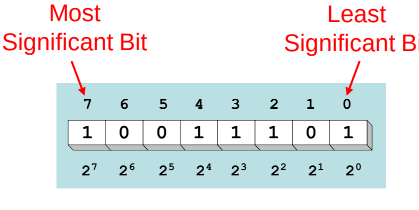
Numbering systems

Binary Numbers

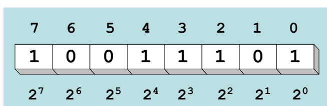
- Each binary digit (called a bit) is either 1 or 0
- Bits have no inherent meaning, they can represent …
  - Unsigned and signed integers
  - Fractions
  - Characters
  - Images, sound, etc.



- Bit Numbering
  - Least significant bit (LSB) is rightmost (bit 0)
  - Most significant bit (MSB) is leftmost (bit 7 in an 8-bit number)

Decimal Value of Binary Numbers

- Each bit represents a power of 2
- Every binary number is a sum of powers of 2
- Decimal Value = $(d_{n-1} \times 2^{n-1}) + \ldots + (d_1 \times 2^1) + (d_0 \times 2^0)$
- Binary $(10011101)_2 = \quad 2^7 + 2^4 + 2^3 + 2^2 + 1 = 157$



Some common powers of 2

2^n	Decimal Value	2^n	Decimal Value
2^0	1	2^8	256
2^1	2	2^9	512
2^2	4	2^{10}	1024
2^3	8	2^{11}	2048
2^4	16	2^{12}	4096
2^5	32	2^{13}	8192
2^6	64	2^{14}	16384
2^7	128	2^{15}	32768

Positional Number Systems

Different Representations of Natural Numbers

XXVII	Roman numerals (not positional)
27	Radix-10 or decimal number (positional)
11011_2	Radix-2 or binary number (also positional)

Fixed-radix positional representation with *n* digits

Number N in radix $r = (d_{n-1}d_{n-2} \ldots d_1d_0)_r$

N_r Value $= d_{n-1} \times r^{n-1} + d_{n-2} \times r^{n-2} + \ldots + d_1 \times r + d_0$

Examples: $(11011)_2 = \quad 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2 + 1 = 27$

$(2107)_8 = \quad 2 \times 8^3 + 1 \times 8^2 + 0 \times 8 + 7 = 1095$

Convert Decimal to Binary

- Repeatedly divide the decimal integer by 2
- Each remainder is a binary digit in the translated value
- Example: Convert 37_{10} to Binary

Division	Quotient	Remainder
37 / 2	18	1
18 / 2	9	0
9 / 2	4	1
4 / 2	2	0
2 / 2	1	0
1 / 2	0	1

least significant bit

$37 = (100101)_2$

most significant bit

stop when quotient is zero

Decimal to Binary Conversion

- $N = (d_{n-1} \times 2^{n-1}) + ... + (d_1 \times 2^1) + (d_0 \times 2^0)$
- Dividing N by 2 we first obtain
  - $\text{Quotient}_1 = (d_{n-1} \times 2^{n-2}) + \dots + (d_2 \times 2) + d_1$
  - $\text{Remainder}_1 = d_0$
  - Therefore, first remainder is least significant bit of binary number
- Dividing first quotient by 2 we first obtain
  - $\text{Quotient}_2 = (d_{n-1} \times 2^{n-3}) + \dots + (d_3 \times 2) + d_2$
  - $\text{Remainder}_2 = d_1$
- Repeat dividing quotient by 2
  - Stop when new quotient is equal to zero
  - Remainders are the bits from least to most significant bit

Popular Number Systems

- Binary Number System: Radix = 2
  - Only two digit values: 0 and 1
  - Numbers are represented as 0s and 1s
- Octal Number System: Radix = 8
  - Eight digit values: 0, 1, 2, …, 7
- Decimal Number System: Radix = 10
  - Ten digit values: 0, 1, 2, …, 9
- Hexadecimal Number Systems: Radix = 16
  - Sixteen digit values: 0, 1, 2, …, 9, A, B, …, F
  - A = 10, B = 11, …, F = 15
- Octal and Hexadecimal numbers can be converted easily to Binary and vice versa

Octal and Hexadecimal Numbers

- Octal = Radix 8
- Only eight digits: 0 to 7
- Digits 8 and 9 not used
- Hexadecimal = Radix 16
- 16 digits: 0 to 9, A to F
- A=10, B=11, …, F=15
- First 16 decimal values (0 to15) and their values in binary, octal and hex. Memorize table

Decimal Radix 10	Binary Radix 2	Octal Radix 8	Hex Radix 16
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	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

- Binary, Octal, and Hexadecimal are related:

  Radix 16 = 2^4 and Radix 8 = 2^3

- Hexadecimal digit = 4 bits and Octal digit = 3 bits

- Starting from least-significant bit, group each 4 bits into a hex digit or each 3 bits into an octal digit

