INFORMATION REPRESENTATION: PART 2

- Signed Integer representation
- Fixed Point representation

Signed Integers: introduction

With n bits, we have 2^n distinct values.

assign about half to positive integers (1 through 2^{n-1})
and about half to negative (- 2^{n-1} through -1)

that leaves two values: one for 0, and one extra

Positive integers

just like unsigned – zero in *most significant* (MS) bit
00101 = 5

Negative integers: 3 methods

- sign-magnitude – set MS bit to show negative, other bits are the same as unsigned. Standard method used in decimal representation

  10101 = -5
- One’s complement (1C)
- Two’s complement (2C)

Signed Magnitude

- Leading bit (MS) is the sign bit: 0 stands for +, 1 stands for –
- Example: 5 bits: 1 for sign + 4 for magnitude

$$A = a_4a_3a_2a_1a_0$$
$$= (-1)^{a4}\,(a_3 2^3 + a_2 2^2 + a_1 2^1 + a_0 2^0)$$

With (N+1) bits (from 0 to N), range is:
$-2^N + 1 < i < 2^N - 1$

-4	10100
-3	10011
-2	10010
-1	10001
-0	10000
+0	00000
+1	00001
+2	00010
+3	00011
+4	00100

One's Complement (1C)

- If A is positive: standard binary representation + MSB = 0
- If A is negative:
  1. take –A (which is positive)
  2. generate binary representation using positive number rule
  3. Invert all bits. Necessarily MSB = 1

With (N+1) bits (from 0 to N), range is:
$-2^N + 1 < i < 2^N - 1$

-4	11011
-3	11100
-2	11101
-1	11110
-0	11111
+0	00000
+1	00001
+2	00010
+3	00011
+4	00100

One’s Complement: property + generalization

- Let us assume (N+1) bits representation, 1C(A) will denote the binary representation of A using 1’s Complement representation, then

  $1C(A) + 1C(-A) = 111.....111 = 2^{(N+1)} - 1$

  $00010 + 11101 = 11111$

- Let us assume decimal representation, 9’s Complement can be defined in a similar manner as 1’s complement.
- Example: 2 digits available:
  - Positive numbers: for numbers from 0 to 49, standard decimal representation is used. So, +21 is represented by 21. No sign symbol is used.
  - Negative numbers: for numbers from -49 to 0, first consider opposite number (which is positive) and then complement every digit to 9. So -46 is represented by 53. Again no sign symbol is used.

Nine's complement representation

- If A is positive: standard decimal representation (no sign symbol used)
- If A is negative:
  1. take –A (which is positive)
  2. generate standard decimal representation (for positive number)
  3. Complement to 9 all digits, one by one.

- Let us assume 9's Complement representation on N digits, let us denote the 9's Complement of A by 9C(A), then

  $9C(A) + 9C(-A) = 999\ldots..999 = 10^N - 1$

  $357 + 642 = 999$

Two's Complement (True complement)

- If A is positive: standard binary representation + MSB = 0
- If A is negative:
  1. take –A (which is positive)
  2. generate standard binary representation
  3. Invert all bits. One's complement is generated.
  4. Add 1 to result. Necessarily MSB = 1
- REMARK: -0 represented by 11111 then adding 1 gives 100000. The 6th leftmost bit has to be dropped therefore +0 and -0 have the same representation 00000

-16	10000
…	…
-3	11101
-2	11110
-1	11111
0	00000
+1	00001
+2	00010
+3	00011
…	…
+15	01111

With (N+1) bits (from 0 to N), range is:
$-2^N < i < 2^N - 1$

Two's Complement: properties

- Let us assume (N+1) bits representation and A be a positive number, 2C(A) (resp. 1C(A)) will denote the binary representation of A using 2's Complement representation

