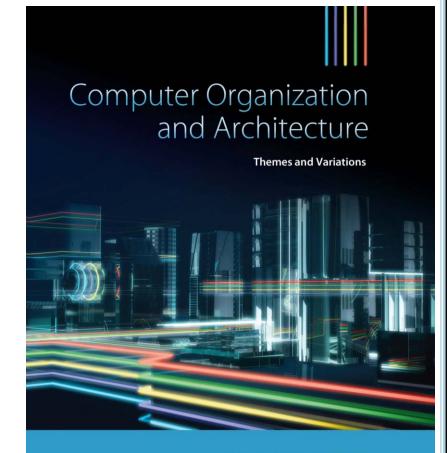
Part 2

CHAPTER 2

Computer
Arithmetic and
Digital Logic



Alan Clements

1

These slides are being provided with permission from the copyright for in-class (CS2208B) use only. The slides must not be reproduced or provided to anyone outside of the class.

All download copies of the slides and/or lecture recordings are for personal use only. Students must destroy these copies within 30 days after receipt of final course evaluations.



Music: "Corporate Success" by Scott Holmes, used under <u>Attribution-NonCommercial License</u>

Signed Integers

- ☐ Signed numbers can be represented in many ways.
- ☐ Computer designers have adopted three techniques:
 - o sign and magnitude,
 - o biased representation, and
 - o two's complement.

Sign and Magnitude Representation

- \square An *n*-bit word has 2^n possible values
 - o for example, an eight-bit word can represent 256 different numbers.
- ☐ One way of representing a negative number is to take the most-significant bit and reserve it to indicate the sign of the number.
 - o 0 represents positive numbers and => the first bit shows the negative.
 - o 1 represents negative numbers.
- □ In an eight-bit word, using sign and magnitude representation How many positive numbers can be represented? How many negative numbers can be represented?
- ☐ The value of a *sign and magnitude number* can be expressed as (-1) $^{S} \times M$ where S is the sign bit and M is the magnitude of the number.
 - \circ If S = 0, $(-1)^0 = +1$ and the number is positive.
 - o If S = 1, $(-1)^1 = -1$ and the number is negative.
- \Box for example, in 8 bits we can <u>interpret</u> the numbers 00001101_2 and 10001101_2 as $+13_{10}$ and -13_{10} , respectively.
- □ Sign and magnitude representation is
 - o not generally used in integer arithmetic,
 - however, it is used in floating-point arithmetic.

20

Excess-8 code

Biased Representation

- □ Assume that we have numbers ranging from −8 to +8
- ☐ To go around the negative sign, we can *shift the scale* (by adding a bias = 8 to all numbers) to have only non-negative values.
 - The original number = -8 → The biased number =
 - The original number = -7 \rightarrow The biased number =
 - \circ The original number = $0 \rightarrow$ The biased number =

 - The original number = +7
 The biased number = 15
 The original number = +8
 The biased number = 16
- ☐ To convert a biased number to its original value, you need to shift back the scale (by subtracting the bias from the biased number).
 - The biased number = $0 \rightarrow$ The original number = -8
 - The biased number = $1 \rightarrow$ The original number = -7
 - The biased number = $8 \rightarrow$ The original number = 0
 - The biased number = 15
 → The original number = +7
 The biased number = 16
 → The original number = +8
- ☐ In this representation, biased numbers are unsigned integers; yet, they represent both positive and negative values.
- \Box The <u>biased representation</u> is also called <u>excess-K</u>, where K is a pre-specified biasing value

Complementary Arithmetic

- ☐ A number and its *complement* add up to a constant;
 - o In *n*-digit <u>decimal</u> ten's complement (radix complement) arithmetic, if P is a number then its ten's complement is Q, where $P + Q = 10^{\text{n}}$.
 - the *ten's* complement of 12 is 88 because 12 + 88 = 100.
 - the *ten's* complement of 88 is 12 because 88 + 12 = 100.
 - o In *n-bit* <u>binary</u> two's complement (radix complement) arithmetic, if P is a number then its two's complement is Q, where $P + Q = 2^n$.
 - the *two*'s complement of 101_2 is 011_2 because $101_2 + 011_2 = 1000_2$.
 - the *two's* complement of 011_2 is 101_2 because $011_2 + 101_2 = 1000_2$.

