Lab Report: Digital Logic

Introduction

The aim of the Digital Logic Lab was to construct a simple *4-bit Arithmetic Logic Unit (ALU)* in order to demonstrate methods of using *Boolean Algebra* to manipulate and solve various logic problems. An ALU is used as the basis of a microprocessor and enables the microprocessor to evaluate arithmetic expressions in binary.

The ALU was designed using a series of *gates*. The circuits were then simulated using a computer equipped with Altera's MaxPlus+II. This meant that the circuits could be constructed, tested and modified quickly without purchasing any real components.

This report will guide the reader through the Boolean algebra that was learnt and how it was used to design various parts of the ALU.

Theory

Boolean Algebra

Boolean Algebra was developed by George Boole in the 19th century in order to determine the truth or falsehood of logical prepositions. Boolean algebra has been especially successful in the Electronics and Computing fields because it can be used to model binary systems and the resulting equations can easily be implemented using *gates*.

1st Year Electronics Laboratory – Autumn Term, 2001

A more complete description of the Boolean Algebra and notation used in this report can be found in the 1st Year Electronics Laboratory – Autumn Term, 2001, but the most important parts will be explained here.

de Morgan's Theorem

$$\overline{?A?\ B?}=\overline{A}\cdot\overline{B}$$

$$\overline{?A\cdot B?}=\overline{A}?\ \overline{B}$$

Using these equations, it can be seen that any logical expression involving AND, NOT and OR can be rewritten using only AND and NOT or only OR and NOT.

$$\overline{A}?\ \overline{B}=\overline{?A\cdot B?}$$

The expression on the right of the equation is known as a NAND gate.

$$\overline{A}\cdot\overline{B}=\overline{?A?\ B?}$$

The expression on the right of the equation is known as a NOR gate.

This idea can be developed further by showing that AND operations can be performed using NAND and NOT gates;

$$A\cdot B=\overline{\overline{?A\cdot B?}}$$

and that NOT operations can be performed using a NAND gate;

$$\overline{?A\cdot A?}=\overline{A}$$

1st Year Electronics Laboratory – Autumn Term, 2001

In summary, any logical function involving AND, NOT and OR can be rewritten in terms of NAND or NOR gates. Therefore it is only necessary to implement a NAND or NOR gate in hardware to enable any logical function to be built. Most commonly, expressions are implemented using NAND gates instead of NOR gates.

Associative Law

$$A\cdot B\cdot C=?A\cdot B?\cdot C=A\cdot ?B\cdot C?$$

Distributive Law

$$A ?B? C?=?A \cdot B?? ?A C?$$

Gates

At the most basic level, gates are simply electronically controlled switches. These switches can then be grouped together to perform functions such as logical AND, NOT and OR thus forming the gate. The switches used in the gates in this report were constructed from *Complementary metal-oxide-semiconductor Field Effect Transistors (CMOS FETs)*. CMOS FETs have three terminals: Source, Drain and Gate and there are two types available: n-channel and p-channel. When a voltage is applied to the Gate of an n-channel CMOS FET, current can flow from the Source to the Drain, therefore, turning the switch on. A p-channel works in the opposite way, so that a current only flows from the Source to the Drain when no voltage is applied to the Gate.

