Instituto Universitario Politécnico

"Santiago Mariño"

Unidad Nº III – Unidad Aritmética-Lógica

Elaborado por: Ing. Víctor Valencia.

Unidad Aritmética-Lógica

Es la parte del computador que realiza realmente las operaciones aritméticas y lógicas con los datos. El resto de los elementos del computador están principalmente para suministrar datos a la ALU. A fin de que ésta los procese, y para recuperar los resultados.

Los datos se presentan a la ALU en registros, y en registros se almacenan los resultados de las operaciones. La ALU también activará indicadores (flags) como resultados de operaciones. Por ejemplo, un indicador de desbordamiento se pondrá a 1 si el resultado de una operación excede la longitud del registro en donde este debe almacenarse. La UC proporciona las señales que gobiernan el funcionamiento de la ALU y la transferencia de datos dentro y fuera de la ALU.

Unidad Aritmética-Lógica

Interconexión general de la ALU con el resto del CPU

Aritmerica del computador

La aritmética de un computador se realiza normalmente con dos tipos de números muy diferentes: *enteros y en coma flotante*. En ambos casos, la representación elegida es un aspecto de diseño crucial que se toma en cuanta en todo sistema computacional; otro aspecto es cómo se realizan las operaciones aritméticas.

Representación de enteros

En el sistema binario, cualquier numero puede representarse tan solo con los dígitos 1 y 0, el signo menos, y la coma de la base (que separa la parte entera de la decimal, el punto en otros países). Ejemplo:

$$-1101{,}0101_2 = -13{,}3125_{10}$$

El computador no puede distinguir tanto los signos como la coma. Para representar números sólo puede utilizarse dígitos 0 y 1. Si utilizamos números sólo enteros no negativos, su representación sería inmediata.

Representación de enteros

Una palabra de ocho bits puede representar números desde 0 hasta 255, entre los que se encuentran:

00000000 = 0

00000001 = 1

00101001 = 41

10000000 = 128

11111111 = 255

En general, si una secuencia de n dígitos binarios a_{n-1} a_{n-2} … a_1 a_0 es interpretada como un entero sin signo A, su valor es:

$$A = \sum_{i=0}^{n-1} 2^i a_i$$

Representación en signo y magnitud

Dado a que usar enteros sin signo es insuficiente en muchos casos en los que se necesita representar tanto positivos como negativos. Existen convenciones alternativas para solucionar dicho problema. Todas implican tratar el bit más significativo (el más a la izquierda) de la palabra como un bit de signo:

Si dicho bit es 0 el número es positivo, y si el es 1, el número es negativo.

La forma más sencilla de representación que emplea un bit de signo es la denominada *representación signo-magnitud.* En una palabra de n bits, los n-1 bits de la derecha representan la magnitud del entero por ejemplo:

+18 = 00010010

Representación en signo y magnitud

El caso general puede expresarse así:

$$A = \begin{cases} \sum_{i=0}^{n-2} 2^i a_i & \text{Si } a_{n-1} = 0 \\ -\sum_{i=0}^{n-2} 2^i a_i & \text{Si } a_{n-1} = 1 \end{cases}$$

Esta representación presenta varias limitaciones. Una de ellas es que la suma y la resta requieren tener en cuenta tanto los signos de los números como sus magnitudes relativas para llevar a cabo la operación en cuestión. Otra limitación es que hay dos representaciones del numero 0:

+ 0_{10} = 00000000

- 0_{10} = 10000000 (signo-magnitud)

Representación en complemento a dos

Al igual que la representación signo-magnitud, la representación en complemento a dos usa el bit más significativo como bit de signo, facilitando la comprobación de si el entero es positivo o negativo. Difiere en la forma de interpretar los bits restantes.

