Signed Binary Arithmetic

- In the "real world" of mathematics, computers must represent both positive and negative binary numbers.
- For example, even when dealing with positive arguments, mathematical operations may produce a negative result:
  - Example: 124 – 237 = –113.
- Thus needs to be a consistent method of representing negative numbers in binary computer arithmetic operations.
- There are various approaches, but they all involve using one of the digits of the binary number to represent the sign of the number.
- Two methods are the sign/magnitude representation and the one's complement method of representation.

Binary Sign Representations

- **Sign-magnitude: The left bit is the sign (0 for + numbers and 1 for – numbers).**

	All bits to right are the number magnitude

Left bit is the sign bit

Advantages to sign-magnitude:
- **Simple to implement.**
- **Useful for floating point representation.**

(Big) Disadvantage of sign-magnitude:
- **Sign bit independent of magnitude; can be both + 0 and – 0! (Makes math hard to do).**

- **One's complement: The negative number of the same magnitude as any given positive number is its one's complement.**
  - **If m = 01001100, then m complement (or $\overline{m}$) = 10110011**
  - **The most significant bit is the sign, and is 0 for + binary numbers and – for negative numbers.**
  - **Note the problem: If m = 00000000, then $\overline{m}$ = 11111111; there are two zeros in this method as well.**

Two's Complement Negative Binary Numbers

- **Due to the problems with sign/magnitude and 1's complement, another approach has become the standard for representing the sign of a fixed-point binary number in computer circuits.**
- **Consider the following definition: "The <u>two's complement</u> of a binary integer is the 1's complement of the number plus 1."**
- **Thus if *m* is the 2's complement of *n*, then:** $m = \bar{n} + 1$
- **Examples:**
  - **n = 0101 0100, then m = 1010 1011 + 1 = 1010 1100**
  - **n = 0101 1111, then m = 1010 0000 + 1 = 1010 0001**
  - **n = 0111 1111, then m = 1000 0000 + 1 = 1000 0001**
  - **n = 0000 0001, then m = 1111 1110 + 1 = 1111 1111**
- **Note that to properly represent 2's complement binary numbers, the full group of numbers showing the range of representation must be retained, because <u>the left-most bit is the sign</u> (0 = +, 1 = –).**

Two's Complement Negative Binary Numbers (2)

- **For integer 2's complement representation addition and subtraction of both + and – numbers <u>always work out correctly</u> (within the range of representation), and there is <u>only one 0</u>.**
- **As noted on the previous slide, the left-most bit is <u>always 1 for a negative number</u>, <u>always 0 for a positive number</u>.**
- **This means that an n-bit binary number in two's complement can represent a magnitude range of $\pm\, 2^{n-1} - 1$.**
- **<u>In an n-bit representation, there are no extra bits</u>! If adding 2 n-bit numbers results in n+1 bits, the <u>l</u>eft most bit is discarded! Thus:**
  - **Let n = 0000 0000. Then $m = n + 1$ = 1111 1111 + 1 = (1) 0000 0000 = 0000 0000. The 1 is discarded, since in the computer, there are no extra columns. There are only 8-bits, so the (9th-column) 1 is "thrown away."**
  - **<u>Therefore, the 2's complement of 0 is 0</u>.**

Finding Two's Complements: Examples

- **In the following, remember that for any n-bit computing system, there are no extra bit positions.**
- **To convert a negative decimal number to 2's complement binary:**
  - **Convert the decimal number to a positive binary number.**
  - **Take the 1's complement of that binary number and add 1.**
- **Converting negative numbers (still using a single 8-bit byte length):**
  - **50: 50 = 0011 0010; 1's C. = 1100 1101; 2's C. = 1100 1110.**
  - **127: 127 = 0111 1111; 1's C. = 1000 0000; 2's C. = 1000 0001.**
  - **1: 1 = 0000 0001; 1's C. = 1111 1110; 2's C. =1111 1111.**
- **But: Positive decimal numbers are converted simply to positive binary numbers as before (no 2's complement).**

