UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Representación de datos en la computadora. Aritmética.

Prof. Silvana Panizzo

Segunda parte

Banderas lógicas / Flags

Los Flags son:

- ✓ **Flag V (Overflow):** indica si hubo o no overflow o desborde de registro, después de haberse llevado a cabo la operación. Si almacena un valor “1” significa que hubo overflow; si almacena un valor “0” significa que no hubo overflow.
- ✓ **Flag C (Carry):** indica el acarreo o carry que se produce en la operación. Se pone en “1” si Cn es igual a “1”, o se pone en “0” si Cn es igual a “0”.
- ✓ **Flag S (Sign):** indica el signo del resultado de la operación. Se pone en “1” si el resultado es negativo, o se pone en “0” si el resultado es positivo.
- ✓ **Flag Z (Zero):** indica si el resultado de la operación es 0 o distinto de 0. Se pone en “1” si el resultado es 0, o se pone en “0” si el resultado es distinto de 0.
- ✓ **Flag P (Parity):** si la cantidad de bits 1 del resultado de la operación es impar se pone un “1”, si la cantidad de bits 1 del resultado de la operación es par se pone un “0”.
- ✓ **Flag A (Auxiliary carry):** indica el acarreo o carry que se produce del 4 bit. Se pone en “1” si C4 es igual a “1”, o se pone en “0” si el resultado es negativo.

Desbordamiento

Overflow

Para las formas de variable vistas (Enteras Sin Signo (Naturales), Enteras con Signo y Reales, cuando el resultado de una operación supera los límites definidos por el rango (mayor que el máximo o menor que el mínimo), entonces es incorrecto.

Este error se conoce como *overflow* **de resultado** y actualiza un bit (*flag* de *overflow*) en un registro asociado a la ALU, conocido como registro de estado *o status register*.

Underflow

Solo para variables Reales, cuando el resultado de una operación cae en el hueco definido por los límites inferiores del rango (valores muy cercanos a cero, del que se excluye el cero). El error se conoce como *underflow* **de resultado**.