Si A es positivo, el bit de signo a_{n-1}, es cero. Los restantes bits representan la magnitud del número de la misma forma que en la representación signo-magnitud; es decir:

$$A = \sum_{i=0}^{n-2} 2^i a_i \quad \text{Para A > 0}$$

Representación en complemento a dos

Para un número negativo A, el bit de signo a_{n-1}, es 1. Los n-1 bits restantes pueden tomar cualquiera de los 2^{n-1} combinaciones. Por tanto, el rango de los enteros negativos que pueden representarse es de -1 hasta -2^{n-1}. Ya que el bit de signo es 1, podemos escribir la expresión de los números negativos como:

$$A = -2^{n-1} a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i$$

Representación en complemento a dos

Caja de valores para convertir entre binario en complemento a dos y decimal:

–128	64	32	16	8	4	2	1

(a) Caja de valores de 8 bits en complemento a dos

–128	64	32	16	8	4	2	1
1	0	0	0	0	0	1	1

–128 +2 +1 = –125

(b) Conversión a decimal del número binario 10000011

–128	64	32	16	8	4	2	1
1	0	0	0	1	0	0	0

–120 = –128 +8

Aritmética con enteros

Negación

En la representación signo-magnitud, la negación se obtiene invirtiendo el bit de signo. En la notación de complemento a dos se obtiene en dos pasos:

1. **Cambiar cada 1 por 0, y cada 0 por 1**
2. **Sumarle 1.**

+18 = 00010010 (complemento a dos)
Complemento bit a bit = 11101101
\+ 1

11101110 = -18

Aritmética con enteros

El opuesto del opuesto es el propio número:

```
                    -18 =   11101110 (complemento a dos)
Complemento bit a bit = 00010001
                          +        1
                         ----------
                         00010010 = +18
```

Si existe acarreo en la posición del bit más significativo, es ignorado.

En complemento a dos hay una representación de n bits para el -2^{n-1}, pero no para el 2^{n-1}.

Aritmética con enteros

Suma

La suma efectúa igual que si lo números fuesen enteros sin signo.

1001 = -7 +0101 = 5 --- 1110 = -2	1100 = -4 +0100 = 4 --- 10000 = 0
0011 = 3 +0100 = 4 --- 0111 = 7	1100 = -4 +1111 = -1 --- 11011 = -5
0101 = 5 +0100 = 4 --- 1001 = Desbordamiento	1001 = -7 +1010 = -6 --- 10011 = Desbordamiento

Aritmética con enteros

Suma y resta

En cualquier suma, cuando el resultado es mayor que el permitido por la longitud de la palabra se denomina desbordamiento (overflow). La ALU debe indicarlo para no se intente utilizar el resultado obtenido. Para detectar el desbordamiento se debe observar la siguiente regla:

***Regla de desbordamiento**: al sumar dos números, y ambos son o bien positivos o negativos, se produce desbordamiento si y solo sí el resultado tiene signo opuesto.*

La resta se trata también fácilmente con la siguiente regla:

***Regla de la resta:** para sustraer un número (el sustraendo) de otro (minuendo), se obtiene el complemento a dos del sustraendo (Negación) y se le suma al minuendo.*

Aritmética con enteros

Resta

M = 2 = 0010 S = 7 = 0111 -S = -7 = 1001 0010 = 2 +1001 = -7 --------- 1011 = -5	M = 5 = 0101 S = 2 = 0010 -S = -2 = 1110 0101 = 5 +1110 = -2 --------- 10011 = 3
M = 7 = 0111 S = -7 = 1001 -S = 7 = 0111 0111 = 7 + 0111 = 7 --------- 1110 = Desbordamiento	

Aritmética con enteros

Multiplicación: Enteros sin signo

1. Generación de productos parciales, uno para cada dígito del multiplicador. Estos productos parciales se suman después para para producir el producto final.
2. Cuando el bit del multiplicador es cero, el producto parcial es cero. Cuando el multiplicador es uno, el producto parcial es el multiplicando.
3. El producto final se obtiene sumando todos los productos parciales. Cada producto parcial se desplaza en una posición hacia la izquierda con respecto al precedente.

