Lecture 2:
Caches and Advanced Pipelining

Prof. Fred Chong
ECS 250A Computer Architecture
Winter 1999
(Adapted from Patterson-CS252 Copyright 1998 UCB)

Review, #1

• Designing to Last through Trends
  - Capacity
    - Logic: 2x in 5 years
    - DRAM: 4x in 3 years
    - Disk: 4x in 3 years
    - Processor: (n.a.)
  - Speed
    - 2x in 3 years
    - 2x in 10 years
  - Time to run the task
    - Execution time, response time, latency
  - Tasks per day, hour, week, sec, ns, ...
    - In Response to Trend - Throughput, bandwidth
  - "X is n times faster than Y" means
    \[
    \frac{\text{ExTime}(Y)}{\text{Performance}(X)} = \frac{\text{ExTime}(X)}{\text{Performance}(Y)}
    \]

Review, #2

• Amdahl's Law:
  \[
  \text{Speedup}_{\text{overall}} = \frac{\text{ExTime}_{\text{old}}}{\text{ExTime}_{\text{new}}} = 1 \cdot \left(1 - \frac{\text{Fraction}_{\text{enhanced}}}{\text{ExTime}_{\text{new}}} + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Performance}_{\text{Y}}}ight)
  \]

• CPI Law:

- CPU time = Seconds = Instructions \times Cycles \times Seconds
  - Program
  - CPU time
  \[
  \text{CPU time} = \text{Seconds} = \text{Instructions} \times \text{Cycles} \times \text{Seconds}
  \]

Recap: Who Cares About the Memory Hierarchy?

Processor-DRAM Memory Gap (latency)

- Processor: DRAM
  - 60%/yr.
  - (2X/1.5yr)
- DRAM: 9%/yr.
  - (2X/10 yrs)

Recap: Who Cares About the Memory Hierarchy?

Levels of the Memory Hierarchy

- Registers
- Instruction Operands
- Cache Blocks
- Main Memory Pages
- Disk Files
- Tape

The Principle of Locality

- The Principle of Locality:
  - Program access a relatively small portion of the address space at any instant of time.
- Two Different Types of Locality:
  - **Temporal Locality** (Locality in Time): If an item is referenced, it will tend to be referenced again soon (e.g., loops, reuse)
  - **Spatial Locality** (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon (e.g., straightline code, array access)
- Last 15 years, HW relied on locality for speed
Memory Hierarchy: Terminology
- **Hit**: data appears in some block in the upper level (example: Block X)
  - **Hit Rate**: the fraction of memory access found in the upper level
  - **Hit Time**: Time to access the upper level which consists of RAM access time + Time to determine hit/miss
- **Miss**: data needs to be retrieve from a block in the lower level (Block Y)
  - **Miss Rate**: 1 - Hit Rate
  - **Miss Penalty**: Time to replace a block in the upper level + Time to deliver the block the processor
- **Hit Time << Miss Penalty** (500 instructions on 21264!)

Cache Measures
- **Hit rate**: fraction found in that level
  - So high that usually talk about **Miss rate**
  - Miss rate fallacy: as MIPS to CPU performance, miss rate to average memory access time in memory
- **Average memory-access time** = Hit time + Miss rate x Miss penalty (ns or clocks)

Simplest Cache: Direct Mapped
- 1 KB Direct Mapped Cache, 32B blocks
  - For a $2^M$ byte cache:
    - The uppermost (32 - N) bits are always the Cache Tag
    - The lowest M bits are the Byte Select (Block Size = $2^M$)

Two-way Set Associative Cache
- N-way set associative: N entries for each Cache Index
  - Example: Two-way set associative cache
    - Cache Index selects a "set" from the cache
    - The two tags in the set are compared in parallel
    - Data is selected based on the tag result

