**Topic: Construct an Interpreter for Multiple Accumulators CPU Organization.**

**Course:** CSE360

**Section:** 03

**Prepared By –**

**Group No –** 10

**Group members -**

**Shafia Hasnin**

Department of computer science and engineering East West University

Dhaka, Bangladesh

Email: [2020-1-60-209@std.ewubd.edu](mailto:2020-1-60-201@std.ewubd.edu)

**Samiu Esika Upoma**

Department of computer science and engineering East West University

Dhaka, Bangladesh

Email: [2020-1-60-082@std.ewubd.edu](mailto:2020-1-60-082@std.ewubd.edu)

**Zarin Tasnim Nuzhat**

Department of computer science and engineering East West University

Dhaka, Bangladesh

Email: [2020-1-60-211@std.ewubd.edu](mailto:2020-1-60-211@std.ewubd.edu)

**Submitted To -**

Md. Nawab Yousuf Ali

Department of computer science and engineering East-West University

Dhaka, Bangladesh

Email: nawab@ewubd.edu

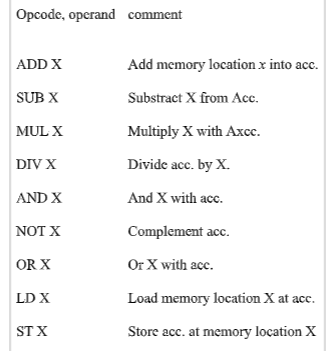
**Abstract:**

We are using multiple general-purpose registers in the CPU Organization. This type of organization is known as a General register-based CPU organization. The computer uses more than one address field in its instruction format. Each address field may specify a general register or a memory word.

**1. Introduction:**

**W**hat is Accumulator? The Accumulator is a register in which intermediate arithmetic logic unit (ALU) results are stored. Modern computer systems often have multiple general-purpose registers that can operate as accumulators, and the term is no longer as common as it once was. Create a translator written in C language that helps convert an assembly language based on a given set of basic instructions for a machine, which has multiple registers and accumulators with all the operands in memory.

**Objective:**

Our goal was to make an assembly language interpreted in C language that has the ability to follow instructions for a machine having multiple registers. There is some set of assembly code instructions in the database. 

**Design:**

The main features of multiple accumulators-based CPU organizations are:

1. The First ALU (Arithmetic-logic Unit) operand is always stored in the accumulators and the second operand is present in the memory.
2. Accumulator is the default address. After manipulating the data, the results are stored in the accumulators.
3. Five-address instruction is used, in this Organization.

In this type of CPU association, the accumulator register is used implicitly for recycling all instructions of a program and storing the results in the accumulator. The instruction format that is used by this CPU association is a ‘five address field’. Due to this, the CPU is known as the “**Five Address Machine**.”

The format of instruction is Opcode + Address

Opcode showed what type of operation should be performed.

There are two types of operations are performed in multiple accumulators-based CPU organization:

1. **Data Transfer Operation –**

In this type of operation, the data is transferred from a source to a destination.

For example: LOAD X(LD X), STORE Y(ST Y)

Here LOAD is a **memory read** operation that is data is transferred from memory to the accumulator and STORE is a **memory write** operation that is data is transferred from the accumulator to memory.

2. **ALU (Arithmetic-Logic Unit) Operation –**

In this type of operation, arithmetic operations are performed on the data.

For example: MUL X

Here 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 CPU organization is first used in **PDP-8 processors** and is used for process control and laboratory applications. It has been totally replaced by the introduction of the new general register-based CPU.

**Advantages:**

1. The advantage of the Five-address format is that it results in short programs when evaluating arithmetic expressions.
2. We can avoid memory references much of the time.
3. Vastly increasing program execution speed.

**Disadvantages:**

1. The disadvantage is that the binary-coded instructions require too many bits to specify five addresses.
2. The program will not run much faster because now instructions only contain further information but each micro-operation (Changing the content of the register, lading address in the address machine, etc.) will be performed in one cycle only.

