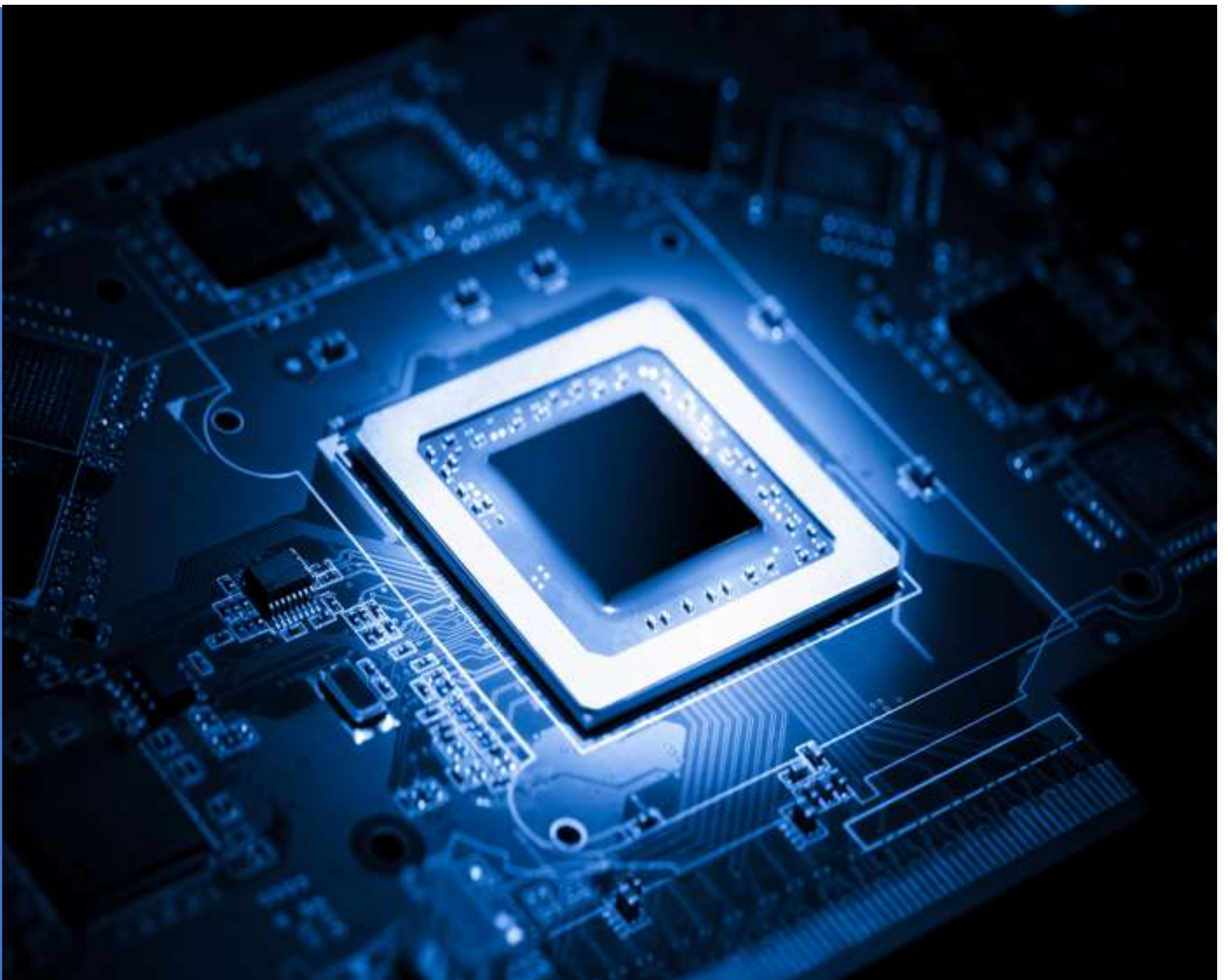
COMPUTER ORGANIZATION



Chapter Four

Data Representation

Dr. Zied & Dr. Jamal A.L. Salah

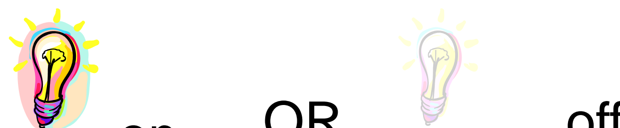
1. Introduction

The CPU processes information obtained from the primary memory and returns the results to memory. There is usually a block of information that the programmer sees as being processed in any one operation and moved to or from memory. This is called a **word**.

