Interpreter: written in C language to interpret assembly language based on some basic instructions for a machine having three registers.

***Abstract* – This paper proposes an approach to interpret the assembly language. The given set of assembly code instructions in the database will be converted for a machine having three registers, which is an accumulator. To accomplish our project we have used C language and used the two ways of completing a mathematical or normal task having three register CPU.**

***Keywords –* accumulator, assembly code, data transfer operation, ALU operation, instruction, interpreter.**

1. INTRODUCTION

In this type of CPU organization, the accumulator register is used implicitly for processing all instructions of a program and store the results into the accumulator. The instruction format that is used by this CPU Organization is one address field. Due to this the CPU is known as One Address Machine. In this CPU Organization, the first ALU operand is always stored into the Accumulator and the second operand is present in the Memory. Accumulator is the default address thus after data manipulation the results are stored into the accumulator. One address instruction is used in this type of organization. The format of instruction is: Opcode + Address. Opcode indicates the type of operation to be performed. Mainly two types of operation are performed in single accumulator based CPU organization. Data transfer operation - In this type of operation, the data is transferred from a source to a destination. For ex: LOAD X, STORE Y. Here LOAD memory read operation that is data is transfer from memory to accumulator and STORE is memory write operation that is data is transferred from accumulator to memory. ALU operation – In this type of operation, arithmetic operations are performed on the data. For ex: MUL X. Where X is the address of the operand. The MUL instruction in this example performs the operation, AC 🡨 AC \* M[X]. AC is the Accumulator and M[X] is the memory word located at location X. This type of organization is first used in PDP-8 processor and is used for process control and laboratory applications. It has been totally replaced by the introduction of the new general register based CPU.

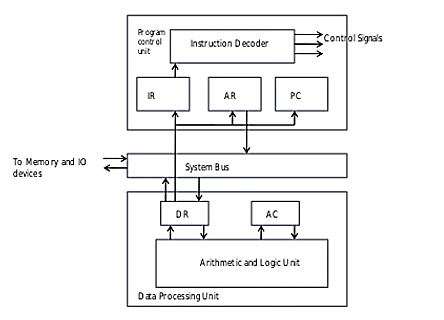
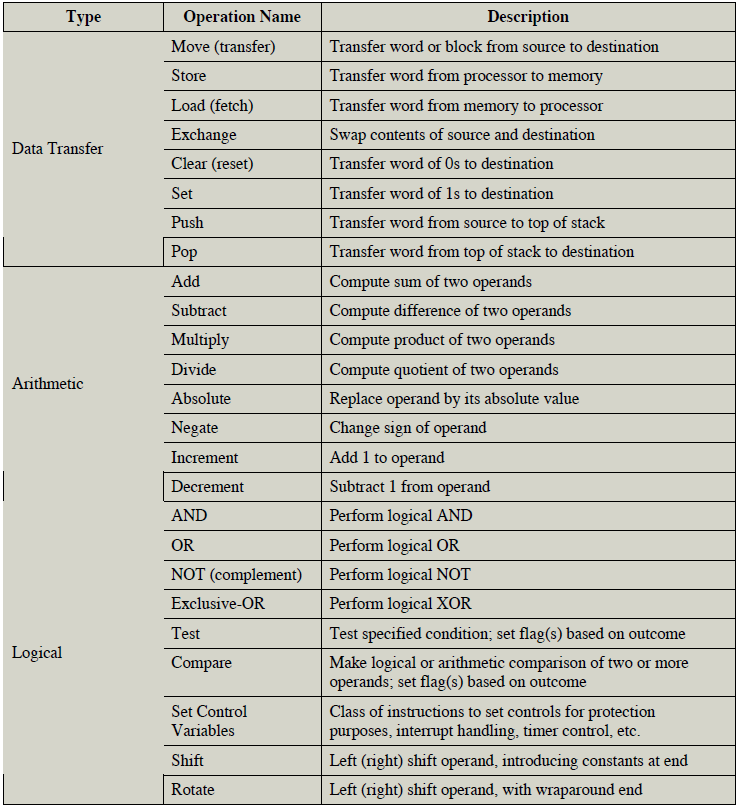
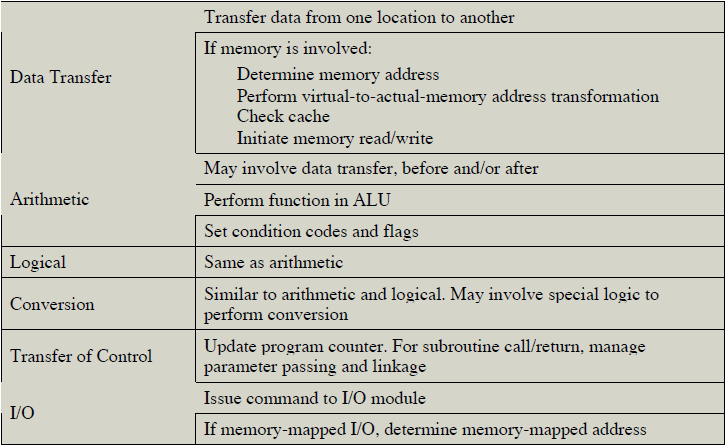


