Numbers

Ray Seyfarth

August 31, 2011

Outline

1. Binary numbers
2. Hexadecimal numbers
3. Integers
4. Floating point numbers
5. Converting decimal numbers to floats
6. Floating point mathematics

Binary numbers

- Decimal place value system

$$
\begin{aligned}
15301201 &= 1 * 10^7 + 5 * 10^6 + 3 * 10^5 + 10^3 + 2 * 10^2 + 1 \\
&= 10000000 + 5000000 + 300000 + 1000 + 200 + 1 \\
&= 15301201
\end{aligned}
$$

- Binary place value system

$$
\begin{aligned}
10101111 &= 2^7 + 2^5 + 2^3 + 2^2 + 2 + 1 \\
&= 128 + 32 + 8 + 4 + 2 + 1 \\
&= 175
\end{aligned}
$$

Bit numbering

bit value	1	0	1	0	1	1	1	1
bit position	7	6	5	4	3	2	1	0

- The least significant bit of a byte is bit 0
- The most significant bit is bit 7
- In `yasm` this number could be written as `10101111b`

Decimal to binary conversion

- Convert 741 to binary
- Repeatedly divide by 2 and keep the remainders

division			remainder	bits
741/2	=	370	1	1
370/2	=	185	0	01
185/2	=	92	1	101
92/2	=	46	0	0101
46/2	=	23	0	00101
23/2	=	11	1	100101
11/2	=	5	1	1100101
5/2	=	2	1	11100101
2/2	=	1	0	011100101
1/2	=	0	1	1011100101

Hexadecimal numbers

- Base 16 numbers
- Use as "digits" 0-9 and A-F (or a-f)
- A=10, B=11, C=12, D=13, E=14, F=15

$$
\begin{aligned}
\texttt{0x2b1a} &= 2 * 16^3 + 11 * 16^2 + 1 * 16 + 10 \\
&= 2 * 4096 + 11 * 256 + 16 + 10 \\
&= 8192 + 2816 + 16 + 10 \\
&= 11034
\end{aligned}
$$

Why use hexadecimal?

- Each hexadecimal digit or "nibble" is 4 bits
- `0x2b1a` = `0010 1011 0001 1010`
- `0x2b1a` = `0010101100011010b`
- Counting 32 bits for a binary pattern would be hard
- Hexadecimal is much easier
- `0xdeadbeef` = `11011110101011011011111011101111b`

Converting decimal to hexadecimal

- Convert 40007 to hexadecimal
- Repeatedly divide by 16 and keep the remainders

division			remainder	hex
40007/16	=	2500	7	7
2500/16	=	156	4	47
156/16	=	9	12	c47
9/16	=	0	9	9c47

Integers

- Integers can be 1, 2, 4 or 8 bytes long
- They can be signed or unsigned

Variety	Bits	Bytes	Minimum	Maximum
unsigned	8	1	0	255
signed	8	1	-128	127
unsigned	16	2	0	65535
signed	16	2	-32768	32767
unsigned	32	4	0	4294967295
signed	32	4	-2147483648	2147483647
unsigned	64	8	0	18446744073709551615
signed	64	8	-9223372036854775808	9223372036854775807

Negative integers

- We use the highest-order bit as a sign bit
- 1 for a sign bit means a negative number
- If we stored -1 as 10000001b
- -1 + 1 would be 10000001b + 00000001b = 100000010b
- Then addition would yield -1 + 1 = -2
- There must be a better way to store negatives
- Hopefully, we can use the same circuitry for positives and negatives

Two's complement integers

- To convert a number to its negative, use two's complement
- Flip all the bits
- Add 1
- Let's convert 1 to -1 with 8 bit numbers

```
        00000001 for the absolute value
        11111110 for the complement
        11111111 after adding 1 to the complement
   -1 = 11111111
```

- Two's complement negative numbers work for addition

More 8 bit signed integers

- They form a cycle if you keep adding 1

```
00000000 =    0
00000001 =    1
00000010 =    2
...
01111111 =  127
10000000 = -128
10000001 = -127
10000010 = -126
...
11111110 =   -2
11111111 =   -1
00000000 =    0
```

Addition

- Let's convert and add -29124 + 125

```
29124  = 0111000111000100
Negate = 1000111000111011
Add 1  = 1000111000111100

  125  = 0000000001111101

Now add  1000111000111100
         0000000001111101
         ----------------
         1000111010111001

Negate   0111000101000110
Add 1    0111000101000111
         28999

So -29124 + 125 = -28999
```

Binary multiplication

```
           1010101
*            10101
------------------
           1010101
         1010101
       1010101
------------------
       11011111001
```

Floating point numbers

- 32 bit, 64 bit and 80 bit numbers
- Stored in IEEE 754 format

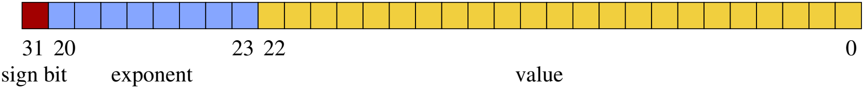
Variety	Bits	Exponent	Exponent Bias	Fraction	Precision
float	32	8	127	23	∼7 digits
double	64	11	1023	52	∼16 digits
long double	80	15	16383	64	19 digits

- Exponents are binary exponents
- An exponent field has the bias added
- A 32 bit exponent field of 128 means a binary exponent 1
- A 32 bit exponent field of 125 means a binary exponent -2
- 0.0 is stored as all bits equal to 0
- Exponent field 255 means "Not a Number"

