Assignment 1: Creating a Simple ALU

Sarah Gillard

CS306: Computer Architecture

April 3, 2024

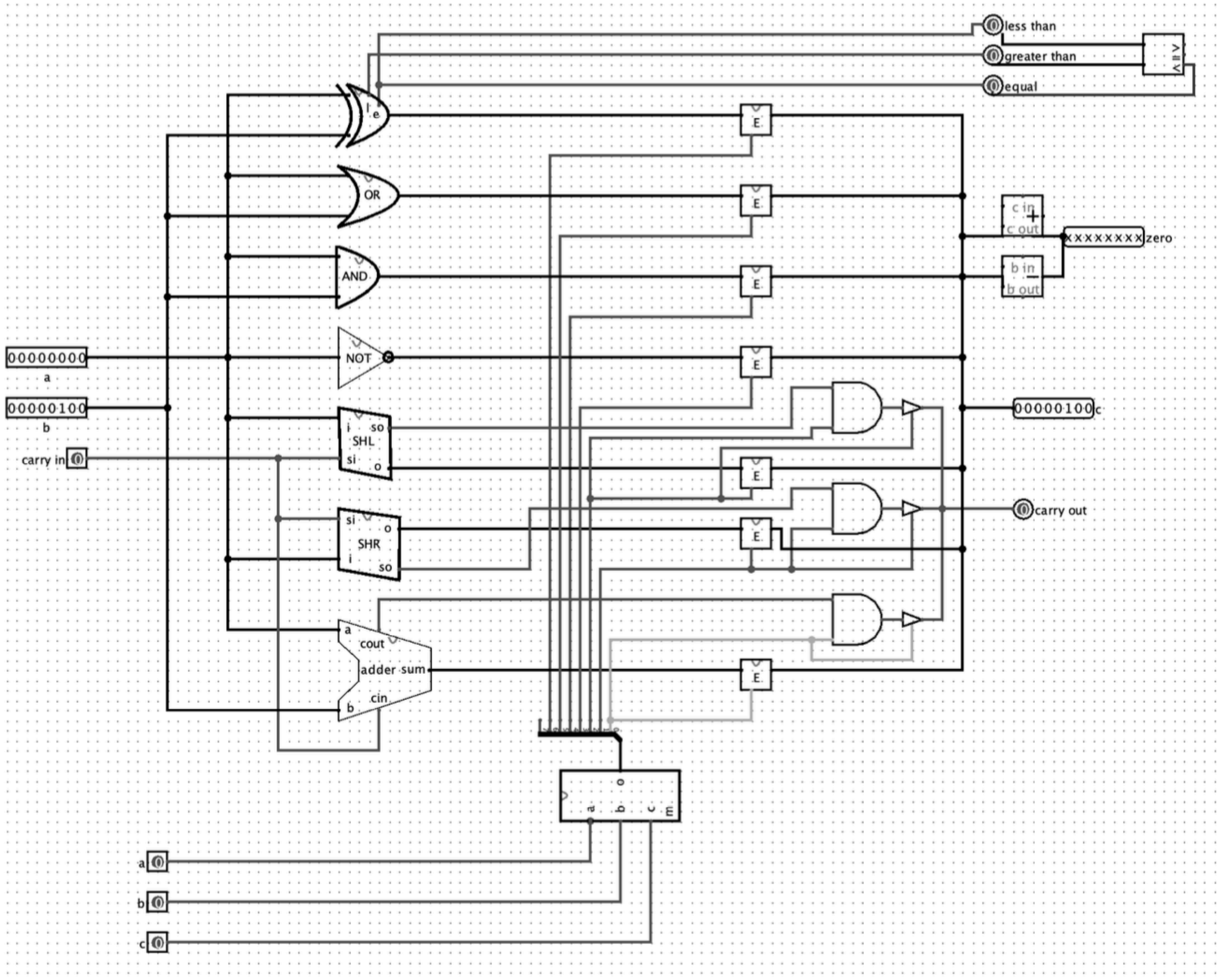
Project Overview

This project entails the creation of a basic CPU using Logisim, accompanied by the development of programs capable of running on it. The project unfolds in four phases. In phase one, you craft and assemble the Arithmetic Logic Unit (ALU) within Logisim. Following this, the second phase involves devising an instruction set that incorporates the functionalities designed in phase one. Phase three entails the creation and integration of a control unit into the ALU, culminating in the establishment of a fully operational CPU. Finally, in phase four, you'll write assembly language programs tailored to the CPU you've constructed.

Phase One

Build an 8-bit ALU using Logisim. This ALU can implement 16 instructions on 8-bit operands. You must include the following nine instructions that the ALU can implement: arithmetic addition, increment, decrement, comparison (with 3 outputs: one for equals, one for less than, and one for greater than), logic bitwise NOT, logic bitwise AND, logic bitwise OR, register right logic shift, and register left logic shift. You should also suggest five more instructions the ALU can implement and justify the importance of the five instructions you could add.

