
16-BIT ARITHMETIC LOGIC UNIT (ALU)

Name	SEC	ID
Ahmed Hassan Abdelhalim Labib	1	91240075
Mohamed Diaa Eldin	3	91240665

Supervised by and presented to Eng. Ahmed Atef

Contents

1.	Overview.....	3
2.	Architecture Specifications	3
3.	Functional Blocks.....	3
4.	Status Flag	3
5.	Code	5
6.	Testbench.....	8
7.	Simulation	10
8.	GitHub	12

1. Overview

This module implements a 16-bit Arithmetic Logic Unit (ALU) designed for a custom processor architecture. It performs arithmetic, bitwise logical, and shift operations based on a 5-bit control signal (F). The design features a comprehensive Status Register to support conditional branching and complex arithmetic (both signed and unsigned).

2. Architecture Specifications

- Data Width: 16-bit signed/unsigned integers.
- Control Width: 5-bit Opcode (F), allowing for up to 32 distinct operations.
- Input Stage: Two 16-bit operands (A, B) and a 1-bit Carry Input (C_{in}).
- Output Stage: One 16-bit Result and a 6-bit Status vector.
- Design Type: Purely combinational logic (`always @(*)`), ensuring result validity immediately after propagation delay.

3. Functional Blocks

The ALU is divided into three primary logical groups:

1. Arithmetic Unit ($F = 00000 - 00111$):
 - Supports Addition (**ADD**) and Subtraction (**SUB**).
 - Supports operations with Carry/Borrow (**ADC**, **SBB**) for multi-word arithmetic.
 - Includes Increment (**INC**) and Decrement (**DEC**).
 - Note: Flags (Overflow, Aux Carry) are calculated manually to ensure precision for signed arithmetic.
2. Logic Unit ($F = 01000 - 01011$):
 - Standard bitwise operations: **AND**, **OR**, **XOR**, and **NOT** (1's complement).
3. Shift/Rotate Unit ($F = 10000 - 10111$):
 - Performs Logical Shifts (Left/Right).
 - Performs Rotations involving the Carry flag (Rotate through Carry), allowing for efficient bit-stream manipulation.

4. Status Flag

1. Carry Flag (C) – Bit [5]

- Function: Indicates an arithmetic carry or borrow generated out of the Most Significant Bit (MSB).

- **Derivation:** The arithmetic operations are performed using a 17-bit width internally. The 17th bit (index 16) is extracted to determine the Carry flag.
- **Formula:** $C = \text{Result}[16]$

2. Zero Flag (Z) – Bit [4]

- **Function:** Indicates whether the output of the operation is effectively zero.
- **Derivation:** This is calculated using a logical NOR reduction across all 16 bits of the result. If any single bit in the result is high (1), the Zero flag is cleared to 0.
- **Formula:** $Z = (\text{Result} == 0)$

3. Negative Flag (N) – Bit [3]

- **Function:** Indicates if the result is negative when interpreted as a signed Two's Complement number.
- **Derivation:** In Two's Complement notation, the MSB determines the sign. The ALU simply copies the MSB of the result to this flag.
- **Formula:** $N = \text{Result}[15]$

4. Overflow Flag (V or OF) – Bit [2]

- **Function:** Indicates Signed Overflow. This occurs when the result of an arithmetic operation exceeds the capacity of the 16-bit signed range (-32,768 to +32,767).
- **Derivation:** Overflow logic detects cases where the result sign is mathematically impossible given the operand signs:
- **Addition:** Overflow occurs if adding two positive numbers yields a negative result, or adding two negative numbers yields a positive result.
- **Subtraction:** Overflow occurs if subtracting a negative number from a positive number yields a negative result (and vice versa).
- **Formula (Addition):** $\text{OF} = (\text{A}[15] == \text{B}[15]) \ \&\& (\text{Result}[15] != \text{A}[15]);$

5. Parity Flag (P) – Bit [1]

- **Function:** Indicates the parity of the result. This design implements Even Parity, meaning the flag is set high if the result contains an even number of 1s.
- **Derivation:** This is calculated using an XNOR reduction of all bits in the result.
- **Formula:** $P = \text{XNOR}(\text{Result}[15:0])$

6. Auxiliary Carry Flag (AC or Aux) – Bit [0]

- **Function:** Indicates a carry-out or borrow-out specifically from the lower nibble (Bit 3).

