

(Advanced) Computer Architecture

Prof. Dr. Hasan Hüseyin BALIK
(8th Week)

Outline

4. The central processing unit

- Processor Structure and Function
- Reduced Instruction Set Computers (RISCs)
- Instruction-Level Parallelism and Superscalar Processors
- Control Unit Operation and Microprogrammed Control



+

4.1 Processor Structure and Function

4.1 Outline

- Processor Organization
- Register Organization
- Instruction Cycle
- Instruction Pipelining
- Processor Organization for Pipelining
- The x86 Processor Family
- The ARM Processor

Processor Organization

Processor Requirements:

- Fetch instruction
 - The processor reads an instruction from memory (register, cache, main memory)
- Interpret instruction
 - The instruction is decoded to determine what action is required
- Fetch data
 - The execution of an instruction may require reading data from memory or an I/O module
- Process data
 - The execution of an instruction may require performing some arithmetic or logical operation on data
- Write data
 - The results of an execution may require writing data to memory or an I/O module
- In order to do these things the processor needs to store some data temporarily and therefore needs a small internal memory

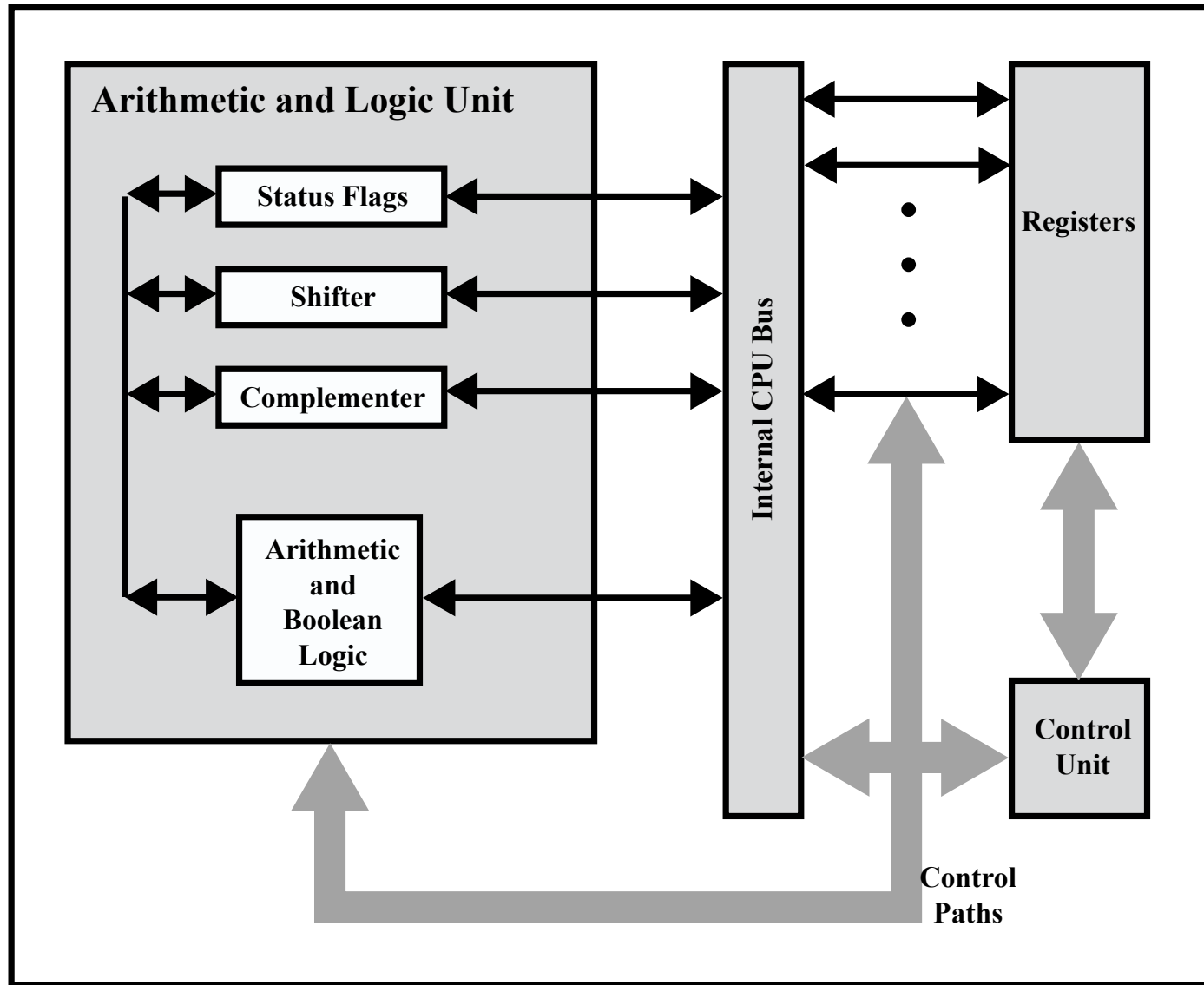


Figure: Internal Structure of the CPU

Register Organization

- Within the processor there is a set of registers that function as a level of memory above main memory and cache in the hierarchy
- The registers in the processor perform two roles:

User-Visible Registers


- Enable the machine or assembly language programmer to minimize main memory references by optimizing use of registers

Control and Status Registers

- Used by the control unit to control the operation of the processor and by privileged operating system programs to control the execution of programs

User-Visible Registers

Referenced by means of
the machine language
that the processor
executes



Categories:

- **General purpose**
 - Can be assigned to a variety of functions by the programmer
- **Data**
 - May be used only to hold data and cannot be employed in the calculation of an operand address
- **Address**
 - May be somewhat general purpose or may be devoted to a particular addressing mode
 - Examples: segment pointers, index registers, stack pointer
- **Condition codes**
 - Also referred to as *flags*
 - Bits set by the processor hardware as the result of operations

Control and Status Registers

Four registers are essential to instruction execution:

- Program counter (PC)
 - Contains the address of an instruction to be fetched
- Instruction register (IR)
 - Contains the instruction most recently fetched
- Memory address register (MAR)
 - Contains the address of a location in memory
- Memory buffer register (MBR)
 - Contains a word of data to be written to memory or the word most recently read

Program Status Word (PSW)



Register or set of registers that contain status information

Common fields or flags include:

- **Sign:** Contains the sign bit of the result of the last arithmetic operation
- **Zero:** Set when the result is 0
- **Carry:** Set if an operation resulted in a carry
- **Equal:** Set if a logical compare result is equality
- **Overflow:** Used to indicate arithmetic overflow
- **Interrupt Enable/Disable:** Used to enable or disable interrupts
- **Supervisor:** Indicates whether the processor is executing in supervisor or user mode

Data registers	
D0	
D1	
D2	
D3	
D4	
D5	
D6	
D7	

Address registers	
A0	
A1	
A2	
A3	
A4	
A5	
A6	
A7	

Program status	
Program counter	
	Status register

(a) MC68000

General registers	
AX	Accumulator
BX	Base
CX	Count
DX	Data

Pointers & index	
SP	Stack ptr
BP	Base ptr
SI	Source index
DI	Dest index

Segment	
CS	Code
DS	Data
SS	Stack
ES	Extrat

Program status	
Flags	
Instr ptr	

(b) 8086

General Registers		
EAX		AX
EBX		BX
ECX		CX
EDX		DX
ESP		SP
EBP		BP
ESI		SI
EDI		DI

Program Status	
FLAGS Register	
Instruction Pointer	

(c) 80386 - Pentium 4

Figure: Example Microprocessor Register Organizations

Instruction Cycle

Includes the following stages:

Fetch

Read the next instruction from memory into the processor

Execute

Interpret the opcode and perform the indicated operation

Interrupt

If interrupts are enabled and an interrupt has occurred, save the current process state and service the interrupt

