**CSE 306**

**Computer Architecture Sessional**

**Assignment-1: 4 bit ALU Simulation**

**Section-B2**

**Group-02**

**Members of the Group:**

**i.2005092- Shahad Shahriar Rahman**

**ii.2005094-Golam Mostofa**

**iii.2005099-Maisha Maksura**

**iv.2005105-Mohammad Ishrak Adit**

**v.2005108- Kazi Redwan Islam**

**A. Introduction**

As an implementation of computational and logical operations in a shared circuit, the Arithmetic Logic Unit (more commonly known as ALU), is embedded within the central processing unit of any computing device. Operated with control signals to instruct an exact operation to follow, the network is made to produce the outcome along with additional information regarding both operands and the effect.

ALU provides basic arithmetic assistance, namely, addition, subtraction, increment, decrement, or transfer of limited bit inputs, and is also able of logical operations like AND, OR, XOR, NOT, and likely so. Any operation can be requested with a predetermined sequence of selection signals and an n-bit output shows the result of the operation on two n-bit inputs given as required, n being any natural number.

Additional data bits are produced by the circuit, which help to analyze the result. These bits are called flags. Carry flag(C) shows if an operation produces a carry. The overflow flag(V), if set, indicates that the result has exceeded the maximum representable value with the limited bits. C and V vary upon the operands of arithmetic operations, but have no value for logical operations and are cleared.

The zero flag(Z) is set only when all output bits are zero after an operation, otherwise, it's reset. Sign flag(S) reflects the MSB and helps analyze signed operations.

An account of the flags can detect the relation between the operands regarding a particular operation.

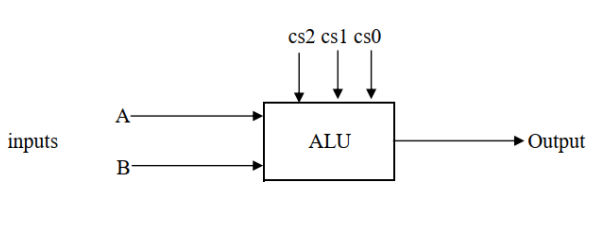
An ALU is built around parallel adder circuits, with supplementary components utilized as necessary. The interconnection between the components is calculated through the analysis of the operations required.

This report includes a description of our attempt at assembling a simplified 4-bit ALU, designed for certain assigned procedures. The circuit is prepared for 4 arithmetic and 2 logical operations.

**B. Problem Specification with Assigned Instructions**

A fully functional 4-bit ALU (Arithmetic Logic Unit) has to be implemented with an efficient design (minimum possible ICs) according to the requirements given in the table of the supporting operations. Additionally, four flags namely Carry (C), Sign (S), Overflow (V), and Zero (Z) have to be designed as well per the rules of Assembly Language with some modifications and exceptions proposed to incorporate flexibility.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Control Signals | | | Function Name | Description |
| cs2 | cs1 | cs0 |
| X | 0 | 0 | Add | A + B |
| 0 | 0 | 1 | Add with Carry | A + B + 1 |
| X | 1 | 0 | XOR | A ⊕ B |
| 0 | 1 | 1 | NEG A | A' + 1 (i.e. - A) |
| 1 | 0 | 1 | Increment A | A + 1 |
| 1 | 1 | 1 | AND | A ∩ B |



* **For NOT Operation:**

Z needs to be 1 or unchanged if NOT operation results in 0000. S remaining unchanged or reflecting the highest order of bit of the result is acceptable. C and V has to be considered as Don’t Care.

* **For AND Operation:**

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

**C. Detailed design steps with k-maps**

1. As we are implementing both arithmetic and logical operations through one single circuit block, we must manipulate the input numbers keeping both types of conditions in mind.

2. We will be given two 4-bit binary numbers and have to the find results of corresponding arithmetic and logical operations. So, first we’ll take two 4-bit inputs, A and B, and transform them to X and Y respectively for each bit, and Cin to Z using some manipulators to meet the conditions. The manipulators are in fact some combinations of the selection bits given.

3. Following the problem specification chart, we can find the general equations of Xi , Yi and Zi using k-maps. As our planned way includes decoder in the circuit, we may have to modify the equations as needed.

4. After being transformed, each bit of two inputs along with corresponding Cin will go into the adder block and will generate the result of the operation.

