CPE 221

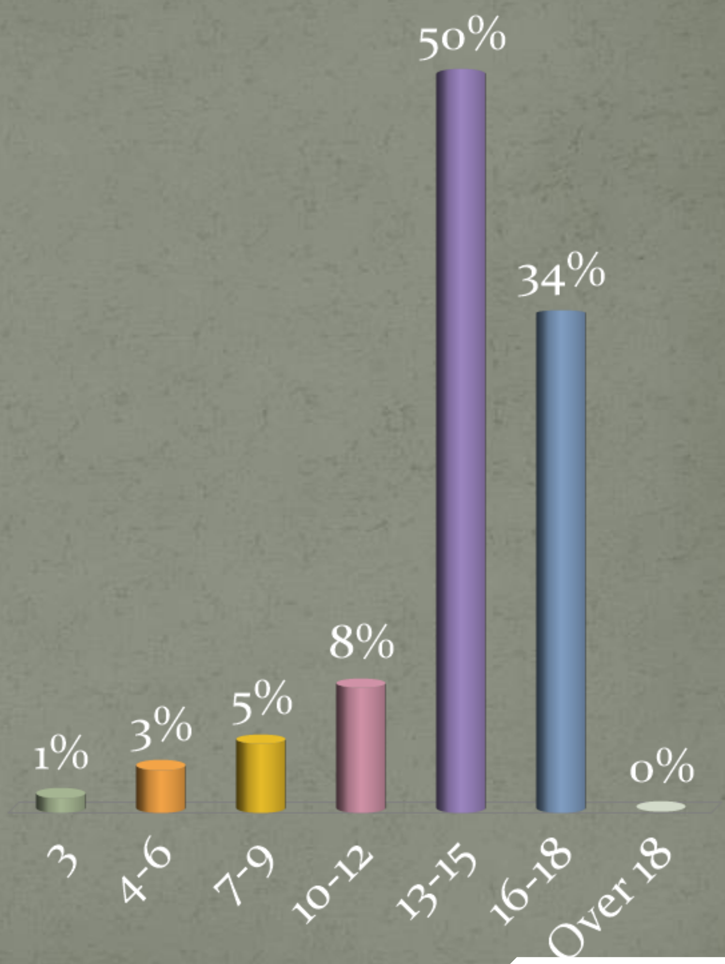
Chapter 1 – Computer Systems Architecture

Dr. Rhonda Kay Gaede

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

I am currently taking ___ hours of classes at UAH.

A. 3
B. 4-6
C. 7-9
D. 10-12
E. 13-15
F. 16-18
G. Over 18



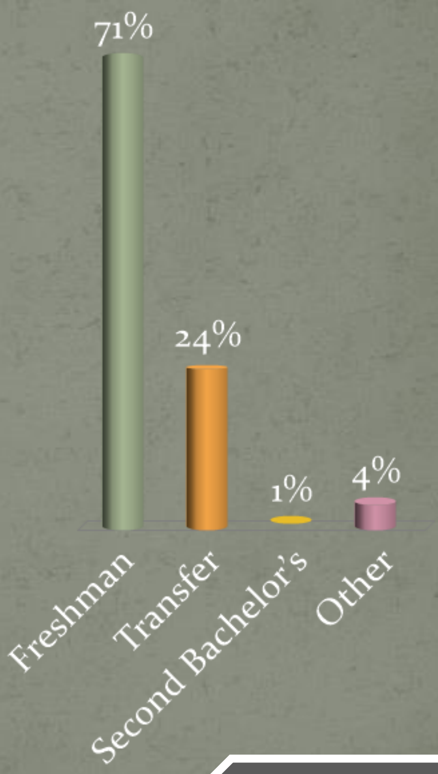
I started at UAH as a ___________

A. Freshman

B. Transfer

C. Second Bachelor’s

D. Other



Course Context

- **Traditional Computing Platforms**

Which has more software?

