Chapter 3 (Cont IV):
Further Techniques for Exploiting ILP
Software Pipelining

• Code reorganization technique to uncover parallelism
• Idea: each iteration contains instructions from several different iterations in the original loop
• The reason: separate the dependent instructions that occur within a single loop iteration
• We need some start-up code before the loop begins and some code to finish up after the loop is completed
Software Pipelining
Software Pipelining

Loop:  LD      F0,0(R1)
       ADDD F4,F0,F2
       SD      F4,0(R1)
       DADDUI  R1,R1,#-8
       BNE  R1,R2,Loop

Loop:  SD F4,16(R1) ;stores into M[i]
       ADDD F4,F0,F2  ; adds to M[i-1]
       LD F0,0(R1)   ; loads M[i-2]
       DADDUI  R1,R1,#-8
       BNE  R1,R2,Loop

It i:   LD F0,0(R1)
       ADDD F4,F0,F2
       SD F4,0(R1)

It I+1: LD F0,0(R1)
        ADDD F4,F0,F2
        SD F4,0(R1)

It I+2: LD F0,0(R1)
        ADDD F4,F0,F2
        SD F4,0(R1)
Software Pipelining

• Every 5 cycles, we get a result (ignoring the startup and cleanup portions)
• Notice that there are no true dependences
• Because the load and store are separated by two iterations:
  – The loop should run for two fewer iterations
  – The startup code is: LD of iterations 1 and 2, ADDD of iteration 1
  – The cleanup code is: ADDD for last iteration and SD for the last two iterations
Software Pipelining

• Register management can be tricky
• Example shown is not hard: registers that are written in one iteration are read in the next one
• If we have long latencies of the dependences:
  – May need to increase the number of iterations between when we write a reg and use it
  – May have to manage the reg use
  – May have to combine software pipelining and loop unrolling
Software Pipelining vs Loop Unrolling

• Software pipelining consumes less code space
• Both yield a better scheduled inner loop
• Each reduces a different type of overhead:
  – LU: branch and counter update code
  – SP: reduces the time when the loop is not running at peak speed (only once at the beginning and once at the end)
Software Pipelining vs Loop Unrolling

- Non-rectangular area: times when the loop is not running at maximum overlap or parallelism between instr
Conditional or Predicated Instructions

- When branch behavior is not well known, compiler techniques may not be of much use
- In this case, use hardware techniques:
  - Add **Conditional** or **Predicated** instructions: used to eliminate branches and to assist the compiler to move instructions up past branches
  - Support Dynamic Speculation: Done by the HW using branch prediction: allow the execution of an instruction before the processor knows that the instruction should execute
Conditional or Predicated Instructions

- An instruction refers to a condition, which is evaluated as part of the instruction execution.
- If condition is true: instruction is executed normally; if false: the instruction is a NO-OP.
- E.g. conditional move: move a value from a reg to another one if a condition is true.

If \( A = 0 \) \{ \( S = T; \} \)

```
BNEZ R1, L
ADDU R2,R3,R0
L:
```

```
CMOVZ R2,R3, R1
;performs the move only if
; third op is 0
```
Conditional or Predicated Instructions

- Allow us to convert control dep to data dep
- In a pipeline: moves the resolution from near the front of the pipeline to the end where the register write occurs!
- Another example: Conditional load, which loads only if the third operand is not zero
  
  \[
  \text{LWC} \quad \text{R8, 20(R12), R10}
  \]

- If this instruction used speculatively, we must ensure that it does not cause an exception
- Therefore, these conditional instructions, if condition not true:
  - No effect (not update any reg)
  - No exception
Conditional or Predicated Instructions

• If R10 contains 0, it better not cause an exception
• Problems with conditional instructions:
  – Those that are annulled, still take time
  – Conditional instructions are most useful when the condition can be evaluated early
  – Sometimes it would be useful to have several conditions
  – Conditional instr may have some speed penalty relative to non conditional

LWC R8, 20 (R10), R10
BEQZ R10, L
LW R8, 20 (R10)