Boolean Arithmetic

Monday, January 24, 2022 12:04 PM

"Counting is the religion of this generation, its hope and salvation." - Gertrude Stein (1874-1946)

Arithmetic Operations

- Addition
- Sign conversion
- Subtraction
- Comparison
- Multiplication
- Division

Start with addition and sign conversion and then build other operations from that.

Binary Numbers

Decimal System is in base 10

$$(6083)_{10} = 6 \cdot 10^3 + 0 \cdot 10^2 + 8 \cdot 10^1 + 3 \cdot 10^0$$

Binary is base 2

$$(10011)_2 = 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 19$$

Nothing is special (not even in math) about base 10. In fact it is quite irritating for computer hardware designers since they constantly have to worry about
binary $\Leftrightarrow$ decimal conversion.

Fixed Word size

Decimal numbers don't work with fixed work sizes there are several numbers bigger and smaller than 16 for example.

Computers use fixed word size such as 8, 16, 32, 64,…

This is often because registers can only hold a certain number of bits.

Using n bits we can measure numbers from $[0, 2^n - 1]$

If you need to represent an number $\geq 2^n$ then you can do so by putting as many n-bit registers together as you need.

Binary addition

Adding binary numbers is similar to adding decimal numbers

	1	0	0	1
+	0	1	0	1
0	1	1	1	0

As you can see there is no overflow

	1	0	1	1
+	0	1	1	1
1	0	0	1	0

This is an example of overflow.

The carrying over of a 1 is called an *overflow*

Signed Binary Numbers

A simple solution is to divide the 2^n binary space into two. The lower half (0xxxxxxx) is for positive numbers and the upper half (1xxxxxxx) is for the negative numbers.

Two's compliment method

Take $2^n - x$ to represent the number in binary (the above method is an easy way to think of it).

$-7 = 2^4 - 7 = 9$
1001
$7 = 7$
0111

Two's compliment has the following attractive properties

- The system codes 2^n signed numbers, ranging from $-(2^{n-1})\ to\ 2^{n-1} - 1$
- The code of any nonnegative number begins with a 0.
- The code of any negative number begins with a 1.

- To obtain the binary code -x from the binary code x, leave all the least significant 0-bits and the first least significant 1-bit of x intact, and flip all of the remaining bits (convert 0's to 1's and vice versa). Alternatively, flip all of the bits and add 1

Method 1:

$$0101 = 5$$
$$1011 = -5$$

$$0111 = 7$$
$$1001 = -7$$

Method 2:

$$0101 = 5$$
$$1010 \rightarrow 1011 = -5$$

$$0111 = 7$$
$$1000 \rightarrow 1001 = -7$$

The application for the two's compliment rule makes it an unsung hero in applied computer science!

Specification

ALU - Arithmetic Logic Unit

Adders -

- *Half-adder:* designed to add two bits
- *Full-adder:* designed to add three bits
- *Adder:* designed to add two *n*-bit numbers
- *Incrementor*: designed to increment any number by 1

Half-adder:

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

```
Chip name: Half-Adder
Input:     a, b
Output:    sum, carry
```

```
Function:  sum = LSB of a + b
           carry = MSB of a + b
```

Full-adder:

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
Input:     a, b, c
Output:    sum, carry
Function:  sum = LSB of a + b + c
           carry = MSB of a + b + c
```

Adder:

```
Chip name: add16
Input:     a[16], b[16]
Output:    out[16]
Function:  Adds two 16-bit numbers.
           The overflow bit is ignored
```

Incrementor:

```
Chip name: Inc16
Input:     in[16]
Output:    out[16]
Function:  out = in + 1
Comment:   overflow bit is ignored.
```

The ALU

The ALU is the computational center of the CPU.

The Hack ALU's set of operations:

zx	nx	zy	ny	f	no	out
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	$x\|y$

Furthermore,

```
If (zx == 1) then x = 0;
If (zy == 1) then y = 0;
If (nx == 1) then x = !x;
If (ny == 1) then y = !y;
If (f  == 1) then out = x + y
             else out = x & y
If (no == 1) then out = !out
```

```
Chip name: ALU
Input:     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,            // if f==1 then out=add(x, y) else out=and(x, y)
            no            // Negate the output
Output:     out[16],      // 16-bit Output
            zr,           // if out==0 zr=1 else zr=0
            ng            // if out< 0 ng=1 else ng=0
Function:
            if zx x=0      // 16-bit 0 constant
            if nx x=!x,    // Bit-wise negation
            if zy y=0      // 16-bit 0 constant
            if ny y=!y,    // Bit-wise negation
            if f out=x + y  // Integer two's compliment addition
            else out=x&y   // Bit-wise And
            if no out=!out // Bit-wise negation
            if out==0 zr=1 else zr=0 // 16-bit equality comparison
            if out> 0 ng=1 else ng=0 // two's compliment comparison
Comment:    The overflow bit is ignored
```