DATA REPRESENTATION

Data Types

Complements

Fixed Point Representations

Floating Point Representations

Other Binary Codes

Error Detection Codes

DATA REPRESENTATION

Information that a Computer is dealing with

- *** Data**
  - **- Numeric Data**
    - **Numbers(Integer, real)**
  - **- Non-numeric Data**
    - **Letters, Symbols**

- *** Relationship between data elements**
  - **- Data Structures**
    - **Linear Lists, Trees, Rings, etc**

- *** Program(Instruction)**

NUMERIC DATA REPRESENTATION

Data

Numeric data - numbers(integer, real)
Non-numeric data - symbols, letters

Number System

Nonpositional number system
- **Roman number system**

Positional number system
- **Each digit position has a value called a *weight* associated with it**
- **Decimal, Octal, Hexadecimal, Binary**

Base (or radix) R number
- **Uses R distinct symbols for each digit**
- **Example** $A_R = a_{n-1}\, a_{n-2} \ldots a_1\, a_0\, .a_{-1} \ldots a_{-m}$

Radix point(.) separates the integer portion and the fractional portion

R = 10	**Decimal number system,**		**R = 2**	**Binary**
R = 8	**Octal,**		**R = 16**	**Hexadecimal**

WHY POSITIONAL NUMBER SYSTEM IN DIGITAL COMPUTERS ?

Major Consideration is the *COST* and *TIME*

- **Cost of building *hardware***
  Arithmetic and Logic Unit, CPU, Communications
- **Time to processing**

Arithmetic - Addition of Numbers - *Table for Addition*

*** Non-positional Number System**
- **Table for addition is infinite**
  --> Impossible to build, very expensive even if it can be built

*** Positional Number System**
- **Table for Addition is finite**
  --> Physically realizable, but cost wise the smaller the table size, the less expensive --> Binary is favorable to Decimal

Binary Addition Table

	0	1
0	0	1
1	1	10

Decimal Addition Table

	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	10
2	2	3	4	5	6	7	8	9	10	11
3	3	4	5	6	7	8	9	10	11	12
4	4	5	6	7	8	9	10	11	12	13
5	5	6	7	8	9	10	11	12	13	14
6	6	7	8	9	10	11	12	13	14	15
7	7	8	9	10	11	12	13	14	15	16
8	8	9	10	11	12	13	14	15	16	17
9	9	10	11	12	13	14	15	16	17	18

REPRESENTATION OF NUMBERS - POSITIONAL NUMBERS

Decimal	Binary	Octal	Hexadecimal
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Binary, octal, and hexadecimal conversion

1	2	7	5	4	3	**Octal**
1	010	111	101	100	011	**Binary**
	A	F	6	3		**Hexa**

1 0 1 0 1 1 1 1 0 1 1 0 0 0 1 1

CONVERSION OF BASES

Base R to Decimal Conversion

$$A = a_{n-1}\ a_{n-2}\ \ a_{n-3} \ldots a_0 \,.\, a_{-1} \ldots a_{-m}$$

$$V(A) = \Sigma\, a_k\, R^k$$

$(736.4)_8 = 7 \text{ x } 8^2 + 3 \text{ x } 8^1 + 6 \text{ x } 8^0 + 4 \text{ x } 8^{-1}$

$\qquad = 7 \text{ x } 64 + 3 \text{ x } 8 + 6 \text{ x } 1 + 4/8 = (478.5)_{10}$

$(110110)_2 = \ldots = (54)_{10}$

$(110.111)_2 = \ldots = (6.785)_{10}$

$(F3)_{16} = \ldots = (243)_{10}$

$(0.325)_6 = \ldots = (0.578703703 \ldots\ldots\ldots\ldots\ldots)_{10}$

Decimal to Base R number

- Separate the number into its *integer* and *fraction* parts and convert each part separately.
- Convert *integer part* into the base R number
  → successive divisions by R and accumulation of the remainders.
- Convert *fraction part* into the base R number
  → successive multiplications by R and accumulation of integer digits

EXAMPLE

Convert 41.6875_{10} to base 2.

