Computer Architecture

Section 6C4

Group 17

Section #	NAME	ID
6c4	*Mjd Mohammed Alamri*	*443007585*
6c4	[redacted]	
6c4		
6c4	[redacted]	
6c4		

Abstract:

The purpose of this project is to design, simulate and implement a Arithmetic Logic Unit **(**ALU**)** using Active-HDL Verilog.
The ALU performs various arithmetic and logical operations, including addition, subtraction, logical AND, logical OR, logical NOT and logical XOR.

Our goal is to create a highly efficient and accurate ALU that can perform all five operations reliably. We used Active-HDL Verilog to design the ALU and Control units, which involves writing code that describes the behavior of the circuit.

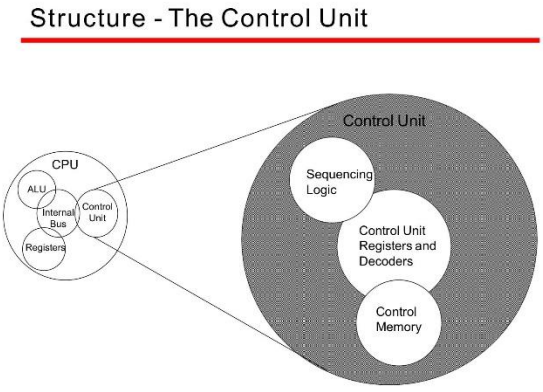
Introduction:

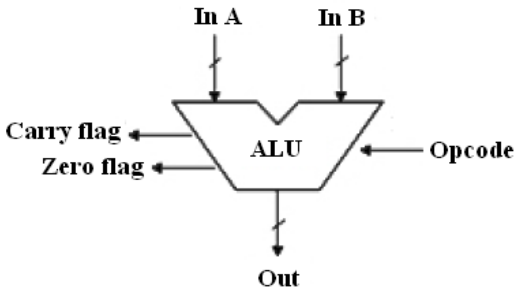
Since our goal is to create an arithmetic logic unit (ALU) and control unit (CU) we have designed one using Verilog Language.

First, we will Discuss what ALU is? ALU is a major component of the computer its main job is to do all processes related to arithmetic and logic operations in some computers the ALU is divided into two unit's arithmetic unit (AU) and logic unit (LU). [1]

For the CU, it is circuitry that directs operations within a computer's processor. It helps the computer's units to know how to respond to instructions received from a program. [2]

And this are pictures to describe the design of each unit:





Explain what ALU is And how does the ALU control work:

As previously said, ALU is a major component of the computer. Its main job is to do all processes related to arithmetic and logic operations, therefore our project's job is to do all the operations we need such as: ADD, SUB, NOT, AND, OR and XOR.

An ALU can be designed to calculate many different operations. When the operations become more complex, the ALU will also become more expensive. [3]

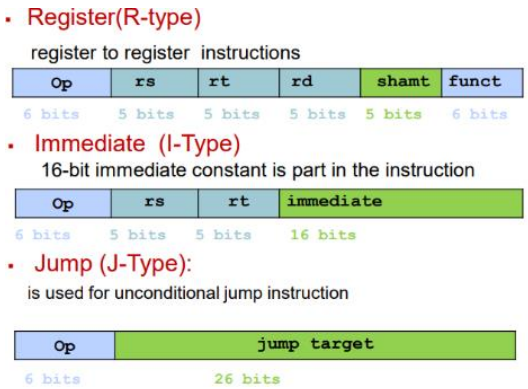
Different operation as carried out by ALU can be categorized as follows:

-logical operations: AND, OR, NOT and XOR.

-bit-shifting operations: shift a certain number either to left or right.

-Arithmetic operations: ADD and SUB. [4]

Since we are using MIPS ISA, there will be 3 different Instruction Formats, R-type, I-type, and J-type each one has exactly 32 bits wide but divided into different sections as shown in the below picture.