- **Derivation:** The ALU monitors the summation of the first 4 bits (3:0). If the sum exceeds 15 (0xF), the Aux flag is asserted.
- **Formula:** $AUX = (A[3:0] + B[3:0] + C_{in}) > 15$

5. Code

```

1 module alu (
2     input [15:0] A,
3     input [15:0] B,
4     input [4:0] F,
5     input Cin,
6     output reg [15:0] Result,
7     output [5:0] Status // {C, Z, N, V, P, A}
8 );
9
10    reg      C;
11    reg      V;
12    reg      Aux;
13
14
15    assign Status[5] = C;
16    assign Status[4] = (Result == 16'h0000);
17    assign Status[3] = Result[15];
18    assign Status[2] = V;
19    assign Status[1] = ~^Result;
20    assign Status[0] = Aux;
21
22    always @(*) begin
23
24        Result = 16'h0000;
25        C = 0;
26        V = 0;
27        Aux = 0;
28
29        case (F)
30
31 /*=====Arithmetic=====*/
32         5'b00000: begin
33             Result = 16'h0000;
34             C=0;
35             V=0;
36             Aux=0;
37         end
38
39         5'b000001: begin
40             {C,Result} = A + 1'b1;
41             V = (A == 16'h7FFF);
42             Aux = (({1'b0, A[3:0]} + 1'b1) > 5'hF);
43         end

```

```

44
45      5'b00010: begin
46          Result = 16'h0000;
47          C=0;
48          V=0;
49          Aux=0;
50      end
51
52      5'b00011: begin
53          {C,Result} = A - 1'b1;
54          // C    = ~Res[16];
55          V    = (A == 16'h8000);
56          Aux   = (A[3:0] == 4'h0);
57      end
58
59      5'b00100: begin
60          {C,Result} = A + B;
61          V    = (A[15] == B[15]) && (Result[15] != A[15]);
62          Aux  = (({1'b0, A[3:0]} + {1'b0, B[3:0]}) > 5'hF);
63      end
64
65      5'b00101: begin
66          {C,Result} = A + B + Cin;
67          V    = (A[15] == B[15]) && (Result[15] != A[15]);
68          Aux  = (({1'b0, A[3:0]} + {1'b0, B[3:0]} + Cin) > 5'hF);
69      end
70
71      5'b00110: begin
72          {C,Result} = A - B;
73          V    = (A[15] != B[15]) && (Result[15] != A[15]);
74          Aux   = (A[3:0] < B[3:0]);
75      end
76
77      5'b00111: begin
78          {C,Result} = A - B - Cin;
79          V    = (A[15] != B[15]) && (Result[15] != A[15]);
80          Aux   = ({1'b0, A[3:0]} < ({1'b0, B[3:0]} + Cin));
81      end
82
83
84 /*=====logic=====*/
85
86
87      5'b01000: Result = A & B;
88      5'b01001: Result = A | B;
89      5'b01010: Result = A ^ B;
90      5'b01011: Result = ~A;
91
92
93
94 /*=====Shift=====*/
95

```

```

96      5'b10000: begin
97          C = A[15];
98          Result = A << 1;
99      end
100
101     5'b10001: begin
102         C = A[0];
103         Result = A >> 1;
104     end
105
106     5'b10010: begin
107         C = A[15];
108         Result = A << 1;
109     end
110
111     5'b10011: begin
112         C = A[0];
113         Result = {A[15],A[15:1]};
114     end
115
116     5'b10100: begin
117         C = A[15];
118         Result = {A[14:0], A[15]};
119     end
120
121     5'b10101: begin
122         C = A[0];
123         Result = {A[0], A[15:1]};
124     end
125
126     5'b10110: begin
127         C = A[15];
128         Result = {A[14:0], Cin};
129     end
130
131     5'b10111: begin
132         C = A[0];
133         Result = {Cin, A[15:1]};
134     end
135
136     default: begin
137         Result = 16'h0000;
138         C=0;
139         V=0;
140         Aux=0;
141     end
142     endcase
143 end
144
145 endmodule

