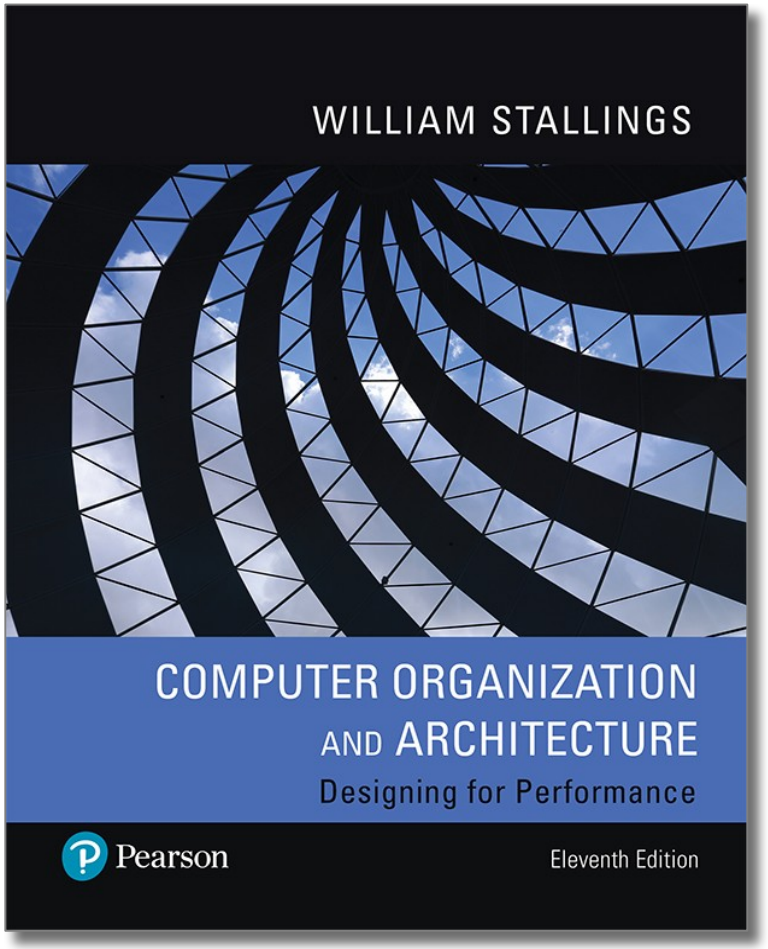
Computer Organization and Architecture
Designing for Performance

11th Edition



Chapter 16

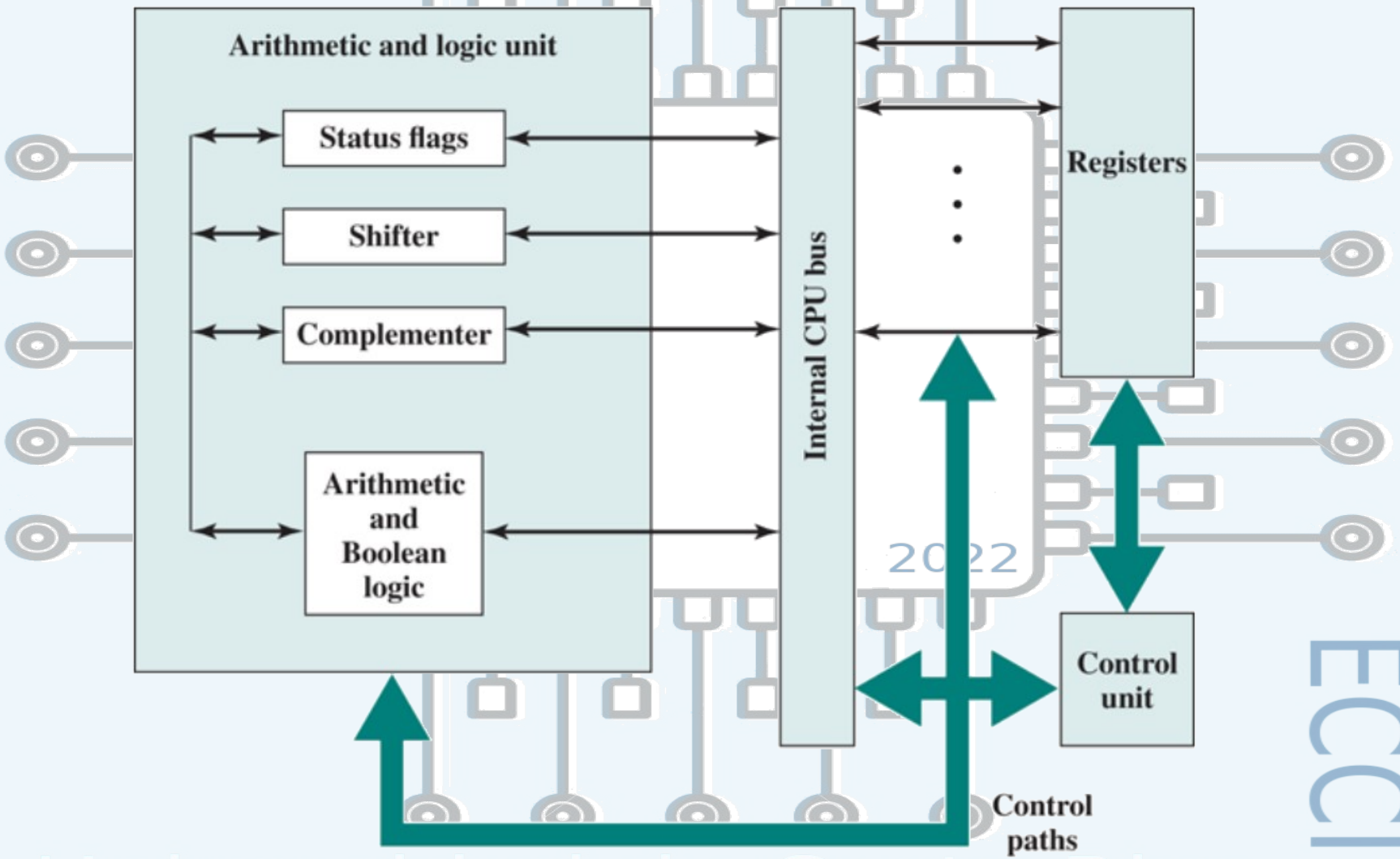
Processor Structure and Function

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 16.1
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

Table 16.1
Condition Codes

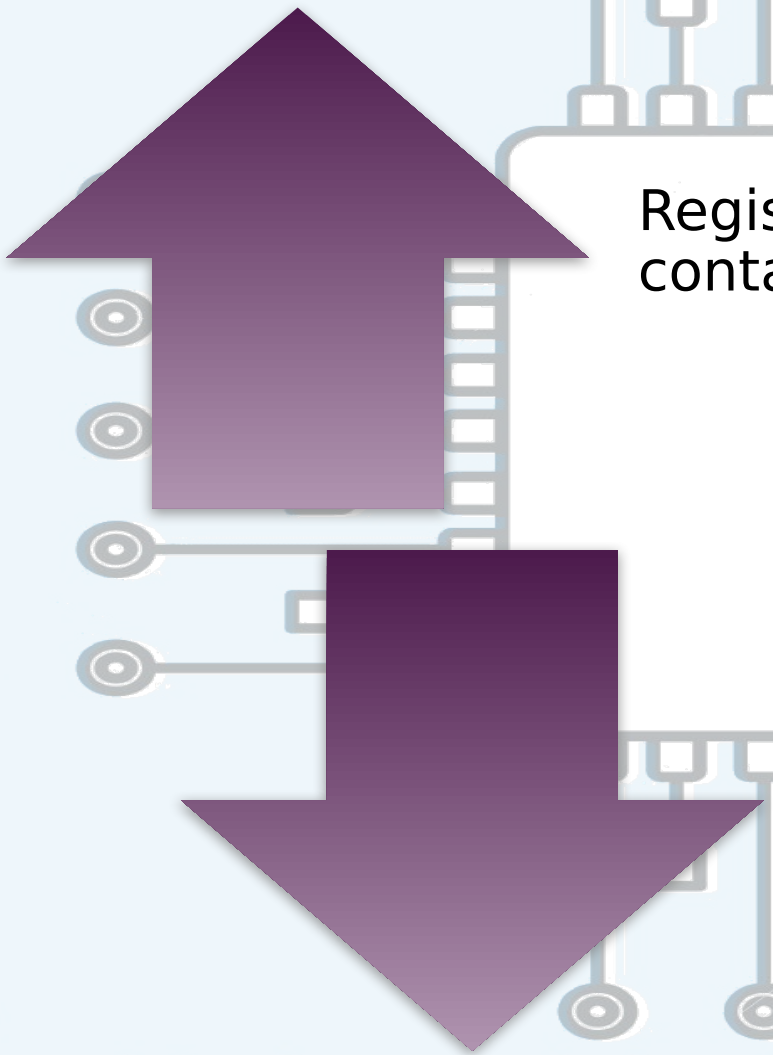
Advantages	Disadvantages
1. Because condition codes are set by normal arithmetic and data movement instructions, they should reduce the number of COMPARE and TEST instructions needed. 2. Conditional instructions such as BRANCH are simplified relative to composite instructions, such as TEST and BRANCH. 3. Condition codes facilitate multiway branches. For example, a TEST instruction can be followed by two branches, one on less than or equal to zero and one on greater than zero. 4. Condition codes can be saved on the stack during subroutine calls along with other register information.	1. Condition codes add complexity, both to the hardware and software. Condition code bits are often modified in different ways by different instructions, making life more difficult for both the microprogrammer and compiler writer. 2. Condition codes are irregular; they are typically not part of the main data path, so they require extra hardware connections. 3. Often condition code machines must add special non-condition-code instructions for special situations anyway, such as bit checking, loop control, and atomic semaphore operations. 4. In a pipelined implementation, condition codes require special synchronization to avoid conflicts.

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)

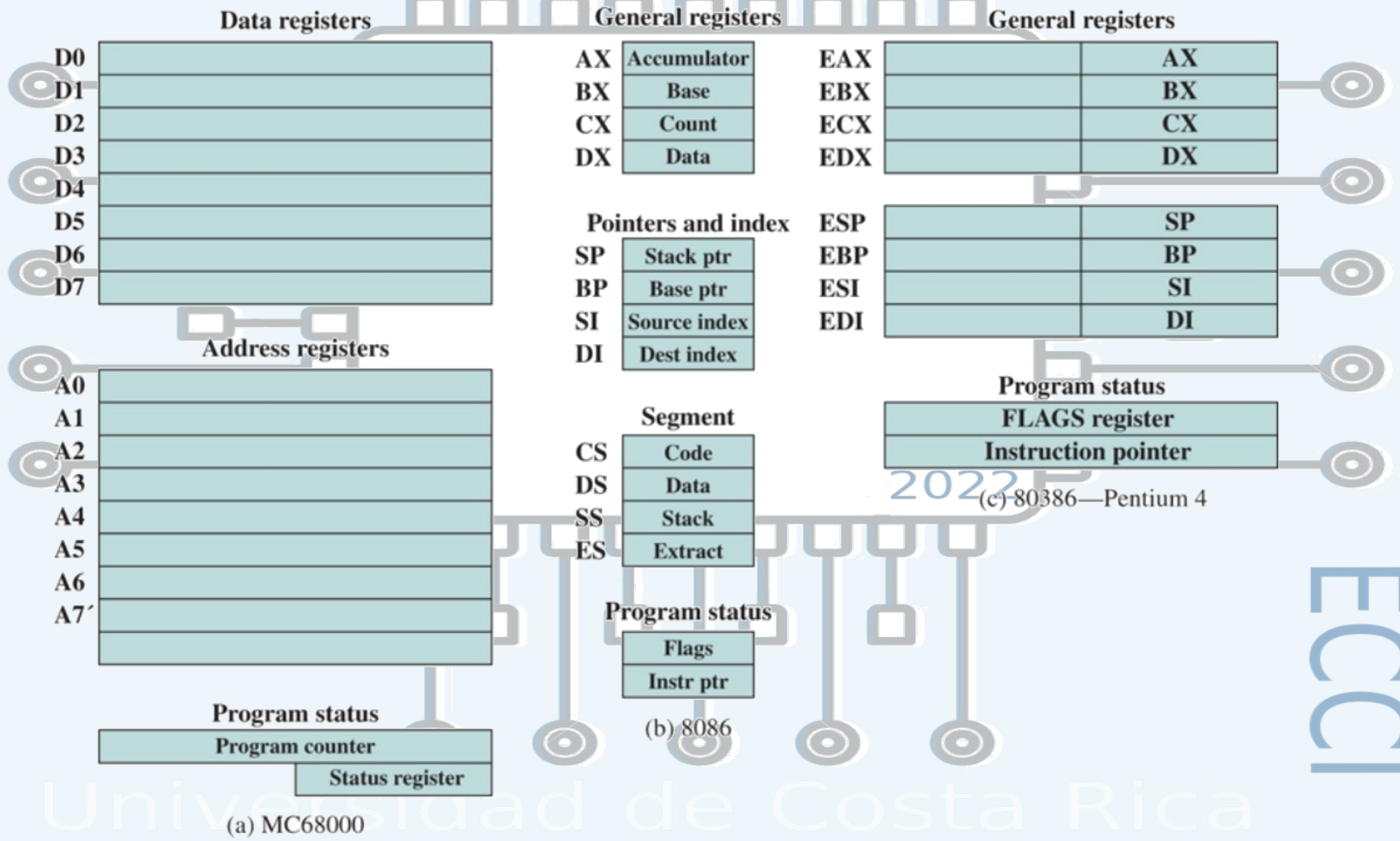


Register or set of registers that contain status information

Common fields or flags include:

- Sign
- Zero
- Carry
- Equal
- Overflow
- Interrupt Enable/Disable
- Supervisor

Figure 16.2 Example Microprocessor Register Organizations



Instruction Cycle

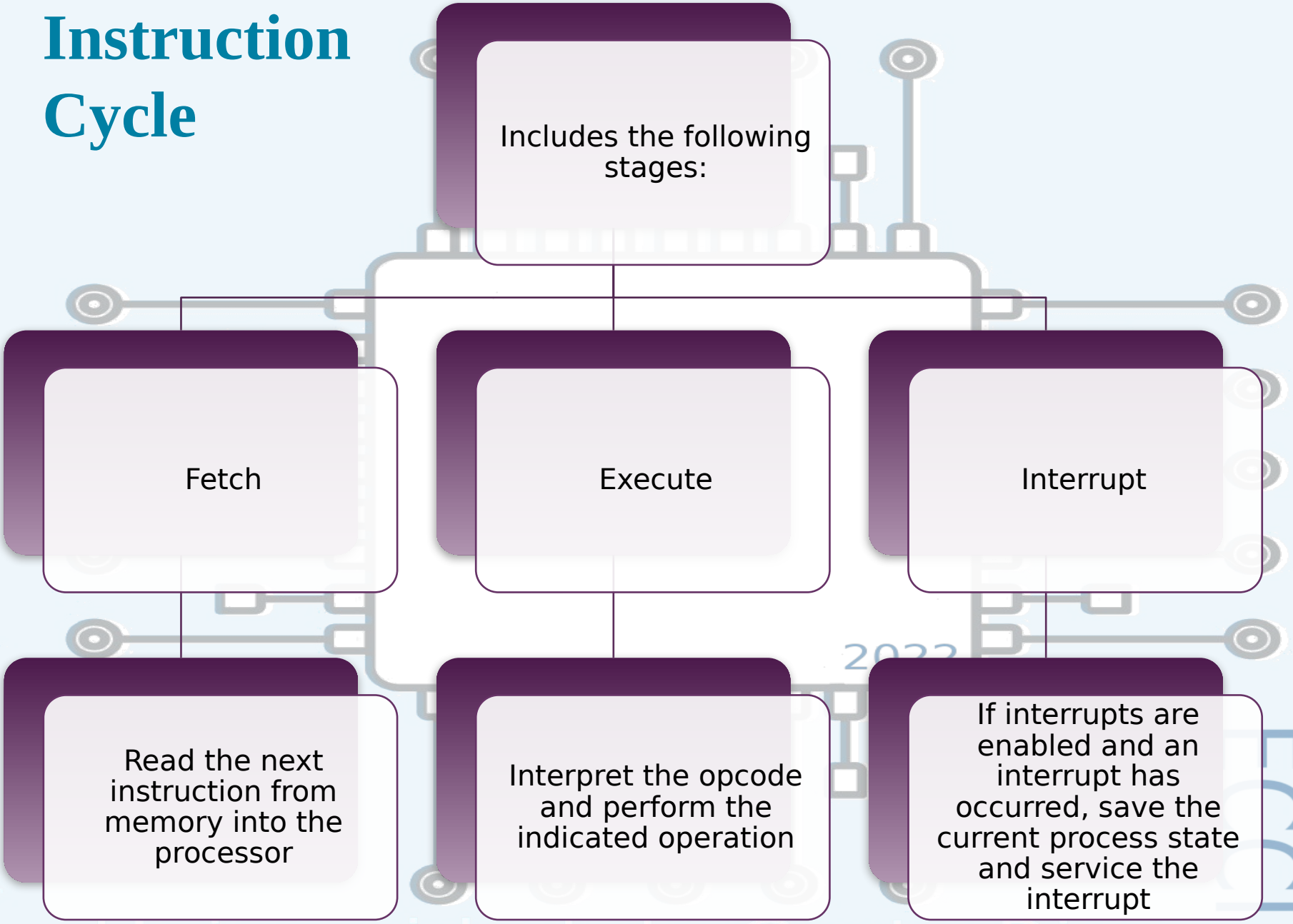


Figure 16.3
The Instruction Cycle

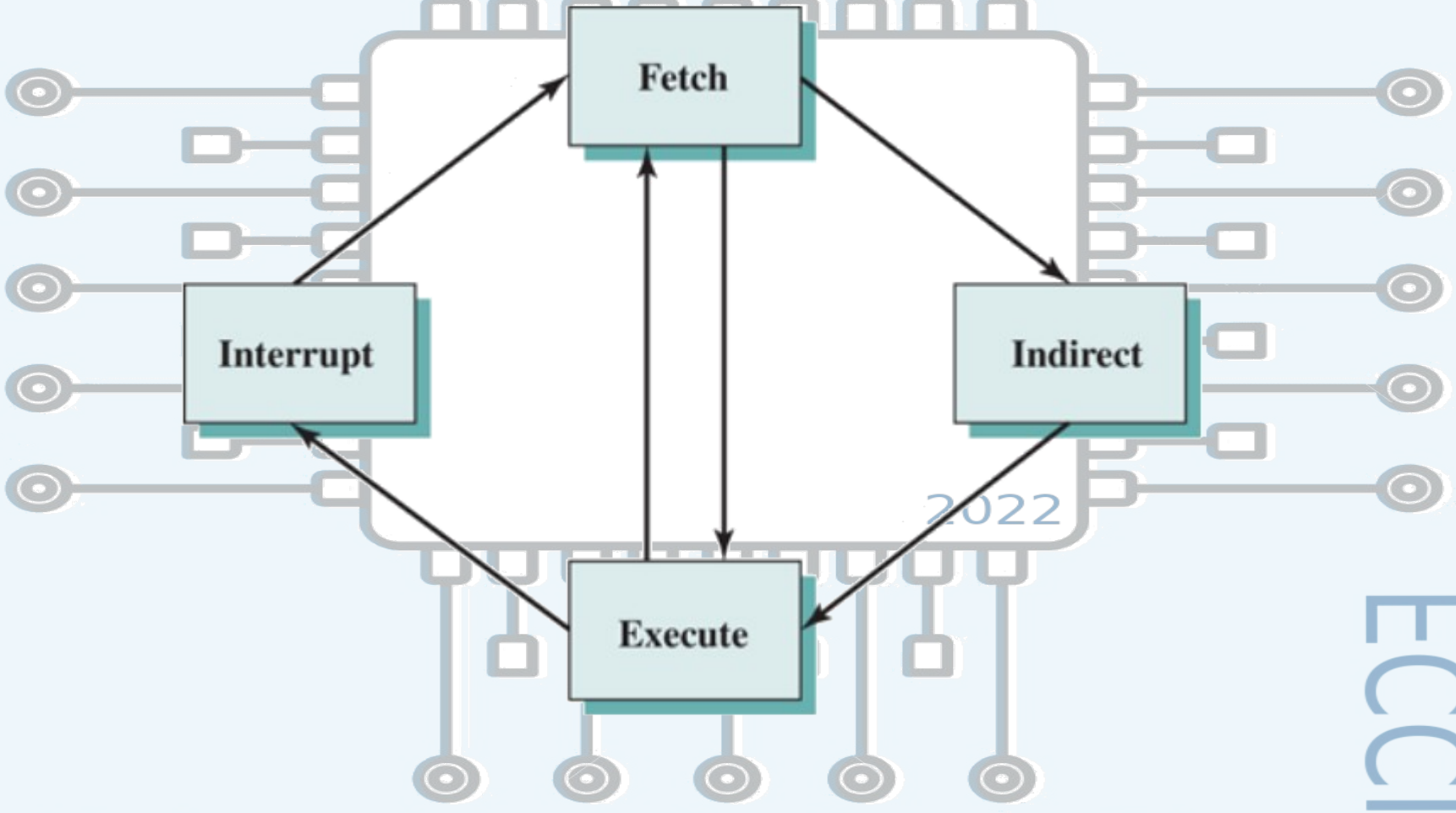


Figure 16.4
Instruction Cycle State Diagram

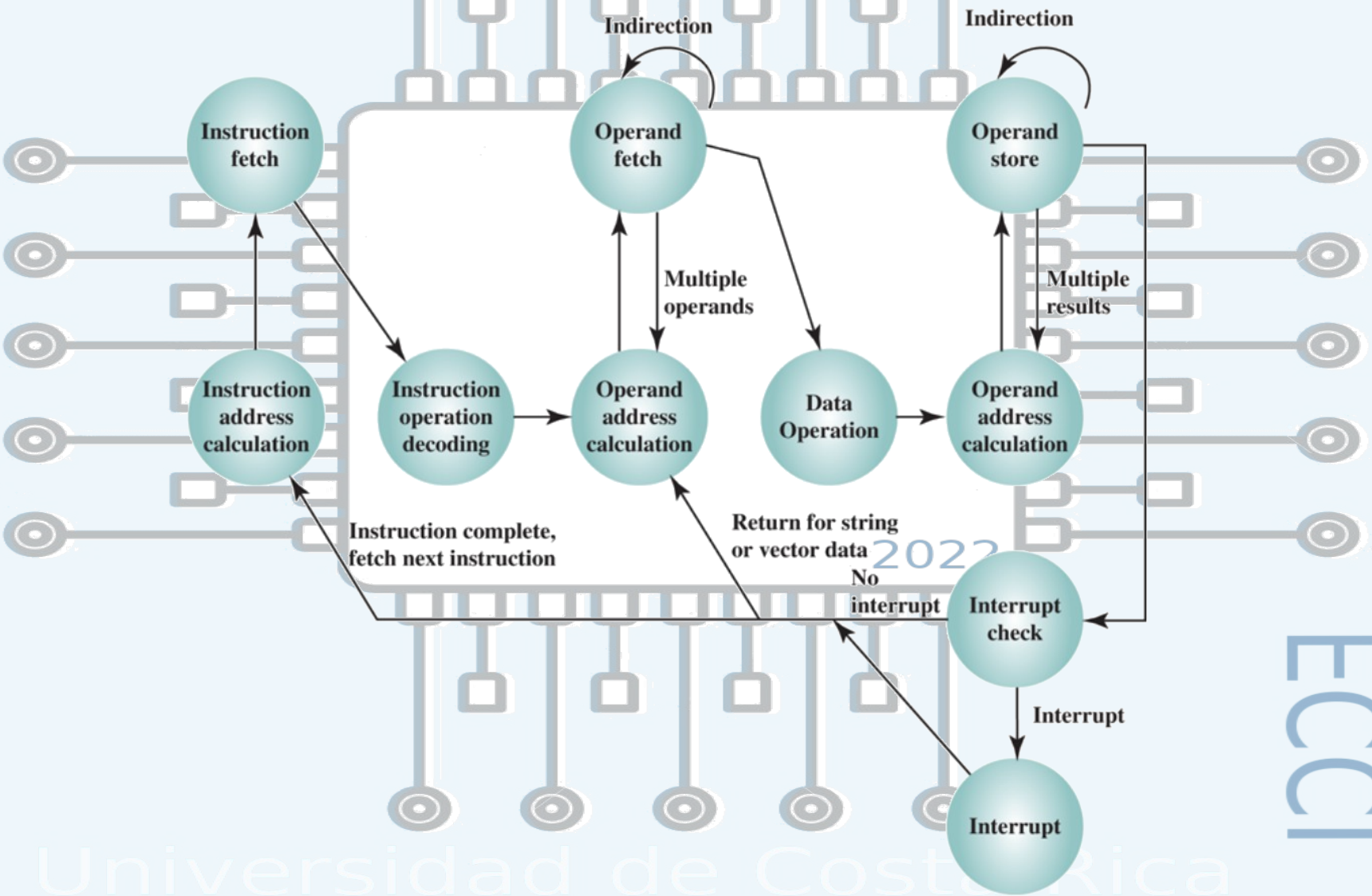


Figure 16.5
Data Flow, Fetch Cycle

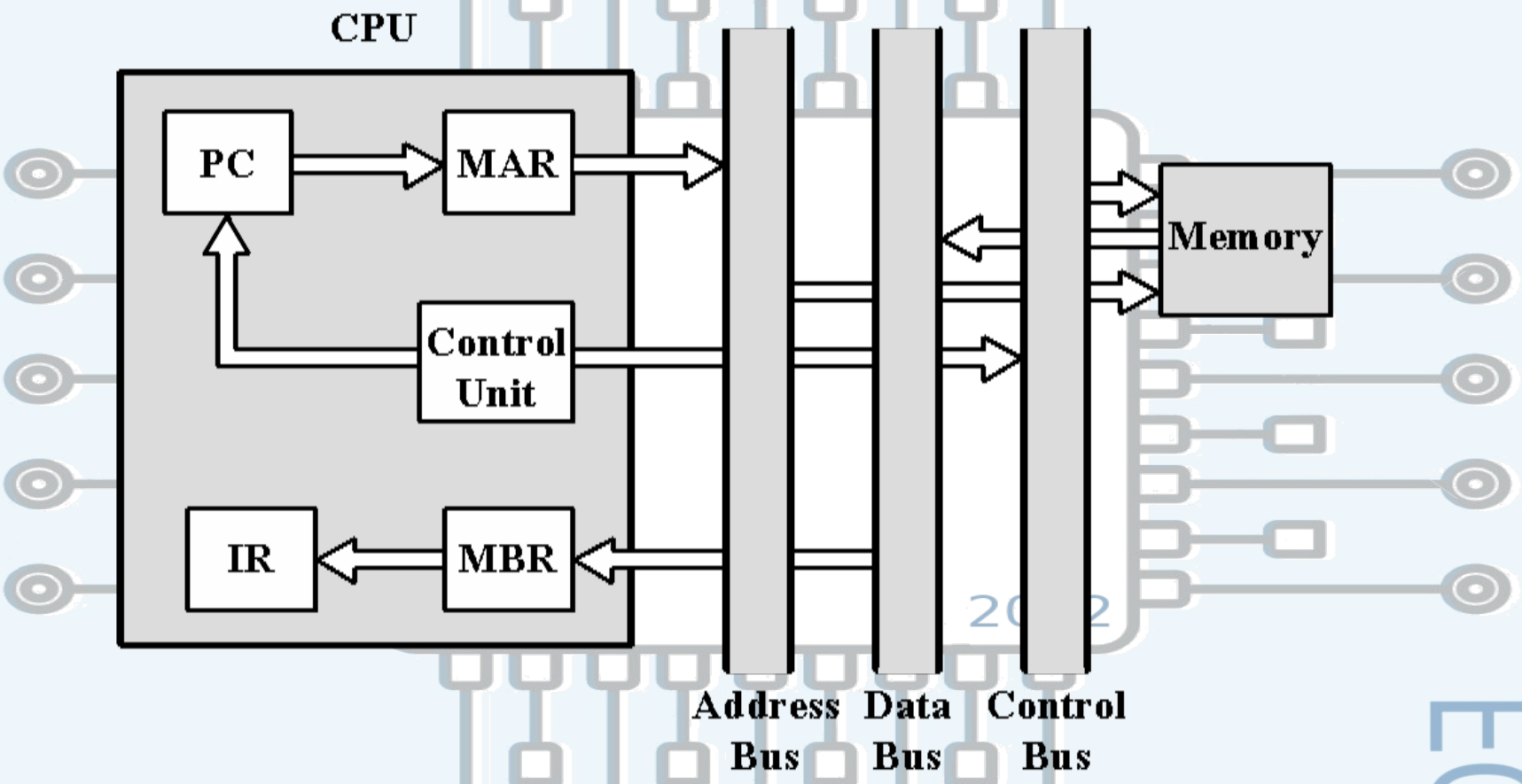


Figure 16.6
Data Flow, Indirect Cycle

