Project Report

On

8-BIT ALU

Abstract:

The central processing unit (CPU) of a computer is comprised with digital circuits called arithmetic logic units (ALU) that are capable of performing billions of arithmetic and logic operations every second. Learn how to define arithmetic logic unit (ALU), the design and function of an ALU, and how ALUs perform logic and arithmetic calculations to process binary sequences.

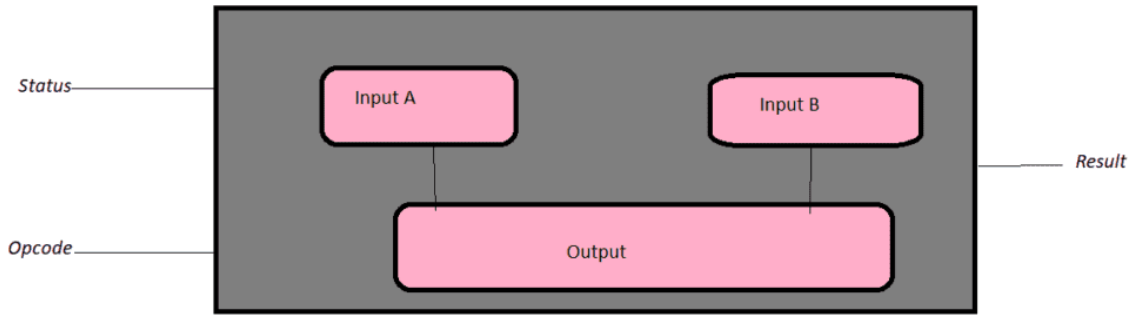
ALU(Arithmetic and Logical Unit):-

An arithmetic logic unit (ALU) is a digital circuit used to perform arithmetic and logic operations. It represents the fundamental building block of the central processing unit (CPU) of a computer. Modern CPUs contain very powerful and complex ALUs. In addition to ALUs, modern CPUs contain a control unit (CU).

Most of the operations of a CPU are performed by one or more ALUs, which load data from input registers. A register is a small amount of storage available as part of a CPU. The control unit tells the ALU what operation to perform on that data, and the ALU stores the result in an output register. The control unit moves the data between these registers, the ALU, and memory.

WORKING MODE OF ARITHMETIC LOGIC UNIT

ALU performs Arithmetic and Logical Operations.



Arithmetic Operations include Addition, Subtraction, Multiplication, and Division.

Logical Operations include operations using AND, OR, and NOT. It does comparison of operations.

The Computer manipulates and stores numbers in terms of 0's and 1's. Transistor switches are used to do these operations as they accept values only in terms of 0's and 1's.

Open and Closed Switch concept is used.

Open switch is a device in which no current passes through and it represents the value '0'. Closed switch is a device in which current passes through and it represents the value '1'.

Multiple transistors can be connected and the resulted output can be obtained. The first transistor can be connected to the second one and in turn, control the operation of the second transistor. The second transistor can be switched ON or OFF depending on the state if the first processor. This is called 'GATE' (logical gates) Gate is the one that allows the flow of current.

Now let us see in detail about Gates. There are 3 gates. AND, OR, and NOT gate.

OR gate:
In this, we give two inputs and get one output. If Input A is 0 and input B is 0, then output C is 0. If input A is 0 and input B is 1, then output C is 1. If input A is 1 and input B is 0, then output C is 1. If input A is 1 and input B is 1 then output C is also 1.

AND gate:
This AND gate again have two inputs and gives one output. If Input A is 0 and B is 1, then output is 0. If input A is 0 and input B is 1, then output C is 0. If input A is 1 and input B is 0, then output C is 0. If input A is 1 and input B is 1 then output C is also 1.

NOT gate:
NOT gate generally consists of 1 input and 1 output. If input A is 0 then output B is 1. If input A is 1 then output B is 0.

XOR gate:
The X or is the reverses version of the OR gate. In XOR gate if Input A is 0 and B is 0 then output C is 0. if input A is 0 and B is 1 then output C is 1. If input A is 1 and input B is 0 then output C is 1. If input A is 1 and B is 1 then output C is 1.