Example: +67 (using method of successive div.) → 0100 0011

Two's Complement Binary to Decimal

- **Converting the "other direction" (2's complement to decimal) is also simple. Simply do the following:**
  - **Check the sign bit (left-most bit).**
  - **If the sign bit is 0 (positive number), simply convert the number directly to a positive decimal number as we learned previously.**
  - **If the sign bit is 1, the number is a 2's complement negative number. To convert this number to decimal:**
    - **Take the 2's complement of the negative binary number.**
    - **Convert the resulting + number to decimal and add a negative sign.**

Two's Complement Binary to Decimal (2)

- **Binary 2's complement-to-decimal examples, negative numbers:**

  1111 1111 → 0000 0000+1 = 0000 0001 = 1; → – 1.

  1010 0011 → 0101 1100+1 = 0101 1101 = 93; → – 93.

  1000 1111 → 0111 0000+1 = 0111 0001 = 113; → – 113.

  1000 0010 → 0111 1101+1 = 0111 1110 = 126; → – 126.

- **But for a positive binary number:**

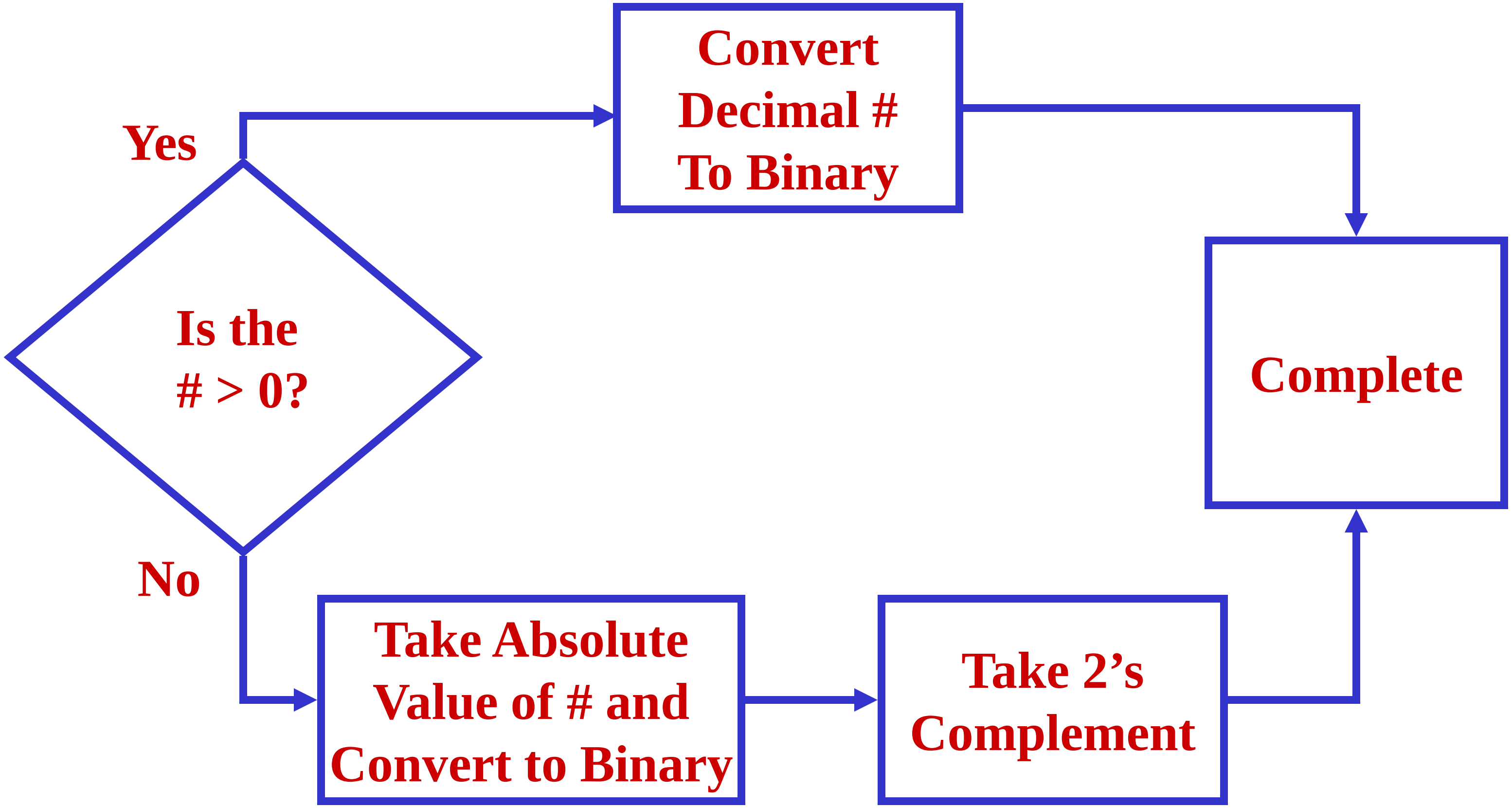
  0000 0001 → Not a negative number → 1.

