4004CEM
COMPUTER ARCHITECTURE & NETWORKS

CHAPTER 3:

DATA REPRESENTATION

LEARNING OUTCOMES

To understand the conversion and representing numbers, characters and symbols.

CONTENT

Representing numbers

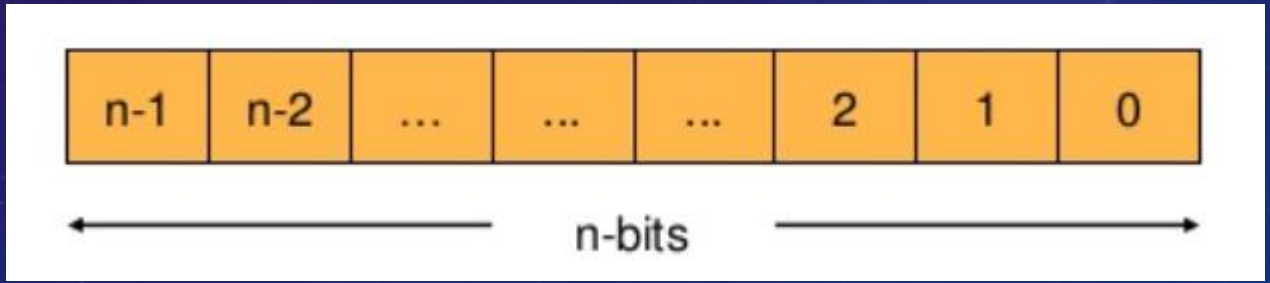
- Unsigned (no negative or positive sign in front of numbers e.g. 7,8,10)
- Signed (has sign in front of numbers e.g. -7, +8, -10)
- Floating point (e.g. 8.304)

Representing characters & symbols

- ASCII
- Unicode

DATA REPRESENTATION IN COMPUTERS

- Data Representation refers to the form in which data is stored, processed, and transmitted.
- Devices such as smartphones, iPods, and computers store data in digital formats that can be handled by electronic circuitry.
- Data are stored in Registers.
- Registers are limited in number & size. (e.g. 8 bits, 16 bits)



