BACHELOR OF COMPUTER APPLICATIONS (B.C.A)
SEMESTER – III
COMPUTER ARCHITECTURE & ASSEMBLY LANGUAGE

Mr. Vijay Prakash Mishra

Assistant Professor

Department Of Computer Application

Jagatpur P. G. College, Varanasi

(Affiliated To Mahatma Gandhi Kashi Vidyapeeth, Varanasi)

Email- vijayprakashmishra1971@gmail.Com

UNIT - VI

ASSEMBLY LANGUAGE

- Each personal computer has a microprocessor that manages the computer's arithmetical, logical, and control activities.
- Each family of processors has its own set of instructions for handling various operations such as getting input from keyboard, displaying information on screen and performing various other jobs. These set of instructions are called 'machine language instructions'.
- A processor understands only machine language instructions, which are strings of 1's and 0's. However, machine language is too obscure and complex for using in software development. So, the low-level assembly language is designed for a specific family of processors that represents various instructions in symbolic code and a more understandable form.

ADVANTAGES OF ASSEMBLY LANGUAGE

- Having an understanding of assembly language makes one aware of –
  - How programs interface with OS, processor, and BIOS;
  - How data is represented in memory and other external devices;
  - How the processor accesses and executes instruction;
  - How instructions access and process data;
  - How a program accesses external devices.
- Other advantages of using assembly language are –
  - It requires less memory and execution time;
  - It allows hardware-specific complex jobs in an easier way;
  - It is suitable for time-critical jobs;
  - It is most suitable for writing interrupt service routines and other memory resident programs.

ASSEMBLER

- An assembler is a program that converts assembly language into machine code. It takes the basic commands and operations from assembly code and converts them into binary code that can be recognized by a specific type of processor.
- Assemblers are similar to compilers in that they produce executable code. However, assemblers are more simplistic since they only convert low-level code (assembly language) to machine code.
- Most programs are written in high-level programming languages and are compiled directly to machine code using a compiler. However, in some cases, assembly code may be used to customize functions and ensure they perform in a specific way.

ASSEMBLY LEVEL INSTRUCTION

- The executable instructions or simply instructions tell the processor what to do. Each instruction consists of an operation code (opcode). Each executable instruction generates one machine language instruction.

Instruction	Format
AND	AND operand1, operand2
OR	OR operand1, operand2
XOR	XOR operand1, operand2
TEST	TEST operand1, operand2
NOT	NOT operand1

MACRO

- Writing a macro is another way of ensuring modular programming in assembly language.
- A macro is a sequence of instructions, assigned by a name and could be used anywhere in the program.
- In NASM, macros are defined with **%macro** and **%endmacro** directives.
- The macro begins with the %macro directive and ends with the %endmacro directive.

SYNTAX FOR MACRO

```
%macro macro_name  number_of_params
<macro body>
%endmacro
```

PROGRAM LOOPS

- The JMP instruction can be used for implementing loops. For example, the following code snippet can be used for executing the loop-body 10 times

```
MOV     CL, 10
L1:
<LOOP-BODY>
DEC     CL
JNZ     L1
```

- The processor instruction set, however, includes a group of loop instructions for implementing iteration. The basic LOOP instruction has the following syntax :

```
LOOP    label
```

PROGRAM LOOPS(CONTD.)

```
LOOP    label
```

- Where, *label* is the target label that identifies the target instruction as in the jump instructions. The LOOP instruction assumes that the ECX register contains the loop count. When the loop instruction is executed, the ECX register is decremented and the control jumps to the target label, until the ECX register value, i.e., the counter reaches the value zero.

THE INC & DEC INSTRUCTION

- The INC instruction is used for incrementing an operand by one. It works on a single operand that can be either in a register or in memory.
- The INC instruction has the following syntax – INC Destination
- The DEC instruction is used for decrementing an operand by one. It works on a single operand that can be either in a register or in memory.
- The DEC instruction has the following syntax – DEC Destination

THE ADD AND SUB INSTRUCTIONS

- The ADD and SUB instructions are used for performing simple addition/subtraction of binary data in byte, word and doubleword size, i.e., for adding or subtracting 8-bit, 16-bit or 32-bit operands, respectively.
- The ADD and SUB instructions have the following syntax – ADD/SUB destination, source
- The ADD/SUB instruction can take place between –
  - Register to register
  - Memory to register
  - Register to memory
  - Register to constant data
  - Memory to constant data
- However, like other instructions, memory-to-memory operations are not possible using ADD/SUB instructions. An ADD or SUB operation sets or clears the overflow and carry flags.

INPUT-OUTPUT PROGRAMMING

- **Input/Output** (I/O) instructions are used to input data from peripherals, output data to peripherals, or read/write input/output controls. Early computers used special hardware to handle I/O devices. The trend in modern computers is to map I/O devices in memory, allowing the direct use of any instruction that operates on memory for handling I/O.
- **IN** Input; MIX; initiate transfer of information from the input device specified into consecutive locations starting with M, block size implied by unit
- **OUT** Output; MIX; initiate transfer of information from consecutive locations starting with M to the output device specified, block size implied by unit
- **IOC** Input-Output Control; MIX; initiate I/O control operation to be performed by designated device
- **JRED** Jump Ready; MIX; Jump if specified unit is ready (completed previous IN, OUT, or IOC operation); if jump occurs, J-register loaded with the address of the instruction which would have been next if the jump had not been taken
- **JBUS** Jump Busy; MIX; Jump if specified unit is not ready (not yet completed previous IN, OUT, or IOC operation); if jump occurs, J-register loaded with the address of the instruction which would have been next if the jump had not been taken

EXCERCISE

- Write A few Basic Assembly Level Codes for the following:
  - Take two inputs A and B, add them
  - Use the same inputs for multiplication, subtraction and division
  - Write the HELLO WORLD Program in Assembly Language.
- Write Short Notes:
  - Assembly Language
  - Logical Instructions
  - Arithmetic Instructions
  - Looping Construc

DECLARATION

"The content is exclusively meant for academic purpose and for enhancing teaching and learning. Any other use for economic/commercial purpose is strictly prohibited. The users of the content shall not distribute, disseminate or share it with anyone else and its use is restricted to advancement of individual knowledge. The information provided in this e-content is authentic and best as per knowledge".

Vijay Prakash Mishra

Assistant Professor

Department of Computer Application

Jagatpur P. G. College, Varanasi

THANK YOU!!!