Interpreting Bits: Binary and Hexadecimal Numbers

(hex)

Z. Jerry Shi
Department of Computer Science and Engineering
University of Connecticut

CSE3666: Introduction to Computer Architecture

Outline

- Binary numbers
  - Addition and subtraction
- Two's complement numbers
  - Addition and subtraction
  - Negation
  - Sign extension
- Hexadecimal numbers
- ASCII
- Practice

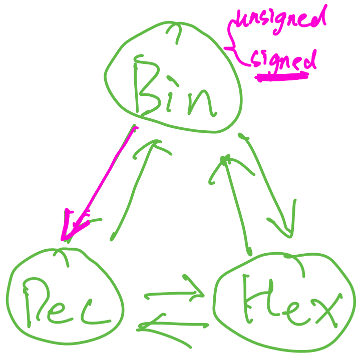
  https://zhijieshi.github.io/cse3666/binarynumbers/

n	2^n
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048
12	4096

Reading: Section 2.4, and hex to binary conversion in Section 2.5

Questions

- What are the decimal representations of the following binary numbers? What can we learn from these exercises?



Q1: 1101 → $(13)_{10}$

$8+4+1$

Q2: 11010

$2^4+2^3+2^1=(26)_{10}$

Q3: 110100

$2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

$2^5+2^4+2^2=(52)_{1}$

Questions

- What are the decimal representations of the following binary numbers? What can we learn from these exercises?

Q1: 1101

$$1\times2^3+1\times2^2+0\times2^1+1\times2^0=13$$

1101

Q2: 11010

$$1\times2^4+1\times2^3+0\times2^2+1\times2^1+0\times2^0$$
$$=2\times(1\times2^3+1\times2^2+0\times2^1+1\times2^0)+0=26$$

Q3: 110100

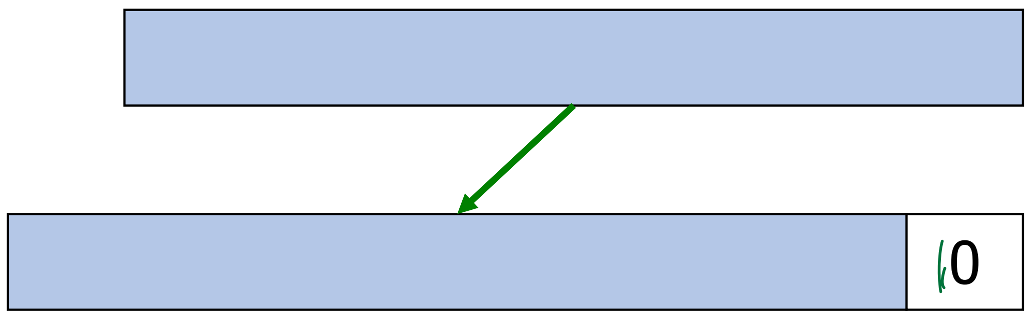
Can you quickly find out the answer to Q3?

Shift bits

Given v,

- Shift left by one bit, the value becomes $2v$

= ×2



- Shift right by one bit, the value becomes v / 2

= /2

n-bit binary numbers

- Very often, the number of bits available is fixed
- The range of values that can be represented by n bits is

  0 to $2^n - 1$

$x = \{0, 1\}$

$xx: \{00, 01, 10, 11\}$

xxx: 000, 001, 010, 011, 100, 101, 110, 111

$xxxx$: 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111

n bit: 2^n

$0 \sim 2^n - 1$

For example:

$0 \quad 2^{10} - 1$

10 bits can represent any values from 0 to 1,023

16 bits can represent any values from 0 to 65,535 $2^{16} - 1$

32 bits can represent any values from 0 to 4,294,967,295

$(0, \quad 2^{32} - 1)$

Powers of 2:

1K: $2^{10} = 1024$ 64K: $2^{16} = 65,536$ 1M: $2^{20} = 1,048,576$

Question

- At least, how many bits do you need to represent 100 different values?

