SNS COLLEGE OF TECHNOLOGY

Coimbatore-35.

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

COURSE NAME : 19ITT202 – COMPUTER ORGANIZATION AND ARCHITECTURE

II YEAR/ III SEMESTER

UNIT – I Basic Structure of Computers

Topic: Assembly Language

Mrs. M. Lavanya
Assistant Professor
Department of Computer Science and Engineering

Assembly Language

- Machine instructions are represented by patterns of 0s and 1s. Therefore, we use symbolic names to represent patterns in the program.
- So far, we have used normal words, such as Move, Add, Increment and Branch for 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 R3 referred to Register 3 & LOC referred to 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***.
- An assembly language is **a type of low-level programming language that is intended to communicate directly with a computer's hardware**. Unlike machine language, which consists of binary and hexadecimal characters, assembly languages are designed to be readable by humans.
- The set of rules for using the mnemonics in the specification of complete instructions and programs is called the ***syntax*** of the language.

- Programs written in an assembly language can be automatically translated into a sequence of machine instructions by a program called an ***assembler***.
- The assemble program is one of a collection of utility programs that are a part of the system software. The assembler stores sequence of machine instructions in the computer memory.

  → A user program (Set of alphanumeric characters) entered into memory through keyboard.

  → When the assembler program is executed, it reads the user program, analyses it & then generates the desired machine language program (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.***
- The assembled machine language program is called an ***object Program***.

Example- **MOVE R0,SUM**

ADD #5,R3

Add instruction may be written as **ADDI 5,R3**

The suffix I in the mnemonic ADDI states that the source operand is given in the Immediate addressing mode.

In Indirect addressing if No. 5 is to be placed in memory location whose address is held in register R2. Can be Specified as **MOVE #5,(R2)** or **MOVEI 5,(R2)**

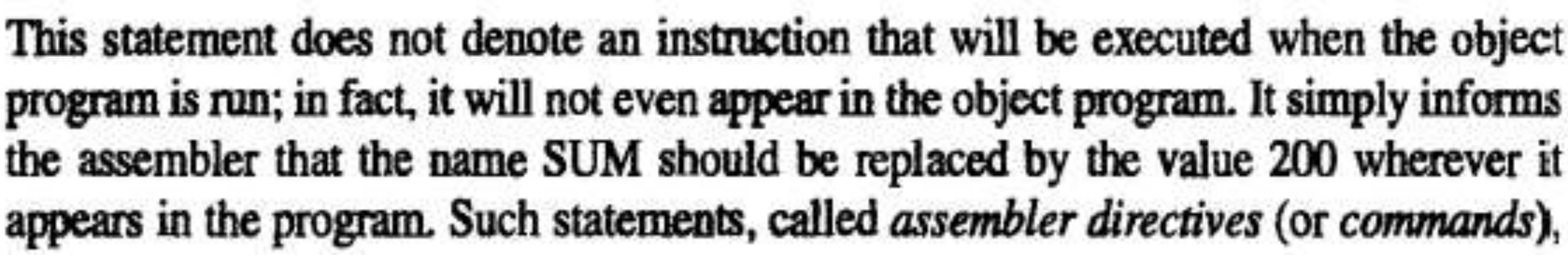
Assembler Directives

- The assembly language allows the programmer to specify other information needed to translate the source program into the object program.