1. **NAND:** The NAND gate is also called the "negated AND" gate. It can be used to implement various logic functions including AND, OR, and NOT. This is useful as they are versatile for performing various arithmetic and logic operations (Burch, C.).
2. **XNOR:** The XNOR gate is also known as the "exclusive-NOR" gate. Its output is true if the inputs are the same and false if the inputs are different, so it is useful for equality comparison (Burch, C).
3. **XOR:** This performs the logical XOR operation between two operands. XOR is used to filter records to return results where either, but not both, conditions are true (Bass, B., 2018). It is also called "exclusive OR."
4. **Bit Finder:** This component can be used to identify specific bits within a binary number. This is useful for purposes such as testing the state of individual bits or performing bitwise operations (Burch, C.).
5. **Divider:** This component essentially performs unsigned division (Burch, C.) and it is useful for performing division operations, which is necessary for many computing tasks and is a step in many algorithms.


less than
greater than
equal
< = >
l
e
OR
AND
NOT
00000000
a
00000100
b
carry in
i
so
SHL
si
o
SHR
a
cout
adder sum
cin
b
E
c in
c out
+
b in
b out
−
xxxxxxxx zero
00000100 c
carry out
o
a
b
c
m
a
b
c

<u>Phase Two & Three</u>

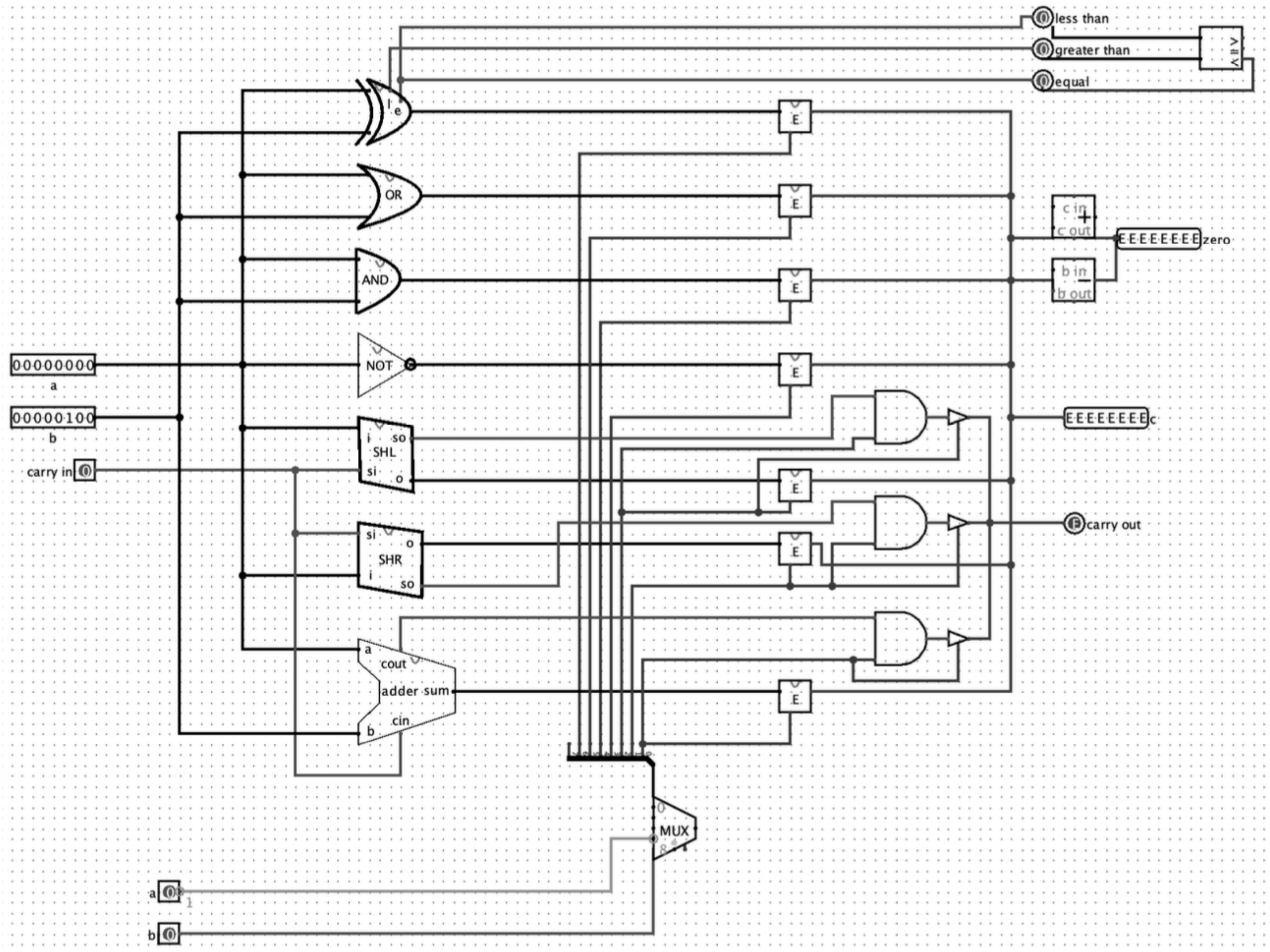
Design the instruction set of the ALU/CPU as follows:

- **Create the opcode table for the ALU by giving a binary code and a name for each instruction you built in Logisim in phase one.**
- **Decide how many operands you want your instructions to handle and justify your choice.**
- **In Logisim, add a multiplexor to the circuit you built in phase one that chooses one of the available operations.**

<u>Opcode table</u>

Numeric Code	Operation	Symbolic Code
0000	Arithmetic Addition	ADD
0001	Increment	INC
0010	Decrement	DEC
0011	Comparison	CMP
0100	Logic Bitwise NOT	NOT
0101	Logic Bitwise AND	AND
0110	Logic Bitwise OR	OR
0111	Register Right Logic Shift	RRLS
1000	Register Left Logic Shift	RLLS
1001	Negated AND NAND	NAND
1010	Exclusive-NOR XNOR	XNOR
1011	Exclusive-Or XOR	XOR
1100	Bit Finder	BITF
1101	Divider	DIV

My instructions will handle two operands and the result will be stored in one of the input registers. This choice allows for more flexibility in operations and is commonly used in CPUs.


less than
greater than
equal
OR
AND
NOT
SHL
SHR
adder
cout
sum
cin
00000000
a
00000100
b
carry in
zero
c
carry out
MUX
a
b

Phase Four

In order to be able to write assembly language for the CPU we need to add two instructions (without implementation). Write the following programs:

- **Write a program that adds operands until the new value to be added is 0. You do not need to implement the input operations to modify the contents of the registers. Just assume that by the end of each iteration, the register content is modified.**

```
# Set initial value of accumulator (ACC) to 0
MOV ACC, 0

LOOP:
   # Read operand from input
   READ operand

   # If operand is 0, exit loop
   CMP operand, 0
   JZ END_LOOP

   # If operand is negative, decrement operand
   CMP operand, 0
   JL DECREMENT

   # Add operand to ACC
   ADD ACC, operand

   # Print current value of ACC
   PRINT ACC

   # Repeat loop
   JMP LOOP

DECREMENT:
   # Decrement operand
   SUB operand, 1

   # Add decremented operand to ACC
   ADD ACC, operand

   # Print current value of ACC
   PRINT ACC

   # Repeat loop
   JMP LOOP

END_LOOP:
   # End of loop
```

- **Write a program that shifts the content of a register until the least significant bit is 0. Think of a way to stop shifting if the content of the register is 11111111 and add it to your program.**

```
# Set initial value of register (REG)
MOV REG, initial_value

# Repeat until the least significant bit is 0 or register content is 11111111
        LOOP:
# Shift the register content right by one bit
        RRLS REG, 1

# Check if the least significant bit of the register content is 1 and the register content is not 11111111
        CMP register
        CONTINUE

# Exit the loop if the condition is not met
        JMP END
        CONTINUE:
# Check if the least significant bit of the register content is 1
        TEST REG, 1
        JNE LOOP  # Continue looping if the least significant bit is 1

        END:
# Print current value of register
        PRINT REG
```

References

Aljendi, S. (2018, April 21). *Creating an assembly language using an instruction set.* Study.com. https://study.com/academy/lesson/creating-an-assembly-language-using-an-instruction-set.html

Anonymous. *CS107 x86-64 reference sheet.* Stanford.edu. https://web.stanford.edu/class/cs107/resources/x86-64-reference.pdf

Burch, C. *Logisim: a graphical tool for designing and simulating logic circuits.* Carl Burch. http://www.cburch.com/logisim/docs/2.7/en/html/libs/index.html

Bass, B. (2018, August 27). *Basic computer architecture instruction types: functions & examples.* Study.com. https://study.com/academy/lesson/basic-computer-architecture-instruction-types-functions-examples.html

Ding, Zuoliu. (2019, January 28). *40 basic practices in assembly language programming.* Code Project. https://www.codeproject.com/Articles/1116188/40-Basic-Practices-in-Assembly-Language-Programmin

Fox, Charles. (2024). *Computer Architecture: From The Stone Age to the Quantum Age.* No Starch Press.

Mcgowan, Ross. (2018, January 19). *8 BIT CPU ALU* [Video]. YouTube. https://www.youtube.com/watch?v=MTGZRYdpdzE&list=PLZlHzKk21aImqCiV71iE2I1dUE5LNejQk&index=9

Mcgowan, Ross. (2018, February 15). *8 BIT CPU TOP LEVEL* [Video Playlist]. YouTube. https://www.youtube.com/playlist?list=PLZlHzKk21aImqCiV71iE2I1dUE5LNejQk