Introduction
In this lesson we will introduce and study
✓ How to represent information for computer
✓ Different numeric bases
✓ Expressive power of different representations

Representing Information
To work inside a computer
  Must understand how to represent information
    Way that computer can understand
    Today computer doesn’t grock our thoughts
      Getting close though

Until then must be able to express
  Numbers
  Symbols or characters
    Letters
    Control

Computers represent information as collections of 0’s and 1’s
  Expressed by state of electronic signals
  Called bits
    Bit only stores one thing

Stored in
  Memory
    RAM
    Registers
      Like small memories

In computer or any digital system
  Working with bits
  Only have limited number of bits

Can define specific groupings of bits
  Collections grouped in binary increments
    1, 2, 4, 8, 16...

Give special names to some groupings

  Bit – 1
  Nybble - 4
  Bytes - 8
  Words - 16, 32, 64...
Numbers
First thing important to represent is numbers
2 kinds
Integers
Floating point

Bases
Before learning to represent different kinds of numbers
Learn that numbers can be expressed in different bases

Most familiar base is base 10
Also called decimal

Computers and software programs generally use base 2
Also called binary

Several other bases in common usage in computer field
Base 8
Also called octal
Base 16
Also called hex or hexadecimal

Integer Numbers
Let’s now see how we can numeric information

An integer can be either
Unsigned
Express only positive numbers
Signed
Express both positive and negative numbers

Unsigned Integers
Most computer systems use weighted number system
Means we have collection of symbols
Let’s call this set S

Because it’s familiar
Let’s see how decimal works
*Decimal*

Our set of symbols

\[ S = \{0..9\} \]

To express number

Assign weight to symbol based upon its position in number

For decimal weights will be

1, 10, 100, 1000,...

Sum of weights multiplied by symbol value = number

**Example**

789

7

Assign weight \(10^2\)

Value = 7 \(\times\) \(10^2\)

8

Assign weight \(10^1\)

Value = 8 \(\times\) \(10^1\)

9

Assign weight \(10^0\)

Value = 9 \(\times\) \(10^0\)

Total - 700 + 80 + 9 = 789

Thus in general for decimal numbers

\[ \text{value} = \sum_{i=0}^{n-1} s_i \times 10^i \]

*Binary*

\[ S = \{0..1\} \]

To express number

Assign weight to symbol based upon its position in number

For binary weights will be

1, 2, 4, 8,...

Sum of weights multiplied by symbol value = number
Example
101
1
  Assign weight $2^2$
  Value = $1 \cdot 2^2$
0
  Assign weight $2^1$
  Value = $0 \cdot 2^1$
1
  Assign weight $2^0$
  Value = $1 \cdot 2^0$
Total - $4 + 0 + 1 = 5$

Thus in general for binary numbers
\[ \text{value} = \sum_{i=0}^{n-1} s_i \times 2^i \]

Octal
S = \{0,..,7\}
To express number
Assign weight to symbol based upon its position in number
For octal weights will be
1, 8, 64, 512……
Sum of weights multiplied by symbol value = number
Example
652
6
  Assign weight $8^2$
  Value = $6 \cdot 8^2$
5
  Assign weight $8^1$
  Value = $5 \cdot 8^1$
2
  Assign weight $8^0$
  Value = $1 \cdot 8^0$
Total - $384 + 40 + 1 = 425$

Thus in general for octal numbers
\[ \text{value} = \sum_{i=0}^{n-1} s_i \times 8^i \]
Hexadecimal

S = \{0..9, A..F\}

To express number

Assign weight to symbol based upon its position in number

For hex weights will be

1, 16, 256, 4096, ..... 

Sum of weights multiplied by symbol value = number

Hex symbols A..F

Represent number values 10..15

Example

\text{A4C}

A

Assign weight 16^2

Value = A \cdot 16^2

4

Assign weight 16^1

Value = 4 \cdot 16^1

C

Assign weight 16^0

Value = C \cdot 16^0

Total = 2560 + 64 + 12 = 2636

Thus in general for hex numbers

\text{value} = \sum_{i=0}^{n-1} s_i \times 16^i

Thus in general for any base B

\text{value} = \sum_{i=0}^{n-1} s_i \times B^i

How Big

Important to understand

For specified number of symbols

How large number can be represented

For 32 bit computer – what does that mean

How large of an integer can we represent

Sets a limit on expressive power of representation

Let’s again start with decimal
**Decimal**

With 3 decimal digits
Largest number
999

Can rewrite as
\[10^3 - 1\]

Observe 3 is number of digits

Or in general
\[10^n - 1\]

**Binary**

With 3 binary digits
Largest number
111

Can also express as
\[2^3 - 1\]

Observe 3 is number of digits

Or in general
\[2^n - 1\]

**Octal**

With 3 octal digits
Largest number
888

Can also express as
\[8^3 - 1\]

