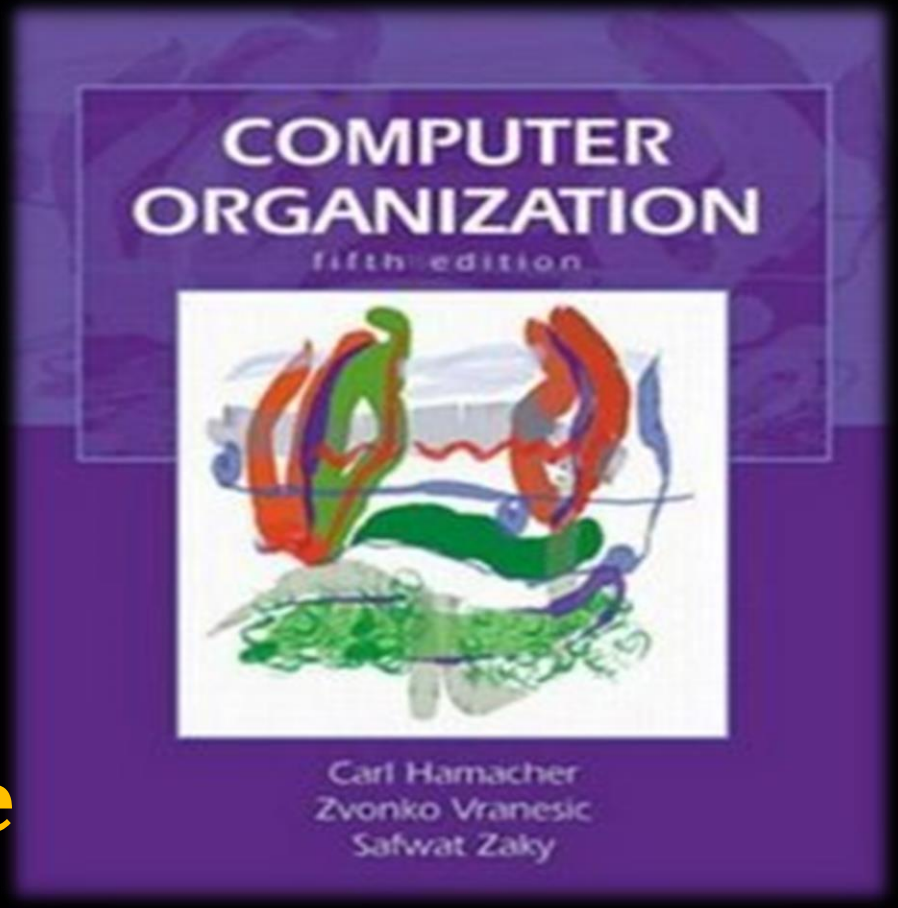
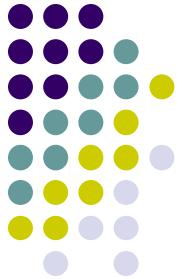
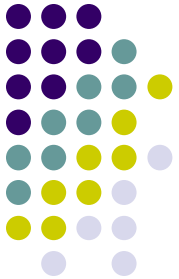
CSE 2213:
Computer Architecture



