Chapter 5: Computer Systems Organization

Invitation to Computer Science,
C++ Version, Fourth Edition
*Updated on 1 April 2009**

Objectives

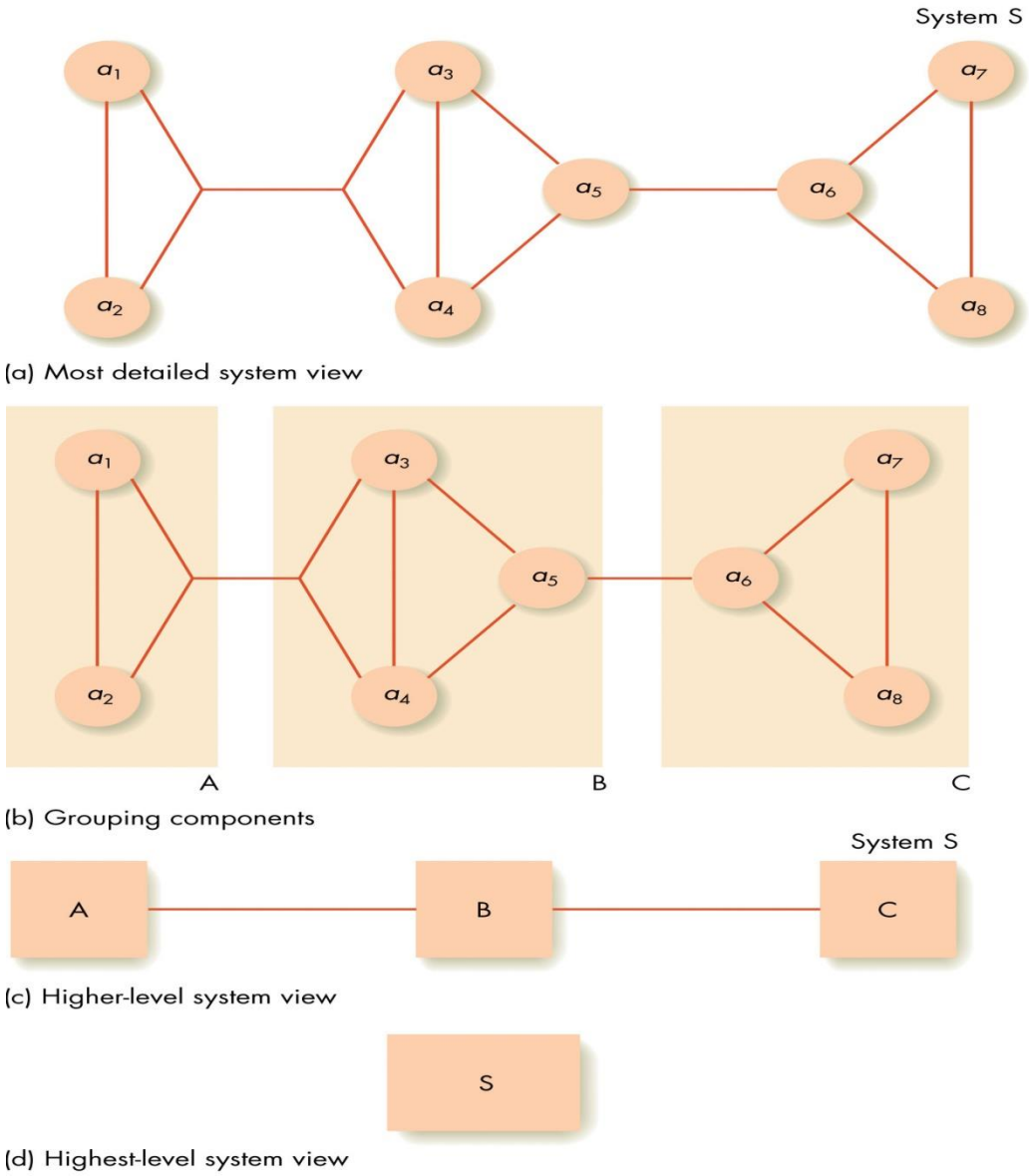
In this chapter, you will learn about

- The components of a computer system
- Putting all the pieces together—the Von Neumann architecture
- Non-Von Neumann architectures

Introduction

- Computer organization examines the computer as a collection of interacting “functional units”
- Functional units may be built out of the circuits already studied
- Higher level of abstraction assists in understanding by reducing complexity

Figure 5.1
The Concept of Abstraction



The Components of a Computer System

- Von Neumann architecture has four functional units
  - Memory
  - Input/Output
  - Arithmetic/Logic unit
  - Control unit
- Sequential execution of instructions
- Stored program concept

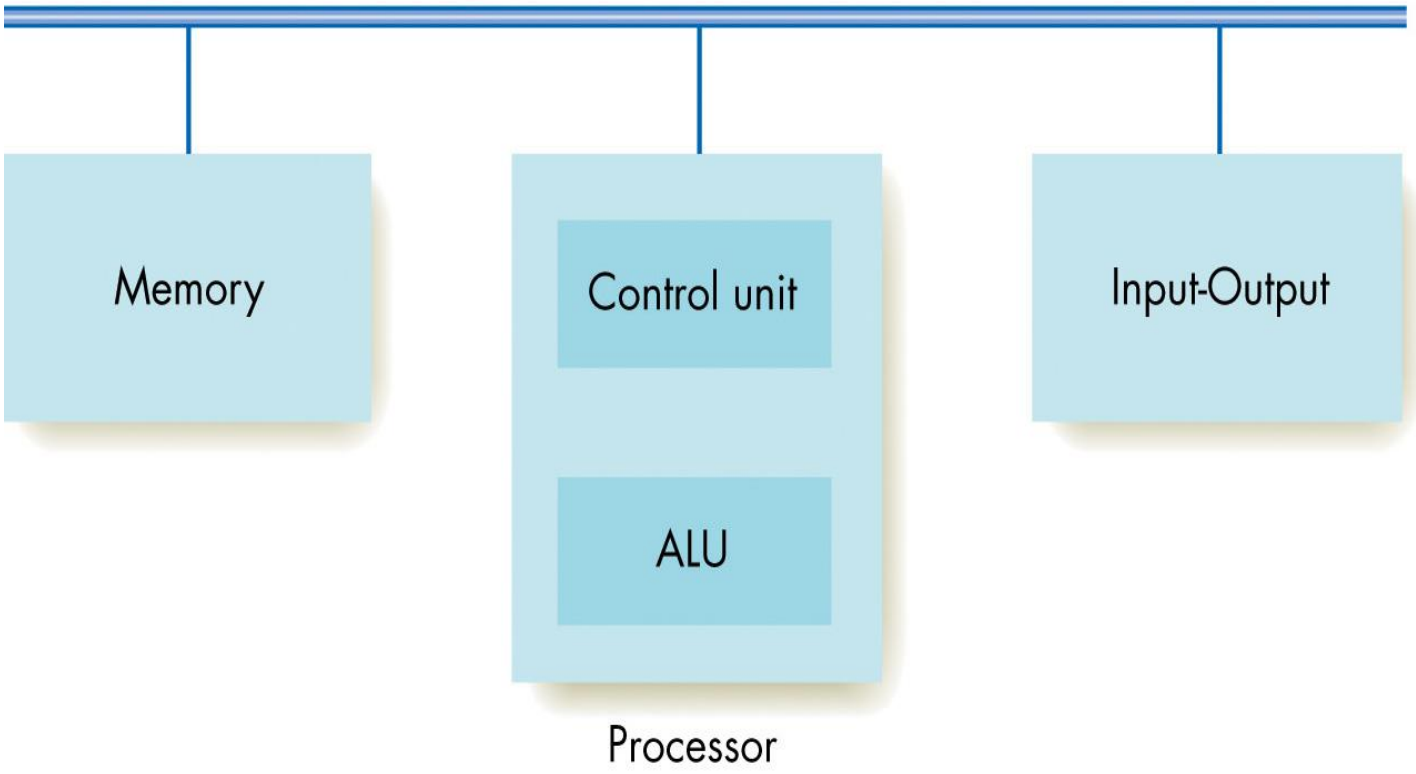
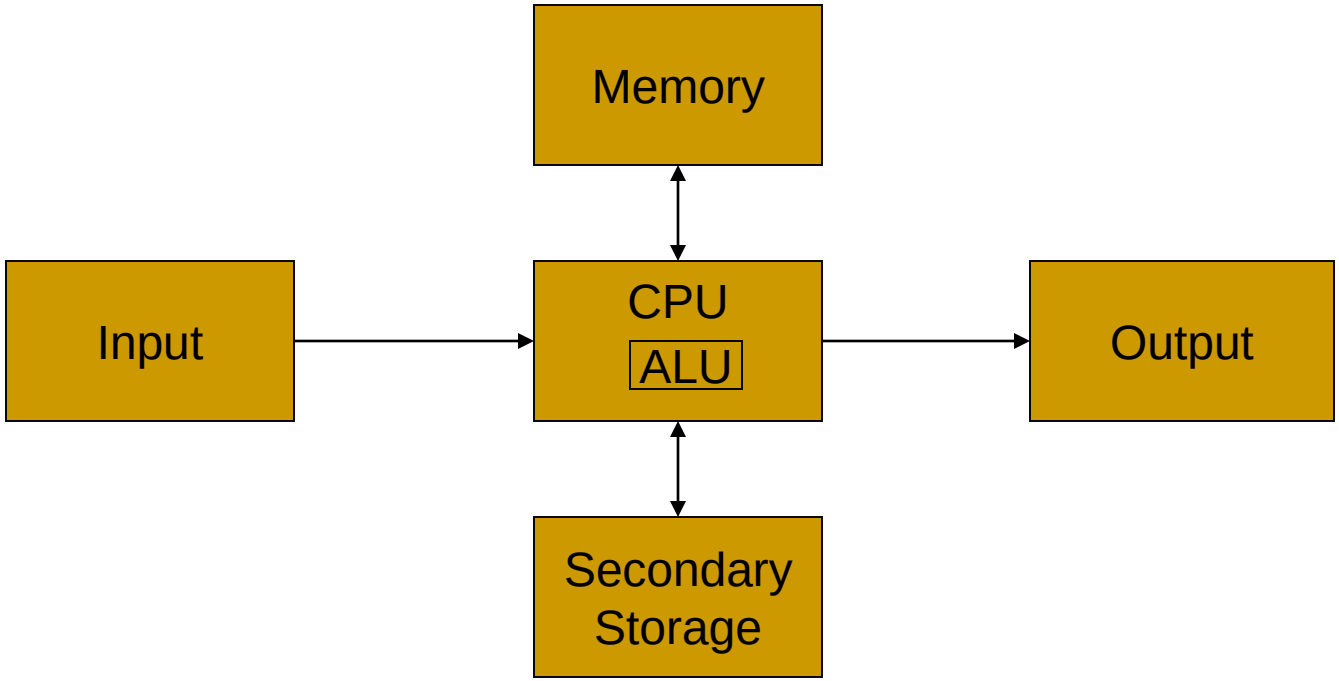