Disadvantage of Set Associative Cache
- N-way Set Associative Cache v. Direct Mapped Cache:
  - N comparators vs. 1
  - Extra MUX delay for the data
  - Data comes AFTER Hit/Miss
- In a direct mapped cache, Cache Block is available BEFORE Hit/Miss:
  - Possible to assume a hit and continue. Recover later if miss.
4 Questions for Memory Hierarchy

Q1: Where can a block be placed in the upper level? (Block placement)
- direct mapped - 1 place
- n-way set associative - n places
- fully-associative - any place

Q2: How is a block found if it is in the upper level? (Block identification)
- Tag on each block
  - No need to check index or block offset
- Increasing associativity shrinks index, expands tag

Q3: Which block should be replaced on a miss? (Block replacement)
- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - Random
    - Associativity: 2-way 4-way 8-way
    - Size LRURandom LRURandom LRURandom
  - Size:
    - 16 KB 5.2% 5.7% 4.7% 5.3% 4.4% 5.0%
    - 64 KB 1.9% 2.0% 1.5% 1.7% 1.4% 1.5%
    - 256 KB 1.15% 1.17% 1.13% 1.13% 1.12% 1.12%

Q4: What happens on a write? (Write strategy)
- Write through—The information is written to both the block in the cache and to the block in the lower-level memory.
- Write back—The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced.
- Is block clean or dirty?
- Pros and Cons of each?
  - WT: read misses cannot result in writes
  - WB: no repeated writes to same location
- WT always combined with write buffers so that don’t wait for lower level memory

Write Buffer for Write Through

- A Write Buffer is needed between the Cache and Memory
  - Processor: writes data into the cache and the write buffer
  - Memory controller: write contents of the buffer to memory
- Write buffer is just a FIFO:
  - Typical number of entries: 4
  - Works fine if: Store frequency (w.r.t. time) << 1 / DRAM write cycle
- Memory system designer’s nightmare:
  - Store frequency (w.r.t. time) -> 1 / DRAM write cycle
  - Write buffer saturation

Block Address
Tag
Index
Block offset

Write through
Write back

Impact of Memory Hierarchy on Algorithms

• Today CPU time is a function of (ops, cache misses) vs. just f(ops):
  - What does this mean to Compilers, Data structures, Algorithms?
  - Quicksort: fastest comparison based sorting algorithm when all keys fit in memory
  - Radix sort: also called “linear time” sort because for keys of fixed length and fixed radix a constant number of passes over the data is sufficient independent of the number of keys
  - For Alphastation 250, 32 byte blocks, direct mapped L2 2MB cache, 8 byte keys, from 4000 to 4000000

Quicksort vs. Radix as vary number keys: Instructions

Quicksort vs. Radix as vary number keys: Cache misses

A Modern Memory Hierarchy

• By taking advantage of the principle of locality:
  - Present the user with as much memory as is available in the cheapest technology
  - Provide access at the speed offered by the fastest technology

Basic Issues in VM System Design

- size of information blocks that are transferred from secondary to main storage (M)
- block of information brought into M, and M is full, then some region of M must be released to make room for the new block
  - replacement policy
  - which region of M is to hold the new block
  - missing item fetched from secondary memory only on the occurrence of a fault
  - demand load policy

Page Organization

virtual and physical address space partitioned into blocks of equal size
Address Map

\[ V = \{0, 1, \ldots, n-1\} \text{ virtual address space} \]
\[ M = \{0, 1, \ldots, m-1\} \text{ physical address space} \]

MAP: \( V \rightarrow M \cup \{0\} \) address mapping function

\[ MAP(a) = a' \text{ if data at virtual address } a \text{ is present in physical address } a' \text{ and } a' \in M \]

\[ = 0 \text{ if data at virtual address } a \text{ is not present in } M \]

Processor Name Space V
Addr Trans Mechanism
Fault handler
Main Memory
Secondary Memory

Physical address

Virtual address

OS performs this transfer

Paging Organization

Virtual Address and a Cache

TLBs

Translation Look-Aside Buffers

Reducing Translation Time
Overlapped Cache & TLB Access

![Diagram showing Overlapped Cache & TLB Access]

Problems With Overlapped TLB Access

Overlapped access only works as long as the address bits used to index into the cache do not change as the result of VA translation.