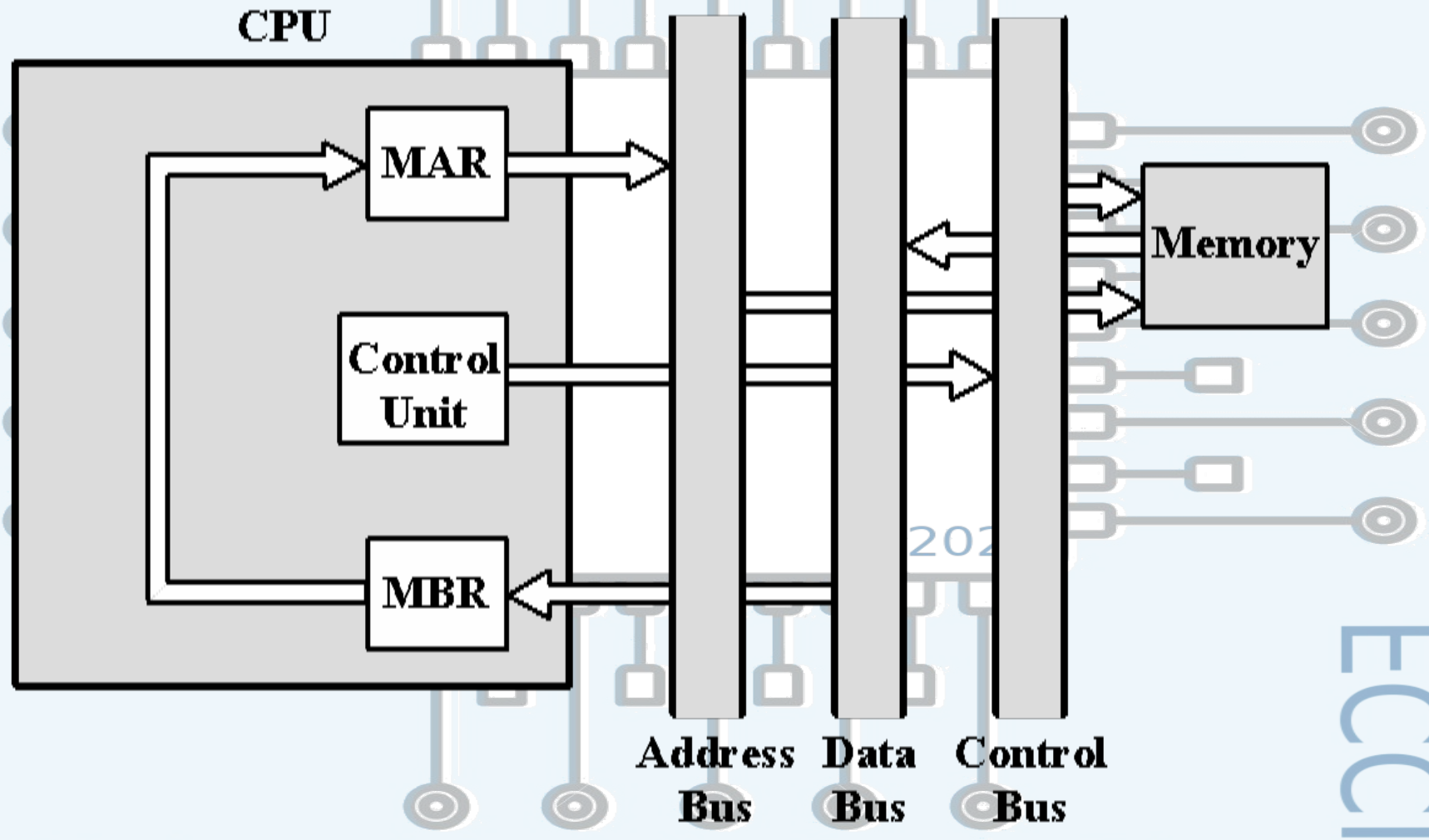
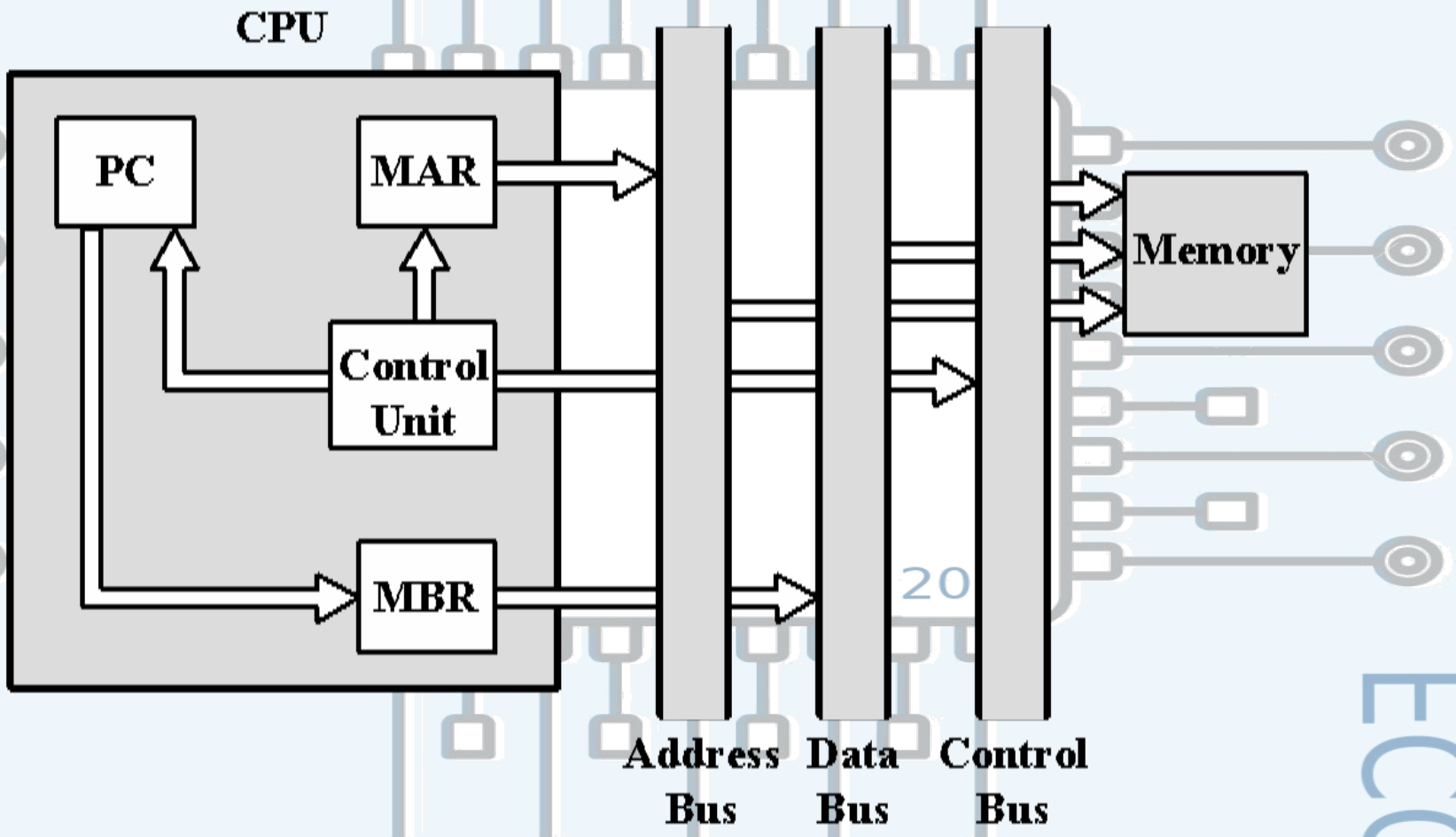


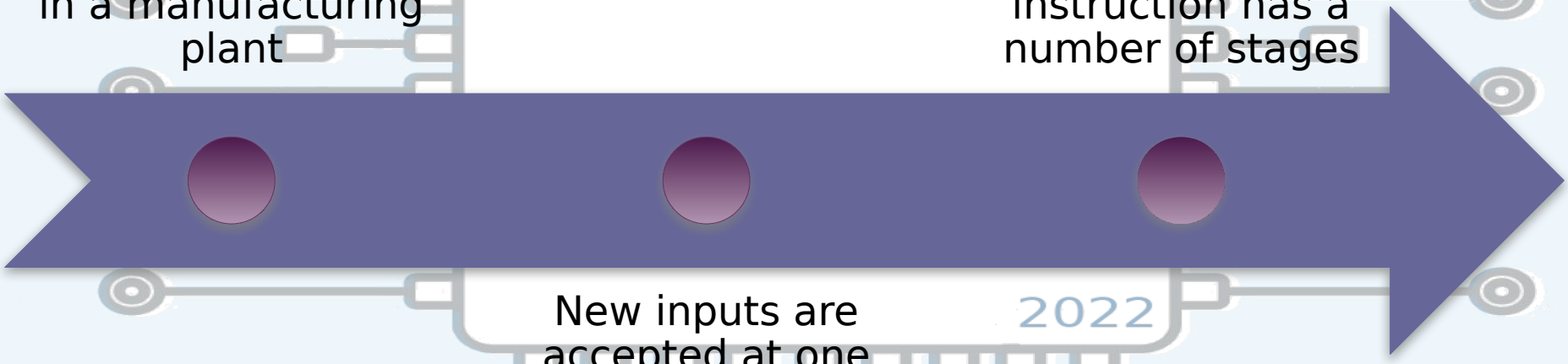
Figure 16.7
Data Flow, Interrupt Cycle



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

Figure 16.8
Two-Stage Instruction Pipeline

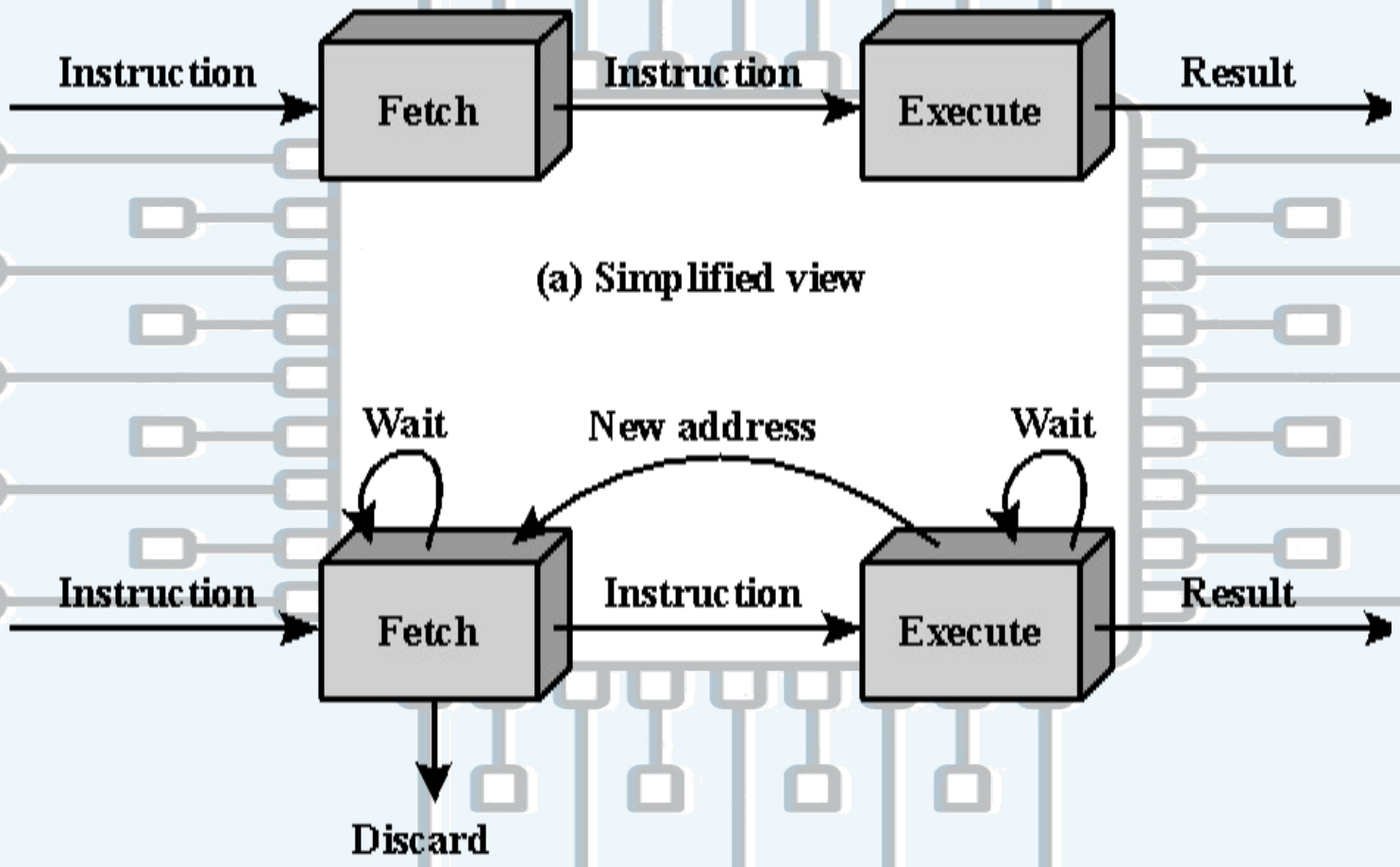
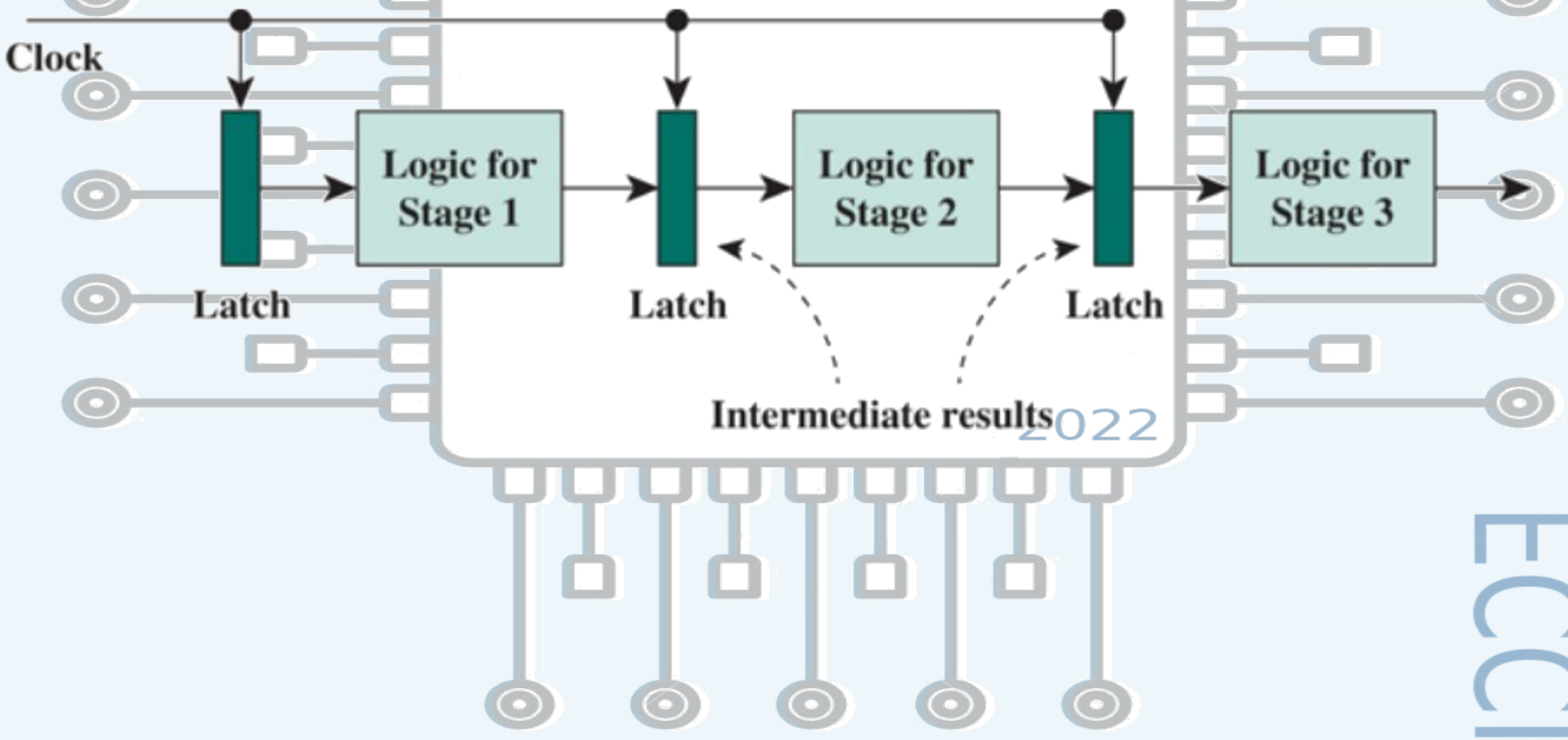


Figure 16.9
Simplified Pipeline Architecture



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 16.10 Timing Diagram for Instruction Pipeline Operation

