

Chapter Three

Data Representation and Computer Arithmetic

- Number Systems and Conversion
- Units of Data Representation
- Coding Methods
- Binary Arithmetic
- Complements
- Fixed and Floating points representation
- (Boolean Algebra) and Logic Circuits

Number systems and conversion

- Number Systems
 - Decimal
 - Binary
 - Octal
 - Hexadecimal
- Conversion

Decimal systems

- **The decimal system**

- Base 10 with ten distinct digits (0, 1, 2, ..., 9)
- Any number greater than 9 is represented by a combination of these digits
- The weight of a digit based on power of 10

Example:

- The number 81924 is actually the sum of:
 $(8 \times 10^4) + (1 \times 10^3) + (9 \times 10^2) + (2 \times 10^1) + (4 \times 10^0)$

Binary systems

Computers use the binary system to store and compute numbers.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
0 or 1	0 or 1	0 or 1	0 or 1	0 or 1	0 or 1	0 or 1	0 or 1

To represent any decimal number using the binary system, each place is simply assigned a value of either 0 or 1. To convert binary to decimal, simply add up the value of each place.

Example:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	0	0	1	1	0	0	1
128	0	0	16	8	0	0	1

128 + 0 + 0 + 16 + 8 + 0 + 0 + 1 = 153

10011001 = 153

Binary systems

- **The binary system (0 & 1)**

- The two digits representation is called a binary system
- Two electrical states – **on (1) & off (0)**
- The position weights are based on the **power of 2**
- The various combination of the two digits representation gives us the final value

Examples :

- i) 1011011 in binary = 91 in decimal
- ii) 1101.01 in binary = 13.25 in decimal

Binary Fractions

Binary fractions can also be represented:

Position Value: 2^{-1} 2^{-2} 2^{-3} 2^{-4} 2^{-5} etc.

Fractions: $\frac{1}{2}$ $\frac{1}{4}$ $\frac{1}{8}$ $\frac{1}{16}$ $\frac{1}{32}$

Decimal: .5 .25 .125 .0625 .03125

Binary into Decimal conversion

5th 4th 3rd 2nd 1st 0th

$$\begin{aligned} 1\ 0\ 1\ 1\ 1\ 1_2 &= (1 \times 2^5) + (0 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) \\ &= 32 + 8 + 4 + 2 + 1 \\ &= 47_{10} \end{aligned}$$

$$1011001_2 = 89_{10}$$

Exercise :

Convert the following binary numbers into their decimal equivalent

- $1110100_2 = (?)_{10}$
- $101101.1101_2 = (?)_{10}$

Conversion of Decimal to Binary

Divide by 2 (**remainder division**) till the dividend is zero and read remainders in reverse order. The right column shows result of integer division

Read answer in this direction, **Write** answer left to right

mod 2	637/2
1	318
0	159
1	79
1	39
1	19
1	9
1	4
0	2
0	1
1	0

$$637_{10} = 1001111101_2$$

➤ Convert 789_{10} to base 2

To Binary Fractions - Conversions

Multiply by 2 till enough digits are obtained, say 8, or a product is zero.

Read answer
in this direction
write it left
to right

$$\begin{array}{r} \overline{.637_{10}} \\ 1.274 \\ 0.548 \\ 1.096 \\ 0.192 \\ 0.384 \\ 0.768 \\ 1.536 \\ 1.072 \end{array}$$

Ans= 0.10100011₂

➤ Convert 0.325₁₀ to base 2

Octal system

- Octal system
 - Base 8 systems (0, 1, 2, ..., 7)
 - Used to give shorthand ways to deal with the long strings of 1 & 0 created in binary
 - Numbers 0 .. 7 can be represented by three binary digits

❖ Examples :-

i) $(3137)_8 = 1631_{10}$

ii) 134 in octal = 1011100 in binary

iii) $(6)_8 = (110)_2$

iv) 432.2 in octal = 282.25 in decimal

v) 123.45 in octal = 001010011.100101 in binary

Hexadecimal systems

- The Hexadecimal system
 - Base 16 system
 - 0 .. 9 and letters A .. F for sixteen place holders needed
 - A = 10, B = 11, ..., F = 15
 - Used in programming as a short cut to the binary number systems
 - Can be represented by four binary digits

❖ Examples :-

i) $(1D7F)_{16} = 7551_{10}$

ii) 6B2 in hexadecimal = 011010110010 in binary

iii) 101000010111 in binary = A17 in hexadecimal

Exercise – Convert ...

Decimal	Binary	Octal	Hexa- decimal
29.8			
	101.1101		
		3.07	
			C.82

Exercise – Convert ...

Answer

Decimal	Binary	Octal	Hexa- decimal
29.8	11101.110011...	35.63...	1D.CC...
5.8125	101.1101	5.64	5.D
3.109375	11.000111	3.07	3.1C
12.5078125	1100.10000010	14.404	C.82

Bits

- How many bits does a computer use to store an integer?
 - Some models = 32 bits
 - Other models = 64 bits
- What if we try to compute or store a larger integer?
 - If we try to compute a value larger than the computer can store, we get an arithmetic overflow error.