Figure 5.2
Components of the Von Neumann Architecture

Computer Organization*

- Logical organization of computer



Memory and Cache

- Information stored and fetched from memory subsystem
- Random access memory maps addresses to memory locations
- Cache memory keeps values currently in use in faster memory to speed access times

Memory and Cache (continued)

- RAM (random access memory)
  - Memory made of addressable cells
  - Current standard cell size is 8 bits
  - All memory cells accessed in equal time
  - Memory address
    - Unsigned binary number *N* long
    - Address space is then 2^N cells

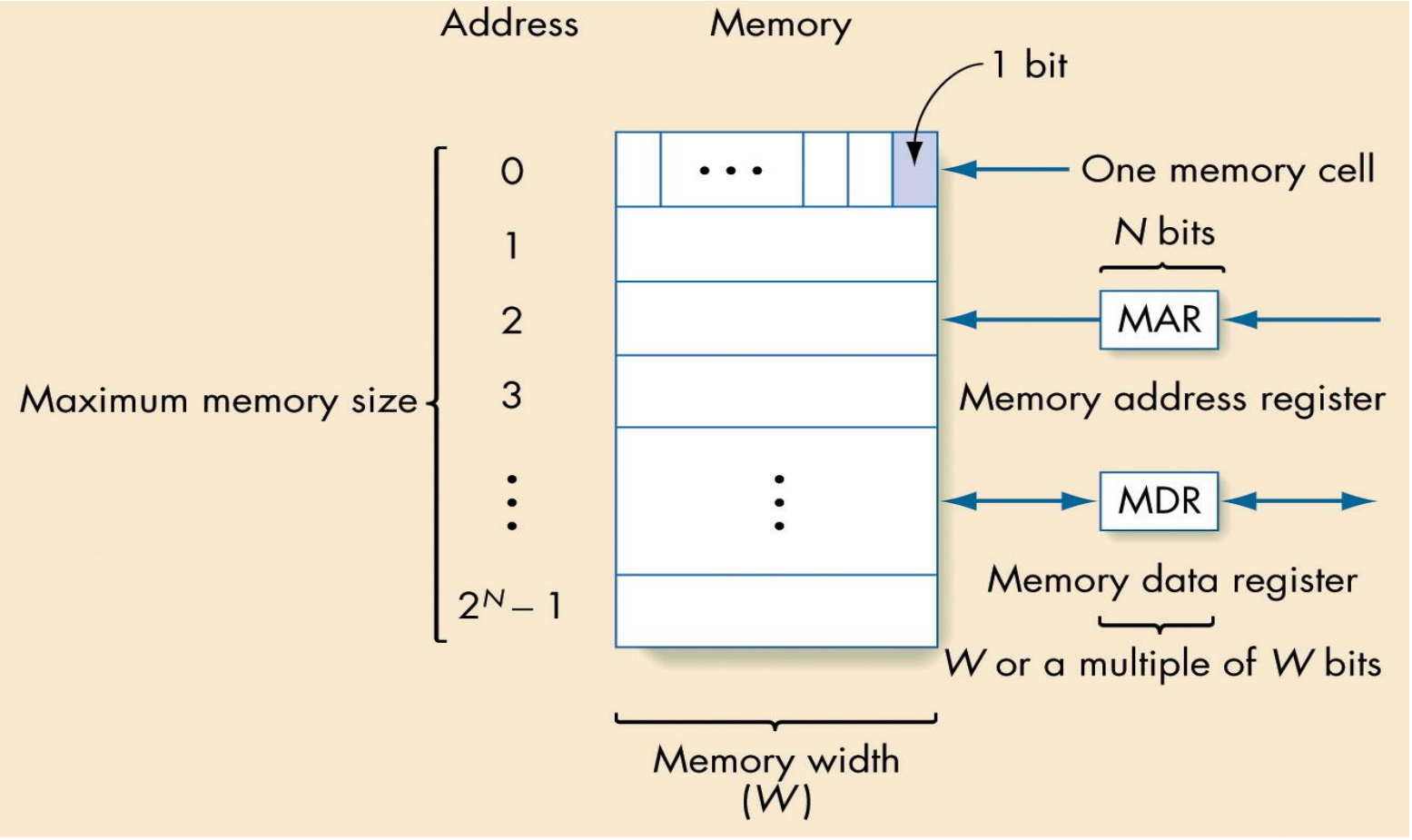


Figure 5.3
Structure of Random Access Memory

Memory and Cache (continued)

- Parts of the memory subsystem
  - Fetch/store controller
    - Fetch: Retrieve a value from memory
    - Store: Store a value into memory
  - Memory address register (MAR)
  - Memory data register (MDR)
  - Memory cells with decoder(s) to select individual cells

Memory and Cache (continued)

- Fetch operation
  - The address of the desired memory cell is moved into the MAR
  - Fetch/store controller signals a fetch, accessing the memory cell
  - The value at the MAR's location flows into the MDR

Memory and Cache (continued)

- Store operation
  - The address of the cell where the value should go is placed in the MAR
  - The new value is placed in the MDR
  - Fetch/store controller signals a store, copying the MDR's value into the desired cell

Memory and Cache (continued)

- Memory register
  - Very fast memory location
  - Given a name, not an address
  - Serves some special purpose
  - Modern computers have dozens or hundreds of registers

Powers of Two*

2^{10}	1K =1,024	*kilo*	10^3 1,000
2^{20}	1M=1,048,576	*mega*	10^6 1,000,000
2^{30}	1G=1,07,741,824	*giga*	10^9 1,000,000,000
2^{40}	1T=1,099,511,627,776	*tera*	10^{12} 1,000,000,000,000
2^{50}	1P=1,125,899,906,842,624	*peta*	10^{15}

Powers of Two*

2^{10}	1K =1,024 10^3=*1,000*	*kilo*	10^3 1,000
2^{20}	1M=1,048,576 *1,000,000*	*mega*	10^6 1,000,000
2^{30}	1G=1,07,741,824 *1,000,000,000*	*giga*	10^9 1,000,000,000
2^{40}	1T=1,099,511,627,776 *1,000,000,000,000*	*tera*	10^{12} 1,000,000,000,000
2^{50}	1P=1,125,899,906,842,624	*peta*	10^{15}
	1,000,000,000,000,000		

Powers of 10 as text

# Bytes	Base 10 value	Amt of text
1 byte	10^0	1 character
1 kilobyte (thousand)	10^3	1 typed page
1 megabyte (million)	10^6	2-3 novels
1 gigabyte (billion)	10^9	Dept. library
1 terabyte (trillion)	10^{12}	University library
1 petabyte (quadrillion)	10^{15}	All libraries. In N. America
1 exabyte (quintillion)	10^{18}	All words ever printed in history
1 zettabyte (hexillion)	10^{21}	-
1 yottabyte (heptillion)	10^{24}	-

Some sizes dictated by the structure of main memory*

With our instruction having the form of

- 4 bits for the op code
- 12 bits for the address

if we plan to have one instruction per memory cell, then we need to have for our computer

An MAR (memory address register) of 12 bits.

A memory size of at most $2^{12} = 2^{2} * 2^{10} = 4K$

A memory width of 4 + 12 = 16 bits

If MDR (memory data register) is 16 bits, then the largest sized number is
$0111\ 1111\ 1111\ 1111_2 = 2^{15} - 1 = 32{,}768.$

Other Components of Memory Unit*

Besides the Random Access Memory and the MAR and MDR, two other components exist:

1) Fetch/store controller: Sends a signal to Fetch or Store

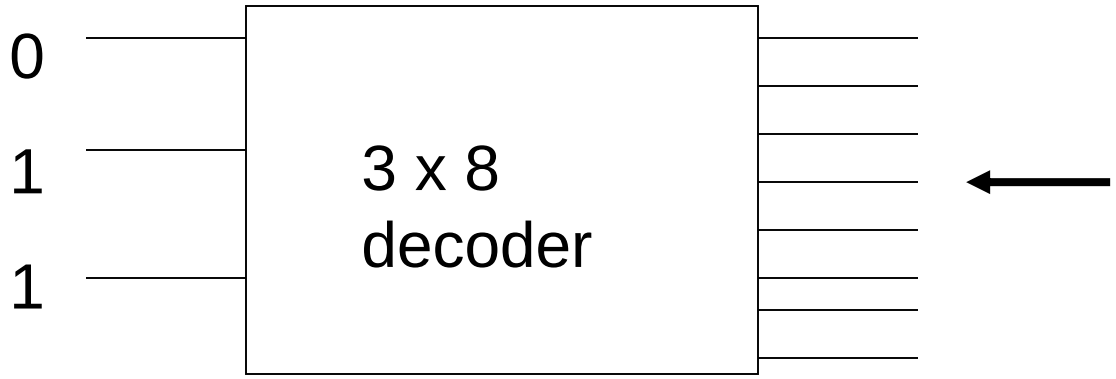
2) Memory decoder circuits: (Ref, Chpt 4, pg 180-182)

A N x 2^N decoder has N input lines and 2^N output lines.

When the N input lines are set to 0s or 1s and the N values are interpreted as a binary number, they represent all the numbers between 0 and 2^N-1.

The output to the decoder is a 1 on the line identified by the value of the input and a 0 on all the other lines.