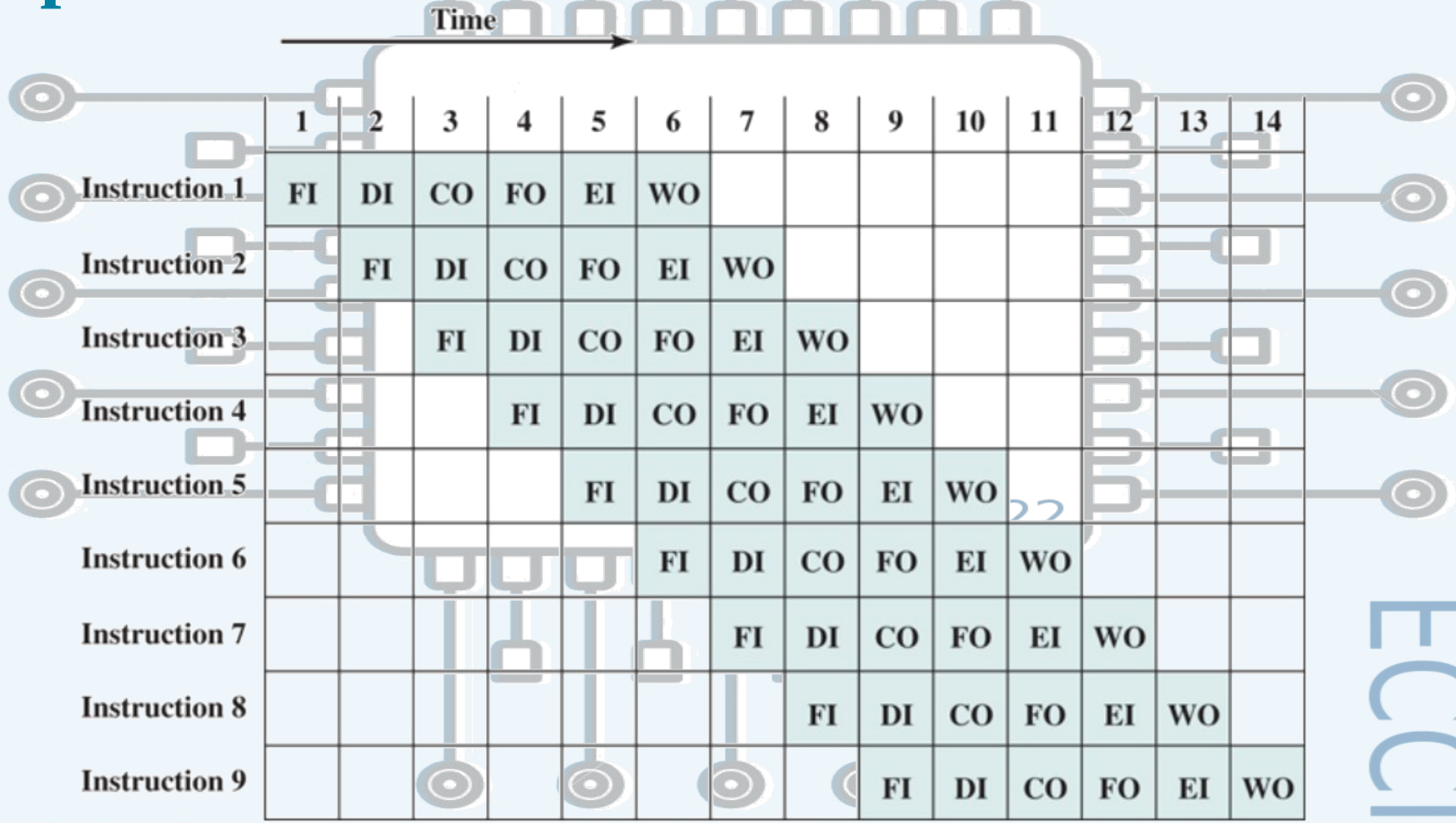


Figure 16.11
The Effect of a Conditional Branch on Instruction Pipeline Operation

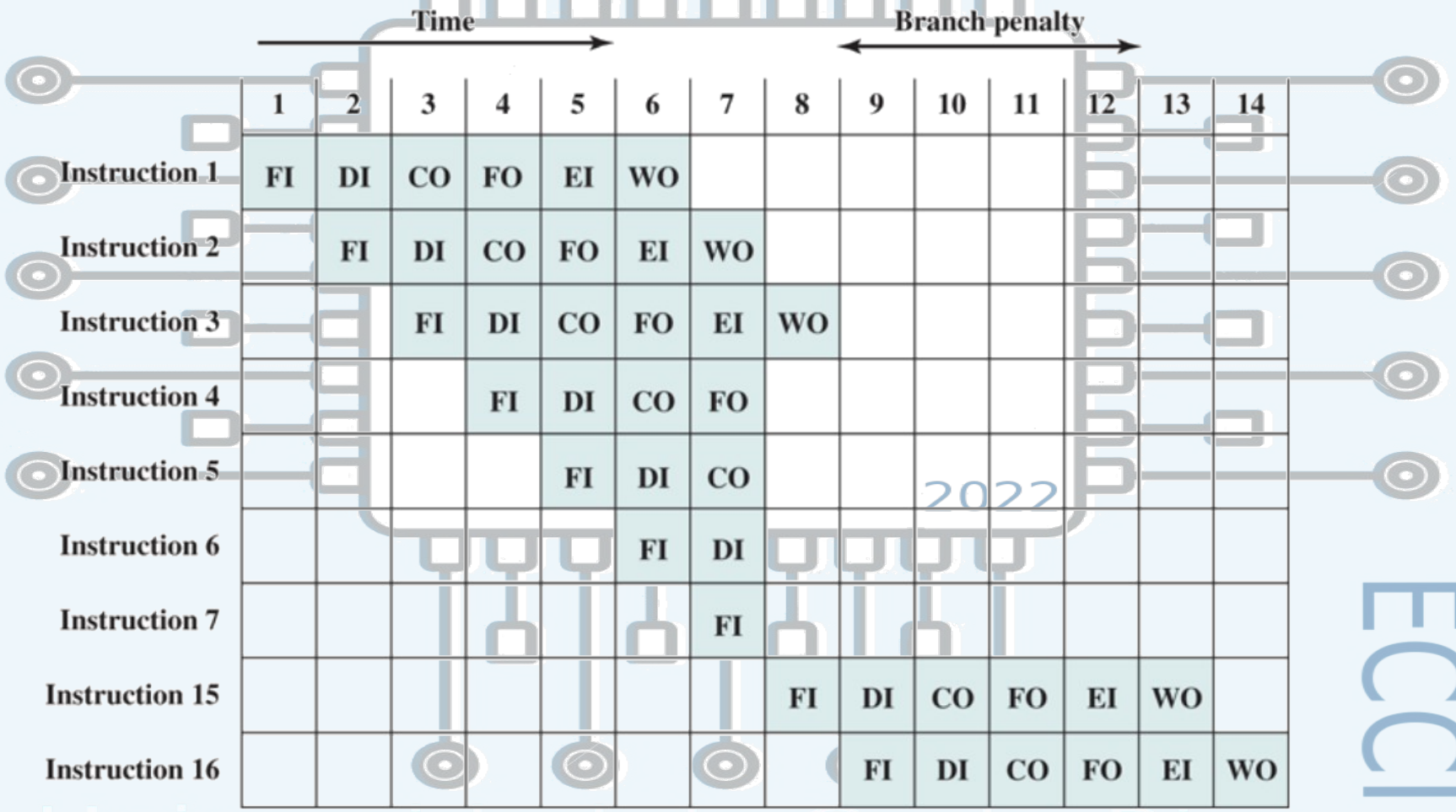


Figure 16.12
Six-Stage CPU Instruction Pipeline

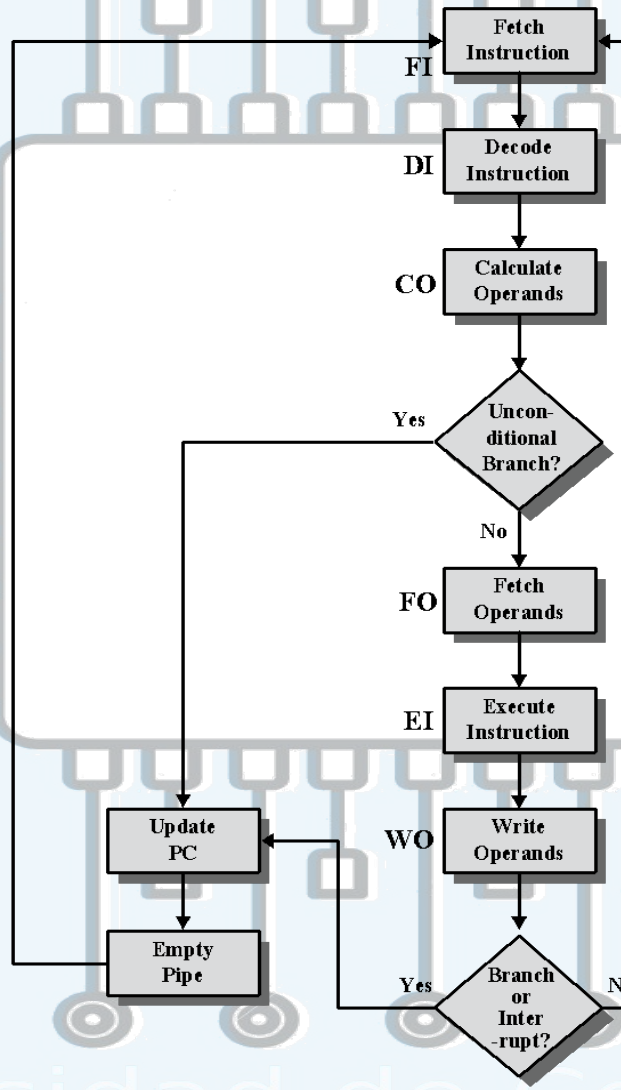


Figure 16.13
An Alternative Pipeline Depiction

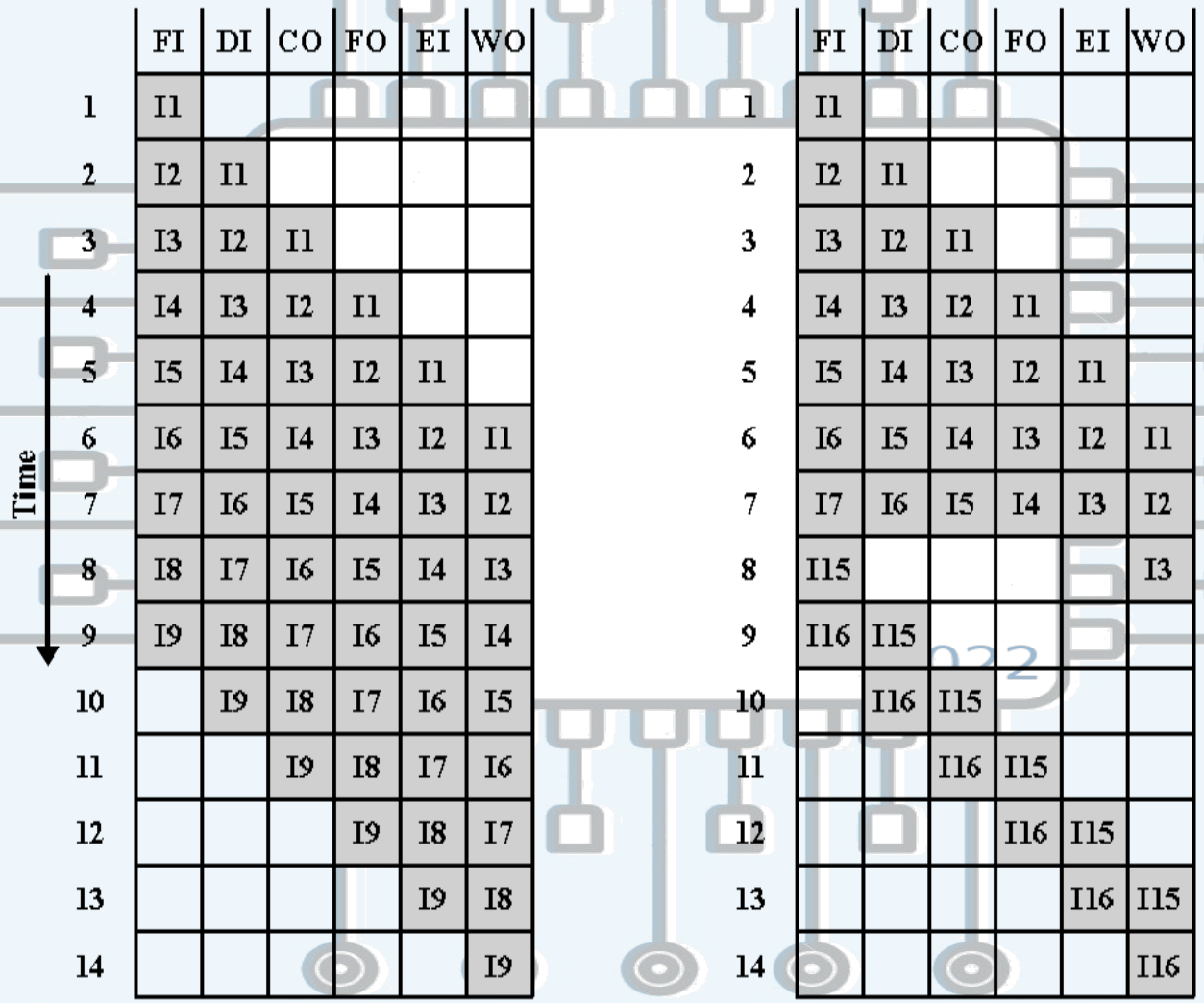
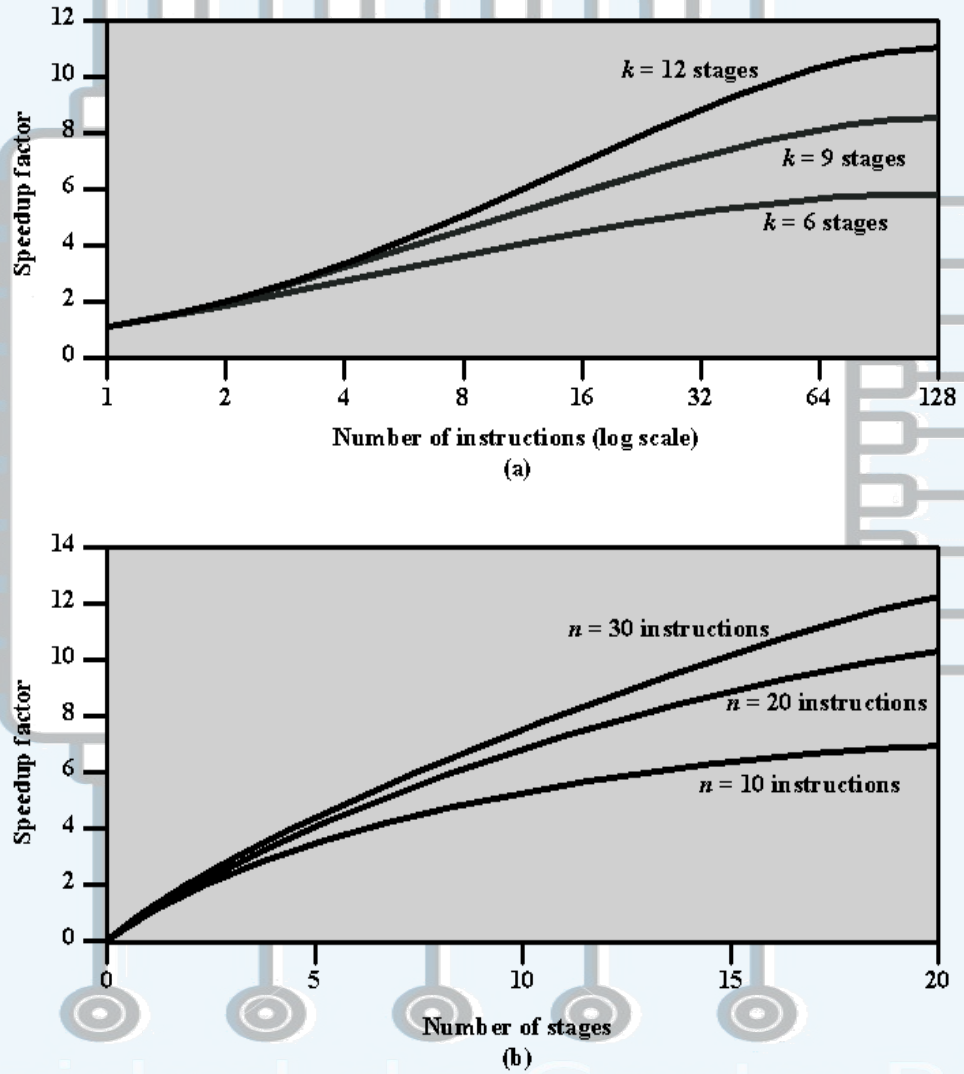


