ASSEMBLY LANGUAGE

Machine instructions are represented by patterns of 0s and 1s. Such patterns are awkward to deal with when discussing or preparing programs. Therefore, we use symbolic names to represent the pattern. So far, we have used normal words, such as Move, Add, Increment, and Branch, for the instruction operations to represent the corresponding binary code patterns. When writing programs for a specific computer, such words are normally replaced by acronyms called mnemonics, such as MOV, ADD, INC, and BR. Similarly, we use the notation R3 to refer to register 3, and LOC to refer to a memory location. A complete set of such symbolic names and rules for their use constitute a programming language, generally referred to as an assembly language.

Programs written in an assembly language can be automatically translated into a sequence of machine instructions by a program called an assembler. When the assembler program is executed, it reads the user program, analyzes it, and then generates the desired machine language program. The latter contains patterns of 0s and 1s specifying instructions that will be executed by the computer. The user program in its original alphanumeric text format is called a source program, and the assembled machine language program is called an object program.

ASSEMBLER DIRECTIVES:-

In addition to providing a mechanism for representing instructions in a program, the assembly language allows the programmer to specify other information needed to translate the source program into the object program. We have already mentioned that we need to assign numerical values to any names used in a program. Suppose that the name SUM is used to represent the value 200. This fact may be conveyed to the assembler program through a statement such as

```
SUM EQU 200
```

This statement does not denote an instruction that will be executed when the object program is run; in fact, it will not even appear in the object program. It simply informs the assembler that the name SUM should be replaced by the value 200 wherever it appears in the program. Such statements, called assembler directives (or commands), are used by the assembler while it translates a source program into an object program.

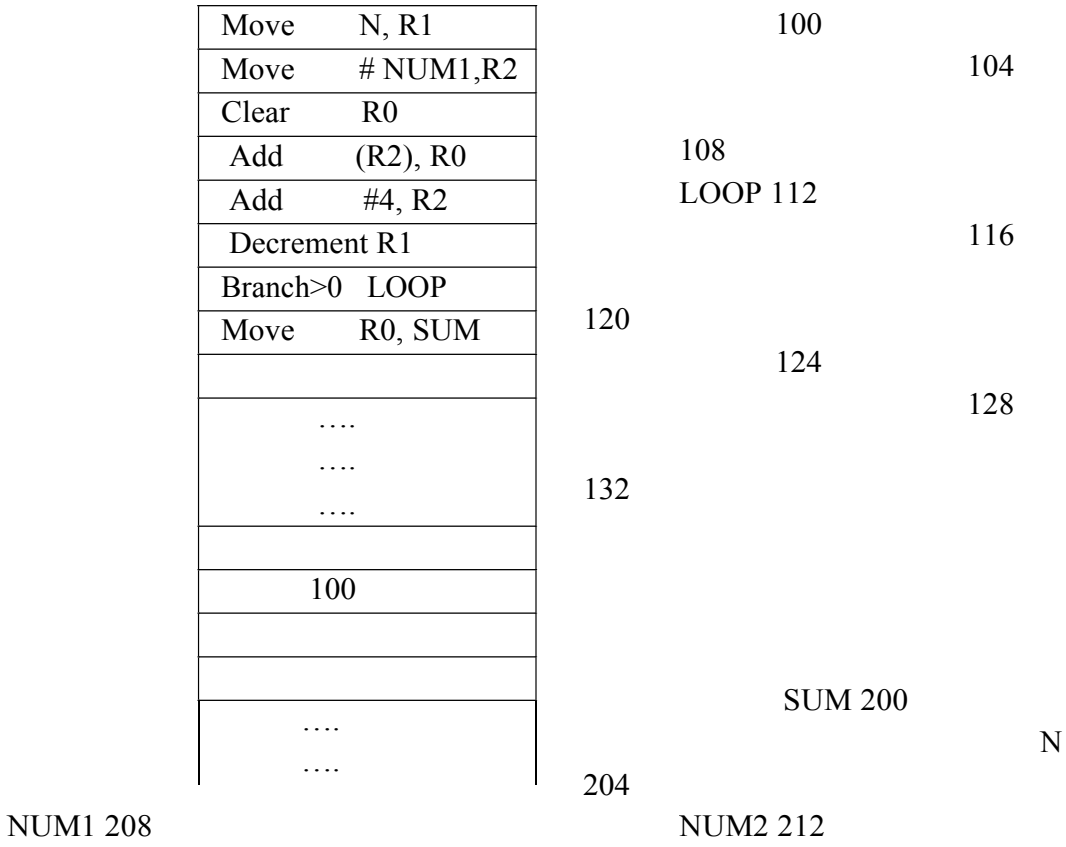

Fig 2.17 Memory arrangement for the program in fig b.

ASSEMBLY AND EXECUTION OF PRGRAMS:-

A source program written in an assembly language must be assembled into a machine language object program before it can be executed. This is done by the assembler program, which replaces all symbols denoting operations and addressing modes with the binary codes used in machine instructions, and replaces all names and labels with their actual values.

The assembler assigns addresses to instructions and data blocks, starting at the address given in the ORIGIN assembler directives. It also inserts constants that may be given in DATAWORD commands and reserves memory space as requested by RESERVE commands.

As the assembler scans through a source programs, it keeps track of all names and the numerical values that correspond to them in a symbol table. Thus, when a name appears a second time, it is replaced with its value from the table. A problem arises when

a name appears as an operand before it is given a value. For example, this happens if a forward branch is required. A simple solution to this problem is to have the assembler scan through the source program twice. During the first pass, it creates a complete symbol table. At the end of this pass, all names will have been assigned numerical values. The assembler then goes through the source program a second time and substitutes values for all names from the symbol table. Such an assembler is called a two-pass assembler.

The assembler stores the object program on a magnetic disk. The object program must be loaded into the memory of the computer before it is executed. For this to happen, another utility program called a loader must already be in the memory.

When the object program begins executing, it proceeds to completion unless there are logical errors in the program. The user must be able to find errors easily. The assembler can detect and report syntax errors. To help the user find other programming errors, the system software usually includes a debugger program. This program enables the user to stop execution of the object program at some points of interest and to examine the contents of various processor registers and memory locations.

NUMBER NOTATION:-

When dealing with numerical values, it is often convenient to use the familiar decimal notation. Of course, these values are stored in the computer as binary numbers. In some situations, it is more convenient to specify the binary patterns directly. Most assemblers allow numerical values to be specified in different ways, using conventions that are defined by the assembly language syntax. Consider, for example, the number 93, which is represented by the 8-bit binary number 01011101. If this value is to be used an immediate operand, it can be given as a decimal number, as in the instructions.

```
ADD   #93, R1
```

Or as a binary number identified by a prefix symbol such as a percent sign, as in

```
ADD   #%01011101, R1
```

Binary numbers can be written more compactly as hexadecimal, or hex, numbers, in which four bits are represented by a single hex digit. In hexadecimal representation, the decimal value 93 becomes 5D. In assembly language, a hex representation is often identified by a dollar sign prefix. Thus, we would write

```
ADD   #$5D, R1
```