Example*

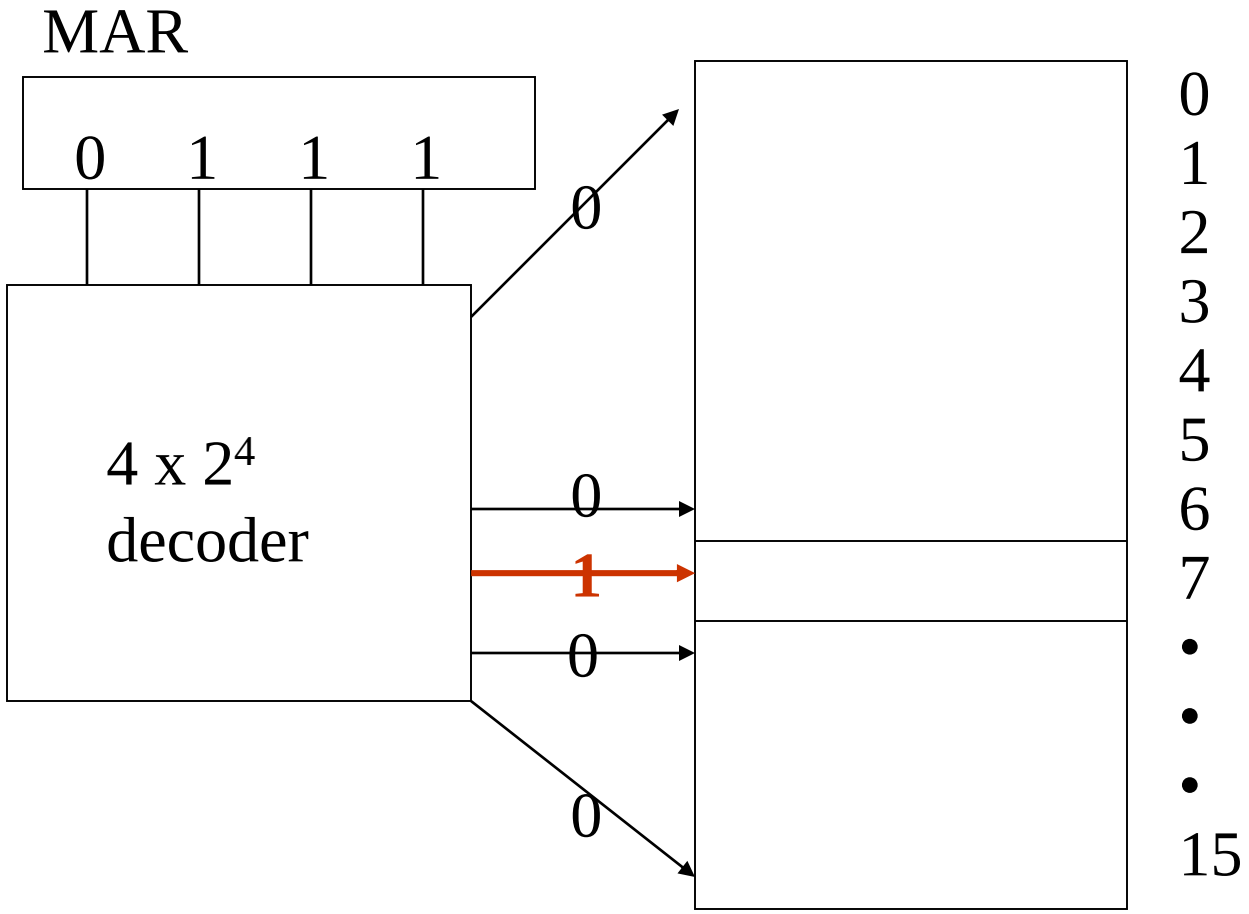


011_2 = 3 so the line labeled 3, the 4th from the top outputs a 1 and all other lines output a 0.

A decoder selects one line for a pulse, when the input lines are interpreted as a binary number.

Why is this useful for a memory unit?

Using the Decoder Circuit to Select Memory Locations*



The Decoder Circuit*

Decoder: can be built from AND-OR-NOT gates

See Figure 4.29 on page 181 for a 2 x 4 decoder circuit.

As with all circuits, to build a decoder,

1) Build a truth table for the circuit

(For example, for a 3 x 8 decoder, there are 8 rows, 3 input choices, and 8 output values).

2) Use the sum-of-products algorithm to find the Boolean expression for the truth table.

3) Build the circuit.

The decoder circuit doesn't scale well--- i.e. as the number of bits in the MAR increases, the number of output lines for the decoder goes up exponentially.

Most computers today have an MAR of 32 bits. Thus, if the memory was laid out as we showed it, we would need a 32 x 2^{32} decoder!

Note 2^{32} is 2^2 2^{30} = 4 G

So most memory is not 1 dimensional, but 2-dimensional (or even 3-dimensional if banked memory is used).

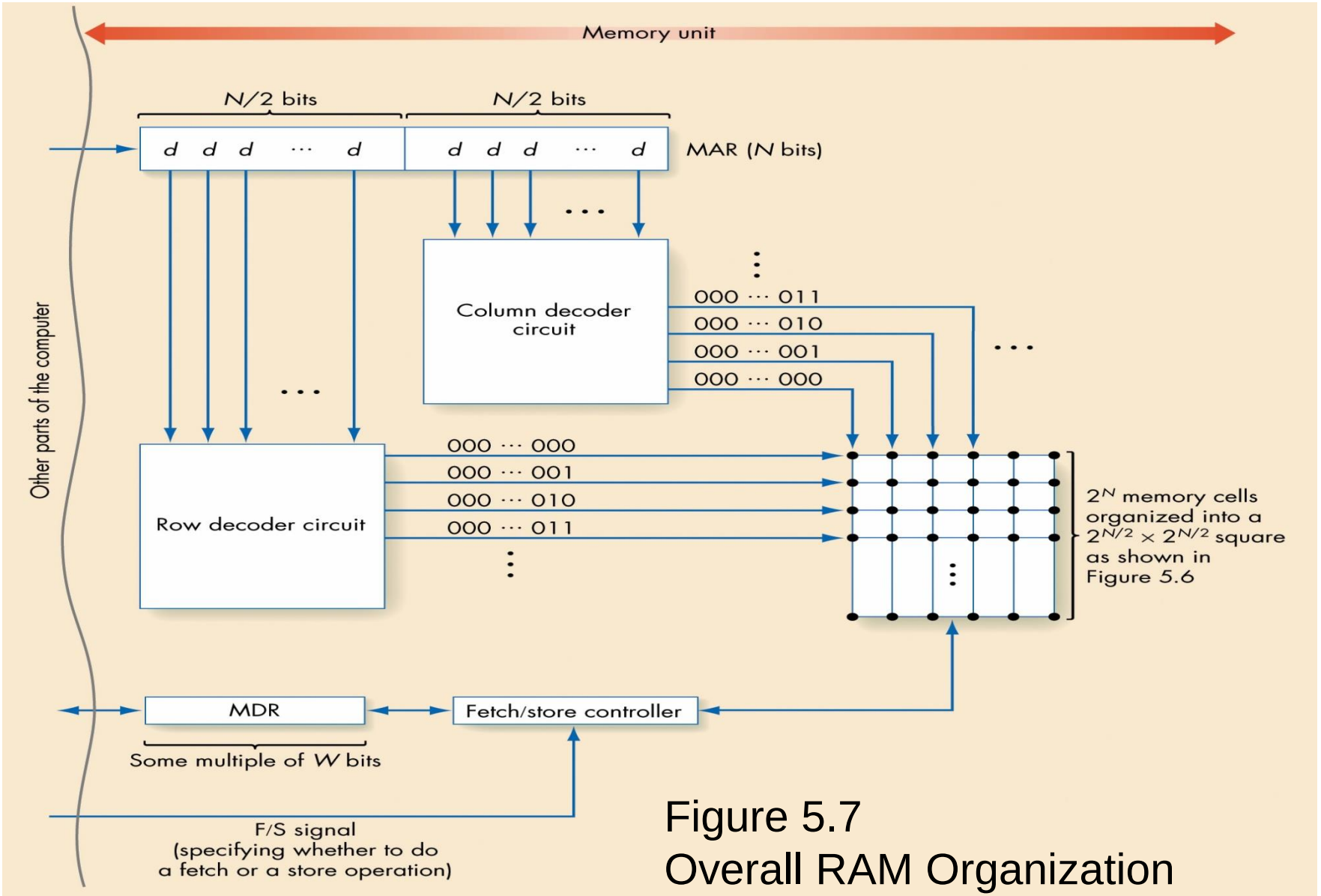
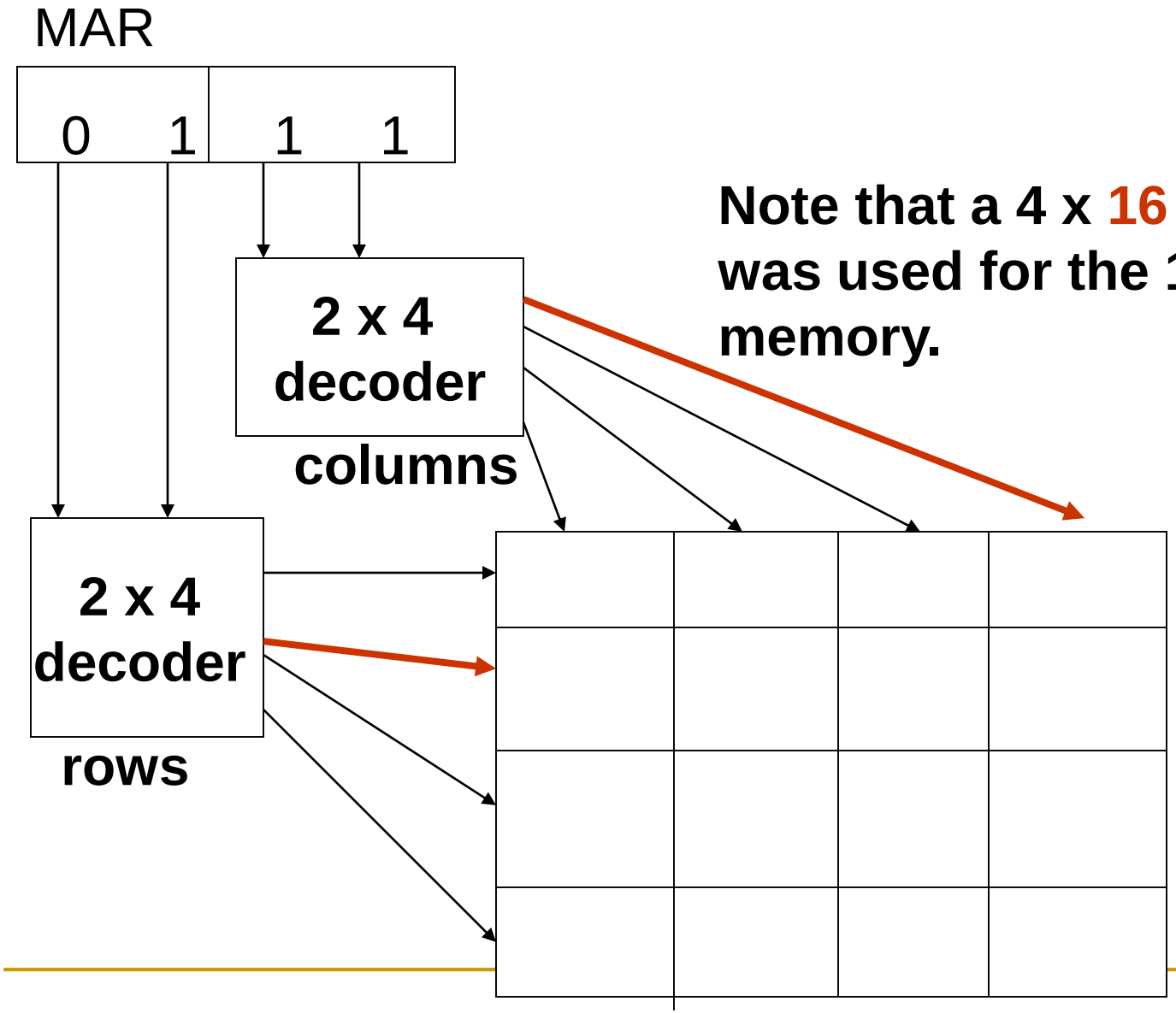