Esquema de registros para suma de 8 bits

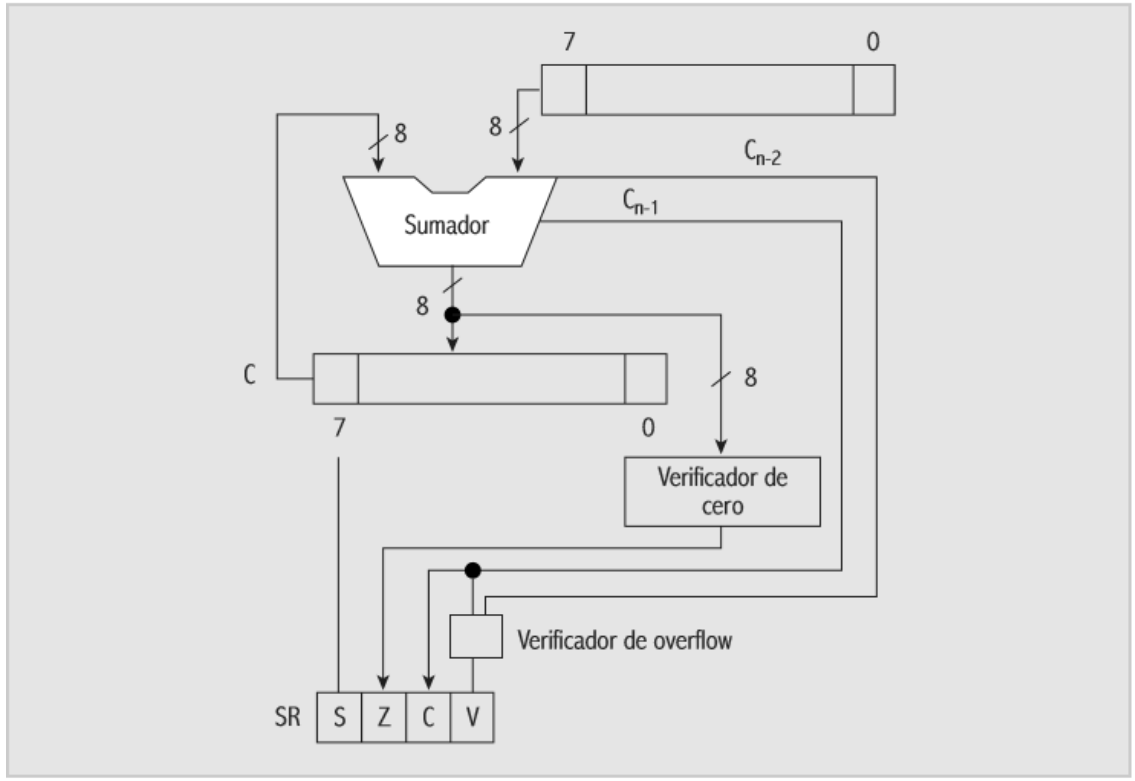


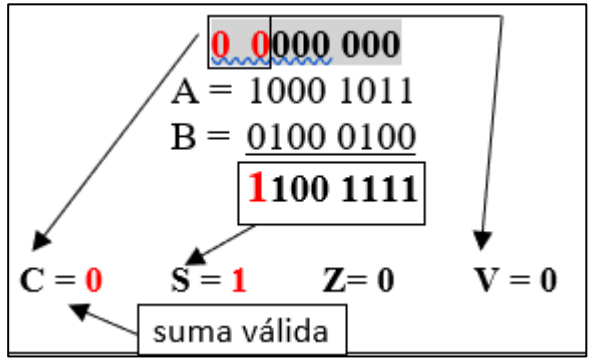
Fig. 4.1. Esquema de registros para suma de 8 bits.

Grafico del libro Arquitectura de Computadoras de Patricia Quiroga. Capitulo 4

ALU: Unidad Aritmético lógica

$A = -117_{(10)}$ $B = +68_{(10)}$

(-117) + 68 = -49



Operaciones en Convenio Punto Fijo con C2

✓ **Suma paso a paso**

$116_{(10)} + 5_{(10)} = 121_{(10)}$ n=8 bits

1) Paso a binario los operandos.

$116_{(10)} = 1110100_{(2)}$

$5_{(10)} = 101_{(2)}$

2) Completo el formato a n=8 bits

$116_{(10)} = 1110100_{(2)} = 01110100_{(2)}$

$5_{(10)} = 101_{(2)} = 00000101_{(2)}$

3) Realizar la suma en formato a n=8 bits

$$\begin{array}{r} 00000100 \\ 01110100 \\ + \quad 00000101 \\ \hline 01111001 \end{array}$$

4) Verifico el resultado y el signo obtenido

$01111001(2) = +121_{(10)}$

5) Obtener valores Flags

S	Z	V	C
0	0	0	0

Operaciones en Convenio Punto Fijo con C2

✓ **Resta por C2 paso a paso**

$116_{(10)} - 5_{(10)} = 111_{(10)}$ n=8 bits

1) Paso a binario los operandos.

$116_{(10)} = 1110100_{(2)}$

$5_{(10)} = 101_{(2)}$

2) Completo el formato a n=8 bits

$116_{(10)} = 1110100_{(2)} = 01110100_{(2)}$

$5_{(10)} = 101_{(2)} = 00000101_{(2)}$

3) C2 a los valores negativos

$5_{(10)} = 00000101_{(2)}$

$-5_{(10)} = 11111010_{(2)}$ Invierto bits

$1_{(2)}$ Sumo 1

$11111011_{(2)}$

4) Realizar la suma en formato a n=8 bits

$$\begin{array}{r} 11110000 \\ 01110100 \\ + \quad 11111011 \\ \hline 01101111 \end{array}$$

5) Verifico el resultado y el signo obtenido

01101111(2) = + $111_{(10)}$

5) Obtener valores Flags

S	Z	V	C
0	0	0	1

Actividad

Ejercicios:

Realizar las siguientes operaciones en formato de n=8 bits en convenio punto fijo con C2 para los valores negativos. Indique el resultado de las banderas lógicas en cada caso.

1) 124(10) + 15 (10) =
2) -7(10) – 6 (10) =
3) 126(10) + 1 (10) =
4) 256(10) – 256(10) =

- ✓ Realizar TP 3 Enunciado en el Campus.

Endianness - Extremidad

Referencia al orden en el que se almacenan los bytes dentro de la memoria.