Integer = 41

41	
20	1
10	0
5	0
2	1
1	0
0	1

$(41)_{10} = (101001)_2$

Fraction = 0.6875

$$\begin{array}{r} 0.6875 \\ \times \quad 2 \\ \hline 1.3750 \\ \times \quad 2 \\ \hline 0.7500 \\ \times \quad 2 \\ \hline 1.5000 \\ \times \quad 2 \\ \hline 1.0000 \end{array}$$

$(0.6875)_{10} = (0.1011)_2$

$$(41.6875)_{10} = (101001.1011)_2$$

Exercise

Convert $(63)_{10}$ to base 5: $(223)_5$
Convert $(1863)_{10}$ to base 8: $(3507)_8$
Convert $(0.63671875)_{10}$ to hexadecimal: $(0.A3)_{16}$

COMPLEMENT OF NUMBERS

Two types of complements for base R number system:

- R's complement and (R-1)'s complement

The (R-1)'s Complement

Subtract each digit of a number from (R-1)

Example

- 9's complement of 835_{10} is 164_{10}

- 1's complement of 1010_2 is 0101_2(bit by bit complement operation)

The R's Complement

Add 1 to the low-order digit of its (R-1)'s complement

Example

- 10's complement of 835_{10} is 164_{10} + 1 = 165_{10}

- 2's complement of 1010_2 is 0101_2 + 1 = 0110_2

FIXED POINT NUMBERS

Numbers: Fixed Point Numbers and Floating Point Numbers

Binary Fixed-Point Representation

$X = x_n x_{n-1} x_{n-2} \ldots x_1 x_0 .\ x_{-1} x_{-2} \ldots x_{-m}$

Sign Bit(x_n): 0 for positive - 1 for negative

Remaining Bits($x_{n-1} x_{n-2} \ldots x_1 x_0 .\ x_{-1} x_{-2} \ldots x_{-m}$)

SIGNED NUMBERS

Need to be able to represent both *positive* and *negative* numbers

- Following 3 representations

Signed magnitude representation
Signed 1's complement representation
Signed 2's complement representation

Example: Represent +9 and -9 in 7 bit-binary number

Only one way to represent +9 ==> 0 001001
Three different ways to represent -9:

In signed-magnitude: 1 001001
In signed-1's complement: 1 110110
In signed-2's complement: 1 110111

In general, in computers, fixed point numbers are represented either integer part only or fractional part only.

2's COMPLEMENT REPRESENTATION WEIGHTS

- **Signed 2's complement representation follows a "weight" scheme similar to that of unsigned numbers**
  - **Sign bit has negative weight**
  - **Other bits have regular weights**

$$X = x_n\ x_{n-1}\ \dots\ x_0$$

$$\Rightarrow\ V(X) = -\,x_n \times 2^n + \sum_{i=0}^{n-1} x_i \times 2^i$$

ARITHMETIC ADDITION: SIGNED MAGNITUDE

[1] Compare their signs
[2] If two signs are the *same* ,
ADD* the two magnitudes - Look out for an *overflow
[3] If *not the same* , compare the relative magnitudes of the numbers and then *SUBTRACT* the smaller from the larger --> need a subtractor to add
[4] Determine the sign of the result

6 + 9

```
    6    0110
+)  9    1001
-----------------
   15    1111 -> 01111
```

-6 + 9

```
    9    1001
-)  6    0110
-----------------
    3    0011 -> 00011
```

6 + (- 9)

```
    9   1001
-)  6   0110
-----------------
  - 3   0011 -> 10011
```

-6 + (-9)

```
    6   0110
+)  9   1001
-----------------
  -15   1111 -> 11111
```

Overflow **9 + 9 or (-9) + (-9)**

```
          9     1001
+)        9     1001
-----------------------
overflow  (1)0010
```