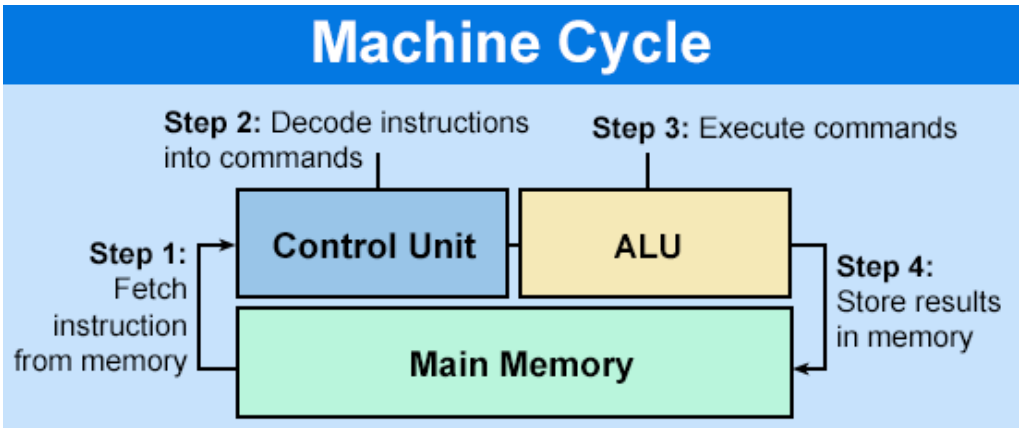
- R-type has 2 main conditions, all the operands must be **register** and opcode must be **zero**. It can do addition, subtraction, shift, NOT, AND, OR and XOR.
- The main condition for I-type the Operands must be Immediate. It is used for ALU instructions fields, load and store and conditional branches.
- J-type used for unconditional jump instruction. [5]

Since we need to perform ADD, SUB, NOT, AND, OR and XOR all we will use is R-type instructions.

If we want to do any instruction of course we will need help from the control unit, the Control Unit is also connected to ALU and main memory through buses. [6]

It will fetch the instruction from the memory and sends it to ALU. After reading the opcode it will know which operation to perform and then after the ALU finish the calculations it will send back the output to the main memory.

This is a basic diagram to describe the relationship in the simplest way:



There are 9 signals that will be out of CU, **RegDst**, **ALUSrc**, **MemtoReg**, **RegWrite**, **MemRead**, **MemWrite**, **Branch**, **ALUOp** (we wrote it as 2bits). They will be determined by which format or operation chosen. Since we will be doing R-type the signals will be as following:

- RegDst = 1
- ALUSrc = 0
- MemtoReg = 0
- RegWrite = 1
- MemRead = 0
- MemWrite = 0
- Branch = 0
- ALUOp (2bits) = 10 . [7]

Snapshots of the code:

-module dd:

```
4       module dd ();
5        wire [3:0] CONT;
6        ALUu_CONTROLu at1(CONT);
7        ALUoperator at2 (CONT,result,carry,ZeroFlag,input1,input2);
8       endmodule
```

This is the basic module, here we connect module ALUu_CONTROLu and module ALUoperator to be able to do the simulation.

-module ALUu_CONTROLu:

```
10  module ALUu_CONTROLu(CONT);
11
12  output wire [3:0] CONT;
13  wire [1:0] ALUOp;
14  CONTROL_UNIT atalso(RegDst,ALUSrc,MemtoReg,RegWrite,MemRead,MemWrite,Branch,ALUOp,OPc);
15  ALU_UNIT at(ALUOp,FUN,CONT);
16
17  endmodule
```

Here we join module CONTROL_UNIT and module ALU_UNIT as one design to be able to connect them to the rest of the program at module dd.

-module ALU_UNIT:

```
module ALU_UNIT(ALUOp,FUN,CONT);
input[1:0] ALUOp;
input[5:0] FUN;
output reg[3:0] CONT;

always @(ALUOp or FUN)//==============================
  begin
  //what kind of opration is it?
    if (ALUOp == 2'b10) // it stands for R-type
     begin

    //add+
      if ( FUN == 6'b100000)
      begin
        CONT=4'b0010;
      end
   //subtract-
      else if (FUN==6'b100010)
     begin
        CONT=4'b0110;
      end
      //and&&
        else if (FUN==6'b100100)
     begin
        CONT=4'b0000;
      end
   //OR||
       else if (FUN==6'b100101)
     begin
        CONT=4'b0001;
      end
  //NOT ~
    else if (FUN==6'b100111)
      begin
        CONT=4'b1100;
      end
    //XOR
    else if (FUN==6'b100110)
      begin
        CONT=4'b0100;
      end
    else
      begin
        CONT=4'bxxxx;
      end

    end

  end//=========================================
  endmodule
```

This module is responsible for determining what arithmetic or logic operation will be used based on what function will be entered and what control signal will be an output.

