Test Cases

Overall Note: Not all numbers have a 16b pre-fix because some are already written as 16 bits long. If that's the case, it doesn't make a difference whether we write 16b in front or not.
i.e. 16b0 = 0000000000000000

Outputs

- alu.out – 16 bits output
- alu.z – check all output is 0
- alu.v – check overflow
- alu.n (S_{31})– check negative output

Overflow occurs when the two operands have the same sign, but the result has a sign opposite of the operands. i.e. 1) Pos + Pos = Neg or 2) Neg + Neg = Pos

ADD – 6b0

Case	a	b	result
positive + positive	16b10 = 2	16b11110 = 30	**(alu.out)** 16b0000000000100000 = 32 **(alu.z) = 0** **(alu.v) = 0** **(alu.n) = 0**
positive + positive (overflow)	16b0111111111111111 (15 1's)	16b0111111111111111 (15 1's)	**(alu.out)** 16b1111111111111110 = overflow **(alu.z) = 0** **(alu.v) = 1** **(alu.n) = 1**
Boundary	16b0111111111111111 = 32767	16b0 = 0	**(alu.out)** 16b0111111111111111 = 32767 **(alu.z) = 0** **(alu.v) = 0** **(alu.n) = 0**
negative + positive	1111111111111111 = -1	0111111111111111 = 32767	**(alu.out)** 16b0111111111111110 = 32766 **(alu.z) = 0** **(alu.v) = 0** **(alu.n) = 0**
negative + negative	1111111111111111 = -1	1111111111111111 = -1	**(alu.out)** 1111111111111110 = -2 **(alu.z) = 0** **(alu.v) = 0** **(alu.n) = 1**

SUB – 6b1

Case	a	b	result
positive - positive	16b10 = 2	16b11110 = 30	**(alu.out)** 16b1111111111100100 = -28 **(alu.z) = 0** **(alu.v) = 0** **(alu.n) = 1**
Positive - negative	16b0111111111111111 = 32767	16b1111111111111111 = -1	**(alu.out)** 16b1000000000000000 = 32768 **(alu.z) = 0** **(alu.v) = 1** **(alu.n) = 1**
negative - positive (Boundary)	1111111111111111 = -1	0111111111111111 = 32767	**(alu.out)** 16b1000000000000000 = -32768 **(alu.z) = 0** **(alu.v) = 0** **(alu.n) = 1**
negative - negative	1111111111111111 = -1	1111111111111111 = -1	**(alu.out)** 16b0 = 0 **(alu.z) = 1** **(alu.v) = 0** **(alu.n) = 0**
Overflow	16b1011111111111111 = -16385	16b0100000000000000 = 16384	**(alu.out)** 10111111111111111 (17 bit output) = -32769 **(alu.z) = 0** **(alu.v) = 1** **(alu.n) = 0**

AND – 6b011000

Output will be 'high' if and only if all inputs are 'high'; every matching digit (if there are)

a	b	result (alu.out)
16b111	16b111	16b111 (true)

OR – 6b011110

Output will be 'high' if every ith digit of either a and b is 1

a	b	result (alu.out)
16b101	16b011	16b111

XOR – 6b010110

Output will be high if only one of the ith digit of the two 16-bit inputs are 1

Case	a	b	result (alu.out)
true OR; false XOR	16b110	16b111	16b001

LDR – 6b011010

Returns the value of A regardless of B

a	b	result (alu.out)
16b0	16b111111	16b0

Shifting: a is the number we are shifting. b is the number of bits we are shifting left/right by. (b is 16 bits but the maximum number of bits we can shift by is 16)

SHL – 6b100000

a	b	result (alu.out)
16b1111	16b010 = 2	16b111100

SHR – 6b100001

a	b	result (alu.out)
16b1111	16b010 = 2	16b0011

SRA – 6b100011

Pad left bits of original number with a[16]

a	b	result (alu.out)
16b1000000000000000	16b010 = 2	16b1110000000000000

CMPEQ – 110011

Output is ‘high’ iff a==b

Case	a	b	result (alu.out)
# bits in a and b are equal; $a = b$	16b1100	16b1100	16b1 (true)
Different # bits $a \neq b$	16b11100	161100	16b0 (false)

CMPLT – 110101

Output is ‘high’ iff a<b

Case	a	b	result (alu.out)
$a < b$	16b1100	16b1111	16b1 (true)
$a = b$	16b1100	16b1100	16b0 (false)

CMPLE – 110111

Output is ‘high’ iff a<=b

Case	a	b	result (alu.out)
$a < b$	16b1100	16b1111	16b1 (true)
$a = b$	16b1100	16b1100	16b1 (true)

MULTIPLY – 6b000010

Case	a	b	result (alu.out)
positive x positive	16b11 = 3	16b11 = 3	0000000000001001 = 9
negative x negative	1111111111111101 = -3	1111111111111101 = -3	0000000000001001 = 9
positive x negative	1111111111111101 = -3	16b11 = 3	1111111111110111 = -9
0 x negative	16b0	1111111111111101	16b0