Figure 16.14
Speedup Factors with Instruction Pipelining

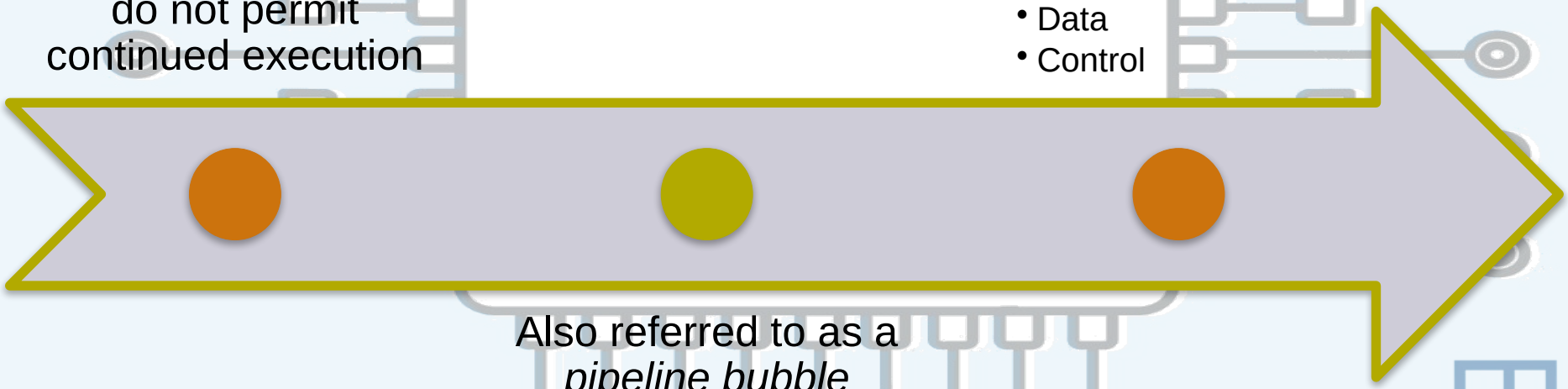


Pipeline Hazards

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

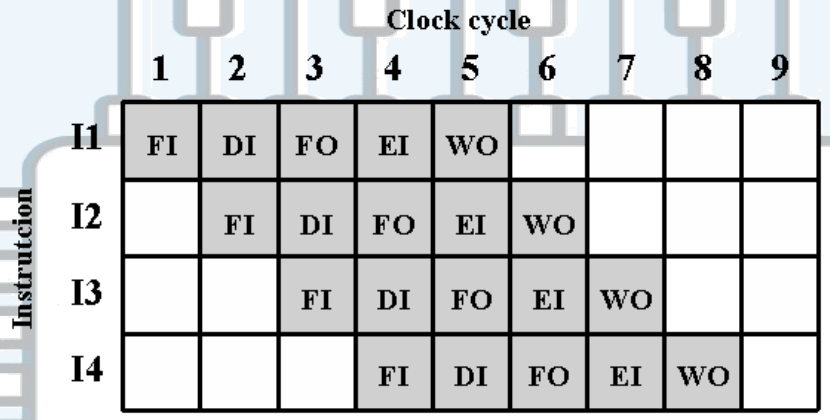
There are three types of hazards:

- Resource
- Data
- Control

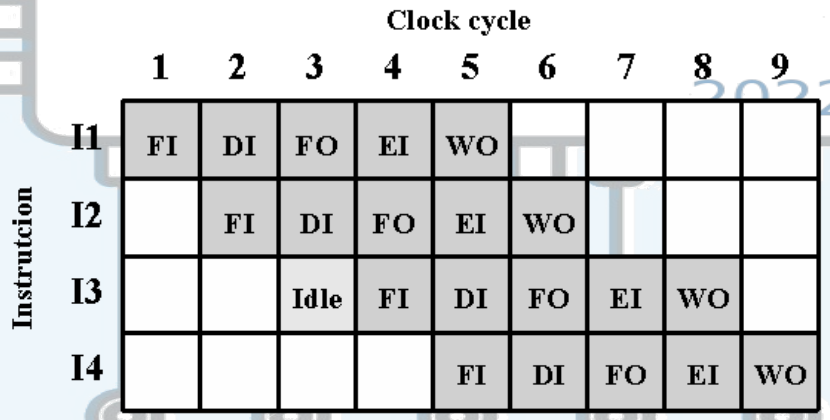


Also referred to as a *pipeline bubble*

Figure 16.15
Example of Resource Hazard

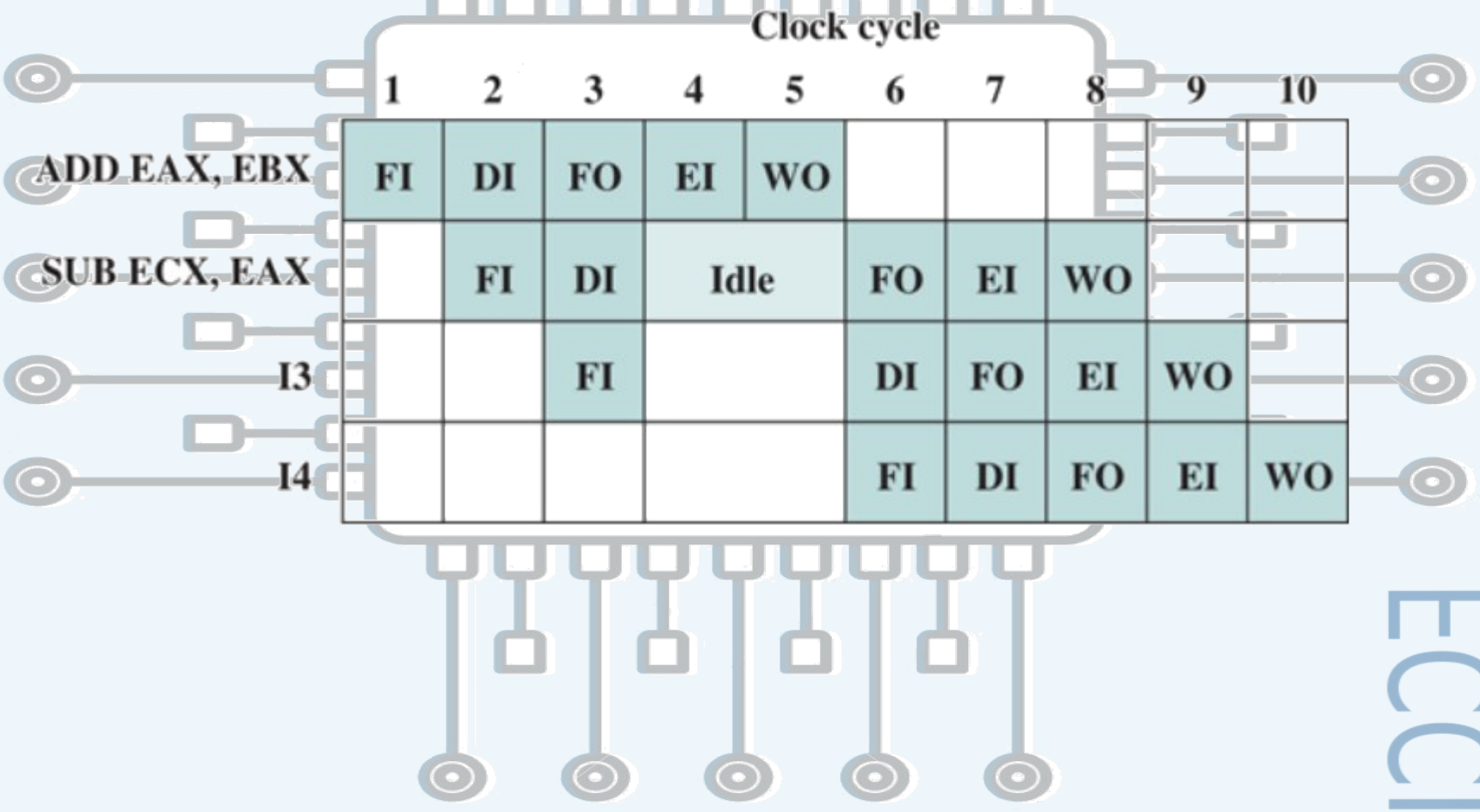


(a) Five-stage pipeline, ideal case



(b) I1 source operand in memory

Figure 16.16
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

- 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

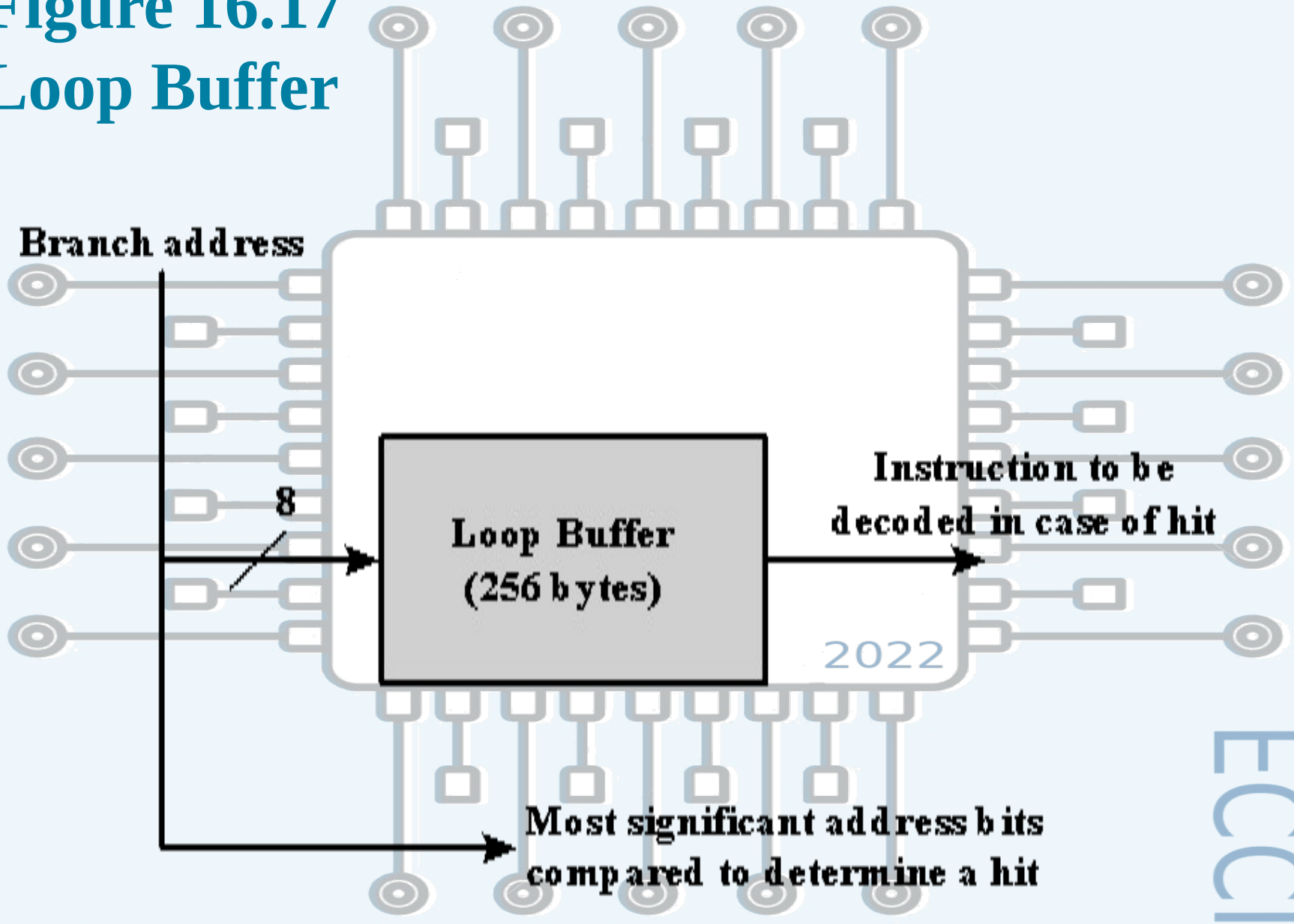
Prefetch Branch Target

- 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

Figure 16.17
Loop Buffer



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

4. Taken/not taken switch
5. Branch history table

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

Figure 16.18
Branch Prediction Flowchart

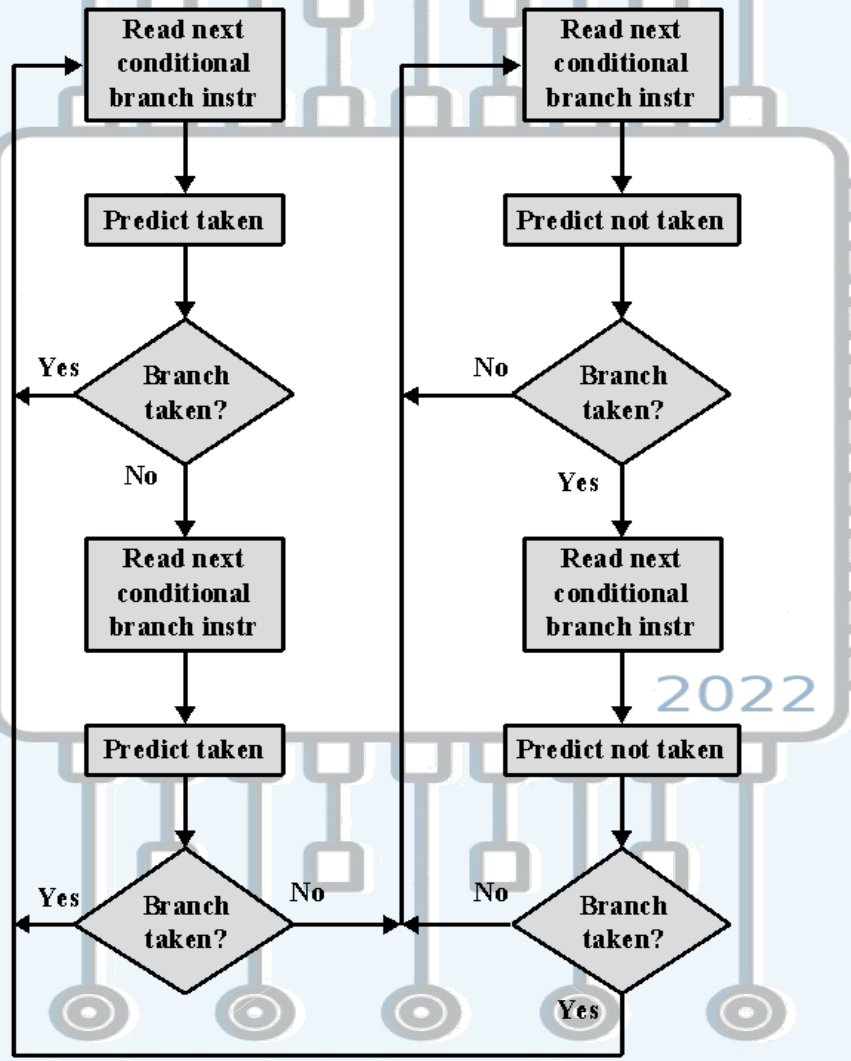


Figure 16.19
Branch Prediction State Diagram

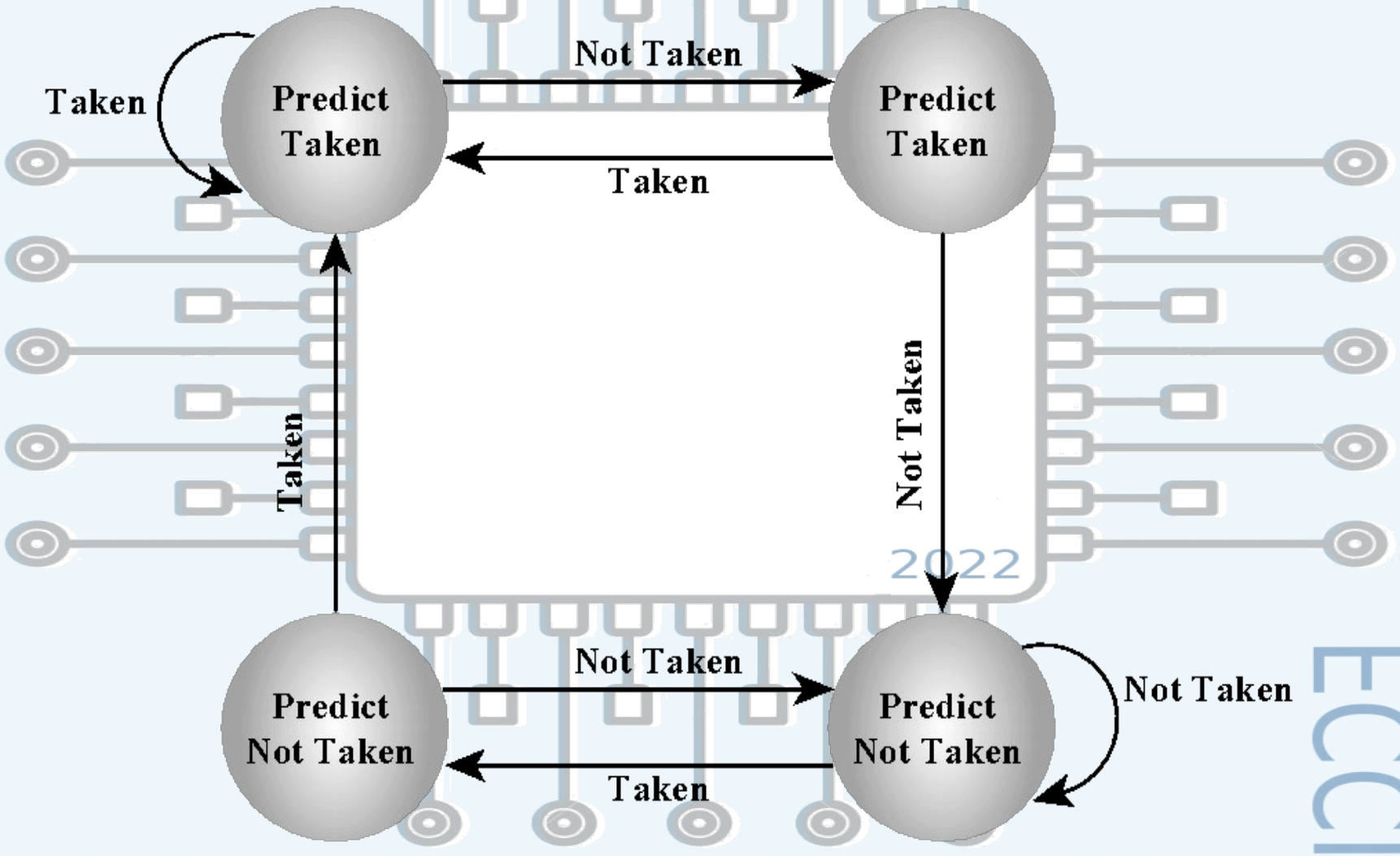
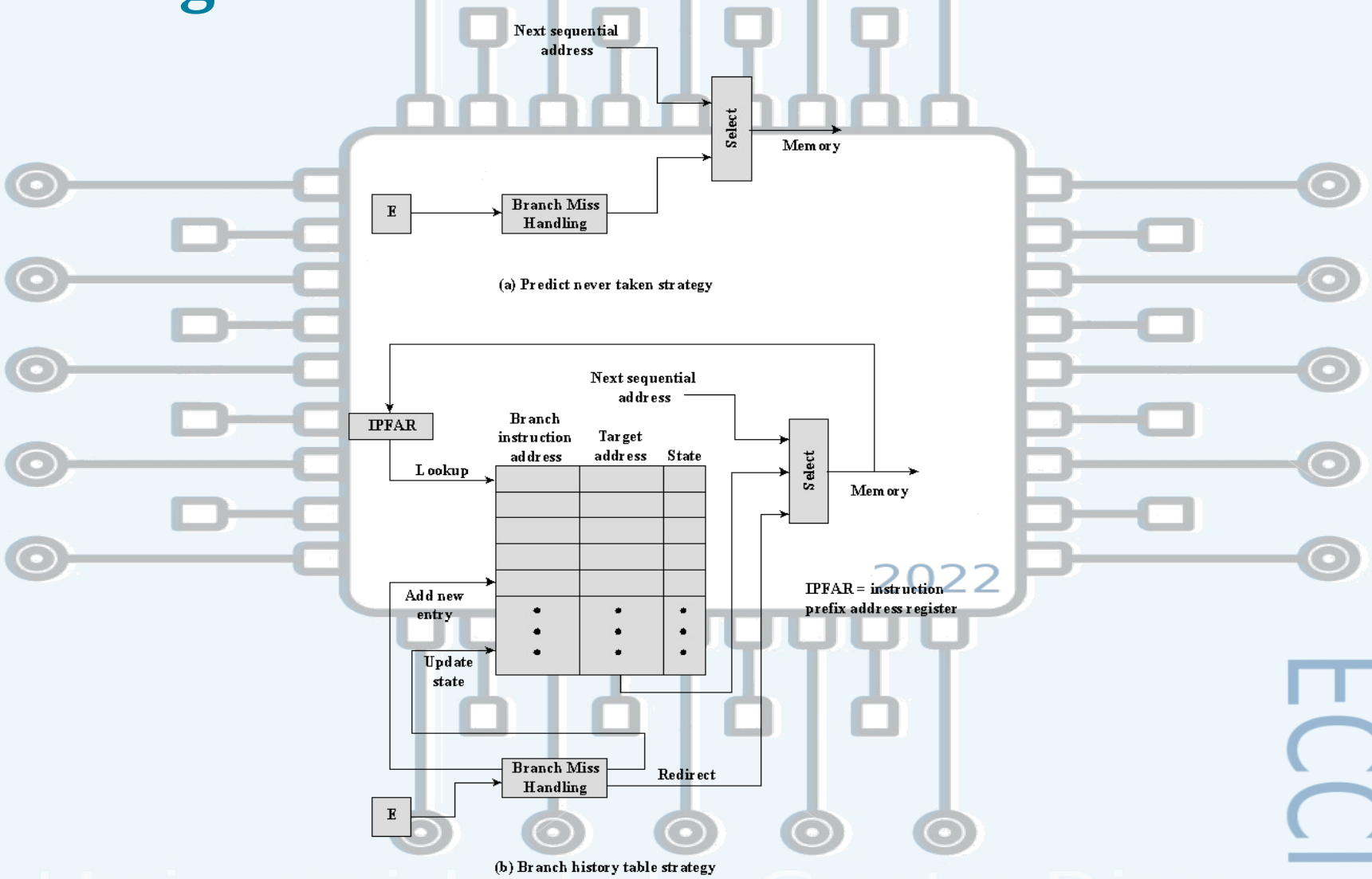


Figure 16.20
Dealing with Branches



Intel 80486 Pipelining

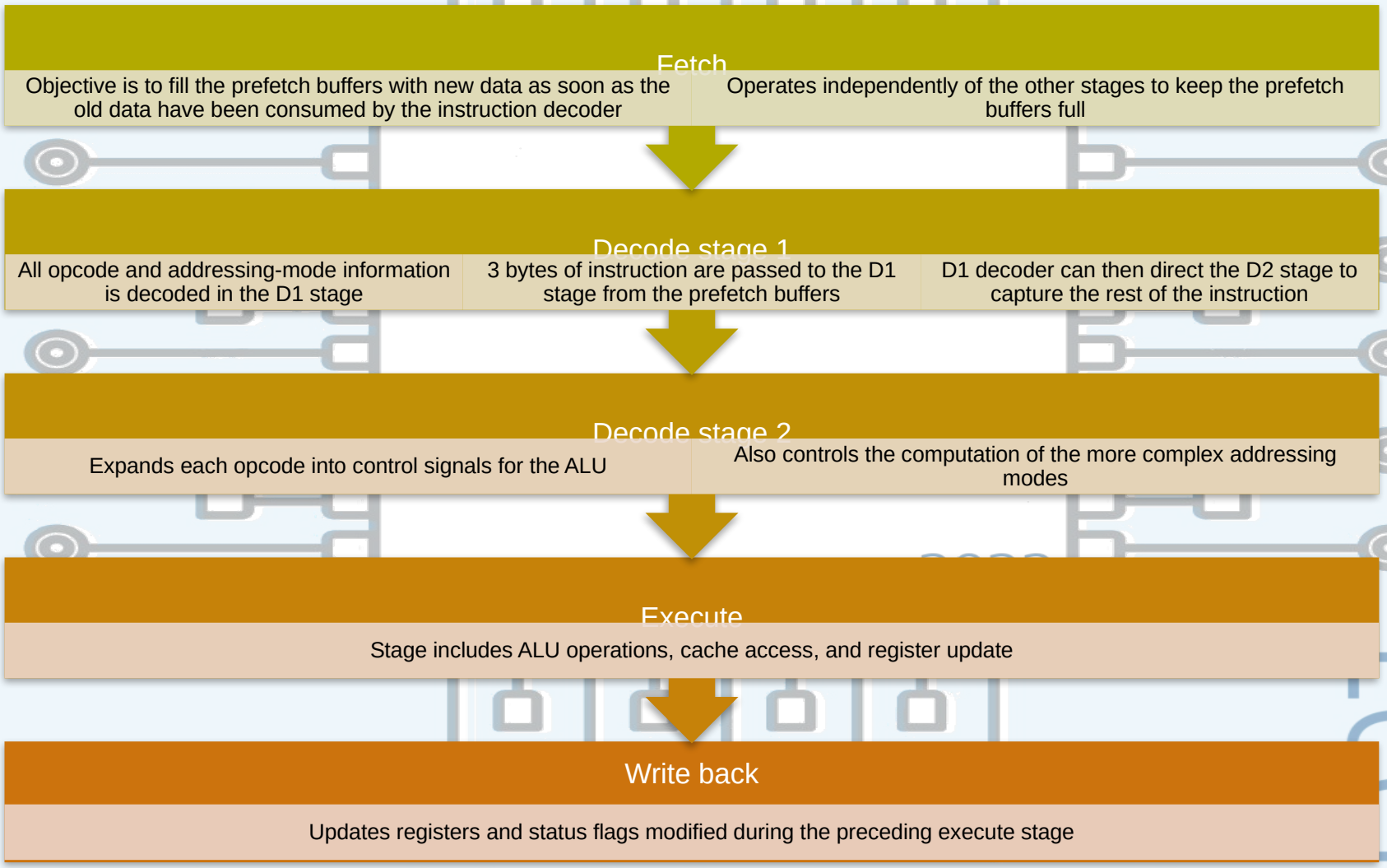
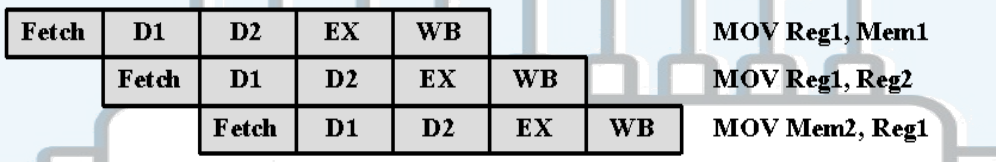
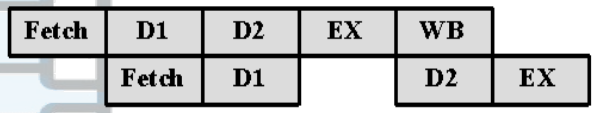


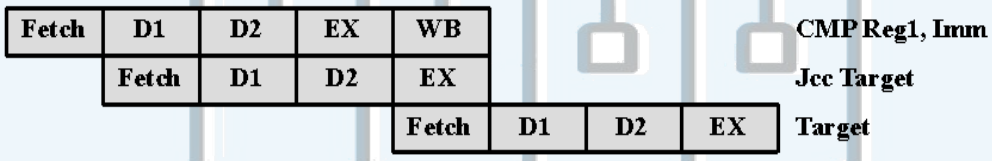
Figure 16.21
80486 Instruction Pipeline Examples



