LECTURE - 15

Further Topics in ILP

- Multiple issue
- Software support
- Hardware support

Increasing ILP through Multiple Issue

- With at most one issue per cycle, min CPI possible is 1
  - But there are multiple functional units
  - Hence use multiple issue
- Two ways to do multiple issue
  - Superscalar processor
    - Issue varying number of instructions per cycle
    - Static or dynamic scheduling
  - Very Large Instruction Word (VLIW)
    - Issue a fixed number of instructions

Superscalar DLX

- Simple version: two instructions issued per cycle
  - One integer (load, store, branch, integer ALU) and one FP
  - Instructions paired and aligned on 64-bit boundaries – int first, FP next

	CC1	CC2	CC3	CC4	CC5	CC6
Integer	IF	ID	EX	MEM	WB	
FP	IF	ID	EX	MEM	WB	
Integer		IF	ID	EX	MEM	WB
FP		IF	ID	EX	MEM	WB

Superscalar DLX (continued)

- No conflicts, almost...
  - Assuming separate register sets, only FP load, store, move cause problems
    - Structural hazard on register port
    - New RAW hazard between a pair of instructions
  - Structural hazard:
    - Detect, and do not issue the FP operation
    - Or, provide additional register ports
  - RAW hazard:
    - Detect, and do not issue the FP operation
- Also, result of LD cannot be used for 3 instns.

Static Scheduling in the Superscalar DLX: An Example

```
Loop:   LD      F0, 0(R1)     // F0 is array element
        ADDD    F4, F0, F2    // F2 has the scalar 'C'
        SD      0(R1), F4     // Stored result
        SUBI    R1, R1, 8     // For next iteration
        BNEZ    R1, Loop      // More iterations?
```

```
Loop:   LD      F0, 0(R1)
        LD      F6, -8(R1)
        LD      F10, -8(R1)     ADDD  F4, F0, F2
        LD      F14, -8(R1)     ADDD  F8, F6, F2
        LD      F18, -8(R1)     ADDD  F12, F10, F2
        SD      0(R1), F4       ADDD  F16, F14, F2
        SD      -8(R1), F8      ADDD  F20, F18, F2
        SD      -16(R1), F12
        SUBI    R1, R1, #40
        SD      -24(R1), F16
        BNEZ    R1, Loop
```

Dynamic Scheduling in the Superscalar DLX

- Scoreboard or Tomasulo can be applied
- Should preserve in-order issue!
  - Use separate data structures for Int and FP
- When the instruction pair has a dependence
  - We wish to issue both in the same cycle
  - Two approaches:
    - Pipeline the issue stage, so that it runs twice as fast
    - Exclude load/store buffers from the set of RSs

Multiple Issue using VLIW

- Superscalar ==> too much hardware
  - For hazard detection, scheduling
- Alternative: let compiler do all the scheduling
  - VLIW (Very Large Instruction Word)
  - E.g., an VLIW may include 2 Int, 2 FP, 2 mem, and a branch

Limitations to Multiple Issue

- Why not 10 issues per cycle? Why not 20?
- Three limitations:
  - Inherent ILP limitations in programs
  - Hardware costs (even for VLIW)
    - Memory/register bandwidth
  - Implementation issues:
    - Superscalar: complexity of hardware logic
    - VLIW: increased code size, binary compatibility problems

