Data Representation

COE 301

Computer Organization

Prof. Muhamed Mudawar

College of Computer Sciences and Engineering

King Fahd University of Petroleum and Minerals

Presentation Outline

- Positional Number Systems
- Binary and Hexadecimal Numbers
- Base Conversions
- Integer Storage Sizes
- Binary and Hexadecimal Addition
- Signed Integers and 2's Complement Notation
- Sign Extension
- Binary and Hexadecimal subtraction
- Carry and Overflow
- Character Storage

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 *k* digits

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

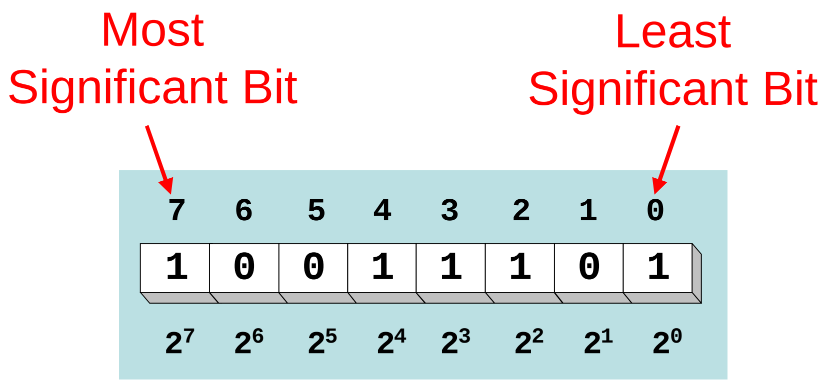
Value $= d_{k-1} \times r^{k-1} + d_{k-2} \times r^{k-2} + \ldots + d_1 \times r + d_0$

Examples: $(11011)_2 = 1\times2^4 + 1\times2^3 + 0\times2^2 + 1\times2 + 1 = 27$

$(2103)_4 = 2\times4^3 + 1\times4^2 + 0\times4 + 3 = 147$

Binary Numbers

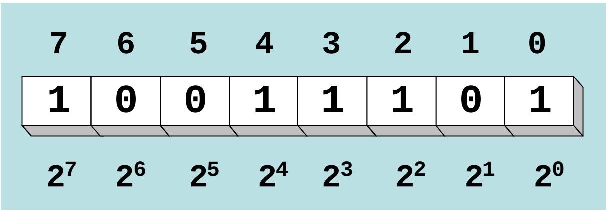
- ❖ Each binary digit (called bit) is either 1 or 0
- ❖ Bits have no inherent meaning, can represent
  - ✧ Unsigned and signed integers
  - ✧ Characters
  - ✧ Floating-point numbers
  - ✧ Images, sound, etc.



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

Converting Binary to Decimal

- 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 $(10011101)_2 = 2^7 + 2^4 + 2^3 + 2^2 + 1 = 157$



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

Convert Unsigned Decimal to Binary

- ❖ Repeatedly divide the decimal integer by 2
- ❖ Each remainder is a binary digit in the translated value

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 (first remainder)

most significant bit (last remainder)

stop when quotient is zero

$37 = (100101)_2$

Hexadecimal Integers

- 16 Hexadecimal Digits: 0 – 9, A – F
- More convenient to use than binary numbers

Binary, Decimal, and Hexadecimal Equivalents

Binary	Decimal	Hexadecimal	Binary	Decimal	Hexadecimal
0000	0	0	1000	8	8
0001	1	1	1001	9	9
0010	2	2	1010	10	A
0011	3	3	1011	11	B
0100	4	4	1100	12	C
0101	5	5	1101	13	D
0110	6	6	1110	14	E
0111	7	7	1111	15	F

Converting Binary to Hexadecimal

- Each hexadecimal digit corresponds to 4 binary bits
- Example:

  Convert the 32-bit binary number to hexadecimal

  1110 1011 0001 0110 1010 0111 1001 0100

- Solution:

E	B	1	6	A	7	9	4
1110	1011	0001	0110	1010	0111	1001	0100

Converting Hexadecimal to Decimal

- Multiply each digit by its corresponding power of 16

  Value = $(d_{n-1} \times 16^{n-1}) + (d_{n-2} \times 16^{n-2}) + ... + (d_1 \times 16) + d_0$

- Examples:

