Numbers, Signed Numbers and Numbers as bits

Instructor: Dr. Vinicius Prado da Fonseca (vpradodafons@online.mun.ca)

Positional Numbers

- Positional notation
    - Digit multiplied by the base raised to the power of the digit position
- The value of a number in a base (radix), r with n positions is given by:
- Position zero is the rightmost digit (least significant)
- Position n-1 is the left most (most significant) position
- c_i is the digit at position i
- 0,1,2,3,...,r−1 digits
    - {0, 1} base 2
    - {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} base 10
- The position determines its significance

$$\sum_{0}^{n-1} c_i \cdot r^i$$

Positional Numbers

- 421_{10}

- 421_5

$$\sum_0^{n-1} c_i \cdot r^i$$

Digital circuits store only two values

- 11010_2

Positional Numbers

- 481_{10}

$4 \times 10^2 + 8 \times 10^1 + 1 \times 10^0 = 481_{10}$

- 421_5

$4 \times 5^2 + 2 \times 5^1 + 1 \times 5^0 = 111_{10}$

Digital circuits store only two values

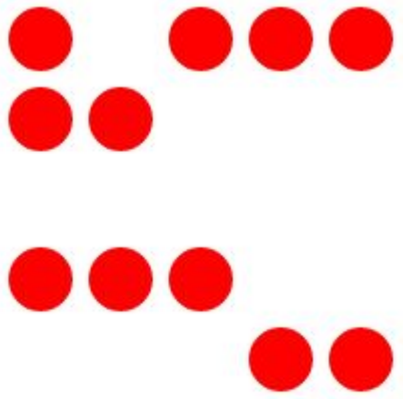
- 11010_2

$1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 26_{10}$

$$\sum_0^{n-1} c_i \cdot r^i$$

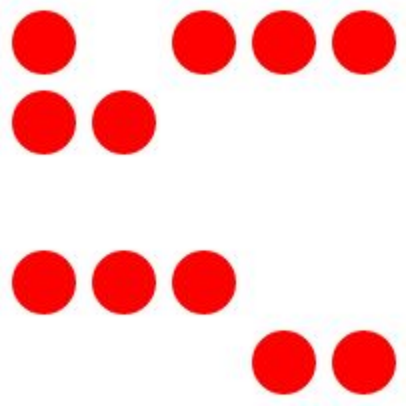
Positional Numbers

- Radix is chosen by convenience
- Computer systems uses radix 2, 8, 16
    - Easily converted
    - Compact representation
- How to describe the number of red dots in bases 2, 5 and 10?
    - Base 10 factors: 1, 10, 100, 1000,...
    - Base 2 factors: 1, 2, 4, 8, 16, ...
    - Base 5 factors: 1, 5, 25, 125, ...



Positional Numbers

- Radix is chosen by convenience
- Computer systems uses radix 2, 8, 16
  - Easily converted
  - Compact representation
- How to describe the number of red dots in bases 2, 5 and 10?
  - Base 10 factors: 1, 10, 100, 1000,...
  - Base 2 factors: 1, 2, 4, 8, 16, ...
  - Base 5 factors: 1, 5, 25, 125, ...
- 11_{10}, 1011_2, 21_5
- Online notes exercises



Converting To Decimal

- Computing the result in decimal arithmetic:

$321_4 =$

Converting To Decimal

- Computing the result in decimal arithmetic:

$321_4 = 3 \times 4^2 + 2 \times 4^1 + 1 \times 4^0 = 3 \times 16 + 2 \times 4 + 1 = 48 + 8 + 1 = 57$

- int constructor in Python:

>>> int("0101", 2) = $1*2^2 + 1*2^0 = 4 + 1 = 5_{10}$

>>> int("0101", 6) = $1*6^2 + 1*6^0 = 36 + 1 = 37_{10}$

>>> int("0101", 9) = $1*9^2 + 1*9^0 = 81 + 1 = 82_{10}$

Python, be default, will display a number using decimal. Thus int() converts the string into a integer, and Python displays the integer using decimal (i.e., there are two conversions).