A **word** may represent a number to be used in numerical calculations or one or more characters of nonnumeric information. The length of the word differs from machine to machine. On most machines, however, a word consists of a number of bits, the most common lengths being between 8 and 64 bits.

The word "bit" is a contraction of "binary digit", meaning a digit that can take one of the two values 0 or 1, just as a decimal digit can take one of the ten values 0, 1, 2… 9. Although human beings commonly use decimal digit to represent numbers, machines commonly use binary digits because most physical devices used in machines can retain one of two states most reliably, for example, an on or off switch, a positive or negative voltage, north or south magnetization.

bit



byte

A word, consisting of a number of bits, may be used to represent anything that the user cares to have it represent. Each bit of an N-bit word can be a 0 or 1 independently, so that a word can assume any of 2^N different states. With one word, the user can represent one out of not more than 2^N different objects, such as numbers, letters, species of trees, etc. The most common use of a word is for number and character representation. Much of today's computer terminology reflects the fact that, in the early days, computers handled mainly numbers. If a word consists of N bits, it is usually drawn as figure below.

A computer word

The bits have been numbered arbitrarily from 0 to N-1. Bit N-1 is often referred to as the left-hand bit or most-significant bit and bit 0 as the right-hand bit or least-significant bit. The word is referred to as being N bit long.

When a group of bits is used to represent a number, the so-called natural binary-coding scheme is usually employed. Because a single bit can represent only two states, the value, or weight, of each bit or digit in a word is a power of two–thus the bits in word, starting from the right-hand side, include the number of 1's, 2's, 4's, 8's, etc., in the number. Thus if 7 bits, say

$$b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0$$

Are used to represent a number, the value of that number is

$$b_6*2^6+ b_5*2^5+ b_4*2^4+ b_3*2^3+ b_2*2^2+ b_1*2^1+ b_0*2^0$$

Dr. Ziad & Dr. Jalal A. L. Salah

The largest value that this can take is the binary number 1111111, representing the decimal value 127, that is

$$1*2^6+ 1 *2^5+ 1 *2^4+ 1 *2^3+ 1 *2^2+ 1 *2^1+ 1*2^0$$

$$64 + 32 + 16 + 8 + 4 + 2 + 1$$

$$127$$

The smallest value is the binary number 0000000, representing the value 0. Obviously, such a representation handles positive integers only, but one common way of handling negative numbers is to use one more bit to indicate the sign.

This is frequently the left-most bit, with a 0 indicating a positive number and 1 a negative number. Thus, numbers between -127 and +127 could be representing in 8 bits, the first being used for a sign and the remaining 7 for the magnitude of the number. This is called the sign-magnitude system.

Dr. Jalal e A.L. Salah

0	**1**	**1**	**1**	**1**	**1**	**1**	**1**
B7 (Sign bit)	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**	**b0**

Represent the value +127

1	**1**	**1**	**1**	**1**	**1**	**1**	**1**
B7 (Sign bit)	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**	**b0**

Represent the value -127

2. Character Code

Besides representing numbers, words are often used to represent nonnumeric data such as alphabetic characters. In program translation, business data processing, and work such as the differentiation of algebraic expressions or word-frequency studies, it is often necessary to represent natural language text such as English or American.

To do this, each character (letter, digit, punctuation mark, etc.) is represented by a different bit pattern. The number of bits needed to represent a character is determined by the number of different characters. N bit can represent up to 2^N characters. For example, 64, 128, Or 256 characters can be represented with 6, 7 or 8 bits, respectively.

