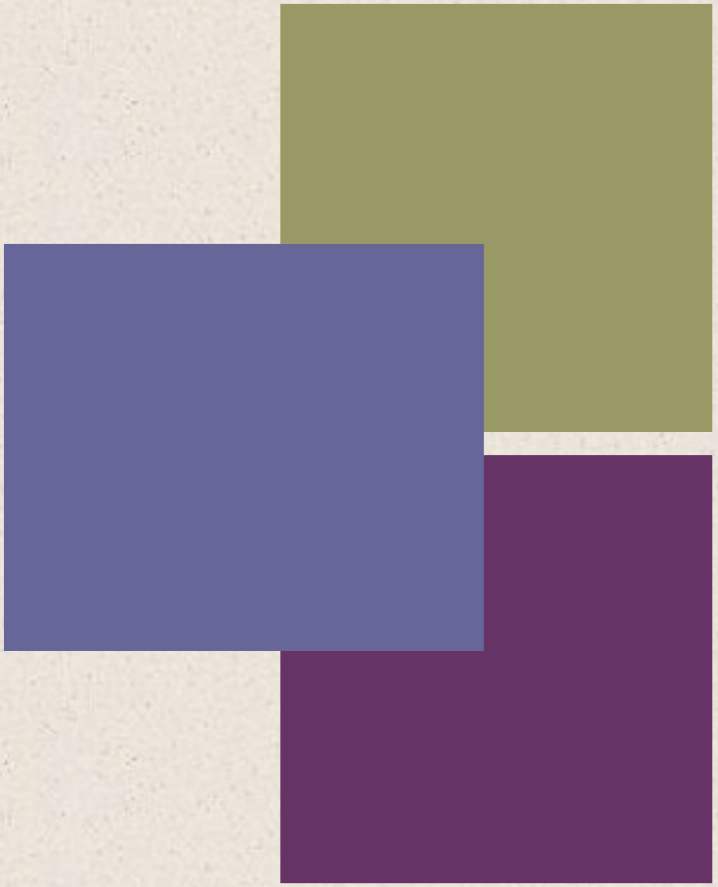
William Stallings
Computer Organization and Architecture
10th Edition

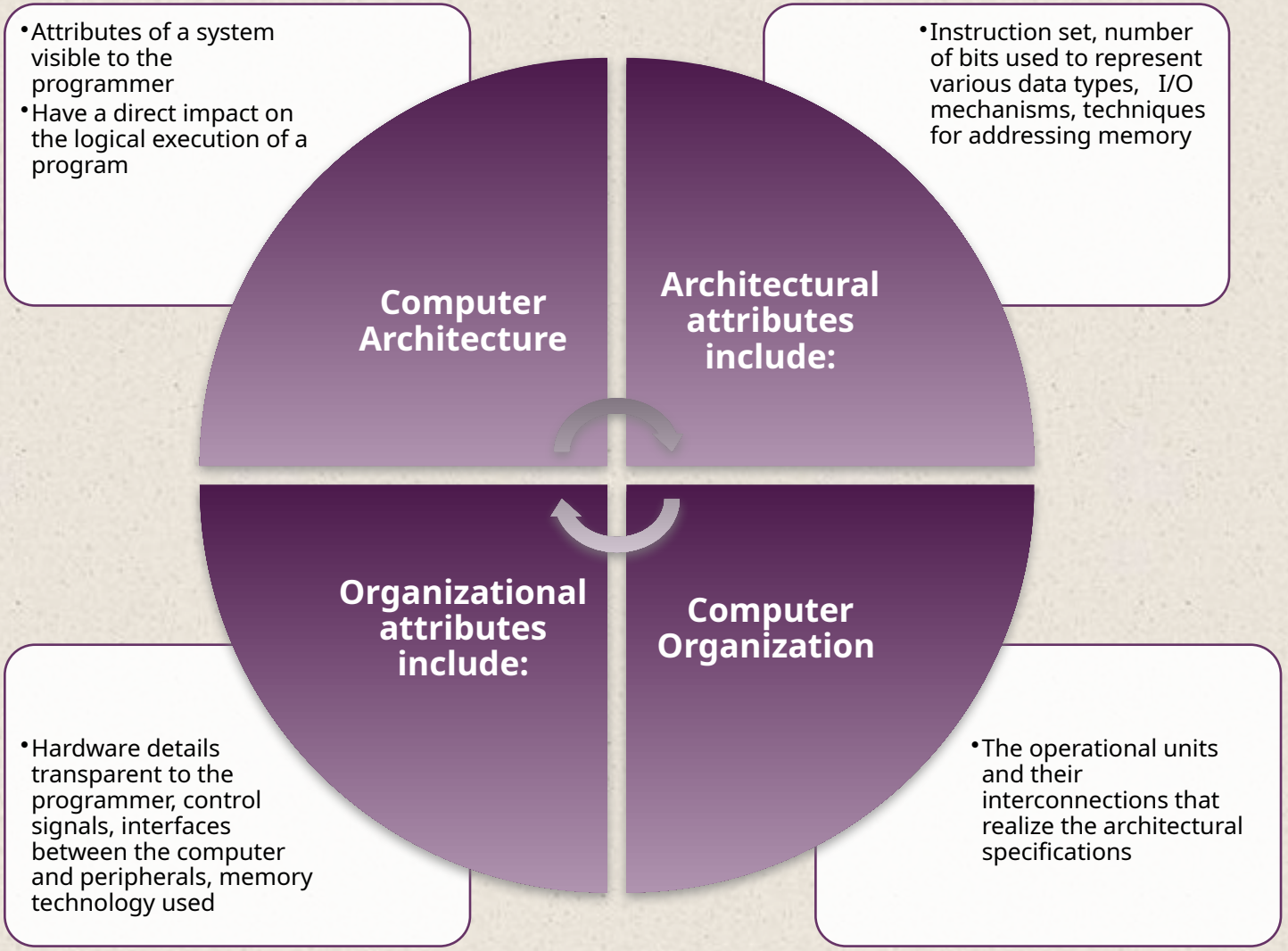


Chapter 1

Basic Concepts and Computer Evolution

Computer Architecture

Computer Organization



IBM System 370 Architecture

- IBM System/370 architecture
  - Was introduced in 1970
  - Included a number of models
  - Could upgrade to a more expensive, faster model without having to abandon original software
  - New models are introduced with improved technology, but retain the same architecture so that the customer's software investment is protected
  - Architecture has survived to this day as the architecture of IBM's mainframe product line

Structure and Function

- Hierarchical system
  - Set of interrelated subsystems
- Hierarchical nature of complex systems is essential to both their design and their description
- Designer need only deal with a particular level of the system at a time
  - Concerned with structure and function at each level

- Structure
  - The way in which components relate to each other
- Function
  - The operation of individual components as part of the structure

Function

- There are four basic functions that a computer can perform:
  - Data processing
    - Data may take a wide variety of forms and the range of processing requirements is broad
  - Data storage
    - Short-term
    - Long-term
  - Data movement
    - Input-output (I/O) - when data are received from or delivered to a device (peripheral) that is directly connected to the computer
    - Data communications – when data are moved over longer distances, to or from a remote device
  - Control
    - A control unit manages the computer's resources and orchestrates the performance of its functional parts in response to instructions

Structure

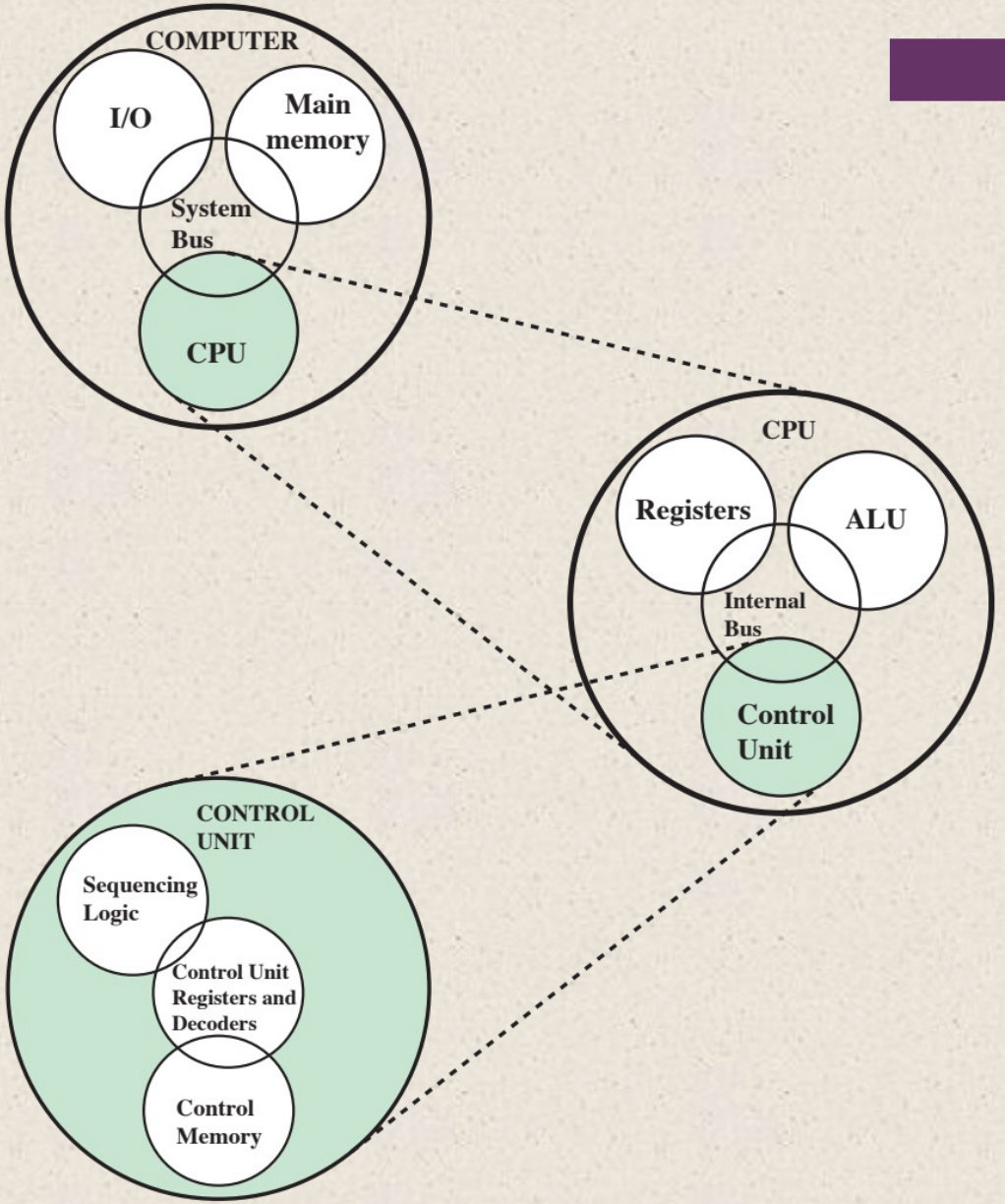
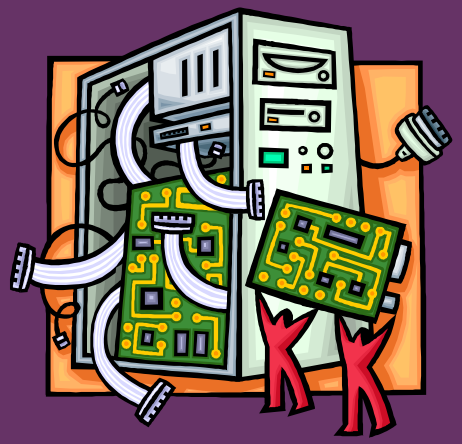


Figure 1.1 A Top-Down View of a Computer

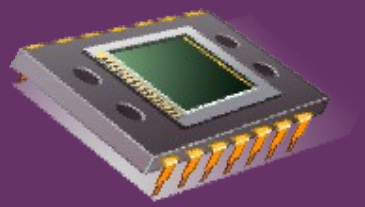
There are four main structural components of the computer:

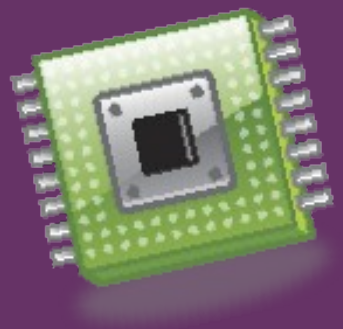


- CPU – controls the operation of the computer and performs its data processing functions
- Main Memory – stores data
- I/O – moves data between the computer and its external environment
- System Interconnection – some mechanism that provides for communication among CPU, main memory, and I/O

CPU

Major structural components:





- Control Unit
  - Controls the operation of the CPU and hence the computer
- Arithmetic and Logic Unit (ALU)
  - Performs the computer's data processing function
- Registers
  - Provide storage internal to the CPU
- CPU Interconnection
  - Some mechanism that provides for communication among the control unit, ALU, and registers

Multicore Computer Structure

- Central processing unit (CPU)
  - Portion of the computer that fetches and executes instructions
  - Consists of an ALU, a control unit, and registers
  - Referred to as a processor in a system with a single processing unit
- Core
  - An individual processing unit on a processor chip
  - May be equivalent in functionality to a CPU on a single-CPU system
  - Specialized processing units are also referred to as cores
- Processor
  - A physical piece of silicon containing one or more cores
  - Is the computer component that interprets and executes instructions
  - Referred to as a *multicore processor* if it contains multiple cores

Cache Memory

- Multiple layers of memory between the processor and main memory
- Is smaller and faster than main memory
- Used to speed up memory access by placing in the cache data from main memory that is likely to be used in the near future
- A greater performance improvement may be obtained by using multiple levels of cache, with level 1 (L1) closest to the core and additional levels (L2, L3, etc.) progressively farther from the core

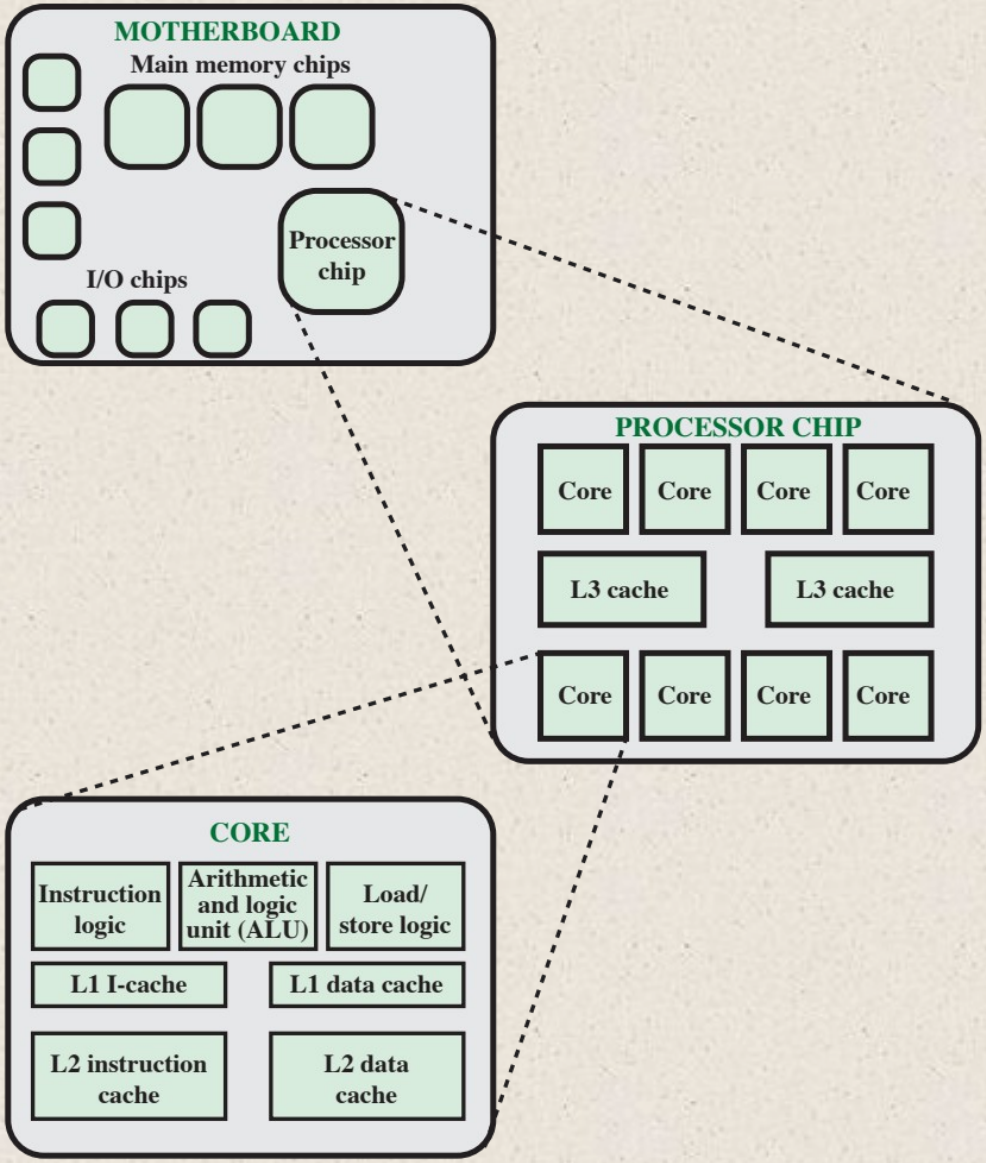


Figure 1.2 Simplified View of Major Elements of a Multicore Computer

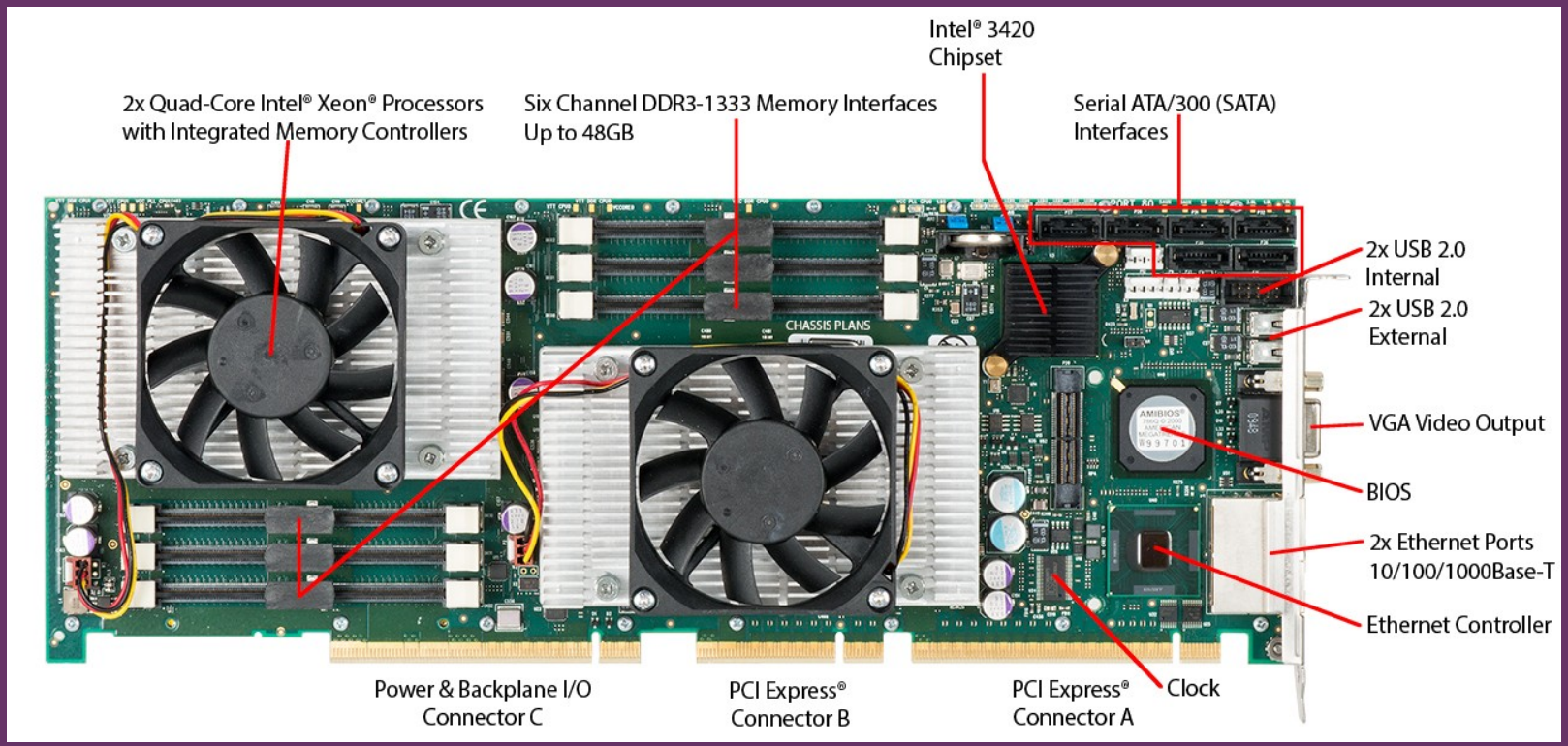


Figure 1.3
Motherboard with Two Intel Quad-Core Xeon Processors

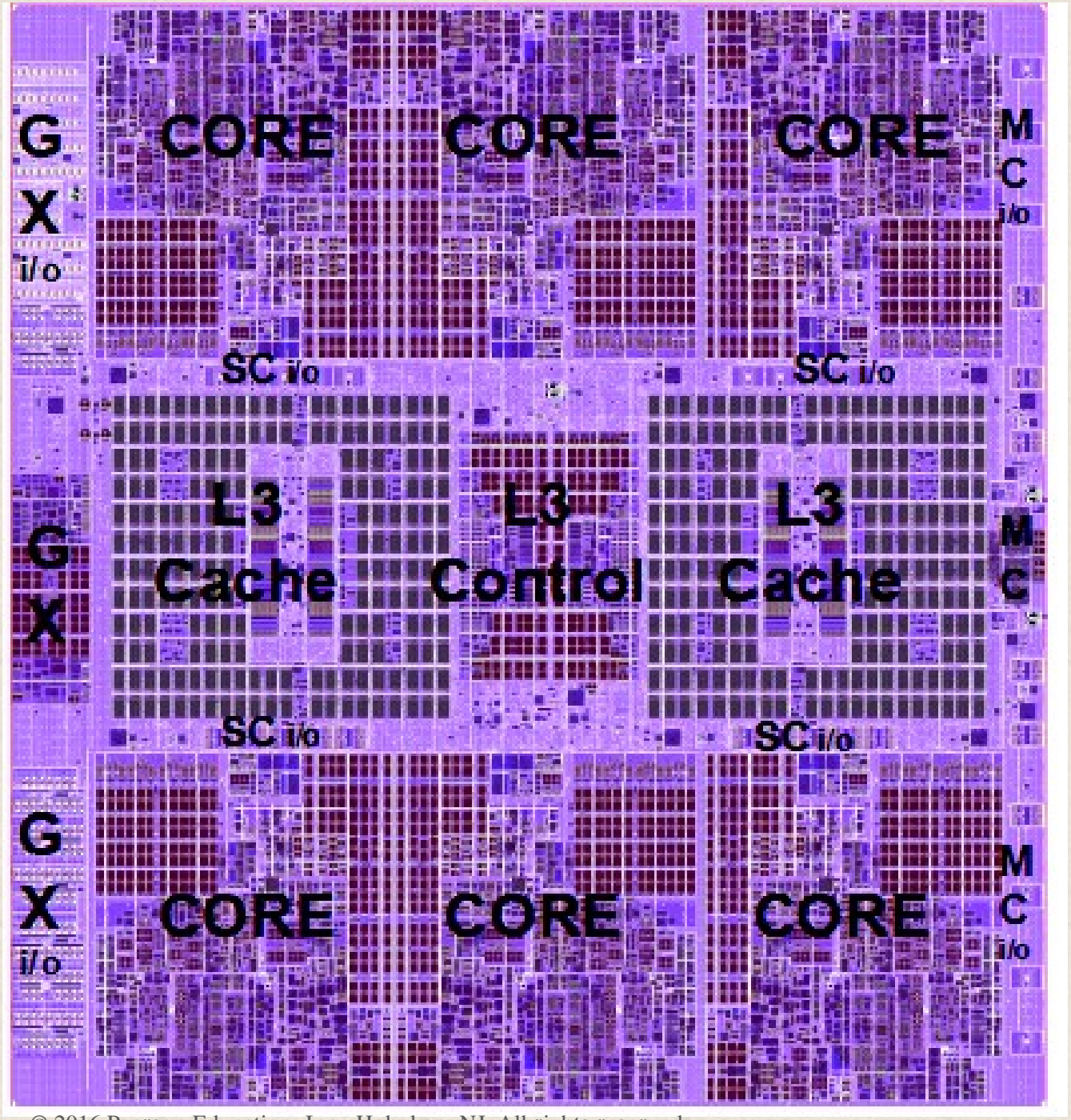


Figure 1.4

zEnterprise EC12 Processor Unit (PU) Chip Diagram

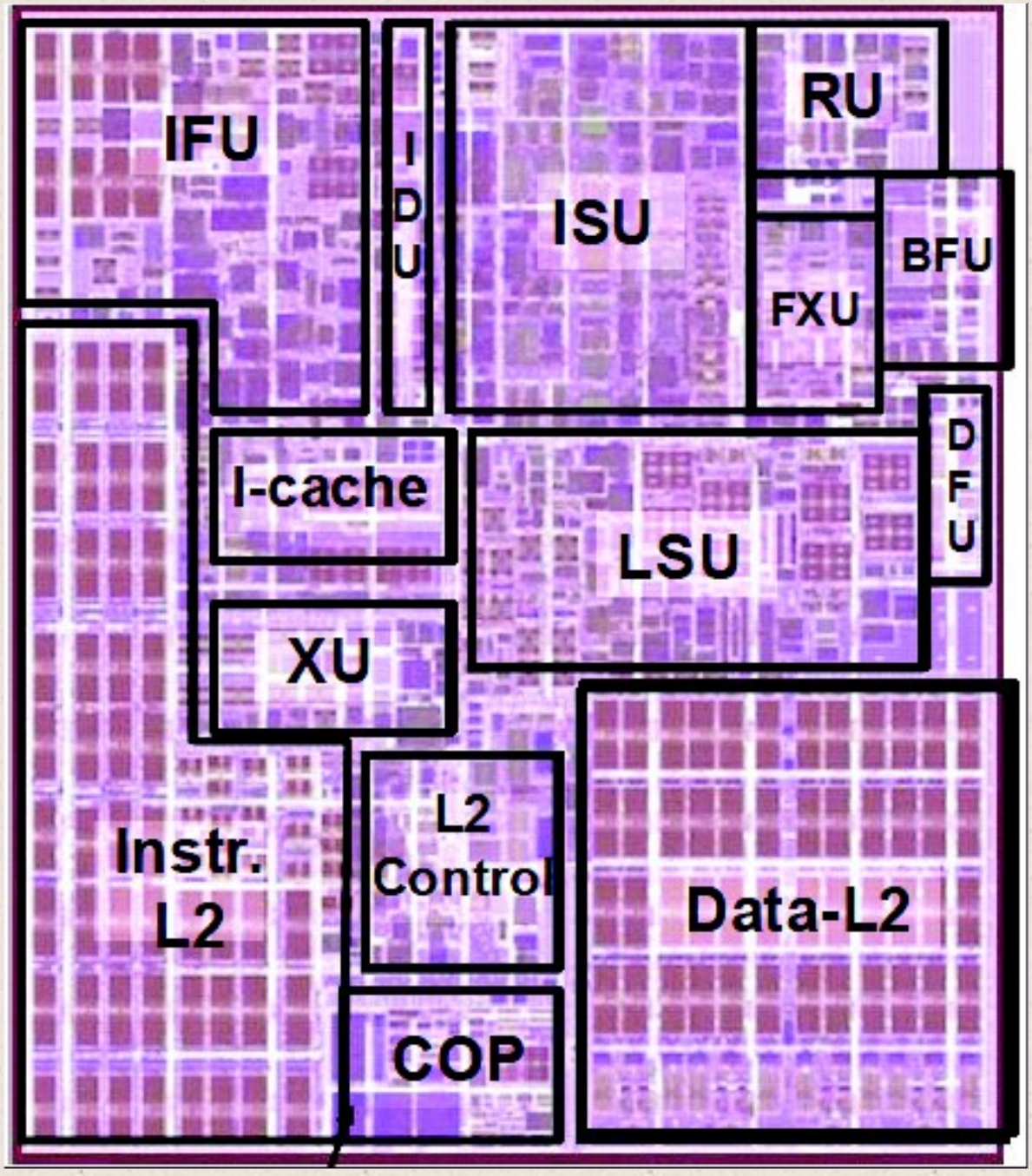


Figure 1.5

zEnterprise
EC12
Core Layout

History of Computers
First Generation: Vacuum Tubes

- Vacuum tubes were used for digital logic elements and memory
- IAS computer
  - Fundamental design approach was the stored program concept
    - Attributed to the mathematician John von Neumann
    - First publication of the idea was in 1945 for the EDVAC
  - Design began at the Princeton Institute for Advanced Studies
  - Completed in 1952
  - Prototype of all subsequent general-purpose computers

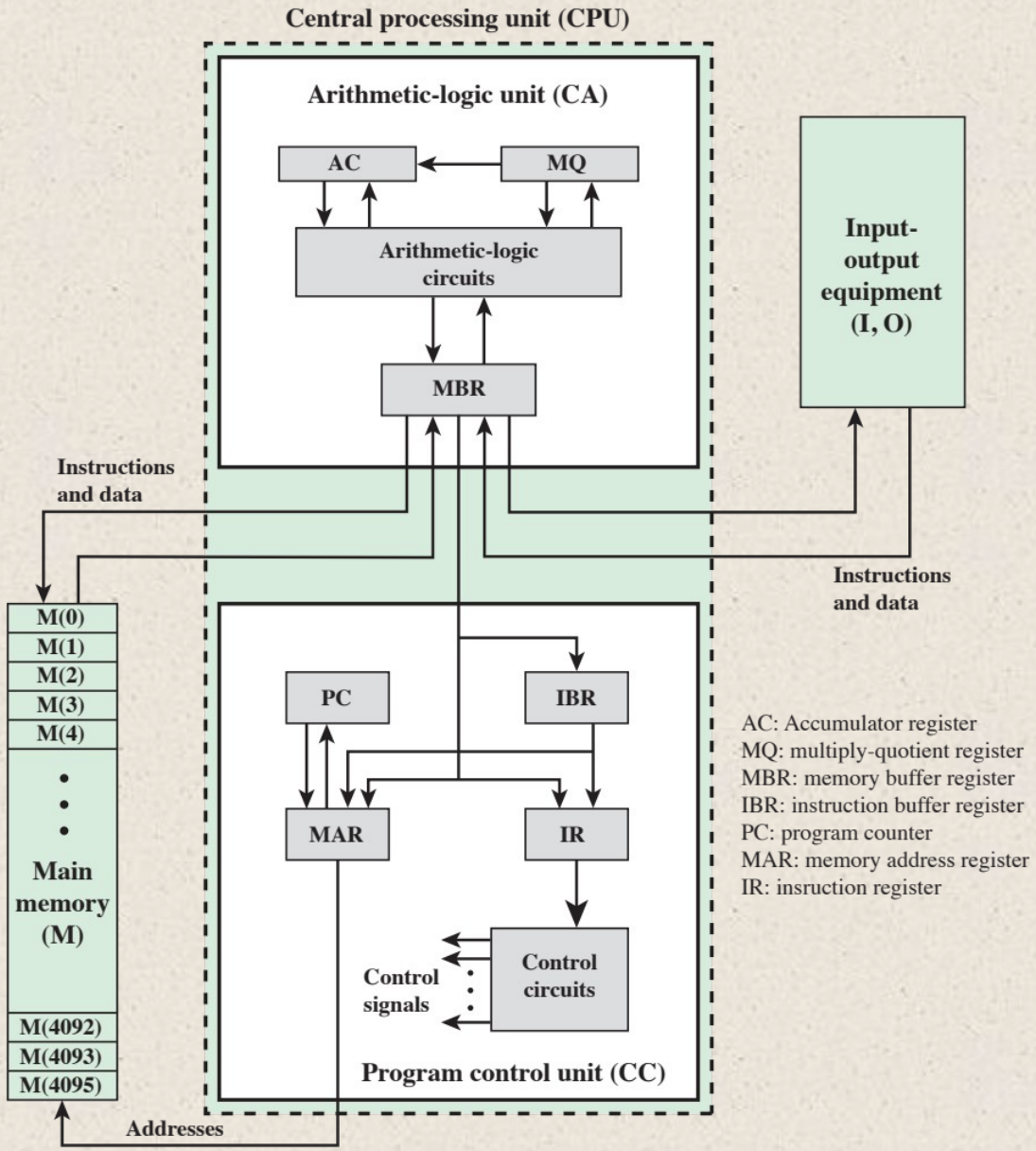
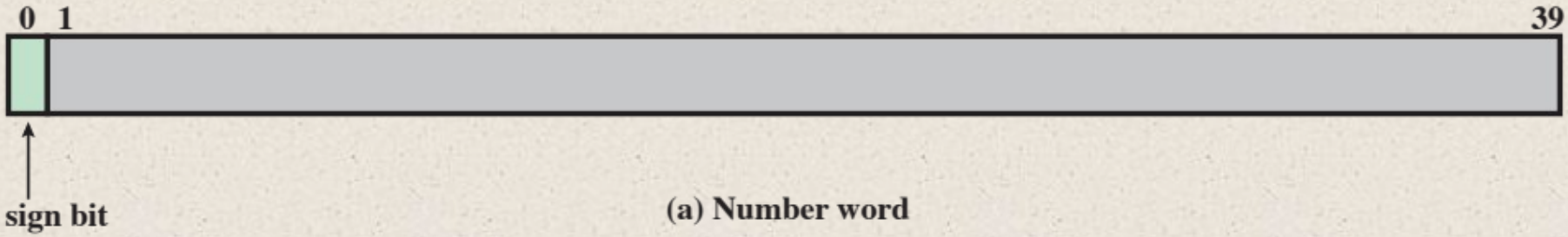
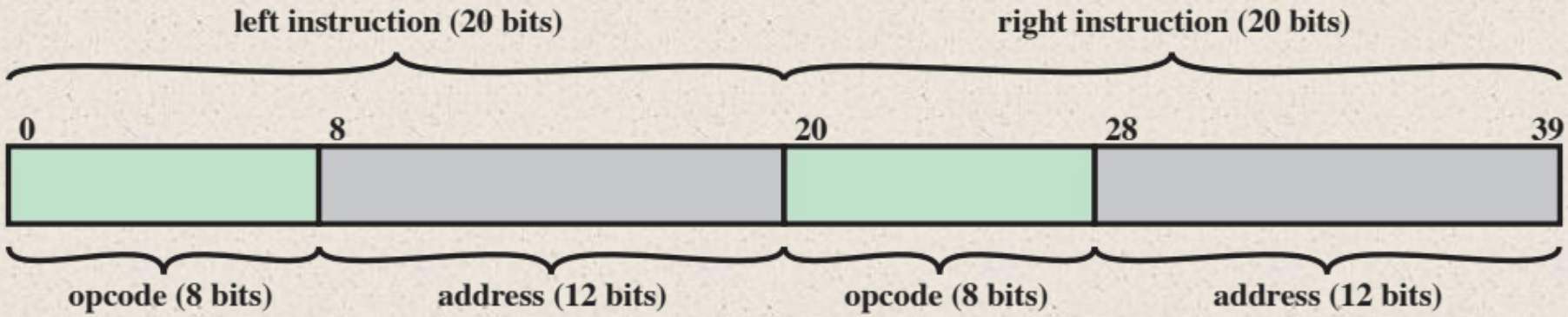


Figure 1.6 IAS Structure



(a) Number word



(b) Instruction word

Figure 1.7 IAS Memory Formats

Registers

Memory buffer register (MBR)	• Contains a word to be stored in memory or sent to the I/O unit • Or is used to receive a word from memory or from the I/O unit
Memory address register (MAR)	• Specifies the address in memory of the word to be written from or read into the MBR
Instruction register (IR)	• Contains the 8-bit opcode instruction being executed
Instruction buffer register (IBR)	• Employed to temporarily hold the right-hand instruction from a word in memory
Program counter (PC)	• Contains the address of the next instruction pair to be fetched from memory
Accumulator (AC) and multiplier quotient (MQ)	• Employed to temporarily hold operands and results of ALU operations

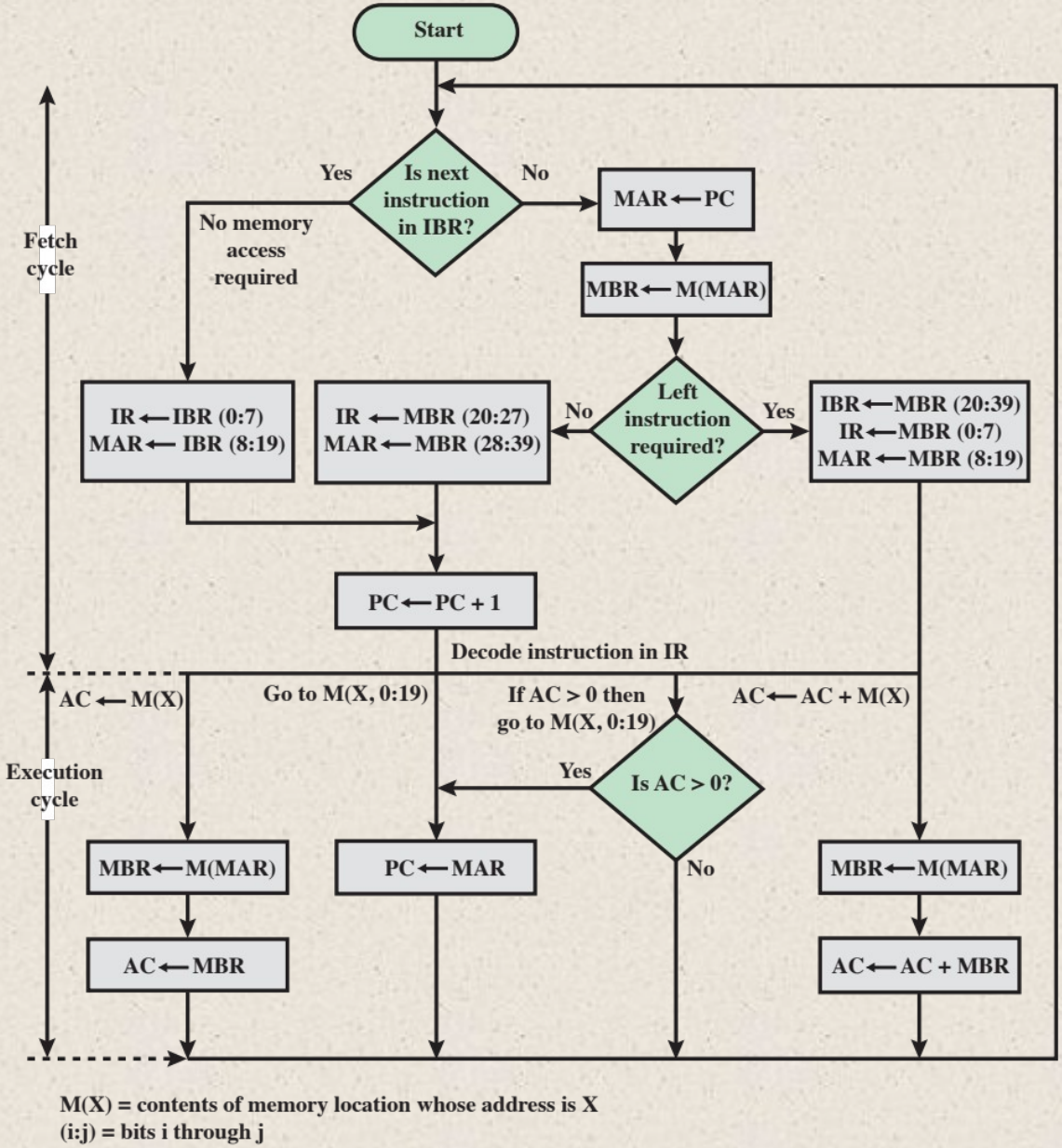


Figure 1.8 Partial Flowchart of IAS Operation

Instruction Type	Opcode	Symbolic Representation	Description
Data transfer	00001010	LOAD MQ	Transfer contents of register MQ to the accumulator AC
	00001001	LOAD MQ,M(X)	Transfer contents of memory location X to MQ
	00100001	STOR M(X)	Transfer contents of accumulator to memory location X
	00000001	LOAD M(X)	Transfer M(X) to the accumulator
	00000010	LOAD –M(X)	Transfer –M(X) to the accumulator
	00000011	LOAD |M(X)|	Transfer absolute value of M(X) to the accumulator
	00000100	LOAD –|M(X)|	Transfer –|M(X)| to the accumulator
Unconditional branch	00001101	JUMP M(X,0:19)	Take next instruction from left half of M(X)
	00001110	JUMP M(X,20:39)	Take next instruction from right half of M(X)
Conditional branch	00001111	JUMP+ M(X,0:19)	If number in the accumulator is nonnegative, take next instruction from left half of M(X)
		JUMP + M(X,20:39)	*If number in the accumulator is nonnegative, take next instruction from right half of M(X)*
Arithmetic	00000101	ADD M(X)	Add M(X) to AC; put the result in AC
	00000111	ADD |M(X)|	Add |M(X)| to AC; put the result in AC
	00000110	SUB M(X)	Subtract M(X) from AC; put the result in AC
	00001000	SUB |M(X)|	Subtract |M(X)| from AC; put the remainder in AC
	00001011	MUL M(X)	Multiply M(X) by MQ; put most significant bits of result in AC, put least significant bits in MQ
	00001100	DIV M(X)	Divide AC by M(X); put the quotient in MQ and the remainder in AC
	00010100	LSH	Multiply accumulator by 2; i.e., shift left one bit position
	00010101	RSH	Divide accumulator by 2; i.e., shift right one position
Address modify	00010010	STOR M(X,8:19)	Replace left address field at M(X) by 12 rightmost bits of AC
	00010011	STOR M(X,28:39)	Replace right address field at M(X) by 12 rightmost bits of AC

Table 1.1

The IAS Instruction Set

(Table can be found on page 17 in the textbook.)

History of Computers
Second Generation: Transistors

- Smaller
- Cheaper
- Dissipates less heat than a vacuum tube
- Is a *solid state device* made from silicon
- Was invented at Bell Labs in 1947
- It was not until the late 1950's that fully transistorized computers were commercially available

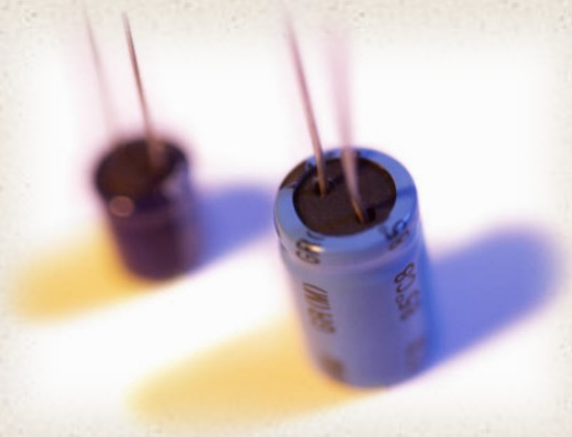
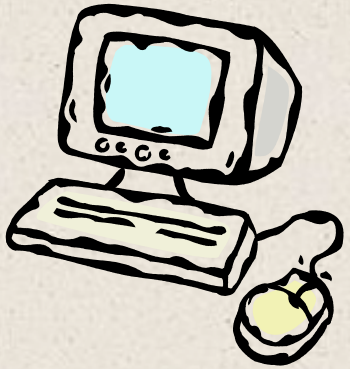


Table 1.2
Computer Generations

Generation	Approximate Dates	Technology	Typical Speed (operations per second)
1	1946–1957	Vacuum tube	40,000
2	1957–1964	Transistor	200,000
3	1965–1971	Small and medium scale integration	1,000,000
4	1972–1977	Large scale integration	10,000,000
5	1978–1991	Very large scale integration	100,000,000
6	1991-	Ultra large scale integration	>1,000,000,000

Second Generation Computers

- Introduced:
  - More complex arithmetic and logic units and control units
  - The use of high-level programming languages
  - Provision of *system software* which provided the ability to:
    - Load programs
    - Move data to peripherals
    - Libraries perform common computations



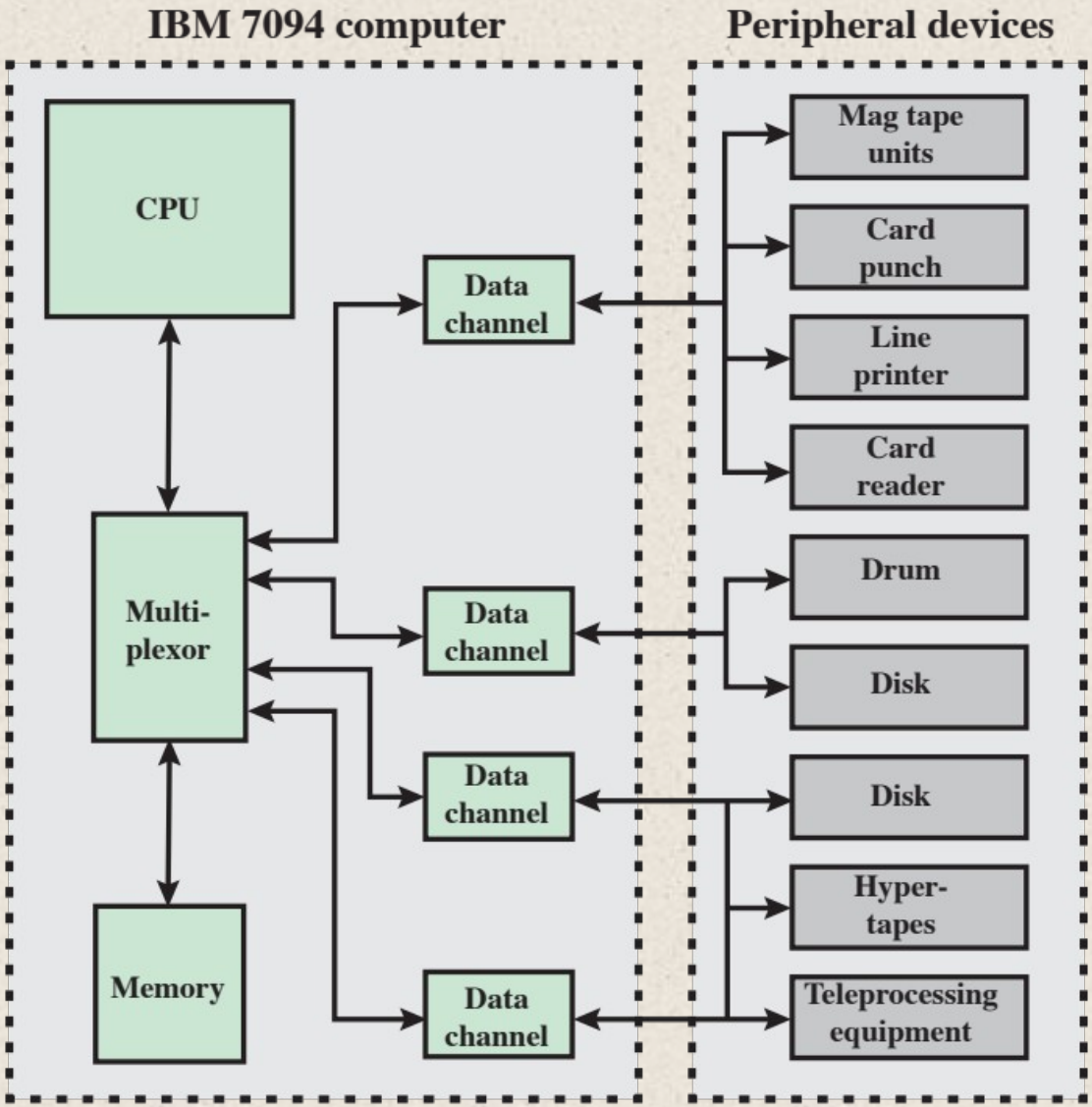


Figure 1.9 An IBM 7094 Configuration

History of Computers

Third Generation: Integrated Circuits

- 1958 – the invention of the integrated circuit
- *Discrete component*
  - Single, self-contained transistor
  - Manufactured separately, packaged in their own containers, and soldered or wired together onto masonite-like circuit boards
  - Manufacturing process was expensive and cumbersome
- The two most important members of the third generation were the IBM System/360 and the DEC PDP-8

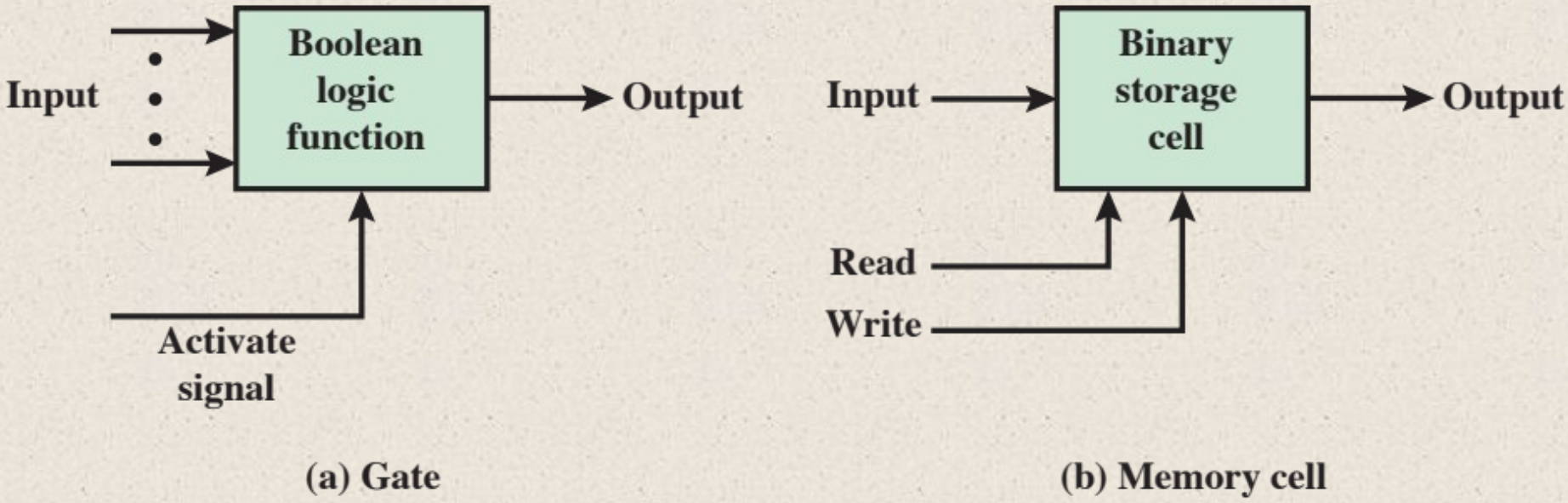


Figure 1.10 Fundamental Computer Elements

Integrated Circuits

- Data storage – provided by memory cells
- Data processing – provided by gates
- Data movement – the paths among components are used to move data from memory to memory and from memory through gates to memory
- Control – the paths among components can carry control signals

- A computer consists of gates, memory cells, and interconnections among these elements
- The gates and memory cells are constructed of simple digital electronic components
- Exploits the fact that such components as transistors, resistors, and conductors can be fabricated from a semiconductor such as silicon
- Many transistors can be produced at the same time on a single wafer of silicon
- Transistors can be connected with a processor metallization to form circuits

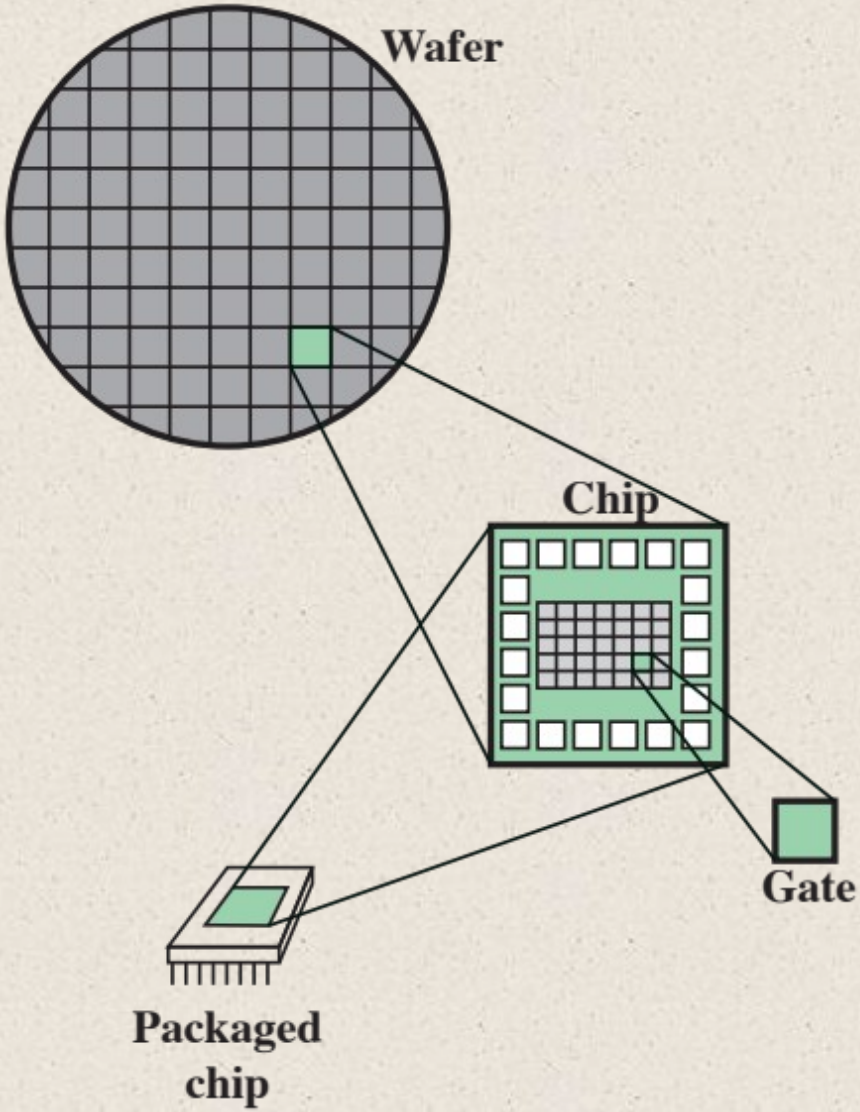


Figure 1.11 Relationship Among Wafer, Chip, and Gate

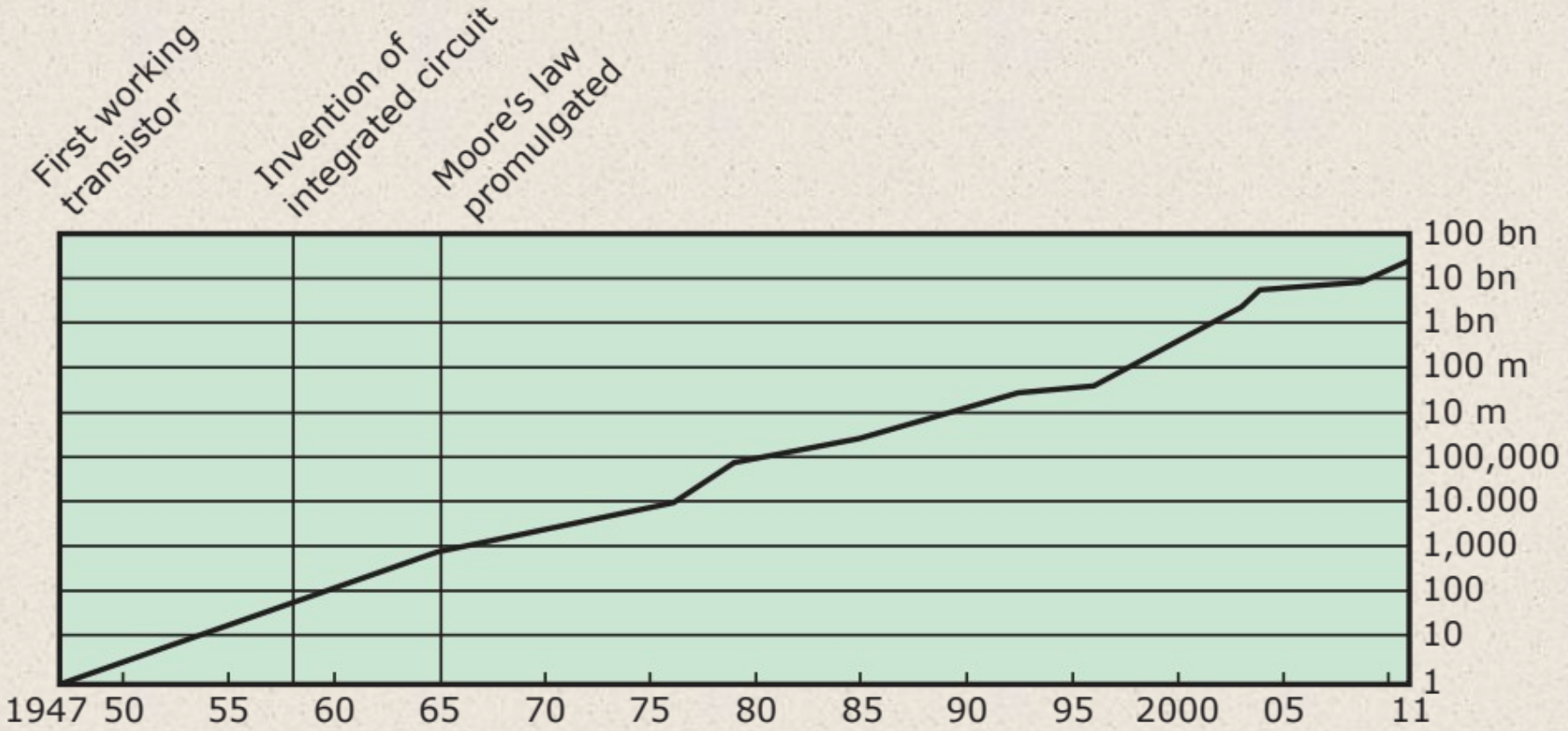


Figure 1.12 Growth in Transistor Count on Integrated Circuits (DRAM memory)

Moore's Law

1965; Gordon Moore – co-founder of Intel

Observed number of transistors that could be put on a single chip was doubling every year

The pace slowed to a doubling every 18 months in the 1970's but has sustained that rate ever since

Consequences of Moore's law:

- **The cost of computer logic and memory circuitry has fallen at a dramatic rate**
- **The electrical path length is shortened, increasing operating speed**
- **Computer becomes smaller and is more convenient to use in a variety of environments**
- **Reduction in power and cooling requirements**
- **Fewer interchip connections**

IBM System/360

- Announced in 1964
- Product line was incompatible with older IBM machines
- Was the success of the decade and cemented IBM as the overwhelmingly dominant computer vendor
- The architecture remains to this day the architecture of IBM's mainframe computers
- Was the industry's first planned family of computers
  - Models were compatible in the sense that a program written for one model should be capable of being executed by another model in the series

Family Characteristics

- Similar or identical instruction set
- Similar or identical operating system
- Increasing speed
- Increasing number of I/O ports
- Increasing memory size
- Increasing cost

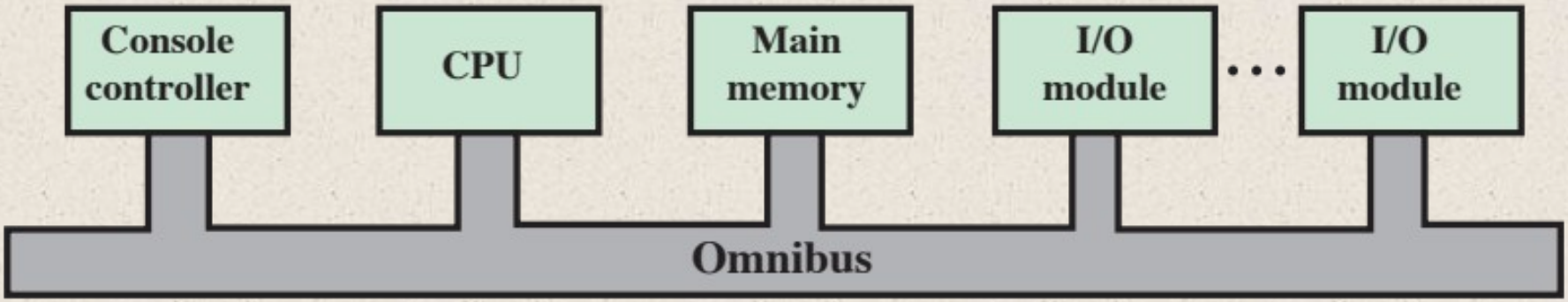
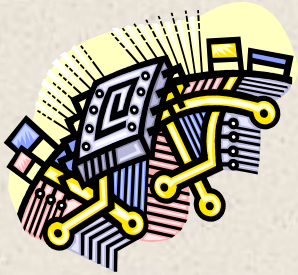


Figure 1.13 PDP-8 Bus Structure

Later Generations

LSI
Large Scale Integration

VLSI
Very Large Scale Integration



Semiconductor Memory
Microprocessors

ULSI
Ultra Large Scale Integration

Semiconductor Memory

In 1970 Fairchild produced the first relatively capacious semiconductor memory

- Chip was about the size of a single core
- Could hold 256 bits of memory
- Non-destructive
- Much faster than core

In 1974 the price per bit of semiconductor memory dropped below the price per bit of core memory

- There has been a continuing and rapid decline in memory cost accompanied by a corresponding increase in physical memory density
- Developments in memory and processor technologies changed the nature of computers in less than a decade

Since 1970 semiconductor memory has been through 13 generations

- Each generation has provided four times the storage density of the previous generation, accompanied by declining cost per bit and declining access time

Microprocessors

- The density of elements on processor chips continued to rise
  - More and more elements were placed on each chip so that fewer and fewer chips were needed to construct a single computer processor
- 1971 Intel developed 4004
  - First chip to contain all of the components of a CPU on a single chip
  - Birth of microprocessor
- 1972 Intel developed 8008
  - First 8-bit microprocessor
- 1974 Intel developed 8080
  - First general purpose microprocessor
  - Faster, has a richer instruction set, has a large addressing capability

Evolution of Intel Microprocessors

	4004	8008	8080	8086	8088
Introduced	1971	1972	1974	1978	1979
Clock speeds	108 kHz	108 kHz	2 MHz	5 MHz, 8 MHz, 10 MHz	5 MHz, 8 MHz
Bus width	4 bits	8 bits	8 bits	16 bits	8 bits
Number of transistors	2,300	3,500	6,000	29,000	29,000
Feature size (μm)	10	8	6	3	6
Addressable memory	640 Bytes	16 KB	64 KB	1 MB	1 MB

(a) 1970s Processors

Evolution of Intel Microprocessors

	80286	386TM DX	386TM SX	486TM DX CPU
Introduced	1982	1985	1988	1989
Clock speeds	6 MHz - 12.5 MHz	16 MHz - 33 MHz	16 MHz - 33 MHz	25 MHz - 50 MHz
Bus width	16 bits	32 bits	16 bits	32 bits
Number of transistors	134,000	275,000	275,000	1.2 million
Feature size (μm)	1.5	1	1	0.8 - 1
Addressable memory	16 MB	4 GB	16 MB	4 GB
Virtual memory	1 GB	64 TB	64 TB	64 TB
Cache	—	—	—	8 kB

(b) 1980s Processors

Evolution of Intel Microprocessors

	486TM SX	Pentium	Pentium Pro	Pentium II
Introduced	1991	1993	1995	1997
Clock speeds	16 MHz - 33 MHz	60 MHz - 166 MHz,	150 MHz - 200 MHz	200 MHz - 300 MHz
Bus width	32 bits	32 bits	64 bits	64 bits
Number of transistors	1.185 million	3.1 million	5.5 million	7.5 million
Feature size (μm)	1	0.8	0.6	0.35
Addressable memory	4 GB	4 GB	64 GB	64 GB
Virtual memory	64 TB	64 TB	64 TB	64 TB
Cache	8 kB	8 kB	512 kB L1 and 1 MB L2	512 kB L2

(c) 1990s Processors

Evolution of Intel Microprocessors

	Pentium III	Pentium 4	Core 2 Duo	Core i7 EE 4960X
Introduced	1999	2000	2006	2013
Clock speeds	450 - 660 MHz	1.3 - 1.8 GHz	1.06 - 1.2 GHz	4 GHz
Bus width	64 bits	64 bits	64 bits	64 bits
Number of transistors	9.5 million	42 million	167 million	1.86 billion
Feature size (nm)	250	180	65	22
Addressable memory	64 GB	64 GB	64 GB	64 GB
Virtual memory	64 TB	64 TB	64 TB	64 TB
Cache	512 kB L2	256 kB L2	2 MB L2	1.5 MB L2/15 MB L3
Number of cores	1	1	2	6

(d) Recent Processors

The Evolution of the Intel x86 Architecture

- Two processor families are the Intel x86 and the ARM architectures
- Current x86 offerings represent the results of decades of design effort on complex instruction set computers (CISCs)
- An alternative approach to processor design is the reduced instruction set computer (RISC)
- ARM architecture is used in a wide variety of embedded systems and is one of the most powerful and best-designed RISC-based systems on the market

Highlights of the Evolution of the Intel Product Line:

8080

- World's first general-purpose microprocessor
- 8-bit machine, 8-bit data path to memory
- Was used in the first personal computer (Altair)

8086

- A more powerful 16-bit machine
- Has an instruction cache, or queue, that prefetches a few instructions before they are executed
- The first appearance of the x86 architecture
- The 8088 was a variant of this processor and used in IBM's first personal computer (securing the success of Intel

80286

- Extension of the 8086 enabling addressing a 16-MB memory instead of just 1MB

80386

- Intel's first 32-bit machine
- First Intel processor to support multitasking

80486

- Introduced the use of much more sophisticated and powerful cache technology and sophisticated instruction pipelining
- Also offered a built-in math coprocessor

Highlights of the Evolution of the Intel Product Line:

Pentium

- Intel introduced the use of superscalar techniques, which allow multiple instructions to execute in parallel

Pentium Pro

- Continued the move into superscalar organization with aggressive use of register renaming, branch prediction, data flow analysis, and speculative execution

Pentium II

- Incorporated Intel MMX technology, which is designed specifically to process video, audio, and graphics data efficiently

Pentium III

- Incorporated additional floating-point instructions
- Streaming SIMD Extensions (SSE)

Pentium 4

- Includes additional floating-point and other enhancements for multimedia

Core

- First Intel x86 micro-core

Core 2

- Extends the Core architecture to 64 bits
- Core 2 Quad provides four cores on a single chip
- More recent Core offerings have up to 10 cores per chip
- An important addition to the architecture was the Advanced Vector Extensions instruction set

Embedded Systems

- The use of electronics and software within a product
- Billions of computer systems are produced each year that are embedded within larger devices
- Today many devices that use electric power have an embedded computing system
- Often embedded systems are tightly coupled to their environment
  - This can give rise to real-time constraints imposed by the need to interact with the environment
    - Constraints such as required speeds of motion, required precision of measurement, and required time durations, dictate the timing of software operations
  - If multiple activities must be managed simultaneously this imposes more complex real-time constraints

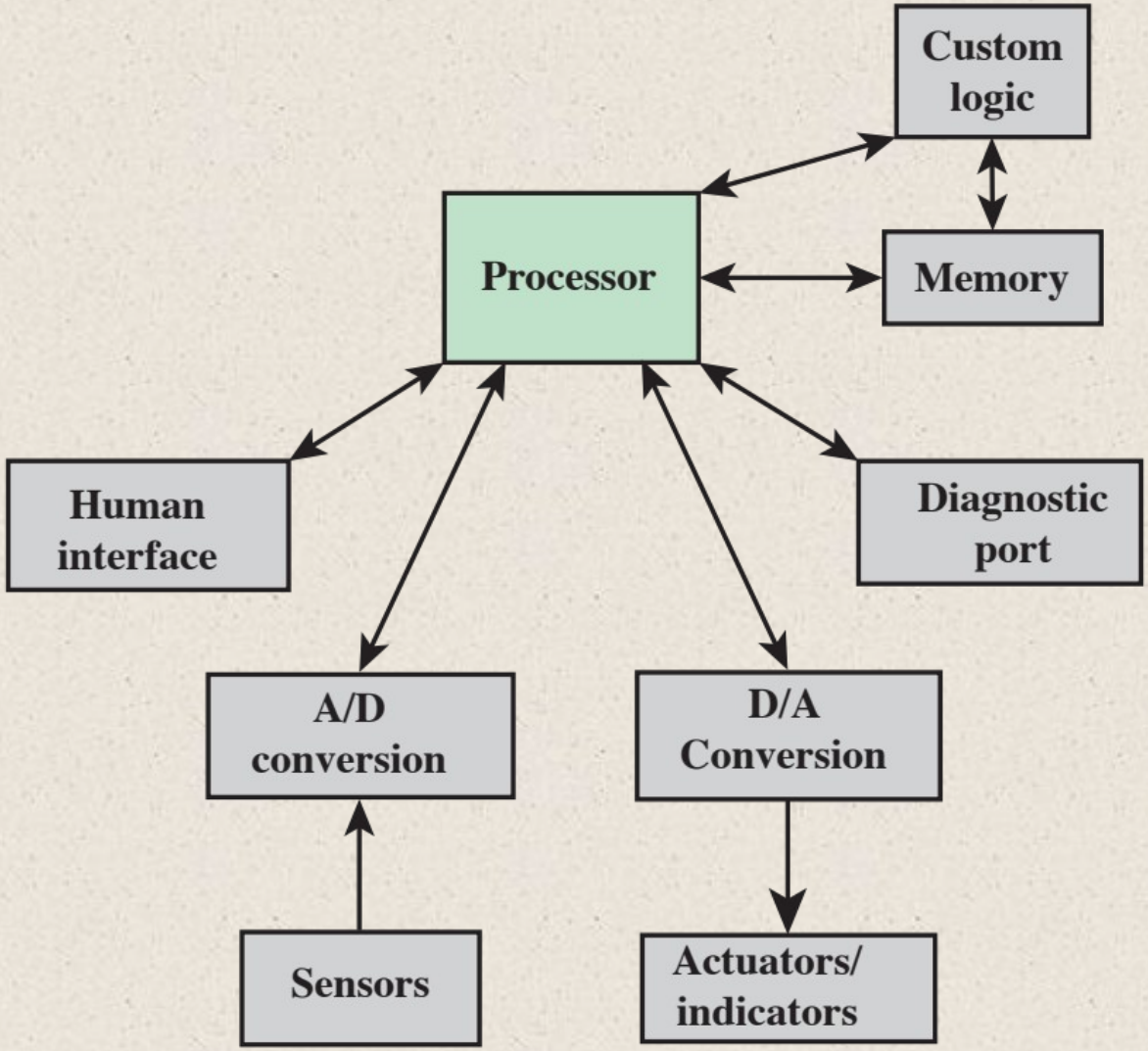


Figure 1.14 Possible Organization of an Embedded System

The Internet of Things (IoT)

- Term that refers to the expanding interconnection of smart devices, ranging from appliances to tiny sensors
- Is primarily driven by deeply embedded devices
- Generations of deployment culminating in the IoT:
  - Information technology (IT)
    - PCs, servers, routers, firewalls, and so on, bought as IT devices by enterprise IT people and primarily using wired connectivity
  - Operational technology (OT)
    - Machines/appliances with embedded IT built by non-IT companies, such as medical machinery, SCADA, process control, and kiosks, bought as appliances by enterprise OT people and primarily using wired connectivity
  - Personal technology
    - Smartphones, tablets, and eBook readers bought as IT devices by consumers exclusively using wireless connectivity and often multiple forms of wireless connectivity
  - Sensor/actuator technology
    - Single-purpose devices bought by consumers, IT, and OT people exclusively using wireless connectivity, generally of a single form, as part of larger systems
- It is the fourth generation that is usually thought of as the IoT and it is marked by the use of billions of embedded devices

Embedded Operating Systems

Application Processors versus Dedicated Processors

- There are two general approaches to developing an embedded operating system (OS):
  - Take an existing OS and adapt it for the embedded application
  - Design and implement an OS intended solely for embedded use

- Application processors
  - Defined by the processor's ability to execute complex operating systems
  - General-purpose in nature
  - An example is the smartphone – the embedded system is designed to support numerous apps and perform a wide variety of functions
- Dedicated processor
  - Is dedicated to one or a small number of specific tasks required by the host device
  - Because such an embedded system is dedicated to a specific task or tasks, the processor and associated components can be engineered to reduce size and cost

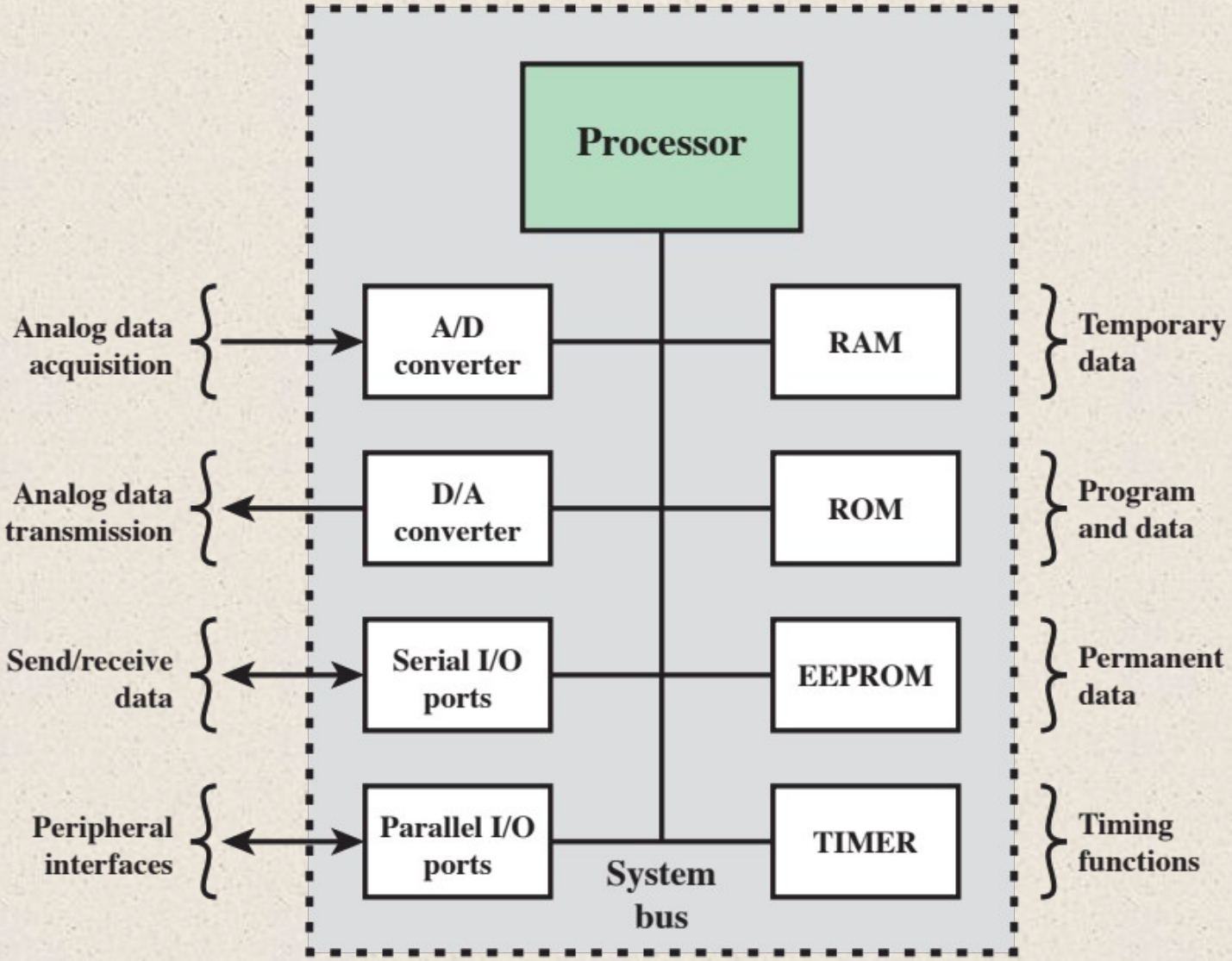
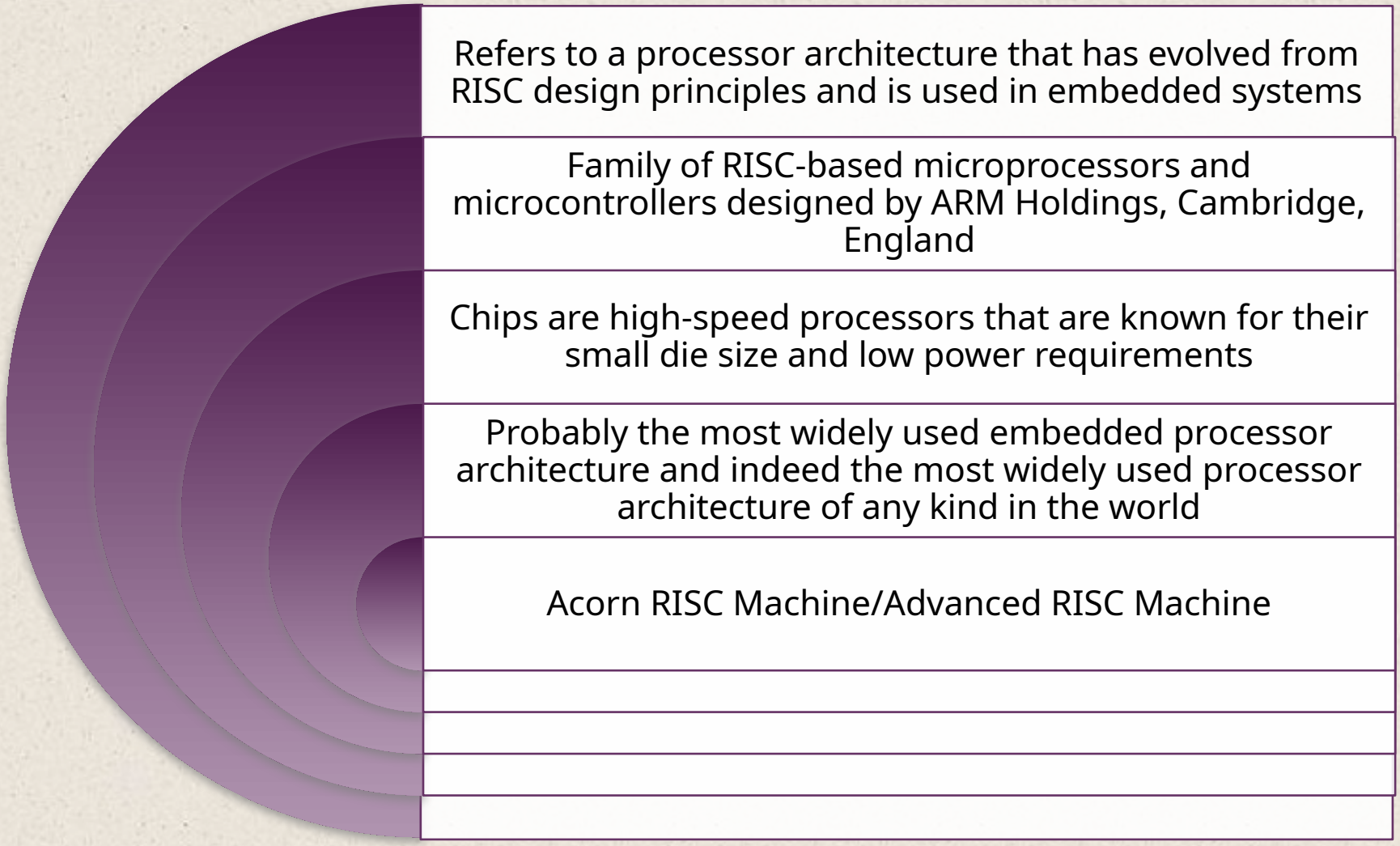


Figure 1.15 Typical Microcontroller Chip Elements

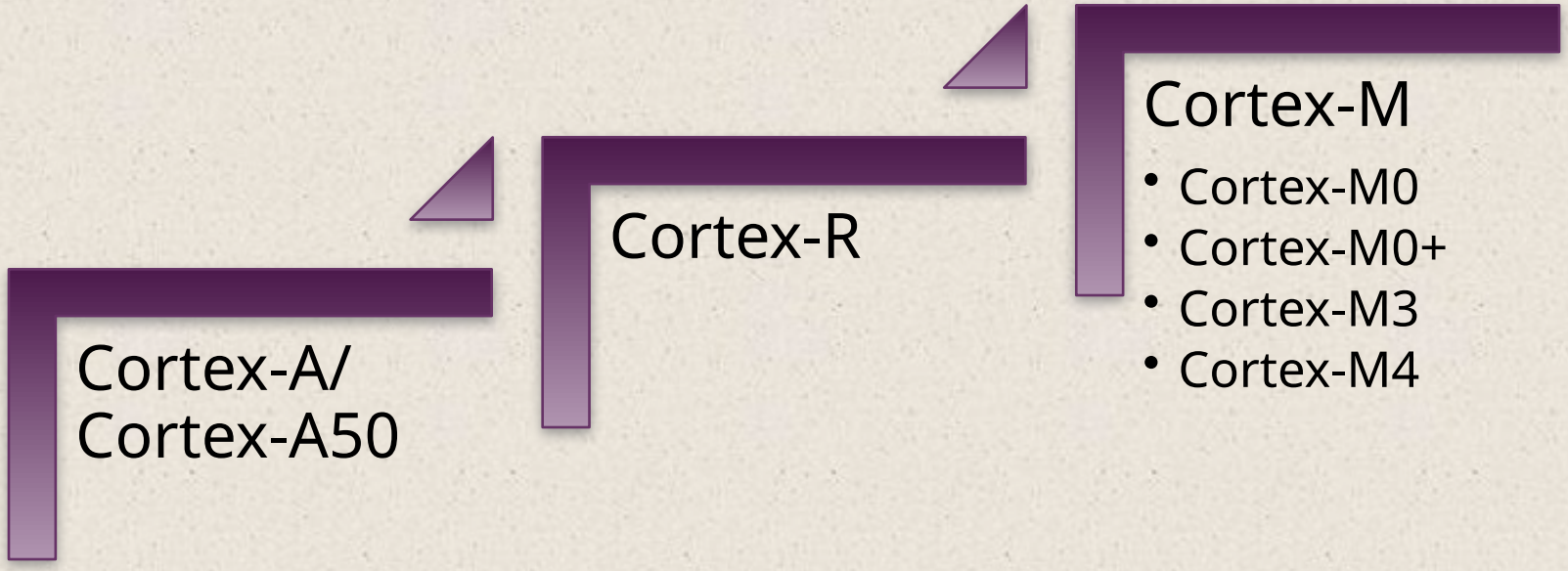
Deeply Embedded Systems

- Subset of embedded systems
- Has a processor whose behavior is difficult to observe both by the programmer and the user
- Uses a microcontroller rather than a microprocessor
- Is not programmable once the program logic for the device has been burned into ROM
- Has no interaction with a user
- Dedicated, single-purpose devices that detect something in the environment, perform a basic level of processing, and then do something with the results
- Often have wireless capability and appear in networked configurations, such as networks of sensors deployed over a large area
- Typically have extreme resource constraints in terms of memory, processor size, time, and power consumption

ARM



ARM Products



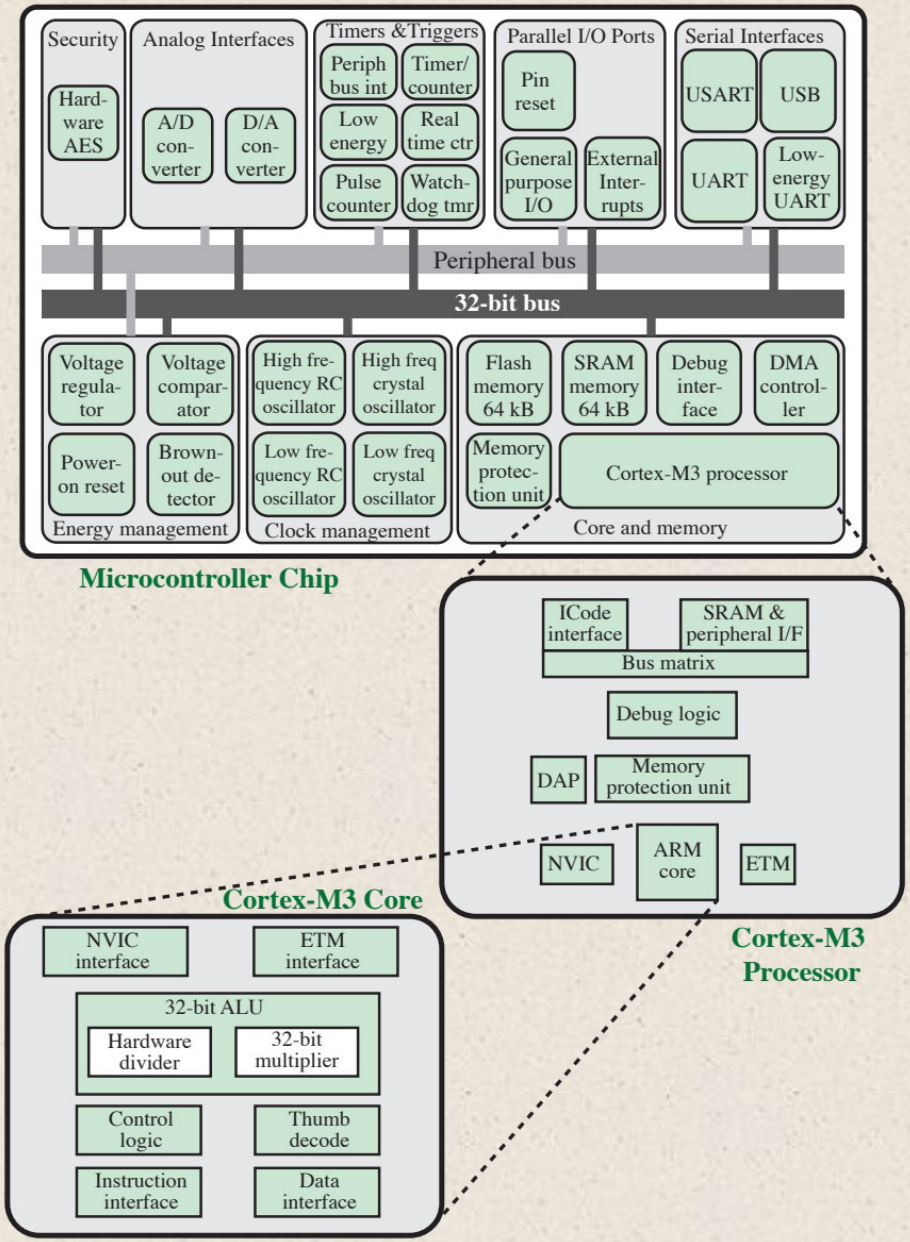


Figure 1.16 Typical Microcontroller Chip Based on Cortex-M3

Cloud Computing

- NIST defines cloud computing as:

  "A model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction."

- You get economies of scale, professional network management, and professional security management
- The individual or company only needs to pay for the storage capacity and services they need
- Cloud provider takes care of security

Cloud Networking

- Refers to the networks and network management functionality that must be in place to enable cloud computing
- One example is the provisioning of high-performance and/or high-reliability networking between the provider and subscriber
- The collection of network capabilities required to access a cloud, including making use of specialized services over the Internet, linking enterprise data center to a cloud, and using firewalls and other network security devices at critical points to enforce access security policies

Cloud Storage

- Subset of cloud computing
- Consists of database storage and database applications hosted remotely on cloud servers
- Enables small businesses and individual users to take advantage of data storage that scales with their needs and to take advantage of a variety of database applications without having to buy, maintain, and manage the storage assets

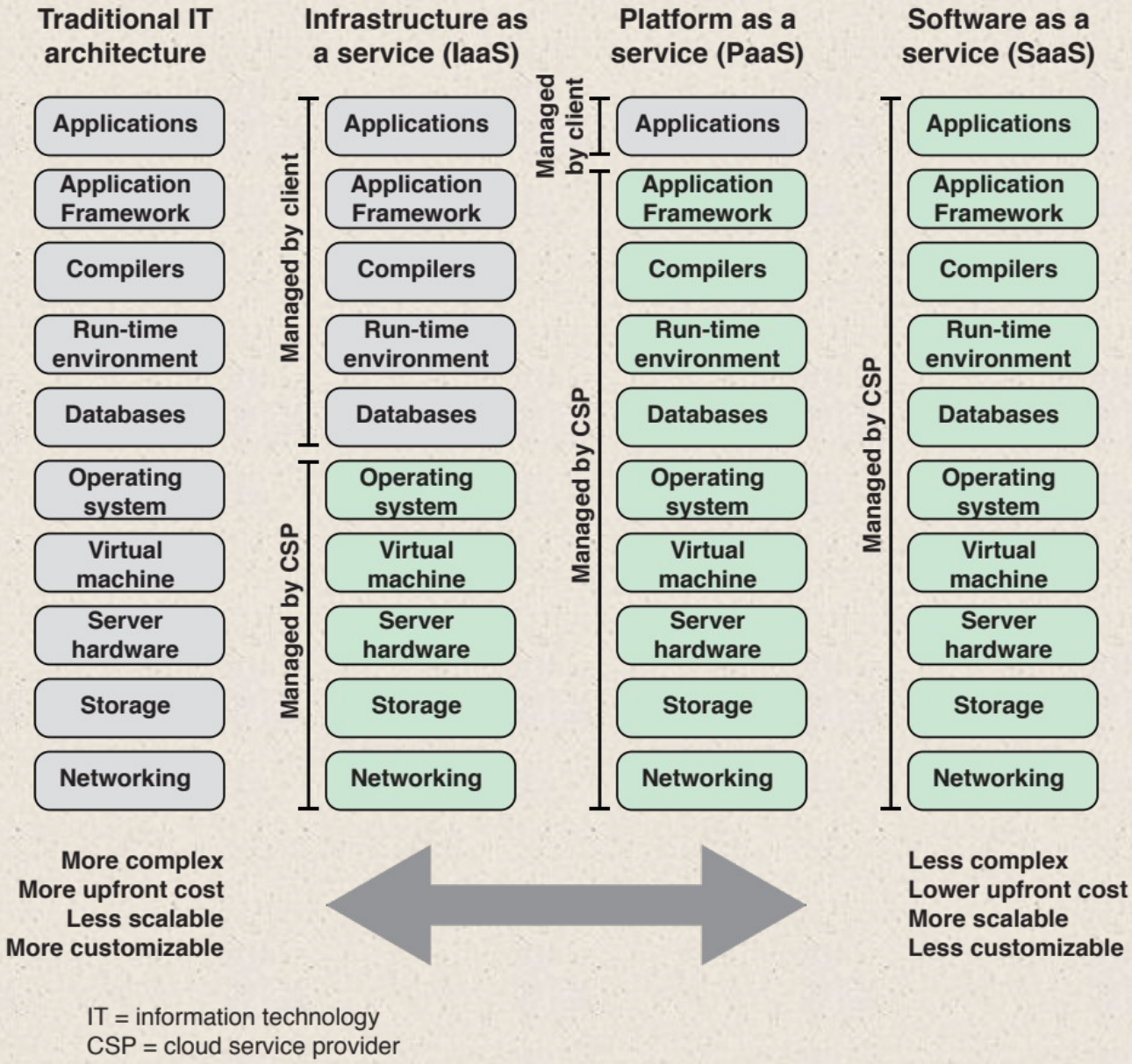


Figure 1.17 Alternative Information Technology Architectures

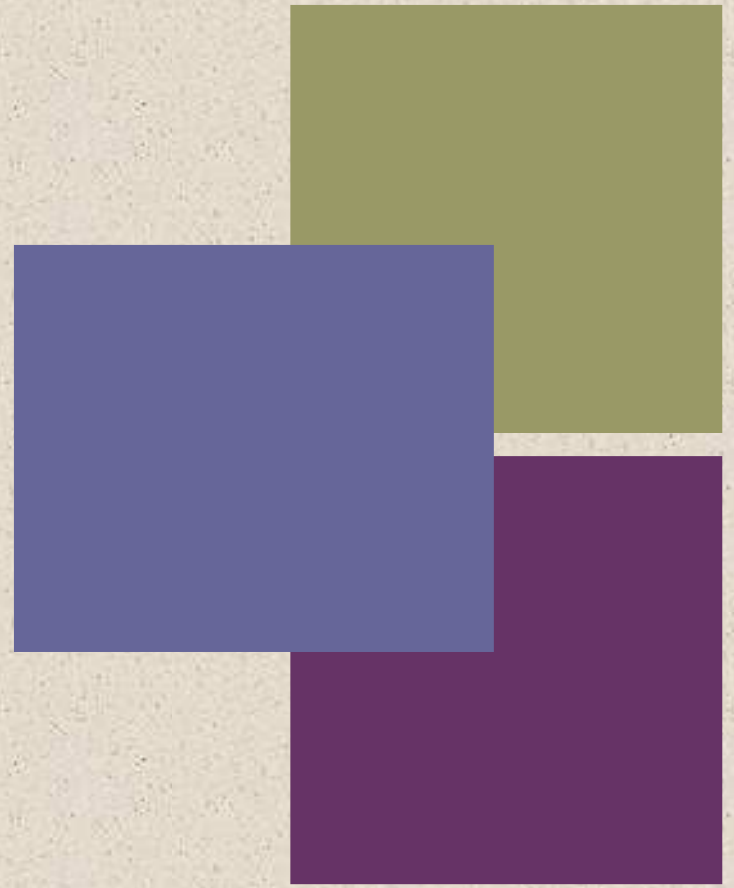
Summary

Chapter 1

Basic Concepts and Computer Evolution

- Organization and architecture
- Structure and function
- Brief history of computers
  - The First Generation: Vacuum tubes
  - The Second Generation: Transistors
  - The Third Generation: Integrated Circuits
  - Later generations
- The evolution of the Intel x86 architecture
- Cloud computing
  - Basic concepts
  - Cloud services
- Embedded systems
  - The Internet of things
  - Embedded operating systems
  - Application processors versus dedicated processors
  - Microprocessors versus microcontrollers
  - Embedded versus deeply embedded systems
- ARM architecture
  - ARM evolution
  - Instruction set architecture
  - ARM products

William Stallings
Computer Organization and Architecture
10th Edition



Chapter 2

Performance Issues

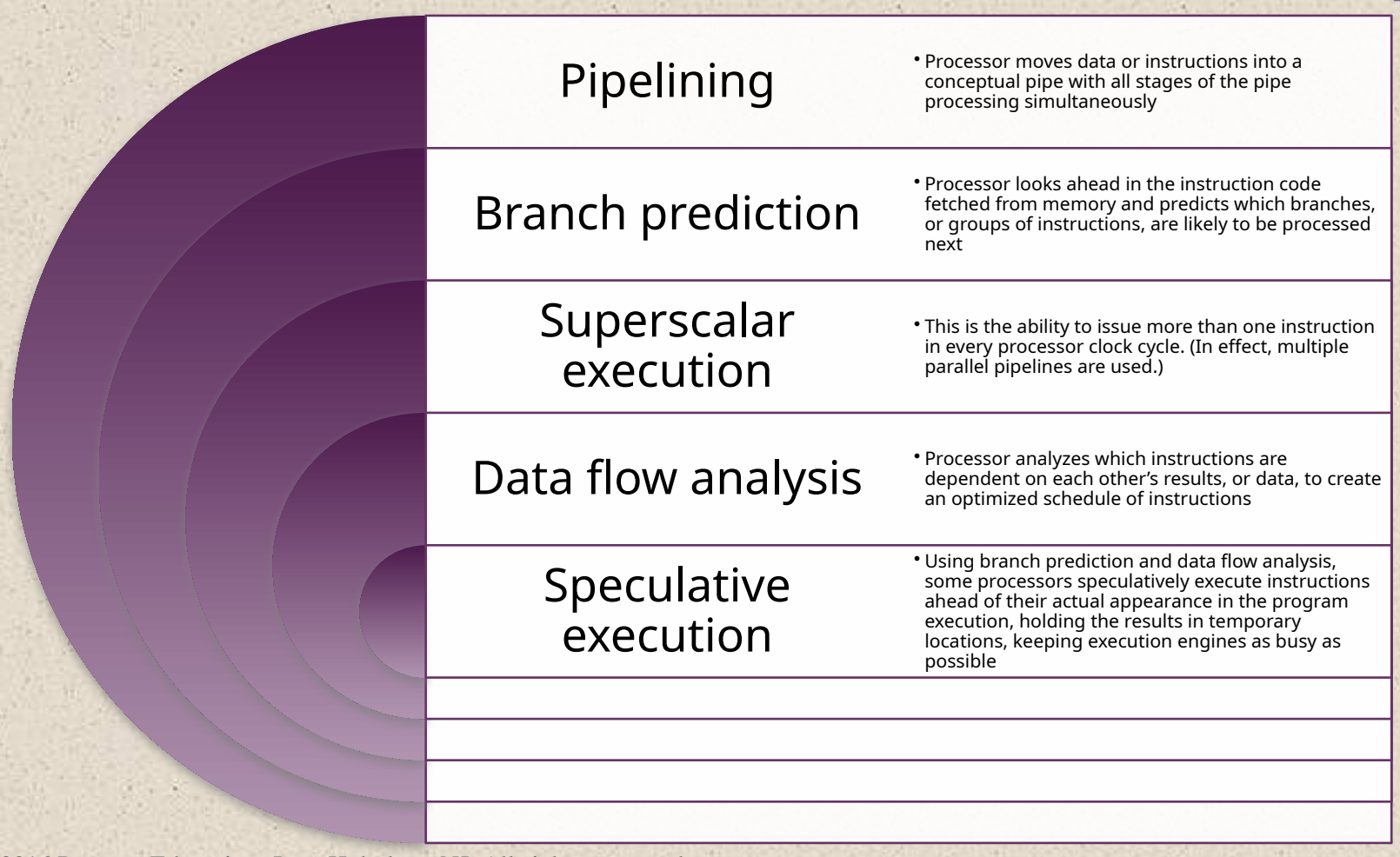
Designing for Performance

- The cost of computer systems continues to drop dramatically, while the performance and capacity of those systems continue to rise equally dramatically
- Today's laptops have the computing power of an IBM mainframe from 10 or 15 years ago
- Processors are so inexpensive that we now have microprocessors we throw away
- Desktop applications that require the great power of today's microprocessor-based systems include:
  - Image processing
  - Three-dimensional rendering
  - Speech recognition
  - Videoconferencing
  - Multimedia authoring
  - Voice and video annotation of files
  - Simulation modeling

- Businesses are relying on increasingly powerful servers to handle transaction and database processing and to support massive client/server networks that have replaced the huge mainframe computer centers of yesteryear
- Cloud service providers use massive high-performance banks of servers to satisfy high-volume, high-transaction-rate applications for a broad spectrum of clients

Microprocessor Speed

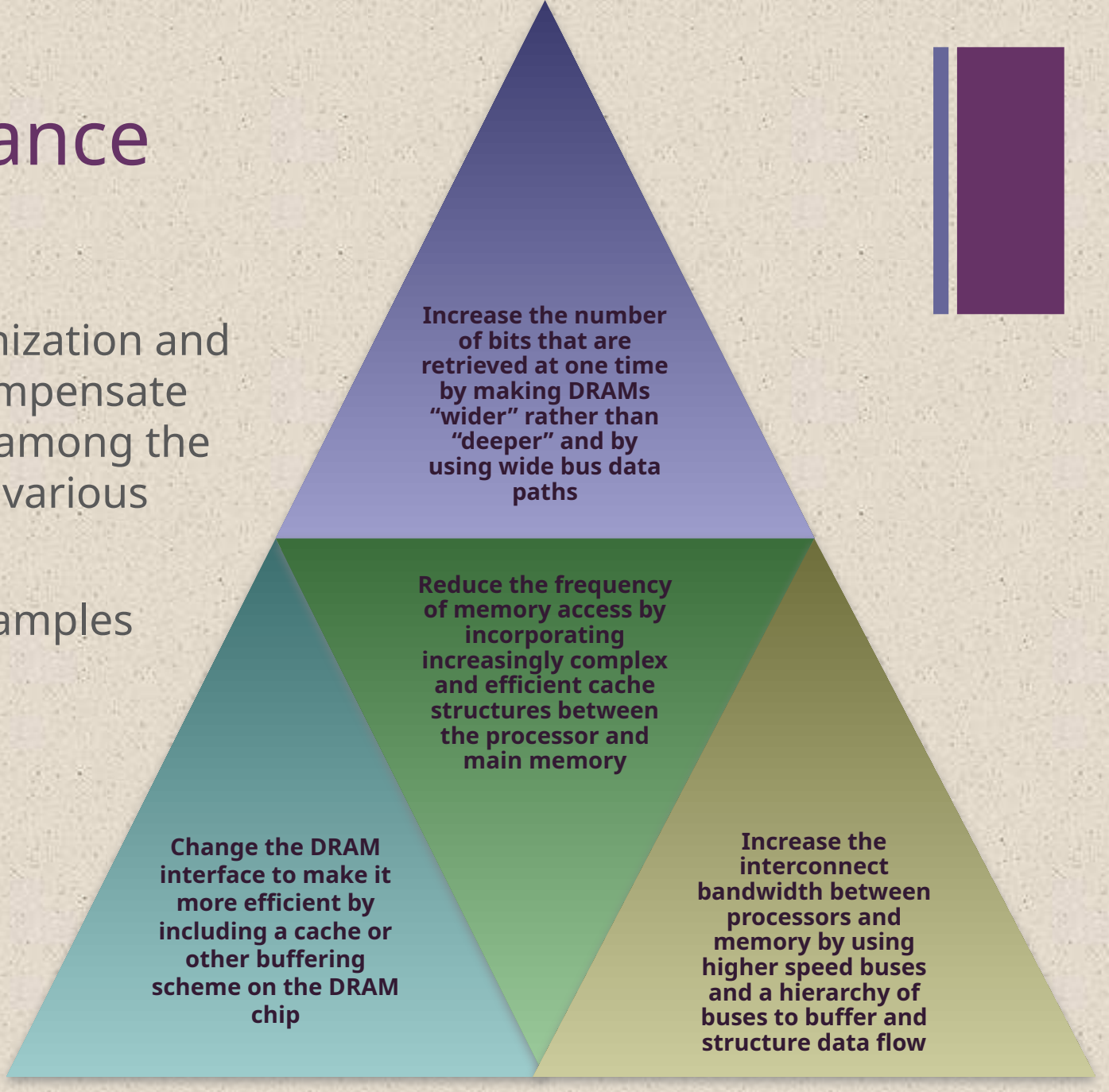
Techniques built into contemporary processors include:



Pipelining	• Processor moves data or instructions into a conceptual pipe with all stages of the pipe processing simultaneously
Branch prediction	• Processor looks ahead in the instruction code fetched from memory and predicts which branches, or groups of instructions, are likely to be processed next
Superscalar execution	• This is the ability to issue more than one instruction in every processor clock cycle. (In effect, multiple parallel pipelines are used.)
Data flow analysis	• Processor analyzes which instructions are dependent on each other's results, or data, to create an optimized schedule of instructions
Speculative execution	• Using branch prediction and data flow analysis, some processors speculatively execute instructions ahead of their actual appearance in the program execution, holding the results in temporary locations, keeping execution engines as busy as possible

Performance Balance

- Adjust the organization and architecture to compensate for the mismatch among the capabilities of the various components
- Architectural examples include:



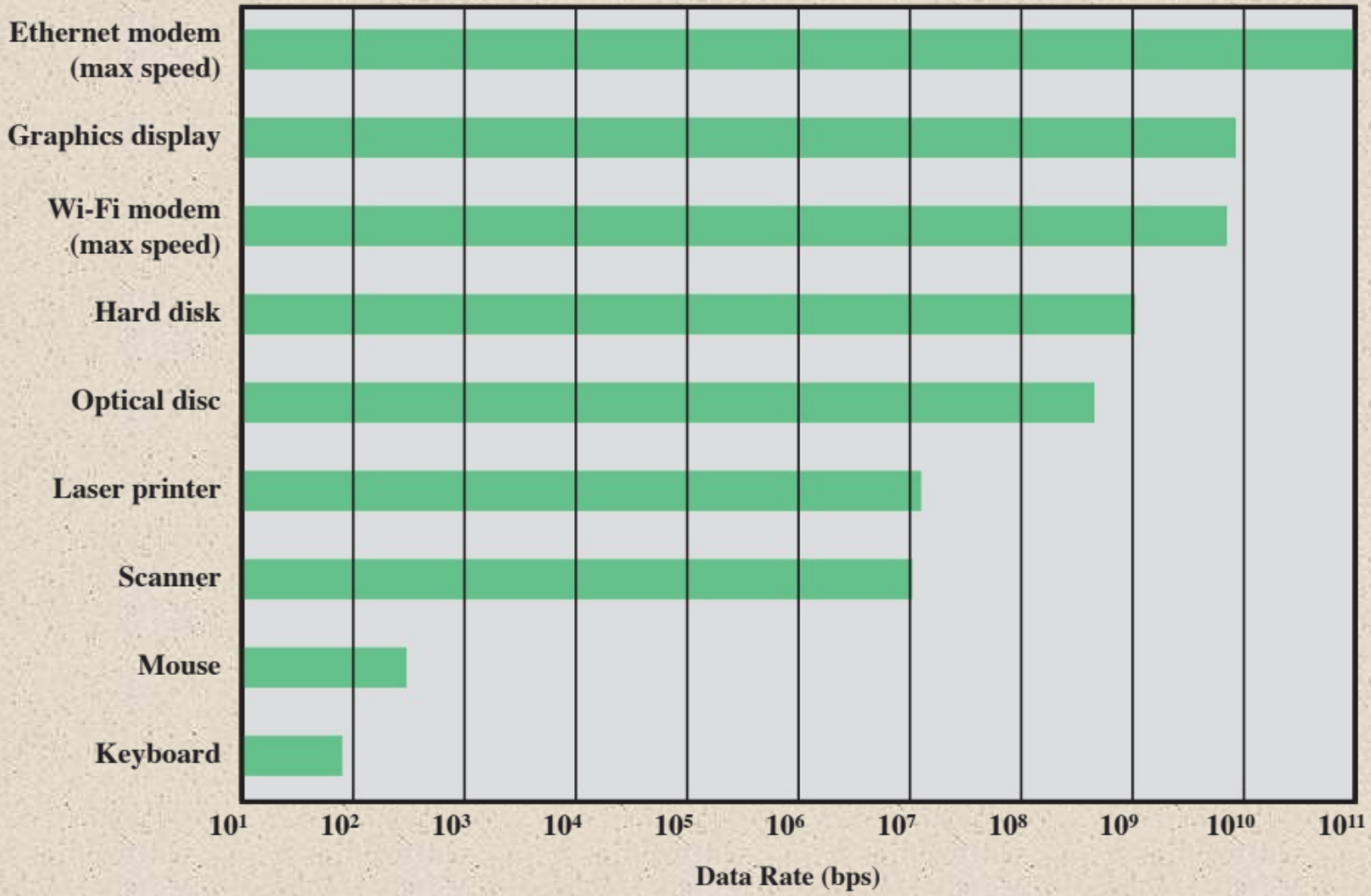
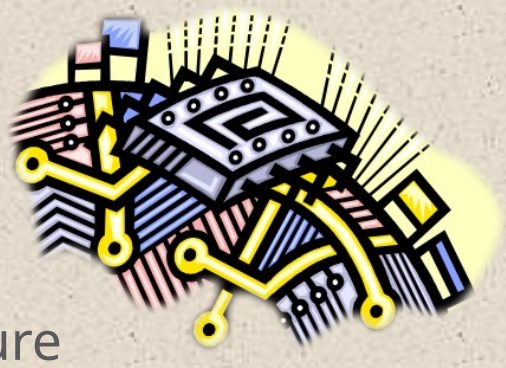


Figure 2.1 Typical I/O Device Data Rates

Improvements in Chip Organization and Architecture

- Increase hardware speed of processor
  - Fundamentally due to shrinking logic gate size
    - More gates, packed more tightly, increasing clock rate
    - Propagation time for signals reduced
- Increase size and speed of caches
  - Dedicating part of processor chip
    - Cache access times drop significantly
- Change processor organization and architecture
  - Increase effective speed of instruction execution
  - Parallelism



Problems with Clock Speed and Login Density

- Power
  - Power density increases with density of logic and clock speed
  - Dissipating heat
- RC delay
  - Speed at which electrons flow limited by resistance and capacitance of metal wires connecting them
  - Delay increases as the RC product increases
  - As components on the chip decrease in size, the wire interconnects become thinner, increasing resistance
  - Also, the wires are closer together, increasing capacitance
- Memory latency
  - Memory speeds lag processor speeds

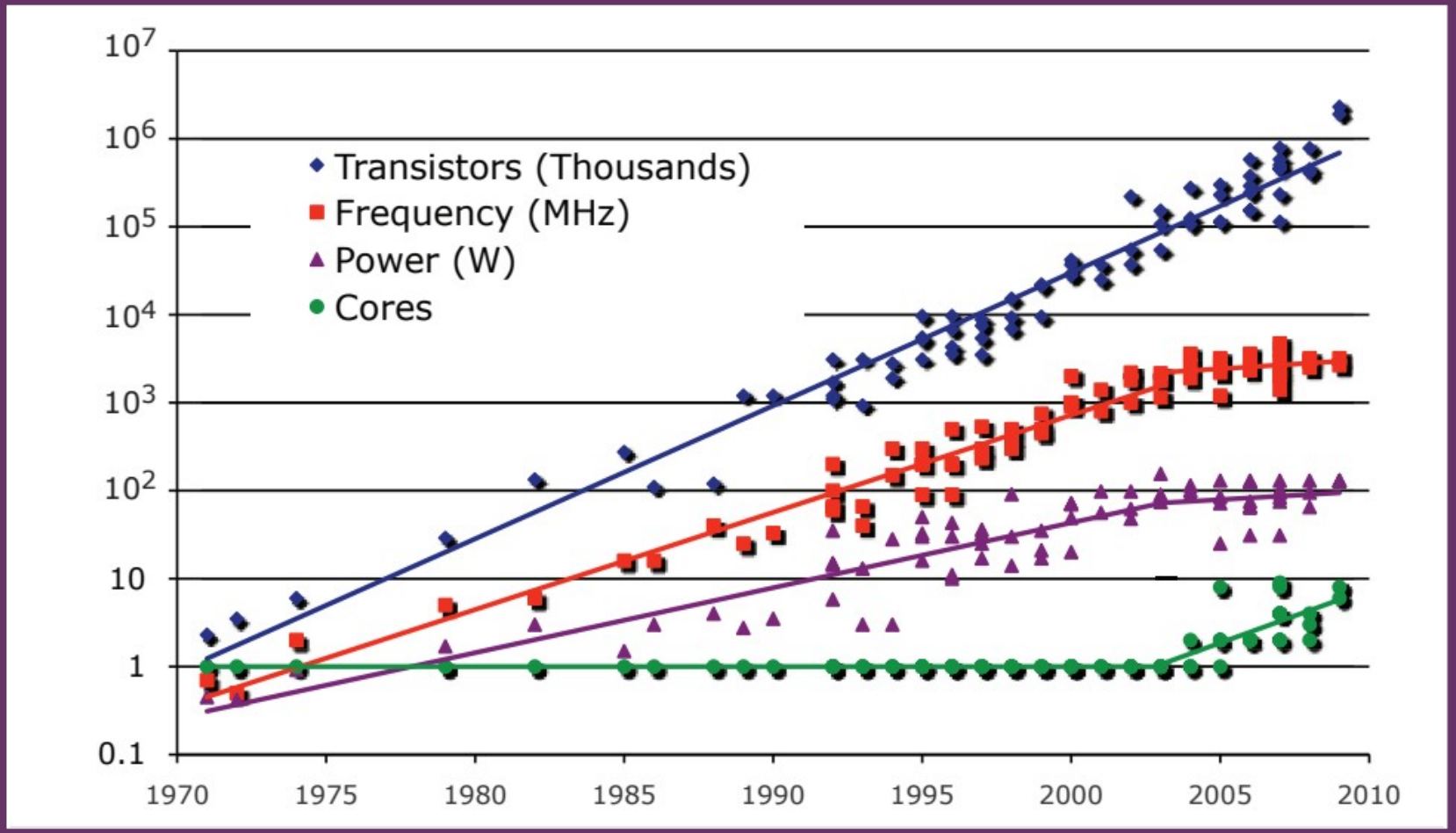
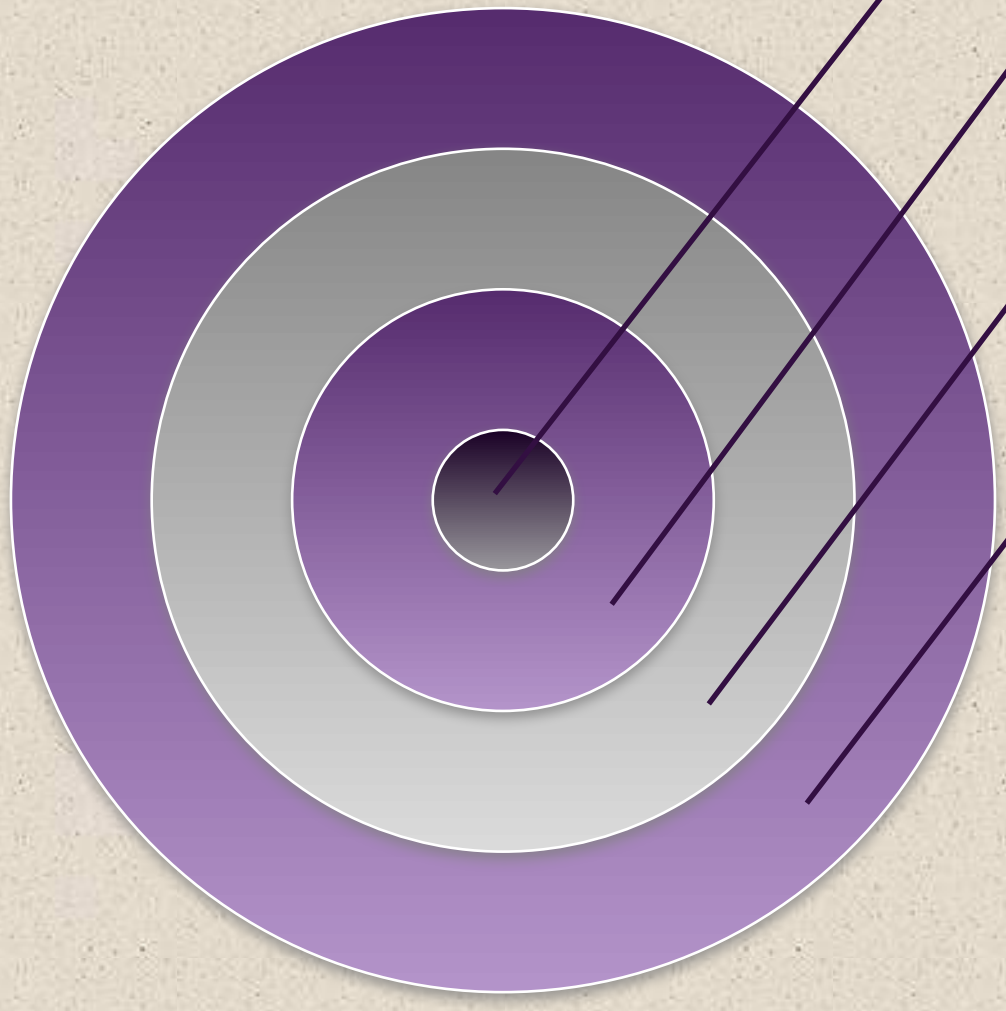


Figure 2.2 Processor Trends

Multicore



The use of multiple processors on the same chip provides the potential to increase performance without increasing the clock rate

Strategy is to use two simpler processors on the chip rather than one more complex processor

With two processors larger caches are justified

As caches became larger it made performance sense to create two and then three levels of cache on a chip

Many Integrated Core (MIC) Graphics Processing Unit (GPU)

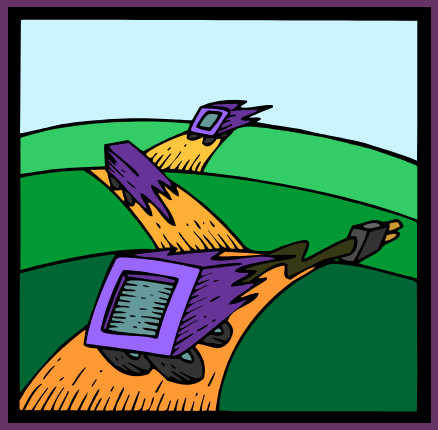
MIC

- Leap in performance as well as the challenges in developing software to exploit such a large number of cores
- The multicore and MIC strategy involves a homogeneous collection of general purpose processors on a single chip

GPU

- Core designed to perform parallel operations on graphics data
- Traditionally found on a plug-in graphics card, it is used to encode and render 2D and 3D graphics as well as process video
- Used as vector processors for a variety of applications that require repetitive computations

Amdahl's Law



- Gene Amdahl
- Deals with the potential speedup of a program using multiple processors compared to a single processor
- Illustrates the problems facing industry in the development of multi-core machines
  - Software must be adapted to a highly parallel execution environment to exploit the power of parallel processing
- Can be generalized to evaluate and design technical improvement in a computer system

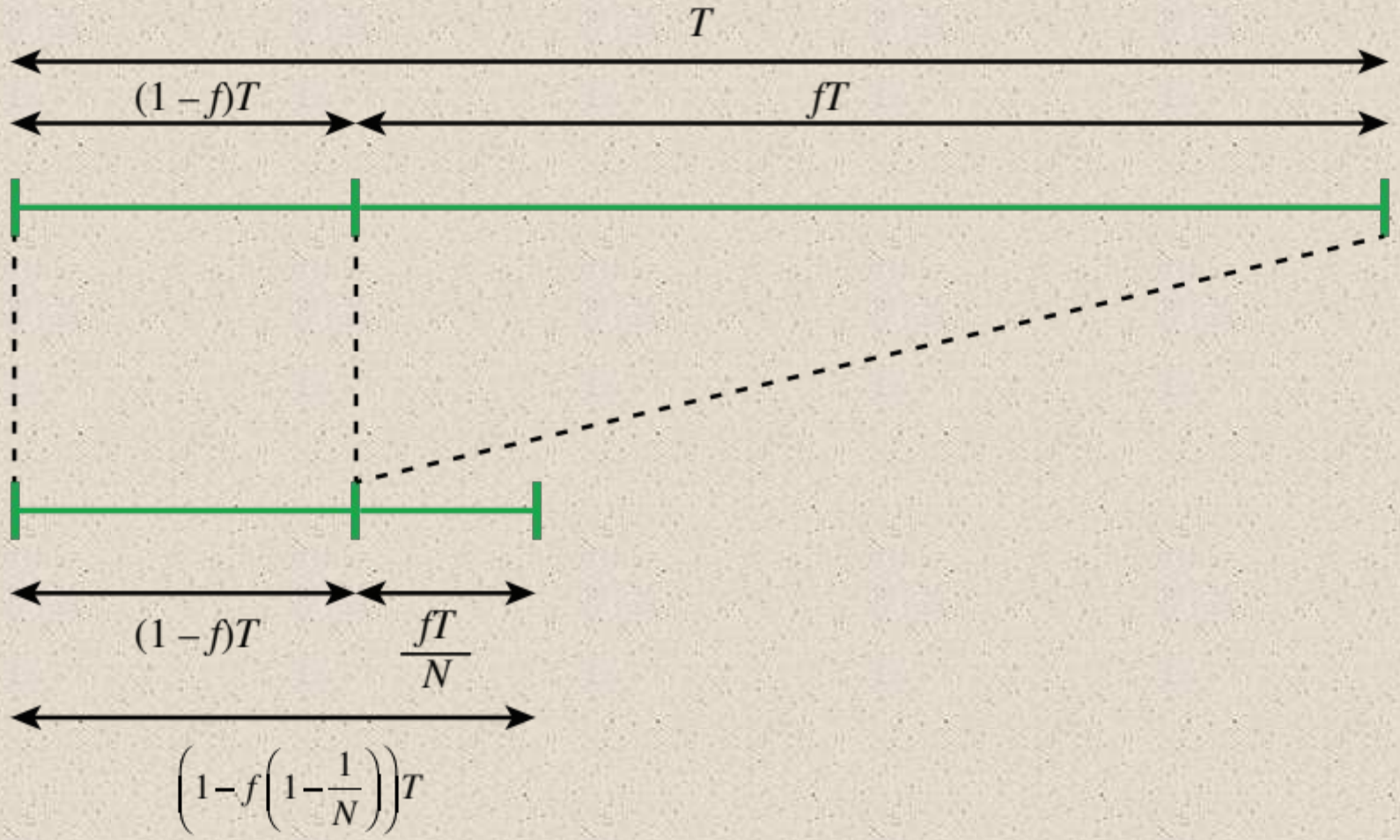


Figure 2.3 Illustration of Amdahl's Law

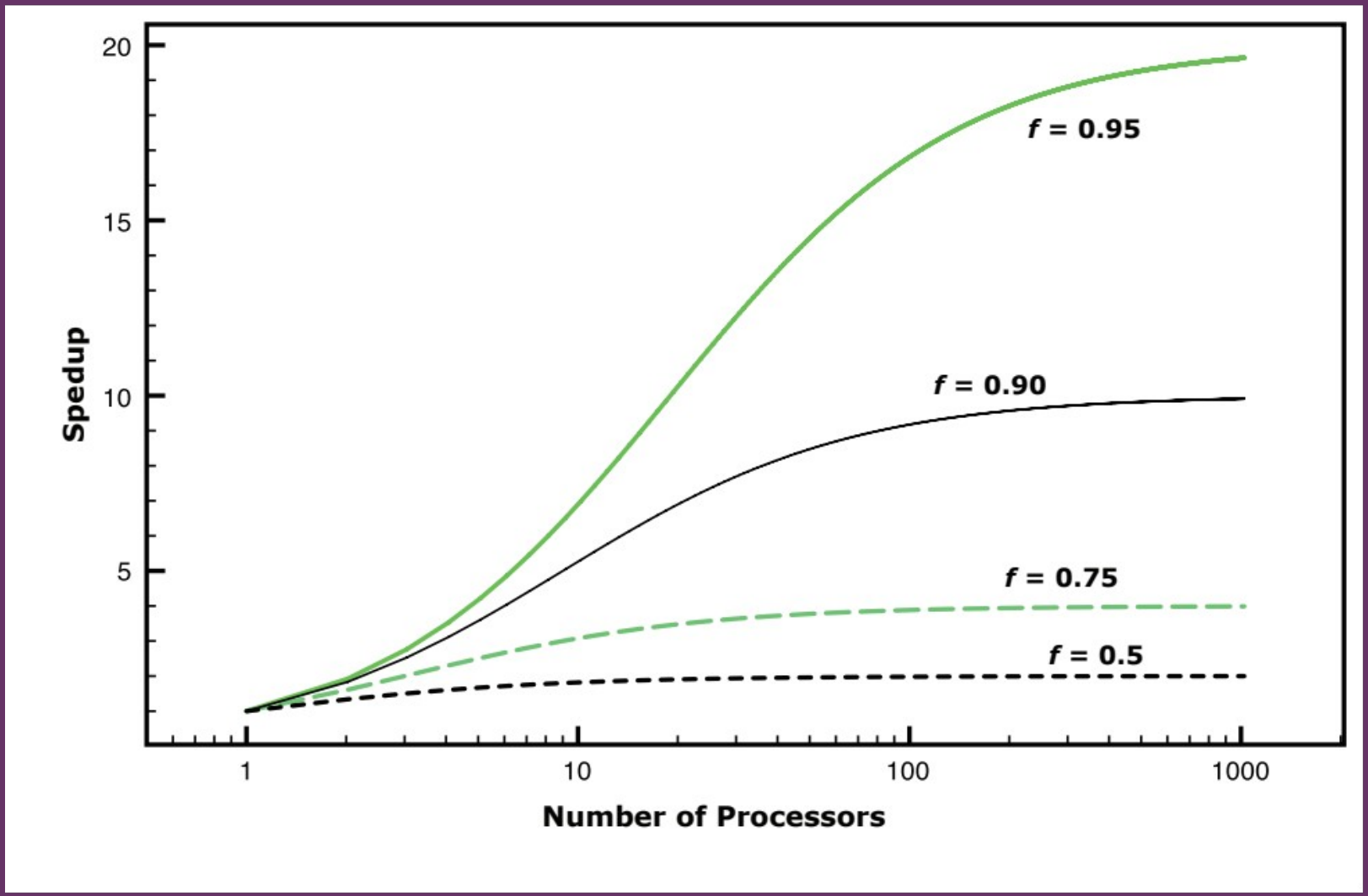
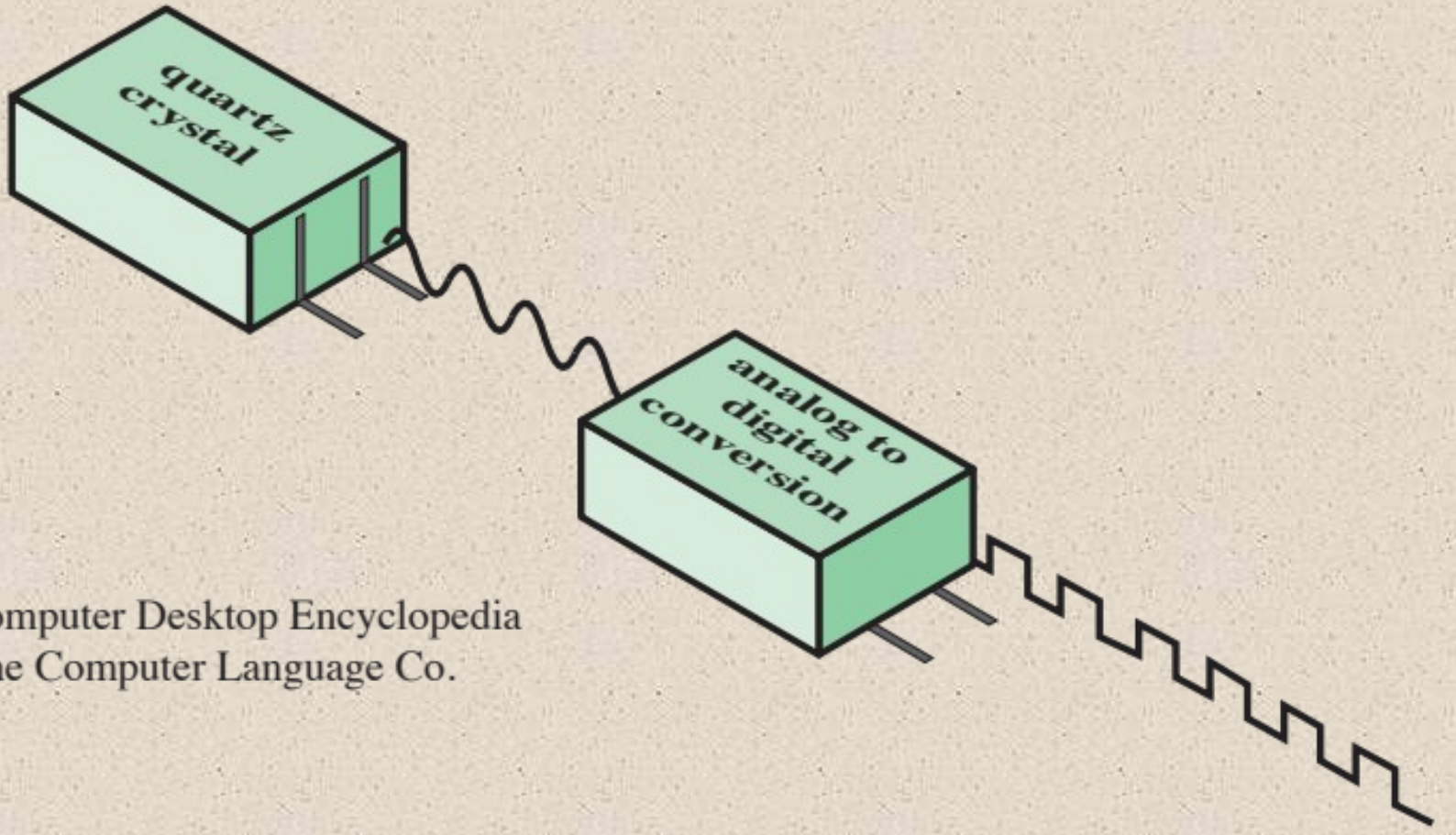


Figure 2.4 Amdahl's Law for Multiprocessors

Little's Law

- Fundamental and simple relation with broad applications
- Can be applied to almost any system that is statistically in steady state, and in which there is no leakage
- Queuing system
  - If server is idle an item is served immediately, otherwise an arriving item joins a queue
  - There can be a single queue for a single server or for multiple servers, or multiple queues with one being for each of multiple servers
- Average number of items in a queuing system equals the average rate at which items arrive multiplied by the time that an item spends in the system
  - Relationship requires very few assumptions
  - Because of its simplicity and generality it is extremely useful



From Computer Desktop Encyclopedia
1998, The Computer Language Co.

Figure 2.5 System Clock

	I_c	p	m	k	τ
Instruction set architecture	X	X			
Compiler technology	X	X	X		
Processor implementation		X			X
Cache and memory hierarchy				X	X

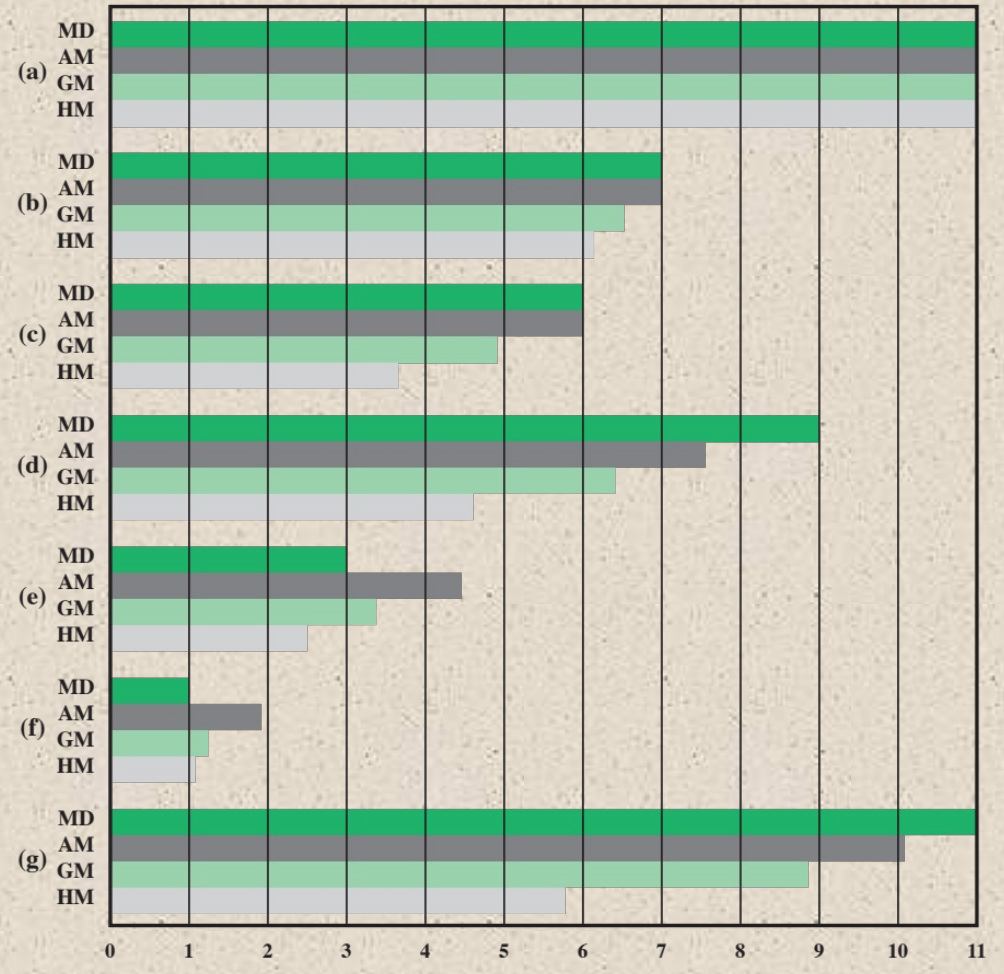
Table 2.1 Performance Factors and System Attributes

Calculating the Mean

The use of benchmarks to compare systems involves calculating the mean value of a set of data points related to execution time

The three common formulas used for calculating a mean are:

- Arithmetic
- Geometric
- Harmonic



(a) Constant (11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11)
(b) Clustered around a central value (3, 5, 6, 6, 7, 7, 7, 8, 8, 9, 11)
(c) Uniform distribution (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)
(d) Large-number bias (1, 4, 4, 7, 7, 9, 9, 10, 10, 11, 11)
(e) Small-number bias(1, 1, 2, 2, 3, 3, 5, 5, 8, 8, 11)
(f) Upper outlier (11, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)
(g) Lower outlier (1, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11)

MD = median
AM = arithmetic mean
GM = geometric mean
HM = harmonic mean

Figure 2.6 Comparison of Means on Various Data Sets (each set has a maximum data point value of 11)

- An Arithmetic Mean (AM) is an appropriate measure if the sum of all the measurements is a meaningful and interesting value

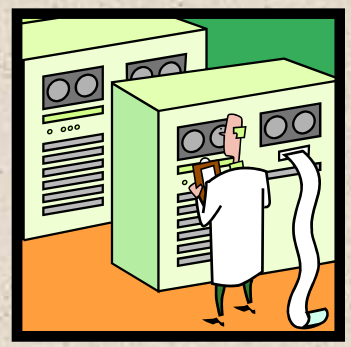
- The AM is a good candidate for comparing the execution time performance of several systems

For example, suppose we were interested in using a system for large-scale simulation studies and wanted to evaluate several alternative products. On each system we could run the simulation multiple times with different input values for each run, and then take the average execution time across all runs. The use of multiple runs with different inputs should ensure that the results are not heavily biased by some unusual feature of a given input set. The AM of all the runs is a good measure of the system's performance on simulations, and a good number to use for system comparison.

- The AM used for a time-based variable, such as program execution time, has the important property that it is directly proportional to the total time
  - If the total time doubles, the mean value doubles

Arithmetic

Mean



	Computer A time (secs)	Computer B time (secs)	Computer C time (secs)	Computer A rate (MFLOPS)	Computer B rate (MFLOPS)	Computer C rate (MFLOPS)
Program 1 (10^8 FP ops)	2.0	1.0	0.75	50	100	133.33
Program 2 (10^8 FP ops)	0.75	2.0	4.0	133.33	50	25
Total execution time	2.75	3.0	4.75			
Arithmetic mean of times	1.38	1.5	2.38			
Inverse of total execution time (1/sec)	0.36	0.33	0.21			
Arithmetic mean of rates				91.67	75.00	79.17
Harmonic mean of rates				72.72	66.67	42.11

Table 2.2

A Comparison of Arithmetic and Harmonic Means for Rates

Table 2.3 A Comparison of Arithmetic and Geometric Means for Normalized Results

(a) Results normalized to Computer A

	Computer A time	Computer B time	Computer C time
Program 1	2.0 (1.0)	1.0 (0.5)	0.75 (0.38)
Program 2	0.75 (1.0)	2.0 (2.67)	4.0 (5.33)
Total execution time	2.75	3.0	4.75
Arithmetic mean of normalized times	1.00	1.58	2.85
Geometric mean of normalized times	1.00	1.15	1.41

(b) Results normalized to Computer B

	Computer A time	Computer B time	Computer C time
Program 1	2.0 (2.0)	1.0 (1.0)	0.75 (0.75)
Program 2	0.75 (0.38)	2.0 (1.0)	4.0 (2.0)
Total execution time	2.75	3.0	4.75
Arithmetic mean of normalized times	1.19	1.00	1.38
Geometric mean of normalized times	0.87	1.00	1.22

Table 2.4 Another Comparison of Arithmetic and Geometric Means for Normalized Results

(a) Results normalized to Computer A

	Computer A time	Computer B time	Computer C time
Program 1	2.0 (1.0)	1.0 (0.5)	0.20 (0.1)
Program 2	0.4 (1.0)	2.0 (5.0)	4.0 (10)
Total execution time	2.4	3.00	4.2
Arithmetic mean of normalized times	1.00	2.75	5.05
Geometric mean of normalized times	1.00	1.58	1.00

(b) Results normalized to Computer B

	Computer A time	Computer B time	Computer C time
Program 1	2.0 (2.0)	1.0 (1.0)	0.20 (0.2)
Program 2	0.4 (0.2)	2.0 (1.0)	4.0 (2)
Total execution time	2.4	3.00	4.2
Arithmetic mean of normalized times	1.10	1.00	1.10
Geometric mean of normalized times	0.63	1.00	0.63

Benchmark Principles

- Desirable characteristics of a benchmark program:

1. It is written in a high-level language, making it portable across different machines
2. It is representative of a particular kind of programming domain or paradigm, such as systems programming, numerical programming, or commercial programming
3. It can be measured easily
4. It has wide distribution

System Performance Evaluation Corporation (SPEC)

- Benchmark suite
  - A collection of programs, defined in a high-level language
  - Together attempt to provide a representative test of a computer in a particular application or system programming area
- SPEC
  - An industry consortium
  - Defines and maintains the best known collection of benchmark suites aimed at evaluating computer systems
  - Performance measurements are widely used for comparison and research purposes

SPEC

CPU2006

- Best known SPEC benchmark suite
- Industry standard suite for processor intensive applications
- Appropriate for measuring performance for applications that spend most of their time doing computation rather than I/O
- Consists of 17 floating point programs written in C, C++, and Fortran and 12 integer programs written in C and C++
- Suite contains over 3 million lines of code
- Fifth generation of processor intensive suites from SPEC

Benchmark	Reference time (hours)	Instr count (billion)	Language	Application Area	Brief Description
400.perlbench	2.71	2,378	C	Programming Language	PERL programming language interpreter, applied to a set of three programs.
401.bzip2	2.68	2,472	C	Compression	General-purpose data compression with most work done in memory, rather than doing I/O.
403.gcc	2.24	1,064	C	C Compiler	Based on gcc Version 3.2, generates code for Opteron.
429.mcf	2.53	327	C	Combinatorial Optimization	Vehicle scheduling algorithm.
445.gobmk	2.91	1,603	C	Artificial Intelligence	Plays the game of Go, a simply described but deeply complex game.
456.hmmer	2.59	3,363	C	Search Gene Sequence	Protein sequence analysis using profile hidden Markov models.
458.sjeng	3.36	2,383	C	Artificial Intelligence	A highly ranked chess program that also plays several chess variants.
462.libquantum	5.76	3,555	C	Physics / Quantum Computing	Simulates a quantum computer, running Shor's polynomial-time factorization algorithm.
464.h264ref	6.15	3,731	C	Video Compression	H.264/AVC (Advanced Video Coding) Video compression.
471.omnetpp	1.74	687	C++	Discrete Event Simulation	Uses the OMNet++ discrete event simulator to model a large Ethernet campus network.
473.astar	1.95	1,200	C++	Path-finding Algorithms	Pathfinding library for 2D maps.
483.xalancbmk	1.92	1,184	C++	XML Processing	A modified version of Xalan-C++, which transforms XML documents to other document types.

Table 2.5

SPEC CPU2006 Integer Benchmarks

(Table can be found on page 69 in the textbook.)

Benchmark	Reference time (hours)	Instr count (billion)	Language	Application Area	Brief Description
410.bwaves	3.78	1,176	Fortran	Fluid Dynamics	Computes 3D transonic transient laminar viscous flow.
416.gamess	5.44	5,189	Fortran	Quantum Chemistry	Quantum chemical computations.
433.milc	2.55	937	C	Physics / Quantum Chromodynamics	Simulates behavior of quarks and gluons
434.zeusmp	2.53	1,566	Fortran	Physics / CFD	Computational fluid dynamics simulation of astrophysical phenomena.
435.gromacs	1.98	1,958	C, Fortran	Biochemistry / Molecular Dynamics	Simulate Newtonian equations of motion for hundreds to millions of particles.
436.cactusADM	3.32	1,376	C, Fortran	Physics / General Relativity	Solves the Einstein evolution equations.
437.leslie3d	2.61	1,273	Fortran	Fluid Dynamics	Model fuel injection flows.
444.namd	2.23	2,483	C++	Biology / Molecular Dynamics	Simulates large biomolecular systems.
447.dealII	3.18	2,323	C++	Finite Element Analysis	Program library targeted at adaptive finite elements and error estimation.
450.soplex	2.32	703	C++	Linear Programming, Optimization	Test cases include railroad planning and military airlift models.
453.povray	1.48	940	C++	Image Ray-tracing	3D Image rendering.
454.calculix	2.29	3,04`	C, Fortran	Structural Mechanics	Finite element code for linear and nonlinear 3D structural applications.
459.GemsFDTD	2.95	1,320	Fortran	Computational Electromagnetics	Solves the Maxwell equations in 3D.
465.tonto	2.73	2,392	Fortran	Quantum Chemistry	Quantum chemistry package, adapted for crystallographic tasks.
470.lbm	3.82	1,500	C	Fluid Dynamics	Simulates incompressible fluids in 3D.
481.wrf	3.10	1,684	C, Fortran	Weather	Weather forecasting model
482.sphinx3	5.41	2,472	C	Speech recognition	Speech recognition software.

Table 2.6

SPEC CPU2006 Floating-Point Benchmarks

(Table can be found on page 70 in the textbook.)

Terms Used in SPEC Documentation

- Benchmark
  - A program written in a high-level language that can be compiled and executed on any computer that implements the compiler
- System under test
  - This is the system to be evaluated
- Reference machine
  - This is a system used by SPEC to establish a baseline performance for all benchmarks
    - Each benchmark is run and measured on this machine to establish a reference time for that benchmark
- Base metric
  - These are required for all reported results and have strict guidelines for compilation
- Peak metric
  - This enables users to attempt to optimize system performance by optimizing the compiler output
- Speed metric
  - This is simply a measurement of the time it takes to execute a compiled benchmark
    - Used for comparing the ability of a computer to complete single tasks
- Rate metric
  - This is a measurement of how many tasks a computer can accomplish in a certain amount of time
    - This is called a throughput, capacity, or rate measure
    - Allows the system under test to execute simultaneous tasks to take advantage of multiple processors

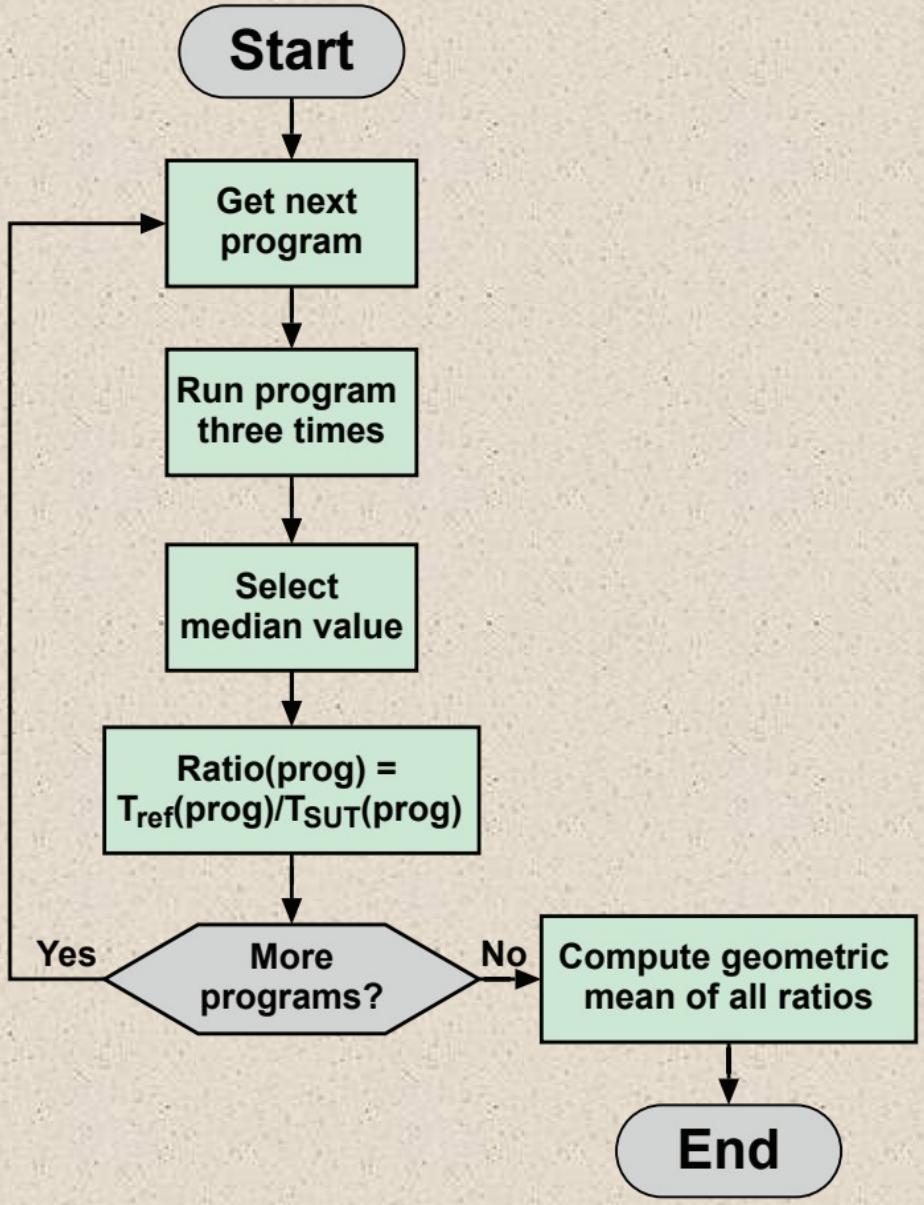


Figure 2.7 SPEC Evaluation Flowchart

Table 2.7 Some SPEC CINT2006 Results

(a) Sun Blade 1000

Benchmark	Execution time	Execution time	Execution time	Reference time	Ratio
400.perlbench	**3077**	3076	3080	9770	3.18
401.bzip2	**3260**	3263	3260	9650	2.96
403.gcc	2711	2701	**2702**	8050	2.98
429.mcf	2356	**2331**	2301	9120	3.91
445.gobmk	3319	**3310**	3308	10490	3.17
456.hmmer	2586	**2587**	2601	9330	3.61
458.sjeng	3452	**3449**	3449	12100	3.51
462.libquantum	**10318**	10319	10273	20720	2.01
464.h264ref	5246	5290	**5259**	22130	4.21
471.omnetpp	2565	**2572**	2582	6250	2.43
473.astar	2522	**2554**	2565	7020	2.75
483.xalancbmk	2014	2018	**2018**	6900	3.42

(b) Sun Blade X6250

Benchmark	Execution time	Execution time	Execution time	Reference time	Ratio	Rate
400.perlbench	497	497	**497**	9770	19.66	78.63
401.bzip2	613	614	**613**	9650	15.74	62.97
403.gcc	529	**529**	529	8050	15.22	60.87
429.mcf	**472**	472	473	9120	19.32	77.29
445.gobmk	637	637	**637**	10490	16.47	65.87
456.hmmer	446	446	**446**	9330	20.92	83.68
458.sjeng	**631**	632	630	12100	19.18	76.70
462.libquantum	**614**	614	614	20720	33.75	134.98
464.h264ref	830	**830**	830	22130	26.66	106.65
471.omnetpp	619	620	**619**	6250	10.10	40.39
473.astar	580	580	**580**	7020	12.10	48.41
483.xalancbmk	**422**	422	422	6900	16.35	65.40

Summary

Chapter 2

Performance Issues

- Designing for performance
  - Microprocessor speed
  - Performance balance
  - Improvements in chip organization and architecture
- Multicore
- MICs
- GPGPUs
- Amdahl's Law
- Little's Law

- Basic measures of computer performance
  - Clock speed
  - Instruction execution rate
- Calculating the mean
  - Arithmetic mean
  - Harmonic mean
  - Geometric mean
- Benchmark principles
- SPEC benchmarks

William Stallings
Computer Organization and Architecture
10th Edition

Chapter 3

A Top-Level View of Computer Function and Interconnection

Computer Components

- Contemporary computer designs are based on concepts developed by John von Neumann at the Institute for Advanced Studies, Princeton
- Referred to as the *von Neumann architecture* and is based on three key concepts:
  - Data and instructions are stored in a single read-write memory
  - The contents of this memory are addressable by location, without regard to the type of data contained there
  - Execution occurs in a sequential fashion (unless explicitly modified) from one instruction to the next
- *Hardwired program*
  - The result of the process of connecting the various components in the desired configuration

Hardware and Software Approaches

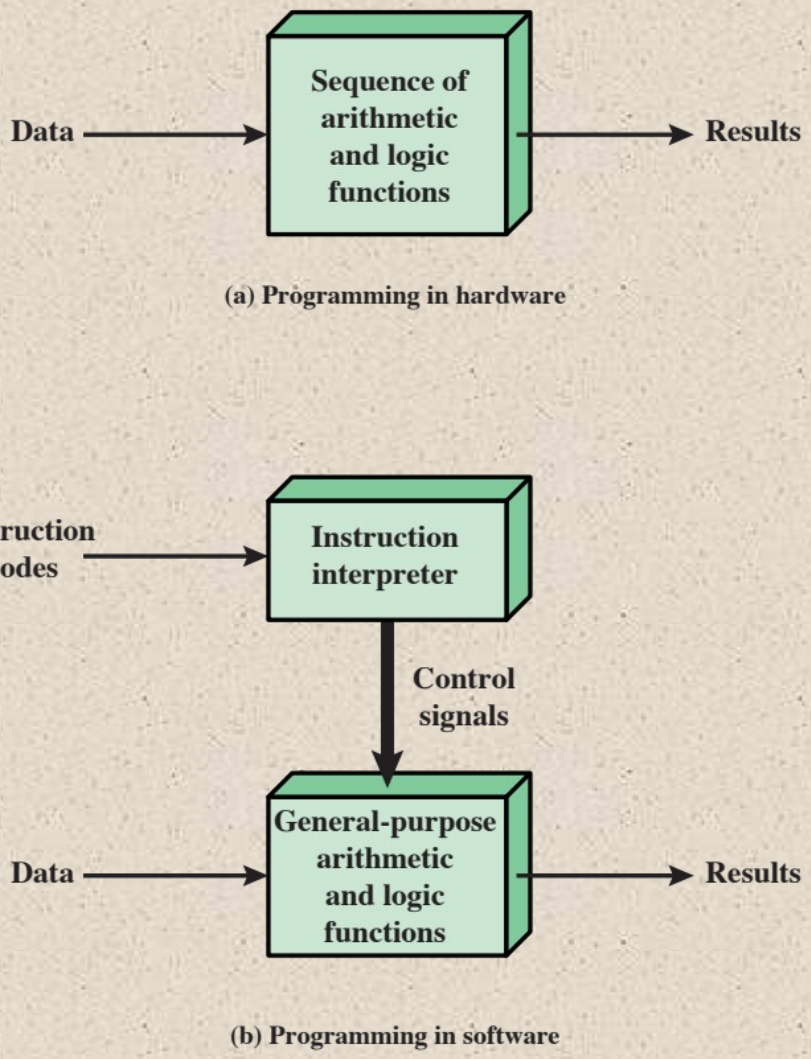


Figure 3.1 Hardware and Software Approaches

Software

- A sequence of codes or instructions
- Part of the hardware interprets each instruction and generates control signals
- Provide a new sequence of codes for each new program instead of rewiring the hardware

Software

Major components:

- CPU
  - Instruction interpreter
  - Module of general-purpose arithmetic and logic functions
- I/O Components
  - Input module
    - Contains basic components for accepting data and instructions and converting them into an internal form of signals usable by the system
  - Output module
    - Means of reporting results

I/O Components

Memory address register (MAR)

- Specifies the address in memory for the next read or write

Memory buffer register (MBR)

- Contains the data to be written into memory or receives the data read from memory

I/O address register (I/OAR)

- Specifies a particular I/O device

I/O buffer register (I/OBR)

- Used for the exchange of data between an I/O module and the CPU

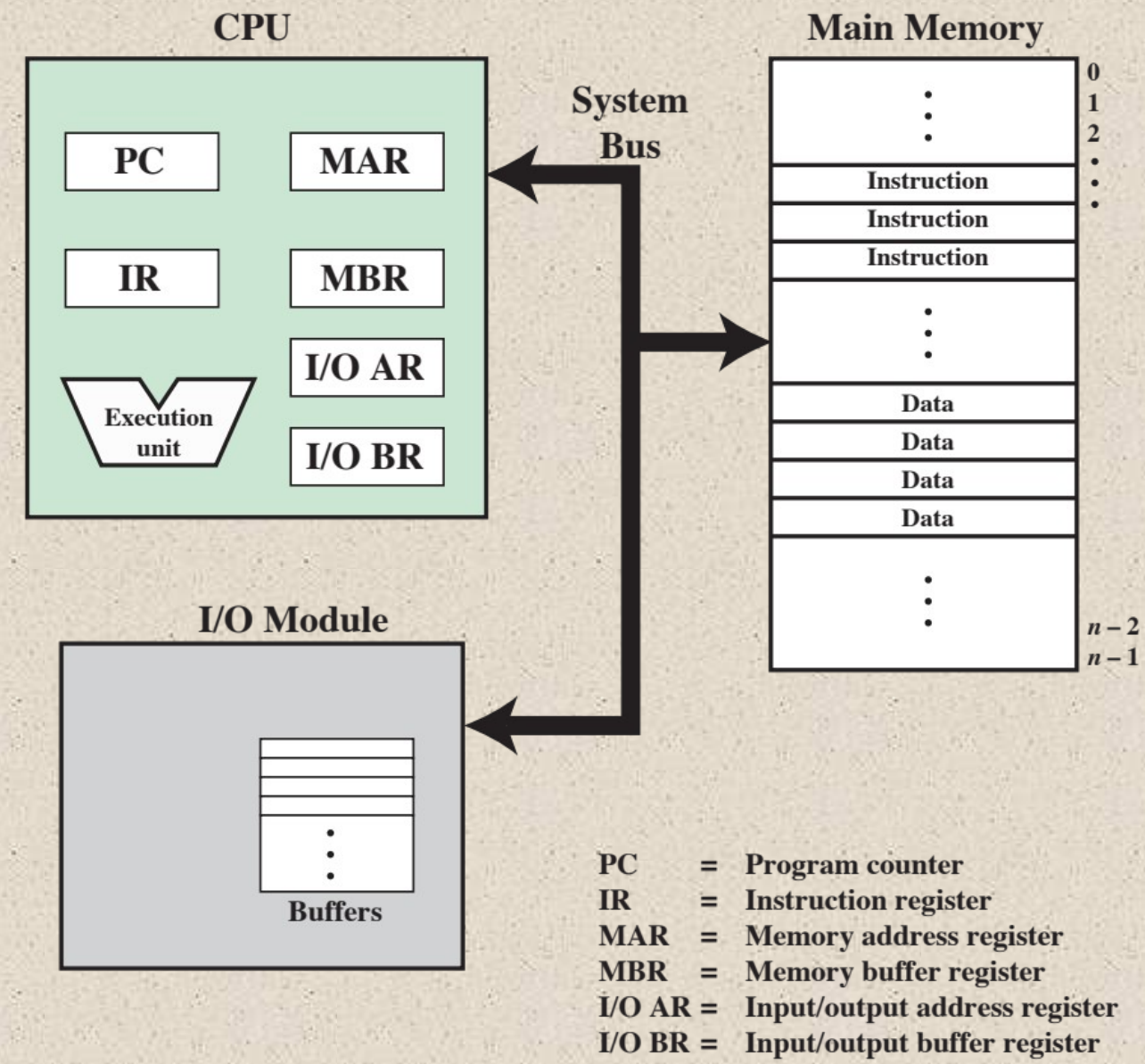


Figure 3.2 Computer Components: Top-Level View

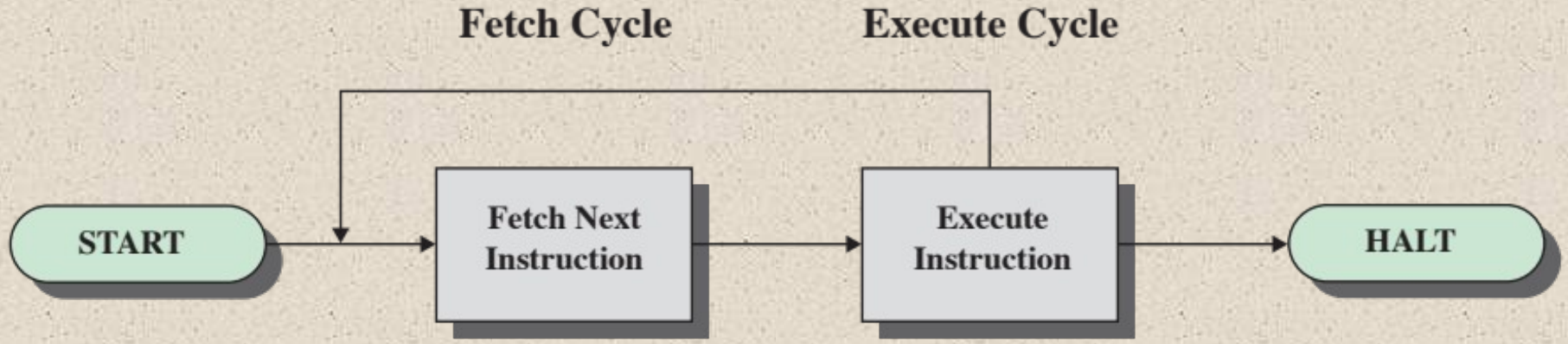
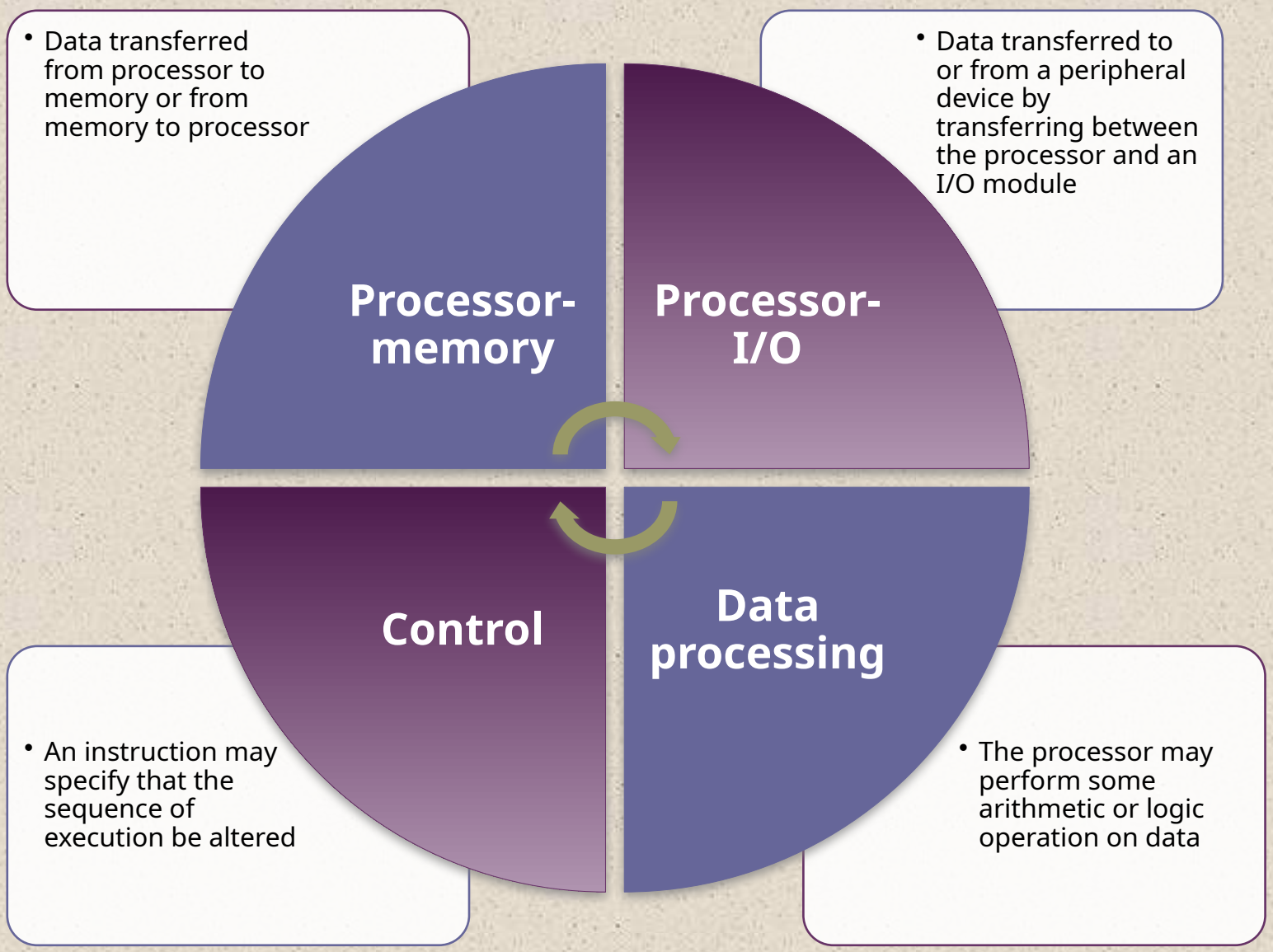


Figure 3.3 Basic Instruction Cycle

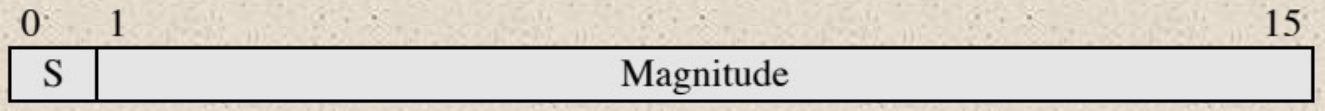
Fetch Cycle

- At the beginning of each instruction cycle the processor fetches an instruction from memory
- The program counter (PC) holds the address of the instruction to be fetched next
- The processor increments the PC after each instruction fetch so that it will fetch the next instruction in sequence
- The fetched instruction is loaded into the instruction register (IR)
- The processor interprets the instruction and performs the required action

Action Categories



(a) Instruction format



(b) Integer format

Program Counter (PC) = Address of instruction
Instruction Register (IR) = Instruction being executed
Accumulator (AC) = Temporary storage

(c) Internal CPU registers

0001 = Load AC from Memory
0010 = Store AC to Memory
0101 = Add to AC from Memory

(d) Partial list of opcodes

Figure 3.4 Characteristics of a Hypothetical Machine

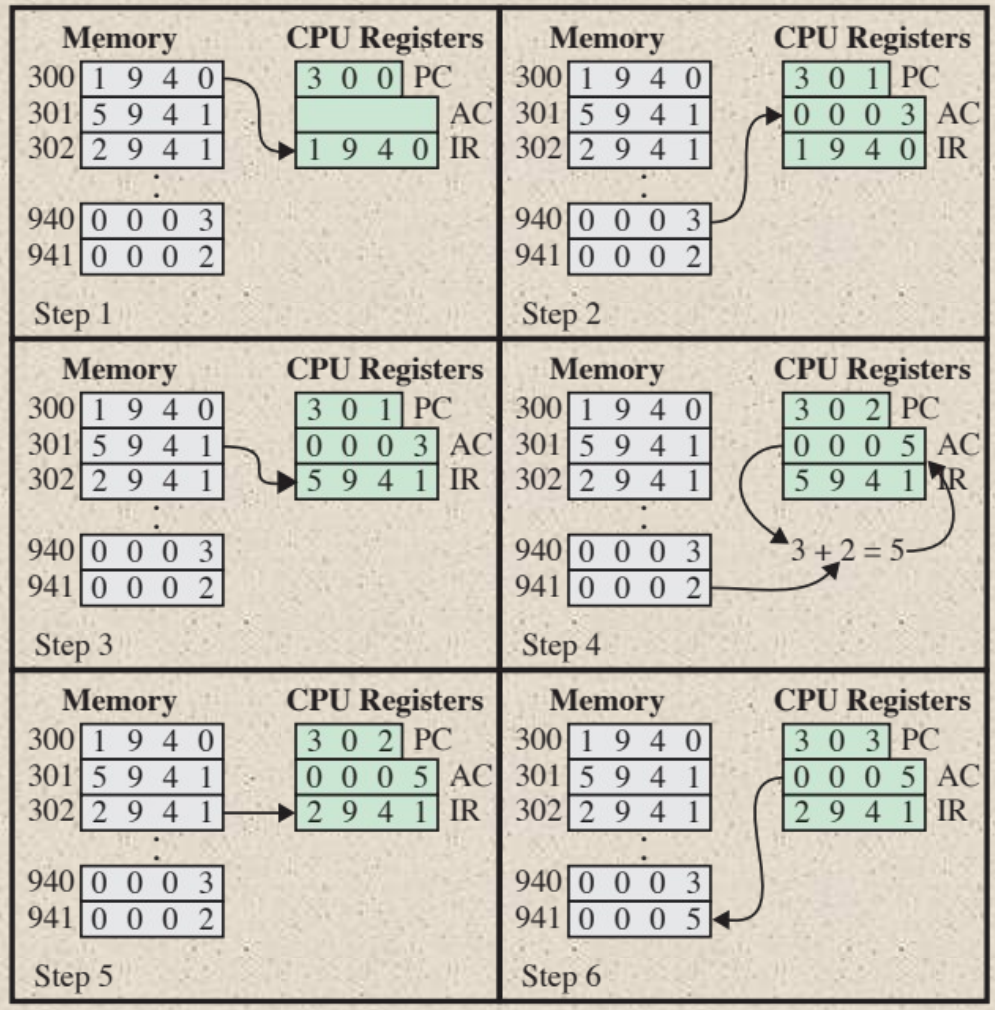


Figure 3.5 Example of Program Execution (contents of memory and registers in hexadecimal)

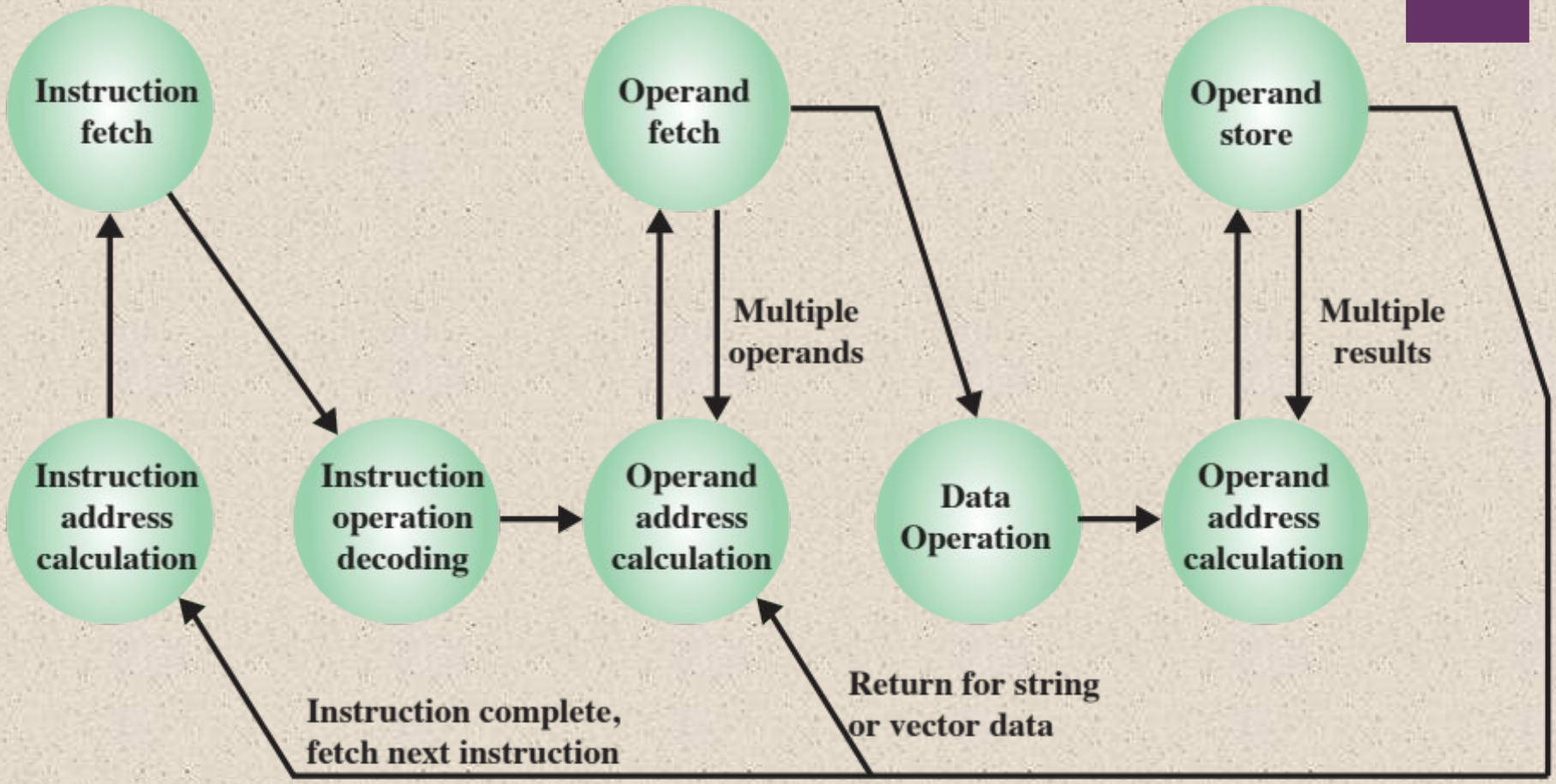


Figure 3.6 Instruction Cycle State Diagram

Program	Generated by some condition that occurs as a result of an instruction execution, such as arithmetic overflow, division by zero, attempt to execute an illegal machine instruction, or reference outside a user's allowed memory space.
Timer	Generated by a timer within the processor. This allows the operating system to perform certain functions on a regular basis.
I/O	Generated by an I/O controller, to signal normal completion of an operation, request service from the processor, or to signal a variety of error conditions.
Hardware failure	Generated by a failure such as power failure or memory parity error.

