

Computer Architecture & Networks

CHAPTER 4

FUNCTIONAL ORGANIZATION

LEARNING OUTCOMES

To learn and understand ;

1. Instruction Pipelining
2. Introduction to Instruction-Level Parallelism (ILP)

Today.....

Instruction Pipelining and Instruction-level Parallelism

- Pipeline Strategy
- Pipeline Performance
- Pipeline hazards
- Dealing with branches
- Superscalar Systems
- Instruction-level Parallelism

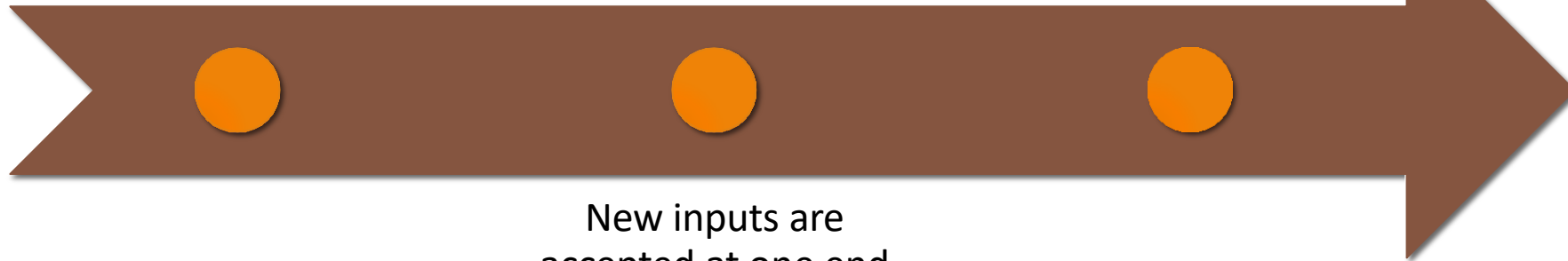
Pipelining Strategy

类似于在制造厂中使用装配线

Similar to the use of an assembly line in a manufacturing plant

To apply this concept to instruction execution we must recognize that an instruction has a number of stages

要将此概念应用于指令执行，我们必须认识到指令有多个阶段



New inputs are accepted at one end before previously accepted inputs appear as outputs at the other end

在先前接受的输入显示为另一端的输出之前，在一端接受新输入

Pipelining

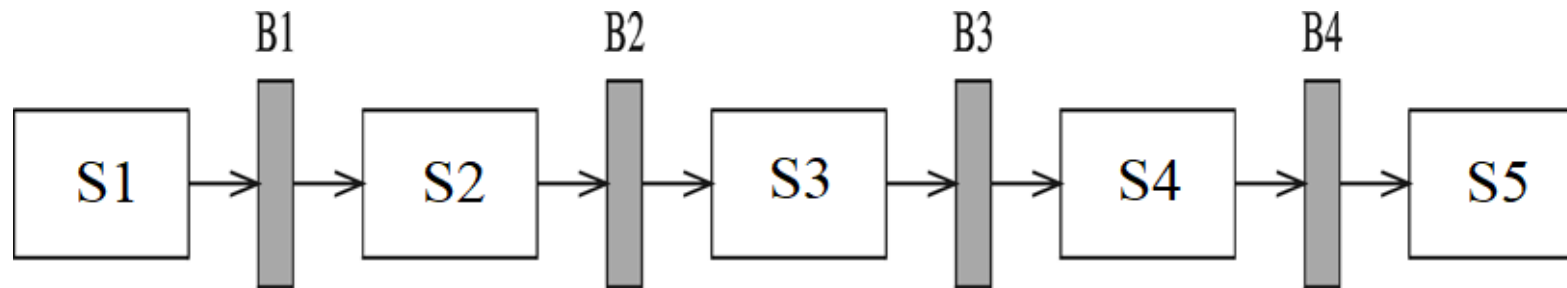
Pipelining overlaps the execution of multiple instructions in parallel 流水线重叠并行执行多条指令

Data path is divided into **stages** which all operate simultaneously 数据路径分为多个阶段，所有阶段同时运行

Pipeline stages ideally have to have the same time (**clock cycle**) 理想情况下，管道级必须具有相同的时间（时钟周期）

Pipeline **buffers (latches)** prevent data entering next stage before previous data has left
管道缓冲区（锁存器）防止数据在前一个数据离开前进入下一个阶段

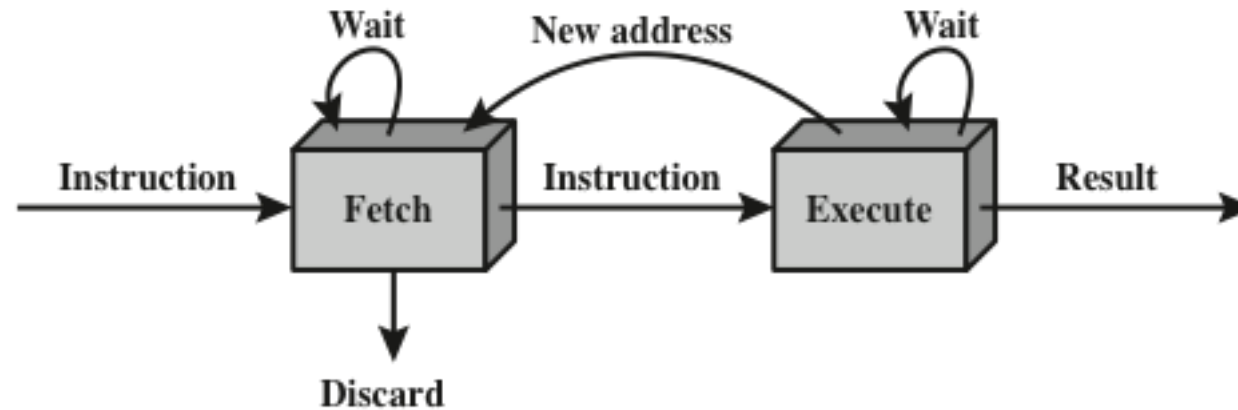
Increasing the number of stages allows higher number of overlapped instructions but also increases the total pipeline latch latency 增加阶段数允许更多的重叠指令，但也会增加总的管道锁存延迟