Support for ILP

- Software (compiler) support
- Hardware support
- Combination of both

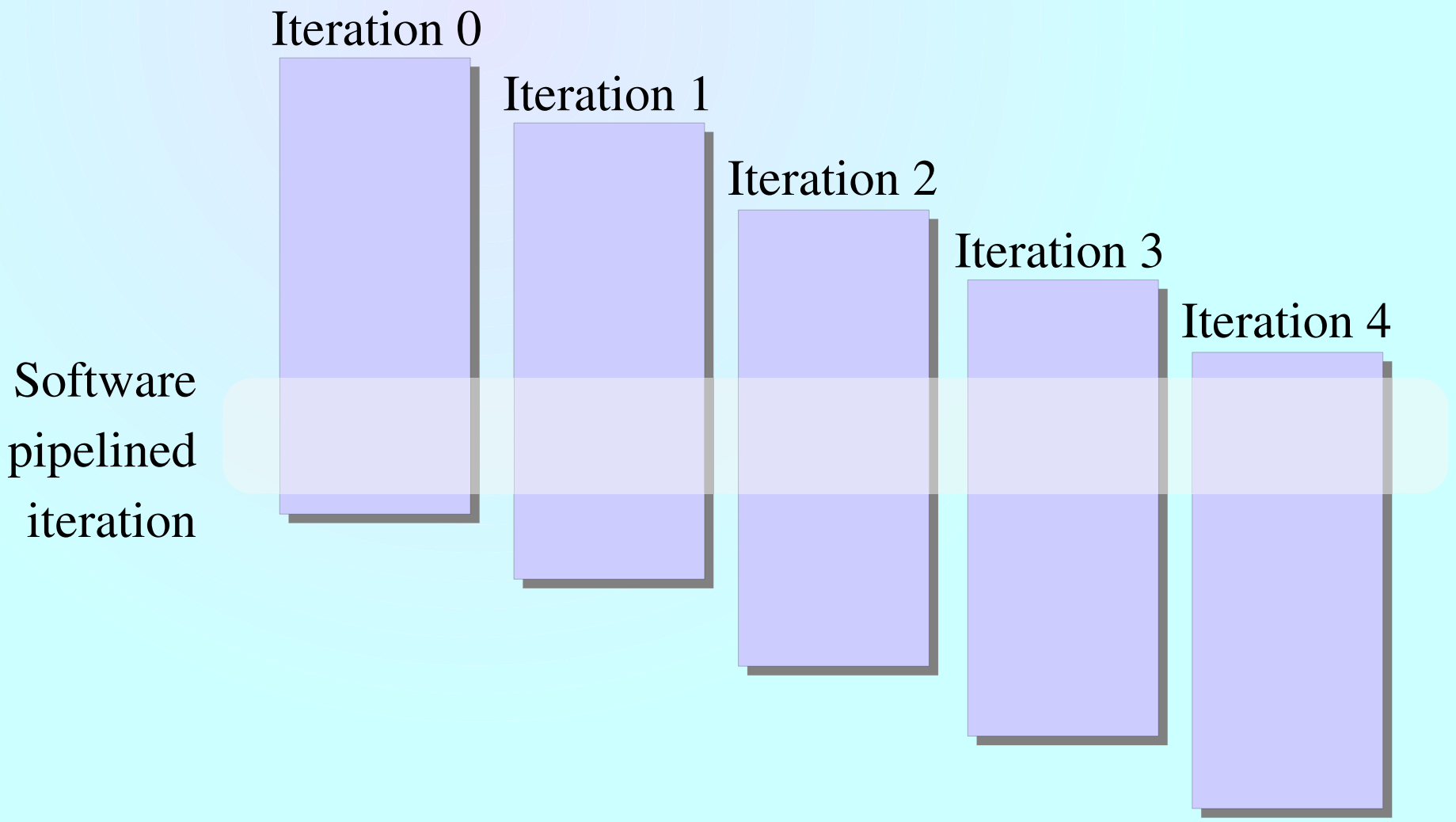
Compiler Support for ILP

- Loop unrolling:
  - Dependence analysis is a major component
  - Analysis is simple when array indices are linear in the loop variable (called *affine* indices)
- Limitations to dependence analysis:
  - Pointers
  - Indirect indexing
  - Analysis has to consider corner cases too

Compiler Support for ILP (continued)

- Two important techniques:
  - Software pipelining
  - Trace scheduling
- Software pipelining: reorganize a loop such that each iteration is made from instructions chosen from *different iterations* of the original loop

Software Pipelining



Software Pipelining in Our Example

```
Loop:  LD     F0, 0(R1)    // F0 is array element
       ADDD   F4, F0, F2   // F2 has the scalar 'C'
       SD     0(R1), F4    // Stored result
       SUBI   R1, R1, 8    // For next iteration
       BNEZ   R1, Loop     // More iterations?
```

```
Iter i:    LD     F0, 0(R1)
           ADDD   F4, F0, F2
           SD     0(R1), F4
Iter i+1:  LD     F0, 0(R1)
           ADDD   F4, F0, F2
           SD     0(R1), F4
Iter i+2:  LD     F0, 0(R1)
           ADDD   F4, F0, F2
           SD     0(R1), F4
```

Software Pipelined Loop

```
Loop:  SD     16(R1), F4
       ADDD   F4, F0, F2
       LD     F0, 0(R1)
       SUBI   R1, R1, 8
       BNEZ   R1, Loop
```

Trace Scheduling

- Compiler picks a program *trace* which it considers most likely
  - Schedule instructions from the trace
  - And branches into and out of the trace
  - Also need bookkeeping instructions in case the trace is not taken during execution