  $1C(-A) + 1 = 2C(-A)$

- And

  $2C(A) + 2C(-A) = 100\ldots..000 = 2^{(N+1)}$

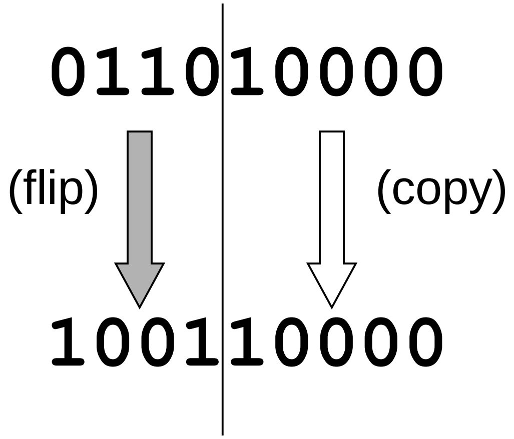
  $00010 + 11110 = 100000$

Two’s Complement Shortcut

If A is negative then consider –A

1. First method: flip all bits and then add one
2. Second method: copy bits from right to left until (and including) the first “1”, flip remaining bits to the left

```
   011010000
   100101111   (1’s comp)
 +         1
 ___________
   100110000
```



From Two’s Complement to Decimal

- If A is positive (MSB =0) : use standard binary to decimal conversion
- If A is negative (MSB=1):
  1. Compute A-1
  2. Flip all bits
  3. Use standard binary to decimal conversion.
  4. Insert a minus sign in front.

EXAMPLE: Decimal value 10111.

1. 10111 – 1 = 10110
2. Then 01001 which corresponds to 9
3. MSB =1 implies that 10111 represents -9

From Two's Complement to Decimal (simpler)

Let us assume a 2C representation on (N+1) bits

$A = a_N a_{(N-1)} \ldots . a_2 a_1 a_0$

$= -a_N 2^N + a_{(N-1)} 2^{(N-1)} + \ldots . + a_2 2^2 + a_1 2^1 + a_0 2^0$

MAJOR POINT: The first term of the sum is precede by a minus sign.

Example: 10111

$-16 + 4 + 2 + 1 = -9$

Sign Extension (SEXT): SM representation

- Let us assume sign + magnitude representation. Increasing the number of bits to represent a number simply amounts to padding with zeros after the sign bit.

4-bit	**8-bit**
`0100` (4)	`00000100` (still 4)
`1100` (-4)	`10000100` (still -4)

Sign Extension (SEXT): 1C representation

- Let us assume sign + magnitude representation. Increasing the number of bits to represent a number simply amounts to padding with zeros after the sign bit.

4-bit	**8-bit**
`0100` (4)	`00000100` (still 4)
`1011` (-4)	`00001011` (11 not -4)

Instead, replicate the MS bit -- the sign bit:

4-bit	**8-bit**
`0100` (4)	`00000100` (still 4)
`1011` (-4)	`11111011` (still -4)

Sign Extension (SEXT): 2C representation

Let us assume 2C representation. Padding with zeros will not work

4-bit	**8-bit**
`0100` (4)	`00000100` (still 4)
`1100` (-4)	`00001100` (12 not -4)

Instead, replicate the MS bit -- the sign bit:

4-bit	**8-bit**
`0100` (4)	`00000100` (still 4)
`1100` (-4)	`11111100` (still -4)

Operating on signed integers: addition

Let us assume a brute force method:

- Consider the sign bit as a regular data bit
- This means adding sign bits together
- Propagating carry on the sign bits from the magnitude bits.

Simple but a priori hard to expect correct results ☺.

Let us try with examples with our signed integer representations.

We will compute (+5+7), ((-5)+7), (5+(-7)), and ((-5) + (-7)) using 5 bits representations.

Sign + Magnitude: 5 (1+4) Bits addition (1)

				1	1	1	
+5	SM		0	0	1	0	1
+7	SM		0	0	1	1	1
+12	SM		0	1	1	0	0

.

Sign + Magnitude: 5 (1+4) Bits addition (2)

				1	1	1	
+5	SM		0	0	1	0	1
+7	SM		0	0	1	1	1
+12	+12		0	1	1	0	0

				1	1	1	
-5	SM		1	0	1	0	1
+7	SM		0	0	1	1	1
+2	-12		1	1	1	0	0

.

