CS252
Graduate Computer Architecture
Lecture 5

Software Scheduling around Hazards
Out-of-order Scheduling

September 15, 2000

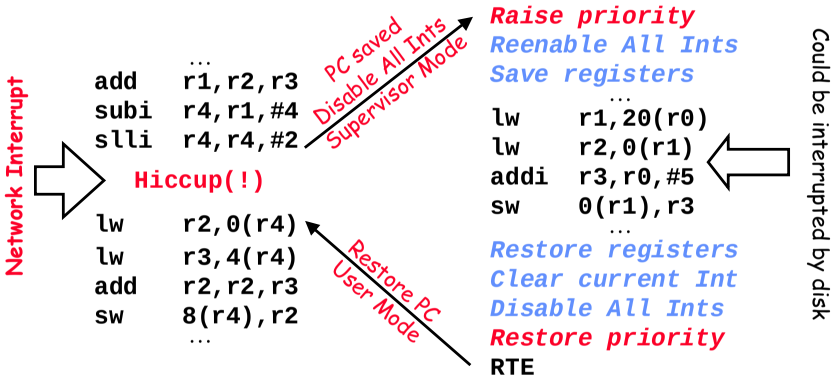
Prof. John Kubiatowicz

Review: Control Flow and Exceptions

- RISC vs CISC was about virtualizing the CPU interface, not simple vs complex instructions
- Control flow is the biggest problem for computer architects. This is getting worse:
  - Modern computer languages such as C++ and Java user many smaller procedure calls (method invocations)
  - Networked devices need to respond quickly to many external events.
- Talked about CRISP method of merging multiple instructions together in on-chip cache
  - This was actually a limited form of recompilation for on-chip VLIW. We will see this in greater detail later
- Interrupts vs Polling: two sides of a coin
  - Interrupts ensure predictable handling of devices (can be guaranteed to happen by OS)
  - Polling has lower overhead if device events frequent
  - Interrupts have lower overhead if device events infrequent

Review: Device Interrupt

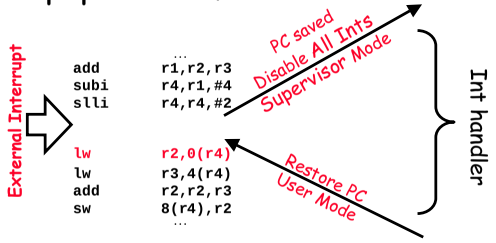
(Say, arrival of network message)



Note that priority must be raised to avoid recursive interrupts!

Review: Precise Interrupts/Exceptions

- An interrupt or exception is considered *precise* if there is a single instruction (or interrupt point) for which:
  - All instructions before that have committed their state
  - No following instructions (including the interrupting instruction) have modified any state.
- This means, that you can restart execution at the interrupt point and "get the right answer"
  - Implicit in our previous example of a device interrupt:
    - » Interrupt point is at first lw instruction



Review: Precise interrupt point requires multiple PCs to describe in presence of delayed branches

```
      addi  r4,r3,#4
      sub   r1,r2,r3
PC:   bne   r1,there
PC+4: and   r2,r3,r5
      <other insts>
```

⇦ Interrupt point described as <PC,PC+4>

```
      addi  r4,r3,#4
      sub   r1,r2,r3
PC:   bne   r1,there
PC+4: and   r2,r3,r5
      <other insts>
```

⇦ Interrupt point described as:

<PC+4,there> (branch was taken)
or
<PC+4,PC+8> (branch was not taken)

Impact of Hazards on Performance

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?