1st Year Electronics Laboratory – Autumn Term, 2001

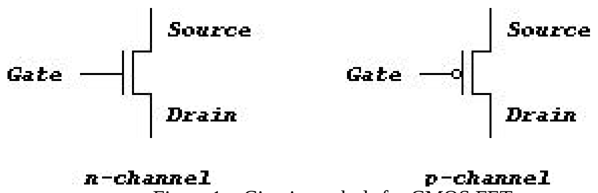


Figure 1 – Circuit symbols for CMOS FETs

Practical Implementation of NAND and NOR Gates

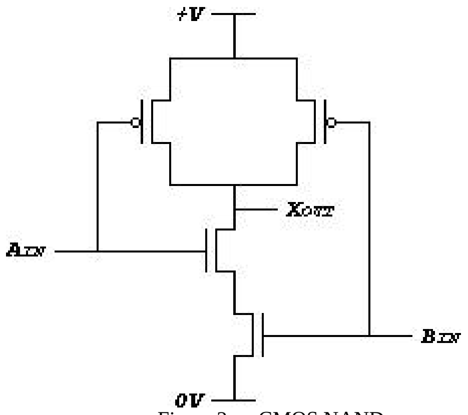


Figure 2a – CMOS NAND

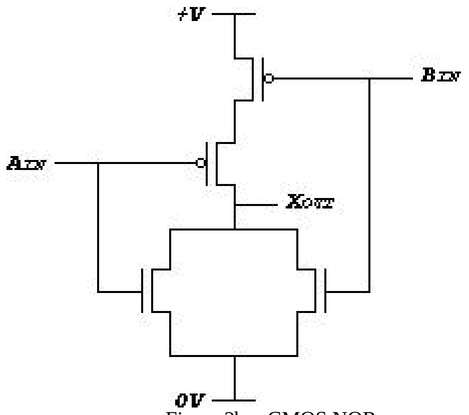


Figure 2b – CMOS NOR

Two's Complement

For a *normal* (unsigned) binary number, the weighting of all the bits is positive;

$$?0011?_2=0\times 2^3? 0\times 2^2? 1\times 2^1? 1\times 2^0=?3?_{10}$$

In order to express negative numbers, the *most significant bit* is given a negative weighting as follows;

$$?1101?_2=1\times ?- 2^3?? 1\times 2^2? 0\times 2^1? 1\times 2^0=?- 3?_{10}$$

An unsigned binary number can be turned into a two's complement number by inverting all the bits and adding one to the result.

1st Year Electronics Laboratory – Autumn Term, 2001

Experiments

The first task in constructing a 4-bit Arithmetic Logic Unit was to provide a facility by which two 4-bit binary numbers - the inputs - could be added together. Then, using the two's complement theory it was not difficult to extend the ALU in order to provide a method of subtracting the two binary inputs from each other. Using these two facilities it was then possible to make the ALU count by fixing one of the inputs at 1 and adding or subtracting it to or from the other input Finally an instruction was added that allowed the ALU to left shift the bits on input A. This results in multiplication by two of input A.

These tasks were accomplished through a series of experiments that are described below;

- Half Adder
- Full adder.
- 4-bit Serial Adder (4-bit ripple through adder).
- Subtracting Inputs.
- INC / DEC functions.
- Left shift

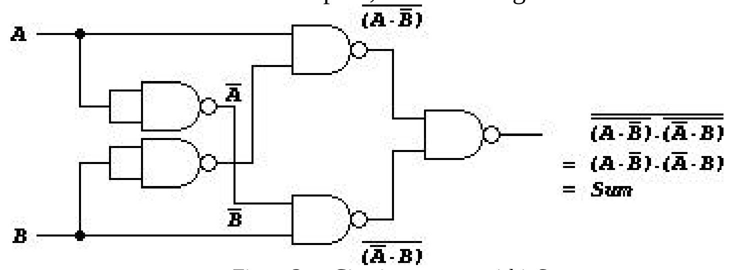
Half Adder

The truth table for 1-bit binary arithmetic is as follows;

A	B	Sum	Carry
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Table 1 – Truth tablefor 1-bit binary arithmetic

In order to generate the 1-bit sum of two 1-bit inputs, the following circuit was constructed;



A
B
$\overline{A}$
$\overline{B}$
$\overline{(A\cdot\overline{B})}$
$\overline{(\overline{A}\cdot B)}$
$\overline{\overline{(A\cdot\overline{B})}\cdot\overline{(\overline{A}\cdot B)}}$
$= (A\cdot\overline{B})\cdot(\overline{A}\cdot B)$
$= Sum$

Figure 3a – Circuit to generate 1-bit Sum

From the truth table *(Table 1)*, it can be seen that the carry output is only high when both inputs are high. Therefore, the following equation is true;