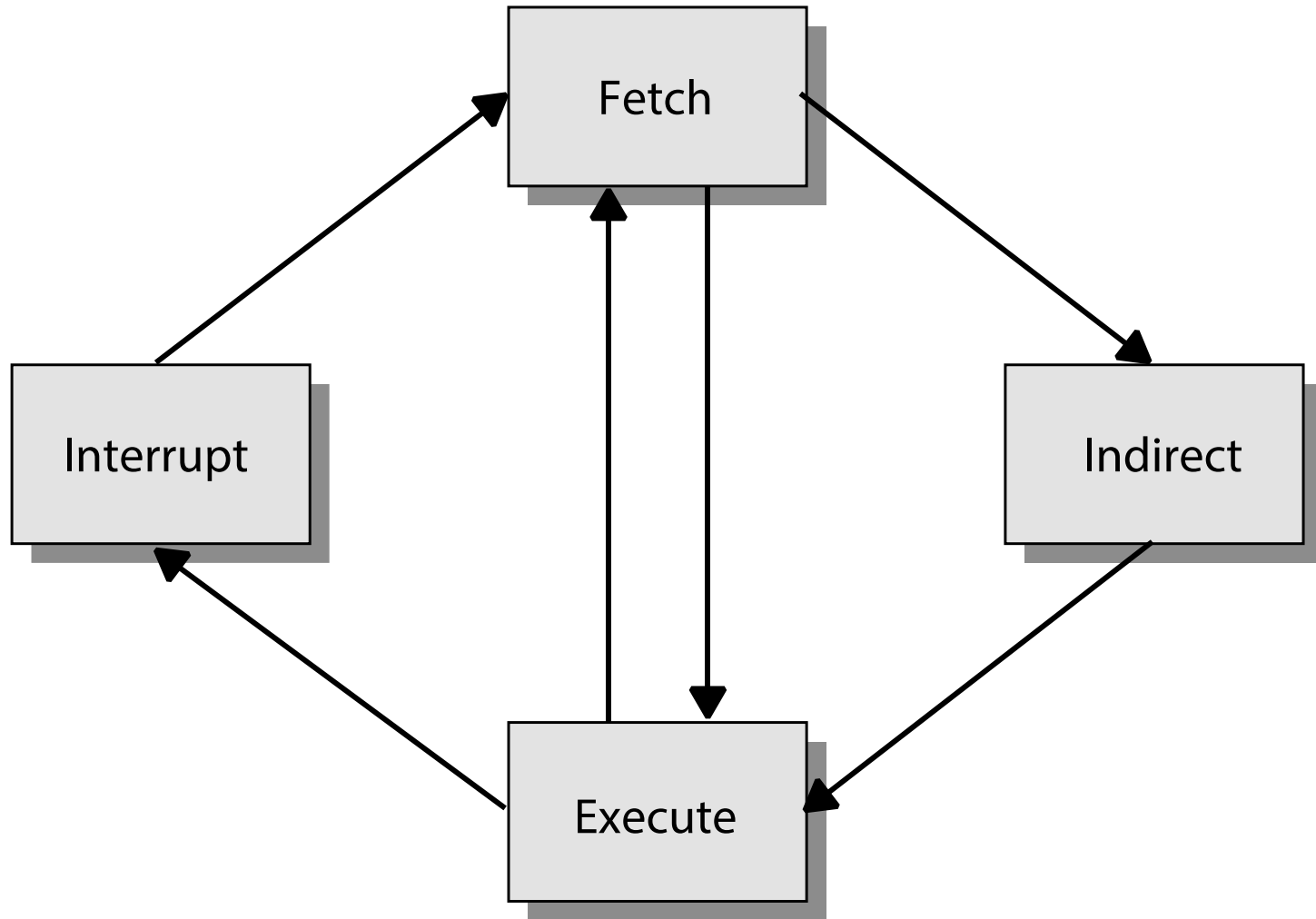


Figure: The Instruction Cycle

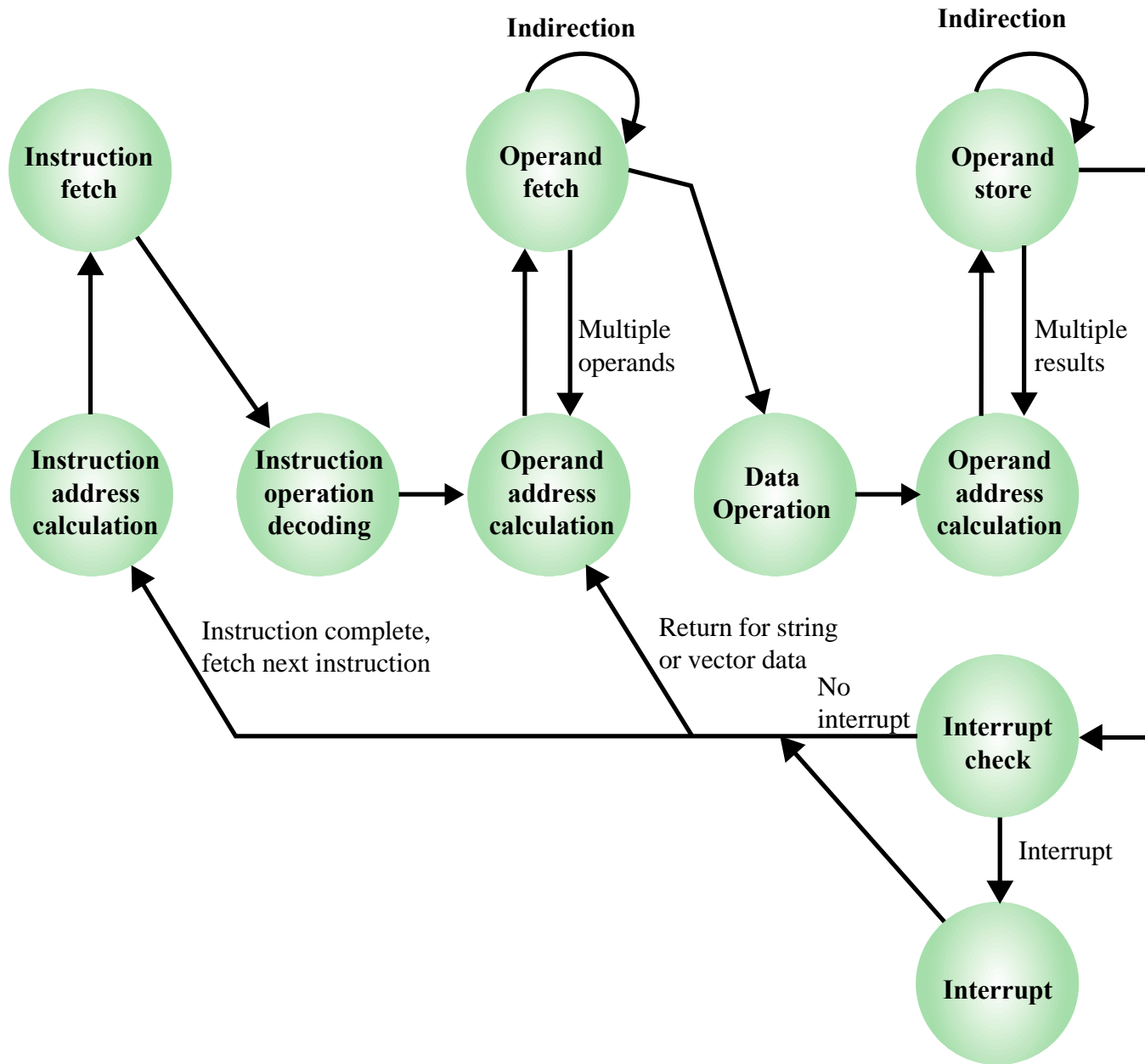
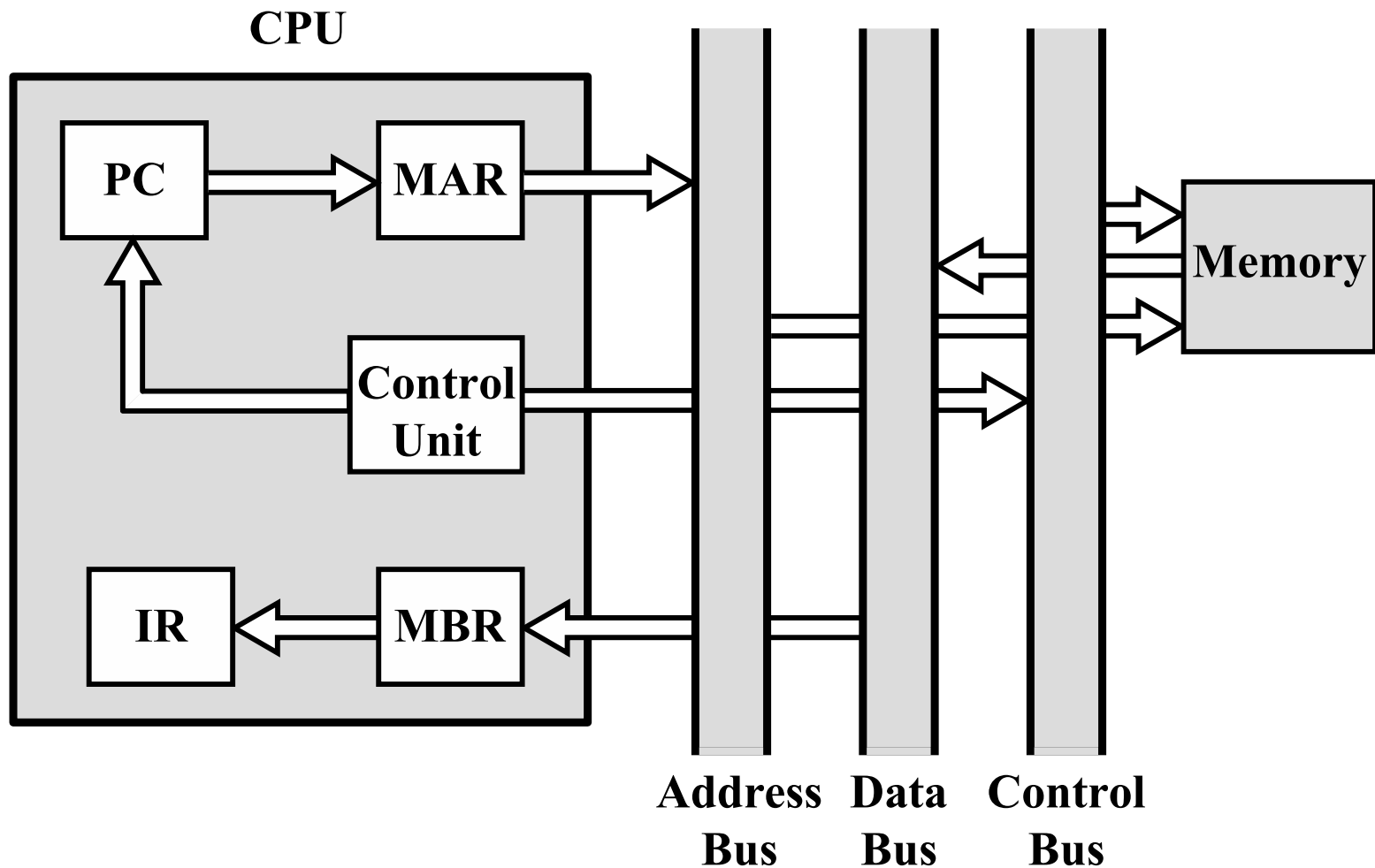