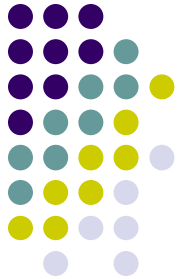


Chapter 2
Machine Instructions and Programs



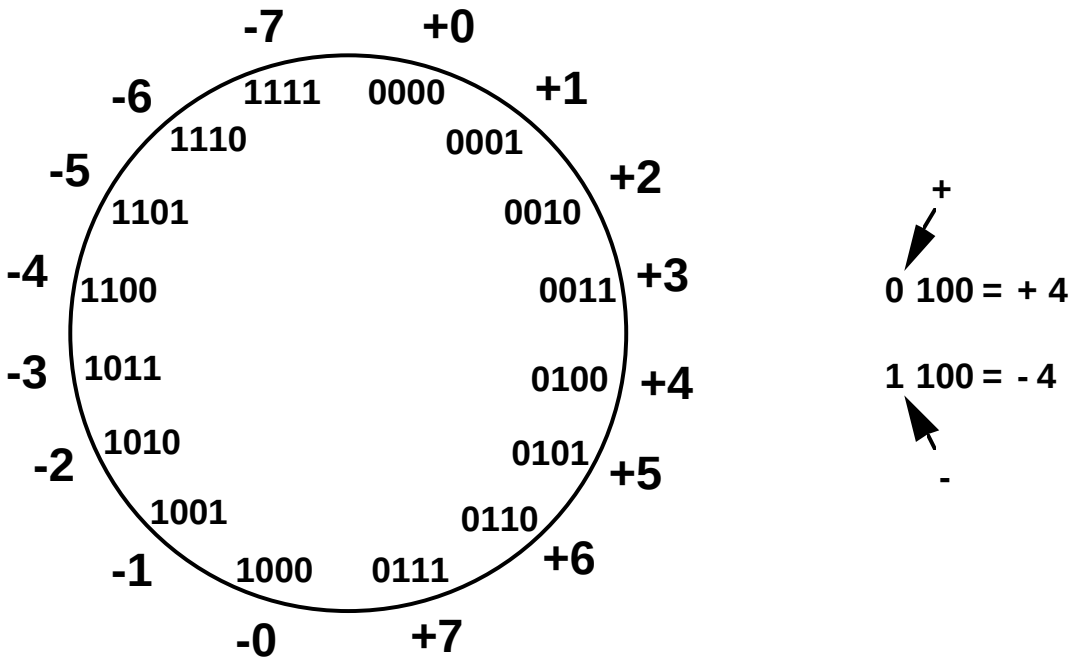
Number, Arithmetic Operations, and Characters

Signed Integer Representations



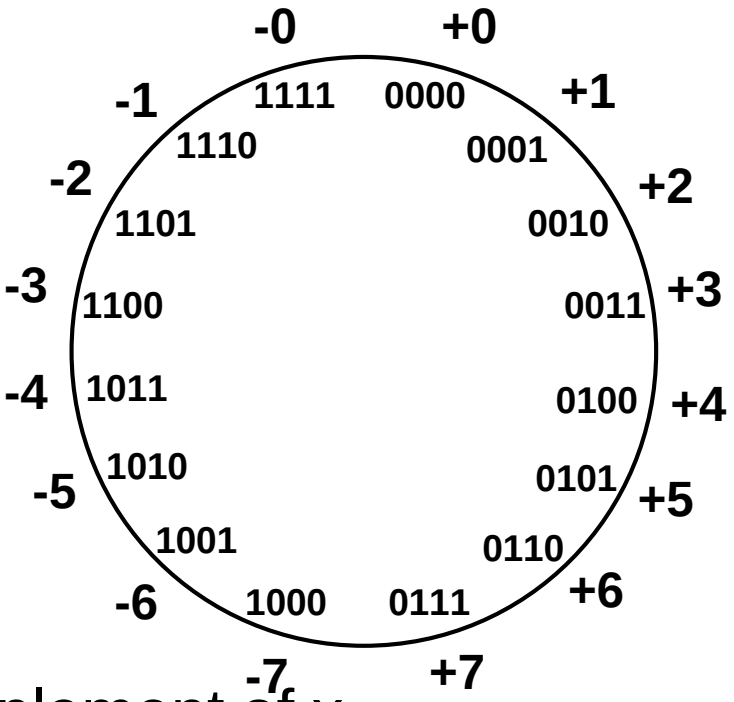
- 3 major representations:
  - Sign and magnitude
  - One’s complement
  - Two’s complement
- Assumptions for the Next Example:
  - 4-bit machine word
  - 16 different values can be represented
  - Roughly half are positive, half are negative

