# ⇒ Levels of Programming Languages:-

→ Language Categories:-

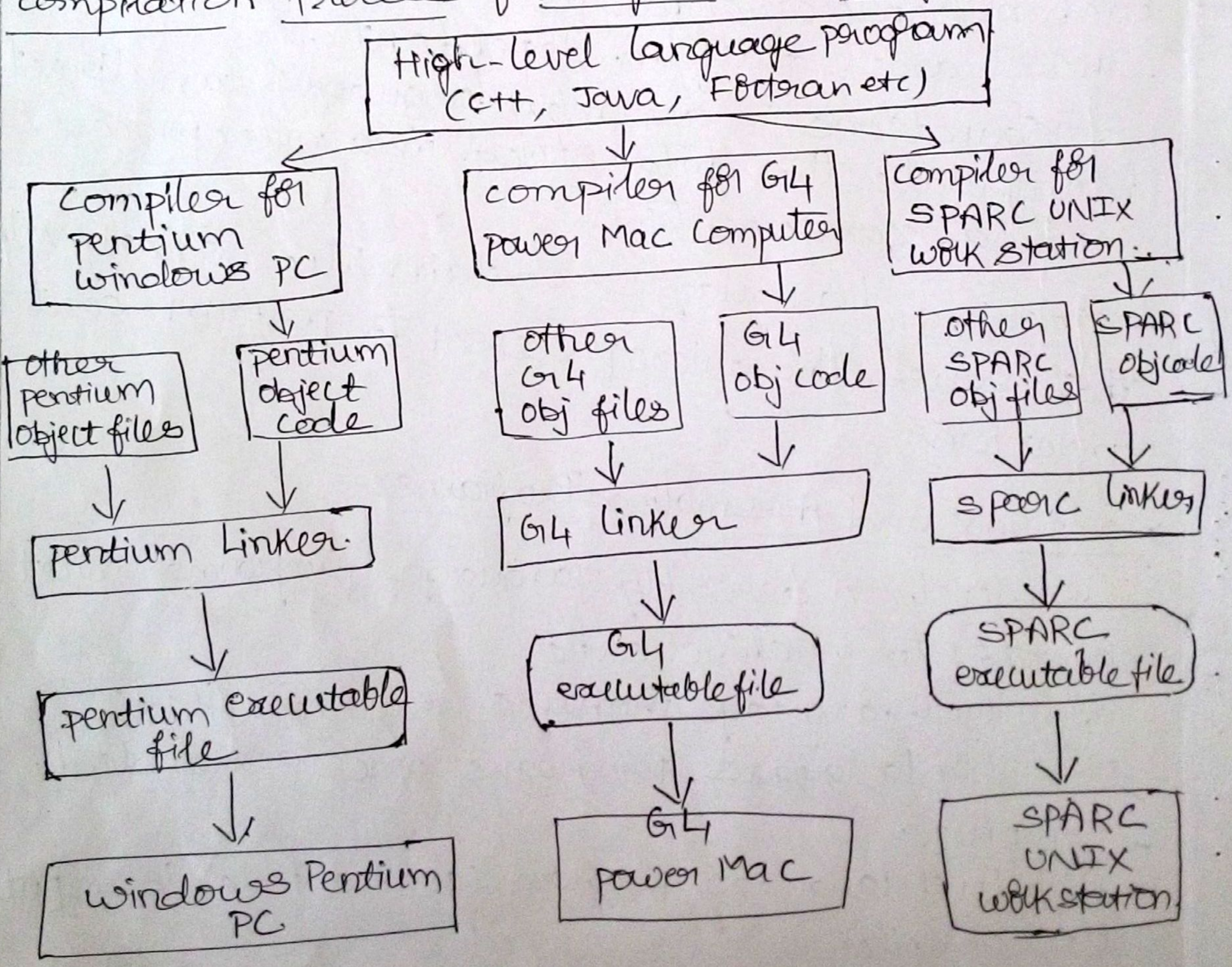
- Computer programming languages are divided into three categories.
  - High-level languages - C++, Java
  - Assembly Languages
  - Machine Languages / Low-level languages
- Languages, with the highest level of abstraction are collectively refeared to as high-level languages.
- High-level languages are platform independent.
- Assembly languages are at a much lower level of abstraction. Each microprocessor has its own assembly language.
- These are platform-dependent.
- Instructions in assembly languages can directly manipulate the data stored in a microprocessor's internal components.
- Machine Languages contain the binary values that cause the microprocessor to perform certain operations.

→ Compiling and Assembling Programs:-

- High-level & assembly language programs must be converted to machine code.
- High-level language programs are compiled; assembly language programs are assembled.
- Compiler:-
  - High-level language program (source code) is input to the compiler
  - checks every statement of source code for any syntax errors; if no errors generate an object code.