- Example: Convert 32-bit number into octal and hex

3 5 3 0 5 5 2 3 6 2 4	**Octal**
1 1 1 0 1 0 1 1 0 0 0 1 0 1 1 0 1 0 1 0 0 1 1 1 1 0 0 1 0 1 0 0	**32-bit binary**
E B 1 6 A 7 9 4	**Hexadecimal**

Converting Octal & Hex to Decimal

- Octal to Decimal: $N_8 = (d_{n-1} \times 8^{n-1}) + \ldots + (d_1 \times 8) + d_0$

- Hex to Decimal: $N_{16} = (d_{n-1} \times 16^{n-1}) + \ldots + (d_1 \times 16) + d_0$

- Examples:

  $(7204)_8 = (7 \times 8^3) + (2 \times 8^2) + (0 \times 8) + 4 = 3716$

  $(3BA4)_{16} = (3 \times 16^3) + (11 \times 16^2) + (10 \times 16) + 4 = 15268$

Converting Decimal to Hexadecimal

- Repeatedly divide the decimal integer by 16
- Each remainder is a hex digit in the translated value
- Example: convert 422 to hexadecimal

Division	Quotient	Remainder
422 / 16	26	6
26 / 16	1	A
1 / 16	0	1

least significant digit

most significant digit

$422 = (1A6)_{16}$

stop when quotient is zero

- To convert decimal to octal divide by 8 instead of 16

Important Properties

- How many possible digits can we have in Radix *r* ?

  r digits: 0 to $r - 1$

- What is the result of adding 1 to the largest digit in Radix *r*?

  Since digit *r* is not represented, result is $(10)_r$ in Radix *r*

  Examples: $1_2 + 1 = (10)_2$ $\quad$ $7_8 + 1 = (10)_8$

  $9_{10} + 1 = (10)_{10}$ $\quad$ $F_{16} + 1 = (10)_{16}$

Representing Fractions

❖ A number N_r in *radix* r can also have a fraction part:

$$N_r = \underbrace{d_{n-1}d_{n-2} \dots d_1 d_0}_{\text{Integer Part}} \,.\, \underbrace{d_{-1}\, d_{-2} \dots d_{-m+1}\, d_{-m}}_{\text{Fraction Part}} \qquad 0 \le d_i < r$$

Radix Point

❖ The number N_r represents the value:

$$N_r = d_{n-1} \times r^{n-1} + \dots + d_1 \times r + d_0 + \quad \textbf{(Integer Part)}$$

$$d_{-1} \times r^{-1} + d_{-2} \times r^{-2} \dots + d_{-m} \times r^{-m} \quad \textbf{(Fraction Part)}$$

$$N_r = \sum_{i=0}^{i=n-1} d_i \times r^i \; + \sum_{j=-m}^{j=-1} d_j \times r^j$$

Examples of Numbers with Fractions

❖ $(2409.87)_{10}$ $= 2\times10^3 + 4\times10^2 + 9 + 8\times10^{-1} + 7\times10^{-2}$

❖ $(1101.1001)_2$ $= 2^3 + 2^2 + 2^0 + 2^{-1} + 2^{-4} = 13.5625$

❖ $(703.64)_8$ $= 7\times8^2 + 3 + 6\times8^{-1} + 4\times8^{-2} = 451.8125$

❖ $(A1F.8)_{16}$ $= 10\times16^2 + 16 + 15 + 8\times16^{-1} = 2591.5$

❖ $(423.1)_5$ $= 4\times5^2 + 2\times5 + 3 + 5^{-1} = 113.2$

❖ $(263.5)_6$ Digit 6 is NOT allowed in radix 6

Converting Decimal Fraction to Binary

- Convert N = 0.6875 to Radix 2
- Solution: Multiply N by 2 repeatedly & collect integer bits

Multiplication	New Fraction	Bit	
0.6875 × 2 = 1.375	0.375	1	→ First fraction bit
0.375 × 2 = 0.75	0.75	0	
0.75 × 2 = 1.5	0.5	1	
0.5 × 2 = 1.0	0.0	1	→ Last fraction bit

- Stop when new fraction = 0.0, or when enough fraction bits are obtained
- Therefore, $N = 0.6875 = (0.1011)_2$
- Check $(0.1011)_2 = 2^{-1} + 2^{-3} + 2^{-4} = 0.6875$

More Conversion Examples

- Convert N = 139.6875 to Octal (Radix 8)
- Solution: N = 139 + 0.6875 (split integer from fraction)
- The integer and fraction parts are converted separately

Division	Quotient	Remainder
139 / 8	17	3
17 / 8	2	1
2 / 8	0	2

Multiplication	New Fraction	Digit
0.6875 × 8 = 5.5	0.5	5
0.5 × 8 = 4.0	0.0	4

- Therefore, $139 = (213)_8$ and $0.6875 = (0.54)_8$
- Now, join the integer and fraction parts with radix point

