

Lecture: Static ILP

- Topics: loop unrolling, software pipelines (Sections C.5, 3.2)

Loop Example

LD -> any : 1 stall
 FPALU -> any: 3 stalls
 FPALU -> ST : 2 stalls
 IntALU -> BR : 1 stall

```
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
	NOP		
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		

Assembly code

10-cycle
 schedule

Smart Schedule

LD -> any : 1 stall
FPALU -> any: 3 stalls
FPALU -> ST : 2 stalls
IntALU -> BR : 1 stall

```
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?

Problem 1

LD -> any : 1 stall
FPMUL -> any: 5 stalls
FPMUL -> ST : 4 stalls
IntALU -> BR : 1 stall

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

Source code

```
Loop:  L.D      F0, 0(R1)      ; F0 = array element  
       MUL.D   F4, F0, F2     ; multiply scalar  
       S.D     F4, 0(R2)     ; store result  
       DADDUI  R1, R1, #-8    ; decrement address pointer  
       DADDUI  R2, R2, #-8    ; decrement address pointer  
       BNE    R1, R3, Loop    ; branch if R1 != R3  
       NOP
```

Assembly code

- How many cycles do the default and optimized schedules take?

Problem 1

LD -> any : 1 stall
FPMUL -> any: 5 stalls
FPMUL -> ST : 4 stalls
IntALU -> BR : 1 stall

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

Source code

```
Loop:  L.D      F0, 0(R1)      ; F0 = array element  
       MUL.D   F4, F0, F2     ; multiply scalar  
       S.D     F4, 0(R2)     ; store result  
       DADDUI  R1, R1, #-8    ; decrement address pointer  
       DADDUI  R2, R2, #-8    ; decrement address pointer  
       BNE    R1, R3, Loop    ; branch if R1 != R3  
       NOP
```

Assembly code

- How many cycles do the default and optimized schedules take?

Unoptimized: LD 1s MUL 4s SD DA DA BNE 1s -- 12 cycles

Optimized: LD DA MUL DA 2s BNE SD -- 8 cycles


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)
```



LD -> any : 1 stall
FPALU -> any: 3 stalls
FPALU -> ST : 2 stalls
IntALU -> BR : 1 stall

- 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

Problem 2

LD -> any : 1 stall
FPMUL -> any: 5 stalls
FPMUL -> ST : 4 stalls
IntALU -> BR : 1 stall

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

Source code

```
Loop:  L.D      F0, 0(R1)      ; F0 = array element  
       MUL.D   F4, F0, F2     ; multiply scalar  
       S.D     F4, 0(R2)     ; store result  
       DADDUI  R1, R1, #-8    ; decrement address pointer  
       DADDUI  R2, R2, #-8    ; decrement address pointer  
       BNE    R1, R3, Loop    ; branch if R1 != R3  
       NOP
```

Assembly code

- How many unrolls does it take to avoid stall cycles?

Problem 2

LD -> any : 1 stall
FPMUL -> any: 5 stalls
FPMUL -> ST : 4 stalls
IntALU -> BR : 1 stall

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

Source code

```
Loop:  L.D      F0, 0(R1)      ; F0 = array element  
       MUL.D   F4, F0, F2     ; multiply scalar  
       S.D     F4, 0(R2)     ; store result  
       DADDUI  R1, R1, #-8    ; decrement address pointer  
       DADDUI  R2, R2, #-8    ; decrement address pointer  
       BNE    R1, R3, Loop    ; branch if R1 != R3  
       NOP
```

Assembly code

- How many unrolls does it take to avoid stall cycles?

Degree 2: LD LD MUL MUL DA DA 1s SD BNE SD

Degree 3: LD LD LD MUL MUL MUL DA DA SD SD BNE SD

– 12 cyc/3 iterations

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)

Problem 3

LD -> any : 1 stall
FPMUL -> any: 5 stalls
FPMUL -> ST : 4 stalls
IntALU -> BR : 1 stall

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

Source code