Figure 5.7
Overall RAM Organization

2-D MEMORY*


MAR
0
1
1
1
2 x 4 decoder
columns
2 x 4 decoder
rows
Note that a 4 x 16 decoder was used for the 1-D memory.

Cache Memory

- Memory access is much slower than processing time
- Faster memory is too expensive to use for all memory cells
- Locality principle
  - Once a value is used, it is likely to be used again
- Small size, fast memory just for values currently in use speeds computing time

Input/Output and Mass Storage

- Communication with outside world and external data storage
  - Human interfaces: Monitor, keyboard, mouse
  - Archival storage: Not dependent on constant power
- External devices vary tremendously from each other

Input/Output and Mass Storage (continued)

- Volatile storage
  - Information disappears when the power is turned off
  - Example: RAM
- Nonvolatile storage
  - Information does not disappear when the power is turned off
  - Example: Mass storage devices such as disks and tapes

Input/Output and Mass Storage (continued)

- Mass storage devices
  - Direct access storage device
    - Hard drive, CD-ROM, DVD
    - Uses its own addressing scheme to access data
  - Sequential access storage device
    - Tape drive
    - Stores data sequentially
    - Used for backup storage these days

Input/Output and Mass Storage* (continued)

- Direct access storage devices
  - Data stored on a spinning disk
  - Disk divided into concentric rings (sectors)
  - Read/write head moves from one ring to another while disk spins
  - Access time depends on
    - Time to move head to correct sector (track): **Seek time**
    - Time for sector to spin to data location: **Latency**

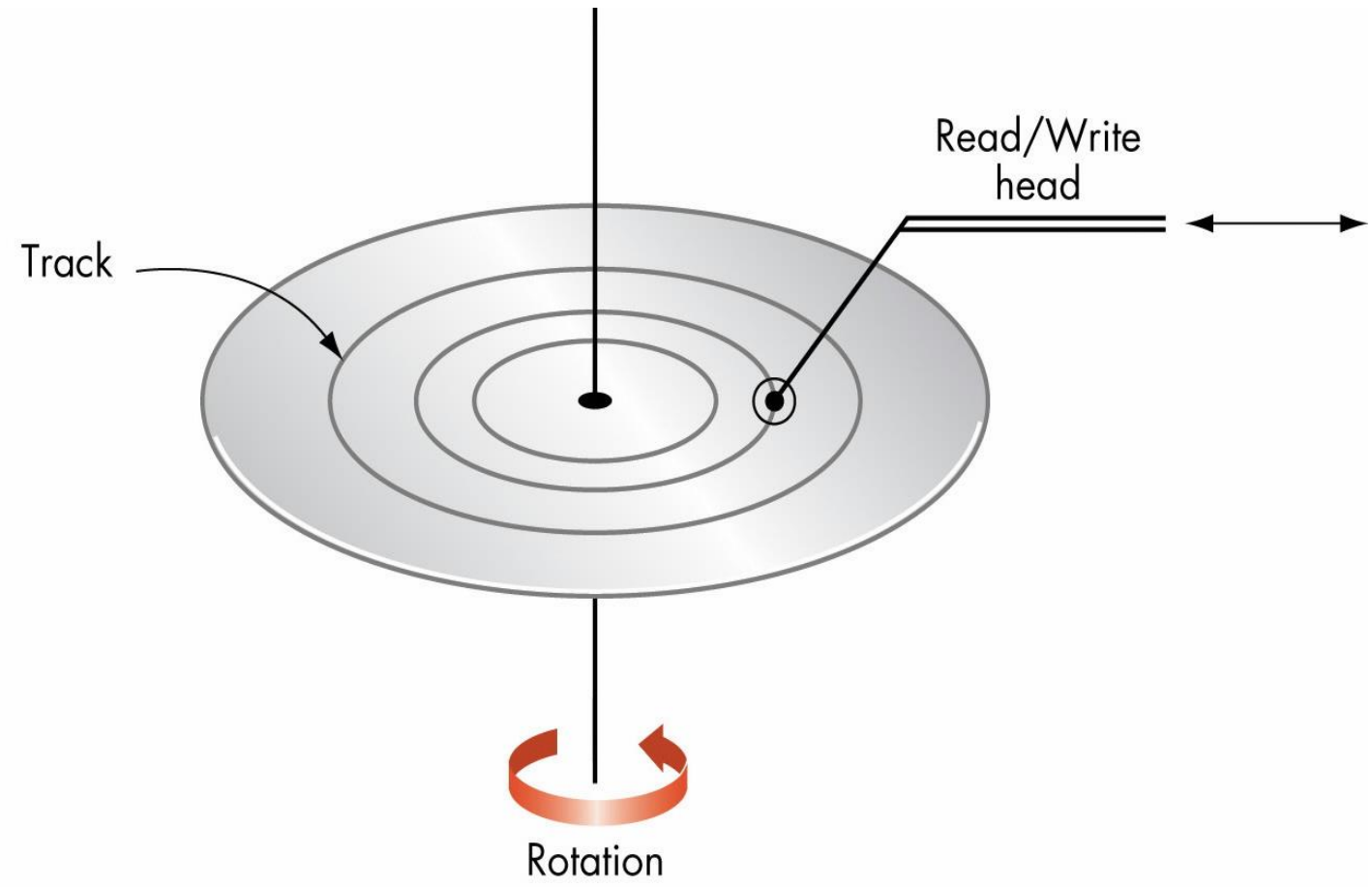


Figure 5.8
Overall Organization of a Typical Disk

Input/Output and Mass Storage (continued)

- I/O controller
  - Intermediary between central processor and I/O devices
  - Processor sends request and data, then goes on with its work
  - I/O controller interrupts processor when request is complete

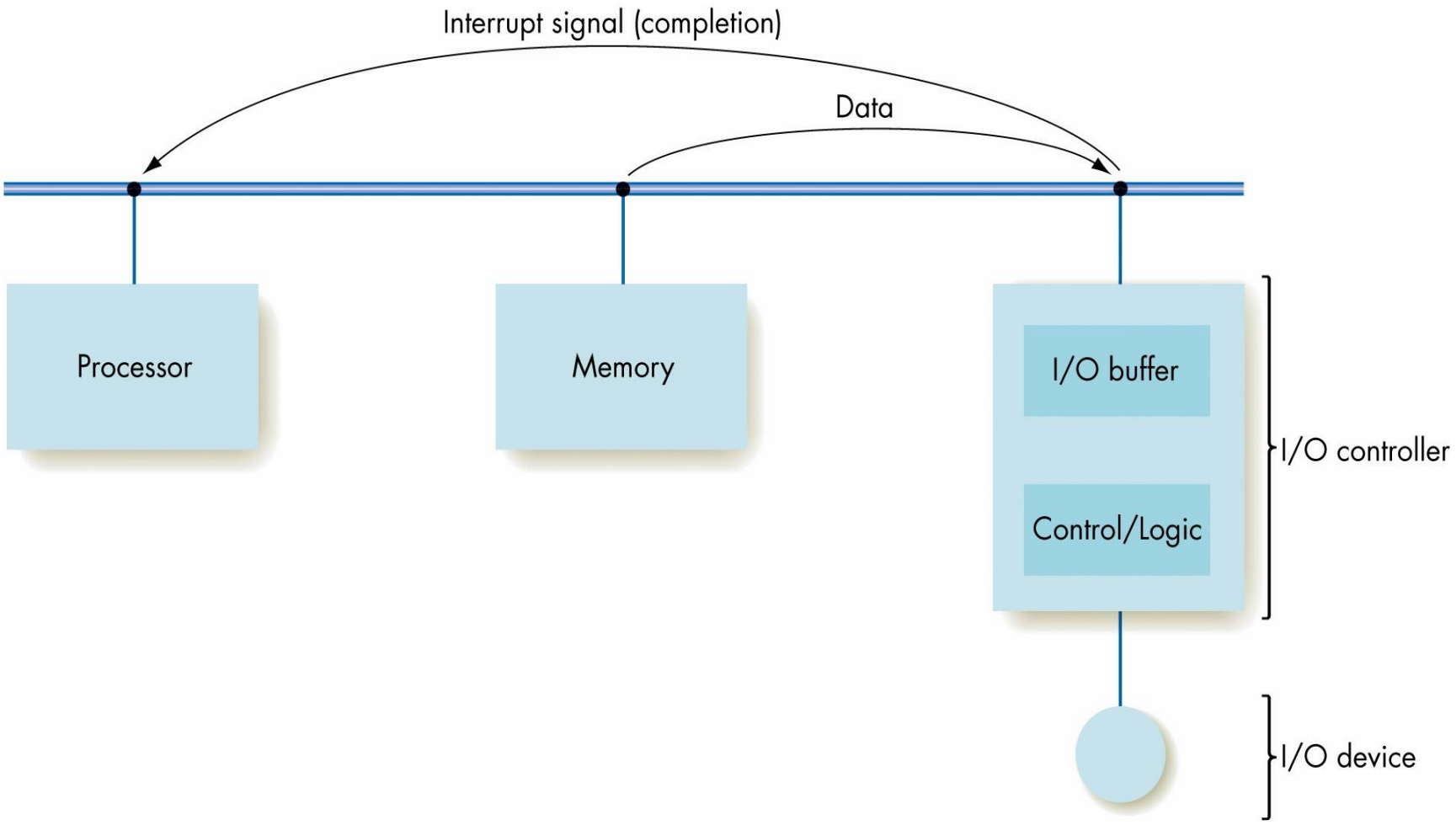


Figure 5.9
Organization of an I/O Controller

The Arithmetic/Logic Unit*

- ***A**rithmetic and **L**ogic **U**nit*
  - *"Manufacturing" section of computer*
  - *Contains decision mechanisms and can make comparisons*
- Actual computations are performed
- Primitive operation circuits
  - Arithmetic *[+, -, *, /]*
  - Comparison *[equality or CE, GT, LT, NEQ]*
  - Logic *[AND, OR, NOT, XOR]* Data inputs and results stored in registers
- Multiplexor selects desired output