Example: suppose everything the same except that the cache is increased to 8 K bytes instead of 4 K:

![Diagram showing Problems With Overlapped TLB Access]

Summary #1/4:

- **The Principle of Locality:**
  - Program access a relatively small portion of the address space at any instant of time.
  - Temporal Locality: Locality in Time
  - Spatial Locality: Locality in Space

- **Three Major Categories of Cache Misses:**
  - Compulsory Misses: sad facts of life. Example: cold start misses.
  - Capacity Misses: increase cache size
  - Conflict Misses: increase cache size and/or associativity.

- **Nightmare Scenario:** ping pong effect!

- **Write Policy:**
  - Write Through: needs a write buffer. Nightmare: WB saturation
  - Write Back: control can be complex

Summary #2 / 4: The Cache Design Space

- Several interacting dimensions
  - cache size
  - block size
  - associativity
  - replacement policy
  - write-through vs write-back
- The optimal choice is a compromise
  - depends on access characteristics
  - workload
  - use (L-cache, D-cache, TLB)
- Simplicity often wins

Summary #3/4: TLB, Virtual Memory

- Caches, TLBs, Virtual Memory all understood by examining how they deal with 4 questions: 1) Where can block be placed? 2) How is block found? 3) What block is repalced on miss? 4) How are writes handled?
- Page tables map virtual address to physical address
- TLBs are important for fast translation
- TLB misses are significant in processor performance
- How block is replaced on miss?
- How are writes handled?
- Can SW automatically manage 64KB across many programs?
- 100X DRAM growth removed the controversy
- Today VM allows many processes to share single memory without swapping all processes to disk; today VM protection is more important than memory hierarchy
- Page tables map virtual address to physical address
- TLB misses are significant in processor performance
- Today CPU time is a function of (ops, cache misses) vs. just f(ops):
  - What does this mean to Compilers, Data structures, Algorithms?
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

<table>
<thead>
<tr>
<th>TWO Cycle</th>
<th>IF</th>
<th>IS</th>
<th>RF</th>
<th>EX</th>
<th>DF</th>
<th>TC</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Latency</td>
<td>IF</td>
<td>IS</td>
<td>RF</td>
<td>EX</td>
<td>DF</td>
<td>DS</td>
<td>TC</td>
</tr>
<tr>
<td>IF</td>
<td>IS</td>
<td>RF</td>
<td>EX</td>
<td>DF</td>
<td>DS</td>
<td>TC</td>
<td>WB</td>
</tr>
<tr>
<td>IF</td>
<td>IS</td>
<td>RF</td>
<td>EX</td>
<td>DS</td>
<td>TC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF</td>
<td>IS</td>
<td>RF</td>
<td>EX</td>
<td>DF</td>
<td>DS</td>
<td>TC</td>
<td></td>
</tr>
<tr>
<td>IF</td>
<td>IS</td>
<td>RF</td>
<td>EX</td>
<td>DF</td>
<td>DS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF</td>
<td>IS</td>
<td>RF</td>
<td>EX</td>
<td>DF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF</td>
<td>IS</td>
<td>RF</td>
<td>EX</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF</td>
<td>IS</td>
<td>RF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| THREE Cycle | IF | IS | RF | DFS | TC | WB |
| Branch Latency | IF | IS | RF | DFS | TC | WB |
| IF | IS | RF | DFS |
| IF | IS | RF | DFS |
| IF | IS | RF | DFS |
| IF | IS | RF |
| IF | IS | RF |
| IF | IS | RF |
| IF | IS | RF |

- **R4000 Performance**
  - Not ideal CPI of 1:
    - Load stalls (1 or 2 clock cycles)
    - Branch stalls (2 cycles + unifield slots)
    - FP result stalls: RAW data hazard (latency)
    - FP structural stalls: Not enough FP hardware (parallelism)

