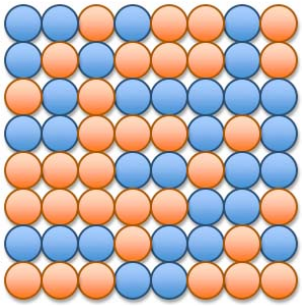
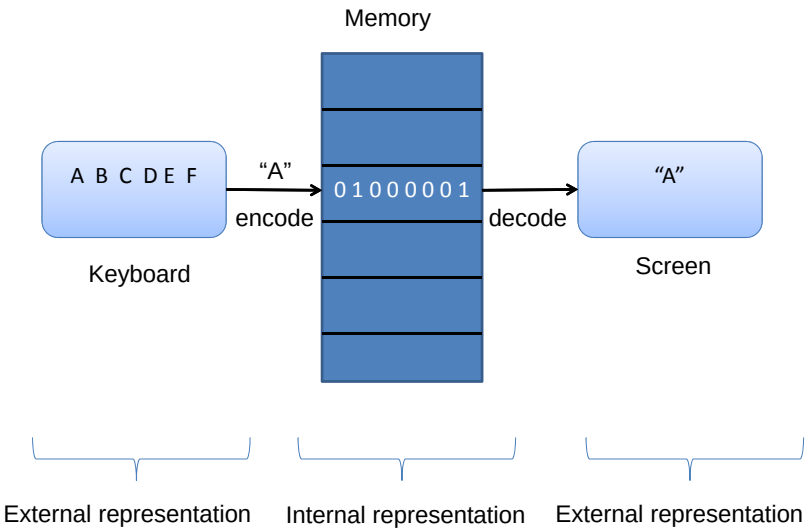


UNIT 7A

Data Representation: Numbers and Text

Representing Data



Digital Data

- We need ways of representing
  - values of data types such as integers, real numbers, text, …
  - computer instructions
- Memory is finite
  - there is a maximum size of bits that can be used to represent values of data types

3

Representing Integers

- An integer can be represented using binary.
- Example: $10 = 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0$

$$= \underset{\uparrow 2^3}{1}\ \underset{\uparrow 2^2}{0}\ \underset{\uparrow 2^1}{1}\ \underset{\uparrow 2^0}{0}\ [2]$$

- An integer can be represented using 8 bits, 16 bits, 32 bits ..
- An integer can be:
  - unsigned (always considered non-negative)
  - signed (positive or negative)

4

Unsigned Integers

- Every bit represents a power of 2.
- Example (8 bits):

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

From Decimal to Binary

Example: Converting 181 to binary

181 = 90 * 2 + 1
90 = 45 * 2 + 0
45 = 22 * 2 + 1
22 = 11 * 2 + 0
11 = 5 * 2 + 1
5 = 2 * 2 + 1
2 = 1 * 2 + 0
1 = 0 * 2 + 1

Read the answer from bottom to top:

1 0 1 1 0 1 0 1

Unsigned Integers

- With 8 bits

____	____	____	____	____	____	____	____
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

- The minimum we can represent is 0
- The maximum we can represent is 256

Unsigned Integers: Range

bits	minimum	maximum
8	0	$2^8 - 1$ (255)
16	0	$2^{16} - 1$ (65,535)
32	0	$2^{32} - 1$ (4,294,967,295)

Modular Arithmetic

- What about large numbers?
  - Is *(n+n) - n* always equal to *n + (n - n)* ?
- Modular arithmetic has pleasing properties
  - We can carry out any arithmetic operation modulo 2^p for the precision *p*

	1 0 0 1	= 9	
+	1 0 1 0	= 10	
	(1) 0 0 1 1	= 19	= 3 mod 16

overflow can be ignored

9

Signed Integers

- Every bit represents a power of 2 except the "left-most" bit, which represents the sign of the number (0 = positive, 1 = negative)
- Example for positive integer (8 bits):

0							
+	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	1	1	0	1	0	0
+		2^5	2^4		2^2		

32 + 16 + 4 = +52

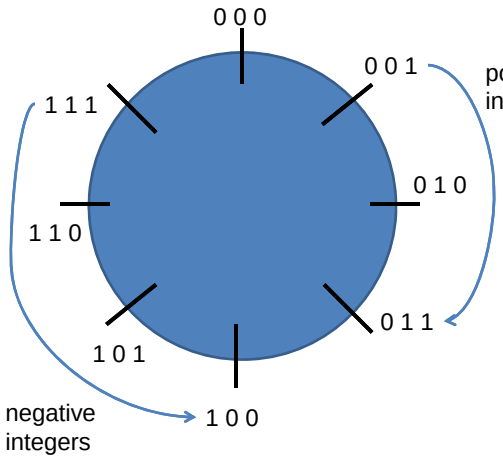
15110 Principles of Computing, Carnegie Mellon University - CORTINA

10

Representing Negative Numbers

- Are 1 0 0 0 0 0 0 and 0 0 0 0 0 0 0 0 equal? +0 and - 0 are both zero but have different bit patterns.
- We define negative numbers as additive inverse: -x is the number y such that x + y = 0.