The Arithmetic/Logic Unit (continued)

- ALU process
  - Values for operations copied into ALU's input register locations
  - All circuits compute results for those inputs
  - Multiplexor selects the one desired result from all values
  - Result value copied to desired result register

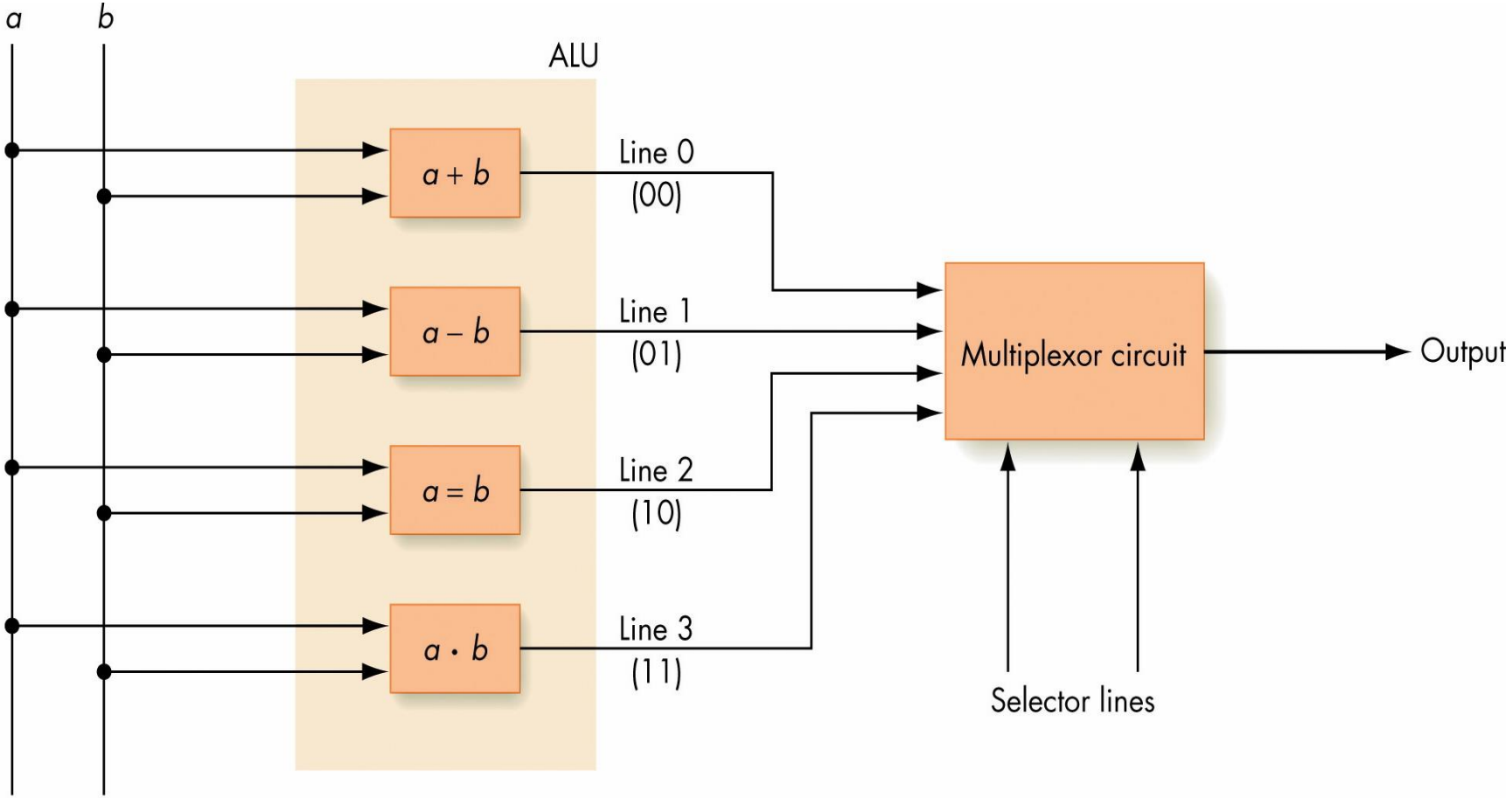


Figure 5.12

Using a Multiplexor Circuit to Select the Proper ALU Result

The Control Unit

- Manages stored program execution
- Task
  - Fetch from memory the next instruction to be executed
  - Decode it: Determine what is to be done
  - Execute it: Issue appropriate command to ALU, memory, and I/O controllers

Machine Language Instructions

- Can be decoded and executed by control unit
- Parts of instructions
  - Operation code (op code)
    - Unique unsigned-integer code assigned to each machine language operation
  - Address fields
    - Memory addresses of the values on which operation will work

Figure 5.14
Typical Machine Language Instruction Format

Machine Language Instructions*

	Instruction	Binary Op Code	Meaning
0	Load	0000	
1	Store	0001	
2	Clear	0010	
3	Add	0011	
4	Increment	0100	
5	Subtract	0101	
6	Decrement	0110	
7	Compare	0111	
8	Jump	1000	
9	Jump GT	1001	
10	Jump EQ	1010	
11	Jump LT	1011	
12	Jump NEQ	1100	
13	In	1101	
14	Out	1110	
15	Halt	1111	

Machine Language Instructions (continued)

- Operations of machine language
  - Data transfer operations
    - Move values to and from memory and registers
  - Arithmetic/logic operations
    - Perform ALU operations that produce numeric values

Machine Language Instructions (continued)

- Operations of machine language (continued)
  - Compare operations
    - Compare two values and set an indicator on the basis of the results of the compare; set register bits
  - Branch operations
    - Jump to a new memory address to continue processing

Control Unit Registers and Circuits

- Parts of control unit
  - Links to other subsystems
  - Instruction decoder circuit
  - Two special registers
    - Program counter (PC)
      - Stores the memory address of the next instruction to be executed
    - Instruction register (IR)
      - Stores the code for the current instruction

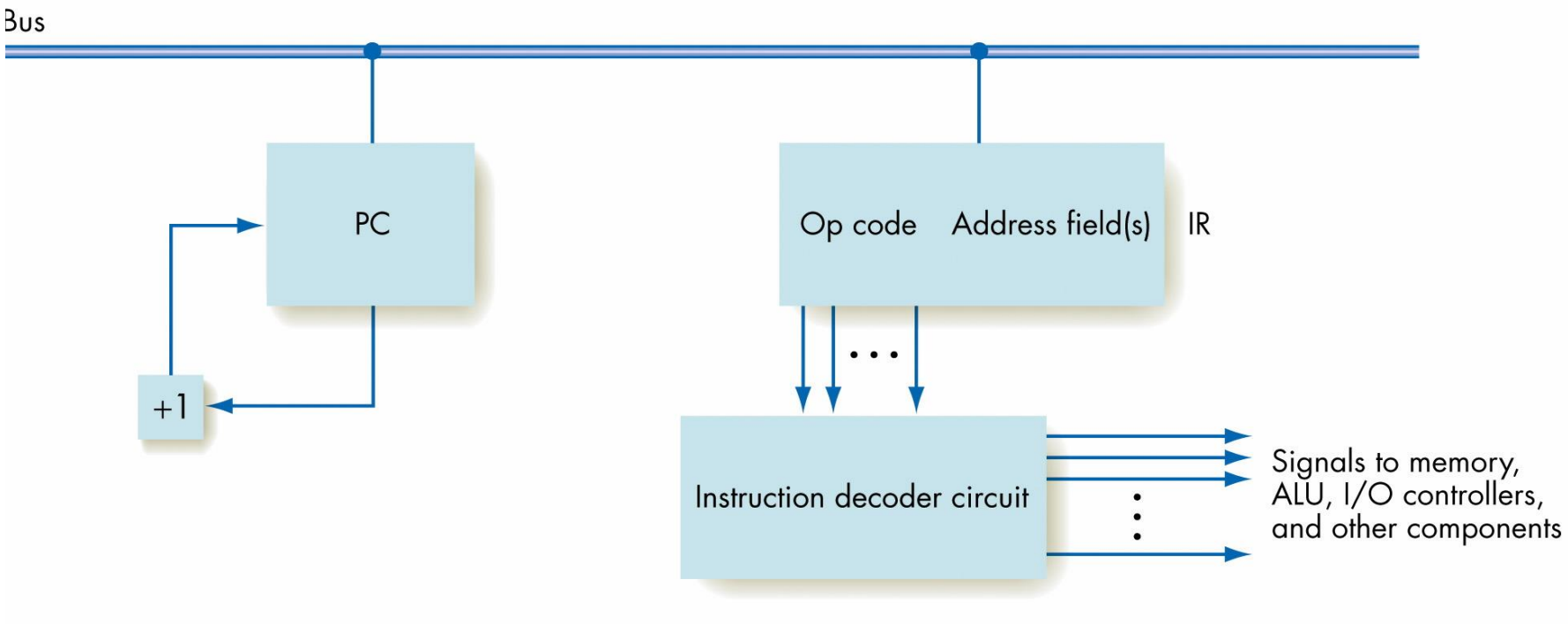


Figure 5.16
Organization of the Control Unit Registers and Circuits

Putting All the Pieces Together—the Von Neumann Architecture

- Subsystems connected by a bus
  - Bus: Wires that permit data transfer among them
- At this level, ignore the details of circuits that perform these tasks: Abstraction!
- Computer repeats fetch-decode-execute cycle indefinitely

