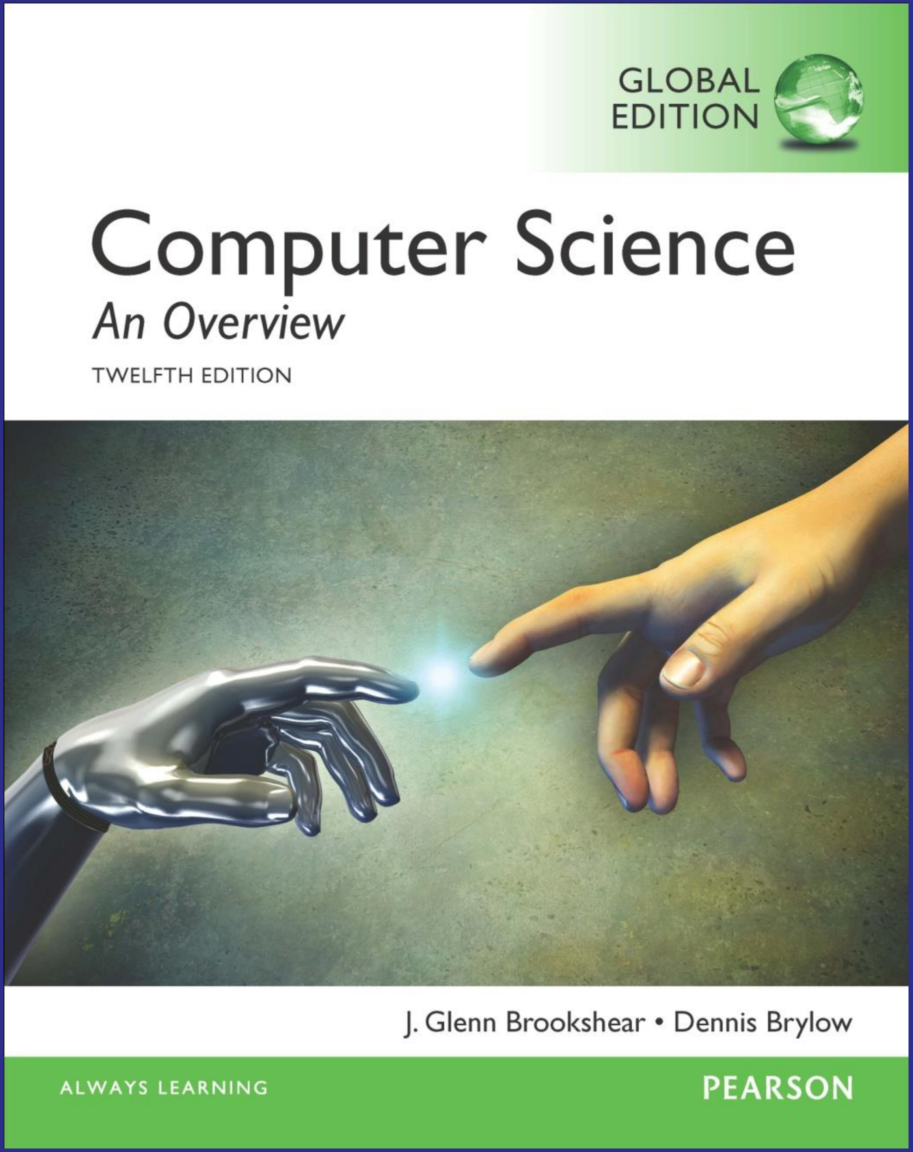
Chapter 2: Data Manipulation



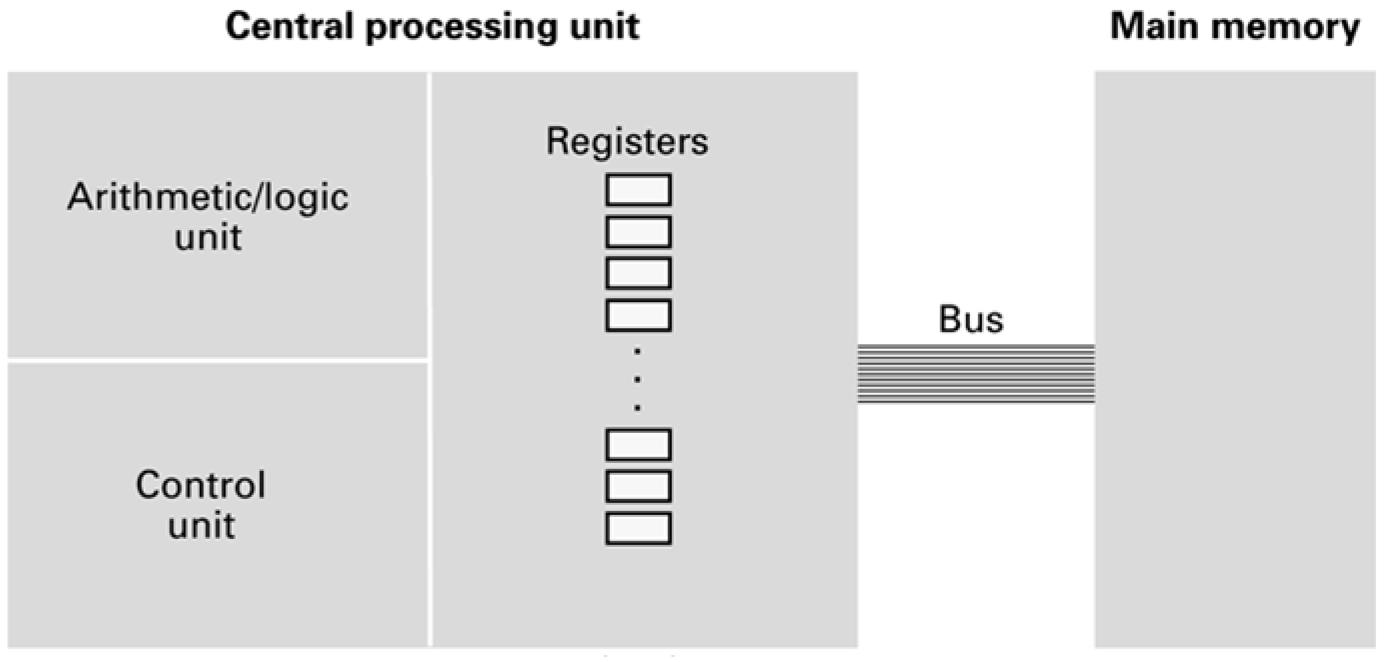
PEARSON

Chapter 2: Data Manipulation

Computer Architecture

- Central Processing Unit (CPU) or processor
  - Arithmetic/Logic unit versus Control unit
  - Registers
    - General purpose
    - Special purpose
- Bus
- Motherboard

Figure 2.1 **CPU and main memory connected via a bus**



Stored Program Concept

A program can be encoded as bit patterns and stored in main memory. From there, the CPU can then extract the instructions and execute them. In turn, the program to be executed can be altered easily.