  $(1234)_{16} = (1 \times 16^3) + (2 \times 16^2) + (3 \times 16) + 4 =$

  Decimal Value 4660

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

  Decimal Value 15268

Converting Decimal to Hexadecimal

- ❖ Repeatedly divide the decimal integer by 16
- ❖ Each remainder is a hex digit in the translated value

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

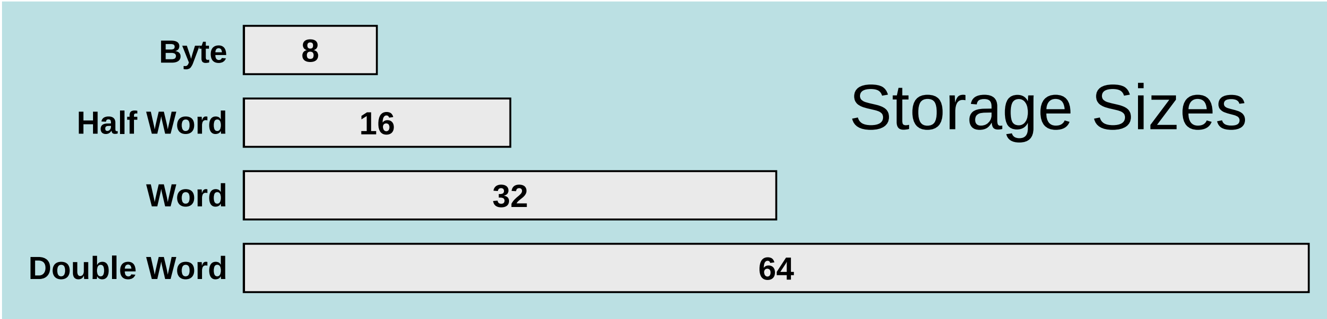
Remainder 6: least significant digit

Remainder 1: most significant digit

Quotient 0: stop when quotient is zero

Decimal 422 = 1A6 hexadecimal

Integer Storage Sizes



Storage Type	Unsigned Range	Powers of 2
Byte	0 to 255	0 to $(2^{8} – 1)$
Half Word	0 to 65,535	0 to $(2^{16} – 1)$
Word	0 to 4,294,967,295	0 to $(2^{32} – 1)$
Double Word	0 to 18,446,744,073,709,551,615	0 to $(2^{64} – 1)$

What is the largest 20-bit unsigned integer?

Answer: $2^{20} – 1 = 1,048,575$

Binary Addition

- Start with the least significant bit (rightmost bit)
- Add each pair of bits
- Include the carry in the addition, if present

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**	

Hexadecimal Addition

- Start with the least significant hexadecimal digits
- Let Sum = summation of two hex digits
- If Sum is greater than or equal to 16
  - Sum = Sum – 16 and Carry = 1
- Example:

```
carry:        1   1       1
         1 C 3 7 2 8 6 A
    +    9 3 9 5 E 8 4 B
         ---------------
         A F C D 1 0 B 5
```

A + B = 10 + 11 = 21
Since 21 ≥ 16
Sum = 21 – 16 = 5
Carry = 1

Signed Integers

- Several ways to represent a signed number
  - Sign-Magnitude
  - Biased
  - 1's complement
  - 2's complement
- Divide the range of values into 2 equal parts
  - First part corresponds to the positive numbers (≥ 0)
  - Second part correspond to the negative numbers (< 0)
- Focus will be on the 2's complement representation
  - Has many advantages over other representations
  - Used widely in processors to represent signed integers

Two's Complement Representation

- ❖ Positive numbers
  - ✧ Signed value = Unsigned value
- ❖ Negative numbers
  - ✧ Signed value = Unsigned value – 2^n
  - ✧ n = number of bits
- ❖ Negative weight for MSB
  - ✧ Another way to obtain the signed value is to assign a negative weight to most-significant bit