Diagram: Single Accumulator based CPU organization

1. BACKGROUND STUDY

From the studies of assembly language we have found out that there are a lot built-in functions or operations. Among them some are displayed below:





1. PROPOSED WORK

We wanted to create a program where the main purpose is to interpret the assembly code instruction for the ease of working with assembly language. As, in our daily life we do not use assembly language normally but the purpose of using the machine we have to learn the assembly language. We used C language to accomplish our target of interpreting the language in which we will be taking input the assembly codes and work in the background on that process to complete the task. Previously, the introduction and background study was put for the better understanding of our project.

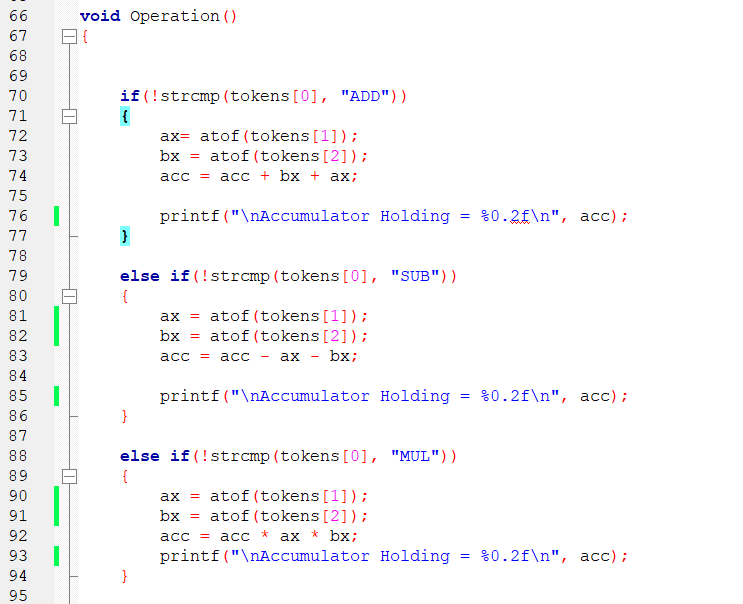
1. EXPERIMENTAL ANALYSIS & RESULT

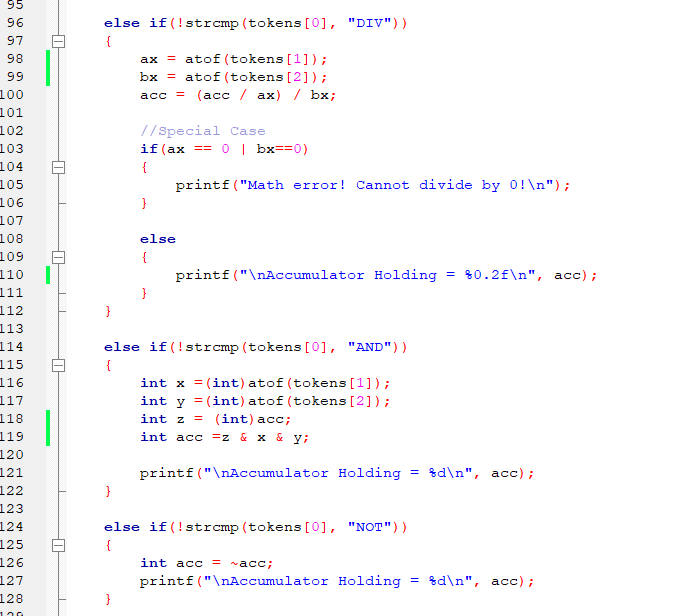
For the better understanding of our project, we will be discussing the code. We are putting some of our project demonstration for the full proof bug testing and the way of running the project.

