

ILP: COMPILER-BASED TECHNIQUES

Mahdi Nazm Bojnordi

Assistant Professor

School of Computing

University of Utah

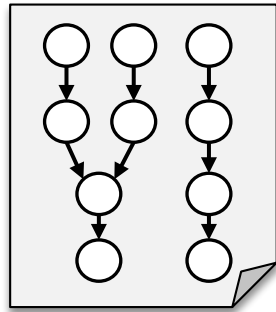
Overview

- Announcements
 - ▣ Homework 2: due on Sept. 18th
- This lecture
 - ▣ Program execution
 - ▣ Loop optimization
 - ▣ Superscalar pipelines
 - ▣ Software pipelining

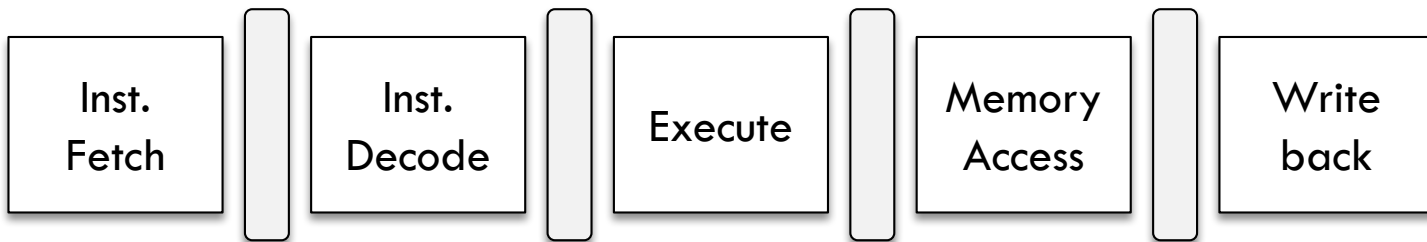
Big Picture

- **Goal:** improving performance

Software (ILP and IC)



Hardware (IPC)

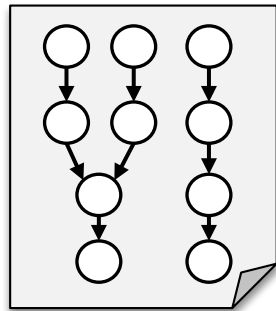


Big Picture

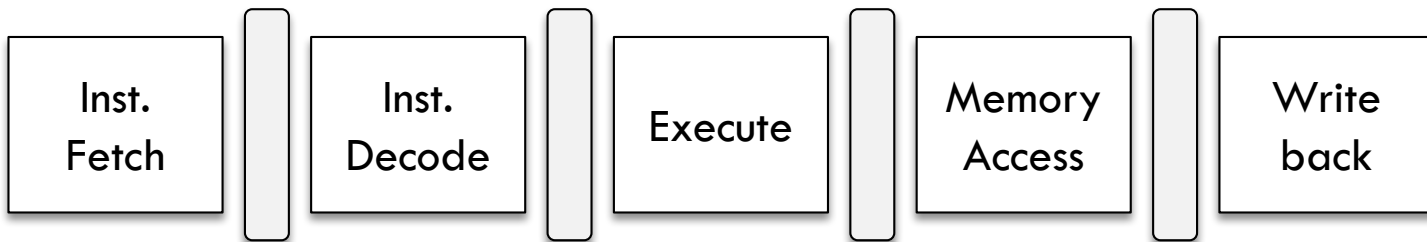
- **Goal:** improving performance

Software (ILP and IC)

$$\text{Performance} = (\text{IPC} \times F) / \text{IC}$$



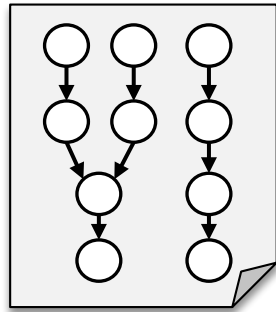
Hardware (IPC)



Big Picture

- **Goal:** improving performance

Software (ILP and IC)



$$\text{Performance} = (\text{IPC} \times \text{F}) / \text{IC}$$

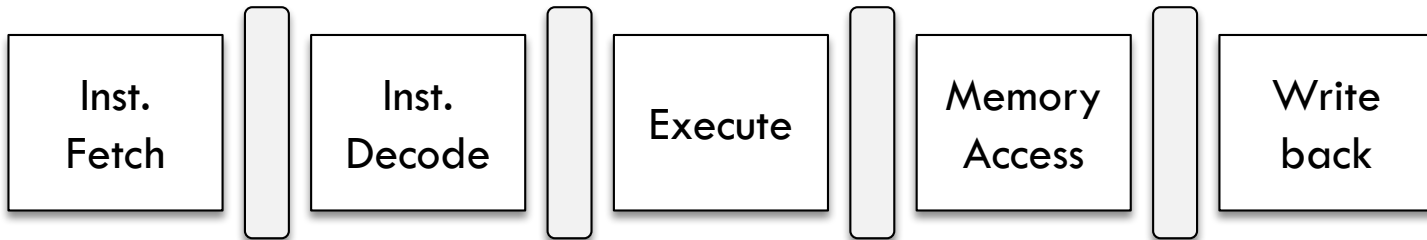
Increasing IPC:

1. Improve ILP
2. Exploit more ILP

Increasing F:

1. Deeper pipeline
2. Faster technology

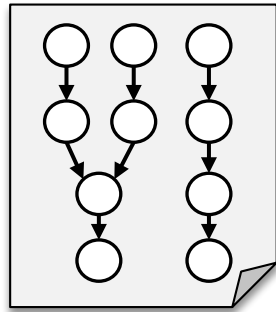
Hardware (IPC)



Big Picture

- **Goal:** improving performance

Software (ILP and IC)



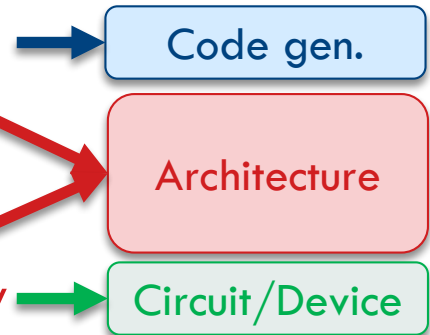
$$\text{Performance} = (\text{IPC} \times F) / \text{IC}$$

Increasing IPC:

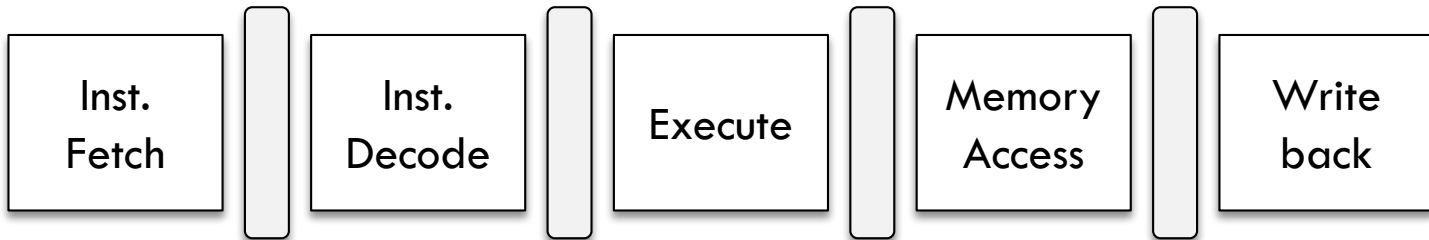
1. Improve ILP
2. Exploit more ILP

Increasing F:

1. Deeper pipeline
2. Faster technology



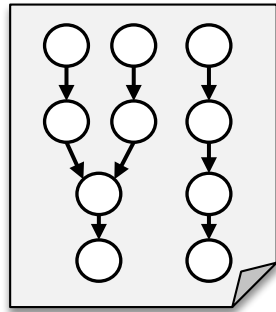
Hardware (IPC)



Big Picture

- **Goal:** improving performance

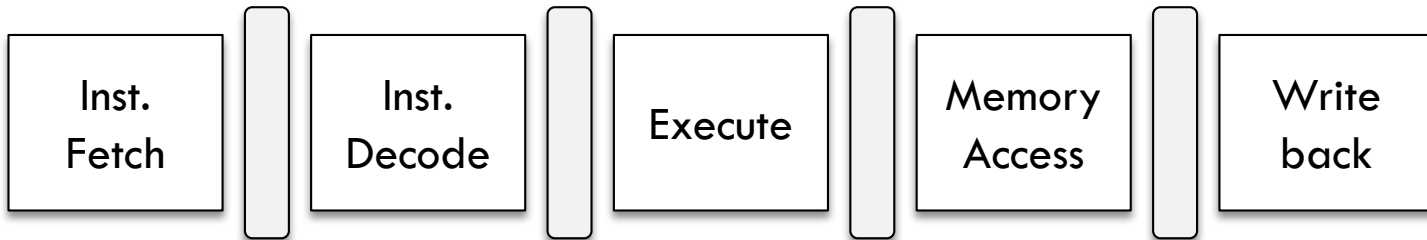
Software (ILP and IC)



Architectural Techniques:

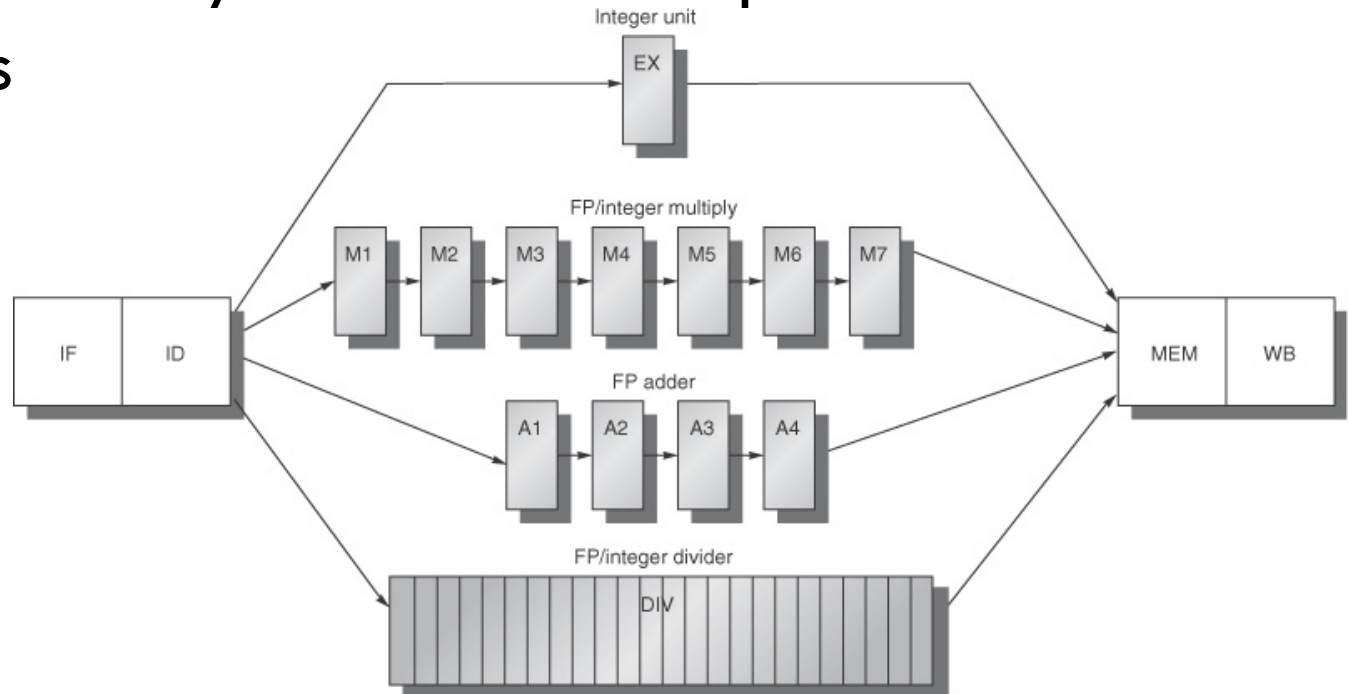
- Deep pipelining
 - Ideal speedup = n times!
- Exploiting ILP
 - Dynamic scheduling (HW)
 - Static scheduling (SW)

Hardware (IPC)



