Number Systems and Circuits for Addition

CS 231

Dianna Xu

1

Decimal Number System

(Base-10 number system)

123

$= \mathbf{1} \times 100 + \mathbf{2} \times 10 + \mathbf{3}$

$= \mathbf{1} \times 10^2 + \mathbf{2} \times 10^1 + \mathbf{3} \times 10^0$

$= 123_{10}$

base

2

Binary Number System

(Base-2 number system)

Bit: **B**inary Dig**it**

101_2

$= \mathbf{1} \times 2^2 + \mathbf{0} \times 2^1 + \mathbf{1} \times 2^0$

$= \mathbf{1} \times 4 + \mathbf{0} \times 2 + \mathbf{1}$

$= 5_{10}$

3

Capacity of Binary Numbers

- 1 bit can distinguish 2 states (0 or 1).
- An n-bit binary number can distinguish 2^n states.

4

Octal Number System

(Base-8 number system)

173_8

$= \mathbf{1} \times 8^2 + \mathbf{7} \times 8^1 + \mathbf{3} \times 8^0$

$= \mathbf{1} \times 64 + \mathbf{7} \times 8 + \mathbf{3}$

$= 123_{10}$

5

Hexadecimal Number System

(Base-16 number system)

Character correspondence:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

$9AB_{16}$

$= \mathbf{9} \times 16^2 + \mathbf{10} \times 16^1 + \mathbf{11} \times 16^0$

$= \mathbf{9} \times 256 + \mathbf{10} \times 16 + \mathbf{11}$

$= 2475_{10}$

ex

6

Hexadecimal

- Often written with a ‘0x’ prefix
  - 0x10 is 10_{16} or 16_{10}
  - 0x100 is 100_{16}, or 256_{10}
- Binary numbers easily translate:
  - In blocks of 4

Table 1.5.3

Decimal	Hexadecimal	4-Bit Binary Equivalent
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111


```
7f 45 4c 46 01 02 01 00 00 00 00 00 00 00 00 00   .ELF............
00 02 00 02 00 00 00 01 00 01 06 68 00 00 00 34   ...........h...4
00 00 12 b8 00 00 00 00 00 34 00 20 00 05 00 28   .........4. ...(
00 19 00 17 00 00 00 06 00 00 00 34 00 01 00 34   ...........4...4
00 00 00 00 00 00 00 a0 00 00 00 a0 00 00 00 05   ................
00 00 00 00 00 00 00 03 00 00 00 d4 00 00 00 00   ................
00 00 00 00 00 00 00 11 00 00 00 00 00 00 00 04   ................
00 00 00 00 00 00 00 01 00 00 00 00 00 01 00 00   ................
00 00 00 00 00 00 08 4e 00 00 08 4e 00 00 00 05   .......N...N....
00 01 00 00 00 00 00 01 00 00 08 50 00 02 08 50   ...........P...P
00 00 00 00 00 00 01 28 00 00 01 2c 00 00 00 07   .......(...,....
00 01 00 00 00 00 00 02 00 00 08 c0 00 02 08 c0   ................
00 00 00 00 00 00 00 a8 00 00 00 00 00 00 00 07   ................
00 00 00 00 2f 75 73 72 2f 6c 69 62 2f 6c 64 2e   ..../usr/lib/ld.
73 6f 2e 31 00 00 00 00 00 00 00 1f 00 00 00 2a   so.1...........*
00 00 00 1f 00 00 00 27 00 00 00 22 00 00 00 00   .......'..."....
00 00 00 20 00 00 00 00 00 00 00 00 00 00 00 00   ... ............
```

8

Byte

1 Byte

- 8 bits
- Representable by 2 hexadecimal characters
- Can distinguish 256 (2^8) states

9

Base-*k*-to-Decimal Conversion

- 101_2
  $= \mathbf{1} \times 2^2 + \mathbf{0} \times 2^1 + \mathbf{1} \times 2^0$
  $= \mathbf{1} \times 4 + \mathbf{0} \times 2 + \mathbf{1}$
  $= 5_{10}$
- ☺☯💣$_k$ =
  = ☺ × k^2 + ☯ × k^1 + 💣 × k^0
  $= ?_{10}$

More generally,

$$value_{10} = \sum_{i=0}^{n} \left(p_i \times k^i\right)$$

10

Decimal-to-Binary Conversion

$5_{10} = 101_2$

2) 5
2) 2 ... 1 ← the rightmost bit (LSB)
1 ... 0 ← the second bit from the right
↑
the leftmost bit (MSB)
MSB = Most Significant Bit
LSB = Least Significant Bit

11

Decimal to hexadecimal

$2475_{10} = 9AB_{16}$

16) 2475
16) 154 ... 11← the rightmost bit (LSB)
9 ... 10←second bit from the right
↑
the leftmost bit (MSB)
Decimal to Base-*k* conversions work the same way

12

How to add binary numbers

- Consider adding two 1-bit binary numbers *x* and *y*
  - 0+0 = 0
  - 0+1 = 1
  - 1+0 = 1
  - 1+1 = 10

x	*y*	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

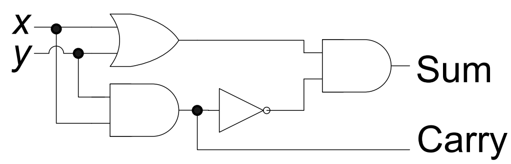
- Carry is *x* AND *y*
- Sum is *x* XOR *y*
- The circuit to compute this is called a half-adder