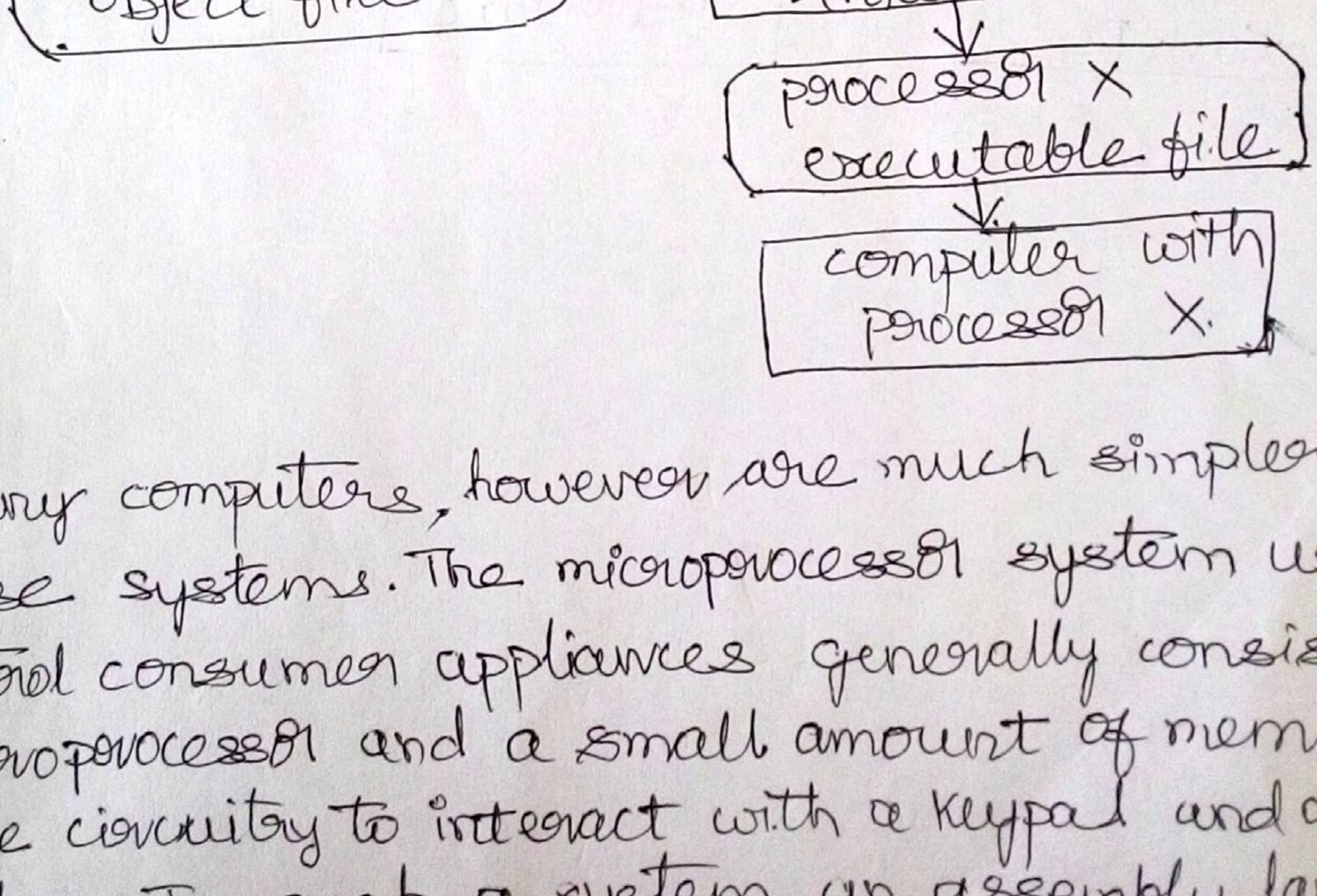
- A Linker combines object code with any other required object code to get executable file.
- A Loader copies the executable file into memory. Microprocessor then runs the machine code contained in that file.
- As high-level languages are platform-independent, the same high-level source code can be compiled to run on different microprocessors and OS, & computing platforms.