$Carry = A \cdot B$ Carry = A **AND** B

An **AND** Gate can be constructed from 2 NAND Gates;

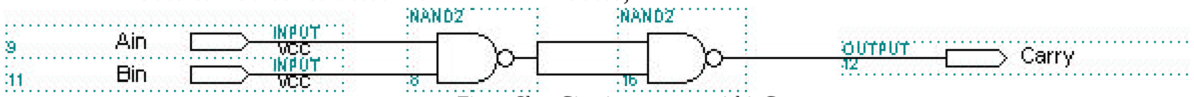


Figure 3b – Circuit to generate 1-bit Carry

Full Adder

In order to be able to perform arithmetic on numbers that are more than 1-bit long, it is necessary to consider the carry bit from previous stages. The truth table for a full adder is as follows;

A	B	Carry in	Carry out	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1

A	**B**	**Carry in**	**Carry out**	**Sum**
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Table 2 – Truth tablefor a 1-bit Full Adder

From *Table 2*, the following equations for *Carry out* and *Sum* can be derived;

Carry out

$$Carry\ Out=C_{OUT}\qquad Carry\ in=C_{IN}$$

$$C_{OUT}=?\overline{A}\cdot B?\,C_{IN}?\ ?A\cdot\overline{B}?\,C_{IN}?A\cdot B?\,\overline{C_{IN}}?\ ?A\cdot B?\,C_{IN}$$

Using $A\,?B?\ C?=?A\cdot B??\ ?A\cdot C?$;

$$C_{OUT}=C_{IN}\,[?\overline{A}\cdot B??\ ?A\cdot\overline{B}?]?\ ?A\cdot B?\,[\overline{C_{IN}}?\ C_{IN}]$$

$$C_{OUT}=C_{IN}\,(A\oplus B)?\ ?A\cdot B?\qquad\qquad \oplus = \textbf{XOR -}\quad ?A\cdot\overline{B}??\ ?\overline{A}\cdot B?$$

Equation 1a – Carry out for a Full Adder

Sum

$$Sum=\overline{A}\,?\overline{B}\cdot C_{IN}??\ \overline{A}\,?B\cdot\overline{C_{IN}}??\ A\,?\overline{B}\cdot\overline{C_{IN}}??\ A\,?B\cdot C_{IN}?$$

Using $A\,?B?\ C?=?A\cdot B??\ ?A\cdot C?$

$$Sum=C_{IN}\ ?\overline{A}?\ \overline{B}??\ \overline{C_{IN}}\ ?\overline{A}\cdot B??\ \overline{C_{IN}}\ ?A\cdot\overline{B}??\ C_{IN}\ ?A\cdot B?$$

$$Sum=C_{IN}\,[?\overline{A}\cdot\overline{B}??\ ?A\cdot B?]?\ \overline{C_{IN}}\ [?\overline{A}\cdot B??\ ?A\cdot\overline{B}?]$$

Equation 1b – Sum for a Full Adder

From equations 1a and 1b, it was noticed that the parts highlighted in red are the equation for the Sum of a half adder and the parts highlighted in blue are the equation for the Carry of a half adder. It was then possible to construct the full adder from the following circuit;