```

6. Testbench

```
1 `timescale 1ns / 1ps
2
3 module alu_tb;
4     reg [15:0] A;
5     reg [15:0] B;
6     reg [4:0] F;
7     reg Cin;
8     wire [15:0] Result;
9     wire [5:0] Status; // Status Flags [C, Z, N, V, P, A]
10    wire C_flag = Status[5];
11    wire Z_flag = Status[4];
12    wire N_flag = Status[3];
13    wire O_flag = Status[2];
14    wire P_flag = Status[1];
15    wire A_flag = Status[0];
16    alu uut (
17        .A(A),
18        .B(B),
19        .F(F),
20        .Cin(Cin),
21        .Result(Result),
22        .Status(Status)
23    );
24
25 initial begin
26     A = 0;
27     B = 0;
28     F = 0;
29     Cin = 0;
30
31 /*=====Arithmetic=====*/
32     // 1. F = 00000: N/A
33     F = 5'b00000; A = 16'h0000; B = 16'h0000; Cin = 0;
34     #10;
35     // 2. F = 00001: INC (Increment A)
36     F = 5'b00001; A = 16'h0B05; B = 16'h0000; Cin = 0;
37     #10;
38     // 3. F = 00010: N/A
39     F = 5'b00010; A = 16'h0B05; B = 16'h0000; Cin = 0;
40     #10;
41     // 4. F = 00011: DEC (Decrement A)
42     F = 5'b00011; A = 16'h0B05; B = 16'h0006; Cin = 0;
43     #10;
44     // 5. F = 00100: ADD
```

```

45      F = 5'b00100; A = 16'h0BA5; B = 16'h0003; Cin = 0;
46      #10;
47      // ADD Overflow Test
48      F = 5'b00100; A = 16'h7FFF; B = 16'h0001; Cin = 0;
49      #10;
50      // 6. F = 00101: ADD CARRY
51      F = 5'b00101; A = 16'h0AB5; B = 16'h0CF3; Cin = 1;
52      #10;
53      // 7. F = 00110: SUB
54      F = 5'b00110; A = 16'h050A; B = 16'h0BAC; Cin = 0;
55      #10;
56      // 8. F = 00111: SUB BORROW