- ☐ In binary arithmetic, the *two's complement* of a number is formed by
 - Subtracting the number from 2^n .

the

The *two's* complement of 01100101_2 is $100000000_2 - 01100101_2 = 10011011_2$

 \triangleright Flipping (inverting) all the bits of the number and adding 1.

The *two*'s complement of 01100101_2 is $10011010_2 + 1_2 = 10011011_2$.



- Processing all the bits of the number from the <u>least significant bit</u> (LSB) towards the <u>most significant bit</u> (MSB)
 - > copying all the zeros until the first 1 is reached,
 - > copying that 1,
 - > flipping (inverting) all the remaining bits.

The *two's* complement of $01100|100_2$ is $10011|100_2$. The *two's* complement of $0110010|1_2$ is $1001101|1_2$.

- ☐ We are interested in *complementary* arithmetic because *subtracting* a number is the same as *adding its complement*.
 - o To subtract 001100101_2 from a binary number, we just need to add its *two*'s complement (i.e., 110011011_2) to the number.

- \square The *two's* complement of an *n*-bit binary value, N, is defined as $2^n N$.
- □ If $N = 5_{10} = 00000101_2$ (8-bit arithmetic), the *two's* complement of N is given by $2^8 00000101_2 = 100000000_2 00000101_2 = 11111011_2$.
- □ 11111011₂ represents -00000101_2 (i.e., -5_{10}) or -123_{10} depending only on whether we interpret 11111011₂ as a *two's complement integer* or as a *sign-magnitude integer*, respectively.

This is _-123 not +123

This is sign-magnitude not unsigned

The outcome is different if interpreted in different way.

☐ This example demonstrates 8-bit *two's* complement arithmetic. We begin by writing down the representations of +5, -5, +7 and -7.

$$+5_{10} = {\color{red}00000101}_2 \quad -5_{10} = {\color{red}11111011}_2 \quad +7_{10} = {\color{red}00000111}_2 \quad -7_{10} = {\color{red}11111001}_2$$

 \square Now $7_{10} - 5_{10}$ can be calculated as $7_{10} + (-5_{10})$ by adding the binary value for 7_{10} to the *two's* complement of 5_{10} .

- the ninth-bit is ignored.

The result is correct if the left-hand carry-out is ignored.

 \square Now consider the addition of -7_{10} to $+5_{10}$, i.e., $+5_{10}$ + (-7_{10})

The result is 111111110_2 (the carry-out bit is 0).

The expected answer is -2_{10} ; that is, $2^8 - 2_{10} = 100000000_2 - 00000010_2 = 111111110_2$.

- □ *Two's* complement arithmetic is not magic.
- \square Consider the calculation Z = X Y in n-bit arithmetic
 - \circ we do it by *adding* the *two's* complement of Y to X.
- \square The *two's* complement of Y is defined as $2^n Y$.

We get

$$Z = X - Y = X + (2^n - Y) = 2^n + (X - Y).$$

- ☐ This is
 - \circ the desired result, X-Y
 - o If the result of X-Y is <u>negative</u>, the *two's* complement will be automatically calculated, i.e., $2^n + (X-Y)$, where (X-Y) is <u>negative</u>.
 - Otherwise, we discard the *unwanted carry-out digit* (i.e., 2^n) at the leftmost position

Time examples using radix complement

In the 24H system (i.e., radix = 24), assume that

- □ the time right now is 2 am and we want to subtract 5 hours from 2 am
 - The radix complement of 5 is 24 5 = 19
 - o To subtract 5 from 2 in the 24H system, it is equivalent to adding the 2 to the complement of 5, i.e., 2 + 19 = 21 (correct answer)
 - The 21 can also be read as −3 in the radix complement system
- □ the time right now is 5 am and we want to subtract 2 hours from 5 am
 - The radix complement of 2 is 24 2 = 22
 - o To subtract 2 from 5 in the 24H system, it is equivalent to adding the 5 to the complement of 2, i.e., 5 + 22 = 27
 - As all the time values must be between 0 and 23, we need to subtract a full 24H from the result, i.e., 27 24 = 3 (correct answer)

$$-X = -9_{10} = 100000000_2 - 00001001_2 = 11110111_2$$

 $-Y = -6_{10} = 100000000_2 - 00000110_2 = 11111010_2$

Let $X = 9_{10} = 00001001_2$ and $Y = 6_{10} = 00000110_2$

1.
$$+X$$
 +9 00001001
 $+Y$ +6 +00000110
00001111

2.
$$+x$$
 +9 00001001
 $-x$ -6 +11111010
+3 10000011

11111

11111101 is a negative *two's* complement number The two's complement of **11111101** is **00000011**₂ *Hence*, **11111101** = **-00000011**₂ = -3_{10}

11110001 is a negative *two*'s complement number The two's complement of **11110001** is **00001111**₂ *Hence*, **11110001** = **-00001111**₂ = -15_{10}

All four examples give the result we would expect when the result is interpreted as a *two's* complement number.

