Lecture 3:
Introduction to Advanced Pipelining

Review: Evaluating Branch Alternatives

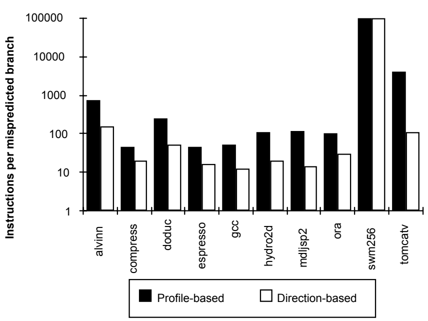
- Two part solution:
  - Determine branch taken or not sooner, AND
  - Compute taken branch address earlier

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

Scheduling scheme	*Branch penalty*	*CPI*	*speedup v. unpipelined*	*speedup v. stall*
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

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



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?

Opérations multi-cycles

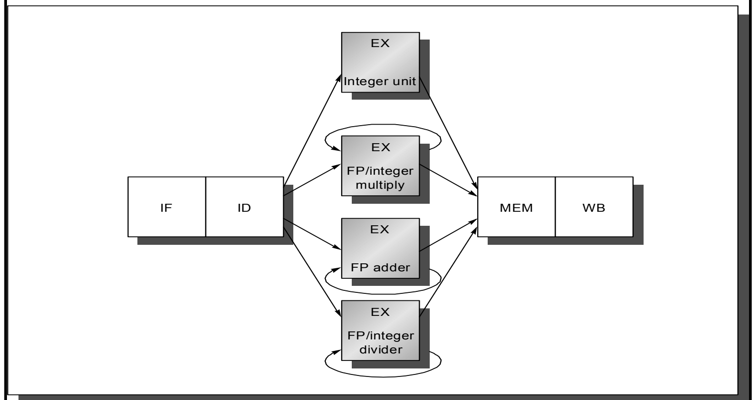


FIGURE 3.42 The DLX pipeline with three additional unpipelined, floating-point, functional units.

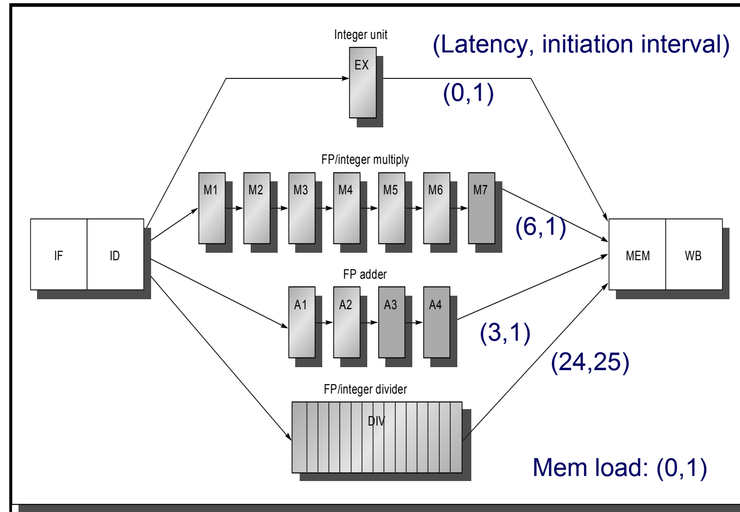


FIGURE 3.44 A pipeline that supports multiple outstanding FP operations.

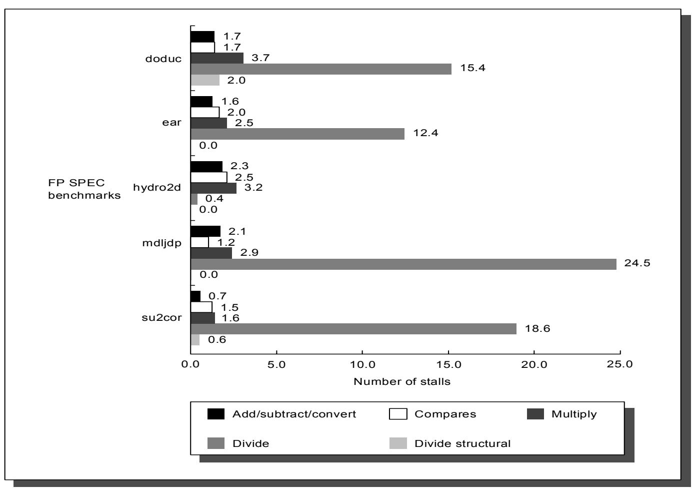


FIGURE 3.48 Stalls per FP operation for each major type of FP operation.