Binary, Octal, and Hexadecimal

- Binary, base 2, has the digits:

  0,1

- Octal, base 8 (`0o`), has the digits:

  0, 1, 2, 3, 4, 5, 6, 7

- The digits for hexadecimal (`0x`), base 16, are:

  0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

A = 10_{10} B = 11_{10} C = 12_{10} D = 13_{10} E = 14_{10} F = 15_{10}

Binary, Octal, and Hexadecimal

Octal	Binary
0_8	000_2
1_8	001_2
2_8	010_2
3_8	011_2
4_8	100_2
5_8	101_2
6_8	110_2
7_8	111_2

Binary, Octal, and Hexadecimal

- One position in an octal number - three positions in a binary number
- any grouping of three binary digits can be converted into octal

73_8 =

123_8 =

6210_8 =

Binary, Octal, and Hexadecimal

- One position in an octal number - three positions in a binary number
- any grouping of three binary digits can be converted into octal

$73_8 = 111\ 011_2$

$123_8 = 001\ 010\ 011_2$

$6210_8 = 110\ 010\ 001\ 000_2$

Binary, Octal, and Hexadecimal

Hexadecimal	Binary	Hexadecimal	Binary
0_{16}	0000_2	8_{16}	1000_2
1_{16}	0001_2	9_{16}	1001_2
2_{16}	0010_2	A_{16}	1010_2
3_{16}	0011_2	B_{16}	1011_2
4_{16}	0100_2	C_{16}	1100_2
5_{16}	0101_2	D_{16}	1101_2
6_{16}	0110_2	E_{16}	1110_2
7_{16}	0111_2	F_{16}	1111_2

Binary, Octal, and Hexadecimal

- One position in an hex number - four positions in a binary number
- any grouping of four binary digits can be converted into hexadecimal

AF_{16} =

123_{16} =

$CAFE_{16}$ =

Binary, Octal, and Hexadecimal

- One position in an hex number - four positions in a binary number
- any grouping of four binary digits can be converted into hexadecimal

$AF_{16} = 1010\ 1111_2$

$123_{16} = 0001\ 0010\ 0011_2$

$CAFE_{16} = 1100\ 1010\ 1111\ 1110_2$

Decimal To Binary, Octal, and Hex Algorithm

- Place value technique
  - From previous slides: $\mathbf{321_4 = 3 \times 4^2 + 2 \times 4^1 + 1 \times 4^0 = 3 \times 16 + 2 \times 4 + 1 = 48 + 8 + 1 = 57_{10}}$
- Successive Division Technique
  - Example
    - 67_{10} to binary
    - 126_{10} to hexadecimal

Decimal To Binary, Octal, and Hex Algorithm

- Mod operation can be used in programming languages

```
def to_bin( v ) :
      s = ""
      while v > 0:
            s += str(v % 2)          # Extract least significant binary digit using mod
            v //= 2                  # Go to the next digit
      if len(s) == 0:
            return '0'
      else:
            return s[::-1]           # reverse string
```

- Change “2” for other bases octal or hex (needs character conversions)

Numbers as Bits

8-bit stored in a computer

MSB LSB

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

$$\sum_{0}^{8-1} b_i . 2^i$$

0	1	1	0	0	1	1	1

$1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 103_{10}$

Numbers as Bits

Smallest unsigned 8-bit stored in a computer: 0

0_7	0_6	0_5	0_4	0_3	0_2	0_1	0_0

Largest unsigned 8-bit stored in a computer: 255

1_7	1_6	1_5	1_4	1_3	1_2	1_1	1_0

Range of a 8-bit unsigned integer is 0 to 2^8-1 = 255

Range of 16-bit unsigned integer is 0 to 2^{16}-1 = 655352

n-bit unsigned integer has a range of 0 to 2^n-1

MEMORIAL UNIVERSITY
www.mun.ca

Signed Numbers

- Positive, negative or zero.
  - Sign Magnitude, Two's Complement, and One's Complement
- The number of bits are important, it defines the sign bit.
- Sign Magnitude (8-bit):

$$v = \begin{cases} \sum_{0}^{6} b_i \cdot 2^i & \text{if } s_7 = 0 \\ -\sum_{0}^{6} b_i \cdot 2^i & \text{if } s_7 = 1 \end{cases}$$

