Binary Number System

- Two digits: 0 and 1.
  - Every digit position has a weight that is a power of 2.
  - *Base* or *radix* is 2.
- Examples:

$$110 = 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$$

$$101.01 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2}$$

Binary to Decimal Conversion

- Each digit position of a binary number has a weight.
  – Some power of 2.
- A binary number:

$$B = b_{n-1}\, b_{n-2} \ldots\ldots b_1\, b_0\, .\, b_{-1}\, b_{-2} \ldots\ldots b_{-m}$$

where b_i are the binary digits.

Corresponding value in decimal:

$$D = \sum_{i=-m}^{n-1} b_i\, 2^i$$

Some Examples

1. $101011 \rightarrow 1\times2^5 + 0\times2^4 + 1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0 = 43$

   $(101011)_2 = (43)_{10}$

2. $.0101 \rightarrow 0\times2^{-1} + 1\times2^{-2} + 0\times2^{-3} + 1\times2^{-4} = .3125$

   $(.0101)_2 = (.3125)_{10}$

3. $101.11 \rightarrow 1\times2^2 + 0\times2^1 + 1\times2^0 + 1\times2^{-1} + 1\times2^{-2} = 5.75$

   $(101.11)_2 = (5.75)_{10}$

Decimal to Binary Conversion

- Consider the integer and fractional parts separately.
- For the integer part:
  - Repeatedly divide the given number by 2, and go on accumulating the remainders, until the number becomes zero.
  - Arrange the remainders *in reverse order*.
- For the fractional part:
  - Repeatedly multiply the given fraction by 2.
    - **Accumulate the integer part (0 or 1).**
    - **If the integer part is 1, chop it off.**
  - Arrange the integer parts *in the order* they are obtained.

$$
\begin{array}{r|l}
2 & 15 \\ \hline
2 & 7 \;—\; 1 \\ \hline
2 & 3 \;—\; 1 \\ \hline
 & 1 \;—\; 1
\end{array}
$$

$$(15)_{10} = (1111)_2$$

$$
\begin{array}{r}
0.25 \\
\times\ 2 \\ \hline
\boxed{0}.50 \\
\times\ 2 \\ \hline
\boxed{1}.00
\end{array}
$$

$$(0.25)_{10} = (0.01)_2$$

Examples

2	239	
2	119	--- 1
2	59	--- 1
2	29	--- 1
2	14	--- 1
2	7	--- 0
2	3	--- 1
2	1	--- 1
2	0	--- 1

$(239)_{10} = (11101111)_2$

2	64	
2	32	--- 0
2	16	--- 0
2	8	--- 0
2	4	--- 0
2	2	--- 0
2	1	--- 0
2	0	--- 1

$(64)_{10} = (1000000)_2$

$.634 \times 2 = 1.268$

$.268 \times 2 = 0.536$

$.536 \times 2 = 1.072$

$.072 \times 2 = 0.144$

$.144 \times 2 = 0.288$

:

$(.634)_{10} = (.10100......)_2$

37.0625

$(37)_{10} = (100101)_2$

$(.0625)_{10} = (.0001)_2$

$\therefore (37.0625)_{10} =$

$(100101\ .\ 0001)_2$

Hexadecimal Number System

- A compact way to represent binary numbers.
  - Group of four binary digits are represented by a hexadecimal digit.
  - Hexadecimal digits are 0 to 9, A to F.

Hex	Binary	Hex	Binary
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

32-bit number.

1010	0000	0001	0010	1000	1001	0000	0001	– Binary
A	0	1	2	8	9	0	1	– Hexadecimal.

A0128901

$$(A12)_H = (1010\ 0001\ 0010)_2$$

A 1 2

Binary to Hexadecimal Conversion

- For the integer part:
  - Scan the binary number from *right to left*.
  - Translate each group of four bits into the corresponding hexadecimal digit.
    - Add *leading* zeros if necessary.
- For the fractional part:
  - Scan the binary number from *left to right*.
  - Translate each group of four bits into the corresponding hexadecimal digit.
    - Add *trailing* zeros if necessary.

