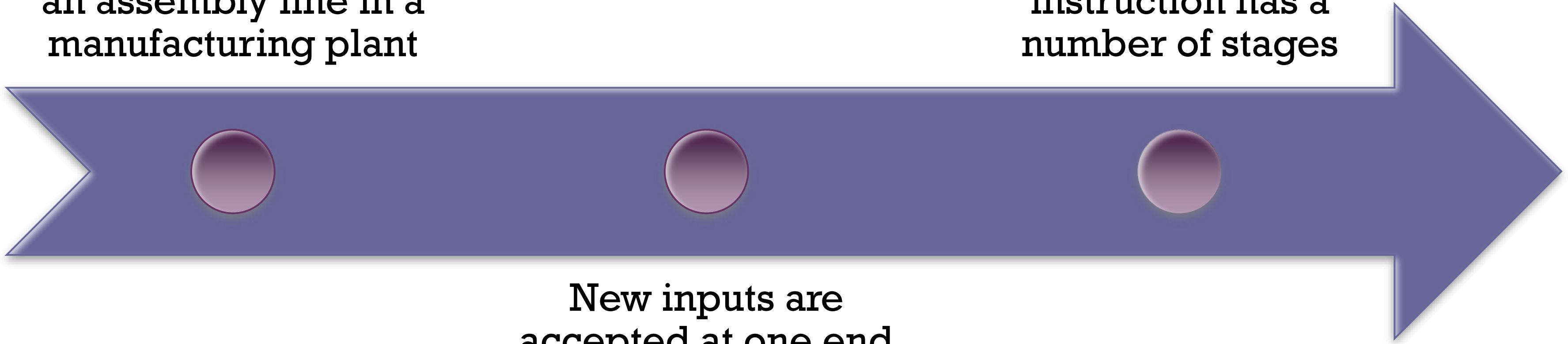
Lecture B - 03

Instruction pipelining and operation of control unit

Pipelining Strategy

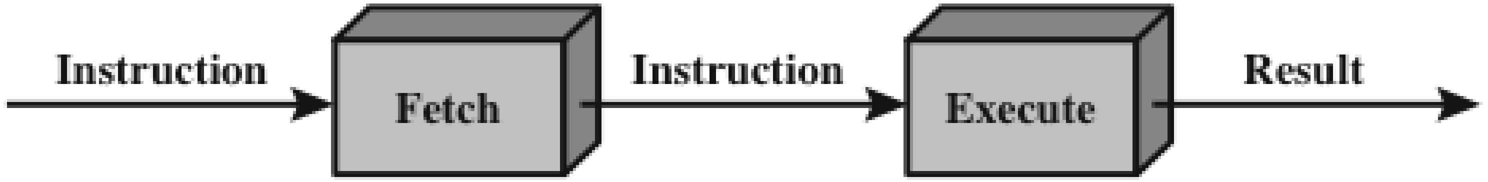


Similar to the use of an assembly line in a manufacturing plant

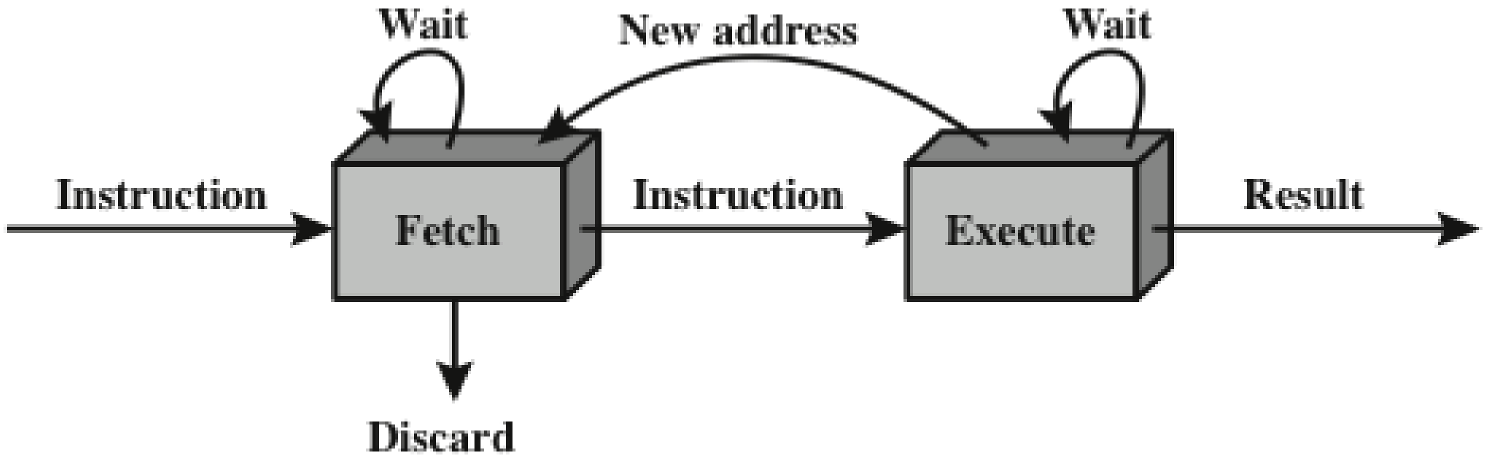
New inputs are accepted at one end before previously accepted inputs appear as outputs at the other end

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

Two-Stage Instruction Pipeline



(a) Simplified view



(b) Expanded view

Figure 14.9 Two-Stage Instruction Pipeline

Additional Stages

- 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
  - This may involve displacement, register indirect, indirect, or other forms of address calculation
- Fetch operands (FO)
  - Fetch each operand from memory
  - 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

Timing Diagram for Instruction Pipeline Operation

Time →

	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

Six-stage pipeline reduces execution time for 9 instructions
from 54 time units to 14 time units

Instruction Pipelining: Calculating Performance

- Total time units for the instruction pipeline is

$$T_{k,n} = [k + (n-1)]\tau$$

- Total time units for the instruction without pipeline is

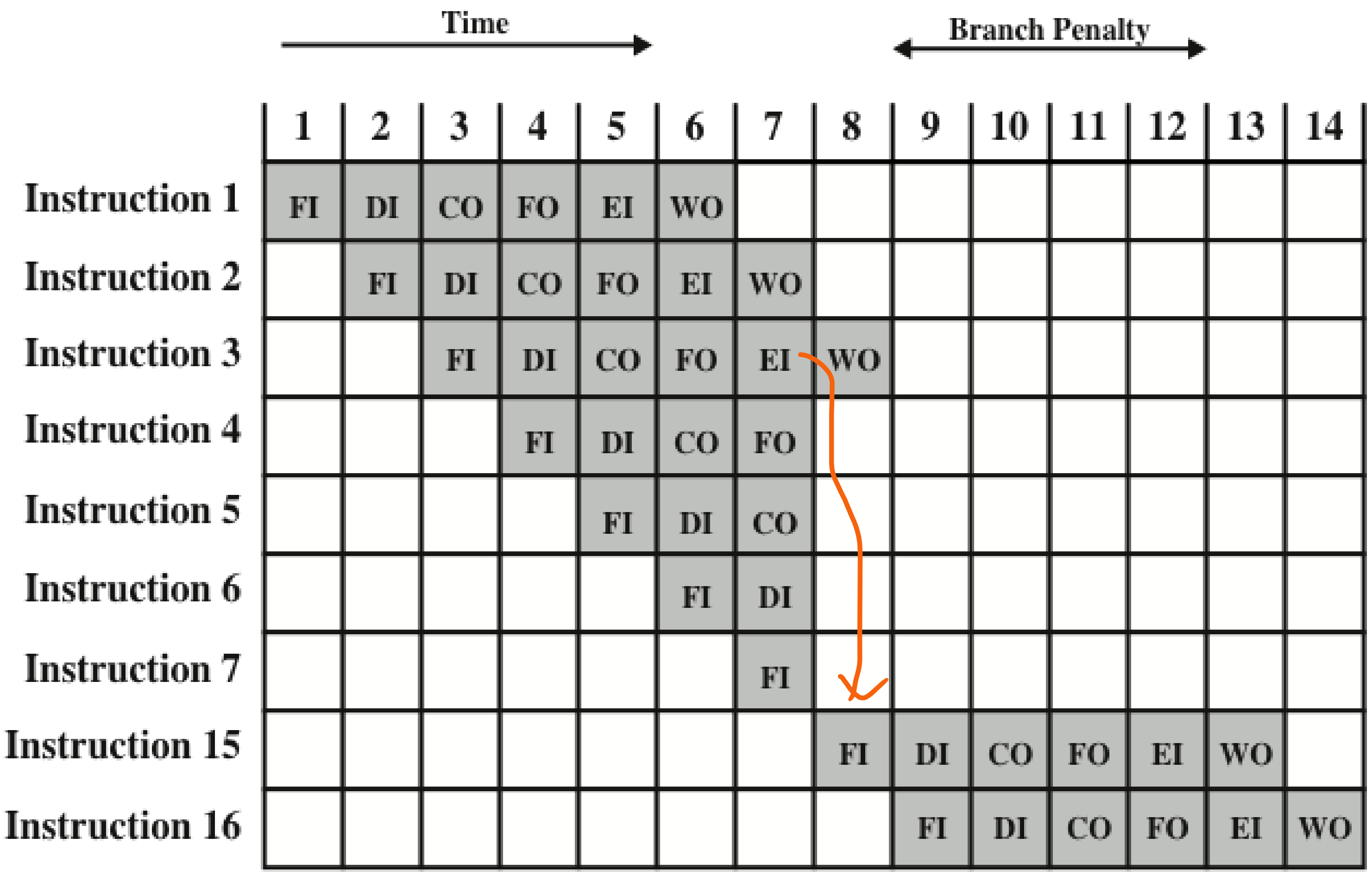
$$T_{1,n} = nk\tau$$

k = stages of the pipeline, n = number of instructions, τ = cycle time

- The speedup factor for the instruction pipeline compared to execution without the pipeline is defined as

$$S_k = \frac{T_{1,n}}{T_{k,n}} = \frac{nk\tau}{[k+(n-1)]\tau} = \frac{nk}{k+(n-1)}$$

The Effect of a Conditional Branch on Instruction Pipeline Operation



	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							
Instruction 5					FI	DI	CO							
Instruction 6						FI	DI							
Instruction 7							FI							
Instruction 15								FI	DI	CO	FO	EI	WO	
Instruction 16									FI	DI	CO	FO	EI	WO