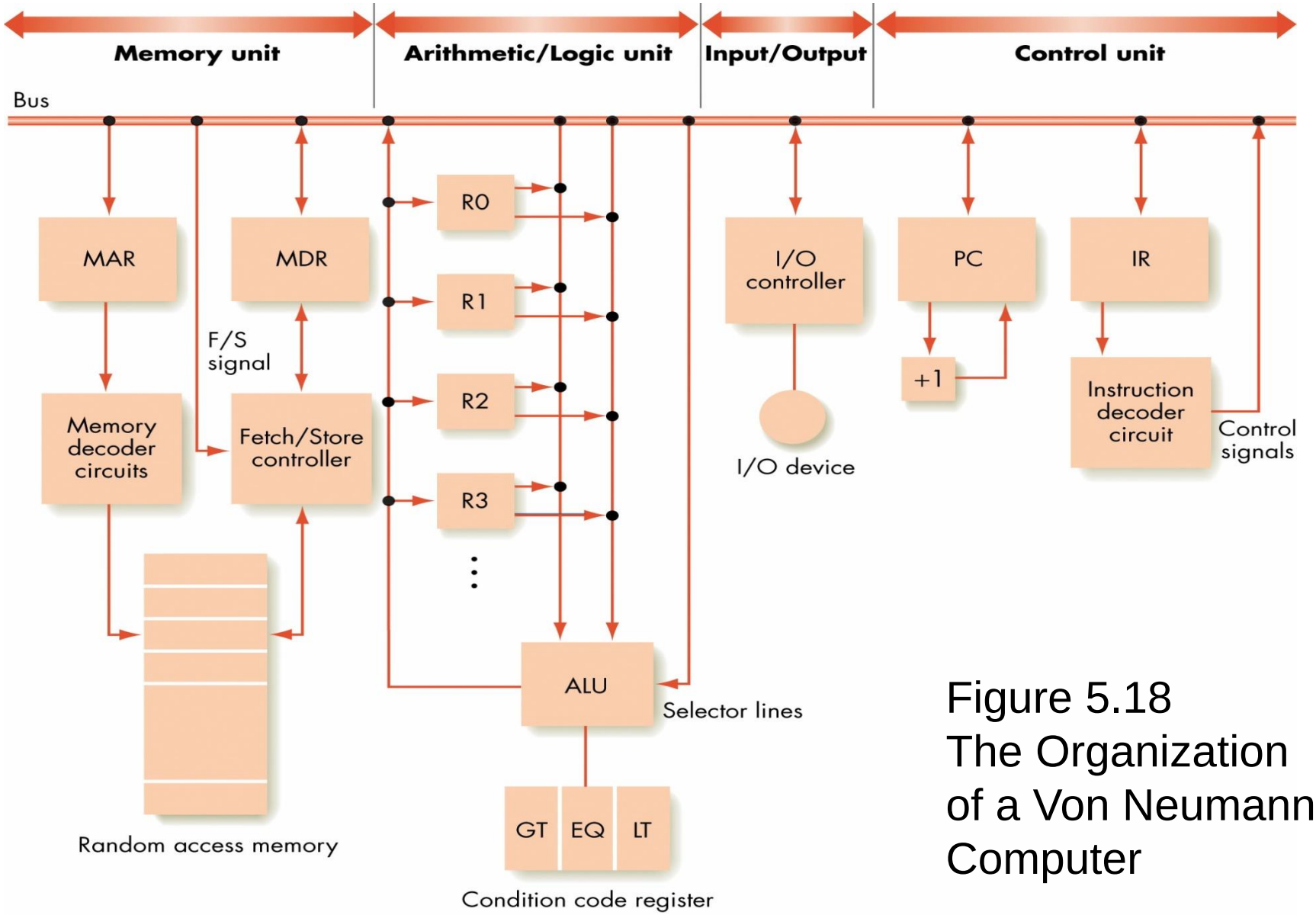


Figure 5.18
The Organization of a Von Neumann Computer

Non-Von Neumann Architectures

- Physical limitations on speed of Von Neumann computers
- Non-Von Neumann architectures explored to bypass these limitations
- Parallel computing architectures can provide improvements; multiple operations occur at the same time

~~Non-Von Neumann Architectures~~

Parallel Architectures

- Physical limitations on speed of Von Neumann computers
- Non-Von Neumann architectures explored to bypass these limitations
- Parallel computing architectures can provide improvements; multiple operations occur at the same time

SIMD Parallel Architectures

- SIMD architecture
  - Single instruction/multiple data
  - Multiple processors running in parallel
  - All processors execute same operation at one time
  - Each processor operates on its own data
  - Suitable for vector operations

