**COMPUTER ORGANIZATION**

**ASSEMBLY LANGUAGE**

At first let us discuss about the programming language and where we are using those languages.

A programming language is a formal language comprising a set of instructions used to produce various kinds of output. It allows humans to communicate instructions to computers in a manner that they can understand and execute. Programming languages consist of syntax rules and semantics that determine how instructions are structured and interpreted. These instructions can be used to create software applications, websites, algorithms, and more. Programming languages vary widely in their complexity, purpose and application domain and new languages contribute to emerge as technology evolves.

And now let us discuss the assembly language.

Assembly language is a low-level programming language that provides a way to communicate directly with a computer's hardware. It uses mnemonic codes and symbols that represent specific machine instructions. It has direct interaction with the hardware of the system. Unlike high-level programming languages, assembly language offers minimal abstraction from the underlying hardware. Programmers must have a deep understanding of the system architecture and the behavior of individual instructions.

Let us see how we can write the code in assembly language,

At first we should know about the mnemonic in assembly language. Then what is mnemonics? Let us see now,

A mnemonic is a symbolic representation of an instruction or operation. Mnemonics are designed to be more human-readable and easier to remember than the binary machine code instructions they represent. Each mnemonic corresponds to a specific machine instruction or operation that the processor can execute.

Example:

Let us consider the instruction,

ADD R3,R2,R1

Here ADD is the mnemonic which is used to add two registers and store in the third register.

R1,R2,R3 are registers.

Now let us see few more mnemonics,

|  |  |  |  |
| --- | --- | --- | --- |
| S.No | Mnemonic | | Uses |
| 1 | add | Used to add two values | |
| 2 | addi | Add the immediate value | |
| 3 | sub | Used to sub two values | |
| 4 | mul | Used to multiply two values | |
| 5 | div | Used to divide two values | |
| 6 | rem | Used to store the reminder of two values | |
| 7 | and | Used to perform and operation | |
| S.No | Mnemonics | Uses | |
| 8 | or | Used to perform or operation | |
| 9 | beq | Used to compare two values | |
| 10 | bge | Used to check which is greater value | |
| 11 | lw | Load the word | |
| 12 | sw | Stores the word | |

Let us see few examples using these mnemonics in the instruction.

ADD R2,R3,#10

In this instruction adds the number 5 to the contents of register R3 and puts the result into register R2. The number sign is not the only way to denote the Immediate addressing mode. In some assembly languages, the Immediate addressing mode is indicated in the OP code mnemonic. For example, the previous Add instruction may be written as,

ADDI R2,R3,10

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

Let us see the simple mips program to sum the first 10 natural numbers.

li $t0, 0

li $t1, 1

loop:

add $t0, $t0, $t1

addi $t1,$t1,1

bgt $t1, 10, exit

j loop

exit:

sw $t0, sum

Here’s Explanation for the above code

li $t0, 0: Initializes register $t0 to 0, which will hold the sum.

li $t1, 1: Initializes register $t1 to 1, which will be the loop counter.

loop label: Marks the beginning of the loop.

add $t0, $t0, $t1: Adds the loop counter to the sum.

addi $t1, $t1, 1: Increments the loop counter.

bgt $t1, 10, exit: Branches to exit if the loop counter is greater than 10.

j loop: Jumps back to the loop label to repeat the loop.

exit label: Marks the end of the program.

sw $t0, sum: Stores the final sum in the sum variable.

**Real world Applications using Assembly language**

Operating Systems

Assembly language is often used in the development of operating systems, as it provides low-level access to the system's hardware and is capable of executing critical operations that are not possible with higher-level languages.

Device Drivers

Assembly language is used to develop device drivers, which are software components that allow a computer to interact with specific hardware devices.

Embedded Systems

Assembly language is widely used in the development of embedded systems, such as microcontrollers and other specialized devices that have limited processing power and memory.

Performance-critical applications

Assembly language can be used to optimize performance-critical sections of code in high-performance applications, such as video games, simulations, and scientific computing.

Reverse Engineering

Assembly language is often used in reverse engineering, as it provides a detailed understanding of the underlying machine code and can be used to analyze malware, debug software, and study the internal workings of operating systems and other software.

**Current Trends in the Assembly Language Industry**

With cybersecurity threats increasing, there's an increased emphasis on using assembly language for security assessment and developing more secure firmware and low-level software components.

The Internet of Things (IoT) sector drives demand for assembly language expertise to develop highly efficient and optimized firmware for microcontrollers and embedded devices.

A specialized but growing interest in retro computing and game development for vintage systems is leading to renewed interest in assembly language programming for these platforms.

There's a trend towards integrating assembly language blocks within high-level language code (e.g., C or C++) for crucial performance sections, merging efficiency with developer productivity.