$2^5 = 32$ X

$2^6 = 64$ X

100

$2^7 = 128$

$2^8 = 256$

A. 5
B. 6
C. 7
D. 8
E. Don't know

Addition of numbers

Decimal numbers

$$
\begin{array}{rcccc}
 & 6 & 5 & 3 & 9 \\
+ & 7 & 1 & 6 & 3 \\
1 & 0 & 1 & 1 & \\
\hline
 & 3 & 7 & 0 & 2
\end{array}
$$

Carries

Binary numbers

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

(10)

(2)

1 1 1 1

1 1 1 1

Subtraction of binary numbers

0b1101 – 0b0111

$$\begin{array}{r} 13 \\ 9 \\ \hline 5 \end{array}$$

$$\begin{array}{ccccc} & \cancel{1} & \cancel{1} & \cancel{0} & 1 \\ - & 0 & 1 & 1 & 1 \\ \hline & 0 & 1 & 1 & 0 \end{array}$$

How about negative numbers?

-3 -2 -1 0 1 2 3 4 ?

Suppose we have 3 bits.

-4 -3 -2 -1 0 1 2 3 ?

We know how to convert each number to decimal.

How can we represent negative numbers with bits?

mapping

?

Bits	In decimal	?
000	0	
001	1	
010	2	
011	3	
100	4	
101	5	
110	6	
111	7	

Two's complement numbers

- The most popular method is two's complement numbers
  - There are other schemes. Almost all processors now use 2's complement
- Use half of bit patterns for negative values (which half?)

Bits	As binary	As 2's complement
`000`	`0`	0
`001`	`1`	1
`010`	`2`	2
`011`	`3`	3
`100`	`4`	(~~-1~~) ? -4
`101`	`5`	(~~-2~~) ? -3
`110`	`6`	(~~-3~~) ? -2
`111`	`7`	(~~-4~~) ? -1

Reading two's complement numbers

Given an *n*-bit 2's complement number

$$b_{n-1}\, b_{n-2} \ldots b_2 b_1 b_0$$

The value is

$$-b_{n-1}\times 2^{n-1} + b_{n-2}\times 2^{n-2} + \cdots + b_2\times 2^2 + b_1\times 2^1 + b_0\times 2^0$$

Another way:

If the sign is 0, the value = the unsigned value.

If the sign is 1, the value = the unsigned value -2^n

4-bit two's complement number:

2^3 2^2 2^1 2^0

```
0b1001 = -8 + 0 + 0 + 1 = -7 = 9 - 16
0b1100 = -8 + 4 + 0 + 0 = -4 = 12 - 16
```

2^3 2^2

Example: 3-bit binary numbers

We have two ways to interpret the bits

unsigned: binary numbers (without sign)

signed: two's complement numbers

Bits	Unsigned	Signed
000	0	0
001	1	1
010	2	2
011	3	3
100	4	-4
101	5	-3
110	6	-2
111	7	-1

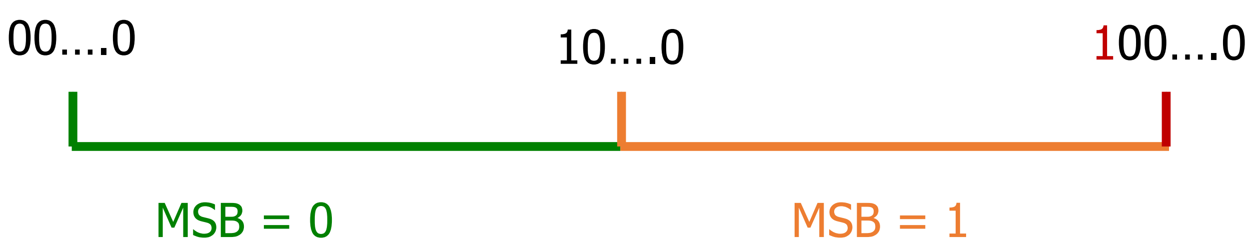
Why 2's complement numbers?

