Fundamentals of Computer Architecture (2)

Dr. Qin Zhou

Coventry University
email: q.zhou@coventry.ac.uk

Topics Covered

- Number representation - decimal, binary, hexadecimal and Binary Coded Decimal (BCD);
- Conversion between different bases;
- Binary arithmetic;
- Signed representations - sign and modulus, 1's complement, 2's complement and floating point;
- Logic operations - AND, OR and NOT;
- Data representation - ASCII and Unicode

Representing Numbers
Base 10 - Decimal

The digits:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

- The decimal number 947 in powers of 10 is:

$$9 \times 10^{2} + 4 \times 10^{1} + 7 \times 10^{0}$$

Value	**100** **(10^2)**	**10** **(10^1)**	**1** **(10^0)**
Decimal number	9	4	7

Representing Numbers
Base 2 – Binary

Digits: 0, 1

- The binary number 11001 in powers of 2 is:

$$1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

$$= 16 + 8 + 0 + 0 + 1 = 25$$

Value	**16 (2^4)**	**8 (2^3)**	**4 (2^2)**	**2 (2^1)**	**1 (2^0)**
Binary number	1	1	0	0	1

Representing Numbers

- When the radix of a number is something other than 10, the base is denoted by a subscript.
  - Sometimes, the subscript 10 is added for emphasis:

$$11001_2 = 25_{10}$$

Representing Numbers Base 16 - Hexadecimal

The digits:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

- The decimal number $53F_{16}$ in powers of 16 is:

$$5 \times 16^{2} + 3 \times 16^{1} + 15 \times 16^{0}$$
$$= 5 \times 256 + 3 \times 16 + 15 \times 1 = 1343$$

Value	**256 (16^2)**	**16 (16^1)**	**1 (16^0)**
Decimal number	5	3	F

BCD – Binary Coded Decimal

BCD Representation

Decimal Digit	BCD 8 4 2 1
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1

Example:

Decimal: 91

Binary : 10010001

Bit, Byte and …

A ***bit*** is the most basic unit of information in a computer.

- It is a state of “on” or “off” in a digital circuit.
- Sometimes they represent **high** or **low** voltage

A ***byte*** is a group of eight bits.. It is the smallest possible *addressable* unit of computer storage.

…Word

A ***word*** is a contiguous group of bytes.

- Words can be any number of bits or bytes.
- Word sizes of 16, 32, or 64 bits are most common.

Converting between Bases

Decimal ⟺ Binary Conversions

- Binary → Decimal:

8-bit binary number 00100111

Value	128 (2^7)	64 (2^6)	32 (2^5)	16 (2^4)	8 (2^3)	4 (2^2)	2 (2^1)	1 (2^0)
Binary number	0	0	1	0	0	1	1	1

$$1 \times 2^5 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 39$$

$$00100111_2 = 39_{10}$$

Converting between Bases

Decimal ⟺ Binary Conversions

- Decimal → Binary : 28

Value	128 (2^7)	64 (2^6)	32 (2^5)	16 (2^4)	8 (2^3)	4 (2^2)	2 (2^1)	1 (2^0)
Binary number	0	0	0	1	1	1	0	0

$$28 - 1 \times 2^4 = 12$$

$$12 - 1 \times 2^3 = 4$$

$$4 - 1 \times 2^2 = 0$$

So, $28_{10} = 00011100_2$

Converting between Bases

Hex ⟷ Binary Conversions

- Hex → Binary : $ABCD_{16}$ or 0xABCD

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

$ABCD_{16}$

$= 1010101111001101_2$

Converting between Bases

Hex ⟺ Binary Conversions

Binary → Hex : 1010001001000100_2

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

1010001001000100_2

=

$A244_{16}$

Converting between Bases

Decimal ⟷ Hex Conversions

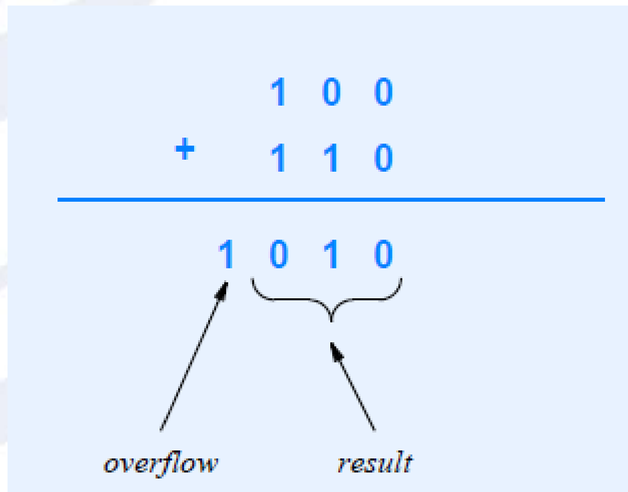
- Decimal → Hex :