If 8 bits are used to represent each character, but the word length is much greater than 8, say 32, it is clearly inefficient to store just one character in each word. In this case, several characters are packed into a word. Thus four 8-bit characters may be packed into a 32-bit word. If 6-bit characters are used, similar packing occurs. For example, the CDC CYBER 170 series packs ten 6-bit characters into its 60-bit word.

Modern computers need to represent more than 64 characters because it is convenient to represent upper- and lower-case letters, decimal digits, and a number of special characters. Because 7 turn out to be an awkward number of bits, 8 bits are used. One such code is known as ASCII (American Standard Code for Information Interchange). It uses only 7 of the bits for information. Table below gives the ASCII character set.

Dr. Zied & Dr. Jalal & M.L. Salah

Bits 3 to 6	Bits 0 to 2							
	000	001	010	011	100	101	110	111
0000	Nul	Soh	Stx	Etx	Eot	Enq	Ack	Bel
0001	Bs	Ht	Nl	Vt	Np	Cr	So	Si
0010	Dte	Dc1	Dc2	Dc3	Dc4	Nak	Syn	Etb
0011	Can	Em	Sub	Esc	Fs	Gs	Rs	Us
0100	Sp	!	"	#	$	%	&	'
0101	(	)	*	+	,	-	.	
0110	0	1	2	3	4	5	6	7
0111	8	9	:	;	<	=	>	?
1000	@	A	B	C	D	E	F	G
1001	H	I	J	K	L	M	N	O
1010	P	Q	R	S	T	U	V	W
1011	X	Y	Z	[	\	]	'	_

Single character entries in this table are printable characters (on those devices that have all such characters available). The two-letter and three-letter entries represent control characters and other nonprinting characters that cause various actions on typical terminals. For example, the **sp** and **bs** represent space and back space, respectively, whereas **cr** represents carriage return.

Not unusually, the largest computer manufacture has its own standard, known as EBCDIC (Extended BCD Interchange Code). It is an outgrowth of an earlier, very common representation of the uppercase letters, then ten digit 0 to 9, and the twelve special characters + - * / , . ( ) $ = ' and space (forty-eight characters altogether) that used 6 bit.

That code was known as the BCD code (Binary Code Decimal). With a few variations, it was standard for many computers and could be found in many machine manuals. The ECDIC code is an 8-bits code that can

Dr. Zied & Dr. Jalal

handle lower-case alphabetic characters and many other special characters in addition to those in the BCD code.

3. Information

❖ **Data format (data representation inside computer):**

The information can be stored in the memory can be divided into:

1- Data

2- Instruction

❖ **Data format:**

There are several formats can used to store the data it is to operate on. The formats may be summarized as follows:

1- Number format

A- Integer number

B- Real number

2- Alphanumeric codes

0010100101001010010101010
0101010101010100101000
0101111010101010101111
1111101111111011111111111
0010100101001010010101010
0101010101010100101000
0101111010101010101111

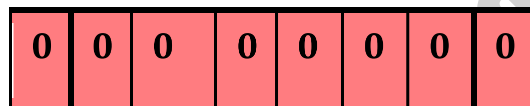
& Dr. Jalal & A.L. Salah

4. Integer Number

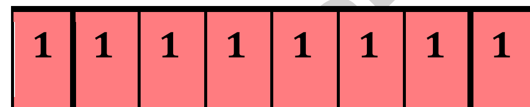
The binary number system is the most conventional internal representation for number in a digital computer. If there are n-bits in a group, the number of possible combinations of 0's, and 1's is 2n

Example

If there are 8-bits, so the number of possible combinations is 28 =256. If the group of bits are used to represent the non-signed integer, the integer number from 0 to 2 n-1 can be represented (8 bits can be representing the non-signed integer numbers from 0 to 255).



