CMSC 130 – Logic Design and Digital Computer Circuits
Handout # 1: DATA REPRESENTATION

Number Systems

Given a number in base **B**, having **n** digits in the integer part (left of the radix point) and **m** digits in the fractional part (right of the radix point), that number can be represented in positional form as follows:

$$a_{n-1}a_{n-2}a_{n-3}a_{n-4}\ldots a_2a_1a_0.a_{-1}a_{-2}\ldots a_{-m} = \underbrace{a_{n-1}B^{n-1}+a_{n-2}B^{n-2}+\ldots+a_2B^2+a_1B^1+a_0B^0}_{\text{Integer part}}+\underbrace{a_{-1}B^{-1}+a_{-2}B^{-2}+\ldots+a_{-m}B^{-m}}_{\text{Fractional part}}$$

System	Base	Digits Used	Sample	Decimal Equivalent
Decimal	10	0..9	12	12
Binary	2	0,1	1011	11
Octal	8	0..7	275	189
Hexadecimal	16	0..9,A..F	A4	164

Base Conversions

Octal, Binary and Hexadecimal to Decimal

- Convert the number in positional form and evaluate to obtain its decimal equivalent.

Decimal to Binary, Octal and Hexadecimal

The DABBLE method:

1. Consider the integer part first.
   a. Divide the integer part by **B** (base/radix).
   b. Obtain the remainder.
   c. Get the resulting quotient and divide it by **B** again.
   d. Obtain a new remainder.
   e. IF (quotient!=0) GOTO c.
   f. Read the sequence of remainders backwards to obtain the integer part in base **B.**
2. Now consider the fractional part.
   a. Multiply the fractional part by **B**.
   b. Obtain the integer part of the product.
   c. Get the fractional part of the resulting product and multiply it by **B** again.
   d. Obtain a new integer part of the product.
   e. Repeat step c and d until a sufficient accuracy is obtained.
   f. The resulting sequence of integer parts is the equivalent fractional part in base **B.**
3. Combine the integer part in #1 and the fractional part in #2 to get the equivalent number in base **B.**

Binary to Octal

- Moving from the radix point outward, group the binary digits in 3's.
- Pad in 0's to fill the 3 slots in a group.
- Convert each 3 digit binary number into its Octal equivalent.

Octal to Binary

- Convert each Octal digit to its 3-digit binary equivalent.

Binary to Hexadecimal

- Moving from the radix point outward, group the binary digits in 4's.
- Pad in 0's to fill the 4 slots in a group.
- Convert each 4 digit binary number into its Hexadecimal equivalent.

Hexadecimal to Binary

- Convert each Hexadecimal digit to its 4-digit binary equivalent.

Hexadecimal to Octal

- Hexadecimal to Decimal then to Octal or Hexadecimal to Binary then to Octal.

Octal to Hexadecimal

- Octal to Decimal then to Hexadecimal or Octal to Binary then to Hexadecimal

Binary Codes

Binary Codes are values represented in 0's and 1's to facilitate manipulation in digital systems.

Coding Systems for Decimal Values

1. BCD (Binary Coded Decimal) – 8421 " weight placement code."
2. Excess-3 – add 3 to decimal value and then convert to BCD.
3. 84-2-1 – "weight placement code."
4. 2421 – "weight placement code."
5. Biquinary – 5043210 "weight placement code".

Decimal Digit	**BCD 8421**	**XS3**	**84-2-1**	**2421**	**Biquinary 5043210**
0	0000	0011	0000	0000	0100001
1	0001	0100	0111	0001	0100010
2	0010	0101	0110	0010	0100100
3	0011	0110	0101	0011	0101000
4	0100	0111	0100	0100	0110000
5	0101	1000	1011	1011	1000001
6	0110	1001	1010	1100	1000010
7	0111	1010	1001	1101	1000100
8	1000	1011	1000	1110	1001000
9	1001	1100	1111	1111	1010000

Alphanumeric Codes

Binary coding of letter, symbols and digits.

1. ASCII – American Standard Code for Information Interchange.
2. EBCDIC – Extended BCD Interchange Code.

Character	**7-bit ASCII code**	**8-bit EBCDIC code**
A	0100 0001	1100 0001
B	0100 0010	1100 0010
Z	0101 1010	1110 1001
0	0011 0000	1111 0000
1	0011 0001	1111 0001
9	0011 1001	1111 1001
blank	0010 0000	0100 0000

Fixed-Point Representation

The Fixed-Point representation is a method used to represent integer values.

Magnitude Representation (MR)

- MR makes use of all the bits to represent the magnitude. Because of this, it cannot be used to represent negative values.
- An **n** bit MR will have a range of values $0 \ldots 2^n - 1$. Thus having 2^n distinct values.

Signed-Magnitude Representation (SMR)

- Unlike MR, SMR makes use of the leftmost bit to represent the sign of a number and the rest of the bits for the magnitude.
- If the leftmost bit is 1 then the number is negative. If it is 0 then it is positive.
- An **n**-bit SMR will have the range of values $-(2^{n-1}-1) \ldots (2^{n-1}-1)$. Also yielding 2^n distinct values.

Signed-Complement Representation (SCR)

- Same with SMR, SCR makes use of the leftmost bit as sign bit, however, if the sign bit is 1 (i.e. the number is negative), the complement of the remaining bits is taken.
- There are two ways of SCR:

1's complement form

In 1's complement of SCR the negative number's magnitude bits are in 1's complement (complement the bits).

Note: We need the 1's complement to make subtraction and other logical operations of these numbers easy to implement.