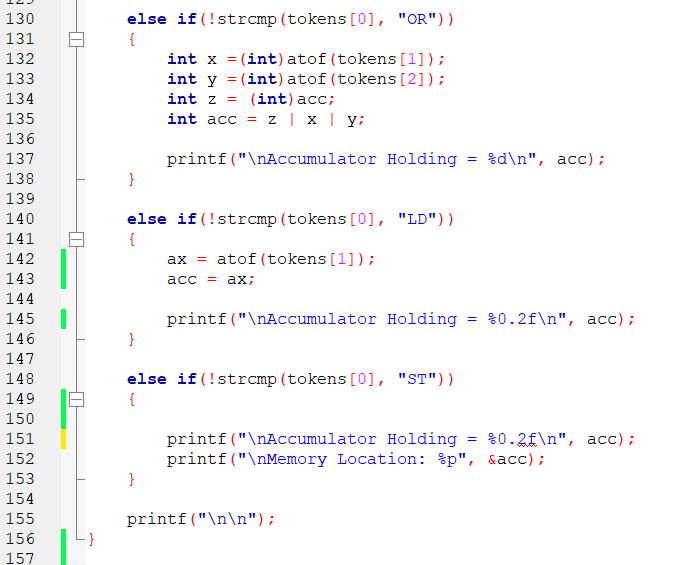
The **Input**() module takes input from the user in the form of assembly code then separates and stores the tokens in an array.



The Operation() module takes the assembly code commands (tokens) stored in the array and performs the instructed operations and shows output.







We have added ONE feature into our project which is calculating fractional values with ADD, SUB, MUL, DIV, LD, ST.

**Input (assembly code)** LD, ADD, SUB, MUL, DIV, ST

LD 45.78

ADD 56.89,34.98

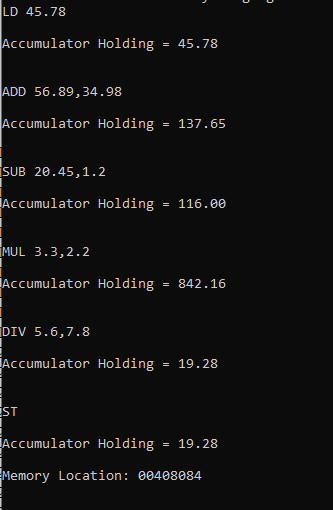
SUB 20.45,1.2

MUL 3.3,2.2

DIV 5.6,7.8

ST

**Output**

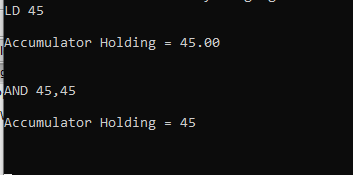


**Input (assembly code)** Bitwise AND

LD 45

AND 45,45

**Output**

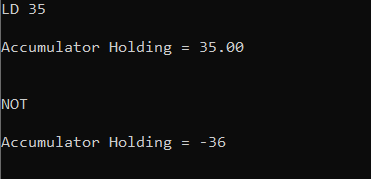


**Input (assembly code)** Bitwise NOT

LD 35

NOT

**Output**



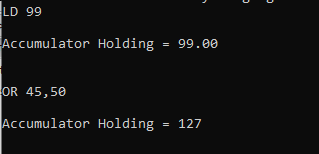
Bitwise complement of any number N is – (N+1). [Bitwise complement of N = ~N (represented in 2’s complement form) and 2’s complement of ~N = - (~ (~N) +1) = - (N+1).]

**Input (assembly code)** Bitwise OR

LD 99

OR 45,50

**Output**



1. CONCLUSION

In our project we tried our best to solve all the bugs that occurred during designing this code. Hence in conclusion, we were successful to make the code fully functional without much of an error. In future we would work on the storage to became temporary. In terms of future improvement we will recommend exhaustive bug testing and bug fixes also with some other addition of functions we can use the code comfortably without writing long codes.

1. REFERENCES
2. William Stallings, Computer Organization and Architecture: Designing for Performance (Eighth Edition).
3. Sudipta Dandapat, Introduction of single Accumulator based CPU organization, <https://www.geeksforgeeks.org/introduction-of-single-accumulator-based-cpu-organization/>
4. Programiz, Bitwise Operators in C Programming, https://www.programiz.com/c-programming/bitwise-operators?fbclid=IwAR16Co\_yvbJhSUTjd6BANOKXGE4MG0Cw\_ob2SBhwysq8muqzvmWovNGWVM