Properties of Two's complement Numbers

- The two's complement system is a true complement system in that +X + (-X) = 0.
- There is one unique zero 00...0. It has ±0
 This is not the case with the signed magnitude number system
- The most-significant bit of a two's complement number is a sign bit.
 - The number is *positive* if the most-significant bit is θ , and
 - *negative* if it is 1.

This is the same as the signed magnitude number system

- The range of an *n*-bit *two's* complement number is from -2^{n-1} to $+2^{n-1}-1$.
 - For n = 4, the range is -8 to +7.

The total number of different values is $2^n = 16$

- 8 negative,
- zero and
- 7 positive.
- For n = 8, the range is -128 to +127.

The total number of different values is $2^n = 256$

- 128 negative,
- zero and
- 127 positive.
- How about the signed magnitude number system?

- \square The *range* of *two's* complement numbers in n-bits is from -2^{n-1} to $+2^{n-1}-1$.
- What happens if we violate this rule by carrying out an operation whose result falls outside the range of values that can be represented by *two's* complement numbers? —> OverHow.
 - In a *five-bit* representation, the range of *valid two's complement numbers* is

 -16 to +15.

Case 1

```
5 = 00101
+7 = 00111
12   01100 = 12_{10}
```

Case 2

```
12 = 01100 
 + 13 = 01101 
 an overflow occurred.
```

 $\overline{25}$ $\overline{11001} = -7_{10}$ (as a *two's* complement value)

In Case 1 we get the expected answer of $+12_{10}$, but

In Case 2 we get a negative result because the sign bit is 1.

- If the answer were regarded as an *unsigned binary number* it would be +25, which is, of course, the correct answer.
- However, as the *two's* complement system has been chosen to represent signed numbers, all answers must be interpreted in this light.

- ☐ If we add together two negative numbers whose total is less than −16, we will go out of range.
 - o For example, if we add $-9 = 10111_2$ and $-12 = 10100_2$, we get:

```
-9 = 10111
-12 = +10100
-21 gives a positive result 01011_2 = +11_{10}
```

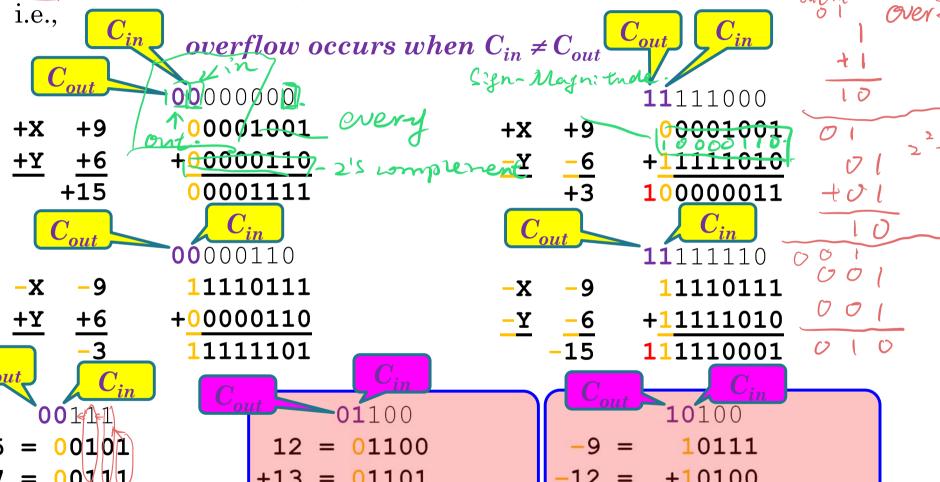
- ☐ The last two examples demonstrate *arithmetic overflow*
- □ *Arithmetic overflow* occurs during a *two's complement addition* when
 - o the result of *adding two positive* numbers *yields a negative* result, or
 - o the result of adding two negative numbers yields a positive result.
- \Box If the sign bits of A and B are the same but the sign bit of the result is different, it means arithmetic overflow has occurred.
- □ Overflow <u>never</u> occurs when adding two numbers with opposite signs
- ☐ If $\mathbf{a_{n-1}}$ is the sign bit of A, $\mathbf{b_{n-1}}$ is the sign bit of B, and $\mathbf{s_{n-1}}$ is the sign bit of the sum of A and B,
 then