- Math: the signed/unsigned values are congruent modulo 2^n

$$-1 \equiv 2^n - 1 \bmod 2^n$$

- We can use the same method/circuit to perform addition and subtraction for both unsigned and signed numbers

- For 2's complement numbers, if the sign is 1, we look at how far it is away from the next 0 (2^n)



2's complement numbers: range of values

- When dealing with 2's complement numbers, we need to know the number of bits
  - For example, 4-bit, 8-bit, 16-bit, etc., 2's complement number.
  - We always write leading 0s for 2's complement numbers

1 0 0 0 0

0 0 0 1 0 0 0 0

Given an *n*-bit 2'c complement number:

$$b_{n-1}\, b_{n-2} \ldots b_2 b_1 b_0$$

$$-b_{n-1}\times 2^{n-1} + b_{n-2}\times 2^{n-2} + \cdots + b_2\times 2^2 + b_1\times 2^1 + b_0\times 2^0$$

3-bit $\begin{cases} \text{smallest}: -4 \\ \text{largest}: 3 \end{cases}$

What are the smallest and largest values can be represented by these bits?

$-b_{n-1} \times 2^{n-1}$

Example: Range of values

For *n*-bit two's complement numbers:

-2^{n-1} can be represented but 2^{n-1} cannot

Examples

Number of bits	Smallest	Largest
8	-2^7 –128	2^7-1 127
12	-2^{11} –2048	$2^{11}-1$ 2047
16	–32768	32767
32	–2,147,483,648	2,147,483,647

What are the bits for the smallest values?

What are the bits for the largest values?

Negate 2's complement numbers

Given the bits representing x, find out the bits for $-x$

x can be positive or negative

Steps:

1. Complement all the bits in x, i.e., $1 \rightarrow 0$ and $0 \rightarrow 1$ | flip all bits
2. Add 1 to the complemented bits | +1

Explanation:

The two steps do: $\underline{0 - x} = (-1 + 1) - x = (-1 - x) + 1$

Bit pattern of -1 is 111…111

"Subtract x from -1" is the same as "flip the bits in x"

Example: negate 2's complement numbers

From +2 to –2	From –2 to + 2

`0000 0010   where we start`
`1111 1101   flip bits`
`        1`
`1111 1110   add 1`
	`1111 1110   where we start`
	`0000 0001   flip bits`
	`        1`
	`0000 0010   add 1`

0000 0010

① flip bits:

1111 1101

② +1: $\underline{1111\ 1110}$ $(-2)_{10}$

1111 1110

① flip: 0000 0001

② +1: 0000 0010

Question

- What is the value of the following 2's complement numbers?

① using the $-b_{n-1} \times 2^{n-1} + \cdots$ $= -128 + 64 + 32 + 16 + 2 = -14$

② using the negate

a. flip: 0000 1101

+1: 0000 1110 = 14

-14

8 bits

$-2^7\ 2^6\ 2^5\ 2^4 \qquad 2^1$

```
1111 0010
1110 1010
```

-22

12 bits

```
1111 1111 0010
1111 1110 1010
```

-14

-22

Addition and subtraction of 2's complement numbers

- Addition: Same methods as (unsigned) binary numbers

- Subtraction:

$$a - b = a + (-b) = a + (\sim b) + 1$$

Flip bits in b

We just need an adder!

Exercises

signed: −1
unsigned: 15

Find the sum and differences of 1111_2 and 0100_2.

Keep the lower 4 bits of the results.

$1111 + 0100$ 3

$1111 - 0100$ −5

Sign Extension alignment

- Representing a 2's complement number with more bits
  - And preserve the value!

unsigned: pad "0"

signed: { sign = 0, pad "0" ; sign = 1, pad "1" }

- Replicate the sign bit to the left
  - Compared with unsigned values where we just extend with 0s

Examples: 8-bit to 16-bit

0000 0010 => 0000 0000 0000 0010 (same for signed and unsigned)

1111 1110 => 1111 1111 1111 1110 (sign extension for signed)

1111 1110 => 0000 0000 1111 1110 (0 extension for unsigned)

Sign extension or 0 extension? Depends on how we interpret bits

Question

- Which of the following signed number can be saved in a byte and still has the correct value?