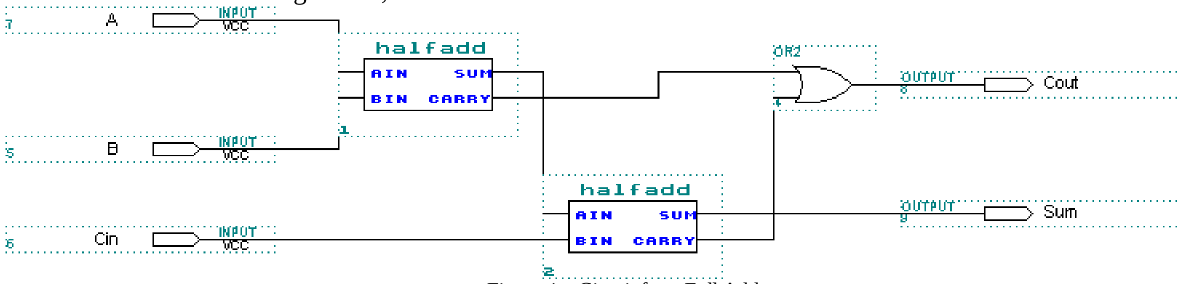
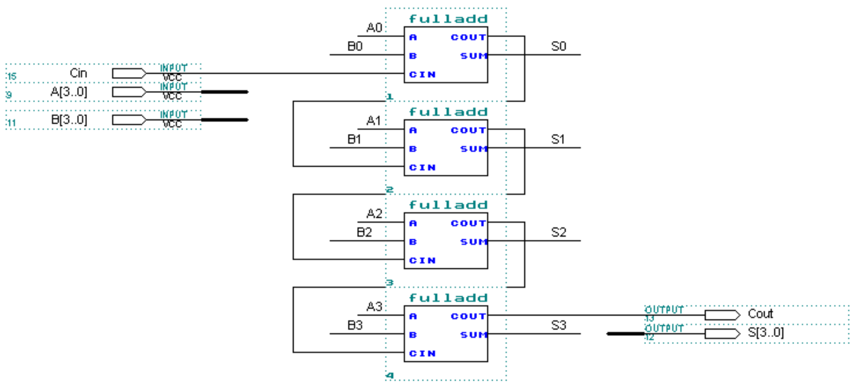


Figure 4 – Circuit for a Full Adder

4-bit Serial Adder

N 1-bit full adders can be cascaded to form an N-bit adder by connecting the carry out of one stage to the carry in of the next stage. A 4-bit serial (or ripple-through) adder was constructed;

Figure 5 – Circuit for a 4-bit Serial Adder



Subtracting the inputs

By representing the inputs as *two's complement* numbers at the input stage to the adder, if a negative number is placed on the input, subtraction takes place;

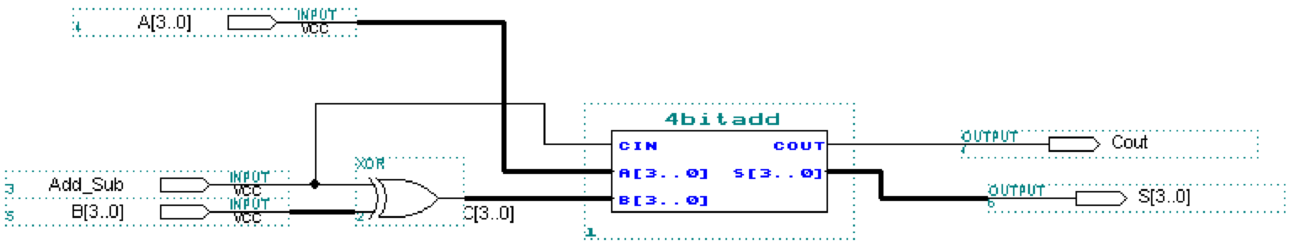


Figure 6 – Circuit to subtract input B from input A

When the user wants subtract B from A, setting Add_Sub high will deliver the two's complement of B to the adder input.

INC / DEC functions

By fixing B as one and selecting or deselecting Add_Sub, input A can be incremented or decremented;