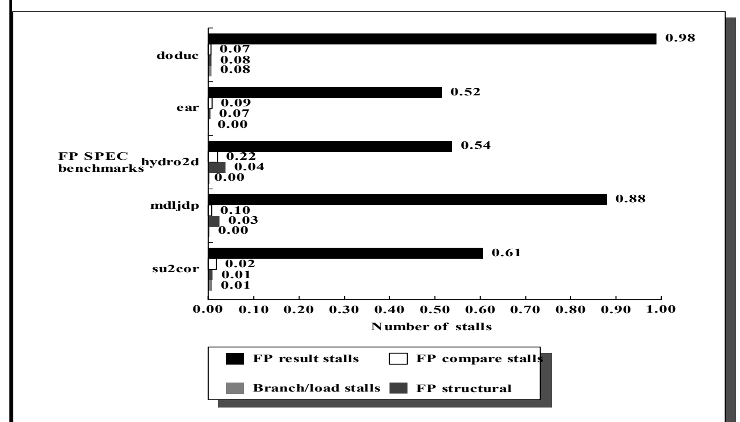


FIGURE 3.49 The stalls occurring for the DLX FP pipeline for the five FP SPEC benchmarks.

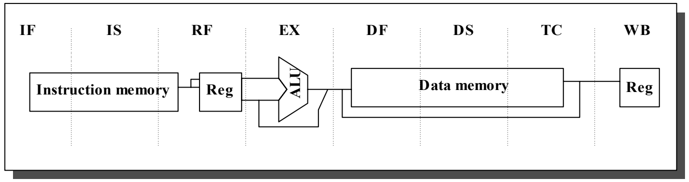


FIGURE 3.50 The eight-stage pipeline structure of the R4000 uses pipelined instruction and data caches.

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?

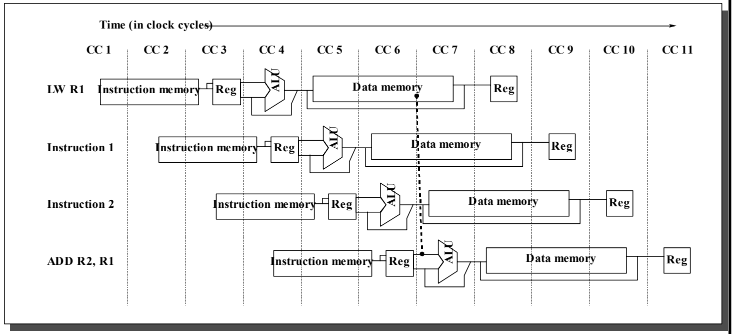


FIGURE 3.51 The structure of the R4000 integer pipeline leads to a two-cycle load delay.

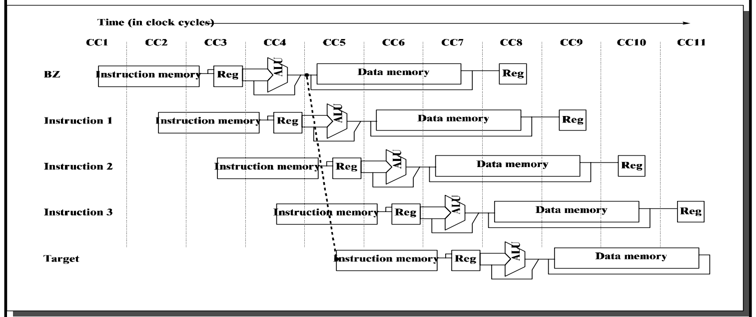


FIGURE 3.53 The basic branch delay is three cycles, since the condition evaluation is performed during EX.