Table 3.1

Classes of Interrupts

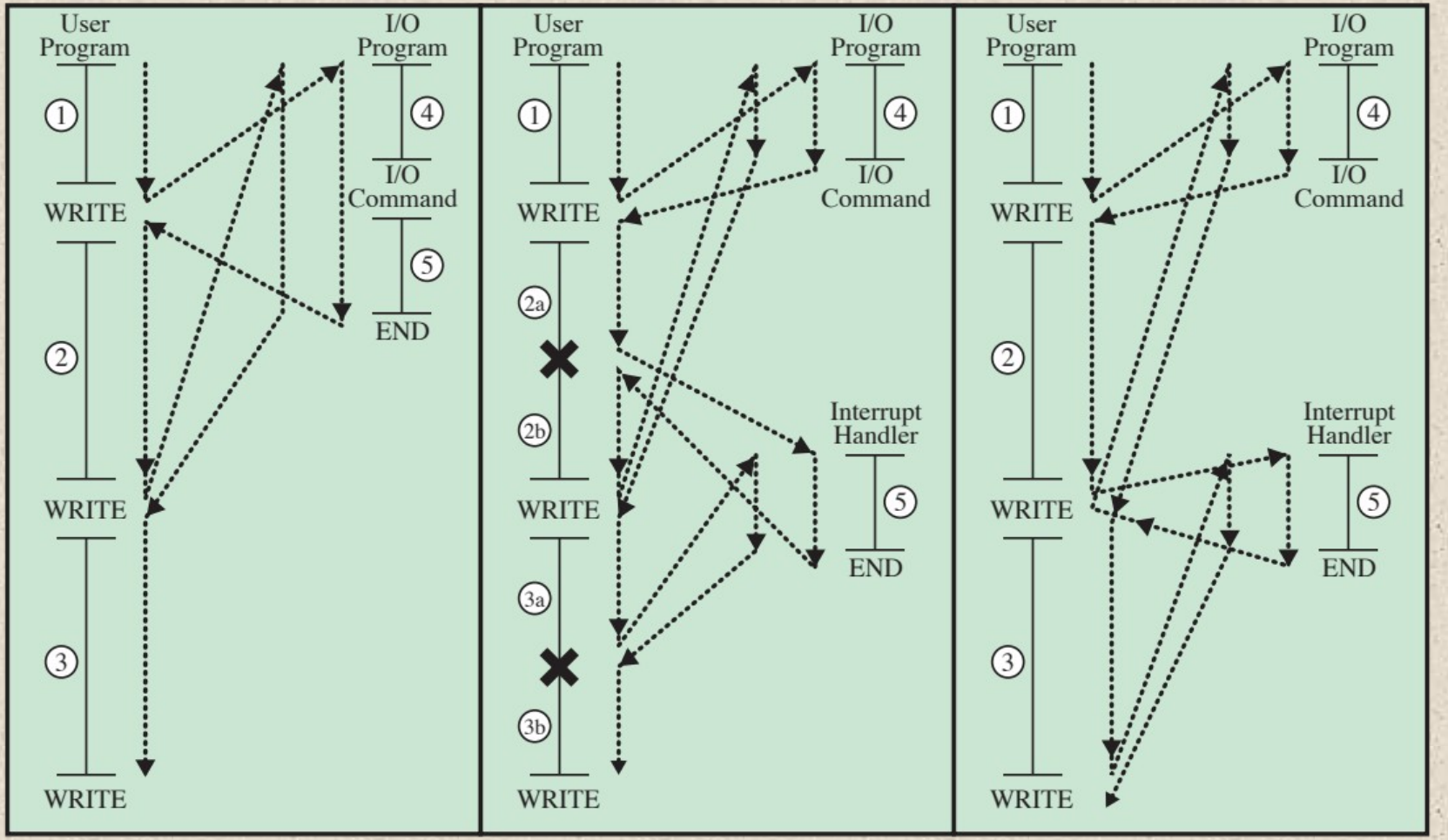


Figure 3.7 Program Flow of Control Without and With Interrupts

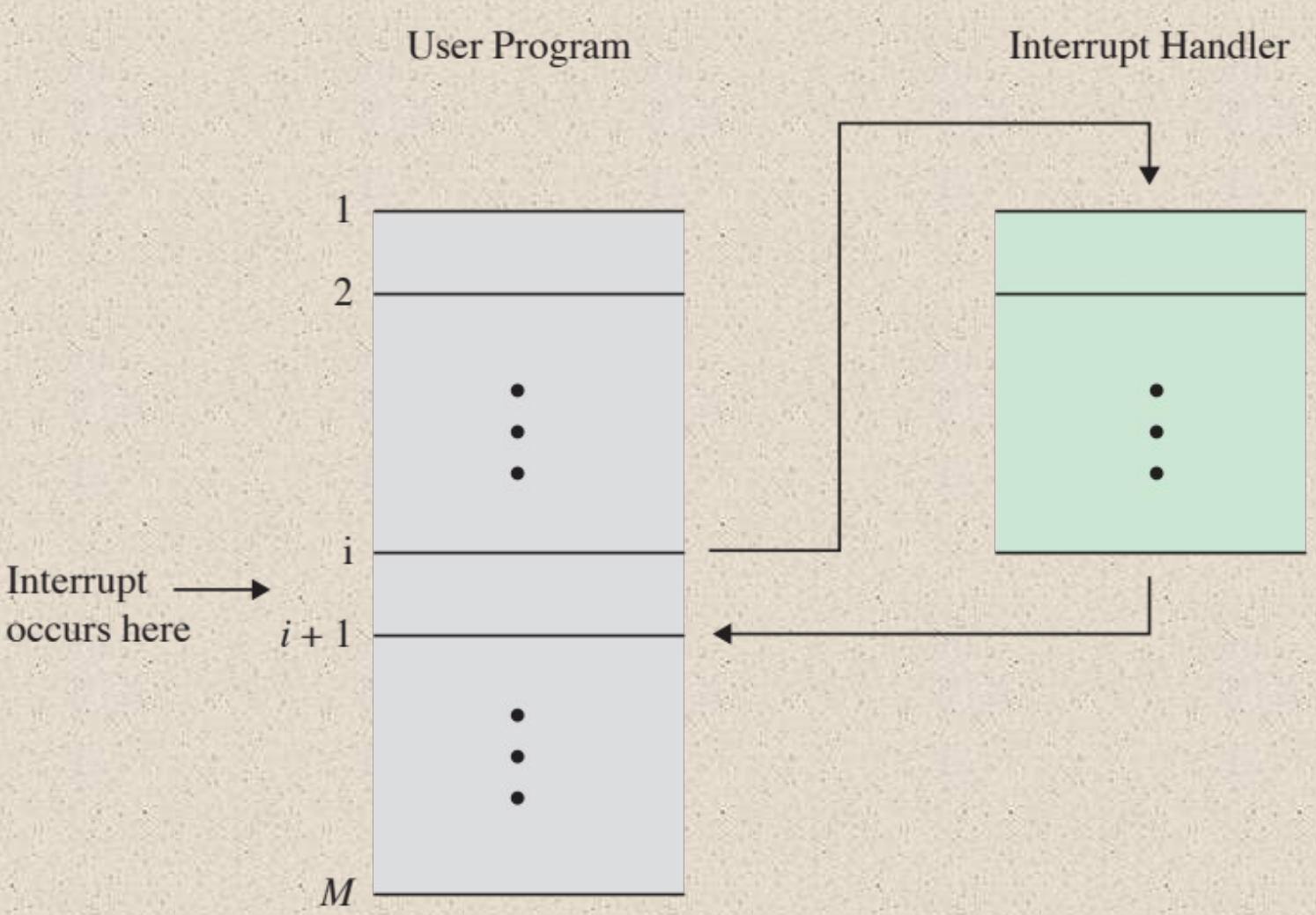


Figure 3.8 Transfer of Control via Interrupts

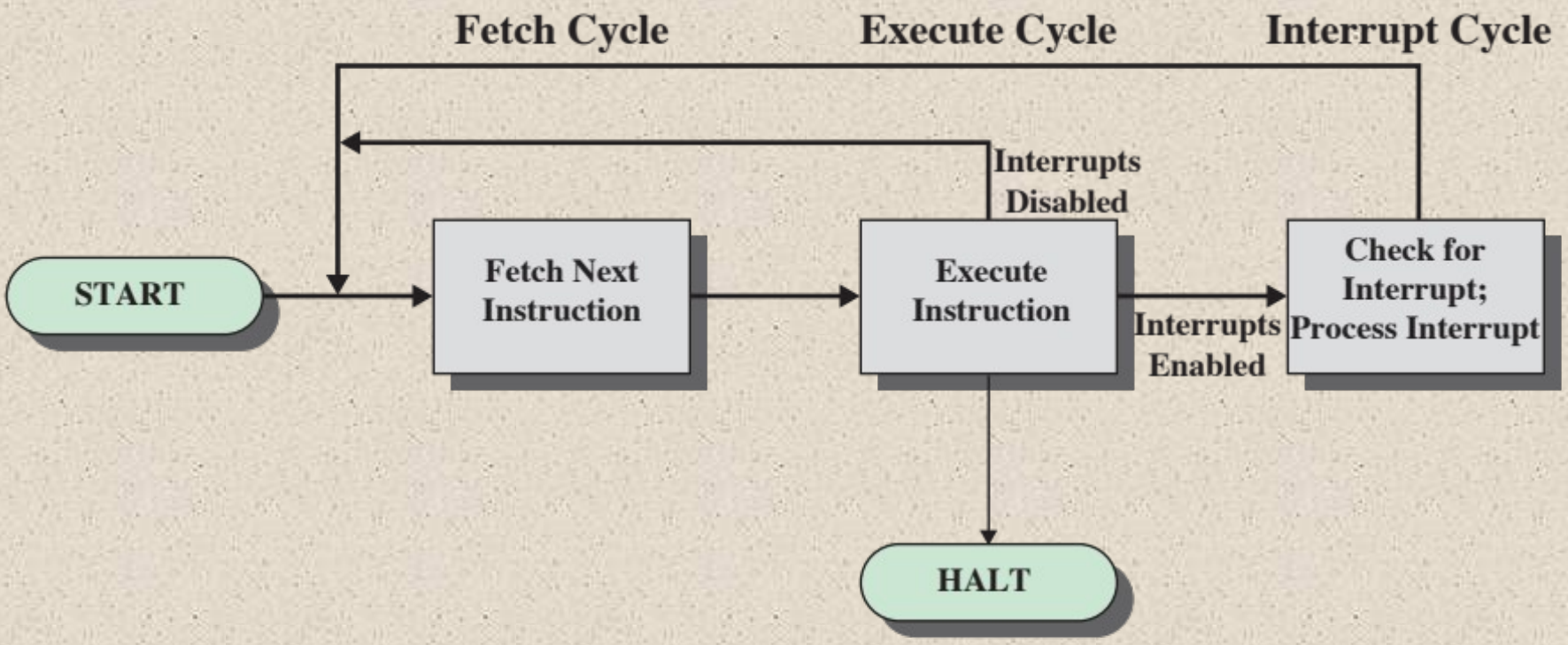


Figure 3.9 Instruction Cycle with Interrupts

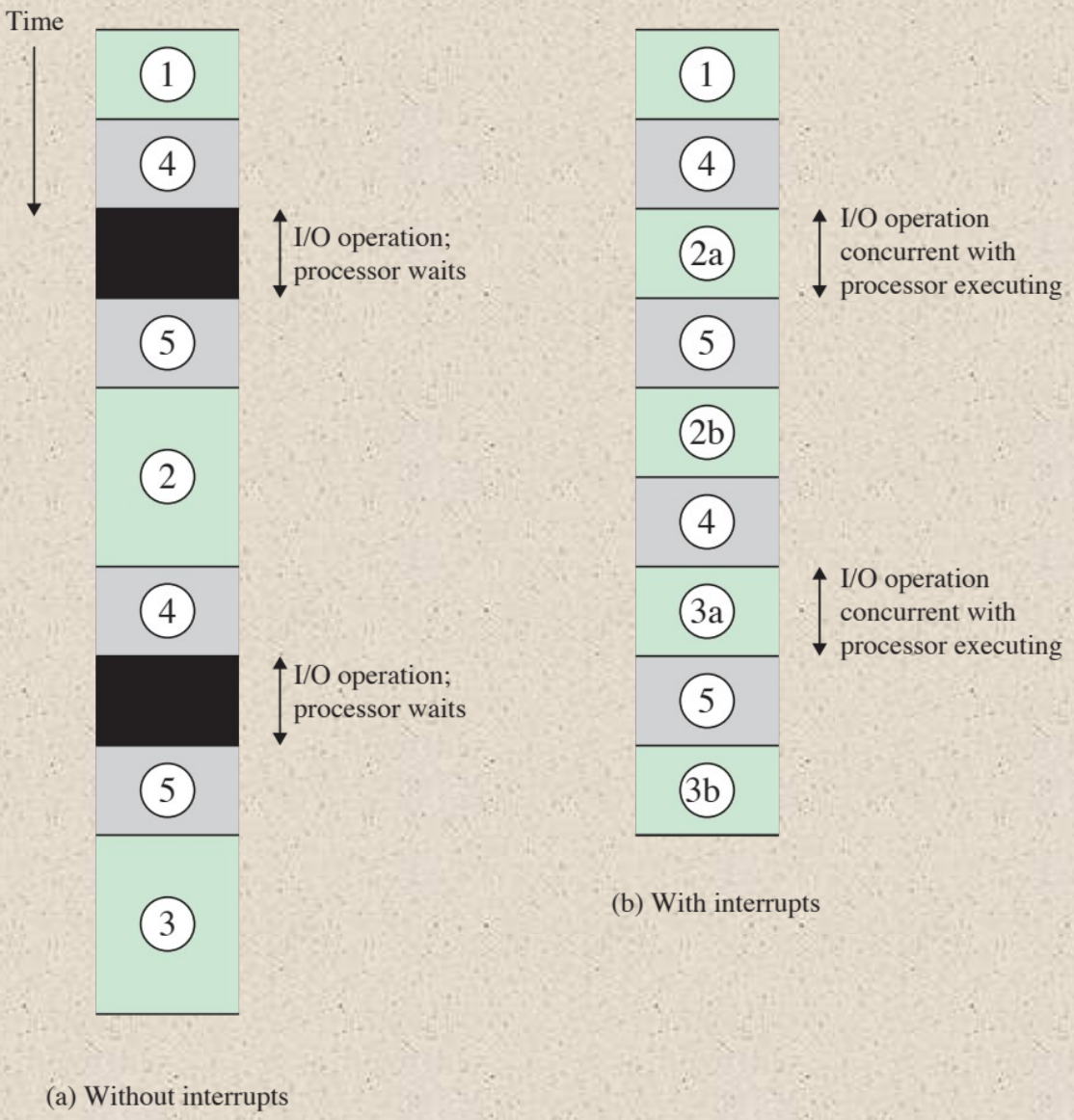


Figure 3.10 Program Timing: Short I/O Wait

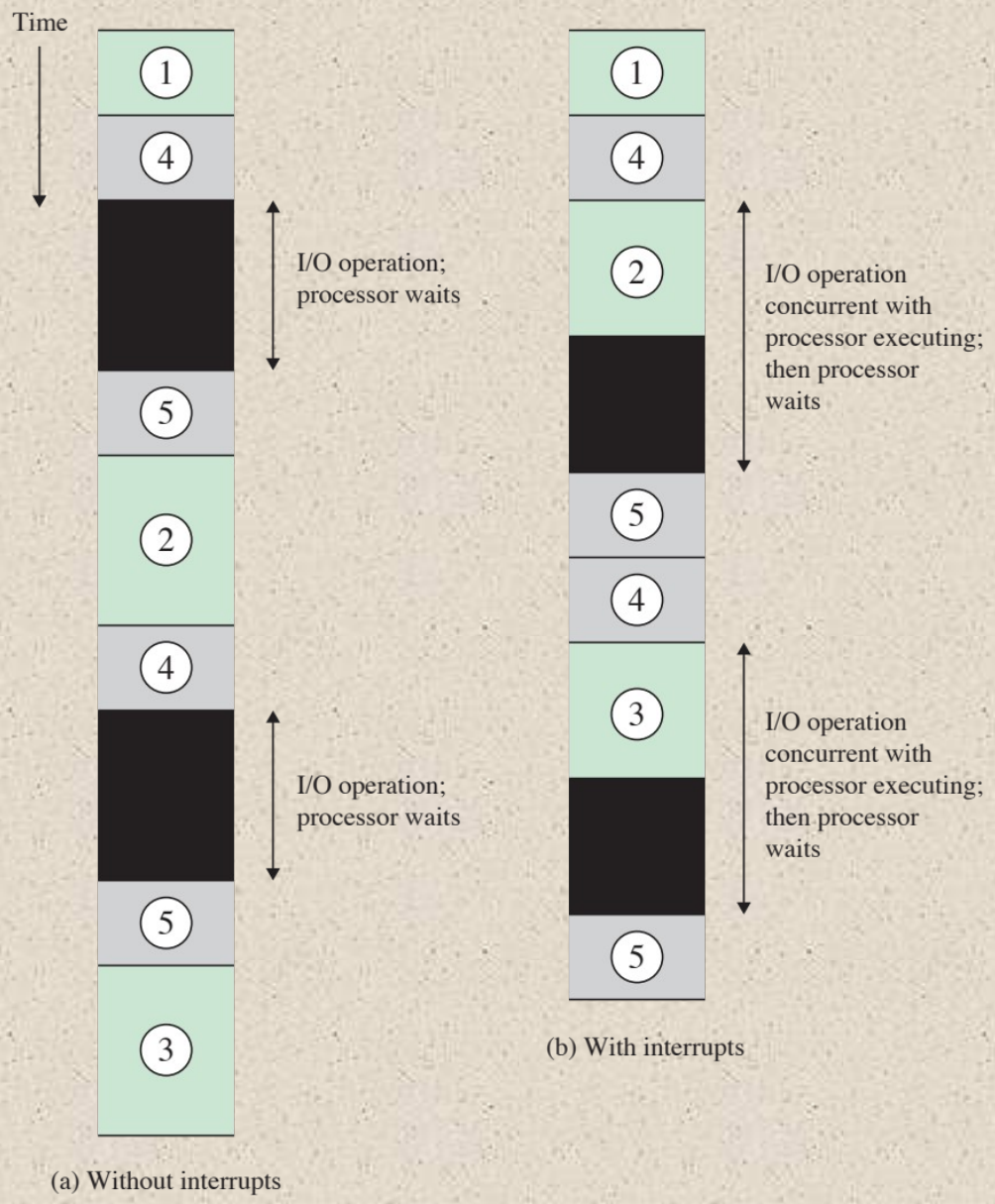


Figure 3.11 Program Timing: Long I/O Wait

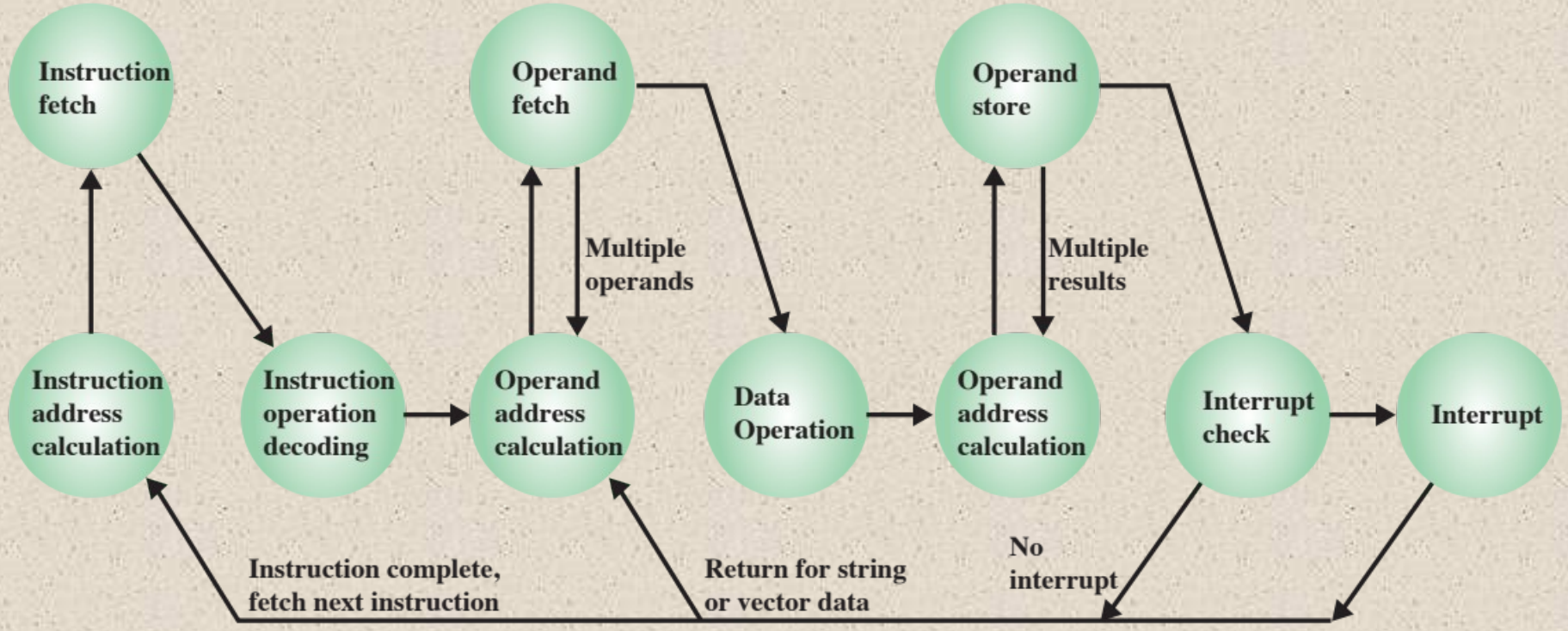
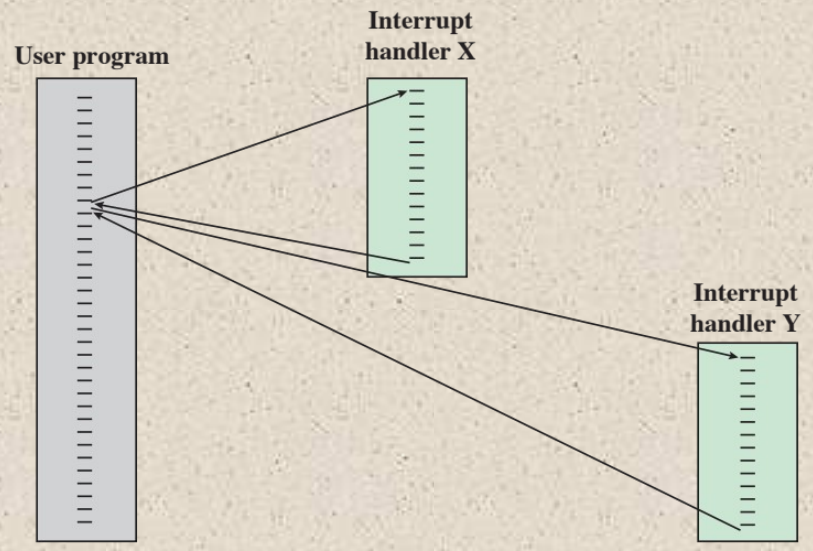
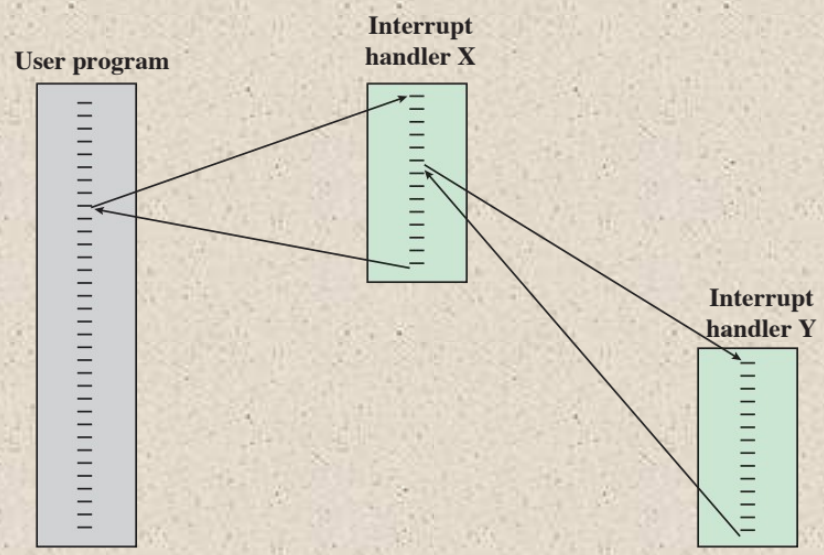


Figure 3.12 Instruction Cycle State Diagram, With Interrupts



(a) Sequential interrupt processing



(b) Nested interrupt processing

Figure 3.13 Transfer of Control with Multiple Interrupts

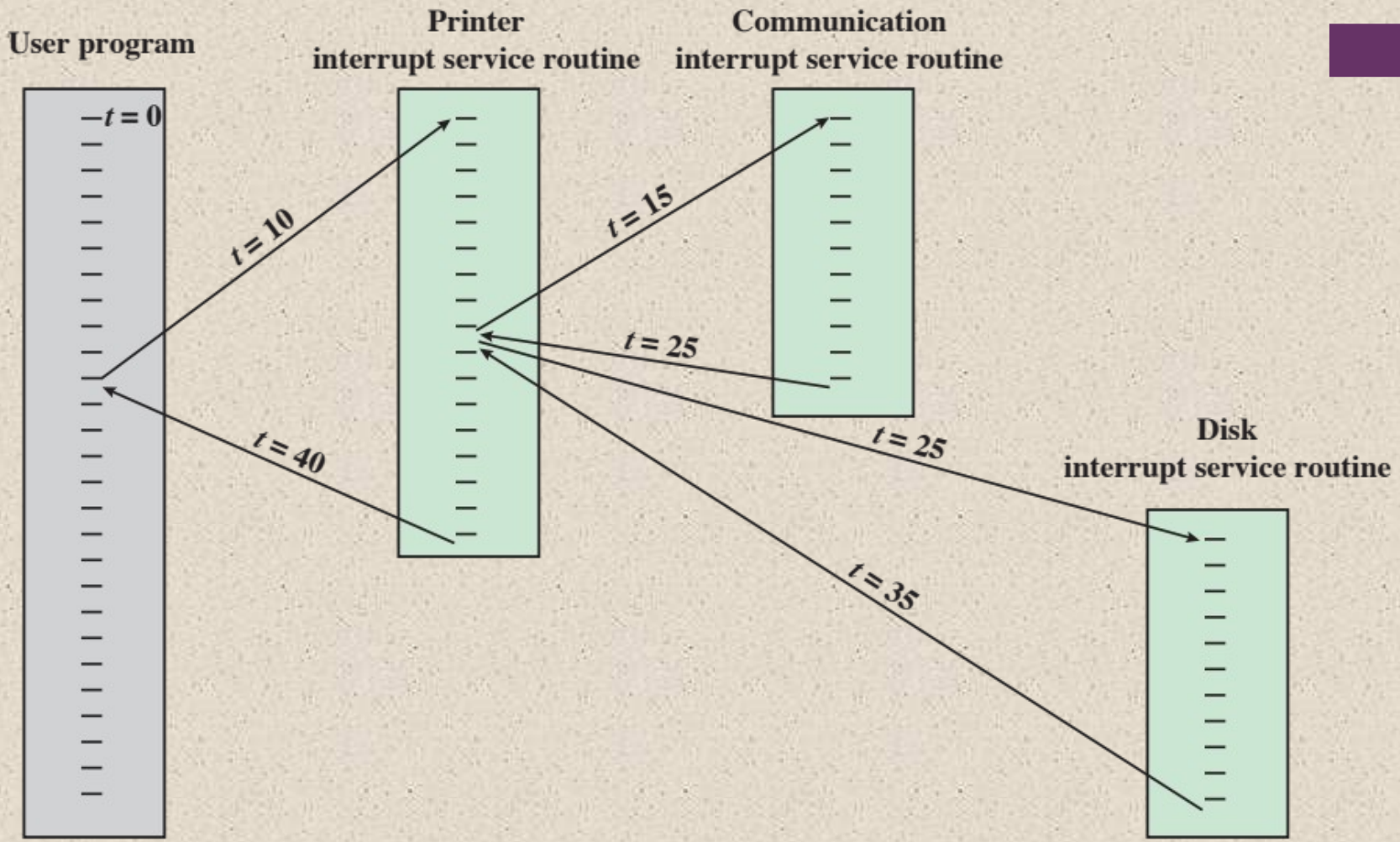


Figure 3.14 Example Time Sequence of Multiple Interrupts

I/O Function

- I/O module can exchange data directly with the processor
- Processor can read data from or write data to an I/O module
  - Processor identifies a specific device that is controlled by a particular I/O module
  - I/O instructions rather than memory referencing instructions
- In some cases it is desirable to allow I/O exchanges to occur directly with memory
  - The processor grants to an I/O module the authority to read from or write to memory so that the I/O memory transfer can occur without tying up the processor
  - The I/O module issues read or write commands to memory relieving the processor of responsibility for the exchange
  - This operation is known as direct memory access (DMA)

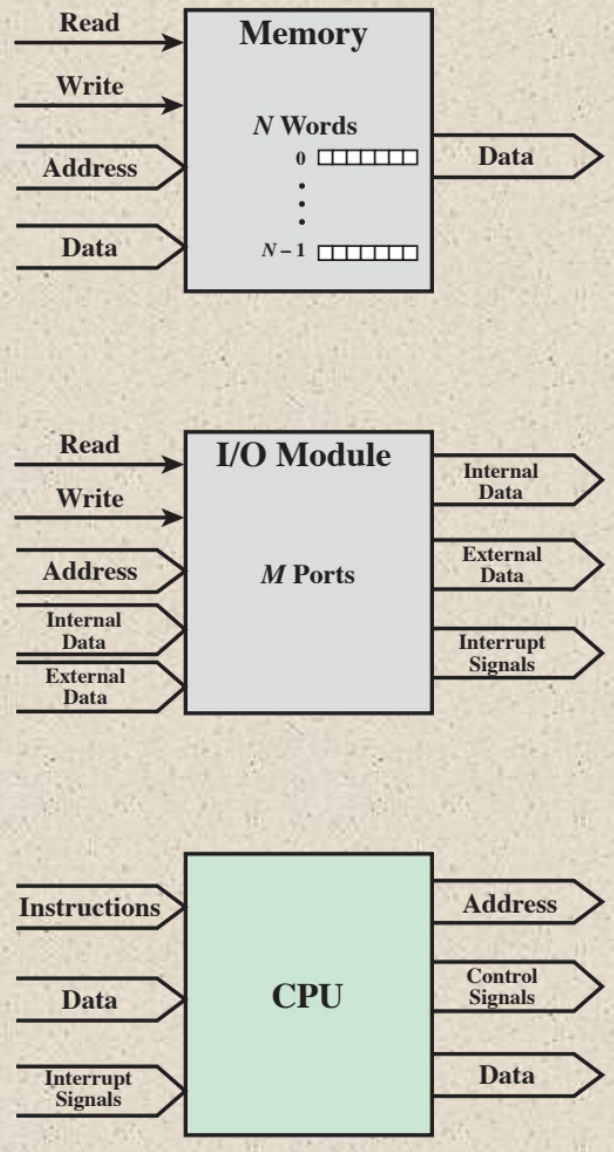


Figure 3.15 Computer Modules

The interconnection structure must support the following types of transfers:

Memory to processor	Processor to memory	I/O to processor	Processor to I/O	I/O to or from memory
Processor reads an instruction or a unit of data from memory	Processor writes a unit of data to memory	Processor reads data from an I/O device via an I/O module	Processor sends data to the I/O device	An I/O module is allowed to exchange data directly with memory without going through the processor using direct memory access

Bus Interconnection

A communication pathway connecting two or more devices

- Key characteristic is that it is a shared transmission medium

Signals transmitted by any one device are available for reception by all other devices attached to the bus

- If two devices transmit during the same time period their signals will overlap and become garbled

Typically consists of multiple communication lines

- Each line is capable of transmitting signals representing binary 1 and binary 0

Computer systems contain a number of different buses that provide pathways between components at various levels of the computer system hierarchy

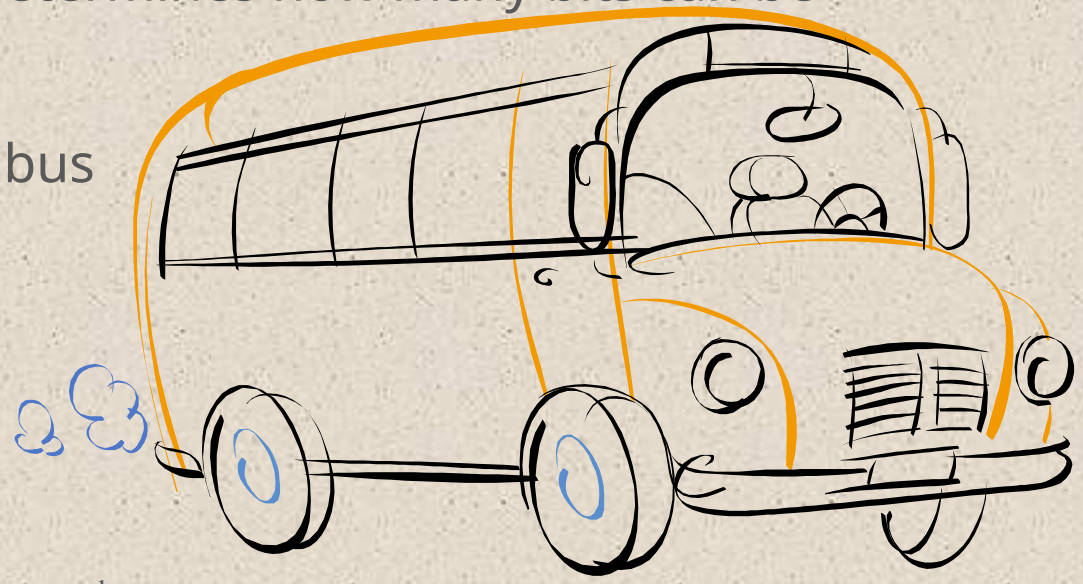
System bus

- A bus that connects major computer components (processor, memory, I/O)

The most common computer interconnection structures are based on the use of one or more system buses

Data Bus

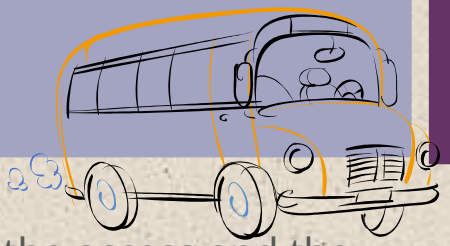
- Data lines that provide a path for moving data among system modules
- May consist of 32, 64, 128, or more separate lines
- The number of lines is referred to as the *width* of the data bus
- The number of lines determines how many bits can be transferred at a time
- The width of the data bus is a key factor in determining overall system performance



Address Bus

- Used to designate the source or destination of the data on the data bus
  - If the processor wishes to read a word of data from memory it puts the address of the desired word on the address lines
- Width determines the maximum possible memory capacity of the system
- Also used to address I/O ports
  - The higher order bits are used to select a particular module on the bus and the lower order bits select a memory location or I/O port within the module

Control Bus



- Used to control the access and the use of the data and address lines
- Because the data and address lines are shared by all components there must be a means of controlling their use
- Control signals transmit both command and timing information among system modules
- Timing signals indicate the validity of data and address information
- Command signals specify operations to be performed

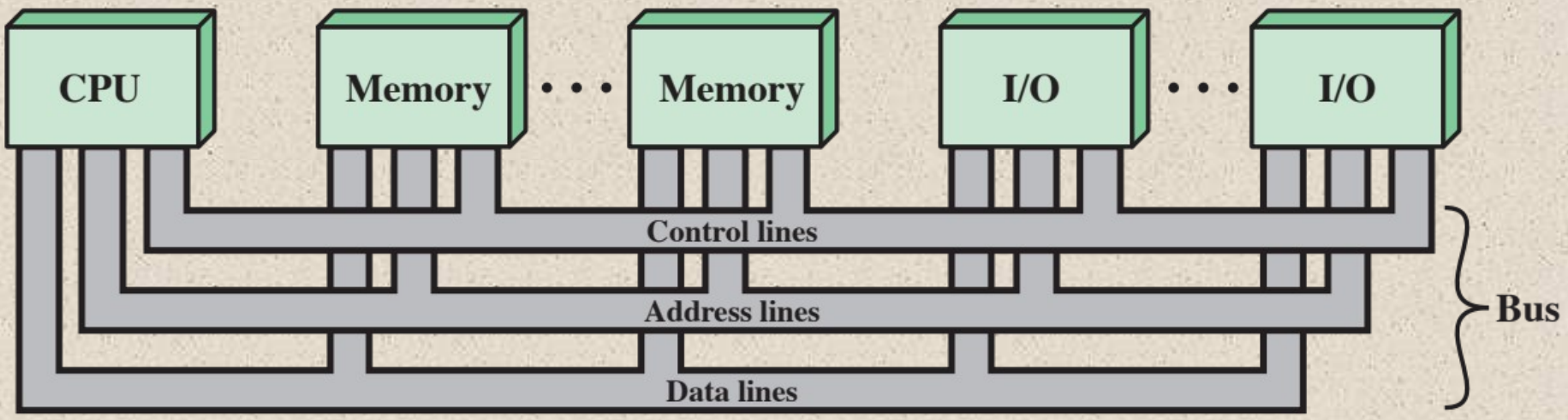


Figure 3.16 Bus Interconnection Scheme

Point-to-Point Interconnect

Principal reason for change was the electrical constraints encountered with increasing the frequency of wide synchronous buses

At higher and higher data rates it becomes increasingly difficult to perform the synchronization and arbitration functions in a timely fashion

A conventional shared bus on the same chip magnified the difficulties of increasing bus data rate and reducing bus latency to keep up with the processors

Has lower latency, higher data rate, and better scalability

Quick Path Interconnect

QPI

- Introduced in 2008
- Multiple direct connections
  - Direct pairwise connections to other components eliminating the need for arbitration found in shared transmission systems
- Layered protocol architecture
  - These processor level interconnects use a layered protocol architecture rather than the simple use of control signals found in shared bus arrangements
- Packetized data transfer
  - Data are sent as a sequence of packets each of which includes control headers and error control codes

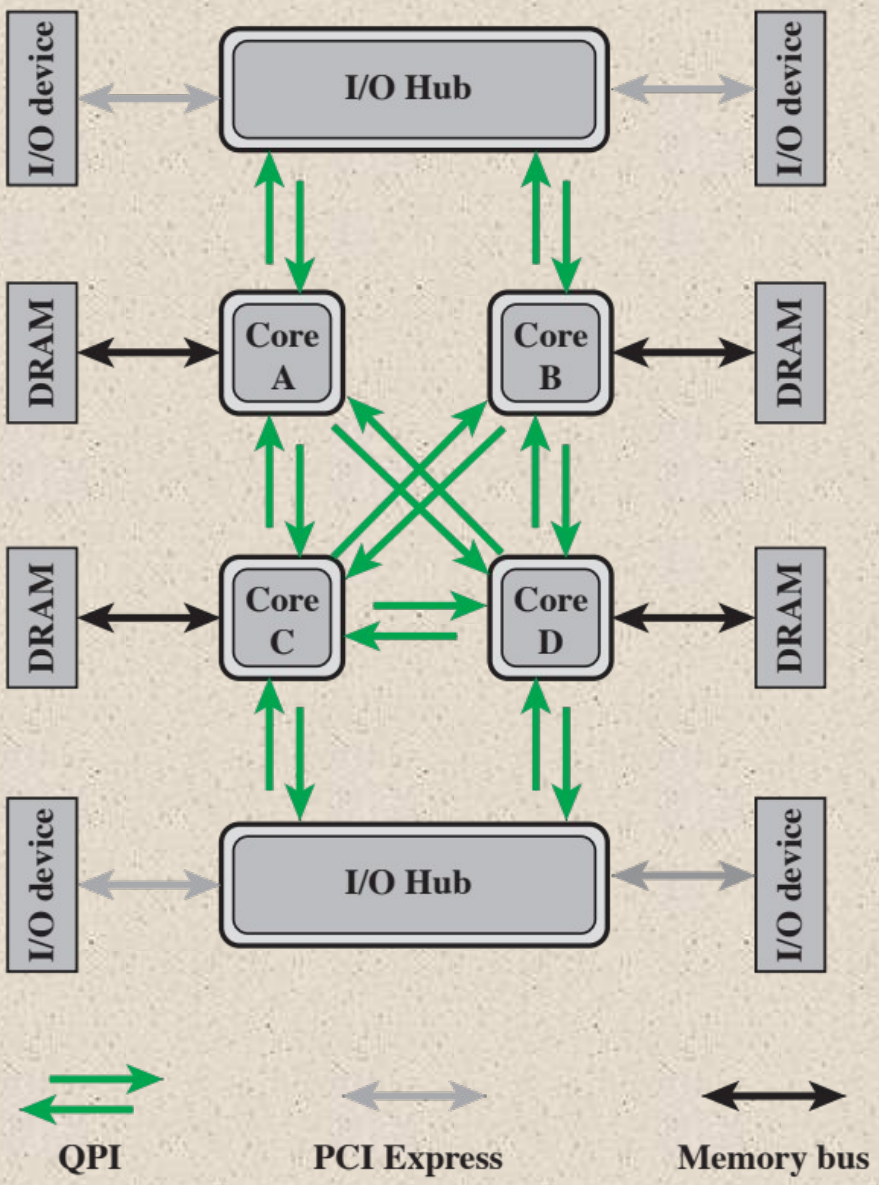


Figure 3.17 Multicore Configuration Using QPI

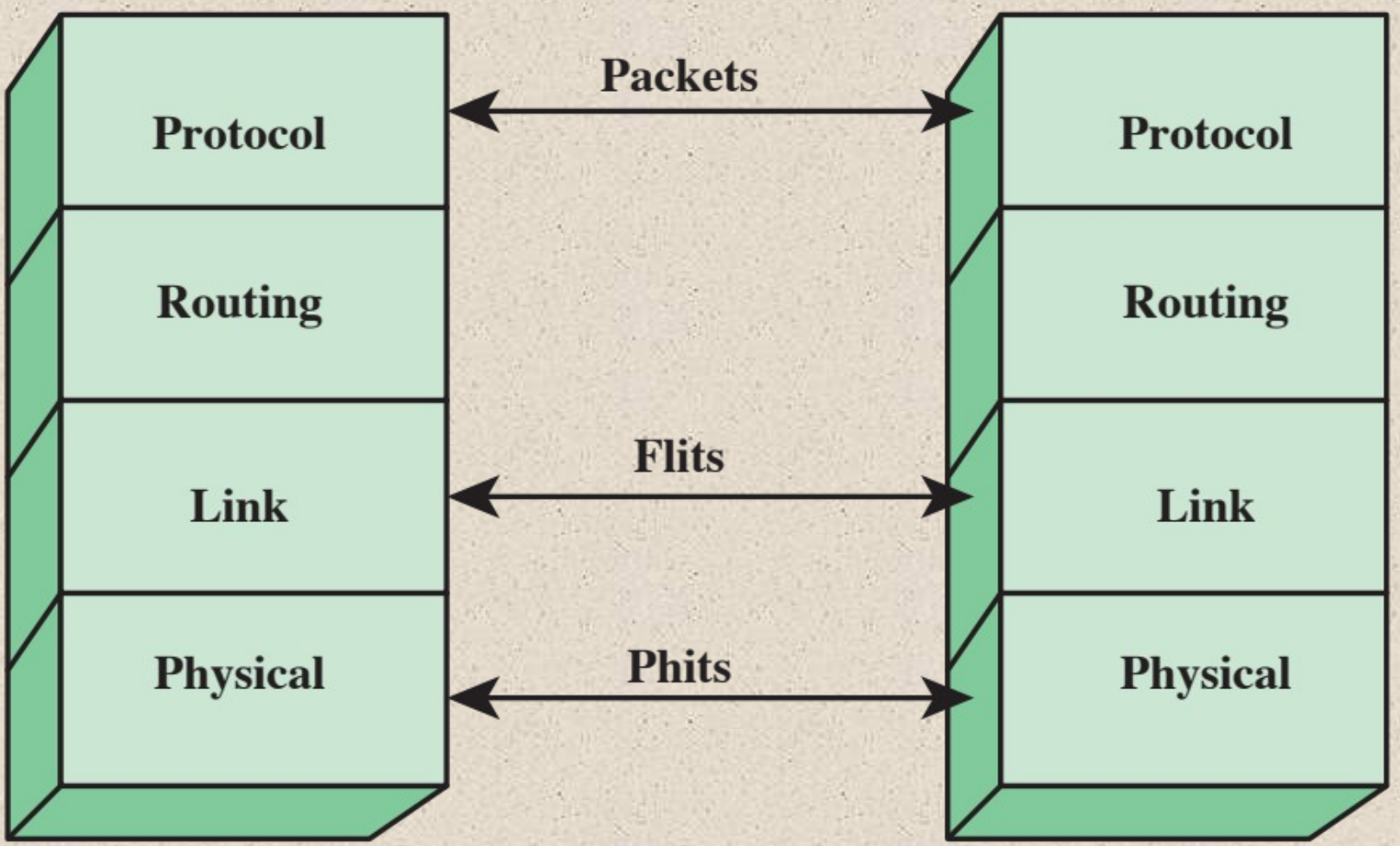


Figure 3.18 QPI Layers

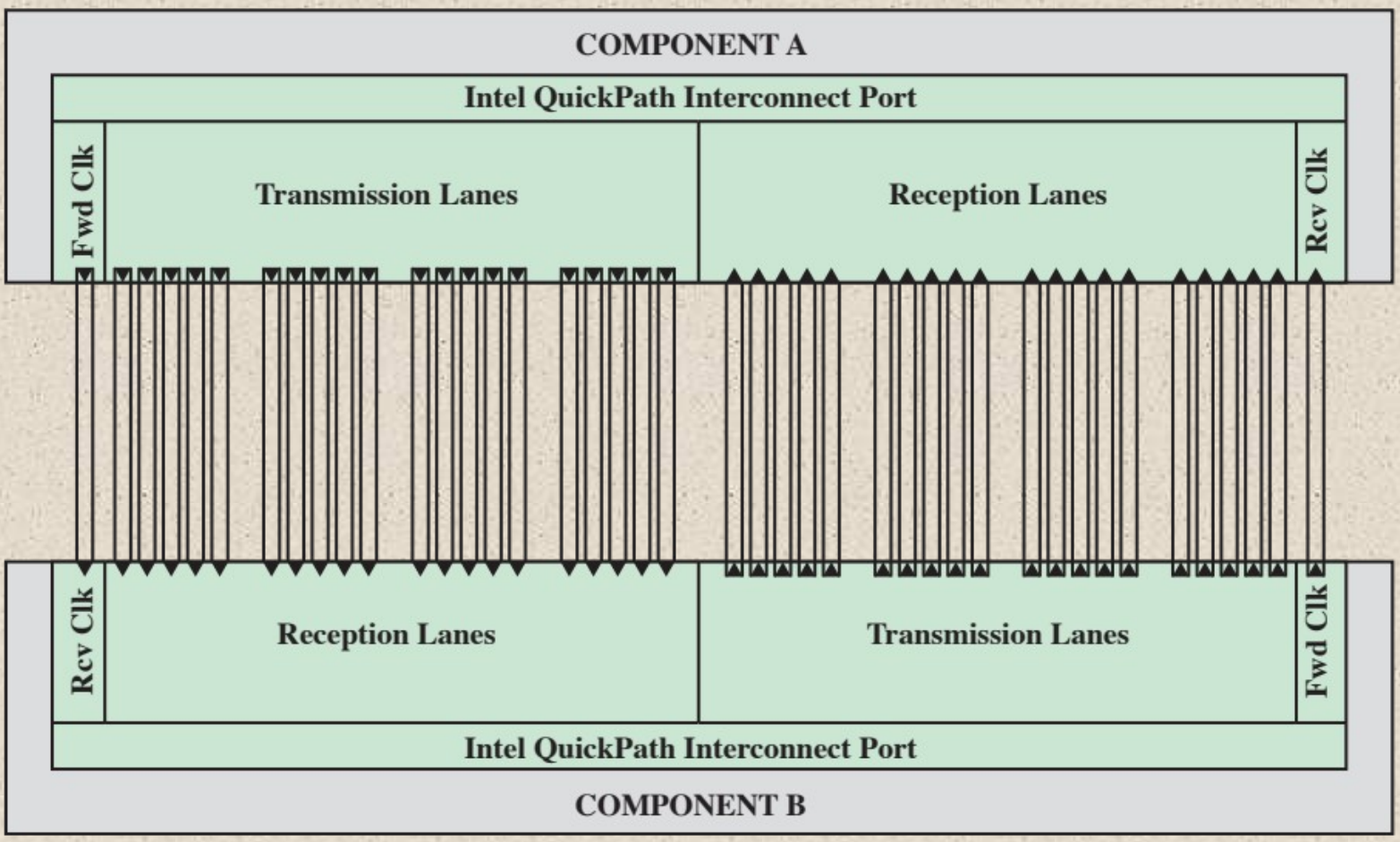


Figure 3.19 Physical Interface of the Intel QPI Interconnect

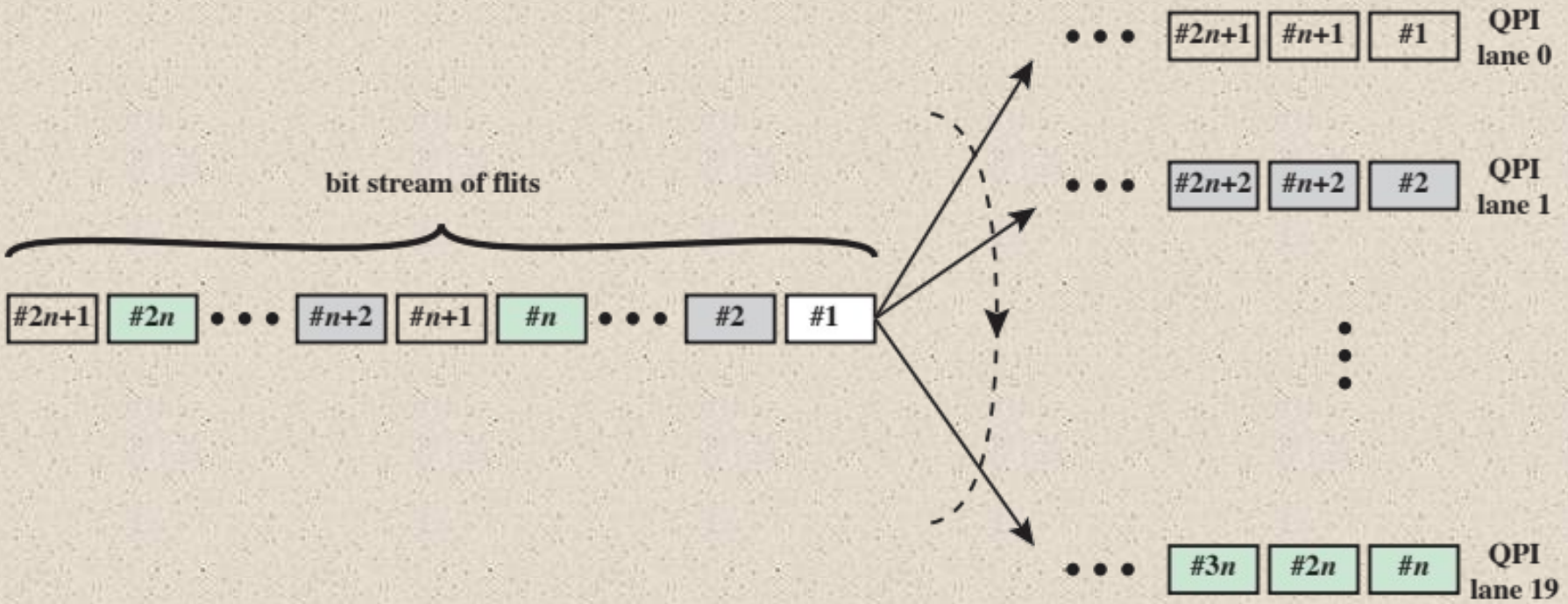


Figure 3.20 QPI Multilane Distribution

QPI Link Layer

- Performs two key functions: *flow control* and *error control*
  - Operate on the level of the flit (flow control unit)
  - Each flit consists of a 72-bit message payload and an 8-bit error control code called a *cyclic redundancy check* (CRC)

- Flow control function
  - Needed to ensure that a sending QPI entity does not overwhelm a receiving QPI entity by sending data faster than the receiver can process the data and clear buffers for more incoming data

- Error control function
  - Detects and recovers from bit errors, and so isolates higher layers from experiencing bit errors

QPI Routing and Protocol Layers

Routing Layer

- Used to determine the course that a packet will traverse across the available system interconnects
- Defined by firmware and describe the possible paths that a packet can follow

Protocol Layer

- Packet is defined as the unit of transfer
- One key function performed at this level is a cache coherency protocol which deals with making sure that main memory values held in multiple caches are consistent
- A typical data packet payload is a block of data being sent to or from a cache

Peripheral Component Interconnect (PCI)

- A popular high bandwidth, processor independent bus that can function as a mezzanine or peripheral bus
- Delivers better system performance for high speed I/O subsystems
- PCI Special Interest Group (SIG)
  - Created to develop further and maintain the compatibility of the PCI specifications
- PCI Express (PCIe)
  - Point-to-point interconnect scheme intended to replace bus-based schemes such as PCI
  - Key requirement is high capacity to support the needs of higher data rate I/O devices, such as Gigabit Ethernet
  - Another requirement deals with the need to support time dependent data streams

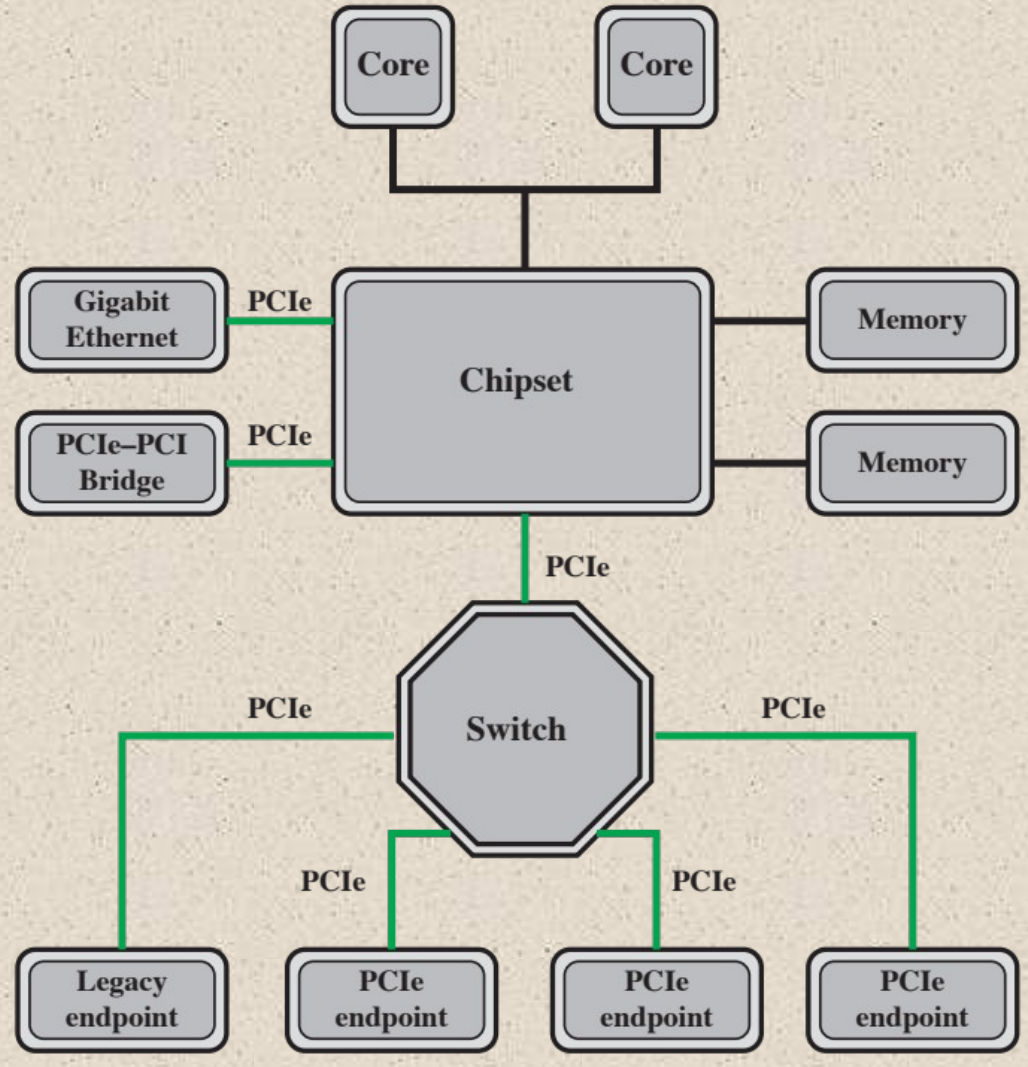


Figure 3.21 Typical Configuration Using PCIe

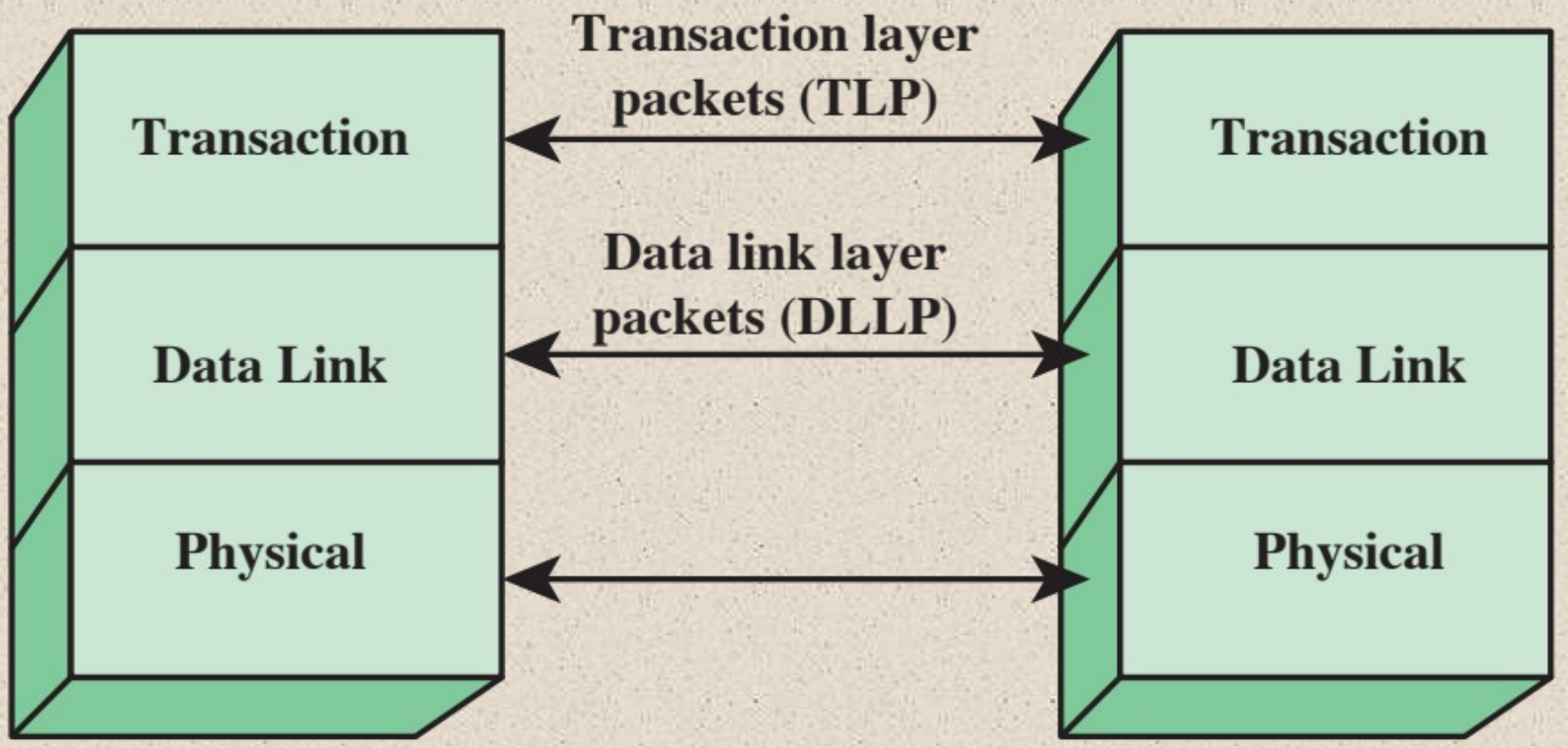


Figure 3.22 PCIe Protocol Layers

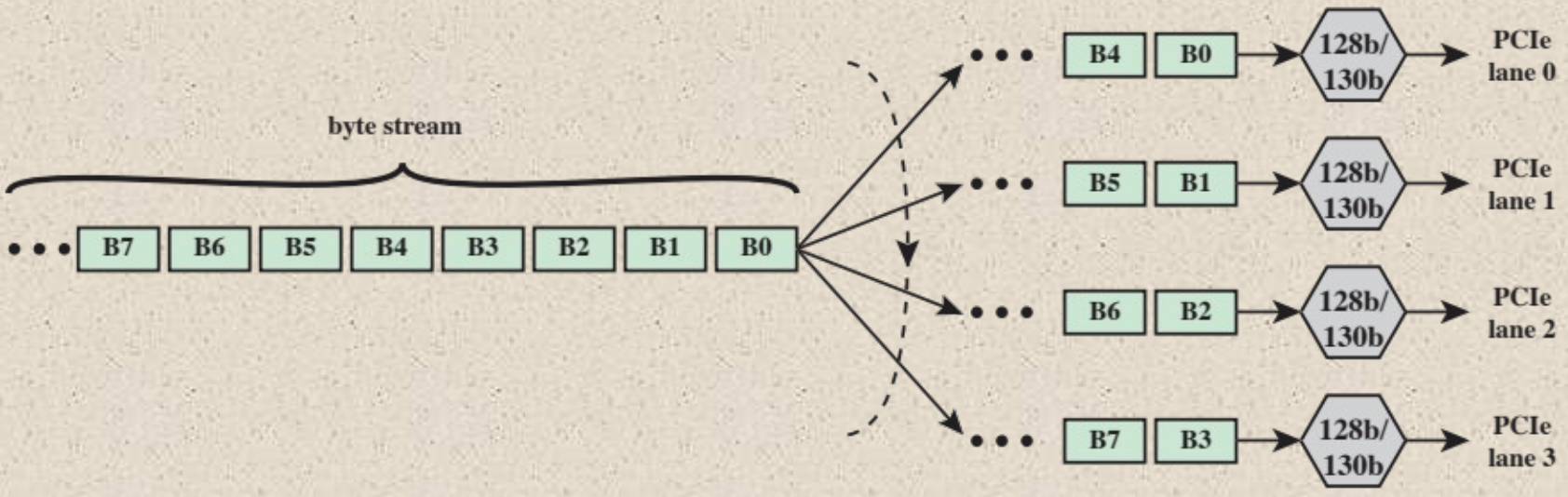


Figure 3.23 PCIe Multilane Distribution

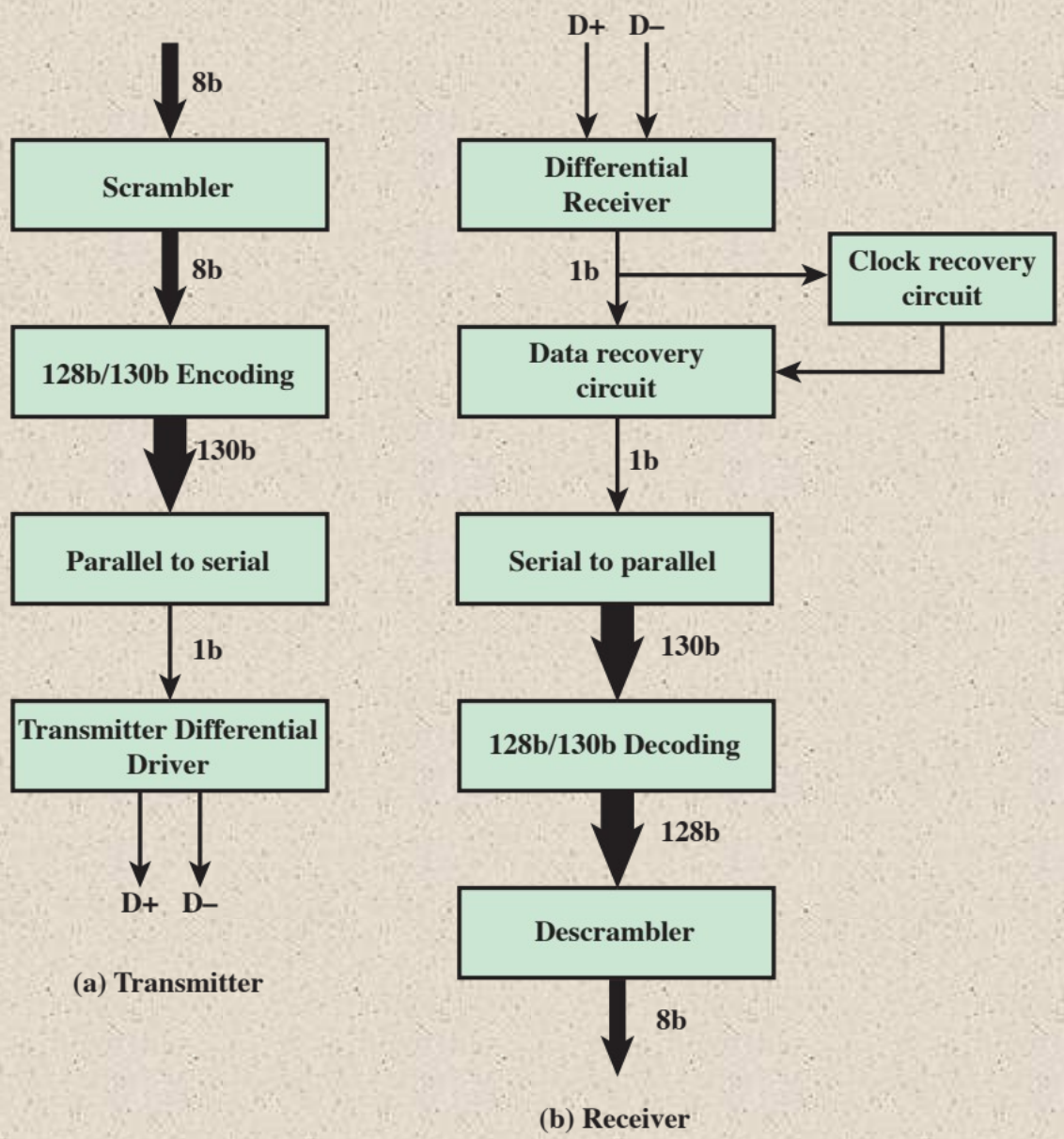


Figure 3.24 PCIe Transmit and Receive Block Diagrams

PCIe

Transaction Layer (TL)

- Receives read and write requests from the software above the TL and creates request packets for transmission to a destination via the link layer
- Most transactions use a *split transaction* technique
  - A request packet is sent out by a source PCIe device which then waits for a response called a *completion* packet
- TL messages and some write transactions are posted transactions (meaning that no response is expected)
- TL packet format supports 32-bit memory addressing and extended 64-bit memory addressing

The TL supports four address spaces:

- Memory
  - The memory space includes system main memory and PCIe I/O devices
  - Certain ranges of memory addresses map into I/O devices
- Configuration
  - This address space enables the TL to read/write configuration registers associated with I/O devices
- I/O
  - This address space is used for legacy PCI devices, with reserved address ranges used to address legacy I/O devices
- Message
  - This address space is for control signals related to interrupts, error handling, and power management

Table 3.2
PCIe TLP Transaction Types

Address Space	TLP Type	Purpose
Memory	Memory Read Request	Transfer data to or from a location in the system memory map.
	Memory Read Lock Request	
	Memory Write Request	
I/O	I/O Read Request	Transfer data to or from a location in the system memory map for legacy devices.
	I/O Write Request	
Configuration	Config Type 0 Read Request	Transfer data to or from a location in the configuration space of a PCIe device.
	Config Type 0 Write Request	
	Config Type 1 Read Request	
	Config Type 1 Write Request	
Message	Message Request	Provides in-band messaging and event reporting.
	Message Request with Data	
Memory, I/O, Configuration	Completion	Returned for certain requests.
	Completion with Data	
	Completion Locked	
	Completion Locked with Data	

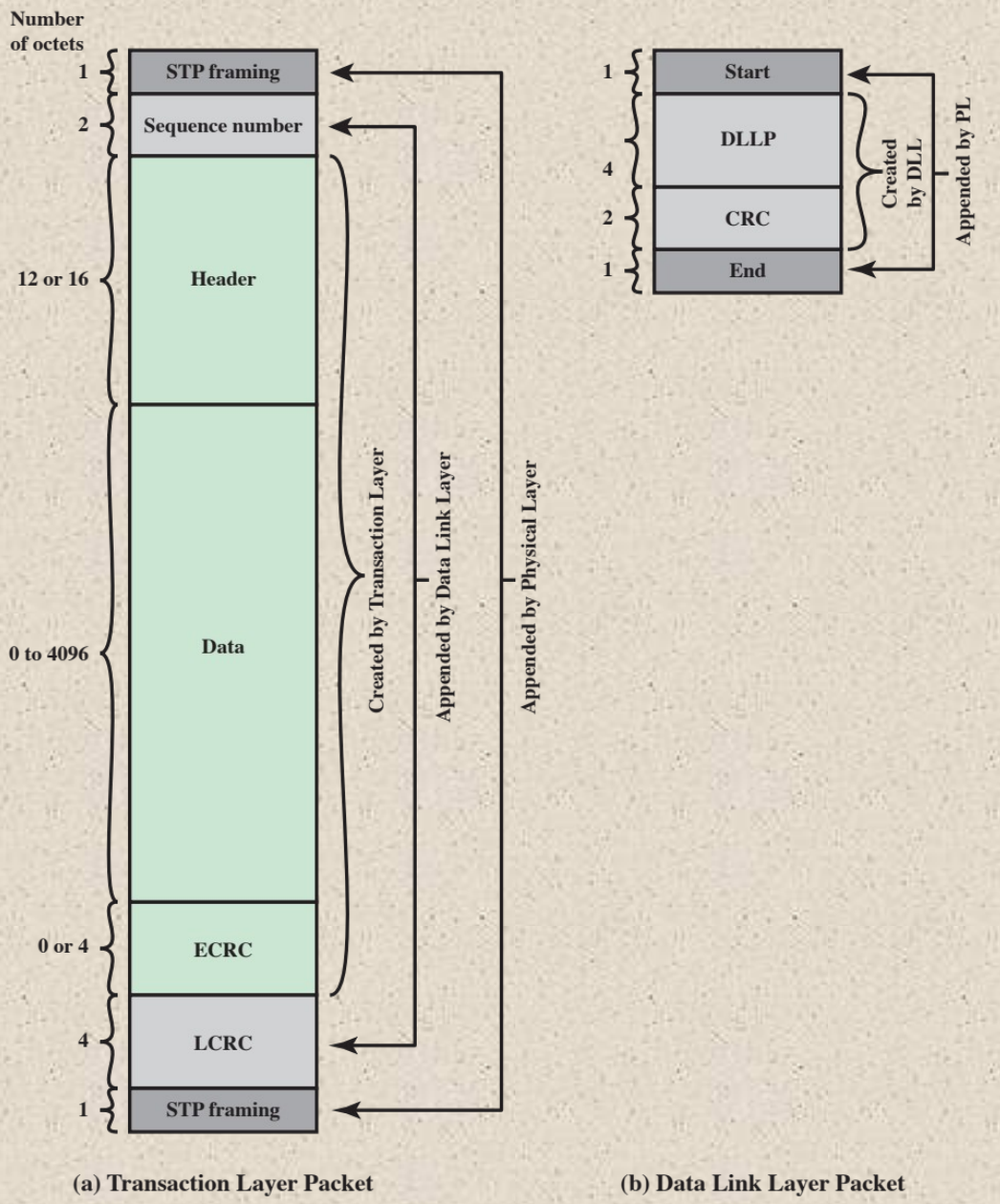


Figure 3.25 PCIe Protocol Data Unit Format

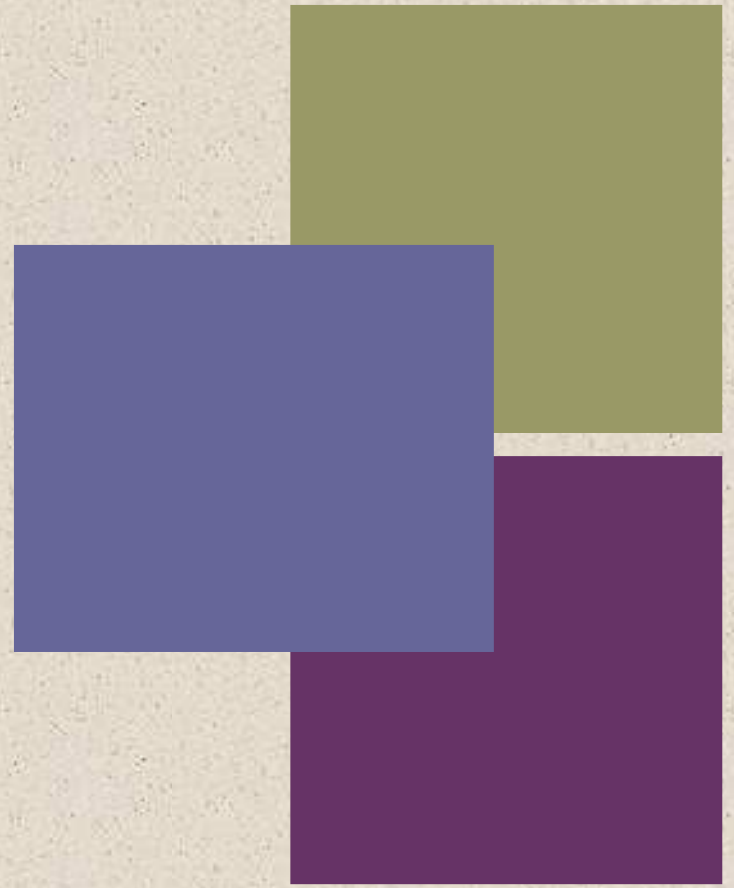
Summary

A Top-Level View of Computer Function and Interconnection

Chapter 3

- Computer components
- Computer function
  - Instruction fetch and execute
  - Interrupts
  - I/O function
- Interconnection structures
- Bus interconnection
- Point-to-point interconnect
  - QPI physical layer
  - QPI link layer
  - QPI routing layer
  - QPI protocol layer
- PCI express
  - PCI physical and logical architecture
  - PCIe physical layer
  - PCIe transaction layer
  - PCIe data link layer

William Stallings
Computer Organization and Architecture
10th Edition



Chapter 4
Cache Memory

Location	**Performance**
Internal (e.g. processor registers, cache, main memory)	Access time
External (e.g. optical disks, magnetic disks, tapes)	Cycle time
Capacity	Transfer rate
Number of words	**Physical Type**
Number of bytes	Semiconductor
Unit of Transfer	Magnetic
Word	Optical
Block	Magneto-optical
Access Method	**Physical Characteristics**
Sequential	Volatile/nonvolatile
Direct	Erasable/nonerasable
Random	**Organization**
Associative	Memory modules

Table 4.1
Key Characteristics of Computer Memory Systems

Characteristics of Memory Systems

- Location
  - Refers to whether memory is internal and external to the computer
  - Internal memory is often equated with main memory
  - Processor requires its own local memory, in the form of registers
  - Cache is another form of internal memory
  - External memory consists of peripheral storage devices that are accessible to the processor via I/O controllers
- Capacity
  - Memory is typically expressed in terms of bytes
- Unit of transfer
  - For internal memory the unit of transfer is equal to the number of electrical lines into and out of the memory module

Method of Accessing Units of Data

Sequential access

- Memory is organized into units of data called records
- Access must be made in a specific linear sequence
- Access time is variable

Direct access

- Involves a shared read-write mechanism
- Individual blocks or records have a unique address based on physical location
- Access time is variable

Random access

- Each addressable location in memory has a unique, physically wired-in addressing mechanism
- The time to access a given location is independent of the sequence of prior accesses and is constant
- Any location can be selected at random and directly addressed and accessed
- Main memory and some cache systems are random access

Associative

- A word is retrieved based on a portion of its contents rather than its address
- Each location has its own addressing mechanism and retrieval time is constant independent of location or prior access patterns
- Cache memories may employ associative access

Capacity and Performance:

The two most important characteristics of memory

Three performance parameters are used:

Access time (latency)

- For random-access memory it is the time it takes to perform a read or write operation
- For non-random-access memory it is the time it takes to position the read-write mechanism at the desired location

Memory cycle time

- Access time plus any additional time required before second access can commence
- Additional time may be required for transients to die out on signal lines or to regenerate data if they are read destructively
- Concerned with the system bus, not the processor

Transfer rate

- The rate at which data can be transferred into or out of a memory unit
- For random-access memory it is equal to 1/(cycle time)

Memory

- The most common forms are:
  - Semiconductor memory
  - Magnetic surface memory
  - Optical
  - Magneto-optical
- Several physical characteristics of data storage are important:
  - Volatile memory
    - Information decays naturally or is lost when electrical power is switched off
  - Nonvolatile memory
    - Once recorded, information remains without deterioration until deliberately changed
    - No electrical power is needed to retain information
  - Magnetic-surface memories
    - Are nonvolatile
  - Semiconductor memory
    - May be either volatile or nonvolatile
  - Nonerasable memory
    - Cannot be altered, except by destroying the storage unit
    - Semiconductor memory of this type is known as read-only memory (ROM)
- For random-access memory the organization is a key design issue
  - Organization refers to the physical arrangement of bits to form words

Memory Hierarchy

- Design constraints on a computer's memory can be summed up by three questions:
  - How much, how fast, how expensive
- There is a trade-off among capacity, access time, and cost
  - Faster access time, greater cost per bit
  - Greater capacity, smaller cost per bit
  - Greater capacity, slower access time
- The way out of the memory dilemma is not to rely on a single memory component or technology, but to employ a memory hierarchy

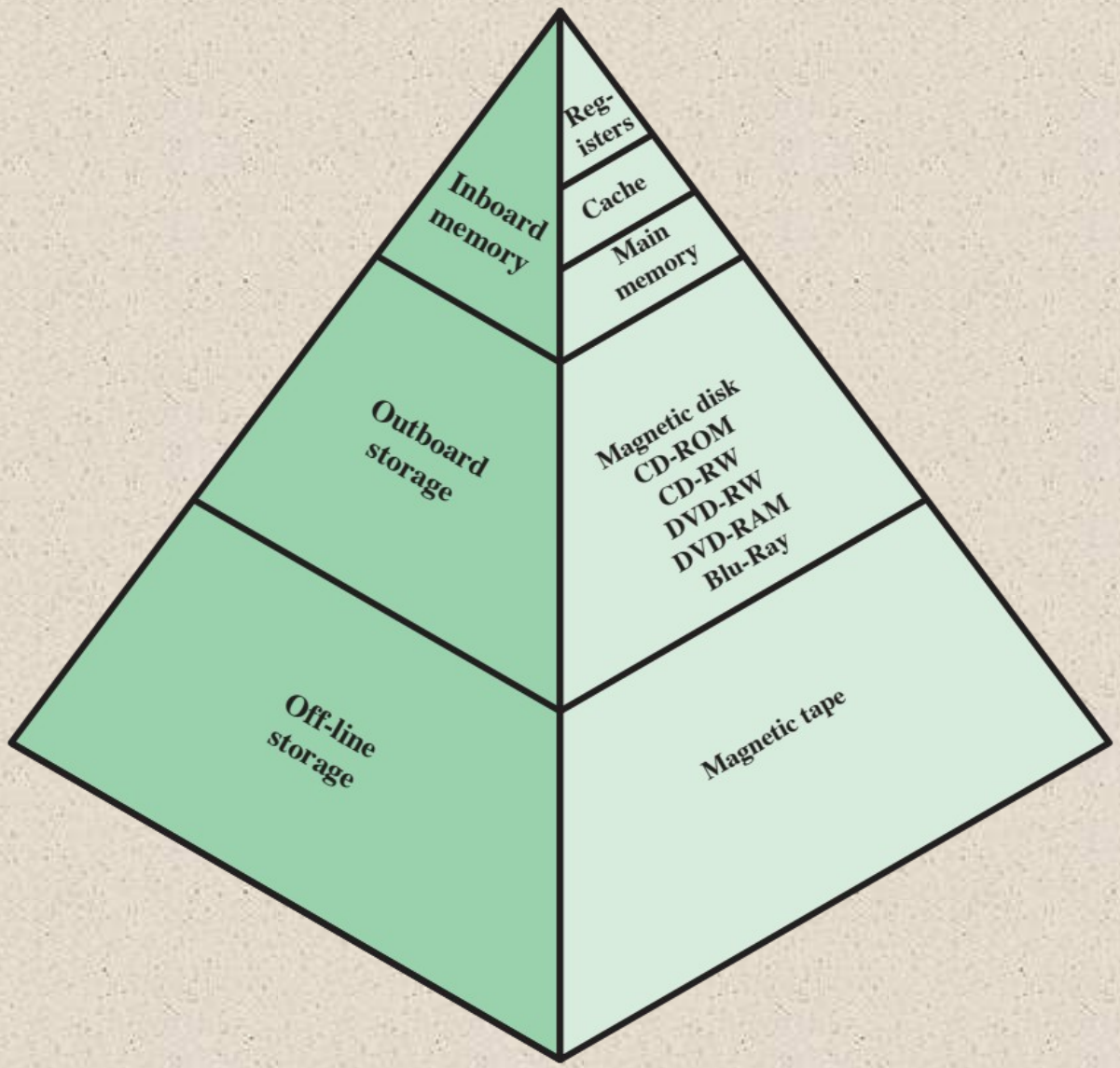


Figure 4.1 The Memory Hierarchy

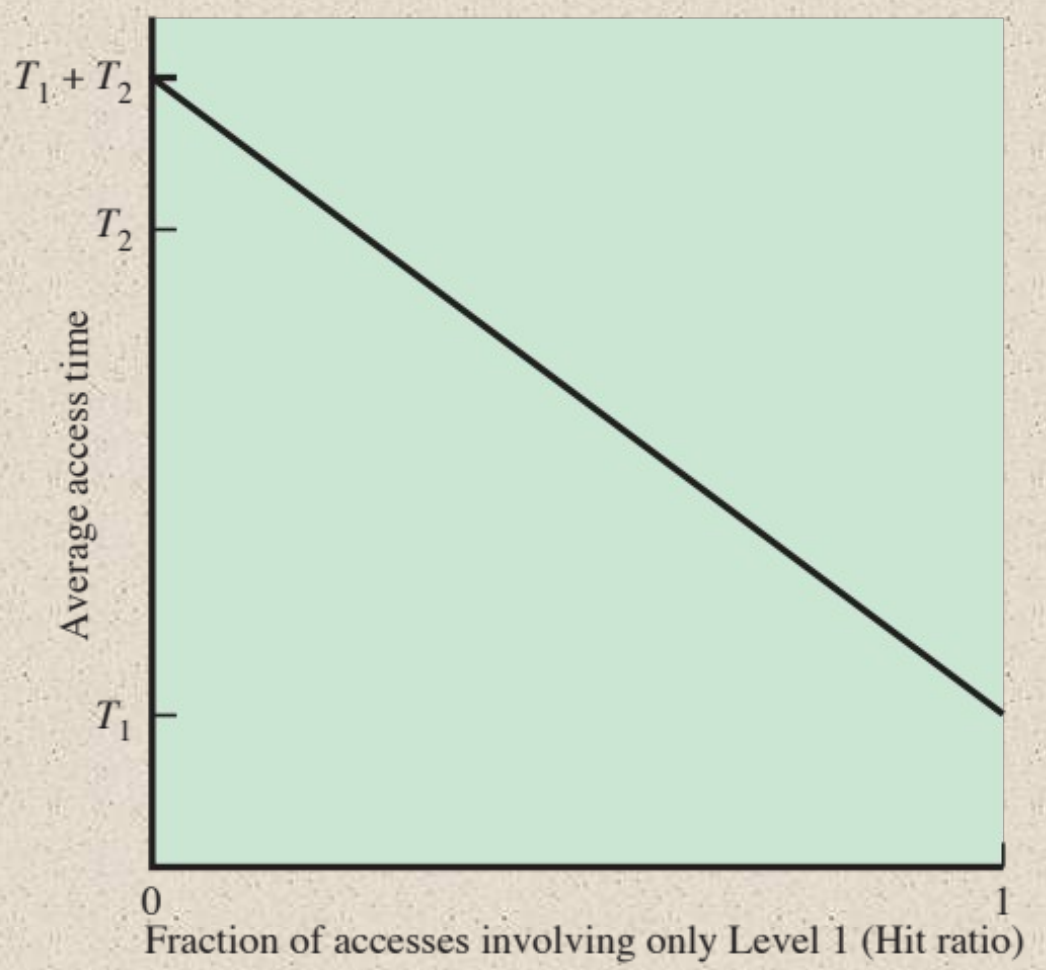


Figure 4.2 Performance of a Simple Two-Level Memory

Memory

- The use of three levels exploits the fact that semiconductor memory comes in a variety of types which differ in speed and cost
- Data are stored more permanently on external mass storage devices
- External, nonvolatile memory is also referred to as **secondary** memory or **auxiliary** memory
- Disk cache
  - A portion of main memory can be used as a buffer to hold data temporarily that is to be read out to disk
  - A few large transfers of data can be used instead of many small transfers of data
  - Data can be retrieved rapidly from the software cache rather than slowly from the disk

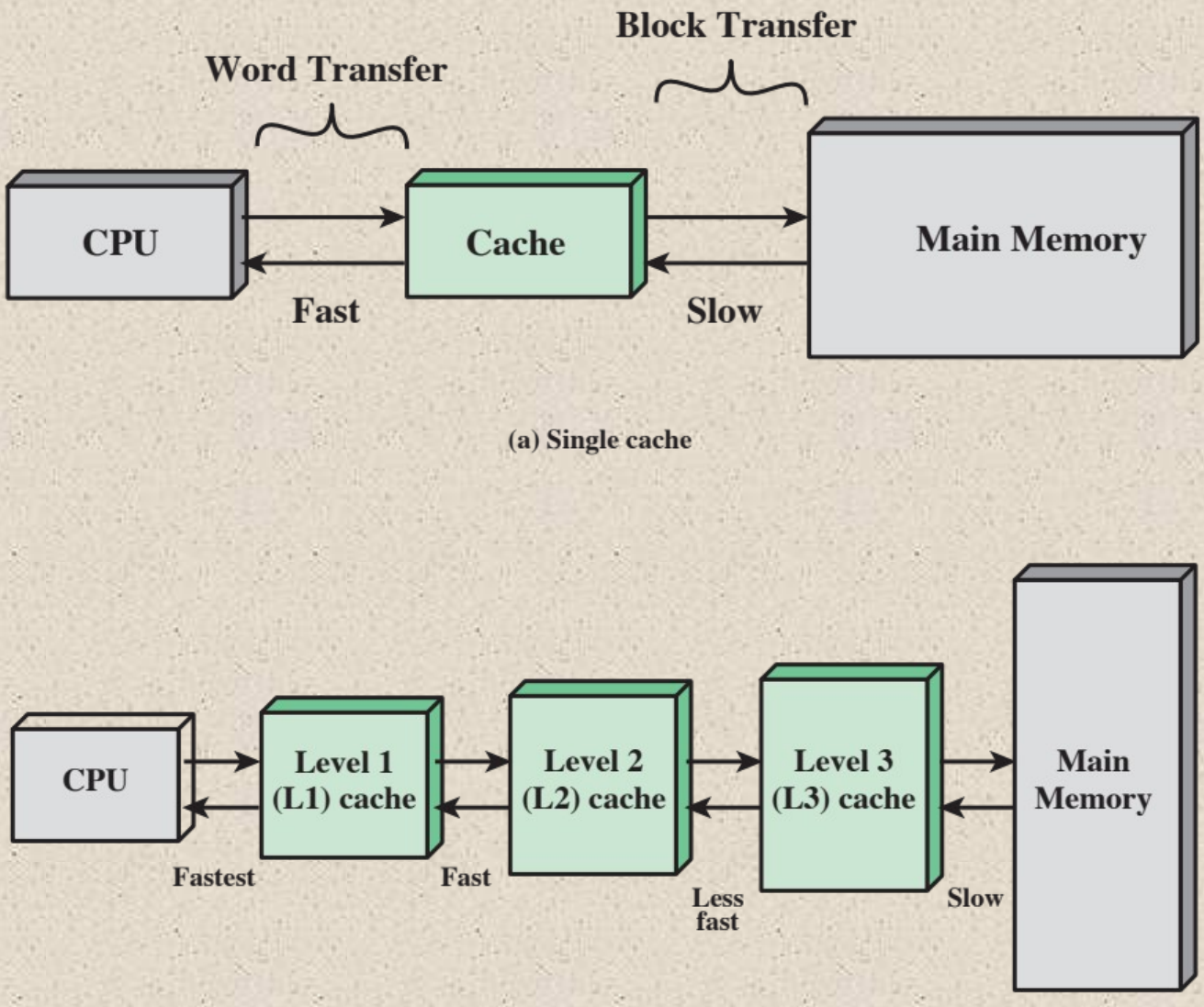


Figure 4.3 Cache and Main Memory

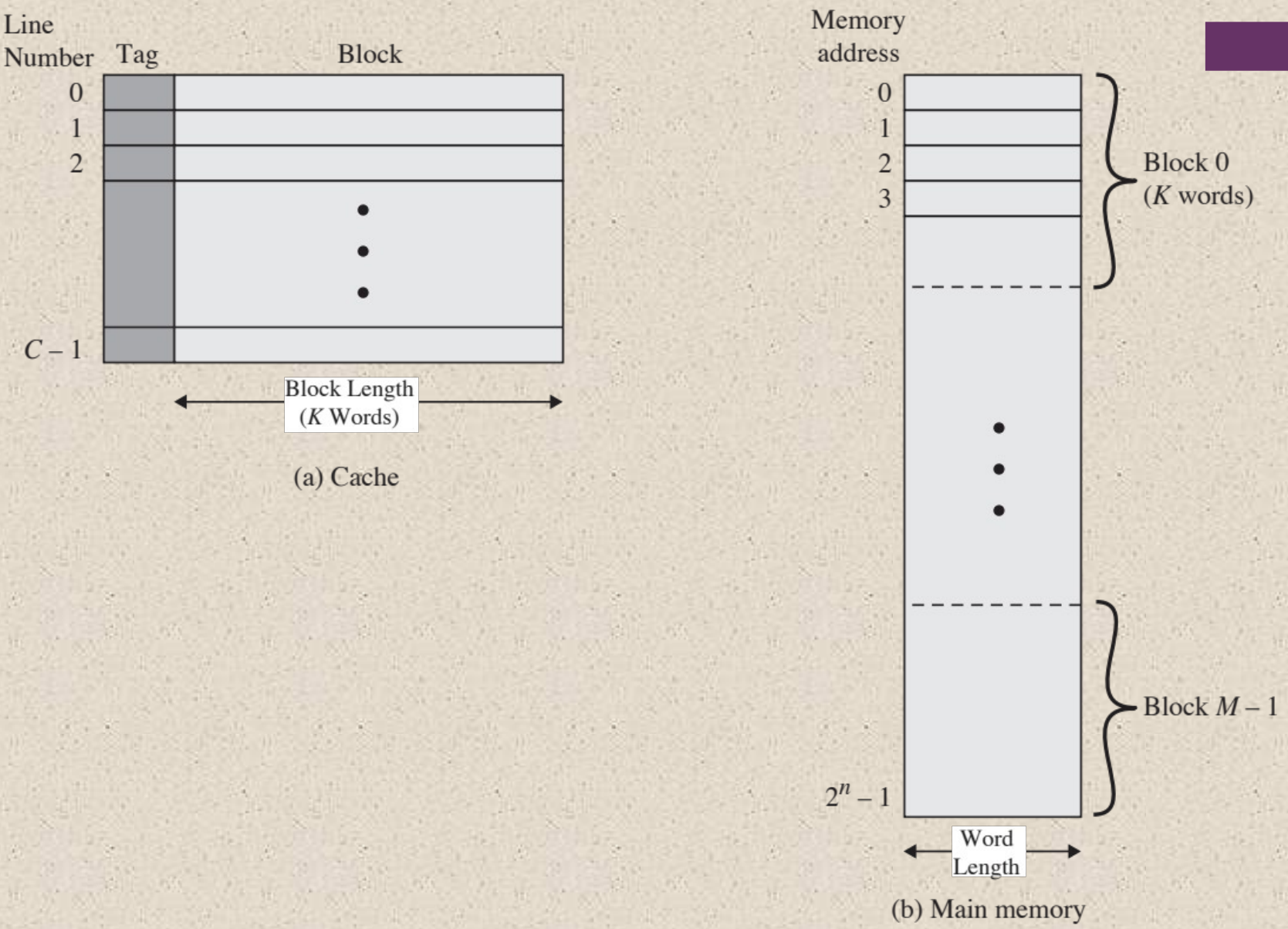


Figure 4.4 Cache/Main-Memory Structure

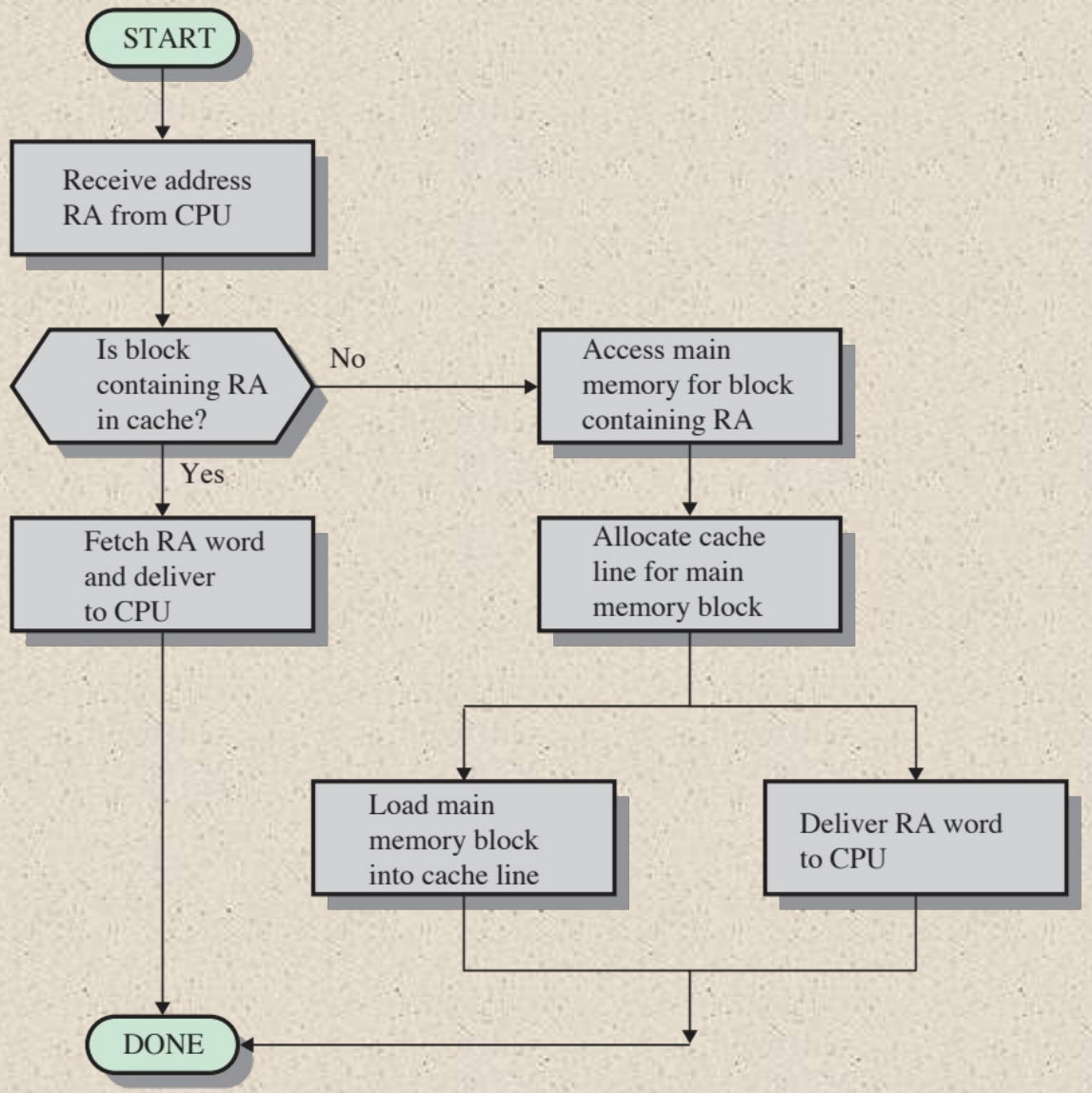


Figure 4.5 Cache Read Operation

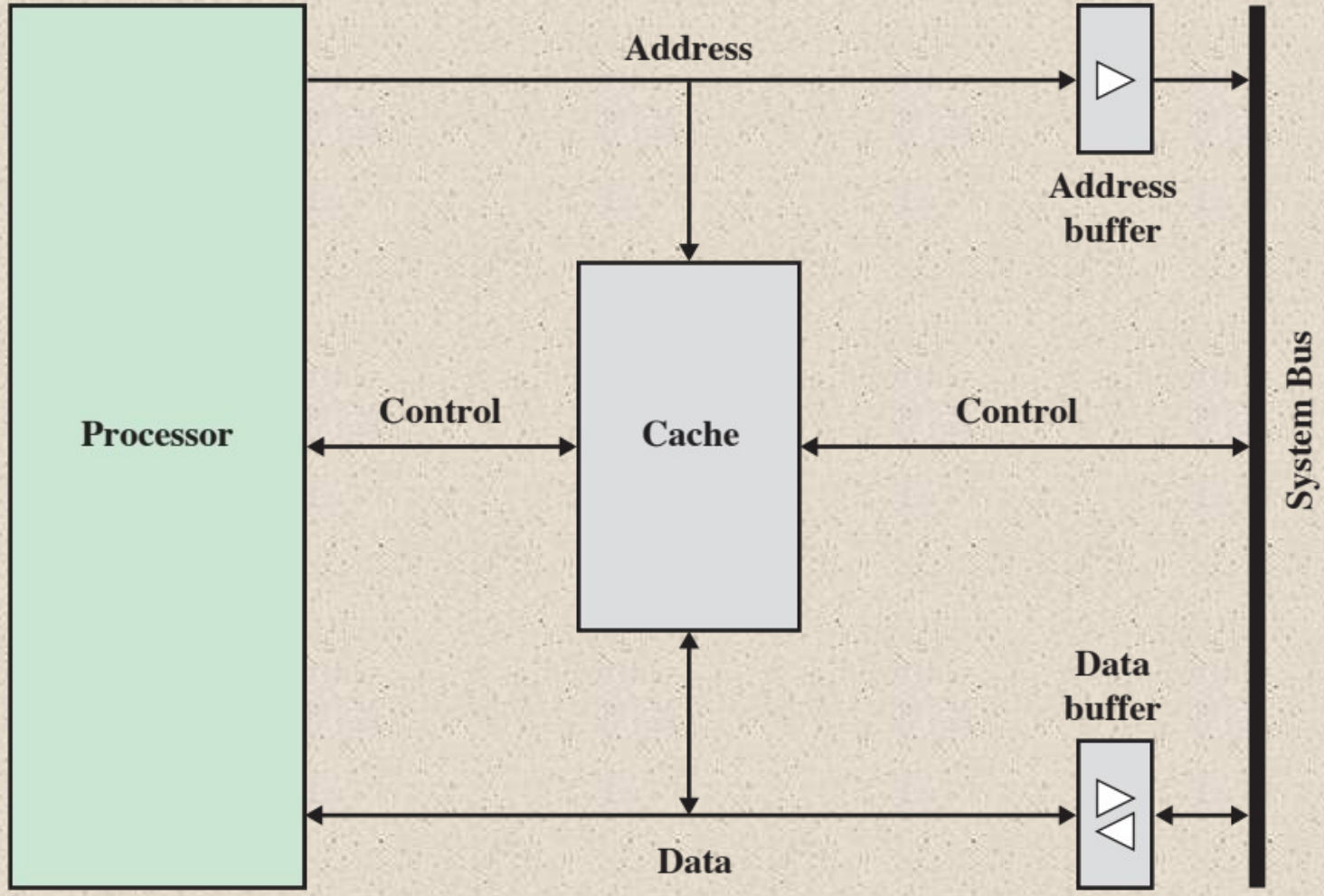


Figure 4.6 Typical Cache Organization

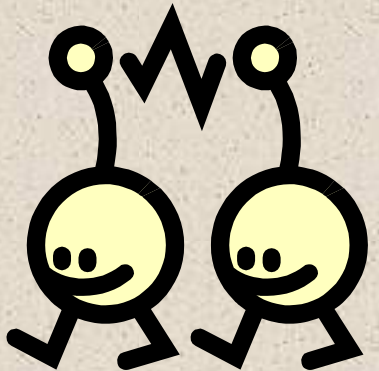
Cache Addresses	**Write Policy**
Logical	Write through
Physical	Write back
Cache Size	**Line Size**
Mapping Function	**Number of caches**
Direct	Single or two level
Associative	Unified or split
Set Associative	
Replacement Algorithm	
Least recently used (LRU)	
First in first out (FIFO)	
Least frequently used (LFU)	
Random	

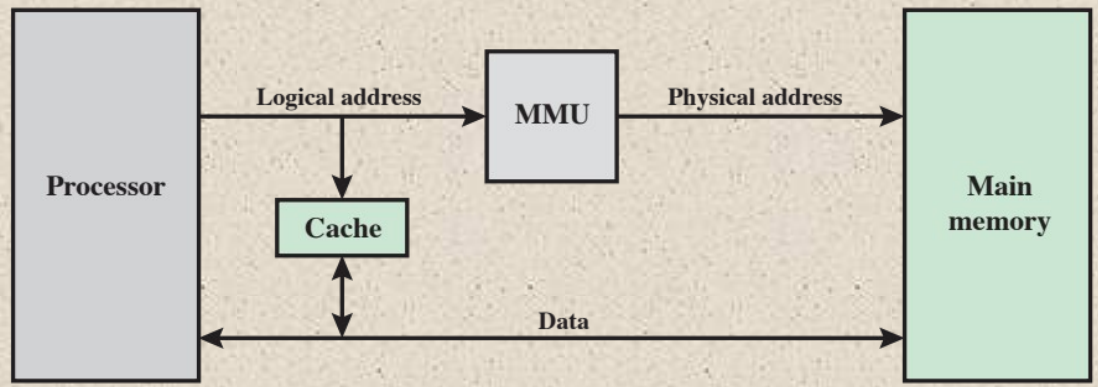
Table 4.2
Elements of Cache Design

Cache Addresses

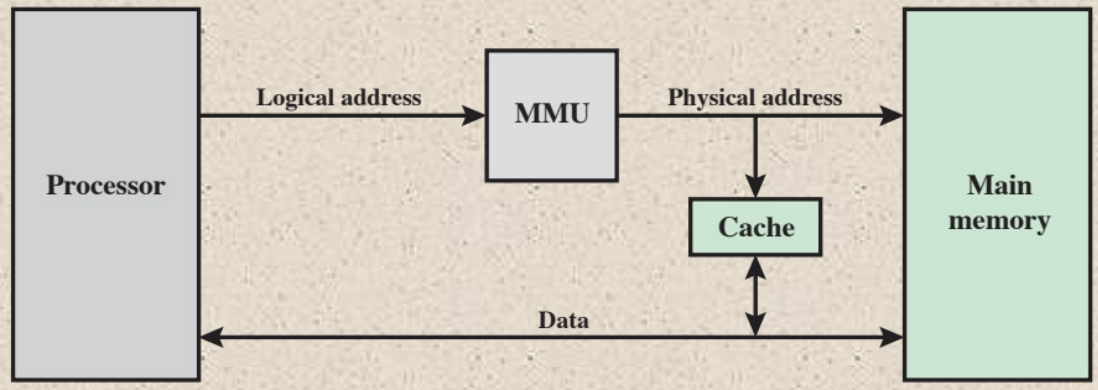
Virtual Memory

- Virtual memory
  - Facility that allows programs to address memory from a logical point of view, without regard to the amount of main memory physically available
  - When used, the address fields of machine instructions contain virtual addresses
  - For reads to and writes from main memory, a hardware memory management unit (MMU) translates each virtual address into a physical address in main memory





(a) Logical Cache



(b) Physical Cache

Figure 4.7 Logical and Physical Caches

Processor	Type	Year of Introduction	L1 Cachea	L2 cache	L3 Cache
IBM 360/85	Mainframe	1968	16 to 32 kB	—	—
PDP-11/70	Minicomputer	1975	1 kB	—	—
VAX 11/780	Minicomputer	1978	16 kB	—	—
IBM 3033	Mainframe	1978	64 kB	—	—
IBM 3090	Mainframe	1985	128 to 256 kB	—	—
Intel 80486	PC	1989	8 kB	—	—
Pentium	PC	1993	8 kB/8 kB	256 to 512 KB	—
PowerPC 601	PC	1993	32 kB	—	—
PowerPC 620	PC	1996	32 kB/32 kB	—	—
PowerPC G4	PC/server	1999	32 kB/32 kB	256 KB to 1 MB	2 MB
IBM S/390 G6	Mainframe	1999	256 kB	8 MB	—
Pentium 4	PC/server	2000	8 kB/8 kB	256 KB	—
IBM SP	High-end server/ supercomputer	2000	64 kB/32 kB	8 MB	—
CRAY MTAb	Supercomputer	2000	8 kB	2 MB	—
Itanium	PC/server	2001	16 kB/16 kB	96 KB	4 MB
Itanium 2	PC/server	2002	32 kB	256 KB	6 MB
IBM POWER5	High-end server	2003	64 kB	1.9 MB	36 MB
CRAY XD-1	Supercomputer	2004	64 kB/64 kB	1MB	—
IBM POWER6	PC/server	2007	64 kB/64 kB	4 MB	32 MB
IBM z10	Mainframe	2008	64 kB/128 kB	3 MB	24-48 MB
Intel Core i7 EE 990	Workstaton/ server	2011	6 × 32 kB/32 kB	1.5 MB	12 MB
IBM zEnterprise 196	Mainframe/ Server	2011	24 × 64 kB/ 128 kB	24 × 1.5 MB	24 MB L3 192 MB L4

Table 4.3

Cache Sizes of Some Processors

[a] Two values separated by a slash refer to instruction and data caches.

[b] Both caches are instruction only; no data caches.

(Table can be found on page 134 in the textbook.)

Mapping Function

- Because there are fewer cache lines than main memory blocks, an algorithm is needed for mapping main memory blocks into cache lines
- Three techniques can be used:

Direct

- The simplest technique
- Maps each block of main memory into only one possible cache line

Associative

- Permits each main memory block to be loaded into any line of the cache
- The cache control logic interprets a memory address simply as a Tag and a Word field
- To determine whether a block is in the cache, the cache control logic must simultaneously examine every line's Tag for a match

Set Associative

- A compromise that exhibits the strengths of both the direct and associative approaches while reducing their disadvantages

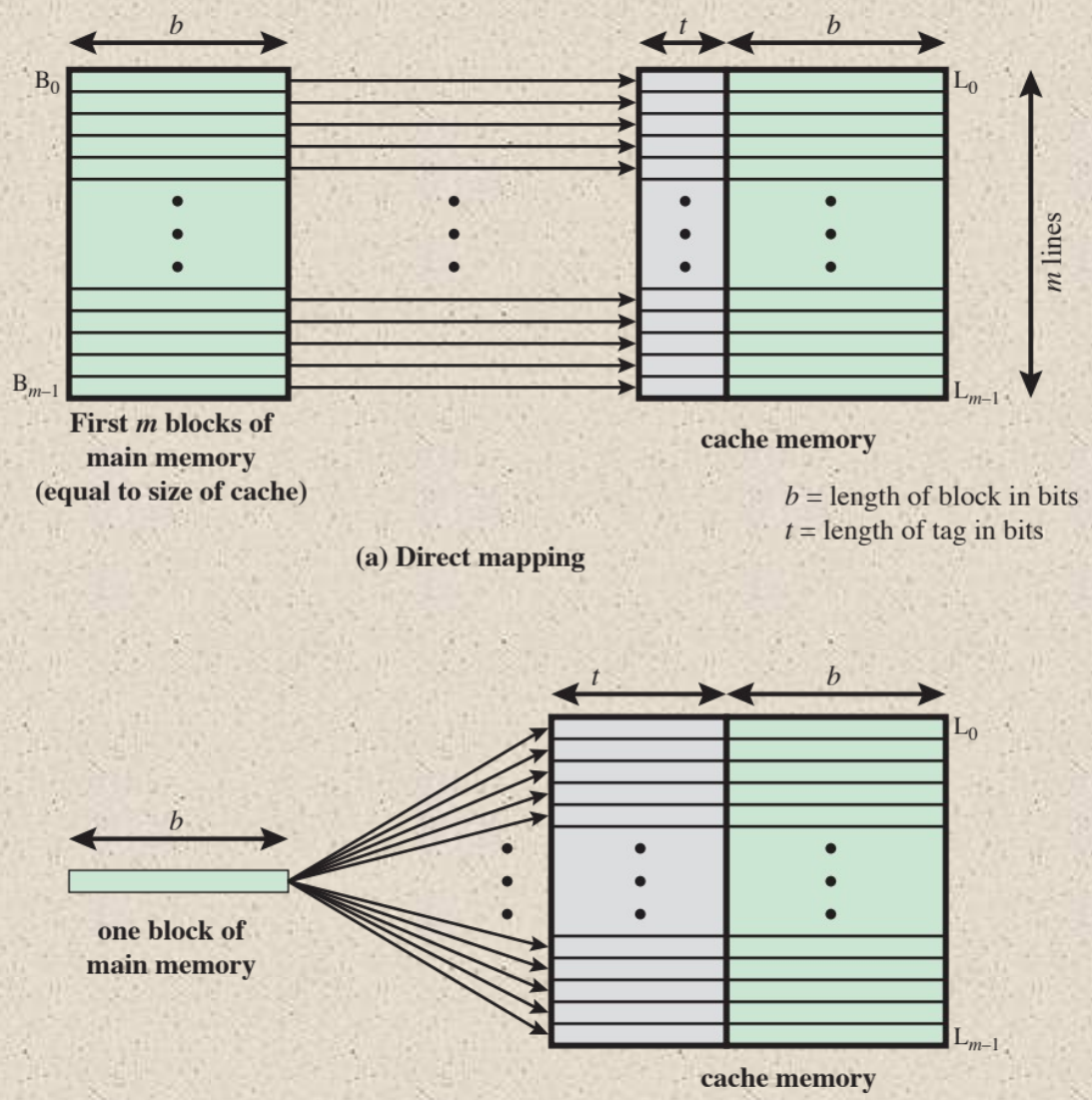


Figure 4.8 Mapping From Main Memory to Cache: Direct and Associative

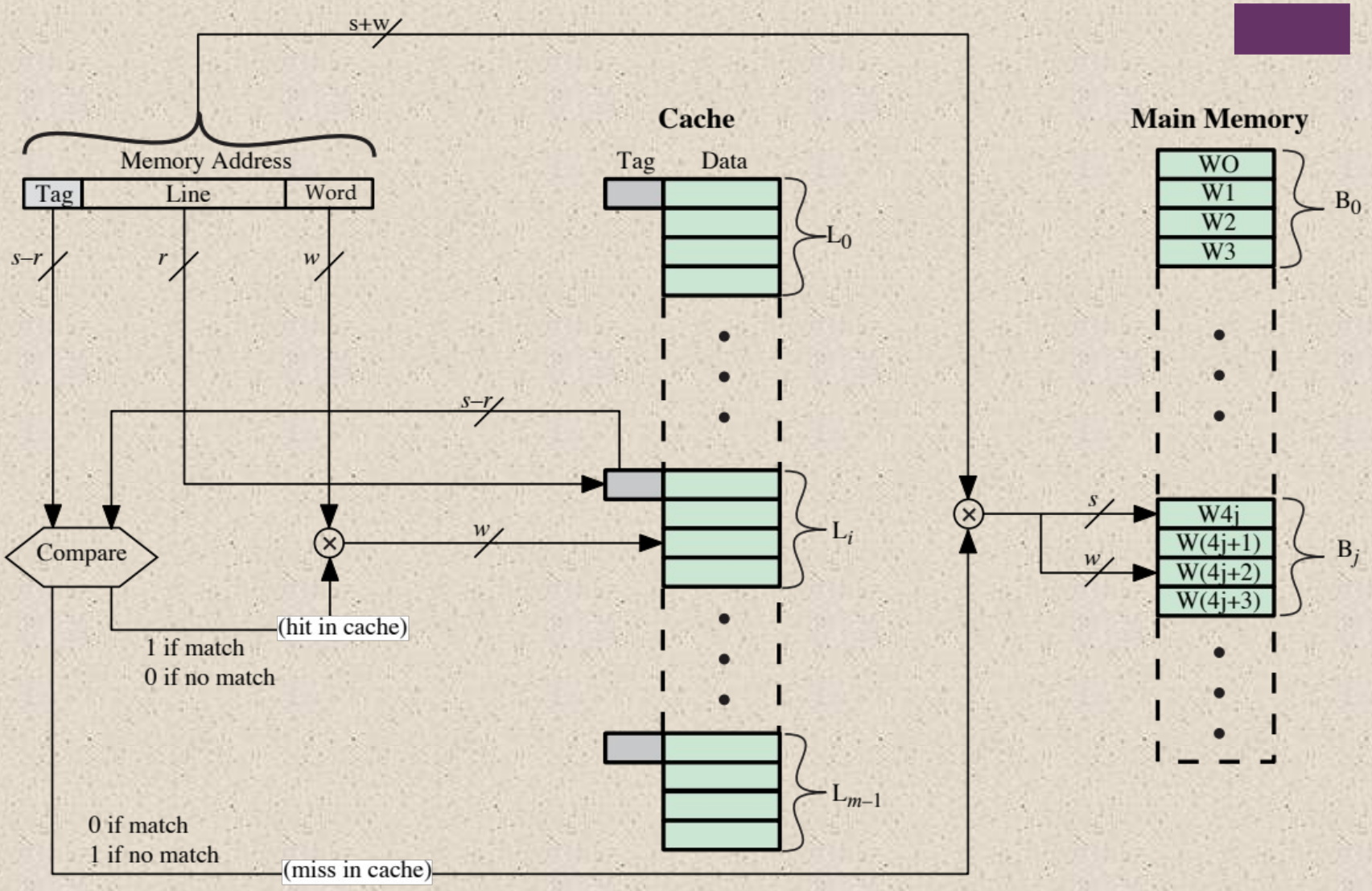


Figure 4.9 Direct-Mapping Cache Organization

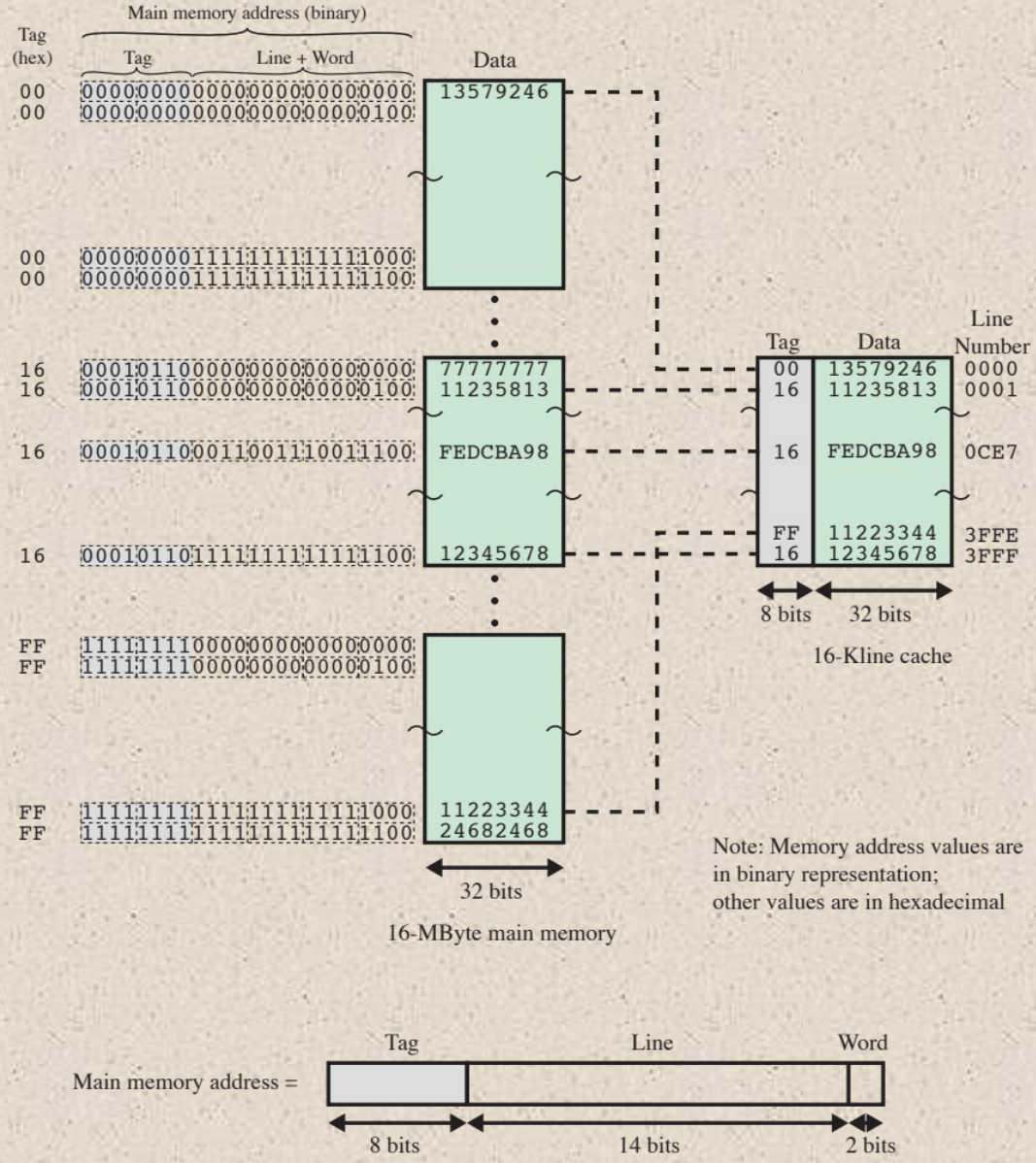
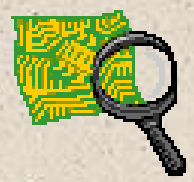


Figure 4.10 Direct Mapping Example

Direct Mapping Summary

- Address length = (s + w) bits
- Number of addressable units = 2^{s+w} words or bytes
- Block size = line size = 2^{w} words or bytes
- Number of blocks in main memory = $2^{s+w}/2^{w} = 2^{s}$
- Number of lines in cache = m = 2^{r}
- Size of tag = (s – r) bits



Victim Cache

- Originally proposed as an approach to reduce the conflict misses of direct mapped caches without affecting its fast access time
- Fully associative cache
- Typical size is 4 to 16 cache lines
- Residing between direct mapped L1 cache and the next level of memory

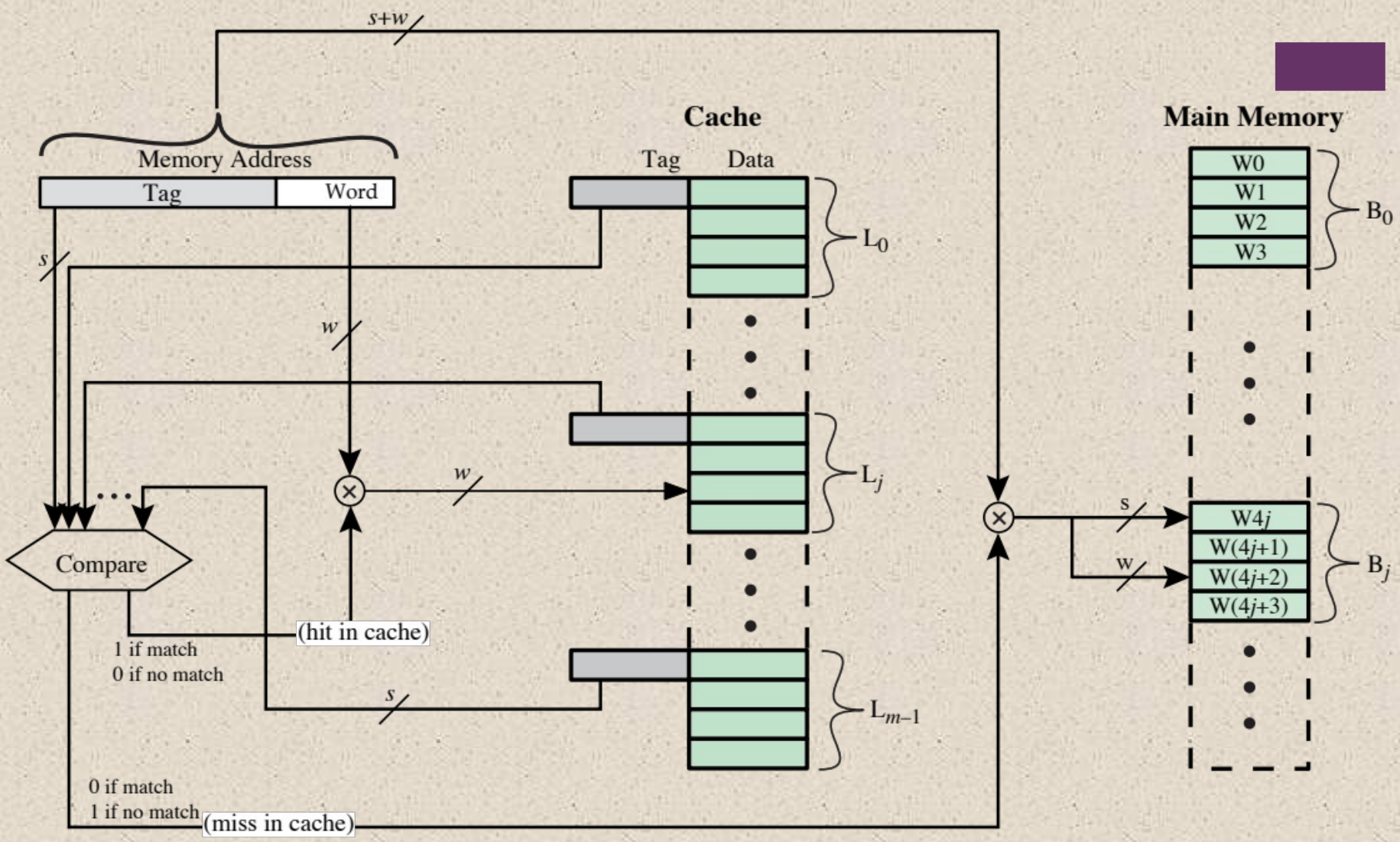


Figure 4.11 Fully Associative Cache Organization

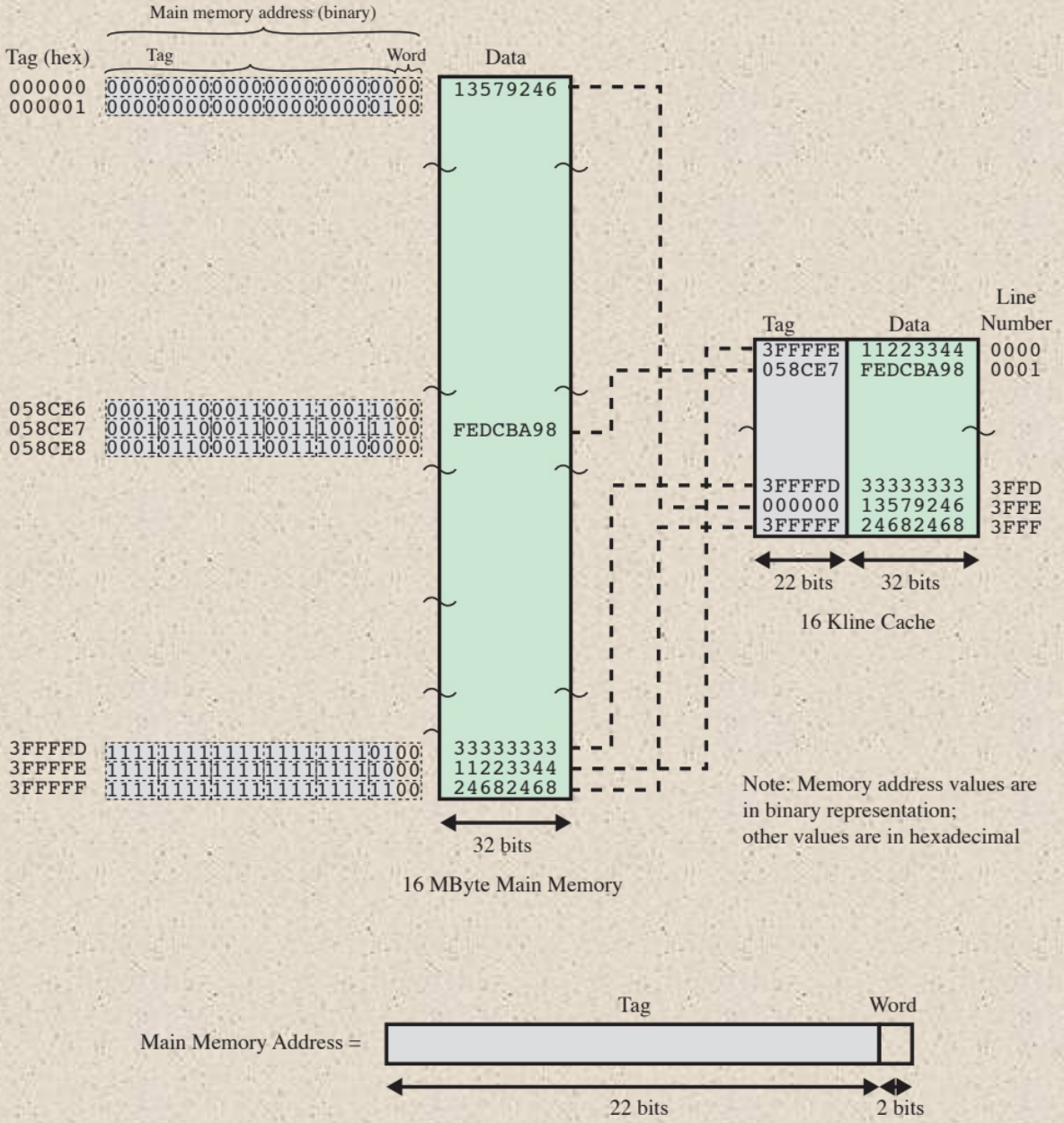
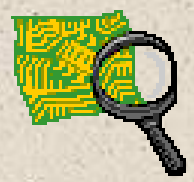


Figure 4.12 Associative Mapping Example

Associative Mapping Summary

- Address length = (s + w) bits
- Number of addressable units = 2^{s+w} words or bytes
- Block size = line size = 2^{w} words or bytes
- Number of blocks in main memory = $2^{s+w}/2^{w} = 2^{s}$
- Number of lines in cache = undetermined
- Size of tag = s bits



Set Associative Mapping

- Compromise that exhibits the strengths of both the direct and associative approaches while reducing their disadvantages
- Cache consists of a number of sets
- Each set contains a number of lines
- A given block maps to any line in a given set
- e.g. 2 lines per set
  - 2 way associative mapping
  - A given block can be in one of 2 lines in only one set

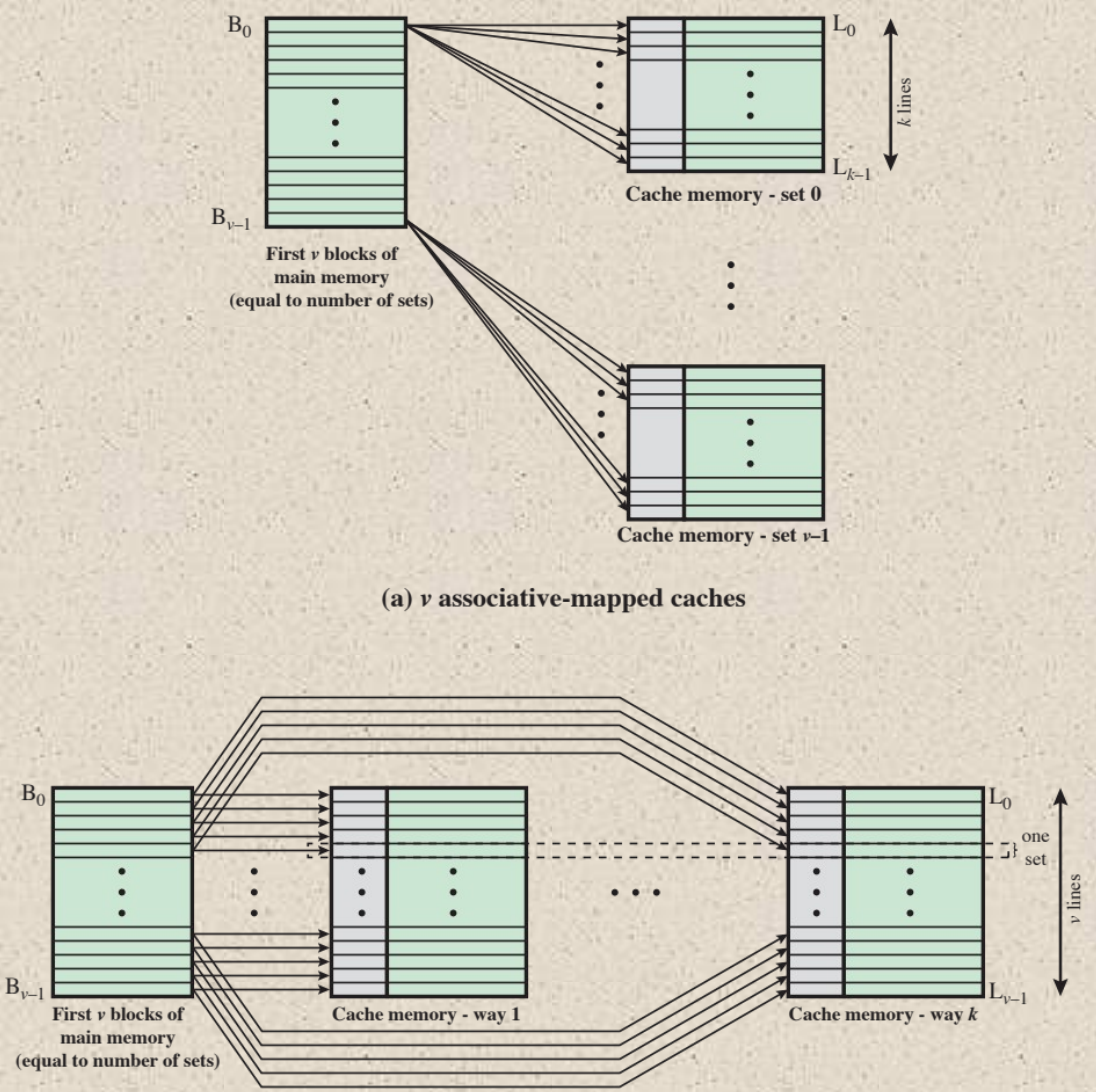


Figure 4.13 Mapping From Main Memory to Cache: *k*-way Set Associative

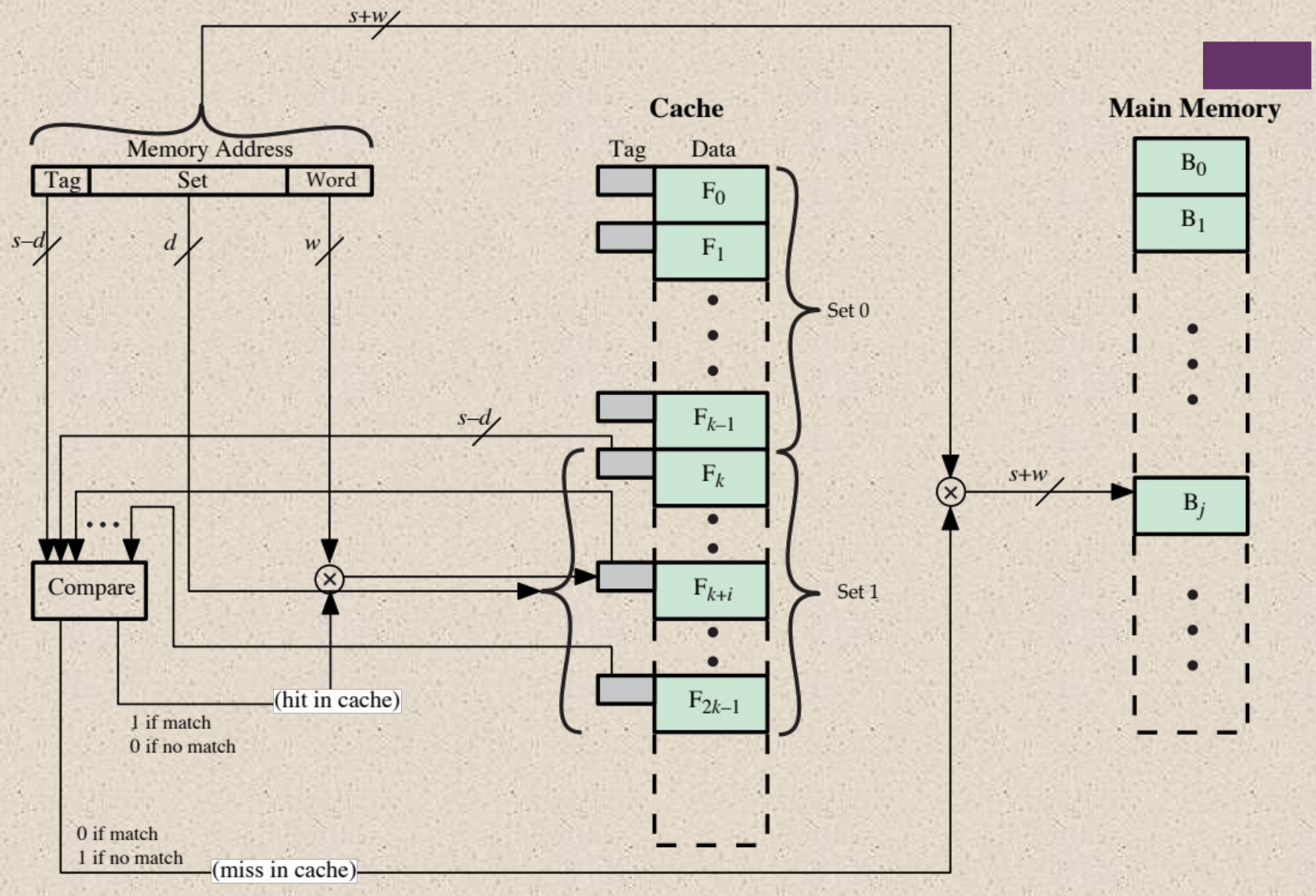
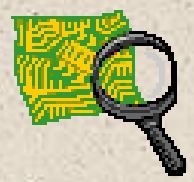


Figure 4.14 *k*-Way Set Associative Cache Organization

Set Associative Mapping Summary

- Address length = (s + w) bits
- Number of addressable units = 2^{s+w} words or bytes
- Block size = line size = 2^{w} words or bytes
- Number of blocks in main memory = $2^{s+w}/2^{w}=2^{s}$
- Number of lines in set = k
- Number of sets = v = 2^{d}
- Number of lines in cache = m=kv = k * 2^{d}
- Size of cache = $k * 2^{d+w}$ words or bytes
- Size of tag = (s – d) bits



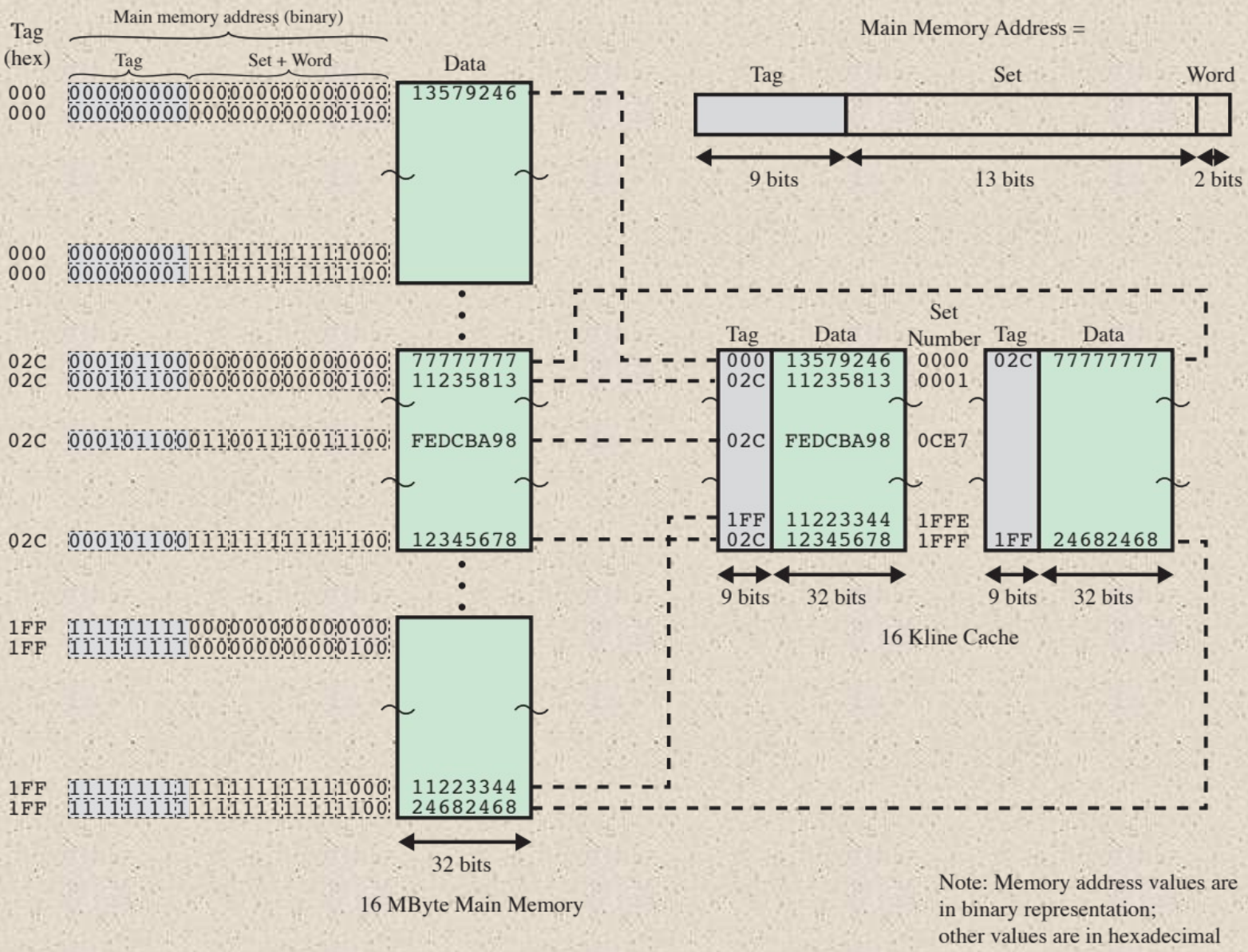


Figure 4.15 Two-Way Set Associative Mapping Example

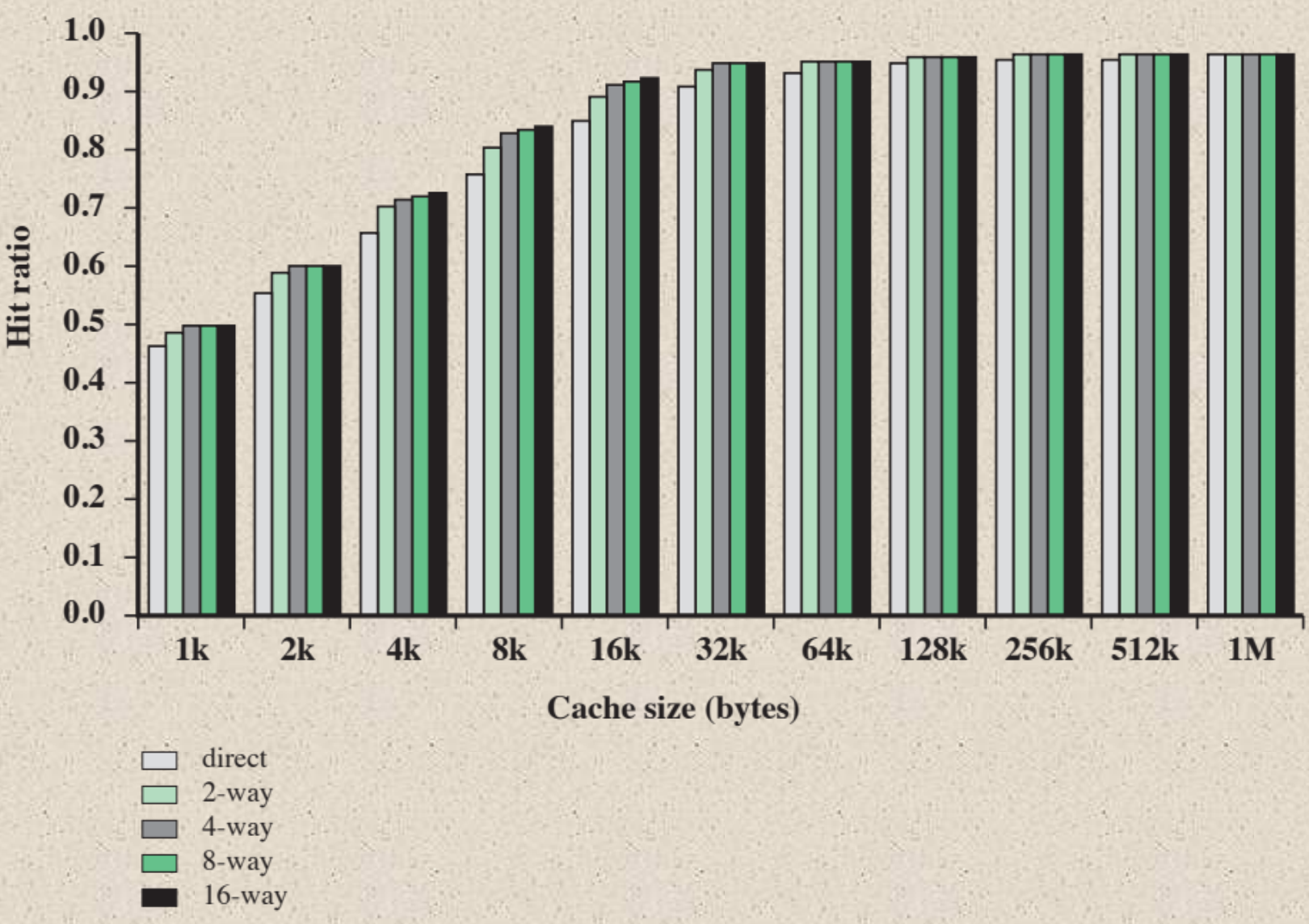
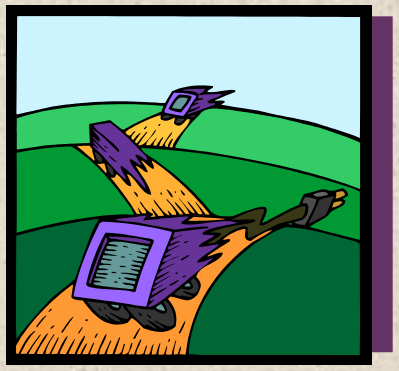


Figure 4.16 Varying Associativity over Cache Size

Replacement Algorithms



- Once the cache has been filled, when a new block is brought into the cache, one of the existing blocks must be replaced
- For direct mapping there is only one possible line for any particular block and no choice is possible
- For the associative and set-associative techniques a replacement algorithm is needed
- To achieve high speed, an algorithm must be implemented in hardware

The most common replacement algorithms are:

- Least recently used (LRU)
  - Most effective
  - Replace that block in the set that has been in the cache longest with no reference to it
  - Because of its simplicity of implementation, LRU is the most popular replacement algorithm
- First-in-first-out (FIFO)
  - Replace that block in the set that has been in the cache longest
  - Easily implemented as a round-robin or circular buffer technique
- Least frequently used (LFU)
  - Replace that block in the set that has experienced the fewest references
  - Could be implemented by associating a counter with each line

Write Policy

When a block that is resident in the cache is to be replaced there are two cases to consider:

If the old block in the cache has not been altered then it may be overwritten with a new block without first writing out the old block

If at least one write operation has been performed on a word in that line of the cache then main memory must be updated by writing the line of cache out to the block of memory before bringing in the new block

There are two problems to contend with:

More than one device may have access to main memory

A more complex problem occurs when multiple processors are attached to the same bus and each processor has its own local cache - if a word is altered in one cache it could conceivably invalidate a word in other caches

Write Through and Write Back

- Write through
  - Simplest technique
  - All write operations are made to main memory as well as to the cache
  - The main disadvantage of this technique is that it generates substantial memory traffic and may create a bottleneck
- Write back
  - Minimizes memory writes
  - Updates are made only in the cache
  - Portions of main memory are invalid and hence accesses by I/O modules can be allowed only through the cache
  - This makes for complex circuitry and a potential bottleneck

Line Size

When a block of data is retrieved and placed in the cache not only the desired word but also some number of adjacent words are retrieved

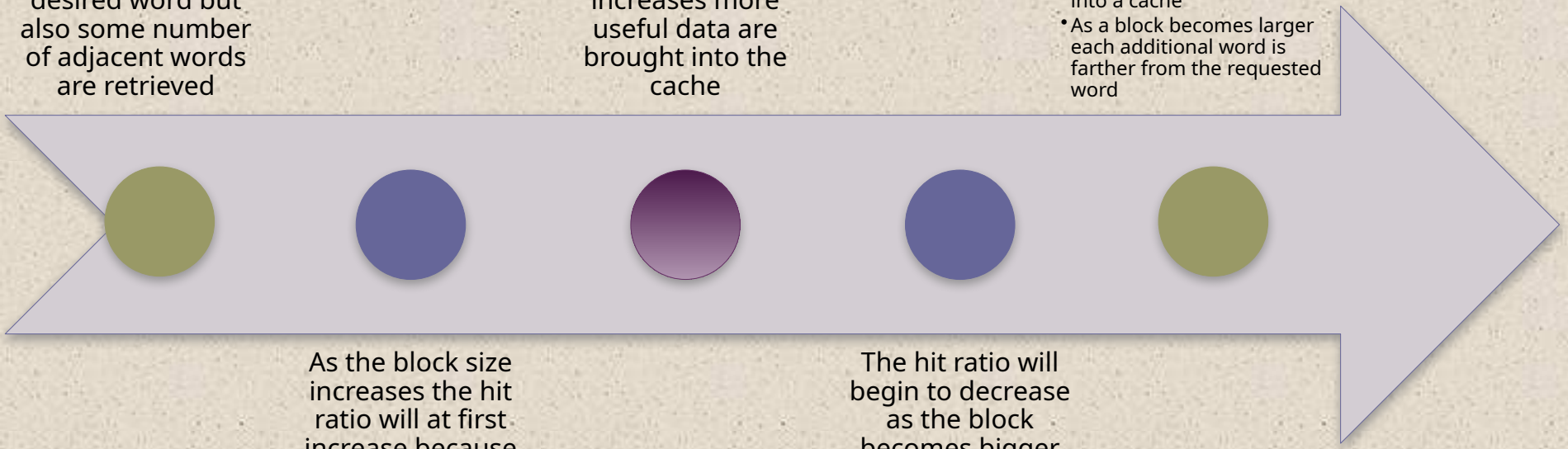
As the block size increases the hit ratio will at first increase because of the principle of locality

As the block size increases more useful data are brought into the cache

The hit ratio will begin to decrease as the block becomes bigger and the probability of using the newly fetched information becomes less than the probability of reusing the information that has to be replaced

Two specific effects come into play:

- Larger blocks reduce the number of blocks that fit into a cache
- As a block becomes larger each additional word is farther from the requested word



Multilevel Caches

- As logic density has increased it has become possible to have a cache on the same chip as the processor
- The on-chip cache reduces the processor's external bus activity and speeds up execution time and increases overall system performance
  - When the requested instruction or data is found in the on-chip cache, the bus access is eliminated
  - On-chip cache accesses will complete appreciably faster than would even zero-wait state bus cycles
  - During this period the bus is free to support other transfers
- Two-level cache:
  - Internal cache designated as level 1 (L1)
  - External cache designated as level 2 (L2)
- Potential savings due to the use of an L2 cache depends on the hit rates in both the L1 and L2 caches
- The use of multilevel caches complicates all of the design issues related to caches, including size, replacement algorithm, and write policy

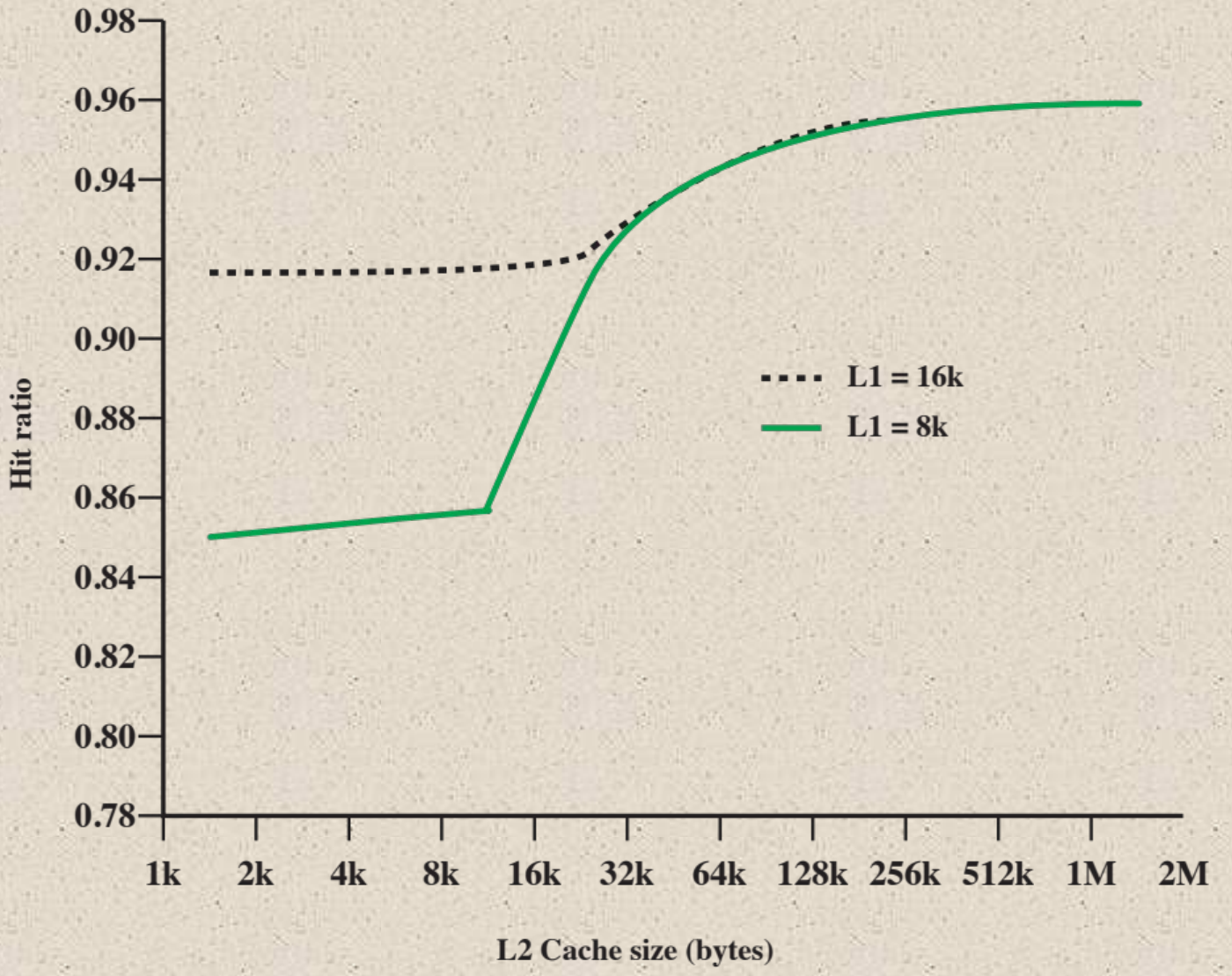


Figure 4.17 Total Hit Ratio (L1 and L2) for 8 Kbyte and 16 Kbyte L1

Unified Versus Split Caches

- Has become common to split cache:
  - One dedicated to instructions
  - One dedicated to data
  - Both exist at the same level, typically as two L1 caches
- Advantages of unified cache:
  - Higher hit rate
    - Balances load of instruction and data fetches automatically
    - Only one cache needs to be designed and implemented
- Trend is toward split caches at the L1 and unified caches for higher levels
- Advantages of split cache:
  - Eliminates cache contention between instruction fetch/decode unit and execution unit
    - Important in pipelining

Problem	Solution	Processor on which Feature First Appears
External memory slower than the system bus.	Add external cache using faster memory technology.	386
Increased processor speed results in external bus becoming a bottleneck for cache access.	Move external cache on-chip, operating at the same speed as the processor.	486
Internal cache is rather small, due to limited space on chip	Add external L2 cache using faster technology than main memory	486
Contention occurs when both the Instruction Prefetcher and the Execution Unit simultaneously require access to the cache. In that case, the Prefetcher is stalled while the Execution Unit's data access takes place.	Create separate data and instruction caches.	Pentium
Increased processor speed results in external bus becoming a bottleneck for L2 cache access.	Create separate back-side bus that runs at higher speed than the main (front-side) external bus. The BSB is dedicated to the L2 cache.	Pentium Pro
	Move L2 cache on to the processor chip.	Pentium II
Some applications deal with massive databases and must have rapid access to large amounts of data. The on-chip caches are too small.	Add external L3 cache.	Pentium III
	Move L3 cache on-chip.	Pentium 4

Table 4.4

Intel Cache Evolution

(Table is on page 150 in the textbook.)

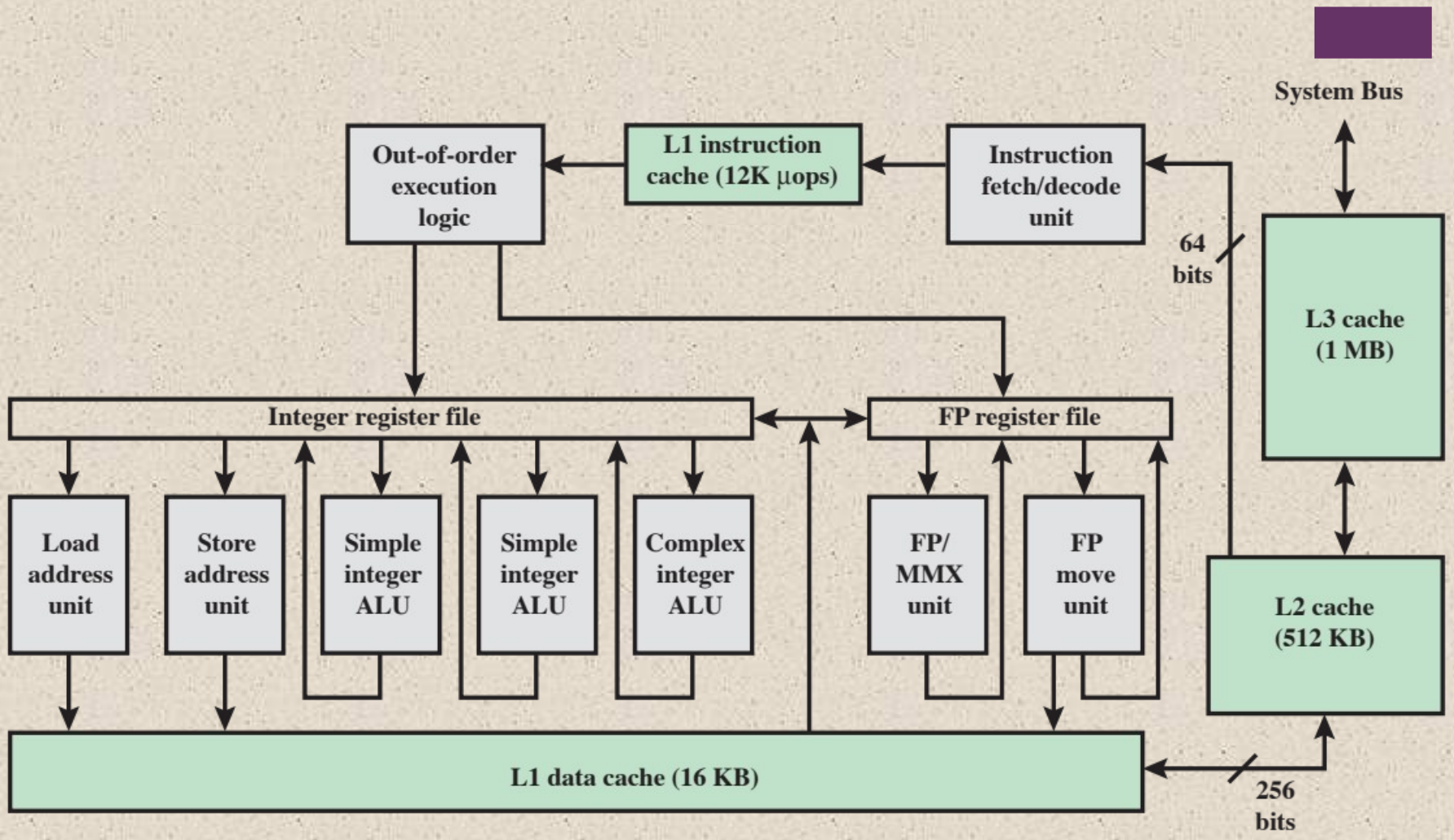


Figure 4.18 Pentium 4 Block Diagram

Table 4.5 Pentium 4 Cache Operating Modes

Control Bits		Operating Mode		
CD	NW	Cache Fills	Write Throughs	Invalidates
0	0	Enabled	Enabled	Enabled
1	0	Disabled	Enabled	Enabled
1	1	Disabled	Disabled	Disabled

Note: CD = 0; NW = 1 is an invalid combination.

Summary

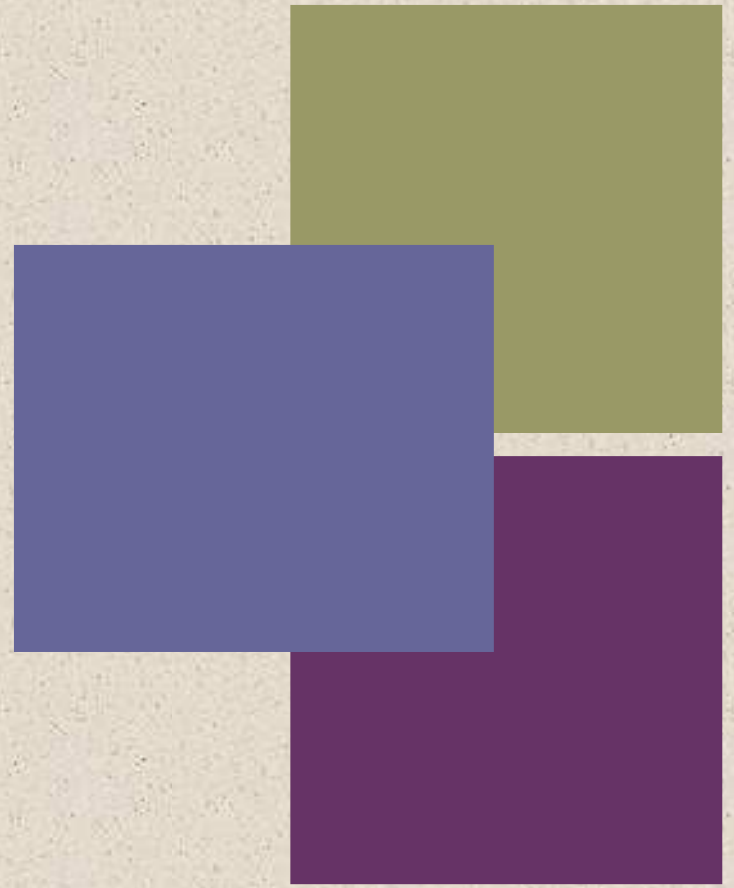
Chapter 4

Cache Memory

- Computer memory system overview
  - Characteristics of Memory Systems
  - Memory Hierarchy
- Cache memory principles
- Pentium 4 cache organization

- Elements of cache design
  - Cache addresses
  - Cache size
  - Mapping function
  - Replacement algorithms
  - Write policy
  - Line size
  - Number of caches

William Stallings
Computer Organization and Architecture
10th Edition



Chapter 5
Internal Memory

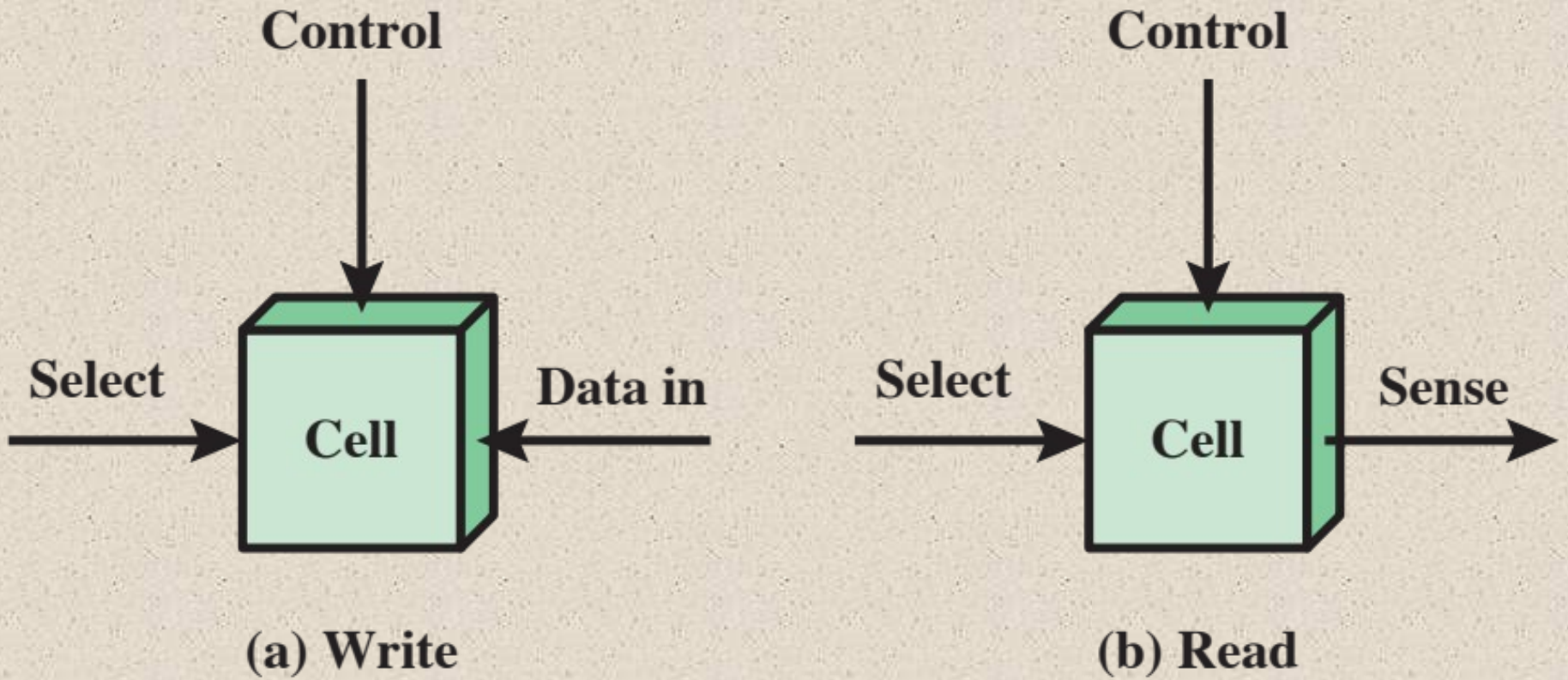


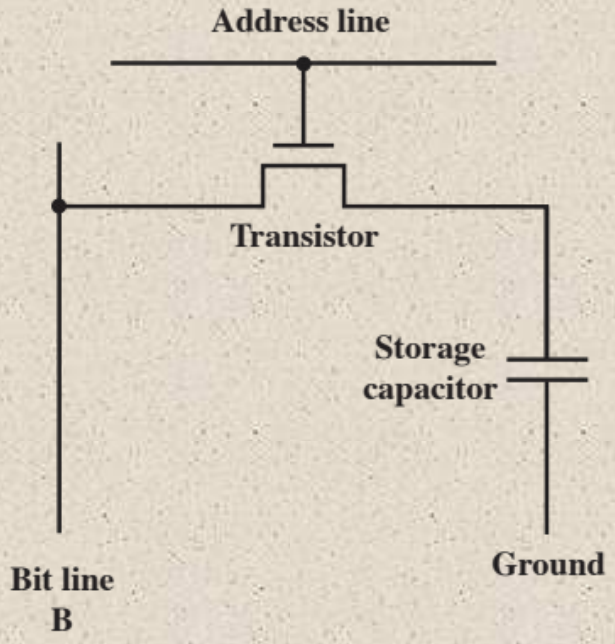
Figure 5.1 Memory Cell Operation

Memory Type	Category	Erasure	Write Mechanism	Volatility
Random-access memory (RAM)	Read-write memory	Electrically, byte-level	Electrically	Volatile
Read-only memory (ROM)	Read-only memory	Not possible	Masks	Nonvolatile
Programmable ROM (PROM)				
Erasable PROM (EPROM)	Read-mostly memory	UV light, chip-level	Electrically	
Electrically Erasable PROM (EEPROM)		Electrically, byte-level		
Flash memory		Electrically, block-level		

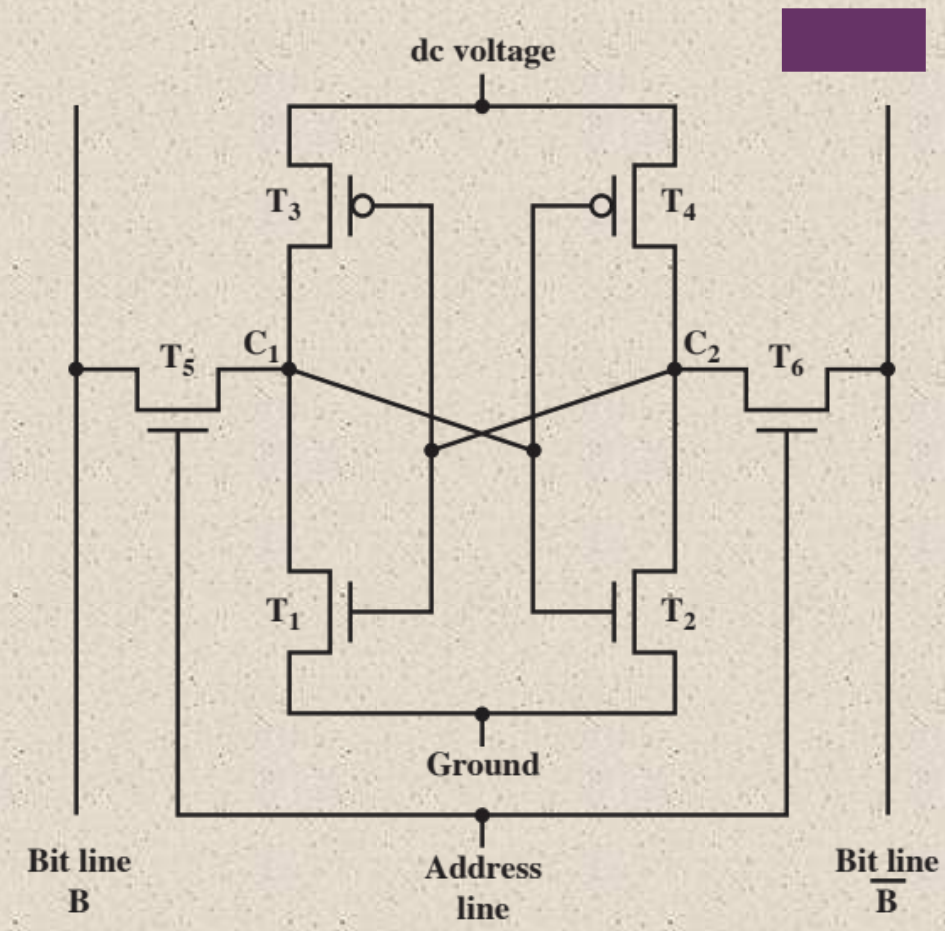
Table 5.1
Semiconductor Memory Types

Dynamic RAM (DRAM)

- RAM technology is divided into two technologies:
  - Dynamic RAM (DRAM)
  - Static RAM (SRAM)
- DRAM
  - Made with cells that store data as charge on capacitors
  - Presence or absence of charge in a capacitor is interpreted as a binary 1 or 0
  - Requires periodic charge refreshing to maintain data storage
  - The term *dynamic* refers to tendency of the stored charge to leak away, even with power continuously applied



(a) Dynamic RAM (DRAM) cell

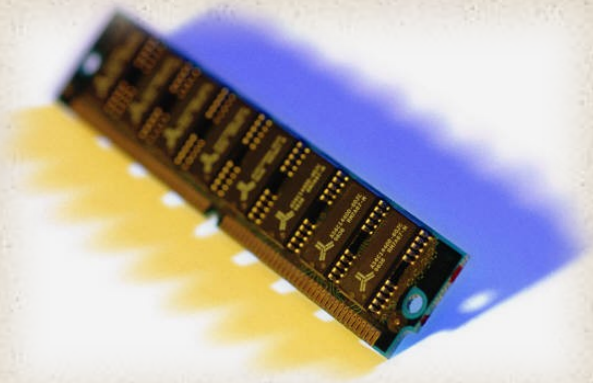


(b) Static RAM (SRAM) cell

Figure 5.2 Typical Memory Cell Structures

Static RAM (SRAM)

- Digital device that uses the same logic elements used in the processor
- Binary values are stored using traditional flip-flop logic gate configurations
- Will hold its data as long as power is supplied to it



SRAM versus DRAM

- Both volatile
  - Power must be continuously supplied to the memory to preserve the bit values
- Dynamic cell
  - Simpler to build, smaller
  - More dense (smaller cells = more cells per unit area)
  - Less expensive
  - Requires the supporting refresh circuitry
  - Tend to be favored for large memory requirements
  - Used for main memory
- Static
  - Faster
  - Used for cache memory (both on and off chip)

SRAM

DRAM

Read Only Memory (ROM)

- Contains a permanent pattern of data that cannot be changed or added to
- No power source is required to maintain the bit values in memory
- Data or program is permanently in main memory and never needs to be loaded from a secondary storage device
- Data is actually wired into the chip as part of the fabrication process
  - Disadvantages of this:
    - No room for error, if one bit is wrong the whole batch of ROMs must be thrown out
    - Data insertion step includes a relatively large fixed cost

Programmable ROM (PROM)

- Less expensive alternative
- Nonvolatile and may be written into only once
- Writing process is performed electrically and may be performed by supplier or customer at a time later than the original chip fabrication
- Special equipment is required for the writing process
- Provides flexibility and convenience
- Attractive for high volume production runs

Read-Mostly Memory

EPROM

- Erasable programmable read-only memory
- Erasure process can be performed repeatedly
- More expensive than PROM but it has the advantage of the multiple update capability

EEPROM

- Electrically erasable programmable read-only memory
- Can be written into at any time without erasing prior contents
- Combines the advantage of non-volatility with the flexibility of being updatable in place
- More expensive than EPROM

Flash Memory

- Intermediate between EPROM and EEPROM in both cost and functionality
- Uses an electrical erasing technology, does not provide byte-level erasure
- Microchip is organized so that a section of memory cells are erased in a single action or “flash”

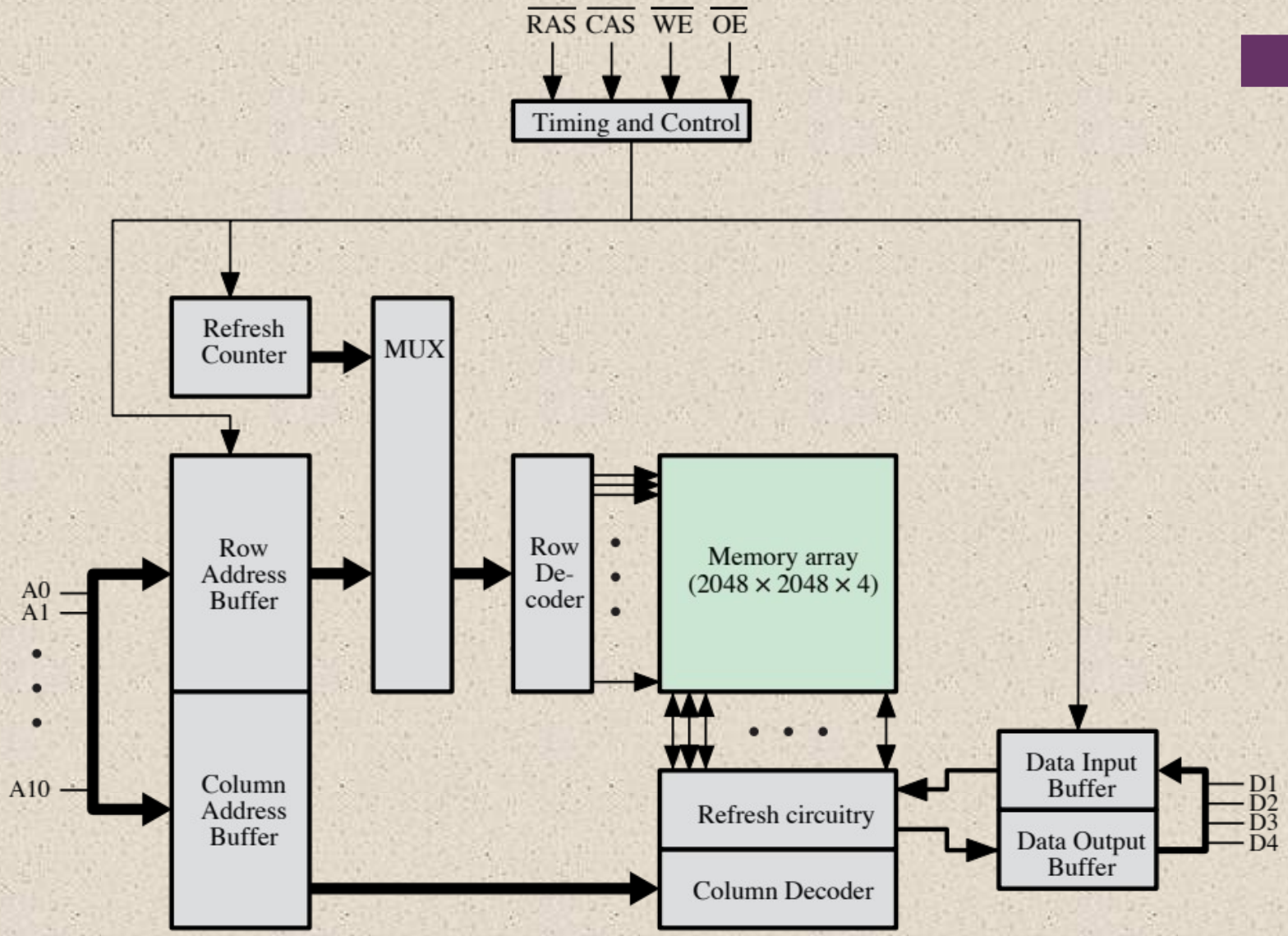
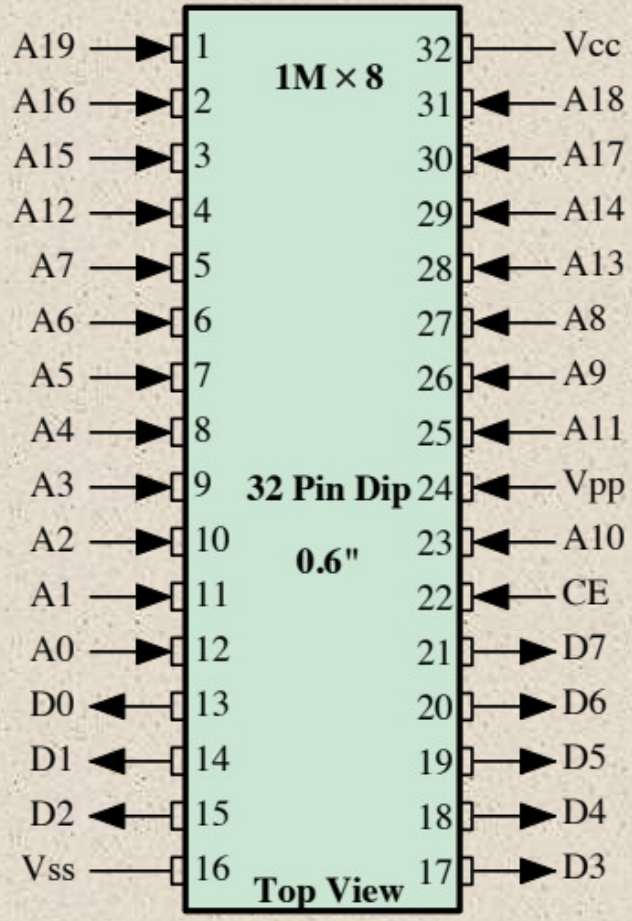
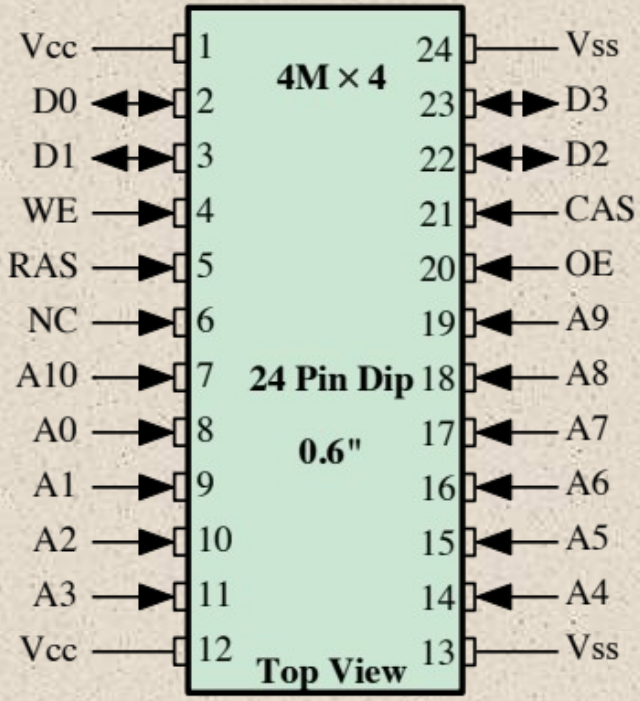


Figure 5.3 Typical 16 Megabit DRAM (4M × 4)



(a) 8 Mbit EPROM



(b) 16 Mbit DRAM

Figure 5.4 Typical Memory Package Pins and Signals

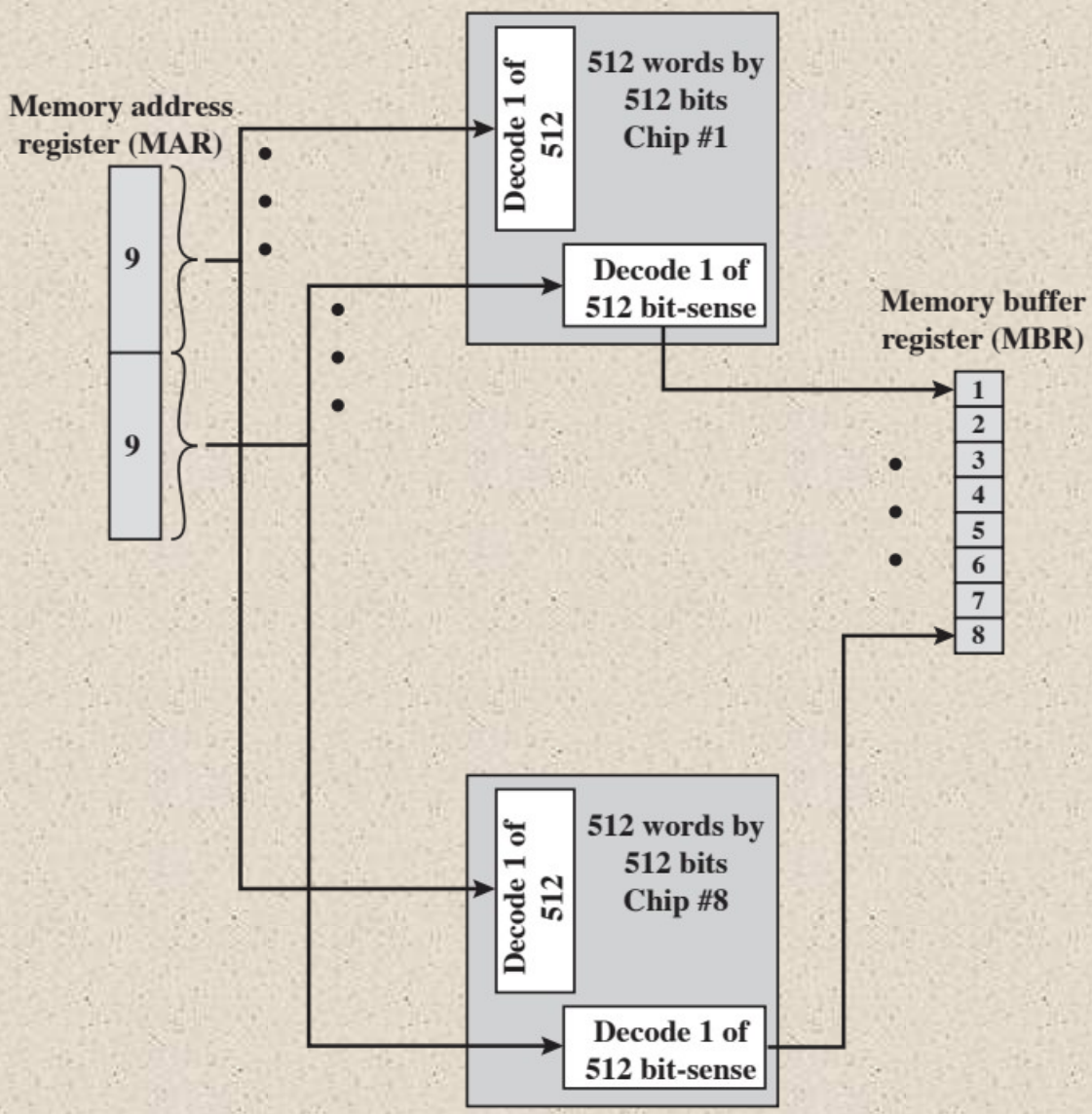


Figure 5.5 256-KByte Memory Organization

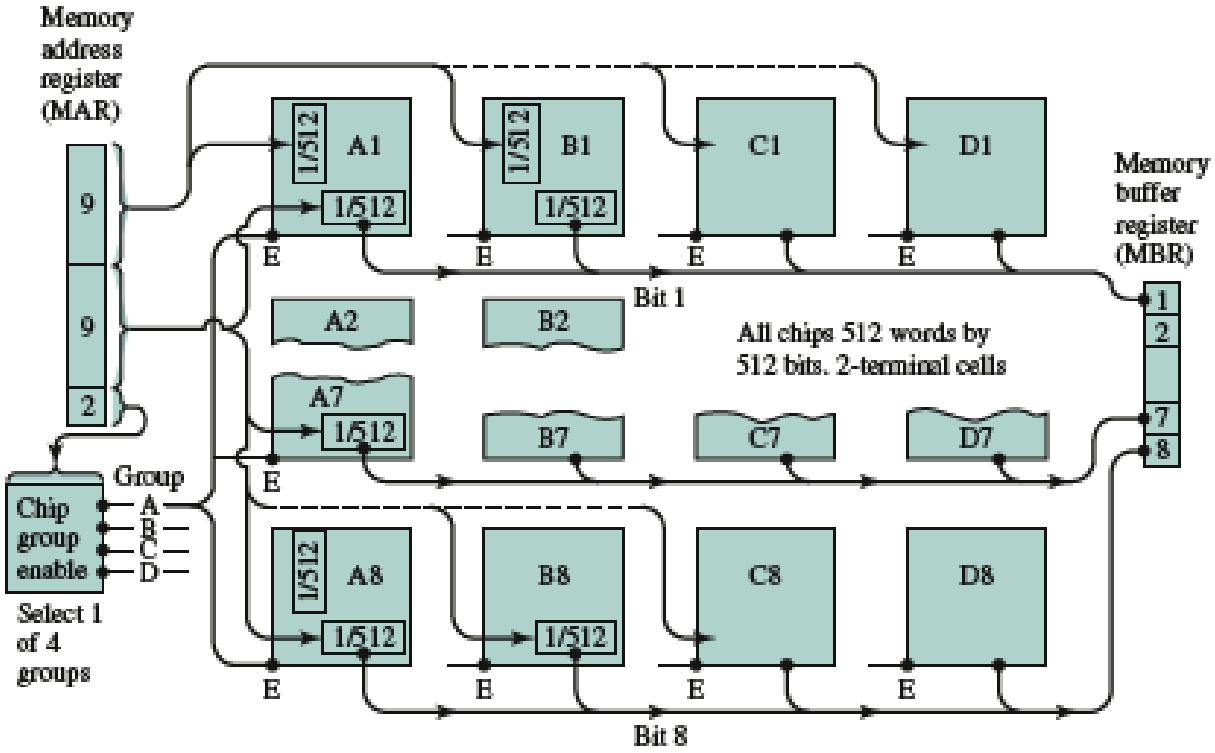
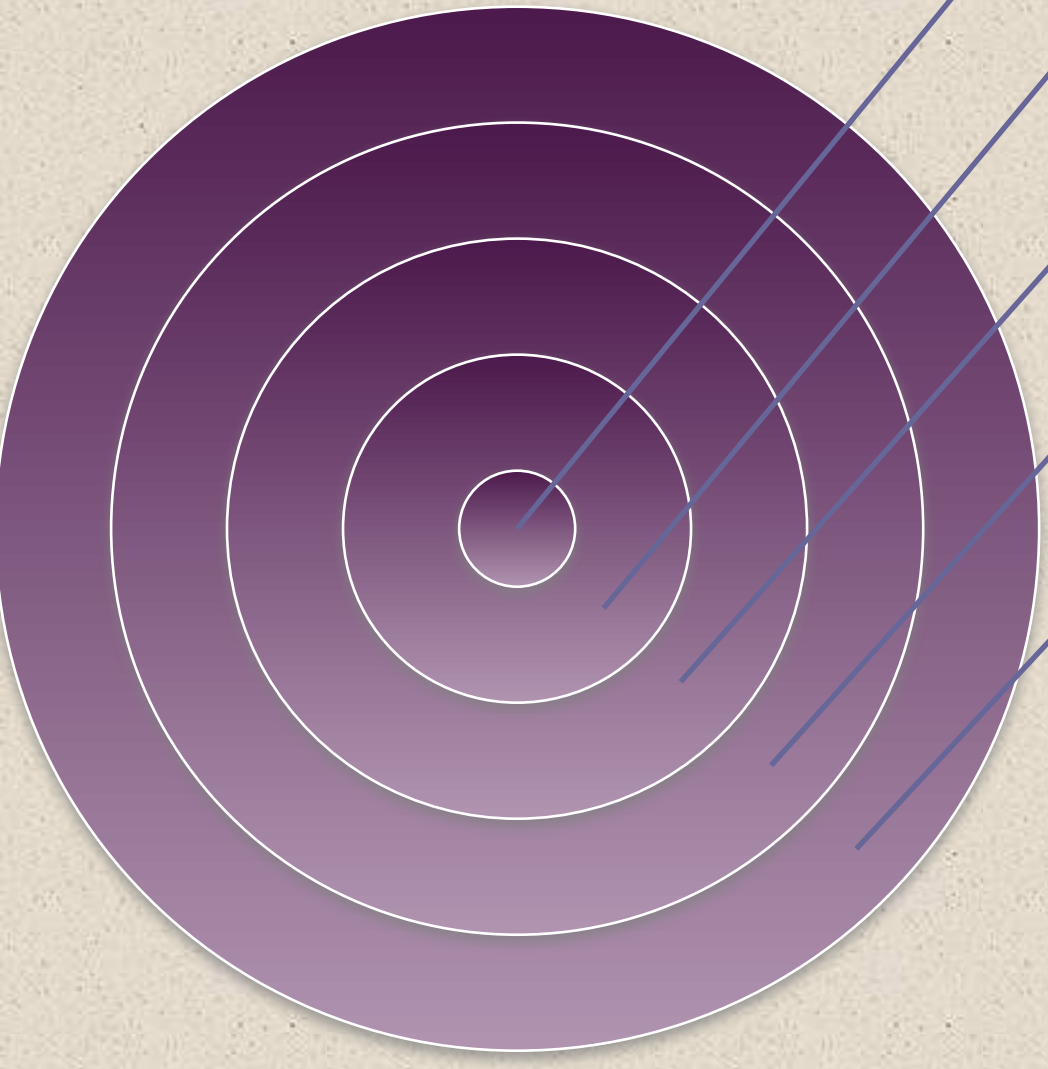


Figure 5.6 1-Mbyte Memory Organization

Interleaved Memory



Composed of a collection of DRAM chips

Grouped together to form a *memory bank*

Each bank is independently able to service a memory read or write request

K banks can service *K* requests simultaneously, increasing memory read or write rates by a factor of *K*

If consecutive words of memory are stored in different banks, the transfer of a block of memory is speeded up

Error Correction

- Hard Failure
  - Permanent physical defect
  - Memory cell or cells affected cannot reliably store data but become stuck at 0 or 1 or switch erratically between 0 and 1
  - Can be caused by:
    - Harsh environmental abuse
    - Manufacturing defects
    - Wear
- Soft Error
  - Random, non-destructive event that alters the contents of one or more memory cells
  - No permanent damage to memory
  - Can be caused by:
    - Power supply problems
    - Alpha particles

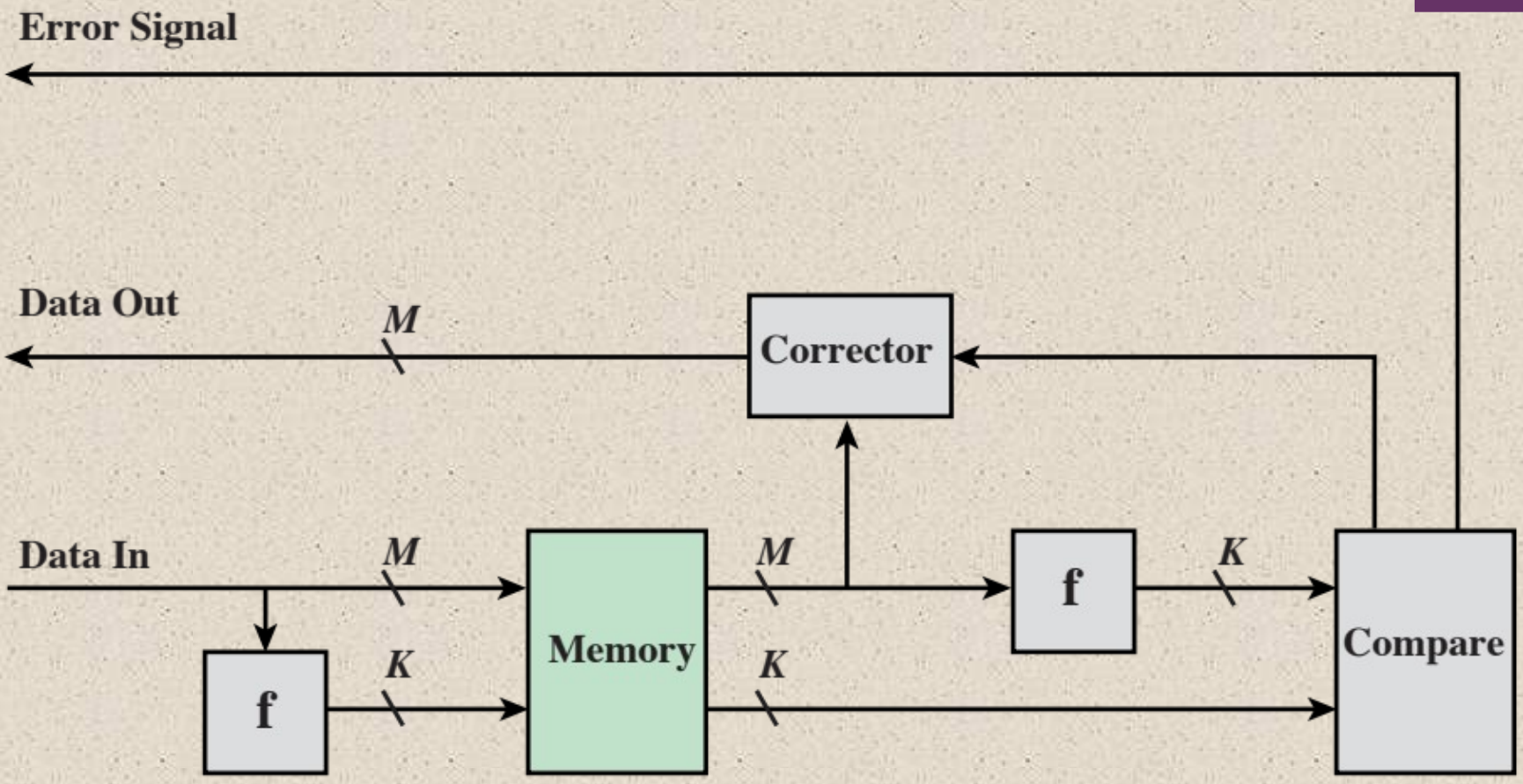


Figure 5.7 Error-Correcting Code Function

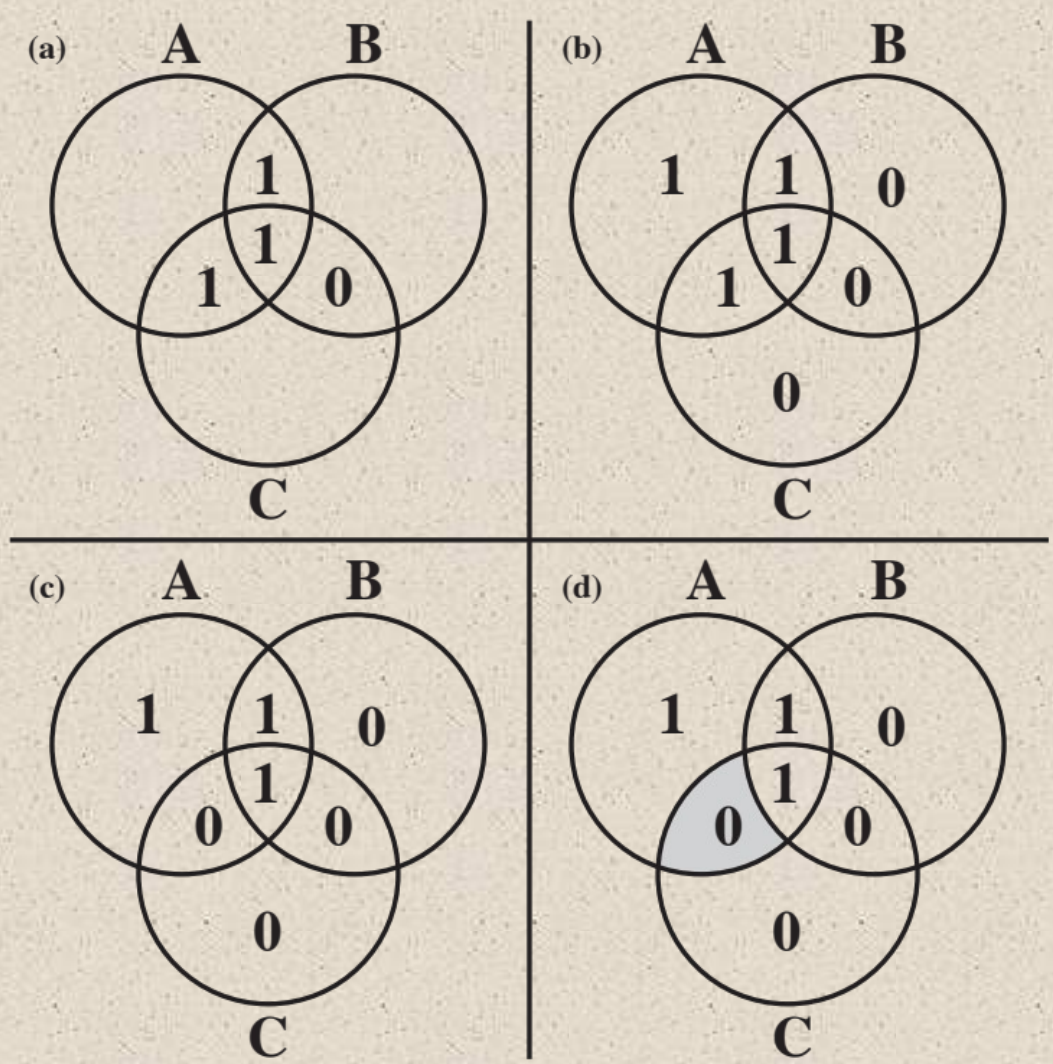


Figure 5.8 Hamming Error-Correcting Code

	Single-Error Correction		Single-Error Correction/ Double-Error Detection	
Data Bits	Check Bits	% Increase	Check Bits	% Increase
8	4	50	5	62.5
16	5	31.25	6	37.5
32	6	18.75	7	21.875
64	7	10.94	8	12.5
128	8	6.25	9	7.03
256	9	3.52	10	3.91

Table 5.2
Increase in Word Length with Error Correction

Bit Position	12	11	10	9	8	7	6	5	4	3	2	1
Position Number	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Data Bit	D8	D7	D6	D5		D4	D3	D2		D1		
Check Bit					C8				C4		C2	C1

Figure 5.9 Layout of Data Bits and Check Bits

Bit position	12	11	10	9	8	7	6	5	4	3	2	1
Position number	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Data bit	D8	D7	D6	D5		D4	D3	D2		D1		
Check bit					C8				C4		C2	C1
Word stored as	0	0	1	1	0	1	0	0	1	1	1	1
Word fetched as	0	0	1	1	0	1	1	0	1	1	1	1
Position Number	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Check Bit					0				0		0	1

Figure 5.10 Check Bit Calculation

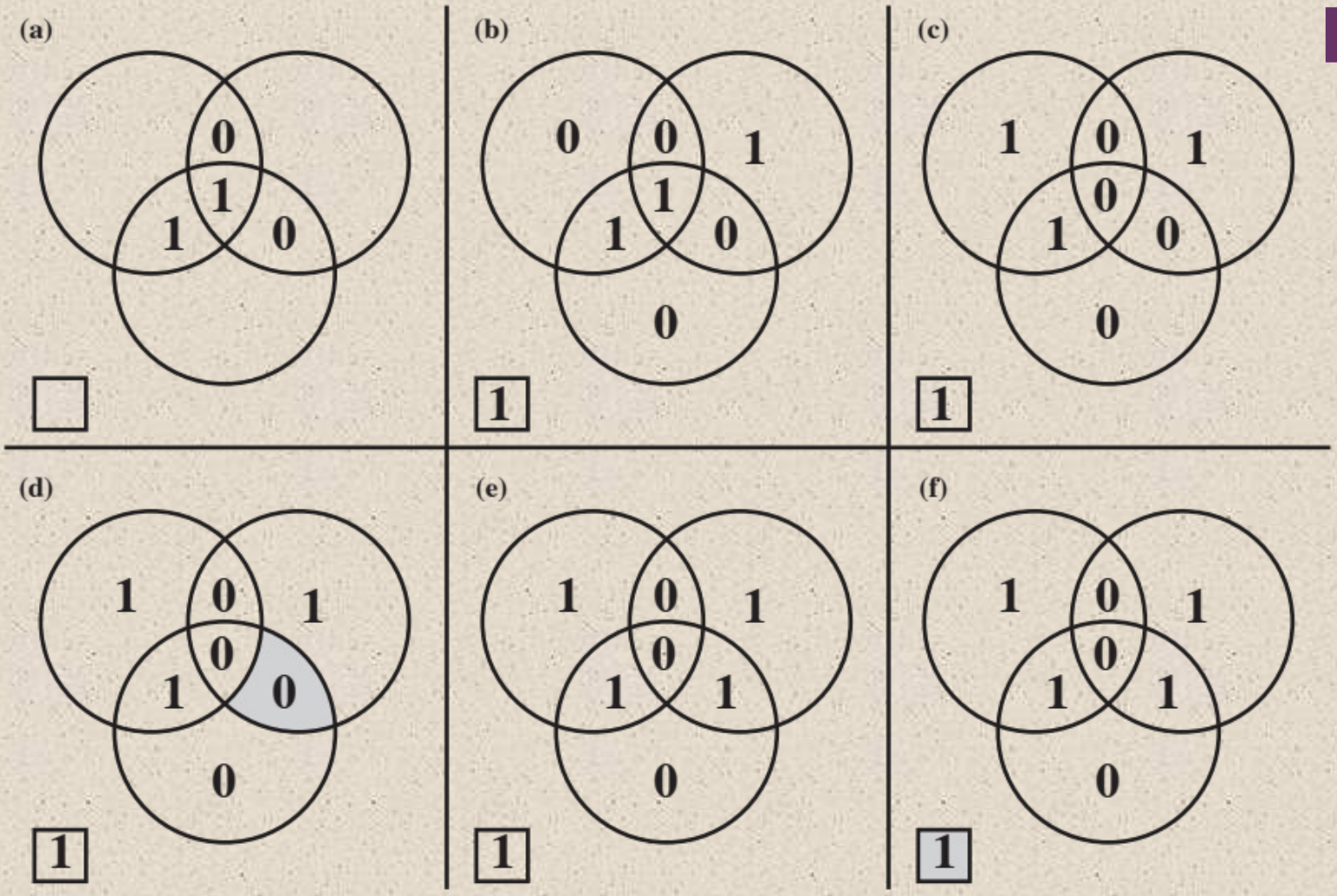


Figure 5.11 Hamming SEC-DED Code

Advanced DRAM Organization

- One of the most critical system bottlenecks when using high-performance processors is the interface to main internal memory
- The traditional DRAM chip is constrained both by its internal architecture and by its interface to the processor's memory bus
- A number of enhancements to the basic DRAM architecture have been explored
  - The schemes that currently dominate the market are SDRAM and DDR-DRAM

Synchronous DRAM (SDRAM)

One of the most widely used forms of DRAM

Exchanges data with the processor synchronized to an external clock signal and running at the full speed of the processor/memory bus without imposing wait states

With synchronous access the DRAM moves data in and out under control of the system clock

- The processor or other master issues the instruction and address information which is latched by the DRAM
- The DRAM then responds after a set number of clock cycles
- Meanwhile the master can safely do other tasks while the SDRAM is processing

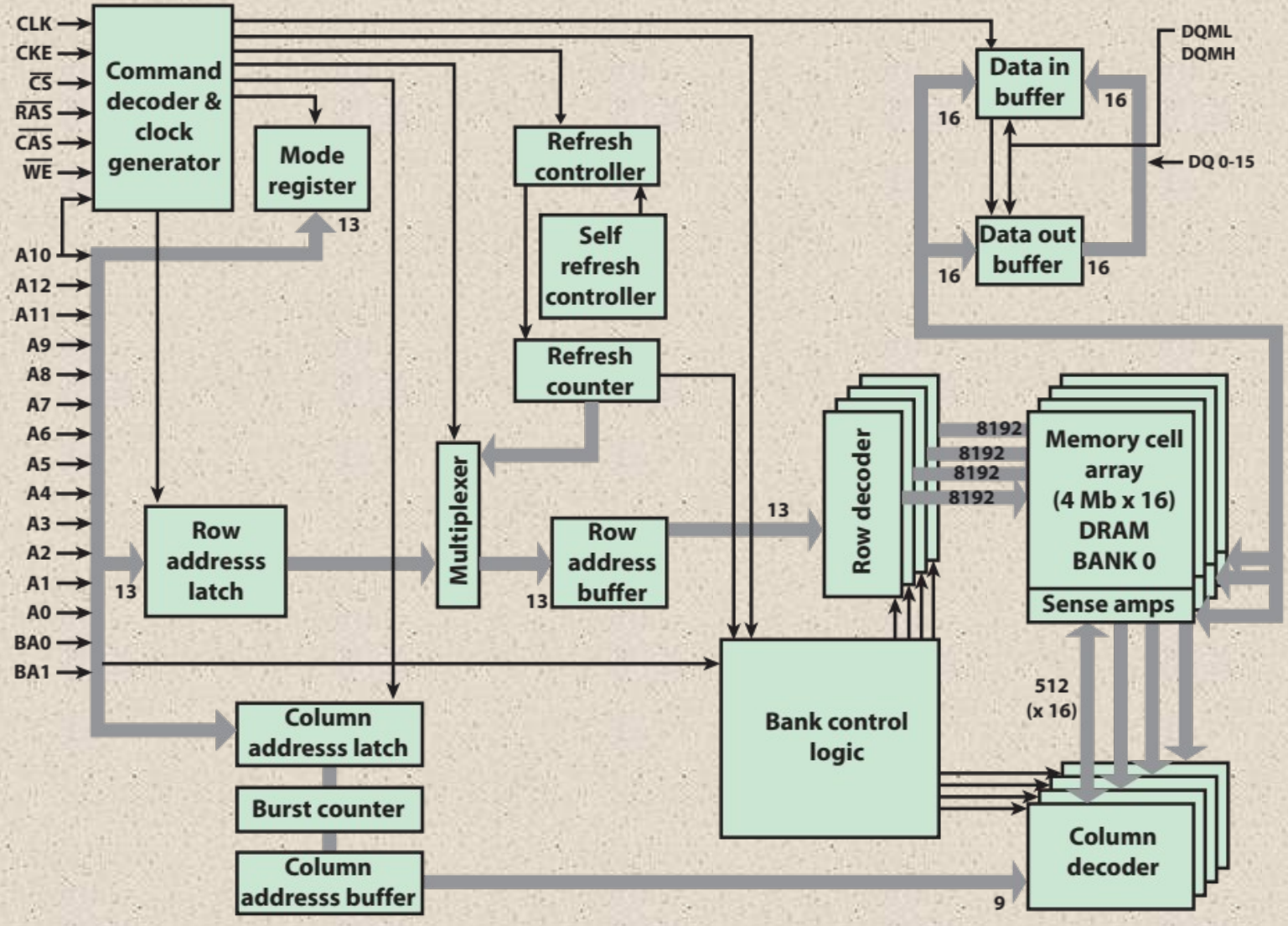


Figure 5.12 256-Mb Synchronous Dynamic RAM (SDRAM)

A0 to A12	Address inputs
BA0, BA1	Bank address lines
CLK	Clock input
CKE	Clock enable
$\overline{CS}$	Chip select
$\overline{RAS}$	Row address strobe
$\overline{CAS}$	Column address strobe
$\overline{WE}$	Write enable
DQ0 to DQ15	Data input/output
DQM	Data mask

Table 5.3

SDRAM Pin Assignments

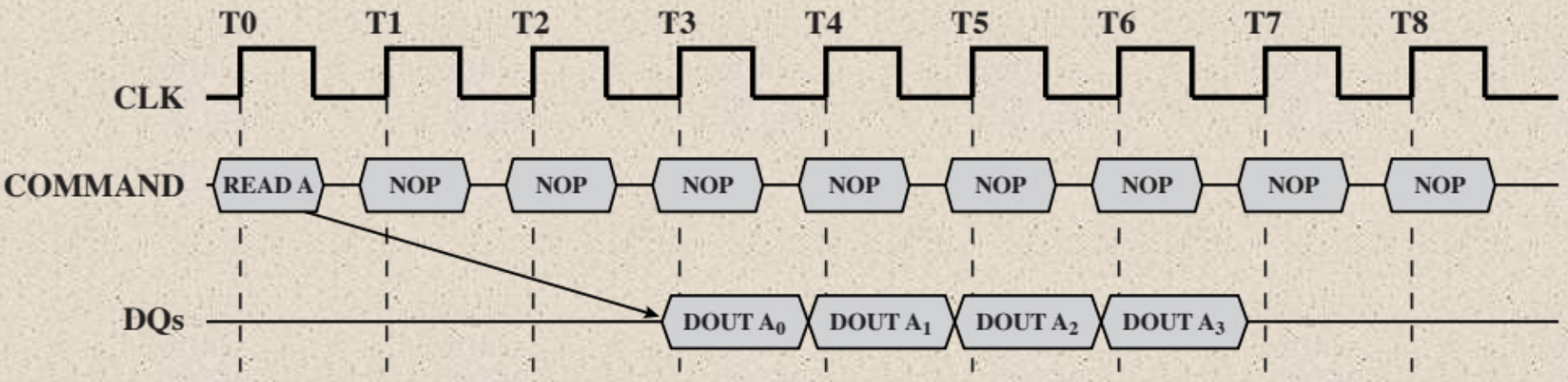


Figure 5.13 SDRAM Read Timing (Burst Length = 4, $\overline{\text{CAS}}$ latency = 2)

Double Data Rate SDRAM (DDR SDRAM)

- Developed by the JEDEC Solid State Technology Association (Electronic Industries Alliance's semiconductor-engineering-standardization body)
- Numerous companies make DDR chips, which are widely used in desktop computers and servers
- DDR achieves higher data rates in three ways:
  - First, the data transfer is synchronized to both the rising and falling edge of the clock, rather than just the rising edge
  - Second, DDR uses higher clock rate on the bus to increase the transfer rate
  - Third, a buffering scheme is used

	DDR1	DDR2	DDR3	DDR4
Prefetch buffer (bits)	2	4	8	8
Voltage level (V)	2.5	1.8	1.5	1.2
Front side bus data rates (Mbps)	200—400	400—1066	800—2133	2133—4266

Table 5.4
DDR Characteristics

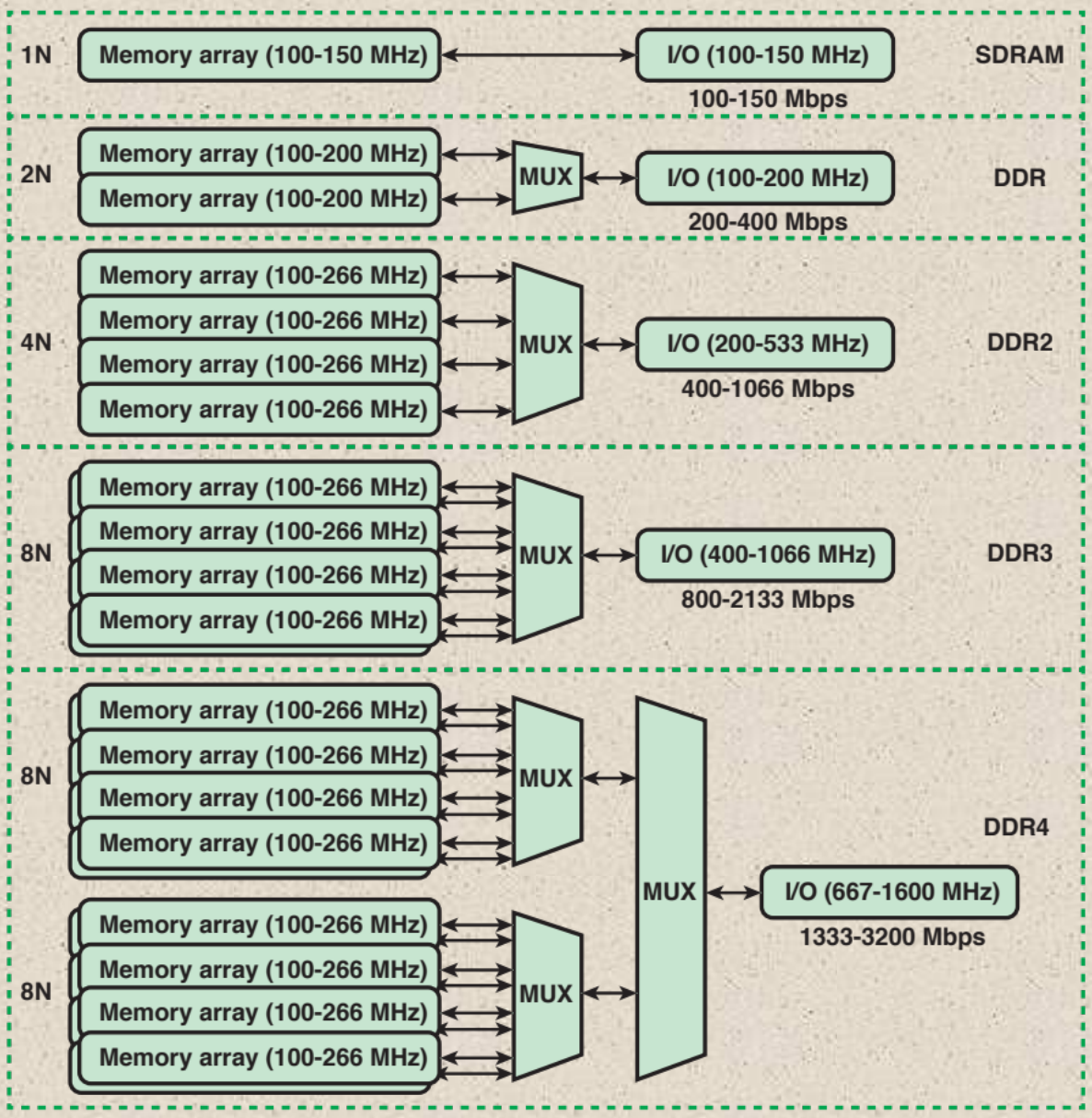
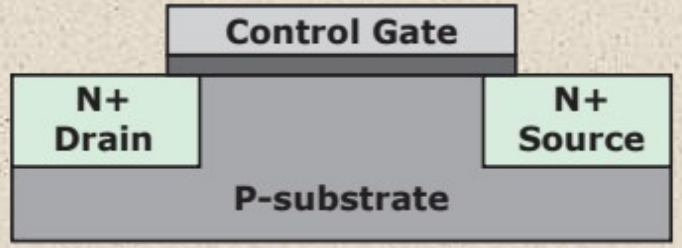


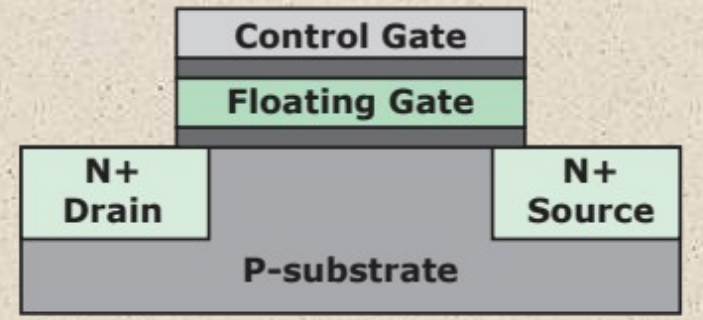
Figure 5.14 DDR Generations

Flash Memory

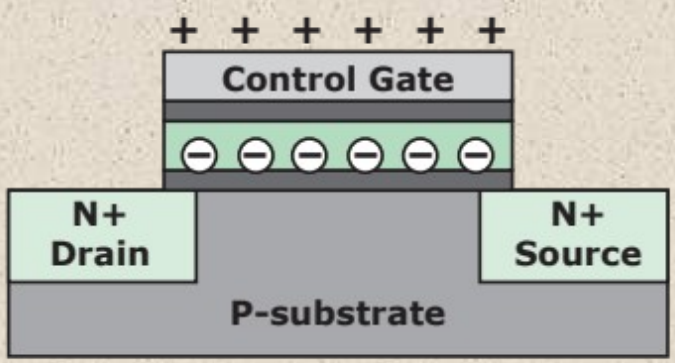
- Used both for internal memory and external memory applications
- First introduced in the mid-1980's
- Is intermediate between EPROM and EEPROM in both cost and functionality
- Uses an electrical erasing technology like EEPROM
- It is possible to erase just blocks of memory rather than an entire chip
- Gets its name because the microchip is organized so that a section of memory cells are erased in a single action
- Does not provide byte-level erasure
- Uses only one transistor per bit so it achieves the high density of EPROM



