UNIVERSIDAD NACIONAL AUTONOMA
DE MEXICO

FACULTAD DE INGENIERIA

DISEÑO DE LA UNIDAD ARITMETICA PARA EL SIMMP-1

T E S I S

QUE PARA OBTENER EL TITULO DE

INGENIERO MECANICO ELECTRICISTA

P R E S E N T A :

JESUS GERARDO YLIZALITURRI RODRIGUEZ

México, D. F. 1992

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

INDICE

INTRODUCCIÓN.

En la presente tesis se describe el diseño de una unidad aritmética para el SIMMP-1. SIMMP-1 significa: sistema interconectado de microcomputadora y microprocesador paralelo 1. El SIMMP-1 es un diseño digital ideado y construido por un grupo de profesores de la facultad de ingeniería, y dirigidos por el director de ésta tesis. Este sistema propone la alternativa de interconectar o comunicar, a un sistema basado en el microprocesador Z-80, con una microcomputadora PC (compatible con IBM). La arquitectura, el mapa de memoria y el funcionamiento del sistema monotablilla, así como el editor del SIMMP-1, se describen con detalle en el apéndice.

La idea de interconectar un microprocesador con una PC, es interesante de sobremanera, pues de ésta forma se pueden probar y depurar programas para el Z-80, directamente en el sistema Z-80, sin la necesidad de grabar estos en memorias EPROM y así evitar

que, en caso de que algún programa en particular no funcione como se esperaba, se tenga que borrar (quemar) la memoria y volverla a grabar para una vez más probar el programa dentro del sistema.

Para conseguir sus objetivos y evitar los problemas anteriores, el SIMMP-1 cuenta con un editor manejable desde la PC, en el cual se pueden crear, verificar y corregir programas, bajarlos mediante una interfase serial a la memoria RAM del sistema real Z80, ordenar la ejecución del mismo y, por último observar directamente en pantalla el mapa de memoria. De ésta manera, se pueden verificar resultados para la correcta evaluación del programa y, permitirle al diseñador grabar su programa en la memoria EPROM con la certeza de que éste ya funciona. Así, el SIMMP-1, junto con su editor conforman un sistema digital completo para la correcta evaluación de software de algún sistema Z80 .

La unidad aritmética que propone esta tesis, no sólo dotará al SIMMP-1 de una herramienta más para enriquecer su aprovechamiento, sino que contribuirá al diseño de posteriores unidades aritméticas para otros sistemas que se basen en otros microprocesadores más poderosos o más integrados. A través de la experiencia nos hemos percatado que, para la correcta aplicación de algoritmos en el procesamiento digital de señales (v.g.), es indispensable contar con una unidad confiable que pueda realizar operaciones aritméticas con cierto grado de precisión. Simplemente hay que pensar que, al utilizar una computadora con la cual podamos definir algún

algoritmo digital y simular resultados, no nos preocupa el cómo logra la computadora realizar por ejemplo, una multiplicación o una función trigonométrica (especificando si es entera o real nuestra variable); pero a la hora que queremos implementar nuestro algoritmo sin la ayuda de la computadora y con *x* o *y* microprocesador, es entonces cuando empiezan los problemas, pues en lo primero en que se debe pensar (aunque generalmente es lo primero que se omite) es en cómo se van a realizar las operaciones aritméticas. Existe la posibilidad, por ejemplo, de utilizar un coprocesador aritmético en nuestro sistema; con tal elección, habremos resuelto el problema de una manera fácil; y no sólo fácil, sino que el sistema tendrá la capacidad de obtener resultados de operaciones aritméticas, con una gran precisión y a una velocidad apreciable de procesamiento. Tal elección se justifica, si los requerimientos de precisión y velocidad antes mencionados son indispensables, y sobre todo el último; pero si no, estaremos pagando un precio muy caro, pues un coprocesador aritmético, de la familia que se trate, cuesta tanto o más que el sistema primario completo, mientras tal coprocesador no se justifique, queda la opción de diseñar una unidad aritmética mediante algoritmos, e instalarla en la memoria para invocar a esta cuando se requiera.

Ahora, para continuar, contestemos la siguiente pregunta: qué es ésta unidad aritmética? ó, cómo es ésta unidad aritmética?. Simple. La unidad aritmética que propone esta tesis, es una serie

de programas, basados en algoritmos específicos, que logran obtener resultados válidos a operaciones aritméticas elementales (suma,resta,multiplicación y división) deseadas. La característica fundamental de esta unidad aritmética, es la de poder realizar operaciones en punto flotante, logrando así resultados de buena precisión. En lo que respecta a la velocidad de procesamiento, apuntemos que ésta dependerá de la velocidad del mismo microprocesador y, de la longitud de los formatos que se utilicen. Algo importante que hay que mencionar es que por muy veloz que sea un microprocesador, nunca logrará realizar operaciones aritméticas a la velocidad de un coprocesador aritmético, el cual fue hecho precisamente para eso. Sin embargo, las hay cientos de aplicaciones en que la velocidad para realizar operaciones aritméticas no sea fundamental y, por lo tanto una unidad como la que proponemos, bien puede solventar tales aplicaciones. Hay que señalar que, aunque esta unidad aritmética es un diseño para un sistema en particular: el SIMMP-1, su forma y capacidades pueden aprovecharse para cualquier sistema en cuya arquitectura opere un microprocesador Z-80. Además, la tesis propone bases prácticas para la fácil elaboración de alguna otra unidad aritmética para otro microprocesador. Es interesante señalar, que al saber cómo se hace una de éstas unidades aritméticas, algunos proyectos que requerían de una, investigando aquí, pueden ser implementados con más facilidad.

Por lo demás; los capítulos que van formando la tesis, están dispuestos de tal manera que al ir avanzando en ellos se establezcan las nociones para la exacta comprensión del diseño final.

El capítulo I del sistema numérico binario, explica los conocimientos básicos para la correcta generación de números que pueden ser procesados por los sistemas digitales, así como las representaciones y formatos utilizados en los mismos.

En el capítulo II de aritmética binaria, se presentan los elementos básicos para definir la suma binaria, que es la base de la aritmética binaria, a partir de elementos de lógica digital. Se presentan además las distintas formas de aritmética en sus diferentes representaciones numéricas binarias.

El capítulo III de aritmética de punto flotante, explica el formato y las definiciones de los elementos del mismo, así también ilustra los algoritmos para resolver las operaciones aritméticas en éste formato.

El capítulo IV dedicado a la unidad aritmética de punto flotante del SIMMP1, simplemente especifica los pasos seguidos tanto para la organización como para el manejo de los registros del microprocesador, así como el empleo del estándar de la IEEE para el formato de punto flotante de 32-bits. También se incluyen los listados completos correctos y explicados de los programas en nemónicos y emsamblador de la unidad aritmética.

El capítulo V sobre las aplicaciones, se expone un ejemplo para mostrar la utilidad de la unidad aritmética, ya como bloque instalado en el SIMMP-1. El ejemplo que se presenta aquí, es la obtención de la función trigonométrica sen(x) a partir de la serie de McLaurin; se explica la estructura de la serie, la conformación que nos conviene de la misma y, el modo en que se adaptó con el uso de las rutinas aritméticas de punto flotante. Se presenta el programa emsamblador de la función y se proponen aplicaciones similares.

En el último capítulo VI se presentan las conclusiones, que no son sino los alcances de ésta unidad aritmética como herramienta de cálculo dentro de un sistema digital, y en particular dentro del SIMMP1; su precisión y sus tiempos aproximados de cálculo, se mencionan conclusiones de las aplicaciones. También los pormenores que se presentaron en la realización de la tesis.

En el apéndice A, aparece información sustancial del SIMMP1, como su arquitectura, su mapa de memoria y una explicación de su editor de emsamblador cruzado.

En el apéndice B, presentamos dos programas en lenguaje de alto nivel, que apoyaron el desarrollo de la unidad aritmética.

En el apéndice C, realizamos unos programas en emsamblador, útiles para la comunicación de números en punto flotante y valores del mundo real. Con la bibliografía damos por terminado el objetivo de ésta tesis.

CAPITULO I

EL SISTEMA NUMÉRICO BINARIO.

En este capítulo discutiremos el sistema numérico binario y en particular, describiremos las diversas maneras en que los números pueden ser expresados en representación binaria, esto es, en un sistema numérico que utiliza sólo dos dígitos: 0 y 1. Esta representación binaria es utilizada en las computadoras y en los sistemas digitales en general, porque los elementos básicos con los cuales dichos sistemas son construidos, son de una naturaleza binaria. La comprensión de la representación numérica binaria, es un requisito para los subsecuentes capítulos.

1.1. NOTACIÓN POSICIONAL.

La representación de números en la notación posicional, ha recibido una aceptación universal y debemos de justificar su importancia. Cuando damos una secuencia de dígitos decimales, v.g. 40795, podemos encontrar que cada dígito es multiplicado por un factor o un *peso* que es una potencia de 10 y que depende de la posición del dígito en el número, por ejemplo $40795 = 4x10^4+0x10^3+7x10^2+9x10^1+5x10^0$. Así cada número decimal de n dígitos es una suma de los coeficientes de *peso*.

$$N_{10} = a_{n-1}x10^{n-1}+a_{n-2}x10^{n-2}+\ldots+a_1x10^1+a_0x10^0 = \sum_{i=0}^{n-1} a_i x10^i \qquad (1.1)$$

donde a_i es el coeficiente del peso 10^i.

Ahora, la notación posicional es una representación corta de la ecuación 1.1, en donde el signo y los *pesos* son omitidos, así:

$$N_{10} = a_{n-1}\ a_{n-2}\ \ldots\ a_1\ a_0 \qquad (1.2)$$

En la ecuación 1.2 notamos la forma general de la representación posicional. En el caso anterior la base o raíz r fue 10, pero podemos expresar cualquier otra raíz r, la ecuación 1.2 puede ser escrita en su forma general

$$N_r = a_{n-1}r^{n-1}+a_{n-2}r^{n-2}+\ldots+a_1r^1+a_0r^0 = \sum a_i r^i \qquad (1.3)$$

teniendo un rango de enteros de 0 a $(r-1)$, es decir, que en un sistema numérico de base o raíz r, se tendrán r diferentes dígitos para representar los números.

Lo anterior, lo podemos extender para expresar números fracciones. En el sistema decimal representamos un número fracción por los dígitos que están puestos a la derecha del *punto raíz* ó, en este caso, punto decimal. Por ejemplo, el número 0.235 en base 10 tiene un valor $2x10^{-1}+3x10^{-2}+5x10^{-3}$. En general, el valor de cualquier número fracción n_r de base r teniendo m dígitos después del punto raíz es:

$$n_r = \ . \ a_{-1}r^{-1}+a_{-2}r^{-2}+\ldots+a_{-m+1}r^{-m+1}+a_{-m}r^{-m} = \sum_{i=-m}^{-1} a_i r^i \qquad (1.4)$$

Combinando las ecuaciones 1.3 y 1.4 obtenemos una expresión general para cualquier entero, fracción ó, mezcla de ambos de *(n+m)* dígitos en base r.

$$N = N_r+n_r = [a_{n-1}r^{n-1}+\ldots+a_0r^0 \ . \ a_{-1}r^{-1}+\ldots+a_{-m}r^{-m}]_r$$

$$= \sum_{i=-m}^{n-1} a_i r^i \qquad (1.5)$$

1.2. EL NÚMERO BINARIO.

El sistema numérico binario es el más convencional y sencillo de implementar para el uso interno de las computadoras digitales, y como tal es un sistema numérico posicional. En ésta forma de representación, un número es codificado como un vector de n dígitos binarios o bits (BInary digiT), en el cual cada bit tiene un peso

de acuerdo a su posición en el vector. Asociado al sistema numérico, tenemos la base ó raíz *r*. Cada dígito tiene un valor entero en el rango de *0* a *r-1*, por lo tanto para el sistema numérico binario donde *r=2*, cada dígito, y ahora llamado bit, tiene el valor numérico de *0* o de *1*.

Considérese un vector de n-bits de la forma:

$$A = a_{n-1} \quad a_{n-2} \quad \ldots \quad a_1 \quad a_0 \tag{1.6}$$

donde $a_i = 0$ o 1 para $0 \leq i \leq n-1$. Este vector puede representar valores enteros positivos en el rango de *0 a* 2^n-1, donde :

$$A = a_{n-1}x2^{n-1} + \ldots + a_1x2^1 + a_0x2^0 \tag{1.7a}$$

Así, el valor de A, puede ser escrito como:

$$A = \sum_{i=0}^{n-1} a_i 2^i \tag{1.7b}$$

Ambas representaciones son para los enteros positivos. Esto puede ser extendido como ya lo apuntamos, para incluir fracciones, por ejemplo:

$$A = a_{n-1}x2^{n-1}+\ldots+a_1x2^1+a_0x2^0 . \; a_{-1}x2^{-1}+a_{-2}x2^{-2}+\ldots+a_{-m}x2^{-m} \tag{1.8}$$

Ligando los dígitos binarios, podemos ver como ejemplo que el número binario 1101.110 con su punto binario, puede ser escrito según la ecuación 1.5 y representar la cantidad

$1x2^3+1x2^2+0x2^1+1x2^0+1x2^{-1}+1x2^{-2}+0x2^{-3} = 13.7$

1.3. CONVERSIÓN DE BASE.

El uso común de números binarios y decimales en los sistemas digitales requiere procedimientos para convertir un número dado en

uno equivalente de otra base. Ahora discutiremos un método general de conversión. Consideremos el número decimal N_{10} el cual tiene una parte entera de n dígitos y una parte fracción de m dígitos

$$N_{10} = d_{n-1}\ d_{n-2}\ \ldots\ d_1\ d_0.d_{-1}\ d_{-2}\ \ldots\ d_{-m} \qquad (1.9)$$

donde d_i representa el i-ésimo dígito decimal. La cantidad numérica representada por la ecuación 1.8 también puede ser expresada en cualquier otra base; sin embargo, el número de dígitos difiere generalmente. Por ejemplo, en base 2 el número N_2 tendrá s enteros y t dígitos fracción; en la notación posicional, esto puede ser expresado como

$$N_2 = b_{s-1}\ b_{s-2}\ \ldots\ b_1\ b_0.b_{-1}\ b_{-2}\ \ldots\ b_{-t} \qquad (1.10)$$

La conversión entre base debe tratarse separadamente, en entera y fracción.

1.3.1. Conversión de base de un número entero. Igualando las partes enteras del número en base 10 y en base 2, tenemos

$$N_{10}(\text{entero}) = b_{s-1}2^{s-1}+b_{s-2}2^{s-2}+\ldots+b_1 2^1 + b_0 \qquad (1.11a)$$

Factorizando el 2 de la ecuación, tenemos

$$N_{10}(\text{entero}) = 2[b_{s-1}2^{s-2}+b_{s-2}2^{s-3}+ \ \ldots\ + b_1] + b_0 \qquad (1.11b)$$

o también $\qquad N_{10}(\text{entero}) = 2A_1 + b_0$

donde A_1 es un polinomio en los paréntesis cuadrados de la ecuación 1.11b y b_0 es el *residuo*, el siguiente bit b_1, es obtenido factorizando 2 de A_1

$$A_1 = 2A_2 + b_1$$

factorizando sucesivamente a 2

$$A_2 = 2A_3 + b_2$$

$$A_3 = 2A_4 + b_3$$

$$\ldots$$

$$A_i = 2A_{i+1} + b_i$$

Donde A_i de un polinomio de un grado mayor que A_{i+1} y b_i es el residuo binario. En el s-avo paso se completa el proceso, produciendo el residuo b_{s-1}. hay que notar que los residuos de b_0 hasta b_{s-1} son los dígitos binarios de la parte entera de N_2 en la ecuación 1.10.

EJEMPLO. Convertir a binario el entero decimal 53.

$$53 \div 2 = 26, \text{ residuo } 1 = b_0 \text{ (LSB)}$$

$$26 \div 2 = 13, \text{ residuo } 0 = b_1$$

$$13 \div 2 = 6, \text{ residuo } 1 = b_2$$

$$6 \div 2 = 3, \text{ residuo } 0 = b_3$$

$$3 \div 2 = 1, \text{ residuo } 1 = b_4$$

$$1 \div 2 = 0, \text{ residuo } 1 = b_5 \text{ (MSB)}$$

El resultado binario se obtiene leyendo de abajo a arriba así $53_{10} = 110101_2$.

1.3.2. Conversión de base de un número fracción. Para convertir la parte fracción de un número decimal a un número binario en forma de fracción, tenemos

$$N_{10} \text{ (fracción)} = b_{-1}2^{-1} + b_{-2}2^{-2} + \ldots + b_{-L}2^{-L} \qquad (1.12a)$$

Multiplicando por 2 la ecuación 1.12a produce

$$2N_{10}\text{(fracción)} = b_{-1} + \{b_{-2}2^{-1} + b_{-3}2^{-2} + \ldots + b_{-L}2^{-L+1}\} \qquad (1.12b)$$

Donde b_{-1} es 0 o 1, y la expresión en los paréntesis cuadrados es

menor que 1. t multiplicaciones sucesivas por 2 produce b_{-1}, b_{-2},..b_{-t}, en ese orden. El proceso es completado, cuando la parte fracción es 0 después de una multiplicación por 2 ó, cuando la exactitud deseada es obtenida.

EJEMPLO. Convertir a binario la fracción decimal $.39_{10}$.

$.39 \times 2 = 0.78 \quad b_{-1} = 0$

$.78 \times 2 = 1.56 \quad b_{-2} = 1$

$.56 \times 2 = 1.12 \quad b_{-3} = 1$

$.12 \times 2 = 0.24 \quad b_{-4} = 0$

$.24 \times 2 = 0.48 \quad b_{-5} = 0$

$.48 \times 2 = 0.96 \quad b_{-6} = 0$

$.96 \times 2 = 1.92 \quad b_{-7} = 1$

$.92 \times 2 = 1.84 \quad b_{-8} = 1$

$.84 \times 2 = 1.68 \quad b_{-9} = 1$

$.68 \times 2 = 1.36 \quad b_{-10} = 1$

Detenemos el proceso aquí; aunque incurrimos en un *error de redondeo*, este, tiene un valor menor de 2^{-10}, esto es, $e < 1/1024$. El equivalente binario de la fracción decimal se obtiene leyendo los b_i de arriba a abajo: $.39_{10} = .0110001111_2 + e$.

1.4. REPRESENTACIÓN BINARIA, OCTAL Y HEXADECIMAL.

Otras raíces usadas por las computadoras digitales son: la octal base *8*, la decimal base *10*, y la hexadecimal base *16*. Los ocho símbolos de la base octal son de *0 al 7*, para la decimal del *0 al 9*, la hexadecimal del *0 al 9* y luego de la *A a la F*, los

últimos símbolos de la base hexadecimal han sido adoptados por convención y corresponden a los valores del *10 al 15* respectivamente.

A partir de que 2^3 igual a *8* y, 2^4 es 16, cada dígito octal corresponde a *3 dígitos binarios* y cada dígito hexadecimal corresponde a *4 dígitos binarios*. Las conversiones de binario a octal y a hexadecimal son fácilmente hechas, haciendo particiones del número binario, asignando grupos de tres *(r=8)* ó de cuatro *(r=16)* respectivamente. Formándolos empezando del punto binario

EJEMPLO. Convertir de binario a hexadecimal y a octal el número binario $11001.11101 = 25.90625_{10}$.

Para convertir a octal, formamos grupos de tres y si faltan dígitos completamos con ceros. Para convertir a hexadecimal formamos grupos de cuatro y si faltan dígitos completamos con ceros. Así:

$$\underbrace{011}_{3}\ \underbrace{001}_{1}\ .\ \underbrace{111}_{7}\ \underbrace{010}_{2}{}_{2} = 31.72_8$$

$$\underbrace{0001}_{1}\ \underbrace{1001}_{9}\ .\ \underbrace{1110}_{E}\ \underbrace{1000}_{8}{}_{2} = 19.E8_{16}$$

1.5. REPRESENTACIONES NUMÉRICAS.

Todas las operaciones matemáticas pueden ser expresadas en términos de las cuatro operaciones aritméticas básicas ó elementales: suma, resta, multiplicación y división. Estas operaciones pueden ser ejecutadas en tres modos de operación, correspondientes a los tres tipos de representación numérica más usadas : *representación en punto fijo, representación en BCD y, representación en punto flotante*.

1.5.1. Representación en punto fijo. La aritmética de punto fijo, es usada principalmente en problemas donde los datos son representados con un punto ó raíz fijo, esto es, que el punto binario reside en una posición fija dentro del número. Las operaciones de punto fijo pueden ser subdivididas en dos categorías: en aritmética entera, en donde el punto binario está colocado a la extrema derecha y, en donde el punto binario está a la extrema izquierda. Una gran mayoría de los diseños en punto fijo usa la aritmética entera.

Formato entero [X---XX.], formato fracción [.X---XX] X={0 o 1}.

EJEMPLO Representar en punto fijo en sus dos formatos el número 1110100001_2 y su equivalente en decimal.

Formato entero $1110100001. = 929._{10}$

Formato fracción $.1110100001 = 0.9072265625_{10}$

1.5.2. Representación decimal. La representación decimal es usada en operaciones aritméticas, operaciones de edición ó, visualización de datos o resultados. algunas calculadoras de

bolsillo para su operación, que sólo es desplegar en un display datos ó, un único resultado, para evitar problemas de conversión entre códigos, emplean la representación numérica en BCD. Los datos decimales pueden ser expresados en *paquetes*. Estos *paquetes* de cuatro bits pueden representar los dígitos (0 - 9), el signo del número, la longitud del número y la posición del punto decimal. No hay una regla general para formar números en BCD, y cada compañía o grupo de diseño adopta el formato que le conviene.

1.5.3. Representación en punto flotante. La representación en punto flotante y su aritmética, es usada principalmente en cálculos científicos y de ingeniería, en el cual es frecuente el escalamiento[1] de magnitud. Las operaciones en esta representación, pueden ser subdivididas en: operaciones normalizadas, que son aquellas que requieren que sus mantisas tengan su MSB=1; y las no normalizadas, que son aquellas operaciones que no requieren que el MSB de las mantisas sea 1. La explicación detallada del concepto de punto flotante, se discute en el capítulo tres.

1.6. REPRESENTACIÓN DE NUMEROS ENTEROS.

Hasta ahora hemos establecido al número como entero o fracción, y siempre positivo. Si sólo enteros positivos habrán de ser representados en punto fijo, entonces una palabra de n-bits permitirá un rango de 0 a 2^n-1, si v.g. tenemos 8 bits, el rango sería del 0 a 255.

[1]Escalamiento significa multiplicar a una variable por una constante.

Sin embargo, tanto enteros positivos como negativos son usados en los cálculos, y un esquema de codificación deberá ser propuesto para que tanto positivos como los negativos sean distribuidos correctamente, y deberá además de tener una forma tal que se distinga fácilmente el signo con una sencilla prueba, la detección del cero deberá ser simple y la ejecución de las cuatro operaciones básicas deberá ser fácilmente implementada. Generalmente el *MSB* es usualmente reservado para el signo. Considere el siguiente número *binario A*:

$$A = a_{n-1} \quad a_{n-2} \quad \cdots \quad a_1 \quad a_0$$

donde el dígito signo a_{n-1} tiene el siguiente valor

$$a_{n-1} \begin{cases} 0 & \text{si } |A| \geq 0 \\ 1 & \text{si } |A| < 0 \end{cases}$$

y los dígitos de a_{n-2} a a_0 que restan en A, indican el valor real de A ó, en su caso, su valor complementado. Hay tres maneras convencionales de representar números positivos o negativos en el sistema numérico binario: complemento a uno, complemento de dos y, magnitud con signo.

