Computer Arithmetic

Number Systems

Binary:

Hexadecimal:

Word Size: (Fixed) number of bits used to represent a number

Integer Representation

Representing arbitrary numbers

Human: $-1101.0101_2 = -13.3125_{10}$

Computer:
- Only binary digits
- No minus signs
- No dot (period)

Fixed point Representation:
radix point (binary point) assumed to be to the right of the rightmost digit.

Non Negative Integer Representation

If we want to represent nonnegative integers only

Then

If an *n*-bit sequence of binary digits $b_{n-1}b_{n-2} \ldots b_0$ is interpreted as an unsigned integer A, its value is

$$A = \sum_{i=0}^{n-1} 2^i b_i$$

An 8-bit can represent the numbers from 0 -255

Sign-magnitude representation

Sign-magnitude representation: Most significant bit (*sign bit*) used to indicate the sign and the rest represent the magnitude. if

sign bit = 0 Positive number
sign bit = 1 Negative number

$$A = \begin{cases} A = \sum_{i=0}^{n-2} 2^i b_i & \text{if } a_n = 0 \\ A = -\sum_{i=0}^{n-2} 2^i b_i & \text{if } a_n = 1 \end{cases}$$

+18 = 00010010

-18 = 10010010

Sign-magnitude representation

Problems with sign-magnitude representation:

✪Addition and subtraction:

Require examination of both sign and magnitude

✪Representing zero: +0 and -0

+0 = 00000000

-0 = 10000000

Twos complement representation

Characteristics of twos complement representation and arithmetic

✪Range: -2^{n-1} to $2^{n-1} - 1$, one zero (For an n-bit word)

✪Negation: Take the Boolean complement of each bit of the corresponding positive number and add 1 to the resulting bit pattern

✪Expansion of bit length: Add additional bit positions to the left and fill in with the value of the original sign bit.

✪Overflow rule: If two numbers have the same sign are added, then overflow occurs iif (if and only if) the result has the opposite sign.

✪Subtraction Rule: To subtract B from A, take the twos complement of B and add it to A

Twos complement representation

Conversion between twos complement and decimal

Decimal	Sign Magnitude	Twos Complement
+7	0111	0111
+6	0110	0110
+5	0101	0101
+4	0100	0100
+3	0011	0011
+2	0010	0010
+1	0001	0001
+0	0000	0000

Decimal	Sign Magnitude	Twos Complement
-0	1000	------
-1	1001	1111
-2	1010	1110
-3	1011	1101
-4	1100	1100
-5	1101	1011
-6	1110	1010
-7	1111	1001

Awkward to human, but very convenient for computer….

Twos complement representation

Conversion between twos complement and decimal

$$A = -2^{n-1} b_{n-1} + \sum_{i=0}^{n-2} 2^i b_i$$

Value range for an *n*-bit number

Positive Number Range: $0 \sim 2^{n-2}$

Negative number range: $-1 \sim -2^{n-2}$

Examples:

Twos complement representation

Conversion between different bit lengths

+18 = 00010010 (sign magnitude, 8-bit)
+18 = 0000000000010010 (sign magnitude, 16-bit)
-18 = 10010010 (sign magnitude, 8-bit)
-18 = 1000000000010010 (sign magnitude, 16-bit)

+18 = 00010010 (twos complement, 8-bit)
+18 = 0000000000010010 (twos complement, 16-bit)
-18 = 11101110 (twos complement, 8-bit)
-18 = 1111111111101110 (twos complement, 16-bit)

Fixed point Representation

Integer Arithmetic

Negation

Sign-magnitude: Invert the sign bit

Twos complement:

✪Invert each bit (including the sign bit).
✪Treat the result as unsigned binary integer, and add 1

E.g.

Integer Arithmetic

Addition and Subtraction

```
 1001              1100
+0101             +0100
 1110 = -2        10000 = 0
(a)(-7)+(+5)      (b)(-4)+(+4)

 0011              1100
+0100             +1111
 0111 = 7         11011 = -5
(c)(+3)+(+4)      (d)(-4)+(-1)

 0101              1001
+0100             +1010
 1001 = Overflow  10011 = Overflow
(e)(+5)+(+4)      (f)(-7)+(-6)
```

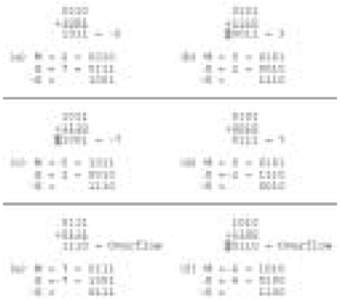
Overflow

Result larger than can be held in the word size being used resulting in overflow.

If two numbers have the same sign are added, then overflow occurs iif (if and only if) the result has the opposite sign.

Carry bit ignored

Integer Arithmetic



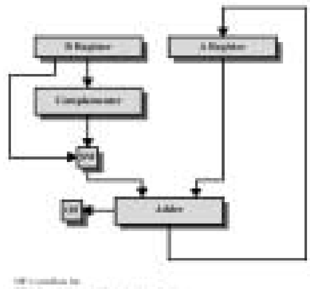
(M - S)

Subtraction

To subtract one number (subtrahend) from another number minuend), take the twos complement (negation) of the subtrahend and add it to the minuend.

