REPUBLIC OF CAMEROON

Peace- Work-Fatherland

REPUBLIQUE DU CAMEROUN

Paix-Travail-Patrie

THE UNIVERSITY OF BAMENDA

UNIVERSITE DE BAMENDA

THE COLLEGE OF TECHNOLOGY

(COLTECH)

ECOLE DE TECHNOLOGIE

COMPUTER ENGINEERING

SOFTWARE ENGINEERING

INFORMATION SYSTEMS: UML METHODOLOGY

Failing to plan is planning to fail.

PRESENTED BY:

MBAH LESKY TAGWANG

UBA21PB015

2021/2022

DIGITAL ARITHMETIC

1. Binary Addition

Binary addition works with the same principle as decimal and the other bases.

- In decimal and other bases, '0' added to a number equals that number.
- If '1' is added to a number, the next higher number in that system is the result.
- If a digit is added to another digit and the result exceeds the greatest digit in that system (the result will be a 2-digit number). The least significant (right) digit is written and the most significant (left) digit is carried. e.g. in base 10, 7 + 5 = 12 which exceeds 9 (the greatest digit in decimal). So, we right down 2 and carry 1.
- Here are the basic rules of binary addition:

		Sum	Carry
-	0 + 0 = 0	0	0 (no carry)
-	0 + 1 = 1	1	0
-	1 + 0 = 1	1	0
-	1 + 1 = 10	0	1
-	1 + 1 + 1 = 11	1	1

1.1. Addition of larger bit binary numbers

The addition of larger bit integers or floating-point numbers is performed in a column wise manner while using the basic addition rules. The numbers to be added are stacked up with their LSB (least significant bit) aligned horizontally. Using the basics rules, the LSB of the 2 numbers are added and their carry is added to the NSB (next significant bit). Here is an illustration. Consider 2 binary numbers with 5 bits $A_4A_3A_2A_1A_0$ AND $B_4B_3B_2B_1B_0$, where A_0 and B_0 represent the LSB of the 2 numbers and A4 and B4 representing the MSB of the 2 numbers. We start adding the LSB (A_0 and B_0) using the basic rules of addition and add its carry to the result of the addition of the next significant bits (A_1 and B_1). The process continues until the MSB where the carry of their sums become the MSB of the result. See illustration below (S_N is the n^{th} sum and C_N the n^{th} carry):

* Both numbers must not be of the same number of bits.

i.

			C_0	
A_4	A_3	A_2	A_1	A_0
B_4	B_3	B_2	B_1	B_0
				S_0

ii.

		C_1	C_0	
A_4	A_3	A_2	A_1	A_0
B_4	B_3	B_2	B_1	B_0
			S_1	S_0

iii.

	C_2	C_1	C_0	
A_4	A_3	A_2	A_1	A_0
B_4	B_3	B_2	B_1	B_0
		S_2	S_1	S_0

iv.

C_3	C_2	C_1	C_0	
A_4	A_3	A_2	A_1	A_0
B_4	B_3	B_2	B_1	B_0
	S_3	S_2	S_1	S_0

iv.

	C_3	C_2	C_1	C_0	
	A_4	A_3	A_2	A_1	A_0
	B_4	B_3	B_2	B_1	B_0
C_4	S_4	S_3	S_2	S_1	S_0

EXAMPLE: addition of 11011 and 1110

Carry:	1	1	1	1	0	
		1	1	0	1	1
			1	1	1	0
Sum:	1	0	1	0	0	1

1.2. Addition using 2's complement method

2's complement is used in the following cases:

- Addition of positives numbers
- Addition of negative numbers
- Addition of a positive and a negative number

The 2's complement of a decimal is gotten by;

- Converting that decimal to binary
- If the number is positive, then it is already in 2's complement. If it is negative, we proceed by;
- Switching all 1's to 0's and 0's to 1's (1's complement)
- Add 1 to the result.

Addition can then be carried out normally as above.

