

Wencong Huang (Team Member A)

Jared Orange (Team Member B),

Marcos Buznego (Team Member C),

CS 2200: Theory of Computer Science

HW 2

September 20th 2018

Problem 1

| And.cmp | | | | | |
|---------|--|---|--|---|-----|
| 1 | | a | | b | out |
| 2 | | 0 | | 0 | 0 |
| 3 | | 0 | | 1 | 0 |
| 4 | | 1 | | 0 | 0 |
| 5 | | 1 | | 1 | 1 |

```
6  /**
7   * And gate:
8   * out = 1 if (a == 1 and b == 1)
9   *      0 otherwise
10  */
11
12  CHIP And {
13    ... IN a, b;
14    ... OUT out;
15
16    ... PARTS:
17    ... Nand(a=a,b=b,out=nout);
18    ... Not(in=nout,out=out);
19  }
```

```

6  load And.hdl,
7  output-file And.out,
8  compare-to And.cmp,
9  output-list a%B3.1.3 b%B3.1.3 out%B3.1.3;
10
11  set a 0,
12  set b 0,
13  eval,
14  output;
15
16  set a 0,
17  set b 1,
18  eval,
19  output;
20
21  set a 1,
22  set b 0,
23  eval,
24  output;
25
26  set a 1,
27  set b 1,
28  eval,
29  output;

```

| Not.cmp | | | | |
|---------|--|----|--|-----|
| 1 | | in | | out |
| 2 | | 0 | | 1 |
| 3 | | 1 | | 0 |
| 4 | | | | |

```

6  /**
7   * Not gate:
8   * out = not in
9   */
10
11  CHIP Not {
12      ... IN in;
13      ... OUT out;
14
15      ... PARTS:
16          ... // Put your code here:
17          ... Nand(a=in, b=in, out=out);
18  }

```

```

6  load Not.hdl,
7  output-file Not.out,
8  compare-to Not.cmp,
9  output-list in%B3.1.3 out%B3.1.3;
10
11  set in 0,
12  eval,
13  output;
14
15  set in 1,
16  eval,
17  output;

```

| Or.cmp | | | | | |
|--------|--|---|--|---|-----|
| 1 | | a | | b | out |
| 2 | | 0 | | 0 | 0 |
| 3 | | 0 | | 1 | 1 |
| 4 | | 1 | | 0 | 1 |
| 5 | | 1 | | 1 | 1 |

```

6  /**
7  * Or gate:
8  * out = 1 if (a == 1 or b == 1)
9  *      0 otherwise
10 */
11
12 CHIP Or {
13     ... IN a, b;
14     ... OUT out;
15
16     ... PARTS:
17     ... // Put your code here:
18     » Nand(a=a, b=a, out=notA);
19     » Nand(a=b, b=b, out=notB);
20     » Nand(a=notA, b=notB, out=out);
21 }

```

```

6  load Or.hdl,
7  output-file Or.out,
8  compare-to Or.cmp,
9  output-list a%B3.1.3 b%B3.1.3 out%B3.1.3;
10
11  set a 0,
12  set b 0,
13  eval,
14  output;
15
16  set a 0,
17  set b 1,
18  eval,
19  output;
20
21  set a 1,
22  set b 0,
23  eval,
24  output;
25
26  set a 1,
27  set b 1,
28  eval,
29  output;

```

| | <i>Xor.cmp</i> | | | | | |
|---|----------------|---|--|---|--|-----|
| 1 | | a | | b | | out |
| 2 | | 0 | | 0 | | 0 |
| 3 | | 0 | | 1 | | 1 |
| 4 | | 1 | | 0 | | 1 |
| 5 | | 1 | | 1 | | 0 |
| 6 | | | | | | |

```
6  /**  
7  * Exclusive-or gate:  
8  * out = not (a == b)  
9  */  
10  
11  CHIP Xor {  
12  ... IN a, b;  
13  ... OUT out;  
14  
15  ... PARTS:  
16  ... // Put your code here:  
17  ... Not(in=a, out=aNot);  
18  ... Not(in=b, out=bNot);  
19  ... And(a=aNot, b=b, out=out1);  
20  ... And(a=a, b=bNot, out=out2);  
21  ... Or(a=out1, b=out2, out=out);  
22  }
```

```
6  load Xor.hdl,  
7  output-file Xor.out,  
8  compare-to Xor.cmp,  
9  output-list a%B3.1.3 b%B3.1.3 out%B3.1.3;  
10   
11  set a 0,  
12  set b 0,  
13  eval,  
14  output;  
15   
16  set a 0,  
17  set b 1,  
18  eval,  
19  output;  
20   
21  set a 1,  
22  set b 0,  
23  eval,  
24  output;  
25   
26  set a 1,  
27  set b 1,  
28  eval,  
29  output;
```

| Mux.cmp | | | | | | |
|---------|---|---|------------|-----|--|---|
| 1 | a | b | <u>sel</u> | out | | ↵ |
| 2 | 0 | 0 | 0 | 0 | | ↵ |
| 3 | 0 | 0 | 1 | 0 | | ↵ |
| 4 | 0 | 1 | 0 | 0 | | ↵ |
| 5 | 0 | 1 | 1 | 1 | | ↵ |
| 6 | 1 | 0 | 0 | 1 | | ↵ |
| 7 | 1 | 0 | 1 | 0 | | ↵ |
| 8 | 1 | 1 | 0 | 1 | | ↵ |
| 9 | 1 | 1 | 1 | 1 | | ↵ |

```

6  /**
7   * Multiplexor:
8   * out = a if sel == 0
9   *      b otherwise
10  */
11
12  CHIP Mux {
13      ... IN a, b, sel;
14      ... OUT out;
15
16      ... PARTS:
17          // Put your code here:
18          ... Not(in=sel, out=nsel);
19          ... And(a=sel, b=b, out=c1);
20          ... And(a=nsel, b=a, out=c2);
21          ... Or(a=c1, b=c2, out=out);
22  }