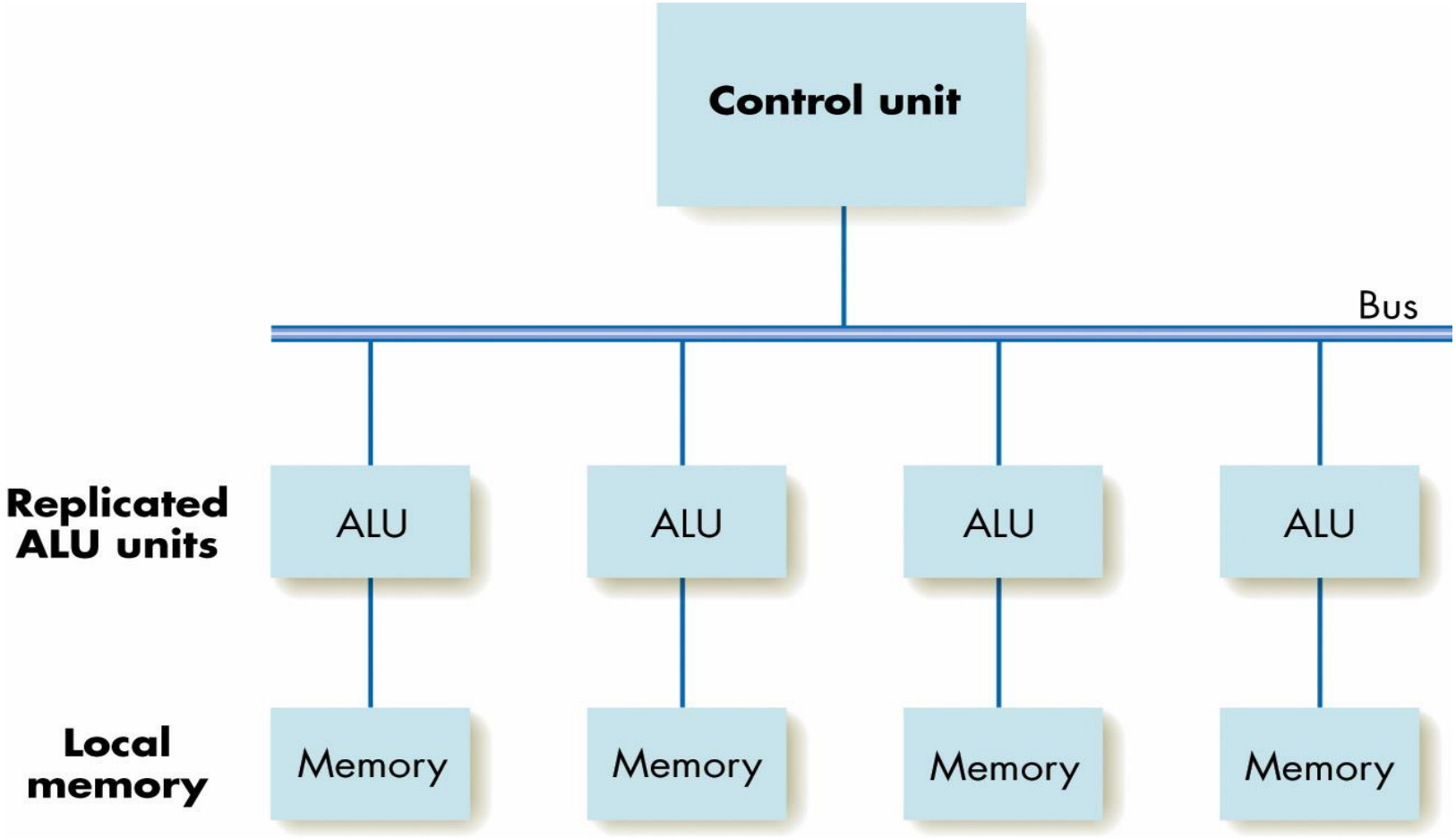


Figure 5.21
A SIMD Parallel Processing System

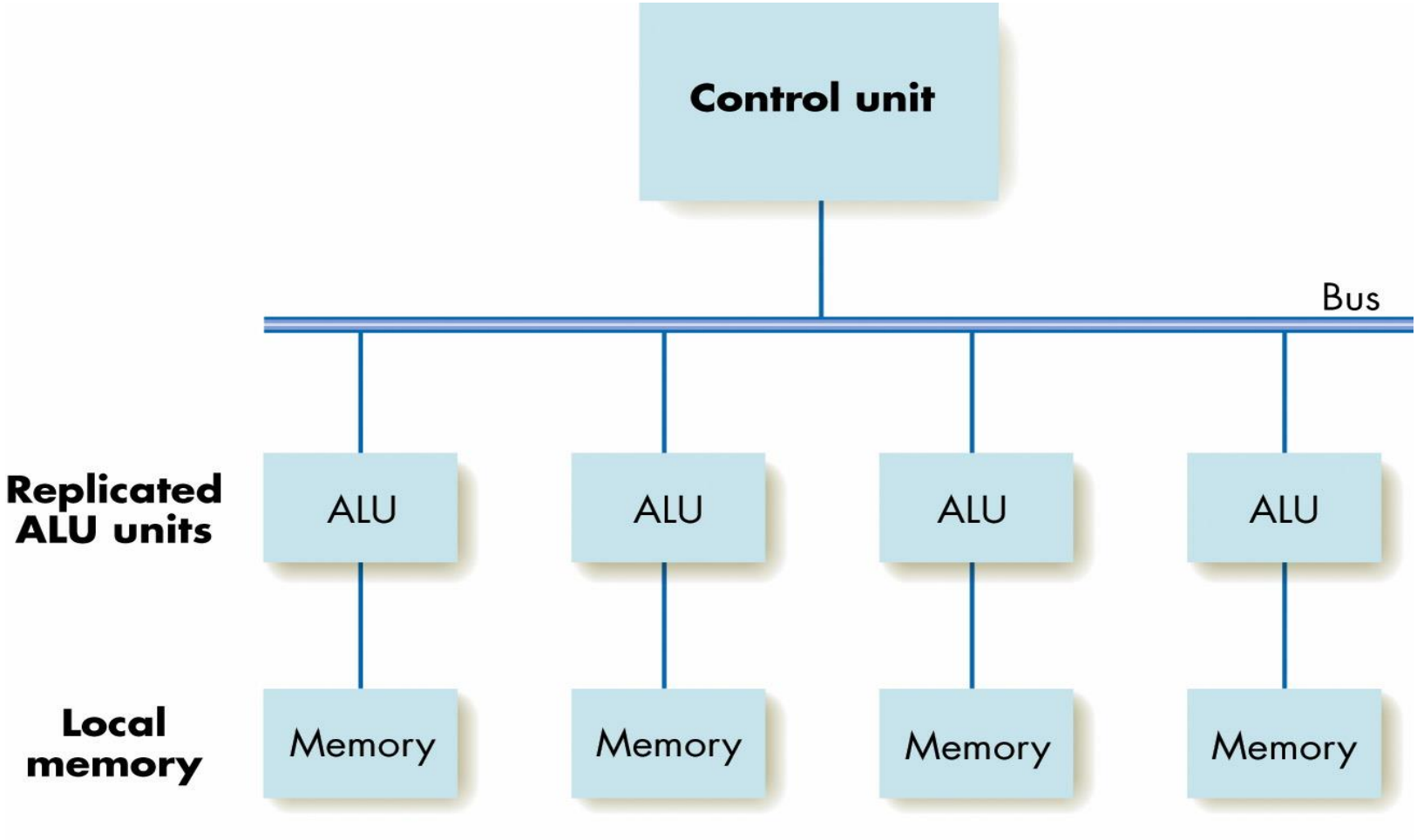
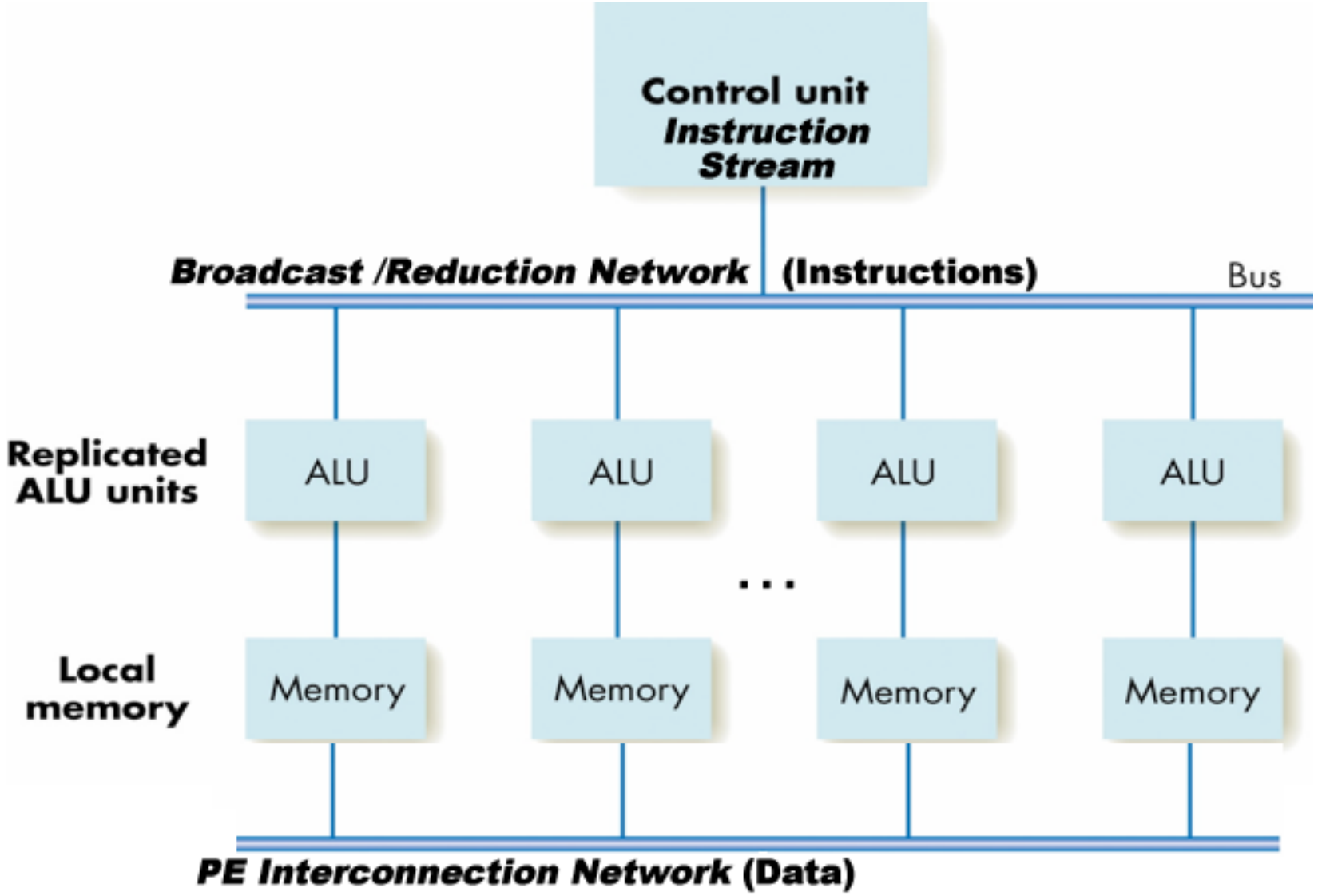


Figure 5.21
A SIMD Parallel Processing System

Associative SIMD

- **Associative Computer:** A SIMD computer with certain additional hardware features.
  - Features can be supported (less efficiently) in software by a traditional SIMD
  - The name “associative” is due to its ability to locate items in the memory of PEs by content rather than location.
    - Uses associative features to simulate an associative memory
- The ASC model (for ASsociative Computing) identifies the properties assumed for an associative computer



An Associative SIMD Parallel Processing System

(Associative components in *italics*)

ASC on Metal

ASC

SIMD with Additional Features

Associative Functions

Associative Search

Search via Content, not Memory Address

Associative Functions

ClearSpeed

"SIMD" Accelerator (64-bit FP)

50 GFLOPS peak performance

25W average power dissipation

NVIDIA Tesla GPGPU

Stream Processor (32-bit FP)

518 Peak GFLOPS on Tesla Series

170W peak, 120W typical

MIMD Parallel Architectures

- MIMD architecture
  - Multiple instruction/multiple data
  - Multiple processors running in parallel
  - Each processor performs its own operations on its own data
  - Processors communicate with each other

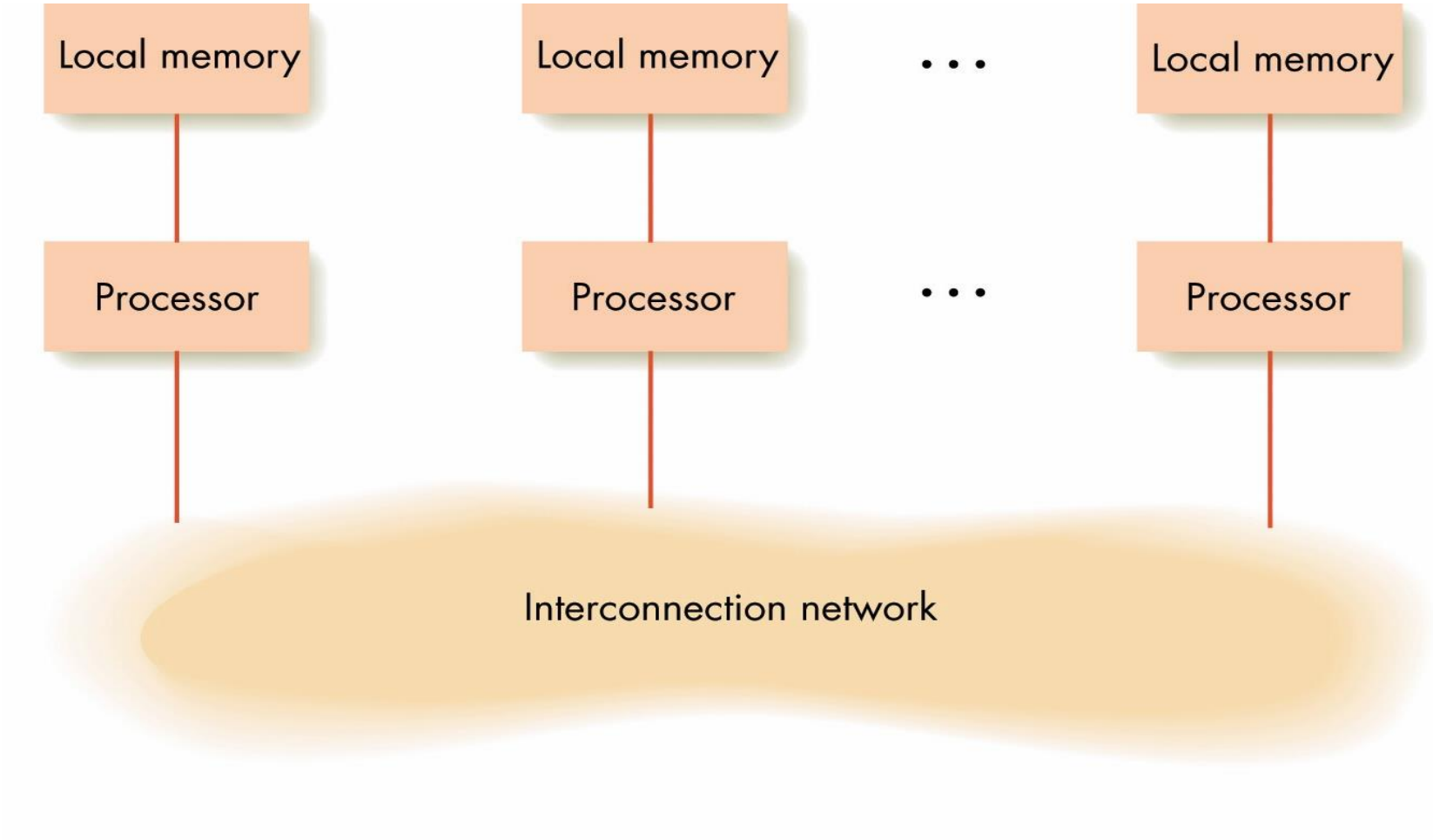


Figure 5.22
Model of MIMD Parallel Processing

More "flavors" of MIMD architectures - Grid Computing

- Grid computing - using homogeneous or heterogeneous systems for a common computational goal
  - Thousands of systems that may be separated by long physical distances
  - A form of **distributed computing** like SETI@home and Folding@home

