Lecture 2
Number Systems used in Computers

- Objectives - To understand:
  - Base of number systems: decimal, binary, octal and hexadecimal
  - Textual information stored as ASCII
  - Binary addition/subtraction, multiplication
  - Binary logical operations
  - Unsigned and signed binary number systems
  - Fixed point binary representations
  - Floating point representations
- By the end of the lecture, you should be able to:
  - Convert between numbers represented in different bases
  - Convert between fixed point and floating point numbers
  - Perform simple binary arithmetic and logical operations
  - Read and interpret hexadecimal numbers with reasonable speed

Decimal number system

- We are familiar with decimal number representation. For example:

Hundreds	*Tens*	*Ones*		*Tenths*	*Hundredths*
10^2	10^1	10^0		10^{-1}	10^{-2}
4	**6**	**2**	.	**1**	**5**

- The value of this number is calculated as:

```
4*10²  =  4*100  =    400.
6*10¹  =  6*10   =     60.
2*10⁰  =  2*1    =      2.
1*10⁻¹ =  1*.1   =      0.1
5*10⁻² =  5*.01  =  +   0.05
                      462.15
```

- In general, the relationship between a digit, its position, and the base of the system is given by:

$$DIGIT * BASE^{POSITION \#}$$

The bases of a number system

- There are no reasons why one should be restricted to using base-10 (decimal) numbers only.
- Computers and digital electronics use a binary number system where the base (or radix) is 2:

$$DIGIT * 2^{POSITION \#}$$

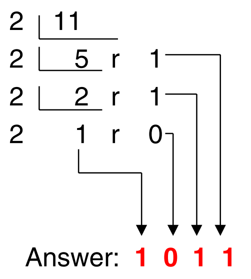
Fours	*Twos*	*Ones*		*Halves*	*Fourths*
2^2	2^1	2^0		2^{-1}	2^{-2}
1	**1**	**0**	.	**1**	**1**

- For example, the value of this binary number is:

```
1*2²  =  1*4   =     4.
1*2¹  =  1*2   =     2.
0*2⁰  =  0*1   =     0.
1*2⁻¹ =  1*.5  =     0.5
1*2⁻² =  1*.25 =  +  0.25
                     6.75
```

Converting decimal integers to binary

- Repeatedly divide the decimal number by 2 (the base of the binary system).
- Division by 2 will either give a remainder of 1 or 0.
- Collecting the remainders gives the binary answer.
- Convert 11_{10} into binary



```
2 | 11
2 |  5  r  1
2 |  2  r  1
2    1  r  0
```

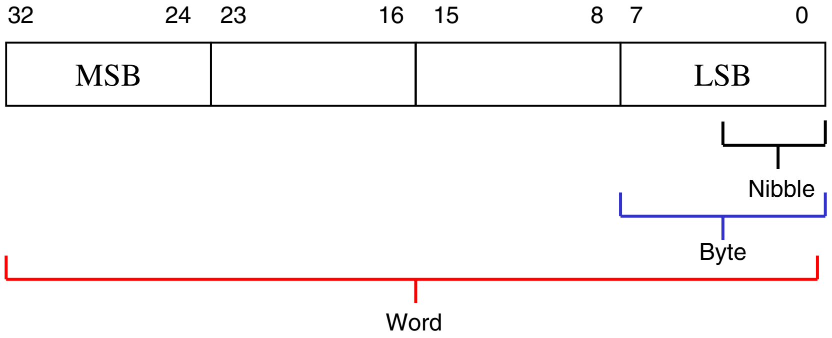
Answer: **1 0 1 1**

Octal and hexadecimal number systems

Binary	Octal	Decimal	Hexadecimal
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	A
1011	13	11	B
1100	14	12	C
1101	15	13	D
1110	16	14	E
1111	17	15	F
Base-2	*Base-8*	*Base-10*	*Base-16*

Nibbles, Bytes, Words

- Internal datapaths inside computers could be different width - for example 4-bit, 8-bit, 16-bit or 32-bit.
- For example: ARM processor uses 32-bit internal datapath
- WORD = 32-bit for ARM



Hexadecimal representation

- Convenient to divide any size of binary numbers into nibbles
- Represent each nibble as hexadecimal - much more compact
- Example:

0100 1101 0110 1011 1000 0011 0000 1111

4 D 6 B 8 3 0 F

- All microprocessor instructions are represented in hexadecimal
- Convert from decimal to hexadecimal is the same as converting to binary, except, divide by 16 instead of 2:

16 | 237

14 r 13

Answer: E D$_h$

Representing Text in ASCII

- Textual information must also be stored as binary numbers.
- Each character is represented as a 7-bit number known as ASCII codes (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange**)**
- For example, ‘A’ is represented by 41$_h$ and ‘a’ by 61$_h$