Two-Stage Instruction Pipeline



(a) Simplified view



(b) Expanded view

Figure 14.9 Two-Stage Instruction Pipeline

Stage Step

FI > DI > CO > FO > EI > WO

1 + 2 = 3, operand可以是1和2

Additional Stages

Fetch instruction (FI)

- Read the next expected instruction into a buffer 将下一条预期指令读入缓冲区

Decode instruction (DI)

- Determine the opcode and the operand specifiers 确定操作码和操作数说明符

Calculate operands (CO)

- Calculate the effective address of each source operand 计算每个源操作数的有效地址
- This may involve displacement, register indirect, indirect, or other forms of address calculation 这可能涉及移位、寄存器间接、间接或其他形式的地址计算

Fetch operands (FO)

- Fetch each operand from memory 从内存中提取每个操作数
- Operands in registers need not be fetched 不需要提取寄存器中的操作数

Execute instruction (EI)

- Perform the indicated operation and store the result, if any, in the specified destination operand location 执行指示的操作并将结果（如果有）存储在指定的目标操作数位置

Write operand (WO)

- Store the result in memory 将结果存储在内存中

Six-stage instruction pipeline

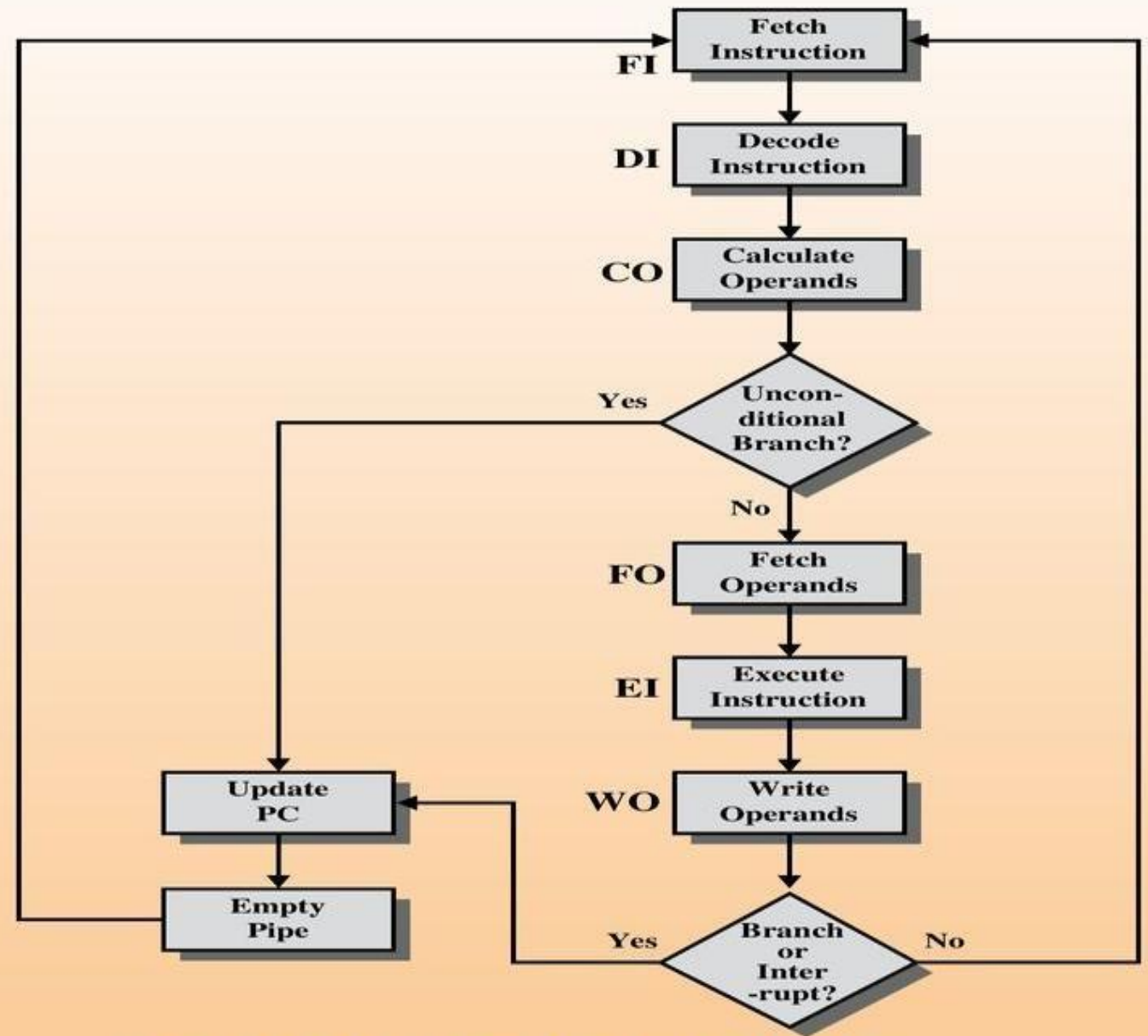


Figure 14.12 Six-Stage Instruction Pipeline

Timing Diagram for Instruction Pipeline Operation

Time →

Cycle →

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO								
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO						