A. Space Shuttle

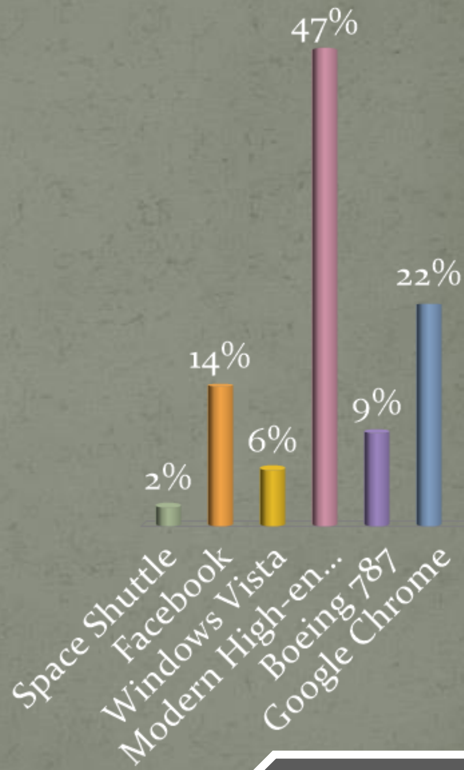
B. Facebook

C. Windows Vista

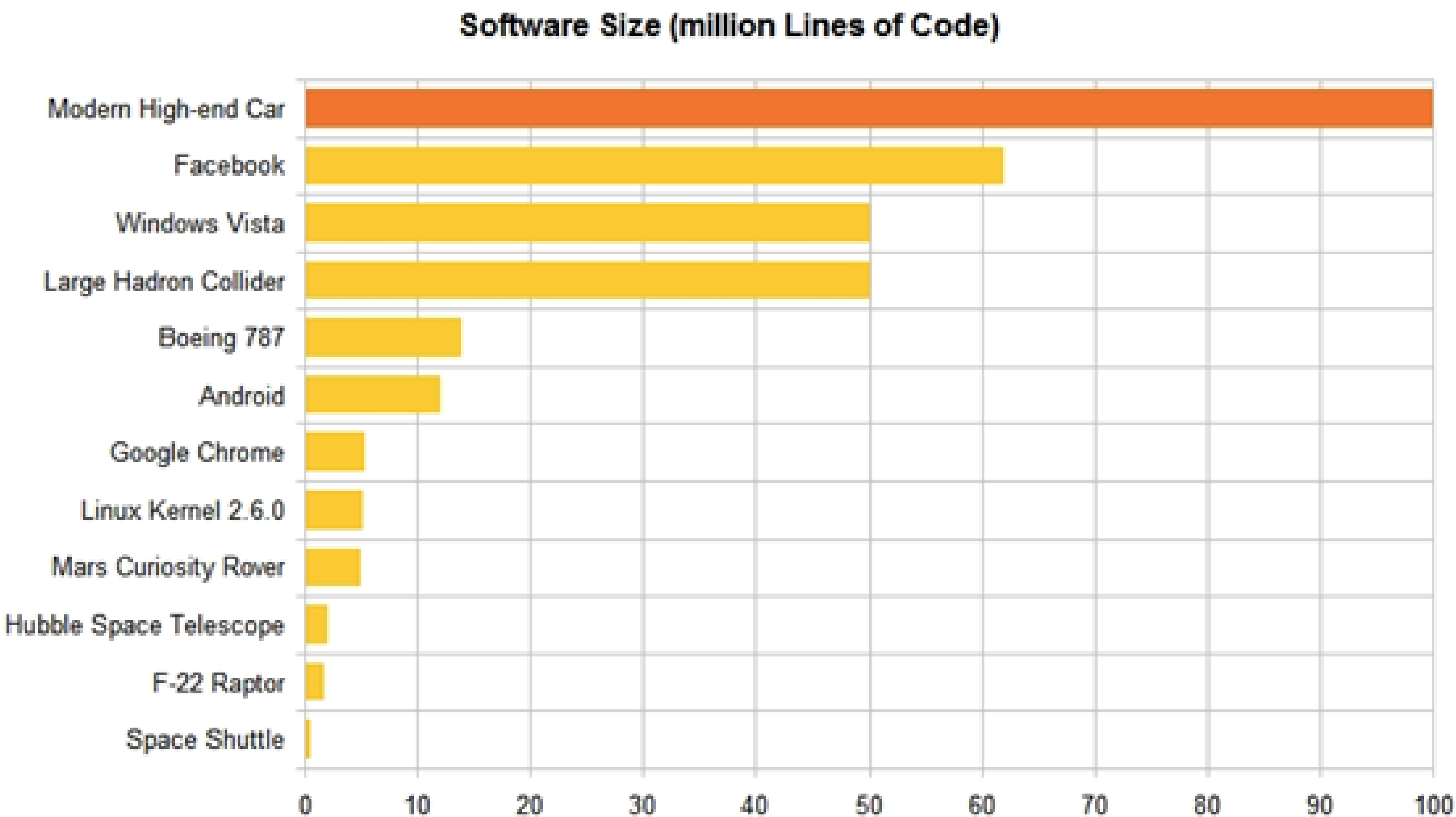
D. Modern High-end Car

E. Boeing 787

F. Google Chrome



Course Context



Course Context

- **Embedded Systems**









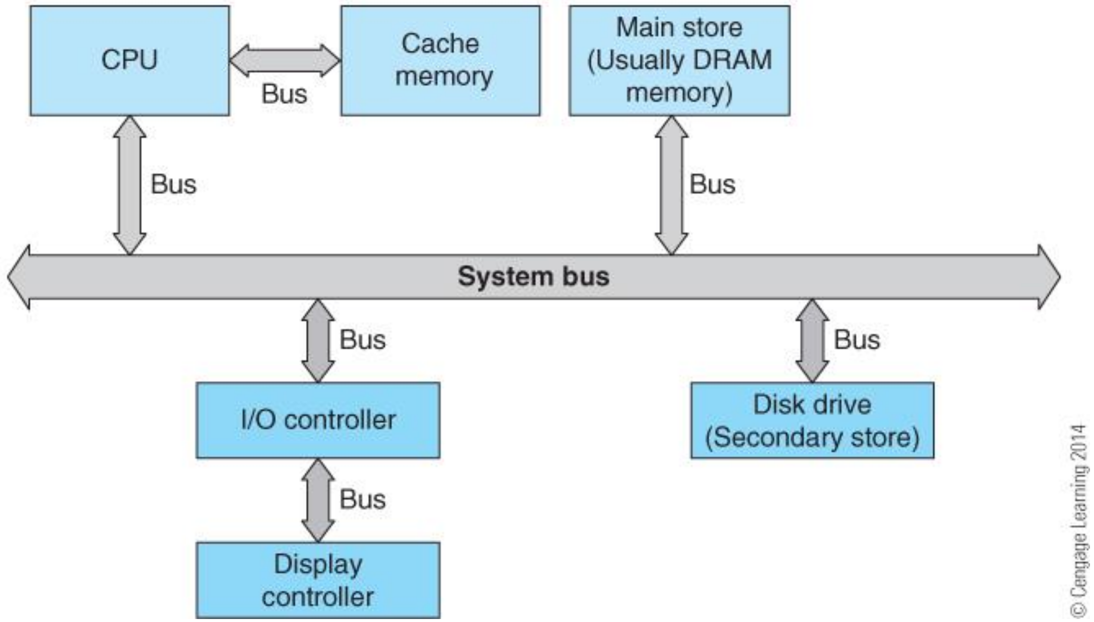


Course Goals

- Multiple levels of computer operation
  - Application level
  - High Level Language(s), HLL, level(s) ENG 101 CPE 211
  - ✓ Assembly/machine language level: instruction set
  - ✓ System architecture level: subsystems & connections
  - ✓ Digital logic level: gates, memory elements, buses
  - Electronic design level EE 315
  - Semiconductor physics level EE 310
- Interactions and relations between levels
  - View of machine at each level
  - Tasks and tools at each level
- Historical perspective
- Trends and research activities

1.1 What is Computer Systems Architecture?

FIGURE 1.1 Structure of a computer system



One challenge is that subsystems improve at different rates. For decades, __processors__ improved faster than _hard drives_. As _solid-state disks_ become more prevalent, this problem is alleviated for now. At the same time, processors are now _power-limited_ and increases in _clock speeds_ are no longer possible so we now must make _multi-cores_ work.

1.1 What is a Computer?

- Computers are dedicated or general-purpose
- Dedicated (Embedded) – CPE 323, also specialized hardware (CPE 322)
- General Purpose (CPE 221)
  - It can be programmed to solve any problem (within its limitations)
  - A key feature of almost all general-purpose computers is that the program and its data are held in the same memory
  - This kind of computer is called von-Neumann (alternate is called Harvard)