Sign + Magnitude: 5 (1+4) Bits addition (3)

				1	1	1	
+5	SM		0	0	1	0	1
+7	SM		0	0	1	1	1
+12	+12		0	1	1	0	0

				1	1	1	
-5	SM		1	0	1	0	1
+7	SM		0	0	1	1	1
+2	-12		1	1	1	0	0

				1	1	1	
+5	SM		0	0	1	0	1
-7	SM		1	0	1	1	1
-2	-12		1	1	1	0	0

Sign + Magnitude: 5 (1+4) Bits addition (4)

				1	1	1	
+5	SM		0	0	1	0	1
+7	SM		0	0	1	1	1
+12	+12		0	1	1	0	0

				1	1	1	
-5	SM		1	0	1	0	1
+7	SM		0	0	1	1	1
+2	-12		1	1	1	0	0

				1	1	1	
+5	SM		0	0	1	0	1
-7	SM		1	0	1	1	1
-2	-12		1	1	1	0	0

		1		1	1	1	
-5	SM		1	0	1	0	1
-7	SM		1	0	1	1	1
-12	+12	1	0	1	1	0	0

1 Complement: 5 (1+4) Bits addition (1)

				1	1	1	
+5	1C		0	0	1	0	1
+7	1C		0	0	1	1	1
+12	+12		0	1	1	0	0

1 Complement: 5 (1+4) Bits addition (2)

				1	1	1	
+5	1C		0	0	1	0	1
+7	1C		0	0	1	1	1
+12	+12		0	1	1	0	0

			1	1	1		
-5	1C		1	1	0	1	0
+7	1C		0	0	1	1	1
+2	+1	1	0	0	0	0	1

▪

1 Complement: 5 (1+4) Bits addition (3)

				1	1	1	
+5	1C		0	0	1	0	1
+7	1C		0	0	1	1	1
+12	+12		0	1	1	0	0

			1	1	1		
-5	1C		1	1	0	1	0
+7	1C		0	0	1	1	1
+2	+1	1	0	0	0	0	1

+5	1C		0	0	1	0	1
-7	1C		1	1	0	0	0
-2	-2		1	1	1	0	1

1 Complement: 5 (1+4) Bits addition (4)

				1	1	1	
+5	1C		0	0	1	0	1
+7	1C		0	0	1	1	1
+12	+12		0	1	1	0	0

			1	1	1		
-5	1C		1	1	0	1	0
+7	1C		0	0	1	1	1
+2	+1	1	0	0	0	0	1

+5	1C		0	0	1	0	1
-7	1C		1	1	0	0	0
-2	-2		1	1	1	0	1

		1	1				
-5	1C		1	1	0	1	0
-7	1C		1	1	0	0	0
-12	-13	1	1	0	0	1	0

2 Complement: 5 (1+4) Bits addition (1)

				1	1	1	
+5	2C		0	0	1	0	1
+7	2C		0	0	1	1	1
+12	+12		0	1	1	0	0

.

2 Complement: 5 (1+4) Bits addition (2)

				1	1	1	
+5	2C		0	0	1	0	1
+7	2C		0	0	1	1	1
+12	+12		0	1	1	0	0

			1	1	1	1	
-5	2C		1	1	0	1	1
+7	2C		0	0	1	1	1
+2	+2	1	0	0	0	1	0

.

