



North South University

Department of Electrical & Computer Engineering

Project Report

Implementation Part # No: 1 of 5

Project Title: Design and Simulation of a N-bit MIPS CPU/M

Course Code: CSE332

Course Name: Computer Organization & Architecture

Name & ID: Nawal Ayesha Khan, 1911301042

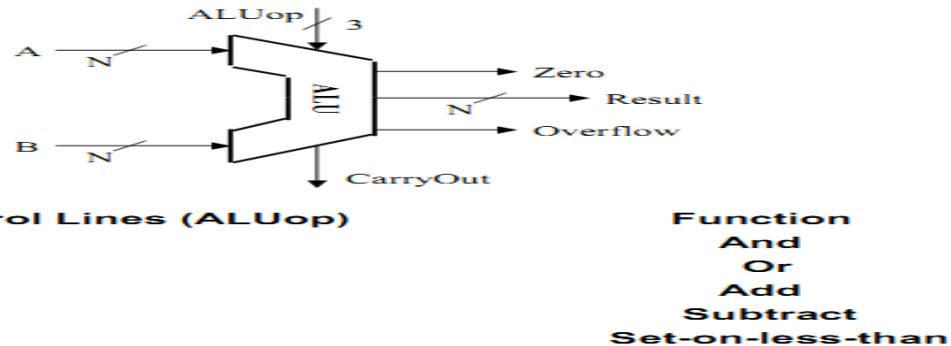
Dateline of this Part-#1 submission: with-in one (1) week from post in Google classroom or by a specific date, if given.

Deadline of Complete Submission: One (1) week before the last class of Spring 2021

Objectives:

Part #1- Design and simulation of a N-bits ALU with the following functional specification (e.g., Block diagram, Truth Table).

Functional Specification of the ALU



Inputs		Outputs						
ALUOp		Funct field, Fx						Operation
Binvert	Ainvert	F5	F4	F3	F2	F1	F0	
0	0					0	0	010 (add)
1	0				X	0		110 (sub)
1	1				X	0		011 (nand)
1	1				X	1		100 (nor)
0	0					0	1	000 (and)
0	0					1	0	001 (or)
0	0					1	1	111 (mult)

Truth table for ALU control bits

N=13

Design Process (with K-map/QM-method etc.):

Truth table:

ALU Op			Fx fields		Outputs		
C	B	A	F ₁	F ₀	x	y	z
0	0	0	0	0	0	1	0
0	1	0	1	0	1	1	0
0	1	1	1	0	0	1	1
0	1	1	1	1	1	0	0
0	0	0	0	1	0	0	0
0	0	0	1	0	0	0	1
0	0	0	1	1	1	1	1

* K-map:

$$C=0$$

$x = BA/F_1 F_0$		00	01	11	10
		00	①	①	①
00	00	01	11	10	
01					
11					
10					

$$\therefore x = C'B'A'F_1 F_0 + C'B A F_1 F_0 \\ + C'B A' F_1 F_0'$$

$y = BA/F_1 F_0$		00	01	11	10
		00	①	①	①
00	00	01	11	10	
01					
11					
10					

$$y = C'BF_1 F_0' + C'B'A'F_1 F_0' \\ + C'B'A'F_1 F_0$$

$z = BA/F_1 F_0$		00	01	11	10
		00	01	11	10
00	00	01	11	10	
01					
11					
10					

$$z = C'B'A'F_1 + C'B A F_1 F_0'$$

* QM method

$x =$	Group	minterm	C B A F ₁ F ₀
G_1	3	0 0 0	1 1
	10	0 1 0	1 0
G_2	15	0 1 1	1 1

Groups can not be merged any further.

$$\therefore x = C'B'A'F_1F_0 + C'B'A'F_1F_0' + C'B'A'F_1F_0$$

$y =$	Group	minterm	C B A F ₁ F ₀
G_1	0	0 0 0	0 0
	3	0 0 0	1 1
G_2	10	0 1 0	1 0
	14	0 1 1	1 0