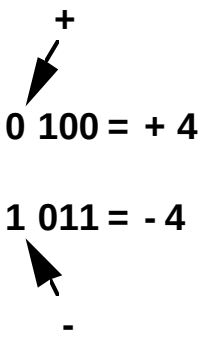
Sign and Magnitude Representation



High order bit is sign: 0 = positive (or zero), 1 = negative
Three low order bits is the magnitude: 0 (000) thru 7 (111)
Number range for n bits = +/- $(2^{n-1} - 1)$
Problems: Two representations for 0 (0000 is +0, 1000 is –0) (see the number wheel)
Some Complexities in Addition, Subtraction

One's Complement Representation

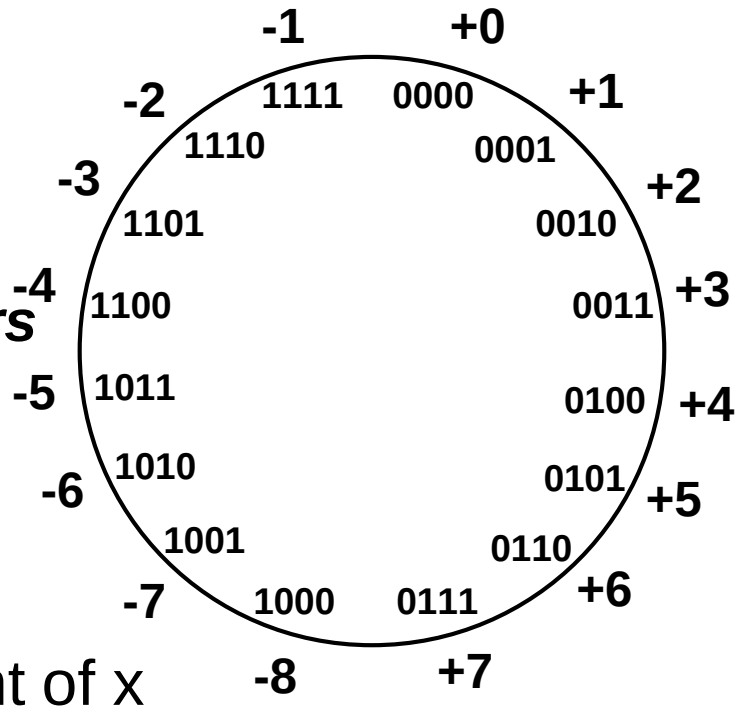


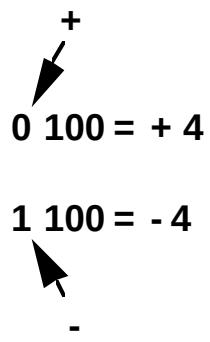


- - x = 1's complement of x
- 1's complement is invert 0 to 1 and 1 to 0
- Two representations for 0 (0000 is +0, 1111 is -0). causes some problems Some complexities in addition, subtraction
- Subtraction (X-Y) implemented by addition & 1's complement (x – y = X + 1's complement of Y = X + Y')

Two's Complement Representation

like 1's comp except negative numbers shifted one position clockwise





- - x = 2's complement of x
- 2's complement is just 1's complement + 1
- Only one representation for 0 (0000 => 1111+1 => 10000 => 0000 in 4 bits, ignore the carry out / MSB 1)
- Addition, Subtraction Very Simple
- One more negative number than positive number (Not a major problem)

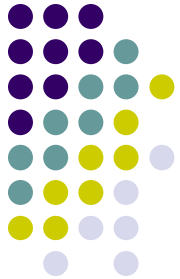
Binary, Signed-Integer Representations

Page 28

B	Values represented		
$b_3 b_2 b_1 b_0$	Sign and magnitude	1's complement	2's complement
0 1 1 1	+ 7	+ 7	+ 7
0 1 1 0	+ 6	+ 6	+ 6
0 1 0 1	+ 5	+ 5	+ 5
0 1 0 0	+ 4	+ 4	+ 4
0 0 1 1	+ 3	+ 3	+ 3
0 0 1 0	+ 2	+ 2	+ 2
0 0 0 1	+ 1	+ 1	+ 1
0 0 0 0	+ 0	+ 0	+ 0
1 0 0 0	- 0	- 7	- 8
1 0 0 1	- 1	- 6	- 7
1 0 1 0	- 2	- 5	- 6
1 0 1 1	- 3	- 4	- 5
1 1 0 0	- 4	- 3	- 4
1 1 0 1	- 5	- 2	- 3
1 1 1 0	- 6	- 1	- 2
1 1 1 1	- 7	- 0	- 1

Figure 2.1. Binary, signed-integer representations.

