2's Compliment, Floats

Introduction to Computer Systems, Fall 2022

Instructor: Travis McGaha

TAs:

Ali Krema	Andrew Rigas	Anisha Bhatia
Audrey Yang	Craig Lee	Daniel Duan
David LuoZhang	Eddy Yang	Ernest Ng
Heyi Liu	Janavi Chadha	Jason Hom
Katherine Wang	Kyrie Dowling	Mohamed Abaker
Noam Elul	Patricia Agnes	Patrick Kehinde Jr.
Ria Sharma	Sarah Luthra	Sofia Mouchtaris

How was your three day weekend?

Powered by Poll Everywhere

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Logistics pt. 1

- Pre-Semester Survey: Due Friday 9/9 @ 11:59 pm
  - No late extensions for this
  - Graded on completion

- Check in 00: Due Monday 9/12 @ 4:59 pm
  - Check-ins are due before Monday lectures
  - Make sure you are caught up with material relevant for new topics
  - Unlimited attempts, public "tests"

Logistics pt. 2

- HW00 Binary Quiz: Due Next Friday 9/16 @ 11:59 pm
  - Quiz On Canvas
  - Opens tonight at midnight
  - Should have everything you need after this lecture (some topics will be covered more in depth in Monday's lecture tho)

- PollEverywhere Registration
  - Will start counting participation next lecture
  - Will leave polls open after this lecture so that people can "test" their registration.

- Some OH posted on the course website
- Recitation information coming soon

Lecture Outline

- **Binary Review**
  - **What is binary**
  - **Encodings**
- Length Constraints
- 2's Compliment & Integer Operations
- Floats

Lecture 1 Take-aways

- A Bit is a Binary "Digit"
  - Can contain the value of either a 0 or a 1

- Bits are the "atom" of data for computers

- We can represent anything in binary by using different encodings!
  - Numbers, colors, characters, emojis, code, etc..
  - We also saw how we can do some of these conversions ourselves

The Meaning of Bits

- *A sequence of bits can have many meanings!*

- Consider the hex sequence 0x4E6F21
  - Common interpretations include:
  - The decimal number 5140257
  - The characters "No!"
  - The background color of this slide
  - The real number 7.203034×10^{-39}

- A series of bits can also be code!

- It is up to the program/programmer to decide how to ***interpret*** the sequence of bits

Bits used to encode numbers

- Bits can be used to represent a number in base 2 format
  - Each "bit" can represent 2 different values (1 or 0)
  - Each "bit" is weighted by its position

- Example: 101

$$(1 * 2^2) \quad + \quad (0 * 2^1) \quad + \quad (1 * 2^0)$$

$$4 \quad + \quad 0 \quad + \quad 1$$

$$5$$

To note that a value is in base 2, a prefix '**`0b`**' is often used
Example: **`0b101`**

Poll Everywhere

pollev.com/tqm

- What integer value does 0b00101010 represent?

A. 84

B. 48

C. 42

D. 38

E. I'm not sure

Poll Everywhere

pollev.com/tqm

- What integer value does 0b00101010 represent

A. 84

B. 48

C. 42

D. 38

E. I'm not sure

0b00101010

$$\ldots (1 * 2^5) + (0 * 2^4) + (1 * 2^3) + (0 * 2^2) + (1 * 2^1) + (0 * 2^0)$$

$$0 + 0 + 32 + 0 + 8 + 0 + 2 + 0$$

$$32 + 8 + 2$$

$$42$$

Decimal to Binary Conversion: Powers of 2

- Algorithm:
  - Subtract the largest power of two <= number
  - Put a one in the corresponding bit position
  - Repeat until number is 0

- Example: 104
  - 104 - 64 = 40 — 64 is 2^6, so bit 6 is a '1'
  - 40 - 32 = 8 — 32 is 2^5, so bit 5 is a '1'
  - 8 – 8 = 0 — 8 is 2^3, so bit 3 is a '1'
  - 104 = 0b1101000

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

Decimal to Binary Conversion: Division

- Algorithm:
  - Divide by two – remainder will be the next smallest bit
  - Keep dividing until answer is 0

- Example: 104
  - 104 / 2 = 52 r 0 bit 0 is 0
  - 52 / 2 = 26 r 0 bit 1 is 0
  - 26 / 2 = 13 r 0 bit 2 is 0
  - 13 / 2 = 6 r 1 bit 3 is 1
  - 6 / 2 = 3 r 0 bit 4 is 0
  - 3 / 2 = 1 r 1 bit 5 is 1
  - 1 / 2 = 0 r 1 bit 6 is 1
  - 104 = 0b1101000

Hexadecimal

- Base 16 representation of numbers
- Allows us to represent binary with fewer "digits"

- Prefixes to identify the base
  - 0b11110011 == 0xF3
    ^ **b**inary ^ he**x**

- Conversion examples
  - 0b010 -> 0b0010 -> 0x2
  - 0x1 -> 0b0001

Decimal	Binary	Hex
0	0000	0x0
1	0001	0x1
2	0010	0x2
3	0011	0x3
4	0100	0x4
5	0101	0x5
6	0110	0x6
7	0111	0x7
8	1000	0x8
9	1001	0x9
10	1010	0xA
11	1011	0xB
12	1100	0xC
13	1101	0xD
14	1110	0xE
15	1111	0xF

Bits encoded to represent Characters

- We can encode binary values to represent characters

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Any questions on the content introduced in last lecture?

