Comparative Study of Computer Languages

Computer Organisation and Evolution of Computer Languages

Part III

R. Ramaswamy

A computer does not have only the single function of computation but also several other functions like reading the data and the program, storing them, doing arithmetic as per the program, outputting the result in a printed form in the desired format and so on. Each of these functions are done by separate functional units, but all of them act in coordination under a common control. Thus, it is said that a computer is a huge system composed of several related functional units which act under a common control to achieve a common objective.

A system is a composite entity made of interrelated elementary units or sub-systems which jointly perform a task under the direction of a single control. The human being is also a system, composed of different functional units acting under a single control –the brain. In short, a system is a composite entity whose constituents work together to achieve a common goal. The constituents of a computer also act together under a common control to accomplish the common job of data processing and so we say that the computer is a system.

Computer organisation

Computer is a system composed of different functional units for performing different jobs. It consists of the following major functional units:

1. Input unit,
2. Memory unit,
3. Arithmetic logic unit,
4. Output unit, and
5. Control unit.

The function of the input unit is to read the data and the program while the memory unit stores the data, the program and the result. The function of the arithmetic logic unit is to do the necessary computation as per the program of instructions and the output unit prints the result in a useful form.

There must be somebody to control the activities of these functional units. Just as the brain controls the functioning of the different human organs, the control unit controls the functions of the other units of the computer. The different functional units and their interrelationship is schematically represented in Fig. 6.

The thick lines indicate the flow of data, whereas the dotted lines denote the flow of control information. One can notice that the control pervades through all the functional units, whereas the data flows between specific units only. As far as the user is concerned, each functional unit is a black box, i.e. one need not know what these functional units are made of and how they do their assigned job. Suffice it to know that the different functional units are made of elec-

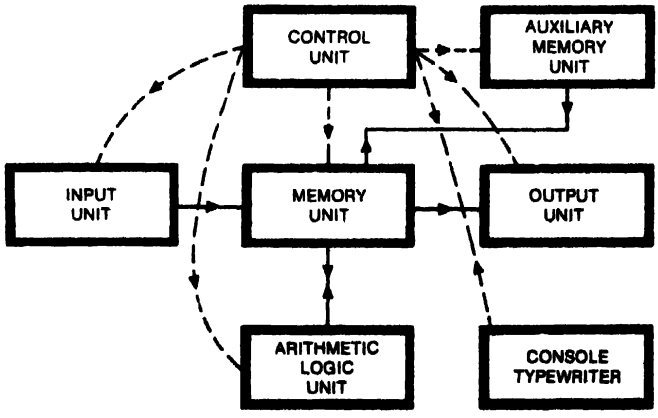

Fig. 6: Schematic representation of the functional units in a computer.

tronic components which are capable of doing the above functions quickly, accurately, reliably and completely automatically.

In addition to the automatic control, there is also a manual control provided in the computer. This is known as the console. Messages can be communicated through the typewriter attached to the console called the console typewriter. The computer will respond to the queries of the operator by typing messages on the typewriter or by displaying on a television screen attached to the console. This is called the visual display unit (VDU).

The computer can do any processing on the data only if the program and the data reside on the main memory. For solving large problems with large volume of data, the computer requires large memory space. To provide memory space within the computer will be relatively costly and so the computers are provided with auxiliary memory which is located outside. Just as the humans use paper media as their auxiliary memory, computers use magnetic discs.

When the auxiliary memory is attached to the computer, the control unit takes charge of the auxiliary memory unit also and directs it to transfer information to the main memory and back, as and when required. The combination of the control unit, the memory unit and the arithmetic logic unit is called the central processor unit or the CPU.

Evolution of computer languages

The development of computer languages started with the discovery of the computer in 1948. The earliest computer language developed for the machine is called the machine language. A machine language program is a set of instructions written for implementation by the machine using only the two digits - 1 and 0.

Every instruction or data is coded by a suitable combination of 1s and 0s. Just as a combination of dots and dashes is used to send messages by telegraph, a suitable combination of 1s and 0s is used to establish communication with the machines. So a machine language program consists of a coded string of 1s and 0s, each string code having some specific meaning for the computer.

