Learning Activity 3

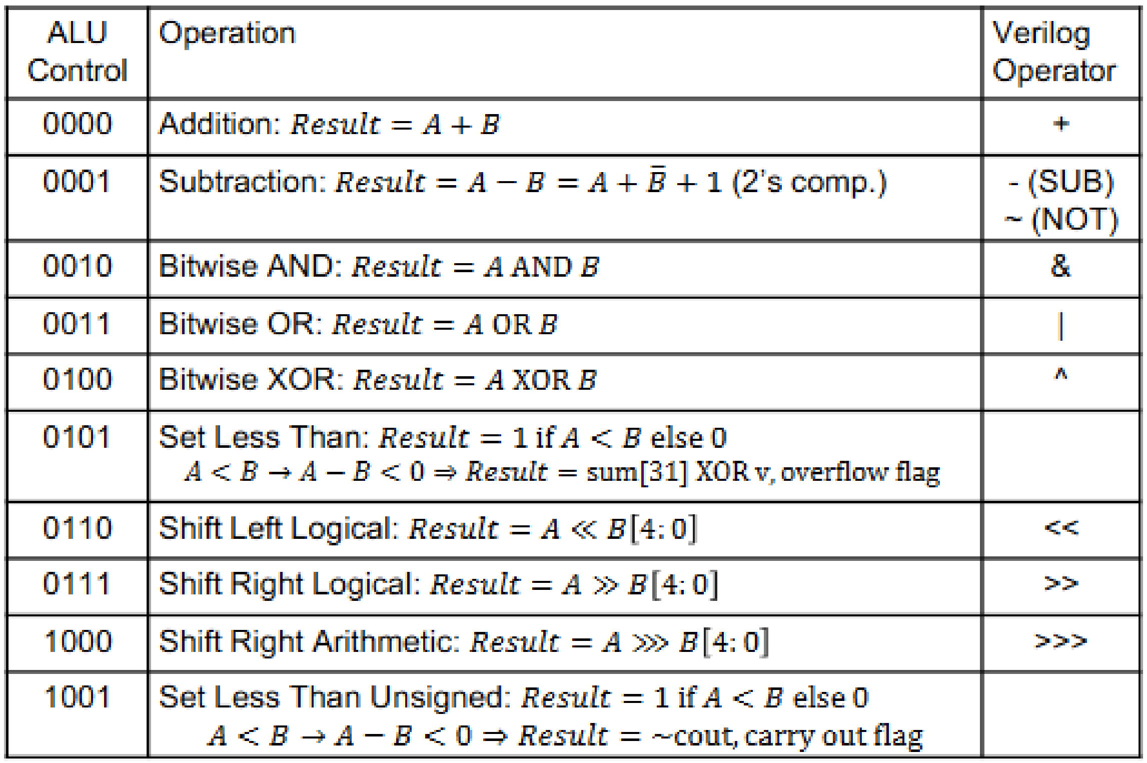
By Alex Clunan

ECE 4435 Computer Architecture & Design

February 24, 2025

1 Problem Statement

Design and test an ALU that implements the following 32 bit functions in Fig 1:



ALU Control	Operation	Verilog Operator
0000	Addition: $Result = A + B$	+
0001	Subtraction: $Result = A - B = A + \bar{B} + 1$ (2's comp.)	- (SUB) ~ (NOT)
0010	Bitwise AND: $Result = A \text{ AND } B$	&
0011	Bitwise OR: $Result = A \text{ OR } B$	|
0100	Bitwise XOR: $Result = A \text{ XOR } B$	^
0101	Set Less Than: $Result = 1 \text{ if } A < B \text{ else } 0$ $A < B \rightarrow A - B < 0 \Rightarrow Result = \text{sum}[31] \text{ XOR v, overflow flag}$	
0110	Shift Left Logical: $Result = A \ll B[4{:}0]$	<<
0111	Shift Right Logical: $Result = A \gg B[4{:}0]$	>>
1000	Shift Right Arithmetic: $Result = A \ggg B[4{:}0]$	>>>
1001	Set Less Than Unsigned: $Result = 1 \text{ if } A < B \text{ else } 0$ $A < B \rightarrow A - B < 0 \Rightarrow Result = \sim\text{cout, carry out flag}$	