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

Six Stage Instruction Pipeline

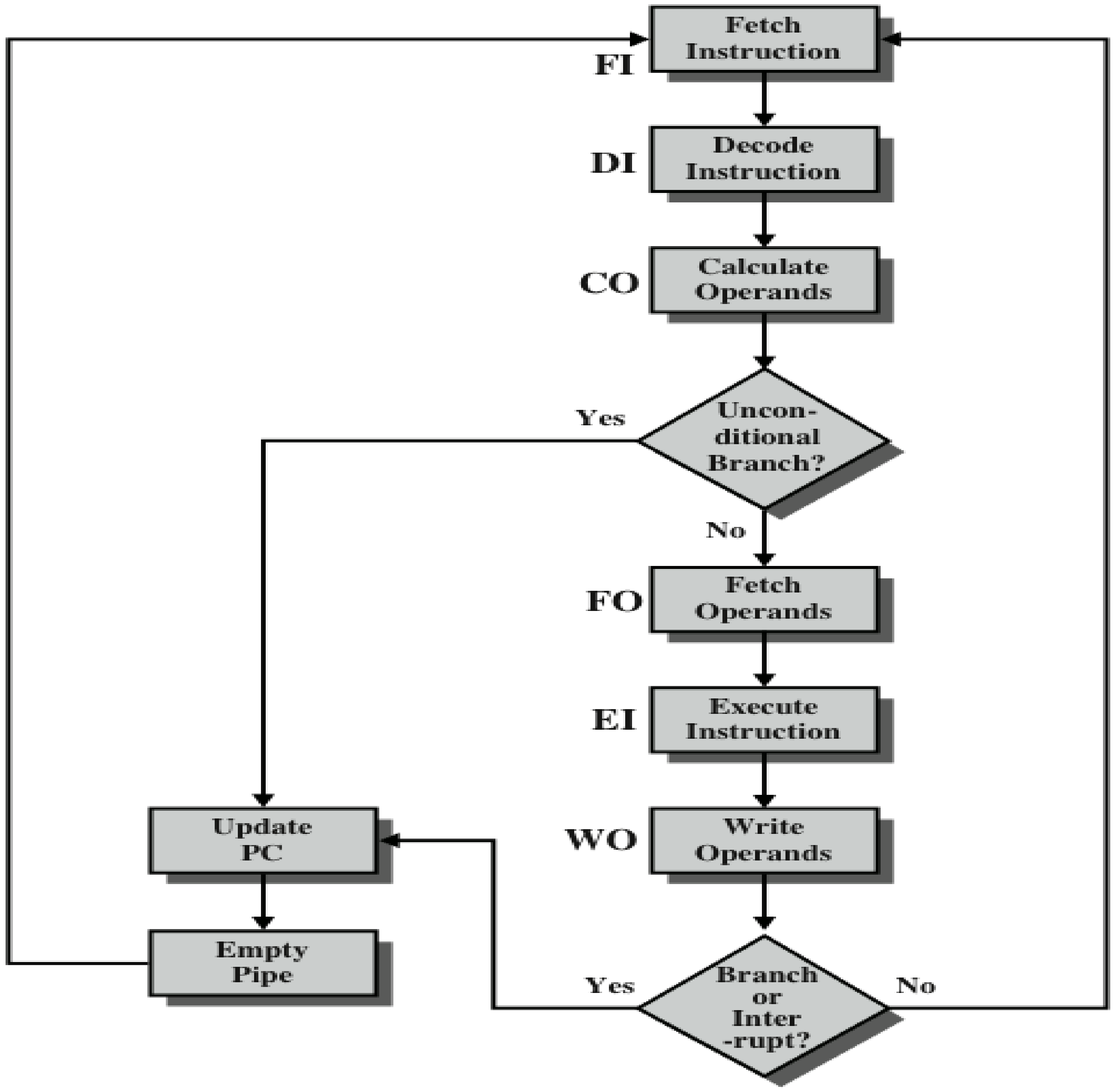
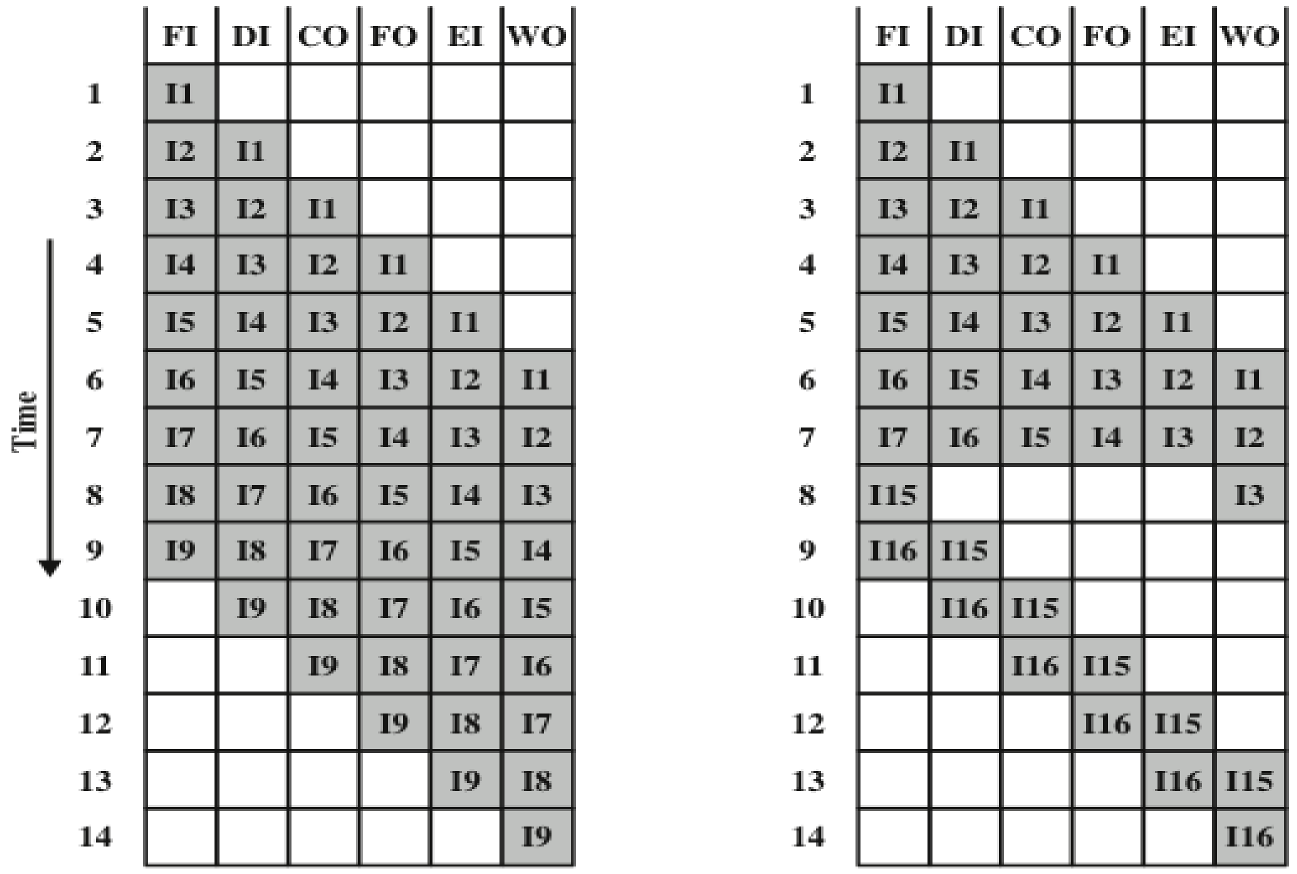


Figure 14.12 Six-Stage Instruction Pipeline

Alternative Pipeline Depiction



	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

Speedup Factors with Instruction Pipelining

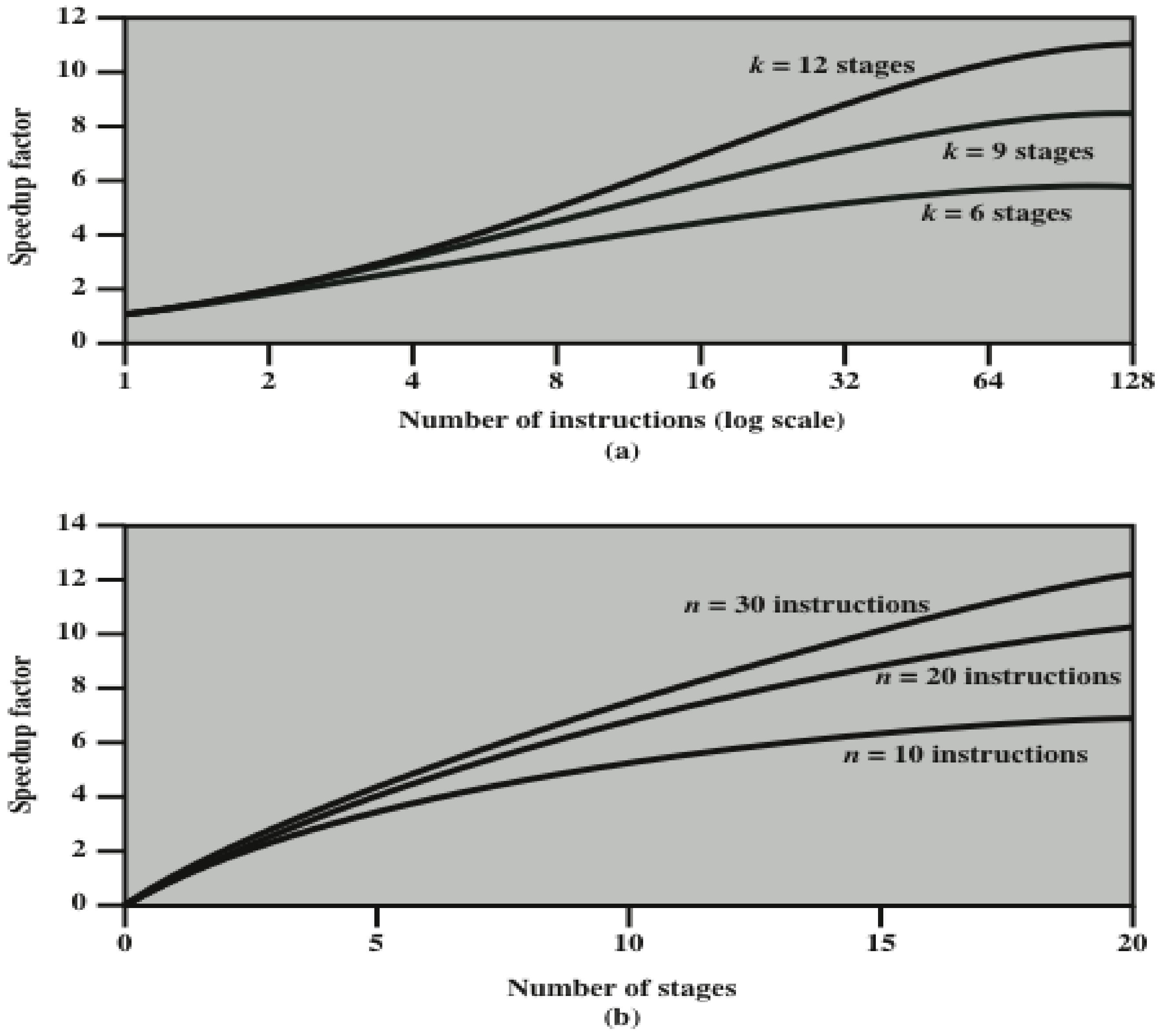


Figure 14.14 Speedup Factors with Instruction Pipelining

Pipeline Hazards

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

Also referred to as a *pipeline bubble*

There are three types of hazards:

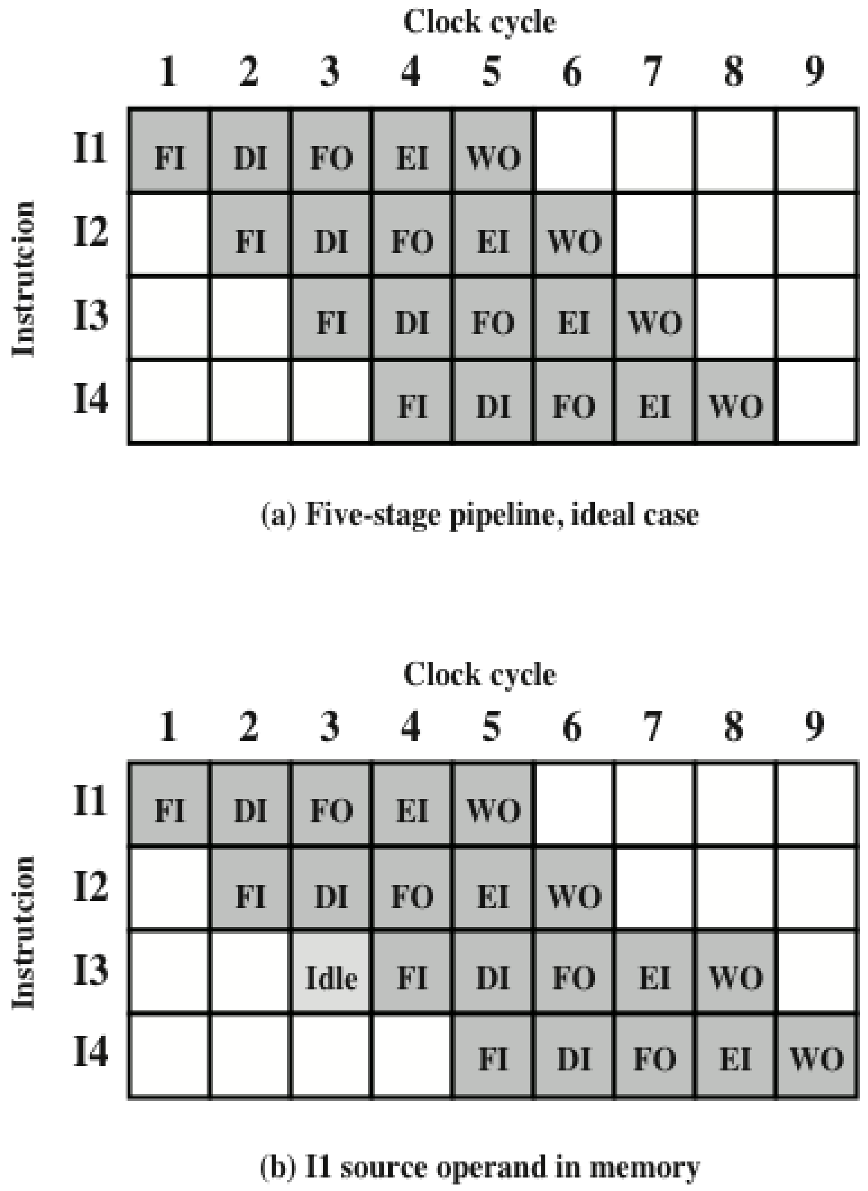
- Resource
- Data
- Control

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*



Clock cycle

Instrutcion	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	

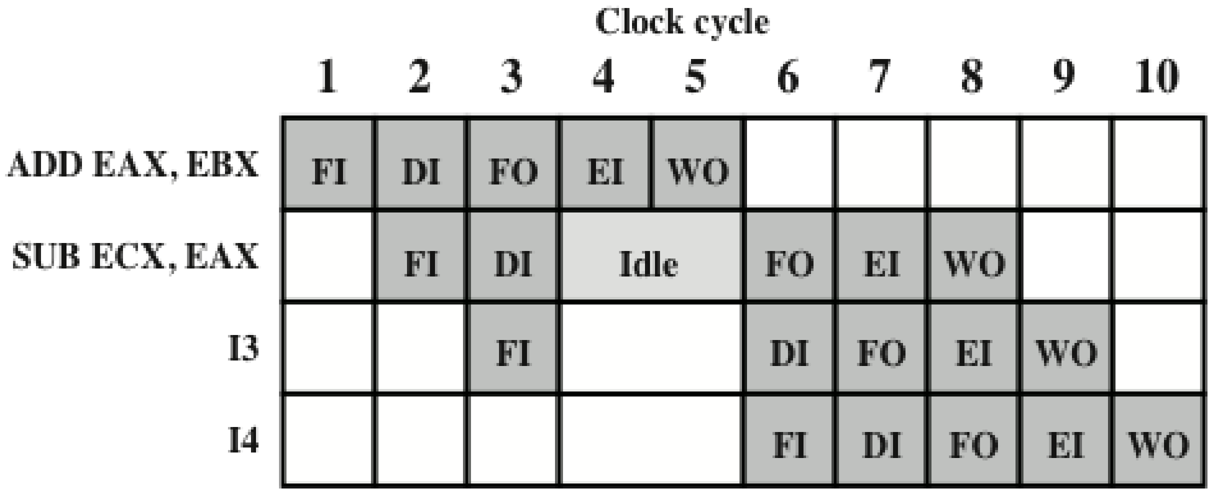
(a) Five-stage pipeline, ideal case

Clock cycle

Instrutcion	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	WO

(b) I1 source operand in memory

Figure 14.15 Example of Resource Hazard



	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

RAW

Data Hazards

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

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

- Also known as a *branch hazard*
- Occurs when the pipeline makes the wrong decision on a branch prediction
- Brings instructions into the pipeline that must subsequently be discarded
- Dealing with Branches:
  - Multiple streams
  - Prefetch branch target
  - Loop buffer
  - Branch prediction
  - Delayed branch

Multiple Streams

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

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

Prefetch Branch Target

- 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
- If the branch is taken, the target has already been prefetched

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

Branch Prediction

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

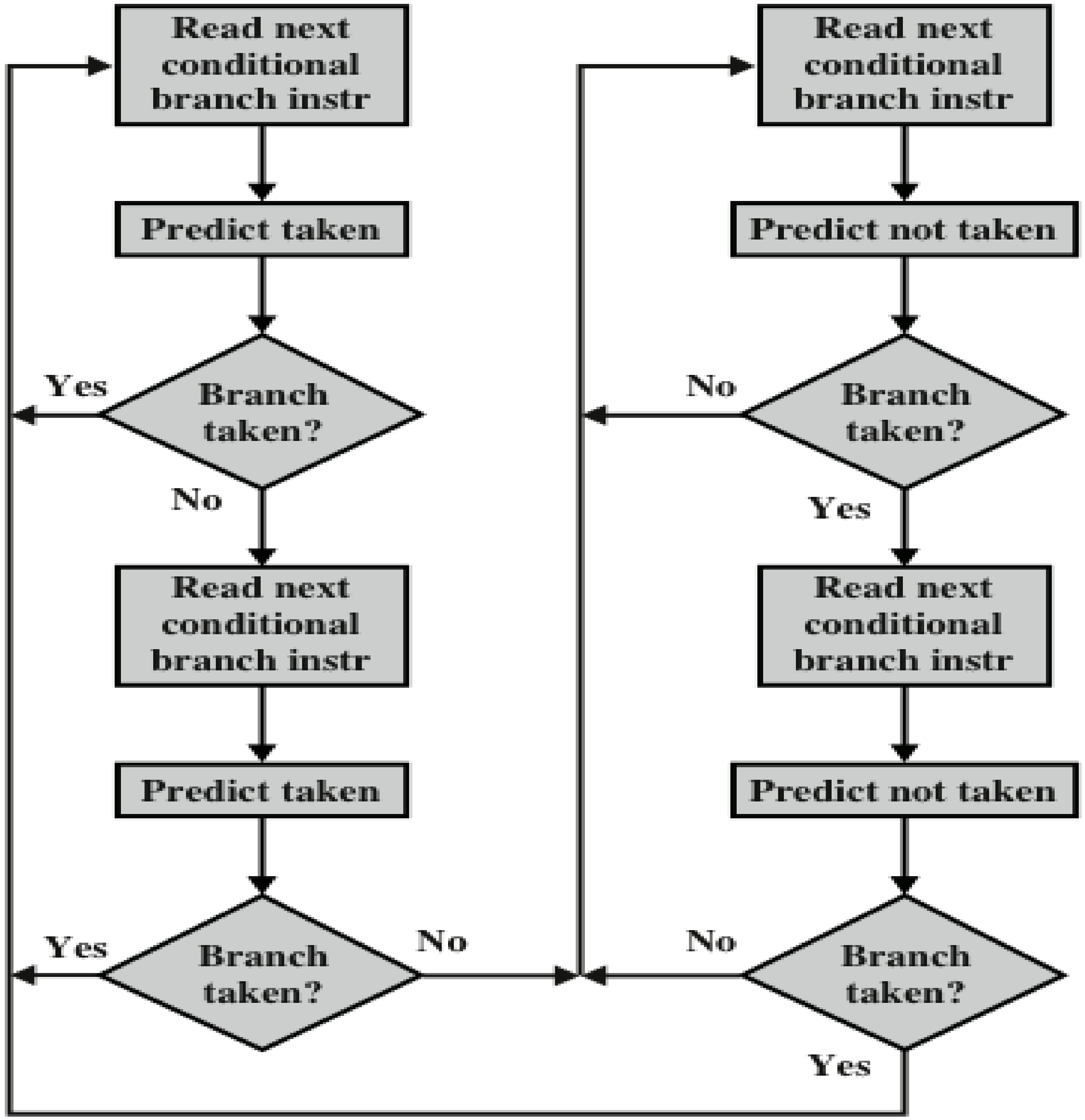
1. Predict never taken
2. Predict always taken
3. Predict by opcode