5 bit Examples:

	SCR	Decimal Equivalent
1.	01111	15
	11110	<u>Solution:</u> The leftmost bit is 1, therefore this number is **negative**. The rest of the bits 1110 are the magnitude of the number in 1's complement. We then get the 1's complement of 1110, which is 0001 or **1** in decimal. <u>Answer:</u> -1
2.	00001	1
3.	10101	<u>Solution:</u> The leftmost bit is 1, therefore this number is **negative**. We then get the 1's complement of 0101, which is 1010 or 10 in decimal. <u>Answer:</u> -10
4.	01010	10
5.	00000	POSITIVE ZERO
6.	11111	NEGATIVE ZERO

2's complement form

The magnitude part of this SCR is stored in 2's complement (complement the bits and add 1).

5 bit Examples:

	SCR	Decimal Equivalent
1.	01101	13
2.	10110	Solution: The leftmost bit is 1, therefore this number is **negative**. We then get the 1's complement of 0110, which is 1001. We then add 1 to 1001 to get the 2's complement, which is 1010 or 10. Answer: -10
3.	10001	Solution: The leftmost bit is 1, therefore this number is **negative**. We then get the 1's complement of 0001, which is 1110. Then add 1 to get 1111 or 15 in decimal. Answer: -15
4.	10010	Solution: The leftmost bit is 1, therefore this number is **negative**. We then get the 1's complement of 0010, which is 1101. Then add 1 to get 1110 or 14 in decimal. Answer: -14

Floating-Point Representation

Floating Point Representation is used to represent real numbers using the following notation:

$$\text{MANTISSA X BASE}^{\text{exponent}}$$

Common Floating Point Representation

Notation: SB - sign bit E - exponent HB - hidden bit M - mantissa

1. PDP-11 Format
   - Base 2
   - Mantissa length
   - Normalized : binary point is to the left of the leftmost non-zero digit
     Hidden bit included
   - Exponent length : 8 bits in excess 128
   - Negative numbers indicated by 1 in sign bit
   - 32 bit storage

Examples:

1. 12
   a. Convert to binary: $12 \rightarrow 1100_2$
   b. Normalize: $.1100 \times 2^4$
   c. Express exponent in excess - 128 $4 + 128 = 132 = 1000\ 0100_2$
   d. Floating-point Representation

12 =	0	10000100 .	1	10000000000000000000000
	SB	E	HB	M

2. -5
   a. Convert to binary: -5 -> 101_2
   b. Normalize: $.101 \times 2^3$
   c. Express exponent in excess - 128: $3 + 128 = 131 = 1000\ 0011_2$
   d. Floating-point Representation

-5 =	1	10000011 .	1	01000000000000000000
	SB	E	HB	M

3. 0.125
   a. Convert to binary: .125 -> 0.001_2
   b. Normalize: $.1 \times 2^{-2}$
   c. Express exponent in excess - 128: $-2 + 128 = 126 = 01111110_2$
   d. Floating-point Representation

0.125 =	0	01111110 .	1	0000000000000000000000
	SB	E	HB	M

2. Institute of Electrical and Electronics Engineers (IEEE)

- Base 2
- Mantissa length: 24 bits
- Normalized : binary point is to the right of the leftmost non-zero digit
  Hidden bit included (leftmost non zero digit)
- Exponent length : 8 bits in excess 127
- Negative numbers indicated by 1 in sign bit
- 32 bit storage

Examples:

1. 12
   a. Convert to binary: 12 -> 1100_2
   b. Normalize: 1.100×2^3
   c. Express exponent in excess - 127: $3 + 127 = 130 = 1000\ 0010_2$
   d. Floating-point Representation

12 =	0	10000010 .	1	1000000000000000000000
	SB	E	HB	M

2. -5
   a. Convert to binary: -5 -> 101_2
   b. Normalize: 1.01×2^2
   c. Express exponent in excess - 127: $2 + 127 = 129 = 1000\ 0001_2$
   d. Floating-point Representation

-5 =	1	10000001 .	1	0100000000000000000000
	SB	E	HB	M

3. 0.125
   a. Convert to binary: .125 -> 0.001_2
   b. Normalize: 1.0×2^{-3}
   c. Express exponent in excess - 127 $-3 + 127 = 124 = 01111100_2$
   d. Floating-point Representation

0.125 =	0	01111100 .	1	00000000000000000000000
	SB	E	HB	M

3. IBM Format

- Base 16
- Mantissa length: 24 bits
- Normalized if the first hex digit after the radix point is not zero is not 0000
  No hidden bit
- Exponent length : 7 bits in excess 64
- Negative numbers indicated by 1 in sign bit
- 32 bit storage

Examples:

1. 12
   a. Convert to binary: 12 -> 1100_2
   b. Normalize: $.1100 \times 16^1$
   c. Express exponent in excess - 64 $1+ 64 = 65 = 100\ 0001_2$
   d. Floating-point Representation

12 =	0	1000001 .	110000000000000000000000
	SB	E	M

2. -5
   a. Convert to binary: -5 -> 101_2
   b. Normalize: $.0101 \times 16^1$
   c. Express exponent in excess - 64 $1 + 64 = 65 = 100\ 0001_2$
   d. Floating-point Representation

-5 =	1	1000001 .	010100000000000000000000
	SB	E	M

3. 0.125
   a. Convert to binary: .125 -> 0.001_2
   b. Normalize: $.001 \times 16^0$
   c. Express exponent in excess - 64 $0 + 64 = 64 = 1000000_2$
   d. Floating-point Representation

0.125 =	0	1000000 .	001000000000000000000000
	SB	E	M