The next development is the discovery of the assembly language, in which the binary codes are replaced by alphabetic codes. Though the alphabetic codes are somewhat more meaningful than number codes, they are difficult to learn, since they are not akin to spoken languages.

Then comes the discovery of high level languages which are more akin to spoken languages. FORTRAN, COBOL, BASIC, PL/1 and PASCAL fall in the category of high level languages. They are easy to learn, since they are very much akin to the spoken English language.

The ultimate aim of computer scientists is to make the computers understand the spoken languages, so that man will be relieved from the burden of learning a new language for the sake of establishing communication with the computers. Till then, man has to study these high level languages if at all he wants to communicate with the computer machines.

Machine language programs

We have said that a machine language program will appear as a coded string of numbers 1 and 0 in the binary notation. For example, a machine language program to instruct the computer to add two numbers stored in the locations I and J, and finally store the result in the location K may look somewhat as follows:

```
0001  1111  1000
0011  1110  1000
0111  1000  1100
```

It is obvious that the above string of 1s and 0s is completely unintelligible to the layman. Even for the well versed, it is a difficult job to remember the code correctly and write it without making any mistakes. What do these strings of binary digits mean?

Taking the first line, the string of first four digits '0001' may mean 'store'. The string of next four digits '1111' may mean 'the contents of the location I'. The string of the last four digits '1000' may mean 'a location called the accumulator'. Thus the combination of 12 binary digits in the first line may mean 'store the contents of I in the accumulator'.

The 12 digits in the second line may mean 'add the contents of J with the number stored in the accumulator and store the result in the accumulator'. The 12-digit string in the third line may mean, 'move the contents of the accumulator to the location K'.

It can be noticed that since the machine language program is a string of binary digits having no resemblance to any spoken language, checking the program becomes very difficult, if not impossible. Again different computers have different codes and so the programmer has to learn the codes for each machine, which is time consuming and laborious.

Another serious drawback with the machine language is that one must have a knowledge of the machine parts to write the program, i.e. the machine language is highly machine dependent, and so a programmer who does not know about the machine cannot write a good machine language program. No wonder that man's efforts were directed towards reducing this communication gap between the man and the machine and bringing the machines nearer to the common man.

Assembly language programs

It has been seen that in the machine language, number codes are used to identify the different memory locations, their contents and the different operations. In the assembly language, the number codes are replaced by mnemonic name codes, which are easier to remember and identify. For example, an assembly language program for the problem given in the last section may look somewhat as follows:

```
LDA I    (meaning load I in the accumulator)
ADD J    (meaning add the contents of J with the
          contents of the accumulator)
```

MVA K (meaning move the contents of the accumulator to the location J)

It can be seen from the above that there is one to one correspondence between the machine language and the assembly language codes. Assembly languages are simply modified machine languages, where the binary codes are replaced by alphabetic codes which are somewhat easier to identify. In other respects, the assembly languages also suffer from the same disadvantages as machine languages, i.e. the assembly languages are also highly machine dependent and learning them is also laborious and time consuming.

The machine languages and the assembly languages are called the low level languages. Efforts made in the direction of making the program more meaningful and compact culminated in the development of the so-called high level languages.

High level languages

High level languages are more akin to English. The formulae used for computation look almost like algebraic formulae and the instructions for doing the different operations also look almost like ordinary English. In addition, the instructions are macros, in that one high level language instruction may be equal to a large number of machine language instructions, thus making the program very compact. Some of the popular languages in current use are FORTRAN, COBOL, BASIC, PL/1 and PASCAL.

The vast gap between the machine or the assembly languages and the high level languages can be seen in the following versions of the high level language instructions for the problem given earlier:

K = I + J (FORTRAN version)
ADD I TO J GIVING K (COBOL version)
K = I + J (BASIC version)
K = I + J (PL/1 version)
K: = I + J (PASCAL version)

It can be seen that the above instructions, or statements as they are called, are more intelligible than any of the machine or the assembly language instructions. We also notice that one statement in a high level language is equivalent to three statements in the low level language. In fact, it can be many more. That is why, the high level languages are macros. Again high level languages are practically machine independent and a person who has not seen a machine and does not know the machine parts can still write a good program following the rules of the language.

Whatever the high level language in which the humans write the programs for the problems, the computer will understand only the machine language. So the high level language programs have to be translated into the machine language. This is done by a special program called the compiler.

