What do computers do?
- Manipulate stored information
- Manipulation: operations ...
- More on this later
- Information is data: how is it represented?
- Basic information: numbers
- Human beings have represented numbers throughout history
- Roman numerals
- Decimal system

Number System
- Comprises of
  - Set of numbers or elements
  - Operations on them (+, -, /, *)
  - Rules that define properties of operations (identity, inverse...)
- Need to assign value to numbers
- Let us take decimal system
  - 1's place, 10's place, 100's place etc
  - Base 10
  - Value: \( \sum i \times 10^i \)
  - Humans use decimal

Binary numbers
- Base 2; each digit is 0 or 1
- Each bit in place \( i \) has value \( 2^i \)
- Binary representation is used in computers
- Easy to represent by switches (on/off)
- Manipulation by Digital logic in hardware
- But hard for humans to read
- \((13)_{10} = (1101)_{2}\)
Hexadecimal representation

- Binary hard to read for humans
  - Especially 16 bits, 32 bits, 64 bits
  - 1010101001010101
- Base 16 representation or hex representation
- Symbols = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}
- Value = \( \sum_i \times 16^i \)
- First 10 (0 through 9) symbols are same as decimal numbers
- A=10, B=11, C=12, D=13, E=14, F=15

Decimal to binary

```c
main()
{
  int N, i=0; int Maxbits=32; int bitarray[Maxbits];
  scanf("%d", &N);
  for (i=0; i<Maxbits; i++) bitarray[i]=0; /* initialize */
  i=0;
  while (N > 0){ /* MSB is bit 0, LSB is bit 31 */
    bitarray[Maxbits-1-i]=N%2; /* mod function */
    N=N/2;
    i++;
  }
  i=0; for (i=0; i<Maxbits; i++) printf("%d",bitarray[i]);
}
```

Same for Hex

- Example: Convert 10 in decimal to binary
  - The code for binary works for any base with modifications
- For hex, replace by N%16 and N=N/16 replace reminders > 9 by A, B, C, D, E and F
- Example: Convert 240 to HEX

Binary to decimal

- Convert 110110 to decimal
  - Value = \( i \times 2^i \)
  - Value = \( 2^5 2^4 2^2 2^1 \)
    - \( = 32 + 16 + 4 + 2 \)
    - \( = 54 \)
Converting Hex to binary

- Each digit in Hex can be represented by 4 bit binary (base 16) or nibble
- Convert 2A8C to binary
- 0010 1010 0100 1100

Convert Binary to hex

- Group binary bits into groups of four
- Replace each nibble by a hex digit
- Example 101101110011100
- 101101110011100
- B79C or 0xB79C
- In C, numeric constants starting with 0x are interpreted as being in hexadecimal

Examples

- 0x8F7A93 to binary
- 1011011110011100 to hex
- 0xC4E5 to decimal

Decimal and binary fractions

- In decimal, digits to the right of radix point have value 1/10^i for each digit in the i^{th} place
- 0.25 is 2/10 + 5/100
- Similarly, in binary, digits to the right of radix point have value 1/2^i for each i^{th} place after the decimal point.
- Binary Fractions are the same except the base is different
- 8.625 is 1000.101
Decimal to binary example

- 0.625 to binary
- ANS: 0.101
- 0.625*2 = 1.25
- output 1
- 0.25*2 = 0.5
- output 0
- 0.5*2 = 1
- output 1
- Exit

Algorithm
/* precondition: 0 < number < 1 */
number = decimal fraction
while [number > 0]
  number = number * 2 /* shift left */
  if (number >= 1) {Output result = 1; number = number - 1}
  else Output result = 0

--

Decimal to binary fractions

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.1</td>
</tr>
<tr>
<td>0.25</td>
<td>0.01</td>
</tr>
<tr>
<td>0.625</td>
<td>0.101</td>
</tr>
<tr>
<td>0.75</td>
<td>0.11</td>
</tr>
</tbody>
</table>