With a *n*-bit register

- Min value 0
- Max value 2^n-1

DATA REPRESENTATION

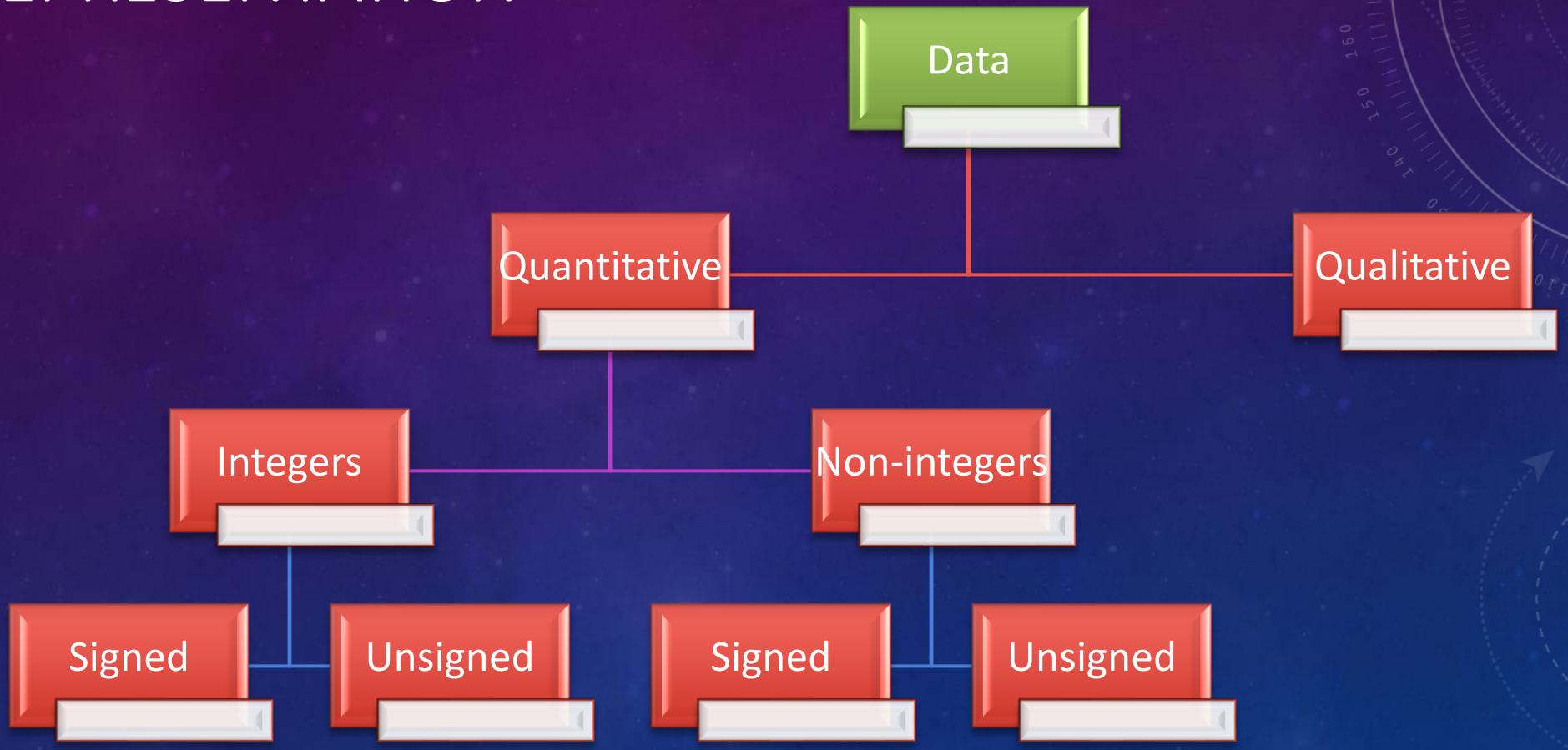
Qualitative

- Represents quality or characteristics
- Not proportional to a value
- Name, Address

Quantitative

- Quantifiable
- Proportional to value a
- No of students, marks for 120CT, CGPA

DATA REPRESENTATION



QUANTITATIVE NUMBERS

- Integers
  - Unsigned 20
  - Signed +20, -20
- Non-integers
  - Floating point number -10.25, 3.33333…., 1/8 = 0.125

SIGNED INTEGERS

- We need a way to represent negative values
- 2 representations
  - Sign & Magnitude representation
  - Complement method

SIGN & MAGNITUDE REPRESENTATION

- *n*-bit unsigned magnitude & sign bit (S)
- If S
  - 0 – integer is a positive or zero
  - 1 – integer is negative or zero
- Range – $(2^n - 1)$ to + $(2^n - 1)$

EXAMPLE – SIGN & MAGNITUDE

- ❑ If 8 bit register is used what are min & max numbers?
- ❑ What are 0000 0000 and 1000 0000 in decimal?
  - Representation of zero is not unique

SIGNED BINARY NUMBERS

Signed Magnitude

MSB = sign bit

For an 8 bit word, -11_{10} is 10001011

- Easy for human, but difficult to implement in hardware

SIGN & MAGNITUDE

- Advantages
  - Sign reversal
  - Finding absolute value |a|
    - Flip sign bit
- Disadvantage
  - Adding a negative of a number is not the same as subtraction
    - e.g. add 2 and -3
    - Needed different operations
  - Zero is not unique

SIGNED INTEGER REPRESENTATION

- The conversions presented so far involved only positive numbers.
- To **represent negative values**, computer systems allocate the high order bit to indicate the sign of a value.
  - The high order is the leftmost bit in a byte. It is also called the most significant bit.
- The remaining bits contain the value of the number.

SIGNED INTEGER REPRESENTATION

- There are three ways in which signed binary numbers may be expressed:
  - Signed magnitude
  - One’s complement
  - Two’s complement
- In an 8 bit word, signed magnitude representation places the absolute value of the number in the 7 bits to the right of the sign bit.

SIGNED MAGNITUDE REPRESENTATION

- For example, in a 8 bit signed magnitude,

  POSITIVE 3 is : 0 0 0 0 0 0 1 1

  NEGATIVE 3 is : 1 0 0 0 0 0 1 1

- Computers perform arithmetic operations on signed magnitude numbers in much the same way as humans carry out pencil and paper arithmetic.
  - Humans often ignore the signs of the operands while performing a calculation, applying the appropriate sign after the calculation is complete.

SIGNED MAGNITUDE REPRESENTATION

- Binary addition is as easy as it gets. You need to know only four rules:

  0 + 0 = 0

  0 + 1 = 1

  1 + 0 = 1

  1 + 1 = 1 0

- The simplicity of this system makes it possible for digital circuits to carry out arithmetic operations.