13

The half-adder

- Sum = *x* XOR *y*
- Carry = *x* AND *y*

x	*y*	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



14

Using half adders

- We can then use a half-adder to compute the sum of two binary numbers?

```
   1 0 0
   1 1 0 0
 + 1 1 1 0
   ? 0 1 0
```

15

How to fix this

- We need to create an adder that can take a carry bit as an additional input
  - Inputs: *x*, *y*, carry in
  - Outputs: sum, carry out
- This is called a full adder
  - Will add *x* and *y* with a half-adder
  - Will add the sum of that to the carry in
- What about the carry out?
  - Final CO is 1 if:
  - *x*+*y* = 10
  - *x*+*y* = 01 and carry in = 1

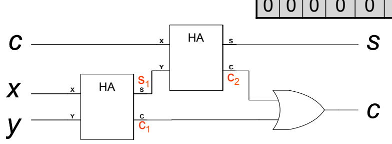
x	*y*	*c*	carry	sum
1	1	1	1	1
1	1	0	1	0
1	0	1	1	0
1	0	0	0	1
0	1	1	1	0
0	1	0	0	1
0	0	1	0	1
0	0	0	0	0

16

The full adder

- The "HA" boxes are half-adders

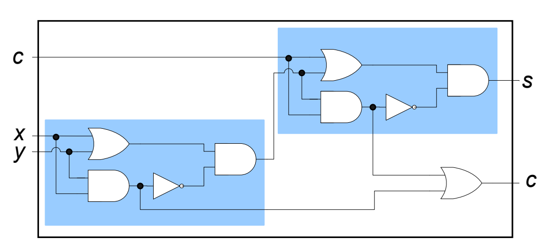
x	*y*	*c*	s_1	c_1	c_2	carry	sum
1	1	1	0	1	0	1	1
1	1	0	0	1	0	1	0
1	0	1	1	0	1	1	0
1	0	0	1	0	0	0	1
0	1	1	1	0	1	1	0
0	1	0	1	0	0	0	1
0	0	1	0	0	0	0	1
0	0	0	0	0	0	0	0



17

The full adder

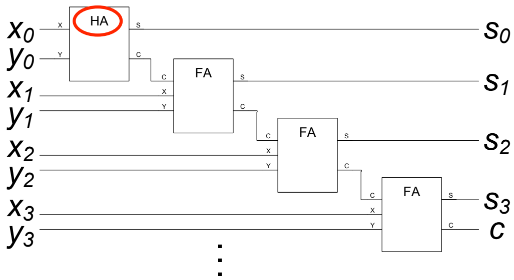
- The full circuitry of the full adder



18

Adding bigger binary numbers

- Just chain full adders together



Parallel adders and number of gates

- A half adder has 4 logic gates
- A full adder has two half adders plus an OR gate
  - Total of 9 logic gates
- To add *n* bit binary numbers,
  - 1 HA + *n*-1 FAs
- To add 32 bit binary numbers,
  - 1 HA + 31 FA = 4+9*31 = 283 logic gates
- To add 64 bit binary numbers,
  - 1 HA + 63 FA = 4+9*63 = 571 logic gates

20

More about logic gates

- To implement a logic gate in hardware, you use a transistor
- Transistors are all enclosed in an "IC", or integrated circuit
- 1971 – Intel's first microprocessor (4004): 2300 transistors
- 1993 – Intel Pentium processor: 3.1 million
- 2006 – Dual-core Itanium 2: 1.7 billion
- 2011 – 10-core Xeon Westmere-Ex: 2.6 billion

21

Two's Complement

- Given a positive integer *a*, the two's complement of *a* is the n-bit representation of $2^n - a$
- $2^8 - 35 = 256 - 35 = 221 = 11011101_2$
- *a*'s two's complement represents *-a*
- Always relative to a fixed bit length
- Bit length of 32 and 64 are most commonly used

22

One's Complement

- An easier way to calculate two's complement
- $2^8 - a = (2^8 - 1) - a + 1$
- $2^8 - 1 = 11111111_2$
- Subtracting any binary number from all 1's is equivalent to negating all bits, i.e. taking the one's complement

23

Example

$$2^8 - 35 = (2^8 - 1) - 35 + 1 =$$

$$11111111_2$$

$$-$$

$$00100011_2$$

$$11011100_2 + 1 = 11011101_2 = 221$$

24

Two's Complement Again

- To find the two's complement of a positive integer *a*:
  - Write the n-bit binary representation for *a*
  - Negate all bits
  - Add 1 to the resulting binary notation

25

8-bit Representations

Integer	8-bit Binary	2's complement
127	01111111	
126	01111110	
...	...	
2	00000010	
1	00000001	
0	00000000	
-1	11111111	2^8 - 1
-2	11111110	2^8 - 2
-3	11111101	2^8 - 3
...	...	
-127	10000001	2^8 - 127
-128	10000000	2^8 - 128

26

Addition with Negative Numbers

$$64 - 15 = 64 + (-15) =$$
$$01000000_2 + ((11111111_2 - 00001111_2) + 1_2) =$$
$$01000000_2 + 11110001_2$$
$$01000000_2$$
$$11110001_2$$
$$00110001_2 = 49$$

27

From ThinkGeek (http://www.thinkgeek.com)