Binary numbers with binary points

$$
\begin{aligned}
0.1_2 &= 2^{-1} \\
&= 0.5 \\
1.11_2 &= 1 + 2^{-1} + 2^{-2} \\
&= 1 + 0.5 + 0.25 \\
&= 1.75 \\
1001.1001_2 &= 2^3 + 1 + 2^{-1} + 2^{-4} \\
&= 8 + 1 + 0.5 + 0.0625 \\
&= 9.5625 \\
1.0010101 * 2^3 &= 1001.0101 \\
&= 2^3 + 1 + 2^{-2} + 2^{-4} \\
&= 8 + 1 + 0.25 + 0.0625 \\
&= 9.3125
\end{aligned}
$$

Implicit 1 bit



- Normalized floats have exponent fields from 1 to 254
- For these floats there will be at least one 1 bit in the number
- IEEE 754 uses implicit 1 bits
- For non-zero floats, they can be written in "scientific" notation
  - $1011.10101 = 1.01110101 * 2^3$
  - The leading 1 bit is not stored
  - The value (fraction) field is 01110101000000000000000
- So we have 23 bits of fraction with 1 implicit bit = 24 bits
- The sign bit is flipped to negate a float (1 means negative)

Floating point storage

- Consider consider this listing by `yasm`

```
                                      %line 1+1 fp.asm
                                      [section .data]
00000000 00000000                     zero dd 0.0
00000004 0000803F                     one dd 1.0
00000008 000080BF                     neg1 dd -1.0
0000000C 0000E03F                     a dd 1.75
00000010 0000F542                     b dd 122.5
00000014 CDCC8C3F                     d dd 1.1
00000018 F9021550                     e dd 10000000000.0
```

- The bytes are backwards
- 1.0 should be represented logically as 3F800000
- 0 sign bit, 127 exponent field, 0 for the fraction field

Floating point storage (2)

```
4 00000004 0000803F            one dd 1.0
5 00000008 000080BF            neg1 dd -1.0
6 0000000C 0000E03F            a dd 1.75
7 00000010 0000F542            b dd 122.5
```

- All these have a lot of 0 bits in the fractions
- They are all exactly equal to a sum of a few powers of 2
- $1 = 2^0$
- $1.75 = 2^0 + 2^{-1} + 2^{-2}$
- $122.5 = 2^6 + 2^5 + 2^4 + 2^3 + 2^1 + 2^{-1}$
- -1.0 differs from 1.0 only in the sign bit

Floating point storage (3)

```
8 00000014 CDCC8C3F                    d dd 1.1
```

- 1.1 is a repeating binary number
- The number in "proper" order is `3F8CCCCD`
- The exponent field is 127, so the exponent is 1
- The number is $1.00011001100110011001101_2$
- It looks like $1.1 = 1.000\overline{1100}$

Converting decimal numbers to floats

- Determine the sign bit and work with the absolute value
- Convert the whole part of the decimal number
- Convert the fraction
- Express in binary scientific notation
- Build the exponent field by adding 127 bias
- Drop the leading 1 to get the fraction field
- Example: convert -12.25
  - Sign bit is 1
  - Whole part is $12 = 1100_2$
  - Fraction is $0.25 = 0.01$
  - Scientific notation $12.25 = 1.10001_2 * 2^3$

$$
\begin{aligned}
-12.25 &= 1\ 10000010\ 10001000000000000000000 \\
&= \texttt{0xC1440000}
\end{aligned}
$$

Converting decimal number to float (2)

- The only non-obvious step is coverting the fractional part to a binary fraction.
- Suppose you have a decimal number $x = .abcdefgh$
- Then if you multiple x by 2, the only possible result is $2x < 1$ or $1 \le 2x < 2$
- If $2x < 1$, then $x < 0.5$, which means the first bit after the binary point is 0.
- If $2x \ge 1$, then $x \ge 0.5$, which means the first bit after the binary point is 1.
- So we set the first bit and work on the remaining fractional part of $2x$ to get the next bit.
- This process continues until we reach $x = 0$ or we have enough bits.

Converting decimal number to float (3)

- Let's convert -121.6875 to a binary number
- First the sign is 0
- $121 = 1111001_2$
- Now it's time to work on .6875

Multiply		Result	Binary
$.6875 * 2$	$=$	1.375	$.1_2$
$.375 * 2$	$=$	0.75	$.10_2$
$.75 * 2$	$=$	1.5	$.101_2$
$.5 * 2$	$=$	1.0	$.1011_2$

- $-121.6875 = -1111001.1011_2$
- $-121.6875 = -1.1110011011_2 * 2^6$
- As a binary float `1 10000101 11100110110000000000000`
- Expressed in hexadecimal: `0xC2F36000`

Floating point addition

- Let's add 41.275 and 0.315
- 41.275 = `101001.010001100110011010` in binary
- 0.325 = `0.0101000010100011110101110` in binary
- As with decimals, we align the numbers and add

```
   101001.010001100110011010
+       0.0101000010100011110101110
-----------------------------------
   101001.1001011100001010010101110
```

- There are 31 digits in the answer
- The answer must be rounded to 24 bits
- Rounding the last 7 bits means truncation in this case
- We get `0x42265c29` which is 41.59 (approximately)

Floating point multiplication

- Let's multiply 7.5 and 4.375

$$
\begin{array}{rrcr}
 & 7.5 & = & 111.1_2 \\
* & 4.375 & = & 100.011_2 \\
\hline
 & & & 1111_2 \\
 & & & 11110_2 \\
 & & & 111100000_2 \\
\hline
 & & & 100000.1101_2
\end{array}
$$

- Conversion to float format should be apparent by now