unsigned

n bits $0 - 2^n - 1$

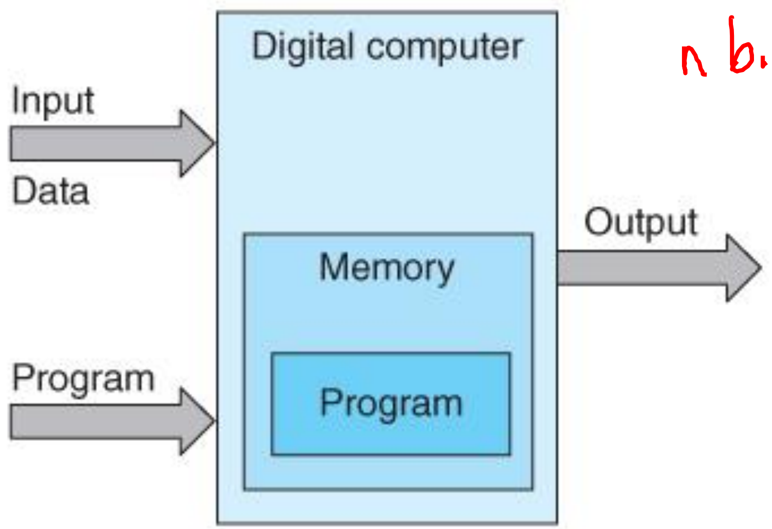
signed

n bits -2^{n-1} $+2^{n-1} - 1$

8 bits $0 - 255$

-128 $+127$

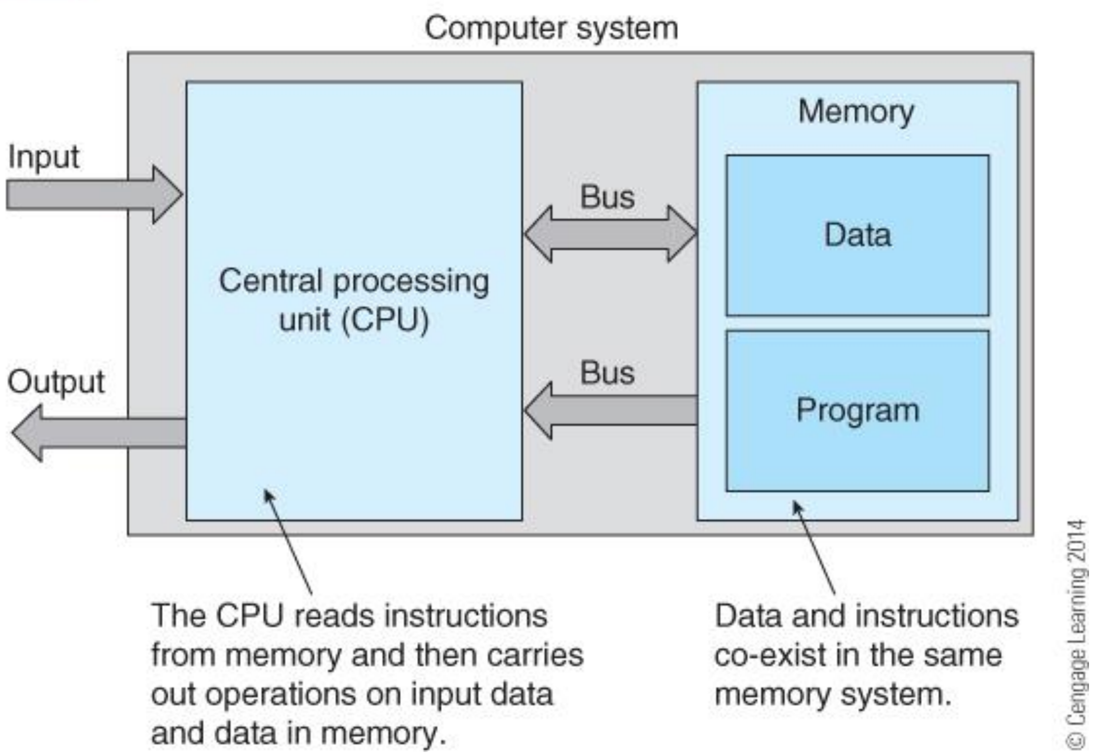
FIGURE 1.2 The general-purpose computer



1.1 What is a Computer? (More Details)

- We take data and instructions out of ___memory___ and put them in ___registers___ for faster access
- A ___register___ is specified in terms of the number of _bits_ it holds, typically _32_ or _64_-bit for modern PCs, smaller for embedded systems.

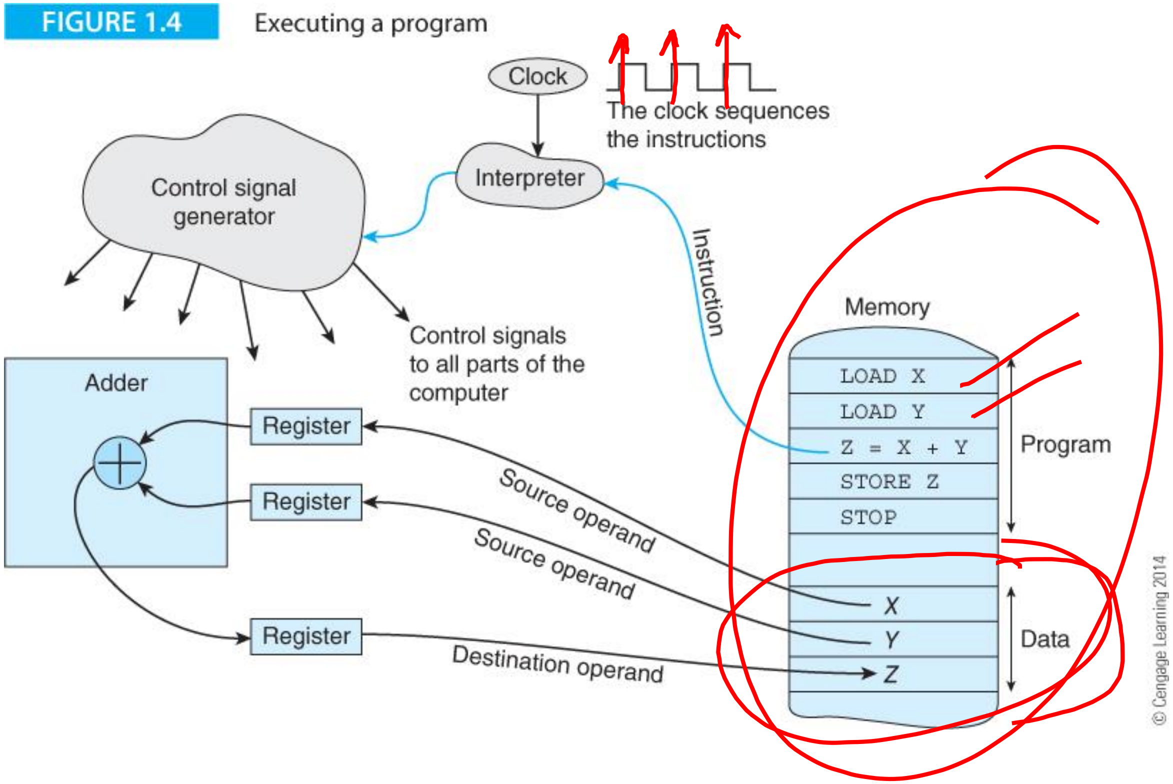
FIGURE 1.3 Structure of the stored program computer



1.1 Program Execution

- Everything happens at the direction of the __clock__.
- The __actions__ of the hardware support the __operations__ of a __high-level language__, such as A = B, or C = A + B.
- The operation shown below is Z = X + Y.

