Loop Example

x[i] → F0

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

Source code

x[1000] → x[1]

```
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
```

Assembly code

Set R1 → x[1000]
F2 = s
R2 → x[0]

R1 → x[i]
R1-8 → x[i-1]
R1 != x[0] ?

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   nop
```

3 instrs of word

5-line Assembly code

10-cycles

10-cycle schedule

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
```

Assembly code

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

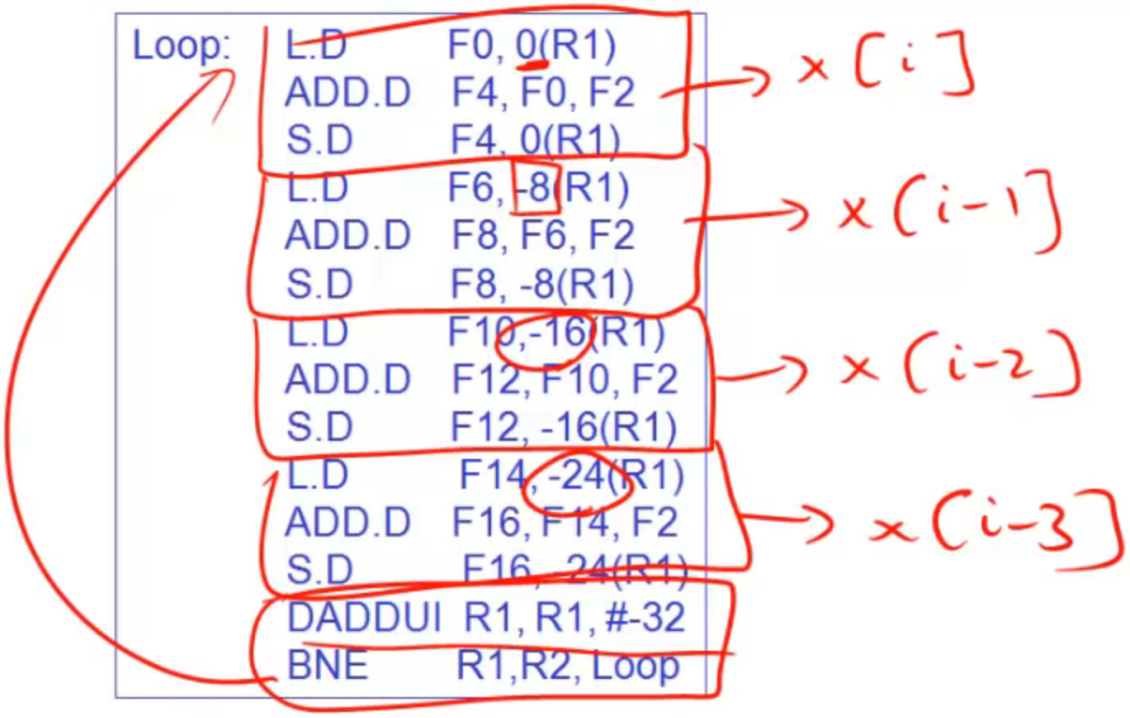
10-cycle schedule

3 lines of work
LD-ADD-SD

5 assembly instrs

10-cycles