Instruction 4				FI	DI	CO	FO	EI	WO					
Instruction 5					FI	DI	CO	FO	EI	WO				
Instruction 6						FI	DI	CO	FO	EI	WO			
Instruction 7							FI	DI	CO	FO	EI	WO		
Instruction 8								FI	DI	CO	FO	EI	WO	
Instruction 9									FI	DI	CO	FO	EI	WO

Figure 14.10 Timing Diagram for Instruction Pipeline Operation

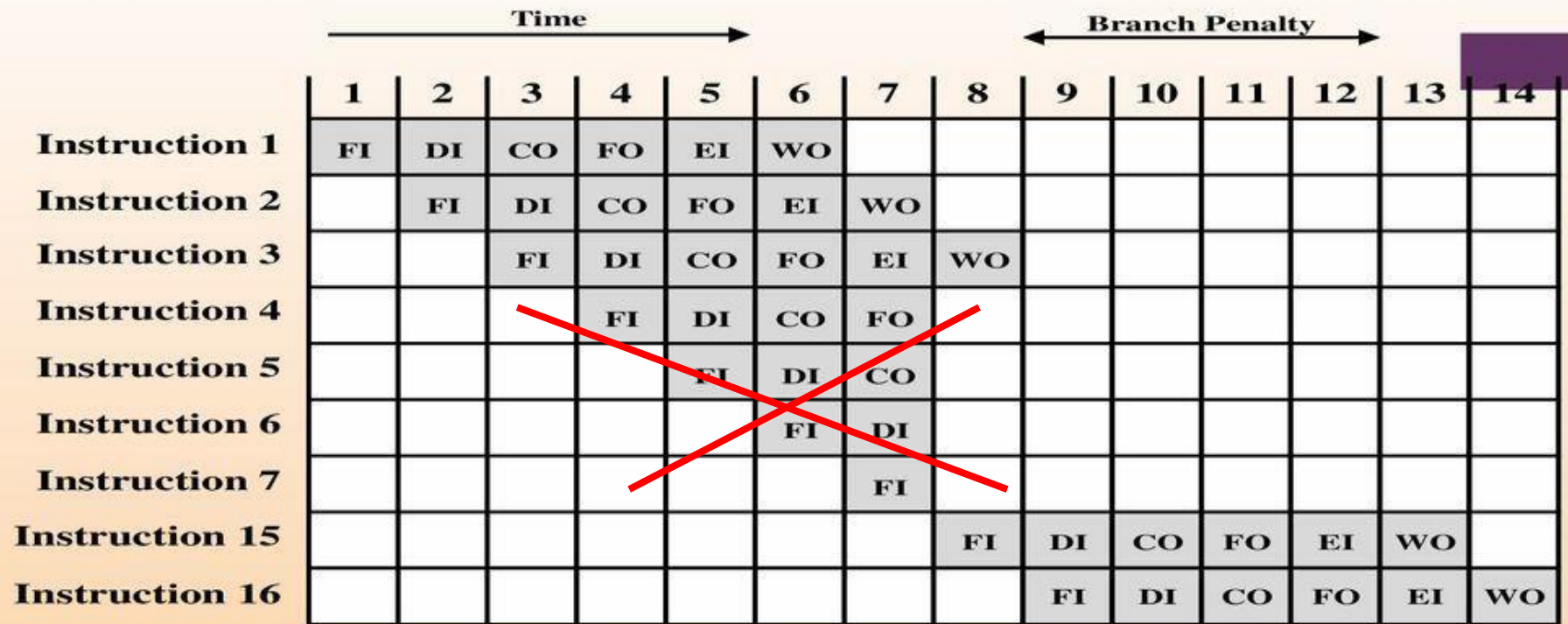


Figure 14.11 The Effect of a Conditional Branch on Instruction Pipeline Operation

Another difficulty is the conditional branch instruction, which can invalidate several instruction fetches. A similar unpredictable event is an interrupt.

Figure 14.11 illustrates the effects of the conditional branch, using the same program as Figure 14.10. Assume that instruction 3 is a conditional branch to instruction 15. Until the instruction is executed, there is no way of knowing which instruction will come next. The pipeline, in this example, simply loads the next instruction in sequence (instruction 4) and proceeds. The branch is not determined until the end of time unit 7. At this point, the pipeline must be cleared of instructions that are not useful. During time unit 8, instruction 15 enters the pipeline. No instructions complete during time units 9 through 12; this is the performance penalty incurred because we could not anticipate the branch.