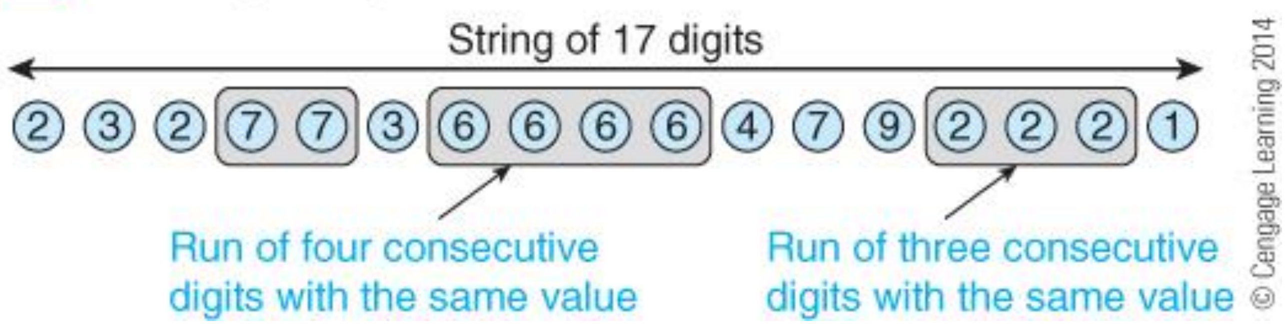
FIGURE 1.4 Executing a program



1.4 The Stored Program Computer

- A computer is a _tool_ to _solve_ __problem__
- Consider the problem of finding the longest sequence of repeated digits in a stream of digits.
- In the figure shown, the longest run of repeated digits is _four_ consecutive _sixes_.

FIGURE 1.7 A string of digits



1.4 The Problem Solution (Pseudocode)

```
i                     The current position in the string
New_Digit             The value of the current digit just read
Current_Run_Value     The value of the elements in the current run
Current_Run_length    The length of the current run
Max_Run               The length of the longest run we've found so far
```

```
Read the first digit in the string and call it New_Digit
Set the Current_Run_Value to New_Digit
Set the Current_Run_Length to 1
Set the Max_Run to 1
REPEAT
  Read New_Digit
  IF New_Digit is the same as Current_Run_Value
  THEN Current_Run_Length = Current_Run_Length + 1
  ELSE {Current_Run_Length = 1
    Current_Run_Value = New_Digit}
  IF Current_Run_Length > Max_Run
  THEN Max_Run = Current_Run_Length
UNTIL no more digits to read
```

1.4 The Problem Solution (Language)

FIGURE 1.11 Memory map of a program and its data

0	`i = 21`
1	`New_Digit = Memory(i)`
2	`Set Current_Run_Value to New_Digit`
3	`Set the Current_Run_Length to 1`
4	`Set the Max_Run to 1`
5	`REPEAT`
6	`i = i + 1`
7	`  New_Digit = Memory(i)`
8	`  IF New_Digit = Current_Run_Value`
9	`    THEN Current_Run_Length = Current_Run_Length + 1`
10	`    JUMP to 13`
11	`    ELSE Current_Run_Length = 1;`
12	`         Current_Run_Value = New_Digit`
13	`  IF Current_Run_Length > Max_Run`
14	`    THEN Max_Run = Current_Run_Length`
15	`UNTIL i = 37`
16	`Stop`
17	**`New_Digit`**
18	**`Current_Run_Value`**
19	**`Current_Run_Length`**
20	**`Max_Run`**
21	**`2 (the first digit in the string)`**
22	**`3`**
23	**`2`**
23	**`7`**
…	…
37	**`1 (the last digit in the string)`**

1.4 Introducing Register Transfer Notation

RTL is a *notation* used to __define__ __operations__. Square brackets indicate the __contents__ of a __memory__ location. The expression M[15] = Max_Run means "the __contents__ of __memory__ location 15 contains the __value__ of Max_Run".

address = 15
data Max_Run

The backward arrow symbol, $\leftarrow$, indicates a __data__ __transfer__.

For example, $M[15] \leftarrow M[15] + 1$ is interpreted as "the content of memory location 15 is increased by 1 and the result put in memory location 15". Consider:

a. M[20] = 5

b. $M[20] \leftarrow 6$

c. $M[20] \leftarrow M[6]$

(a) M[20] has the value of 5

(b) putting 6 in M[20]

(c) copy the value found in M[6] into M[20]