    Decimal → Binary → Hex

- Hex → Decimal:

    Hex → Binary → Decimal

Binary Arithmetic



Signed Integer Representation

- The conversions we have so far presented have involved only positive numbers.
- To represent negative values, computer systems allocate the highest-order bit to indicate the sign of a value.
  - The highest-order bit is the leftmost bit in a byte. It is also called the most significant bit.
- The remaining bits contain the value of the number.

Signed Integer Representation

- There are three ways in which signed binary numbers may be expressed:
  - Signed magnitude,
  - One’s complement and
  - Two’s complement.
- In an 8-bit number, signed magnitude representation places the absolute value of the number in the 7 bits to the right of the sign bit.

Signed Integer Representation

- For example, in **8-bit signed magnitude,** positive 3 is: `00000011`
- Negative 3 is: `10000011`
- Computers perform arithmetic operations on signed magnitude numbers in much the same way as humans carry out pencil and paper arithmetic.
  - Humans often ignore the signs of the operands while performing a calculation, applying the appropriate sign after the calculation is complete.

Signed Integer Representation

- Example:
  - Using signed magnitude binary arithmetic, find the sum of 75 and 46.
- First, convert 75 and 46 to binary, and arrange as a sum, but separate the (positive) sign bits from the magnitude bits.

```
0     1001011
0  +  0101110
      -------
```

Signed Integer Representation

- Example:
  - Using signed magnitude binary arithmetic, find the sum of 75 and 46.
- Just as in decimal arithmetic, we find the sum starting with the rightmost bit and work left.

```
0   1001011
0 + 0101110
  ---------
          1
```

Signed Integer Representation

- Example:
  - Using signed magnitude binary arithmetic, find the sum of 75 and 46.
- In the second bit, we have a carry, so we note it above the third bit.

```
          1
0   1001011
0 + 0101110
  ---------
         01
```

Signed Integer Representation

- Example:
  - Using signed magnitude binary arithmetic, find the sum of 75 and 46.
- The third and fourth bits also give us carries.

$$
\begin{array}{cccccccccc}
 & & & & 1 & 1 & 1 & & & \\
0 & & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\
0 & + & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\
\hline
 & & & & & 1 & 0 & 0 & 1
\end{array}
$$

Signed Integer Representation

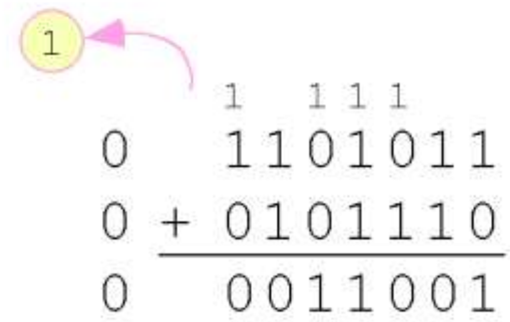
- Example:
  - Using signed magnitude binary arithmetic, find the sum of 75 and 46.
- Once we have worked our way through all eight bits, we are done.

```
         1 1 1
0    1 0 0 1 0 1 1
0  + 0 1 0 1 1 1 0
   ---------------
0    1 1 1 1 0 0 1
```

In this example, we were careful to pick two values whose sum would fit into seven bits. If that is not the case, we have a problem.

Signed Integer Representation

- Example:
  - Using signed magnitude binary arithmetic, find the sum of 107 and 46.
- We see that the carry from the seventh bit *overflows* and is discarded, giving us the erroneous result: 107 + 46 = 25.



Signed Integer Representation

- Signed magnitude representation is easy for people to understand, but it requires complicated computer hardware.
- Another disadvantage of signed magnitude is that it allows two different representations for zero: positive zero and negative zero.
- For these reasons (amongst others) computers systems employ *complement systems* for numeric value representation.