Time
↓

	FI	DI	CO	FO	EI	WO
1	I1					
2	I2	I1				
3	I3	I2	I1			
4	I4	I3	I2	I1		
5	I5	I4	I3	I2	I1	
6	I6	I5	I4	I3	I2	I1
7	I7	I6	I5	I4	I3	I2
8	I8	I7	I6	I5	I4	I3
9	I9	I8	I7	I6	I5	I4
10		I9	I8	I7	I6	I5
11			I9	I8	I7	I6
12				I9	I8	I7
13					I9	I8
14						I9

(a) No branches

	FI	DI	CO	FO	EI	WO
1	I1					
2	I2	I1				
3	I3	I2	I1			
4	I4	I3	I2	I1		
5	I5	I4	I3	I2	I1	
6	I6	I5	I4	I3	I2	I1
7	I7	I6	I5	I4	I3	I2
8	I15					I3
9	I16	I15				
10		I16	I15			
11			I16	I15		
12				I16	I15	
13					I16	I15
14						I16

(b) With conditional branch

Figure 14.13 An Alternative Pipeline Depiction

In Figure 14.13a (which corresponds to Figure 14.10), the pipeline is full at time 6, with 6 different instructions in various stages of execution, and remains full through time 9; we assume that instruction I9 is the last instruction to be executed. In Figure 14.13b, (which corresponds to Figure 14.11), the pipeline is full at times 6 and 7. At time 7, instruction 3 is in the execute stage and executes a branch to instruction 15. At this point, instructions I4 through I7 are flushed from the pipeline, so that at time 8, only two instructions are in the pipeline, I3 and I15.

Speedup Factors with Instruction Pipelining

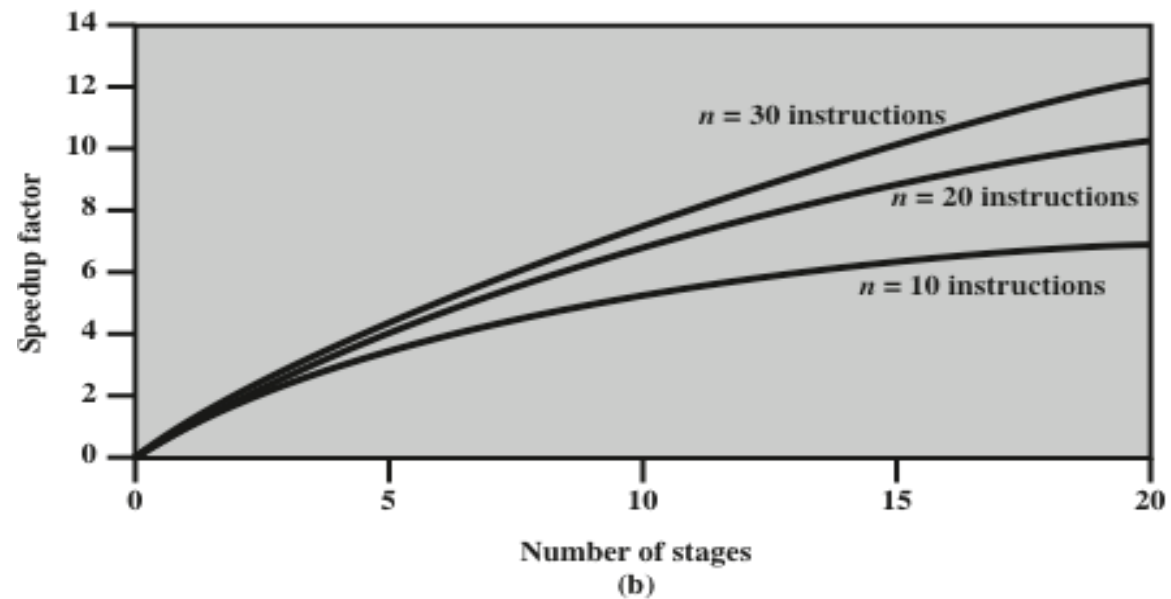
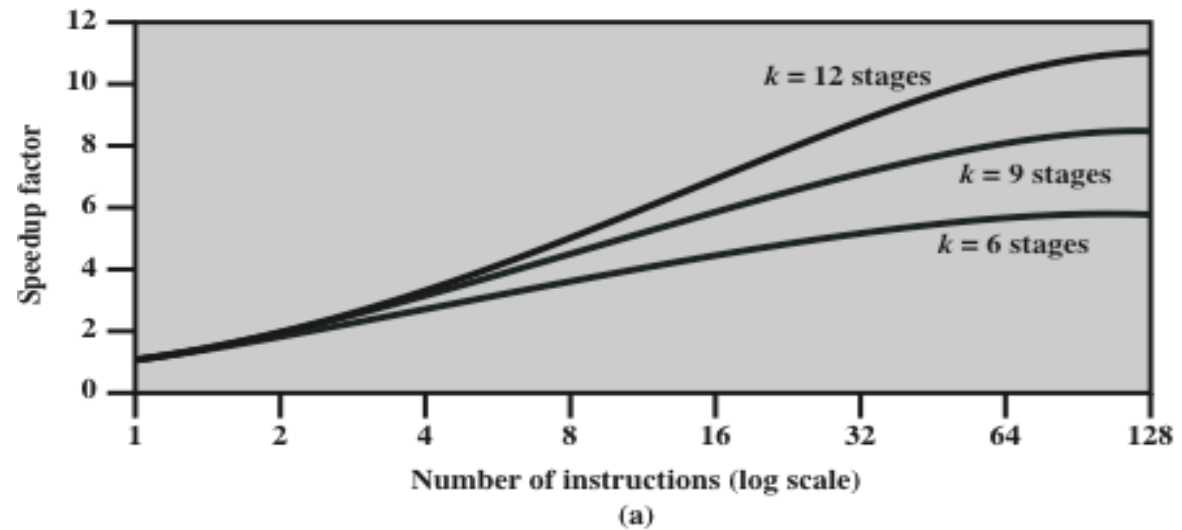