Case Study: MIPS R4000

TWO Cycle Load Latency

IF	IS	RF	EX	DF	$\boxed{\text{DS}}$	TC	WB
	IF	IS	RF	EX	DF	DS	TC
		IF	IS	RF	EX	DF	DS
			IF	IS	RF	$\boxed{\text{EX}}$	DF
				IF	IS	RF	EX
					IF	IS	RF
						IF	IS
							IF

THREE Cycle Branch Latency
(conditions evaluated during EX phase)

IF	IS	RF	$\boxed{\text{EX}}$	DF	DS	TC	WB
	IF	IS	RF	EX	DF	DS	TC
		IF	IS	RF	EX	DF	DS
			IF	IS	RF	EX	DF
				$\boxed{\text{IF}}$	IS	RF	EX
					IF	IS	RF
						IF	IS
							IF

Delay slot plus two stalls
Branch likely cancels delay slot if not taken

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**

MIPS FP Pipe Stages

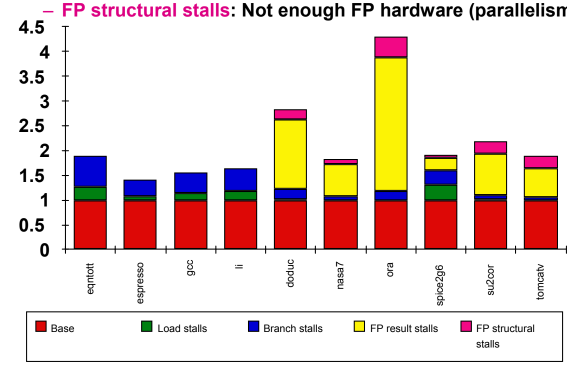
FP Instr	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*…*
Add, Subtract	**U**	**S+A**	**A+R**	**R+S**					
Multiply	**U**	**E+M**	**M**	**M**	**M**	**N**	**N+A**	**R**	
Divide	**U**	**A**	**R**	**D^{28}**	**…**	**D+A**	**D+R, D+R, D+A, D+R, A, R**		
Square root	**U**	**E**	**$(A+R)^{108}$**		**…**	**A**	**R**		
Negate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

Stages:

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

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)**



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

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
```

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
Integer op	Integer op	0

- Where are the stalls?

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
```

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
Integer op	Integer op	0

FP Loop Showing Stalls

```
1  Loop: LD    F0,0(R1)           1  Loop: LD    F0,0(R1)
2        stall                    4        SUBI  R1,R1,8
3        ADDD  F4,F0,F2           3        ADDD  F4,F0,F2
4        stall                    2        stall
5        stall                    5        BNEZ  R1,Loop
6        SD    0(R1),F4           6        SD    8(R1),F4
7        SUBI  R1,R1,8
8        stall
9        BNEZ  R1,Loop     ;branch R1!=zero
10       stall             ;delayed branch slot
```

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

- 10 clocks: Rewrite code to minimize stalls?

Revised FP Loop Minimizing Stalls

```
1 Loop:  LD     F0,0(R1)
4        SUBI   R1,R1,8
3        ADDD   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?

Unroll Loop Four Times (straightforward way)

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

```
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
```

?

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

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?

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 depedent, 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)?

Where are the data dependencies?

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


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.

Where are the name dependencies?

```
1 Loop: LD    F0,0(R1)
2       ADDD  F4,F0,F2
3       SD    0(R1),F4
4       LD    F0,-8(R1)
2       ADDD  F4,F0,F2
3       SD    -8(R1),F4
7       LD    F0,-16(R1)
8       ADDD  F4,F0,F2
9       SD    -16(R1),F4
10      LD    F0,-24(R1)
11      ADDD  F4,F0,F2
12      SD    -24(R1),F4
13      SUBI  R1,R1,#32
14      BNEZ  R1,LOOP
15      NOP
```

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

How can remove them?

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"

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:**

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

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

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.

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)**

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
....
```

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

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 structural and data hazards
  - Scoreboard dates to CDC 6600 in 1963

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)

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

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.