(a) Transistor structure

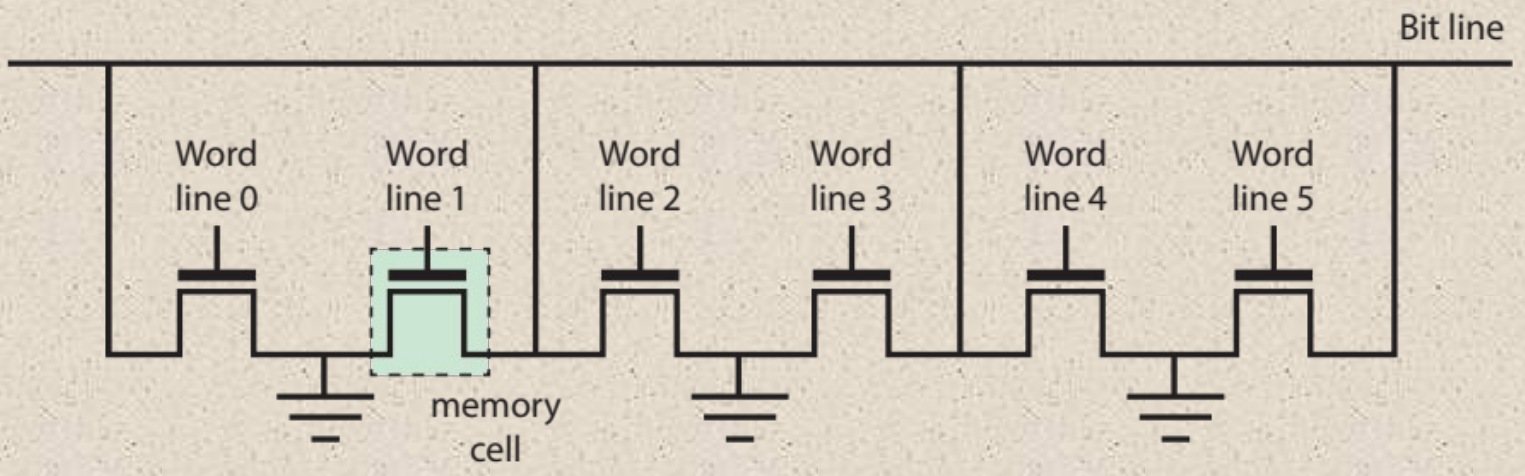


(b) Flash memory cell in one state

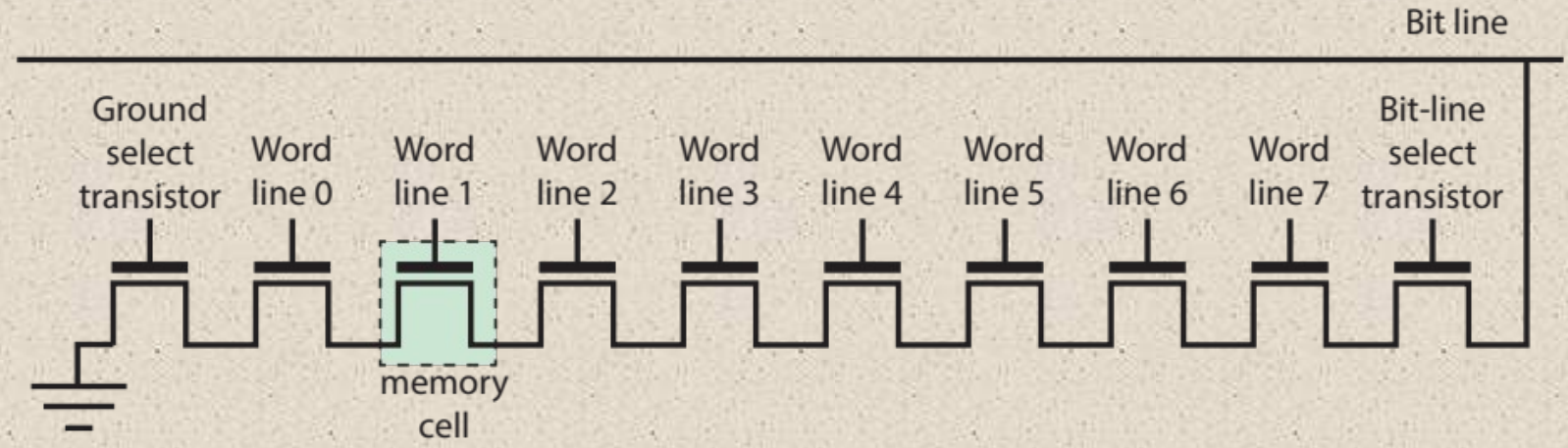


(c) Flash memory cell in zero state

Figure 5.15 Flash Memory Operation



(a) NOR flash structure



(b) NAND flash structure

Figure 5.16 Flash Memory Structures

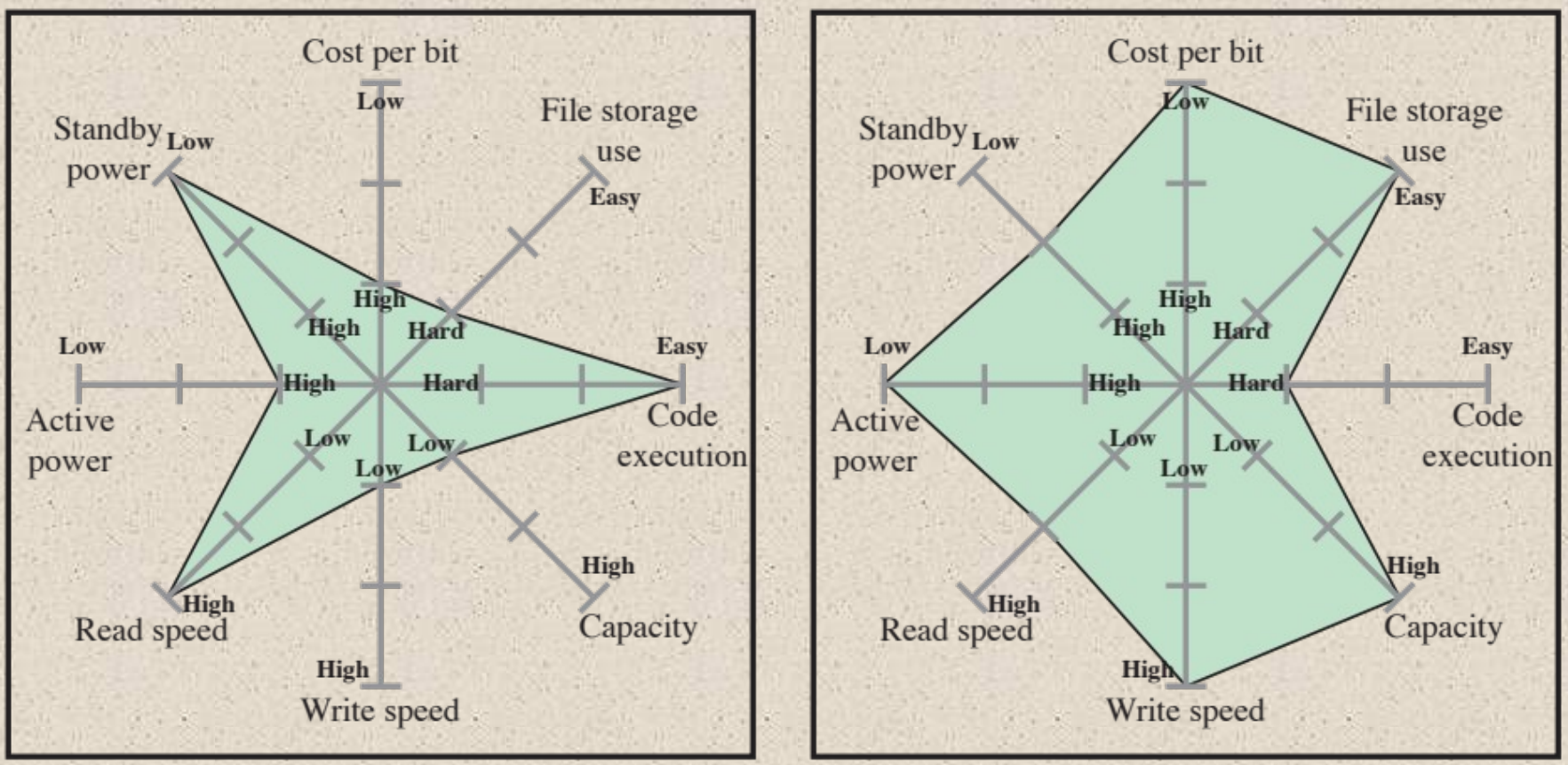


Figure 5.17 Kiviat Graphs for Flash Memory

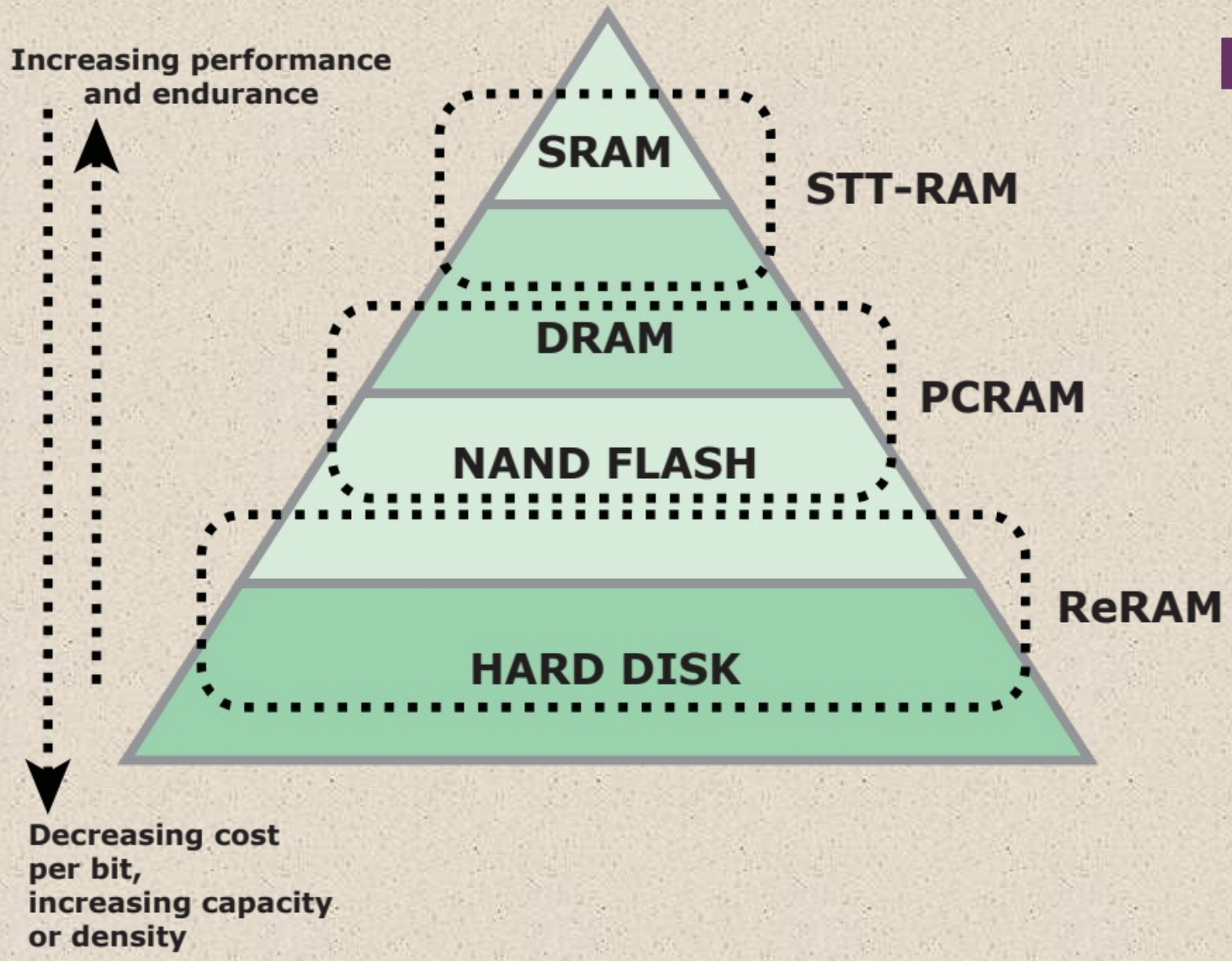
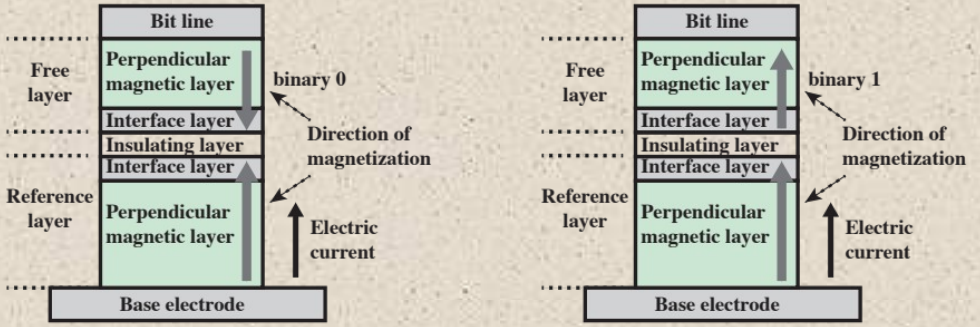
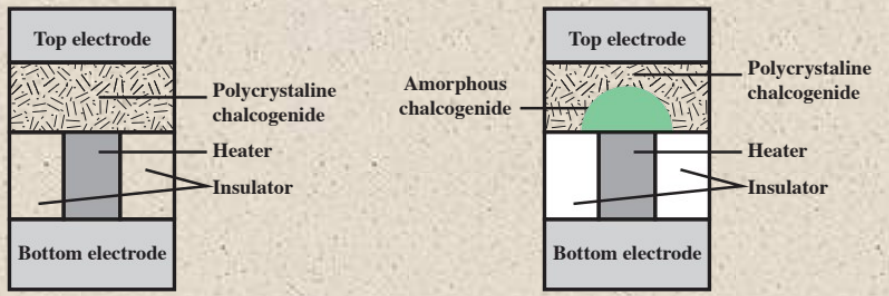


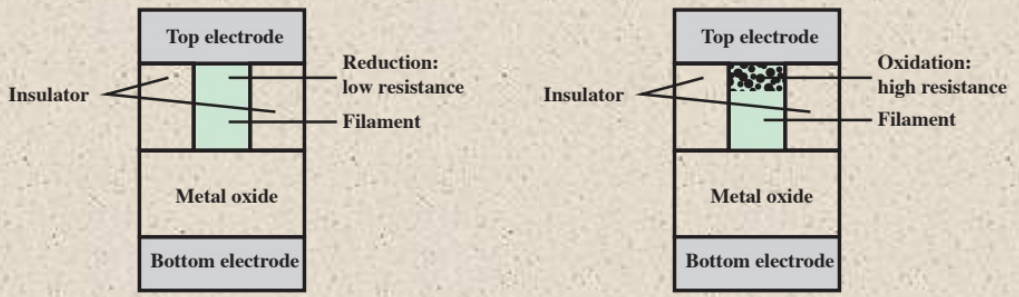
Figure 5.18 Nonvolatile RAM within the Memory Hierarchy



(a) STT-RAM



(b) PCRAM



(c) ReRAM

Figure 5.19 Nonvolatile RAM Technologies

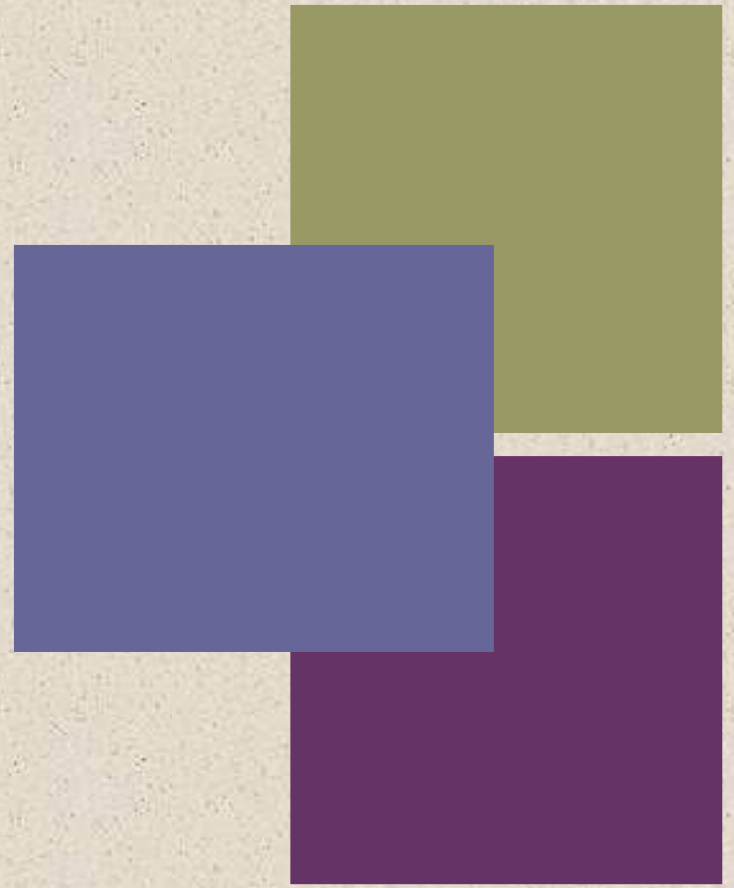
Summary

Chapter 5

Internal Memory

- Semiconductor main memory
  - Organization
  - DRAM and SRAM
  - Types of ROM
  - Chip logic
  - Chip packaging
  - Module organization
  - Interleaved memory
- Error correction
- DDR DRAM
  - Synchronous DRAM
  - DDR SDRAM
- Flash memory
  - Operation
  - NOR and NAND flash memory
- Newer nonvolatile solid-state memory technologies

William Stallings
Computer Organization and Architecture
10th Edition



Chapter 6
External Memory

Magnetic Disk

- A disk is a circular *platter* constructed of nonmagnetic material, called the *substrate,* coated with a magnetizable material
  - Traditionally the substrate has been an aluminium or aluminium alloy material
  - Recently glass substrates have been introduced
- Benefits of the glass substrate:
  - Improvement in the uniformity of the magnetic film surface to increase disk reliability
  - A significant reduction in overall surface defects to help reduce read-write errors
  - Ability to support lower fly heights
  - Better stiffness to reduce disk dynamics
  - Greater ability to withstand shock and damage

Magnetic Read and Write Mechanisms

Data are recorded on and later retrieved from the disk via a conducting coil named the *head*

- In many systems there are two heads, a read head and a write head
- During a read or write operation the head is stationary while the platter rotates beneath it

The write mechanism exploits the fact that electricity flowing through a coil produces a magnetic field

Electric pulses are sent to the write head and the resulting magnetic patterns are recorded on the surface below, with different patterns for positive and negative currents

The write head itself is made of easily magnetizable material and is in the shape of a rectangular doughnut with a gap along one side and a few turns of conducting wire along the opposite side

An electric current in the wire induces a magnetic field across the gap, which in turn magnetizes a small area of the recording medium

Reversing the direction of the current reverses the direction of the magnetization on the recording medium

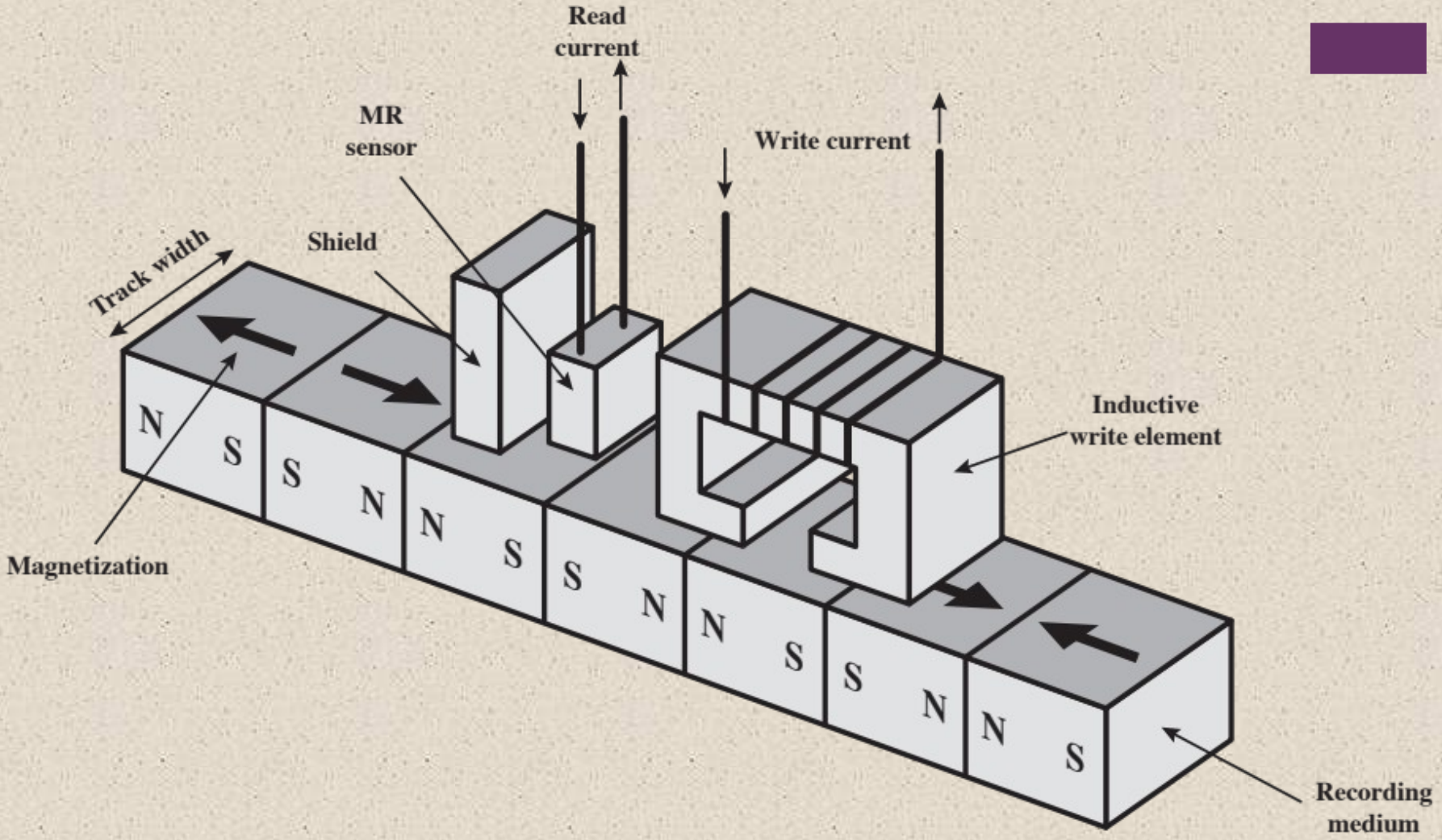


Figure 6.1 Inductive Write/Magnetoresistive Read Head

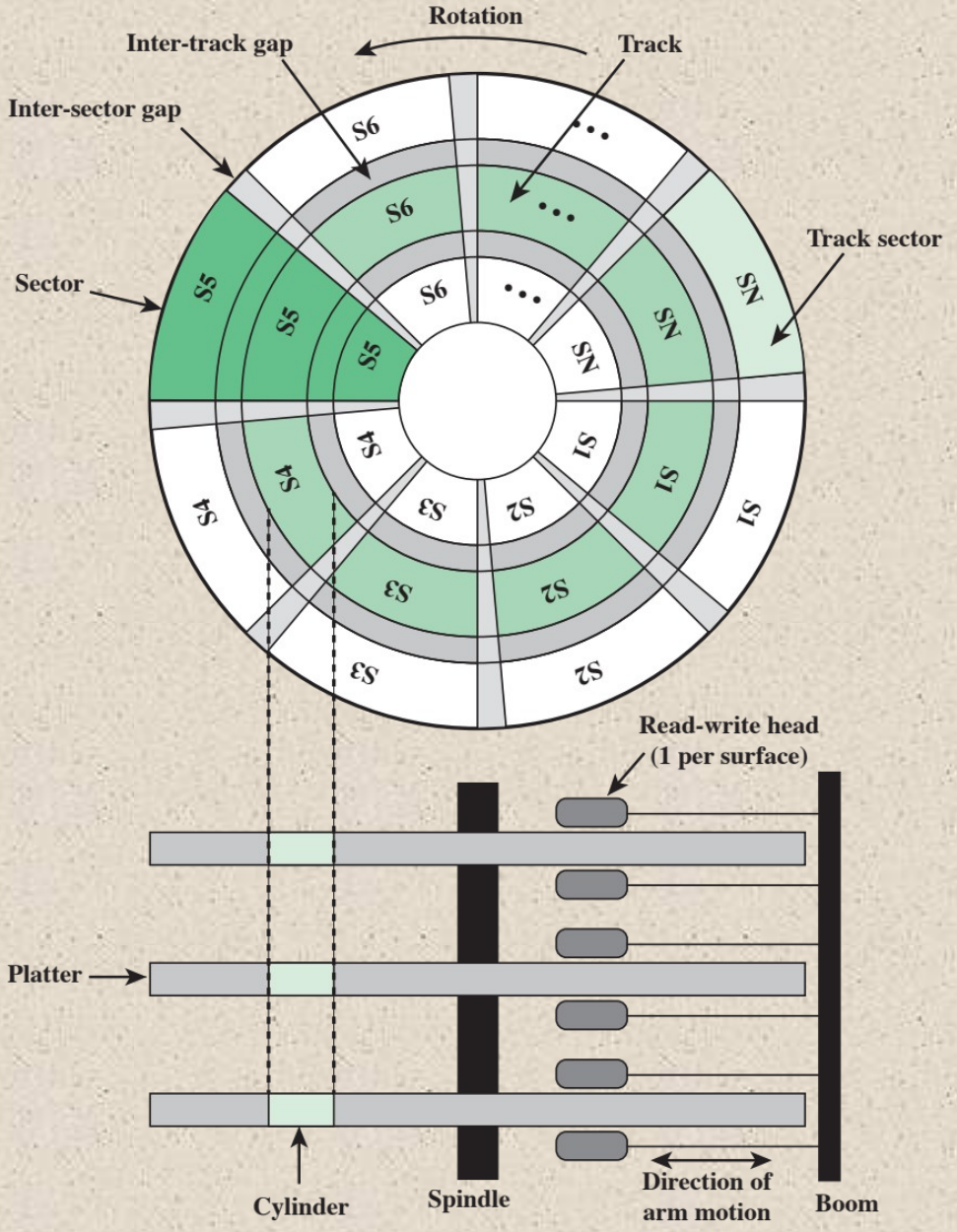


Figure 6.2 Disk Data Layout

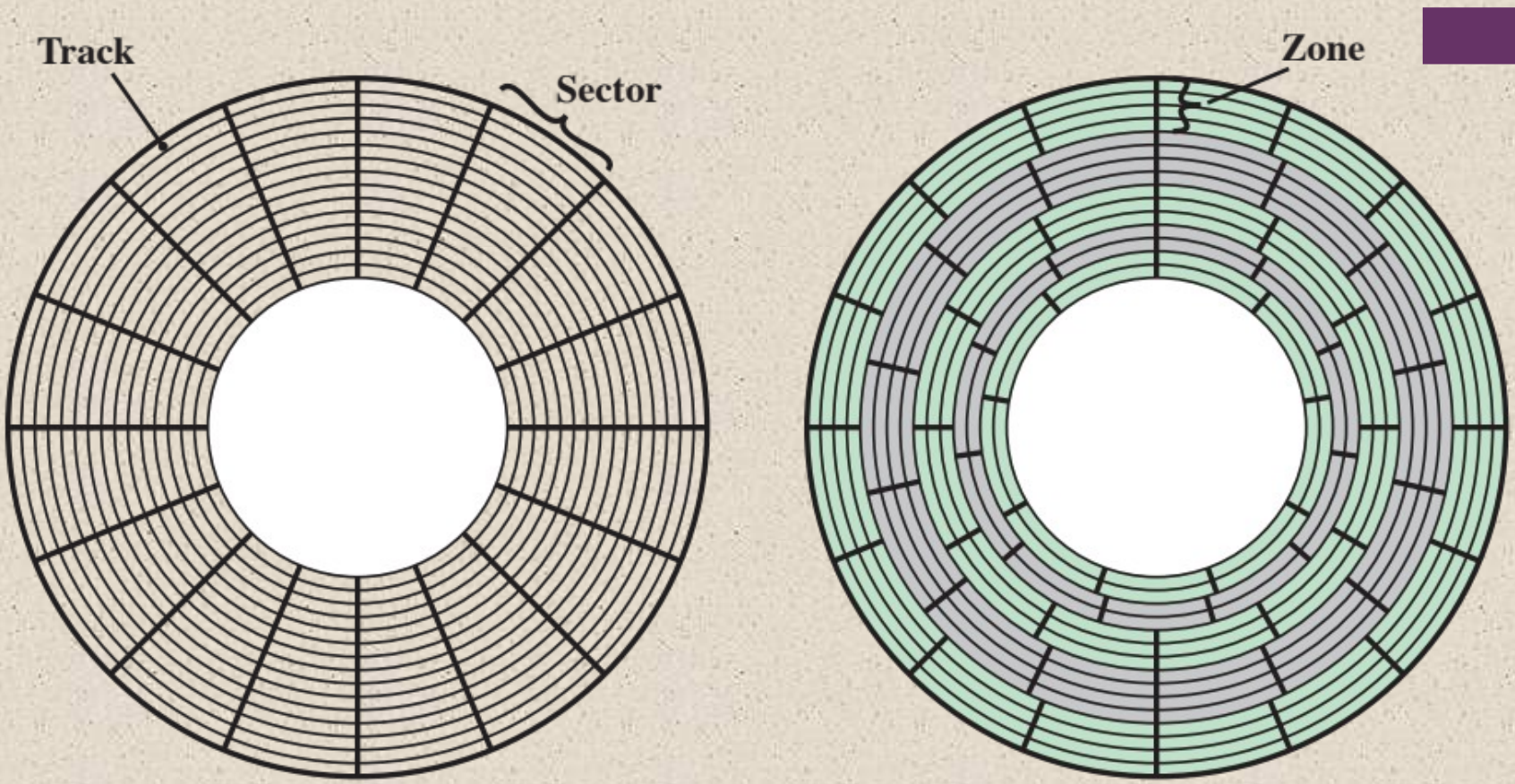


Figure 6.3 Comparison of Disk Layout Methods

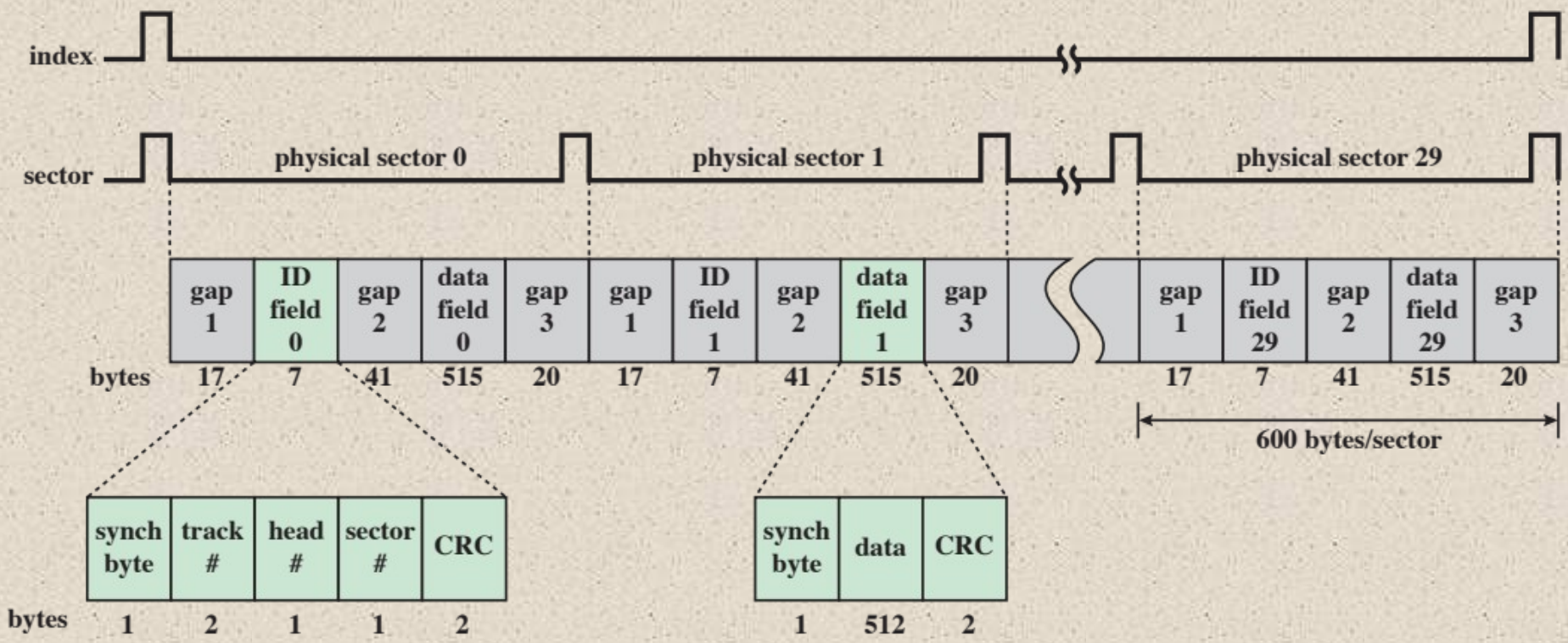


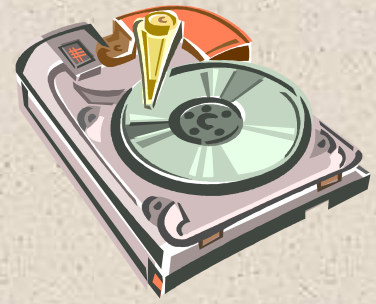
Figure 6.4 Winchester Disk Format (Seagate ST506)

Head Motion	**Platters**
Fixed head (one per track)	Single platter
Movable head (one per surface)	Multiple platter
Disk Portability	**Head Mechanism**
Nonremovable disk	Contact (floppy)
Removable disk	Fixed gap
	Aerodynamic gap (Winchester)
Sides	
Single sided	
Double sided	

Table 6.1
Physical Characteristics of Disk Systems

Characteristics

- Fixed-head disk
  - One read-write head per track
  - Heads are mounted on a fixed ridged arm that extends across all tracks
- Movable-head disk
  - One read-write head
  - Head is mounted on an arm
  - The arm can be extended or retracted
- Non-removable disk
  - Permanently mounted in the disk drive
  - The hard disk in a personal computer is a non-removable disk
- Removable disk
  - Can be removed and replaced with another disk
  - Advantages:
    - Unlimited amounts of data are available with a limited number of disk systems
    - A disk may be moved from one computer system to another
  - Floppy disks and ZIP cartridge disks are examples of removable disks
- Double sided disk
  - Magnetizable coating is applied to both sides of the platter



Disk Classification

The head mechanism provides a classification of disks into three types

- The head must generate or sense an electromagnetic field of sufficient magnitude to write and read properly
- The narrower the head, the closer it must be to the platter surface to function
  - A narrower head means narrower tracks and therefore greater data density
- The closer the head is to the disk the greater the risk of error from impurities or imperfections

Winchester Heads

- Used in sealed drive assemblies that are almost free of contaminants
- Designed to operate closer to the disk's surface than conventional rigid disk heads, thus allowing greater data density
- Is actually an aerodynamic foil that rests lightly on the platter's surface when the disk is motionless
  - The air pressure generated by a spinning disk is enough to make the foil rise above the surface

Table 6.2
Typical Hard Disk Drive Parameters

Characteristics	Seagate Enterprise	Seagate Barracuda XT	Seagate Cheetah NS	Seagate Laptop HDD
Application	Enterprise	Desktop	Network attached storage, application servers	Laptop
Capacity	6 TB	3 TB	600 GB	2 TB
Average seek time	4.16 ms	N/A	3.9 ms read 4.2 ms write	13 ms
Spindle speed	7200 rpm	7200 rpm	10, 075 rpm	5400 rpm
Average latency	4.16 ms	4.16 ms	2.98	5.6 ms
Maximum sustained transfer rate	216 MB/s	149 MB/s	97 MB/s	300 MB/s
Bytes per sector	512/4096	512	512	4096
Tracks per cylinder (number of platter surfaces)	8	10	8	4
Cache	128 MB	64 MB	16 MB	8 MB

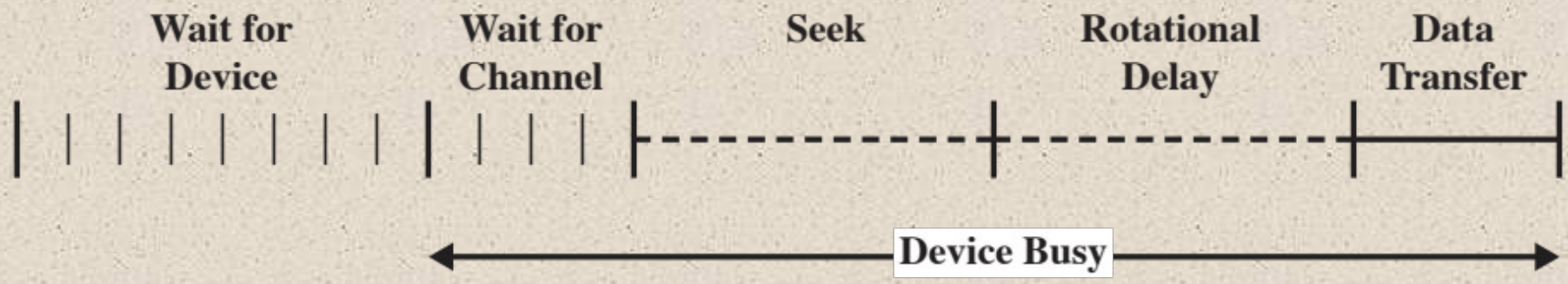


Figure 6.5 Timing of a Disk I/O Transfer

Disk Performance Parameters

- When the disk drive is operating the disk is rotating at constant speed
- To read or write the head must be positioned at the desired track and at the beginning of the desired sector on the track
  - Track selection involves moving the head in a movable-head system or electronically selecting one head on a fixed-head system
  - Once the track is selected, the disk controller waits until the appropriate sector rotates to line up with the head
- Seek time
  - On a movable–head system, the time it takes to position the head at the track
- Rotational delay *(rotational latency)*
  - The time it takes for the beginning of the sector to reach the head
- Access time
  - The sum of the seek time and the rotational delay
  - The time it takes to get into position to read or write
- Transfer time
  - Once the head is in position, the read or write operation is then pe as the sector moves under the head
  - This is the data transfer portion of the operation

RAID

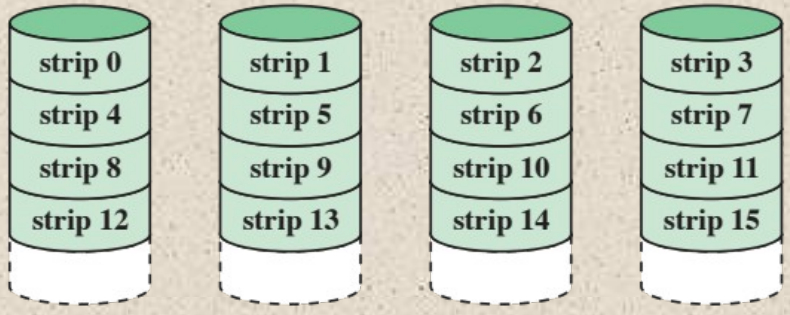
Redundant Array of Independent Disks

- Consists of 7 levels
- Levels do not imply a hierarchical relationship but designate different design architectures that share three common characteristics:
  1) Set of physical disk drives viewed by the operating system as a single logical drive
  2) Data are distributed across the physical drives of an array in a scheme known as striping
  3) Redundant disk capacity is used to store parity information, which guarantees data recoverability in case of a disk failure

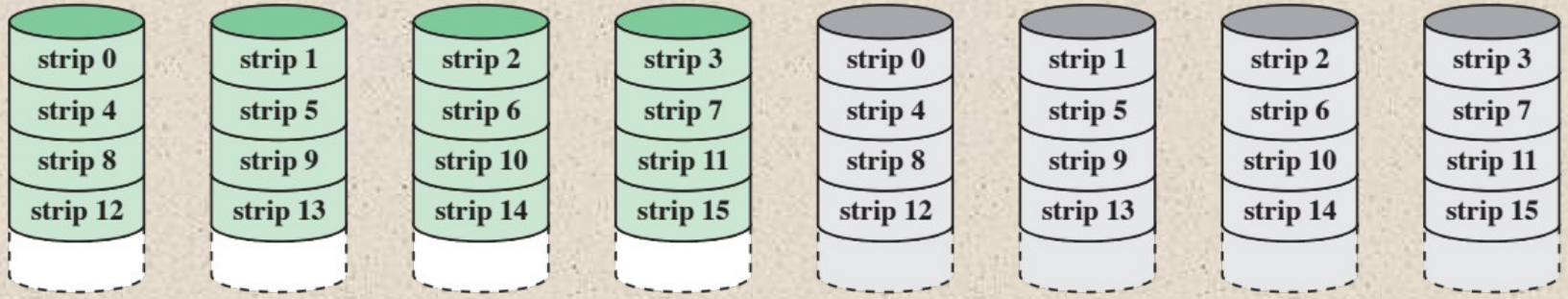
Table 6.3
RAID Levels

Category	Level	Description	Disks Required	Data Availability	Large I/O Data Transfer Capacity	Small I/O Request Rate
Striping	0	Nonredundant	N	Lower than single disk	Very high	Very high for both read and write
Mirroring	1	Mirrored	$2N$	Higher than RAID 2, 3, 4, or 5; lower than RAID 6	Higher than single disk for read; similar to single disk for write	Up to twice that of a single disk for read; similar to single disk for write
Parallel access	2	Redundant via Hamming code	$N + m$	Much higher than single disk; comparable to RAID 3, 4, or 5	Highest of all listed alternatives	Approximately twice that of a single disk
	3	Bit-interleaved parity	$N + 1$	Much higher than single disk; comparable to RAID 2, 4, or 5	Highest of all listed alternatives	Approximately twice that of a single disk
Independent access	4	Block-interleaved parity	$N + 1$	Much higher than single disk; comparable to RAID 2, 3, or 5	Similar to RAID 0 for read; significantly lower than single disk for write	Similar to RAID 0 for read; significantly lower than single disk for write
	5	Block-interleaved distributed parity	$N + 1$	Much higher than single disk; comparable to RAID 2, 3, or 4	Similar to RAID 0 for read; lower than single disk for write	Similar to RAID 0 for read; generally lower than single disk for write
	6	Block-interleaved dual distributed parity	$N + 2$	Highest of all listed alternatives	Similar to RAID 0 for read; lower than RAID 5 for write	Similar to RAID 0 for read; significantly lower than RAID 5 for write

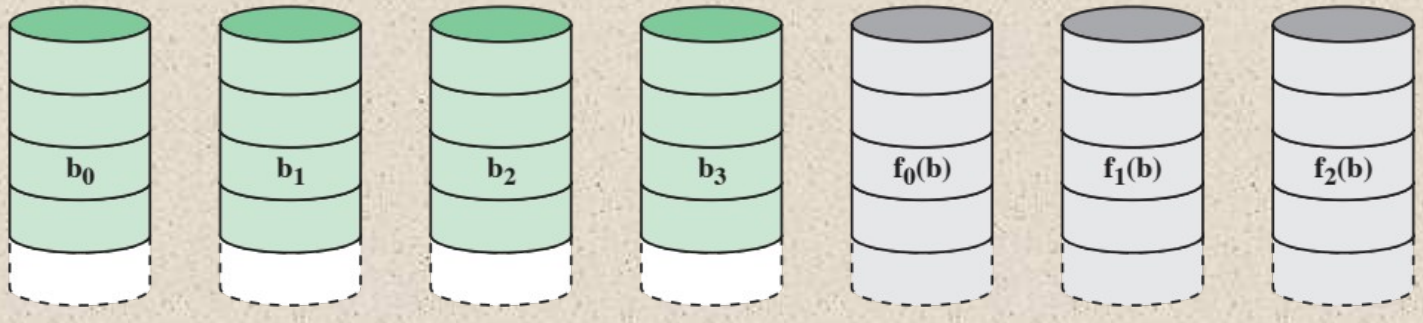
N = number of data disks; m proportional to log N



(a) RAID 0 (non-redundant)

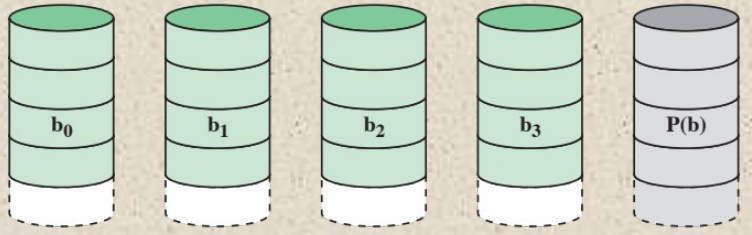


(b) RAID 1 (mirrored)

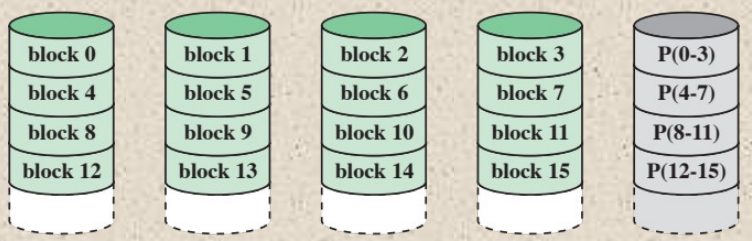


(c) RAID 2 (redundancy through Hamming code)

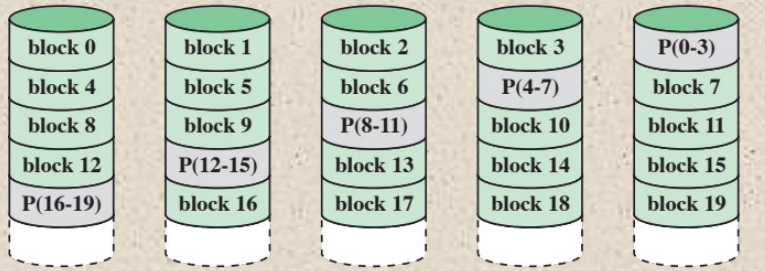
Figure 6.6 RAID Levels (page 1 of 2)



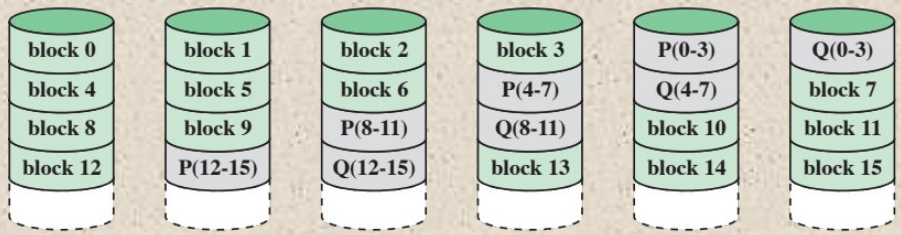
(d) RAID 3 (bit-interleaved parity)



(e) RAID 4 (block-level parity)



(f) RAID 5 (block-level distributed parity)



(g) RAID 6 (dual redundancy)

Figure 6.6 RAID Levels (page 2 of 2)

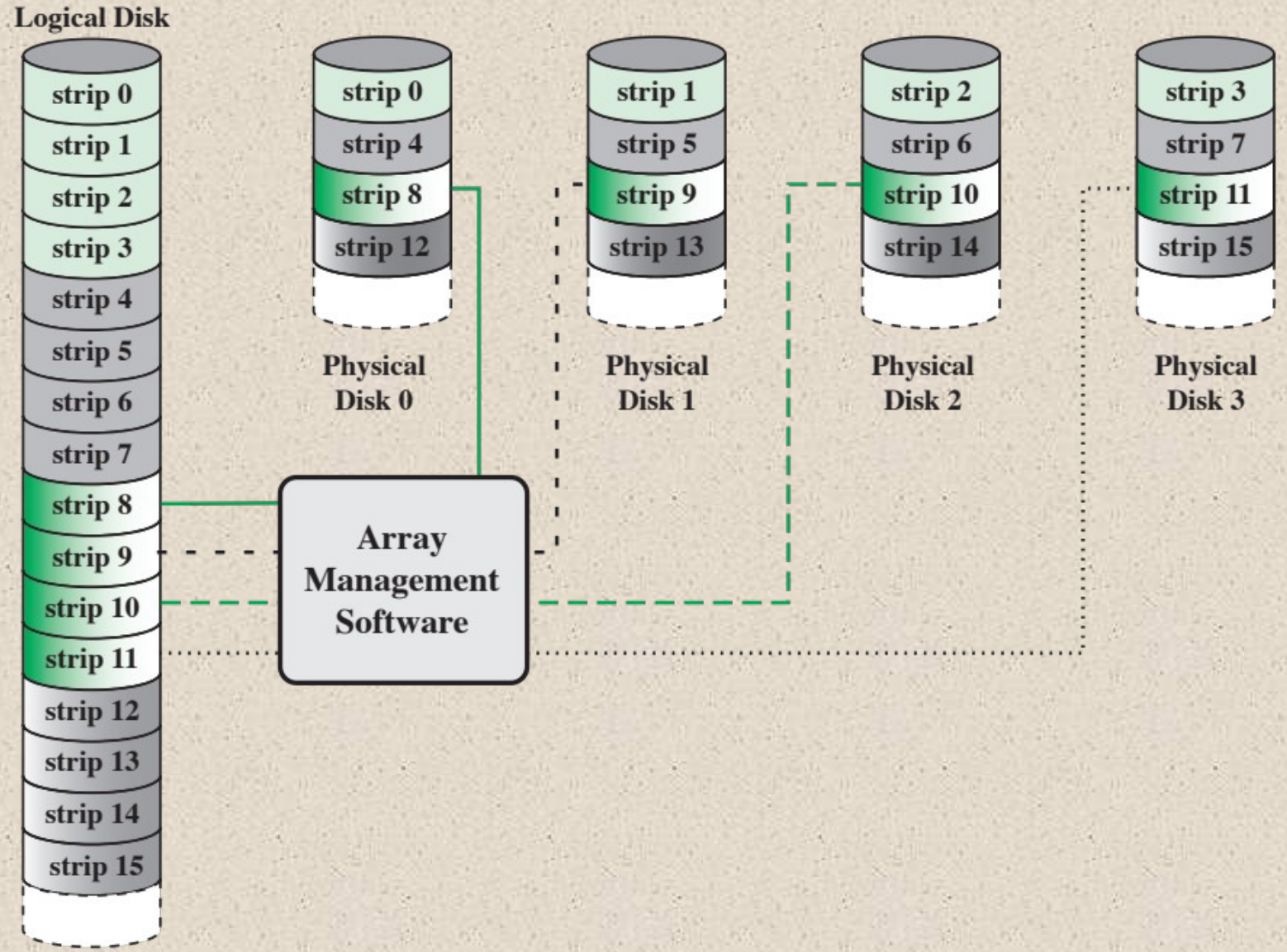


Figure 6.7 Data Mapping for a RAID Level 0 Array

RAID Level 0

- Addresses the issues of request patterns of the host system and layout of the data
- Impact of redundancy does not interfere with analysis

Raid 0

RAID 0 for High Data Transfer Capacity

- For applications to experience a high transfer rate two requirements must be met:
  1. A high transfer capacity must exist along the entire path between host memory and the individual disk drives
  2. The application must make I/O requests that drive the disk array efficiently

RAID 0 for High I/O Request Rate

- For an individual I/O request for a small amount of data the I/O time is dominated by the seek time and rotational latency
- A disk array can provide high I/O execution rates by balancing the I/O load across multiple disks
- If the strip size is relatively large multiple waiting I/O requests can be handled in parallel, reducing the queuing time for each request

RAID Level 1

Characteristics

- Differs from RAID levels 2 through 6 in the way in which redundancy is achieved
- Redundancy is achieved by the simple expedient of duplicating all the data
- Data striping is used but each logical strip is mapped to two separate physical disks so that every disk in the array has a mirror disk that contains the same data
- RAID 1 can also be implemented without data striping, although this is less common

Positive Aspects

- A read request can be serviced by either of the two disks that contains the requested data
- There is no “write penalty”
- Recovery from a failure is simple, when a drive fails the data can be accessed from the second drive
- Provides real-time copy of all data
- Can achieve high I/O request rates if the bulk of the requests are reads
- Principal disadvantage is the cost

RAID Level 2

Characteristics

- Makes use of a parallel access technique
- In a parallel access array all member disks participate in the execution of every I/O request
- Spindles of the individual drives are synchronized so that each disk head is in the same position on each disk at any given time
- Data striping is used
  - Strips are very small, often as small as a single byte or word

Performance

- An error-correcting code is calculated across corresponding bits on each data disk and the bits of the code are stored in the corresponding bit positions on multiple parity disks
- Typically a Hamming code is used, which is able to correct single-bit errors and detect double-bit errors
- The number of redundant disks is proportional to the log of the number of data disks
- Would only be an effective choice in an environment in which many disk errors occur

RAID Level 3

Redundancy

- Requires only a single redundant disk, no matter how large the disk array
- Employs parallel access, with data distributed in small strips
- Instead of an error correcting code, a simple parity bit is computed for the set of individual bits in the same position on all of the data disks
- Can achieve very high data transfer rates

Performance

- In the event of a drive failure, the parity drive is accessed and data is reconstructed from the remaining devices
- Once the failed drive is replaced, the missing data can be restored on the new drive and operation resumed
- In the event of a disk failure, all of the data are still available in what is referred to as *reduced mode*
- Return to full operation requires that the failed disk be replaced and the entire contents of the failed disk be regenerated on the new disk
- In a transaction-oriented environment performance suffers

RAID Level 4

Characteristics

- Makes use of an independent access technique
  - In an independent access array, each member disk operates independently so that separate I/O requests can be satisfied in parallel
- Data striping is used
  - Strips are relatively large
- To calculate the new parity the array management software must read the old user strip and the old parity strip

Performance

- Involves a write penalty when an I/O write request of small size is performed
- Each time a write occurs the array management software must update not only the user data but also the corresponding parity bits
- Thus each strip write involves two reads and two writes

RAID Level 5

Characteristics

- Organized in a similar fashion to RAID 4
- Difference is distribution of the parity strips across all disks
- A typical allocation is a round-robin scheme
- The distribution of parity strips across all drives avoids the potential I/O bottleneck found in RAID 4

RAID Level 6

Characteristics

- Two different parity calculations are carried out and stored in separate blocks on different disks
- Advantage is that it provides extremely high data availability
- Three disks would have to fail within the mean time to repair (MTTR) interval to cause data to be lost
- Incurs a substantial write penalty because each write affects two parity blocks

Level	Advantages	Disadvantages	Applications
0	I/O performance is greatly improved by spreading the I/O load across many channels and drives No parity calculation overhead is involved Very simple design Easy to implement	The failure of just one drive will result in all data in an array being lost	Video production and Editing Image editing Pre-press applications Any application requiring high bandwidth
1	100% redundancy of data means no rebuild is necessary in case of a disk failure, just a copy to the replacement disk Under certain circumstances, RAID 1 can sustain multiple simultaneous drive failures Simplest RAID storage subsystem design	Highest disk overhead of all RAID types (100%) - inefficient	Accounting Payroll Financial Any application requiring very high availability
2	Extremely high data transfer rates possible The higher the data transfer rate required, the better the ratio of data disks to ECC disks Relatively simple controller design compared to RAID levels 3,4 & 5	Very high ratio of ECC disks to data disks with smaller word sizes - inefficient Entry level cost very high - requires very high transfer rate requirement to justify	No commercial implementations exist / not commercially viable

Table 6.4

RAID Comparison (page 1 of 2)

3	Very high read data transfer rate Very high write data transfer rate Disk failure has an insignificant impact on throughput Low ratio of ECC (parity) disks to data disks means high efficiency	Transaction rate equal to that of a single disk drive at best (if spindles are synchronized) Controller design is fairly complex	Video production and live streaming Image editing Video editing Prepress applications Any application requiring high throughput
4	Very high Read data transaction rate Low ratio of ECC (parity) disks to data disks means high efficiency	Quite complex controller design Worst write transaction rate and Write aggregate transfer rate Difficult and inefficient data rebuild in the event of disk failure	No commercial implementations exist / not commercially viable
5	Highest Read data transaction rate Low ratio of ECC (parity) disks to data disks means high efficiency Good aggregate transfer rate	Most complex controller design Difficult to rebuild in the event of a disk failure (as compared to RAID level 1)	File and application servers Database servers Web, e-mail, and news servers Intranet servers Most versatile RAID level
6	Provides for an extremely high data fault tolerance and can sustain multiple simultaneous drive failures	More complex controller design Controller overhead to compute parity addresses is extremely high	Perfect solution for mission critical applications

Table 6.4

RAID Comparison (page 2 of 2)

SSD Compared to HDD

- SSDs have the following advantages over HDDs:
- High-performance input/output operations per second (IOPS)
- Durability
- Longer lifespan
- Lower power consumption
- Quieter and cooler running capabilities
- Lower access times and latency rates

	NAND Flash Drives	Seagate Laptop Internal HDD
File copy/write speed	200—550 Mbps	50—120 Mbps
Power draw/battery life	Less power draw, averages 2–3 watts, resulting in 30+ minute battery boost	More power draw, averages 6–7 watts and therefore uses more battery
Storage capacity	Typically not larger than 512 GB for notebook size drives; 1 TB max for desktops	Typically around 500 GB and 2 TB maximum for notebook size drives; 4 TB max for desktops
Cost	Approx. $0.50 per GB for a 1-TB drive	Approx $0.15 per GB for a 4-TB drive

Table 6.5
Comparison of Solid State Drives and Disk Drives

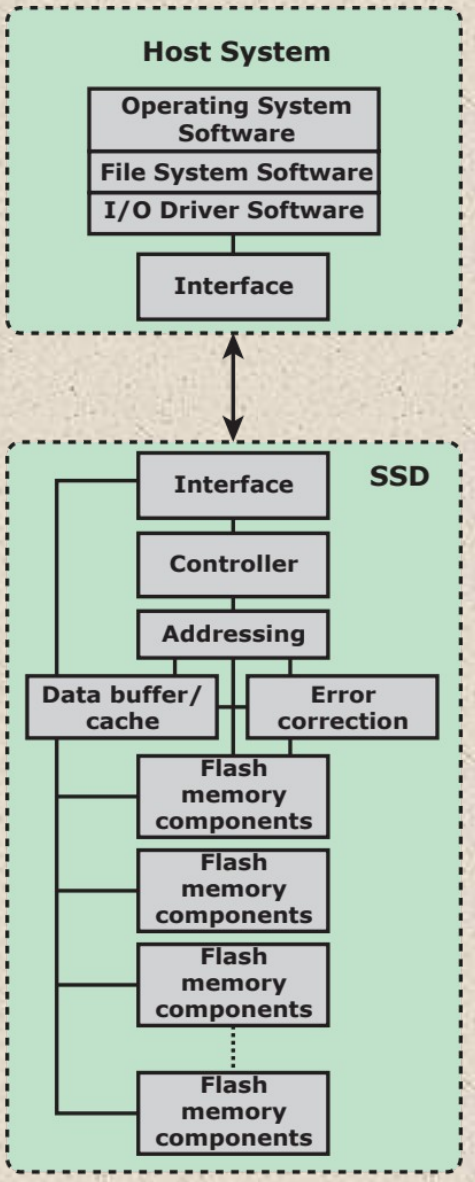


Figure 6.8 Solid State Drive Architecture

Practical Issues

There are two practical issues peculiar to SSDs that are not faced by HDDs:

- SDD performance has a tendency to slow down as the device is used
  - The entire block must be read from the flash memory and placed in a RAM buffer
  - Before the block can be written back to flash memory, the entire block of flash memory must be erased
  - The entire block from the buffer is now written back to the flash memory
- Flash memory becomes unusable after a certain number of writes
  - Techniques for prolonging life:
    - Front-ending the flash with a cache to delay and group write operations
    - Using wear-leveling algorithms that evenly distribute writes across block of cells
    - Bad-block management techniques
  - Most flash devices estimate their own remaining lifetimes so systems can anticipate failure and take preemptive action

CD

Compact Disk. A nonerasable disk that stores digitized audio information. The standard system uses 12-cm disks and can record more than 60 minutes of uninterrupted playing time.

CD-ROM

Compact Disk Read-Only Memory. A nonerasable disk used for storing computer data. The standard system uses 12-cm disks and can hold more than 650 Mbytes.

CD-R

CD Recordable. Similar to a CD-ROM. The user can write to the disk only once.

CD-RW

CD Rewritable. Similar to a CD-ROM. The user can erase and rewrite to the disk multiple times.

DVD

Digital Versatile Disk. A technology for producing digitized, compressed representation of video information, as well as large volumes of other digital data. Both 8 and 12 cm diameters are used, with a double-sided capacity of up to 17 Gbytes. The basic DVD is read-only (DVD-ROM).

DVD-R

DVD Recordable. Similar to a DVD-ROM. The user can write to the disk only once. Only one-sided disks can be used.

DVD-RW

DVD Rewritable. Similar to a DVD-ROM. The user can erase and rewrite to the disk multiple times. Only one-sided disks can be used.

Blu-Ray DVD

High definition video disk. Provides considerably greater data storage density than DVD, using a 405-nm (blue-violet) laser. A single layer on a single side can store 25 Gbytes.

Table 6. 6

Optical Disk Products

Compact Disk Read-Only Memory (CD-ROM)

- Audio CD and the CD-ROM share a similar technology
  - The main difference is that CD-ROM players are more rugged and have error correction devices to ensure that data are properly transferred
- Production:
  - The disk is formed from a resin such as polycarbonate
  - Digitally recorded information is imprinted as a series of microscopic pits on the surface of the polycarbonate
    - This is done with a finely focused, high intensity laser to create a master disk
  - The master is used, in turn, to make a die to stamp out copies onto polycarbonate
  - The pitted surface is then coated with a highly reflective surface, usually aluminum or gold
  - This shiny surface is protected against dust and scratches by a top coat of clear acrylic
  - Finally a label can be silkscreened onto the acrylic

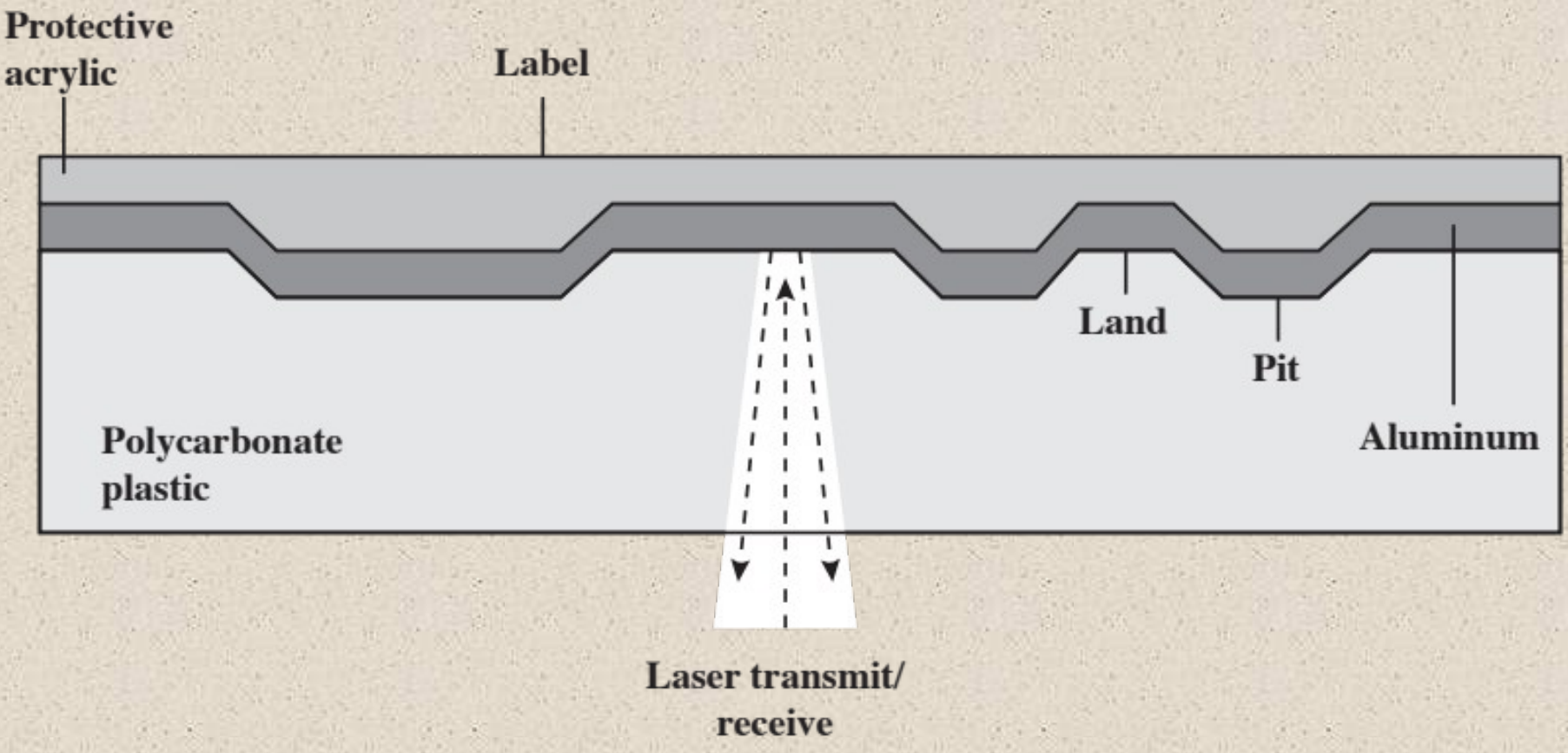


Figure 6.9 CD Operation

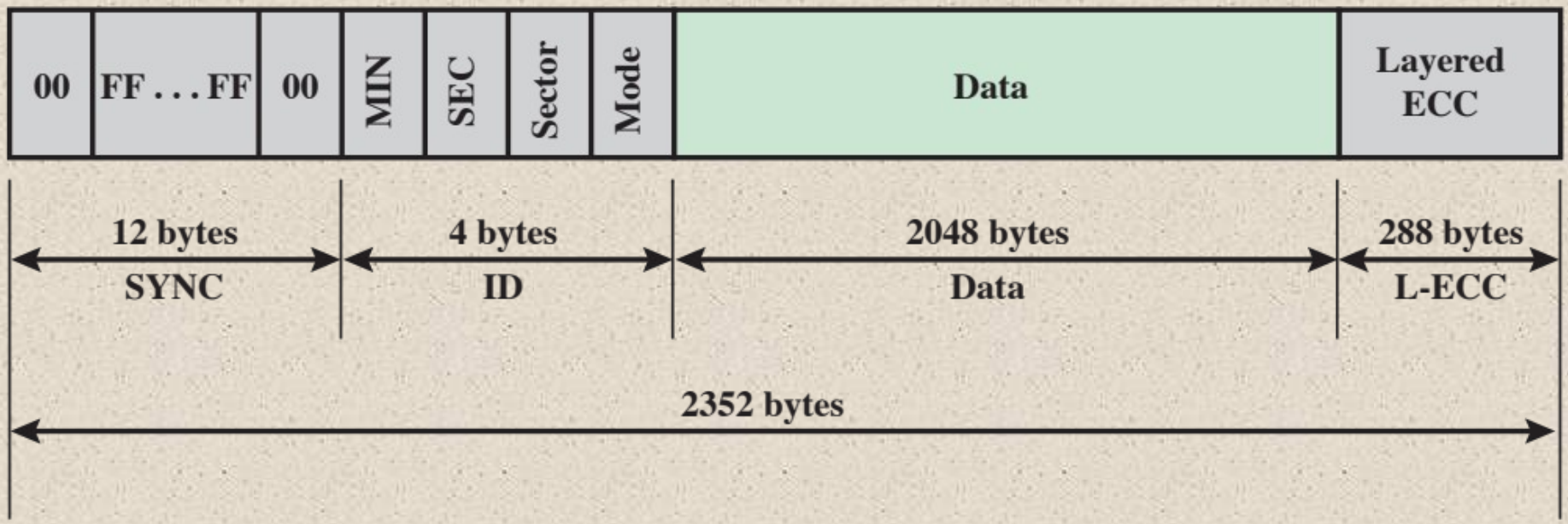
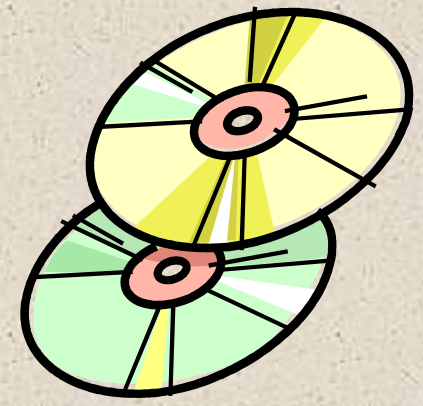


Figure 6.10 CD-ROM Block Format

CD-ROM

- CD-ROM is appropriate for the distribution of large amounts of data to a large number of users
- Because the expense of the initial writing process it is not appropriate for individualized applications
- The CD-ROM has two advantages:
  - The optical disk together with the information stored on it can be mass replicated inexpensively
  - The optical disk is removable, allowing the disk itself to be used for archival storage
- The CD-ROM disadvantages:
  - It is read-only and cannot be updated
  - It has an access time much longer than that of a magnetic disk drive

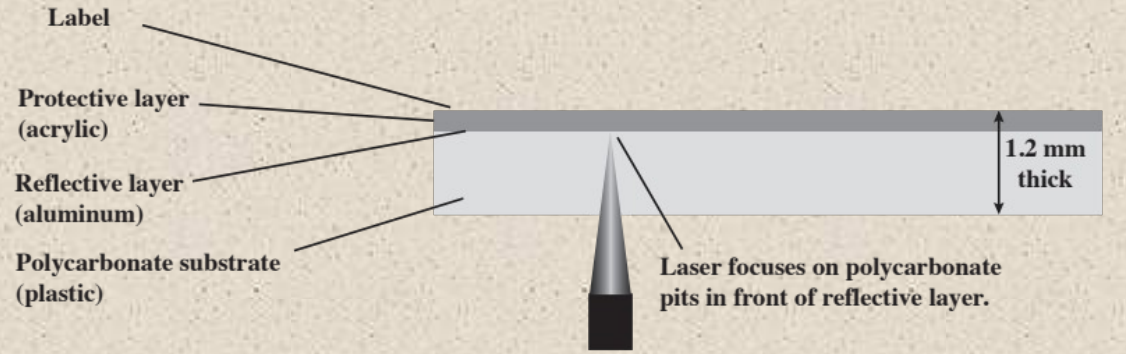


CD Recordable (CD-R)

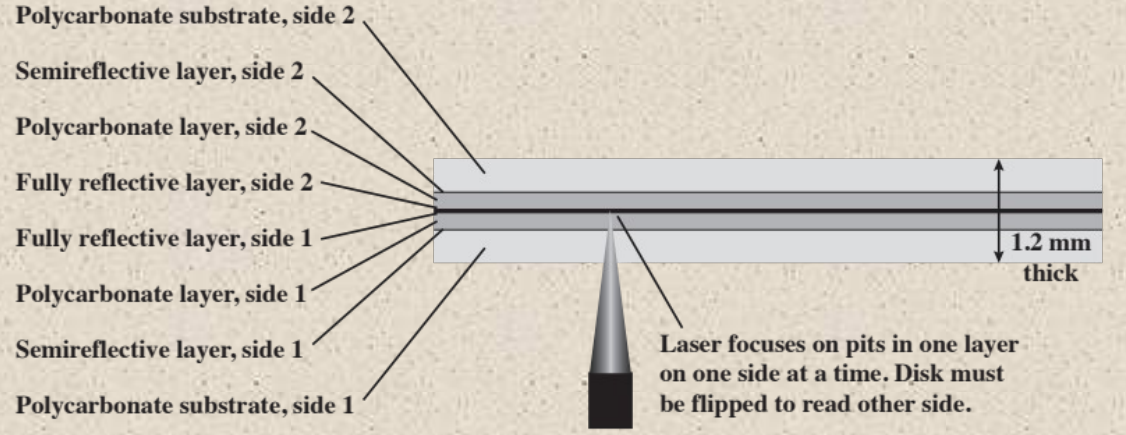
- Write-once read-many
- Accommodates applications in which only one or a small number of copies of a set of data is needed
- Disk is prepared in such a way that it can be subsequently written once with a laser beam of modest-intensity
- Medium includes a dye layer which is used to change reflectivity and is activated by a high-intensity laser
- Provides a permanent record of large volumes of user data

CD Rewritable (CD-RW)

- Can be repeatedly written and overwritten
- Phase change disk uses a material that has two significantly different reflectivities in two different phase states
- Amorphous state
  - Molecules exhibit a random orientation that reflects light poorly
- Crystalline state
  - Has a smooth surface that reflects light well
- A beam of laser light can change the material from one phase to the other
- Disadvantage is that the material eventually and permanently loses its desirable properties
- Advantage is that it can be rewritten



(a) CD-ROM - Capacity 682 MB



(b) DVD-ROM, double-sided, dual-layer - Capacity 17 GB

Figure 6.11 CD-ROM and DVD-ROM

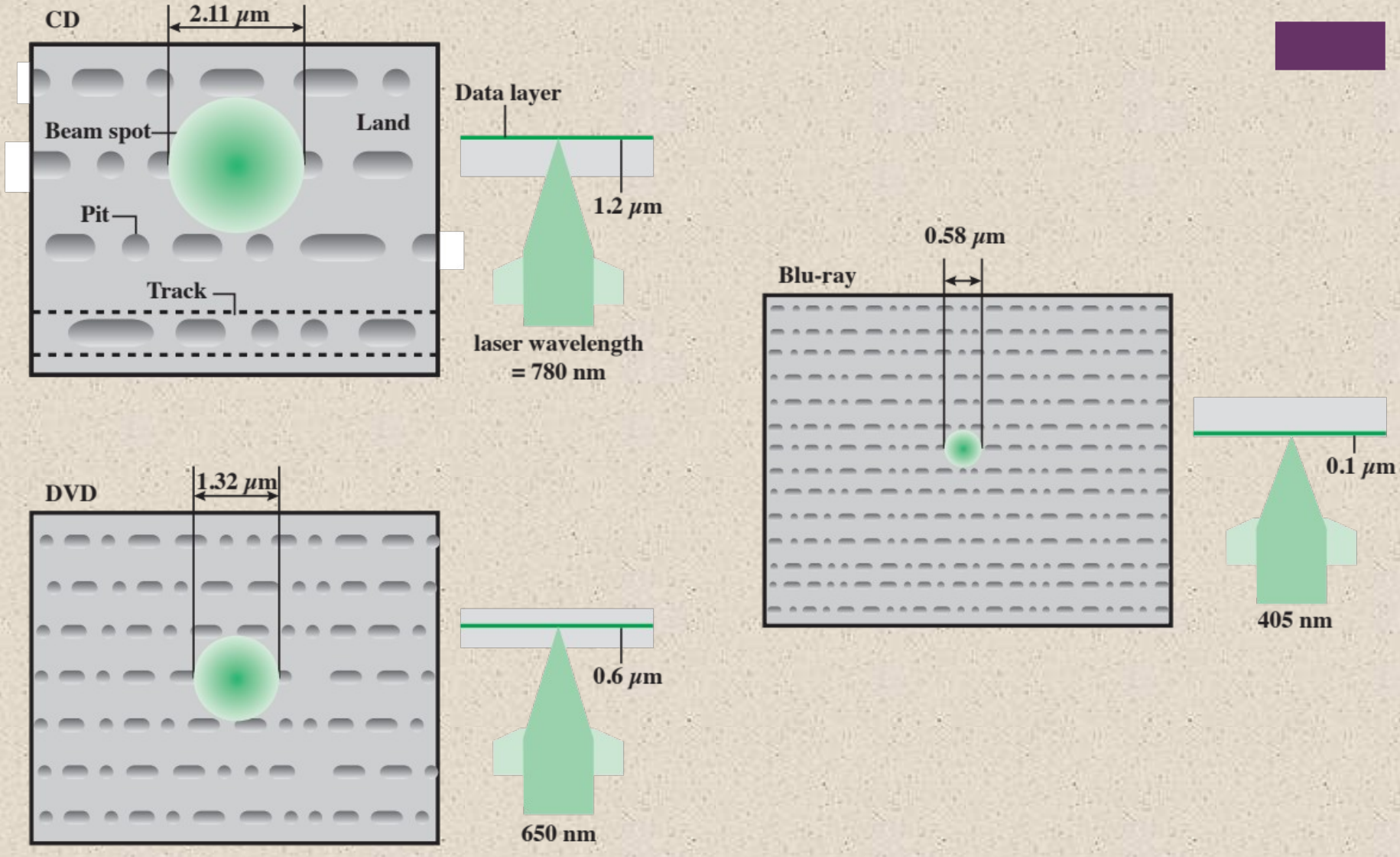
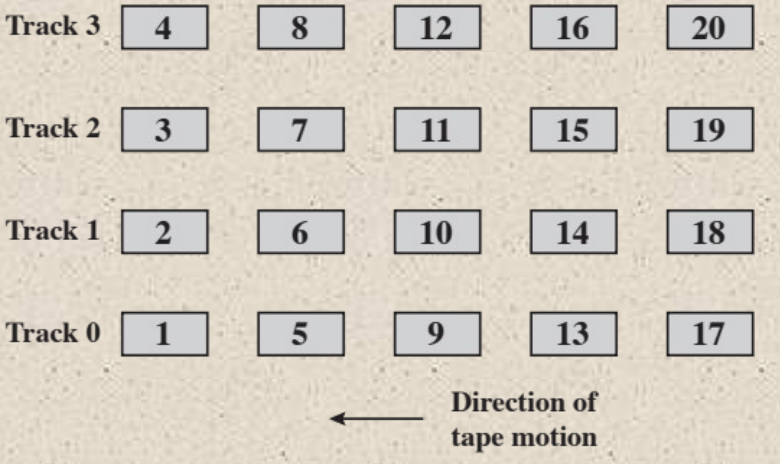


Figure 6.12 Optical Memory Characteristics

Magnetic Tape

- Tape systems use the same reading and recording techniques as disk systems
- Medium is flexible polyester tape coated with magnetizable material
- Coating may consist of particles of pure metal in special binders or vapor-plated metal films
- Data on the tape are structured as a number of parallel tracks running lengthwise
- Serial recording
  - Data are laid out as a sequence of bits along each track
- Data are read and written in contiguous blocks called *physical records*
- Blocks on the tape are separated by gaps referred to as *inter-record gaps*

(a) Serpentine reading and writing



(b) Block layout for system that reads/writes four tracks simultaneously

Figure 6.13 Typical Magnetic Tape Features

Table 6.7
LTO Tape Drives

	LTO-1	LTO-2	LTO-3	LTO-4	LTO-5	LTO-6	LTO-7	LTO-8
Release date	2000	2003	2005	2007	2010	TBA	TBA	TBA
Compressed capacity	200 GB	400 GB	800 GB	1600 GB	3.2 TB	8 TB	16 TB	32 TB
Compressed transfer rate (MB/s)	40 MB/s	80 MB/s	160 MB/s	240 MB/s	280 MB/s	525 MB/s	788 MB/s	1.18 GB/s
Linear density (bits/mm)	4880	7398	9638	13250	15142			
Tape tracks	384	512	704	896	1280			
Tape length	609 m	609 m	680 m	820 m	846 m			
Tape width (cm)	1.27	1.27	1.27	1.27	1.27			
Write elements	8	8	16	16	16			
WORM?	No	No	Yes	Yes	Yes	Yes	Yes	Yes
Encryption Capable?	No	No	No	Yes	Yes	Yes	Yes	Yes
Partitioning?	No	No	No	No	Yes	Yes	Yes	Yes

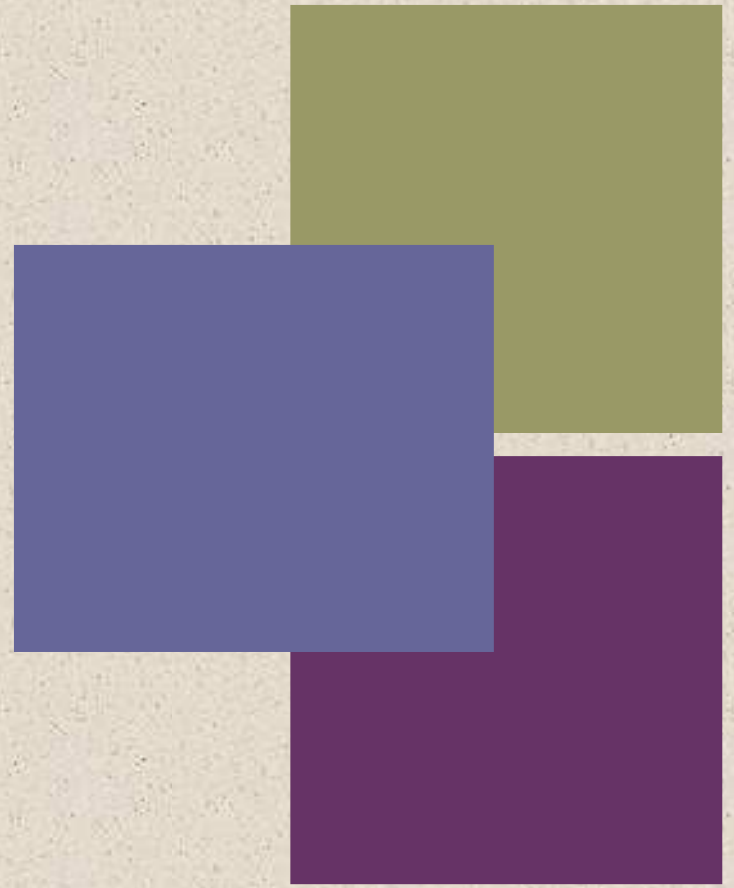
Summary

Chapter 6

External Memory

- Magnetic disk
  - Magnetic read and write mechanisms
  - Data organization and formatting
  - Physical characteristics
  - Disk performance parameters
- Solid state drives
  - SSD compared to HDD
  - SSD organization
  - Practical issues
- Magnetic tape
- RAID
  - RAID level 0
  - RAID level 1
  - RAID level 2
  - RAID level 3
  - RAID level 4
  - RAID level 5
  - RAID level 6
- Optical memory
  - Compact disk
  - Digital versatile disk
  - High-definition optical disks

William Stallings
Computer Organization and Architecture
10th Edition



Chapter 7

Input/Output

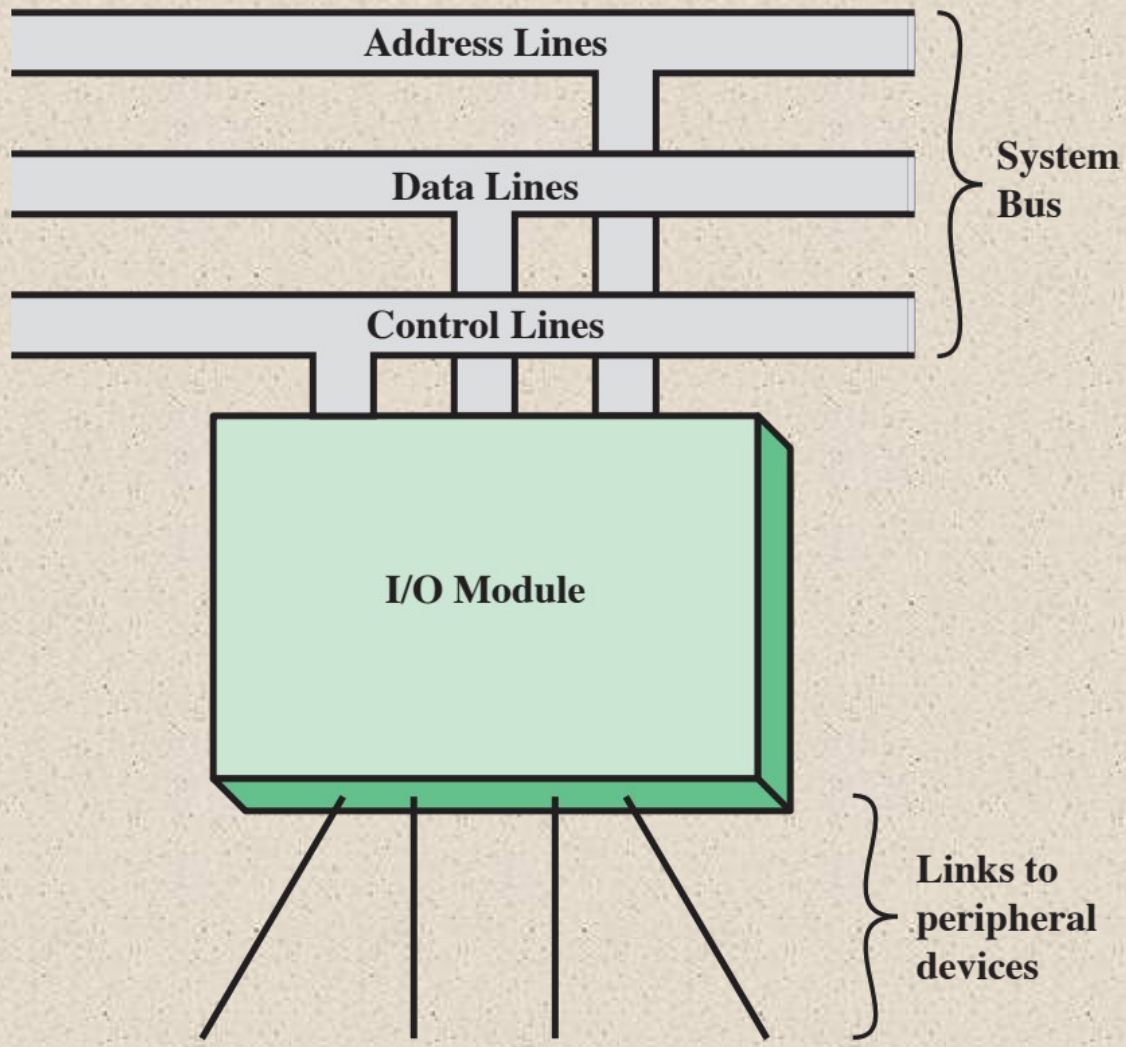


Figure 7.1 Generic Model of an I/O Module

External Devices

- Provide a means of exchanging data between the external environment and the computer
- Attach to the computer by a link to an I/O module
  - The link is used to exchange control, status, and data between the I/O module and the external device
- *Peripheral device*
  - An external device connected to an I/O module

Three categories:

- Human readable
  - Suitable for communicating with the computer user
  - Video display terminals (VDTs), printers
- Machine readable
  - Suitable for communicating with equipment
  - Magnetic disk and tape systems, sensors and actuators
- Communication
  - Suitable for communicating with remote devices such as a terminal, a machine readable device, or another computer

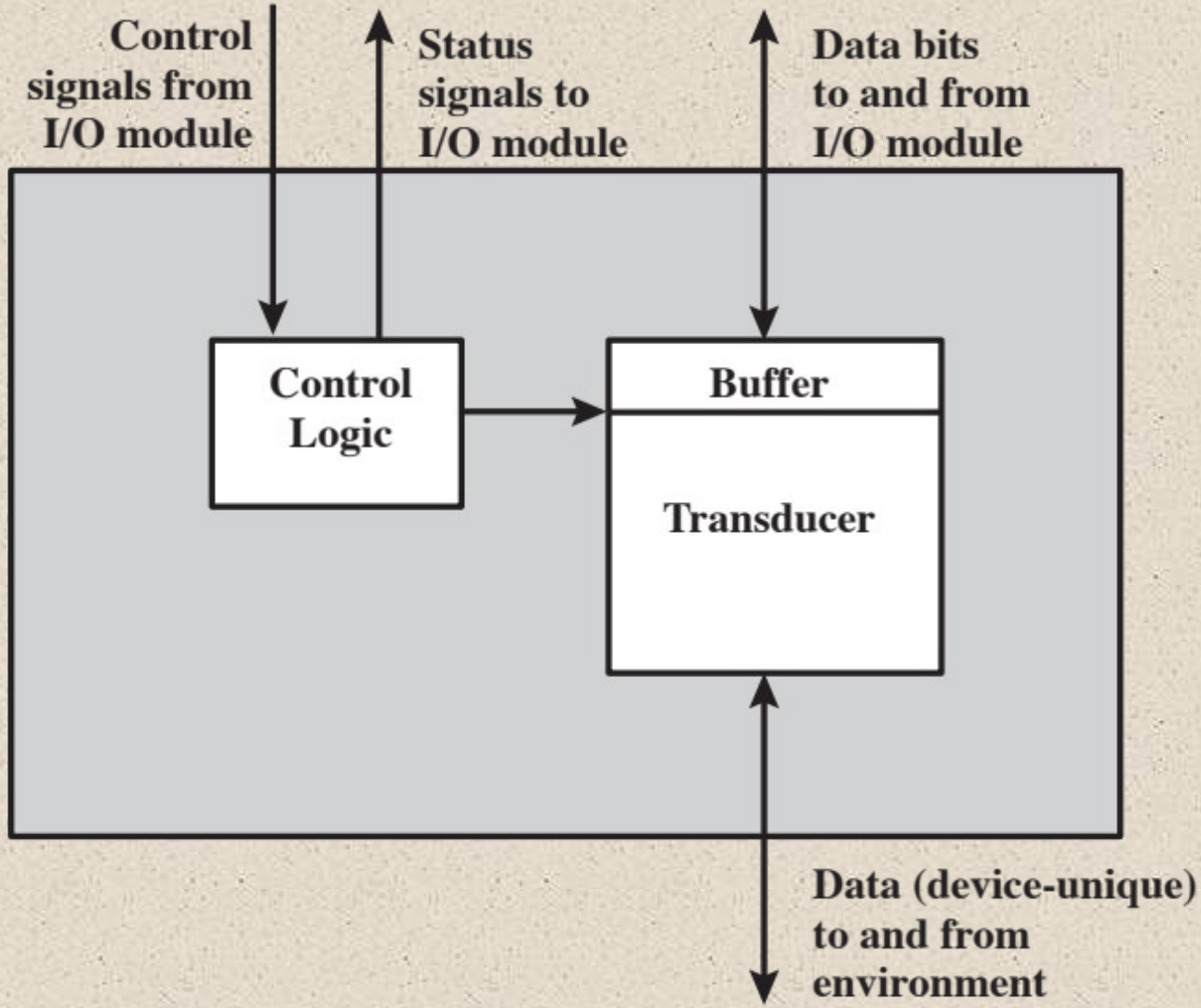


Figure 7.2 Block Diagram of an External Device

Keyboard/Monitor

- Most common means of computer/user interaction
- User provides input through the keyboard
- The monitor displays data provided by the computer

International Reference Alphabet (IRA)

- Basic unit of exchange is the character
  - Associated with each character is a code
  - Each character in this code is represented by a unique 7-bit binary code
    - 128 different characters can be represented
- Characters are of two types:
  - Printable
    - Alphabetic, numeric, and special characters that can be printed on paper or displayed on a screen
  - Control
    - Have to do with controlling the printing or displaying of characters
    - Example is carriage return
    - Other control characters are concerned with communications procedures

Keyboard Codes

- When the user depresses a key it generates an electronic signal that is interpreted by the transducer in the keyboard and translated into the bit pattern of the corresponding IRA code
- This bit pattern is transmitted to the I/O module in the computer
- On output, IRA code characters are transmitted to an external device from the I/O module
- The transducer interprets the code and sends the required electronic signals to the output device either to display the indicated character or perform the requested control function

The major functions for an I/O module fall into the following categories:

Control and timing
- Coordinates the flow of traffic between internal resources and external devices

Processor communication
- Involves command decoding, data, status reporting, address recognition

Device communication
- Involves commands, status information, and data

Data buffering
- Performs the needed buffering operation to balance device and memory speeds

Error detection
- Detects and reports transmission errors

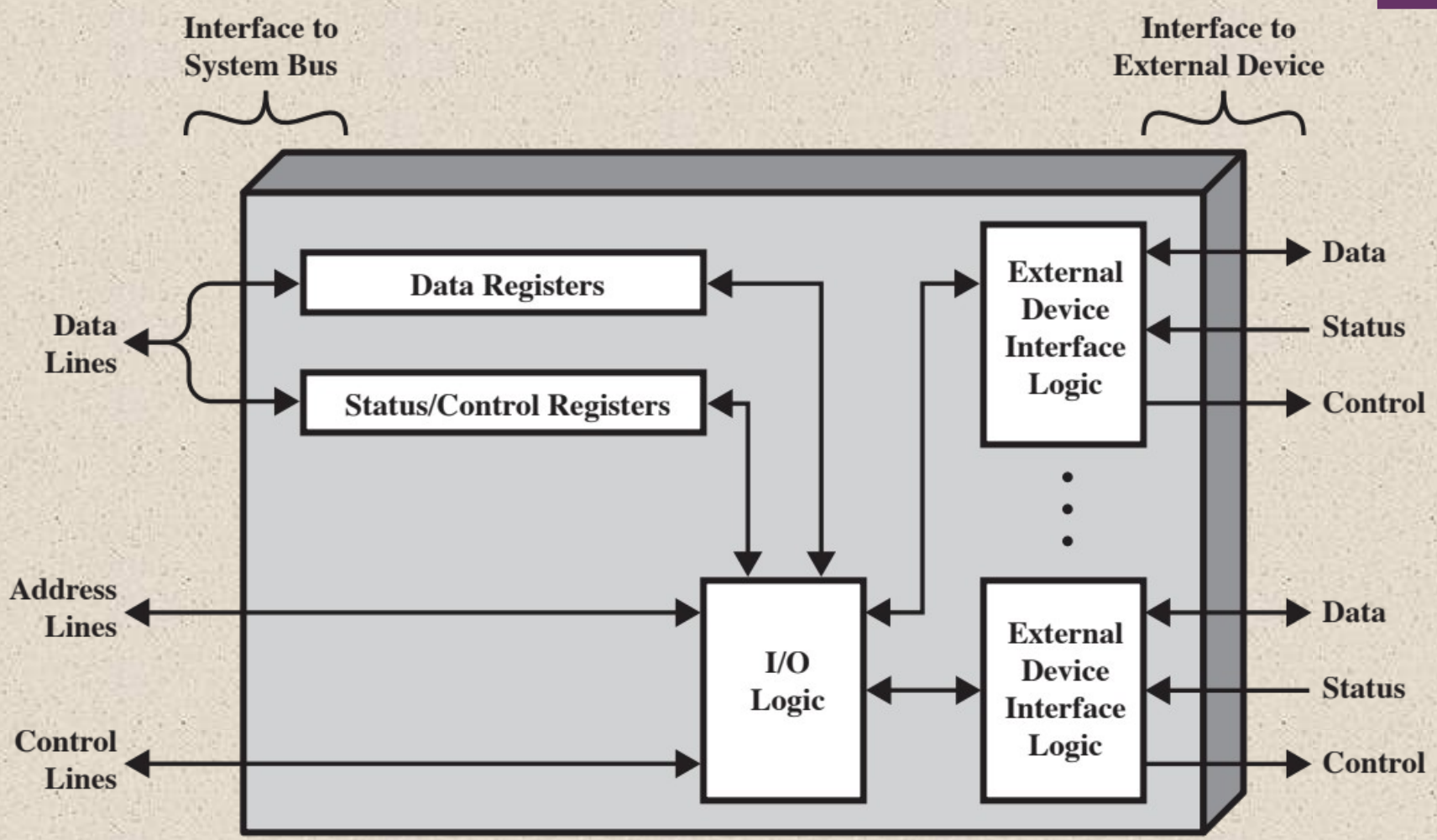


Figure 7.3 Block Diagram of an I/O Module

Programmed I/O

Three techniques are possible for I/O operations:

- Programmed I/O
  - Data are exchanged between the processor and the I/O module
  - Processor executes a program that gives it direct control of the I/O operation
  - When the processor issues a command it must wait until the I/O operation is complete
  - If the processor is faster than the I/O module this is wasteful of processor time
- Interrupt-driven I/O
  - Processor issues an I/O command, continues to execute other instructions, and is interrupted by the I/O module when the latter has completed its work
- Direct memory access (DMA)
  - The I/O module and main memory exchange data directly without processor involvement

Table 7.1
I/O Techniques

	No Interrupts	Use of Interrupts
I/O-to-memory transfer through processor	Programmed I/O	Interrupt-driven I/O
Direct I/O-to-memory transfer		Direct memory access (DMA)

I/O Commands

- There are four types of I/O commands that an I/O module may receive when it is addressed by a processor:

  1) Control
     - used to activate a peripheral and tell it what to do

  2) Test
     - used to test various status conditions associated with an I/O module and its peripherals

  3) Read
     - causes the I/O module to obtain an item of data from the peripheral and place it in an internal buffer

  4) Write
     - causes the I/O module to take an item of data from the data bus and subsequently transmit that data item to the peripheral

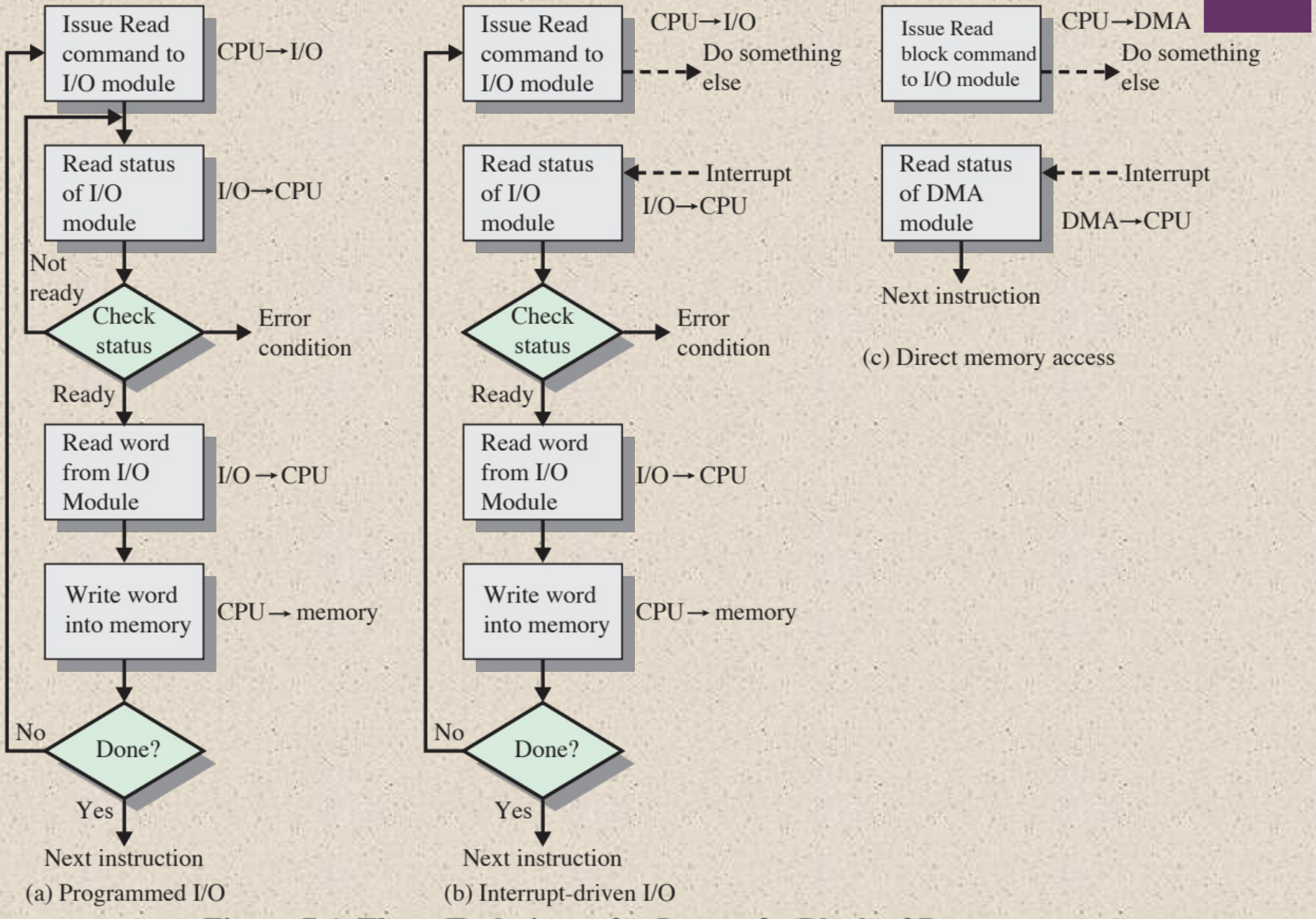


Figure 7.4 Three Techniques for Input of a Block of Data

I/O Instructions

With programmed I/O there is a close correspondence between the I/O-related instructions that the processor fetches from memory and the I/O commands that the processor issues to an I/O module to execute the instructions

The form of the instruction depends on the way in which external devices are addressed

Each I/O device connected through I/O modules is given a unique identifier or address

When the processor issues an I/O command, the command contains the address of the desired device

Thus each I/O module must interpret the address lines to determine if the command is for itself

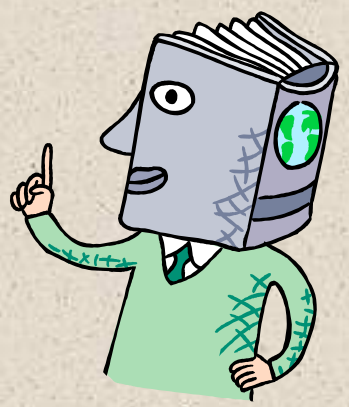
Memory-mapped I/O

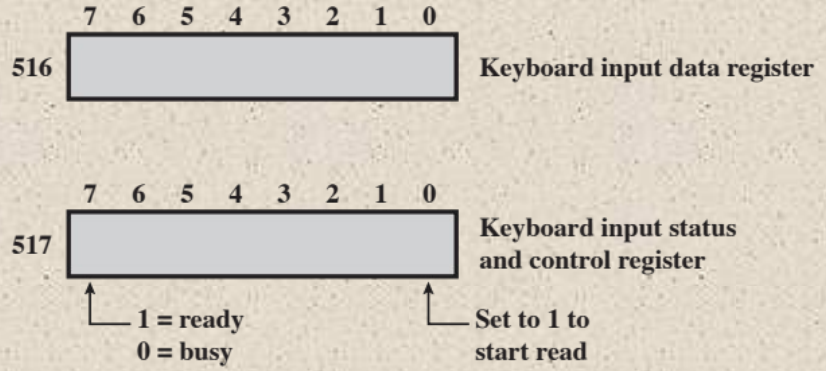
There is a single address space for memory locations and I/O devices

A single read line and a single write line are needed on the bus

I/O Mapping Summary

- Memory mapped I/O
  - Devices and memory share an address space
  - I/O looks just like memory read/write
  - No special commands for I/O
    - Large selection of memory access commands available
- Isolated I/O
  - Separate address spaces
  - Need I/O or memory select lines
  - Special commands for I/O
    - Limited set





ADDRESS	INSTRUCTION	OPERAND	COMMENT
200	Load AC	"1"	Load accumulator
	Store AC	517	Initiate keyboard read
202	Load AC	517	Get status byte
	Branch if Sign = 0	202	Loop until ready
	Load AC	516	Load data byte

(a) Memory-mapped I/O

ADDRESS	INSTRUCTION	OPERAND	COMMENT
200	Load I/O	5	Initiate keyboard read
201	Test I/O	5	Check for completion
	Branch Not Ready	201	Loop until complete
	In	5	Load data byte

(b) Isolated I/O

Figure 7.5 Memory-Mapped and Isolated I/O

Interrupt-Driven I/O

The problem with programmed I/O is that the processor has to wait a long time for the I/O module to be ready for either reception or transmission of data

An alternative is for the processor to issue an I/O command to a module and then go on to do some other useful work

The I/O module will then interrupt the processor to request service when it is ready to exchange data with the processor

The processor executes the data transfer and resumes its former processing

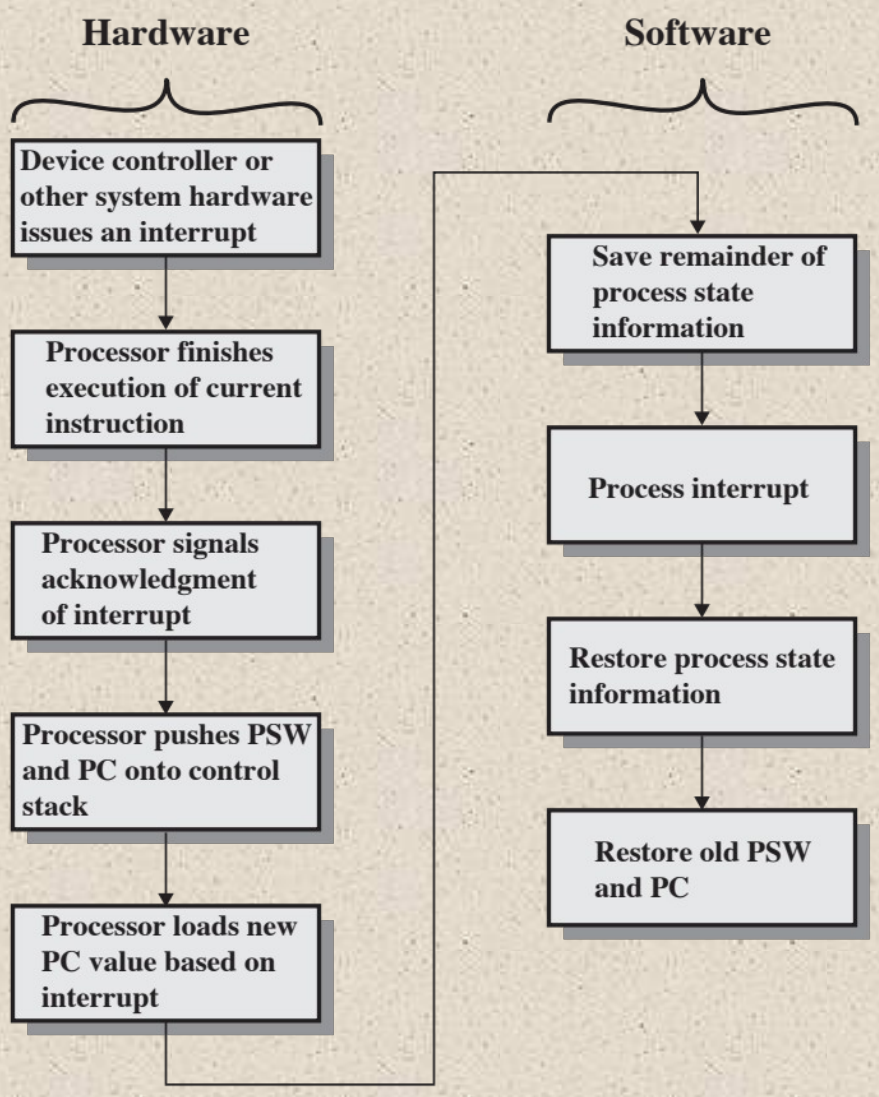


Figure 7.6 Simple Interrupt Processing

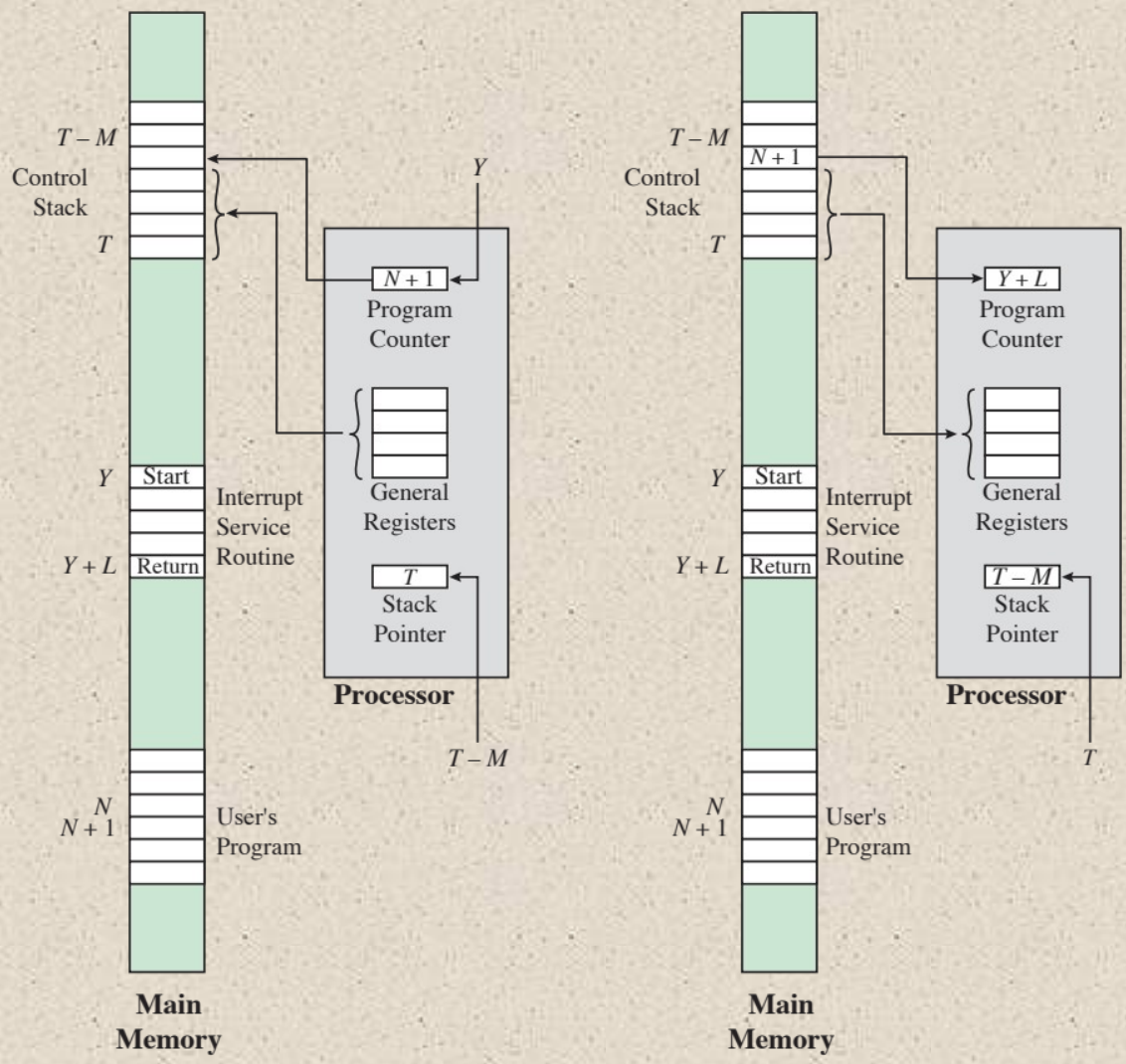


Figure 7.7 Changes in Memory and Registers for an Interrupt

Design Issues

Two design issues arise in implementing interrupt I/O:

- Because there will be multiple I/O modules how does the processor determine which device issued the interrupt?
- If multiple interrupts have occurred how does the processor decide which one to process?

Device Identification

Four general categories of techniques are in common use:

- **Multiple interrupt lines**
  - Between the processor and the I/O modules
  - Most straightforward approach to the problem
  - Consequently even if multiple lines are used, it is likely that each line will have multiple I/O modules attached to it
- **Software poll**
  - When processor detects an interrupt it branches to an interrupt-service routine whose job is to poll each I/O module to determine which module caused the interrupt
  - Time consuming
- **Daisy chain (hardware poll, vectored)**
  - The interrupt acknowledge line is daisy chained through the modules
  - Vector – address of the I/O module or some other unique identifier
  - Vectored interrupt – processor uses the vector as a pointer to the appropriate device-service routine, avoiding the need to execute a general interrupt-service routine first
- **Bus arbitration (vectored)**
  - An I/O module must first gain control of the bus before it can raise the interrupt request line
  - When the processor detects the interrupt it responds on the interrupt acknowledge line
  - Then the requesting module places its vector on the data lines

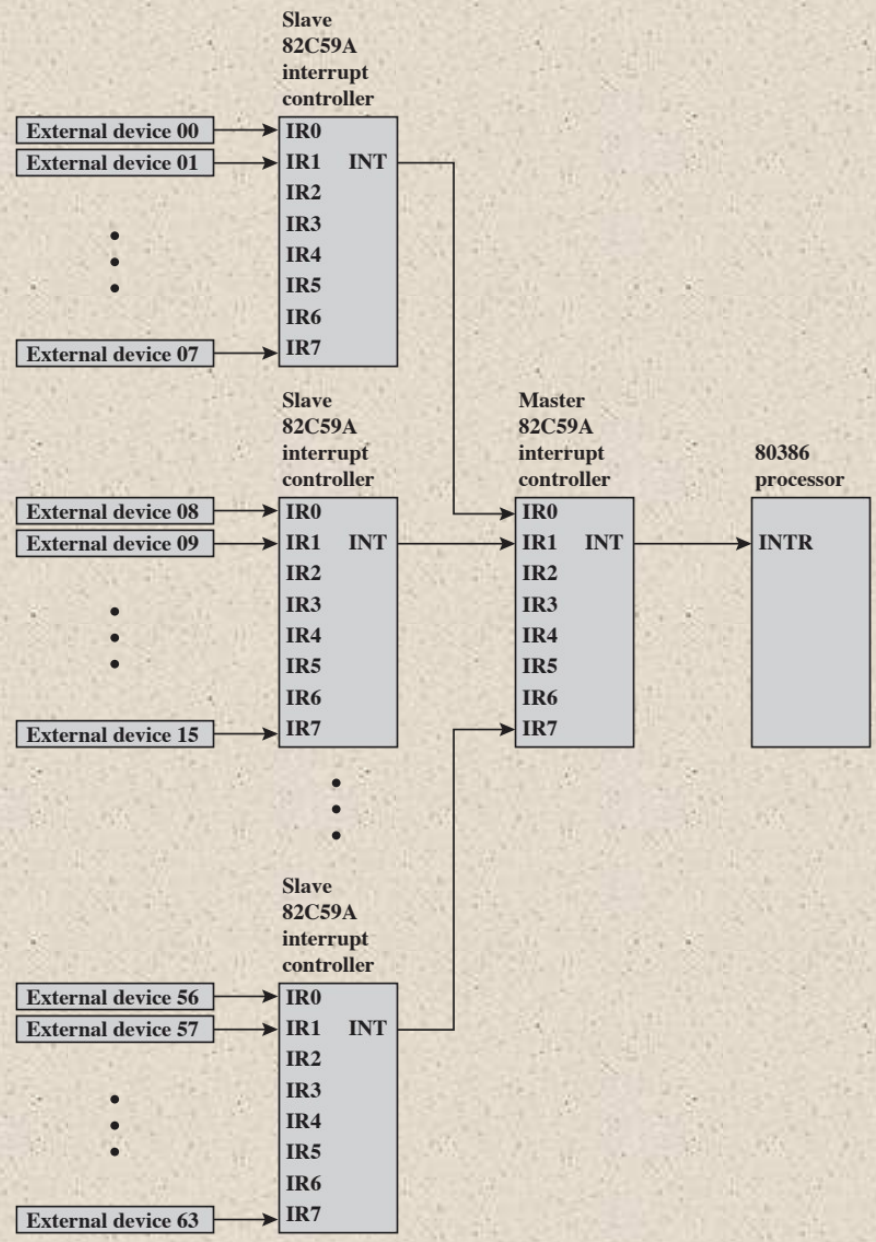


Figure 7.8 Use of the 82C59A Interrupt Controller

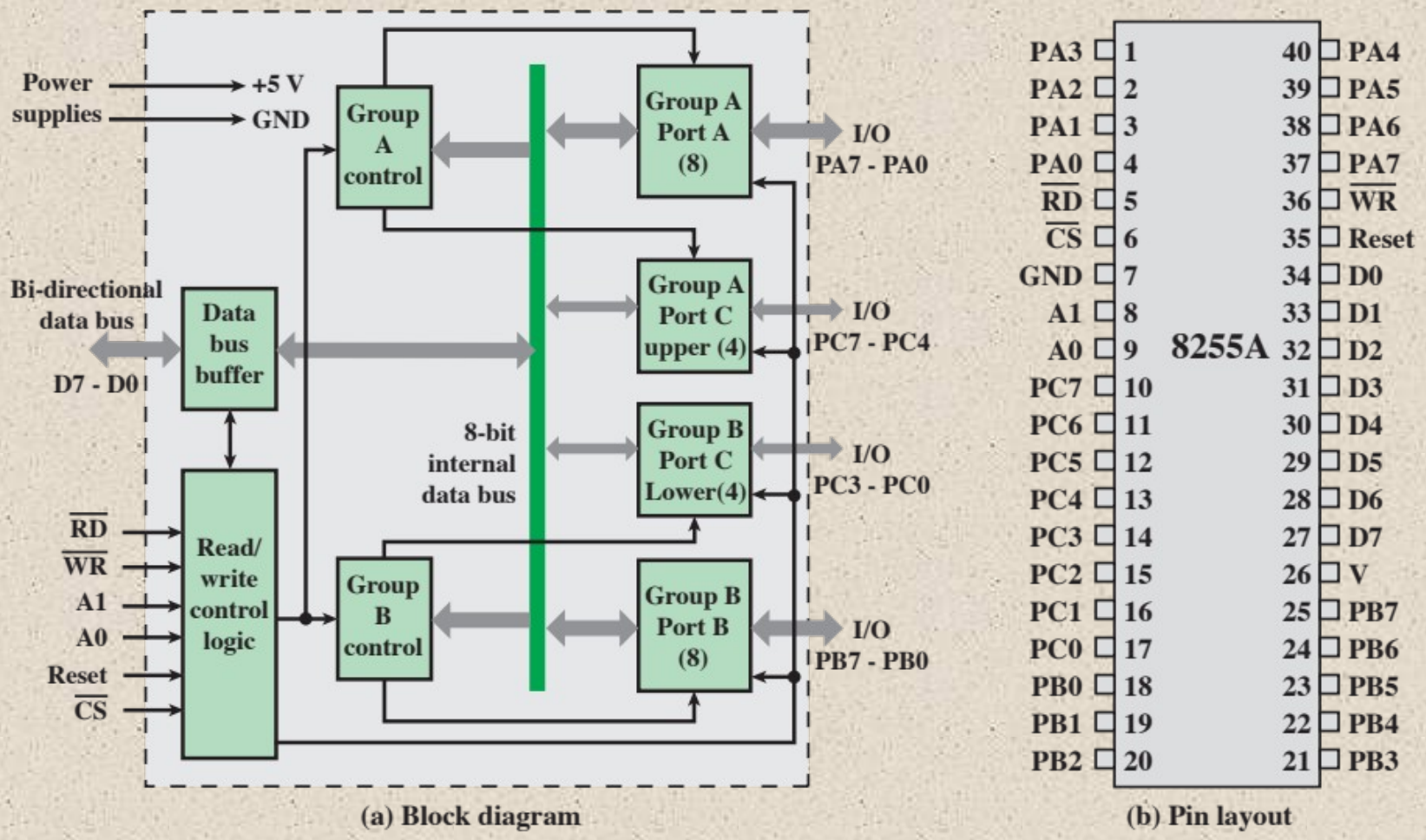
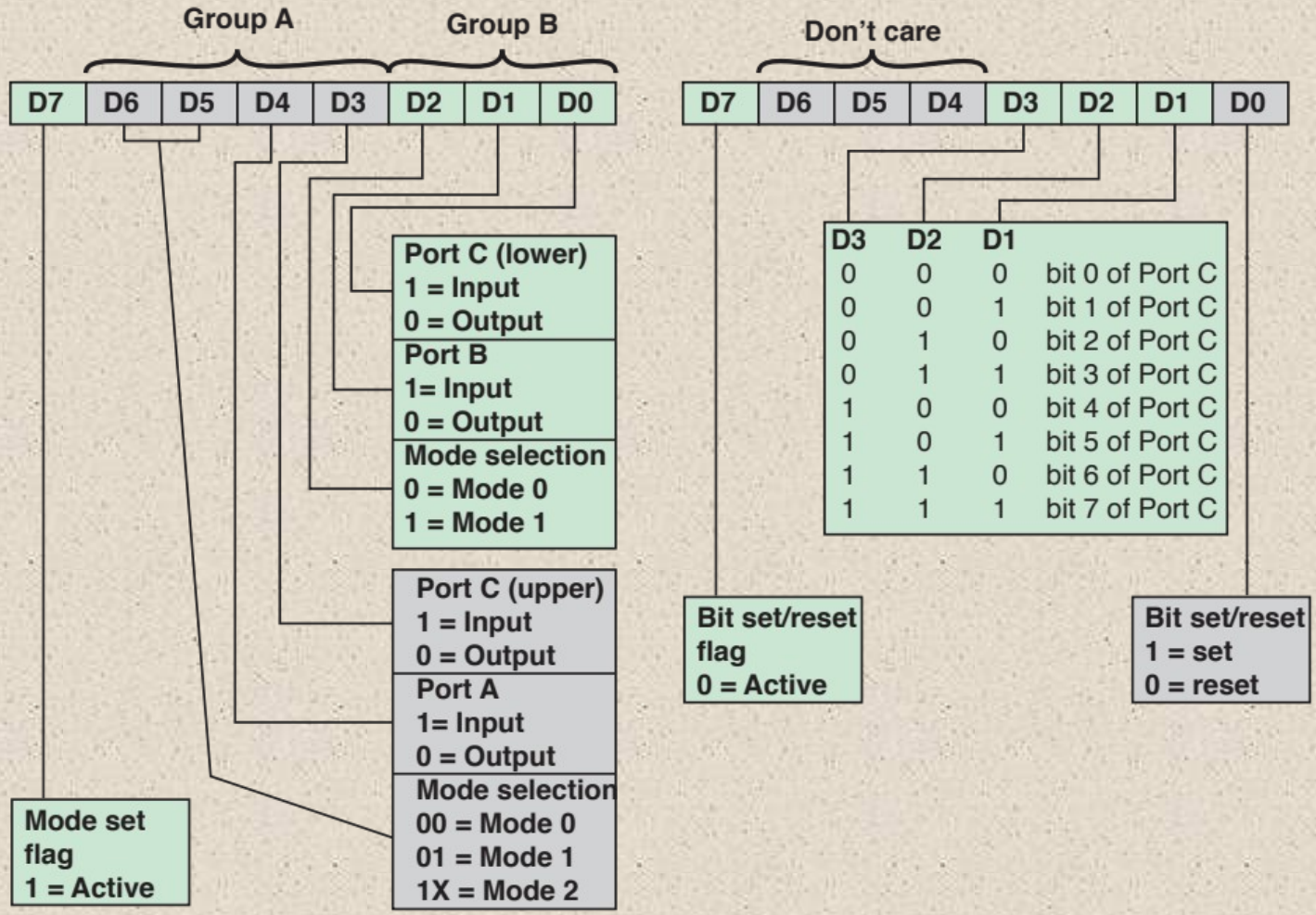


Figure 7.9 The Intel 8255A Programmable Peripheral Interface



(a) Mode definition of the 8255 control register to configure the 8255

(b) Bit definitions of the 8255 control register to modify single bits of port C

Figure 7.10 The Intel 8255A Control Word

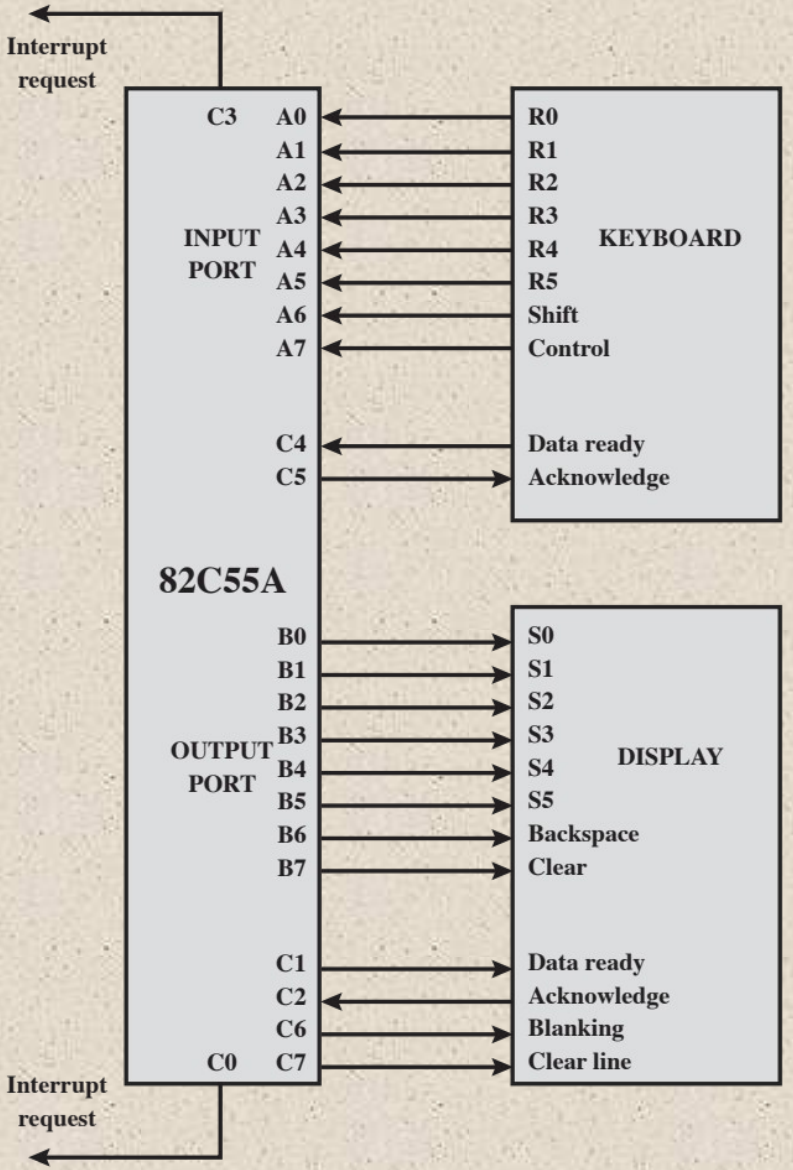


Figure 7.11 Keyboard/Display Interface to 82C55A

Drawbacks of Programmed and Interrupt-Driven I/O

- Both forms of I/O suffer from two inherent drawbacks:
  1) The I/O transfer rate is limited by the speed with which the processor can test and service a device
  2) The processor is tied up in managing an I/O transfer; a number of instructions must be executed for each I/O transfer

- When large volumes of data are to be moved a more efficient technique is *direct memory access* (DMA)

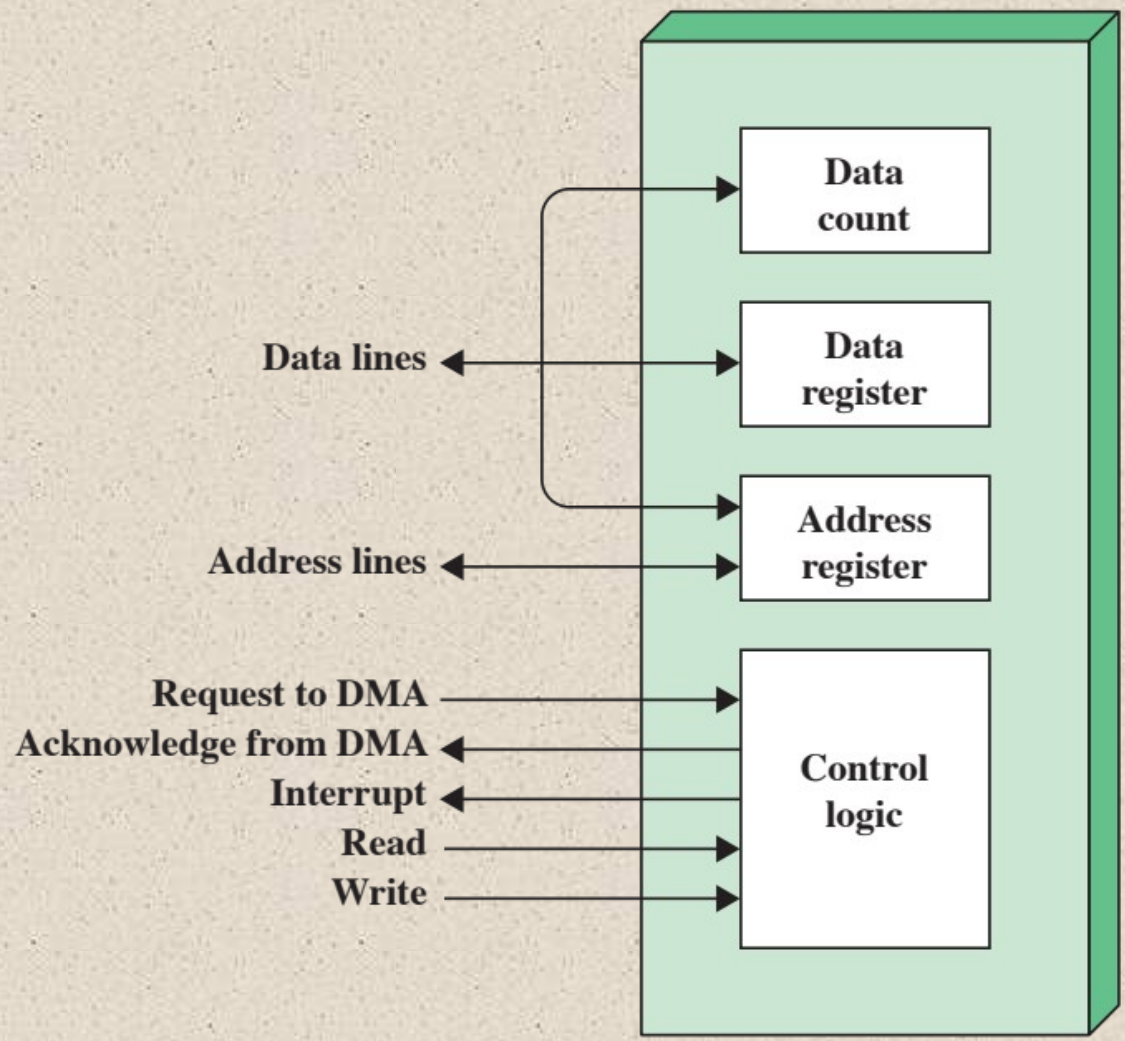


Figure 7.12 Typical DMA Block Diagram

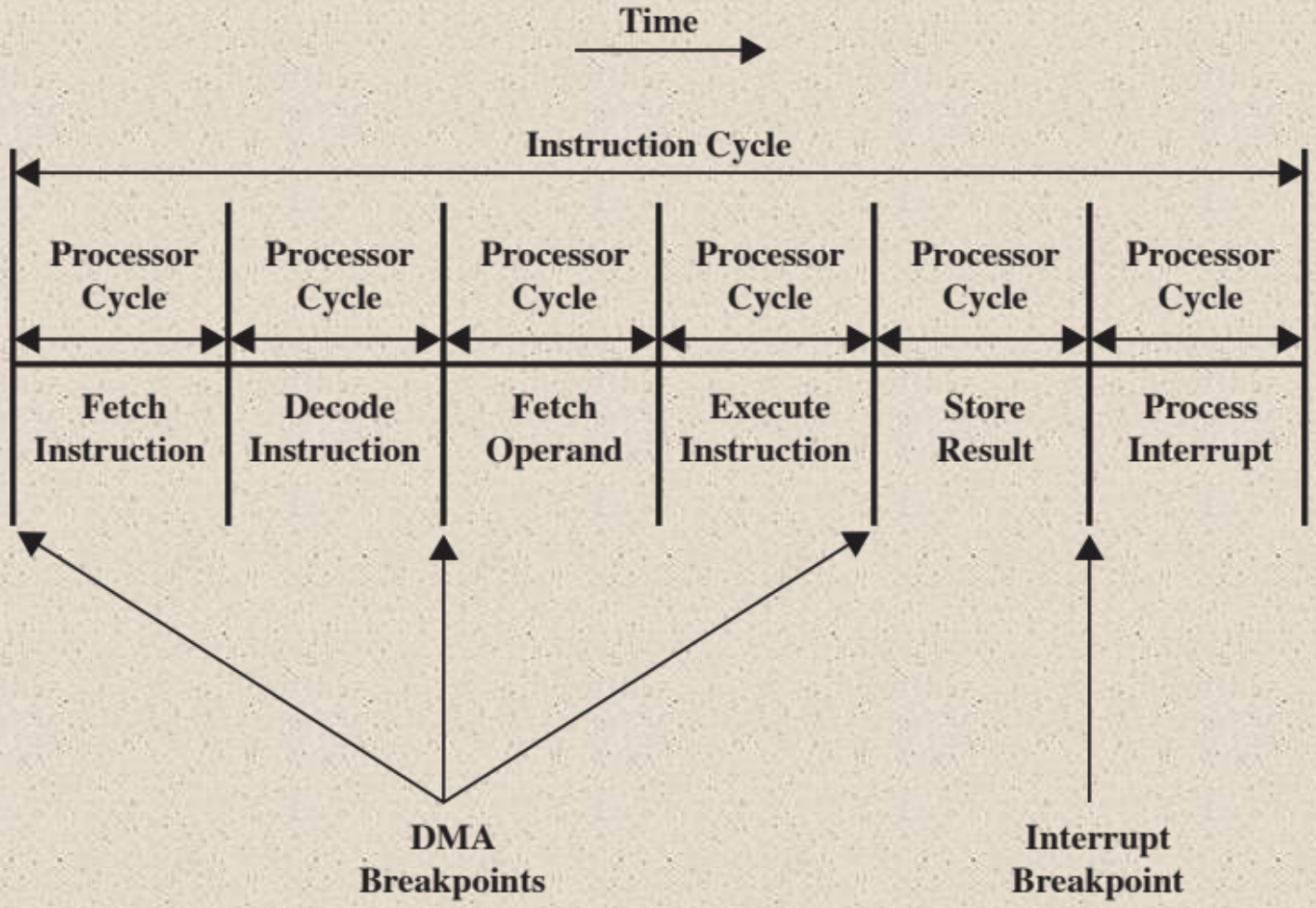


Figure 7.13 DMA and Interrupt Breakpoints During an Instruction Cycle

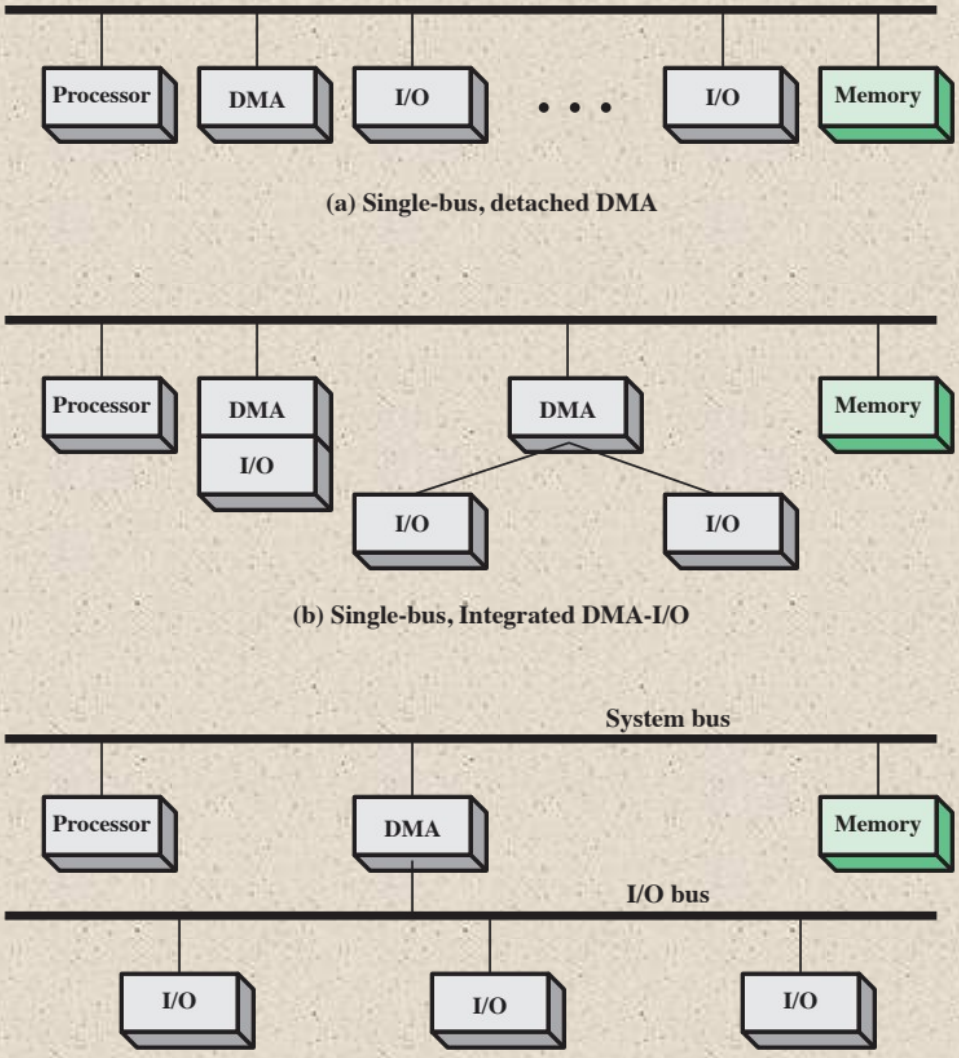
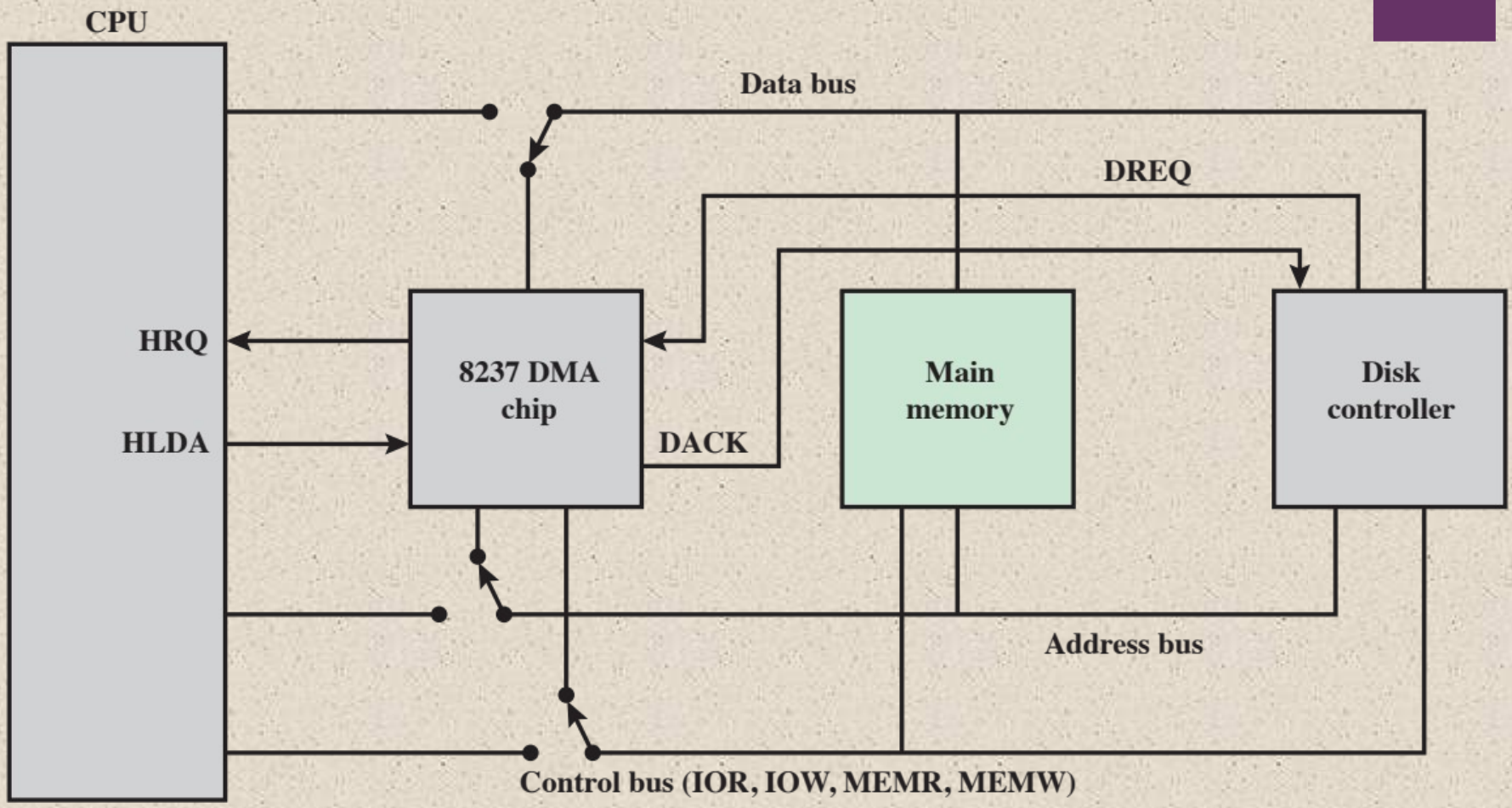


Figure 7.14 Alternative DMA Configurations



DACK = DMA acknowledge
DREQ = DMA request
HLDA = HOLD acknowledge
HRQ = HOLD request

Figure 7.15 8237 DMA Usage of System Bus

Fly-By DMA Controller

Data does not pass through and is not stored in DMA chip

- DMA only between I/O port and memory
- Not between two I/O ports or two memory locations

Can do memory to memory via register

8237 contains four DMA channels

- Programmed independently
- Any one active
- Numbered 0, 1, 2, and 3

Table 7.2

Intel 8237A Registers

Bit	Command	Status	Mode	Single Mask	All Mask
D0	Memory-to-memory E/D	Channel 0 has reached TC	Channel select	Select channal mask bit	Clear/set channel 0 mask bit
D1	Channel 0 address hold E/D	Channel 1 has reached TC			Clear/set channel 1 mask bit
D2	Controller E/D	Channel 2 has reached TC		Clear/set mask bit	Clear/set channel 2 mask bit
D3	Normal/compressed timing	Channel 3 has reached TC	Verify/write/read transfer	Not used	Clear/set channel 3 mask bit
D4	Fixed/rotating priority	Channel 0 request	Auto-initialization E/D		Not used
D5	Late/extended write selection	Channel 0 request	Address increment/decrement select		
D6	DREQ sense active high/low	Channel 0 request			
D7	DACK sense active high/low	Channel 0 request	Demand/single/block/cascade mode select		

E/D = enable/disable
TC = terminal count

Direct Cache Access (DCA)

- DMA is not able to scale to meet the increased demand due to dramatic increases in data rates for network I/O
- Demand is coming primarily from the widespread deployment of 10-Gbps and 100-Gbps Ethernet switches to handle massive amounts of data transfer to and from database servers and other high-performance systems
- Another source of traffic comes from Wi-Fi in the gigabit range
- Network Wi-Fi devices that handle 3.2 Gbps and 6.76 Gbps are becoming widely available and producing demand on enterprise systems

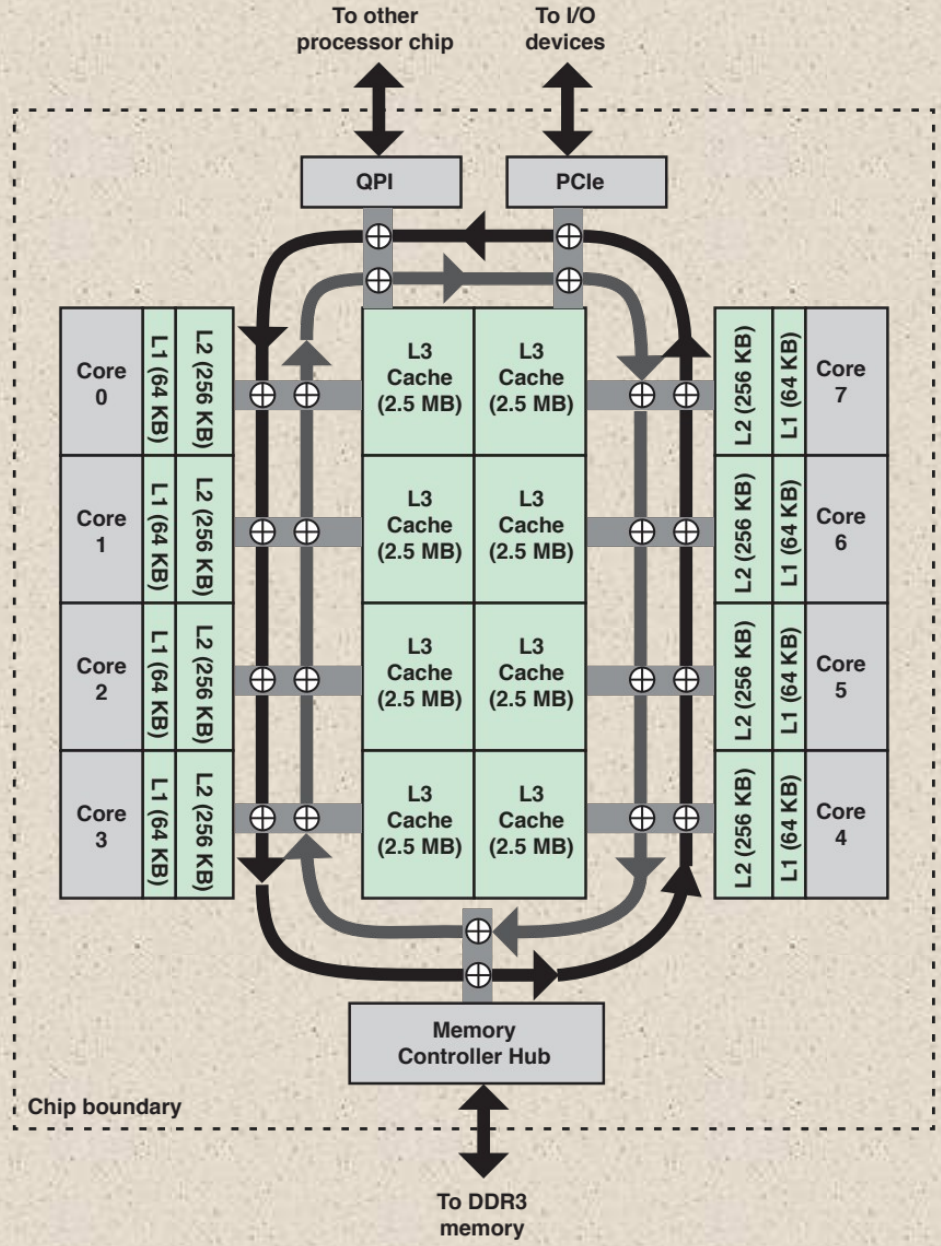


Figure 7.16 Xeon E5-2600/4600 Chip Architecture

Cache-Related Performance Issues

- Network traffic is transmitted in the form of a sequence of protocol blocks called packets or protocol data units
- The lowest, or link, level protocol is typically Ethernet, so that each arriving and departing block of data consists of an Ethernet packet containing as payload the higher-level protocol packet
- The higher-level protocols are usually the Internet Protocol (IP), operating on top of Ethernet and the Transmission Control Protocol (TCP), operating on top of IP
- The Ethernet payload consists of a block of data with a TCP header and an IP header
- For outgoing data, Ethernet packets are formed in a peripheral component, such as in I/O controller or network interface controller (NIC)
- For incoming traffic, the I/O controller strips off the Ethernet information and delivers the TCP/IP packet to the host CPU

Cache-Related Performance Issues

For both outgoing and incoming traffic the core, main memory, and cache are all involved

In a DMA scheme, when an application wishes to transmit data, it places that data in an application-assigned buffer in main memory

- The core transfers this to a system buffer in main memory and creates the necessary TCP and IP headers, which are also buffered in system memory
- The packet is then picked up via DMA for transfer via the NIC
- This activity engages not only main memory but also the cache
- Similar transfers between system and application buffers are required for incoming traffic

Packet Traffic Steps:

Incoming

- Packet arrives
- DMA
- NIC interrupts host
- Retrieve descriptors and headers
- Cache miss occurs
- Header is processed
- Payload transferred

Outgoing

- Packet transfer requested
- Packet created
- Output operation invoked
- DMA transfer
- NIC signals completion
- Driver frees buffer

Direct Cache Access Strategies

Simplest strategy was implemented as a prototype on a number of Intel Xeon processors between 2006 and 2010

This form of DCA applies only to incoming network traffic	The DCA function in the memory controller sends a prefetch hint to the core as soon as the data is available in system memory	This enables the core to prefetch the data packet from the system buffer

Much more substantial gains can be realized by avoiding the system buffer in main memory altogether

The packet and packet descriptor information are accessed only once in the system buffer by the core	For incoming packets, the core reads the data from the buffer and transfers the packet payload to an application buffer	It has no need to access that data in the system buffer again	Cache injection	Implemented in Intel's Xeon processor line, referred to as Direct Data I/O

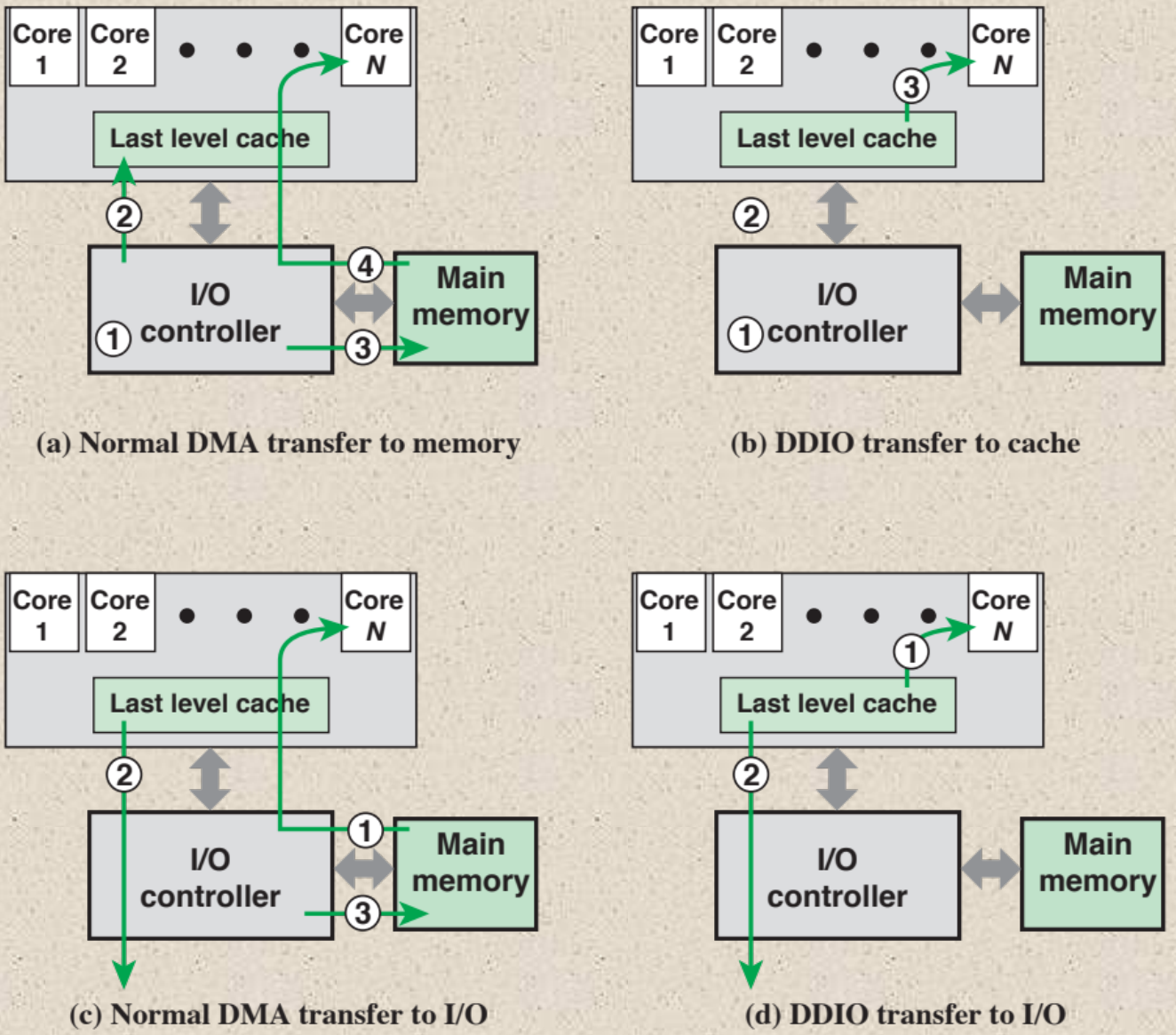


Figure 7.17 Comparison of DMA and DDIO

Evolution of the I/O Function

1. The CPU directly controls a peripheral device.
2. A controller or I/O module is added. The CPU uses programmed I/O without interrupts.
3. Same configuration as in step 2 is used, but now interrupts are employed. The CPU need not spend time waiting for an I/O operation to be performed, thus increasing efficiency.
4. The I/O module is given direct access to memory via DMA. It can now move a block of data to or from memory without involving the CPU, except at the beginning and end of the transfer.
5. The I/O module is enhanced to become a processor in its own right, with a specialized instruction set tailored for I/O
6. The I/O module has a local memory of its own and is, in fact, a computer in its own right. With this architecture a large set of I/O devices can be controlled with minimal CPU involvement.

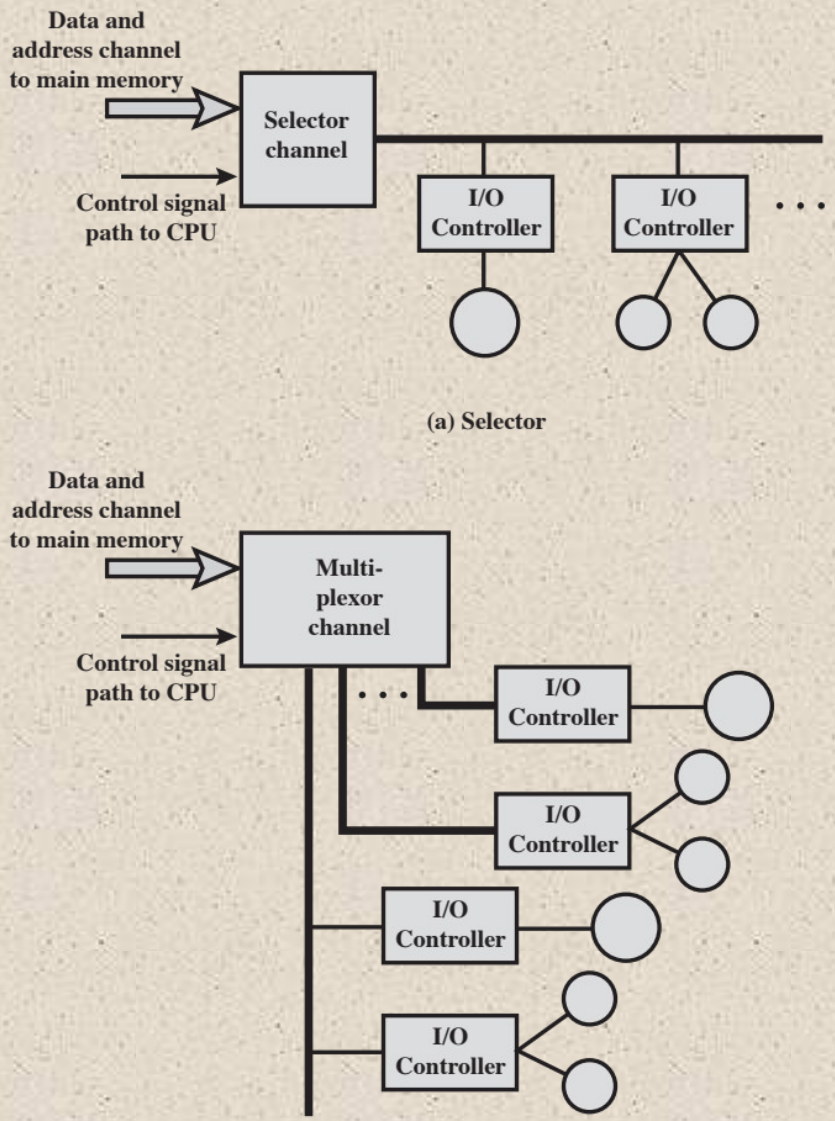


Figure 7.18 I/O Channel Architecture

Universal Serial Bus (USB)

- Widely used for peripheral connections
- Is the default interface for slower speed devices
- Commonly used high-speed I/O
- Has gone through multiple generations
  - USB 1.0
    - Defined a *Low Speed* data rate of 1.5 Mbps and a *Full Speed* rate of 12 Mbps
  - USB 2.0
    - Provides a data rate of 480 Mbps
  - USB 3.0
    - Higher speed bus called *SuperSpeed* in parallel with the USB 2.0 bus
    - Signaling speed of *SuperSpeed* is 5 Gbps, but due to signaling overhead the usable data rate is up to 4 Gbps
  - USB 3.1
    - Includes a faster transfer mode called *SuperSpeed+*
    - This transfer mode achieves a signaling rate of 10 Gbps and a theoretical usable data rate of 9.7 Gbps
- Is controlled by a root host controller which attaches to devices to create a local network with a hierarchical tree topology

FireWire Serial Bus

- Was developed as an alternative to small computer system interface (SCSI) to be used on smaller systems, such as personal computers, workstations, and servers
- Objective was to meet the increasing demands for high I/O rates while avoiding the bulky and expensive I/O channel technologies developed for mainframe and supercomputer systems
- IEEE standard 1394, for a High Performance Serial Bus
- Uses a daisy chain configuration, with up to 63 devices connected off a single port
- 1022 FireWire buses can be interconnected using bridges
- Provides for hot plugging which makes it possible to connect and disconnect peripherals without having to power the computer system down or reconfigure the system
- Provides for automatic configuration
- No terminations and the system automatically performs a configuration function to assign addresses

SCSI

- Small Computer System Interface
- A once common standard for connecting peripheral devices to small and medium-sized computers
- Has lost popularity to USB and FireWire in smaller systems
- High-speed versions remain popular for mass memory support on enterprise systems
- Physical organization is a shared bus, which can support up to 16 or 32 devices, depending on the generation of the standard
  - The bus provides for parallel transmission rather than serial, with a bus width of 16 bits on earlier generations and 32 bits on later generations
  - Speeds range from 5 Mbps on the original SCSI-1 specification to 160 Mbps on SCSI-3 U3

Thunderbolt

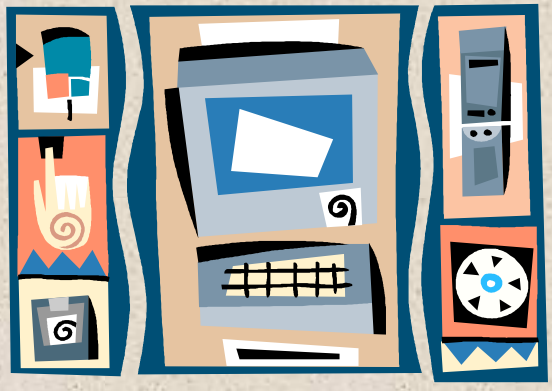
- Most recent and fastest peripheral connection technology to become available for general-purpose use
- Developed by Intel with collaboration from Apple
- The technology combines data, video, audio, and power into a single high-speed connection for peripherals such as hard drives, RAID arrays, video-capture boxes, and network interfaces
- Provides up to 10 Gbps throughput in each direction and up to 10 Watts of power to connected peripherals

InfiniBand

- I/O specification aimed at the high-end server market
- First version was released in early 2001
- Heavily relied on by IBM zEnterprise series of mainframes
- Standard describes an architecture and specifications for data flow among processors and intelligent I/O devices
- Has become a popular interface for storage area networking and other large storage configurations
- Enables servers, remote storage, and other network devices to be attached in a central fabric of switches and links
- The switch-based architecture can connect up to 64,000 servers, storage systems, and networking devices

PCI Express

- High-speed bus system for connecting peripherals of a wide variety of types and speeds



SATA

- Serial Advanced Technology Attachment
- An interface for disk storage systems
- Provides data rates of up to 6 Gbps, with a maximum per device of 300 Mbps
- Widely used in desktop computers and in industrial and embedded applications

Ethernet

- Predominant wired networking technology
- Has evolved to support data rates up to 100 Gbps and distances from a few meters to tens of km
- Has become essential for supporting personal computers, workstations, servers, and massive data storage devices in organizations large and small
- Began as an experimental bus-based 3-Mbps system
- Has moved from bus-based to switch-based
  - Data rate has periodically increased by an order of magnitude
  - There is a central switch with all of the devices connected directly to the switch
- Ethernet systems are currently available at speeds up to 100 Gbps

Wi-Fi

- Is the predominant wireless Internet access technology
- Now connects computers, tablets, smart phones, and other electronic devices such as video cameras TVs and thermostats
- In the enterprise has become an essential means of enhancing worker productivity and network effectiveness
- Public hotspots have expanded dramatically to provide free Internet access in most public places

- As the technology of antennas, wireless transmission techniques, and wireless protocol design has evolved, the IEEE 802.11 committee has been able to introduce standards for new versions of Wi-Fi at higher speeds
- Current version is 802.11ac (2014) with a maximum data rate of 3.2 Gbps

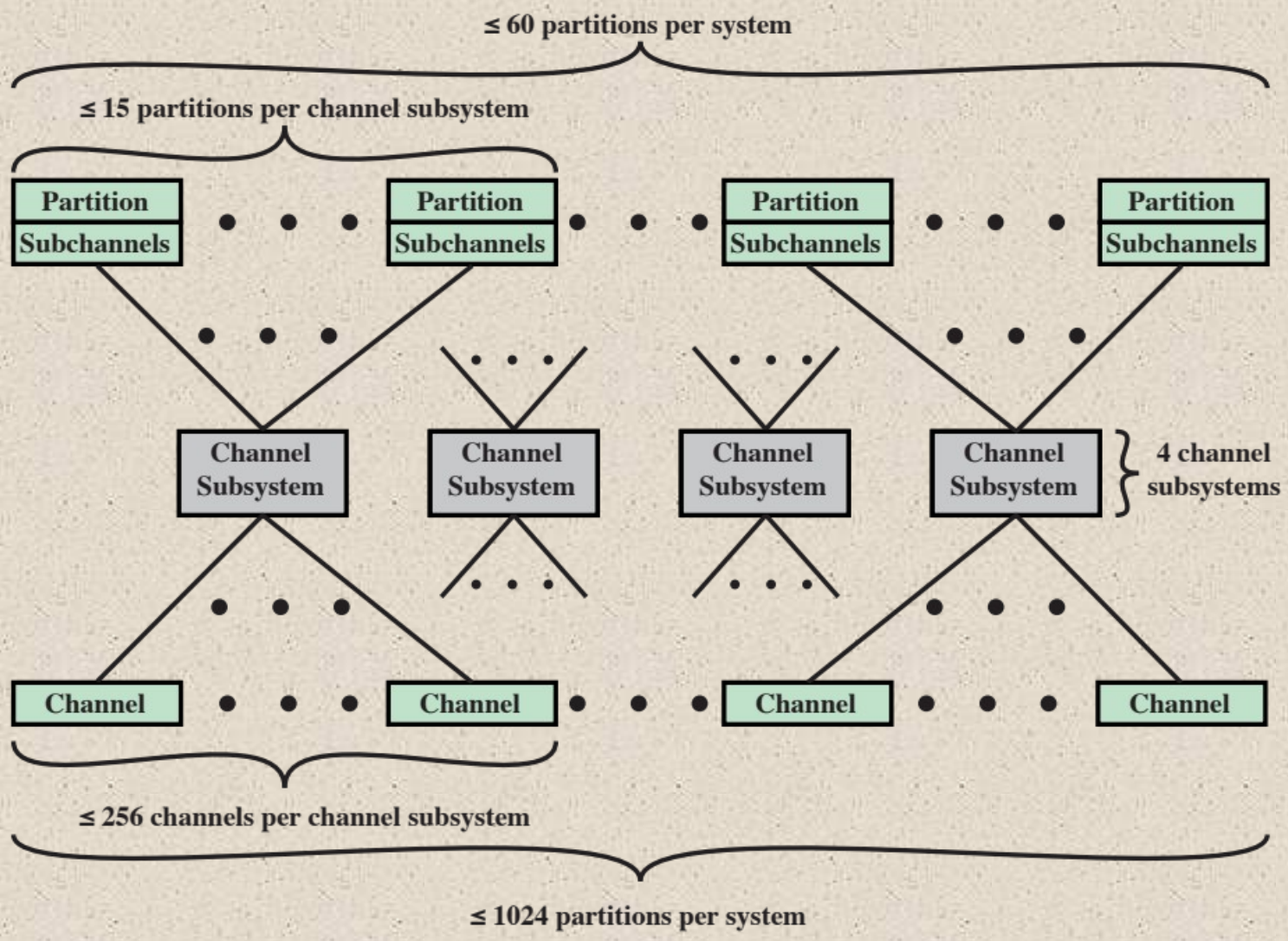
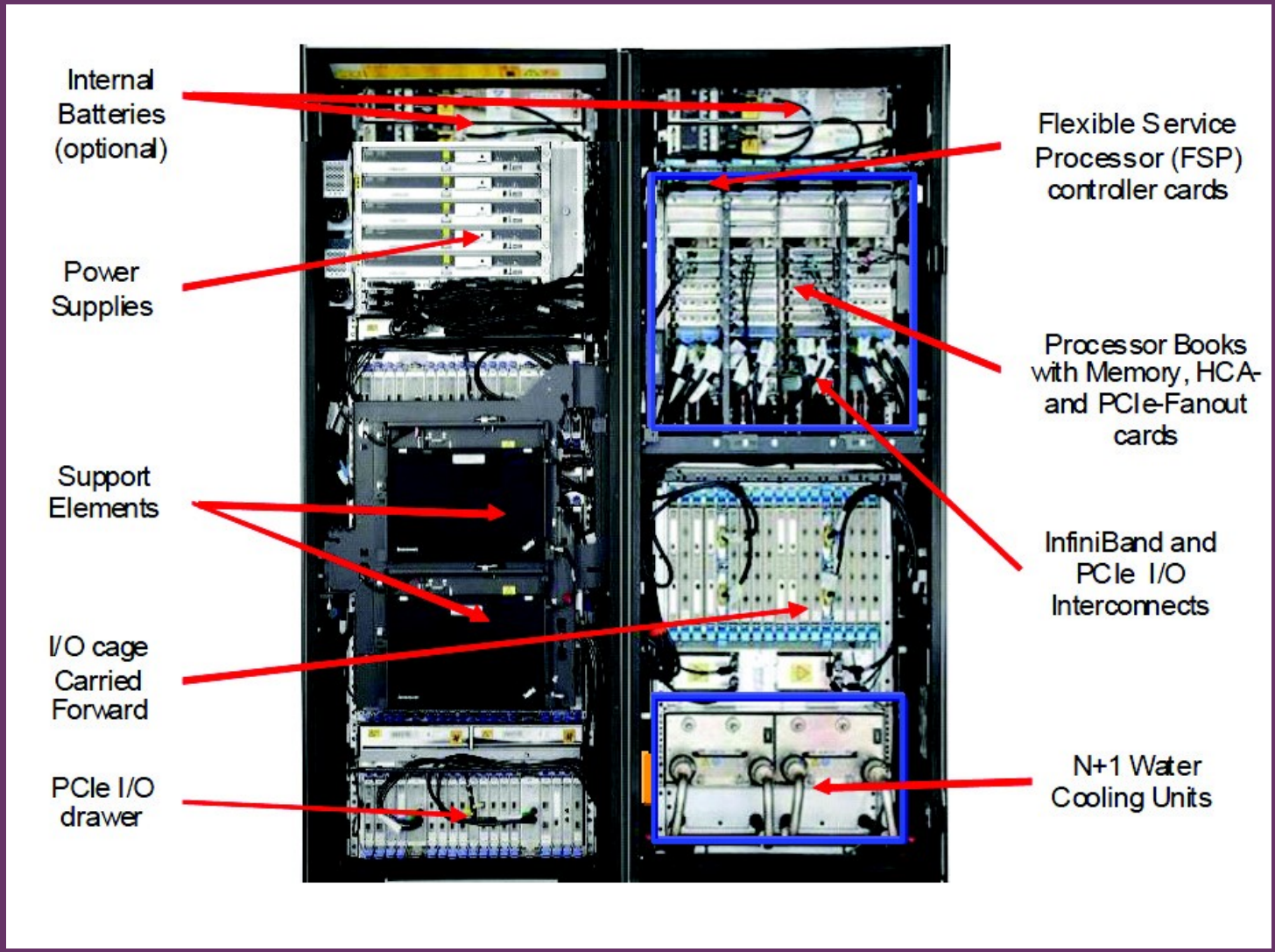


Figure 7.19 IBM EC12 I/O Channel Subsystem Structure

Figure 7.20
IBM zEC12 I/O Frames-Front View



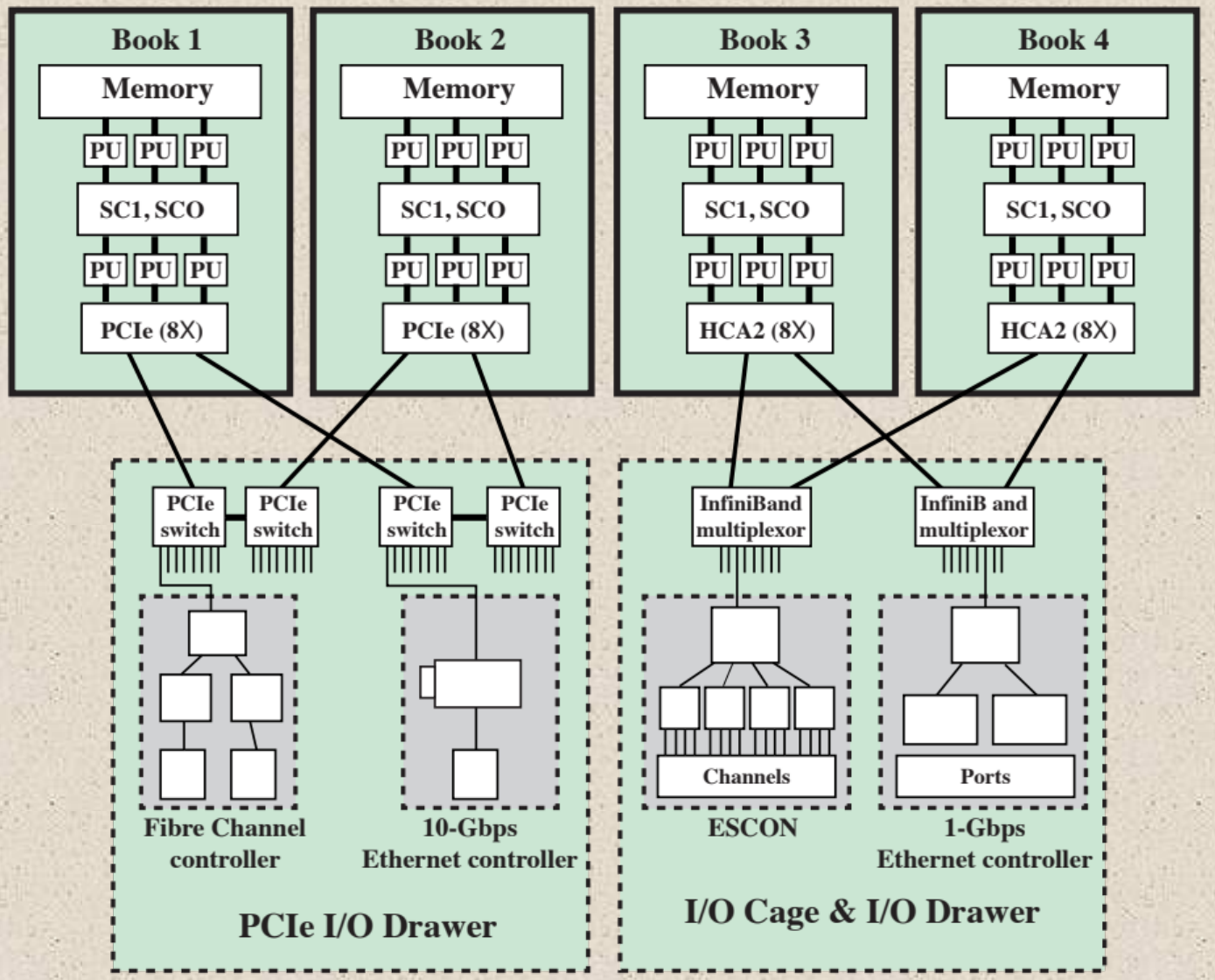


Figure 7.21 IBM EC12 I/O System Structure

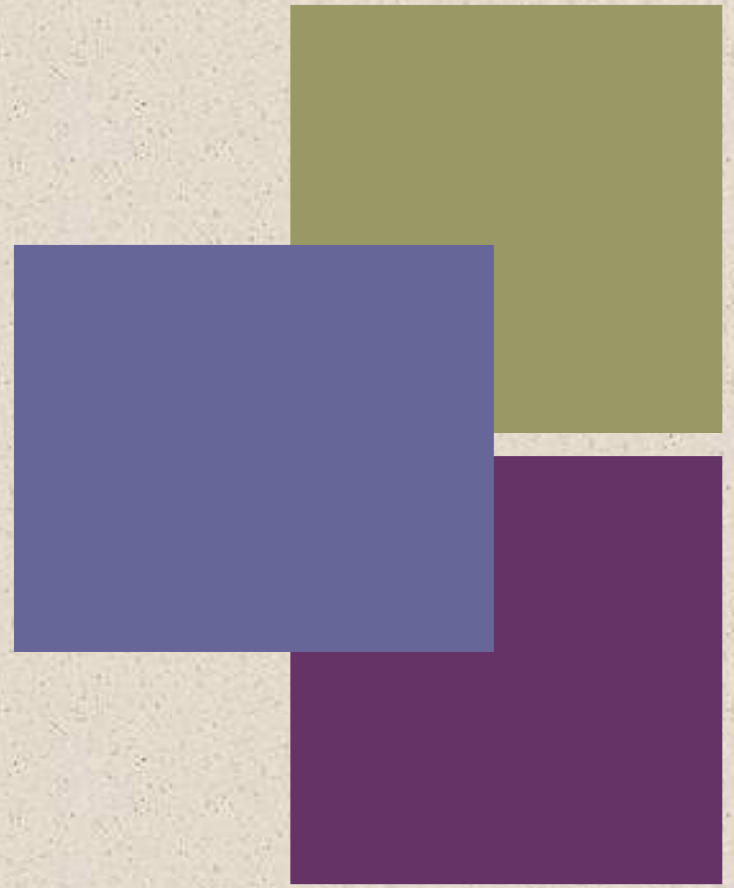
Summary

Chapter 7

Input/Output

- External devices
  - Keyboard/monitor
  - Disk drive
- I/O modules
  - Module function
  - I/O module structure
- Programmed I/O
  - Overview of programmed I/O
  - I/O commands/instructions
- Direct memory access
  - Drawbacks of programmed and interrupt-driven I/O
  - DMA function
  - Intel 8237A DMA controller
- Interrupt-driven I/O
  - Interrupt processing
  - Design issues
  - Intel 82C59A interrupt controller
  - Intel 82C55A programmable peripheral interface
- Direct Cache Access
  - DMA using shared last-level cache
  - Cache-related performance issues
  - Direct cache access strategies
  - Direct data I/O
- I/O channels and processors
  - The evolution of the I/O function
  - Characteristics of I/O channels

William Stallings
Computer Organization and Architecture
10th Edition



Chapter 8

Operating System Support

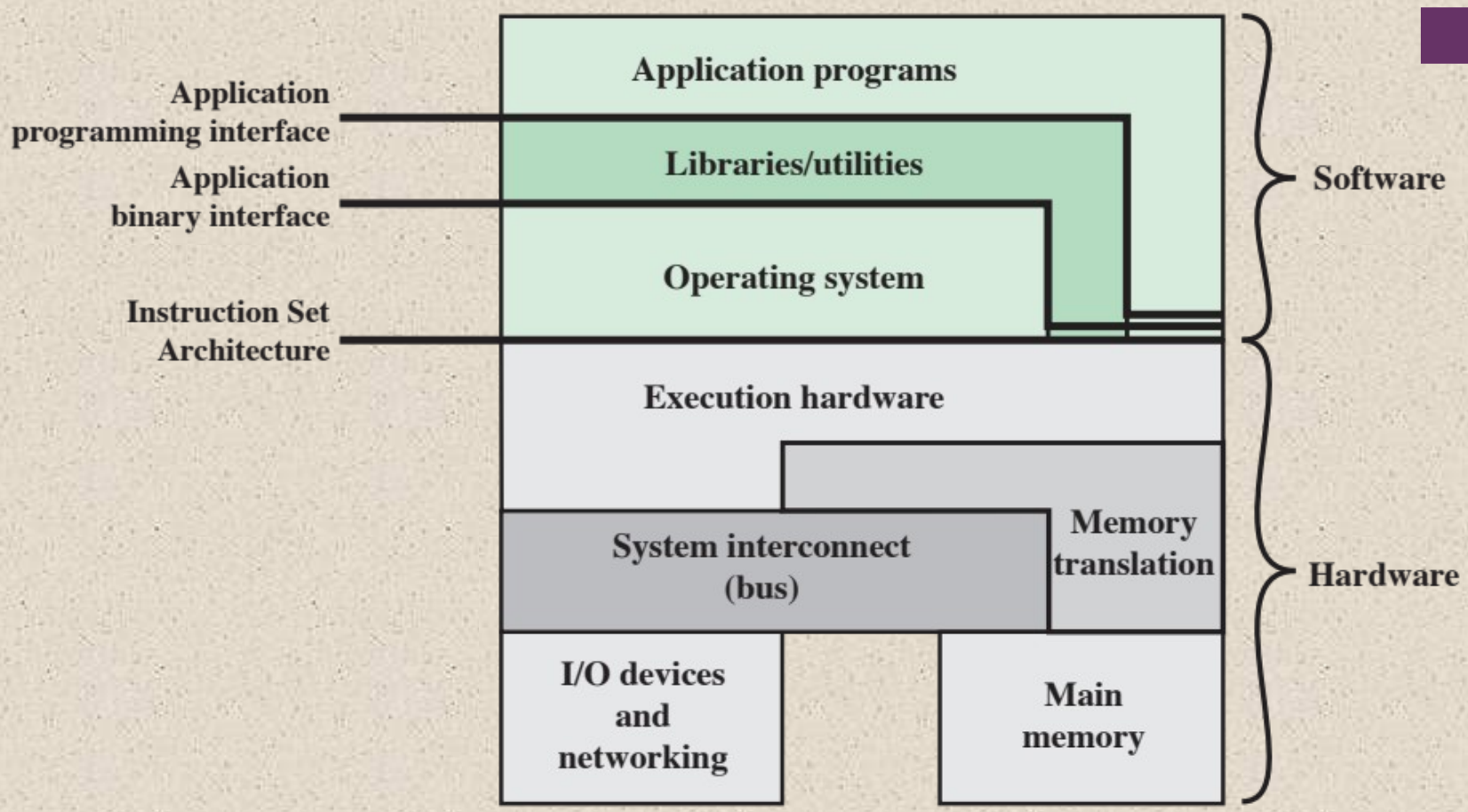


Figure 8.1 Computer Hardware and Software Structure

Operating System (OS) Services

- The most important system program
- Masks the details of the hardware from the programmer and provides the programmer with a convenient interface for using the system
- The OS typically provides services in the following areas:
  - Program creation
  - Program execution
  - Access to I/O devices
  - Controlled access to files
  - System access
  - Error detection and response
  - Accounting

Interfaces

- Key interfaces in a typical computer system:

Instruction set architecture (ISA)

- Defines the machine language instructions that a computer can follow
- Boundary between hardware and software

Application binary interface (ABI)

- Defines a standard for binary portability across programs
- Defines the system call interface to the operating system and the hardware resources and services available in a system through the user ISA

Application programming interface (API)

- Gives a program access to the hardware resources and services available in a system through the user ISA supplemented with high-level language (HLL) library calls
- Using an API enables application software to be ported easily to other systems that support the same API

Operating System as Resource Manager

A computer is a set of resources for the movement, storage, and processing of data and for the control of these functions

- The OS is responsible for managing these resources

The OS as a control mechanism is unusual in two respects:

- The OS functions in the same way as ordinary computer software – it is a program executed by the processor
- The OS frequently relinquishes control and must depend on the processor to allow it to regain control

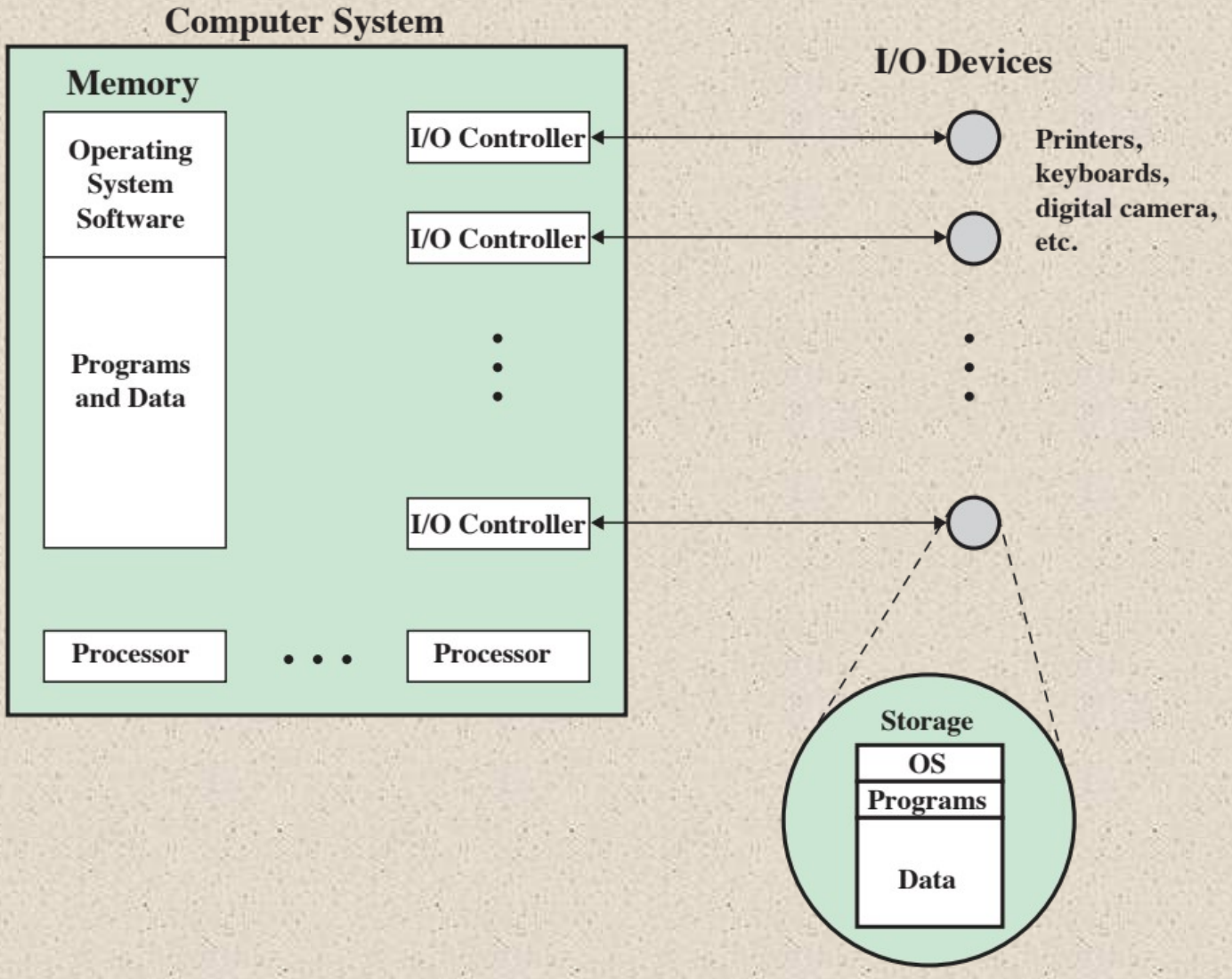


Figure 8.2 The Operating System as Resource Manager

Types of Operating Systems

- Interactive system
  - The user/programmer interacts directly with the computer to request the execution of a job or to perform a transaction
  - User may, depending on the nature of the application, communicate with the computer during the execution of the job
- Batch system
  - Opposite of interactive
  - The user's program is batched together with programs from other users and submitted by a computer operator
  - After the program is completed results are printed out for the user

Early Systems

- From the late 1940s to the mid-1950s the prog
  interacted directly with the computer hardware – ther
  OS
  - Processors were run from a console consisting of display lights, toggle switches, some form of input device and a printer
- Problems:
  - Scheduling
    - Sign-up sheets were used to reserve processor time
      - This could result in wasted computer idle time if the user finished early
      - If problems occurred the user could be forced to stop before resolving the problem
  - Setup time
    - A single program could involve
      - Loading the compiler plus the source program into memory
      - Saving the compiled program
      - Loading and linking together the object program and common functions

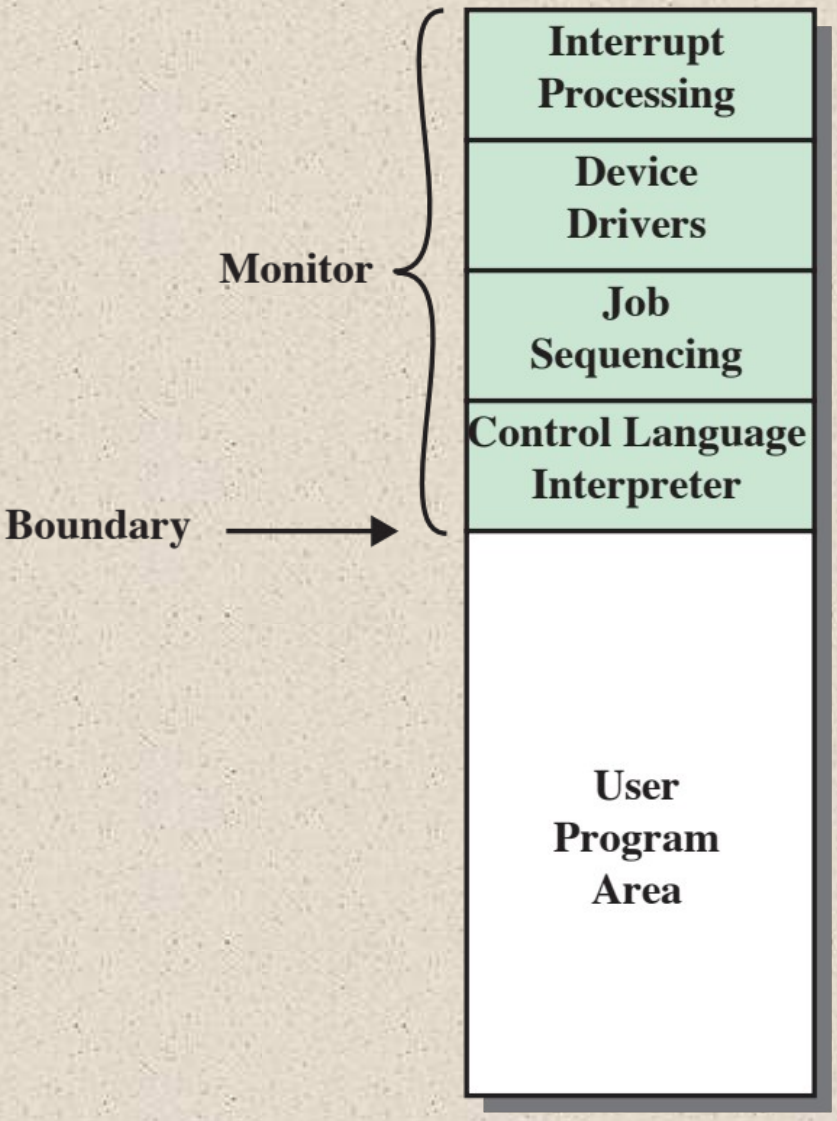


Figure 8.3 Memory Layout for a Resident Monitor

From the View of the Processor . . .

- Processor executes instructions from the portion of main memory containing the monitor
  - These instructions cause the next job to be read in another portion of main memory
  - The processor executes the instruction in the user's program until it encounters an ending or error condition
  - Either event causes the processor to fetch its next instruction from the monitor program
- The monitor handles setup and scheduling
  - A batch of jobs is queued up and executed as rapidly as possible with no idle time
- Job control language (JCL)
  - Special type of programming language used to provide instructions to the monitor
- Example:
  - $JOB
  - $FTN
  - ... Some Fortran instructions
  - $LOAD
  - $RUN
  - ... Some data
  - $END

**Each FORTRAN instruction and each item of data is on a separate punched card or a separate record on tape. In addition to FORTRAN and data lines, the job includes job control instructions, which are denoted by the beginning "$".