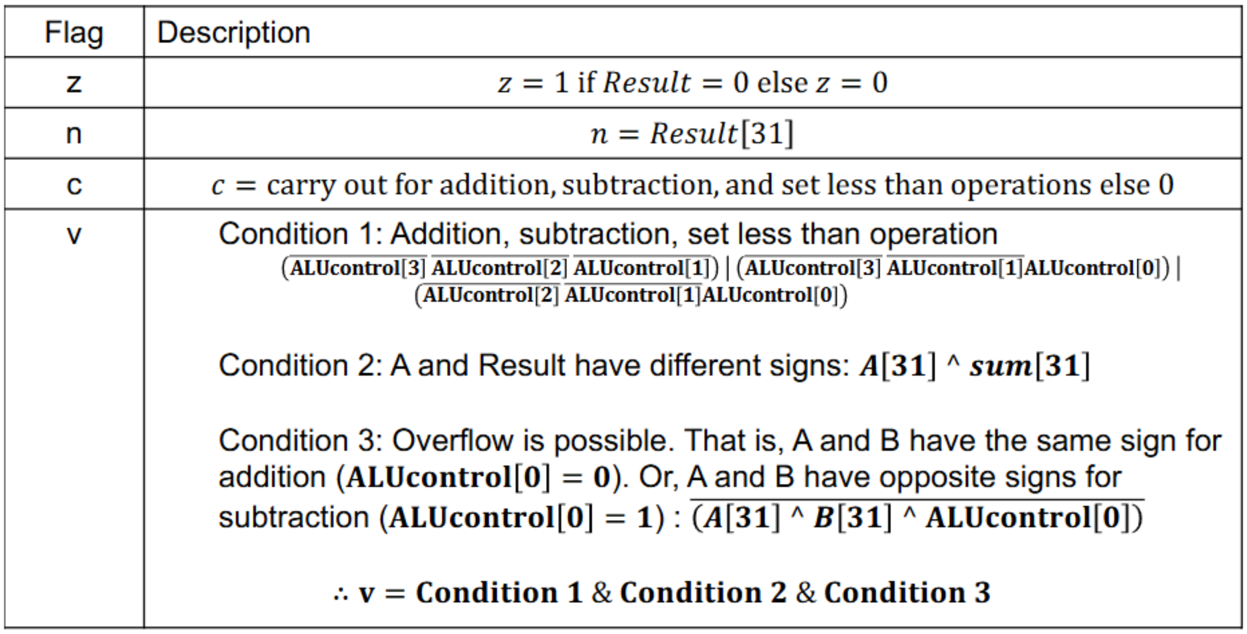
Fig 1. ALU Operations (from class slides)

The ALU should also generate zero, negative, carry out, and overflow flags. Then write assembly for the load_register_values function which loads the value of the name of the register into itself. I.e. r10 = 0xA after the function.

2 Analytical Design

The approach taken to design this ALU was to generate the value of each function, and then select the correct data path using the ALUcontrol signal and muxes. The ADD, SUBTRACT, AND, OR, XOR, SLL, SRL, SRA, and LESS THAN functions are all generated at the start of the data path. These are all fed into muxes that eventually output the result of the ALU. The inputs to the same muxes share similar bits in the lower part of the ALUcontrol signal. For example, the AND and OR functions have the same top bits, but the LSBs are different. Those signals were combined using a mux, and the rest of the signals were combined in a similar way. This resulted in the result output being set correctly.

Next, the flags were set. The flags are stored in a 4 bit vector with overflow, carry, negative, and zero flags in order from the MSB to the LSB. The zero flag is set by inverting the result and anding all the singular bits in that. The negative flag is set by checking if the MSB of the result is 1. The carry out flag is set by calculating if there is a carry in the add/subtract module and then anding that output with certain bits of the ALUcontrol signal to check if an addition, subtraction, or comparison is running. The overflow flag also checks the ALUcontrol signal for the same value, and also checks for a few more values. A table showing the flag assignments is shown in Fig 2.