```



```
6  load Mux.hdl,
7  output-file Mux.out,
8  compare-to Mux.cmp,
9  output-list a%B3.1.3 b%B3.1.3 sel%B3.1.3 out%B3.1.3;
10
11  set a 0,
12  set b 0,
13  set sel 0,
14  eval,
15  output;
16
17  set sel 1,
18  eval,
19  output;
20
21  set a 0,
22  set b 1,
23  set sel 0,
24  eval,
25  output;
26
27  set sel 1,
28  eval,
29  output;
30
31  set a 1,
32  set b 0,
33  set sel 0,
34  eval,
35  output;
36
37  set sel 1,
38  eval,
```

```
39  output; ↵
40  ↵
41  set a 1, ↵
42  set b 1, ↵
43  set sel 0, ↵
44  eval, ↵
45  output; ↵
46  ↵
47  set sel 1, ↵
48  eval, ↵
49  output; ↵
```

| And16.cmp | | | |
|-----------|------------------|------------------|------------------|
| 1 | a | b | out |
| 2 | 0000000000000000 | 0000000000000000 | 0000000000000000 |
| 3 | 0000000000000000 | 1111111111111111 | 0000000000000000 |
| 4 | 1111111111111111 | 1111111111111111 | 1111111111111111 |
| 5 | 1010101010101010 | 0101010101010101 | 0000000000000000 |
| 6 | 0011110011000011 | 0000111111110000 | 0000110011000000 |
| 7 | 0001001000110100 | 1001100001110110 | 000100000110100 |

```

6  /**
7   * 16-bit bitwise And:
8   * for i = 0..15: out[i] = (a[i] and b[i])
9   */
10
11 CHIP And16 {
12     ... IN a[16], b[16];
13     ... OUT out[16];
14
15     ... PARTS:
16         ... // Put your code here:
17         » And(a=a[0], b=b[0], out=out[0]);
18         » And(a=a[1], b=b[1], out=out[1]);
19         » And(a=a[2], b=b[2], out=out[2]);
20         » And(a=a[3], b=b[3], out=out[3]);
21         » And(a=a[4], b=b[4], out=out[4]);
22         » And(a=a[5], b=b[5], out=out[5]);
23         » And(a=a[6], b=b[6], out=out[6]);
24         » And(a=a[7], b=b[7], out=out[7]);
25         » And(a=a[8], b=b[8], out=out[8]);
26         » And(a=a[9], b=b[9], out=out[9]);
27         » And(a=a[10], b=b[10], out=out[10]);
28         » And(a=a[11], b=b[11], out=out[11]);
29         » And(a=a[12], b=b[12], out=out[12]);
30         » And(a=a[13], b=b[13], out=out[13]);
31         » And(a=a[14], b=b[14], out=out[14]);
32         » And(a=a[15], b=b[15], out=out[15]);
33     }

```

```
6  load And16.hdl,␣
7  output-file And16.out,␣
8  compare-to And16.cmp,␣
9  output-list a%B1.16.1 b%B1.16.1 out%B1.16.1;␣
10 ␣
11  set a %B0000000000000000,␣
12  set b %B0000000000000000,␣
13  eval,␣
14  output;␣
15 ␣
16  set a %B0000000000000000,␣
17  set b %B1111111111111111,␣
18  eval,␣
19  output;␣
20 ␣
21  set a %B1111111111111111,␣
22  set b %B1111111111111111,␣
23  eval,␣
24  output;␣
25 ␣
26  set a %B1010101010101010,␣
27  set b %B0101010101010101,␣
28  eval,␣
29  output;␣
30 ␣
31  set a %B0011110011000011,␣
32  set b %B0000111111110000,␣
33  eval,␣
34  output;␣
35 ␣
36  set a %B0001001000110100,␣
37  set b %B1001100001110110,␣
38  eval,␣
39  output;
```

| DMux.cmp | | | | | |
|----------|----|-----|---|---|--|
| 1 | in | sel | a | b | |
| 2 | 0 | 0 | 0 | 0 | |
| 3 | 0 | 1 | 0 | 0 | |
| 4 | 1 | 0 | 1 | 0 | |
| 5 | 1 | 1 | 0 | 1 | |

```

6  /**
7   * Demultiplexor:
8   * {a, b} = {in, 0} if sel == 0
9   *           {0, in} if sel == 1
10  */
11
12  CHIP DMux {
13    ... IN in, sel;
14    ... OUT a, b;
15
16    ... PARTS:
17    ... // Put your code here:
18    ... Not(in=sel, out=notsel);
19    ... And(a=in, b=sel, out=b);
20    ... And(a=in, b=notsel, out=a);
21  }

```

```

6  load DMux.hdl,
7  output-file DMux.out,
8  compare-to DMux.cmp,
9  output-list in%B3.1.3 sel%B3.1.3 a%B3.1.3 b%B3.1.3;
10
11  set in 0,
12  set sel 0,
13  eval,
14  output;
15
16  set sel 1,
17  eval,
18  output;
19
20  set in 1,
21  set sel 0,
22  eval,
23  output;
24
25  set sel 1,
26  eval,
27  output;

```

| Not16.cmp | | | |
|-----------|-------------------|-------------------|--|
| 1 | in | out | |
| 2 | 0000000000000000 | 1111111111111111 | |
| 3 | 1111111111111111 | 0000000000000000 | |
| 4 | 1010101010101010 | 0101010101010101 | |
| 5 | 00111110011000011 | 11000011001111100 | |
| 6 | 0001001000110100 | 11101101111001011 | |

```

6  /**
7   * 16-bit Not:
8   * for i=0..15: out[i] = not in[i]
9   */
10
11 CHIP Not16 {
12     ... IN in[16];
13     ... OUT out[16];
14
15     ... PARTS:
16         ... Nand(a=in[0], b=in[0], out=out[0]);
17         ... Nand(a=in[1], b=in[1], out=out[1]);
18         ... Nand(a=in[2], b=in[2], out=out[2]);
19         ... Nand(a=in[3], b=in[3], out=out[3]);
20         ... Nand(a=in[4], b=in[4], out=out[4]);
21         ... Nand(a=in[5], b=in[5], out=out[5]);
22         ... Nand(a=in[6], b=in[6], out=out[6]);
23         ... Nand(a=in[7], b=in[7], out=out[7]);
24         ... Nand(a=in[8], b=in[8], out=out[8]);
25         ... Nand(a=in[9], b=in[9], out=out[9]);
26         ... Nand(a=in[10], b=in[10], out=out[10]);
27         ... Nand(a=in[11], b=in[11], out=out[11]);
28         ... Nand(a=in[12], b=in[12], out=out[12]);
29         ... Nand(a=in[13], b=in[13], out=out[13]);
30         ... Nand(a=in[14], b=in[14], out=out[14]);
31         ... Nand(a=in[15], b=in[15], out=out[15]);
32
33 }