Terminology

- **Machine instruction:** An instruction (or command) encoded as a bit pattern recognizable by the CPU
- **Machine language:** The set of all instructions recognized by a machine

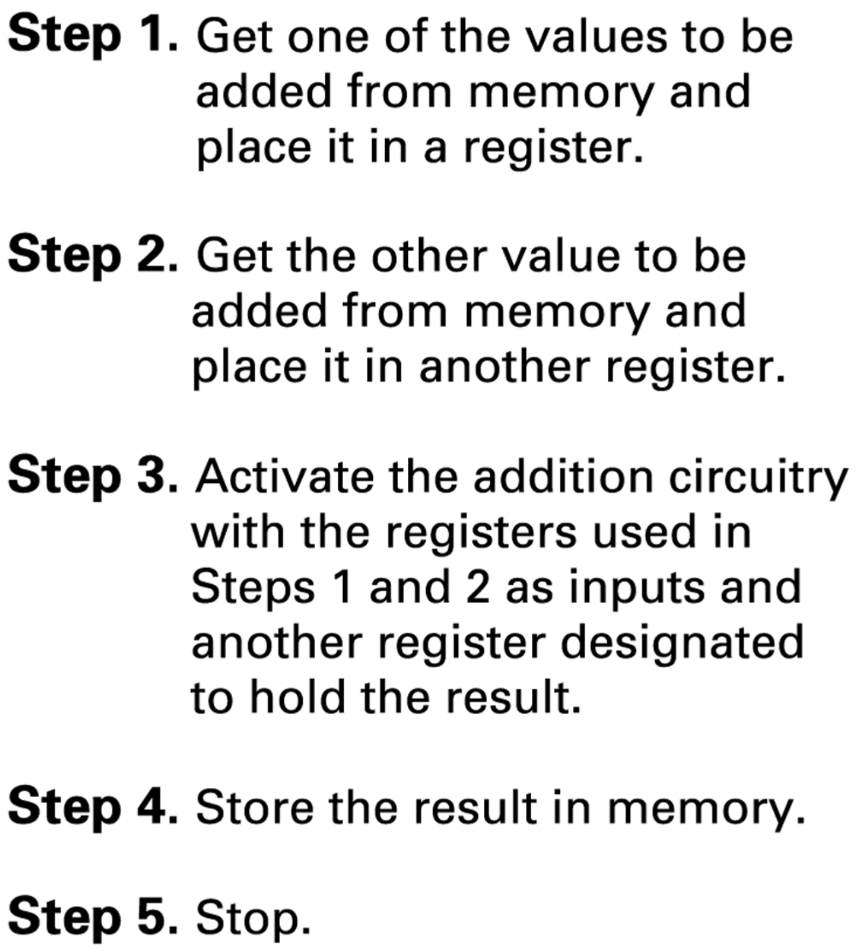
Machine Language Philosophies

- Reduced Instruction Set Computing (RISC)
  - Few, simple, efficient, and fast instructions
  - Examples: PowerPC from Apple/IBM/Motorola and ARM
- Complex Instruction Set Computing (CISC)
  - Many, convenient, and powerful instructions
  - Example: Intel

Machine Instruction Types

- Data Transfer: copy data from one location to another
- Arithmetic/Logic: use existing bit patterns to compute a new bit patterns
- Control: direct the execution of the program

Figure 2.2 **Adding values stored in memory**



Step 1. Get one of the values to be added from memory and place it in a register.

Step 2. Get the other value to be added from memory and place it in another register.

Step 3. Activate the addition circuitry with the registers used in Steps 1 and 2 as inputs and another register designated to hold the result.

Step 4. Store the result in memory.

Step 5. Stop.

Figure 2.3 **Dividing values stored in memory**

Step 1. LOAD a register with a value from memory.

Step 2. LOAD another register with another value from memory.

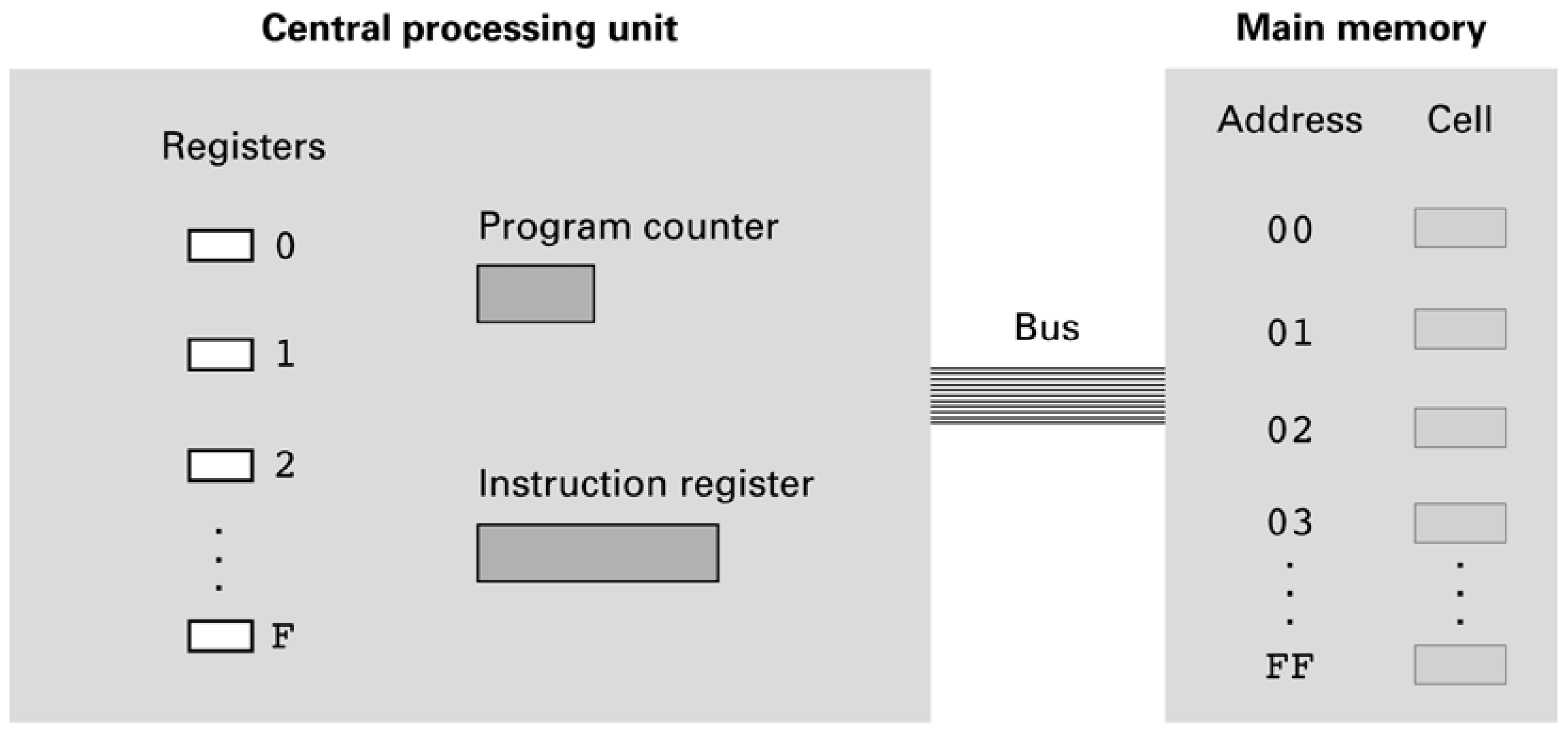
Step 3. If this second value is zero, JUMP to Step 6.

Step 4. Divide the contents of the first register by the second register and leave the result in a third register.

Step 5. STORE the contents of the third register in memory.

Step 6. STOP.

