The number system

Number system and Binary numbers
What is the number system?

The number system is a writing system for expressing numbers. It is composed of a collection of *symbols* used to represent small numbers, together with a system of rules for representing larger numbers.

<table>
<thead>
<tr>
<th>Number Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System</strong></td>
</tr>
<tr>
<td>Binary</td>
</tr>
<tr>
<td>Octal</td>
</tr>
<tr>
<td>Decimal</td>
</tr>
<tr>
<td>Hexadecimal</td>
</tr>
</tbody>
</table>

Example of a 32-bit instruction

```
LDR <Rd>, [<Rn>, #<immed_5> * 4]
```
Example of a 32-bit instruction

Instruction format (machine language)

- **Machine Language**
  - Computers do not understand “add R8, R17, R18”
  - Instructions are translated to machine language (1s and 0s)

- **Example:**
  
  $\text{add R8, R17, R18} \rightarrow$

  \[
  \begin{align*}
  &00000010 \ 00110010 \ 01000000 \ 00100000
  \end{align*}
  \]

The unary number system

What is the first numeral system we learn?
What is the most natural numeral system?

The unary uses one symbol.
This is the simplest way of counting
The Roman number system

Numbers in the Roman system are represented by combinations of letters from the Latin alphabet.

Roman numerals used today, have seven symbols, each with a fixed integer value:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>I</th>
<th>V</th>
<th>X</th>
<th>L</th>
<th>C</th>
<th>D</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>1</td>
<td>5</td>
<td>10</td>
<td>50</td>
<td>100</td>
<td>500</td>
<td>1,000</td>
</tr>
</tbody>
</table>

The base of a number system

A radix, or base, is the number of unique digits, including zero, used to represent numbers in a positional numeral system.

image credit: http://thepictutor.com/decimal.html
The base of a number system

Generalized form of positional systems in Base B:
Commonly used numeral systems include:

**Binary** (Base 2): 0, 1

**Octal** (Base 8): 0, 1, 2, 3, 4, 5, 6, 7

**Decimal** (Base 10): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

**Hexadecimal** (Base 16): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Consider a number $N$ with $n+1$ digits:

$$N = a_n \times a_{n-1} \times a_{n-2} \times \ldots \times a_0$$

$N_B = a_n \times B^n + a_{n-1} \times B^{n-1} + a_{n-2} \times B^{n-2} + \ldots + a_2 \times B^1 + a_0 \times B^0$
The Egyptian number system

The Egyptians had a base-10 system of hieroglyphs for numerals. They separate symbols for one unit, one ten, one hundred, one thousand, one ten thousand, one hundred thousand, and one million.

e.g. The number 276 in hieroglyphs, requires fifteen symbols:
The Egyptian number system

<table>
<thead>
<tr>
<th>1</th>
<th>10</th>
<th>100</th>
<th>1000</th>
<th>10000</th>
<th>100000</th>
<th>1 * 10^6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Egyptian numeral hieroglyphs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

e.g. The number 4622 in hieroglyphs, requires these symbols:

Decimal to binary conversion

Conversion of a number from decimal to binary:

To convert a decimal number to binary:

1. First, subtract the largest possible power of two

2. Keep subtracting the next largest possible power of 2 from the remainder, marking 1s in each column where this is possible and 0s where it is not.
**Decimal to binary conversion**

Converting the decimal (base 10) number 86 to binary (base 2)

64 is the largest power of 2 that goes into 86. The result is:

<table>
<thead>
<tr>
<th></th>
<th>64</th>
<th>32</th>
<th>16</th>
<th>8</th>
<th>4</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

86 - 64 is 22. 32 is larger than 22, so a 0 is placed in the bit for the value 32. 16 is less than 22, so a 1 is placed in the bit for the value 16.

<table>
<thead>
<tr>
<th></th>
<th>64</th>
<th>32</th>
<th>16</th>
<th>8</th>
<th>4</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>?</td>
</tr>
</tbody>
</table>

22 - 16 is 6. 8 is larger than 6.
The bits 4+2 equal 6 so each of those bits become 1

<table>
<thead>
<tr>
<th></th>
<th>64</th>
<th>32</th>
<th>16</th>
<th>8</th>
<th>4</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>?</td>
</tr>
</tbody>
</table>

This 86 in decimal is 1010110 in binary. $86_{10} = 1010110_2$

---

**Binary to decimal conversion**

Conversion of a number from binary to decimal:

$$N_2 = 1 \times 2^9 + 0 \times 2^8 + 0 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 +$$
$$0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

$$= 537$$

---

Houngninou CSCE 110: Programming I 15

---

Houngninou CSCE 110: Programming I 16
Binary conversion: bin()

The `bin()` function converts an integer to a binary string.

**input:** the integer to convert to binary

**output:** a binary string

If the input is not an integer, `bin()` raises a `TypeError` exception.

```
>>> # The decimal number 50 in binary
>>> # '0b' is the prefix for a binary number
>>> bin(50)
'0b110010'
>>> bin(357)
'0b101100101'
>>> # ignoring the prefix '0b' in the output of bin()
>>> bin(357)[2:]
'101100101'
```

Binary conversion: bin()

The `bin()` function converts an integer to a binary string.

**input:** the integer to convert to binary

**output:** a binary string

If the input is not an integer, `bin()` raises a `TypeError` exception.

```
> numbers = [25, 16, '128']
for n in numbers:
    try:
        print(bin(n))
    except TypeError:
        print("Incorrect type for binary conversion")

0b11001
0b100000
Incorrect type for binary conversion
```
Decimal conversion: int()

The int() function converts any base k number to decimal.