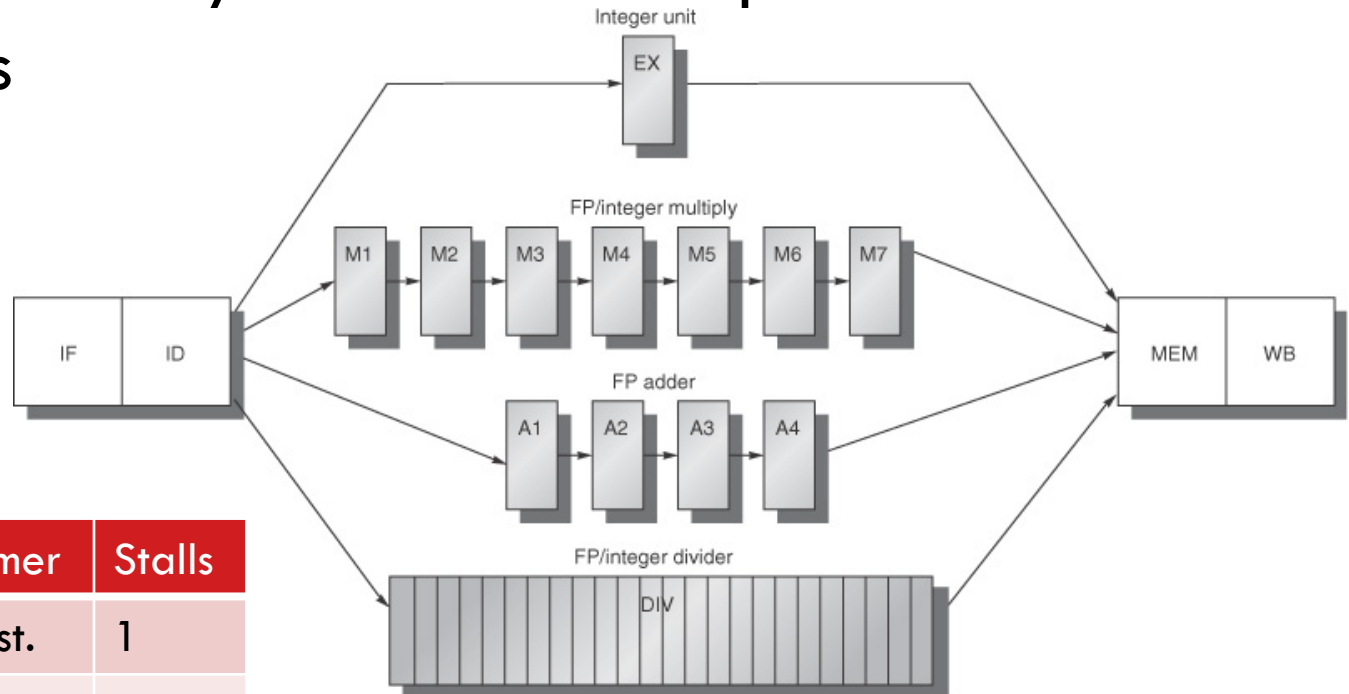
Processor Pipeline

- Necessary stall cycles between dependent instructions



Processor Pipeline

- Necessary stall cycles between dependent instructions



© 2007 Elsevier, Inc. All rights reserved.

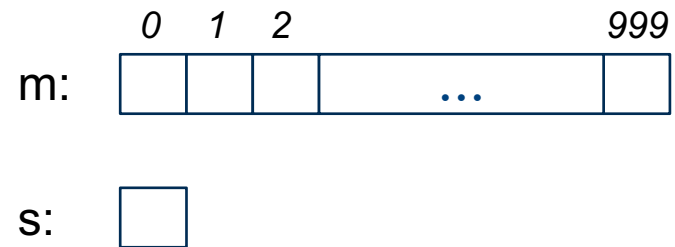
Producer	Consumer	Stalls
Load	Any Inst.	1
fp.ALU	Store	2
fp.ALU	Any other	3
int.ALU	Branch	1

Program

- Loop book-keeping overheads

Producer	Consumer	Stalls
Load	Any Inst.	1
fp.ALU	Store	2
fp.ALU	Any other	3
int.ALU	Branch	1

Goal: adding s to all of the array elements



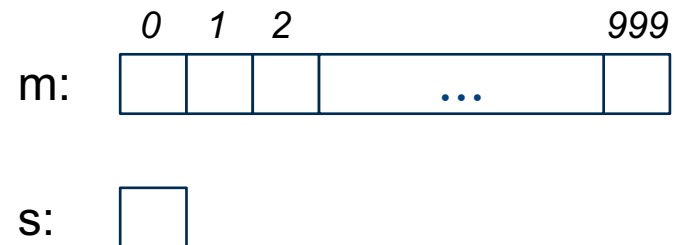
Program

□ Loop book-keeping overheads

```
do {  
    m[i] = m[i] + s;  
    i = i - 1;  
} while(i != j)
```

Producer	Consumer	Stalls
Load	Any Inst.	1
fp.ALU	Store	2
fp.ALU	Any other	3
int.ALU	Branch	1

Goal: adding s to all of the array elements



Program

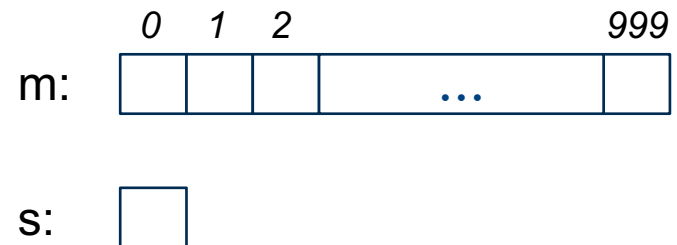
□ Loop book-keeping overheads

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

```
do {
    m[i] = m[i] + s;
    i = i - 1;
} while(i != j)
```

Producer	Consumer	Stalls
Load	Any Inst.	1
fp.ALU	Store	2
fp.ALU	Any other	3
int.ALU	Branch	1

Goal: adding s to all of the array elements



Execution Schedule

- Diverse impact of stall cycles on performance

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

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

Producer	Consumer	Stalls
Load	Any Inst.	1
fp.ALU	Store	2
fp.ALU	Any other	3
int.ALU	Branch	1

Schedule 1:

5 stall cycles

3 loop body instructions

2 loop counter instructions

Loop Optimization

Loop Optimization

- Re-ordering and changing immediate values

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

Schedule 1:

5 stall cycles

3 loop body instructions

2 loop counter instructions

Loop Optimization

- Re-ordering and changing immediate values

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

Schedule 2:

- 1 stall cycle
- 3 loop body instructions
- 2 loop counter instructions

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

Schedule 1:

- 5 stall cycles
- 3 loop body instructions
- 2 loop counter instructions

Loop Unrolling

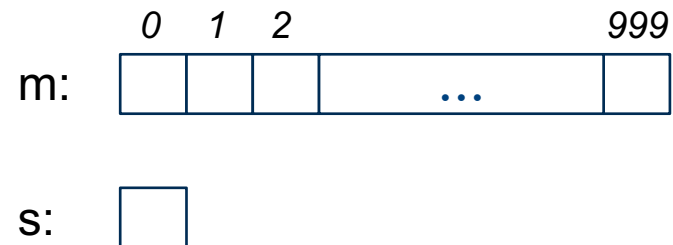
- Reducing loop overhead by unrolling

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

Schedule 2:

- 1 stall cycle
- 3 loop body instructions
- 2 loop counter instructions

Goal: adding s to all of the array elements



Loop Unrolling

- Reducing loop overhead by unrolling

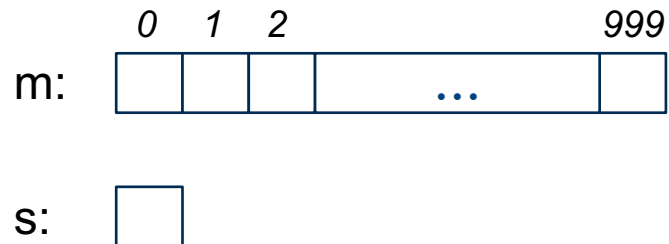
```
Loop:  L.D    F0, 0(R1)
        ADDI.D R1, R1, #-8
        ADD.D  F4, F0, F2
        stall
        BNE   R1, R2, Loop
        S.D   F4, 8(R1)
```

```
do {
    m[i-0] = m[i-0] + s;
    m[i-1] = m[i-1] + s;
    m[i-2] = m[i-2] + s;
    m[i-3] = m[i-3] + s;
    i = i-4;
} while(i != j)
```

Goal: adding s to all of the array elements

Schedule 2:

- 1 stall cycle
- 3 loop body instructions
- 2 loop counter instructions



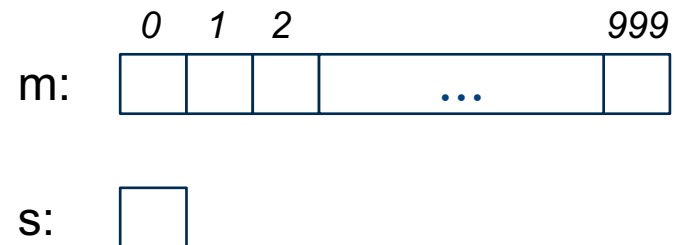
Loop Unrolling

- Reducing loop overhead by 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)
      ADDI.D  R1, R1, #-32
      BNE    R1,R2, Loop
```

```
do {
    m[i-0] = m[i-0] + s;
    m[i-1] = m[i-1] + s;
    m[i-2] = m[i-2] + s;
    m[i-3] = m[i-3] + s;
    i = i-4;
} while(i != j)
```

Goal: adding s to all of the array elements



Loop Unrolling

- Reducing loop overhead by 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)
        ADDI.D  R1, R1, #-32
        BNE    R1,R2, Loop
```

Schedule 3:

14 stall cycles

12 loop body instructions

2 loop counter instructions

Instruction Reordering

- Eliminating stall cycles by unrolling and scheduling

```
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)
        ADDI.D  R1, R1, #-32
        BNE    R1,R2, Loop
        S.D     R1,R2, 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)
        ADDI.D  R1, R1, #-32
        S.D     F12, 16(R1)
        BNE    R1,R2, Loop
        S.D     F16, 8(R1)
```

IPC Limit

- Eliminating stall cycles by unrolling and scheduling

Schedule 4:

0 stall cycles

12 loop body instructions

2 loop counter instructions

+ IPC = 1

- more instructions

- more registers

```
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)
       ADDI.D R1, R1, #-32
       S.D    F12, 16(R1)
       BNE   R1,R2, Loop
       S.D    F16, 8(R1)
```

IPC Limit

- Eliminating stall cycles by unrolling and scheduling

Schedule 4:

0 stall cycles

12 loop body instructions

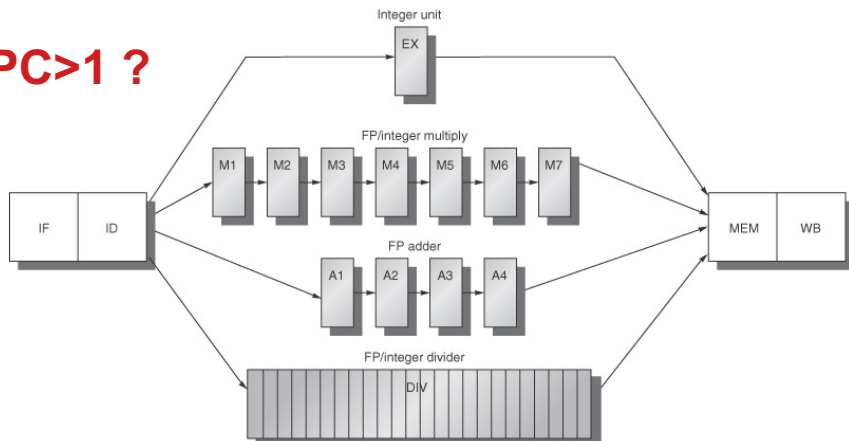
2 loop counter instructions

+ IPC = 1

- more instructions

- more registers

IPC > 1 ?



```
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)
       ADDI.D R1, R1, #-32
       S.D    F12, 16(R1)
       BNE   R1,R2, Loop
       S.D    F16, 8(R1)
```

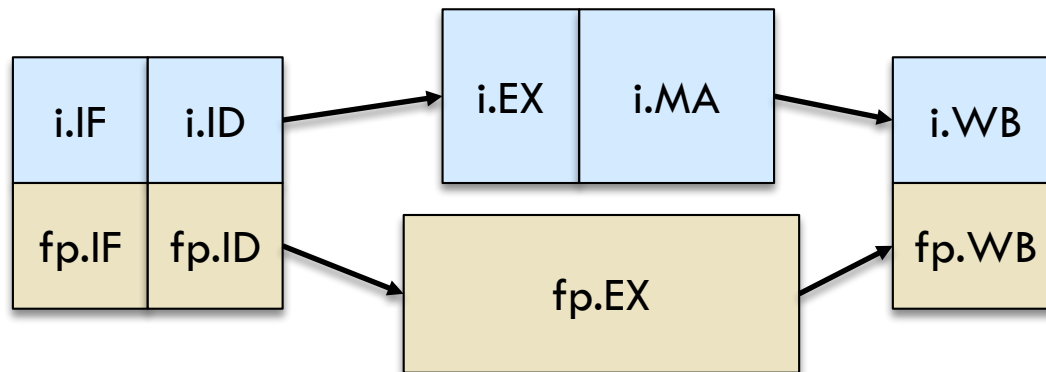
Summary of Scalar Pipelines

- Upper bound on throughput
 - ▣ $IPC \leq 1$
- Unified pipeline for all functional units
 - ▣ Underutilized resources
- Inefficient freeze policy
 - ▣ A stall cycle delays all the following cycles
- Pipeline hazards
 - ▣ Stall cycles result in limited throughput

Superscalar Pipelines

Superscalar Pipelines

- Separate integer and floating point pipelines
 - ▣ An instruction packet is fetched every cycle
 - Very large instruction word (VLIW)
 - ▣ Inst. packet has one fp. and one int. slots
 - ▣ Compiler's job is to find instructions for the slots
 - ▣ $IPC \leq 2$



Superscalar Pipelines

- Forming instruction packets

```
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)
        ADDI.D R1, R1, #-32
        S.D    F12, 16(R1)
        BNE   R1, R2, Loop
        S.D    F16, 8(R1)
```

**Floating-point
operations**

Superscalar Pipelines

- Ideally, the number of empty slots is zero

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

```
NOP
NOP
ADD.D  F4, F0, F2
ADD.D  F8, F6, F2
ADD.D  F12, F10, F2
ADD.D  F16, F14, F2
NOP
NOP
NOP
NOP
```

Superscalar Pipelines

- Ideally, the number of empty slots is zero

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

```
NOP
NOP
ADD.D  F4, F0, F2
ADD.D  F8, F6, F2
ADD.D  F12, F10, F2
ADD.D  F16, F14, F2
NOP
NOP
NOP
NOP
```

Schedule 5:

0 stall cycles

8 loop body packets

2 loop overhead cycles

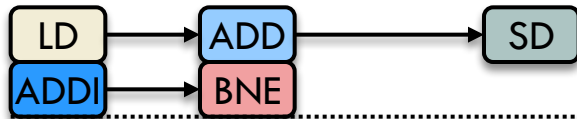
IPC = 1.4

Software Pipelining

Software Pipelining

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

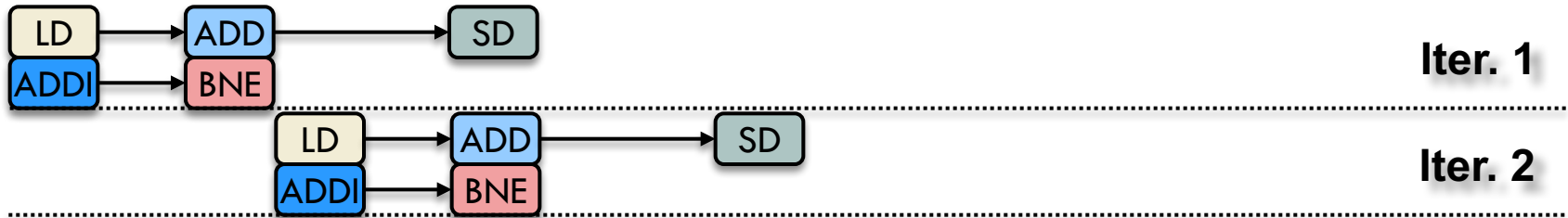
Software Pipelining



Iter. 1

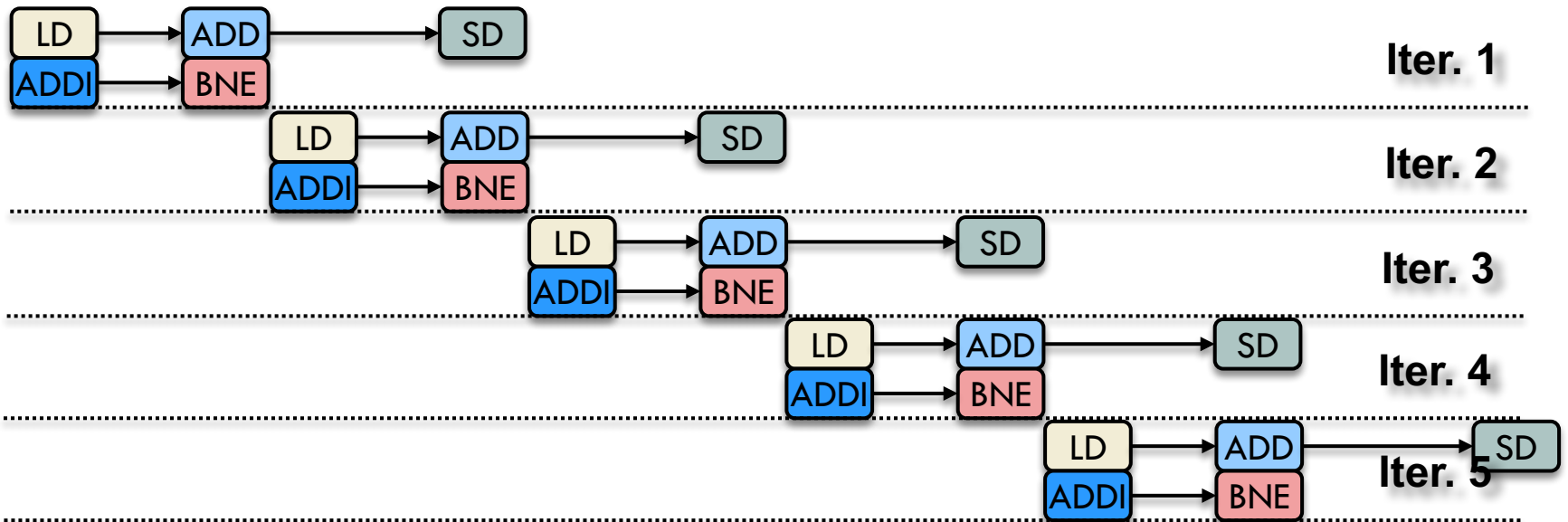
```
Loop:  L.D      F0, 0(R1)
        stall
        ADD.D   F4, F0, F2
        stall
        stall
        S.D     F4, 0(R1)
        ADDI.D  R1, R1, #-8
        stall
        BNE     R1, R2, Loop
        stall
```