Inputs: ALUop (ALU operation (R-type), 2bits ,it is an output from module CONTROL_UNIT) , FUN(function value, 6bits).
Output: CONT(control signal,4bits).

-CONTROL_UNIT:

```
70    module CONTROL_UNIT(RegDst,ALUSrc,MemtoReg,RegWrite,MemRead,MemWrite,Branch,ALUOp,OPc);
71          //(MAIN CONTROL)
72    input[5:0] OPc;
73    output reg [1:0] ALUOp;
74    output reg RegDst,ALUSrc,MemtoReg,RegWrite,MemRead,MemWrite,Branch;
75    //-------
76    always @(OPc)
77      if (OPc== 0)
78        begin
79          RegDst=1'b1; //1
80          ALUSrc=1'b0; //2
81          MemtoReg=1'b0; //3
82          RegWrite=1'b1;//4
83          MemRead=1'b0; //5
84          MemWrite=1'b0;//6
85          Branch=1'b0; //7
86          ALUOp=2'b10;//8 & 9
87        end
88    endmodule
```

This module is responsible for determining the 9 control signals for R-type.

Inputs: OPc(Operation code, 6 bits) .
Output: RegDst,ALUSrc,MemtoReg,RegWrite,MemRead,MemWrite,Branch(control signals, all are 1 bit), ALUop(control signal, 2bits(ALUop1 and ALUop0)).

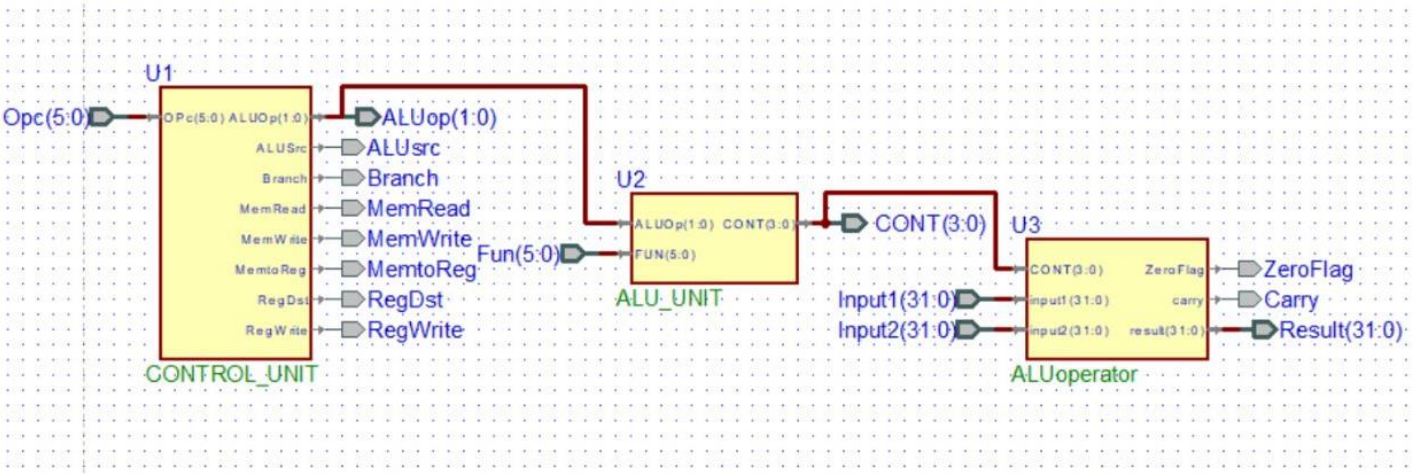
-module ALUoperator:

```
90    module ALUoperator(CONT,result,carry,ZeroFlag,input1,input2);
91    input [3:0] CONT;
92    input [31:0]input1,input2;
93    output reg[31:0] result;
94    output reg ZeroFlag,carry;
95
96    always @(input1 or input2 or CONT)
97      begin
98        //carry with add only-------
99        carry = 1'b0;
100       if (CONT==4'b0010)//add+
101         begin
102           assign {carry,result} = input1+input2; //there might be carry
103         end
104       else if (CONT==4'b0110)//subtract-
105         begin
106             result = input1 - input2; //no carry could happen
107         end
108         else if (CONT==4'b0000)//and&&
109         begin
110             result = input1 & input2;
111         end
112       else if (CONT==4'b0001)//OR||
113         begin
114             result = input1 | input2;
115         end
116       else if (CONT==4'b1100)//NOT ~
117         begin
118             result = ~( input1 | input2 );
119         end
120       else if (CONT==4'b0100)//XOR ^
121         begin
122            result = input1 ^ input2;
123         end
124       else
125         begin
126           result = 32'hxxxxxxxx; //each X indicates 4 binary bits in hexa (no value)
127         end
128    //--------------------zero flag-----
129     if (result==32'd0)
130       begin
131       ZeroFlag=1'b1;
132       end
133     else
134       begin
135       ZeroFlag=1'b0;
136       end
137
138     end
139   endmodule
```