5. Using the result bits and Carry bits of multiple steps from the adder block, we’ll then generate Carry Flag (C), Sign Flag (S), Overflow Flag (V), and Zero Flag (Z).

K-maps:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **s2** | **s1** | **s0** | **Operation** | **Xi** | **Yi** | **Zi** |
| X | 0 | 0 | Add | A | B | 0 |
| 0 | 0 | 1 | Add with carry | A | B | 1 |
| X | 1 | 0 | XOR | A | B | 0 |
| 0 | 1 | 1 | NEG A | A’ | 0 | 1 |
| 1 | 0 | 1 | Increment A | A | 0 | 1 |
| 1 | 1 | 1 | AND | A+B’ | B’ | 0 |

Xi:

Let, Xi=m1Ai+m2Ai’+m3Bi’

m1: s1s0

s2 00 01 11 10

0

0 0

m1 = s2+s1’+s0’

1 = (s2’ s1 s0)’

m2:

s1s0

s2 00 01 11 10

1

0 0

m2 = s2’ s1 s0

1

m3:

s1s0

s2 00 01 11 10

0 0

m3 = s2 s1 s0

1

1

So, Xi = (s2’ s1 s0)’ Ai + (s2’ s1 s0) Ai’ + ( s2 s1 s0 ) Bi’

= ( Ai (+) (s2’ s1 s0)) + ( s2 s1 s0 ) Bi’

Yi:

Let, Yi = n1Bi (+) n2

n1:

s1s0

s2 00 01 11 10

1

0 0

1

1

1 n1 = s0’ + s2’s1’s0

1

1

We didn’t merge cells because we can get the minterm directly from decoder

n2:

Yi will be Bi’ only when s2 s1 s0 = 1

So, n2 = s2 s1 s0

And for this case, so that we get Yi = Bi (+) 1 , n1 will be n1 = s0’+s2’s1’s0+s2 s1 s0

So, Yi = (s0’+s2’s1’s0+s2 s1 s0) Bi  (+) s2 s1 s0

Zi:

Let, Zi = p Cin

For only XOR and AND operation, Zi = 0 , and for the other cases Zi  = Cin

So, for XOR and AND operation, p = 0, otherwise p = 1

p:

s1s0

s2 00 01 11 10

0 0

1

1

1

1

1

1

To get the minterms directly from decoder and to minimize total ICs, we’ll not merge the cells here.

p= s2’ s1’s0’ + s2’ s1’s0 + s2’ s1 s0 + s2 s1’s0’ + s2 s1’s0

= m0 + m1 + m3 + m4 + m5  (expressing in minterms)

Z0 = p s0

Zi>0 = p Cout i-1

**D. Truth Table**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **cs2** | **cs1** | **cs0** | **Cin** | **Function** | **Xi** | **Yi** | **Zi** | **Cout** |
| 0 | 0 | 0 | 0 | ADD | A | B | X | X |
| 0 | 0 | 1 | 1 | ADD with Carry | A | B | X | X |
| 0 | 1 | 0 | 0 | X-OR | A | B | 0 | 0 |
| 0 | 1 | 1 | 1 | NEG | A’ | 0 | X | X |
| 1 | 0 | 0 | 0 | ADD | A | B | X | X |
| 1 | 0 | 1 | 1 | INC | A | 0 | X | X |
| 1 | 1 | 0 | 0 | X-OR | A | B | 0 | 0 |
| 1 | 1 | 1 | 0 | AND | A+B’ | B’ | 0 | 0 |