```
Loop:  L.D      F0, 0(R1)      ; F0 = array element  
       MUL.D   F4, F0, F2     ; multiply scalar  
       S.D     F4, 0(R2)     ; store result  
       DADDUI  R1, R1, #-8    ; decrement address pointer  
       DADDUI  R2, R2, #-8    ; decrement address pointer  
       BNE    R1, R3, Loop    ; branch if R1 != R3  
       NOP
```

Assembly code

- How many unrolls does it take to avoid stalls in the superscalar pipeline?

Problem 3

LD -> any : 1 stall
FPMUL -> any: 5 stalls
FPMUL -> ST : 4 stalls
IntALU -> BR : 1 stall

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

Source code

```
Loop:  L.D      F0, 0(R1)      ; F0 = array element  
        MUL.D   F4, F0, F2     ; multiply scalar  
        S.D     F4, 0(R2)     ; store result  
        DADDUI  R1, R1, #-8    ; decrement address pointer  
        DADDUI  R2, R2, #-8    ; decrement address pointer  
        BNE    R1, R3, Loop    ; branch if R1 != R3  
        NOP
```

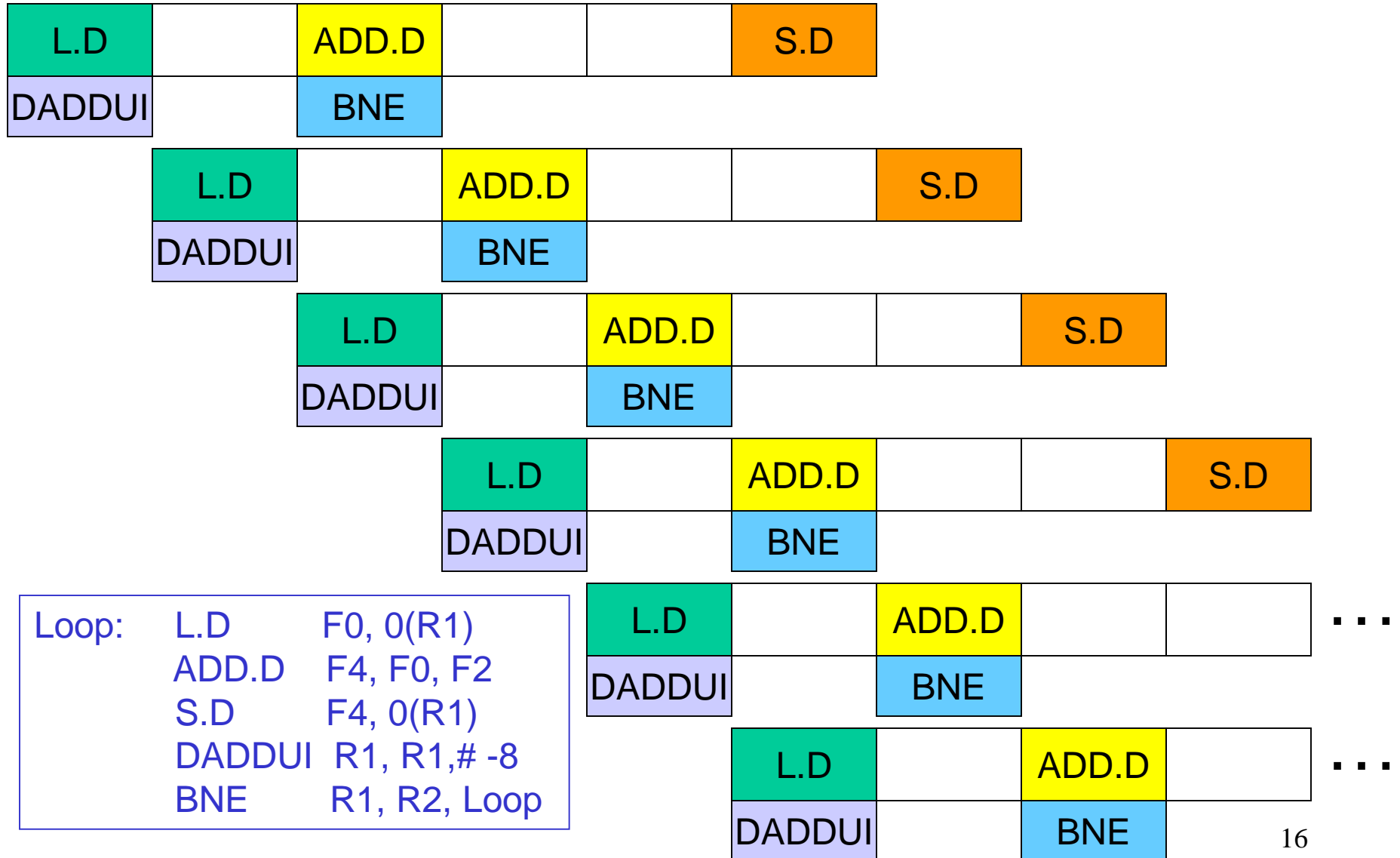
Assembly code

- How many unrolls does it take to avoid stalls in the superscalar pipeline?

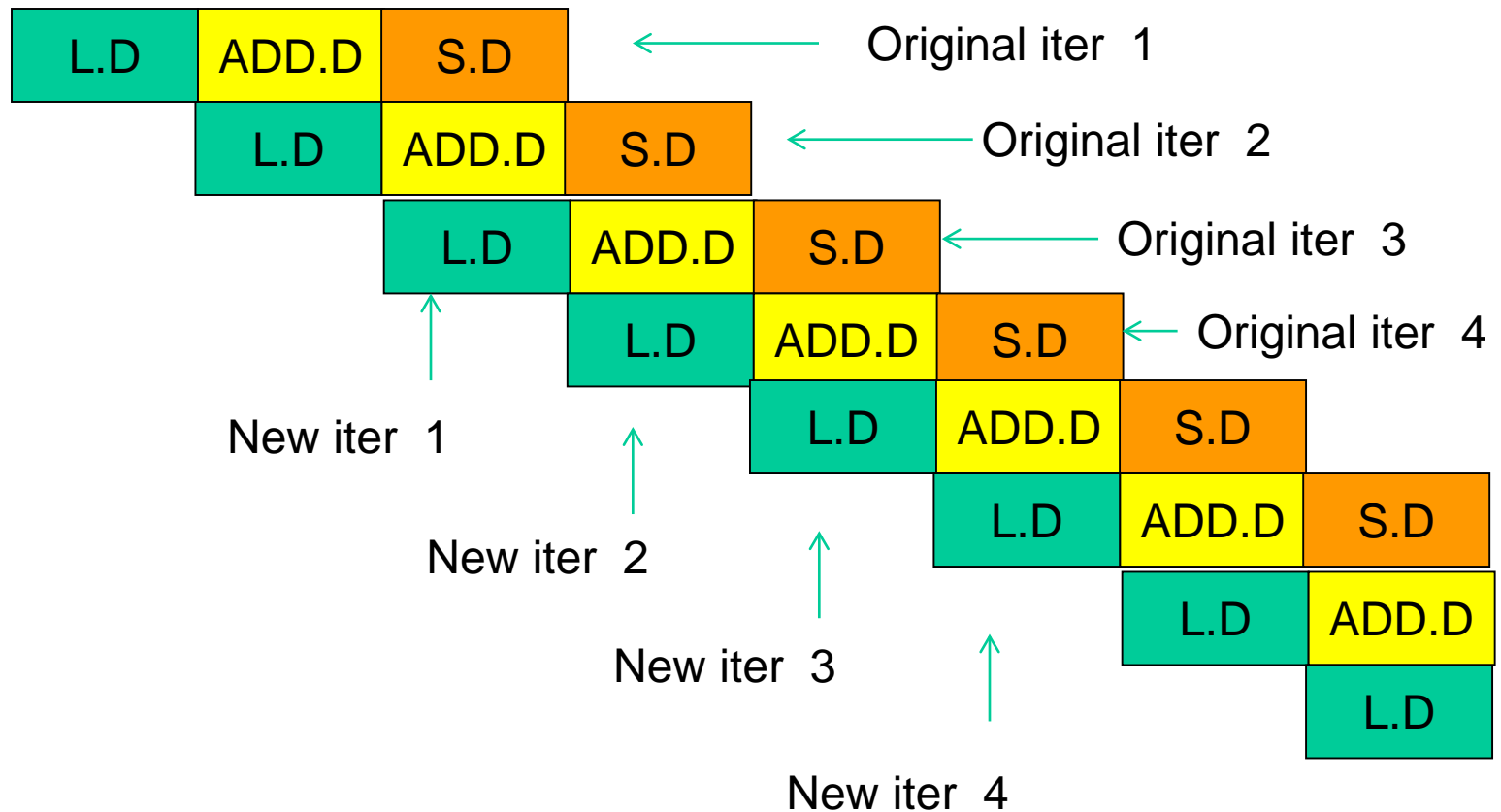
```
LD  
LD  
LD  MUL  
LD  MUL  
LD  MUL  
LD  MUL  
LD  MUL  
SD  MUL
```

7 unrolls. Could also make do with 5 if we moved up the DADDUIs.

Software Pipeline?!



Software Pipeline



Software Pipelining

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



