1 Introduction

First proposed by John Von Neumann, ALU stands as a cornerstone in the realm of computer architecture ever since its coinage in 1945. With diminishing transistor dimensions, implementing and incorporating wider-word ALUs in tiny microprocessors has kept becoming easier and cheaper. In the light of our own experimentation, this report aims at exploring the strategies of ALU design in considerable depth. We first provide a modest introduction to the theoretical foundation. Afterwards we present our methodology, outcomes and scopes of further exploration.

1.1 Definiton

An arithmetic logic unit abbreviated as ALU is a combinational digital circuit that performs arithmetic and bitwise operations on integer binary numbers.

1.2 Overview

Figure 1 presents a block diagram of a 4-bit ALU which takes two 4 bit numbers A($A_3A_2A_1A_0$) and B($B_3B_2B_1B_0$) as inputs and 3 control selection bits cs_2, cs_1, cs_0. After performing the operation, it outputs 5 bits namely, $C_{out}, S_3, S_2, S_1, S_0$ Additionally, it updates the 4 flags named S, C, V, Z in the Flags register accordingly.

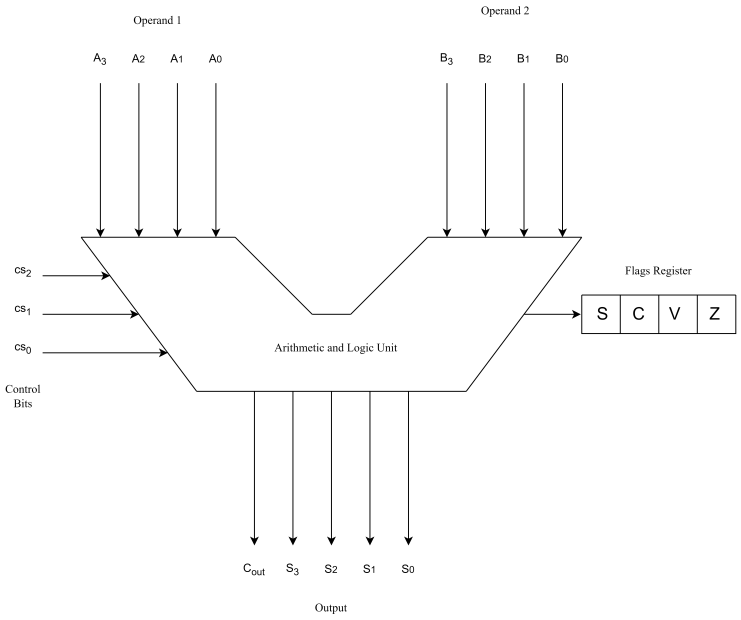

Figure 1: Block Diagram of 4 bit ALU

To generalize, the inputs for an n-bit ALU are two n-bit binary numbers and k control selection bits, where the value of k is design specific. And it outputs an n-bit binary number S and one output carry. Alongside these, it calculates the 4-flags and store them in the Flags register.

1.3 Components

1.3.1 Control Selection

A number of mutually independent distinct input bits are used as control selection bits which determine the nature of the operation to be performed on the operands. Sometimes, one of these bits, usually the most significant one, is used solely to distinguish between arithmetic and logical operations. However, cases depending upon the requirements may well appear where one single bit is not adequate to meet this purpose.

Each of the 2^k combinations of the possible inputs in the control bits is mapped to a single operation defined on the operands. It is, however, permissible to have multiple combinations of selection bits to map to the same operation.

1.3.2 Operands

Two independent operands usually consisting of the same number of bits are fed into the ALU through circuitry. All operations determined by the control selection bits are applied on these two operands.

For operations which do not impose symmetry upon the operands, for instance, subtraction, order of the operation matters. In this case, usually subtraction means subtraction of B from A.

For bitwise logical operations, the corresponding action is performed bit by bit and the output is also generated bit by bit and no two distinct bits affect others' outputs.

Operands are not usually used directly into the circuitry. Most often, each bit of an operand goes through identical circuitry and thus produces the corresponding input bits which need to be passed into the full adder.

1.3.3 Arithmetic Unit

Full adder, although usually interpreted as a circuitry for adding two n-bit numbers and generating the summation with output carry, can be utilized in multifaceted ways, thereby allowing a designer to efficiently perform seemingly disjoint arithmetic operations by intelligently manipulating the different input bits of a full adder. Regarding the full adder, one significant consideration is that, ideally the full-adder must include an internal carry-lookahead mechanism, Otherwise race conditions may appear and may pose to be an impediment towards accurate calculations.