1	0	1	1	0	1	0	0
-128	64	32	16	8	4	2	1

= -128 + 32 + 16 + 4 = -76

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

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

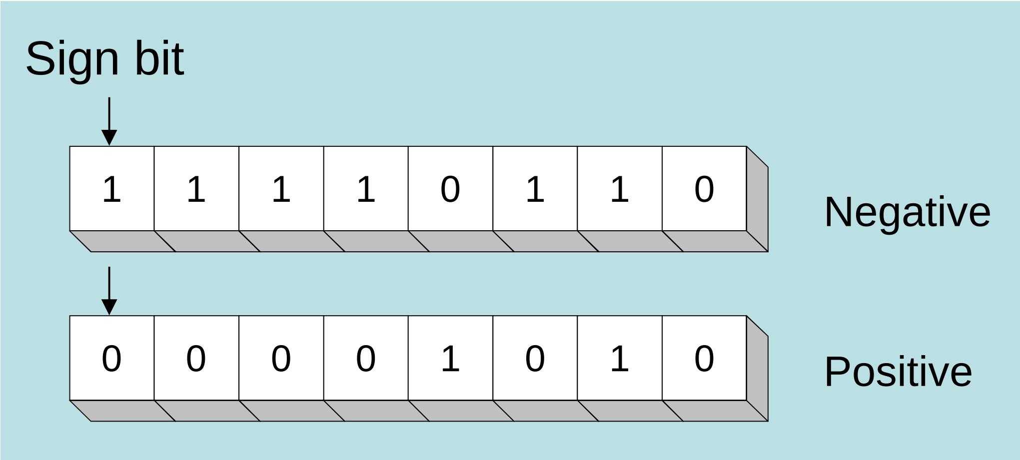
= 00100100 (least significant 1)

2's Complement

= 11011100

Sign Bit

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



For Hexadecimal Numbers, check most significant digit

If highest digit is > 7, then value is negative

Examples: 8A and C5 are negative bytes

B1C42A00 is a negative word (32-bit signed integer)

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

Two's Complement of a Hexadecimal

- To form the two's complement of a hexadecimal
  - Subtract each hexadecimal digit from 15
  - Add 1
- Examples:

  2's complement of **6A3D = 95C2 + 1 = 95C3**

  2's complement of **92F15AC0 = 6D0EA53F + 1 = 6D0EA540**

  2's complement of **FFFFFFFF = 00000000 + 1 = 00000001**

- No need to convert hexadecimal to binary

Binary Subtraction

- ❖ When subtracting A – B, convert B to its 2's complement
- ❖ Add A to (–B)

```
borrow:   1  1        1              carry:  1  1        1  1
          0 1 0 0 1 1 0 1                    0 1 0 0 1 1 0 1
–         0 0 1 1 1 0 1 0     ⇒     +        1 1 0 0 0 1 1 0  (2's complement)
          ---------------                    ---------------
          0 0 0 1 0 0 1 1                    0 0 0 1 0 0 1 1  (same result)
```

- ❖ Final carry is ignored, because
  - ✧ Negative number is sign-extended with 1's
  - ✧ You can imagine infinite 1's to the left of a negative number
  - ✧ Adding the carry to the extended 1's produces extended zeros

Hexadecimal Subtraction

16 + 5 = 21

```
Borrow:  1 1       1              Carry: 1   1 1 1 1
         B14FC675                        B14FC675
       - 839EA247                      + 7C615DB9   (2's complement)
         --------                        --------
         2DB1242E                        2DB1242E   (same result)
```

- ❖ When a borrow is required from the digit to the left, then

  Add 16 (decimal) to the current digit's value
- ❖ Last Carry is ignored

Ranges of Signed Integers

For *n*-bit signed integers: Range is -2^{n-1} to $(2^{n-1} - 1)$

Positive range: 0 to $2^{n-1} - 1$

Negative range: -2^{n-1} to -1

Storage Type	Signed Range	Powers of 2
Byte	–128 to +127	-2^{7} to $(2^{7} - 1)$
Half Word	–32,768 to +32,767	-2^{15} to $(2^{15} - 1)$
Word	–2,147,483,648 to +2,147,483,647	-2^{31} to $(2^{31} - 1)$
Double Word	–9,223,372,036,854,775,808 to +9,223,372,036,854,775,807	-2^{63} to $(2^{63} - 1)$

Practice: What is the range of signed values that may be stored in 20 bits?

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)