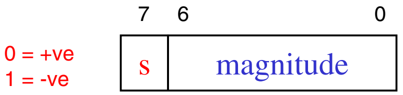
b_3 - b_0

b_6 - b_4

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

Signed numbers Basics

- So far, numbers are assumed to be unsigned (i.e. positive)
- How to represent signed numbers?
- Solution 1: **Sign-magnitude** - Use one bit to represent the **sign**, the remain bits to represent **magnitude**



$+27 = 0001\ 1011_b$
$-27 = 1001\ 1011_b$

  - ❖ Problem: need to handle sign and magnitude separately.
- Solution 2: **One's complement** - If the number is negative, invert each bits in the magnitude

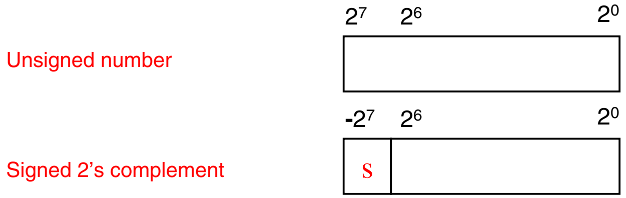
$+27 = 0001\ 1011_b$
$-27 = 1110\ 0100_b$

  - ❖ Not convenient for arithmetics - add 27 to -27 results in $1111\ 1111_b$
  - ❖ Two zero values

Two's complement

- Solution 3: **Two's complement** - represent negative numbers by taking its magnitude, invert all bits and add one:

Positive number	$+27 = 0001\ 1011_b$
Invert all bits	$1110\ 0100_b$
Add 1	$-27 = 1110\ 0101_b$



$$x = -b_{N-1}2^{N-1} + b_{N-2}2^{N-2} + \bullet\bullet\bullet + b_1 2^1 + b_0 2^0$$

Why 2's complement representation?

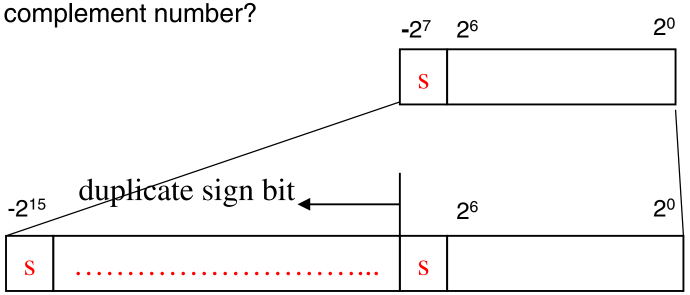
- If we represent signed numbers in 2's complement form, subtraction is the same as addition to negative (2's complemented) number.

27	$0001\ 1011_b$
- 17	$0001\ 0001_b$
+ 10	$0000\ 1010_b$
+27	$0001\ 1011_b$
+ - 17	$1110\ 1111_b$
+10	$0000\ 1010_b$

- Note that the range for 8-bit unsigned and signed numbers are different.
  - ❖ 8-bit unsigned: 0 …… +255
  - ❖ 8-bit 2's complement signed number: -128 …… +127

Sign Extension

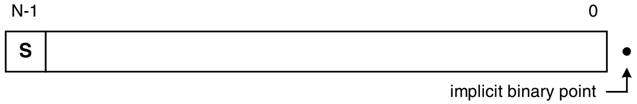
- How to translate an 8-bit 2's complement number to a 16-bit 2's complement number?



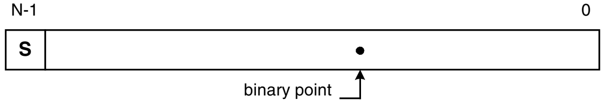
- This operation is known as **sign extension**.

Fixed point representation

- So far, we have concentrated on **integer** representation with the **fractional** part.
- There is an implicit binary point to the right:

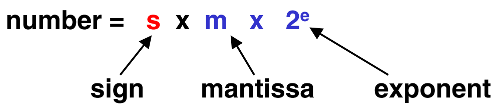


- In general, the binary point can be in the middle of the word:



Idea of floating point representation

- Although fixed point representation can cope with numbers with fractions, the range of values that can represented is still limited.
- Alternative: use the equivalent of scientific notation, but in binary:



- For example:

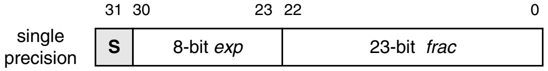
10.5 in fixed point **1010.1_b**

Move binary point to left **$1.0101_b \times 2^3$**

10.5 = 1.3125 x 8

IEEE-754 standard floating point

- 32-bit single precision floating point:



$$x = -1^s \times 2^{exp-127} \times 1.frac$$

$$1.175 \times 10^{-38} < |x| < 1.7 \times 10^{38}$$

- MSB is sign-bit (same as fixed point)
- 8-bit exponent in bias-127 integer format (i.e. add 127 to it)
- 23-bit to represent only the fractional part of the mantissa. The MSB of the mantissa is ALWAYS ‘1’, therefore it is not stored