Loop Unrolling



- 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)
```

no dep

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

14 cyc
4 iter worth of work
3.5 cyc/iter

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

3 cyc/iter

Loop Unrolling

loop $1 \rightarrow n$

unroll by $k = 4$

- 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 mod k) iterations of the original loop

large loop iter $1 \rightarrow \lfloor \frac{n}{4} \rfloor$

orig small loop $\lfloor \frac{n}{4} \rfloor \times 4 \rightarrow n$

n mod 4 iters

$n = 1007$ iters

$k = 4$

$1 \rightarrow 1004$

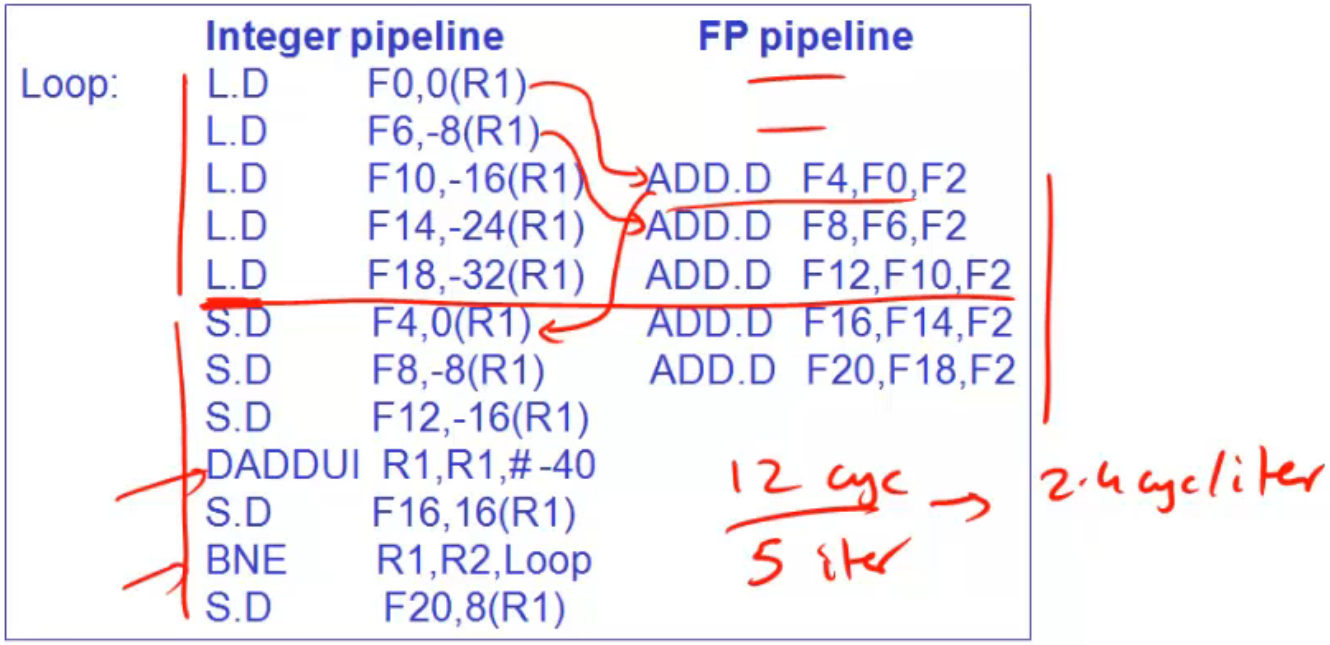