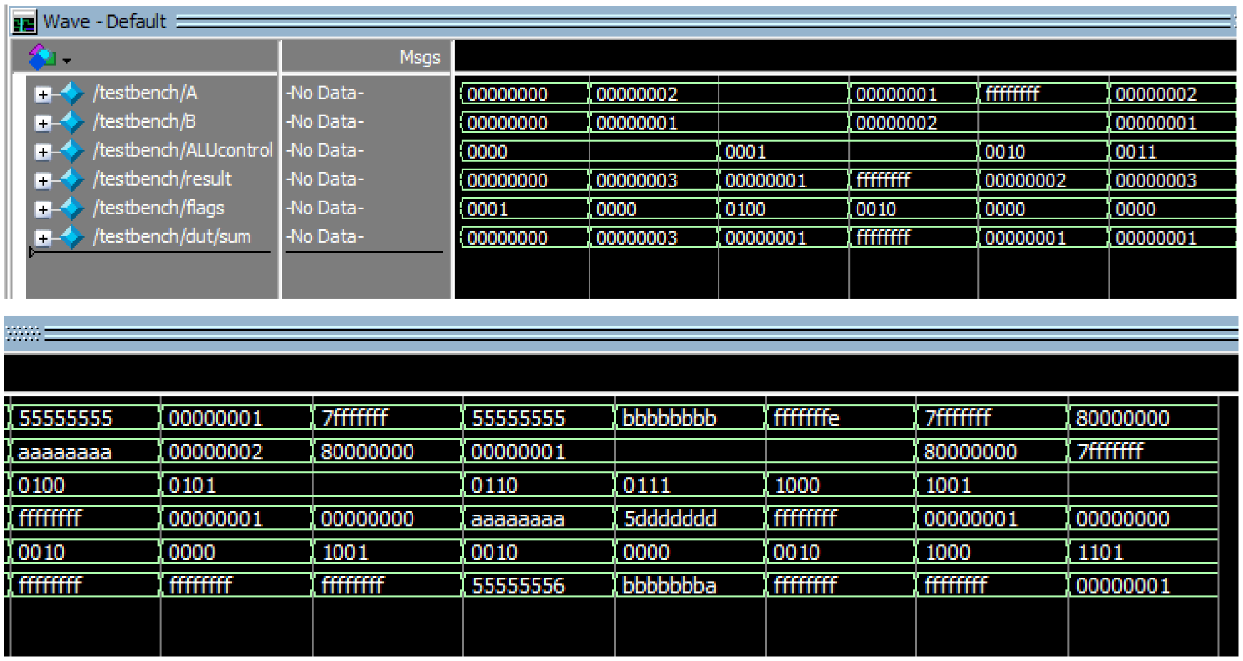
Flag	Description
z	$z = 1 \text{ if } Result = 0 \text{ else } z = 0$
n	$n = Result[31]$
c	$c = \text{carry out for addition, subtraction, and set less than operations else } 0$
v	Condition 1: Addition, subtraction, set less than operation $(\overline{\mathbf{ALUcontrol[3]}}\ \overline{\mathbf{ALUcontrol[2]}}\ \overline{\mathbf{ALUcontrol[1]}}) \mid (\overline{\mathbf{ALUcontrol[3]}}\ \overline{\mathbf{ALUcontrol[1]}}\mathbf{ALUcontrol[0]}) \mid (\overline{\mathbf{ALUcontrol[2]}}\ \overline{\mathbf{ALUcontrol[1]}}\mathbf{ALUcontrol[0]})$ Condition 2: A and Result have different signs: $\boldsymbol{A[31]} \wedge \boldsymbol{sum[31]}$ Condition 3: Overflow is possible. That is, A and B have the same sign for addition ($\mathbf{ALUcontrol[0] = 0}$). Or, A and B have opposite signs for subtraction ($\mathbf{ALUcontrol[0] = 1}$) : $\overline{(\boldsymbol{A[31]} \wedge \boldsymbol{B[31]} \wedge \mathbf{ALUcontrol[0]})}$ $\therefore \mathbf{v = Condition\ 1\ \&\ Condition\ 2\ \&\ Condition\ 3}$

Fig 2. Flag Conditions (from class slides)

Finally, some assembly code was generated to test the load_register_values function.The code sets all usable registers to their respective values (i.e. r2 = 2) using the ALU instructions in Fig1. The code was tested in the RARs 1.6 RISC-V simulator.