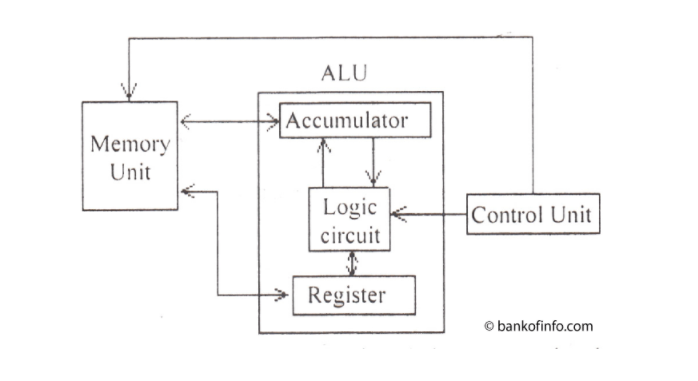


Figure: Multiple Accumulators ALU

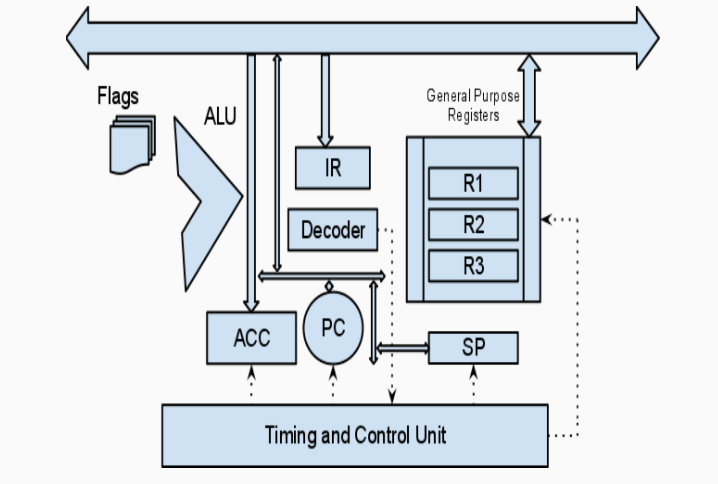


Figure: Multiple Accumulators-based CPU Organization

**Implementation:**

As instructed we used C language to design an interpreter and the implementation of the source code is described below-

Two important modules of the code are

1. getValues();
2. ParseCommand();

**I) get Values();** takes input from the user in the form of assembly code, separates and stores the tokens in an array.

**II) Parse Command()**; takes the assembly code commands (tokens) stored in the array performs the instructed operations and shows the output.

**getValues():**

void getValues()

{

fgets(input, sizeof(input), stdin);

int len = strlen(input);

input[len-1] = '\0';

int i;

int m = 0, n = 0;

for(i=0; input[i]!='\0'; i++)

{

check\_value[m][n] = '\0';

if(input[i] == ' ')

{

m++;

n = 0;

}

else

{

check\_value[m][n++] = input[i];

check\_value[m][n] = '\0';

}

}

}

**Function1():**

void Function1()

{

int x;

if(!strcmp(check\_value[0], "ADD1"))

{

x = atoi(check\_value[1]);

acc1 = acc1 + x;

printf("\naccumulator1 = %d\n", acc1);

}

else if(!strcmp(check\_value[0], "SUB1"))

{

x = atoi(check\_value[1]);

acc1 = acc1 - x;

printf("\naccumulator1 = %d\n", acc1);

}

else if(!strcmp(check\_value[0], "MUL1"))

{

x = atoi(check\_value[1]);

acc1 = acc1 \* x;

printf("\naccumulator1 = %d\n", acc1);

}

else if(!strcmp(check\_value[0], "DIV1"))

{

x = atoi(check\_value[1]);

acc1 = acc1 / x;

if(x == 0)

{

printf("Math error! Cannot divide by 0!\n");

}

else

{

printf("\naccumulator1 = %d\n", acc1);

}

}

else if(!strcmp(check\_value[0], "AND1"))