1.6.1. Complemento a uno. Los enteros positivos en el rango de 0 a $2^{n-1}-1$ son representados como $[0\ a_{n-2}\ a_{n-1}\ \cdots\ a_1\ a_0]$. Los números negativos en el rango de 0 a $2^{n-1}-1$ están representados, obteniendo primero su valor absoluto y luego complementando cada bit a su respectivo valor positivo, esto es, 1 por el 0, y el 0 por el 1. Así un número positivo estará representado por:

$$A = 0 \quad a_{n-2}\ a_{n-3} \ldots a_1\ a_0$$

v.g. $$A = 0 \quad 1\ 1\ 0\ 1\ 1\ 1\ 0\ 0_2$$

y el mismo valor es representado como un número negativo así:

$$A = 1 \quad \bar{a}_{n-2}\ \bar{a}_{n-3} \ldots \bar{a}_1\ \bar{a}_0$$

$$A = 1 \quad 0\ 0\ 1\ 0\ 0\ 0\ 1\ 1_2$$

Comparar aquí los signos de dos números es directo, porque el *MSB* del número positivo es cero y el *MSB* del número negativo es uno. Desafortunadamente, hay una representación dual para el cero, por que una palabra de todos ceros, que sería el (+)0, se vuelve en una palabra de todo unos, que es el (-)0 cuando es complementada. Esto le quita un número a la numeración.

1.6.2. Complemento a dos. los números positivos en el rango de: 0 a 2^{n-1} siguen siendo representados en el sistema binario posicional usual, como: $A = 0\ a_{n-2}\ a_{n-3} \ldots a_1\ a_0$. Los números negativos son obtenidos mediante la suma de un *1* a la representación de complemento a *1* del valor positivo deseado. Así, un número negativo es representado por:

$$\bar{A}+1 = 1\ \bar{a}_{n-2}\ \bar{a}_{n-3} \ldots \bar{a}_1\ \bar{a}_0 + 1$$

Por eso, para formar un número negativo en la notación de complemento a *2*, se toma el número positivo correspondiente, se complementa cada bit y se suma un 1 en la posición *LSB*. Esta suma puede cambiar todos los *n*-bits de el número; por ejemplo, el complemento a *2* de *00000000* se vuelve 11111111 + 1 = 1 0000000, La

prueba de signo es otra vez, una simple comparación de dos bits, y aquí hay sólo una representación del cero; por que la fila de ceros cuando es negada se vuelve otra fila de ceros y, el acarreo es descartado. El rango es representado de: -2^{n-1} *a* $2^{n-1}-1$, por ejemplo, para 8 bits, el rango sería de -128 a +127, donde -2^{n-1} es representado por un 1 seguido de *(n-1)* ceros. La adición y substracción son más sencillas aquí que en la complementación a *1*, y el resultado siempre queda en notación de complemento a *2*.

1.6.3. Magnitud con signo. En ésta representación el signo es independiente de la magnitud, 0 para signo (+) y, 1 el signo (-). Así, un número positivo tiene un rango de 0 a $2^{n-1}-1$ y un número negativo estará en el rango de: 0 a $-(2^{n-1}-1)$ por lo tanto un número (+) puede ser representado como :

$$A = 0 \quad a_{n-2} \quad a_{n-3} \ \ldots \ a_1 \quad a_0$$

v.g. $A = 0 \quad 1\ 1\ 0\ 1\ 1\ 1\ 0 \quad 0_2 = +220$

y un número negativo será representado así:

$$-A = 1 \quad a_{n-2} \quad a_{n-3} \ \ldots \ a_1 \quad a_0$$

v.g. $-A = 1 \quad 1\ 1\ 0\ 1\ 1\ 1\ 0 \quad 0_2 = -220$

En la notación de magnitud con signo, la versión positiva de *A* difiere de la versión negativa *(-)A* sólo en el dígito de signo. La porción de la magnitud de $[a_{n-2}\ a_{n-3} \ \ldots \ a_1\ a_0]$ es idéntica en ambos casos para *A y (-)A*.

Un problema con la notación de magnitud con signo, es la representación dual del número cero, existiendo el *(-)0 y el (+)0*. Un segundo problema ocurre cuando al sumar dos números de signo

opuesto, la magnitud debe ser comparada para determinar el signo del resultado; esto no ocurre con las otras notaciones. La notación de magnitud con signo es frecuentemente usada en la representación de la mantisa en el formato de punto flotante.

CAPITULO II

LA ARITMÉTICA BINARIA.

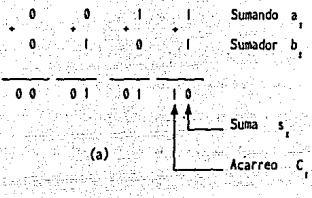
El objetivo de este capítulo es revisar los conceptos básicos de aritmética binaria a partir de bloques de lógica combinacional. las entradas a estos que se suponen valores lógicos, ahora serán interpretados como valores numéricos, explicaremos la forma de sumar y restar números binarios. Después, procederemos a definir la aritmética en sus diferentes representaciones: de complemento a 1, de complemento a 2 y, de magnitud con signo.

2.1 LA ARITMÉTICA DIGITAL.

2.1.1 Suma de dos números binarios. Definamos la suma binaria. En la figura 2.1a sumamos los dígitos a_i y b_i, la suma genera un dígito s_i, ésta puede también generar un dígito de arrastre C_i, cuyo significado numérico es un orden mayor que los dígitos sumandos. La tabla de verdad para generar los bits de suma y de arrastre es la figura 2.1b. El bit de suma lo genera la compuerta XOR, $s_i = a_i \oplus b_i$, mientras que el arrastre lo genera la operación AND, $C_i = a_i b_i$, La estructura del circuito se denomina *sumador medio*, cuyo símbolo está en la figura 2.1d.

2.1.2 El sumador completo. Como en la suma de dos dígitos, ahora, con tres dígitos a_i b_i y C_i se genera una suma s_i y un arrastre C_{i+1}. Las tablas de verdad y los mapas K para s_i y C_{i+1} están en la figura 2.2a y b, respectivamente. Realizaciones de dos compuertas, aparecen en la figura 2.2c. Un circuito adecuado para realizar la adición de tres bits se denomina *sumador completo* y se representa por el símbolo de la figura 2.2d. En el siguiente apartado, la suma columna a columna, requiere combinar tres dígitos : a_i b_i y C_i.

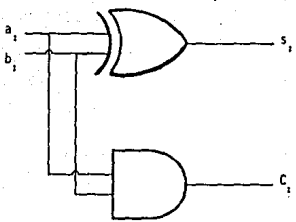
2.1.3 Suma en paralelo. Calculemos la suma de dos números binarios $A = a_{n-1}\ a_{n-2}\ \ldots\ a_0$ y $B = b_{n-1}\ b_{n-2}\ \ldots b_0$ de *n* dígitos. La suma S posiblemente tendrá un dígito más que los sumandos, así que $S = s_n\ s_{n-1}\ s_{n-2}\ \ldots s_0$. Un método de suma, en el que los operandos se suman simultáneamente se muestra en la figura 2.3a. Este método utiliza *n* sumadores completos para efectuar la suma total de dos operandos de A y B *n* bits. Al principio, el primer sumador que



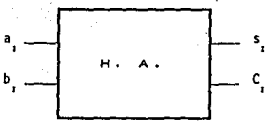
(a)

a_i	b_i	s_i	C_i
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

(b)



(c)



(d)

FIGURA 2.1 (a) SUMA BINARIA. (b) TABLA DE VERDAD DE LA SUMA BINARIA.
(c) ESTRUCTURA DE COMPUERTAS DEL SUMADOR MEDIO.
(d) SIMBOLO DEL SUMADOR MEDIO : H.A. HALF ADDER.

combina a_0 y b_0 podría ser un sumador medio (H.A.), esto si y sólo si, el arrastre de entrada C_I es siempre cero. Entonces, el primer sumador combina a_0 y b_0 para obtener los bits de suma s_0 y arrastre C_0, el bit de arrastre se combina en el siguiente sumador con los bits a_1 y b_1 para generar s_1 y C_1 y así sucesivamente. A este tipo de sumadores también se le conoce como sumador de rizado, por el hecho de que el acarreo generado esta siendo rizado de un sumador a otro. En la figura 2.3b se presenta el diagrama del sumador paralelo utilizando FULL ADDER's. Existe un método para evitar ese rizado que retrasa la generación de la suma final, este método es el conocido como: generador de arrastre posterior (Look ahead carry), pero tal tema queda fuera de los objetivos de la tesis.

2.1.4 Resta binaria. Respecto a la resta binaria de dos dígitos, solamente apuntaremos, que tal operación utiliza elementos similares a los usados en la suma binaria. Presentamos su tabla de verdad y dos ejemplos:

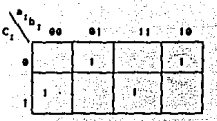
A	B	D	C
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

```
C=  1       1              1 1 1   1
      0 1 1 0 1            1 1 1 0 1 0
    - 1 1 0 1 1          - 0 1 1 1 0 1
    -----------          -------------
D=    1 0 0 1 0            0 1 1 1 0 1
```

Podemos demostrar la veracidad de estos resultados, realizando las mismas restas en complemento a dos. A partir de esta información podemos realizar un análisis similar al propuesto para la suma, pero la omitiremos pues quedaría fuera de los objetivos de ésta tesis.

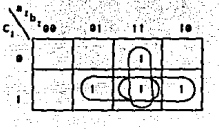
C_i	a_i	b_i	s_i	C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(a)



$$S = \bar{a}_i b_i \bar{C}_i + a_i \bar{b}_i \bar{C}_i + \bar{a}_i \bar{b}_i C_i + a_i b_i C_i$$

(b)



$$C_{i+1} = a_i b_i + a_i C_i + b_i C_i$$

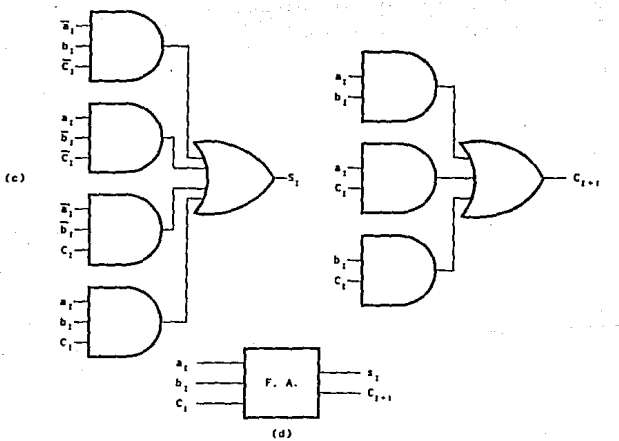
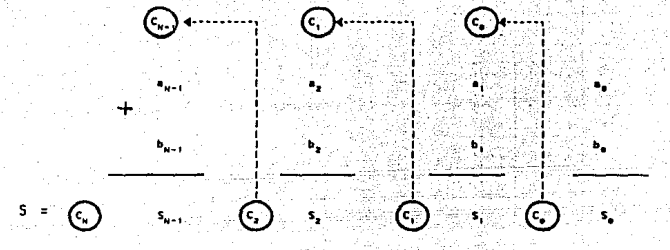
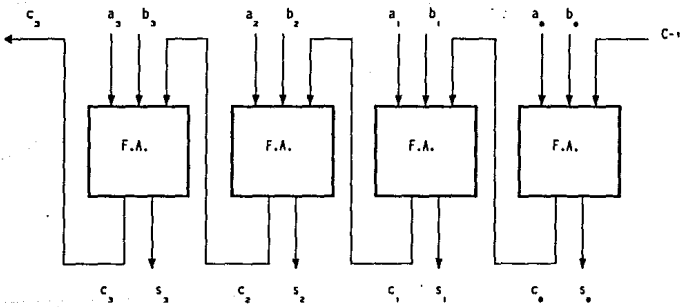


FIGURA 2.2 (a) TABLA DE VERDAD DEL SUMADOR (b) DIAGRAMAS K.
(c) ESTRUCTURA DE COMPUERTAS.
(d) SIMBOLO DEL SUMADOR COMPLETO : FULL ADDER (F.A.).



(a)



(b)

FIGURA 2.3 (a) LA SUMA BINARIA EN PARALELO.
(b) SUMA EN PARALELO USANDO SUMADORES COMPLETOS.

2.2 LA ARITMÉTICA DE PUNTO FIJO.

Ya que sabemos cómo los circuitos digitales pueden realizar la operación básica de la suma binaria, ahora nos corresponde darle un correcto empleo y definir la forma en que son operados los números binarios. Las representaciones usadas forman un sólo grupo dentro de la aritmética binaria, y es conocida como aritmética de punto fijo. Tales números se especifican por su magnitud, por su signo y por su punto binario. La posición del punto binario se necesita para interpretar al número como: entero ó, como fracción. La representación del punto binario en un registro, se complica por el hecho de que se supone en una posición entre dos flip-flops de un registro, pero por conveniencia, se especifica la posición de antemano, y se decide que el punto esté a la extrema derecha (entero) ó, a la extrema izquierda (fracción). Entero [11011101.], fracción [.11011101]. En cualquier caso el punto binario no está presente. La aritmética de complemento a 1 por su carácter dual en el cero no es difundida, así que la omitiremos sin perder información sustancial. La aritmética que nos interesa tratar es la de complemento a dos y la de magnitud con signo.

2.2.1 Suma en complemento a dos. Este procedimiento es muy simple y puede enunciarse como sigue: Sume los dos números, incluyendo el signo del bit, y descarte cualquier acarreo del bit situado más a la izquierda (signo). Dos ejemplos numéricos para la suma de números binarios en complemento a dos, se muestra a

continuación.

+6	0	000110	-6	1	111010
+9	0	001001	+9	0	001001
+15	0	001111	+3	0	000011
+6	0	000110	-9	1	110111
-9	1	110111	-9	1	110111
-3	1	111101	-18	1	101110

2.2.2 Resta en complemento a dos. La resta de dos números binarios en complemento a dos es muy simple y puede enunciarse como sigue: tome el complemento a dos del substraendo (incluyendo el bit de signo) y sumelo al del minuendo (incluyendo el bit de signo). Este procedimiento utiliza el hecho de que una operación de resta puede ser cambiada a una operación de suma si el signo del substraendo se cambia. Esto se demuestra por las siguientes relaciones (*B* es el substraendo):

$$(\pm A) - (- B) = (\pm A) + (+ B)$$

$$(\pm A) - (+ B) = (\pm A) + (- B)$$

Debido a la disponibilidad de procedimientos simples para sumar y restar los números en complemento a dos, la mayoría de los microprocesadores adoptan esta representación sobre la forma más familiar de magnitud con signo. La razón para que el complemento a dos sea usualmente elegido sobre el complemento a uno es la de evitar la ocurrencia de un cero negativo.

2.2.3 Sobreflujo en la suma. Cuando dos números de *n* dígitos se suman y la suma ocupa *n+1* dígitos, decimos que ocurre un *sobreflujo*. Un sobreflujo es un problema en un sistema digital, debido a que la longitud de los registros del procesador y de la misma memoria es finita. Un resultado de *n+1* bits no se puede acomodar en un registro de longitud *n*. Por esta razón muchos sistemas verifican la ocurrencia de un sobreflujo, y cuando ello ocurre, encienden un bit para que el usuario pueda verificar el error. Un sobreflujo no puede ocurrir después de una suma, si un número es positivo y el otro es negativo, puesto que al agregar un número positivo a uno negativo, el resultado (positivo o negativo) es menor en magnitud, que el más grande de los dos números originales. Un sobreflujo puede ocurrir si los dos números que se suman son ambos positivos o ambos negativos. Cuando los dos números están en la representación de complemento a dos con signo y se suman, el bit del signo se trata como parte del número y el acarreo no necesariamente indica un sobreflujo.

Sumar dos números en la representación de complemento a dos, da un resultado incorrecto cuando ocurre un sobreflujo. Esto se presenta debido a que un sobreflujo del número de bits siempre cambia de resultado y da una respuesta errónea de *n* bits. Para observar mejor como sucede esto, consideremos el ejemplo siguiente. Dos números con signo 70 y 80 son almacenados en registros de 8 bits. La máxima capacidad del registro es 127 y la máxima capacidad

negativa es - 128. Puesto que la suma de los números es de 150, esta excede la capacidad del registro.

acarreo	0 1	acarreo	1 0
+70	0 1000110	-70	1 0111010
+80	0 1010000	-80	1 0110000
+150	1 0010110	-150	0 1101010

En el primer caso, el resultado de 8 bits que debería ser positivo tiene un bit de signo negativo. En el segundo caso el resultado de 8 bits debería haber sido negativo y tiene un bit de signo positivo. Note que si el acarreo de la posición del bit de signo se toma como el signo del resultado, entonces la respuesta de 9 bits sería correcta, pero 9 bits simplemente no caben en un sistema de 8 bits. Obviamente la respuesta binaria de 8 bits es incorrecta y sumar números binarios en la representación de complemento a dos, no da el resultado correcto cuando ocurre un sobreflujo. Una condición de sobreflujo puede detectarse observando el acarreo *dentro* de la posición del bit de signo. Si estos dos acarreos no son iguales, la condición de sobreflujo va a ocurrir. Esto lo vemos en los ejemplos anteriores, en donde estos dos acarreos se muestran explícitamente. Si los dos acarreos son aplicados a una compuerta XOR, un sobreflujo se detectaría cuando la salida de la compuerta fuese 1.

Existen algoritmos para la multiplicación y la división en complemento a 2, pero lo que nos interesa tratar son las de magnitud con signo, por ser la fracción del formato en punto flotante representada en esta forma.

2.3 LA ARITMÉTICA DE MAGNITUD CON SIGNO.

2.3.1 Algoritmo de suma y resta. Designaremos la magnitud de los números por *A* y *B*. Cuando los números con signo se suman o se restan, encontramos que hay ocho condiciones diferentes para considerar, dependiendo del signo de los números y de la operación que se realice. Estas condiciones se muestran en la tabla 2.1. La última columna es necesaria para evitar un cero negativo. En otras palabras, cuando dos números iguales se restan, el resultado debería ser +0 y no -0. El algoritmo para la suma y resta se deriva de la tabla y puede enunciarse como sigue: cuando los signos de A y B son idénticos en la suma ó diferentes en la resta, sume las dos magnitudes y mantenga el signo de A al resultado. Cuando los signos de A y B son diferentes en la suma ó iguales en la resta, compare las magnitudes y reste el número menor del número mayor. El signo del resultado es el de *A*, si y solo si $A > B$ ó, el complemento del signo de *A*, si $A < B$. Si las dos magnitudes son iguales, reste *B* de *A* y tome el signo del resultado como un (+). El procedimiento que se debe seguir para signos idénticos en el algoritmo de la suma es el mismo que el de signos diferentes en el algoritmos de resta, y viceversa. El diagrama de flujo para el algoritmo de suma y resta se presenta en la figura 2.4. Los signos A_s y B_s son operados con una compuerta XOR, si la salida de la compuerta es 0 significa que los signos son idénticos, si esta es 1 entonces los signos son diferentes. Para una operación de suma,

los signos idénticos indica que las magnitudes se suman. Para una operación de resta los signos diferentes indican sumar magnitudes

Operación	sume las magnitudes	Reste las magnitudes		
		Cuando A > B	Cuando A < B	Cuando A = B
(+A)+(+B)	+(A+B)			
(+A)+(-B)		+(A-B)	-(B-A)	+(A-B)
(-A)+(+B)		-(A-B)	+(B-A)	+(A-B)
(-A)+(-B)	-(A+B)			
(+A)-(+B)		+(A-B)	-(B-A)	+(A-B)
(+A)-(-B)	+(A+B)			
(-A)-(-B)	-(A+B)			
(-A)-(-B)		-(A-B0)	+(B-A)	+(A-B)

Tabla 2.1 Suma y resta de numeros en magnitud con signo.

Las magnitudes se suman con una operación $EA \leftarrow A + B$, en donde E es un bit de acarreo, este acarreo después de una suma, si es 0, indica un condición de sobreflujo y entonces sería transferido a un bit bandera *AVF* (Adder over flow).

Las dos magnitudes se restan si los signos son diferentes para una operación de suma o idénticos para una operación de resta. Cuando las magnitudes se restan $A - B$ no puede ocurrir un sobreflujo, así que *AVF* estará en 0. Un 0 en E indica que $A \geq B$ y el número en A es el resultado correcto. Si este número es 0, el signo de A debe hacerse positivo para evitar un cero negativo. Un 1 en E indica que $A < B$. Para éste caso es necesario tomar el complemento a dos del valor en A, esta operación puede realizarse con un $A \leftarrow \bar{A} + 1$, sin embargo en otra rutas del diagrama de flujo,

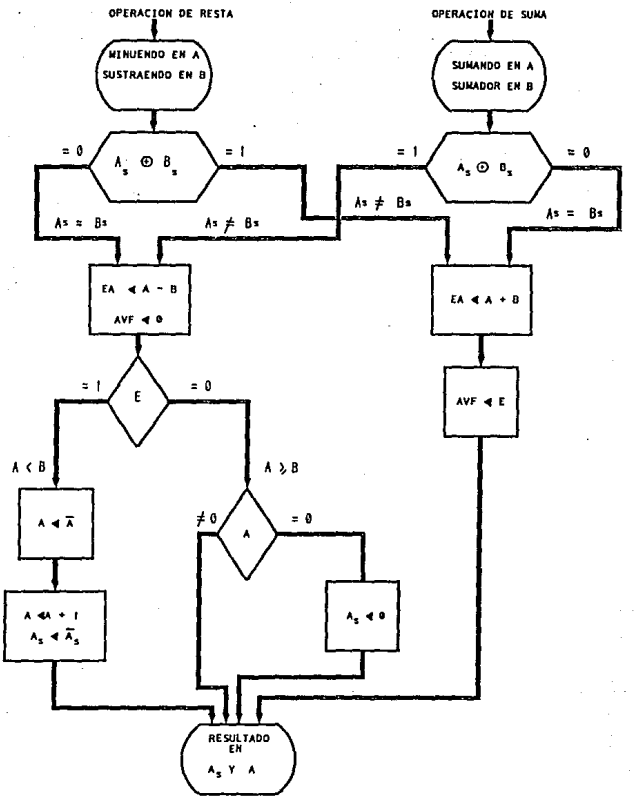


FIGURA 2.4 DIAGRAMA DE FLUJO PARA LAS OPERACIONES DE SUMA Y RESTA EN LA REPRESENTACION DE MAGNITUD CON SIGNO.

el signo del resultado es el mismo que el de A_s, de tal manera que no se requiere cambio en A_s. Sin embargo cuando $A < B$, el signo del resultado es el complemento del signo original de A, es entonces necesario complementar A_s para obtener el signo correcto. El resultado final se encuentra en el registro A y su signo en A_s.