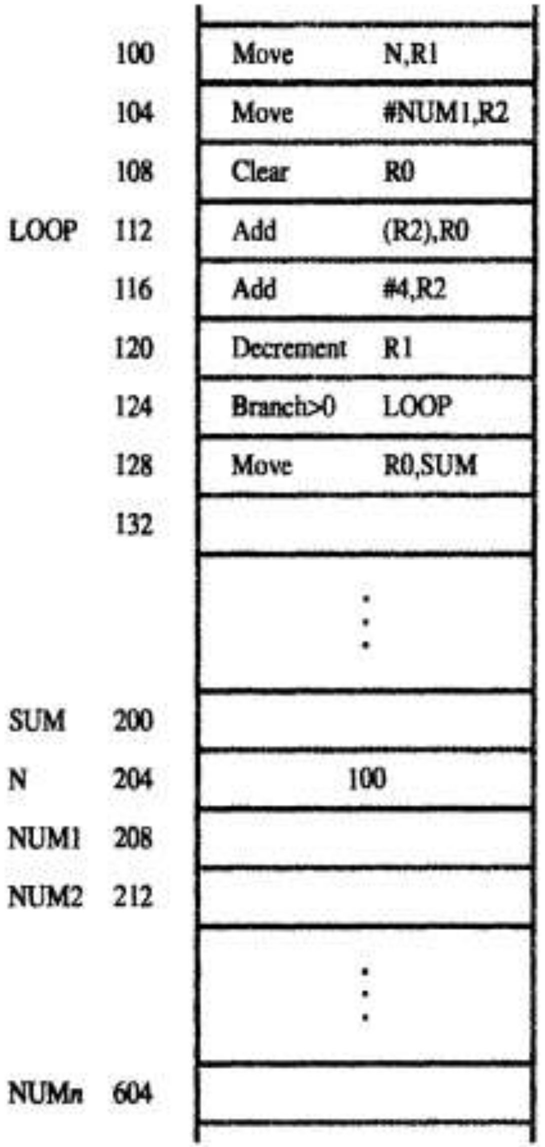
- Assembler Directives are **instructions used by the assembler while it translates a source program into an object program,** by helping to automate the assembly process and to improve program readability

Suppose that the name SUM is used to represent the value 200.