Figure 2.4 **The architecture of the machine described in Appendix C**



Parts of a Machine Instruction

- **Op-code:** Specifies which operation to execute
- **Operand:** Gives more detailed information about the operation
  - Interpretation of operand varies depending on op-code

Figure 2.5 **The composition of an instruction for the machine in Appendix C**

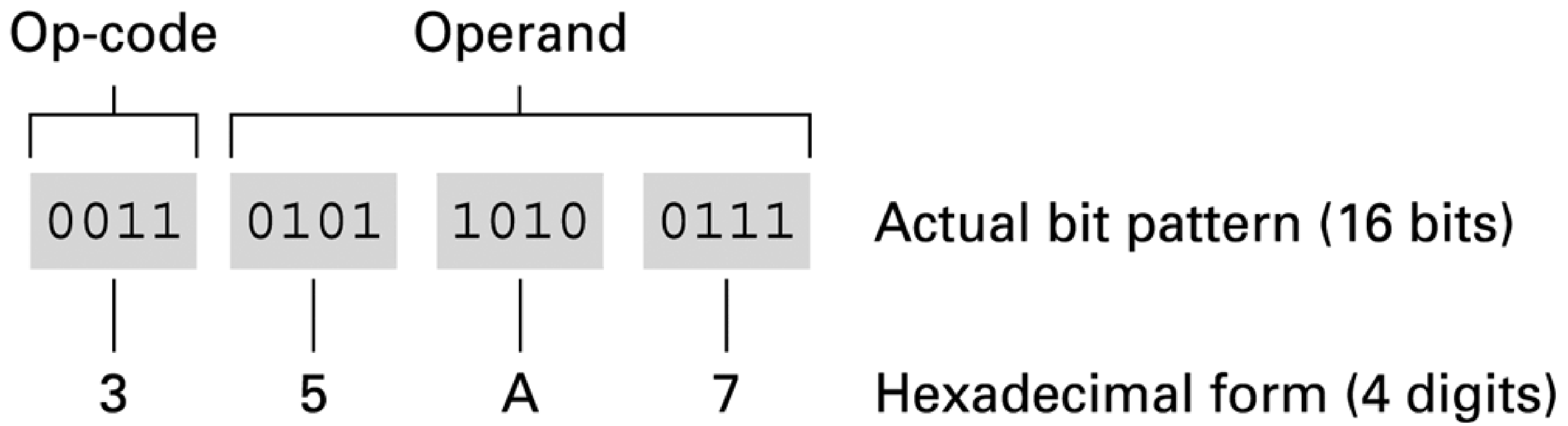


Figure 2.6 **Decoding the instruction 35A7**

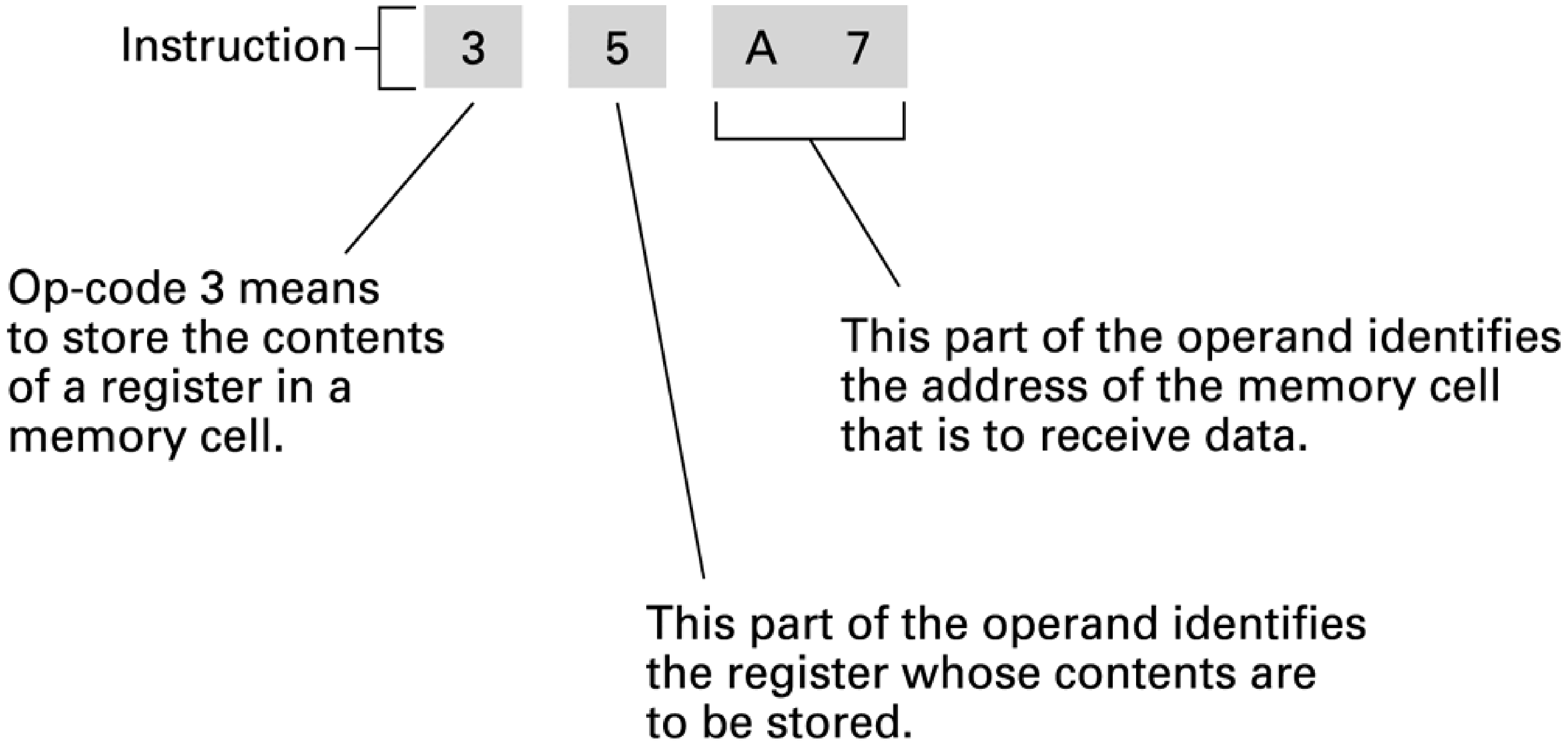
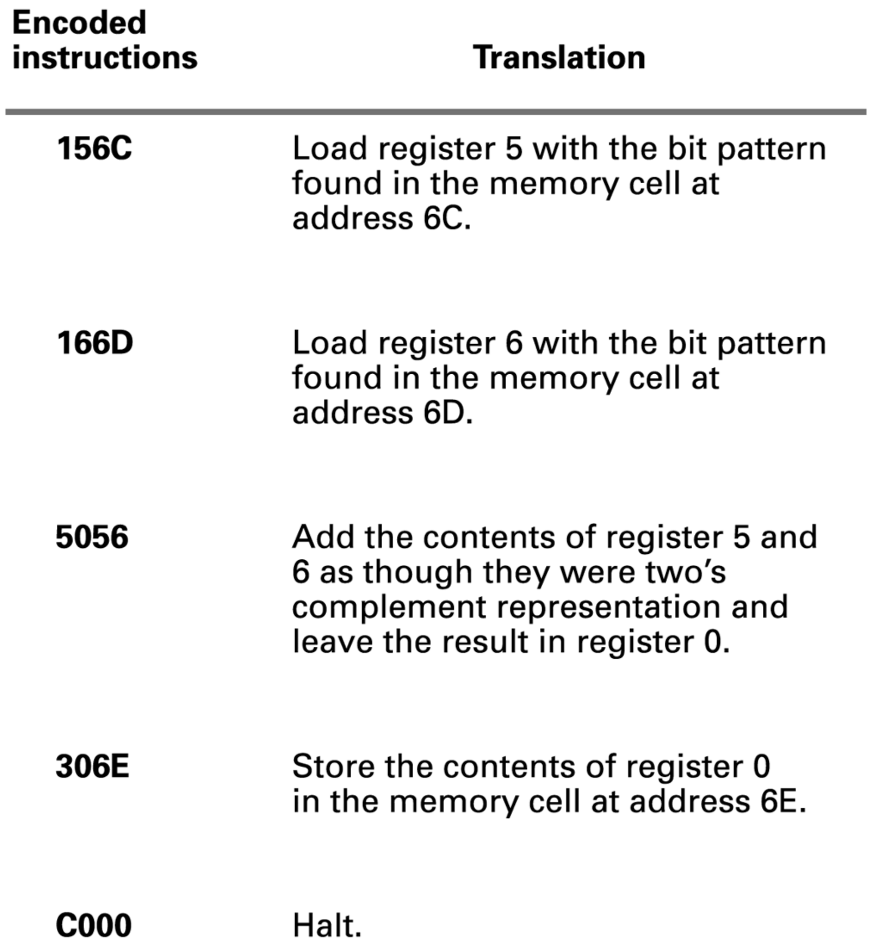