SIGNED MAGNITUDE REPRESENTATION

- Example 1: BOTH Positive (+ve)

  Using signed magnitude binary arithmetic, find the sum of 75 and 46.

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

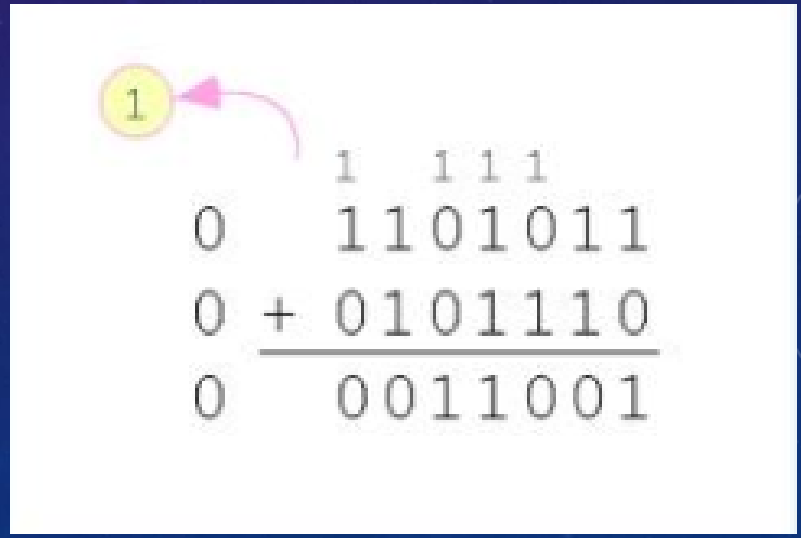
- First, convert 75 and 46 to binary and arrange as a sum, but separate the (positive) sign bits from the magnitude bits.

SIGNED MAGNITUDE REPRESENTATION

- Example 2: with overflow

  Using signed magnitude binary arithmetic, find the sum of 107 and 46.

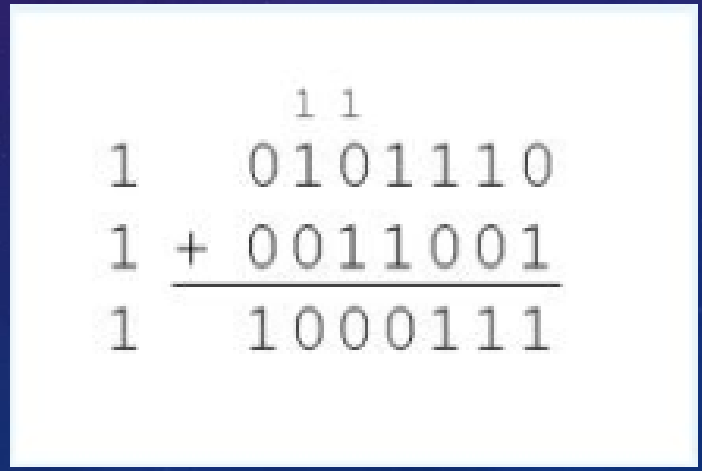
- We see that the carry from the seventh bit overflows and is discarded, giving us the erroneous result: 107 + 45 = 25.



SIGNED MAGNITUDE REPRESENTATION

- The signs in signed magnitude representation work just like signs in pencil and paper arithmetic.
  - Example 3 : BOTH NEGATIVE

    Using signed magnitude binary arithmetic, find the sum of -46 and -25.

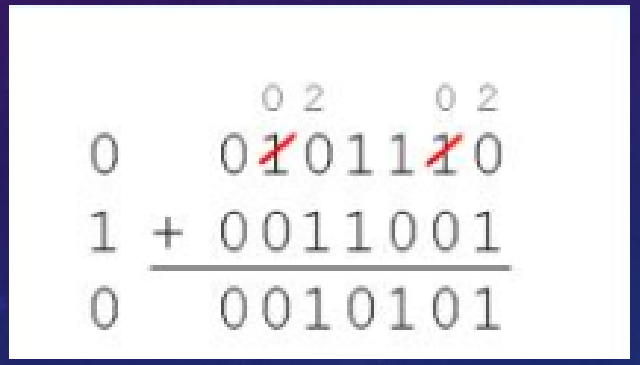


SIGNED MAGNITUDE REPRESENTATION

- Mixed sign addition (or subtraction) is done the same way

  Example 4: Different sign

  Using signed magnitude binary arithmetic, find the sum of 46 and -25.



