# Chapter 5
## Computer Systems Organization

INVITATION TO
Computer Science

6TH EDITION

# Objectives

After studying this chapter, students will be able to:

- Enumerate the characteristics of the Von Neumann architecture
- Describe the components of a RAM system, including how fetch and store operations work, and the use of cache memory to speed up access time
- Explain why mass storage devices are important, and how DASDs like hard drives or DVDs work
- Diagram the components of a typical ALU and illustrate how the ALU data path operates

# Objectives (continued)

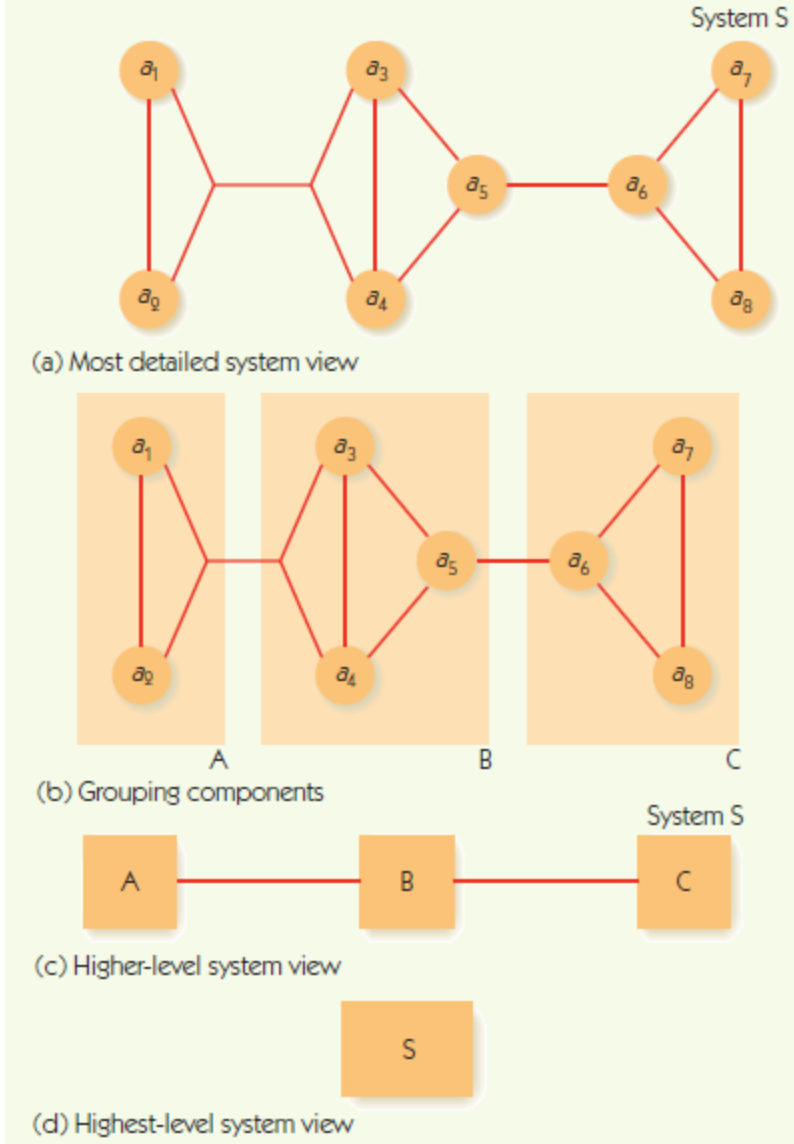
After studying this chapter, students will be able to:

- Describe the control unit's Von Neumann cycle, and explain how it implements a stored program
- List and explain the types of instructions and how they are encoded
- Diagram the components of a typical Von Neumann machine
- Show the sequence of steps in the fetch, decode, and execute cycle to perform a typical instruction
- Describe non-Von Neumann parallel processing systems

# Introduction

- This chapter changes the **level of abstraction**
- Focus on **functional units** and **computer organization**
- A **hierarchy of abstractions** hides unneeded details
- Change focus from transistors, to gates, to circuits as the basic unit

**FIGURE 5.1**



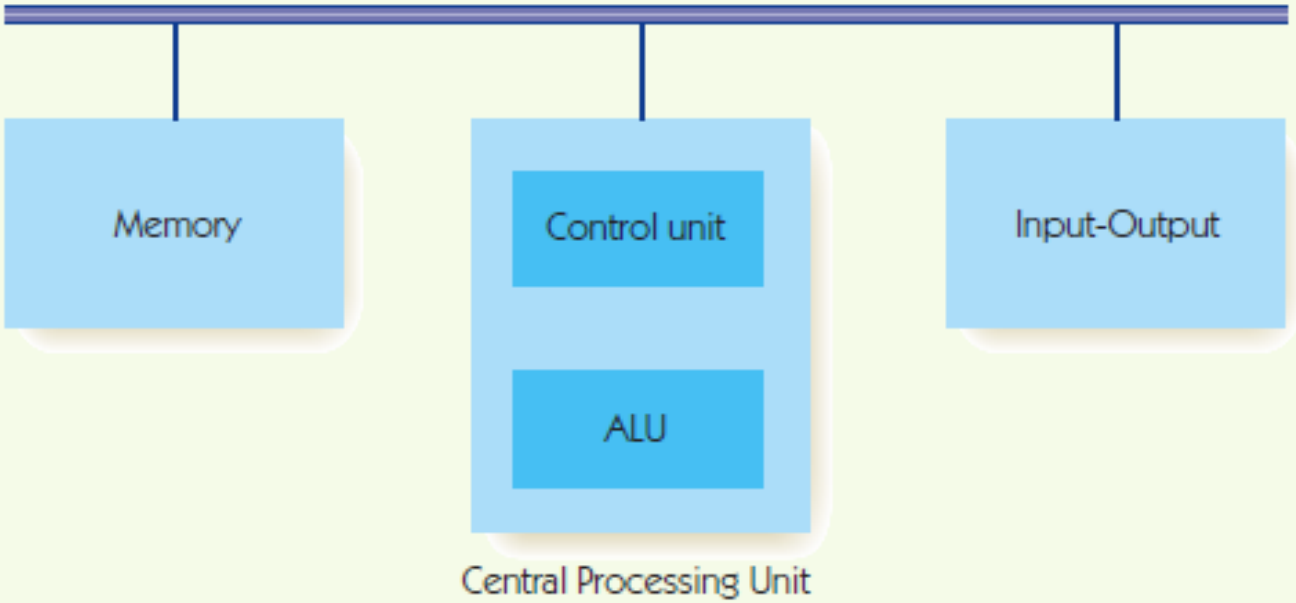

The concept of abstraction

# The Components of a Computer System

**Von Neumann architecture**

- Foundation for nearly all modern computers
- Characteristics:
  - **Central Processing Unit (CPU)**
    - memory
    - input/output
    - arithmetic/logic unit
    - control unit
  - Stored program concept
  - Sequential execution of instructions