Group	$minterm$	C B A F ₁ F ₀
G_2'	10, 14	0 1 - 1 0

Groups can't be merged any further

$$\therefore y = C'B'F_1F_0' + C'B'A'F_1F_0' + C'B'A'F_1F_0$$

$z =$	Group	minterm	C B A F ₁ F ₀
G_1	2	0 0 0	1 0
	3	0 0 0	1 1
G_2	14	0 1 1	1 0

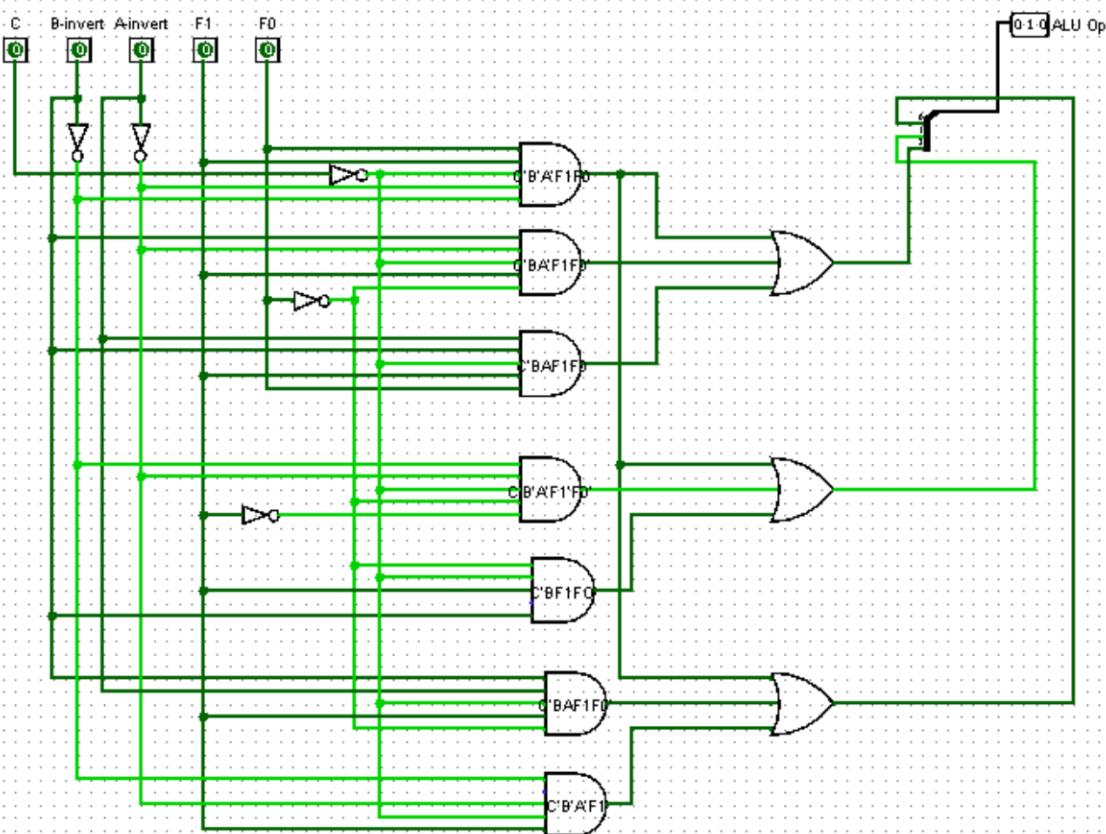
Group	minterm	C	B	A	F_1	F_0
G1'	2, 3	0	0	0	1	-

