Lecture #8

****

Review -- 1 min

****

- o longer pipelines
  - MIPS R4000
  - floating point
- o ILP: compiler perspective
  - basic block not enough
  - loop-level parallelism easy to see/analyze
  - dependencies restrict reordering
    - data dependencies -- real
    - name dependencies -- avoid via renaming
      - memory dependencies hardest to determine

- o control dependencies
  - can't move instructions from after a branch to before it
    - so it is no longer controlled by a branch
  - can't move instructions from before a branch to after
    - it so it is controlled by a branch
  - restrictions relaxed for speed
    - preserve order of exceptions
    - preserve data flow

****

Outline - 1 min

****

HW for Instruction Parallelism

- CDC6600 Scoreboard
- Scoreboard example

****

Lecture - 20 min

****

HW For ILP

Why HW at run time?

- Works when can't know dependence at compile time
  - e.g. memory references
  - cross basic blocks
- 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    F8, F8, F14
```

subtract doesn't depend on add, so let it start!

- out of order execution → out of order completion

  → WAR, WAW Hazards

Scoreboard info

CDC 6600 1960's

scoreboard = a board (like functional unit board) that summarizes all hazard detection in 1 place

Divide ID stage into 2 parts (Issue, read operands)

1. Issue – decode instructions;
   check for structural hazards
   check for WAW hazards
   (Didn't expect WAW to be common, so stall)

   Note: if Issue stalls, no later instructions issue

2. Read operands – wait until no data hazards, then
   then read operands

→ resolve RAW hazards dynamically

3. Execute

for this to matter, need multiple exec

→ multiple ex units or pipelined ex units

4. Write Result

stall instruction if WAR hazard

example

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

→ stall SUBD until ADDD reads operands

characteristics

1. in-order issue
2. out of order execution
3. out of order completion

Implementation: 3 parts

"If I made this a HW question, this is pretty much what you would come up with…"

<draw picture as you list pieces>

1. Instruction status

which of 4 steps instruction is in

2. Functional Unit Status

state of FU; 9 fields per FU

Busy

Op – operation to perform in unit (e.g. ADD or SUB)
Fi – destination register number
Fj, Fk – source register numbers
Qj, Qk – functional units producing source regs Fj, Fk
Rj, Rk – flags indicating when Fj and Fk are ready

3. Register Result Status
   which FU will write register

Questions:

1. Issue checks for structural hazard. Looking at scoreboard, how can you know if there is a structural hazard?
2. Issue checks for WAW hazard. Looking at scoreboard, how can you know if there is a WAW hazard?
3. Read operands resolves RAW hazards. Suppose no RAW hazard exists, how does Read Operands know that it can allow instruction to proceed?
4. Suppose a RAW hazard exists – an issued instruction depends on an instruction that is executing. How does Read Operands know that it has to make the instruction wait?
5. When a FU completes an operation that clears a RAW hazard, what does scoreboard entry for waiting instruction do?
6. How does WB stage detect WAR hazards? How can it tell the difference between WAR and RAW?

   This one is a little complicated. When the result completes, it sees that a FU status entry indicates that a Fj or Fk entry is waiting to read the register that the instruction is getting ready to write. (Simple enough) The only problem is: what order should the read/write proceed (RAW or WAR)? Answer: if Rj/Rk indicates that the register is "Ready" then stall the write – we're in WAR; otherwise let the write proceed, we're in RAW.

****

Admin - 3 min
****
comments on project proposals

suggestion – look at class page for pointers to tools, traces that may be useful for project.

hw2 part A due Friday

****
Lecture - 24 min
****

<Remainder of lecture: Scoreboard example using slides on overhead projector>

Most of it is pretty obvious, but note on cycle 20/21 – seems like add can write back in cycle 20 (scoreboard the same in both cycles) The reason it can't is a structural hazard – only one FU may read or write a register during a cycle; make write wait (otherwise would get wrong answer).

Limitations of CDC 6600 scoreboard
----
1. limited instruction parallelism
   limited to basic block
2. limited by scoreboard window size (# scoreboard entries)
   determines how far ahead can look for independent instructions
3. # and types of functional units
4. presence of antidependencies and output dependencies

1 – implementation detail; fixed in current out-of-order machines
2,3 – parameters – spend more money

4. register renaming – tomasulo algorithm (tomorrow)

****

Summay - 1 min

****

Hardware > software

    code compatibility

    run-time dependencies

Scoreboard – solves RAW; stalls on WAR, WAW, structural