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

<table>
<thead>
<tr>
<th>FP Instr</th>
<th>1 2 3 4 5 6 7 8 ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add, Subtract</td>
<td>U S=A A+R R+S</td>
</tr>
<tr>
<td>Multiply</td>
<td>U E M M M M N= A R</td>
</tr>
<tr>
<td>Divide</td>
<td>U A R D+ ... D+A D=A, D=R, D=A, D=R, A, R</td>
</tr>
<tr>
<td>Square root</td>
<td>U E (A=W)? ... A R</td>
</tr>
<tr>
<td>Negate</td>
<td>U S</td>
</tr>
<tr>
<td>Absolute value</td>
<td>U S</td>
</tr>
<tr>
<td>FP compare</td>
<td>U A R</td>
</tr>
</tbody>
</table>

MIPS FP Pipe Stages

- **Stages:**
  - M First stage of multiplier
  - N Second stage of multiplier
  - R Rounding stage
  - S Operand shift stage
  - U Unpack FP numbers

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
SUBI R1,R1,8 ;decrement pointer 8B (DW)
BNEZ R1,Loop ;branch R1!=zero
NOP ;delayed branch slot

Instruction Instruction Latency in producing result using result 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 FI0(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 Instruction Latency in producing result using result 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

• 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 R1,R4 altered when move past SUBI

Swap BNEZ and SD by changing address of SD

Unroll Loop Four Times (straightforward way)

Rewrite loop to minimize stalls?

1 Loop: LD F0,0(R1)
2 ADDD F4,F0,F2
3 SD 0(R1),F4 ;drop SUBI & BNEZ
4 ADDD F4,F0,F2
5 ADDD F4,F0,F2
6 ADDD F4,F0,F2
7 ADDD F4,F0,F2
8 ADDD F4,F0,F2
9 ADDD F4,F0,F2
10 ADDD F4,F0,F2
11 ADDD F4,F0,F2
12 ADDD F4,F0,F2
13 SUBI R1,R1,8 altered to 4'0
14 BNEZ R1,_LOOP ;alter when move past SUBI
15 NOP

6 clocks: Unroll loop 4 times code to make faster? FTC.W99 60

15 x 4 x (1+2) = 27 clock cycles, or 6.8 per iteration
Assumes R1 is multiple of 4

Unrolled Loop That Minimizes Stalls

• What assumptions made when moved code?

1 Loop: LD F0,0(R1)
2 ADDD F4,F0,F2
3 ADDD F4,F0,F2
4 ADDD F4,F0,F2
5 ADDD F4,F0,F2
6 ADDD F4,F0,F2
7 ADDD F4,F0,F2
8 ADDD F4,F0,F2
9 ADDD F4,F0,F2
10 ADDD F4,F0,F2
11 ADDD F4,F0,F2
12 SUBI R1,R1,8 altered to 4'0
13 BNEZ R1,_LOOP
14 SD 0(R1),F4 ;drop SUBI & BNEZ

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 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)?

Where are the data dependencies?

1 Loop: LD F0,0(R1)
2 ADD F4,F0,F2
3 ADD F4,F0,F2
4 BNEZ R1,Loop :delayed branch
5 SD R(R1),F4 :alter 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 ADD F4,F0,F2
3 ADD F4,F0,F2
4 BNEZ R1,Loop :drop SUBI & BNEZ
5 SD R(R1),F4 :drop SUBI & BNEZ
6 ADD F4,F0,F2
7 ADD F4,F0,F2
8 ADD F4,F0,F2
9 ADD F4,F0,F2
10 LD F6, SUB(R1)
11 ADD F6,F0,F2
12 ADD F6,F0,F2
13 SUBI R1,R1,16 :alter to 4*8
14 BNEZ R1,Loop :alter to 4*8
15 NOP

How can remove them?

Called “register renaming”

Where are the name dependencies?