```

```
6  load Not16.hdl,␣
7  output-file Not16.out,␣
8  compare-to Not16.cmp,␣
9  output-list in%B1.16.1 out%B1.16.1;␣
10 ␣
11  set in %B0000000000000000,␣
12  eval,␣
13  output;␣
14 ␣
15  set in %B1111111111111111,␣
16  eval,␣
17  output;␣
18 ␣
19  set in %B1010101010101010,␣
20  eval,␣
21  output;␣
22 ␣
23  set in %B0011110011000011,␣
24  eval,␣
25  output;␣
26 ␣
27  set in %B0001001000110100,␣
28  eval,␣
29  output;
```


| Or16.cmp | | * | | | | | | |
|----------|--|------------------|--|------------------|--|-------------------|--|---|
| 1 | | a | | b | | out | | ↓ |
| 2 | | 0000000000000000 | | 0000000000000000 | | 0000000000000000 | | ↓ |
| 3 | | 0000000000000000 | | 1111111111111111 | | 1111111111111111 | | ↓ |
| 4 | | 1111111111111111 | | 1111111111111111 | | 1111111111111111 | | ↓ |
| 5 | | 1010101010101010 | | 0101010101010101 | | 1111111111111111 | | ↓ |
| 6 | | 0011110011000011 | | 0000111111110000 | | 00111111111110011 | | ↓ |
| 7 | | 0001001000110100 | | 1001100001110110 | | 1001101001110110 | | ↓ |

```

6  /**
7   * 16-bit bitwise Or:
8   * for i = 0..15 out[i] = (a[i] or b[i])
9   */
10
11 CHIP Or16 {
12     ... IN a[16], b[16];
13     ... OUT out[16];
14
15     ... PARTS:
16     ... // Put your code here:
17     » Or(a=a[0], b=b[0], out=out[0]);
18     » Or(a=a[1], b=b[1], out=out[1]);
19     » Or(a=a[2], b=b[2], out=out[2]);
20     » Or(a=a[3], b=b[3], out=out[3]);
21     » Or(a=a[4], b=b[4], out=out[4]);
22     » Or(a=a[5], b=b[5], out=out[5]);
23     » Or(a=a[6], b=b[6], out=out[6]);
24     » Or(a=a[7], b=b[7], out=out[7]);
25     » Or(a=a[8], b=b[8], out=out[8]);
26     » Or(a=a[9], b=b[9], out=out[9]);
27     » Or(a=a[10], b=b[10], out=out[10]);
28     » Or(a=a[11], b=b[11], out=out[11]);
29     » Or(a=a[12], b=b[12], out=out[12]);
30     » Or(a=a[13], b=b[13], out=out[13]);
31     » Or(a=a[14], b=b[14], out=out[14]);
32     » Or(a=a[15], b=b[15], out=out[15]);
33
34 }

```

```
6  load Or16.hdl,␣
7  output-file Or16.out,␣
8  compare-to Or16.cmp,␣
9  output-list a%B1.16.1 b%B1.16.1 out%B1.16.1;␣
10 ␣
11  set a %B0000000000000000,␣
12  set b %B0000000000000000,␣
13  eval,␣
14  output;␣
15  ␣
16  set a %B0000000000000000,␣
17  set b %B1111111111111111,␣
18  eval,␣
19  output;␣
20  ␣
21  set a %B1111111111111111,␣
22  set b %B1111111111111111,␣
23  eval,␣
24  output;␣
25  ␣
26  set a %B1010101010101010,␣
27  set b %B0101010101010101,␣
28  eval,␣
29  output;␣
30  ␣
31  set a %B0011110011000011,␣
32  set b %B0000111111110000,␣
33  eval,␣
34  output;␣
35  ␣
36  set a %B0001001000110100,␣
37  set b %B1001100001110110,␣
38  eval,␣
39  output;
```

Or8Way.cmp

| 1 | | in | | out | |
|---|--|----------|--|-----|--|
| 2 | | 00000000 | | 0 | |
| 3 | | 11111111 | | 1 | |
| 4 | | 00010000 | | 1 | |
| 5 | | 00000001 | | 1 | |
| 6 | | 00100110 | | 1 | |

```
6  /**  
7   * 8-way Or:  
8   * out = (in[0] or in[1] or ... or in[7])  
9   */  
10  
11 CHIP Or8Way {  
12     ... IN in[8];  
13     ... OUT out;  
14  
15     ... PARTS:  
16         // Put your code here:  
17         Or(a=in[0], b=in[1], out=c1);  
18         Or(a=in[2], b=in[3], out=c2);  
19         Or(a=in[4], b=in[5], out=c3);  
20         Or(a=in[6], b=in[7], out=c4);  
21         Or(a=c1, b=c2, out=c5);  
22         Or(a=c3, b=c4, out=c6);  
23         Or(a=c5, b=c6, out=out);  
24     }
```

```
6  load Or8Way.hdl,␣  
7  output-file Or8Way.out,␣  
8  compare-to Or8Way.cmp,␣  
9  output-list in%B2.8.2 out%B2.1.2;␣  
10 ␣  
11  set in %B00000000,␣  
12  eval,␣  
13  output;␣  
14 ␣  
15  set in %B11111111,␣  
16  eval,␣  
17  output;␣  
18 ␣  
19  set in %B00010000,␣  
20  eval,␣  
21  output;␣  
22 ␣  
23  set in %B00000001,␣  
24  eval,␣  
25  output;␣  
26 ␣  
27  set in %B00100110,␣  
28  eval,␣  
29  output;
```

| DMux4Way.cmp | | | | | | | |
|--------------|----|-----|---|---|---|---|--|
| 1 | in | sel | a | b | c | d | |
| 2 | 0 | 00 | 0 | 0 | 0 | 0 | |
| 3 | 0 | 01 | 0 | 0 | 0 | 0 | |
| 4 | 0 | 10 | 0 | 0 | 0 | 0 | |
| 5 | 0 | 11 | 0 | 0 | 0 | 0 | |
| 6 | 1 | 00 | 1 | 0 | 0 | 0 | |
| 7 | 1 | 01 | 0 | 1 | 0 | 0 | |
| 8 | 1 | 10 | 0 | 0 | 1 | 0 | |
| 9 | 1 | 11 | 0 | 0 | 0 | 1 | |

```

6  /**
7   * 4-way demultiplexor:
8   * {a, b, c, d} = {in, 0, 0, 0} if sel == 00
9   *           {0, in, 0, 0} if sel == 01
10  *           {0, 0, in, 0} if sel == 10
11  *           {0, 0, 0, in} if sel == 11
12  */
13
14  CHIP DMux4Way {
15      ... IN in, sel[2];
16      ... OUT a, b, c, d;
17
18      ... PARTS:
19          ... DMux(in=in, sel=sel[1], a=t1, b=t2);
20          ... DMux(in=t1, sel=sel[0], a=a, b=b);
21          ... DMux(in=t2, sel=sel[0], a=c, b=d);
22  }

