LECTURE - 12

ILP: Recall

- Improving ILP == reducing stalls
- Loop unrolling enlarges the basic block
  - More parallelism
  - More opportunity for better scheduling
- Dependences:
  - Data dependence
  - Name dependence
  - Control dependence

Handling Control Dependence

- Control dependence need not be maintained
- We need to maintain:
  - *Exception behaviour* – do not cause new exceptions
  - *Data flow* – ensure the right data item is used
- Speculation and conditional instructions are techniques to get around control dependence

Loop Unrolling: a Relook

- Our example:

```
for(int i = 1000; i >= 1; i = i-1) {
    x[i] = x[i] + C; // FP
}
```

- Consider:

```
for(int i = 1000; i >= 1; i = i-1) {
    A[i-1] = A[i] + C[i];      // S1
    B[i-1] = B[i] + A[i-1];    // S2
}
```

  - S2 is dependent on S1
  - S1 is dependent on its previous iteration; same case with S2
    - *Loop-carried dependence* ==> loop iterations have to be in-order

Removing Loop-Carried Dependence

- Another example:

```
for(int i = 1000; i >= 1; i = i-1) {
    A[i] = A[i] + B[i];        // S1
    B[i-1] = C[i] + D[i];      // S2
}
```

- S1 depends on the prior iteration of S2
  - Can be removed (no cyclic dependence)

```
A[1000] = A[1000] + B[1000];
for(int i = 1000; i >= 2; i = i-1) {
    B[i-1] = C[i] + D[i];      // S2
    A[i-1] = A[i-1] + B[i-1]; // S1
}
B[0] = C[1] + D[1];
```

Static vs. Dynamic Scheduling

- Static scheduling: limitations
  - Dependences may not be known at compile time
  - Even if known, compiler becomes complex
  - Compiler has to have knowledge of pipeline
- Dynamic scheduling
  - Handle dynamic dependences
  - Simpler compiler
  - Efficient even if code compiled for a different pipeline

Dynamic Scheduling

- For now, we will focus on overcoming data hazards
- The idea:
  - DIVD F0, F2, F4
  - ADDD F10, F0, F8
  - SUBD F12, F8, F14
    - SUBD can proceed without waiting for DIVD

CDC 6600: A Case Study

- IF stage: fetch instructions onto a *queue*
- ID stage is split into two stages:
  - *Issue:* decode and check for structural hazards
  - *Read operands:* check for data hazards
- Execution may begin, and may complete *out-of-order*
  - Complications in exception handling
  - Ignore for now
- What is the logic for data hazard checks?

The CDC Scoreboard

- Out-of-order completion ==> WAR and WAW hazards possible
- Scoreboard: a data-structure for all hazard detection in the presence of out-of-order execution/completion
- All instructions "cons ult" the scoreboard to detect hazards

The Scoreboard Solution

- Three components:
  - Stages of the pipeline:
    - Issue (ID1), Read-operands (ID2), EX, WB
  - Data structure (in hardware)
  - Logic for hazard detection, stalling

Scoreboard Control & the Pipeline Stages

- Issue (ID1): decode, check if functional unit is free, and if a previous instruction has the same destination register
  - No such hazard ==> scoreboard issues to the appropriate functional unit
    - Note: structural/WAW hazards prevented by stalling here
    - Note: stall here ==> IF queue will grow
- Read operands (ID2):
  - Operand is available if no earlier instruction is going to write it, or if the register is being written currently
  - RAW hazards are resolved here

Scoreboard Control & the Pipeline Stages (continued)

- Execute (EX):
  - Functional units perform execution
  - Scoreboard is notified on completion
- Write-Back (WB):
  - Check for WAR hazards
    - Stall on detection
    - Write-back otherwise

Some Remarks

- WAW causes stall in ID1, WAR causes stall in WB
- No forwarding logic
  - Output written as soon as it is available (and no WAR hazard)
- Structural hazard possible in register read/write
  - CDC has 16 functional units, and 4 buses

The Scoreboard Data-Structures

- Instruction status
- Functional unit status
- Register result status
-
- Randy Katz's CS252 slides... (Lecture 10, Spring 1996)
  - Scoreboard pipeline control
  - A detailed example

Limitations of the Scoreboard

- Speedup of 1.7 for (compiled) FORTRAN, speedup of 2.5 for hand-coded assembly
- Scoreboard only in basic-block!
- Some hazards still cause stalls:
  - Structural
  - WAR, WAW