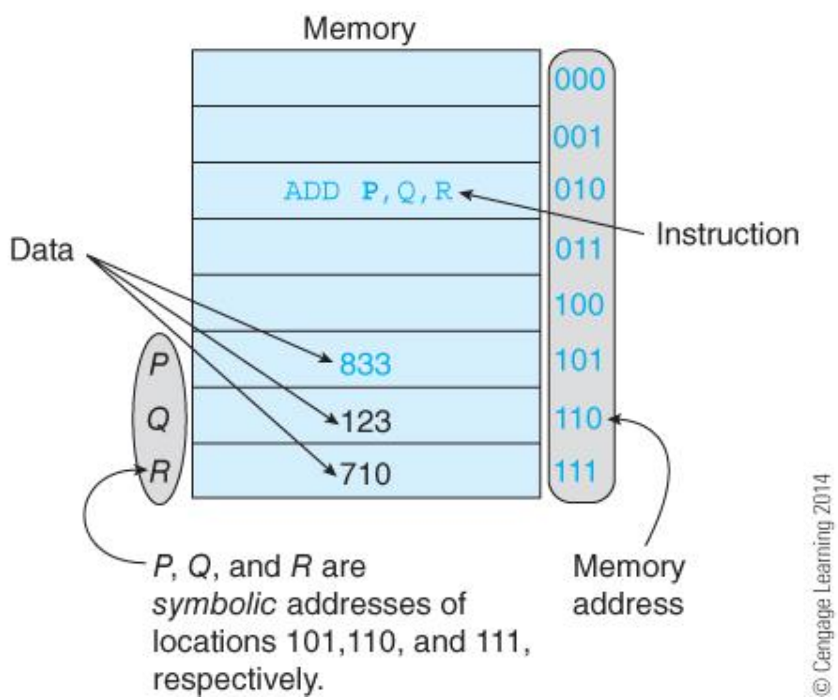
1.5 The Stored Program Concept

```
Stored_program_machine
    Point to the first instruction in memory
    REPEAT
        Read instruction at the memory location pointed at
        Point to the next instruction
        Decode the instruction read from memory
        Execute the instruction
    FOREVER
End
```