Instruction
3
5
A 7
Op-code 3 means to store the contents of a register in a memory cell.
This part of the operand identifies the address of the memory cell that is to receive data.
This part of the operand identifies the register whose contents are to be stored.

Figure 2.7 **An encoded version of the instructions in Figure 2.2**



Encoded instructions	Translation
156C	Load register 5 with the bit pattern found in the memory cell at address 6C.
166D	Load register 6 with the bit pattern found in the memory cell at address 6D.
5056	Add the contents of register 5 and 6 as though they were two’s complement representation and leave the result in register 0.
306E	Store the contents of register 0 in the memory cell at address 6E.
C000	Halt.

Program Execution

- Controlled by two special-purpose registers
  - Program counter: address of next instruction
  - Instruction register: current instruction
- Machine Cycle
  - Fetch
  - Decode
  - Execute

Figure 2.8 **The machine cycle**

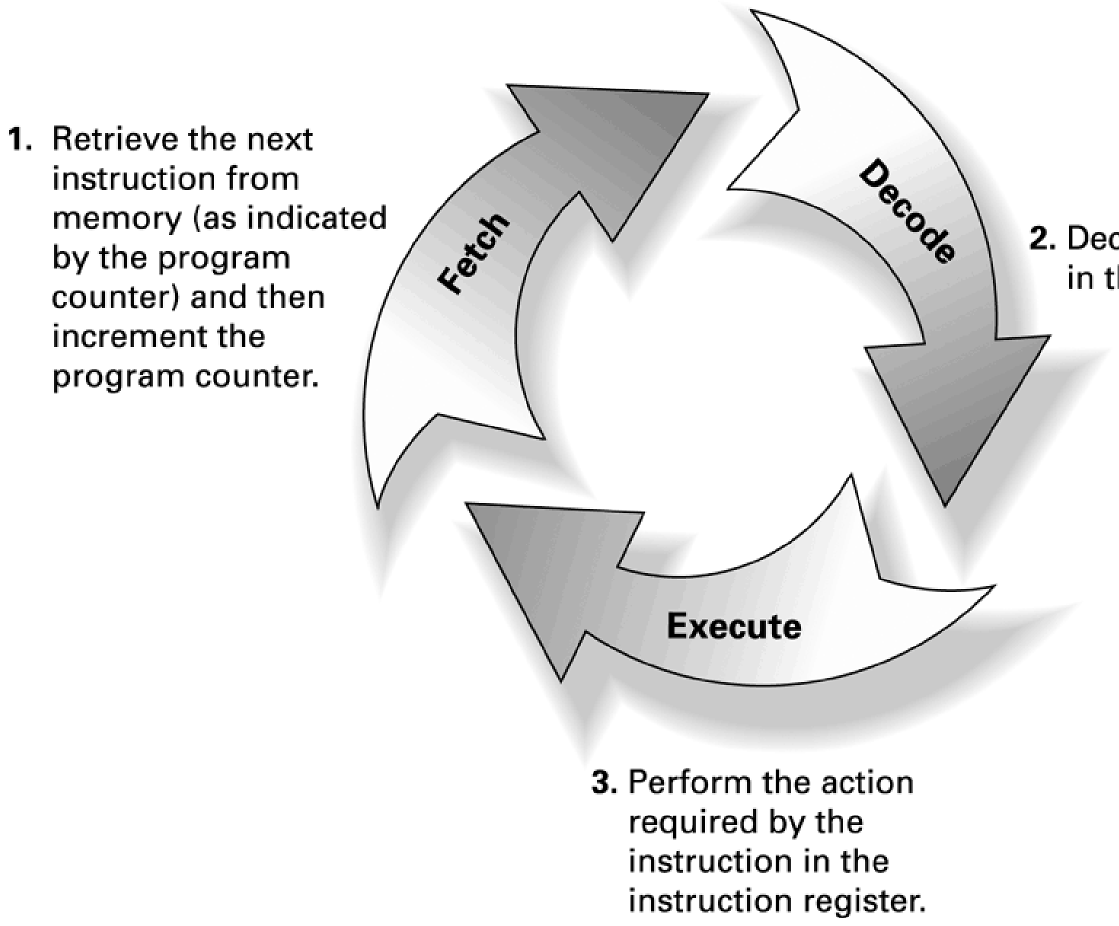

1. Retrieve the next instruction from memory (as indicated by the program counter) and then increment the program counter.
Fetch
Decode
2. Decode the bit pattern in the instruction register.
Execute
3. Perform the action required by the instruction in the instruction register.