  0000 1111 → Not a negative number → 15.

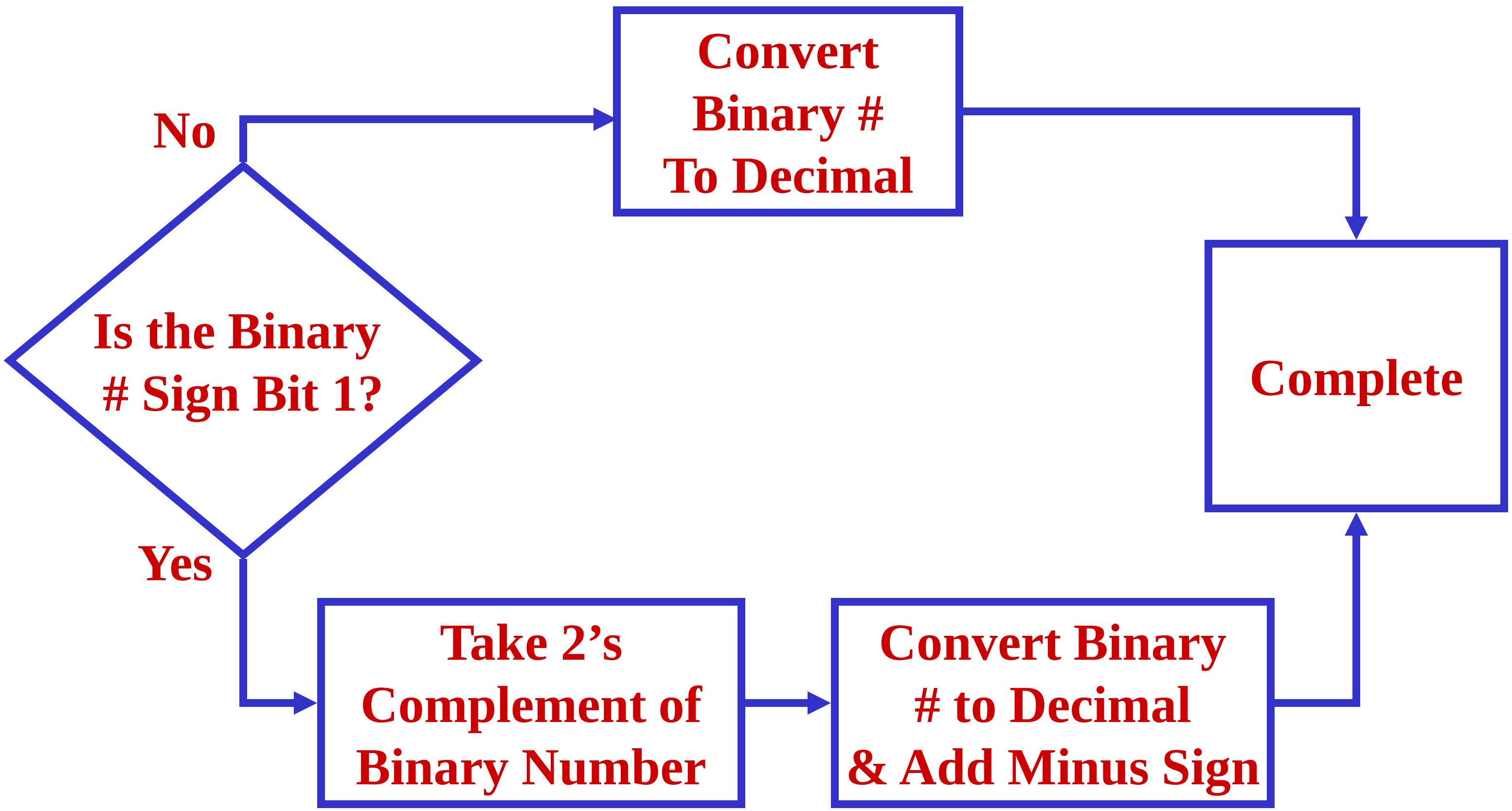
  0110 1100 → Not a negative number → 108.