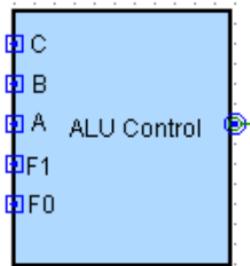
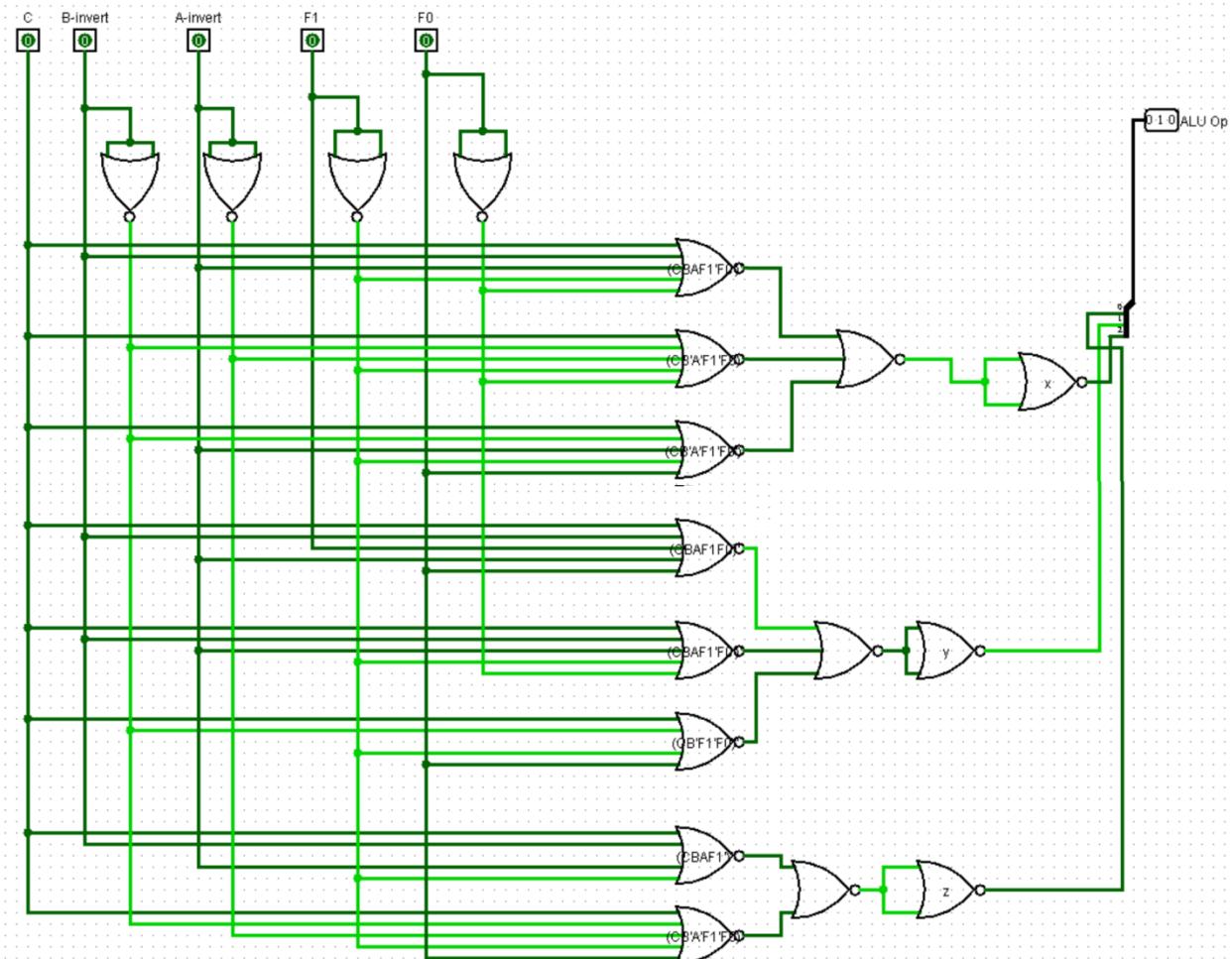
Groups can't be merged any further.