FIGURE 5.2

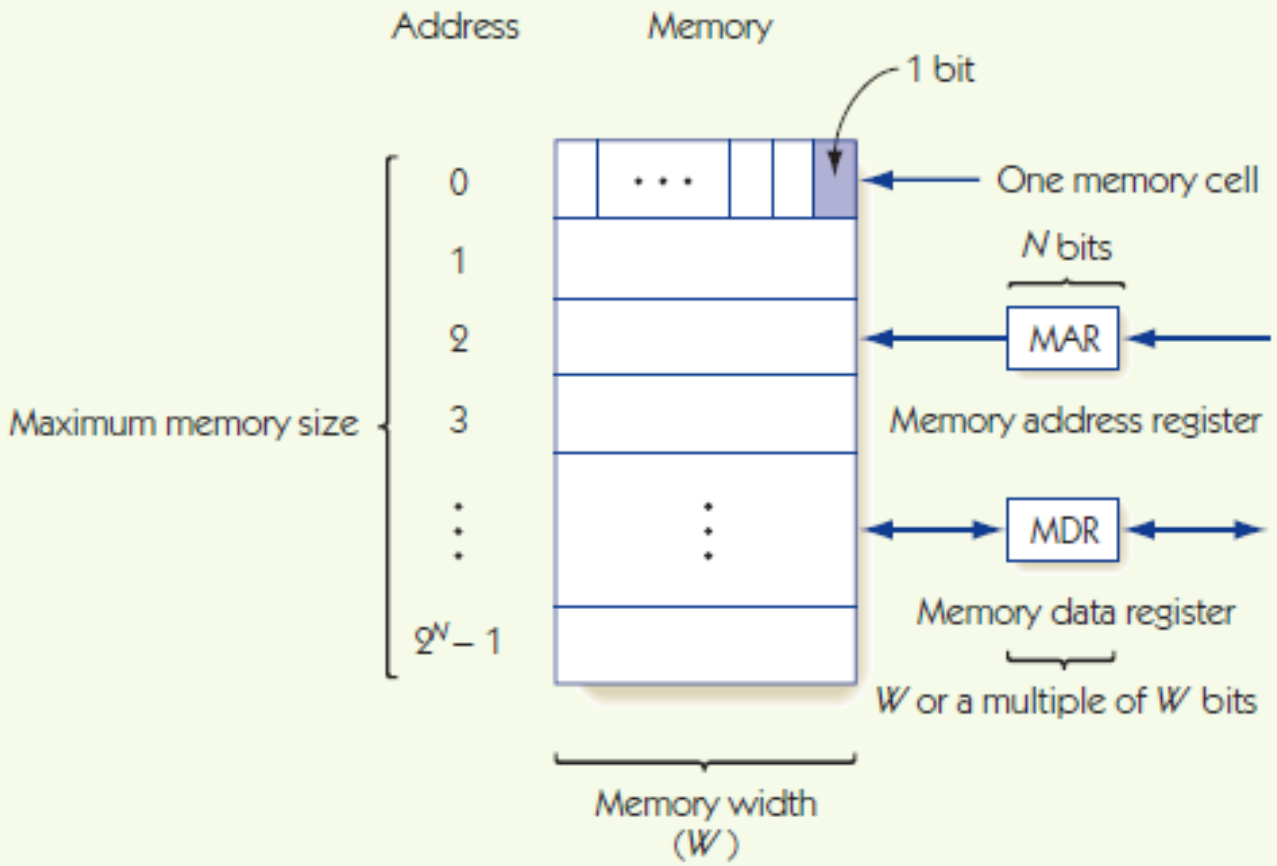



Components of the Von Neumann architecture

# The Components of a Computer System Memory and Cache (continued)

- **Memory**: functional unit where data is stored/ retrieved
- **Random access memory (RAM)**
  - Organized into **cells**, each given a unique **address**
  - Equal time to access any cell
  - Cell values may be read and changed
- **Cell size/memory width** typically 8 bits
- **Maximum memory size/address space** is $2^N$, where N is length of address

FIGURE 5.3




Structure of random access memory

FIGURE 5.4

| N | Maximum Memory Size ($2^N$) |
|---|---|
| 16 | 65,536 |
| 20 | 1,048,576 |
| 22 | 4,194,304 |
| 24 | 16,777,216 |
| 32 | 4,294,967,296 |
| 40 | 1,099,511,627,776 |
| 50 | 1,125,899,906,842,624 |

Maximum memory sizes

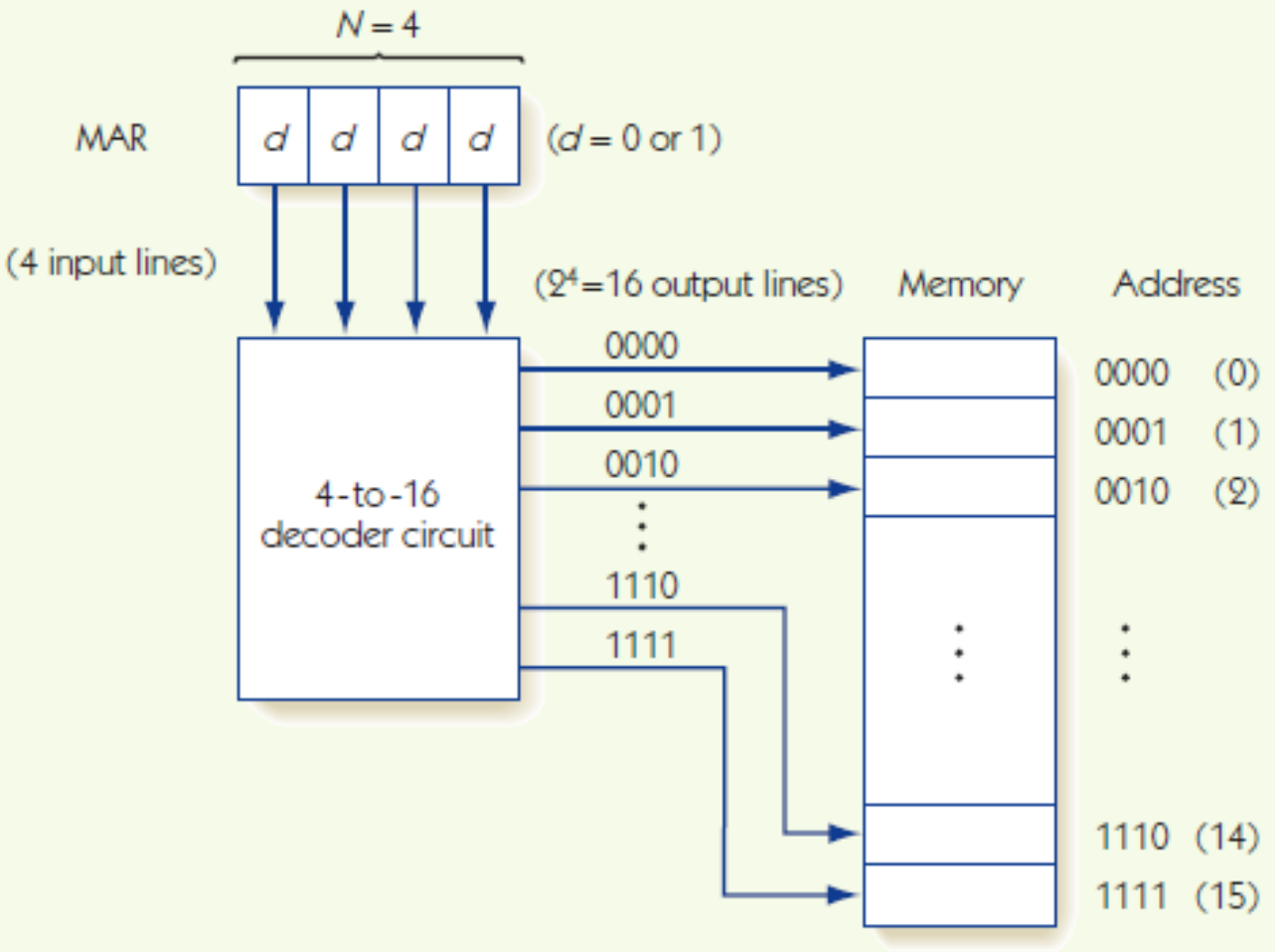
# The Components of a Computer System Memory and Cache (continued)

- Fetch: retrieve from memory (**nondestructive fetch)**
- Store: write to memory (**destructive store)**
- **Memory access time**
  - time required to fetch/store
  - Modern RAM requires 5-10 **nanoseconds**
- **MAR** holds memory address to access
- **MDR** receives data from fetch, holds data to be stored

# The Components of a Computer System Memory and Cache (continued)

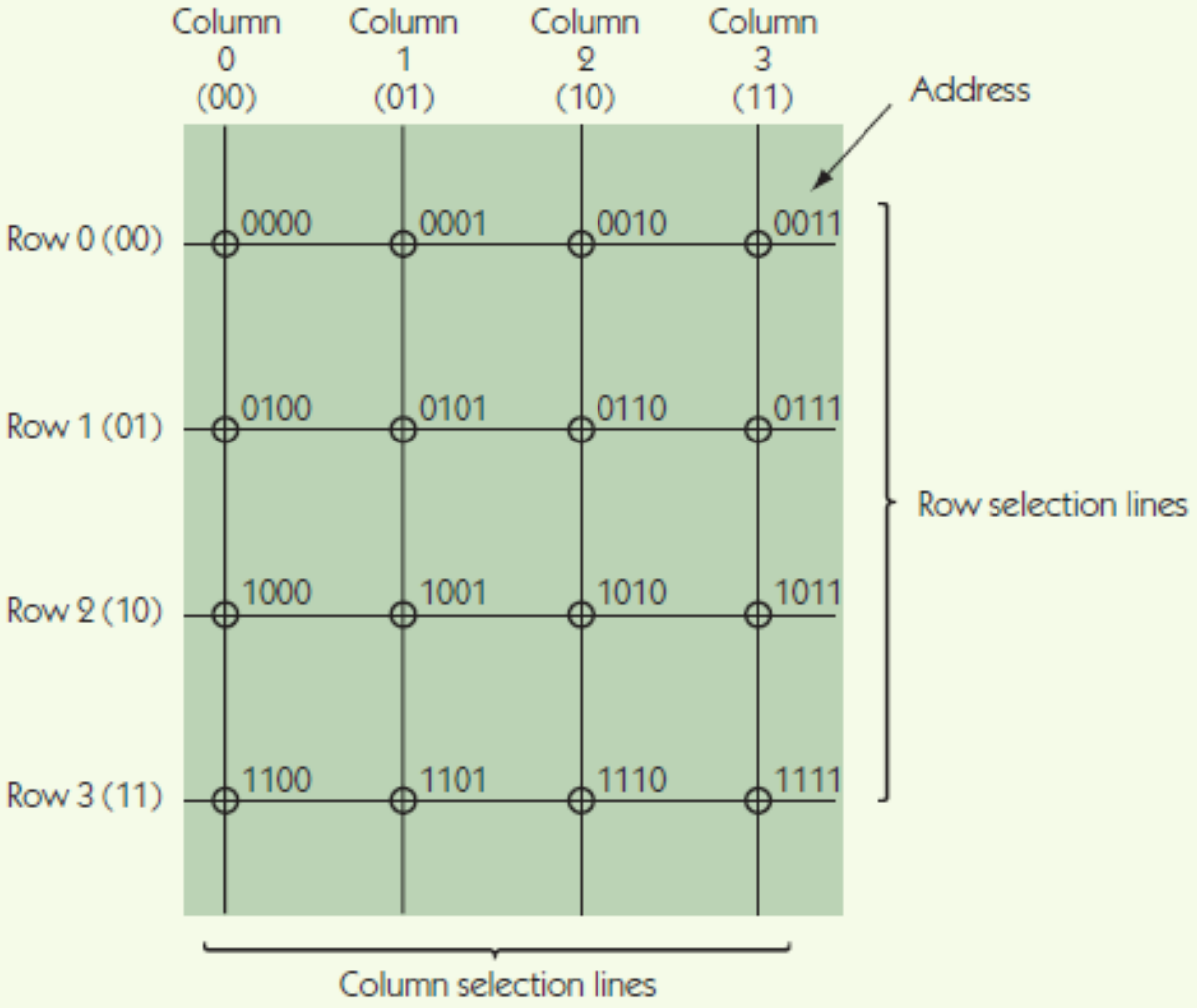
- Memory system circuits: decoder and fetch/store controller
- Decoder converts MAR into signal to a specific memory cell
  - One-dimensional versus two-dimensional memory organization
- Fetch/Store controller = traffic cop for MDR
  - Takes in a signal that indicates fetch or store
  - Routes data flow to/from memory cells and MDR