Figure: Instruction Cycle State Diagram



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

Figure: Data Flow, Fetch Cycle

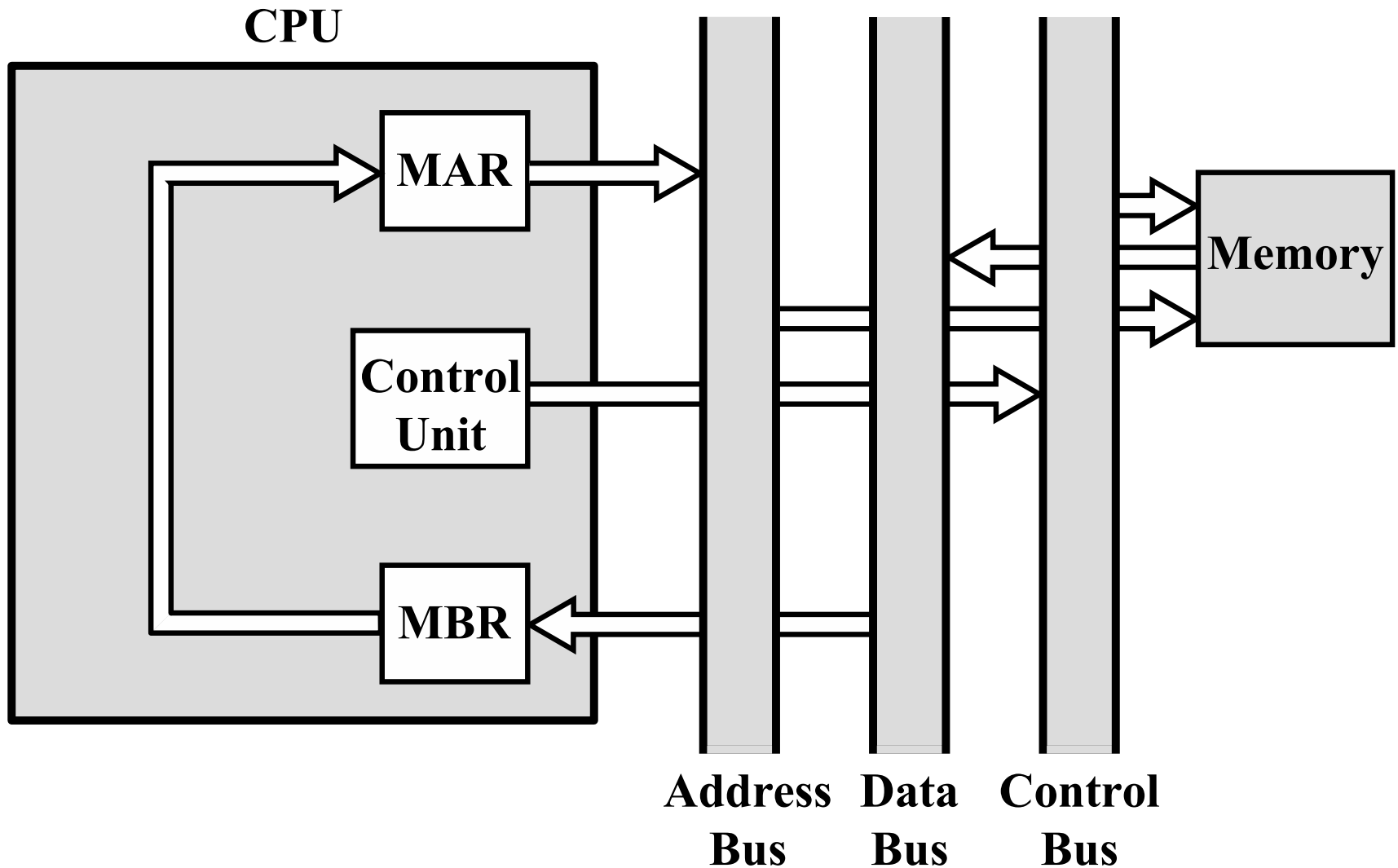


Figure: Data Flow, Indirect Cycle

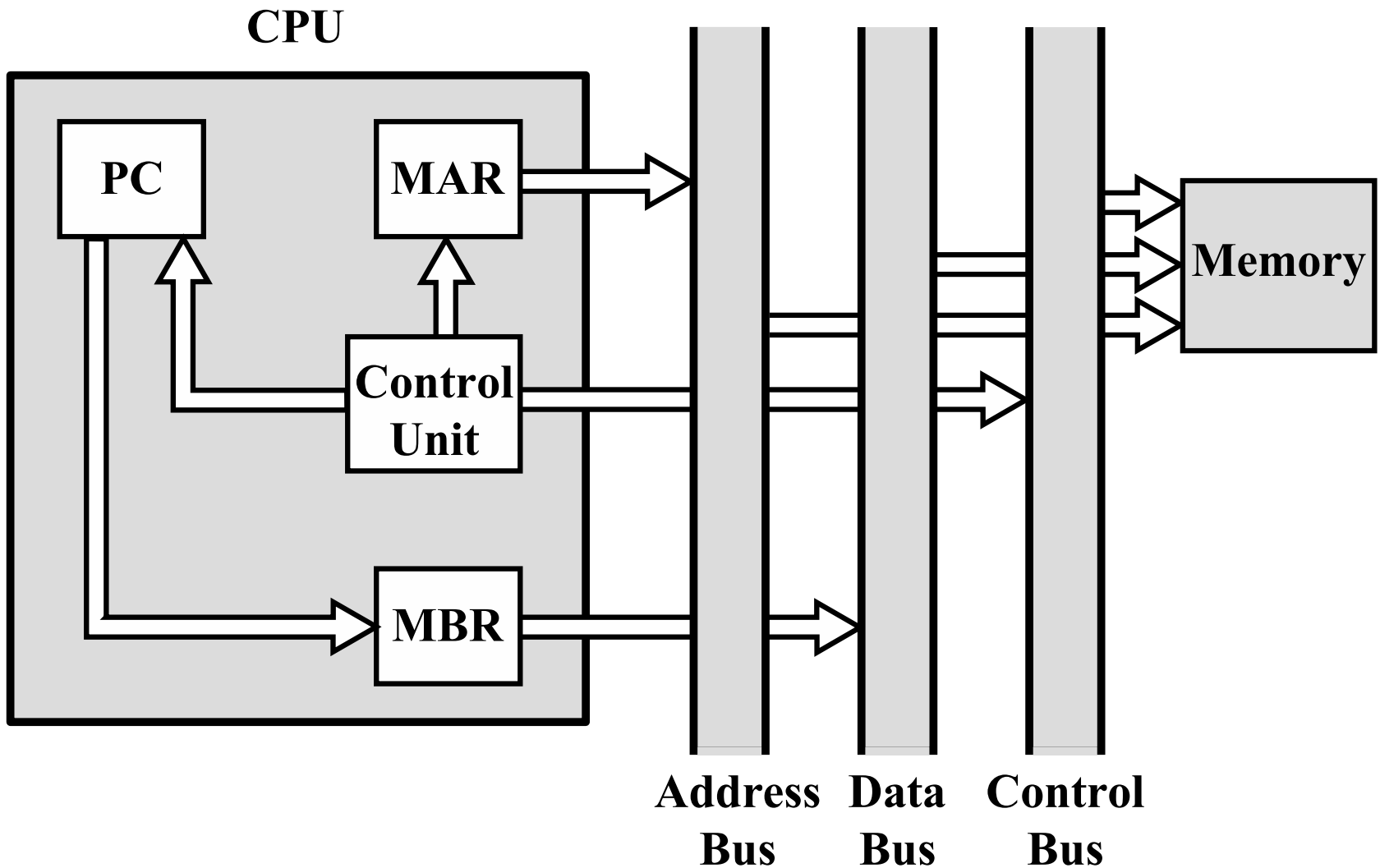


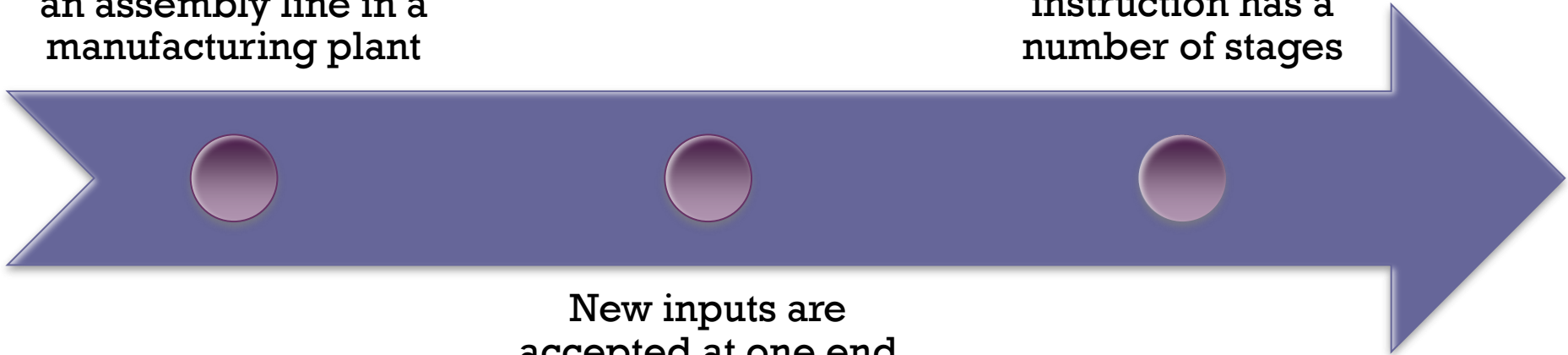
Figure: Data Flow, Interrupt Cycle

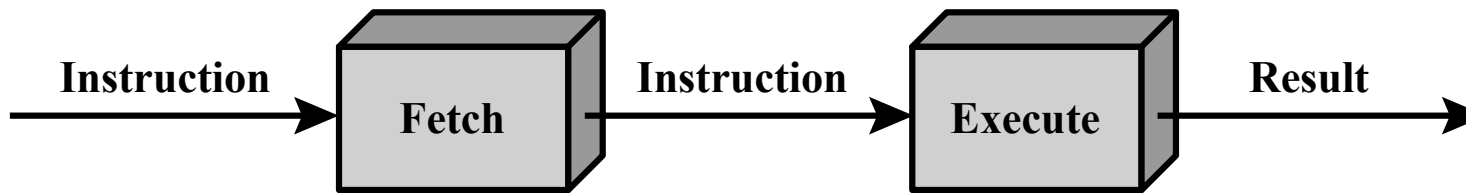
Pipelining Strategy

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