1 Loop: LD F0,0(R1)
2 ADD F4,F0,F2
3 ADD F4,F0,F2
4 ADD F4,F0,F2
5 ADD F4,F0,F2
6 ADD F4,F0,F2
7 ADD F4,F0,F2
8 ADD F4,F0,F2
9 ADD F4,F0,F2
10 ADD F4,F0,F2
11 ADD F4,F0,F2
12 ADD F4,F0,F2
13 SUBI R1,R1,16 :alter to 4*8
14 BNEZ R1,Loop :alter to 4*8
15 NOP

Compiler Perspectives on Code Movement

• Again Name Dependences 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) -8(R1) -16(R1) -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.

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)
   ADD F4,F0,F2
   SD 0(R1),F4
   SUBI R1,R1,8
   BEQZ R1,exit
   LD F0,0(R1)
   ADD F4,F0,F2
   SD 0(R1),F4
   SUBI R1,R1,8
   BEQZ R1,exit
   LD F0,0(R1)

2. 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

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];
  }

  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
  ADD F10,F0,F8
  SUBI F12,F8,F14

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

- 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
  3. Scoreboards allow instruction to execute whenever
     1 & 2 hold, not waiting for prior instructions
  4. 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, it stalls the instruction.

Example:

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

Detailed Scoreboard Pipeline Control

<table>
<thead>
<tr>
<th>Instruction status</th>
<th>Wall unit</th>
<th>Bookkeeping</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issue</td>
<td>Busy(FU) = yes, O(FU) = op, F(FU) = S1; F(FU) = S1; F(FU) = S1; Result(F1); Result(F2); Result(F3); F(FU) = FU; F(FU) = FU; F(FU) = FU</td>
<td></td>
</tr>
<tr>
<td>Read operands</td>
<td>Rj and Fk</td>
<td>Rj = No; Rk = No</td>
</tr>
<tr>
<td>Execution complete</td>
<td>Functional unit done</td>
<td></td>
</tr>
<tr>
<td>Write result</td>
<td>Result(Fi(FU)) = 0; Busy(FU) = No</td>
<td></td>
</tr>
</tbody>
</table>

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 -)
   - Fj—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

Scoreboard Example
### Scoreboard Example Cycle 1

<table>
<thead>
<tr>
<th>Instruction name</th>
<th>Read</th>
<th>Execute/Write</th>
<th>Functional unit status</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD F6 R2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD F3 R1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULT F2 F4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIVD F10 F0 F6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDI F10 F2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instruction status</th>
<th>Issue</th>
<th>operands complete</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Functional unit status</th>
<th>dest</th>
<th>S1</th>
<th>S2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time</th>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Fi</th>
<th>Fj?</th>
<th>Fk?</th>
<th>Qj</th>
<th>Qk</th>
<th>Rj</th>
<th>Rk</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Register result status</th>
<th>Clock</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Scoreboard Example Cycle 2

<table>
<thead>
<tr>
<th>Instruction name</th>
<th>Read</th>
<th>Execute/Write</th>
<th>Functional unit status</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD F6 R2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD F3 R1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULT F2 F4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIVD F10 F0 F6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDI F10 F2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instruction status</th>
<th>Issue</th>
<th>operands complete</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Functional unit status</th>
<th>dest</th>
<th>S1</th>
<th>S2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time</th>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Fi</th>
<th>Fj?</th>
<th>Fk?</th>
<th>Qj</th>
<th>Qk</th>
<th>Rj</th>
<th>Rk</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Register result status</th>
<th>Clock</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Scoreboard Example Cycle 3

<table>
<thead>
<tr>
<th>Instruction name</th>
<th>Read</th>
<th>Execute/Write</th>
<th>Functional unit status</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD F6 R2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD F3 R1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULT F2 F4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIVD F10 F0 F6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDI F10 F2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instruction status</th>
<th>Issue</th>
<th>operands complete</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Functional unit status</th>
<th>dest</th>
<th>S1</th>
<th>S2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time</th>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Fi</th>
<th>Fj?</th>
<th>Fk?</th>
<th>Qj</th>
<th>Qk</th>
<th>Rj</th>
<th>Rk</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Register result status</th>
<th>Clock</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Scoreboard Example Cycle 4