**FIGURE 5.5**



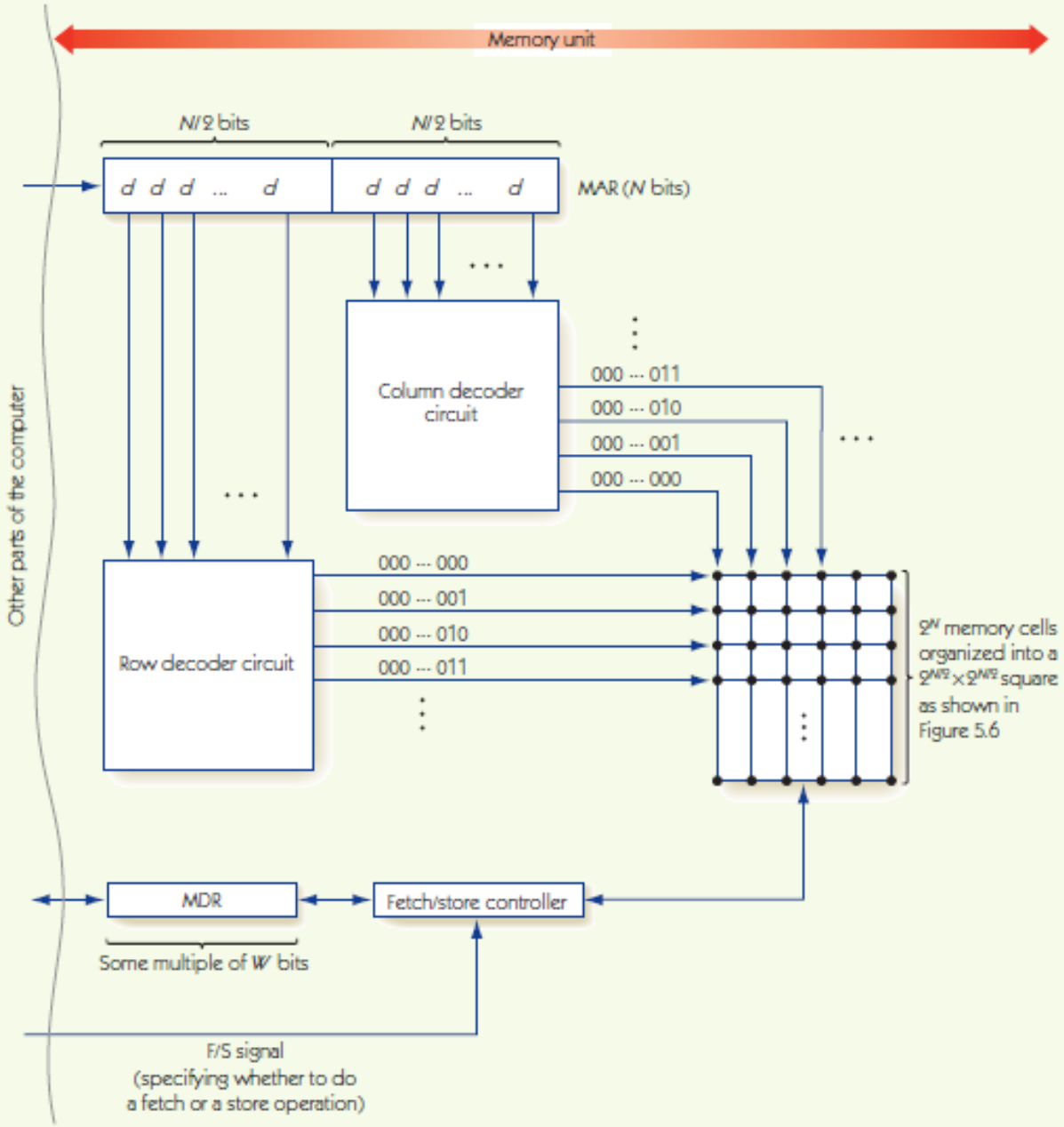

Organization of memory and the decoding logic

**FIGURE 5.6**




Two-dimensional memory organization

FIGURE 5.7




Overall RAM organization

# The Components of a Computer System Memory and Cache (continued)

- RAM speeds increased more slowly than CPU speeds
- **Cache memory** is fast but expensive
- **Principle of locality:**
  - Values close to recently-accessed memory are more likely to be accessed
  - Load neighbors into cache and keep recent there
- **Cache hit rate:** percentage of times values are found in cache

# The Components of a Computer System Input/Output and Mass Storage

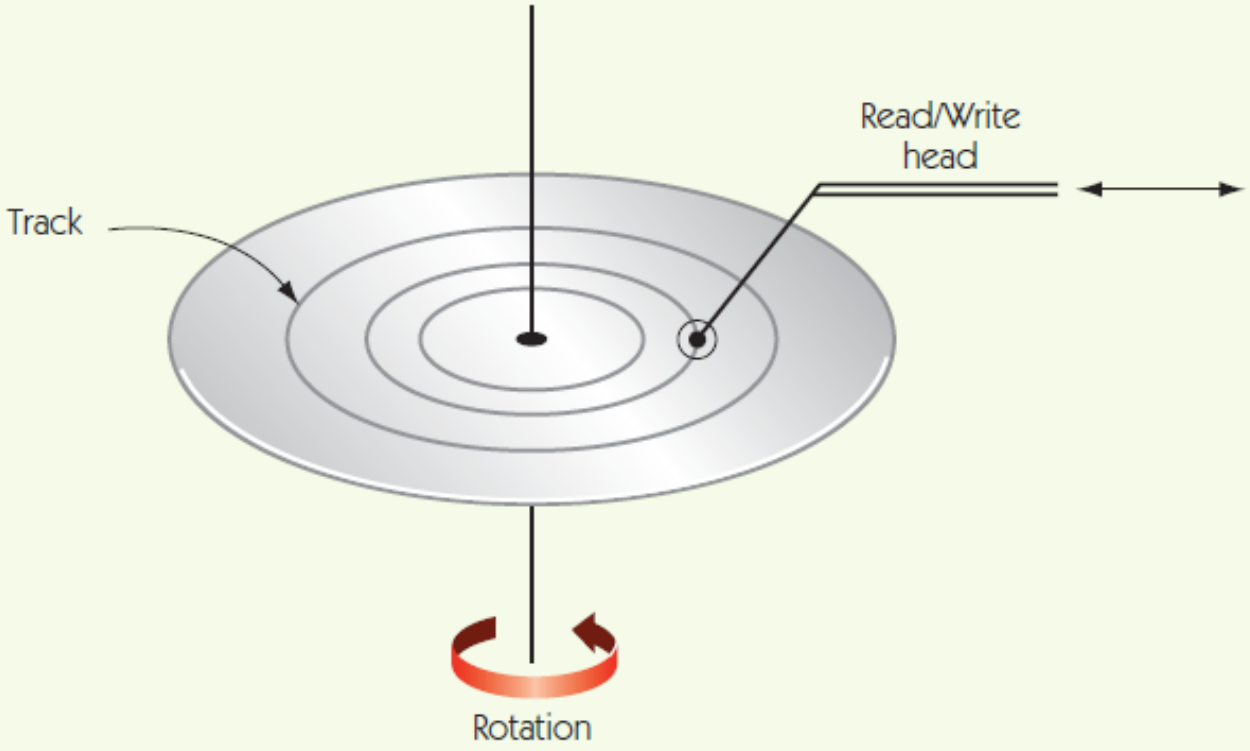
- **Input/Output (I/O)** connects the processor to the outside world:
  - Humans: keyboard, monitor, etc.
  - Data storage: hard drive, DVD, flash drive
  - Other computers: network
- RAM = **volatile memory** (gone without power)
- **Mass storage systems** = **nonvolatile memory**
  - **Direct access storage devices (DASDs)**
  - **Sequential access storage devices (SASDs)**

# The Components of a Computer System I/O and Mass Storage (continued)

**DASDs**

- Hard drives, CDs, DVDs contain disks
  - **Tracks:** concentric rings around disk surface
  - **Sectors:** fixed size segments of tracks, unit of retrieval
  - Time to retrieve data based on:
    - **seek time**
    - **latency**
    - **transfer time**
- Other non-disk DASDs: flash memory, optical