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





(a) Simplified view

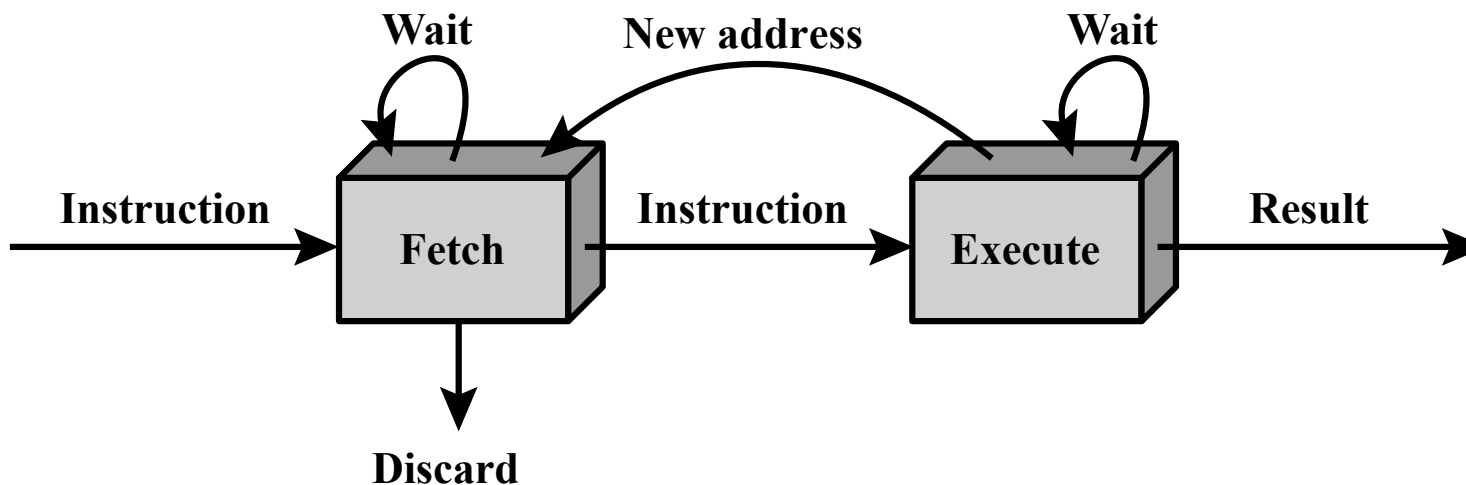


Figure: Two-Stage Instruction Pipeline

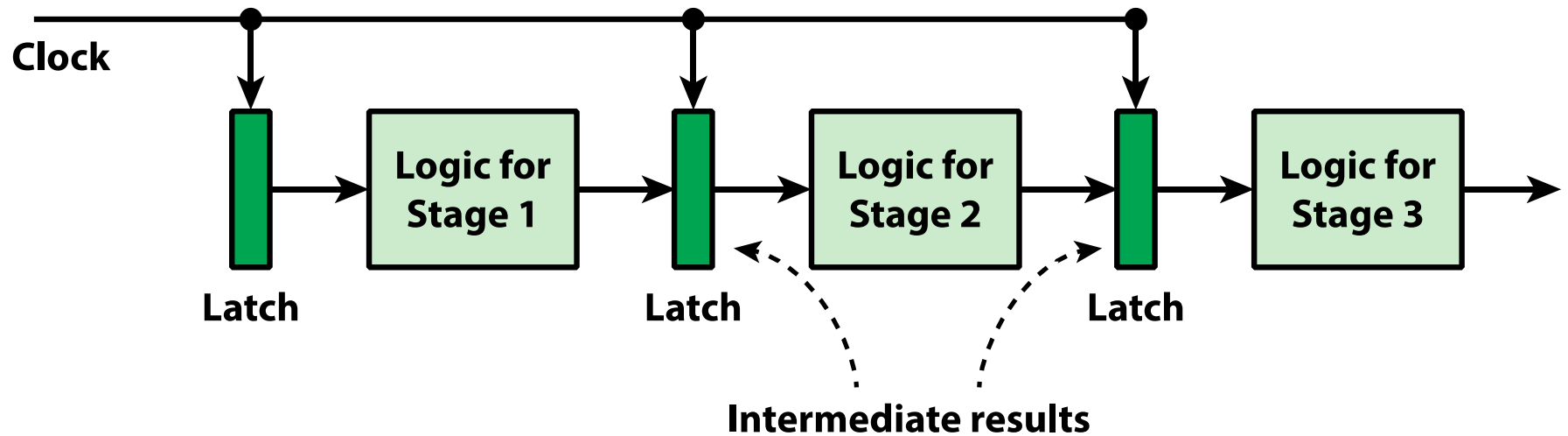


Figure: Simplified Pipeline Architecture

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

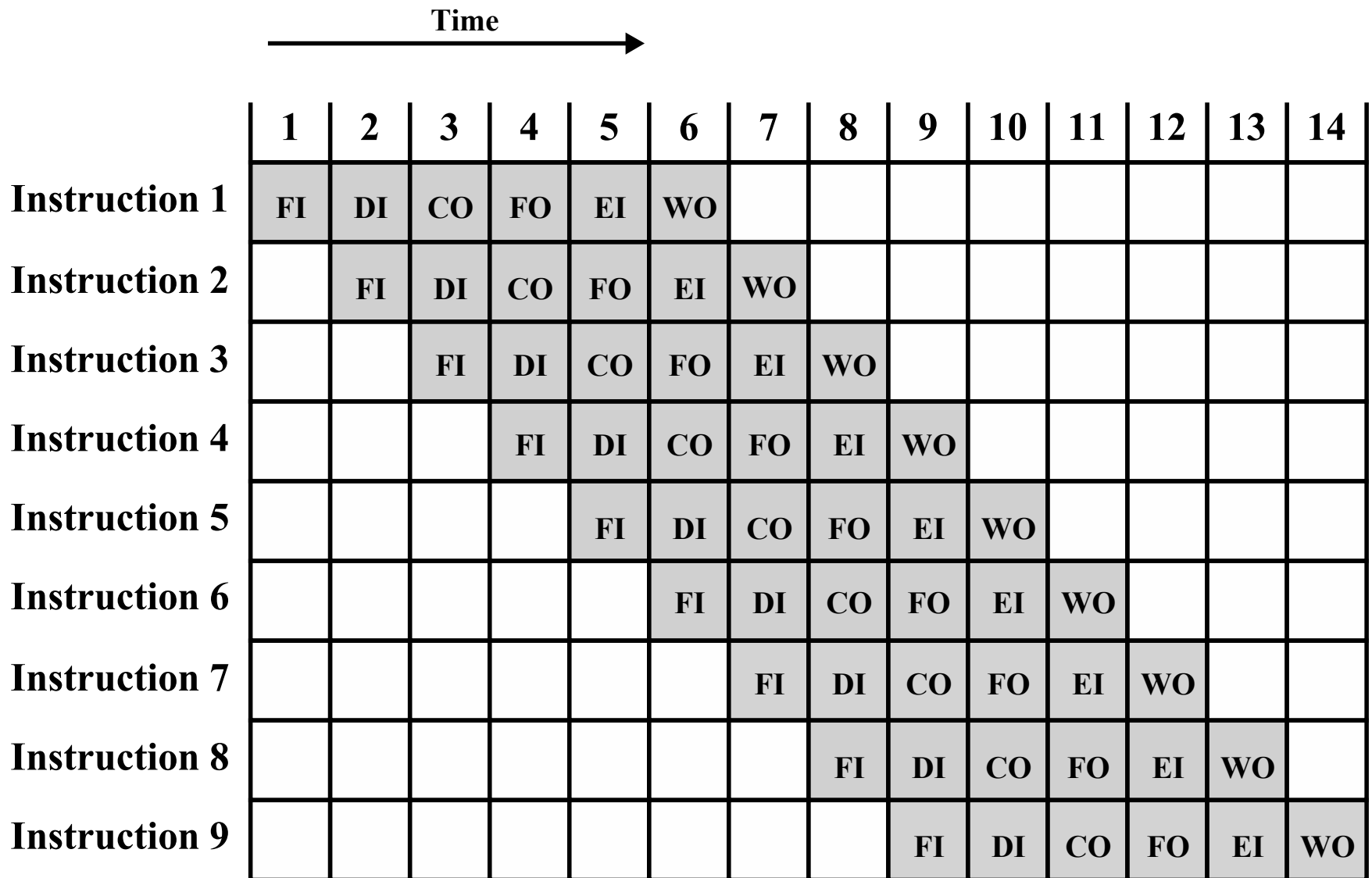


Figure: Timing Diagram for Instruction Pipeline Operation