Figure 14.14 Speedup Factors with Instruction Pipelining

Pipeline Hazards

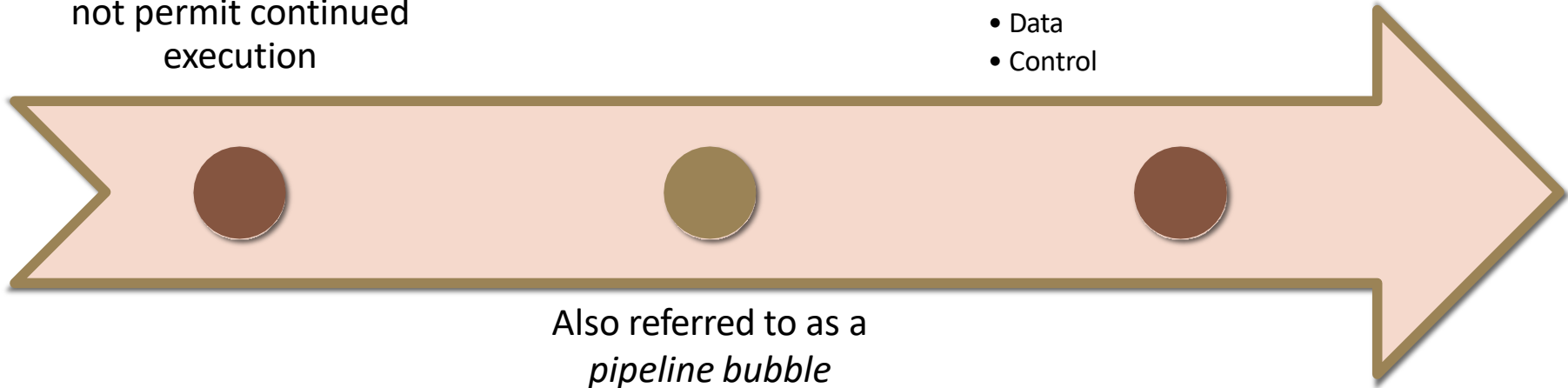
由于条件不允许
继续执行，管道
或部分管道必须
停止时发生

Occur when the pipeline,
or some portion of the
pipeline, must stall
because conditions do
not permit continued
execution

有三种类型的危害：资源、数据、控制

There are three types of
hazards:

- Resource
- Data
- Control



Also referred to as a
pipeline bubble
也称为管道气泡

Resource Hazards

A resource hazard occurs when two or more instructions that are already in the pipeline need the same resource

The result is that the instructions must be executed in serial rather than parallel for a portion of the pipeline

A resource hazard is sometimes referred to as a **structural hazard**

当已经在管道中的两个或多个指令需要相同的资源时，就会发生资源危险

结果是，对于管道的一部分，指令必须串行执行，而不是并行执行

资源危害有时被称为结构危害

	Clock cycle								
	1	2	3	4	5	6	7	8	9
	I1	FI	DI	FO	EI	WO			
	I2		FI	DI	FO	EI	WO		
	I3			FI	DI	FO	EI	WO	
	I4				FI	DI	FO	EI	WO

(a) Five-stage pipeline, ideal case

	Clock cycle								
	1	2	3	4	5	6	7	8	9
	I1	FI	DI	FO	EI	WO			
	I2		FI	DI	FO	EI	WO		
	I3			Idle	FI	DI	FO	EI	WO
	I4					FI	DI	FO	EI

(b) I1 source operand in memory

Have to wait one clock cycle before it can start

Figure 14.15 Example of Resource Hazard

Data Hazards 数据危害

A data hazard occurs when there is a conflict in the access of an operand location

当操作数位置的访问发生冲突时，会发生数据危险

		Clock cycle									
		1	2	3	4	5	6	7	8	9	10
ADD EAX, EBX		FI	DI	FO	EI	WO					
SUB ECX, EAX			FI	DI	Idle		FO	EI	WO		
I3				FI			DI	FO	EI	WO	
I4							FI	DI	FO	EI	WO

Figure 14.16 Example of Data Hazard

Types of Data Hazard

Read after write (RAW), or true dependency

- An instruction modifies a register or memory location
- Succeeding instruction reads data in memory or register location
- Hazard occurs if the read takes place before write operation is complete

Write after read (WAR), or antidependency

- An instruction reads a register or memory location
- Succeeding instruction writes to the location
- Hazard occurs if the write operation completes before the read operation takes place

Write after write (WAW), or output dependency

- Two instructions both write to the same location
- Hazard occurs if the write operations take place in the reverse order of the intended sequence

Control Hazard 控制危险

Also known as a **branch hazard**

Occurs when the pipeline makes the wrong decision on a branch prediction 当管道对分支预测做出错误决策时发生

Brings instructions into the pipeline that must subsequently be discarded Dealing with Branches:

将随后必须丢弃的指令放入管道中处理分支：

- Multiple streams
 - Prefetch branch target
 - Loop buffer
 - Branch prediction
 - Delayed branch
- 多流
预取分支目标
循环缓冲区
分支预测
延迟分支