Common arithmetic operations include transfer, addition, addition with carry, increment, decrement, subtraction, subtraction with borrow, negation (2's complement). The mathematical interpretation of the aforementioned operations is presented in the table below:

Name of operation	**Equation**
Transfer A	A
Addition	A + B
Addition with Carry	A + B + 1
Subtraction	A + B' + 1 (A - B)
Subtraction with borrow	A + B' (A-B-1)
Negation	A' + 1 (-A)

Table 1: Arithmetic Operations

1.3.4 Logic Unit

Bitwise logical operations on n-bit operands is just as important in terms of computation as are the arithmetic ones. These operations include bitwise AND, OR, Exclusive-OR (XOR) and One's complement. But depending on requirements, more complex logical operations may as well be needed. The calculations of the logic unit may either be carried out in a separate circuit or may be incorporated into the arithmetic unit after careful manipulations. In the former case, additional circuit elements need to be introduced to merge the two units and to run the appropriate unit for respective selection of control bits.

Alongside bitwise logical operations, another type of bitwise operation named Logical Shift is often required. This may involve right shift, left shift or bit rotation. Whether this operation will be implemented inside a separate unit or inside the ones mentioned above is often a matter of design choice.

1.3.5 Output

Just as it happens in full adder, the output is generated as an n-bit number in the n distinct bits of the outputs of full adder. Additionally an extra bit named output carry (C_{out}) is generated. If overflow does not happen for the particular scenario, then the output corresponds to the result of the intended operation that is defined by the combination of the selection bits.

It is to note that, for some particular operations, the output is to be perceived after disregarding the output carry. For instance, since subtraction is carried out as 2's complement, the output, when C_{out} is considered, is actually equal to the desired result plus 2^n. Therefore to interpret the correct difference between the operands, one needs to ignore the final output carry.

1.3.6 Flags Register

Last but not the least, an ALU leaves valuable information regarding the overall nature of the computation in a register called Flags Register. Usually there are 4 flags that are updated accordingly after performing the computations on the operands. However, in case of some operations, some particular flags may as well need to be kept constant.

The details regarding the equation and significance of each flag is shown in the table below:

Name of Flag	**Equation**	**Significance**
Sign (S)	S_3	It denotes the most significant bit of the output from full adder. Thus this flag helps determine the sign of the output.
Carry (C)	C_{out}	It denotes the output carry.
Overflow (V)	$C_{out} \oplus C_3$	This flag indicates whether there occurred an overflow while performing the calculation. If this flag is set, it means that the apparent output is not equal to the intended result and corrective measures must be taken while interpreting the output.
Zero (Z)	$\overline{S_3 + S_2 + S_1 + S_0}$	It indicates whether all bits of the output are zero.

Table 2: Equation and Significance of Flags

1.4 Design Strategies

With varying requirements, design strategies may differ from each other by a significant amount. Additionally, not all design processes focus on optimizing the same parameter. However, to broadly classify, the design process may incorporate one of the following two prevalent patterns:

1.4.1 Disjoint Implementation

One simple approach involves separate implementation of arithmetic and logic unit circuits. In this case, the Arithmetic unit is carried out with a full adder and the logic unit calculates the bitwise operations with direct involvement of corresponding gates and multiplexers. Then, to merge these two circuits, an additional multiplexer is needed to differentiate and determine which circuit will be active for which combinations.

While this approach seems easy to understand and implement, the major drawback is that, while combining the two circuits together, it needs a number of unnecessary ICs which may exceed size and cost restrictions.

1.4.2 Unified Implementation

An alternative approach considers converting the logical operations to equivalent arithmetic operations and thereby performing them in the same circuit as of the arithmetic one. Although the design process tends to get a bit complicated in this approach, the tradeoff is well worth it when the reduction in the number of ICs is considered.

2 Problem Specification with Assigned Instructions

Design a 4-bit ALU with three control selection bits cs_0, cs_1, cs_2. The assigned instructions and corresponding combinations of control selection bits are shown in the table below:

Control Signals			Functions	Description
cs2	cs1	cs0		
0	X	0	Add	$A + B$
0	0	1	Neg A	$-A(\overline{A} + 1)$
0	1	1	Add with Carry	$A + B + 1$
1	0	0	Increment A	$A + 1$
1	0	1	AND	$A.B$
1	1	X	XOR	$A \oplus B$

Table 3: Problem Specification