Data sizes

- All Information is represented in binary form but require different sizes
- Characters in 1 byte, integers 2 to 4 bytes, real numbers 4 to 8 bytes

<table>
<thead>
<tr>
<th>C declaration</th>
<th>32-bit machine</th>
<th>64-bit machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Short int</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>int</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>pointer</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>float</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>double</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

Big endian vs little endian

- A binary representation of a number in memory may require multiple bytes
- How to determine value of a sequence of bytes
- Most Significant byte first ... big endian
- Least Significant byte first ... little endian
- Depends on the type of machine
- Why we need to know:
  - Look at machine code
  - If we are to interpret bytes and determine value
  - One computer (big endian) sending data to another computer (small endian)
    - Need to convert into standard form before transmitting
Representing integers

- Signed integers
  - Unsigned or natural numbers
  - Directly represent as binary
  - What about negative numbers?
- Signed Magnitude
  - MSB is sign bit
  - 4 is 0100 -4 1100
  - 3 is 0011 -3 1011
  - 2 is 0010 -2 1010
  - 1 is 0001 -1 1001
  - 0 is 0000 ..what is 1000?
- Problems: two zeros +ve 0 and –ve 0
  - normal bit-wise addition does not work... -1 + 4

Representing Integers

- Divide the binary space into two halves
- One with leading 0s are +ve
- One with leading 1s are –ve
- This is called two’s complement representation
- Two bit binary numbers
  - 00 01 10 11
  - 0 1 -2 -1
- Three bit

2s complement advantages

- Adding two integers works as normal arithmetic
- Normal arithmetic works for 2’s complement (ignore carry)
- Only 1 zero
- X+ (-X) = 0
- Used in almost all computers

Negate a 2s complement

- invert the bits and add 1
  - X = 001 110 \rightarrow 111
- All 1s is -1, so add 1 to make it zero
- So, to get –X, invert all bits and add 1
Finding 2s complement

- Take the binary representation and invert all bits
  - Step 1: 0 to 1 and 1 to 0
  - Step 2: Then add 1
- Example: 00101 (5)
- (-5) is ............
- Find 2s complement of 9

Finding 2s complement

- Copy all bits from right to left until first 1
- Atlantic ocean rule!!
- At
- Then flip all bits
  - 0100 1
  - 1011 1
- Example: find 2s complement of 0100, 0110
  - 0100 0110

Decimal value of 2s complement

- Most significant bit has –ve value
- $X_{n-1}, X_{n-2}, \ldots, X_1, X_0$
- Value of MSB is $-2^{(n-1)}$
- Rest is same as +ve binary number
- 110011 ... what is the decimal value?

Decimal value of 2’s complement

- If leading bit or MSB is 1, take 2’s complement to get +ve number
- Determine decimal value of number
- And add –ve sign to it
- If leading bit or MSB is 0, compute as normal
- Example 1100110_{TWO}
**1s complement**

- Alternative representation
- Negative numbers represented by the complement
- Max numbers that can be represented is halved
- When adding negative numbers carry needs to be added to the result
  - `-3 + (-2)`
  - 1001
  - 101
  - 001 \(\rightarrow\) add carry
  - 1011
  - 0110 \(\rightarrow\) 2 but correct answer is (-5)
  - Overflow

**ASCII**

- Characters are also stored as bits
- 1 byte is 8 bits but MSB is used for error detection
- ASCII represents typical keys on a keyboard
- 7 bits is 128 different possibilities
- 7 bit ASCII included printable and non-printable characters
- Non-printable for controlling printer such as LF or linefeed

**ASCII table**

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:</td>
<td>1:</td>
<td>2:</td>
<td>3:</td>
<td>4:</td>
<td>5:</td>
<td>6:</td>
<td>7:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
</table>

**unicode**

- Extended to 16 bit characters
- Represent other characters (Japanese)
- Java supports this format
- In Unicode, the first 128 characters are ASCII
Floating point