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

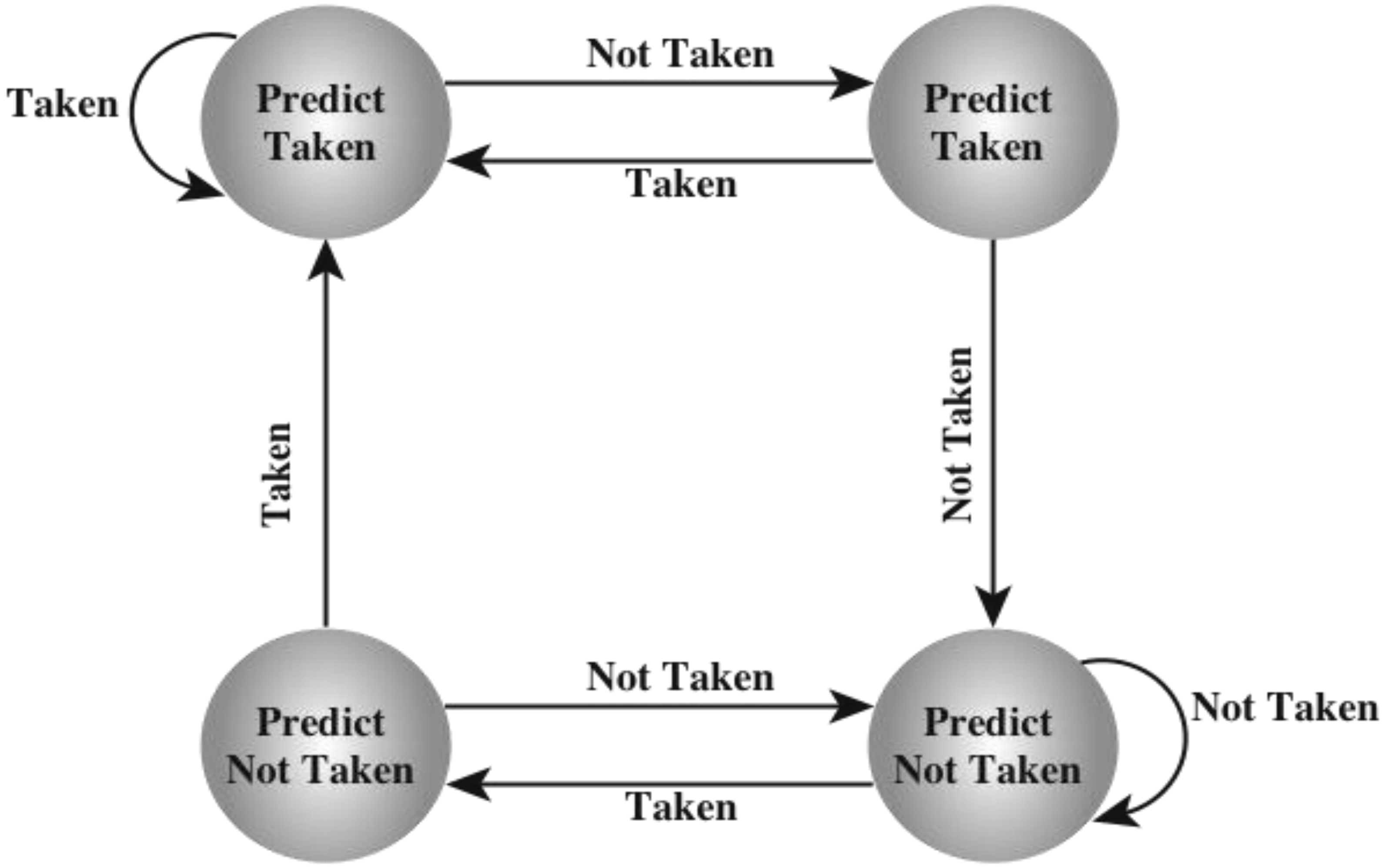
1. Taken/not taken switch
2. Branch history table

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

Branch Prediction Flow Chart


Read next conditional branch instr
Predict taken
Branch taken?
Yes
No
Read next conditional branch instr
Predict taken
Branch taken?
Yes
No
Read next conditional branch instr
Predict not taken
Branch taken?
No
Yes
Read next conditional branch instr
Predict not taken
Branch taken?
No
Yes

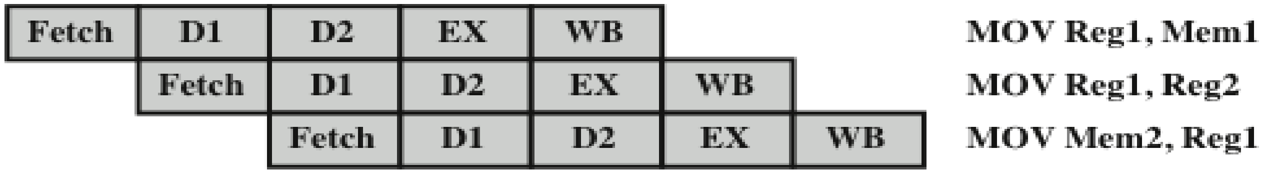
Branch Prediction State Diagram



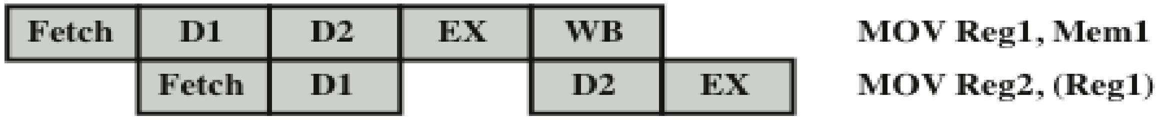
Example - Intel 80486 Pipelining

- Fetch
  - Objective is to fill the prefetch buffers with new data as soon as the old data have been consumed by the instruction decoder
  - Operates independently of the other stages to keep the prefetch buffers full
- Decode stage 1
  - All opcode and addressing-mode information is decoded in the D1 stage
  - 3 bytes of instruction are passed to the D1 stage from the prefetch buffers
  - D1 decoder can then direct the D2 stage to capture the rest of the instruction
- Decode stage 2
  - Expands each opcode into control signals for the ALU
  - Also controls the computation of the more complex addressing modes
- Execute
  - Stage includes ALU operations, cache access, and register update
- Write back
  - Updates registers and status flags modified during the preceding execute stage