Software Pipelining

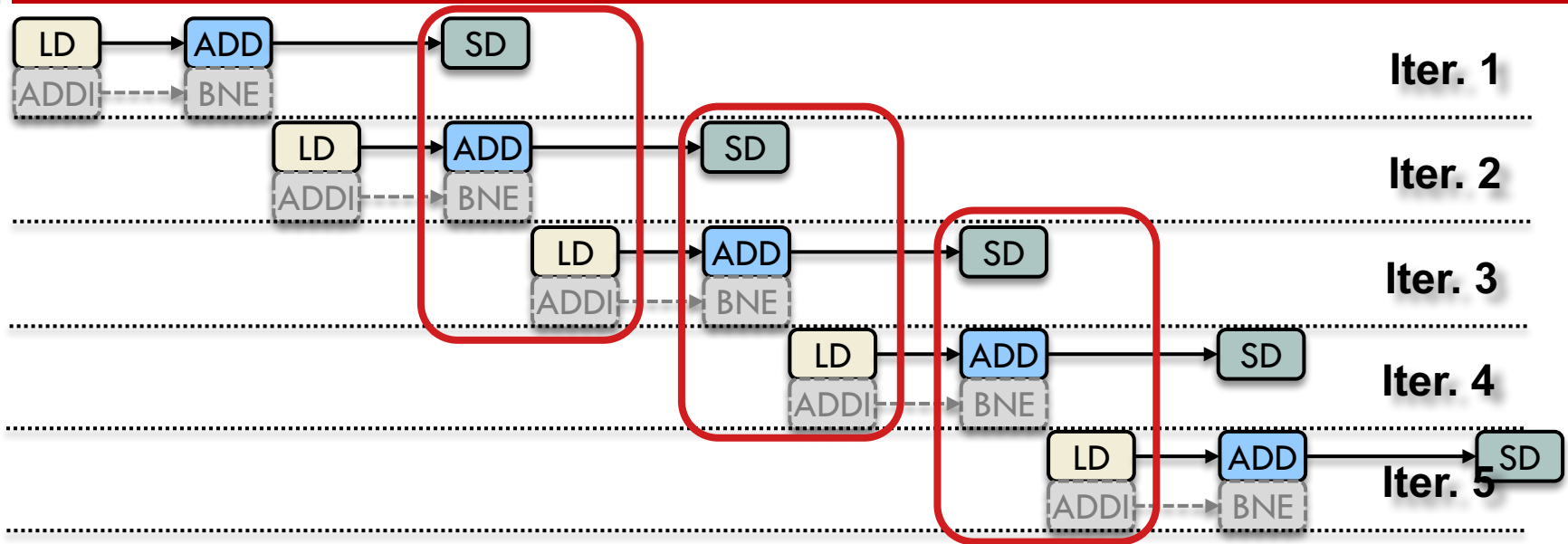


```
Loop:  L.D      F0, 0(R1)
      stall
      ADD.D   F4, F0, F2
      stall
      stall
      S.D     F4, 0(R1)
      ADDI.D  R1, R1, #-8
      stall
      BNE    R1, R2, Loop
      stall
```

Software Pipelining



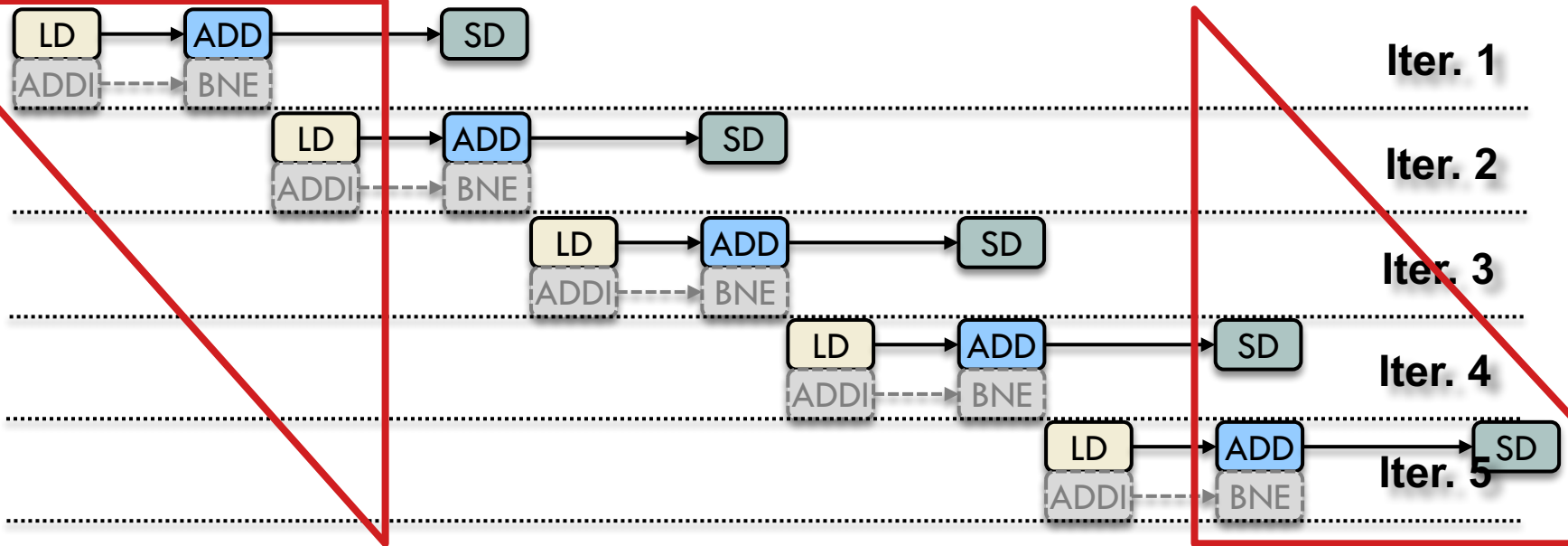
Software Pipelining



loop: S.D (1)
 ADD.D (2)
 L.D (3)
 ADDI.D
 BNE

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

Software Pipelining



Prologue and Epilogue?