Slides developed by Amir Roth of University of Pennsylvania with sources that included University of Wisconsin slides by Mark Hill, Guri Sohi, Jim Smith, and David Wood.

Slides enhanced by Milo Martin, Mark Hill, Alvin Lebeck, Dan Sorin, and David Wood with sources that included Profs. Asanovic, Falsafi, Hoe, Lipasti, Shen, Sohi, Vijaykumar, and Wood.

Administrative
- Homework #2 due (upload through blackboard)
- Homework #3 up
- Midterm moved to October 19.

Review: Superblocks
- First trace scheduling construct: superblock
  - Use when branch is highly biased
  - Fuse blocks from most frequent path: A,C,D
  - Schedule
  - Create repair code in case real path was A,B,D

Review: ISA Support for Load-Branch Speculation
- IA-64: change insn 2 to speculative load ldf.s
  - "Speculative" means advanced past some unknown branch
  - Processor keeps exception bit with register f8
  - Inserted insn chk.s checks exception bit
  - If exception, jump to yet more repair code (arghhh...)
- IA-64 also contains ldf.sa

Predication
- Conventional control
  - Conditionally executed insns also conditionally fetched
- Predication
  - Conditionally executed insns unconditionally fetched
  - Full predication (ARM, IA-64)
    - Can tag every insn with predicate, but extra bits in instruction
  - Conditional moves (Alpha, IA-32)
    - Construct appearance of full predication from one primitive
      - movneg r1, r2, r3 // if (r1=0) r3=r2;
      - May require some code duplication to achieve desired effect
      - Only good way of adding predication to an existing ISA
- If-conversion: replacing control with predication
  - Good if branch is unpredictable (save mis-prediction)
  - But more instructions fetched and "executed"
ISA Support for Predication

- IA-64: change branch 1 to set-predicate insn `fspne`
- Change insns 2 and 4 to predicated insns
  - `ldf.p` performs `ldf` if predicate `p1` is true
  - `stf.np` performs `stf` if predicate `p1` is false

```
0: ldf Y(r1),f2
1: fspne f2,p1
2: ldf.p p1,W(r1),f2
4: stf.np p1,f0,Y(r1)
5: ldf X(r1),f4
6: mulf f4,f2,f6
7: stf f6,Z(r1)
```

Hyperblock Scheduling

- Second trace scheduling construct: **hyperblock**
  - Use when branch is not highly biased
  - Fuse all four blocks: A,B,C,D
  - Use predication to conditionally execute insns in B and C
  - Schedule

```
A
0: ldf Y(r1),f2
1: fspne f2,p1
2: ldf.p p1,W(r1),f2
4: stf.np p1,f0,Y(r1)
5: ldf X(r1),f4
6: mulf f4,f2,f6
7: stf f6,Z(r1)
```

Static Scheduling Summary

- Goal: increase scope to find more independent insns
- Loop unrolling
  - Simple
    - Expands code size, can’t handle recurrences or non-loops
- Trace scheduling
  - Superblocks and hyperblocks
  - Works for non-loops
    - More complex, requires ISA support for speculation and predication
    - Requires nasty repair code

Multiple Issue Summary

- Problem spots
  - Wide fetch + branch prediction -> trace cache?
  - N^2 dependence cross-check
  - N^2 bypass -> clustering?
- Implementations
  - Statically scheduled superscalar
  - VLIW/EPIC
- What’s next:
  - Finding more ILP by relaxing the in-order execution requirement

Now: Dynamic Scheduling I

- Dynamic scheduling
  - Out-of-order execution
- Scoreboard
  - Dynamic scheduling with WAW/WAR
- Tomasulo’s algorithm
  - Add register renaming to fix WAW/WAR
- Next unit
  - Adding speculation and precise state
  - Dynamic load scheduling

The Problem With In-Order Pipelines

- What’s happening in cycle 4?
  - `mulf` stalls due to **RAW hazard**
    - OK, this is a fundamental problem
  - `subf` stalls due to **pipeline hazard**
    - Why? `subf` can’t proceed into D because `addf` is there
      - That is the only reason, and it isn’t a fundamental one
- Why can’t `subf` go into D in cycle 4 and E+ in cycle 6?
Dynamic Scheduling: The Big Picture

