CSC 2400: Computer Systems

Information as Bits

•1

What kinds of data do we need to represent?

- Numbers – integers, floating point, …
- Text – characters, strings, …
- Images – pixels, colors, shapes, …
- Sound
- Instructions
- …

•2

Integers

•3

Unsigned Integers

- An n-bit unsigned integer represents 2^n values: from 0 to 2^n-1
- Example for n = 3:

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

•4

Signed Integers

- How do computers differentiate between positive and negative integers?
  - Positive integers have the most significant bit (left bit) 0
  - Negative integers have the most significant bit (left bit) 1

- Negative integer representations:
  1. Sign-Magnitude
  2. One’s Complement
  3. Two’s Complement

1. Sign-Magnitude

- Reserve the leftmost bit to represent the sign:

  0 means positive

  1 means negative

- Examples

$$\boxed{0}\ \boxed{0\ 1\ 0\ 1\ 1\ 0\ 0} \rightarrow 44$$

$$\boxed{1}\ \boxed{0\ 1\ 0\ 1\ 1\ 0\ 0} \rightarrow -44$$

Sign Magnitude

- Hard to do arithmetic this way, so it is rarely used
  - What is the result of 44 – 44?

Exercise

- Assume 8-bit sign-magnitude representation for integers
- What is the decimal value of

```
11010110
```

•7

1. Sign-Magnitude (contd.)

$\boxed{0}\ \boxed{0\ 1\ 0\ 1\ 1\ 0\ 0} \rightarrow$ **44**

$\boxed{1}\ \boxed{0\ 1\ 0\ 1\ 1\ 0\ 0} \rightarrow$ **-44**

Sign Magnitude

- For numbers represented on n bits:
  - Range of positive integers: from 0 to $(2^{n-1} - 1)$
  - Range of negative integers: from $-(2^{n-1} - 1)$ to -1

•8

Exercise

- Assume 8-bit sign-magnitude representation for integers
- What is the smallest value you can represent in this system?
- What is the largest value you can represent in this system?

•9

•2. One’s Complement

- The One’s complement of an n-bit number N is given by $(2^n – 1) – N$.
- For example, the 5-bit representation of $(12)_{10}$ is:

```
                    0 1 1 0  0
The One’s           complement  is: (2^5 – 1) – 12:
   31               1 1 1 1  1
   12              - 0 1 1 0  0
```

•10

2. One's Complement

- Leftmost bit is 0 for positive numbers

```
0 0 1 0 1 1 0 0 ➔  44
```

- To obtain the corresponding negative number (-44), flip every bit:

```
1 1 0 1 0 0 1 1 ➔  -44
```

In short, the one's complement of a positive number can be obtained by flipping 1s to 0s and 0s to 1s

•11

2. One's Complement (contd.)

- What is the result of 44 – 44?

```
0 0 1 0 1 1 0 0   ( 44)
1 1 0 1 0 0 1 1   (-44)
_______________
```

- Issue: two different representations for zero

•12

•3. Two’s Complement

- ❑The Two’s complement of an n-bit number N is given by $(2^n) - N$.
- ❑$(2^n) - N$ can be written as $(2^n - 1) - N + 1$.
- ❑$(2^n - 1) - N$ is One’s complement. Hence, Two’s complement is One’s Complement + 1

•13

3. Two’s Complement

- ❑ Leftmost bit is 0 for positive numbers

```
0 0 1 0 1 1 0 0 ➔  44
```

- ❑ To obtain the corresponding negative number -44, add 1 to the one’s complement of 44:

```
  1 1 0 1 0 0 1 1 ➔  one’s complement
+ 0 0 0 0 0 0 0 1
  ───────────────
  1 1 0 1 0 1 0 0 ➔  two’s complement
```

•14

3. Two's Complement (contd.)

- What is the result of 44 – 44?

```
0 0 1 0 1 1 0 0   ( 44)
1 1 0 1 0 1 0 0   (-44)
_______________
```

- Used by most computer systems
- For numbers represented on *n* bits:

  Range of integers: • from -2^{n-1} to $2^{n-1}-1$

•15

Two's Complement to Decimal

1. If leading bit is one, take two's complement to get a positive number
2. Convert to decimal: add powers of 2 that have "1" in corresponding bit positions
3. If original number was negative, add a minus sign

$$\boxed{\begin{aligned} X &= 01101000_{(2)} \\ &= 2^6+2^5+2^3{}_{(10)} = 64+32+8_{(10)} \\ &= 104_{(10)} \end{aligned}}$$

Assuming 8-bit two's complement numbers.