{

x = atoi(check\_value[1]);

acc1 = acc1 & x;

printf("\naccumulator1 = %d\n", acc1);

}

else if(!strcmp(check\_value[0], "NOT1"))

{

acc1 = ~acc1;

printf("\naccumulator1 = %d\n", acc1);

}

else if(!strcmp(check\_value[0], "OR1"))

{

x = atoi(check\_value[1]);

acc1 = acc1 | x;

printf("\naccumulator1 = %d\n", acc1);

}

else if(!strcmp(check\_value[0], "LD1"))

{

x = atoi(check\_value[1]);

acc1 = x;

printf("\naccumulator1 = %d\n", acc1);

}

else if(!strcmp(check\_value[0], "ST1"))

{

printf("\naccumulator1 = %d\n", acc1);

printf("\nstored in memory location: %p", &acc1);

}

printf("\n");

}

**Function2():**

void Function2() ///Accumulator 2

{

int x;

if(!strcmp(check\_value[0], "ADD2"))

{

x = atoi(check\_value[1]);

acc2 = acc2 + x;

//acc2 = acc1 + acc2;

printf("\naccumulator2 = %d\n", acc2);

}

else if(!strcmp(check\_value[0], "SUB2"))

{

x = atoi(check\_value[1]); ///Convert String to Intiger

acc2 = acc2 - x;

//acc2 = acc1 - acc2;

printf("\naccumulator2 = %d\n", acc2);

}

else if(!strcmp(check\_value[0], "MUL2"))

{

x = atoi(check\_value[1]); ///Convert String to Intiger

acc2 = acc2 \* x;

//acc2 = acc1 \* acc2;

printf("\naccumulator2 = %d\n", acc2);

}

else if(!strcmp(check\_value[0], "DIV2"))

{

x = atoi(check\_value[1]); ///Convert String to Intiger

acc2 = acc2 / x;

//acc2 = acc1 / acc2;

if(x == 0)

{

printf("Math error! Cannot divide by 0!\n");

}

else

{

printf("\naccumulator2 = %d\n", acc2);

}

}

else if(!strcmp(check\_value[0], "AND2"))

{

x = atoi(check\_value[1]); ///Convert String to Intiger

acc2 = acc2 & x;

//acc2 = acc1 & acc2;

printf("\naccumulator2 = %d\n", acc2);

}

else if(!strcmp(check\_value[0], "NOT2"))

{

acc2 = ~acc2;

printf("\naccumulator2 = %d\n", acc2);

}

else if(!strcmp(check\_value[0], "OR2"))

{

x = atoi(check\_value[1]); ///Convert String to Intiger

acc2 = acc2 | x;

//acc2 = acc1 | acc2;

printf("\naccumulator2 = %d\n", acc2);

}

else if(!strcmp(check\_value[0], "LD2"))

{

x = atoi(check\_value[1]); ///Convert String to Intiger

acc2 = x;

printf("\naccumulator2 = %d\n", acc2);

}

else if(!strcmp(check\_value[0], "ST2"))

{

printf("\naccumulator2 = %d\n", acc2);

printf("\nstored in memory location: %p", &acc2);

}

printf("\n");

}

**Function3():**

void Function3() ///Accumulator 3

{

int x;

if(!strcmp(check\_value[0], "ADD3"))

{

x = atoi(check\_value[1]); ///Convert String to Intiger

acc3 = acc3 + x;

printf("\naccumulator3 = %d\n", acc3);

}

else if(!strcmp(check\_value[0], "SUB3"))

{

x = atoi(check\_value[1]); ///Convert String to Intiger

acc3 = acc3 - x;

printf("\naccumulator3 = %d\n", acc3);

}

else if(!strcmp(check\_value[0], "MUL3"))

{

x = atoi(check\_value[1]); ///Convert String to Intiger

acc3 = acc3 \* x;

printf("\naccumulator3 = %d\n", acc3);

}

else if(!strcmp(check\_value[0], "DIV3"))

