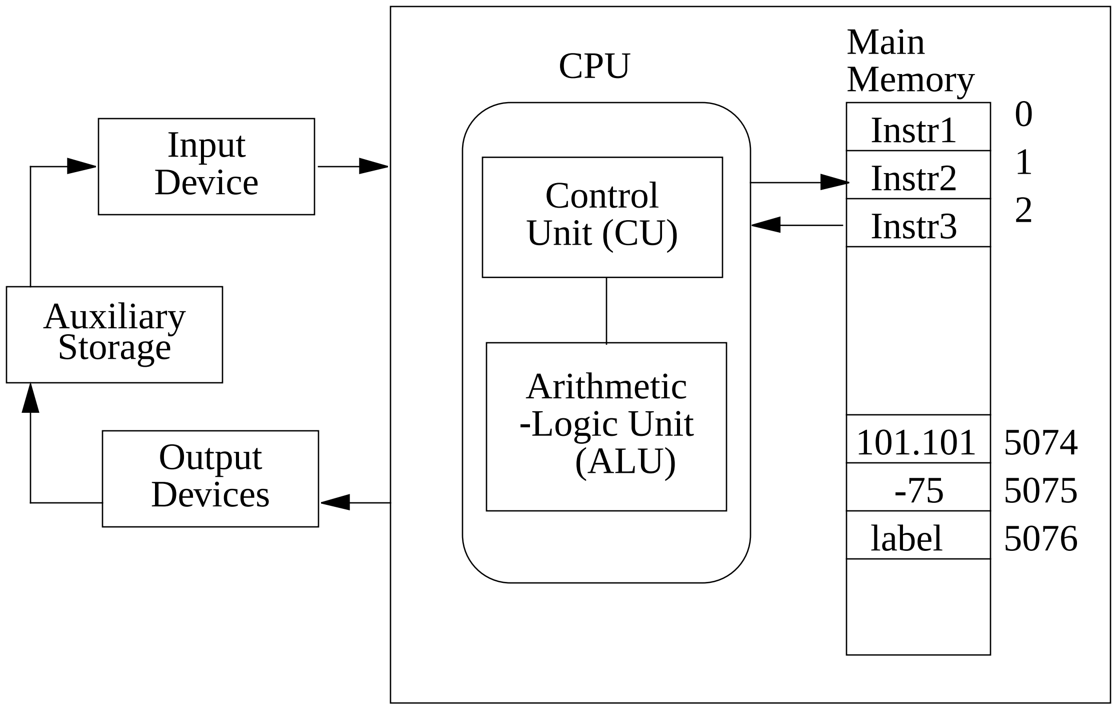
Chapter 1: General Topics

Digital Computer Hardware



Central Processing Unit (CPU):

- Transfers Information Info, out of, and between memory locations.

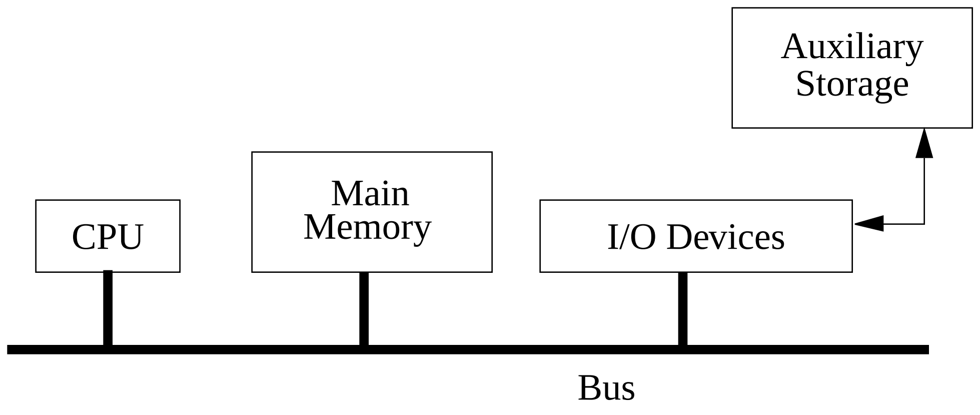
  Carries out instructions stored in memory.

Main (Internal) Memory:

- Stores Instructions and data values.

  Storage locations referenced (Internally) with integer values 0,1,2, . . .