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

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

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	∀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))← 0; Busy(FU)← No

Scoreboard Example

Instruction status			Read	Executi	Write
Instruction j k		Issue	operanc	complet	Result
LD F6 34+ R2					
LD F2 45+ R3					
MULT F0 F2 F4					
SUBD F8 F6 F2					
DIVD F10 F0 F6					
ADDD F6 F8 F2					

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								
	Divide	No								

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

Scoreboard Example Cycle 1

Instruction status			Read	Executic	Write
Instruction j k		Issue	operand	complete	Result
LD F6 34+ R2		1			
LD F2 45+ R3					
MULT F0 F2 F4					
SUBD F8 F6 F2					
DIVD F10 F0 F6					
ADDD F6 F8 F2					

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	Yes	Load	F6		R2				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status		F0	F2	F4	F6	F8	F10	F12	...	F30
Clock 1	FU				Integer					

Scoreboard Example Cycle 2

Instruction status			Read	Executic	Write
Instruction j k		Issue	operand	complete	Result
LD F6 34+ R2		1	2		
LD F2 45+ R3					
MULT F0 F2 F4					
SUBD F8 F6 F2					
DIVD F10 F0 F6					
ADDD F6 F8 F2					

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	Yes	Load	F6		R2				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

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

- **Issue 2nd LD?**

Scoreboard Example Cycle 3

Instruction status			Read	Executic	Write
Instruction j k		Issue	operand	complete	Result
LD F6 34+ R2		1	2	3	
LD F2 45+ R3					
MULT F0 F2 F4					
SUBD F8 F6 F2					
DIVD F10 F0 F6					
ADDD F6 F8 F2					

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	Yes	Load	F6		R2				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status		F0	F2	F4	F6	F8	F10	F12	...	F30
Clock 3	FU				Integer					

- **Issue MULT?**

Scoreboard Example Cycle 4

Instruction status				Read	Executic	Write
Instruction	j	k	Issue	operand	complet	Result
LD F6	34+	R2	1	2	3	4
LD F2	45+	R3				
MULT F0	F2	F4				
SUBD F8	F6	F2				
DIVD F10	F0	F6				
ADDD F6	F8	F2				

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	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
4 FU				Integer					

Scoreboard Example Cycle 5

Instruction status				Read	Executic	Write
Instruction	j	k	Issue	operand	complet	Result
LD F6	34+	R2	1	2	3	4
LD F2	45+	R3	5			
MULT F0	F2	F4				
SUBD F8	F6	F2				
DIVD F10	F0	F6				
ADDD F6	F8	F2				

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	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				Read	Executic	Write
Instruction	j	k	Issue	operand	complet	Result
LD F6	34+	R2	1	2	3	4
LD F2	45+	R3	5	6		
MULT F0	F2	F4	6			
SUBD F8	F6	F2				
DIVD F10	F0	F6				
ADDD F6	F8	F2				

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	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				Read	Executic	Write
Instruction	j	k	Issue	operand	complet	Result
LD F6	34+	R2	1	2	3	4
LD F2	45+	R3	5	6	7	
MULT F0	F2	F4	6			
SUBD F8	F6	F2	7			
DIVD F10	F0	F6				
ADDD F6	F8	F2				

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	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								

Register result status									
Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
7 FU	Mult1	Integer			Add				

- **Read multiply operands?**

Scoreboard Example Cycle 8a

Instruction status	j	k	Issue	Read operand	Executic complet	Write Result
LD F6	34+	R2	1	2	3	4
LD F2	45+	R3	5	6	7	
MULT 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 for j Qj	FU for k Qk	Fj? Rj	Fk? 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

Register result status

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

Scoreboard Example Cycle 8b

Instruction status	j	k	Issue	Read operand	Executic complet	Write Result
LD F6	34+	R2	1	2	3	4
LD F2	45+	R3	5	6	7	8
MULT 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 for j Qj	FU for k 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	j	k	Issue	Read operand	Executic complet	Write Result
LD F6	34+	R2	1	2	3	4
LD F2	45+	R3	5	6	7	8
MULT 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 for j Qj	FU for k 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 & SUBD? Issue ADDD?**