compilation Process for high-level programs:-



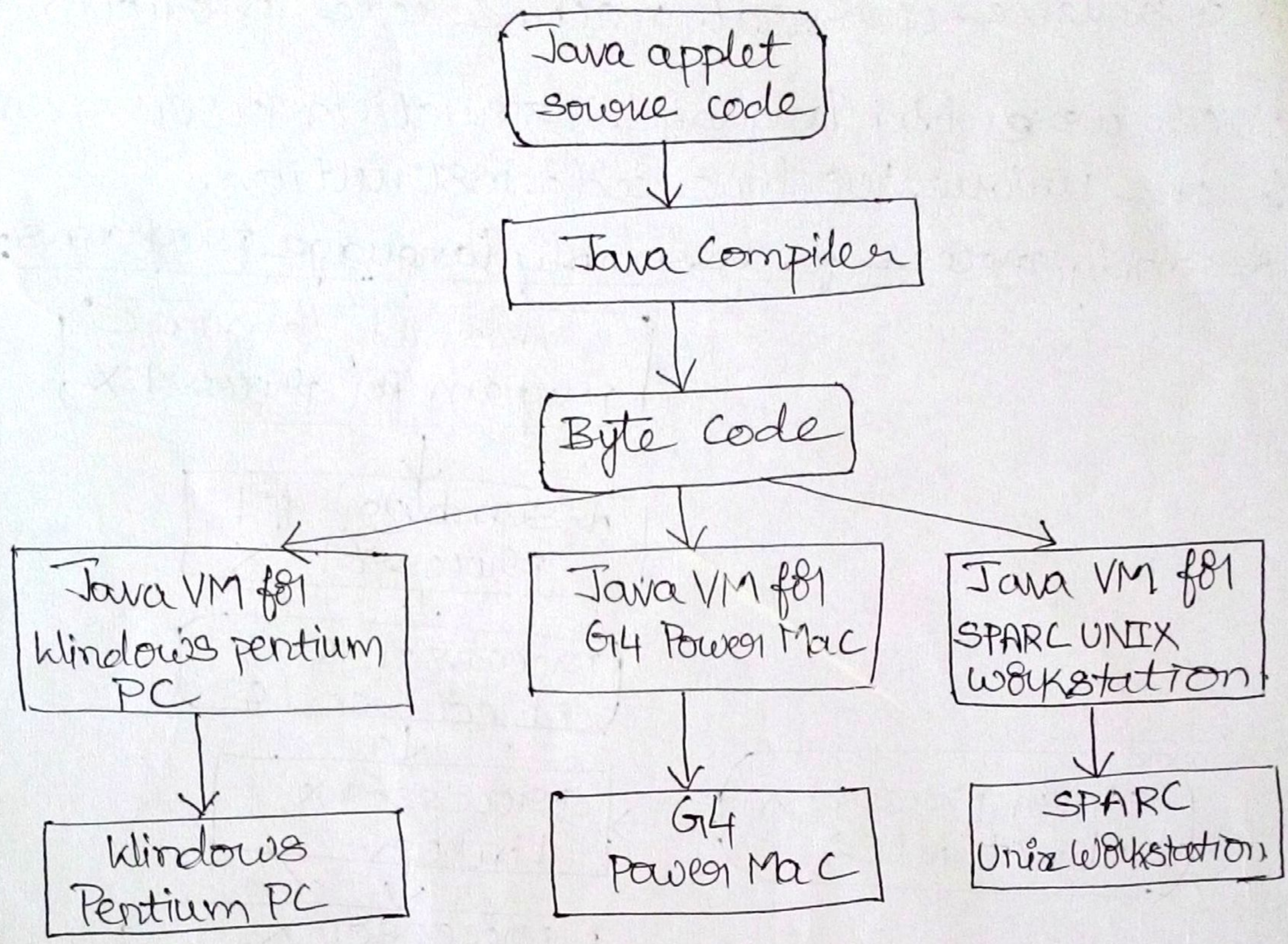
- A high-level language statement is usually converted to a sequence of several machine code instructions
- Each assembly language instruction corresponds to one unique machine code instruction.

Assembly process for assembly language programs:-




- Many computers, however are much simpler than these systems. The microprocessor system used to control consumer appliances generally consist of a microprocessor and a small amount of memory and some circuitry to interact with a keypad and a digital display. In such a system, an assembly language program would be assembled to generate object code, but this might be the end of the process.

## → compilation process for Java applets :-



## ⇒ Assembly Language Instructions:-

- Assembly language instructions can be grouped together based on the types of operations they perform.
  - Data Transfer Instructions
  - Data operation Instructions
  - Program control Instructions

### → Data Transfer Instructions.

- These instruction only copy the value to its destination.
- These instructions typically perform one of the following transfers.
  - Load data from memory into the microprocessor.
  - Store data from the microprocessor into memory.
  - Move data within the microprocessor
  - Input data to the microprocessor.
  - output data from the microprocessor.

### → Data Operation Instructions:-

- These perform some operation using one or two data values (operands) and store the result.
- Arithmetic instruction makeup a large part of the data operation instructions. (ADD, SUB, MUL, DIV).
- A special class of Arithmetic instructions are floating point instructions.
- Logic instructions perform basic logical operations on data. They are AND, OR & XOR.
- shift instructions, shift the bits of a data value.