Multiple Streams 多流

A simple pipeline suffers a penalty for a branch instruction because it must choose one of two instructions to fetch next and may make the wrong choice

简单管道会因分支指令而受到惩罚，因为它必须从两条指令中选择一条来获取下一条指令，并且可能会做出错误的选择

蛮力方法是复制管道的初始部分，并允许管道使用两个流获取两条指令

A brute-force approach is to replicate the initial portions of the pipeline and allow the pipeline to fetch both instructions, making use of two streams

Drawbacks:

- With multiple pipelines there are contention delays for access to the registers and to memory
- Additional branch instructions may enter the pipeline before the original branch decision is resolved

缺点：

- 对于多个管道，存在访问的争用延迟将寄存器和内存连接起来
- 其他分支指令可能会在原始分支决策已解决

Prefetch Branch Target

预取分支目标

- 识别条件分支时，除了分支后面的指令外，还将预取分支的目标
- When a conditional branch is recognized, the target of the branch is prefetched, in addition to the instruction following the branch
- Target is then saved until the branch instruction is executed
 - 然后将目标保存到分支指令被执行
- If the branch is taken, the target has already been prefetched
 - 如果执行了分支，则目标已执行已预取
- IBM 360/91 uses this approach
 - IBM360/91使用这种方法

Loop Buffer

Small, very-high speed memory maintained by the instruction fetch stage of the pipeline and containing the n most recently fetched instructions, in sequence

Benefits:

- Instructions fetched in sequence will be available without the usual memory access time
- If a branch occurs to a target just a few locations ahead of the address of the branch instruction, the target will already be in the buffer
- This strategy is particularly well suited to dealing with loops
- Similar in principle to a cache dedicated to instructions
- **Differences:**
 - The loop buffer only retains instructions in sequence
 - Is much smaller in size and hence lower in cost

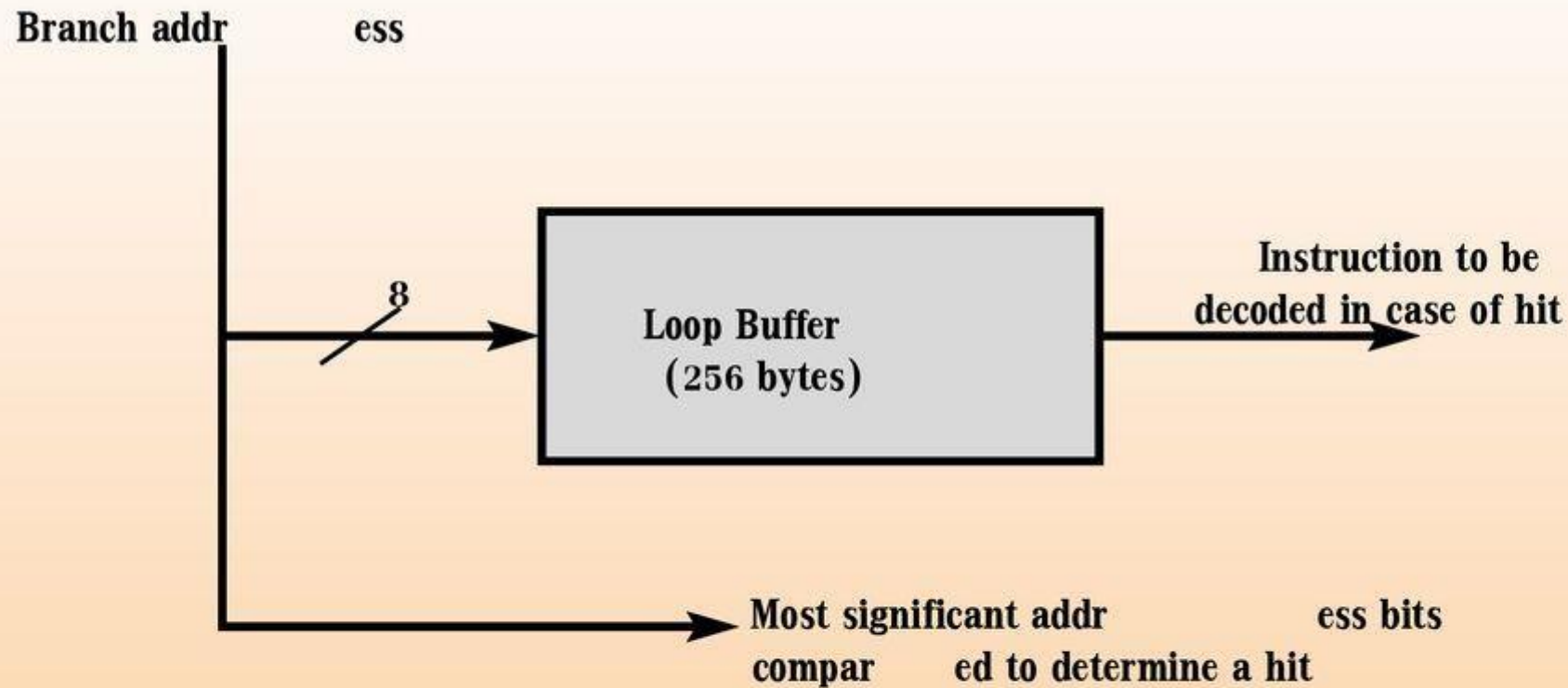


Figure 14.17 Loop Buffer

Figure 14.17 gives an example of a loop buffer. If the buffer contains 256 bytes, and byte addressing is used, then the least significant 8 bits are used to index the buffer. The remaining most significant bits are checked to determine if the branch target lies within the environment captured by the buffer.