3 Numerical Verification

The ALU was tested using a non-exhaustive testbench. The testbench checked both the result and flag values, and outputted a message to the console for every test. The messages were generated using the assert function given in testbenches on previous assignments. The waveform is shown in Fig 3.

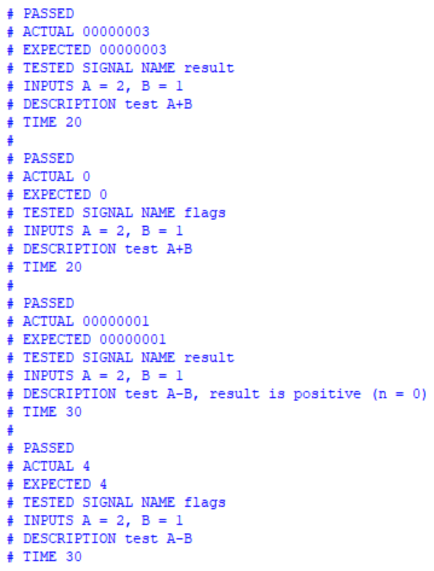


```
/testbench/A           -No Data-  00000000  00000002            00000001  ffffffff  00000002
/testbench/B           -No Data-  00000000  00000001            00000002            00000001
/testbench/ALUcontrol  -No Data-  0000                0001                0010      0011
/testbench/result      -No Data-  00000000  00000003  00000001  ffffffff  00000002  00000003
/testbench/flags       -No Data-  0001      0000      0100      0010      0000      0000
/testbench/dut/sum     -No Data-  00000000  00000003  00000001  ffffffff  00000001  00000001
```

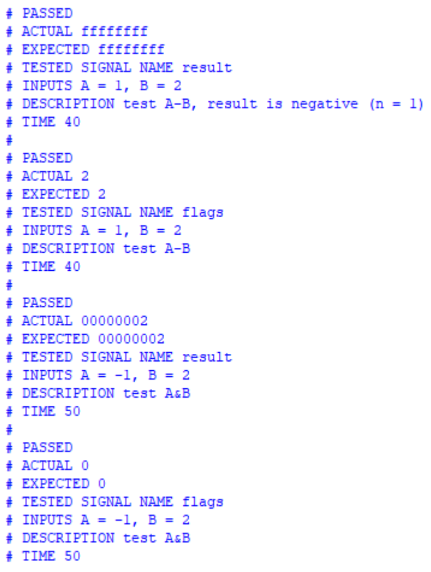
```
55555555  00000001  7fffffff  55555555  bbbbbbbb  fffffffe  7fffffff  80000000
aaaaaaaa  00000002  80000000  00000001                      80000000  7fffffff
0100      0101                0110      0111      1000      1001
ffffffff  00000001  00000000  aaaaaaaa  5ddddddd  ffffffff  00000001  00000000
0010      0000      1001      0010      0000      0010      1000      1101
ffffffff  ffffffff  ffffffff  55555556  bbbbbbba  ffffffff  ffffffff  00000001
```

Fig 3. Waveform of Testbench Output

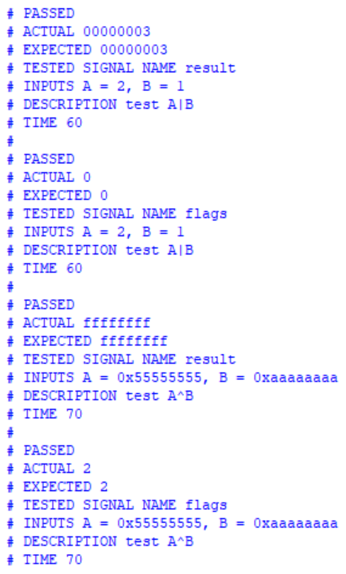
The console output that checks the expected versus actual output is shown in Fig 4.