57      F = 5'b00111; A = 16'h000A; B = 16'h0003; Cin = 1;
58      /*=====logic=====*/
59      #10;
60      // 9. F = 01000: AND
61      F = 5'b01000; A = 16'hFFFF; B = 16'h00FF; Cin = 0;
62      #10;
63      // 10. F = 01001: OR
64      F = 5'b01001; A = 16'hF000; B = 16'h000F; Cin = 0;
65      #10;
66      // 11. F = 01010: XOR
67      F = 5'b01010; A = 16'hAAAA; B = 16'h5555; Cin = 0;
68      #10;
69      // 12. F = 01011: NOT
70      F = 5'b01011; A = 16'h1010; B = 16'h0000; Cin = 0;
71      /*=====Shift=====*/
72      #10;
73      // 13. F = 10000: SHL (Shift Left Logical)
74      F = 5'b10000; A = 16'h0501; B = 16'h0001; Cin = 0;
75      #10;
76      // 14. F = 10001: SHR (Shift Right Logical)
77      F = 5'b10001; A = 16'h0C02; B = 16'h0001; Cin = 0;
78      #10;
79      // 15. F = 10010: SAL (Arithmetic Shift Left)
80      F = 5'b10010; A = 16'h7FFF; B = 16'h0001; Cin = 0;
81      #10;
82      // 16. F = 10011: SAR (Arithmetic Shift Right)
83      F = 5'b10011; A = 16'h80F0; B = 16'h0001; Cin = 0;
84      #10;
85      // 17. F = 10100: ROL (Rotate Left)
86      F = 5'b10100; A = 16'h8001; B = 16'h0001; Cin = 0;
87      #10;
88      // 18. F = 10101: ROR (Rotate Right)
89      F = 5'b10101; A = 16'h8001; B = 16'h0001; Cin = 0;
90      #10;
91      // 19. F = 10110: RCL (Rotate Through Carry Left)
92      F = 5'b10110; A = 16'h8008; B = 16'h0001; Cin = 1;
93      #10;
94      // 20. F = 10111: RCR (Rotate Through Carry Right)
95      F = 5'b10111; A = 16'h8009; B = 16'h0001; Cin = 1;
96      #10;

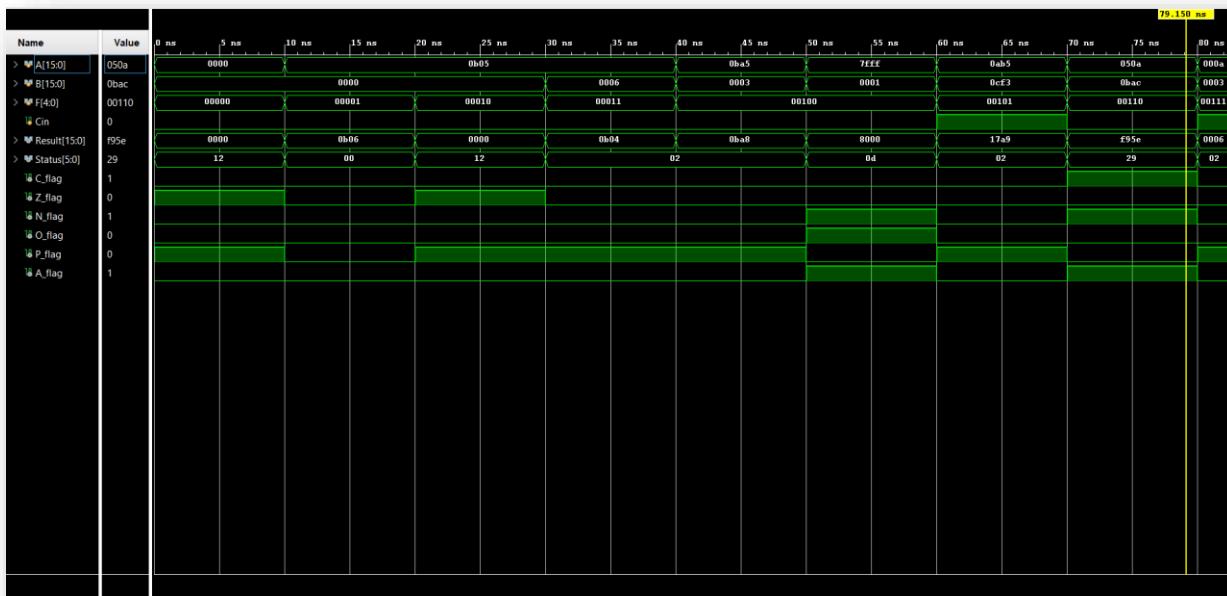
```