NOR gate:
In the NOR gate if input A is 0 and B is 0 then output C is 1. if input A is 0 and B is 1 then output C is 0. If input A is 1 and input B is 0 then output C is 0. If input A is 1 and B is 1 then output C is 0.

NAND gate:
In the NAND gate, if input A is 0 and B is 0 then output C is 1. If input A is 0 and B is 1 then output C is 1. If input A is 1 and input B is 0 then output C is 1. If input A is 1 and B is 1 then output C is 0.

BIT SHIFTING OPERATIONS

A bit shifting operation is done to shift the most significant bit either to the right or left. There are three types of bit-shifting operations:

LEFT ARITHMETIC SHIFT:
In a left Arithmetic shift, the most significant bit is shifted towards the right. The zeros are shifted on the right.

RIGHT ARITHMETIC SHIFT:
In a right Arithmetic shift, the most significant bit is shifted towards the left. The zeros are shifted in the left.

RIGHT LOGICAL SHIFT:
In the Right Local Shift, the zeros are shifted to the left and the point to be noted is the least significant bit is lost.

ARITHMETIC OPERATIONS

Arithmetic operations here mean addition and subtraction. Multiplication and division are varied rarely used or not used at all. In such cases addition is used as a substitute for multiplication and subtraction is used as a substitute for the division.

PARTS OF ARITHMETIC LOGIC UNIT

Arithmetic Logic Unit consists of:

- Input and Output Access to Controller (Central Processing Unit).
- Main Memory or Random-Access Memory (RAM)
- Input / Output Devices.

The electronic path through which the inputs and outputs flow is called a bus.

The input sometimes consists of an operational code (opcode) that contains the instruction (machine instruction) and at times even a format code.

The Operation code (Opcode) tells the computer what operation needs to be done and also paves the way for the operand to do the task. Here comes the Arithmetic and Logical Separation.

It can simply add two numbers which are called arithmetic operations or it can compare two numbers and produce the output which is called logical operation.

The desired output that we get us now being checked by the format code. It tells whether the obtained output is a fixed bit number (which is an integer) or a floating bot number (which is a decimal).

The output is then placed in a place called Register.

Registers are temporary storage spaces available on the computer. They are quickly accessible to the computer's processor. They are generally a small amount storage but they tend to act very fast. They at times consists of hardware devices and these devices may be Read-Only or Write-Only.

The registers check if the given operation has been performed successfully.

If the output is not stored in the Registers,

it is at times stored in the Machine status word. Machine Status Word machine the permanent space on the memory, while registers are the temporary ones. It then determines the result of the operation that is performed.

In general, the ALU is comprised of the storage spaces for the inputs that are being given by TTE users, operations that are being performed by the user, and the output that is extracted.

The Accumulated Results are stored in the Accumulator.

Accumulator is generally used to store intermediate results. The operations performed and the flow of bits in between is in turn controlled by the Gates. The Gates are controlled by Sequence Logic Units (SLU). They use a separate algorithm or formula for each running code. In ALU, we can store negative values as well. Two operators can be compared and found here bits do not match with each other.

Arithmetic Logic Unit Slices called ALU slices perform operations on a single bit. There is only one ALU slice for each bit in the operation.

CONFIGURATIONS OF THE ALU

Now we have to define how ALU's interact with the processor. Each ALU consists of the following configurations:

- Instruction Set Architecture (ISA)
- Accumulator
- Stack
- Register to Register
- Register Stack
- Register Memory

ACCUMULATOR:

Accumulator consists of the intermediate results of each operation. This means that Instruction Set Architecture is less complex as there is the need only to store one bit (two bits on other devices). There would not be any need for it to store the destination too.

They are less complex and are generally very fast but the additional codes need to be written to fill the Accumulator with proper values to make it more stable. Unfortunately, Accumulators find it very difficult to execute parallelism with a single processor. Desktop Calculator is an example of an Accumulator.