- Instructions fetch/decoded/renamed into Instruction Buffer
  - Also called “instruction window” or “instruction scheduler”
  - Instructions (conceptually) check ready bits every cycle
    - Execute when ready

Ready Table

P2 P3 P4 P5 P6 P7
--- --- --- --- --- ---
Time Time Time Time Time Time

- Time
- Insns: add p2, p3, p4
- sub p2, p4, p5
- mul p2, p5, p6
- div p4, 4, p7

Register Renaming

- To eliminate WAW and WAR hazards

Example

- Names: r1, r2, r3
- Locations: p1, p2, p3, p4, p5, p6, p7
- Original mapping: r1 → p1, r2 → p2, r3 → p3, p4–p7 are "free"

- Rename
  - Removes WAW and WAR dependences
  - Leaves RAW intact!

Map Table

<table>
<thead>
<tr>
<th>Raw insns</th>
<th>Free list</th>
<th>Renamed insns</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1, r2, r3</td>
<td>p1, p2, p3</td>
<td>r1, r2, r3</td>
</tr>
<tr>
<td>p4, p5, p6</td>
<td></td>
<td>p2, r3, r1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r2, p3, p4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>p4, p5, p6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>p5, p6, p7</td>
</tr>
</tbody>
</table>

Static Instruction Scheduling

- Issue: time at which insns execute
- Schedule: order in which insns execute
  - Related to issue, but the distinction is important

Scheduling: re-arranging insns to enable rapid issue

- Static: by compiler
- Requires knowledge of pipeline and program dependences
  - Pipeline scheduling: the basics
  - Requires large scheduling scope full of independent insns
    - Loop unrolling, software pipelining: increase scope for loops
    - Trace scheduling: increase scope for non-loops

Anything software can do … hardware can do better

Before We Continue

- If we can do this in software…
- …why build complex (slow-clock, high-power) hardware?
  - Performance portability
    - Don’t want to recompile for new machines
  - More information available
    - Memory addresses, branch directions, cache misses
  - More registers available (??)
    - Compiler may not have enough to fix WAR/WAW hazards
  - Easier to speculate and recover from mis-speculation
    - Flush instead of recover code
    - But compiler has a larger scope
    - Compiler does as much as it can (not much)
    - Hardware does the rest

Dynamic Scheduling - OoO Execution

- Dynamic scheduling
  - Totally in the hardware
  - Also called “out-of-order execution” (OoO)
- Fetch many instructions into instruction window
  - Use branch prediction to speculate past (multiple) branches
  - Flush pipeline on branch misprediction
- Rename to avoid false dependencies (WAW and WAR)
- Execute instructions as soon as possible
  - Handling memory dependencies more tricky (much more later)
- Commit instructions in order
  - Anything strange happens before commit, just flush the pipeline
- Current machines: 64-100+ instruction scheduling window

Motivation Dynamic Scheduling

- Dynamic scheduling (out-of-order execution)
  - Execute insns in non-sequential (non-VonNeumann) order...
    - Reduce RAW stalls
    - Increase pipeline and functional unit (FU) utilization
      - Original motivation was to increase FP unit utilization
    - Expose more opportunities for parallel issue (ILP)
    - Not in-order → can be in parallel
    - ...but make it appear like sequential execution
      - Important
      - But difficult
      - Next unit

Register Renaming

- To eliminate WAW and WAR hazards

Example

- Names: r1, r2, r3
- Locations: p1, p2, p3, p4, p5, p6, p7
- Original mapping: r1 → p1, r2 → p2, r3 → p3, p4–p7 are “free”
Going Forward: What’s Next

- We’ll build this up in steps over the next several lectures
  - “Scoreboarding” - first OoO, no register renaming
  - “Tomasulo’s algorithm” - adds register renaming
  - Handling precise state and speculation
    - P6-style execution (Intel Pentium Pro)
    - R10k-style execution (MIPS R10k)
  - Handling memory dependencies
    - Conservative and speculative

- Let’s get started!