↳ 8-bit

1111 (−1)

1111 1111 (−1)

A. 0b1011 1000 0001

B. 0b1111 1000 0001

What if they are unsigned? None

Hexadecimal

- The radix is 16 and there are 16 digits

Hex digits	0 – 9	a	b	c	d	e	f
Decimal value	0 – 9	10	11	12	13	14	15

  - Use 0x or a subscript of 16 to indicate hexadecimal numbers

    0xABCD or $ABCD_{16}$

$(XXXXXXXXX)_{16}$ → 0 ~ f

$\updownarrow$

$(XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX)_2$ → 0 ~ 1

Why hexadecimal?

- More compact for representing bits
  - Hexadecimal representation is shorter than binary representation
  - Easier for human to read/write/compute
  - Easy to convert between hexadecimal digits and bits

Conversion between hexadecimal and binary

- Hexadecimal is more compact to represent bits
  - Each hex digit represents 4 bits
  - 8 hex digits for 32 bits, and 16 for 64 bits

Mapping between hex digits and bits

0	0b0000	4	0b0100	8	0b1000	C (12)	0b1100
1	0b0001	5	0b0101	9	0b1001	D (13)	0b1101
2	0b0010	6	0b0110	A (10)	0b1010	E (14)	0b1110
3	0b0011	7	0b0111	B (11)	0b1011	F (15)	0b1111

Example: to binary: convert each hex digit to 4 bits

$ECA8\ 6420_{16}$

$= 1110\ 1100\ 1010\ 1000\ 0110\ 0100\ 0010\ 0000_2$

E C A 8 6 4 2 0

Example

Sometimes the number of bits is not a multiple of 4

bits to hex: 0 extended to a multiple of 4

hex to bits: only keep the bits at the right end

- This is part of the green card

7 bits	3 bits		hex
OPCODE	FUNCT3	FUNCT7 OR IMM	HEXADECIMAL
0000011	000		03/0
0000011	001		03/1
0000011	010		03/2

0X03

hex

Question

- What is the hexadecimal representation of the following bits?
  - Note that we do not care how the bits are interpreted
  - They could be unsigned or signed

1010 1001 1010

A 9 A

1100 0111 0110

C 7 6

Addition of hexadecimal numbers

	7	5	A	9
+	6	F	B	3
0	1	1	0	
	E	5	5	C

10

11

21
-16

21 = 16 + 5

We (humans) convert hex digits to decimal and then convert results back

9 + 3 = 12 = 0xC

0xA + 0xB + 0
= 10 + 11 + 0
= 21
=16 + 5
=0x15

0x5 + 0xF + 1
= 0x5 + 0x10
= 0x15

1 + 7 + 6 = 14 = 0xE

Similarly, we can do subtraction. Try yourself.

ASCII: Representing Characters

We also use bits to represent characters!

- ASCII: a standard that use 7 bits to represent 128 characters
  - Including digits, English letters, and special characters
  - And 33 control characters

Example: 65 for 'A', 66 for 'B', 110 for '^'

- An ASCII character is stored in a byte
  - Only use 7 bits. The MSB is always 0
  - Latin-1 extends ASCII to 256 characters (using all 8 bits in a byte)

ASCII Table (partial)

ASCII values are in decimal
Control characters (0 – 31) are not shown

ASCII value	Character	ASCII value	Character	ASCII value	Character	ASCII value	Character	ASCII value	Character	ASCII value	Character
32	space	48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	'	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o	127	DEL

Representation and interpretation

- A value has different representations

  0b1010 = 0xA = 10

  binary hex Dec

  - Computers only deal with bits
  - You can write in any format. Compiler/assembler converts it to the same bits, if the representation is supported