Powered by Poll Everywhere

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Lecture Outline

- Binary Review
  - What is binary
  - Encodings
- **Length Constraints**
- **2’s Compliment & Integer Operations**
- Floats

Aside: Length Terminology

- Bit:
  - a binary "digit", either a 1 or a 0
- Byte:
  - 8 bits (two hexadecimal digits)
  - E.g., 0b11110111 or 0xF7
- Nibble:
  - 4 bits (one hexadecimal digit)
  - E.g., 0b1011 or 0xB

Data Lengths

- Computers are physical machines
  - there is a limit to how many bits/bytes we can store

- In C:
  - `char`'s are usually 1 byte (8 bits)
    - 1 byte = 8 bits → 2^8 different values
    - $2^8 = 256$
  - `int`'s are usually 4 bytes (32 bits)
    - 4 bytes = 32 bits → 2^{32} different values
    - $2^{32} = 4{,}294{,}967{,}296$

- N bits represents 2^N possible values

Aside: Bit Significance

- Most Significant Bit (MSB):
  - If we treat the bits as an integer, the bit that *most* affects the magnitude of the binary integer
  - (The left most bit)

- Least Significant Bit (LSB):
  - If we treat the bits as an integer, the bit that *least* affects the magnitude of the binary integer
  - (The right most bit)

- Example with 4 bits:

MSB LSB

`0b0101`

Signed Numbers?

- With our current understanding of number encoding, a 4-byte **`int`** can contain any value between 0 and $2^{32} - 1$

- How do we store negative values?
  - Common initial Guess: have an additional bit dedicated for the "sign", 0 means positive, 1 is negative.
    - This leads to the existence of '0' and '-0'
    - leads to awkwardness with how arithmetic is done
  - Instead, we use Two's compliment!

2's Compliment

- Except for the Most Significant Bit (MSB), it is the same as unsigned.
  - MSB is equal to its normal value in unsigned, but negated

- Consider the 4-bit number: **1011**

- Unsigned:

1011

$(1*2^3)+(0*2^2)+(1*2^1)+(1*2^0)$

$2^3+2^1+2^0$

$8+2+1$

11

Signed 2C:

1011

$(\underline{-1}*2^3)+(0*2^2)+(1*2^1)+(1*2^0)$

$(\underline{-1}*2^3)+2^1+2^0$

$-8+2+1$

-5

Poll Everywhere

pollev.com/tqm

- What 2C integer value does 0b1110 represent?
  - Assuming an integer is 4 bits in this scenario

A. -1

B. -2

C. -3

D. 0

E. I'm not sure

Poll Everywhere

pollev.com/tqm

- What 2C integer value does 0b1110 represent?
  - Assuming an integer is 4 bits in this scenario

A. -1

B. -2

C. -3

D. 0

E. I'm not sure

1110

$(-1*2^3)+(1*2^2)+(1*2^1)+(0*2^0)$

$(-1*2^3)+2^2+2^1$

$-8+4+2$

-2

Binary Addition

- Binary Addition works just like base-10
  - Add from right to left, propagating carry
  - Turns out, this works for both unsigned and 2C numbers
    (4-bit integers in this example)

	Unsigned values	2C values
1010	(10)	(-6)
+ 0011	(3)	(3)
1101	(13)	(-3)

Binary Addition: Overflow

- Real Integers are infinite
- `int`s have finite width, limited by hardware
- **Overflow**: when an operation's result is too large to fit in the type's range
  - Not always "problematic"

- Example with two 4-bit integers:

	Unsigned values	2C values
1111	(15)	(-1)
+ 0001	(1)	(1)
10000	(0)	(0)
This 1 gets cut off!	Problematic with unsigned ☹	Correct result with 2C ☺

Note: 2c Overflow can still be problematic

Poll Everywhere

pollev.com/tqm

- Is there problematic overflow if we add the following 4-bit 2C numbers? 0b1110 + 0b1001

A. Yes, there is problematic overflow

B. No, there is not problematic overflow

C. I'm not sure

Poll Everywhere

pollev.com/tqm

- Is there problematic overflow if we add the following 4-bit 2C numbers? 0b1110 + 0b1001

A. Yes, there is problematic overflow

B. No, there is not problematic overflow

C. I'm not sure

		2C values
	1110	(-2)
+	1001	(-7)
	10111	(7) ?