  $N = 139.6875 = (213.54)_8$

Simplified Conversions

- Converting fractions between Binary, Octal, and Hexadecimal can be simplified
- Starting at the radix pointing, the integer part is converted from right to left and the fractional part is converted from left to right
- Group 4 bits into a hex digit or 3 bits into an octal digit

← integer: right to left — — fraction: left to right →

7 2 6 1 3 . 2 4 7 4 5 2	Octal
1 1 1 0 1 0 1 1 0 0 0 1 0 1 1 . 0 1 0 1 0 0 1 1 1 1 0 0 1 0 1 0 1	Binary
7 5 8 B . 5 3 C A 8	Hexadecimal

- Use binary to convert between octal and hexadecimal

Adding Bits

- 1 + 1 = 2, but 2 should be represented as $(10)_2$ in binary
- Adding two bits: the sum is S and the carry is C

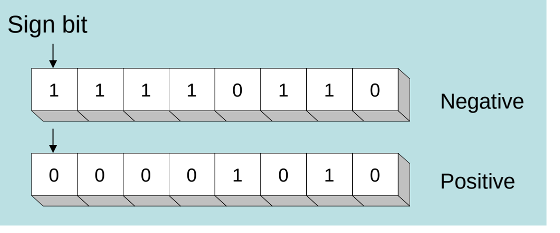
X	0	0	1	1
+ Y	+ 0	+ 1	+ 0	+ 1
C S	0 0	0 1	0 1	1 0

- Adding three bits: the sum is S and the carry is C

0	0	0	0	1	1	1	1
0	0	1	1	0	0	1	1
+ 0	+ 1	+ 0	+ 1	+ 0	+ 1	+ 0	+ 1
0 0	0 1	0 1	1 0	0 1	1 0	1 0	1 1

Signed Integers

- ❖ Highest bit indicates the sign
- ❖ 1 = negative
- ❖ 0 = positive



❖ There are three formats for representing negative numbers

- Sign-magnitude
- 1's complement
- 2's complement

Sign-magnitude

- Sign-magnitude uses one bit for the sign (0=+, 1=-) and the remaining bits represent the magnitude of the number as in the case of unsigned numbers
- For example, using 4-bit numbers

  +5=0101 -5=1101

  +3=0011 -3=1011

  +7=0111 -7=1111
- Although this is easy to understand, it is not well suited for use in computers

1's complement representation

- In the 1's complement scheme, an *n*-bit negative number *K*, is obtained simply by complementing each bit of the number, including the sign bit.

Using 4-bit, write -5 and -3 in 1's complement representation

$5=(0101)_2$ >>>>>>> $-5=(1010)_2$

$3=(0011)_2$ >>>>>>>>> $-3=(1100)_2$

2's complement representation

In the 2's complement scheme, an *n*-bit negative number *K*, is obtained simply by adding 1 to its 1's complement

Forming the Two's Complement

starting value	00100100 = +36
step1: reverse the bits (1's complement)	11011011
step 2: add 1 to the value from step 1	+ 1
sum = 2's complement representation	11011100 = -36

Sum of an integer and its 2's complement must be zero:

00100100 + 11011100 = 00000000 (8-bit sum) ⇒ Ignore Carry

Another way to obtain the 2's complement:

Start at the least significant 1

Leave all the 0s to its right unchanged

Complement all the bits to its left

Binary Value

= 00100$\boxed{1}$00 least significant 1

2's Complement

= 11011$\boxed{1}$00

Two's Complement Representation

- Positive numbers
  - Signed value = Unsigned value

8-bit Binary value	Unsigned value	Signed value
00000000	0	0
00000001	1	+1
00000010	2	+2
. . .	. . .	. . .
01111110	126	+126
01111111	127	+127
10000000	128	-128
10000001	129	-127
. . .	. . .	. . .
11111110	254	-2
11111111	255	-1

Binary Addition

❖ Start with the least significant bit (rightmost bit)

❖ Add each pair of bits

carry		**1**	**1**	**1**	**1**				
	0	0	1	1	0	1	1	0	(54)
+	0	0	0	1	1	1	0	1	(29)
	0	1	0	1	0	0	1	1	(83)
bit position:	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	

Binary Subtraction

❖ When subtracting A – B, convert B to its 2's complement

❖ Add A to (–B), and ignore the end carry (if any)

carry:1 1 1 1

0 1 0 0 1 1 0 1 – 0 0 1 1 1 0 1 0

⇒

0 1 0 0 1 1 0 1
\+ 1 1 0 0 0 1 1 0 (2's complement)