Figure 2.9 **Decoding the instruction B258**

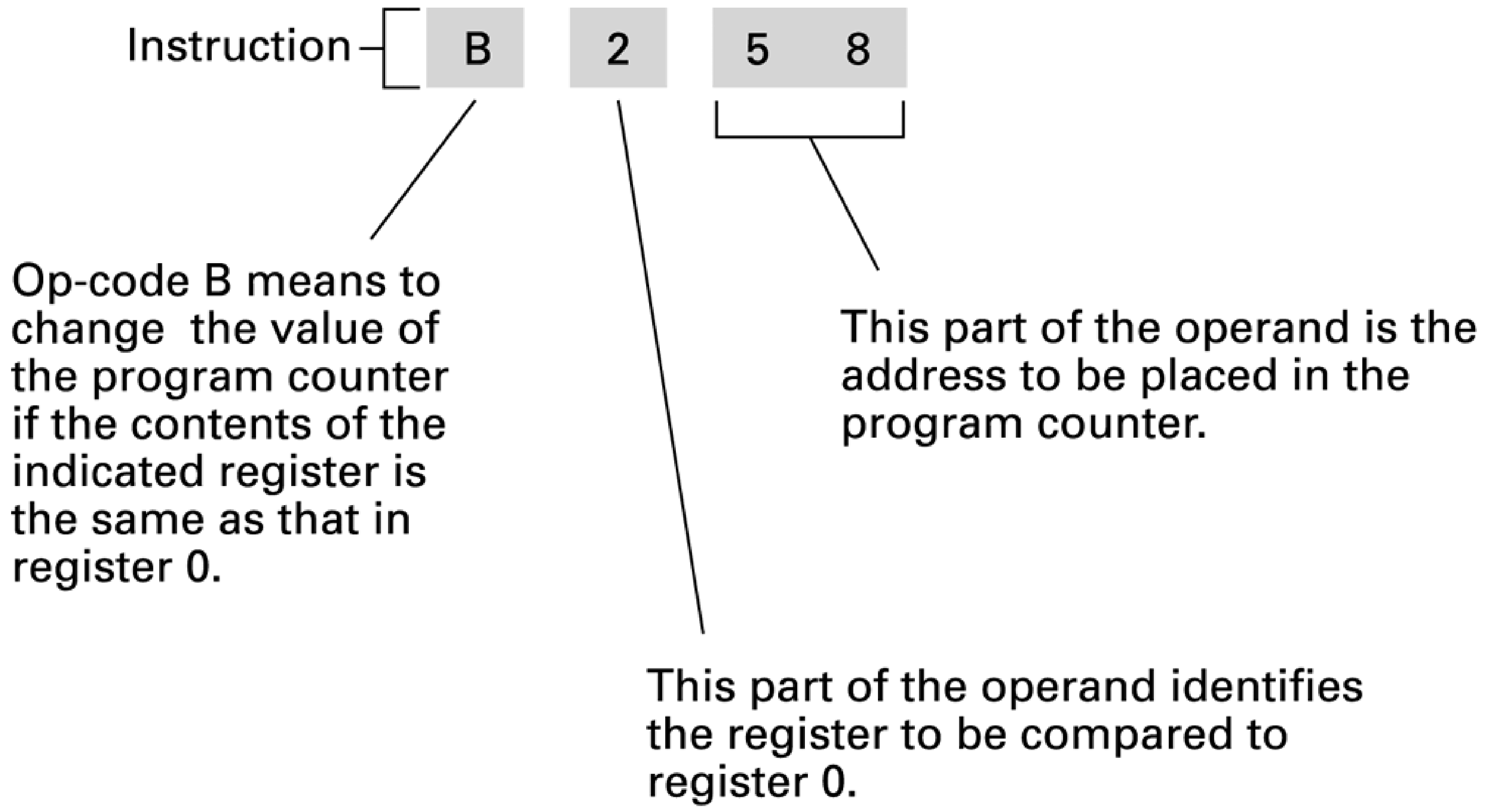

Instruction
B
2
5 8
Op-code B means to change the value of the program counter if the contents of the indicated register is the same as that in register 0.
This part of the operand is the address to be placed in the program counter.