STACK:

Stack is the place where the latest operation performed is stored. It is a small register. It stores programs in top-down order. As new instructions come in, they push to put the old instructions.

REGISTER-REGISTER ARCHITECTURE:

It is otherwise called a 3-register operation machine. It consists of 2 source instructions and also a place for 1 destination instruction. This ISA needs to be more in length to hold three operands (2 source and 1 destination). The word length should be longer and also it would be difficult to write the results back to the Registers after the end of the operations. Write back rule would be followed at this place and therefore it would cause more issues with synchronization.

A good example of the Register-register network is the MIPS component. It uses two operands for input and a third separate component for output. Each requires a separate memory and therefore the space is difficult to maintain, it has to be premium always. Additionally, some operations might be difficult to perform.

REGISTER – STACK ARCHITECTURE:

It is generally a combination of Register and Accumulator operations. In the register-stack Architecture, the operations that need to be performed are pushed onto the top of the stack and the results are stored in the top of the stack as well.

Complicated mathematical operations have to be decomposed using the Reverse polish method. This reverse polish methodology can be difficult for some programmers whereas easy for other programmers because they use the concept of binary trees to represent operands.

New Hardware needs to be created to carry out Push and Pop operations in addition to the operations that are being performed to detect and handle the errors caused in stack memory. (Full stack may be pushed or an empty stack may be popped).

These machines are at times called as “0 – operand machines “as there is no necessity to carry out any new operation and everything occurs on the same stack location.

IA-32uses Register to stack to store floating values and register to Register to store integers. It uses registers as its first operand and register or main memory as its second operand.

Accumulator stores the results obtained. The length of the instructions here is too long this making the architecture too complex to understand and implement.

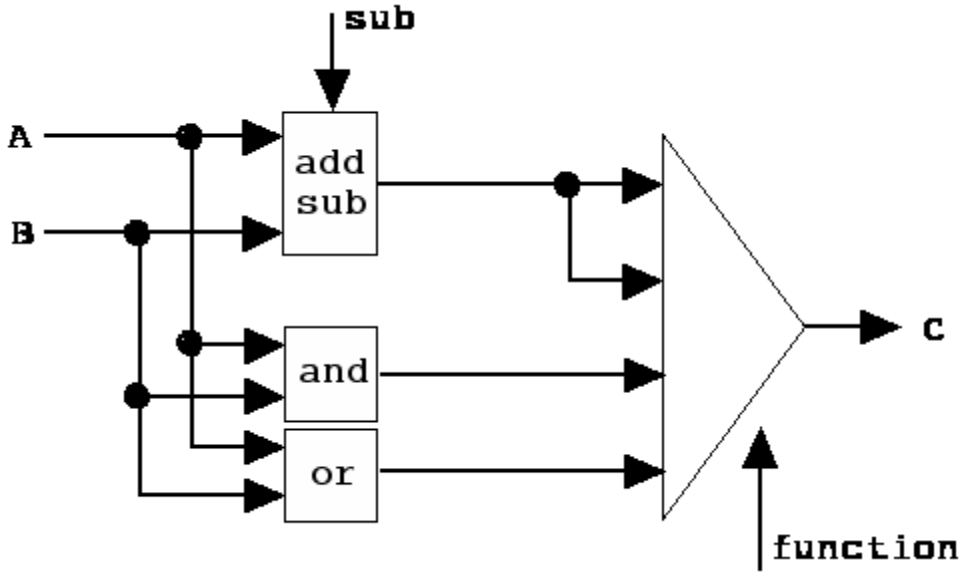
Let’s take AX and BX as two operands and we have to add them. The result would be stored back in AX as well.

REGISTER AND MEMORY:
It is one of the most complicated architectures. In this one operand comes from the external memory and the other operand comes from the register. The reason for it being complicated is because each instruction needs to be stored in a full memory space which might be very long.

Practically this technology cannot be used separately, and is generally integrated with Register-Register Register technology.

CICC architecture uses this format where complete memory spaces would be occupied to store instruction words.