Generally speaking:
- Addition can only overflow 1 bit
- Unsigned: Any "extra bit" is problematic
- With 2C,
  if the MSB of the two inputs are the same,
  but MSB of output is different,
  overflow is problematic

2C Negation

- If we have a 2C bit pattern, we can negate the number by flipping each bit and then adding 1

- Example with 4-bit 2c numbers:

```
       1001  (-7)          0101   (5)          0000   (0)
flip                flip                flip
       0110                1010                1111
 +1                  +1                  +1
       0111  (7)           1011  (-5)         10000   (0)
```

Binary Subtraction

- To perform subtraction of two 2C numbers (X – Y), you can just negate Y and then add
  - (X – Y) = X + (-Y)

```
   1011   (-5)              1011      (-5)
-  1010   (-6)   ---->  +   0110      (6)
  -----                    -----
                           10001      (1)
```

Decimal -> 2c

- How do we convert a decimal number to a 2's Compliment (2C) number?

- Consider the number: 3
  - Positive number, so do the same thing we did for binary numbers previously

- Consider the number: -3
  - Find the bit pattern for +3 and then negate the bit pattern

Lecture Outline

- Binary Review
  - What is binary
  - Encodings
- Length Constraints
- 2's Compliment & Integer Operations
- **Floats**

Non-whole numbers

- We can now represent numbers that are negative or positive, how can we represent numbers that aren’t whole? (e.g 240.25)

Fixed Point Notation

- What if we stuck an implicit "binary point" into our integer representation
  - "Binary point" is analogous to a "decimal point"
- 2C addition and subtraction still work

Problem:
How do we represent values like
6.626×10^{-34}?
Fixed point would require 110 bits

$2^{-1} = 0.5$

$2^{-2} = 0.25$

$2^{-3} = 0.125$

```
  00101000.101 (40.625)
+ 11111110.110 (-1.25)
  ------------
  00100111.011 (39.375)
```

Aside: Scientific Notation

- In Decimal: $-2.5 * 10^1$ = -25
  - Sign: whether we are negative or positive
  - Ones place: Always starts with a non-zero digit
    - (unless overall expression is 0)
  - Mantissa: Everything after the decimal point
  - Exponent: We are in base 10, so we raise 10 to this value

Binary Scientific Notation

- Scientific notation in Binary: $-1.1001 * 2^4$
  - Sign: whether we are negative or positive
  - Ones place: Always starts with a non-zero 'bit'
    - (unless overall expression is 0)

A non-zero bit
must be **1**!
This 1 can be implicit

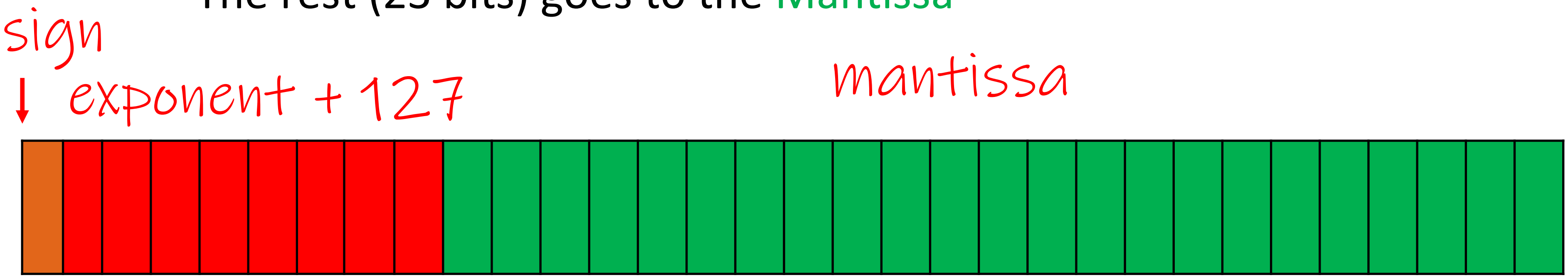
  - Mantissa: Everything after the binary point
  - Exponent: We are in base 2, so we raise 2 to this value

- We can represent a scientific notation binary number with only the Sign, Mantissa, and Exponent

IEEE Floating Point Notation

- We can represent a scientific notation binary number with only the Sign, Mantissa, and Exponent

- Allocate 32 bits, with
  - First bit goes to the Sign (1 for negative, 0 for non-negative)
  - The next 8 bits go to the Exponent + 127 (as an unsigned 8-bit int)
    - This means the exponent must fall between -127 and 128
  - The rest (23 bits) goes to the Mantissa



IEEE Floating Point Example

- Consider -0.75
  - Mark the sign bit then ignore it — 0.75
  - Convert number to fixed point binary — 0.11
    - Similar strategies to decimal -> unsigned int
  - Multiply by '1' — $0.11 = 0.11 * 2^0$
  - Shift the point by changing the exponent — $1.1 * 2^{-1}$
    - Shift bits to the left: decrement exponent
    - Shifting bits to the right: increment exponent
  - Add bias to the exponent then store — Exponent = -1 + bias = 126
    - "bias" for floats is 127
  - Store mantissa — $1.1 * 2^{-1}$
    - Truncate extra bits, or pad out with 0's if not enough

1 0 1 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0