Two's complement example



Bit pattern	Decimal value
0 0 0	0
0 0 1	+ 1
0 1 0	+ 2
0 1 1	+ 3
1 0 0	- 4
1 0 1	- 3
1 1 0	- 2
1 1 1	- 1

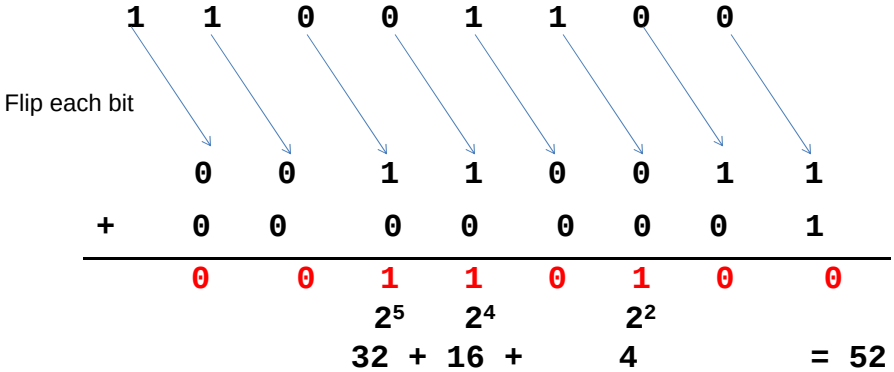
Adding + n to – n gives 0
For example: 011 + 101 = 000

Signed Integers: 2's complement

- When the leftmost bit is a 1, the integer is negative.
- To find its magnitude, we take the 2's complement of this number.
  - The 2's complement is obtained by flipping each bit of the number (from 0 to 1, or 1 to 0) and then adding 1 to that number.

Signed Integers: Negative

What value is this 8-bit signed integer?



Negative since left most bit is 1

So, **11001100 = -52**

Signed Integers: Negative

Example: How do you store -52 in 8 bits?
Start with +52:

52 = **32 + 16 + 4**

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	1	1	0	1	0	$0_{[2]}$

Flip each bit:

1 1 0 0 1 0 1 1

Add 00000001 (in base 2):

1 1 0 0 1 1 0 0 = -52

2's complement property

- When you add a number to its 2's complement (in binary), you always get 0.
  - Remember, you're using base 2 arithmetic.
- Example (using 8 bits):

```
  00110100    +52
+ 11001100    -52
----------------
  00000000      0
```

Signed Integers: Range

bits	minimum	maximum
8	-2^7	$2^7 - 1$
	(–128)	(+127)
	10000000 (binary)	01111111 (binary)
16	-2^{15}	$2^{15} - 1$
	(–32,768)	(+32,767)
32	-2^{31}	$2^{31} - 1$
	(–2,147,483,648)	(+2,147,483,647)

Text: ASCII standard

- ASCII (American Standard Code for Information Interchange)
  - 7-bit code to represent standard U.S. characters on a keyboard
  - Typically stored using 8 bits.
  - The 8^{th} bit is sometimes used for parity (more on this shortly).

ASCII table

ASCII Code Chart

	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		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
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

- 2^7 characters presented in a $2^3 * 2^4$ table.
- Values are represented in hexadecimal (base 16).
- ASCII code for “M” is 4D (hex).

ASCII Example

- The ASCII code for “M” is 4D hexadecimal.
- Conversion from base 16 to base 2:

hex	binary	hex	binary	hex	binary	hex	binary
0	0000	4	0100	8	1000	C	1100
1	0001	5	0101	9	1001	D	1101
2	0010	6	0110	A	1010	E	1110
3	0011	7	0111	B	1011	F	1111

- 4D (hex) = 0100 1101 (binary) = 77 (decimal)
  (leftmost bit can be used for parity)

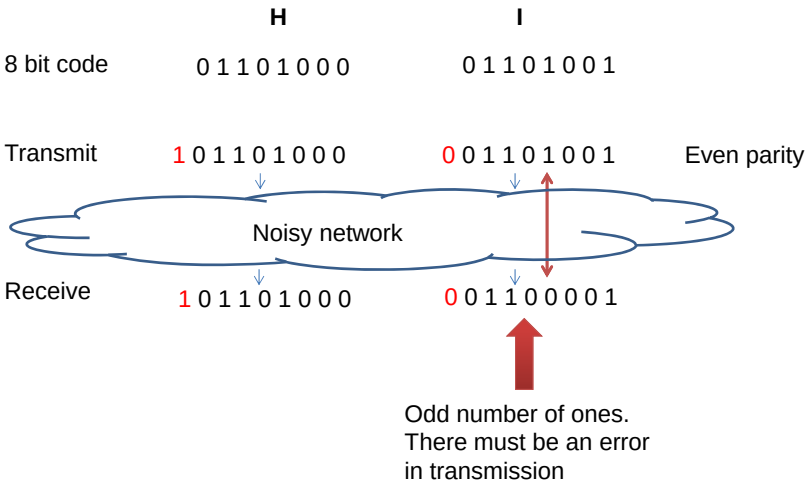
Parity

- To detect transmission errors, the 8th (leftmost) bit could be used as an error-detection bit.
- Even parity: Set the leftmost bit so that the number of 1's in the byte is even.
- Odd parity: Set the leftmost bit so that the number of 1's in the byte is odd.

