Name: Brian Zick

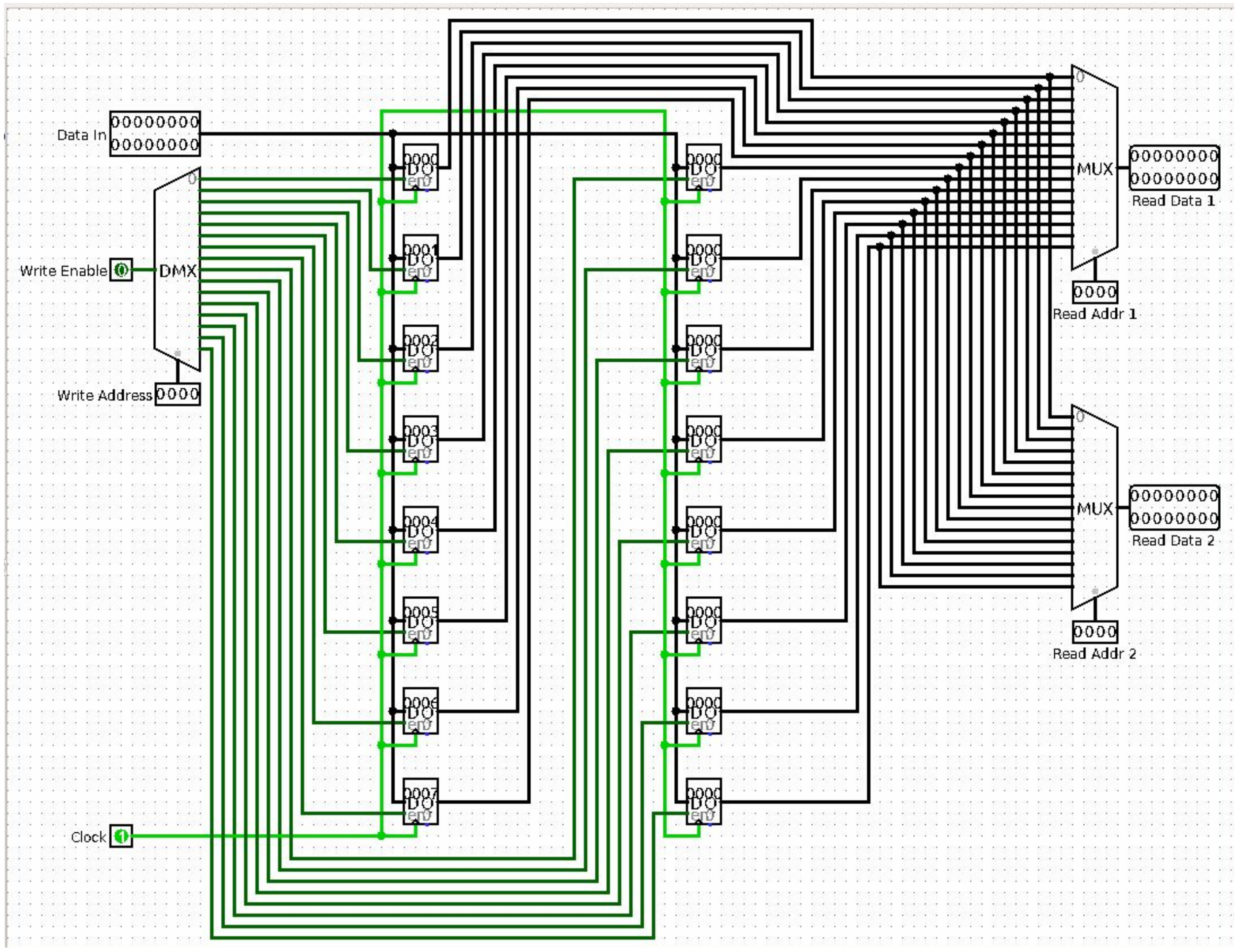
Processor Name: Zick 2700

Processor Description: 16-bit single cycle CPU with branch and jump support

Score Sheet

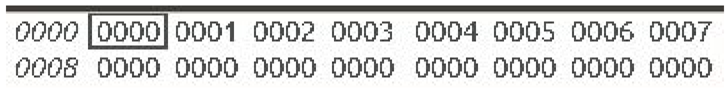
Feature	Value	Passes Tests	Screenshots	Score
Basic: ALU	10	X	See below	
Basic: R and I type	10	X	See below	
Basic: LW and SW	10	X	See below	
Basic: Assembler	20	X	See below	
Basic: Writeup	20	X	See below	
A la Carte				
16-bit CPU	10	X	See below	
Jump Instructions	10	X	See below	
Branch-if-equal	10	X	See below	
Carry Look-Ahead Adder	10	X	See below	
Total	110			(earned)

Basic:
ALU Proof:
The following screenshot demonstrates that the compiled Basic-R-Type-rev2.asm program executes correctly on my processor, and therefore the ALU functions as desired.