Case Study: MIPS R4000

TWO Cycle Load Latency	IF	IS	RF	EX	DF	DS	TC	WB
		IF	IS	RF	EX	DF	DS	TC
			IF	IS	RF	EX	DF	DS
				IF	IS	RF	EX	DF
					IF	IS	RF	EX
						IF	IS	RF
							IF	IS
								IF
THREE Cycle Branch Latency (conditions evaluated during EX phase)	IF	IS	RF	EX	DF	DS	TC	WB
		IF	IS	RF	EX	DF	DS	TC
			IF	IS	RF	EX	DF	DS
				IF	IS	RF	EX	DF
					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

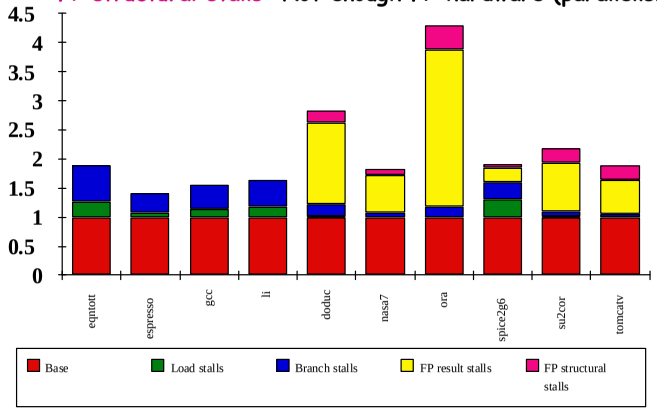
FP Instr	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	…
Add, Subtract	U	S+A	A+R	R+S					
Multiply	U	E+M	M	M	M	N	N+A	R	
Divide	U	A	R	D^{28}	…	D+A	D+R,	D+R,	D+A, D+R, A, R
Square root	U	E	$(A+R)^{108}$	…	A	R			
Negate	U	S							
Absolute value	U	S							
FP compare	U	A	R						

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
  - First SW, then HW approaches
- 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*	*Execution in cycles*	*Latency in cycles*
FP ALU op	Another FP ALU op	4	3
FP ALU op	Store double	3	2
Load double	FP ALU op	1	1
Load double	Store double	1	0
Integer op	Integer op	1	0

Where are the stalls?

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       BNEZ  R1,Loop    ;branch R1!=zero
9       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

- 9 clocks: Rewrite code to minimize stalls?

Revised FP Loop Minimizing Stalls

```
1 Loop: LD    F0,0(R1)
2       stall
3       ADDD  F4,F0,F2
4       SUBI  R1,R1,8
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       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
```

1 cycle stall

2 cycles stall

Rewrite loop to minimize stalls?

15 + 4 x (1+2) = 27 clock cycles, or 6.8 per iteration

Assumes R1 is multiple of 4

Unrolled Loop That Minimizes Stalls

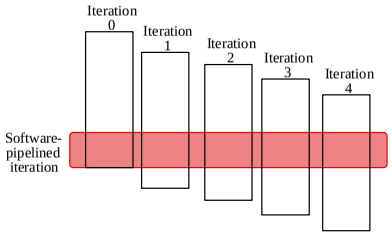
```
1 Loop: LD     F0,0(R1)
2       LD     F6,-8(R1)
3       LD     F10,-16(R1)
4       LD     F14,-24(R1)
5       ADDD   F4,F0,F2
6       ADDD   F8,F6,F2
7       ADDD   F12,F10,F2
8       ADDD   F16,F14,F2
9       SD     0(R1),F4
10      SD     -8(R1),F8
11      SD     -16(R1),F12
12      SUBI   R1,R1,#32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

Another possibility: Software Pipelining

- Observation: if iterations from loops are independent, then can get more ILP by taking instructions from different iterations
- Software pipelining: reorganizes loops so that each iteration is made from instructions chosen from different iterations of the original loop (- Tomasulo in SW)



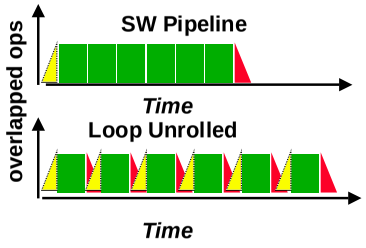
Software Pipelining Example

Before: Unrolled 3 times

```
1  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 SUBI  R1,R1,#24
11 BNEZ  R1,LOOP
```

After: Software Pipelined