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

•18

Another Example

Assume 8-bit two's complement numbers.

$X = 11100110_{(2)}$

1. Leading bit is one, so take two's complement to get a positive number

$$\begin{aligned} -X &= 00011001 + 00000001_{(2)} \\ &= 00011010_{(2)} \end{aligned}$$

2. Convert to decimal

$-X = 2^4+2^3+2^1{}_{(10)} = 16+8+2_{(10)} = 26_{(10)}$

3. Add a minus sign

$X = -(-X) = -26_{(10)}$

•19

More Examples

$$\begin{aligned} X &= 00100111_{two} \\ &= 2^5+2^2+2^1+2^0 = 32+4+2+1 \\ &= 39_{ten} \end{aligned}$$

$$\begin{aligned} X &= 11100110_{two} \\ -X &= 00011010 \\ &= 2^4+2^3+2^1 = 16+8+2 \\ &= 26_{ten} \\ X &= -26_{ten} \end{aligned}$$

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

Assuming 8-bit 2's complement numbers.

•20

Exercise

- Assume 8-bit two's complement representation for integers
- What is the decimal value of

```
11010110
```

•21

Exercises

- Assuming 4-bit two's complement representation, what is the decimal value of $1011_{(2)}$?
- Assuming 5-bit two's complement representation, what is the decimal value of $1011_{(2)}$?
- What is -2 in 4-bit two's complement representation?
- What is -2 in 6-bit two's complement representation?

•22

Exercise

- Assume 8-bit 2's complement representation for integers
- What is the smallest value you can represent in this system?
- What is the largest value you can represent in this system?

•23

Binary Number Representation Summary

- Leftmost bit 0 indicates positive number
- Leftmost bit 1 indicates negative number
- To negate a binary value:
  - sign-magnitude: flip the sign bit
  - one's complement: take the one's complement
  - two's complement: take the two's complement
- Binary to decimal (two's complement):
  - normal conversion from binary to decimal, accounting for most significant bit having negative weight

•26

Floating-Point Numbers

- Decimal System: 11.625 analyzed as

10^1	10^0		10^{-1}	10^{-2}	10^{-3}
1	1	.	6	2	5

$$11.625 = (1 \times 10) + 1 + (6 \times 10^{-1}) + (2 \times 10^{-2}) + (5 \times 10^{-3})$$

- Binary System:

$$\begin{array}{cccccccc} 2^3 & 2^2 & 2^1 & 2^0 & . & 2^{-1} & 2^{-2} & 2^{-3} \\ 1 & 0 & 1 & 1 & . & 1 & 0 & 1_2 \end{array} = (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) + (1 \times 2^{-1}) + (0 \times 2^{-2}) + (1 \times 2^{-3}) = 11.625_{10}$$

•31

Floating-Point Numbers

Table 1.5 Binary Weights for an 8-Bit Fraction

2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}
1/2	1/4	1/8	1/16	1/32	1/64	1/128	1/256
0.5	0.25	0.125	0.0625	0.03125	0.015625	0.0078125	0.00390625

You try it:

$10010.01001_{(2)}$ = ______________________________$_{(10)}$

•32

How to Store Floating-Point Numbers?

- We have no way to store the point separating the whole part from the fractional part!
- Standard committees (IEEE) came up with a way to store floating point numbers

•33

Floating-Point Normalization

- Every floating-point binary number (**except for zero**) can be normalized by choosing the exponent so that the radix point falls to the right of the leftmost 1 bit

$37.25_{(10)} = 100101.01_{(2)} = 1.0010101 \times \mathbf{2}^{5}$

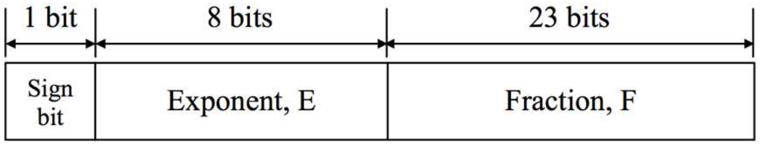
$7.625_{(10)} = 111.101_{(2)} = 1.11101 \times \mathbf{2}^{2}$

$0.3125_{(10)} = 0.0101_{(2)} = 1.\underbrace{\boxed{01}}_{\text{fraction, mantissa, significand}} \times \mathbf{2}^{\overset{\text{exponent}}{\boxed{-2}}}$

•34

IEEE Floating-Point Standard (Single Precision, 32 bits)

- Sign-Magnitude: sign bit *S*, exponent *E* and fraction ***F***



$$N = -1^S \times 1.\text{fraction} \times 2^{\text{exponent}-127},\ 1 \le \text{exponent} \le 254$$

- Special values:
  - E = 0, F = 0 represents 0.0
  - Exponent with all bits 1 (value 255) is reserved to represent ±infinity (if *F* = 0) and NaN (Not a Number, if *F* != 0)

•35

How would 15213.0 be stored?