<table>
<thead>
<tr>
<th>Instruction name</th>
<th>Read</th>
<th>Execute/Write</th>
<th>Functional unit status</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD F6 R2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD F3 R1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULT F2 F4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIVD F10 F0 F6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDI F10 F2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instruction status</th>
<th>Issue</th>
<th>operands complete</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Functional unit status</th>
<th>dest</th>
<th>S1</th>
<th>S2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time</th>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Fi</th>
<th>Fj?</th>
<th>Fk?</th>
<th>Qj</th>
<th>Qk</th>
<th>Rj</th>
<th>Rk</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Register result status</th>
<th>Clock</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Scoreboard Example Cycle 5

<table>
<thead>
<tr>
<th>Instruction name</th>
<th>Read</th>
<th>Execute/Write</th>
<th>Functional unit status</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD F6 R2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD F3 R1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULT F2 F4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIVD F10 F0 F6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDI F10 F2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instruction status</th>
<th>Issue</th>
<th>operands complete</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Functional unit status</th>
<th>dest</th>
<th>S1</th>
<th>S2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time</th>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Fi</th>
<th>Fj?</th>
<th>Fk?</th>
<th>Qj</th>
<th>Qk</th>
<th>Rj</th>
<th>Rk</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Register result status</th>
<th>Clock</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Scoreboard Example Cycle 6

<table>
<thead>
<tr>
<th>Instruction name</th>
<th>Read</th>
<th>Execute/Write</th>
<th>Functional unit status</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD F6 R2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD F3 R1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULT F2 F4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIVD F10 F0 F6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDI F10 F2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instruction status</th>
<th>Issue</th>
<th>operands complete</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Functional unit status</th>
<th>dest</th>
<th>S1</th>
<th>S2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time</th>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Fi</th>
<th>Fj?</th>
<th>Fk?</th>
<th>Qj</th>
<th>Qk</th>
<th>Rj</th>
<th>Rk</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Register result status</th>
<th>Clock</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**Issue 2nd LD?**

---

**Issue MULT?**
### Scoreboard Example Cycle 7

**Instruction status**
- Read
- Execution
- Write

**Instruction**
- `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**
- `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`
- `Div Yes Div F10 F0 F6 Mult1 No Yes`

**Register result status**
- `Clock F0 F2 F4 F6 F8 F10 F12 ... F30`

- `FU Mult1`
- `Add`
- `Divide`

- **Observations:**
  - Read multiply operands?

---

### Scoreboard Example Cycle 8a

**Instruction status**
- Read
- Execution
- Write

**Instruction**
- `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**
- `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 Yes Div F10 F0 F6 Mult1 No Yes`

**Register result status**
- `Clock F0 F2 F4 F6 F8 F10 F12 ... F30`

- `FU Mult1`
- `Add`
- `Divide`

- **Observations:**
  - Read operands for MULT & SUBD?
  - Issue ADDD?

---

### Scoreboard Example Cycle 8b

**Instruction status**
- Read
- Execution
- Write

**Instruction**
- `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 9`
- `DIVD F10 F0 F6 8`
- `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 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`

- `FU Mult1`
- `Add`

- **Observations:**
  - Read operands for DIVD?
### Scoreboard Example Cycle 13

#### Instruction status

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Read</th>
<th>Execute/Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD F6, 34+ R2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LD F2, 45+ R3</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MULTD F0, F2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SUBD F8, F6</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DIVD F10, F0</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ADDD F6, F8</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

