We to learn. however, many users dislike

We assembly languages, which represent these detailed machine instructions in a code that is easier to understand than strings of 0s and Is, are still frequently used by some programmers, but they are beyond the grasp of most end users. Machine language flourished during the first generation (1951-1958) and assembly languages during the second generation (1959-1964) of modern computing history. The next languages to appear, which generally characterized the third generation of modern computing history (1965-1970) and are still very widely used today, were high-level languages. Included in this category are what have come to be known as the “ traditional” types of programming languages, for example, BASIC, COBOL, FORTRAN, Pascal, PL/1, and APL. Well over 100 such languages are currently in use, and many of them are available in several versions.

High-level languages differ from their low- level predecessors in that they require less coding detail. For example, low- level predecessors in that they require less coding detail. For example, low- level languages require programmers to assign values to specific storage locations and registers in the CPU.

High-level languages, however, don’t burden programmers with these tasks; the translators that convert high-level languages to machine language supply the detail. As a result, programs created in high-level languages are shorter and easier to write than those written in their low-level counterparts. Some high-level languages, such as BASIC, are relatively easy for even users to learn. However, many users dislike programming in any high-level language whatsoever. Some feel there are too many rules to remember and

the step-by-step logic involved is too complex. Others simply are too busy to do the volume of programming these languages require. Yet many of these users could benefit by writing their own programs if this task were somehow made easier.

Very-high-level languages- many of which became widely used during the fourth generation of modern computing history (starting about 1971)- have been developed to meet this need. We can compare a very-high-level language to a knowledgeable chauffeur. To get where we want to go, we need only give the chauffeur very general instructions, such as “ Take me to City Hall,” instead of detailed ones, like “ From here you make a right; then three blocks later make a left…”. Similarly, with very-high-level languages we need only prescribe what the computer is to do rather than how it is to do it.

For example, we might sum numbers with a very-high-level language by pointing on the screen to the instruction “ SUM” and them pointing to the numbers to be summed. This is much easier to do than creating a mathematical looping procedure to “ teach” the computer system how to sum numbers and then typing in and debugging that procedure, as would be required with a log-level or high- level language. Thus, low and high level language are sometimes called procedural languages, because they require people to define detailed statements that represent sequential steps, or procedures, to be performed during program execution. Very-high-level languages, in contrast, are called nonprocedural languages.

Very-high-level languages, along with advances in interactive display technology, are bringing more people into contact with computers than ever before. These languages have some serious disadvantages, however. First, they lack flexibility; generally each is designed to do one specific kind of task. You can’t, for example, process a payroll with a word processing language.

Second, these languages do not adequately support many important applications., nonetheless, for the areas in which they are available and suitable, these languages offer obvious advantages for both programmers and users.