- Monitor, or batch OS, is simply a computer program
  - It relies on the ability of the processor to fetch instructions from various portions of main memory in order to seize and relinquish control alternately

Desirable Hardware Features

- Memory protection
  - User program must not alter the memory area containing the monitor
  - The processor hardware should detect an error and transfer control to the monitor
  - The monitor aborts the job, prints an error message, and loads the next job
- Timer
  - Used to prevent a job from monopolizing the system
  - If the timer expires an interrupt occurs and control returns to monitor
- Privileged instructions
  - Can only be executed by the monitor
  - If the processor encounters such an instruction while executing a user program an error interrupt occurs
  - I/O instructions are privileged so the monitor retains control of all I/O devices
- Interrupts
  - Gives the OS more flexibility in relinquishing control to and regaining control from user programs

Read one record from file	15 μs
Execute 100 instructions	1 μs
Write one record to file	15 μs
TOTAL	31 μs

$$\text{Percent CPU utilization} = \frac{1}{31} = 0.032 = 3.2\%$$

Figure 8.4 System Utilization Example

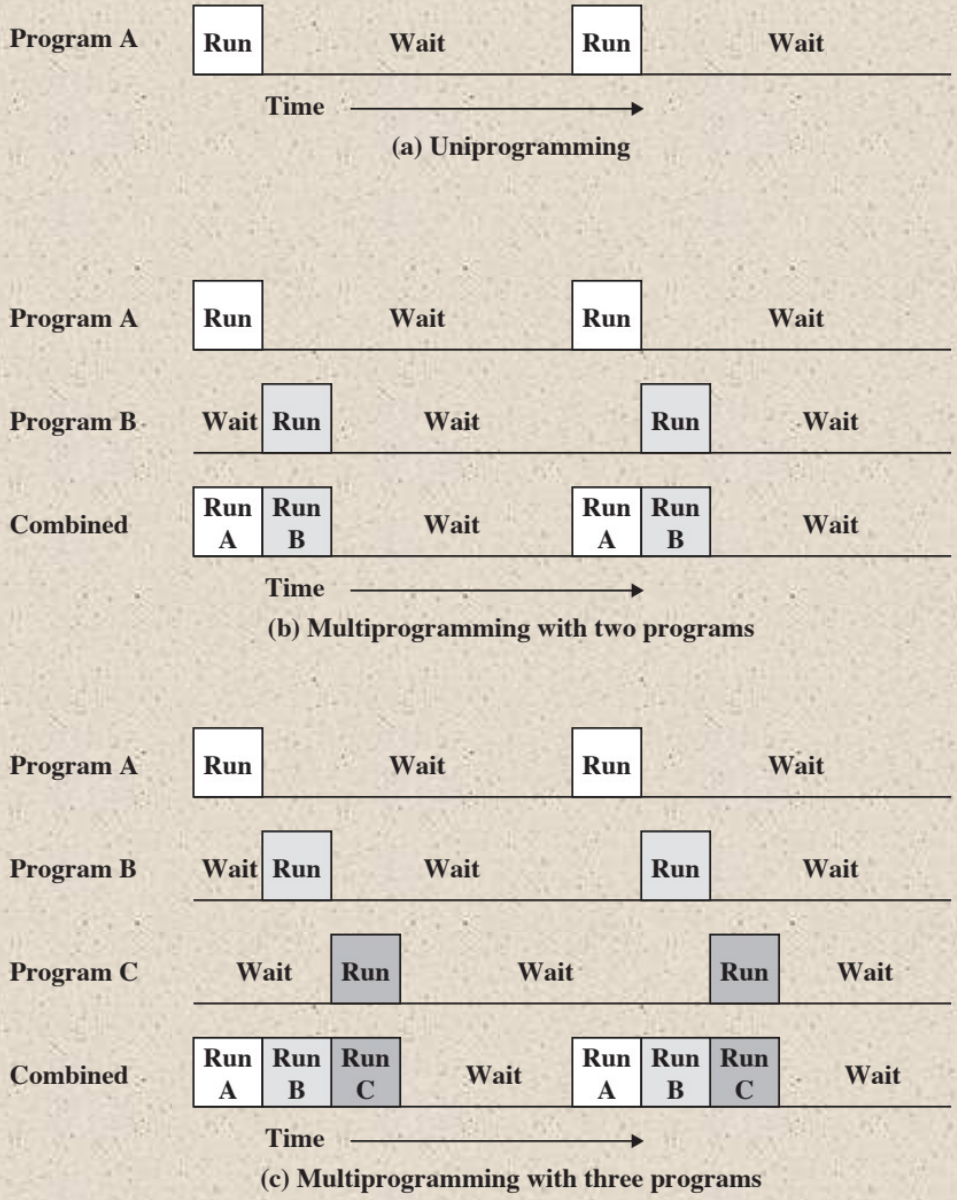


Figure 8.5 Multiprogramming Example

Table 8.1
Sample Program Execution Attributes

	JOB1	JOB2	JOB3
Type of job	Heavy compute	Heavy I/O	Heavy I/O
Duration	5 min	15 min	10 min
Memory required	50 M	100 M	80 M
Need disk?	No	No	Yes
Need terminal?	No	Yes	No
Need printer?	No	No	Yes

Table 8.2
Effects of Multiprogramming on Resource Utilization

	Uniprogramming	Multiprogramming
Processor use	20%	40%
Memory use	33%	67%
Disk use	33%	67%
Printer use	33%	67%
Elapsed time	30 min	15 min
Throughput rate	6 jobs/hr	12 jobs/hr
Mean response time	18 min	10 min

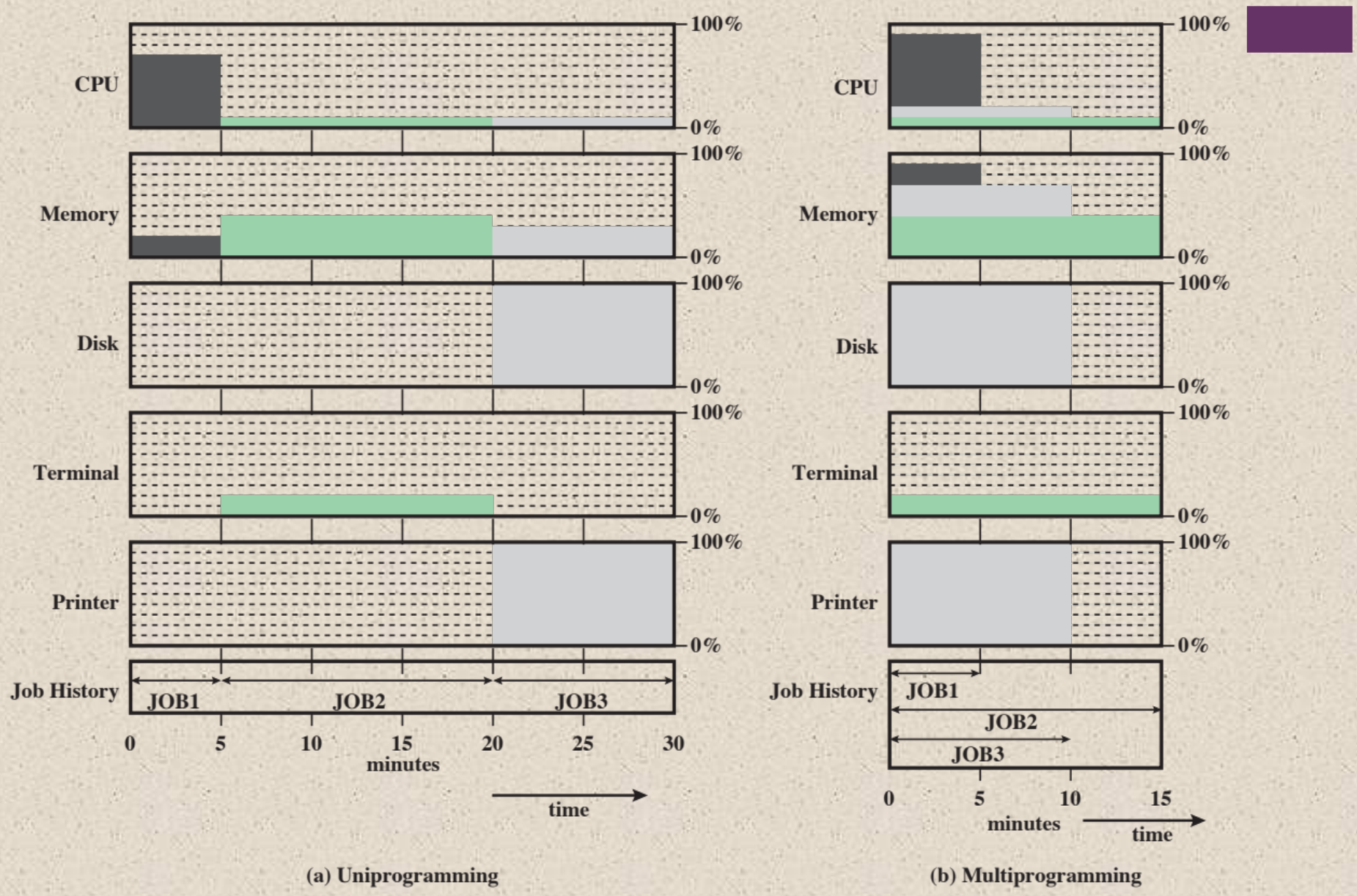


Figure 8.6 Utilization Histograms

Time Sharing Systems

- Used when the user interacts directly with the computer
- Processor's time is shared among multiple users
- Multiple users simultaneously access the system through terminals, with the OS interleaving the execution of each user program in a short burst or quantum of computation
- Example:
  - If there are *n* users actively requesting service at one time, each user will only see on the average 1/*n* of the effective computer speed

Table 8.3
Batch Multiprogramming versus Time Sharing

	Batch Multiprogramming	Time Sharing
Principal objective	Maximize processor use	Minimize response time
Source of directives to operating system	Job control language commands provided with the job	Commands entered at the terminal

Table 8.4 Types of Scheduling

Long-term scheduling	The decision to add to the pool of processes to be executed
Medium-term scheduling	The decision to add to the number of processes that are partially or fully in main memory
Short-term scheduling	The decision as to which available process will be executed by the processor
I/O scheduling	The decision as to which process's pending I/O request shall be handled by an available I/O device

Long Term Scheduling

Determines which programs are submitted for processing

Once submitted, a job becomes a process for the short term scheduler

In some systems a newly created process begins in a swapped-out condition, in which case it is added to a queue for the medium-term scheduler

Batch system

- Newly submitted jobs are routed to disk and held in a batch queue
- The long-term scheduler creates processes from the queue when it can

Time-sharing system

- A process request is generated when a user attempts to connect to the system
- OS will accept all authorized comers until the system is saturated
- At that point a connection request is met with a message indicating that the system is full and to try again later

Medium-Term Scheduling and Short-Term Scheduling

Medium-Term

- Part of the swapping function
- Swapping-in decision is based on the need to manage the degree of multiprogramming
- Swapping-in decision will consider the memory requirements of the swapped-out processes

Short-Term

- Also known as the dispatcher
- Executes frequently and makes the fine-grained decision of which job to execute next

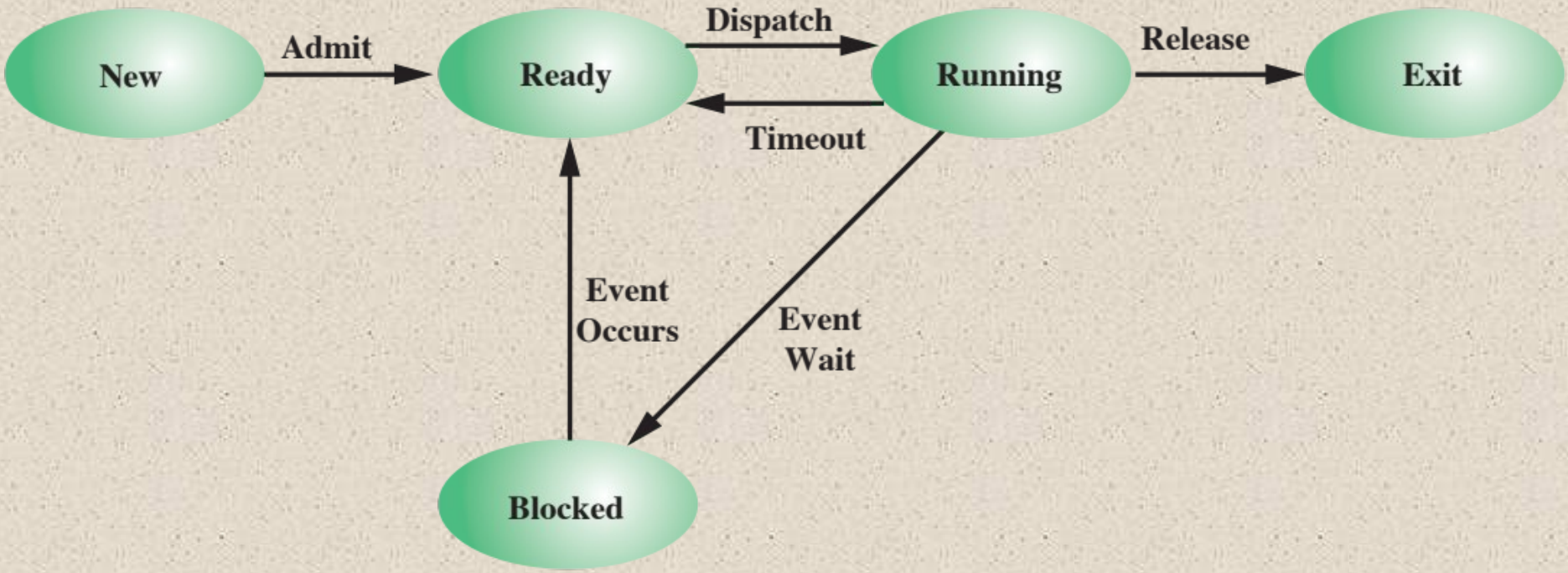


Figure 8.7 Five-State Process Model

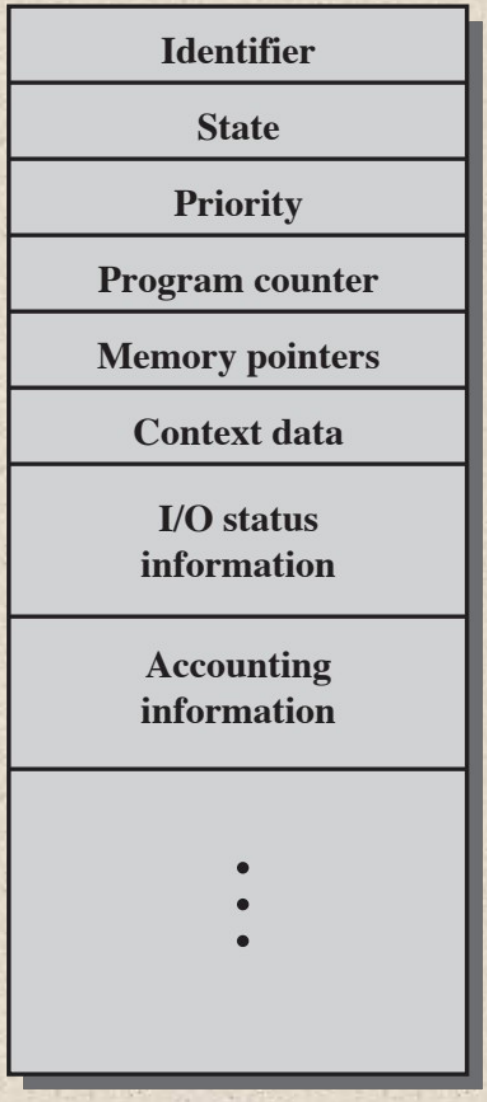


Figure 8.8 Process Control Block

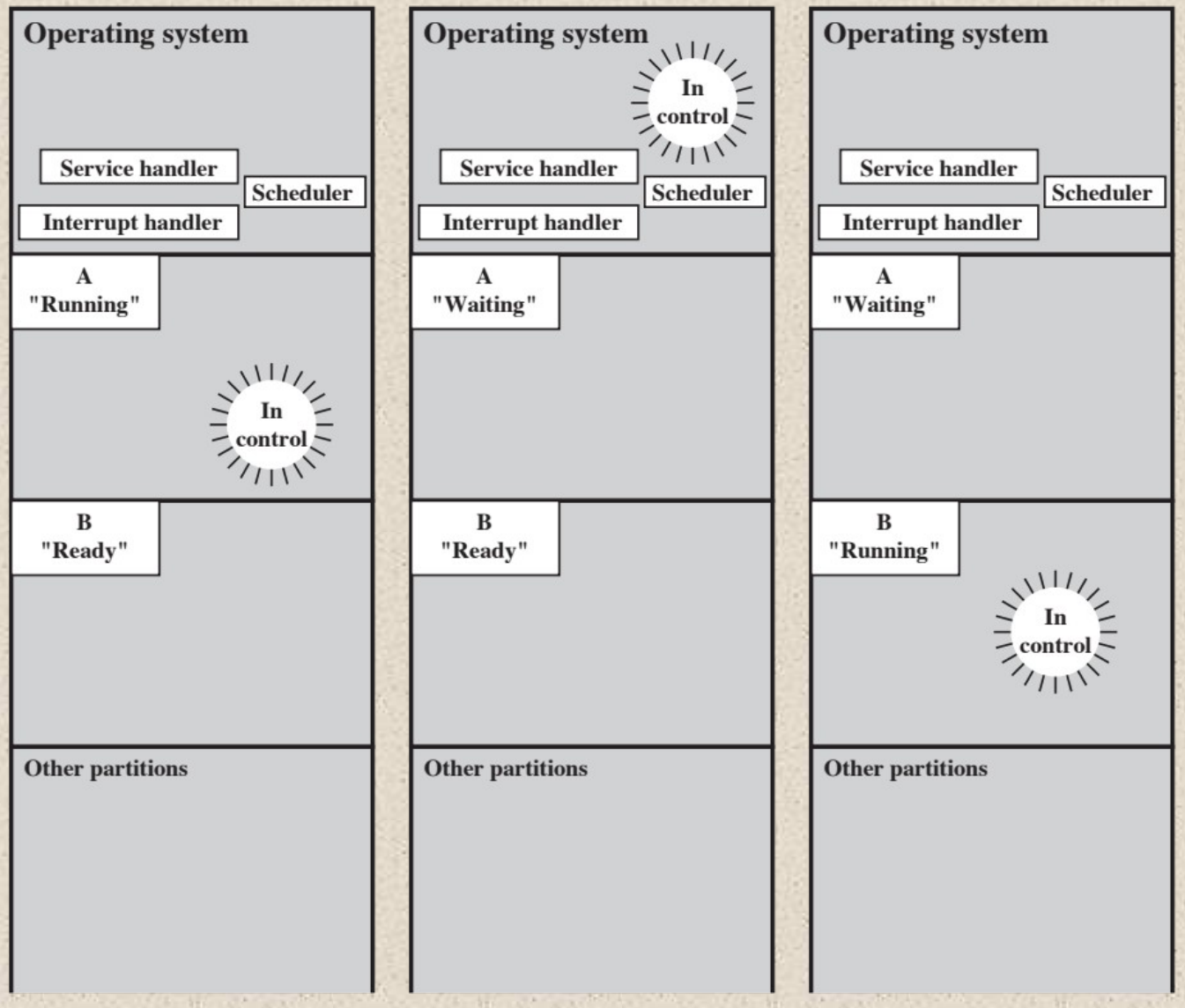


Figure 8.9 Scheduling Example

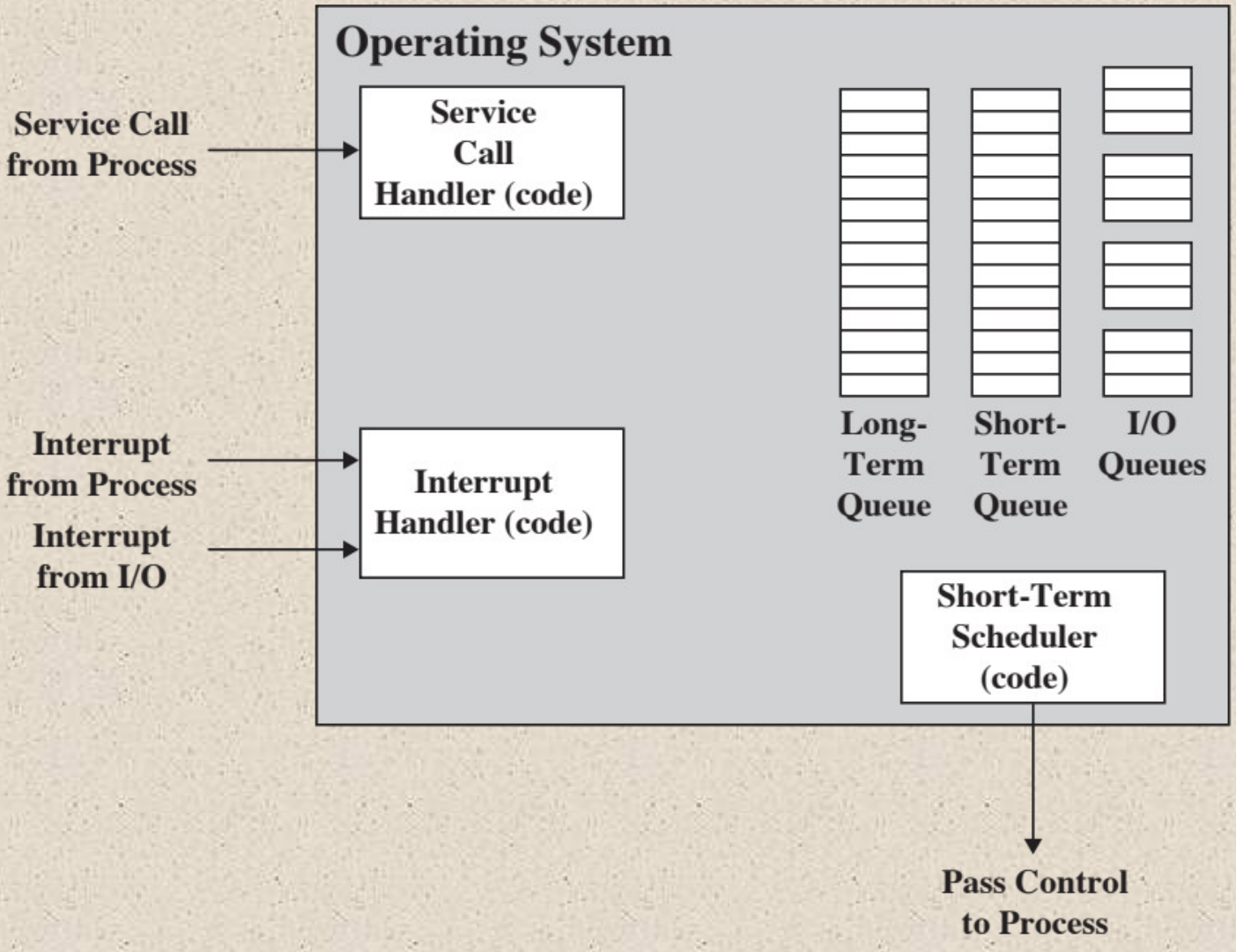


Figure 8.10 Key Elements of an Operating System for Multiprogramming

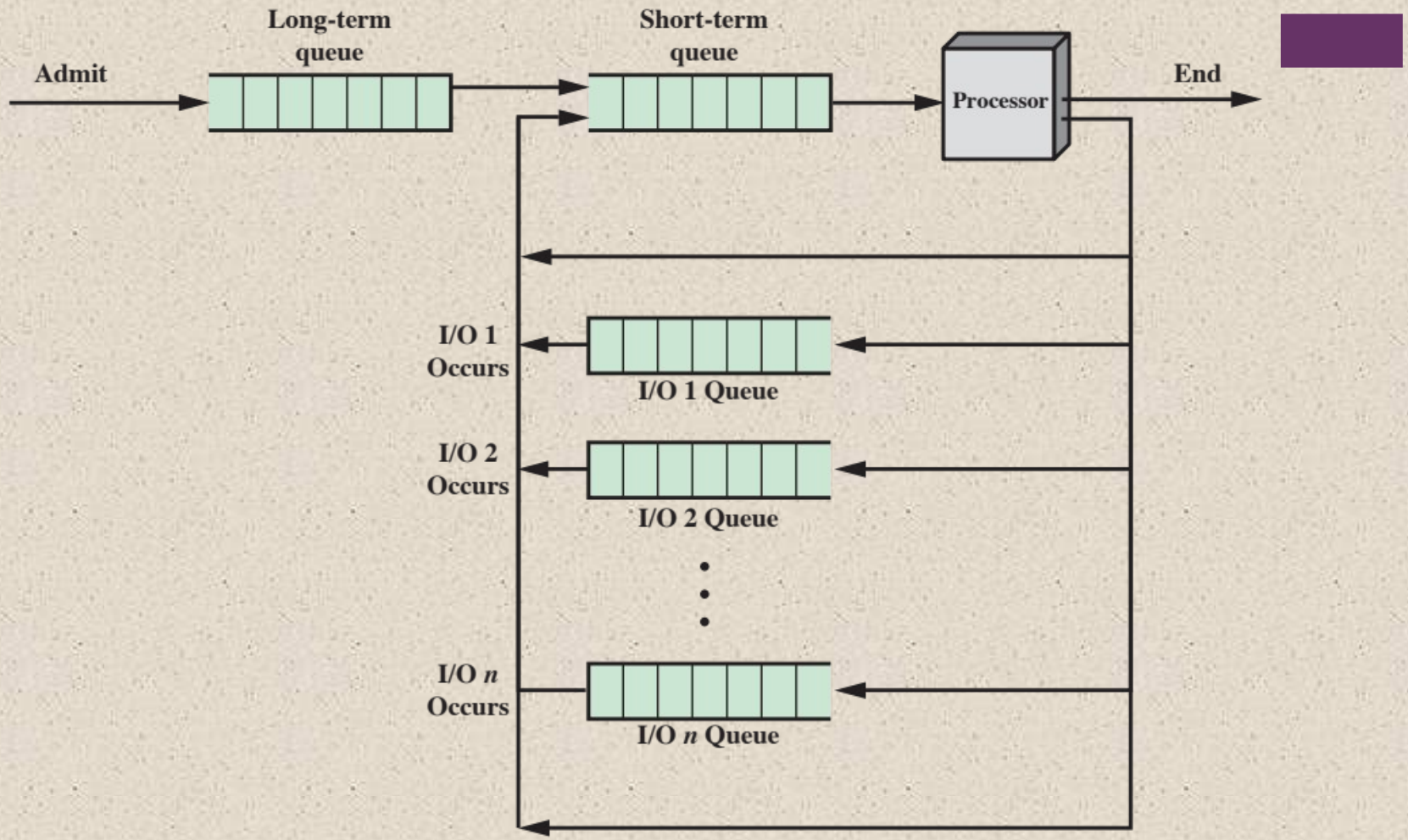


Figure 8.11 Queuing Diagram Representation of Processor Scheduling

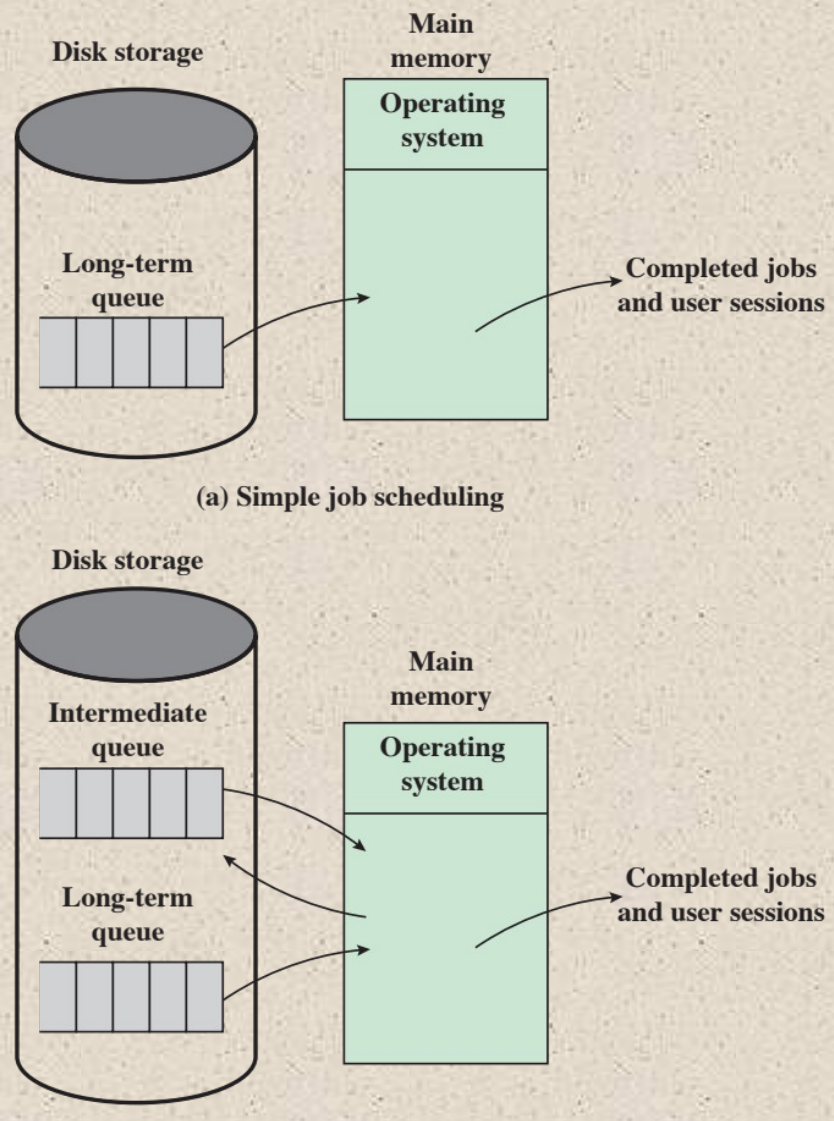


Figure 8.12 The Use of Swapping

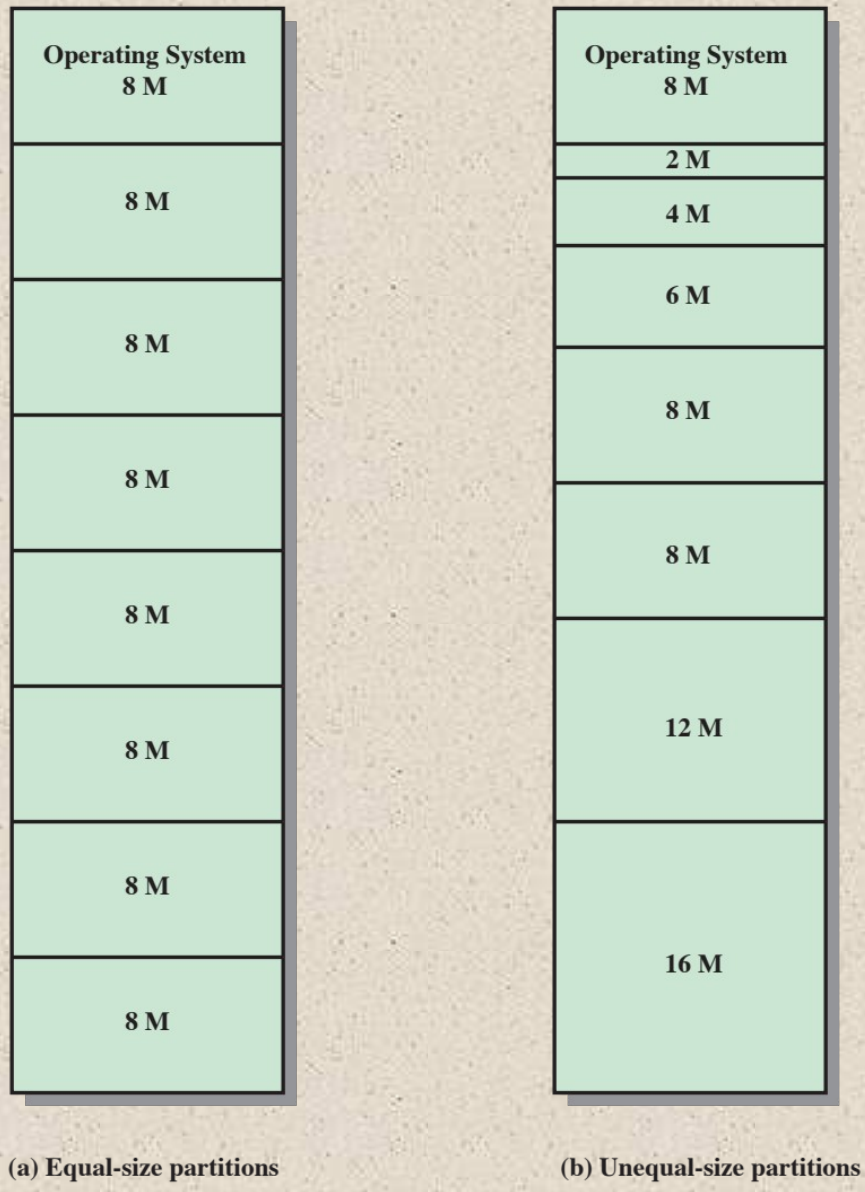


Figure 8.13 Example of Fixed Partitioning of a 64-Mbyte Memory

Logical address
- expressed as a location relative to the beginning of the program

Physical address
- an actual location in main memory

Base address
- current starting location of the process

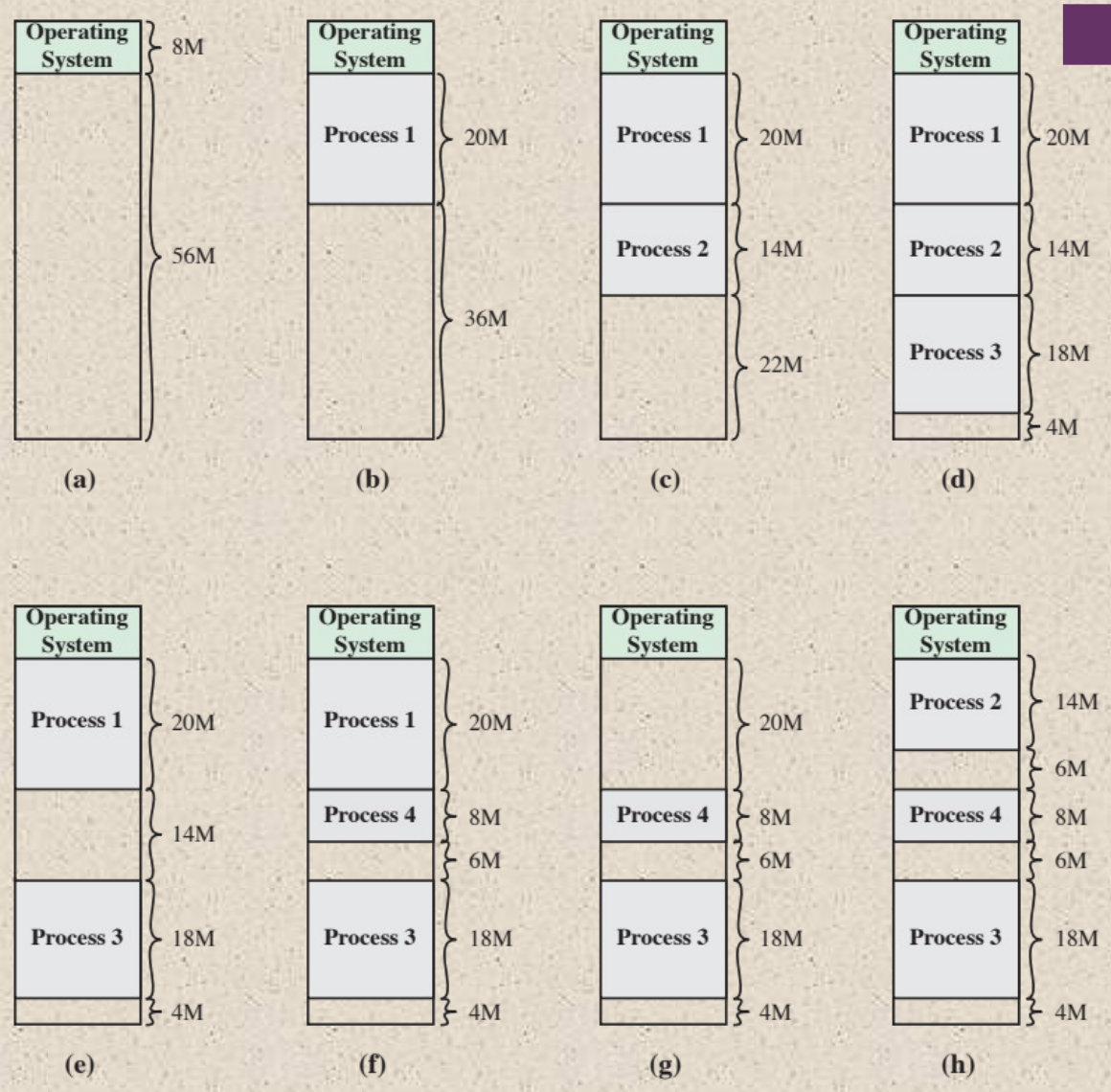


Figure 8.14 The Effect of Dynamic Partitioning

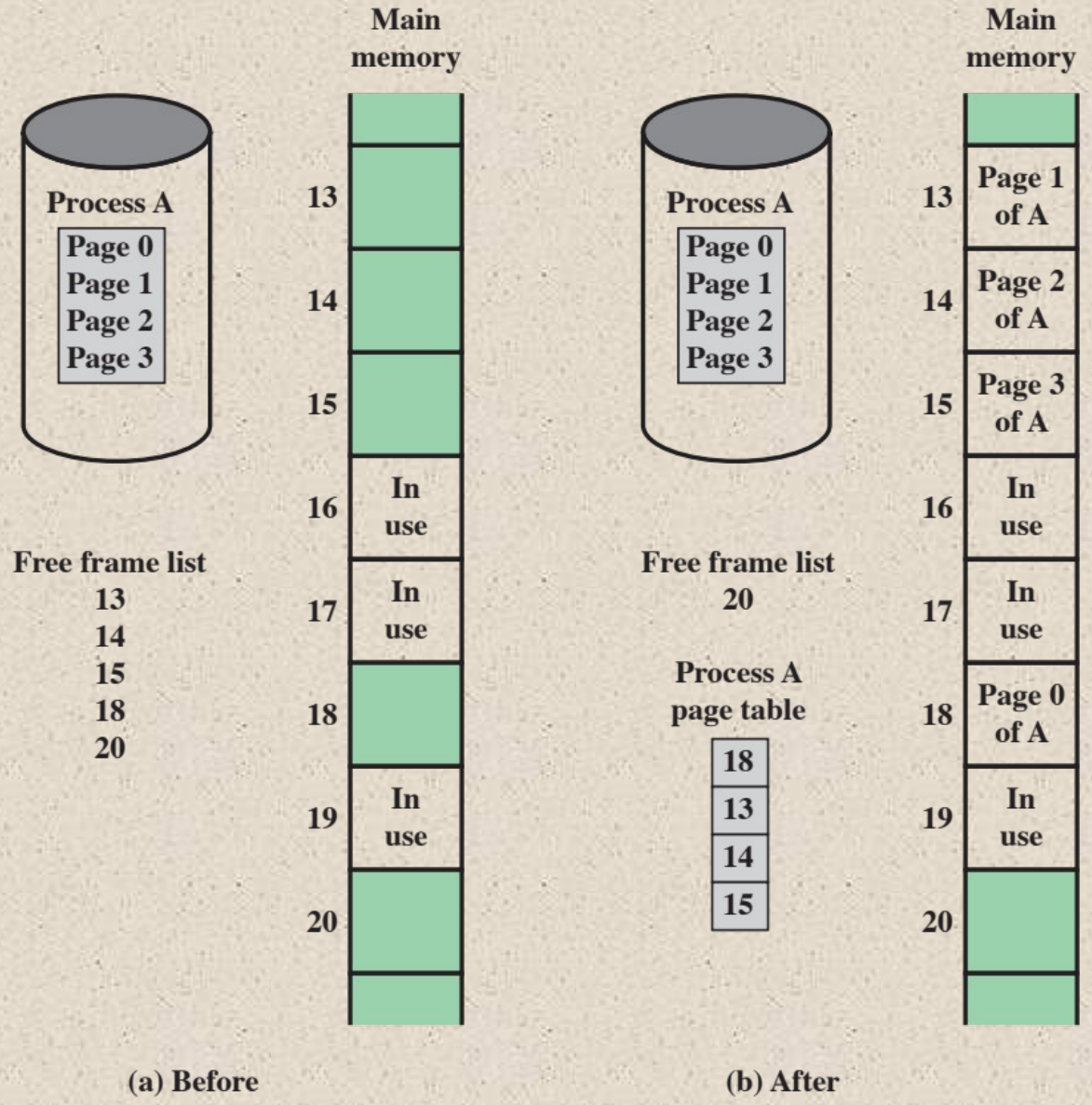


Figure 8.15 Allocation of Free Frames

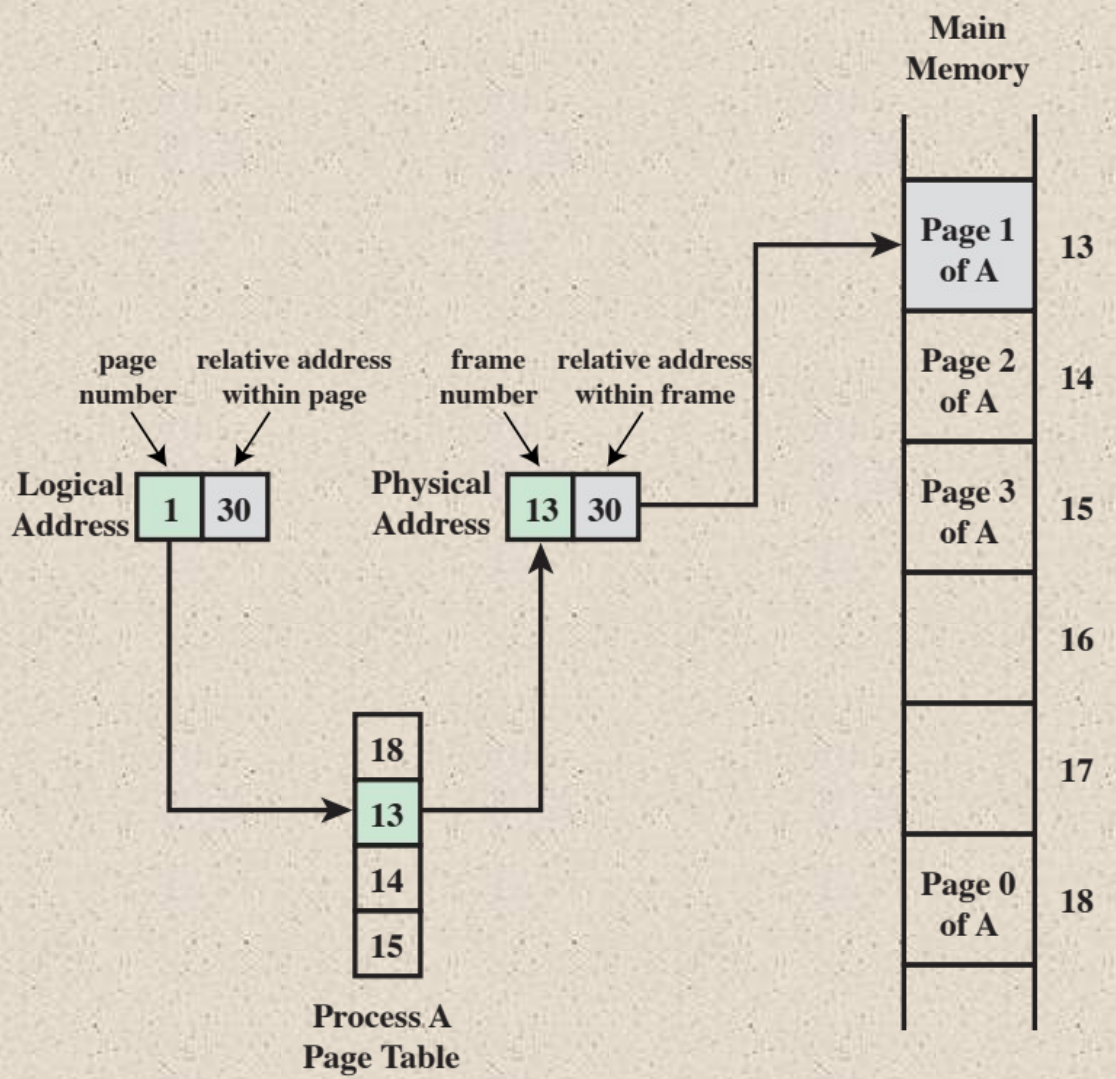


Figure 8.16 Logical and Physical Addresses

Virtual Memory

Demand Paging

- Each page of a process is brought in only when it is needed
- Principle of locality
  - When working with a large process execution may be confined to a small section of a program (subroutine)
  - It is better use of memory to load in just a few pages
  - If the program references data or branches to an instruction on a page not in main memory, a *page fault* is triggered which tells the OS to bring in the desired page
- Advantages:
  - More processes can be maintained in memory
  - Time is saved because unused pages are not swapped in and out of memory
- Disadvantages:
  - When one page is brought in, another page must be thrown out (*page replacement)*
  - If a page is thrown out just before it is about to be used the OS will have to go get the page again
  - *Thrashing*
    - When the processor spends most of its time swapping pages rather than executing instructions

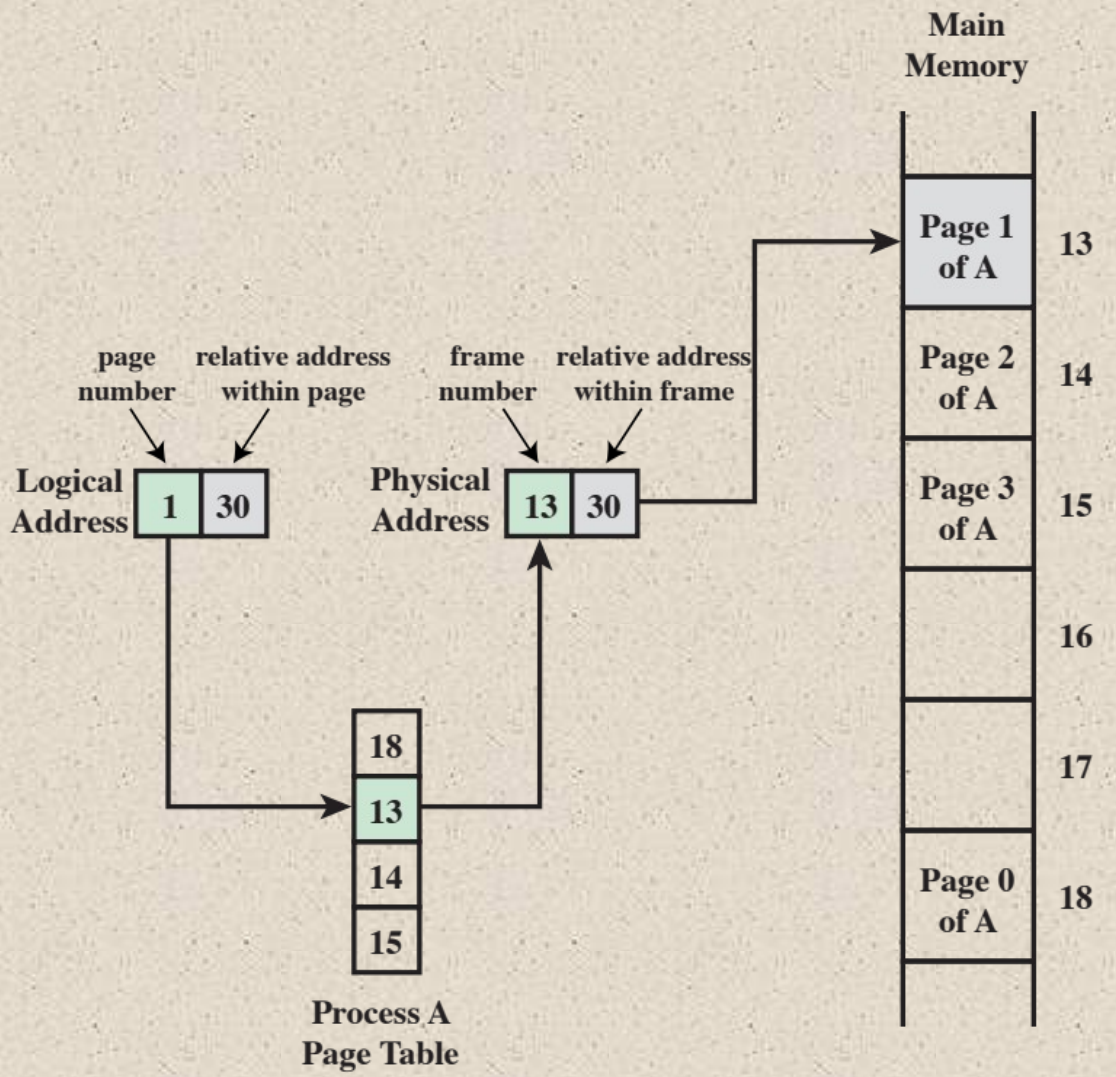


Figure 8.16 Logical and Physical Addresses

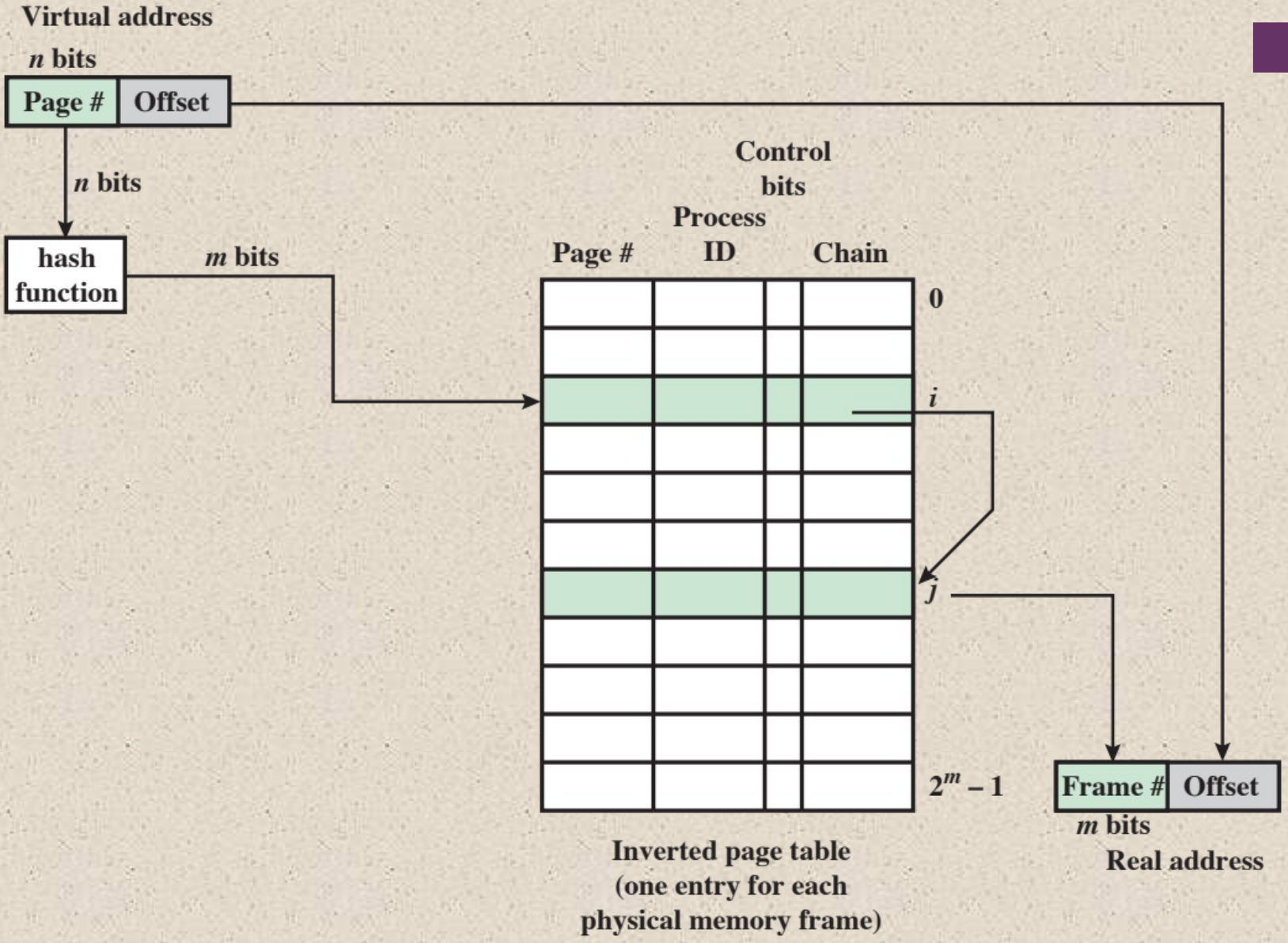


Figure 8.17 Inverted Page Table Structure

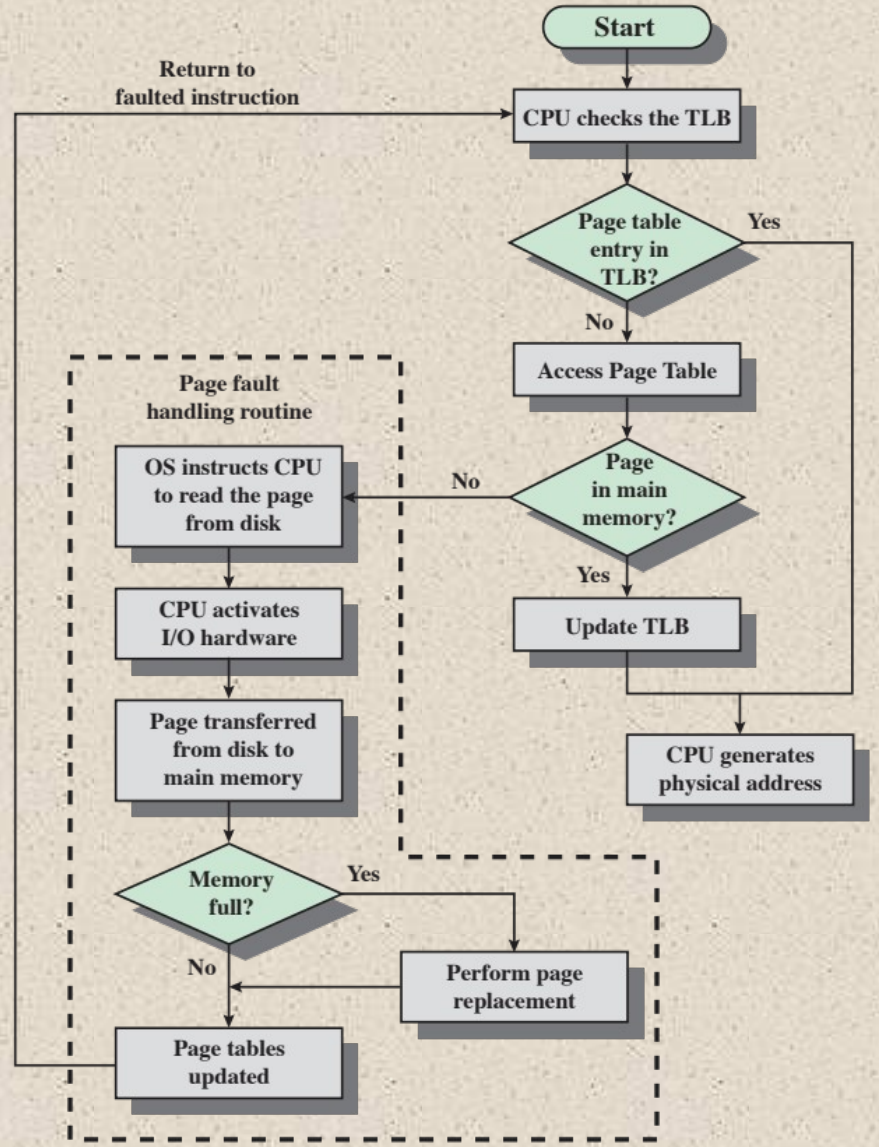


Figure 8.18 Operation of Paging and Translation Lookaside Buffer (TLB) [FURH87]

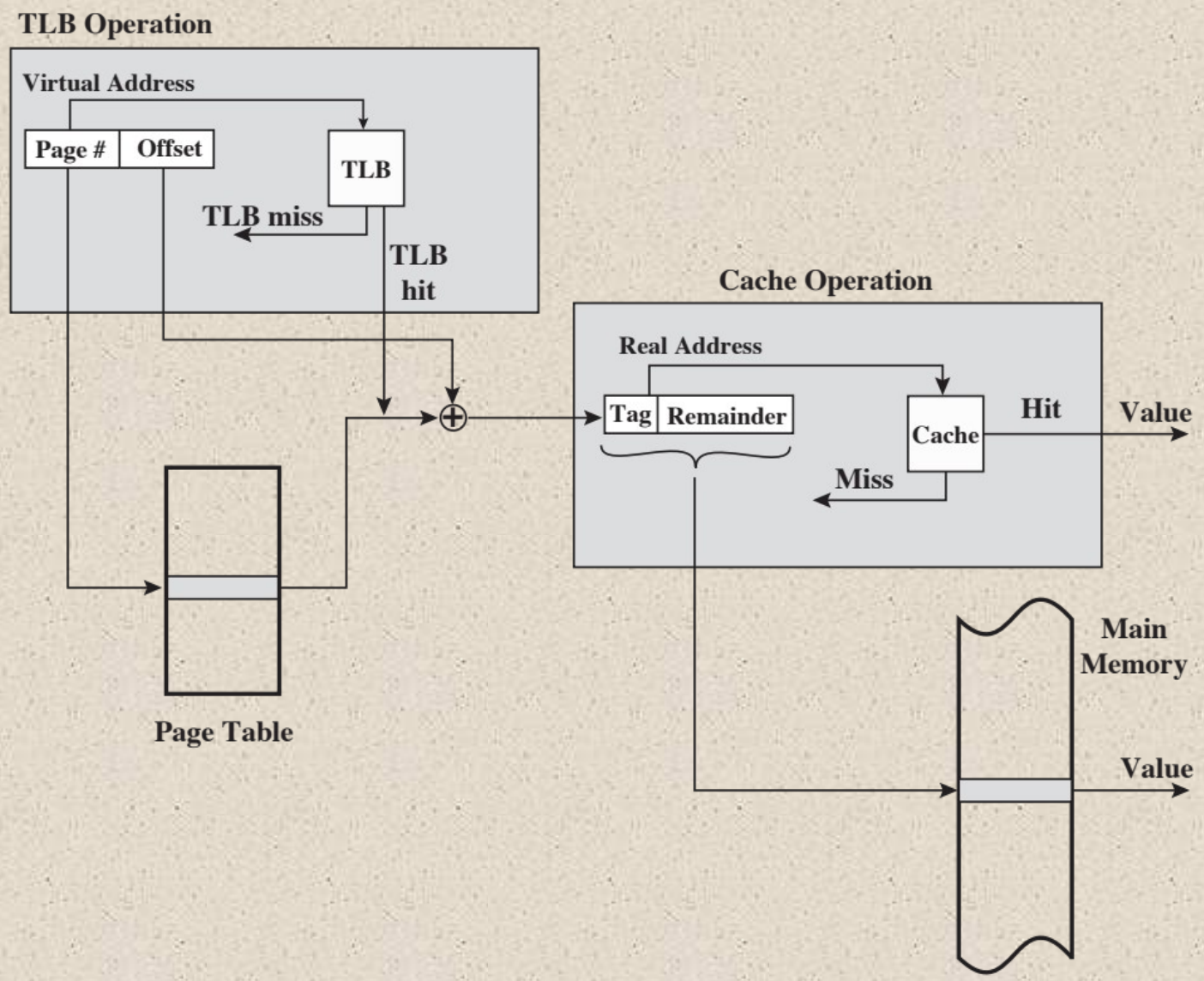


Figure 8.19 Translation Lookaside Buffer and Cache Operation

Segmentation

- Usually visible to the programmer
- Provided as a convenience for organizing programs and data and as a means for associating privilege and protection attributes with instructions and data
- Allows the programmer to view memory as consisting of multiple address spaces or segments

- Advantages:
  - Simplifies the handling of growing data structures
  - Allows programs to be altered and recompiled independently without requiring that an entire set of programs be re-linked and re-loaded
  - Lends itself to sharing among processes
  - Lends itself to protection

- Includes hardware for both segmentation and paging
- Unsegmented unpaged memory
  - Virtual address is the same as the physical address
  - Useful in low-complexity, high performance controller applications
- Unsegmented paged memory
  - Memory is viewed as a paged linear address space
  - Protection and management of memory is done via paging
  - Favored by some operating systems
- Segmented unpaged memory
  - Memory is viewed as a collection of logical address spaces
  - Affords protection down to the level of a single byte
  - Guarantees that the translation table needed is on-chip when the segment is in memory
  - Results in predictable access times
- Segmented paged memory
  - Segmentation is used to define logical memory partitions subject to access control, and paging is used to manage the allocation of memory within the partitions
  - Operating systems such as UNIX System V favor this view

Segmentation

- Each virtual address consists of a 16-bit segment reference and a 32-bit offset
  - Two bits of segment reference deal with the protection mechanism
  - 14 bits specify segment
- Unsegmented virtual memory is 2^{32} = 4Gbytes
- Segmented virtual memory is 2^{46}=64 terabytes (Tbytes)
- Physical address space employs a 32-bit address for a maximum of 4 Gbytes
- Virtual address space is divided into two parts
  - One-half is global, shared by all processors
  - The remainder is local and is distinct for each process

Segment Protection

- Associated with each segment are two forms of protection:
  - Privilege level
  - Access attribute
- There are four privilege levels
  - Most protected (level 0)
  - Least protected (level 3)
- Privilege level associated with a data segment is its “classification”
- Privilege level associated with a program segment is its “clearance”
- An executing program may only access data segments for which its clearance level is lower than or equal to the privilege level of the data segment
- The privilege mechanism also limits the use of certain instructions

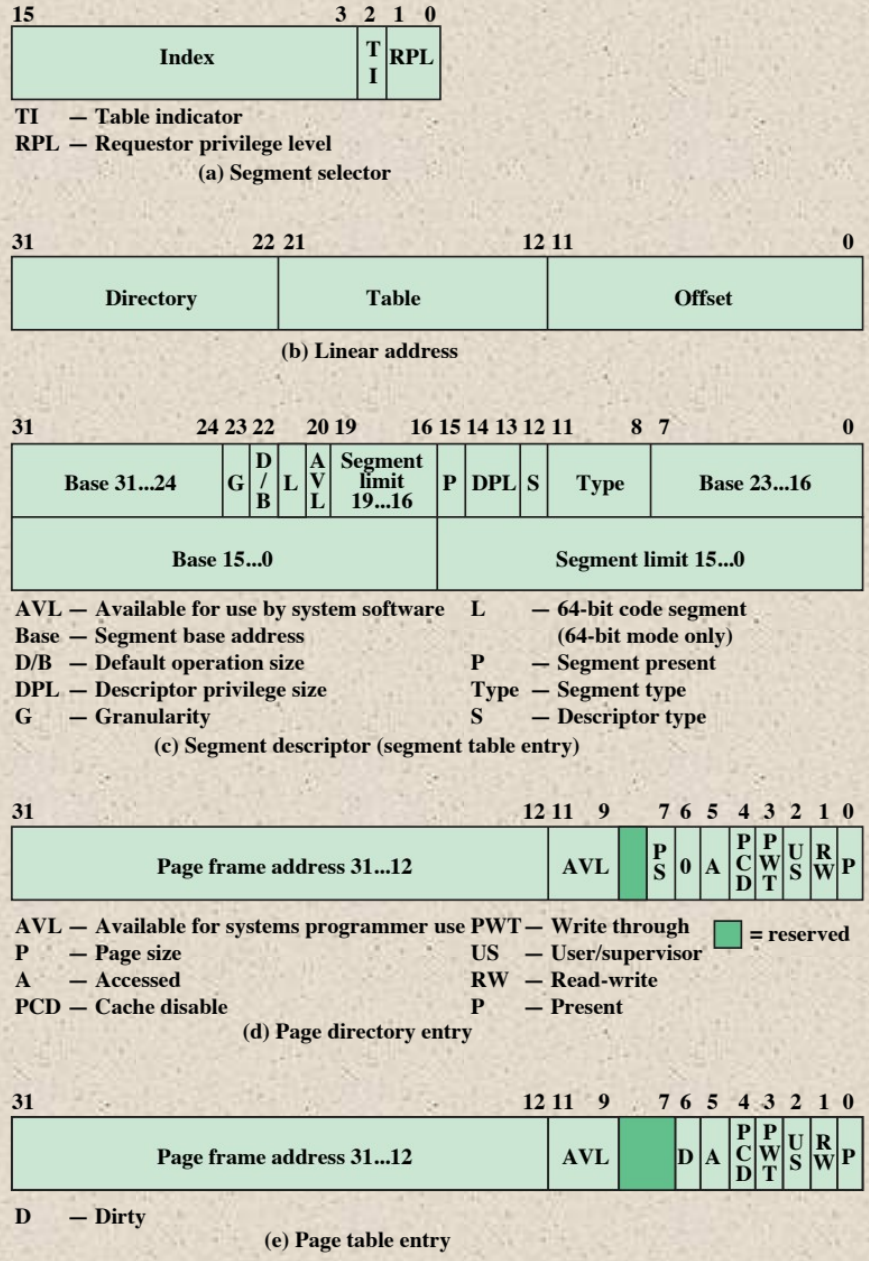


Figure 8.20 Intel x86 Memory Management Formats

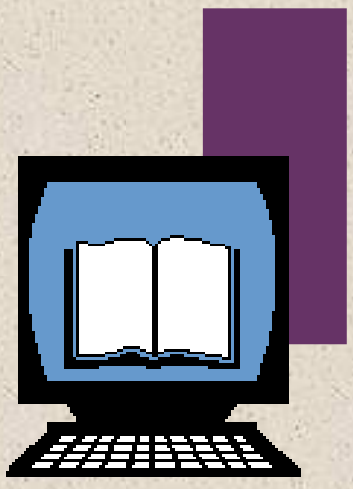
Table 8.5
x86 Memory Management Parameters (page 1 of 2)

Segment Descriptor (Segment Table Entry)
Base Defines the starting address of the segment within the 4-GByte linear address space.
D/B bit In a code segment, this is the D bit and indicates whether operands and addressing modes are 16 or 32 bits.
Descriptor Privilege Level (DPL) Specifies the privilege level of the segment referred to by this segment descriptor.
Granularity bit (G) Indicates whether the Limit field is to be interpreted in units by one byte or 4 KBytes.
Limit Defines the size of the segment. The processor interprets the limit field in one of two ways, depending on the granularity bit: in units of one byte, up to a segment size limit of 1 MByte, or in units of 4 KBytes, up to a segment size limit of 4 GBytes.
S bit Determines whether a given segment is a system segment or a code or data segment.
Segment Present bit (P) Used for nonpaged systems. It indicates whether the segment is present in main memory. For paged systems, this bit is always set to 1.
Type Distinguishes between various kinds of segments and indicates the access attributes.

Table 8.5
x86 Memory Management Parameters (page 2 of 2)

Page Directory Entry and Page Table Entry
Accessed bit (A)
This bit is set to 1 by the processor in both levels of page tables when a read or write operation to the corresponding page occurs.
Dirty bit (D)
This bit is set to 1 by the processor when a write operation to the corresponding page occurs.
Page Frame Address
Provides the physical address of the page in memory if the present bit is set. Since page frames are aligned on 4K boundaries, the bottom 12 bits are 0, and only the top 20 bits are included in the entry. In a page directory, the address is that of a page table.
Page Cache Disable bit (PCD)
Indicates whether data from page may be cached.
Page Size bit (PS)
Indicates whether page size is 4 KByte or 4 MByte.
Page Write Through bit (PWT)
Indicates whether write-through or write-back caching policy will be used for data in the corresponding page.
Present bit (P)
Indicates whether the page table or page is in main memory.
Read/Write bit (RW)
For user-level pages, indicates whether the page is read-only access or read/write access for user-level programs.
User/Supervisor bit (US)
Indicates whether the page is available only to the operating system (supervisor level) or is available to both operating system and applications (user level).

Paging



- Segmentation may be disabled
  - In which case linear address space is used
- Two level page table lookup
  - First, page directory
    - 1024 entries max
    - Splits 4 Gbyte linear memory into 1024 page groups of 4 Mbyte
    - Each page table has 1024 entries corresponding to 4 Kbyte pages
    - Can use one page directory for all processes, one per process or mixture
    - Page directory for current process always in memory
  - Use TLB holding 32 page table entries
  - Two page sizes available, 4k or 4M

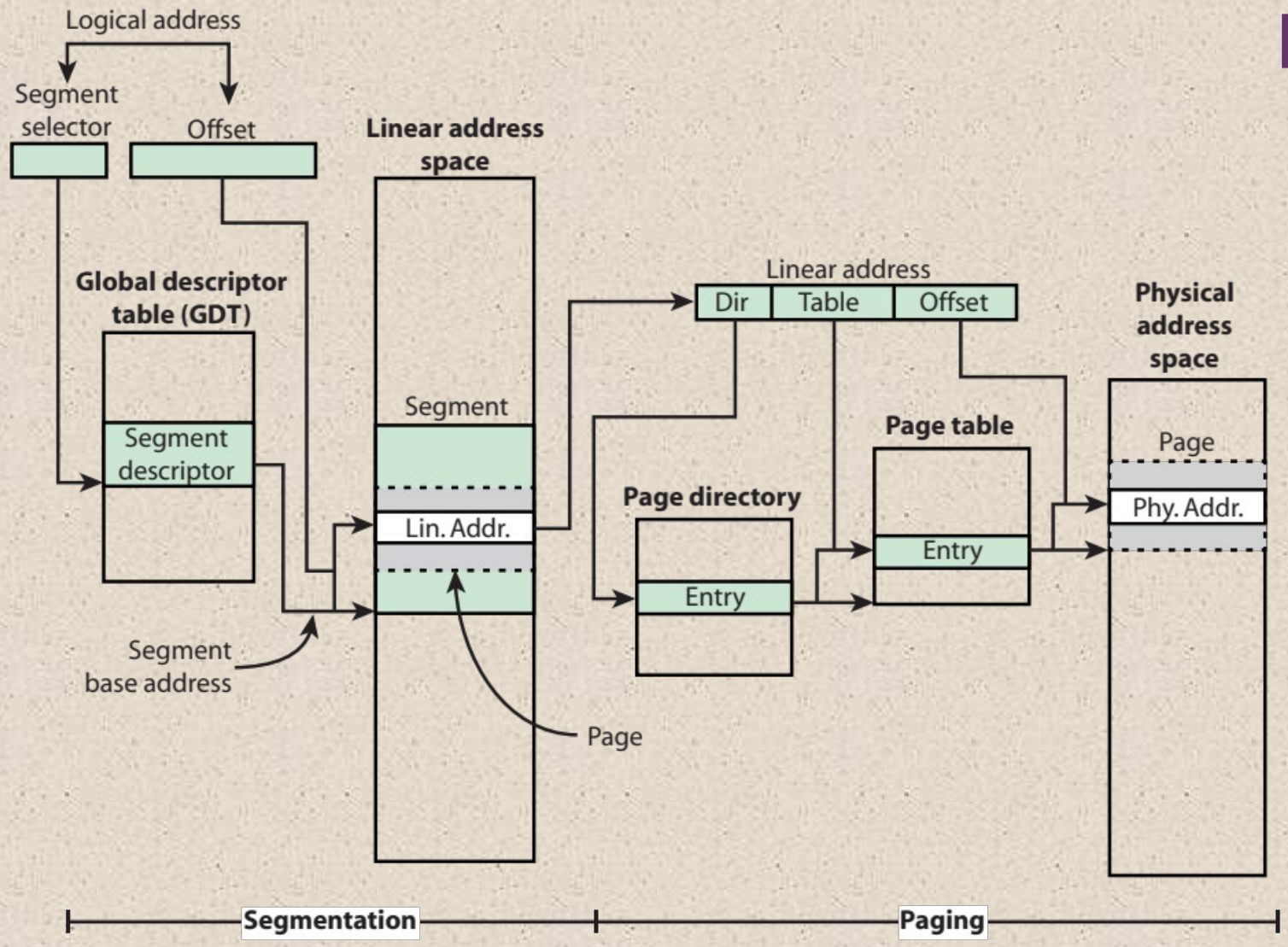


Figure 8.21 Intel x86 Memory Address Translation Mechanisms

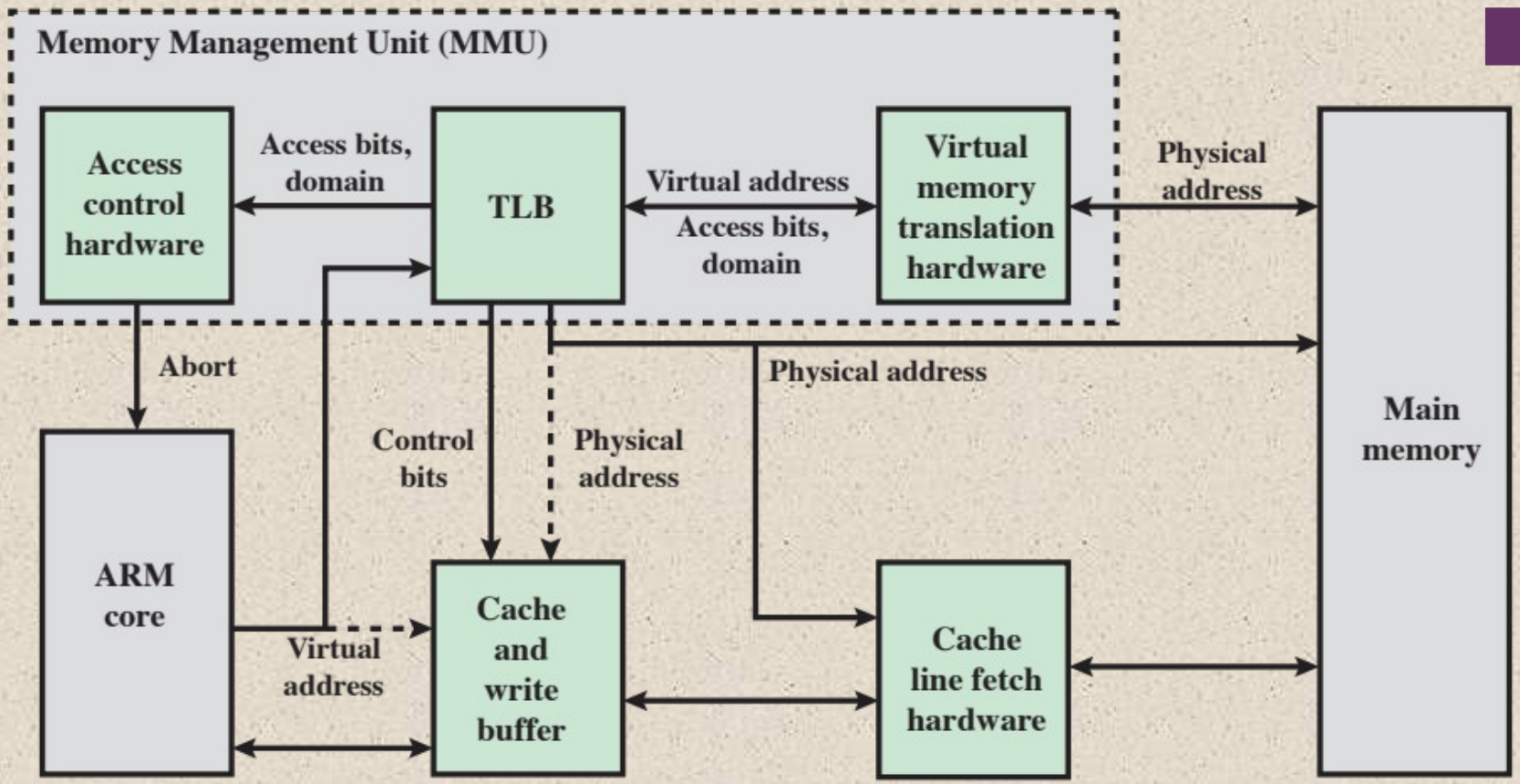


Figure 8.22 ARM Memory System Overview

Virtual Memory Address Translation

- The ARM supports memory access based on either sections or pages
- Supersections (optional)
  - Consist of 16-MB blocks of main memory
- Sections
  - Consist of 1-MB blocks of main memory
- Large pages
  - Consist of 64-kB blocks of main memory
- Small pages
  - Consist of 4-kB blocks of main memory
- Sections and supersections are supported to allow mapping of a large region of memory while using only a single entry in the TLB
- The translation table held in main memory has two levels:
  - First-level table
    - Holds section and supersection translations, and pointers to second-level table
  - Second-level tables
    - Hold both large and small page translations

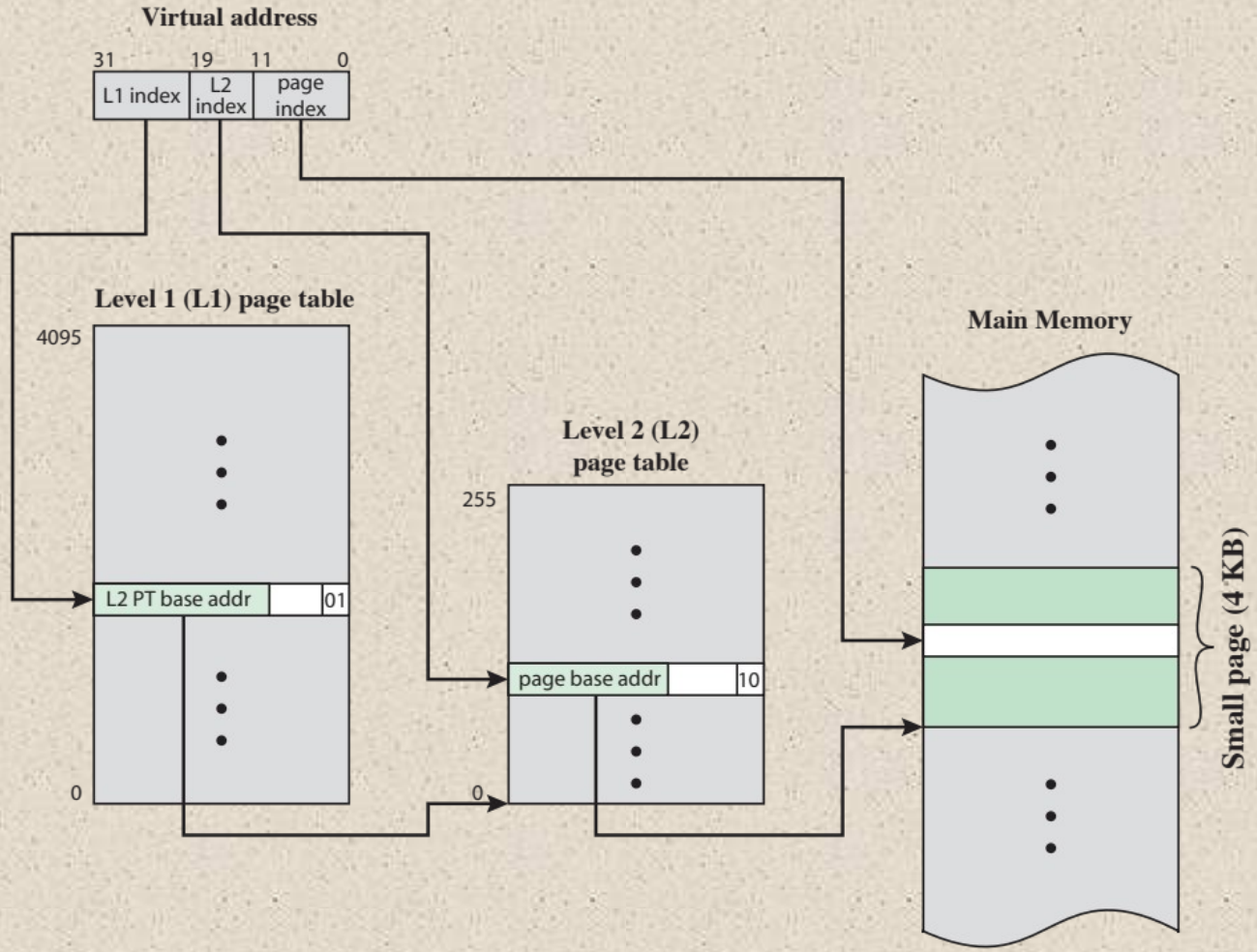
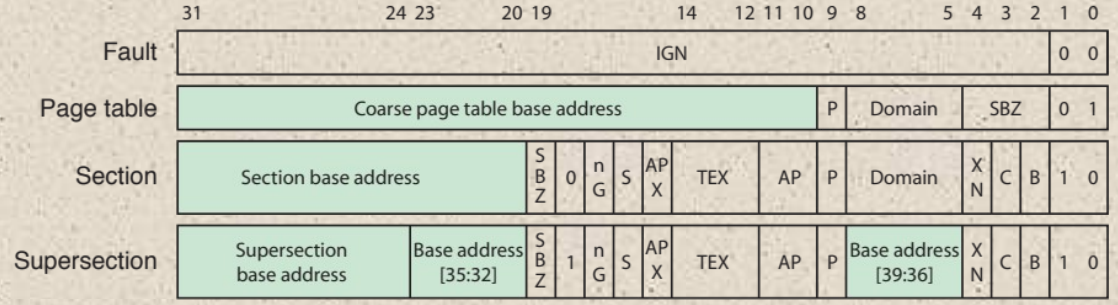
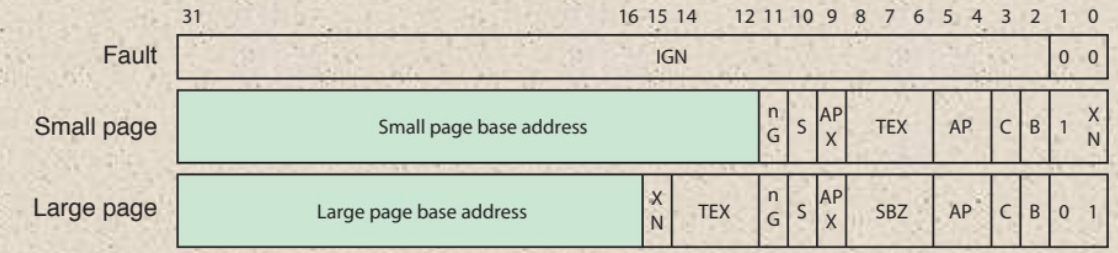


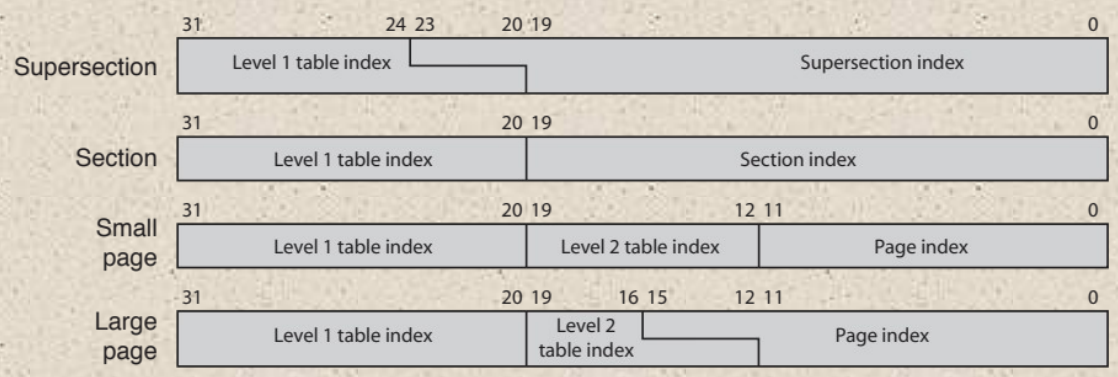
Figure 8.23 ARM Virtual Memory Address Translation for Small Pages



(a) Alternative first-level descriptor formats



(b) Alternative second-level descriptor formats



(c) Virtual memory address formats

Figure 8.24 ARM Memory Management Formats

Table 8.6 ARM Memory-Management Parameters

Access Permission (AP), Access Permission Extension (APX)
These bits control access to the corresponding memory region. If an access is made to an area of memory without the required permissions, a Permission Fault is raised.

Bufferable (B) bit
Determines, with the TEX bits, how the write buffer is used for cacheable memory.

Cacheable (C) bit
Determines whether this memory region can be mapped through the cache.

Domain
Collection of memory regions. Access control can be applied on the basis of domain.

not Global (nG)
Determines whether the translation should be marked as global (0), or process specific (1).

Shared (S)
Determines whether the translation is for not-shared (0), or shared (1) memory.

SBZ
Should be zero.

Type Extension (TEX)
These bits, together with the B and C bits, control accesses to the caches, how the write buffer is used, and if the memory region is shareable and therefore must be kept coherent.

Execute Never (XN)
Determines whether the region is executable (0) or not executable (1).

Access Control

- The AP access control bits in each table entry control access to a region of memory by a given process
- A region of memory can be designated as:
  - No access
  - Read only
  - Read-write
- The region can be privileged access only, reserved for use by the OS and not by applications
- ARM employs the concept of a domain:
  - Collection of sections and/or pages that have particular access permissions
  - The ARM architecture supports 16 domains
  - Allows multiple processes to use the same translation tables while maintaining some protection from each other
- Two kinds of domain access are supported:
  - Clients
    - Users of domains that must observe the access permissions of the individual sections and/or pages that make up that domain
  - Managers
    - Control the behavior of the domain and bypass the access permissions for table entries in that domain

Summary

Chapter 8

Operating System Support

- Operating system overview
  - Operating system objectives and functions
  - Types of operating systems
- Scheduling
  - Long-term scheduling
  - Medium-term scheduling
  - Short-term scheduling
- Intel x86 memory management
  - Address space
  - Segmentation
  - paging
- Memory management
  - Swapping
  - Partitioning
  - Paging
  - Virtual memory
  - Translation lookaside buffer
  - Segmentation
- ARM memory management
  - Memory system organization
  - Virtual memory address translation
  - Memory-management formats
  - Access control

William Stallings
Computer Organization and Architecture
10th Edition

Chapter 9

Number Systems

The Decimal System

- System based on decimal digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) to represent numbers
- For example the number 83 means eight tens plus three:

$$83 = (8 * 10) + 3$$

- The number 4728 means four thousands, seven hundreds, two tens, plus eight:

$$4728 = (4 * 1000) + (7 * 100) + (2 * 10) + 8$$

- The decimal system is said to have a ***base***, or ***radix***, of 10. This means that each digit in the number is multiplied by 10 raised to a power corresponding to that digit's position:

$$83 = (8 * 10^1) + (3 * 10^0)$$

$$4728 = (4 * 10^3) + (7 * 10^2) + (2 * 10^1) + (8 * 10^0)$$

Decimal Fractions

- The same principle holds for decimal fractions, but negative powers of 10 are used. Thus, the decimal fraction 0.256 stands for 2 tenths plus 5 hundredths plus 6 thousandths:

$$0.256 = (2 * 10^{-1}) + (5 * 10^{-2}) + (6 * 10^{-3})$$

- A number with both an integer and fractional part has digits raised to both positive and negative powers of 10:

$$442.256 = (4 * 10^{2}) + (4 + 10^{1}) + (2 * 10^{0}) + (2 * 10^{-1}) + (5 * 10^{-2}) + (6 * 10^{-3})$$

- ***Most significant digit***
  - The leftmost digit (carries the highest value)
- ***Least significant digit***
  - The rightmost digit

Table 9.1
Positional Interpretation of a Decimal Number

4	7	2	2	5	6
100s	10s	1s	tenths	hundredths	thousandths
10^2	10^1	10^0	10^{-1}	10^{-2}	10^{-3}
position 2	position 1	position 0	position −1	position −2	position −3

Positional Number Systems

- Each number is represented by a string of digits in which each digit position *i* has an associated weight r^i, where *r* is the *radix*, or *base,* of the number system.
- The general form of a number in such a system with radix *r* is

$$(\ldots a_3 a_2 a_1 a_0 . a_{-1} a_{-2} a_{-3} \ldots)_r$$

where the value of any digit a_i *is an integer in the range* $0 \leq a_i < r$. The dot between a_0 *and* a_{-1} is called the ***radix point.***

Table 9.2
Positional Interpretation of a Number in Base 7

Position	4	3	2	1	0	−1
Value in exponential form	7^4	7^3	7^2	7^1	7^0	7^{-1}
Decimal value	2401	343	49	7	1	1/7

The Binary System

- Only two digits, 1 and 0
- Represented to the base 2
- The digits 1 and 0 in binary notation have the same meaning as in decimal notation:

$$0_2 = 0_{10}$$

$$1_2 = 1_{10}$$

- To represent larger numbers each digit in a binary number has a value depending on its position:

$$10_2 = (1 * 2^1) + (0 * 2^0) = 2_{10}$$

$$11_2 = (1 * 2^1) + (1 * 2^0) = 3_{10}$$

$$100_2 = (1 * 2^2) + (0 * 2^1) + (0 * 2^0) = 4_{10}$$

and so on. Again, fractional values are represented with negative powers of the radix:

$$1001.101 = 2^3 + 2^0 + 2^{-1} + 2^{-3} = 9.625_{10}$$

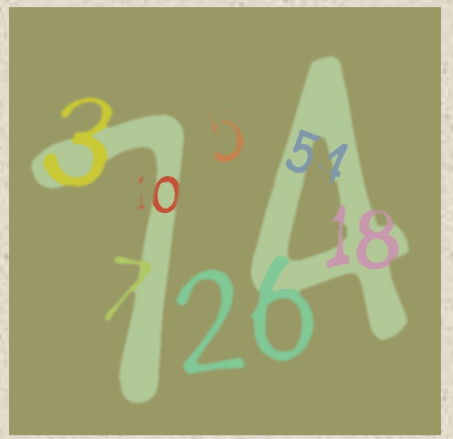
Converting Between Binary and Decimal

Binary notation to decimal notation:

- Multiply each binary digit by the appropriate power of 2 and add the results

Decimal notation to binary notation:

- Integer and fractional parts are handled separately



Integers

For the integer part, recall that in binary notation, an integer represented by

$$b_{m-1}b_{m-2} \ldots b_2b_1b_0 \qquad b_i = 0 \text{ or } 1$$

has the value

$$(b_{m-1} * 2^{m-1}) + (b_{m-2} * 2^{m-2}) + \ldots + (b_1 * 2^1) + b_0$$

Suppose it is required to convert a decimal integer N into binary form. If we divide N by 2, in the decimal system, and obtain a quotient N_1 and a remainder R_0, we may write

$$N = 2 * N_1 + R_0 \qquad R_0 = 0 \text{ or } 1$$

Next, we divide the quotient N_1 by 2. Assume that the new quotient is N_2 and the new remainder R_1. Then

$$N_1 = 2 * N_2 + R_1 \qquad R_1 = 0 \text{ or } 1$$

so that

$$N = 2(2N_2 + R_1) + R_0 = (N_2 * 2^2) + (R_1 * 2^1) + R_0$$

If next

$$N_2 = 2N_3 + R_2$$

we have

$$N = (N_3 * 2^3) + (R_2 * 2^2) + (R_1 * 2^1) + R_0$$

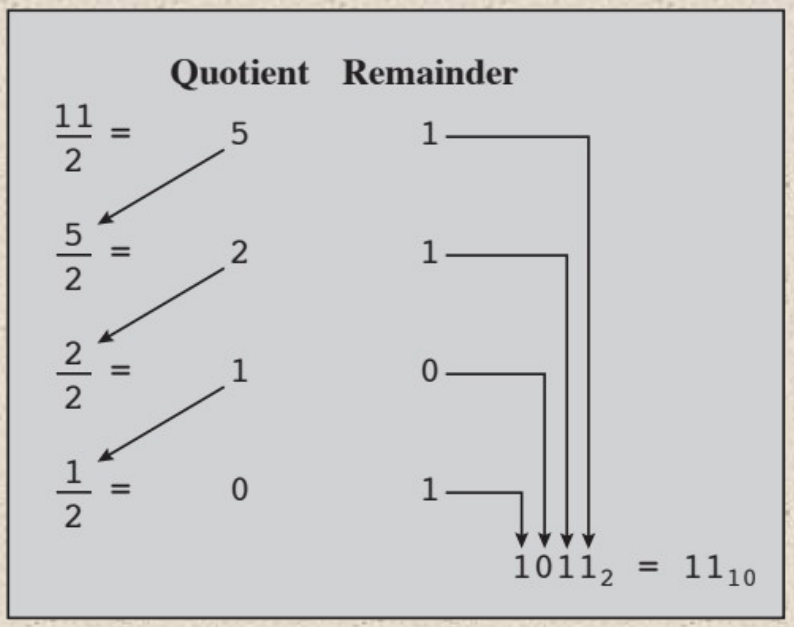
Continued . . .

Integers

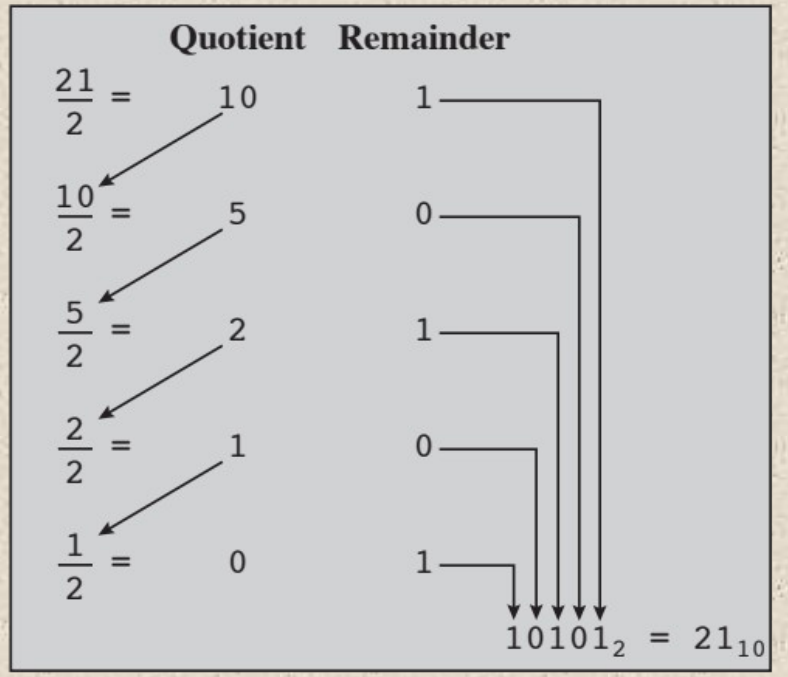
Because $N > N_1 > N_{2\ldots}$, continuing this sequence will eventually produce a quotient $N_{m-1} = 1$ (except for the decimal integers 0 and 1, whose binary equivalents are 0 and 1, respectively) and a remainder R_{m-2}, which is 0 or 1. Then

$$N = (1 * 2^{m-1}) + (R_{m-2} * 2^{m-2}) + \ldots + (R_2 * 2^2) + (R_1 * 2^1) + R_0$$

which is the binary form of *N*. Hence, we convert from base 10 to base 2 by repeated divisions by 2. The remainders and the final quotient, 1, give us, in order of increasing significance, the binary digits of *N*.



(a) 11_{10}



(b) 21_{10}

Figure 9.1 Examples of Converting from Decimal Notation to Binary Notation for Integers

Fractions

For the fractional part, recall that in binary notation, a number with a value between 0 and 1 is represented by

$$0.b_{-1}b_{-2}b_{-3}\ldots \qquad b_i = 0 \text{ or } 1$$

and has the value

$$(b_{-1} * 2^{-1}) + (b_{-2} * 2^{-2}) + (b_{-3} * 2^{-3}) \ldots$$

This can be rewritten as

$$2^{-1} * (b_{-1} + 2^{-1} * (b_{-2} + 2^{-1} * (b_{-3} + \ldots) \ldots))$$

Suppose we want to convert the number $F\ (0 < F < 1)$ from decimal to binary notation. We know that F can be expressed in the form

$$F = 2^{-1} * (b_{-1} + 2^{-1} * (b_{-2} + 2^{-1} * (b_{-3} + \ldots) \ldots))$$

If we multiply *F* by 2, we obtain,

$$2 * F = b_{-1} + 2^{-1} * (b_{-2} + 2^{-1} * (b_{-3} + \ldots) \ldots)$$

Continued . . .

Fractions

From this equation, we see that the integer part of (2 * *F), which must be* either 0 or 1 because $0 < F < 1$, *is simply* b_{-1}. *So we can say* $(2 * F) = b_{-1} + F_1$, where $0 < F_1 < 1$ *and where*

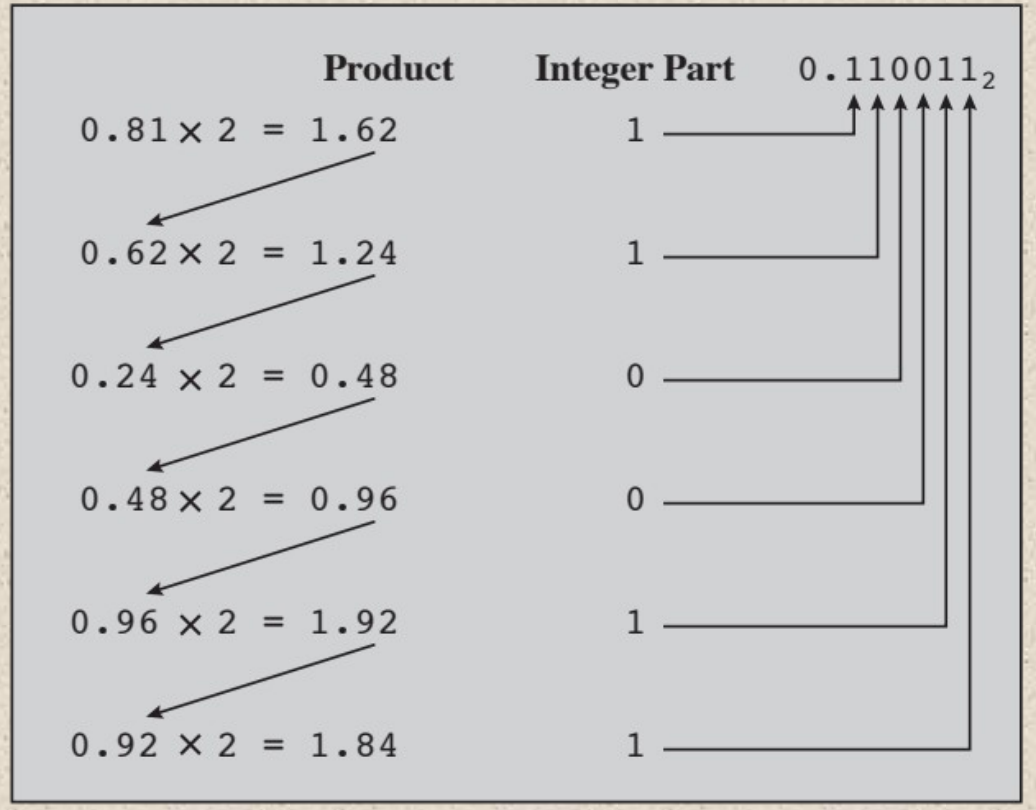
$$F_1 = 2\text{-}1 * (b_{-2} + 2^{-1} * (b_{-3} + 2^{-1} * (b_{-4} + \ldots) \ldots))$$

To find b_{-2}, we repeat the process.

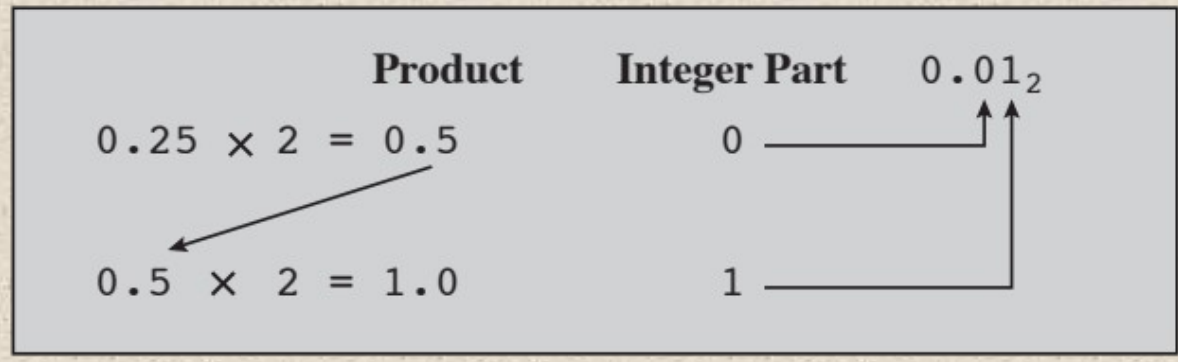
At each step, the fractional part of the number from the previous step is multiplied by 2. The digit to the left of the decimal point in the product will be 0 or 1 and contributes to the binary representation, starting with the most significant digit. The fractional part of the product is used as the multiplicand in the next step.

Figure 9.2

Examples of Converting from Decimal Notation To Binary Notation For Fractions



(a) $0.81_{10} = 0.110011_2$ (approximately)



(b) $0.25_{10} = 0.01_2$ (exactly)

Hexadecimal Notation

- Binary digits are grouped into sets of four bits, called a *nibble*
- Each possible combination of four binary digits is given a symbol, as follows:

0000 = 0	0100 = 4	1000 = 8	1100 = C
0001 = 1	0101 = 5	1001 = 9	1101 = D
0010 = 2	0110 = 6	1010 = A	1110 = E
0011 = 3	0111 = 7	1011 = B	1111 = F

- Because 16 symbols are used, the notation is called *hexadecimal* and the 16 symbols are the *hexadecimal digits*
- Thus

$$2C_{16} = (2_{16} * 16^1) + (C_{16} * 16^0)$$

$$= (2_{10} * 16^1) + (12_{10} * 16^0) = 44$$

Table 9.3

Decimal, Binary, and Hexadecimal

Decimal (base 10)	Binary (base 2)	Hexadecimal (base 16)
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	0001 0000	10
17	0001 0001	11
18	0001 0010	12
31	0001 1111	1F
100	0110 0100	64
255	1111 1111	FF
256	0001 0000 0000	100

Hexadecimal Notation

Not only used for representing integers but also as a concise notation for representing any sequence of binary digits

Reasons for using hexadecimal notation are:

- It is more compact than binary notation
- In most computers, binary data occupy some multiple of 4 bits, and hence some multiple of a single hexadecimal digit
- It is extremely easy to convert between binary and hexadecimal notation

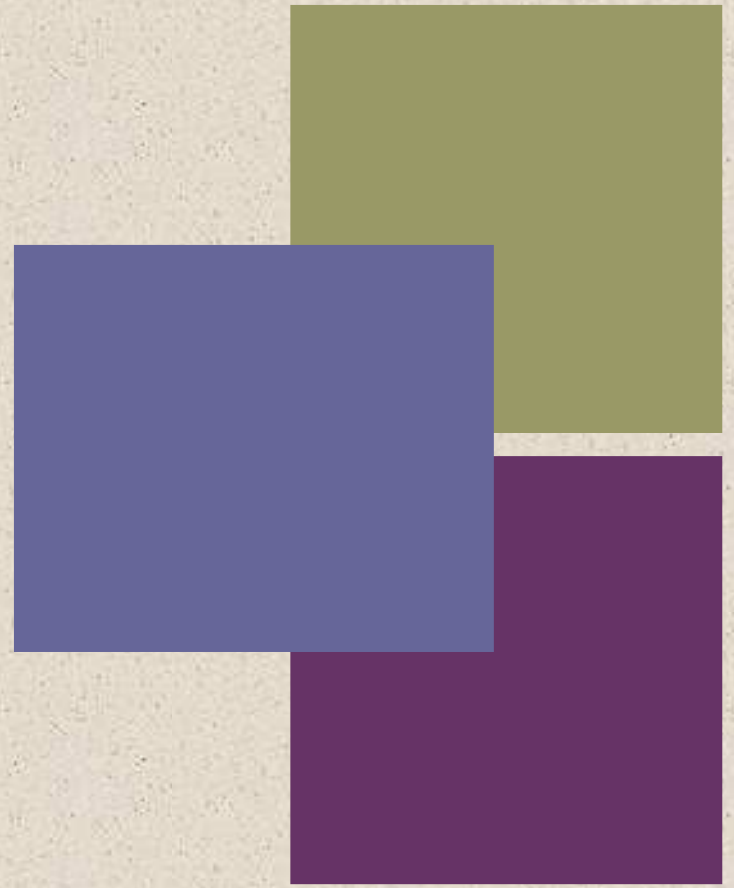
Summary

Chapter 9

Number Systems

- The decimal system
- Positional number systems
- The binary system
- Converting between binary and decimal
  - Integers
  - Fractions
- Hexadecimal notation

William Stallings
Computer Organization and Architecture
10th Edition

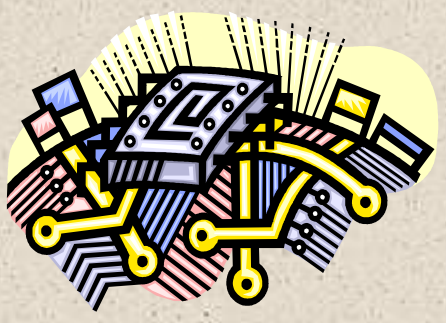


Chapter 10

Computer Arithmetic

Arithmetic & Logic Unit (ALU)

- Part of the computer that actually performs arithmetic and logical operations on data
- All of the other elements of the computer system are there mainly to bring data into the ALU for it to process and then to take the results back out
- Based on the use of simple digital logic devices that can store binary digits and perform simple Boolean logic operations



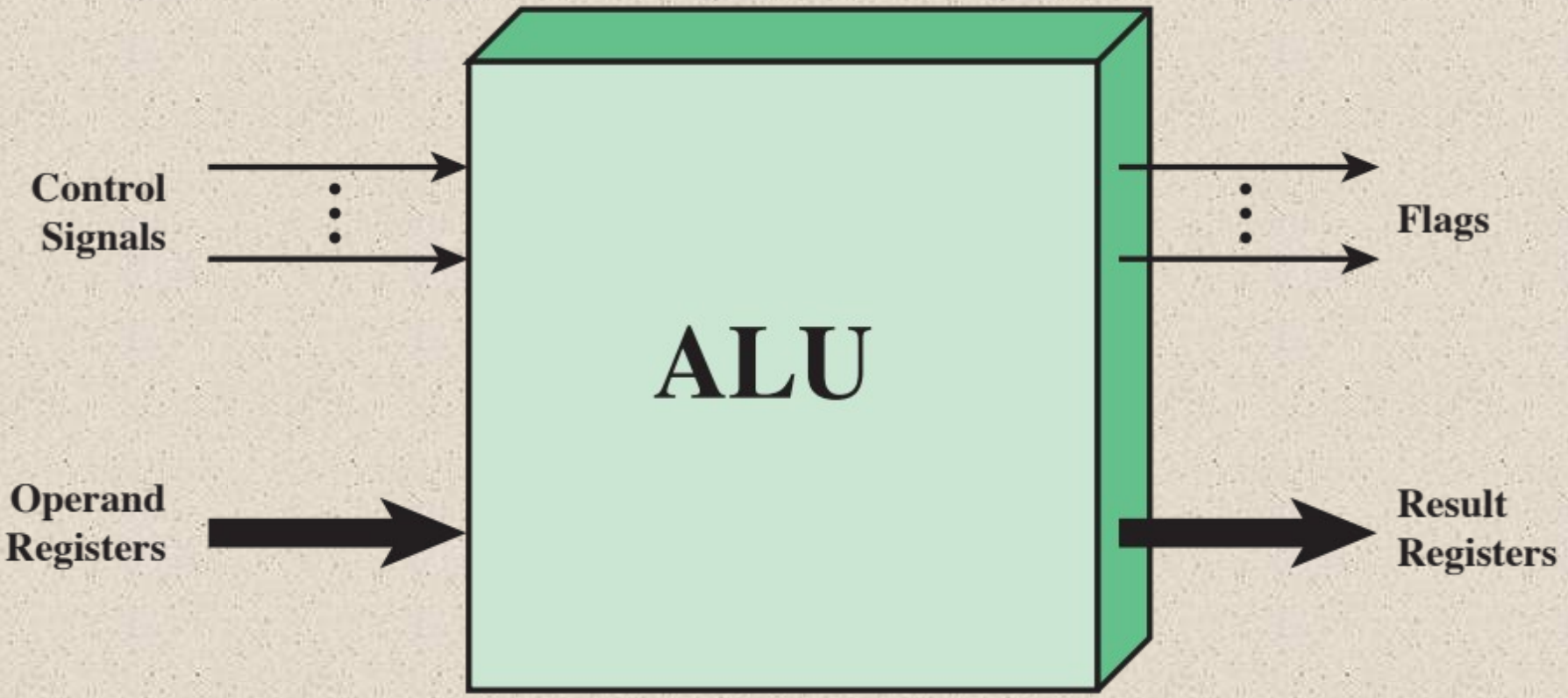


Figure 10.1 ALU Inputs and Outputs

Integer Representation

- In the binary number system arbitrary numbers can be represented with:
  - The digits zero and one
  - The minus sign (for negative numbers)
  - The period, or ***radix point*** (for numbers with a fractional component)
- For purposes of computer storage and processing we do not have the benefit of special symbols for the minus sign and radix point
- Only binary digits (0,1) may be used to represent numbers

Sign-Magnitude Representation

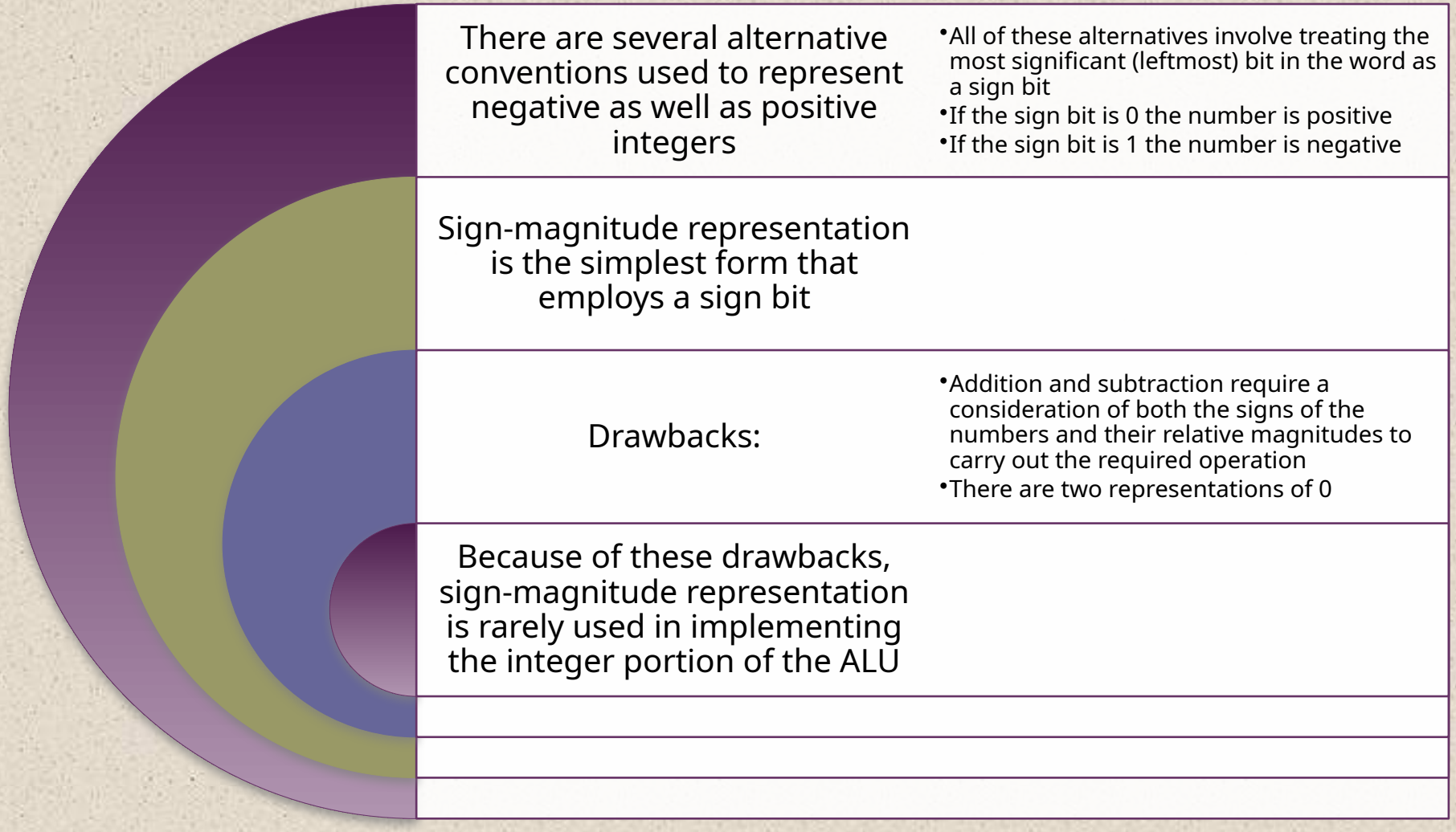


Table 10.1
Characteristics of Twos Complement Representation and Arithmetic

Range	-2_{n-1} through $2_{n-1} - 1$
Number of Representations of Zero	One
Negation	Take the Boolean complement of each bit of the corresponding positive number, then add 1 to the resulting bit pattern viewed as an unsigned integer.
Expansion of Bit Length	Add additional bit positions to the left and fill in with the value of the original sign bit.
Overflow Rule	If two numbers with the same sign (both positive or both negative) are added, then overflow occurs if and only if the result has the opposite sign.
Subtraction Rule	To subtract *B* from *A*, take the twos complement of *B* and add it to *A*.

Table 10.2
Alternative Representations for 4-Bit Integers

Decimal Representation	Sign-Magnitude Representation	Twos Complement Representation	Biased Representation
+8	—	—	1111
+7	0111	0111	1110
+6	0110	0110	1101
+5	0101	0101	1100
+4	0100	0100	1011
+3	0011	0011	1010
+2	0010	0010	1001
+1	0001	0001	1000
+0	0000	0000	0111
–0	1000	—	—
–1	1001	1111	0110
–2	1010	1110	0101
–3	1011	1101	0100
–4	1100	1100	0011
–5	1101	1011	0010
–6	1110	1010	0001
–7	1111	1001	0000
–8	—	1000	—

–128	64	32	16	8	4	2	1

(a) An eight-position two's complement value box

–128	64	32	16	8	4	2	1
1	0	0	0	0	0	1	1

–128 +2 +1 = –125

(b) Convert binary 10000011 to decimal

–128	64	32	16	8	4	2	1
1	0	0	0	1	0	0	0

–120 = –128 +8

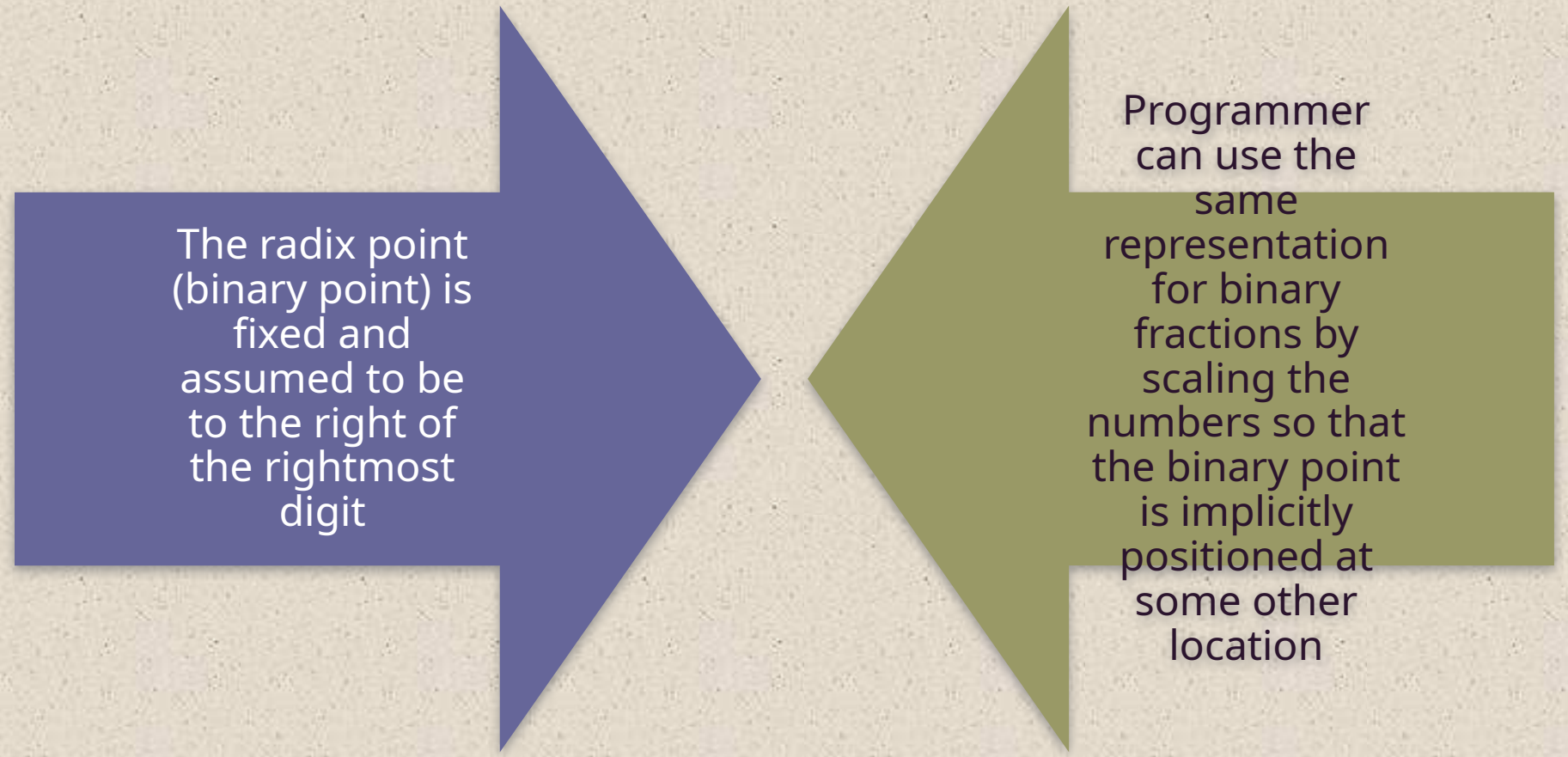
(c) Convert decimal –120 to binary

Figure 10.2 Use of a Value Box for Conversion Between Twos Complement Binary and Decimal

Range Extension

- Range of numbers that can be expressed is extended by increasing the bit length
- In sign-magnitude notation this is accomplished by moving the sign bit to the new leftmost position and fill in with zeros
- This procedure will not work for twos complement negative integers
  - Rule is to move the sign bit to the new leftmost position and fill in with copies of the sign bit
  - For positive numbers, fill in with zeros, and for negative numbers, fill in with ones
  - This is called *sign extension*

Fixed-Point Representation



Negation

- Twos complement operation
  - Take the Boolean complement of each bit of the integer (including the sign bit)
  - Treating the result as an unsigned binary integer, add 1

```
                    +18 = 00010010 (twos complement)
bitwise complement = 11101101
                       +        1
                      ----------
                      11101110 = -18
```

- The negative of the negative of that number is itself:

```
                    -18 =  11101110 (twos complement)
bitwise complement =  00010001
                       +        1
                      ----------
                       00010010 = +18
```

Negation Special Case 1

0 = 00000000 (twos complement)

Bitwise complement = 11111111

Add 1 to LSB + 1

Result 100000000

Overflow is ignored, so:

- 0 = 0

Negation Special Case 2

-128 = 10000000 (twos complement)

Bitwise complement = 01111111

Add 1 to LSB + 1

Result 10000000

So:

-(-128) = -128 X

Monitor MSB (sign bit)

It should change during negation

1001 = –7 +0101 = 5 1110 = –2	1100 = –4 +0100 = 4 10000 = 0
(a) (–7) + (+5)	(b) (–4) + (+4)
0011 = 3 +0100 = 4 0111 = 7	1100 = –4 +1111 = –1 11011 = –5
(c) (+3) + (+4)	(d) (–4) + (–1)
0101 = 5 +0100 = 4 1001 = Overflow	1001 = –7 +1010 = –6 10011 = Overflow
(e) (+5) + (+4)	(f) (–7) + (–6)

Figure 10.3 Addition of Numbers in Twos Complement Representation

Overflow

Rule

OVERFLOW RULE:

If two numbers are added, and they are both positive or both negative, then overflow occurs if and only if the result has the opposite sign.

Subtraction

Rule

SUBTRACTION RULE:

To subtract one number (subtrahend) from another (minuend), take the twos complement (negation) of the subtrahend and add it to the minuend.

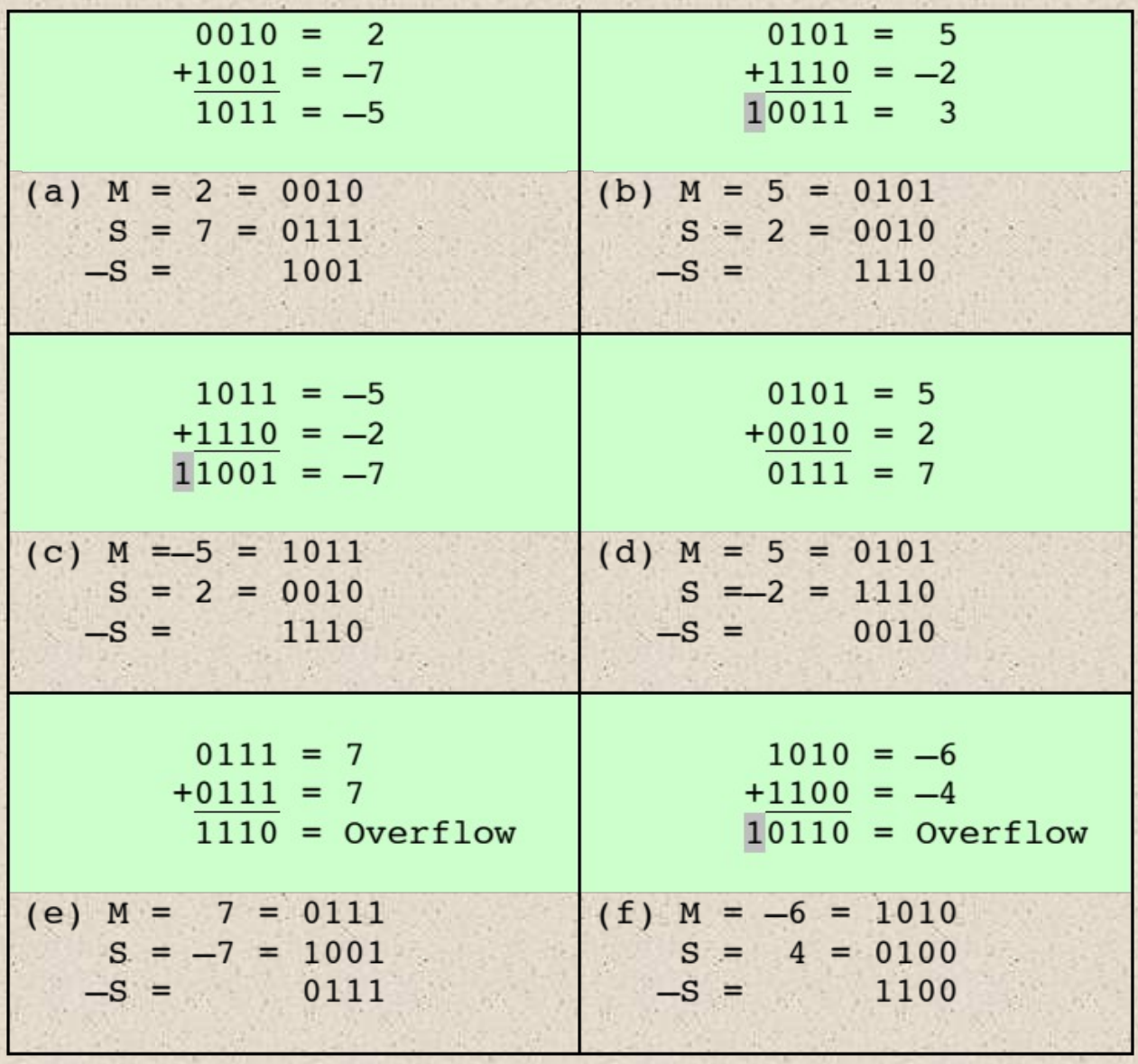


Figure 10.4 Subtraction of Numbers in Twos Complement Representation (M – S)

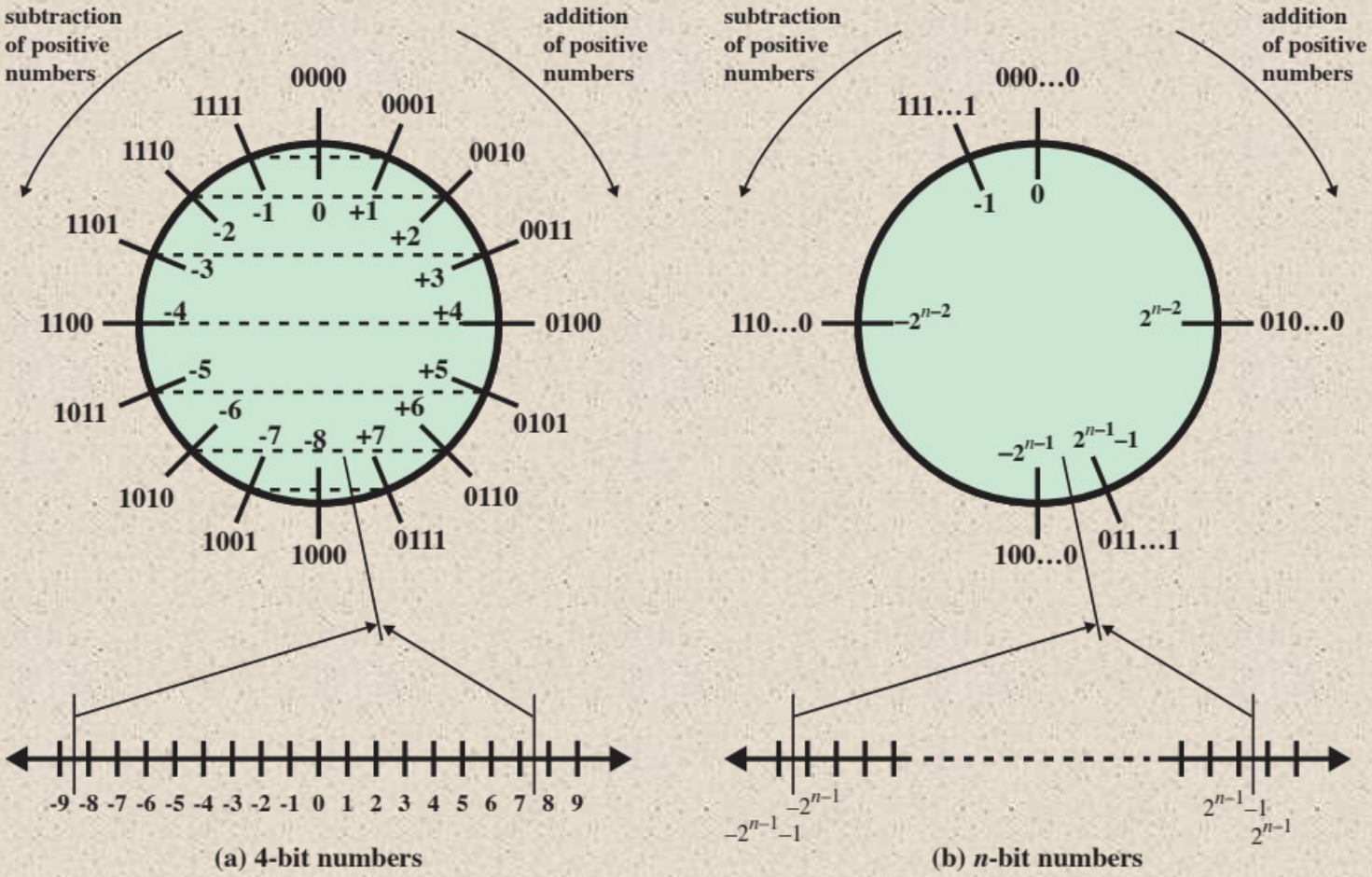
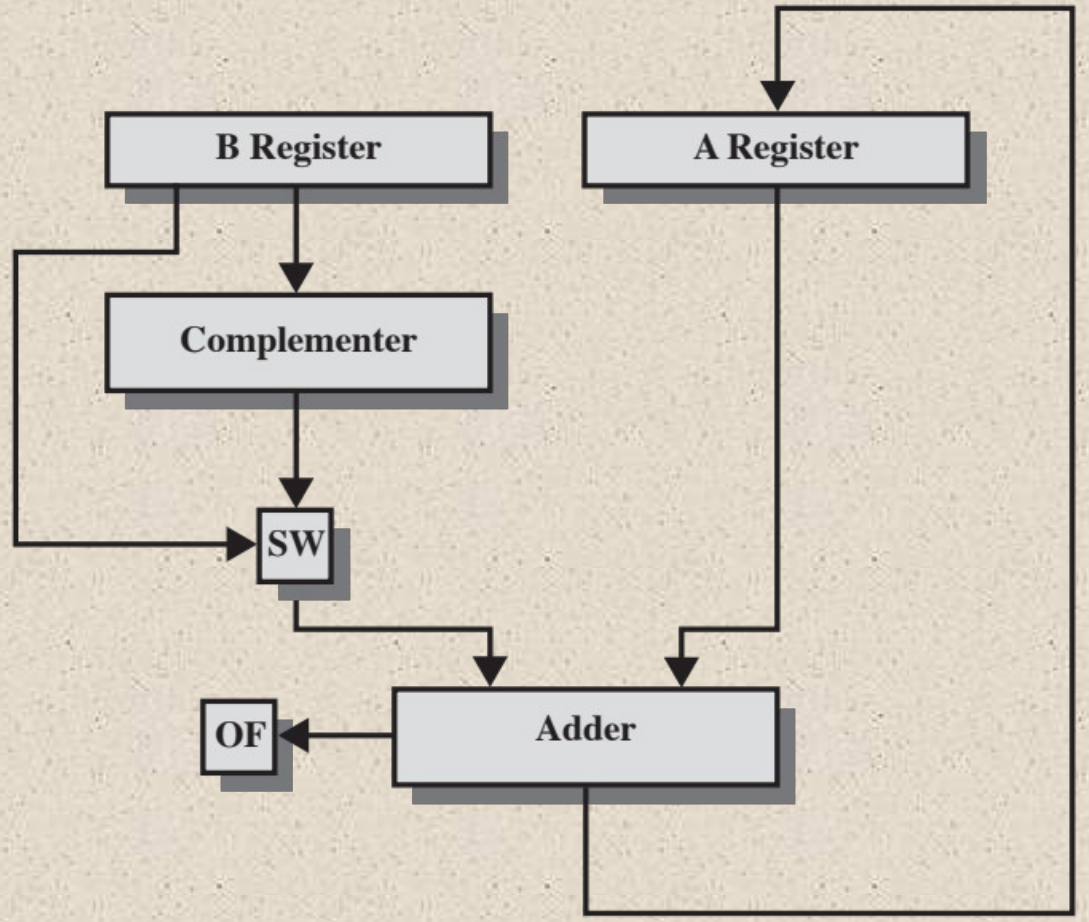


Figure 10.5 Geometric Depiction of Twos Complement Integers



OF = overflow bit
SW = Switch (select addition or subtraction)

Figure 10.6 Block Diagram of Hardware for Addition and Subtraction

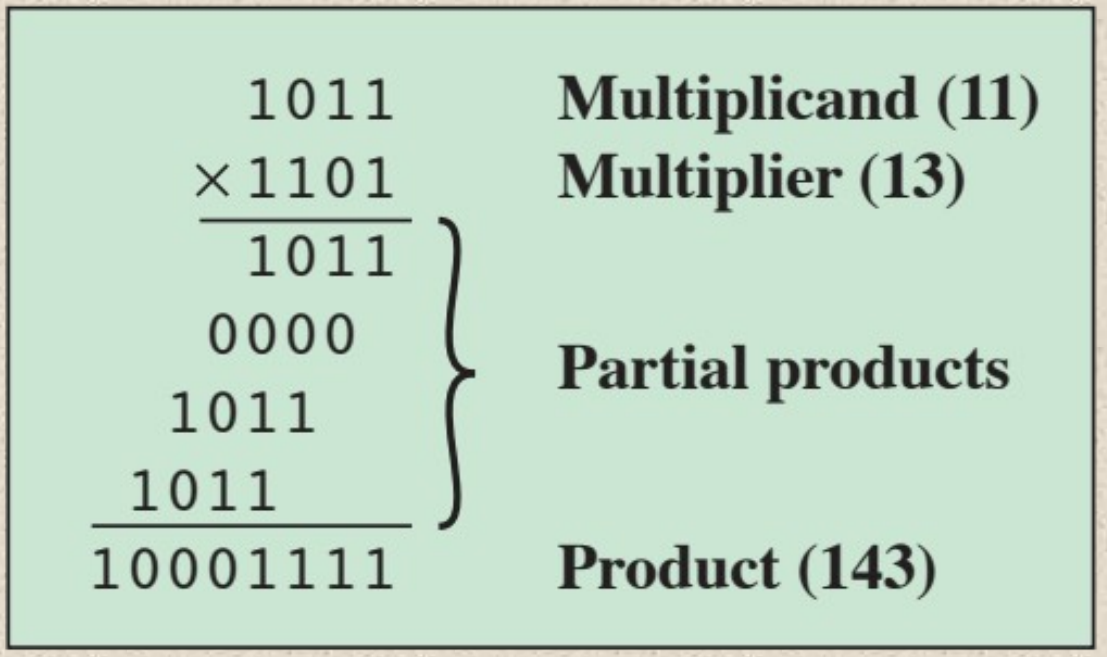
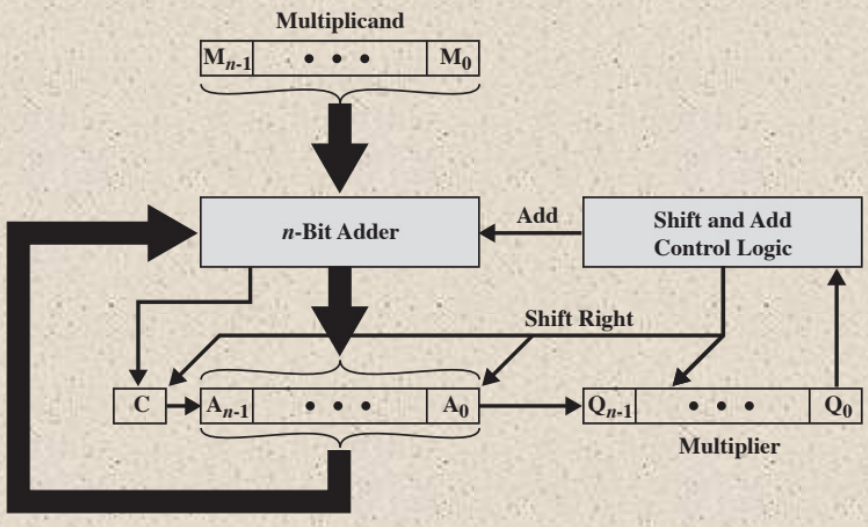


Figure 10.7 Multiplication of Unsigned Binary Integers



(a) Block Diagram

C	A	Q	M		
0	0000	1101	1011	Initial Values	
0	1011	1101	1011	Add	First Cycle
0	0101	1110	1011	Shift	
0	0010	1111	1011	Shift	Second Cycle
0	1101	1111	1011	Add	Third Cycle
0	0110	1111	1011	Shift	
1	0001	1111	1011	Add	Fourth Cycle
0	1000	1111	1011	Shift	

(b) Example from Figure 9.7 (product in A, Q)

Figure 10.8 Hardware Implementation of Unsigned Binary Multiplication

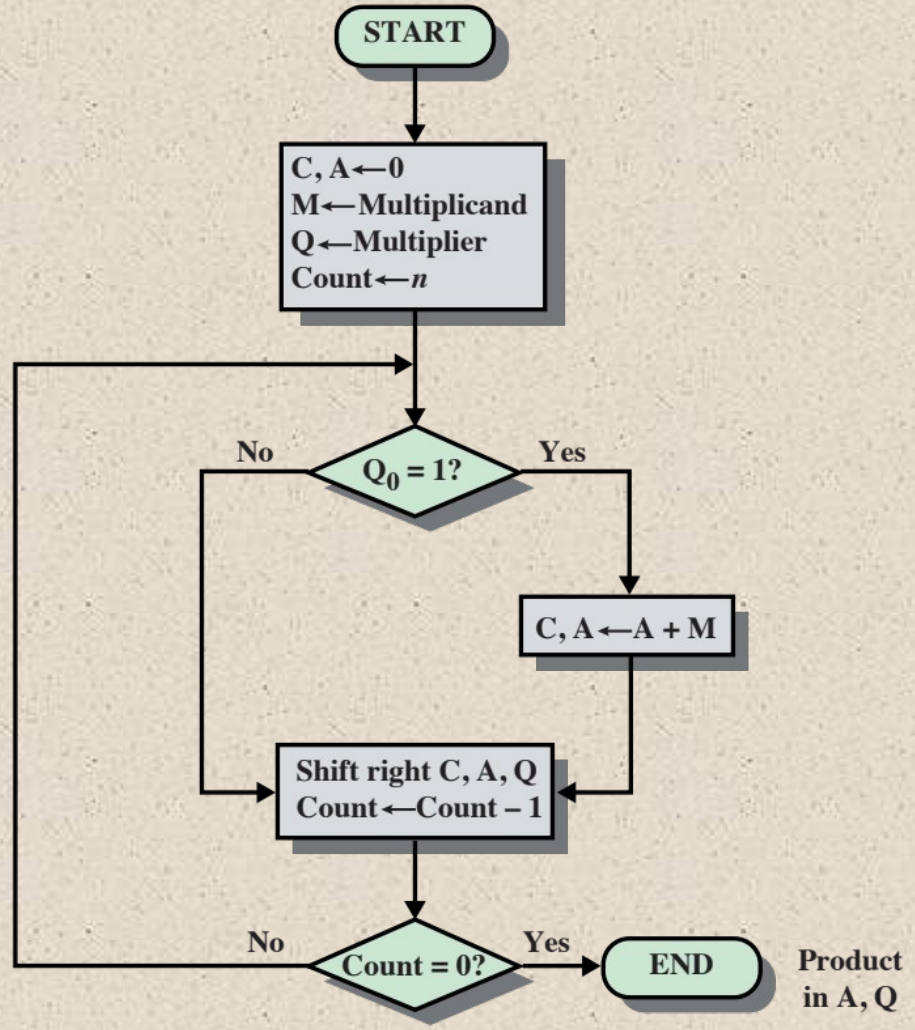


Figure 10.9 Flowchart for Unsigned Binary Multiplication

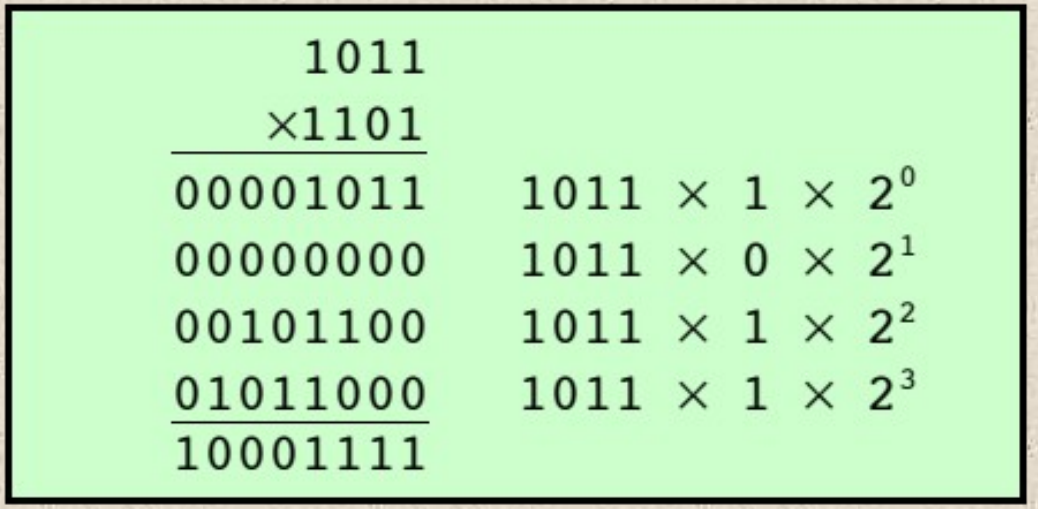
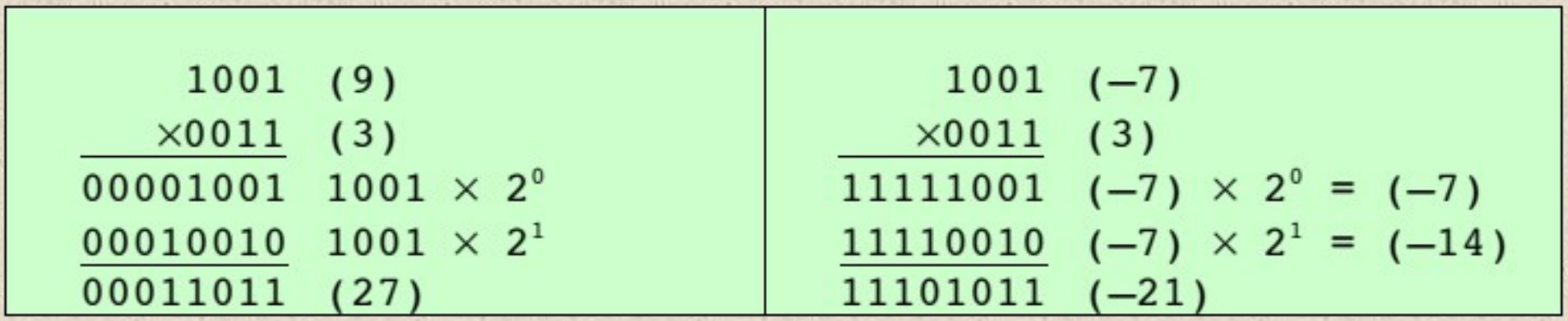


Figure 10.10 Multiplication of Two Unsigned 4-Bit Integers Yielding an 8-Bit Result



(a) Unsigned integers

(b) Twos complement integers

Figure 10.11 Comparison of Multiplication of Unsigned and Twos Complement Integers

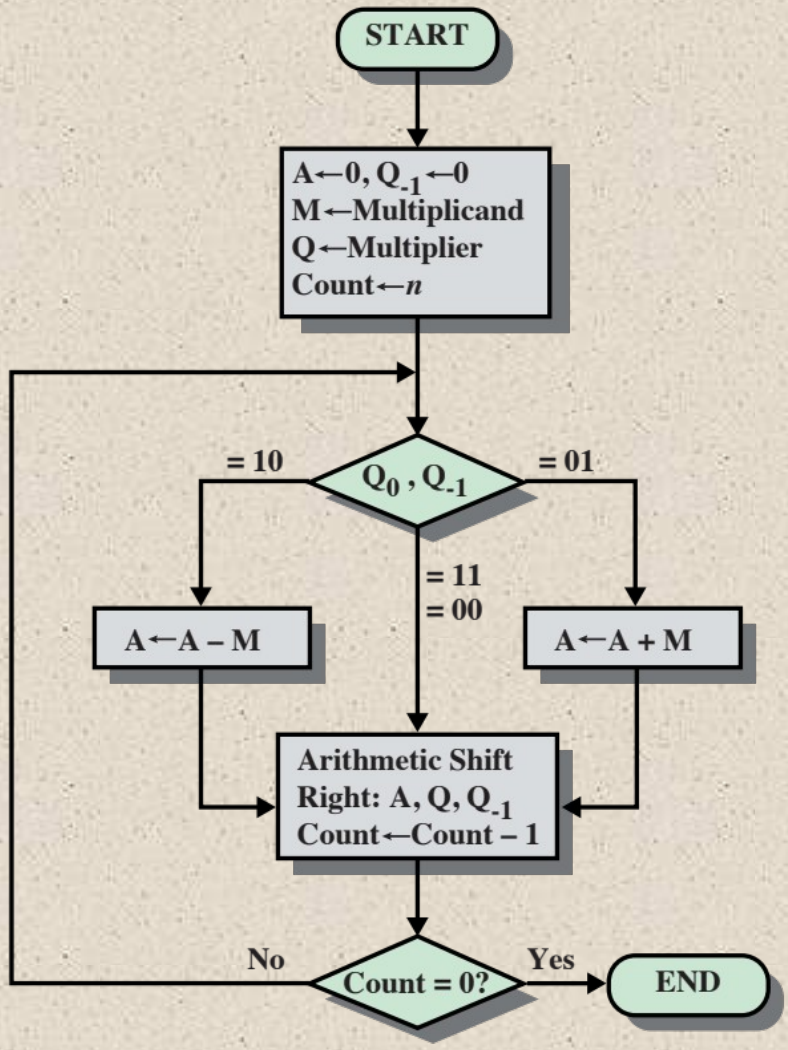


Figure 10.12 Booth's Algorithm for Twos Complement Multiplication

A	Q	Q_{-1}	M		
0000	0011	0	0111	Initial Values	
1001	0011	0	0111	$A \leftarrow A - M$	First Cycle
1100	1001	1	0111	Shift	
1110	0100	1	0111	Shift	Second Cycle
0101	0100	1	0111	$A \leftarrow A + M$	Third Cycle
0010	1010	0	0111	Shift	
0001	0101	0	0111	Shift	Fourth Cycle

Figure 10.13 Example of Booth's Algorithm (7× 3)

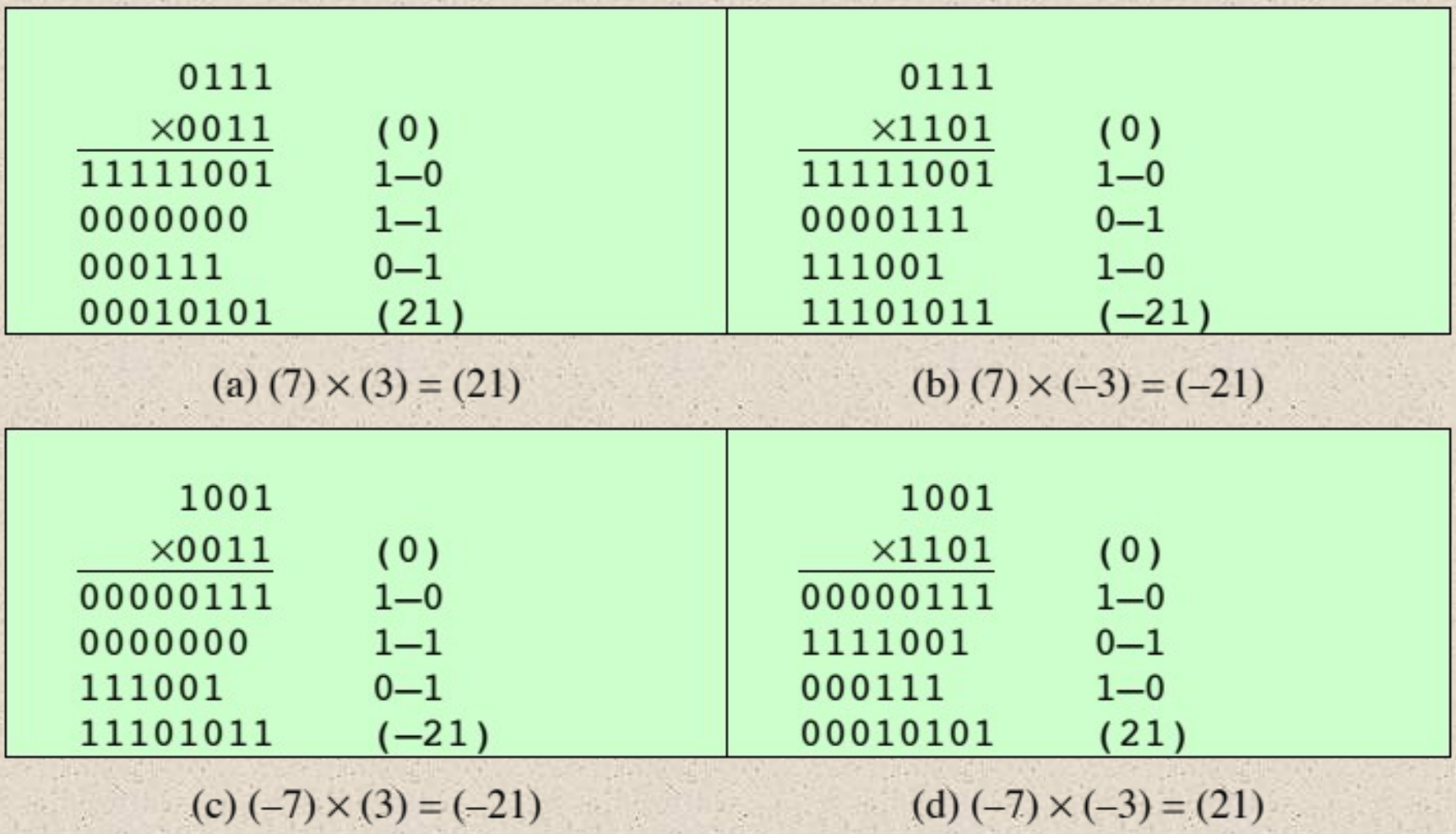


Figure 10.14 Examples Using Booth's Algorithm

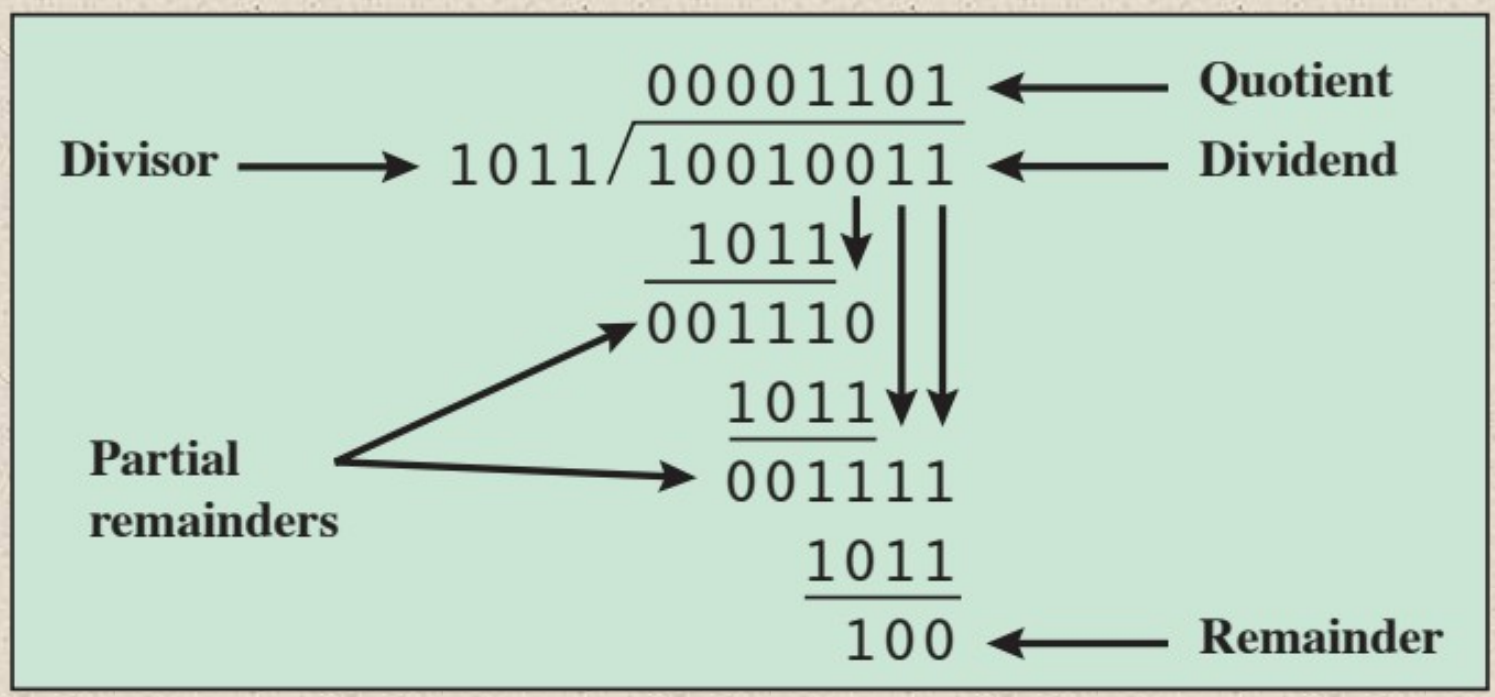


Figure 10.15 Example of Division of Unsigned Binary Integers

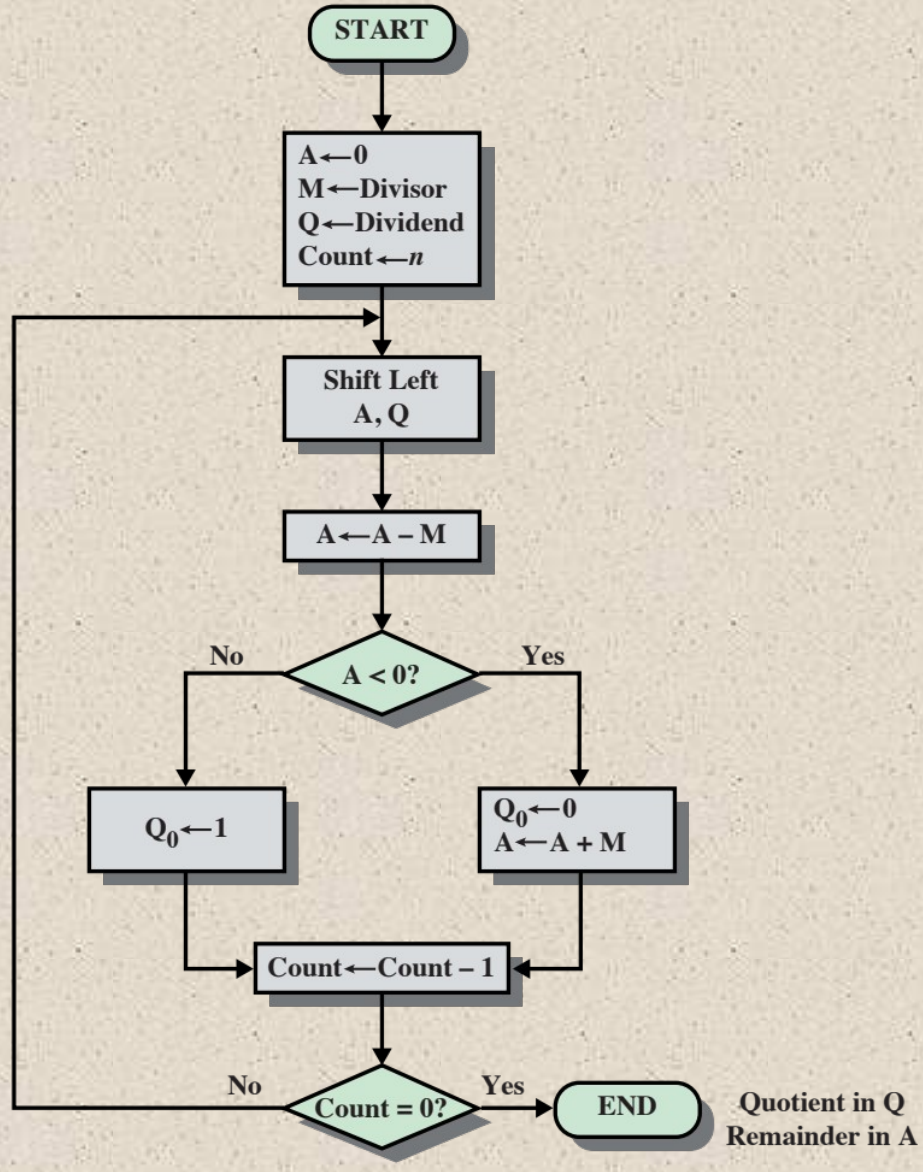


Figure 10.16 Flowchart for Unsigned Binary Division

A	Q	
0000	0111	Initial value
0000	1110	Shift
1101		Use twos complement of 0011 for subtraction
1101		Subtract
0000	1110	Restore, set $Q_0 = 0$
0001	1100	Shift
1101		
1110		Subtract
0001	1100	Restore, set $Q_0 = 0$
0011	1000	Shift
1101		
0000	1001	Subtract, set $Q_0 = 1$
0001	0010	Shift
1101		
1110		Subtract
0001	0010	Restore, set $Q_0 = 0$

Figure 10.17 Example of Restoring Twos Complement Division (7/3)

Floating-Point Representation

Principles

- With a fixed-point notation it is possible to represent a range of positive and negative integers centered on or near 0
- By assuming a fixed binary or radix point, this format allows the representation of numbers with a fractional component as well
- Limitations:
  - Very large numbers cannot be represented nor can very small fractions
  - The fractional part of the quotient in a division of two large numbers could be lost

(a) Format

$1.1010001 \times 2^{10100} = 0\ 10010011\ 10100010000000000000000 = 1.6328125 \times 2^{20}$

$-1.1010001 \times 2^{10100} = 1\ 10010011\ 10100010000000000000000 = -1.6328125 \times 2^{20}$

$1.1010001 \times 2^{-10100} = 0\ 01101011\ 10100010000000000000000 = 1.6328125 \times 2^{-20}$

$-1.1010001 \times 2^{-10100} = 1\ 01101011\ 10100010000000000000000 = -1.6328125 \times 2^{-20}$

(b) Examples

Figure 10.18 Typical 32-Bit Floating-Point Format

Floating-Point

Significand

- The final portion of the word
- Any floating-point number can be expressed in many ways

The following are equivalent, where the significand is expressed in binary form:

$0.110 * 2^5$

$110 * 2^2$

$0.0110 * 2^6$

- *Normal number*
  - The most significant digit of the significand is nonzero

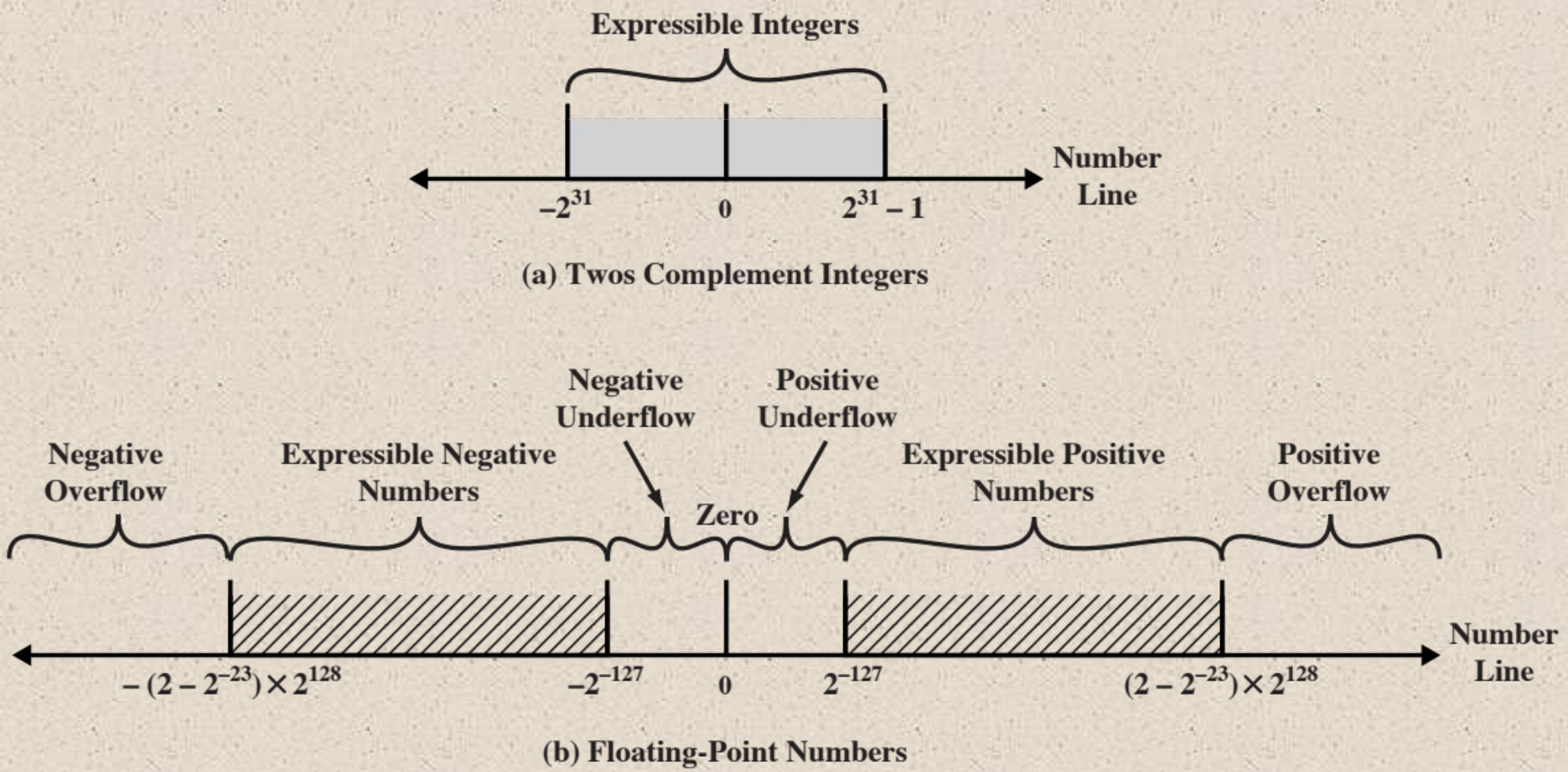


Figure 10.19 Expressible Numbers in Typical 32-Bit Formats

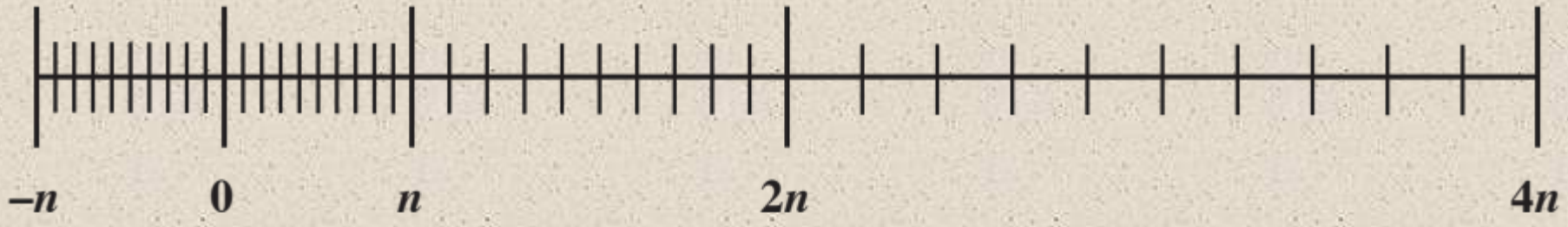


Figure 10.20 Density of Floating-Point Numbers

IEEE Standard 754

Most important floating-point representation is defined

Standard was developed to facilitate the portability of programs from one processor to another and to encourage the development of sophisticated, numerically oriented programs

Standard has been widely adopted and is used on virtually all contemporary processors and arithmetic coprocessors

IEEE 754-2008 covers both binary and decimal floating-point representations

IEEE 754-2008

- Defines the following different types of floating-point formats:
- Arithmetic format
  - All the mandatory operations defined by the standard are supported by the format. The format may be used to represent floating-point operands or results for the operations described in the standard.
- Basic format
  - This format covers five floating-point representations, three binary and two decimal, whose encodings are specified by the standard, and which can be used for arithmetic. At least one of the basic formats is implemented in any conforming implementation.
- Interchange format
  - A fully specified, fixed-length binary encoding that allows data interchange between different platforms and that can be used for storage.

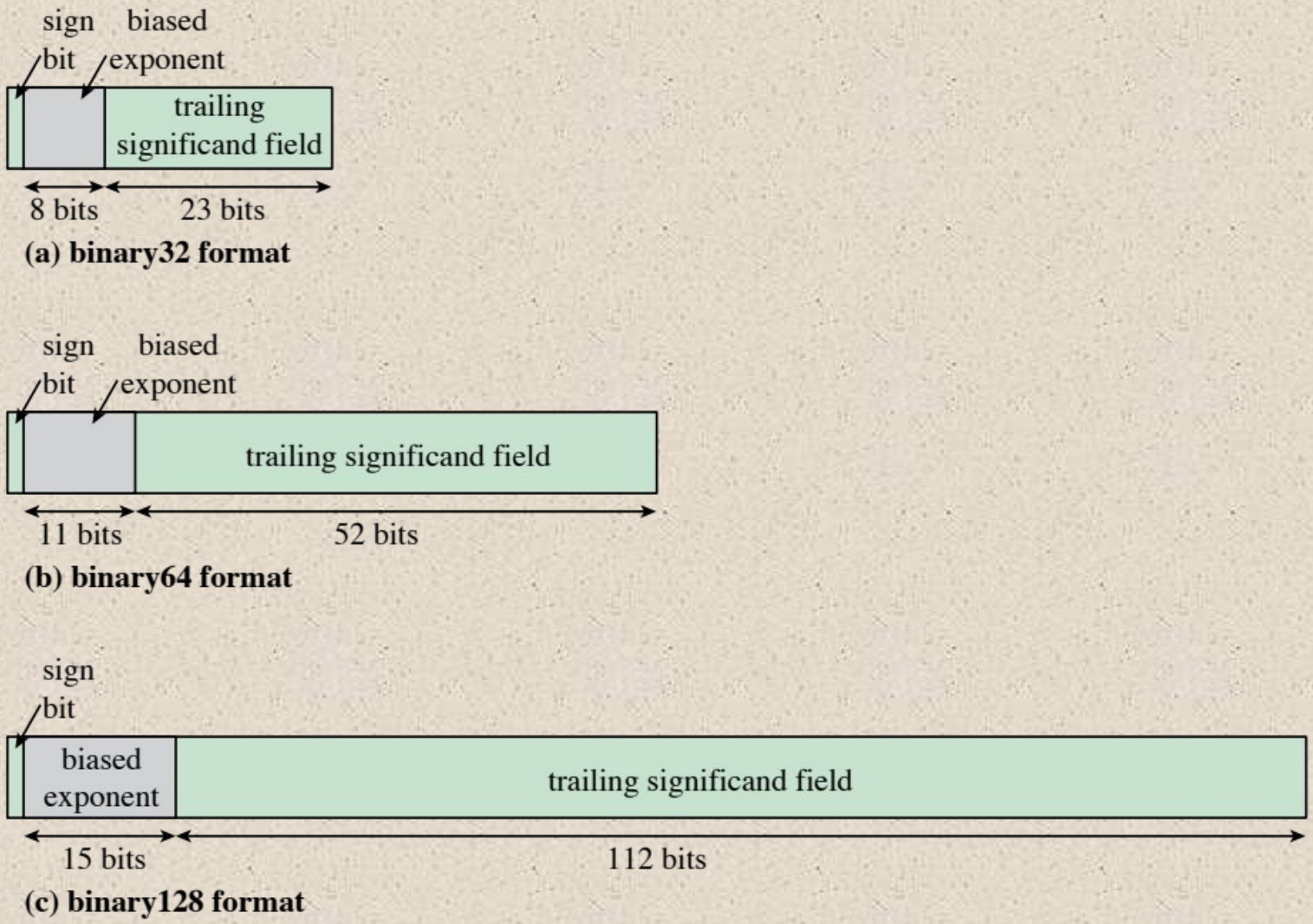


Figure 10.21 IEEE 754 Formats

Table 10.3 IEEE 754 Format Parameters

Parameter	Format		
	binary32	**binary64**	**binary128**
Storage width (bits)	32	64	128
Exponent width (bits)	8	11	15
Exponent bias	127	1023	16383
Maximum exponent	127	1023	16383
Minimum exponent	–126	–1022	–16382
Approx normal number range (base 10)	$10_{-38}, 10_{+38}$	$10_{-308}, 10_{+308}$	$10_{-4932}, 10_{+4932}$
Trailing significand width (bits)*	23	52	112
Number of exponents	254	2046	32766
Number of fractions	2_{23}	2_{52}	2_{112}
Number of values	$1.98 \times 2_{31}$	$1.99 \times 2_{63}$	$1.99 \times 2_{128}$
Smallest positive normal number	2_{-126}	2_{-1022}	2_{-16362}
Largest positive normal number	$2_{128} - 2_{104}$	$2_{1024} - 2_{971}$	$2_{16384} - 2_{16271}$
Smallest subnormal magnitude	2_{-149}	2_{-1074}	2_{-16494}

* not including implied bit and not including sign bit

Additional Formats

Extended Precision Formats

- Provide additional bits in the exponent (extended range) and in the significand (extended precision)
- Lessens the chance of a final result that has been contaminated by excessive roundoff error
- Lessens the chance of an intermediate overflow aborting a computation whose final result would have been representable in a basic format
- Affords some of the benefits of a larger basic format without incurring the time penalty usually associated with higher precision

Extendable Precision Format

- Precision and range are defined under user control
- May be used for intermediate calculations but the standard places no constraint or format or length

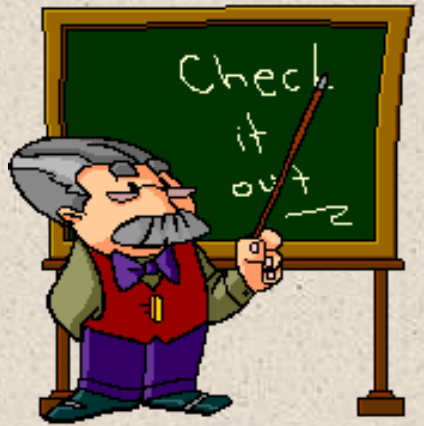


Table 10.4
IEEE Formats

Format	Format Type		
	Arithmetic Format	Basic Format	Interchange Format
binary16			X
binary32	X	X	X
binary64	X	X	X
binary128	X	X	X
binary{k} ($k = n \times 32$ for $n > 4$)	X		X
decimal64	X	X	X
decimal128	X	X	X
decimal{k} ($k = n \times 32$ for $n > 4$)	X		X
extended precision	X		
extendable precision	X		

Table 10.5
Interpretation of IEEE 754 Floating-Point Numbers (page 1 of 3)

	Sign	Biased exponent	Fraction	Value
positive zero	0	0	0	0
negative zero	1	0	0	–0
plus infinity	0	all 1s	0	∞
Minus infinity	1	all 1s	0	$-\infty$
quiet NaN	0 or 1	all 1s	$\neq 0$; first bit = 1	qNaN
signaling NaN	0 or 1	all 1s	$\neq 0$; first bit = 0	sNaN
positive normal nonzero	0	$0 < e < 255$	f	$2_{e-127}(1.f)$
negative normal nonzero	1	$0 < e < 255$	f	$-2_{e-127}(1.f)$
positive subnormal	0	0	$f \neq 0$	$2_{e-126}(0.f)$
negative subnormal	1	0	$f \neq 0$	$-2_{e-126}(0.f)$

(a) binary32 format

Table 10.5
Interpretation of IEEE 754 Floating-Point Numbers (page 2 of 3)

	Sign	Biased exponent	Fraction	Value
positive zero	0	0	0	0
negative zero	1	0	0	–0
plus infinity	0	all 1s	0	∞
Minus infinity	1	all 1s	0	$-\infty$
quiet NaN	0 or 1	all 1s	$\neq 0$; first bit = 1	qNaN
signaling NaN	0 or 1	all 1s	$\neq 0$; first bit = 0	sNaN
positive normal nonzero	0	$0 < e < 2047$	f	$2_{e-1023}(1.f)$
negative normal nonzero	1	$0 < e < 2047$	f	$-2_{e-1023}(1.f)$
positive subnormal	0	0	$f \neq 0$	$2_{e-1022}(0.f)$
negative subnormal	1	0	$f \neq 0$	$-2_{e-1022}(0.f)$

(a) binary64 format

Table 10.5
Interpretation of IEEE 754 Floating-Point Numbers (page 3 of 3)

	Sign	Biased exponent	Fraction	Value
positive zero	0	0	0	0
negative zero	1	0	0	–0
plus infinity	0	all 1s	0	∞
minus infinity	1	all 1s	0	$-\infty$
quiet NaN	0 or 1	all 1s	$\neq 0$; first bit = 1	qNaN
signaling NaN	0 or 1	all 1s	$\neq 0$; first bit = 0	sNaN
positive normal nonzero	0	all 1s	f	$2_{e-16383}(1.f)$
negative normal nonzero	1	all 1s	f	$-2_{e-16383}(1.f)$
positive subnormal	0	0	$f \neq 0$	$2_{e-16383}(0.f)$
negative subnormal	1	0	$f \neq 0$	$-2_{e-16383}(0.f)$

(a) binary128 format

Table 10.6 Floating-Point Numbers and Arithmetic Operations

Floating Point Numbers	Arithmetic Operations
$X = X_S \times B^{X_E}$ $Y = Y_S \times B^{Y_E}$	$\left.\begin{aligned} X + Y &= \left(X_s \times B^{X_E - Y_E} + Y_s\right) \times B^{Y_E} \\ X - Y &= \left(X_s \times B^{X_E - Y_E} - Y_s\right) \times B^{Y_E} \end{aligned}\right\} X_E \le Y_E$ $X \times Y = \left(X_s \times Y_s\right) \times B^{X_E + Y_E}$ $\frac{X}{Y} = \left(\frac{X_s}{Y_s}\right) \times B^{X_E - Y_E}$

Examples:

$X = 0.3 \times 10^2 = 30$
$Y = 0.2 \times 10^3 = 200$

$X + Y = (0.3 \times 10_{2-3} + 0.2) \times 10_3 = 0.23 \times 10_3 = 230$
$X - Y = (0.3 \times 10_{2-3} - 0.2) \times 10_3 = (-0.17) \times 10_3 = -170$
$X \times Y = (0.3 \times 0.2) \times 10_{2+3} = 0.06 \times 10_5 = 6000$
$X \div Y = (0.3 \div 0.2) \times 10_{2-3} = 1.5 \times 10_{-1} = 0.15$

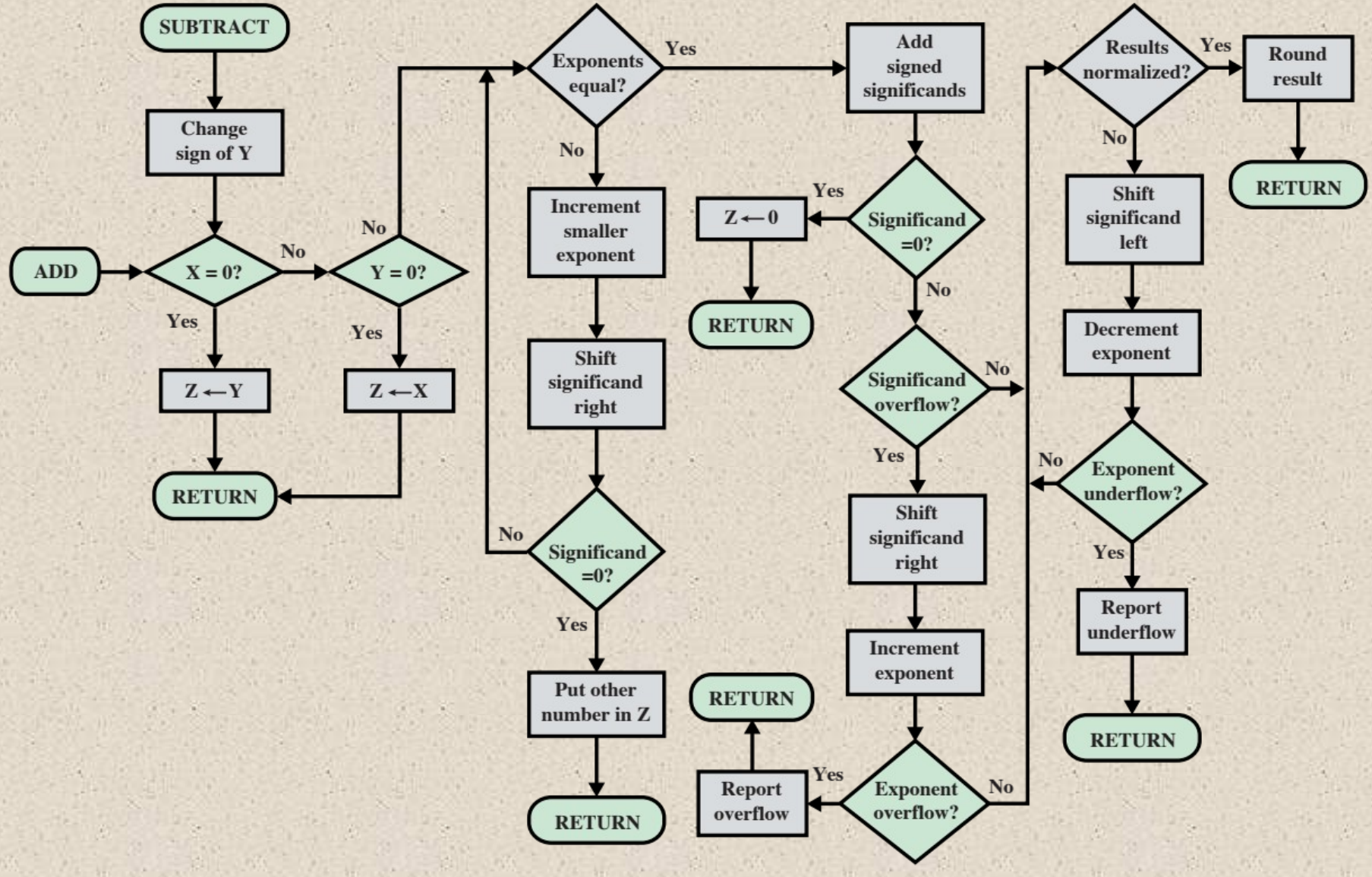


Figure 10.22 Floating-Point Addition and Subtraction ($Z \leftarrow X \pm Y$)

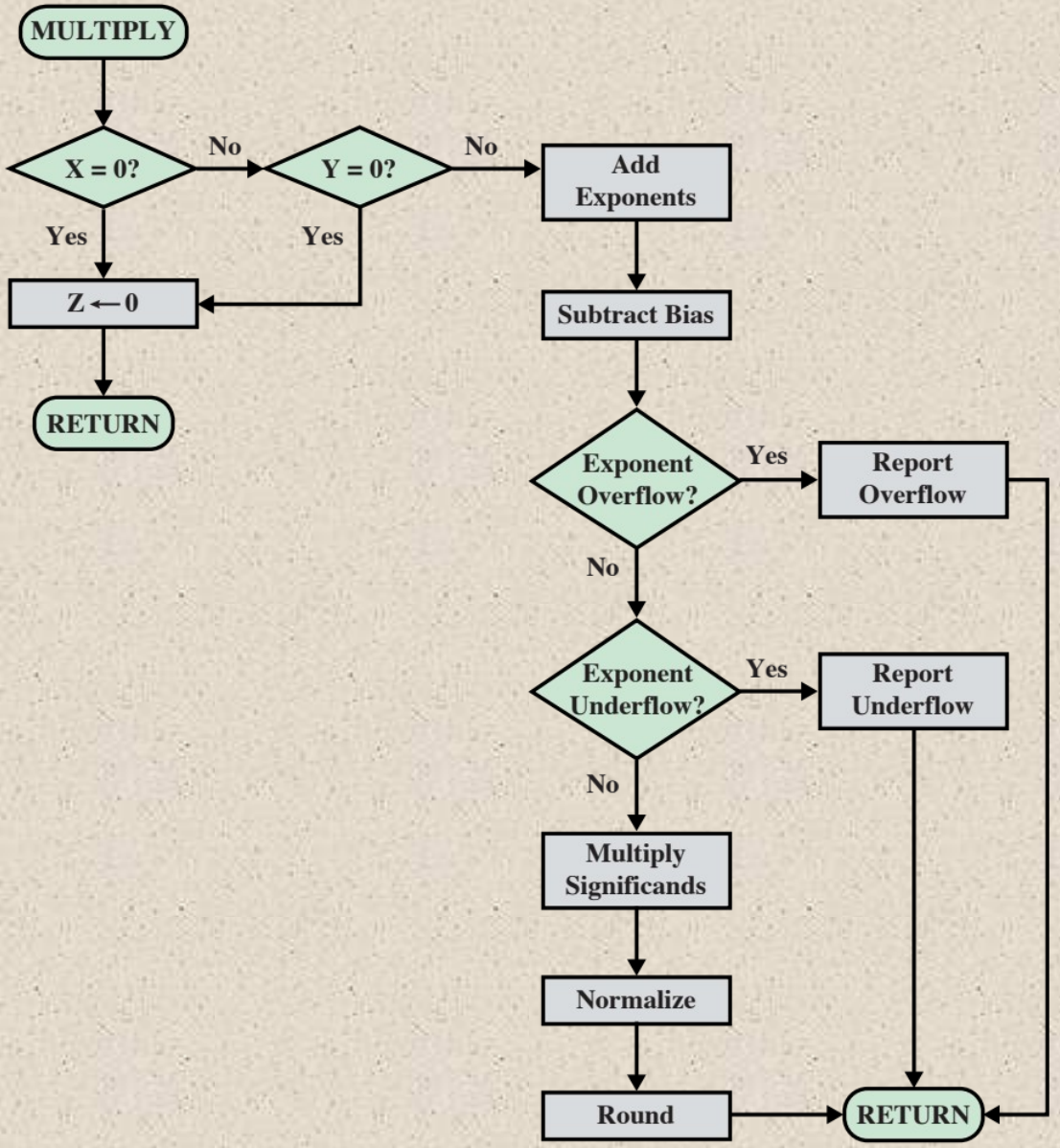


Figure 10.23 Floating-Point Multiplication (Z← X× Y)

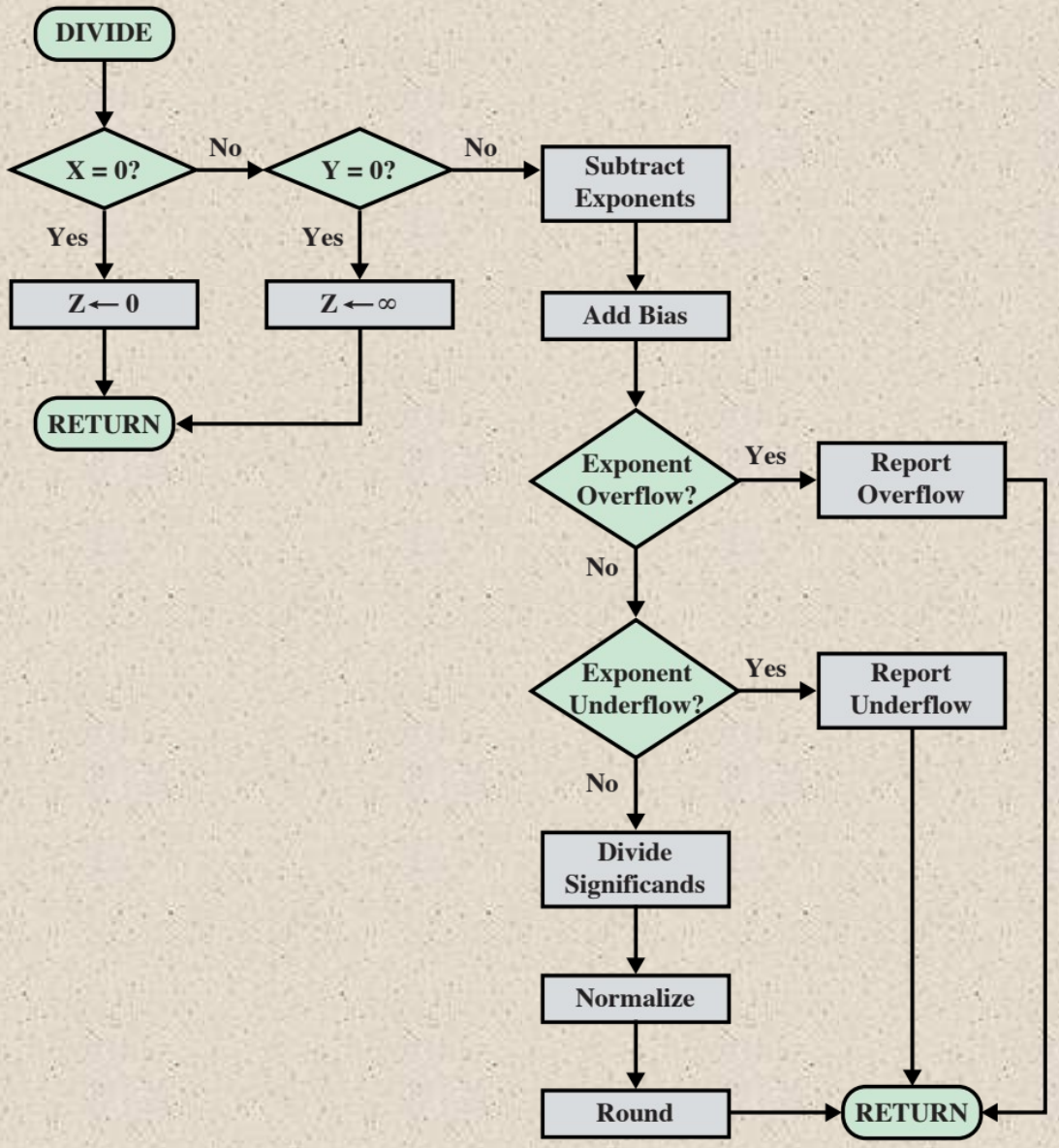


Figure 10.24 Floating-Point Division (Z← X/Y)

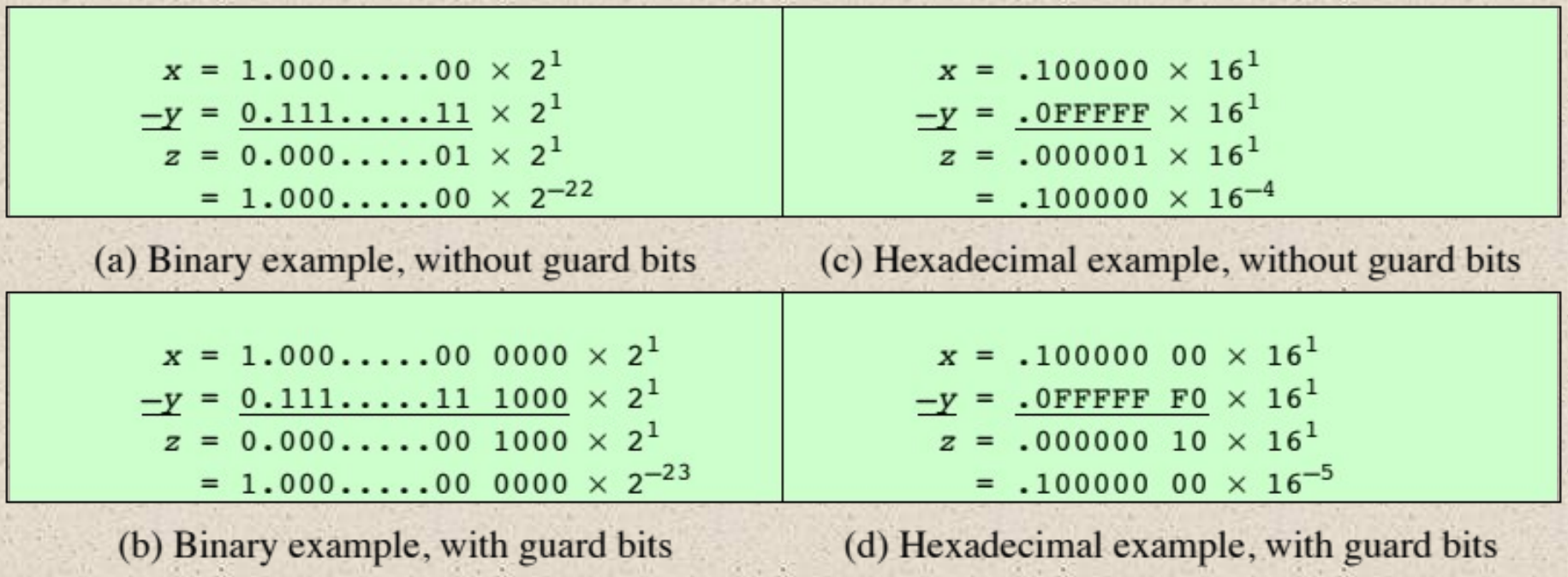


Figure 10.25 The Use of Guard Bits

Precision Considerations

Rounding

- IEEE standard approaches:
  - Round to nearest:
    - The result is rounded to the nearest representable number.
  - Round toward +∞ :
    - The result is rounded up toward plus infinity.
  - Round toward -∞:
    - The result is rounded down toward negative infinity.
  - Round toward 0:
    - The result is rounded toward zero.

Interval Arithmetic

- Provides an efficient method for monitoring and controlling errors in floating-point computations by producing two values for each result
- The two values correspond to the lower and upper endpoints of an interval that contains the true result
- The width of the interval indicates the accuracy of the result
- If the endpoints are not representable then the interval endpoints are rounded down and up respectively
- If the range between the upper and lower bounds is sufficiently narrow then a sufficiently accurate result has been obtained

- *Minus infinity* and *rounding to plus* are useful in implementing interval arithmetic

Truncation

- *Round toward zero*
- Extra bits are ignored
- Simplest technique
- A consistent bias toward zero in the operation
  - Serious bias because it affects every operation for which there are nonzero extra bits

IEEE Standard for Binary Floating-Point Arithmetic

Infinity

Is treated as the limiting case of real arithmetic, with the infinity values given the following interpretation:

$$-\infty < \text{(every finite number)} < +\infty$$

For example:

$5 + (+\infty) = +\infty$

$5 - (+\infty) = -\infty$

$5 + (-\infty) = -\infty$

$5 - (-\infty) = +\infty$

$5 * (+\infty) = +\infty$

$5 \div (+\infty) = +0$

$(+\infty) + (+\infty) = +\infty$

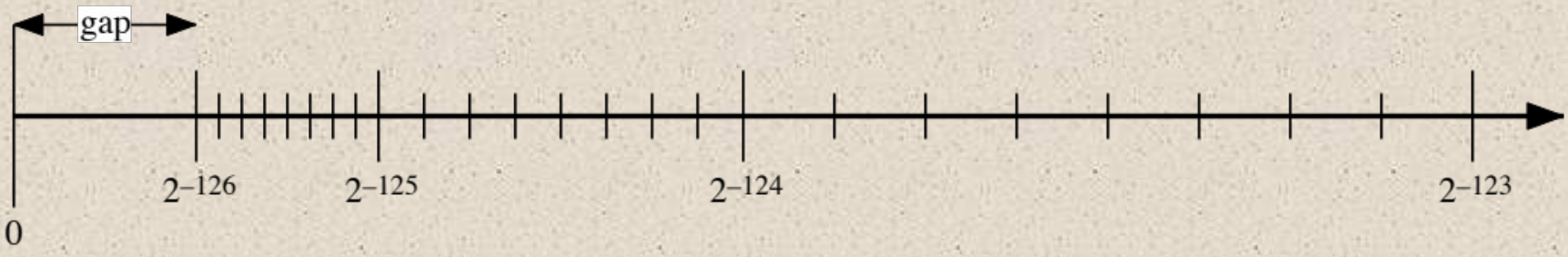
$(-\infty) + (-\infty) = -\infty$

$(-\infty) - (+\infty) = -\infty$

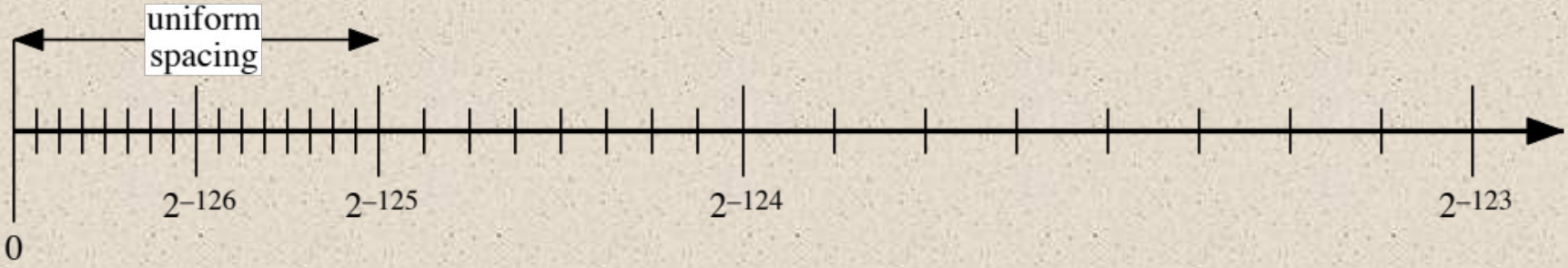
$(+\infty) - (-\infty) = +\infty$

Table 10.7
Operations that Produce a Quiet NaN

Operation	Quiet NaN Produced by
Any	Any operation on a signaling NaN
Add or subtract	Magnitude subtraction of infinities: $(+\infty) + (-\infty)$ $(-\infty) + (+\infty)$ $(+\infty) - (+\infty)$ $(-\infty) - (-\infty)$
Multiply	$0 \times \infty$
Division	$\frac{0}{0}$ or $\frac{\infty}{\infty}$
Remainder	x REM 0 or ∞ REM y
Square root	$\sqrt{x}$ where $x < 0$



(a) 32-bit format without subnormal numbers



(b) 32-bit format with subnormal numbers

Figure 10.26 The Effect of IEEE 754 Subnormal Numbers

Summary

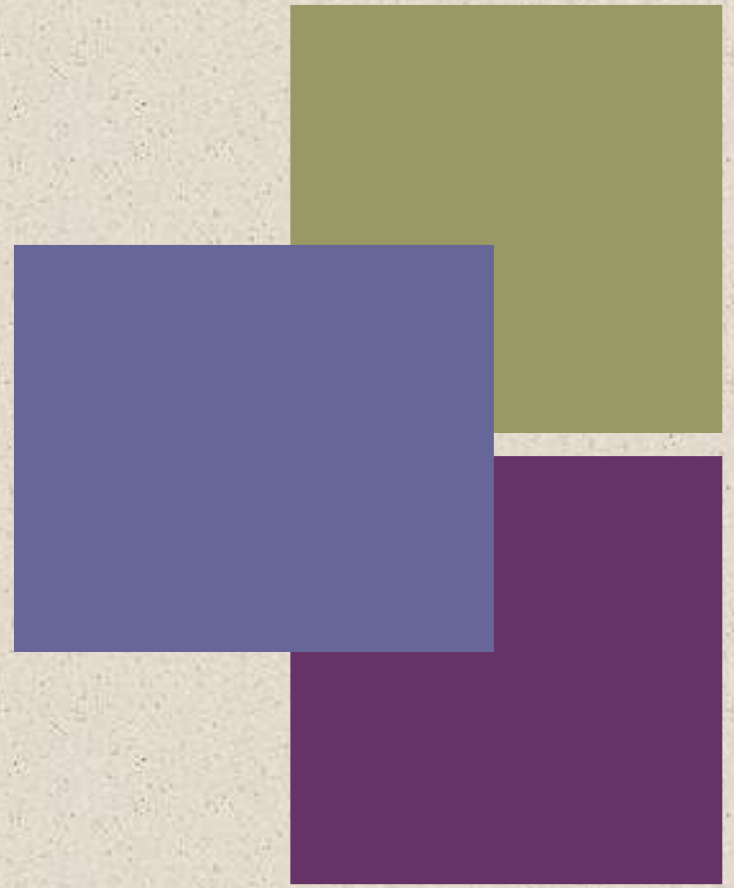
Chapter 10

Computer Arithmetic

- ALU
- Integer representation
  - Sign-magnitude representation
  - Twos complement representation
  - Range extension
  - Fixed-point representation
- Floating-point representation
  - Principles
  - IEEE standard for binary floating-point representation

- Integer arithmetic
  - Negation
  - Addition and subtraction
  - Multiplication
  - Division
- Floating-point arithmetic
  - Addition and subtraction
  - Multiplication and division
  - Precision consideration
  - IEEE standard for binary floating-point arithmetic

William Stallings
Computer Organization and Architecture
10th Edition



Chapter 11
Digital Logic

Boolean Algebra

- Mathematical discipline used to design and analyze the behavior of the digital circuitry in digital computers and other digital systems
- Named after George Boole
  - English mathematician
  - Proposed basic principles of the algebra in 1854
- Claude Shannon suggested Boolean algebra could be used to solve problems in relay-switching circuit design
- Is a convenient tool:
  - Analysis
    - It is an economical way of describing the function of digital circuitry
  - Design
    - Given a desired function, Boolean algebra can be applied to develop a simplified implementation of that function

Boolean Variables and Operations

- Makes use of variables and operations
  - Are logical
  - A variable may take on the value 1 (TRUE) or 0 (FALSE)
  - Basic logical operations are AND, OR, and NOT
- AND
  - Yields true (binary value 1) if and only if both of its operands are true
  - In the absence of parentheses the AND operation takes precedence over the OR operation
  - When no ambiguity will occur the AND operation is represented by simple concatenation instead of the dot operator
- OR
  - Yields true if either or both of its operands are true
- NOT
  - Inverts the value of its operand

Table 11.1 Boolean Operators

(a) Boolean Operators of Two Input Variables

P	Q	NOT P ($\overline{P}$)	P AND Q ($P \cdot Q$)	P OR Q ($P + Q$)	P NAND Q ($\overline{P \cdot Q}$)	P NOR Q ($\overline{P + Q}$)	P XOR Q ($P \oplus Q$)
0	0	1	0	0	1	1	0
0	1	1	0	1	1	0	1
1	0	0	0	1	1	0	1
1	1	0	1	1	0	0	0

(b) Boolean Operators Extended to More than Two Inputs (A, B, . . .)

Operation	Expression	Output = 1 if
AND	$A \cdot B \cdot \ldots$	All of the set {A, B, …} are 1.
OR	$A + B + \ldots$	Any of the set {A, B, …} are 1.
NAND	$\overline{A \cdot B \cdot \ldots}$	Any of the set {A, B, …} are 0.
NOR	$\overline{A + B + \ldots}$	All of the set {A, B, …} are 0.
XOR	$A \oplus B \oplus \ldots$	The set {A, B, …} contains an odd number of ones.