Among the machines using a loop buffer are some of the CDC machines (Star-100, 6600, 7600) and the CRAY-1. A specialized form of loop buffer is available on the Motorola 68010, for executing a three-instruction loop involving the DBcc (decrement and branch on condition) instruction (see Problem 14.14). A three-word buffer is maintained, and the processor executes these instructions repeatedly until the loop condition is satisfied.

Branch Prediction

Various techniques can be used to predict whether a branch will be taken:

1. Predict never taken
2. Predict always taken
3. Predict by opcode



- These approaches are static
- They do not depend on the execution history up to the time of the conditional branch instruction

1. Taken/not taken switch
2. Branch history table



- These approaches are dynamic
- They depend on the execution history

Constraints

Instruction level parallelism

- Refers to the degree to which the instructions of a program can be executed in parallel
- A combination of compiler based optimization and hardware techniques can be used to maximize instruction level parallelism

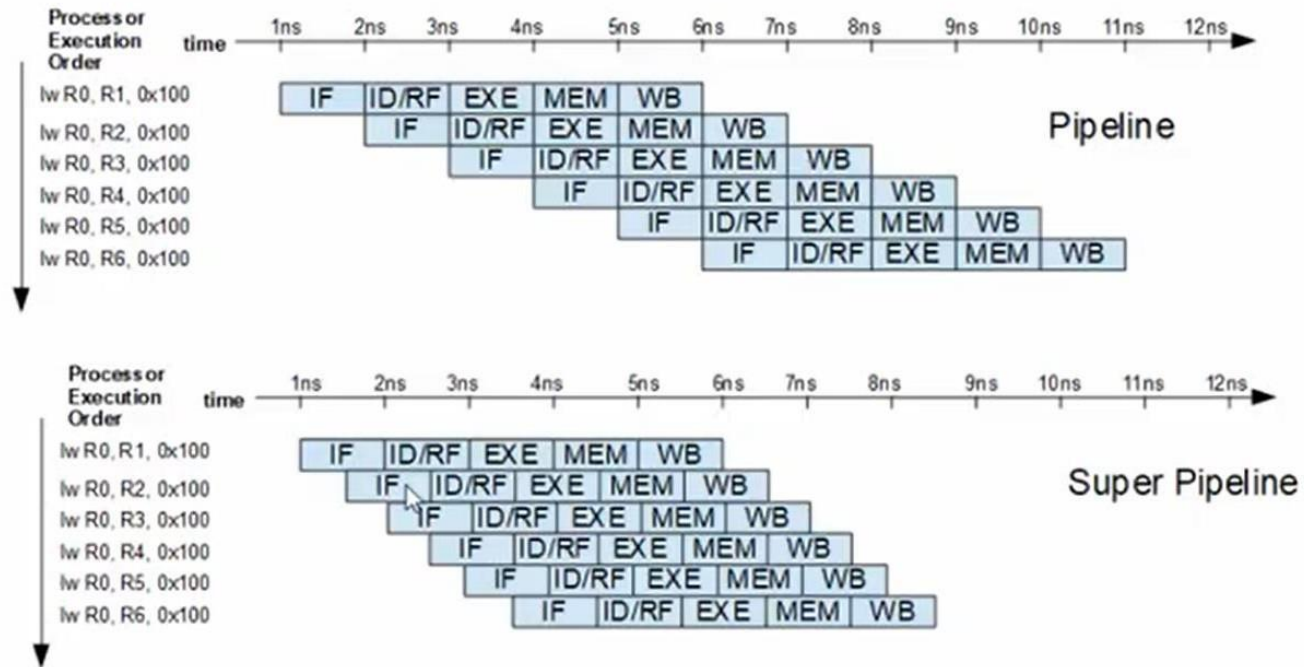
Limitations:

- True data dependency
- Procedural dependency
- Resource conflicts
- Output dependency
- Antidependency

$$ILP = IPC$$

Super pipeline

- Super pipeline implements stage overlap at half clock cycle.