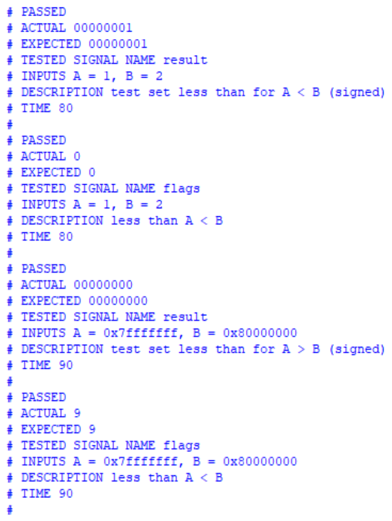
```
# PASSED
# ACTUAL 00000003
# EXPECTED 00000003
# TESTED SIGNAL NAME result
# INPUTS A = 2, B = 1
# DESCRIPTION test A+B
# TIME 20
#
# PASSED
# ACTUAL 0
# EXPECTED 0
# TESTED SIGNAL NAME flags
# INPUTS A = 2, B = 1
# DESCRIPTION test A+B
# TIME 20
#
# PASSED
# ACTUAL 00000001
# EXPECTED 00000001
# TESTED SIGNAL NAME result
# INPUTS A = 2, B = 1
# DESCRIPTION test A-B, result is positive (n = 0)
# TIME 30
#
# PASSED
# ACTUAL 4
# EXPECTED 4
# TESTED SIGNAL NAME flags
# INPUTS A = 2, B = 1
# DESCRIPTION test A-B
# TIME 30
```



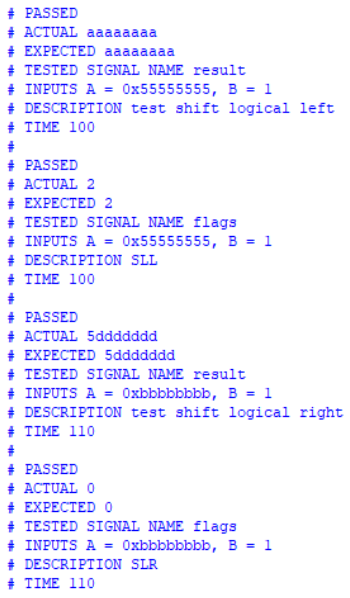
```
# PASSED
# ACTUAL fffffff
# EXPECTED fffffff
# TESTED SIGNAL NAME result
# INPUTS A = 1, B = 2
# DESCRIPTION test A-B, result is negative (n = 1)
# TIME 40
#
# PASSED
# ACTUAL 2
# EXPECTED 2
# TESTED SIGNAL NAME flags
# INPUTS A = 1, B = 2
# DESCRIPTION test A-B
# TIME 40
#
# PASSED
# ACTUAL 00000002
# EXPECTED 00000002
# TESTED SIGNAL NAME result
# INPUTS A = -1, B = 2
# DESCRIPTION test A&B
# TIME 50
#
# PASSED
# ACTUAL 0
# EXPECTED 0
# TESTED SIGNAL NAME flags
# INPUTS A = -1, B = 2
# DESCRIPTION test A&B
# TIME 50
```



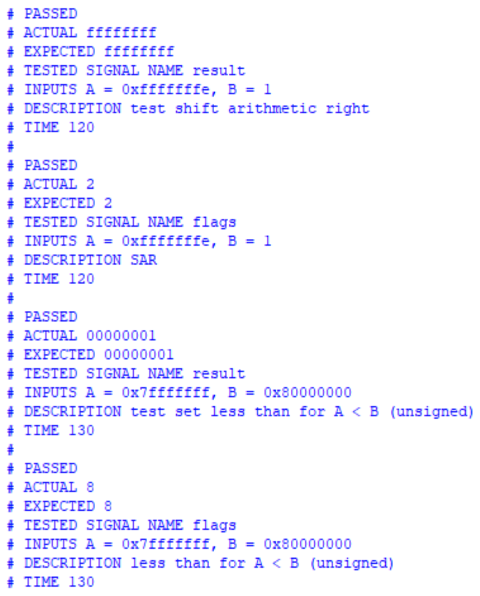
```
# PASSED
# ACTUAL 00000003
# EXPECTED 00000003
# TESTED SIGNAL NAME result
# INPUTS A = 2, B = 1
# DESCRIPTION test A|B
# TIME 60
#
# PASSED
# ACTUAL 0
# EXPECTED 0
# TESTED SIGNAL NAME flags
# INPUTS A = 2, B = 1
# DESCRIPTION test A|B
# TIME 60
#
# PASSED
# ACTUAL fffffff
# EXPECTED fffffff
# TESTED SIGNAL NAME result
# INPUTS A = 0x55555555, B = 0xaaaaaaaa
# DESCRIPTION test A^B
# TIME 70
#
# PASSED
# ACTUAL 2
# EXPECTED 2
# TESTED SIGNAL NAME flags
# INPUTS A = 0x55555555, B = 0xaaaaaaaa
# DESCRIPTION test A^B
# TIME 70
```