Table 11.2
Basic Identities of Boolean Algebra

Basic Postulates		
$A \cdot B = B \cdot A$	$A + B = B + A$	Commutative Laws
$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$	$A + (B \cdot C) = (A + B) \cdot (A + C)$	Distributive Laws
$1 \cdot A = A$	$0 + A = A$	Identity Elements
$A \cdot \overline{A} = 0$	$A + \overline{A} = 1$	Inverse Elements
Other Identities		
$0 \cdot A = 0$	$1 + A = 1$	
$A \cdot A = A$	$A + A = A$	
$A \cdot (B \cdot C) = (A \cdot B) \cdot C$	$A + (B + C) = (A + B) + C$	Associative Laws
$\overline{A \cdot B} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A} \cdot \overline{B}$	DeMorgan's Theorem

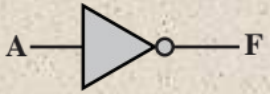
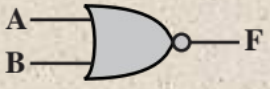
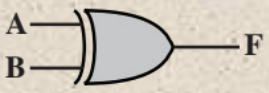
Name	Graphical Symbol	Algebraic Function	Truth Table
AND		$F = A \cdot B$ or $F = AB$	A B | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$F = A + B$	A B | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
NOT		$F = \overline{A}$ or $F = A'$	A | F 0 | 1 1 | 0
NAND		$F = \overline{AB}$	A B | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$F = \overline{A + B}$	A B | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
XOR		$F = A \oplus B$	A B | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0

Figure 11.1 Basic Logic Gates

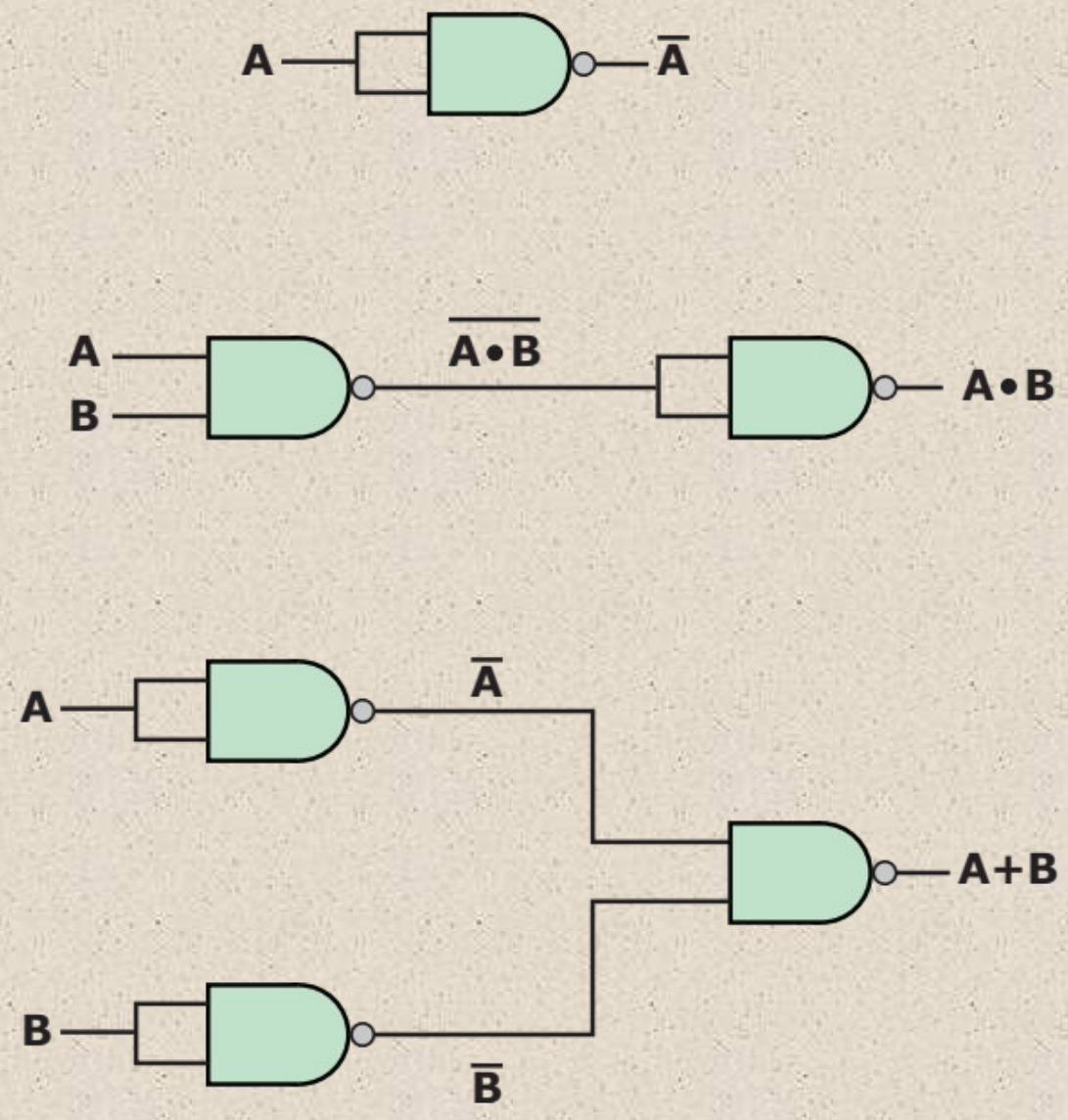


Figure 11.2 Some Uses of NAND Gates

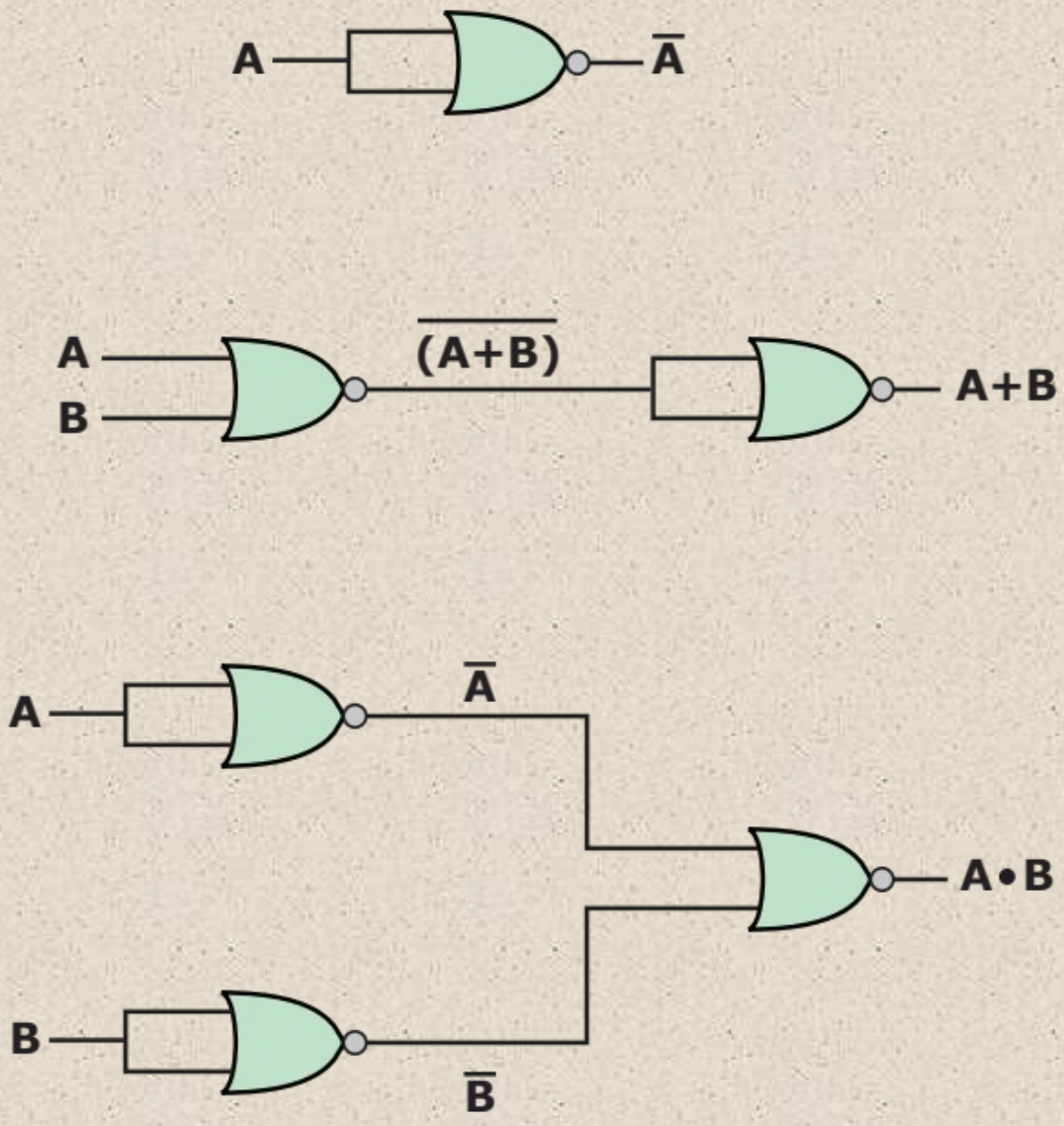
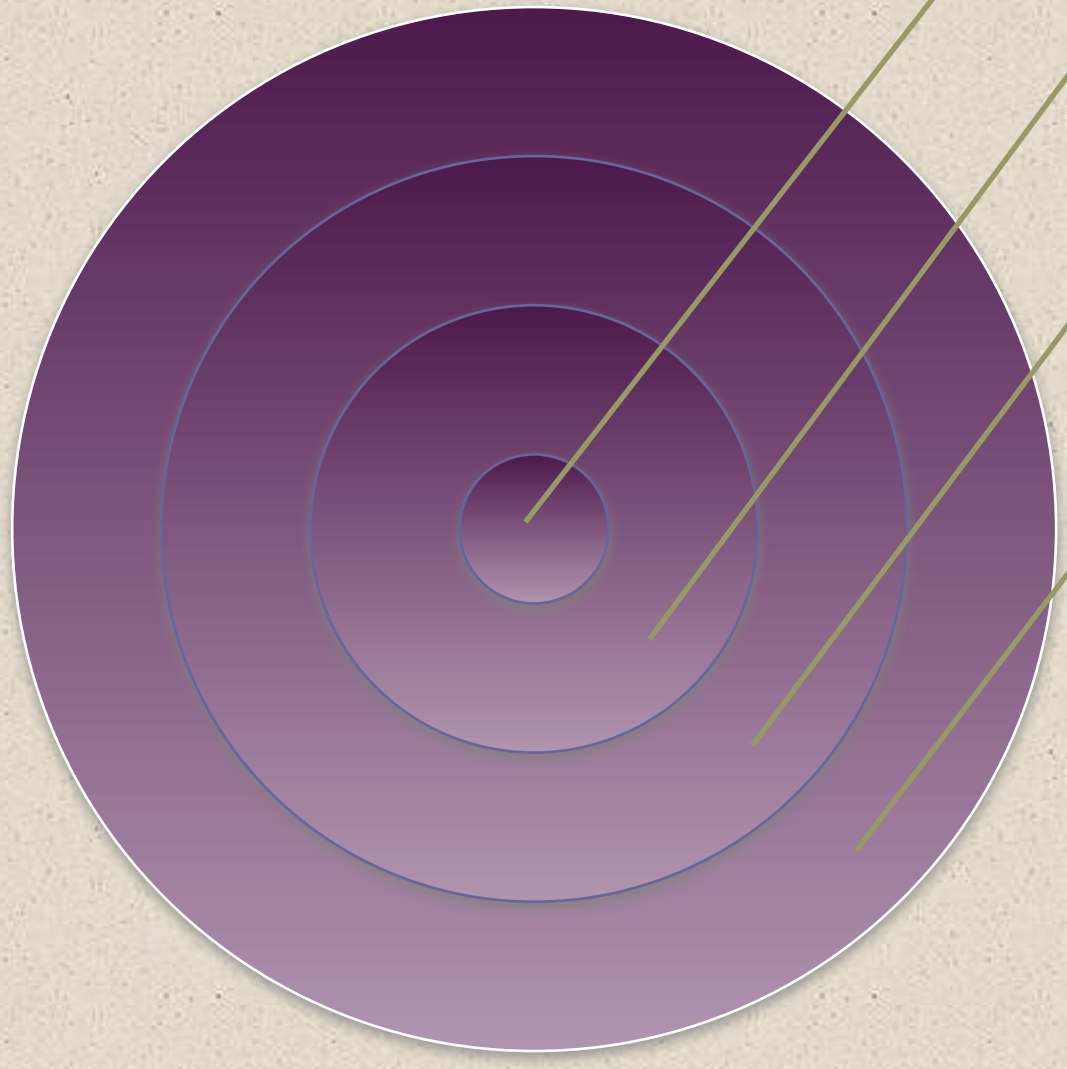


Figure 11.3 Some Uses of NOR Gates

Combinational Circuit



An interconnected set of gates whose output at any time is a function only of the input at that time

The appearance of the input is followed almost immediately by the appearance of the output, with only gate delays

Consists of *n* binary inputs and *m* binary outputs

Can be defined in three ways:

- Truth table
  - For each of the 2^n possible combinations of input signals, the binary value of each of the *m* output signals is listed
- Graphical symbols
  - The interconnected layout of gates is depicted
- Boolean equations
  - Each output signal is expressed as a Boolean function of its input signals

Table 11.3
A Boolean Function of Three Variables

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

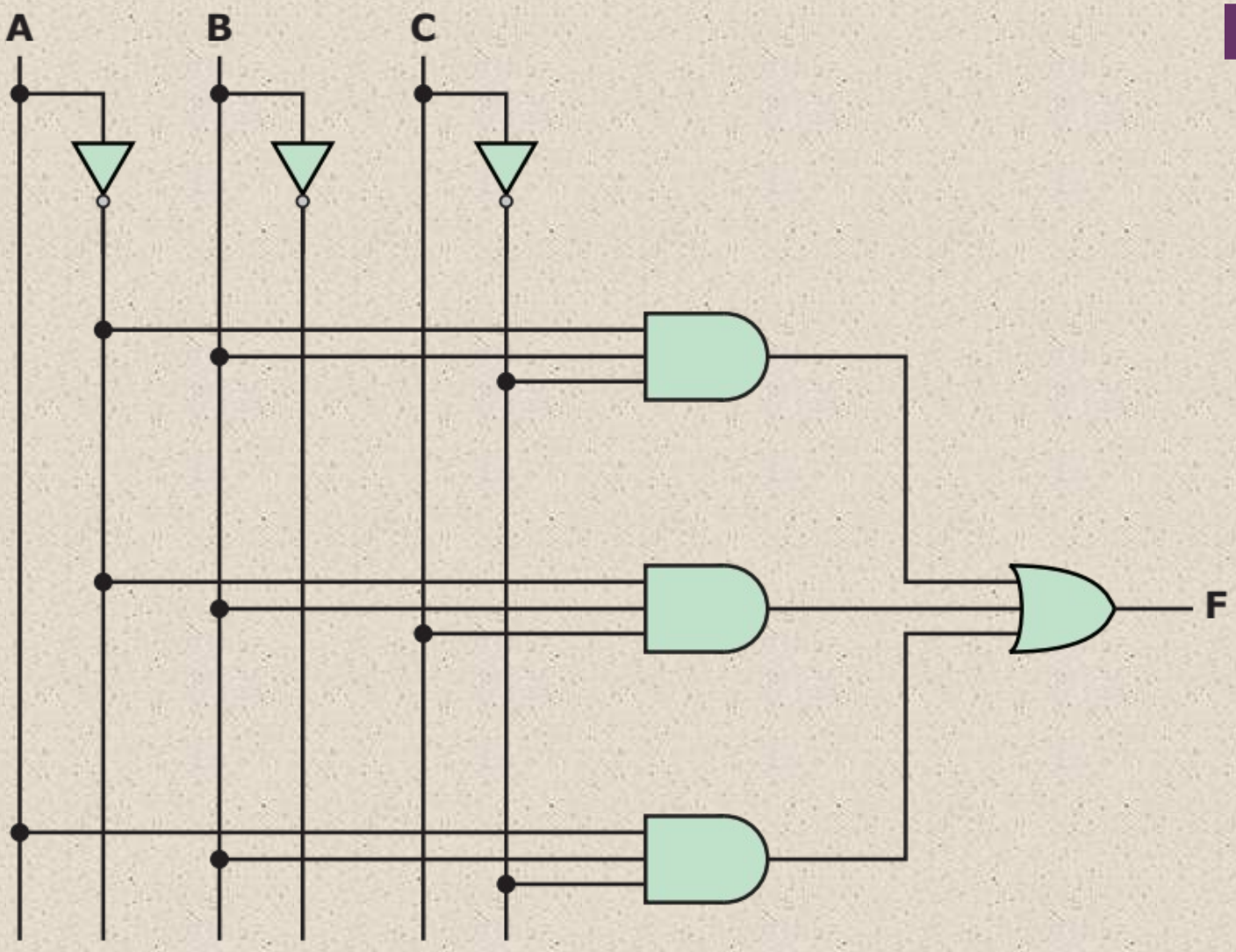


Figure 11.4 Sum-of-Products Implementation of Table 11.3

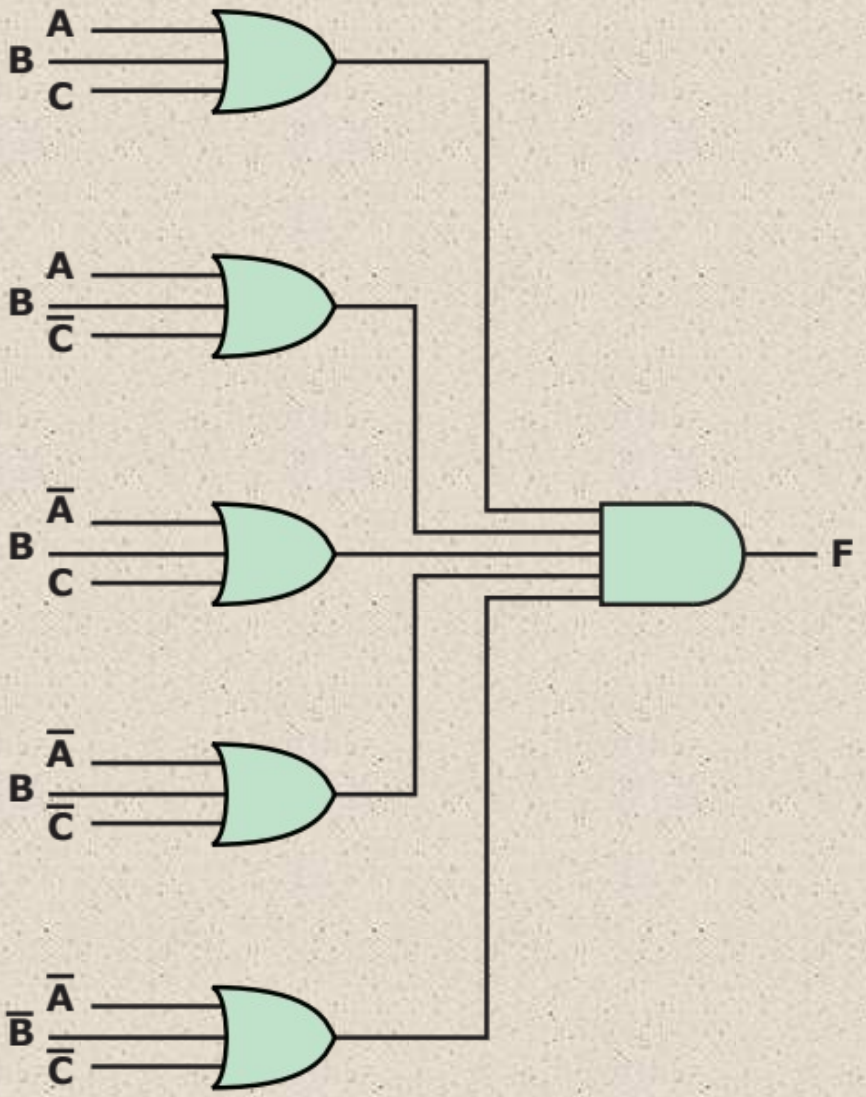


Figure 11.5 Product-of-Sums Implementation of Table 11.3

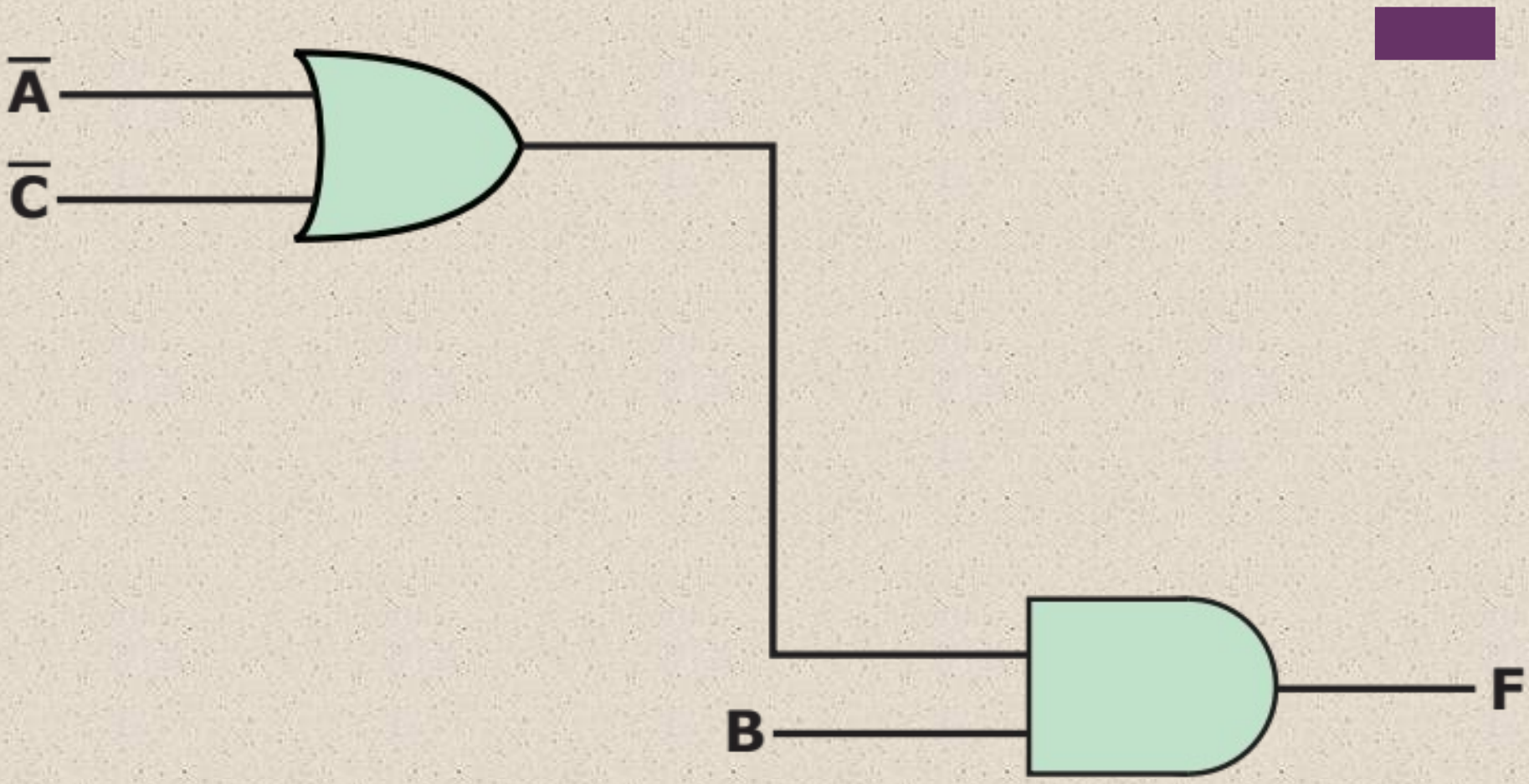


Figure 11.6 Simplified Implementation of Table 11.3

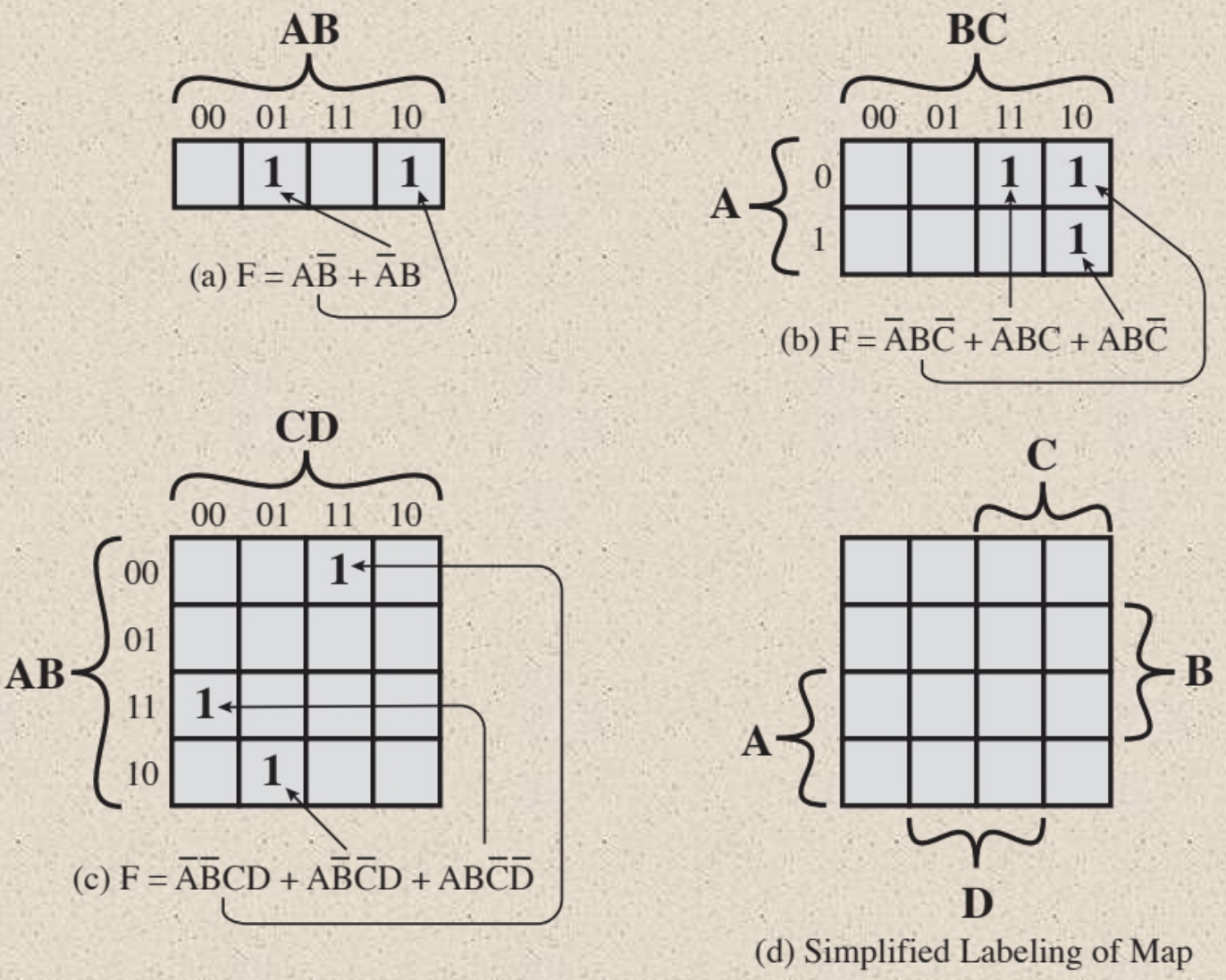


Figure 11.7 The Use of Karnaugh Maps to Represent Boolean Functions

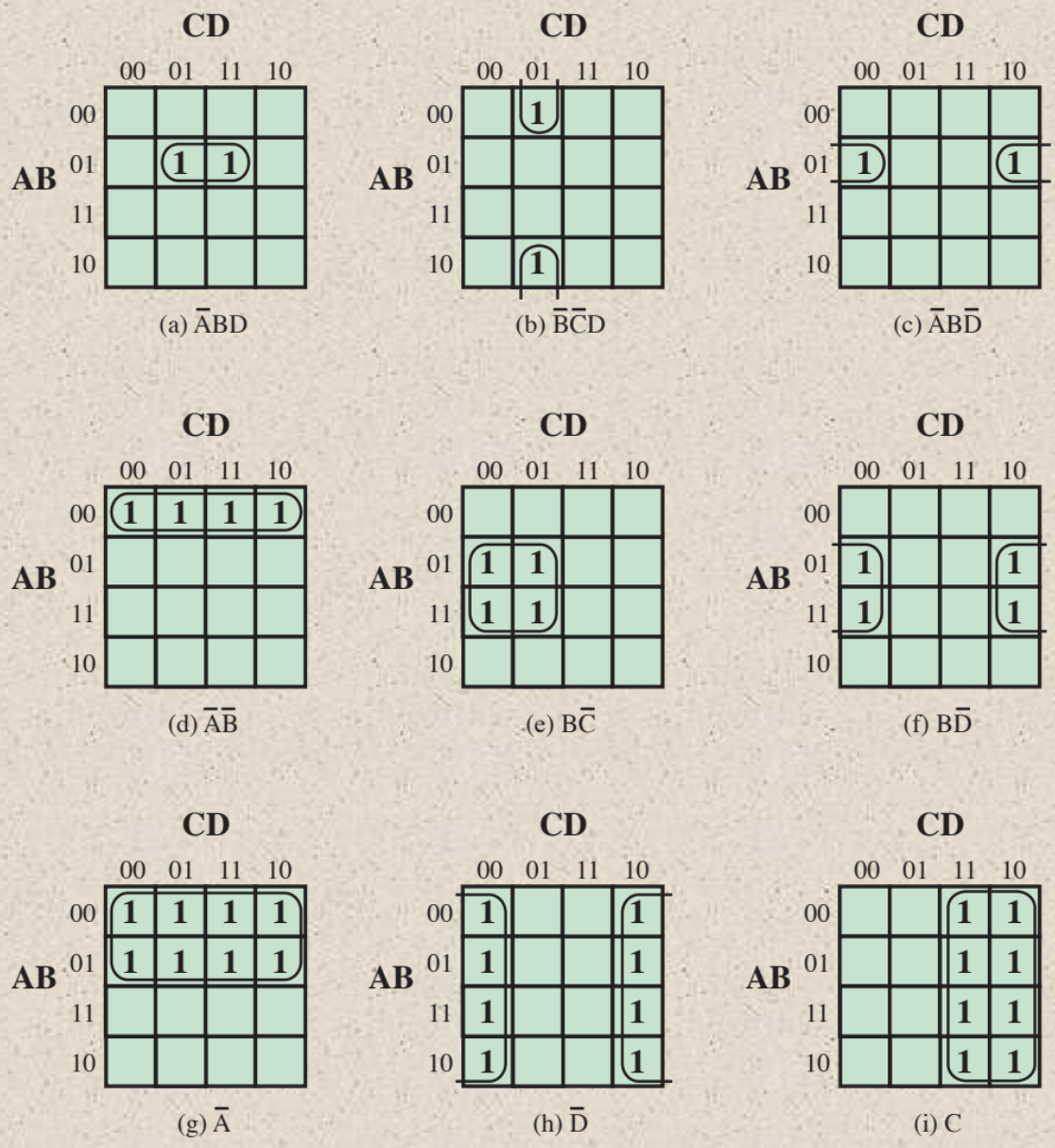
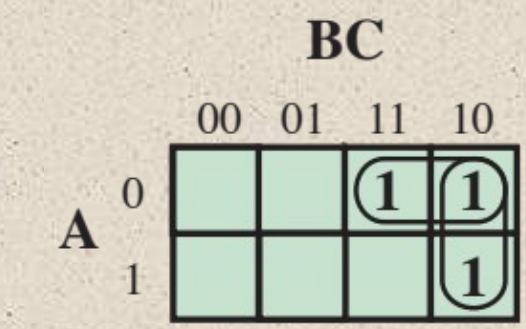
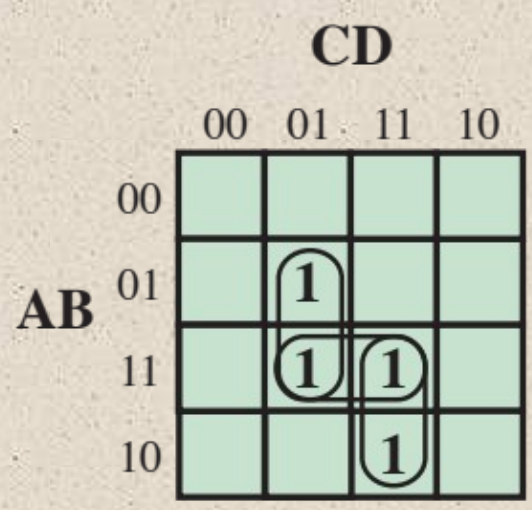


Figure 11.8 Example Use of Karnaugh Maps



(a) $F = \bar{A}B + B\bar{C}$



(b) $F = B\bar{C}D + ACD$

Figure 11.9 Overlapping Groups

Table 11.4
Truth Table for the One-Digit Packed Decimal Incrementer

	Input					Output			
Number	A	B	C	D	Number	W	X	Y	Z
0	0	0	0	0	1	0	0	0	1
1	0	0	0	1	2	0	0	1	0
2	0	0	1	0	3	0	0	1	1
3	0	0	1	1	4	0	1	0	0
4	0	1	0	0	5	0	1	0	1
5	0	1	0	1	6	0	1	1	0
6	0	1	1	0	7	0	1	1	1
7	0	1	1	1	8	1	0	0	0
8	1	0	0	0	9	1	0	0	1
9	1	0	0	1	0	0	0	0	0
Don't care condition	1	0	1	0		d	d	d	d
	1	0	1	1		d	d	d	d
	1	1	0	0		d	d	d	d
	1	1	0	1		d	d	d	d
	1	1	1	0		d	d	d	d
	1	1	1	1		d	d	d	d

Table 11.4 Truth Table for the One-Digit Packed Decimal Incrementer

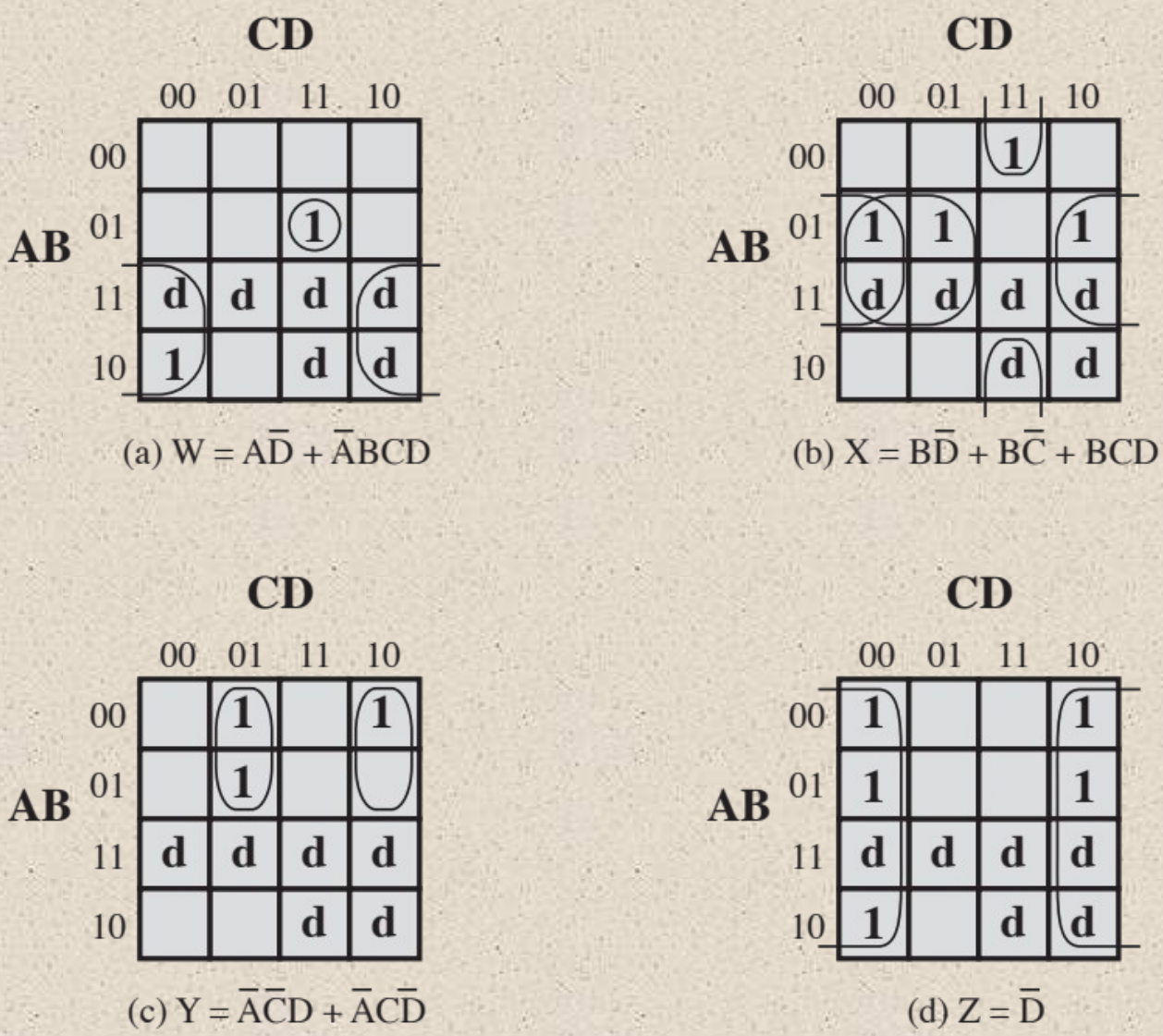


Figure 11.10 Karnaugh Maps for the Incrementer

Table 11.5
First Stage of Quine-McCluskey Method

(for F = ABCD + AB D + AB + A CD + BCD + BC + B D + D)

Product Term	Index	A	B	C	D	
$\bar{A}\bar{B}\bar{C}D$	1	0	0	0	1	✓
$\bar{A}B\bar{C}D$	5	0	1	0	1	✓
$\bar{A}BC\bar{D}$	6	0	1	1	0	✓
$AB\bar{C}\bar{D}$	12	1	1	0	0	✓
$\bar{A}BCD$	7	0	1	1	1	✓
$A\bar{B}CD$	11	1	0	1	1	✓
$AB\bar{C}D$	13	1	1	0	1	✓
$ABCD$	15	1	1	1	1	✓

Table 11.6
Last Stage of Quine-McCluskey Method

(for F = ABCD + AB D + AB + A CD + BCD + BC + B D + D)

	$ABCD$	$AB\bar{C}D$	$AB\bar{C}\bar{D}$	$A\bar{B}CD$	$\bar{A}BCD$	$\bar{A}BC\bar{D}$	$\bar{A}B\bar{C}D$	$\square\square\square D$
BD	X	X			X		X	
$\bar{A}\bar{C}D$							[X]	⊗
$\bar{A}BC$					[X]	⊗		
$AB\bar{C}$		[X]	⊗					
ACD	[X]			⊗				

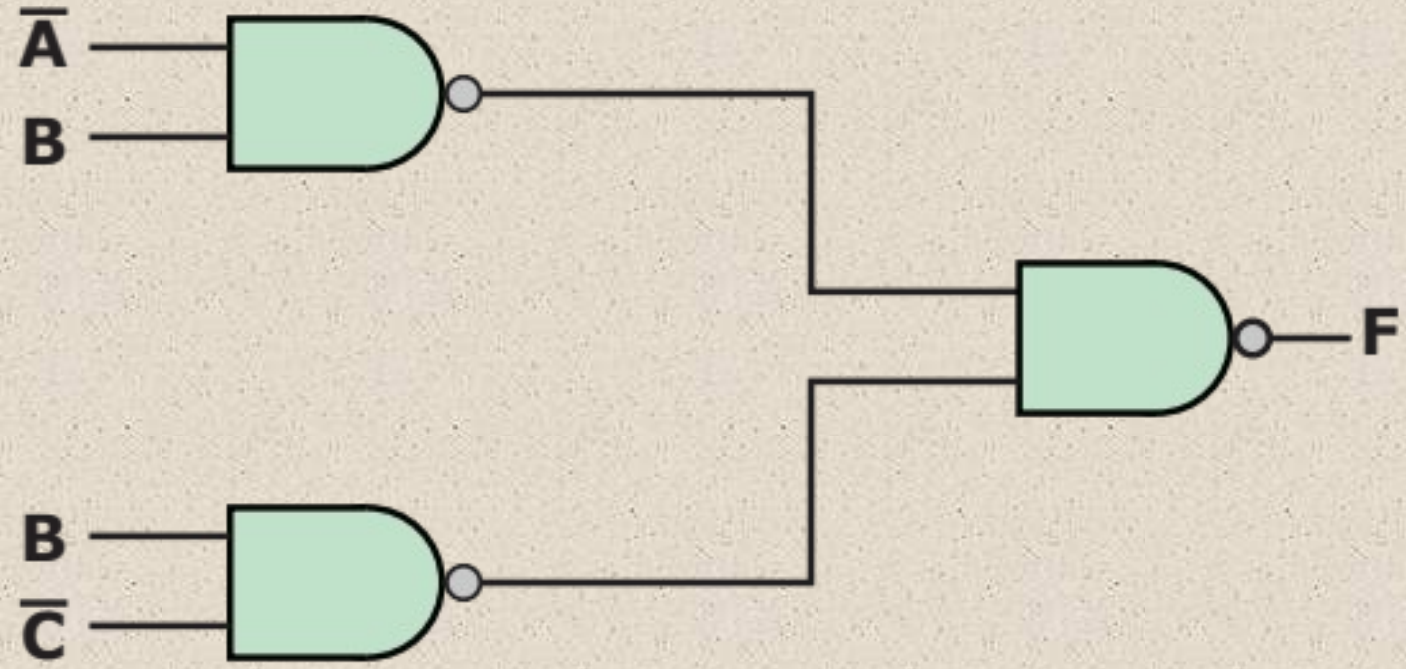


Figure 11.11 NAND Implementation of Table 11.3

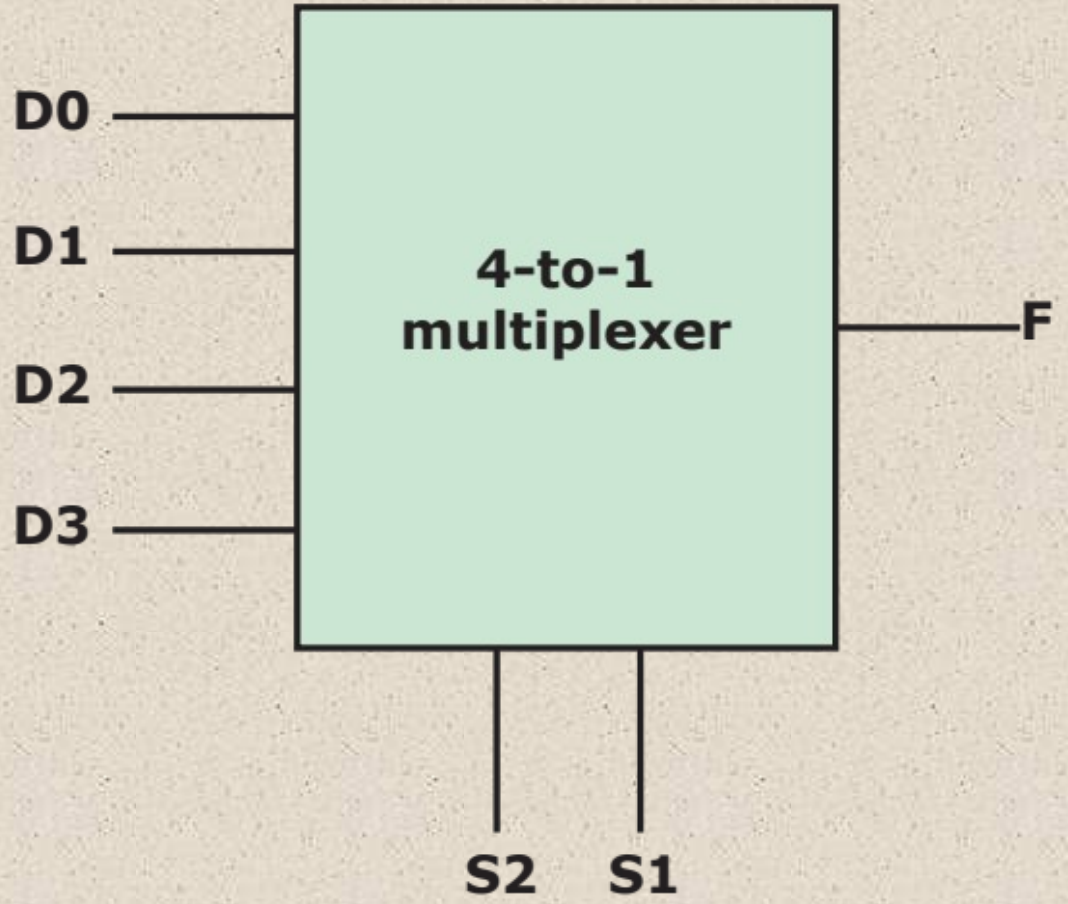


Figure 11.12 4-to-1 Multiplexer Representation

Table 11.7
4-to-1 Multiplexer Truth Table

S2	S1	F
0	0	D0
0	1	D1
1	0	D2
1	1	D3

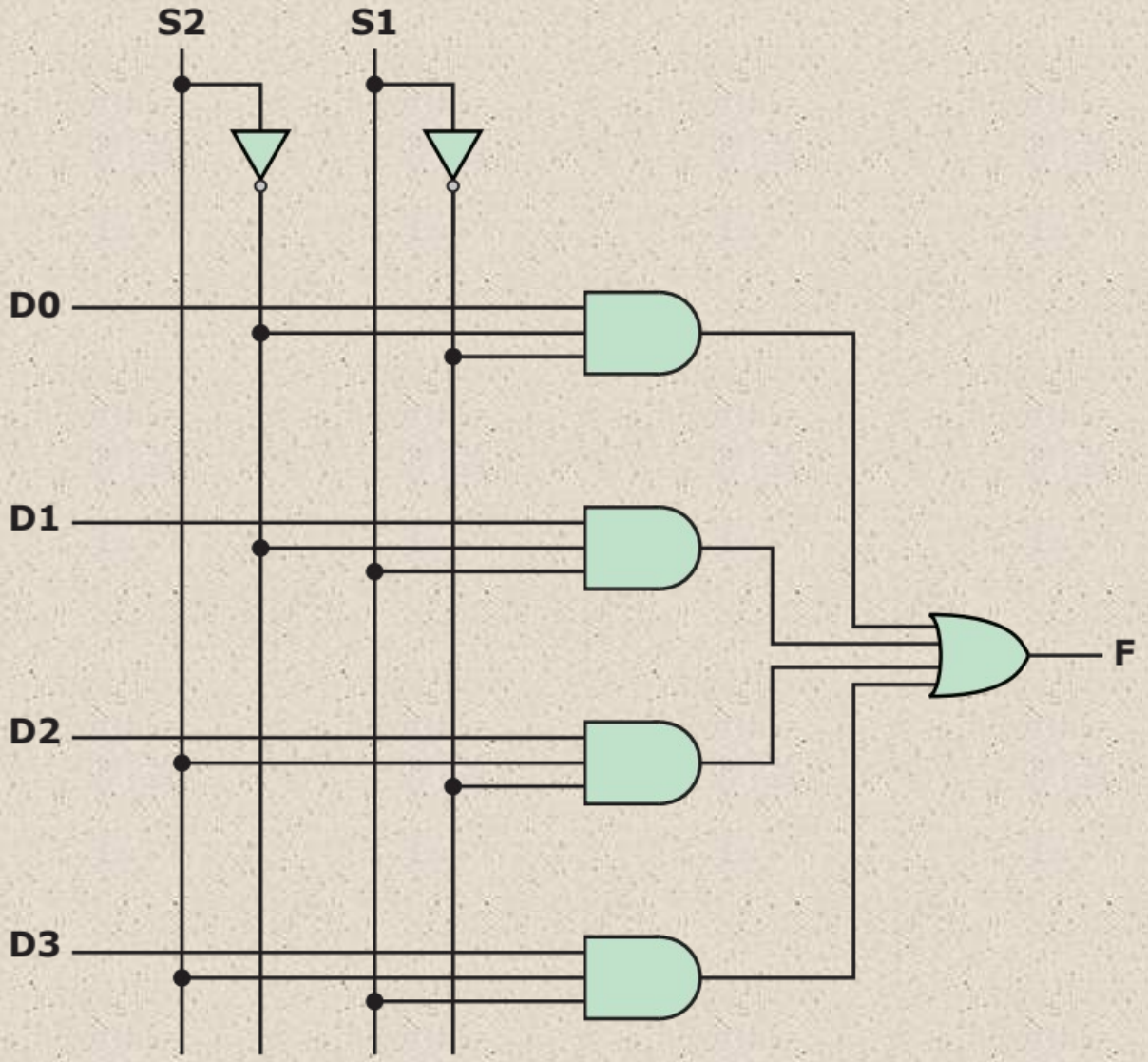


Figure 11.13 Multiplexer Implementation

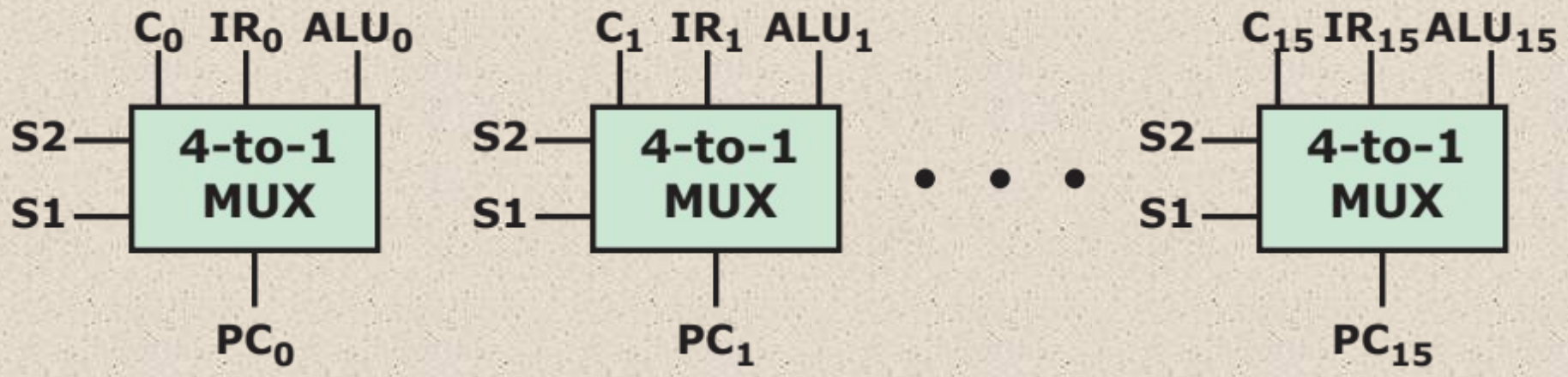


Figure 11.14 Multiplexer Input to Program Counter

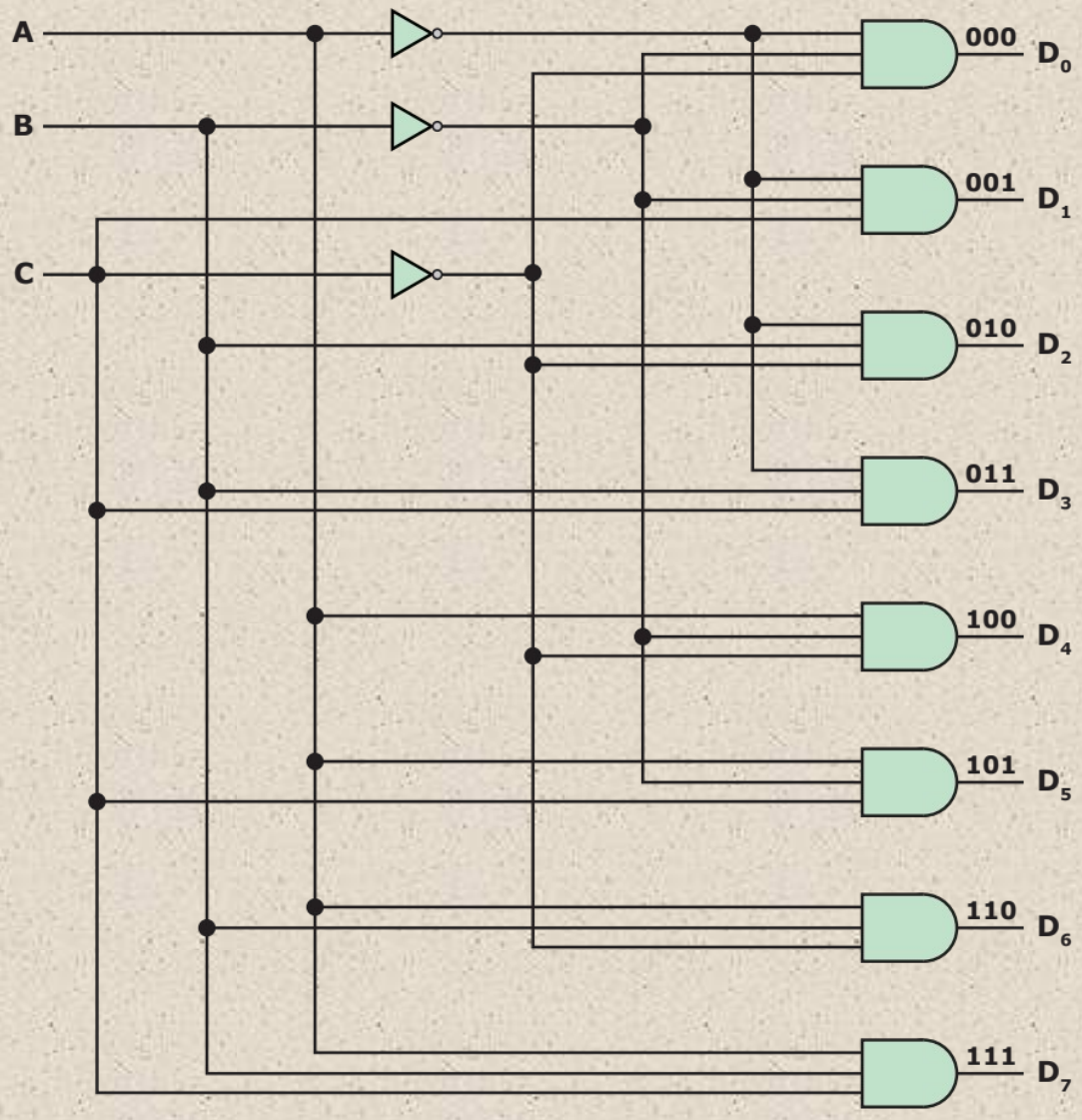


Figure 11.15 Decoder with 3 Inputs and $2^3 = 8$ Outputs

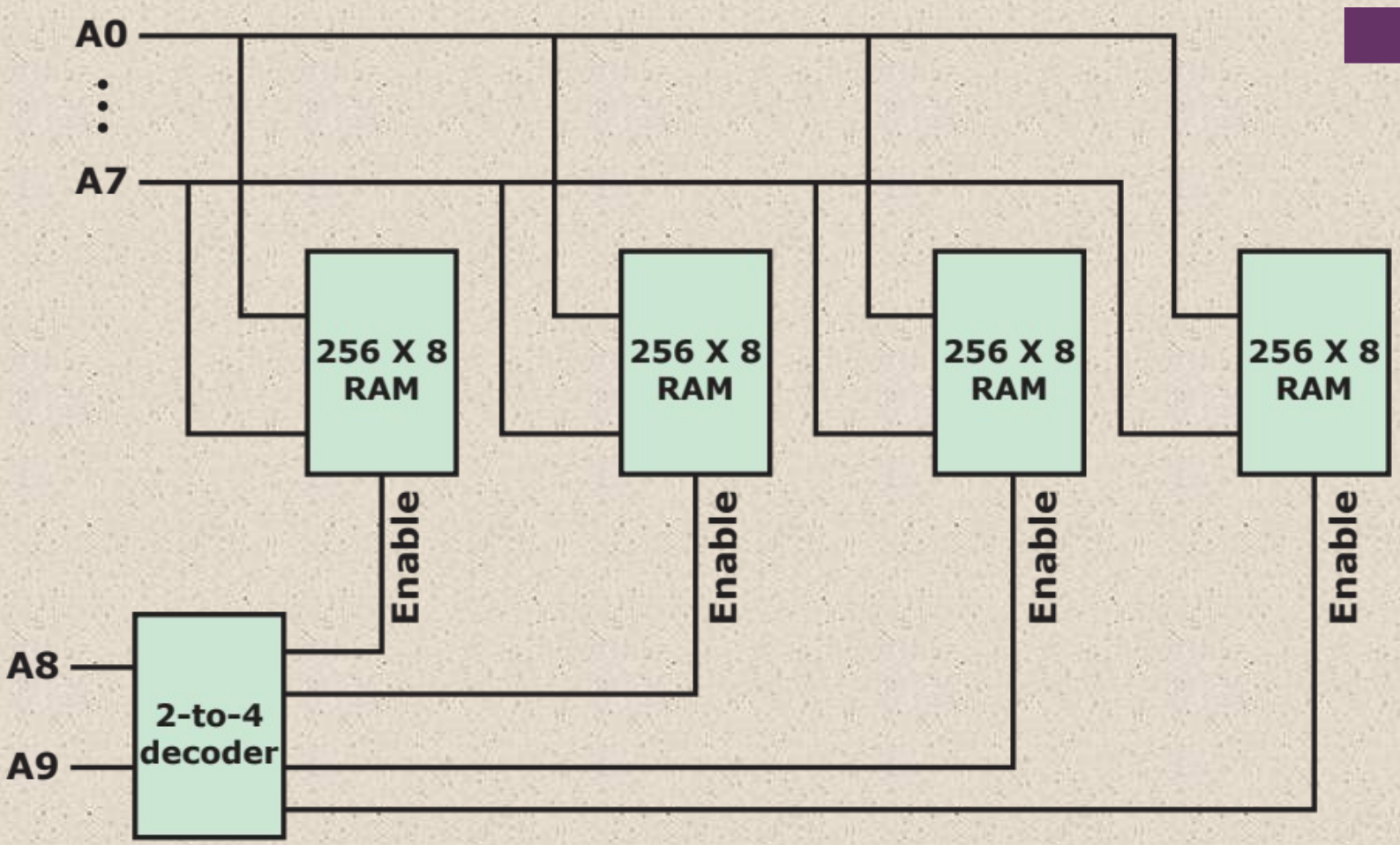


Figure 11.16 Address Decoding

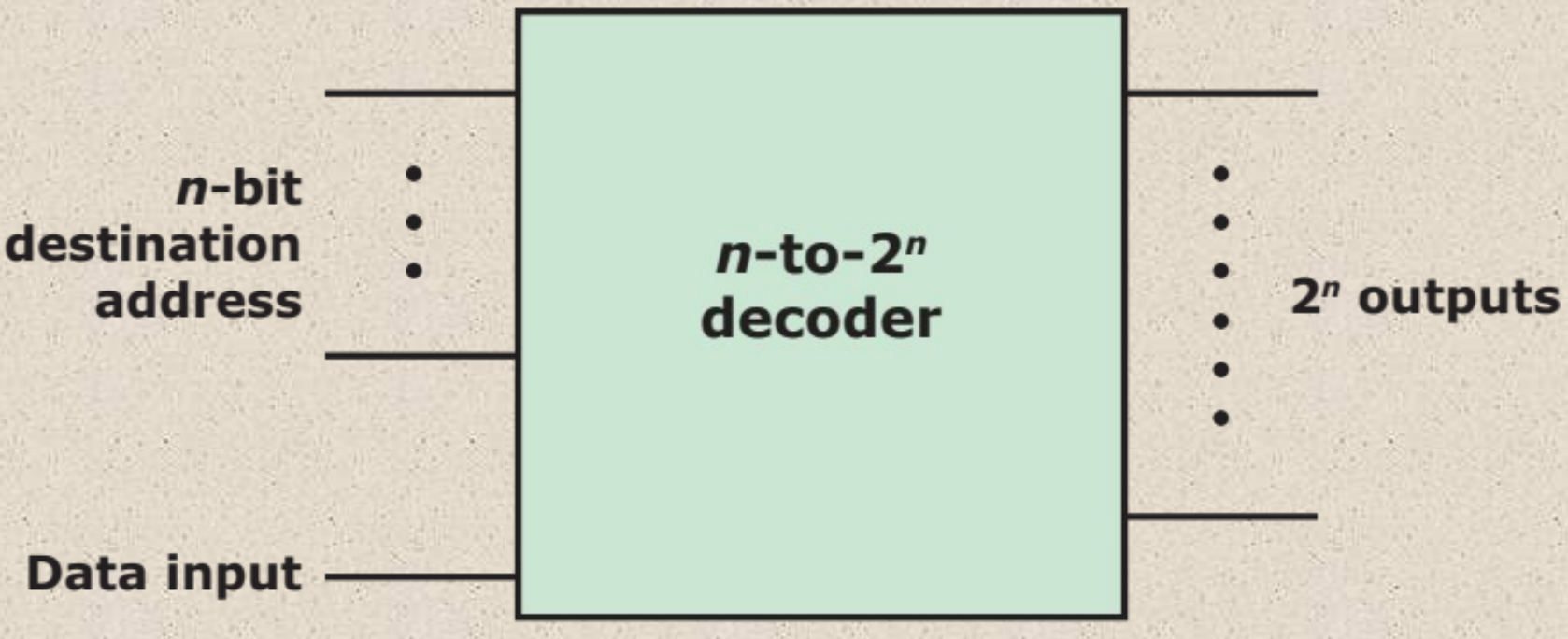


Figure 11.17 Implementation of a Demultiplexer Using a Decoder

Read-Only Memory (ROM)

- Memory that is implemented with combinational circuits
  - Combinational circuits are often referred to as “memoryless” circuits because their output depends only on their current input and no history of prior inputs is retained
- Memory unit that performs only the read operation
  - Binary information stored in a ROM is permanent and is created during the fabrication process
  - A given input to the ROM (address lines) always produces the same output (data lines)
  - Because the outputs are a function only of the present inputs, ROM is a combinational circuit

Table 11.8
Truth Table for a ROM

Input				Output			
X_1	X_2	X_3	X_4	Z_1	Z_2	Z_3	Z_4
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

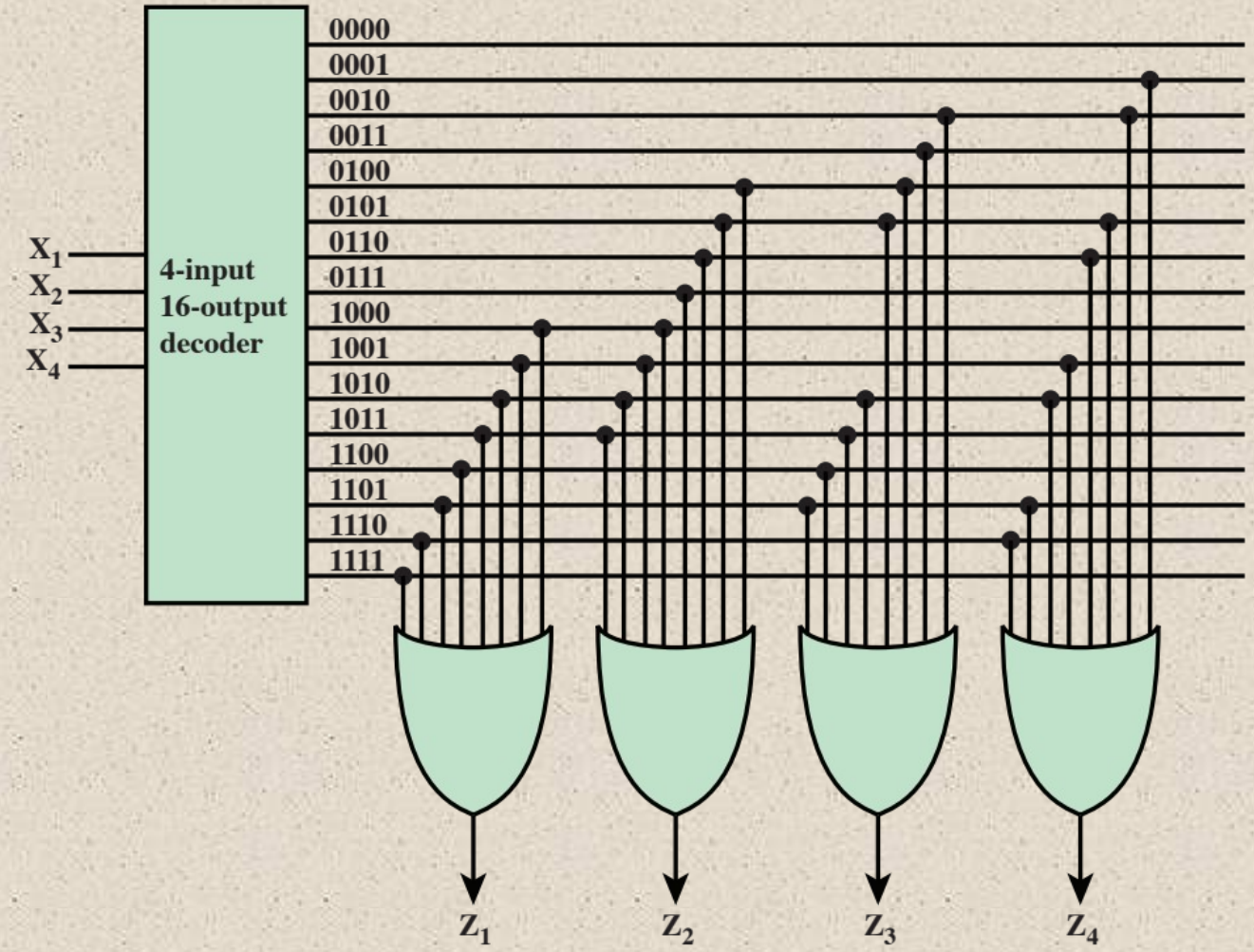


Figure 11.18 A 64-Bit ROM

Table 11.9
Binary Addition Truth Tables

(a) Single-Bit Addition

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

(b) Addition with Carry Input

C_{in}	A	B	Sum	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

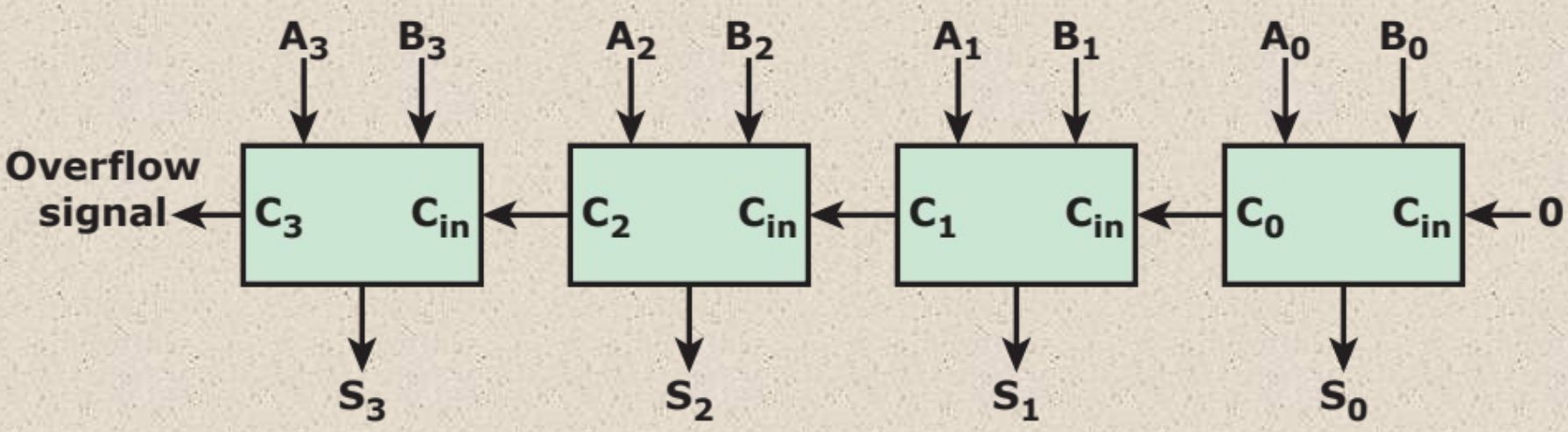


Figure 11.19 4-Bit Adder

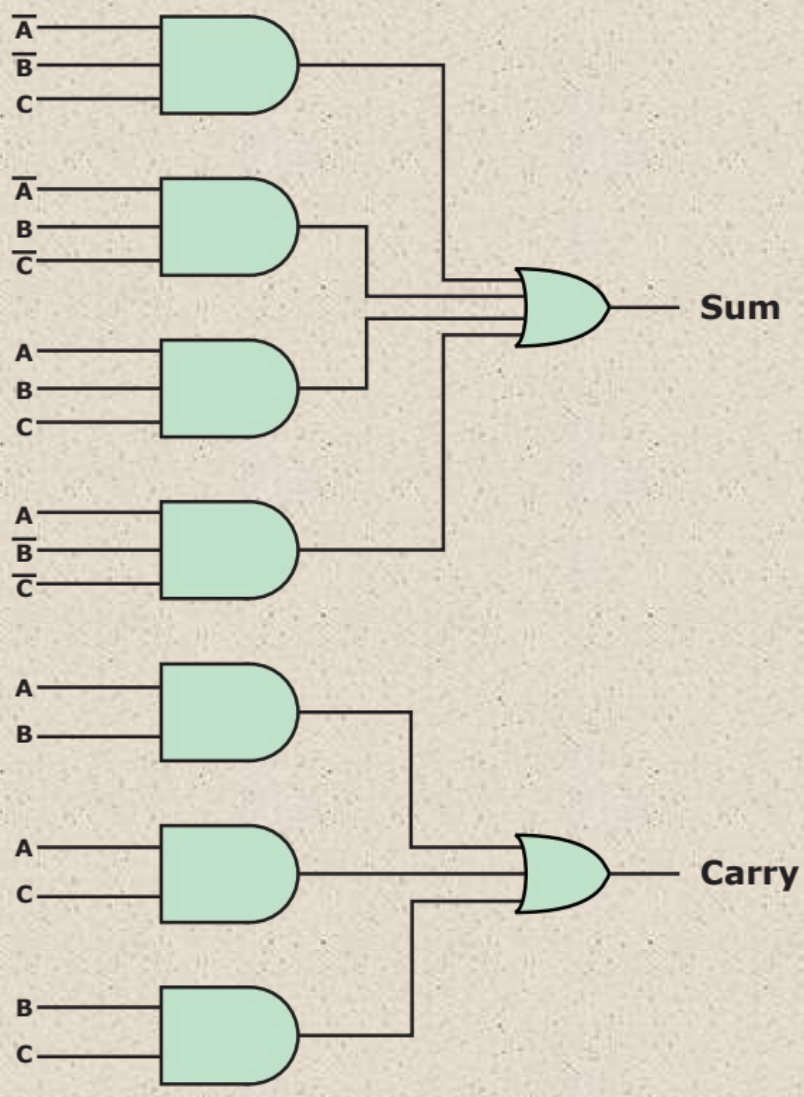


Figure 11.20 Implementation of an Adder

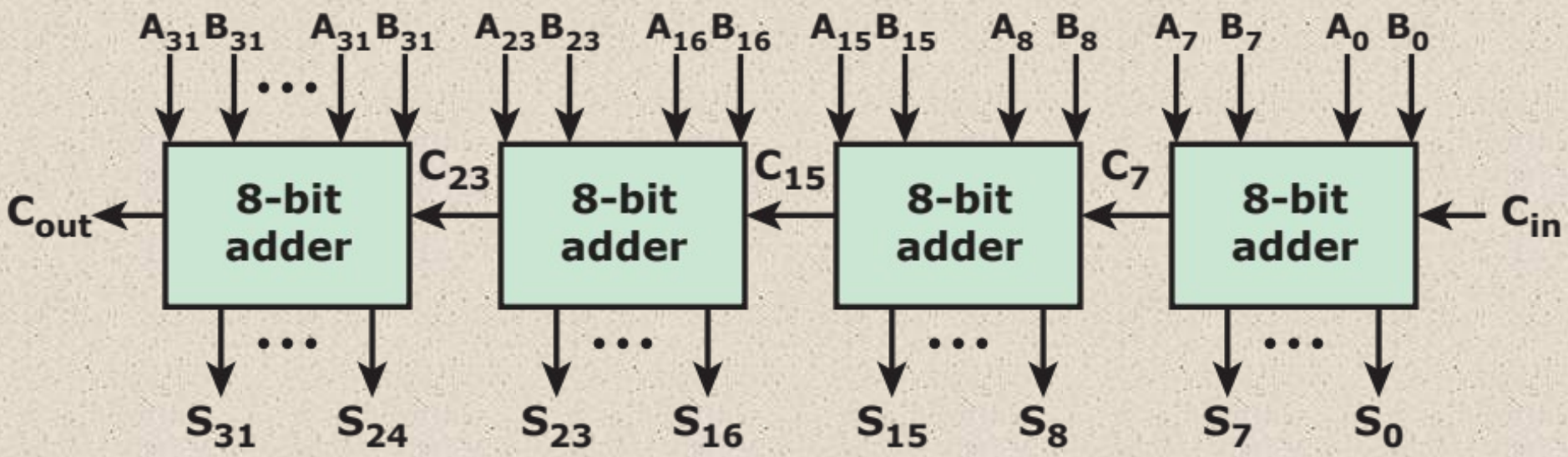
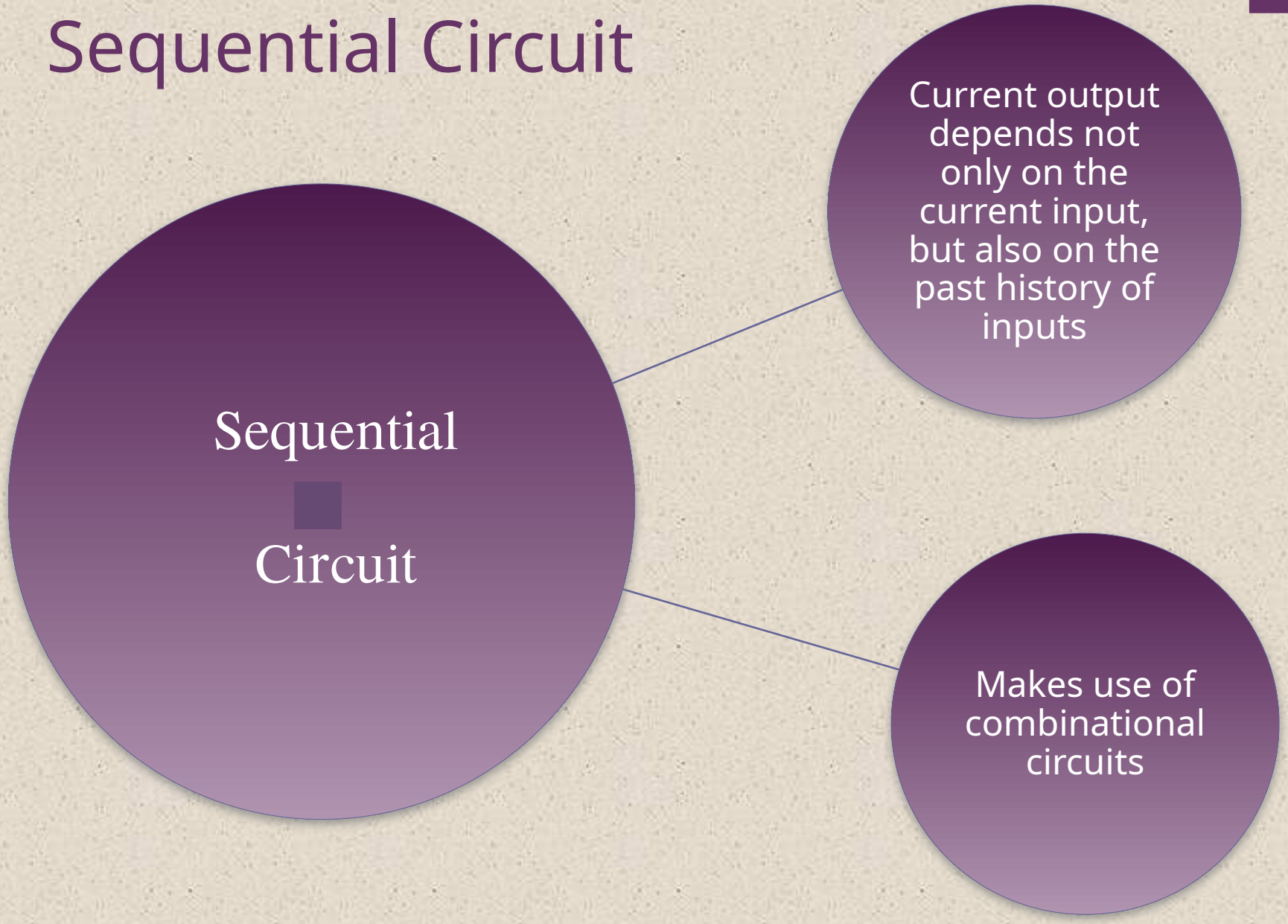


Figure 11.21 Construction of a 32-Bit Adder Using 8-Bit Adders

Sequential Circuit



Flip-Flops

- Simplest form of sequential circuit
- There are a variety of flip-flops, all of which share two properties:

1. The flip-flop is a bistable device. It exists in one of two states and, in the absence of input, remains in that state. Thus, the flip-flop can function as a 1-bit memory.
2. The flip-flop has two outputs, which are always the complements of each other.

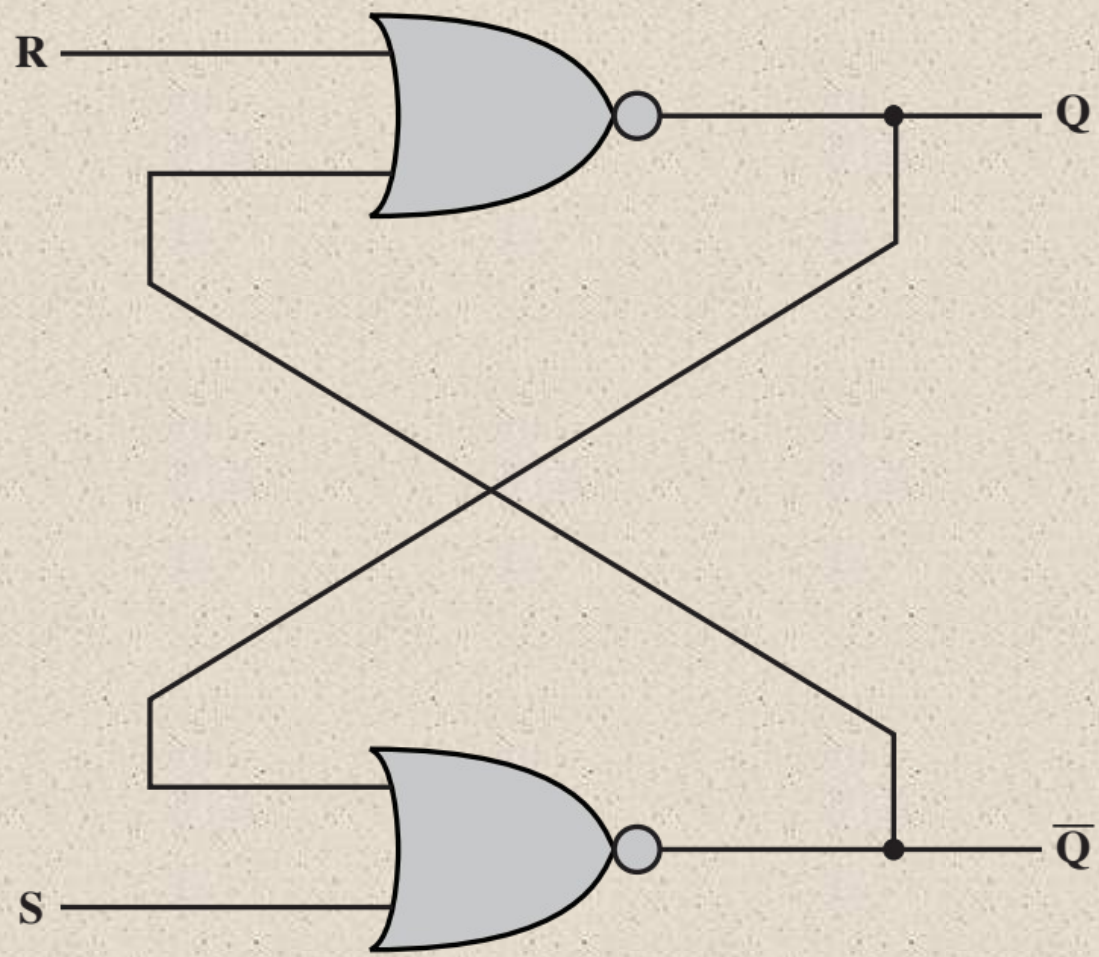


Figure 11.22 The S-R Latch Implemented with NOR Gates

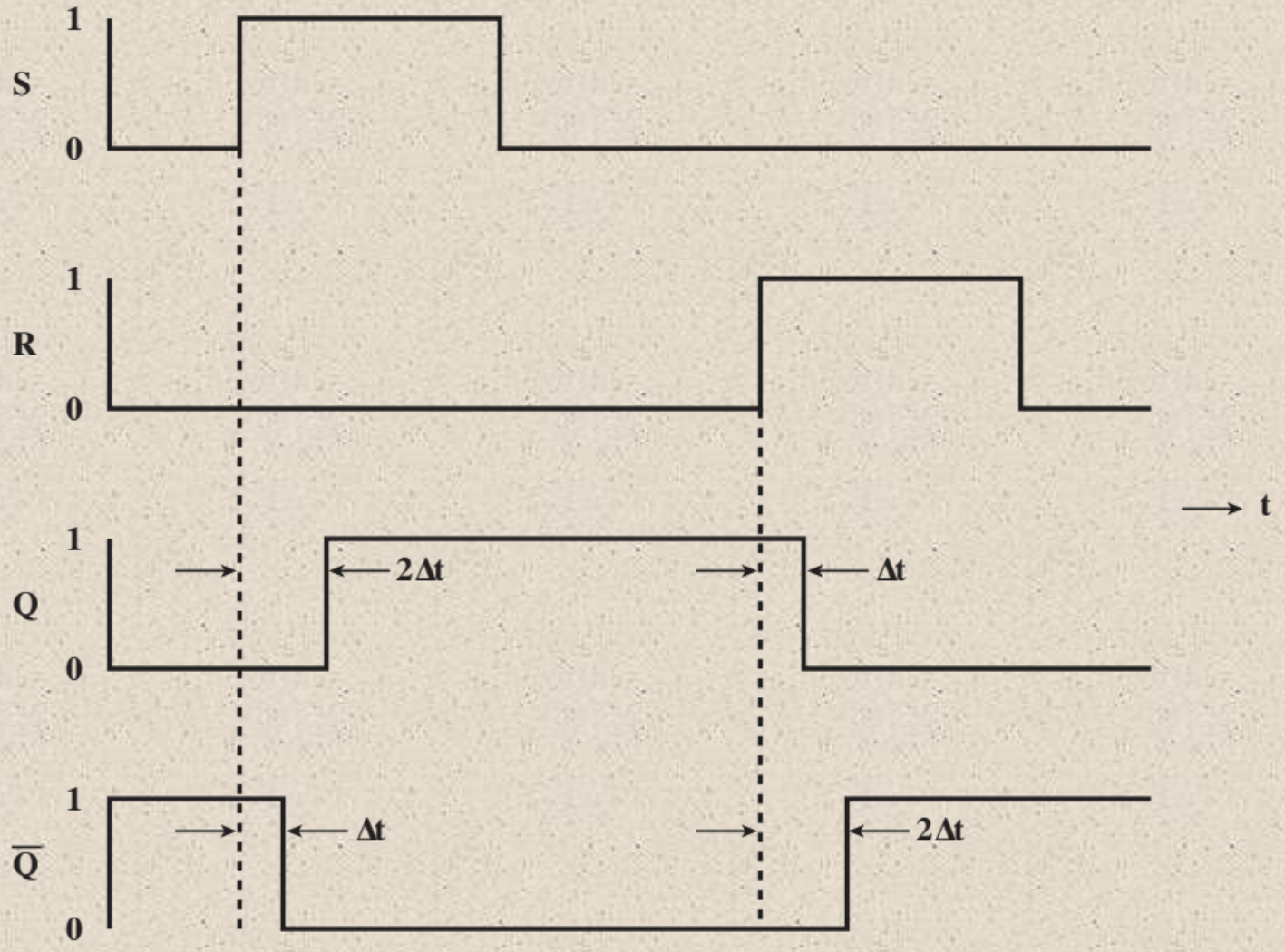


Figure 11.23 NOR S-R Latch Timing Diagram

Table 11.10 The S-R Latch

(a) Characteristic Table

Current Inputs SR	Current State Q_n	Next State Q_{n+1}
00	0	0
00	1	1
01	0	0
01	1	0
10	0	1
10	1	1
11	0	—
11	1	—

(b) Simplified Characteristic Table

S	R	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	—

(c) Response to Series of Inputs

t	0	1	2	3	4	5	6	7	8	9
S	1	0	0	0	0	0	0	0	1	0
R	0	0	0	1	0	0	1	0	0	0
Q_{n+1}	1	1	1	0	0	0	0	0	1	1

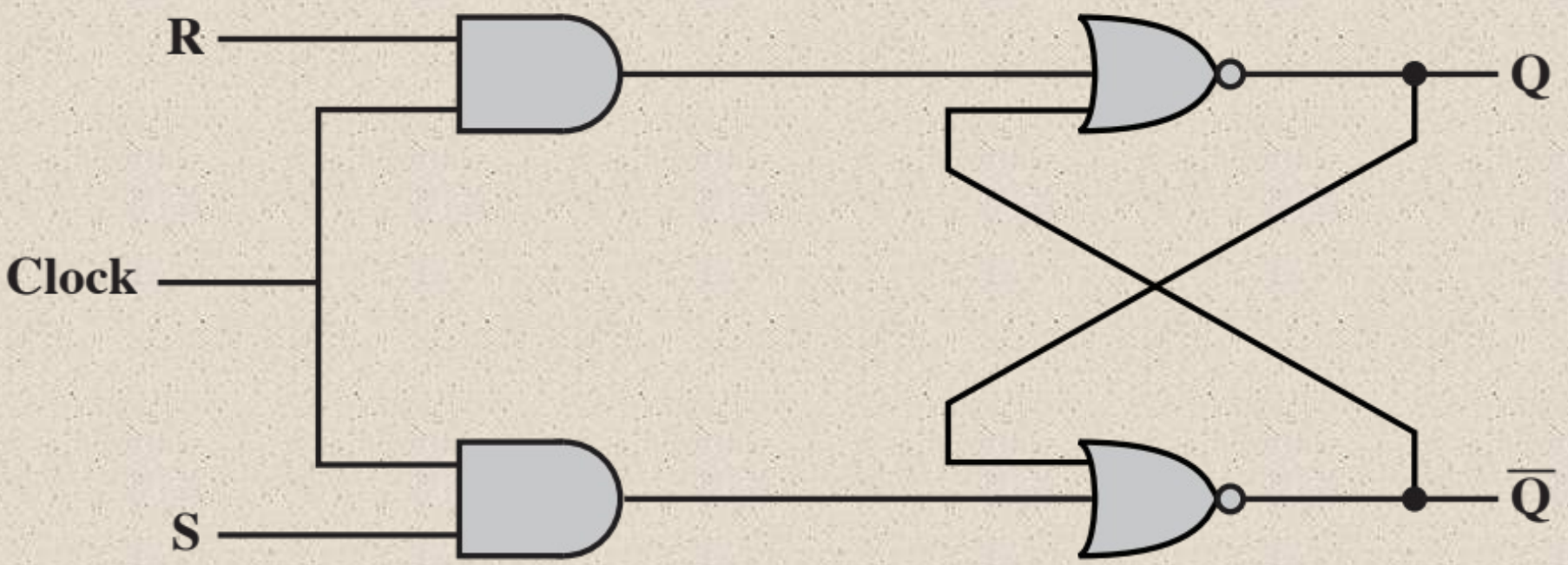


Figure 11.24 Clocked S-R Flip Flop

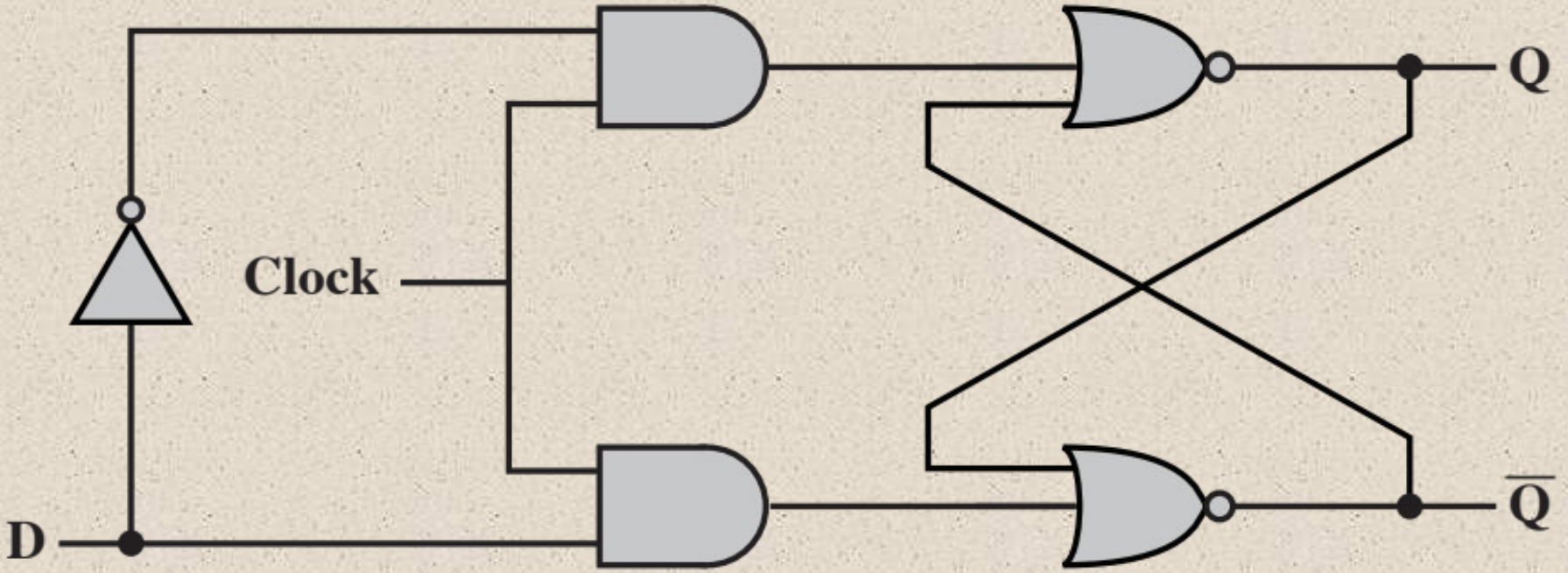


Figure 11.25 D Flip Flop

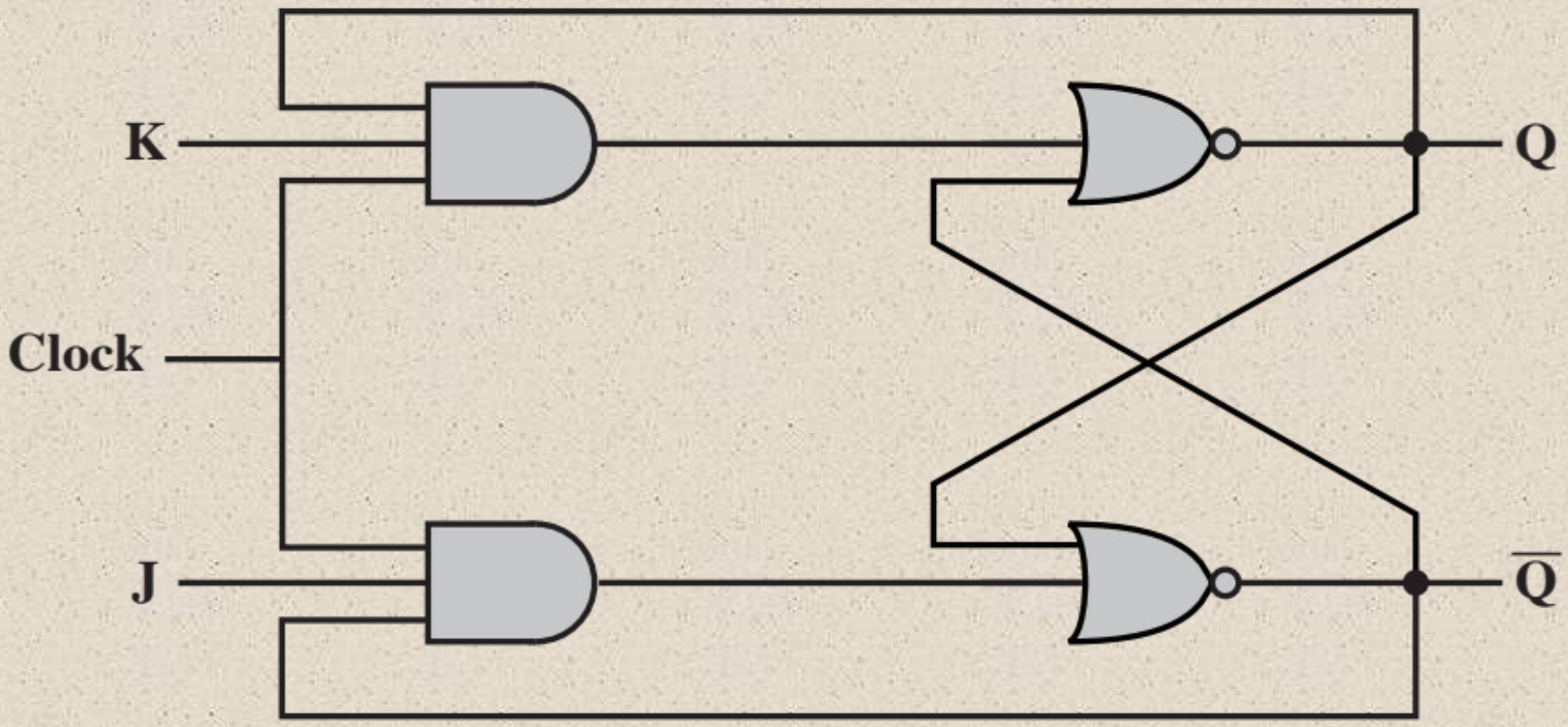


Figure 11.26 J-K Flip Flop

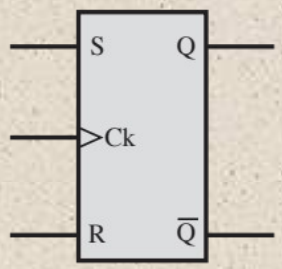
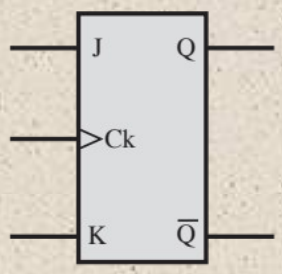
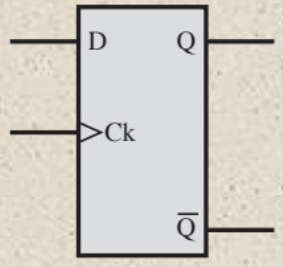
Name	Graphical Symbol	Truth Table
S-R		S R Q_{n+1} 0 0 Q_n 0 1 0 1 0 1 1 1 –
J-K		J K Q_{n+1} 0 0 Q_n 0 1 0 1 0 1 1 1 $\overline{Q_n}$
D		D Q_{n+1} 0 0 1 1

Figure 11.27 Basic Flip-Flops

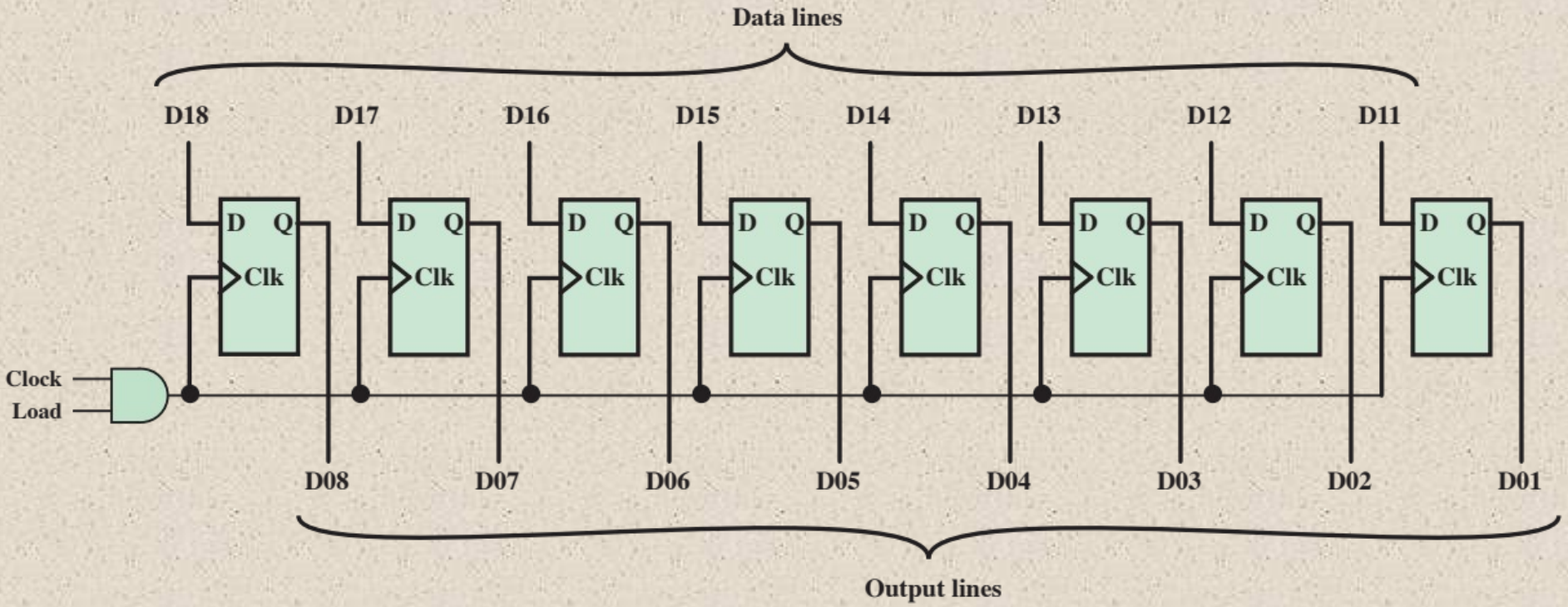


Figure 11.28 8-Bit Parallel Register

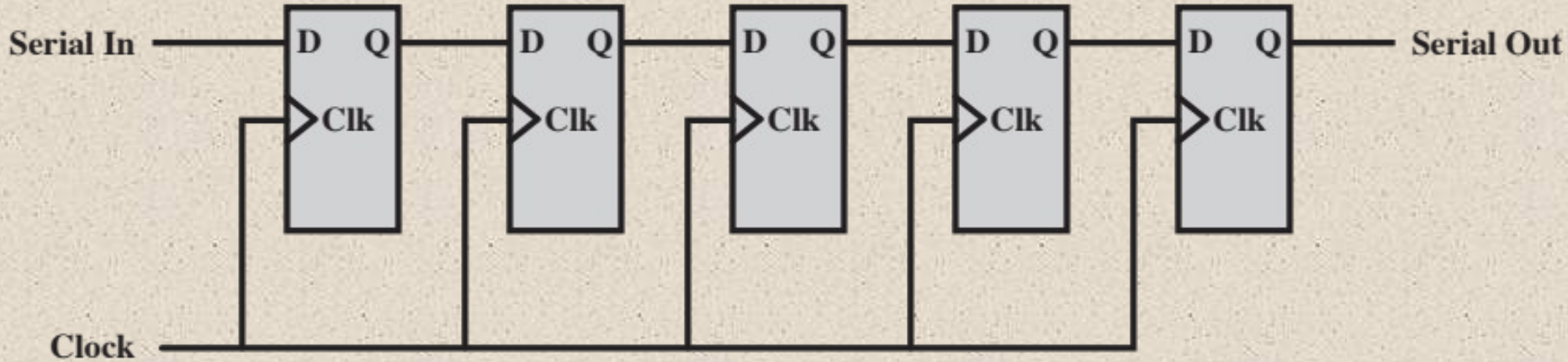


Figure 11.29 5-Bit Shift Register

Counter

- A register whose value is easily incremented by 1 modulo the capacity of the register
- After the maximum value is achieved the next increment sets the counter value to 0
- An example of a counter in the CPU is the program counter
- Can be designated as:
  - Asynchronous
    - Relatively slow because the output of one flip-flop triggers a change in the status of the next flip-flop
  - Synchronous
    - All of the flip-flops change state at the same time
    - Because it is faster it is the kind used in CPUs

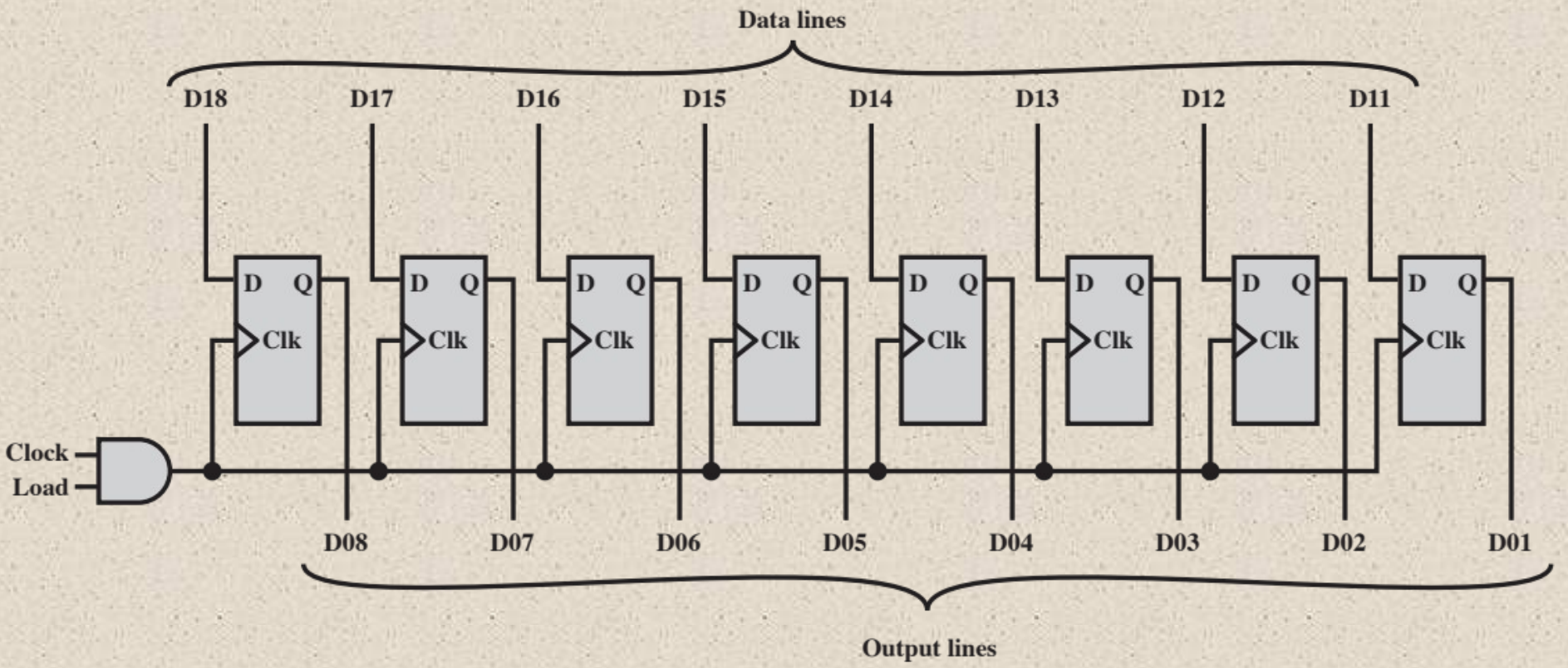
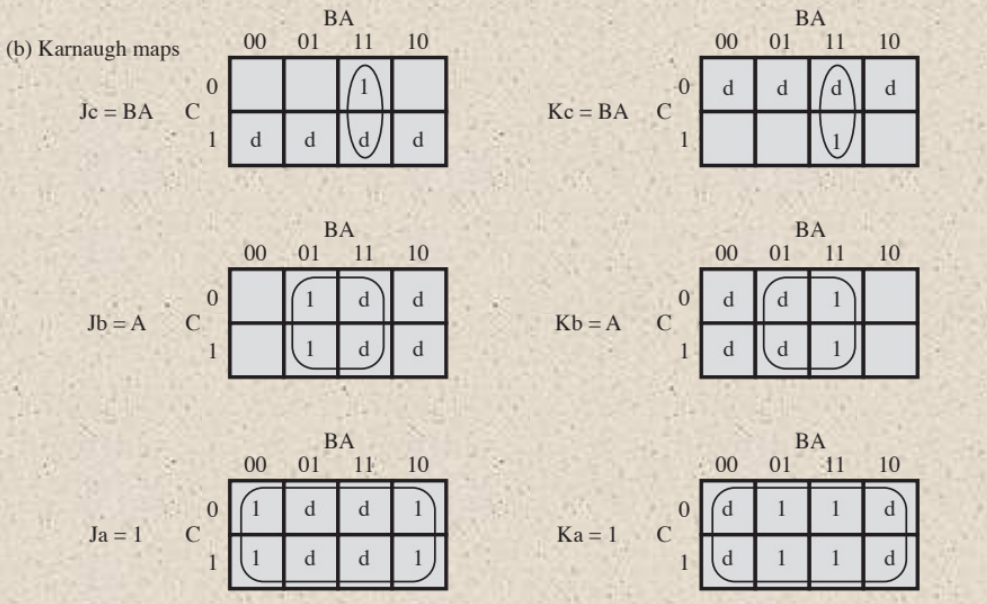


Figure 11.30 8-Bit Parallel Register

(a) Truth table

C	B	A	Jc	Kc	Jb	Kb	Ja	Ka
0	0	0	0	d	0	d	1	d
0	0	1	0	d	1	d	d	1
0	1	0	0	d	d	0	1	d
0	1	1	1	d	d	1	d	1
1	0	0	d	0	0	d	1	d
1	0	1	d	0	1	d	d	1
1	1	0	d	0	d	0	1	d
1	1	1	d	1	d	1	d	1

(b) Karnaugh maps



(c) Logic diagram

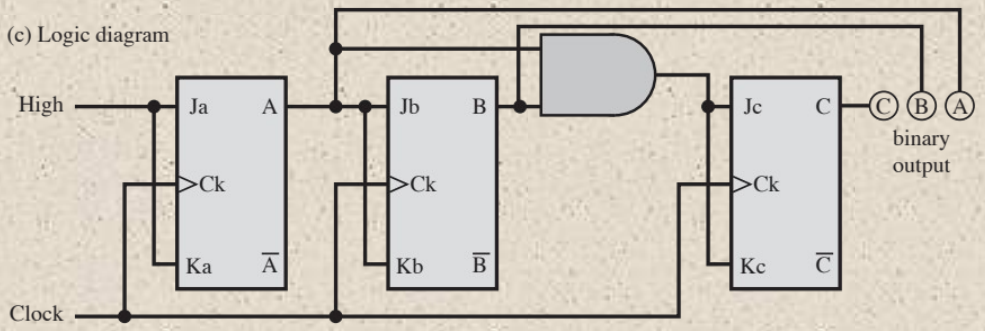


Figure 11.31 Design of a Synchronous Counter

Table

11.11

PLD

Terminology

Programmable Logic Device (PLD)

A general term that refers to any type of integrated circuit used for implementing digital hardware, where the chip can be configured by the end user to realize different designs. Programming of such a device often involves placing the chip into a special programming unit, but some chips can also be configured "in-system". Also referred to as a field-programmable device (FPD).

Programmable Logic Array (PLA)

A relatively small PLD that contains two levels of logic, an AND-plane and an OR-plane, where both levels are programmable.

Programmable Array Logic (PAL)

A relatively small PLD that has a programmable AND-plane followed by a fixed OR-plane.

Simple PLD (SPLD)

A PLA or PAL.

Complex PLD (CPLD)

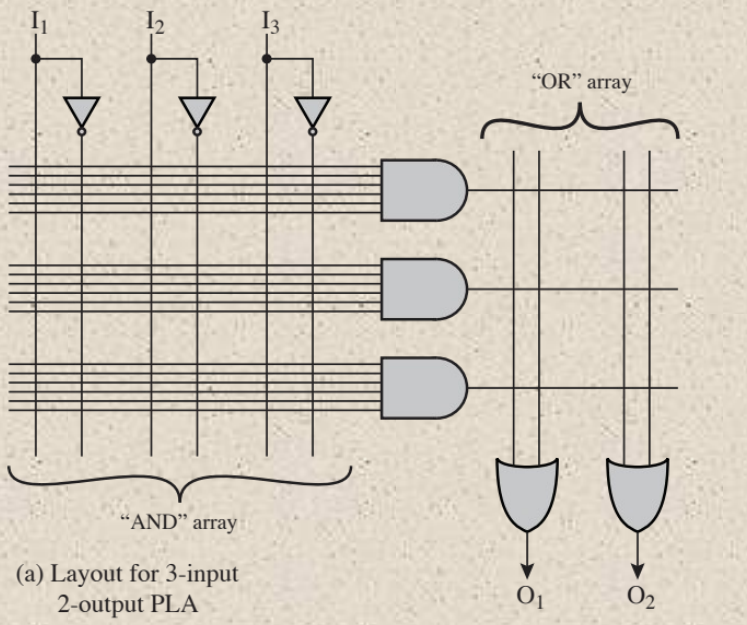
A more complex PLD that consists of an arrangement of multiple SPLD-like blocks on a single chip.

Field-Programmable Gate Array (FPGA)

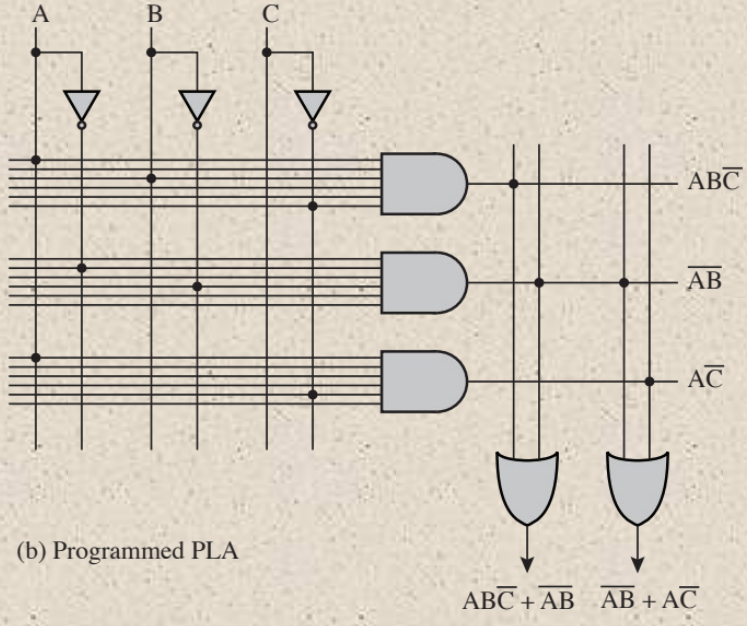
A PLD featuring a general structure that allows very high logic capacity. Whereas CPLDs feature logic resources with a wide number of inputs (AND planes), FPGAs offer more narrow logic resources. FPGAs also offer a higher ratio of flip-flops to logic resources than do CPLDs.

Logic Block

A relatively small circuit block that is replicated in an array in an FPD. When a circuit is implemented in an FPD, it is first decomposed into smaller sub-circuits that can each be mapped into a logic block. The term logic block is mostly used in the context of FPGAs, but it could also refer to a block of circuitry in a CPLD.



(a) Layout for 3-input 2-output PLA



(b) Programmed PLA

Figure 11.32 An Example of a Programmable Logic Array

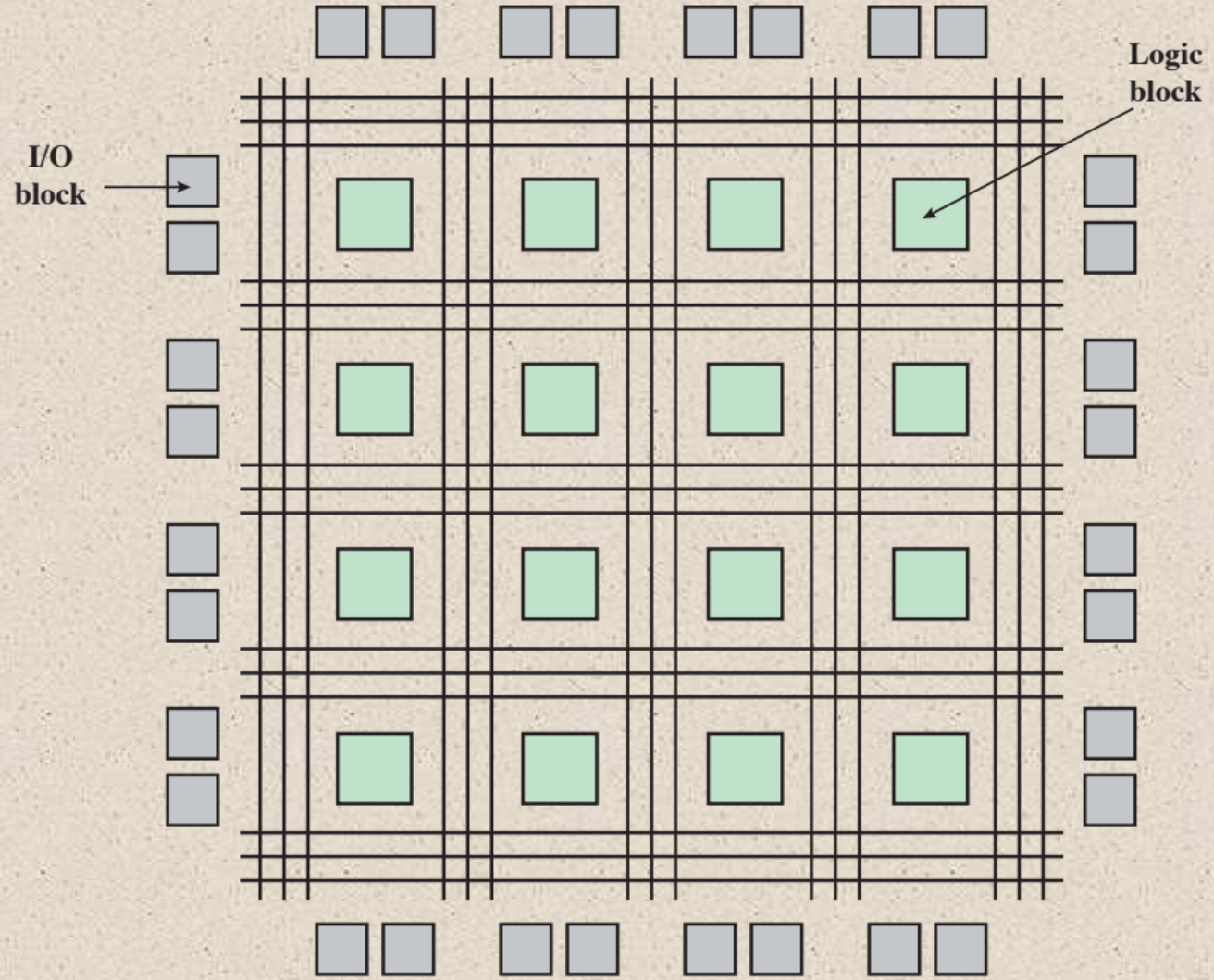


Figure 11.33 Structure of an FPGA

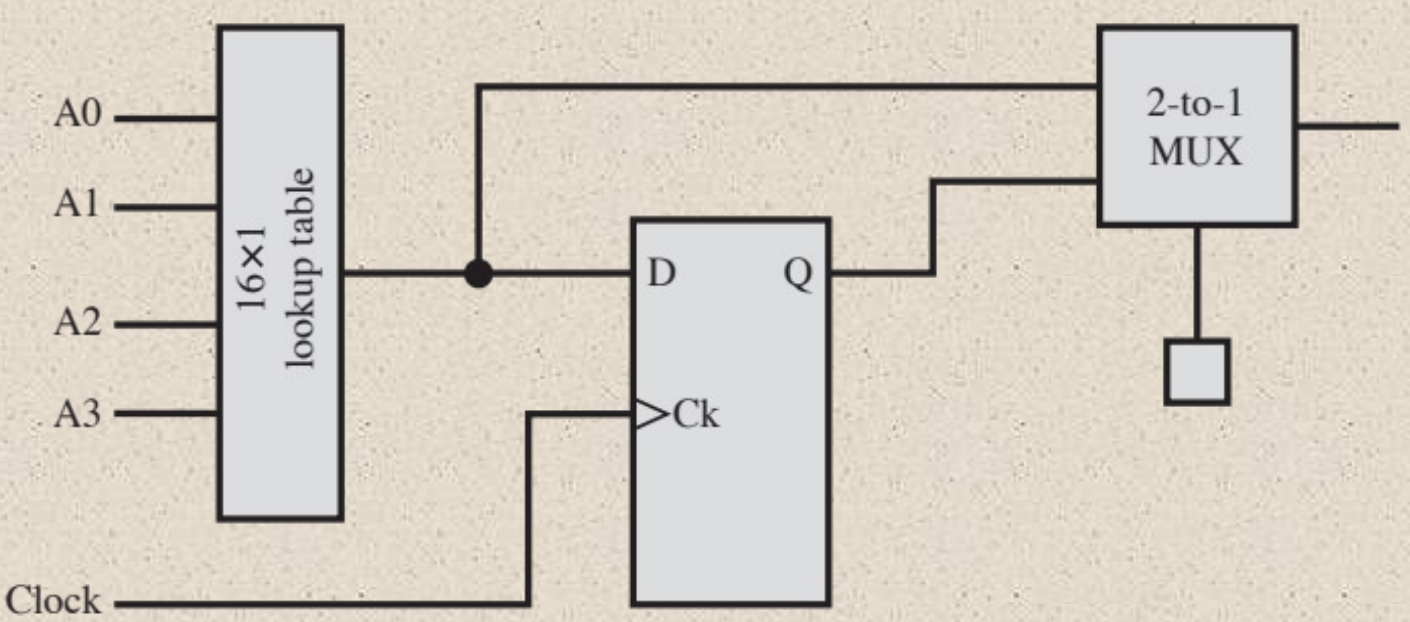


Figure 11.34 A Simple FPGA Logic Block

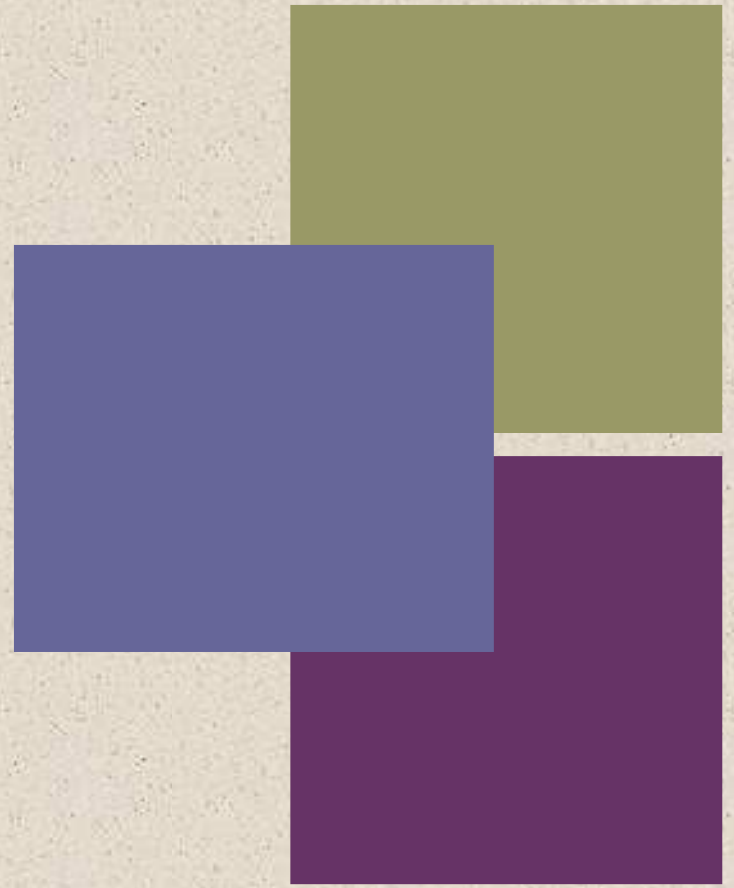
Summary

Chapter 11

Digital Logic

- Boolean Algebra
- Gates
- Combinational Circuits
  - Implementation of Boolean Functions
  - Multiplexers
  - Decoders
  - Read-Only-Memory
  - Adders
- Sequential Circuits
  - Flip-Flops
  - Registers
  - Counters
- Programmable Logic Devices
  - Programmable Logic Array
  - Field-Programmable Gate Array

William Stallings
Computer Organization and Architecture
10th Edition

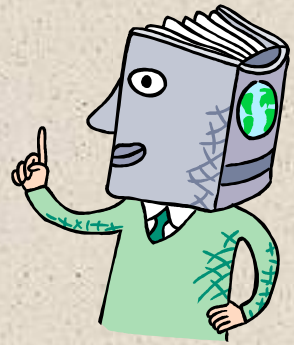


Chapter 12

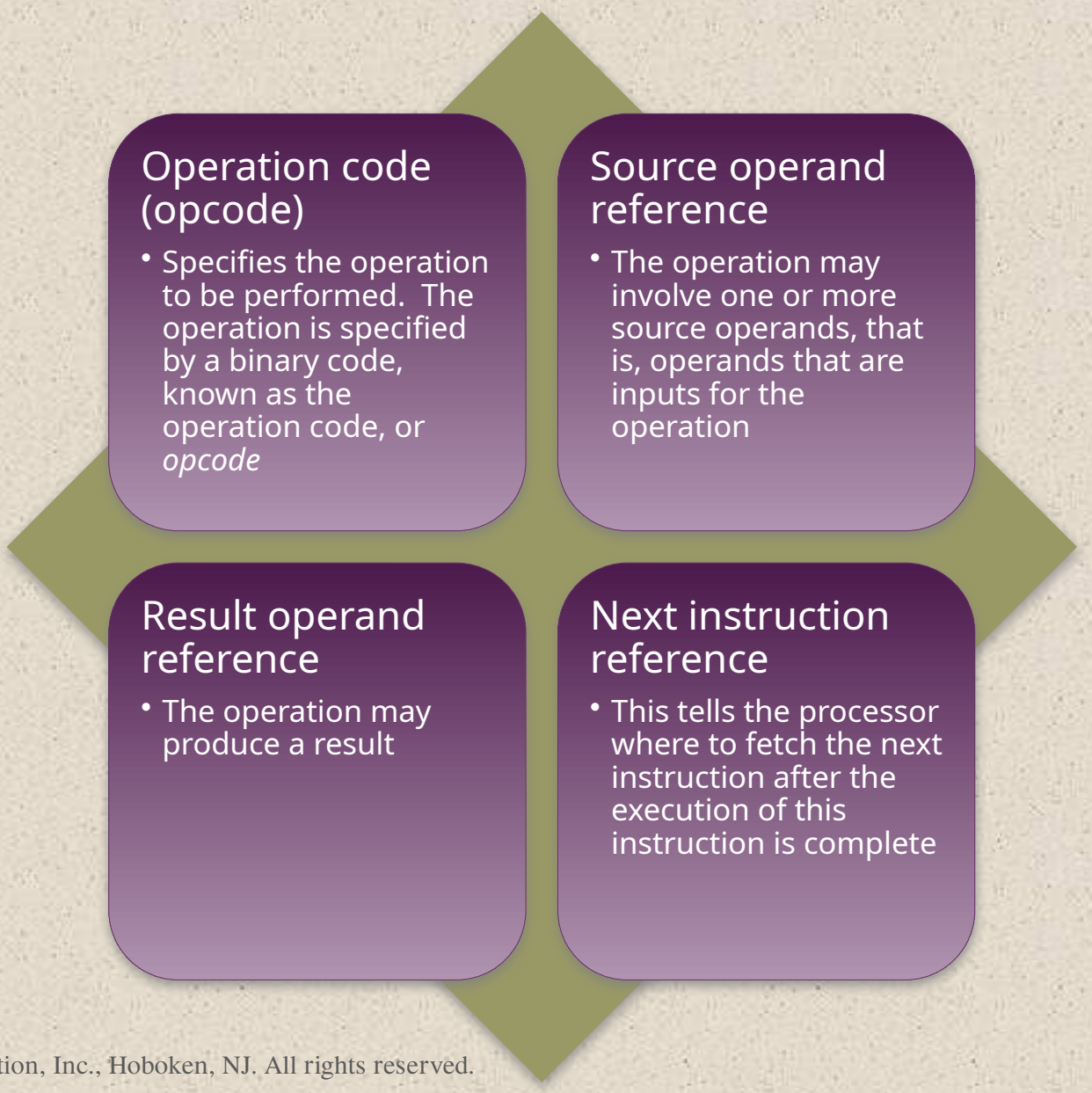
Instruction Sets: Characteristics and Functions

Machine Instruction Characteristics

- The operation of the processor is determined by the instructions it executes, referred to as *machine instructions* or *computer instructions*
- The collection of different instructions that the processor can execute is referred to as the processor's *instruction set*
- Each instruction must contain the information required by the processor for execution



Elements of a Machine Instruction



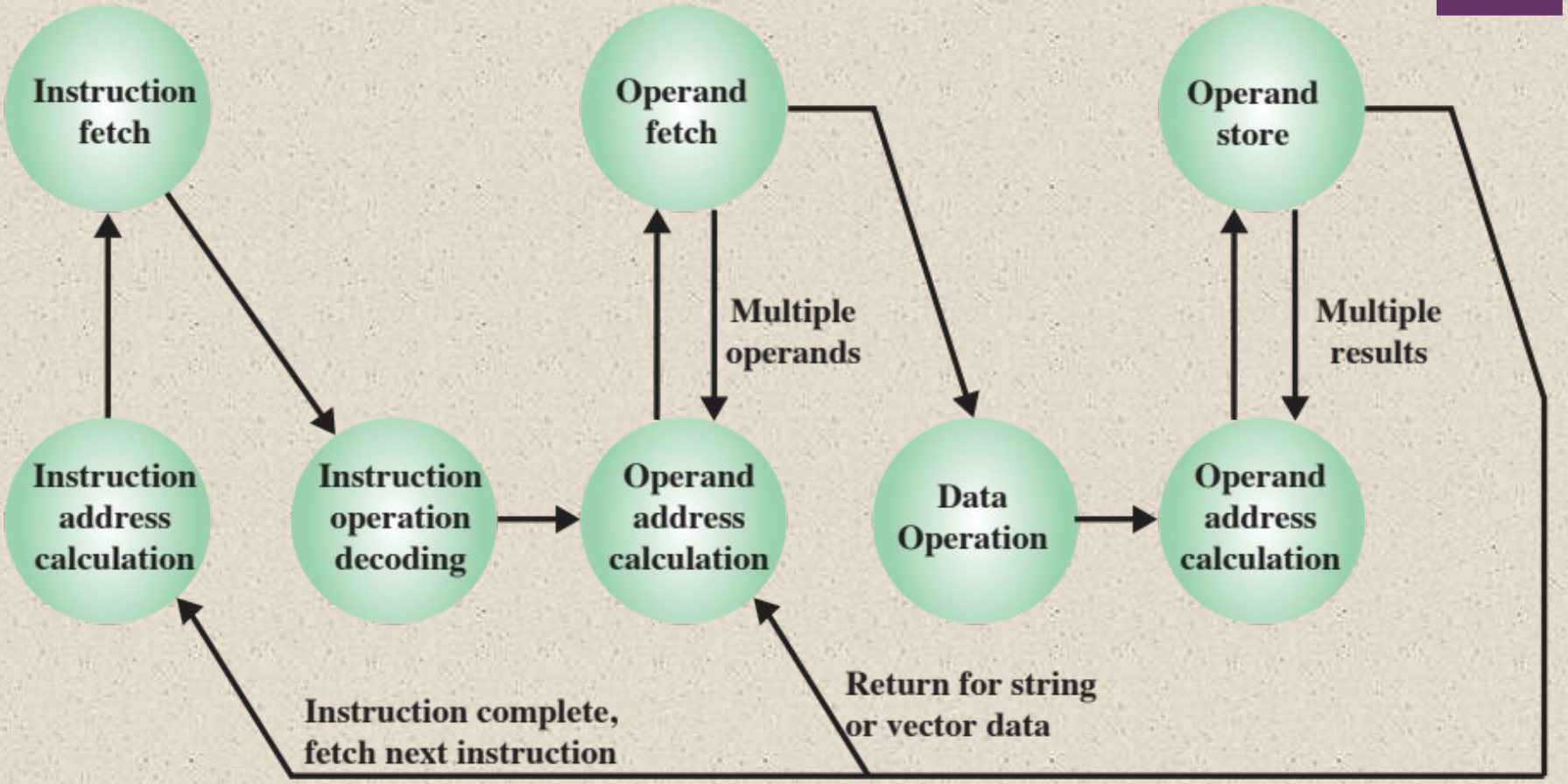


Figure 12.1 Instruction Cycle State Diagram

Source and result operands can be in one of four areas:

1) Main or virtual memory
   - As with next instruction references, the main or virtual memory address must be supplied

2) I/O device
   - The instruction must specify the I/O module and device for the operation. If memory-mapped I/O is used, this is just another main or virtual memory address

3) Processor register
   - A processor contains one or more registers that may be referenced by machine instructions.
   - If more than one register exists each register is assigned a unique name or number and the instruction must contain the number of the desired register

4) Immediate
   - The value of the operand is contained in a field in the instruction being executed

Instruction Representation

- Within the computer each instruction is represented by a sequence of bits
- The instruction is divided into fields, corresponding to the constituent elements of the instruction

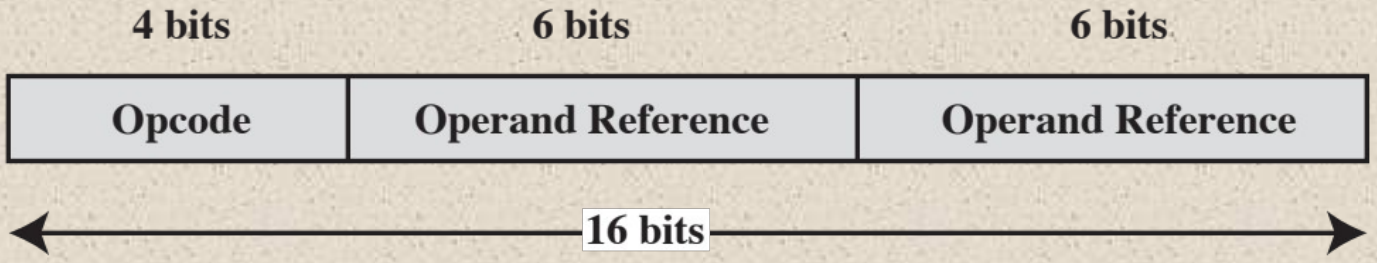


Figure 12.2 A Simple Instruction Format

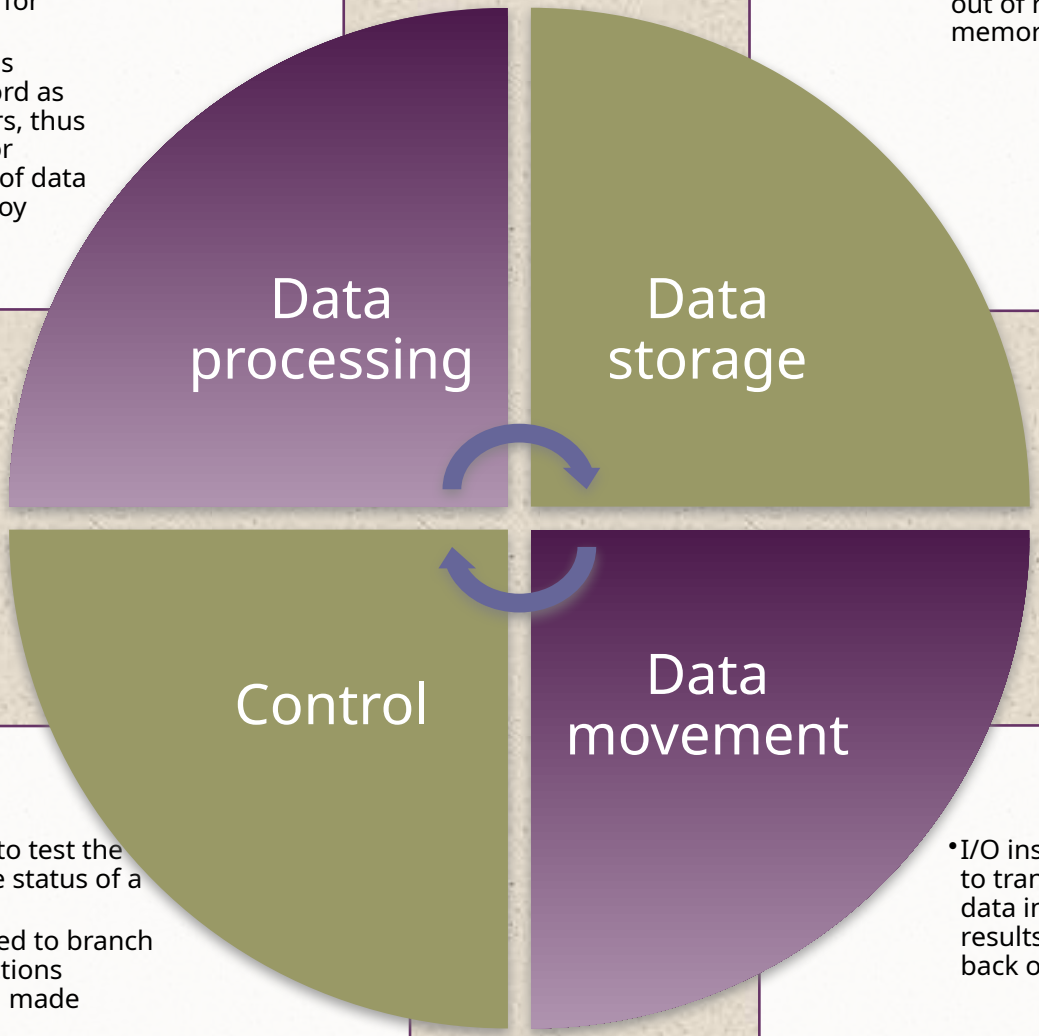
Instruction Representation

- Opcodes are represented by abbreviations called *mnemonics*
- Examples include:
  - ADD Add
  - SUB Subtract
  - MUL Multiply
  - DIV Divide
  - LOAD Load data from memory
  - STOR Store data to memory
- Operands are also represented symbolically
- Each symbolic opcode has a fixed binary representation
  - The programmer specifies the location of each symbolic operand

Instruction Types

- Arithmetic instructions provide computational capabilities for processing numeric data
- Logic (Boolean) instructions operate on the bits of a word as bits rather than as numbers, thus they provide capabilities for processing any other type of data the user may wish to employ

- Movement of data into or out of register and or memory locations



- Test instructions are used to test the value of a data word or the status of a computation
- Branch instructions are used to branch to a different set of instructions depending on the decision made

- I/O instructions are needed to transfer programs and data into memory and the results of computations back out to the user

Instruction		Comment
SUB	Y, A, B	Y ← A – B
MPY	T, D, E	T ← D × E
ADD	T, T, C	T ← T + C
DIV	Y, Y, T	Y ← Y ÷ T

(a) Three-address instructions

Instruction		Comment
MOVE	Y, A	Y ← A
SUB	Y, B	Y ← Y – B
MOVE	T, D	T ← D
MPY	T, E	T ← T × E
ADD	T, C	T ← T + C
DIV	Y, T	Y ← Y ÷ T

(b) Two-address instructions

Instruction		Comment
LOAD	D	AC ← D
MPY	E	AC ← AC × E
ADD	C	AC ← AC + C
STOR	Y	Y ← AC
LOAD	A	AC ← A
SUB	B	AC ← AC – B
DIV	Y	AC ← AC ÷ Y
STOR	Y	Y ← AC

(c) One-address instructions

Figure 12.3 Programs to Execute $Y = \dfrac{A - B}{C + (D \times E)}$

Table 12.1
Utilization of Instruction Addresses (Nonbranching Instructions)

Number of Addresses	Symbolic Representation	Interpretation
3	OP A, B, C	A ← B OP C
2	OP A, B	A ← A OP B
1	OP A	AC ← AC OP A
0	OP	T ← (T – 1) OP T

AC = accumulator
T = top of stack
(T – 1) = second element of stack
A, B, C = memory or register locations

Instruction Set Design

Very complex because it affects so many aspects of the computer system

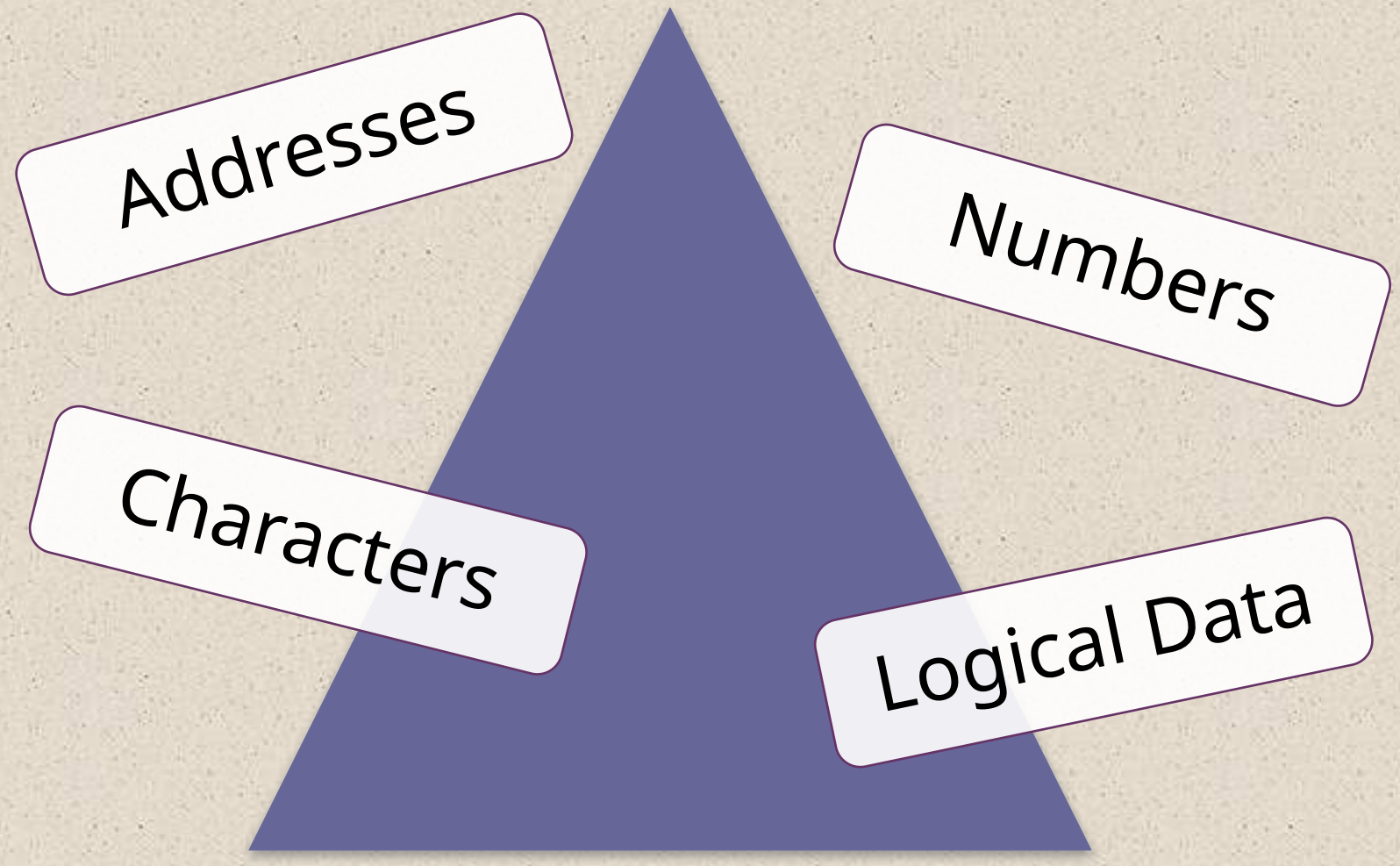
Defines many of the functions performed by the processor

Programmer's means of controlling the processor

Fundamental design issues:

Operation repertoire	Data types	Instruction format	Registers	Addressing
• How many and which operations to provide and how complex operations should be	• The various types of data upon which operations are performed	• Instruction length in bits, number of addresses, size of various fields, etc.	• Number of processor registers that can be referenced by instructions and their use	• The mode or modes by which the address of an operand is specified

Types of Operands



Numbers

- All machine languages include numeric data types
- Numbers stored in a computer are limited:
  - Limit to the magnitude of numbers representable on a machine
  - In the case of floating-point numbers, a limit to their precision
- Three types of numerical data are common in computers:
  - Binary integer or binary fixed point
  - Binary floating point
  - Decimal
- Packed decimal
  - Each decimal digit is represented by a 4-bit code with two digits stored per byte
  - To form numbers 4-bit codes are strung together, usually in multiples of 8 bits

Characters

- A common form of data is text or character strings
- Textual data in character form cannot be easily stored or transmitted by data processing and communications systems because they are designed for binary data
- Most commonly used character code is the International Reference Alphabet (IRA)
  - Referred to in the United States as the American Standard Code for Information Interchange (ASCII)
- Another code used to encode characters is the Extended Binary Coded Decimal Interchange Code (EBCDIC)
  - EBCDIC is used on IBM mainframes

Logical Data

- An *n*-bit unit consisting of *n* 1-bit items of data, each item having the value 0 or 1
- Two advantages to bit-oriented view:
  - Memory can be used most efficiently for storing an array of Boolean or binary data items in which each item can take on only the values 1 (true) and 0 (false)
  - To manipulate the bits of a data item
    - If floating-point operations are implemented in software, we need to be able to shift significant bits in some operations
    - To convert from IRA to packed decimal, we need to extract the rightmost 4 bits of each byte

Data Type	Description
General	Byte, word (16 bits), doubleword (32 bits), quadword (64 bits), and double quadword (128 bits) locations with arbitrary binary contents.
Integer	A signed binary value co4ntained in a byte, word, or doubleword, using twos complement representation.
Ordinal	An unsigned integer contained in a byte, word, or doubleword.
Unpacked binary coded decimal (BCD)	A representation of a BCD digit in the range 0 through 9, with one digit in each byte.
Packed BCD	Packed byte representation of two BCD digits; value in the range 0 to 99.
Near pointer	A 16-bit, 32-bit, or 64-bit effective address that represents the offset within a segment. Used for all pointers in a nonsegmented memory and for references within a segment in a segmented memory.
Far pointer	A logical address consisting of a 16-bit segment selector and an offset of 16, 32, or 64 bits. Far pointers are used for memory references in a segmented memory model where the identity of a segment being accessed must be specified explicitly.
Bit field	A contiguous sequence of bits in which the position of each bit is considered as an independent unit. A bit string can begin at any bit position of any byte and can contain up to 32 bits.
Bit string	A contiguous sequence of bits, containing from zero to $2_{32} - 1$ bits.
Byte string	A contiguous sequence of bytes, words, or doublewords, containing from zero to $2_{32} - 1$ bytes.
Floating point	See Figure 12.4.
Packed SIMD (single instruction, multiple data)	Packed 64-bit and 128-bit data types

Table 12.2

x86 Data Types

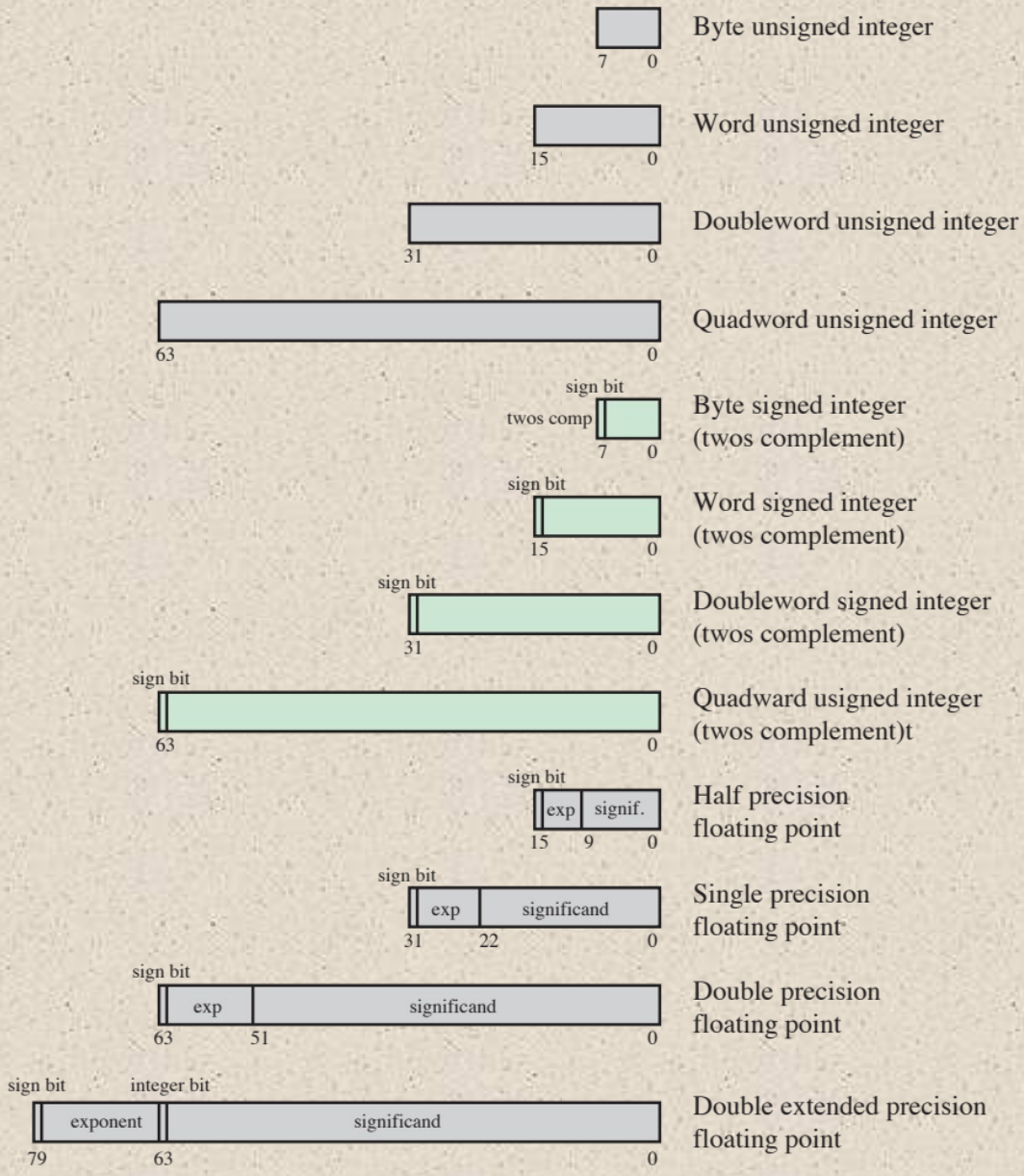


Figure 12.4 x86 Numeric Data Formats

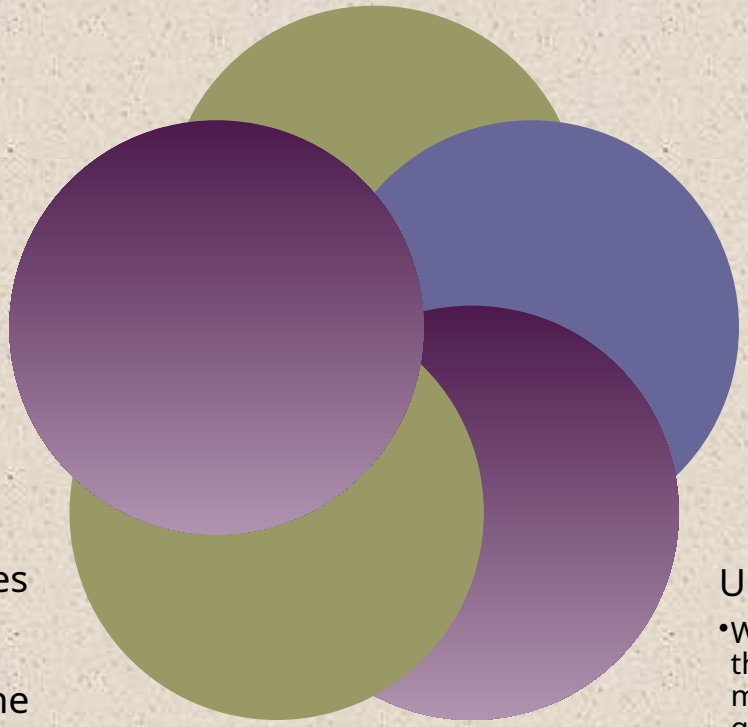
Single-Instruction-Multiple-Data (SIMD) Data Types

- Introduced to the x86 architecture as part of the extensions of the instruction set to optimize performance of multimedia applications
- These extensions include MMX (multimedia extensions) and SSE (streaming SIMD extensions)
- Data types:
  - Packed byte and packed byte integer
  - Packed word and packed word integer
  - Packed doubleword and packed doubleword integer
  - Packed quadword and packed quadword integer
  - Packed single-precision floating-point and packed double-precision floating-point

ARM Data Types

ARM processors support data types of:

- 8 (byte)
- 16 (halfword)
- 32 (word) bits in length



All three data types can also be used for twos complement signed integers

Alignment checking

- When the appropriate control bit is set, a data abort signal indicates an alignment fault for attempting unaligned access

For all three data types an unsigned interpretation is supported in which the value represents an unsigned, nonnegative integer

Unaligned access

- When this option is enabled, the processor uses one or more memory accesses to generate the required transfer of adjacent bytes transparently to the programmer

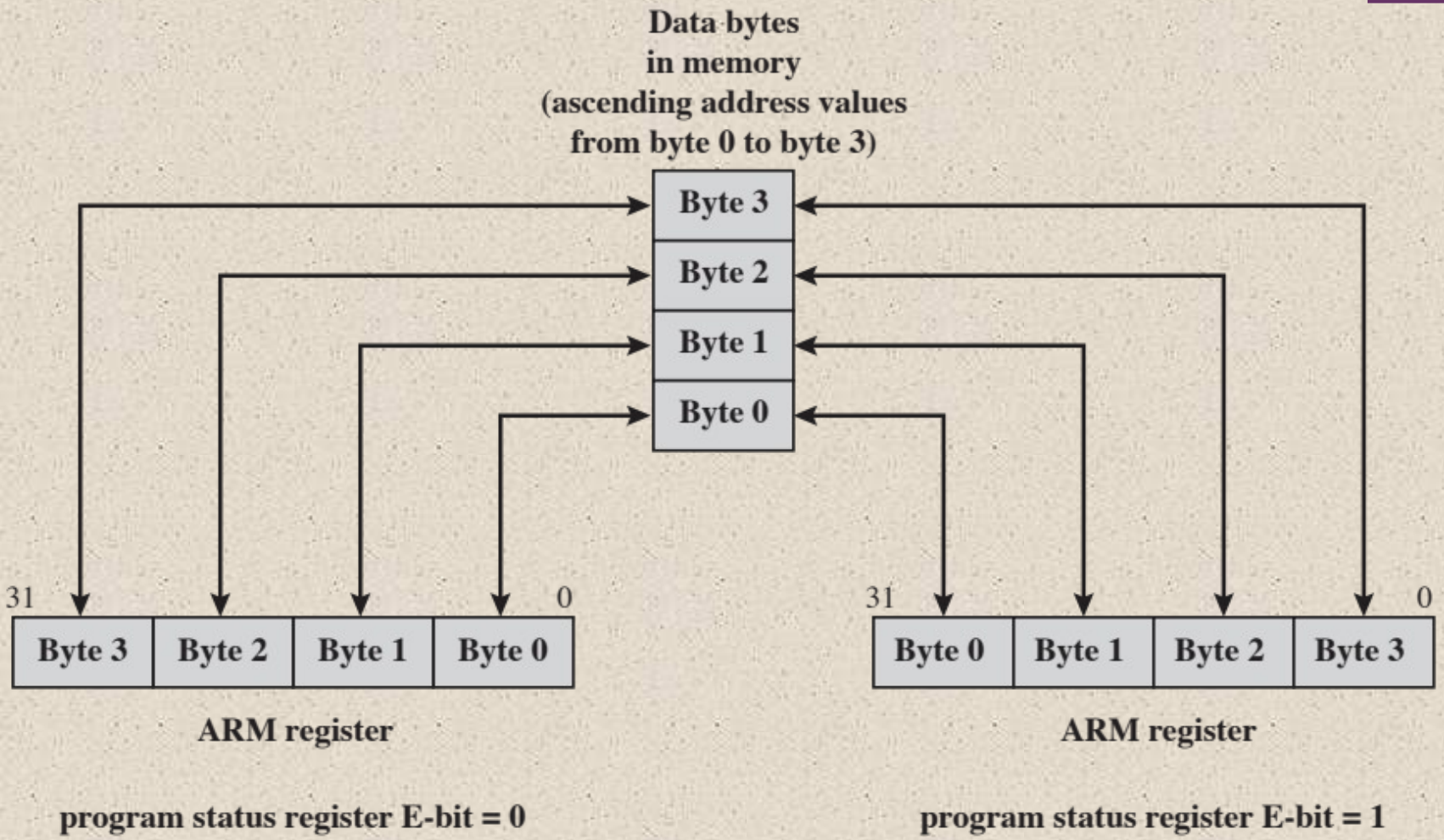


Figure 12.5 ARM Endian Support - Word Load/Store with E-bit

Type	Operation Name	Description
Data Transfer	Move (transfer)	Transfer word or block from source to destination
	Store	Transfer word from processor to memory
	Load (fetch)	Transfer word from memory to processor
	Exchange	Swap contents of source and destination
	Clear (reset)	Transfer word of 0s to destination
	Set	Transfer word of 1s to destination
	Push	Transfer word from source to top of stack
	Pop	Transfer word from top of stack to destination
Arithmetic	Add	Compute sum of two operands
	Subtract	Compute difference of two operands
	Multiply	Compute product of two operands
	Divide	Compute quotient of two operands
	Absolute	Replace operand by its absolute value
	Negate	Change sign of operand
	Increment	Add 1 to operand
	Decrement	Subtract 1 from operand
Logical	AND	Perform logical AND
	OR	Perform logical OR
	NOT (complement)	Perform logical NOT
	Exclusive-OR	Perform logical XOR
	Test	Test specified condition; set flag(s) based on outcome
	Compare	Make logical or arithmetic comparison of two or more operands; set flag(s) based on outcome
	Set Control Variables	Class of instructions to set controls for protection purposes, interrupt handling, timer control, etc.
	Shift	Left (right) shift operand, introducing constants at end
	Rotate	Left (right) shift operand, with wraparound end

Table 12.3

Common Instruction Set Operations (page 1 of 2)

(Table can be found on page 426 in textbook.)

Type	Operation Name	Description
Transfer of Control	Jump (branch)	Unconditional transfer; load PC with specified address
	Jump Conditional	Test specified condition; either load PC with specified address or do nothing, based on condition
	Jump to Subroutine	Place current program control information in known location; jump to specified address
	Return	Replace contents of PC and other register from known location
	Execute	Fetch operand from specified location and execute as instruction; do not modify PC
	Skip	Increment PC to skip next instruction
	Skip Conditional	Test specified condition; either skip or do nothing based on condition
	Halt	Stop program execution
	Wait (hold)	Stop program execution; test specified condition repeatedly; resume execution when condition is satisfied
	No operation	No operation is performed, but program execution is continued
Input/Output	Input (read)	Transfer data from specified I/O port or device to destination (e.g., main memory or processor register)
	Output (write)	Transfer data from specified source to I/O port or device
	Start I/O	Transfer instructions to I/O processor to initiate I/O operation
	Test I/O	Transfer status information from I/O system to specified destination
Conversion	Translate	Translate values in a section of memory based on a table of correspondences
	Convert	Convert the contents of a word from one form to another (e.g., packed decimal to binary)

Table 12.3

Common Instruction Set Operations (page 2 of 2)

(Table can be found on page 426 in textbook.)

Table 12.4
Processor Actions for Various Types of Operations

Data Transfer	Transfer data from one location to another
	If memory is involved: Determine memory address Perform virtual-to-actual-memory address transformation Check cache Initiate memory read/write
Arithmetic	May involve data transfer, before and/or after
	Perform function in ALU
	Set condition codes and flags
Logical	Same as arithmetic
Conversion	Similar to arithmetic and logical. May involve special logic to perform conversion
Transfer of Control	Update program counter. For subroutine call/return, manage parameter passing and linkage
I/O	Issue command to I/O module
	If memory-mapped I/O, determine memory-mapped address

(Table can be found on page 427 in textbook.)

Data Transfer

Most fundamental type of machine instruction

Must specify:

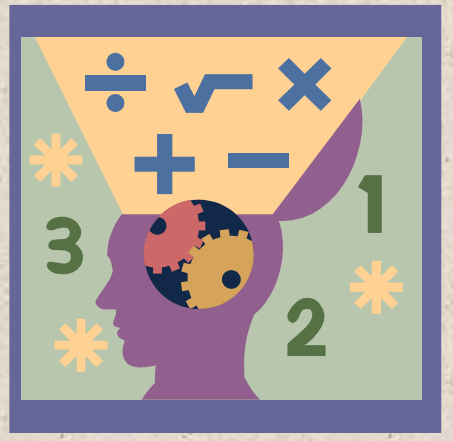
- Location of the source and destination operands
- The length of data to be transferred must be indicated
- The mode of addressing for each operand must be specified

Table 12.5
Examples of IBM EAS/390 Data Transfer Operations

Operation Mnemonic	Name	Number of Bits Transferred	Description
L	Load	32	Transfer from memory to register
LH	Load Halfword	16	Transfer from memory to register
LR	Load	32	Transfer from register to register
LER	Load (Short)	32	Transfer from floating-point register to floating-point register
LE	Load (Short)	32	Transfer from memory to floating-point register
LDR	Load (Long)	64	Transfer from floating-point register to floating-point register
LD	Load (Long)	64	Transfer from memory to floating-point register
ST	Store	32	Transfer from register to memory
STH	Store Halfword	16	Transfer from register to memory
STC	Store Character	8	Transfer from register to memory
STE	Store (Short)	32	Transfer from floating-point register to memory
STD	Store (Long)	64	Transfer from floating-point register to memory

(Table can be found on page 428 in textbook.)

Arithmetic



- Most machines provide the basic arithmetic operations of add, subtract, multiply, and divide
- These are provided for signed integer (fixed-point) numbers
- Often they are also provided for floating-point and packed decimal numbers
- Other possible operations include a variety of single-operand instructions:
  - Absolute
    - Take the absolute value of the operand
  - Negate
    - Negate the operand
  - Increment
    - Add 1 to the operand

  Decrement
    - Subtract 1 from the operand

Table 12.6
Basic Logical Operations

P	Q	NOT P	P AND Q	P OR Q	P XOR Q	P=Q
0	0	1	0	0	0	1
0	1	1	0	1	1	0
1	0	0	0	1	1	0
1	1	0	1	1	0	1

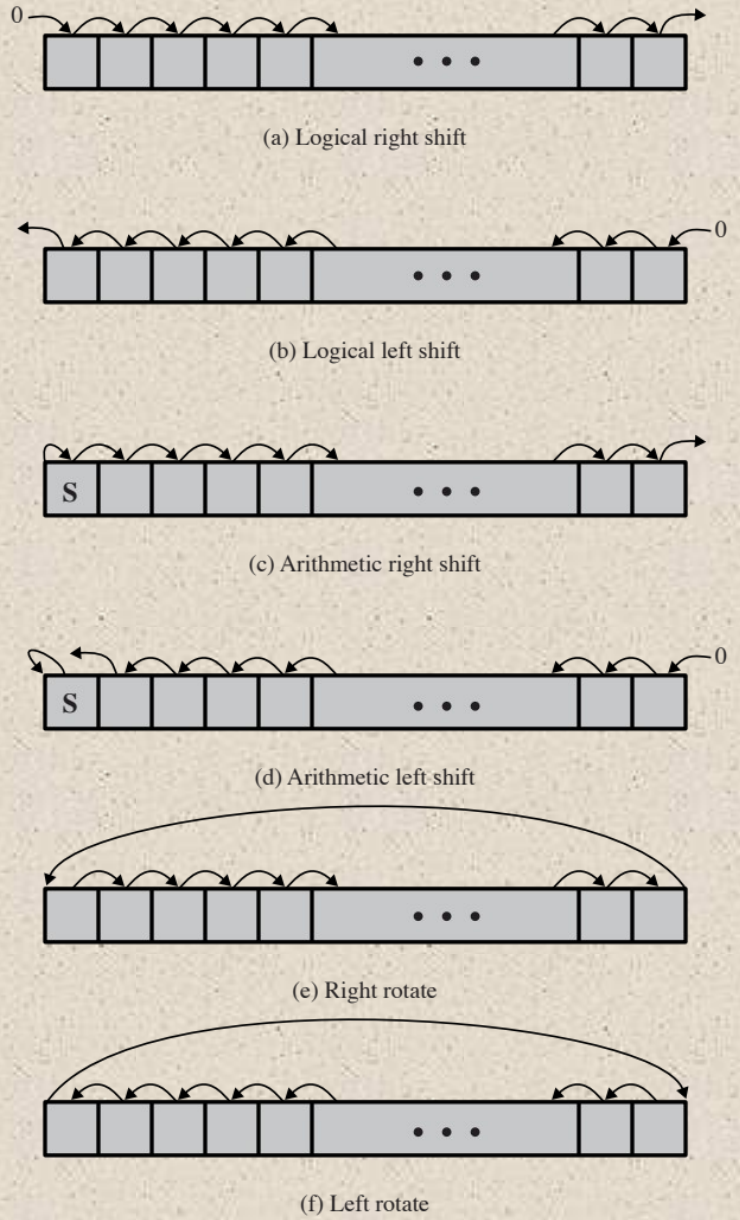


Figure 12.6 Shift and Rotate Operations

Table 12.7

Examples of Shift and Rotate Operations

Input	Operation	Result
10100110	Logical right shift (3 bits)	00010100
10100110	Logical left shift (3 bits)	00110000
10100110	Arithmetic right shift (3 bits)	11110100
10100110	Arithmetic left shift (3 bits)	10110000
10100110	Right rotate (3 bits)	11010100
10100110	Left rotate (3 bits)	00110101

Conversion

Instructions that change the format or operate on the format of data

An example is converting from decimal to binary

An example of a more complex editing instruction is the EAS/390 Translate (TR) instruction

Input/Output

- Variety of approaches taken:
  - Isolated programmed I/O
  - Memory-mapped programmed I/O
  - DMA
  - Use of an I/O processor
- Many implementations provide only a few I/O instructions, with the specific actions specified by parameters, codes, or command words

System Control

Instructions that can be executed only while the processor is in a certain privileged state or is executing a program in a special privileged area of memory

Typically these instructions are reserved for the use of the operating system

Examples of system control operations:

- A system control instruction may read or alter a control register
- An instruction to read or modify a storage protection key
- Access to process control blocks in a multiprogramming system

Transfer of Control

- Reasons why transfer-of-control operations are required:
  - It is essential to be able to execute each instruction more than once
  - Virtually all programs involve some decision making
  - It helps if there are mechanisms for breaking the task up into smaller pieces that can be worked on one at a time
- Most common transfer-of-control operations found in instruction sets:
  - Branch
  - Skip
  - Procedure call

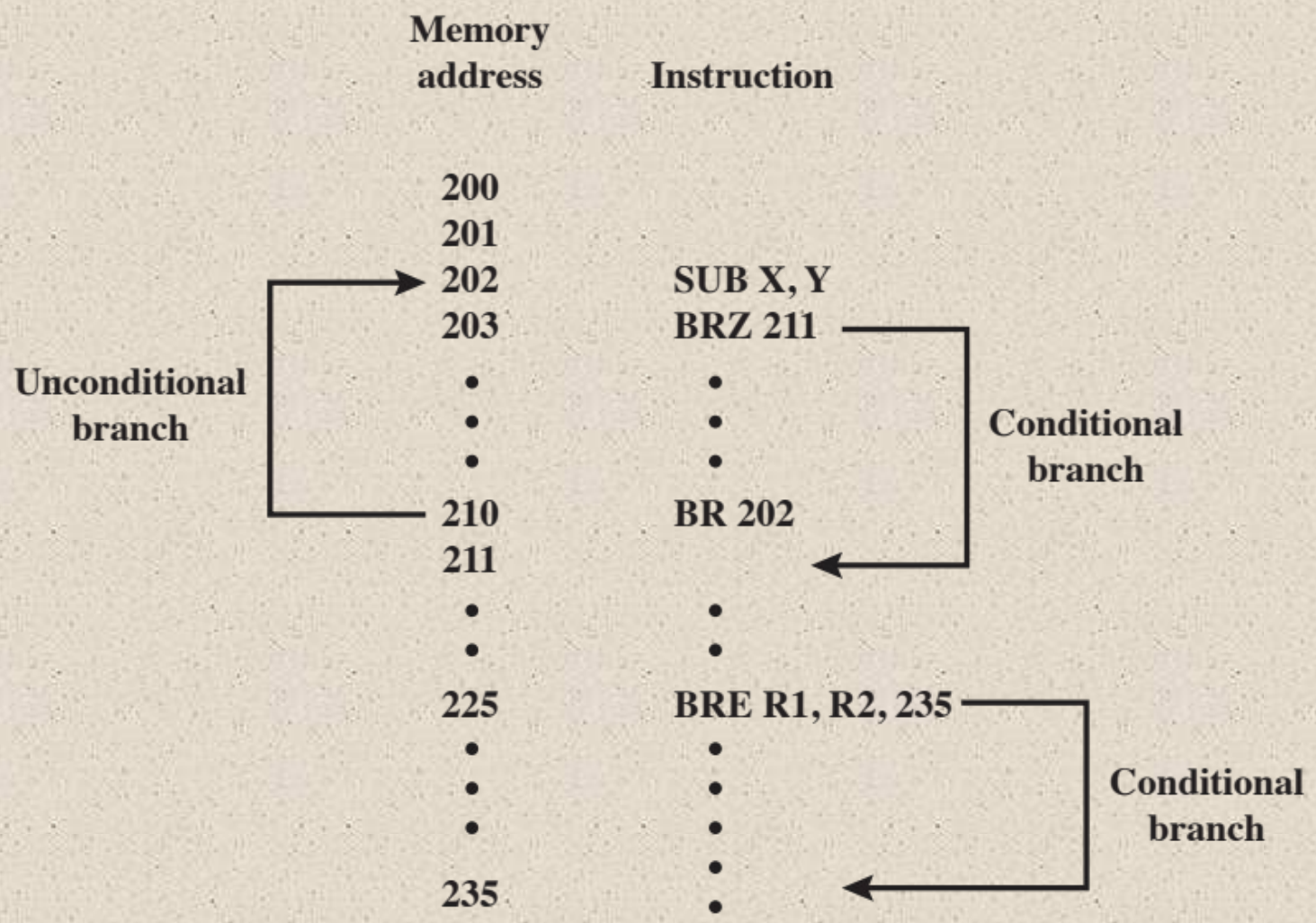


Figure 12.7 Branch Instructions

Skip Instructions

Includes an implied address

Typically implies that one instruction be skipped, thus the implied address equals the address of the next instruction plus one instruction length

Because the skip instruction does not require a destination address field it is free to do other things

Example is the increment-and-skip-if-zero (ISZ) instruction

Procedure Call Instructions

- Self-contained computer program that is incorporated into a larger program
  - At any point in the program the procedure may be invoked, or *called*
  - Processor is instructed to go and execute the entire procedure and then return to the point from which the call took place
- Two principal reasons for use of procedures:
  - Economy
    - A procedure allows the same piece of code to be used many times
  - Modularity
- Involves two basic instructions:
  - A call instruction that branches from the present location to the procedure
  - Return instruction that returns from the procedure to the place from which it was called

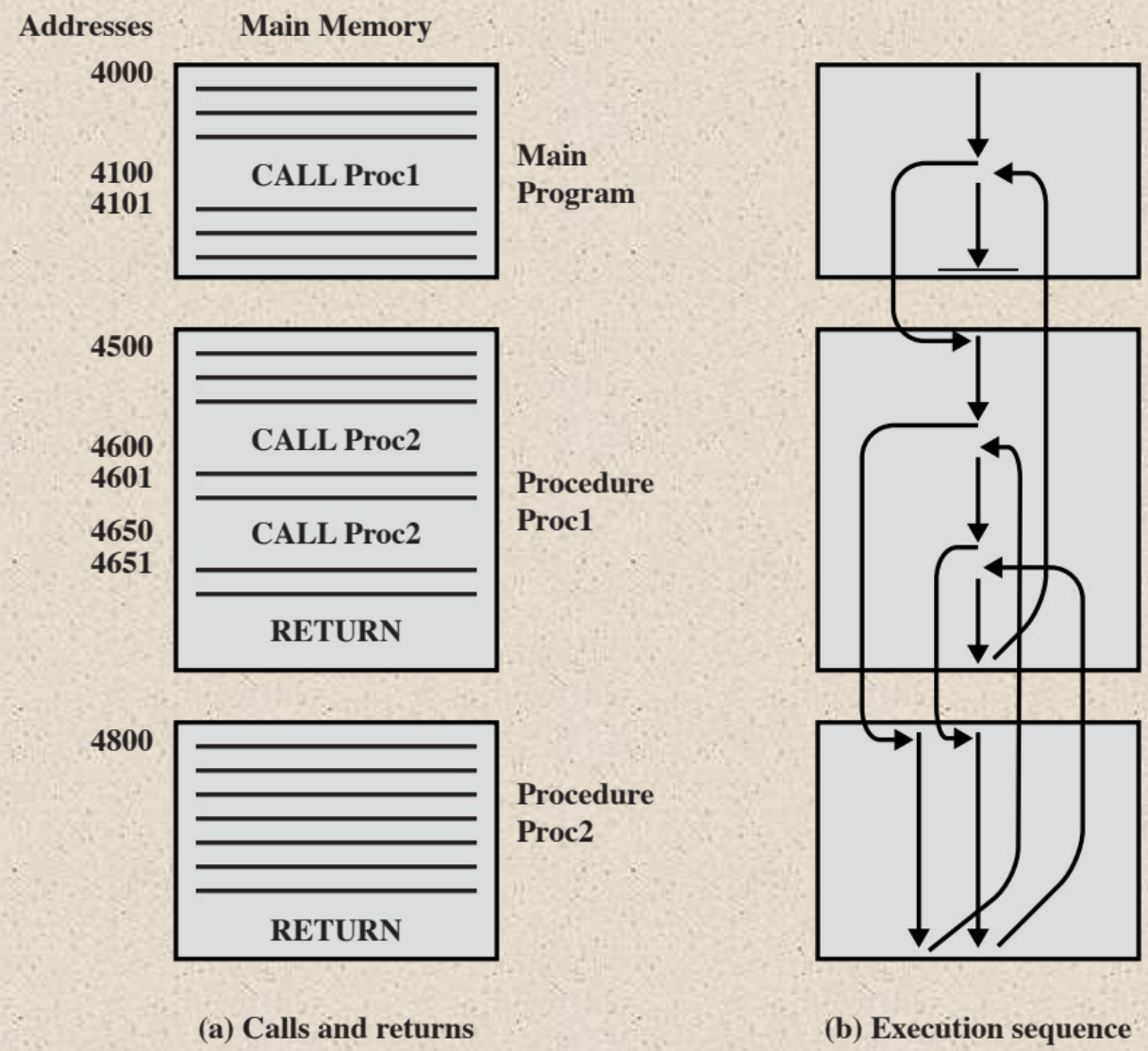


Figure 12.8 Nested Procedures

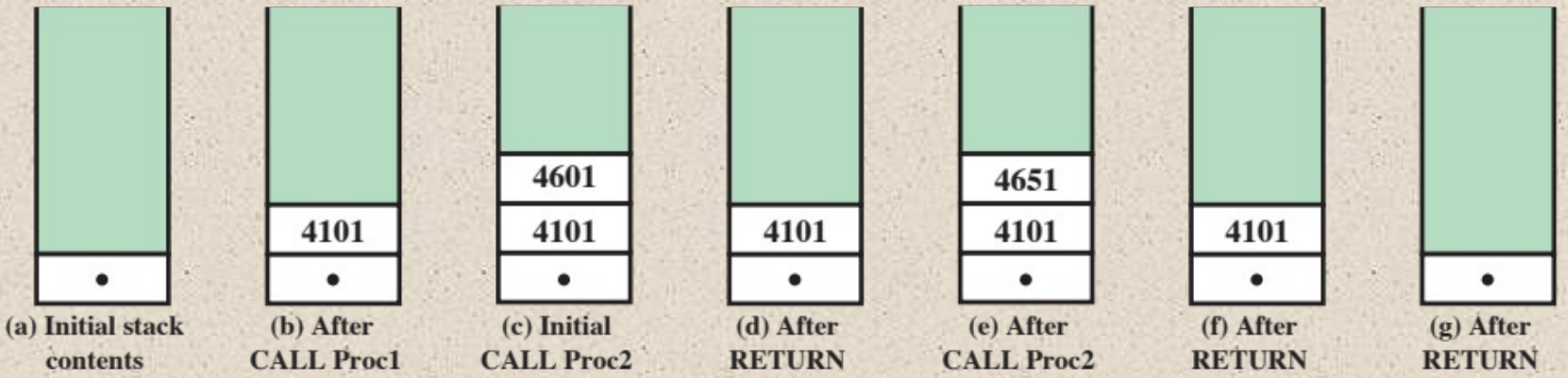


Figure 12.9 Use of Stack to Implement Nested Procedures of Figure 12.8

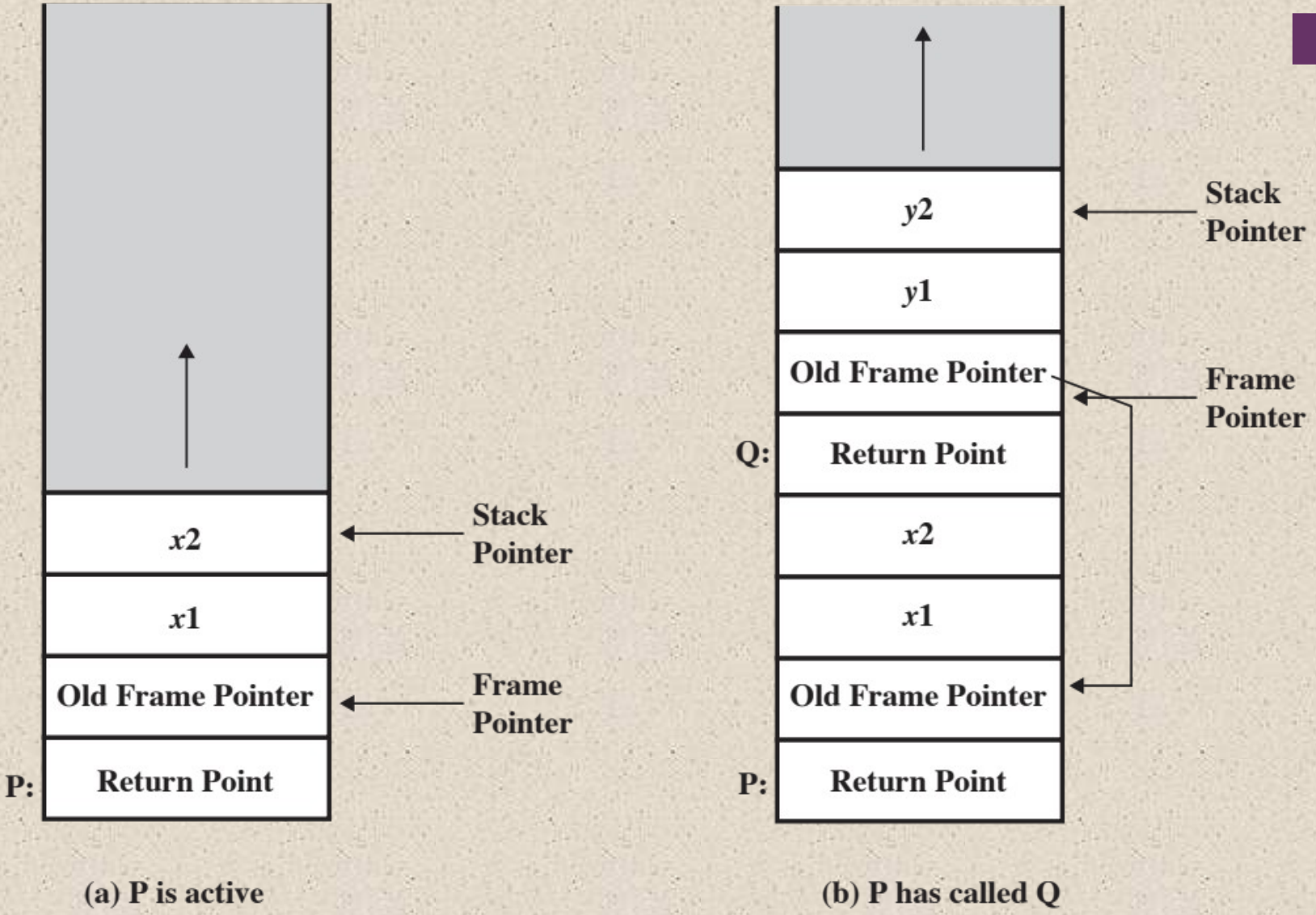


Figure 12.10 Stack Frame Growth Using Sample Procedures P and Q

x86 Operation Types

- The x86 provides a complex array of operation types including a number of specialized instructions
- The intent was to provide tools for the compiler writer to produce optimized machine language translation of high-level language programs
- Provides four instructions to support procedure call/return:
  - CALL
  - ENTER
  - LEAVE
  - RETURN
- When a new procedure is called the following must be performed upon entry to the new procedure:
  - Push the return point on the stack
  - Push the current frame pointer on the stack
  - Copy the stack pointer as the new value of the frame pointer
  - Adjust the stack pointer to allocate a frame

Table 12.8
x86 Status Flags

Status Bit	Name	Description
CF	Carry	Indicates carrying or borrowing out of the left-most bit position following an arithmetic operation. Also modified by some of the shift and rotate operations.
PF	Parity	Parity of the least-significant byte of the result of an arithmetic or logic operation. 1 indicates even parity; 0 indicates odd parity.
AF	Auxiliary Carry	Represents carrying or borrowing between half-bytes of an 8-bit arithmetic or logic operation. Used in binary-coded decimal arithmetic.
ZF	Zero	Indicates that the result of an arithmetic or logic operation is 0.
SF	Sign	Indicates the sign of the result of an arithmetic or logic operation.
OF	Overflow	Indicates an arithmetic overflow after an addition or subtraction for twos complement arithmetic.

Symbol	Condition Tested	Comment
A, NBE	CF=0 AND ZF=0	Above; Not below or equal (greater than, unsigned)
AE, NB, NC	CF=0	Above or equal; Not below (greater than or equal, unsigned); Not carry
B, NAE, C	CF=1	Below; Not above or equal (less than, unsigned); Carry set
BE, NA	CF=1 OR ZF=1	Below or equal; Not above (less than or equal, unsigned)
E, Z	ZF=1	Equal; Zero (signed or unsigned)
G, NLE	[(SF=1 AND OF=1) OR (SF=0 and OF=0)] AND [ZF=0]	Greater than; Not less than or equal (signed)
GE, NL	(SF=1 AND OF=1) OR (SF=0 AND OF=0)	Greater than or equal; Not less than (signed)
L, NGE	(SF=1 AND OF=0) OR (SF=0 AND OF=1)	Less than; Not greater than or equal (signed)
LE, NG	(SF=1 AND OF=0) OR (SF=0 AND OF=1) OR (ZF=1)	Less than or equal; Not greater than (signed)
NE, NZ	ZF=0	Not equal; Not zero (signed or unsigned)
NO	OF=0	No overflow
NS	SF=0	Not sign (not negative)
NP, PO	PF=0	Not parity; Parity odd
O	OF=1	Overflow
P	PF=1	Parity; Parity even
S	SF=1	Sign (negative)

Table 12.9

x86 Condition Codes for Conditional Jump and SETcc Instructions

(Table can be found on page 440 in the textbook.)

Table 12.10

MMX Instruction Set

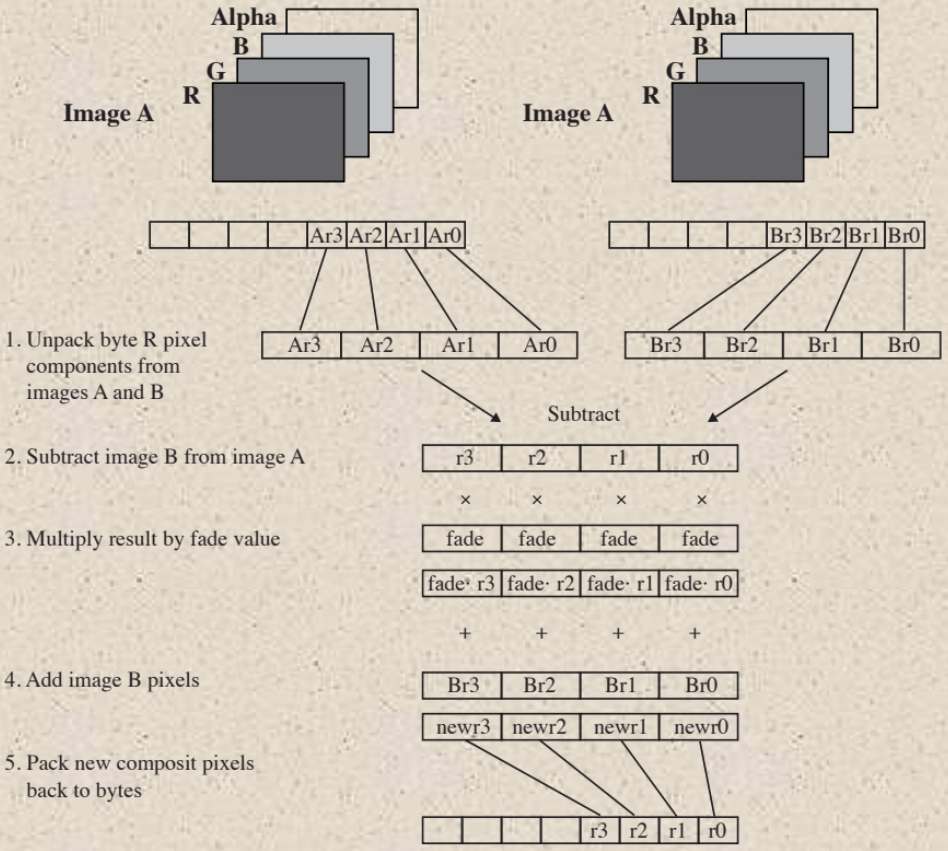
Category	Instruction	Description
Arithmetic	PADD [B, W, D]	Parallel add of packed eight bytes, four 16-bit words, or two 32-bit doublewords, with wraparound.
	PADDS [B, W]	Add with saturation.
	PADDUS [B, W]	Add unsigned with saturation
	PSUB [B, W, D]	Subtract with wraparound.
	PSUBS [B, W]	Subtract with saturation.
	PSUBUS [B, W]	Subtract unsigned with saturation
	PMULHW	Parallel multiply of four signed 16-bit words, with high-order 16 bits of 32-bit result chosen.
	PMULLW	Parallel multiply of four signed 16-bit words, with low-order 16 bits of 32-bit result chosen.
	PMADDWD	Parallel multiply of four signed 16-bit words; add together adjacent pairs of 32-bit results.
Comparison	PCMPEQ [B, W, D]	Parallel compare for equality; result is mask of 1s if true or 0s if false.
	PCMPGT [B, W, D]	Parallel compare for greater than; result is mask of 1s if true or 0s if false.
Conversion	PACKUSWB	Pack words into bytes with unsigned saturation.
	PACKSS [WB, DW]	Pack words into bytes, or doublewords into words, with signed saturation.
	PUNPCKH [BW, WD, DQ]	Parallel unpack (interleaved merge) high-order bytes, words, or doublewords from MMX register.
	PUNPCKL [BW, WD, DQ]	Parallel unpack (interleaved merge) low-order bytes, words, or doublewords from MMX register.
Logical	PAND	64-bit bitwise logical AND
	PNDN	64-bit bitwise logical AND NOT
	POR	64-bit bitwise logical OR
	PXOR	64-bit bitwise logical XOR
Shift	PSLL [W, D, Q]	Parallel logical left shift of packed words, doublewords, or quadword by amount specified in MMX register or immediate value.
	PSRL [W, D, Q]	Parallel logical right shift of packed words, doublewords, or quadword.
	PSRA [W, D]	Parallel arithmetic right shift of packed words, doublewords, or quadword.
Data Transfer	MOV [D, Q]	Move doubleword or quadword to/from MMX register.
State Mgt	EMMS	Empty MMX state (empty FP registers tag bits).

Note: If an instruction supports multiple data types [byte (B), word (W), doubleword (D), quadword (Q)], the data types are indicated in brackets.

(Table can be found on page 442 in the textbook.)

x86 Single-Instruction, Multiple-Data (SIMD) Instructions

- 1996 Intel introduced MMX technology into its Pentium product line
  - MMX is a set of highly optimized instructions for multimedia tasks
- Video and audio data are typically composed of large arrays of small data types
- Three new data types are defined in MMX
  - Packed byte
  - Packed word
  - Packed doubleword
- Each data type is 64 bits in length and consists of multiple smaller data fields, each of which holds a fixed-point integer



MMX code sequence performing this operation:

```
pxor      mm7, mm7      ;zero out mm7
movq      mm3, fad_val  ;load fade value replicated 4 times
movd      mm0, imageA   ;load 4 red pixel components from image A
movd      mm1, imageB   ;load 4 red pixel components from image B
punpckblw mm0, mm7      ;unpack 4 pixels to 16 bits
punpckblw mm1, mm7      ;unpack 4 pixels to 16 bits
psubw     mm0, mm1      ;subtract image B from image A
pmulhw    mm0, mm3      ;multiply the subtract result by fade values
padddw    mm0, mm1      ;add result to image B
packuswb  mm0, mm7      ;pack 16-bit results back to bytes
```

Figure 12.11 Image Compositing on Color Plane Representation

ARM Operation Types

- Load and store instructions
- Branch instructions
- Data-processing instructions
- Multiply instructions
- Parallel addition and subtraction instructions
- Extend instructions
- Status register access instructions

Code	Symbol	Condition Tested	Comment
0000	EQ	Z = 1	Equal
0001	NE	Z = 0	Not equal
0010	CS/HS	C = 1	Carry set/unsigned higher or same
0011	CC/LO	C = 0	Carry clear/unsigned lower
0100	MI	N = 1	Minus/negative
0101	PL	N = 0	Plus/positive or zero
0110	VS	V = 1	Overflow
0111	VC	V = 0	No overflow
1000	HI	C = 1 AND Z = 0	Unsigned higher
1001	LS	C = 0 OR Z = 1	Unsigned lower or same
1010	GE	N = V [(N = 1 AND V = 1) OR (N = 0 AND V = 0)	Signed greater than or equal
1011	LT	N ≠ V [(N = 1 AND V = 0) OR (N = 0 AND V = 1)]	Signed less than
1100	GT	(Z = 0) AND (N = V)	Signed greater than
1101	LE	(Z = 1) OR (N ≠ V)	Signed less than or equal
1110	AL	—	Always (unconditional)
1111	—	—	This instruction can only be executed unconditionally

Table 12.11

ARM Conditions for Conditional Instruction Execution

(Table can be found on Page 445 in the textbook.)