Assembly Program statement is **SUM EQU 200**

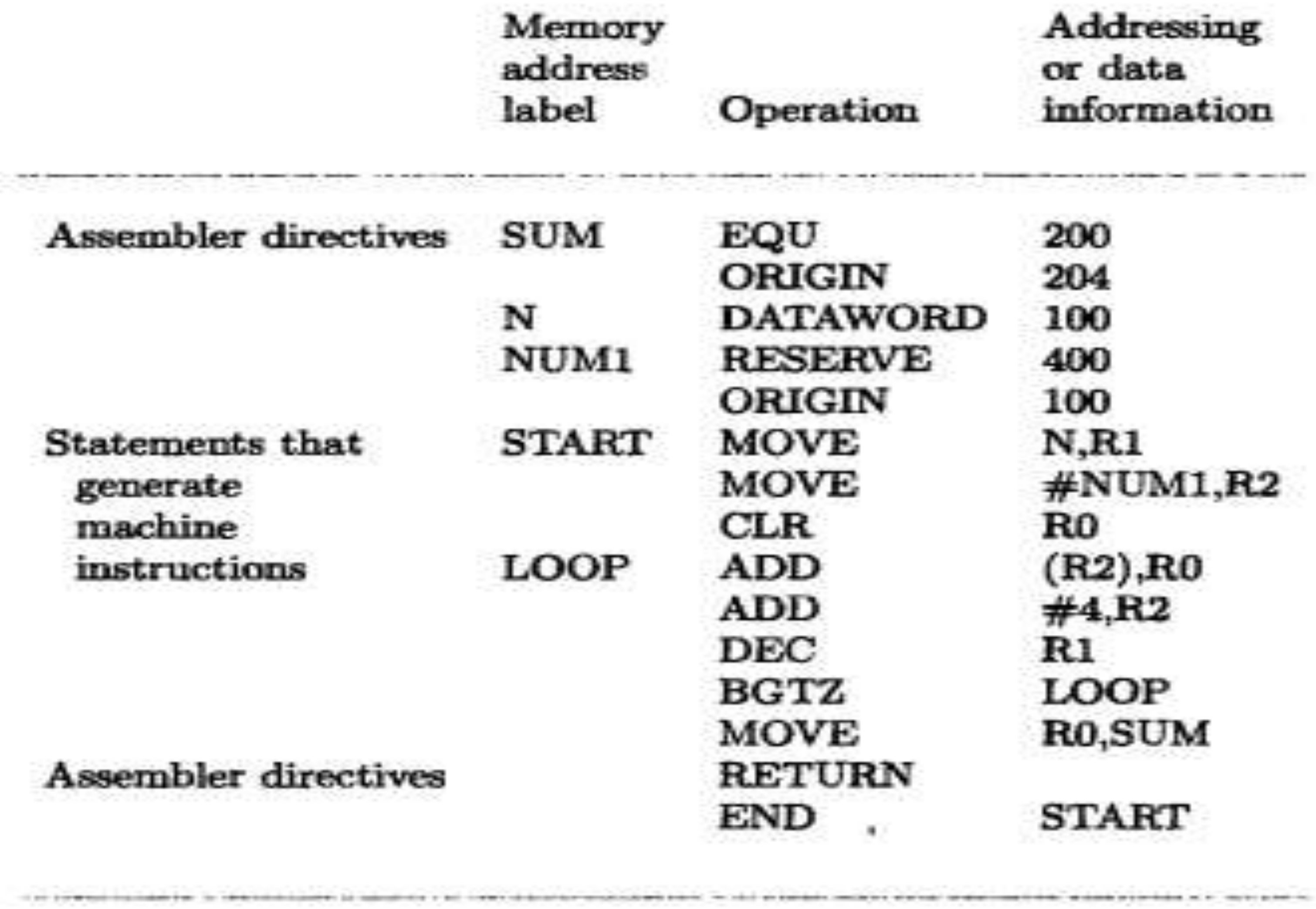


This statement does not denote an instruction that will be executed when the object program is run; in fact, it will not even appear in the object program. It simply informs the assembler that the name SUM should be replaced by the value 200 wherever it appears in the program. Such statements, called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



	100	Move	N,R1
	104	Move	#NUM1,R2
	108	Clear	R0
LOOP	112	Add	(R2),R0
	116	Add	#4,R2
	120	Decrement	R1
	124	Branch>0	LOOP
	128	Move	R0,SUM
	132		
		⋮	
SUM	200		
N	204	100	
NUM1	208		
NUM2	212		
		⋮	
NUM*n*	604		

Figure 2.17 Memory arrangement for the program in Figure 2.12.



	Memory address label	Operation	Addressing or data information
Assembler directives	SUM	EQU	200
		ORIGIN	204
	N	DATAWORD	100
	NUM1	RESERVE	400
		ORIGIN	100
Statements that generate machine instructions	START	MOVE	N,R1
		MOVE	#NUM1,R2
		CLR	R0
	LOOP	ADD	(R2),R0
		ADD	#4,R2
		DEC	R1
		BGTZ	LOOP
		MOVE	R0,SUM
Assembler directives		RETURN	
		END	START

Figure 2.18 Assembly language representation for the program in Figure 2.17.

- Equate Directive (EQU) → Informs the assembler about the value of SUM
- ORIGIN → Tells the assembler program, where in the memory to place the data block that follows.
- DATAWORD Directive → used to inform the assembler about the value to be loaded in memory location address.
- RESERVE Directive → Declares the memory block to be reserved for data.
- END Directive → Tells the assembler about the end of source program text.
- RETURN Directive → Identifies the point at which execution of the program should be Terminated.

- Most assembly language require statements in a source program to be written in the form

Label Operation Operand(s) Comment

Label → Optional name associated with memory address (SUM, N, NUM1, START & LOOP)
Operation → Op-code mnemonic
Operand(s) → Information of one or more operands
Comment → Make the program easier to understand

- These 4 fields are separated by one or more blank characters.

Assembly and Execution of Programs

Symbol Table

As assembler scans the source program, it keeps track of all names and its corresponding numerical values in a symbol table.

Two pass Assembler-

The assembler scans (goes through) the source program 2 times.

During 1st pass, it creates symbol table names and assign values & during 2nd pass, it substitutes values for all names in symbol table.

In the execution of assembler program, the **Loader** begins execution unless an logical error or syntax error appears in the program.

To help the user find other programming errors, the system software usually includes a **debugger program.**

Number Notation

When dealing with numerical values

- Decimal Number – **ADD #93, R1**

- Binary Number – **ADD #%01011101, R1**

- Hexadecimal Number – **ADD #S5D, R1**

THANK
YOU!