- The sign of the result gets the sign of the number that is larger.
- Note the ‘borrows’ from the second and sixth bits.

SIGNED MAGNITUDE REPRESENTATION

- Signed magnitude representation is easy for people to understand, but it requires complicated computer hardware.
- Another disadvantage of signed magnitude is that it allows two different representations for zero: positive zero and negative zero.
- For these reason (among others) computers systems employ complement systems for numeric value representation.

ONE'S COMPLEMENT REPRESENTATION

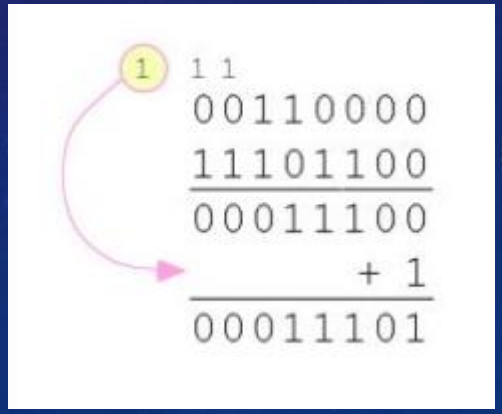
- For example, in 8 bit one's complement,
- POSITIVE 3 is : 0 0 0 0 0 0 1 1
- NEGATIVE 3 is : 1 1 1 1 1 1 0 0
  - In one's complement, as with signed magnitude, negative values are indicated by a 1 in the high order bit
  - Complement systems are useful because they eliminate the need for special circuitry for subtraction.
  - The difference of two values is found by adding the minuend to the complement of the subtrahend.

ONE'S COMPLEMENT REPRESENTATION

- With one's complement addition, the carry bit is "carried around" and added to the sum.
  - Example : Using one's complement binary arithmetic, find the sum of 48 and -19

    We note that 19 in one's complement is 00010011,

    so -19 in one's complement is 11101100



One's Complement

Invert all bits for -ve

11_{10} = 00001011

-11_{10} = 11110100

Addition: involves 'end-around' carry

13 + (-2)

```
      0 0 0 0 1 1 0 1
      1 1 1 1 1 1 0 1  +
  overflow bit -> 1 0 0 0 0 1 0 1 0
end around carry                1
                ---------------
      0 0 0 0 1 0 1 1   (11₁₀)
```

Subtraction: invert the subtrehand and then add

17 - 5 **0 0 0 1 0 0 0 1** **0 0 0 1 0 0 0 1**

0 0 0 0 0 1 0 1 - invert 5 -> 1 1 1 1 1 0 1 0

overflow bit-> 1 0 0 0 0 1 0 1 1

end around carry 1

(12_{10}) 0 0 0 0 1 1 0 0

Note: **there are two forms for 0, 0000...00 and 1111...11**.

ONE'S COMPLEMENT REPRESENTATION

- Although the 'end carry around' adds some complexity, one's complement is simpler to implement than signed magnitude.
- But it still has disadvantage of having two different representations for zero: positive zero and negative zero.
- Two's complement solves this problem.
- Two's complement is the radix complement of the binary numbering system.

TWO'S COMPLEMENT REPRESENTATION

- To express a value in two's complement:
  - If the number is positive, just convert it to binary and you are done
  - If the number is negative, find the one's complement of the number and then add 1.

  Example:
  - In 8 – bit one's complement, positive 3 is : 0 0 0 0 0 0 1 1
  - Negative 3 is one's complement is : 1 1 1 1 1 1 0 0
  - Adding 1 gives us -3 in two's complement form 1 1 1 1 1 1 0 1.

Two's Complement

the number scheme generally used

To form the negative version of a number

1. take the positive binary version of the number
2. invert all the bits
3. add 1

'end around' carry is no longer needed.

ONE'S COMPLEMENT REPRESENTATION

With two's complement arithmetic, all we do is add our two binary numbers. Just discard any carries emitting from the high order bit.