Parity Example

- The character "M" is transmitted using odd parity.
- "M" in ASCII (7-bits) is 1001101.
- Using odd parity, we transmit 11001101 since this makes the number of 1's odd.
- If the receiver receives a character with an even number of 1's, the receiver knows something went wrong and requests a retransmission.
  - If two bits are flipped during transmission, we can't detect this with this simple parity scheme, however the probability of 2 or more bits in error is extremely low.

Parity Example



23

Representing Real Numbers

- Fixed point places a radix point somewhere in the middle of the digits
- Floating point uses scientific notation:

  *Significand * Base* ^{Exponent}

- Several ways to represent a single number

5.00×10^{0} ← Normalized form puts the radix after the first nonzero digit

$= 0.05 \times 10^{2}$

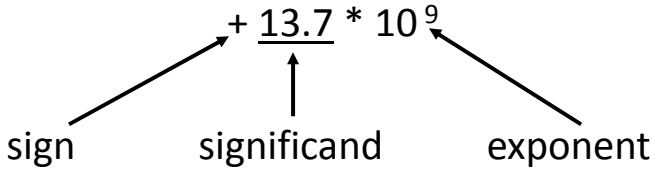
$= 5000 \times 10^{-3}$

24

Floating Point Numbers

Age of the Universe in years:

$+ \underline{13.7} * 10^{9}$



- Floating point numbers are commonly represented as a binary number with these three components.

Binary in Scientific Notation

- Decimal number 5.75 can be represented in binary as follows:

$5.75 = 5 + 0.75$

$= 101 + 0.11$ (i.e. 1/2 + 1/4)

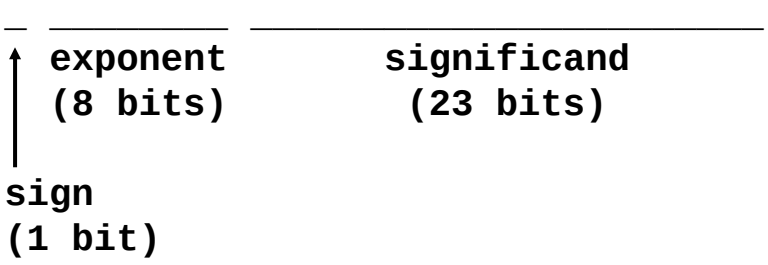
$= 101.11 * 2^{0}$

$= 10.111 * 2^{1}$

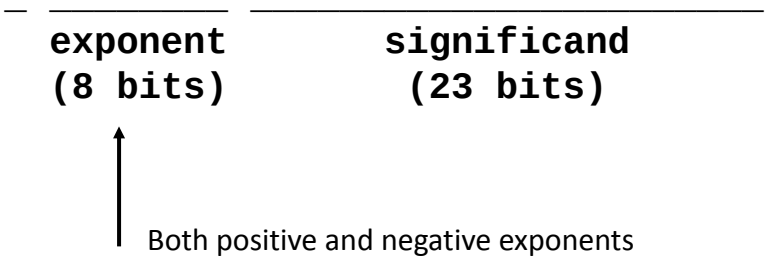
$= 1.0111 * 2^{2}$

IEEE-754 standard

- Most common encoding of floating point numbers on computers today.
- 32-bit ("single-precision") floating point:



IEEE-754 standard



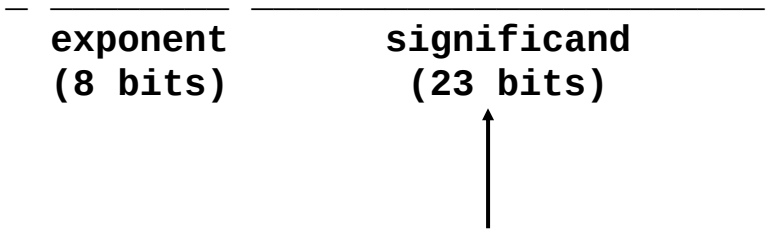
Both positive and negative exponents

Exponent range: min -126, max +127

Stores exponent as unsigned 8-bit integer, offset by 127

For example -6 is represented as 121

IEEE-754 standard



Always assumes the form 1.XXXXXXXXX in binary.

Does not store the leading 1.

29

Example: IEEE-754

- Floating point number in binary:
  - 1.0110111 X $2^{00011010}$