Thus, we have seen all the units of ALU and their procedures related to memory. Thigh it would be a little complex, we have to try and master those to obtain the best results.



ALU DESIGNING IN VERILOG:

Main entity:

```
module alu(

        input [7:0] A,B,  // ALU 8-bit Inputs

        input [3:0] ALU_Sel,// ALU Selection

        output [7:0] ALU_Out, // ALU 8-bit Output

        output CarryOut // Carry Out Flag

    );

    reg [7:0] ALU_Result;

    wire [8:0] tmp;

    assign ALU_Out = ALU_Result; // ALU out

    assign tmp = {1'b0,A} + {1'b0,B};

    assign CarryOut = tmp[8]; // Carryout flag

    always @(*)
```

```
begin
    case(ALU_Sel)
    4'b0000: // Addition
      ALU_Result = A + B ;
    4'b0001: // Subtraction
      ALU_Result = A - B ;
    4'b0010: // Multiplication
      ALU_Result = A * B;
    4'b0011: // Division
      ALU_Result = A/B;
    4'b0100: // Logical shift left
      ALU_Result = A<<1;
     4'b0101: // Logical shift right
      ALU_Result = A>>1;
     4'b0110: // Rotate left
      ALU_Result = {A[6:0],A[7]};
     4'b0111: // Rotate right
      ALU_Result = {A[0],A[7:1]};
      4'b1000: //  Logical and
      ALU_Result = A & B;
      4'b1001: //  Logical or
      ALU_Result = A | B;
      4'b1010: //  Logical xor
      ALU_Result = A ^ B;
      4'b1011: //  Logical nor
      ALU_Result = ~(A | B);
      4'b1100: // Logical nand
      ALU_Result = ~(A & B);
      4'b1101: // Logical xnor
      ALU_Result = ~(A ^ B);
      4'b1110: // Greater comparison
```

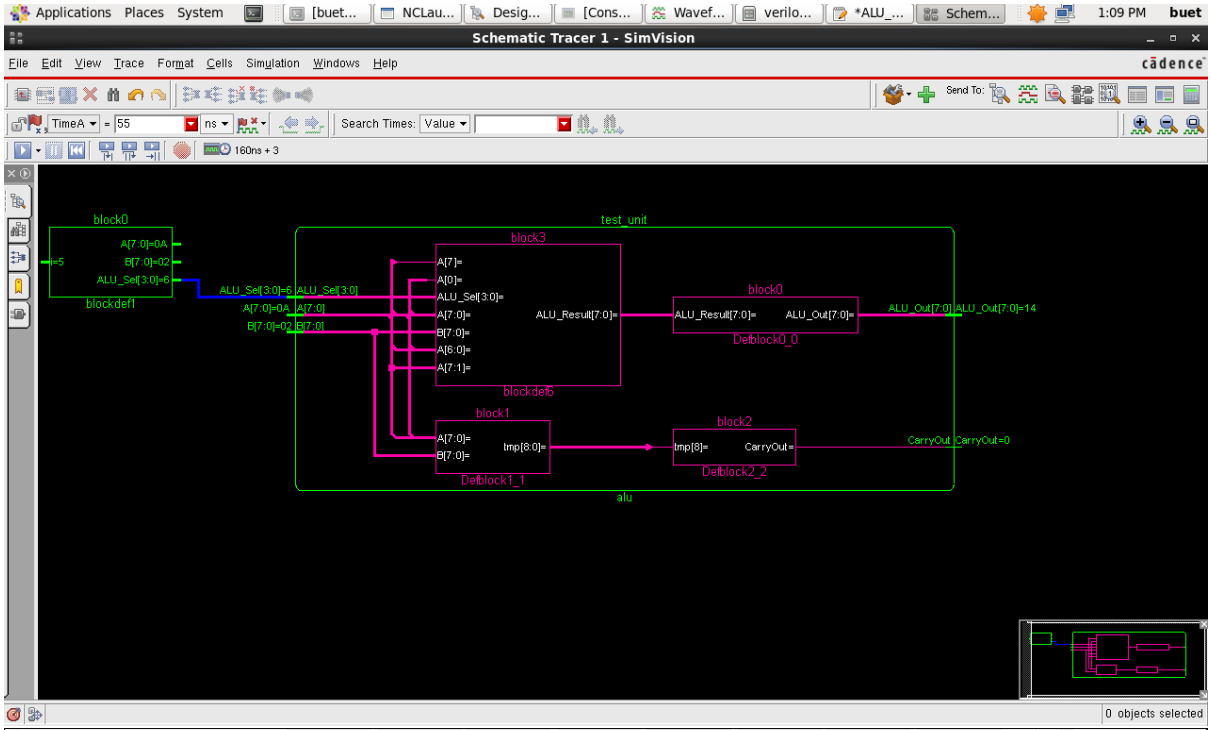
```
          ALU_Result = (A>B)?8'd1:8'd0 ;
         4'b1111: // Equal comparison
           ALU_Result = (A==B)?8'd1:8'd0 ;
         default: ALU_Result = A + B ;
       endcasewhat
   end
endmodule
```