Information transfer between hardware components is along an electrical connection called a Bus:



Special Architectures

Vector Processor

- a processor with multiple ALUs, which can simultaneously perform an arithmetic or logic operation on corresponding elements of two lists (vectors).

Multiprocessing System

- has multiple CPUs for simultaneous processing of different programs.

Supercomputer Systems - based on vector processor and multiprocessor technology.

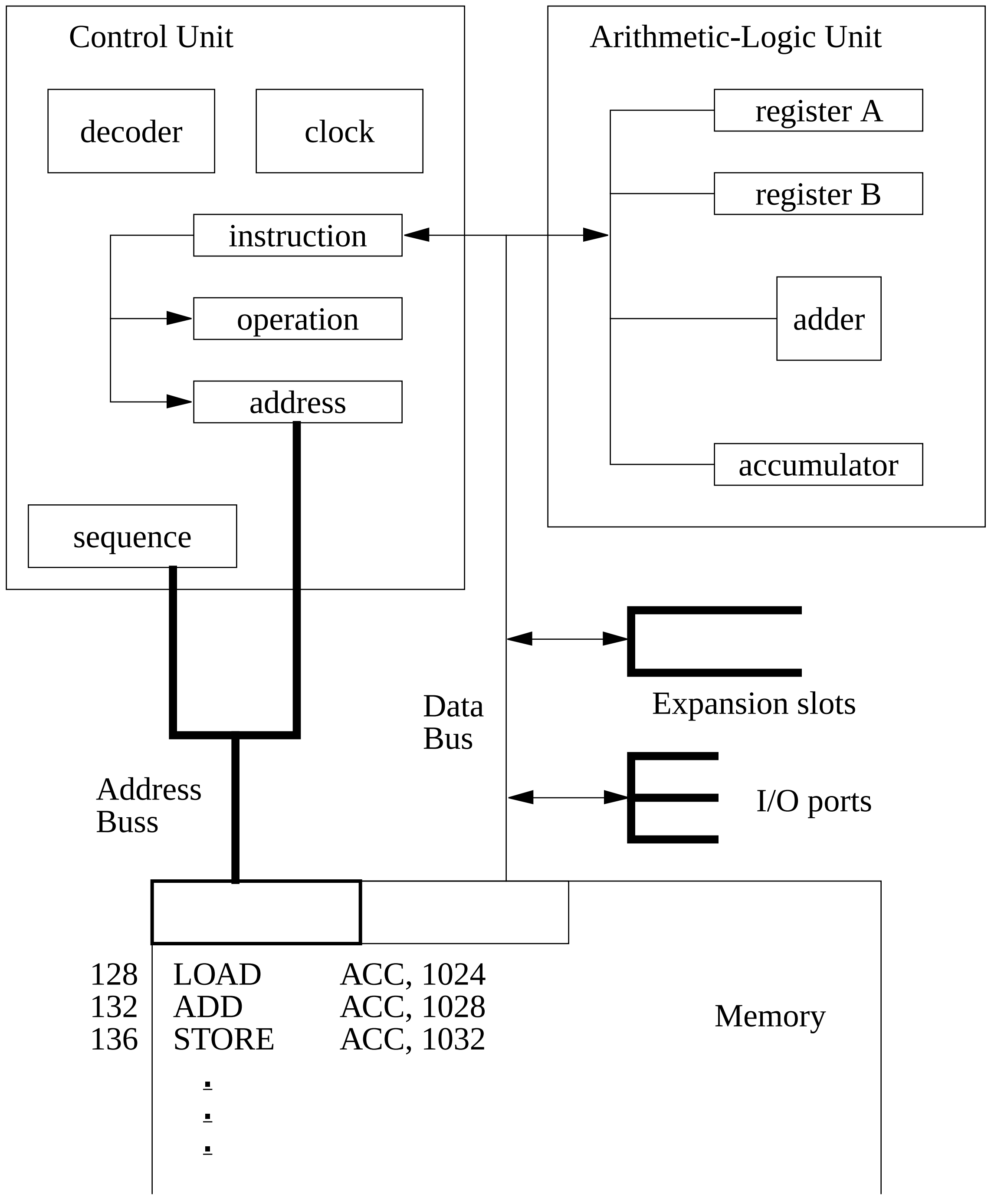
• Cray XMP

• Cyber 205

• Convex

• Connection Machine (“massively parallel”)

Basic Operation of CPU



CPU Circuitry

adder - ALU circuitry that performs arithmetic operations (add, subtract, multiply, divide) and comparisons.

decoder - CU circuitry that interprets and instruction.

clock - CU circuitry that synchronizes operations of the system, sending out millions of pulses (or “ticks”) per second. Clock speeds expressed in Megahertz (Mhz).