- Bits can be interpreted in different ways
  - E.g., unsigned numbers, 2's complement numbers
  - We are going to learn a few more ways

2^{10}

0000 → 1

1111 → f

Memorize

  Powers of 2, at least to 1024

  Mapping between single hex digits and 4 bits

  Mapping between single hex digits and numbers in [0, 15]

Example

- The immediate in the following ADDI instructions are the same!
  - Character '0' is not the same as number 0

0b 0011 0000

```
addi  s1, x0, '0'
addi  s2, x0, 48
addi  s3, x0, 0x30
# s1 == s2 and s1 == s3
# lower eight bits are 0011 0000

add   s4, x0, x0
# s4 != s1
```

ASCII

Dec

hex

zero

Reg number = 0

Study the remaining slides yourself

2's-Complement (signed) numbers

- Need to know the number of bits to read 2's complement numbers

- The left-most bit (the MSB) is the sign bit

  0: for non-negative numbers. The value is the same as unsigned.

  1: for negative numbers. The value is the unsigned value – 2^n

- Some commonly seen representations:

  –1: 0b 1111 1111 … 1111

  –2: 0b 1111 1111 … 1110

  Most-negative: 0b 1000 0000 … 0000

  Most-positive: 0b 0111 1111 … 1111

Example: counting by 5 in hexadecimal

Count by 5 in hexadecimal, starting from 0.

0x0,		
0x5,	0xA,	0xF,
0x14,	0x19,	0x1E,
0x23,	0x28,	0x2D,
0x32,	0x37,	…

Do you see any patterns?