```
1  SD    0(R1),F4 ; Stores M[i]
2  ADDD  F4,F0,F2 ; Adds to M[i-1]
3  LD    F0,-16(R1); Loads M[i-2]
4  SUBI  R1,R1,#8
5  BNEZ  R1,LOOP
```



- Symbolic Loop Unrolling
  - Maximize result-use distance
  - Less code space than unrolling
  - Fill & drain pipe only once per loop vs. once per each unrolled iteration in loop unrolling

5 cycles per iteration

CS 252 Administrivia

- Textbook Reading for Lectures 5 to 7
  - Computer Architecture: A Quantitative Approach, Chapter 4, Appendix B
- Complete list of papers that I have "handed" out is now off the handouts page
  - I have indicated which papers are in the ISCA Retrospective
  - More papers there than I have actually required. Check them out!
- Reading assignment for Wednesday will be up later today (tonight?).
  - Will try to have a couple of reading assignments up ahead of time

Compiler Perspectives on Code Movement

- Compiler concerned about dependencies in program
- Whether or not a HW hazard depends on pipeline
- Try to schedule to avoid hazards that cause performance losses
- (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 ("memory disambiguation" problem):
  - 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     ;drop SUBI & BNEZ
4      LD     F0,-8(R1)
5      ADDD   F4,F0,F2
6      SD     -8(R1),F4    ;drop SUBI & BNEZ
7      LD     F0,-16(R1)
8      ADDD   F4,F0,F2
9      SD     -16(R1),F4   ;drop SUBI & BNEZ
10     LD     F0,-24(R1)
11     ADDD   F4,F0,F2
12     SD     -24(R1),F4
13     SUBI   R1,R1,#32    ;alter to 4*8
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

- Name Dependencies are Hard to discover 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.
  - An instruction that is not control dependent on a branch cannot be moved to after the branch (or its execution will be 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=0; 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

- For our prior example, each iteration was distinct
- Implies that iterations can't be executed in parallel, Right????

Does a loop-carried dependence mean there is no parallelism???

- Consider:

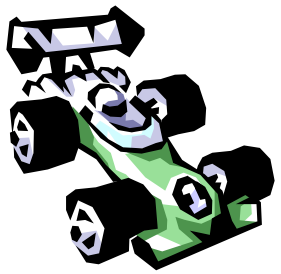
```
for (i=0; i< 8; i=i+1) {
      A = A + C[i];    /* S1 */
}
```

Could compute:

```
"Cycle 1":  temp0 = C[0] + C[1];
            temp1 = C[2] + C[3];
            temp2 = C[4] + C[5];
            temp3 = C[6] + C[7];
"Cycle 2":  temp4 = temp0 + temp1;
            temp5 = temp2 + temp3;
"Cycle 3":  A = temp4 + temp5;
```

- Relies on associative nature of "+".
- See "Parallelizing Complex Scans and Reductions" by Allan Fisher and Anwar Ghuloum (handed out next week)

Speeding things up: Can hardware help?



Can HW get CPI closer to 1?

- 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 #1:
  Allow instructions behind stall to proceed

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

  Out-of-order execution ⇒ out-of-order completion?
- Key idea #2: Register Renaming