```
Loop:  S.D    F4, 16(R1)
        ADD.D  F4, F0, F2
        L.D    F0, 0(R1)
        DADDUI R1, R1, # -8
        BNE   R1, R2, Loop
```

- Advantages: achieves nearly the same effect as loop unrolling, but without the code expansion – an unrolled loop may have inefficiencies at the start and end of each iteration, while a sw-pipelined loop is almost always in steady state – a sw-pipelined loop can also be unrolled to reduce loop overhead
- Disadvantages: does not reduce loop overhead, may require more registers

Problem 4

LD -> any : 1 stall
FPMUL -> any: 5 stalls
FPMUL -> ST : 4 stalls
IntALU -> BR : 1 stall

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

Source code

```
Loop:  L.D      F0, 0(R1)      ; F0 = array element  
       MUL.D   F4, F0, F2     ; multiply scalar  
       S.D     F4, 0(R2)     ; store result  
       DADDUI  R1, R1, #-8    ; decrement address pointer  
       DADDUI  R2, R2, #-8    ; decrement address pointer  
       BNE     R1, R3, Loop   ; branch if R1 != R3  
       NOP
```

Assembly code

- Show the SW pipelined version of the code and does it cause stalls?

Problem 4

LD -> any : 1 stall
FPMUL -> any: 5 stalls
FPMUL -> ST : 4 stalls
IntALU -> BR : 1 stall

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

Source code

```
Loop:  L.D      F0, 0(R1)      ; F0 = array element  
        MUL.D   F4, F0, F2     ; multiply scalar  
        S.D     F4, 0(R2)     ; store result  
        DADDUI  R1, R1, #-8    ; decrement address pointer  
        DADDUI  R2, R2, #-8    ; decrement address pointer  
        BNE    R1, R3, Loop    ; branch if R1 != R3  
        NOP
```

Assembly code

- Show the SW pipelined version of the code and does it cause stalls?

```
Loop:  S.D     F4, 0(R2)  
        MUL   F4, F0, F2  
        L.D   F0, 0(R1)  
        DADDUI R2, R2, #-8  
        BNE   R1, R3, Loop  
        DADDUI R1, R1, #-8
```

There will be no stalls

Title

- Bullet