ARITHMETIC ADDITION: SIGNED 2's COMPLEMENT

Add the two numbers, including their sign bit, and discard any carry out of leftmost (sign) bit - Look out for an *overflow*

Example

```
     6   0  0110
+)   9   0  1001
    15   0  1111
```

```
    -6   1  1010
+)   9   0  1001
     3   0  0011
```

```
     6   0  0110
+)  -9   1  0111
    -3   1  1101
```

```
    -9      1  0111
+)  -9      1  0111
   -18   (1)0  1110
```

$x'_{n-1}y'_{n-1}s_{n-1}$
$(c_{n-1} \oplus c_n)$

overflow

```
     9   0  1001
+)   9   0  1001
    18   1  0010
```

$x_{n-1}y_n s'_{n-1}$
$(c_{n-1} \oplus c_n)$

2 operands have the same sign
and the result sign changes

$x_{n-1}y_{n-1}s'_{n-1} + x'_{n-1}y'_{n-1}s_{n-1} = c_{n-1} \oplus c_n$

ARITHMETIC ADDITION: SIGNED 1's COMPLEMENT

Add the two numbers, including their sign bits.

- If there is a carry out of the most significant (sign) bit, the result is incremented by 1 and the carry is discarded.

Example

```
        6    0  0110
+)     -9    1  0110
-----------------------
       -3    1  1100
```

end-around carry

```
       -6      1     1001
+)      9      0     1001
----------------------------
       (1) 0(1)0010
+)                      1
----------------------------
        3      0     0011
```

not overflow $(c_{n-1} \oplus c_n) = 0$

```
       -9    1  0110
+)     -9    1  0110
-----------------------
          (1)0  1100
+)                 1
-----------------------
             0  1101
```

```
        9   0      1001
+)      9   0      1001
--------------------------
            1  (1)0010
```

overflow
$(c_{n-1} \oplus c_n)$

COMPARISON OF REPRESENTATIONS

* Easiness of negative conversion

  S + M > 1's Complement > 2's Complement

* Hardware

  - S+M: Needs an adder and a subtractor for Addition
  - 1's and 2's Complement: Need only an adder

* Speed of Arithmetic

  2's Complement > 1's Complement(end-around C)

* Recognition of Zero

  2's Complement is fast

ARITHMETIC SUBTRACTION

Arithmetic Subtraction in 2's complement

Take the complement of the subtrahend (including the sign bit) and add it to the minuend including the sign bits.

$$(\pm A) - (- B) = (\pm A) + B$$

$$(\pm A) - B = (\pm A) + (- B)$$

FLOATING POINT NUMBER REPRESENTATION

* The location of the fractional point is not fixed to a certain location
* The range of the representable numbers is wide

$$F = EM$$

m_n	$e_k e_{k-1} \ldots e_0$	$m_{n-1} m_{n-2} \ \ldots \ m_0 \,.\, m_{-1} \ \ldots \ m_{-m}$
sign	exponent	mantissa

- Mantissa
  Signed fixed point number, either an integer or a fractional number

- Exponent
  Designates the position of the radix point

Decimal Value

$$V(F) = V(M) * R^{V(E)}$$

M: Mantissa
E: Exponent
R: Radix

FLOATING POINT NUMBERS

Example

sign		sign	
0	.1234567	0	04
mantissa		exponent	

$$\Longrightarrow +.1234567 \times 10^{+04}$$

Note:

In Floating Point Number representation, only Mantissa(M) and Exponent(E) are explicitly represented. The Radix(R) and the position of the Radix Point are implied.