- First, $15213_{(10)} = 11101101101101_{(2)}$
- Normalize to $1.1101101101101_{(2)} \times 2^{13}$
  - The true exponent is 13, so the biased E is

    E = 13 + 127 (Bias) = $140_{(10)}$ = $10001100_{(2)}$
  - The fraction is

    F = $\underline{1101101101101}0000000000_{(2)}$

Floating Point Representation:

Hex:	4	6	6	D	B	4	0	0
Binary:	0100	0110	0110	1101	1011	0100	0000	0000

•37

How would 15213.5 be stored?

- First, $15213.5_{(10)} = 11101101101101.1_{(2)}$
- Normalize to $1.11011011011011_{(2)} \times 2^{13}$
  - The true exponent is 13, so the biased E is

    E = 13 + 127 (Bias) = $140_{(10)}$ = $10001100_{(2)}$
  - The fraction is

    F = $\underline{11011011011011}000000000_{(2)}$

Floating Point Representation:

Hex:	4	6	6	D	B	5	0	0
Binary:	0100	0110	0110	1101	1011	0110	0000	0000

How would 23.75 be stored?

- First, $23.75_{(10)} = 10111.11_{(2)}$
- Normalize to $1.011111_{(2)} \times 2^{4}$
- The true exponent is 4, so the biased E is

  E = 4 + 127 (Bias) = $131_{(10)}$ = $10000011_{(2)}$
- The fraction is

  F = $\underline{011111}00000000000000000_{(2)}$

Floating-Point Representation:

Hex:	4	6	6	D	B	4	0	0
Binary:	0100	0001	1011	1110	0000	0000	0000	0000

How would -23.75 be stored?

- Just change the sign bit:

Hex:	4	6	6	D	B	4	0	0
Binary:	1100	0001	1011	1110	0000	0000	0000	0000

- Do not take the two's complement!

•40

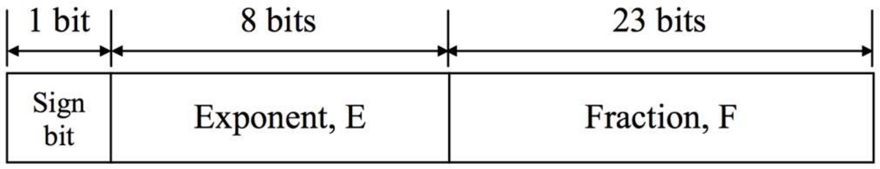
Floating-Point Numbers

IEEE Floating-Point Standard

•41

IEEE Floating-Point Standard (Single Precision, 32 bits)

- Sign-Magnitude: sign bit *S*, exponent *E* and fraction ***F***



- The binary exponent is not stored directly. Instead, *E* is the sum of the true exponent and 127. This *biased exponent* is always non-negative (seen as magnitude only).
- The fraction part assumes a normalized significand in the form 1.F (so we get the extra leading bit for free)

•42

How would 23.75 be stored?

- First, $23.75_{(10)} = 10111.11_{(2)}$
- Normalize to $1.011111_{(2)} \times 2^4$
- The true exponent is 4, so the biased E is

  $E = 4 + 127 \text{ (Bias)} = 131_{(10)} = 10000011_{(2)}$

- The fraction is

  $F = \underline{011111}00000000000000000_{(2)}$

Floating-Point Representation:

```
Hex:       4    6    6    D    B    4    0    0
Binary:  0100 0001 1011 1110 0000 0000 0000 0000
```

•43

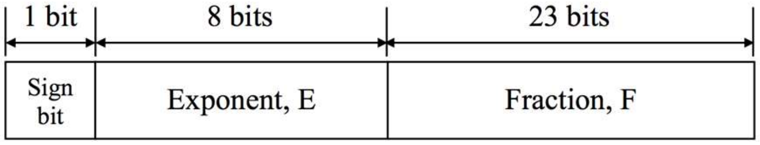
Exercise 1

- Find the IEEE representation of 40.0

•44

IEEE Floating-Point Standard (Single Precision, 32 bits)

- Sign-Magnitude: sign bit *S*, exponent *E* and fraction ***F***



$$N = -1^{S} \times 1.\text{fraction} \times 2^{\text{exponent}-127}, \; 1 \le \text{exponent} \le 254$$