Representing Unsigned Integers

- How does a 16-bit computer represent the value 14?

0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

- What is the largest 16-bit integer?

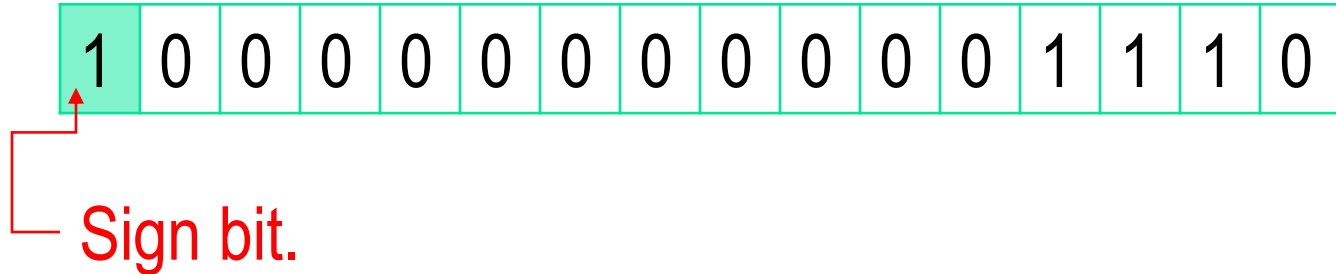
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

$$= 1 \times 2^{15} + 1 \times 2^{14} + \dots + 1 \times 2^1 + 1 \times 2^0 = 65,535$$

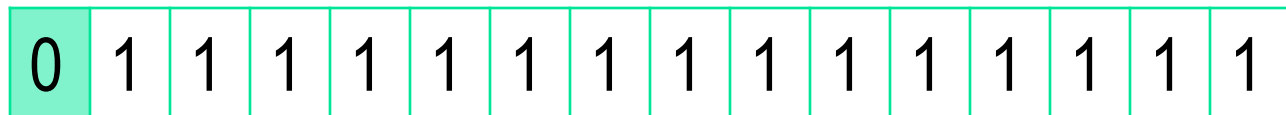
If we just add 1 to this number, for example, overflow will occur. We will get a wrong result.

Representing Signed Integers

- How does a 16 bit computer represent the value -14?



- What is the largest 16-bit signed integer?



$$= 1 \times 2^{14} + 1 \times 2^{13} + \dots + 1 \times 2^1 + 1 \times 2^0 = 32,767$$

- Problem → the value 0 is represented twice!
 - Most computers use a different representation, called two's complement.

Signed-magnitude representation

- Also called, “sign-and-magnitude representation”
- A number consists of a magnitude and a symbol representing the sign
- Usually 0 means positive, 1 negative
 - Sign bit
 - The number is represented with 1 sign bit to the left, followed by magnitude bits

Machine arithmetic with signed-magnitude representation

- Takes several steps to add a pair of numbers
 - Examine signs of the addends
 - If same, add magnitudes and give the result the same sign as the operands
 - If different, must...
 - Compare magnitude of the two operands
 - Subtract smaller number from larger
 - Give the result the sign of the larger magnitude operand
 - If magnitudes are equal and sign is different; two representations of zero problem
- For this reason the signed-magnitude representation is not as popular as one might think because of its “naturalness”

Complement number systems

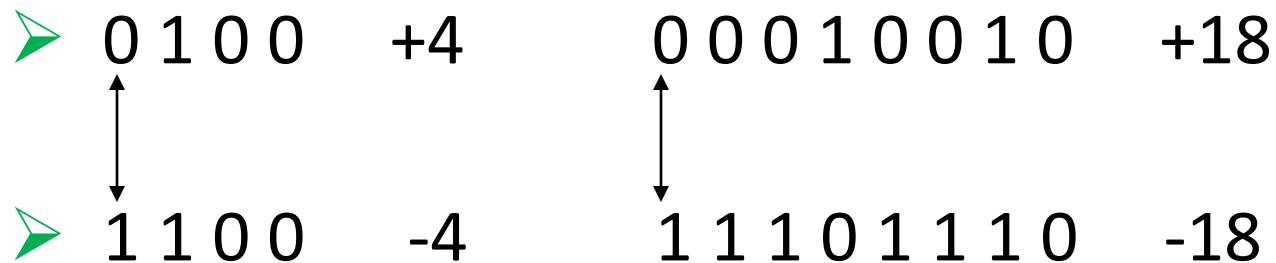
- Negates a number by *taking its complement* instead of negating the sign
- Not natural for humans, but better for machine arithmetic
- Will describe 2 complement number systems
 - *Radix complement* – very popular in real computers
 - Must first decide how many bits to represent the number – say n .
 - Complement of a number = $r^n - \text{number}$
 - **Example: 2's Complement**
 - *Diminished radix-complement* – not very useful, may skip it or not spend much time on it
 - **Example: 1's Complement**: $r^n - \text{number} - 1$

Two's-complement representation

- Just **radix-complement** when **radix = 2**
- The most used representation of integers in **computers** and other **digital arithmetic circuits**
- **0** and **positive numbers**: leftmost bit = 0
- **Negative numbers**: leftmost bit = 1
- Representation of zero
 - i.e. **0** is represented as **0000** using **4-bit** binary sequence.
- To find a number's **2's-complement** — just flip all the bits and **add 1**

Properties of Two's Complement Notation

- Relationship between $+n$ and $-n$.