2.3.2 Algoritmo de la multiplicación. La multiplicación de dos números binarios en representación de magnitud con signo se hace por un proceso de operaciones sucesivas de desplazamiento y suma. Este proceso se ilustra mejor con un ejemplo numérico como se muestra en la siguiente figura.

```
   11          1 0 1 1      Multiplicando
 x 6         x 0 1 1 0      Multiplicador
-----      ---------------
  66          0 0 0 0
            1 0 1 1
          1 0 1 1
        0 0 0 0
      ---------------
      1 0 0 0 0 1 0        Producto
```

El proceso consiste en observar los bits sucesivos del multiplicador, primero el bit menos significativo. Si el bit del multiplicador es 1, se copia el multiplicando, de otra manera se copian cero. Los números copiados en forma sucesiva son desplazados una posición a la izquierda con respecto al número previo. finalmente los números se suman y su suma forma el producto. El signo del producto se determina de los signos del multiplicando y del multiplicador: B_s XOR D_s.

Cuando la multiplicación es implementada en un sistema

digital, es conveniente cambiar el proceso ligeramente. Primero, en lugar de proporcionar registros para almacenar y sumar simultáneamente tantos números binarios como haya bits en el multiplicador, con un sumador para hacer las sumas y sucesivamente ir acumulando los productos parciales en un registro es suficiente. Segundo, en lugar de desplazar el multiplicando a la izquierda, el producto parcial, es desplazado a la derecha, y así quedan en las posiciones relativas requeridas. Tercero, cuando el bit correspondiente del multiplicador es cero, no es necesario sumar todos los ceros al producto parcial, puesto que ello no altera su valor. El multiplicador se almacena en el registro *D* y su signo en D_s. El contador de secuencia *SC* es inicialmente puesto a un número que es igual al número de bits del multiplicando. El contador es decrementado en 1 después de formar cada producto parcial. Cuando el contenido del contador llega a ser cero, el producto es formado y el proceso se detiene. Inicialmente el multiplicando está en el registro *B* y el multiplicador en *D*. La suma de *A* que es el producto parcial y B_s forman un producto parcial el cual es transferido al registro *EA*. Ambos productos parciales y el multiplicador se desplazan a la derecha. Este desplazamiento será denotado por el enunciado *shr EAD* para designar el desplazamiento a la derecha. El bit menos significativo de *A* es desplazado a la posición más significativa de *D*. El bit *E* es desplazado a la posición más significativas de *A* y un *0* es puesto en *E*. Después del

desplazamiento, un bit del producto parcial es desplazado en *D*, recorriendo los bits del multiplicador una posición a la derecha. De esta manera la posición más a la derecha del registro *D* designada por D_0, contendrá el bit del multiplicador que debe ser verificado posteriormente.

La figura 2.5 es el diagrama de flujo del algoritmo de multiplicación. Inicialmente, el multiplicando está en *B* y el multiplicador en *D*. Sus signos correspondientes están en B_s y D_s respectivamente. Los signos son operados y, *A* y *D* se colocan para corresponder al signo del producto, puesto que el producto de doble longitud será almacenado en los registros *A* y *D*. Los registros *A* y *E* son puestos a cero y el contador de secuencia *SC* es puesto a un número igual al número de bits del multiplicador. Después de la inicialización, el bit de orden inferior del multiplicador en D_0, se verifica. Si es 1, el multiplicador en *B* se suma al producto parcial presente en *A*. Si es un 0, no se hace nada. El registro *EAD* es entonces desplazado una vez a la derecha para formar el nuevo producto parcial. El contador de secuencia es decrementado en 1 y su nuevo valor es verificado. Si no es igual a cero el proceso es repetido y se forma un nuevo producto parcial. El proceso se detiene, cuando $SC = 0$. El producto parcial formado en *A* es desplazado en *D* un bit a la vez y eventualmente reemplaza al multiplicador. El producto final está disponible tanto en *A* como en *D*, con *A* reteniendo los bits más significativos.

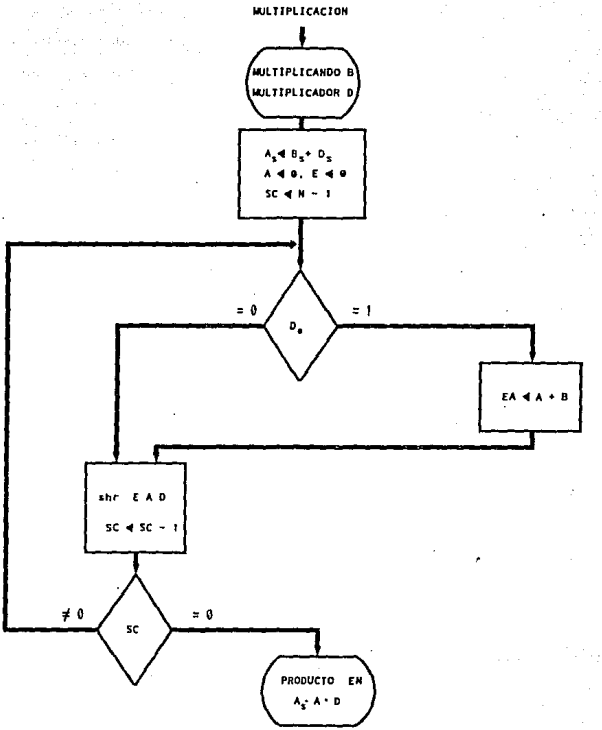


FIGURA 2.5 DIAGRAMA DE FLUJO DE LA MULTIPLICACION EN LA REPRESENTACION DE MAGNITUD CON SIGNO.

2.3.3. Algoritmo de la división. La división de dos números binarios en representación de magnitud con signo se hace con un proceso de operaciones sucesivas de comparación y desplazamiento. La división binaria es más simple que la división decimal, debido a que los dígitos del cociente son 0 o 1, y no hay necesidad de estimar cuantas veces el divisor cabe en el dividendo o residuo parcial. El proceso de división se ilustra con un ejemplo numérico en la siguiente figura. La magnitud del divisor *B* consta de 4 bits y la del dividendo *A* de 8 bits. Para realizar las comparaciones realizaremos una suma entre el dividendo y el complemento a dos del divisor, recordemos que cualquier acarreo generado se descarta.

```
                                            0 1 1 1 1           15
                                      ___________________      ____
                        B = 0 1 1 1 | 0 1 1 0 1 1 0 1        7 |109
                    B̄ + 1 = 1 0 0 1   1 0 0 1                    4
                                      _______
                              A<B     1 1 1 1

Restauramos último                    0 1 1 0 1
dividendo, recorremos B                 1 0 0 1
y restamos                              _______
                              A≥B       0 1 1 0

No restauramos,                         0 1 1 0 1
recorremos y restamos                     1 0 0 1
                                          _______
                              A≥B         0 1 1 0

No restauramos,                           0 1 1 0 0
recorremos y restamos                       1 0 0 1
                                            _______
                              A≥B           0 1 0 1

No restauramos,                             0 1 0 1 1
recorremos y restamos                         1 0 0 1
                                              _______
                              A≥B             0 1 0 0   Residuo
```

Los 4 bits más significativos del dividendo son comparados con el divisor, para esto se realiza una resta obteniendo el complementado a dos del divisor y sumándolo al dividendo. El MSB de este resultado es un 1, significa que es negativo y por tanto $A<B$ entra un 0 en esa posición del cociente. Recorremos una posición a la derecha al divisor respecto al dividendo original, realizamos una segunda resta y el MSB del resultado nos indica que $A>B$ en esta posición, entra un 1 en el cociente. No restauramos el dividendo sino tomamos la última diferencia (residuo parcial) para la siguiente comparación, concatenamos el siguiente bit del dividendo original, que hace el mismo efecto de recorrer al divisor; volvemos a comparar. Esta operación de recorrer y comparar se hace mientras haya bits en el dividendo. Si la última resta fue positiva ($A\geq B$) el residuo final será precisamente esa diferencia. Pero si la resta fue negativa ($A<B$) el residuo será el minuendo de tal resta.

Cuando la división es implementada en un sistema digital, es conveniente cambiar el proceso ligeramente. En lugar de desplazar el divisor a la derecha, el dividendo o el residuo parcial, es desplazado a la izquierda, de esta forma dejamos a los dos números en la posición relativa requerida y, vamos creando el cociente Q a partir de los espacios que van apareciendo a la extrema derecha. Las restas requeridas se logran sumando a A el complemento a 2 de B ó, si se cuenta con una instrucción de resta directa se evita la complementación. La información acerca de las magnitudes relativas

esta disponible en el acarreo final. Por lo tanto los registros para implementar la división son el divisor en *B*, y el dividendo en *AQ*. *AQ* es de doble longitud que *B*. El registro *EAQ* se desplaza a la izquierda con un cero insertado en Q_0 (LSB de Q) y el valor previo de *E* se pierde. El dividendo es desplazado a la izquierda y el divisor es restado. La información acerca de la magnitud relativa esta disponible en *E*. Si *E*=0 esto significa que $A \geq B$. Un bit de cociente 1 es insertado en Q_0 y el residuo parcial es desplazado a la izquierda para repetir el proceso. Si *E*=1 esto significa que $A < B$ de tal manera que el cociente en Q_0 permanece en cero (insertado desde el desplazamiento). El valor de *B* es entonces sumado para restaurar el residuo parcial en *A* a su valor previo. El residuo parcial es desplazado y el proceso se repite de nuevo hasta que se formen los bits del cociente. Note que mientras el residuo parcial es desplazado a la izquierda, los bits del cociente son desplazados también y después de n-1 desplazamientos, el cociente está en *Q* y el residuo final está en *A*.

El algoritmo para dividir se muestra en el diagrama de flujo de la figura 2.6. El dividendo está en *A* y en *Q*, y el divisor en *B*. El signo del resultado es $Q_s = A_s$ XOR B_s. Una constante se coloca en el contador de secuencia *SC* para especificar el números de bits en el cociente. Una condición de sobreflujo de división se verifica restando el divisor *B* del dividendo *A*. Cuando el dividendo es dos veces más largo que el divisor, la condición de sobreflujo debe

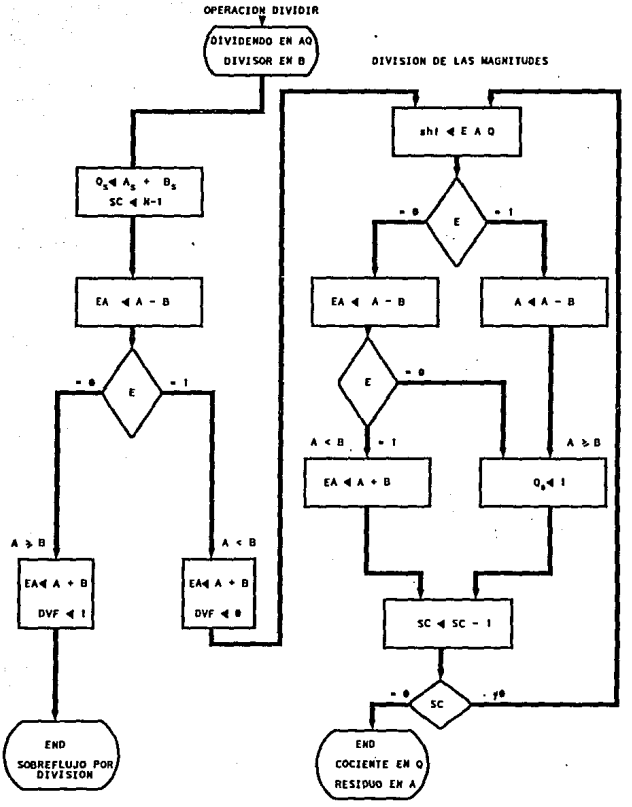


FIGURA 2.6 DIAGRAMA DE FLUJO PARA LA OPERACION DE DIVISION EN LA REPRESENTACION DE MAGNITUD CON SIGNO.

enunciarse como sigue: *Una condición de sobreflujo de división ocurre si la mitad de los bits de orden superior del dividendo constituyen un número mayor o igual que el divisor.* Si E=0 $A \geq B$ y se enciende el bit de sobreflujo de división *DVF*, la operación se termina prematuramente. Si $E=1$ $A < B$, y no ocurre sobreflujo de división de tal manera que el valor del dividendo es restaurado sumando B a A. Otro problema asociado con la división es el hecho de que una división por cero se debe de evitar.

La división de las magnitudes comienza desplazando el dividendo en AQ a la izquierda con el bit de orden superior desplazado en E. Si el bit puesto en E es 1 sabemos que $EA>B$ debido a que EA consta de un 1 seguido de n bits. En este caso, B debe restarse de EA y un 1 insertarse en Q_0 para el bit del cociente. Puesto que el registro A no tiene el bit de orden superior del dividendo (el cual esta en E), su valor es $EA-2^{n-1}$. Sumando a este valor el complemento a 2 de B se obtiene:

$$(EA - 2^{n-1}) + (2^{n-1} - B) = EA - B$$

El acarreo de esta suma no se transfiere a E si deseamos que E permanezca en 1. Si la operación de desplazamiento a la izquierda inserta un 0 en E, el divisor se resta, ya sea sumando su complemento a 2, o si el sistema consta de resta directa, se resta; el acarreo es transferido a E. Si $E=0$ ello significa que $A \geq B$, por consiguiente Q_0 es puesto a 1. Si $E=1$, ello significa que $A<B$ y el número original es restaurado sumando B a A. En el último caso,

dejamos un 0 en Q_0 (Un 0 ya se había insertado en el desplazamiento). Este proceso se repite de nuevo con el registro *A* reteniendo el residuo parcial. Después de *n-1* veces, la magnitud del cociente se forma en el registro *Q* y el residuo se encuentra en el registro *A*. El signo del cociente esta en Q_n y el signo del residuo esta en A_n y es el mismo que el signo original del dividendo.

CAPITULO III

ARITMÉTICA DE PUNTO FLOTANTE

Muchos lenguajes de alto nivel tienen la facilidad para especificar números en punto flotante. La manera más común es especificarlos con un enunciado de declaración *real* ó, si son números en punto fijo, estos se especifican con un enunciado de declaración *entero*. Cualquier computadora que utilice un compilador para tal lenguaje de programación de alto nivel, debe tener una provisión para manejar operaciones aritméticas en punto flotante.

Estas operaciones a menudo se incluyen en el hardware del sistema, ya sea de forma discreta ó, en forma de un circuito integrado VLSI llamado coprocesador aritmético. Pero, si no hay tal hardware disponible en el sistema, el compilador a usar debe contar con un paquete de subrutinas software de punto flotante. Aunque el método de hardware es más costoso, es mucho más eficiente que el método de software. Si embargo los hay sistemas que por su aplicación, tan sólo requieren una unidad de punto flotante como la que describiremos aquí.

En el capítulo mostraremos y explicaremos los algoritmos para la realización de las operaciones aritméticas en el formato de punto flotante. Inicialmente definiremos el formato en sí, su importancia así como sus características y limitaciones. Una vez comprendido el formato de punto flotante procederemos a examinar cada uno de los algoritmos: el de suma y resta, que forman uno solo, el de multiplicación y por último el de división.

La notación de números en la representación de punto fijo es conveniente para representar enteros de valor relativamente pequeño. En la aritmética punto fijo, se debe mantener la posición correcta del punto binario todo el tiempo, y cuando hay algún problema de rangos, se escalan los valores a discreción.

Los coprocesadores aritméticos y unidades aritméticas de punto flotante, manejan el factor de escalamiento automáticamente. Los elementos adicionales, según el caso, complican el sistema relativamente, sin embargo, las operaciones aritméticas son más

eficientes. Consideremos el rango de valores representados por un número de punto fijo de 16-bits.

$$+2^{15} - 1 = +32\ 767$$

con el bit más significativo representando el signo

0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

El máximo valor negativo es $-2^{15} = -32,768$ con el bit más significativo representando al signo

1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

ninguno de estos límites es suficientes para cálculos científicos, que podrían involucrar números como por ejemplo:

34 200 000 000 x 0.0000000762

La multiplicación anterior es más fácil de leer y entender si es escrita en notación científica, como

$$(342 \times 10^8) \times (7.62 \times 10^{-8})$$

La notación de punto flotante no es sino la notación científica.

3.1. EL FORMATO DE PUNTO FLOTANTE. La aritmética de punto flotante ofrece la ventaja de eliminar el problema del factor de escalamiento y, también expande el rango de valores sobre la aritmética de punto fijo. Un número en punto flotante consiste de dos partes: una fracción *f* y un exponente *e*. Las dos partes representan a un número que es obtenido multiplicando *f*-veces una base que es elevada a la potencia *e*; esto es, el número en punto flotante puede ser expresado como

$$A = f \times r^e \qquad (3.1)$$

Donde *f* y *e* son ambas signadas y *r* es la base o raíz. La parte fracción y la parte del exponente han sido llamadas con una gran variedad de nombres, pero ninguno de estos ha sido estandarizado. Por ejemplo, la fracción frecuentemente es referida como la *mantisa,* y el exponente es algunas veces llamado *característica.*

La fracción puede ser representada en cualquiera de los sistemas numéricos de punto fijo descritos en el capitulo 2, esto es: magnitud con signo, complemento a dos o complemento a uno. La mayoría de los sistemas usan la representación del número en magnitud con signo.

Mediante un ajuste del exponente *e*, el punto base puede hacerse "flotar" a través de la fracción; por ésta razón especifica la notación $f \times r^e$ es llamada *representación en punto flotante* de un número *A*. Considere un ejemplo numérico en el sistema decimal (r=10), con representación en punto flotante.

$$A = 0.0000068421 \times 10^{+3}$$

Este puede ser representado en los siguientes dos formatos:

$$A = 0.0000068421, \quad +3$$

$$A = 0.6842100000, \quad -2$$

Donde los enteros +3 y -2 son las magnitudes de los exponentes, con una base implícita de 10. La fracción puede ser recorrida *k* posiciones a la izquierda y, simultáneamente el valor del exponente puede ser incrementado *k*-unidades sin cambiar el valor real del número. La fracción también puede ser recorrida a la derecha con un incremento correspondiente en el exponente. En el ejemplo anterior

la fracción fue recorrida a la izquierda cinco dígitos de posición y el exponente fue incrementado de acuerdo a ello. Este corrimiento de la fracción y escalamiento del exponente ocurre frecuentemente en las operaciones en punto flotante.

Un formato binario de 32-bits para un número en punto flotante es mostrado en la figura 3.1; éste incluye una parte fracción de 23-bits, un exponente signado de 8-bits (positivos y negativos) y, un bit de signo que indica el signo del número. El bit más significativo del exponente (bit 30) indica el signo del exponente en complemento a dos, esto es (+) si el bit 30 = 0, (-) si el bit 30 = 1

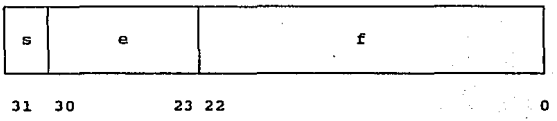


Figura 3.1 Formato de 32 bit para un número en punto flotante.

Puesto que la precisión es importante, debe haber tantos bit significativos en la fracción como sea posible, esto se logra normalizando al número real. La normalización binaria se realiza recorriendo los bits de la fracción a la izquierda hasta que el bit a la derecha inmediata del punto base sea 1, recordando que se debe decrementar el exponente de acuerdo a los corrimientos. Con un exponente en complemento a 2 de ocho bits se tiene un rango de -128 a +127; significa esto, que el factor de escalamiento para base 2

tiene un rango de 2^{-128} a 2^{127}. La magnitud de la fracción normalizada tiene un valor absoluto dentro del rango

$$^1/_2 \leq | f | < 1 \qquad (3.2)$$

lo que dice que cualquier número binario normalizado debe tener una fracción ≥ 0.5. La única excepción es un número en punto flotante igual a 0, pues un 0 no puede ser normalizado porque no tiene ningún dígito no cero, éste se representa en punto flotante mediante una fracción con sólo ceros y con un exponente también con sólo ceros.

3.2. EXPONENTES SESGADOS. El exponente *e* puede ser un entero positivo o negativo. Cuando sumamos o restamos dos números en punto flotante, los exponentes deben ser comparados y forzados a ser iguales, resultando una operación de corrimiento sobre una de las fracciones. La operación de comparación puede ser realizada sin involucrar los signos de los exponentes, para ello se convierten a números positivos todos los exponentes, esto se logra añadiendo una constante positiva a cada exponente cada que un número en punto flotante vaya a ser formado. Internamente, esto hace positivos a todos los exponentes. Esta constante de sesgo (bias), tiene una magnitud igual a la que tiene el exponente más negativo. Si el exponente cubre *m* bits de longitud, entonces la constante de sesgo es $+2^{m-1}$ y, todos los exponentes *e* serán sesgados por 2^{m-1}; esto quiere decir que los exponentes están representados como

$$e_{sesgado} = e + 2^{m-1} \qquad (3.3)$$

Un exponente sin sesgo en la notación complemento a 2 tiene el siguiente rango de valores

$$-2^{m-1} \leq e_{\text{sin sesgo}} \leq 2^{m-1} - 1$$

con ocho bits, como ya vimos, tendríamos un rango de

$$-128 \leq e_{\text{sin sesgo}} \leq 127$$

Después de sumar la constante de sesgo, el exponente se convierte en un entero positivo en el siguiente rango

$$0 \leq e_{\text{sesgado}} \leq 2^{m} - 1 \qquad (3.4)$$

con ocho bits tendremos, todos los exponente positivos de

$$0 \leq e_{\text{sesgado}} \leq 255 \qquad (3.5)$$

Hay dos razones importantes para hacer uso de los exponentes sesgados. La primera de éstas razones, es que todos los exponentes positivos pueden brindar alguna simplificación en el momento de implementar el sistema. La otra razón se refiere a la forma de representar el cero en punto flotante. Matemáticamente, 0 multiplicado por cualquier número es igual a cero; teóricamente, hay varias maneras de representar al cero en la notación en punto flotante, dado que la fracción igual a cero desprecia el valor del exponente. En algunos sistemas cuando el resultado del cálculo es una fracción cero, el exponente es dejado en cualquier valor que resulte al final de la operación. Este resultado es un cero que no es único, y un cero único es deseado en cualquier diseño de algún sistema. En la aritmética de punto fijo, un cero único es representado mediante un número con puros ceros, mientras que en la aritmética de punto flotante un cero puede ser definido como una

fracción 0, con un exponente sesgado, que está en su forma más negativa, esto es, un exponente con puros ceros. Con exponentes sin sesgo, el exponente más pequeño posible, es el exponente más negativo; con un exponente sesgado, el exponente más pequeño posible es el 0. Usando exponentes sesgados, la comparación de los exponentes es relativamente directa, porque los exponentes son positivos; y una simple comparación es suficiente.

Una manera de comparar dos operandos es restar uno del otro y verificar el signo del resultado. Si los exponentes son sesgados nos queda:

$$e_{1\,sesgado} = e_1 + 2^{m-1}$$

$$y\ e_{2\,sesgado} = e_2 + 2^{m-1}$$

La substracción puede ser realizada añadiendo el complemento a dos de $e_{2\,sesgado}$ *a* $e_{1\,sesgado}$. *Si la substracción produce un acarreo de salida en la posición más significativa, entonces* $e_1 < e_2$, si no hay acarreo después de la resta significa que $e_1 > e_2$, si $e_1 - e_2 = 0$ entonces $e_1 = e_2$.

La ventaja de los exponentes sesgados, es que son siempre números positivos y es fácil de comparar sus magnitudes relativas sin tomar en cuenta sus signos y, de ésta manera, se puede usar una simple operación de comparación de magnitud para conocer la magnitud relativa durante el alineamiento de la fracción. Esta es la principal razón para usar sesgo. Otra ventaja es que el valor más pequeño de un exponente sesgado contiene sólo ceros y, por lo tanto la representación del 0 en punto flotante, es una fracción de puros ceros y un exponente también de puros ceros.