FIGURE 5.8

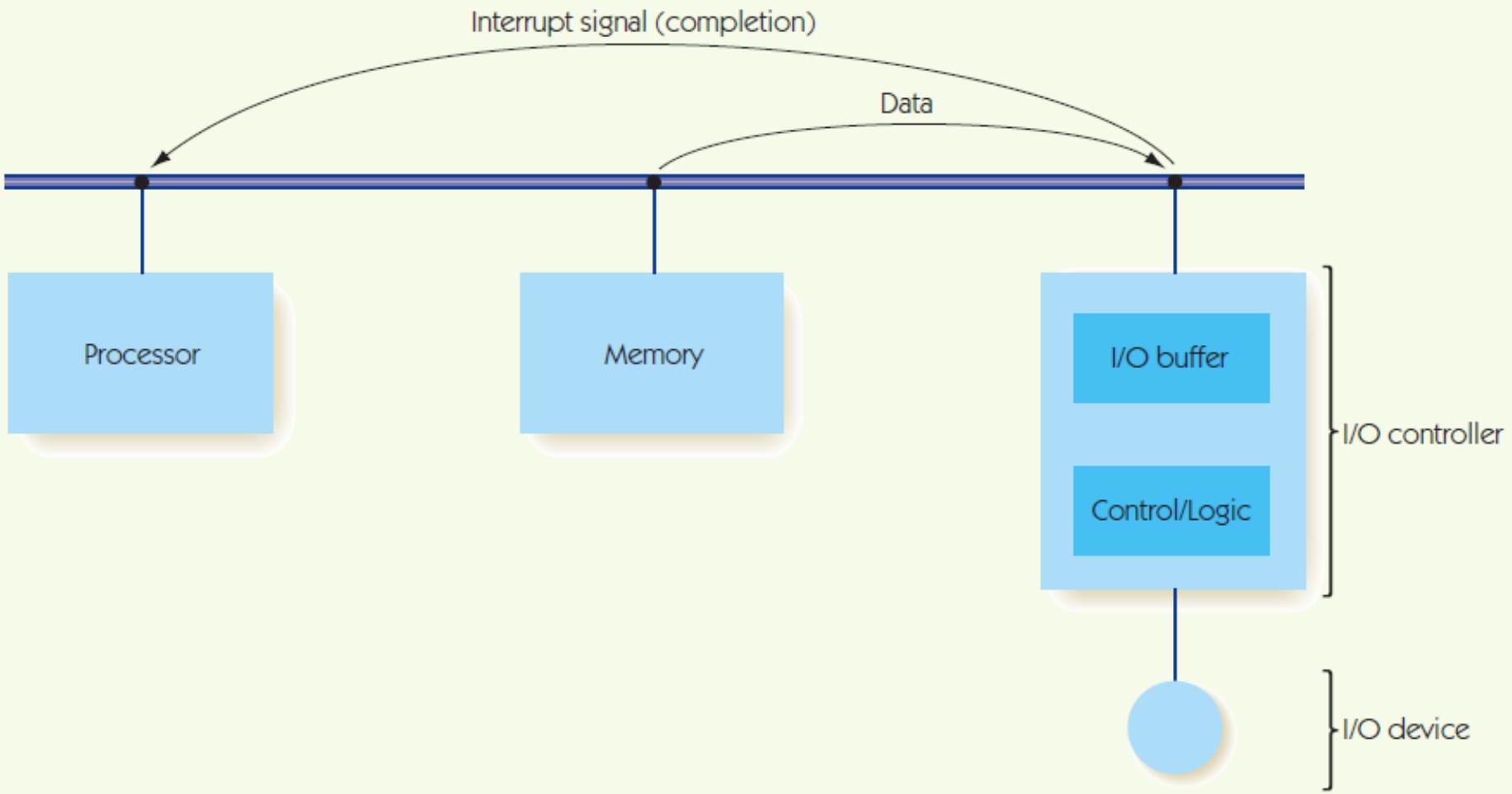



Overall organization of a typical disk

# The Components of a Computer System I/O and Mass Storage (continued)

- DASDs and SASDs are orders of magnitude slower than RAM: (microseconds or milliseconds)
- **I/O Controller** manages data transfer with slow I/O devices, freeing processor to do other work
- Controller sends an **interrupt signal** to processor when I/O task is done

FIGURE 5.9

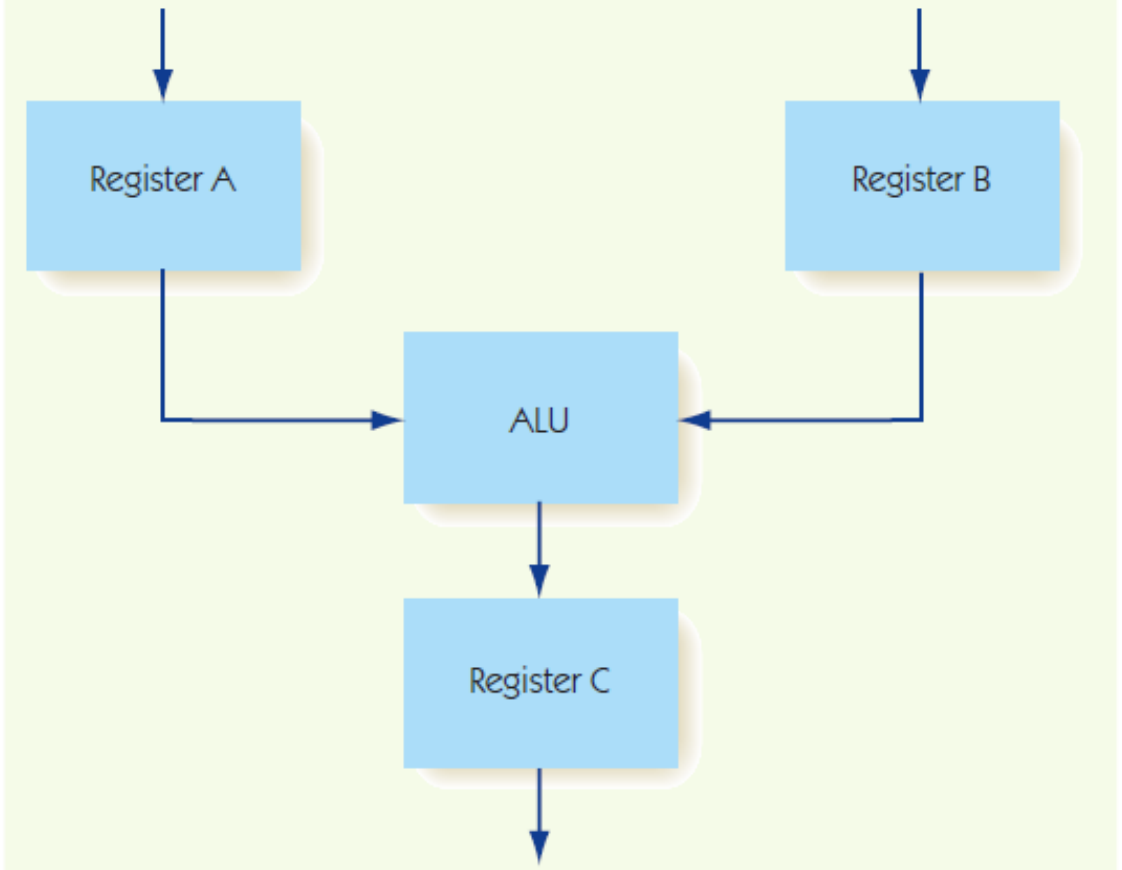



Organization of an I/O controller

# The Components of a Computer System The Arithmetic/Logic Unit

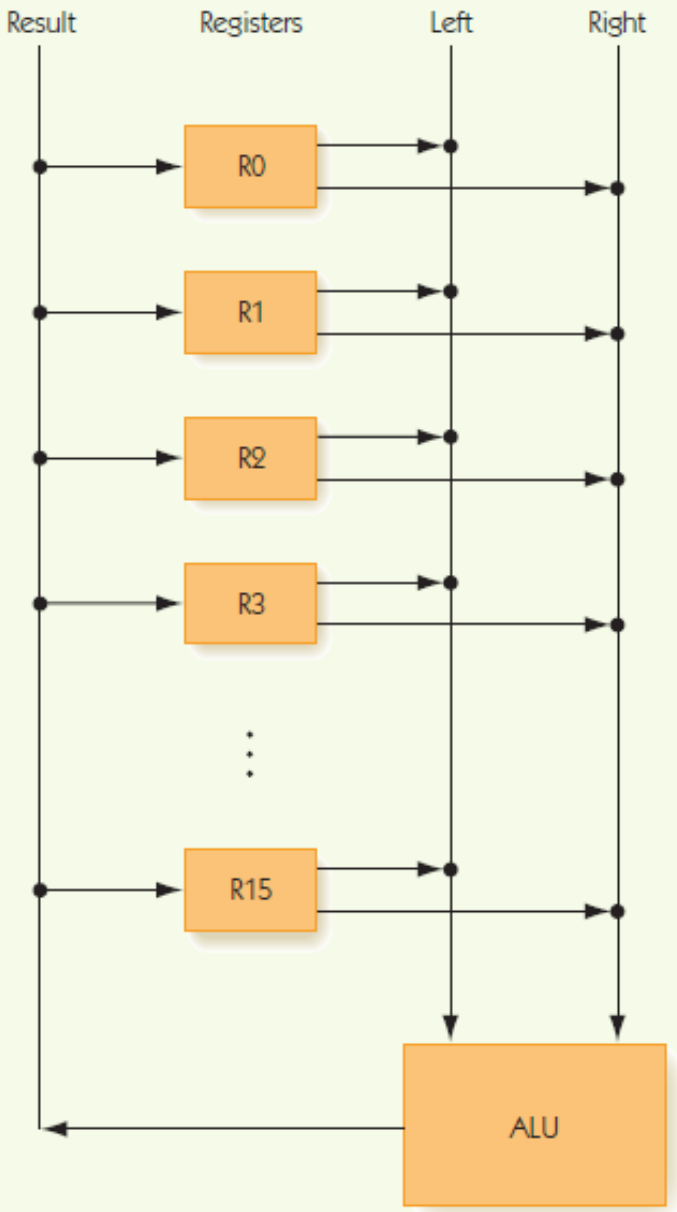
- ALU is part of **processor**
- Contains circuits for arithmetic:
  - addition, subtraction, multiplication, division
- Contains circuits for comparison and logic:
  - equality, and, or, not
- Contains **registers**: super-fast, dedicated memory connected to circuits
- **Data path:** how information flows in ALU
  - from registers to circuits
  - from circuits back to registers

