CS3350B Computer Organization Chapter 2: Synchronous Circuits Prelude

Iqra Batool

Department of Computer Science
University of Western Ontario, Canada

Monday Janaury 29, 2024

Outline

1 Everything on a Computer is a Number

Radix Representations

Radix is the base number in some numbering system.

In a radix r representation digits (d_i) are from the set $\{0, 1, \ldots, r-1\}$

$$x = d_{n-1} \times r^{n-1} + d_{n-2} \times r^{n-2} + \cdots + d_1 \times r^1 + d_0 \times r^0$$

- $r = 10 \implies$ decimal, $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- $r = 2 \implies$ binary, $\{0, 1\}$
- $r = 8 \implies$ octal, $\{0, 1, 2, 3, 4, 5, 6, 7\}$
- $r = 16 \implies$ hexadecimal, $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f\}$

Refresh: Decimal to Binary

$$(13)_{10} = (1 \times 10^1) + (3 \times 10^0)$$

$$(1101)_2 = (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) = 8 + 4 + 0 + 1 = (13)_{10}$$

Unsigned Binary Integers

Unsigned Integers $\implies$ the normal representation

An n-bit number:

$$x = x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_1 2^1 + x_0 2^0$$

- Has a factor up to 2^{n-1}.
- Has a range: 0 to $(2^n - 1)$
- Example

$$
\begin{aligned}
& 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1011_2 \\
=\ & 0 + \cdots + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\
=\ & 0 + \cdots + 8 + 0 + 2 + 1 = 11_{10}
\end{aligned}
$$

- Using 32 bits: 0 to $+4{,}294{,}967{,}295$

Signed Binary Integers (1/2)

How to encode a negative sign?

One's Complement: Invert unsigned representation to get negative.

- Get value by inverting all bits then multiply by -1.
- Leading bit decides if negative or not.
- All positive numbers have the same representation as unsigned.

In one's compliment:

- $(0101)_2 = (0101)_2 = 5$
- $(1101)_2 = -1 \times (0010)_2 = -2$
- $(0000)_2 = (0000)_2 = 0$
- $(1111)_2 = -1 \times (0000)_2 = -0$????

One's compliment is rarely used:

- Signed zero.
- Weird borrowing required in arithmetic.

Signed Binary Integers (2/2)

How to encode a negative sign?

Two's Complement: Invert all the bits with respect to 2^n

- Same as treating leading bit as negative in expansion.
- Leading bit decides if negative or not.
- All positive numbers have the same representation as unsigned.

In two's compliment:

- $(0101)_2 = (0101)_2 = 5$
- $(1101)_2 = -1 \times 2^3 + (0101)_2 = -8 + 5 = \text{-3}$
- $(0000)_2 = (0000)_2 = 0$
- $(1111)_2 = -1 \times 2^3 + (0111)_2 = -1$

Two's Complement

Advantages:

- Arithmetic is the same whether positive or negative:

$$(0101)_2 = 5$$
$$+ \ (1101)_2 = -3$$
$$\overline{\quad(0010)_2 = 2 \quad} \quad \text{(Throw away carry bit)}$$

- No signed 0.
- One extra value represented with same number of bits.

For an n-bit number:

- Range of values is -2^{n-1} to $2^{n-1} - 1$

Same Bits Different Numbers

It is important to realize that the same bit pattern can represent different numbers.

$$(1001\ 1010)_2 \implies (154)_{10} \quad \text{interpretted as unsigned}$$
$$\implies (-102)_{10} \quad \text{interpretted as two's compliment}$$

Can be disastrous in programming!

```
unsigned int a = (1 << 31); // a = 2147483648
int b = a;                  // b = -2147483648
```

Signed Negation

In two's compliment, **bit-wise complement** then **add 1**.

$$6 = (0110)_2 = (0 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (0 \times 2^0)$$

$$\Downarrow \text{ compliment}$$

$$(1001)_2 = (-1 \times 2^3) + (0 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) = -8 + 1$$

$$\Downarrow \text{ add one}$$

$$(1001)_2 + (0001)_2 = (1010)_2 = -8 + 0 + 2 + 0 = -6$$

Also works in reverse! (from negative to positive)

↪ Still, compliment then add 1.

↪ $-6 = (1010)_2 \Rightarrow (0101)_2 + 1 \Rightarrow (0110)_2 = 6$

Signed Extension

Signed Extension:

- Represent a number using more bits but keep numerical value.
- Very easy in two's compliment!
- Copy the signed bit to the left until desired number of bits.

Examples: 8-bit to 16-bit

- 2: 0000 0010 $\Rightarrow$ 0000 0000 0000 0010
- -2: 1111 1110 $\Rightarrow$ 1111 1111 1111 1110
- -10: 1111 0110 $\Rightarrow$ 1111 1111 1111 0110

Note: Truncation (representing a number using less bits) is tricky and you must know what you're doing.

Logical Shift

Logical Shift:

- Shift the bits left or right a specified number of times.
- Fills the vacancies with 0s on shift left and shift right.
- Throw away any bits that flow out.
- << (shift left) and >> (shift right) in C (unsigned).

Examples (in 8 bits):

- $2 << 3 = (0000\ 0010) << 3 = (0001\ 0000) = 16$.
- $8 >> 2 = (0000\ 1000) >> 2 = (0000\ 0010) = 2$.
- $-4 >> 1 = (1111\ 1100) >> 1 = (0111\ 1110) = 126$.
  - ↪ This last one is ambiguous if it is logical or arithmetic shift. In high-level programming languages the right shift operator is usually an *arithmetic shift*...

Arithmetic Shift

Arithmetic Shift:

- Shift the bits left or right a specified number of times.
- Fills the vacancies with 0s on shift left.
- Fills the vacancies with 1s on shift right if number is negative.
- Fills the vacancies with 0s on shift right if number is positive.
- Throw away any bits that flow out.
- << (shift left) and >> (shift right) in C (signed).

Examples (in 8 bits):

- $2 << 3 = (0000\ 0010) << 3 = (0001\ 0000) = 16$.
- $8 >> 2 = (0000\ 1000) >> 2 = (0000\ 0010) = 2$.
- $-4 >> 1 = (1111\ 1100) >> 1 = (1111\ 1110) = -2$.