Table 1: Truth Table

**E. Block Diagram**

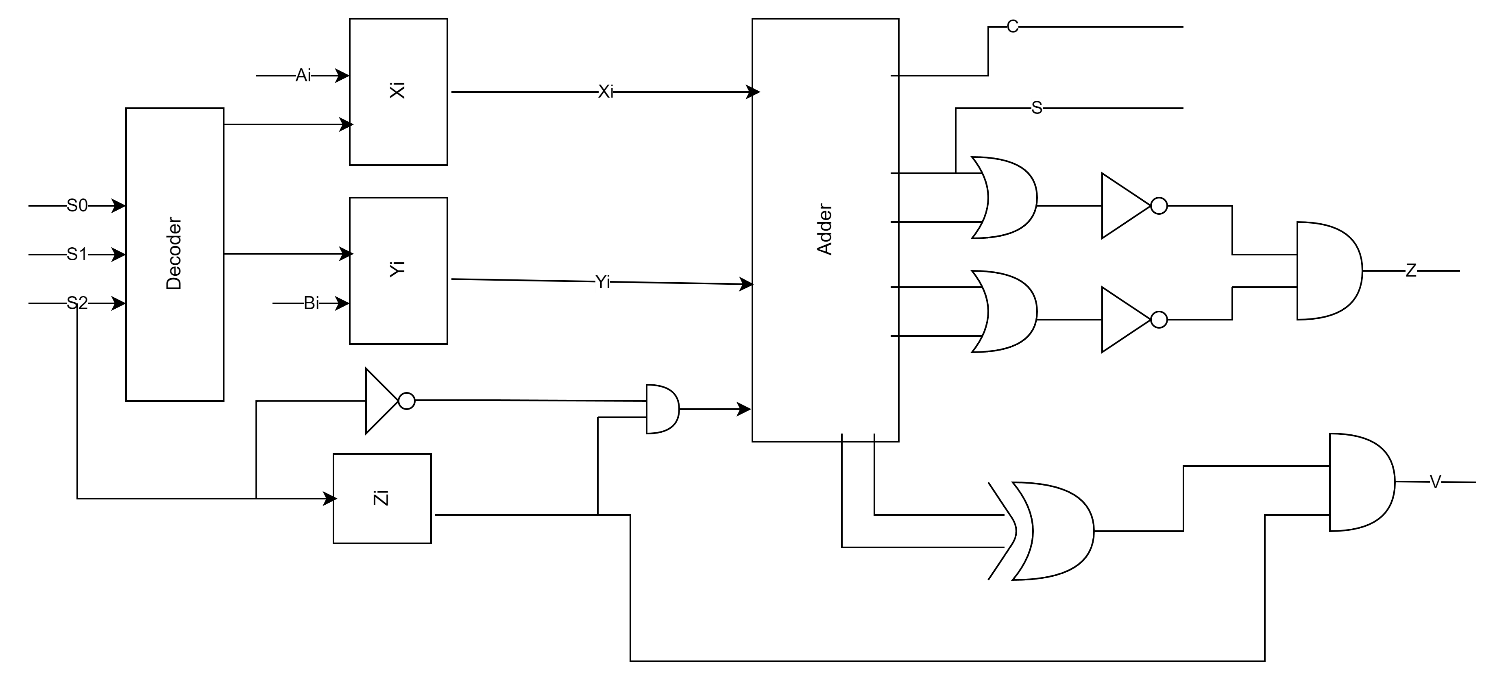
****

Figure :Block Diagram

**F. Complete Circuit Diagram**

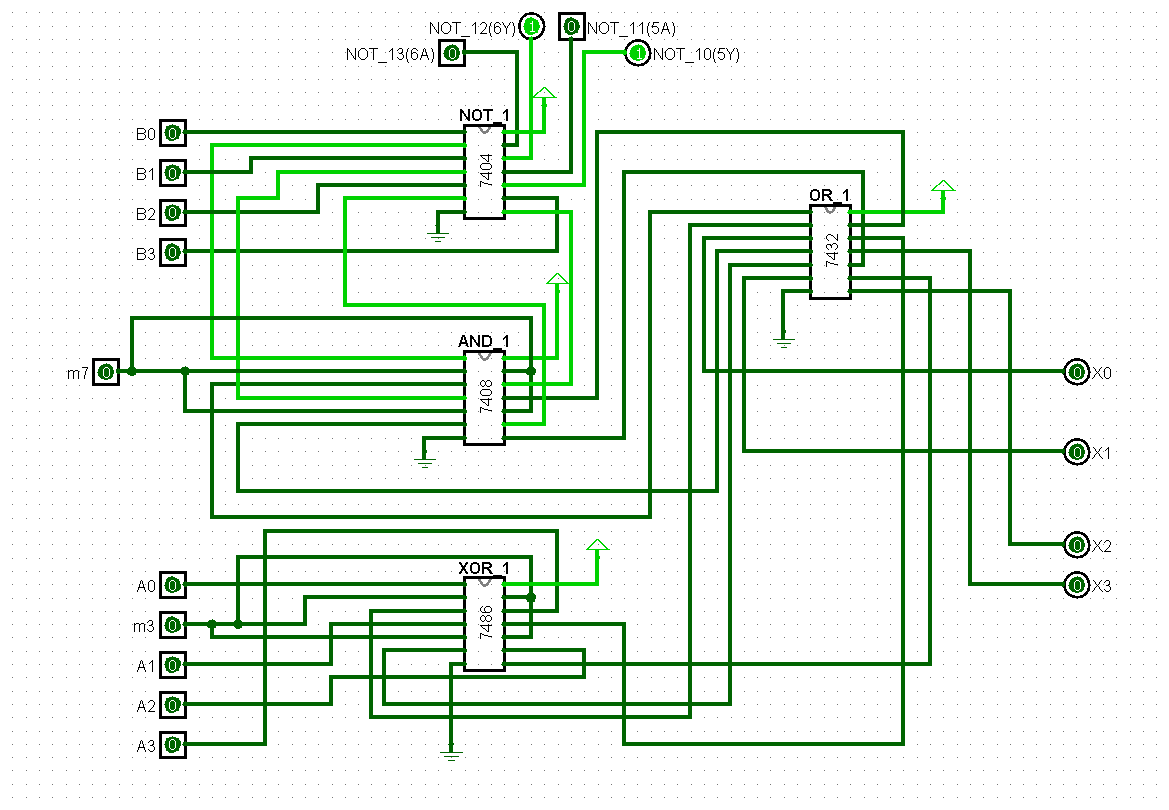


Figure 2: Circuit for Xi

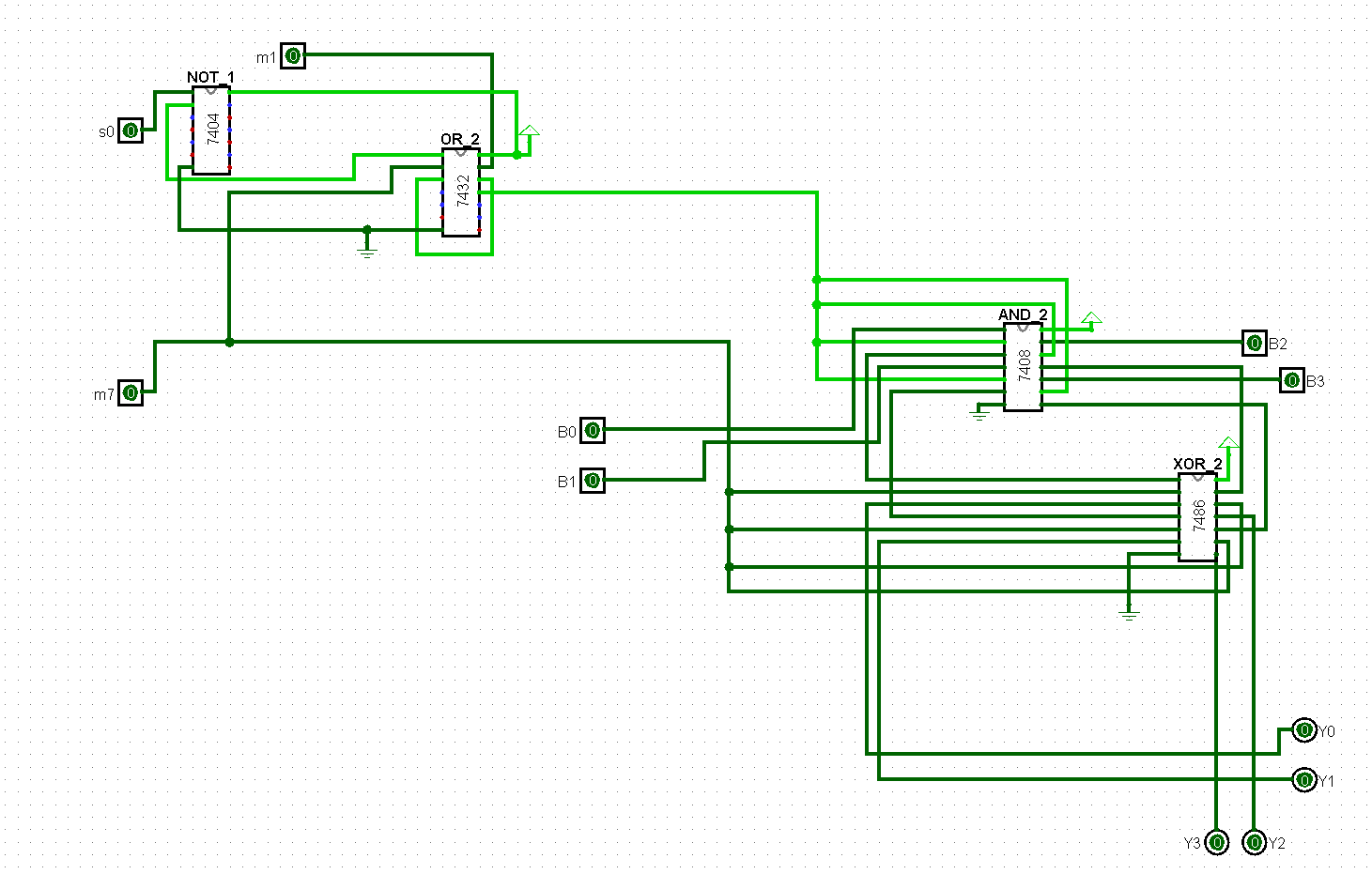


Figure 3: Circuit for Yi

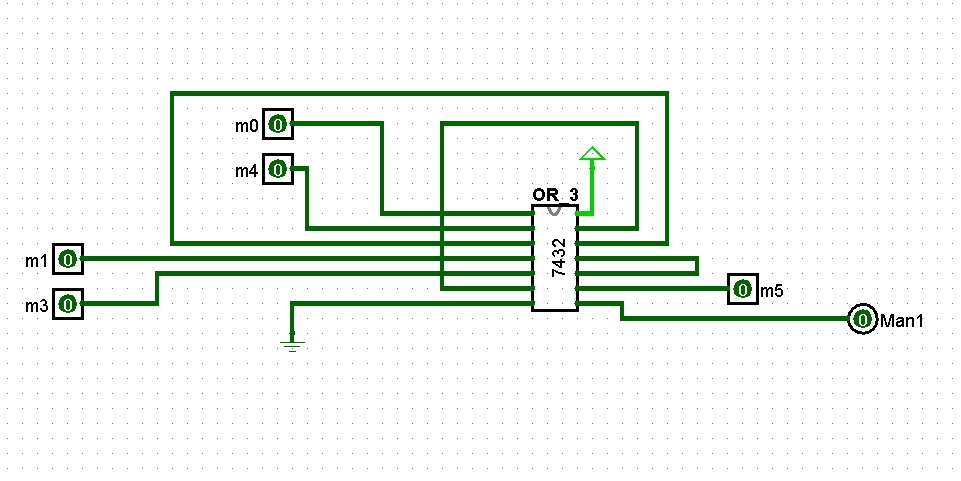


Figure 4: Circuit for Zi

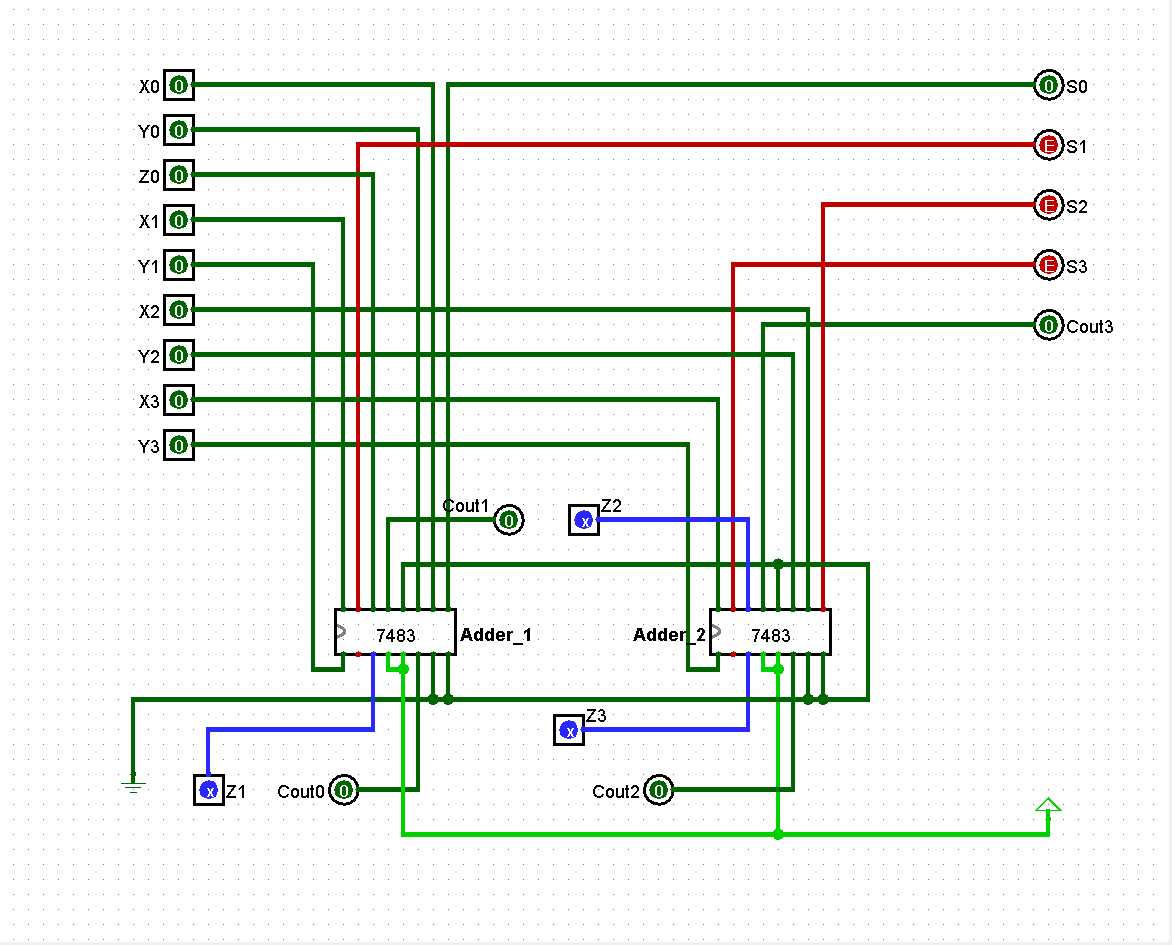


Figure 5: Adder Circuit

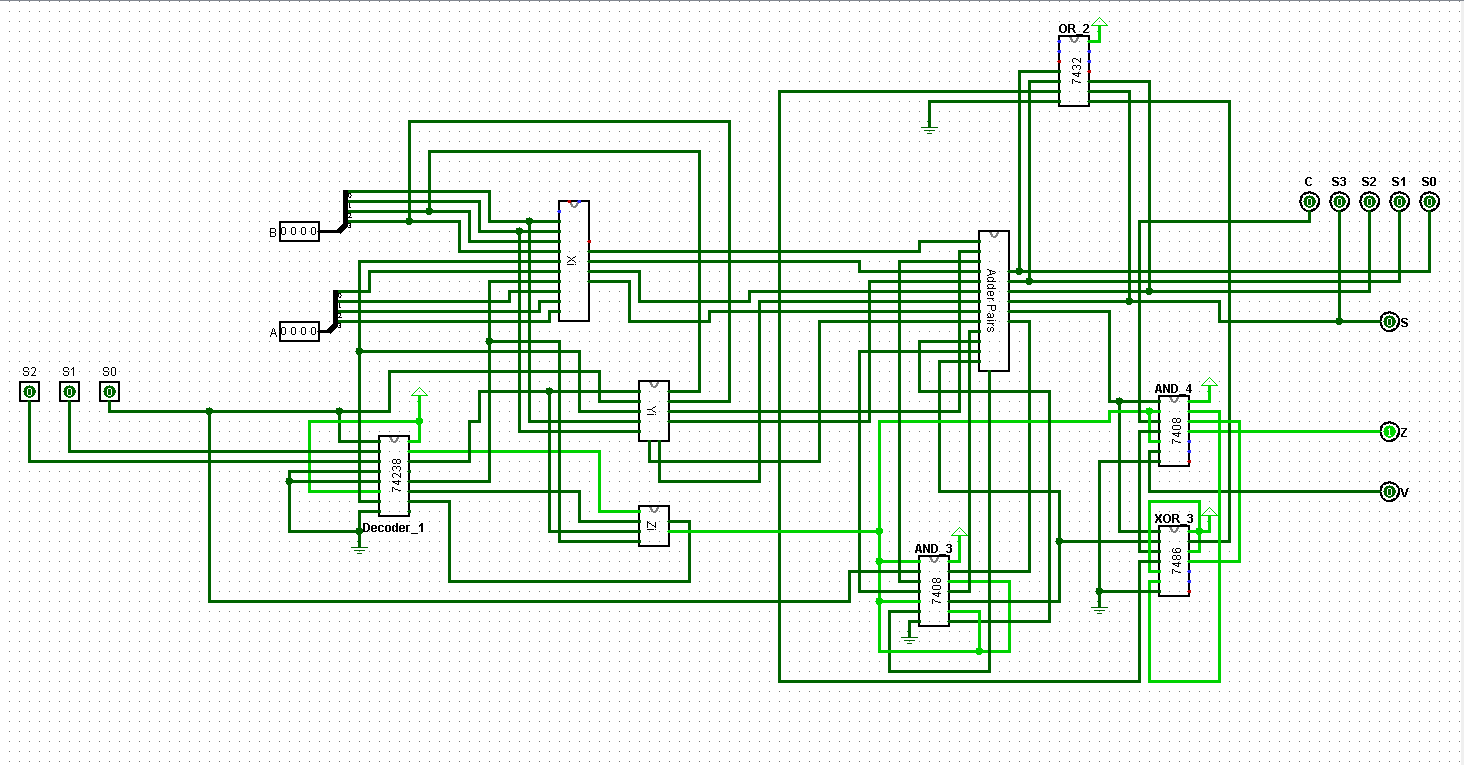


Figure 6: The ALU

**G. ICs Used with Count as A Chart**

|  |  |
| --- | --- |
| **IC** | **Quantity** |
| **IC 7404** | **1** |
| **IC 7408** | **4** |
| **IC 7432** | **3** |