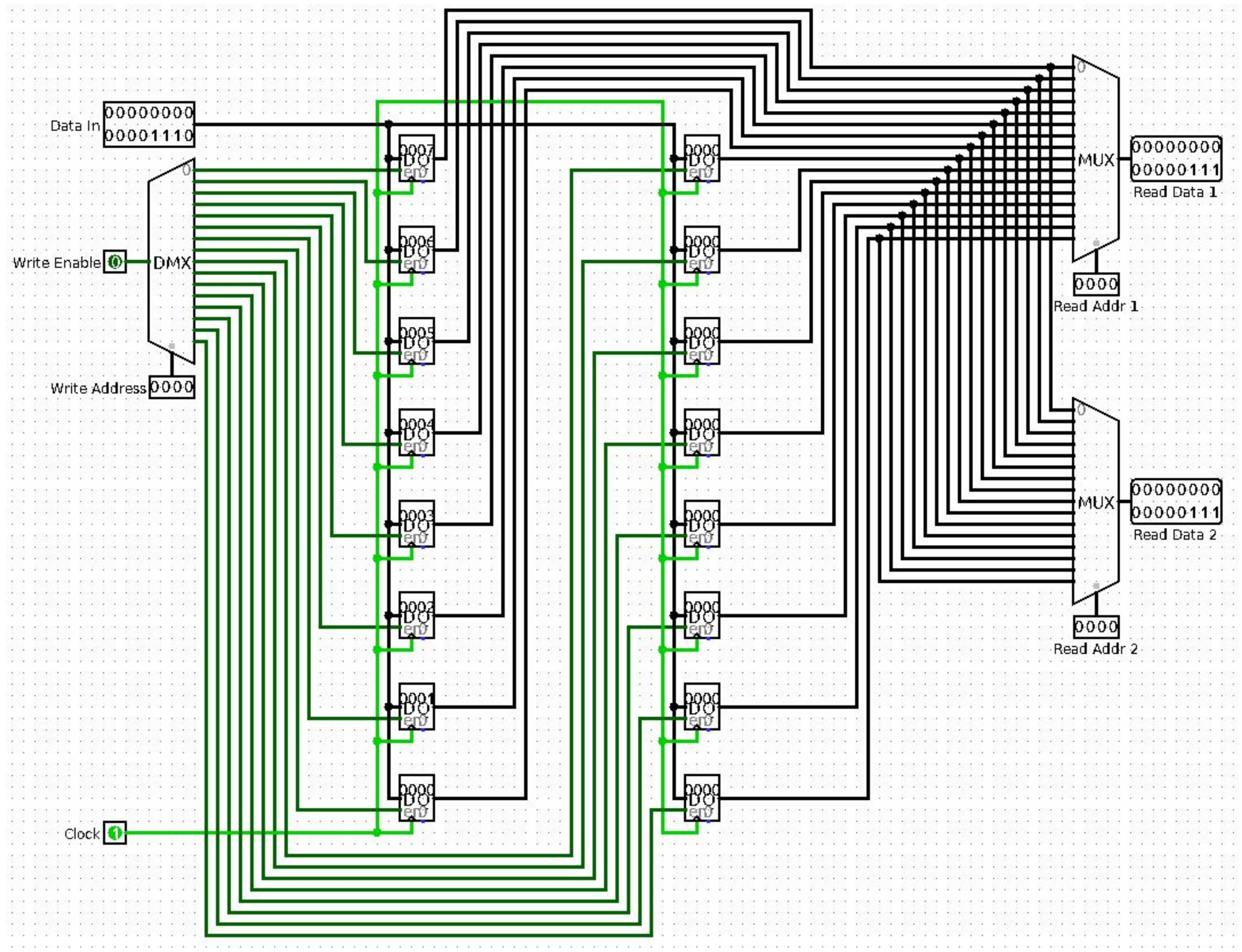

Data In
00000000
00000000
Write Enable
DMX
Write Address
0000
Clock
MUX
Read Data 1
Read Addr 1
Read Data 2
Read Addr 2

R and I type and LW/SW Proof:

The following images show the result of Basic-LW-SW-rev.asm which demonstrates that my processor can handle R and I type instructions including load word and store word.



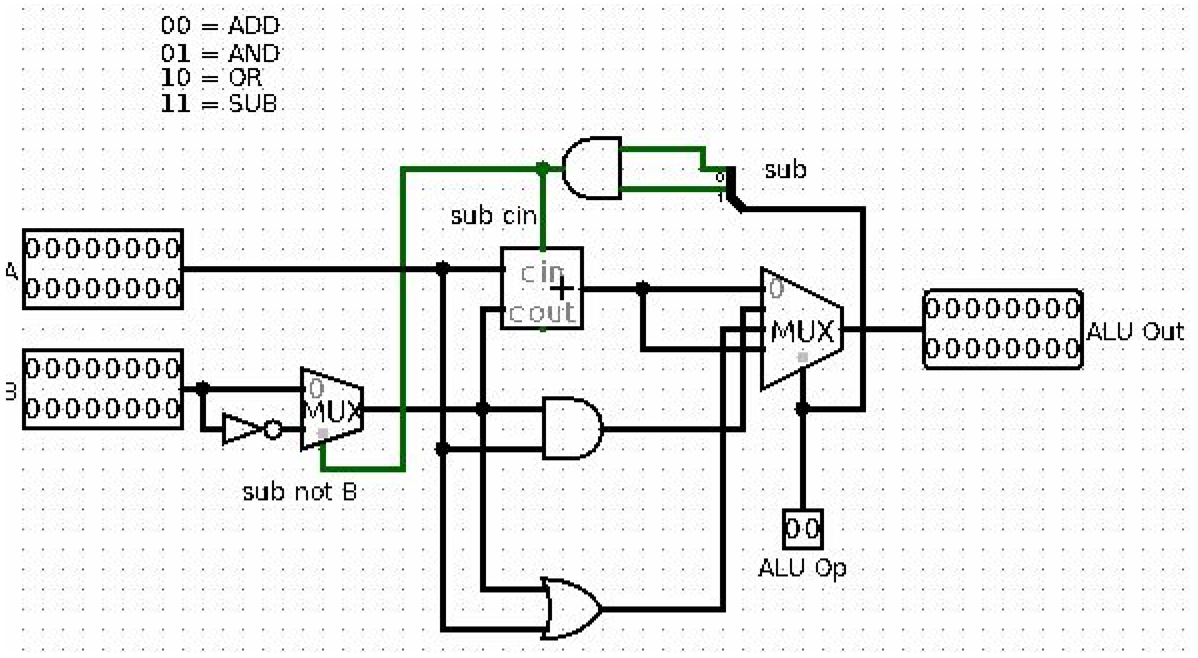
```
0000  0000  0001  0002  0003  0004  0005  0006  0007
0008  0000  0000  0000  0000  0000  0000  0000  0000
```



