Lecture 2:
Caches and Advanced Pipelining

Prof. Fred Chong
ECS 250A Computer Architecture
Winter 1999

(Adapted from Patterson CS252 Copyright 1998 UCB)

FTC.W99 1

Review, #1

- Designing to Last through Trends

	Capacity	Speed
Logic	2x in 3 years	2x in 3 years
DRAM	4x in 3 years	2x in 10 years
Disk	4x in 3 years	2x in 10 years
Processor	(n.a.)	2x in 1.5 years

- Time to run the task
  - Execution time, response time, latency
- Tasks per day, hour, week, sec, ns, …
  - Throughput, bandwidth
- "X is n times faster than Y" means

$$\frac{ExTime(Y)}{ExTime(X)} = \frac{Performance(X)}{Performance(Y)}$$

FTC.W99 2

Review, #2

- Amdahl's Law:

$$Speedup_{overall} = \frac{ExTime_{old}}{ExTime_{new}} = \frac{1}{(1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

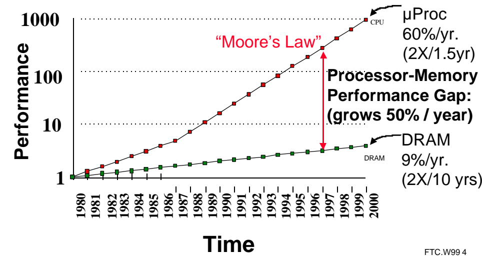
- CPI Law:

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

- Execution time is the REAL measure of computer performance!
- Good products created when have:
  - Good benchmarks
  - Good ways to summarize performance
- Die Cost goes roughly with die area^4

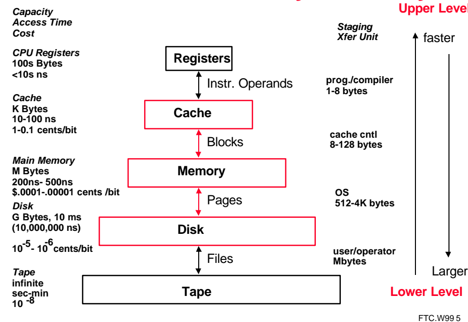
FTC.W99 3

Recap: Who Cares About the Memory Hierarchy?



FTC.W99 4

Levels of the Memory Hierarchy



FTC.W99 5

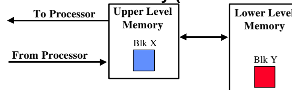
The Principle of Locality

- 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 localilty for speed

FTC.W99 6

Memory Hierarchy: Terminology

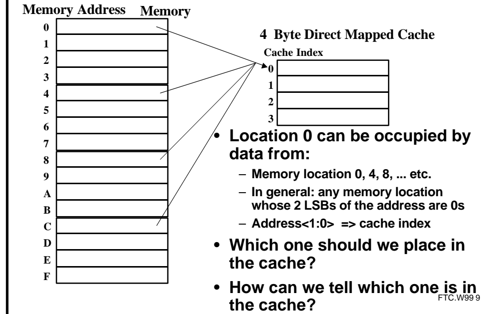
- Hit: data appears in some block in the upper level (example: Block X)
  - Hit Rate: the fraction of memory access found in the upper level
  - Hit Time: Time to access the upper level which consists of RAM access time + Time to determine hit/miss
- Miss: data needs to be retrieve from a block in the 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)
- *Miss penalty*: time to replace a block from lower level, including time to replace in CPU
  - *access time*: time to lower level = f(latency to lower level)
  - *transfer time*: time to transfer block =f(BW between upper & lower levels)

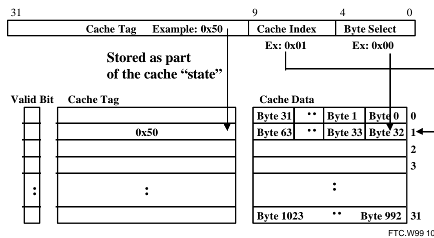
Simplest Cache: Direct Mapped



- Location 0 can be occupied by data from:
  - Memory location 0, 4, 8, ... etc.
  - In general: any memory location whose 2 LSBs of the address are 0s
  - Address<1:0> => cache index
- Which one should we place in the cache?
- How can we tell which one is in the cache?