Dynamic Scheduling as Loop Unrolling

- Three steps of loop unrolling
  - Step I: combine iterations
    - Increase scheduling scope for more flexibility
  - Step II: pipeline schedule
    - Reduce impact of RAW hazards
  - Step III: rename registers
    - Remove WAR/WAW violations that result from scheduling

Loop Example: SAX (SAXPY – PY)

- SAX (Single-precision A X)
  - Only because there won’t be room in the diagrams for SAXPY

```c
for (i=0;i<N;i++)
  Z[i] = A*X[i];
```

- Consider two iterations, ignore branch
  - ldf, mulf, stf, addi, ldf, mulf, stf

New Pipeline Terminology

- In-order pipeline
  - Often written as F,D,X,W (multi-cycle X includes M)
  - Example pipeline: 1-cycle int (including mem), 3-cycle pipelined FP

New Pipeline Diagram

```
<table>
<thead>
<tr>
<th>Ins</th>
<th>D</th>
<th>X</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldf X(r1),f1</td>
<td>c1</td>
<td>c2</td>
<td>c3</td>
</tr>
<tr>
<td>mulf f0,f1,f2</td>
<td>c3</td>
<td>c4+</td>
<td>c7</td>
</tr>
<tr>
<td>stf f2,Z(r1)</td>
<td>c7</td>
<td>c8+</td>
<td>c9+</td>
</tr>
<tr>
<td>addi r1,4,r1</td>
<td>c8+</td>
<td>c9+</td>
<td>c10+</td>
</tr>
<tr>
<td>blt r1,r2,0</td>
<td>c10+</td>
<td>c11+</td>
<td></td>
</tr>
</tbody>
</table>
```

- Alternative pipeline diagram
  - Down: insns
  - Across: pipeline stages
  - In boxes: cycles
  - Basically: stages ↔ cycles
  - Convenient for out-of-order

The Problem With In-Order Pipelines

- In-order pipeline
  - Structural hazard: 1 insn register (latch) per stage
    - 1 insn per stage per cycle (unless pipeline is replicated)
    - Younger insn can’t “pass” older insn without “clobbering” it
  - Out-of-order pipeline
    - Implement “passing” functionality by removing structural hazard
### Instruction Buffer

- **Trick:** *insn buffer* (many names for this buffer)
  - Basically: a bunch of latches for holding insns
  - Implements iteration fusing... here is your scheduling scope
- **Split D into two pieces**
  - Accumulate decoded insns in buffer in-order
  - Buffer sends insns down rest of pipeline out-of-order

### Dispatch and Issue

- **Dispatch (D):** first part of decode
  - Allocate slot in insn buffer
    - New kind of structural hazard (insn buffer is full)
  - In order: stall back-propagates to younger insns
- **Issue (S):** second part of decode
  - Send insns from insn buffer to execution units
    - Out-of-order: wait doesn't back-propagate to younger insns

### Dispatch and Issue with Floating-Point

### Dynamic Scheduling Algorithms

- **Three parts to loop unrolling**
  - Scheduling scope: insn buffer
  - Pipeline scheduling and register renaming: **scheduling algorithm**
- **Look at two register scheduling algorithms**
  - **Register scheduler:** scheduler based on register dependences
    - **Scoreboard**
      - No register renaming → limited scheduling flexibility
    - **Tomasulo**
      - Register renaming → more flexibility, better performance
- **Big simplification in this unit:** **memory scheduling**
  - Pretend register algorithm magically knows memory dependences
  - A little more realism next unit

### Scheduling Algorithm I: Scoreboard

- **Scoreboard**
  - Centralized control scheme: insn status explicitly tracked
  -Insn buffer: *Functional Unit Status Table (FUST)*
- **First implementation:** CDC 6600 [1964]
  -16 separate non-pipelined functional units (7 int, 4 FP, 5 mem)
  -No register bypassing
- **Our example:** "Simple Scoreboard"
  -5 FU: 1 ALU, 1 load, 1 store, 2 FP (3-cycle, pipelined)