Compiler programs are used to translate the high level language programs to machine language programs. The high level language programs are called the source programs. The machine language program is called the object program.

The compiler programs check the grammar of the source program before translating the same to object programs. Any grammatical or syntax error in the source program will be pointed out immediately by the compiler program. Only if the source program is grammatically correct, the compiler program will translate it to the object program.

Requirements of high level languages

Earlier it was mentioned that the ordinary spoken languages like English are not suitable for communication with the computers. It is because these languages do not have unambiguous meanings for a particular set of words. Computers being machines cannot distinguish ambiguities in meanings. The computer requires to be exactly told in an unambiguous language as to what is to be done. One set of words must mean only one thing. Hence, any computer language must satisfy the following requirements:

1. The language must be context free.
2. The language must be unambiguous.
3. The language must be both semantically and syntactically exact.

Consider the following statements in English:

1. He is sitting under the table.
2. He is using a logarithm table.

The meaning of the word 'table' in the above two statements depends on the context. Such context dependence is not suitable for computer languages and that is the reason why the spoken languages are unsuitable for computers. Only one meaning can be ascribed to a structure or a string of characters in a computer language.

Let us now look at the following statement: 'They are flying kites'. This may mean that some people referred to by 'they' are flying kites. In another sense, 'they' refer to flying kites which are some sort of insects. Thus, it can be said that the language is grammatically or syntactically ambiguous, since the same pronoun 'they' refer to two different things. We say that the grammar or the syntax is not unambiguous, but the computer cannot distinguish such syntactical unambiguities.

Let us now consider the following statement. 'The table is eating the chair'. The sentence is grammatically correct, but its meaning is absurd, i.e. the semantics or the meaning is illogical. Computer languages must be both syntactically exact and semantically logical.

The development of computer languages requires lot of skill and ingenuity together with a knowledge of the anatomy of the computer. The high level languages have been so ingenuously developed that man does not see the machine at all when he learns the language but only sees the grammar. In other words, the grammar of the computer language makes the computer transparent to man. Thus we say that the high level computer languages are machine independent. It is this single feature that makes the learning of high level

computer languages easy for all.

Computer languages and microcomputers

With the advent of a large number of microcomputers in the market, the different manufacturers have made some changes in the standard rules originally prescribed for the different languages. This has resulted in the appearance of different versions for the same language. Fortunately, the variations are only minor in nature and the readers can refer to the minor changes by referring to the manual of that particular computer. When one becomes a computer professional, one must refer only to manuals. The programming lessons and other programming books must be consulted only for learning the general philosophy of computer languages.

In this serial, the character sets in the different languages are introduced first, followed by the rules for coding the constants or the different data types dealt with by the different languages, the rules for coding the variables or identifiers and special words if any in the different languages, facts about variables and constants in the different computer languages, the rules for coding arithmetical, relational and logical operators and forming expressions in different languages.

It has already been pointed out that statements in computer languages give complete meaning to the computer just like sentences give complete meaning to the humans in spoken languages. So, the rules for coding different statements for doing different jobs are dealt with next.

Just as a plot or a theme is required for writing stories, in a similar way algorithms are required to write programs. The development of algorithms for different problems are also described.

A computer program operates on data. Data has to be properly arranged and organised for easy access by the computer. It can be kept independent and grouped into arrays or records. Different applications require different types of groupings. We have to study how the different languages support the different data groupings or data structures as they are called.

Finally, we have to study complete programs for different types of problems (scientific and business) in the different languages.

Let us first study the character sets used in the different computer languages.

FORTRAN characters

The first thing to do before starting the study of any language is to learn the characters used in that language. There are three types of characters in every language, namely, the alphabetic, numeric and special. The following are the different types of characters used in FORTRAN for coding the various entities:

Alphabetic: A B C D E F G H I J K L M N
O P Q R S T U V W X Y Z (26)

Numeric: 0 1 2 3 4 5 6 7 8 9 (10)

Special:
+ plus
- minus
/ slash (division symbol)
* asterisk (multiplication symbol)
= equality sign (assignment operator symbol)
. decimal point
, comma
(opening bracket
) closing bracket
' quote mark
$ dollar sign (11)

In all, there are 26 + 10 + 11, i.e. 47 characters used in FORTRAN. Since we are already familiar with these characters, learning the character set in FORTRAN does not give rise to any problem.