```

```
6 load DMux4Way.hdl,␣
7 output-file DMux4Way.out,␣
8 compare-to DMux4Way.cmp,␣
9 output-list in%B2.1.2 sel%B2.2.2 a%B2.1.2 b%B2.1.2 c%B2.1.2 d%B2.1.2;␣
10 ␣
11 set in 0,␣
12 set sel %B00,␣
13 eval,␣
14 output;␣
15 ␣
16 set sel %B01,␣
17 eval,␣
18 output;␣
19 ␣
20 set sel %B10,␣
21 eval,␣
22 output;␣
23 ␣
24 set sel %B11,␣
25 eval,␣
26 output;␣
27 ␣
28 set in 1,␣
29 set sel %B00,␣
30 eval,␣
31 output;␣
32 ␣
33 set sel %B01,␣
34 eval,␣
35 output;␣
36 ␣
37 set sel %B10,␣
38 eval,
```



```

6  /**
7   * 8-way demultiplexor:
8   * {a, b, c, d, e, f, g, h} = {in, 0, 0, 0, 0, 0, 0, 0} if sel == 000
9   *                               {0, in, 0, 0, 0, 0, 0, 0} if sel == 001
10  *                               etc.
11  *                               {0, 0, 0, 0, 0, 0, 0, in} if sel == 111
12  */
13
14  CHIP DMux8Way {
15      IN in, sel[3];
16      OUT a, b, c, d, e, f, g, h;
17
18      PARTS:
19      // Put your code here:
20      DMux(in=in, sel=sel[2], a=c1, b=c2);
21      DMux4Way(in=c1, sel[0]=sel[0], sel[1]=sel[1], a=a, b=b, c=c, d=d);
22      DMux4Way(in=c2, sel[0]=sel[0], sel[1]=sel[1], a=e, b=f, c=g, d=h);
23  }

```

```
6 load DMux8Way.hdl,d
7 output-file DMux8Way.out,d
8 compare-to DMux8Way.cmp,d
9 output-list in%B2.1.2 sel%B2.3.2 a%B2.1.2 b%B2.1.2 c%B2.1.2 d%B2.1.2 e%B2.1.2 f%B2.1.2 g%B2.1.2 h%B2.1.2;d
10 d
11 set in 0,d
12 set sel %B000,d
13 eval,d
14 output;d
15 d
16 set sel %B001,d
17 eval,d
18 output;d
19 d
20 set sel %B010,d
21 eval,d
22 output;d
23 d
24 set sel %B011,d
25 eval,d
26 output;d
27 d
28 set sel %B100,d
29 eval,d
30 output;d
31 d
32 set sel %B101,d
33 eval,d
34 output;d
35 d
36 set sel %B110,d
37 eval,d
38 output;d
```

```
39  ↵
40  set sel %B111, ↵
41  eval, ↵
42  output; ↵
43  ↵
44  set in 1, ↵
45  set sel %B000, ↵
46  eval, ↵
47  output; ↵
48  ↵
49  set sel %B001, ↵
50  eval, ↵
51  output; ↵
52  ↵
53  < set sel %B010, ↵
54  eval, ↵
55  output; ↵
56  ↵
57  set sel %B011, ↵
58  eval, ↵
59  output; ↵
60  ↵
61  set sel %B100, ↵
62  eval, ↵
63  output; ↵
64  ↵
65  set sel %B101, ↵
66  eval, ↵
67  output; ↵
68  ↵
69  set sel %B110, ↵
70  eval, ↵
```

```
71 output;
72
73 set sel %B111,
74 eval,
75 output;
76
```

| Mux16.cmp | | | | | |
|-----------|------------------|------------------|-----|------------------|--|
| 1 | a | b | sel | out | |
| 2 | 0000000000000000 | 0000000000000000 | 0 | 0000000000000000 | |
| 3 | 0000000000000000 | 0000000000000000 | 1 | 0000000000000000 | |
| 4 | 0000000000000000 | 0001001000110100 | 0 | 0000000000000000 | |
| 5 | 0000000000000000 | 0001001000110100 | 1 | 0001001000110100 | |
| 6 | 1001100001110110 | 0000000000000000 | 0 | 1001100001110110 | |
| 7 | 1001100001110110 | 0000000000000000 | 1 | 0000000000000000 | |
| 8 | 1010101010101010 | 0101010101010101 | 0 | 1010101010101010 | |
| 9 | 1010101010101010 | 0101010101010101 | 1 | 0101010101010101 | |

```

6  /**
7   * 16-bit multiplexor:
8   * for i = 0..15 out[i] = a[i] if sel == 0
9   *                        b[i] if sel == 1
10  */
11
12  CHIP Mux16 {
13    ... IN a[16], b[16], sel;
14    ... OUT out[16];
15
16    ... PARTS:
17    ... // Put your code here:
18    ... Mux(a=a[0], b=b[0], sel=sel, out=out[0]);
19    ... Mux(a=a[1], b=b[1], sel=sel, out=out[1]);
20    ... Mux(a=a[2], b=b[2], sel=sel, out=out[2]);
21    ... Mux(a=a[3], b=b[3], sel=sel, out=out[3]);
22    ... Mux(a=a[4], b=b[4], sel=sel, out=out[4]);
23    ... Mux(a=a[5], b=b[5], sel=sel, out=out[5]);
24    ... Mux(a=a[6], b=b[6], sel=sel, out=out[6]);
25    ... Mux(a=a[7], b=b[7], sel=sel, out=out[7]);
26    ... Mux(a=a[8], b=b[8], sel=sel, out=out[8]);
27    ... Mux(a=a[9], b=b[9], sel=sel, out=out[9]);
28    ... Mux(a=a[10], b=b[10], sel=sel, out=out[10]);
29    ... Mux(a=a[11], b=b[11], sel=sel, out=out[11]);
30    ... Mux(a=a[12], b=b[12], sel=sel, out=out[12]);
31    ... Mux(a=a[13], b=b[13], sel=sel, out=out[13]);
32    ... Mux(a=a[14], b=b[14], sel=sel, out=out[14]);
33    ... Mux(a=a[15], b=b[15], sel=sel, out=out[15]);
34  }

```

```
6 load Mux16.hdl,
7 output-file Mux16.out,
8 compare-to Mux16.cmp,
9 output-list a%B1.16.1 b%B1.16.1 sel%D2.1.2 out%B1.16.1;
10
11 set a 0,
12 set b 0,
13 set sel 0,
14 eval,
15 output;
16
17 set sel 1,
18 eval,
19 output;
20
21 set a %B0000000000000000,
22 set b %B0001001000110100,
23 set sel 0,
24 eval,
25 output;
26
27 set sel 1,
28 eval,
29 output;
30
31 set a %B1001100001110110,
32 set b %B0000000000000000,
33 set sel 0,
34 eval,
35 output;
36
```

```

37  set sel 1,
38  eval,
39  output;
40
41  set a %B1010101010101010,
42  set b %B0101010101010101,
43  set sel 0,
44  eval,
45  output;
46
47  set sel 1,
48  eval,
49  output;