Addition and Subtraction – 2's Complement



If carry-in to the high order bit = carry-out then ignore carry

4	0100	-4	1100
+ 3	0011	+ (-3)	1101
7	0111	-7	11001

if carry-in differs from carry-out then overflow

4	0100	-4	1100
- 3	1101	+ 3	0011
1	10001	-1	1111

Simpler addition scheme makes two's complement the most common choice for integer number systems within digital systems

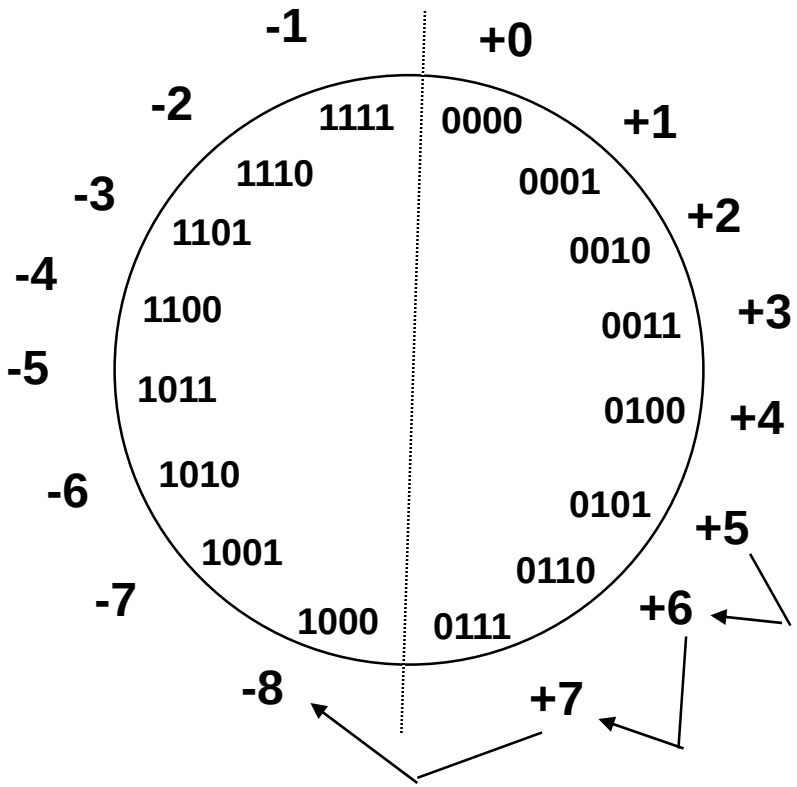
2's-Complement Add and Subtract Operations

Page 31

(a)	0 0 1 0	(+2)		(b)	0 1 0 0	(+4)	
	+ 0 0 1 1	(+3)			+ 1 0 1 0	(- 6)	
	0 1 0 1	(+5)			1 1 1 0	(- 2)	
(c)	1 0 1 1	(- 5)		(d)	0 1 1 1	(+7)	
	+ 1 1 1 0	(- 2)			+ 1 1 0 1	(- 3)	
	1 0 0 1	(- 7)			0 1 0 0	(+4)	
(e)	1 1 0 1	(- 3)	⇒		1 1 0 1		
	- 1 0 0 1	(- 7)			+ 0 1 1 1		
					0 1 0 0	(+4)	
(f)	0 0 1 0	(+2)	⇒		0 0 1 0		
	- 0 1 0 0	(+4)			+ 1 1 0 0		
					1 1 1 0	(- 2)	
(g)	0 1 1 0	(+6)	⇒		0 1 1 0		
	- 0 0 1 1	(+3)			+ 1 1 0 1		
					0 0 1 1	(+3)	
(h)	1 0 0 1	(- 7)	⇒		1 0 0 1		
	- 1 0 1 1	(- 5)			+ 0 1 0 1		
					1 1 1 0	(- 2)	
(i)	1 0 0 1	(- 7)	⇒		1 0 0 1		
	- 0 0 0 1	(+1)			+ 1 1 1 1		
					1 0 0 0	(- 8)	
(j)	0 0 1 0	(+2)	⇒		0 0 1 0		
	- 1 1 0 1	(- 3)			+ 0 0 1 1		
					0 1 0 1	(+5)	