Si tenemos la siguiente cadena en hexadecimal 90AB12CD para almacenar en la memoria, según el orden:

✓ Big Endian → Motorola

90AB12CD

Representación de la Dirección o posición de memoria	Valor almacenado (hexadecimal)	Valor almacenado (binario)
xxx0	90	10010000
xxx1	AB	10101011
xxx2	12	00010010
xxx3	CD	11001101

✓ Little Endian → Intel

90AB12CD

Representación de la Dirección o posición de memoria	Valor almacenado (hexadecimal)	Valor almacenado (binario)
xxx0	CD	11001101
xxx1	12	00010010
xxx2	AB	10101011
xxx3	90	10010000

Atención:
No se invierten los octetos o bytes.
Se almacena CD, no DC;
Se almacena 12, no 21;
y así sucesivamente…

✓ Los dos → ARM y Power PC

Código de representación binaria de números decimales

- ✓ Convenio de coma o punto flotante IEEE 754 precisión simple

$$\pm M \times B^{\pm E}$$

M: Mantisa
B: La base del sistema
E: Exponente

- ✓ Se pueden representar números enteros y decimales.
- ✓ Utiliza mantisa fraccionaria normalizada con bit implícito.
- ✓ La base del sistema es siempre 2, pero no se registra en el formato.
- ✓ El exponente, es sesgado (o excedido) y es un numero tal que permita que ese campo no tenga signo.

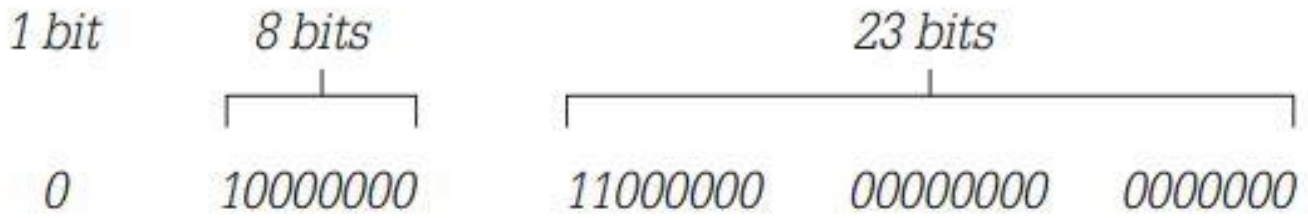
1 bit	8 bit	23 bit
Signo	**Característica**	**Mantisa**
	$\pm E + 127$ (Exponente sesgado) (Exceso a $2^{n-1}-1$)	1,(bit implícito)

Código de representación binaria de números decimales

Convenio de coma o punto flotante IEEE 754 precisión simple

Si ahora representamos $+3,5_{(10)} = +11,1_{(2)} = +1,11_{(2)} \cdot 10_{(2)}^{+1(2)}$ calculamos la *característica* como (+1) + 01111111 = 10000000

El signo es positivo, por lo tanto, el formato queda:



Código de representación binaria de números decimales

Como queda representada en memoria la variable A definida como Real en Simple precisión:

A = - 53,25

Primero debemos pasar el decimal a Binario natural. TENER EN CUENTA QUE PUNTO FLOTANTE NO TRABAJA CON COMPLEMENTO.

Da como resultado en binario A = - 110101,01

Luego normalizamos: - 1,1010101 x 10^{101} donde el exponente p está también en binario que es el 5 decimal.

Con esto pasamos a la representación:

Campo Bit de Signo S = 1 por ser número negativo

Campo exponente (Característica) C = p + 127, es decir 5 + 127 = 132 que se representará con los 8 bits de dicho campo. (10000101)

Campo mantisa (es lo que está en la parte fraccionaria de la expresión normalizada) es 1010101

Con esto podemos ver como queda la representación de la variable en los 32 bits que utiliza este format.

S	±E + 127	M

1 10000100 10101010000000000000000

A = 0,125 0 01111100 00000000000000000000000

A = - 205,2 1 10000110 10011010011001100110011

Actividad

Como queda representada en memoria la variable A definida como Real en Simple precisión:

A = - 53,25

- ✓ Primero debemos pasar el decimal a Binario natural. TENER EN CUENTA QUE PUNTO FLOTANTE NO TRABAJA CON COMPLEMENTO.
- ✓ Da como resultado en binario A = - 110101,01
- ✓ Luego normalizamos: - 1,1010101 x 10101 donde el exponente p está también en binario que es el 5 decimal.
- ✓ Con esto pasamos a la representación:
- ✓ Campo Bit de Signo S = 1 por ser número negativo
- ✓ Campo exponente (Característica) C = p + 127, es decir 5 + 127 = 132 que se representará con los 8 bits de dicho campo. (10000101)
- ✓ Campo mantisa (es lo que está en la parte fraccionaria de la expresión normalizada) es 1010101
- ✓ Con esto podemos ver como queda la representación de la variable en los 32 bits que utiliza este formato.