(a) No Data Load Delay in the Pipeline

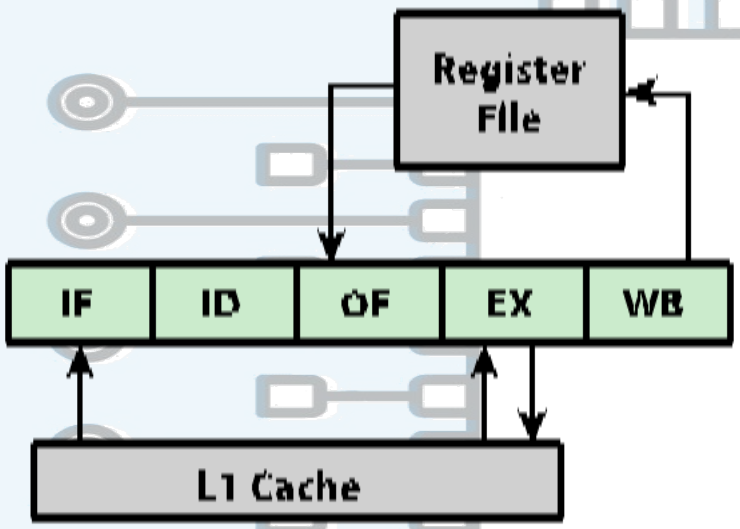


(b) Pointer Load Delay

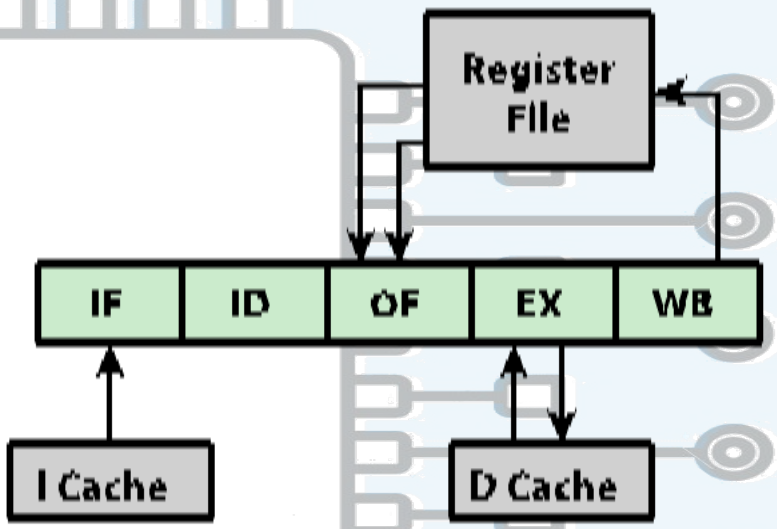


(c) Branch Instruction Timing

Figure 16.22
Approaches to Pipeline Organization



(a) Simple Pipeline Organization



(b) Performance Enhancements

Figure 16.23
Improved Pipeline Organization

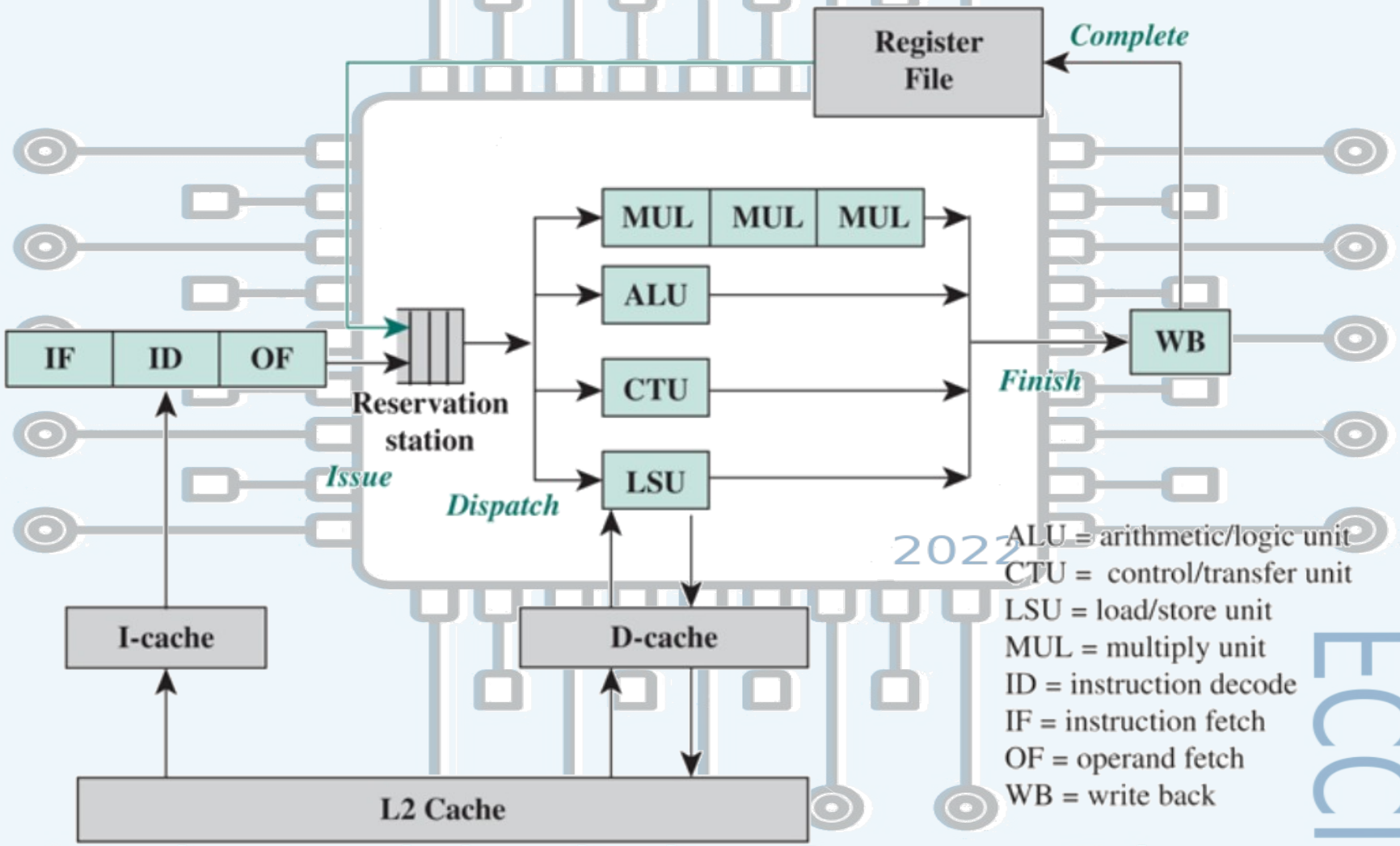


Figure 16.24
Reservation Station Contents

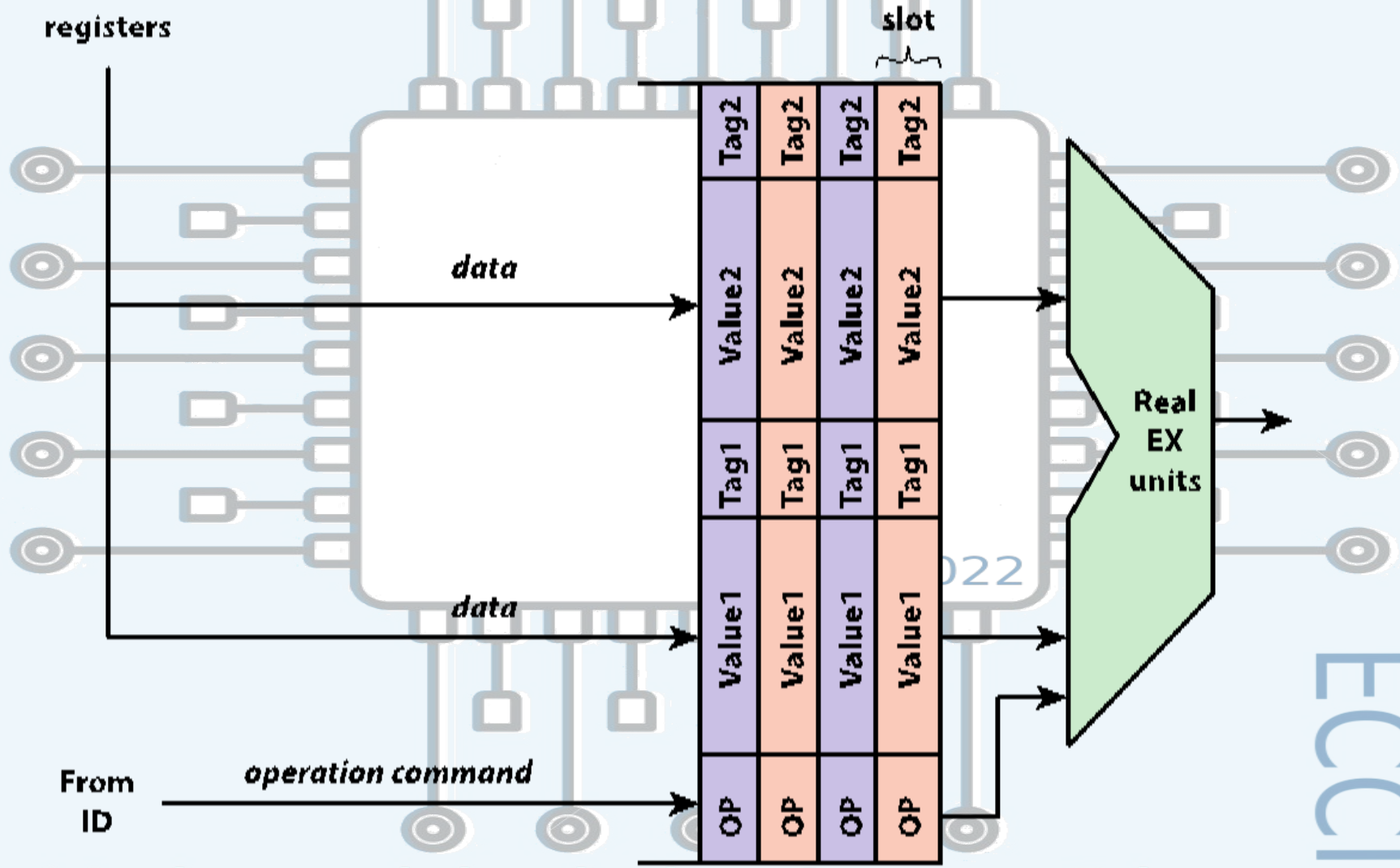


Table 16.2

x86 Processor Registers

(a) Integer Unit in 32-bit Mode

Type	Number	Length (bits)	Purpose
General	8	32	General-purpose user registers
Segment	6	16	Contain segment selectors
EFLAGS	1	32	Status and control bits
Instruction Pointer	1	32	Instruction pointer

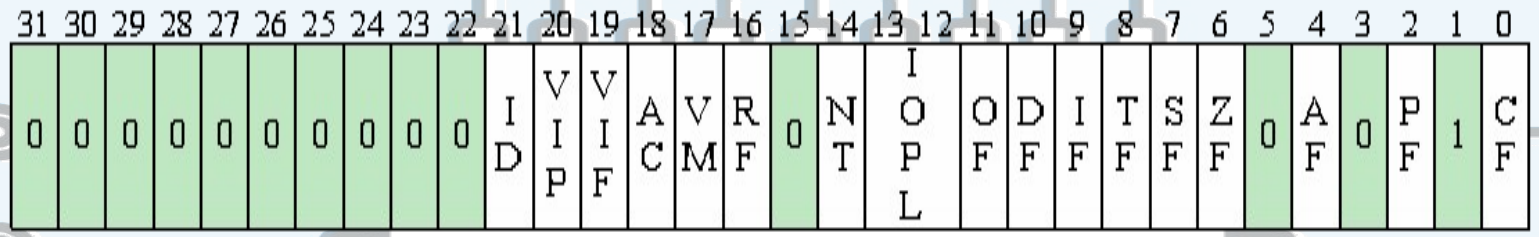
(b) Integer Unit in 64-bit Mode

Type	Number	Length (bits)	Purpose
General	16	32	General-purpose user registers
Segment	6	16	Contain segment selectors
RFLAGS	1	64	Status and control bits
Instruction Pointer	1	64	Instruction pointer

(c) Floating-Point Unit

Type	Number	Length (bits)	Purpose
Numeric	8	80	Hold floating-point numbers
Control	1	16	Control bits
Status	1	16	Status bits
Tag Word	1	16	Specifies contents of numeric registers
Instruction Pointer	1	48	Points to instruction interrupted by exception
Data Pointer	1	48	Points to operand interrupted by exception

Figure 16.25
x86 EFLAGS Register

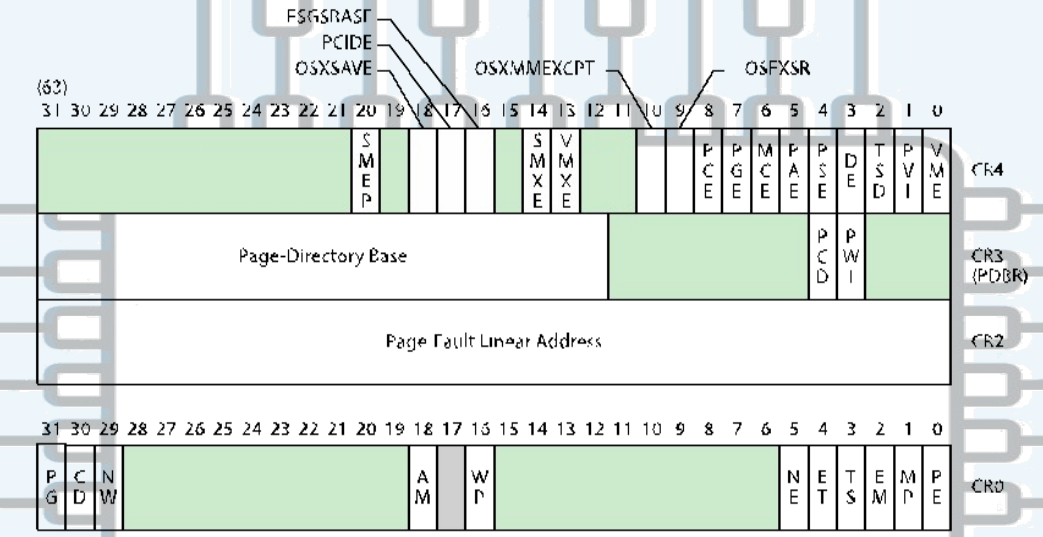


X ID = Identification flag
X VIP = Virtual interrupt pending
X VIF = Virtual interrupt flag
X AC = Alignment check
X VM = Virtual 8086 mode
X RF = Resume flag
X NT = Nested task flag
X IOPL = I/O privilege level
S OF = Overflow flag

C DF = Direction flag
X IF = Interrupt enable flag
X TF = Trap flag
S SF = Sign flag
S ZF = Zero flag
S AF = Auxiliary carry flag
S PF = Parity flag
S CF = Carry flag

S Indicates a Status Flag
C Indicates a Control Flag
X Indicates a System Flag
Shaded bits are reserved

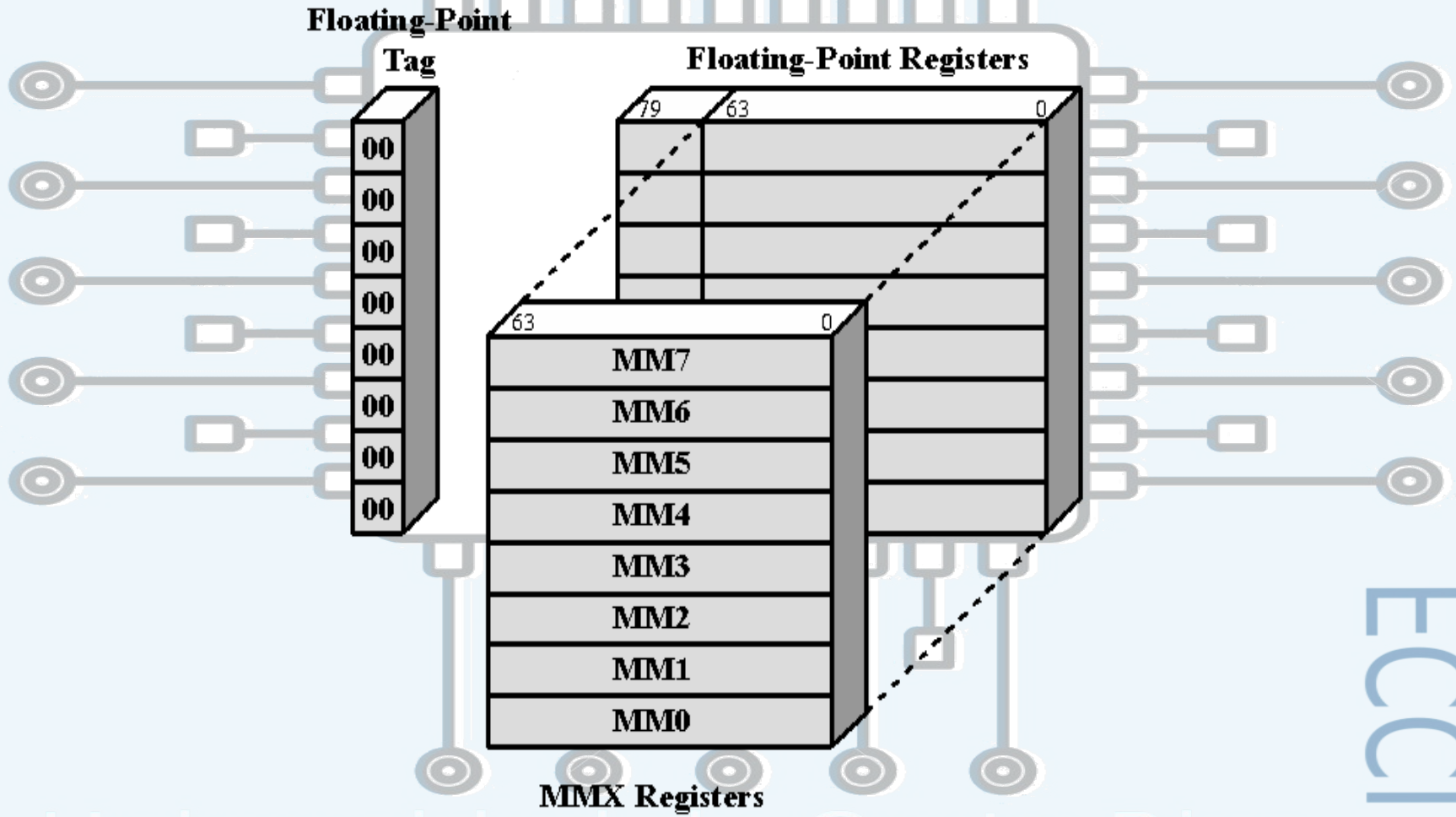
Figure 16.26
x86 Control Registers



shaded area indicates reserved bits

OSXSAVE	=	XSAVE enable bit	VME	=	Virtual 8086 Mode Extensions
PCIDE	–	Enables process-context identifiers	PCD	–	Page-level Cache Disable
FSGSBASE	=	Enables segment base instructions	PWT	=	Page-level Writes Transparent
SMXE	=	Enable Safer mode extensions	PG	=	Paging
VMXE	=	Enable virtual machine extensions	CD	=	Cache Disable
OSXMMEXCPT	=	Support unmasked SIMD FP exceptions	NW	=	Not Write Through
OSFXSR	=	Support FXSAVE, FXSTOR	AM	=	Alignment Mask
PCE	=	Performance Counter Enable	WP	=	Write Protect
PGE	=	Page Global Enable	NE	=	Numeric Error
MCE	=	Machine Check Enable	ET	=	Extension Type
PAE	–	Physical Address Extension	TS	–	Task Switched
PSE	=	Page Size Extensions	EM	=	Emulation
DE	=	Debug Extensions	MP	=	Monitor Coprocessor
TSD	=	Time Stamp Disable	PE	=	Protection Enable
PVI	=	Protected Mode Virtual Interrupt			