$251 \left[\quad \right]$

3 small iters

↑

n mod k

Superscalar Pipelines



- 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)

Superscalar Pipelines

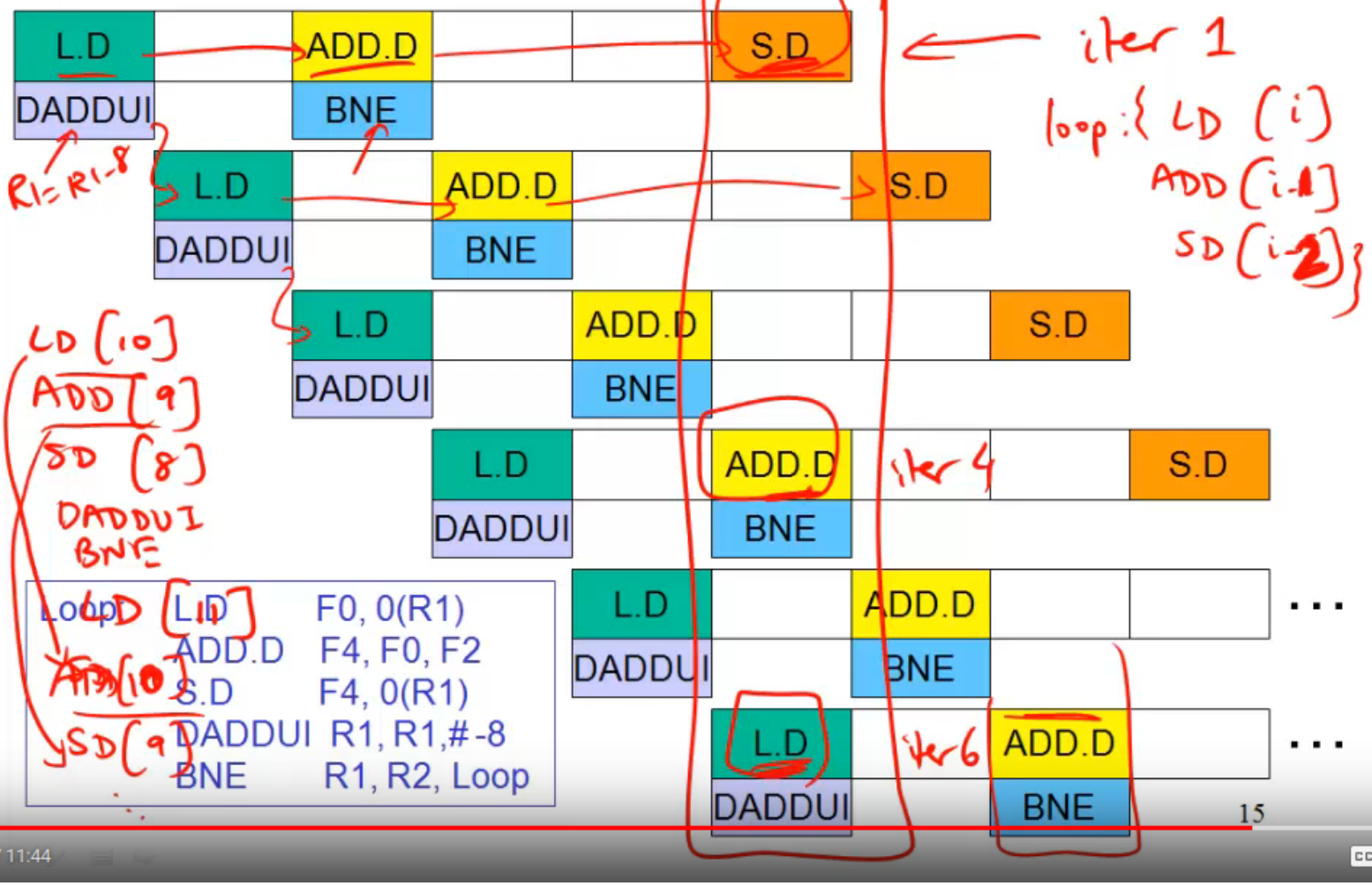
VLIW

	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)	

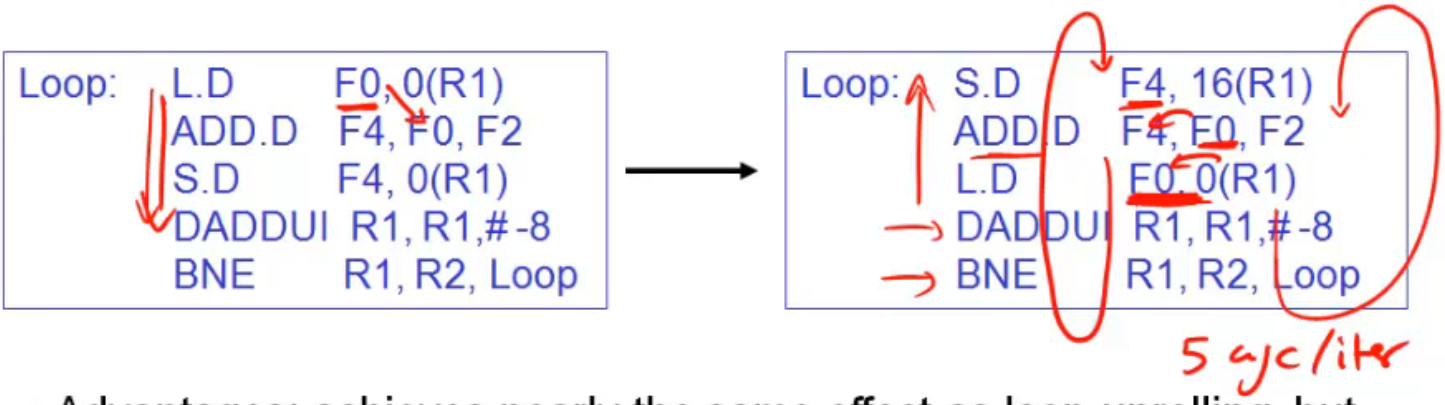
[I1 I2 I3] #
[I5 I6]
[I7]
12 cyc → 5 iter
2-4 cycle iter

- 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)

Software Pipeline?!



Software Pipelining



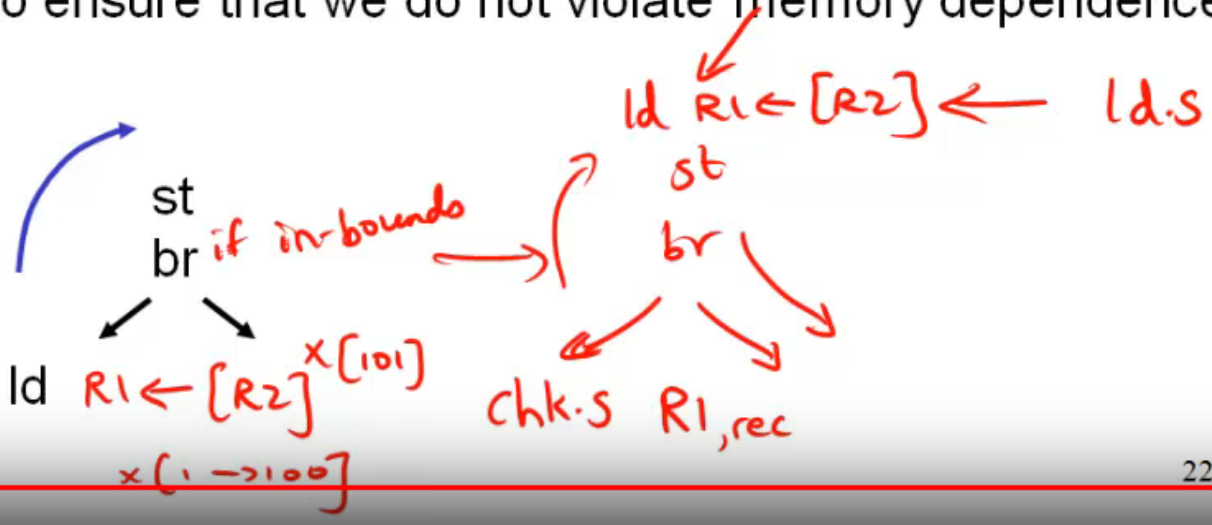
- Advantages: achieves nearly the same effect as loop unrolling, but without the code expansion – an unrolled loop may have