	F6		8	21		
ADDDF6	F8	F2		13	14	16	22

Functional unit status		dest	S1	S2	FU for j		FU for k		Fj?	Fk?
Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
40	Divide	Yes	Div	F10	F0	F6			Yes	Yes

Register result status		F0	F2	F4	F6	F8	F10	F12	...	F30
Clock										
22	FU									

DAP.F96 63

Scoreboard Example Cycle 61

Instruction status				Read Executio. Write			
Instruction	j	k		Issue	operands complete	Result	
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTF0	F2	F4		6	9	19	20
SUBDF8	F6	F2		7	9	11	12
DIVDF10	F0	F6		8	21	61	
ADDDF6	F8	F2		13	14	16	22

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

Register result status		F0	F2	F4	F6	F8	F10	F12	...	F30
Clock										
61	FU									

DAP.F96 64

Scoreboard Example Cycle 62

<u>Instruction status</u>				<u>Read</u>		<u>Execution</u>		<u>Write</u>	
Instruction	j	k		Issue	operands complete	Result			
LD F6 34+ R2				1	2	3	4		
LD F2 45+ R3				5	6	7	8		
MULT 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		

<u>Functional unit status</u>		dest	S1	S2	FU for j		FU for k		Fj?	Fk?
Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
	0 Divide	No								

<u>Register result status</u>		F0	F2	F4	F6	F8	F10	F12	...	F30
Clock										
62	FU									

DAP.F96 65

CDC 6600 Scoreboard

- Speedup 1.7 from compiler; 2.5 by hand
BUT slow memory (no cache) limits benefit
- 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

DAP.F96 66

Summary

- **Instruction Level Parallelism (ILP) in SW or HW**
- **Loop level parallelism is easiest to see**
- **SW parallelism dependencies defined for program, hazards if HW cannot resolve**
- **SW dependencies/compiler sophistication determine if compiler can unroll loops**
 - Memory dependencies hardest to determine
- **HW exploiting ILP**
 - Works when can't know dependence at run time
 - Code for one machine runs well on another
- **Key idea of Scoreboard: Allow instructions behind stall to proceed (Decode => Issue instr & read operands)**
 - Enables out-of-order execution => out-of-order completion
 - ID stage checked both for structural

DAP.F96 67