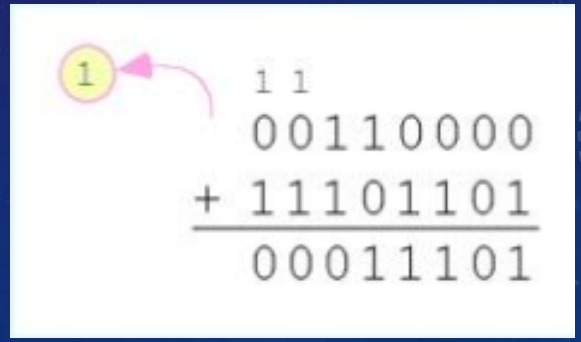
Example:

Using one's complement binary arithmetic, find the sum of 48 and -19.

We note that 19 in one's complement is : 0 0 0 1 0 0 1 1

So -19 in one's complement is : 1 1 1 0 1 1 0 0

And -19 in two's complement is : 1 1 1 0 1 1 0 1



e.g. 2's complement representation of -7 in 8-bit format:

+7	inverted	1 added
0 0 0 0 0 1 1 1	1 1 1 1 1 0 0 0	1 1 1 1 1 0 0 1

Note. A quick way is to:

1. start at the RHS and moving to the left
2. ignore all 0s and the first 1
3. invert all bits to the left of this

OVERFLOW

When we use any finite number of bits to represent a number, we always run the risk of the result of our calculations becoming too the result of our calculations becoming too large to be stored in the computer.

While we cannot always prevent overflow, we can always detect overflow.

In complement arithmetic, an overflow condition is easy to detect.

Overflow

00001000		(8)
01111011	+	(123)
10000011		(-125)

-125 which is totally wrong

This is known as an **overflow**.

Hardware Detection:

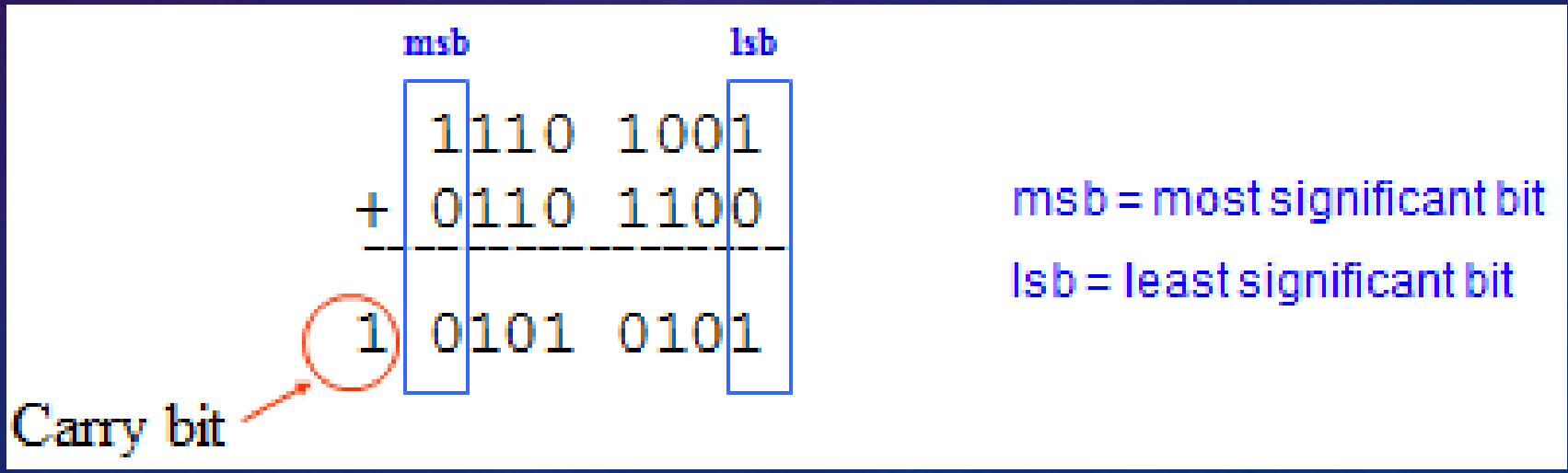
carry into the sign bit XOR carry out of the end = overflow indication bit

Software Detection:

test overflow indication bit and take appropriate action if there is an overflow (result is wrong)