CPU Registers

- fast-access storage areas.

instruction register - holds instruction that CPU is currently executing.

operation register - holds operation code for instruction that CPU is currently processing.

address register - holds address for data needed to execute the instruction.

sequence register - holds address of next instruction to be executed.

accumulator - holds results of arithmetic operation.

general-purpose registers - such as A and B, hold data to be processed.

Memory Classifications

- **RAM** (Random-Access Memory)

Stores information, such as your programs, data, and “outline” font shapes, that can be both placed into and retrieved from storage locations.

- **ROM** (Read-Only Memory)

Stores system-related “stuff”, such as programs that are not to be altered by a user and fixed-shape bit-mapped fonts.

Memory Divisions

- **BIT** (BInary Digit)

Each bit position in main memory constructed with a “flip-flop” circuit, which can be flipped between two states (called 0 and 1).

- **Byte**

Storage size for one character of information (usually 8 bits).

Characters = letters, decimal digits, punctuation symbols, arithmetic operators, etc.

Have various Binary Codes for characters (ASII, EBCDIC).

- **Word**

Group of bits addressed and manipulated as a unit. Word length is the number of bits that can be transferred in one step between CPU and main memory. Word length also determines size of numbers that can be processed in one step by CPU. Usually, bus size equals word length, but some small computers transfer half a word at a time along bus.

Word length is typically an integral number of bytes, and values from 8 to 64 bits.

Memory Size

- Usually expressed in terms of number of bytes (characters) that can be stored.
- Units of measurement:

Kilobytes	(KB)
Megabytes	(MB)
Gigabytes	(GB)

where K = 1,024 = 2^{10}

($M = K^2$, $G = K^3$)

Examples:

64	KB = 65,536 bytes
512	KB = 524,28 bytes
40	MB = 41,943,040 bytes

Input Devices

Keyboard

Disk Drive

Magnetic Tape Drive

Card Reader

Paper Tape Reader

Mouse

Joystick

Track Ball

Light Pen

Tablet (“Digitizer Pad”)

Data Glove (Virtual-Reality Systems)

Touch-Sensitive Screen

Button Boxes

Dials, Switch Boxes

Optical Scanners

Magnetic-Ink Character Recognition Systems

Voice

Output Devices

Raster Video Monitor

Vector Displays, Plasma Panels, LCDs, etc.

Disk Drive

Magnetic Tape Drive *

Card Punch

Paper Tape Punch

Printer

Plotter

Head Sets (Virtual-Reality Systems)

Other 3D Display Devices

Stereoscopic Displays (Glasses + Raster Monitor)

Voice and Sound Systems

Presentation Devices:

Microfilm

35mm Slides

Overhead Transparencies

---

* External Storage Devices
(also called Secondary or Auxiliary Storage)

- Direct-Access Devices (Disks: Hard, Floppy
- Sequential-Access Devices (Tape)

Computer Software

- Programs (and their documentation)

System Software:

- Operating Systems
- Translators and Interpreters
- Editor Programs, etc.

Applications Software:

- Word Processors
- Desk-Top Publishing System
- Equation Makers (e.g., Math Type)
- Math-Stat Packages
- Mathematics Systems (e.g., Mathematica)
- Graphics Packages
- Accounting Packages
- Data-Management Packages
- User Programs

Programming - Process for obtaining a computer solution
to a problem.

1. Problem **Definition**

2. Refine, Generalize, Decompose the problem definition.
   (i.e., Identify subproblems, I/O, etc.)

3. Develop **Algorithm**
   (processing steps to solve problem)

4. Write the “Program” (**Code**).
   (instruction sequence to be carried out by the computer)

5. **Test and Debug** the Program.