Smallest Number



Largest Number

There are three widely used techniques for representations both positive and negative number (signed number)

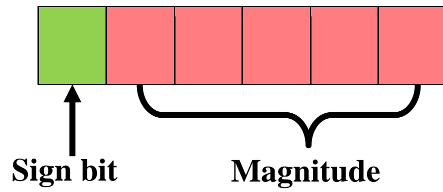
1- sign magnitude format

2- 1's complement format

3- 2's complement format

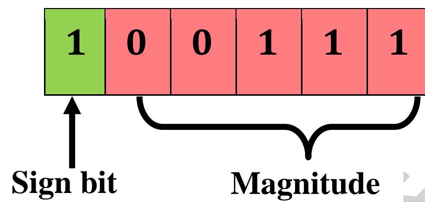
❖ Signed magnitude format

A signed value (negative and positive) is written by writing its magnitude (absolute value) and then placing a sign (1 for negative and 0 for positive) to the left of the magnitude. Therefore, if a number us to be stored in n-bits, the magnitude is place in the n-1 rights most bits and let the left bit indicate the sign (0 = +, 1 = -)

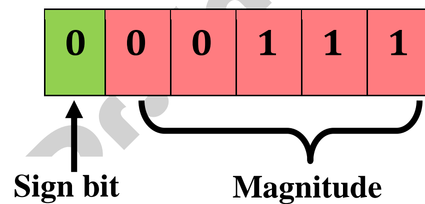


Example:

Represent (-7, +7) in the word size equal to 6 bits by using signed magnitude method.



represent -7



represent +7

The range of integer that can be express in a group of 8-bits by using sign magnitude method is.

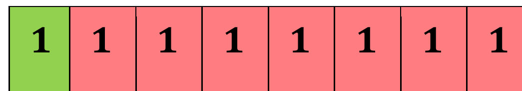
$$(2^{n-1}-1) \text{ to } +(2^{n-1}-1)$$

$$(2^{8-1}-1) \text{ to } +(2^{8-1}-1)$$

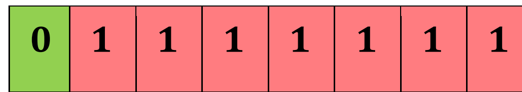
$$(2^{7}-1) \text{ to } +(2^{7}-1)$$

$$(128-1) \text{ to } +(128-1)$$

$$(127) \text{ to } +(127)$$



Large negative number



Large positive number

In general, for N-bits, the complete range that can be represented in signed magnitude is form:

$$[(-2^{(n-1)} -1) \text{ to } (+2^{(n-1)} -1]$$

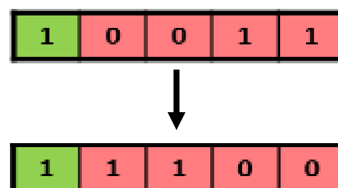
❖ **Signed 1's complement**

To represent the positive number in 1's complement, the left most bit is used to represent the positive sign (0) and the magnitude place in (n-1) right most bits.

To represent the negative values in 1's complement are obtained by complementing each bit of the representation of the corresponding positive value.

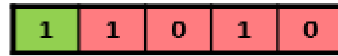
Example:

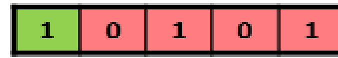
Represent -3 and -10 by using 1's complemented by using 5-bit.



To represent -3

Dr. Zied & Dr. Jalal & A.L. Salah





To represent -10

The range of integer that can express in a group of n-bits by using 1's complement is.

$$(-2^{(n-1)} - 1) \text{ to } (+2^{(n-1)} - 1)$$

Example:

What are the largest signed number and smallest signed number in 4 bits word size by using 1's complement?