- With integers range is limited
- short int 2 bytes $2^{16}$
- unsigned short int
  - Unsigned: Range is natural number 0 to 65536
  - Signed: 2's complement is -32768 to 32767
- for int (4 bytes) -2147483648 to 2147483647
- Can also be represented as magnitude and exponent
  - $2.147483647 \times 10^9$

Scientific notation

- $2.147483647 \times 10^9$
  - Number of digits in Mantissa $\rightarrow$ precision
  - Exponent gives range
  - A binary number can also be expressed in this form
  - $10111011 \rightarrow 1.0111011 \times 2^7$

Same number with varying exponent

- 10101 $\rightarrow$ 21 ….. In decimal
- 1010.1 $\times 2 \rightarrow 10.5 \times 2$
- 101.01 $\times 2^2 \rightarrow 5.25 \times 4$
- 10.101 $\times 2^3 \rightarrow 2.625 \times 8$
- Too many ways to represent the same number
- Need a standard representation

IEEE floating point standard

- Most computers follow IEEE 754 standard
- Single precision (32 bits)
- Double precision (64 bits)
- Extended precision (80 bits)
**Floating point in C**

- 32 bits, single precision or float
- 1 sign bit, 23 bits for mantissa, 8 bits for exponent
- Exponent is power of 2
- Signed magnitude
  - Sign bit is 1 for –ve numbers, sign bit is 0 for +ve numbers
  - Exponent has 8 bits
  - 0 to 255 or -128 to +127
  - \(2^{127}\) is approx \(10^{38}\)
- Range is \(-3.4 \times 10^{38}\) to \(+3.4 \times 10^{38}\)

**Exponent bias**

- The binary exponent is represented by adding a bias of 127
- Saves an extra bit for sign bit
- Note: it is not 2s complement
- \(2^3\) is \(2^{130}\) which is 2 \(^{10000010}\)
- \(2^5\) is \(2^{127}\) which is \(2^{01111111}\)
- \(2^{-3}\) is \(2^{124}\) which is \(2^{01111100}\)

**Normalization**

- The exponent value is adjusted so that the mantissa has a 1 before the radix point
- 0.25 is \(0.01 \times 2^0\) adjusted to \(1 \times 2^{-2}\)
- 4.625 is \(100.101 \times 2^0\) adjusted to \(1.00101 \times 2^2\)
- 64 is \(1000000 \times 2^6\) adjusted \(1 \times 2^6\)
- In this way, the mantissa always has 1 digit with value 1
- This can be omitted to save a bit!!!

**Decimal to floating point**

- 5.625
- In binary
  - 101.101 \(\rightarrow 1.01101 \times 2^2\)
- Exponent field has value 2
  - add 127 to get 129
- Exponent is 1000001
- Mantissa is 01101
- Sign bit is 0
  - 0 01101000000000000000000 1000001
One more example

- Convert 12.375 to floating point representation
- Binary is 1100.011

Floating point to decimal

What is the decimal value of the above?

Exponent is \((10000010) - 127\)

Mantissa is 1.01001

Sign bit is -1

\[(1 - 2s) \times (1 + f) \times 2^{e-bias}\]

Special cases

- 0 0000000 00000000000000000000000000000000
- All zeros 0 (+0)
- 1 0000000 00000000000000000000000000000000
- All zeros with sign bit is -0 (-0)
- 0 1111111 00000000000000000000000000000000
- Note: the minimum value in the exponent after adding the bias should be 1. Hence, the minimum value for the exponent in single precision is \(-126\) \((=1-127\text{ or }x1-x7F)\)
- \(+\) Infinity
- 1 111111 00000000000000000000000000000000
- \(-\)Infinity
- Note: the maximum value in the exponent after adding the bias is 254. Hence, the maximum value for the exponent in single precision is \(+127\) \((=254-127\text{ or }x7E-x7F)\)

Extended precision

- 80 bits used to represent a real number
- 1 sign bit, 15 bit exponent, 64 bit mantissa
- 20 decimal digits of accuracy
- \(10^{-4932}\) to \(10^{4932}\)
- Not supported in C