Examples

1. $(\underline{1011}\ \underline{0100}\ \underline{0011})_2 = (B43)_{16}$
2. $(\underline{10}\ \underline{1010}\ \underline{0001})_2 = (2A1)_{16}$ *Two leading 0s are added*
3. $(.\underline{1000}\ \underline{010})_2 = (.84)_{16}$ *A trailing 0 is added*
4. $(\underline{101}\,.\,\underline{0101}\ \underline{111})_2 = (5.5E)_{16}$ *A leading 0 and trailing 0 are added*

Hexadecimal to Binary Conversion

- Translate every hexadecimal digit into its 4-bit binary equivalent.
- Examples:

$(3A5)_{16} = (0011\ 1010\ 0101)_2$

$(12.3D)_{16} = (0001\ 0010\ .\ 0011\ 1101)_2$

$(1.8)_{16} = (0001\ .\ 1000)_2$

How are Hexadecimal Numbers Written?

- Using the suffix "H" or using the prefix "0x".
- Examples:
  - ADDI R1,2AH // Add the hex number 2A to register R1
  - 0x2AB4 // The 16-bit number 0010 1010 1011 0100
  - 0xFFFFFFFF // The 32-bit number for the all-1 string

Unsigned Binary Numbers

- An *n*-bit binary number can have 2^n distinct combinations.
  - For example, for n=3, the 8 distinct combinations are:
  000, 001, 010, 011, 100, 101, 110, 111 (0 to 2^3-1 = 7 in decimal).

Number of bits (n)	Range of Numbers
8	0 to 2^8-1 (255)
16	0 to 2^{16}-1 (65535)
32	0 to 2^{32}-1 (4294967295)
64	0 to 2^{64}-1

- An *n*-bit binary integer:

$$b_{n-1}b_{n-2}\dots b_2b_1b_0$$

- Equivalent unsigned decimal value:

$$D = b_{n-1}2^{n-1} + b_{n-2}2^{n-2} + \dots + b_2 2^2 + b_1 2^1 + b_0 2^0$$

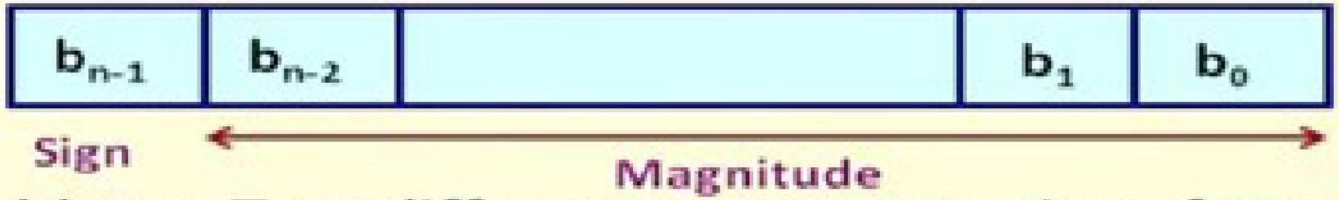
- Each digit position has a weight that is some power of 2.

Signed Integer Representation

- Many of the numerical data items that are used in a program are signed (positive or negative).
  - Question:: *How to represent sign?*
- Three possible approaches:
  a) Sign-magnitude representation
  b) One's complement representation
  c) Two's complement representation

(a) Sign-magnitude Representation

- For an n-bit number representation:
  - The most significant bit (MSB) indicates sign (0: positive, 1: negative).
  - The remaining (n-1) bits represent the magnitude of the number.
- Range of numbers: $-(2^{n-1}-1)$ to $+(2^{n-1}-1)$



- A problem: Two different representations for zero.

  +0: 0 00 .. 000 and -0: 1 00 .. 000

SIGN MAGNITUDE

RANGE: $-(2^{n-1}-1)$ to $+(2^{n-1}-1)$

Example : $n=4$

RANGE: $-(2^3-1)$ to $+(2^3-1)$

$= -7 \qquad = +7$

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

(b) Ones Complement Representation

- Basic idea:
  - Positive numbers are represented exactly as in sign-magnitude form.
  - Negative numbers are represented in 1's complement form.
- How to compute the 1's complement of a number?
  - Complement every bit of the number ($1 \rightarrow 0$ and $0 \rightarrow 1$).
  - MSB will indicate the sign of the number (0: positive, 1: negative).