  0111 1111 → Not a negative number → 127.

Decimal → 2's Complement Binary



2’s Complement Binary → Decimal



Exercise 1

- **Convert the following:**

78	**--------**
– 106	**--------**
11101011	**--------**
11000010	**--------**
01010101	**--------**

Two's Complement Binary Math

- **If we represent binary numbers in 2's complement form, simple addition and subtraction is easy.**
- **To subtract binary number b from a, simply take the 2's complement of b, and add to a. That is:**

$$a - b = a + (\text{2's comp. of } b) = a + (\bar{b} + 1) = a + \bar{b} + 1$$

- **Adding numbers is the same as decimal addition. To add a positive and negative number, simply perform the addition and the minus sign (i.e., the left-most bit in the number) will take care of itself (assuming the result is within the range of representation).**

Two's Complement Math

- **Subtract 0111 0101 from 0111 1100:**

  The 2's complement of 0111 0101 is 1000 1010 +1 = 1000 1011.

Adding:	**0111 1100**	**Check:**	**124**
"Thrown away"	**+ 1000 1011**		**–117**
	(1)0000 0111		**007**

- **Add 1100 0001 + 0110 1110:**

  Note that the 2's complement of 1100 0001 is 0011 1111, so the first number is equivalent to – 63 decimal.

Adding:	**1100 0001**	**Check:**	**– 63**
"Thrown away"	**+0110 1110**		**+110**
	(1)0010 1111		**+ 47**

Two's Complement Binary Math (2)

- **Subtract 1101 1101 from 0101 1100 (note we are <u>subtracting</u> a <u>negative number</u>):**

Adding:

	0101 1100	Check:	92	92
2's complement of 1101 1101.	+ 0010 0011		–(– 35)	+ 35
	0111 1111			+ 127

- **Add 1000 0001 + 0111 0010:**

Adding:

0111 0010	Check:	114
+1000 0001		+ (– 127)
1111 0011		– 13

Check!

Check: 2's C of 1111 0011 = 0000 1101 = 13, so the number = – 13.

"Outside the Range of Representation"

- A number is "outside the range of representation," when it cannot be represented by the bits available.
- For 8-bit, 2's complement numbers (since left-most bit is the sign bit):
  - Biggest + number to be represented = 0111 1111 (+127).
  - Biggest negative number possible is 1000 0001 (= – 127, 2's C of 0111 1111).
- Numbers outside the range of representation of n bits <u>may only be represented by adding more bits</u>.

<u>Examples</u>:

0111 0011	115
+ <u>0011 1111</u>	+ <u>63</u>
1011 0010	178
= – 78!!!	
1001 0000	– 112
<u>1110 0000</u>	– <u>32</u>
0111 0000	– 144
= + 112!!!	

Summary: 2's Complement Binary Math

- For integral mathematics in all modern computers, 2's complement arithmetic is the customary approach.
  - No ambiguous values crop up in CPU operations.
  - A binary adder can subtract with minor modifications.
  - The 2's complement binary math unit is simple.
- For 2's complement subtraction, the algorithm is very simple:
  - Take the 2's complement of the number being subtracted.
  - Add the two numbers together.
  - Note that the sign takes care of itself ("assuming the answer is within the range of representation").