#### Functional unit status

<table>
<thead>
<tr>
<th>Time</th>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Fi</th>
<th>Fj</th>
<th>Fk</th>
<th>Qj</th>
<th>Qk</th>
<th>Rj</th>
<th>Rk</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Mult1</td>
<td>Yes</td>
<td>Mult</td>
<td>F0</td>
<td>F2</td>
<td>F4</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mult2</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Add</td>
<td>Yes</td>
<td>Add</td>
<td>F6</td>
<td>F8</td>
<td>F2</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Divide</td>
<td>Yes</td>
<td>Divide</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Register result status

<table>
<thead>
<tr>
<th>Clock</th>
<th>FU</th>
<th>Mult1</th>
<th>Add</th>
<th>Divide</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

### Scoreboard Example Cycle 14

#### Instruction status

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Read</th>
<th>Execute/Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD F6, 34+ R2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LD F2, 45+ R3</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MULTD F0, F2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SUBD F8, F6</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DIVD F10, F0</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ADDD F6, F8</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

#### Functional unit status

<table>
<thead>
<tr>
<th>Time</th>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Fi</th>
<th>Fj</th>
<th>Fk</th>
<th>Qj</th>
<th>Qk</th>
<th>Rj</th>
<th>Rk</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Mult1</td>
<td>Yes</td>
<td>Mult</td>
<td>F0</td>
<td>F2</td>
<td>F4</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mult2</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Add</td>
<td>Yes</td>
<td>Add</td>
<td>F6</td>
<td>F8</td>
<td>F2</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Divide</td>
<td>Yes</td>
<td>Divide</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Register result status

<table>
<thead>
<tr>
<th>Clock</th>
<th>FU</th>
<th>Mult1</th>
<th>Add</th>
<th>Divide</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

### Scoreboard Example Cycle 15

#### Instruction status

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Read</th>
<th>Execute/Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD F6, 34+ R2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LD F2, 45+ R3</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MULTD F0, F2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SUBD F8, F6</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DIVD F10, F0</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ADDD F6, F8</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

#### Functional unit status

<table>
<thead>
<tr>
<th>Time</th>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Fi</th>
<th>Fj</th>
<th>Fk</th>
<th>Qj</th>
<th>Qk</th>
<th>Rj</th>
<th>Rk</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Mult1</td>
<td>Yes</td>
<td>Mult</td>
<td>F0</td>
<td>F2</td>
<td>F4</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mult2</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Add</td>
<td>Yes</td>
<td>Add</td>
<td>F6</td>
<td>F8</td>
<td>F2</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Divide</td>
<td>Yes</td>
<td>Divide</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Register result status

<table>
<thead>
<tr>
<th>Clock</th>
<th>FU</th>
<th>Mult1</th>
<th>Add</th>
<th>Divide</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

### Scoreboard Example Cycle 16

#### Instruction status

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Read</th>
<th>Execute/Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD F6, 34+ R2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LD F2, 45+ R3</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MULTD F0, F2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SUBD F8, F6</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DIVD F10, F0</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ADDD F6, F8</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

#### Functional unit status

<table>
<thead>
<tr>
<th>Time</th>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Fi</th>
<th>Fj</th>
<th>Fk</th>
<th>Qj</th>
<th>Qk</th>
<th>Rj</th>
<th>Rk</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Mult1</td>
<td>Yes</td>
<td>Mult</td>
<td>F0</td>
<td>F2</td>
<td>F4</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mult2</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Add</td>
<td>Yes</td>
<td>Add</td>
<td>F6</td>
<td>F8</td>
<td>F2</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Divide</td>
<td>Yes</td>
<td>Divide</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Register result status

<table>
<thead>
<tr>
<th>Clock</th>
<th>FU</th>
<th>Mult1</th>
<th>Add</th>
<th>Divide</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

### Scoreboard Example Cycle 17

#### Instruction status

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Read</th>
<th>Execute/Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD F6, 34+ R2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LD F2, 45+ R3</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MULTD F0, F2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SUBD F8, F6</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DIVD F10, F0</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ADDD F6, F8</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