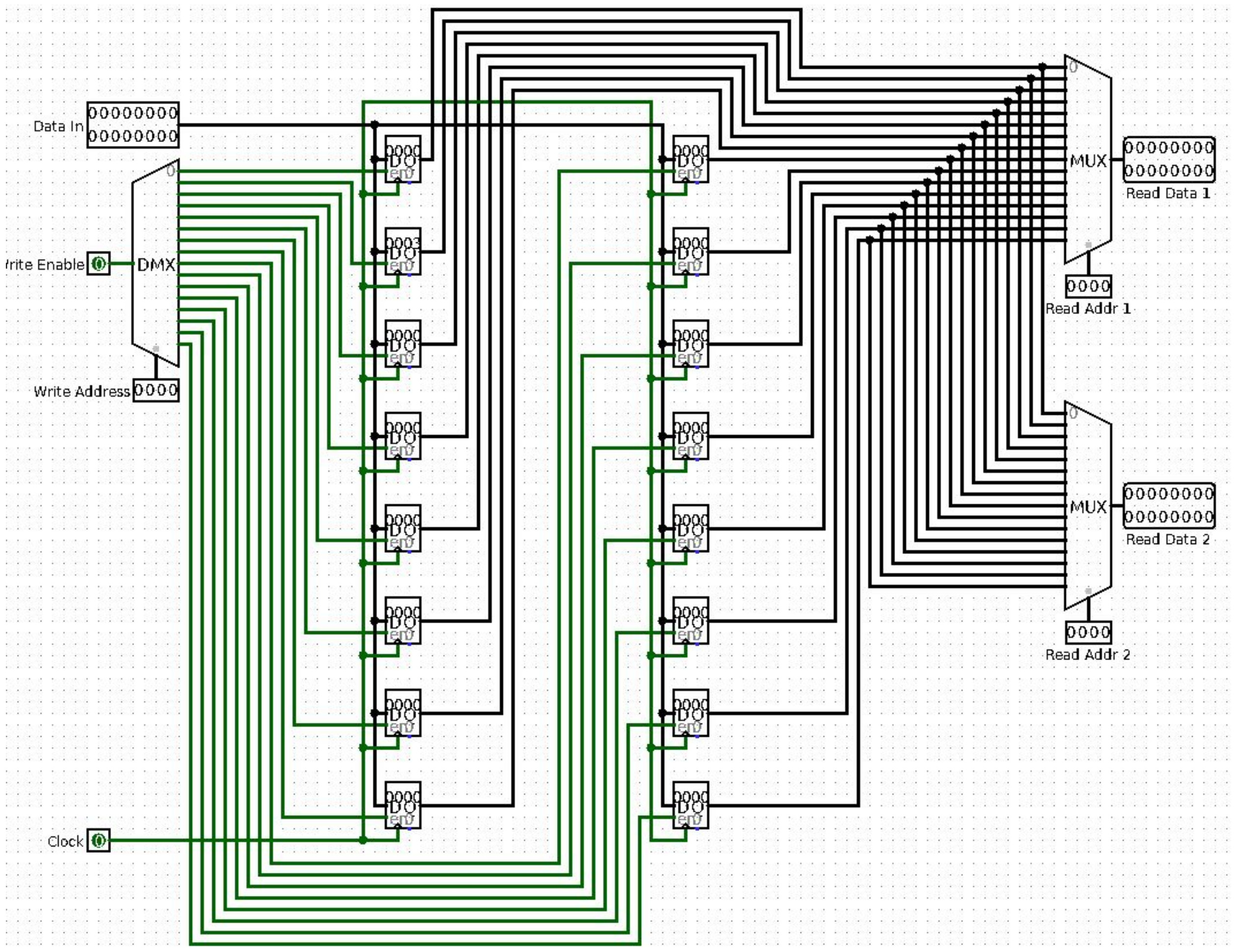
Kitchen Sink:

16-bit CPU Proof:

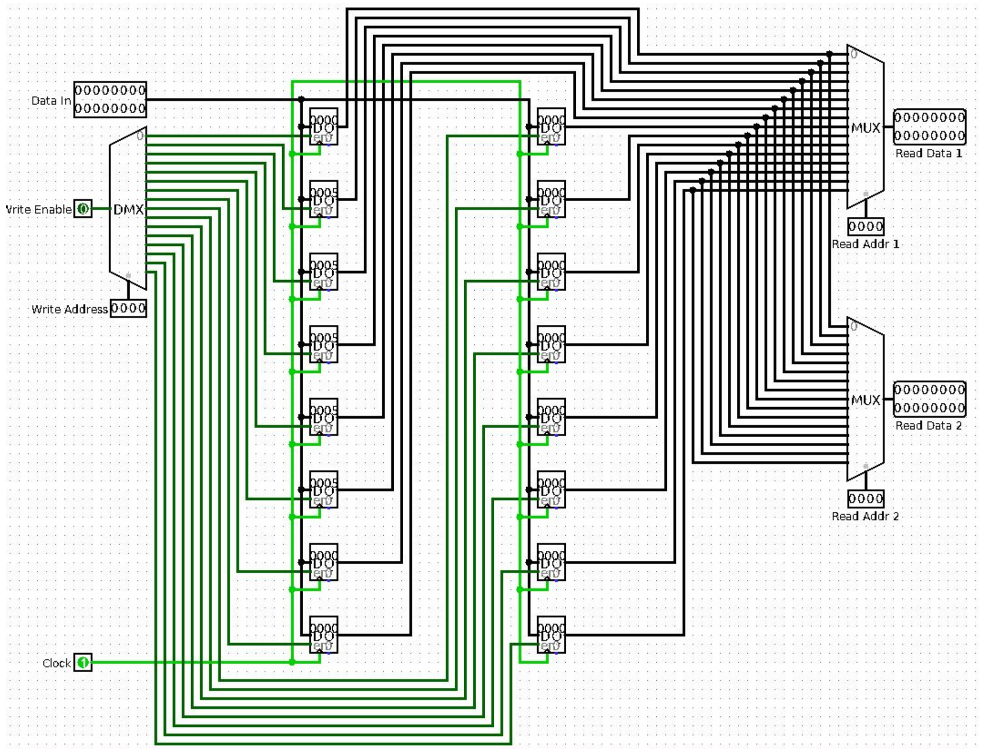
This is an image of my 16-bit ALU, which in conjunction with the many pictures above and below of my 16x16bit register file confirms I have a 16-bit CPU.

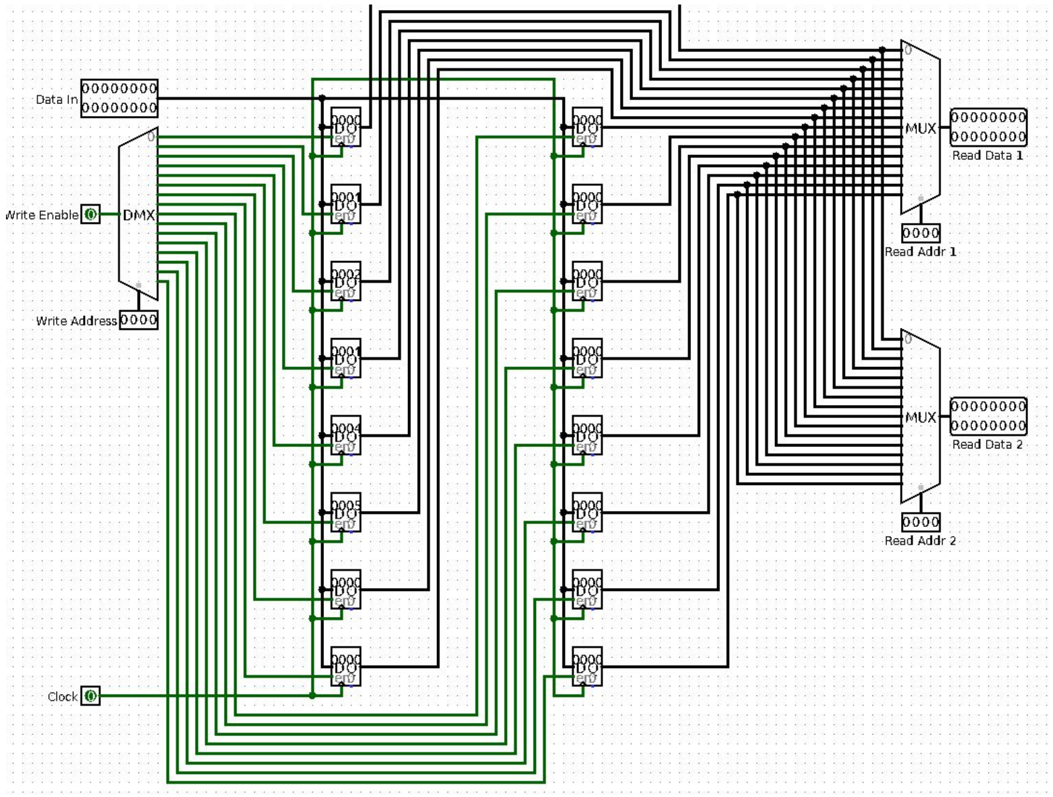


Jump Instructions Proof:
This image demonstrates that the program testjump-absolute.asm executes correctly.



