【CA】Day5

Course	Nand to Tetris
Study Date	@February 7, 2022

【Ch2】Boolean Arithmetic

2.1 Background

In general, let $x = x_n x_{n-1} ... x_0$ be a string of digits. The value of x in base b, denoted $(x)_b$ is defined as follows:

$$(x_n x_{n-1} \ldots x_0)_b = \sum_{i=0}^{n} x_i \cdot b^i$$

- The system can code a total of 2^n signed numbers, of which the maximal and minimal numbers are $2^{n-1} - 1$ and -2^{n-1}, respectively

2.2 Specification

2.5.1 Adders

We'll focus on the following hierarchy of adders:

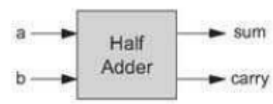
- Half-adder: designed to add two bits
- Full-adder: designed to add three bits
- Adder: designed to add two n-bit numbers

We will also specify a special-purpose adder, called an incrementer, designed to add 1 to a given number.

Half-adder

The first step on our way to adding binary numbers is to be able to add two bits. Let us call the least significant bit sum, and the most significant bit carry.

Inputs		Outputs	
a	b	carry	sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



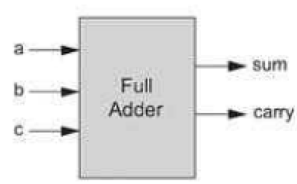

```
Chip name: HalfAdder
Inputs:    a, b
Outputs:   sum, carry
Function:  sum   = LSB of a + b
           carry = MSB of a + b
```

Figure 2.2 Half-adder, designed to add 2 bits.

Full-Adder

Now that we know how to add two bits, the following figure presents a full-adder chip, designed to add three bits. The full-adder chip also provides the carry bit and the sum bit.

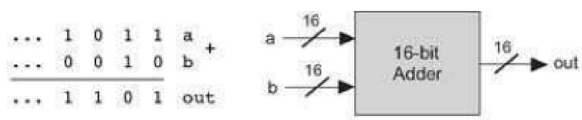
a	b	c	carry	sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1




```
Chip name: FullAdder
Inputs:    a, b, c
Outputs:   sum, carry
Function:  sum = LSB of a + b + c
           carry = MSB of a + b + c
```

Adder

Memory and register chips represent integer number by n-bit patterns, n begin 16, 32, 64 and so forth-depending on the computer platform. The chip whose job is to add such numbers is called a multi-bit adder, or simply adder. The following figure presents a 16-bit adder.




```
Chip name: Add16
Inputs:    a[16], b[16]
Outputs:   out[16]
Function:  out = a + b
Comment:   Integer 2's complement addition.
           Overflow is neither detected nor handled.
```

Figure 2.4 16-bit adder. Addition of two *n*-bit binary numbers for any n is "more of the same."

Incrementer

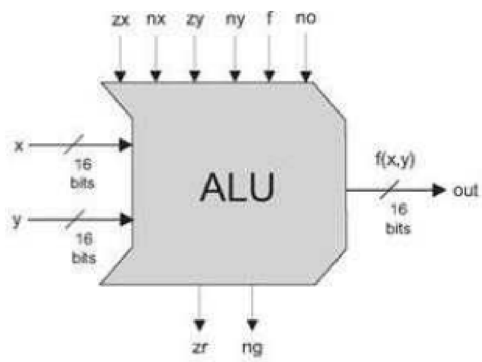
It is convenient to ahve a special-purpose chip dedicatged to adding the constant 1 to a given number. Here is the specification of a 16-bit incrementer:

```
Chip name: Inc16
Inputs:    in[16]
Outputs:   out[16]
Function:  out=in+1
Comment:   Integer 2's complement addition.
           Overflow is neither detected nor handled.
```

2.2.2 The Arithmetic Logic Unit(ALU)

The Hack ALU computes a fixed set of functions $out = f_i(x, y)$ where x and y are the chip's two 16-bit inputs, out is the chip's 16-bit output, and f_i is an arithmetic or logical function selected from a fixed repertoire of eighteen possible functions.

We instruct the ALU which function to compute by setting six input bits, called control bits, to selected binary values. The exact input-output specification is given in the following figure.




```
Chip name: ALU
Inputs:    x[16], y[16],    // Two 16-bit data inputs
           zx,              // Zero the x input
           nx,              // Negate the x input
           zy,              // Zero the y input
           ny,              // Negate the y input
           f,               // Function code: 1 for Add, 0 for And
           no               // Negate the out output
Outputs:   out[16],         // 16-bit output
           zr,              // True iff out=0
           ng               // True iff out<0
Function:  if zx then x = 0        // 16-bit zero constant
           if nx then x = !x       // Bit-wise negation
           if zy then y = 0        // 16-bit zero constant
           if ny then y = !y       // Bit-wise negation
           if f then out = x + y   // Integer 2's complement addition
                else out = x & y   // Bit-wise And
           if no then out = !out   // Bit-wise negation
           if out=0 then zr = 1 else zr = 0  // 16-bit eq. comparison
           if out<0 then ng = 1 else ng = 0  // 16-bit neg. comparison
Comment:   Overflow is neither detected nor handled.
```

Note that each one of the six control bits instructs the ALU to carry out a certain elementary operation.

Since the overall operation is driven by six control bits, the ALU can potentially compute $2^6 = 64$ different functions. Eighteen of these functions are documented in the figure below.

These bits instruct how to preset the x input		These bits instruct how to preset the y input		This bit selects between + / And	This bit inst. how to postset out	Resulting ALU output
zx	nx	zy	ny	f	no	out=
if zx then x=0	if nx then x=!x	if zy then y=0	if ny then y=!y	if f then out=x+y else out=x&y	if no then out=!out	f(x,y)=
1	0	1	0	1	0	0
1	1	1	1	1	1	1
1	1	1	0	1	0	-1
0	0	1	1	0	0	x
1	1	0	0	0	0	y
0	0	1	1	0	1	!x
1	1	0	0	0	1	!y
0	0	1	1	1	1	-x
1	1	0	0	1	1	-y
0	1	1	1	1	1	x+1
1	1	0	1	1	1	y+1
0	0	1	1	1	0	x-1
1	1	0	0	1	0	y-1
0	0	0	0	1	0	x+y
0	1	0	0	1	1	x-y
0	0	0	1	1	1	y-x
0	0	0	0	0	0	x&y
0	1	0	1	0	1	x|y

Figure 2.6 The ALU truth table. Taken together, the binary operations coded by the first six columns effect the function listed in the right column (we use the symbols !, &, and | to represent the operators Not, And, and Or, respectively, performed bit-wise). The complete ALU truth table consists of sixty-four rows, of which only the eighteen presented here are of interest.