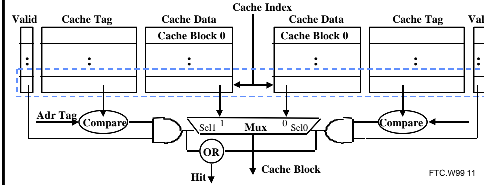
1 KB Direct Mapped Cache, 32B blocks

- For a 2 ** N byte cache:
  - The uppermost (32 - N) bits are always the Cache Tag
  - The lowest M bits are the Byte Select (Block Size = 2 ** M)



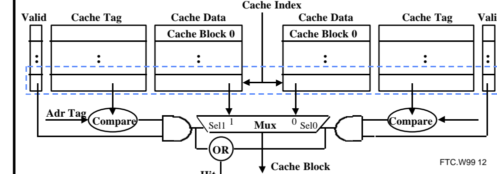
Two-way Set Associative Cache

- N-way set associative: N entries for each Cache Index
  - N direct mapped caches operates in parallel (N typically 2 to 4)
- 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

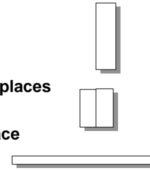
- N-way Set Associative Cache v. Direct Mapped Cache:
  - N comparators vs. 1
  - Extra MUX delay for the data
  - Data comes AFTER Hit/Miss
- In a direct mapped cache, Cache Block is available BEFORE Hit/Miss:
  - Possible to assume a hit and continue. Recover later if miss.



4 Questions for Memory Hierarchy

- Q1: Where can a block be placed in the upper level? *(Block placement)*
- Q2: How is a block found if it is in the upper level? *(Block identification)*
- Q3: Which block should be replaced on a miss? *(Block replacement)*
- Q4: What happens on a write? *(Write strategy)*

Q1: Where can a block be placed in the upper level?



- 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?

- 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?

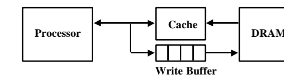
- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - Random
  - LRU (Least Recently Used)

Associativity:	2-way		4-way		8-way	
Size	LRU	Random	LRU	Random	LRU	Random
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 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

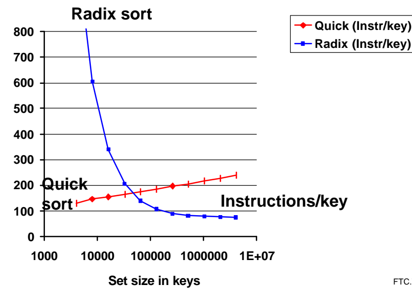


- A Write Buffer is needed between the Cache and Memory
  - Processor: writes data into the cache and the write buffer
  - Memory controller: write contents of the buffer to memory
- Write buffer is just a FIFO:
  - Typical number of entries: 4
  - Works fine if: Store frequency (w.r.t. time) << 1 /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

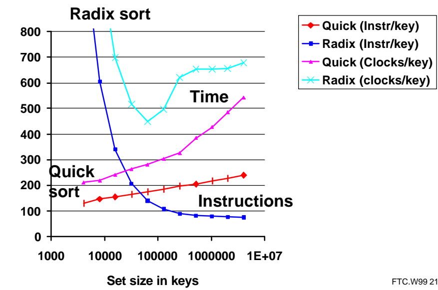
Impact of Memory Hierarchy on Algorithms

- Today CPU time is a function of (ops, cache misses) vs. just f(ops):
  What does this mean to Compilers, Data structures, Algorithms?
- "The Influence of Caches on the Performance of Sorting" by A. LaMarca and R.E. Ladner. *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, January, 1997, 370-379.
- 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

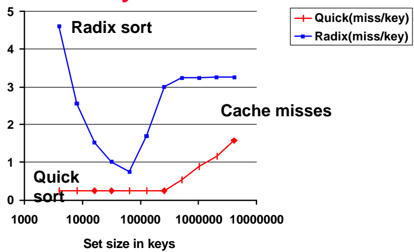
Quicksort vs. Radix as vary number keys: Instructions



Quicksort vs. Radix as vary number keys: Instrs & Time



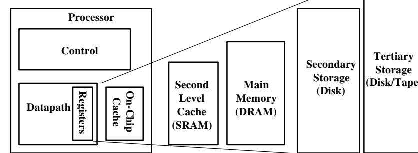
Quicksort vs. Radix as vary number keys: Cache misses



What is proper approach to fast algorithms?

A Modern Memory Hierarchy

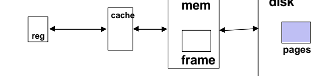
- By taking advantage of the principle of locality:
  - Present the user with as much memory as is available in the cheapest technology.
  - Provide access at the speed offered by the fastest technology.