Signed Integer Representation

One's complement: flip the bits of a binary number.

Signed Integer Representation

- For example, in 8-bit one's complement, positive 3 is: 00000011
- Negative 3 is: 11111100
- In one's complement, as with signed magnitude, negative values are indicated by a 1 in the highest order bit.
- Complement systems are useful because they eliminate the need for subtraction. The difference of two values is found by adding the minuend to the complement of the subtrahend.

Signed Integer Representation

- With one's complement addition, the carry bit is "carried around" and added to the sum.
  - Example**:** Using one's complement binary arithmetic, find the sum of 48 and - 19

```
1   1 1
    00110000
    11101100
    --------
    00011100
         + 1
    --------
    00011101
```

We note that 19 in one's complement is `00010011`,
so -19 in one's complement is: `11101100`.

Signed Integer Representation

- Although the “end-around carry” adds some complexity, one’s complement is simpler to implement than signed magnitude.
- But, it still has the disadvantage of having two different representations for zero: positive zero and negative zero.
- Two’s complement solves this problem.

Signed Integer Representation

- To express a value in two's complement:
  - If the number is positive, just convert it to binary and you're done (similar to one's complement).
  - If the number is negative, find the one's complement of the number and then add 1.
- Example:
  - In 8-bit one's complement, positive 7 is: `00000111`
  - Negative 7 in one's complement is: `11111000`
  - Adding 1 gives us -7 in two's complement form: `11111001`.

Signed Integer Representation

The simple rule for obtaining the 2's complement representation of the negative of a number is

– Flip the bits

– Add 1

	Decimal Representation
0 0 0 0 0 1 1 1	+7
1 1 1 1 1 0 0 0	Flip the bits
1	Add 1
1 1 1 1 1 0 0 1	Result represents –7

Signed Integer Representation

- With two's complement arithmetic, all we do is add our two binary numbers. Just discard any carries emitting from the highest order bit.
  - Example: Using two's complement binary arithmetic, find the sum of 48 and - 19.

$$\begin{array}{rr} 1 & 1\,1 \\ & 0\,0\,1\,1\,0\,0\,0\,0 \\ + & 1\,1\,1\,0\,1\,1\,0\,1 \\ \hline & 0\,0\,0\,1\,1\,1\,0\,1 \end{array}$$

We note that 19 in one's complement is: `00010011`,
so -19 in one's complement is: `11101100`,
and -19 in two's complement is: `11101101`.

Signed Integer Representation

- When we use any finite number of bits to represent a number, we always run the risk of the result of our calculations becoming too large to be stored in the computer.
- While we can't always prevent overflow, we can always *detect* overflow.
- In complement arithmetic, an overflow condition is easy to detect.

Signed Integer Representation

- Example:
  - Using two's complement binary arithmetic, find the sum of 107 and 46.
- We see that the nonzero carry from the seventh bit *overflows* into the sign bit, giving us the erroneous result: 107 + 46 = -103.

```
   1 1   1 1 1
   0 1 1 0 1 0 1 1
 + 0 0 1 0 1 1 1 0
 -----------------
   1 0 0 1 1 0 0 1
```

<u>Rule for detecting signed two's complement overflow:</u> When the "carry in" and the "carry out" of the sign bit differ, overflow has occurred.

Signed Integer Representation

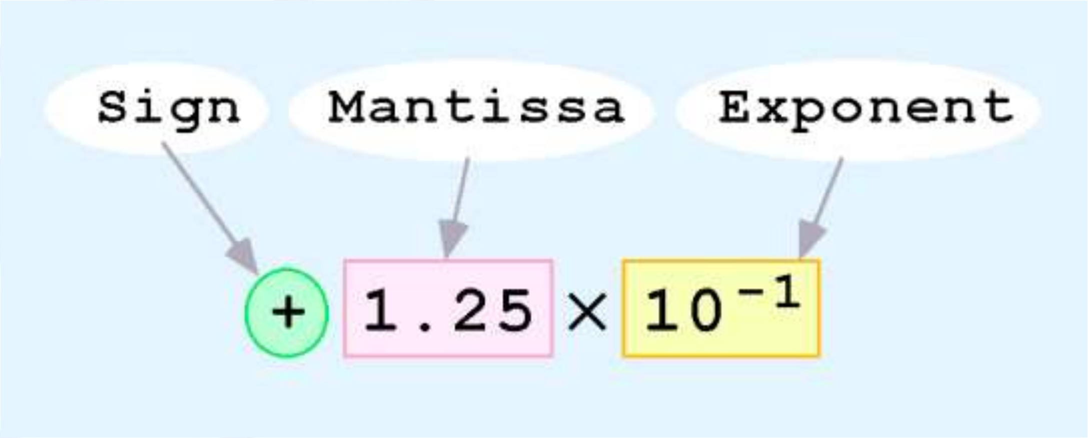
- Signed and unsigned numbers are both useful.
  - For example, memory addresses are always unsigned.
- Using the same number of bits, unsigned integers can express twice as many values as signed numbers.
- Trouble arises if an unsigned value “wraps around.”
  - In four bits: 1111 + 1 = 0000.
- Good programmers stay alert for this kind of problem.