3.3. ARITMÉTICA NORMALIZADA DE PUNTO FLOTANTE. En ésta sección se definirán las operaciones aritméticas de punto flotante: adición, substracción, multiplicación y división. Si

$$A_1 = f_1 \times r^{e1}$$

y $$A_2 = f_2 \times r^{e2}$$

Donde **f** es la fracción normalizada, esto es , que su **MSB = 1 ,e** es el exponente y **r** la base. La fracción **f**, que es un operando signado con **n** dígitos significativos (excluyendo el signo), y tiene el siguiente rango:

$$^1/2 \leq |f| \leq 1 - r^{-n} < 1 \qquad (3.6)$$

El termino $1 - r^{-n}$ representa el valor máximo de la fracción; con $n=3$

$$1 - r^{-3} = 1 - 2^{-3}$$
$$= 1 - {}^1/_8$$
$$= {}^7/_8$$

la configuración de bits de la fracción para n=3 sería:0.111 = $1x2^{-1}$ + $1x2^{-2}$ + $1x2^{-3}$ = 0.875 y 0.875 es el valor máximo que puede alcanzar la fracción, con 3 bits. También el exponente sesgado e es un entero positivo con m dígitos significativos y tiene un rango de:

$$0 \leq |e| \leq r^m - 1 \qquad (3.7)$$

Para un exponente de cuatro bits, donde m = 4, el valor máximo de e será

$$e_3 \quad e_2 \quad e_1 \quad e_0$$
$$1 \quad 1 \quad 1 \quad 1$$
$$r^m - 1 = 2^4 - 1 = 15$$

3.3.1. La adición binaria en punto flotante se define como

$$A_1 + A_2 = (f_1 \times 2^{e1}) + (f_2 \times 2^{e2})$$
$$= [f_1 + (f_2 \times 2^{-(e1-e2)})] \times 2^{e1} \quad para\ e_1 > e_2$$
$$= [(f_1 \times 2^{-(e2-e1)}) + f_2] \times 2^{e2} \quad para\ e_1 \leq e_2 \qquad (3.8)$$

Donde $2^{-(e1-e2)}$ es el factor de corrimiento que es multiplicado por la fracción con el exponente más pequeño. La ecuación 3.8 establece que para $e_1 > e_2$, f_1 es sumada a la fracción f_2 alineada (recorrida a la derecha) y la fracción resultante es caracterizada con el exponente mayor. Un ejemplo usando el factor de corrimiento es

$$A_1 = 0 . 1 0 1 1 0 0 \times 2^5$$
$$A_2 = 0 . 1 1 1 0 0 0 \times 2^2$$

la fracción f_2 tiene el exponente menor y $e_1 - e_2 = 3$. Por lo tanto

$$f_{2\ alineada} = 0 . 1 1 1 0 0 0 \times 2^{-(3)}$$
$$= 0 . 1 1 1 0 0 0 \times 1/8$$
$$A_2 = 0 . 0 0 0 1 1 1 \times 2^5$$

Esto indica que f_2 alineada = $f_2/8$. La división por 8 se logra mediante un corrimiento a la derecha de tres posiciones, que alinea propiamente a f_2 con f_1. La ecuación 3.8 muestra que el punto binario de cada uno de los dos operandos A_1 y A_2 debe ser alineado antes de que la adición sea realizada. Esto se consigue comparando las magnitudes relativas de los dos exponentes y recorriendo la fracción con el exponente más pequeño $| e_1 - e_2 |$ posiciones de bits a la derecha. La adición de las fracciones, procede, entonces, con el exponente más grande usado como exponente del resultado de la suma.

3.3.2. La substracción en punto flotante está definida como

$$A_1 - A_2 = (f_1 \times 2^{e1}) - (f_2 \times 2^{e2})$$
$$= [f_1 - (f_2 \times 2^{-(e1-e2)})] \times 2^{e1} \quad para \quad e_1 > e_2$$
$$= [(f_1 \times 2^{-(e2-e1)}) - f_2] \times 2^{e2} \quad para \quad e_1 \leq e_2 \qquad (3.9)$$

El comentario mencionado para la adición lo es también para la substracción. La misma comparación y procedimiento de alineación es usado.

3.3.3. La multiplicación en punto flotante está definida como

$$A_1 \times A_2 = (f_1 \times 2^{e1}) \times (f_2 \times 2^{e2})$$
$$= f_1 \times f_2 \times 2^{e1+e2} \qquad (3.10)$$

La multiplicación en punto flotante requiere una multiplicación de las fracciones en punto fijo y, una adición en punto fijo del los exponentes. Cuando una operación de multiplicación toma lugar de acuerdo a la ecuación 3.10, el valor de la fracción resultante está en el rango $^1/2^2 \leq | f_1 \times f_2 | < 1$, para f_1, f_2 diferentes de 0, y cae dentro del rango $^1/2^2 \leq | f_1 \times f_2 | < {}^1/_2$, una normalización es requerida. Un corrimiento a la izquierda de un bit de posición es suficiente para normalizar el resultado. La configuración binaria para este rango es 0.010 ... $0 \leq | f_1 \times f_2 | <$ 0.011 ... 1; en este caso, la ecuación 3.10 es reemplazada por

$$A_1 \times A_2 = [(f_1 \times f_2) \times 2] \times 2^{(e1+e2)-1}$$

Esto es, el resultado es multiplicado por 2, lo cual es equivalente a recorrer a la izquierda un bit de posición, y el exponente resultante es decrementado en 1. Sin embargo, cuando el resultado cae dentro del rango $^1/_2 \leq | f_1 \times f_2 | < 1$ sabemos que

una posterior normalización no es requerida, porque el producto resultante está en forma correctamente normalizada.

3.3.4. La división en punto flotante está definida como

$$\frac{A_1}{A_2} = \frac{f_1 \times 2^{e1}}{f_2 \times 2^{e2}}$$

$$= \frac{f_1}{f_2} \times 2^{e1-e2} \qquad (3.11)$$

Cuando una operación de división toma lugar según la ecuación 3.11, el valor de la fracción resultante está en el rango

$$\frac{1}{2} \leq \left| \frac{f_1}{f_2} \right| < 2$$

para f1 ≠ 0, f2 ≠ 0, y el dividendo menor al divisor. Esto indica que una posterior normalización no es requerida. Sin embargo, cuando el dividendo es mayor o igual que el divisor, entonces un sobreflujo de la fracción ocurre por que

$$1 \leq \left| \frac{f_1}{f_2} \right| < 2$$

y la ecuación 3.11 se vuelve

$$A_1/A_2 = [(f_1/f_2) \times 2^{-1}] \times 2^{(e1-e2)+1} \qquad (3.11b)$$

El termino 2^{-1} recorre al cociente un bit de posición a la derecha, y el exponente resultante es incrementado en 1.

3.4. SOBREFLUJO Y SUBFLUJO EN LA ARITMÉTICA DE PUNTO FLOTANTE. Algunas complicaciones pueden ocurrir como resultado de las operaciones en punto flotante. Una adición puede generar un acarreo de salida en la posición del bit más significativa, resultando un *sobreflujo en la fracción*. Esto es, cuando se suman dos números del mismo signo ó, cuando se restan dos números de diferente signo. Una situación puede ocurrir donde $1 \leq f < 2$, esto establece, que la fracción resultado puede estar en el rango 1.00 ·· 0 a 1.11·· 1 . Este problema puede ser resuelto mediante un simple corrimiento de la fracción a la derecha y, el acarreo de salida tomará la posición MSB.

$$A_1 + A_2 = \{[\ f_1 + (f_2 \times 2^{-(e1-e2)})\ ] \times 2^{-1}\ \} \times 2^{e1+1} \quad para \quad e_1 > e_2$$
$$= \{[(\ f_1 \times 2^{-(e2-e1)}) + f_2\ ] \times 2^{-1}\ \} \times 2^{e2+1} \quad para \quad e_1 \leq e_2$$

(3.12)

El termino 2^{-1} en la ecuación 3.12 es el factor de corrimiento, que mueve el resultado un bit de posición a la derecha. Este factor de corrimiento es $^1/_2$, el cual divide el resultado en 2. También el exponente es incrementado en 1. Hay que notar la similitud entre las ecuaciones 3.12 y 3.8, la primera representa una suma que no requiere ningún corrimiento. La ecuación para una operación de substracción en punto flotante, que corrige un sobreflujo de la fracción es dada por la ecuación siguiente

$$A_1 - A_2 = \{[\ f_1 - (f_2 \times 2^{-(e1-e2)})\ ] \times 2^{-1}\} \times 2^{e1+1} \quad para \quad e_1 > e_2$$
$$= \{[\ f_1 \times 2^{-(e2-e1)}) - f_2] \times 2^{-1}\} \times 2^{e1+1} \quad para \quad e_1 \leq e_2$$

(3.13)

Cuando se alinean los operandos y se ajustan los exponentes durante una suma o resta, el MSB = 1 de la fracción de exponente menor puede perderse en la extrema derecha cuando esta es recorrida a la derecha, a esto se le conoce como *subflujo de la fracción*. Otra complicación aparece cuando la fracción resultante de una operación aritmética es 0; en éste caso, el exponente es puesto a cero, además la fracción no puede ser normalizada. *Un número con una fracción 0 y un exponente 0 se le llama un 0 verdadero. Un cero verdadero puede resultar de una operación aritmética debido a la magnitud particular de el operando. Se puede forzar a un número ser un 0 verdadero cuando:*

1. El resultado de la fracción en una suma o resta es 0.
2. Una o ambas fracciones en una multiplicación son 0.
3. La fracción del dividendo en una división es 0.
4. Cuando ocurre un subflujo en el exponente.

Durante una multiplicación o una división, los exponentes son sumados o restados respectivamente; cuando los exponentes son sumados, el número resultante puede ser muy grande, inclusive exceder el limite superior permitido en el campo del exponente, a esto se le conoce como *sobreflujo de exponente*. Cuando los exponentes son restados, el resultado puede ser un exponente muy pequeño para ser representado, a esto se le conoce como *subflujo del exponente*, e indica que el número excede el mínimo valor permitido. Si durante una operación de suma o resta, la diferencia en los exponentes corresponde a un número de corrimientos que es

mayor al número total de bits de la fracción, entonces el resultado es igual al operando de mayor magnitud, indicando que uno de los operandos es insignificante cuando es comparado con el otro operando. Sería más lógico terminar el corrimiento exactamente después de *n*-corrimientos. La lógica puede ser diseñada para detectar si $|e_1 - e_2| > n$; en este caso, los corrimientos no ocurrirían, y el resultado es puesto igual al operando más grande. *Siempre que ocurra un sobreflujo o subflujo del exponente, una señal deberá ser generada por el sistema.*

3.5. PRECISIÓN. Si la fracción tiene una longitud de *n*-bits, entonces el número puede ser representado con *n*-bits de precisión. En otras palabras, la precisión de un sistema numérico en punto flotante será el número de dígitos de la fracción. La precisión simple, se refiere a aquellas operaciones definidas con operandos estándar de 32-bits. La doble precisión, simplemente, es el doble de la longitud del formato, esto es, 64-bits. El estándar propuesto para la simple precisión y para la doble precisión en el formato de punto flotante por la IEEE es:

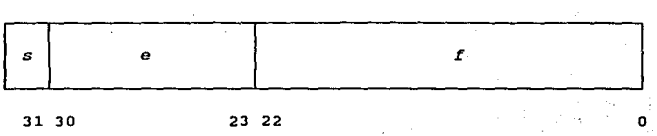


Figura 3.2a Estándar de la IEEE de simple presición.

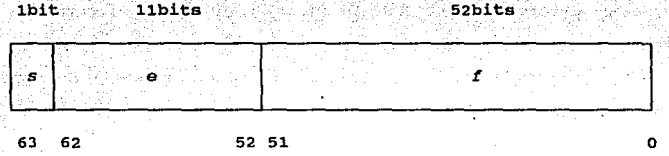


Figura 3.2b Estándar de la IEEE de doble precisión.

Hay numerosos formatos para punto flotante y, la mayoría de compañías tienen diferentes diseños. Algunos diseños también incluyen elementos para precisión simple extendida y para doble precisión extendida. En la multiplicación, la precisión de un resultado es al suma de la precisión de los operandos. Los formatos de doble precisión son necesarios para incrementar el número de bits significativos en la fracción, permitiendo aumentar la exactitud o el rango de valores. Debe ser notado que, cuando los números son introducidos a una computadora, despreciando su exactitud original, estos quedan limitados a la exactitud de la máquina. Después de sumar o restar dos números de n-bits de precisión simple, el resultado será de precisión simple, pero puede haber un acarreo de salida y, esto puede considerarse como un formato de precisión múltiple. Cuando se multiplican dos números de n-bits de precisión simple, el producto será un número $2n$-bits (o $2n - 1$ bits), que es también doble precisión. Si sólo la precisión simple es permitida en el producto, entonces la mitad inferior debe ser truncada o redondeada. Lo mismo, cuando se dividen un dividendo de $2n$-bits y un divisor de n-bits, se produce un cociente de n-bits

y un residuo también de *n*-bits, entonces una palabra de doble precisión es requerida para archivar el cociente y el residuo.

Ya sea que se requiera la precisión simple o la precisión doble para una operación en particular, el sistema de punto flotante puede ser condicionado para operar en un modo particular; esto puede lograrse definiendo una instrucción de longitud de campo.

3.6. ALGORITMO DE ADICIÓN Y SUBSTRACCIÓN. La adición y la substracción serán tratadas en el mismo algoritmo, puesto que la substracción es una suma con uno de los operandos complementado. Ambos operandos se les considera con sus fracciones normalizadas y sus exponentes sesgados. La operación de adición requiere que los exponentes de los operandos sean iguales, así que la fracción cuyo exponente sea menor, será recorrida a la derecha y su exponente ajustado. El algoritmo de la adición y substracción puede ser dividida en seis pasos consecutivos :

1. Verificación de operandos ceros.
2. Alineamiento de las fracciones.
3. Adición o substracción de las fracciones.
4. El exponente del resultado es el exponente mayor.
5. Normalizar el resultado si es necesario.
6. Verificar el subreflujo o subflujo.

Los registros requeridos para la operación de adición o substracción son los registros AR y BR, cada uno consta de tres

partes: el registro para el signo, el registro para el exponente y el registro para la fracción. En donde **a** es el registro de exponente, **A**$_s$ es el registro de signo y **A** es el registro de la fracción. El operando **BR** es definido de forma similar.

El diagrama de flujo para el algoritmo de la adición o substracción se muestra en la figura 3.3. Antes de empezar la operación actual los dos números en punto flotante están cargados en los registros **AR** y **BR** respectivamente. Los operandos están en forma normalizada y el resultado ya sea suma o diferencia será también normalizado y cargado en el registro **AR**. La operación será **AR** ← **AR** ± **BR**. El primer paso es la verificación de ceros, si **BR** es cero la operación termina con el resultado en **AR**. Si **BR** no es cero, se pregunta por el contenido de **AR**, si **AR** es cero, el contenido de **BR** se transfiere a **AR**, si la operación que se iba a realizar era una substracción, se invierte el signo del resultado, si era una adición se conserva igual. Si ambos operandos son diferente de cero la operación continua. El siguiente paso es comparar la magnitud relativa de los exponentes **a** y **b**. Si los dos son iguales, esto indica que las fracciones están correctamente alineadas y la operación aritmética puede ser realizada; pero si no son iguales, entonces la fracción de menor exponente será recorrida a la derecha y su exponente incrementado en 1, este proceso se repite hasta que los dos exponentes sean iguales. Una vez que las fracciones han sido correctamente alineadas, la secuencia puede proceder a la adición o substracción de los operandos, según se haya escogido

ésta previamente. Si la operación es una adición y los signos de los operandos son iguales (A_s XOR B_s = 0), entonces las fracciones son sumadas y cualquier acarreo generado estará en E. Si los signos no son los mismos, entonces la operación es equivalente a una substracción. Un proceso similar ocurre si la operación es una substracción; si los signos son diferentes (A_s XOR B_s = 1) entonces las fracciones son simplemente sumadas y, cualquier acarreo generado es colocado en E. La posnormalización de la fracción la mostramos en la figura 3.3. Cuando el resultado de una substracción genera un acarreo significa que el resultado es negativo y que está en complemento a dos; como nuestra representación para punto flotante es de magnitud con signo, entonces a ese resultado lo complementamos a dos e invertimos su signo; pero si no hubo ningún acarreo después de la substracción el resultado es positivo y no necesita cambios, tan sólo verificar si es o no cero el resultado de tal substracción; si la fracción fue cero, se pone todo el resultado = 0: fracción, exponente y signo. El resultado haya sido positivo o negativo se tiene que normalizar, así que en ambos casos se pregunta por el bit mas significativo, si no es 1 se recorre una vez a la izquierda la fracción y se decrementa el exponente en 1, esto se repite hasta que el MSB sea 1. Ya obtenida la normalización se verifica un posible sobreflujo o un posible subflujo, en ambos casos esto se realiza sobre el exponente, si el exponente es mayor al limite mayor permitido para un exponente, se encenderá un testigo de sobreflujo en la suma: ADD OVF, pero si el exponente es

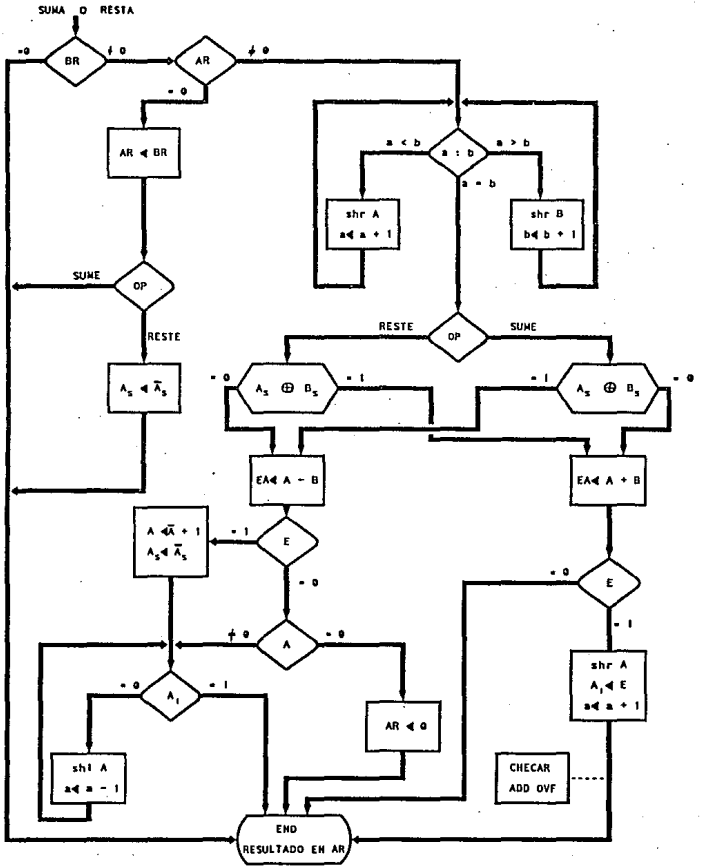


FIGURA 3.3 DIAGRAMA DE FLUJO PARA LA OPERACION DE SUMA Y RESTA EN PUNTO FLOTANTE.

menor al límite menor, entonces se puede encender un testigo de subflujo ó, una opción también válida es dejar el resultado igual a cero. Volviendo al caso de una suma verdadera de fracciones, si se generó un acarreo de salida sólo hay que recorrer una vez a la derecha el acarreo y la fracción, e incrementar en 1 el exponente. Pero alguien puede preguntar si se puede dar el caso en el que el acarreo lo mismo que el MSB de la fracción resultante sean cero. No. Como sabemos, durante el alineamiento de las fracciones, una fue recorrida y la otra conservo su MSB = 1, así que cuando se sumaron, el MSB = 1 paso a ser el MSB de la f resultante. Si no hubo acarreo generado, no hay necesidad de normalizar. Respecto al signo del resultado, para el caso de una suma verdadera de fracciones, este permanecerá con su valor.

Es posible que a la hora de normalizar y operar un incremento sobre el exponente, este puede salirse del rango permitido y, causar por lo tanto un sobreflujo, en este caso habrá que encender un testigo de sobreflujo en la suma : ADD OVF.

3.7. ALGORITMO DE LA MULTIPLICACIÓN. La multiplicación de dos números en punto flotante requiere que las fracciones sean multiplicadas y que los exponentes sean sumados. La fracción es multiplicada a manera de punto fijo. La multiplicación es realizada sobre dos operandos normalizados en punto flotante utilizando los exponentes sesgados. El signo del producto es determinado por los signos de la fracción, A_s XOR B_s. Sin embargo si alguno de los

operandos es cero o ambos, el signo del resultado es positivo. El algoritmo de la multiplicación en punto flotante consta de seis partes principales:

1. Verificación de operandos ceros.
2. Determinar el sigo del producto.
3. Sumar los exponentes.
4. Multiplicar las fracciones.
5. Normalizar el producto.
6. Verificación de sobreflujo y de subflujo.

Los registros requeridos son tres: AR, BR, y DR. Los registros AR y BR contienen operandos normalizados en punto flotante. El diagrama de flujo del algoritmo para la multiplicación se muestra en las figura 3.4. Inicialmente el multiplicando está en AR, y el multiplicador en BR y, un registro D es puesto a cero. Los dos operandos son verificados para determinar si alguno de ellos es cero, si resulta que alguno fue cero, el producto será cero. Si ninguno de los operandos fue cero, el proceso continua con la determinación del signo del producto, A_s XOR B_s. La secuencia continua con la suma de los exponente. La suma se transfiere a el registro a. Al sumarse los exponentes, la suma tendrá el sesgo dos veces, así que será necesario restar una vez el sesgo; las sumas y restas en esta secuencia son en complemento a dos, a partir de esto, podemos determinar un posibilidad de un error por sobreflujo.

La multiplicación de las fracciones se hace de la misma manera que la multiplicación en punto fijo, es decir a partir de sumas y

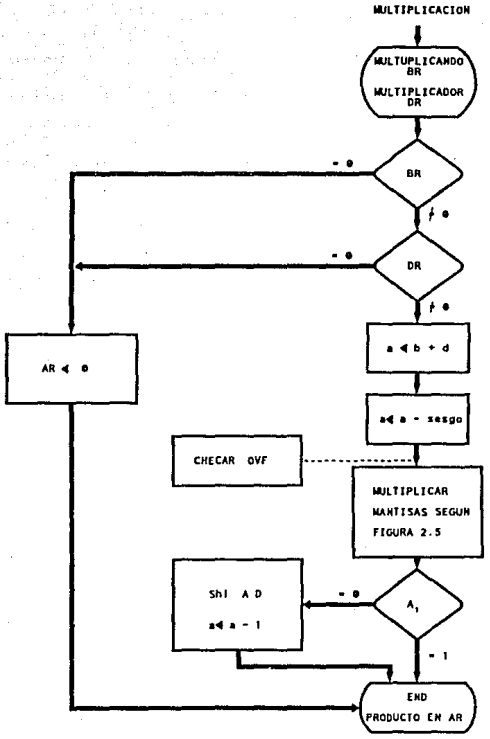


FIGURA 3.4 DIAGRAMA DE FLUJO PARA LA MULTIPLICACION EN PUNTO FLOTANTE.

corrimientos. SC es la secuencia que es cargada con el número de bits de la fracción, A contiene el producto parcial inicial, que es cero; el valor LSB D_0 es verificado, si es 1, se suma el multiplicando y el producto parcial, si es 0 no se suman; en ambos casos el siguiente paso será recorrer una posición a la derecha los registros E A D en serie. El conteo de secuencia es decrementado en 1, el proceso continua con la inspección del bit de siguiente orden del multiplicador y concluye cuando SC = 0. En ese momento, todos los bits del multiplicador han sido examinados. EL valor de la fracción está dentro del rango $0.25 \leq f \leq 1$ y un cero existe en el MSB, y por tanto será necesario normalizar la fracción. Puesto que el producto total y correcto consta de 2n bits, la parte baja de la f será truncada para tener una fracción según el formato establecido de n bits para la fracción. Una vez sumados los exponentes, multiplicadas las fracciones y normalizado el producto, quedaría solamente verificar el sobreflujo del exponente. Para este paso, el problema puede suceder así: si después de sumar los exponentes y de restar una vez el sesgo, existiese un acarreo, esto indica un sobreflujo, y será necesario encender una bandera de estado para que pueda ser verificada por el usuario.