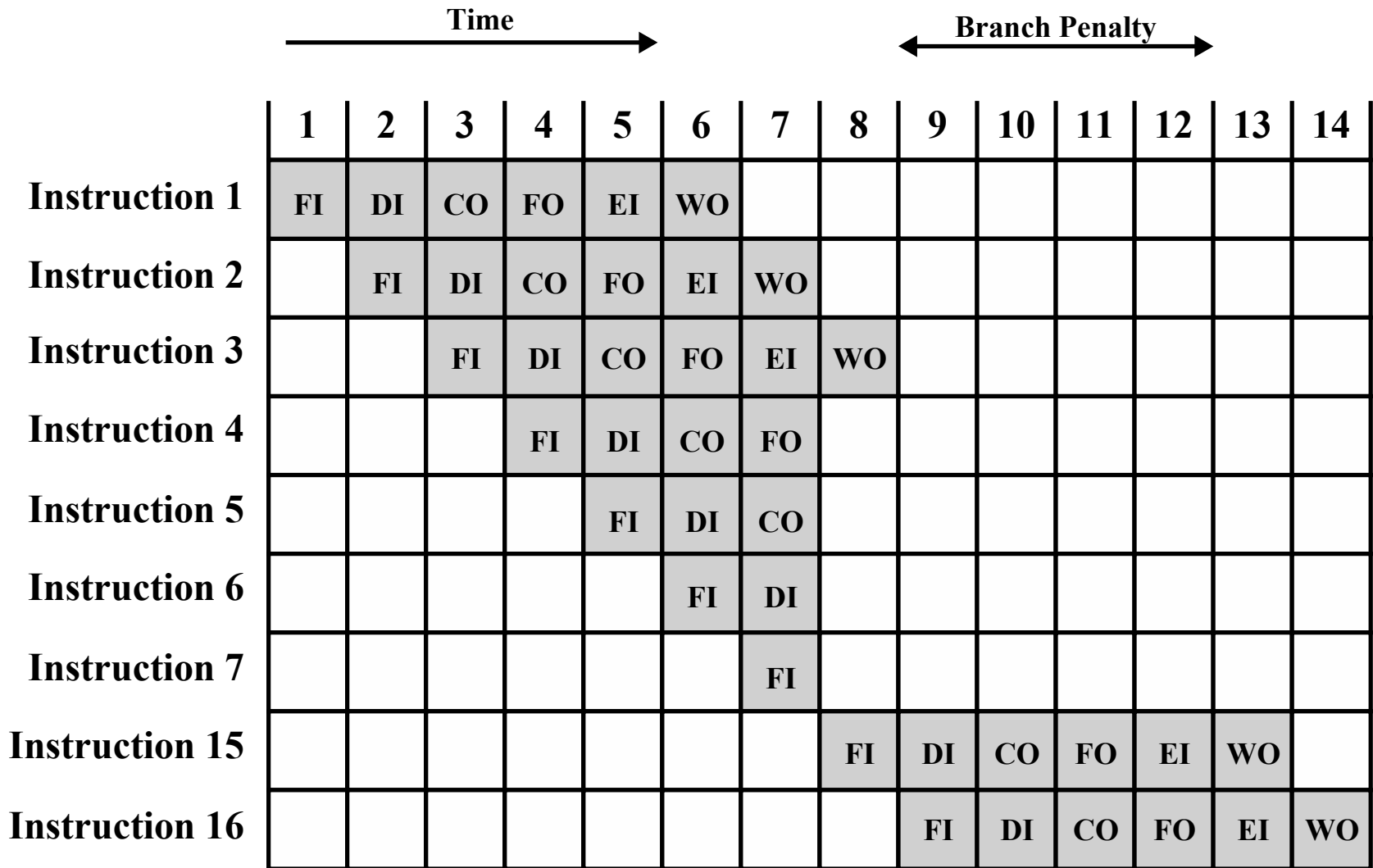


Figure: The Effect of a Conditional Branch on Instruction Pipeline Operation

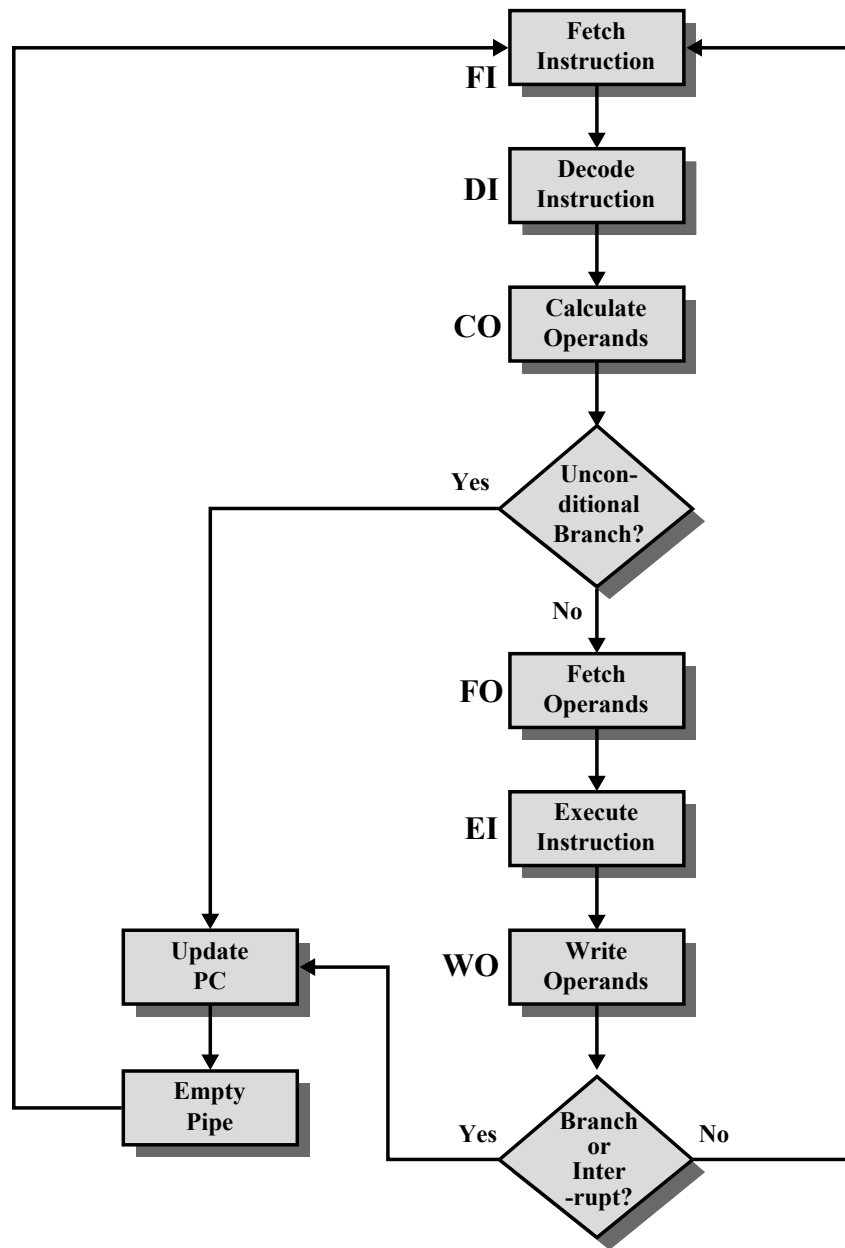


Figure: Six-Stage CPU Instruction Pipeline

	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

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	I15					I3
9	I16	I15				
10		I16	I15			
11			I16	I15		
12				I16	I15	
13					I16	I15
14						I16