| **IC 7486** | **1** |
| **IC 7483** | **2** |
| **IC 74238** | **1** |
| **Total** | **14** |

**H. The simulator used along the version number:**

Logisim Generic version 2.7.1

**I. Discussions**

An ALU (Arithmetic Logic Unit) is a combinational digital circuit that performs arithmetic and bitwise operations on integer binary numbers. The provided assignment instructs us to design and implement a 4-bit efficient ALU circuit that can perform certain arithmetic and logical operations based on the combinations of selection bits.

Efficiency has always been a priority while designing every part of the circuit. Necessary boolean expressions and equations have been derived and simplified using truth tables and K-maps. Since we were told to deliver an IC-level design, our derived boolean expressions were employed with a minimum number of ICs.

Using a 3X8 Decoder allowed us to access all the necessary boolean minterms the selection bits can offer. Certain techniques were followed to access and manipulate the carry bits of the 4-bit parallel adder. To utilize the deployed ICs to a full extent, we had to come up with some non-conventional methods such as attaining the invert of a bit by XORing it with 1 instead of NOTing it. Our model was well-designed and it ensured a proper division of labour. Every member of the team implemented their respective sub-circuits on LOGISIM and later they were merged to form a fully functional circuit.

Before the hardware implementation, every IC, wire, LED, and switch was checked in case it showed faulty behavior. Discussions were done before the implementation for the proper placement of equipments. The ICs were placed in a manner that resulted in a low number of crossing wires.

The wires were connected following the pin diagram of the ICs. Moreover, the ground and power connections were double-checked. Despite the challenges faced, we were able to keep our design tidy with the help of Sticky Notes. Necessary resistors were used to control current flow through LEDs. The LEDs were connected maintaining their polarity.

Through inspection, intuition, and some reverse engineering, we examined the correctness of the output of our ALU circuit.

**J. Contribution**

|  |  |
| --- | --- |
| **Student ID** | **Contribution** |
| 2005092 | Designed the adder subcircuit. |
| 2005094 | Jointly designed Xi subcircuit. |
| 2005099 | Optimized the design. Designed Zi subcircuit. |
| 2005105 | Designed Yi subcircuit. |
| 2005108 | Jointly designed Xi subcircuit & Decoder subcircuit. |