Floating-Point Representation

- The signed magnitude, 1’s complement, and 2’s complement representations as such are not useful in scientific or business applications that deal with real number values over a wide range.
- Floating-point representation solves this problem.

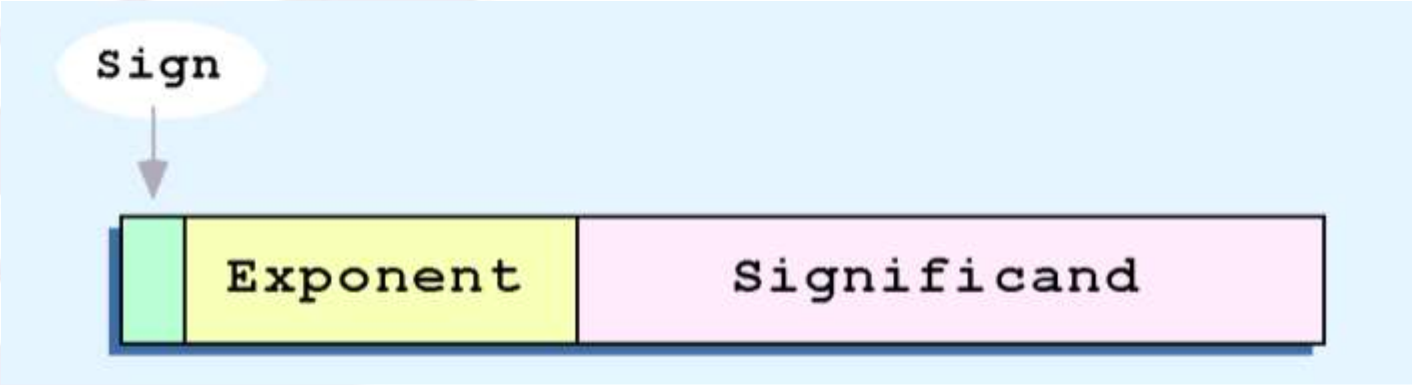
Floating-Point Representation

- Computers use a form of scientific notation for floating-point representation
- Numbers written in scientific notation have three components:



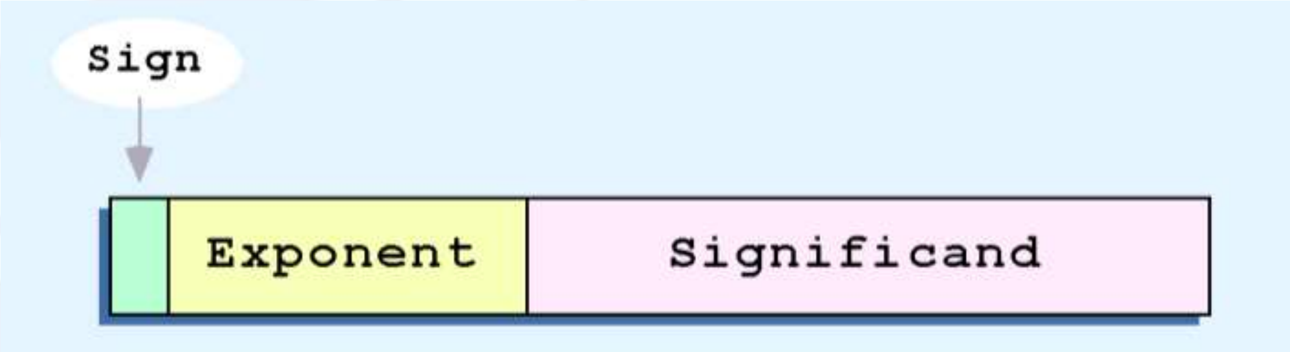
Floating-Point Representation

- Computer representation of a floating-point number consists of three fixed-sized fields:



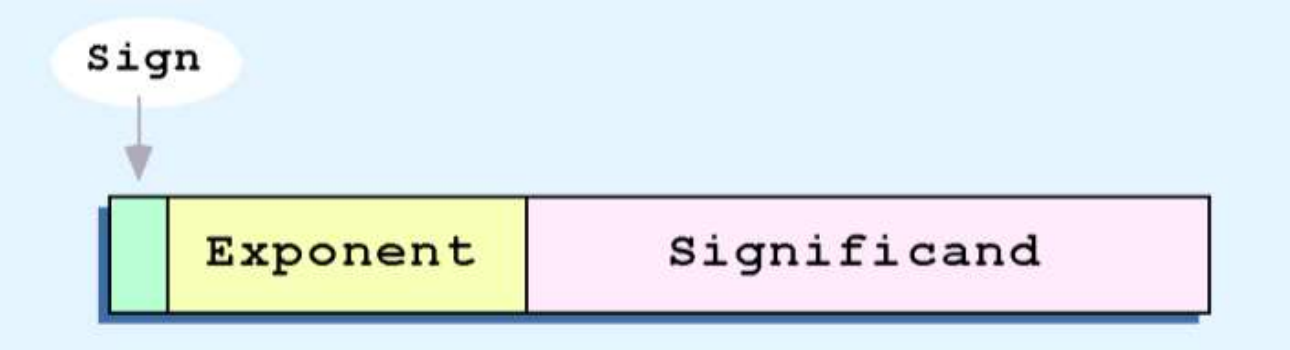
- This is the standard arrangement of these fields.

Floating-Point Representation



- The one-bit sign field is the sign of the stored value.
- The size of the exponent field, determines the range of values that can be represented.
- The size of the significand determines the precision of the representation.

Floating-Point Representation



- The IEEE-754 *single precision* floating point standard uses an 8-bit exponent and a 23-bit significand.
- The IEEE-754 *double precision* standard uses an 11-bit exponent and a 52-bit significand.

Logical Operations
Digital Logic

- Can implement Boolean functions with transistors
- Five volts represents Boolean *1*
- Zero volts represents Boolean *0*

Logical Operations
Boolean Logic

- Mathematical basis for digital circuits
- Three basic functions: *and, or,* and *not*

A	B	A and B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A or B
0	0	0
0	1	1
1	0	1
1	1	1

A	not A
0	1
1	0

Coventry University

Logical Operations
Truth Tables For Nand and Nor Gates

A	B	A nand B
0	0	1
0	1	1
1	0	1
1	1	0

A	B	A nor B
0	0	1
0	1	0
1	0	0
1	1	0

Logical Operations

0	0	1	0	0	1	1	1	X
0	0	1	0	1	1	1	0	Y
0	0	1	0	0	1	1	0	X AND Y

0	0	1	0	0	1	1	1	X
0	0	1	0	1	1	1	0	Y
0	0	1	0	1	1	1	1	X OR Y

0	0	1	0	0	1	1	1	X
1	1	0	1	1	0	0	0	NOT X

Dealing with Text ASCII Codes

High Hexadecimal Digit (columns) / **Low Hexadecimal Digit** (rows)

	0	1	2	3	4	5	6	7
0	NUL	DCL		0	@	P	‘	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	’	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	–	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

ASCII

– For example, the ASCII code for the letter ‘R’ is found as follows:

- The column that ‘R’ is in is labelled with the hexadecimal digit 5;
- The row that ‘R’ is in is labelled with the hexadecimal digit 2;
- This produces the hex value 0x52.

– ASCII is being replaced by 16-bit Unicode.

References

1. Douglas E. Comer: Essentials of Computer Architecture. Available: http://www.eca.cs.purdue.edu

2. Mark Burrell: Fundamentals of Computer Architecture.

Available: http://www.brittunculi.com/foca/materials/

3. Linda Null and Julia Lobur. The Essentials of Computer Organization And Architecture.

Available: http://users.dickinson.edu/~jmac/courses/previous/fall-2008-comp251/book-powerpoints/