Figure: An Alternative Pipeline Depiction

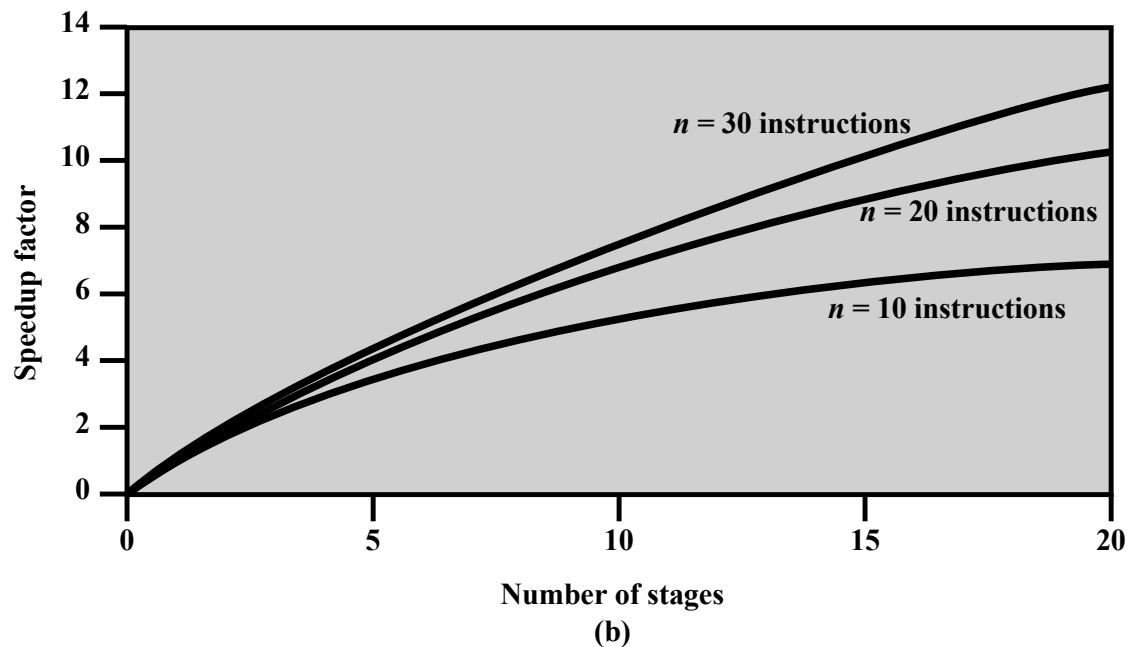
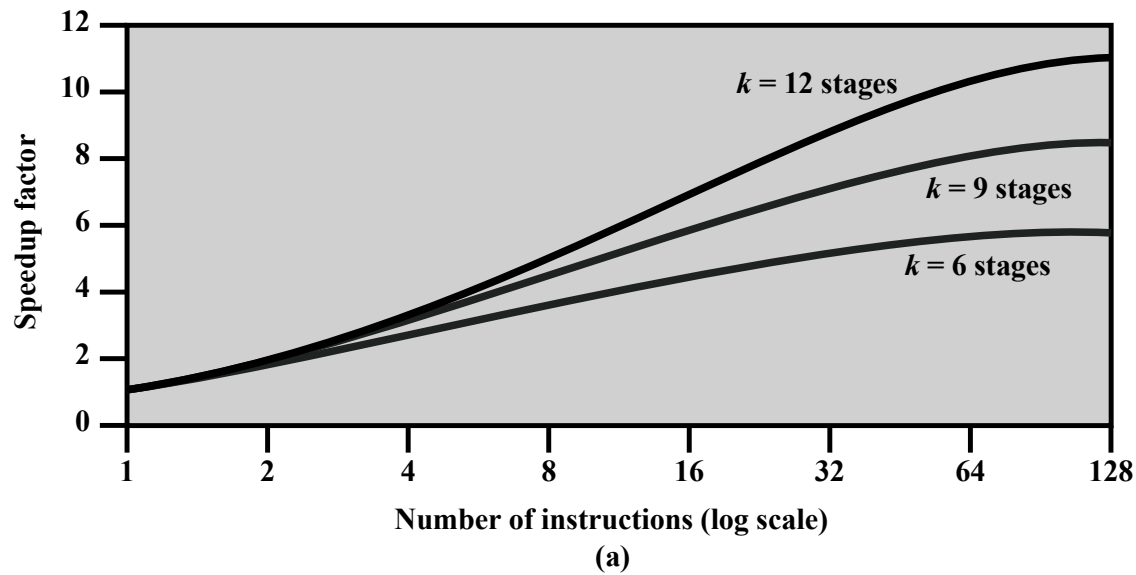


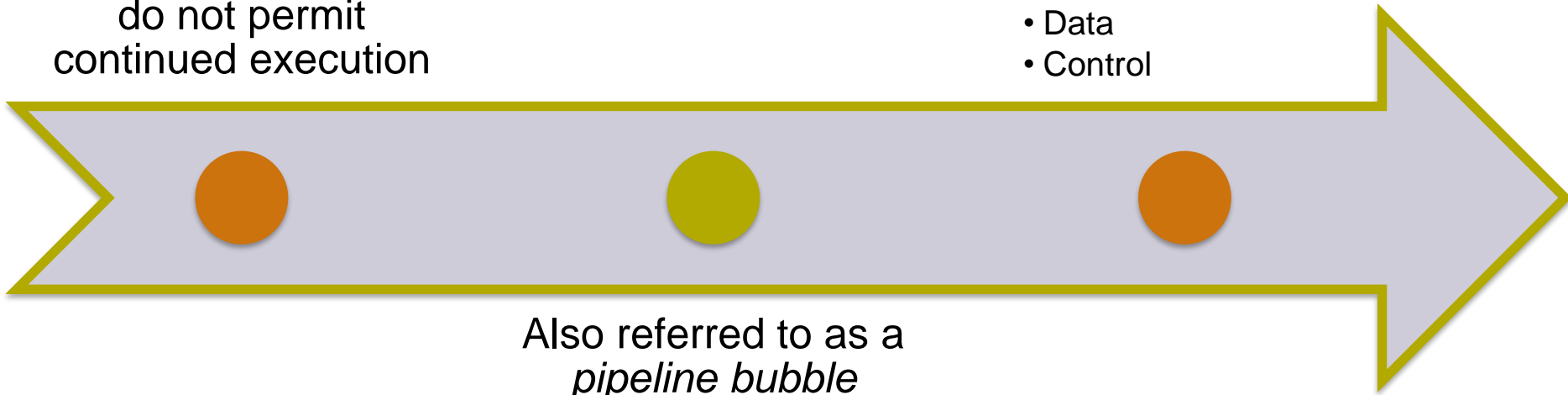
Figure: Speedup Factors with Instruction Pipelining

Pipeline Hazards

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

There are three types of hazards:

- Resource
- Data
- Control



		Clock cycle								
		1	2	3	4	5	6	7	8	9
Instrucion	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
Instrucion	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: 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

Figure: Example of Data Hazard

Types of Data Hazard

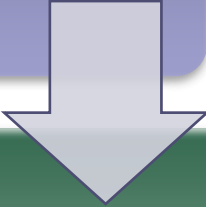
- Read after write (RAW), or true dependency
 - An instruction modifies a register or memory location
 - Succeeding instruction reads data in memory or register location
 - Hazard occurs if the read takes place before write operation is complete
- Write after read (WAR), or antidependency
 - An instruction 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

- 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
- IBM 360/91 uses this approach

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:

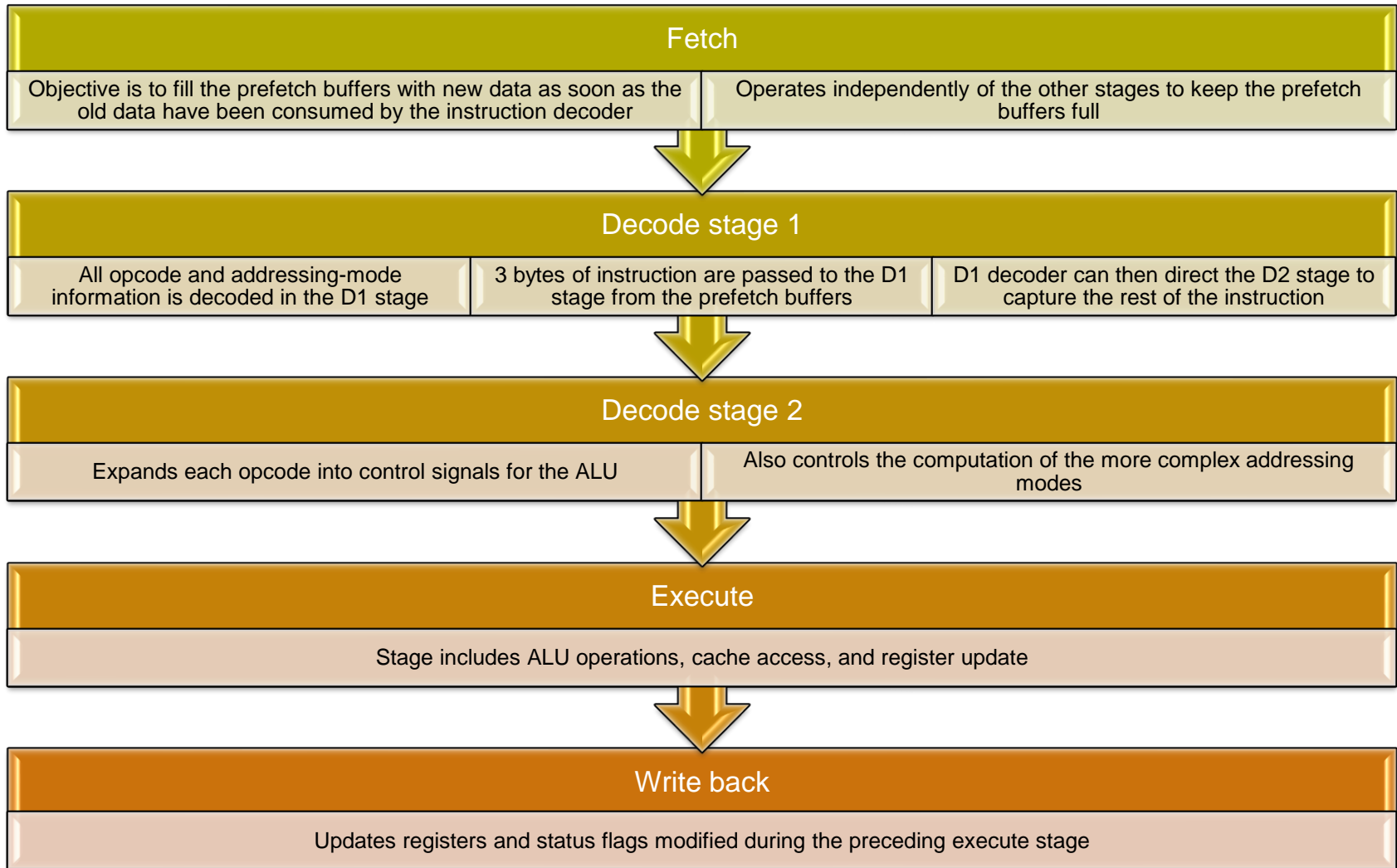
1. Predict never taken
2. Predict always taken
3. Predict by opcode

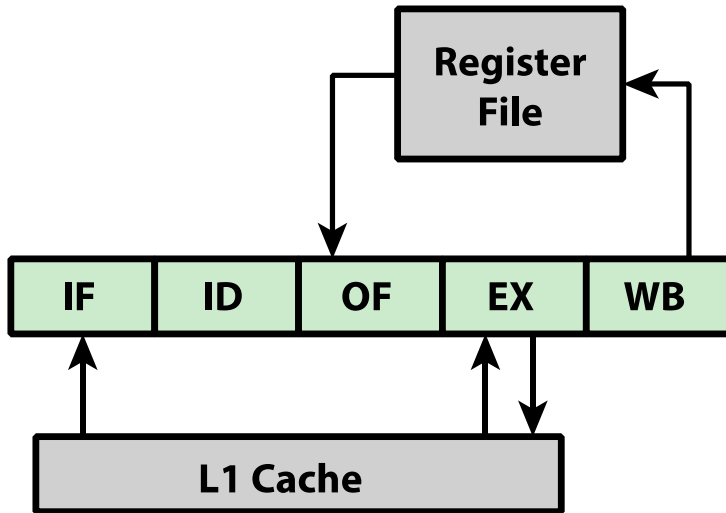
- These approaches are static
- They do not depend on the execution history up to the time of the conditional branch instruction

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

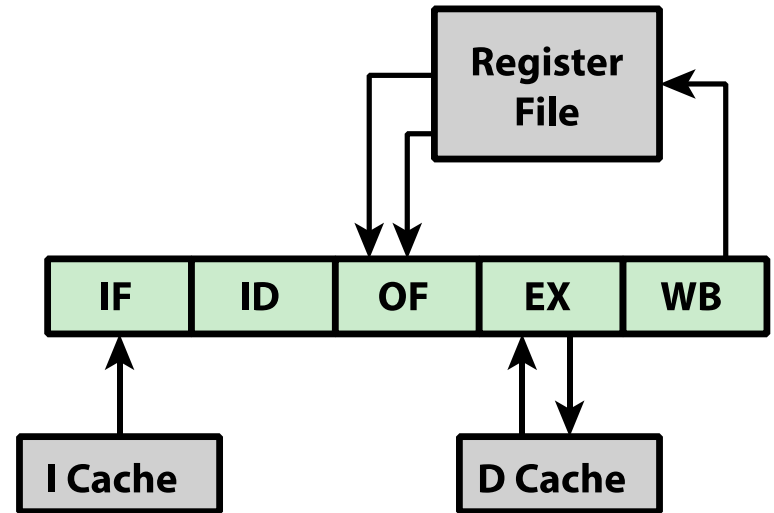
- These approaches are dynamic
- They depend on the execution history

Intel 80486 Pipelining





(a) Simple Pipeline Organization



(b) Performance Enhancements

Figure: Approaches to Pipeline Organization

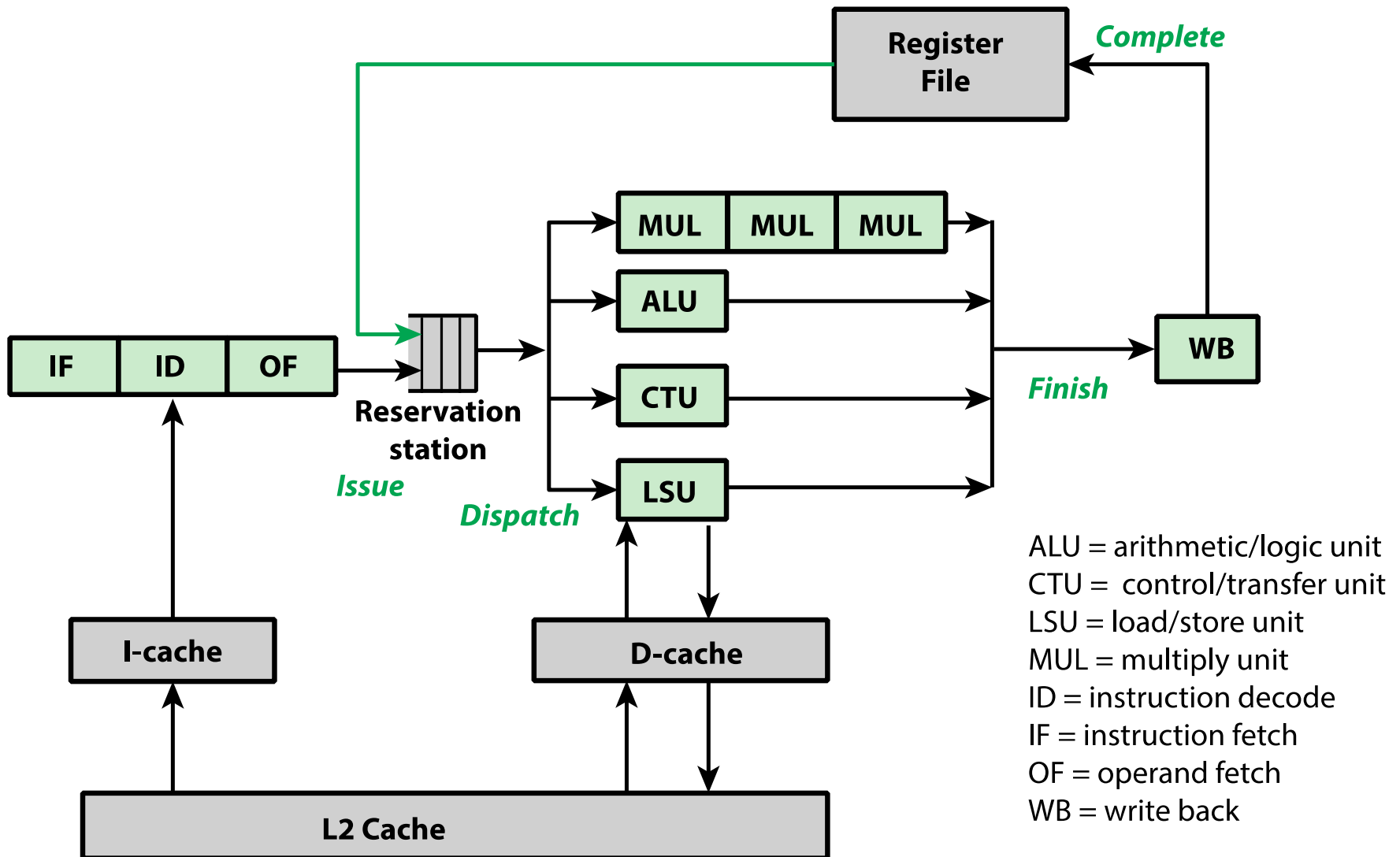


Figure: Improved Pipeline Organization

Interrupt Processing

Interrupts and Exceptions

- Interrupts
 - Generated by a signal from hardware and it may occur at random times during the execution of a program
 - Maskable
 - Nonmaskable
- Exceptions
 - Generated from software and is provoked by the execution of an instruction
 - Processor detected
 - Programmed
- Interrupt vector table
 - Every type of interrupt is assigned a number
 - Number is used to index into the interrupt vector table