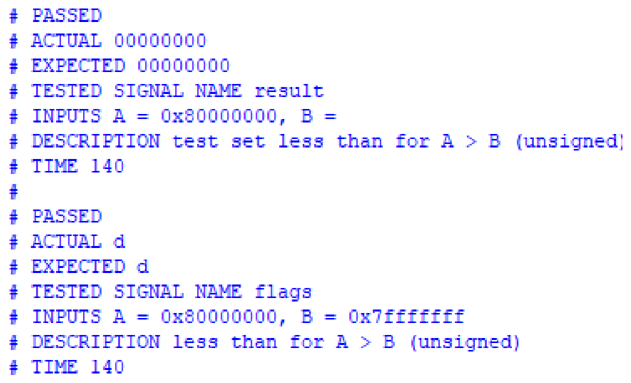
```
# PASSED
# ACTUAL 00000001
# EXPECTED 00000001
# TESTED SIGNAL NAME result
# INPUTS A = 1, B = 2
# DESCRIPTION test set less than for A < B (signed)
# TIME 80
#
# PASSED
# ACTUAL 0
# EXPECTED 0
# TESTED SIGNAL NAME flags
# INPUTS A = 1, B = 2
# DESCRIPTION less than A < B
# TIME 80
#
# PASSED
# ACTUAL 00000000
# EXPECTED 00000000
# TESTED SIGNAL NAME result
# INPUTS A = 0x7fffffff, B = 0x80000000
# DESCRIPTION test set less than for A > B (signed)
# TIME 90
#
# PASSED
# ACTUAL 9
# EXPECTED 9
# TESTED SIGNAL NAME flags
# INPUTS A = 0x7fffffff, B = 0x80000000
# DESCRIPTION less than A < B
# TIME 90
#
```



```
# PASSED
# ACTUAL aaaaaaaa
# EXPECTED aaaaaaaa
# TESTED SIGNAL NAME result
# INPUTS A = 0x55555555, B = 1
# DESCRIPTION test shift logical left
# TIME 100
#
# PASSED
# ACTUAL 2
# EXPECTED 2
# TESTED SIGNAL NAME flags
# INPUTS A = 0x55555555, B = 1
# DESCRIPTION SLL
# TIME 100
#
# PASSED
# ACTUAL 5ddddddd
# EXPECTED 5ddddddd
# TESTED SIGNAL NAME result
# INPUTS A = 0xbbbbbbbb, B = 1
# DESCRIPTION test shift logical right
# TIME 110
#
# PASSED
# ACTUAL 0
# EXPECTED 0
# TESTED SIGNAL NAME flags
# INPUTS A = 0xbbbbbbbb, B = 1
# DESCRIPTION SLR
# TIME 110
```



```
# PASSED
# ACTUAL fffffff
# EXPECTED fffffff
# TESTED SIGNAL NAME result
# INPUTS A = 0xfffffffe, B = 1
# DESCRIPTION test shift arithmetic right
# TIME 120
#
# PASSED
# ACTUAL 2
# EXPECTED 2
# TESTED SIGNAL NAME flags
# INPUTS A = 0xfffffffe, B = 1
# DESCRIPTION SAR
# TIME 120
#
# PASSED
# ACTUAL 00000001
# EXPECTED 00000001
# TESTED SIGNAL NAME result
# INPUTS A = 0x7fffffff, B = 0x80000000
# DESCRIPTION test set less than for A < B (unsigned)
# TIME 130
#
# PASSED
# ACTUAL 8
# EXPECTED 8
# TESTED SIGNAL NAME flags
# INPUTS A = 0x7fffffff, B = 0x80000000
# DESCRIPTION less than for A < B (unsigned)
# TIME 130
```