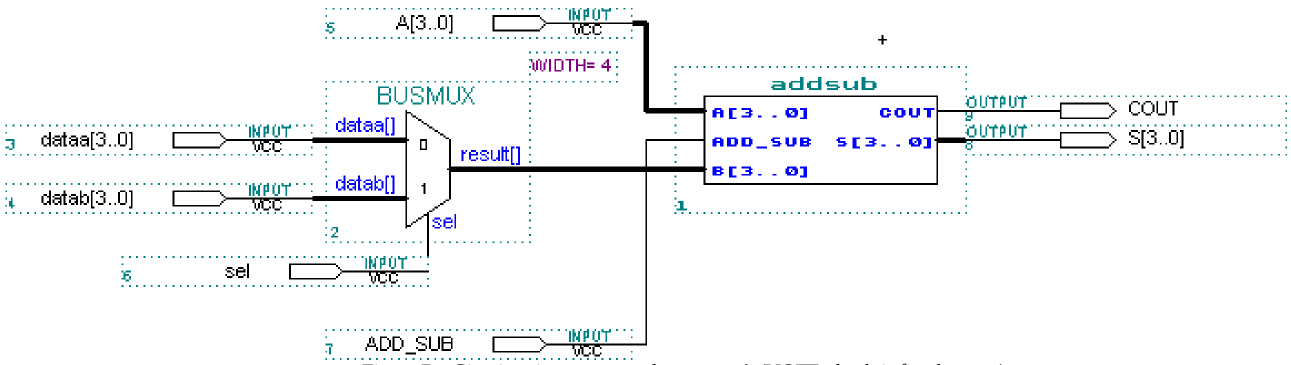
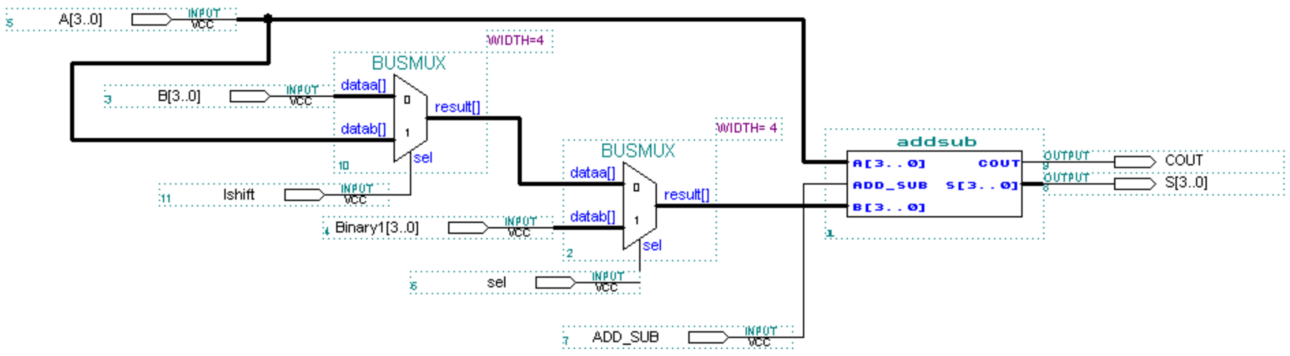


Figure 7 – Circuit to increment or decrement A (NOTE: datab is fixed to one)

Left shift

Left shifting the bits in input A is equivalent to multiplying it by two. Therefore, by copying input A to input B and adding, the required output will be produced;

Figure 8 – Circuit to left shift A (NOTE: Binary1 is fixed to one)



Results

The ALU was tested as it was built, and the results for each stage can be found in Appendix A

The entire ALU was then tested by iterating through every possible combination of inputs;