S	±E + 127	M
1	10000100	10101010000000000000000

Actividad

Una vez en formato se almacena en Big Endian o en Little Endian:

11000010 01010101 00000000 00000000

Representación de la Dirección o posición de memoria	Valor almacenado (hexadecimal)	Valor almacenado (binario)
xxx0	A2	11000010
xxx1	55	01010101
xxx2	00	00000000
xxx3	00	00000000

Big Endian

Representación de la Dirección o posición de memoria	Valor almacenado (hexadecimal)	Valor almacenado (binario)
xxx0	00	00000000
xxx1	00	00000000
xxx2	55	01010101
xxx3	A2	11000010

Little Endian

B = 0,125 0 01111100 00000000000000000000000 3E000000

C = - 205,2 1 10000110 10011010011001100110011 C34D3333

Overflow y Underflow – Coma flotante

Valores que representan el cero y el infinito:

- ✓ **La representación del Número cero en este convenio es con los 32 bits iguales a cero**
- ✓ **El infinito se representa con todos 1 en el campo exponente (característica) y todos 0 en la mantisa.**

Como consecuencia:

Característica mínima 00000000 por lo tanto exponente mínimo -127

Característica máxima 11111111 por lo tanto el exponente máximo 128

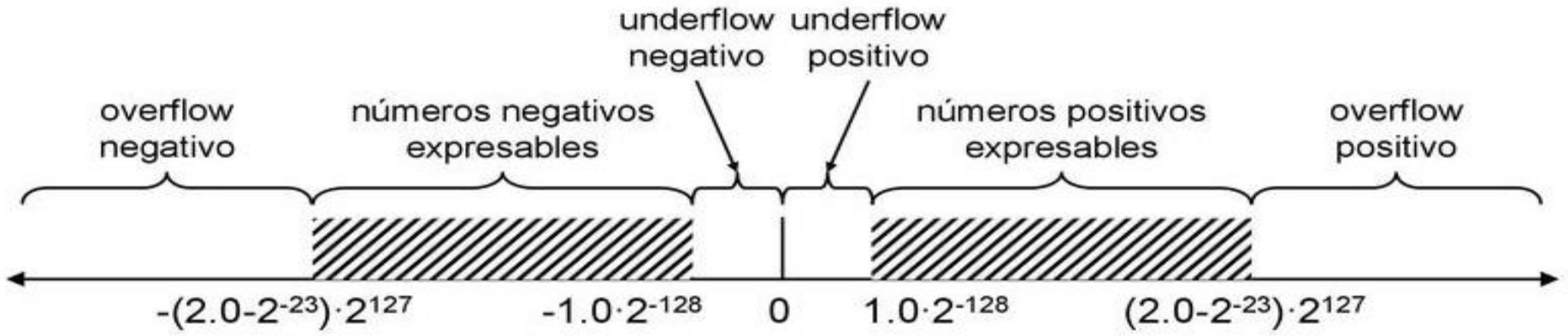
Es decir que los valores extremos de las características se los utiliza para representar el cero y el infinito, es decir que el resto de los valores puede variar entre 00000001 y 11111110 y esto da como resultado no poder representar números muy cercanos a cero.

Overflow y Underflow – Coma flotante

Limites de representación de las variables reales:

Para simple precisión serán: Mínimo aproximadamente -10^{38} y Máximo aproximadamente 10^{38}

Pero a raíz de los valores reservados para la Característica mencionados anteriormente resulta que números muy cercanos al cero no podrán representarse y los mismos son:
Entre -10^{-38} y 10^{-38} aproximadamente excluyendo el cero que si se puede representar.



Convenios IEEE 754

- ✓Punto flotante a su vez puede trabajar con lo que se llama Doble Precisión y también con Precisión Extendida.
- ✓Para estos casos varía tanto el campo característica como el campo mantisa. En el primer caso se logra ampliar los límites de representación (poder representar números aún mas grandes y mas chicos), y en el segundo se aumenta la precisión con la cual vamos la variable se define (cuantos decimales de precisión va a tener)

	Cantidad de Bits				Exceso a sumar al exponente
	Signo	Característica	Mantisa	Total de bits	
SIMPLE PRECISION	1	8	23	32	127
DOBLE PRECISION	1	11	52	34	1023
PRECISION EXTENDIDA	1	15	64	80	16383