This part of the operand identifies the register to be compared to register 0.

Figure 2.10 **The program from Figure 2.7 stored in main memory ready for execution**

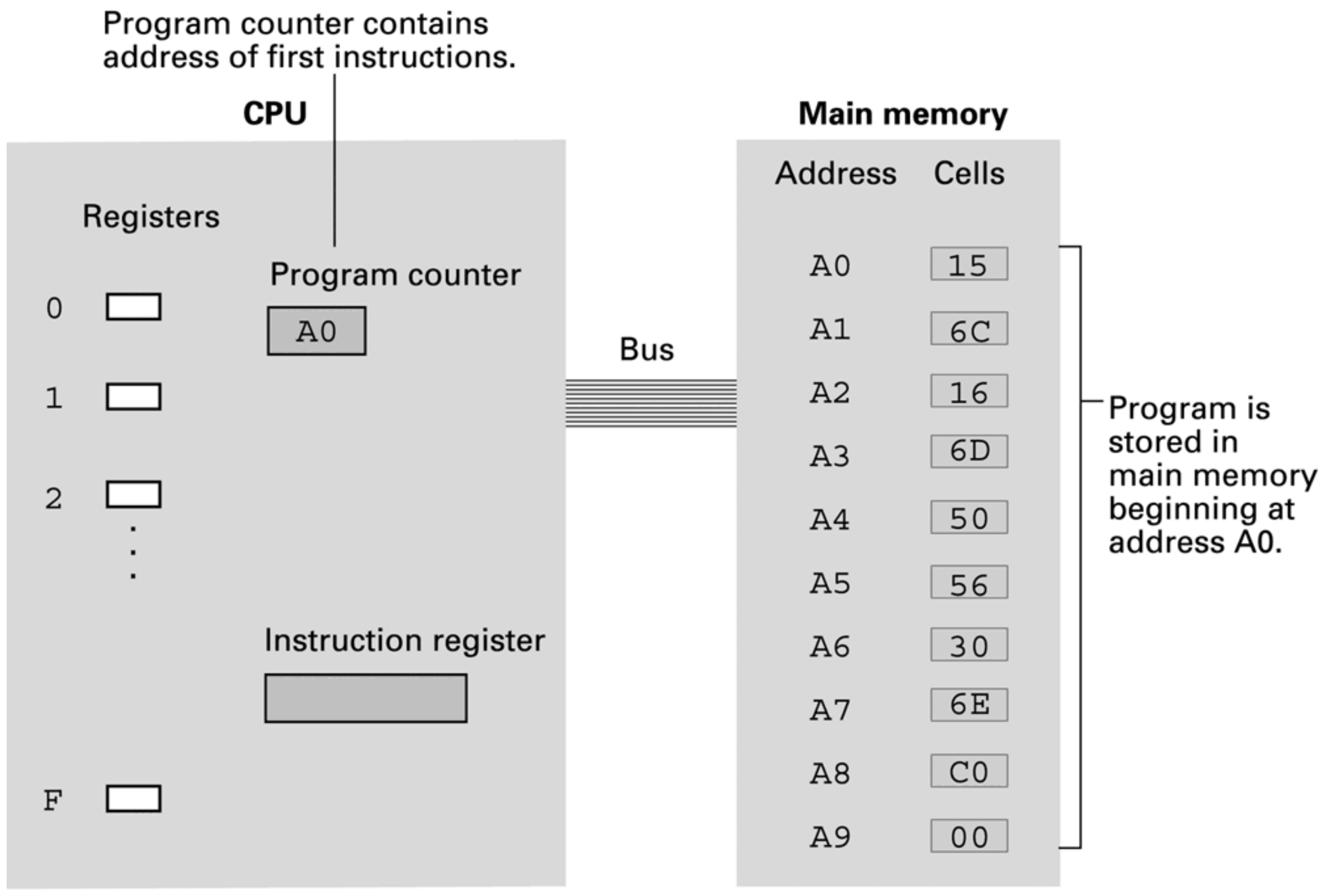
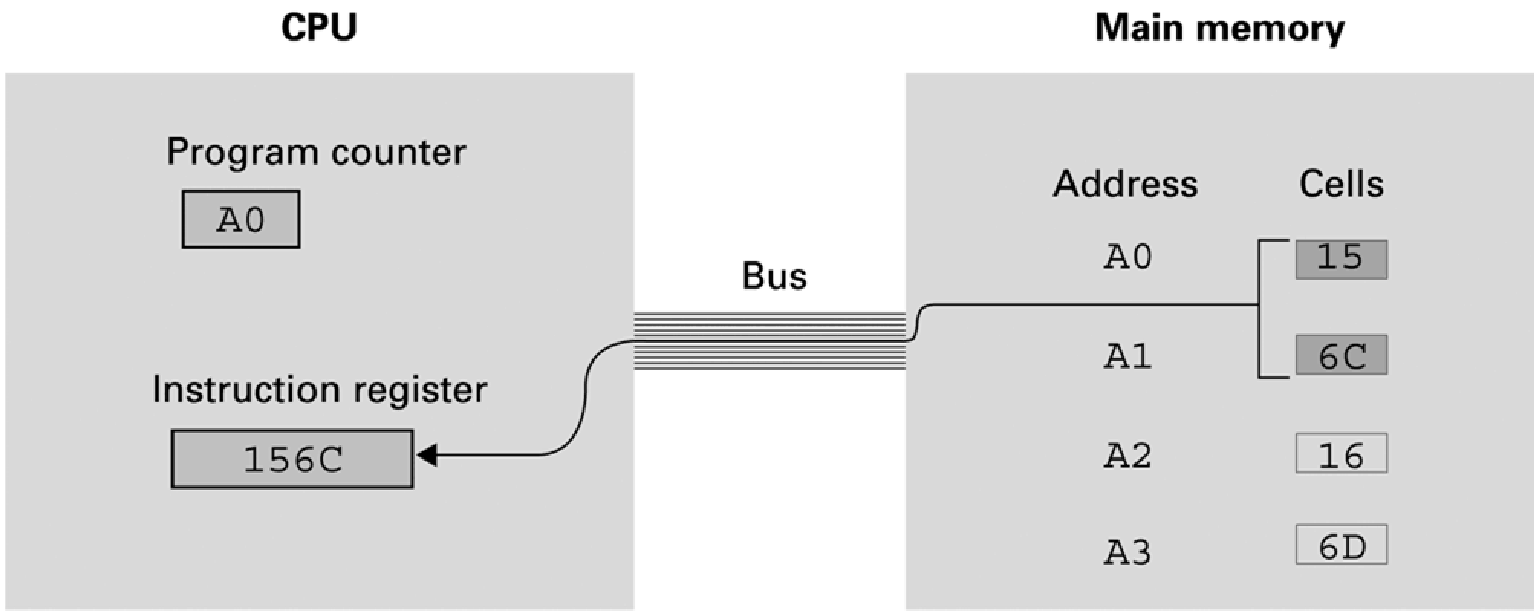
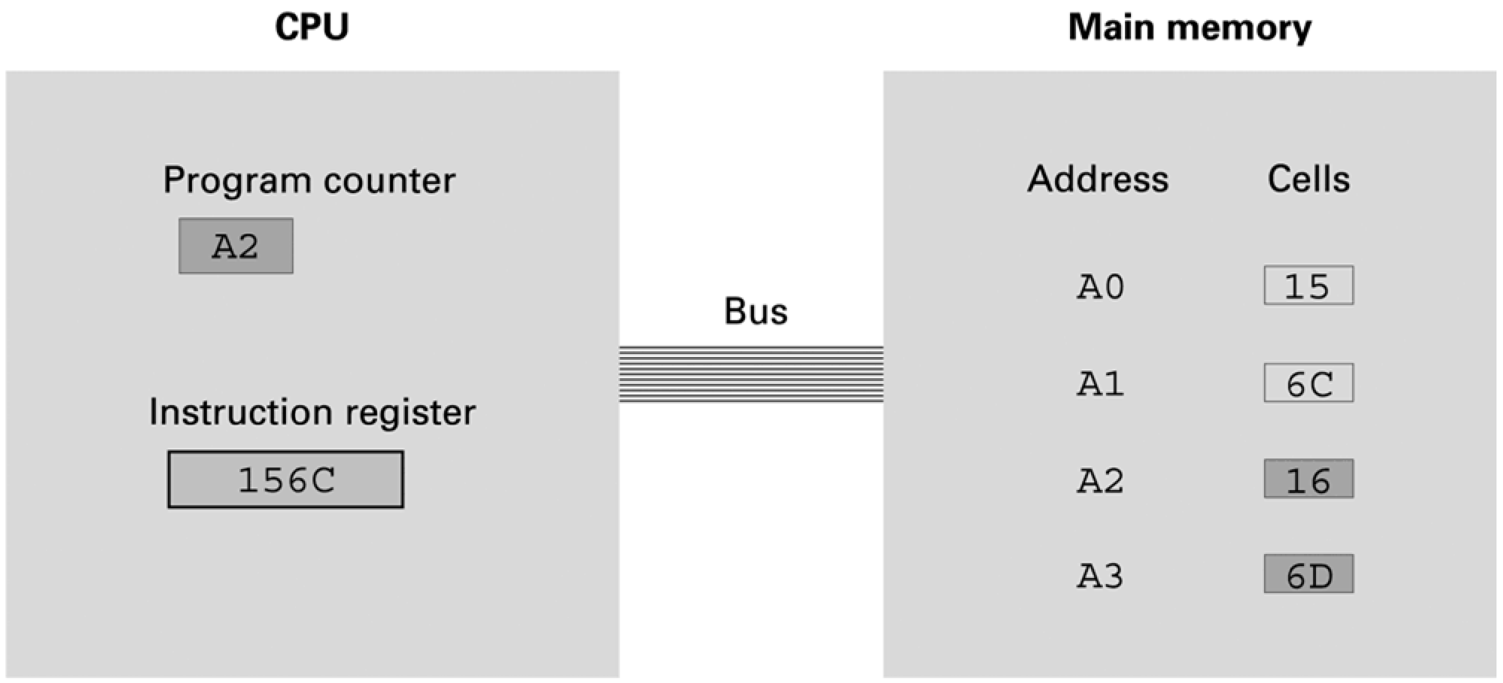