FIGURE 5.10




Three-register ALU organization

FIGURE 5.11



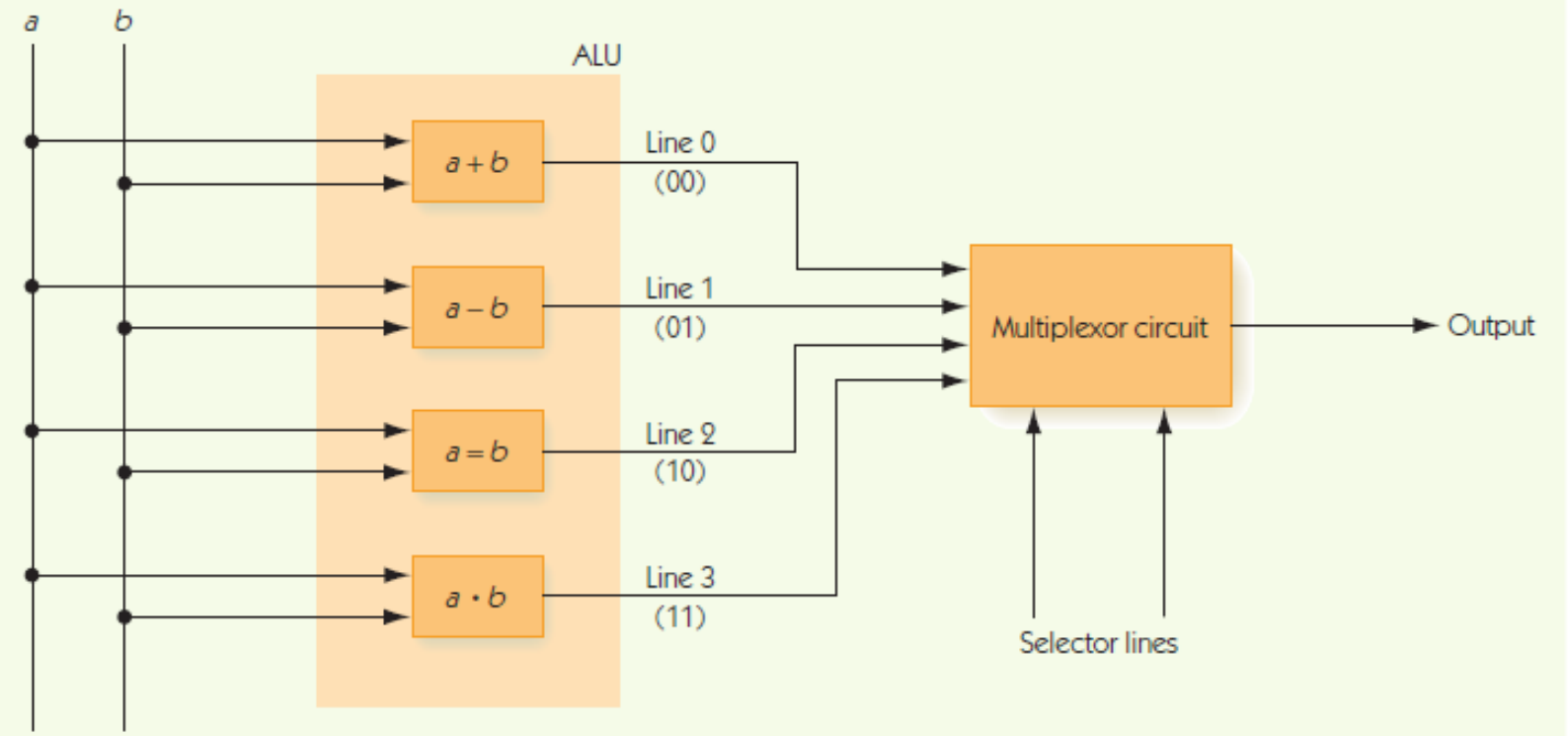

Multiregister ALU organization

# The Components of a Computer System The ALU (continued)

- How to choose which operation to perform?
  - Option 1:  decoder signals one circuit to run
  - Option 2: run all circuits, multiplexor selects one output from all circuits
- In practice, option 2 is usually chosen

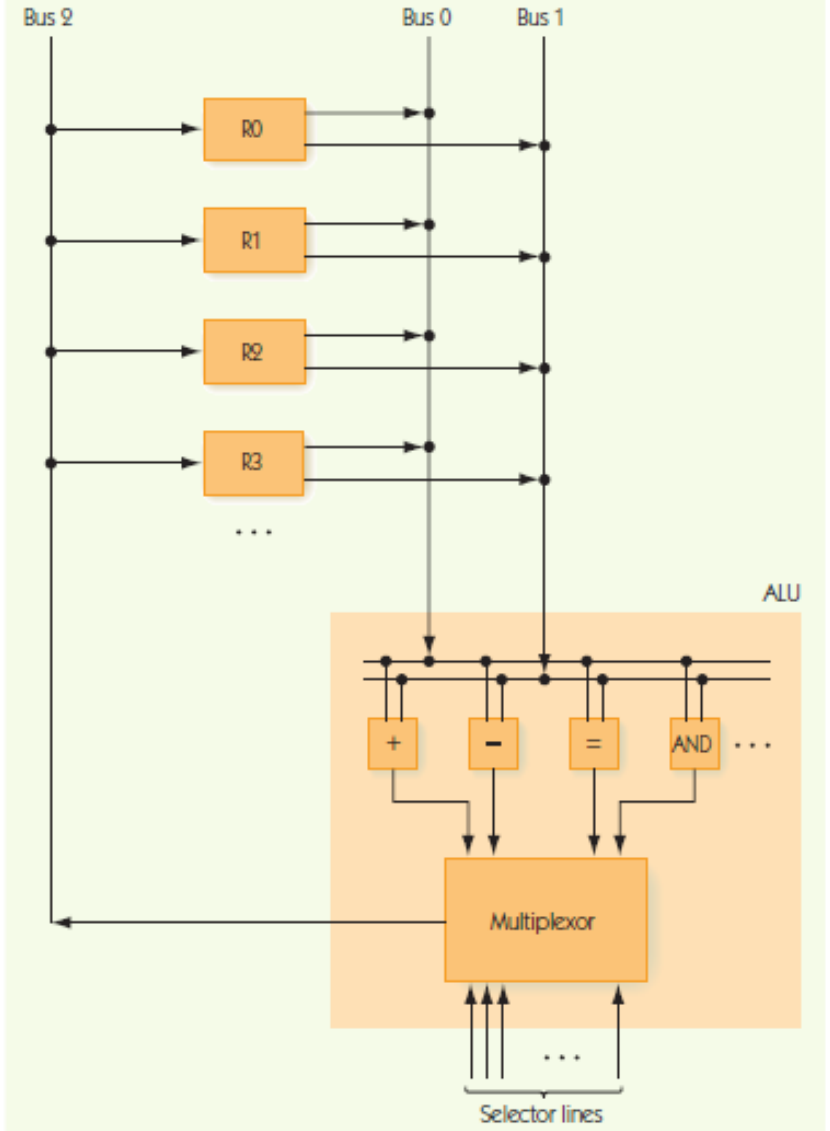
- Information flow:
  - Data comes in from outside ALU to registers
  - Signal comes to multiplexor, which operation
  - Result goes back to register, and then to outside

FIGURE 5.12




Using a multiplexor circuit to select the proper ALU result

FIGURE 5.13




Overall ALU organization

# The Components of a Computer System The Control Unit

- **Stored program** characteristic:
  - Programs are encoded in binary and stored in computer's memory
- **Control unit** fetches instructions from memory, decodes them, and executes them
- Instructions encoded:
  - Operation code (op code) tells which operation
  - Addresses tell which memory addresses/registers to operate on

**FIGURE 5.14**


Typical machine language instruction format

# The Components of a Computer System The Control Unit (continued)

- **Machine language**:
  - Binary strings that encode instructions
  - Instructions can be carried out by hardware
  - Sequences of instructions encode algorithms
- **Instruction set**:
  - The instructions implemented by a particular chip
  - Each kind of processor speaks a different language

# The Components of a Computer System The Control Unit (continued)

- **RISC machines** and **CISC machines**
  - Reduced instruction set computers:
    - small instruction sets
    - each instruction highly optimized
    - easy to design hardware
  - Complex instruction set computers:
    - large instruction set
    - single instruction can do a lot of work
    - complex to design hardware
- Modern hardware: some RISC and some CISC

# The Components of a Computer System The Control Unit (continued)

Kinds of instructions:

- Data transfer, e.g., move data from memory to register
- Arithmetic, e.g., add, but also “and”
- Comparison, compare two values
- Branch, change to a non-sequential instruction
  - Branching allows for conditional and loop forms
  - E.g., JUMPLT *a* = If previous comparison of A and B found A < B, then jump to instruction at address *a*

**FIGURE 5.15**

| Address | Contents |
|---|---|
| 100 | Value of *a* |
| 101 | Value of *b* |
| 102 | Value of *c* |

| Algorithmic notation | Address | Contents | (Commentary) |
|---|---|---|---|
| | | ⋮ | |
| 1. Set *a* to the value *b* + *c* | 50 | LOAD 101 | Put the value of *b* into register R. |
| | 51 | ADD 102 | Add *c* to register R. It now holds *b* + *c*. |
| | 52 | STORE 100 | Store the contents of register R into *a*. |
| | | | |
| | | | |
| 2. If *a* > *b* then | 50 | COMPARE 100, 101 | Compare *a* and *b* and set condition codes. |
| set *c* to the value *a* | 51 | JUMPGT 54 | Go to location 54 if *a* > *b*. |
| Else | 52 | MOVE 101, 102 | Get here if *a* ≤ *b*, so move *b* into *c* |
| set *c* to the value *b* | 53 | JUMP 55 | and skip the next instruction. |
| | 54 | MOVE 100, 102 | Move *a* into *c*. |
| | 55 | • • • | Next statement begins here. |
| | | | |

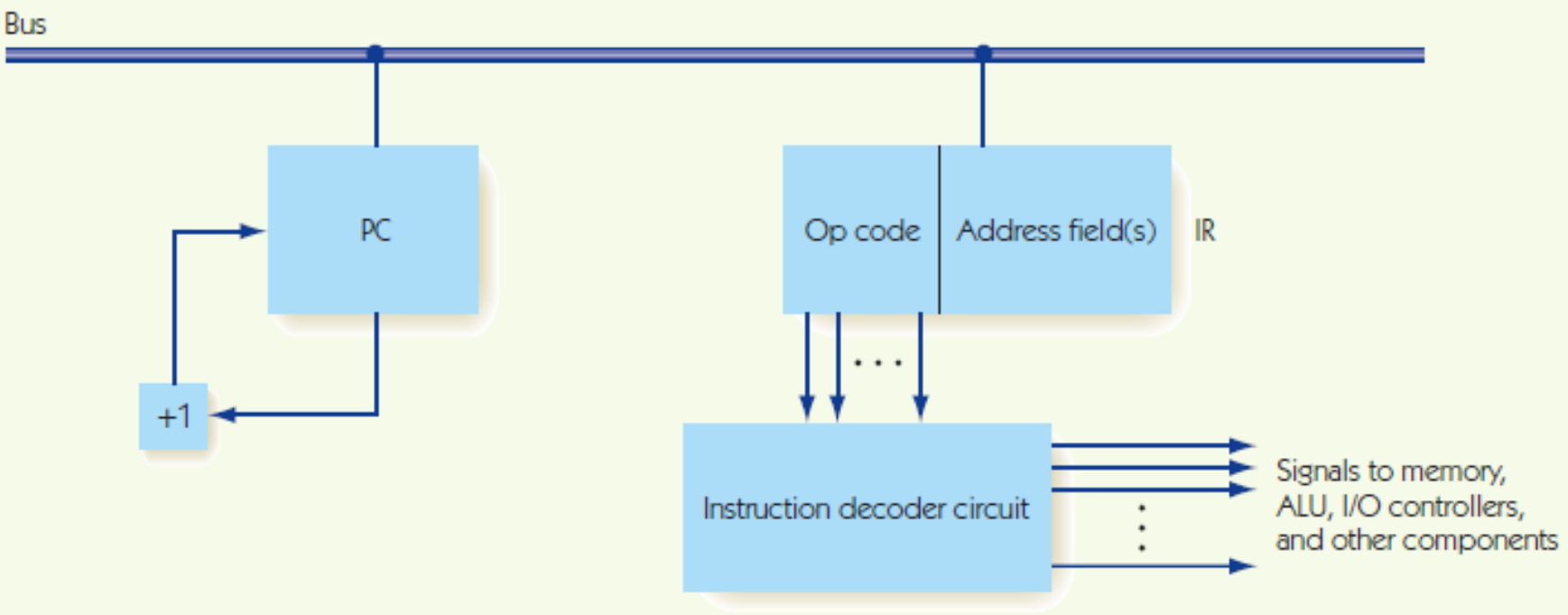
Examples of simple machine language instruction sequences

# The Components of a Computer System The Control Unit (continued)

Control unit contains:

- **Program counter (PC)** register: holds address of next instruction
- **Instruction register (IR)**: holds encoding of current instruction
- Instruction decoder circuit
  - Decodes op code of instruction, and signals helper circuits, one per instruction
    - Helpers send addresses to proper circuits
    - Helpers signal ALU, I/O controller, memory

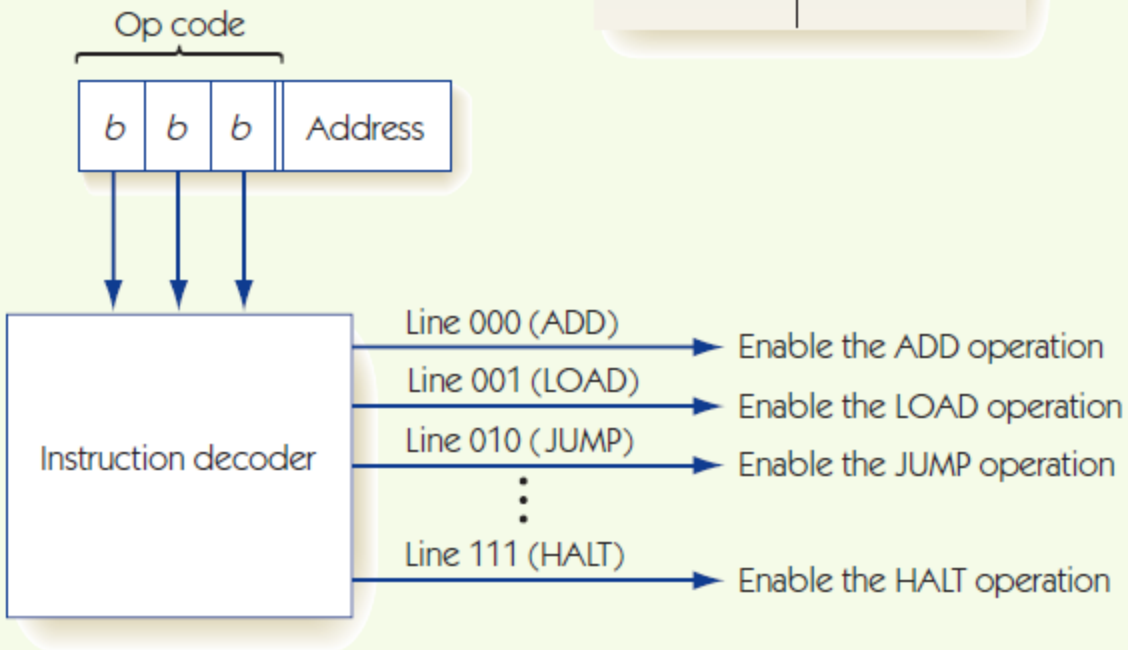
FIGURE 5.16




Organization of the control unit registers and circuits

FIGURE 5.17

| Op code | Instruction |
|---|---|
| 000 | ADD |
| 001 | LOAD |
| 010 | JUMP |
| ⋮ | ⋮ |
| 111 | HALT |