Example

A binary number +1001.11 in 16-bit floating point number representation (6-bit exponent and 10-bit fractional mantissa)

	Sign	Exponent	Mantissa
	0	0 00100	100111000
or	0	0 00101	010011100

CHARACTERISTICS OF FLOATING POINT NUMBER REPRESENTATIONS

Normal Form

- There are many different floating point number representations of the same number
  → Need for a unified representation in a given computer

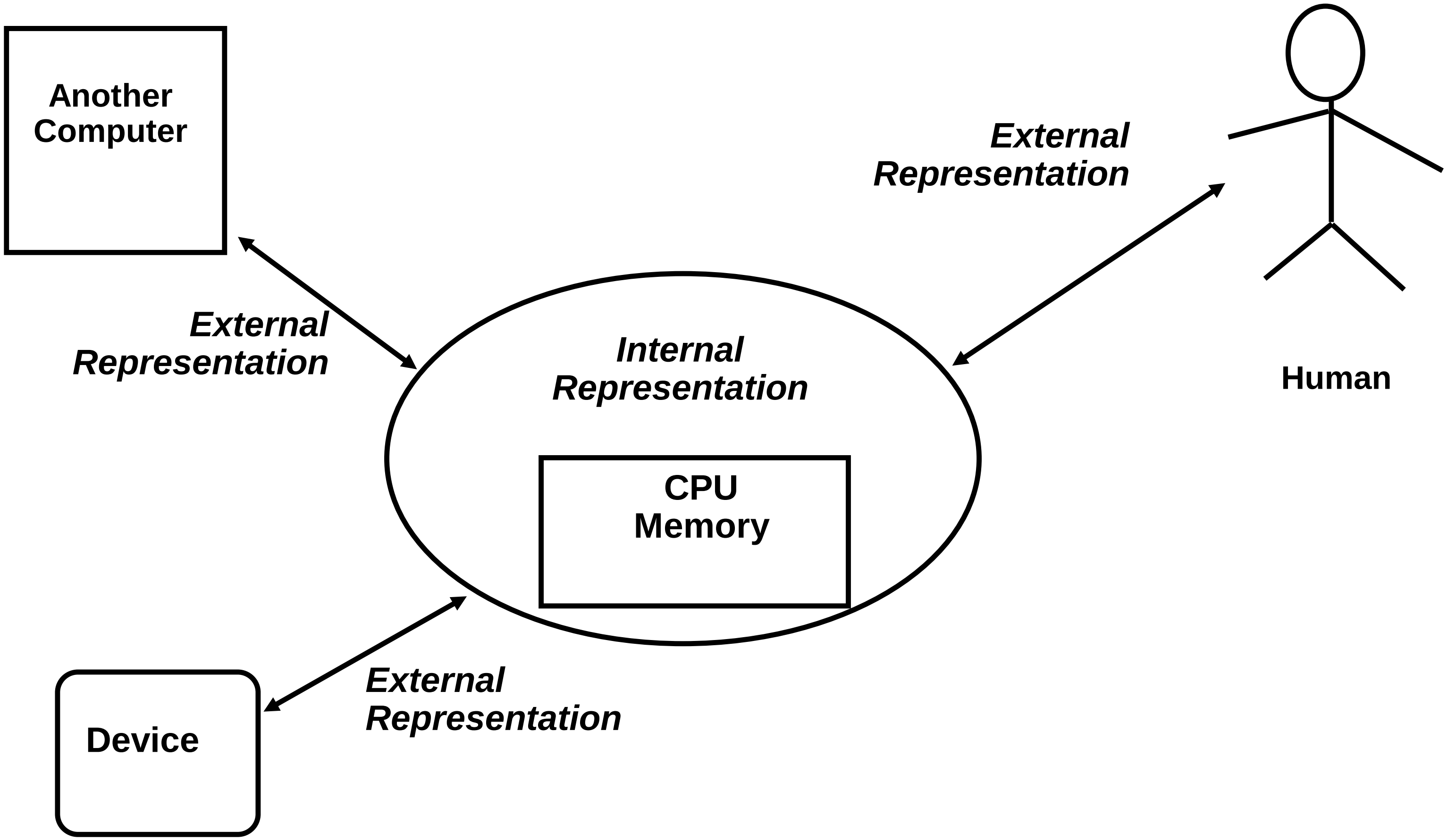
- *the most significant position of the mantissa contains a non-zero digit*