6. Run Program.

{**Documentation** - prepare during development of program.}

Problem-Solving Example:

- **Problem Statement**

$$\text{Compute: } \sum_{k=1}^{10} \frac{1}{k} = \frac{1}{1} + \frac{1}{2} + \ldots + \frac{1}{10}$$

(a harmonic series)

- **Refinement of Problem Specification**

Generalize problem by computing:

$$\sum_{k=1}^{n} \frac{1}{k}$$ for "any" $n \geq 1$.

Also for other ranges, $k \neq 0$.
And for any input function $f(k)$.

Input = starting/stopping values for k.
Output = calculated sum.

No modularization or subparts needed for this simple problem.

- **Algorithm**

Compute sum being a loop.

i.e., repeatedly adding $1/k$ to previously calculated sum at each step.

Algorithm steps can be specified with

- Natural Language (e.g., English)
- Pseudocode
- Flowchart, or other type of flow diagram

  (typically used in documentation)

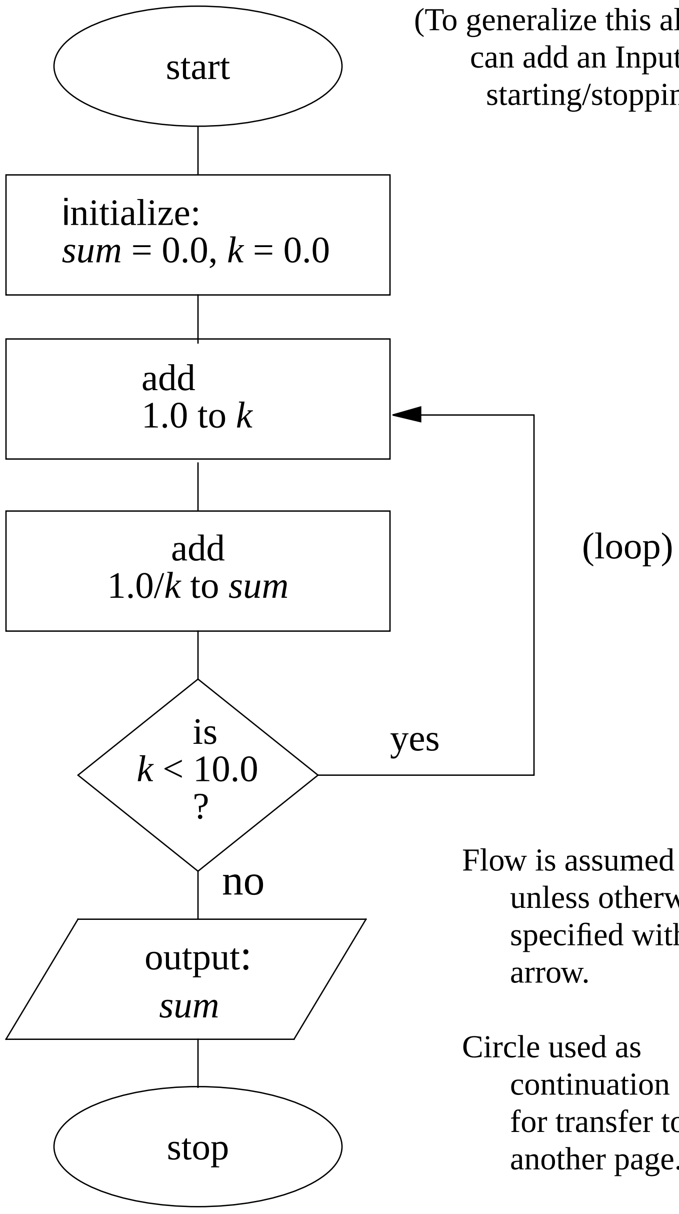
<u>Natural-Language Algorithm</u>

```
Store value 0.0 in storage location called k.
Store value 0.0 in storage location called sum.
Repeat following steps 10 times:
    increment k by the value 1.0
    increment sum by the value 1.0/k
Print out the value stored in location sum.
```

<u>Pseudocode Algorithm</u>

```
initialize: sum = 0.0
do (k = 1, 10)
    sum = sum + 1.0/k
enddo
print sum
```

Flowchart Algorithm



(To generalize this algorithm, we can add an Input box for starting/stopping values.)

Flow is assumed down unless otherwise specified with an arrow.

Circle used as continuation symbol for transfer to another page.

Next Step:

Write Program - the instruction sequence corresponding to the algorithm.