2 Complement: 5 (1+4) Bits addition (3)

				1	1	1	
+5	2C		0	0	1	0	1
+7	2C		0	0	1	1	1
+12	+12		0	1	1	0	0

			1	1	1	1	
-5	2C		1	1	0	1	1
+7	2C		0	0	1	1	1
+2	+2	1	0	0	0	1	0

						1	
+5	2C		0	0	1	0	1
-7	2C		1	1	0	0	1
-2	-2		1	1	1	1	0

2 Complement: 5 (1+4) Bits addition (4)

				1	1	1	
+5	2C		0	0	1	0	1
+7	2C		0	0	1	1	1
+12	+12		0	1	1	0	0

			1	1	1	1	
-5	2C		1	1	0	1	1
+7	2C		0	0	1	1	1
+2	+2	1	0	0	0	1	0

						1	
+5	2C		0	0	1	0	1
-7	2C		1	1	0	0	1
-2	-2		1	1	1	1	0

		1	1		1	1	
-5	2C		1	1	0	1	1
-7	2C		1	1	0	0	1
-12	-12	1	1	0	1	0	0

2 Complement: 5 (1+4) Bits: general addition (1)

			0	x	x	x	
A	2C		0	X	X	X	X
B	2C		0	X	X	X	X
			0	X	X	X	X

OK

+5	2C		0	0	1	0	1
+8	2C		0	1	0	0	0
+13			0	1	1	0	1

2 Complement: 5 (1+4) Bits: general addition (2)

			0	x	x	x	
A	2C		0	X	X	X	X
B	2C		0	X	X	X	X
			0	X	X	X	X

			1	x	x	x	
A	2C		0	X	X	X	X
B	2C		0	X	X	X	X
			1	X	X	X	X

OK

OVERFLOW: adding 2 positive numbers generates a negative number

+5	2C		0	0	1	0	1
+8	2C		0	1	0	0	0
+13			0	1	1	0	1

			1				
+10	2C		0	1	0	1	0
+7	2C		0	1	0	0	1
+17			1	0	0	1	1

2 Complement: 5 (1+4) Bits: general addition (3)

			0	x	x	x	
A	2C		0	X	X	X	X
B	2C		1	X	X	X	X
			1	X	X	X	X

OK

+5	2C		0	0	1	0	1
-6	2C		1	1	0	1	0
-1			1	1	1	1	1

2 Complement: 5 (1+4) Bits: general addition (4)

			0	x	x	x	
A	2C		0	X	X	X	X
B	2C		1	X	X	X	X
			1	X	X	X	X

OK

			1	x	x	x	
A	2C		0	X	X	X	X
B	2C		1	X	X	X	X
		1	0	X	X	X	X

OK

+5	2C		0	0	1	0	1
-6	2C		1	1	0	1	0
-1			1	1	1	1	1

			1	1			
+12	2C		0	1	1	0	0
-2	2C		1	1	1	1	0
+10		1	0	1	0	1	0

2 Complement: 5 (1+4) Bits: general addition (5)

			1	x	x	x	
A	2C		1	X	X	X	X
B	2C		1	X	X	X	X
		1	1	X	X	X	X

OK

			1	1	1	1	
-5	2C		1	1	0	1	1
-3	2C		1	1	1	0	1
-8		1	1	1	0	0	0

2 Complement: 5 (1+4) Bits: general addition (6)

			1	x	x	x	
A	2C		1	X	X	X	X
B	2C		1	X	X	X	X
		1	1	X	X	X	X

			0	x	x	x	
A	2C		1	X	X	X	X
B	2C		1	X	X	X	X
		1	0	X	X	X	X

OK

OVERFLOW: adding 2 negative numbers generates a positive number

			1	1	1	1	
-5	2C		1	1	0	1	1
-3	2C		1	1	1	0	1
-8		1	1	1	0	0	0

			1				
-12	2C		1	0	1	0	0
-8	2C		1	1	0	0	0
-20		1	0	1	1	0	0

2C Subtraction

Negate subtrahend (2nd no.) and add.

- assume all integers have the same number of bits
- Ignore/drop carry out
- for now, assume that difference fits in n-bit 2’s comp. representation

```
    01101000 (104)       11110110 (-10)
 –  00010000 (16)    –   11110111 (-9)

    01101000 (104)       11110110 (-10)
 +  11110000 (-16)   +   00001001 (9)
    --------             --------
    01011000 (88)        11111111 (-1)
```

Assuming 8-bit 2’s complement numbers.

Addition/substraction on 2C numbers: wrapup

- Perform addition on numbers represented by the same number of bits: if necessary use Sign Extension
- Addition is very simple: consider the sign as a regular data bit: add and propagate carry.
- Substraction is also simple.
- We have overflow if:
  1. signs of both operands are the same, and
  2. sign of sum is different.
- Another overflow test -- easy for hardware:

  carry into MS bit does not equal carry out

Limitations of integer representations

- Most numbers are not integer!

  Even with integers, there are two other considerations:

- Range:

  The magnitude of the numbers we can represent is determined by how many bits we use:

  - e.g. with 32 bits the largest number we can represent is about +/- 2 billion, far too small for many purposes.

- Precision:

  The exactness with which we can specify a number:

  - e.g. a 32 bit number gives us 31 bits of precision, or roughly 9 figure precision in decimal repesentation.

- We need (at least) another data type! In fact we will use 2!

Introduction to Fixed Point Representation

Our decimal system handles non-integer *real/rational numbers* numbers by adding yet another symbol - the decimal comma (,) to make a *fixed point* notation:

e.g. 3456,78 = $3 \times 10^3 + 4 \times 10^2 + 5 \times 10^1 + 6 \times 10^0 + 7 \times 10^{-1} + 8 \times 10^{-2}$

REMARK: Anglo Saxon world uses Decimal point (.) instead of Decimal comma (,)

Easy extension to binary

e.g. 11,1101 = $1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4}$

e.g. 11,1101 = 2 + 1 + 0,5 + 0,25 + 0, 0625 = 3,8125

Fixed-Point Representation

How can we represent fractions?