The syntax of int() method is: int(x, base)

```python
>>> # binary conversion to integer
>>> print(f"1110 is integer: {int('1110', 2)}")
>>> # hexadecimal conversion to integer
>>> print(f"AB is integer: {int('ABC', 16)}")
1110 is integer: 14
AB is integer: 2748
```

Decimal conversion: int()

The syntax of int() method is: int(x, base)

input x: number or string to convert

input base: base of the number in x

output: an integer

```python
>>> int('241103')
241103
>>> int('241103', 10)
241103
>>> int('100101', 2)
37
>>> int('9401BA', 16)
9699770
```
Number system: hexadecimal numbers

Conversion using Python functions.

The `hex()` function converts an integer to a hexadecimal number.

```python
def hex_conversion():
    numbers = [25, 16, 128]
    for n in numbers:
        try:
            print(hex(n))
        except TypeError:
            print("Incorrect type for binary conversion")

> [evaluate hex_conversion.py]
0x19
0x10
0x80
```

Number system: Binary addition

Rules for binary addition

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 10

for 1 + 1, we write down a zero in the right-most column and carry over a one to the next column.
Number system: Binary addition

1  # Convert decimal 19 to binary number '0b10011'
2  print(bin(19))
3
4  # 0b on the previous line means the result is binary
5  print(int('10011', 2))
6
7  # the 4 binary addition rules
8  rule1 = bin(int('0', 2) + int('0', 2))  # 0 + 0 = 0
9  rule2 = bin(int('0', 2) + int('1', 2))  # 0 + 1 = 1
10 rule3 = bin(int('1', 2) + int('0', 2))  # 1 + 0 = 1
11 rule4 = bin(int('1', 2) + int('1', 2))  # 1 + 1 = 10
12
13  # Adding binary numbers 10101 and 110 results in 11011
14  number = bin(int('10101', 2) + int('110', 2))
15  print(number)

Number system: bitwise operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>~</td>
<td>Binary NOT</td>
</tr>
<tr>
<td></td>
<td>Each bit of the output is 1 if the bit of x is 0, otherwise it is 0.</td>
</tr>
<tr>
<td>&amp;</td>
<td>Binary AND</td>
</tr>
<tr>
<td></td>
<td>Each bit of the output is 1 if the bit of x AND the bit of y is 1, otherwise it's 0.</td>
</tr>
<tr>
<td></td>
<td>Binary OR</td>
</tr>
<tr>
<td></td>
<td>Each bit of the output is 0 if the bit of x AND the bit of y is 0, otherwise it's 1.</td>
</tr>
<tr>
<td>^</td>
<td>Binary XOR</td>
</tr>
<tr>
<td></td>
<td>Each bit of the output is 1 if the bit of x is the one complement of the bit of y, otherwise it's 0.</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>Binary Left Shift</td>
</tr>
<tr>
<td></td>
<td>x &lt;&lt; y returns x with the bits shifted to the left by y places. Performs a multiplication of x by 2^y.</td>
</tr>
<tr>
<td>&gt;&gt;</td>
<td>Binary Right Shift</td>
</tr>
<tr>
<td></td>
<td>x &gt;&gt; y returns x with the bits shifted to the right by y places. Performs a floor division of x by 2^y.</td>
</tr>
</tbody>
</table>
Bitwise operator: NOT (~)

The NOT of a binary number is the value obtained by inverting all the bits in the binary number.

- We flip 0 to 1
- We flip 1 to 0

Truth table:

Bitwise operator: NOT (~)

How to read the NOT of a binary number?

In programming we read the NOT of a binary number in 2's complement format.

2's complement is a method of representing signed integers on computers.

To get the 2's complement of a binary number:

1. Invert all the bits.
2. Add 1 to the inverted bits.
Bitwise operator: NOT (~)

e.g.: What is ~12 in binary and in decimal?
Convert 12 to binary: 12 = 1100
Flip to bits: ~12 = 0011
To read 0011 in decimal, perform the 2’s complement of 0011.

<table>
<thead>
<tr>
<th>2's complement of 0011</th>
</tr>
</thead>
<tbody>
<tr>
<td>~12:</td>
</tr>
<tr>
<td>0011</td>
</tr>
<tr>
<td>flip the bits:</td>
</tr>
<tr>
<td>1100</td>
</tr>
<tr>
<td>add 1:</td>
</tr>
<tr>
<td>1101</td>
</tr>
</tbody>
</table>

1101 = 13

Answer: ~12 = 0011 = -13
Bitwise operator: AND (&)

The AND of a set of operands is 1 if and only if all of the operands are 1’s.

Truth table:
Bitwise operator: OR (|)

The OR of a set of operands is 1 if at least one of the operands is a 1.

Truth table:
Bitwise operator: XOR (^)

The XOR of a set of operands is 1 if both the operands differ.

Truth table:
Left shift: (<<)

A logical shift is a bitwise operation that shifts all the bits of its operand.

\[ n << x \] is \( n \) with the bits shifted to the left by \( x \) places.

Note: the left shift performs a multiplication of \( n \) by \( 2^x \).
Right shift: (>>)

A logical shift is a bitwise operation that shifts all the bits of its operand.

\( n >> x \) is \( n \) with the bits shifted to the right by \( x \) places.

Note: the right shift performs a floor division of \( n \) by \( 2^x \).
End