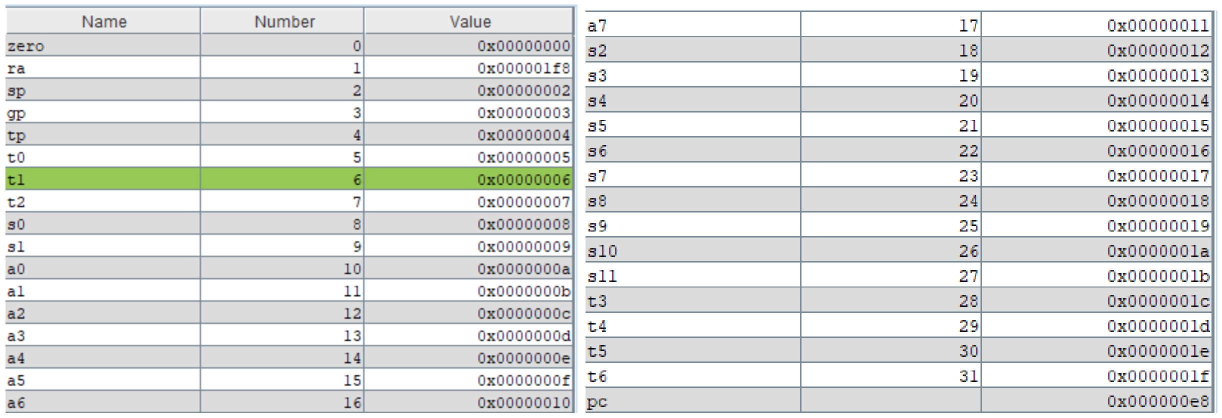


```
# PASSED
# ACTUAL 00000000
# EXPECTED 00000000
# TESTED SIGNAL NAME result
# INPUTS A = 0x80000000, B =
# DESCRIPTION test set less than for A > B (unsigned)
# TIME 140
#
# PASSED
# ACTUAL d
# EXPECTED d
# TESTED SIGNAL NAME flags
# INPUTS A = 0x80000000, B = 0x7fffffff
# DESCRIPTION less than for A > B (unsigned)
# TIME 140
```

Fig 4. Console Output of Testbench Tests

These outputs show that for all the tests, all the results and flags match the expected values.

Next, the assembly program was tested. Fig 5 shows the status of the registers after all the values are set.



Name	Number	Value
zero	0	0x00000000
ra	1	0x000001f8
sp	2	0x00000002
gp	3	0x00000003
tp	4	0x00000004
t0	5	0x00000005
t1	6	0x00000006
t2	7	0x00000007
s0	8	0x00000008
s1	9	0x00000009
a0	10	0x0000000a
a1	11	0x0000000b
a2	12	0x0000000c
a3	13	0x0000000d
a4	14	0x0000000e
a5	15	0x0000000f
a6	16	0x00000010
a7	17	0x00000011
s2	18	0x00000012
s3	19	0x00000013
s4	20	0x00000014
s5	21	0x00000015
s6	22	0x00000016
s7	23	0x00000017
s8	24	0x00000018
s9	25	0x00000019
s10	26	0x0000001a
s11	27	0x0000001b
t3	28	0x0000001c
t4	29	0x0000001d
t5	30	0x0000001e
t6	31	0x0000001f
pc		0x000000e8

Fig 5. Registers x0 - x31 Loaded with their Respective Register Numbers

These show that all the registers have their respective numbers loaded in which verifies the program.

4 Summary

The requirement for this assignment was to design an ALU with designated functions and flags, and to generate assembly code to test those functions.

The logic functions for the ALU were created, and then were connected to the output using a series of muxes. The flags were generated using combinatorial logic based on certain logical values. Finally assembly was created to test these functions using the RISC-V instruction set. “The circuit was implemented using Verilog, a hardware implementation was synthesized from the Verilog implementation, and finally the design was verified numerically to confirm that the specified digital circuit design solution satisfies the problem requirements.” (from Example Submission for learning activity 0)