### Scoreboard Data Structures

- **FU Status Table**
  - **FU, busy, op, R, R1, R2:** destination/source register names
  - **T:** destination register tag (FU producing the value)
  - **T1,T2:** source register tags (FU producing the values)
- **Register Status Table**
  - **T:** tag (FU that will write this register)
  - Tags interpreted as ready-bits
    - Tag == 0 → Value is ready in register file
    - Tag != 0 → Value is not ready, will be supplied by T
- **Insn status table**
  - S,X bits for all active insns (issue & execute)
Simple Scoreboard Data Structures

-Insn fields and status bits
-Tags
-Values

FU Status
R1 R2 R op T1 T2 CAMs
Insn value

Reg Status
Fetched insns

Scoreboard Pipeline

- New pipeline structure: F, D, S, X, W
  - F (fetch)
    - Same as it ever was
  - D (dispatch)
    - Structural or WAW hazard? Stall: allocate scoreboard entry
  - S (issue)
    - RAW hazard? Wait: read registers, go to execute
  - X (execute)
    - Execute operation, notify scoreboard when done
  - W (writeback)
    - WAR hazard? Wait: write register, free scoreboard entry
    - W and RAW-dependent S in same cycle
    - W and structural-dependent D in same cycle

Scoreboard Dispatch (D)

- Stall for WAW or structural (Scoreboard, FU) hazards
  - Allocate scoreboard entry
  - Copy Reg Status for input registers
  - Set Reg Status for output register

Scoreboard Issue (S)

- Wait for RAW register hazards
  - Read registers

Issue Policy and Issue Logic

- Issue
  - If multiple insns ready, which one to choose? Issue policy
    - Oldest first? Safe
    - Longest latency first? May yield better performance
  - Select logic: implements issue policy
    - W→1 priority encoder
    - W: window size (number of scoreboard entries)

Scoreboard Execute (X)

- Execute insn
Scoreboard Writeback (W)

- Wait for WAR hazard
  - Write value into regfile, clear Reg Status entry
  - Compare tag to waiting insn input tags, match ? clear input tag
  - Free scoreboard entry

Scoreboard Data Structures

Scoreboard: Cycle 1

<table>
<thead>
<tr>
<th>Insn Status</th>
<th>Reg Status</th>
<th>D</th>
<th>S</th>
<th>X</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldf X(r1),f1</td>
<td>F</td>
<td>R</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>mul f0,f1,f2</td>
<td>P</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>addi r1,4,r1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>ldf X(r1),f1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>mul f0,f1,f2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>stf f2,Z(r1)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

FU Status

<table>
<thead>
<tr>
<th>FU</th>
<th>Busy</th>
<th>sp</th>
<th>R1</th>
<th>R2</th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP1</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP2</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

allocate

Scoreboard: Cycle 2

<table>
<thead>
<tr>
<th>Insn Status</th>
<th>Reg Status</th>
<th>D</th>
<th>S</th>
<th>X</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldf X(r1),f1</td>
<td>E</td>
<td>c1</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>mul f0,f1,f2</td>
<td>E</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>addi r1,4,r1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>ldf X(r1),f1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>mul f0,f1,f2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>stf f2,Z(r1)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

FU Status

<table>
<thead>
<tr>
<th>FU</th>
<th>Busy</th>
<th>sp</th>
<th>R1</th>
<th>R2</th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP1</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP2</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

allocate

FP1 is ready
→ issue mulf

Scoreboard: Cycle 3

<table>
<thead>
<tr>
<th>Insn Status</th>
<th>Reg Status</th>
<th>D</th>
<th>S</th>
<th>X</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldf X(r1),f1</td>
<td>E</td>
<td>c1</td>
<td>c2</td>
<td>-</td>
<td>c3</td>
</tr>
<tr>
<td>mul f0,f1,f2</td>
<td>E</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>stf f2,Z(r1)</td>
<td>E</td>
<td>c3</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>ldf X(r1),f1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>mul f0,f1,f2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>stf f2,Z(r1)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

