

Lecture 5: Static ILP Basics

- Topics: loop unrolling, VLIW
(Sections 2.1 – 2.2)

Static vs Dynamic Scheduling

- Arguments against dynamic scheduling:
 - requires complex structures to identify independent instructions (scoreboards, issue queue)
 - high power consumption
 - low clock speed
 - high design and verification effort
 - the compiler can “easily” compute instruction latencies and dependences – complex software is always preferred to complex hardware (?)

Loop Scheduling

- Revert back to the 5-stage in-order pipeline
- The compiler's job is to minimize stalls
- Focus on loops: account for most cycles, relatively easy to analyze and optimize
- Recall: a load has a two-cycle latency (1 stall cycle for the consumer that immediately follows), FP ALU feeding another \rightarrow 3 stall cycles, FP ALU feeding a store \rightarrow 2 stall cycles, int ALU feeding a branch \rightarrow 1 stall cycle, one delay slot after a branch

Loop Example

```
for (i=1000; i>0; i--)  
    x[i] = x[i] + s;
```

Source code

```
Loop:  L.D      F0, 0(R1)      ; F0 = array element  
       ADD.D   F4, F0, F2     ; add scalar  
       S.D     F4, 0(R1)     ; store result  
       DADDUI  R1, R1, # -8   ; decrement address pointer  
       BNE    R1, R2, Loop    ; branch if R1 != R2
```

Assembly code

Loop Example

```
for (i=1000; i>0; i--)  
    x[i] = x[i] + s;
```

Source code

```
Loop:  L.D      F0, 0(R1)      ; F0 = array element  
        ADD.D   F4, F0, F2    ; add scalar  
        S.D     F4, 0(R1)    ; store result  
        DADDUI  R1, R1, # -8  ; decrement address pointer  
        BNE    R1, R2, Loop  ; branch if R1 != R2
```

Assembly code

```
Loop:  L.D      F0, 0(R1)      ; F0 = array element  
        stall  
        ADD.D   F4, F0, F2    ; add scalar  
        stall  
        stall  
        S.D     F4, 0(R1)    ; store result  
        DADDUI  R1, R1, # -8  ; decrement address pointer  
        stall  
        BNE    R1, R2, Loop  ; branch if R1 != R2  
        stall
```

10-cycle
schedule

Smart Schedule

```
Loop:  L.D      F0, 0(R1)
      stall
      ADD.D   F4, F0, F2
      stall
      stall
      S.D     F4, 0(R1)
      DADDUI  R1, R1, # -8
      stall
      BNE    R1, R2, Loop
      stall
```



```
Loop:  L.D      F0, 0(R1)
      DADDUI  R1, R1, # -8
      ADD.D   F4, F0, F2
      stall
      BNE    R1, R2, Loop
      S.D     F4, 8(R1)
```

- By re-ordering instructions, it takes 6 cycles per iteration instead of 10
- We were able to violate an anti-dependence easily because an immediate was involved
- Loop overhead (instrs that do book-keeping for the loop): 2
Actual work (the ld, add.d, and s.d): 3 instrs
Can we somehow get execution time to be 3 cycles per iteration?

Loop Unrolling

```
Loop:  L.D      F0, 0(R1)
        ADD.D   F4, F0, F2
        S.D     F4, 0(R1)
        L.D     F6, -8(R1)
        ADD.D   F8, F6, F2
        S.D     F8, -8(R1)
        L.D     F10,-16(R1)
        ADD.D   F12, F10, F2
        S.D     F12, -16(R1)
        L.D     F14, -24(R1)
        ADD.D   F16, F14, F2
        S.D     F16, -24(R1)
        DADDUI  R1, R1, #-32
        BNE    R1,R2, Loop
```

- Loop overhead: 2 instrs; Work: 12 instrs
- How long will the above schedule take to complete?

Scheduled and Unrolled Loop

```
Loop:  L.D      F0, 0(R1)
        L.D      F6, -8(R1)
        L.D      F10,-16(R1)
        L.D      F14, -24(R1)
        ADD.D    F4, F0, F2
        ADD.D    F8, F6, F2
        ADD.D    F12, F10, F2
        ADD.D    F16, F14, F2
        S.D      F4, 0(R1)
        S.D      F8, -8(R1)
        DADDUI   R1, R1, # -32
        S.D      F12, 16(R1)
        BNE     R1,R2, Loop
        S.D      F16, 8(R1)
```

- Execution time: 14 cycles or 3.5 cycles per original iteration

Loop Unrolling

- Increases program size
- Requires more registers
- To unroll an n -iteration loop by degree k , we will need (n/k) iterations of the larger loop, followed by $(n \bmod k)$ iterations of the original loop

Automating Loop Unrolling

- Determine the dependences across iterations: in the example, we knew that loads and stores in different iterations did not conflict and could be re-ordered
- Determine if unrolling will help – possible only if iterations are independent
- Determine address offsets for different loads/stores
- Dependency analysis to schedule code without introducing hazards; eliminate name dependences by using additional registers

Superscalar Pipelines

Integer pipeline	FP pipeline
Handles L.D, S.D, ADDUI, BNE	Handles ADD.D

- What is the schedule with an unroll degree of 4?

Superscalar Pipelines

	Integer pipeline	FP pipeline
Loop:	L.D F0,0(R1)	
	L.D F6,-8(R1)	
	L.D F10,-16(R1)	ADD.D F4,F0,F2
	L.D F14,-24(R1)	ADD.D F8,F6,F2
	L.D F18,-32(R1)	ADD.D F12,F10,F2
	S.D F4,0(R1)	ADD.D F16,F14,F2
	S.D F8,-8(R1)	ADD.D F20,F18,F2
	S.D F12,-16(R1)	
	DADDUI R1,R1,# -40	
	S.D F16,16(R1)	
	BNE R1,R2,Loop	
	S.D F20,8(R1)	

- Need unroll by degree 5 to eliminate stalls
- The compiler may specify instructions that can be issued as one packet
- The compiler may specify a fixed number of instructions in each packet:
Very Large Instruction Word (VLIW)

Title

- Bullet