COMPSCI 2GA3 Tutorial 5 Note

Note:

This note does NOT cover all the materials in Chapter 3 -- Only the ones rated to sample questions of this tutorial are included.

For any questions about the tutorials and courses, feel free to contact me. (Email: wangm235@mcmaster.ca)

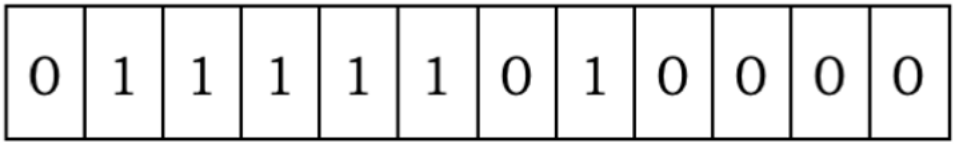
GLHF :)

Mingzhe Wang

Three different things that could raise confusion

N-bit representation (i.e. string of binary digits, sequence of bits)

e.g. A 12-bit representation.



It's just a string of binary digits, we DON'T know its value.
Because it is a string, you can write it as "0b011111010000"

Higher-radix notation

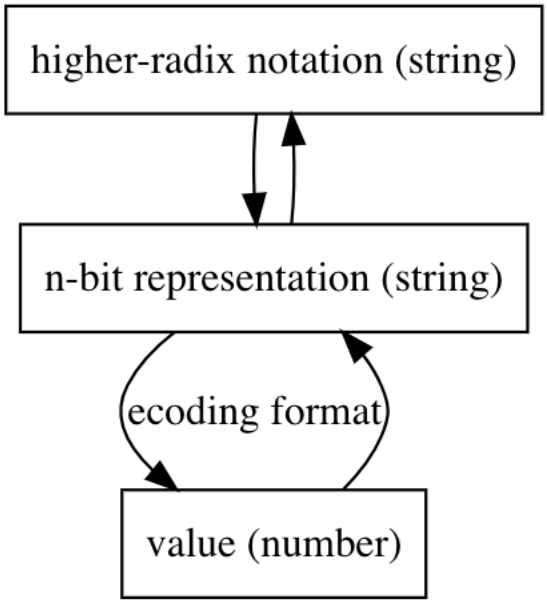
> Long strings of binary digits are tedious and error-prone to transcribe, so we usually use a higher-radix notation, choosing the radix so that it's simple to recover the original bits string.

e.g. For the above 12-bit representation, it can be represented by a higher-radix notation:
in base 16, it can be represented by "0x7D0";
in base 10, it can be represented by "2000".
(Note: 0b is the prefix notation for binary string, 0x is the prefix notation for hexadecimal string, no prefix notation for decimal string)

Encoding format

To know what's the value represented by a N-bit representation, we need to know its encoding format. There are three encoding formats: default encoding, signed magnitude encoding, and two's complement encoding.

Conclusion



NOTE: ONLY IF we know BOTH the N-bit representation (or its higher-radix notation) and its encoding format, we can determine the value (number in mathematical meaning) of this representation. Otherwise, it's just a string of binary digits.

Encoding format continuing…

We know that we have three encoding formats: default encoding, signed magnitude encoding, and two's complement encoding. Now let's explore their meaning.

Default encoding

Encoding Positive Integers

It is straightforward to encode positive integers as a sequence of bits. Each bit is assigned a weight. Ordered from right to left, these weights are increasing powers of 2. The value of an N-bit number encoded in this fashion is given by the following formula:

$$v = \sum_{i=0}^{N-1} 2^i b_i$$

2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	1	1	1	1	1	0	1	0	0	0	0

$V = 0*2^{11} + 1*2^{10} + 1*2^9 + \ldots$
$= 1024 + 512 + 256 + 128 + 64 + 16$
$= 2000$

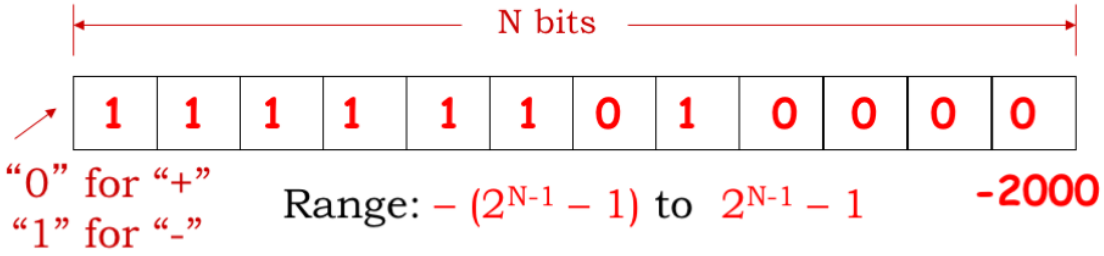
Smallest number: 0 Largest number: 2^N-1

Signed magnitude encoding

Encoding Signed Integers

We use a signed magnitude representation for decimal numbers, encoding the sign of the number (using "+" and "-") separately from its magnitude (using decimal digits).