Largest number $= (+2^{(n-1)} - 1)$

$= (+2^{(4-1)} - 1)$

$= (+2^{(3)} - 1)$

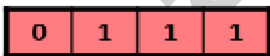
$= +7$

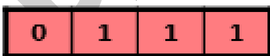
Smallest number $= (-2^{(n-1)} - 1)$

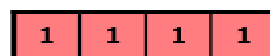
$= (-2^{(4-1)} - 1)$

$= (-2^{(3)} - 1)$

$= -7$







To represent +7

To represent -7

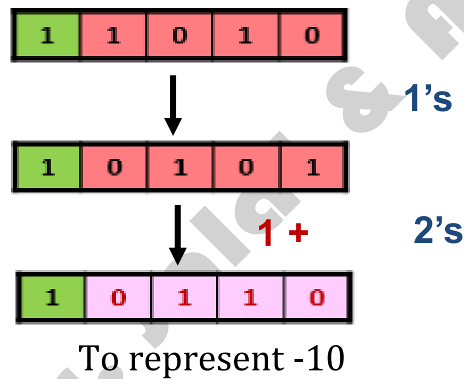
Dr. Zied & Dr. Jalal & A.L. Salah

❖ Signed 2's complement

To represent the positive number in 2's complement, the left most bit is used to represent the positive sign (0) and the magnitude place in (n-1) right most bits. So 2's complement is like the signed magnitude and 1's complement when represent the positive number.

Example:

Represent (-10) by using 2's complemented by using 5-bit.



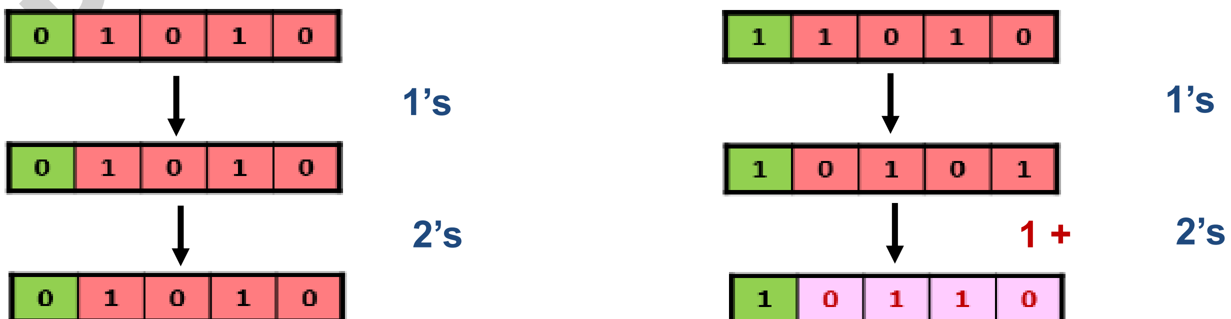
To represent -10

The complete range of values of n-bits with one signed bit by using 2's complement is.

$$(-2^{(n-1)}) \text{ to } (+2^{(n-1)}-1)$$

Example:

Represent (+10, -10) by using 2's complemented using 5-bit



Dr. Zied & Dr. Jala & A.L. Salah

Example:

The complete range of values of 4 bits word size with signed bit by using 2's complement

$$-2^{(n-1)} \text{ to } +(2^{(n-1)}-1)$$

$$-2^{(4-1)} \text{ to } +(2^{(4-1)}-1)$$

$$-2^{(3)} \text{ to } +(2^{(3)}-1)$$

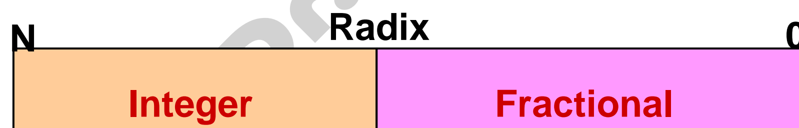
$$-8 \text{ to } +7$$

5. Real Number

❖ **Fixed-Point Representation**

In fixed-point representation, a specific radix point - called a decimal point in English and written ".", is chosen so there is a fixed number of bits to the right and a fixed number of bits to the left of the radix point.

The bits to the left of the radix point are called the integer bits. The bits to the right of the radix point are called the fractional bits



Example:

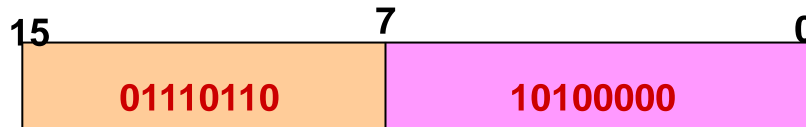
Assume a 16-bit fractional number with 8 magnitude bits and 8 radix bits, which is typically represented as 8.8 representations.

