BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY

CSE306 : COMPUTER ARCHITECTURE SESSIONAL

# 4-BIT ARITHMETIC AND LOGICAL UNIT

**Section: B1**
**Group: 04**
*Participating Rolls:*
1905065
1905067
1905069
1905076
1905077

*Submitted to*
Dr. Rifat Shahriyar
Md. Toufikuzzaman

December 21, 2022

# 1 Introduction

An arithmetic unit, or ALU, enables computers to perform mathematical operations on binary numbers. It is the part of a central processing unit that carries out arithmetic and logic operations on the operands in computer instruction words. As well as performing basic mathematical and logical operations, the arithmetic unit may also output a series of 'flags' which provide further information about the status of a result: whether it is zero, or there is a carry out, or an overflow has occurred. This is important as it enables a computational machine to perform more complex behaviors like conditional branching. It has two binary numbers and control bits as input where binary numbers are operands and control bits control the operation to be performed.

# 2 Problem Specification

We're tasked to construct a simplified version of ALU for computers. The ALU should have two 4-bit binary numbers as inputs as well as 3 control bits to define the operation. The output should be a 4-bit binary number which is the result of the operation(defined by the control bits) performed on the two 4-bit input binary numbers. The output should also contain a carry flag, overflow flag, sign flag and zero flag. Flags will be affected as per the rules of Assembly Language. Our ALU is required to output flags under a simplified special rule.
For OR and XOR Operation:

- $C$ and $V$ should be cleared (0) after the operation.
- $S$ and $Z$ should be changed according to the output.

The ALU is allowed to use only 2 input SSI (AND, OR, NOT, XOR, etc.) and MSI (MUX, Decoder, Adder, etc.). We're required to build the ALU with the minimum possible number of ICs.
For simulation, the use of any circuit analyzing software (e.g., logisim) is permitted. And circuit design in software must be at the IC level.
The simplified ALU we're tasked to construct is summarized by this table:

| $cs_2$ | $cs_1$ | $cs_0$ | **Operation** |
|---|---|---|---|
| 0 | 0 | 0 | Addition |
| 0 | 0 | 1 | Transfer A |
| X | 1 | 0 | Sub with Borrow |
| 0 | 1 | 1 | OR |
| 0 | 0 | X | Increment A |
| 1 | 1 | 1 | XOR |

## 3 Design Steps

Firstly, we define the four bits of operand A and B as $A_4A_3A_2A_1$ and $B_4B_3B_2B_1$ respectively, where $A_4$ and $B_4$ are the most significant bits of $A$ and $B$ respectively. Similarly, $cs_2, cs_1, cs_0$ are defined as selector bits, deciding which operations are being carried out. Hence, there are 11 inputs in total.

The operations that we are intending to carry out can be categorized into two types: Logical and Arithmetic. We can observe here that two operations; namely 4-bit XOR (111) and 4-bit OR (011) are logical operations and the remaining four operations are arithmetic.

- 4-bit OR: We used a quad 4-bit 7432 IC where we inserted four bits of operand-A and four bits of operand-B, where the 4-bit output will be evaluated accordingly.
- 4-bit XOR: We used a quad 4-bit 7486 IC where we inserted four bits of operand-A and four bits of operand-B, where the 4-bit output will be evaluated accordingly.

Now, we have 2 sets of 4-bit outputs, and we need to decide between them. For this, we need a quad 2x1 74157 MUX, we denote it with MUX1. We determined the equation for the select bit for M1 using K-map in section 4.3.

Next there are four arithmetic operations. If we express all operations using only addition operation, then the equations are:

- Addition
- Subtraction with Borrow
- Transfer
- Increment

It is evident that, if we try to further categorize these four operations into two types, the obvious criterion would be based on what the second operand is i.e., whether the second operand is simply zero (transfer and increment) or some form of input (addition and subtraction with borrow). This can be done using another 74157 IC, we denote it by MUX2. The select bit for MUX2 shall be discussed later.

