Lecture: Static ILP

- Topics: predication, speculation (Sections C.5, 3.2)

Predication

- A branch within a loop can be problematic to schedule
- Control dependences are a problem because of the need to re-fetch on a mispredict
- For short loop bodies, control dependences can be converted to data dependences by using predicated/conditional instructions

Predicated or Conditional Instructions

```
if (R1 == 0)
   R2 = R2 + R4
else
   R6 = R3 + R5
   R4 = R2 + R3
```

```
R7 = !R1
R8 = R2
R2 = R2 + R4   (predicated on R7)
R6 = R3 + R5   (predicated on R1)
R4 = R8 + R3   (predicated on R1)
```

Predicated or Conditional Instructions

- The instruction has an additional operand that determines whether the instr completes or gets converted into a no-op

- Example: lwc R1, 0(R2), R3 (load-word-conditional) will load the word at address (R2) into R1 if R3 is non-zero; if R3 is zero, the instruction becomes a no-op

- Replaces a control dependence with a data dependence (branches disappear) ; may need register copies for the condition or for values used by both directions

```
if (R1 == 0)
   R2 = R2 + R4
else
   R6 = R3 + R5
   R4 = R2 + R3
```

```
R7 = !R1 ;  R8 = R2 ;
R2 = R2 + R4   (predicated on R7)
R6 = R3 + R5   (predicated on R1)
R4 = R8 + R3   (predicated on R1)
```

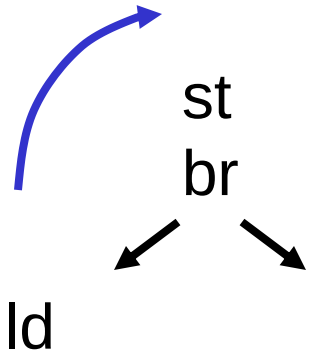
Complications

- Each instruction has one more input operand – more register ports/bypassing
- If the branch condition is not known, the instruction stalls (remember, these are in-order processors)
- Some implementations allow the instruction to continue without the branch condition and squash/complete later in the pipeline – wasted work
- Increases register pressure, activity on functional units
- Does not help if the br-condition takes a while to evaluate

Support for Speculation

- In general, when we re-order instructions, register renaming can ensure we do not violate register data dependences

- However, we need hardware support
  - to ensure that an exception is raised at the correct point
  - to ensure that we do not violate memory dependences



Detecting Exceptions

- Some exceptions require that the program be terminated (memory protection violation), while other exceptions require execution to resume (page faults)
- For a speculative instruction, in the latter case, servicing the exception only implies potential performance loss
- In the former case, you want to defer servicing the exception until you are sure the instruction is not speculative
- Note that a speculative instruction needs a special opcode to indicate that it is speculative

Program-Terminate Exceptions

- When a speculative instruction experiences an exception, instead of servicing it, it writes a special NotAThing value (NAT) in the destination register
- If a non-speculative instruction reads a NAT, it flags the exception and the program terminates (it may not be desireable that the error is caused by an array access, but the segfault happens two procedures later)
- Alternatively, an instruction (the *sentinel*) in the speculative instruction's original location checks the register value and initiates recovery

Memory Dependence Detection

- If a load is moved before a preceding store, we must ensure that the store writes to a non-conflicting address, else, the load has to re-execute
- When the speculative load issues, it stores its address in a table (Advanced Load Address Table in the IA-64)
- If a store finds its address in the ALAT, it indicates that a violation occurred for that address
- A special instruction (the *sentinel*) in the load's original location checks to see if the address had a violation and re-executes the load if necessary

Title

- Bullet