Grid Computing

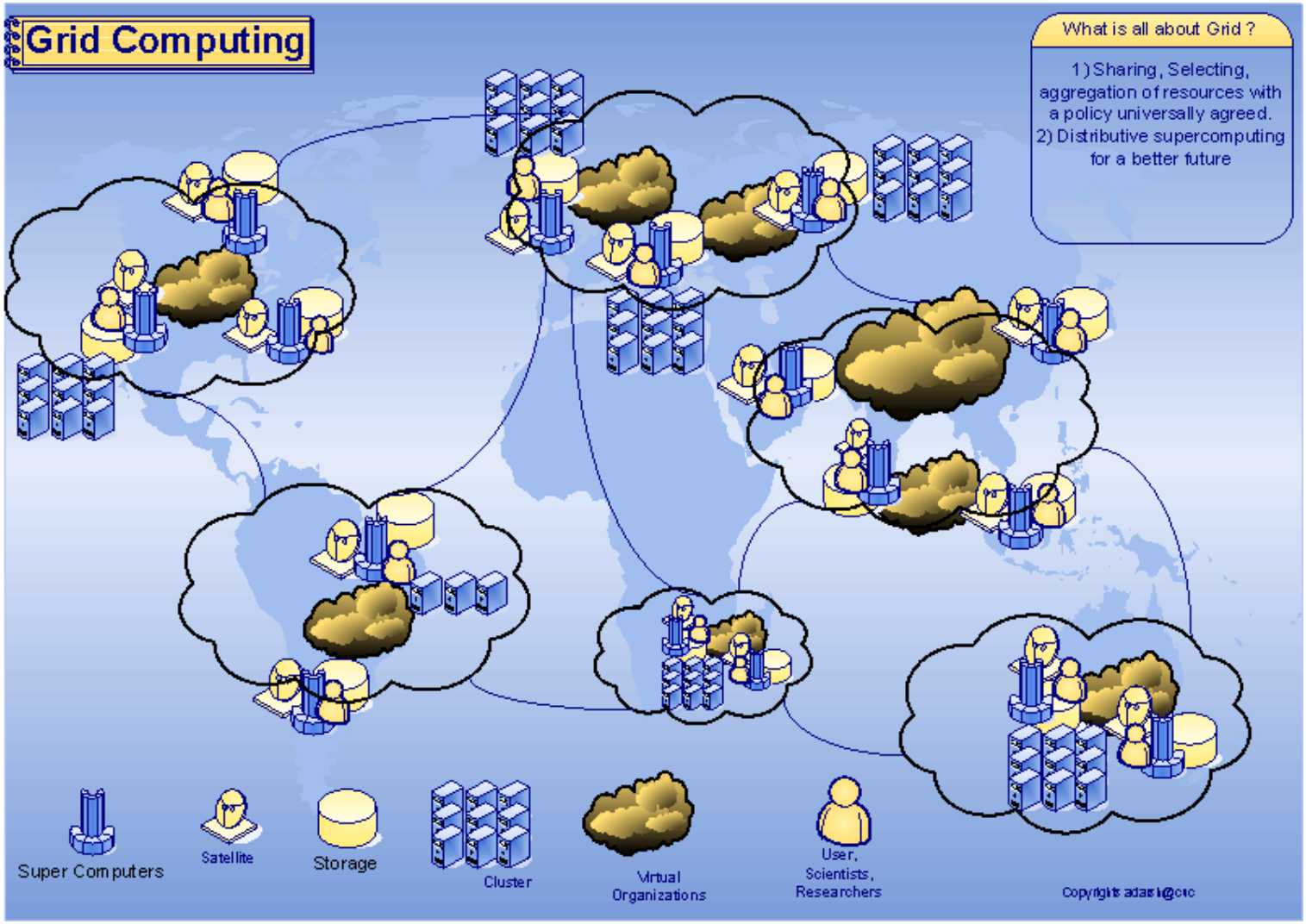


Image source: http://www.adarshpatil.com/newsite/images/grid-computing.gif

More "flavors" of MIMD architectures - Cloud Computing

- Cloud Computing - Internet or "cloud" based development and computing
  - Resources are provided as a service over the Internet.
  - Users need no knowledge of, expertise in, or control over the technology used (and usually won't)
  - Amazon Elastic Compute Cloud (Amazon EC2)
  - Apple's Mobile Me / Microsoft's "Cloud OS" Azure

Cloud Computing…Sort of

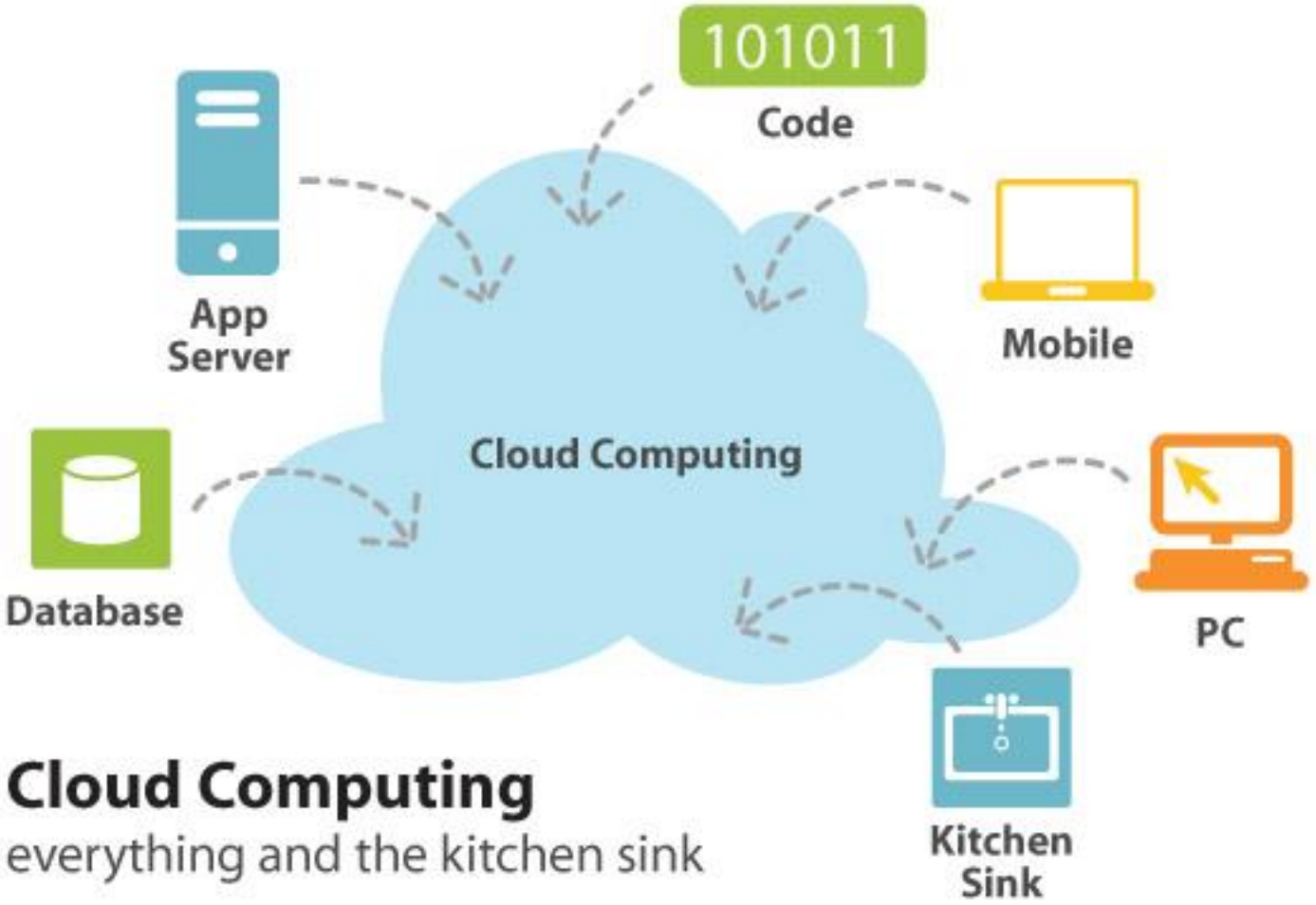


Image Source: http://infreemation.net/cloud-computing-linear-utility-or-complex-ecosystem/

General-purpose Programming on Graphics Processors (GPGPU)

- Hybrid MIMD and SIMD computing
- Uses the computation power of graphics card for non-graphics applications

NVIDIA C870 Tesla GPGPU

- 518 Peak GFLOPS on Tesla
- 170W peak, 120W typical

NVIDIA GeForce 9600GT

Summary of Level 2

- Focus on how to design and build computer systems
- Chapter 4
  - Binary codes
  - Transistors
  - Gates
  - Circuits

Summary of Level 2 (continued)

- Chapter 5
  - Von Neumann architecture
  - Shortcomings of the sequential model of computing
  - Parallel computers

Summary

- Computer organization examines different subsystems of a computer: memory, input/output, arithmetic/logic unit, and control unit
- Machine language gives codes for each primitive instruction the computer can perform and its arguments
- Von Neumann machine: Sequential execution of a stored program
- Parallel computers improve speed by doing multiple tasks at one time