```
DIVD   F0,F2,F4          DIVD F0,F2,F4
ADDD   F10,F0,F8         ADDD F10,F0,F8
SUBD   F0,F8,F14   ⇒     SUBD F100,F8,F14
MULD   F6,F10,F0         MULD F6,F10,F100
```

  Totally removes WAR and WAW hazards.

Moving beyond the five-stage pipeline:

- Why limit performance for slow/less frequent ops?
  - Variable latencies/out-of-order execution desirable
- How do we prevent WAR and WAW hazards?
- How do we deal with variable latency?
  - Forwarding for RAW hazards harder.

Instruction	Clock Cycle Number																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
LD F6,34(R2)	IF	ID	EX	MEM	WB												
LD F2,45(R3)		IF	ID	EX	MEM	WB											
MULTD F0,F2,F4			IF	ID	stall	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	MEM	WB
SUBD F8,F6,F2				IF	ID	A1	A2	MEM	WB								
DIVD F10,F0,F6					IF	ID	stall	stall	stall	stall	stall	stall	stall	stall	stall	D1	D2
ADDD F6,F8,F2						IF	ID	A1	A2	MEM	WB						

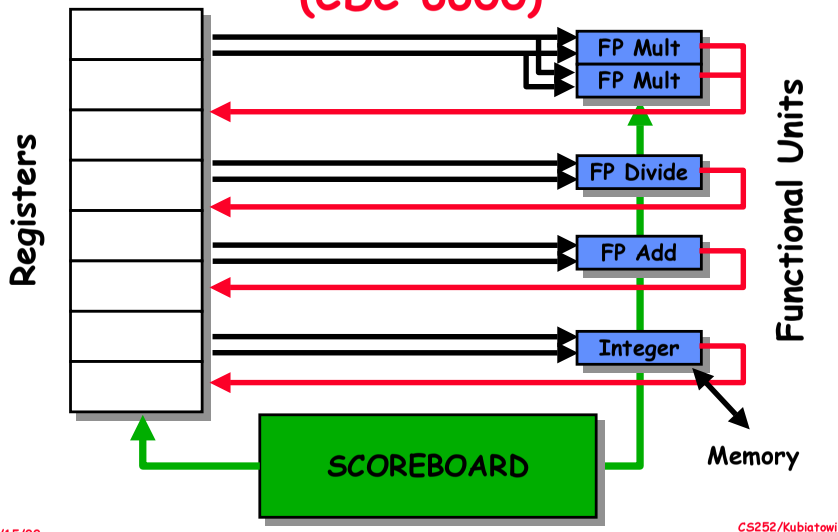
RAW

WAR

Scoreboard: a bookkeeping technique

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

Scoreboard Architecture (CDC 6600)



Scoreboard Implications

- Out-of-order completion => WAR, WAW hazards?
- Solutions for WAR:
  - Stall writeback until registers have been read
  - Read registers only during Read Operands stage
- Solution for WAW:
  - Detect hazard and stall issue of new instruction until other instruction completes
- No register renaming!
- Need to have multiple instructions in execution phase => multiple execution units or pipelined execution units
- Scoreboard keeps track of dependencies between instructions that have already issued.
- Scoreboard replaces ID, EX, WB with 4 stages

Four Stages of Scoreboard Control

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

Four Stages of Scoreboard Control

- 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.
- Write result—finish execution (WB)
  - Stall until no WAR hazards with previous instructions:

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

CDC 6600 scoreboard would stall SUBD until ADDD reads operands

Three Parts of the Scoreboard

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

Scoreboard Example

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2				
LD	F2	45+	R3				
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status:

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

Register result status:

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

Detailed Scoreboard Pipeline Control

Instruction status	Wait until	Bookkeeping
Issue	Not busy (FU) and not result(D)	Busy(FU)← yes; Op(FU)← op; Fi(FU)← `D'; Fj(FU)← `S1'; Fk(FU)← `S2'; Qj← Result('S1'); Qk← Result(`S2'); Rj← not Qj; Rk← not Qk; 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: Cycle 1

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1			
LD	F2	45+	R3				
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F6		R2				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status:

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

Scoreboard Example: Cycle 2

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2		
LD	F2	45+	R3				
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F6		R2				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status:

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

- Issue 2nd LD?

Scoreboard Example: Cycle 3

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	
LD	F2	45+	R3				
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F6		R2				No
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status:

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

- Issue MULT?

Scoreboard Example: Cycle 4

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3				
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status:

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

Register result status:

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

Scoreboard Example: Cycle 5

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5			
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F2		R3				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
5	FU		Integer							

Scoreboard Example: Cycle 6

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6		
MULTD	F0	F2	F4	6			
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F2		R3				Yes
	Mult1	Yes	Mult	F0	F2	F4	Integer		No	Yes
	Mult2	No								
	Add	No								
	Divide	No								

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
6	FU	Mult1	Integer							

Scoreboard Example: Cycle 7

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	
MULTD	F0	F2	F4	6			
SUBD	F8	F6	F2	7			
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F2		R3				No
	Mult1	Yes	Mult	F0	F2	F4	Integer		No	Yes
	Mult2	No								
	Add	Yes	Sub	F8	F6	F2		Integer	Yes	No
	Divide	No								

Register result status:

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

- Read multiply operands?

Scoreboard Example: Cycle 8a
(First half of clock cycle)

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	
MULTD	F0	F2	F4	6			
SUBD	F8	F6	F2	7			
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F2		R3				No
	Mult1	Yes	Mult	F0	F2	F4	Integer		No	Yes
	Mult2	No								
	Add	Yes	Sub	F8	F6	F2		Integer	Yes	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

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

Scoreboard Example: Cycle 8b
(Second half of clock cycle)

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6			
SUBD	F8	F6	F2	7			
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
	Add	Yes	Sub	F8	F6	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

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

Scoreboard Example: Cycle 9

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9		
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	No								
10	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
2	Add	Yes	Sub	F8	F6	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Note → Remaining

Register result status:

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