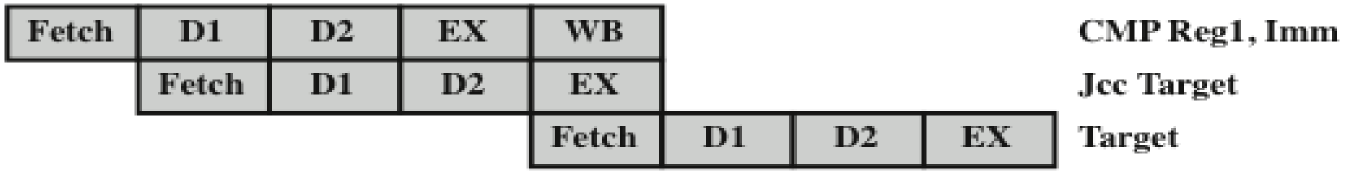
80486 Instruction Pipeline Examples



(a) No Data Load Delay in the Pipeline



(b) Pointer Load Delay

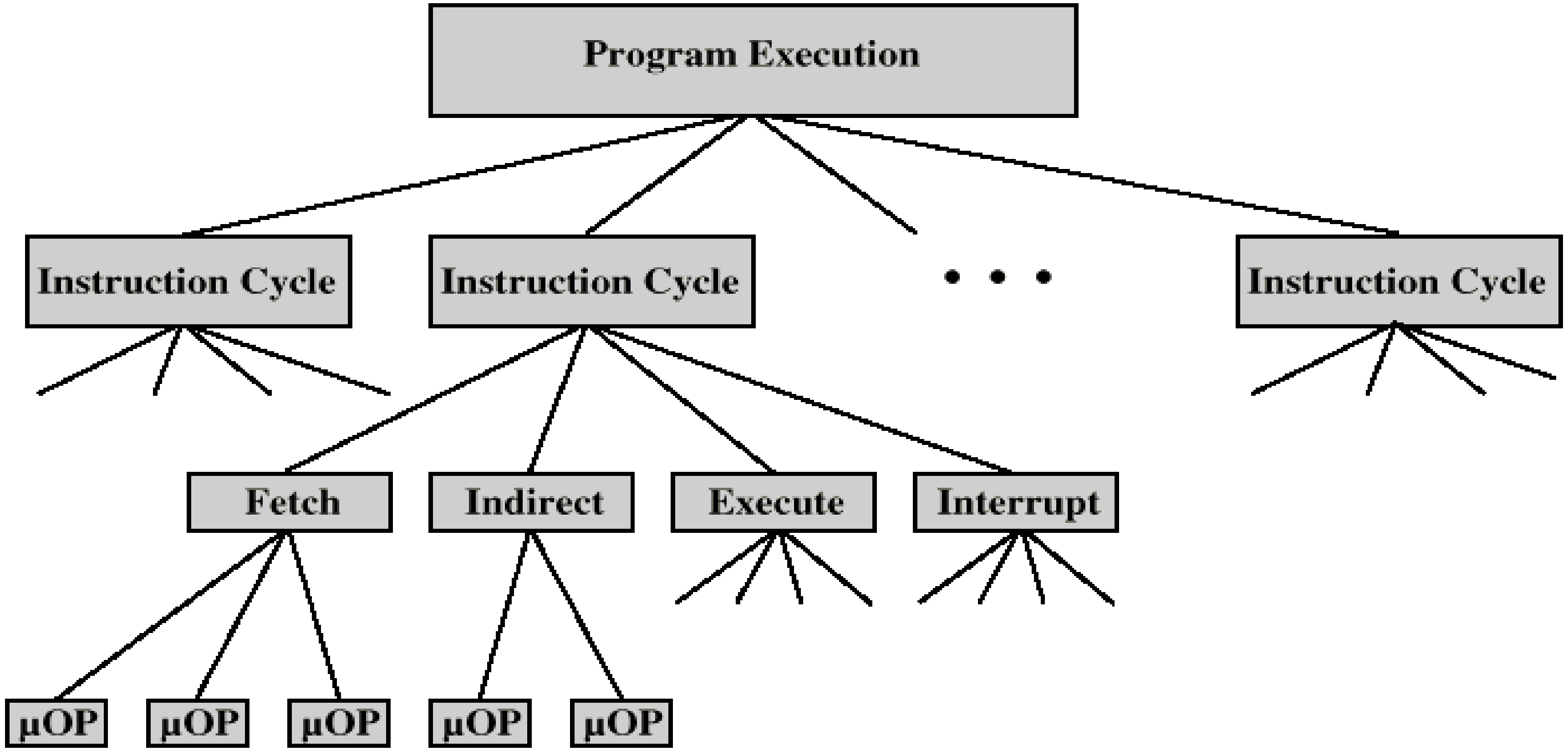


(c) Branch Instruction Timing

Basic Elements of Processor

- Execution of a program consists of operations involving the following processor elements
  - ALU
    - Essential functional unit of the CPU
  - Registers
    - Store data internal to the CPU
    - e.g. status information to manage instruction sequencing (program status word); data that go to or come from the ALU, memory and I/O modules
  - Internal data paths
    - Used to move data between registers, and between registers and ALU
  - External data paths
    - Link registers to memory and I/O modules (system bus)
  - Control Unit
    - Causes operation to happen within the CPU

Constituent Elements of Program Execution



Control Unit Tasks

- **Two tasks performed by the control unit of a processor:**
  - Causes the processor to **execute micro-ops in the proper sequence** determined by the program being executed
  - **Generates the control signals** that cause each micro-op to be executed.

- **The control signals generated by the Control Unit**
  - Cause the opening and closing of logic gates
  - Results in the transfer of data to and from registers and the operation of the ALU.

Techniques for Implementing Control Unit

- **Hard-wired Implementation:**
  - The Control Unit is a combinatorial circuit.
  - Its input logic signals, governed by the current machine instruction, are transferred into a set of output control signals
- **Microprogrammed Implementation/Control (MI):**
  - The logic of the Control Unit is specified by a **microprogram**
  - Each machine language instruction is translated into a sequence of lower-level control unit instructions
    - Called **microinstructions**
    - Process of translation is called microprogramming
  - A microprogram consists of a sequence of instructions in a microprogramming language

Functional Requirements of the Control Unit

- Behavior or functioning of the CPU was decomposed into elementary operations called micro-operations.
- Define the functional requirements of the Control Unit (those functions that the Control Unit must perform)
  - Basis for the design and implementation of the control unit
- Three-step process that leads to a characterization of the CU:
  - Define the basic elements of the processor
  - Describe micro-operations that the processor performs
  - Determine the functions that the CU must perform to cause a micro-op to be performed

Control Unit Operation

- How is the decomposing of the functioning of the processor into elementary operations (micro operations) done?
- How does the control unit cause the sequence to occur?
  - Model for the control unit
  - Two basic functions (Sequencing/Execution)
  - Data paths and Control Signals
  - Hardwired Implementation – Control Logic
- CPU with internal bus – organization

Micro-Operations

- **Program Execution**
  - Consists of the sequential execution of instructions
  - subdivided into various instruction cycles
- **Instruction Cycle**
  - divided into subtasks
  - Fetch, indirect, execute and interrupt with fetch and execute always occurring
- **Micro-operations**
  - Each cycle made up of a sequence of more fundamental operations
  - called micro-operations.
  - The prefix micro means that each step is very simple and accomplishes very little.
  - Functional, or atomic operations of a processor
  - Micro-operations may involve
    - transfer between registers
    - between registers and external bus
    - a simple ALU operation

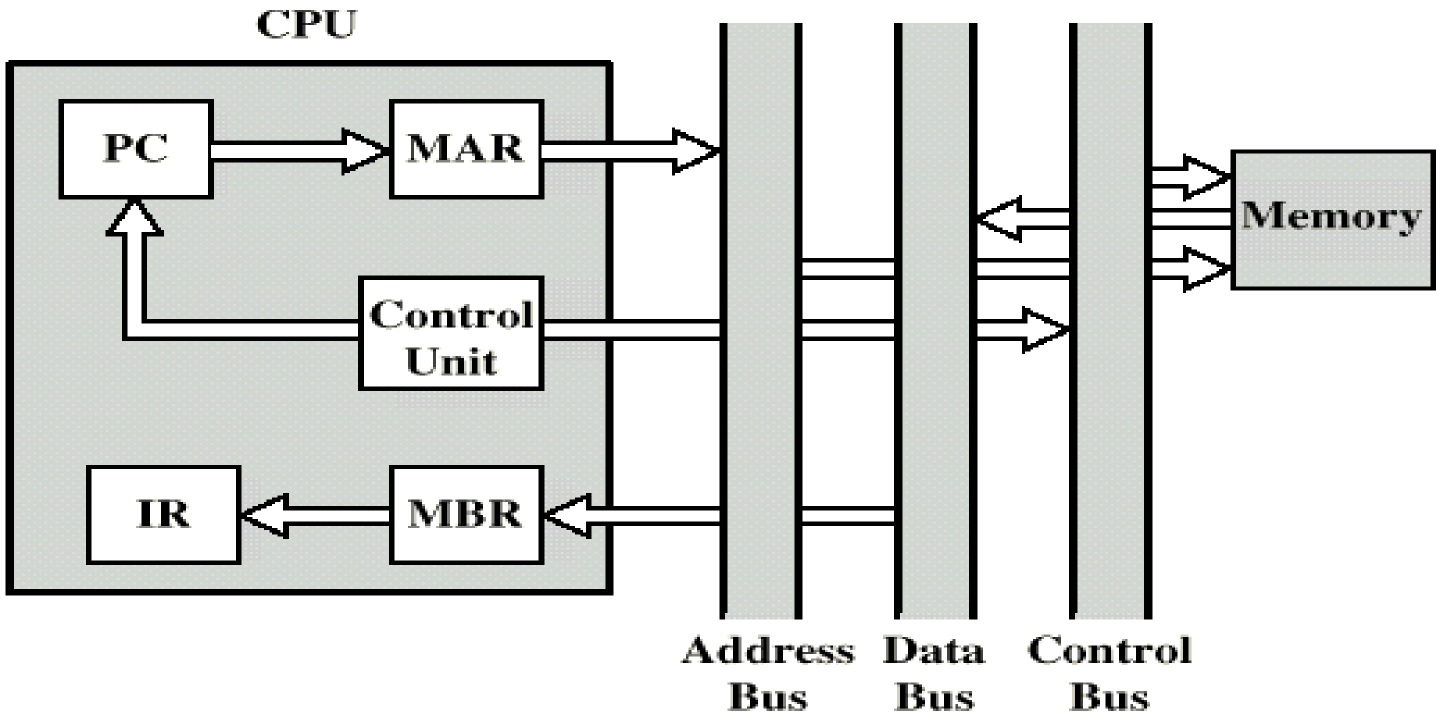
Types of Micro-operation Categories

- Operations involving processor elements consists of a sequence of micro-operations
- Categories of micro-operations
  - Transfer data between registers
  - Transfer data from register to an external interface
    - e.g. system bus
  - Transfer data from an external interface to a register
  - Perform arithmetic or logical operations, using registers for input and output.

Micro-Operations – Fetch Cycle (1)

- Fetch cycle occurs at the beginning of each instruction.
- In this operation four registers are involved.
  - **Memory Address Register (MAR)**
    - Connected to address lines of the system bus
    - Specifies address for read or write operation
  - **Memory Buffer Register (MBR)**
    - Connected to data lines of the system bus
    - Holds data to stored in memory or the last data read from memory
  - **Program Counter (PC)**
    - Holds address of next instruction to be fetched
  - **Instruction Register (IR)**
    - Holds last instruction fetched

Micro-Operations – Fetch Cycle (2) Data Flow



MBR = Memory buffer register
MAR = Memory address register
IR = Instruction register
PC = Program counter

Micro-Operations – Fetch Cycle (3) Sequence

- **STEP 1:Identification of the memory location**
  - The address of next instruction to be executed is in PC
  - Move that address to the MAR (this is the only register connected to the address lines of the system bus)
- **STEP 2: Bringing the instruction from memory**
  - Address (MAR) is placed on address bus
  - Control unit issues READ command on the control bus
  - Result (data from memory) appears on data bus
  - Data from data bus copied into MBR
  - PC incremented by I (in parallel with data fetch from memory) – I is the instruction length
- **STEP 3: Move the contents of the MBR to IR**
  - MBR is now free for further data fetches

Micro-Operations – Fetch Cycle (4)
Sequence of events

MAR		
MBR		
PC	0000000001100100	**0064H**
IR		
AC		

(a) Beginning

MAR	0000000001100100	**0064H**
MBR		
PC	0000000001100100	0064H
IR		
AC		

(b) First Step

MAR	0000000001100100	0064H
MBR	0001000000100000	**1020H**
PC	0000000001100101	**0065H**
IR		
AC		

(c) Second Step

MAR	0000000001100100	0064H
MBR	0001000000100000	1020H
PC	0000000001100101	0065H
IR	0001000000100000	**1020H**
AC		

(d) Third step

Micro-Operations – Fetch Cycle (5) Symbolic Sequence of Events

■ t_1:	MAR <- (PC)	(t_1 - first time unit)
■ t_2:	MBR <- Memory	(t_2-second time unit)
	PC <- (PC) +I	(I-Instruction Length)
■ t_3:	IR <- (MBR)	(t_3-third time unit)

t_1, t_2, t_3 are successive time units.