3.8. LA DIVISIÓN. La división de dos números en punto flotante requiere que las fracciones sean divididas en punto fijo y, que los exponentes sean restados en complemento a dos. Ambos operandos tienen sus fracciones normalizadas y sus exponentes

sesgados. El algoritmo de la división en punto flotante se divide en seis partes:

1. Verificación de operandos ceros.
2. Determinación del signo del cociente.
3. Alineación del dividendo.
4. Resta de los exponentes.
5. División de las fracciones.
6. Verificación de sobreflujo.

Son tres los registros que se usan en el procedimiento: AR será el dividendo y BR será el divisor, la parte baja del registro dividendo la llamaremos Q, que al final tendrá el cociente. EL diagrama de flujo de la división en punto flotante se muestra en la figura 3.5. El dividendo es cargado en AR y Q=0, el divisor será cargado en BR. Primero se revisan los operandos, si el divisor es cero, se enciende una bandera de división entre cero y termina la operación. El signo A_s del dividendo permanece igual, puesto que será el signo del residuo, Q_s es A_s XOR B_s. Después, si la parte alta del dividendo (contenido en A) es mayor o igual al valor del divisor, entonces el dividendo debe ser alineado. La magnitud relativa de las dos fracciones se conoce con una resta $A \leftarrow A-B$, (en complemento a dos) y el bit de acarreo nos lo dirá; si el acarreo E=1 el resultado es negativo, entonces $B > A$; si E=0 el resultado de la resta es positiva o cero, determinándose que $A \geq B$, en tal caso el alineamiento de A será necesario. Para realizar el alineamiento hay que restaurar los datos que se cambiaron durante

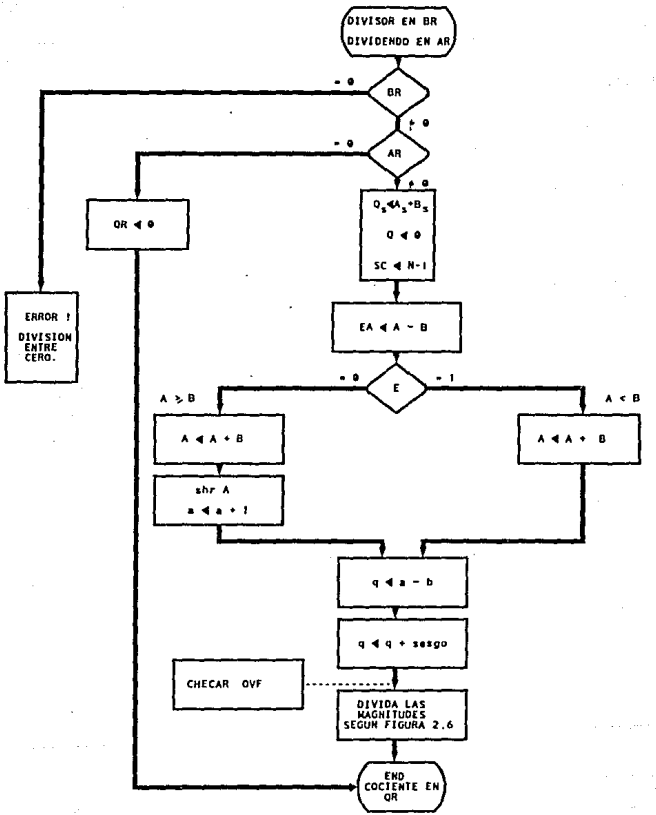


FIGURA 3.5 DIAGRAMA DE FLUJO PARA LA DIVISION EN PUNTO FLOTANTE.

la resta anterior, así que se realiza una suma A ← A + B. El alineamiento de las fracciones se logra con un simple corrimiento a la derecha de A y un incremento en su exponente. Si el alineamiento no fue requerido, de todas formas habrá que restaurarse los datos. Los exponentes se restan a-b y se examina el acarreo de salida, si el acarreo E=1 significa que a < b, y si E=0 significa que a ≥ b, en ambos casos se restaura una vez el sesgo, y se volverá a examinar el acarreo. Si para a ≥ b E=0 entonces no hay problemas de sobreflujo y continua la secuencia de la división, pero si E=1, el exponente resultado se sale del límite superior del formato, y habrá que encender una bandera de sobreflujo. Si se dio el caso de que a < b y después de sumar una vez el sesgo E=1 significa que no hay problemas de subflujo y la secuencia continua, pero si E=0 significa un subflujo del exponente, esto es, que el resultado se fue más abajo del límite inferior del formato, en este caso se puede encender una bandera de subflujo y dejar como resultado el valor mínimo del formato (un cero verdadero) y, terminar la operación. La siguiente secuencia es la división de las fracciones, que es exactamente la misma que se explico en el formato de punto fijo; con la diferencia de que aquí, estamos dividiendo fracciones. El registro A contendrá la fracción del dividendo, B la del divisor y Q=0. El contador de secuencia es cargado al número de bits de fracción. Al final de la secuencia Q contendrá a la fracción cociente, y A el residuo que puede o no despreciarse. No hay necesidad de normalizar al cociente.

CAPITULO IV

LA UNIDAD ARITMÉTICA

DE PUNTO FLOTANTE DEL SIMMP1.

En éste capítulo realizaremos todos los conocimientos anteriores sobre el sistema numérico binario, la aritmética binaria y, sobre todo la aritmética de punto flotante.

El objetivo será realizar las rutinas en lenguaje emsamblador de las operaciones aritméticas: suma, resta, multiplicación y división, en punto flotante.

Tales rutinas deben cumplir con ciertos requisitos o aspectos. Uno de estos es, que tanto datos como resultados, deben expresarse según el estándar propuesto por la IEEE para el formato de punto flotante de 32 bits; se persigue esto, para que en caso de comunicar estos datos o resultados, con algún otro sistema que siga el mismo estándar, no se encuentre con problemas de esta índole. Otro requerimiento que nos hemos propuesto, es de hacer lo posible para que los datos que se operen en cada una de las rutinas, se haga en su totalidad dentro de la misma CPU Z80; así, los registros (AR, BR y demás) presentados en el capítulo anterior, y formateados a 32 bits según IEEE, tendrán que distribuirse para su operación en los registros internos de la CPU Z80. Esta tarea no representará ningún problema, puesto que la CPU Z80 consta de suficientes registros de propósito general, además de los registros alternos, y así distribuir los datos y demás parámetros necesarios en cada operación.

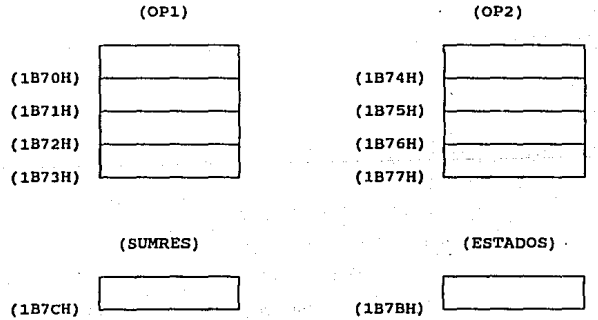
Las rutinas aritméticas seguirán los pasos de los algoritmos presentados en el capítulo anterior y, se apoyarán en los conceptos ahí formulados.

Lo primero que definiremos, será el formato estándar de la IEEE para números en la representación de punto flotante de 32 bits. El estándar dice que se tomarán 23 bits para la mantisa, 8 bits para el exponente, y un bit para el signo de la mantisa. La representación de la mantisa será en la conocida por magnitud con signo, el exponente, que ubicará la posición del punto (por tal

razón lo de punto flotante), se le trabajará siempre con un sesgo, de tal manera que todo exponente siempre será positivo. El signo de la mantisa será un bit = 0 cuando esta sea positiva y, cuando sea negativa un bit = 1. La IEEE propone para la mantisa el formato normalizado, donde el MSB = 1 y el punto binario reside una posición a su derecha, este formato es el más común y con el cual el usuario esta más acostumbrado. En estos formatos normalizados existen dos tipos, los no extendidos y los extendidos. En los no extendidos sólo la parte fracción de la mantisa se guarda en memoria y el MSB es implícito e igual a 1. En los formatos extendidos el MSB es explícito y reside en memoria junto con su fracción. Nosotros hemos elegido trabajar con la mantisa del tipo no extendida con el MSB = 1 explícito. El punto binario residirá a la extrema izquierda convirtiendo nuestra mantisa en sólo fracción. Esta elección estriba en que, trabajando así, al proceso se le facilita la operación de la mantisa; además que datos y resultados pueden, si es necesario, convertirse fácilmente a uno de los otros tipos de formatos normalizados. En lo que respecta a el exponente, la IEEE define que con 8 bits, supuestos en complemento a dos, se tendrán valores que van de -128 hasta +127, al sesgar cualquier valor en este rango, se convertirá en uno forzosamente positivo, y con un nuevo rango de 0 a 255.

Los registros en donde serán guardados y tomados los datos y los resultados respectivamente, son (OP1) y (OP2), de cuatro bytes cada uno y, serán los mismos para todas las

operaciones. En (OP1) estará el sumando para la suma, minuendo para la resta, multiplicando para la multiplicación y dividendo para la división. En (OP2) estará el sumador para la suma, el substraendo para la resta, el multiplicador para la multiplicación y el divisor para la división. El registro (SUMRES) es un byte que contiene información para identificar la operación a realizar en el programa de suma y resta: 00H para realizar una suma y, 01H para realizar una operación de resta. El registro (Estados), proporciona al usuario condiciones de excepción para diagnosticar anomalías, tales como sobreflujo en la suma, sobreflujo en la multiplicación, sobreflujo en la división y división entre cero. El bit 4 (ADCDAC) de ESTADOS se expilca en el apéndice C. Todos estos registros se presentan a continuación en forma gráfica.

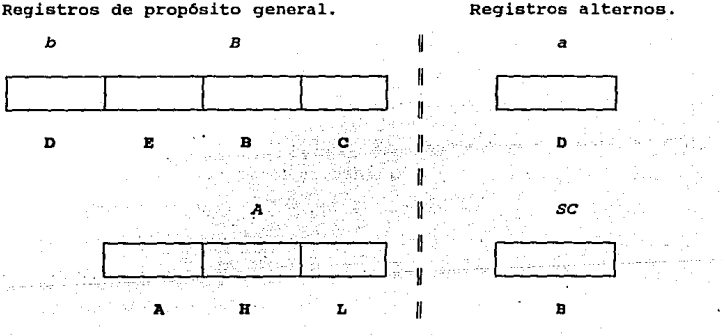


Estas son direcciones válidas del mapa de memoria del SIMMP1.

Con todo lo anterior dicho, procederemos a explicar en forma general cada uno de los programas. Comenzaremos por el programa que realiza las operaciones de sumar y restar números en formato de punto flotante. Lo hemos llamado SUMFLOT.

4.1. SUMFLOT.

Los registros *AR* y *BR* según el algoritmo de suma y resta del capítulo anterior, fueron distribuidos en los archivos de la CPU Z80 de la siguiente forma.



Las letras en cursivas indican registros del algoritmo, y las letras en negritas los registros internos de la CPU Z80.

Aunque en el programa se usan instrucciones de intercambio de

datos entre los registros de propósito general y los registros alternos, esta es la disposición genérica e inicial.

Lo siguiente es la explicación del programa emsamblador de la rutina SUMFLOT.

Los operandos los tenemos inicialmente en (OP1) y en (OP2). El primer paso será verificar los ceros, así que, siguiendo los pasos del algoritmo del capítulo anterior, preguntamos por *BR* que esta en (OP2). Cargamos en el acumulador de la CPU el byte donde se encuentra el MSB (1B76H), recorremos el bit 6 hasta el acarreo C. Si C=0 significa que la fracción de *BR* es cero y el resultado de la operación es *AR*, sin importar si se iba a realizar una suma o una resta. Si C=1 significa que nuestra fracción normalizada de *BR* no es cero y podemos preguntar por *AR*. Cargamos en el acumulador de la CPU el byte donde se encuentra el MSB de la fracción de *AR*, y le aplicamos el mismo procedimiento. Si C=1 significa que *AR* no es cero y podremos continuar la operación. Si C=0 significa la fracción de AR es cero, y el resultado final será *BR*. Sabiendo que el resultado de la operación es *BR*, sólo bastará preguntar que operación se iba a realizar. Tomamos el número que tenemos en el registro (SUMRES) si es una suma la que íbamos a realizar el resultado es *BR* sin ninguna alteración. Si se trata de una resta, cambiamos el signo de *BR* y, lo colocamos en (OP1) terminando así la operación.

Una vez que ya sabemos que ninguno de los dos operandos es cero. Vamos a OKSUM, en donde cargamos correctamente a (OP1) y a

(OP2) en los registros de la CPU. En este paso, recorremos una vez a la izquierda a ambos números, con el objeto de eliminar temporalmente el signo y, tengamos ocupando a las fracciones y a los exponentes registros de 8 bits sólo para ellos.

A continuación, comparamos los exponentes de los operandos para alinear correctamente las mantisas y se pueda realizar la suma o resta de éstas. En COMPA, cargamos una cuenta o secuencia, a un número igual a 24 (18H en hexadecimal), que es el número de bits de nuestras fracciones, con el fin de que no se hagan más comparaciones de las necesarias. Comparamos los exponentes, movilizando los registros D y D' con el acumulador de la CPU. Realizamos una resta *a-b*, si hay un 1 en acarreo, significa que el resultado de esta resta es negativo y por tanto a < *b*, vamos a ALTHB. Si la bandera Z=1 significa que *a-b*=0 y por lo tanto *a=b*, vamos a AEQB. Si C=0 significa que a > *b*, vamos a AGTHB; aquí, recorremos a la izquierda una posición la fracción *B* que está en E B C, e incrementamos una vez su exponente. Este proceso de comparación se repite, hasta que sean iguales los exponentes ó, hasta que se haga cero la cuenta o secuencia, manifestando que la fracción menor, que se estaba alineando para corresponder a la fracción mayor, ya ha perdido todos sus bit significativos. En la etiqueta ALTHB ocurre lo mismo pero con *AR*.

Después de haber alineado correctamente las mantisas, estamos en AEQB y procedemos a preguntar por la operación que se desea realizar. Cargamos en A el registro (SUMRES), le restamos un valor

de uno al acumulador, y preguntamos por la bandera Z. Si Z=1, significa que 01H-01H=0 y que la operación que se desea hacer es una resta, así que vamos a la etiqueta RESTE, en el caso de que Z=0, nos vamos a la etiqueta SUME. Aquí, cargamos los signos de (OP1) y de (OP2), puesto que en el paso en que acomodamos las mantisas y los exponentes, a los signos los quitamos del contexto; una vez instalados los signos, les aplicamos una compuerta XOR, verificamos el bit en cuestión y, si es 1 quiere decir que los signos de los operandos son diferentes y en realidad haremos una resta, así que nos iremos a la etiqueta RESTAR; pero si el bit verificado es 0_r entonces realizaremos una suma real sobre las mantisas alineadas. En la etiqueta SUMAR, sumamos las mantisas que están en E B C y en A H L, inmediatamente después preguntamos por el último acarreo generado; si C=1, entonces recorremos éste bit dentro de la mantisa, incrementando una vez el exponente *a* o *b*, pues ambos contienen el mismo valor; en este momento podemos verificar el sobreflujo en el exponente. Si el exponente pasa del valor numérico 255, llamamos al registro de estados y encendemos el bit 0 = 1, luego lo devolvemos a su dirección y termina la operación. Si el acarreo de la suma real es = 0, entonces nos vamos a normalizar la mantisa a la etiqueta NORMA. Si no hay problemas de sobreflujo en la suma, continuamos colocando, a partir de corrimiento el signo de la suma que es el original del operando mayor; cargamos el resultado que esta en D A H L a (OP1) y, termina la operación. Estando en la etiqueta RESTAR, cargamos los

signos de los operandos, si al hacer el XOR, el bit 0=1, nos vamos a la etiqueta SUMAR, y si es 0, realizaremos una resta real. Restamos las mantisas y verificamos el acarreo, si es C=1 significa que el resultado fue negativo, esto por ser en complemento a dos las restas que realiza el microprocesador; pero la representación de nuestros números son en magnitud con signo, así que nos vamos a COMPLE en donde tendremos que sacar el complemento a dos de ese resultado y complementar el signo del resultado, que sería el del operando mayor; bastaría entonces normalizar el resultado yendo a NORMA. Si el resultado de la resta fue positiva, C=0, sólo hay que verificar si el resultado fue cero, si no fue cero nos vamos a la etiqueta NORMA, si el resultado fue cero cargamos ceros en (OP1) y termina la operación.

```
OP CODE     ETIQUETAS  NEMÓNICOS        COMENTARIOS

3A 76 1B    SUMFLOT    LD A,(1B76H)     ; A ← MSB de BR
17                     RLA              ;
17                     RLA              ;
30 7D                  JR NC,BETAS      ; Si MSB=0 → BR=0
3A 72 1B               LD A,(1B72H)     ; A ← MSB de AR
17                     RLA              ;
17                     RLA              ;
38 28                  JR C, OKSUM      ; Si AR ≠ 0 continuamos.
3A 7C 1B               LD A,(1B7CH)     ; Si AR = 0 preguntamos
```

OP CODE	ETIQUETAS	NEMÓNICOS	COMENTARIOS
D6 01		SUB 01H	; por la operación que
28 0E		JR Z,CPLAS	; se iba a realizar.
2A 74 1B		LD HL,(1B74H)	; (OP1) ← (*BR*)
22 70 1B		LD (1B70H),HL	;
2A 76 1B		LD HL,(1B76H)	;
22 72 1B		LD (1B72H),HL	;
18 61		JR BETAS	;
2A 74 1B	CPLBS	LD HL,(1B74H)	; $B_i \leftarrow \overline{B_i}$
22 70 1B		LD (1B70H),HL	; (OP1) ← *BR*
2A 76 1B		LD HL,(1B76H)	;
CB 14		RL H	;
3F		CCF	;
CB 1C		RR H	;
22 72 1B		LD (1B72H),HL	;
18 4E		JR BETAS	;
ED 4B 74 1B	OKSUM	LD BC,(1B74H)	; DEBC ←(*BR*)
ED 5B 76 1B		LD DE,(1B76H)	;
CB 21		SLA C	; Eliminamos el signo
CB 10		RL B	; de *BR* temporalmente.
CB 13		RL E	;
CB 12		RL D	;
D9		EXX	;*BR* a registros alternos.
2A 70 1B		LD HL,(1B70H)	; DEHL ← (*AR*)
ED 5B 72 1B		LD DE,(1B72H)	;

OP CODE	ETIQUETAS	NEMÓNICOS	COMENTARIOS
CB 25		SLA L	; Eliminamos el signo
CB 14		RL H	; de *AR* temporalmente.
CB 13		RL E	;
CB 12		RL D	;
7D		LD A,L	; Acomodo final.
D9		EXX	;
6F		LD L,A	;
D9		EXX	;
7C		LD A,H	;
D9		EXX	;
67		LD H,A	;
D9		EXX	;
7B		LD A,E	;
18		LD B,18H	; Máximo número de
D9	COMPA	EXX	; comparaciones = 24.
08		EX AF,AF'	; Acomodamos para
D9		EXX	; hacer la resta
7A		LD A,D	; de comparación.
D9		EXX	;
92		SUB D	; Resta de comparación.
38 1D		JR C,ALTHBS	; Si C=1 *a < b*
28 38		JR Z,AEQBS	; Si Z=1 *a = b*
14		INC D	; *a > b, b←b+1*
CB 3B		SRL E	; Shr *B*

```
OP CODE     ETIQUETAS   NEMÓNICOS        COMENTARIOS
CB 18                   RR B             ;
CB 19                   RR C             ;
08                      EX AF,AF'        ;
D9                      EXX              ;
10 EB                   DJNZ COMPA       ; Vuelve a comparar y
                                         ; decrementa cuenta.
                                         ; BR insignificante
                                         ; respecto a AR, termina
                                         ; operación (OP1) ← AR.
18 50       BETAS       JR BETA          ; Puente. BR=0
D9          ALTHBS      EXX              ;
14                      INC D            ; a ← a+1
D9                      EXX              ;
08                      EX AF,AF'        ;
CB 3F                   SRL A            ; Shr A
CB 1C                   RR H             ;
CB 1D                   RR L             ;
D9                      EXX              ;
10 CE                   DJNZ COMPA       ; Vuelve a comparar
                                         ; y decrementa cuenta
ED 4B 74 1B             LD BC,(1B74H)    ; AR insignificante
2A 76 1B                LD HL,(1B76H)    ; respecto a BR, termina
ED 43 70 1B             LD (1B70H),BC    ; operación (OP1) ← BR.
22 72 1B                LD (1B72H),HL    ;
```

OP CODE	ETIQUETAS	NEMÓNICOS	COMENTARIOS
18 E1		JR BETAS	; Puente.
3A 7C 1B	AEQBS	LD A,(1B7CH)	; *a = b*, indentificar
D6 01		SUB 01H	; operación a realizar.
28 2E		JR Z,RESTE	; Si Z=1 ir a RESTE.
3A 73 1B	SUME	LD A,(1B73H)	; Si Z=0 SUME,identificar
D9		EXX	; el signo de *AR*,
4F		LD C,A	; guardamos temporalmente
			; el signo de A.
3A 77 1B		LD A,(1B77H)	;Identificar signo de *BR*.
A9		XOR C	; *A, XOR B,*
CB 7F		BIT 7,A	;
20 2E		JR NZ,RESTAR	; Diferentes RESTAR.
D9	SUMAR	EXX	; Signos iguales SUMAR.
08		EX AF,AF'	;
09		ADD HL,BC	; Suma de fracciones.
8B		ADC A,E	;
30 5D		JR NC,NORS	; C=0 ir a normalizar
			; la fracción.
1F		RRA	; Normalizamos el C=1
CB 1C		RR H	;
CB 1D		RR L	;
14		INC D	; Incrementar exponente.
18		JR NC, ALFA	; Verificamos sobreflujo.
08		EX AF,AF'	; Si hubo sobreflujo

```
OP CODE     ETIQUETAS   NEMÓNICOS        COMENTARIOS
3A 7B 1B                LD A,(1B7BH)     ; traemos ESTADOS.
CB C7                   SET 0,1          ; Encendemos OVFSUM.
32 7B 1B                LD (1B7BH),A     ; Regresamos ESTADOS.
08                      EX AF,AF'        ; Restauramos acumulador.
D9              ALFA    EXX              ; Traer el signo de A y,
CB 21                   SLA C            ; lo metemos al acarreo.
D9                      EXX              ;
CB 1A                   RR D             ; Hacemos corrimiento
1F                      RRA              ; a partir de carry para
CB 1C                   RR H             ; formar  resultado final
CB 1D                   RR L             ; en DEHL.
5F                      LD E,A           ;
ED 53 72 1B             LD (1B72H),DE    ; (OP1) ← DEHL
22 70 1B                LD (1B70H),HL    ;
18 52         BETA      JR FINS          ; Termina la operación.
3A 73 1B      RESTE     LD A,(1B73H)     ; Identificar signo de A.
D9                      EXX              ;
4F                      LD C,A           ; se guarda.
3A 77 1B                LD A,(1B77H)     ; Identificar signo de B.
A9                      XOR C            ; A, XOR B,
CB 7F                   BIT 7,A          ;
20 D2                   JR NZ,SUMAR      ;Signos diferentes SUMAR.
D9            RESTAR    EXX              ; Signos iguales RESTAR.
08                      EX AF,AF'        ;
```