program counter

1.5 An Instruction and its Operands

FIGURE 1.13 Relationship between instruction and operands



operator

$P = Q + R$

operands

destination

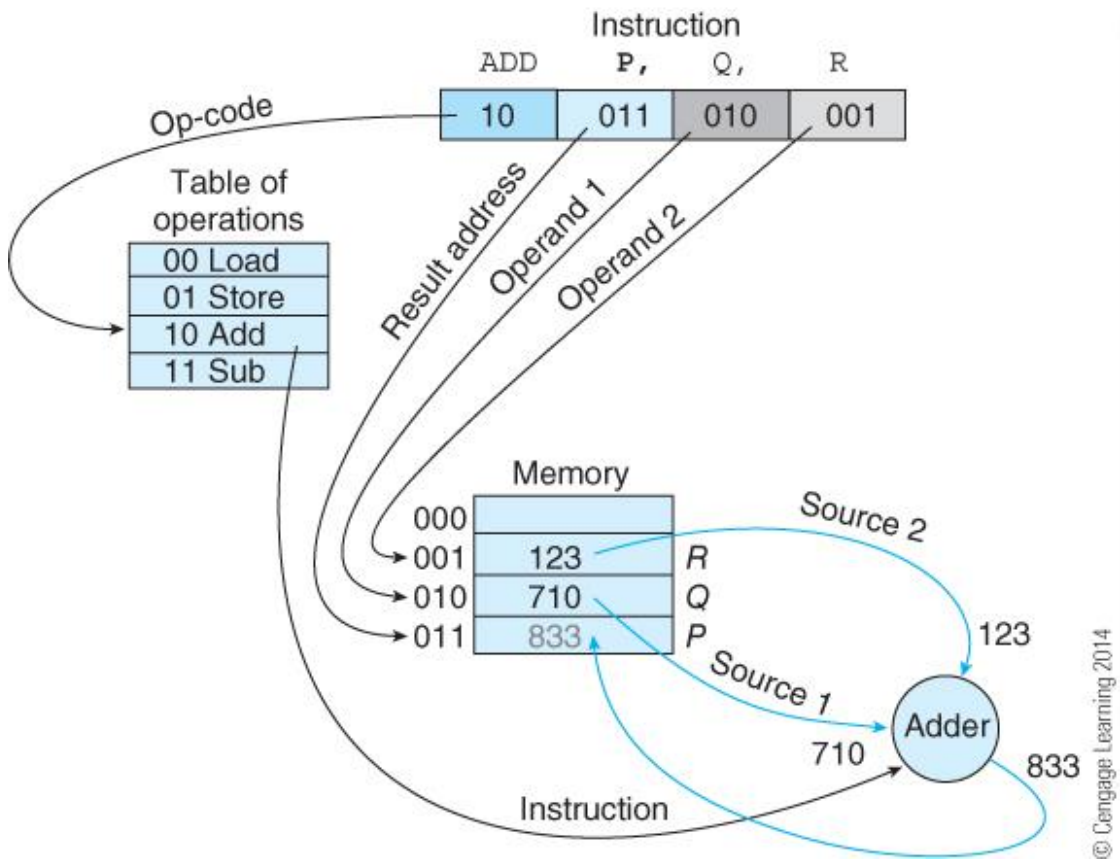
$\&P = 5$
$\&Q = 6$
$\&R = 7$

$P = 833$
$Q = 123$
$R = 710$