Representaciones redundantes

- El cambio de un bit en el almacenamiento o la manipulación de datos origina resultados erróneos.
- Para detectar e incluso corregir estas alteraciones se usan códigos que agregan **“bits redundantes”.**
- Por ejemplo, los bits de paridad se agregan al dato en el momento de su envío y son corroborados en el momento de su recepción por algún componente de la computadora, y lo mismo sucede al revés.
- El resultado de “medir” la cantidad de “ruido” que puede afectar la información se representa con un coeficiente conocido como **tasa de error**.
- De acuerdo con el valor de la tasa de error, se selecciona un código, entre los distintos desarrollados, para descubrir e incluso corregir los errores.
- Estos códigos reciben el nombre de **códigos de paridad.**

Representaciones redundantes

- **Paridad vertical simple o a nivel carácter**

Al final de cada carácter se incluye un bit, de manera que la suma de "unos" del carácter completo sea par (paridad par) o impar (paridad impar). Este tipo de codificación se denomina paridad vertical. Suele acompañar la transmisión de octetos en los periféricos y la memoria.

Los ejemplos muestran el bit de paridad que se agrega al octeto, en el caso de usar paridad par o impar:

00110010	*1*	*se agrega un uno que permita obtener paridad par*
01001011	*0*	*se agrega un cero para lograr paridad par*

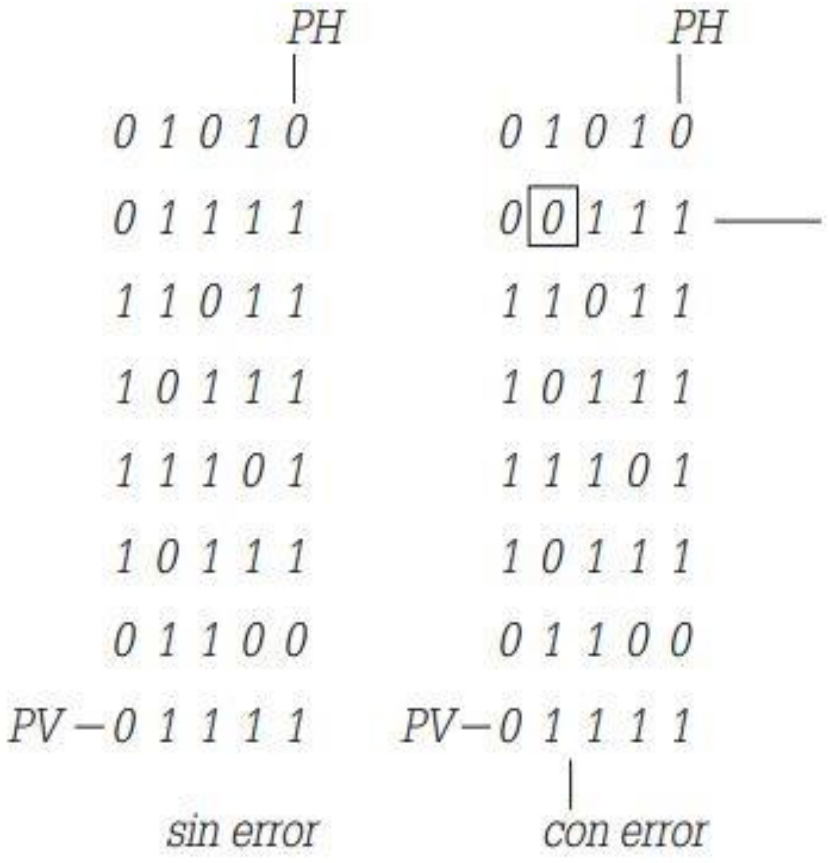
Este código de detección de errores sólo se utiliza si la tasa de error en la cadena de bits que se han de transmitir no es superior a uno; esto es, si hay dos bits erróneos no permite detectarlos.

Representaciones redundantes

- **Paridad entrelazada**

Utilizando en conjunto el código de paridad vertical con el horizontal se construye el código de paridad entrelazada, que además de detectar errores permite corregirlos.

Ejemplo: Supongamos una transmisión de cuatro caracteres en código ASCII de 7 bits, con paridad par. El primer grupo indica cómo deberían llegar los caracteres a su lugar de destino, de no haberse producido errores en la transmisión.
El segundo grupo indica con un recuadro el bit que tuvo un error en la transmisión.



Representaciones redundantes

- **Paridad entrelazada**

La forma de detectar el error es verificar tanto los bits del mensaje como los de paridad, generando un nuevo bit de paridad (en este caso llamado Verificador) Donde hay 1 es que hay error.

La forma de corregir el error es invertir el bit señalado como erróneo.

Si hay más de un error en la misma fila o columna, quizá se pueda detectar, pero no se podrá determinar con exactitud cuál es el error.