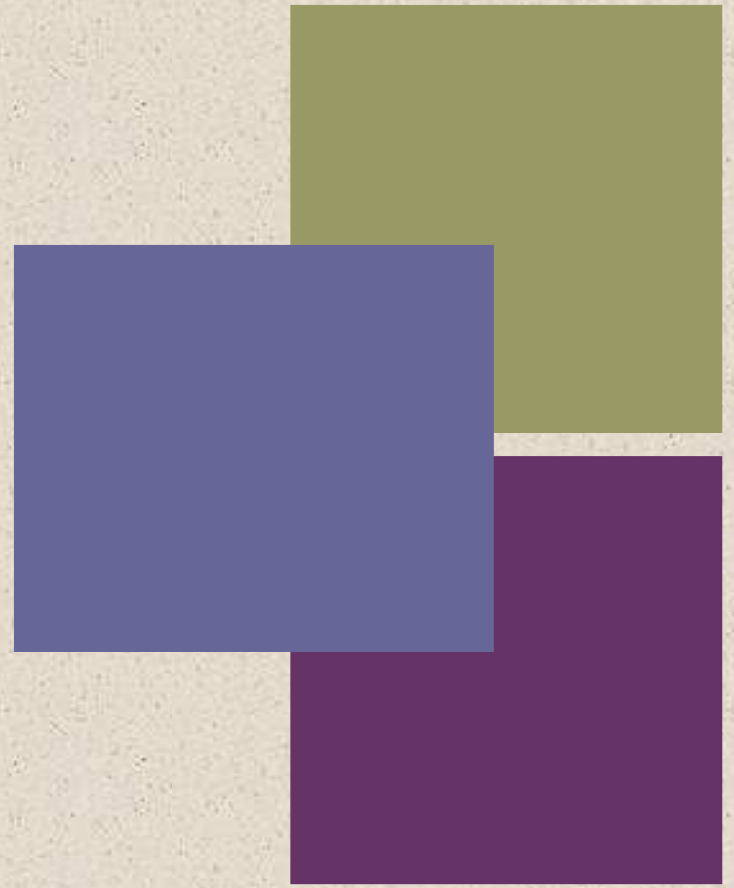
Summary

Chapter 12

Instruction Sets: Characteristics and Functions

- Machine instruction characteristics
  - Elements of a machine instruction
  - Instruction representation
  - Instruction types
  - Number of addresses
  - Instruction set design
- Types of operands
  - Numbers
  - Characters
  - Logical data
- Intel x86 and ARM data types
- Types of operations
  - Data transfer
  - Arithmetic
  - Logical
  - Conversion
  - Input/output
  - System control
  - Transfer of control
- Intel x86 and ARM operation types

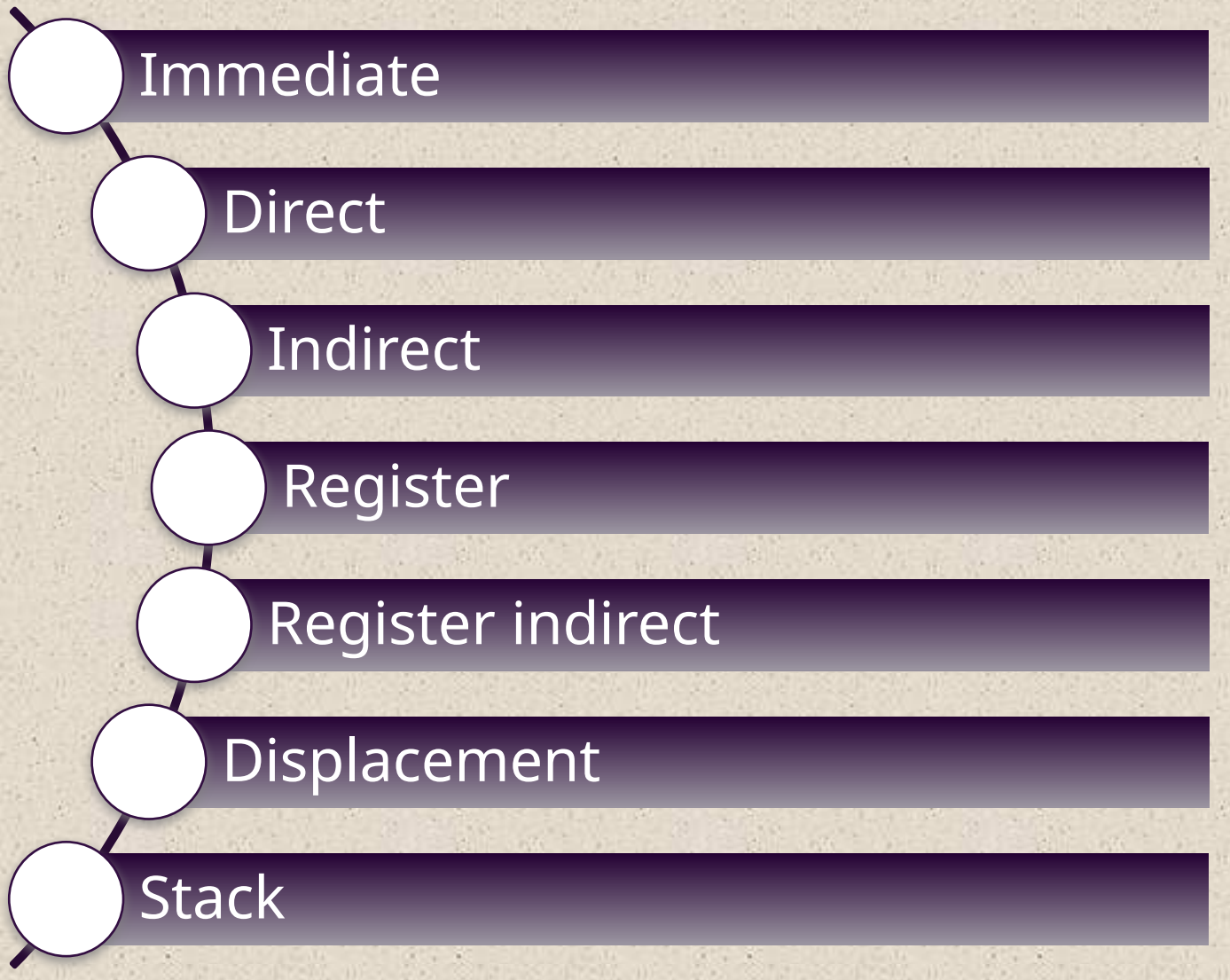
William Stallings
Computer Organization and Architecture
10th Edition



Chapter 13

Instruction Sets: Addressing Modes and Formats

Addressing Modes



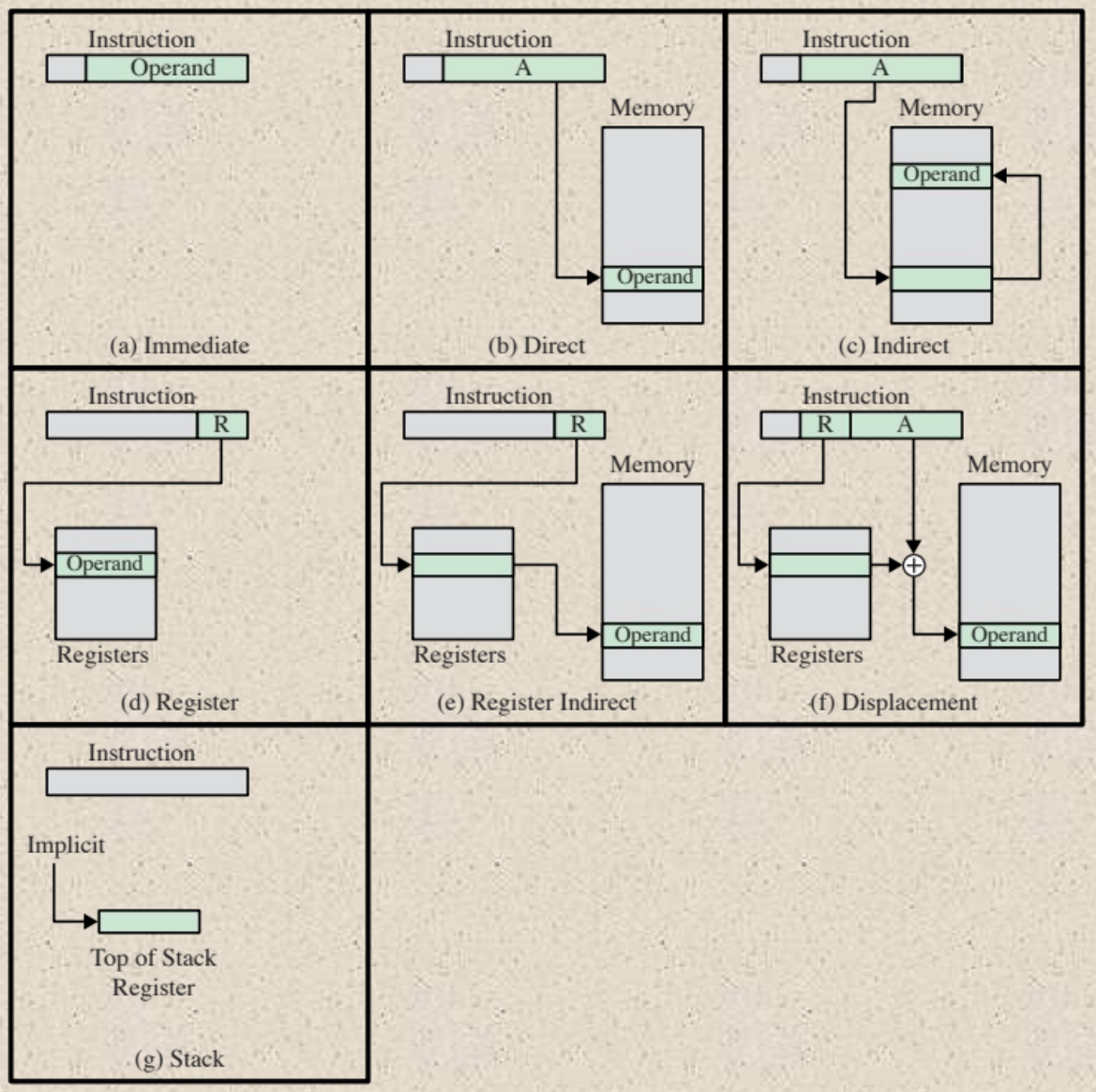


Figure 13.1 Addressing Modes

Table 13.1
Basic Addressing Modes

Mode	Algorithm	Principal Advantage	Principal Disadvantage
Immediate	Operand = A	No memory reference	Limited operand magnitude
Direct	EA = A	Simple	Limited address space
Indirect	EA = (A)	Large address space	Multiple memory references
Register	EA = R	No memory reference	Limited address space
Register indirect	EA = (R)	Large address space	Extra memory reference
Displacement	EA = A + (R)	Flexibility	Complexity
Stack	EA = top of stack	No memory reference	Limited applicability

Immediate Addressing

- Simplest form of addressing
- Operand = A
- This mode can be used to define and use constants or set initial values of variables
  - Typically the number will be stored in twos complement form
  - The leftmost bit of the operand field is used as a sign bit
- Advantage:
  - No memory reference other than the instruction fetch is required to obtain the operand, thus saving one memory or cache cycle in the instruction cycle
- Disadvantage:
  - The size of the number is restricted to the size of the address field, which, in most instruction sets, is small compared with the word length

Direct Addressing

Address field contains the effective address of the operand

Effective address (EA) = address field (A)

Was common in earlier generations of computers

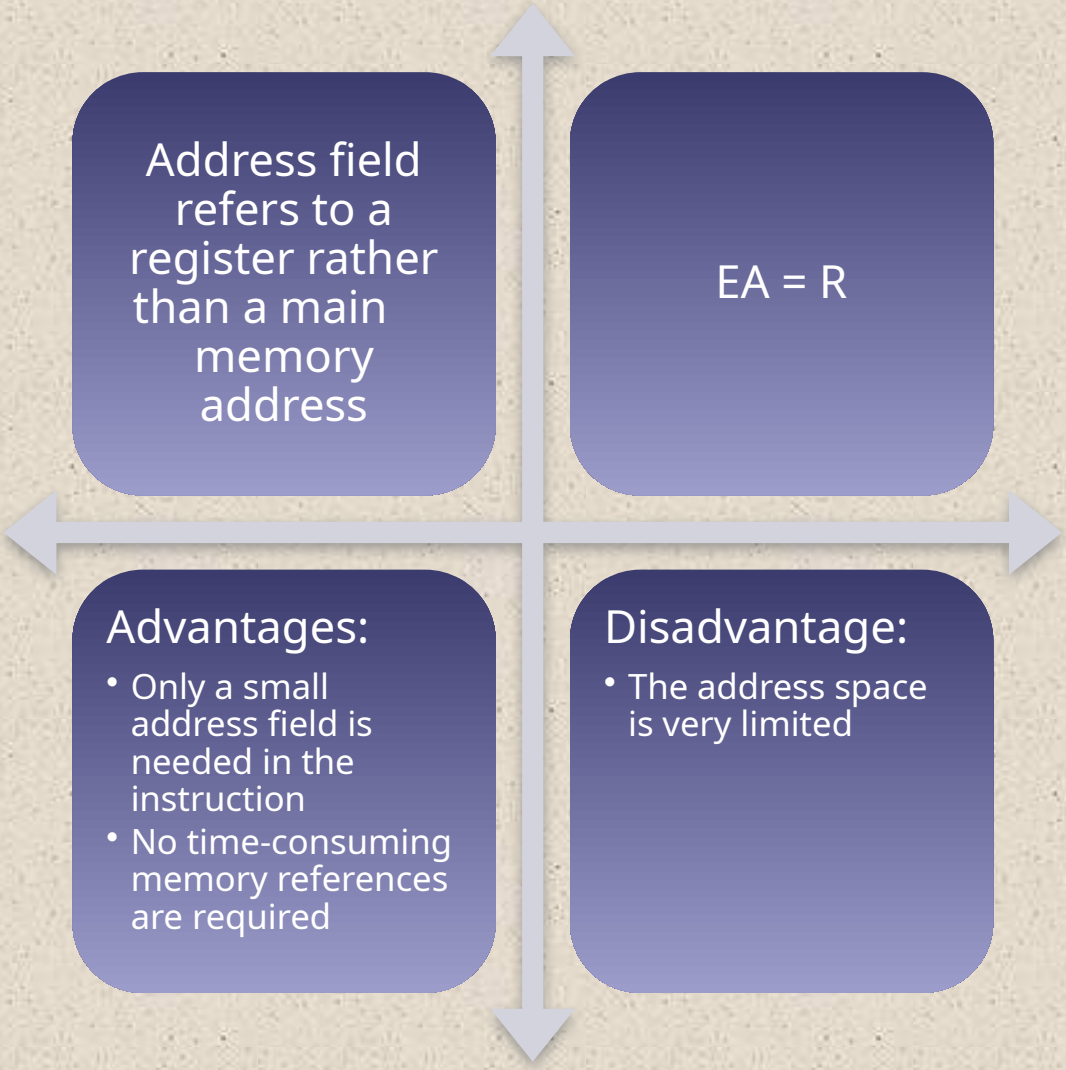
Requires only one memory reference and no special calculation

Limitation is that it provides only a limited address space

Indirect Addressing

- Reference to the address of a word in memory which contains a full-length address of the operand
- EA = (A)
  - Parentheses are to be interpreted as meaning *contents of*
- Advantage:
  - For a word length of *N* an address space of 2^N is now available
- Disadvantage:
  - Instruction execution requires two memory references to fetch the operand
    - One to get its address and a second to get its value
- A rarely used variant of indirect addressing is multilevel or cascaded indirect addressing
  - EA = (. . . (A) . . .)
  - Disadvantage is that three or more memory references could be required to fetch an operand

Register Addressing



Register Indirect Addressing

- Analogous to indirect addressing
  - The only difference is whether the address field refers to a memory location or a register
- EA = (R)
- Address space limitation of the address field is overcome by having that field refer to a word-length location containing an address
- Uses one less memory reference than indirect addressing

Displacement Addressing

- Combines the capabilities of direct addressing and register indirect addressing
- EA = A + (R)
- Requires that the instruction have two address fields, at least one of which is explicit
  - The value contained in one address field (value = A) is used directly
  - The other address field refers to a register whose contents are added to A to produce the effective address
- Most common uses:
  - Relative addressing
  - Base-register addressing
  - Indexing

Relative Addressing

The implicitly referenced register is the program counter (PC)

- The next instruction address is added to the address field to produce the EA
- Typically the address field is treated as a twos complement number for this operation
- Thus the effective address is a displacement relative to the address of the instruction

Exploits the concept of locality

Saves address bits in the instruction if most memory references are relatively near to the instruction being executed

Base-Register Addressing

- The referenced register contains a main memory address and the address field contains a displacement from that address
- The register reference may be explicit or implicit
- Exploits the locality of memory references
- Convenient means of implementing segmentation
- In some implementations a single segment base register is employed and is used implicitly
- In others the programmer may choose a register to hold the base address of a segment and the instruction must reference it explicitly

Indexing

- The address field references a main memory address and the referenced register contains a positive displacement from that address
- The method of calculating the EA is the same as for base-register addressing
- An important use is to provide an efficient mechanism for performing iterative operations
- Autoindexing
  - Automatically increment or decrement the index register after each reference to it
  - EA = A + (R)
  - (R) ← (R) + 1
- Postindexing
  - Indexing is performed after the indirection
  - EA = (A) + (R)
- Preindexing
  - Indexing is performed before the indirection
  - EA = (A + (R))

Stack Addressing

- A stack is a linear array of locations
  - Sometimes referred to as a *pushdown list* or *last-in-first-out queue*
- A stack is a reserved block of locations
  - Items are appended to the top of the stack so that the block is partially filled
- Associated with the stack is a pointer whose value is the address of the top of the stack
  - The stack pointer is maintained in a register
  - Thus references to stack locations in memory are in fact register indirect addresses
- Is a form of implied addressing
- The machine instructions need not include a memory reference but implicitly operate on the top of the stack

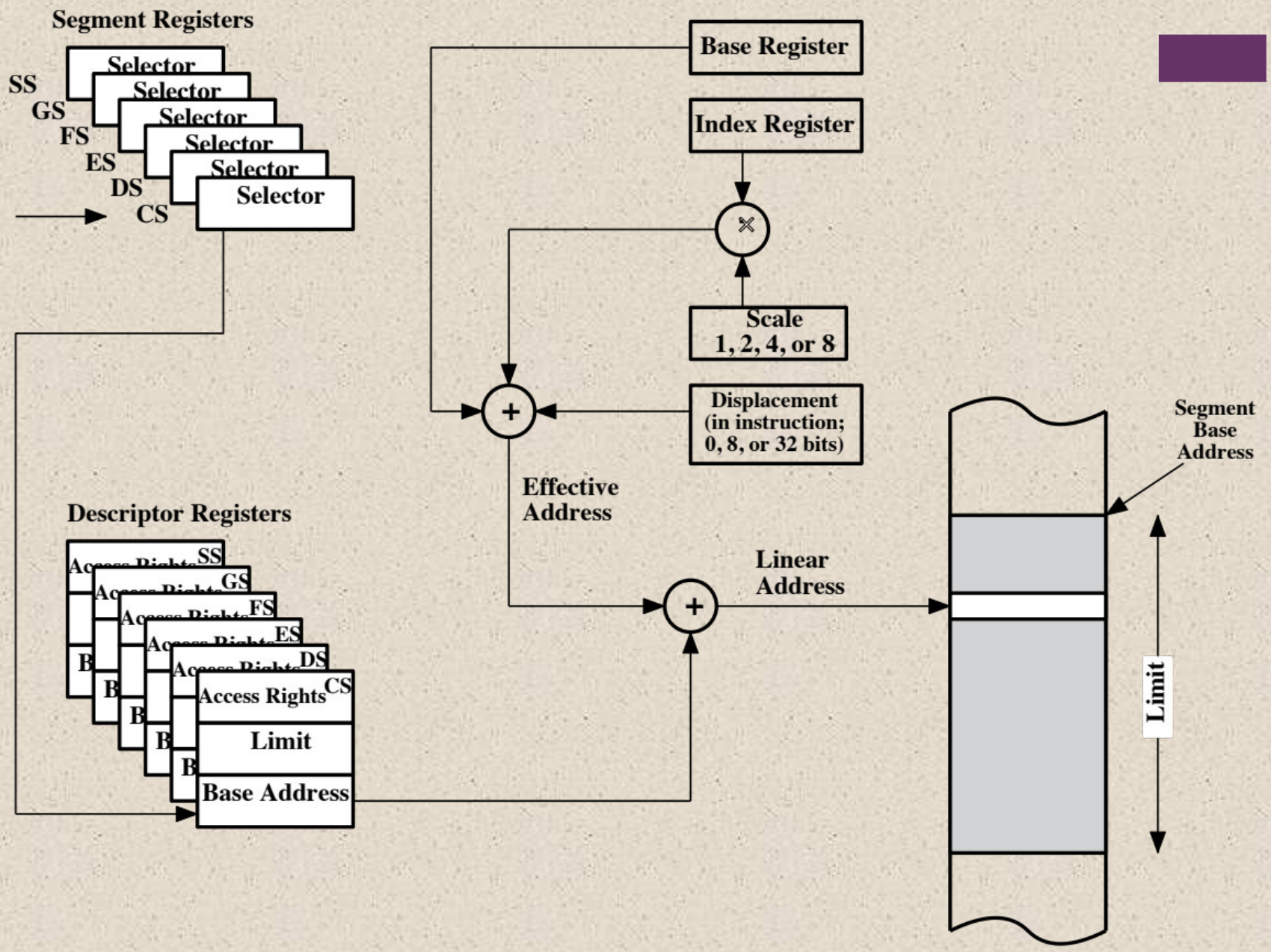


Figure 13.2 x86 Addressing Mode Calculation

Table 13.2
x86 Addressing Modes

Mode	Algorithm
Immediate	Operand = A
Register Operand	LA = R
Displacement	LA = (SR) + A
Base	LA = (SR) + (B)
Base with Displacement	LA = (SR) + (B) + A
Scaled Index with Displacement	LA = (SR) + (I) × S + A
Base with Index and Displacement	LA = (SR) + (B) + (I) + A
Base with Scaled Index and Displacement	LA = (SR) + (I) × S + (B) + A
Relative	LA = (PC) + A

LA = linear address
(X) = contents of X
SR = segment register
PC = program counter
A = contents of an address field in the instruction
R = register
B = base register
I = index register
S = scaling factor

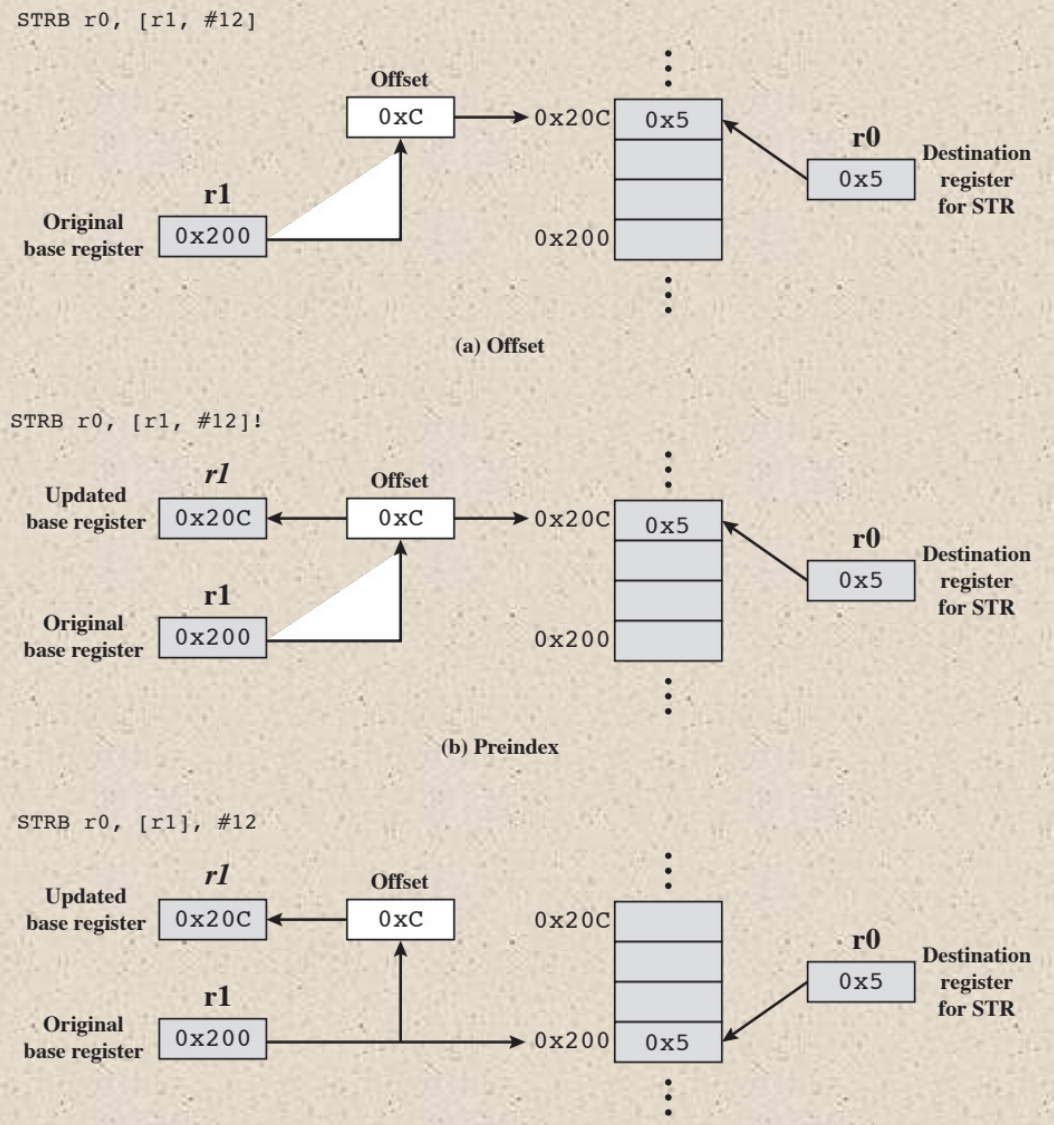


Figure 13.3 ARM Indexing Methods

ARM Data Processing Instruction Addressing and Branch Instructions

- Data processing instructions
  - Use either register addressing or a mixture of register and immediate addressing
  - For register addressing the value in one of the register operands may be scaled using one of the five shift operators

- Branch instructions
  - The only form of addressing for branch instructions is immediate
  - Instruction contains 24 bit value
    - Shifted 2 bits left so that the address is on a word boundary
    - Effective range ± 32MB from from the program counter

```
LDMxx r10, {r0, r1, r4}
STMxx r10, {r0, r1, r4}
```

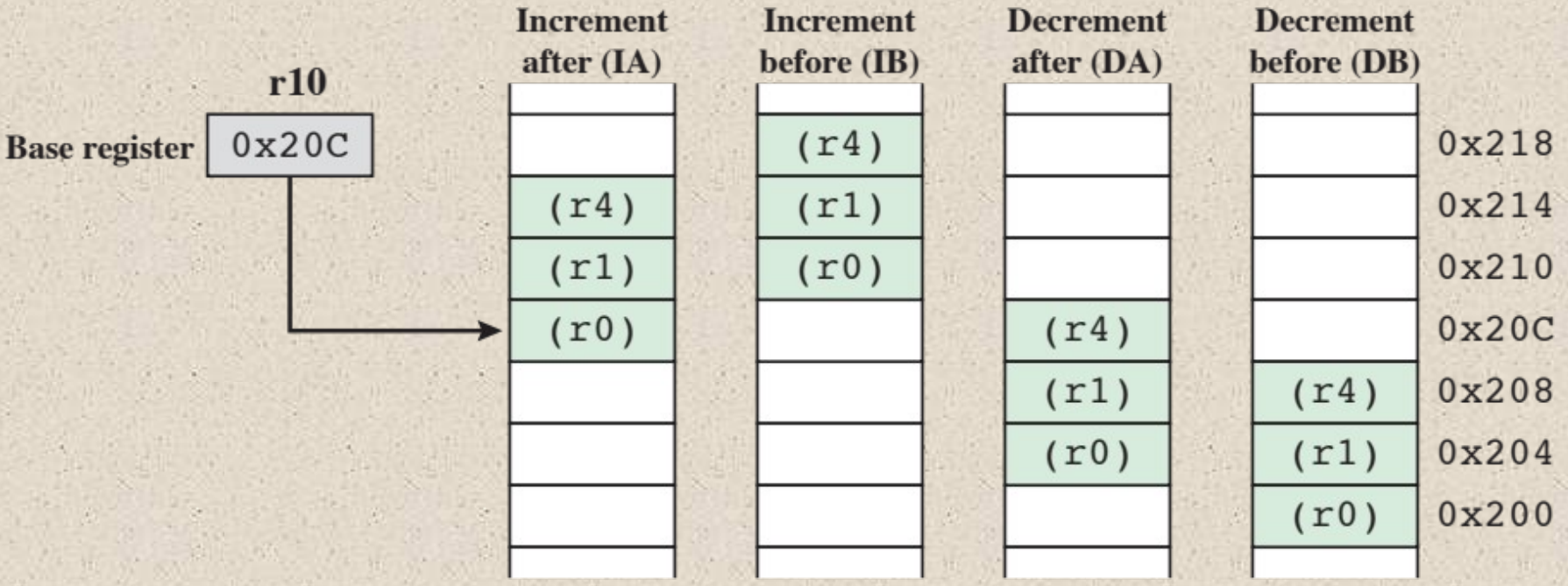


Figure 13.4 ARM Load/Store Multiple Addressing

Instruction Formats

Define the layout of the bits of an instruction, in terms of its constituent fields

Must include an opcode and, implicitly or explicitly, indicate the addressing mode for each operand

For most instruction sets more than one instruction format is used

Instruction Length

- Most basic design issue
- Affects, and is affected by:
  - Memory size
  - Memory organization
  - Bus structure
  - Processor complexity
  - Processor speed
- Should be equal to the memory-transfer length or one should be a multiple of the other
- Should be a multiple of the character length, which is usually 8 bits, and of the length of fixed-point numbers

Allocation of Bits

- Number of addressing modes
- Number of operands
- Register versus memory
- Number of register sets
- Address range
- Address granularity

Memory Reference Instructions

Opcode			D/I	Z/C	Displacement						
0		2	3	4	5						11

Input/Output Instructions

1	1	0	Device							Opcode	
0		2	3						8	9	11

Register Reference Instructions

Group 1 Microinstructions

1	1	1	0	CLA	CLL	CMA	CML	RAR	RAL	BSW	IAC
0	1	2	3	4	5	6	7	8	9	10	11

Group 2 Microinstructions

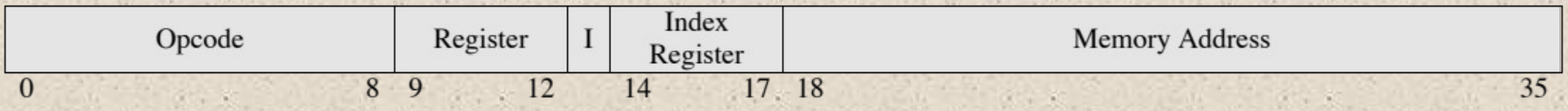
1	1	1	1	CLA	SMA	SZA	SNL	RSS	OSR	HLT	0
0	1	2	3	4	5	6	7	8	9	10	11

Group 3 Microinstructions

1	1	1	1	CLA	MQA	0	MQL	0	0	0	1
0	1	2	3	4	5	6	7	8	9	10	11

D/I = Direct/Indirect address
Z/C = Page 0 or Current page
CLA = Clear Accumulator
CLL = Clear Link
CMA = CoMplement Accumulator
CML = CoMplement Link
RAR = Rotate Accumultator Right
RAL = Rotate Accumulator Left
BSW = Byte SWap
IAC = Increment ACcumulator
SMA = Skip on Minus Accumulator
SZA = Skip on Zero Accumulator
SNL = Skip on Nonzero Link
RSS = Reverse Skip Sense
OSR = Or with Switch Register
HLT = HaLT
MQA = Multiplier Quotient into Accumulator
MQL = Multiplier Quotient Load

Figure 13.5 PDP-8 Instruction Formats

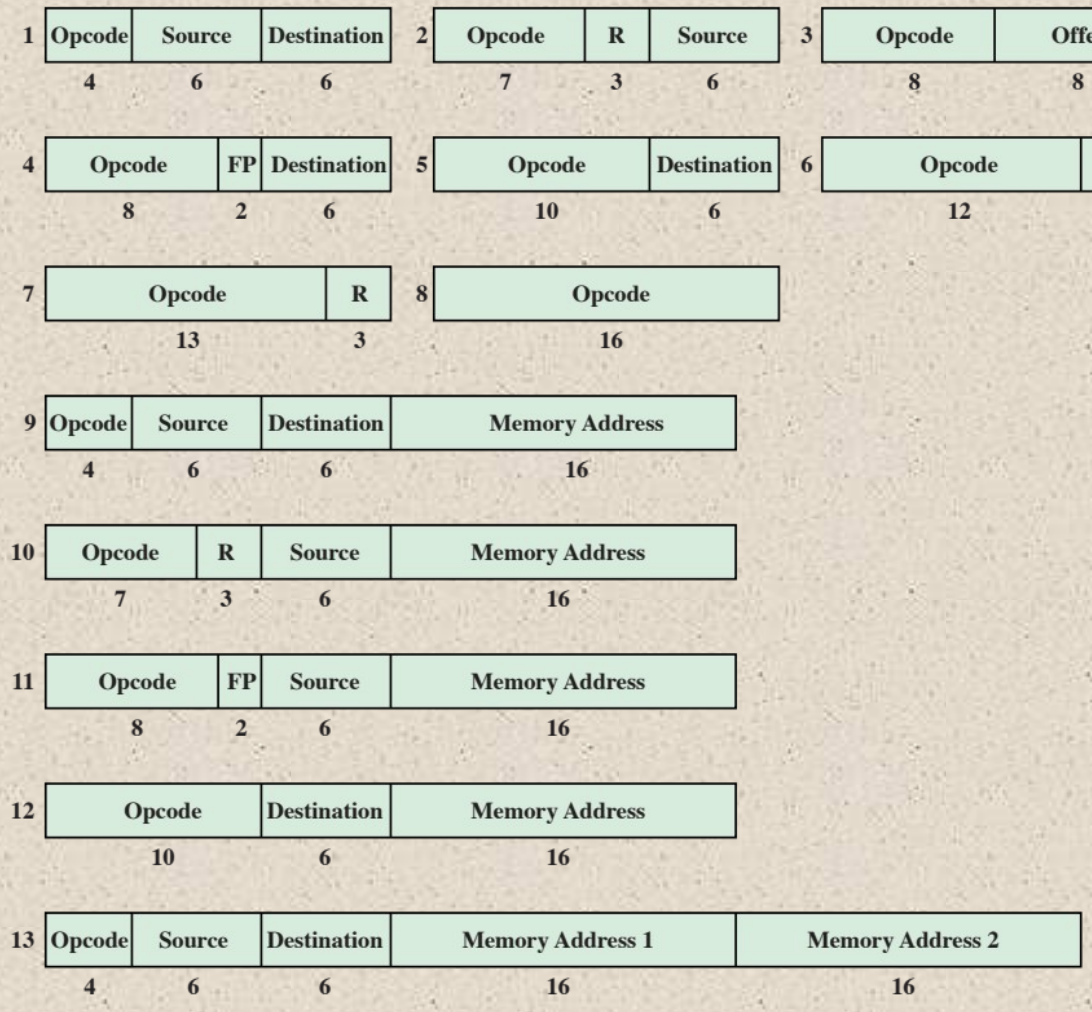


I = indirect bit

Figure 13.6 PDP-10 Instruction Format

Variable-Length Instructions

- Variations can be provided efficiently and compactly
- Increases the complexity of the processor
- Does not remove the desirability of making all of the instruction lengths integrally related to word length
  - Because the processor does not know the length of the next instruction to be fetched a typical strategy is to fetch a number of bytes or words equal to at least the longest possible instruction
  - Sometimes multiple instructions are fetched



Numbers below fields indicate bit length
Source and Destination each contain a 3-bit addressing mode field and a 3-bit register number
FP indicates one of four floating-point registers
R indicates one of the general-purpose registers
CC is the condition code field

Figure 13.7 Instruction Formats for the PDP-11

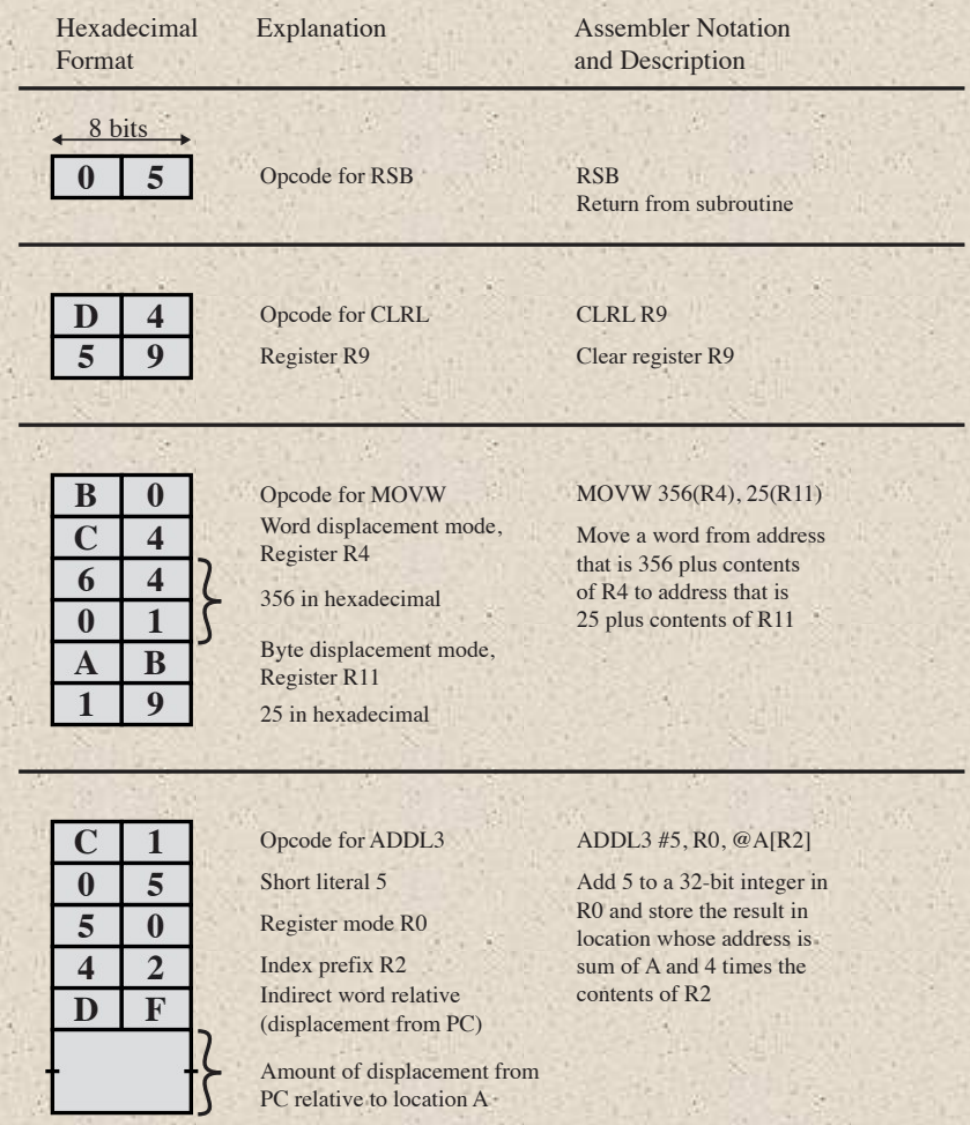


Figure 13.8 Examples of VAX Instructions

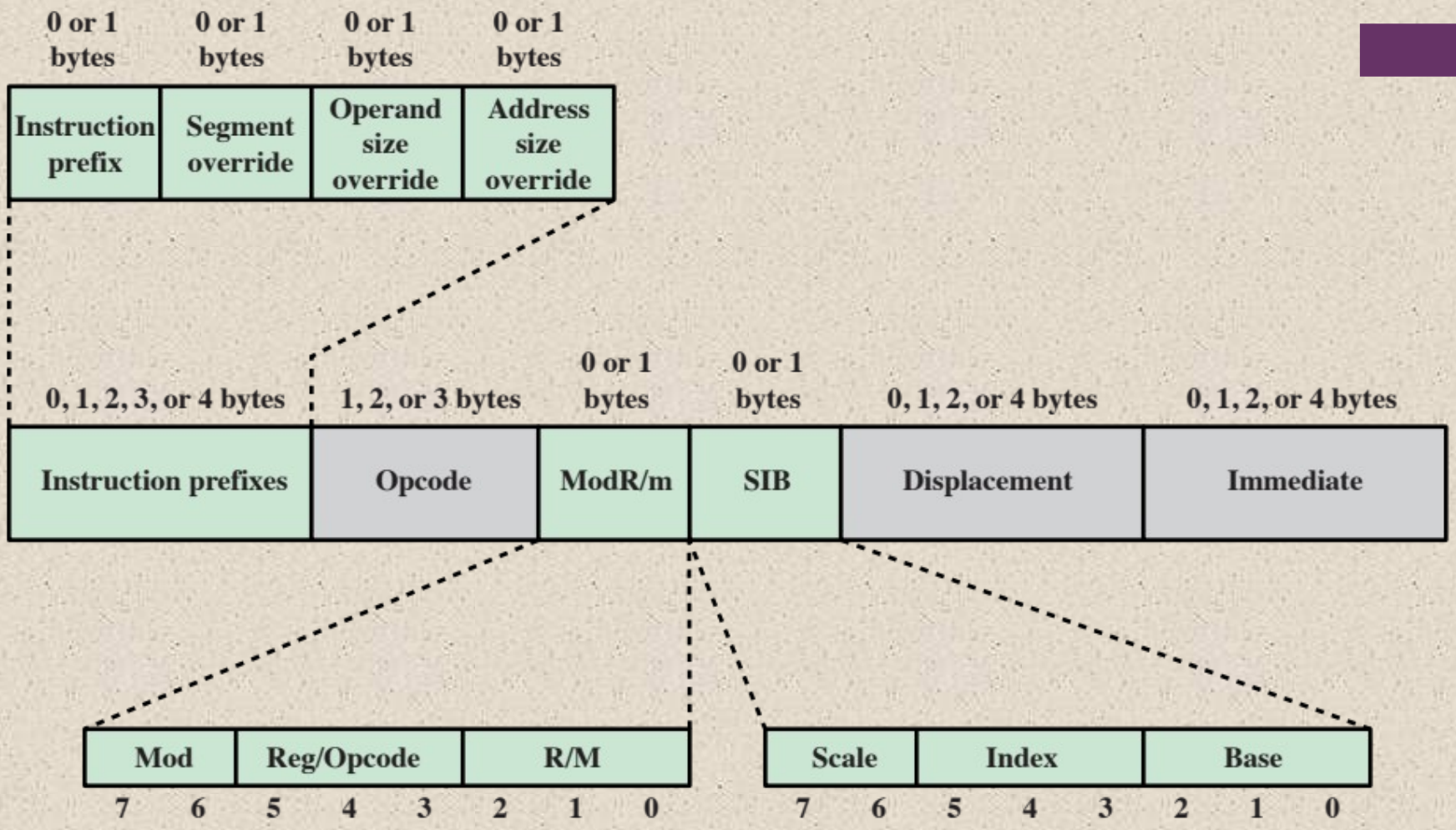
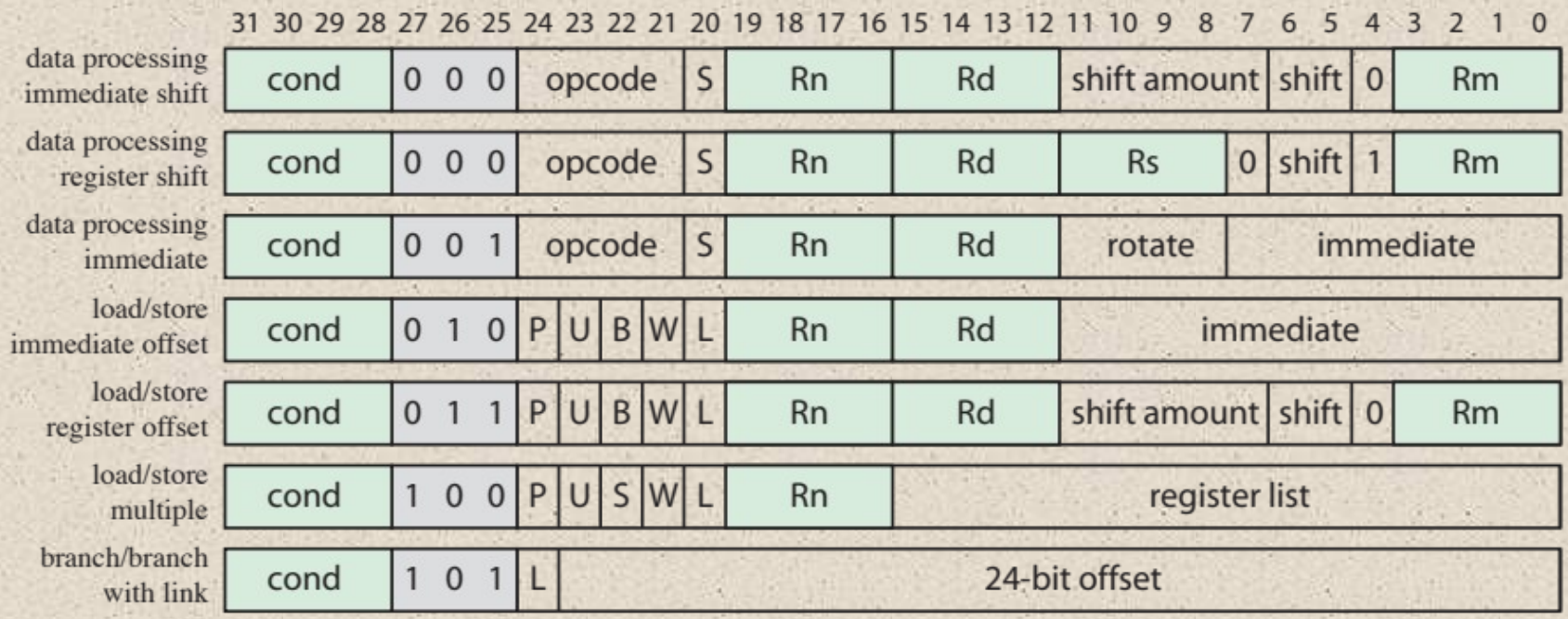


Figure 13.9 x86 Instruction Format



S = For data processing instructions, signifies that the instruction updates the condition codes

S = For load/store multiple instructions, signifies whether instruction execution is restricted to supervisor mode

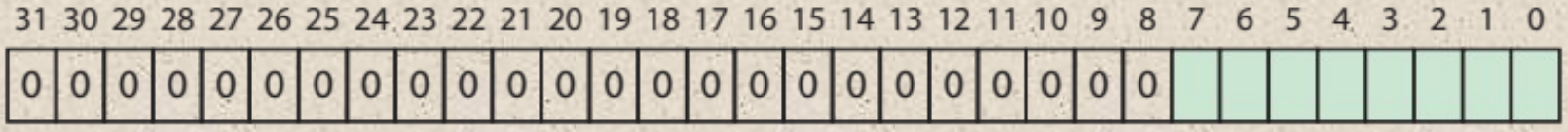
P, U, W = bits that distinguish among different types of addressing_mode

B = Distinguishes between an unsigned byte (B==1) and a word (B==0) access

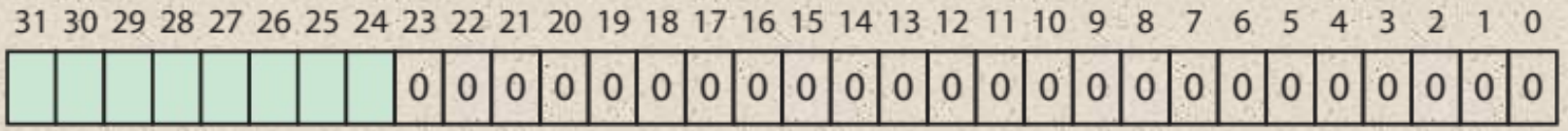
L = For load/store instructions, distinguishes between a Load (L==1) and a Store (L==0)

L = For branch instructions, determines whether a return address is stored in the link register

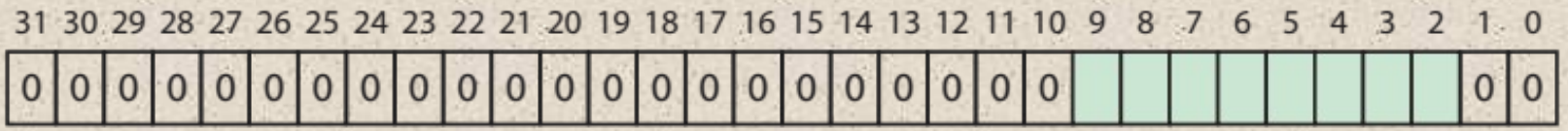
Figure 13.10 ARM Instruction Formats



ror #0 - range 0 through 0x000000FF - step 0x00000001



ror #8 - range 0 through 0xFF000000 - step 0x01000000



ror #30 - range 0 through 0x000003FC - step 0x00000004

Figure 13.11 Examples of Use of ARM Immediate Constants

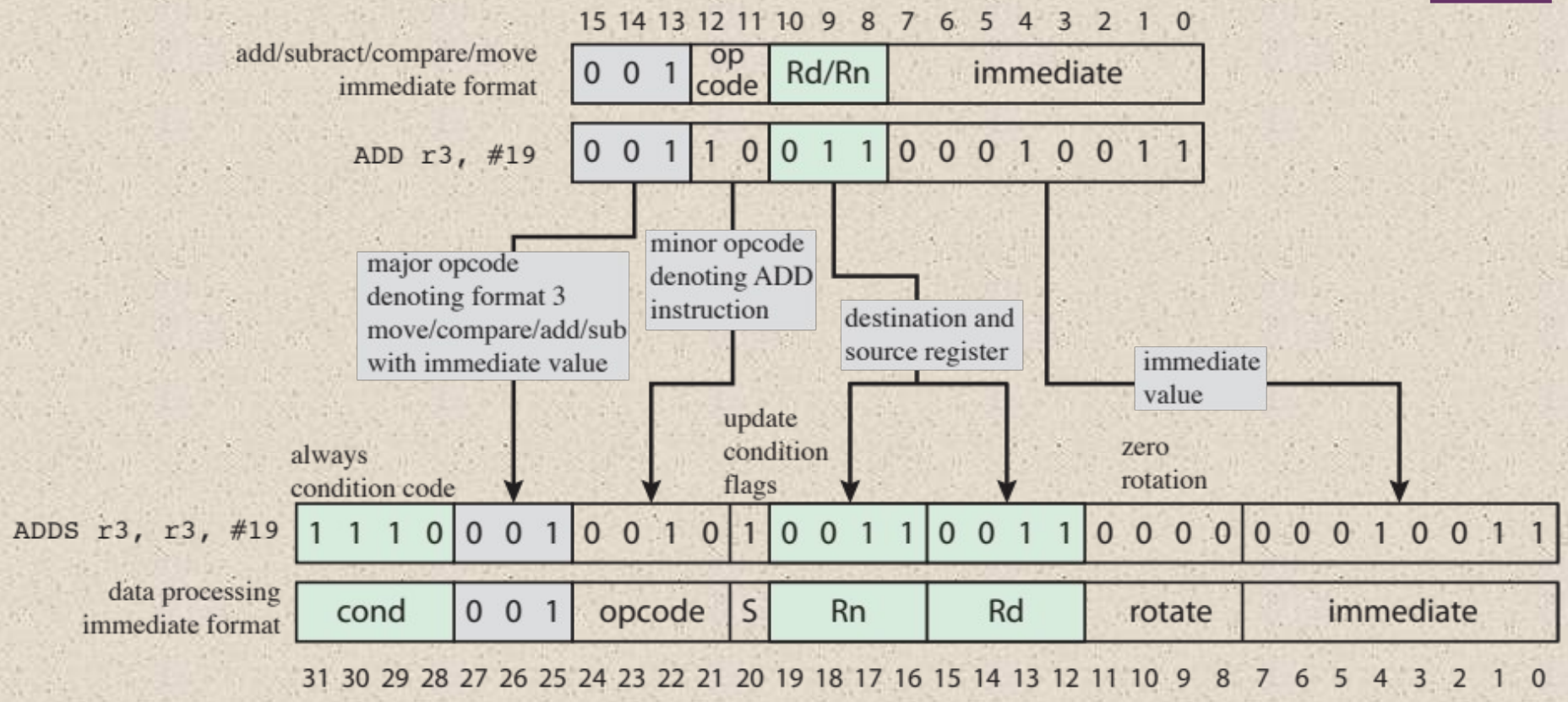
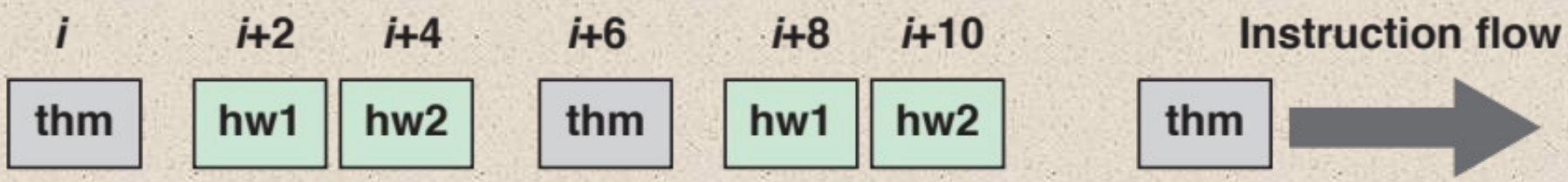


Figure 13.12 Expanding a Thumb ADD Instruction into its ARM Equivalent

Thumb-2 Instruction Set

- The only instruction set available on the Cortex-M microcontroller products
- Is a major enhancement to the Thumb instruction set architecture (ISA)
  - Introduces 32-bit instructions that can be intermixed freely with the older 16-bit Thumb instructions
  - Most 32-bit Thumb instructions are unconditional, whereas almost all ARM instructions can be conditional
  - Introduces a new If-Then (IT) instruction that delivers much of the functionality of the condition field in ARM instructions
- Delivers overall code density comparable with Thumb, together with the performance levels associated with the ARM ISA
- Before Thumb-2 developers had to choose between Thumb for size and ARM for performance



Halfword 1 [15:13]	Halfword1 [12:11]	Length	Functionality
Not 111	xx	16 bits (1 halfword)	16-bit Thumb instruction
111	00	16 bits (1 halfword)	16-bit Thumb unconditional branch instruction
111	Not 00	32 bits (2 halfwords)	32-bit Thumb-2 instruction

Figure 13.13 Thumb-2 Encoding

Address	Contents			
101	0010	0010	101	2201
102	0001	0010	102	1202
103	0001	0010	103	1203
104	0011	0010	104	3204
201	0000	0000	201	0002
202	0000	0000	202	0003
203	0000	0000	203	0004
204	0000	0000	204	0000

(a) Binary program

Address	Contents
101	2201
102	1202
103	1203
104	3204
201	0002
202	0003
203	0004
204	0000

(b) Hexadecimal program

Address	Instruction	
101	LDA	201
102	ADD	202
103	ADD	203
104	STA	204
201	DAT	2
202	DAT	3
203	DAT	4
204	DAT	0

(c) Symbolic program

Label	Operation	Operand
FORMUL	LDA	I
	ADD	J
	ADD	K
	STA	N
I	DATA	2
J	DATA	3
K	DATA	4
N	DATA	0

(d) Assembly program

Figure 13.14 Computation of the Formula N = I + J+ K

Summary

Chapter 13

Instruction Sets: Addressing Modes and Formats

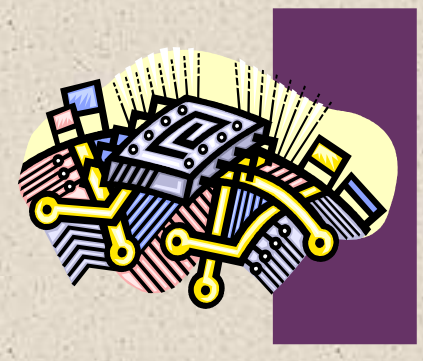
- Addressing modes
  - Immediate addressing
  - Direct addressing
  - Indirect addressing
  - Register addressing
  - Register indirect addressing
  - Displacement addressing
  - Stack addressing
- Assembly language
- x86 addressing modes
- ARM addressing modes
- Instruction formats
  - Instruction length
  - Allocation of bits
  - Variable-length instructions
- X86 instruction formats
- ARM instruction formats

William Stallings
Computer Organization and Architecture
10th Edition

Chapter 14

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

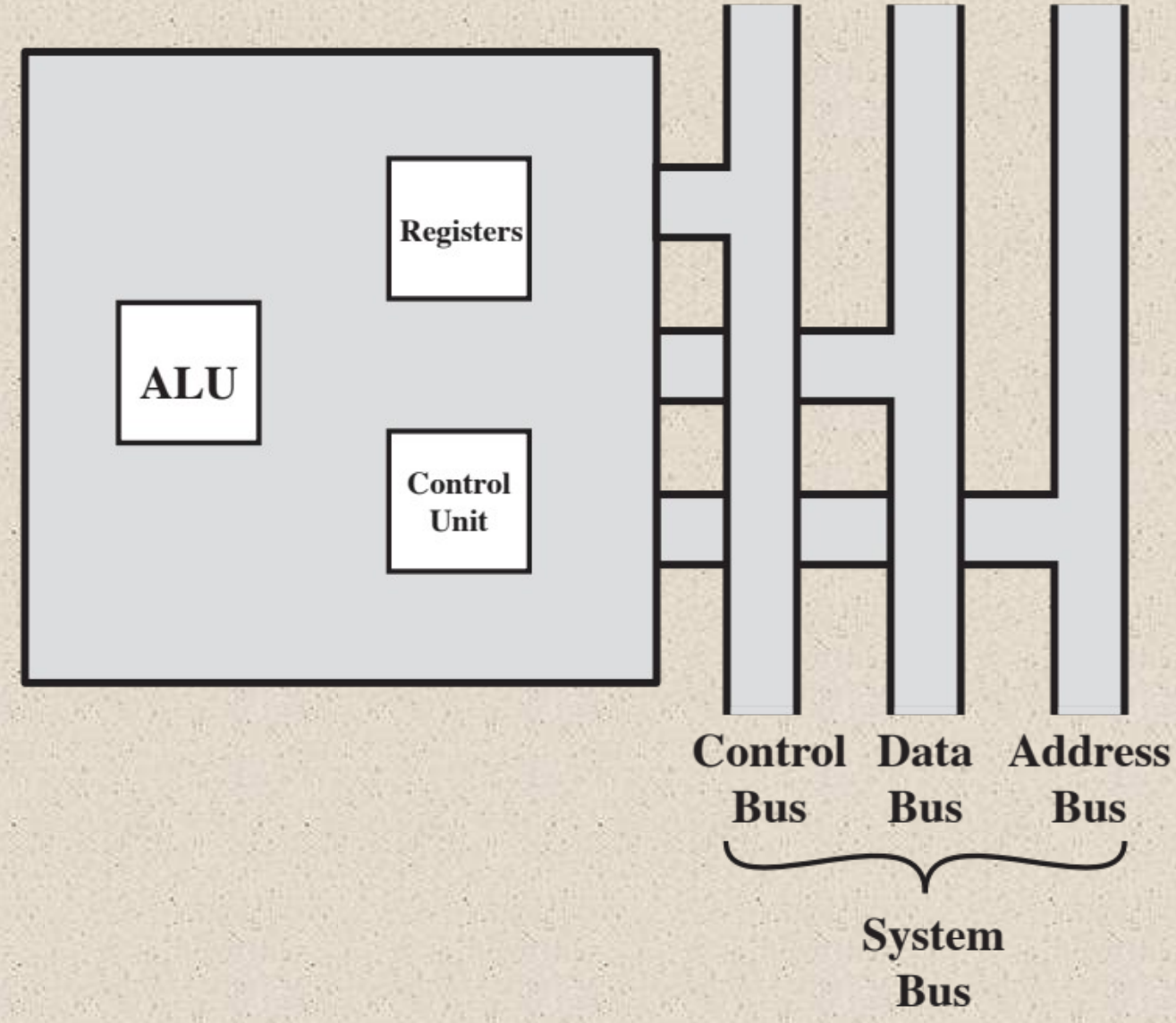


Figure 14.1 The CPU with the System Bus

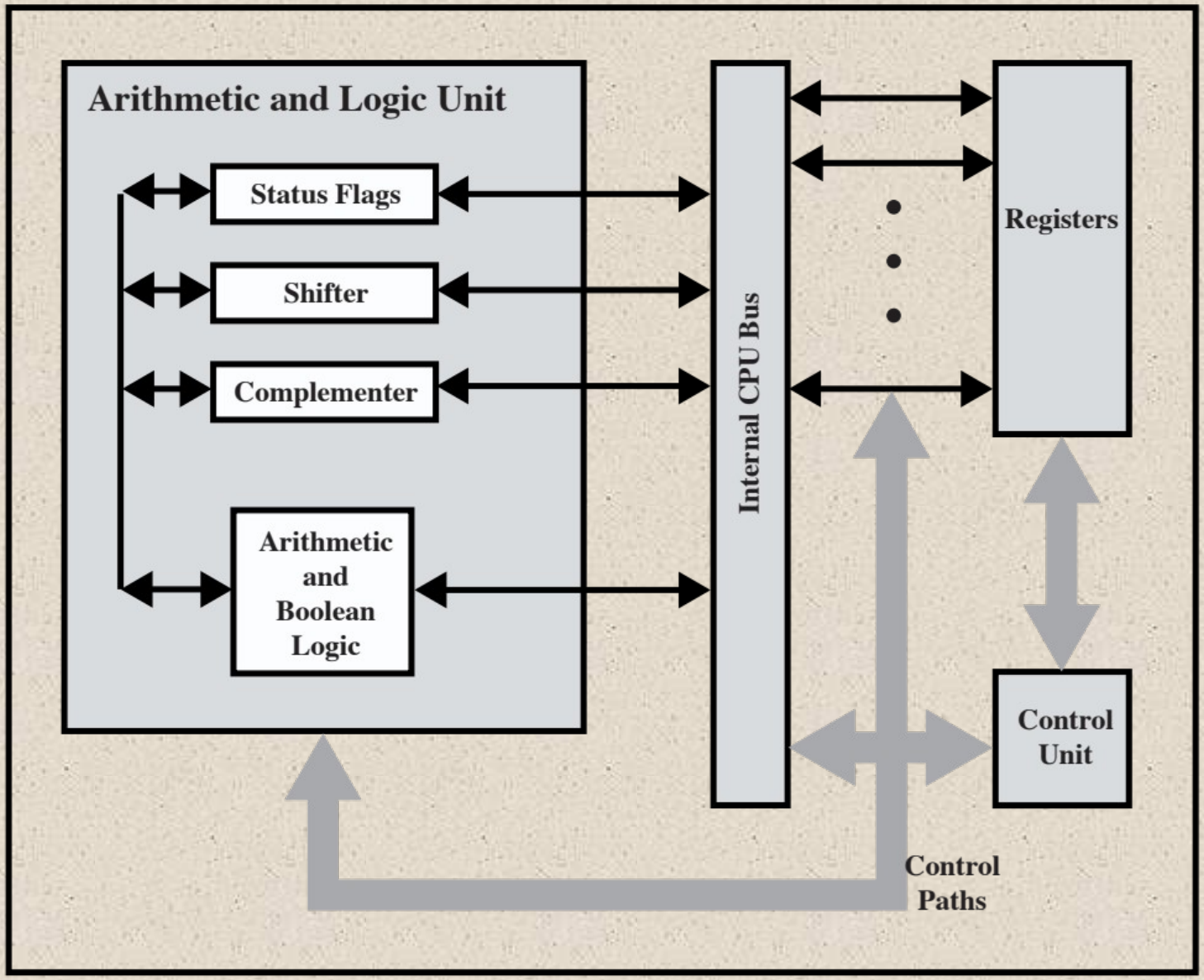


Figure 14.2 Internal Structure of the CPU

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 14.1
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.	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. Conditional instructions, such as BRANCH are simplified relative to composite instructions, such as TEST AND BRANCH.	2. Condition codes are irregular; they are typically not part of the main data path, so they require extra hardware connections.
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.	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. Condition codes can be saved on the stack during subroutine calls along with other register information.	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

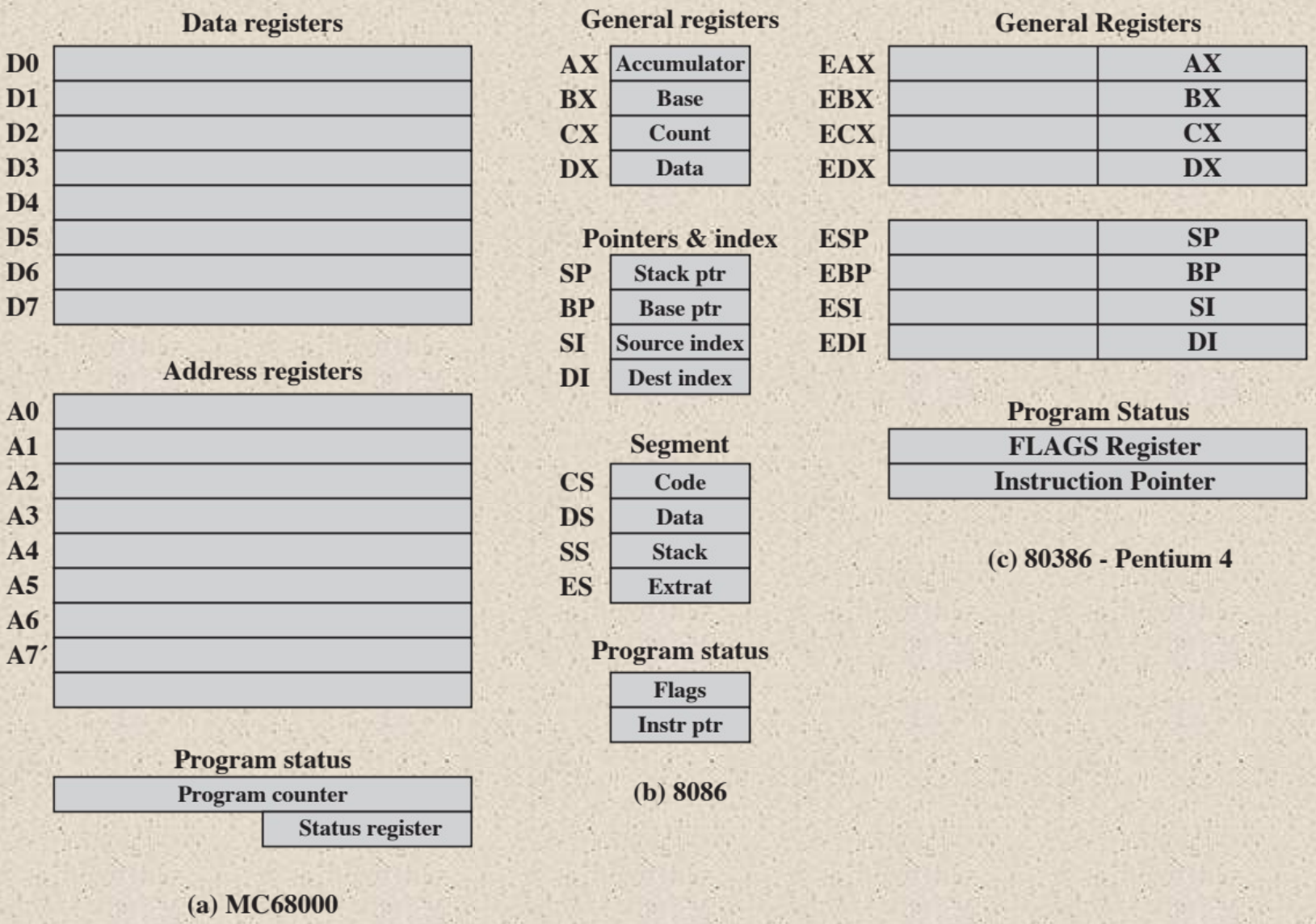
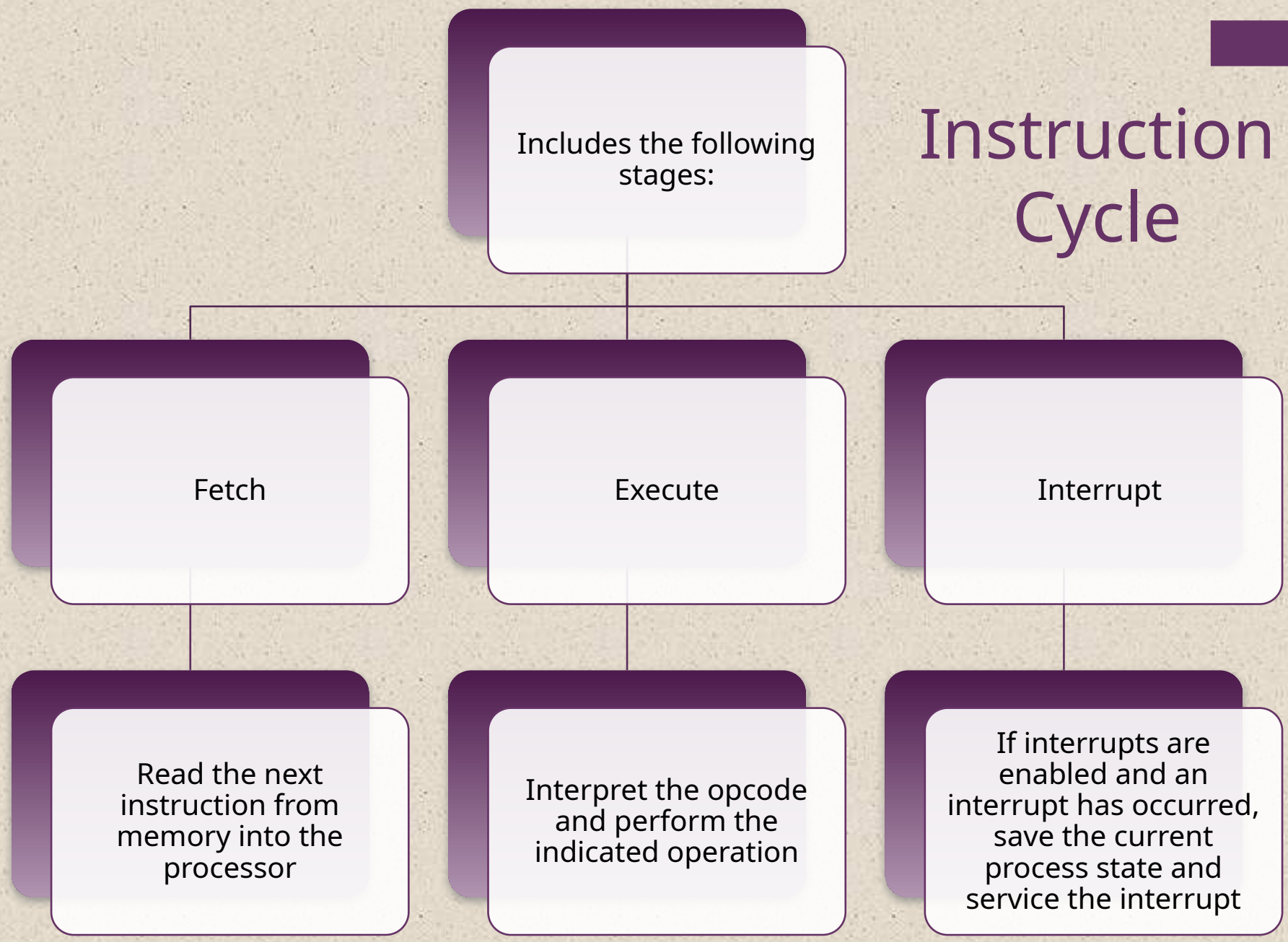


Figure 14.3 Example Microprocessor Register Organizations

Instruction Cycle



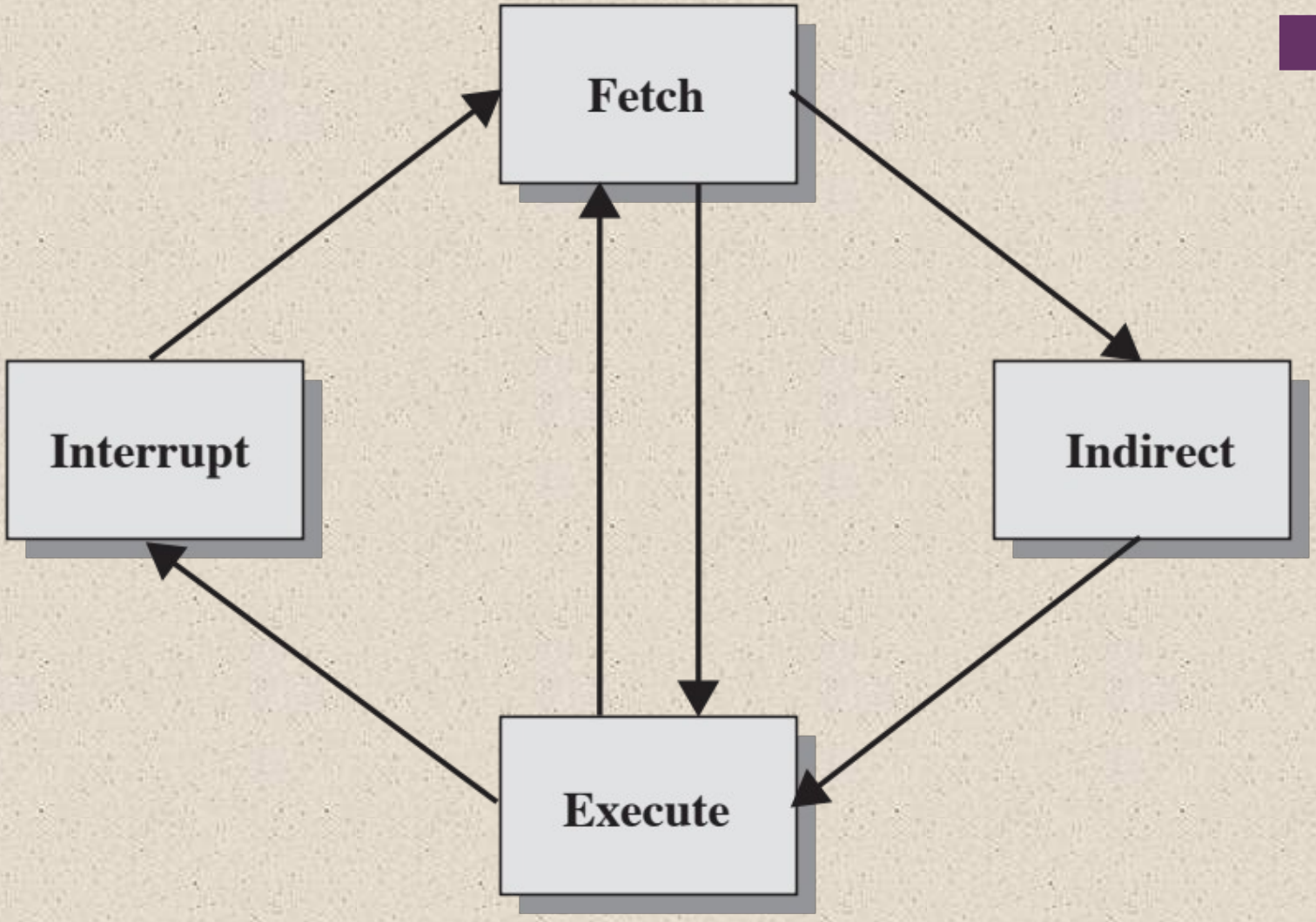


Figure 14.4 The Instruction Cycle

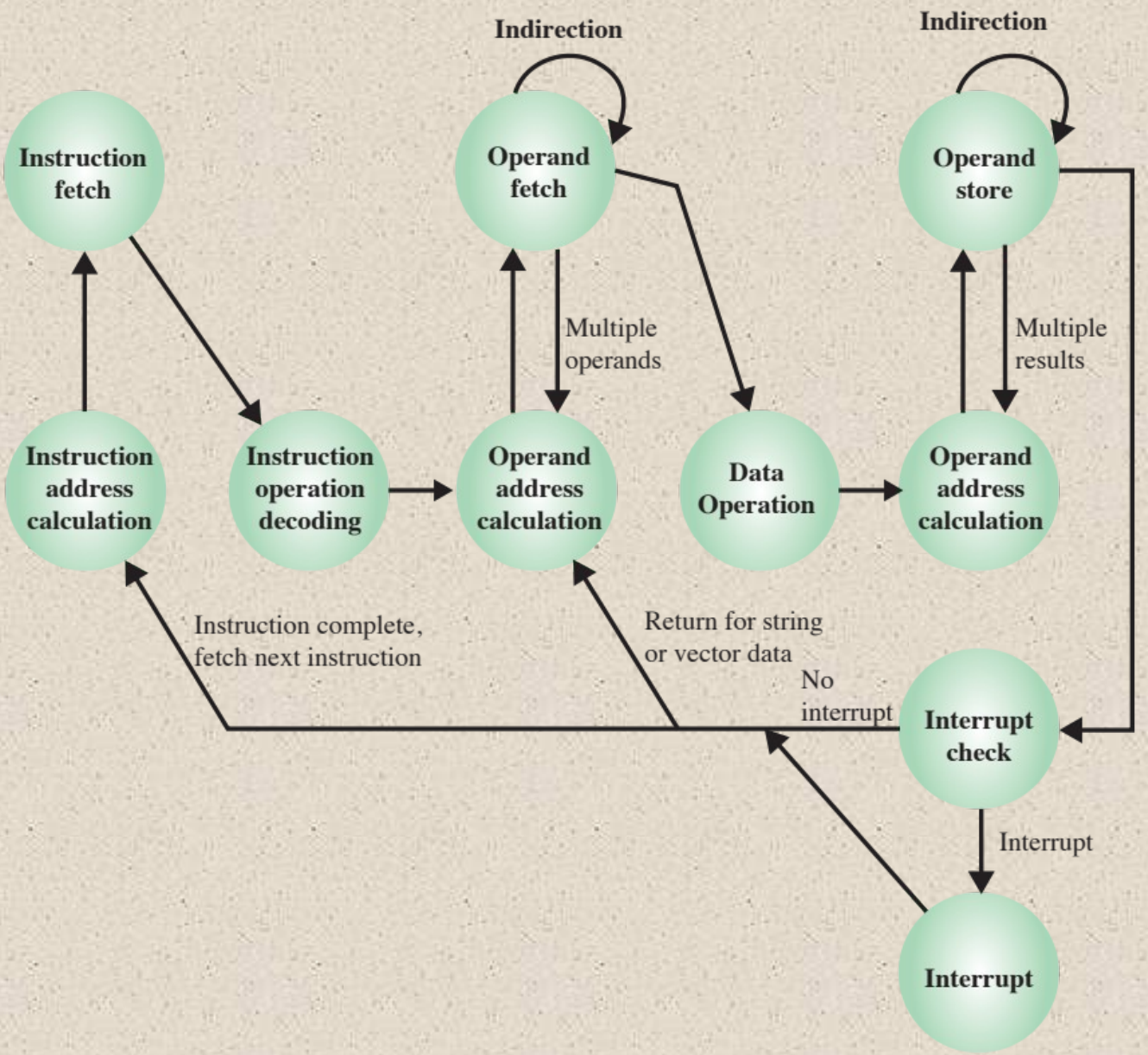
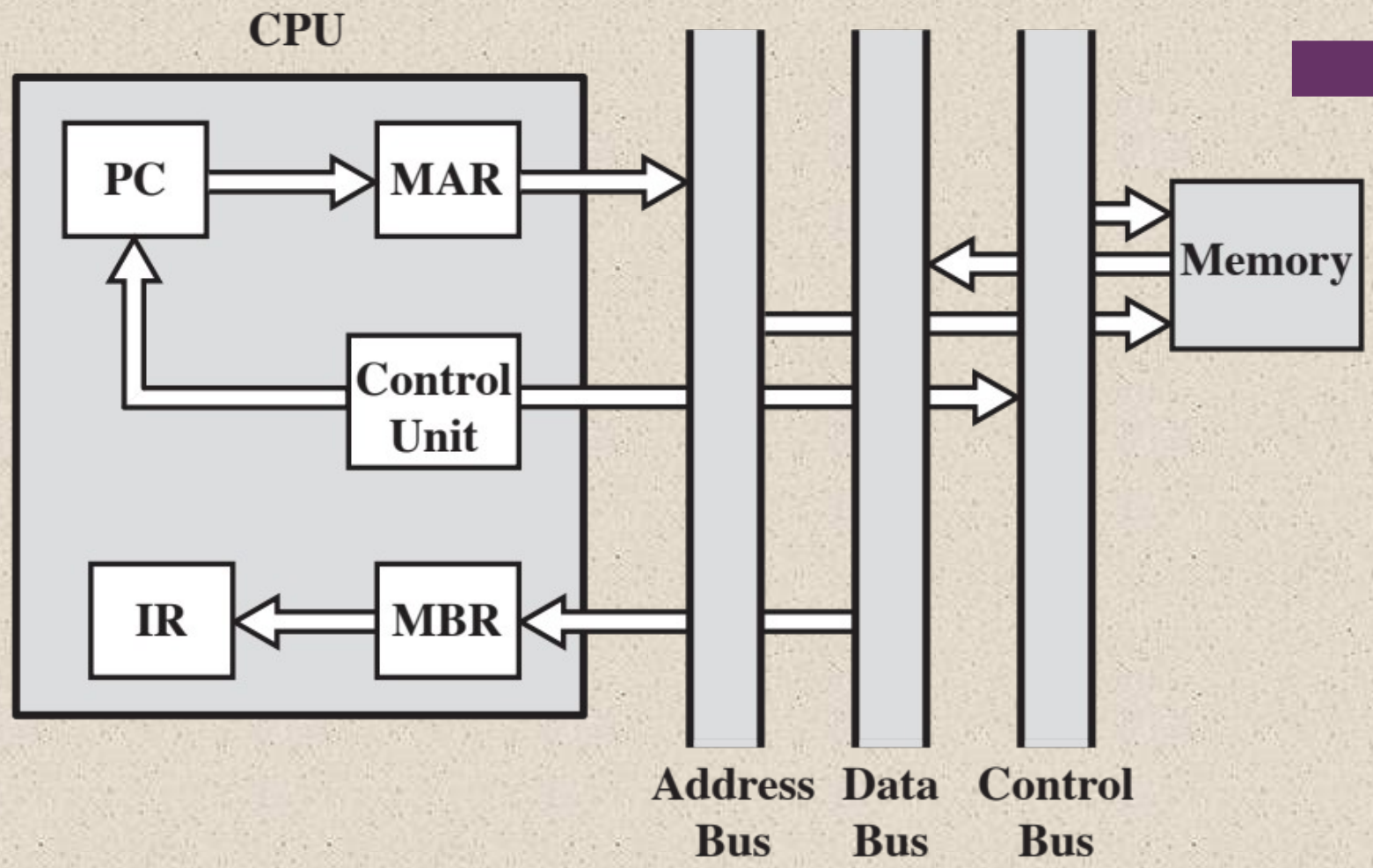


Figure 14.5 Instruction Cycle State Diagram



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

Figure 14.6 Data Flow, Fetch Cycle

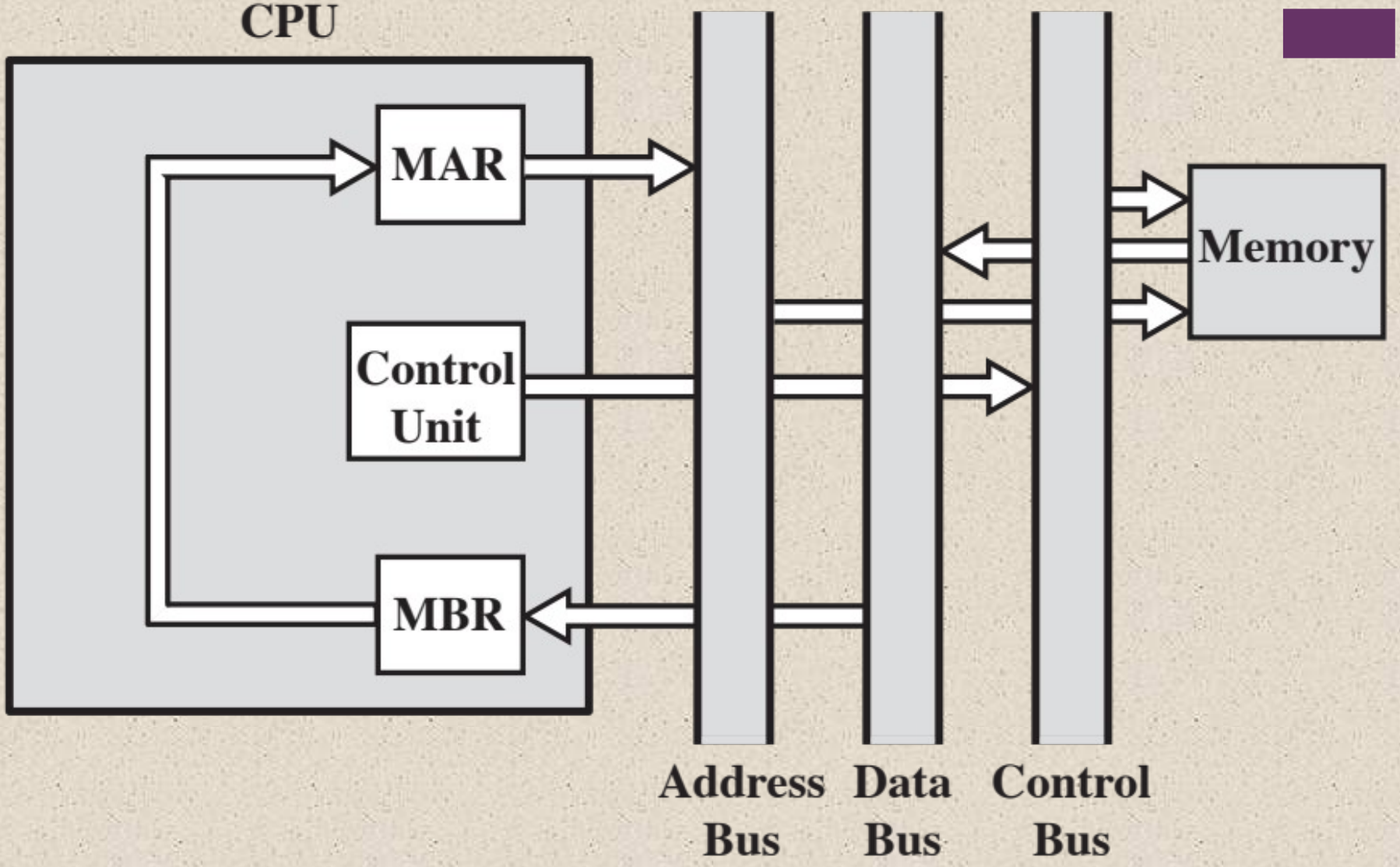


Figure 14.7 Data Flow, Indirect Cycle

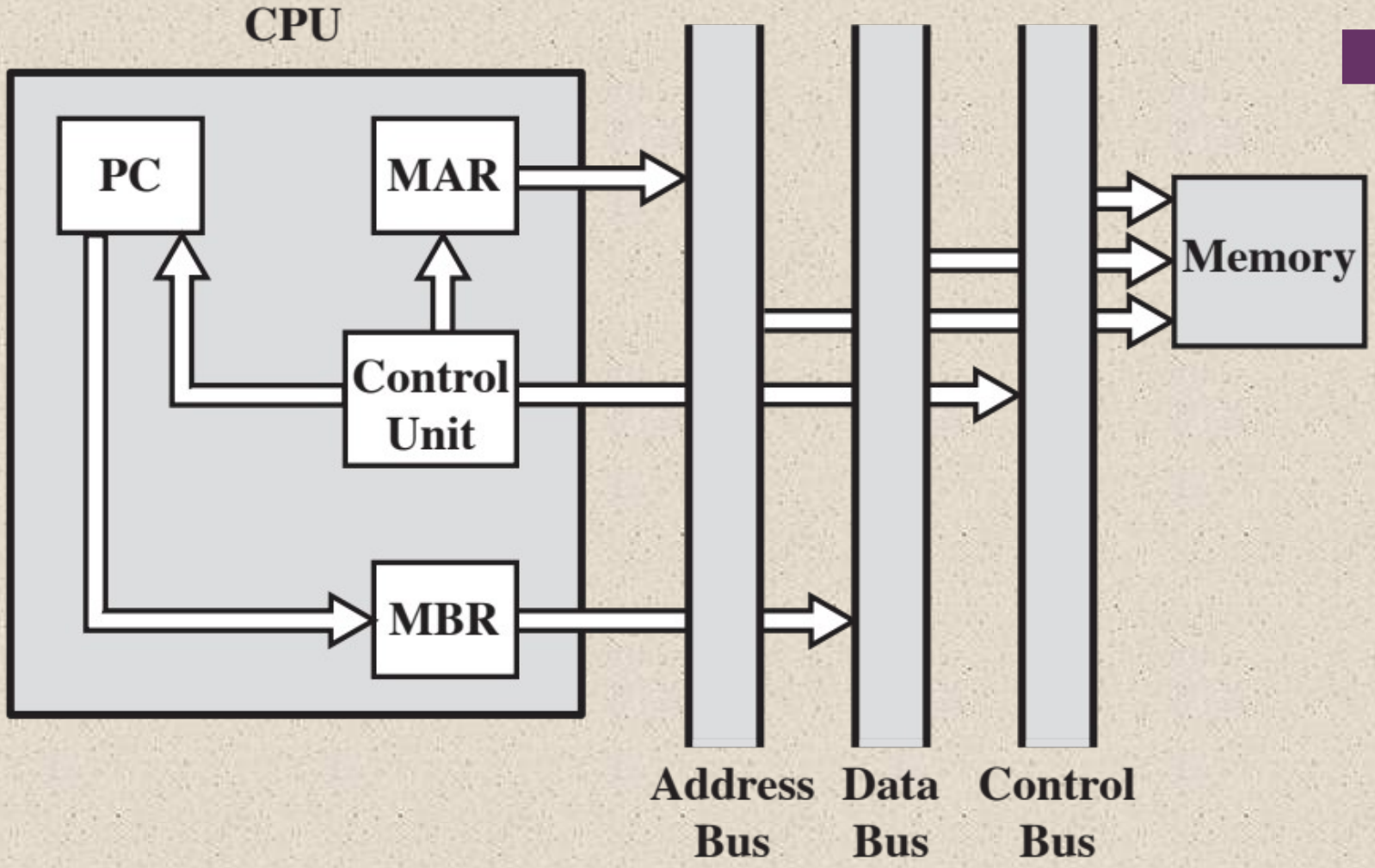
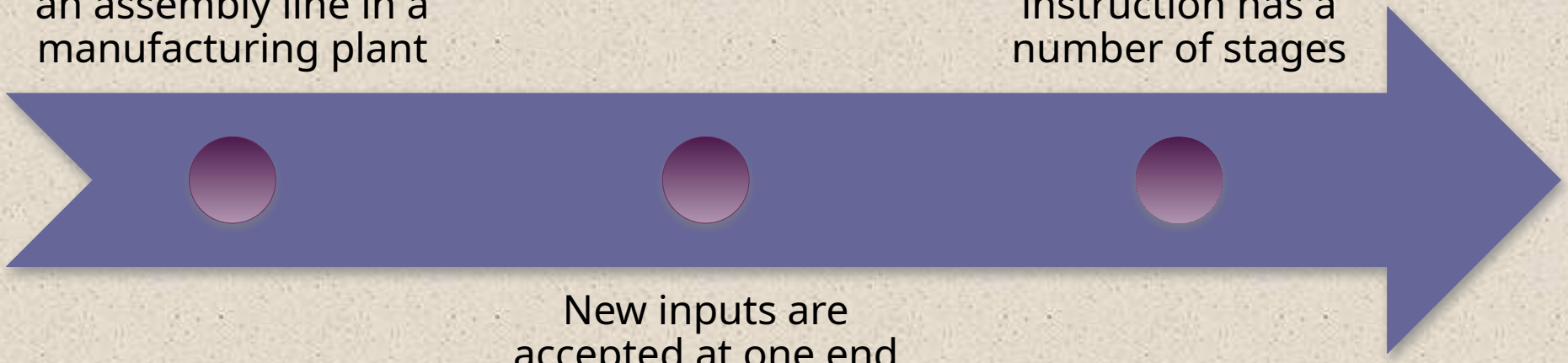


Figure 14.8 Data Flow, Interrupt Cycle

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

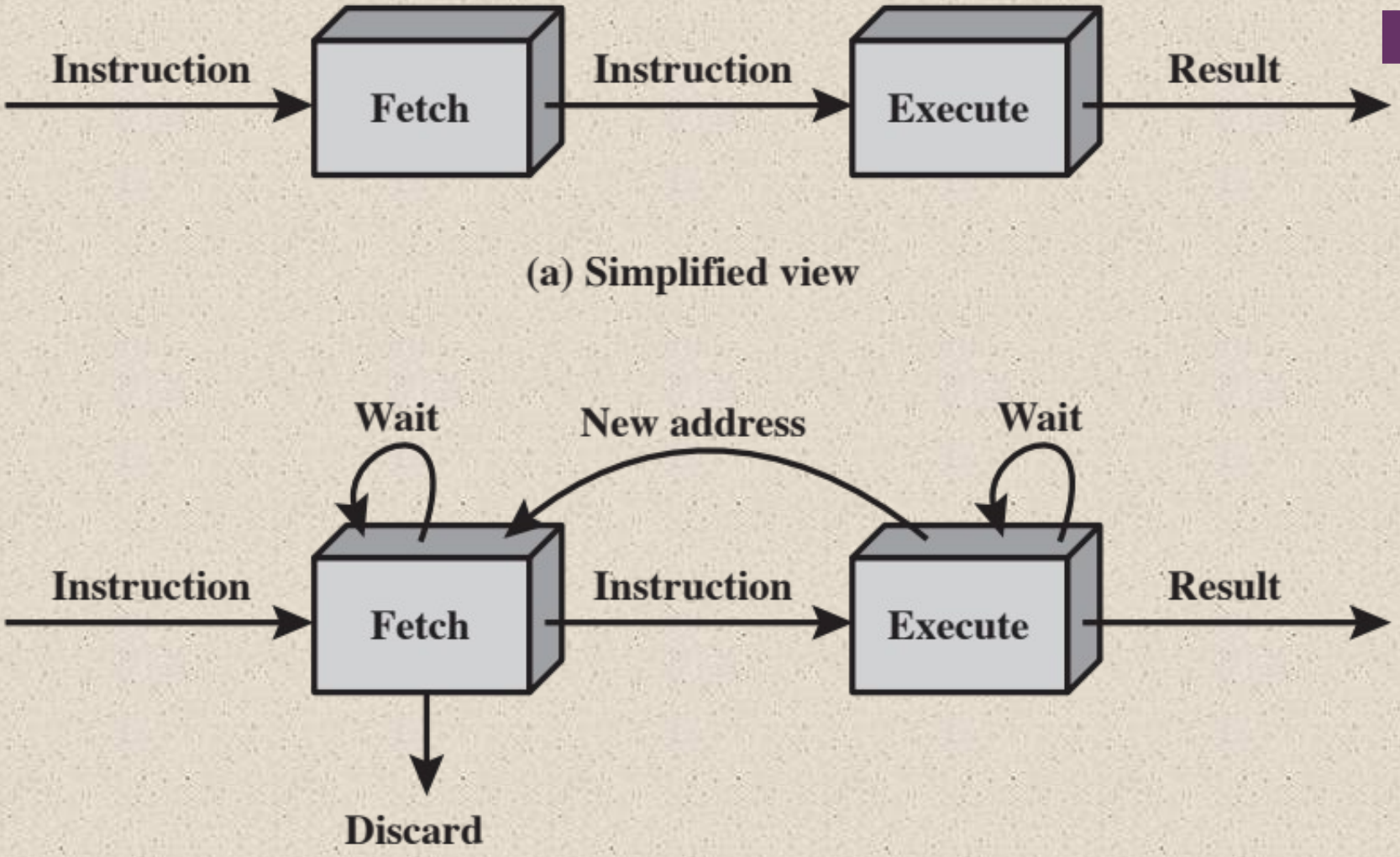


Figure 14.9 Two-Stage Instruction Pipeline

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

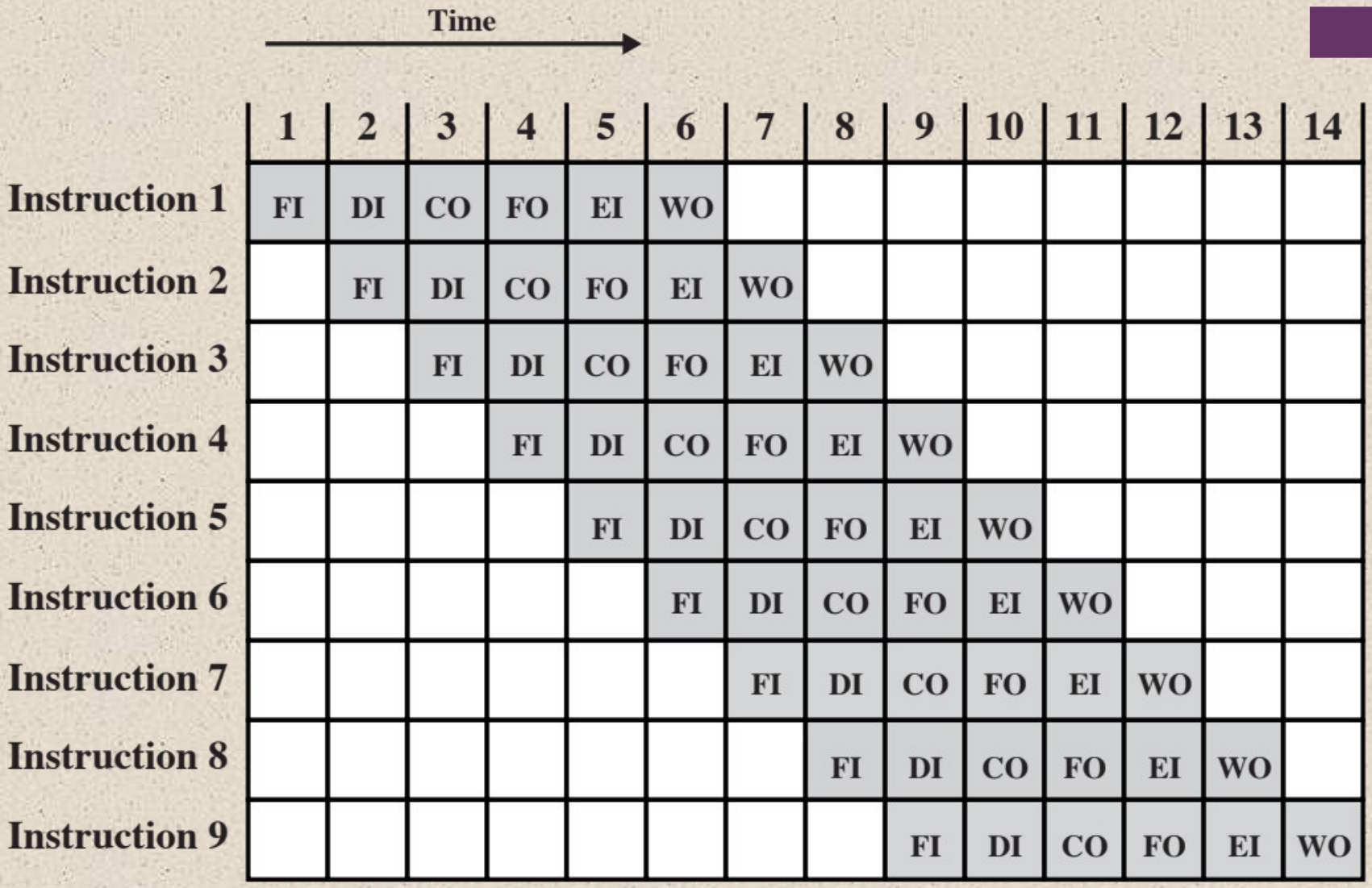


Figure 14.10 Timing Diagram for Instruction Pipeline Operation

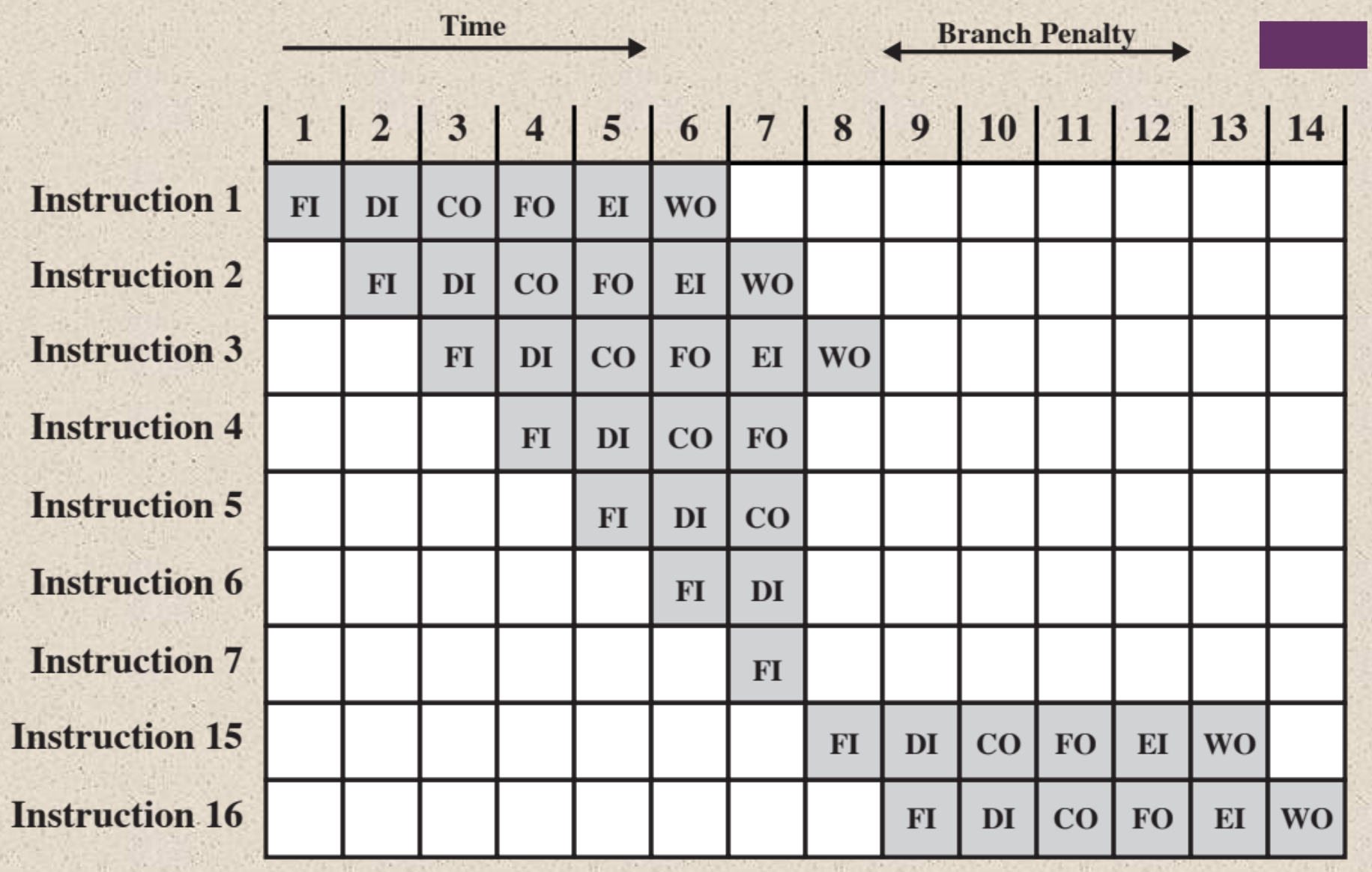


Figure 14.11 The Effect of a Conditional Branch on Instruction Pipeline Operation

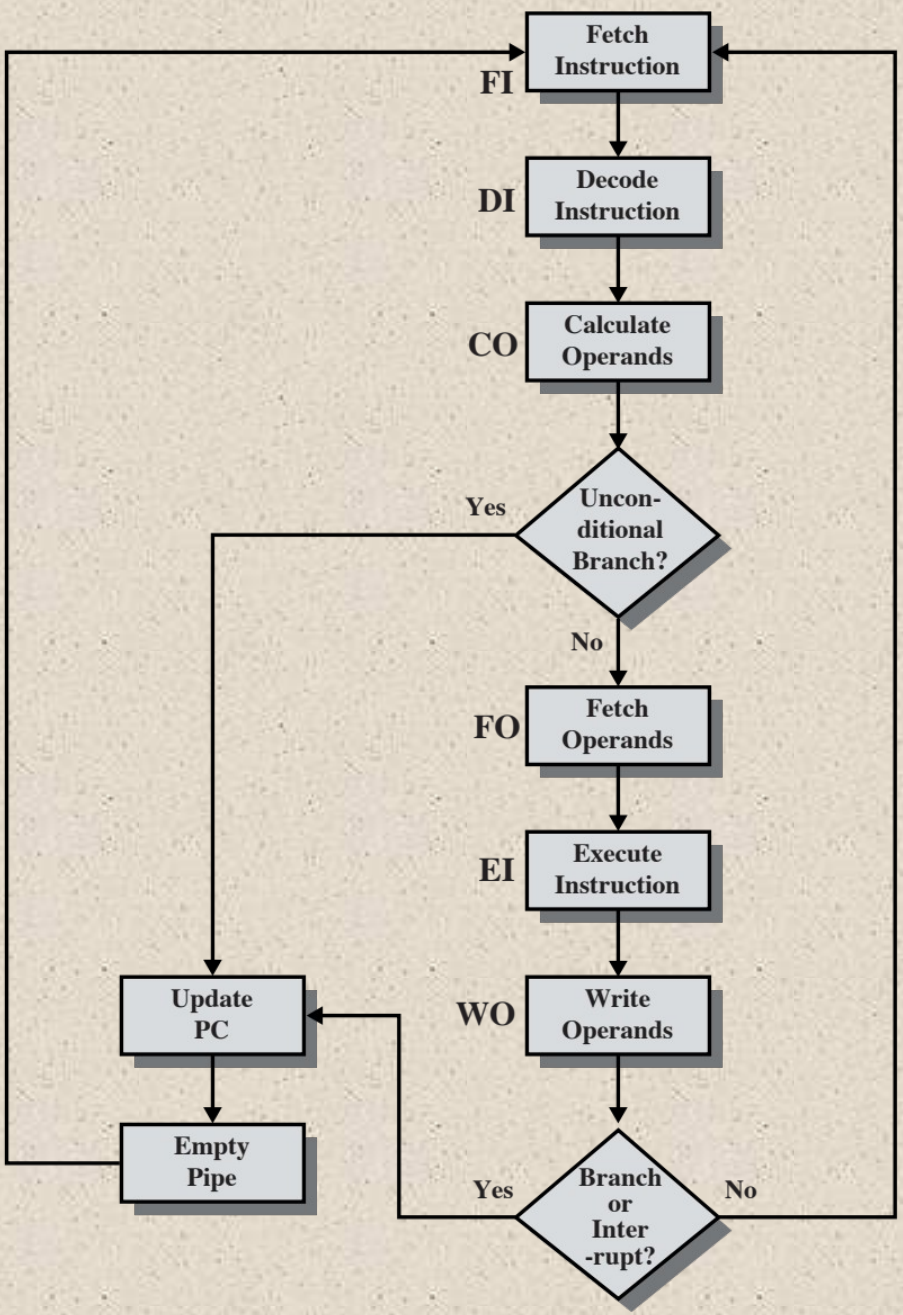


Figure 14.12 Six-Stage Instruction Pipeline

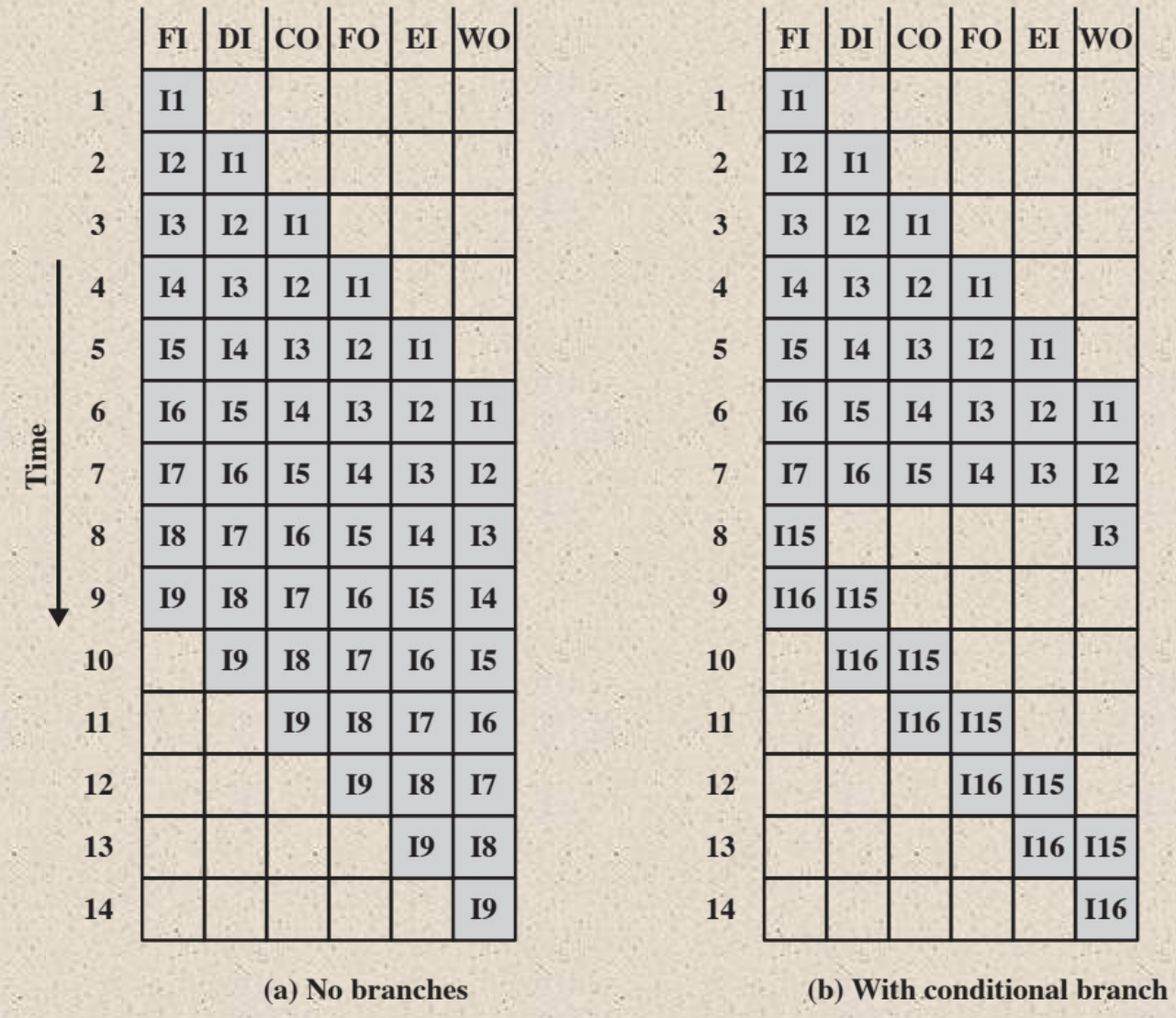
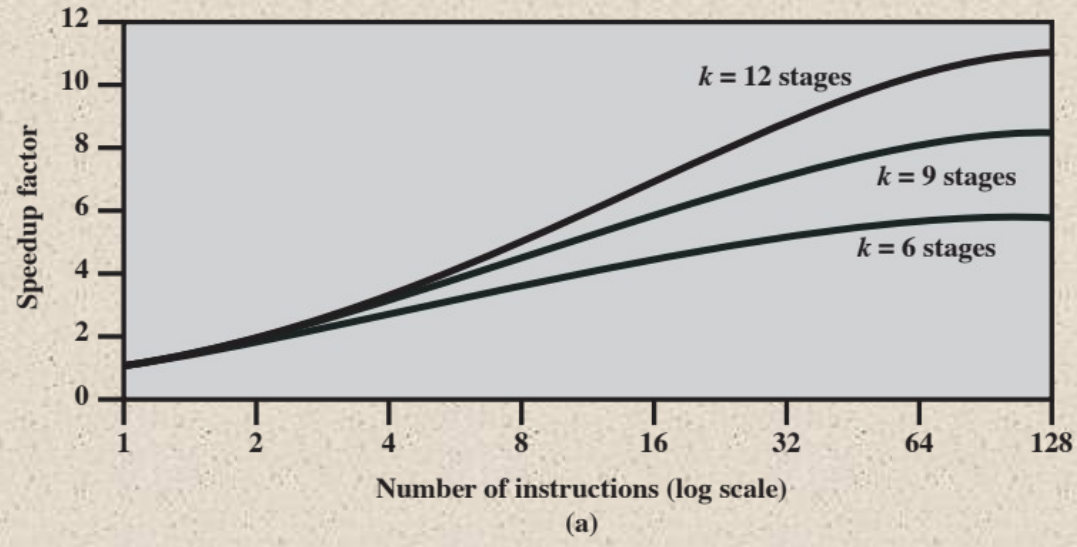


Figure 14.13 An Alternative Pipeline Depiction



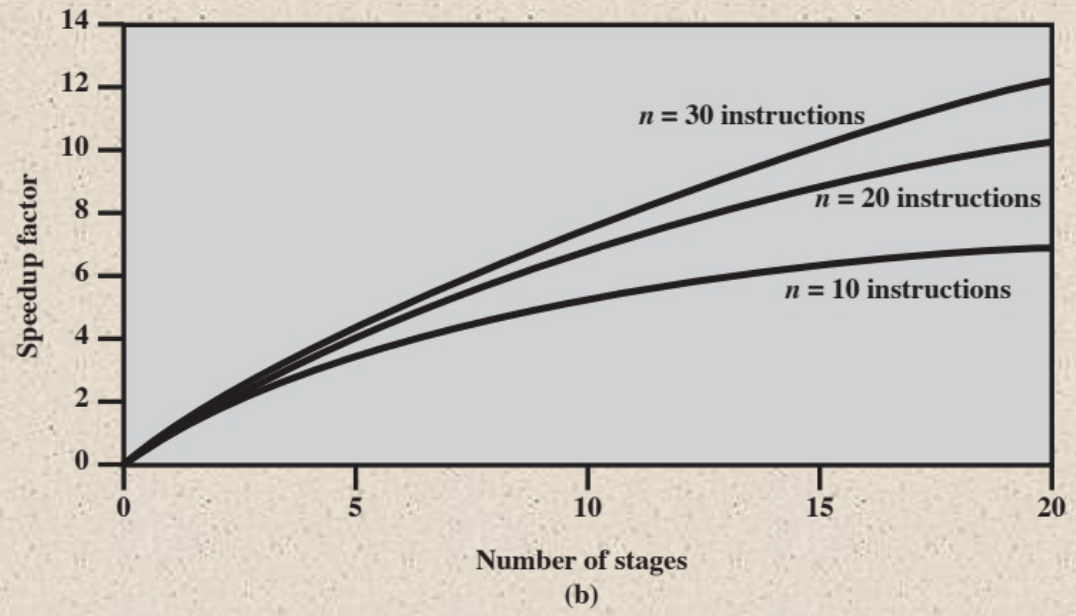


Figure 14.14 Speedup Factors with Instruction Pipelining

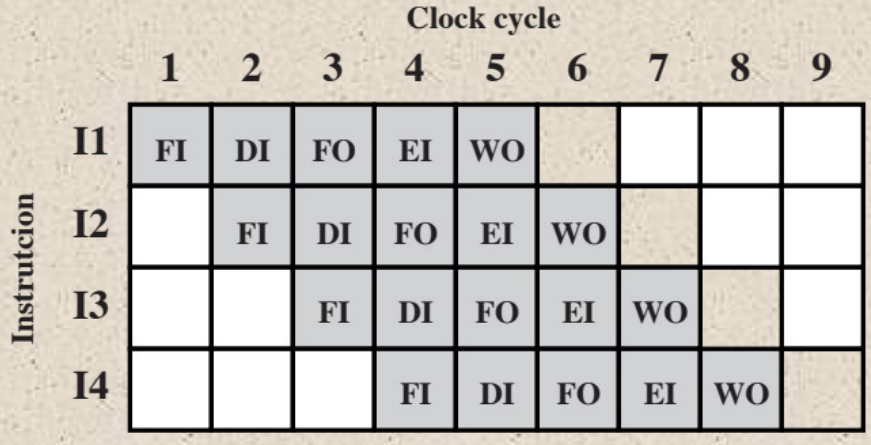
Pipeline Hazards

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

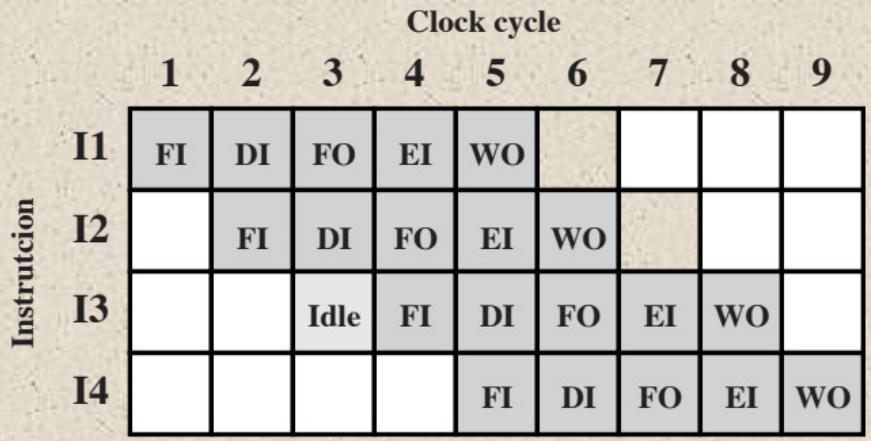
There are three types of hazards:

- Resource
- Data
- Control

Also referred to as a *pipeline bubble*



(a) Five-stage pipeline, ideal case



(b) I1 source operand in memory

Figure 14.15 Example of Resource Hazard

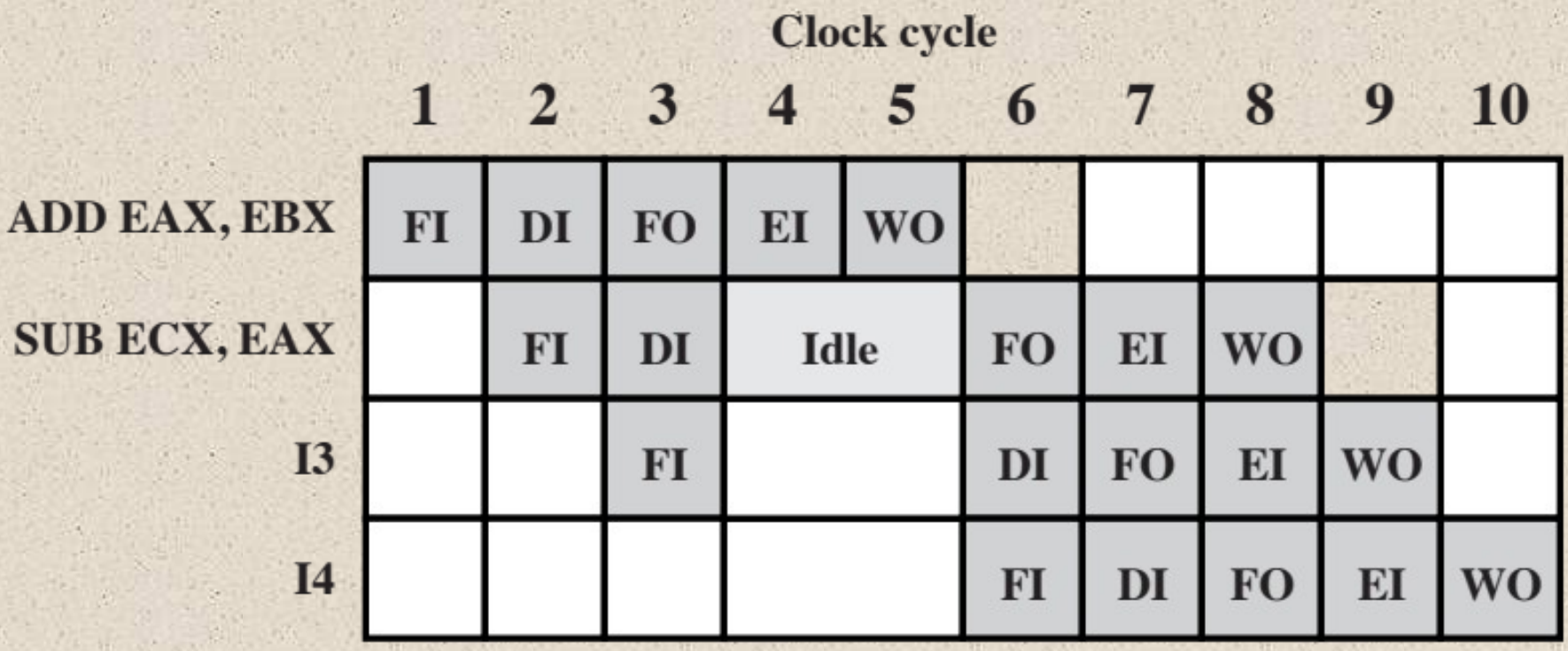


Figure 14.16 Example of Data Hazard

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

- 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

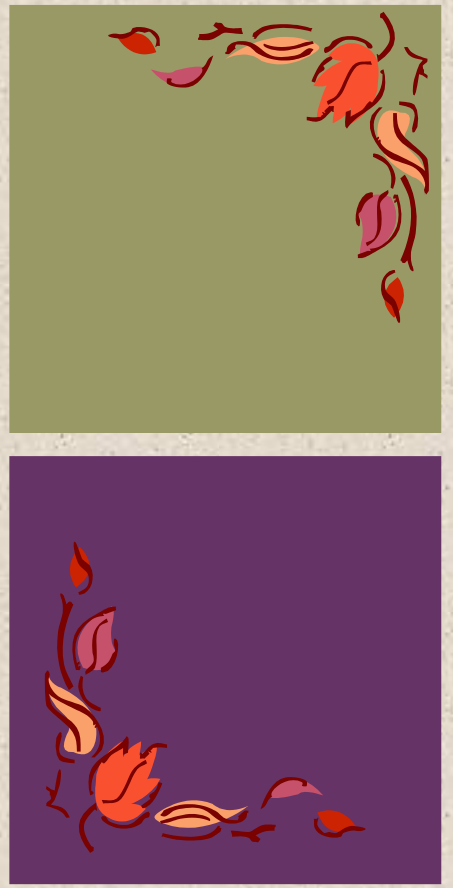
A brute-force approach is to replicate the initial portions of the pipeline and allow the pipeline to fetch both instructions, making use of two streams

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

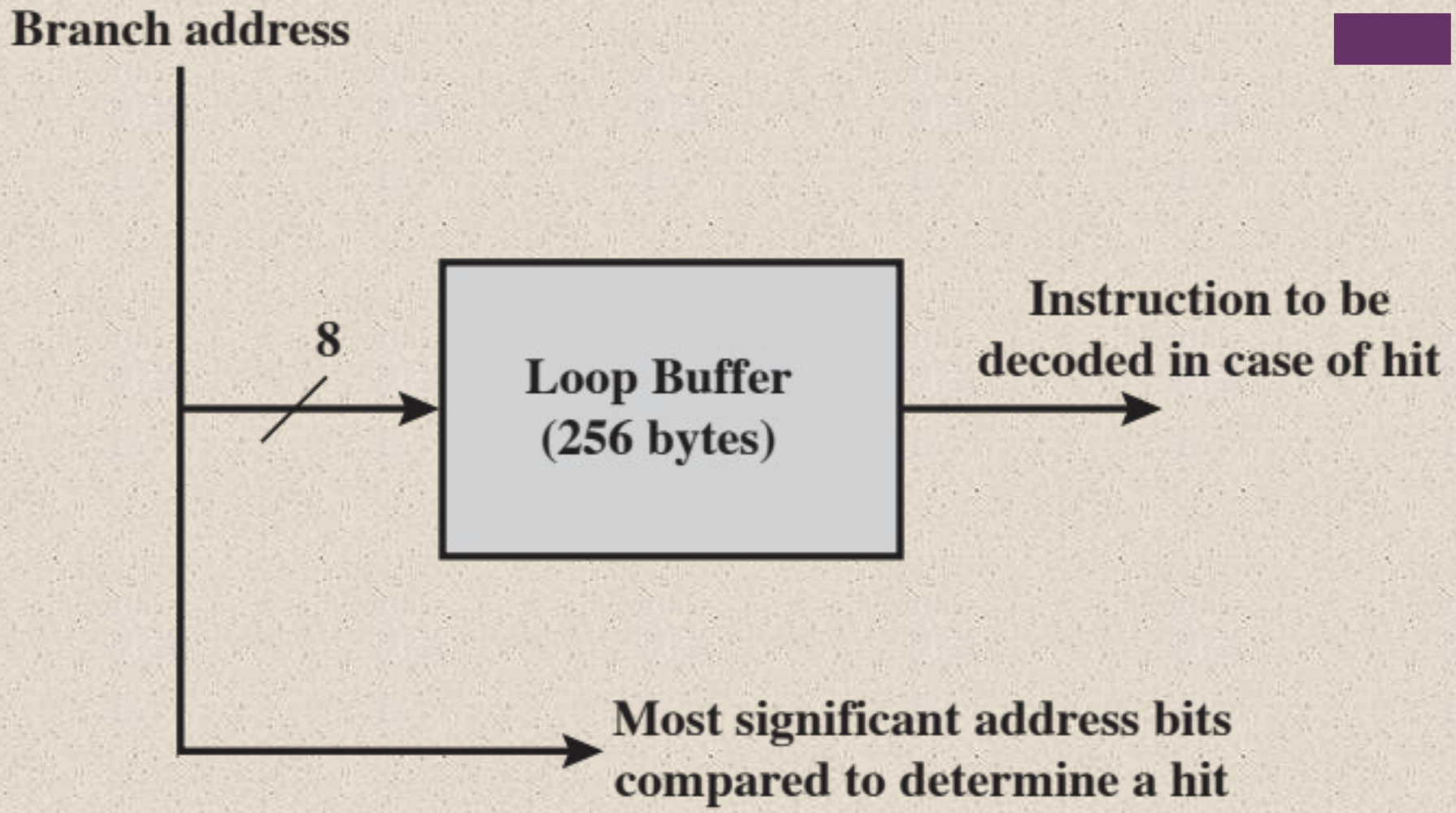


Figure 14.17 Loop Buffer

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

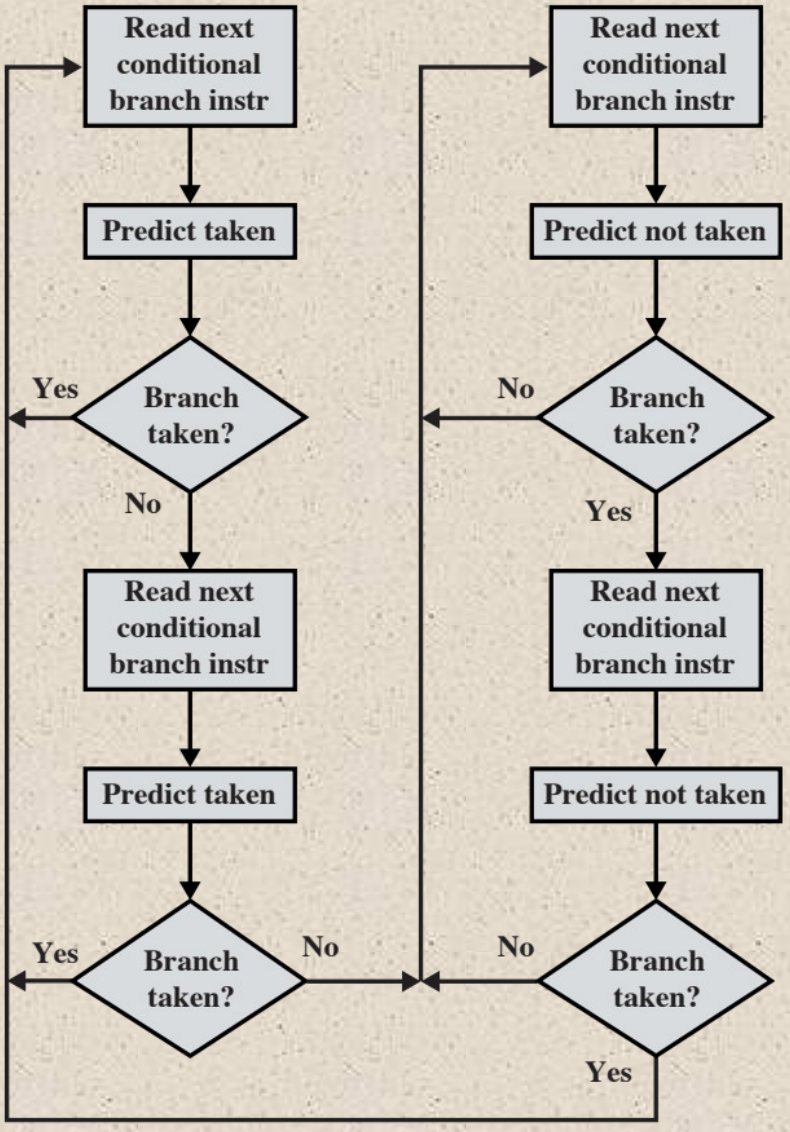


Figure 14.18 Branch Prediction Flow Chart

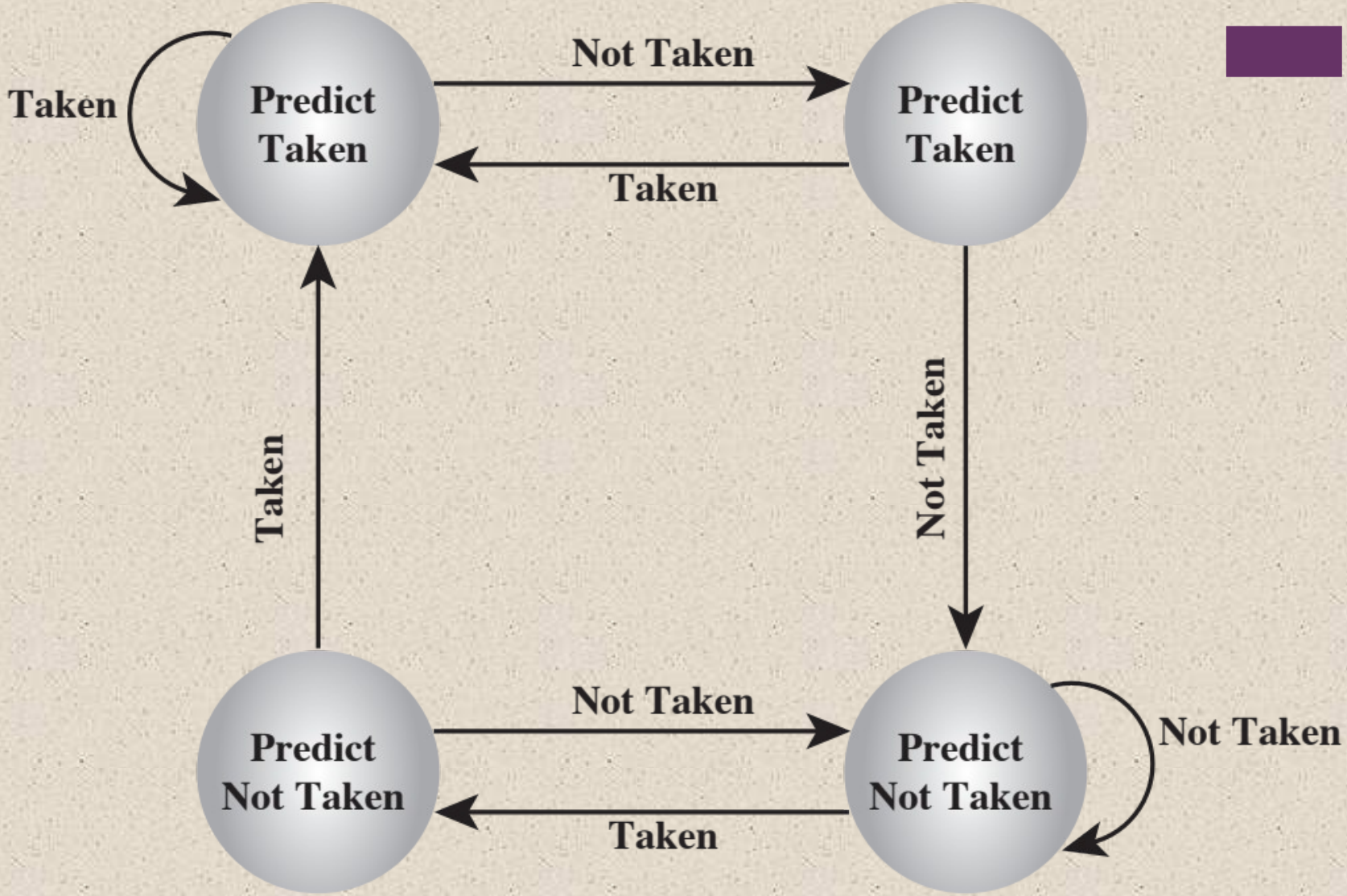


Figure 14.19 Branch Prediction State Diagram

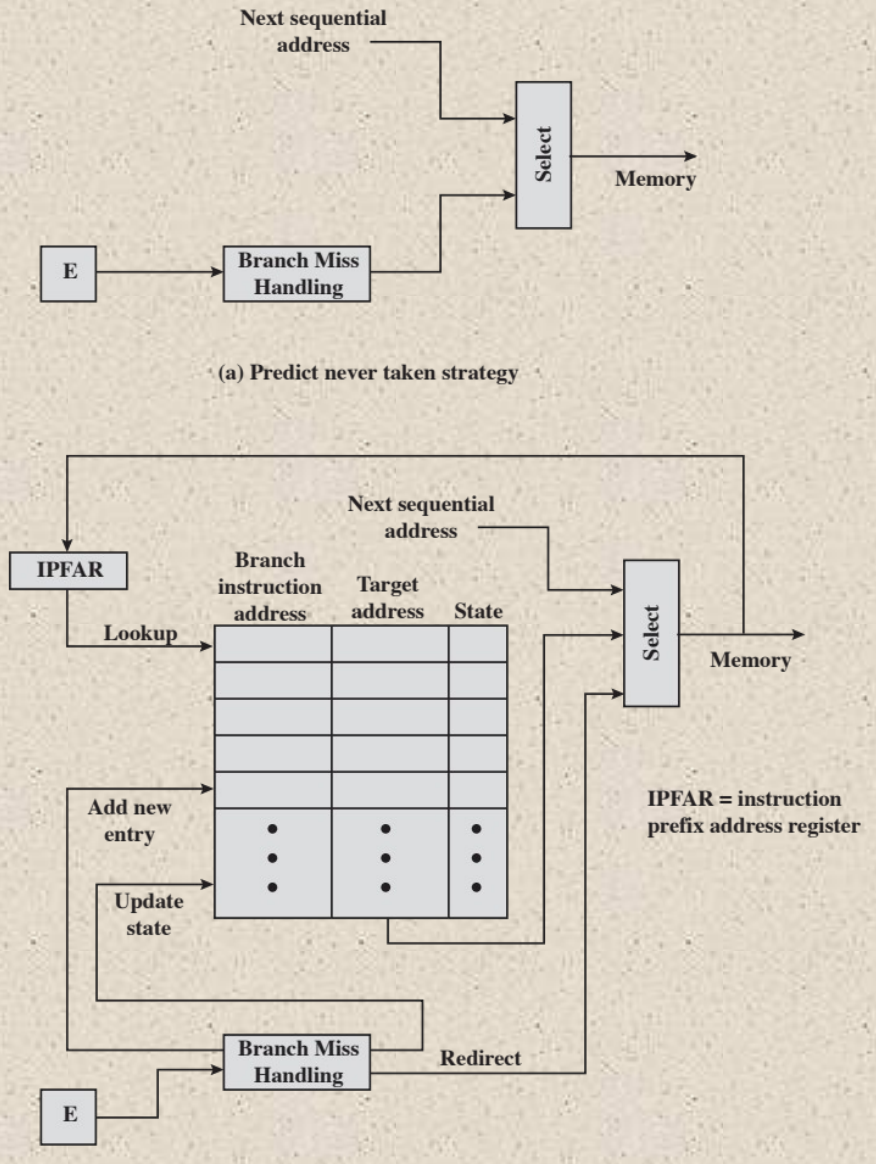


Figure 14.20 Dealing with Branches

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

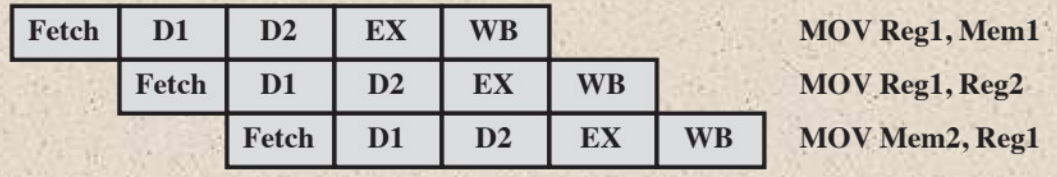
- Expands each opcode into control signals for the ALU
- Also controls the computation of the more complex addressing modes

Execute

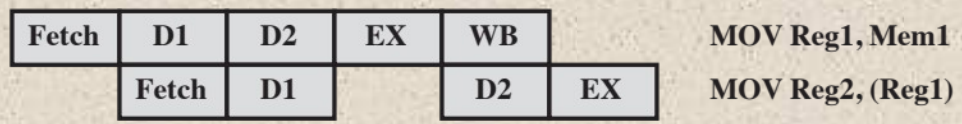
- Stage includes ALU operations, cache access, and register update

Write back

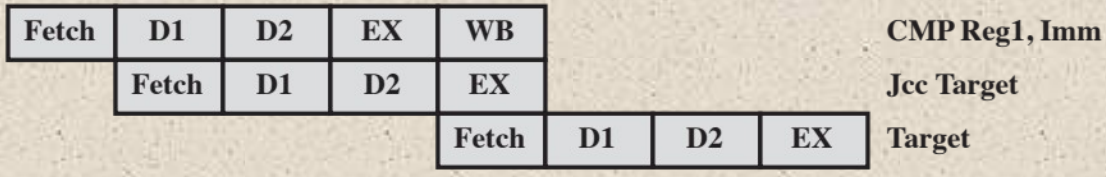
- Updates registers and status flags modified during the preceding execute stage



(a) No Data Load Delay in the Pipeline



(b) Pointer Load Delay



(c) Branch Instruction Timing

Figure 14.21 80486 Instruction Pipeline Examples

Table 14.2

x86 Processor Registers

(a) Integer Unit in 32-bit Mode

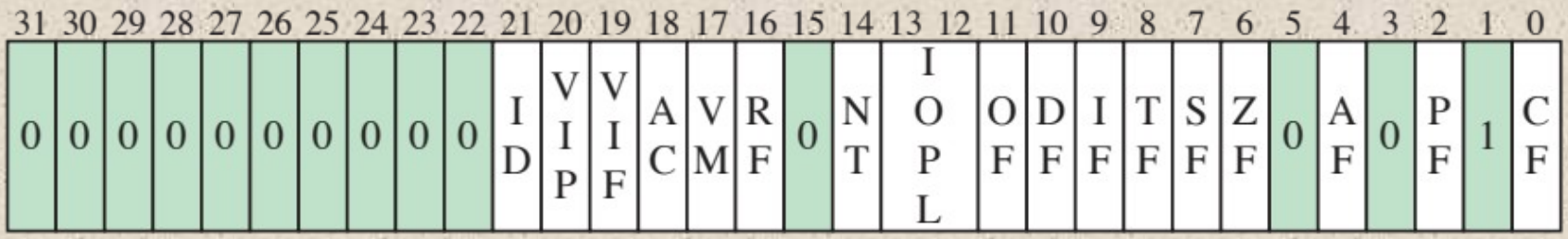
Type	Number	Length (bits)	Purpose
General	8	32	General-purpose user registers
Segment	6	16	Contain segment selectors
EFLAGS	1	32	Status and control bits
Instruction Pointer	1	32	Instruction pointer

(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

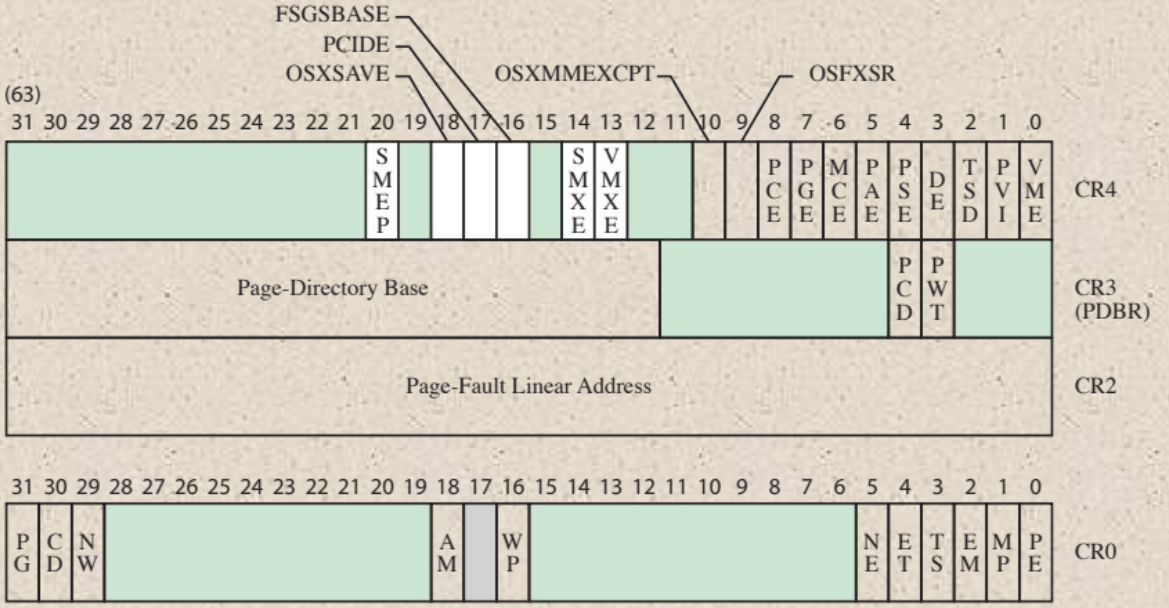


X ID = Identification flag
X VIP = Virtual interrupt pending
X VIF = Virtual interrupt flag
X AC = Alignment check
X VM = Virtual 8086 mode
X RF = Resume flag
X NT = Nested task flag
X IOPL = I/O privilege level
S OF = Overflow flag

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 14.22 x86 EFLAGS Register



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 14.23 x86 Control Registers

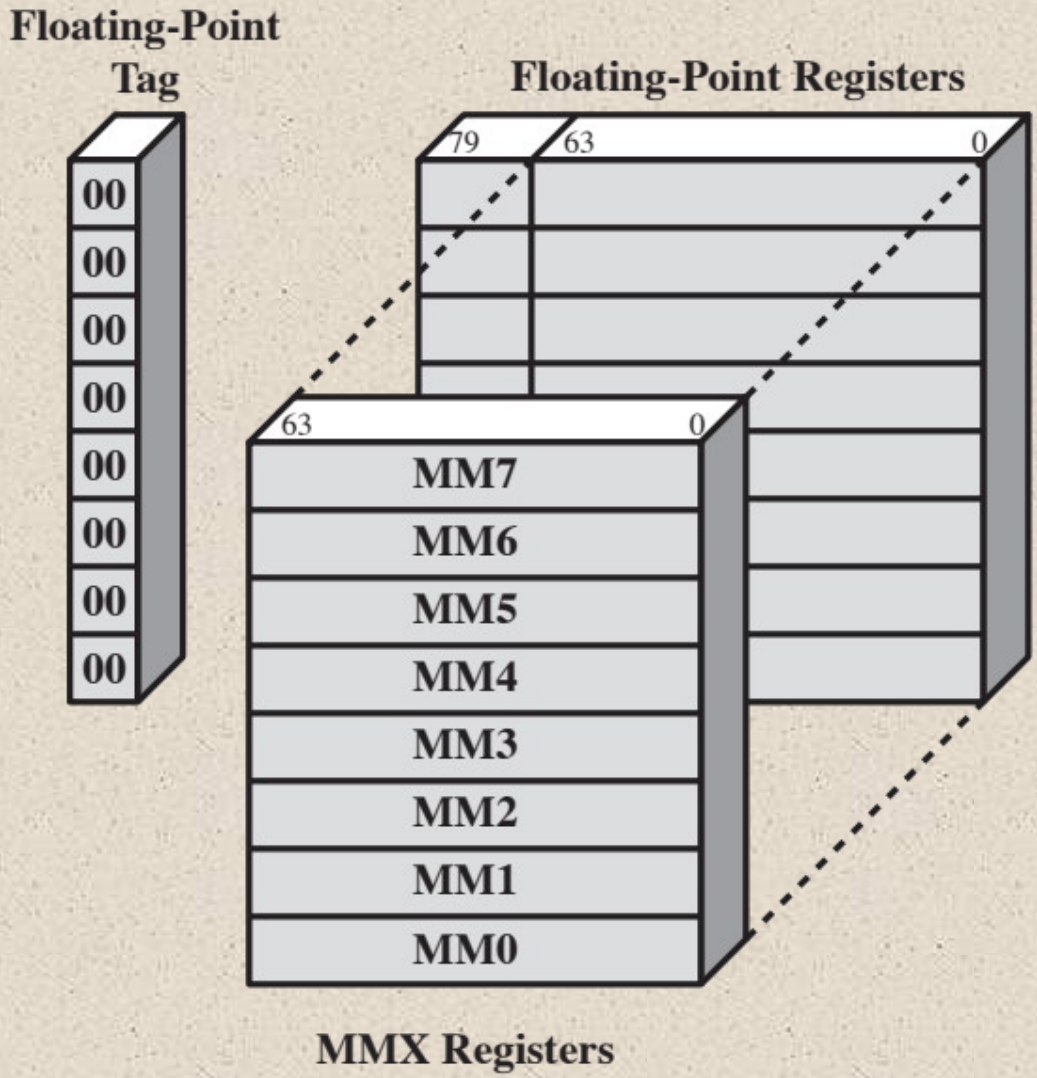


Figure 14.24 Mapping of MMX Registers to Floating-Point Registers

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

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

Table 14.3

x86 Exception and Interrupt Vector Table

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 instructions, thereby improving pipeline efficiency, because pipeline flushing is reduced

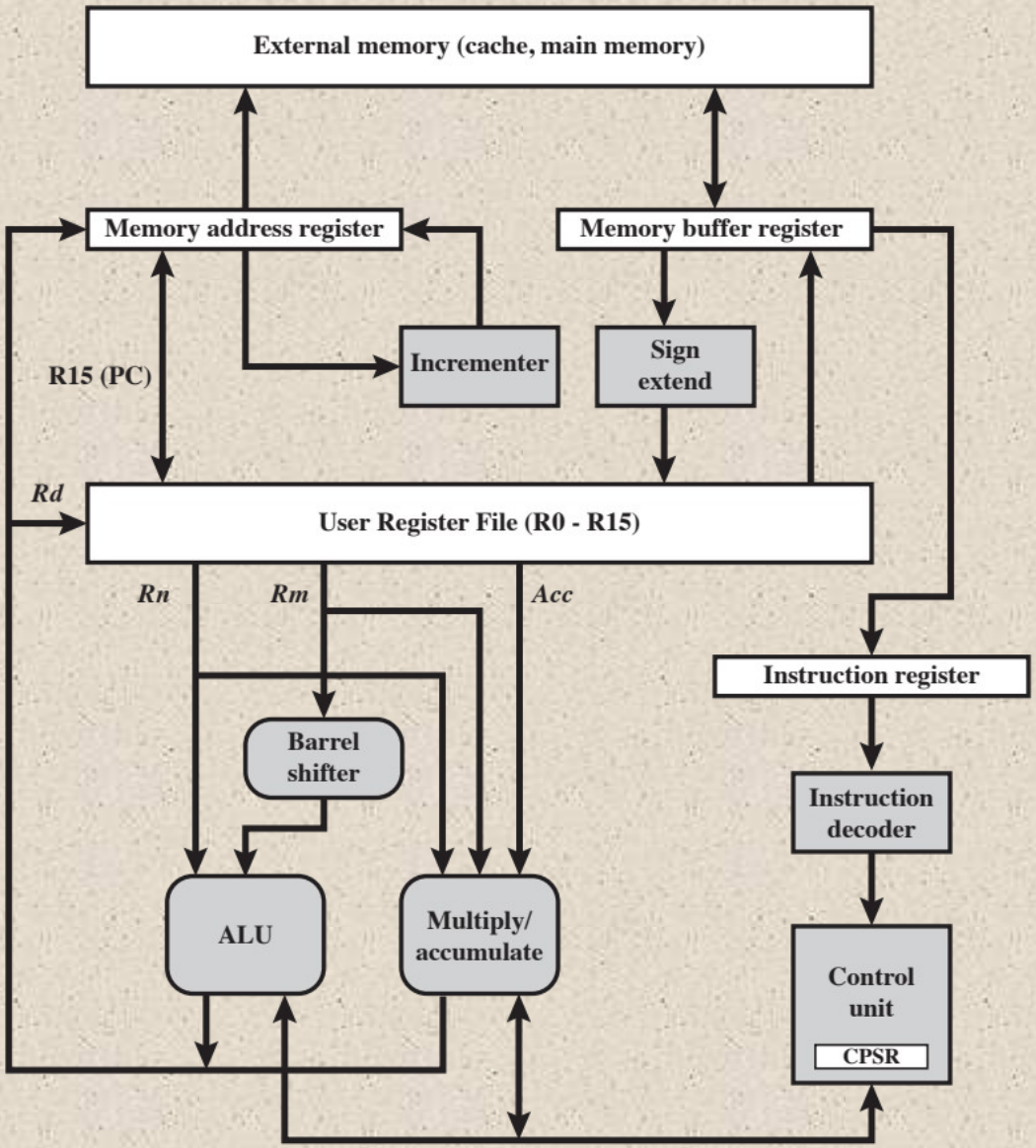
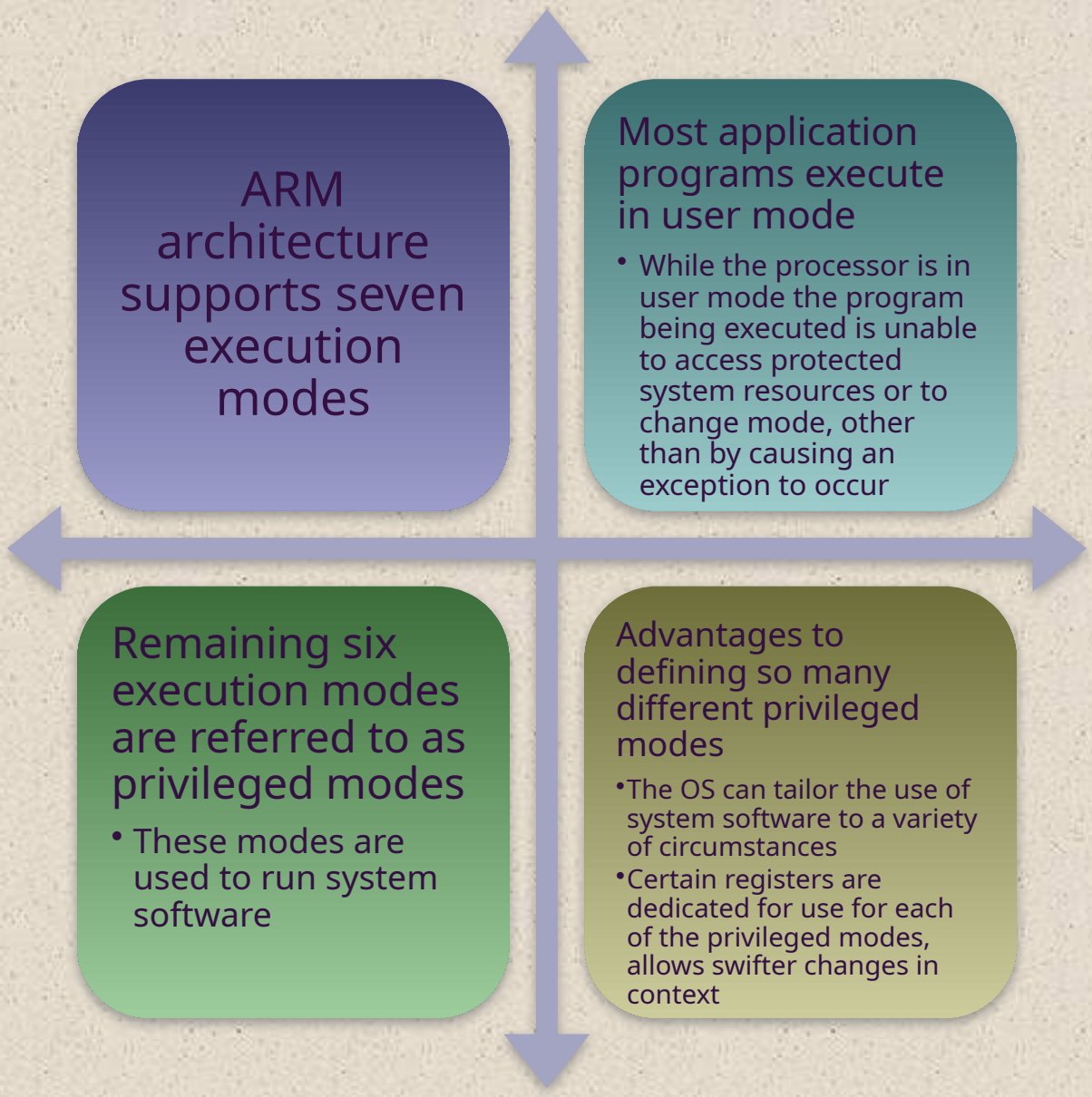


Figure 14.25 Simplified ARM Organization

Processor Modes



Exception Modes

Have full access to system resources and can change modes freely

Entered when specific exceptions occur

Exception modes:

- Supervisor mode
- Abort mode
- Undefined mode
- Fast interrupt mode
- Interrupt mode

System mode:

- Not entered by any exception and uses the same registers available in User mode
- Is used for running certain privileged operating system tasks
- May be interrupted by any of the five exception categories

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 14.26 ARM Register Organization

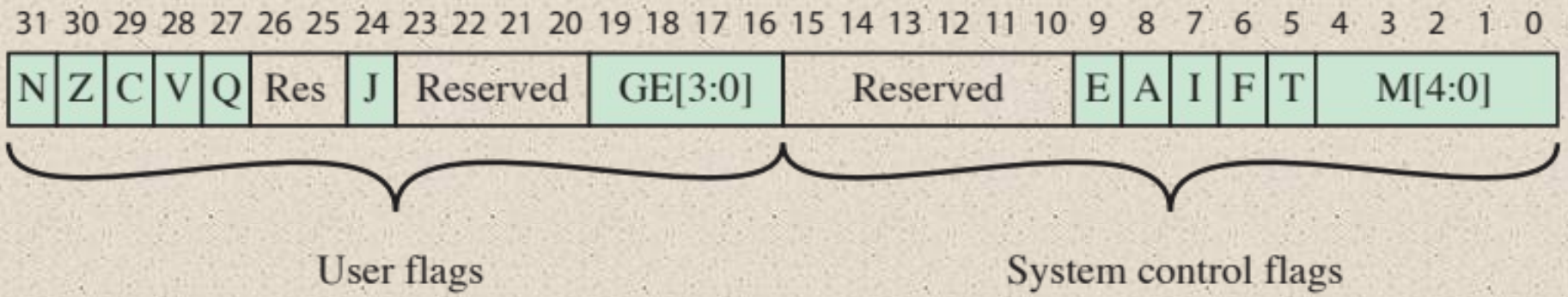


Figure 14.27 Format of ARM CPSR AND SPSR

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.

Table 14.4

ARM Interrupt Vector

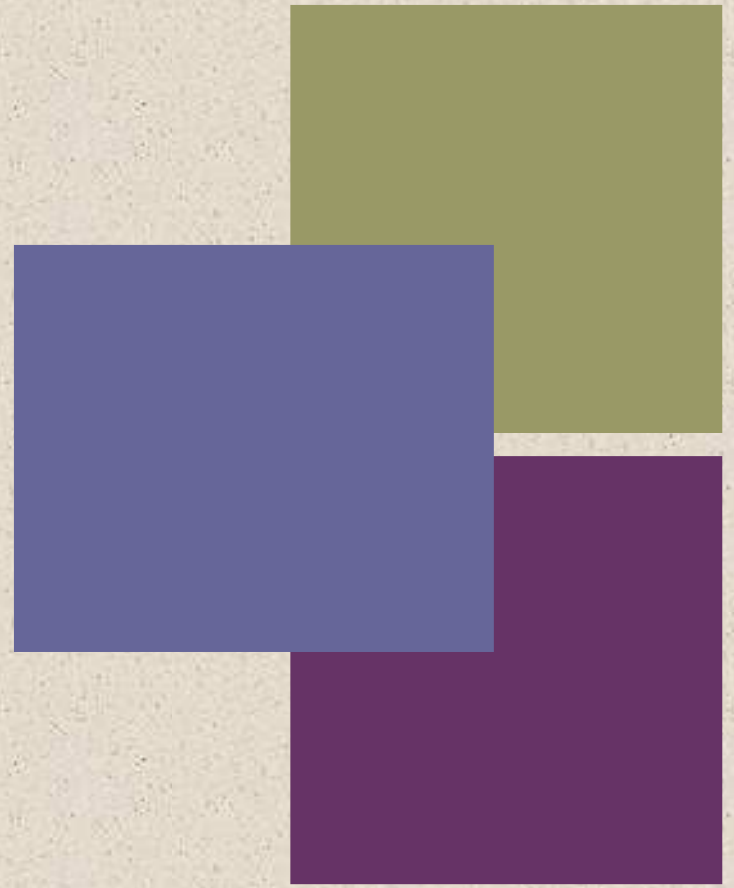
Summary

Chapter 14

Processor Structure and Function

- 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
- 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

William Stallings
Computer Organization and Architecture
10th Edition



Chapter 15

Reduced Instruction Set Computers (RISC)

Table 15.1
Characteristics of Some CISCs, RISCs, and Superscalar Processors

	Complex Instruction Set (CISC)Computer			Reduced Instruction Set (RISC) Computer	
Characteristic	IBM 370/168	VAX 11/780	Intel 80486	SPARC	MIPS R4000
Year developed	1973	1978	1989	1987	1991
Number of instructions	208	303	235	69	94
Instruction size (bytes)	2–6	2–57	1–11	4	4
Addressing modes	4	22	11	1	1
Number of general-purpose registers	16	16	8	40 - 520	32
Control memory size (kbits)	420	480	246	—	—
Cache size (kB)	64	64	8	32	128

(Table can be found on page 538 in the

Table 15.1
Characteristics of Some CISCs, RISCs, and Superscalar Processors

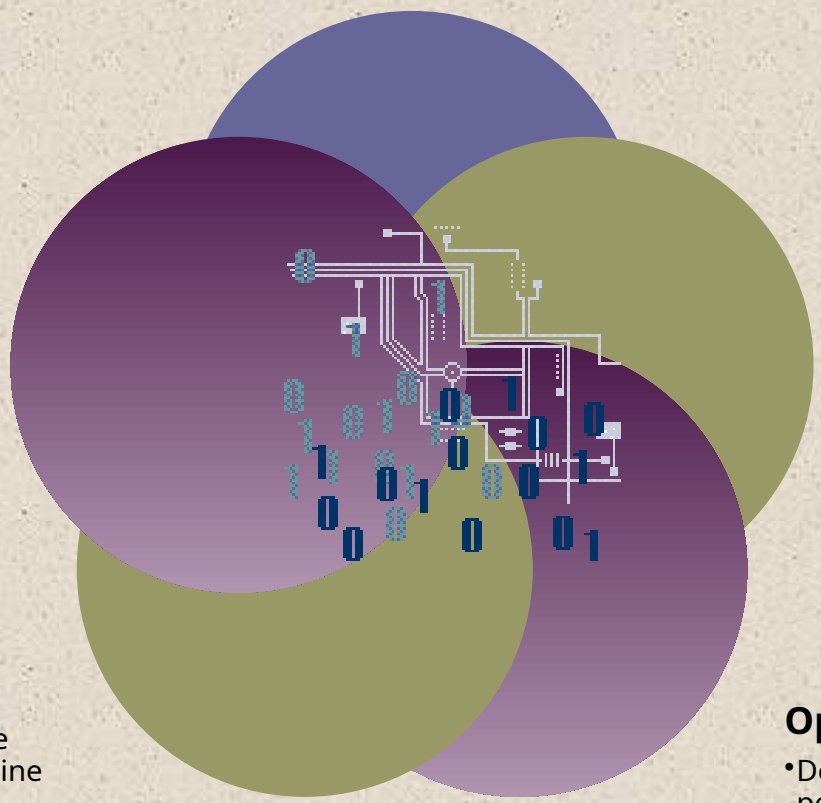
	Superscalar		
Characteristic	PowerPC	Ultra SPARC	MIPS R10000
Year developed	1993	1996	1996
Number of instructions	225		
Instruction size (bytes)	4	4	4
Addressing modes	2	1	1
Number of general-purpose registers	32	40 - 520	32
Control memory size (kbits)	—	—	—
Cache size (kB)	16-32	32	64

(Table can be found on page 538 in the

Instruction Execution Characteristics

High-level languages (HLLs)

- Allow the programmer to express algorithms more concisely
- Allow the compiler to take care of details that are not important in the programmer's expression of algorithms
- Often support naturally the use of structured programming and/or object-oriented design



Execution sequencing

- Determines the control and pipeline organization

Semantic gap

- The difference between the operations provided in HLLs and those provided in computer architecture

Operands used

- The types of operands and the frequency of their use determine the memory organization for storing them and the addressing modes for accessing them

Operations performed

- Determine the functions to be performed by the processor and its interaction with memory

Table 15.2
Weighted Relative Dynamic Frequency of HLL Operations [PATT82a]

	Dynamic Occurrence		Machine-Instruction Weighted		Memory-Reference Weighted	
	Pascal	C	Pascal	C	Pascal	C
ASSIGN	45%	38%	13%	13%	14%	15%
LOOP	5%	3%	42%	32%	33%	26%
CALL	15%	12%	31%	33%	44%	45%
IF	29%	43%	11%	21%	7%	13%
GOTO	—	3%	—	—	—	—
OTHER	6%	1%	3%	1%	2%	1%

(Table can be found on page 540 in the

Table 15.3
Dynamic Percentage of Operands

	Pascal	C	Average
Integer Constant	16%	23%	20%
Scalar Variable	58%	53%	55%
Array/Structure	26%	24%	25%

Table 15.4
Procedure Arguments and Local Scalar Variables

Percentage of Executed Procedure Calls With	Compiler, Interpreter, and Typesetter	Small Nonnumeric Programs
>3 arguments	0–7%	0–5%
>5 arguments	0–3%	0%
>8 words of arguments and local scalars	1–20%	0–6%
>12 words of arguments and local scalars	1–6%	0–3%

(Table can be found on page 541 in the

Implications

- HLLs can best be supported by optimizing performance of the most time-consuming features of typical HLL programs
- Three elements characterize RISC architectures:
  - Use a large number of registers or use a compiler to optimize register usage
  - Careful attention needs to be paid to the design of instruction pipelines
  - Instructions should have predictable costs and be consistent with a high-performance implementation

The Use of a Large Register File

Software Solution	Hardware Solution
Requires compiler to allocate registers	More registers
Allocates based on most used variables in a given time	Thus more variables will be in registers
Requires sophisticated program analysis	

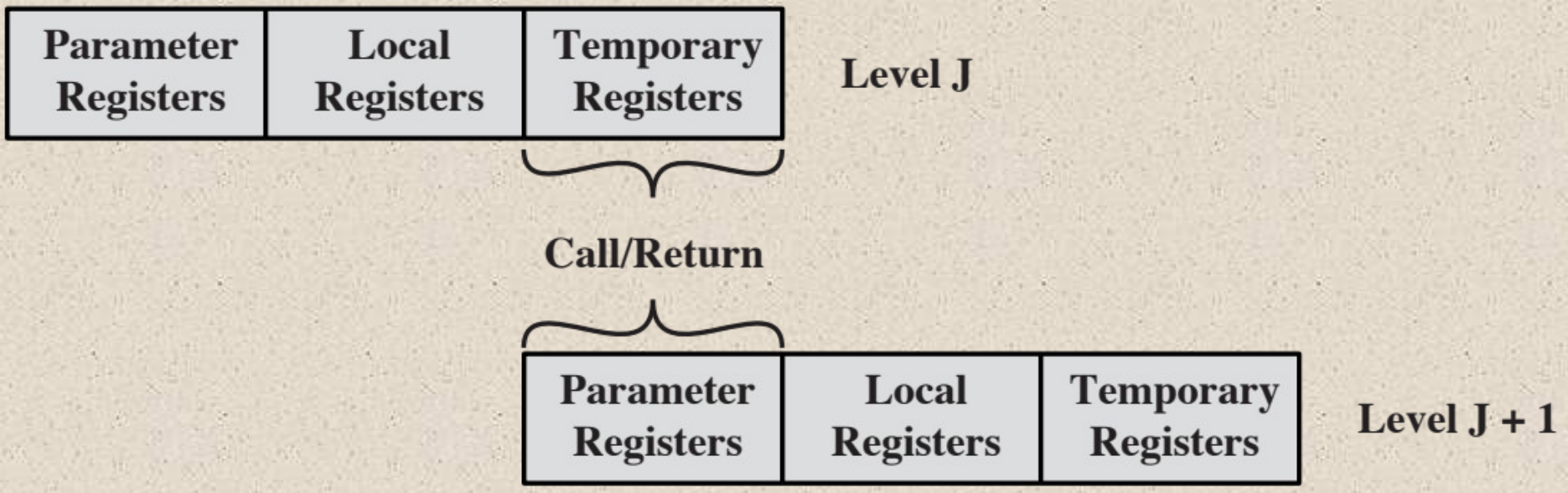


Figure 15.1 Overlapping Register Windows

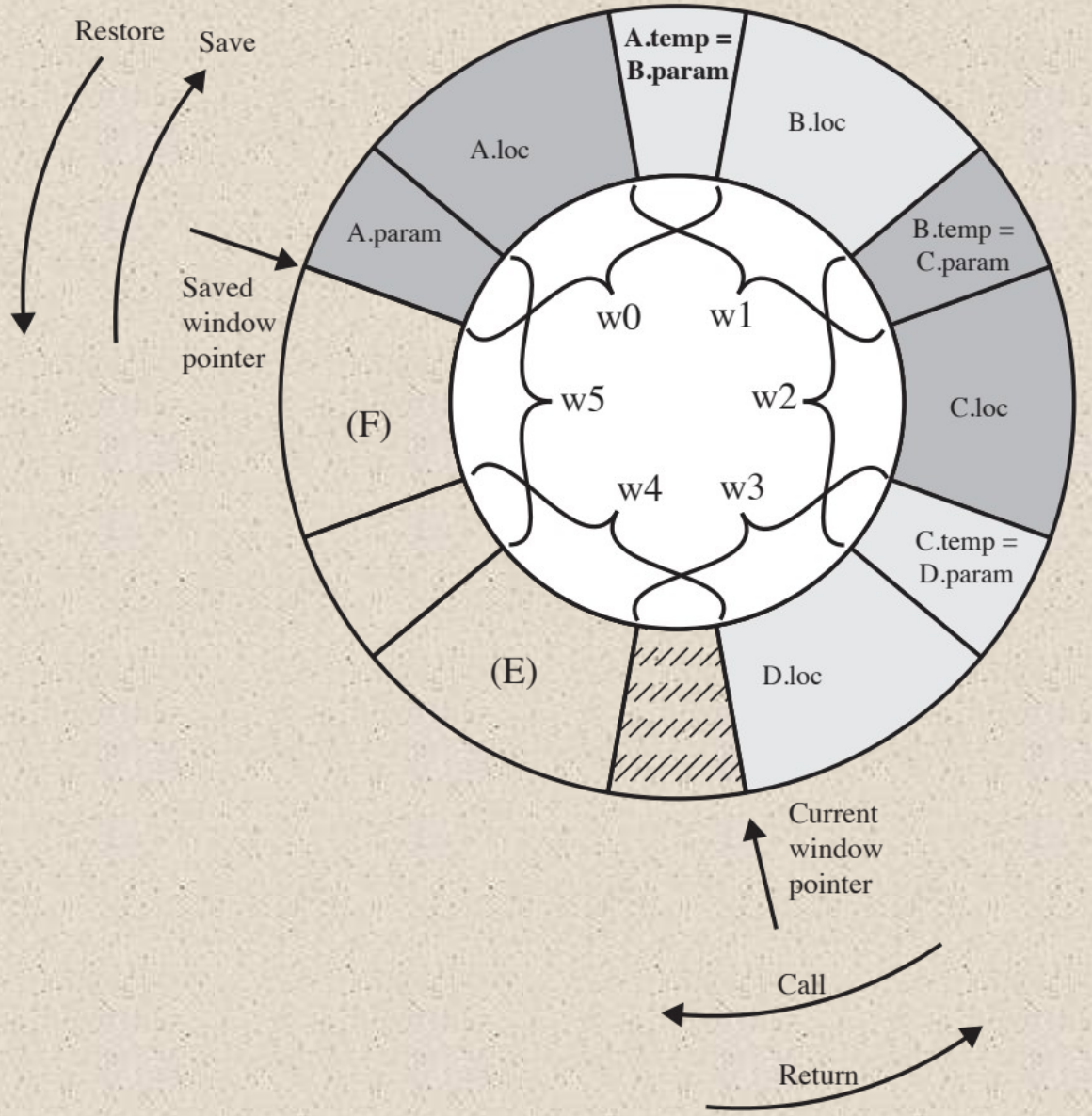


Figure 15.2 Circular-Buffer Organization of Overlapped Windows

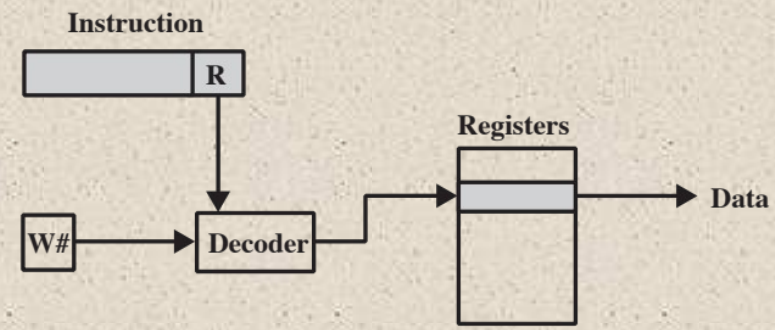
Global Variables

- Variables declared as global in an HLL can be assigned memory locations by the compiler and all machine instructions that reference these variables will use memory reference operands
  - However, for frequently accessed global variables this scheme is inefficient
- Alternative is to incorporate a set of global registers in the processor
  - These registers would be fixed in number and available to all procedures
  - A unified numbering scheme can be used to simplify the instruction format
- There is an increased hardware burden to accommodate the split in register addressing
- In addition, the linker must decide which global variables should be assigned to registers

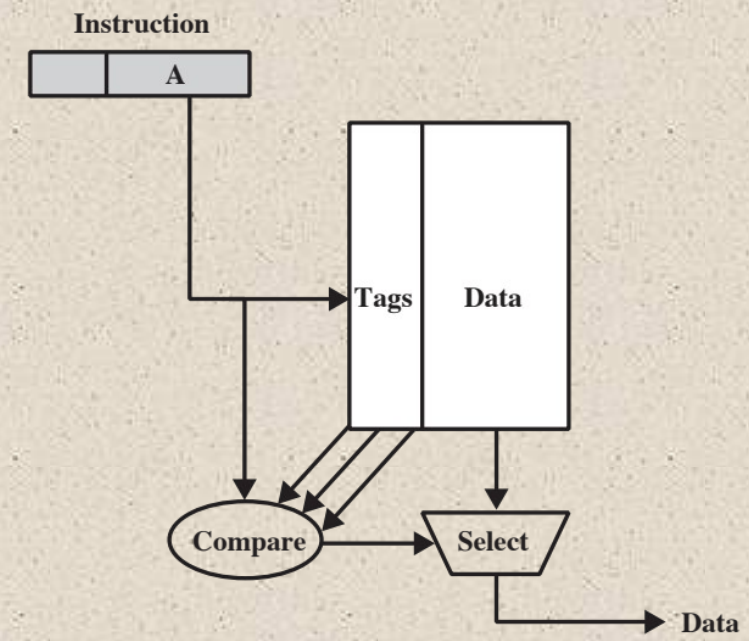
Table 15.5
Characteristics of Large-Register-File and Cache Organizations

Large Register File	Cache
All local scalars	Recently-used local scalars
Individual variables	Blocks of memory
Compiler-assigned global variables	Recently-used global variables
Save/Restore based on procedure nesting depth	Save/Restore based on cache replacement algorithm
Register addressing	Memory addressing
Multiple operands addressed and accessed in one cycle	One operand addressed and accessed per cycle

(Table can be found on page 546 in the

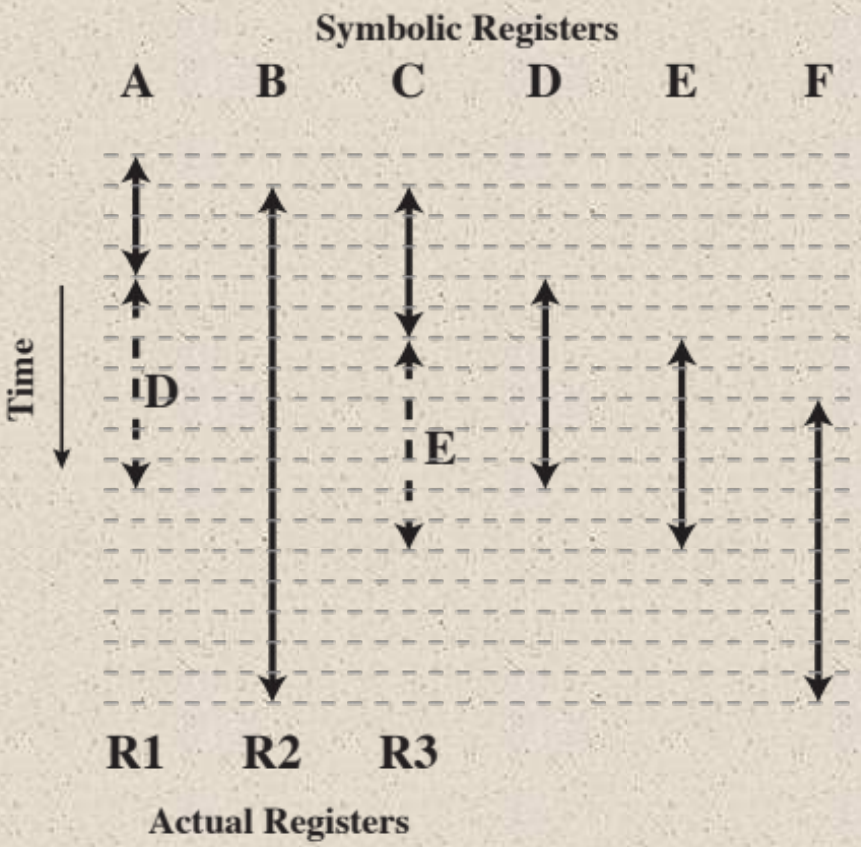


(a) Windows-based register file

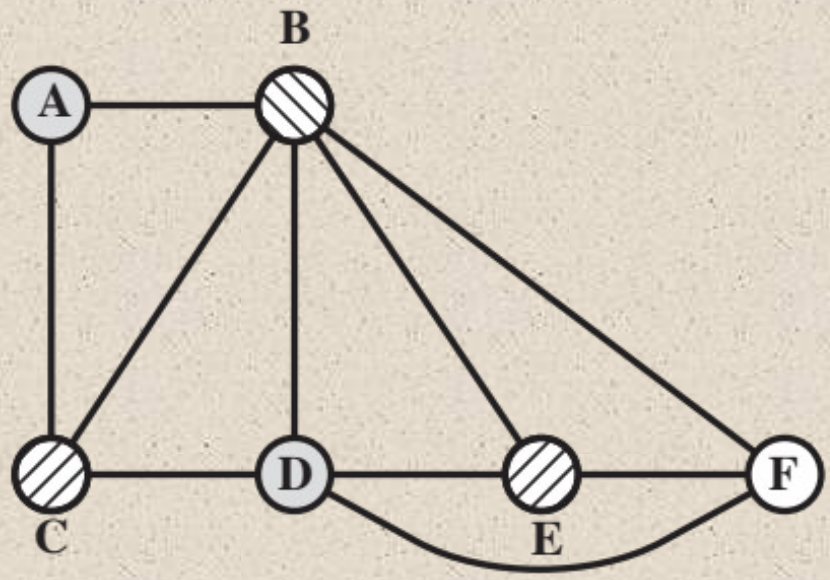


(b) Cache

Figure 15.3 Referencing a Scalar



(a) Time sequence of active use of registers



(b) Register interference graph

Figure 15.4 Graph Coloring Approach

Why CISC ?

(Complex Instruction Set Computer)

- There is a trend to richer instruction sets which include a larger and more complex number of instructions
- Two principal reasons for this trend:
  - A desire to simplify compilers
  - A desire to improve performance
- There are two advantages to smaller programs:
  - The program takes up less memory
  - Should improve performance
    - Fewer instructions means fewer instruction bytes to be fetched
    - In a paging environment smaller programs occupy fewer pages, reducing page faults
    - More instructions fit in cache(s)

Table 15.6
Code Size Relative to RISC I

	[PATT82a] 11 C Programs	[KATE83] 12 C Programs	[HEAT84] 5 C Programs
RISC I	1.0	1.0	1.0
VAX-11/780	0.8	0.67	
M68000	0.9		0.9
Z8002	1.2		1.12
PDP-11/70	0.9	0.71	

(Table can be found on page 550 in the

Characteristics of Reduced Instruction Set Architectures

One machine instruction per machine cycle	• *Machine cycle* --- the time it takes to fetch two operands from registers, perform an ALU operation, and store the result in a register
Register-to-register operations	• Only simple LOAD and STORE operations accessing memory • This simplifies the instruction set and therefore the control unit
Simple addressing modes	• Simplifies the instruction set and the control unit
Simple instruction formats	• Generally only one or a few formats are used • Instruction length is fixed and aligned on word boundaries • Opcode decoding and register operand accessing can occur simultaneously

8	16	16	16
Add	B	C	A

Memory to memory

I = 56, D= 96, M = 152

8	4	16		
Load	RB	B		
Load	RC	B		
Add	RA	RB	RC	
Store	RA	A		

Register to memory

I = 104, D = 96, M = 200

(a) A ← B + C

8	16	16	16
Add	B	C	A
Add	A	C	B
Sub	B	D	D

Memory to memory

I = 168, D= 288, M = 456

8	4	4	4
Add	RA	RB	RC
Add	RB	RA	RC
Sub	RD	RD	RB

Register to register

I = 60, D = 0, M = 60

(b) A ← B + C; B ← A + C; D ← D – B

I = number of bytes occupied by executed instructions
D = number of bytes occupied by data
M = total memory traffic = I + D

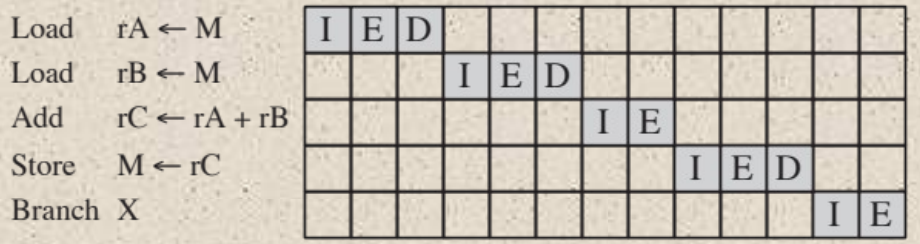
Figure 15.5 Two Comparisons of Register-to-Register and Memory-to-Memory Approaches

Table 15.7
Characteristics of Some Processors

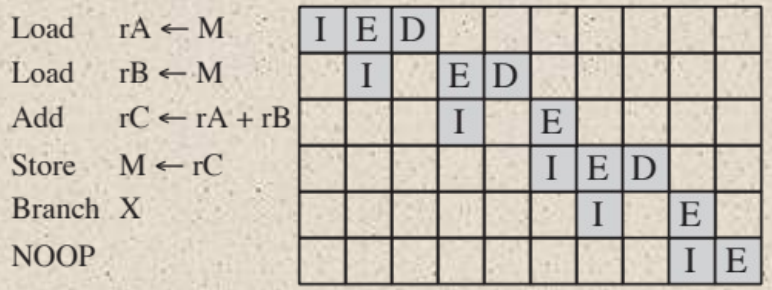
Processor	Number of instruc-tion sizes	Max instruc-tion size in bytes	Number of addressing modes	Indirect addressing	Load/store combined with arithmetic	Max number of memory operands	Unaligned addressing allowed	Max Number of MMU uses	Number of bits for integer register specifier	Number of bits for FP register specifier
AMD29000	1	4	1	no	no	1	no	1	8	3[a]
MIPS R2000	1	4	1	no	no	1	no	1	5	4
SPARC	1	4	2	no	no	1	no	1	5	4
MC88000	1	4	3	no	no	1	no	1	5	4
HP PA	1	4	10[a]	no	no	1	no	1	5	4
IBM RT/PC	2[a]	4	1	no	no	1	no	1	4[a]	3[a]
IBM RS/6000	1	4	4	no	no	1	yes	1	5	5
Intel i860	1	4	4	no	no	1	no	1	5	4
IBM 3090	4	8	2[b]	no[b]	yes	2	yes	4	4	2
Intel 80486	12	12	15	no[b]	yes	2	yes	4	3	3
NSC 32016	21	21	23	yes	yes	2	yes	4	3	3
MC68040	11	22	44	yes	yes	2	yes	8	4	3
VAX	56	56	22	yes	yes	6	yes	24	4	0
Clipper	4[a]	8[a]	9[a]	no	no	1	0	2	4[a]	3[a]
Intel 80960	2[a]	8[a]	9[a]	no	no	1	yes[a]	—	5	3[a]

a RISC that does not conform to this characteristic.
b CISC that does not conform to this characteristic.

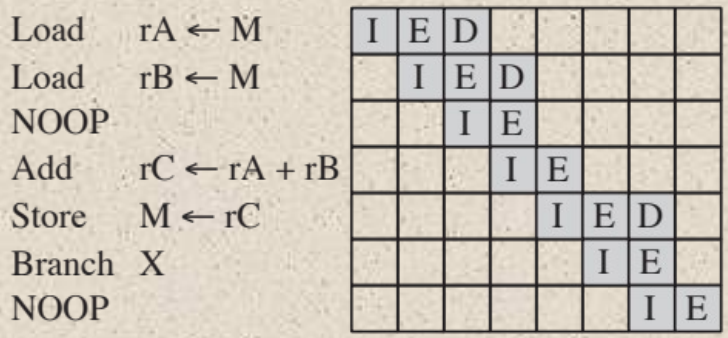
(Table can be found on page 554 in the textbook.)



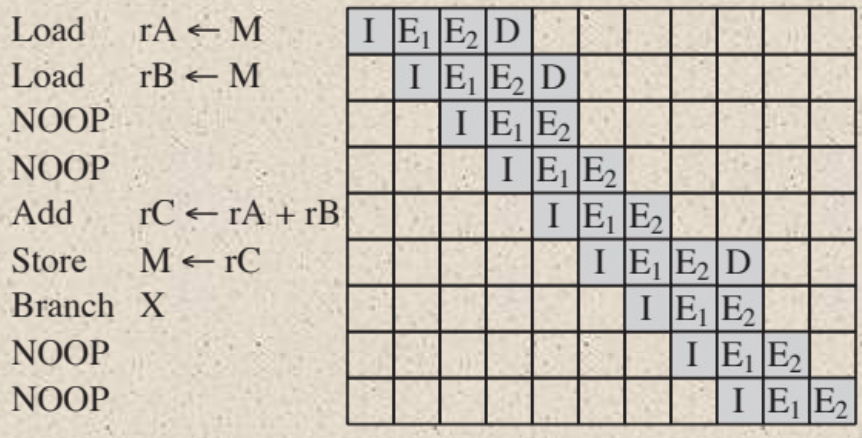
(a) Sequential execution



(b) Two-stage pipelined timing



(c) Three-stage pipelined timing



(d) Four-stage pipelined timing

Figure 15.6 The Effects of Pipelining

Optimization of Pipelining

- Delayed branch
  - Does not take effect until after execution of following instruction
  - This following instruction is the delay slot
- Delayed Load
  - Register to be target is locked by processor
  - Continue execution of instruction stream until register required
  - Idle until load is complete
  - Re-arranging instructions can allow useful work while loading
- Loop Unrolling
  - Replicate body of loop a number of times
  - Iterate loop fewer times
  - Reduces loop overhead
  - Increases instruction parallelism
  - Improved register, data cache, or TLB locality

Table 15.8
Normal And Delayed Branch

Address	Normal Branch		Delayed Branch		Optimized Delayed Branch	
100	LOAD	X, rA	LOAD	X, rA	LOAD	X, rA
101	ADD	1, rA	ADD	1, rA	JUMP	105
102	JUMP	105	JUMP	106	ADD	1, rA
103	ADD	rA, rB	NOOP		ADD	rA, rB
104	SUB	rC, rB	ADD	rA, rB	SUB	rC, rB
105	STORE	rA, Z	SUB	rC, rB	STORE	rA, Z
106			STORE	rA, Z		

(Table can be found on page 557 in the

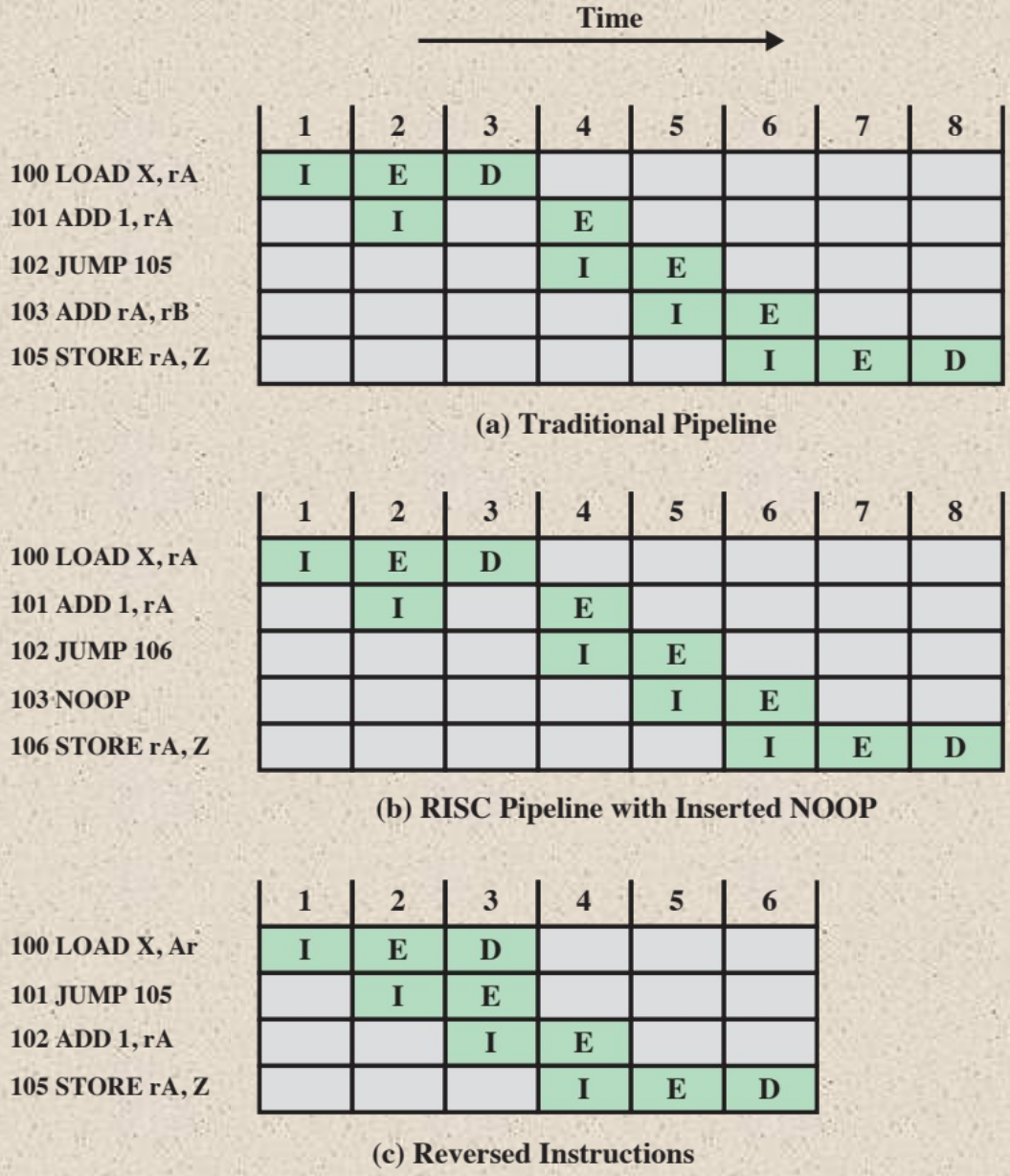


Figure 15.7 Use of the Delayed Branch

```
do i=2, n-1
     a[i] = a[i] + a[i-1] * a[i+1]
end do
```

(a) original loop

```
do i=2, n-2, 2
     a[i] = a[i] + a[i-1] * a[i+i]
     a[i+1] = a[i+1] + a[i] * a[i+2]
end do

if (mod(n-2,2) = i) then
     a[n-1] = a[n-1] + a[n-2] * a[n]
end if
```

(b) loop unrolled twice

Figure 15.8 Loop unrolling

MIPS R4000

One of the first commercially available RISC chip sets was developed by MIPS Technology Inc.

Inspired by an experimental system developed at Stanford

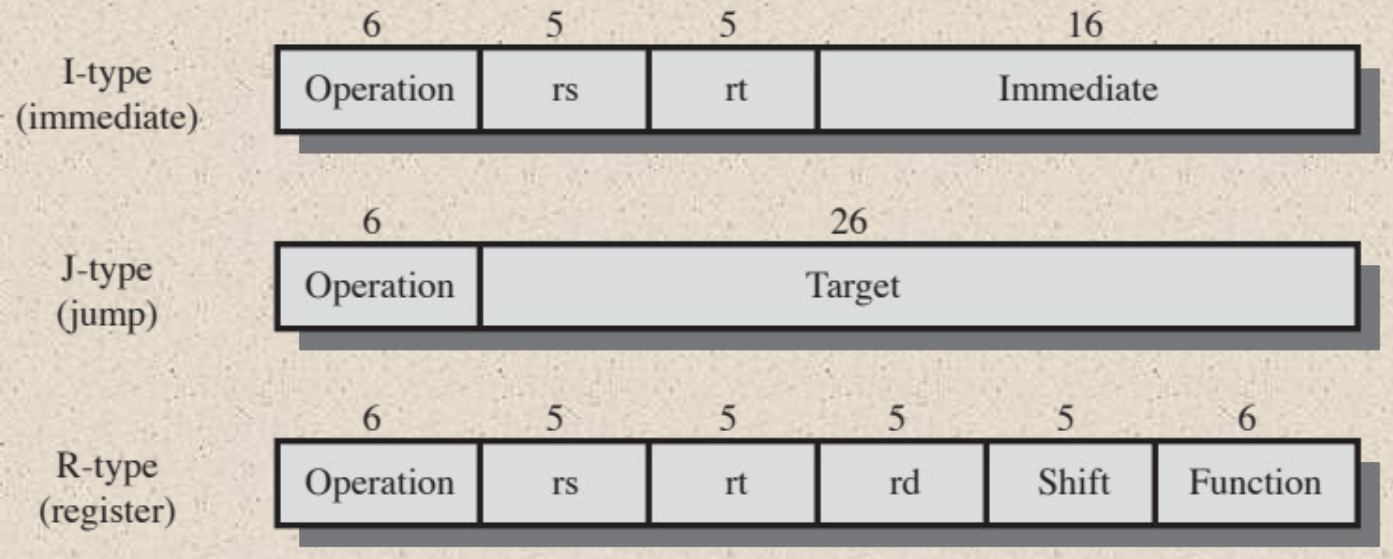
Has substantially the same architecture and instruction set of the earlier MIPS designs (R2000 and R3000)

Uses 64 bits for all internal and external data paths and for addresses, registers, and the ALU

Is partitioned into two sections, one containing the CPU and the other containing a coprocessor for memory management

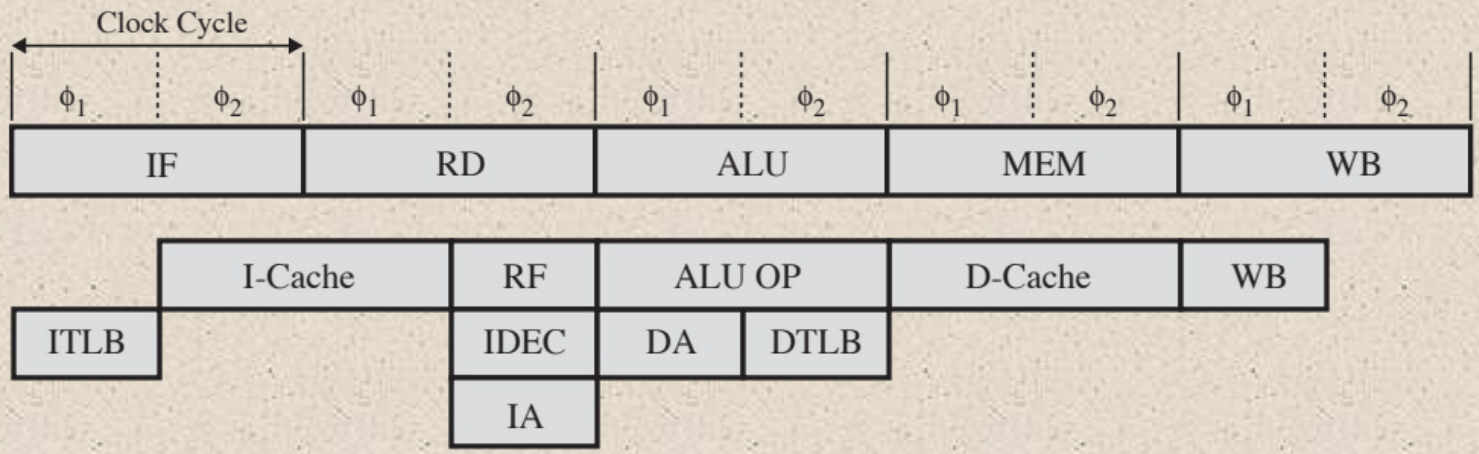
Supports thirty-two 64-bit registers

Provides for up to 128 Kbytes of high-speed cache, half each for instructions and data

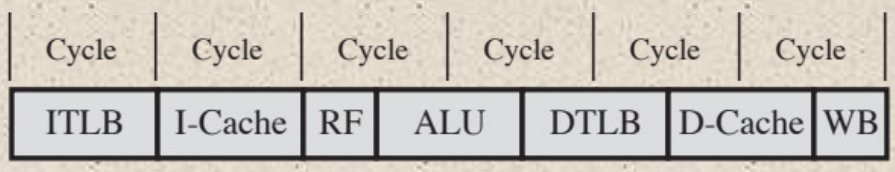


Operation	Operation code
rs	Source register specifier
rt	Source/destination register specifier
Immediate	Immediate, branch, or address displacement
Target	Jump target address
rd	Destination register specifier
Shift	Shift amount
Function	ALU/shift function specifier

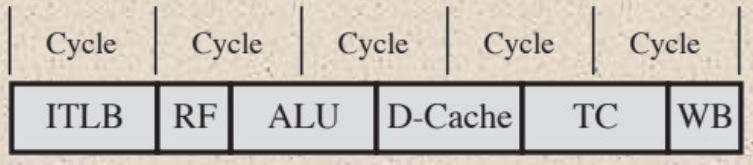
Figure 15.9 MIPS Instruction Formats



(a) Detailed R3000 pipeline



(b) Modified R3000 pipeline with reduced latencies



(c) Optimized R3000 pipeline with parallel TLB and cache accesses

IF	=	Instruction fetch
RD	=	Read
MEM	=	Memory access
WB	=	Write back to register file
I-Cache	=	Instruction cache access
RF	=	Fetch operand from register
D-Cache	=	Data cache access
ITLB	=	Instruction address translation
IDEC	=	Instruction decode
IA	=	Compute instruction address
DA	=	Calculate data virtual address
DTLB	=	Data address translation
TC	=	Data cache tag check

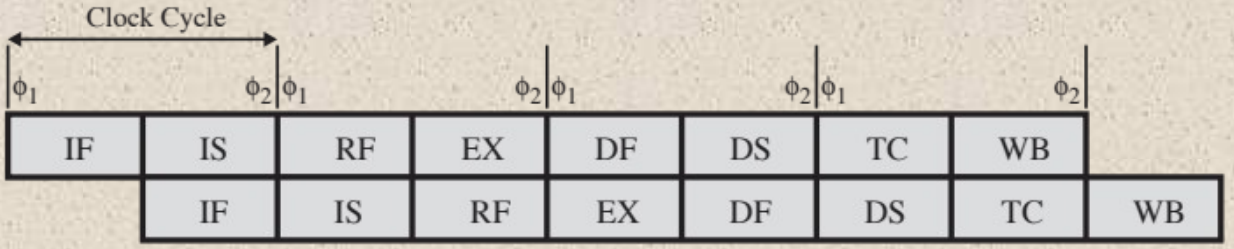
Figure 15.10 Enhancing the R3000 Pipeline

Table 15.9
R3000 Pipeline Stages

Pipeline Stage	Phase	Function
IF	$\phi 1$	Using the TLB, translate an instruction virtual address to a physical address (after a branching decision).
IF	$\phi 2$	Send the physical address to the instruction address.
RD	$\phi 1$	Return instruction from instruction cache. Compare tags and validity of fetched instruction.
RD	$\phi 2$	Decode instruction. Read register file. If branch, calculate branch target address.
ALU	$\phi 1 + \phi 2$	If register-to-register operation, the arithmetic or logical operation is performed.
ALU	$\phi 1$	If a branch, decide whether the branch is to be taken or not. If a memory reference (load or store), calculate data virtual address.
ALU	$\phi 2$	If a memory reference, translate data virtual address to physical using TLB.
MEM	$\phi 1$	If a memory reference, send physical address to data cache.
MEM	$\phi 2$	If a memory reference, return data from data cache, and check tags.
WB	$\phi 1$	Write to register file.