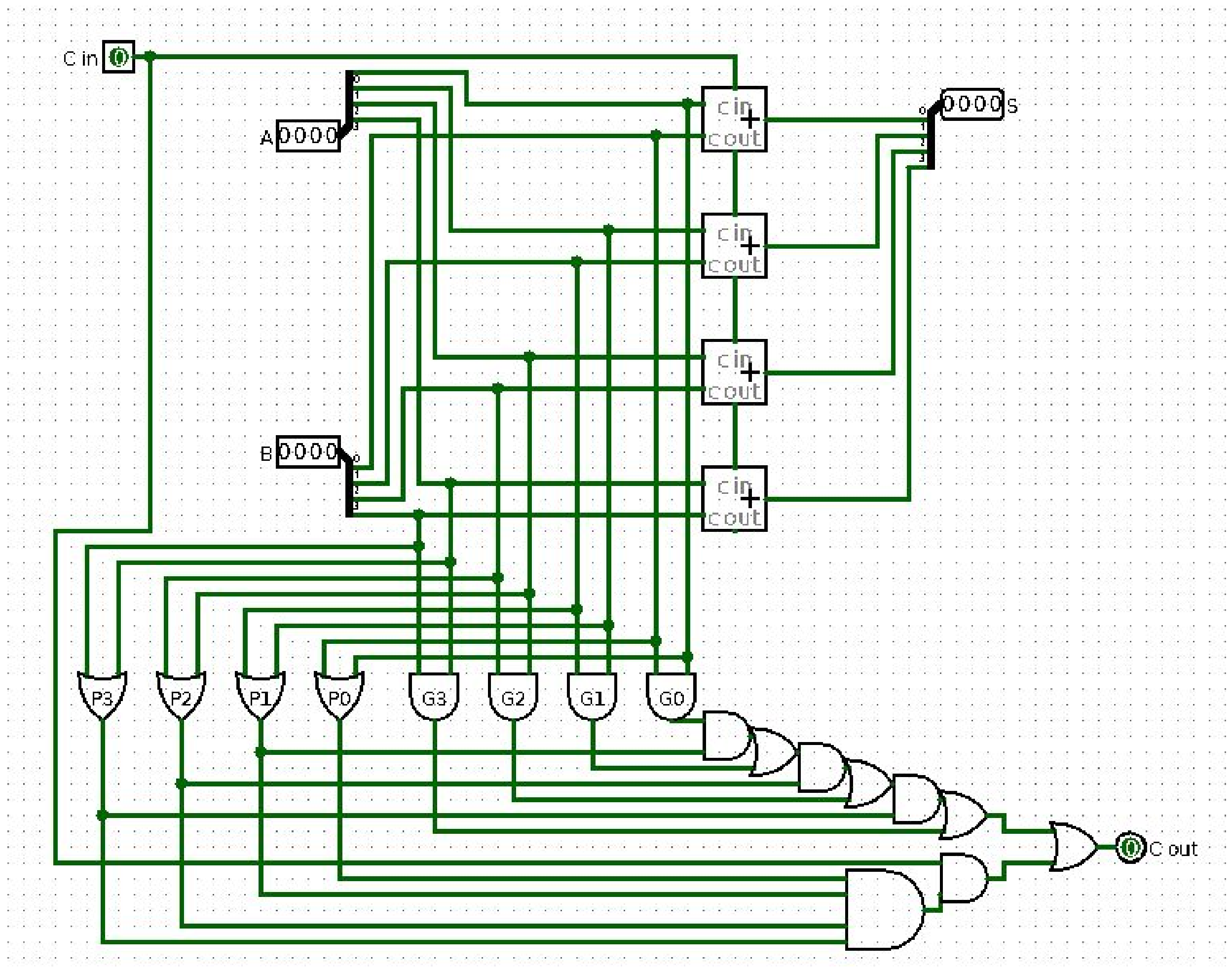
Branch-if-equal Proof:
The next two images show that my processor can process both forward and backwards branches from the testbranch.asm and testbackbranches.asm tests.