Exercise 2

- **Two 2's comp. problems:**

$$\begin{array}{r} 0101\ 0101 \\ +\ \underline{1011\ 0110} \end{array}$$

$$\begin{array}{r} 1111\ 1100 \\ -\ \underline{0101\ 0001} \end{array}$$

Binary Codes

- **Computers also use binary numbers to represent non-numeric information, such as text or graphics.**
- **Binary representations of text, (letters, textual numbers, punctuation symbols, etc.) are called codes.**
- **In a binary code, the binary number is a symbol and does not represent an actual number.**
- **A code normally cannot be "operated on" in the usual fashion – mathematical, logical, etc. That is, one cannot usually add up, for example, two binary codes. It would be like attempting to add A and G!**

The ASCII Alphanumeric Code

- ASCII code represents alphanumeric data in most computers ("American Standard Code for Information Interchange").
  - Data on this transparency is coded in ASCII.
  - ASCII codes are used for virtually all printers today.
  - In the basic ASCII code that we will study, a single byte is used for each character. The least significant 7 bits represent the character. The eighth bit (the most significant bit, or MSB) may be used for error checking.
- "Super ASCII" codes can use all 8 bits (or more) for even more elaborate codes, such as other alphabets and character sets (Greek, Katakana, etc.).

ASCII Code (2)

- **There are 128 basic ASCII characters, 0-127$_{10}$, or 0-7f $_{16}$ (0000 0000 to 0111 1111 binary).**
- **Each ASCII code is unique, for example:**
  - **M = 0100 1101 = 77$_{10}$ = 4D$_{16}$.**
  - **m = 0110 1101 = 109$_{10}$ = 6D$_{16}$.**
  - **Note that the small letters are exactly 32$_{10}$ (20 hex) larger in numerical value than the capital letters.**
- **ASCII characters are stored as bytes in the computer.**
- **ASCII characters are normally represented as pairs of hex numbers (since 1 byte = 2 nibbles = 2 hex numbers).**

Another Binary Code: EBCDIC

- **EBCDIC (Extended Binary Coded Decimal Information Code) is an eight-bit character set developed by International Business Machines (IBM) and used on most IBM computers through the 1970's. It was a precursor code to ASCII.**
- **IBM PC's have used ASCII code from the first models. Most other computer makers have also used the ASCII system since it was developed in the 1960s.**
- **EBCDIC is still used in some IBM computer equipment, mainly in systems with "legacy code" (very old software developed years ago) that was written in the days of EBCDIC.**
- **EBCDIC is an 8-bit code like ASCII, but the assignment of characters is very different. A few examples on the next slide.**

EBCDIC and ASCII Hex Codes

Character	EBCDIC	ASCII
A	C1	41
B	C2	42
a	81	61
b	82	62
1	F1	31
7	F7	37
.	4B	2E
,	6B	2C
“	7F	22

Summary

- **Binary numbers are the numbers of computing. In EE 2310, students must master binary numbers.**
- **Since we live in a decimal world, it is also crucial to understand binary/decimal conversions.**
- **Hexadecimal numbers are also important, since many computer systems use hex readouts to ease the problem of interpreting 32- and 64- bit binary numbers.**
- **Binary codes are numbers that have a different meaning than their simple numerical value.**
- **ASCII is the default text code in most computer systems.**
- **The basic ASCII code set is shown at the end of these notes.**
- **Remember: "There are only 10 types of people in the world – those that understand binary and those that don't!"**

Exercise 3

- **Decode the following ASCII message:**

47-6f-2c-20-55-54-44-20-43-6f-6d-65-74-73-21

All numbers are ASCII characters in hex. Hint: We might say this if we had a football team.

Homework

- **As was discussed last lecture, <u>a good idea is to write down the two or three important things you learned today</u>. Add these to the list you made last time (you did make a list last time, right?).**
- **Write down two or three things you did not clearly understand, as was mentioned before also.**
- **After finishing the assigned reading, if you still have questions, see me during office hours.**

ASCII Codes

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	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	Neg. 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	End 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	Substitution	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	□