Figure 16.27 Mapping of MMX Registers to Floating-Point Registers



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

Table 16.3 x86 Exception and Interrupt Vector Table

Vector Number	Description
0	Divide error; division overflow or division by zero
1	Debug exception; includes various faults and traps related to debugging
2	NMI pin interrupt; signal on NMI pin
3	Breakpoint; caused by INT 3 instruction, which is a 1-byte instruction useful for debugging
4	INTO-detected overflow; occurs when the processor executes INTO with the OF flag set
5	BOUND range exceeded; the BOUND instruction compares a register with boundaries stored in memory and generates an interrupt if the contents of the register is out of bounds
6	Undefined opcode
7	Device not available; attempt to use ESC or WAIT instruction fails due to lack of external device
8	Double fault; two interrupts occur during the same instruction and cannot be handled serially
9	Reserved
10	Invalid task state segment; segment describing a requested task is not initialized or not valid
11	Segment not present; required segment not present
12	Stack fault; limit of stack segment exceeded or stack segment not present
13	General protection; protection violation that does not cause another exception (e.g., writing to a read-only segment)
14	Page fault
15	Reserved
16	Floating-point error; generated by a floating-point arithmetic instruction
17	Alignment check; access to a word stored at an odd byte address or a doubleword stored at an address not a multiple of 4
18	Machine check; model specific
19–31	Reserved
32–255	User interrupt vectors; provided when INTR signal is activated

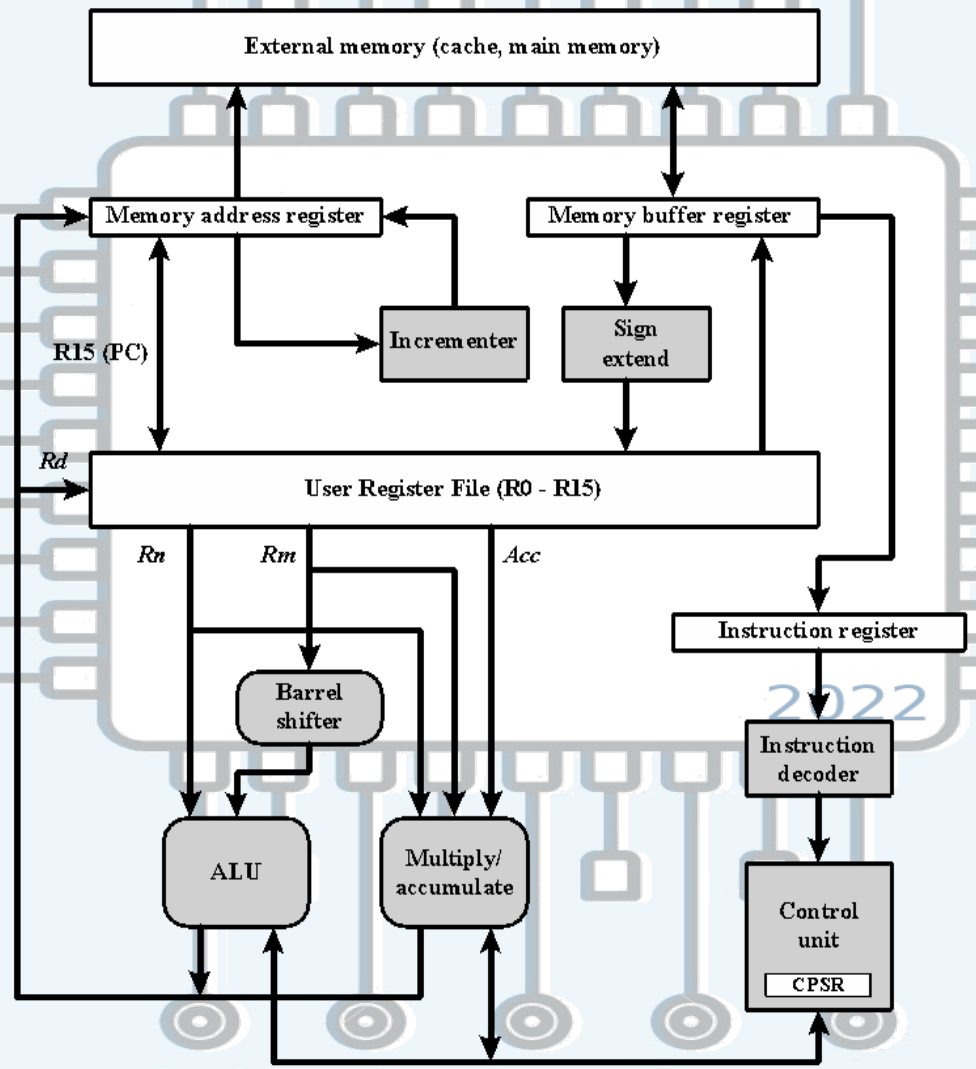
Unshaded: exceptions
Shaded: interrupts

The ARM Processor

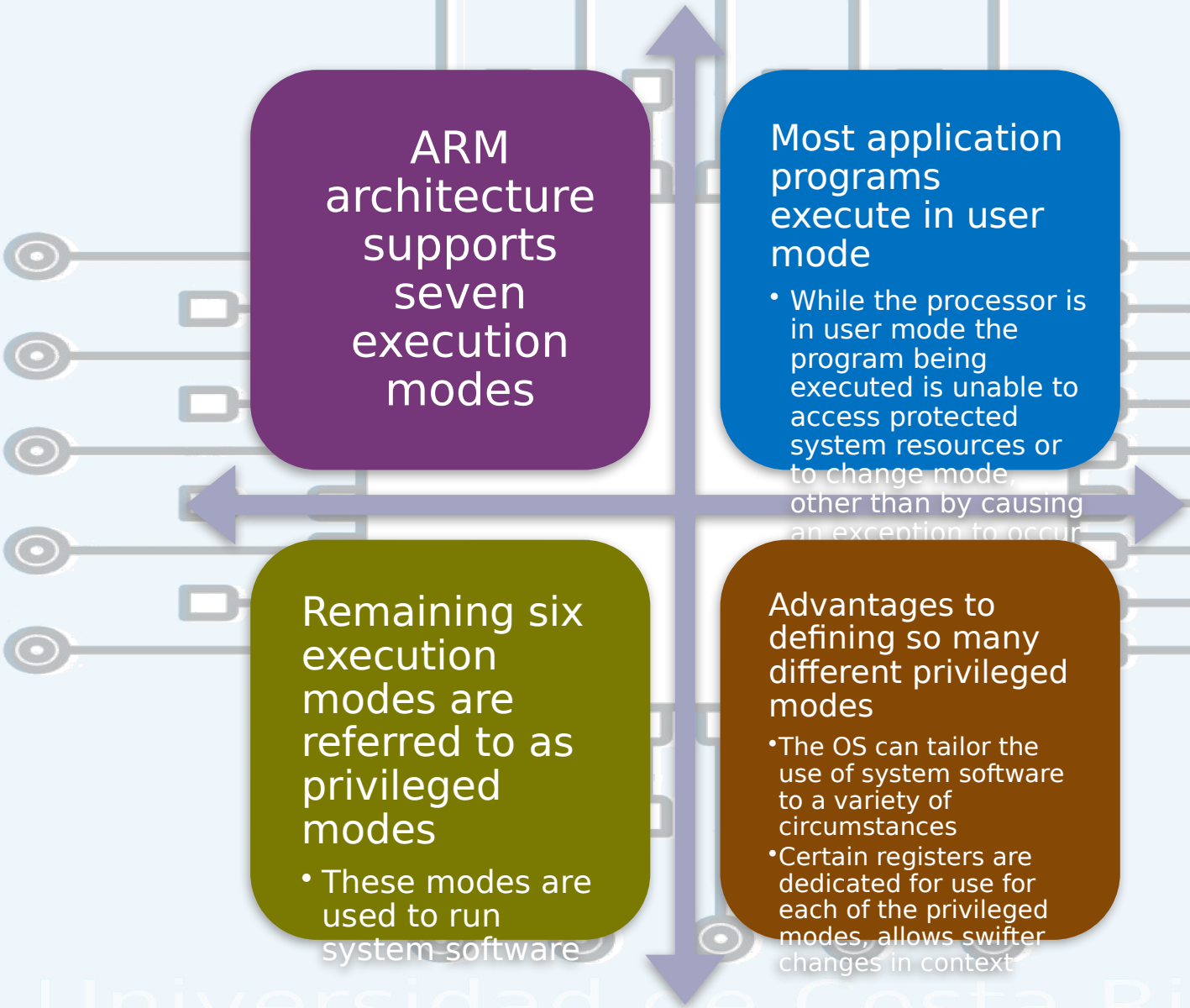
ARM is primarily a RISC system with the following attributes:

- 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

Figure 16.28
Simplified ARM Organization



Processor Modes



Exception Modes

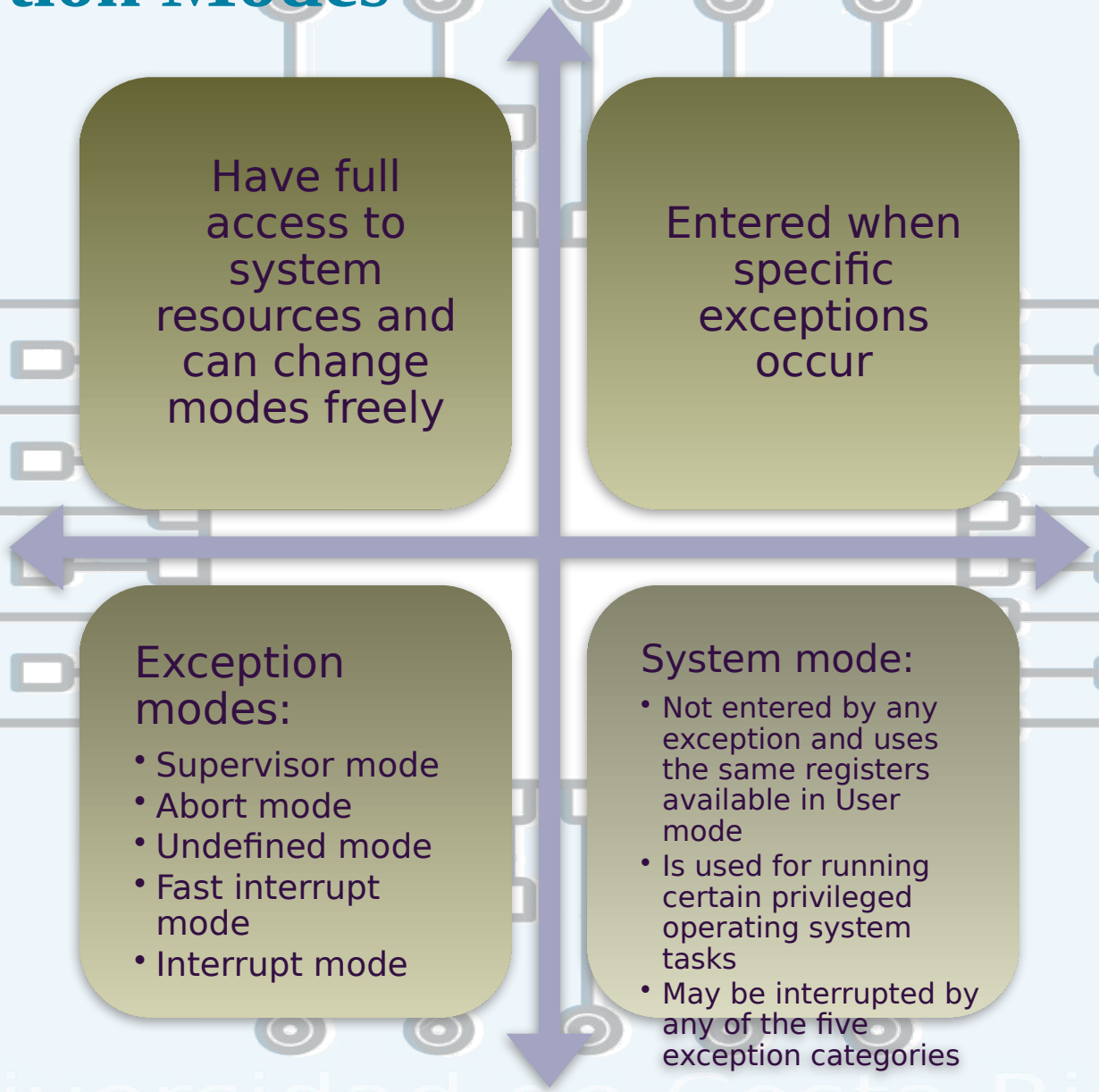


Figure 16.29
ARM Register Organization

Modes						
	Privileged modes					
		Exception modes				
User	System	Supervisor	Abort	Undefined	Interrupt	Fast Interrupt
R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	R8_fiq
R9	R9	R9	R9	R9	R9	R9_fiq
R10	R10	R10	R10	R10	R10	R10_fiq
R11	R11	R11	R11	R11	R11	R11_fiq
R12	R12	R12	R12	R12	R12	R12_fiq
R13 (SP)	R13 (SP)	R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
R14 (LR)	R14 (LR)	R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

Shading indicates that the normal register used by User or System mode has been replaced by an alternative register specific to the exception mode.

SP = stack pointer
LR = link register
PC = program counter

CPSR = current program status register
SPSR = saved program status register

Figure 16.30
Format of ARM CPSR and SPSR

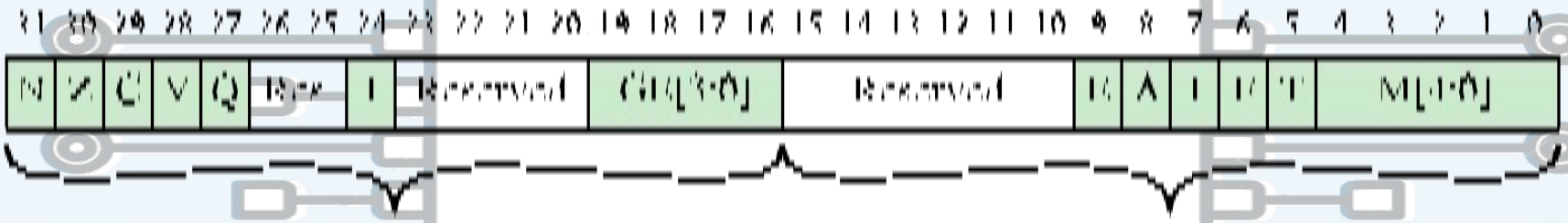


Table 16.4 ARM Interrupt Vector

Exception type	Mode	Normal entry address	Description
Reset	Supervisor	0x00000000	Occurs when the system is initialized.
Data abort	Abort	0x00000010	Occurs when an invalid memory address has been accessed, such as if there is no physical memory for an address or the correct access permission is lacking.
FIQ (fast interrupt)	FIQ	0x0000001C	Occurs when an external device asserts the FIQ pin on the processor. An interrupt cannot be interrupted except by an FIQ. FIQ is designed to support a data transfer or channel process, and has sufficient private registers to remove the need for register saving in such applications, therefore minimizing the overhead of context switching. A fast interrupt cannot be interrupted.
IRQ (interrupt)	IRQ	0x00000018	Occurs when an external device asserts the IRQ pin on the processor. An interrupt cannot be interrupted except by an FIQ.
Prefetch abort	Abort	0x0000000C	Occurs when an attempt to fetch an instruction results in a memory fault. The exception is raised when the instruction enters the execute stage of the pipeline.
Undefined instructions	Undefined	0x00000004	Occurs when an instruction not in the instruction set reaches the execute stage of the pipeline.
Software interrupt	Supervisor	0x00000008	Generally used to allow user mode programs to call the OS. The user program executes a SWI instruction with an argument that identifies the function the user wishes to perform.

Summary

Chapter 16

- Processor organization
- Register organization
  - User-visible registers
  - Control and status registers
- Instruction cycle
  - The indirect cycle
  - Data flow
- The x86 processor family
  - Register organization
  - Interrupt processing

•Processor Structure and Function

- Instruction pipelining
  - Pipelining strategy
  - Pipeline performance
  - Pipeline hazards
  - Dealing with branches
  - Intel 80486 pipelining
- The Arm processor
  - Processor organization
  - Processor modes
  - Register organization
  - Interrupt processing