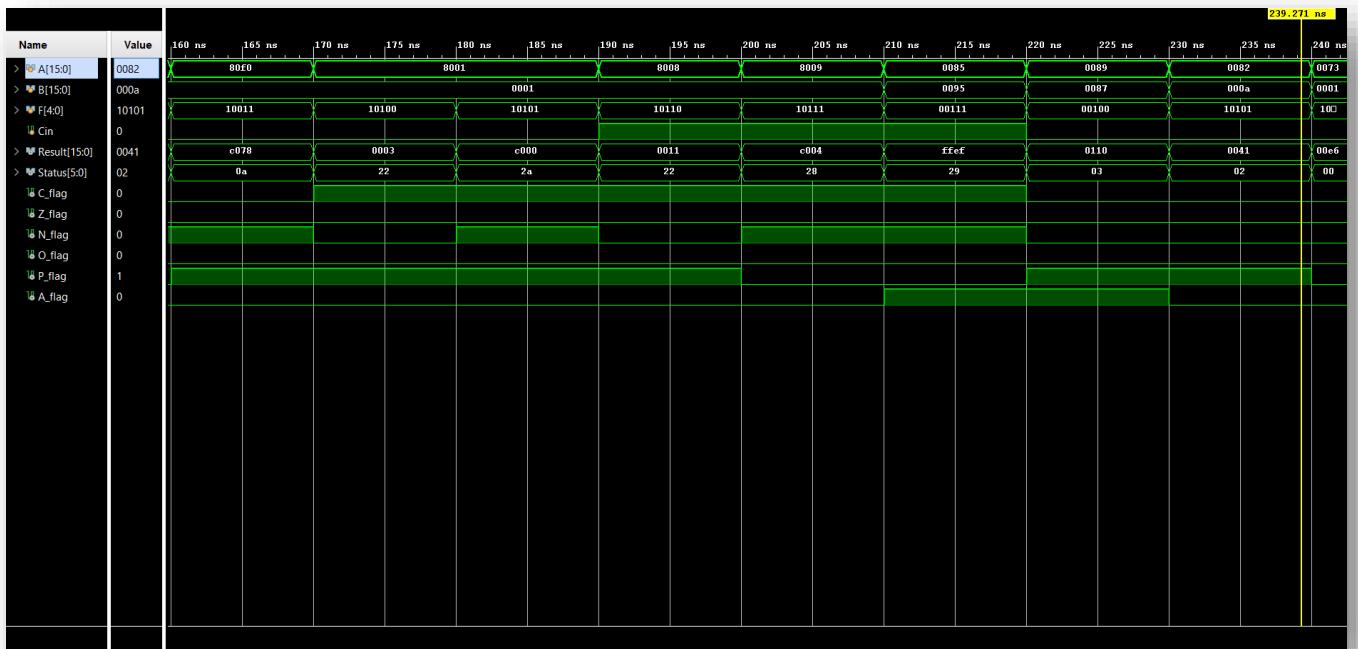
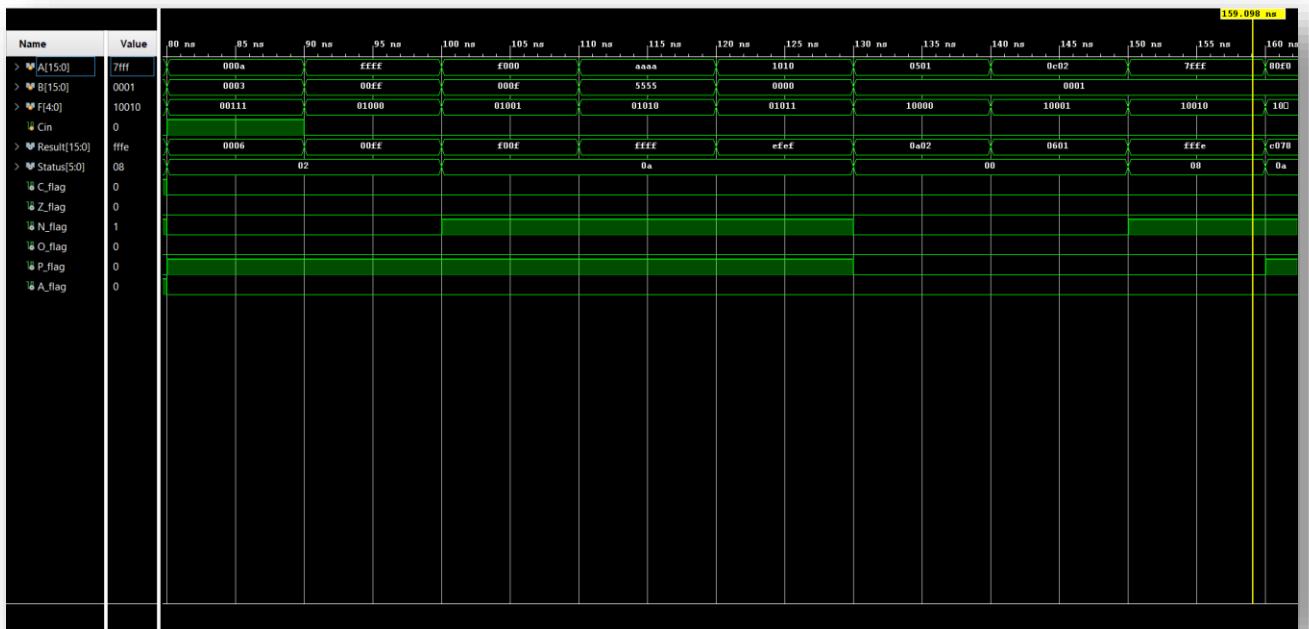
```

97      /*=====Sheet Cases=====*/
98      F = 5'b00111; A = 16'h0085; B = 16'h0095; Cin = 1;
99      #10;          // CASE 1: SBB AL, BL (AL=85, BL=95, CF=1)
100     F = 5'b00100; A = 16'h0089; B = 16'h0087; Cin = 0;
101     #10;          // CASE 2: ADD AL, 87 (AL=89)
102     F = 5'b10101; A = 16'h0082; B = 16'h000A; Cin = 0;
103     #10;          // CASE 3: ROR AL, CL (AL=82, CL=0A)
104     F = 5'b10010; A = 16'h0073; B = 16'h0001; Cin = 0;
105     #10;          // CASE 4: SAL AL, 1 (AL=73)
106     F = 5'b10111; A = 16'h0073; B = 16'h0001; Cin = 1;
107     #10;          // CASE 5: RCR AL, 1 (AL=73, CF=1)
108     F = 5'b00100; A = 16'h0006; B = 16'h0005; Cin = 0;
109     #10;          // CASE 6: ADD BL, 05 (BL=06)
110     F = 5'b01010; A = 16'h0055; B = 16'h00FF; Cin = 0;
111     #10;          // CASE 7: XOR AL, FF (AL=55)