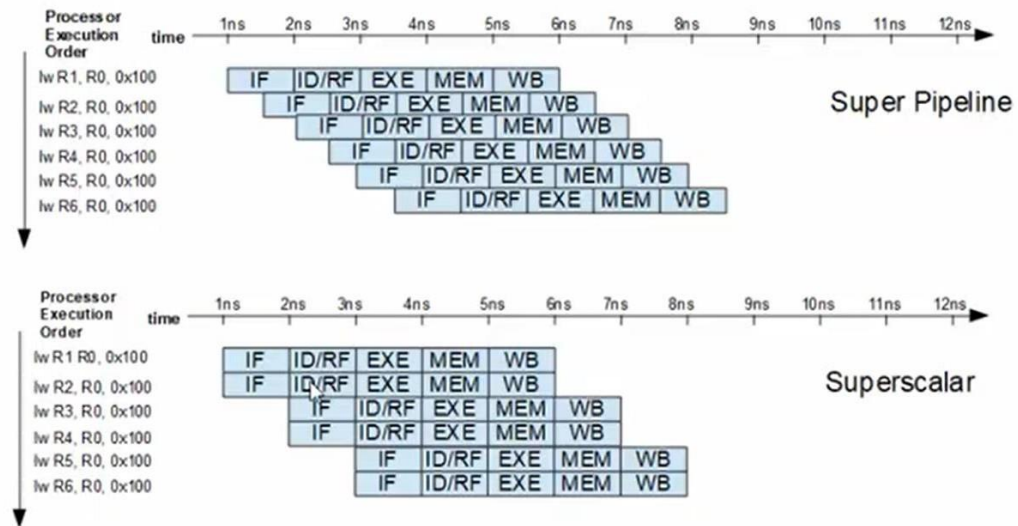
Superscalar pipeline

Superscalar pipeline introduces the ability to execute instructions independently and concurrently in different pipelines which enables more instructions executed in every clock period. A superscalar processor contains multiple copies of the datapath hardware to execute multiple instructions simultaneously.

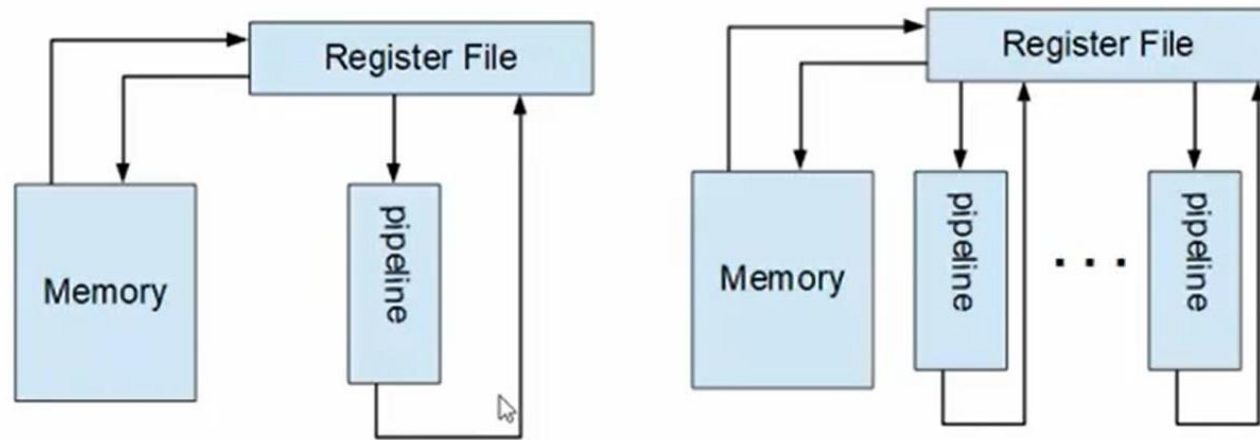
Superscalar

ILP – Superscalar

- Super pipeline implements stage overlap at half clock cycle.



Scalar vs. Superscalar



- Superscalar uses multiple pipelines in parallel.

Design Issues

Instruction-Level Parallelism and Machine Parallelism

Instruction level parallelism

- Instructions in a sequence are independent
- Execution can be overlapped
- Governed by data and procedural dependency

Machine Parallelism

- Ability to take advantage of instruction level parallelism
- Governed by number of parallel pipelines

Exercise

1. Explain what is pipelining

Pipelining is a technique where multiple instructions are overlapped during execution. Pipeline is divided into stages and these stages are connected with one another to form a pipe like structure. Instructions enter from one end and exit from another end.

2. What are the instructions involve in two-stage instruction pipeline?

The execution time will generally be longer than the fetch time. Execution will involve reading and storing operands and the performance of some operation. Thus, the fetch stage may have to wait for some time before it can empty its buffer.

3. What happen when there are more stages in pipeline?

Increase in the number of pipeline stages increases the number of instructions executed simultaneously.

Exercise

4. Define and explain every stage in six-stage instruction pipeline

Fetch instruction (FI) - Read the next expected instruction into a buffer

Decode instruction (DI) - Determine the opcode and the operand specifiers

Calculate operands (CO) - Calculate the effective address of each source operand may involve displacement, register indirect, indirect, or other forms of address calculation

Fetch operands (FO) - Fetch each operand from memory and Operands in registers need not be fetched.

Execute instruction (EI) - Perform the indicated operation and store the result, if any, in the specified destination operand location.

Write operand (WO) - Store the result in memory.

5. What is the effect of conditional branch?

A branch interrupts the normal sequence of program execution.

Exercise

6. Explain how conditional branch take place

There are two possible next instructions, and in general you don't know which one you want until some computation has taken place, either in the branch instruction itself, or some earlier instruction. (slide10)

7. Explain each of the following hazard;

Resource hazards

A resource hazard occurs when two or more instructions that are already in the pipeline need the same resource. As a result, for part of the pipeline, instructions must be executed serially, not in parallel. A resource hazard is sometimes referred to as a structural hazard.

Data hazards

Data danger occurs when access to operand locations conflicts.

Control hazards

It is also known as a branch hazard. It occurs when the pipeline makes the wrong decision on a branch prediction. It brings instructions into the pipeline that must subsequently be discarded dealing with Branches such as multiple streams, prefetch branch target, loop buffer, branch prediction, and delayed branch.

Exercise

8. List the limitation of Instruction Level Parallelism (ILP)

- Instructions in a sequence are independent
- Execution can be overlapped
- Governed by data and procedural dependency