{

x = atoi(check\_value[1]); ///Convert String to Intiger

acc3 = acc3 / x;

if(x == 0)

{

printf("Math error! Cannot divide by 0!\n");

}

else

{

printf("\naccumulator3 = %d\n", acc3);

}

}

else if(!strcmp(check\_value[0], "AND3"))

{

x = atoi(check\_value[1]); ///Convert String to Intiger

acc3 = acc3 & x;

printf("\naccumulator1 = %d\n", acc3);

}

else if(!strcmp(check\_value[0], "NOT3"))

{

acc3 = ~acc3;

printf("\naccumulator3 = %d\n", acc3);

}

else if(!strcmp(check\_value[0], "OR3"))

{

x = atoi(check\_value[1]); ///Convert String to Intiger

acc3 = acc3 | x;

printf("\naccumulator3 = %d\n", acc3);

}

else if(!strcmp(check\_value[0], "LD3"))

{

x = atoi(check\_value[1]); ///Convert String to Intiger

acc3 = x;

printf("\naccumulator3 = %d\n", acc3);

}

else if(!strcmp(check\_value[0], "ST3"))

{

printf("\naccumulator3 = %d\n", acc3);

printf("\nstored in memory location: %p", &acc3);

}

printf("\n");

}

**Function4():**

void Function4() ///Accumulator 4

{

int x;

if(!strcmp(check\_value[0], "ADD4"))

{

x = atoi(check\_value[1]); ///Convert String to Intiger

acc4 = acc4 + x;

printf("\naccumulator4 = %d\n", acc4);

}

else if(!strcmp(check\_value[0], "SUB4"))

{

x = atoi(check\_value[1]); ///Convert String to Intiger

acc4 = acc4 - x;

printf("\naccumulator4 = %d\n", acc4);

}

else if(!strcmp(check\_value[0], "MUL4"))

{

x = atoi(check\_value[1]); ///Convert String to Intiger

acc4 = acc4 \* x;

printf("\naccumulator4 = %d\n", acc4);

}

else if(!strcmp(check\_value[0], "DIV4"))

{

x = atoi(check\_value[1]); ///Convert String to Intiger

acc4 = acc4 / x;

if(x == 0)

{

printf("Math error! Cannot divide by 0!\n");

}

else

{

printf("\naccumulator4 = %d\n", acc4);

}

}

else if(!strcmp(check\_value[0], "AND4"))

{

x = atoi(check\_value[1]); ///Convert String to Intiger

acc4 = acc4 & x;

printf("\naccumulator4 = %d\n", acc4);

}

else if(!strcmp(check\_value[0], "NOT4"))

{

acc4 = ~acc4;

printf("\naccumulator1 = %d\n", acc4);

}

else if(!strcmp(check\_value[0], "OR4"))

{

x = atoi(check\_value[1]); ///Convert String to Intiger

acc4 = acc4 | x;

printf("\naccumulator4 = %d\n", acc4);

}

else if(!strcmp(check\_value[0], "LD4"))

{

x = atoi(check\_value[1]); ///Convert String to Intiger

acc4 = x;

printf("\naccumulator4 = %d\n", acc4);

}

else if(!strcmp(check\_value[0], "ST4"))

{

printf("\naccumulator4 = %d\n", acc4);

printf("\nstored in memory location: %p", &acc4);

}

printf("\n");

}

**ParseCommand():**

void parseCommand(int acc1,int acc2, int acc3, int acc4)

{

int x;

if(!strcmp(check\_value[0], "ADD"))

{

acc = acc1 + acc2 + acc3 + acc4;

printf("\nACCUMULATOR = %d\n", acc);

}

else if(!strcmp(check\_value[0], "SUB"))

{

acc = acc1 - acc2 - acc3 - acc4;

printf("\nACCUMULATOR = %d\n", acc);

}

else if(!strcmp(check\_value[0], "MUL"))

{

acc = acc1 \* acc2 \* acc3 \* acc4;

printf("\nACCUMULATOR = %d\n", acc);