OR

- t_1 : MAR <- (PC)
- t_2 : MBR <- Memory
- t_3 : PC <- (PC) +I
  IR <- (MBR)

Micro-Operations – Fetch Cycle (6) - Rules to follow

- **Two Rules for groupings of micro-operations:**
  - Proper sequence of events must be followed
    - (MAR <- (PC)) must precede (MBR <- Memory)
      - because memory read operation makes use of the address in MAR
  - Conflicts must be avoided
    - Must not read & write with same register at same time-results unpredictable
    - (MBR <- Memory) & (IR <- (MBR)) must not be in same time unit

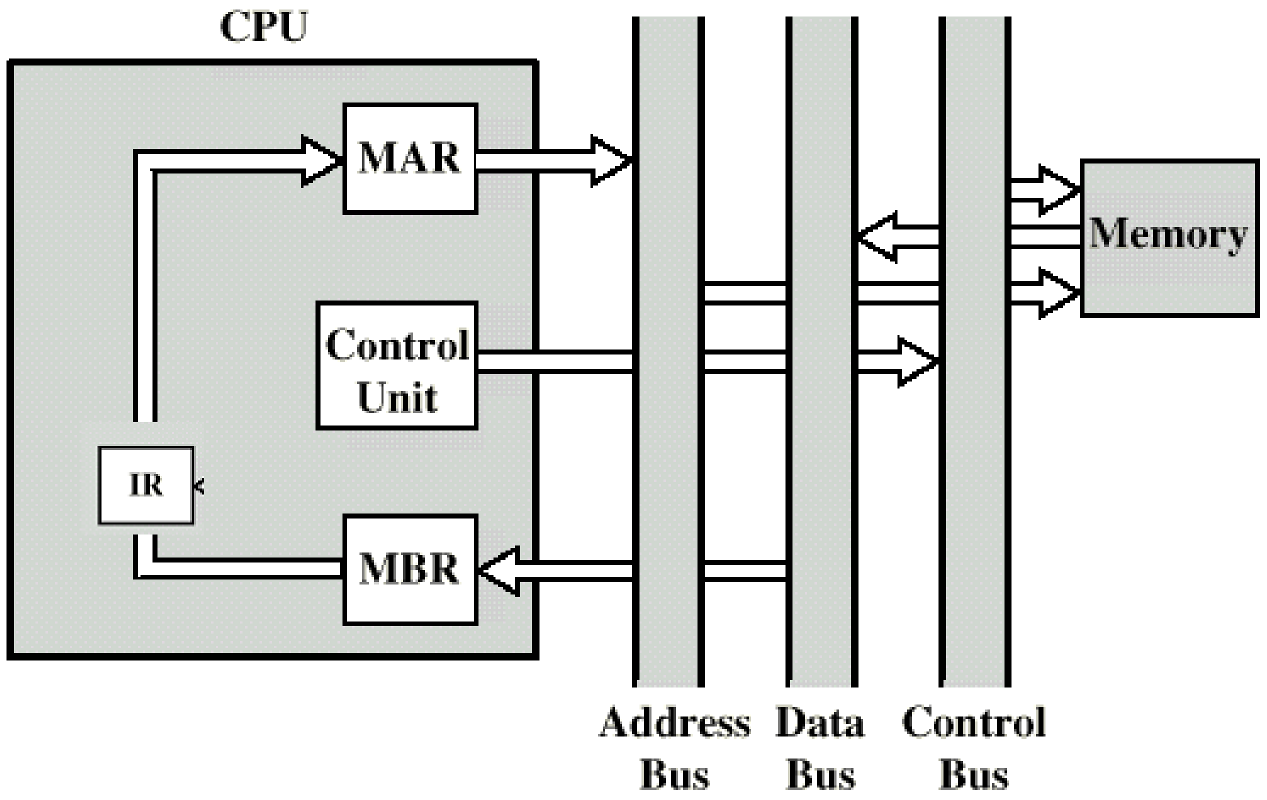
Note:

- PC <- (PC) +I involves addition
  - Use ALU
  - May need additional micro-operations

Micro-Operations – Indirect Cycle (1)

- After Instruction Fetch - Indirect Cycle before execute cycle if the instruction specifies an indirect address
- Assume one-address instruction format
- Micro-operations for indirect cycle
  - t_1: MAR <- ($IR_{address}$) (move address part of IR contents to MAR)
  - t_2: MBR <- memory
  - t_3: $IR_{address}$ <- ($MBR_{address}$) (MAR now is used to fetch the address of the operand address field of IR is updated from MBR. Now IR contains the direct address)
- MBR contains an address
- IR is now in same state as if direct addressing had been used

Micro-Operations - Indirect Cycle (2)



+

6. Assume that Word 10 contains 20, Word 20 contains 30, Word 30 contains 40, and Word 40 contains 50. Given the above memory values and a one-address machine with an accumulator, what values do the following instructions load into the accumulator?

- i) LOAD IMMEDIATE 30
- ii) LOAD DIRECT 30
- iii) LOAD INDIRECT 30
- iv) LOAD IMMEDIATE 10
- v) LOAD DIRECT 40
- vi) LOAD INDIRECT 10

Micro-Operations - Interrupt cycle (1)

- At the completion of execute cycle, if enabled interrupts have occurred, microprocessor will enter interrupt cycle
- Nature of cycle varies from one machine to another
- Simple sequence of events

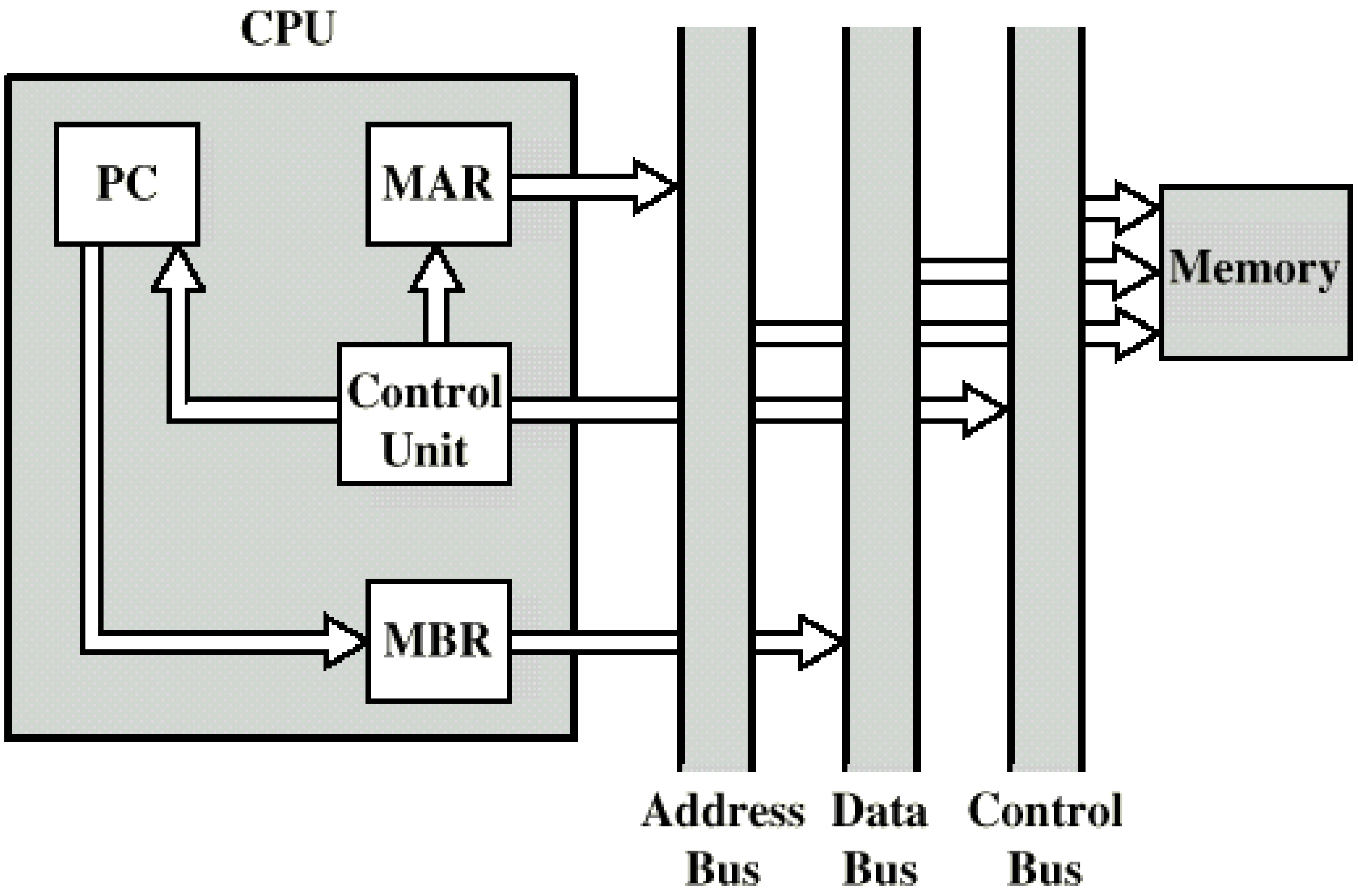
  t_1: MBR $\leftarrow$ (PC) (contents of PC transferred to MBR for saving to enable return from interrupt)

  t_2: MAR $\leftarrow$ Save_address (MAR with stack address)

  PC $\leftarrow$ Routine_address (PC with ISR address)

  t_3: Memory $\leftarrow$ (MBR) (store the old value of PC into stack)

Micro-Operations - Interrupt Cycle (2)



Micro-Operations - Execute Cycle (1)

The fetch, indirect, and interrupt cycles are simple and predictable. Each involves a small, fixed sequence of micro-ops that are repeated each time

- The execute cycle is different for each Opcode
  - machine with n different opcodes - n different sequences of micro-operations
- **Example 1:**

  ADD R1, X - add the contents of location X to Register R1
- Suppose IR contains the ADD instruction
  - t_1: MAR $\leftarrow$ $(IR_{address})$ (The address portion of IR is loaded into MAR)
  - t_2: MBR $\leftarrow$ memory (The referenced memory location is read)
  - t_3: R1 $\leftarrow$ R1 + (MBR) (The content of R1 and MBR are added by the ALU)
- Additional micro-operations may be required
  - To extract register reference from IR
  - Store ALU I/O in intermediate registers