Observe 3 is number of digits

Or in general
\[8^n - 1\]

**Hexadecimal**

With 3 hex digits
Largest number
FFF

Can also express as
\[16^3 - 1\]

Observe 3 is number of digits

Or in general
\[16^n - 1\]
In computer

If computer uses 8 bit words
Largest unsigned integer
\[ 2^8 - 1 = 255 \]

If machine uses 32 bit words
Largest integer
\[ 2^{32} - 1 = 4 \times 10^9 \]

Signed Integers
Let’s work with a computer that uses 32 bit words

Signed number will be represented as
Sign bit on LHS
0 - positive
1 - negative

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | Sign |    | Number |

In C declaration

int a = 10;

Places integer value 10 in binary
Into word in memory

We can view memory as a bookshelf or two dimensional array
Each shelf holds a word
Each book represents a bit
Book is right side up
Represents a 1
Book is upside down
Represents a 0

By switching to signed numbers
We now have a range
\[ +2^{n-1} \] to \[ -2^{n-1} \]

We loose one number
Floating Point Numbers

Clearly we need to be able to express numbers larger than 65K
Done using floating point numbers
Thus

In C declaration

```c
float a = 10.9;
```

Places floating point value 10.9 in binary
Into word in memory

In computer still use 32 bit word

Interpret bits differently

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sign</td>
<td>Exponent</td>
<td>Number</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

First remember scientific notation for decimals from physics

If we have number 11.287593
We write as

0.11287593 \times 10^2

Our rule is to move the decimal point
To the left of the left most digit
Multiply by the appropriate amount

Same rule applies in binary

✓ Our rule is to move the binary point
   To the left of the left most 1 bit
✓ Multiply by the appropriate amount

Now can expressive power given as

\[ \pm (2^{23} - 1) \times 2^{\text{exp}(2^8 - 1)} \]

Someone recognized
Since always storing 1 as most significant bit – (msb)
Why store
Move 1 bit to right and store those

We now are storing 1 additional bit

Scheme now called IEEE Hidden Bit Format
Example
Let’s consider that we can only store 8 bits
For the binary number 1011.1101101

1. First put in standard format
   \[0.10111101101 \times 2^{+4}\]
2. Since we can only store 8 bits our number is
   \[0.10111101 \times 2^{+4}\]
   We loose the last 3 bits - 101
2a. Since we know that the msb will be a 1 instead we can store
   \[0.01111011 \times 2^{+4}\]
   We now have stored one extra bit of precision

By using hidden bit
Can increase to
\[\pm (2^{24} -1) \times 2^{\exp(28 -1)}\]

Characters
Start with 32 bit word

Since computer storing only 0’s and 1’s
How can we store an
Alphabetic character
Asian or Middle Eastern glyph

Use trick used to create secret code

For western alphabet
1. Write down all characters – assume 26
2. Give each character a corresponding number
   \[A = 0, B = 1, C = 2, \ldots\]

Can now write messages in our secret numeric code
No one will ever be able to figure out

Can use same trick inside of computer
1. Write down all characters
   Will now be many more than 26
   We need many special characters
2. Give each character a corresponding number
   \[A = 0, B = 1, C = 2, \ldots\]
3. Now write decimal number as 8 bit binary equivalent
We find that there are several standard secret codes available:
- ASCII
- EBDCIC
- BDCIC  
  Rarely any more

Expressing each character as 8 bit binary number – 1 byte
  Gives us ability to express 256 characters

If we use 16 bit numbers
  Can represent many more characters

32 bit word permits storage of 4 characters
  Approximately half
  Used to express
  Printing
  Non printing or control

In C declaration
  `char a = 'a';`
  Places ASCII value of character a
  Into word in memory
  Word in memory will be stored as
  \[00 00 00 0a_{\text{16}}\]
  For a total of 32 bits

Addresses
  Let’s now consider what else can be stored in a word

  That is what other interpretation can we place on same bits

In computer
  Information stored in memory
    Each piece placed at an address
  Information accessed in memory by giving its address
  Addresses begin at 0 and end at number based upon word size
    Recall with 16 bits
      Largest number  65K
      Range between 0 - 65K
      If those values represent addresses
        Can address upto 65K locations
    For a 32 bit word we have
      We can address approximately 4 M
In C or C++ declarations
Java doesn’t think they have address – they really do

```c
int a = 10;
int* aPtr = &a; // take the address of a and assign to the pointer variable aPtr
```

Place integer value 10 in binary
Into word in memory
Lets say at address 3000

Sets aside another memory word to hold address of a
Puts 3000 into that address
The value at address points to 3000

Summary
In this lesson we learned
✓ Different numeric bases
✓ How to represent information for a computer
  Unsigned and signed integers
  Characters
  Addresses
✓ Expressive power of different representations