Figure 2.11 **Performing the fetch step of the machine cycle**



a. At the beginning of the fetch step the instruction starting at address A0 is retrieved from memory and placed in the instruction register.

Figure 2.11 **Performing the fetch step of the machine cycle (continued)**

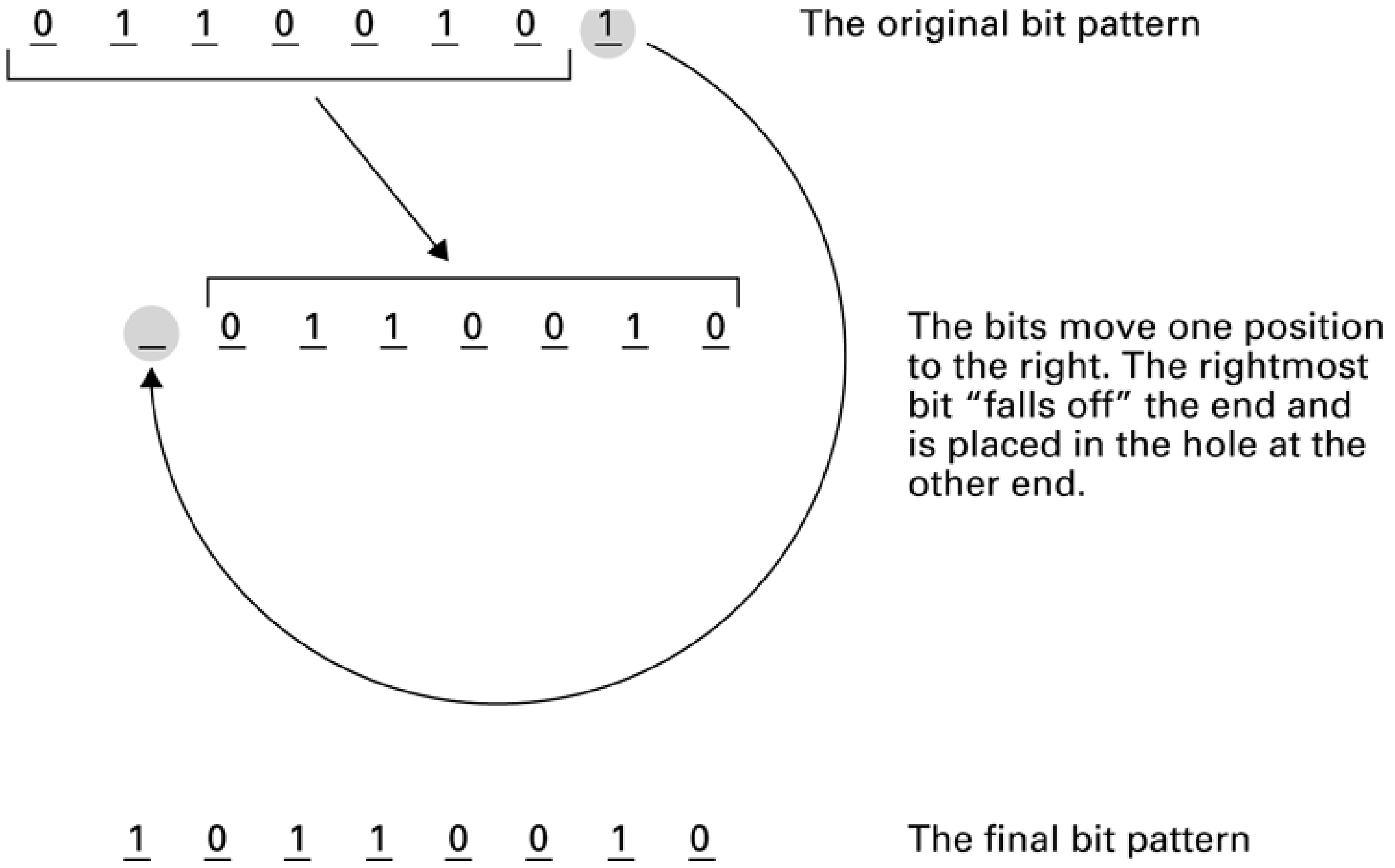


b. Then the program counter is incremented so that it points to the next instruction.

Arithmetic/Logic Operations

- Logic: AND, OR, XOR
  - Masking
- Rotate and Shift: circular shift, logical shift, arithmetic shift
- Arithmetic: add, subtract, multiply, divide
  - Precise action depends on how the values are encoded (two's complement versus floating-point).

Figure 2.12 **Rotating the bit pattern 65 (hexadecimal) one bit to the right**


0 1 1 0 0 1 0 1
The original bit pattern
0 1 1 0 0 1 0
The bits move one position to the right. The rightmost bit "falls off" the end and is placed in the hole at the other end.
1 0 1 1 0 0 1 0
The final bit pattern

Communicating with Other Devices

- **Controller:** An intermediary apparatus that handles communication between the computer and a device
  - Specialized controllers for each type of device
  - General purpose controllers (USB and FireWire)
- **Port:** The point at which a device connects to a computer
- **Memory-mapped I/O:** CPU communicates with peripheral devices as though they were memory cells

Figure 2.13 **Controllers attached to a machine's bus**

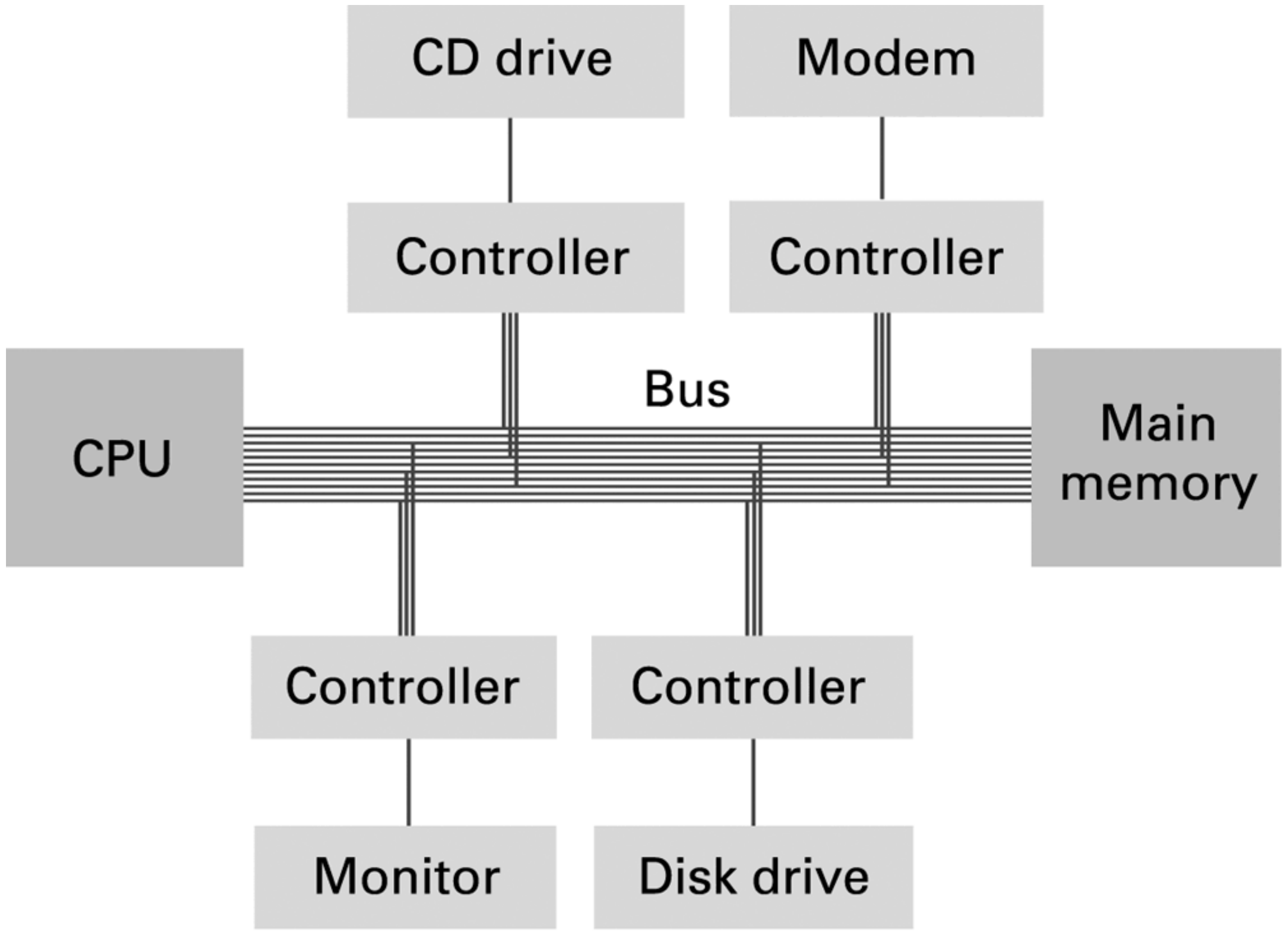
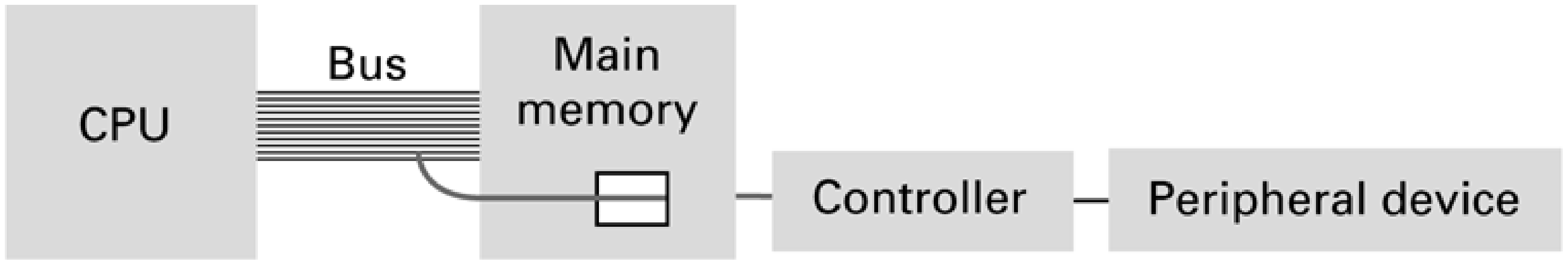


Figure 2.14 **A conceptual representation of memory-mapped I/O**



Communicating with Other Devices (continued)

- **Direct memory access (DMA):** Main memory access by a controller over the bus
- **Von Neumann Bottleneck:** Insufficient bus speed impedes performance
- **Handshaking:** The process of coordinating the transfer of data between components

Communicating with Other Devices (continued)

- **Parallel Communication:** Several communication paths transfer bits simultaneously.
- **Serial Communication:** Bits are transferred one after the other over a single communication path.