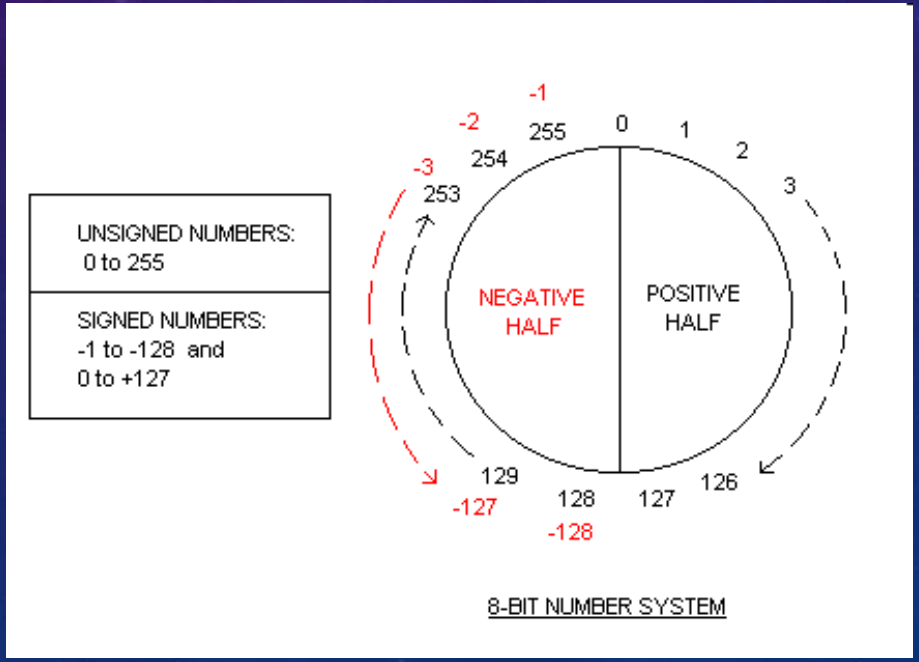
CARRY

- When the result of the calculation produces an extra bit out from the most significant bit position, we call this bit as carry bit.



SIGNED & UNSIGNED NUMBERS

- An 8 bit number system can be used to create 256 combinations (from 0 to 255),
- and the first 128 combinations (0 to 127) represent positive numbers
- and next 128 combinations (128 to 255) represent negative numbers.



SIGNED BINARY NUMBERS

- The first bit, the MSB, is used as the sign bit.
- Take for example, 8-bit signed binary:

8-bit Signed Binary	Decimal Equivalent
01111111	+127
01111110	+126
01111101	+125
...	...
00000011	+3
00000010	+2
00000001	+1
00000000	0
11111111	-1
11111110	-2
11111101	-3
...	...
10000010	-126
10000001	-127
10000000	-128

UNSIGNED BINARY NUMBERS

- All the bits are used as numbers.
- Take for example, the 8-bit unsigned binary:

8-bit Unsigned Binary	Decimal Equivalent
11111111	+255
11111110	+254
11111101	+253
…	…
00000011	+3
00000010	+2
00000001	+1
00000000	0

SPAN & RANGE

- Note that both 8-bit signed and 8-bit unsigned can represent a total of 256 numbers.
- So, they are identical in but different in e:
  - 8-bit signed binary ranges from -128 to +127
  - 8-bit unsigned binary ranges from 0 to 255

Range of Negative numbers

3 bits:

	2’s com	1’s comp	signed mag
000	0	0	0
001	1	1	1
010	2	2	2
011	3	3	3
100	-4	-3	0
101	-3	-2	-1
110	-2	-1	-2
111	-1	0	-3

8 bits: the maximum positive integer is 127; while the maximum negative integer is -127 for the signed magnitude and 1's complement systems, and -128 for the 2's complement system.

SUMMARY

HOW TO CALCULATE 1'S COMPLEMENT FOR A BINARY NUMBER?

The 1's complement can be easily calculated by inverting the 0s & 1s of a given binary number.

Example:

11011000 -> **00100111**

HOW TO CALCULATE 2'S COMPLEMENT FOR A BINARY NUMBER?

1. Find the one's complement by inverting 0s & 1s of a given binary number.
2. Add 1 to the one's complement provides the two's complement.

Example:

00111010

10000101 + 1 = 10000110

HOW TO CALCULATE 1'S COMPLEMENT FOR A DECIMAL NUMBER?

Here is a quick summary of how to find the 1's complement representation of any decimal number *x*.

- If *x* is **positive**, simply convert *x* to binary.
- If *x* is **negative**, write the positive value of *x* in binary
- Reverse each bit.

Example:

a) -34

34 = 32 + 2 = 00100010

-34 = **11011101**

b) +103

+103 = 64 + 32 + 4 + 2 + 1 = 01100111

HOW TO CALCULATE 2'S COMPLEMENT FOR A DECIMAL NUMBER?