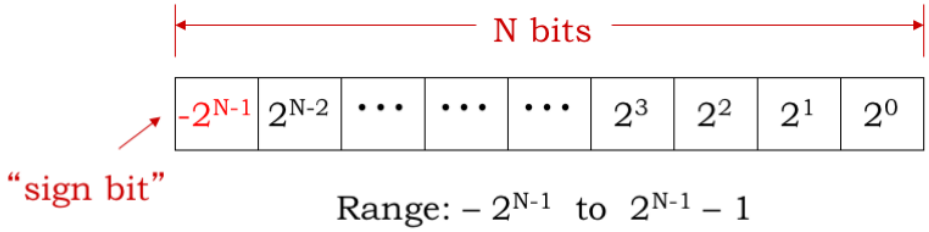
We could adopt that approach for binary representations:



But: two representations for 0 (+0, -0) and we'd need different circuitry for addition and subtraction

Two’s Complement Encoding

In a two’s complement encoding, the high-order bit of the N-bit representation has negative weight:

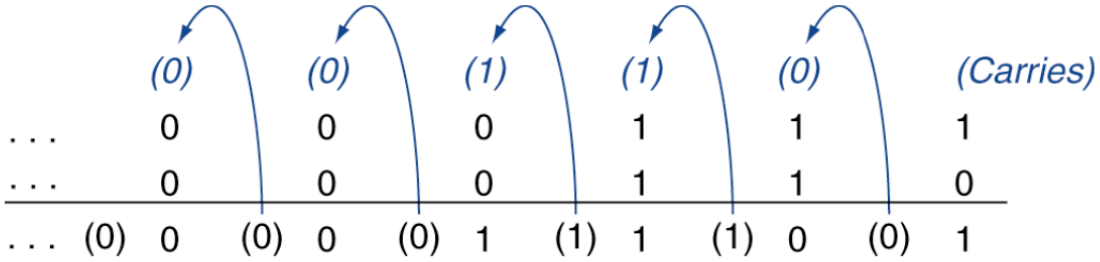


Range: -2^{N-1} to $2^{N-1} - 1$

- Negative numbers have “1” in the high-order bit
- Most negative number: 10…0000 -2^{N-1}
- Most positive number: 01…1111 $+2^{N-1} - 1$
- If all bits are 1: 11…1111 -1
- If all bits are 0: 00…0000 0

Integer Addition

- Example: 7 + 6



- Overflow if result out of range
  - Adding +ve and –ve operands, no overflow
  - Adding two +ve operands
    - Overflow if result sign is 1
  - Adding two –ve operands
    - Overflow if result sign is 0

Integer Subtraction

- Add negation of second operand
- Example: 7 – 6 = 7 + (–6)

  +7: 0000 0000 … 0000 0111
  –6: 1111 1111 … 1111 1010
  +1: 0000 0000 … 0000 0001

- Overflow if result out of range
  - Subtracting two +ve or two –ve operands, no overflow
  - Subtracting +ve from –ve operand
    - Overflow if result sign is 0
  - Subtracting –ve from +ve operand
    - Overflow if result sign is 1

Saturating arithmetic?

Saturation arithmetic is a version of arithmetic in which all operations such as addition and multiplication are limited to a fixed range between a minimum and maximum value.

If the result of an operation is greater than the maximum, it is set ("clamped") to the maximum; if it is below the minimum, it is clamped to the minimum. The name comes from how the value becomes "saturated" once it reaches the extreme values; further additions to a maximum or subtractions from a minimum will not change the result.

For example, if the valid range of values is from –100 to 100, the following *saturating arithmetic operations* produce the following values:

- 60 + 30 → 90.
- 60 + 43 → 100. (*not* the expected 103.)
- (60 + 43) – (75 + 75) → 0. (*not* the expected –47.) (100 – 100 → 0.)
- 10 × 11 → 100. (*not* the expected 110.)
- 99 × 99 → 100. (*not* the expected 9801.)
- 30 × (5 – 1) → 100. (*not* the expected 120.) (30 × 4 → 100.)
- (30 × 5) – (30 × 1) → 70. (*not* the expected 120. *not* the previous 100.) (100 – 30 → 70.)

Tricky? For two's complement encoding, how to get the n-bit representation for a number (value)

More Two's Complement

- Let's see what happens when we add the N-bit values for -1 and 1, keeping an N-bit answer:

```
  11…1111
+ 00…0001
  0000000
```

Just use ordinary binary addition, even when one or both of the operands are negative. 2's complement is perfect for N-bit arithmetic!

- To compute B-A, we'll just use addition and compute B+(-A). But how do we figure out the representation for -A?

$A+(-A) = 0 = 1 + -1$

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

$= \sim A + 1$

$$\begin{array}{r} 1 \\ -A_i \\ \hline \sim A_i \end{array}$$

To negate a two's complement value: bitwise complement and add 1.

You smart guys may have noticed: for non-negative numbers, their n-bit representations are the same for either sign magnitude or two's complement encoding.
So generally,
if the number >= 0, then simply use sign magnitude to get its n-bit representation.
if the number < 0, then use this formula: -A = ~A + 1.
e.g.
get the 8-bit representation of number -105
set A = 105, then
-105 = ~105 + 1
= ~01101001 + 1
= 10010110 + 1
= 10010111
therefore, the 8-bit representation of number -105 is '10010111'.
Done.

Reference

some slides: https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-004-computation-structures-spring-2017/c1/c1s1/index.htm#8