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** $A_R = a_{n-1}\, a_{n-2} \ldots a_1\, a_0\, .a_{-1} \ldots a_{-m}$
- $V(A_R) = \sum_{i=-m}^{n-1} a_i R^i$

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

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

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

Octal	1	2	7	5	4	3
Binary	1	010	111	101	100	011

Binary	1010	1111	0110	0011
Hexa	A	F	6	3

CONVERSION OF BASES

Base R to Decimal Conversion

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

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

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

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

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

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

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

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

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

$$\begin{array}{l} 0.6875 \\ \underline{\times \quad 2} \\ 1.3750 \\ \underline{\times \quad 2} \\ 0.7500 \\ \underline{\times \quad 2} \\ 1.5000 \\ \underline{\times \quad 2} \\ 1.0000 \end{array}$$

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

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

<u>Exercise</u>

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

CHARACTERISTICS OF 3 DIFFERENT REPRESENTATIONS

Complement

Signed magnitude: Complement *only* the sign bit
Signed 1's complement: Complement *all* the bits including sign bit
Signed 2's complement: Take the 2's complement of the number, *including* its sign bit.

Maximum and Minimum Representable Numbers and Representation of Zero

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

Signed Magnitude

Max:	$2^n - 2^{-m}$	011 ... 11.11 ... 1
Min:	$-(2^n - 2^{-m})$	111 ... 11.11 ... 1
Zero:	+0	000 ... 00.00 ... 0
	-0	100 ... 00.00 ... 0

Signed 1's Complement

Max:	$2^n - 2^{-m}$	011 ... 11.11 ... 1
Min:	$-(2^n - 2^{-m})$	100 ... 00.00 ... 0
Zero:	+0	000 ... 00.00 ... 0
	-0	111 ... 11.11 ... 1

Signed 2's Complement

Max:	$2^n - 2^{-m}$	011 ... 11.11 ... 1
Min:	-2^n	100 ... 00.00 ... 0
Zero:	0	000 ... 00.00 ... 0

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

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	$e_k e_{k-1} \dots e_0$	$m_{n-1} m_{n-2} \ \dots \ m_0 \, . \, m_{-1} \ \dots \ m_{-m}$
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	

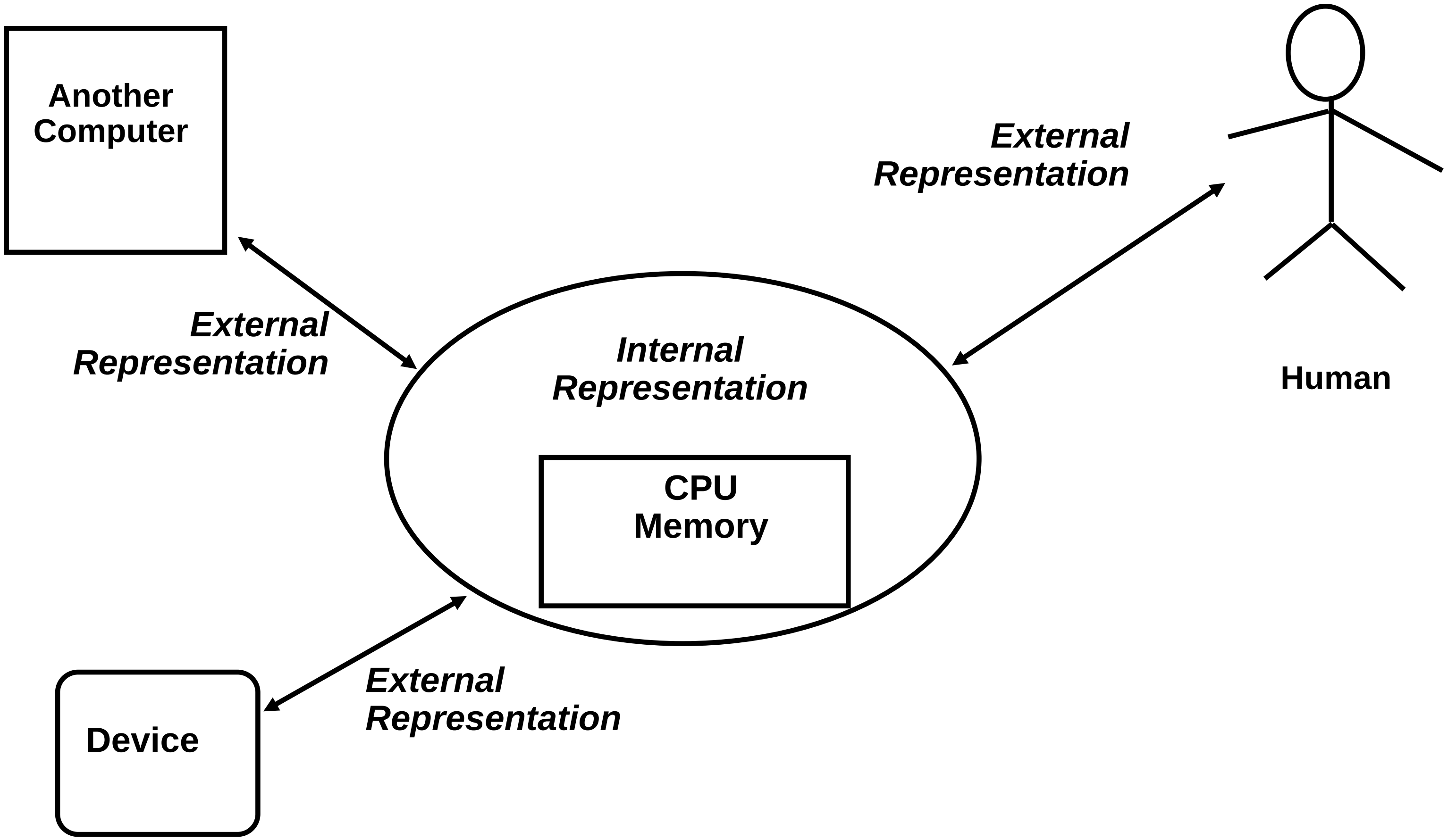
==> $+.1234567 \times 10^{+04}$

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

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

CHARACTER REPRESENTATION ASCII

ASCII (American Standard Code for Information Interchange) Code

MSB (3 bits)

LSB (4 bits)

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

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 $b_6b_5 \ldots b_0$(7-bit information); even parity bit b_{even}

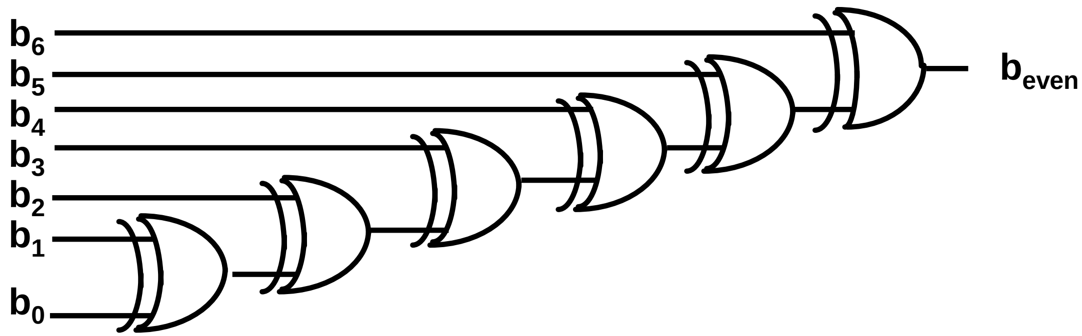
$$b_{even} = b_6 \oplus b_5 \oplus \ldots \oplus b_0$$

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