To encode 118.625, first find the value of the integer bits. The binary representation of 118 is **01110110**, so this is the upper 8 bits of the 16-bit number.

The fractional part of the number is represented as 0.625×2^n where n is the number of fractional bits.

Dr. Zied & Dr. Jalal & A.L. Salah

Because 0.625 x 256 = 160, you can use the binary representation of 160, which is **10100000**, to determine the fractional bits. Thus, the binary representation for 118.625 is 0111 0110 1010 0000.



The major advantage of using fixed-point representation for real numbers is that fixed-point adheres to the same basic arithmetic principles as integers.

The disadvantage of using fixed-point numbers is that fixed-point numbers can represent only a limited range of values, so fixed-point numbers are susceptible to common numeric computational inaccuracies.

Dr. Zied & Dr. Jalal & A. L. Salah

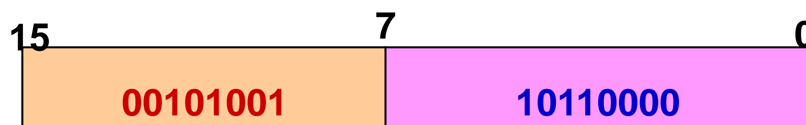
Example:

Represent the real number 41.6875 by using fixed point method

$41 \div 2 = 20$	1
$20 \div 2 = 10$	0
$10 \div 2 = 5$	0
$5 \div 2 = 2$	1
$2 \div 2 = 1$	0
$1 \div 2 = 0$	1

$(41)_{10} = (101001)_2$

$0.6875 * 2 = 1.3750$
$0.3750 * 2 = 0.7500$
$0.7500 * 2 = 1.5000$
$0.5000 * 2 = 1.000$
$(0.6875)_{10} = (0.1011)_2$



41.6875 = 101001.1011

❖ 2-Floating-Point Representation

Using 32 bits to represent a number, positive or negative, the range of possible values is large but there are circumstances when bigger number representations are needed. The way to do this is to use floating point numbers.

The reason for using floating point representation is that the range of possible values is much greater.

Floating point representation is similar to scientific notation and details of how values are represented vary from one machine to another. The number uses a 32-bit string. This string has 3 distinct parts:

The sign bit	**1 bit. With the value 1 for negative and 0 for positive.**
The exponent	**8 bits**
The mantissa	**23 bits**

Example:

Represent the number $(-435.25)_{10}$ in a float point number.

1- The binary representation of $(-435.25)_{10}$ = -110110011.01

2- Sign bit 1 negative.

3- The mantissa is the number begins encoded. Before the number is encoded the point is moved or floated left or right until the value of the number is in range ($1 < n < 2$).

This is known as normalization. Since the first digit is now always going to be 1 there is no need to encode this digit. The mantissa only includes the digits after the point and the leading 1 is assumed.

The number in un-normalized form is -110110011.01

The number in normalization form is -1.1011001101 * 2^8

4- The exponent is shown with excess 127. This means that the machine exponent is the actual exponent with 127 added. This has the effect of giving a range of 0 to 255 for the machine representation while the range of actual values is -128 to +127. If the excess was not used there would have to be a mechanism for showing a negative exponent.

In the example the actual exponent is 1000 (decimal 8). The machine representation is:

0000 1000 (8) + 0111 1111 (127) = 1000 0111 (decimal 135)

Sign	Exponent	Mantissa
1	10000111	10110011010000000000000

Dr. Zied & Dr. Jalal & A.L. Salah

Example:

Represent the number $(5.890625)_{10}$ in a float point number.

1- The binary number $(5.890625)_{10} = (101.111001)_2$

2- The sign bit is 0 for positive.

3- Move the point 2 places to the left:

$1.01111001 * 2^2$ the digits after the point are the mantissa.

4- The exponent will be $(2 + 127 = 129)_{10}$ convert this number to binary will be $(10000001)_2$

Sign	Exponent	Mantissa
0	10000001	01111001000000000000000

Dr. Zied & Dr. Jalal & M. Salah