COBOL characters

The following are the characters used in COBOL for coding purposes.

Alphabetic: A B C D E F G H I J K L M N
O P Q R S T U V W X Y Z (26)

Numeric: 0 1 2 3 4 5 6 7 8 9 (10)

Special:
+ plus
- minus or hyphen symbol
/ slash (division symbol)
* asterisk (multiplication symbol)
= equality sign (assignment operator symbol)
> greater than
< less than
. period or decimal point
, comma
; semicolon
" quote mark
(opening bracket
) closing bracket
$ dollar sign
blank (15)

There are 26 + 10 + 15, i.e. 51 characters in all used for coding purposes in COBOL. Since the character set is taken from English alphabets, Arabic numerals and the symbols used in conventional algebra, there is no special difficulty in learning the scripts of COBOL.

BASIC characters

Alphabetic: A B C D E F G I J K L M
O P Q R W X Y Z (26)

Numeric: 0 1 2 3 8 9 (10)

Special:
+ plus
- minus
/ slash (division symbol)
* asterisk (multiplication symbol)
^ exponentiation operator symbol
= equality sign (assignment operator)
. decimal point
, comma
; semicolon
: colon
" quote mark
(opening bracket
) closing bracket
> greater than
< less than
? question mark
% percentage
# hash symbol
$ dollar sign
\ backslash (20)

Here, the total number of characters are 26 + 10 + 20, i.e. 56 characters used for coding purposes in BASIC. As in other computer languages, these characters are familiar ones and so do not present any difficulty in learning them.

PL/1 characters

The following are the characters used in PL/1 for coding purposes:

Alphabetic: A B C D E F G H I J K L M N
O P Q R S T U V W X Y Z (26)

Numeric: 0 1 2 3 4 5 6 7 8 9 (10)

Special:
+ plus
- minus
/ slash (division symbol)
* asterisk (multiplication symbol)
= equality sign (assignment operator symbol)
_ break or underscore character
(opening bracket
) closing bracket
. decimal point
, comma
; semicolon
: colon
' quote mark
$ dollar sign
> greater than
< less than
& AND operator symbol
¬ NOT operator symbol
| OR operator symbol
% percent
? question mark (21)

In all there are 26 + 10 + 21, i.e. 57 characters used in PL/1 for coding purposes. The symbol ^ is alternatively used for NOT operator. The symbols \ and | are alternatively used for AND operator. So, total number of characters come to 60. Since these characters are familiar ones, no further explanation is required.

PASCAL characters

The following is the character set used in PASCAL for coding purposes.

Alphabetic: A B C D E F G H I J K L M
O P Q R S T U V W X Y Z (26)

Numeric: 0 1 2 3 4 5 6 7 8 9 (10)

Special:
+ plus
- minus
/ slash (division symbol)
* asterisk (multiplication symbol)
= equality symbol
:= assignment operator
. decimal point
, comma
; semicolon
: colon
' quote mark
(opening bracket
) closing bracket
[opening square bracket
] closing square bracket
> greater than
< less than
↑ pointer (18)

Totally, there are 26 + 10 + 18, i.e. 54 characters used in PASCAL for coding purposes. All these characters are familiar ones as in other computer languages.

Blank character

Blank is treated as character only in COBOL. In other languages blank is not strictly treated as a character. But blanks have to be given wherever the grammer rules demand.

While writing, the blank may be indicated by putting the small case letter 'b'. While communicating with the computer the space bar is simply pressed once. For alphabetic characters some computers will also accept small case letters. But for the sake of uniformity we will use only capital letters for coding purposes.

Summary

In this article, the evolution of computer languages leading to the development of high level languages has been discussed. Every language has a character set and a grammer. So also are all computer languages. The character sets used in FORTRAN, COBOL, BASIC, PL/1 and PASCAL have also been described.

It has already been pointed out that the rules of the computer languages have become slightly implementor dependent. So some of the characters given in this article may not be available in all machines and there may also be some slight modifications for the symbols. One mustlalways refer to the manual of the particular machine for the exact list of character set.

(To be continued)