(Table can be found on page 563 in the

(a) Superpipelined implmentation of the optimized R3000 pipeline



(b) R4000 pipeline

IF = Instruction fetch first half
IS = Instruction fetch second half
RF = Fetch operands from register
EX = Instruction execute
IC = Instruction cache
DC = Data cache
DF = Data cache first half
DS = Data cache second half
TC = Tag check
WB = Write back to register file

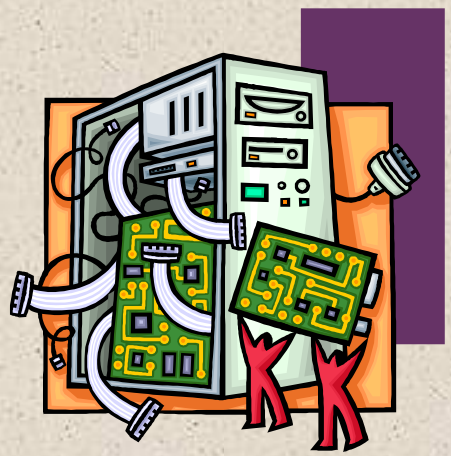
Figure 15.11 Theoretical R3000 and Actual R4000 Superpipelines

R4000 Pipeline Stages

- Instruction fetch first half
  - Virtual address is presented to the instruction cache and the translation lookaside buffer
- Instruction fetch second half
  - Instruction cache outputs the instruction and the TLB generates the physical address
- Register file
  - One of three activities can occur:
    - Instruction is decoded and check made for interlock conditions
    - Instruction cache tag check is made
    - Operands are fetched from the register file
- Tag check
  - Cache tag checks are performed for loads and stores
- Instruction execute
  - One of three activities can occur:
    - If register-to-register operation the ALU performs the operation
    - If a load or store the data virtual address is calculated
    - If branch the branch target virtual address is calculated and branch operations checked
- Data cache first
  - Virtual address is presented to the data cache and TLB
- Data cache second
  - The TLB generates the physical address and the data cache outputs the data
- Write back
  - Instruction result is written back to register file

SPARC

Scalable Processor Architecture



- Architecture defined by Sun Microsystems
- Sun licenses the architecture to other vendors to produce SPARC-compatible machines
- Inspired by the Berkeley RISC 1 machine, and its instruction set and register organization is based closely on the Berkeley RISC model

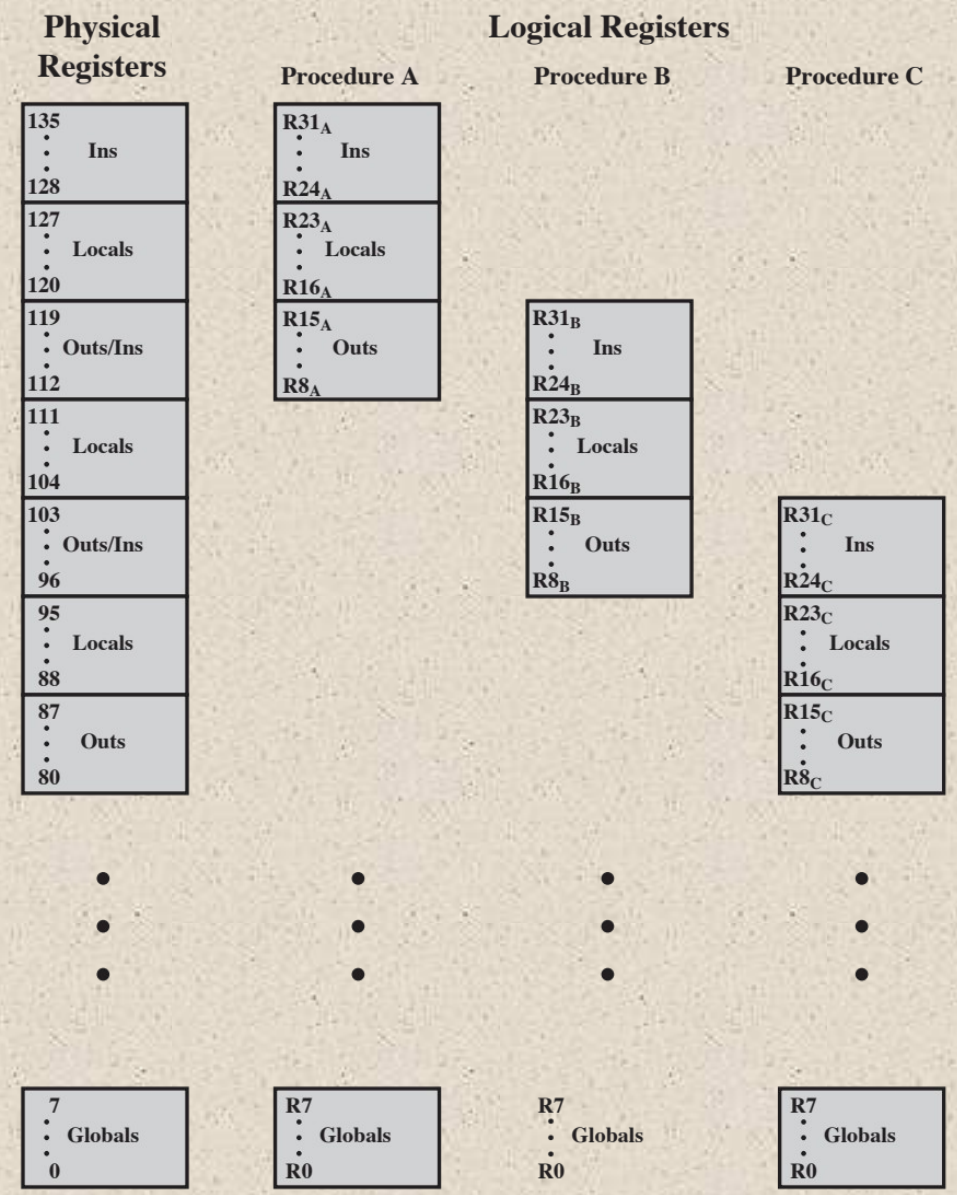


Figure 15.12 SPARC Register Window Layout with Three Procedures

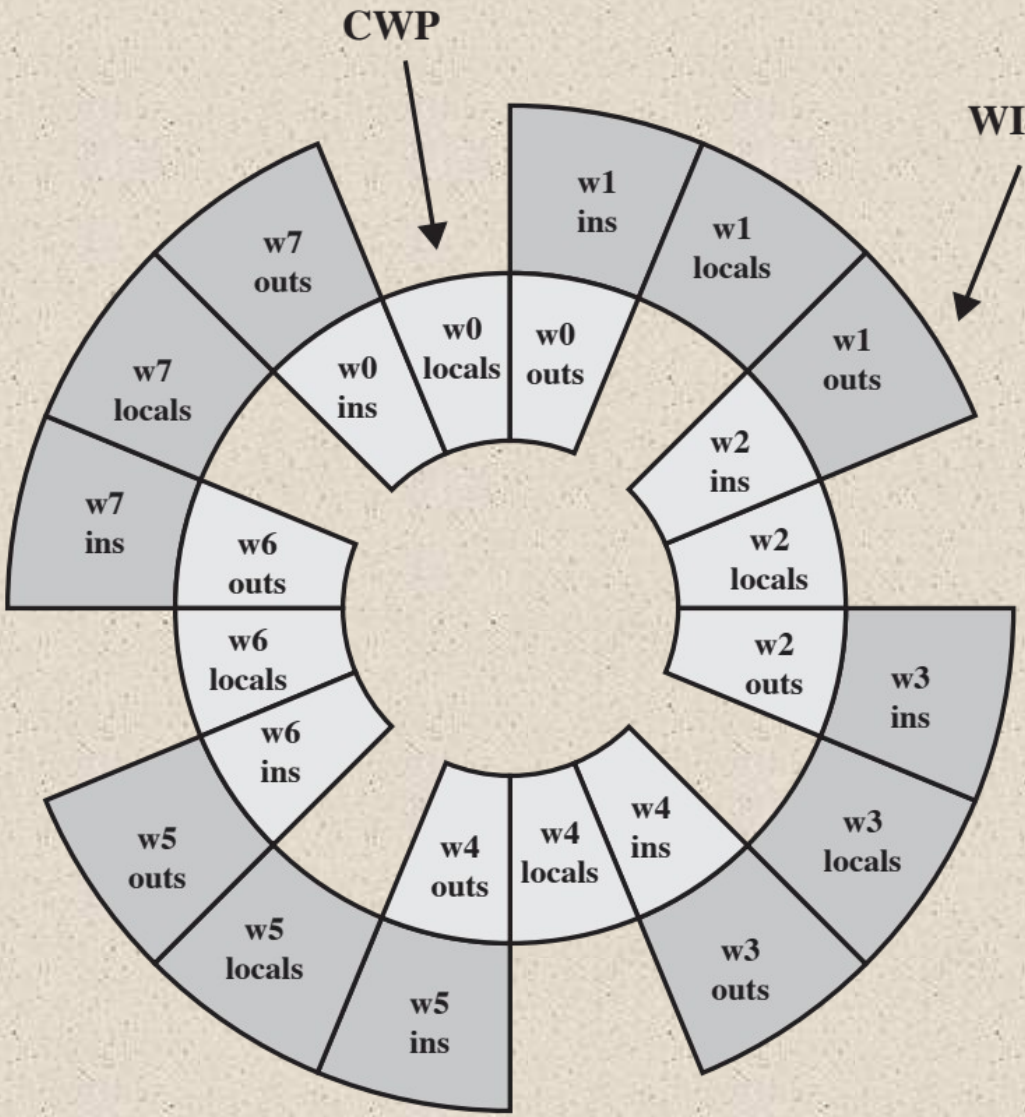


Figure 15.13 Eight Register Windows Forming a Circular Stack in SPARC

Table 15.10
Synthesizing Other Addressing Modes with SPARC Addressing Modes

Instruction Type	Addressing Mode	Algorithm	SPARC Equivalent
Register-to-register	Immediate	operand = A	S2
Load, store	Direct	EA = A	R_0 + S2
Register-to-register	Register	EA = R	R_{S1}, R_{S2}
Load, store	Register Indirect	EA = (R)	R_{S1} + 0
Load, store	Displacement	EA = (R) + A	R_{S1} + S2

S2 = either a register operand or a 13-bit immediate operand

(Table can be found on page 568 in the

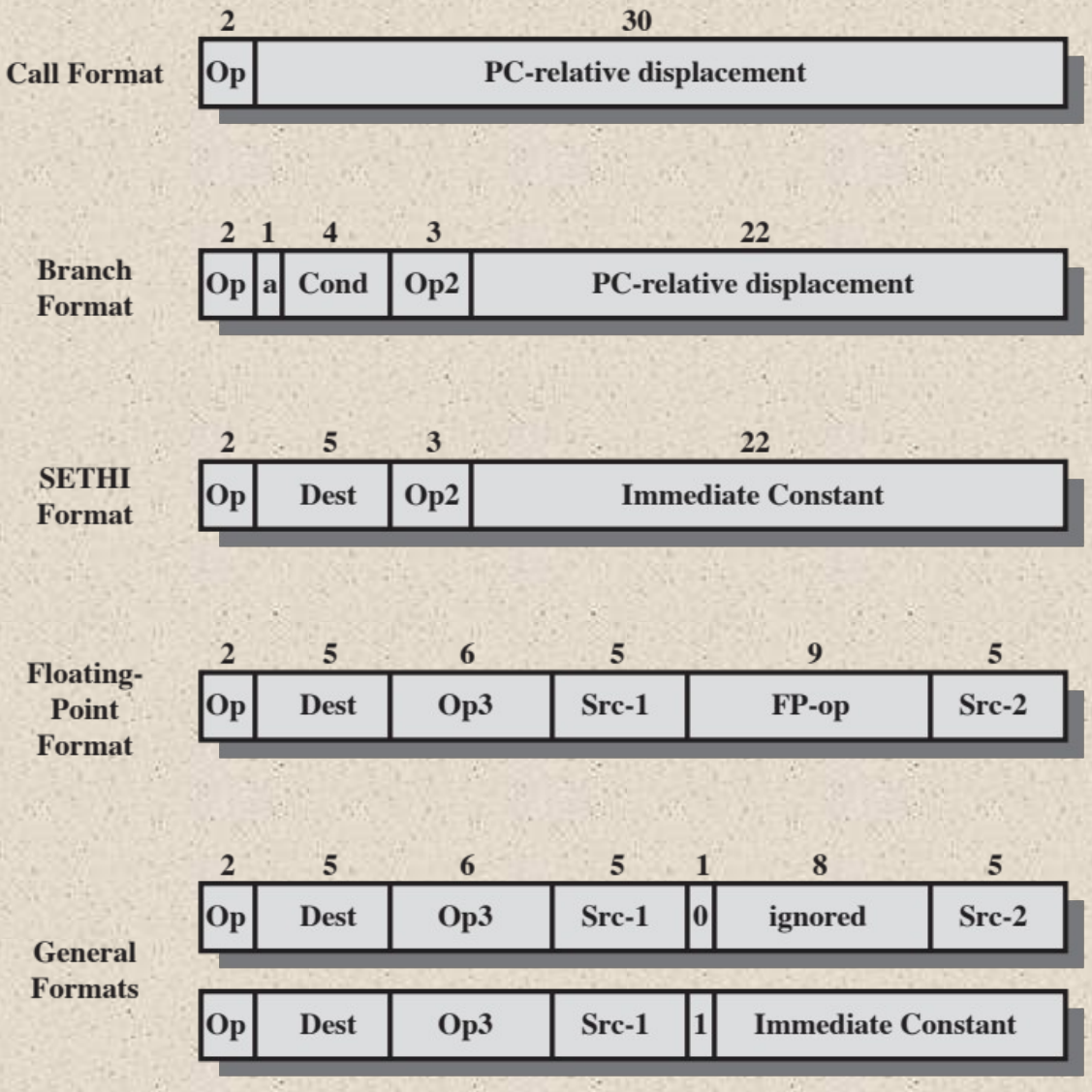


Figure 15.14 SPARC Instruction Formats

RISC versus CISC Controversy

- Quantitative
  - Compare program sizes and execution speeds of programs on RISC and CISC machines that use comparable technology
- Qualitative
  - Examine issues of high level language support and use of VLSI real estate
- Problems with comparisons:
  - No pair of RISC and CISC machines that are comparable in life-cycle cost, level of technology, gate complexity, sophistication of compiler, operating system support, etc.
  - No definitive set of test programs exists
  - Difficult to separate hardware effects from complier effects
  - Most comparisons done on “toy” rather than commercial products
  - Most commercial devices advertised as RISC possess a mixture of RISC and CISC characteristics

Summary

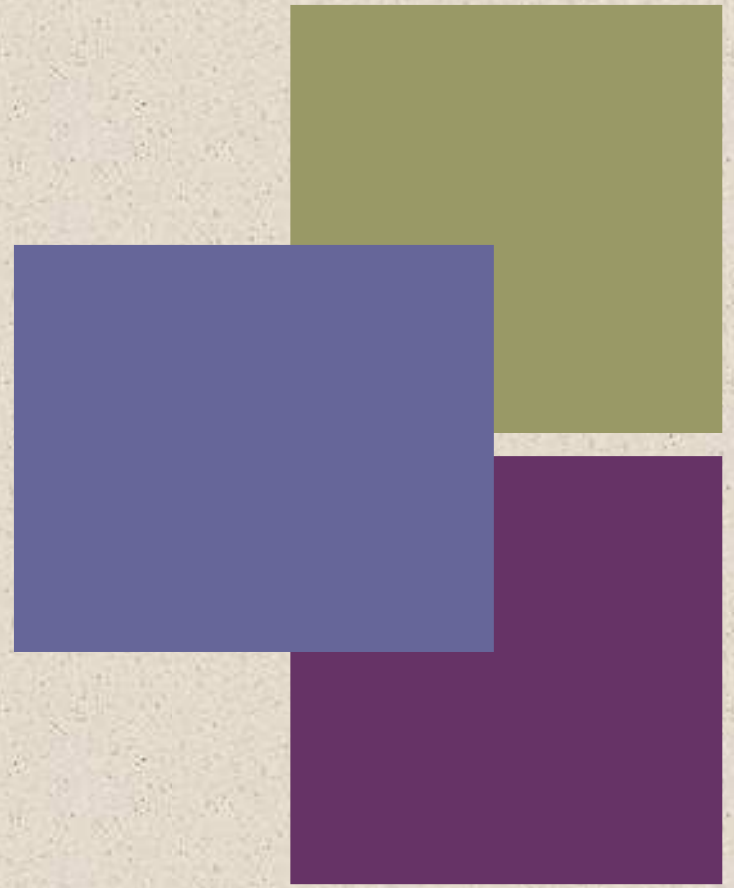
Chapter 15

- Instruction execution characteristics
  - Operations
  - Operands
  - Procedure calls
  - Implications
- The use of a large register file
  - Register windows
  - Global variables
  - Large register file versus cache
- Reduced instruction set architecture
  - Characteristics of RISC
  - CISC versus RISC characteristics

Reduced Instruction Set Computers (RISC)

- RISC pipelining
  - Pipelining with regular instructions
  - Optimization of pipelining
- MIPS R4000
  - Instruction set
  - Instruction pipeline
- SPARC
  - SPARC register set
  - Instruction set
  - Instruction format
- Compiler-based register optimization
- RISC versus CISC controversy

William Stallings
Computer Organization and Architecture
10th Edition



Chapter 16

Instruction-Level Parallelism and Superscalar Processors

Superscalar

Overview

Term first coined in 1987

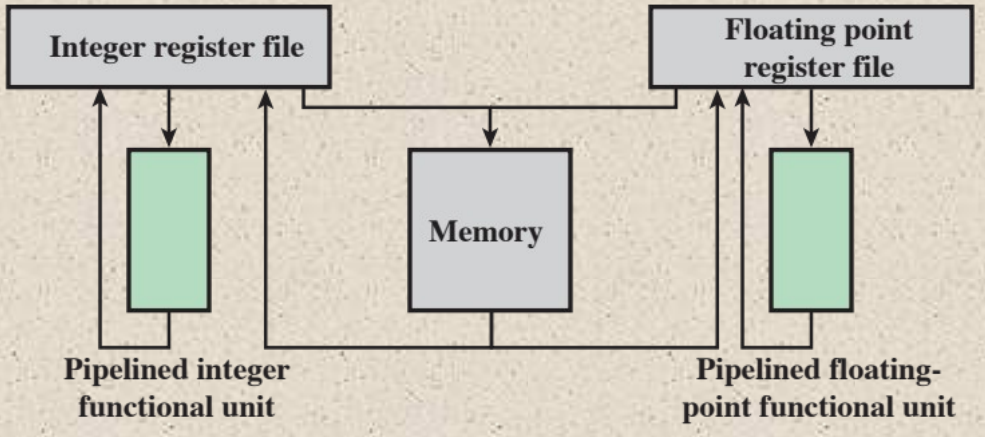
Refers to a machine that is designed to improve the performance of the execution of scalar instructions

In most applications the bulk of the operations are on scalar quantities

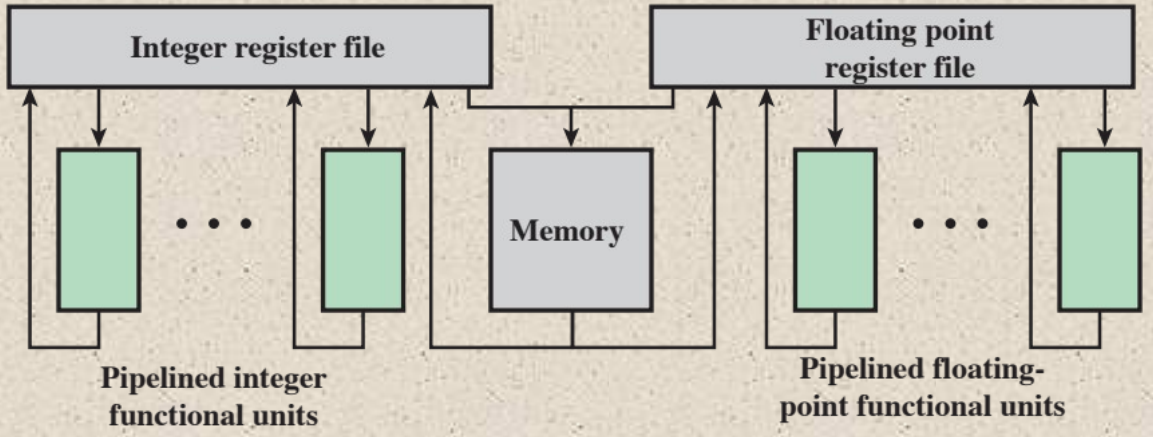
Represents the next step in the evolution of high-performance general-purpose processors

Essence of the approach is the ability to execute instructions independently and concurrently in different pipelines

Concept can be further exploited by allowing instructions to be executed in an order different from the program order



(a) Scalar organization



(b) Superscalar organization

Figure 16.1 Superscalar Organization Compared to Ordinary Scalar Organization

Table 16.1
Reported Speedups of Superscalar-Like Machines

Reference	Speedup
[TJAD70]	1.8
[KUCK77]	8
[WEIS84]	1.58
[ACOS86]	2.7
[SOHI90]	1.8
[SMIT89]	2.3
[JOUP89b]	2.2
[LEE91]	7

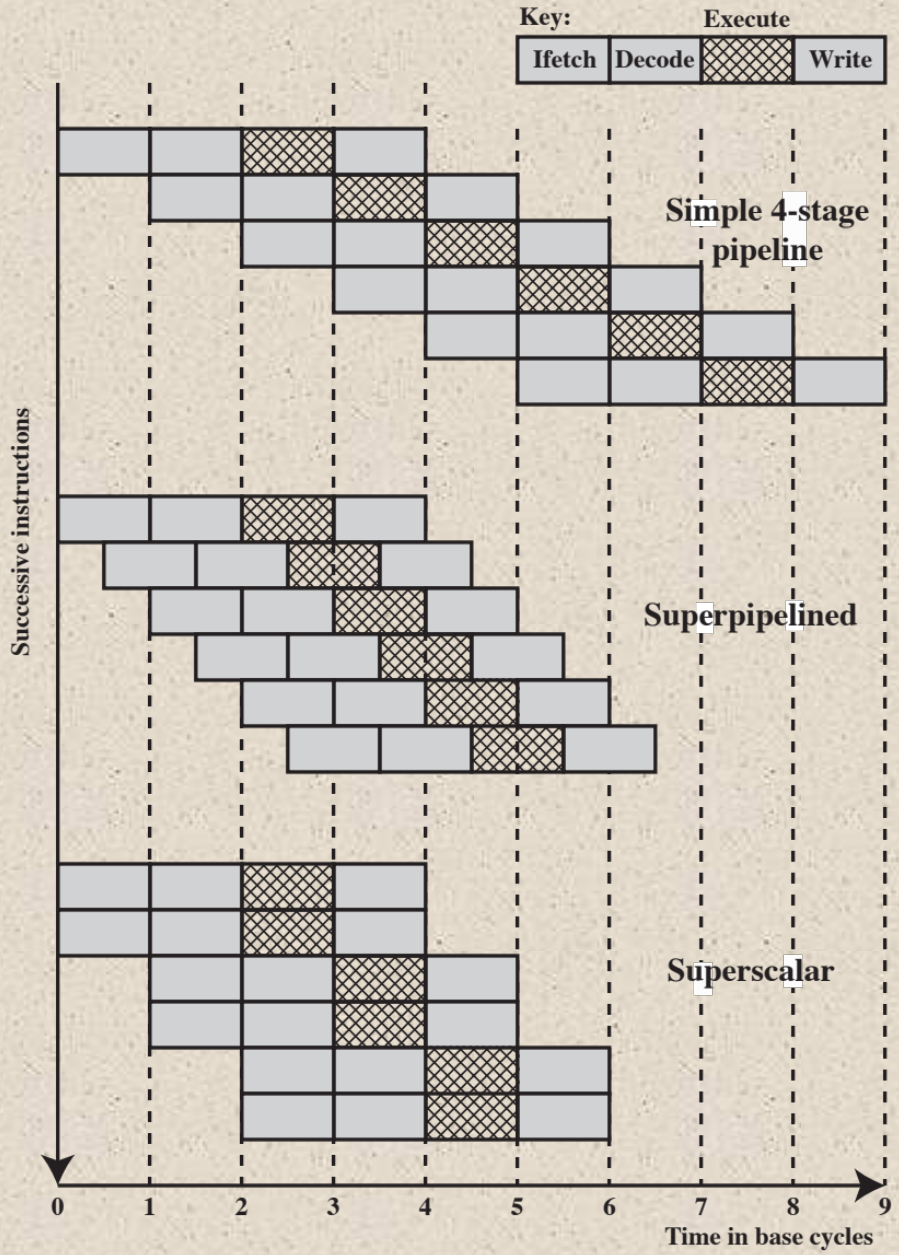


Figure 16.2 Comparison of Superscalar and Superpipeline Approaches

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

Figure 16.3 Effect of Dependencies

Design Issues

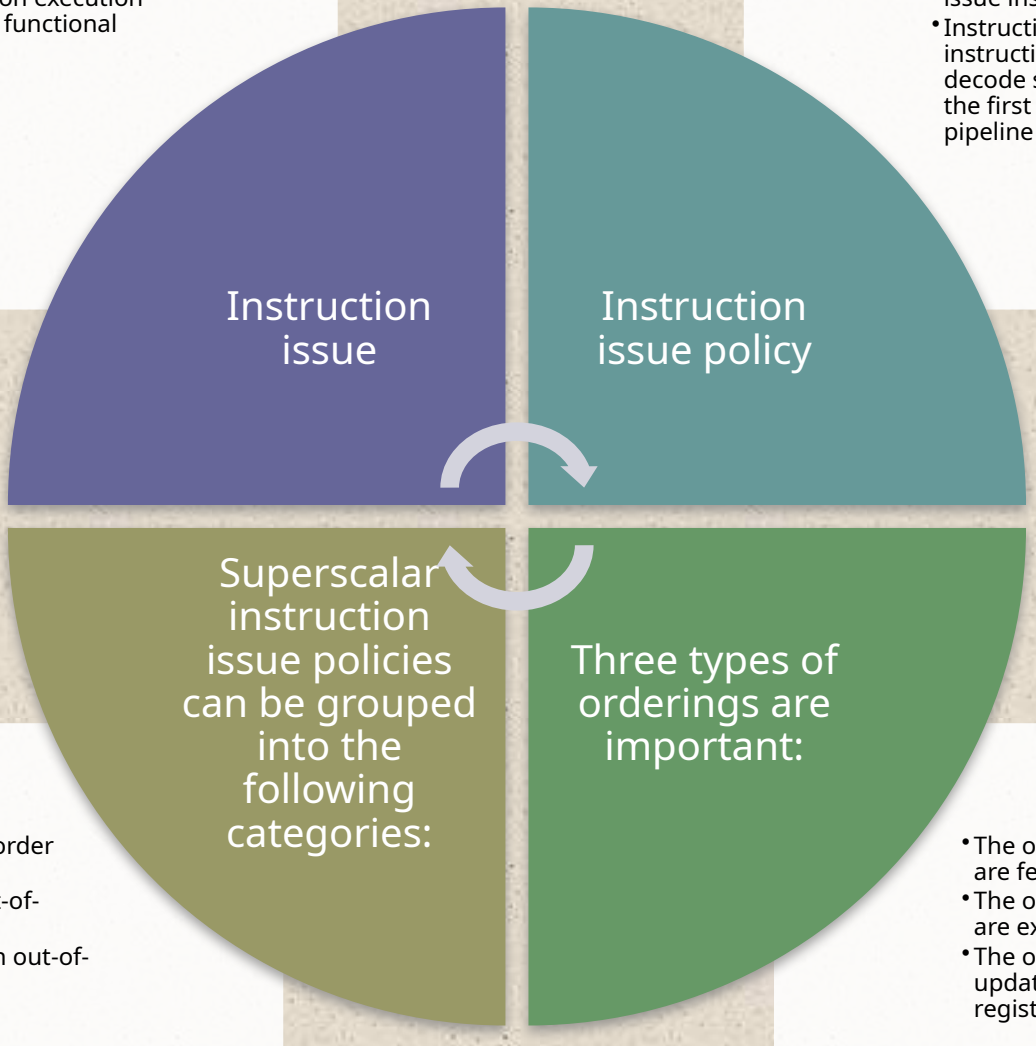
Instruction-Level Parallelism and Machine Parallelism

- 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

Instruction Issue Policy

- Refers to the process of initiating instruction execution in the processor's functional units

- Refers to the protocol used to issue instructions
- Instruction issue occurs when instruction moves from the decode stage of the pipeline to the first execute stage of the pipeline



- In-order issue with in-order completion
- In-order issue with out-of-order completion
- Out-of-order issue with out-of-order completion

- The order in which instructions are fetched
- The order in which instructions are executed
- The order in which instructions update the contents of register and memory locations

Decode		Execute			Write		Cycle
I1	I2						1
I3	I4	I1	I2				2
I3	I4	I1					3
	I4			I3	I1	I2	4
I5	I6			I4			5
	I6		I5		I3	I4	6
			I6				7
					I5	I6	8

(a) In-order issue and in-order completion

Decode		Execute			Write		Cycle
I1	I2						1
I3	I4	I1	I2				2
	I4	I1		I3	I2		3
I5	I6			I4	I1	I3	4
	I6		I5		I4		5
			I6		I5		6
					I6		7

(b) In-order issue and out-of-order completion

Decode		Window	Execute			Write		Cycle
I1	I2							1
I3	I4	*I1,I2*	I1	I2				2
I5	I6	*I3,I4*	I1		I3	I2		3
		I4,I5,I6		I6	I4	I1	I3	4
		I5		I5		I4	I6	5
						I5		6

(c) Out-of-order issue and out-of-order completion

Figure 16.4 Superscalar Instruction Issue and Completion Policies

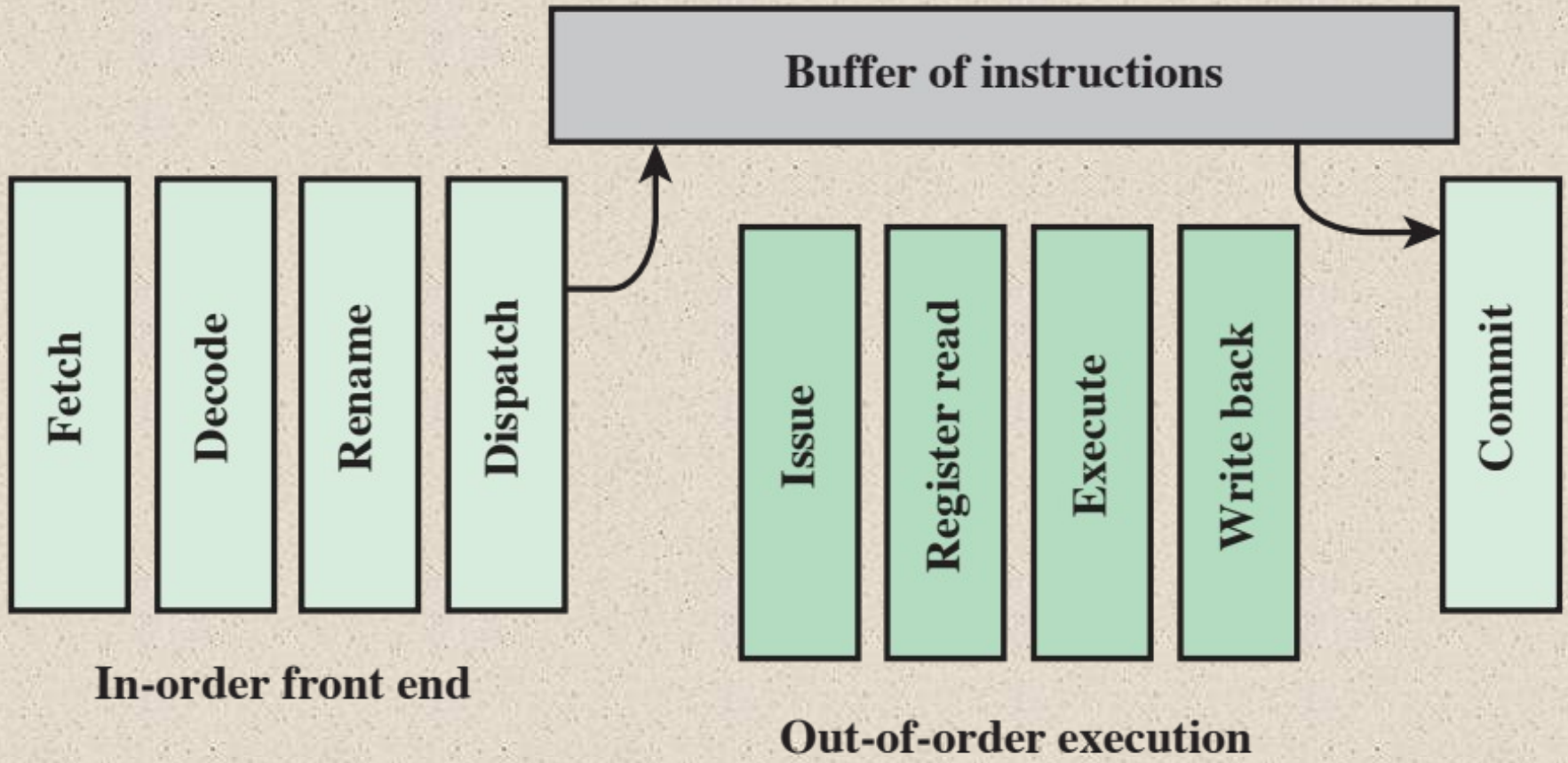


Figure 16.5 Organization for Out-of-Order Issue with Out-of-Order Completion

Register Renaming

Output and antidependencies occur because register contents may not reflect the correct ordering from the program

May result in a pipeline stall

Registers allocated dynamically

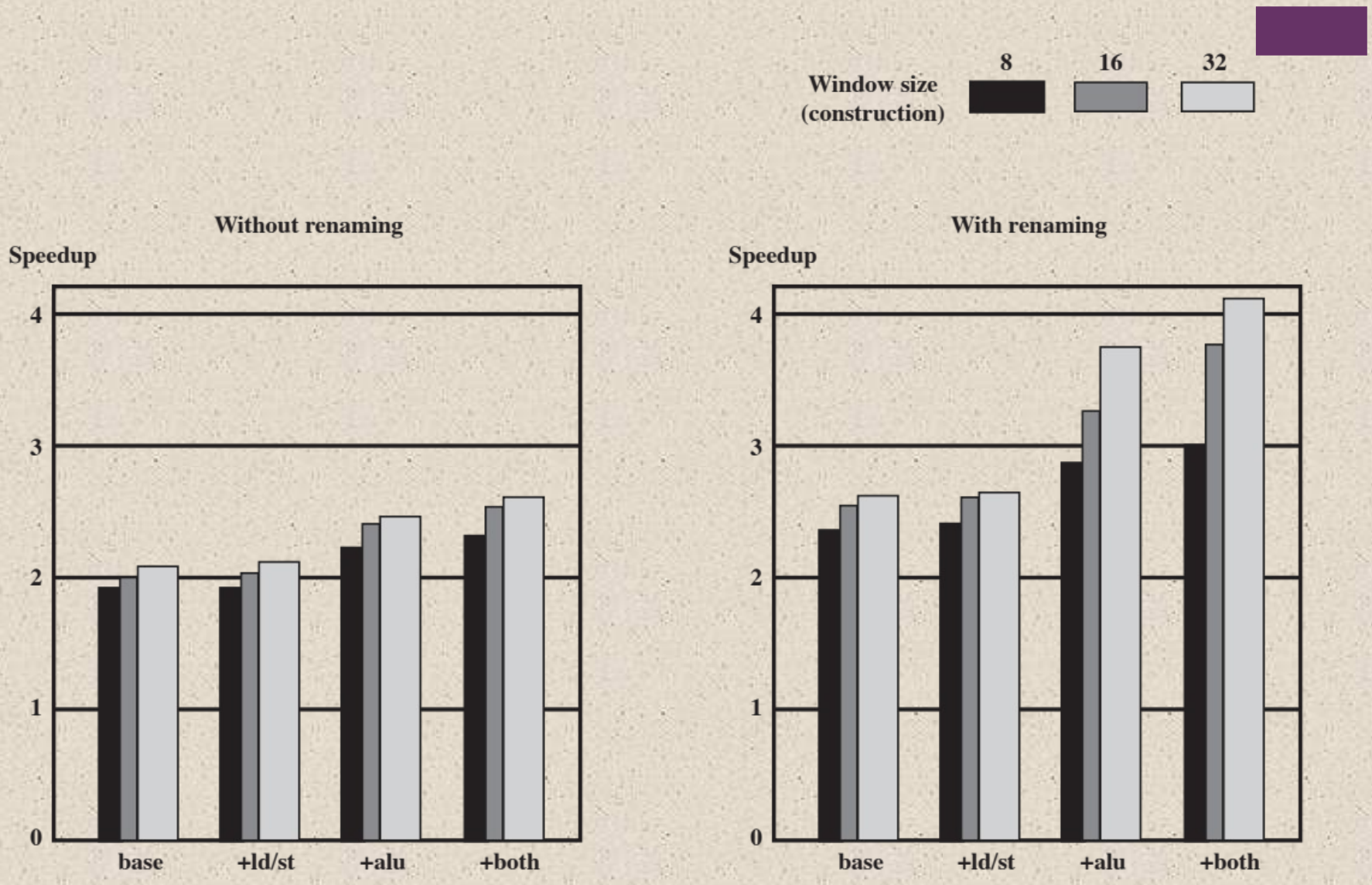


Figure 16.6 Speedups of Various Machine Organizations Without Procedural Dependencies

Branch Prediction

- Any high-performance pipelined machine must address the issue of dealing with branches
- Intel 80486 addressed the problem by fetching both the next sequential instruction after a branch and speculatively fetching the branch target instruction
- RISC machines:
  - Delayed branch strategy was explored
  - Processor always executes the single instruction that immediately follows the branch
  - Keeps the pipeline full while the processor fetches a new instruction stream
- Superscalar machines:
  - Delayed branch strategy has less appeal
  - Have returned to pre-RISC techniques of branch prediction

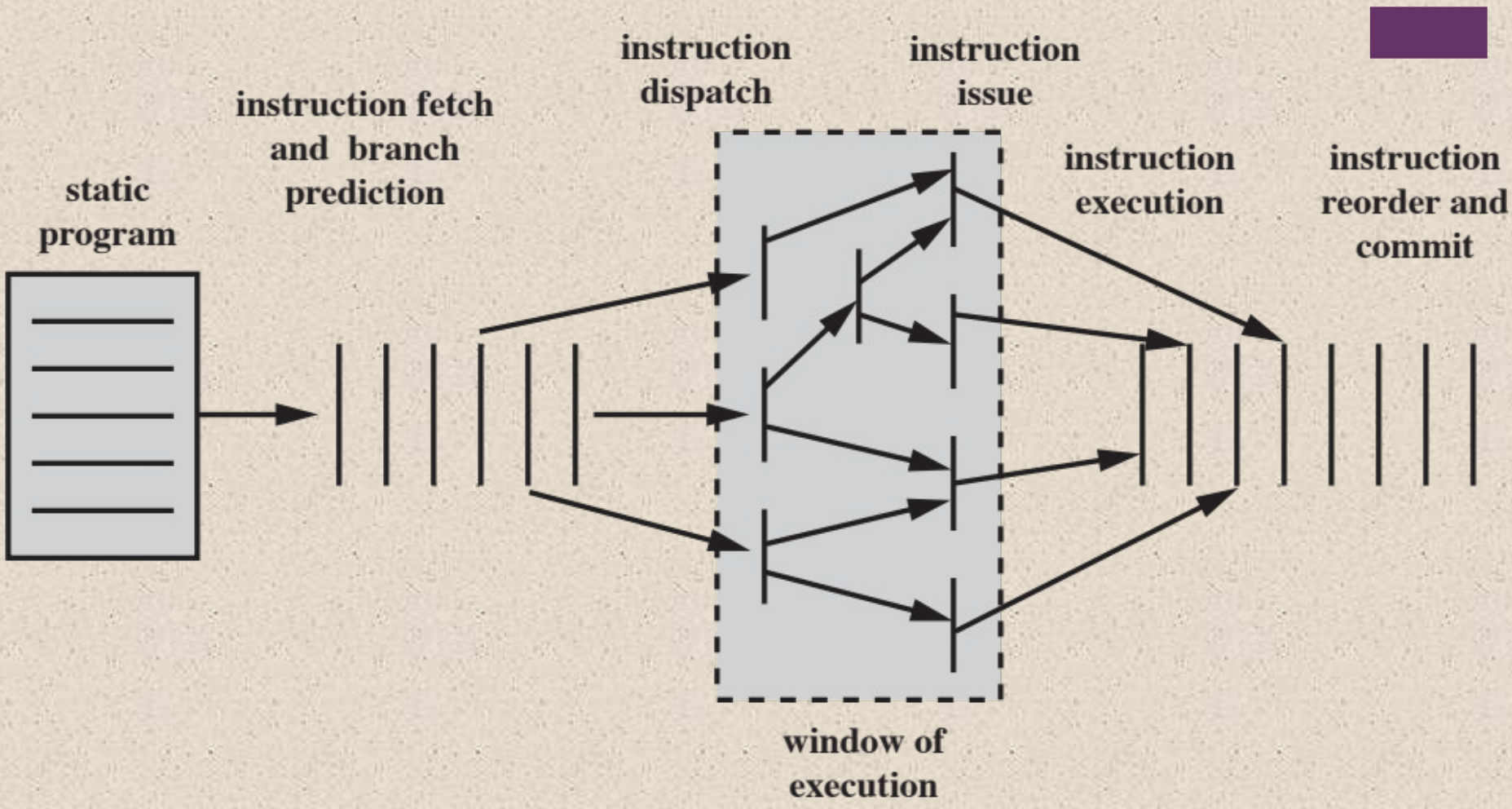


Figure 16.7 Conceptual Depiction of Superscalar Processing

Superscalar Implementation

Key elements:

- Instruction fetch strategies that simultaneously fetch multiple instruction
- Logic for determining true dependencies involving register values, and mechanisms for communicating these values to where they are needed during execution
- Mechanisms for initiating, or issuing, multiple instructions in parallel
- Resources for parallel execution of multiple instructions, including multiple pipelined functional units and memory hierarchies capable of simultaneously servicing multiple memory references
- Mechanisms for committing the process state in correct order

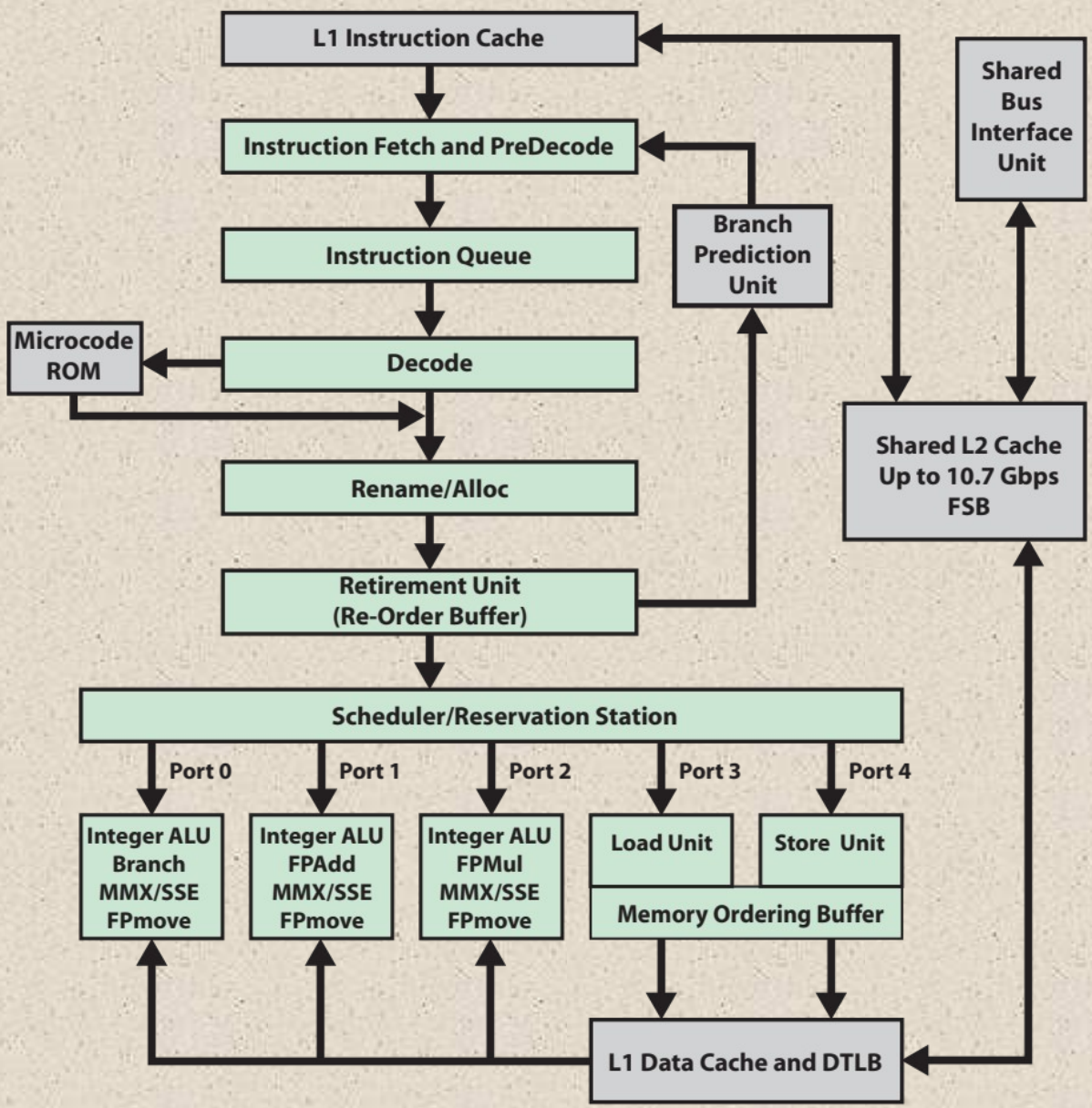


Figure 16.8 Intel Core Microarchitecture

Table 16.2

Cache/Memory Parameters and Performance of Processors Based on Intel Core Microarchitecture

(a) Cache Parameters

Cache Level	Capacity	Associativity (ways)	Line Size (bytes)	Write Update Policy
L1 data	32 kB	8	64	Writeback
L1 instruction	32 kB	8	N/A	N/A
L2 (shared)1	2, 4 MB	8 or 16	64	Writeback
L2 (shared)2	3, 6 MB	12 or 24	64	Writeback
L3 (shared)2	8, 12, 16 MB	15	64	Writeback

Notes:
1. Intel Core Microarchitecture
2. Enhanced Intel Core Microarchitecture

(b) Load/Store Performance

Data Locality	Load		Store	
	Latency	Throughput	Latency	Throughput
L1 data cache	3 clock cycles	1 clock cycles	2 clock cycles	3 clock cycles
L1 data cache of the other core in modified state	14 clock cycles + 5.5 bus cycles	14 clock cycles + 5.5 bus cycles	14 clock cycles + 5.5 bus cycles	N/A
L2 cache	14	3	14	3
Memory	14 clock cycles + 5.5 bus cycles + memory latency	Depends on bus read protocol	14 clock cycles + 5.5 bus cycles + memory latency	Depends on bus read protocol

(Table can be found on page 592 in the textbook.)

Front End

Consists of three major components:

- Branch prediction unit (BPU)
- Instruction fetch and predecode unit
- Instruction queue and decode unit

Branch Prediction Unit

- Helps the instruction fetch unit fetch the most likely instruction to be executed by predicting the various branch types:
  - Conditional
  - Indirect
  - Direct
  - Call
  - Return
- Uses dedicated hardware for each branch type
- Enables the processor to begin executing instructions long before the branch outcome is decided
- A branch target buffer (BTB) is maintained that caches information about recently encountered branch instructions

Instruction Fetch and Predecode

Comprises:

- The instruction translation lookaside buffer (ITLB)
- An instruction prefetcher
- The instruction cache
- The predecode logic

The predecode unit accepts the sixteen bytes from the instruction cache or prefetch buffers and carries out the following tasks:

- Determine the length of the instructions
- Decode all prefixes associated with instructions
- Mark various properties of instruction for the decoders

Predecode unit can write up to six instructions per cycle into the instruction queue

- If a fetch contains more than six instructions, the predecoder continues to decode up to six instructions per cycle until all instruction in the fetch are written to the instruction queue
- Subsequent fetches can only enter predecoding after the current fetch completes

Instruction Queue and Decode Unit

- Fetched instructions are placed in an instruction queue
  - From there the decode unit scans the bytes to determine instruction boundaries
  - The decoder translates each machine instruction into from one to four micro-ops
    - Each of which is a 118-bit RISC instruction
- A few instructions require more than four micro-ops so they are transferred to microcode ROM, which contains the series of micro-ops (five or more) associated with a complex machine instruction
- The resulting micro-op sequence is delivered to the rename/allocator module

Out-of-Order Execution Logic

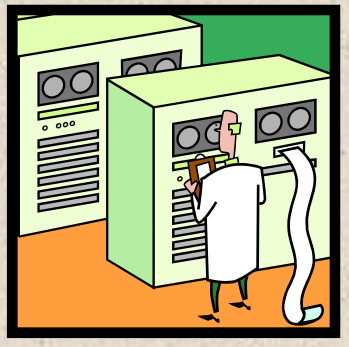
- This part of the processor reorders micro-ops to allow them to execute as quickly as their input operands are ready
- Allocate stage
  - Allocates resources required for execution
  - Performs the following functions:
    - If a needed resource is unavailable for one of the three micro-ops arriving at the allocator during a clock cycle, the allocator stalls the pipeline
    - Allocates a reorder buffer (ROB) entry which tracks the completion status of one of the 126 micro-ops that could be in process at any time
    - Allocates one of the 128 integer or floating-point register entries for the result data value of the micro-op, and possibly a load or store buffer used to track one of the 48 loads or 24 stores in the machine pipeline
    - Allocates an entry in one of the two micro-op queues in front of the instruction schedulers

Reorder Buffer (ROB)

Circular buffer that can hold up to 126 micro-ops and also contains the 128 hardware registers

Each buffer entry consists of the following fields:

- **State**
  - Indicates whether this micro-op is scheduled for execution, has been dispatched for execution, or has completed execution and is ready for retirement
- **Memory address**
  - The address of the Pentium instruction that generated the micro-op
- **Micro-op**
  - The actual operation
- **Alias register**
  - If the micro-op references one of the 16 architectural registers, this entry redirects that reference to one of the 128 hardware registers



- Register renaming
  - The rename stage remaps references to the 16 architectural registers into a set of 128 physical registers
- Micro-op scheduling and dispatching
  - Schedulers are responsible for retrieving micro-ops from the micro-op queues and dispatching these for execution
- Micro-op queuing
  - After resource allocation and register renaming, micro-ops are placed in one of two micro-op queues, where they are held until there is room in the schedulers
- Integer and floating-point execution units
  - The execution units retrieve values from the register files as well as from the L1 data cache

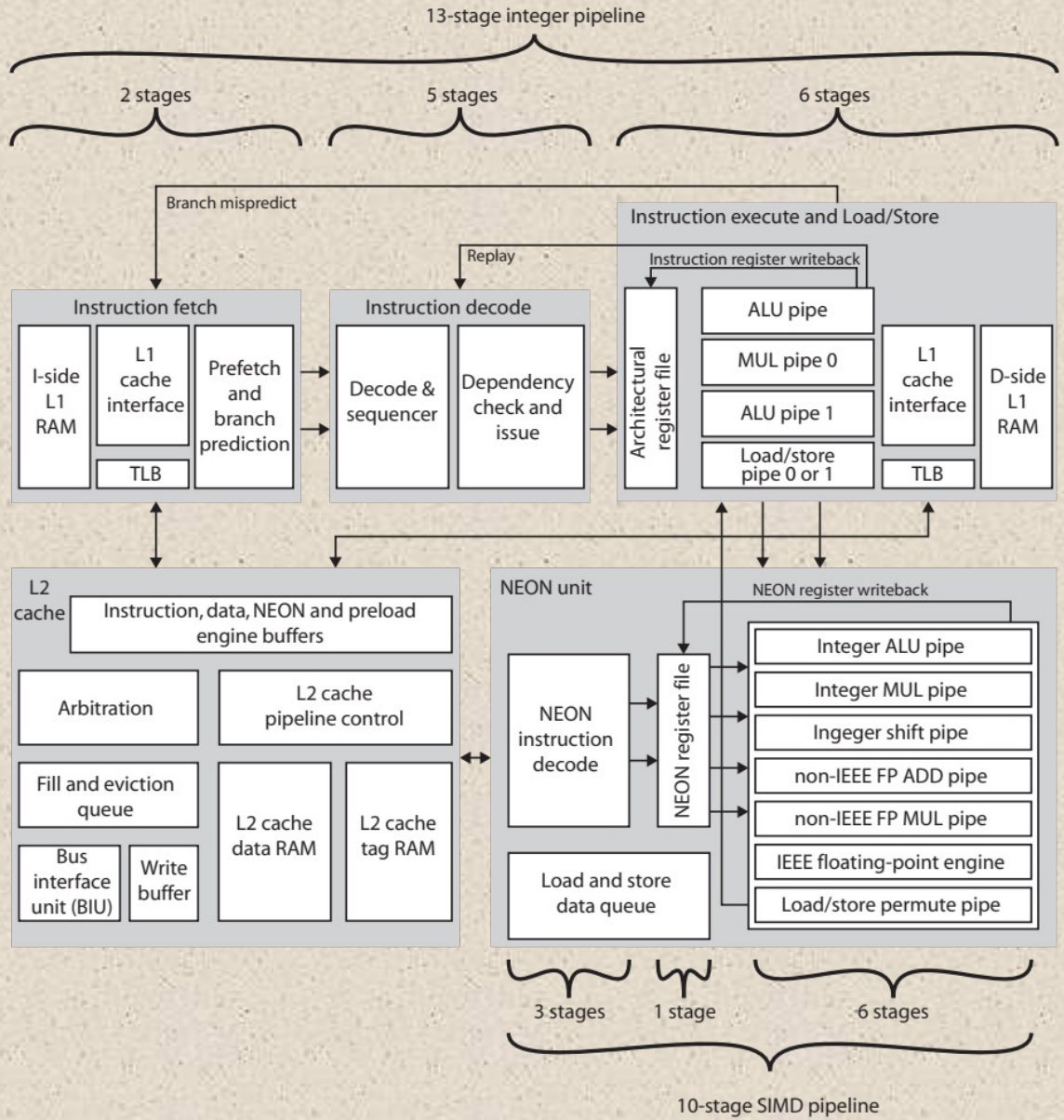


Figure 16.9 Architectural Block Diagram of ARM Cortex-A8

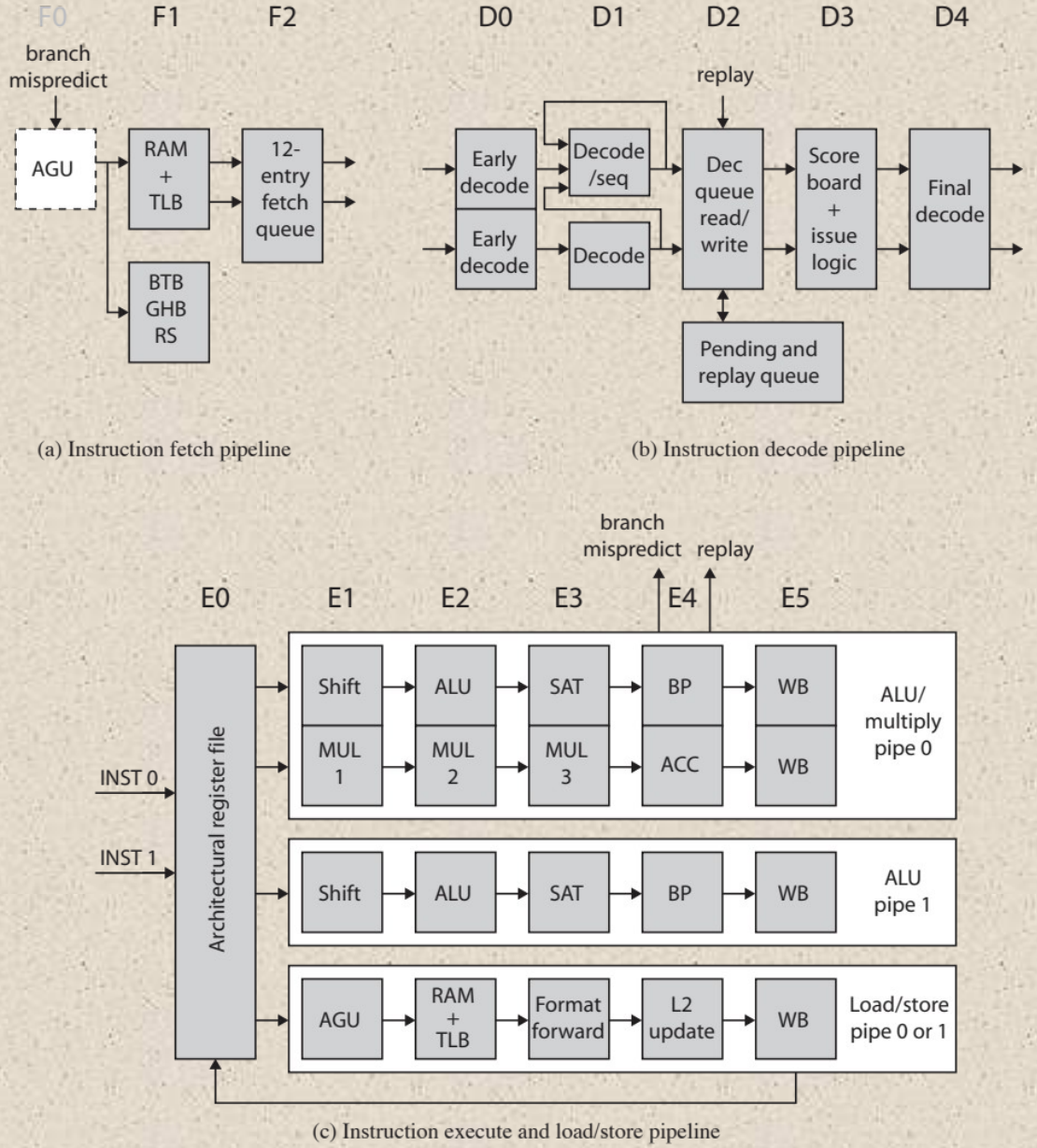


Figure 16.10 ARM Cortex-A8 Integer Pipeline

Instruction Fetch Unit

- Predicts instruction stream
- Fetches instructions from the L1 instruction cache
- Places the fetched instructions into a buffer for consumption by the decode pipeline
- Also includes the L1 instruction cache
- Speculative (there is no guarantee that they are executed)
- Branch or exceptional instruction in the code stream can cause a pipeline flush
- Can fetch up to four instructions per cycle

- F0
  - Address generation unit (AGU) generates a new virtual address
  - Not counted as part of the 13-stage pipeline
- F1
  - The calculated address is used to fetch instructions from the L1 instruction cache
  - In parallel, the fetch address is used to access branch prediction arrays
- F3
  - Instruction data are placed in the instruction queue
  - If an instruction results in branch prediction, new target address is sent to the address generation unit

Instruction Decode Unit

- Decodes and sequences all ARM and Thumb instructions
- Dual pipeline structure, *pipe0* and *pipe1*
  - Two instructions can progress at a time
  - Pipe0 contains the older instruction in program order
  - If instruction in pipe0 cannot issue, instruction in pipe1 will not issue
- All issued instructions progress in order
- Results written back to register file at end of execution pipeline
  - Prevents WAR hazards
  - Keeps track of WAW hazards and recovery from flush conditions straightforward
- Main concern of decode pipeline is prevention of RAW hazards

Instruction Processing Stages

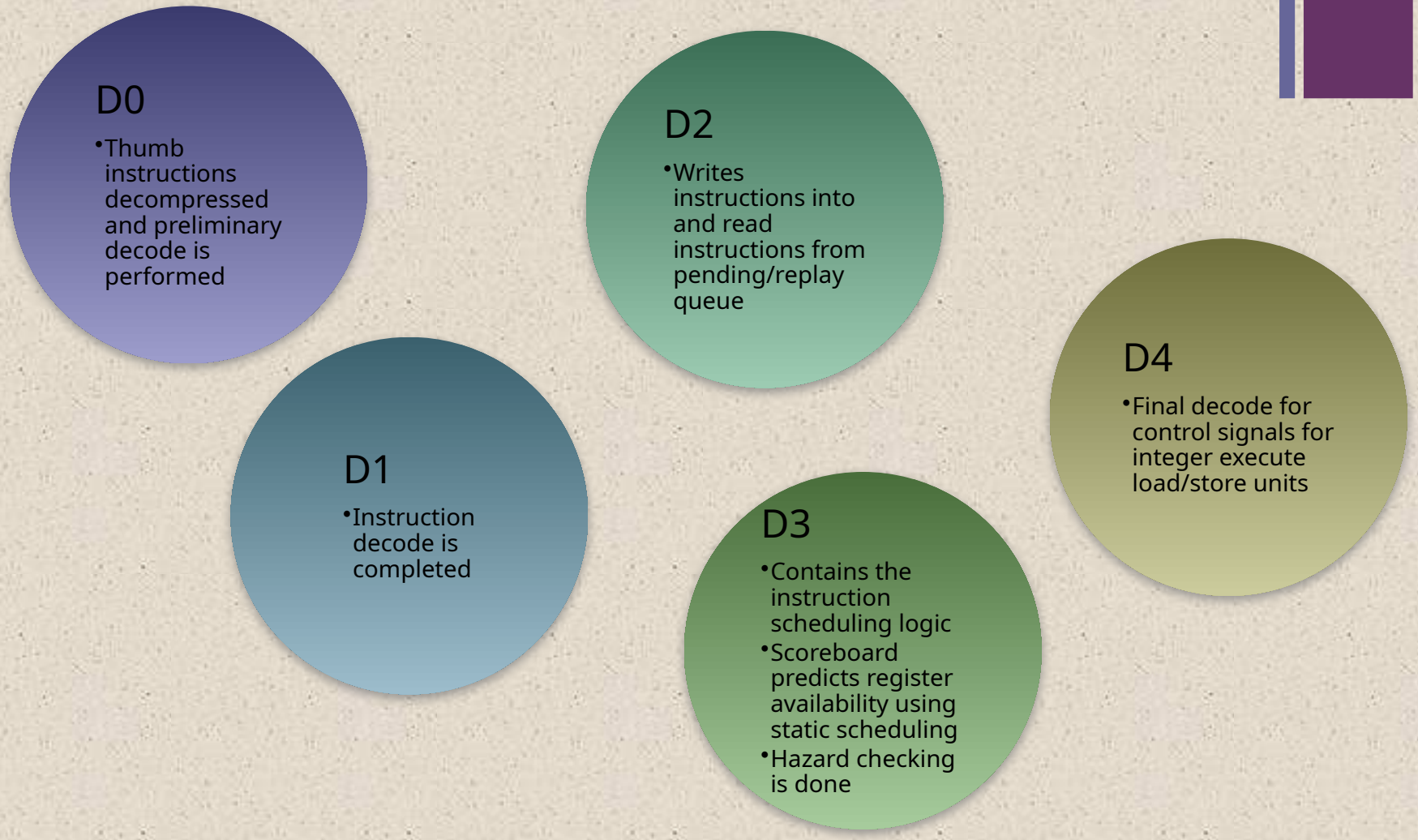


Table 16.3
Cortex-A8 Memory System Effects on Instruction Timings

Replay event	Delay	Description
Load data miss	8 cycles	**1.** A load instruction misses in the L1 data cache. **2.** A request is then made to the L2 data cache. **3.** If a miss also occurs in the L2 data cache, then a second replay occurs. The number of stall cycles depends on the external system memory timing. The minimum time required to receive the critical word for an L2 cache miss is approximately 25 cycles, but can be much longer because of L3 memory latencies.
Data TLB miss	24 cycles	**1.** A table walk because of a miss in the L1 TLB causes a 24-cycle delay, assuming the translation table entries are found in the L2 cache. **2.** If the translation table entries are not present in the L2 cache, the number of stall cycles depends on the external system memory timing.
Store buffer full	8 cycles plus latency to drain fill buffer	**1.** A store instruction miss does not result in any stalls unless the store buffer is full. **2.** In the case of a full store buffer, the delay is at least eight cycles. The delay can be more if it takes longer to drain some entries from the store buffer.
Unaligned load or store request	8 cycles	**1.** If a load instruction address is unaligned and the full access is not contained within a 128-bit boundary, there is a 8-cycle penalty. **2.** If a store instruction address is unaligned and the full access is not contained within a 64-bit boundary, there is a 8-cycle penalty.

(Table can be found on page 601 in the

Table 16.4
Cortex-A8 Dual-Issue Restrictions

Restriction type	Description	Example	Cycle	Restriction

Load/store resource hazard	There is only one LS pipeline. Only one LS instruction can be issued per cycle. It can be in pipeline 0 or pipeline 1	LDR r5, [r6]
		STR r7, [r8]
		MOV r9, r10
			1
			2
			2
				
				Wait for LS unit
				Dual issue possible

Multiply resource hazard	There is only one multiply pipeline, and it is only available in pipeline 0.	ADD r1, r2, r3
		MUL r4, r5, r6
		MUL r7, r8, r9
			1
			2
			3
				
				Wait for pipeline 0
				Wait for multiply unit

Branch resource hazard	There can be only one branch per cycle. It can be in pipeline 0 or pipeline 1. A branch is any instruction that changes the PC.	BX r1 BEQ 0x1000 ADD r1, r2, r3	1 2 2	 Wait for branch Dual issue possible

Data output hazard	Instructions with the same destination cannot be issued in
	the same cycle. This can happen
	with conditional code.
		MOVEQ r1, r2
		MOVNE r1, r3
		
		LDR r5, [r6]
			1
			2
			
			2
				
				Wait because of output dependency
				Dual issue possible

Data source hazard	Instructions cannot be issued if their data is not available. See the scheduling tables for source requirements and stages results.	ADD r1, r2, r3
		ADD r4, r1, r6
		LDR r7, [r4]
			1
			2
			4
				
				Wait for r1
				Wait two cycles for r4

Multi-cycle instructions	Multi-cycle instructions must issue in pipeline 0 and can only dual issue in their last iteration.	MOV r1, r2
		LDM r3, {r4-r7}
		LDM (cycle 2)
		LDM (cycle 3)
		
		ADD r8, r9, r10
			1
			2
			3
			4
			
			4
				Wait for pipeline 0, transfer r4
				Transfer r5, r6
				Transfer r7
				Dual issue possible on last transfer

(Table can be found on page 602 in the

Integer Execute Unit

- Consists of:
  - Two symmetric arithmetic logic unit (ALU) pipelines
  - An address generator for load and store instructions
  - The multiply pipeline
- The instruction execute unit:
  - Executes all integer ALU and multiply operations, including flag generation
  - Generates the virtual addresses for loads and stores and the base write-back value, when required
  - Supplies formatted data for stores and forwards data and flags
  - Processes branches and other changes of instruction stream and evaluates instruction condition codes

- For ALU instructions, either pipeline can be used, consisting of the following stages:
  - E0
    - Access register file
    - Up to six registers for two instructions
  - E1
    - Barrel shifter if needed.
  - E2
    - ALU function
  - E3
    - If needed, completes saturation arithmetic
  - E4
    - Change in control flow prioritized and processed
  - E5
    - Results written back to register

Load/Store Pipeline

- Runs parallel to integer pipeline
- E1
  - Memory address generated from base and index register
- E2
  - Address applied to cache arrays
- E3
  - Load -- data are returned and formatted
  - Store -- data are formatted and ready to be written to cache
- E4
  - Updates L2 cache, if required
- E5
  - Results are written back into the register file

Cycle	Program Counter	Instruction	Timing Description
1	0x00000ed0	BX r14	Dual issue pipeline 0
1	0x00000ee4	CMP r0,#0	Dual issue in pipeline 1
2	0x00000ee8	MOV r3,#3	Dual issue pipeline 0
2	0x00000eec	MOV r0,#0	Dual issue in pipeline 1
3	0x00000ef0	STREQ r3,[r1,#0]	Dual issue in pipeline 0, r3 not needed until E3
3	0x00000ef4	CMP r2,#4	Dual issue in pipeline 1
4	0x00000ef8	LDRLS pc,[pc,r2,LSL #2]	Single issue pipeline 0, +1 cycle for load to pc, no extra cycle for shift since LSL #2
5	0x00000f2c	MOV r0,#1	Dual issue with 2nd iteration of load in pipeline 1
6	0x00000f30	B {pc}+8	#0xf38 dual issue pipeline 0
6	0x00000f38	STR r0,[r1,#0]	Dual issue pipeline 1
7	0x00000f3c:	LDR pc,[r13],#4	Single issue pipeline 0, +1 cycle for load to pc
8	0x0000017c	ADD r2,r4,#0xc	Dual issue with 2nd iteration of load in pipeline 1
9	0x00000180	LDR r0,[r6,#4]	Dual issue pipeline 0
9	0x00000184	MOV r1,#0xa	Dual issue pipeline 1
12	0x00000188	LDR r0,[r0,#0]	Single issue pipeline 0: r0 produced in E3, required in E1, so +2 cycle stall
13	0x0000018c	STR r0,[r4,#0]	Single issue pipeline 0 due to LS resource hazard, no extra delay for r0 since produced in E3 and consumed in E3
14	0x00000190	LDR r0,[r4,#0xc]	Single issue pipeline 0 due to LS resource hazard
15	0x00000194	LDMFD r13!,{r4-r6,r14}	Load multiple: loads r4 in 1st cycle, r5 and r6 in 2nd cycle, r14 in 3rd cycle, 3 cycles total
17	0x00000198	B {pc}+0xda8	#0xf40 dual issue in pipeline 1 with 3rd cycle of LDM
18	0x00000f40	ADD r0,r0,#2 ARM	Single issue in pipeline 0
19	0x00000f44	ADD r0,r1,r0 ARM	Single issue in pipeline 0, no dual issue due to hazard on r0 produced in E2 and required in E2

Table 16.5

Cortex-A8 Example Dual Issue Instruction Sequence for Integer Pipeline

(Table can be found on page 603 in the

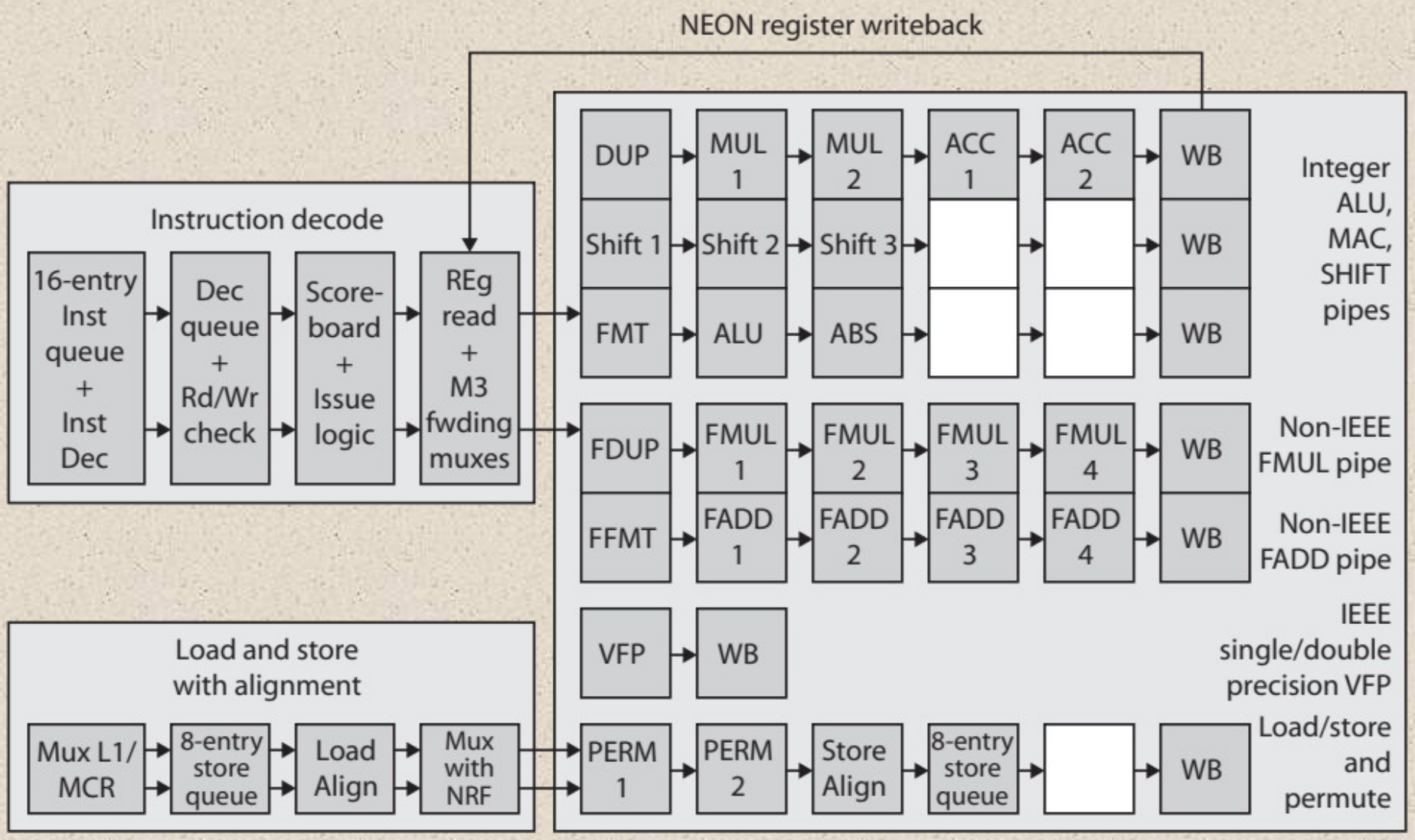


Figure 16.11 ARM Cortex-A8 NEON and Floating-Point Pipeline

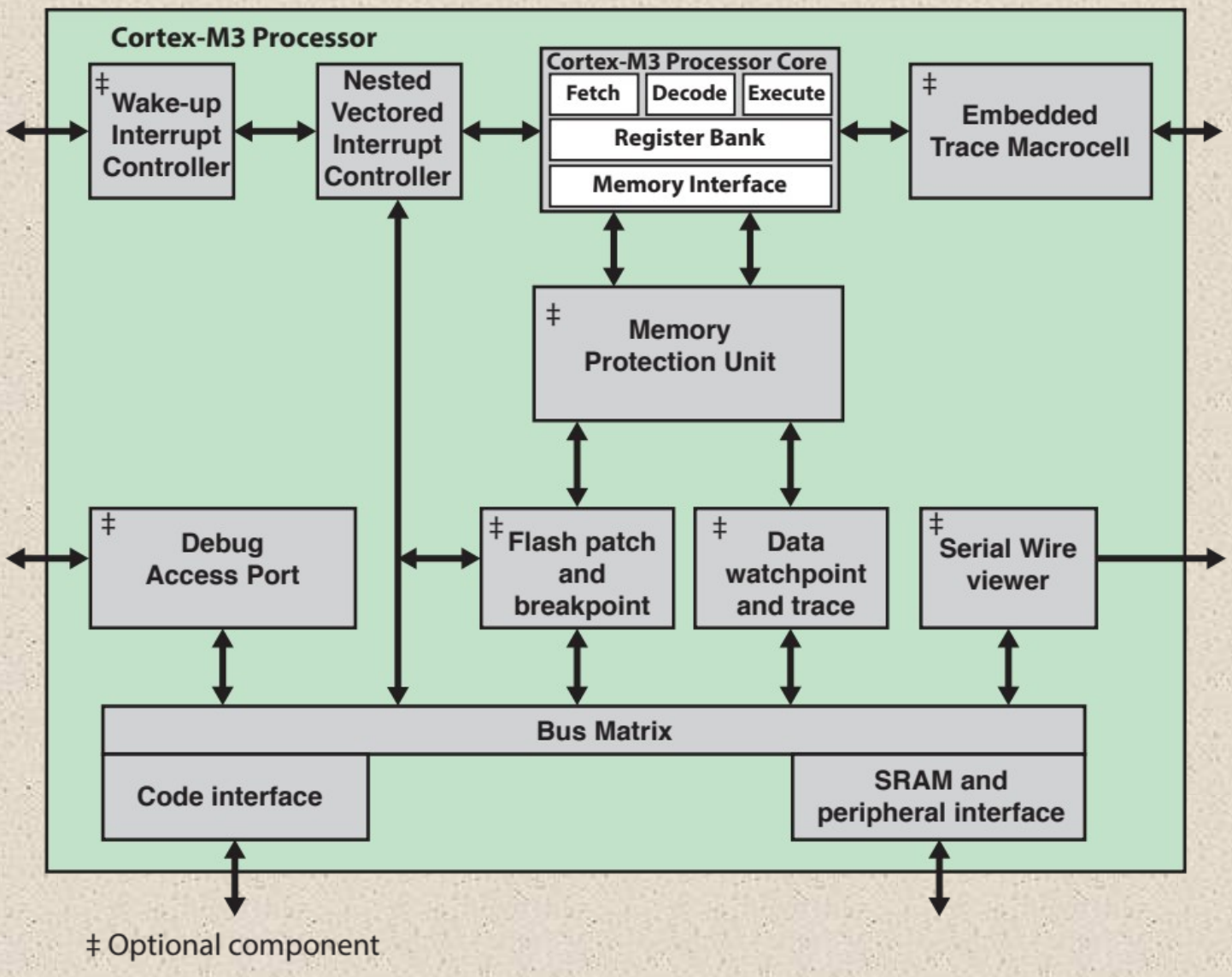


Figure 16.12 ARM Cortex-M3 Block Diagram

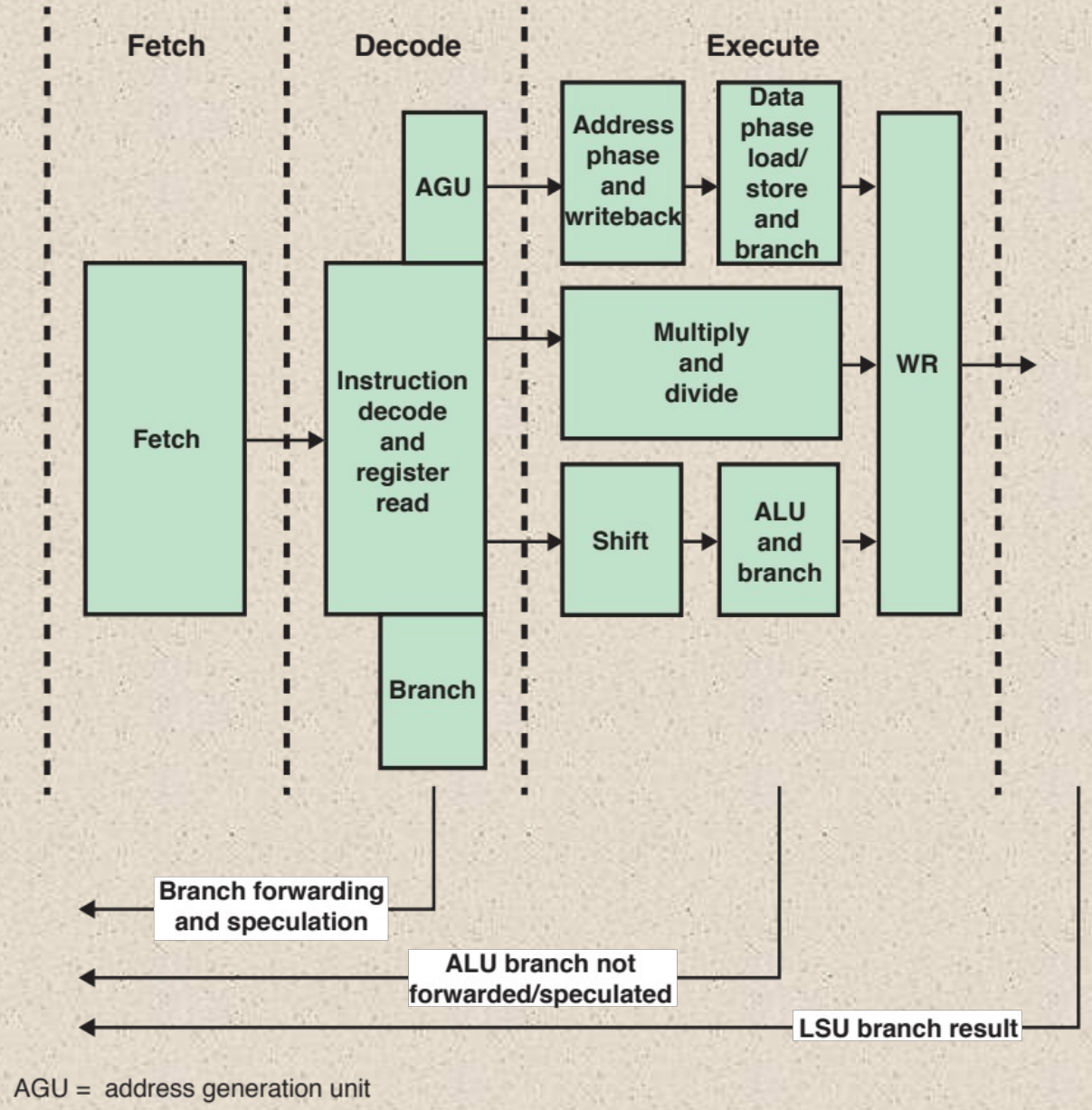


Figure 16.13 ARM Cortex-M3 Pipeline

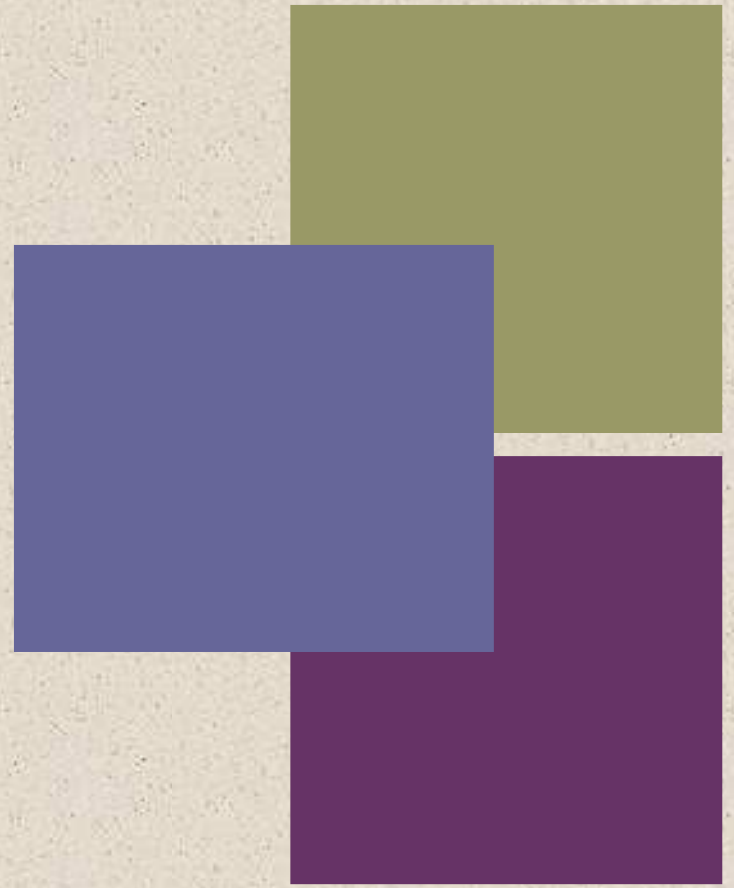
Summary

Chapter 16

Instruction-Level Parallelism and Superscalar Processors

- Superscalar versus Superpipelined
  - Constraints
- Design issues
  - Instruction-level parallelism
  - Machine parallelism
  - Instruction issue policy
  - Register renaming
  - Branch prediction
  - Superscalar execution
  - Superscalar implementation
- Intel core microarchitecture
  - Front end
  - Out-of-order execution logic
  - Integer and floating-point execution units
- ARM Cortex-A8
  - Instruction fetch unit
  - Instruction decode unit
  - Integer execute unit
  - SIMD and floating-point pipeline
- ARM Cortex-M3
  - Pipeline structure
  - Dealing with branches

William Stallings
Computer Organization and Architecture
10th Edition



Chapter 17

Parallel Processing

Multiple Processor Organization

- Single instruction, single data **(SISD)** stream
  - Single processor executes a single instruction stream to operate on data stored in a single memory
  - Uniprocessors fall into this category
- Multiple instruction, single data **(MISD)** stream
  - A sequence of data is transmitted to a set of processors, each of which executes a different instruction sequence
  - Not commercially implemented
- Single instruction, multiple data **(SIMD)** stream
  - A single machine instruction controls the simultaneous execution of a number of processing elements on a lockstep basis
  - Vector and array processors fall into this category
- Multiple instruction, multiple data **(MIMD)** stream
  - A set of processors simultaneously execute different instruction sequences on different data sets
  - SMPs, clusters and NUMA systems fit this category

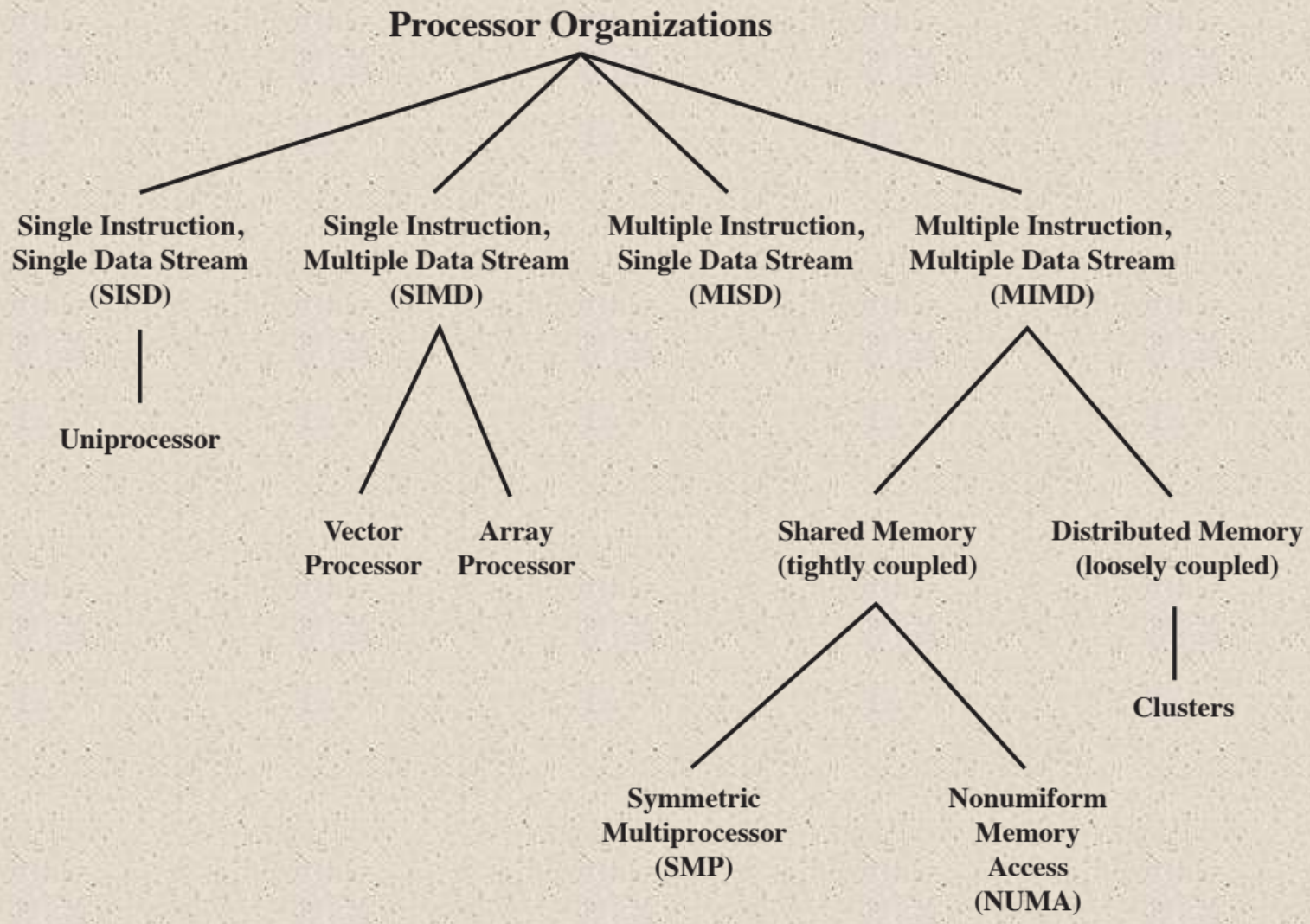


Figure 17.1 A Taxonomy of Parallel Processor Architectures

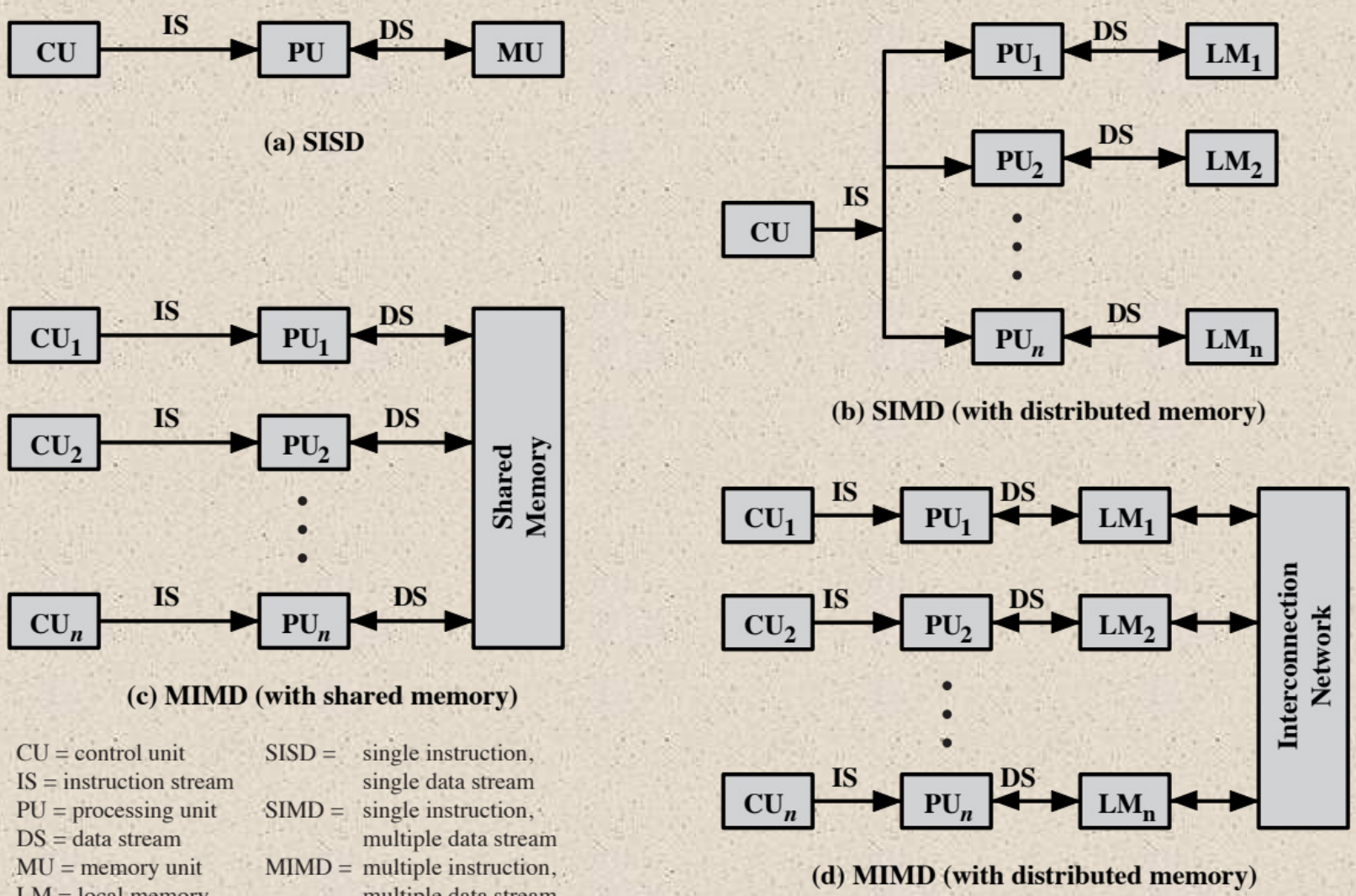


Figure 17.2 Alternative Computer Organizations

Symmetric Multiprocessor (SMP)

A stand alone computer with the following characteristics:

Two or more similar processors of comparable capacity	Processors share same memory and I/O facilities • Processors are connected by a bus or other internal connection • Memory access time is approximately the same for each processor	All processors share access to I/O devices • Either through same channels or different channels giving paths to same devices	All processors can perform the same functions (hence "symmetric")	System controlled by integrated operating system • Provides interaction between processors and their programs at job, task, file and data element levels

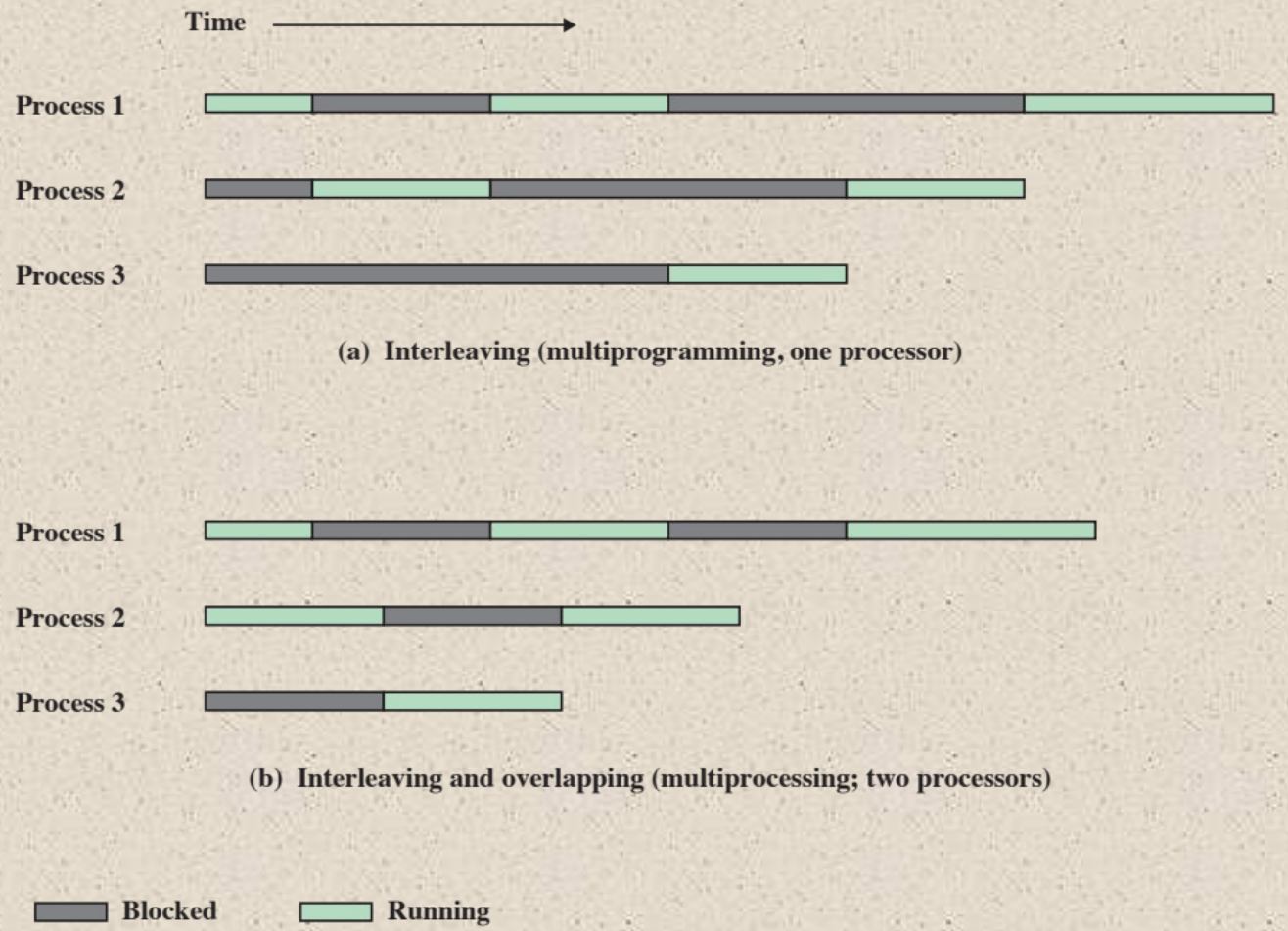


Figure 17.3 Multiprogramming and Multiprocessing

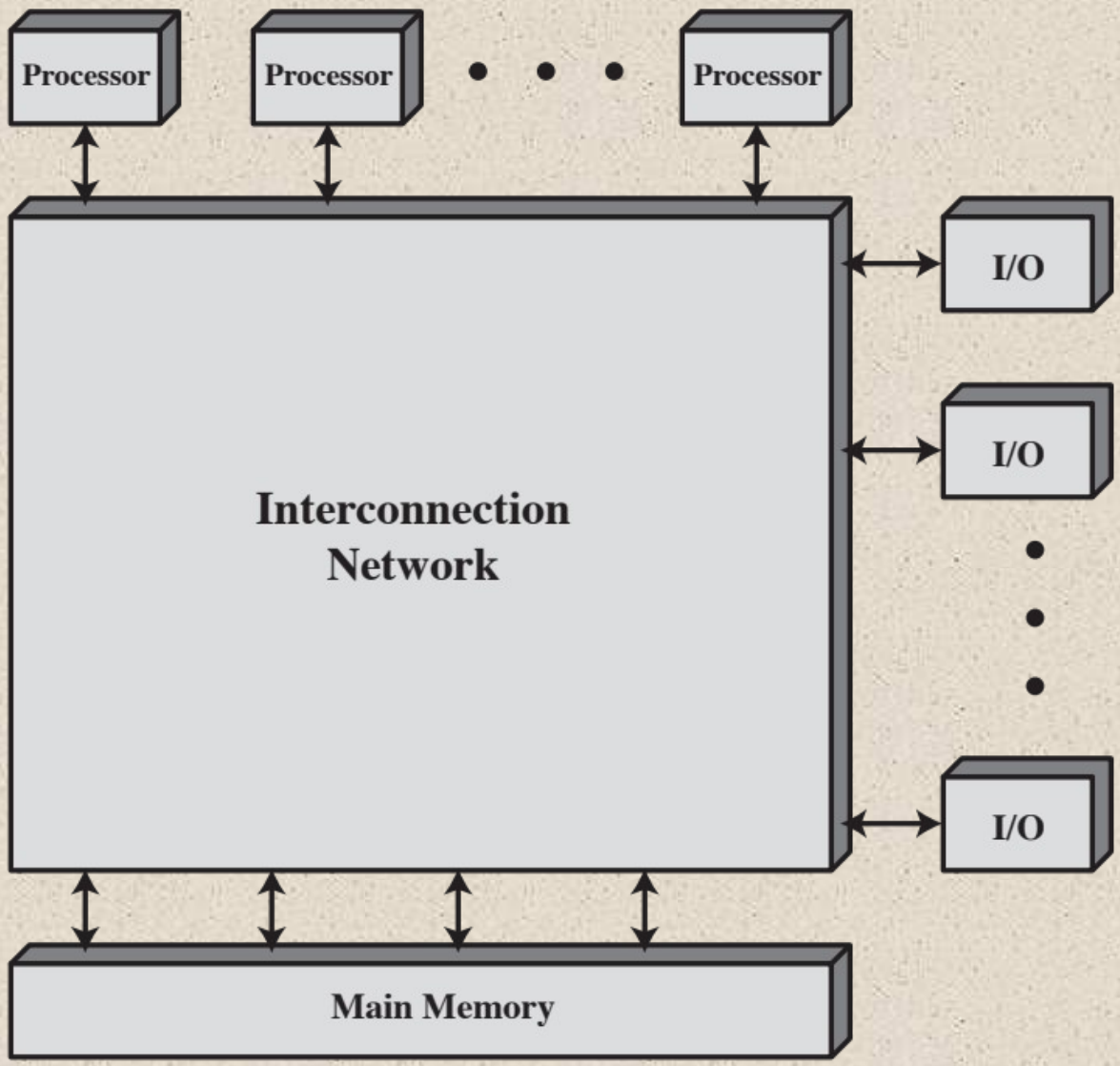


Figure 17.4 Generic Block Diagram of a Tightly Coupled Multiprocessor

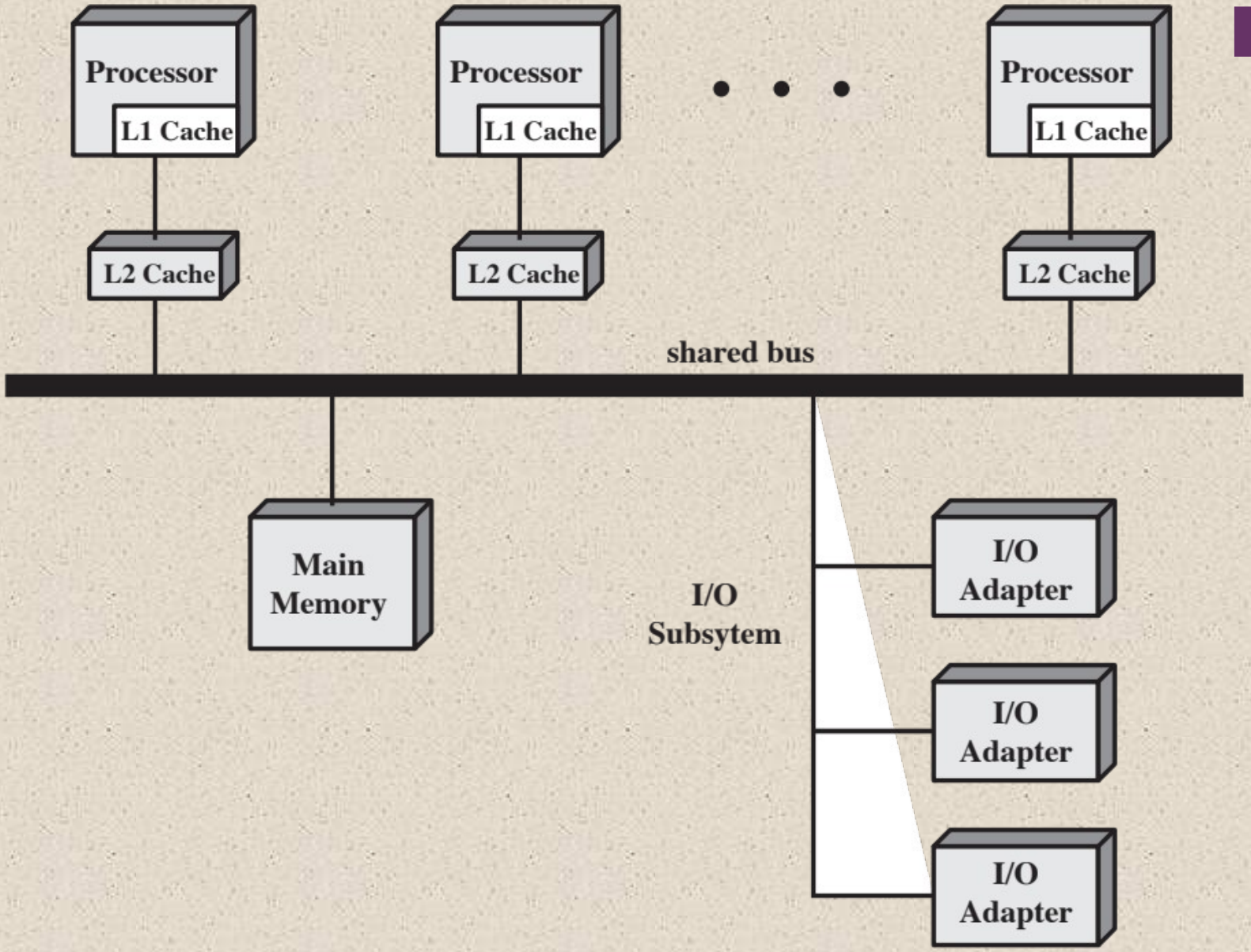
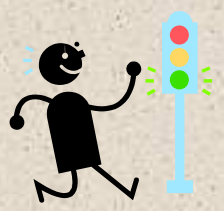


Figure 17.5 Symmetric Multiprocessor Organization

The bus organization has several attractive features:



- Simplicity
  - Simplest approach to multiprocessor organization
- Flexibility
  - Generally easy to expand the system by attaching more processors to the bus
- Reliability
  - The bus is essentially a passive medium and the failure of any attached device should not cause failure of the whole system

Disadvantages of the bus organization:

- Main drawback is performance
  - All memory references pass through the common bus
  - Performance is limited by bus cycle time
- Each processor should have cache memory
  - Reduces the number of bus accesses
- Leads to problems with *cache coherence*
  - If a word is altered in one cache it could conceivably invalidate a word in another cache
    - To prevent this the other processors must be alerted that an update has taken place
  - Typically addressed in hardware rather than the operating system

Multiprocessor Operating System Design Considerations

- **Simultaneous concurrent processes**
  - OS routines need to be reentrant to allow several processors to execute the same IS code simultaneously
  - OS tables and management structures must be managed properly to avoid deadlock or invalid operations
- **Scheduling**
  - Any processor may perform scheduling so conflicts must be avoided
  - Scheduler must assign ready processes to available processors
- **Synchronization**
  - With multiple active processes having potential access to shared address spaces or I/O resources, care must be taken to provide effective synchronization
  - Synchronization is a facility that enforces mutual exclusion and event ordering
- **Memory management**
  - In addition to dealing with all of the issues found on uniprocessor machines, the OS needs to exploit the available hardware parallelism to achieve the best performance
  - Paging mechanisms on different processors must be coordinated to enforce consistency when several processors share a page or segment and to decide on page replacement
- **Reliability and fault tolerance**
  - OS should provide graceful degradation in the face of processor failure
  - Scheduler and other portions of the operating system must recognize the loss of a processor and restructure accordingly

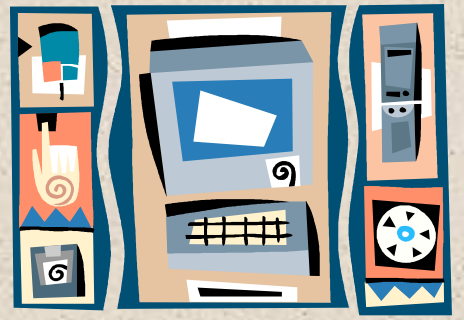
Cache Coherence

Software Solutions

- Attempt to avoid the need for additional hardware circuitry and logic by relying on the compiler and operating system to deal with the problem
- Attractive because the overhead of detecting potential problems is transferred from run time to compile time, and the design complexity is transferred from hardware to software
  - However, compile-time software approaches generally must make conservative decisions, leading to inefficient cache utilization

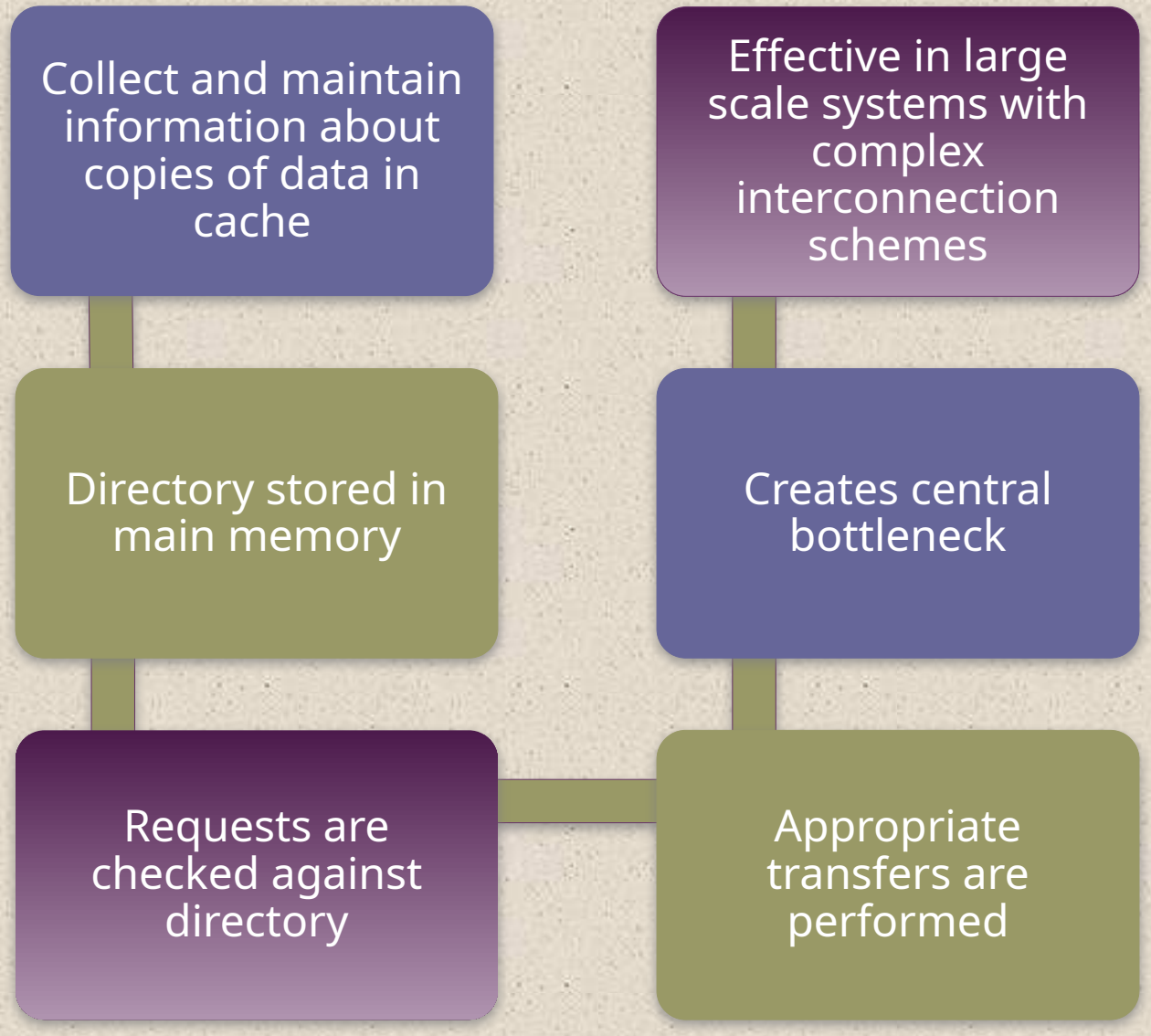
Cache Coherence

Hardware-Based Solutions



- Generally referred to as cache coherence protocols
- These solutions provide dynamic recognition at run time of potential inconsistency conditions
- Because the problem is only dealt with when it actually arises there is more effective use of caches, leading to improved performance over a software approach
- Approaches are transparent to the programmer and the compiler, reducing the software development burden
- Can be divided into two categories:
  - Directory protocols
  - Snoopy protocols

Directory Protocols



Snoopy Protocols

- Distribute the responsibility for maintaining cache coherence among all of the cache controllers in a multiprocessor
  - A cache must recognize when a line that it holds is shared with other caches
  - When updates are performed on a shared cache line, it must be announced to other caches by a broadcast mechanism
  - Each cache controller is able to “snoop” on the network to observe these broadcast notifications and react accordingly
- Suited to bus-based multiprocessor because the shared bus provides a simple means for broadcasting and snooping
  - Care must be taken that the increased bus traffic required for broadcasting and snooping does not cancel out the gains from the use of local caches
- Two basic approaches have been explored:
  - Write invalidate
  - Write update (or write broadcast)

Write Invalidate

- Multiple readers, but only one writer at a time
- When a write is required, all other caches of the line are invalidated
- Writing processor then has exclusive (cheap) access until line is required by another processor
- Most widely used in commercial multiprocessor systems such as the x86 architecture
- State of every line is marked as modified, exclusive, shared or invalid
  - For this reason the write-invalidate protocol is called *MESI*

Write Update

Can be multiple readers and writers

When a processor wishes to update a shared line the word to be updated is distributed to all others and caches containing that line can update it

Some systems use an adaptive mixture of both write-invalidate and write-update mechanisms

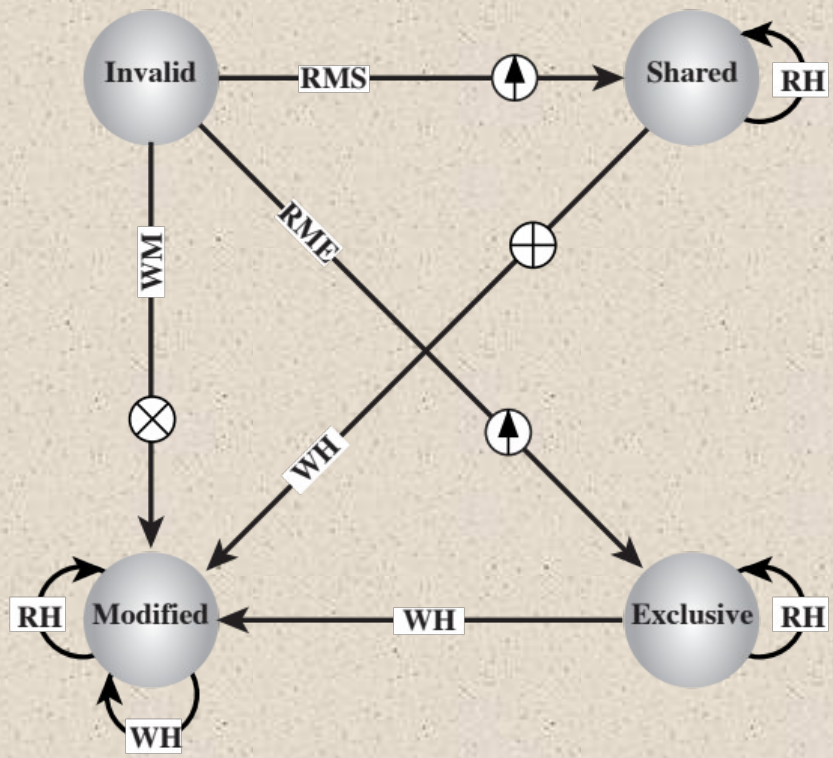
MESI Protocol

To provide cache consistency on an SMP the data cache supports a protocol known as MESI:

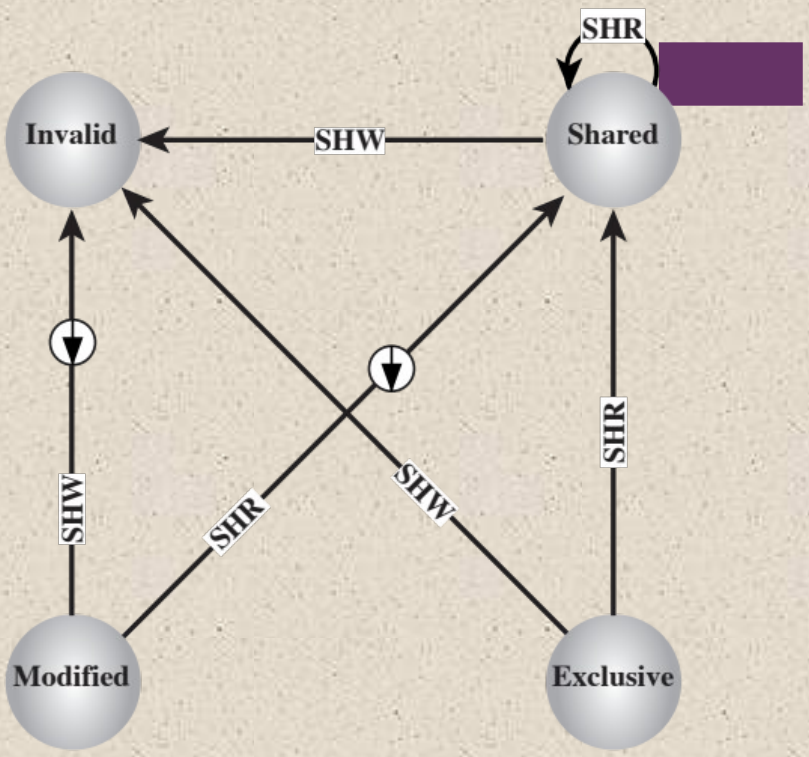
- Modified
  - The line in the cache has been modified and is available only in this cache
- Exclusive
  - The line in the cache is the same as that in main memory and is not present in any other cache
- Shared
  - The line in the cache is the same as that in main memory and may be present in another cache
- Invalid
  - The line in the cache does not contain valid data

Table 17.1
MESI Cache Line States

	M Modified	E Exclusive	S Shared	I Invalid
This cache line valid?	Yes	Yes	Yes	No
The memory copy is…	out of date	valid	valid	—
Copies exist in other caches?	No	No	Maybe	Maybe
A write to this line…	does not go to bus	does not go to bus	goes to bus and updates cache	goes directly to bus



(a) Line in cache at initiating processor



(b) Line in snooping cache

RH	Read hit
RMS	Read miss, shared
RME	Read miss, exclusive
WH	Write hit
WM	Write miss
SHR	Snoop hit on read
SHW	Snoop hit on write or read-with-intent-to-modify

Dirty line copyback

Invalidate transaction

Read-with-intent-to-modify

Cache line fill

Figure 17.6 MESI State Transition Diagram

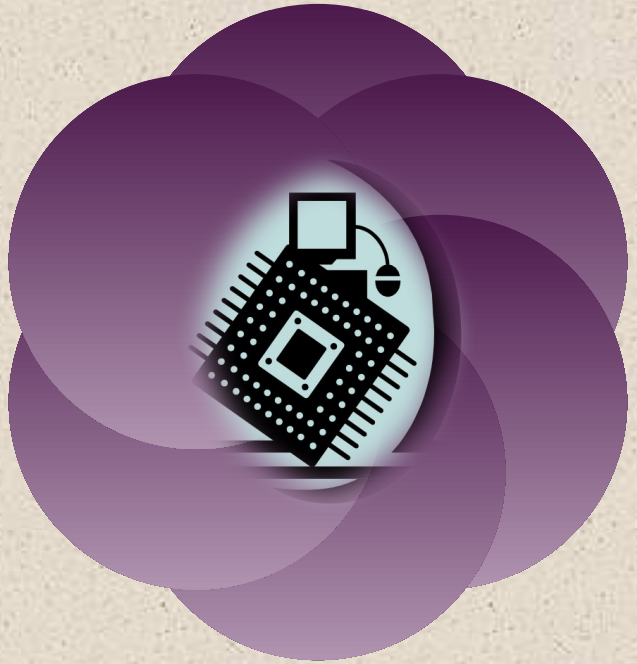
Multithreading and Chip Multiprocessors

- Processor performance can be measured by the rate at which it executes instructions
- MIPS rate = f * IPC
  - f = processor clock frequency, in MHz
  - IPC = average instructions per cycle
- Increase performance by increasing clock frequency and increasing instructions that complete during cycle
- Multithreading
  - Allows for a high degree of instruction-level parallelism without increasing circuit complexity or power consumption
  - Instruction stream is divided into several smaller streams, known as threads, that can be executed in parallel

Definitions of Threads and Processes

Thread in multithreaded processors may or may not be the same as the concept of software threads in a multiprogrammed operating system

Thread is concerned with scheduling and execution, whereas a process is concerned with both scheduling/execution and resource and resource ownership



Thread switch

- The act of switching processor control between threads within the same process
- Typically less costly than process switch

Thread:

- Dispatchable unit of work within a process
- Includes processor context (which includes the program counter and stack pointer) and data area for stack
- Executes sequentially and is interruptible so that the processor can turn to another thread

Process:

- An instance of program running on computer
- Two key characteristics:
  - Resource ownership
  - Scheduling/execution

Process switch

- Operation that switches the processor from one process to another by saving all the process control data, registers, and other information for the first and replacing them with the process information for the second

Implicit and Explicit Multithreading

- All commercial processors and most experimental ones use explicit multithreading
  - Concurrently execute instructions from different explicit threads
  - Interleave instructions from different threads on shared pipelines or parallel execution on parallel pipelines
- Implicit multithreading is concurrent execution of multiple threads extracted from single sequential program
  - Implicit threads defined statically by compiler or dynamically by hardware

Approaches to Explicit Multithreading

- Interleaved
  - Fine-grained
  - Processor deals with two or more thread contexts at a time
  - Switching thread at each clock cycle
  - If thread is blocked it is skipped
- Simultaneous (SMT)
  - Instructions are simultaneously issued from multiple threads to execution units of superscalar processor
- Blocked
  - Coarse-grained
  - Thread executed until event causes delay
  - Effective on in-order processor
  - Avoids pipeline stall
- Chip multiprocessing
  - Processor is replicated on a single chip
  - Each processor handles separate threads
  - Advantage is that the available logic area on a chip is used effectively

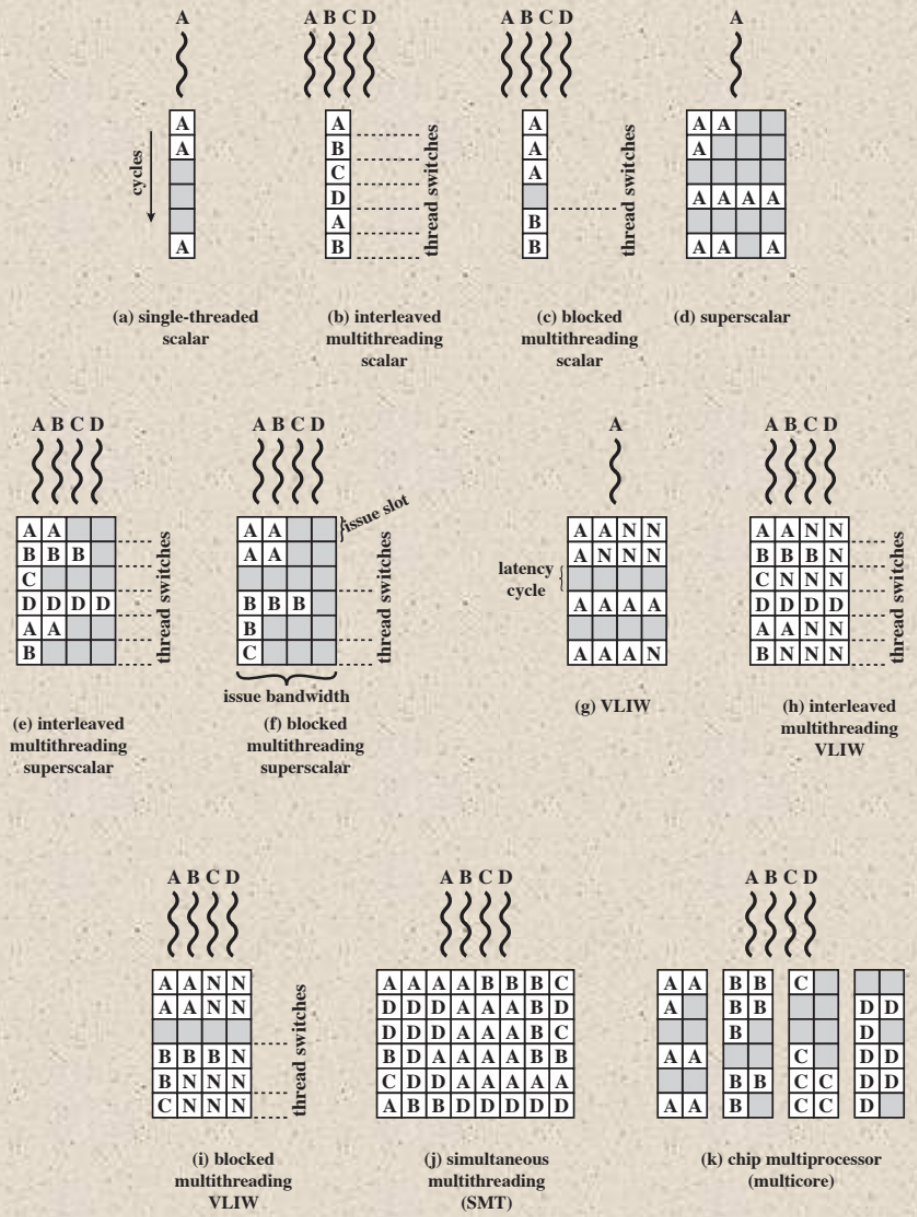
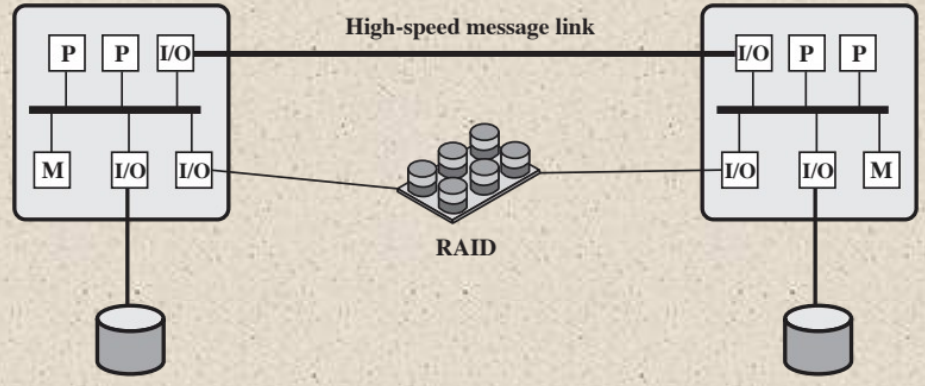


Figure 17.7 Approaches to Executing Multiple Threads

Clusters

- Alternative to SMP as an approach to providing high performance and high availability
- Particularly attractive for server applications
- Defined as:
  - A group of interconnected whole computers working together as a unified computing resource that can create the illusion of being one machine
  - (The term *whole computer* means a system that can run on its own, apart from the cluster)
- Each computer in a cluster is called a node
- Benefits:
  - Absolute scalability
  - Incremental scalability
  - High availability
  - Superior price/performance

(a) Standby server with no shared disk



(b) Shared disk

Figure 17.8 Cluster Configurations

Table 17.2
Clustering Methods: Benefits and Limitations

Clustering Method	Description	Benefits	Limitations
Passive Standby	A secondary server takes over in case of primary server failure.	Easy to implement.	High cost because the secondary server is unavailable for other processing tasks.
Active Secondary:	The secondary server is also used for processing tasks.	Reduced cost because secondary servers can be used for processing.	Increased complexity.
Separate Servers	Separate servers have their own disks. Data is continuously copied from primary to secondary server.	High availability.	High network and server overhead due to copying operations.
Servers Connected to Disks	Servers are cabled to the same disks, but each server owns its disks. If one server fails, its disks are taken over by the other server.	Reduced network and server overhead due to elimination of copying operations.	Usually requires disk mirroring or RAID technology to compensate for risk of disk failure.
Servers Share Disks	Multiple servers simultaneously share access to disks.	Low network and server overhead. Reduced risk of downtime caused by disk failure.	Requires lock manager software. Usually used with disk mirroring or RAID technology.

Operating System Design Issues

- How failures are managed depends on the clustering method used
- Two approaches:
  - Highly available clusters
  - Fault tolerant clusters
- Failover
  - The function of switching applications and data resources over from a failed system to an alternative system in the cluster
- Failback
  - Restoration of applications and data resources to the original system once it has been fixed
- Load balancing
  - Incremental scalability
  - Automatically include new computers in scheduling
  - Middleware needs to recognize that processes may switch between machines

Parallelizing Computation

Effective use of a cluster requires executing software from a single application in parallel

Three approaches are:

Parallelizing complier

- Determines at compile time which parts of an application can be executed in parallel
- These are then split off to be assigned to different computers in the cluster

Parallelized application

- Application written from the outset to run on a cluster and uses message passing to move data between cluster nodes

Parametric computing

- Can be used if the essence of the application is an algorithm or program that must be executed a large number of times, each time with a different set of starting conditions or parameters

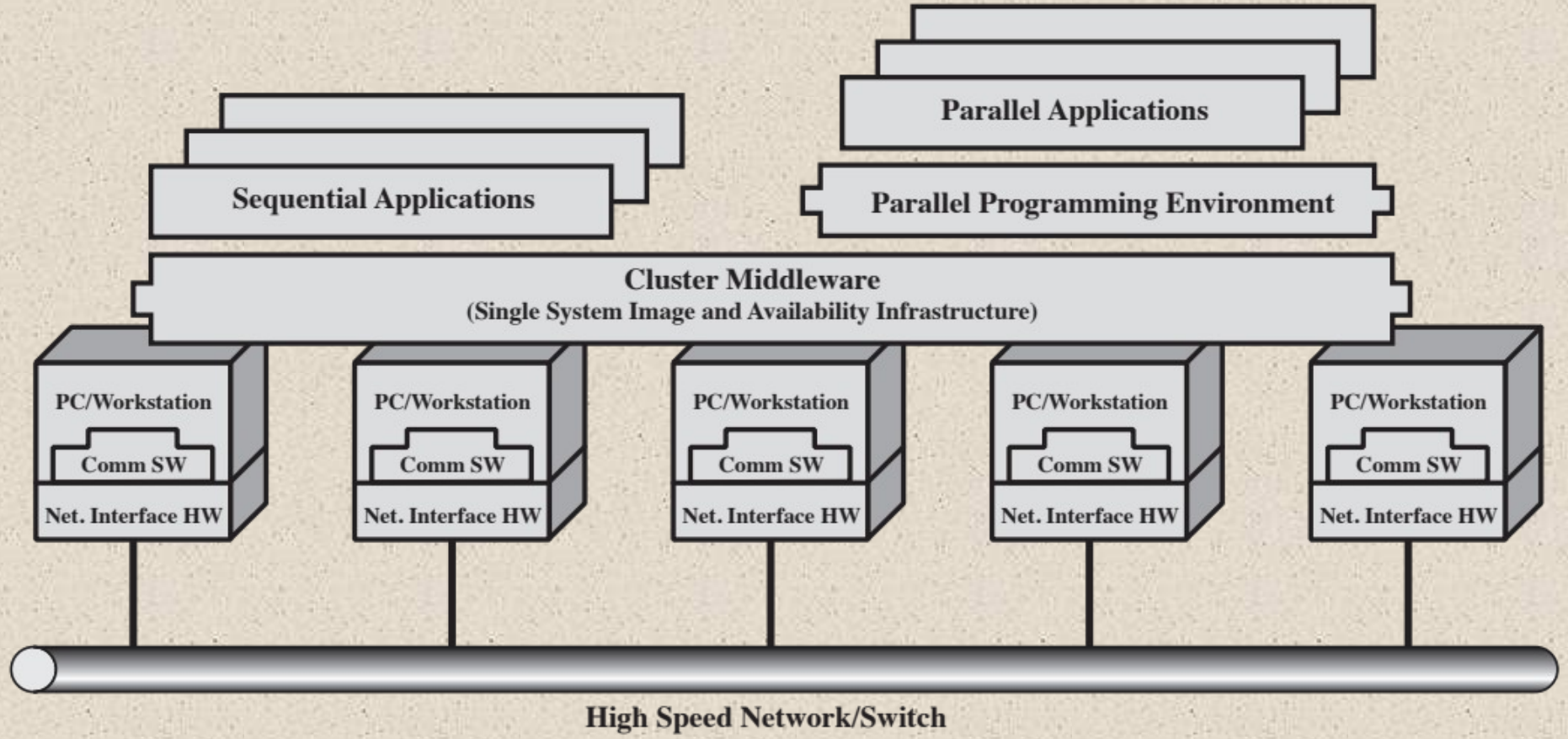


Figure 17.9 Cluster Computer Architecture

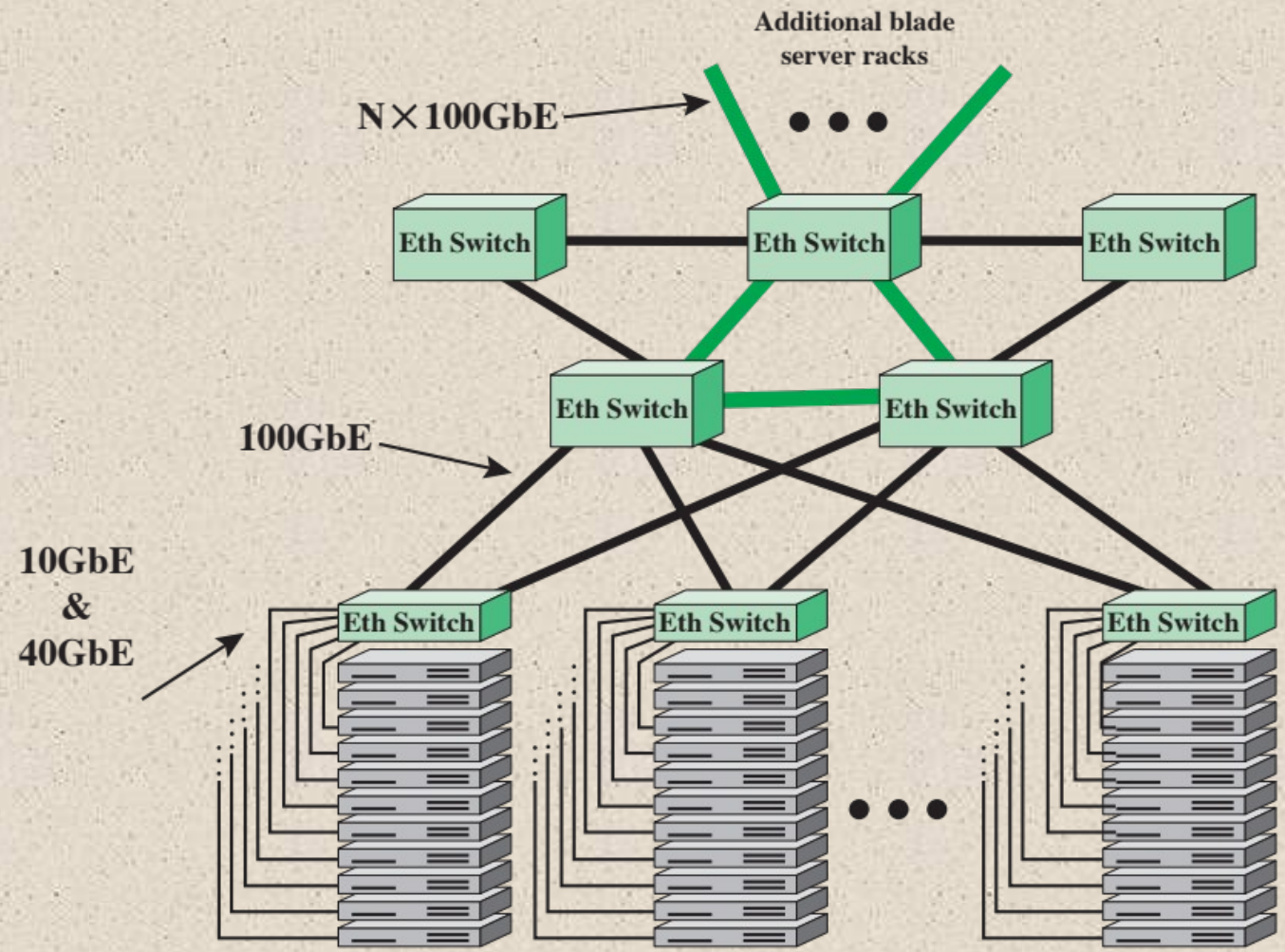


Figure 17.10 Example 100-Gbps Ethernet Configuration for Massive Blade Server Cloud Site

Clusters Compared to SMP

- Both provide a configuration with multiple processors to support high demand applications
- Both solutions are available commercially

SMP	Clustering
Easier to manage and configure	Far superior in terms of incremental and absolute scalability
Much closer to the original single processor model for which nearly all applications are written	Superior in terms of availability
Less physical space and lower power consumption	All components of the system can readily be made highly redundant
Well established and stable	

Nonuniform Memory Access (NUMA)

- Alternative to SMP and clustering
- Uniform memory access (UMA)
  - All processors have access to all parts of main memory using loads and stores
  - Access time to all regions of memory is the same
  - Access time to memory for different processors is the same
- Nonuniform memory access (NUMA)
  - All processors have access to all parts of main memory using loads and stores
  - Access time of processor differs depending on which region of main memory is being accessed
  - Different processors access different regions of memory at different speeds
- Cache-coherent NUMA (CC-NUMA)
  - A NUMA system in which cache coherence is maintained among the caches of the various processors

Motivation

SMP has practical limit to number of processors that can be used

- Bus traffic limits to between 16 and 64 processors

In clusters each node has its own private main memory

- Applications do not see a large global memory
- Coherency is maintained by software rather than hardware

NUMA retains SMP flavor while giving large scale multiprocessing

Objective with NUMA is to maintain a transparent system wide memory while permitting multiple multiprocessor nodes, each with its own bus or internal interconnect system

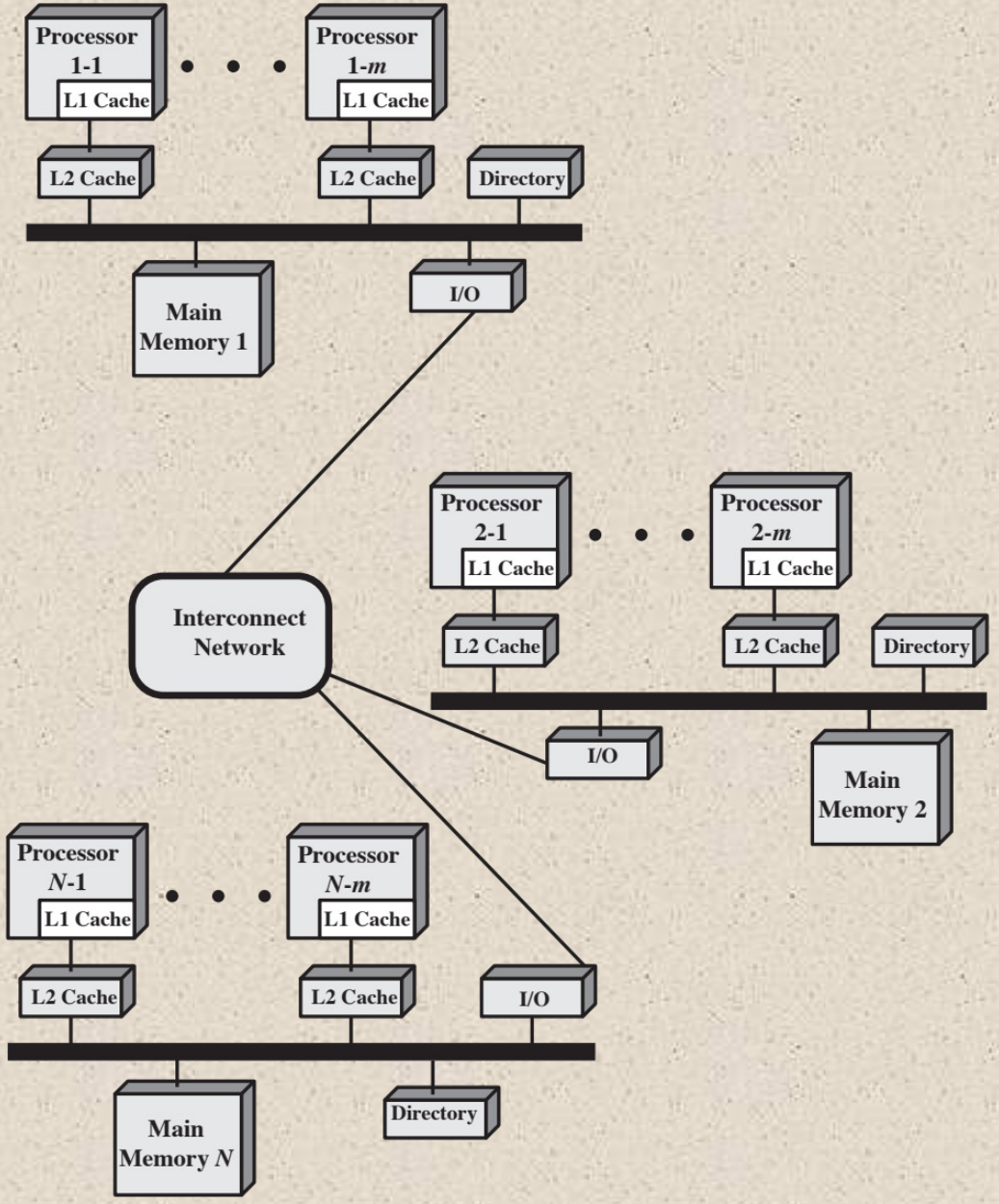


Figure 17.11 CC-NUMA Organization

NUMA Pros and Cons

- Main advantage of a CC-NUMA system is that it can deliver effective performance at higher levels of parallelism than SMP without requiring major software changes
- Bus traffic on any individual node is limited to a demand that the bus can handle
- If many of the memory accesses are to remote nodes, performance begins to break down
- Does not transparently look like an SMP
- Software changes will be required to move an operating system and applications from an SMP to a CC-NUMA system
- Concern with availability

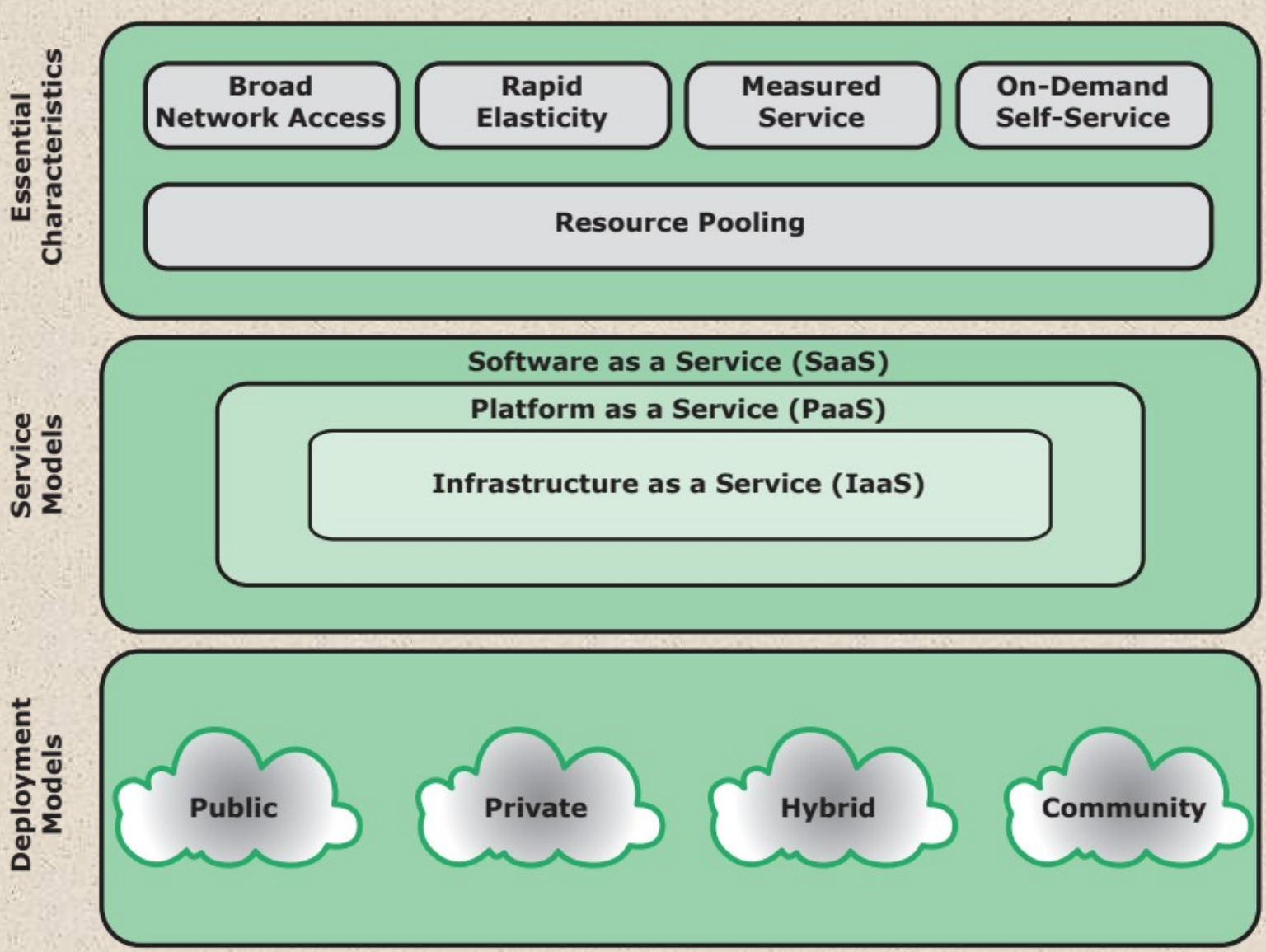


Figure 17.12 Cloud Computing Elements

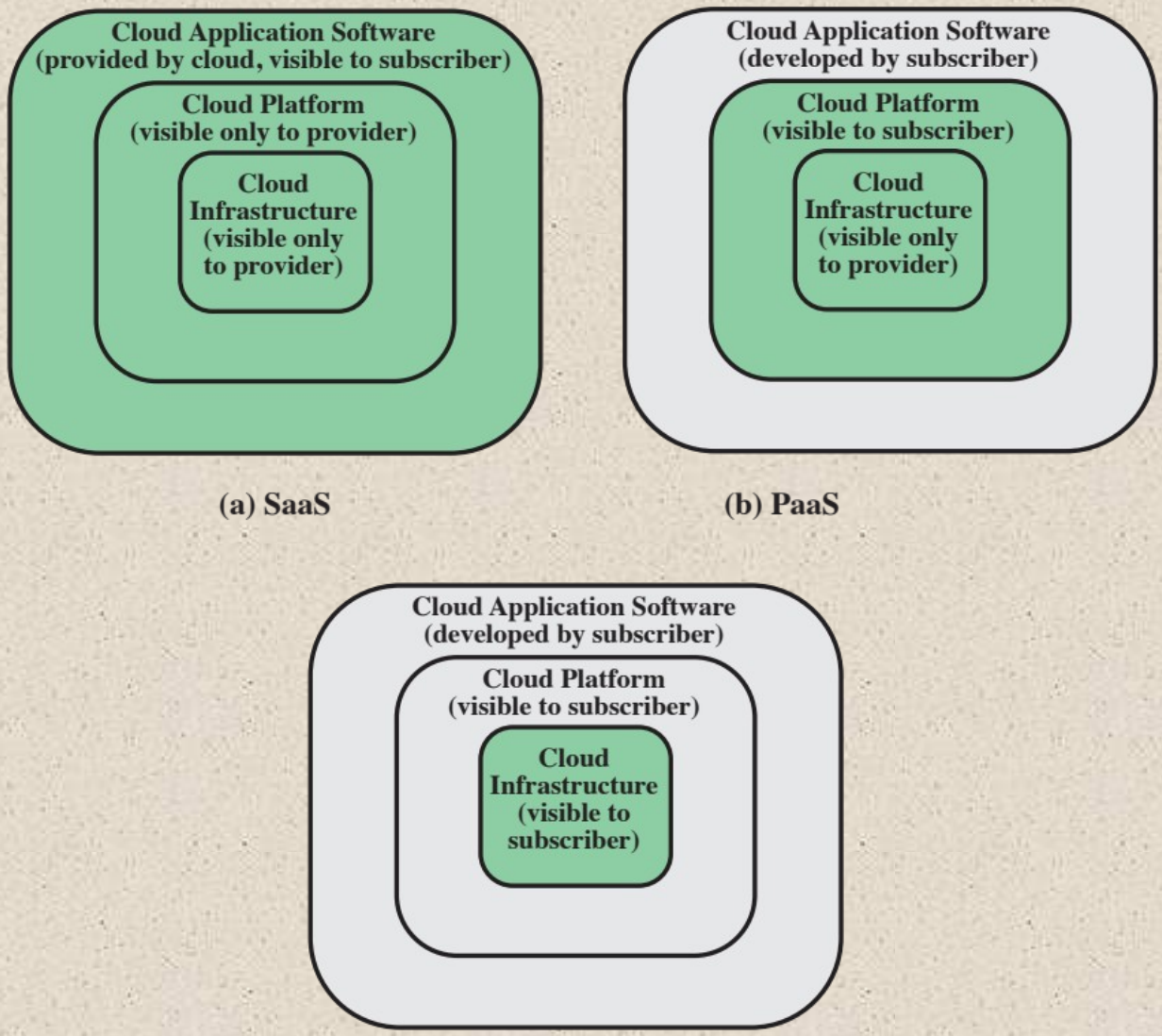


Figure 17.13 Cloud Service Models

Deployment Models

- Public cloud
  - The cloud infrastructure is made available to the general public or a large industry group and is owned by an organization selling cloud services
  - Major advantage is cost
- Private cloud
  - A cloud infrastructure implemented within the internal IT environment of the organization
  - A key motivation for opting for a private cloud is security
- Community cloud
  - Like a private cloud it is not open to any subscriber
  - Like a public cloud the resources are shared among a number of independent orgaizations
- Hybrid cloud
  - The cloud infrastructure is a composition of two or more clouds that remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability
  - Sensitive information can be placed in a private area of the cloud and less sensitive data can take advantage of the cost benefits of the public cloud

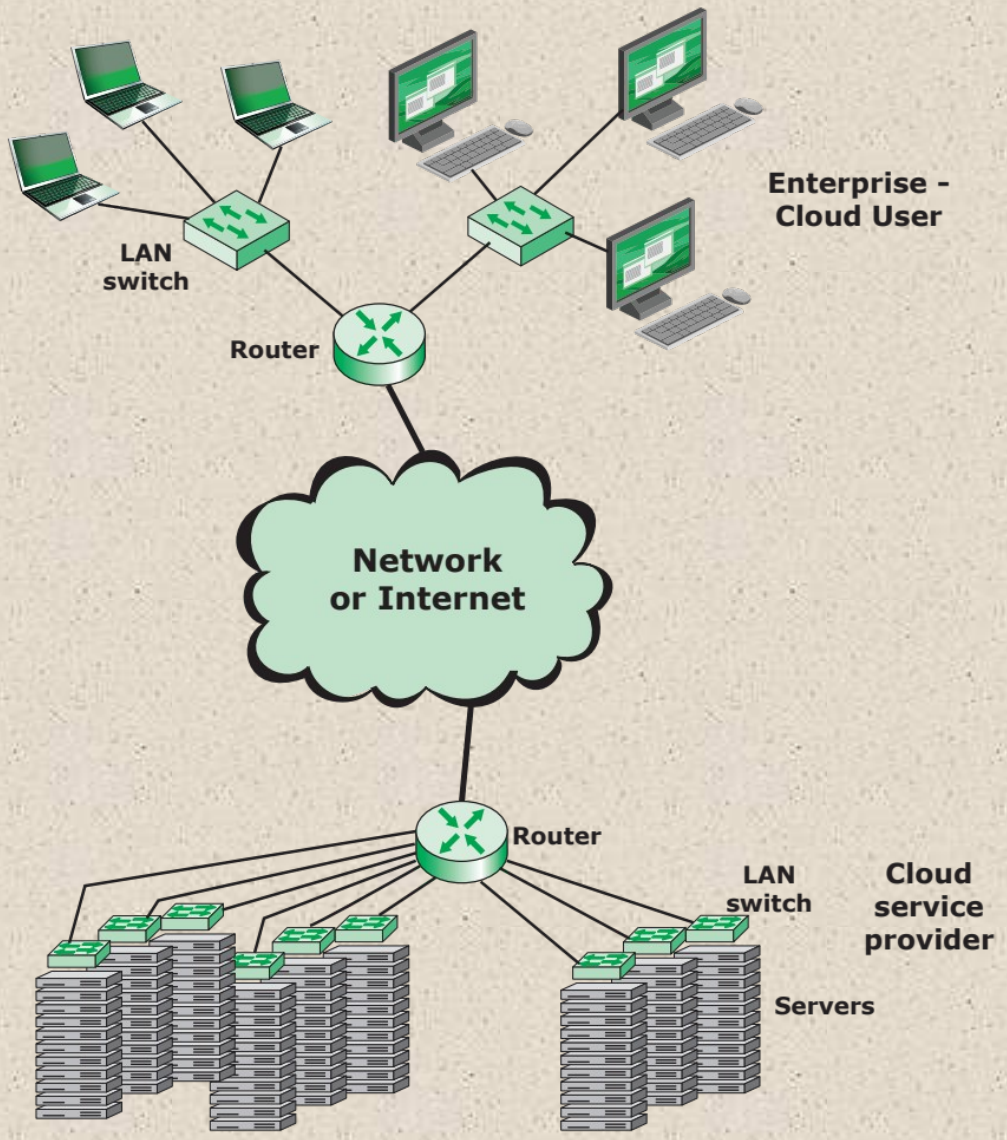


Figure 17.14 Cloud Computing Context

Cloud Computing Reference Architecture

- NIST SP 500-292 establishes a reference architecture, described as:

"The NIST cloud computing reference architecture focuses on the requirements of "what" cloud services provide, not a "how to" design solution and implementation. The reference architecture is intended to facilitate the understanding of the operational intricacies in cloud computing. It does not represent the system architecture of a specific cloud computing system; instead it is a tool for describing, discussing, and developing a system-specific architecture using a common framework of reference."

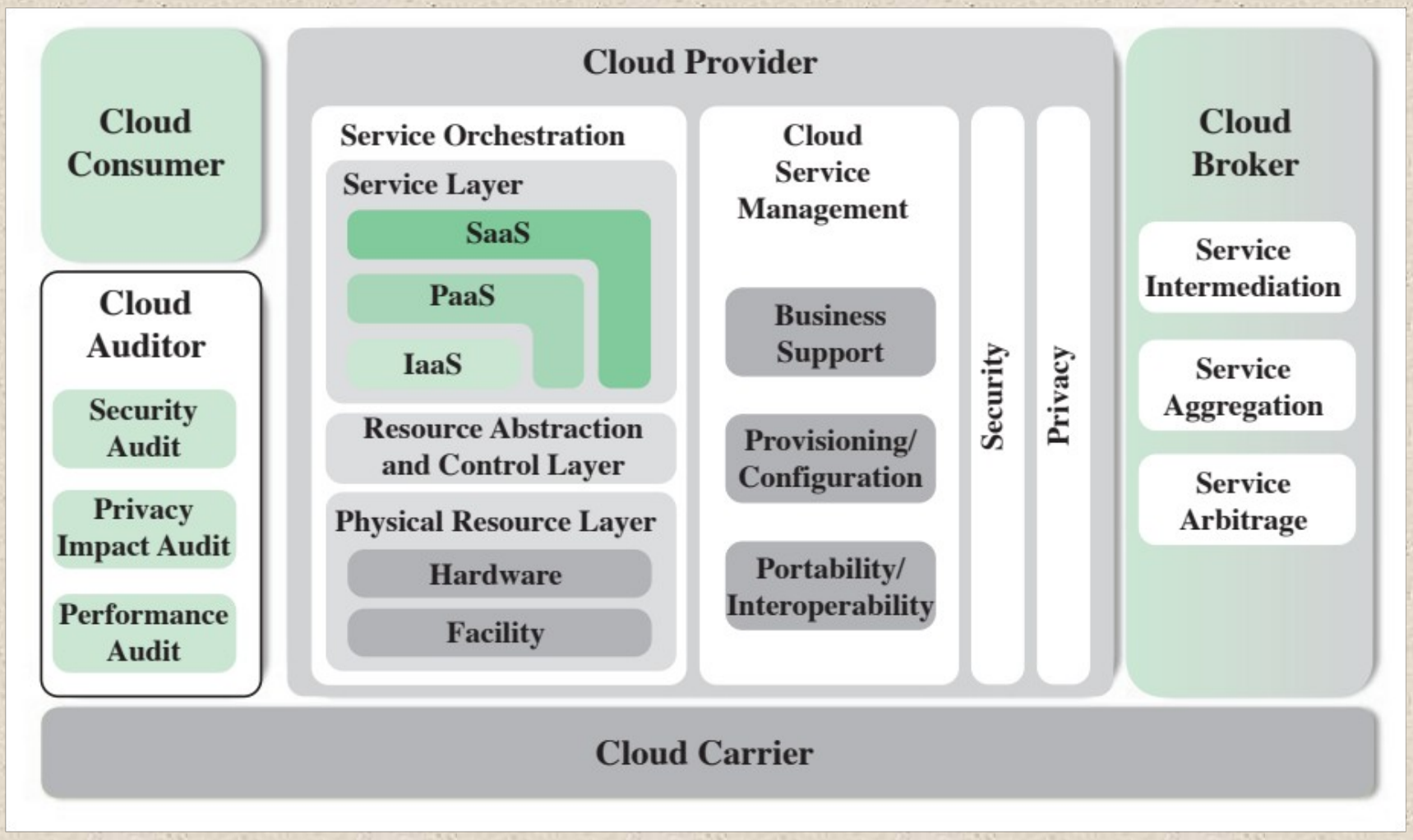


Figure 17.15 NIST Cloud Computing Reference Architecture

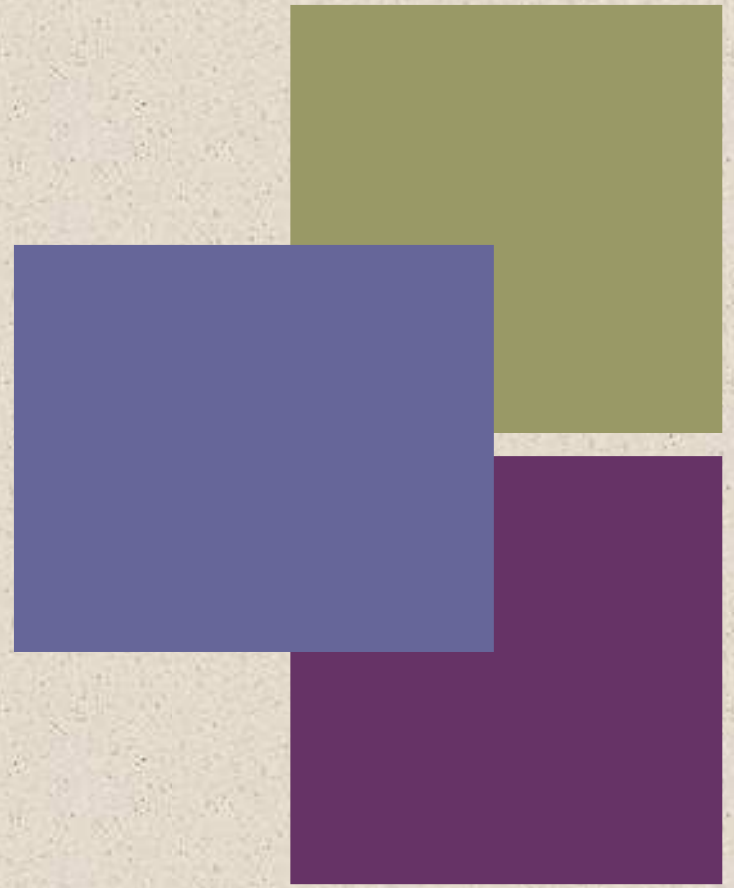
Summary

Chapter 17

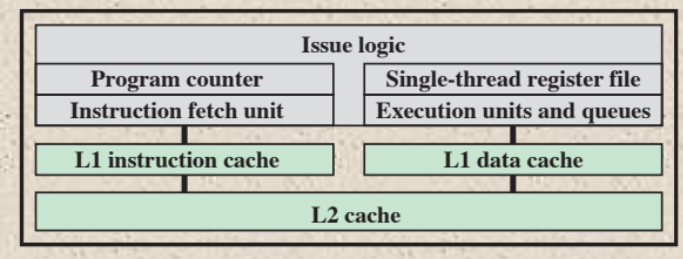
Parallel Processing

- Multiple processor organizations
  - Types of parallel processor systems
  - Parallel organizations
- Symmetric multiprocessors
  - Organization
  - Multiprocessor operating system design considerations
- Cache coherence and the MESI protocol
  - Software solutions
  - Hardware solutions
  - The MESI protocol
- Multithreading and chip multiprocessors
  - Implicit and explicit multithreading
  - Approaches to explicit multithreading
- Clusters
  - Cluster configurations
  - Operating system design issues
  - Cluster computer architecture
  - Blade servers
  - Clusters compared to SMP
- Nonuniform memory access
  - Motivation
  - Organization
  - NUMA Pros and cons
- Cloud computing
  - Cloud computing elements
  - Cloud computing reference architecture

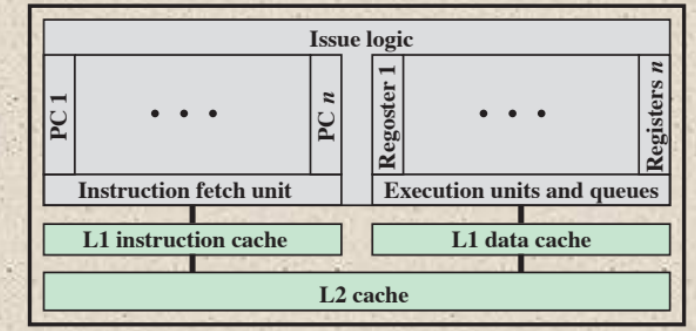
William Stallings
Computer Organization and Architecture
10th Edition



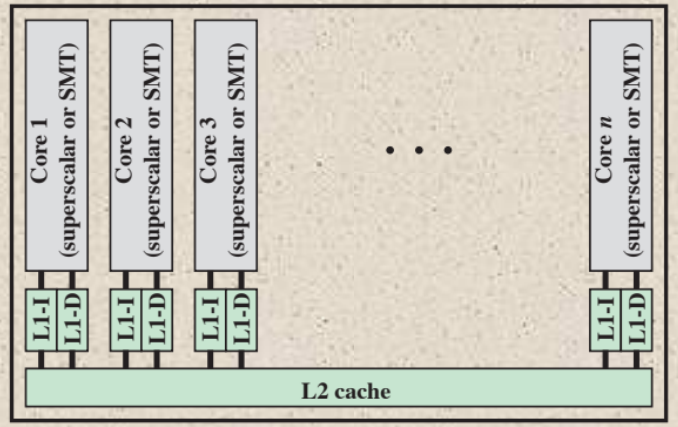
Chapter 18
Multicore Computers



(a) Superscalar



(b) Simultaneous multithreading



(c) Multicore

Figure 18.1 Alternative Chip Organizations

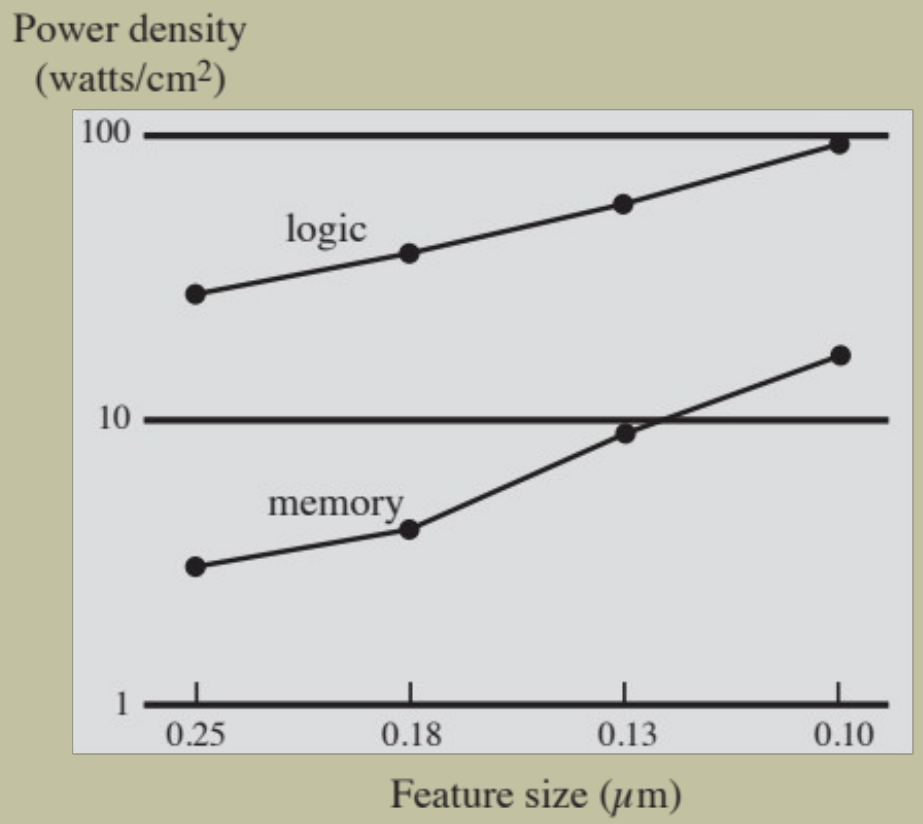
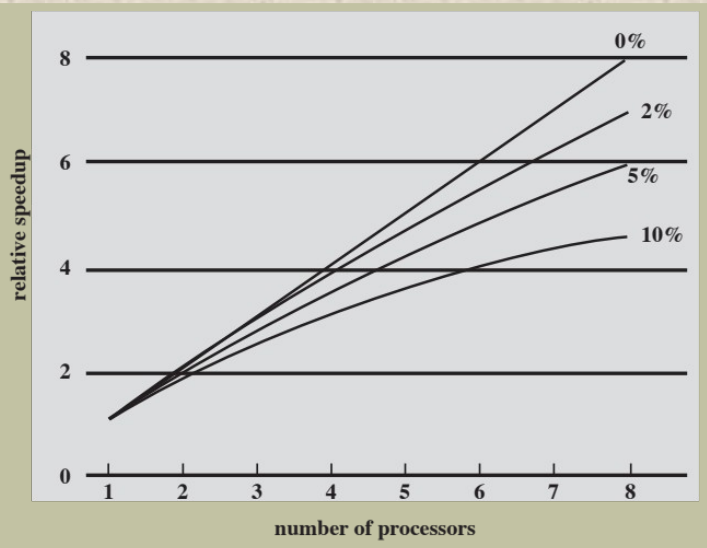


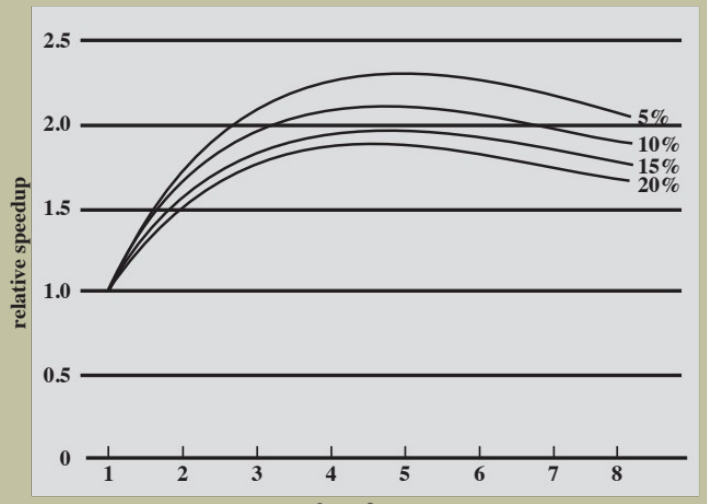
Figure 18.2 Power and Memory Considerations

Powe
r

Memor
y



(a) Speedup with 0%, 2%, 5%, and 10% sequential portions



(b) Speedup with overheads

Figure 18.3 Performance Effect of Multiple Cores

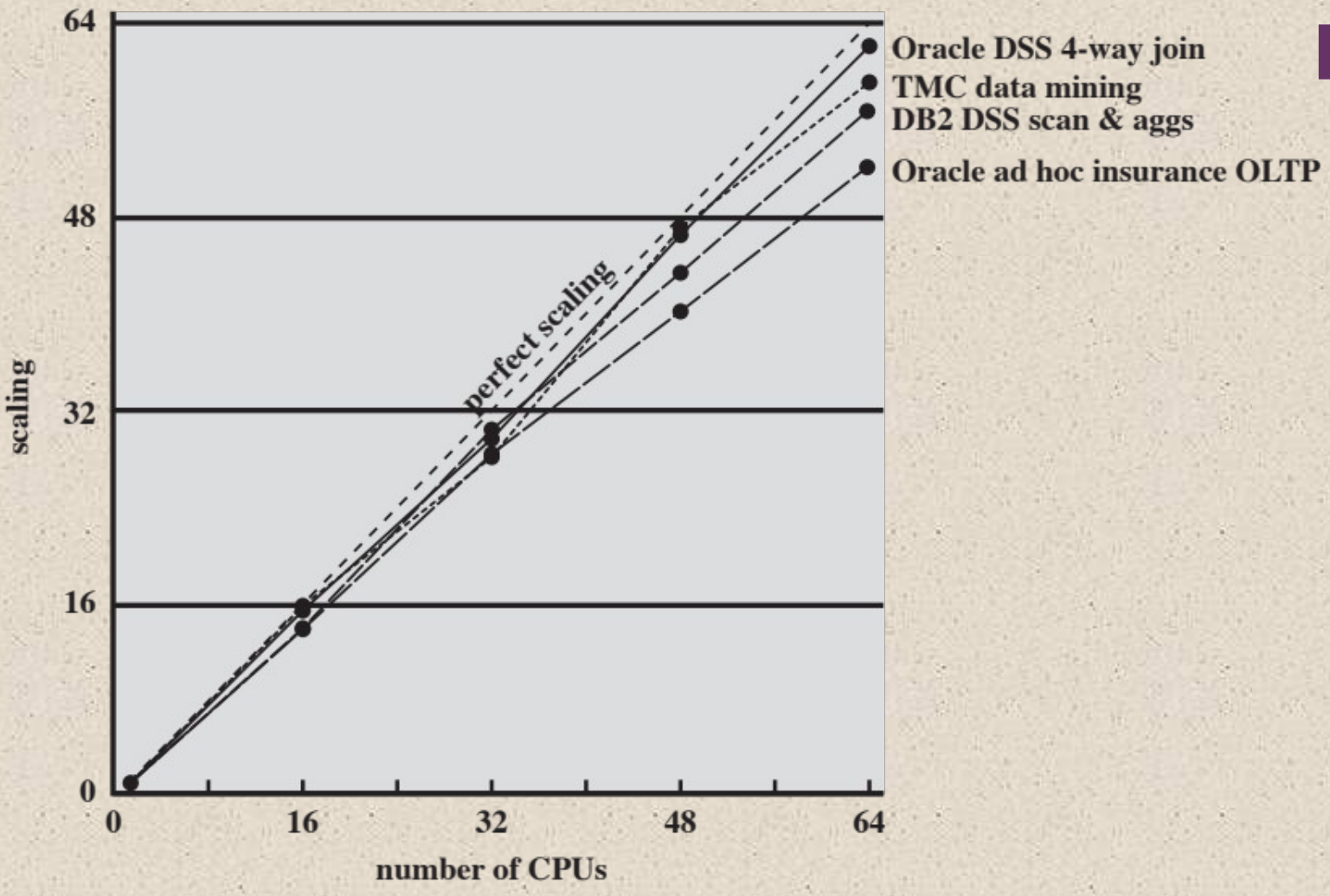


Figure 18.4 Scaling of Database Workloads on Multiple-Processor Hardware

Effective Applications for Multicore Processors

- **Multi-threaded native applications**
  - Thread-level parallelism
  - Characterized by having a small number of highly threaded processes
- **Multi-process applications**
  - Process-level parallelism
  - Characterized by the presence of many single-threaded processes
- **Java applications**
  - Embrace threading in a fundamental way
  - Java Virtual Machine is a multi-threaded process that provides scheduling and memory management for Java applications
- **Multi-instance applications**
  - If multiple application instances require some degree of isolation, virtualization technology can be used to provide each of them with its own separate and secure environment

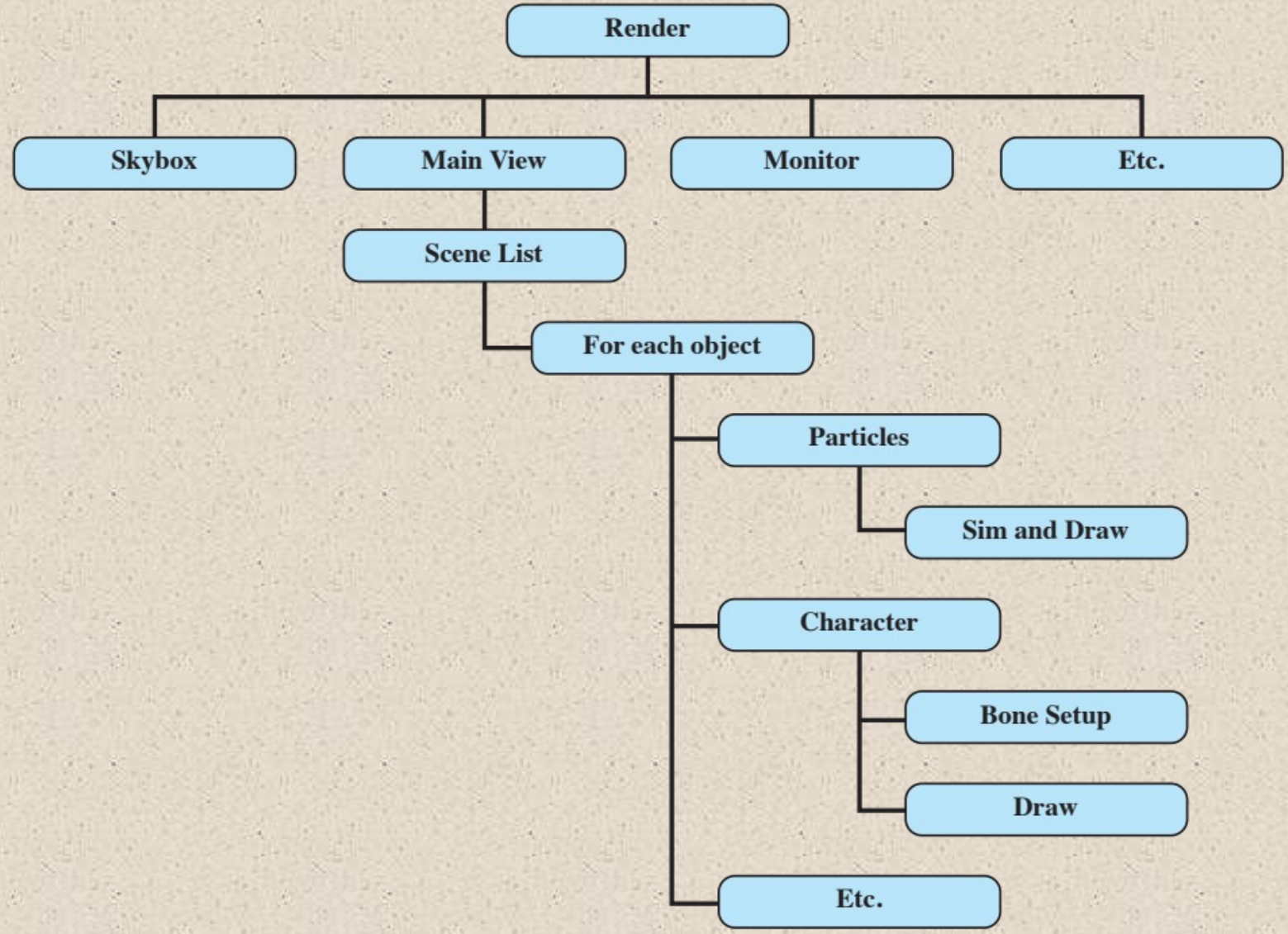


Figure 18.5 Hybrid Threading for Rendering Module

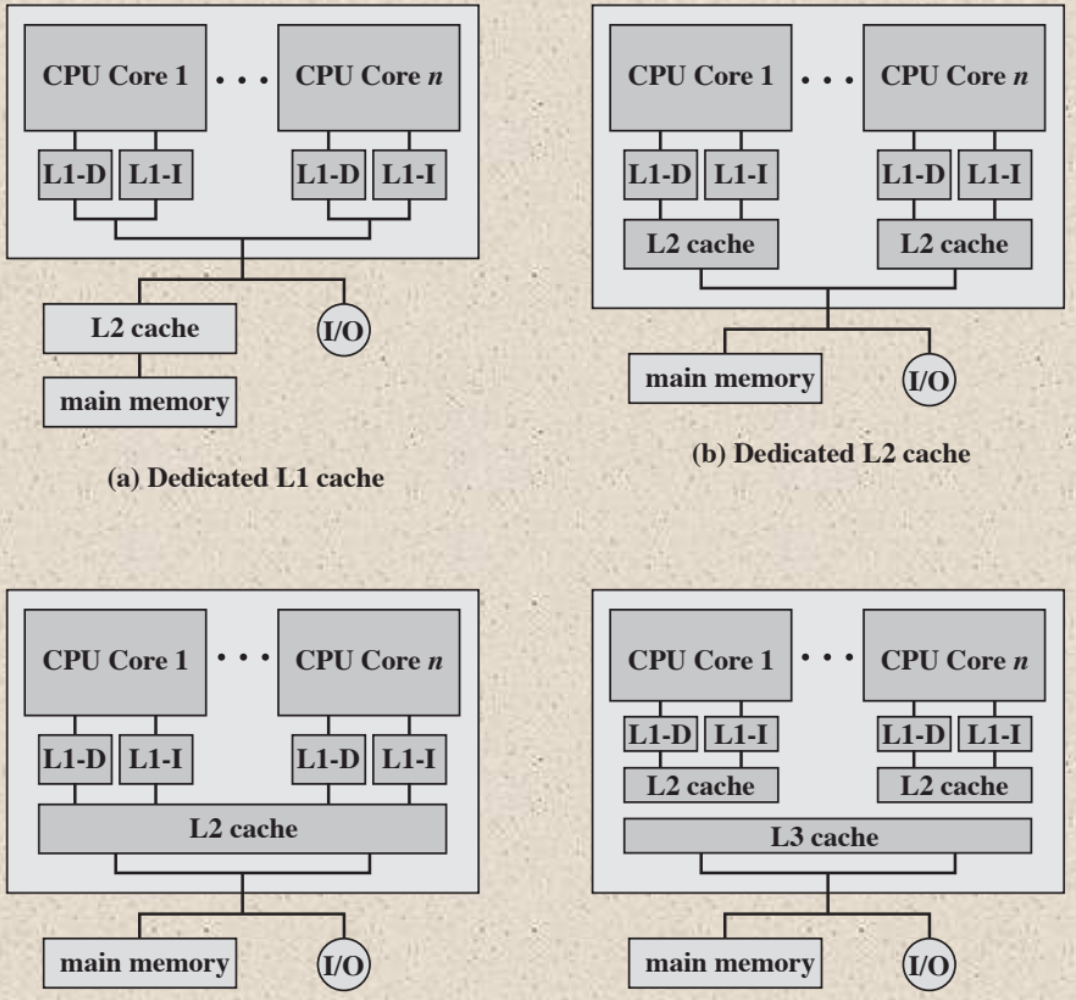


Figure 18.6 Multicore Organization Alternatives

Heterogeneous Multicore Organization

Refers to a processor chip that includes more than one kind of core

The most prominent trend is the use of both CPUs and graphics processing units (GPUs) on the same chip

- This mix however presents issues of coordination and correctness

GPUs are characterized by the ability to support thousands of parallel execution trends

Thus, GPUs are well matched to applications that process large amounts of vector and matrix data

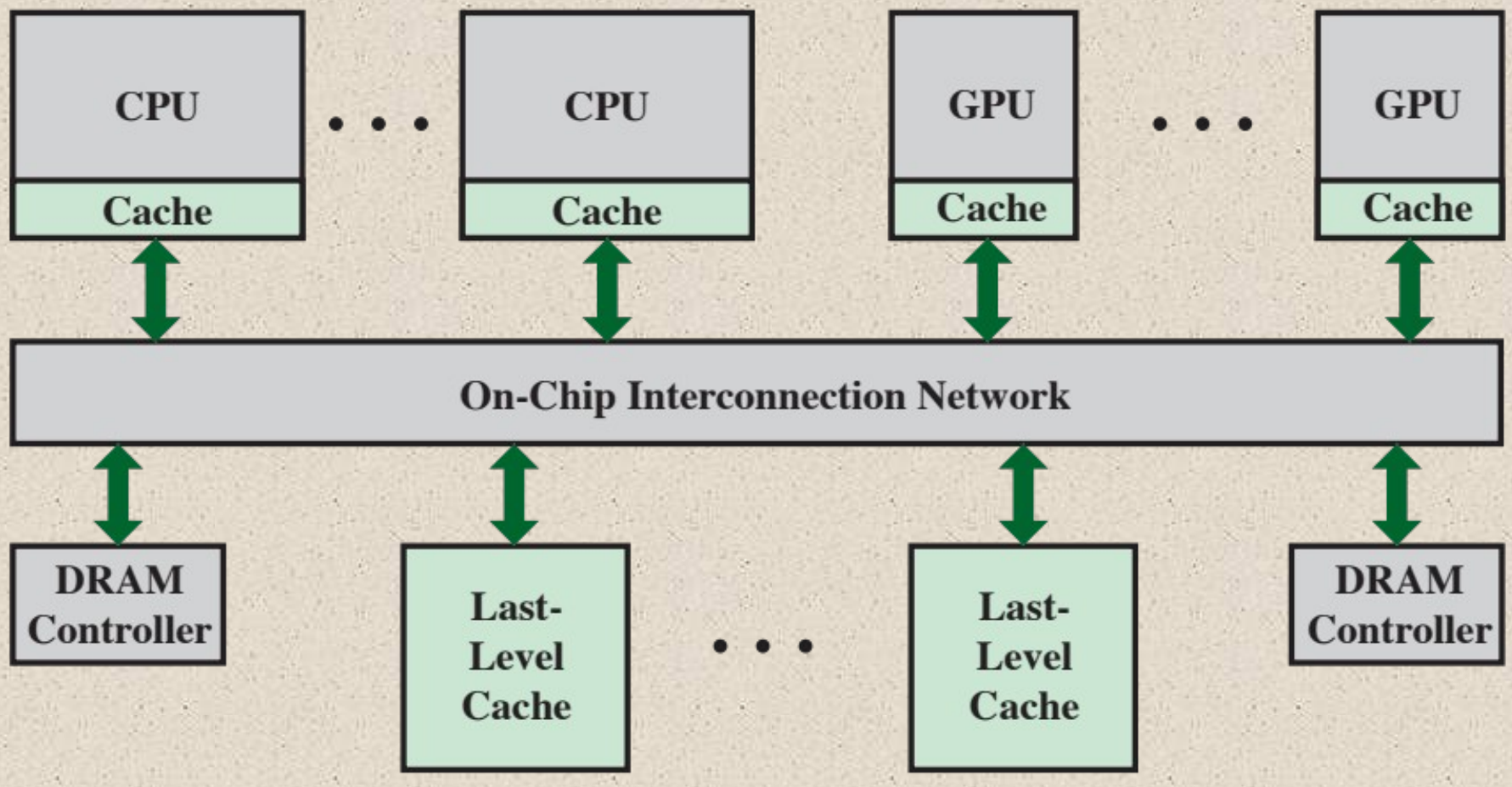


Figure 18.7 Heterogenous Multicore Chip Elements

Table 18.1

Operating Parameters of AMD 5100K Heterogeneous Multicore Processor

	CPU	GPU
Clock frequency (GHz)	3.8	0.8
Cores	4	384
FLOPS/core	8	2
GFLOPS	121.6	614.4

FLOPS = floating point operations per second
FLOPS/core = number of parallel floating point operations that can be performed

Heterogeneous System Architecture (HSA)

- Key features of the HSA approach include:
  - The entire virtual memory space is visible to both CPU and GPU
  - The virtual memory system brings in pages to physical main memory as needed
  - A coherent memory policy ensures that CPU and GPU caches both see an up-to-date view of data
  - A unified programming interface that enables users to exploit the parallel capabilities of the GPUs within programs that rely on CPU execution as well
- The overall objective is to allow programmers to write applications that exploit the serial power of CPUs and the parallel-processing power of GPUs seamlessly with efficient coordination at the OS and hardware level

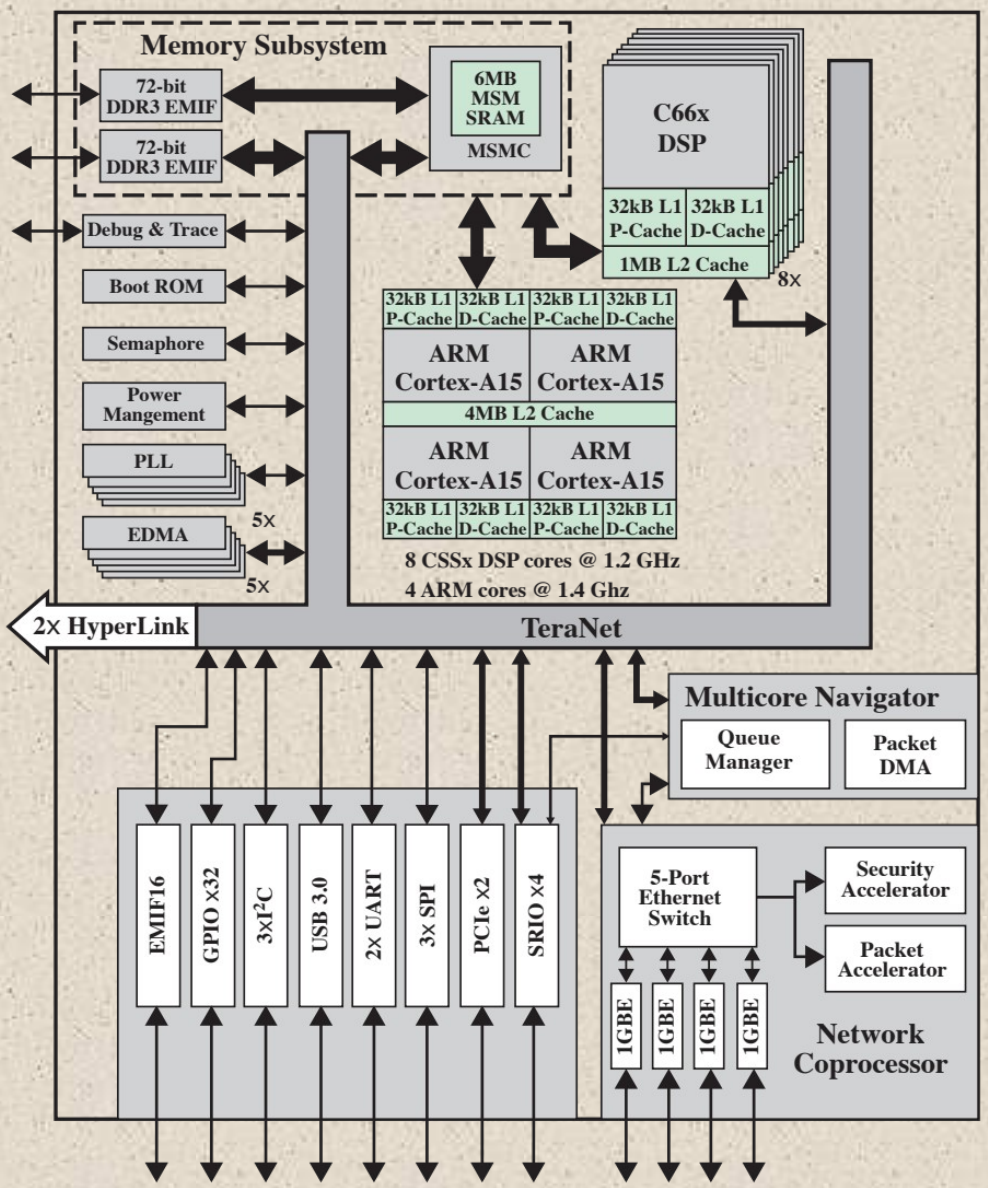


Figure 18.8 Texas Instruments 66AK2H12 Heterogenous Multicore Chip

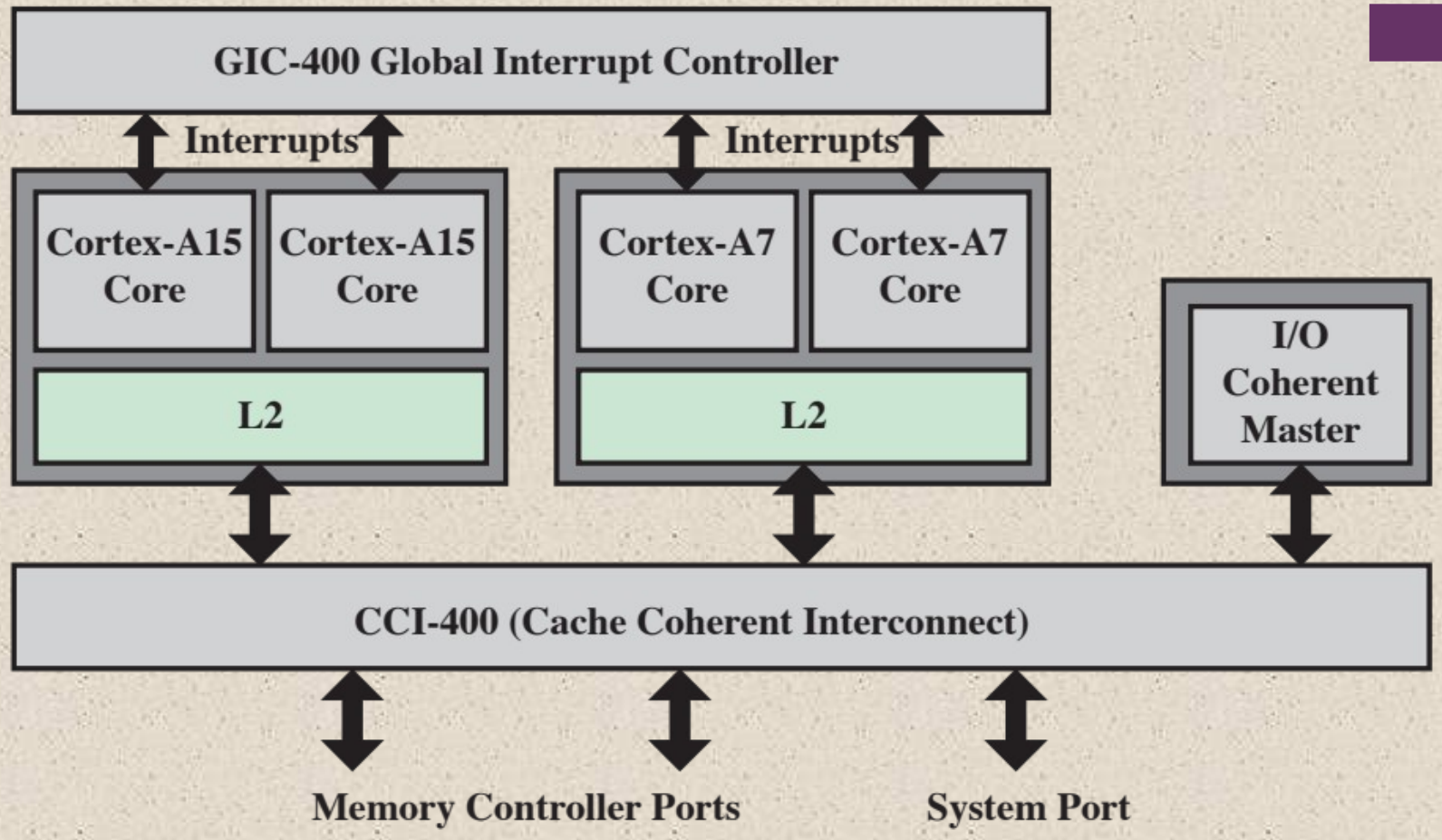
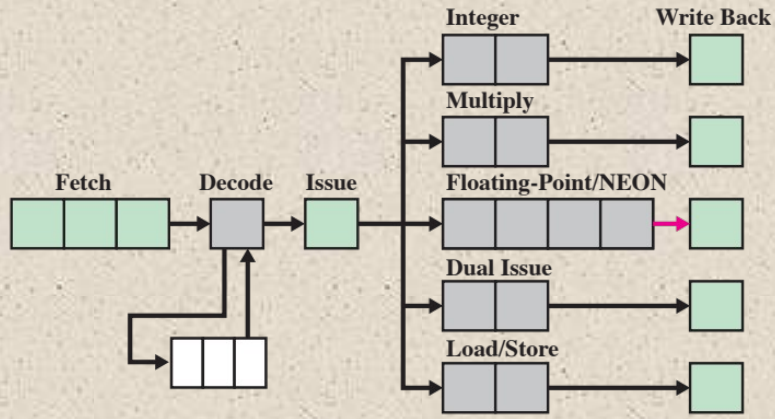
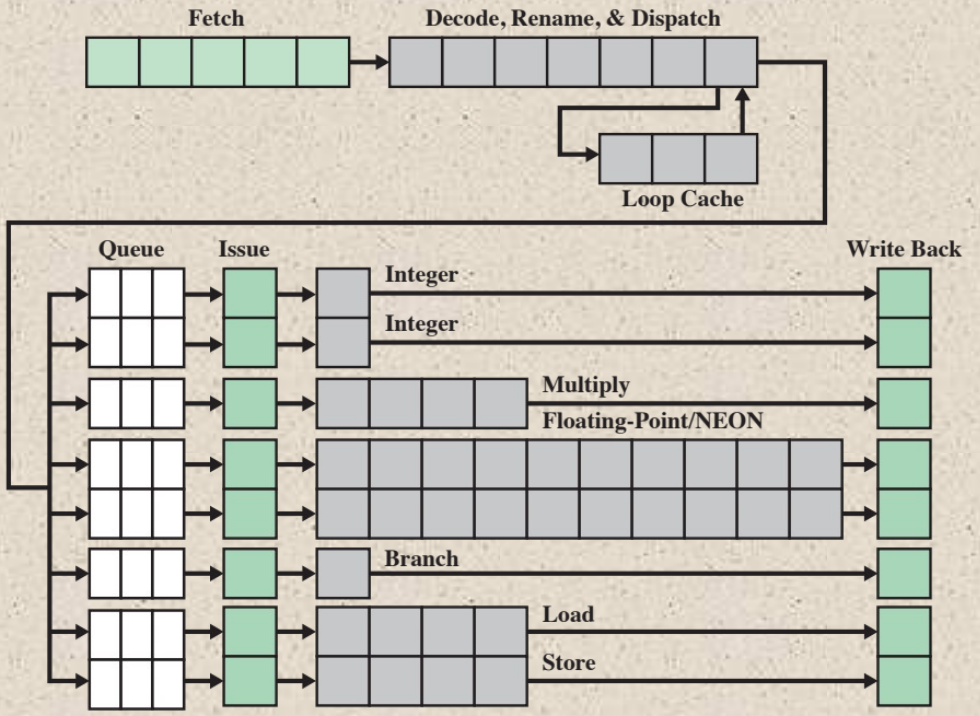


Figure 18.9 Big.Litte Chip Components



(a) Cortex A-7 Pipeline



(b) Cortex A-15 Pipeline

Figure 18.10 Cortex A-7 and A-15 Pipelines

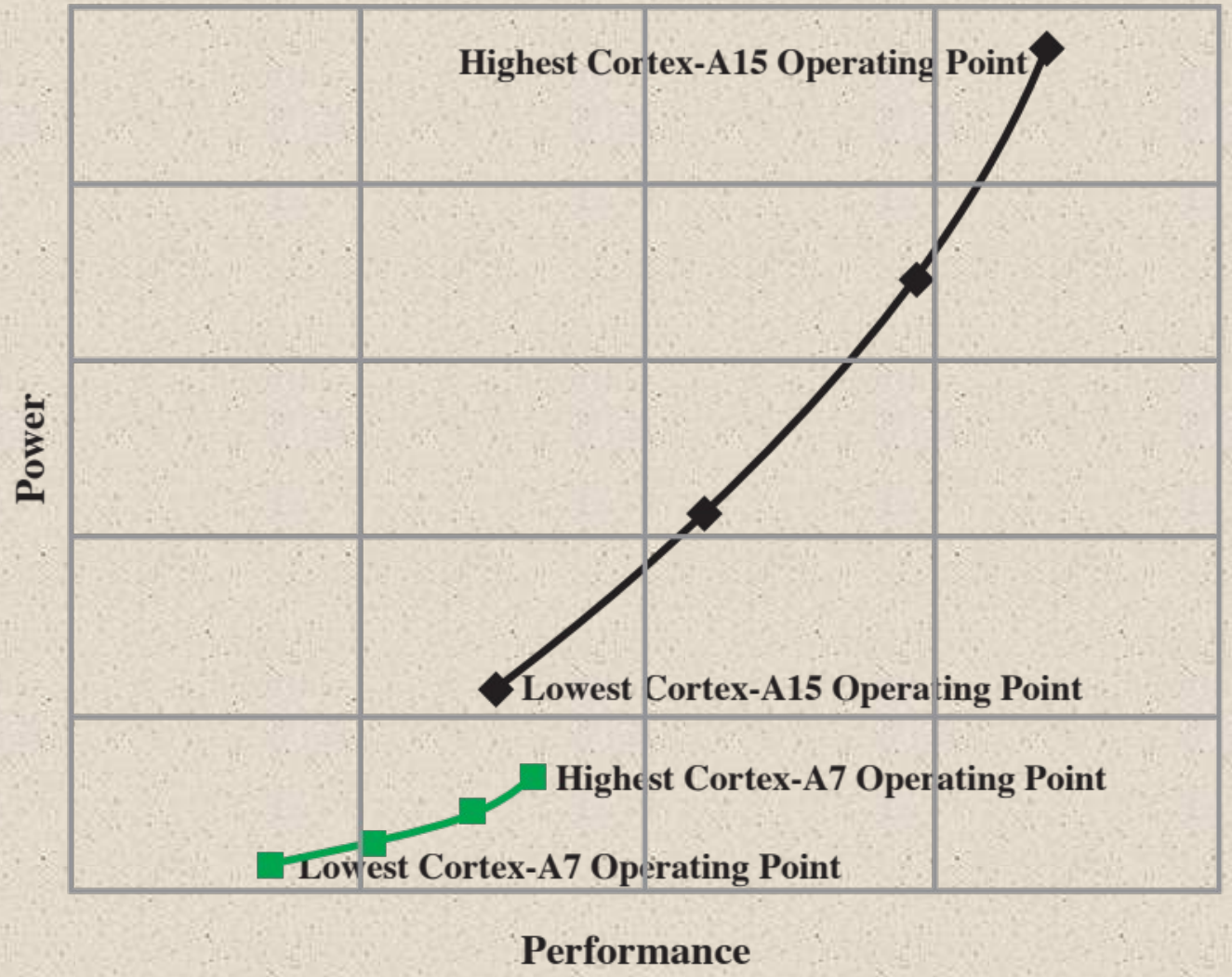


Figure 18.11 Cortex-A7 and A15 Performance Comparison

Cache Coherence

- May be addressed with software-based techniques
  - Software burden consumes too many resources in a SoC chip
- When multiple caches exist there is a need for a cache-coherence scheme to avoid access to invalid data
- There are two main approaches to hardware implemented cache coherence
  - Directory protocols
  - Snoopy protocols
- ACE (Advanced Extensible Interface Coherence Extensions)
  - Hardware coherence capability developed by ARM
  - Can be configured to implement whether directory or snoopy approach
  - Has been designed to support a wide range of coherent masters with differing capabilities
  - Supports coherency between dissimilar processors enabling ARM big.Little technology
  - Supports I/O coherency for un-cached masters, supports masters with differing cache line sizes, differing internal cache state models, and masters with write-back or write-through caches

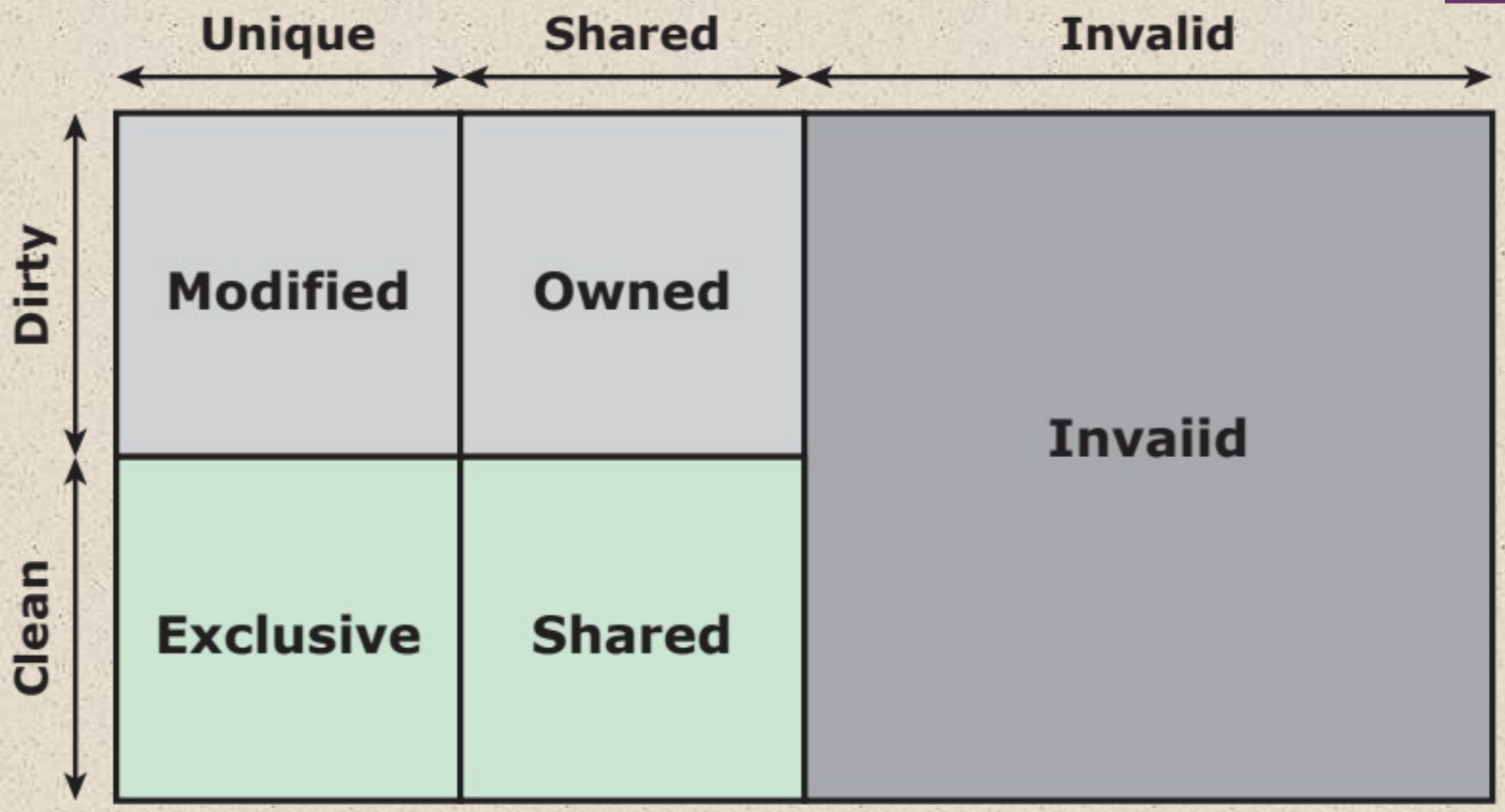


Figure 18.12 ARM ACE Cache Line States

Table 18.2 Comparison of States in Snoop Protocols

(a) MESI

	Modified	Exclusive	Shared	Invalid
Clean/Dirty	Dirty	Clean	Clean	N/A
Unique?	Yes	Yes	No	N/A
Can write?	Yes	Yes	No	N/A
Can forward?	Yes	Yes	Yes	N/A
Comments	Must write back to share or replace	Transitions to M on write	Shared implies clean, can forward	Cannot read

(b) MOESI

	Modified	Owned	Exclusive	Shared	Invalid
Clean/Dirty	Dirty	Dirty	Clean	Either	N/A
Unique?	Yes	Yes	Yes	No	N/A
Can write?	Yes	Yes	Yes	No	N/A
Can forward?	Yes	Yes	Yes	No	N/A
Comments	Can share without write back	Must write back to transition	Transitions to M on write	Shared, can be dirty or clean	Cannot read