The difference between addition and subtraction with borrow is, the second operand in addition is the unfiltered 4-bit user input $B_4B_3B_2B_1$ but for subtraction with borrow, all four bits of the second operand must be inverted. We can execute this using a 7486 IC, where one of the inputs are the four bits of $B$, and the other input should be 0 when the operation is addition, and 1 when the operation is subtraction with borrow. This bit has been determined using K-map in Section 4.1. We shall denote the 4-bit output as $D$ for the rest of the document.

For both Transfer and Increment, if we want to execute them using a 4-bit adder, the second operand in both can be zero, and the Carry-In bit determines whether it is a Transfer (0) or an Increment (1) operation.

Now, after filtering all input bits according to the select bits, we shall fit all Arithmetic operations into a single 7483 four-bit full adder IC. The first operand of the Adder should be the same as the input operand $A$. The second operation can be of two types: zero or $D$. To determine which one to select, we now bring the MUX no.2. The select bit for MUX2 has been determined using K-map in Section 4.4.

Finally for arithmetic operations, we need to determine the carry-in bit. This was

determined using K-maps in Section 4.2.

Thus, at this point, we have two sets of output determined: Logical and Arithmetic. We shall use one final 74157 IC to select the final output. We denote this MUX as MUX3. The select bit for MUX3 has been determined using K-map in Section 4.5.

Thus, we constructed the circuit for the final 4-bit output F. We were also instructed to construct to determine four flag bits: Carry bit (C), Signed Overflow bit (V), Unsigned Overflow bit (S) and Zero (Z).

- **Unsigned Overflow flag (S)**: this was simply done by connecting $F_4$ to the led for S flag.
- **Zero flag (Z)** This flag must set when all four bits of F are zero and must be unset at all other cases. Hence, the equation for Z flag is simply:
  $Z = \overline{F_4 + F_3 + F_2 + F_1}$
  To implement this, we used a 7432 to OR all bits of F, and then invert the result using XOR operation (since, $A \oplus 1 = \overline{A}$ ).
- **Signed Overflow flag (V)**: We can determine the equation for V flag from existing intermediate results that we have already implemented, which has been done using K-maps in Section 4.6. After implementing this equation, we must ensure that V is turned off whenever the select bits are set for Logical Operations. For this, we AND-ed the inverse of MUX3 select bit and the result, and sent it to the led for V.
- **Carry flag (C):** Our 7483 already displays the Carry-out bit of our given inputs, all we need to ensure is C flag is turned off whenever the select bits are set for Logical Operations. Due to AND gate shortages, we implemented this by inserting the inverse of M3 select bit as $V_{cc}$ of the 7483 IC. In this way, whenever the select bits are set for Logical Operations, all outputs of 7483 circuits are set to zero, but works as expected otherwise, making the behavior of the C flag exactly what we had desired.

Therefore, we shall obtain 8 output bits in total.

# 4 Karnaugh Maps

| Input Selector Bits | | | Desired Output | | | | |
|---|---|---|---|---|---|---|---|
| $cs_2$ | $cs_1$ | $cs_0$ | $XOR_B$ | $C_{in}$ | $S_1$ | $S_2$ | $S_3$ |
| 0 | 0 | 0 | 0 | 0 | X | 1 | 0 |
| 0 | 0 | 1 | X | 0 | X | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | X | 1 | 0 |
| 0 | 1 | 1 | X | X | 0 | X | 1 |
| 1 | 0 | 0 | X | 1 | X | 0 | 0 |
| 1 | 0 | 1 | X | 1 | X | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | X | 1 | 0 |
| 1 | 1 | 1 | X | X | 1 | X | 1 |

We shall use the following convention for all 3 bit K-maps hereby:

| | | | |
|---|---|---|---|
| $\overline{cs_2}cs_1\overline{cs_0}$ | $\overline{cs_2}\overline{cs_1}cs_0$ | $\overline{cs_2}cs_1cs_0$ | $\overline{cs_2}cs_1\overline{cs_0}$ |
| $cs_2\overline{cs_1}\overline{cs_0}$ | $cs_2\overline{cs_1}cs_0$ | $cs_2cs_1cs_0$ | $cs_2cs_1\overline{cs_0}$ |