Overflow rule applies here also

Integer Arithmetic



Addition and Subtraction Hardware Block Diagram

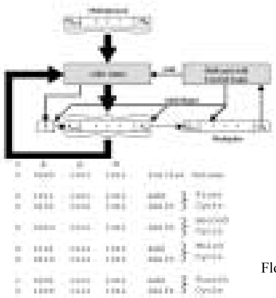
Integer Arithmetic

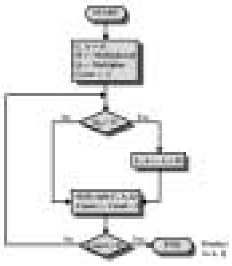
Multiplication: Unsigned binary integers

```
    1011      Multiplicand (11)
   ×1101      Multiplier (13)
    1011  ⎫
   0000   ⎪
  1011    ⎬   Partial products
 1011     ⎭
10001111      Product (143)
```


Integer Arithmetic

Multiplication

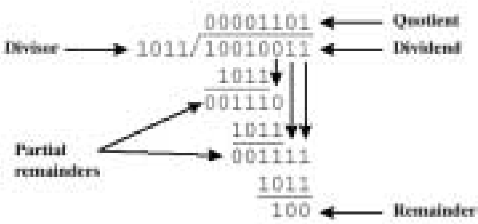




Flowchart for unsigned binary multiplication

Integer Arithmetic (IV)

Division: Unsigned binary integer



Real Numbers

- **Numbers with fractions**
- **Could be done in pure binary**
  1001.1010 = $2^4 + 2^0 + 2^{-1} + 2^{-3}$ =9.625
- **Where is the binary point?**
- **Fixed?**
  Very limited - cannot represent very large or very small numbers
- **Moving?**
  How do you show where it is?

Floating Point Representation

Principles

Scientific notation:

$$543{,}000{,}000{,}000{,}000 = 5.43 \times 10^{14}$$

Slide the decimal point to a convenient location
Keep track of the decimal point use the exponent of 10

Do the same with binary number in the form of

$\pm S \times B^{\pm E}$

- ✪Sign: + or -
- ✪Significant: S
- ✪Exponent: E

Floating Point Representation

Example

- ✪32-bit floating point format.
- ✪Leftmost bit = sign bit (0 positive or 1 negative).
- ✪Exponent in the next 8 bits. Use a biased representation.

A fixed value, called bias, is subtracted from the field to get the true exponent value. Typically, bias = $2^{k-1} - 1$, where k is the number of bits in the exponent field. Also known as excess-N format, where N = bias = $2^{k-1} - 1$. (The bias could take other values)

In this case: 8-bit exponent field, 0 - 255. Bias = 127. Exponent range -127 to +128

- ✪Final portion of word (23 bits in this example) is the significant (sometimes called mantissa).

Floating Point Representation

Many ways to represent a floating point number, e.g.,

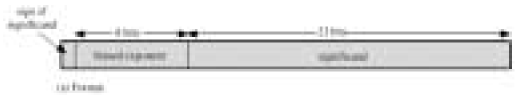
$$0.110 \times 2^{5} \qquad 110 \times 2^{2} \qquad 0.0110 \times 2^{6}$$

Normalization: Adjust the exponent such that the leading bit (MSB) of mantissa is always 1. In this example, a normalized nonzero number is in the form

$$\pm 1.bbb...b \times 2^{\pm E}$$

- ✪Left most bit always 1 - no need to store
- ✪23-bit field used to store 24-bit mantissa with a value between 1 to 2

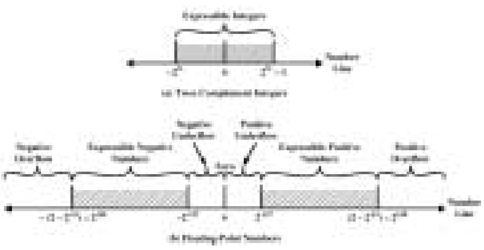
Floating Point Representation



- ✪Sign stored in the first bit
- ✪Left most bit of the TRUE mantissa always 1 - no need to store
- ✪The value of 127 is added to the TRUE exponent to be stored
- ✪The base is 2

Floating Point Representation

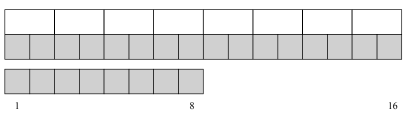
✪Expressible Numbers



Floating Point Representation

✪Range and Precision

The number of individual values - same for any fixed length binary



Range:

Precision:

Floating Point Representation (VII)

IEEE 754 Standard

✪**Single Format and Double Format**

⇨Single Precision format:

- ⇨32 bits, sign = 1 bit, Exponent = 8bits, Mantissa = 32 bits
- ⇨Numbers are normalised to form: $\pm 1.bbb \ldots b \times 2^{\pm E}$; where b = 0 or 1
- ⇨Exponent formatted using excess-127 notation with implied base of 2
- ⇨Theoretical exponent range 2^{-127} to 2^{128}
- ⇨Actuality, exponent values of 0 and 255 used for special values
- ⇨Exponent range restricted to -126 to 127
- ⇨0.0 defined by a mantissa of 0 and the special exponent value of 0
- ⇨Allows + - infinity defined by a mantissa value of 0 and exponent value 255

Floating Point Arithmetic

Addition and Subtraction

✪Check for zero
✪Align the significants
✪Add or subtract the significants
✪Normalise the result

E.g. $0.5566 \times 10^{3} + 0.7778 \times 10^{3}$

$0.5323 \times 10^{2} + 0.7268 \times 10^{-1}$