Representation of Zero

- Zero
  Mantissa = 0

- Real Zero
  Mantissa = 0
  Exponent
  = smallest representable number
  which is represented as
  00 ... 0
  ← Easily identified by the hardware

INTERNAL REPRESENTATION AND EXTERNAL REPRESENTATION



EXTERNAL REPRESENTATION

Numbers

Most of numbers stored in the computer are eventually changed by some kinds of calculations

→ *Internal Representation* for calculation efficiency

→ Final results need to be converted to as *External Representation* for presentability

Alphabets, Symbols, and some Numbers

Elements of these information do not change in the course of processing

→ No needs for Internal Representation since they are not used for calculations

→ External Representation for processing and presentability

Example

Decimal Number: 4-bit Binary Code
BCD(Binary Coded Decimal)

Decimal	BCD Code
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

OTHER DECIMAL CODES

Decimal	BCD(8421)	2421	84-2-1	Excess-3
0	0000	0000	0000	0011
1	0001	0001	0111	0100
2	0010	0010	0110	0101
3	0011	0011	0101	0110
4	0100	0100	0100	0111
5	0101	1011	1011	1000
6	0110	1100	1010	1001
7	0111	1101	1001	1010
8	1000	1110	1000	1011
9	1001	1111	1111	1100

Note: 8,4,2,-2,1,-1 in this table is the weight associated with each bit position.

$d_3\ d_2\ d_1\ d_0$: **symbol in the codes**

BCD: $d_3 \times 8 + d_2 \times 4 + d_1 \times 2 + d_0 \times 1$
$\Rightarrow$ **8421 code.**
2421: $d_3 \times 2 + d_2 \times 4 + d_1 \times 2 + d_0 \times 1$
84-2-1: $d_3 \times 8 + d_2 \times 4 + d_1 \times (-2) + d_0 \times (-1)$
Excess-3: BCD + 3

BCD: It is difficult to obtain the 9's complement.
However, it is easily obtained with the other codes listed above.
$\rightarrow$ **Self-complementing codes**

GRAY CODE

* Characterized by having their representations of the binary integers differ in only one digit between consecutive integers

* Useful in some applications

4-bit Gray codes

Decimal number	Gray g_3	g_2	g_1	g_0	Binary b_3	b_2	b_1	b_0
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	1	0	0	1	0
3	0	0	1	0	0	0	1	1
4	0	1	1	0	0	1	0	0
5	0	1	1	1	0	1	0	1
6	0	1	0	1	0	1	1	0
7	0	1	0	0	0	1	1	1
8	1	1	0	0	1	0	0	0
9	1	1	0	1	1	0	0	1
10	1	1	1	1	1	0	1	0
11	1	1	1	0	1	0	1	1
12	1	0	1	0	1	1	0	0
13	1	0	1	1	1	1	0	1
14	1	0	0	1	1	1	1	0
15	1	0	0	0	1	1	1	1

CHARACTER REPRESENTATION ASCII

ASCII (American Standard Code for Information Interchange) Code

MSB (3 bits)

LSB (4 bits)