## 4.1 Second Operand for XOR-B

| | | | |
|---|---|---|---|
| 0 | $X$ | $X$ | 1 |
| $X$ | $X$ | $X$ | 1 |

$$c = cs_1 \tag{1}$$

## 4.2 Carry In

| | | | |
|---|---|---|---|
| 0 | 0 | $X$ | 0 |
| 1 | 1 | $X$ | 0 |

$$C_{in} = cs_2 \cdot \overline{cs_1} \tag{2}$$

### 4.3 Selector for MUX1

| | | | |
|---|---|---|---|
| $X$ | $X$ | 0 | $X$ |
| $X$ | $X$ | 1 | $X$ |

$$S_1 = cs_2 \tag{3}$$

### 4.4 Selector for MUX2

| | | | |
|---|---|---|---|
| 1 | 0 | $X$ | 1 |
| 0 | 0 | $X$ | 1 |

$$S_2 = \overline{cs_1} \cdot (cs_0 + cs_2) \tag{4}$$

### 4.5 Selector for MUX3

| | | | |
|---|---|---|---|
| 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 |

$$S_3 = cs_0 \cdot cs_1 \tag{5}$$

### 4.6 Signed Overflow Flag

Original equation for $V$ flag is $V = C_4 \oplus C_{out}$

| $C_4$ | $A_4$ | $B_4$ | $C_{out}$ | $E_4$ | $V$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 |

Constructing the K-map for $V$ with respect to on $A_4, B_4, F_4, C_{out}$:

| | | | |
|---|---|---|---|
| 0 | $X$ | 0 | 1 |
| $X$ | 0 | $X$ | 0 |
| 0 | 1 | 0 | $X$ |
| $X$ | 0 | $X$ | 0 |

Which is compatible with the K-map for $A_4 \oplus B_4 \oplus F_4 \oplus C_{out}$ :