- Use a “binary comma” to separate positive from negative powers of two -- just like “decimal comma.”
- 2’s comp addition and subtraction still work.
  - if binary points are aligned

$2^{-1} = 0,5$

$2^{-2} = 0,25$

$2^{-3} = 0,125$

```
  00101000,101 (40,625)
+ 11111110,110 (-1,25)
  ------------
  00100111,011 (39,375)
```

No new operations -- same as integer arithmetic.

From Fixed Point Decimal to Fixed Point Binary (1)

	0,625	
X	2	
=	1,250	1
=	0,250	
X	2	
=	0,5	0
X	2	
=	1,00	1
=	0,00	

Iterative multiplication by 2.

1. Multiply by 2
2. If result greater or equal to 1, insert 1 in the representation and replace 1 by 0 in the process
3. If result is smaller (strictly) than 1, insert 0 in the representation
4. Keep on iterating until 0,00 then STOP

REMARK: the process can never STOP ☺

0,	**1**	**0**	**1**

From Fixed Point Decimal to Fixed Point Binary (2)

0,	0	0	0	1	1	0	0	1	1	…	…	…	…		

	0,1	
X	2	
=	0,2	0
X	2	
=	0,4	0
X	2	
=	0,8	0
X	2	
=	1,6	1
=	0,6	
X	2	
=	1,2	1
=	0,2	

	0,2	
X	2	
=	0,4	0
X	2	
=	0,8	0
X	2	
=	1,6	1
=	0,6	
X	2	
=	1,2	1
=	0,2	

Fixed Point Representation: Generalization

Fixed Point representations can be easily extended to Base β:

$$A = a_{(N-1)}\ldots\ldots a_2 a_1 a_0\ ,\ a_{-1} a_{-2}\ldots\ldots a_{(P-1)} a_P$$

$$= a_{(N-1)}\beta^{(N-1)} + \ldots\ldots + a_2\beta^2 + a_1\beta^1 + a_0\beta^0\ +$$

$$a_{-1}\beta^{-1} + a_{-2}\beta^{-2}\, a_{-1}\beta^{-1} + a_{-2}\beta^{-2}\, a_{-1}\beta^{-1} + a_{-2}\beta^{-2}$$

Where $0 \leq a_i < \beta$ i positive and $0 \leq a_i < \beta$ i negative

Format: Q(N,P, β) where N denotes the number of digits on the left side of the comma (magnitude or integer part), P denotes the number of digits on the right side of the comma (fractional part).

Rounding : Introduction

- Let us consider Q(1,2,10) and let us try to represent 1/3.

  1/3 = 0 , 33333…………..

  The fractional part is an infinite sequence of 3

- We need to have a limited representation to match our format constraints:
  1. Rounding down (arrondi par défaut): 1/3 is represented by 0,33
  2. Rounding up (arrondi par exces): 1/3 is represented by 0,34
  3. Rounding to the nearest (arrondi au plus près): 1/3 is represented by 0,33

- Rounding error: compute
  - | 1/3 – Rep(1/3) | absolute error
  - | 1/3 – Rep(1/3) | / (1/3) relative error

Rounding : generalization (1)

X is a number and let us consider its representation in a given format

1. Rep(X) denotes its representation in the given format
2. Rep-(X) is the largest number exactly represented in the format and inferior or equal to X
3. Rep+(X) is the smallest number exactly represented in the machine and strictly larger than X

Therefore Rep-(X) ≤ X < Rep+(X)

1. Rounding Down Rep(X) = Rep-(X)
2. Rounding Up Rep(X) = Rep+(X)
3. Rounding to nearest (simplified definition)

- Rep(X) = Rep-(X) if |Rep(X) – Rep-(X)| ≤ |Rep(X) – Rep+(X)|
- Rep(X) = Rep+(X) if |Rep(X) – Rep-(X)| > |Rep(X) – Rep+(X)|

Rounding : generalization (2)

Major issue: complexity of computing the rounding.

1. Rounding Down is very easy: drop digits
2. Rounding Up requires some more computation
3. Rounding to the Nearest is the most complex

Errors: Rounding Up or Down generates similar errors, Rounding to the nearest allows to get absolute errors (on average) twice smaller.

Rounding has to be done not only for initial representation but in general after every arithmetic operation.