Speed (ns):	1s	10s	100s	10,000,000s (10s ms)	10,000,000,000s (10s sec)
Size (bytes):	100s	Ks	Ms	Gs	Ts

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 --> *placement policy*
- missing item fetched from secondary memory only on the occurrence of a fault --> *demand load policy*



Paging Organization

virtual and physical address space partitioned into blocks of equal size

page frames

pages

Address Map

V = {0, 1, . . . , n - 1} virtual address space
M = {0, 1, . . . , m - 1} physical address space

n > m

MAP: V --> M U {∅} address mapping function

MAP(a) = a' if data at virtual address a is present in physical address a' and a' in M

= ∅ if data at virtual address a is not present in M

Paging Organization

Virtual Address and a Cache

It takes an extra memory access to translate VA to PA

This makes cache access very expensive, and this is the "innermost loop" that you want to go as fast as possible

ASIDE: Why access cache with PA at all? VA caches have a problem!

synonym / alias problem: two different virtual addresses map to same physical address => two different cache entries holding data for the same physical address!

for update: must update all cache entries with same physical address or memory becomes inconsistent

determining this requires significant hardware, essentially an associative lookup on the physical address tags to see if you have multiple hits; or

software enforced alias boundary: same lsb of VA &PA > cache size

TLBs

A way to speed up translation is to use a special cache of recently used page table entries -- this has many names, but the most frequently used is ***Translation Lookaside Buffer or TLB***

Virtual Address	Physical Address	Dirty	Ref	Valid	Access

Really just a cache on the page table mappings

TLB access time comparable to cache access time (much less than main memory access time)

Translation Look-Aside Buffers

Just like any other cache, the TLB can be organized as fully associative, set associative, or direct mapped

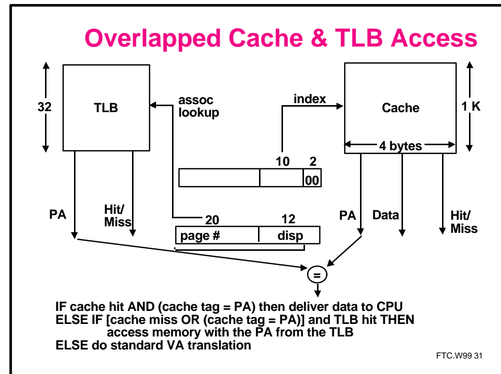
TLBs are usually small, typically not more than 128 - 256 entries even on high end machines. This permits fully associative lookup on these machines. Most mid-range machines use small n-way set associative organizations.

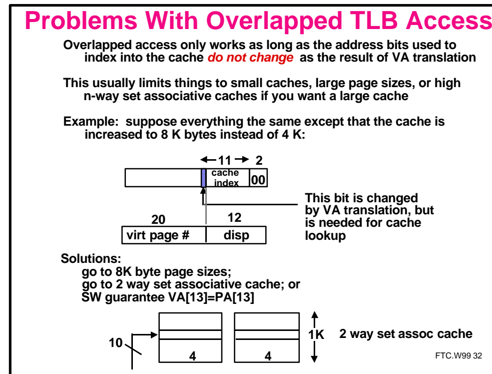
Reducing Translation Time

Machines with TLBs go one step further to reduce # cycles/cache access

They overlap the cache access with the TLB access:

high order bits of the VA are used to look in the TLB while low order bits are used as index into cache

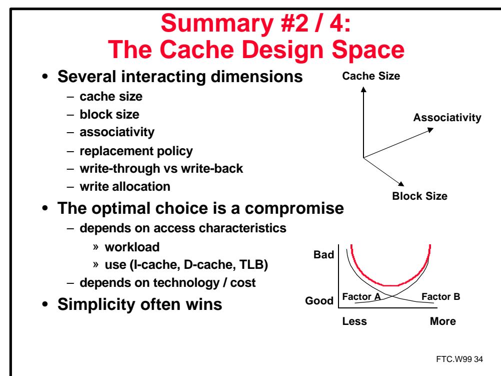




Summary #1/4:

- **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

FTC.W99 33



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
  - funny times, as most systems can't access all of 2nd level cache without TLB misses!

FTC.W99 35

Summary #4/4: Memory Hierachy

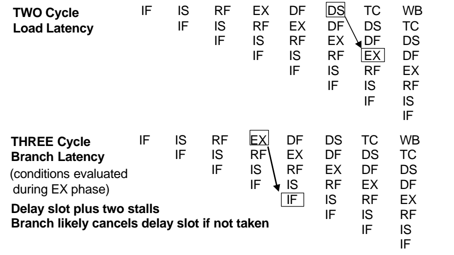
- VIrtual memory was controversial at the time: can SW automatically manage 64KB across many programs?
  - 1000X DRAM growth removed the controversy
- Today VM allows many processes to share single memory without having to swap all processes to disk; today VM protection is more important than memory hierarchy
- Today CPU time is a function of (ops, cache misses) vs. just f(ops):
  What does this mean to Compilers, Data structures, Algorithms?