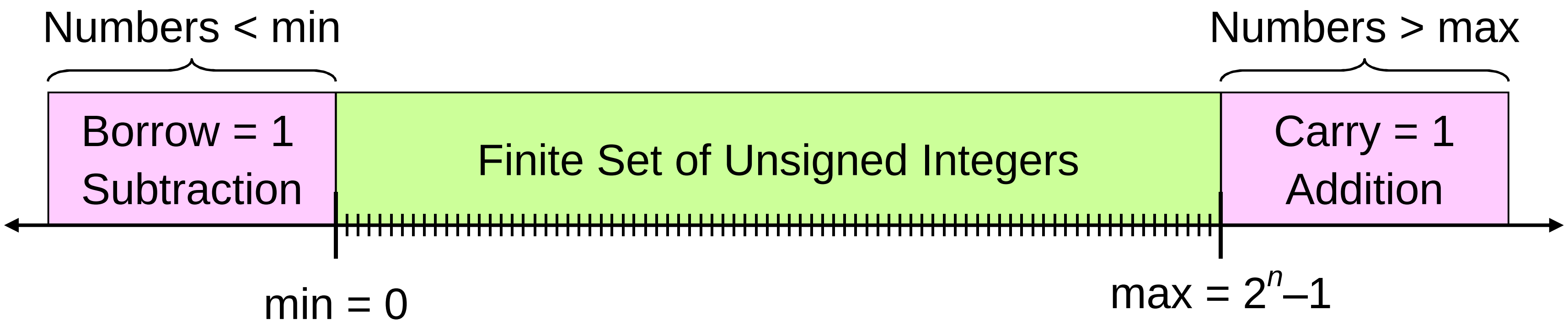
Carry = 0 Overflow = 1

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

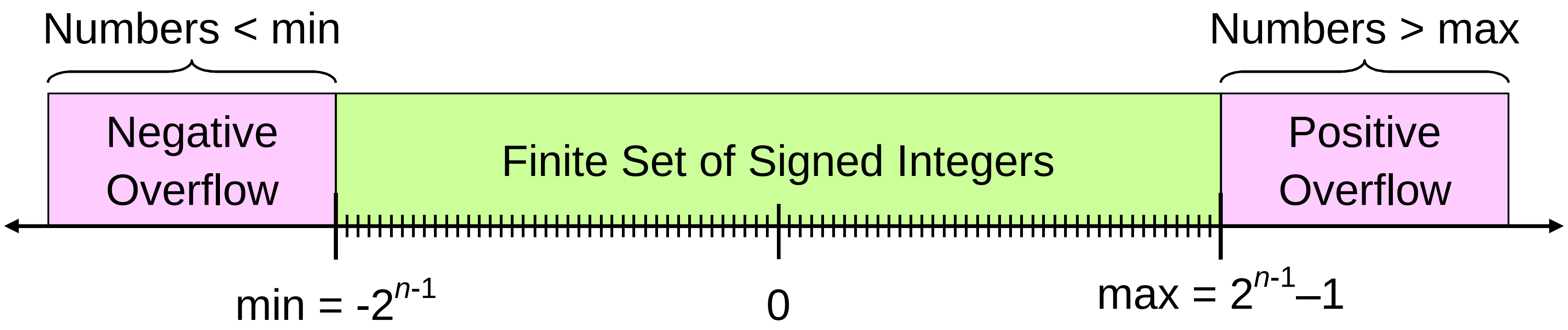
Carry = 1 Overflow = 1

Range, Carry, Borrow, and Overflow

❖ Unsigned Integers: *n*-bit representation



❖ Signed Integers: *n*-bit 2's complement representation



Character Storage

- Character sets
  - Standard ASCII: 7-bit character codes (0 – 127)
  - Extended ASCII: 8-bit character codes (0 – 255)
  - Unicode: 16-bit character codes (0 – 65,535)
  - Unicode standard represents a universal character set
    - Defines codes for characters used in all major languages
    - Used in Windows-XP: each character is encoded as 16 bits
  - UTF-8: variable-length encoding used in HTML
    - Encodes all Unicode characters
    - Uses 1 byte for ASCII, but multiple bytes for other characters
- Null-terminated String
  - Array of characters followed by a NULL character

Printable ASCII Codes

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	space	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

- Examples:
  - ASCII code for space character = 20 (hex) = 32 (decimal)
  - ASCII code for 'L' = 4C (hex) = 76 (decimal)
  - ASCII code for 'a' = 61 (hex) = 97 (decimal)

Control Characters

- The first 32 characters of ASCII table are used for control
- Control character codes = 00 to 1F (hexadecimal)
  - Not shown in previous slide
- Examples of Control Characters
  - Character 0 is the NULL character $\Rightarrow$ used to terminate a string
  - Character 9 is the Horizontal Tab (HT) character
  - Character 0A (hex) = 10 (decimal) is the Line Feed (LF)
  - Character 0D (hex) = 13 (decimal) is the Carriage Return (CR)
  - The LF and CR characters are used together
    - They advance the cursor to the beginning of next line
- One control character appears at end of ASCII table
  - Character 7F (hex) is the Delete (DEL) character