# Introduction

In this lab, we designed an arithmetic logic unit for the MIPS processor in VHDL. The ALU has six operations: and, or, xor, add, subtract, and nor. The output is 32 bits along with flags for carryout, overflow, and zero. There are 4 inputs two of which are 32 bits and a single bit Carry in along with a 4-bit control word. The ALU should ideally perform the expected arithmetic and logic operations with 32 bit inputs correctly.

# Implementation

The solution we designed in VHDL uses 4 inputs two of which are 32 bits declared as STD\_LOGIC\_VECTOR. The carry-in is a single bit declared as STD\_LOGIC and the Cntrl word is 4 bits declared as an STD\_LOGIC\_VECTOR. Within the implementation, there are 4 outputs one of which is 32 bits declared as a STD\_LOGIC\_VECTOR and three other flags each corresponding to zero, carryout, and overflow and each of which are declared as STD\_LOGIC. Within the architecture, using behavioral VHDL we modeled the expected behavior of the ALU. This includes declaring signals for both 32-bit inputs and the 32-bit output. The signal for the output is 33-bits long to account for a carryout. Using a with select when statement we assigned values of Cntrl for each arithmetic and logic operation. Overflow was implemented using a single assignment and the Boolean expression for overflow. Carryout was implemented using another with select when statement where the MSB of the output signal would be checked during the addition operation.

A screen shot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

# Conclusion

Our ALU implementation functioned as desired through complete analysis of the timing diagrams. This is an important step in the MIPS implementation since it is fundamental to being able to execute any sort of arithmetic or logic instruction. Furthermore, it has strengthened our ability to write efficient VHDL as we move forward towards more complex labs.

# Inference

Implementing the ALU has provided us with a better understanding of how assembly language is translated into machine code. We now understand that the binary value of the control word is the same as in machine code and as we move up in the hierarchy in assembly, instructions such as add, subtract, xor etc. get translated into their respective binary values and de-muxed into an operation by the ALU.