```

| Mux4Way16.cmp | | | | | | | |
|---------------|------------------|------------------|------------------|------------------|-----|------------------|--|
| 1 | a | b | c | d | sel | out | |
| 2 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 00 | 0000000000000000 | |
| 3 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 01 | 0000000000000000 | |
| 4 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 10 | 0000000000000000 | |
| 5 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 11 | 0000000000000000 | |
| 6 | 0001001000110100 | 1001100001110110 | 1010101010101010 | 0101010101010101 | 00 | 0001001000110100 | |
| 7 | 0001001000110100 | 1001100001110110 | 1010101010101010 | 0101010101010101 | 01 | 1001100001110110 | |
| 8 | 0001001000110100 | 1001100001110110 | 1010101010101010 | 0101010101010101 | 10 | 1010101010101010 | |
| 9 | 0001001000110100 | 1001100001110110 | 1010101010101010 | 0101010101010101 | 11 | 0101010101010101 | |

```

6  /**
7   * 4-way 16-bit multiplexor:
8   * out = a if sel == 00
9   *      b if sel == 01
10  *      c if sel == 10
11  *      d if sel == 11
12  */
13
14  CHIP Mux4Way16 {
15      IN a[16], b[16], c[16], d[16], sel[2];
16      OUT out[16];
17
18      PARTS:
19      Mux16(a=a, b=b, sel=sel[0], out=aORb);
20      Mux16(a=c, b=d, sel=sel[0], out=cORD);
21      Mux16(a=aORb, b=cORD, sel=sel[1], out=out);
22  }

```



```
6 load Mux4Way16.hdl, ↵
7 output-file Mux4Way16.out, ↵
8 compare-to Mux4Way16.cmp, ↵
9 output-list a%B1.16.1 b%B1.16.1 c%B1.16.1 d%B1.16.1 sel%B2.2.2 out%B1.16.1; ↵
10 ↵
11 set a 0, ↵
12 set b 0, ↵
13 set c 0, ↵
14 set d 0, ↵
15 set sel 0, ↵
16 eval, ↵
17 output; ↵
18 ↵
19 set sel 1, ↵
20 eval, ↵
21 output; ↵
22 ↵
23 set sel 2, ↵
24 eval, ↵
25 output; ↵
26 ↵
27 set sel 3, ↵
28 eval, ↵
29 output; ↵
30 ↵
31 set a %B0001001000110100, ↵
32 set b %B1001100001110110, ↵
33 set c %B1010101010101010, ↵
34 set d %B0101010101010101, ↵
```

```

35  set sel 0,
36  eval,
37  output;
38
39  set sel 1,
40  eval,
41  output;
42
43  set sel 2,
44  eval,
45  output;
46
47  set sel 3,
48  eval,
49  output;
50

```

| Mux8Way16.cmp | | | | | | | | | |
|---------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|--|
| 1 | a | b | c | d | e | f | g | h | |
| 2 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | |
| 3 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | |
| 4 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | |
| 5 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | |
| 6 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | |
| 7 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | |
| 8 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | |
| 9 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000 | |
| 10 | 0001001000110100 | 0010001101000101 | 0011010001010110 | 0100010101100111 | 0101011001111000 | 0110011110001001 | 0111100010011010 | 1000100110101011 | |
| 11 | 0001001000110100 | 0010001101000101 | 0011010001010110 | 0100010101100111 | 0101011001111000 | 0110011110001001 | 0111100010011010 | 1000100110101011 | |
| 12 | 0001001000110100 | 0010001101000101 | 0011010001010110 | 0100010101100111 | 0101011001111000 | 0110011110001001 | 0111100010011010 | 1000100110101011 | |
| 13 | 0001001000110100 | 0010001101000101 | 0011010001010110 | 0100010101100111 | 0101011001111000 | 0110011110001001 | 0111100010011010 | 1000100110101011 | |
| 14 | 0001001000110100 | 0010001101000101 | 0011010001010110 | 0100010101100111 | 0101011001111000 | 0110011110001001 | 0111100010011010 | 1000100110101011 | |
| 15 | 0001001000110100 | 0010001101000101 | 0011010001010110 | 0100010101100111 | 0101011001111000 | 0110011110001001 | 0111100010011010 | 1000100110101011 | |
| 16 | 0001001000110100 | 0010001101000101 | 0011010001010110 | 0100010101100111 | 0101011001111000 | 0110011110001001 | 0111100010011010 | 1000100110101011 | |
| < | 0001001000110100 | 0010001101000101 | 0011010001010110 | 0100010101100111 | 0101011001111000 | 0110011110001001 | 0111100010011010 | 1000100110101011 | |

```

6  ▾ /**
7   * 8-way 16-bit multiplexor:
8   * out = a if sel == 000
9   *      b if sel == 001
10  *      etc.
11  *      h if sel == 111
12  */
13
14  ▾ CHIP Mux8Way16 {
15  ▾   IN a[16], b[16], c[16], d[16],
16     e[16], f[16], g[16], h[16],
17     sel[3];
18   OUT out[16];
19
20   PARTS:
21   // Put your code here:
22   Mux4Way16(a=a, b=b, c=c, d=d, sel=sel[0..1], out=abcORd);
23   Mux4Way16(a=e, b=f, c=g, d=h, sel=sel[0..1], out=efgORh);
24   Mux16(a=abcORd, b=efgORh, sel=sel[2], out=out);
25 }

```

```
6 load Mux8Way16.hdl,d
7 output-file Mux8Way16.out,d
8 compare-to Mux8Way16.comp,d
9 output-list a%B1.16.1 b%B1.16.1 c%B1.16.1 d%B1.16.1 e%B1.16.1 f%B1.16.1 g%B1.16.1 h%B1.16.1 sel%B2.3.2 out%B1.16.1;d
10 d
11 set a 0,d
12 set b 0,d
13 set c 0,d
14 set d 0,d
15 set e 0,d
16 set f 0,d
17 set g 0,d
18 set h 0,d
19 set sel 0,d
20 eval,d
21 output;d
22 d
23 set sel 1,d
24 eval,d
25 output;d
26 d
27 set sel 2,d
28 eval,d
29 output;d
30 d
31 set sel 3,d
32 eval,d
33 output;d
34 d
35 set sel 4,d
36 eval,d
37 output;d
```

```
38  ↵
39  set sel 5, ↵
40  eval, ↵
41  output; ↵
42  ↵
43  set sel 6, ↵
44  eval, ↵
45  output; ↵
46  ↵
47  set sel 7, ↵
48  eval, ↵
49  output; ↵
50  ↵
51  set a %B0001001000110100, ↵
52  set b %B0010001101000101, ↵
53  set c %B0011010001010110, ↵
54  set d %B0100010101100111, ↵
55  set e %B0101011001111000, ↵
56  set f %B0110011110001001, ↵
57  set g %B0111100010011010, ↵
58  set h %B1000100110101011, ↵
59  set sel 0, ↵
60  eval, ↵
61  output; ↵
62  ↵
63  set sel 1, ↵
64  eval, ↵
65  output; ↵
66  ↵
67  set sel 2, ↵
68  eval, ↵
69  output; ↵
```

```
70  ٤
71  set sel 3,٤
72  eval,٤
73  output;٤
74  ٤
75  set sel 4,٤
76  eval,٤
77  output;٤
78  ٤
79  set sel 5,٤
80  eval,٤
81  output;٤
82  ٤
83  set sel 6,٤
84  eval,٤
85  output;٤
86  ٤
87  set sel 7,٤
88  eval,٤
89  output;٤
```