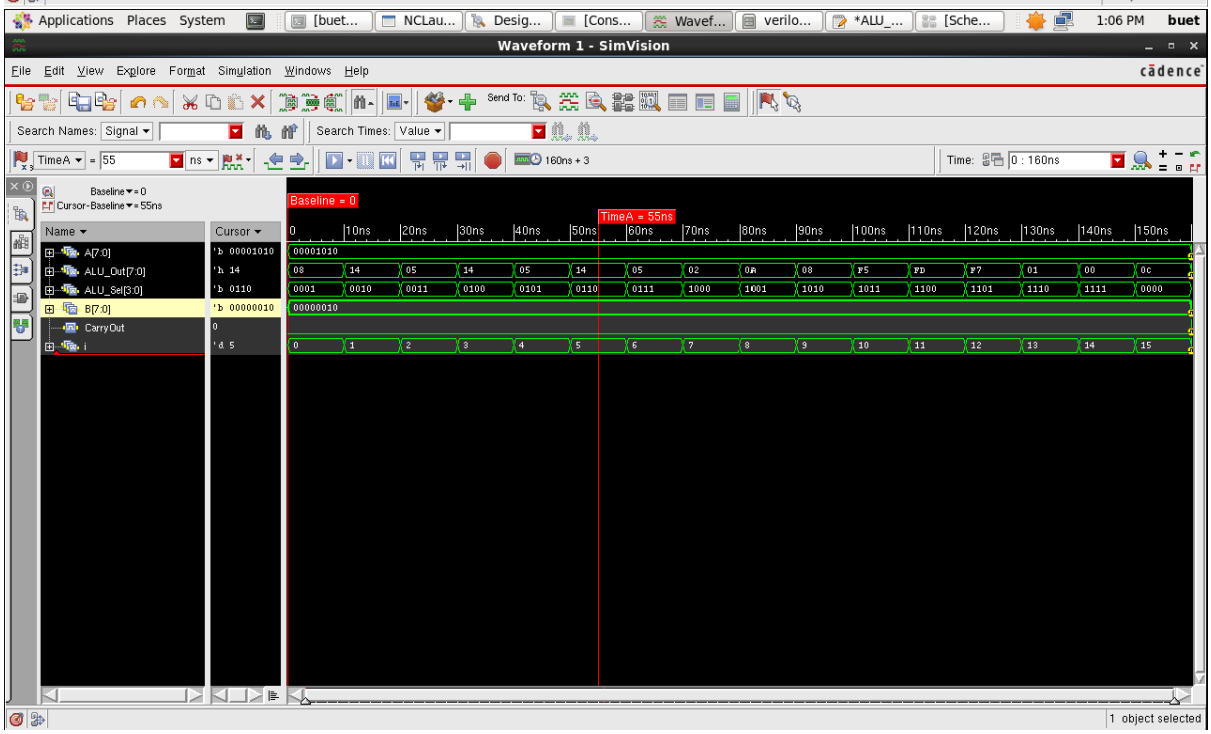
Test bench code:-

```
module tb_alu;
//Inputs
 reg[7:0] A,B;
 reg[3:0] ALU_Sel;
//Outputs
 wire[7:0] ALU_Out;
 wire CarryOut;
 // Verilog code for ALU
 integer i;
 alu test_unit(
          A,B,  // ALU 8-bit Inputs
          ALU_Sel,// ALU Selection
          ALU_Out, // ALU 8-bit Output
          CarryOut // Carry Out Flag
    );
   initial begin
   // hold reset state for 100 ns.
     A = 8'h0A;
     B = 8'h02;
     ALU_Sel = 4'h0;
     for (i=0;i<=15;i=i+1)
     begin
      ALU_Sel = ALU_Sel + 8'h01;
```

```
    #10;
    end
  A = 8'hF6;
    B = 8'h0A;
  end
endmodule
```