- Special values:
  - E = 0, F = 0 represents 0.0
  - Exponent with all bits 1 (value 255) is reserved to represent ±infinity (if *F* = 0) and NaN (Not a Number, if *F* != 0)

•45

Reverse Your Steps:

- Convert to decimal the IEEE 32-bit floating-point number

$$\underbrace{1}_{\text{sign}}\ \underbrace{01111110}_{\text{exponent}}\ \underbrace{10000000000000000000000}_{\text{fraction}}$$

  - Sign is 1, so the number is negative
  - Exponent field is 01111110 = 126 (decimal)
  - Fraction is 100000000000… = 0.5 (decimal)

- Value = $-1.1_{(2)} \times 2^{(126-127)} = -1.1_{(2)} \times 2^{-1} = -0.11_{(2)} = -0.75_{(10)}$

•46

Exercise – Reverse Your Steps

- Convert the following 32 bit number to its decimal floating point equivalent:

```
1     01111101     01010...0
```

•47

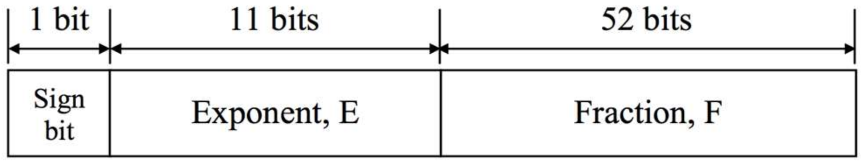
Exercise - Reverse your Steps

- Convert to decimal the IEEE 32-bit floating-point number

```
0    10000011  10011000..0
```

•48

IEEE Floating-Point Standard (Double Precision, 64 bits)



$$N = -1^{S} \times 1.\text{fraction} \times 2^{\text{exponent}-1023},\ 1 \le \text{exponent} \le 2046$$

- Exponent with all bits 1 (value 2047) is reserved to represent ±infinity (if fraction is 0) and NaN (if fraction is not 0)

•49

Approximations: How would 0.1 be stored?

2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}
1/2	1/4	1/8	1/16	1/32	1/64	1/128	1/256
0.5	0.25	0.125	0.0625	0.03125	0.015625	0.0078125	0.00390625

- First, $0.1_{(10)}$ = _ . _ $_{(2)}$
- Normalize to _ . _ $_{(2)}$ x 2^{-4}
- Biased exponent is
- Fraction is

IEEE Floating-Point Representation:

•50

Approximations: How would 0.1 be stored?

Table 1.5 Binary Weights for an 8-Bit Fraction

2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}
1/2	1/4	1/8	1/16	1/32	1/64	1/128	1/256
0.5	0.25	0.125	0.0625	0.03125	0.015625	0.0078125	0.00390625

- In general, it is dangerous to think of floating point values as being "exact”
- Fractions will probably be approximate
  - If the fraction can be exactly expressed in binary, it might still be exact, like 1/2
  - But for example, 1/10 will be an approximate value

•51

ASCII

•54

The ASCII Code

American Standard Code for Information Interchange

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
16	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
32	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
48	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
96	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Lower case: 97-122 and upper case: 65-90
E.g., 'a' is 97 and 'A' is 65 (i.e., 32 apart)

•55

char Constants

- C has char constants (sort of)
- Examples

Constant	Binary Representation (assuming ASCII)	Note
'a'	01100001	letter
'0'	00110000	digit
'\x61'	01100001	hexadecimal form

Use **single** quotes for **char** constant
Use **double** quotes for **string** constant

* Technically 'a' is of type int; automatically truncated to type char when appropriate

•56

More char Constants

- Escape characters

Constant	Binary Representation (assuming ASCII)	Note
'\b'	00001000	backspace
'\f'	00001100	form feed
'\n'	00001010	newline
'\r'	00001101	carriage return
'\t'	00001001	horizontal tab
'\v'	00001011	vertical tab
'\\'	01011100	backslash
'\''	00100111	single quote
'\"'	00100010	double quote
'\0'	00000000	null

Used often

•57

Interesting Properties of ASCII Code

- What is relationship between a decimal digit ('0', '1', …) and its ASCII code?
- What is the difference between an upper-case letter ('A', 'B', …) and its lower-case equivalent ('a', 'b', …)?
- Given two ASCII characters, how do we tell which comes first in alphabetical order?
- Are 128 characters enough? (http://www.unicode.org/)

•58

What did we learn?

- Computer represents everything in binary
  - Integers, floating-point numbers, characters, …
  - Pixels, sounds, colors, etc.

•60