Here is a quick summary of how to find the 2's complement representation of any decimal number *x*. Notice the first three steps are the same as 1's complement.

- If *x* is **positive**, simply convert *x* to binary.

- If *x* is **negative,** write the positive value of *x* in binary
- Reverse each bit.
- Add 1 to the complemented number.

a) +101

+101 = 64 + 32 + 4 + 1 = 01100101

b) -109

+109 = 64 + 32 + 8 + 4 + 1 = 01101101

-109 = 10010010 + 1 = 10010011

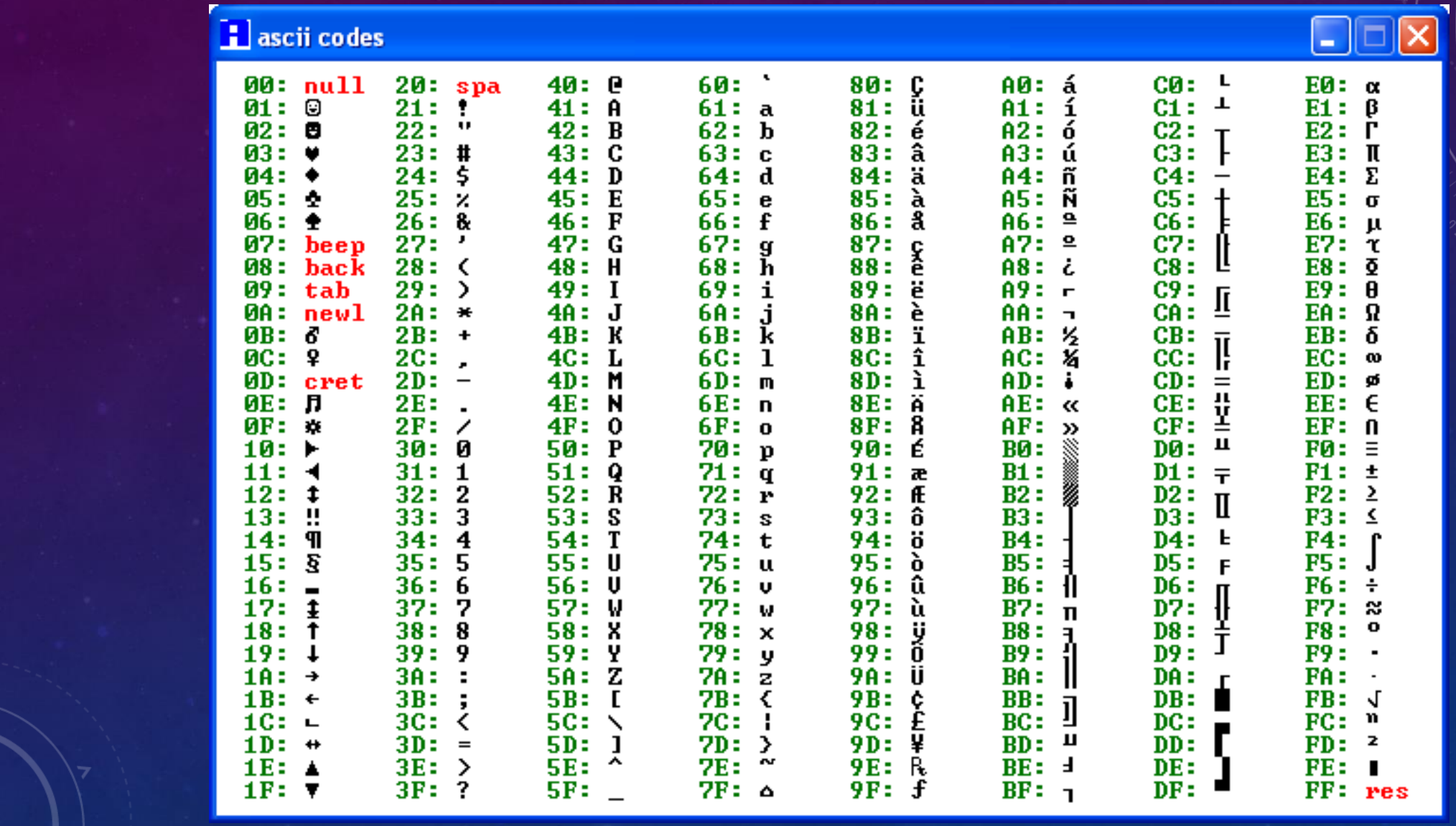
ASCII CODE

- **American Standard Code for Information Interchange (ASCII)**, is a coding standard that can be used for interchanging information, if the information is expressed by English words.
- A Standard ASCII has values from 0 to 127 (7 bits). Represented by 7-bit binary code.
- Extended ASCII uses 0 to 255 (8 bits).