-127 to 127, two zeros, n-bit sign magnitude range of $-(2^{n-1}-1)$ to $(2^{n-1}-1)$

4 bits = -7 to 7

Signed Numbers

- Sign Magnitude

0111

1001

111001

Signed Numbers

- Sign Magnitude

0111 = 7

1001 = -1

111001 = -25

-1 + 1 = 1001 + 0001 = ?

Signed Numbers

- Sign Magnitude

0111 = 7

1001 = -1

111001 = -25

-1 + 1 = 1001 + 0001 = 1010 (-2) (subtraction needs extra components)

Signed Numbers

- One's Complement (invert all bits if the sign bit is 1, convert to decimal, include the sign)

$$s_{n-1} \cdot -(2^{n-1} - 1) + \sum_{0}^{n-2} b_i \cdot 2^i$$

0010 = ?

1000 = ?

1010 = ?

Signed Numbers

- One's Complement

0010 = 2

1000 = -7 (0111)

1010 = -5 (0101)

-1 + 1 = 1110 + 0001 = ?

Signed Numbers

- One's Complement range: $-(2^{n-1}-1)$ to $(2^{n-1}-1)$. 4bits = -7 to 7

-1 + 1 = 1110 + 0001 = 1111 + 1 = 0000 (-0? Two representations for zero, add +1)

0101 (5)
+ 1110 (-1)
=

1101 (-2)
+ 1110 (-1)
=

Signed Numbers

- One's Complement

-1 + 1 = 1110 + 0001 = 1111 + 1 = 0000 (-0? Two representations for zero, add +1)

0101 (5)
\+ 1110 (-1)
= 1 0011 (3) + 1 = 0100 (4) (Extra operation)

1101 (-2)
\+ 1110 (-1)
= 1 1011 (-4) + 1 = 1100 (-3) (Extra operation)

Signed Numbers

- Two's Complement decimal value:

$$s_{n-1} \cdot -2^{n-1} + \sum_{0}^{n-2} b_i \cdot 2^i$$

4-bit two's complement represented by 1011:

n-bit two's complement range of -(2^{n-1}) to (2^{n-1}-1). 4 bits = -8 to 7

Signed Numbers

- Two's Complement decimal value:

$$s_{n-1} \cdot -2^{n-1} + \sum_{0}^{n-2} b_i \cdot 2^i$$

4-bit two's complement represented by 1011:

$$1x(-2)^3 + 0x2^2 + 1x2^1 + 1x2^0 = -5$$

1011 (inv) -> 0100 (+1) -> 0101 (5) (inv) -> 1010 (+1)

n-bit two's complement range of $-(2^{n-1})$ to $(2^{n-1}-1)$

Signed Numbers

- Two's Complement decimal value:
  5 = 0101
  -1 = 0001 => 1110 +1 => 1111 (include the extra operation in the representation)

```
    0101 (5)
  + 1111 (-1)
-----------------
 1   0100 (4) (No extra op, no extra component)
```

Negating A Number

Inverting the signal

- Sign magnitude: flip que sign bit
- One’s complement: invert all bits
- Two’s complement: one’s complement + 1

Example operations

Addition 5 + (-1) = 4

- Sign magnitude

  0101 + 1001 = 0101 - 0001 = 0100 (need subtraction component)

- One's complement (add 1)

  0101 (5) + 1110 (-1) = 0011 (3) + 1 = 0100 (4) (Extra operation)

- Two's complement

  0101 (5) + 1111 (-1) = 0100 (4) (No extra op, no extra component)

Questions?

Next: 03 - Gates for Digital Circuits