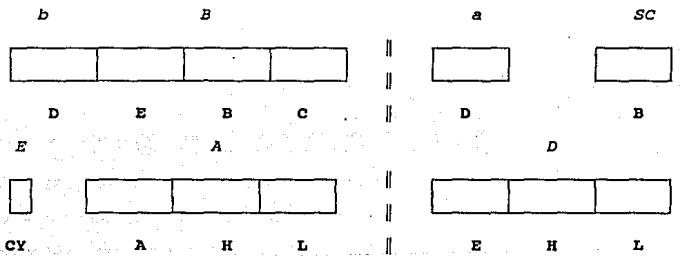
```
OP CODE     ETIQUETAS  NEMÓNICOS        COMENTARIOS
37                     SCF              ; Limpiamos el acarreo
3F                     CCF              ; C=0.
ED 42                  SBC HL,BC        ; Restamos fracciones.
9B                     SBC A,E          ;
38 15                  JR C,COMPLE      ; Si C=1, resultado
                                        ; negativo en complemento
                                        ; a dos.
01 00 00               LD BC,0000H      ; Limpiamos registros
1E 00                  LD E,00H         ; para verificar un
                                        ; resultado = 0.
09                     ADD HL,BC        ; Restamos 0-resultado
20 24                  JR NZ,NORS       ; Si esta parte = 0
                                        ; normalizamos.
8B                     ADC A,E          ; Restamos la otra parte.
20 21                  JR NZ,NORS       ; Si no es cero,
                                        ; normalizamos.
ED 43 70 1B            LD (1B70H),BC  ; Si toda la resta es cero
ED 43 72 1B            LD (1B72H),BC  ; (OP1) ← 0.
18 27                  JR FINS          ; Termina la operación.
08          COMPLE     EX AF,AF'      ; Si la resta fue negativa
7D                     LD A,L           ; complementamos
2F                     CPL              ; resultado.
6F                     LD L,A           ;
7C                     LD A,H           ;
```

```
OP CODE     ETIQUETAS   NEMÓNICOS        COMENTARIOS
2F                      CPL              ;
67                      LD H,A           ;
08                      EX AF,AF'        ;
2F                      CPL              ;
01 01 00                LD BC,0001H      ; Sumamos un 1, por ser
1E 00                   LD E,00H         ; complemento a dos.
09                      ADD HL,BC        ;
8B                      ADC A,E          ;
D9                      EXX              ;
08                      EX AF,AF'        ; Traemos el signo
79                      LD A,C           ; de A y lo negamos.
2F                      CPL              ;
4F                      LD C,A           ; Lo cargamos en C.
D9                      EXX              ; a alternos.
08                      EX AF,AF'        ; la parte MSB de A
17          NORS        RLA A            ; Normalizamos fracción.
38 0A                   JR C,NONORS      ;
1F                      RRA              ;
CB 25                   SLA L            ; Shl AHL.
CB 14                   RL H             ;
CB 17                   RL A             ;
15                      DEC D            ; Decrementar exponente.
18 F3                   JR NORS          ; Preguntamos otra vez.
1F          NONORS      RRA              ; La fracción está norma.
```

OP CODE	ETIQUETAS	NEMÓNICOS	COMENTARIOS
18 99		JR ALFA	; Vamos a ALFA a formar
C9	FINS	RET	; resultado final.

4.2 MULTF.

La siguiente es la explicación de la realización de la rutina de multiplicación de punto flotante en lenguaje emsamblador. La distribución de los parámetros del algoritmo en los registros del microprocesador Z80 es como se muestra a continuación:



Como ya hemos explicado, los registros (OP1) y (OP2) contienen los operandos multiplicando y multiplicador respectivamente. Siguiendo los pasos para este algoritmo del capítulo anterior. Primero verificamos si el multiplicador es cero, para ello cargamos el MSB de la fracción, si es 1 continuamos si es 0 cargamos como resultado en (OP1) un 0. Después si (OP1) no es 0, preguntamos por

(OP2) que es el multiplicando y, hacemos lo mismo que con (OP1). Si ninguno de los operandos fue cero, los instalamos en los registros *BR, D* y *a*, despreciando ambos signos eventualmente; al registro *A*, que tendrá a la fracción producto, lo limpiamos, cargamos la cuenta *SC* en B = 18H = 24 y procedemos a multiplicar las fracciones. Preguntamos por el LSB del multiplicador *D*, si es 1 sumamos *A←A+B* y recorremos a la derecha *EAD*, si fue 0 sólo recorremos sin sumar; decrementamos la cuenta y volvemos a preguntar por el LSB del multiplicando hasta que *SC* = 0. Al final la fracción producto estará en *AD*, y la parte más significativa en *A*. Después sumamos los exponentes *a←a+b*, restamos una vez el sesgo y preguntamos si hay sobreflujo, si hubo uno, traemos el registro ESTADOS y encendemos el bit OVFMULTF; si no hubo, procedemos a normalizar la fracción. En la normalización buscaremos un 1 a lo largo de *AD* y en cada corrimiento a la izquierda decrementamos una vez el exponente. Una vez normalizado el resultado, traemos de (OP1) y de (OP2) los signos de los operandos y les aplicamos un A_s XOR B_s que será el signo final del resultado; acomodamos en DEHL y cargamos estos en (OP1), y así termina MULTF.

OP CODE	ETIQUETAS	NEMÓNICOS	COMENTARIOS
3A 76 1B	MULTF	LD A,(1B76H)	; Verificamos multiplicador
17		RLA	; cero.
17		RLA	;

```
OP CODE     ETIQUETAS  NEMÓNICOS      COMENTARIOS
38 0B                  JR C,TWM       ; Si no es 0 continuamos.
21 00 00    ALFA       LD HL,0000H    ; Si es 0, cargamos 0 en
22 70 1B               LD (1B70H),HL; (OP1) y termina MULTF.
22 72 1B               LD (1B72H),HL;
18 6B                  JR ALFAM       ; Puente.
3A 72 1B    TWM        LD A,(1B72H)   ; Verificamos multiplicando
17                     RLA            ; cero.
17                     RLA            ;
38 02                  JR C,OKMULTF   ; Si no es 0 continuamos.
18 80                  JR ALFA        ; Si es 0, cargamos 0 en
                                      ; (OP1) y termina MULTF.
ED 4B 74 1B OKMULTF    LD BC,(1B74H); Si los operandos = 0
ED 5B 76 1B            LD DE,(1B76H); cargamos multiplicando
CB 21                  SLA C          ; en DEBC, despreciamos
CB 10                  RL B           ; el signo eventualmente.
CB 13                  RL E           ;
CB 12                  RL D           ;
21 00 00               LD HL,0000H    ; Limpiamos registro A.
3E 00                  LD A,00H       ;
D9                     EXX            ; Alternamos registros.
2A 70 1B               LD HL,(1B70H); Cargamos multiplicador
ED 5B 72 1B            LD DE,(1B72H); en DEHL, despreciamos
CB 25                  SLA L          ; el signo eventualmente.
CB 14                  RL H           ;
```

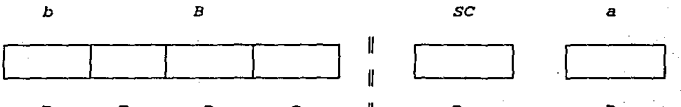
```
OP CODE     ETIQUETAS   NEMÓNICOS      COMENTARIOS
CB 13                   RL E           ;
CB 12                   RL D           ;
06 18                   LD B,18H       ; Cargamos cuenta 18H=24.
CB 0D       LOOPM       RRC L          ; Multiplicamos fracciones.
CB 15                   RL L           ; preguntar por el LSB del
D9                      EXX            ; multiplicando.
30 02                   JR NC,NOADD    ; Si LSB de D =0 NOADD.
09                      ADD HL,BC      ; Si LSB de D =1 sumamos
8B                      ADC A,E        ; EA ← A + B
CB 1F       NOADD       RR A           ; Shr E A D
CB 1C                   RR H           ;
CB 1D                   RR L           ;
D9                      EXX            ;
CB 1B                   RR E           ;
CB 1C                   RR H           ;
CB 1D                   RR L           ;
10 E8                   DJNZ LOOPM     ; Repetimos hasta SC=0.
08                      EX AF,AF'      ; Ubicamos exponentes para
7A                      LD A,D         ; su adición.
D9                      EXX            ;
82                      ADD A,D        ; Sumamos exponentes.
C6 80                   ADD A,80H      ; Restamos una vez el sesgo.
57                      LD D,A         ; Guardamos exponente.
30 08                   JR NC,QWE      ; No sobreflujo continuamos.
```

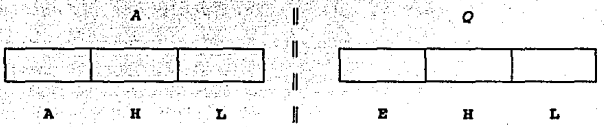
OP CODE	ETIQUETAS	NEMÓNICOS	COMENTARIOS
3A 7B 1B		LD A,(1B7BH)	; Si sobreflujo, ESTADOS.
CB CF		SET 1,A	; Encendemos OVFMULT.
32 7B 1B		LD (1B7BH),A	; Regresamos ESTADOS.
08	QWE	EX AF,AF'	; Continuamos.
D9		EXX	; Alternamos.
06 18		LD B,18H	; Cuenţa para normalizar.
D9	NORM	EXX	;
17		RLA	; Preguntamos por MSB.
38 16		JR C,NONORM	; Si MSB=1 no normalizamos.
1F		RRA	; Regresamos el MSB y
D9		EXX	; Comenzamos a normalizar.
CB 25		SLA L	; Recorremos *Shl EAD.*
CB 14		RL H	; *Shl E A D*
CB 13		RL E	;
D9		EXX	;
CB 15		RL L	;
CB 14		RL H	;
17		RLA	;
15		DEC D	; Decrementamos exponente.
D9		EXX	;
10 EA		DJNZ NORM	; Volvemos a preguntar.
18 16		JR CEROM	; 24 vececs y ningún bit=1
18 1B	ALFAM	JR FINM	; Puente.
1F	NONORM	RRA	; Fracción normalizada.

```
OP CODE     ETIQUETAS  NEMÓNICOS      COMENTARIOS
5F                     LD E,A         ; Reubicamos Parte MSB de
                                      ; fracción.
3A 73 1B               LD A,(1B73H) ; Traemos signo del
47                     LD B,A         ; multiplicando.
3A 77 1B               LD A,(1B77H) ; Signo del multiplicador.
A8                     XOR B          ; Los operamos.
CB 27                  SLA A          ; Acomodamos signo,
CB 1A                  RR D           ; exp. y fracción final.
CB 1B                  RR E           ;
CB 1C                  RR H           ;
CB 1D                  RR L           ;
22 70 1B    CEROM      LD (1B70H),HL; Cargamos producto final
ED 53 72 1B            LD (1B72H),DE; en (OP1).
C9          FINM       RET            ; Termina  MULTF.
```

4.3 DIVF

La siguiente explicación es la que corresponde al programa DIVF, que es la división en punto flotante. La distribución de los parámetros en los registros de la CPU Z80 es la siguiente.





El primer paso es verificar ceros, así que cargamos el MSB del divisor, que está en (OP2). si es 0, encendemos el bit de *DV0* (división entre cero) y termina DIVF. Si no es 0 el divisor, verificamos al dividendo, si es 0, cargamos ceros en (OP1) y termina DIVF. Si ninguno de los operandos fue 0, vamos a OKDIVF. Entonces cargamos y acomodamos los operandos según la distribución propuesta. Inicialmente el parámetro *Q* (cociente) estará en ceros. El paso siguiente, es el alineamiento del dividendo, para ello realizamos una resta $EA \leftarrow A-B$. Si el acarreo $E=1$ significa que la parte más significativa de $A<B$ como se requiere y vamos a ALTHB, si $E=0$ entonces $A \geq B$ y habrá que alinear al dividendo, recorremos una vez su fracción a la derecha AHL e incrementamos una vez su exponente. Una vez alineadas las fracciones, restamos los exponentes y sumamos una vez el sesgo, verificamos sobreflujo y subflujo. Si existe sobreflujo se enciende el bit de ESTADOS OVFDIVF, si fue subflujo se supondrá 0 el resultado. Si no hubo problemas se procede a dividir las fracciones, realizamos un primer corrimiento a la izquierda del dividendo *EAQ* para colocarlo en posición con el divisor, si después del corrimiento $E=1$ significa que $EA>B$ y entra un 1 en Q_0, si el $E=0$ restamos para comparar y

verificamos el acarreo, si E=0 quiere decir que $A \geq B$ entonces entra un 1 y se repite la secuencia LOOPD colocando en posición al siguiente bit menos significativo del dividendo con el divisor, si el acarreo en la resta fue 0, significa que $A<B$ Q_0 permanece en 0, y restauramos el valor del dividendo para volver a colocar en posición al dividendo con el divisor, repitiéndose la secuencia 24 veces. Al final tendremos en Q la fracción cociente de la división. Ésta fracción está correctamente normalizada, así que no hará falta verificar ello. Ahora, bastará designar el signo del cociente. Cargamos los signo de (OP1) y de (OP2) los ubicamos y los operamos $Q_s = A_s$ XOR B_s. Acomodamos el signo, el exponente y la fracción, en los registros DEHL, y los cargamos en (OP1). Termina así DIVF.

```
OP CODE     ETIQUETAS   NEMÓNICOS        COMENTARIOS

3A 76 1B    DIVF        LD A,(1B76H) ; Verificamos divisor 0.
17                      RLA          ;
17                      RLA          ;
38 02                   JR C,TWD     ; Si no es 0, continuamos.
3A 7B 1B                LD A,(1B7BH) ; Si es 0 traer ESTADOS
CB DF                   SET 3,A      ; Encendemos bit DV0.
32 7B 1B                LD (1B7BH),A ; Regresamos ESTADOS.
18 78                   JR ALFAD     ; Termina DIVF.
3A 72 1B    TWD         LD A,(1B72H) ; Verificamos dividendo 0.
```

```
OP CODE     ETIQUETAS  NEMÓNICOS          COMENTARIOS
17                     RLA           ;
17                     RLA           ;
38 05                  JR C,OKDIVF   ; Si no es 0, continuamos.
18 72                  JR CEROD      ; Si es 0, vamos a CERO.
2A 70 1B    OKDIVF     LD HL,(1B70H); Cargamos (OP1) en registro
ED 5B 72 1B            LD DE,(1B72H); dividendo.
CB 25                  SLA L         ; Despreciamos
CB 14                  RL H          ; eventualmente el signo.
CB 13                  RL E          ;
CB 12                  RL D          ;
D9                     EXX           ; Alternamos.
2A 74 1B               LD HL,(1B74H); Cargamos (OP2) en registro
ED 5B 76 1B            LD DE,(1B76H); divisor.
CB 25                  SLA L         ; Despreciamos
CB 14                  RL H          ; eventualmente el signo.
CB 13                  RL E          ;
CB 12                  RL D          ;
D9                     EXX           ; Alternamos.
7B                     LD A,E        ; Acomodamos según
08                     EX AF,AF'     ; distribución propuesta.
D9                     EXX           ;
7B                     LD A,E        ;
D9                     EXX           ;
5F                     LD E,A        ;
```

```
OP CODE     ETIQUETAS   NEMÓNICOS       COMENTARIOS
D9                      EXX             ; Distribuir
7C                      LD A,H          ; parámetros en la
D9                      EXX             ; CPU.
47                      LD B,A          ;
D9                      EXX             ;
7D                      LD A,L          ;
D9                      EXX             ;
4F                      LD C,A          ;
08                      EX AF,AF'       ;
D9                      EXX             ; Alternamos.
21 00 00                LD HL,0000H     ; Cargamos inicialmente
1E 00                   LD E,00H        ; a Q con 0.
06 18                   LD B,18H        ; Cargamos cuenta=18H=24
D9                      EXX             ; Alternamos
37                      SCF             ; Limpiamos el
3F                      CCF             ; acarreo.
ED 42                   SBC HL,BC       ; Alineamos el
9B                      SBC A,E         ; dividendo.
38 0B                   JR C,ALTHBD     ; Si C=1 A<B ir a ALTHBD
09                      ADD HL,BC       ; Si C=0 A≥B se restaura
8B                      ADC A,E         ; el dividendo.
CB 3F                   SRL A           ; Recorremos el
CB 1C                   RR H            ; dividendo a la
CB 1D                   RR L            ; derecha.
```

OP CODE	ETIQUETAS	NEMÓNICOS	COMENTARIOS
14		INC D	; incrementamos exp.
18 02		JR EXPD	; Puente.
09	ALTHBD	ADD HL,BC	; Restauramos el
8B		ADC A,E	; dividendo.
08	EXPD	EX AF,AF'	; Continuamos.
7A		LD A,D	; Ubicamos exponentes
D9		EXX	; alternamos
92		SUB D	; Restamos exponentes
C6 80		ADD A,80H	; sumamos sesgo.
57		LD D,A	; Cargamos exp. en D.
08		EX AF,AF'	; Alternamos.
CB 25	LOOPD	SLA L	; Dividir fracciones.
CB 14		RL H	; *Shl EAQ* para colocar
CB 13		RL E	; en posición al
D9		EXX	; dividendo con el
CB 15		RL L	; divisor.
CB 14		RL H	;
CB 17		RL A	;
38 0E		JR C,CYEQ1	; C=1 $A>B$,Q_0=1,CYEQ1
ED 42	CYEQ0	SBC HL,BC	; C=0 comparamos,
9B		SBC A,E	; $EA-A-B$.
30 0E		JR NC,CYEAEQ1	; Si C=0 $A>B$
09		ADD HL,BC	; Restauramos el
8B		ADC A,E	; dividendo. Continua.

```
OP CODE     ETIQUETAS   NEMÓNICOS          COMENTARIOS
D9                      EXX            ;
10 E7                   DJNZ LOOPD     ; Repetir secuencia.
18 0C                   JR BGT         ; Continuamos.
18 47       ALFAD       JR FIND        ; Puente.
37          CYEQ1       SCF            ; Si C=1, limpiamos C.
3F                      CCF            ;
ED 42                   SBC HL,BC      ; Restamos para
9B                      SBC A,E        ; comparar.
D9          CYEAEQ1     EXX            ;
CB C5                   SET 0,L        ;
10 D9                   DJNZ LOOPD     ;
3A 73 1B      BGT       LD A,(1B73H) ; Traemos al signo del
4F                      LD C,A         ; dividendo.
3A 77 1B                LD A,(1B77H) ; Signo del divisor.
A9                      XOR C          ; Le aplicamos XOR
CB 27                   SLA A          ; Formamos el resultado
CB 1A                   RR D           ; con el signo
CB 1B                   RR E           ; exponente
CB 1C                   RR H           ; y fracción.
CB 1D                   RR L           ;
22 70 1B                LD (1B70H),HL; Cargamos resultado
ED 53 72 1B             LD (1B72H),DE; en (OP1).
18 0D                   JR FIND        ; Termina DIVF.
01 00 00    CEROD       LD BC,0000H  ; Resultado cero.
```

```
OP CODE     ETIQUETAS   NEMÓNICOS       COMENTARIOS
21 00 00                LD HL,0000H   ;
ED 43 70 1B             LD (1B70H),BC;
22 72 1B                LD (1B72H),HL;
C9            FIND      RET           ; Termina DIVF.
```

CAPITULO V

APLICACIONES

DE LA UNIDAD ARITMÉTICA

La aplicación que aquí se presenta, no quiere ser más que una extensión de la misma unidad aritmética, así pues, nuestra aplicación es una aplicación inmediata; quiere decir esto que, contando ya con una unidad aritmética que realiza las operaciones de suma, resta, multiplicación y división en punto flotante, nos proponemos utilizar estas operaciones para contribuir más al enriquecimiento de la unidad aritmética. Basándonos en ésta posibilidad que ya tenemos para realizar operaciones, podemos

definir ciertos algoritmos matemáticos que contribuyan a nuestro objetivo. Por lo tanto, la aplicación que proponemos es la generación de una de las funciones trigonométricas más útiles en las matemáticas: la función SEN(X). con la realización de tal función nos daremos cuenta clara de las capacidades de nuestra unidad aritmética en punto flotante.

Para darnos una idea de los que vamos a hacer y de lo que vamos a requerir para tal objetivo, observemos y analicemos la serie de McClaurin con la cual se aproxima tal función.

$$SEN(x) = x - x^3/3! + x^5/5! - x^7/7! + x^9/9! - x^{11}/11! \ldots\ldots \quad (5.1)$$

Lo primero es definir a x en radianes. Ahora, sabemos que la convergencia exacta se alcanza con un número infinito de elementos de la serie; obviamente esto es imposible de alcanzar, hasta por la computadora más poderosa que pudiera existir, así que sólo buscaremos una buena aproximación, y con seis elementos, como los que tiene la ecuación 5.1, será suficiente.

Necesitaremos generar las potencias de x^2 hasta x^{11} y, tener en memoria, disponible a los factoriales 3!, 5!, 7!, 9! y 11! ; combinar estos elementos y por último sumarlos o restarlos según la ecuación 5.1.

Por lo tanto, habrá que reservar un espacio de memoria RAM para las potencias de *x* y, un espacio de memoria EPROM para los factoriales que son datos fijos. El mapa de memoria que diseñamos para ello se muestra en la figura 5.2 (a) y (b) .

Una vez que ya hemos pensado en los espacios de memoria, hay que especificar una cosa; en realidad el valor de *x* puede estar en

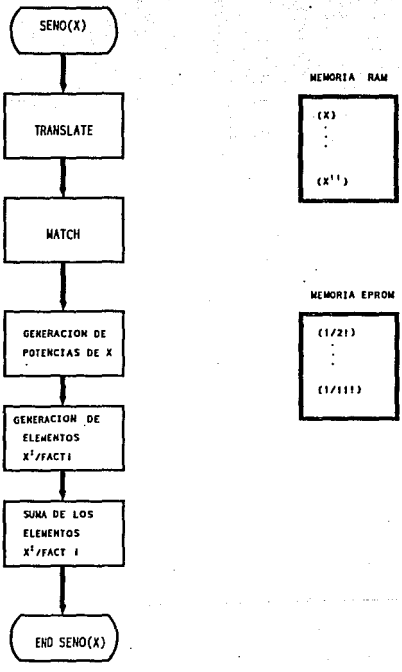


FIGURA 5.1 DIAGRAMA DE BLOQUES PARA LA GENERACION DE LA FUNCION SENO(X).