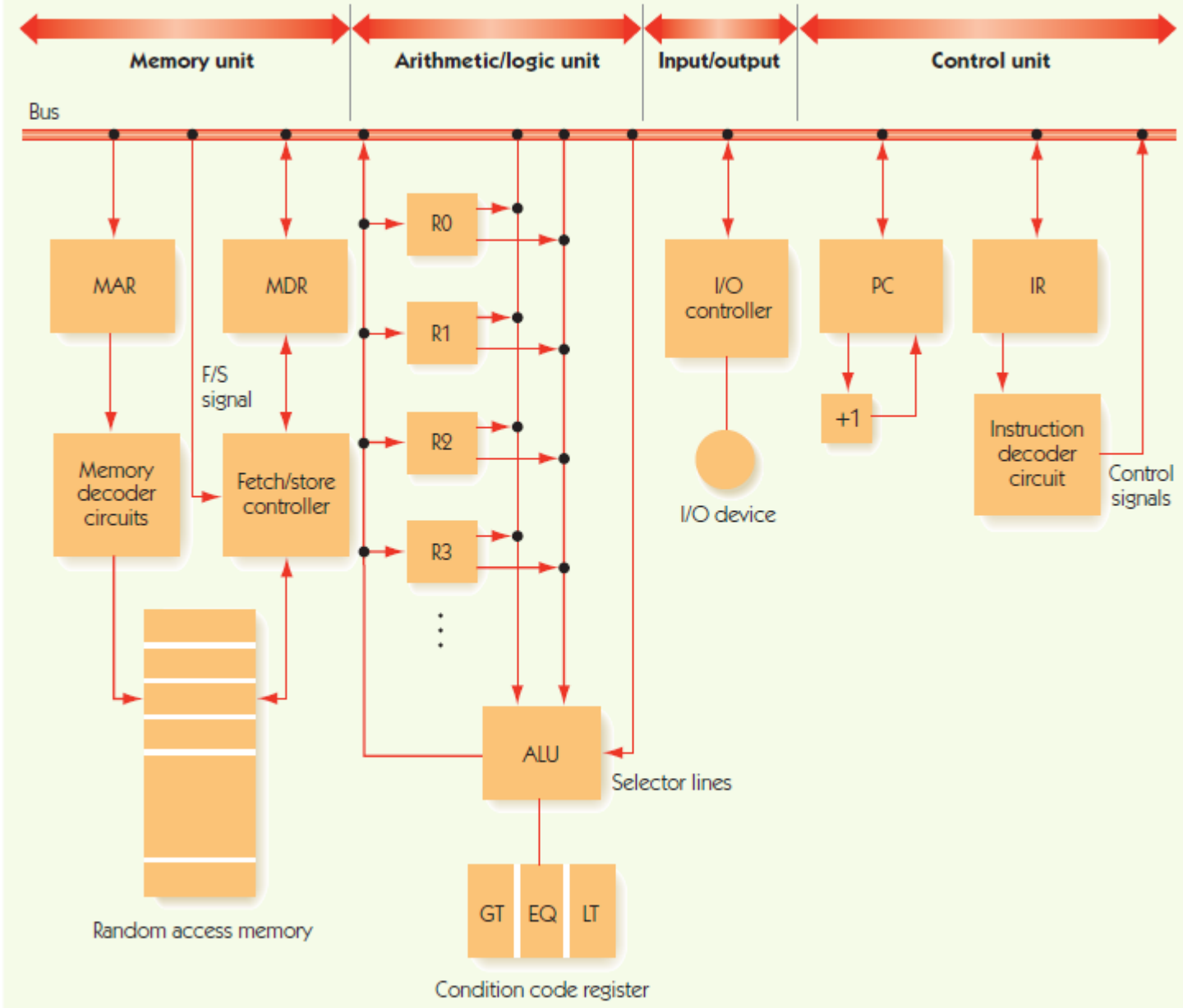



The instruction decoder

# Putting the Pieces Together---the Von Neumann Architecture

- Combine previous pieces: Von Neumann machine
- Fetch-Decode-Execute cycle
  - machine repeats until HALT instruction or error
  - also called Von Neumann cycle
- Fetch phase: get next instruction into memory
- Decode phase: instruction decoder gets op code
- Execute phase: different for each instruction

FIGURE 5.18




The organization of a Von Neumann computer

# Putting the Pieces Together---the Von Neumann Architecture (continued)

Notation for computer's behavior:

| | |
|---|---|
| CON(A) | Contents of memory cell A |
| A -> B | Send value in register A to register B (special registers: PC, MAR, MDR, IR, ALU, R, GT, EQ, LT, +1) |
| FETCH | Initiate a memory fetch operation |
| STORE | Initiate a memory store operation |
| ADD | Instruct the ALU to select the output of the adder circuit |
| SUBTRACT | Instruct the ALU to select the output of the subtract circuit |

# Putting the Pieces Together---the Von Neumann Architecture (continued)

Fetch phase:

| | |
|---|---|
| 1. PC -> MAR | Send address in PC to MAR |
| 2. FETCH | Initiate Fetch, data to MDR |
| 3. MDR -> IR | Move instruction in MDR to IR |
| 4. PC + 1 -> PC | Add one to PC |

Decode phase:

1. $IR_{op}$ -> instruction decoder

FIGURE 5.19

| Binary Op Code | Operation | Meaning |
|---|---|---|
| 0000 | LOAD X | CON(X) → R |
| 0001 | STORE X | R → CON(X) |
| 0010 | CLEAR X | 0 → CON(X) |
| 0011 | ADD X | R + CON(X) → R |
| 0100 | INCREMENT X | CON(X) + 1 → CON(X) |
| 0101 | SUBTRACT X | R – CON(X) → R |
| 0110 | DECREMENT X | CON(X) – 1 → CON(X) |
| 0111 | COMPARE X | if CON(X) > R then GT = 1 else 0<br>if CON(X) = R then EQ = 1 else 0<br>if CON(X) < R then LT = 1 else 0 |
| 1000 | JUMP X | Get the next instruction from memory location X. |
| 1001 | JUMPGT X | Get the next instruction from memory location X if GT = 1. |
| 1010 | JUMPEQ X | Get the next instruction from memory location X if EQ = 1. |
| 1011 | JUMPLT X | Get the next instruction from memory location X if LT = 1. |
| 1100 | JUMPNEQ X | Get the next instruction from memory location X if EQ = 0. |
| 1101 | IN X | Input an integer value from the standard input device and store into memory cell X. |
| 1110 | OUT X | Output, in decimal notation, the value stored in memory cell X. |
| 1111 | HALT | Stop program execution. |

Instruction set for our Von Neumann machine

# Putting the Pieces Together---the Von Neumann Architecture (continued)

LOAD X meaning CON(X) -> R

1. $IR_{addr}$ -> MAR — Send address X to MAR
2. FETCH — Initiate Fetch, data to MDR
3. MDR -> R — Move data in MDR to R

STORE X meaning R -> CON(X)

1. $IR_{addr}$ -> MAR — Send address X to MAR
2. R -> MDR — Send data in R to MDR
3. STORE — Initiate store of MDR to X

# Putting the Pieces Together---the Von Neumann Architecture (continued)

ADD X meaning R + CON(X) -> R

1. $IR_{addr}$ -> MAR Send address X to MAR
2. FETCH Initiate Fetch, data to MDR
3. MDR -> ALU Send data in MDR to ALU
4. R -> ALU Send data in R to ALU
5. ADD Select ADD circuit as result
6. ALU -> R Copy selected result to R

JUMP X meaning get next instruction from X

1. $IR_{addr}$ -> PC Send address X to PC

# Putting the Pieces Together---the Von Neumann Architecture (continued)

COMPARE X  meaning:

if CON(X) > R then GT = 1 else 0

if CON(X) = R then EQ = 1 else 0

if CON(X) < R then LT = 1 else 0

| | | |
|---|---|---|
| 1. | $IR_{addr}$ -> MAR | Send address X to MAR |
| 2. | FETCH | Initiate Fetch, data to MDR |
| 3. | MDR -> ALU | Send data in MDR to ALU |
| 4. | R -> ALU | Send data in R to ALU |
| 5. | SUBTRACT | Evaluate CON(X) – R<br>Sets EQ, GT, and LT |

# Putting the Pieces Together---the Von Neumann Architecture (continued)

JUMPGT X meaning:

if GT = 1 then jump to X

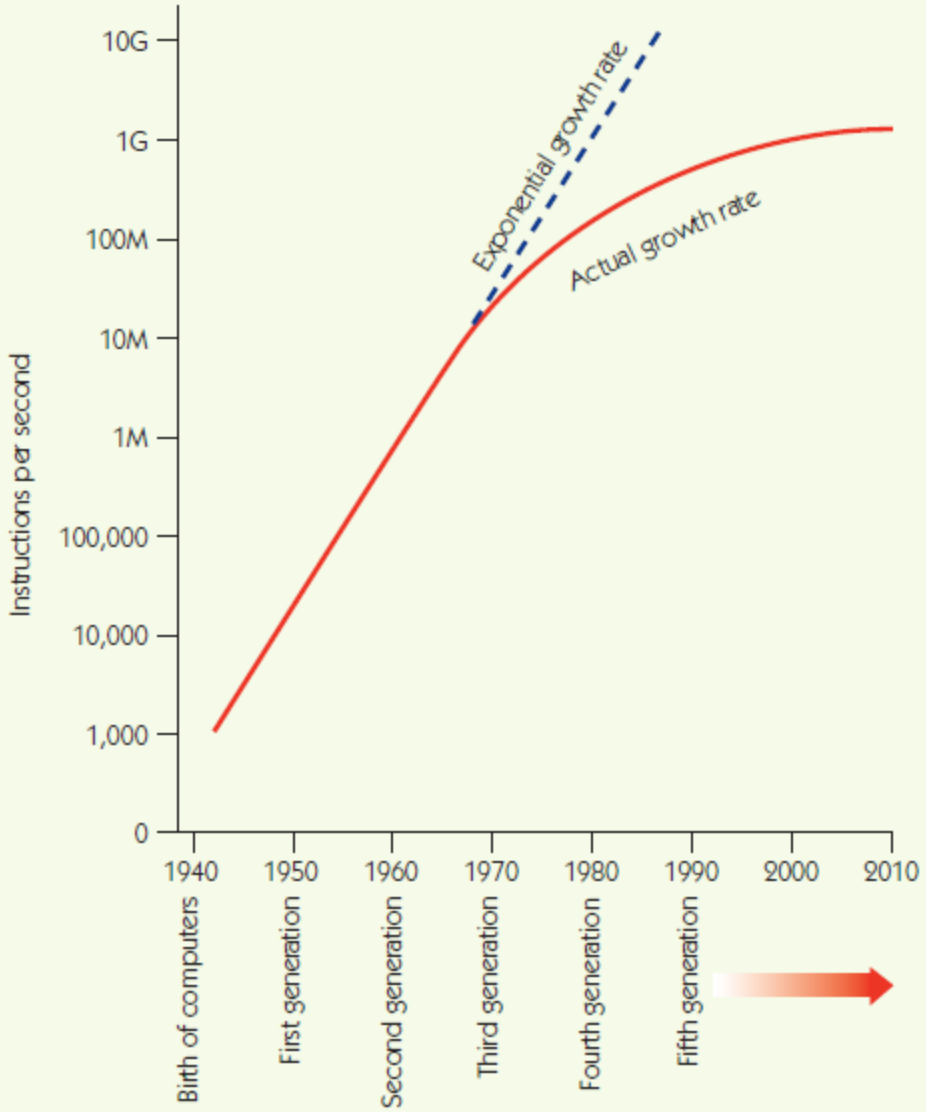
else continue to next instruction

1. IF GT = 1 THEN $IR_{addr}$ -> PC

# Non-Von Neumann Architectures

- Problems to solve are always larger
- Computer chip speeds no longer increase exponentially
- Reducing size puts gates closer together, faster
  - Speed of light pertains to signals through wire
  - Cannot put gates much closer together
  - Heat production increases too fast
- **Von Neumann bottleneck**: inability of sequential machines to handle larger problems