<table>
<tr><th></th><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>
<tr><td>0</td><td>NUL</td><td>DLE</td><td>SP</td><td>0</td><td>@</td><td>P</td><td>‘</td><td>P</td></tr>
<tr><td>1</td><td>SOH</td><td>DC1</td><td>!</td><td>1</td><td>A</td><td>Q</td><td>a</td><td>q</td></tr>
<tr><td>2</td><td>STX</td><td>DC2</td><td>“</td><td>2</td><td>B</td><td>R</td><td>b</td><td>r</td></tr>
<tr><td>3</td><td>ETX</td><td>DC3</td><td>#</td><td>3</td><td>C</td><td>S</td><td>c</td><td>s</td></tr>
<tr><td>4</td><td>EOT</td><td>DC4</td><td>$</td><td>4</td><td>D</td><td>T</td><td>d</td><td>t</td></tr>
<tr><td>5</td><td>ENQ</td><td>NAK</td><td>%</td><td>5</td><td>E</td><td>U</td><td>e</td><td>u</td></tr>
<tr><td>6</td><td>ACK</td><td>SYN</td><td>&</td><td>6</td><td>F</td><td>V</td><td>f</td><td>v</td></tr>
<tr><td>7</td><td>BEL</td><td>ETB</td><td>‘</td><td>7</td><td>G</td><td>W</td><td>g</td><td>w</td></tr>
<tr><td>8</td><td>BS</td><td>CAN</td><td>(</td><td>8</td><td>H</td><td>X</td><td>h</td><td>x</td></tr>
<tr><td>9</td><td>HT</td><td>EM</td><td>)</td><td>9</td><td>I</td><td>Y</td><td>I</td><td>y</td></tr>
<tr><td>A</td><td>LF</td><td>SUB</td><td>*</td><td>:</td><td>J</td><td>Z</td><td>j</td><td>z</td></tr>
<tr><td>B</td><td>VT</td><td>ESC</td><td>+</td><td>;</td><td>K</td><td>[</td><td>k</td><td>{</td></tr>
<tr><td>C</td><td>FF</td><td>FS</td><td>,</td><td><</td><td>L</td><td>\</td><td>l</td><td>|</td></tr>
<tr><td>D</td><td>CR</td><td>GS</td><td>-</td><td>=</td><td>M</td><td>]</td><td>m</td><td>}</td></tr>
<tr><td>E</td><td>SO</td><td>RS</td><td>.</td><td>></td><td>N</td><td>m</td><td>n</td><td>~</td></tr>
<tr><td>F</td><td>SI</td><td>US</td><td>/</td><td>?</td><td>O</td><td>n</td><td>o</td><td>DEL</td></tr>
</table>

ERROR DETECTING CODES

Parity System

- **Simplest method for error detection**
- **One *parity* bit attached to the information**
- ***Even Parity* and *Odd Parity***

Even Parity

- **One bit is attached to the information so that the total number of 1 bits is an even number**

1011001 $\boxed{0}$
1010010 $\boxed{1}$

Odd Parity

- **One bit is attached to the information so that the total number of 1 bits is an odd number**

1011001 $\boxed{1}$
1010010 $\boxed{0}$

PARITY BIT GENERATION

Parity Bit Generation

For $b_6 b_5 \ldots b_0$ (7-bit information); even parity bit b_{even}

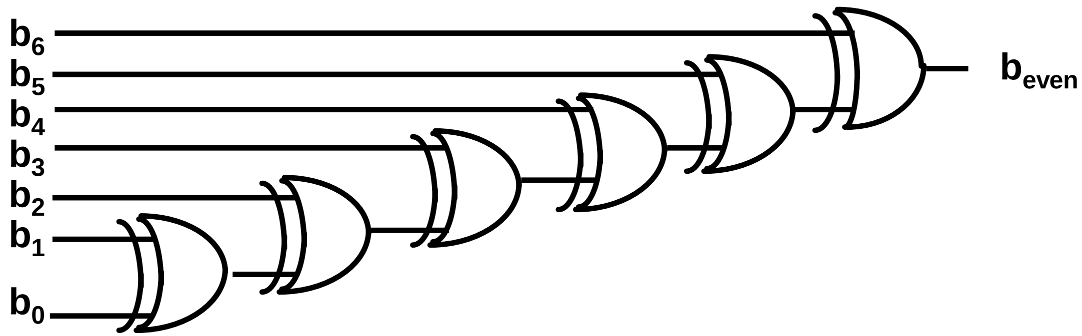
$$b_{even} = b_6 \oplus b_5 \oplus \ldots \oplus b_0$$

For odd parity bit

$$b_{odd} = b_{even} \oplus 1 = \overline{b}_{even}$$

PARITY GENERATOR AND PARITY CHECKER

Parity Generator Circuit (even parity)



Parity Checker