EXAMPLES

1. **Add +12 and +23 using 8 bits**

   Since both numbers are positive, +12 = 00001100 and +23 = 00010111

$$\begin{array}{r} 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0 \\ +\ 0\ 0\ 0\ 1\ 0\ 1\ 1\ 1 \\ \hline 0\ 0\ 1\ 0\ 0\ 0\ 1\ 1 \end{array}$$

   00100011 = 35

2. **Add +61 and -45 using 8 bits**

   +61 = 00111101 and +45 = 00101101. -45 = 11010010 + 1 = 11010011

$$\begin{array}{r} 0\ 0\ 1\ 1\ 1\ 1\ 0\ 1 \\ +\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 1 \\ \hline 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0 \end{array}$$

   The final carry is discarded. +61 + -41 = +16 = 0001000

3. **Add -15 and -33 using 8 bits**

   + 15 = 00001111, -15 = 11110000 + 1 = 11110001

   +33 = 00100001 -33 = 11011110 + 1 = 11011111

$$\begin{array}{r} 1\ 1\ 1\ 1\ 0\ 0\ 0\ 1 \\ +\ 1\ 1\ 0\ 1\ 1\ 1\ 1\ 1 \\ \hline 1\ 1\ 0\ 1\ 0\ 0\ 0\ 0 \end{array}$$

   The final carry can be discarded. -15 + - 33 = -48 = 1101000 in 2's complement.

2's complement notation can be used to perform addition when the result lies from -2^{n-1} to $+2^{n-1} - 1$, where n is the number of bits. e.g. the range for 8 bits is – 128 to + 127 (-2^7 to $+2^7 - 1$)

1.3. Binary addition of floating-point numbers

Just like in the addition of floating-point decimals, binary floating-points can be added like integers (like the binary point isn't there) only difference that the binary point maintains its position in the result

EXAMPLE: add +45.75 and +20.625 using 8 bits for the integer part and 4 bits for the fractional part

+45.75 = 00101101.1100 and +20.625 = 00100010.0110. +45.75 + (+ 20.625) = + 66.375

```
  0 0 1 0 1 1 0 1 . 1 1 0 0
+ 0 0 0 1 0 1 0 0 . 1 0 1 0
  ---------------------------
  0 0 1 0 0 0 1 0 . 0 1 1 0
```

+66.375 = 00100010.0110

1.4. Addition of other bases.

We can add other bases by simply converting them to binary and adding, then convert the resultant back to the base.

EXAMPLE: add $AF1.B3_{16}$ + $FFF.E_{16}$

AF1.B3 = 101011110001.10110011 = 2801.69921875 and FFF.E = 111111111111.1110 = 4095.875

0's are used to make them have equal length.

```
    1 0 1 0 1 1 1 1 0 0 0 1 . 1 0 1 1 0 0 1 1
+   1 1 1 1 1 1 1 1 1 1 1 1 . 1 1 1 0 0 0 0 0
  -------------------------------------------------
  1 1 0 1 0 1 1 1 1 0 0 0 1 . 1 0 0 1 0 0 1 1
```

0001 1010 1111 0001 . 1001 0011 = 1AF1.93 = 6897.57421875

1.5. BCD Addition

BCD numbers are added in excess-3 code form. the steps are as follows:

- Add '0011' (3) to each 4-bit group to give its excess-3 code form.
- Add the two numbers using the basic laws of addition
- Add '0011' to all those four-bit groups that produce a carry, and subtract '0011' from all those four-bit groups that do not produce a carry during addition.
- The result obtained is in excess-3 form

EXAMPLE: add 548 and 271 using BCD

In BCD, 548 = 0101 0100 1000 and 271 = 0010 0111 0001

In excess-3 code (adding 3 to each '0011') 548 = 1000 0111 1101 and 271 = 0101 1010 0100

```
      1 0 0 0  0 1 1 1  1 1 0 1
+     0 1 0 1  1 0 1 0  0 1 0 0
  ---------------------------------
BCD   1 1 1 0  0 0 1 0  0 0 0 1
E3C   1 0 1 1  0 1 0 1  0 1 0 0
```