FTC.W99 36

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



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

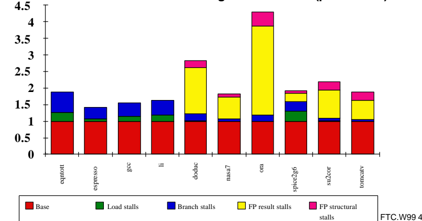
FP Instr	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*…*
Add, Subtract	U	S+A	A+R	R+S					
Multiply	U	E+M	M	M	M	N	N+A	R	
Divide	U	A	R	D^{28}	…	D+A	D+R, D+R, D+A, D+R, A, R		
Square root	U	E	$(A+R)^{108}$		…	A	R		
Negate	U	S							
Absolute value	U	S							
FP compare	U	A	R						

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?

FTC.W99 43

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

FTC.W99 44

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?

FTC.W99 45

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?

FTC.W99 46

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
```

Rewrite loop to minimize stalls?

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

FTC.W99 47

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
When safe to move instructions?

FTC.W99 48

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        ;drop SUBI & BNEZ
4       LD      F0,-8(R1)
2       ADDD    F4,F0,F2
3       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

- Again Name Dependenceis 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.

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 structuralScoreboard 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

Instruction		j	k	Issue	Read operand	Executio complet	Write 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

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								
	Divide	No								

Register result status

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

Scoreboard Example Cycle 1

Instruction status

Instruction		j	k	Issue	Read operands	Execution complete	Write 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

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

Register result status

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

FTC.W99 67

Scoreboard Example Cycle 2

Instruction status

Instruction		j	k	Issue	Read operands	Execution complete	Write 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

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

Register result status

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

- Issue 2nd LD?

FTC.W99 68

Scoreboard Example Cycle 3

Instruction status

Instruction		j	k	Issue	Read operands	Execution complete	Write 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

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

Register result status

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

- Issue MULT?

FTC.W99 69

Scoreboard Example Cycle 4

Instruction status

Instruction		j	k	Issue	Read operands	Execution complete	Write 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

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

Register result status

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

FTC.W99 70

Scoreboard Example Cycle 5

Instruction status

Instruction		j	k	Issue	Read operands	Execution complete	Write 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

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	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

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

FTC.W99 71

Scoreboard Example Cycle 6

Instruction status

Instruction		j	k	Issue	Read operands	Execution complete	Write 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

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	No								
	Divide	No								

Register result status

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

FTC.W99 72

Scoreboard Example Cycle 7

Instruction status

Instruction	j	k	Issue	Read operands	Executio complete	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				
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	No								

Register result status

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

- **Read multiply operands?**

FTC.W99 73

Scoreboard Example Cycle 8a

Instruction status

Instruction	j	k	Issue	Read operands	Executio complete	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			

FTC.W99 74

Scoreboard Example Cycle 8b

Instruction status

Instruction	j	k	Issue	Read operands	Executio complete	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			

FTC.W99 75

Scoreboard Example Cycle 9

Instruction status

Instruction	j	k	Issue	Read operands	Executio complete	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?**

FTC.W99 76

Scoreboard Example Cycle 11

Instruction status

Instruction	j	k	Issue	Read operands	Executio complete	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			

FTC.W99 77

Scoreboard Example Cycle 12

Instruction status

Instruction	j	k	Issue	Read operands	Executio complete	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?**

FTC.W99 78

Scoreboard Example Cycle 13

Instruction status				Issue	Read operands	Executio complete	Write Result
Instruction		j	k				
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			

FTC.W99 79

Scoreboard Example Cycle 14

Instruction status				Issue	Read operands	Executio complete	Write Result
Instruction		j	k				
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			

FTC.W99 80

Scoreboard Example Cycle 15

Instruction status				Issue	Read operands	Executio complete	Write Result
Instruction		j	k				
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			

FTC.W99 81

Scoreboard Example Cycle 16

Instruction status				Issue	Read operands	Executio complete	Write Result
Instruction		j	k				
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			

FTC.W99 82

Scoreboard Example Cycle 17

Instruction status				Issue	Read operands	Executio complete	Write Result
Instruction		j	k				
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?

FTC.W99 83

Scoreboard Example Cycle 18

Instruction status				Issue	Read operands	Executio complete	Write Result
Instruction		j	k				
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			

FTC.W99 84

Scoreboard Example Cycle 19

Instruction status

Instruction		j	k	Issue	Read operands	Executio complete	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	
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

Instruction		j	k	Issue	Read operands	Executio complete	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

Instruction		j	k	Issue	Read operands	Executio complete	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

Instruction		j	k	Issue	Read operands	Executio complete	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

Instruction		j	k	Issue	Read operands	Executio complete	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

Instruction		j	k	Issue	Read operands	Executio complete	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