#### Functional unit status

<table>
<thead>
<tr>
<th>Time</th>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Fi</th>
<th>Fj</th>
<th>Fk</th>
<th>Qj</th>
<th>Qk</th>
<th>Rj</th>
<th>Rk</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Mult1</td>
<td>Yes</td>
<td>Mult</td>
<td>F0</td>
<td>F2</td>
<td>F4</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mult2</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Add</td>
<td>Yes</td>
<td>Add</td>
<td>F6</td>
<td>F8</td>
<td>F2</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Divide</td>
<td>Yes</td>
<td>Divide</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Register result status

<table>
<thead>
<tr>
<th>Clock</th>
<th>FU</th>
<th>Mult1</th>
<th>Add</th>
<th>Divide</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

### Scoreboard Example Cycle 18

#### Instruction status

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Read</th>
<th>Execute/Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD F6, 34+ R2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LD F2, 45+ R3</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MULTD F0, F2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SUBD F8, F6</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DIVD F10, F0</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ADDD F6, F8</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

#### Functional unit status

<table>
<thead>
<tr>
<th>Time</th>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Fi</th>
<th>Fj</th>
<th>Fk</th>
<th>Qj</th>
<th>Qk</th>
<th>Rj</th>
<th>Rk</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mult1</td>
<td>Yes</td>
<td>Mult</td>
<td>F0</td>
<td>F2</td>
<td>F4</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mult2</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Add</td>
<td>Yes</td>
<td>Add</td>
<td>F6</td>
<td>F8</td>
<td>F2</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Divide</td>
<td>Yes</td>
<td>Divide</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Register result status

<table>
<thead>
<tr>
<th>Clock</th>
<th>FU</th>
<th>Mult1</th>
<th>Add</th>
<th>Divide</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Scoreboard Example Cycle 19

**Instruction status**
- **Read**
- **Execution**
- **Write**

**Instruction**
- `j k Issue`
- **operands**
- **complete**

**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**
- **dest**
- **S1**
- **S2**
- **FU for j**
- **FU for k**
- **Fj?**
- **Fk?**

**Register result status**
- **Clock**
- **FU**

---

### Scoreboard Example Cycle 20

**Instruction status**
- **Read**
- **Execution**
- **Write**

**Instruction**
- `j k Issue`
- **operands**
- **complete**

**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**
- **dest**
- **S1**
- **S2**
- **FU for j**
- **FU for k**
- **Fj?**
- **Fk?**

**Register result status**
- **Clock**
- **FU**

---

### Scoreboard Example Cycle 21

**Instruction status**
- **Read**
- **Execution**
- **Write**

**Instruction**
- `j k Issue`
- **operands**
- **complete**

**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**
- **dest**
- **S1**
- **S2**
- **FU for j**
- **FU for k**
- **Fj?**
- **Fk?**

**Register result status**
- **Clock**
- **FU**

---

### Scoreboard Example Cycle 22

**Instruction status**
- **Read**
- **Execution**
- **Write**

**Instruction**
- `j k Issue`
- **operands**
- **complete**

**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**
- **dest**
- **S1**
- **S2**
- **FU for j**
- **FU for k**
- **Fj?**
- **Fk?**

**Register result status**
- **Clock**
- **FU**

---

### Scoreboard Example Cycle 61

**Instruction status**
- **Read**
- **Execution**
- **Write**

**Instruction**
- `j k Issue`
- **operands**
- **complete**

**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**
- **dest**
- **S1**
- **S2**
- **FU for j**
- **FU for k**
- **Fj?**
- **Fk?**

**Register result status**
- **Clock**
- **FU**

---

### Scoreboard Example Cycle 62

**Instruction status**
- **Read**
- **Execution**
- **Write**

**Instruction**
- `j k Issue`
- **operands**
- **complete**

**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**
- **dest**
- **S1**
- **S2**
- **FU for j**
- **FU for k**
- **Fj?**
- **Fk?**

**Register result status**
- **Clock**
- **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), especially 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
- 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