- Read operands for MULT & SUB? Issue ADDD?

Scoreboard Example: Cycle 10

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9		
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	No								
9	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
1	Add	Yes	Sub	F8	F6	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

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

Scoreboard Example: Cycle 11

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	No								
8	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
0	Add	Yes	Sub	F8	F6	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

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

Scoreboard Example: Cycle 12

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	No								
7	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
	Add	No								
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

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

- **Read operands for DIVD?**

Scoreboard Example: Cycle 13

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13			

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	No								
6	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

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

Scoreboard Example: Cycle 14

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14		

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	No								
5	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
2	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

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

Scoreboard Example: Cycle 15

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14		

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	No								
4	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
1	Add	Yes	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

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

Scoreboard Example: Cycle 16

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14	16	

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	No								
3	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
0	Add	Yes	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

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

Scoreboard Example: Cycle 17

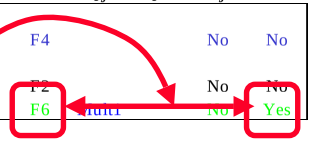
Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14	16	

WAR Hazard!

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	No								
2	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes



Register result status:

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

- **Why not write result of ADD???**

Scoreboard Example: Cycle 18

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14	16	

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	No								
1	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

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

Scoreboard Example: Cycle 19

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14	16	

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	No								
0	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

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

Scoreboard Example: Cycle 20

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14	16	

Functional unit status:

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

Register result status:

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

Scoreboard Example: Cycle 21

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21		
ADDD	F6	F8	F2	13	14	16	

Functional unit status:

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

Register result status:

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

- WAR Hazard is now gone...

Scoreboard Example: Cycle 22

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21		
ADDD	F6	F8	F2	13	14	16	22

Functional unit status:

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

Register result status:

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

Faster than light computation (skip a couple of cycles)

Scoreboard Example: Cycle 61

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21	61	
ADDD	F6	F8	F2	13	14	16	22

Functional unit status:

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

Register result status:

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

Scoreboard Example: Cycle 62

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21	61	62
ADDD	F6	F8	F2	13	14	16	22

Functional unit status:

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

Register result status:

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

Review: Scoreboard Example: Cycle 62

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21	61	62
ADDD	F6	F8	F2	13	14	16	22

Functional unit status:

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

Register result status:

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

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

CDC 6600 Scoreboard

- Historical context:
  - 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*)
  - No register renaming!
  - 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
- Precise interrupts????

Summary #1: Software Scheduling

- 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!
- Instruction Level Parallelism (ILP) found either by compiler or hardware.
- Loop level parallelism is easiest to see
  - SW dependencies/compiler sophistication determine if compiler can unroll loops
  - Memory dependencies hardest to determine => Memory disambiguation
  - Very sophisticated transformations available

Summary #2: Out-of-Order Execution

- HW exploiting ILP
  - Works when can't know dependence at compile 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 & data dependencies
  - Original version didn't handle forwarding.
  - No automatic register renaming
- Next Time: The Tomasulo algorithm
  - Automatic register renaming in hardware
  - Automatic loop unrolling in hardware
  - Ancestor of many major new microprocessors