Micro-Operations - Execute Cycle (2)

- **Example 2:**

  ISZ X (Increment and Skip if Zero)
  (The contents of location X is incremented by 1. If the result is 0, the next instruction is skipped)

  t_1: MAR $\leftarrow$ $(IR_{address})$

  t_2: MBR $\leftarrow$ Memory

  t_3: MBR $\leftarrow$ (MBR) +1

  t_4: Memory $\leftarrow$ (MBR)
  If ((MBR)=0) then (PC $\leftarrow$ (PC) + 1)

- Test and action - implemented as one micro-operation. Updated value storage and test and action can be performed during the same time unit

Micro-Operations - Execute Cycle (3)

- **Example 3:**

BSA X (Branch and Save address)
(Subroutine Call Instruction - Address of instruction that follows the BSA instruction saved in location X and execution starts from location X+I)

t_1: MAR $\leftarrow$ $(IR_{address})$

MBR $\leftarrow$ (PC)

t_2: PC $\leftarrow$ $(IR_{address})$

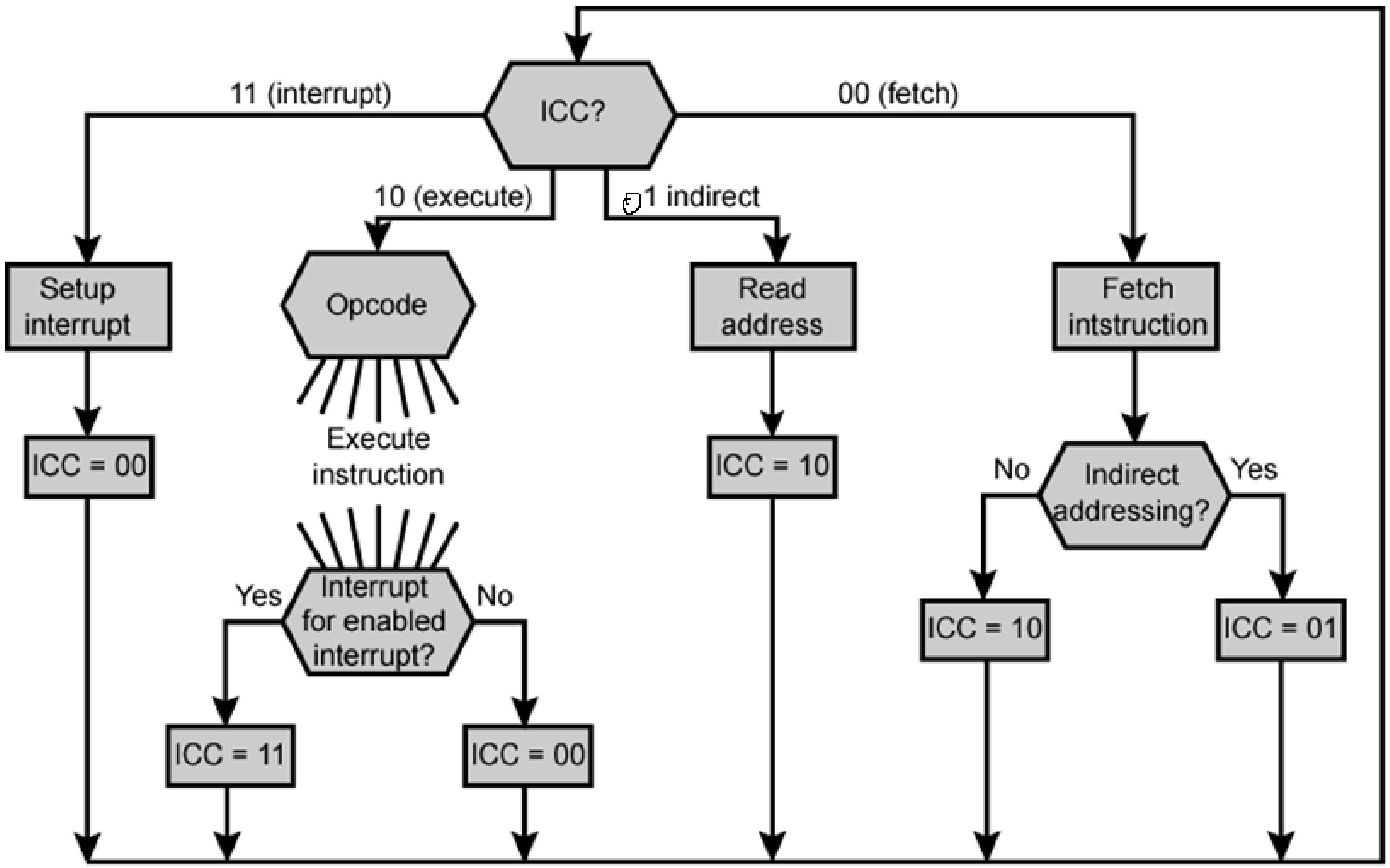
Memory $\leftarrow$ (MBR)

t_3: PC $\leftarrow$ (PC)+I

Flowchart for Instruction Cycle (1) – Complete Sequence of micro-operations

- Assume a new 2-bit reg. called instruction cycle code (ICC)
- Instruction cycle code (ICC) designates which part of cycle processor is in
  - **ICC : 00=Fetch, 01=Indirect, 10= Execute,11=Interrupt**
- Indirect cycle - always followed by execute cycle
- Interrupt cycle- always followed by fetch cycle
- Fetch cycle- depends upon the state - indirect addressing used or not
- Execute cycle - depends upon whether the enabled interrupt has occurred or not
- Summary: Operation of the processor is defined as the performance of a sequence of micro-operations

Flowchart for Instruction Cycle (2) – Complete Sequence of micro-operations



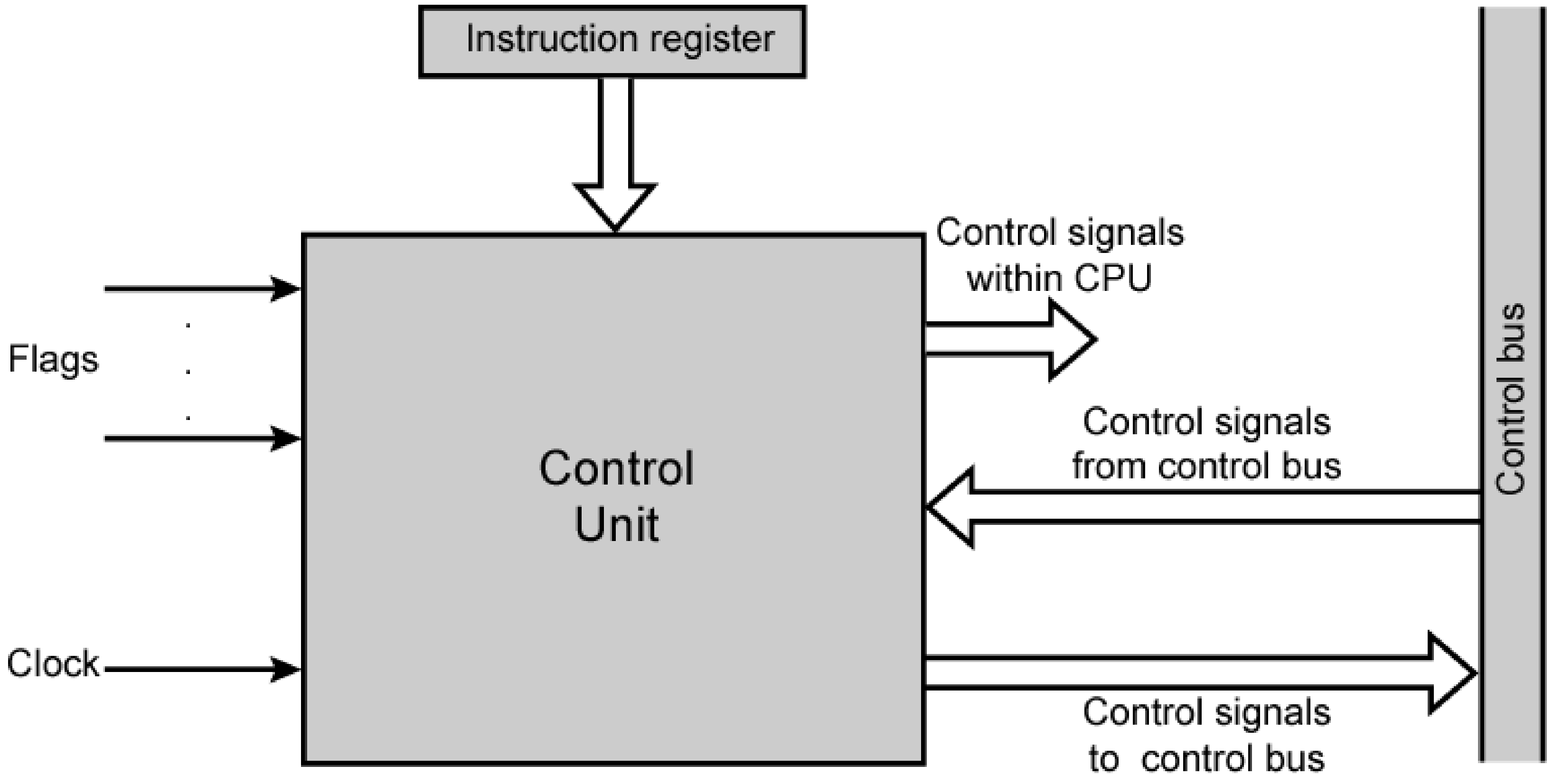
Functions of Control Unit (CU)

The control unit performs two basic tasks:

- Sequencing
  - Based on the program being executed, the CPU causes the proper sequencing of micro-operations
- Execution
  - The Control Unit causes each micro-operation to be performed
- This is done by using Control Signals

Model of Control Unit (1)

http://people.cs.pitt.edu/~childers/CS0447/lectures/datapath3.pdf


Instruction register
Control signals within CPU
Flags
Control Unit
Control signals from control bus
Control bus
Clock
Control signals to control bus

https://www.elprocus.com/what-is-control-unit-components-its-design/

Model of Control Unit (2)
Control Signals (Input)