→ Program Control Instructions:-

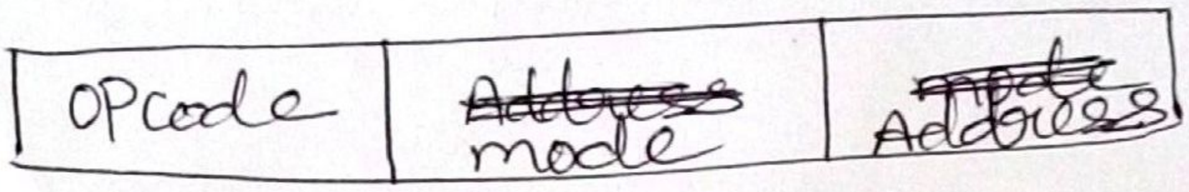
- A Jump & branch instruction is commonly used to go to another part of the program.
- Assembly language instruction set may include instruction to call and return from subroutines.
- An assembly language instruction set may include specific instructions to generate interrupts; these are called software interrupts.
- Halt instruction causes a microprocessor to stop executing instructions, such as at the end of a program.

Data types:-

- integers < signed $-2^{n-1}$ to $2^{n-1}-1$
  unsigned 0 to $2^n-1$
- float
- Boolean values TRUE & FALSE.
- character data.
  characters are stored as binary values encoded using ASCII, EBCDIC, UNICODE.
  ASCII → American Standard Code for Information Interchange
  EBCDIC - Extended Binary Coded Decimal Interchange Code
  UNICODE.

## ⇒ Instruction Formats :-

→ The control unit of CPU interpret each instruction code and provide the necessary control functions needed to process the instruction.




Instruction Format.

- Operationcode (opcode) - specifies the operation to be performed.
- Address - designates a memory address or a processor register.
- mode - designates the way the operand or the effective address is determined.

→ Computers may have instructions of several different lengths containing varying number of addresses. The number of address fields in the instruction format of a computer depends on the internal organization of its registers.

Most computers fall into one of three types of CPU organizations:

1. Single accumulator organization
2. General register organization
3. Stack organization.

① ⇒ uses one address field.

Eg:- ADD X

ADD - Addition
X - address of operand.

AC ← AC + M[X].

② ⇒ needs three register address fields.

ADD R1, R2, R3

R1 ← R2 + R3

- if the destination register is the same as one of the source registers; the no: of address fields in the instruction can be reduced from three to two.

ADD R1, R2 R1 ← R1+R2.

Eg:

MOV R1, R2 denotes transfer R1 ← R2

(3) ⇒ PUSH and POP instruction.

PUSH X will push the word at address X to the top of the stack.

operation-type instructions do not need an address field in stack-organized computer.

ADD this operation POP the top two numbers from stack, add. and push the sum onto the stack.

- TO illustrate the influence of the number of addresses on computer programs, let us consider the evaluation of arithmetic statement

$$X = (A+B)*(C+D)$$

## Three-Address Instructions:

ADD R1, A, B R1 ← M[A] + M[B]

ADD R2, C, D R2 ← M[C] + M[D]

MUL X, R1, R2 M[X] ← R1 * R2

## Two - Address Instructions

| | | |
|---|---|---|
| MOV | R1, A | R1 ← M[A] |
| ADD | R1, B | R1 ← R1 + M[B] |
| MOV | R2, C | R2 ← M[C] |
| ADD | R2, D | R2 ← R2 + M[D] |
| MUL | R1, R2 | R1 ← R1 * R2 |
| MOV | X, R1 | M[X] ← R1 |

## One - Address Instructions :-

- use an implied accumulator (AC) register

| | |
|---|---|
| LOAD A | AC ← M[A] |
| ADD B | AC ← AC + M[B] |
| STORE T | M[T] ← AC |
| LOAD C | AC ← M[C] |
| ADD D | AC ← AC + M[D] |
| MUL T | AC ← AC * M[T] |
| STORE X | M[X] ← AC |

## Zero - Address Instructions :-

TOS = TOP of stack.

| | |
|---|---|
| PUSH A | TOS ← A |
| PUSH B | TOS ← B |
| ADD | TOS ← (A+B) |
| PUSH C | TOS ← C |
| PUSH D | TOS ← D |
| ADD | TOS ← (C+D) |
| MUL | TOS ← (C+D) * (A+B) |
| POP X | M[X] ← TOS |

To evaluate arithmetic expressions in a stack computer, it is necessary to convert the expression into reverse Polish notation.

⇒ Eg:-

Evaluate the arithmetic statement:

$$X = \frac{A - B + C * (D * E - F)}{G + H * K}$$

(a) Using a general register computer with three address instructions

(b) with two address instructions

(c) one address instruction. - accumulator type computer

(d) using a stack organized computer with zero-address operation instruction