Problem 2

| HalfAdder.cmp | | | | | | | | | |
|---------------|--|---|--|---|--|-----|--|-------|--|
| 1 | | a | | b | | sum | | carry | |
| 2 | | 0 | | 0 | | 0 | | 0 | |
| 3 | | 0 | | 1 | | 1 | | 0 | |
| 4 | | 1 | | 0 | | 1 | | 0 | |
| 5 | | 1 | | 1 | | 0 | | 1 | |
| 6 | | | | | | | | | |

```
6  /**
7   * Computes the sum of two bits.
8   */
9
10 CHIP HalfAdder {
11     IN a, b;    // 1-bit inputs
12     OUT sum,    // Right bit of a + b
13     carry;     // Left bit of a + b
14
15     PARTS:
16     // Put your code here:
17     Xor(a=a, b=b, out=sum);
18     And(a=a, b=b, out=carry);
19 }
```

```
6  load HalfAdder.hdl,␣
7  output-file HalfAdder.out,␣
8  compare-to HalfAdder.cmp,␣
9  output-list a%B3.1.3 b%B3.1.3 sum%B3.1.3 carry%B3.1.3;␣
10 ␣
11  set a 0,␣
12  set b 0,␣
13  eval,␣
14  output;␣
15 ␣
16  set a 0,␣
17  set b 1,␣
18  eval,␣
19  output;␣
20 ␣
21  set a 1,␣
22  set b 0,␣
23  eval,␣
24  output;␣
25 ␣
26  set a 1,␣
27  set b 1,␣
28  eval,␣
29  output;␣
```


| FullAdder.cmp | | | | | | | | | | | | |
|---------------|--|---|--|---|--|---|--|-----|--|-------|--|---|
| 1 | | a | | b | | c | | sum | | carry | | ⌘ |
| 2 | | 0 | | 0 | | 0 | | 0 | | 0 | | ⌘ |
| 3 | | 0 | | 0 | | 1 | | 1 | | 0 | | ⌘ |
| 4 | | 0 | | 1 | | 0 | | 1 | | 0 | | ⌘ |
| 5 | | 0 | | 1 | | 1 | | 0 | | 1 | | ⌘ |
| 6 | | 1 | | 0 | | 0 | | 1 | | 0 | | ⌘ |
| 7 | | 1 | | 0 | | 1 | | 0 | | 1 | | ⌘ |
| 8 | | 1 | | 1 | | 0 | | 0 | | 1 | | ⌘ |
| 9 | | 1 | | 1 | | 1 | | 1 | | 1 | | ⌘ |
| 10 | | | | | | | | | | | | |

```

6  /**⌘
7  . * Computes the sum of three bits.⌘
8  . */⌘
9  ⌘
10 CHIP FullAdder {⌘
11     . . . . IN a, b, c; // 1-bit inputs⌘
12     . . . . OUT sum, // Right bit of a + b + c⌘
13     . . . . . . . . carry; // Left bit of a + b + c⌘
14     ⌘
15     . . . . PARTS:⌘
16     . . . . // Put you code here:⌘
17     . . . . HalfAdder(a=a, b=b, sum=w1, carry=c1);⌘
18     . . . . HalfAdder(a=w1, b=c, sum=sum, carry=c2);⌘
19     . . . . Or(a=c1, b=c2, out=carry);⌘
20 }⌘

```

```
6  load FullAdder.hdl,␣
7  output-file FullAdder.out,␣
8  compare-to FullAdder.cmp,␣
9  output-list a%B3.1.3 b%B3.1.3 c%B3.1.3 sum%B3.1.3 carry%B3.1.3;␣
10 ␣
11  set a 0,␣
12  set b 0,␣
13  set c 0,␣
14  eval,␣
15  output;␣
16 ␣
17  set c 1,␣
18  eval,␣
19  output;␣
20 ␣
21  set b 1,␣
22  set c 0,␣
23  eval,␣
24  output;␣
25 ␣
26  set c 1,␣
27  eval,␣
28  output;␣
29 ␣
30  set a 1,␣
31  set b 0,␣
32  set c 0,␣
33  eval,␣
34  output;␣
35 ␣
36  set c 1,␣
37  eval,␣
```

```

38  output;
39
40  set b 1,
41  set c 0,
42  eval,
43  output;
44
45  set c 1,
46  eval,
47  output;
48

```

| | | Add16.cmp | | | |
|---|--|------------------|--|------------------|------------------|
| 1 | | a | | b | out |
| 2 | | 0000000000000000 | | 0000000000000000 | 0000000000000000 |
| 3 | | 0000000000000000 | | 1111111111111111 | 1111111111111111 |
| 4 | | 1111111111111111 | | 1111111111111111 | 1111111111111110 |
| 5 | | 1010101010101010 | | 0101010101010101 | 1111111111111111 |
| 6 | | 0011110011000011 | | 0000111111110000 | 0100110010110011 |
| 7 | | 0001001000110100 | | 1001100001110110 | 1010101010101010 |
| 8 | | | | | |

```

6  /**
7   * Adds two 16-bit values.
8   * The most significant carry bit is ignored.
9   */
10
11  CHIP Add16 {
12      ... IN a[16], b[16];
13      ... OUT out[16];
14
15      ... PARTS:
16      ... // Put you code here:
17      ... FullAdder(a=false, b=a[0], c=b[0], sum=out[0], carry=c0);
18      ... FullAdder(a=c0, b=a[1], c=b[1], sum=out[1], carry=c1);
19      ... FullAdder(a=c1, b=a[2], c=b[2], sum=out[2], carry=c2);
20      ... FullAdder(a=c2, b=a[3], c=b[3], sum=out[3], carry=c3);
21      ... FullAdder(a=c3, b=a[4], c=b[4], sum=out[4], carry=c4);
22      ... FullAdder(a=c4, b=a[5], c=b[5], sum=out[5], carry=c5);
23      ... FullAdder(a=c5, b=a[6], c=b[6], sum=out[6], carry=c6);
24      ... FullAdder(a=c6, b=a[7], c=b[7], sum=out[7], carry=c7);
25      ... FullAdder(a=c7, b=a[8], c=b[8], sum=out[8], carry=c8);
26      ... FullAdder(a=c8, b=a[9], c=b[9], sum=out[9], carry=c9);
27      ... FullAdder(a=c9, b=a[10], c=b[10], sum=out[10], carry=c10);
28      ... FullAdder(a=c10, b=a[11], c=b[11], sum=out[11], carry=c11);
29      ... FullAdder(a=c11, b=a[12], c=b[12], sum=out[12], carry=c12);
30      ... FullAdder(a=c12, b=a[13], c=b[13], sum=out[13], carry=c13);
31      ... FullAdder(a=c13, b=a[14], c=b[14], sum=out[14], carry=c14);
32      ... FullAdder(a=c14, b=a[15], c=b[15], sum=out[15], carry=c15);
33  }