112     F = 5'b00110; A = 16'h56BC; B = 16'h47CD; Cin = 0;
113     #10;          // CASE 8: SUB AX, BX (AX=56BC, BX=47CD)
114     F = 5'b00100; A = 16'h7FFF; B = 16'h3333; Cin = 0;
115     #10;          // CASE 9: ADD DX, CX (DX=7FFF, CX=3333)
116     F = 5'b01000; A = 16'h5555; B = 16'h5AAA; Cin = 0;
117     #10;          // CASE 10: AND AX, BX (AX=5555, CX/BX=5AAA)
118     $stop;
119   end
120
endmodule

```

7. Simulation





Name	Value
> A[15:0]	5555
> B[15:0]	5aaa
> C[40]	01000
Cin	0
> Result[15:0]	5000
> Status[5:0]	02
C_flag	0
Z_flag	0
N_flag	0
O_flag	0
P_flag	1
A_flag	0

Timing Diagram (310.000 ns):

- A[15:0]**: 5555, 0082, 000a, 10101, 0041, 02
- B[15:0]**: 5aaa, 0073, 0001, 10010, 00e6, 00
- C[40]**: 01000, 0006, 0005, 10111, 00100, 28
- Result[15:0]**: 5000, 0055, 00ff, 47c4, 00aa, 00
- Status[5:0]**: 00110, 3333, 00100, b332, 03, 0f
- Cin**: 0
- P_flag**: 1
- A_flag**: 0

8. GitHub

Link: <https://github.com/a7med57/16-Bit-ALU-.git>