(Table can be found on page 676 in the

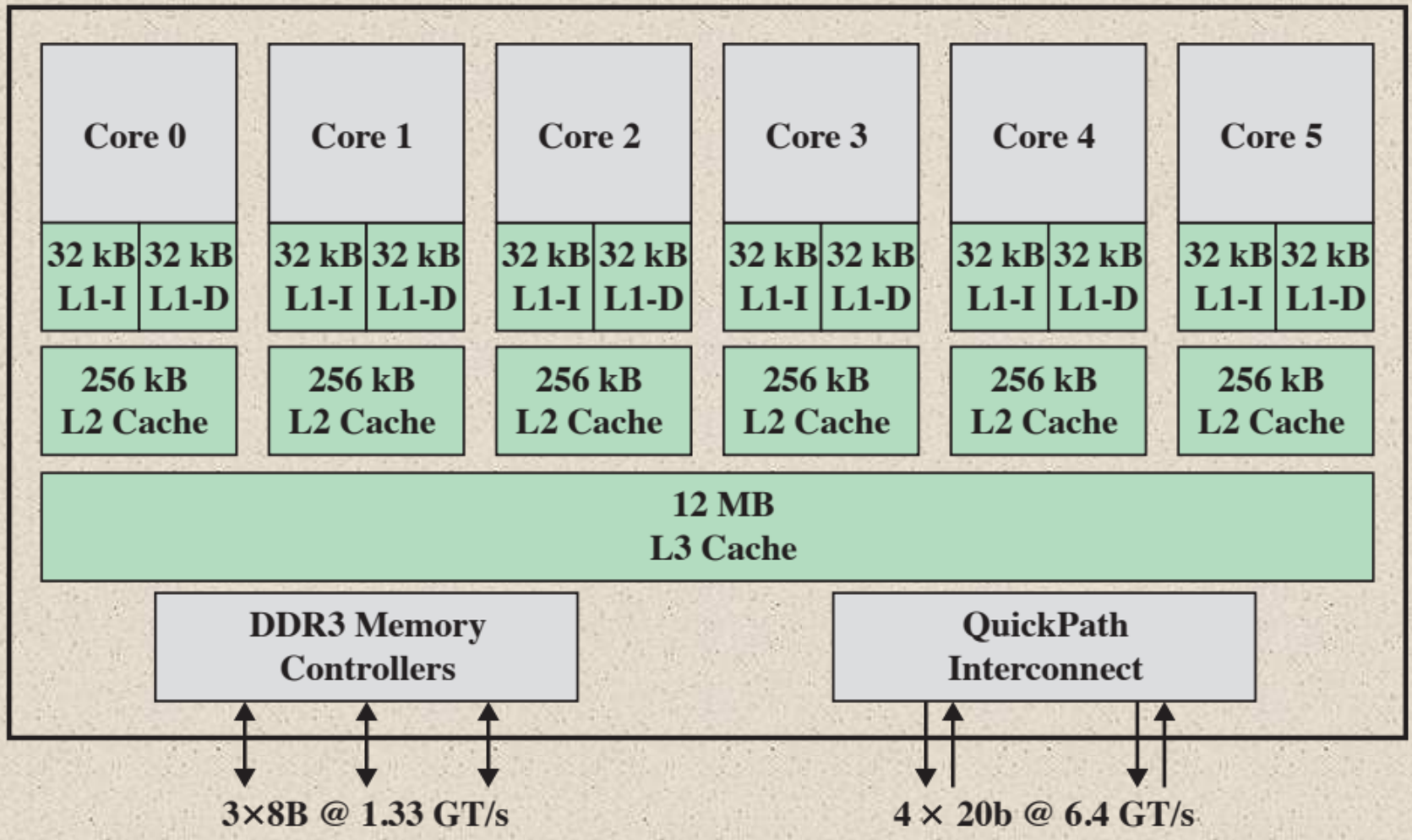


Figure 18.13 Intel Core i7-990X Block Diagram

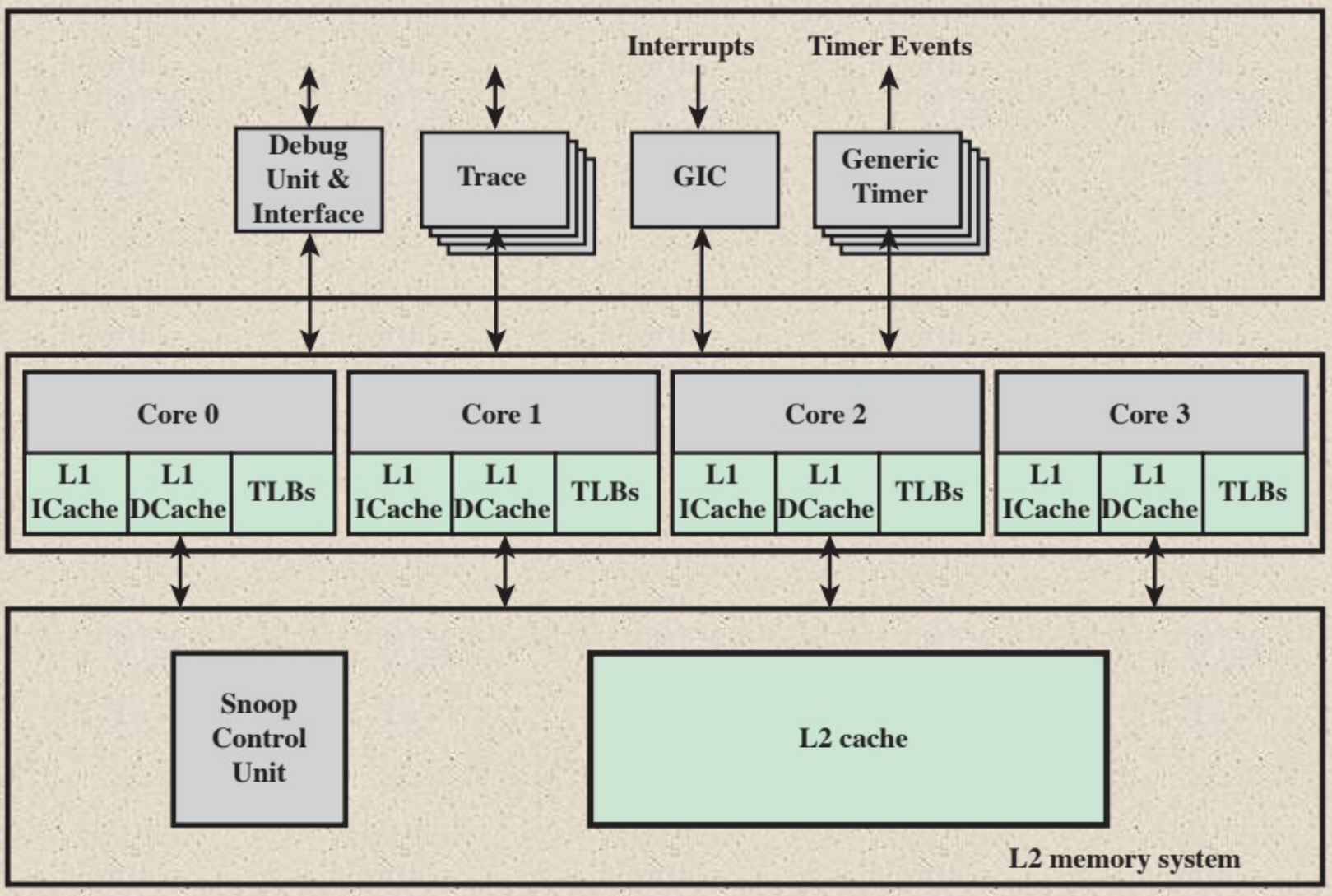


Figure 18.14 ARM Cortex-A15 MPCore Chip Block Diagram

Interrupt Handling

Generic interrupt controller (GIC) provides:

- Masking of interrupts
- Prioritization of the interrupts
- Distribution of the interrupts to the target A15 cores
- Tracking the status of interrupts
- Generation of interrupts by software

GIC

- Is memory mapped
- Is a single functional unit that is placed in the system alongside A15 cores
- This enables the number of interrupts supported in the system to be independent of the A15 core design
- Is accessed by the A15 cores using a private interface through the SCU

GIC

Designed to satisfy two functional requirements:

- Provide a means of routing an interrupt request to a single CPU or multiple CPUs as required
- Provide a means of interprocessor communication so that a thread on one CPU can cause activity by a thread on another CPU

Can route an interrupt to one or more CPUs in the following three ways:

- An interrupt can be directed to a specific processor only
- An interrupt can be directed to a defined group of processors
- An interrupt can be directed to all processors

Interrupts can be:

- Inactive
  - One that is nonasserted, or which in a multiprocessing environment has been completely processed by that CPU but can still be either Pending or Active in some of the CPUs to which it is targeted, and so might not have been cleared at the interrupt source
- Pending
  - One that has been asserted, and for which processing has not started on that CPU
- Active
  - One that has been started on that CPU, but processing s not complete
  - Can be pre-empted when a new interrupt of higher priority interrupts A15 core interrupt processing
- Interrupts come from the following sources:
  - Interprocessor interrupts (IPIs)
  - Private timer and/or watchdog interrupts
  - Legacy FIQ lines
  - Hardware interrupts

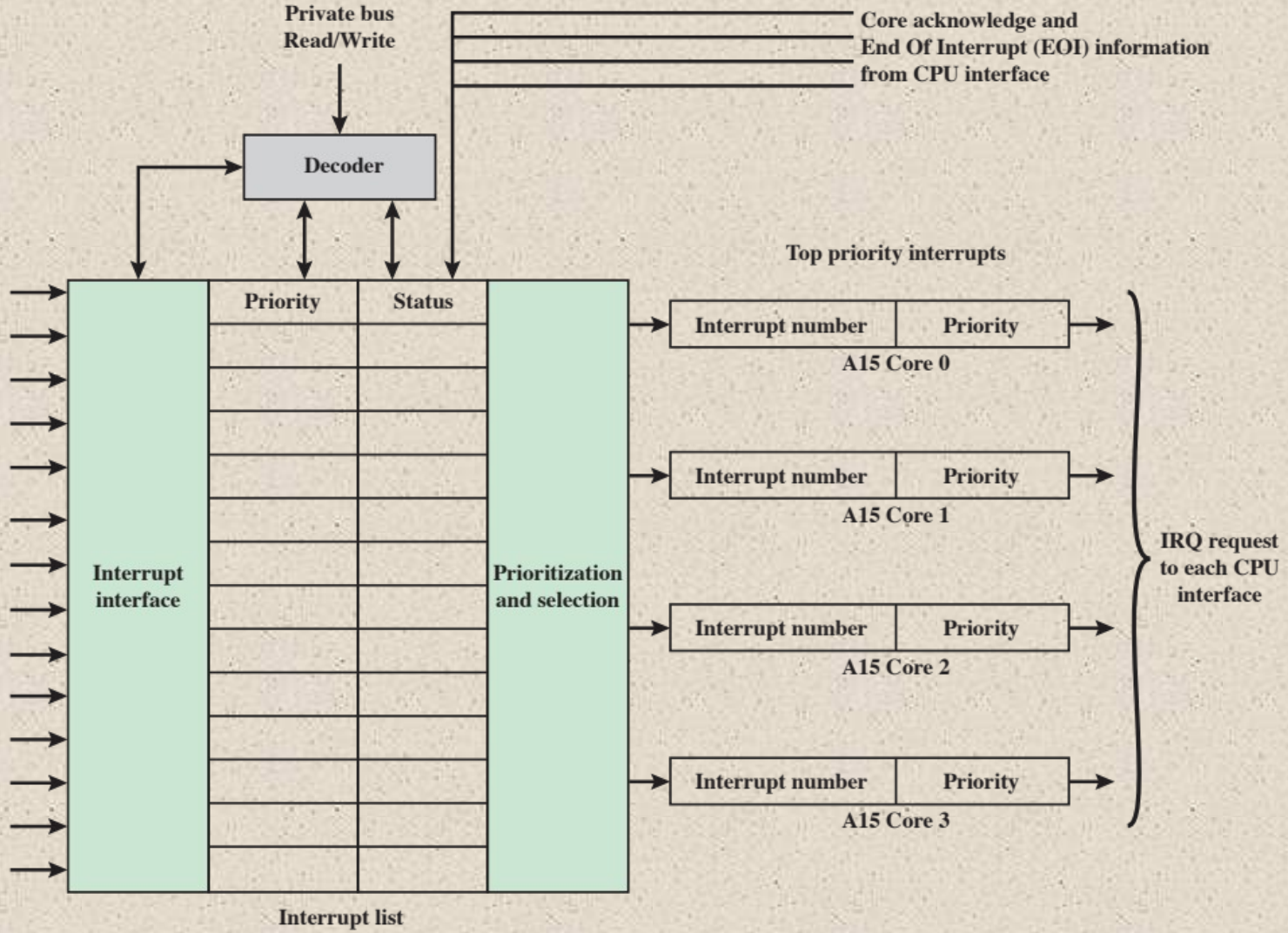
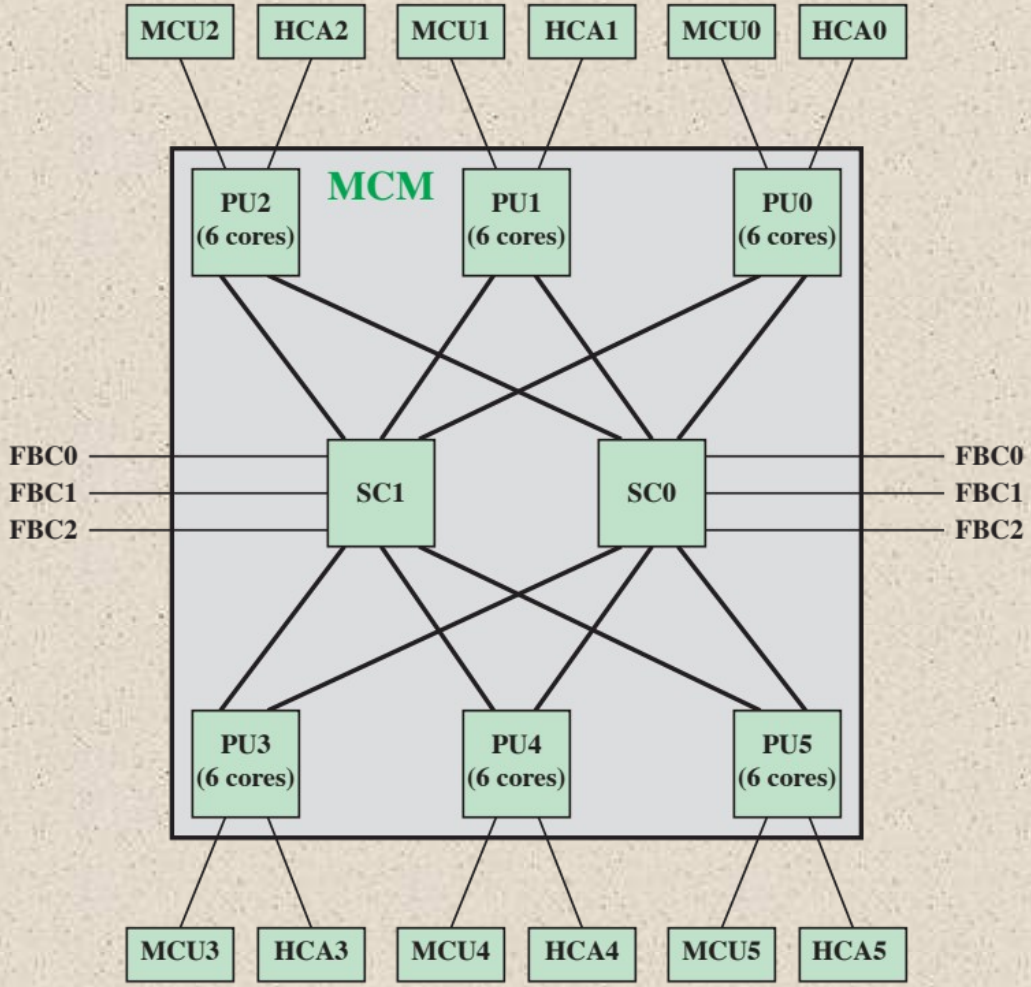


Figure 18.15 Interrupt Distributor Block Diagram

Cache Coherency

- Snoop Control Unit (SCU) resolves most of the traditional bottlenecks related to access to shared data and the scalability limitation introduced by coherence traffic
- L1 cache coherency scheme is based on the MESI protocol
- Direct Data Intervention (DDI)
  - Enables copying clean data between L1 caches without accessing external memory
  - Reduces read after write from L1 to L2
  - Can resolve local L1 miss from remote L1 rather than L2
- Duplicated tag RAMs
  - Cache tags implemented as separate block of RAM
  - Same length as number of lines in cache
  - Duplicates used by SCU to check data availability before sending coherency commands
  - Only send to CPUs that must update coherent data cache
- Migratory lines
  - Allows moving dirty data between CPUs without writing to L2 and reading back from external memory



FBC = fabric book connectivity
HCA = host channel adapter
MCM = multichip module

MCU = memory control unit
PU = processor unit
SC = storage control

Figure 18.16 IBM zEC12 Processor Node Structure

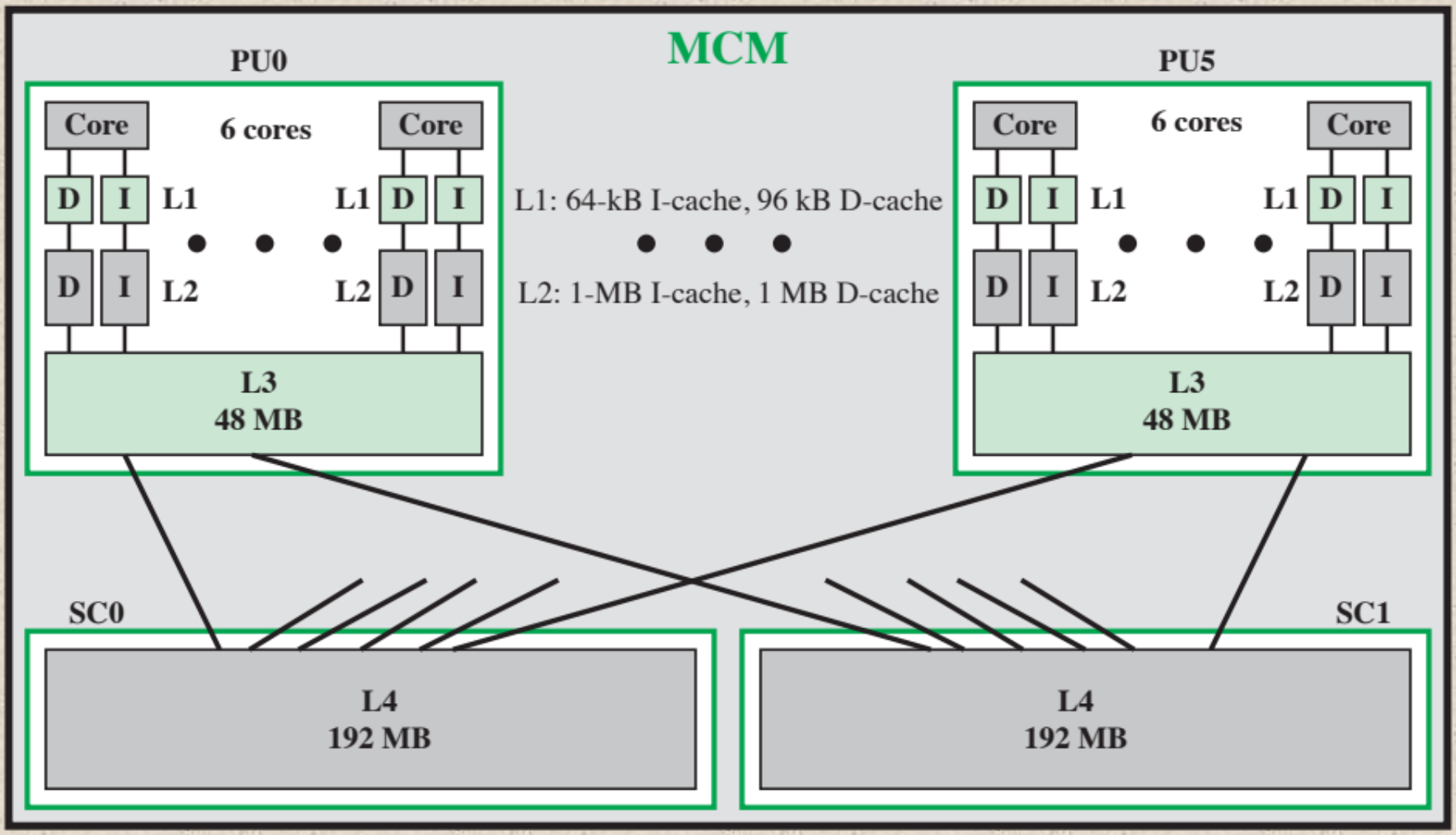


Figure 18.17 IBM zEC12 Cache Hierarchy

Summary

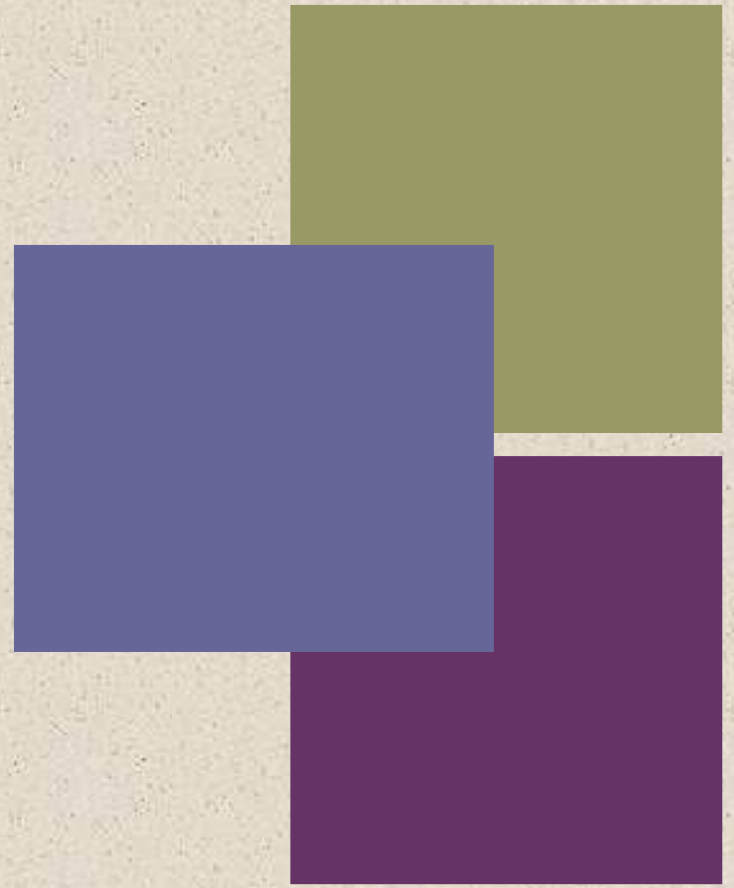
Chapter 18

- Hardware performance issues
  - Increase in parallelism and complexity
  - Power consumption
- Software performance issues
  - Software on multicore
  - Valve game software example
- Intel Core i7-990X
- IBM zEnterprise EC12 mainframe
  - Organization
  - Cache structure

Multicore Computers

- Multicore organization
  - Levels of cache
  - Simultaneous multithreading
- Heterogeneous multicore organization
  - Different instruction set architectures
  - Equivalent instruction set architectures
  - Cache coherence and the MOESI model
- ARM Cortex-A15 MPCore
  - Interrupt handling
  - Cache coherency
  - L2 cache coherency

William Stallings
Computer Organization and Architecture
10th Edition



Chapter 19
General-Purpose Graphic Processing Units

Compute Unified Device Architecture (CUDA)

- A parallel computing platform and programming model created by NVIDIA and implemented by the graphics processing units (GPUs) that they produce
- CUDA C is a C/C++ based language
- Program can be divided into three general sections
  - Code to be run on the host (CPU)
  - Code to be run on the device (GPU)
  - The code related to the transfer of data between the host and the device
- The data-parallel code to be run on the GPU is called a *kernel*
  - Typically will have few to no branching statements
  - Branching statements in the kernel result in serial execution of the threads in the GPU hardware
- A *thread* is a single instance of the kernel function
  - The programmer defines the number of threads launched when the kernel function is called
  - The total number of threads defined is typically in the thousands to maximize the utilization of the GPU processor cores, as well as maximize the available speedup
  - The programmer specifies how these threads are to be bundled

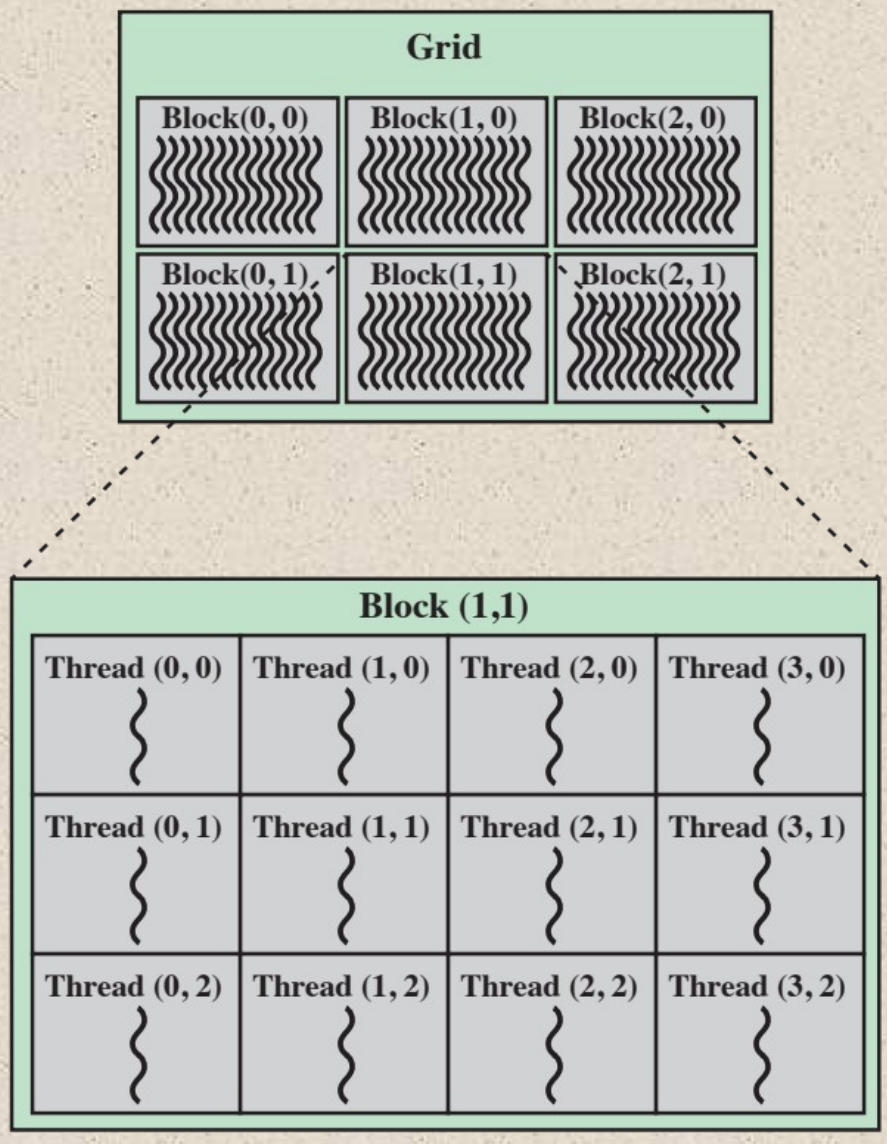


Figure 19.1 Relationship Among Threads, Blocks, and a Grid

Table 19.1

CUDA Terms to GPU's Hardware Components Equivalence Mapping

CUDA Term	Definition	Equivalent GPU Hardware Component
Kernel	Parallel code in the form of a function to be run on GPU	Not applicable
Thread	An instance of the kernel on the GPU	GPU/CUDA processor core
Block	A group of threads assigned to a particular SM	CUDA multiprocessor (SM)
Grid	The GPU	GPU

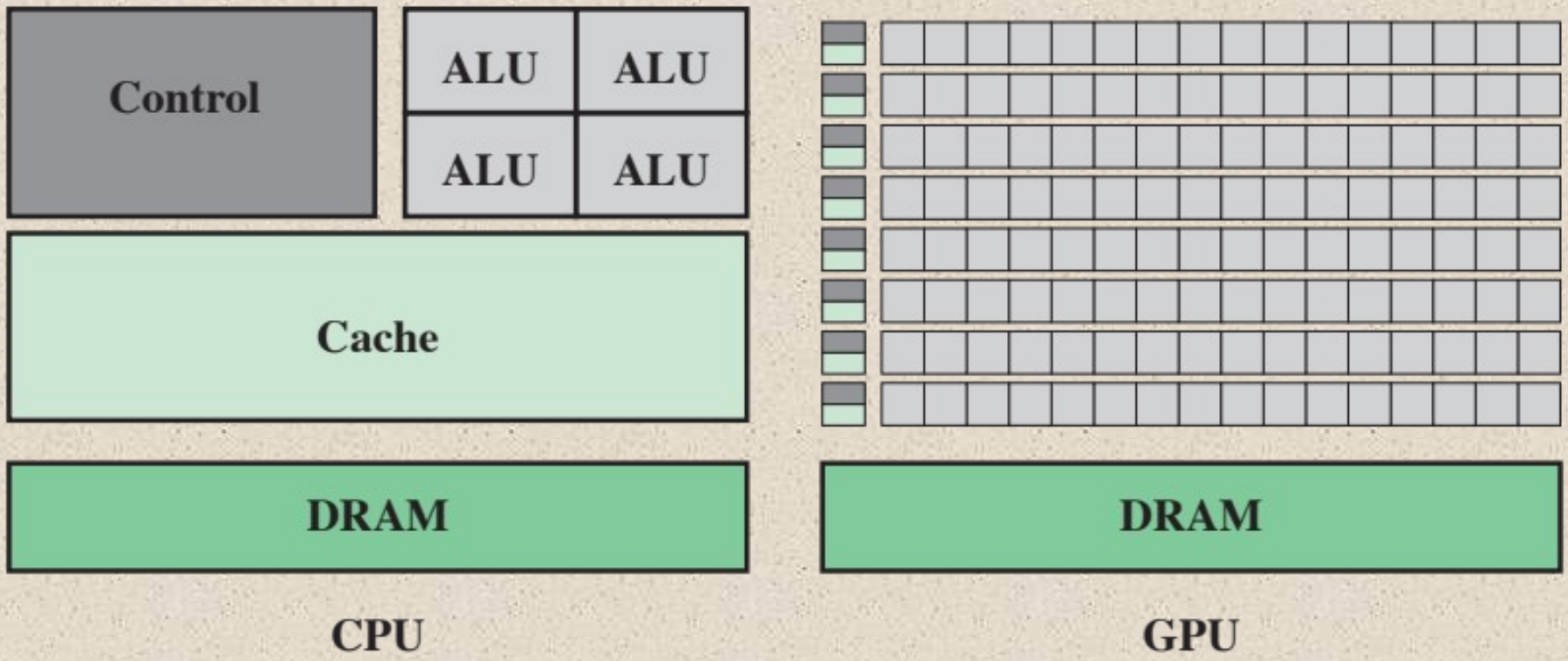


Figure 19.2 CPU vs. GPU Silicon Area/Transistor Dedication

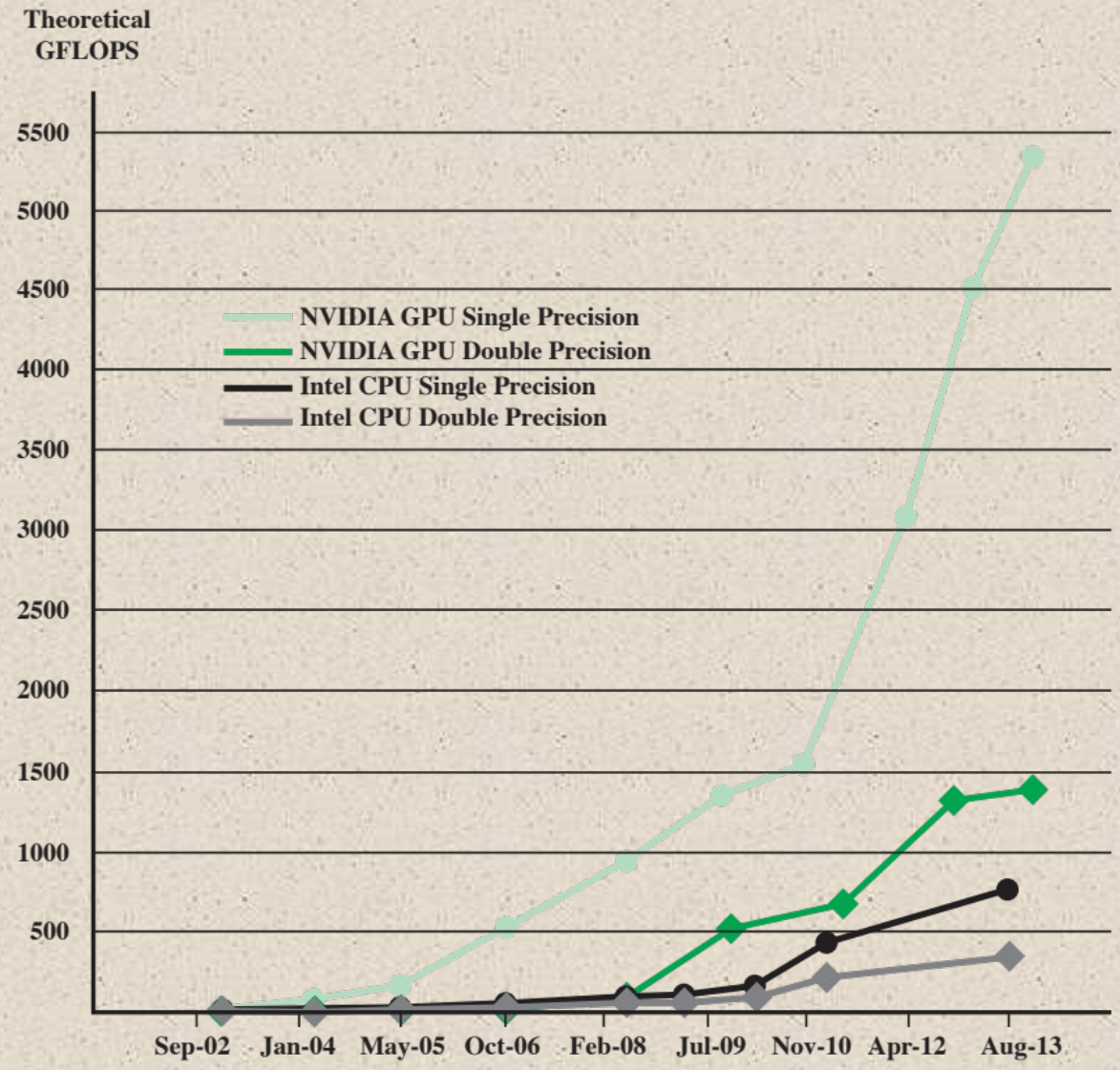


Figure 19.3 Floating-Point Operations per Second for CPU and GPU

GPU Architecture Overview

The historical evolution can be divided up into three major phases:

The first phase would cover early 1980s to late 1990s, where the GPU was composed of fixed, nonprogrammable, specialized processing stages

The second phase would cover the iterative modification of the resulting Phase I GPU architecture from a fixed, specialized, hardware pipeline to a fully programmable processor (early to mid-2000s)

The third phase covers how the GPU/GPGPU architecture makes an excellent and affordable highly parallelized SIMD coprocessor for accelerating the run times of some nongraphics-related programs, along with how a GPGPU language maps to this architecture

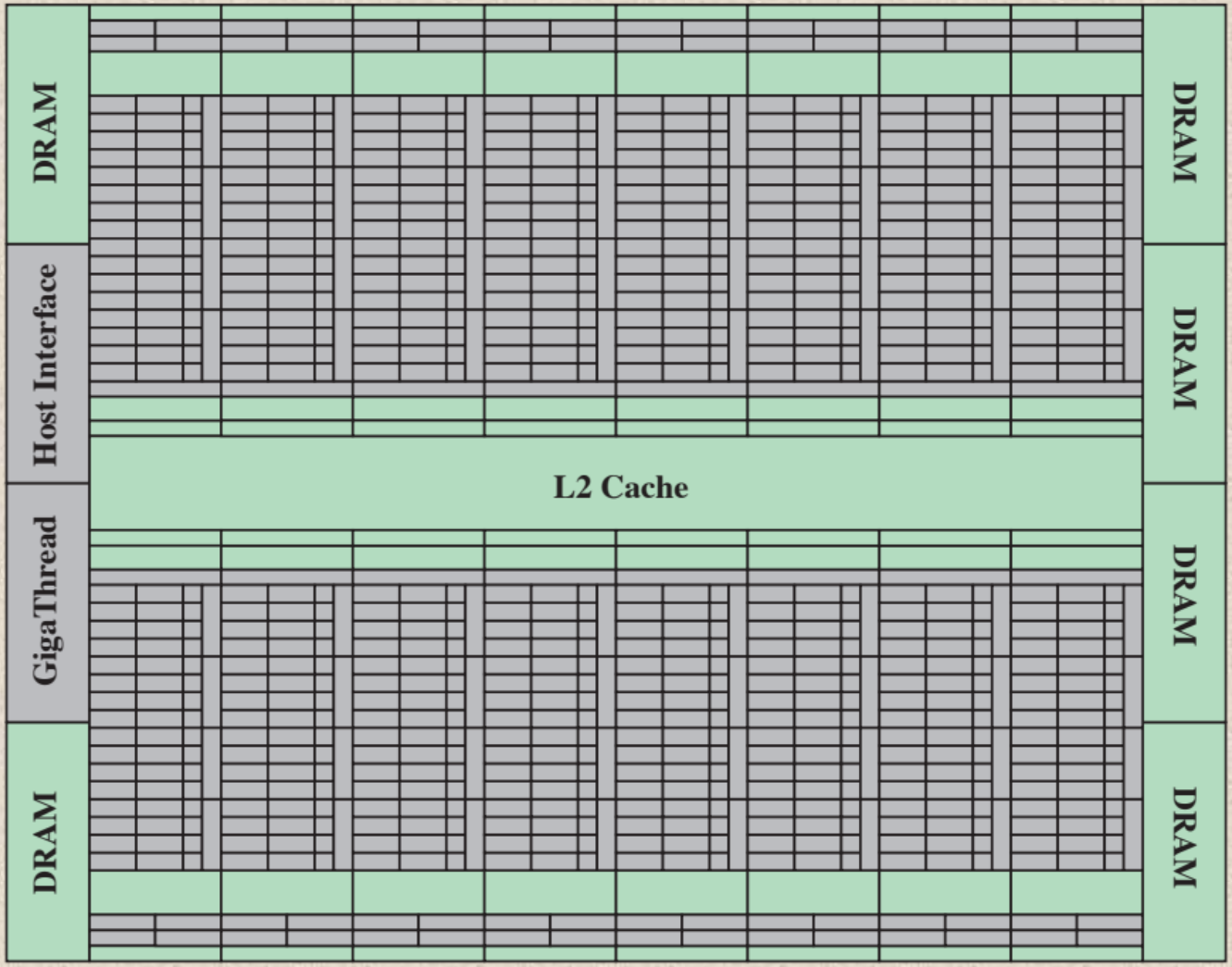


Figure 19.4 NVIDIA Fermi Architecture

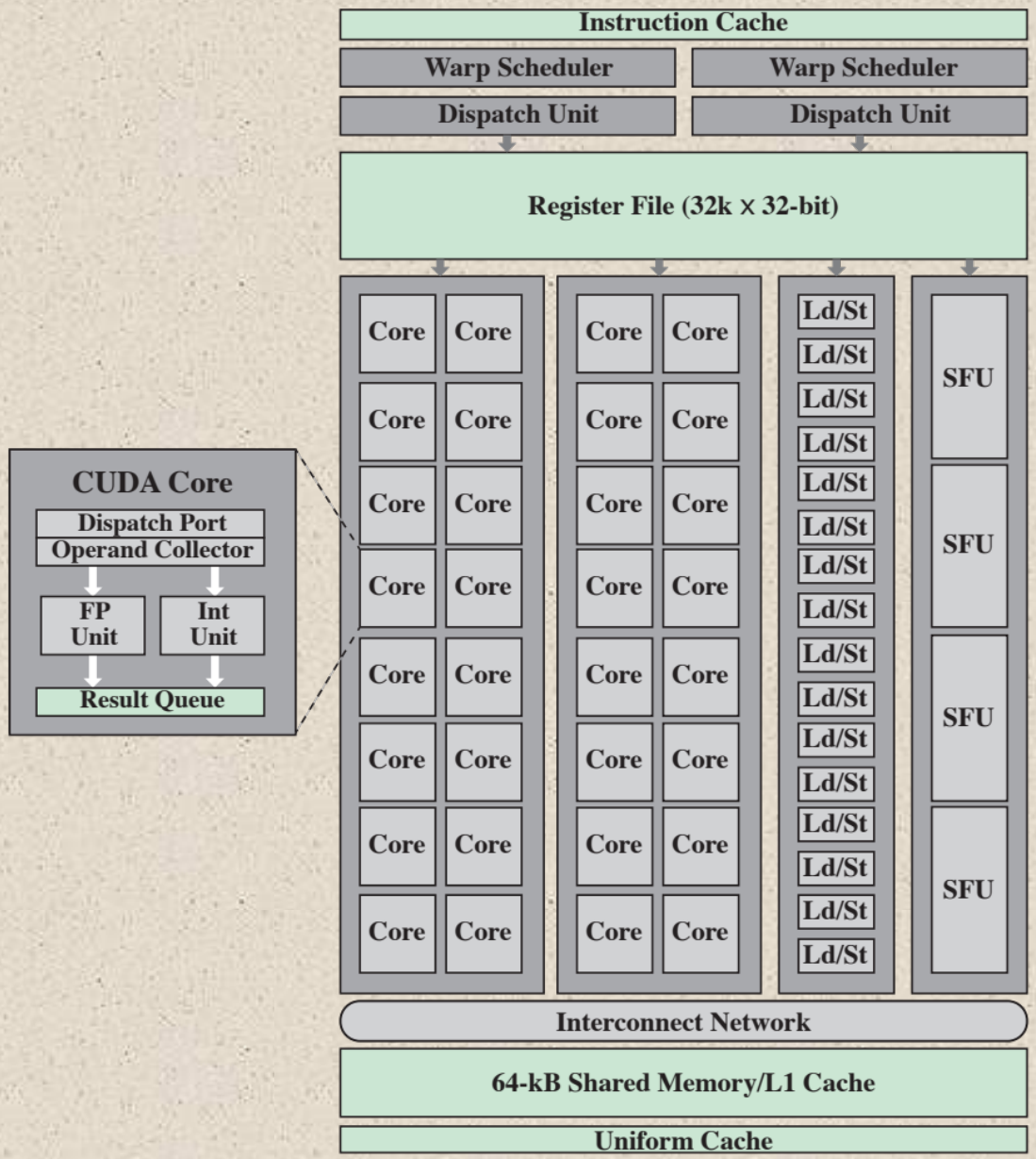


Figure 19.5 Single SM Architecture

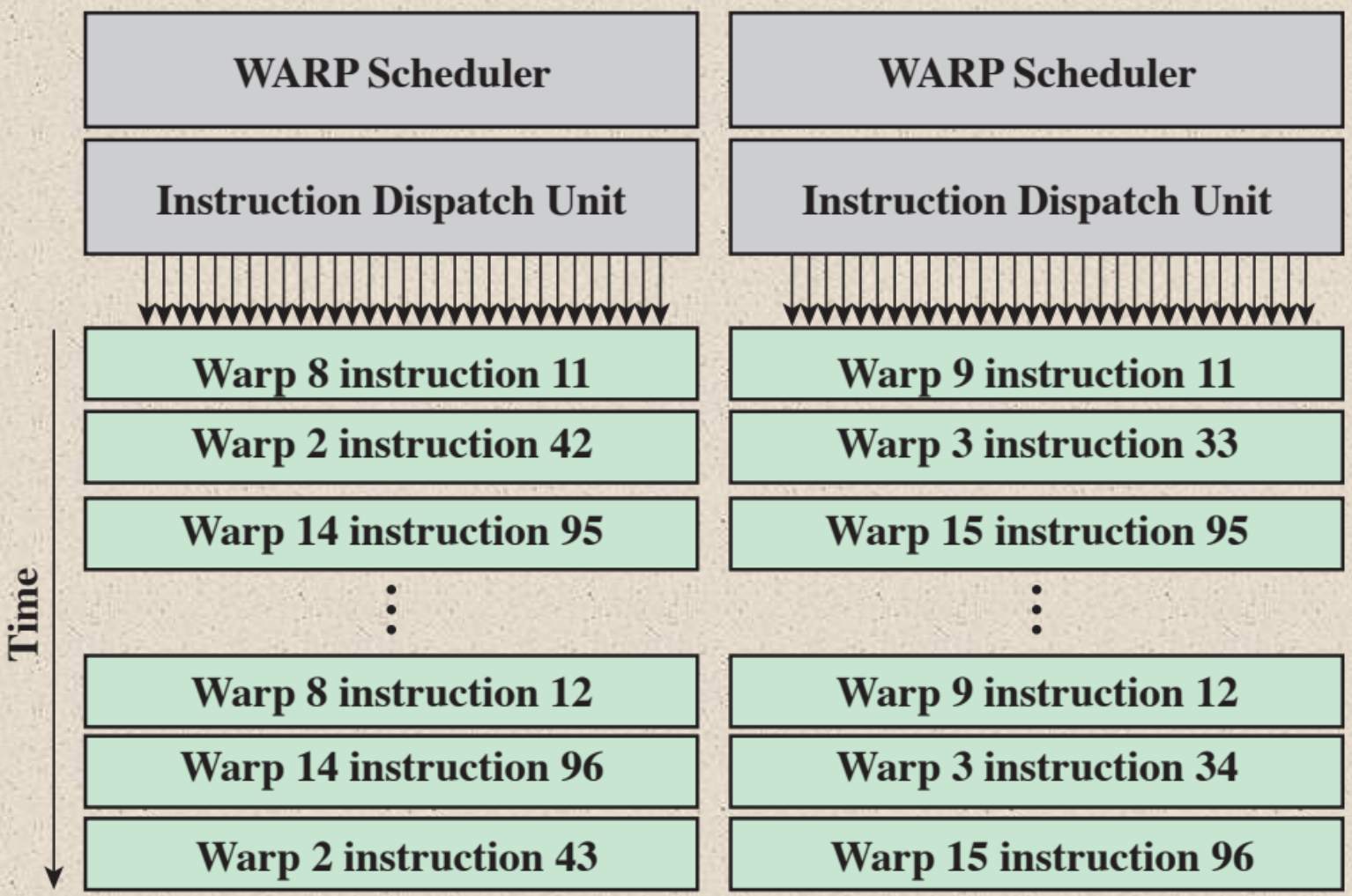


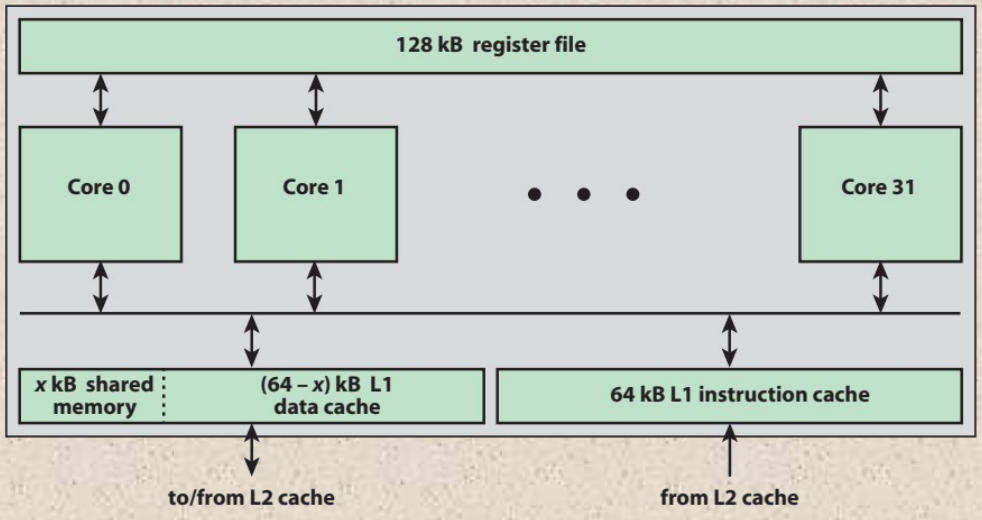
Figure 19.6 Dual Warp Schedulers and Instruction Dispatch Units Run Example

CUDA Cores

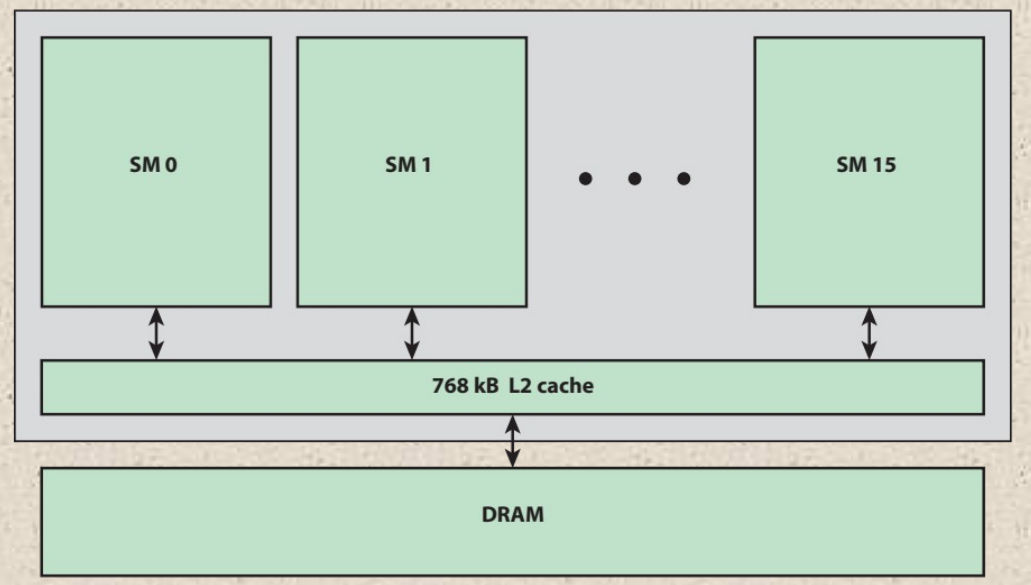
- The NVIDIA GPU processor cores are also known as CUDA cores
- There are a total of 32 CUDA cores dedicated to each SM in the Fermi architecture
- Each CUDA core has two separate pipelines or data paths
  - An integer (INT) unit pipeline
    - Is capable of 32-bit, 64-bit, and extended precision for integer and logic/bitwise operations
  - Floating-point (FP) unit pipeline
    - Can perform a single-precision FP operation, while a double-precision FP operation requires two CUDA cores

Table 19.2
GPU's Memory Hierarchy Attributes

Memory Type	Relative Access Times	Access Type	Scope	Data Lifetime
Registers	Fastest. On-chip	R/W	Single thread	Thread
Shared	Fast. On-chip	R/W	All threads in a block	Block
Local	100× to 150× slower than shared & register. Off-chip	R/W	Single thread	Thread
Global	100× to 150× slower than shared & register. Off-chip.	R/W	All threads & host	Application
Constant	100× to 150× slower than shared & register. Off-chip	R	All threads & host	Application
Texture	100× to 150× slower than shared & register. Off-chip	R	All threads & host	Application



(a) SM memory architecture



(b) Overal memory architecture

Figure 19.7 Fermi Memory Architecture

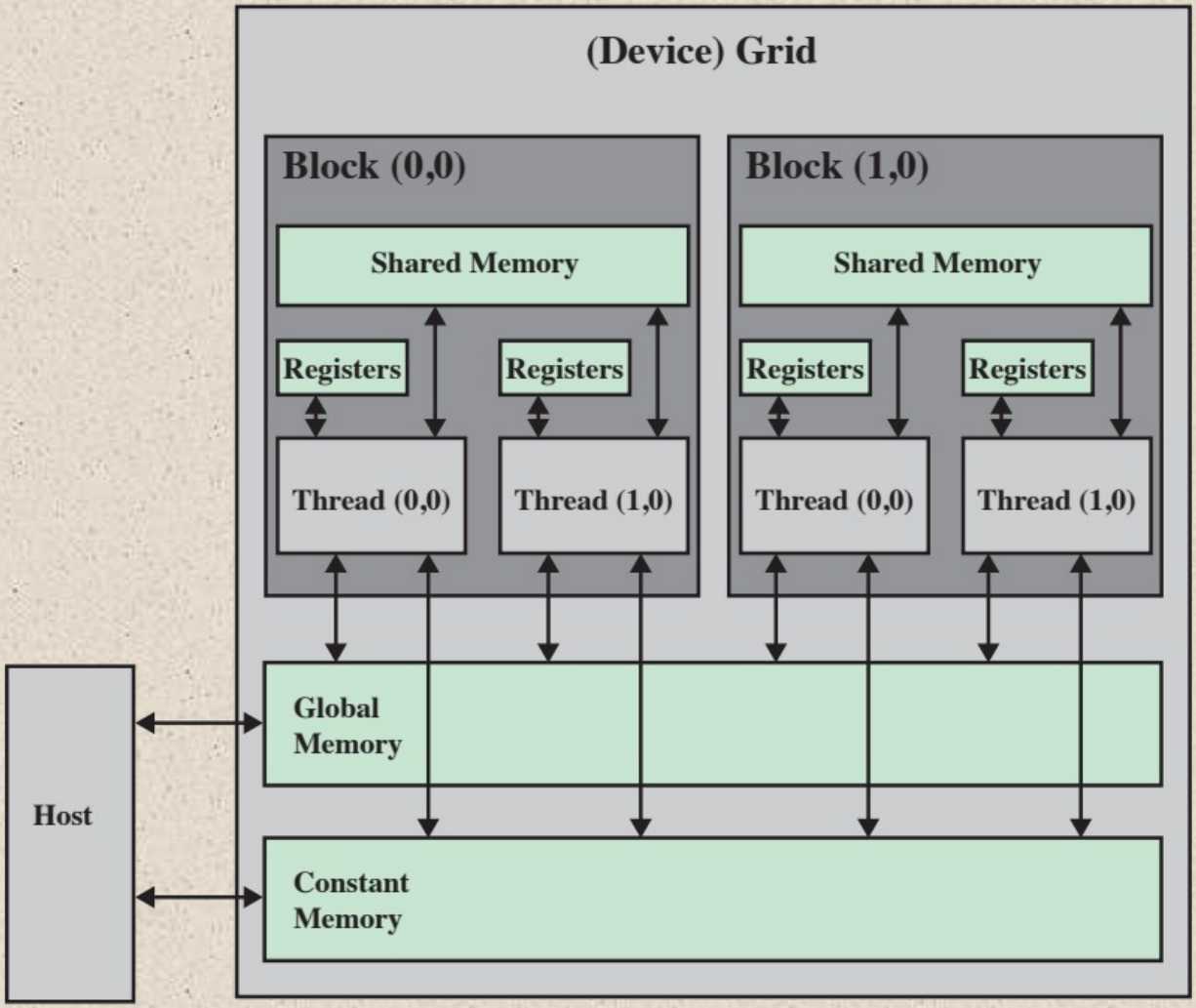


Figure 19.8 CUDA Representation of a GPU's Basic Architecture. The example GPU shown has two SMs and two CUDA cores per SM.

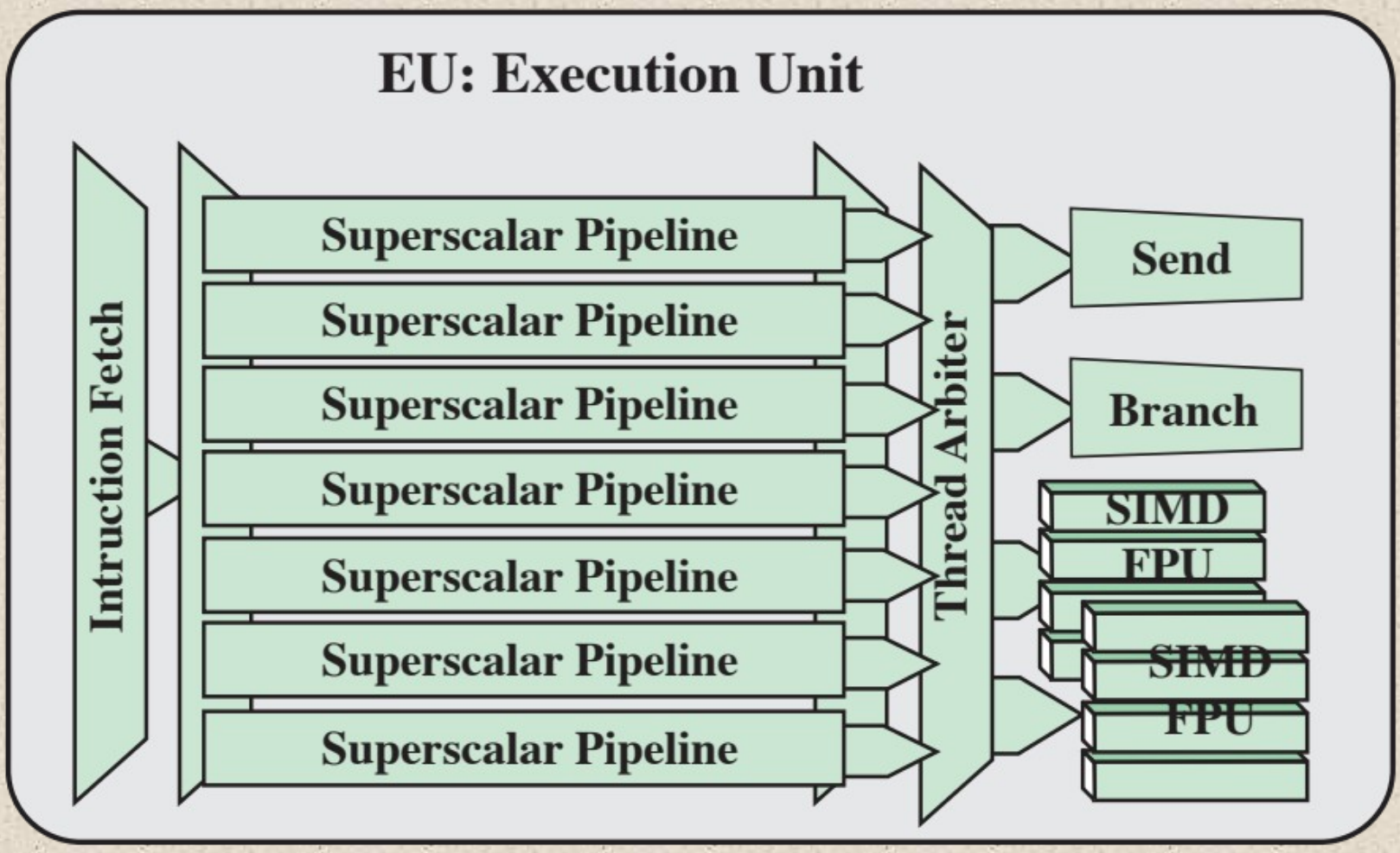


Figure 19.9 Intel Gen8 Execution Unit

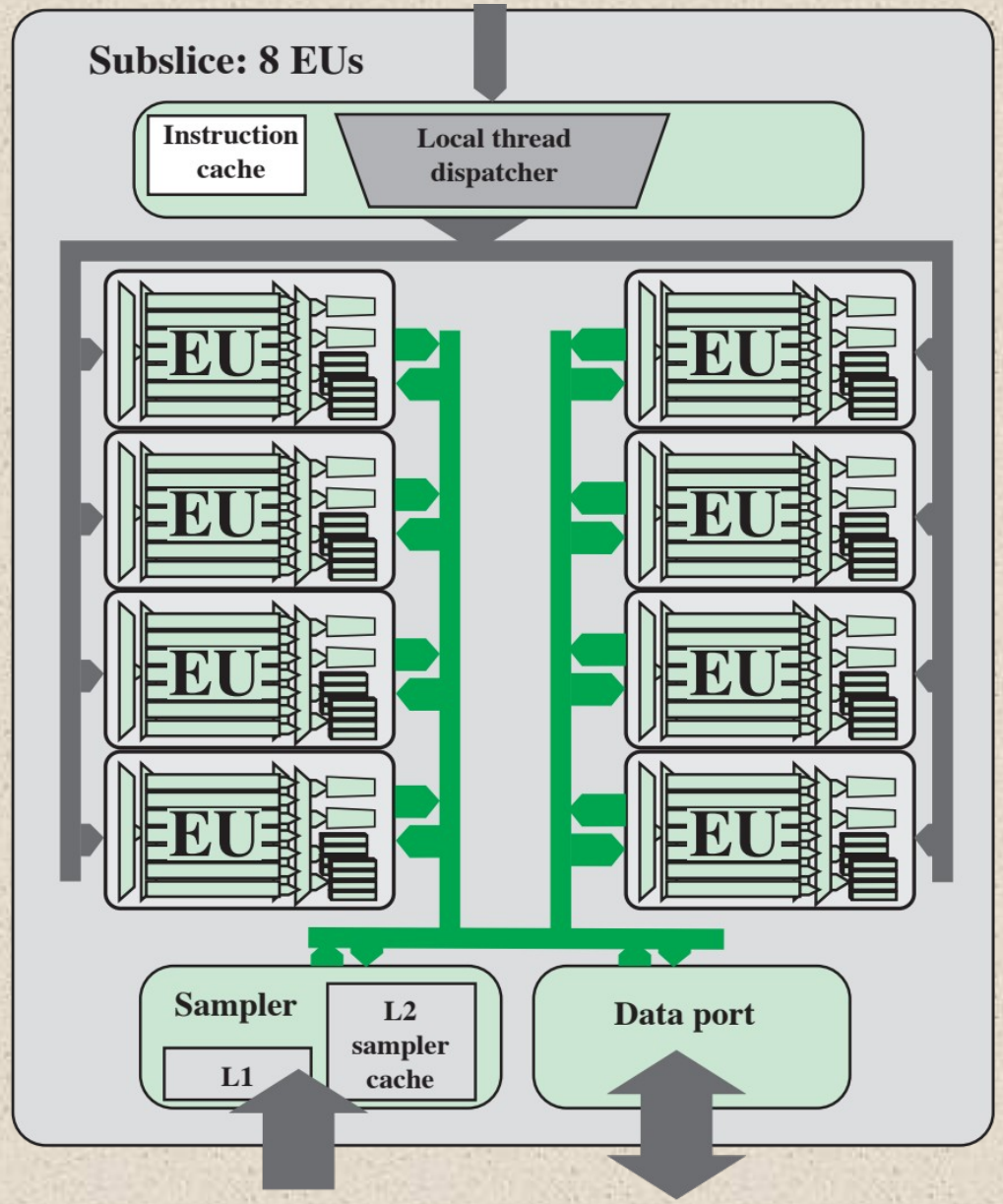


Figure 19.10 Intel Gen8 Subslice

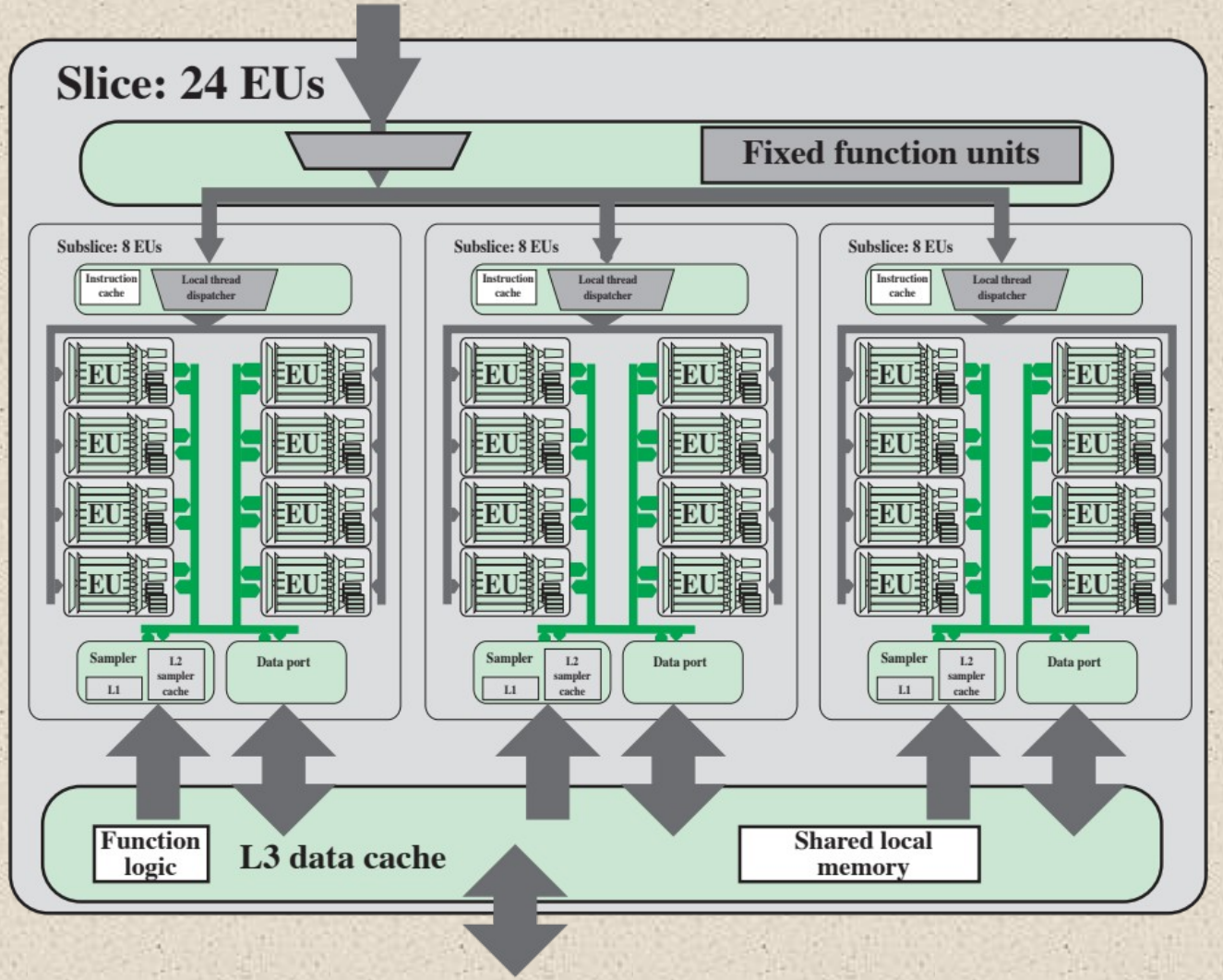


Figure 19.11 Intel Gen8 Slice

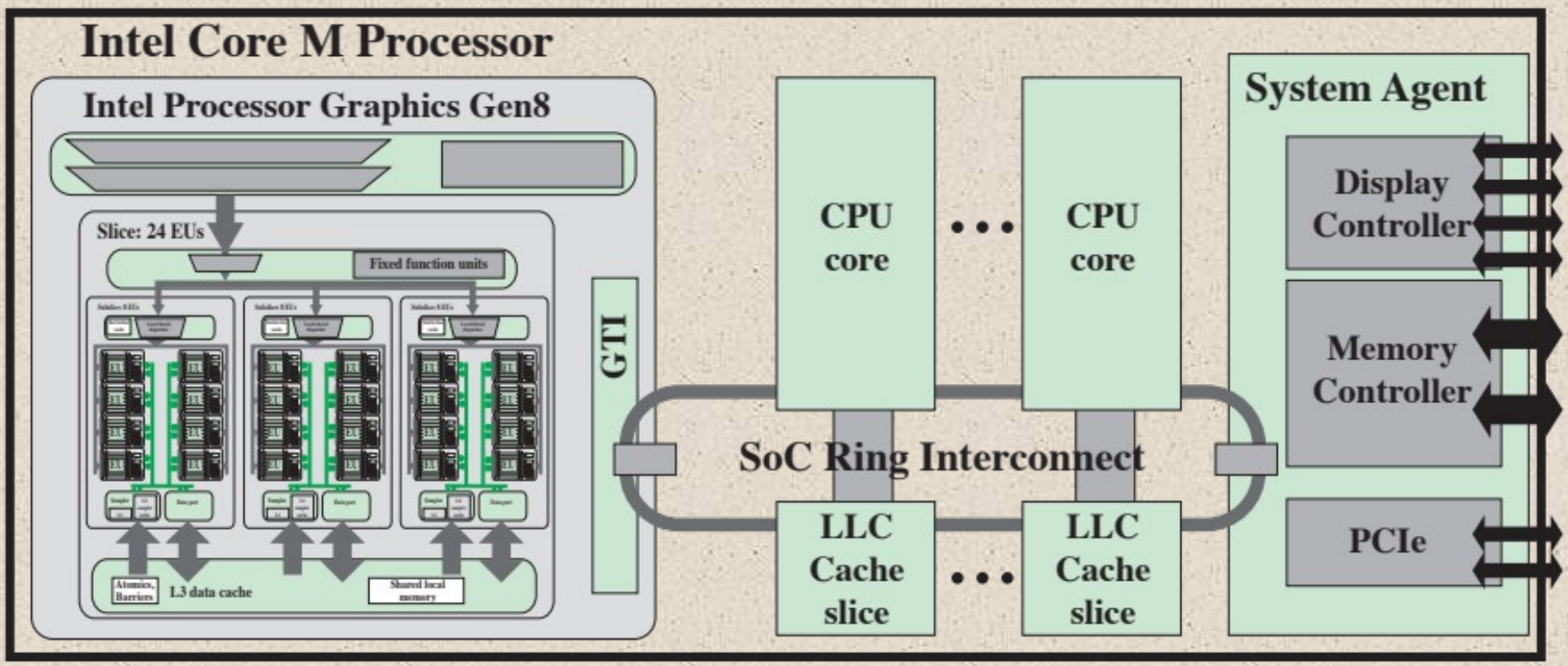


Figure 19.12 Intel Core M Processor SoC

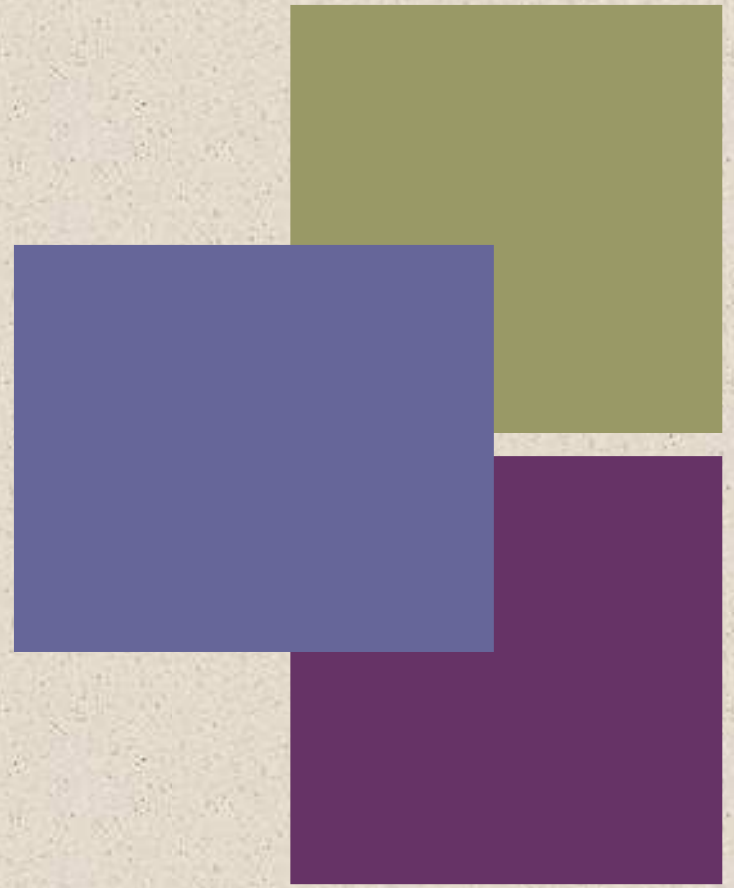
Summary

Chapter 19

General-Purpose Graphic Processing Units

- CUDA basics
- GPU versus CPU
  - Basic differences between CPU and GPU architectures
  - Performance and performance per watt comparison
- Intel's Gen8 GPU
- GPU architecture overview
  - Baseline GPU architecture
  - Full chip layout
  - Streaming multiprocessor architecture details
  - Importance of knowing and programming to your memory types

William Stallings
Computer Organization and Architecture
10th Edition



Chapter 20

Control Unit Operation

Micro-Operations

- The functional, or atomic, operations of a processor
- Series of steps, each of which involves the processor registers
- *Micro* refers to the fact that each step is very simple and accomplishes very little
- The execution of a program consists of the sequential execution of instructions
  - Each instruction is executed during an instruction cycle made up of shorter subcycles (fetch, indirect, execute, interrupt)
  - The execution of each subcycle involves one or more shorter operations (micro-operations)

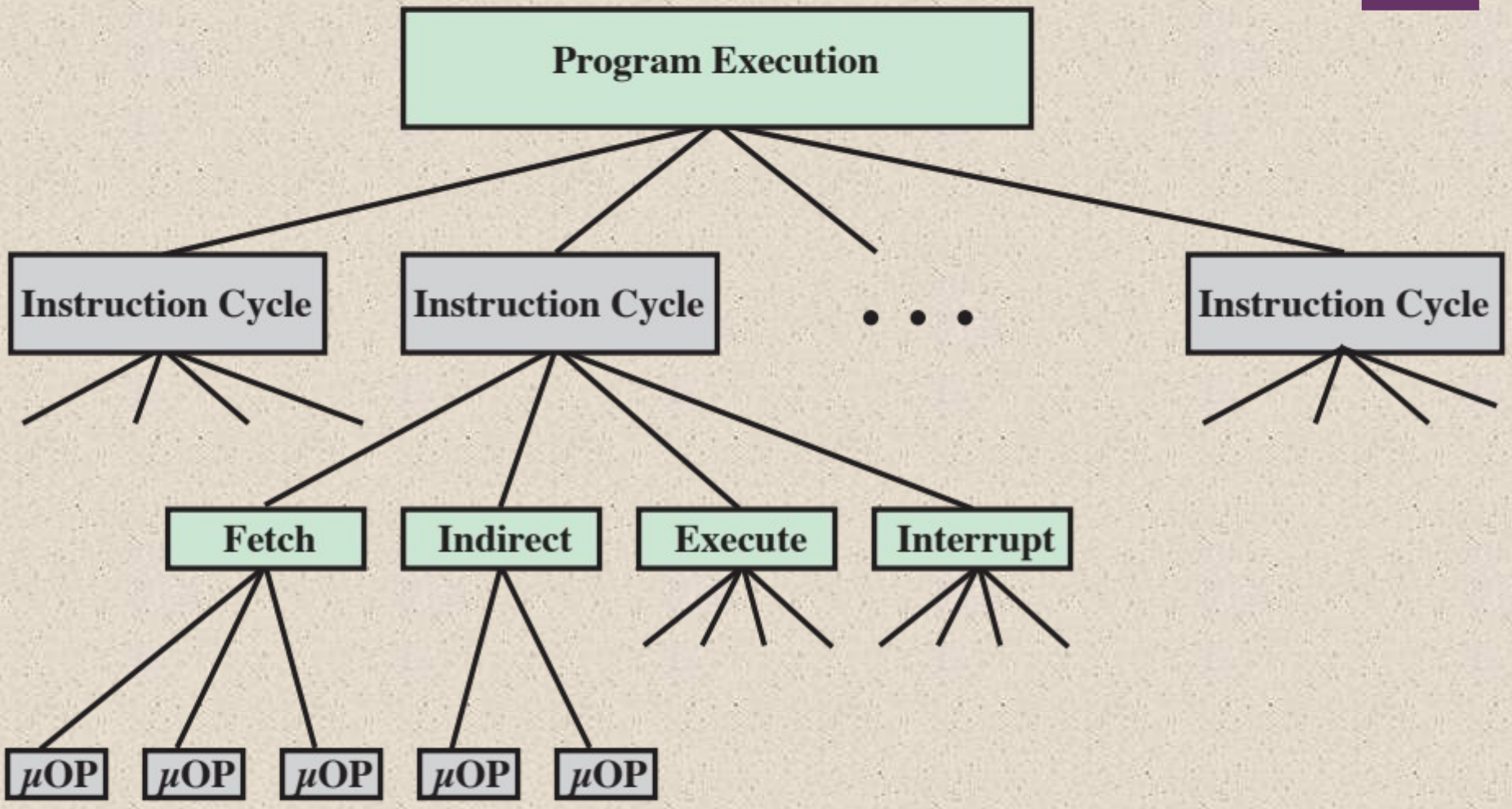


Figure 20.1 Constituent Elements of a Program Execution

The Fetch Cycle

- Occurs at the beginning of each instruction cycle and causes an instruction to be fetched from memory
- Four registers are involved:
  - Memory Address Register (MAR)
    - Connected to address bus
    - Specifies address for read or write operation
  - Memory Buffer Register (MBR)
    - Connected to data bus
    - Holds data to write or last data read
  - Program Counter (PC)
    - Holds address of next instruction to be fetched
  - Instruction Register (IR)
    - Holds last instruction fetched

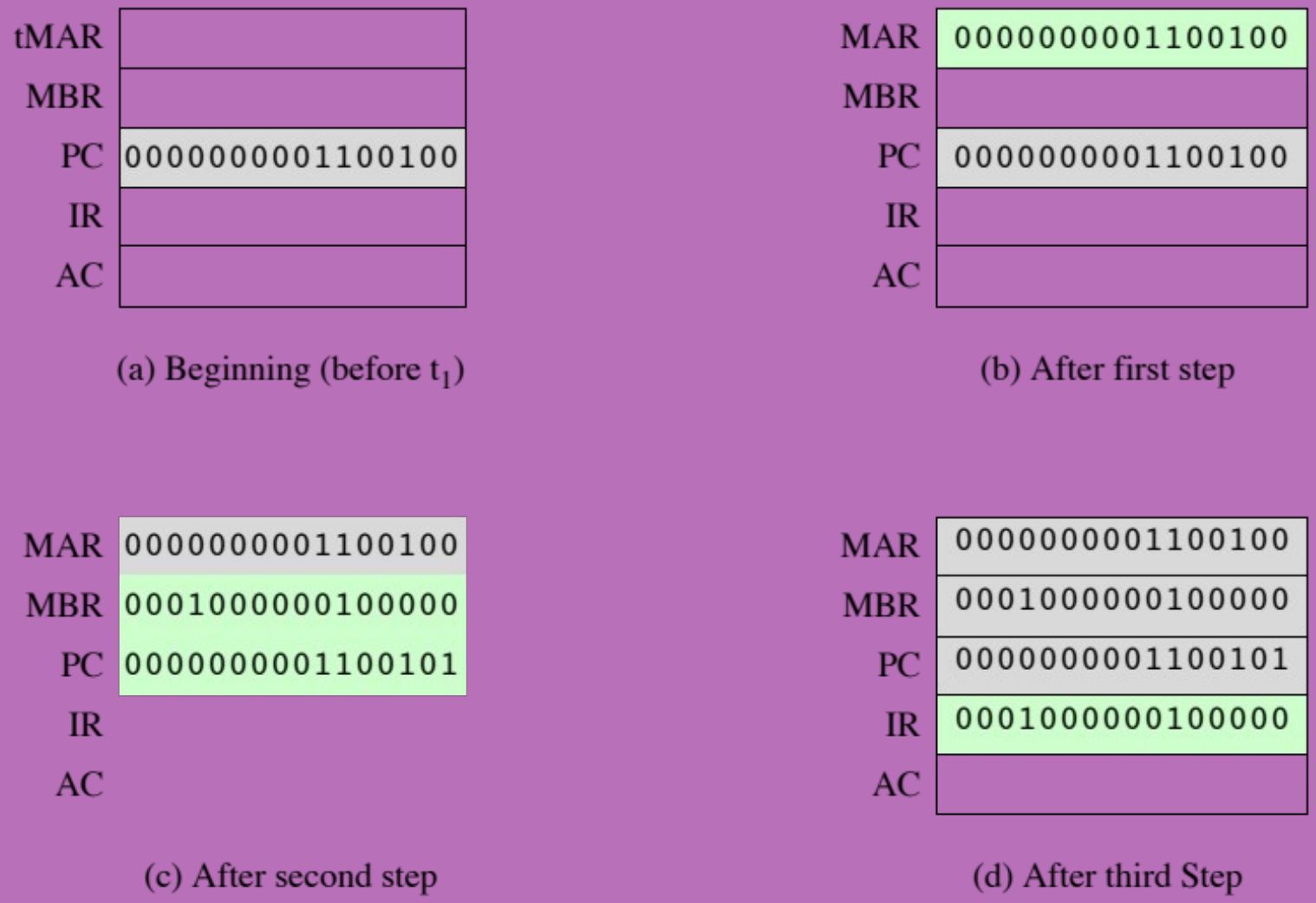
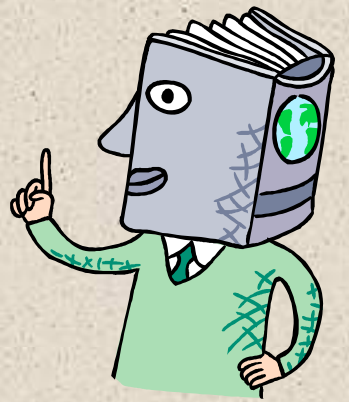


Figure 20.2 Sequence of Events, Fetch Cycle

Rules for Micro-Operations Grouping

- Proper sequence must be followed
  - MAR ← (PC) must precede MBR ← (memory)
- Conflicts must be avoided
  - Must not read and write same register at same time
  - MBR ← (memory) and IR ← (MBR) must not be in same cycle
- One of the micro-operations involves an addition
  - To avoid duplication of circuitry, this addition could be performed by the ALU
  - The use of the ALU may involve additional micro-operations, depending on the functionality of the ALU and the organization of the processor



Indirect Cycle

- Once an instruction is fetched, the next step is to fetch source operands
- Assuming a one-address instruction format, with direct and indirect addressing allowed:
  - If the instruction specifies an indirect address, then an indirect cycle must precede the execute cycle
  - The address field of the instruction is transferred to the MAR
  - This is then used to fetch the address of the operand
  - Finally, the address field of the IR is updated from the MBR, so that it now contains a direct rather than an indirect address
  - The IR is now in the same state as if indirect addressing had not been used, and it is ready for the execute cycle

Interrupt Cycle

- At the completion of the execute cycle, a test is made to determine whether any enabled interrupts have occurred, and if so, the interrupt cycle occurs
- The nature of this cycle varies greatly from one machine to another
- In a simple sequence of events:
  - In the first step the contents of the PC are transferred to the MBR so that they can be saved for return from the interrupt
  - Then the MAR is loaded with the address at which the contents of the PC are to be saved, and the PC is loaded with the address of the start of the interrupt-processing routine
    - These two actions may each be a single micro-operation
    - Because most processors provide multiple types and/or levels of interrupts, it may take one or more additional micro-operations to obtain the Save_Address and the Routine_Address before they can be transferred to the MAR and PC respectively
  - Once this is done, the final step is to store the MBR, which contains the old value of the PC, into memory
  - The processor is now ready to begin the next instruction cycle

Execute Cycle

- Because of the variety of opcodes, there are a number of different sequences of micro-operations that can occur
- Instruction decoding
  - The control unit examines the opcode and generates a sequence of micro-operations based on the value of the opcode
- A simplified add instruction:
  - ADD R1, X (which adds the contents of the location X to register R1)
    - In the first step the address portion of the IR is loaded into the MAR
    - Then the referenced memory location is read
    - Finally the contents of R1 and MBR are added by the ALU
    - Additional micro-operations may be required to extract the register reference from the IR and perhaps to stage the ALU inputs or outputs in some intermediate registers

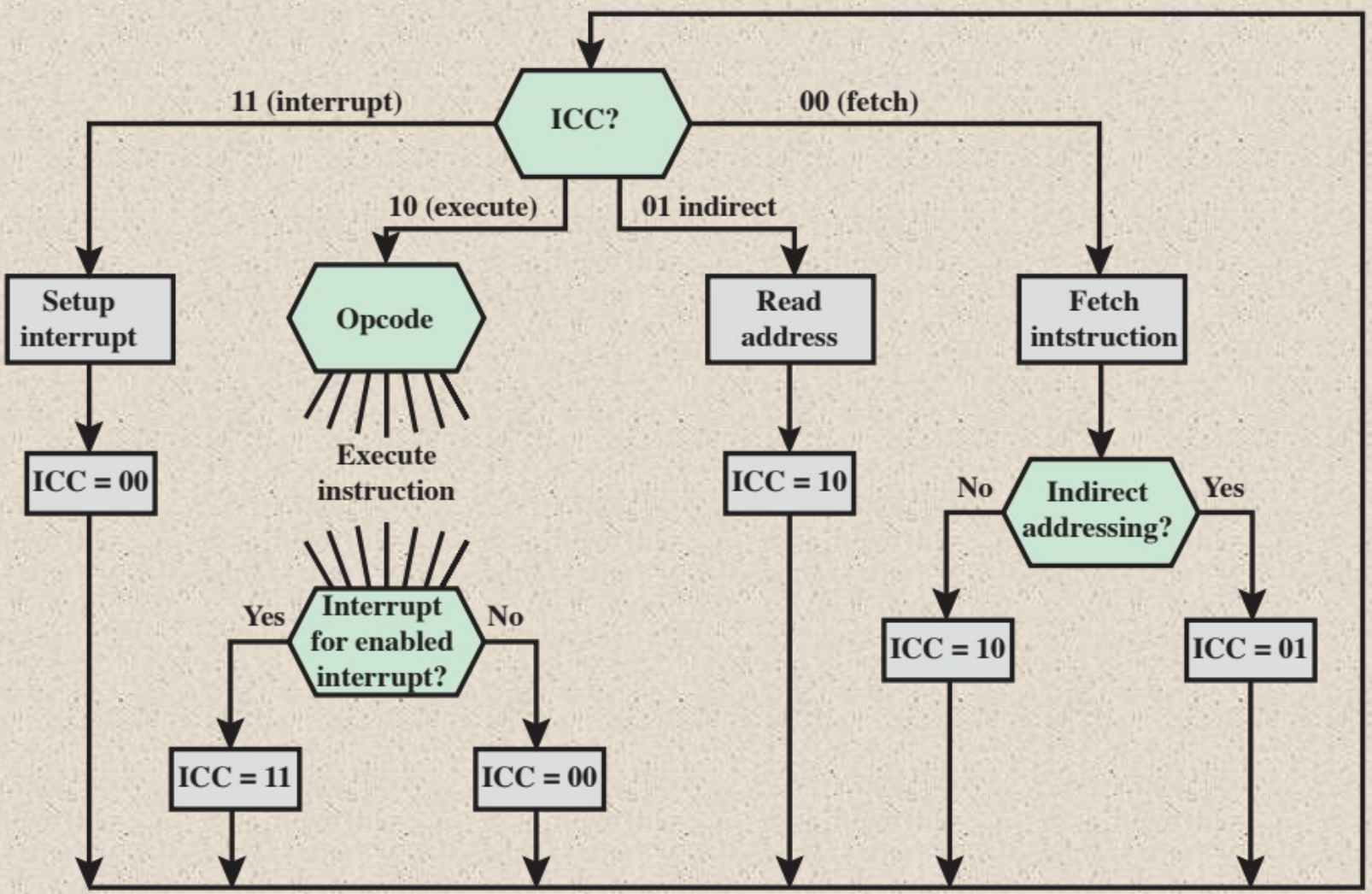


Figure 20.3 Flowchart for Instruction Cycle

Control Unit Functional Requirements

- By reducing the operation of the processor to its most fundamental level we are able to define exactly what it is that the control unit must cause to happen
- Three step process to lead to a characterization of the control unit:
  - Define basic elements of processor
  - Describe micro-operations processor performs
  - Determine the functions that the control unit must perform to cause the micro-operations to be performed
- The control unit performs two basic tasks:
  - Sequencing
  - Execution

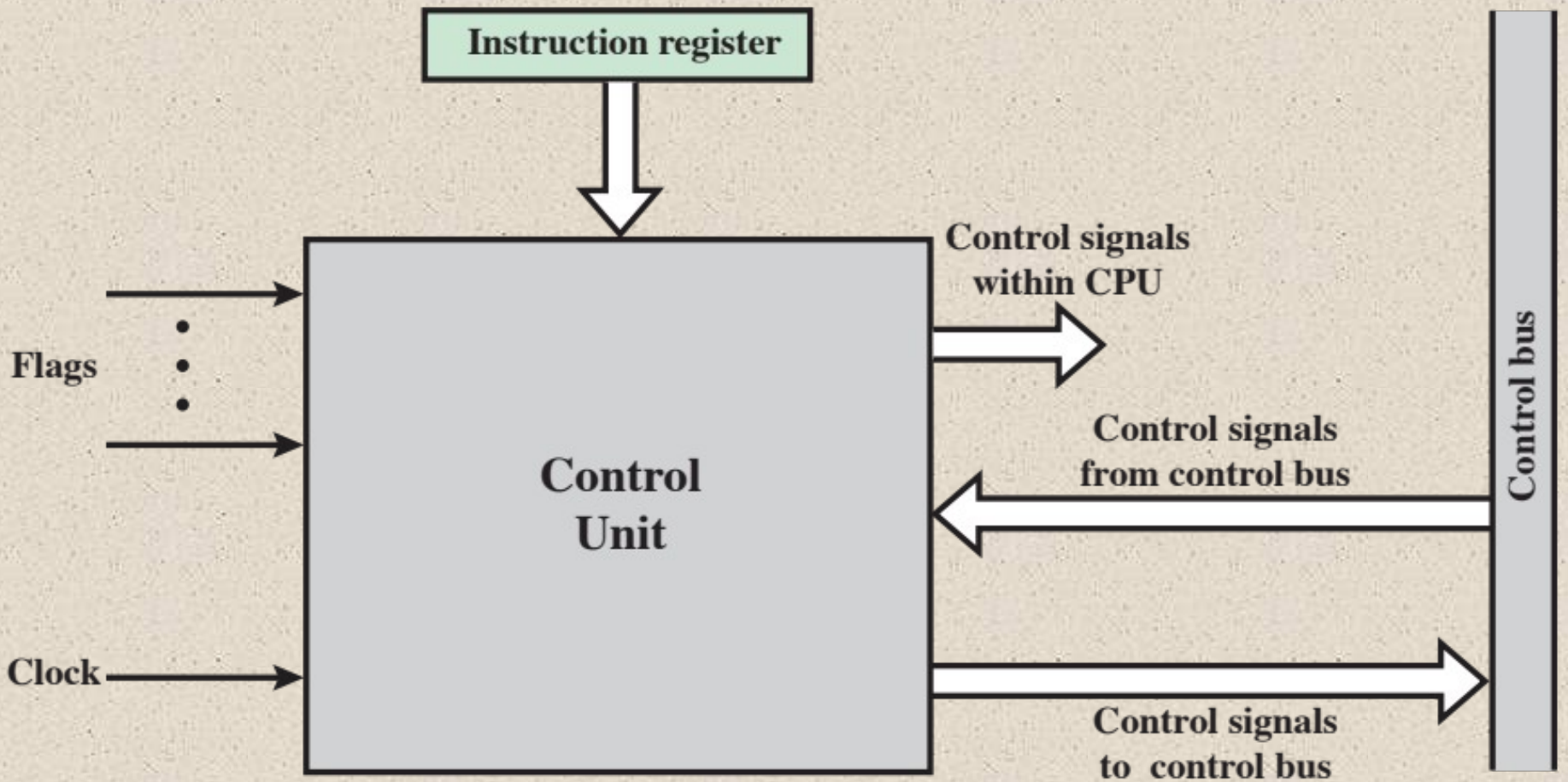


Figure 20.4 Block Diagram of the Control Unit

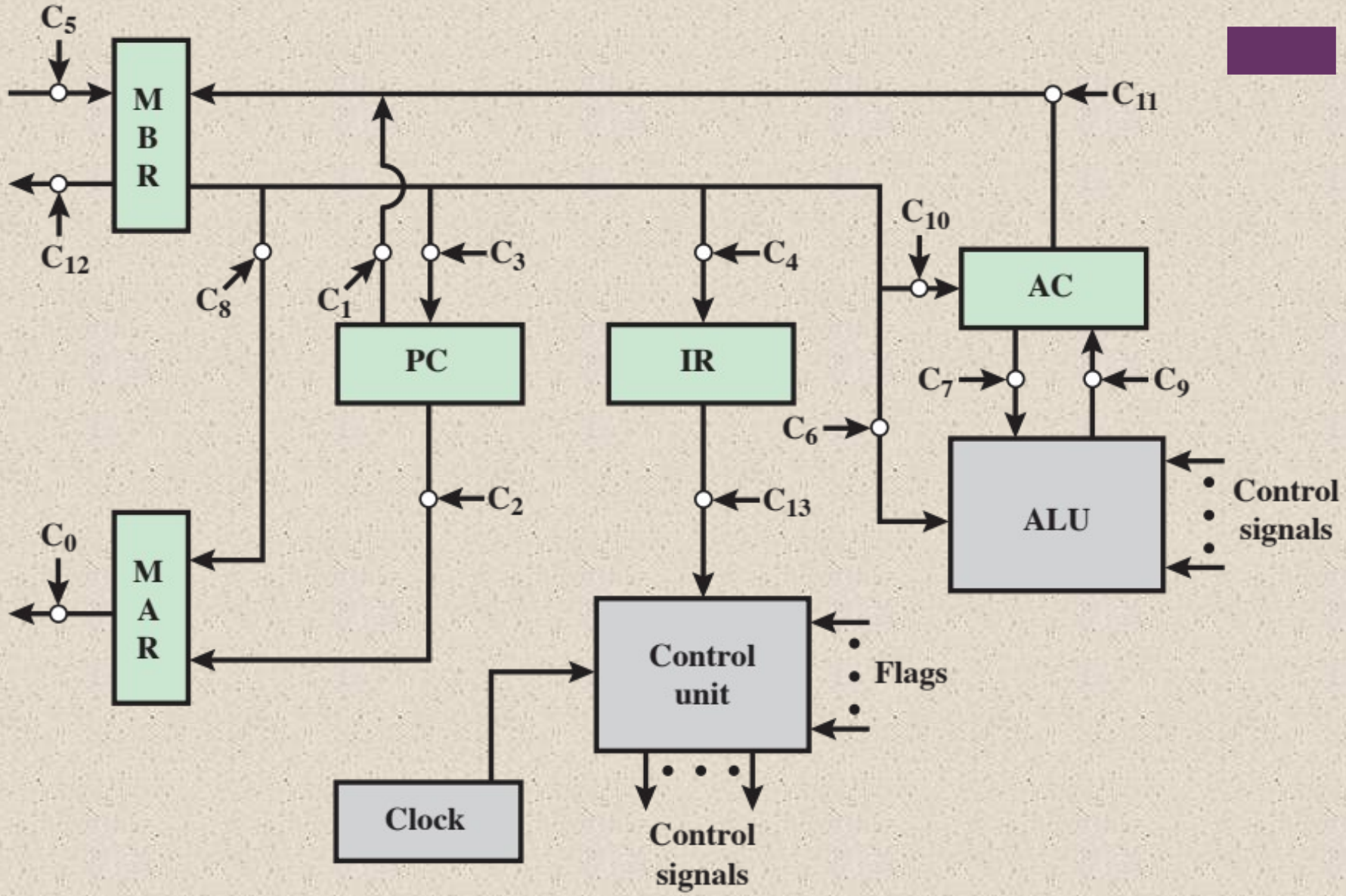


Figure 20.5 Data Paths and Control Signals

Table 20.1 Micro-operations and Control Signals

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

C_R = Read control signal to system bus.

C_W = Write control signal to system bus.

(Table can be found on page 719 in the textbook.)

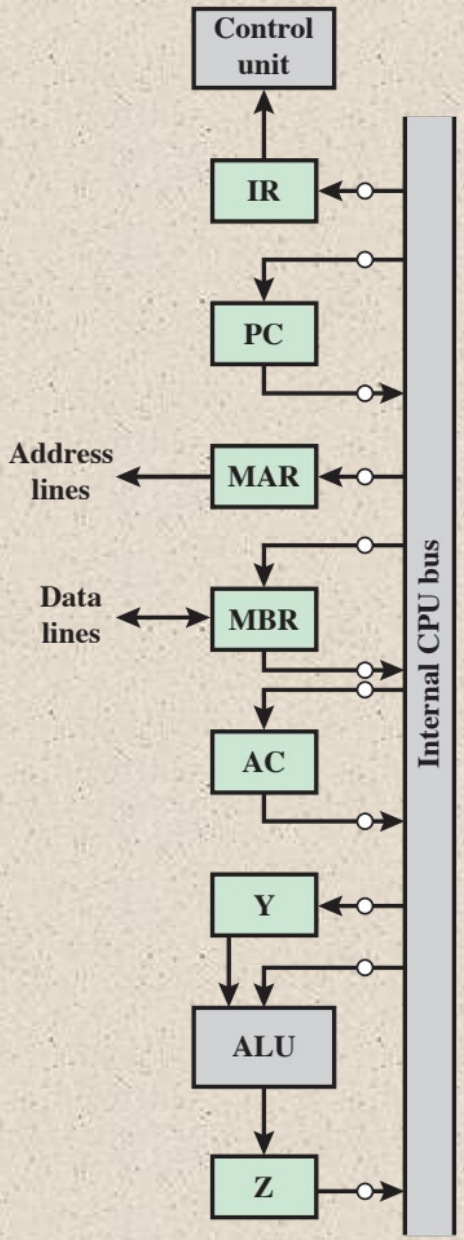


Figure 20.6 CPU with Internal Bus

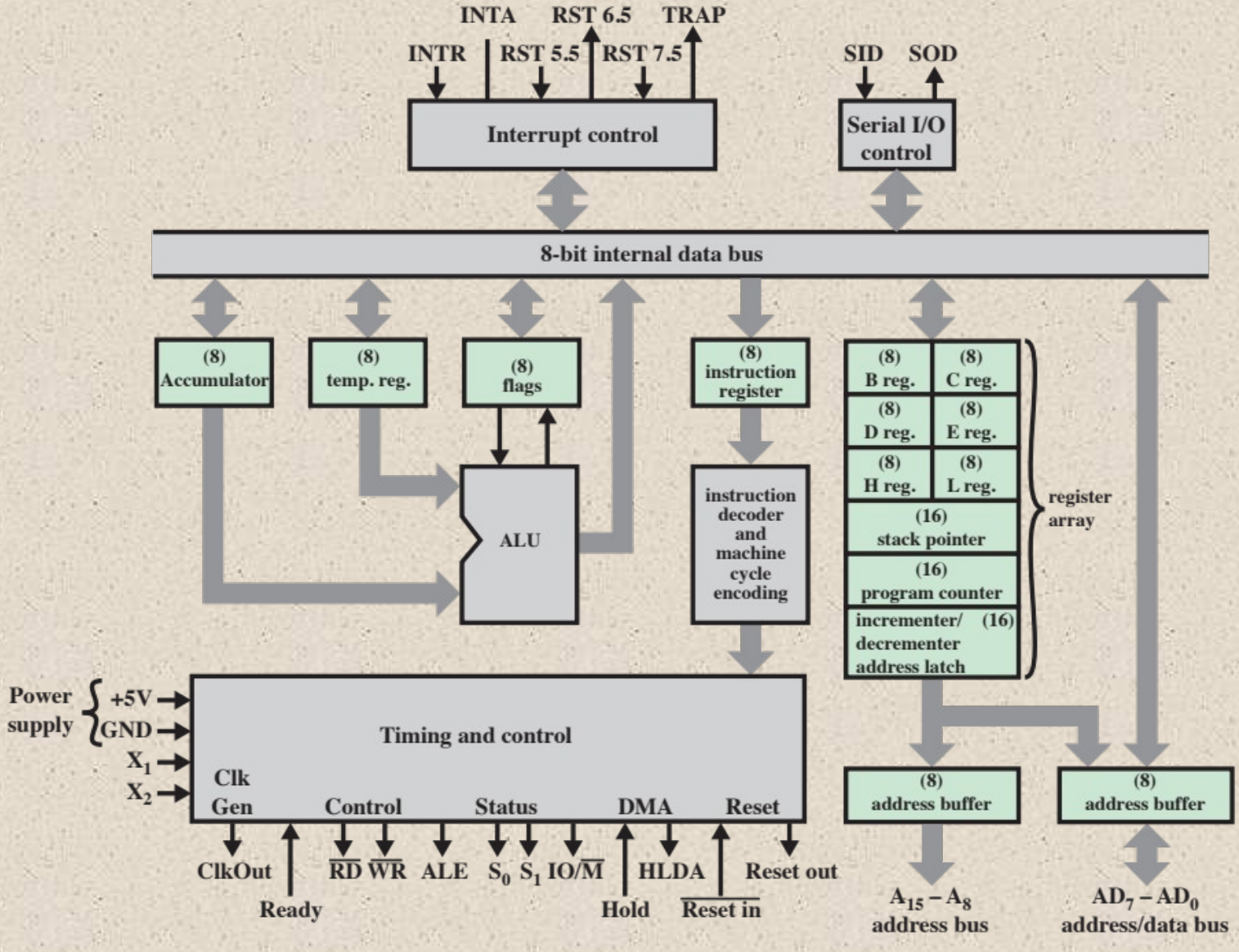


Figure 20.7 Intel 8085 CPU Block Diagram

Address and Data Signals

High Address (A15–A8)
The high-order 8 bits of a 16-bit address.

Address/Data (AD7–AD0)
The lower-order 8 bits of a 16-bit address or 8 bits of data. This multiplexing saves on pins.

Serial Input Data (SID)
A single-bit input to accommodate devices that transmit serially (one bit at a time).

Serial Output Data (SOD)
A single-bit output to accommodate devices that receive serially.

Timing and Control Signals

CLK (OUT)
The system clock. The CLK signal goes to peripheral chips and synchronizes their timing.

X1, X2
These signals come from an external crystal or other device to drive the internal clock generator.

Address Latch Enabled (ALE)
Occurs during the first clock state of a machine cycle and causes peripheral chips to store the address lines. This allows the address module (e.g., memory, I/O) to recognize that it is being addressed.

Status (S0, S1)
Control signals used to indicate whether a read or write operation is taking place.

IO/M
Used to enable either I/O or memory modules for read and write operations.

Read Control (RD)
Indicates that the selected memory or I/O module is to be read and that the data bus is available for data transfer.

Write Control (WR)
Indicates that data on the data bus is to be written into the selected memory or I/O location.

Table 20.2

Intel 8085 External Signals

(page 1 of 2)

(Table can be found on page 722 in the textbook.)

Memory and I/O Initiated Symbols

Hold

Requests the CPU to relinquish control and use of the external system bus. The CPU will complete execution of the instruction presently in the IR and then enter a hold state, during which no signals are inserted by the CPU to the control, address, or data buses. During the hold state, the bus may be used for DMA operations.

Hold Acknowledge (HOLDA)

This control unit output signal acknowledges the HOLD signal and indicates that the bus is now available.

READY

Used to synchronize the CPU with slower memory or I/O devices. When an addressed device asserts READY, the CPU may proceed with an input (DBIN) or output (WR) operation. Otherwise, the CPU enters a wait state until the device is ready.

Interrupt-Related Signals

TRAP

Restart Interrupts (RST 7.5, 6.5, 5.5)

Interrupt Request (INTR)

These five lines are used by an external device to interrupt the CPU. The CPU will not honor the request if it is in the hold state or if the interrupt is disabled. An interrupt is honored only at the completion of an instruction. The interrupts are in descending order of priority.

Interrupt Acknowledge

Acknowledges an interrupt.

CPU Initialization

RESET IN

Causes the contents of the PC to be set to zero. The CPU resumes execution at location zero.

RESET OUT

Acknowledges that the CPU has been reset. The signal can be used to reset the rest of the system.

Voltage and Ground

VCC

+5-volt power supply

VSS

Electrical ground

Table 20.2

Intel 8085 External Signals

(page 2 of 2)

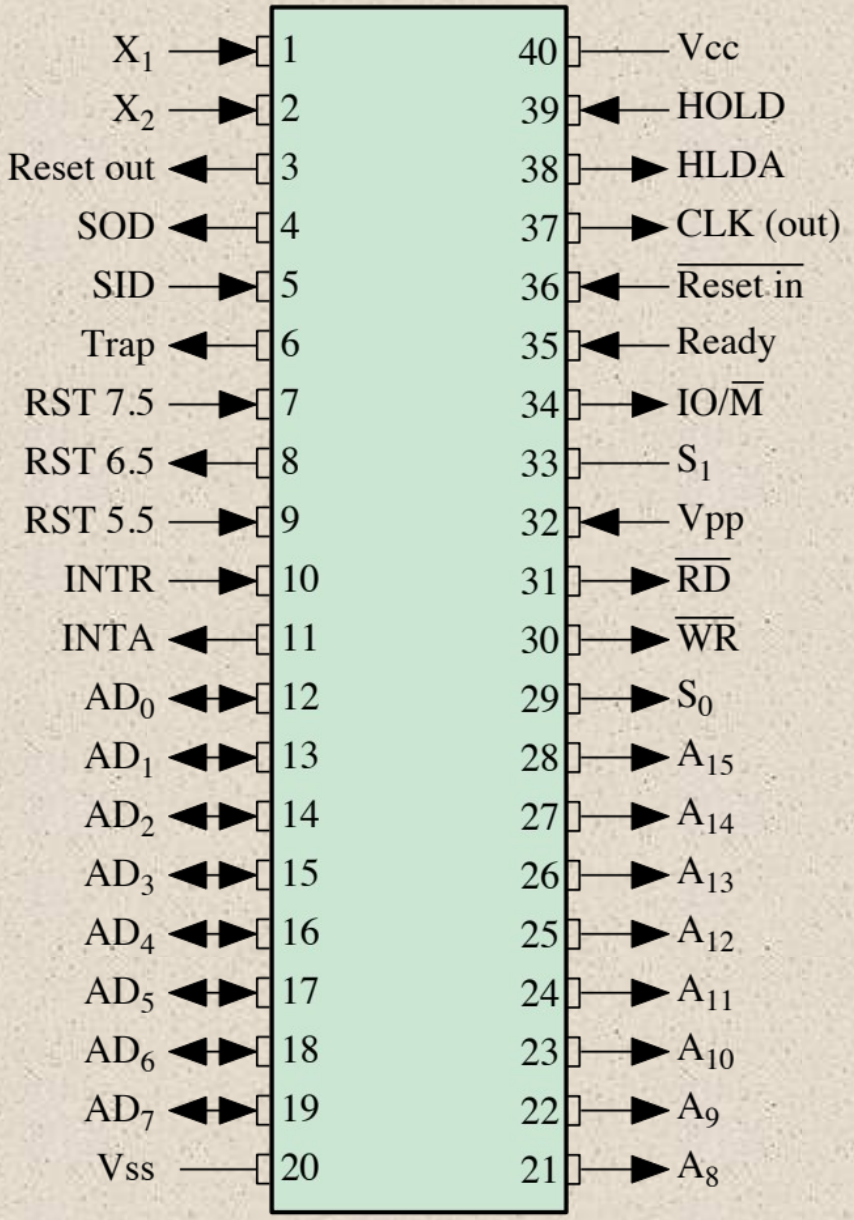


Figure 20.8 Intel 8085 Pin Configuration

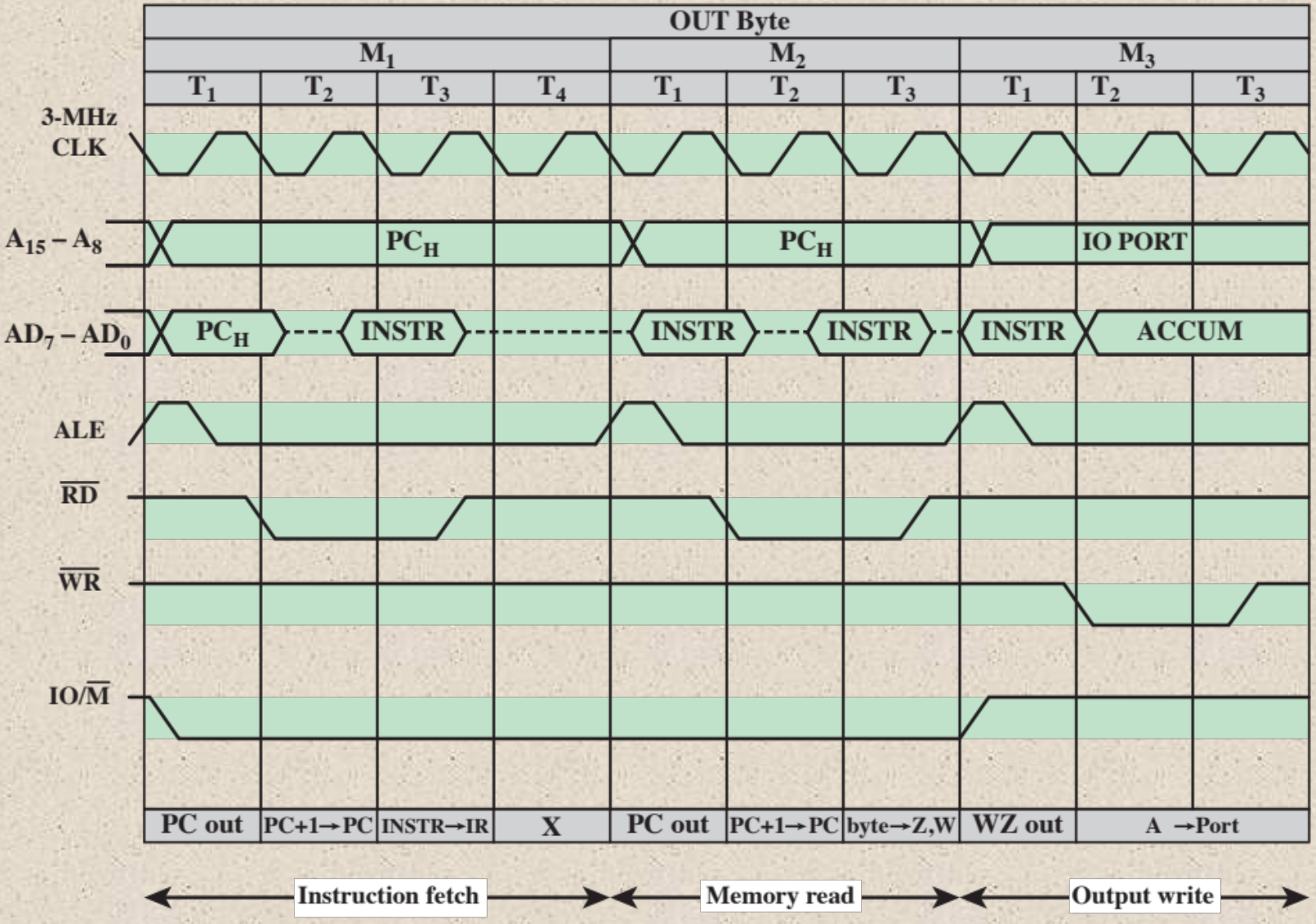


Figure 20.9 Timing Diagram for Intel 8085 OUT Instruction

I1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
I2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
I3	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
I4	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
O1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
O2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
O3	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
O4	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
O5	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
O6	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
O7	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
O8	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
O9	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
O10	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
O11	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
O12	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
O13	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
O14	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
O15	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
O16	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 20.3

A Decoder With Four Inputs and Sixteen Outputs

(Table can be found on page 725 in the

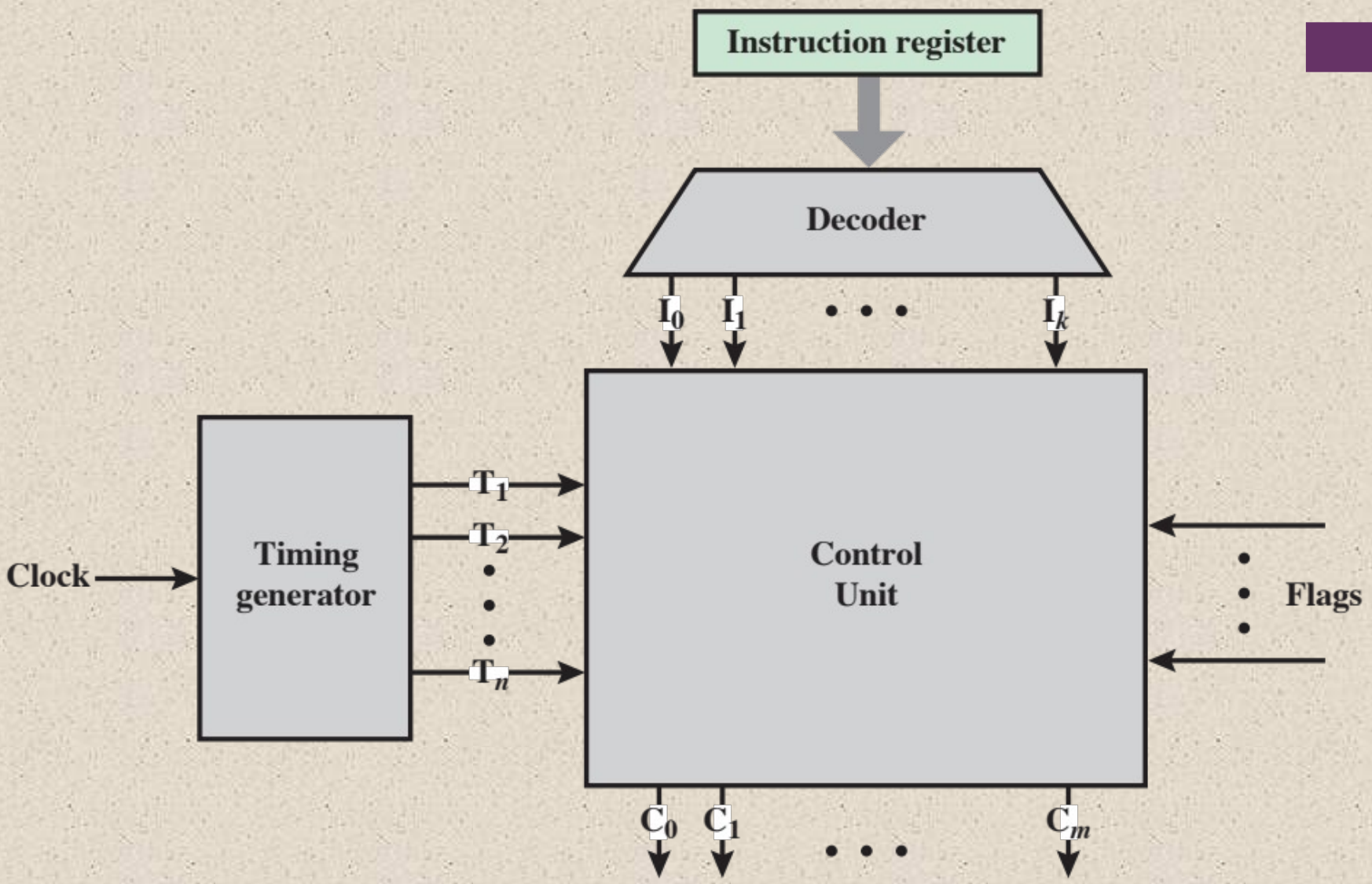


Figure 20.10 Control Unit with Decoded Inputs

Summary

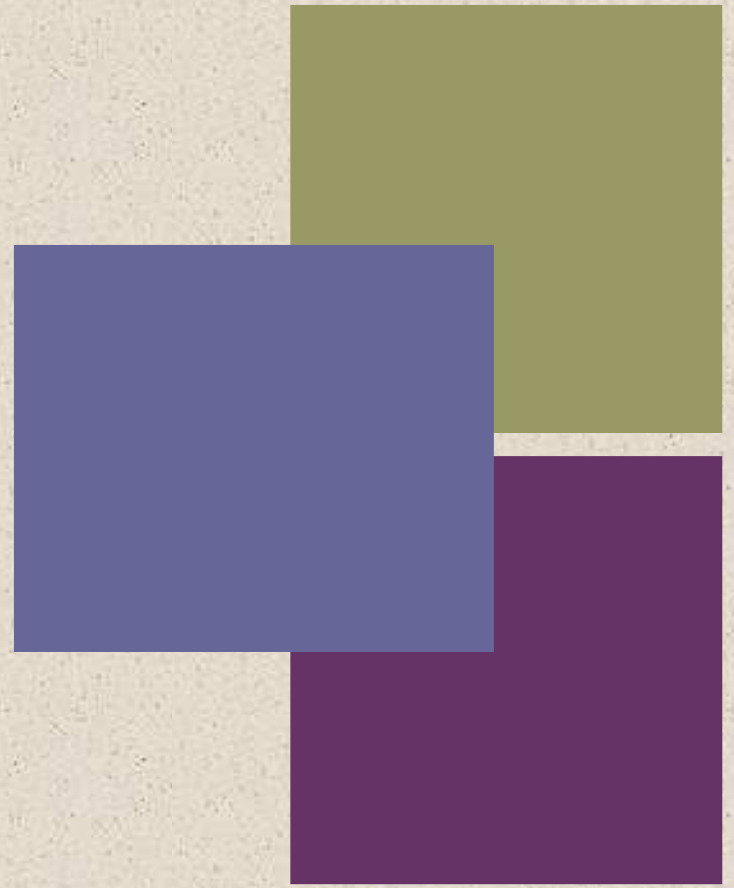
Chapter 20

- Micro-operations
  - The fetch cycle
  - The indirect cycle
  - The interrupt cycle
  - The execute cycle
  - The instruction cycle

Control Unit Operation

- Control of the processor
  - Functional requirements
  - Control signals
  - Internal processor organization
  - The Intel 8085
- Hardwired implementation
  - Control unit inputs
  - Control unit logic

William Stallings
Computer Organization and Architecture
10th Edition



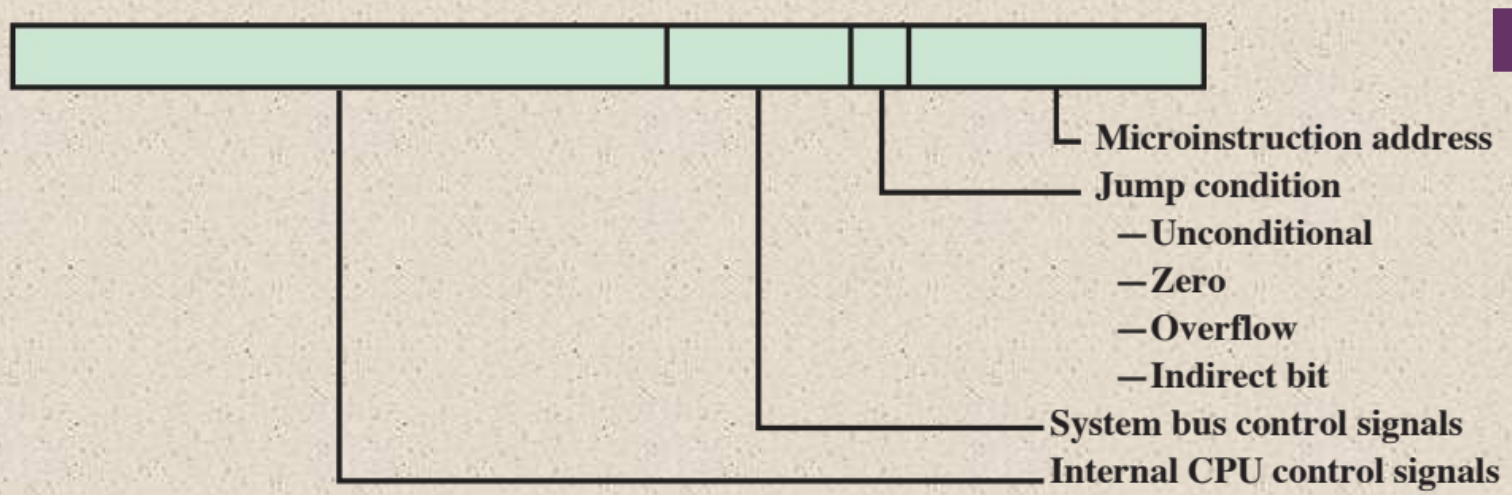
Chapter 21

Microprogrammed Control

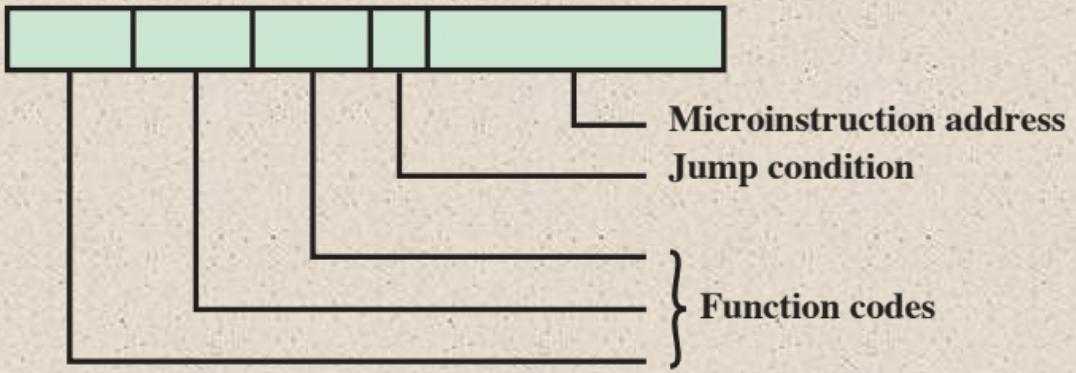
Table 21.1
Machine Instruction Set for Wilkes Example

Order	Effect of Order
A n	$C(Acc) + C(n)$ to Acc_1
S n	$C(Acc) - C(n)$ to Acc_1
H n	$C(n)$ to Acc_2
V n	$C(Acc2) \times C(n)$ to Acc, where $C(n) \geq 0$
T n	$C(Acc_1)$ to n, 0 to Acc
U n	$C(Acc_1)$ to n
R n	$C(Acc) \times 2^{-(n+1)}$ to Acc
L n	$C(Acc) \times 2^{n+1}$ to Acc
G n	IF $C(Acc) < 0$, transfer control to n; if $C(Acc) \geq 0$, ignore (i.e., proceed serially)
I n	Read next character on input mechanism into n
O n	Send $C(n)$ to output mechanism

Notation:
- Acc = accumulator
- Acc_1 = most significant half of accumulator
- Acc_2 = least significant half of accumulator
- n = storage location n
- $C(X)$ = contents of X (X = register or storage location)



(a) Horizontal microinstruction



(b) Vertical microinstruction

Figure 21.1 Typical Microinstruction Formats

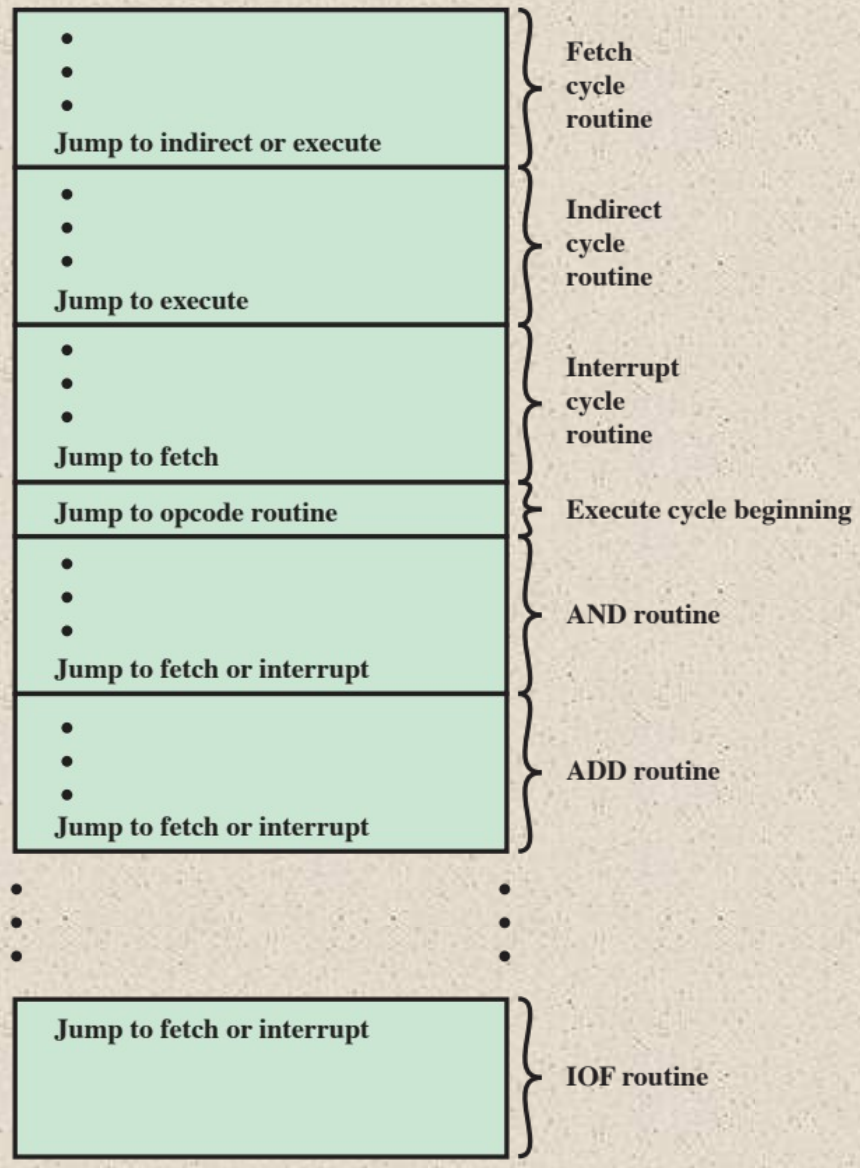


Figure 21.2 Organization of Control Memory

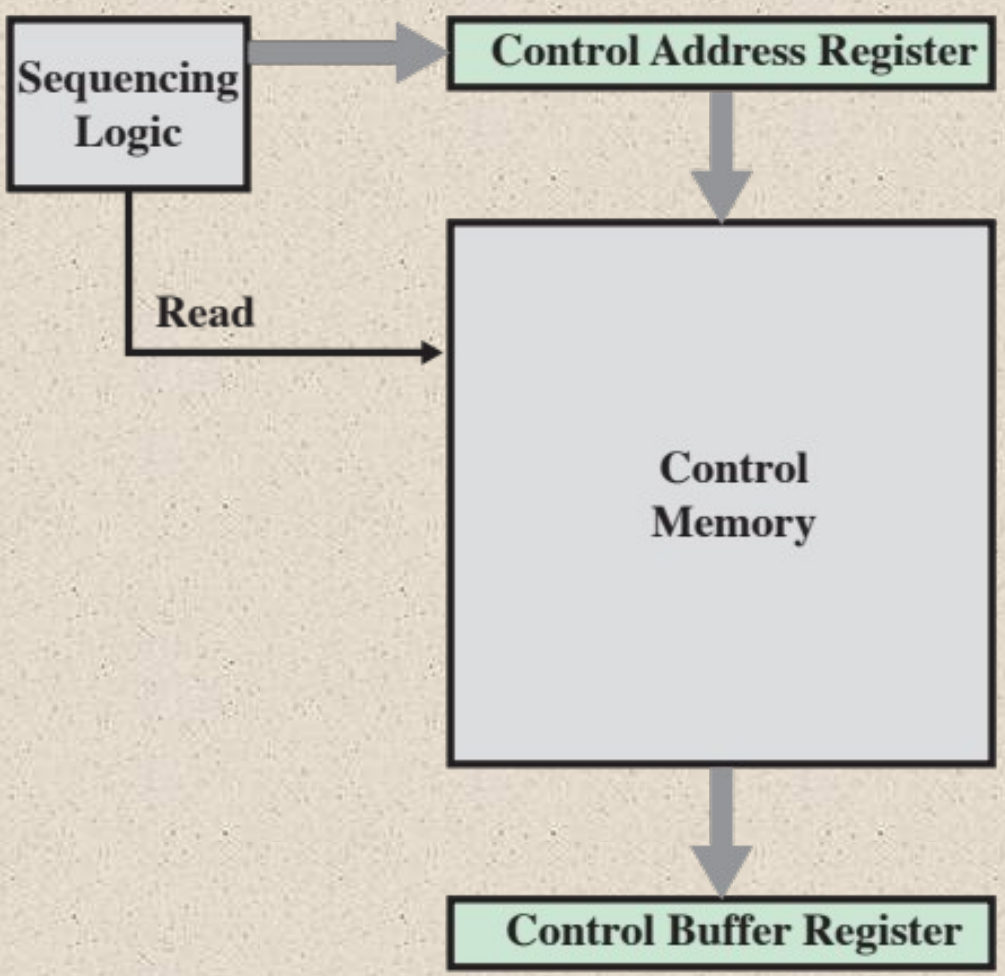


Figure 21.3 Control Unit Microarchitecture

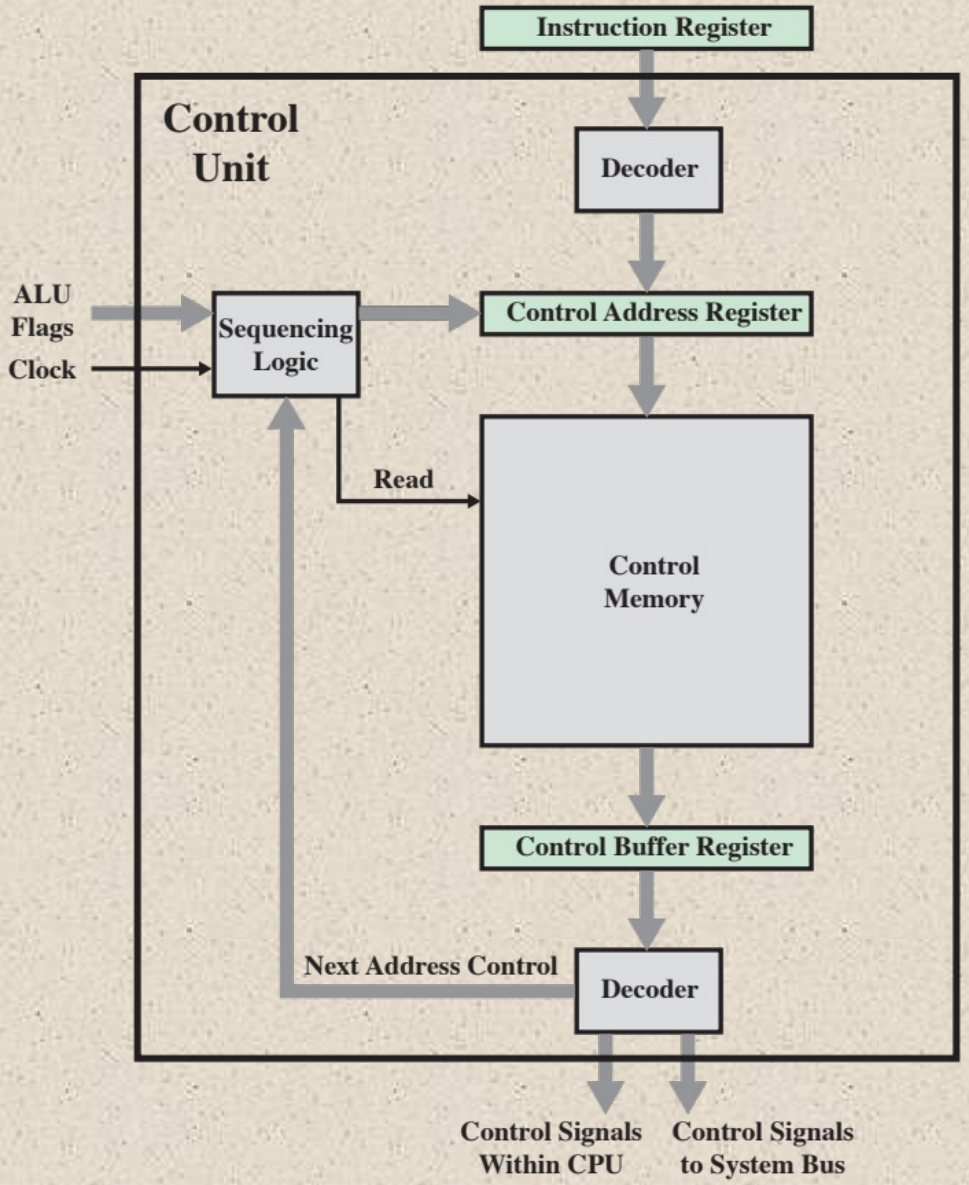


Figure 21.4 Functioning of Microprogrammed Control Unit

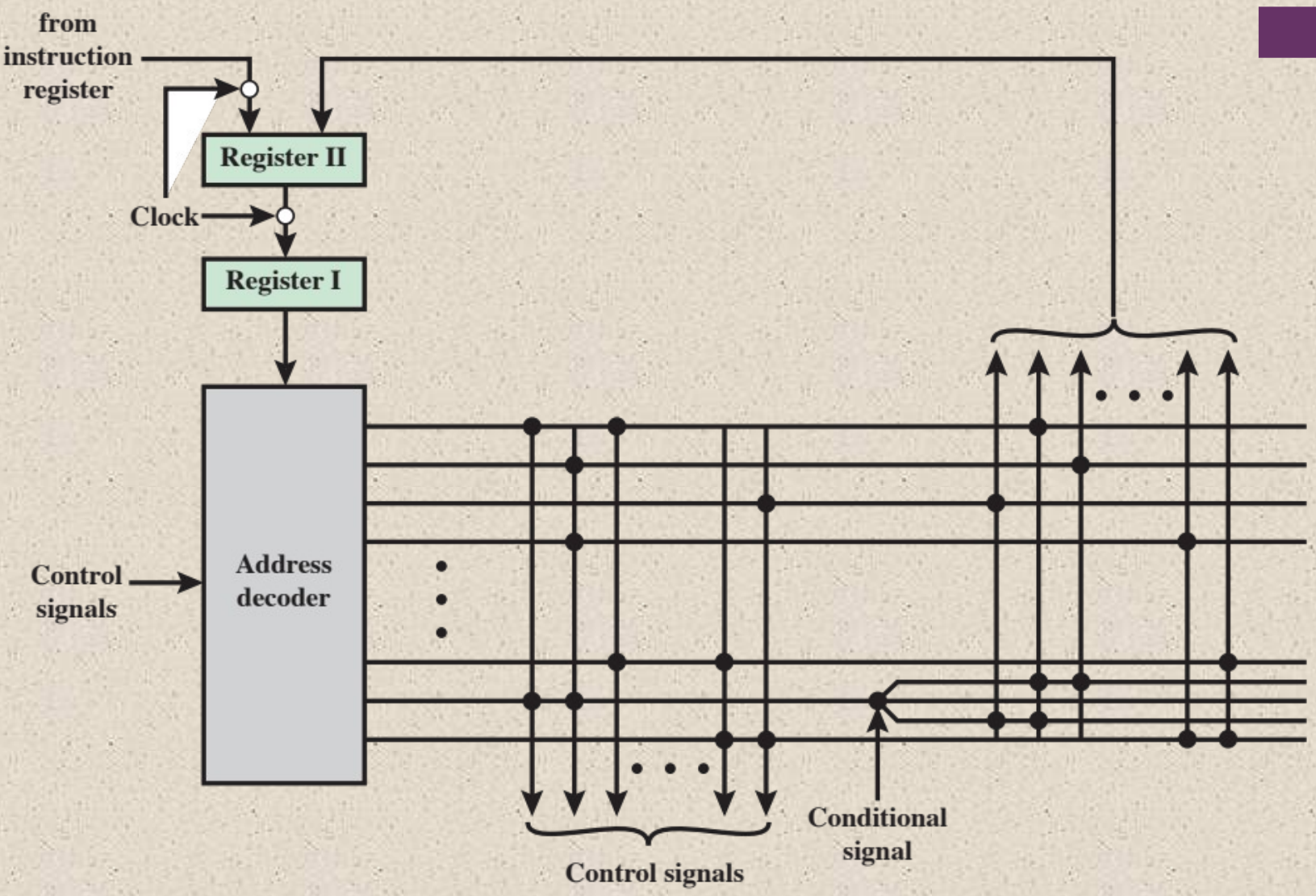


Figure 21.5 Wilkes's Microprogrammed Control Unit

Table 21.2 Microinstructions for Wilkes Example (page 1 of 2)

Notation: $A, B, C, \ldots$ stand for the various registers in the arithmetical and control register units. *C to D* indicates that the switching circuits connect the output of register *C* to the input register *D*; *(D + A)* to *C* indicates that the output register of *A* is connected to the one input of the adding unit (the output of *D* is permanently connected to the other input), and the output of the adder to register *C*. A numerical symbol *n* in quotes (e.g., '*n*') stands for the source whose output is the number *n* in units of the least significant digit.

		Arithmetical Unit	Control Register Unit	Conditional Flip-Flop		Next Micro-instruction	
				Set	Use	0	1
	0		*F* to *G* and *E*			1	
	1		(*G* to '1') to *F*			2	
	2		Store to *G*			3	
	3		*G* to *E*			4	
	4		*E* to decoder			—	
A	5	*C* to *D*				16	
S	6	*C* to *D*				17	
H	7	Store to *B*				0	
V	8	Store to *A*				27	
T	9	*C* to Store				25	
U	10	*C* to Store				0	
R	11	*B* to *D*	*E* to *G*			19	
L	12	*C* to *D*	*E* to *G*			22	
G	13		*E* to *G*	(1)C_5		18	
I	14	Input to Store				0	
O	15	Store to Output				0	
	16	(*D* + Store) to *C*				0	
	17	(*D* − Store) to *C*				0	
	18				1	0	1
	19	*D* to *B* (*R*)*	(*G* − '1') to *E*			20	
	20	*C* to *D*		(1)E_5		21	

(Table can be found on page 738 in the

Table 21.2 Microinstructions for Wilkes Example (page 2 of 2)

21	*D* to *C* (*R*)			1	11	0
22	*D* to *C* (*L*)†	(*G* – '1') to *E*			23	
23	*B* to *D*		(1)E_5		24	
24	*D* to *B* (*L*)			1	12	0
25	'*0*' to *B*				26	
26	*B* to *C*				0	
27	'0' to *C*	'18' to *E*			28	
28	*B* to *D*	*E* to *G*	(1)B_1		29	
29	*D* to *B* (*R*)	(*G* – '1') to *E*			30	
30	*C* to *D* (*R*)		(2)E_5	1	31	32
31	*D* to *C*			2	28	33
32	(*D* + *A*) to *C*			2	28	33
33	*B* to *D*		(1)B_1		34	
34	*D* to *B* (*R*)				35	
35	*C* to *D* (*R*)			1	36	37
36	*D* to *C*				0	
37	(*D* – *A*) to *C*				0	

*Right shift. The switching circuits in the arithmetic unit are arranged so that the least significant digit of the register *C* is placed in the most significant place of register *B* during right shift micro-operations, and the most significant digit of register *C* (sign digit) is repeated (thus making the correction for negative numbers).

†Left shift. The switching circuits are similarly arranged to pass the most significant digit of register *B* to the least significant place of register *C* during left shift micro-operations.

Microinstruction Sequencing

- The two basic tasks performed by a microprogrammed control unit are:
  - Microinstruction sequencing
    - Get the next microinstruction from the control memory
  - Microinstruction execution
    - Generate the control signals needed to execute the microinstruction
    - In designing a control unit, these tasks must be considered together because both affect the format of the microinstruction and the timing of the control unit

Design Considerations

Two concerns are involved in the design of a microinstruction sequencing technique:

The size of the microinstruction

- Minimizing the size of the control memory reduces the cost of that component

The address-generation time

- Execute microinstruction as fast as possible

In executing a microprogram the address of the next microinstruction to be executed is in one of these categories:

Determined by instruction register

- Occurs only once per instruction cycle, just after an instruction is fetched

Next sequential address

- Most common in most designs

Branch

- Are a necessary part of a microprogram

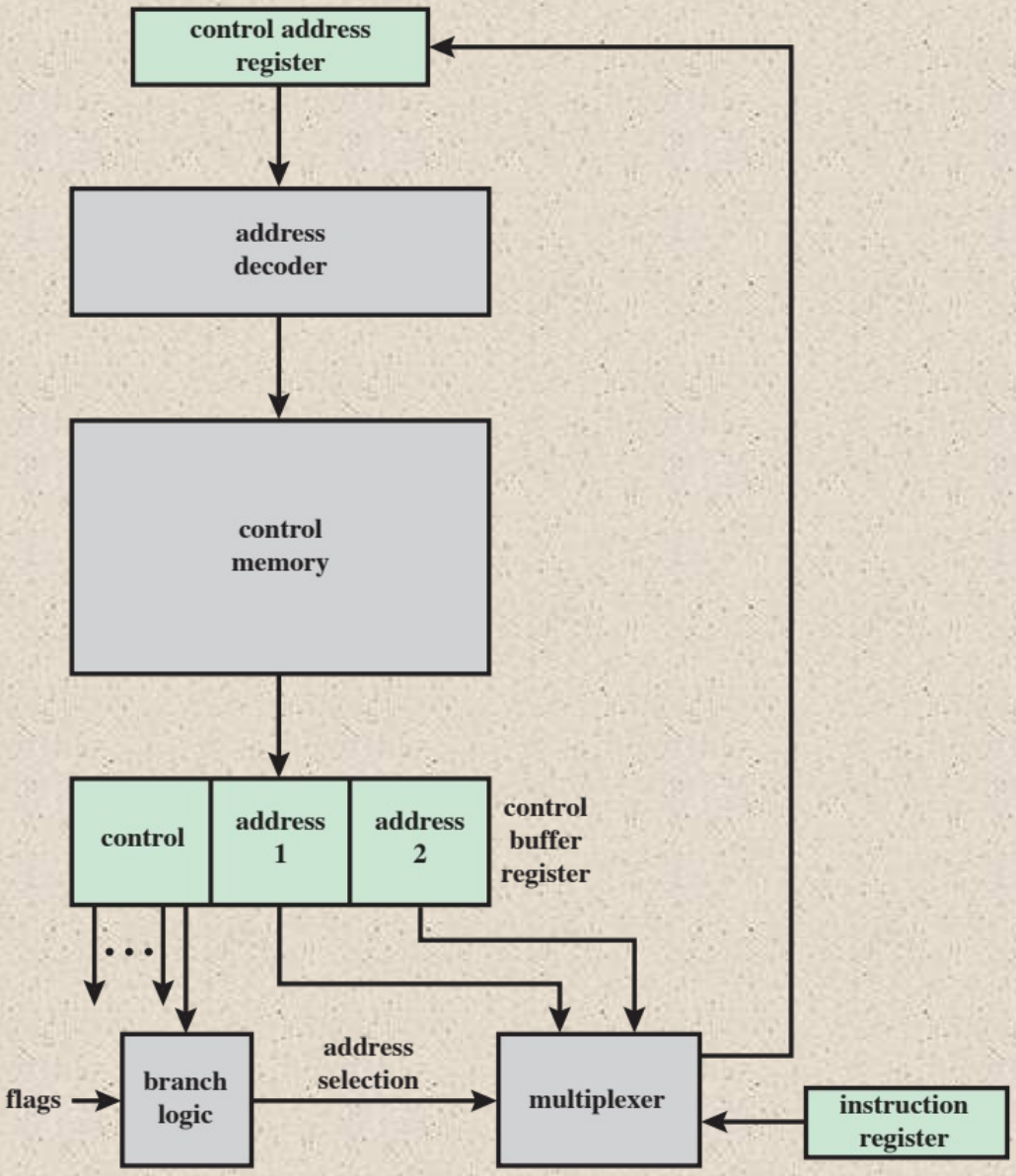


Figure 21.6 Branch Control Logic: Two Address Fields

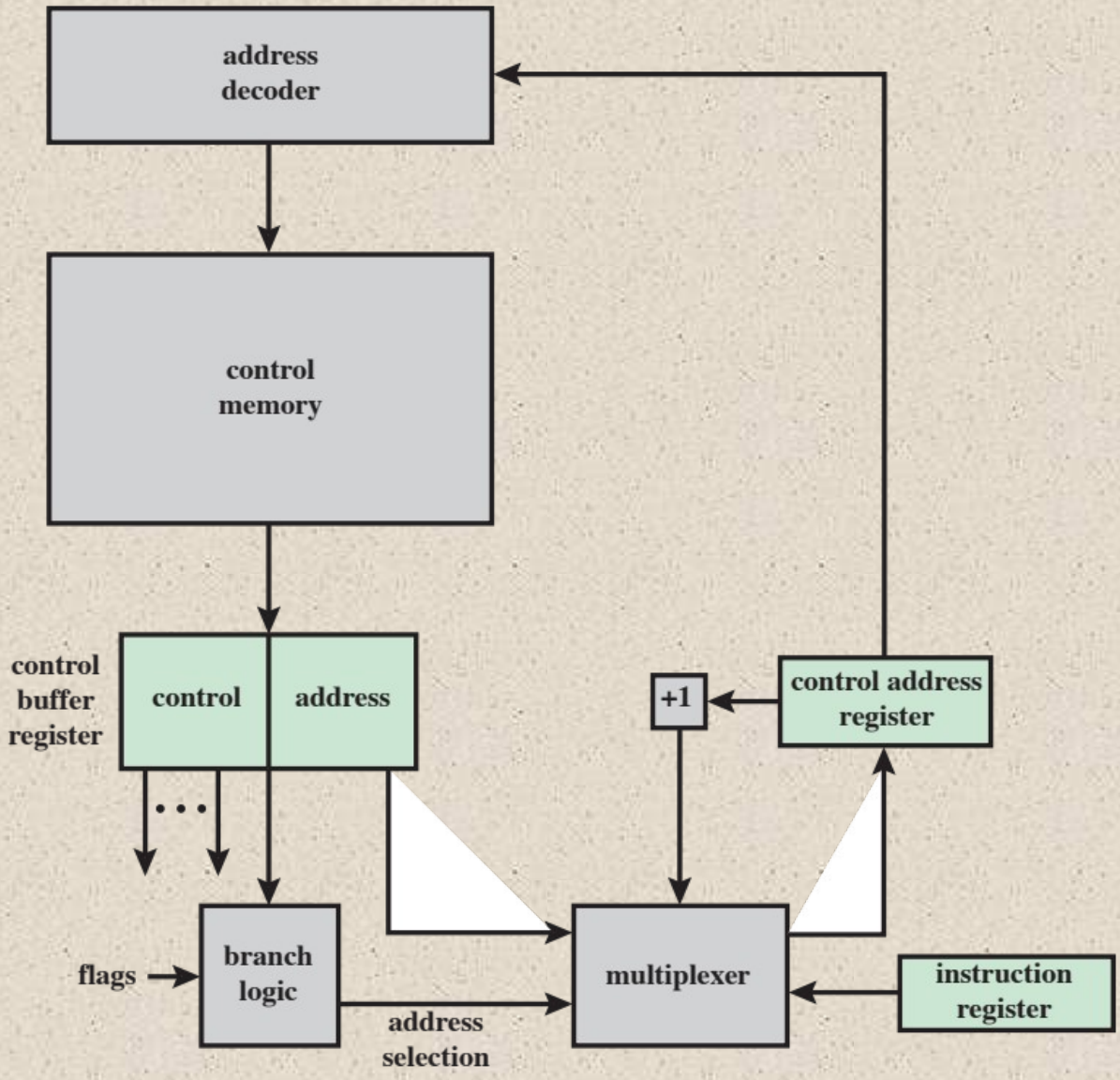


Figure 21.7 Branch Control Logic: Single Address Field

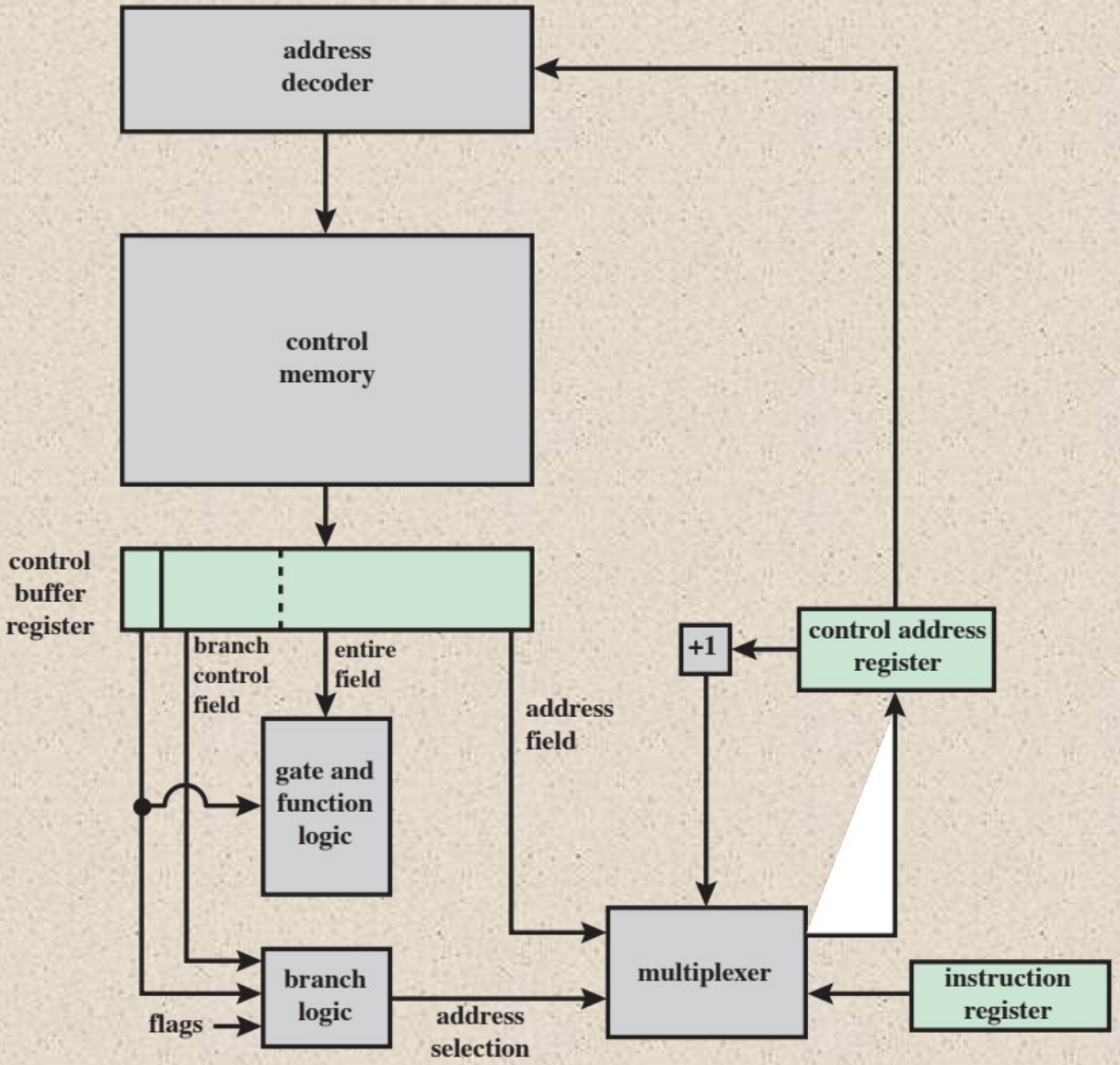


Figure 21.8 Branch Control Logic: Variable Format

Table 21.3
Microinstruction Address Generation Techniques

Explicit	Implicit
Two-field	Mapping
Unconditional branch	Addition
Conditional branch	Residual control

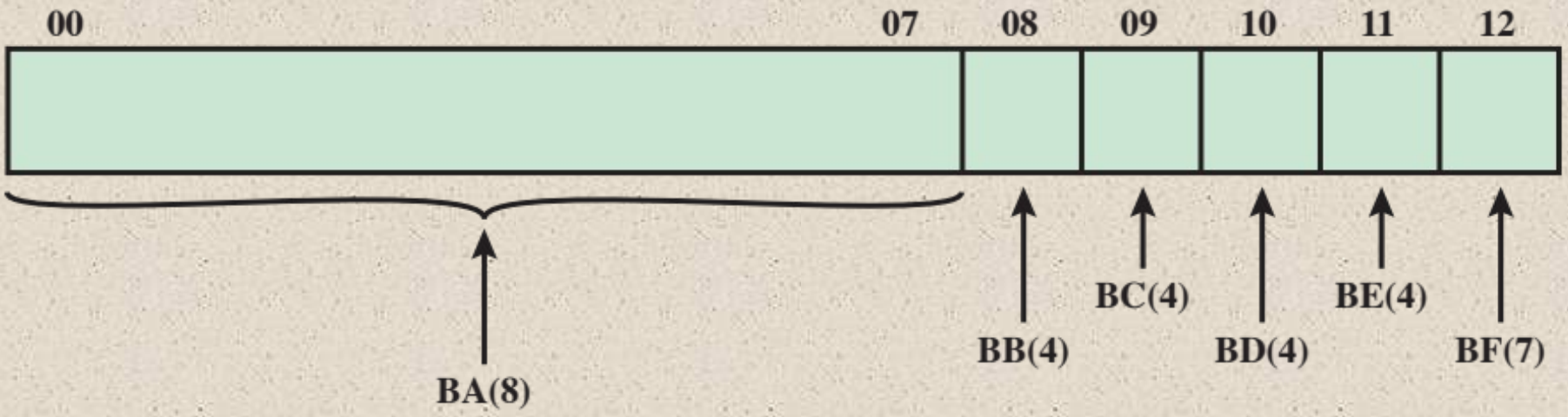


Figure 21.9 IBM 3033 Control Address Register

LSI-11 Microinstruction Sequencing

- LSI-11 is a microcomputer version of a PDP-11, with the main components of the system residing on a single board
- The LSI-11 is implemented using a microprogrammed control unit
- Makes use of a 22-bit microinstruction and a control memory of 2K 22-bit words
- The next microinstruction address is determined in one of five ways:
  - Next sequential address
  - Opcode mapping
  - Subroutine facility
  - Interrupt testing
  - Branch

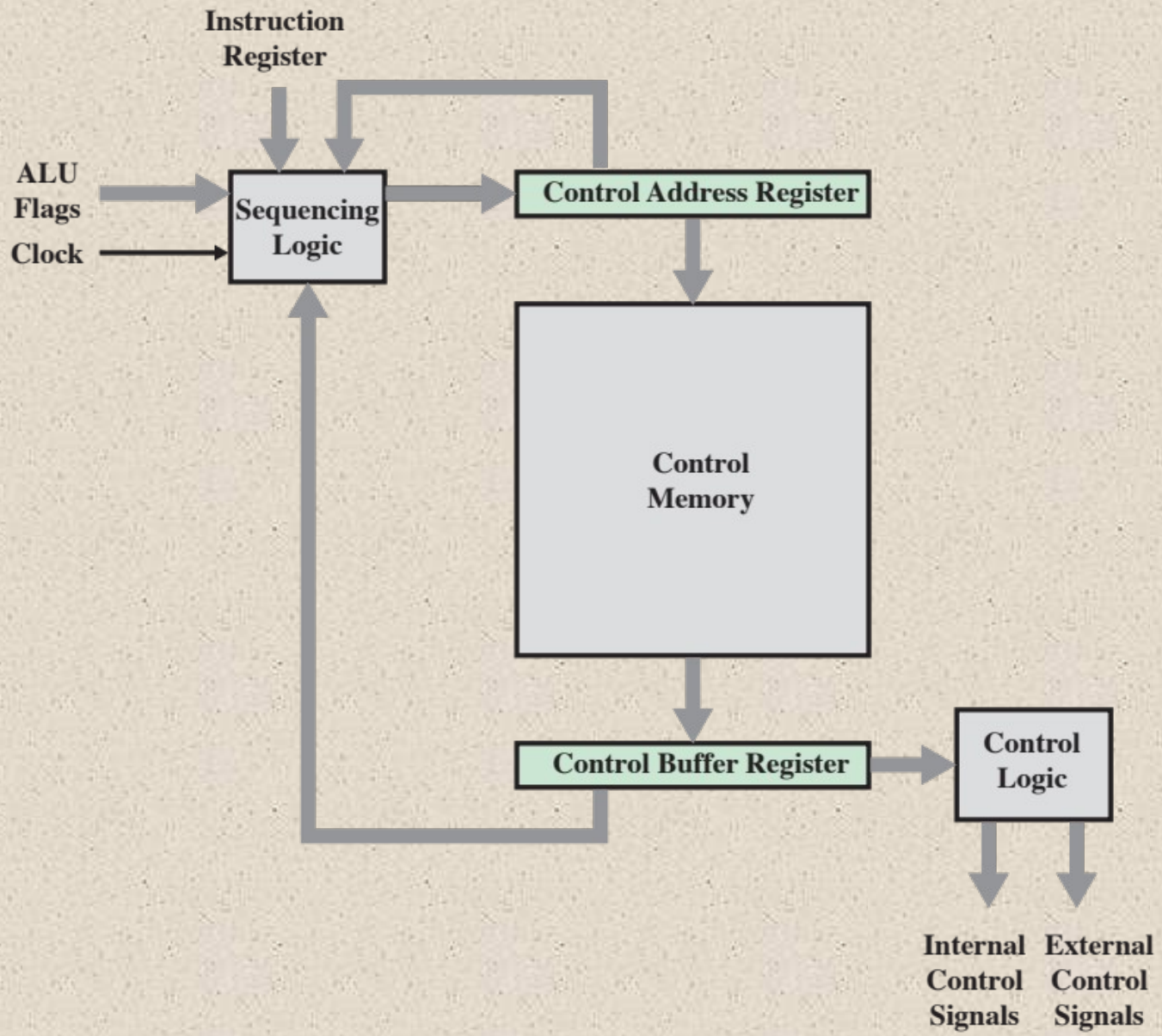
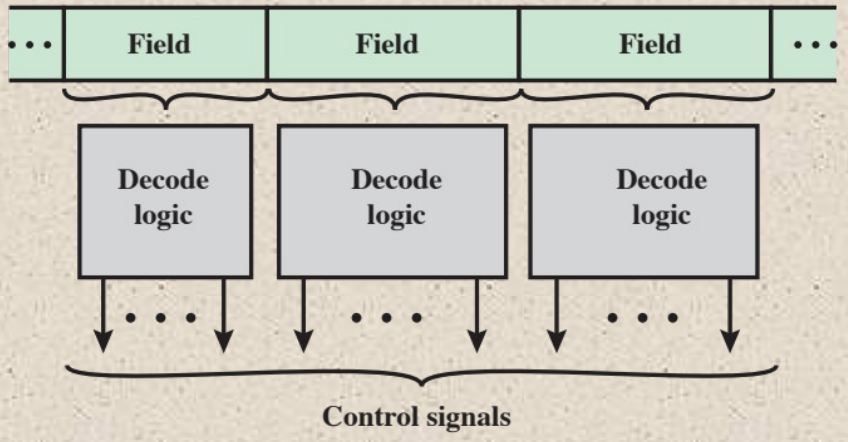


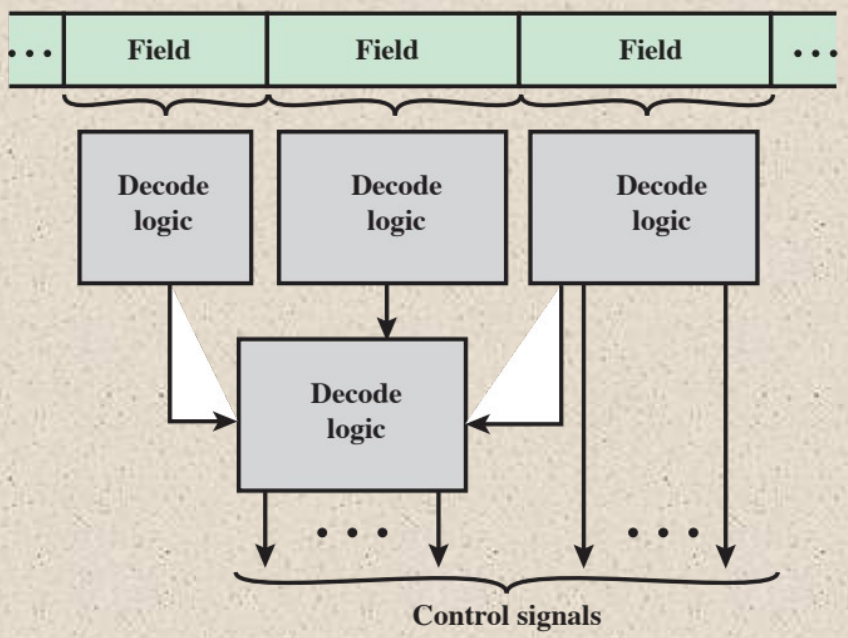
Figure 21.10 Control Unit Organization

Table 21.4
The Microinstruction Spectrum

Characteristics	
Unencoded	Highly encoded
Many bits	Few bits
Detailed view of hardware	Aggregated view of hardware
Difficult to program	Easy to program
Concurrency fully exploited	Concurrency not fully exploited
Little or no control logic	Complex control logic
Fast execution	Slow execution
Optimize performance	Optimize programming
Terminology	
Unpacked	Packed
Horizontal	Vertical
Hard	Soft

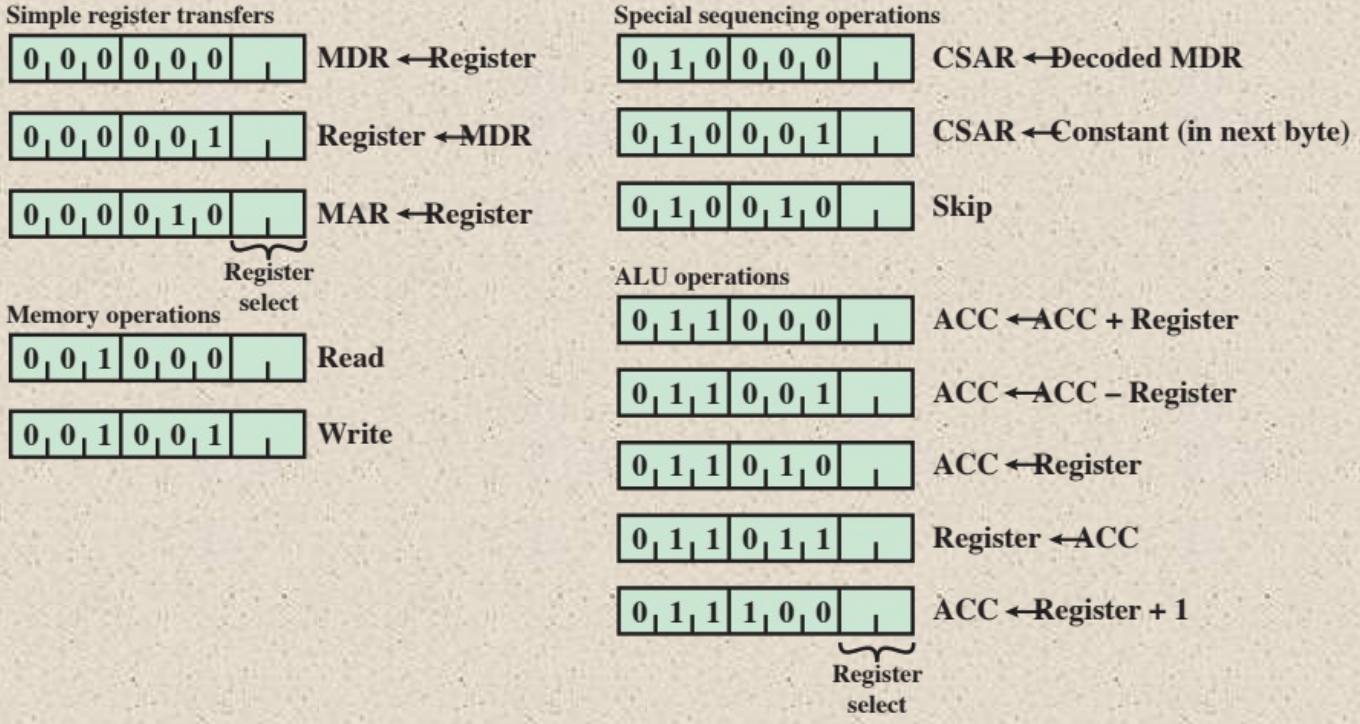


(a) Direct encoding

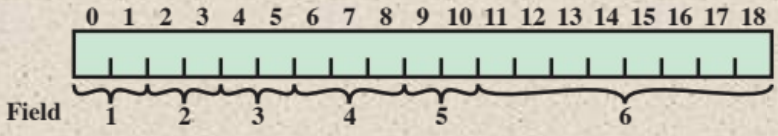


(b) Indirect encoding

Figure 21.11 Microinstruction Encoding



(a) Vertical microinstruction format



Field definition

1 - register transfer	4 - ALU operation
2 - memory operation	5 - register selection
3 - sequencing operation	6 - Constant

(b) Horizontal microinstruction format

Figure 21.12 Alternative Microinstruction Formats for a Simple Machine

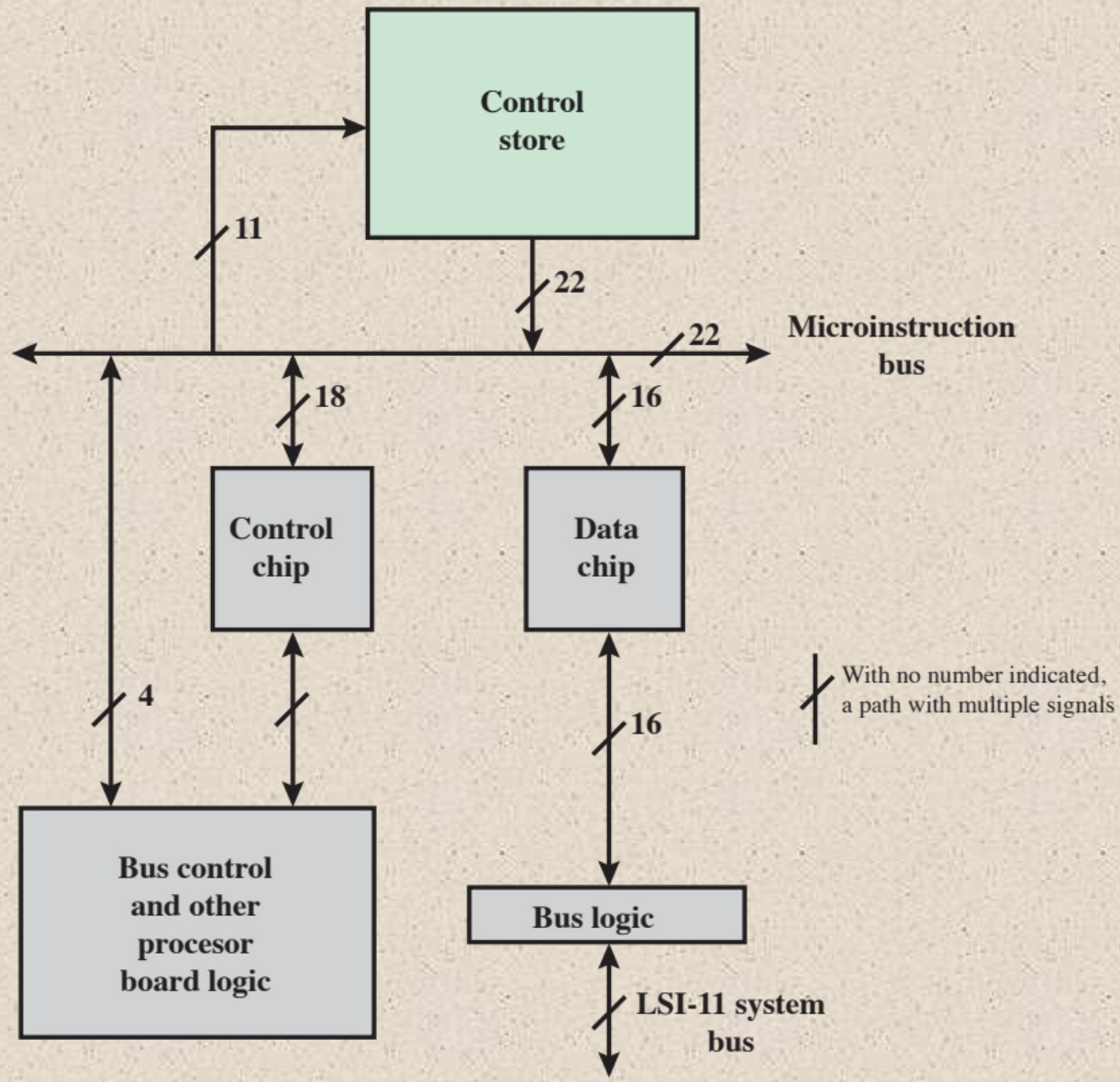


Figure 21.13 Simplified Block Diagram of the LSI-11 Processor

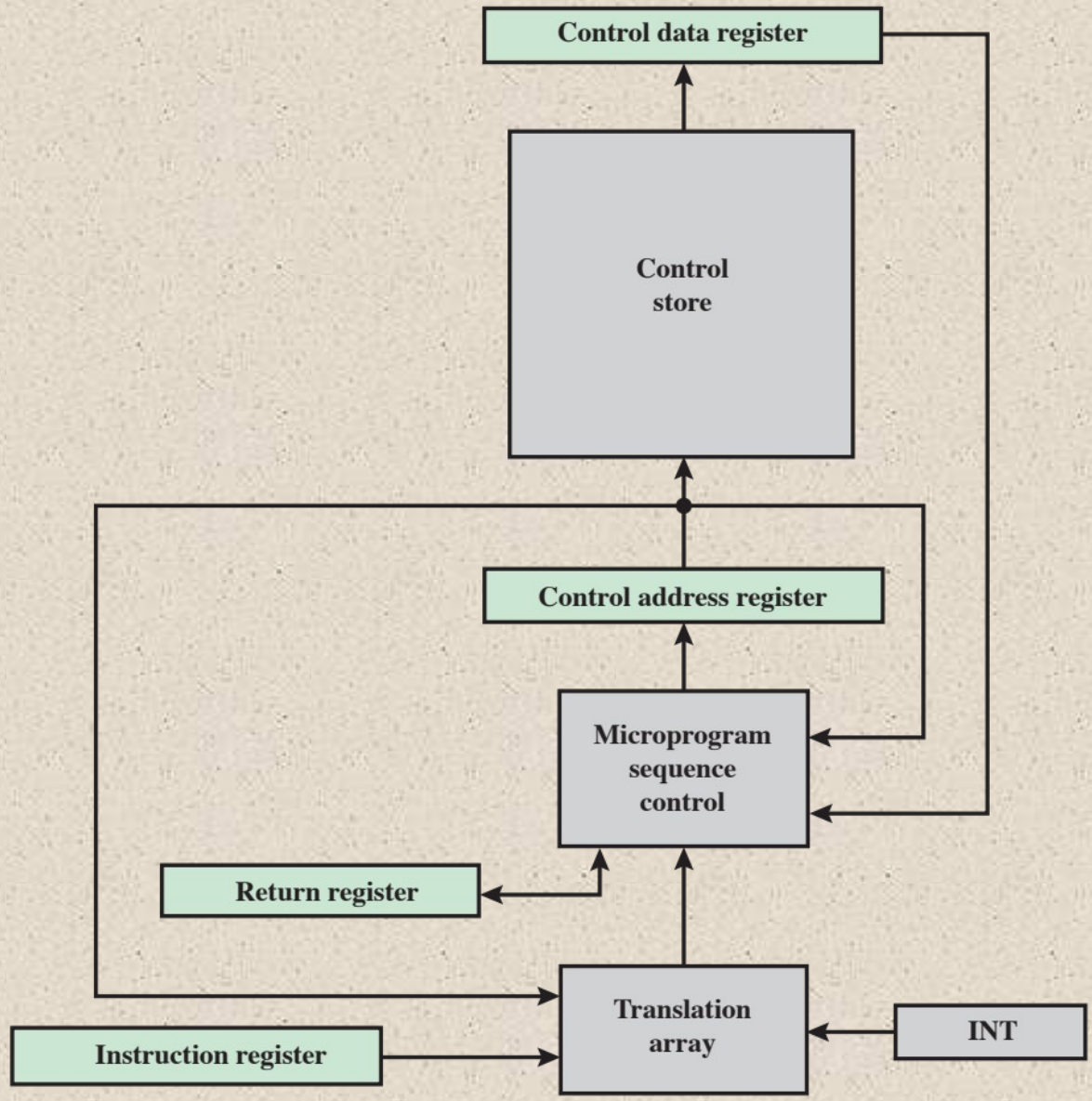


Figure 21.14 Organization of the LSI-11 Control Unit

Table 21.5
Some LSI-11 Microinstructions

Arithmetic Operations

Add word (byte, literal)
Test word (byte, literal)
Increment word (byte) by 1
Increment word (byte) by 2
Negate word (byte)
Conditionally increment (decrement) byte
Conditionally add word (byte)
Add word (byte) with carry
Conditionally add digits
Subtract word (byte)
Compare word (byte, literal)
Subtract word (byte) with carry
Decrement word (byte) by 1

Logical Operations

AND word (byte, literal)
Test word (byte)
OR word (byte)
Exclusive-OR word (byte)
Bit clear word (byte)
Shift word (byte) right (left) with (without) carry
Complement word (byte)

General Operations

MOV word (byte)
Jump
Return
Conditional jump
Set (reset) flags
Load G low
Conditionally MOV word (byte)

Input/Output Operations

Input word (byte)
Input status word (byte)
Read
Write
Read (write) and increment word (byte) by 1
Read (write) and increment word (byte) by 2
Read (write) acknowledge
Output word (byte, status)

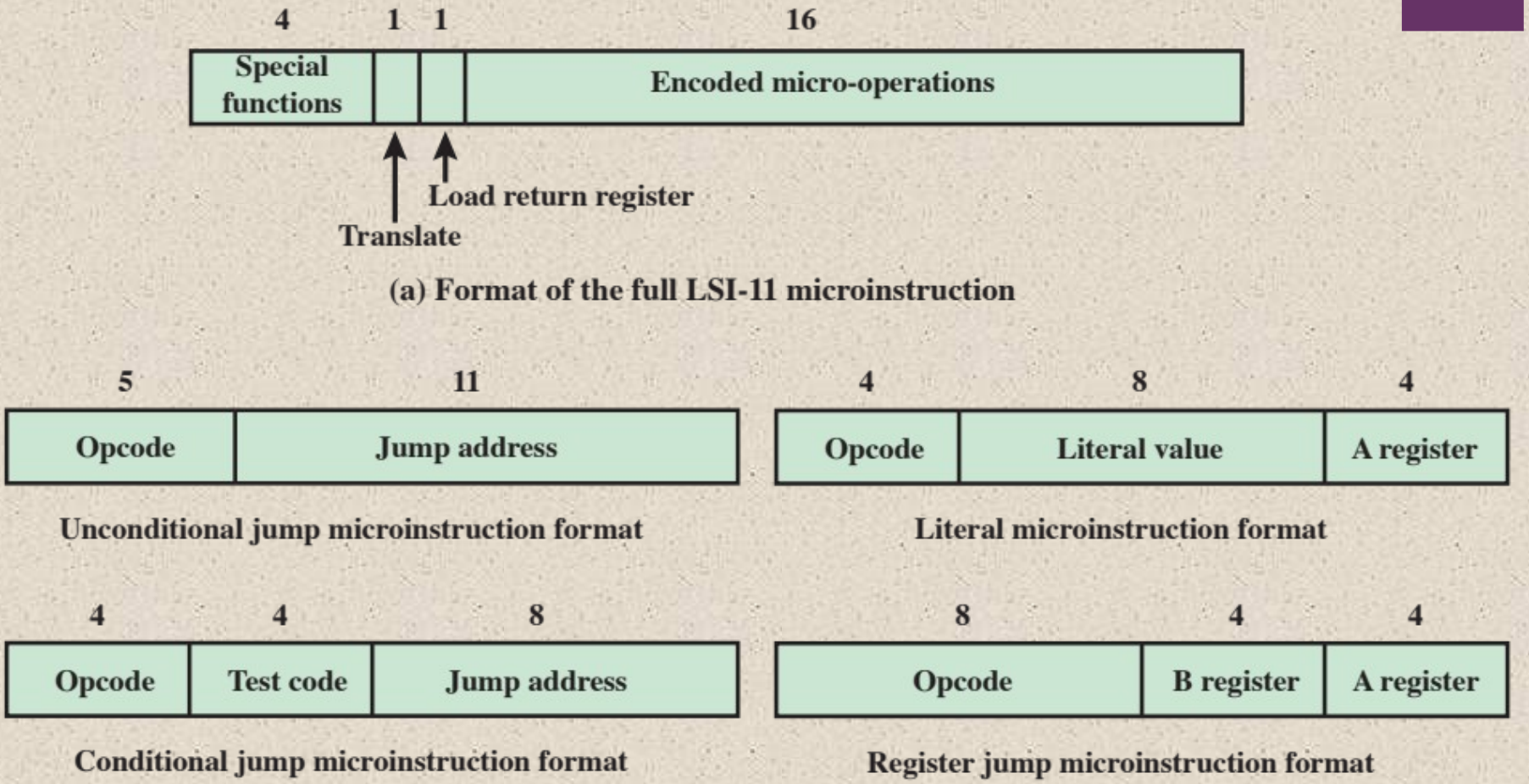


Figure 21.15 LSI-11 Microinstruction Format

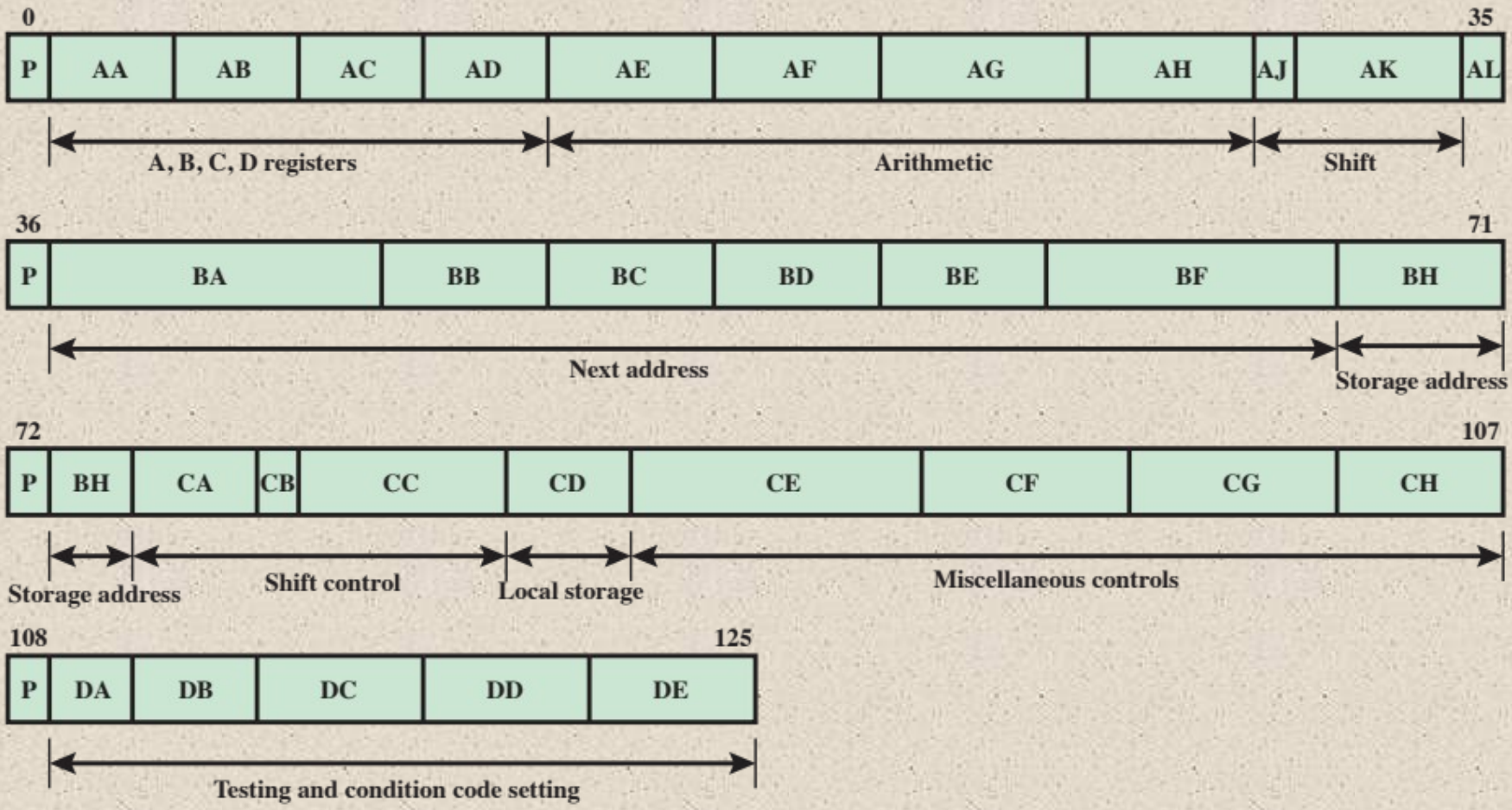


Figure 21.16 IBM 3033 Microinstruction Format

ALU Control Fields	
AA(3)	Load A register from one of data registers
AB(3)	Load B register from one of data registers
AC(3)	Load C register from one of data registers
AD(3)	Load D register from one of data registers
AE(4)	Route specified A bits to ALU
AF(4)	Route specified B bits to ALU
AG(5)	Specifies ALU arithmetic operation on A input
AH(4)	Specifies ALU arithmetic operation on B input
AJ(1)	Specifies D or B input to ALU on B side
AK(4)	Route arithmetic output to shifter
CA(3)	Load F register
CB(1)	Activate shifter
CC(5)	Specifies logical and carry functions
CE(7)	Specifies shift amount
Sequencing and Branching Fields	
AL(1)	End operation and perform branch
BA(8)	Set high-order bits (00–07) of control address register
BB(4)	Specifies condition for setting bit 8 of control address register
BC(4)	Specifies condition for setting bit 9 of control address register
BD(4)	Specifies condition for setting bit 10 of control address register
BE(4)	Specifies condition for setting bit 11 of control address register
BF(7)	Specifies condition for setting bit 12 of control address register

Table 21.6

IBM 3033 Microinstruction Control Fields

(Table can be found on page 755 in the

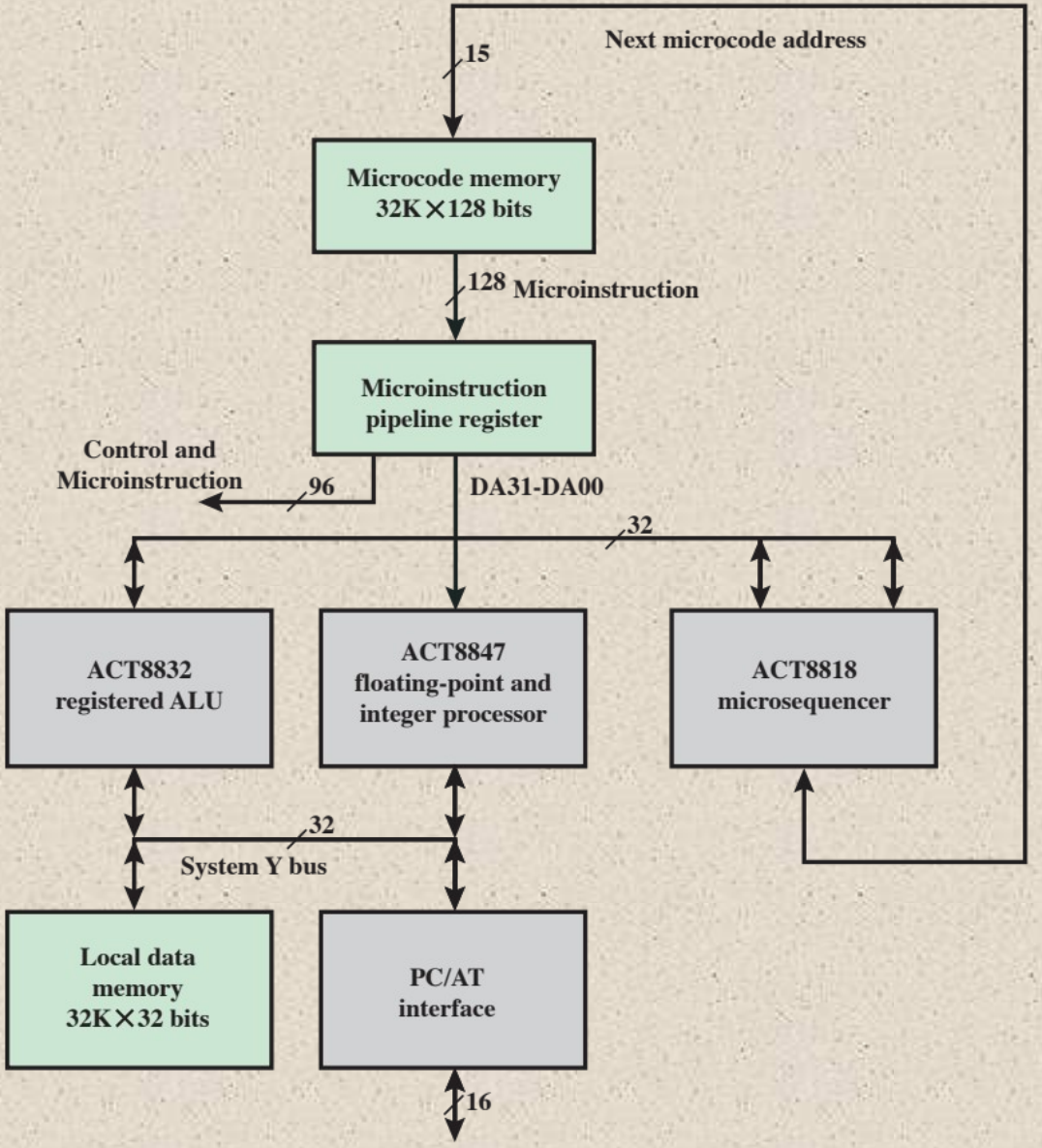


Figure 21.17 TI 8800 Block Diagram

Table 21.7 TI 8800 Microinstruction Format

Field Number	Number of Bits	Description
		Control of Board
1	5	Select condition code input
2	1	Enable/disable external I/O request signal
3	2	Enable/disable local data memory read/write operations
4	1	Load status/do no load status
5	2	Determine unit driving Y bus
6	2	Determine unit driving DA bus
		8847 Floating Point and Integer Processing Chip
7	1	C register control: clock, do not clock
8	1	Select most significant or least significant bits for Y bus
9	1	C register data source: ALU, multiplexer
10	4	Select IEEE or FAST mode for ALU and MUL
11	8	Select sources for data operands: RA registers, RB registers, P register, 5 register, C register
12	1	RB register control: clock, do not clock
13	1	RA register control: clock, do not clock
14	2	Data source confirmation
15	2	Enable/disable pipeline registers
16	11	8847 ALU function
		8832 Registered ALU
17	2	Write enable/disable data output to selected register: most significant half, least significant half
18	2	Select register file data source: DA bus, DB bus, ALU Y MUX output, system Y bus
19	3	Shift instruction modifier
20	1	Carry in: force, do not force
21	2	Set ALU configuration mode: 32, 16, or 8 bits
22	2	Select input to 5 multiplexer: register file, DB bus, MQ register
23	1	Select input to R multiplexer: register file, DA bus
24	6	Select register in file C for write
25	6	Select register in file B for read
26	6	Select register in file A for write
27	8	ALU function
		8818 Microsequencer
28	12	Control input signals to the 8818
		WCS Data Field
29	16	Most significant bits of writable control store data field
30	16	Least significant bits of writable control store data field

(Table can be found on page 758 in the textbook.)

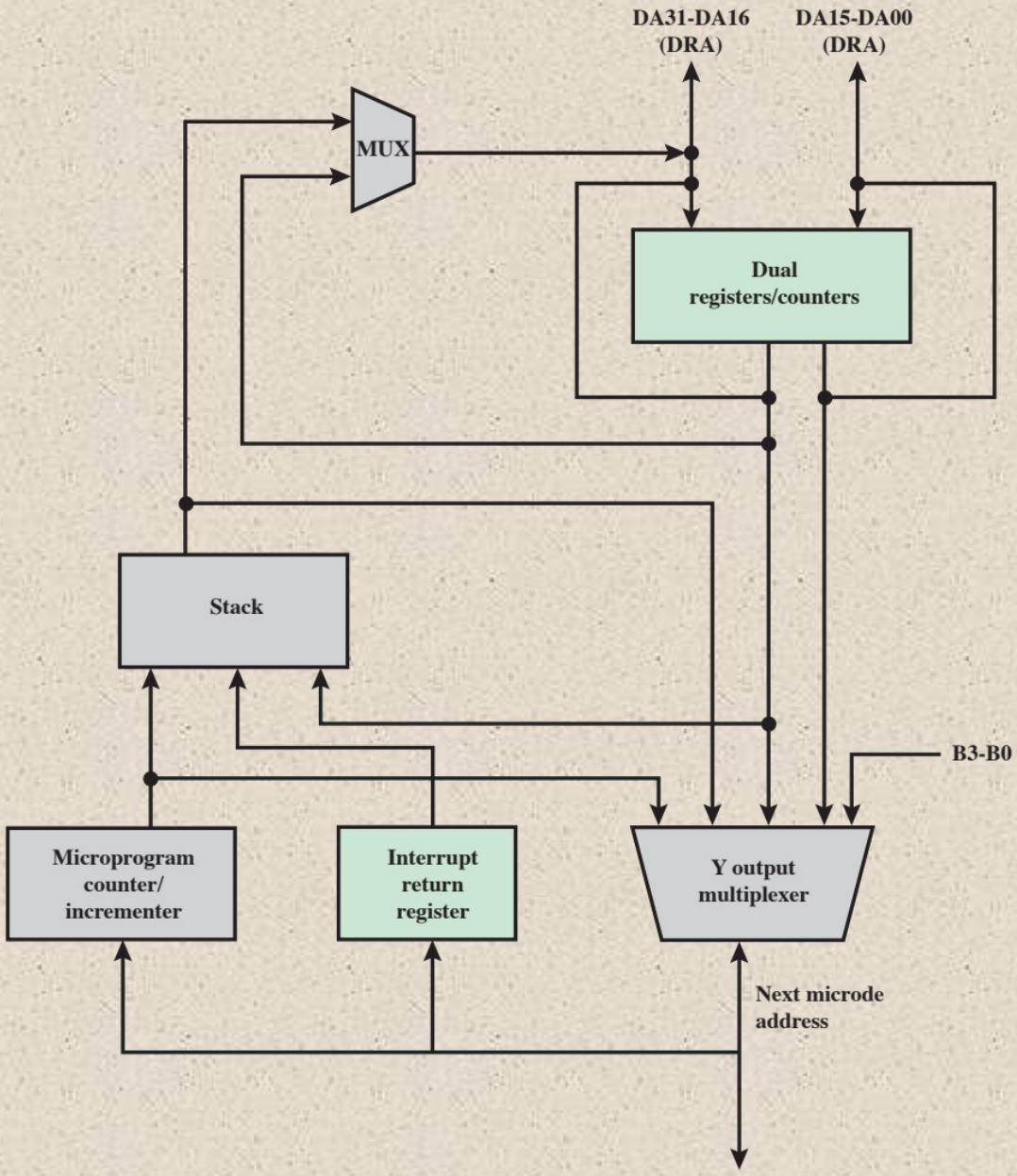


Figure 21.18 TI 8818 Microsequencer

Table 21.8 TI 8818 Microsequencer Microinstruction Bits (Field 28)

Mnemonic	Value	Description
RST8818	000000000110	Reset Instruction
BRA88181	011000111000	Branch to DRA Instruction
BRA88180	010000111110	Branch to DRA Instruction
INC88181	000000111110	Continue Instruction
INC88180	001000001000	Continue Instruction
CAL88181	010000110000	Jump to Subroutine at Address Specified by DRA
CAL88180	010000101110	Jump to Subroutine at Address Specified by DRA
RET8818	000000011010	Return from Subroutine
PUSH8818	000000110111	Push Interrupt Return Address onto Stack
POP8818	100000010000	Return from Interrupt
LOADDRA	000010111110	Load DRA Counter from DA Bus
LOADDRB	000110111110	Load DRB Counter from DA Bus
LOADDRAB	000110111100	Load DRA/DRB
DECRDRA	010001111100	Decrement DRA Counter and Branch If Not Zero
DECRDRB	010101111100	Decrement DRB Counter and Branch If Not Zero

(Table can be found on page 761 in the textbook.)

Table 21.9 TI 8832 Registered ALU Instruction Field (Field 27) (page 1 of 2)

Group 1		Function
ADD	H#01	R + S + Cn
SUBR	H#02	(NOT R) + S + Cn
SUBS	H#03	R = (NOT S) + Cn
INSC	H#04	S + Cn
INCNS	H#05	(NOT S) + Cn
INCR	H#06	R + Cn
INCNR	H#07	(NOT R) + Cn
XOR	H#09	R XOR S
AND	H#0A	R AND S
OR	H#0B	R OR S
NAND	H#0C	R NAND S
NOR	H#0D	R NOR S
ANDNR	H#0E	(NOT R) AND S
Group 2		**Function**
SRA	H#00	Arithmetic right single precision shift
SRAD	H#10	Arithmetic right double precision shift
SRL	H#20	Logical right single precision shift
SRLD	H#30	Logical right double precision shift
SLA	H#40	Arithmetic left single precision shift
SLAD	H#50	Arithmetic left double precision shift
SLC	H#60	Circular left single precision shift
SLCD	H#70	Circular left double precision shift
SRC	H#80	Circular right single precision shift
SRCD	H#90	Circular right double precision shift
MQSRA	H#A0	Arithmetic right shift MQ register
MQSRL	H#B0	Logical right shift MQ register
MQSLL	H#C0	Logical left shift MQ register
MQSLC	H#D0	Circular left shift MQ register
LOADMQ	H#E0	Load MQ register
PASS	H#F0	Pass ALU to Y (no shift operation)
Group 3		**Function**
SET1	H#08	Set bit 1
Set0	H#18	Set bit 0
TB1	H#28	Test bit 1
TB0	H#38	Test bit 0
ABS	H#48	Absolute value
SMTC	H#58	Sign magnitude/twos-complement
ADDI	H#68	Add immediate

(Table can be found on page 764 in the textbook.)

Table 21.9 TI 8832 Registered ALU Instruction Field (Field 27) (page 2 of 2)

SUBI	H#78	Subtract immediate
BADD	H#88	Byte add R to S
BSUBS	H#98	Byte subtract S from R
BSUBR	H#A8	Byte subtract R from S
BINCS	H#B8	Byte increment S
BINCNS	H#C8	Byte increment negative S
BXOR	H#D8	Byte XOR R and S
BAND	H#E8	Byte AND R and S
BOR	H#F8	Byte OR R and S
Group 4		**Function**
CRC	H#00	Cyclic redundancy character accum.
SEL	H#10	Select S or R
SNORM	H#20	Single length normalize
DNORM	H#30	Double length normalize
DIVRF	H#40	Divide remainder fix
SDIVQF	H#50	Signed divide quotient fix
SMULI	H#60	Signed multiply iterate
SMULT	H#70	Signed multiply terminate
SDIVIN	H#80	Signed divide initialize
SDIVIS	H#90	Signed divide start
SDIVI	H#A0	Signed divide iterate
UDIVIS	H#B0	Unsigned divide start
UDIVI	H#C0	Unsigned divide iterate
UMULI	H#D0	Unsigned multiply iterate
SDIVIT	H#E0	Signed divide terminate
UDIVIT	H#F0	Unsigned divide terminate
Group 5		**Function**
LOADFF	H#0F	Load divide/BCD flip-flops
CLR	H#1F	Clear
DUMPFF	H#5F	Output divide/BCD flip-flops
BCDBIN	H#7F	BCD to binary
EX3BC	H#8F	Excess -3 byte correction
EX3C	H#9F	Excess -3 word correction
SDIVO	H#AF	Signed divide overflow test
BINEX3	H#DF	Binary to excess –3
NOP32	H#FF	No operation

(Table can be found on page 765 in the textbook.)

Summary

Chapter 21

- Basic concepts
  - Microinstructions
  - Microprogrammed control unit
  - Wilkes control
  - Advantages and disadvantages
- TI 8800
  - Microinstruction format
  - Microsequencer
  - Registered ALU

Microprogrammed Control

- Microinstruction sequencing
  - Design considerations
  - Sequencing techniques
  - Address generation
  - LSI-11 microinstruction sequencing
- Microinstruction execution
  - Taxonomy of microinstructions
  - Microinstruction encoding
  - LSI-11 Microinstruction execution
  - IBM 3033 Microinstruction execution