```

```
6 load Add16.hdl,
7 output-file Add16.out,
8 compare-to Add16.cmp,
9 output-list a%B1.16.1 b%B1.16.1 out%B1.16.1;
10
11 set a %B000000000000000000,
12 set b %B000000000000000000,
13 eval,
14 output;
15
16 set a %B000000000000000000,
17 set b %B111111111111111111,
18 eval,
19 output;
20
21 set a %B111111111111111111,
22 set b %B111111111111111111,
23 eval,
24 output;
25
26 set a %B1010101010101010,
27 set b %B0101010101010101,
28 eval,
29 output;
30
31 set a %B0011110011000011,
32 set b %B0000111111111000,
33 eval,
34 output;
35
36 set a %B0001001000110100,
37 set b %B1001100001110110,
```

```

38 eval,
39 output;

```

| Inc16.cmp | | | |
|-----------|-------------------|-------------------|--|
| 1 | in | out | |
| 2 | 0000000000000000 | 0000000000000001 | |
| 3 | 1111111111111111 | 0000000000000000 | |
| 4 | 00000000000000101 | 00000000000000110 | |
| 5 | 1111111111111011 | 1111111111111100 | |

```

6  /**
7   * 16-bit incrementer:
8   * out = in + 1 (arithmetic addition)
9   */
10
11 CHIP Inc16 {
12     IN in[16];
13     OUT out[16];
14
15     PARTS:
16     // Put your code here:
17     Add16(a[0..15]=in[0..15], b[0]=true, b[1..15]=false, out[0..15]=out[0..15]);
18 }

```

```
6  load Inc16.hdl,␣  
7  output-file Inc16.out,␣  
8  compare-to Inc16.cmp,␣  
9  output-list in%B1.16.1 out%B1.16.1;␣  
10 ␣  
11 set in %B0000000000000000, // in = 0␣  
12 eval,␣  
13 output;␣  
14 ␣  
15 set in %B1111111111111111, // in = -1␣  
16 eval,␣  
17 output;␣  
18 ␣  
19 set in %B00000000000000101, // in = 5␣  
20 eval,␣  
21 output;␣  
22 ␣  
23 set in %B1111111111111011, // in = -5␣  
24 eval,␣  
25 output;␣
```