0 0 0 1 0 0 1 1

Carry and Overflow

- Carry is important when …
  - Adding or subtracting unsigned integers
  - Indicates that the unsigned sum is out of range
  - Either < 0 or >maximum unsigned *n*-bit value
- Overflow is important when …
  - Adding or subtracting signed integers
  - Indicates that the signed sum is out of range
- Overflow occurs when
  - Adding two positive numbers and the sum is negative
  - Adding two negative numbers and the sum is positive
  - Can happen because of the fixed number of sum bits

Carry and Overflow Examples

- We can have carry without overflow and vice-versa
- Four cases are possible (Examples are 8-bit numbers)

					1				
	0	0	0	0	1	1	1	1	15
+	0	0	0	0	1	0	0	0	8
	0	0	0	1	0	1	1	1	23

Carry = 0 Overflow = 0

1	1	1	1	1					
	0	0	0	0	1	1	1	1	15
+	1	1	1	1	1	0	0	0	248 (-8)
	0	0	0	0	0	1	1	1	7

Carry = 1 Overflow = 0

	1								
	0	1	0	0	1	1	1	1	79
+	0	1	0	0	0	0	0	0	64
	1	0	0	0	1	1	1	1	143 (-113)

Carry = 0 Overflow = 1

1			1	1					
	1	1	0	1	1	0	1	0	218 (-38)
+	1	0	0	1	1	1	0	1	157 (-99)
	0	1	1	1	0	1	1	1	119

Carry = 1 Overflow = 1

Overflow Detection

overflow can be detected if carry into sign-bit does not equal carry out of sign bit.

Sign Extension

Step 1: Move the number into the lower-significant bits

Step 2: Fill all the remaining higher bits with the sign bit

❖ This will ensure that both magnitude and sign are correct

❖ Examples

- ✧ Sign-Extend 10110011 to 16 bits

  `10110011 = -77` ⇨ `11111111 10110011 = -77`

- ✧ Sign-Extend 01100010 to 16 bits

  `01100010 = +98` ⇨ `00000000 01100010 = +98`

❖ Infinite 0s can be added to the left of a positive number

❖ Infinite 1s can be added to the left of a negative number

Shifting the Bits to the Left

❖ What happens if the bits are shifted to the left by 1 bit position?

Before	0	0	0	0	0	1	0	1	= 5
After	0	0	0	0	1	0	1	0	= 10

Multiplication By 2

❖ What happens if the bits are shifted to the left by 2 bit positions?

Before	0	0	0	0	0	1	0	1	= 5
After	0	0	0	1	0	1	0	0	= 20

Multiplication By 4

❖ Shifting the Bits to the Left by *n* bit positions is multiplication by 2^n

❖ As long as we have sufficient space to store the bits

Shifting the Bits to the Right

❖ What happens if the bits are shifted to the right by 1 bit position?

Before	0	0	1	0	0	1	0	**0**	= 36
After	0	0	0	1	0	0	1	0	= 18, **r=0**

Division By 2

❖ What happens if the bits are shifted to the right by 2 bit positions?

Before	0	0	1	0	0	1	**0**	**0**	= 36
After	0	0	0	0	1	0	0	1	= 9, **r=2**

Division By 4

❖ Shifting the Bits to the Right by *n* bit positions is division by 2^n

❖ The **remainder r** is the value of the bits that are **shifted out**

Binary Codes

- ❖ How to represent characters, colors, etc?
- ❖ Define the set of all represented elements
- ❖ Assign a unique binary code to each element of the set
- ❖ Given n bits, a binary code is a mapping from the set of elements to a subset of the 2^n binary numbers

Example

- ❖ Suppose we want to code 7 colors of the rainbow
- ❖ As a minimum, we need 3 bits to define 7 unique values
- ❖ 3 bits define 8 possible combinations
- ❖ Only 7 combinations are needed
- ❖ Code 111 is not used
- ❖ Other assignments are also possible

Color	3-bit code
Red	000
Orange	001
Yellow	010
Green	011
Blue	100
Indigo	101
Violet	110

Binary Coded Decimal (BCD)

- Simplest binary code for decimal digits
- Only encodes ten digits from 0 to 9
- BCD is a weighted code
- The weights are 8,4,2,1
- Same weights as a binary number
- There are six invalid code words

  1010, 1011, 1100, 1101, 1110, 1111
- Example on BCD coding:

  13 $\Leftrightarrow$ $(0001\ 0011)_{BCD}$

Decimal	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
Unused	1010 ... 1111

Warning: Conversion or Coding?

- Do NOT mix up conversion of a decimal number to a binary number with coding a decimal number with a binary code
- $13_{10} = (1101)_2$ This is conversion
- 13 $\Leftrightarrow$ $(0001\ 0011)_{BCD}$ This is coding
- In general, coding requires more bits than conversion
- A number with *n* decimal digits is coded with 4*n* bits in BCD