overflow is defined by the *logical expression* as follow:

$$V = (\mathbf{a_{n-1}})^{c} \cdot (\mathbf{b_{n-1}})^{c} \cdot \mathbf{s_{n-1}} + \mathbf{a_{n-1}} \cdot \mathbf{b_{n-1}} \cdot (\mathbf{s_{n-1}})^{c}$$
where ()^c is the complement binary operation



☐ In practice, real systems detect overflow from the *carry bits* <u>in</u> to (i.e., C_{in}) and <u>out</u> of (i.e., C_{out}) the most-significant bit of an adder;



00111 01100

11001

Overflow occurred

+10100

101011

Overflow occurred

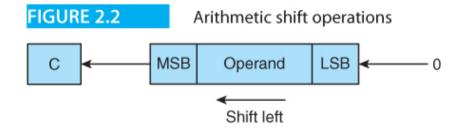
35

Shifting Operations

- \square In a shift operation, the *bits of a word* are *shifted* one or more places *left* or *right*.
- ☐ The *arithmetic* shift is just one type of shift operation.
- ☐ In Chapter 3, we will cover the other types of shift operations.

Shifting Operations (left)

- \Box Figure 2.2(a) describes the arithmetic *shift left*.
 - o A zero enters into the vacated least-significant bit position and
 - the bit *shifted out* of the *most-significant bit* position is recorded in the computer's *carry flag*.
 - \circ In *two's* complement and unsigned numbers:
 - > Arithmetic *shift left* means *multiplying* by 2.
- \square If 00100111_2 (39)₁₀ is shifted one place left it becomes 010011110_2 (78)₁₀.
- \square If 11100011_2 (-29)₁₀ is shifted one place left it becomes 11000110_2 (-58)₁₀.

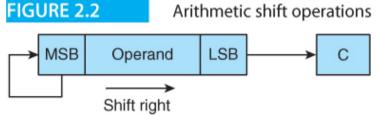


(a) Arithmetic shift left:

A zero enters the least-significant bit position and the most-significant bit is copied into the carry flag. For example: 11000101 becomes 10001010 after shifting one place left.

Shifting Operations (right)

- \Box Figure 2.2(b) describes the arithmetic *shift right*.
 - o the sign bit (the MSB) is *replicated* into the vacated *most-significant bit* position and
 - o the bit *shifted out* of the *least-significant bit* position is recorded in the computer's *carry flag*.
 - o In *two's* complement and unsigned numbers:
 - > Arithmetic *shift right* means *dividing* by 2.
- \square Shifting 00001100_2 (12)₁₀ one place right produces 00000110_2 (6)₁₀.
- \square Shifting 00001101_2 (13)₁₀ one place right produces 00000110_2 (6)₁₀ as well.
- \square Shifting 11100010_2 (-30)₁₀ one place right produces 11110001_2 (-15)₁₀.
- \square Shifting 11100011_2 (-29) one place right produces 11110001_2 (-15) as well.



(b) Arithmetic shift right:

A copy of the most-significant bit enters the most-significant bit position. All other bits are shifted one place right. The least-significant bit is copied into the carry flag.

For example; 00100101 becomes 00010010 after shifting one place right, and 11100101 becomes 11110010 after shifting one place right.

38

Binary Multiplication

☐ Multiplication can be implemented using arithmetic *shift left* and addition operations to add up the partial products as they are formed.

Multiplicand	Multiplier	Step	Pa	ırtıa	al p	rod	uct	\mathbf{S}										
01101001 01101001 01101001 01101001	01001001 01001001 01001001 01001001	1 2 3 4					0	0	0 0 1	0 0 0 0	1 0 0	1 0 0 0	0 0 0	1 0 0	0 0 0	0	1	
01101001	01001001	5				0	0	0	0	0	0	0	0	_				
01101001	01001001	6			0	0	0	0	0	0	0	0						
01101001	0 1 001001	7		0	1	1	0	1	0	0	1							39
01101001	0 1001001	8	0	0	0	0	0	0	0	0								
		Result	0	0	1	1	1	0	1	1	1	1	1	0	0	0	1	

D--4:-1 ---- 1---4-