here we assign each arithmetic and logic operation we will need, and the Result, output zero bit and Carry.

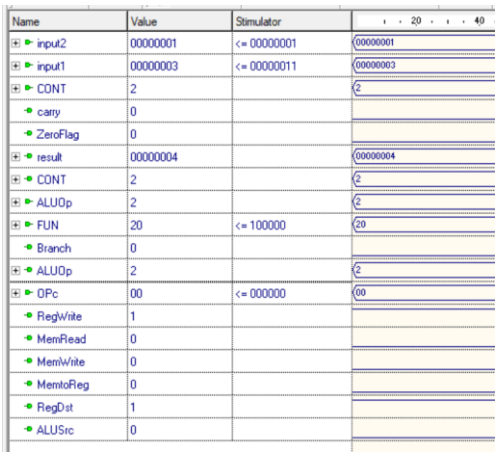
Inputs: CONT(control signal,4bits, it is an output from module ALU_UNIT), input1 and input2(the inputs we will use the operation on , each is 32bits).
Output: result(the output result, 32 bits), ZeroFlag(output zero bit, 1 bit) , carry(carry of the result, 1 bit).

Snapshot of the diagram:



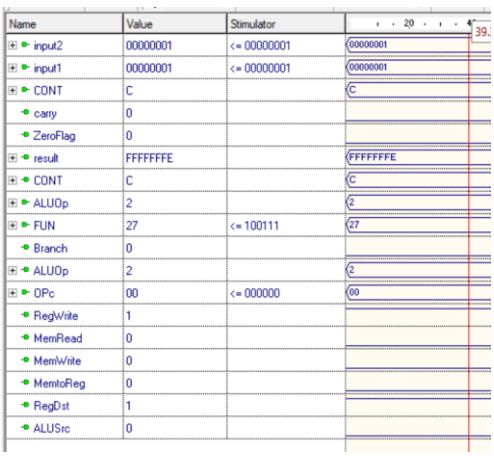
Snapshots of the Simulation trace:

Add:



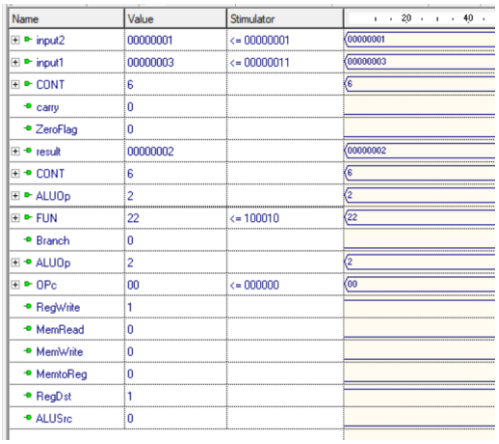
Name	Value	Stimulator
input2	00000001	<= 00000001
input1	00000003	<= 00000011
CONT	2	
carry	0	
ZeroFlag	0	
result	00000004	
CONT	2	
ALUOp	2	
FUN	20	<= 100000
Branch	0	
ALUOp	2	
OPc	00	<= 000000
RegWrite	1	
MemRead	0	
MemWrite	0	
MemtoReg	0	
RegDst	1	
ALUSrc	0	