Data Communication Rates

- Measurement units
  - Bps: Bits per second
  - Kbps: Kilo-bps (1,000 bps)
  - Mbps: Mega-bps (1,000,000 bps)
  - Gbps: Giga-bps (1,000,000,000 bps)
- Bandwidth: Maximum available rate

Programming Data Manipulation

- Programing languages shields users from details of the machine:
  - A single Python statement might map to one, tens, or hundreds of machine instructions
  - Programmer does not need to know if the processor is RISC or CISC
  - Assigning variables surely involves LOAD, STORE, and MOVE op-codes

Bitwise Problems as Python Code

```
print(bin(0b10011010 & 0b11001001))
# Prints '0b10001000'

print(bin(0b10011010 | 0b11001001))
# Prints '0b11011011'

print(bin(0b10011010 ^ 0b11001001))
# Prints '0b1010011'
```

Control Structures

- If statement:

```
if (water_temp > 140):
    print('Bath water too hot!')
```

- While statement:

```
while (n < 10):
    print(n)
    n = n + 1
```

Functions

- **Function**: A name for a series of operations that should be performed on the given parameter or parameters
- **Function call**: Appearance of a function in an expression or statement

```
x = 1034
y = 1056
z = 2078
biggest = max(x, y, z)
print(biggest)   # Prints '2078'
```

Functions (continued)

- **Argument Value**: A value plugged into a parameter
- **Fruitful** functions **return** a value
- **void functions**, or **procedures,** do not return a value

```
import math
sideA = 3.0
sideB = 4.0
# Calculate third side via Pythagorean Theorem
hypotenuse = math.sqrt(sideA**2 + sideB**2)
print(hypotenuse)
```

Input / Output

```
# Calculates the hypotenuse of a right triangle

import math

# Inputting the side lengths, first try
sideA = int(input('Length of side A? '))
sideB = int(input('Length of side B? '))

# Calculate third side via Pythagorean Theorem
hypotenuse = math.sqrt(sideA**2 + sideB**2)

print(hypotenuse)
```

Marathon Training Assistant

```
# Marathon training assistant.

import math

# This function converts a number of minutes and
# seconds into just seconds.
def total_seconds(min, sec):
    return min * 60 + sec

# This function calculates a speed in miles per hour given
# a time (in seconds) to run a single mile.
def speed(time):
    return 3600 / time
```

Marathon Training Assistant (continued)

```
# Prompt user for pace and mileage.
pace_minutes = int(input('Minutes per mile? '))
pace_seconds = int(input('Seconds per mile? '))
miles = int(input('Total miles? '))

# Calculate and print speed.
mph = speed(total_seconds(pace_minutes, pace_seconds))
print('Your speed is ' + str(mph) + ' mph')

# Calculate elapsed time for planned workout.
total = miles * total_seconds(pace_minutes, pace_seconds)
elapsed_minutes = total // 60
elapsed_seconds = total % 60

print('Your elapsed time is ' + str(elapsed_minutes) +
              ' mins ' + str(elapsed_seconds) + ' secs')
```

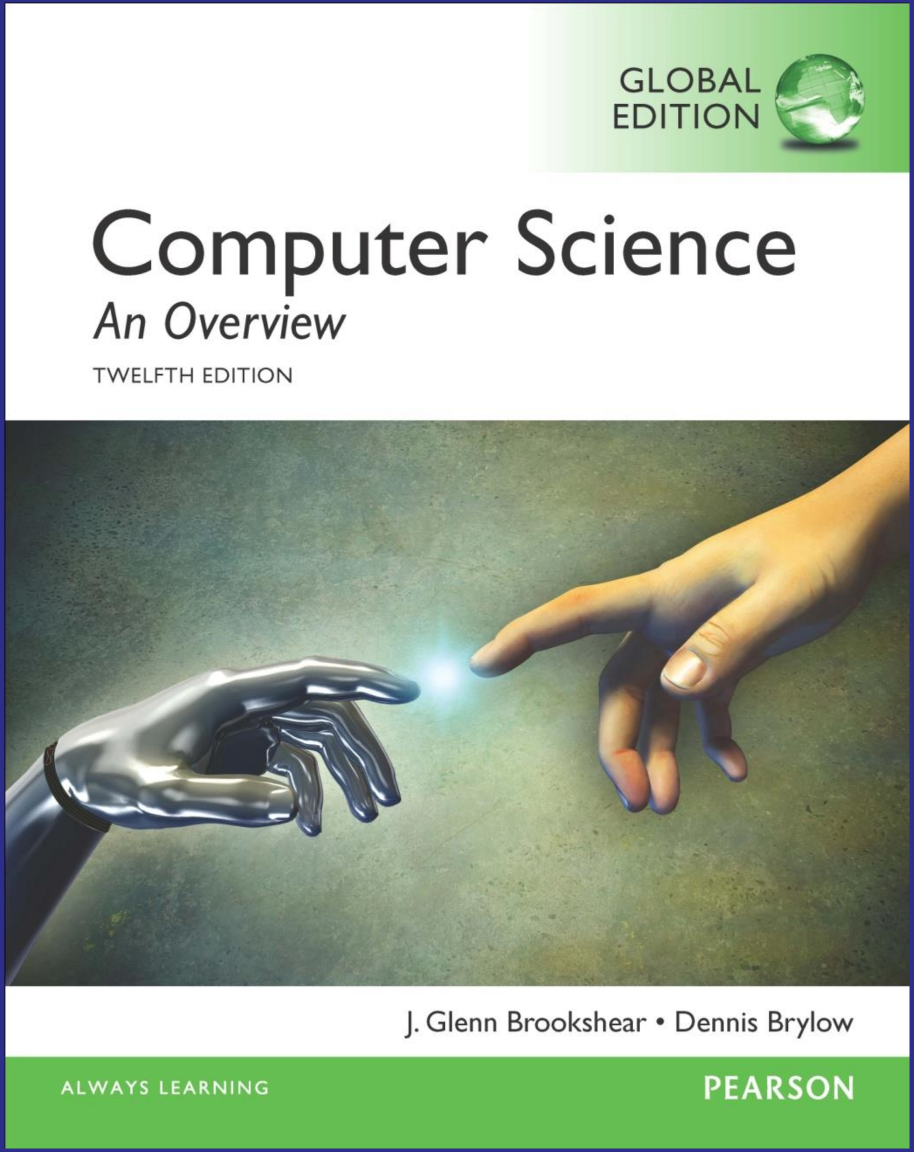
Figure 2.15 **Example Marathon Training Data**

Time Per Mile				Total Elapsed Time	
Minutes	Seconds	Miles	Speed (mph)	Minutes	Seconds
9	14	5	6.49819494584	46	10
8	0	3	7.5	24	0
7	45	6	7.74193548387	46	30
7	25	1	8.08988764044	7	25

Other Architectures

- Technologies to increase throughput:
  - Pipelining: Overlap steps of the machine cycle
  - Parallel Processing: Use multiple processors or cores simultaneously
    - SISD: No parallel processing
    - MIMD: Different programs, different data
    - SIMD: Same program, different data

End of Chapter



PEARSON