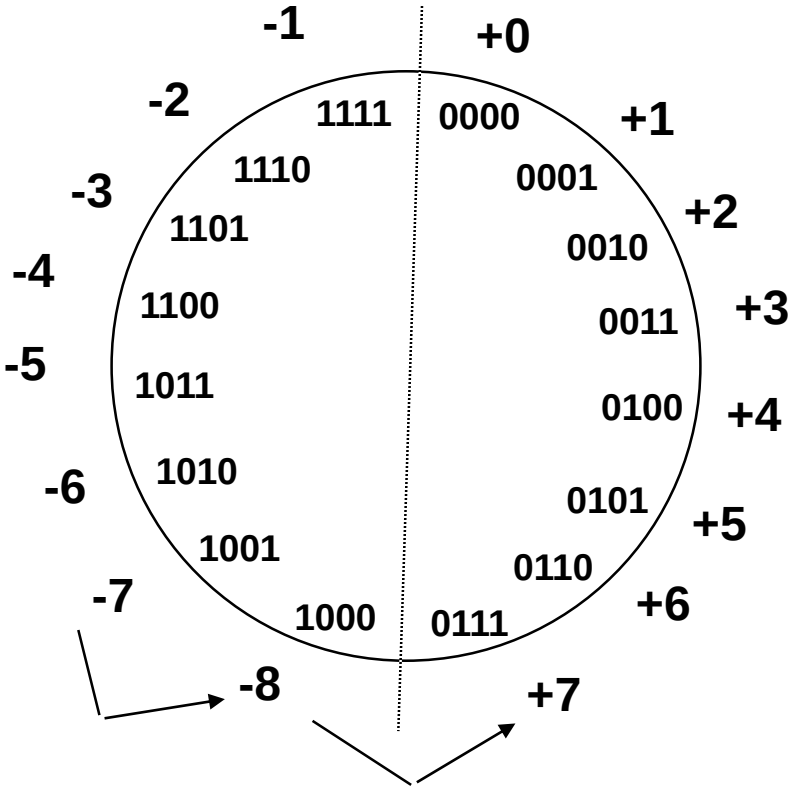
Figure 2.4. 2's-complement Add and Subtract operations.

Overflow Condition - Add two positive numbers to get a negative number or two negative numbers to get a positive number