ASCII CODE

- ASCII reserves the first 32 codes (numbers 0–31 decimal) for control characters (NULL, Line Feed)
- Code 32, the "space" character.
- Code 127, the DELETE control
- Codes 33 to 126, known as the printable characters, represent letters, digits, punctuation marks, and a few miscellaneous symbols.

ASCII CODE


ascii codes
00: null 20: spa 40: @ 60: ` 80: Ç A0: á C0: └ E0: α
01: ☺ 21: ! 41: A 61: a 81: ü A1: í C1: ┴ E1: ß
02: ☻ 22: " 42: B 62: b 82: é A2: ó C2: ┬ E2: Γ
03: ♥ 23: # 43: C 63: c 83: â A3: ú C3: ├ E3: π
04: ♦ 24: $ 44: D 64: d 84: ä A4: ñ C4: ─ E4: Σ
05: ♣ 25: % 45: E 65: e 85: à A5: Ñ C5: ┼ E5: σ
06: ♠ 26: & 46: F 66: f 86: å A6: ª C6: ╞ E6: µ
07: beep 27: ' 47: G 67: g 87: ç A7: º C7: ╟ E7: τ
08: back 28: (48: H 68: h 88: ê A8: ¿ C8: ╚ E8: Φ
09: tab 29:) 49: I 69: i 89: ë A9: ⌐ C9: ╔ E9: Θ
0A: newl 2A: * 4A: J 6A: j 8A: è AA: ¬ CA: ╩ EA: Ω
0B: ♂ 2B: + 4B: K 6B: k 8B: ï AB: ½ CB: ╦ EB: δ
0C: ♀ 2C: , 4C: L 6C: l 8C: î AC: ¼ CC: ╠ EC: ∞
0D: cret 2D: - 4D: M 6D: m 8D: ì AD: ¡ CD: ═ ED: ø
0E: ♫ 2E: . 4E: N 6E: n 8E: Ä AE: « CE: ╬ EE: ε
0F: ☼ 2F: / 4F: O 6F: o 8F: Å AF: » CF: ╧ EF: ∩
10: ► 30: 0 50: P 70: p 90: É B0: ░ D0: ╨ F0: ≡
11: ◄ 31: 1 51: Q 71: q 91: æ B1: ▒ D1: ╤ F1: ±
12: ↕ 32: 2 52: R 72: r 92: Æ B2: ▓ D2: ╥ F2: ≥
13: ‼ 33: 3 53: S 73: s 93: ô B3: │ D3: ╙ F3: ≤
14: ¶ 34: 4 54: T 74: t 94: ö B4: ┤ D4: ╘ F4: ⌠
15: § 35: 5 55: U 75: u 95: ò B5: ╡ D5: ╒ F5: ⌡
16: ▬ 36: 6 56: V 76: v 96: û B6: ╢ D6: ╓ F6: ÷
17: ↨ 37: 7 57: W 77: w 97: ù B7: ╖ D7: ╫ F7: ≈
18: ↑ 38: 8 58: X 78: x 98: ÿ B8: ╕ D8: ╪ F8: °
19: ↓ 39: 9 59: Y 79: y 99: Ö B9: ╣ D9: ┘ F9: ∙
1A: → 3A: : 5A: Z 7A: z 9A: Ü BA: ║ DA: ┌ FA: ·
1B: ← 3B: ; 5B: [7B: { 9B: ¢ BB: ╗ DB: ■ FB: √
1C: ∟ 3C: < 5C: \ 7C: ¦ 9C: £ BC: ╝ DC: ▄ FC: ⁿ
1D: ↔ 3D: = 5D:] 7D: } 9D: ¥ BD: ╜ DD: ▌ FD: ²
1E: ▲ 3E: > 5E: ^ 7E: ~ 9E: ₧ BE: ╛ DE: ▐ FE: ■
1F: ▼ 3F: ? 5F: _ 7F: ⌂ 9F: ƒ BF: ┐ DF: ▀ FF: res