&

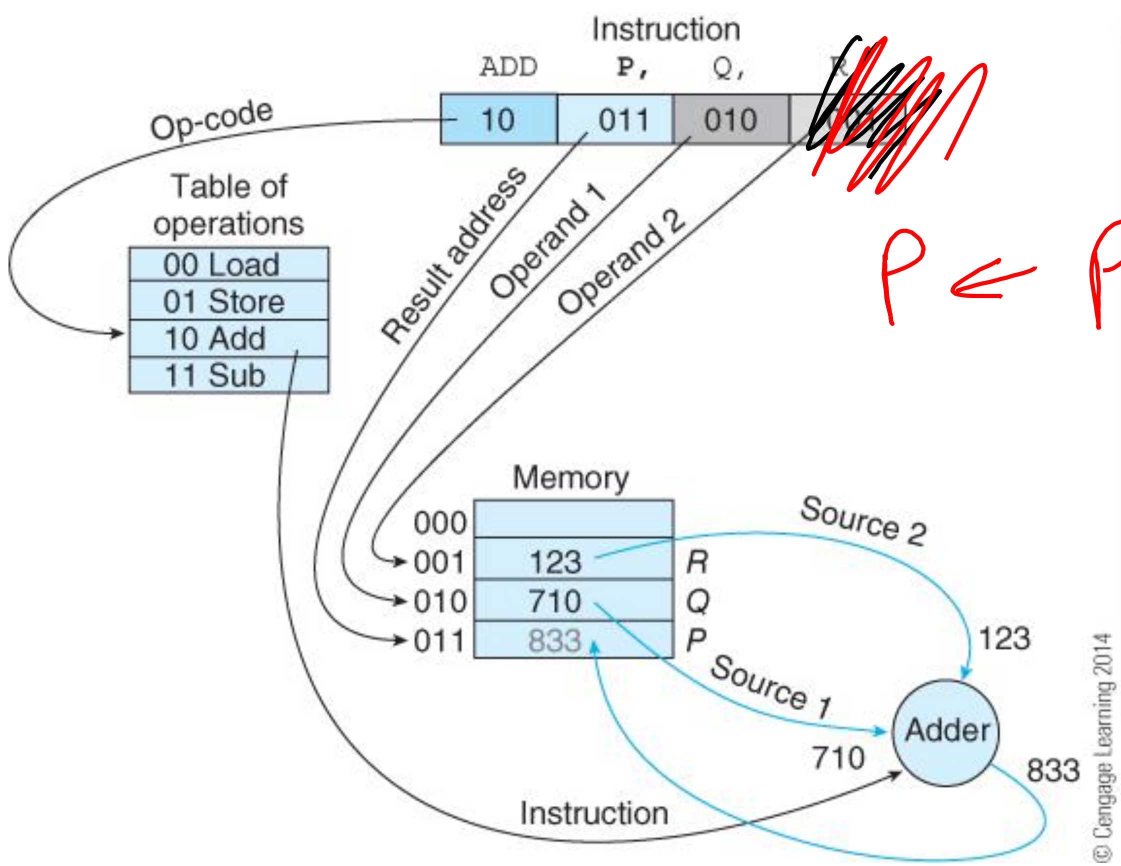
1.5 The Three-Address Instruction

FIGURE 1.14 Interpreting the instruction `ADD P,Q,R`



1.5 The Two-Address Instruction

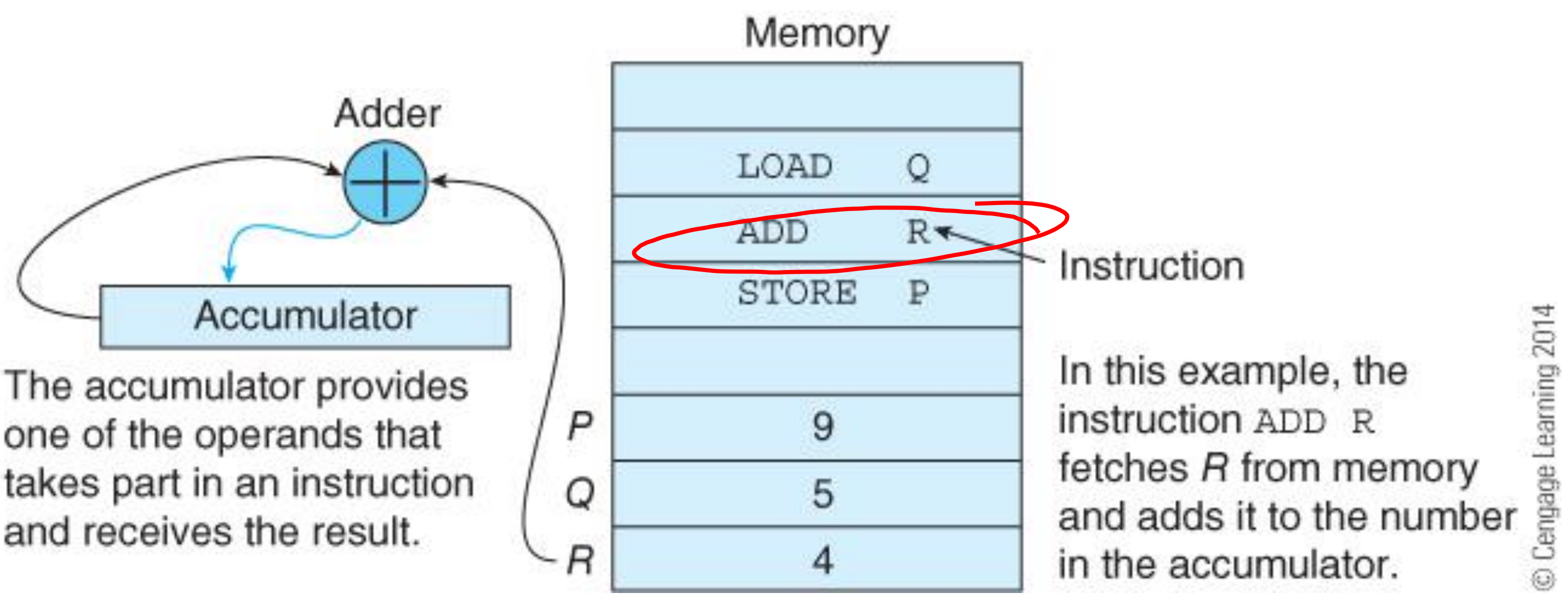
FIGURE 1.14 Interpreting the instruction `ADD P,Q,R`