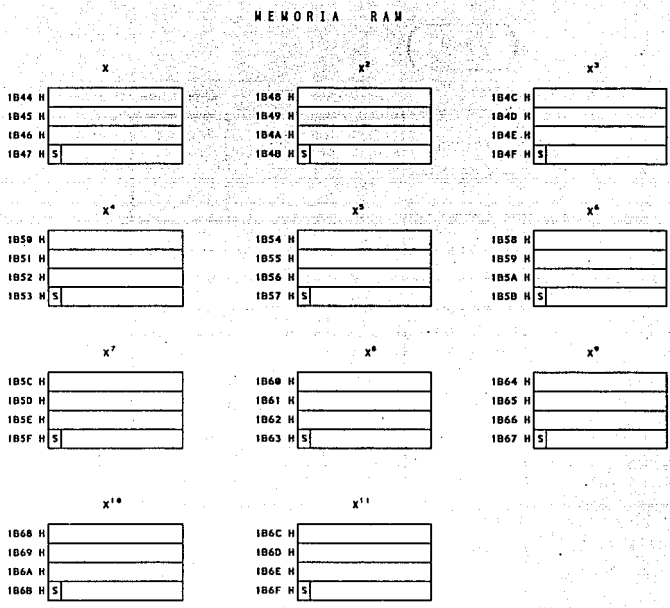


FIGURA 5.2a MAPA DE MEMORIA DE LOS REGISTROS DE DATOS Y DE RESULTADOS PARCIALES PARA LA GENERACION DE LA FUNCION SENO(X) EN EL SISTEMA SIMMP1.

MEMORIA EPROM

1/2!

17DC H		0 0
17DD H		0 0
17DE H		4 0
17DF H	S	4 0

1/3!

17E0 H		5 7
17E1 H		5 5
17E2 H		5 5
17E3 H	S	3 F

1/4!

17E4 H		5 7
17E5 H		5 5
17E6 H		5 5
17E7 H	S	3 E

1/5!

17E8 H		4 5
17E9 H		4 4
17EA H		4 4
17EB H	S	3 D

1/6!

17EC H		D 0
17ED H		0 5
17EE H		D B
17EF H	S	3 B

1/7!

17F0 H		8 0
17F1 H		0 6
17F2 H		6 8
17F3 H	S	3 A

1/8!

17F4 H		8 0
17F5 H		0 6
17F6 H		6 8
17F7 H	S	3 A

1/9!

17F8 H		9 0
17F9 H		7 7
17FA H		5 C
17FB H	S	3 7

1/10!

17FC H		4 0
17FD H		F 9
17FE H		C 9
17FF H	S	3 5

1/11!

1800 H		1 8
1801 H		9 9
1802 H		E B
1803 H	S	3 3

FIGURA 5.25 MAPA DE MEMORIA DE LOS REGISTROS DE DATOS INVERSOS DE FACTORIAL PARA LA GENERACION DE LA FUNCION SENO(X) EN EL SISTEMA SIMMPI.

un rango de $-\infty$ a $+\infty$ aunque, para nuestro sistema tales números serían los más grandes (+/-) que pueden ser representados en nuestro formato de punto flotante. Además, sabemos ahora por experiencia que, la ecuación 5.1, converge adecuadamente, arrojando resultados que se aproximan a los reales, sólo en el rango de $[0,\pi/2]$; así que tendremos, puesto que queremos que nuestro rango de x sea el mayor posible, trasladar y adecuar a x cuando sea necesario, a tal rango.

Un diagrama de bloques que generaliza nuestra aplicación, se muestra en la figura 5.1.

5.1. RUTINA TRANSLATE.

El objetivo de la rutina TRANSLATE, es el trasladar valores de x [rad], que vayan más allá de 2π precisamente al rango $[0,2\pi]$.

Según una apreciación trigonométrica; si tenemos un círculo dividido en radianes, un número x se mueve por el perimetro $[0,2\pi]$. pero si x es mayor a 2π, este sigue en el perimetro y, existe un valor equivalente para él, pero menor a 2π.

Podemos encontrar ese valor equivalente, si dividimos a x, que es mayor a 2π, entre 2π, resultando un valor que es mayor a 2π. Este valor representa el número de veces que es mayor x que 2π. Si le quitamos la parte entera, la fracción que queda es el equivalente del valor original, pero que está entre 0 y 1; sólo basta multiplicar por 2π para que nos resulte el valor equivalente entre $[0,2\pi]$. En ecuaciones el procedimiento de traslado sería:

x [rad] $(-\infty,+\infty)$

$$\frac{x}{2\pi} = N.n \quad ; N.n = \text{\# de veces que } x \text{ es mayor que } 2\pi.$$

$$.n \leftarrow (N.n) - N$$

$$x[0,2\pi] \leftarrow .n * 2\pi$$

El programa en emsamblador sigue los pasos que se muestran en la figura 5.3.

TRANSLATE comienza forzando a x a ser positivo, esto con el fin de suponer números positivos y facilitar las cosas, al salir de TRANSLATE se le devolverá su signo original. Después se realiza una comparación en punto flotante, para ello carga en OP2 el valor de x que está en la dirección (1B44H) y, en OP1 carga un 2π, después hace una resta, cargando en el byte de operación (1B7CH) un 01H y llamando a SUMFLOT. Se verifica entonces el signo del resultado y, si es positivo significa que $x \leq 2\pi$ y el dato x no necesita ser operado y ya puede entonces irse a la rutina MATCHX; pero si el signo es negativo, entonces significa que $x > 2\pi$, en este caso, cargamos en OP1 a x y en OP2 a $1/2\pi$, llamamos entonces a MULTF. Al retornar de la operación al resultado le quitaremos su parte entera, para ello ubicamos el punto binario en la mantisa, esto nos lo dice el exponente al quitarle el sesgo; así, recorremos la mantisa a la izquierda tantas posiciones como unidades tenga el exponente. El nuevo número tendrá exponente igual a cero. La mantisa que fue recorrida, puede que no este correctamente

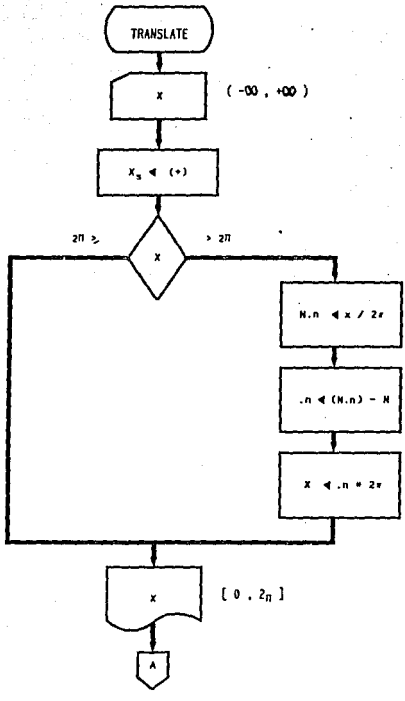


FIGURA 5.3 DIAGRAMA DE FLUJO DE LA RUTINA TRANSLATE, QUE TRASLADA UN VALOR DE X, DEL RANGO (-∞ , +∞) A OTRO DE [0 , 2π].

normalizada, así que se verifica su MSB, si es cero entonces normalizamos; cuando el número está normalizado, se sesga su exponente y se le asigna su signo original. Ahora sólo habrá que multiplicar este dato por 2π, en OP1 estará el primero y, en OP2 el segundo respectivamente, llamamos a MULTF y, el producto resultante estará en el rango requerido $[0,2\pi]$, lo mandamos al registro (x), y saltamos a MATCHX.

5.2. RUTINA MATCHX.

Ahora tenemos un valor x dentro del rango $[0,2\pi]$ y, tenemos que forzarlo, si es necesario, a que entre en el rango $[0,\pi/2]$, pues como lo hemos advertido, es éste el rango en donde nuestra ecuación del sen(x) puede converger. El objetivo pues, es generar un programa en emsamblador que logre éste propósito. El diagrama de flujo de la figura 5.4 en que se hizo posible esto.

Esta rutina MATCHX se basa en comparar y opera un valor x para encontrar uno equivalente dentro del rango $[0,\pi/2]$.

El procedimiento comienza realizando una primera comparación, para esto se hace una resta en punto flotante de $\pi/2 - x$, así que cargamos en OP1 a $\pi/2$ y, en OP2 a x, después cargamos un 01H en el byte de operación (1B7CH) y llamamos a SUMFLOT. Al retornar de la subrutina, el resultado está en OP1, si este resultado es positivo, significa que $x \leq \pi/2$ y podemos ir a calcular la función senx, cargando antes el resultado en (x). Pero si el resultado es negativo significa que $x > \pi/2$ y habrá entonces que hacer una segunda comparación, para ello realizamos la resta $\pi - x$, cargando

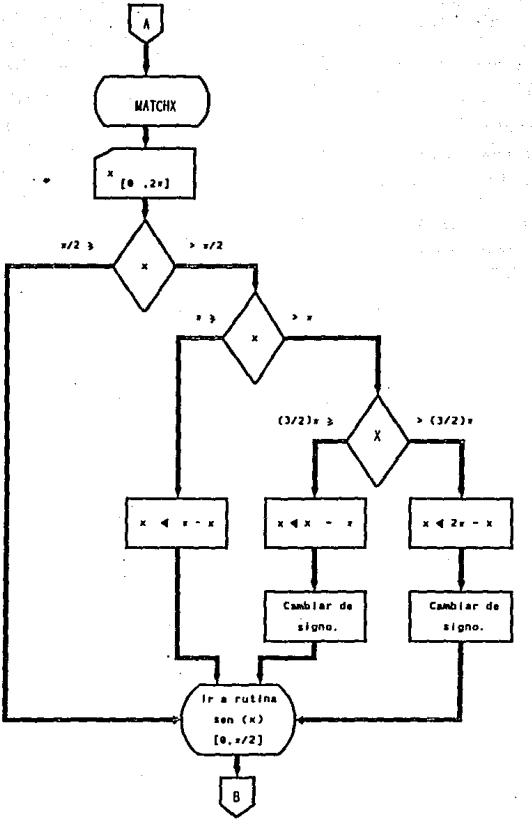


FIGURA 5.4 RUTINA MATCH, ADECUA EL RANGO [0 ,2π] AL RANGO [0, π /2] PARA GENERAR LA FUNCION SENO(X).

en OP1 el valor π, en OP2 continua el valor de x, llamamos a SUMFLOT, (en el byte de operación ya está un 01H de restar) y de regreso de la subrutina verificamos el signo, si es positivo significa que $x \leq \pi$ entonces realizamos la resta $x \leftarrow \pi - x$ y así x estará en el rango $[0,\pi/2]$ siendo suficiente para ir a calcular el senx; pero si el resultado es negativo significa que $x > \pi$ y habrá que realizar una última comparación, de esta manera, realizamos la siguiente resta $(3/2)\pi - x$, cargamos en OP1 $(3/2)\pi$ y llamamos a SUMFLOT, de regreso de la subrutina verificamos el signo, si es positivo realizamos la resta $x \leftarrow x - \pi$, y le cambiamos el signo a x para que el resultado sea negativo, puesto que sen(-x) = -sen(x); si el resultado es negativo, realizamos la resta $x \leftarrow 2\pi - x$ y le cambiamos el signo a x, para ambos casos x estará dentro del rango $[0,\pi/2]$, entonces ya podemos ir a calcular la función sen(x).

5.3. RUTINA SEN(X).

Una vez que el valor de x está dentro del rango $[0,\pi/2]$ ya es posible generar la función sen(x). El diagrama de flujo de la figura 5.5, muestra los pasos generales que se siguieron para nuestro propósito. Primero la generación de las potencias de x; de x^2 a x^{11} y su respectivo archivado en memoria. Después la generación de los elementos $x^i/i!$ de la ecuación 5.1, y por último la suma o resta de estos elementos según la misma ecuación.

Para el primer paso, podemos observar en la página 103 el mapa de memoria donde iremos guardando temporalmente las potencias de x. Cargamos el índice para que señale la primera dirección de éstas

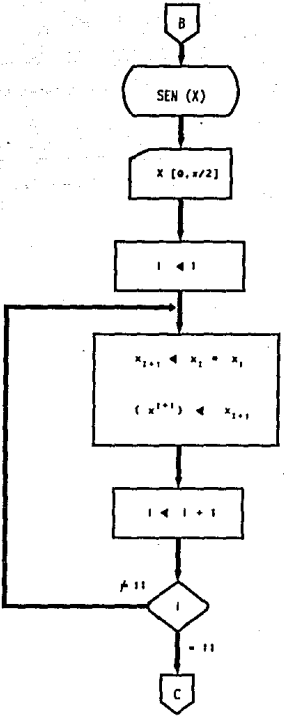


FIGURA 5.5a GENERACION DE POTENCIAS DE X.

(1B48H). Cargamos una cuenta de 11, puesto que según la ecuación 5.1, existe x^{11}, en la dirección (1B7FH); después cargamos en OP1 y en OP2 a *x* y llamamos a MULTF que generará a x^2 y lo guardamos en su dirección específica; una vez hecho esto incrementamos el índice para que apunte a (x^3), en OP1 tenemos a x^2 y en OP1 continua x, llamamos a MULTF y repetimos esto hasta generar x^{11}.

Ya contando con todas las potencias necesarias, procedemos a generar los elementos $x^i/i!$. El mapa de memoria de los inverso factorial de 1/2! hasta 1/11! lo podemos observar en la figura 5.2b. Aunque en la generación del sen(x) sólo se utilizan cinco de estos números, los otros están para su posible utilización en la generación de alguna otra función trigonométrica o logarítmica. Los índices en este paso son IX ← 1B4CH y IY ← 1B20H, uno apunta a (x^3) y el otro a (1/3!) respectivamente. Se carga una cuenta para realizar cinco multiplicaciones de las potencias y de los inversos de factorial respectivos, se multiplica la primera pareja y el producto resultante se guarda en (x^3). De esta forma se van llamando a las parejas respectivas x^i y 1/i! , se apunta hacia sus direcciones, se multiplican y, el resultado se guarda en (x^i). Al final de cinco multiplicaciones tendremos en (x^3), (x^5), (x^7), (x^9) y en (x^{11}) a $x^3/3!$, $x^5/5!$, $x^7/7!$, $x^9/9!$ y $x^{11}/11!$ respectivamente.

El siguiente y último paso, será sumar o restar los elementos $x^i/i!$ que ya tenemos generados. Para iniciar, cargamos al byte de operación (1B7CH) con 00H (sumar), cargamos una cuenta a cinco, colocamos en (OP1) a x, y cargamos a IX con la dirección del primer dato $x^i/i!$ a llamar, éste es $x^3/3!$ y está en la dirección (1B4CH);

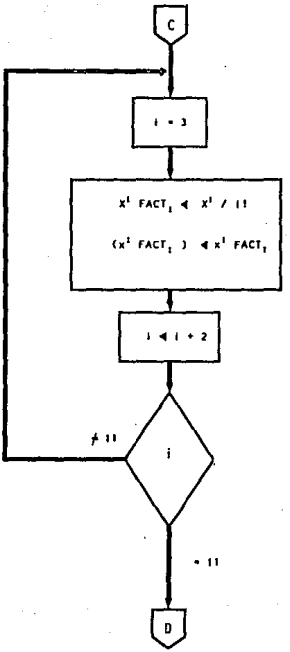


FIGURA 5.5b GENERACION DE LOS ELEMENTOS X^i/ 11

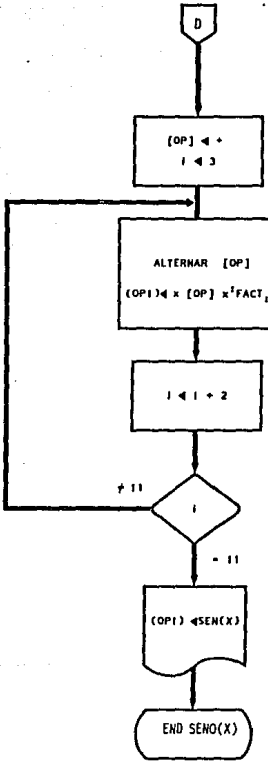


FIGURA 5.5c SUMA O RESTA DE LOS ELEMENTOS $x^I/I!$

éste dato es puesto en (OP2).. (OP1) mantendrá la operación acumulativa. La primera operación a realizar es una resta (OP1) ← $x - x^3/3!$, así que hacemos una operación 01 XOR con lo que tenemos en el byte de operación, que es un 00H, da un 01H (restar), que guardamos en el byte de operación, entonces llamamos a SUMFLOT. Al volver de la subrutina, decrementamos la cuenta a cuatro. Este procedimiento se repite, alternándose (con la función XOR) la operación a realizar y, cargándose en (OP2) el siguiente elemento $x^i/i!$. Después de las seis operaciones, la cuenta es cero, y tendremos en OP1 el valor en radianes de la función sen(x).