5 + 3 = -8

0101 + 0011 = 1000

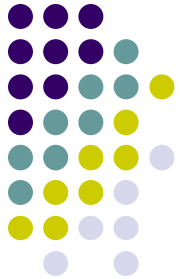


-7 - 2 = +7

1001 + 1100 = 10111 => 0111

x

Overflow Condition – Carry in to MSB ≠ Carry out from MSB



Overflow

5	0 1 1 1 0 1 0 1	
3	0 0 1 1	
-8	1 0 0 0	

-7	1 0 0 0 1 0 0 1	Overflow
-2	1 1 0 0	
7	1 0 1 1 1	

No overflow

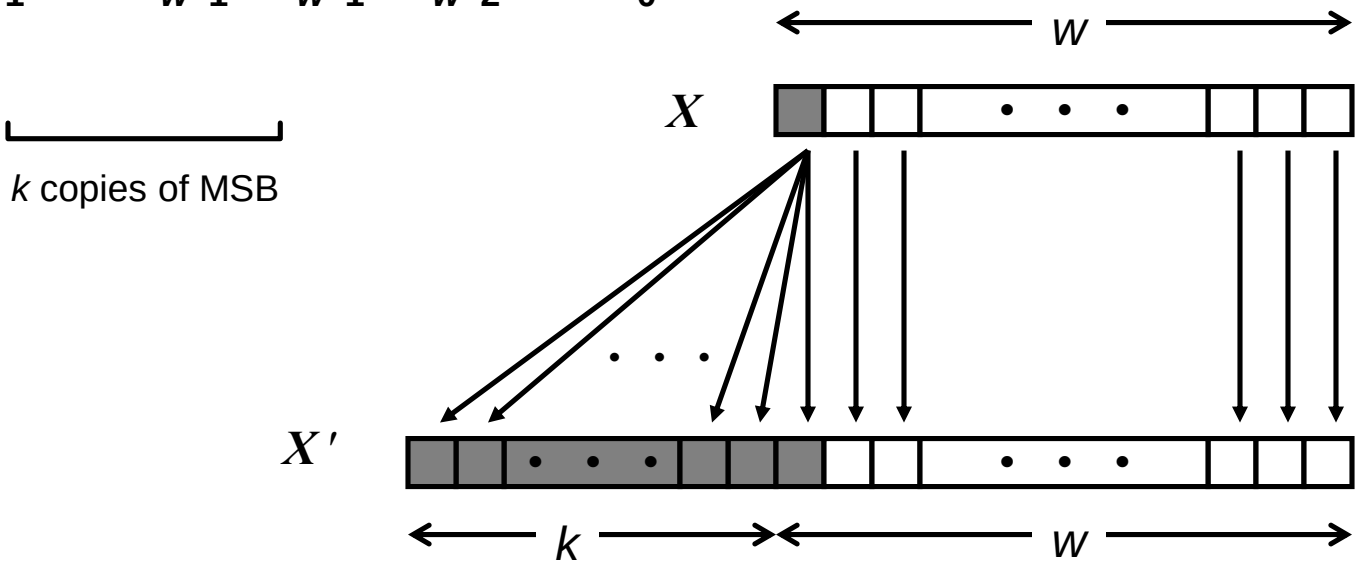
5	0 0 0 0 0 1 0 1	
2	0 0 1 0	
7	0 1 1 1	

-3	1 1 1 1 1 1 0 1	No overflow
-5	1 0 1 1	
-8	1 1 0 0 0	

Two Ways to detect Overflow: (1) when carry-in to the MSB (most significant bit) does not equal carry out from MSB
(2) Add two positive numbers to get a negative number
or, Add two negative numbers to get a positive number

Sign Extension

- Task:
  - Given *w*-bit signed integer *x*
  - Convert it to *w*+*k*-bit integer with same value
- Rule:
  - Make *k* copies of sign bit:
  - $X' = x_{w-1}, \ldots, x_{w-1}, x_{w-1}, x_{w-2}, \ldots, x_0$

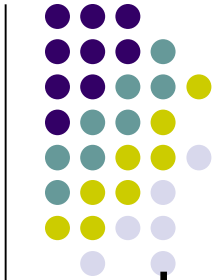


Sign Extension Example

```
short int x =  15213;
int       ix = (int) x;
short int y = -15213;
int       iy = (int) y;
```

	Decimal	Hex	Binary
x	15213	3B 6D	00111011 01101101
ix	15213	00 00 C4 92	00000000 00000000 00111011 01101101
y	-15213	C4 93	11000100 10010011
iy	-15213	FF FF C4 93	11111111 11111111 11000100 10010011

Sample Questions



- What are the different ways to represent signed integer numbers in a computer system? Explain each one with an example. Also, mention their relative advantages and disadvantages.
- Why the 2's complement representation for signed integer is better than the one's complement and the sign and magnitude representation?
- How to detect overflow during any arithmetic operation between two signed integers? Explain with an example.
- What is sign extension? Explain using an example.
- What do you understand by "big-endian" and "little-endian" assignments of memory addresses? Explain. **(Answer: see later!)**