$P \Leftarrow P + Q$

1.5 The One-Address Instruction

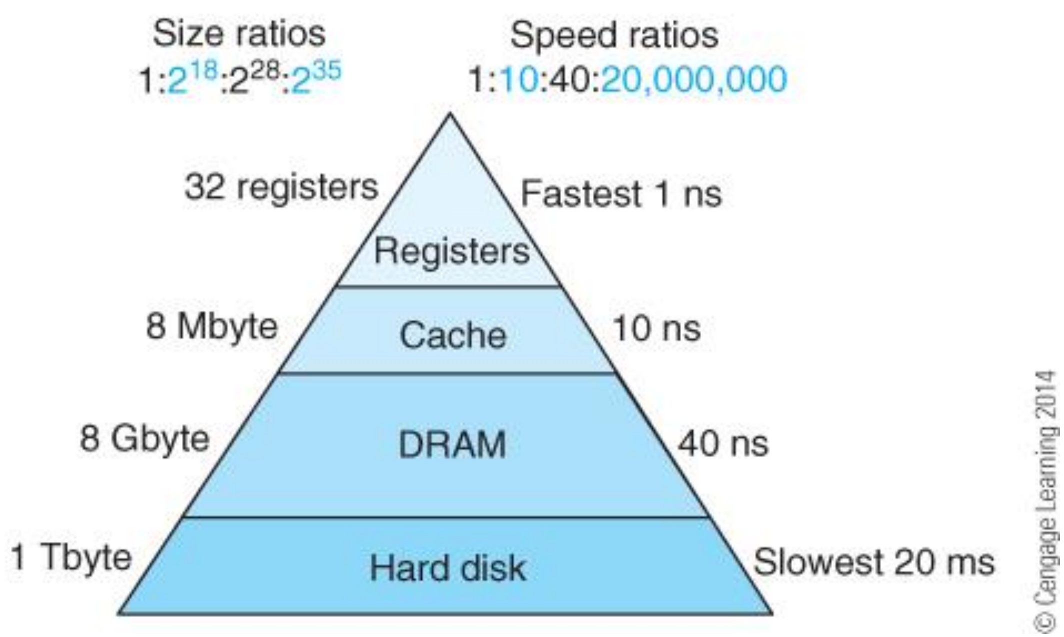
FIGURE 1.15 Single-operand instructions



$ACC \leftarrow ACC + R$

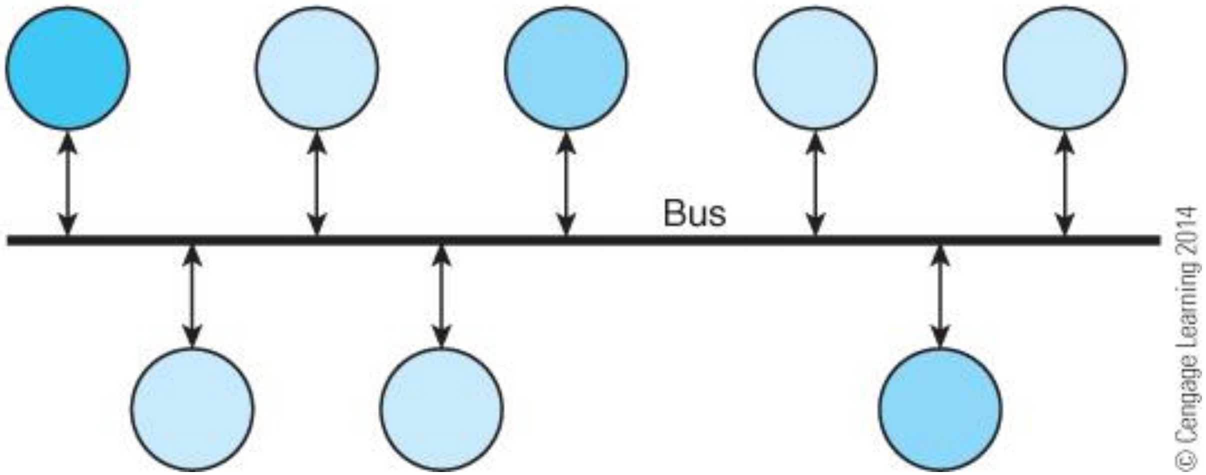
1.6 Overview of the Computer System - Memory

FIGURE 1.16 Memory hierarchy



1.6 Overview of the Computer System – Simple Bus

FIGURE 1.18 A common bus connecting all units



1.6 Overview of the Computer System - Buses

FIGURE 1.19 A system with two interconnected buses