inefficiencies at the start and end of each iteration, while a sw-pipelined loop is almost always in steady state – a sw-pipelined loop can also be unrolled to reduce loop overhead

- Disadvantages: does not reduce loop overhead, may require more registers

Support for Speculation

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

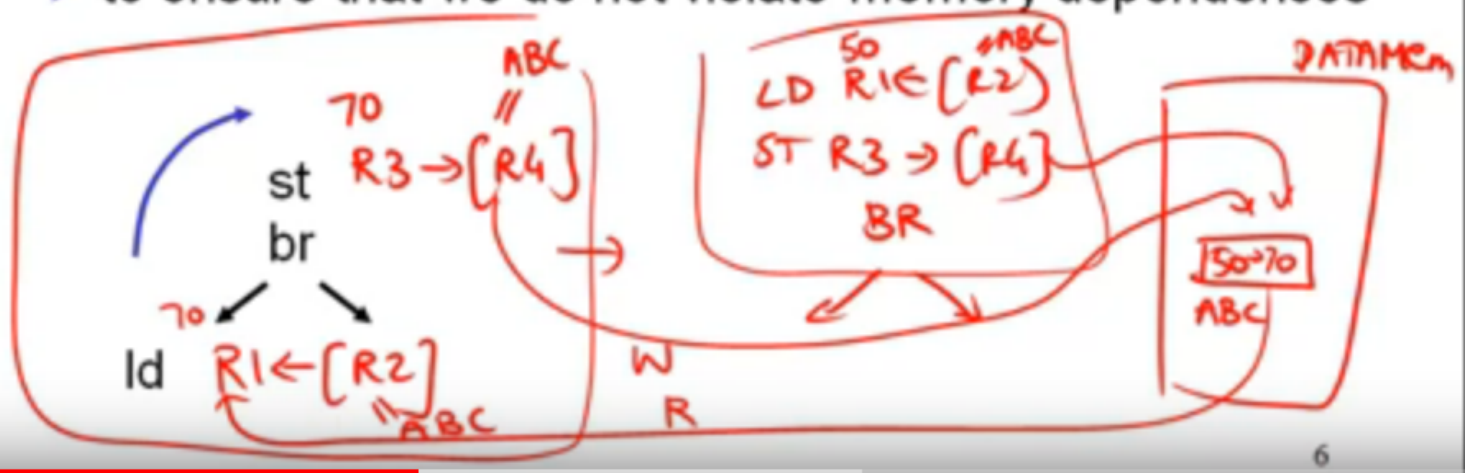
- However, we need hardware support
  - to ensure that an exception is raised at the correct point
  - to ensure that we do not violate memory dependences



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



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