The ARM Processor

ARM is primarily a RISC system with the following attributes:

- Moderate array of uniform registers
- A load/store model of data processing in which operations only perform on operands in registers and not directly in memory
- A uniform fixed-length instruction of 32 bits for the standard set and 16 bits for the Thumb instruction set
- Separate arithmetic logic unit (ALU) and shifter units
- A small number of addressing modes with all load/store addresses determined from registers and instruction fields
- Auto-increment and auto-decrement addressing modes are used to improve the operation of program loops
- Conditional execution of instructions minimizes the need for conditional branch instructions, thereby improving pipeline efficiency, because pipeline flushing is reduced

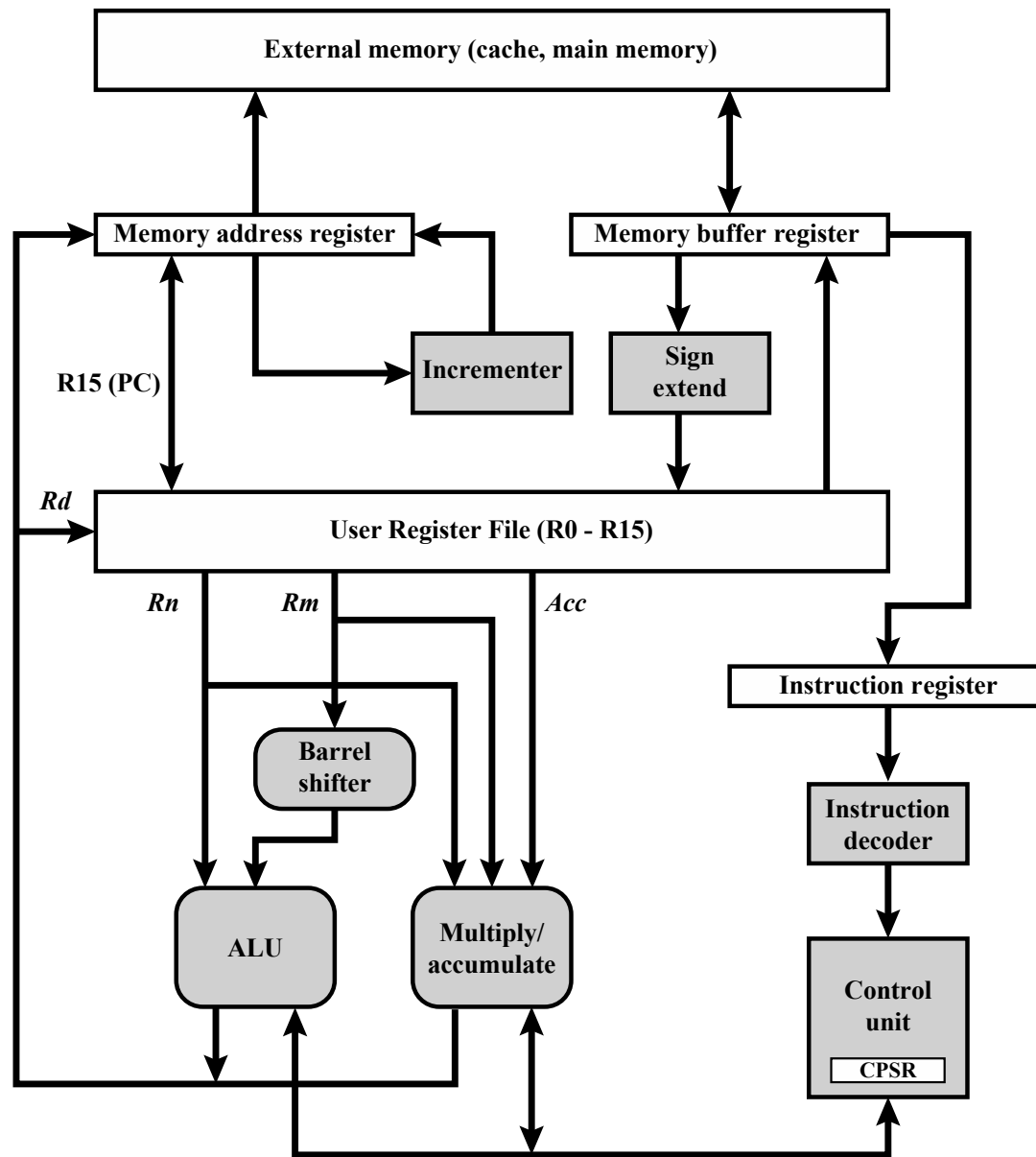
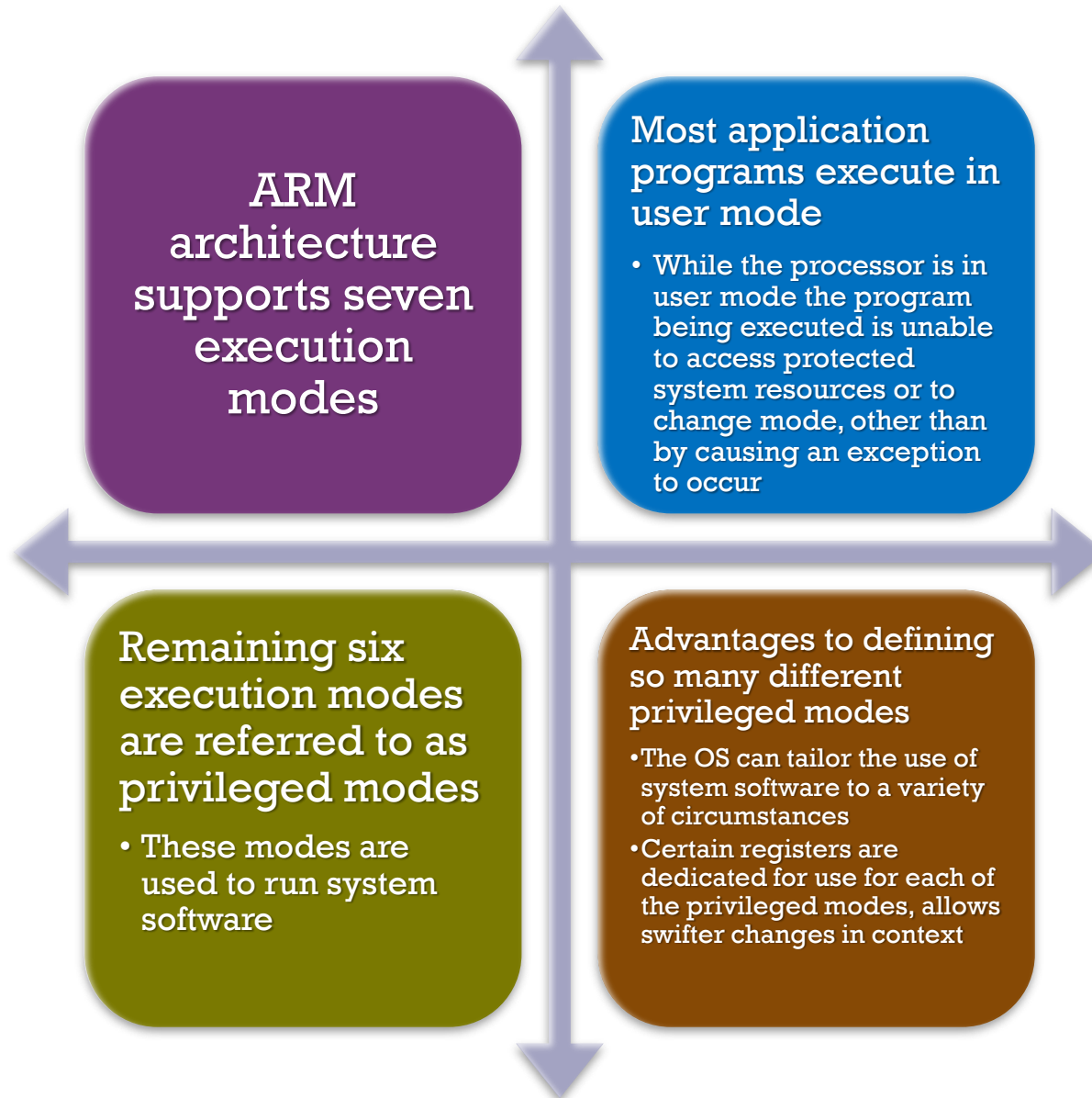


Figure: Simplified ARM Organization

Processor Modes



Exception Modes