Example: radix 10 to radix 1000

We often add thousands separators when writing large numbers

Basically, we convert decimal numbers to radix 1000 numbers

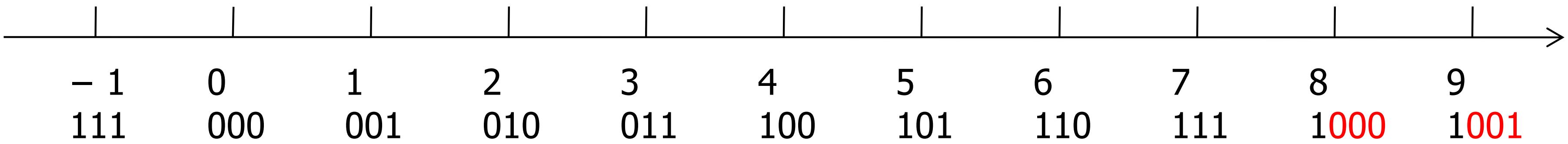
It is easy to do because $1000 = 10^3$

1 light-year = 9,460,730,472,580,800 meters

second digit

first digit

Why 2's complement number works

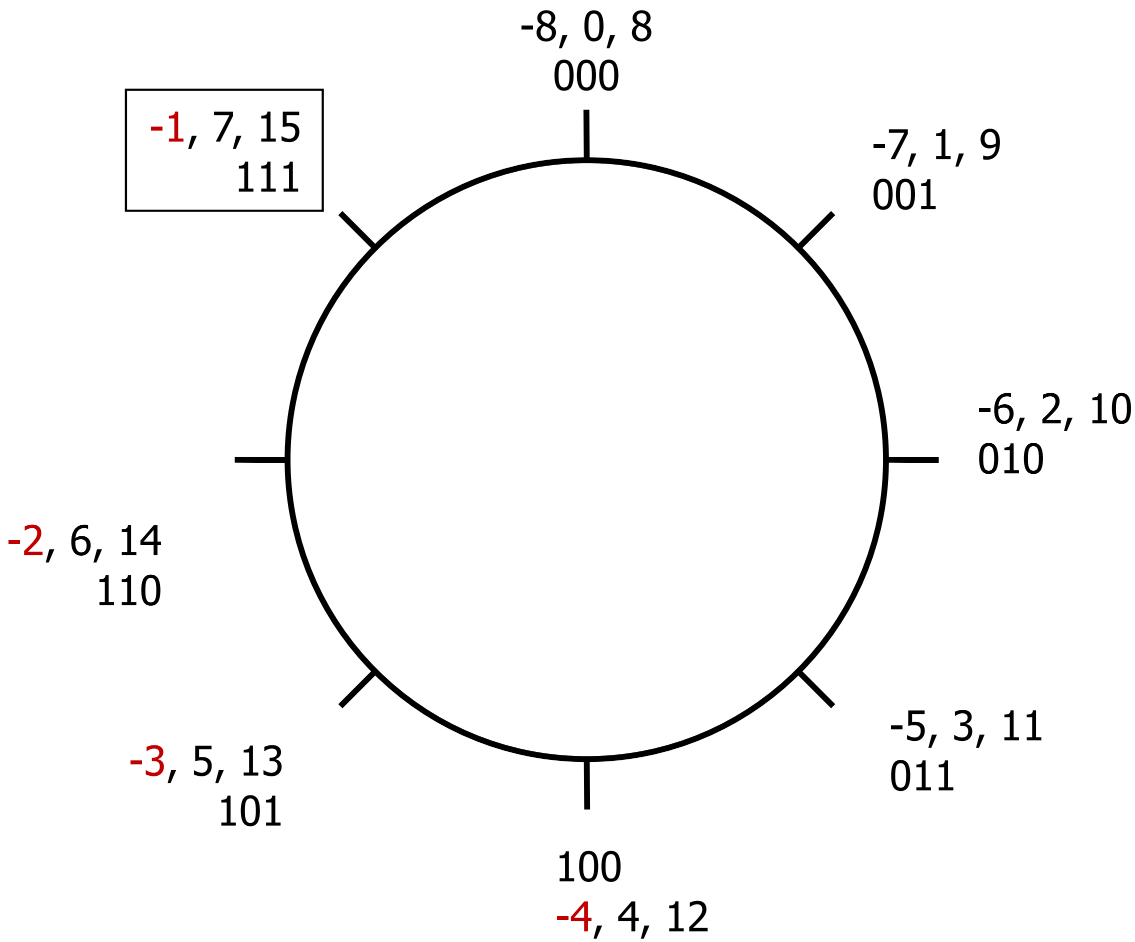


-1 = 7 mod 8

-2 = 6 mod 8

-3 = 5 mod 8

-4 = 4 mod 8



Why 2's complement number works

2's complement picks values in red

unsigned picks values in this column

Bits			
000	-8	0	8
001	-7	1	9
010	-6	2	10
011	-5	3	11
100	-4	4	12
101	-3	5	13
110	-2	6	14
111	-1	7	15

Question

- What is the value of the following 4-bit number?

  1001

We have to agree on how to interpret the bits first.

unsigned:	binary numbers (without sign)
signed:	two's complement numbers

unsigned: 9

signed: -7

There will be a problem if a program writes 9 and another program reads -7

Subtraction of binary numbers

0b1101 – 0b0111

```
                 01  11  10  01
           -      0   1   1   1
Borrows       0   1   1   0
           ----------------------
                  0   1   1   0
```

The borrow bits are from the next higher place.

Subtraction of hexadecimal numbers

	7	5	A	9
-	6	F	B	3
0	1	1	0	
	0	5	F	6

0x1A – 0xB = 16 + 10 – 11 = 15 = 0xF

0x15 – 0xF – 1 = 16 + 5 – 15 – 1 = 0x5

Misc

- Convert a binary number to decimal number
  - It means “find out the value of the unsigned binary number and represent the same value in decimal”.
- Convert a decimal number to an *n*-bit 2’s complement number
  - It means “represent the same value of the decimal number with *n*-bit 2’s complement number”.
  - The bits can be represented by hexadecimal digits, too
- Hexadecimal number as *n*-bit 2’s complement number
  - It means “ treat the bits specified by the hexadecimal digits as *n*-bit 2’s complement number.”