Problem 3

| ALU.cmp | | | | | | | | | | | | | | |
|---------|------------------|------------------|----|----|----|----|---|----|------------------|----|----|--|--|--|
| 1 | x | y | zx | nx | zy | ny | f | no | out | zr | ng | | | |
| 2 | 0000000000000000 | 1111111111111111 | 1 | 0 | 1 | 0 | 1 | 0 | 0000000000000000 | 1 | 0 | | | |
| 3 | 0000000000000000 | 1111111111111111 | 1 | 1 | 1 | 1 | 1 | 1 | 0000000000000001 | 0 | 0 | | | |
| 4 | 0000000000000000 | 1111111111111111 | 1 | 1 | 1 | 0 | 1 | 0 | 1111111111111111 | 0 | 1 | | | |
| 5 | 0000000000000000 | 1111111111111111 | 0 | 0 | 1 | 1 | 0 | 0 | 0000000000000000 | 1 | 0 | | | |
| 6 | 0000000000000000 | 1111111111111111 | 1 | 1 | 0 | 0 | 0 | 0 | 1111111111111111 | 0 | 1 | | | |
| 7 | 0000000000000000 | 1111111111111111 | 0 | 0 | 1 | 1 | 0 | 1 | 1111111111111111 | 0 | 1 | | | |
| 8 | 0000000000000000 | 1111111111111111 | 1 | 1 | 0 | 0 | 0 | 1 | 0000000000000000 | 1 | 0 | | | |
| 9 | 0000000000000000 | 1111111111111111 | 0 | 0 | 1 | 1 | 1 | 1 | 0000000000000000 | 1 | 0 | | | |
| 10 | 0000000000000000 | 1111111111111111 | 1 | 1 | 0 | 0 | 1 | 1 | 0000000000000001 | 0 | 0 | | | |
| 11 | 0000000000000000 | 1111111111111111 | 0 | 1 | 1 | 1 | 1 | 1 | 0000000000000001 | 0 | 0 | | | |
| 12 | 0000000000000000 | 1111111111111111 | 1 | 1 | 0 | 1 | 1 | 1 | 0000000000000000 | 1 | 0 | | | |
| 13 | 0000000000000000 | 1111111111111111 | 0 | 0 | 1 | 1 | 1 | 0 | 1111111111111111 | 0 | 1 | | | |
| 14 | 0000000000000000 | 1111111111111111 | 1 | 1 | 0 | 0 | 1 | 0 | 1111111111111110 | 0 | 1 | | | |
| 15 | 0000000000000000 | 1111111111111111 | 0 | 0 | 0 | 0 | 1 | 0 | 1111111111111111 | 0 | 1 | | | |
| 16 | 0000000000000000 | 1111111111111111 | 0 | 1 | 0 | 0 | 1 | 1 | 0000000000000001 | 0 | 0 | | | |
| 17 | 0000000000000000 | 1111111111111111 | 0 | 0 | 0 | 1 | 1 | 1 | 1111111111111111 | 0 | 1 | | | |
| 18 | 0000000000000000 | 1111111111111111 | 0 | 0 | 0 | 0 | 0 | 0 | 0000000000000000 | 1 | 0 | | | |
| 19 | 0000000000000000 | 1111111111111111 | 0 | 1 | 0 | 1 | 0 | 1 | 1111111111111111 | 0 | 1 | | | |
| 20 | 0000000000010001 | 0000000000000011 | 1 | 0 | 1 | 0 | 1 | 0 | 0000000000000000 | 1 | 0 | | | |
| 21 | 0000000000010001 | 0000000000000011 | 1 | 1 | 1 | 1 | 1 | 1 | 0000000000000001 | 0 | 0 | | | |
| 22 | 0000000000010001 | 0000000000000011 | 1 | 1 | 1 | 0 | 1 | 0 | 1111111111111111 | 0 | 1 | | | |
| 23 | 0000000000010001 | 0000000000000011 | 0 | 0 | 1 | 1 | 0 | 0 | 0000000000010001 | 0 | 0 | | | |
| 24 | 0000000000010001 | 0000000000000011 | 1 | 1 | 0 | 0 | 0 | 0 | 0000000000000011 | 0 | 0 | | | |
| 25 | 0000000000010001 | 0000000000000011 | 0 | 0 | 1 | 1 | 0 | 1 | 1111111111101110 | 0 | 1 | | | |
| 26 | 0000000000010001 | 0000000000000011 | 1 | 1 | 0 | 0 | 0 | 1 | 1111111111111100 | 0 | 1 | | | |
| 27 | 0000000000010001 | 0000000000000011 | 0 | 0 | 1 | 1 | 1 | 1 | 1111111111101111 | 0 | 1 | | | |
| 28 | 0000000000010001 | 0000000000000011 | 1 | 1 | 0 | 0 | 1 | 1 | 1111111111111101 | 0 | 1 | | | |
| 29 | 0000000000010001 | 0000000000000011 | 0 | 1 | 1 | 1 | 1 | 1 | 0000000000010010 | 0 | 0 | | | |
| 30 | 0000000000010001 | 0000000000000011 | 1 | 1 | 0 | 1 | 1 | 1 | 0000000000000100 | 0 | 0 | | | |
| 31 | 0000000000010001 | 0000000000000011 | 0 | 0 | 1 | 1 | 1 | 0 | 0000000000010000 | 0 | 0 | | | |
| 32 | 0000000000010001 | 0000000000000011 | 1 | 1 | 0 | 0 | 1 | 0 | 0000000000000010 | 0 | 0 | | | |
| 33 | 0000000000010001 | 0000000000000011 | 0 | 0 | 0 | 0 | 1 | 0 | 0000000000010100 | 0 | 0 | | | |
| 34 | 0000000000010001 | 0000000000000011 | 0 | 1 | 0 | 0 | 1 | 1 | 0000000000001110 | 0 | 0 | | | |
| 35 | 0000000000010001 | 0000000000000011 | 0 | 0 | 0 | 1 | 1 | 1 | 111111111110010 | 0 | 1 | | | |
| 36 | 0000000000010001 | 0000000000000011 | 0 | 0 | 0 | 0 | 0 | 0 | 0000000000000001 | 0 | 0 | | | |
| 37 | 0000000000010001 | 0000000000000011 | 0 | 1 | 0 | 1 | 0 | 1 | 0000000000010011 | 0 | 0 | | | |


```

29 CHIP ALU {
30     ... IN ...
31     ... x[16], y[16], // 16-bit inputs ...
32     ... zx, // zero the x input?
33     ... nx, // negate the x input?
34     ... zy, // zero the y input?
35     ... ny, // negate the y input?
36     ... f, // compute out = x + y (if 1) or x & y (if 0)
37     ... no; // negate the out output?
38
39     ... OUT ...
40     ... out[16], // 16-bit output
41     ... zr, // 1 if (out == 0), 0 otherwise
42     ... ng; // 1 if (out < 0), 0 otherwise
43
44     ... PARTS:
45     ... // Put your code here:
46     ... Mux16(a=x, b=false, sel=zx, out=zeroedX);
47     ... Not16(in=zeroedX, out=notX);
48     ... Mux16(a=zeroedX, b=notX, sel=nx, out=negX);
49     ... Mux16(a=y, b=false, sel=zy, out=zeroedY);
50     ... Not16(in=zeroedY, out=notY);
51     ... Mux16(a=zeroedY, b=notY, sel=ny, out=negY);
52
53     ... Add16(a=negX, b=negY, out=xPlusY);
54     ... And16(a=negX, b=negY, out=xAndY);
55     ... Mux16(a=xAndY, b=xPlusY, sel=f, out=funcVar);
56     ... Not16(in=funcVar, out=negFuncVar);
57     ... Mux16(a=funcVar, b=negFuncVar, sel=no, out=out, out[0..7]=lowOut, out[8..15]=highOut, out[15]=ng);
58
59     ... Or8Way(in=lowOut, out=zero1);
60     ... Or8Way(in=highOut, out=zero2);
61     ... Or(a=zero1, b=zero2, out=notZr);
62     ... Not(in=notZr, out=zr);
63 }

```

```

1  load ALU.hdl,
2  output-file ALU2.out,
3  compare-to ALU.cmp,
4  output-list x%B1.16.1 y%B1.16.1 zx%B1.1.1 nx%B1.1.1 zy%B1.1.1
5  .....ny%B1.1.1 f%B1.1.1 no%B1.1.1 out%B1.16.1 zr%B1.1.1
6  .....ng%B1.1.1;
7  set x %B0000000000000000, // x = 0
8  set y %B1111111111111111; // y = -1
9  // Test0,
10 set zx 1,
11 set nx 0,
12 set zy 1,
13 set ny 0,
14 set f 1,
15 set no 0,
16 eval,
17 output;
18
19 // Test0,
20 set zx 1,
21 set nx 1,
22 set zy 1,
23 set ny 1,
24 set f 1,
25 set no 1,
26 eval,
27 output;

```

Here are just a couple of tests for the ALU, there are more tests in the ALU.tst file.

a. less than 1296.

b. 2 tests with 36 cases total

c. Unlike the full ALU.cmp, the zr and ng status outputs are ignored. This is only a partial test, so the testing and implementation of the gates used to construct the ALU will be the only things needed. The nostat.tst helps concentrate on getting the correct ALU computation.

d. 1296 and greater.

e. 2 tests with 36 tests total

f. To thoroughly test our ALU, we needed to include tests for the gates used to construct our ALU. We tested our 16 bit AND gate, 16 bit NOT gate, 16 bit Multiplexer, an 8 bit way OR and a regular NOT gate. We had 6 inputs and 3 outs, but if you take the powers of two, it is actually a total of 64 and 8.

Also, ALU NoStat test needed to be included because any comparison failures during the ALU.tst will be caused by errors in the handling of the ignored status outputs.