}

else if(!strcmp(check\_value[0], "DIV"))

{

int a=0,b=0;

a = acc1 / acc2;

b = a / acc3;

acc = b/acc4;

printf("\nACCUMULATOR = %d\n", acc);

}

else if(!strcmp(check\_value[0], "AND"))

{

acc = acc1 & acc2 & acc3 & acc4;

printf("\nACCUMULATOR = %d\n", acc);

}

else if(!strcmp(check\_value[0], "NOT"))

{

acc = ~acc;

printf("\nACCUMULATOR = %d\n", acc);

}

else if(!strcmp(check\_value[0], "OR"))

{

acc = acc1 | acc2 | acc3 | acc4;

printf("\nACCUMULATOR = %d\n", acc);

}

else if(!strcmp(check\_value[0], "LD"))

{

x = atoi(check\_value[1]); ///Convert String to Intiger

acc = x;

printf("\nACCUMULATOR = %d\n", acc);

}

else if(!strcmp(check\_value[0], "ST"))

{

printf("\nACCUMULATOR = %d\n", acc);

printf("\nstored in memory location: %p", &acc);

}

//printf("\n");

}

**Debugging-Test-run:**

Debugging and test runs yielded satisfactory results, as the code was able to correctly interpret assembly language inputs by the user. Screenshots and truth-test results are given below:

**LD, ADD, SUB, MUL, DIV, ST:**

Input:

LD1 **5**

ADD1 **5**

LD2 **10**

MUL2 **10**

ADD

ST

Output:

****

**Bitwise AND:**

Input:

LD1 12

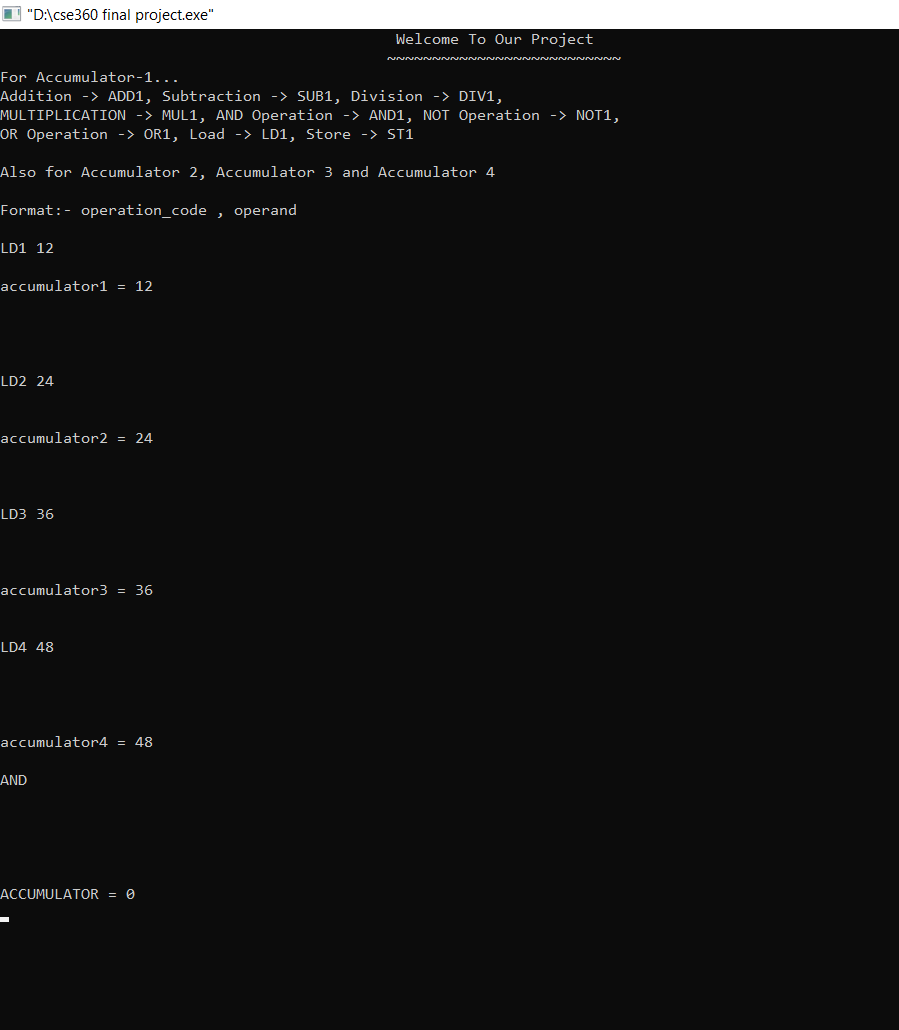
LD2 24

LD3 36

LD4 48

**AND**

Output:



Truth Test:

12 =00001100

24 =00011001

36 =00100100

48 =00110000

AND

—-----------------

00000

**Bitwise OR:**

Input:

LD1 **12**

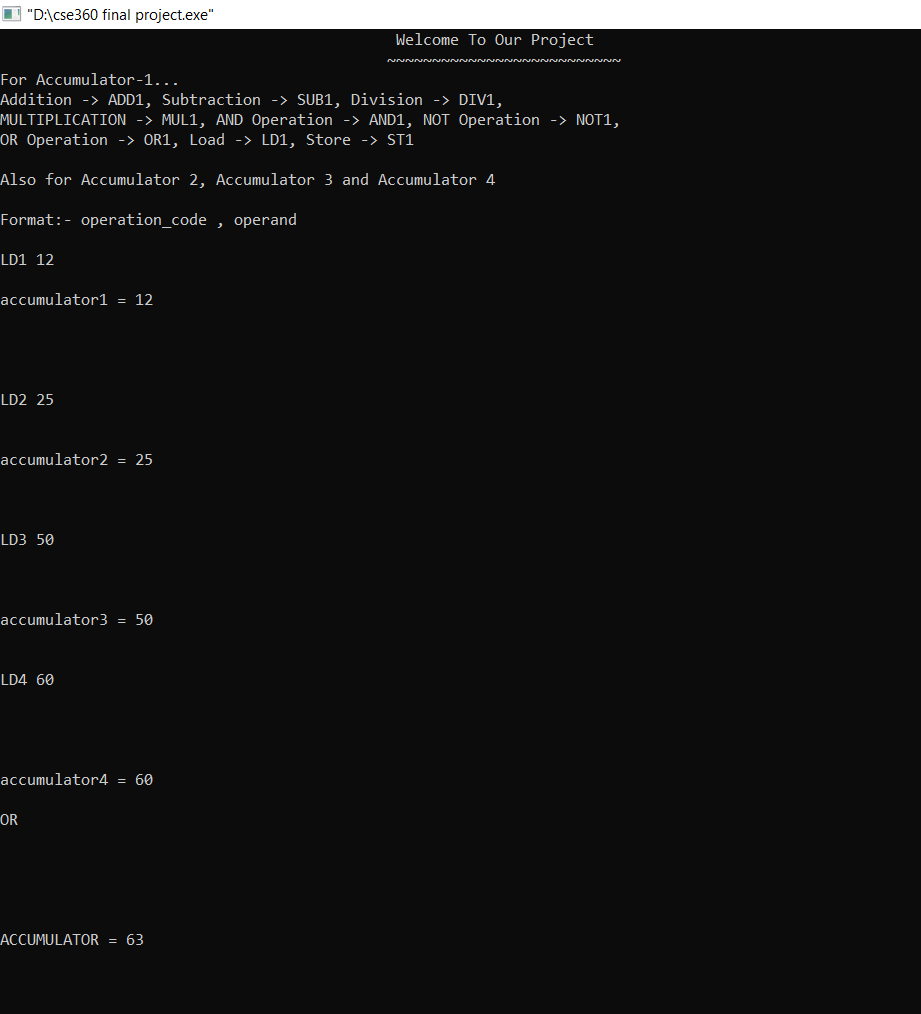
LD2 **25**

**LD3 50**

**LD4 60**

**OR**

Output:



Truth Test:

12 = 00001100

25 =00011001

50 =00110010

60 =00111100

OR

—-----------------

00111111

**Bitwise NOT:**

Input:

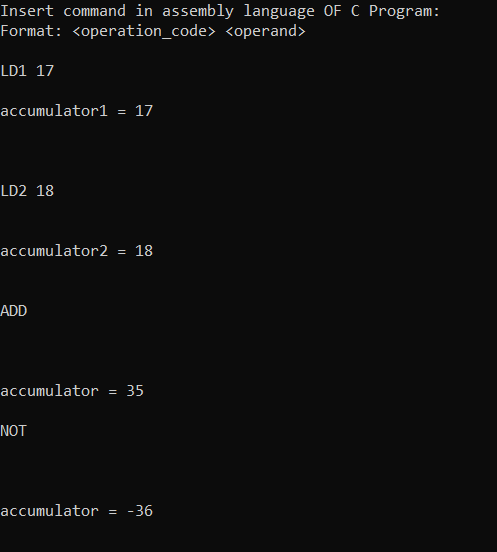
LD1 17

LD2 18

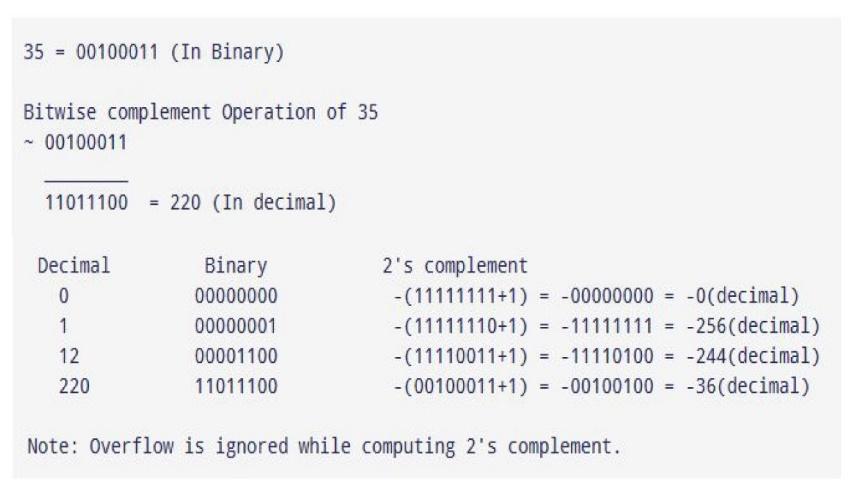
AND

**NOT**

Output:



Truth Test:



**Conclusion and Future Improvements:**

In our project, we tried our best to solve all the bugs while designing this code. Our design gives us the output that we want from it. For illustration, our design can rightly compute the operation of Addition, Multiplication, Subtraction, and Division and so on it gives us the accurate output that we want. We were successful in making the code fully functional without any errors. So that it is clear that, our project is ready. In the Future, we will try to add more accumulators and it will consume less time. The accumulator’s memory location can be saved and we can take this from that location later. We will recommend exhaustive bug testing and bug fixes.

**References:**

1. Computer Organization and Architecture: Designing for Performance (8th Edition) -William Stallings - Prentice-Hall, Inc. Division of Simon and Schuster One Lake Street Upper Saddle River, NJ, United States
2. <http://bankofinfo.com/arithmetic-logic-unit-of-microcomputer/>
3. <https://www.fullchipdesign.com/computer_organization.htm>
4. <https://www.geeksforgeeks.org/computer-organization-instruction-formats-zero-one-two-three-address-instruction/>
5. <https://www.geeksforgeeks.org/introduction-of-general-register-based-cpu-organization/>
6. **https://en.wikipedia.org/wiki/Accumulator\_(computing)**