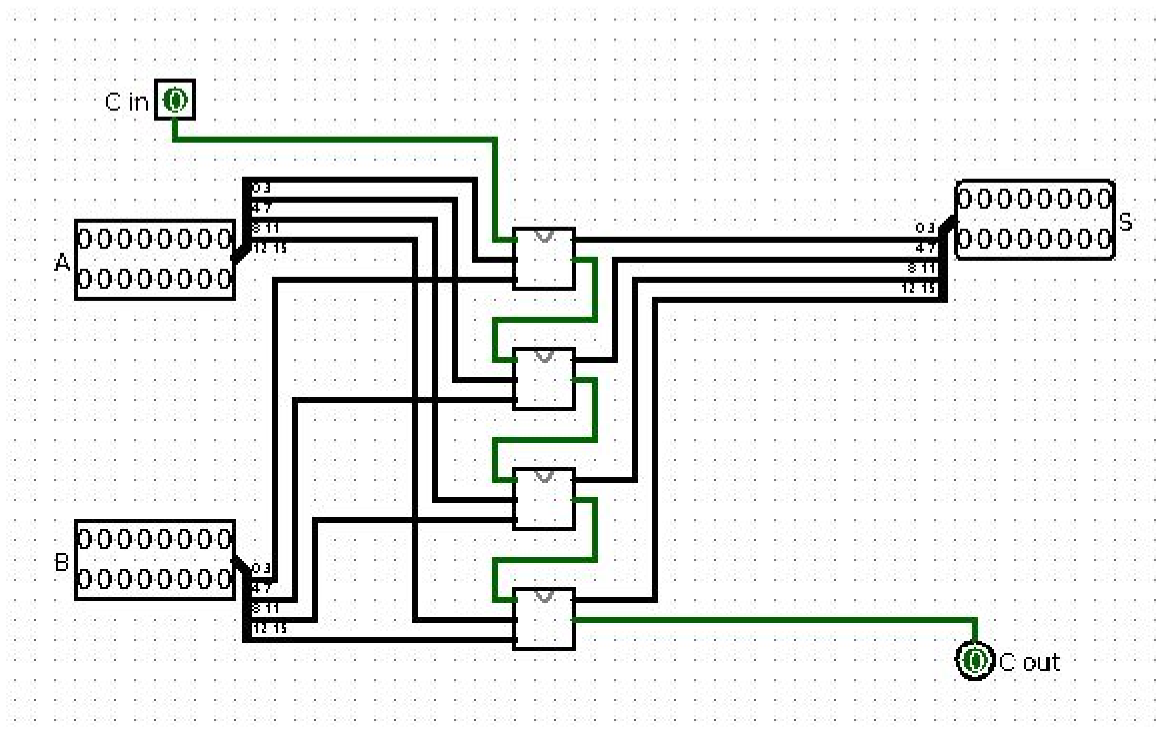


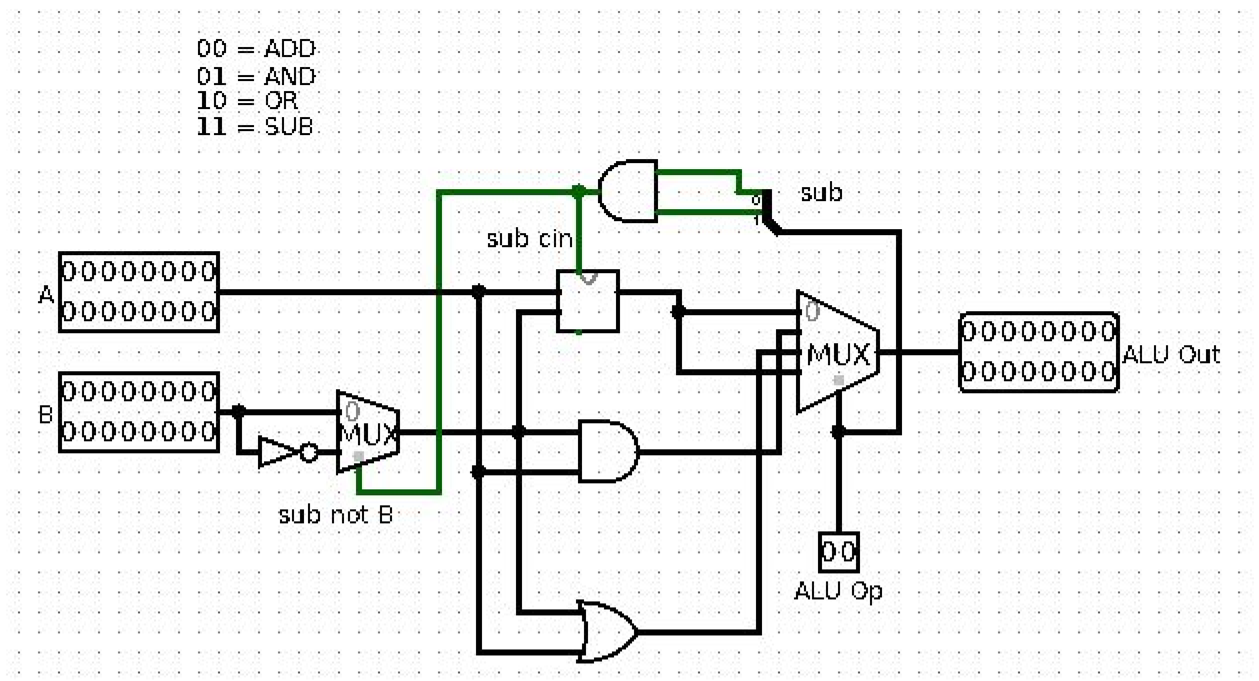


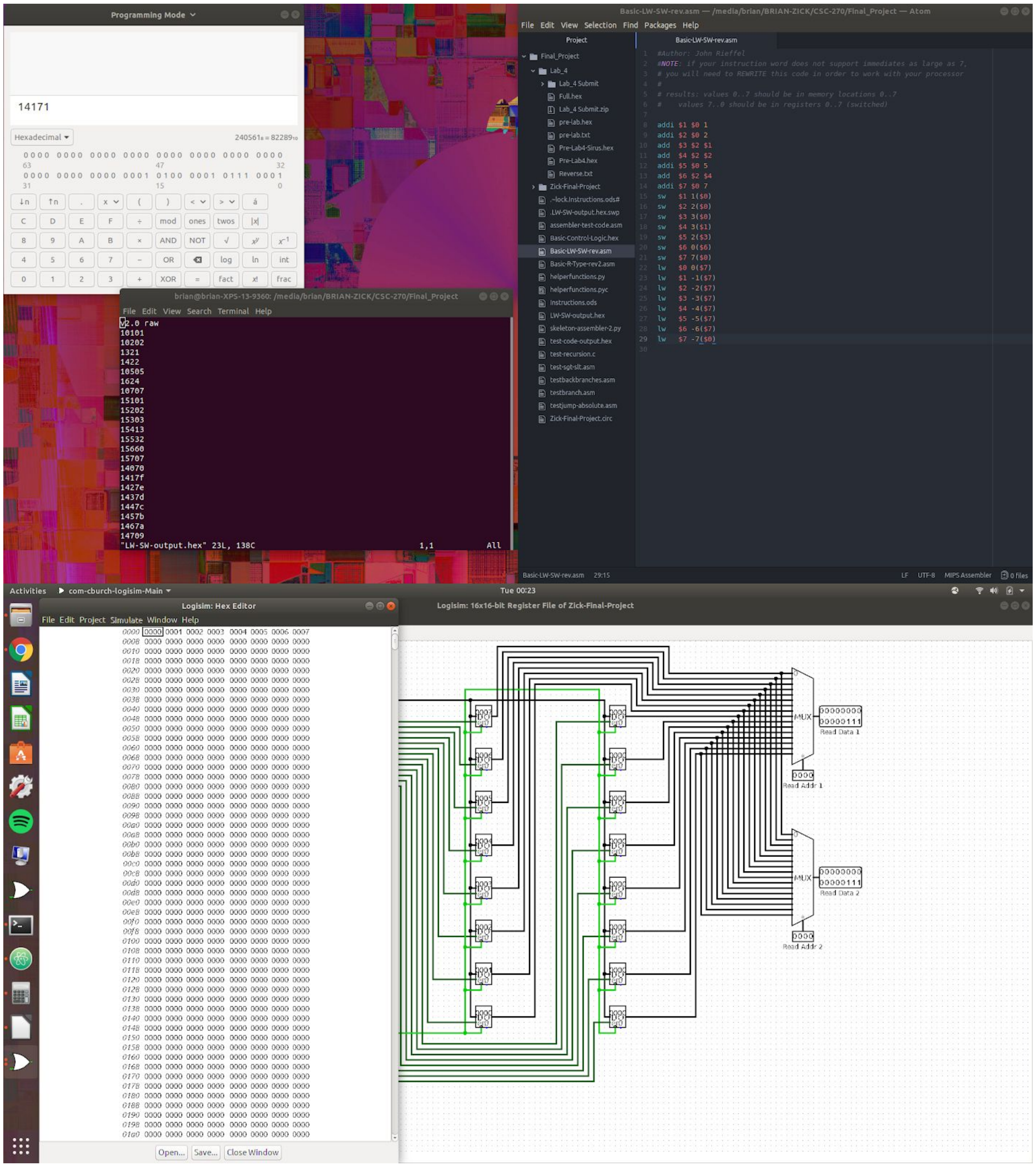
Carry Look-Ahead Adder Proof:

The first image contains a picture of my 4-bit Carry Look-Ahead (CLA) block which is used in the construction of the adder, shown in the second image. The ALU containing this adder is shown in the third image while the fourth image shows proof that the adder functions correctly using the test code in Basic-LW-SW-rev.asm.






00 = ADD
01 = AND
10 = OR
11 = SUB



Basic-LW-SW-rev.asm — /media/brian/BRIAN-ZICK/CSC-270/Final_Project — Atom

```
#Author: John Rieffel
#NOTE: if your instruction word does not support immediates as large as 7,
# you will need to REWRITE this code in order to work with your processor
#
# results: values 0..7 should be in memory locations 0..7
#    values 7..0 should be in registers 0..7 (switched)

addi $1 $0 1
addi $2 $0 2
add  $3 $2 $1
add  $4 $2 $2
addi $5 $0 5
add  $6 $2 $4
addi $7 $0 7
sw   $1 1($0)
sw   $2 2($0)
sw   $3 3($0)
sw   $4 3($1)
sw   $5 2($3)
sw   $6 0($6)
sw   $7 7($0)
lw   $0 0($7)
lw   $1 -1($7)
lw   $2 -2($7)
lw   $3 -3($7)
lw   $4 -4($7)
lw   $5 -5($7)
lw   $6 -6($7)
lw   $7 -7($0)
```

```
Final_Project
  Lab_4
    Lab_4 Submit
    Full.hex
    Lab_4 Submit.zip
    pre-lab.hex
    pre-lab.txt
    Pre-Lab4-Sirus.hex
    Pre-Lab4.hex
    Reverse.txt
  Zick-Final-Project
  .~lock.Instructions.ods#
  .LW-SW-output.hex.swp
  assembler-test-code.asm
  Basic-Control-Logic.hex
  Basic-LW-SW-rev.asm
  Basic-R-Type-rev2.asm
  helperfunctions.py
  helperfunctions.pyc
  Instructions.ods
  LW-SW-output.hex
  skeleton-assembler-2.py
  test-code-output.hex
  test-recursion.c
  test-sgt-slt.asm
  testbackbranches.asm
  testbranch.asm
  testjump-absolute.asm
  Zick-Final-Project.circ
```

brian@brian-XPS-13-9360: /media/brian/BRIAN-ZICK/CSC-270/Final_Project

```
v2.0 raw
10101
10202
1321
1422
10505
1624
10707
15101
15202
15303
15413
15532
15660
15707
14070
1417f
1427e
1437d
1447c
1457b
1467a
14709
"LW-SW-output.hex" 23L, 138C                          1,1          All
```