FU Status

<table>
<thead>
<tr>
<th>FU</th>
<th>Busy</th>
<th>sp</th>
<th>R1</th>
<th>R2</th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP1</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP2</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

allocate

Scoreboard: Cycle 4

<table>
<thead>
<tr>
<th>Insn Status</th>
<th>Reg Status</th>
<th>D</th>
<th>S</th>
<th>X</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldf X(r1),f1</td>
<td>E</td>
<td>c1</td>
<td>c2</td>
<td>c3</td>
<td>e4</td>
</tr>
<tr>
<td>mul f0,f1,f2</td>
<td>E</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>stf f2,Z(r1)</td>
<td>E</td>
<td>c5</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>ldf X(r1),f1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>mul f0,f1,f2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>stf f2,Z(r1)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

FU Status

<table>
<thead>
<tr>
<th>FU</th>
<th>Busy</th>
<th>sp</th>
<th>R1</th>
<th>R2</th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP1</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP2</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

allocate

f1 written → clear
f0 (LD) is ready → issue mulf
In-Order vs. Scoreboard

<table>
<thead>
<tr>
<th>In-Order</th>
<th>Scoreboard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instr</td>
<td>D X W</td>
</tr>
<tr>
<td>ldf X(r1),f1</td>
<td>c1 c2 c3</td>
</tr>
<tr>
<td>mulf f0,f1,f2</td>
<td>c3 c4 c7</td>
</tr>
<tr>
<td>stf f2,X(r1)</td>
<td>c7 c8 c9</td>
</tr>
<tr>
<td>addi r1,4,r1</td>
<td>c8 c9 c10</td>
</tr>
<tr>
<td>ldf X(r1),f1</td>
<td>c10 c11 c12</td>
</tr>
<tr>
<td>mulf f0,f1,f2</td>
<td>c12 c13 c14</td>
</tr>
<tr>
<td>stf f2,X(r1)</td>
<td>c14 c15 c16</td>
</tr>
<tr>
<td>mulf f0,f1,f2</td>
<td>c16 c17 c18</td>
</tr>
<tr>
<td>addi r1,4,r1</td>
<td>c16 c17 c18</td>
</tr>
</tbody>
</table>

- Big speedup?
  - Only 1 cycle advantage for scoreboard
    + Why? addi WAR hazard
    + scoreboard issued addi earlier (c8 → c5)
    + But WAR hazard delayed W until c9
    + Delayed issue of second iteration

In-Order vs. Scoreboard II: Cache Miss

<table>
<thead>
<tr>
<th>In-Order</th>
<th>Scoreboard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instr</td>
<td>D X W</td>
</tr>
<tr>
<td>ldf X(r1),f1</td>
<td>c1 c2 c3</td>
</tr>
<tr>
<td>mulf f0,f1,f2</td>
<td>c3 c4 c7</td>
</tr>
<tr>
<td>stf f2,X(r1)</td>
<td>c7 c8 c9</td>
</tr>
<tr>
<td>addi r1,4,r1</td>
<td>c8 c9 c10</td>
</tr>
<tr>
<td>ldf X(r1),f1</td>
<td>c10 c11 c12</td>
</tr>
<tr>
<td>mulf f0,f1,f2</td>
<td>c12 c13 c14</td>
</tr>
<tr>
<td>addi r1,4,r1</td>
<td>c12 c13 c14</td>
</tr>
<tr>
<td>ldf X(r1),f1</td>
<td>c14 c15 c16</td>
</tr>
<tr>
<td>mulf f0,f1,f2</td>
<td>c16 c17 c18</td>
</tr>
<tr>
<td>stf f2,X(r1)</td>
<td>c18 c17 c16</td>
</tr>
</tbody>
</table>

- Assume
  - 5 cycle cache miss on first ldf
  - Ignore FUST structural hazards
  - Little relative advantage
    + addi WAR hazard (c7 → c13) stalls second iteration

Scoreboard Redux

- The good
  + Cheap hardware
    + InstrStatus + FuStatus + RegStatus ~ 1 FP unit in area
  + Pretty good performance
    + 1.7X for FORTRAN (scientific array) programs

- The less good
  - No bypassing
    + Is this a fundamental problem?
  - Limited scheduling scope
    + Structural/WAW hazards delay dispatch
    + Slow issue of truly-dependent (RAW) insns
    + WAR hazards delay writeback
    + Fix with hardware register renaming