Not:



Name	Value	Stimulator
input2	00000001	<= 00000001
input1	00000001	<= 00000001
CONT	C	
carry	0	
ZeroFlag	0	
result	FFFFFFFE	
CONT	C	
ALUOp	2	
FUN	27	<= 100111
Branch	0	
ALUOp	2	
OPc	00	<= 000000
RegWrite	1	
MemRead	0	
MemWrite	0	
MemtoReg	0	
RegDst	1	
ALUSrc	0	

Sub:

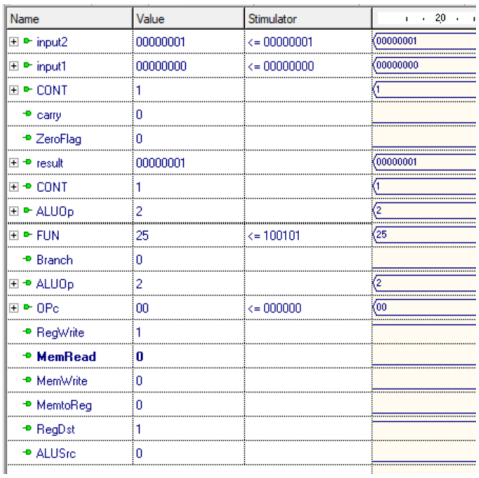


Name	Value	Stimulator
input2	00000001	<= 00000001
input1	00000003	<= 00000011
CONT	6	
carry	0	
ZeroFlag	0	
result	00000002	
CONT	6	
ALUOp	2	
FUN	22	<= 100010
Branch	0	
ALUOp	2	
OPc	00	<= 000000
RegWrite	1	
MemRead	0	
MemWrite	0	
MemtoReg	0	
RegDst	1	
ALUSrc	0	

And:

Name	Value	Stimulator
input2	00000001	<= 00000001
input1	00000000	<= 00000000
CONT	0	
carry	0	
ZeroFlag	1	
result	00000000	
CONT	0	
ALUOp	2	
FUN	24	<= 100100
Branch	0	
ALUOp	2	
OPc	00	<= 000000
RegWrite	1	
MemRead	0	
MemWrite	0	
MemtoReg	0	
RegDst	1	
ALUSrc	0	

Or:



Name	Value	Stimulator
input2	00000001	<= 00000001
input1	00000000	<= 00000000
CONT	1	
carry	0	
ZeroFlag	0	
result	00000001	
CONT	1	
ALUOp	2	
FUN	25	<= 100101
Branch	0	
ALUOp	2	
OPc	00	<= 000000
RegWrite	1	
MemRead	**0**	
MemWrite	0	
MemtoReg	0	
RegDst	1	
ALUSrc	0	

Xor:

Name	Value	Stimulator
input2	00000001	<= 00000001
input1	00000001	<= 00000001
CONT	4	
carry	0	
ZeroFlag	1	
result	00000000	
CONT	4	
ALUOp	2	
FUN	26	<= 100110
Branch	0	
ALUOp	2	
OPc	00	<= 000000
RegWrite	1	
MemRead	0	
MemWrite	0	
MemtoReg	0	
RegDst	1	
ALUSrc	0	

References:

[1]. Tutorialspoints, Arithmetic Logic Unit (ALU) (tutorialspoint.com)

[2]. Computer hope, What is a Control Unit? (computerhope.com)

[3].[4]. Tutorialspoints, Arithmetic Logic Unit (ALU) (tutorialspoint.com)

[5].David A Patterson and John L. Hennessy, Computer Organization and Design, 2014

[6]. QodiQ, ALU(Arithmetic Logic Unit) and CU(Control Unit). Definitions, Types, Working, and Data Path. 2021 - QodiQ

[7] David A Patterson and John L. Hennessy, Computer Organization and Design, 2014