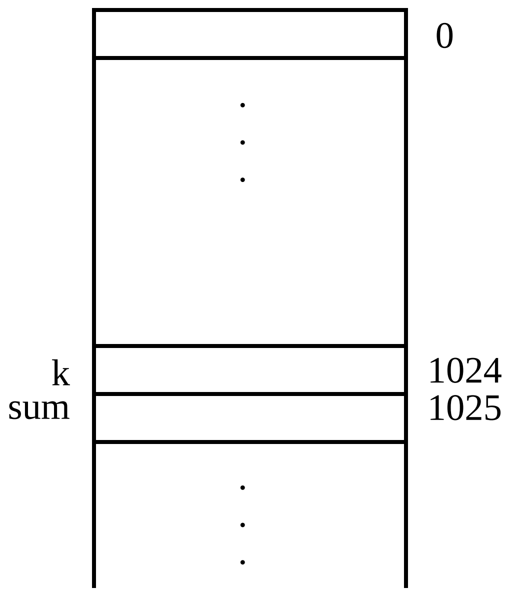
Then enter into computer.

<u>Execution Sequence</u>

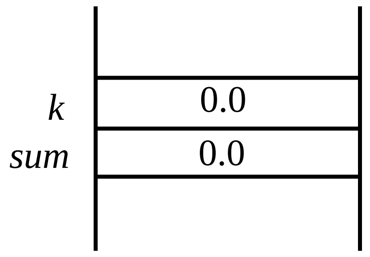
Computer Memory stores:

- **Instructions**
- **Data**

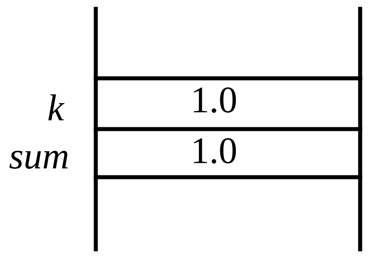
In our example problem, data storage needed for variable names: *k, sum* (also constants such as 10):



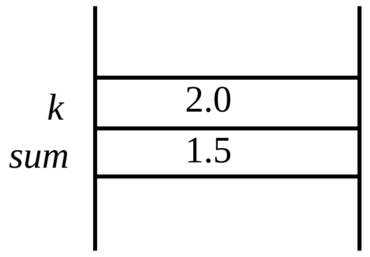
Initial data storage contents:



After the two *add* instructions are processed, the storage contents are:



Contents of location *k* are then checked against the value 10.0, and the *add* instructions repeated:



This process repeats until value in location *k* is 10.0 (and value in location *sum* is 2.9289. . .)

Then, contents of location *sum* is output.

<u>Programming Languages</u>

Classified as

√ **Low Level**

- **Machine Language**

  (binary-based code; machine dependent)

- **Assembly Language**

  (mnemonic form of machine language)

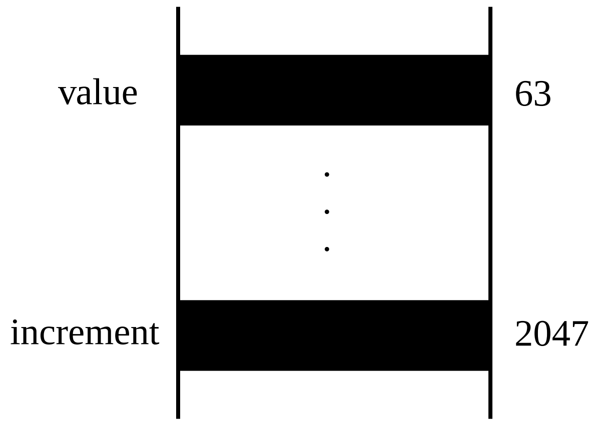
√ **High Level**

- Closer to natural languages
- Generally, machine independent
- Usually, several machine instructions are combined into one high-level instruction.
- <u>Examples</u>:

FORTRAN	
BASIC	(each language with
Pascal	various versions)
PL/I	
C	
Ada	
Lisp	
GPSS	
COBOL	

etc.