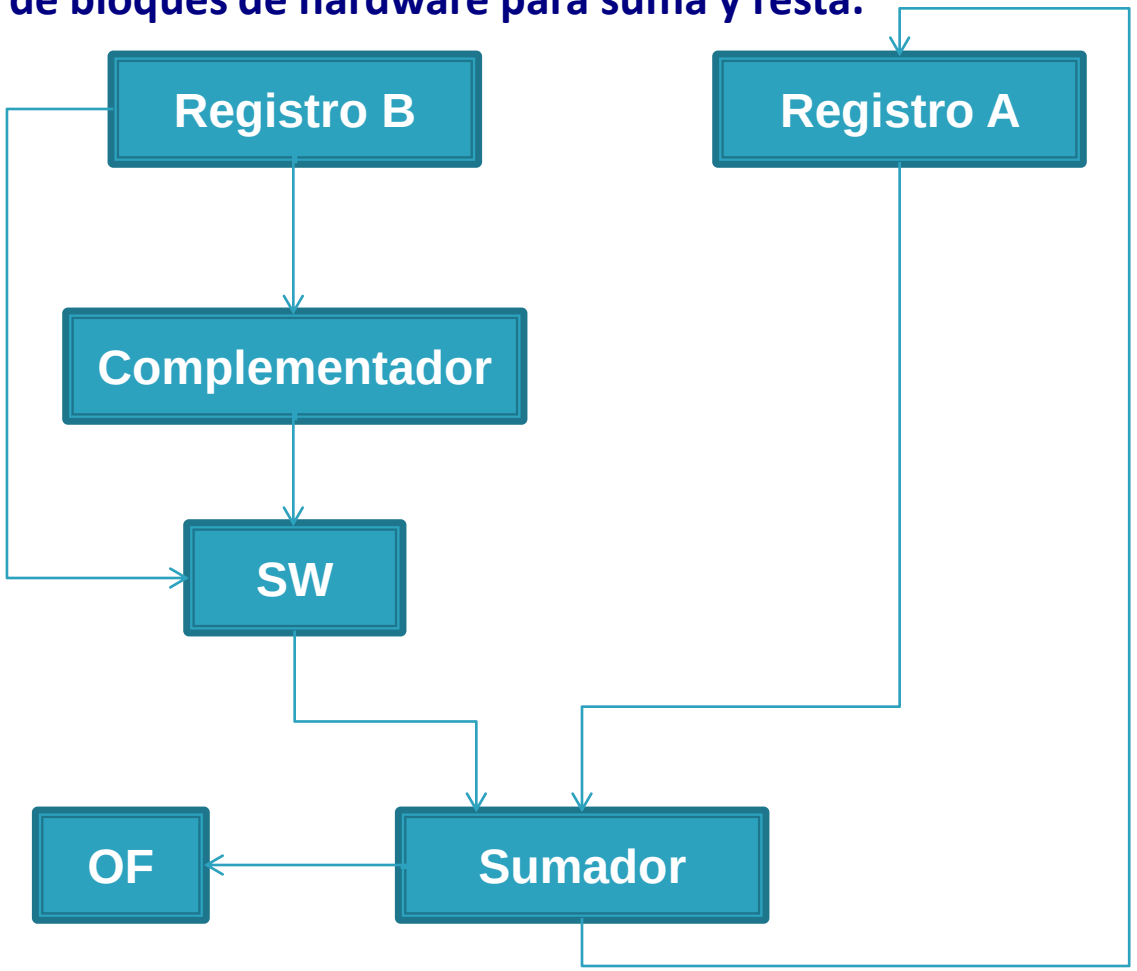
Aritmética con enteros

Multiplicación: Enteros sin signo

				1	0	1	1	Multiplicando (11)
			x	1	1	0	1	Multiplicador (13)
				1	0	1	1	Productos Parciales
			0	0	0	0		
		1	0	1	1			
	1	0	1	1				
1	0	0	0	1	1	1	1	Producto (143)

Aritmética con enteros

Suma y resta: **Diagrama de bloques de hardware para suma y resta.**



SW= conmutador (selecciona suma o resta)
OF= Bit de desbordamiento