| | | | |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |

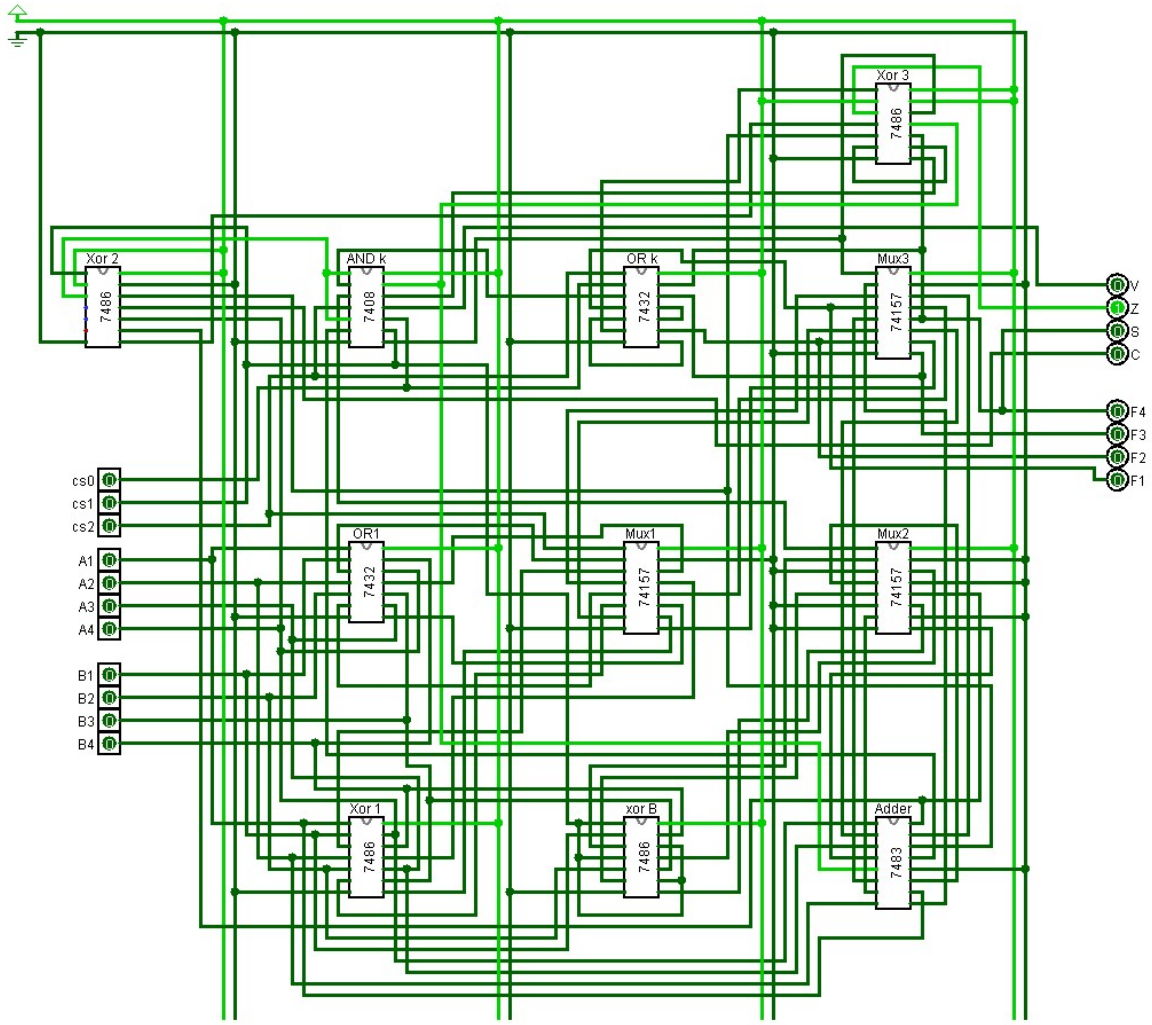
Hence, we can use $V = A_4 \oplus B_4 \oplus F_4 \oplus C_{out}$

## 5 Truth Table

| **Input Selector Bits** | | | **Operation** | **Symbolic** |
|---|---|---|---|---|
| $cs_2$ | $cs_1$ | $cs_0$ | | |
| 0 | 0 | 0 | Addition | $A+B$ |
| 0 | 0 | 1 | Transfer A | $A$ |
| 0 | 1 | 0 | Subtraction with Borrow | $A+\overline{B}$ |
| 0 | 1 | 1 | OR | $A\|B$ |
| 1 | 0 | 0 | Increment A | $A+1$ |
| 1 | 0 | 1 | Increment A | $A+1$ |
| 1 | 1 | 0 | Subtraction with Borrow | $A+\overline{B}$ |
| 1 | 1 | 1 | XOR | $A \oplus B$ |