Logisim: Hex Editor

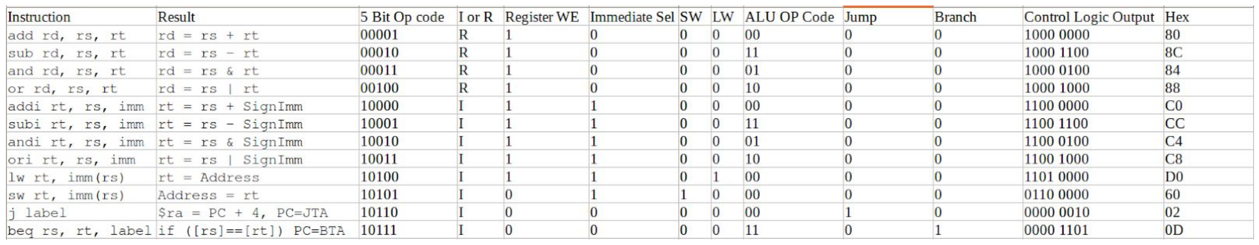
```
0000 0000 0001 0002 0003 0004 0005 0006 0007
0008 0000 0000 0000 0000 0000 0000 0000 0000
0010 0000 0000 0000 0000 0000 0000 0000 0000
0018 0000 0000 0000 0000 0000 0000 0000 0000
0020 0000 0000 0000 0000 0000 0000 0000 0000
0028 0000 0000 0000 0000 0000 0000 0000 0000
0030 0000 0000 0000 0000 0000 0000 0000 0000
0038 0000 0000 0000 0000 0000 0000 0000 0000
0040 0000 0000 0000 0000 0000 0000 0000 0000
0048 0000 0000 0000 0000 0000 0000 0000 0000
0050 0000 0000 0000 0000 0000 0000 0000 0000
0058 0000 0000 0000 0000 0000 0000 0000 0000
0060 0000 0000 0000 0000 0000 0000 0000 0000
0068 0000 0000 0000 0000 0000 0000 0000 0000
0070 0000 0000 0000 0000 0000 0000 0000 0000
0078 0000 0000 0000 0000 0000 0000 0000 0000
0080 0000 0000 0000 0000 0000 0000 0000 0000
0088 0000 0000 0000 0000 0000 0000 0000 0000
0090 0000 0000 0000 0000 0000 0000 0000 0000
0098 0000 0000 0000 0000 0000 0000 0000 0000
00a0 0000 0000 0000 0000 0000 0000 0000 0000
00a8 0000 0000 0000 0000 0000 0000 0000 0000
00b0 0000 0000 0000 0000 0000 0000 0000 0000
00b8 0000 0000 0000 0000 0000 0000 0000 0000
00c0 0000 0000 0000 0000 0000 0000 0000 0000
00c8 0000 0000 0000 0000 0000 0000 0000 0000
00d0 0000 0000 0000 0000 0000 0000 0000 0000
00d8 0000 0000 0000 0000 0000 0000 0000 0000
00e0 0000 0000 0000 0000 0000 0000 0000 0000
00e8 0000 0000 0000 0000 0000 0000 0000 0000
00f0 0000 0000 0000 0000 0000 0000 0000 0000
00f8 0000 0000 0000 0000 0000 0000 0000 0000
0100 0000 0000 0000 0000 0000 0000 0000 0000
0108 0000 0000 0000 0000 0000 0000 0000 0000
0110 0000 0000 0000 0000 0000 0000 0000 0000
0118 0000 0000 0000 0000 0000 0000 0000 0000
0120 0000 0000 0000 0000 0000 0000 0000 0000
0128 0000 0000 0000 0000 0000 0000 0000 0000
0130 0000 0000 0000 0000 0000 0000 0000 0000
0138 0000 0000 0000 0000 0000 0000 0000 0000
0140 0000 0000 0000 0000 0000 0000 0000 0000
0148 0000 0000 0000 0000 0000 0000 0000 0000
0150 0000 0000 0000 0000 0000 0000 0000 0000
0158 0000 0000 0000 0000 0000 0000 0000 0000
0160 0000 0000 0000 0000 0000 0000 0000 0000
0168 0000 0000 0000 0000 0000 0000 0000 0000
0170 0000 0000 0000 0000 0000 0000 0000 0000
0178 0000 0000 0000 0000 0000 0000 0000 0000
0180 0000 0000 0000 0000 0000 0000 0000 0000
0188 0000 0000 0000 0000 0000 0000 0000 0000
0190 0000 0000 0000 0000 0000 0000 0000 0000
0198 0000 0000 0000 0000 0000 0000 0000 0000
01a0 0000 0000 0000 0000 0000 0000 0000 0000
```

Design Decisions:

I decided to make a 16-bit processor with 16 registers to allow for up to 4-bit immediate values and more than enough registers for most programs. In addition, I used a control logic lookup table from the beginning to ease the addition of various features. In addition, I modified my assembler to insert a no-op before the first instruction to ensure that the first instruction gets full executed, and therefore also modified jump instructions accordingly to maintain their original target.Most other decisions were done by recreating the MIPS datapath from the textbook, e.g. branch and jump, as it is better to not fix that which is not yet broken.

Machine Code Instruction Format:
My machine code is a 17-bit instruction word beginning with a 5-bit op-code, the address of that operation in the lookup table, followed by three 4-bit values, rd, rs and rt which are used in various ways as register addresses, immediate values and can even be combined into a 12-bit immediate value for jump instructions.

A mapping of assembly code instructions to machine code can be seen in this table



Instruction	Result	5 Bit Op code	I or R	Register WE	Immediate Sel	SW	LW	ALU OP Code	Jump	Branch	Control Logic Output	Hex
add rd, rs, rt	rd = rs + rt	00001	R	1	0	0	0	00	0	0	1000 0000	80
sub rd, rs, rt	rd = rs - rt	00010	R	1	0	0	0	11	0	0	1000 1100	8C
and rd, rs, rt	rd = rs & rt	00011	R	1	0	0	0	01	0	0	1000 0100	84
or rd, rs, rt	rd = rs | rt	00100	R	1	0	0	0	10	0	0	1000 1000	88
addi rt, rs, imm	rt = rs + SignImm	10000	I	1	1	0	0	00	0	0	1100 0000	C0
subi rt, rs, imm	rt = rs - SignImm	10001	I	1	1	0	0	11	0	0	1100 1100	CC
andi rt, rs, imm	rt = rs & SignImm	10010	I	1	1	0	0	01	0	0	1100 0100	C4
ori rt, rs, imm	rt = rs | SignImm	10011	I	1	1	0	0	10	0	0	1100 1000	C8
lw rt, imm(rs)	rt = Address	10100	I	1	1	0	1	00	0	0	1101 0000	D0
sw rt, imm(rs)	Address = rt	10101	I	0	1	1	0	00	0	0	0110 0000	60
j label	$ra = PC + 4, PC=JTA	10110	I	0	0	0	0	00	1	0	0000 0010	02
beq rs, rt, label	if ([rs]==[rt]) PC=BTA	10111	I	0	0	0	0	11	0	1	0000 1101	0D

which is from the attached spreadsheet further detailing this aspect of the project.

To translate assembly code into hexadecimal machine code, one must proceed line by line through an assembly file following a pseudo-MIPS style. Each instruction is on its own line with each word of the instruction separated only by spaces, no commas. All comments being with a '#' symbol. For each line of this assembly language, we first strip off anything that is a comment. Then we split the line on spaces to separate the operation from the arguments. The operation is translated into assembly using a lookup table of opcodes which the arguments are individually translated into binary and read in. Then each part of the line is concatenated and the whole thing is converted into hex from binary and written into the output .hex file. Then the process begins again with the next line.