To illustrate differences in syntax for language levels, consider how computer could be instructed to subtract two numbers stored in locations 63 and 2047:



- **Machine Language:**

  011111000000111111011111111111

  (6-bit OpCode, 12-bit address fields with values 63 and 2047)

- **Assembly Language:**

  ```
  S    VALUE, INCR
  ```

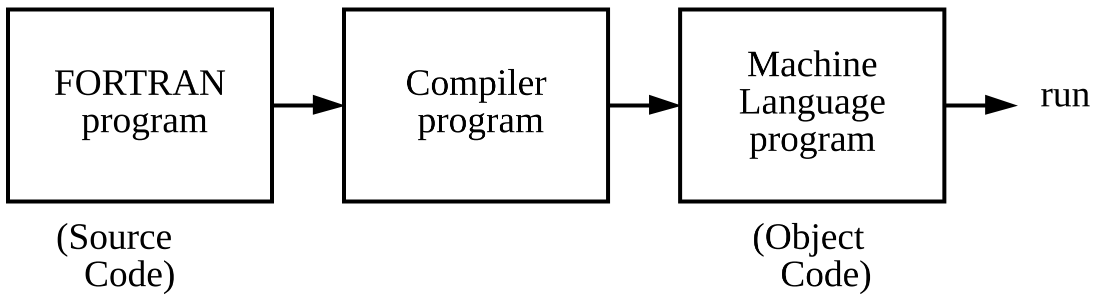
- **Pascal:**

  ```
  value := value - increment
  ```

Programs written in high-level languages must be converted to machine language.

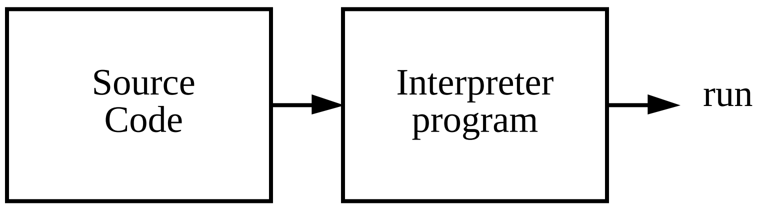
Two approaches:

(1) Compilation



(2) Interpretation

Used with versions of microcomputer BASIC, for example,



No object code created.

Each source code statement is interpreted (translated, executed) each time it is processed.

An assembly language program is converted to machine code with an Assembler Program.

Example Code for $\sum_{k=1}^{n} \frac{1}{k}$ with input value for n.

<u>FORTRAN 77</u>

```
      program series
      real k, sum            {reserves storage locations
      integer n
      sum = 0.0
      k = 0.0
      read*, n               {input a value for n
      do 10 k = 1, n
          sum = sum + 1/k
10    continue
      write (*,*) ‘sum = ‘, sum        {output sum
      stop                       {end of execution
      end                        {end of compilation
```

(Compiler converts all statements above to machine code then execute with “run” statement.)

<u>Mathematica</u>

```
sum = 0.0;
n = Input [“Input an integer.”];
Do [sum = sum + 1.0/k, {k, 1, n}]
Print [“sum = “, sum]
```

(Statements are interpreted and executed in sequence.)

(Each line ended with “return” key; statements executed with “enter” key; semicolon suppresses output.)

Programs written in high-level languages use:

- **<u>Symbol Names (Variables)</u>**

  - to designate memory locations; e.g.,

    k, *sum*

- **<u>Constants</u>**

  3.50, -17, "*sum* = " (or other string notation)
  (floating point, fixed point, character string)
  etc.

- **<u>Lists</u>**

  linear and nonlinear; e.g., arrays, tree
  structures

- **<u>Expressions</u>**

  - specifying operations on variables,
    constants, and lists; e.g.,

    (4.0/3.0) * *pi* * *radius* ^ 3

- **<u>Assignment Statements</u>**

  - assign a data value to a storage location;
  e.g.,
  value1 = *value2* (or other assign notation)
  sum = *sum* + 1.0/*k*

- **<u>Input-Output Statements</u>**

  Examples: *read*, *print*

- **<u>Loop Structures</u>**

  Examples: *do, for, while* - specify how many times a set of statements is to be processed, based on value assigned to a counter or result of a test condition.

- **<u>Decision Structures</u>**

  Typically of the form:

  if (condition) *then* (do something)

- **<u>Built-In and User-Defined Functions</u>**

  sin, cos, tan, sqrt, exp, etc. (built-in) special-application fcns (user-defined)

- **<u>Subprograms</u>**

  Provide for modularization of programs. (Can be user-defined functions.)