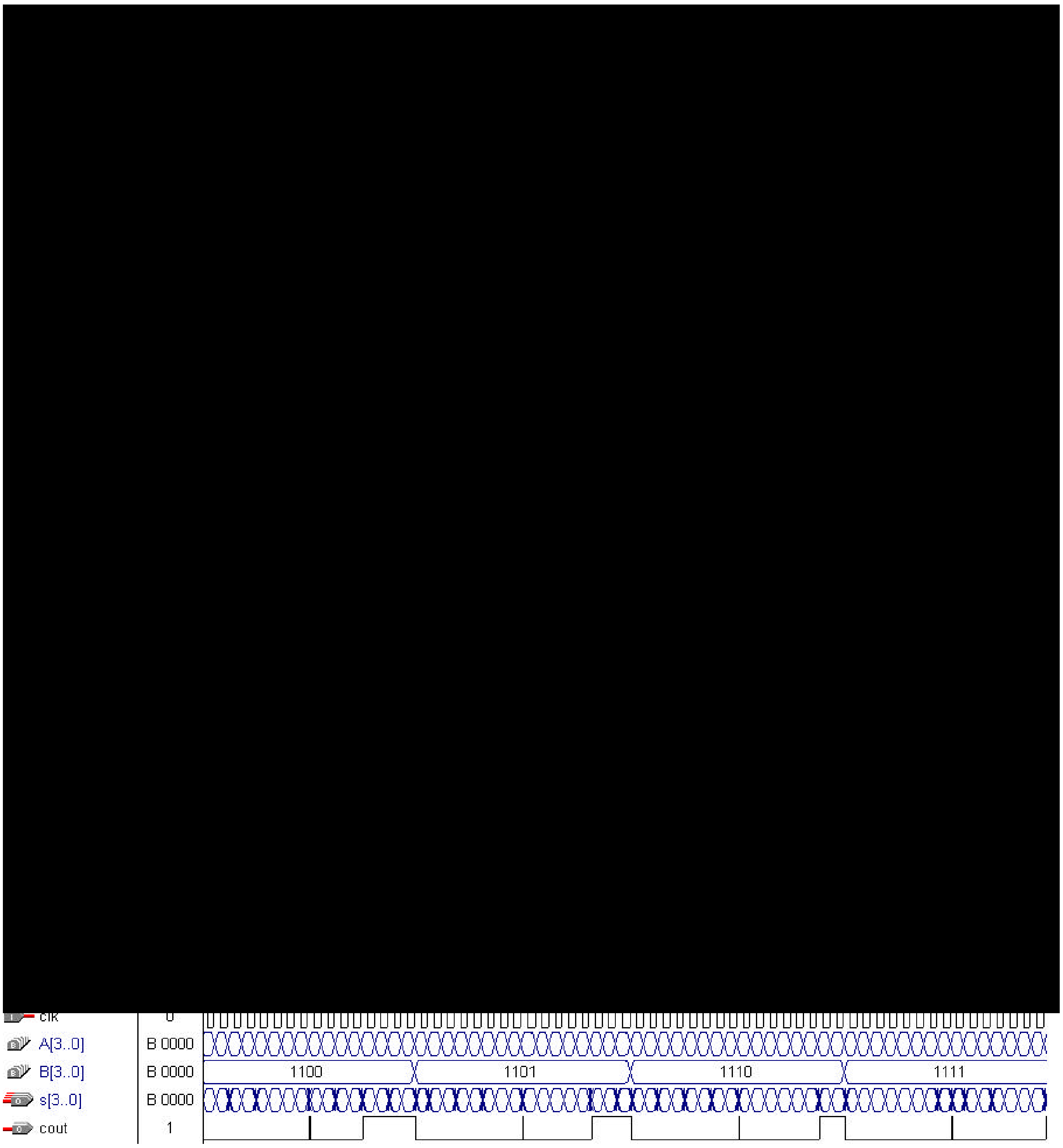


Figure 9 – Results

Discussion and Conclusions

The results show that the Arithmetic Logic Unit behaved as expected. It was however, noticed that there is a small delay between a change in the inputs and the corresponding change in the outputs. Although the delay is not large, it limits the speed at which the ALU can process data. The delay especially manifests itself in the serial adder beacause each stage can only begin successfully evaluating its inputs when the outputs from the previous stage have stabalised. If the input bus was widened, more stages would be added to the serial adder, resulting in a longer delay. This could be overcome by replacing the serial adder with a parallel adder. In a parallel adder, all the bits are evaluated at the same time.

Overall, the investigation was successful in meeting the aim of designing an Arithmetic Logic Unit in order to demonstrate methods of using *Boolean Algebra* to manipulate and solve various logic problems. The ALU also sucessfully performed useful functions in an efficient manner.

Appendix A - Complete result listing

Appendix B - Note about sources

Where information has been based on or paraphrased from a source it is placed between markers as follows;

Information from source

Source Name