- Clock
  - One micro-instruction (or set of parallel micro-instructions) per clock cycle
- Instruction register
  - Op-code and addressing mode of current instruction
  - Determines which micro-operations to perform
- Flags
  - To determine the status of CPU
  - Results of previous operations
  - Example: ISZ (Increment-and-skip-if-zero) instruction
    - Control unit will increment the PC if the zero flag is set
- Control Signals from control bus
  - Interrupts
  - Acknowledgements

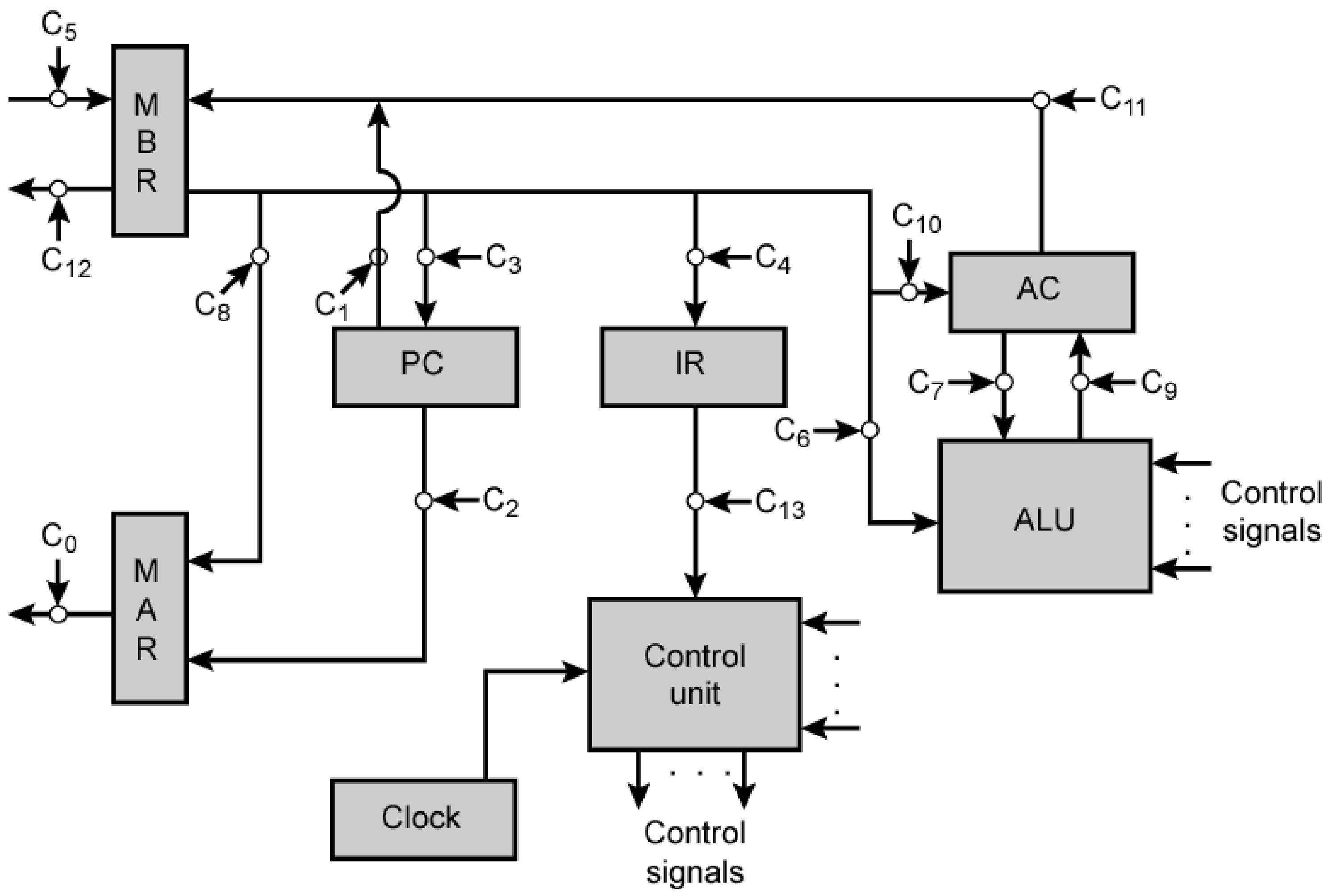
Model of Control Unit (3)
Control Signals - Output

- Control Signals within the CPU
  - Signals that cause data movement from one register to another
  - Signals that activate specific ALU functions
- Control Signals to Control Bus
  - Signals to memory
  - Signals to I/O modules
- These control signals are applied directly as binary inputs to individual logic gates

Example Control Signal Sequence Fetch

- MAR <- (PC) - transfer the contents of PC into MAR
  - Control unit activates control signal to open gates between the bits of PC and the bits of MAR
- MBR <- (memory) - read a word from memory into MBR, increment the PC
  - A control signal that opens gates allowing the contents of MAR onto the address bus
  - A memory read control signal on the control bus
  - A control signal that opens gates, allowing the contents of the data bus to be stored in the MBR
  - Control Signals to logic that add 1 to the contents of the PC and store the result back to PC
- IR <- (MBR)
  - A control signal that opens gates between MBR and IR

Data Paths and Control Signals (1)
An example



Data Paths and Control Signals (3)
An example

Micro-Operations and Control Signals

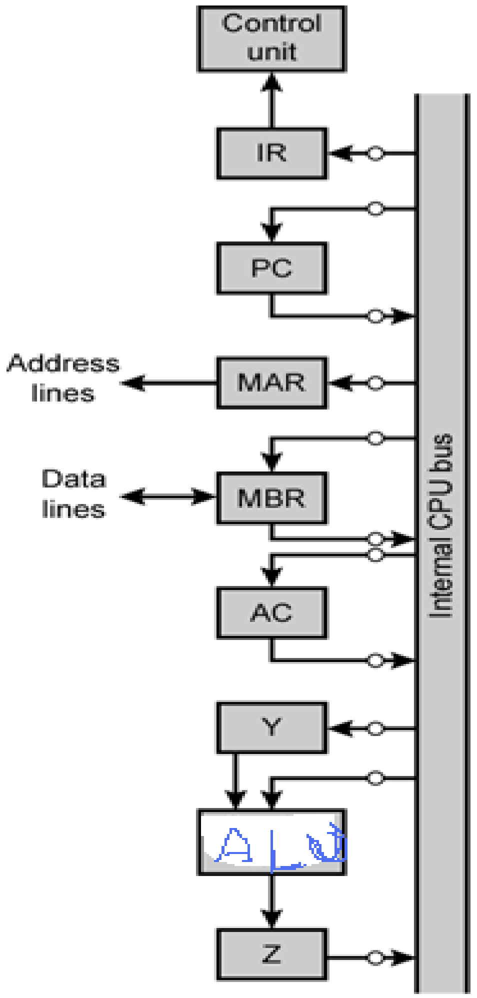
	Micro-operations	Active Control Signals
Fetch:	t_1: MAR $\leftarrow$ (PC)	C_2
	t_2: MBR $\leftarrow$ Memory PC $\leftarrow$ (PC) + 1	C_5, C_R
	t_3: IR $\leftarrow$ (MBR)	C_4
Indirect:	t_1: MAR $\leftarrow$ (IR(Address))	C_8
	t_2: MBR $\leftarrow$ Memory	C_5, C_R
	t_3: IR(Address) $\leftarrow$ (MBR(Address))	C_4
Interrupt:	t_1: MBR $\leftarrow$ (PC)	C_1
	t_2: MAR $\leftarrow$ Save-address PC $\leftarrow$ Routine-address	
	t_3: Memory $\leftarrow$ (MBR)	C_{12}, C_W

C_R	=	Read control signal to system bus.
C_W	=	Write control signal to system bus.

CPU with Internal Bus (1)

- Usually a single internal bus
- Gates control movement of data onto and off the bus
- Control signals control data transfer to and from external systems bus
- Temporary registers needed for proper operation of ALU

CPU with Internal Bus (2)…



- Temporary registers Y and Z needed for proper operation of ALU
  - When an operation involving two operands is performed, one can be obtained from the internal bus, but the other must be obtained from another source
    - The AC could be used for this purpose, but this limits the flexibility of the system
    - Y provides temporary storage for the other input
  - The ALU is a combinatorial circuit with no internal storage
    - When control signals activate an ALU function, the input to the ALU is transformed to the output
    - The output of the ALU cannot be directly connected to the bus, because it will feed back to the input
    - Z provides temporary output storage

CPU with Internal Bus (3)
Execution sequence – An example

- Operation to perform addition of memory contents with Accumulator

t1: MAR	$\leftarrow$	(IR(address))
t2: MBR	$\leftarrow$	Memory
t3: Y	$\leftarrow$	(MBR)
t4: Z	$\leftarrow$	(AC) + (Y)
t5: AC	$\leftarrow$	(Z)

Problem

1. Using 4-stages pipeline (FI, DI, FO and EI), draw the timing diagram for 7 instructions.
2. Assuming that each stage of the above pipeline requires one unit of time cycle and each instruction goes through all the stages, how many time units are required for the 7 instructions?
3. Find the speedup factor for calculation (2) compared to the execution of the instructions without pipeline.

	1	2	3	4	5	6	7	8	9	10
Instruction 1	FI	DI	FO	EI						
Instruction 2		FI	DI	FO	EI					
Instruction 3			FI	DI	FO	EI				
Instruction 4				FI	DI	FO	EI			
Instruction 5					FI	DI	FO	EI		
Instruction 6						FI	DI	FO	EI	
Instruction 7							FI	DI	FO	EI

Total time units required for 7 instructions :**10**

$\tau_k = [k + (n-1)]\, \tau$

Speedup factor: 28/10=**2.8**

$$S_k = \frac{T_{1,n}}{T_{k,n}} = \frac{nk\tau}{[k+(n-1)]\tau} = \frac{nk}{k+(n-1)}$$

Summary

Lecture B - 03

Instruction Pipelining and Operation of Control Unit

- Instruction pipelining
  - Pipelining strategy
  - Pipeline performance
  - Pipeline hazards
  - Dealing with branches

- Control unit operation
  - Micro-operations
  - Hard-wired implementation
  - Micro-programmed control unit

- Slides adopted from:
  - Computer Organization and Architecture, 9th Edition

    William Stallings

    ISBN-10: 013293633X | ISBN-13: 9780132936330