$$\therefore Z = C'B'A'F_1 + C'B'A F_1 F_0'$$

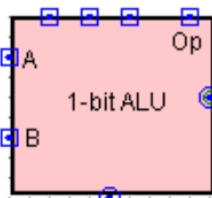
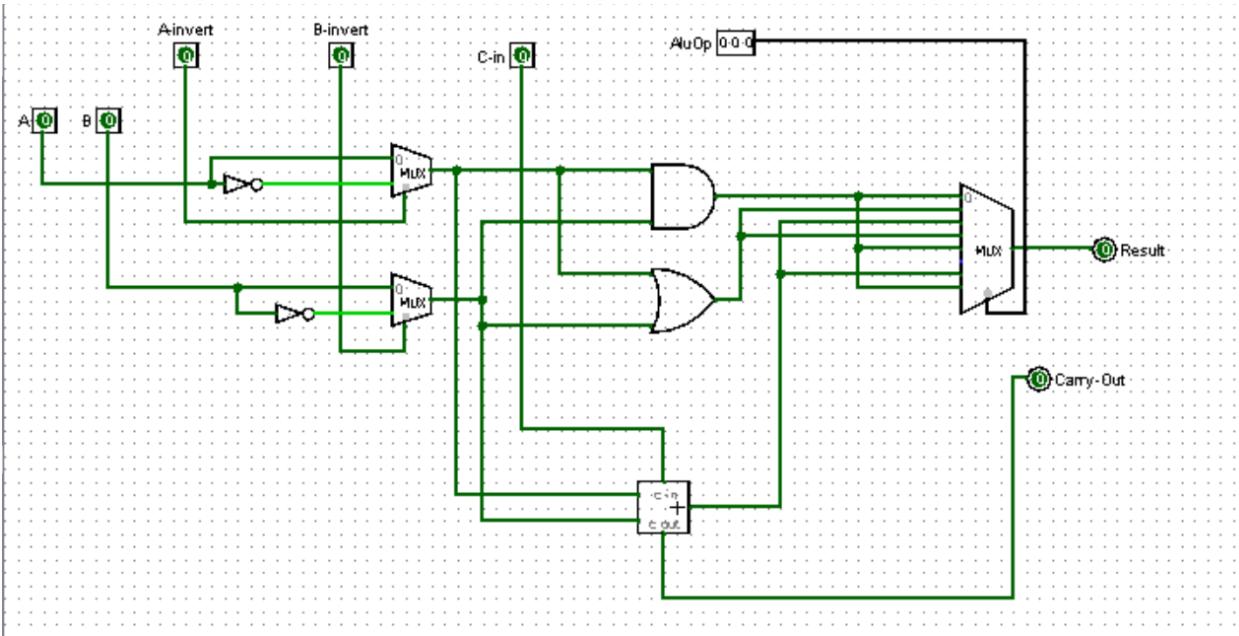
Result of simulation (using LogiSIM):

Circuit diagram for ALU control (Normal and NOR implementation):

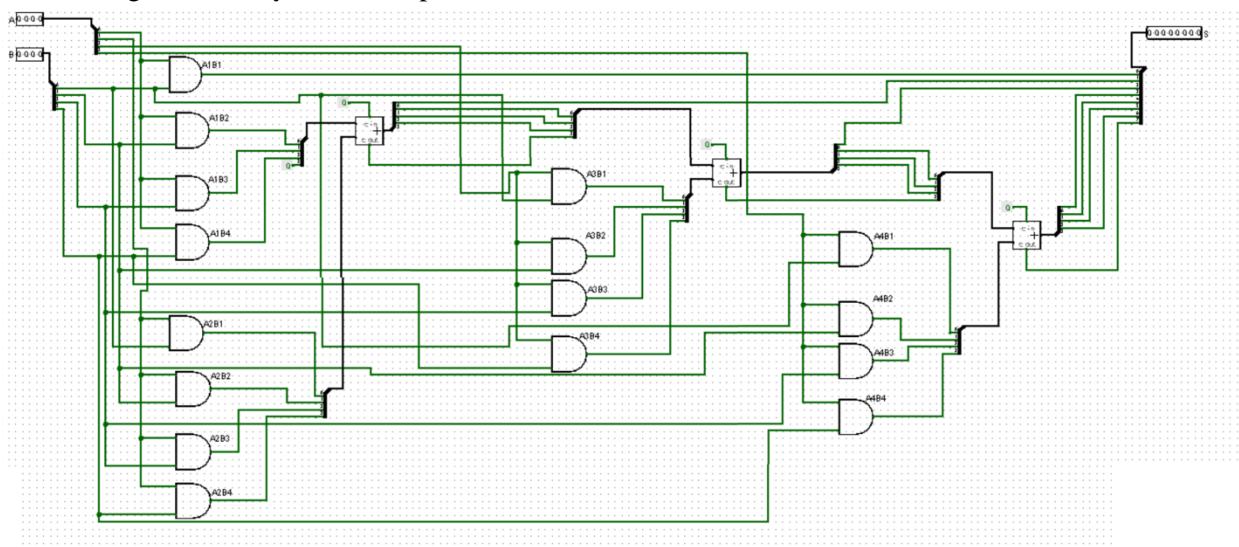




Circuit diagram for 1-bit ALU:

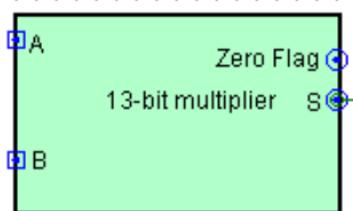
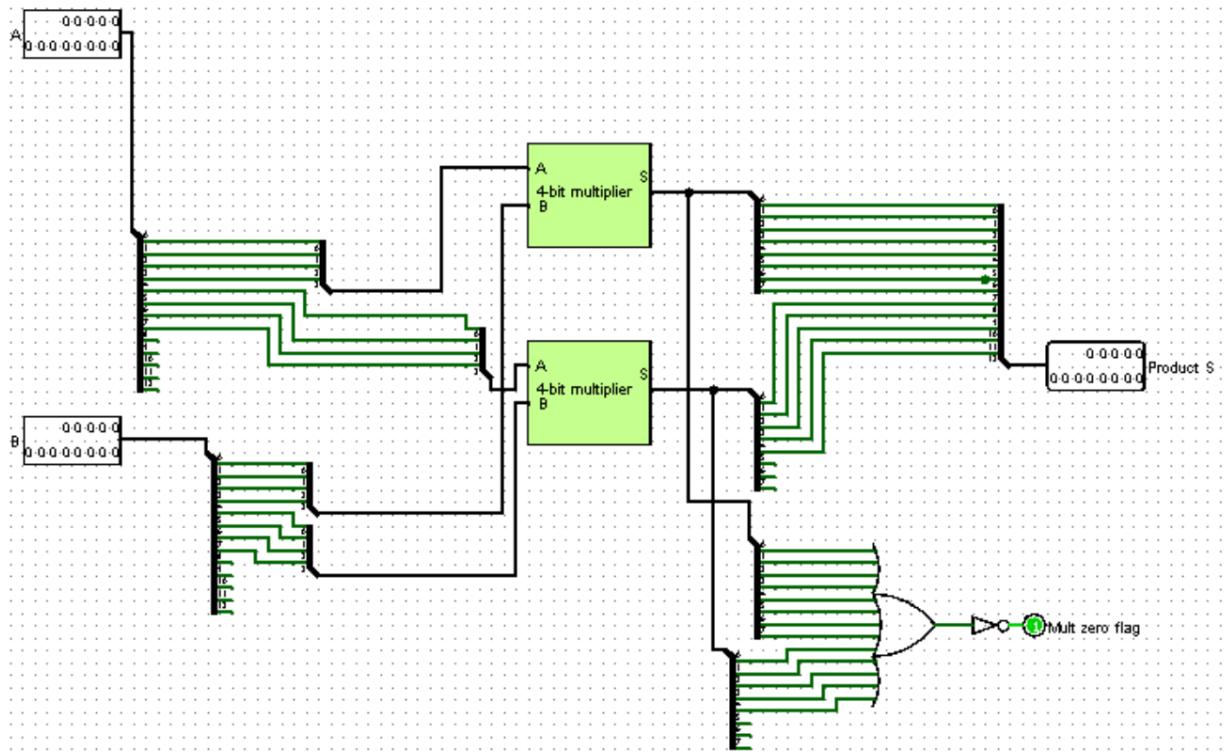


Circuit diagram for 4 by 4-bit multiplier:

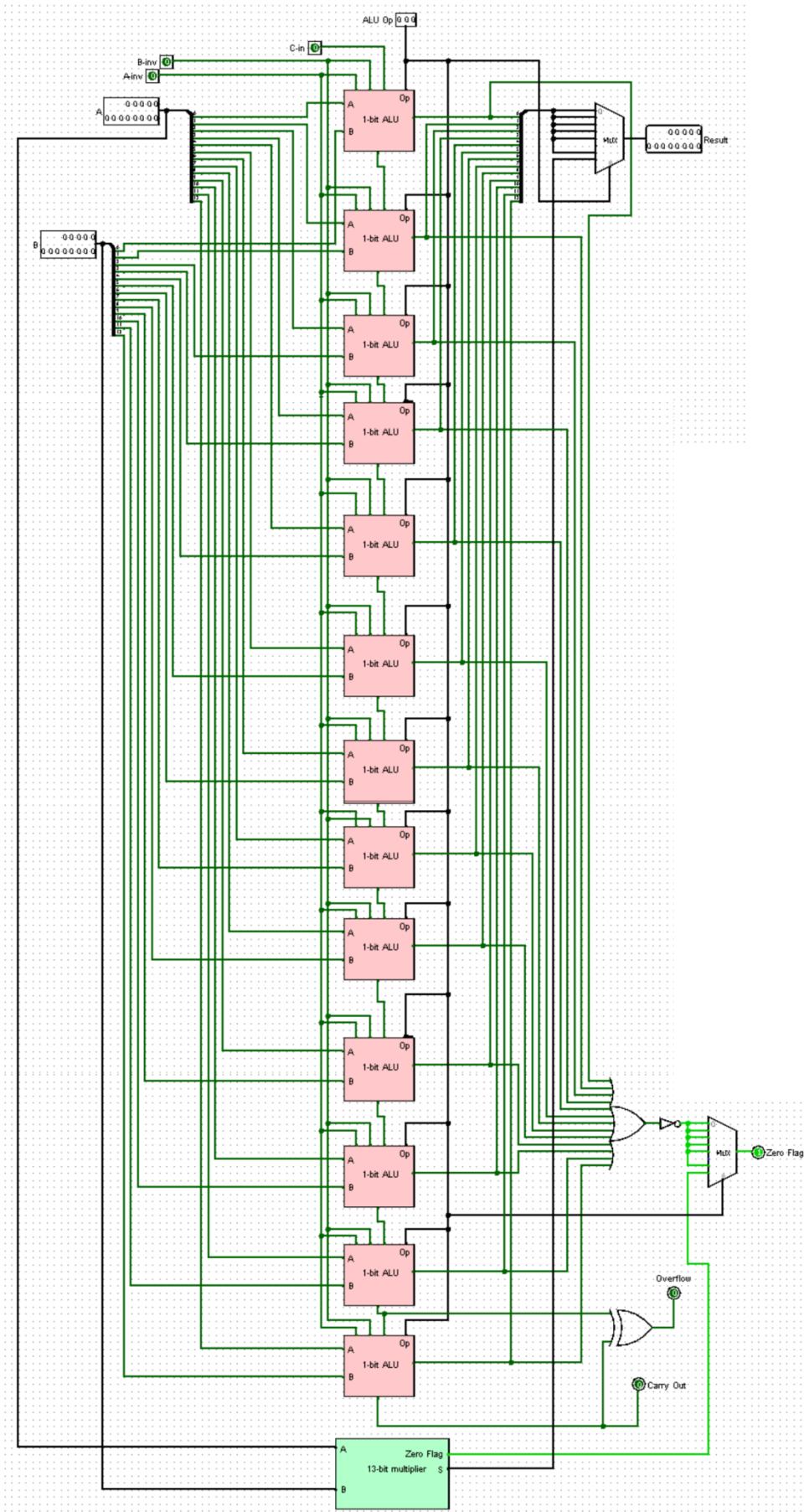




Circuit diagram for 13-bit multiplier:

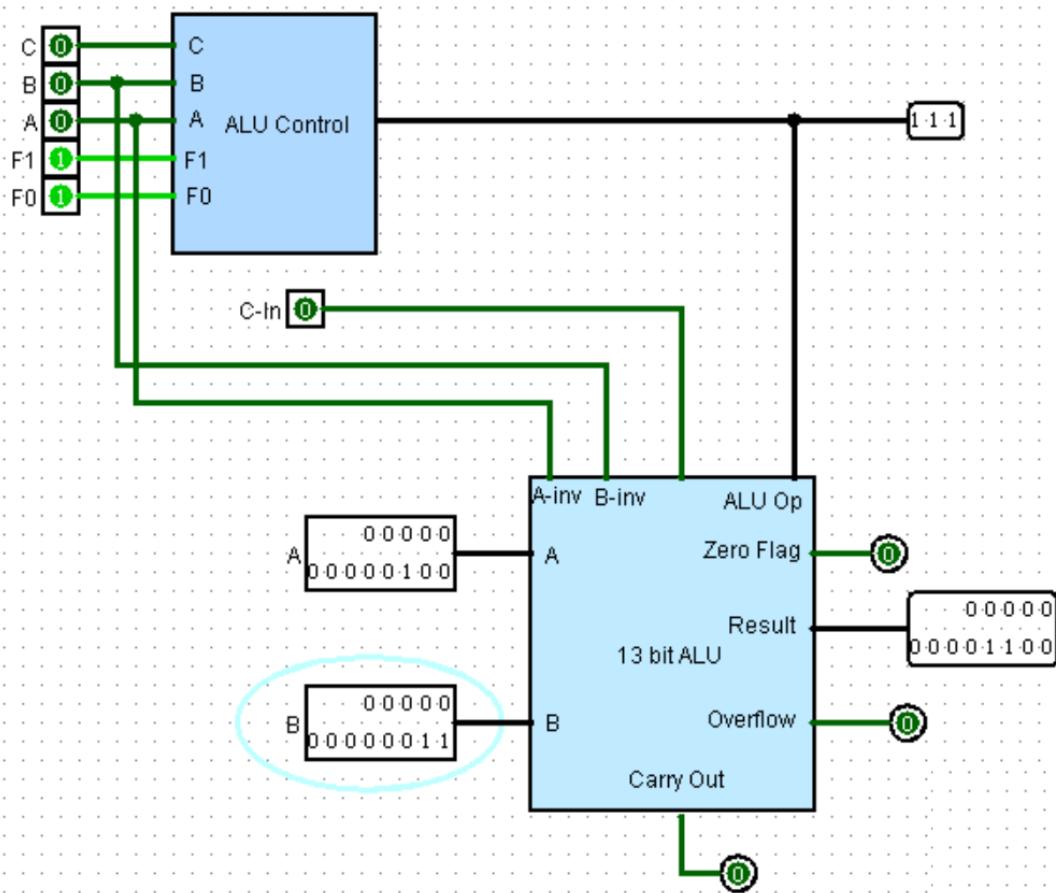


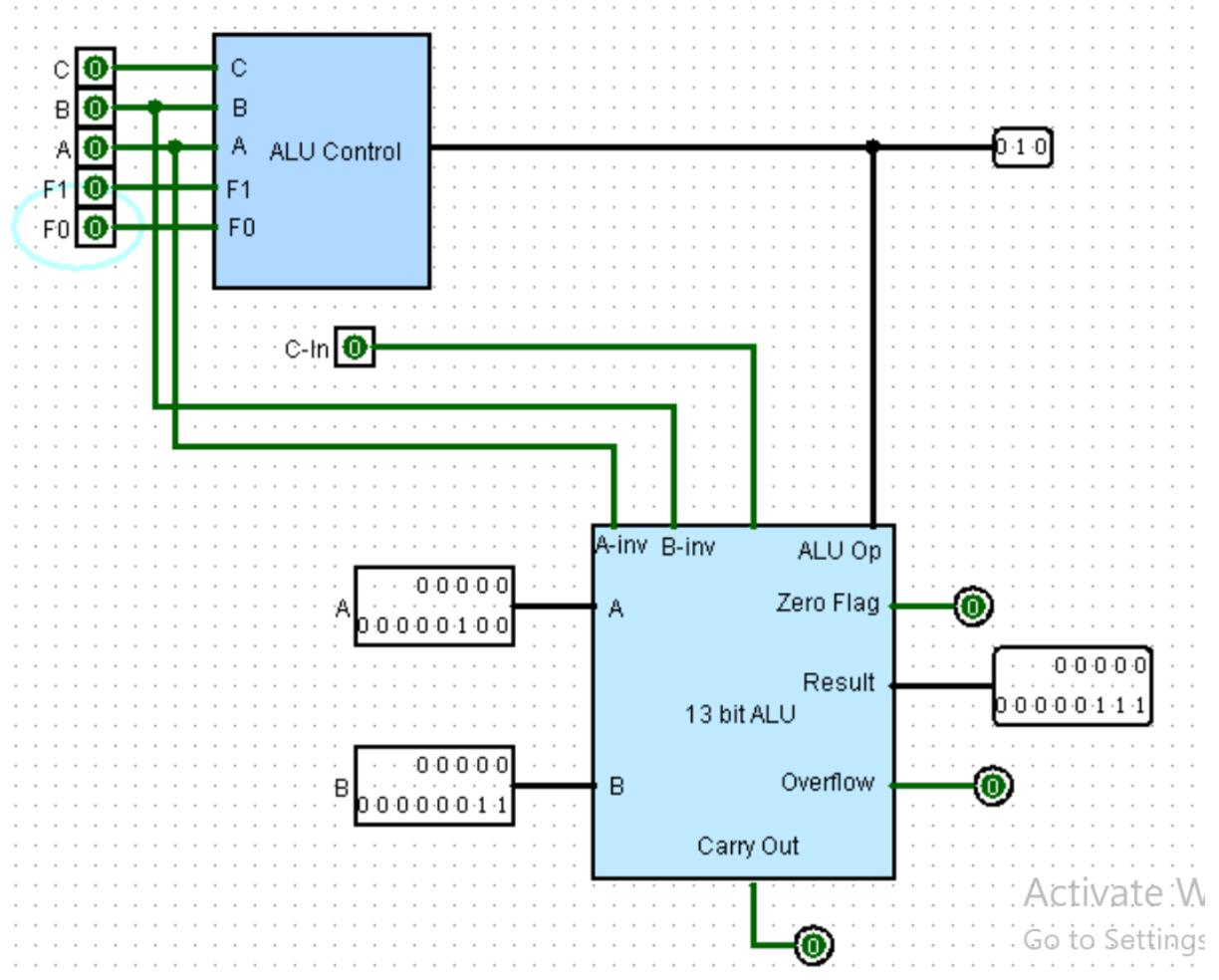
Circuit diagram for 13-bit ALU:





Implementation:





Discussion (improvements, debriefs etc.):

In the first part of the project, we designed a 13-bit ALU with the AND, OR, NAND, NOR, ADD, SUB, and MULT operations, and an ALU control using the inputs C, B-invert, A-invert, and Function fields F1 and F0. We create an ALU control using the expressions we've formed from the truth tables, and simplify the expressions using k-maps and QM method. They are then implemented using logic gates, and the output is used to send values to the 13-bit ALU unit, and will determine which operation to carry out. The output depends on the combination of values we get from Control inputs C, B-invert, A-invert, F1, and F0. We also create a 4-bit multiplier using 4-bit adders and AND gates, and combine these multipliers to create a 13-bit multiplier, designed to multiply the last 6 digits of the inputs the ALU will receive. We create a 1-bit ALU with the AND, OR, NAND, NOR, ADD, and SUB operations, and include a multiplexer to select which operation to carry out. We then stack the 1-bit ALUs to create the 13-bit ALU, and add the MULT operation here by including the 13-bit multiplier. We use a multiplexer to choose whether to display the results for the AND, OR, NAND, NOR, ADD, SUB operations or the MULT operation based on the output that has been sent from the ALU control.

There was the issue of there being no zero-flag output when the multiplication operation was chosen, but this was solved using an internal zero-flag output in the IC for the 13-bit multiplier. This way, we can determine if the output was zero no matter which operation was chosen. The zero-flag output for the 13-bit ALU was created by using an OR gate on the individual results of

the 13-bit ALU and then inverting the output, and based on whichever operation has been chosen, the multiplexer will give the zero-flag output for the multiplication result or the other operations. Overflow was implemented by using a XOR gate on the carry-in and carry-out values of the 1-bit ALU for the most significant bit of the result. Splitters were used to separate and combine the data.