Example for n=4

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

To find the representation of, say, -4, first note that

+4 = 0100

-4 = 1's complement of 0100 = 1011

- Range of numbers that can be represented in 1's complement:

  Maximum :: $+(2^{n-1}-1)$

  Minimum :: $-(2^{n-1}-1)$

- A problem:

  Two different representations of zero.

  +0 → 0 000....0

  -0 → 1 111....1

- Advantage of 1's complement representation:
  - Subtraction can be done using addition.
  - Leads to substantial saving in circuitry.

(c) Twos Complement Representation

- Basic idea:
  - Positive numbers are represented exactly as in sign-magnitude form.
  - Negative numbers are represented in 2's complement form.
- How to compute the 2's complement of a number?
  - Complement every bit of the number ($1 \rightarrow 0$ and $0 \rightarrow 1$), and then *add one* to the resulting number.
  - MSB will indicate the sign of the number (0: positive, 1: negative).

Example for n=4

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

To find the representation of, say, -4, first note that

+4 = 0100

-4 = 2's complement of 0100 = 1011 + 1
= 1100

- Range of numbers that can be represented in 2's complement:

  Maximum :: $+(2^{n-1}-1)$

  Minimum :: -2^{n-1}

- Advantage of 2's complement representation:
  - Unique representation of zero.
  - Subtraction can be done using addition.
  - Leads to substantial saving in circuitry.
- Almost all computers today use 2's complement representation for storing negative numbers.

- Some other features of 2's complement representation
  a) Weighted number representation, with the MSB having weight -2^{n-1}.

-2^{n-1}	2^{n-2}		2^1	2^0
$\mathbf{b_{n-1}}$	$\mathbf{b_{n-2}}$		$\mathbf{b_1}$	$\mathbf{b_0}$

$$D = -b_{n-1}2^{n-1} + b_{n-2}2^{n-2} + \ldots + b_2 2^2 + b_1 2^1 + b_0 2^0$$

  b) Shift left by k positions with zero padding multiplies the number by 2^k.

00010011 = +19 :: Shift left by 2 :: 01001100 = +76
11100011 = -29 :: Shift left by 2 :: 10001100 = -116

-20 : 101100 in 2's complement

$$\begin{array}{cccccc} -2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ \underline{\underline{1}} & 0 & 1 & 1 & 0 & 0 \end{array} \equiv -2^5 + 2^3 + 2^2 = -20$$

$$\begin{array}{ccccccc} -2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 \end{array} \equiv -2^6 + 2^5 + 2^3 + 2^2 = -\underline{\underline{20}}$$

$$\begin{array}{cccccccc} -2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ \underline{\underline{1}} & 1 & 1 & 0 & 1 & 1 & 0 & 0 \end{array} \equiv -2^7 + 2^6 + 2^5 + 2^3 + 2^2 = -\underline{\underline{20}}$$

$\Longrightarrow$ SIGN EXTENSION IN 2'S COMPLEMENT

c) Shift right by k positions with sign bit padding divides the number by 2^k.

00010110 = +22 :: Shift right by 2 :: 00000101 = +5
11100100 = -28 :: Shift right by 2 :: 11111001 = -7

d) The sign bit can be copied as many times as required in the beginning to extend the size of the number (called *sign extension*).

X = 00101111 (8-bit number, value = +47)
Sign extend to 32 bits:
00000000 00000000 00000000 00101111

X = 10100011 (8-bit number, value = -93)
Sign extend to 32 bits:
11111111 11111111 11111111 10100011

−28 :	1 1 1 0 0 1 0 0	(in 8 bits)
Arithmetic Shift Right :	1 1 1 1 0 0 1 0	(−14)
Arithmetic Shift Right :	1 1 1 1 1 0 0 1	(−7)