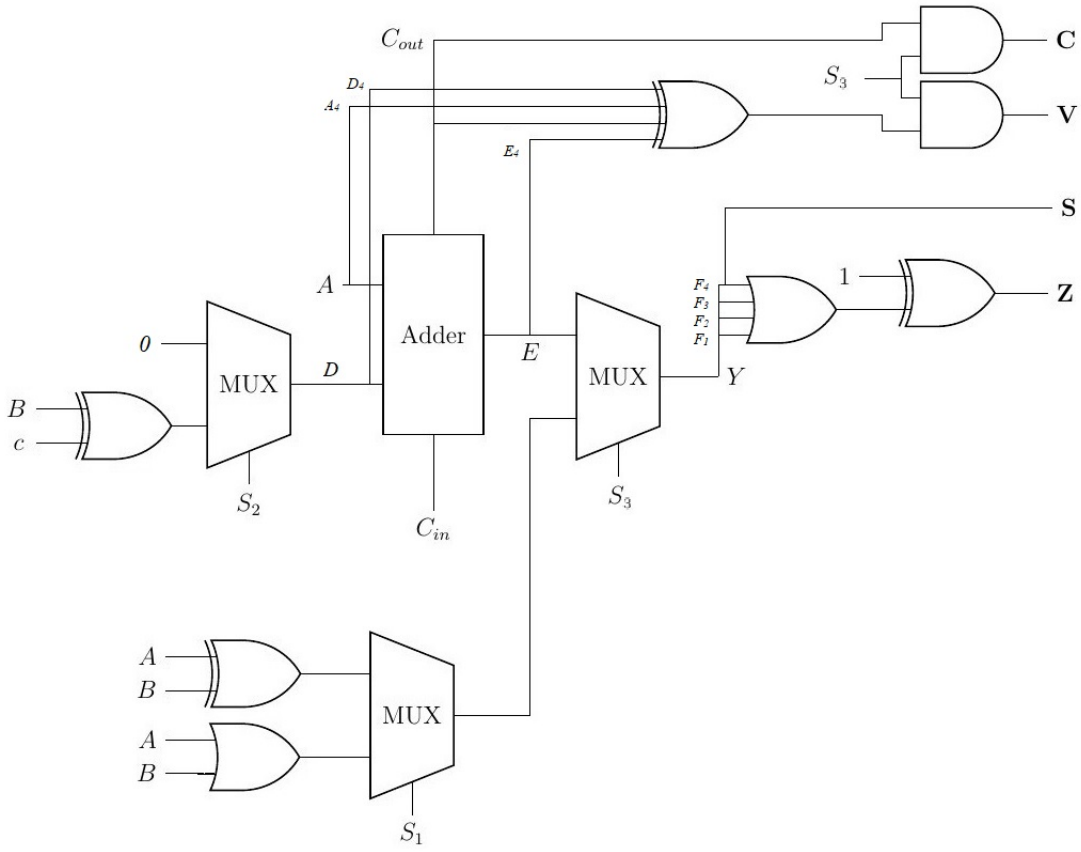
# 6 Diagrams

## 6.1 Circuit Diagram




Circuit Diagram

### 6.2 Block Diagram




Block Diagram

## 7 Tools and Apparatus

### 7.1 Integrated Circuits

| IC Number | IC Name | Count |
|---|---|---|
| SN74HC08N | Quad 2-Input AND | 1 |
| SN74HC32N | Quad 2-Input OR | 2 |
| SN74HC86N | Quad 2-Input XOR | 4 |
| SN74HC83N | 4bit Binary Full Adder | 1 |
| SN74HC157N | Quad 2 to 1 MUX | 3 |

### 7.2 Simulator

Software: Logisim
Version: logisim-win-2.7.1

## 8 Discussion

1. Constant power was supplied using Arduino.
2. Blue LEDs were used for 4 bits of A inputs and 4 bits of outputs F.
3. Green LEDs were used for 4 bits of B inputs and 4 flags.
4. Cis Blinking LEDs (Red-Blue) were used for 3 selector bits.
5. Jumper wires were used for all 4 bit transfers. For all other cases, professional breadboard wires were used.
6. Left pin of the switch was connected with power, right pin was connected with ground. Output power was taken from the middle pin. 1K resistors were used while supplying output powers to input LEDs from the switches, in order to protect the LEDs from damaging.
7. In every breadboard, the upper horizontal shorted lines were used for supplying Power and the lower ones were used as Ground.
8. During logical operations, it was instructed that the C and S flags should be turned off. In order to turn off C and S flags, the inverse of MUX-3 select bit was inserted as Vcc of the 7483 Adder IC. So, whenever the select bits are set for Logical Operations, all outputs of 7483 circuits are set to zero, but works as expected otherwise, making the behavior of the C flag exactly what had been desired.
9. All of the 11 ICs and breadboards were carefully checked before using them in the circuit to ensure that all of those were working perfectly.
10. The outputs were checked multiple times for a large number of input test cases to ensure that our circuit satisfies the expected output values from the truth table.
11. Wires having different sizes were used to make the whole circuit organized and easily understandable for the user.