Two's Complement Notation with 4-bits

Binary Pattern

Value in 2's complement.

0 1 1 1

7

0 1 1 0

6

0 1 0 1

5

0 1 0 0

4

0 0 1 1

3

0 0 1 0

2

0 0 0 1

1

0 0 0 0

0

1 1 1 1

-1

1 1 1 0

-2

1 1 0 1

-3

1 1 0 0

-4

1 0 1 1

-5

1 0 1 0

-6

1 0 0 1

-7

1 0 0 0

-8

Advantages of Two's Complement Notation

- It is easy to add two numbers.

$$\begin{array}{r} 0001 \quad +1 \\ + 0101 \quad +5 \\ \hline 0110 \quad +6 \end{array}$$

$$\begin{array}{r} 1000 \quad -8 \\ + 0101 \quad +5 \\ \hline 1101 \quad -3 \end{array}$$

- Subtraction is 2's complement addition
- Multiplication is just a repeated addition
- Division is just a repeated 2's complement addition
- Two's complement is widely used in *ALU*

Comparison of decimal and 4-bit numbers

Complements and other Notations

<i>Decimal</i>	<i>Two's Complement</i>	<i>Ones' Complement</i>	<i>Signed Magnitude</i>	<i>Excess 2^{m-1}</i>
-8	1000	—	—	0000
-7	1001	1000	1111	0001
-6	1010	1001	1110	0010
-5	1011	1010	1101	0011
-4	1100	1011	1100	0100
-3	1101	1100	1011	0101
-2	1110	1101	1010	0110
-1	1111	1110	1001	0111
0	0000	1111 or 0000	1000 or 0000	1000
1	0001	0001	0001	1001
2	0010	0010	0010	1010
3	0011	0011	0011	1011
4	0100	0100	0100	1100
5	0101	0101	0101	1101
6	0110	0110	0110	1110
7	0111	0111	0111	1111

Excess is $2^{4-1} = 8$; Thus,
retrieved value = stored value - 8

**Decimal numbers, their
two's complements,
ones' complements,
signed magnitude and
excess 2^{m-1} binary codes**

Existence of two
zeros!

EXPLAIN

Two's-Comp Addition and Subtraction Rules

- Starting from 1000 (-8) on up, each successive 2's comp number all the way to 0111 (+7) can be obtained by adding 1 to the previous one, ignoring any carries beyond the 4th bit position
- Since addition is just an extension of ordinary counting, 2's comp numbers can be added by ordinary binary addition!
- No different cases based on operands' signs!
- Overflow possible
 - Occurs if result is out of range
 - Happens if operands are the same sign but sum is a different sign of that of the operands

Storing an integer in two's complement format:

- Convert the integer to an n-bit binary.
- If it is **positive** or **zero**, it is stored as it is. If it is **negative**, take the two's complement and then store it.

Retrieving an integer in two's complement format:

- If the **leftmost bit** is 1, the computer applies the two's complement operation to the n-bit binary. If the leftmost bit is 0, no operation is applied.
- The computer changes the binary to decimal (integer) and corresponding sign is added.

1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111
-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7

Example:

Retrieve the integer that is stored as 11100110 in memory using two's complement format.

Solution:

The leftmost bit is 1, so the integer is negative. The integer needs to be two's complemented before changing to decimal.

Leftmost bit is 1. The sign is negative

1	1	1	0	0	1	1	0
↓	↓	↓	↓	↓	↓	↓	↓

Apply two's complement operation

0	0	0	1	1	0	1	0
---	---	---	---	---	---	---	---

Integer changed to decimal

26

Sign is added

-26

Comparison

Summary of integer representations

<i>Contents of memory</i>	<i>Unsigned</i>	<i>Sign-and-magnitude</i>	<i>Two's complement</i>
0000	0	0	+0
0001	1	1	+1
0010	2	2	+2
0011	3	3	+3
0100	4	4	+4
0101	5	5	+5
0110	6	6	+6
0111	7	7	+7
1000	8	−0	−8
1001	9	−1	−7
1010	10	−2	−6
1011	11	−3	−5
1100	12	−4	−4
1101	13	−5	−3
1110	14	−6	−2
1111	15	−7	−1