Schematics :





ADVANTAGES AND DISADVANTAGES OF ALU

ADVANTAGES:

- It supports parallel architecture.
- It supports applications with high performance.
- It can combine integer and floating-point variables, simultaneously and get the desired output.
- It can also combine two Arithmetic operations in the same code such as addition and subtraction or addition and multiplication or any two operands. Example, A×B+C.
- It has a high range of Accuracy
- It can perform on a very large set of instructions.
- Thy are spaced so evenly that they never part in between.
- They remain uniform through the entire operation
- There is no memory wastage with ALU.
- It is very fast in general and results can be obtained so easily.
- It has no sensitivity issues.
- They minimize the logic gate requirements.
- They are less expensive as they mi infuse gate values.
- Their methods are very easy to master and implement that the other processor methods on the computer.

DISADVANTAGES:

- The concept of pipelining is complex to understand.
- Memory space should be definite. Else bugs would occur in our result.
- Their circuit is complex and therefore amateurs find it difficult to understand.
- Floating variables have more delays
- Design controller is still more difficult to understand.
- Irregularities in latencies is a proven disadvantage.
- Rounding off is another con to be noted. Large numbers are generally rounded off thus impacting accuracy.

SUMMARY

- Thus, Arithmetic Logic Unit is used for performing arithmetic and logical operations.
- It can perform simple arithmetic calculations to complex arithmetic calculations like integration.
- In the case of logical calculations, it performs them using the concept of Gates.
- Everything that is given as input is converted to 0’s and 1’s and does the required calculations.
- Left shifting a bit, right shifting bits, and carry forward concepts are also performed using the Arithmetic Logic Unit (ALU).

- Registers are small storage spaces on the CPU which is used to store intermediate results.
- Stacks are used to store details of the performed which have been performed recently and it uses “Last in First Out” concept.
- Most of the operations that are performed by the CPU are being performed by the ALU.
- The Working mode of CPU functions well only if the ALU works properly.
- The ALU moves data between CPU, registers, and main memory.
- Any range of memory can be stored by ALU like 4-bit, 8-bit, and 16-bit memory.
- It works on the concept of flow of current where 1 means that it is an active switch and 0 means that switch is not active.
- Many memory units are available like Register-Register, Register-Stack, Accumulators, and so on.
- We have to master the working process of gates to design one, thus fulfilling our requirements.

CONCLUSION:

We have seen the ARITHMETIC Logic Unit in detail. We have seen how to use it for arithmetic and logical operations. We have also seen various gates and how to use them. I hope this would be helpful to learn the concepts of the Arithmetic Logic Unit. Feel free to comment and leave your suggestions, s that we could discuss more.

REFERENCES:

1. https://en.wikipedia.org/wiki/Arithmetic_logic_unit
2. https://study.com/academy/lesson/arithmetic-logic-unit-alu-definition-design-function.html
3. https://www.computerhope.com/jargon/a/alu.htm
4. https://www.dictionary.com/browse/alu
5. https://computersciencewiki.org/index.php/Functions_of_the_arithmetic_logic_unit_(ALU)
6. https://www.webopedia.com/TERM/A/ALU.html