Scoreboard Example Cycle 11

Instruction status	j	k	Issue	Read operand	Executic complet	Write Result
LD F6	34+	R2	1	2	3	4
LD F2	45+	R3	5	6	7	8
MULT 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 for j Qj	FU for k Qk	Fj? Rj	Fk? 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

Register result status

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

Scoreboard Example Cycle 12

Instruction status	j	k	Issue	Read operand	Executic complet	Write Result
LD F6	34+	R2	1	2	3	4
LD F2	45+	R3	5	6	7	8
MULT 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 for j Qj	FU for k Qk	Fj? Rj	Fk? 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 Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
12	FU	Mult1					Divide			

- **Read operands for DIVD?**

Scoreboard Example Cycle 13

Instruction status	j	k	Issue	Read operand	Executic complet	Write Result
LD F6	34+	R2	1	2	3	4
LD F2	45+	R3	5	6	7	8
MULT 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 for j Qj	FU for k Qk	Fj? Rj	Fk? 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

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

Scoreboard Example Cycle 14

Instruction status	j	k	Issue	Read operand	Executic complet	Write Result
LD F6	34+	R2	1	2	3	4
LD F2	45+	R3	5	6	7	8
MULT 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 for j Qj	FU for k Qk	Fj? Rj	Fk? 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

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

Scoreboard Example Cycle 15

Instruction status	j	k	Issue	Read operand	Executic complet	Write Result
LD F6	34+	R2	1	2	3	4
LD F2	45+	R3	5	6	7	8
MULT 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 for j Qj	FU for k Qk	Fj? Rj	Fk? 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

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

Scoreboard Example Cycle 16

Instruction status	j	k	Issue	Read operand	Executic complet	Write Result
Instruction	j	k	Issue	operand	complet	Result
LD F6	34+	R2	1	2	3	4
LD F2	45+	R3	5	6	7	8
MULT 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 for j Qj	FU for k Qk	Fj? Rj	Fk? 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

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

Scoreboard Example Cycle 17

Instruction status	j	k	Issue	Read operand	Executic complet	Write Result
Instruction	j	k	Issue	operand	complet	Result
LD F6	34+	R2	1	2	3	4
LD F2	45+	R3	5	6	7	8
MULT 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 for j Qj	FU for k Qk	Fj? Rj	Fk? 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

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

- **Write result of ADDD?**

Scoreboard Example Cycle 18

Instruction status	j	k	Issue	Read operand	Executic complet	Write Result
Instruction	j	k	Issue	operand	complet	Result
LD F6	34+	R2	1	2	3	4
LD F2	45+	R3	5	6	7	8
MULT 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 for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	No								
1	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

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

Scoreboard Example Cycle 19

Instruction status	j	k	Issue	Read operand	Executic complet	Write Result
Instruction	j	k	Issue	operand	complet	Result
LD F6	34+	R2	1	2	3	4
LD F2	45+	R3	5	6	7	8
MULT 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 for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	No								
0	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

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

Scoreboard Example Cycle 20

Instruction status	j	k	Issue	Read operand	Executic complet	Write 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			
ADDD F6	F8	F2	13	14	16	

Functional unit status

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			Yes	Yes
	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	j	k	Issue	Read operand	Executic complet	Write 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		
ADDD F6	F8	F2	13	14	16	

Functional unit status

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

Register result status

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

Scoreboard Example Cycle 22

Instruction status	j	k	Issue	Read operand	Executic complet	Write 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		
ADDD F6	F8	F2	13	14	16	22

Functional unit status

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

Register result status

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

Scoreboard Example Cycle 61

Instruction status	j	k	Issue	Read operand	Executic complet	Write 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	
ADDD F6	F8	F2	13	14	16	22

Functional unit status

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

Register result status

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

Scoreboard Example Cycle 62

Instruction status	j	k	Issue	Read operand	Executic complet	Write 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

Functional unit status

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

Register result status

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

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), especailly integer/load store units**
  - **Do not issue on structural hazards**
  - **Wait for WAR hazards**
  - **Prevent WAW hazards**

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**