FIGURE 5.20




Graph of processor speeds, 1945 to the present

# Non-Von Neumann Architectures (continued)

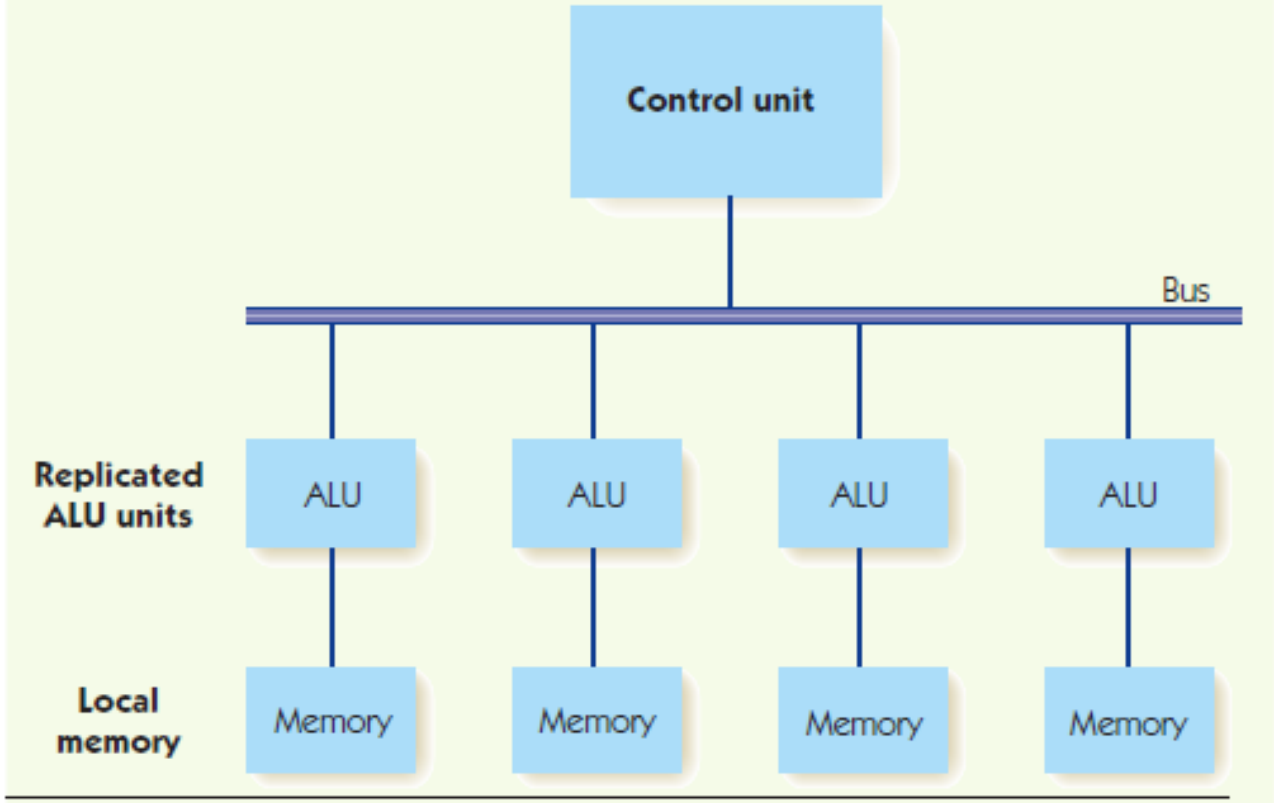
- Non-Von Neumann architectures:
  - Other ways to organize computers
  - Most are experimental/theoretical, EXCEPT parallel processing
- **Parallel processing:**
  - Use many processing units operating at the same time
  - Supercomputers (in the past)
  - Desktop multi-core machines (in the present)
  - “The cloud” (in the future)

# Non-Von Neumann Architectures (continued)

**SIMD parallel processing:**

- Single Instruction stream/Multiple Data streams
- Processor contains one control unit, but many ALUs
- Each ALU operates on its own data
- All ALUs perform exactly the same instruction at the same time
- Older supercomputers, **vector** operations (sequences of numbers)

FIGURE 5.21



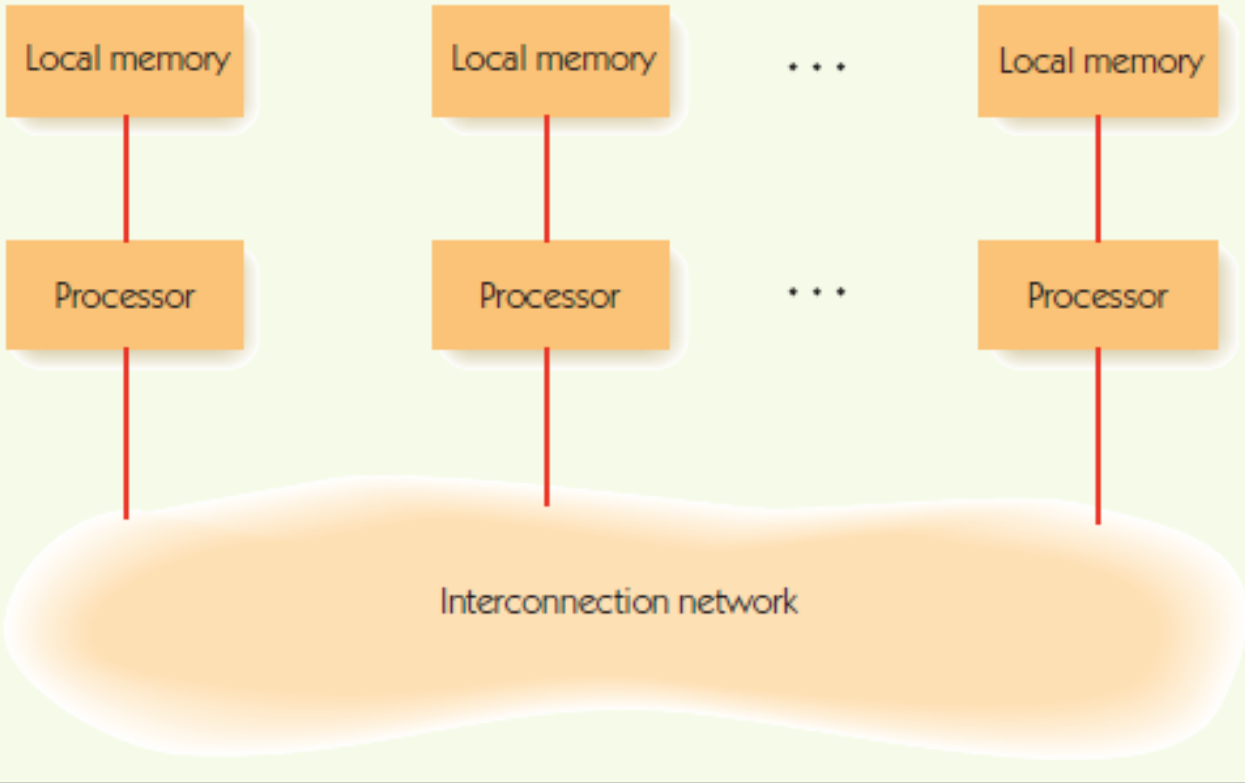

A SIMD parallel processing system

# Non-Von Neumann Architectures (continued)

**MIMD parallel processing:**

- Multiple Instruction streams/Multiple Data streams
- Replicate whole processors, each doing its own thing
- Communication over interconnection network
- Off-the-shelf processors work fine
- Scalable: can always add more processors cheaply
- Communication costs can slow performance

FIGURE 5.22




Model of MIMD parallel processing

# Non-Von Neumann Architectures (continued)

Varieties of MIMD systems:

- Special-purpose systems, newer supercomputers
- **Cluster computing**, standard machines communicating over LAN or WAN
- **Grid computing**, machines of varying power, over large distances/Internet
  - Example: SETI project

- Hot research are: **parallel algorithms**
  - Need to take advantage of all this processing power

# Summary

- We must abstract in order to manage system complexity
- Von Neumann architecture is standard for modern computing
- Von Neumann machines have memory, I/O, ALU, and control unit; programs are stored in memory; execution is sequential unless program says other
- Memory is organized into addressable cells; data is fetched and stored based on MAR and MDR; uses decoder and fetch/store controller

# Summary (continued)

- Mass data storage is nonvolatile; disks store and fetch sectors of data stored in tracks
- I/O is slow, needs dedicated controller to free CPU
- ALU performs computations, moving data to/from dedicated registers
- Control unit fetches, decodes, and executes instructions; instructions are written in machine language
- Parallel processing architectures can perform multiple instructions at one time