5.4. LISTADO DEL PROGRAMA EMSAMBLADOR SENO(X).

```
OP CODE    ETIQUETAS  NEMÓNICOS          COMENTARIOS
2A 44 1B   TRANSLATE  LD HL,(1B44H)   ; (OP1) ← x
22 70 1B              LD (1B70H),HL   ;
2A 46 1B              LD HL,(1B46H)   ;
CB BC                 RES 7,H         ;  x, ← +
22 76 1B              LD (1B76H),HL   ;
21 ED 87              LD HL,87EDH     ; (OP2) ← 2π
22 70 1B              LD (1B70H),HL   ;
21 E4 41              LD HL,41E4H     ;
22 72 1B              LD (1B72H),HL   ;
3E 01                 LD A,01H        ; Operación Resta.
32 7C 1B              LD (1B7CH),A    ;
CD 3D 04              CALL SUMFLOT    ; (OP1) ← 2π - x
3A 73 1B              LD A,(1B73H)    ; Verificar signo.
```

```
OP CODE    ETIQUETAS  NEMÓNICOS          COMENTARIOS
17                    RLA                ;
30 47                 JR NC,MATCH        ; Si es positivo
                                         ; entonces x ≤ 2π.
2A 44 1B   XGTH2PI    LD HL,(1B44H)      ; (OP1) ← x
22 70 1B              LD (1B70H),HL      ;
2A 46 1B              LD HL,(1B46H)      ;
22 72 1B              LD (1B72H),HL      ;
21 0C 7C              LD HL,7C0CH        ; (OP2) ← 1/2π
22 74 1B              LD (1B74H),HL      ;
21 51 3F              LD HL,3F51H        ;
22 76 1B              LD (1B76H),HL      ;
CD A2 03              CALL MULTF         ; (OP1) ← x *1/2π
2A 70 1B              LD HL,(1B70H)      ; DEHL ← (OP1)
ED 5B 72 1B           LD DE,(1B72H)      ;
CB 25                 SLA L              ; Quitamos signo,
CB 14                 RL H               ; recorriendo
CB 13                 RL E               ; a la izquierda
CB 12                 RL D               ;
7A                    LD A,D             ; Restamos sesgo al
D6 80                 SUB 80H            ; exponente que nos
57                    LD D,A             ; dice la posición
47                    LD B,A             ; de punto binario,
                                         ; y lo cargamos como
                                         ; cuenta.
CB 25      MINTER     SLA L              ; Recorremos la
```

```
OP CODE     ETIQUETAS   NEMÓNICOS          COMENTARIOS
CB 14                   RL H               ; mantisa a la
CB 13                   RL E               ; izquierda
10 F8                   DJNZ MINTER        ; e-veces.
50                      LD D,B             ;
CB 23       NORMX       SLA E              ; Normalizamos
38 0D                   JR C,NONORMX       ; el número, si es
CB 1B                   RR E               ; necesario.
CB 25                   SLA L              ;
CB 14                   RL H               ;
CB 13                   RL E               ;
15                      DEC D              ;
18 F1                   JR NORMX           ;
18 35       MATCH       JR MATCHX          ; Puente a matchx.
CB 1B       NONORMX     RR E               ; Ya está norma_
                                           ; lizado el número.
7A                      LD A,D             ;
C6 80                   ADD A,80H          ; Sumamos el sesgo
57                      LD D,A             ; al exponente.
3A 47 1B                LD A,(1B47H)       ; Llamamos al signo
CB 27                   SLA A              ; original y, lo
CB 1A                   RR D               ; cargamos en el
CB 1B                   RR E               ; resultado.
CB 1C                   RR H               ;
CB 1D                   RR L               ;
22 70 1B                LD (1B70H),HL      ; (OP1) ← .n
```

```
OP CODE     ETIQUETAS  NEMÓNICOS          COMENTARIOS
ED 53 72 1B            LD (1B72H),DE   ;
21 ED 87               LD HL,87EDH     ; (OP2) ← 2π
22 74 1B               LD (1B74H),HL   ;
21 E4 41               LD HL,41E4H     ;
22 76 1B               LD (1B76H),HL   ;
CD A2 03               CALL MULTF      ; (OP1) ← .n * 2π
2A 70 1B               LD HL,(1B70H)   ; (x) ← (OP1)
22 44 1B               LD (1B44H),HL   ;
2A 72 1B               LD HL,(1B72H)   ;
22 46 1B               LD (1B46H),HL   ;

21 EF 87    MATCHX     LD HL,87EFH     ; (OP1) ← π/2
22 70 1B               LD (1B70H),HL   ;
21 E4 40               LD HL,40E4H     ;
22 72 1B               LD (1B72H),HL   ;
2A 44 1B               LD HL,(1B44H)   ; (OP2) ← (x)
22 74 1B               LD (1B74H),HL   ;
2A 46 1B               LD HL,(1B46H)   ;
22 76 1B               LD (1B76H),HL   ;
3E 01                  LD A,01H        ; Operación resta.
32 7C 1B               LD (1B7CH),A    ;
CD 3D 04    X1         CALL SUMFLOT    ; (OP1) ← π/2 - x
3A 73 1B               LD A,(1B73H)    ; Verificar signo.
17                     RLA             ;
30 51                  JR NC,LEPIOV2   ; Si (+) a x≤π/2
```

```
OP CODE     ETIQUETAS   NEMÓNICOS          COMENTARIOS
21 EC 87    GPIOV2      LD HL,87ECH     ; (OP1) ← π
22 70 1B                LD (1B70H),HL   ;
21 64 41                LD HL,4164H     ;
22 72 1B                LD (1B72H),HL   ;
CD 3D 04    X2          CALL SUMFLOT    ; (OP1) ← π - x
3A 73 1B                LD A,(1B73H)    ; Verificar signo.
17                      RLA             ;
30 3E                   JR NC,LEPI      ; Si (+) a x≤π
21 F1 65    GPI         LD HL,65F1H     ; (OP1) ← 3/2 π
22 70 1B                LD (1B70H),HL   ;
21 CB 41                LD HL,41CBH     ;
22 72 1B                LD (1B72H),HL   ;
CD 3D 04    X3          CALL SUMFLOT    ;(OP1) ← 3/2 π - x
28 4A                   JR Z,LEPI3      ; Verificar cero.
3A 73 1B                LD A,(1B73H)    ; Verificar signo.
17                      RLA             ;
30 44                   JR NC,LEPI3     ; Si (+) a x≤3/2 π
21 ED 87                LD HL,87EDH     ; (OP1) ← 2π
22 70 1B                LD (1B70H),HL   ;
21 E4 41                LD HL,41E4H     ;
22 72 1B                LD (1B72H),HL   ;
CD 3D 04    X4          CALL SUMFLOT    ; (OP1) ← 2π - x
28 64                   JR Z,XEQ0       ; Verificar cero.
3A 73 1B                LD A,(1B73H)    ; Poner signo
CB FF                   SET 7,A         ; negativo.
```

```
OP CODE     ETIQUETAS   NEMÓNICOS          COMENTARIOS
32 73 1B                LD (1B73H),A    ;
2A 70 1B                LD HL,(1B70H)   ; (x) ← (OP1)
22 44 1B                LD (1B44H),HL   ;  x[0,π/2]
2A 72 1B                LD HL,(1B72H)   ;
22 46 1B                LD (1B46H),HL   ;
18 50       LEPIOV2     JR SENOX        ;secuencia senx.
21 EC 87    LEPI        LD HL,87ECH     ; (OP1) ← π
22 70 1B                LD (1B70H),HL   ;
21 64 41                LD HL,4164H     ;
22 72 1B                LD (1B72H),HL   ;
CD 3D 04                CALL SUMFLOT    ; (OP1) ← π - x
2A 70 1B                LD HL,(1B70H)   ; (x) ← x[0,π/2]
22 44 1B                LD (1B44H),HL   ;
2A 72 1B                LD HL,(1B72H)   ;
22 46 1B                LD (1B46H),HL   ;
18 33                   JR SENOX        ; Calcular senx.
2A 44 1B    LEPI3       LD HL,(1B44H)   ; (OP1) ← (x)
22 70 1B                LD (1B70H),HL   ;
2A 46 1B                LD HL,(1B46H)   ;
22 72 1B                LD (1B72H),HL   ;
21 EC 87                LD HL,87ECH     ; (OP2) ← π
22 74 1B                LD (1B74H),HL   ;
21 64 41                LD HL,4164H     ;
22 76 1B                LD (1B76H),HL   ;
CD 3D 04                CALL SUMFLOT    ; (OP1) ← x - π
```

```
OP CODE     ETIQUETAS  NEMÓNICOS          COMENTARIOS
3A 73 1B               LD A,(1B73H)    ;
CB FF                  SET 7,A         ; x, ← 1
32 73 1B               LD (1B73H),A    ; (x) ← -(OP1)
2A 70 1B               LD HL,(1B70H)   ; x[0,π/2]
22 44 1B               LD (1B44H),HL   ;
2A 72 1B               LD HL,(1B72H)   ;
22 46 1B               LD (1B46H),HL   ;
18 02                  JR SENOX        ; Calcular senx.
18 65          XEQ0    JR XEQ0A        ; Puente
                                       ; a sen(0)=0.
DD 21 48 1B    SENX    LD IX,1B48H     ; Cargar índice
                                       ; de direcciones
                                       ; de potencias de x,
                                       ; incia con x².
2A 44 1B               LD HL,(1B44H)   ; (OP1) ← (x)
22 70 1B               LD (1B70H),HL   ; (OP2) ← (x)
22 74 1B               LD (1B74H),HL   ;
2A 46 1B               LD HL,(1B46H)   ;
22 72 1B               LD (1B72H),HL   ;
22 76 1B               LD (1B76H),HL   ;
06 0A                  LD B,0AH        ; Cuenta ← 11
78             POTDX   LD A,B          ;
32 7F 1B               LD (1B7FH),A    ;
CD A2 03               CALL MULTF      ; (OP1) ← x * x
2A 70 1B               LD HL,(1B70H)   ; (x^i) ← x^i
```

```
OP CODE     ETIQUETAS  NEMÓNICOS          COMENTARIOS
DD 75 00               LD (IX+0),L     ;
DD 74 01               LD (IX+1),H     ;
DD 23                  INC IX          ;
DD 23                  INC IX          ;
2A 72 1B               LD HL,(1B72H)   ;
DD 75 00               LD (IX+0),L     ;
DD 74 01               LD (IX+1),H     ;
DD 23                  INC IX          ; Apuntar a la
DD 23                  INC IX          ; siguiente
                                       ; dirección (x^i).
3A 7F 1B               LD A,(1B7FH)    ;
47                     LD B,A          ;
10 D9                  DJNZ POTDX      ; Repetir.
DD 21 4C 1B SINX       LD IX,1B4CH     ; IX ← (x^i)
FD 21 20 1B            LD IY,1B20H     ; IY ← (1/i!)
06 05                  LD B,05H        ; Cuenta a 6
78          ZUM        LD A,B          ; elementos.
32 7F 1B               LD (1B7FH),A    ;
DD 6E 00               LD L,(IX+0)     ;
DD 66 01               LD H,(IX+1)     ;
22 70 1B               LD (1B70H),HL   ; (OP1) ← (x^(3+j))
DD 23                  INC IX          ;
DD 23                  INC IX          ;
DD 6E 00               LD L,(IX+0)     ;
DD 66 01               LD H,(IX+1)     ;
```

```
OP CODE     ETIQUETAS   NEMÓNICOS          COMENTARIOS
22 72 1B                LD (1B72H),HL   ;
18 02                   JR TOCK         ; Puente.
18 3D       XEQ0A       JR XEQ0B        ; Puente.
FD 6E 00    TOCK        LD L,(IY+0)     ; (OP2) ← (1/i!)
FD 66 01                LD H,(IY+1)     ;
22 74 1B                LD (1B74H),HL   ;
FD 23                   INC IY          ;
FD 23                   INC IY          ;
FD 6E 00                LD L,(IY+0)     ;
FD 66 01                LD H,(IY+1)     ;
22 76 1B                LD (1B76H),HL   ;
CD A2 03                CALL MULTF      ; (OP1) ← x^i * 1/i!
DD 2B                   DEC IX          ; Ubicamos nueva
DD 2B                   DEC IX          ; dirección.
2A 70 1B                LD HL,(1B70H)   ; (x^i/i!) ← (OP1)
DD 75 00                LD (IX+0),L     ;
DD 74 01                LD (IX+1),H     ;
DD 23                   INC IX          ;
DD 23                   INC IX          ;
2A 72 1B                LD HL,(1B72H)   ;
DD 75 00                LD (IX+0),L     ;
DD 74 01                LD (IX+1),H     ;
06 06                   LD B,06H        ;
DD 23       INCRE       INC IX          ; Ubicamos
FD 23                   INC IY          ; siguiente
```

```
OP CODE     ETIQUETAS   NEMÓNICOS          COMENTARIOS
10 FA                   DJNZ INCRE      ; dirección.
18 02                   JR PUENTE1      ; Puente.
18 50       XEQ0B       JR ENDSENX      ; Puente a fin.
3A 7F 1B    PUENTE1     LD A,(1B7FH)    ; llamar a la
47                      LD B,A          ; cuenta.
10 9D                   DJNZ ZUM        ; Repetir hasta
                                        ; que cuenta = 0.
3E 00                   LD A,00H        ; Cargar
32 7C 1B                LD (1B7CH),A    ; operación sumar.
DD 21 4C 1B             LD IX,1B4CH     ; IX ← (x^i/i!)
2A 44 1B                LD HL,(1B44H)   ; (OP1) ← x
22 70 1B                LD (1B70H),HL   ; (OP1) como
2A 46 1B                LD HL,(1B46H)   ; acumulador.
22 72 1B                LD (1B72H),HL   ;
3E 05                   LD A,05H        ; 5 operaciones.
47                      LD B,A          ;
78          SUMSENX     LD A,B          ;
32 7F 1B                LD (1B7FH),A    ; Guardar contéo.
DD 6E 00                LD L,(IX+0)     ; IX contiene al
DD 66 01                LD H,(IX+1)     ; elemento x^i/i! .
22 74 1B                LD (1B74H),HL   ;
DD 23                   INC IX          ;
DD 23                   INC IX          ;
DD 6E 00                LD L,(IX+0)     ;
DD 66 01                LD H,(IX+1)     ;
```

```
OP CODE     ETIQUETAS   NEMÓNICOS        COMENTARIOS
22 76 1B                LD (1B76H),HL  ;
3A 7C 1B                LD A,(1B7CH)   ; Alternamos
0E 01                   LD C,01H       ; operación.
A9                      XOR C          ;
32 7C 1B                LD (1B7CH),A   ;
CD 3D 04                CALL SUMFLOT   ; (OP1) ← x ± x^i/i!
06 06                   LD B,06H       ; Ubicamos
DD 23       YES         INC IX         ; dirección del
10 FC                   DJNZ YES       ; elemento sig.
3A 7F 1B                LD A,(1B7FH)   ; Llamar contéo.
47                      LD B,A         ;
10 CE                   DJNZ SUMSENX   ; Repetir hasta
                                       ; que conteo = 0.
C3 13 01    ENDSENX     JP MONITOR     ; Fin de cálculo
                                       ; de la función
                                       ; sen(x). Volver
                                       ; al monitor del
                                       ; SIMMP1.
```

Esta aplicación que nombramos inmediata por ser de hecho una función trigonométrica y, se le pueda considerar muy cercana a la aritmética, supone otras aplicaciones similares, por ejemplo la generación de las otras funciones trigonométricas: cos(x), tan(x), y funciones trascendentes como los logaritmos, etc.; observando los pasos que seguimos aquí se pueden asimilar más rápidamente las

aplicaciones mencionadas, inclusive podríamos meditar un poco y acercarnos a ver, cómo se implementaría la aritmética de matrices; tales aplicaciones nos dan una idea de lo característico que es diseñar aritmética en un microprocesador y más aun, nos sitúan en un espacio poco explorado del diseño con microprocesadores.

CAPITULO VI

CONCLUSIONES.

En la figura 6.1 podemos observar la distribución final de cada uno de los bloques que conforman, tanto la unidad aritmética: SUMFLOT, MULTF y DIVF así como su aplicación SENO(X), podemos observar el espacio destinado a los valores de los inversos de factorial y el espacio para escribir y leer las potencias de x; se encuentran también los bloques de los programas TURN y CHANGE (ver apéndice C). Las cantidades que aparecen a la izquierda representan las direcciones inicial y final de cada bloque dentro del mapa de memoria del SIMMP1.

Se calcularon los tiempos aproximados de cada bloque dentro del SIMMP1, cuya CPU trabaja a una frecuencia de 2 Mhz. La precisión de datos y resultados es de 23 bits (simple). Para la realización de una suma o resta obtuvimos un tiempo de 1.5 ms, que representa realizar 666.666 sumas o restas por segundo. Para realizar una multiplicación el tiempo fue de 1.4 ms, que representa realizar 714.285 multiplicaciones por segundo y, para la división el tiempo fue 1.8 ms que representa realizar 555.555 divisiones por segundo. El bloque que calcula la función sen(x) arrojó un tiempo aproximado de 35 ms, que representa calcular 28.57 funciones sen(x) en un segundo. A partir de estos resultados, nos podemos dar una idea de las posibilidades de operación de la unidad; analizando x aplicación y, contando el número de operaciones que requiere para realizar sus resultados sabremos v.g., que frecuencia de muestreo se puede usar como máxima, para el caso de una aplicación en que se procesen muestras del mundo real (ver apéndice C). Apuntemos que para conocer los tiempos aproximados de cada bloque, todos los datos fueron aleatorios por lo característico del medio usado: se creó un pequeño programa que genera un estado de set (rutina de tiempo) y que dura tanto tiempo como el bloque que halla sido llamado, ésta rutina se repite hasta que sea reseteada la CPU; este estado de set lo sacamos por un puerto y lo conectamos a un osciloscopio, que muestra una señal cuadrada, el período de esta señal representa el tiempo aproximado de duración de cada bloque.

Como ésta rutina de tiempo se está repitiendo sin interrupción, los datos que toma cada bloque operativo son los mismos que él mismo arrojó en una secuencia anterior. Las posibilidades de la unidad aritmética habilitada en el SIMMP1 son grandes, pues aplicaciones donde se realicen operaciones aritméticas son vastas. Un simple filtro digital o un algoritmo de control digital, una báscula que cense un peso y lo multiplique por un precio por kilo. Las aplicaciones didácticas y comerciales pueden ser innumerables y debemos creer que elaborar proyectos de ésta índole ya no deben representar problemas. Se debe tomar en cuenta que entre más rápido sea el microprocesador en cuestión, la eficiencia de una unidad aritmética de punto flotante tenderá a aumentar. Podemos además incluir en este tipo de diseños, y donde lo amerite, formatos de doble precisión.

Cada bloque funcional tuvo que ser analizado y programado por separado, además de que se les sometieron a decenas y decenas de pruebas, hasta que funcionasen correctamente; y como datos, resultados y direcciones son en hexadecimal, la cantidad de información, que hay que irse acostumbrando a interpretar, se multiplica considerablemente. Recalquemos la necesidad y uso de ciertas herramientas, sin las cuales, el diseño y desarrollo de ésta unidad aritmética de punto flotante y su aplicación inmediata en la generación del sen(x), hubieran sido poco menos que imposibles. 1°, la Utilización de microcomputadoras personales (PC).

2° Uso de un paquete comercial que ensambla programas en nemónico y simula la operación de la CPU Z80. Y 3°, apoyo de programas en alto nivel (BASIC) diseñados y programados para ésta tesis, que nos permitieron la conversión rápida y exacta de valores numéricos decimales, al formatos de punto flotante que usamos (IEEE) y, viceversa (ver apéndice B). Una vez que la unidad aritmética y su aplicación funcionaron en el simulador, vino su prueba final y más importante, ésta consistió, en introducir todos los bloques funcionales al SIMMP1 y hacer, por supuesto, que funcionaran adecuadamente como funcionaron en el simulador. Para ésta tarea tuvimos que grabar en un diskette, que contenía precisamente al editor del SIMMP1 (ver apéndice A), cada uno de los diferentes bloques o programas. El paquete simulador mencionado nos ensambló todos los bloques entregándonos el OP CODE. Aproximadamente 2K bytes de OP CODE tuvieron que ser escritos byte por byte en el editor del SIMMP1. Este trabajo es relativamente fácil, pero muy propenso a errores; aprende uno a escribir y a revisar largas cadenas de bytes.

Respecto a los programas BASIC, consideremos la posibilidad de aumentar elementos a estos, para que puedan convertir *cadenas* de datos provenientes de alguna PC al SIMMP1 y viceversa; inclusive, podrían llegar a ser parte del editor del SIMMP1.

Aunque por su carácter particular, de ser una unidad aritmética para el SIMMP1 (Z80 CPU), se puede llegar a suponer que

ésta no tenga más futuro que el SIMMP1, sin embargo los conocimientos concisos que aporta, pueden servir para alguna investigación que algún interesado realice al respecto. Por otro lado, el hecho de que ésta tesis haya sido la realización de programas en emsamblador y por lo mismo se acerque más al lado programación que al lado electrónico se suponga una tesis de computación, pero cuando un ingeniero en electrónica requiere una unidad aritmética y no la halla; para continuar sus pasos electrónicos (analógicos o digitales) tendrá por fuerza, detenerse y programar. Deteniéndonos en los detalles, podemos añadir: que lograr implementar a nivel emsamblador las operaciones aritméticas y la función sen(x) ha resultado una experiencia de trabajo y ánimo, pues al transcurrir el desarrollo, se da uno cuenta que a estos niveles nada se supone y nada es obvio ó, a veces lo supuesto es tan sólo lógico y hay que implementarlo casi todo. Al principio, es difícil dárse cuenta de todo lo necesario tanto teórico como práctico. La programación a nivel emsamblador es un trabajo que se puede comparar al realizado por hormigas: minúsculas instrucciones que logran construir todo un ejemplo de organización y trabajo; realizando así, un sistema creado a partir de cientos de microinstrucciones. En cierta forma, aquí, la ciencia y la técnica son madera, y los resultados forma. Fue un trabajo de tesis lento, por lo característico de su realización. Su originalidad reside, en que siendo posible su realización, sea difícil encontrar recursos

concretos que apoyen su desarrollo. La realización de otra unidad aritmética podría agilizarse, (conociendo el procesador), si se empieza primero, v.g., por definir el formato de los números, realizar programas en alto nivel para las transformaciones numéricas, contar con un simulador del procesador ó, si se tiene una computadora con el procesador a programar mejor. El trabajo en equipo divide el tiempo de desarrollo, un persona puede dedicarse a hacer funcionar a la suma y resta, otro a la multiplicación y otro más a la división. Sólo me resta agradecer al ingeniero Antonio Salvá Calleja la oportunidad de realizar ésta tesis bajo su dirección, así como a su apoyo y consejos a lo largo de éste trabajo.

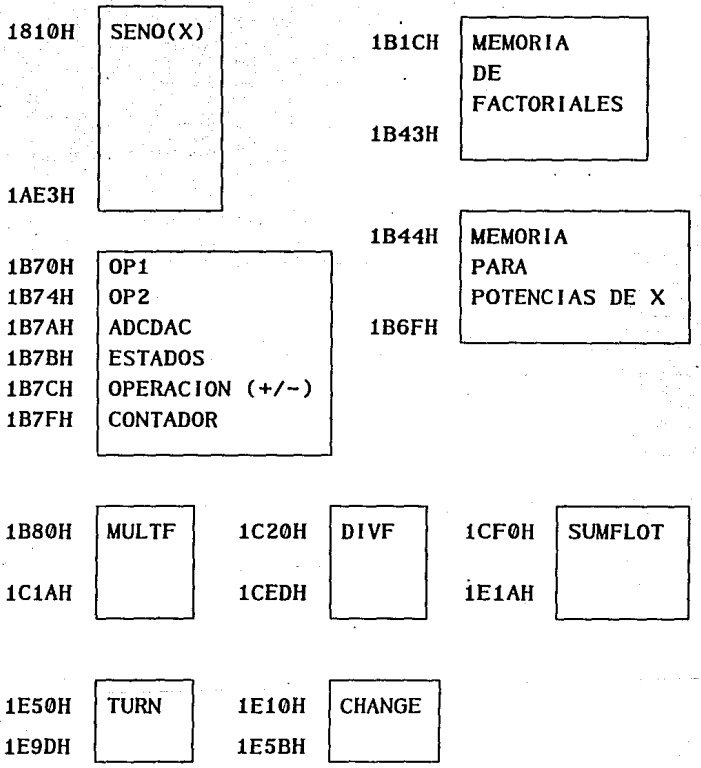


FIGURA 6.1 BLOQUES FUNCIONALES CON SUS DIRECCIONES CORRESPONDIENTES DEL MAPA DE MEMORIA DEL SINMP1.

APÉNDICE A

EL S I M M P - 1

El SIMMP-1 es una computadora contenida en una tarjeta (SBC : single board computer), que tiene la capacidad de ser programada por medio de una computadora tipo PC a través de un enlace serie. El SIMMP-1 fue Diseñado por un grupo de profesores de la facultad de Ingeniería de la UNAM dedicados al diseño y capacitación en instrumentación y control.

EL SIMMP-1 se concibió originalmente para fines educativos, sin embargo, puede ser empleado para diseñar periféricos inteligentes de computadoras que tengan puerto serie RS-232 o bien sistemas autónomos dedicados a aplicaciones de instrumentación y control específicas.

A.1 ELEMENTOS Y CARACTERÍSTICAS DEL SIMMP-1 :

a) Como CPU el Z80.

b) Frecuencia de reloj : 2 MHz.

c) Memoria de sólo lectura : 4 K. Una EPROM 2716 conteniendo el firmware de manejo del sistema y una base para que el usuario pueda colocar otra EPROM 2716 con alguna aplicación especifica. Existe además la posibilidad en la tarjeta para que la EPROM adicional pueda ser programada.

d) Memoria RAM : 4 K . Una memoria RAM estática 6116, donde el usuario puede probar y depurar programas desde una computadora PC y, una base adicional para otra memoria 6116 que el usuario puede colocar si su aplicación lo requiere.

e) Puertos y/o periféricos auxiliares : Puerto triple 8255. Este circuito contiene tres puertos que pueden ser programados por el usuario como de entrada o salida de acuerdo con los requerimientos necesarios en cada aplicación.

f) Temporizador programable 8253. Este circuito contiene tres canales que pueden programarse de diversas maneras, ya sea como contadores o como sistema generador de pulsos (one shot). El primer canal lo reserva el sistema para generar una señal de reloj en la base a la cual está el baudaje de uno de los dos puertos serie del sistema. El segundo canal se emplea como parte del hardware que el sistema requiere para ejecutar programas paso a paso, existiendo facilidades en el sistema para desligar a este canal de dicha aplicación, a modo de que el usuario, pueda emplearlo en alguna

otra aplicación. El tercer canal está libre para uso del usuario.

g) Puerto serie 1 : Este puerto permite una comunicación serie bidireccional con la computadora PC que maneja el sistema, empleando para ello un puerto de salida de 1 bit y un puerto de entrada de 1 bit que son parte del PPI 8255. Los baudajes a los que este puerto puede trabajar son : 300, 600, 1200, y 2400 bauds. El formato de transmisión es asíncrono con 8 bits de datos, un bit de paro y ausencia de bit de paridad. El control de este puerto es por software.

h) Puerto serie 2 : Este puerto esta realizado mediante el circuito integrado 8251, que es un subsistema dedicado exclusivamente a implantar un puerto bidireccional síncrono o asíncrono a modo tal que el microprocesador al cual esta ligado sólo debe programarlo inicialmente para después recibir o transmitir caracteres mediante simples operaciones de lectura o escritura a puerto. El formato de transmisión es asíncrono con 8 bits de datos, 1 bit de paro y ausencia de bit de paridad, el usuario puede hacer modificaciones en el formato de transmisión escribiendo el software necesario para ello.

i) Líneas de selección de puertos : El usuario dispone de cuatro líneas de selección de puerto que son las salidas Y_3, Y_4, Y_6, y Y_7 del CI 74LS138 usado como paginador de puertos por el sistema, las 16 direcciones de puerto asociadas pueden verse en el mapa de puertos del SIMMP-1.

j) Líneas de selección de memoria : El usuario dispone también

de las cuatro líneas siguientes de selección de memoria Y_6, Y_7, Y_5 y Y_2 asociadas con el 74LS138 usado por el sistema como paginador de memoria; las direcciones de memoria asociadas pueden verse en el mapa de memoria del SIMMP-1.

k) El usuario dispone además, completamente, de los buses de datos, direcciones y control, con la excepción de las entradas NMI y WAIT, usadas por el sistema como parte de la lógica de ejecución paso a paso y, de programación de la EPROM auxiliar del sistema. Las líneas de los buses se encuentran disponibles en conectores dentro y fuera de la tarjeta.

l) Programador de EPROM : El sistema cuenta con facilidades que permiten programar la EPROM auxiliar 2716 del sistema con fuente de programación proveniente de disco en computadora PC.

En la figura A.1 se presenta la arquitectura específica del SIMMP-1.

A.2 PAGINACIÓN DE MEMORIA DEL SIMMP-1.

En la figura A.2 se presenta el mapa de memoria del SIMMP-1. La paginación esta hecha en bloques de 2k bytes, pudiéndose decodificar hasta 16k bytes, esto es, de la dirección 0000H hasta la 3FFFH. En la versión inicial del sistema, únicamente se ocupan 8k bytes de memoria, de los cuales 4K bytes son de memoria RAM situados de la dirección 1800H a la 27FFH y, 4K bytes de EPROM situados de la dirección 0000H a la 0FFFH. Físicamente, la paginación de la memoria se realiza mediante el CI 74LS138 (ver

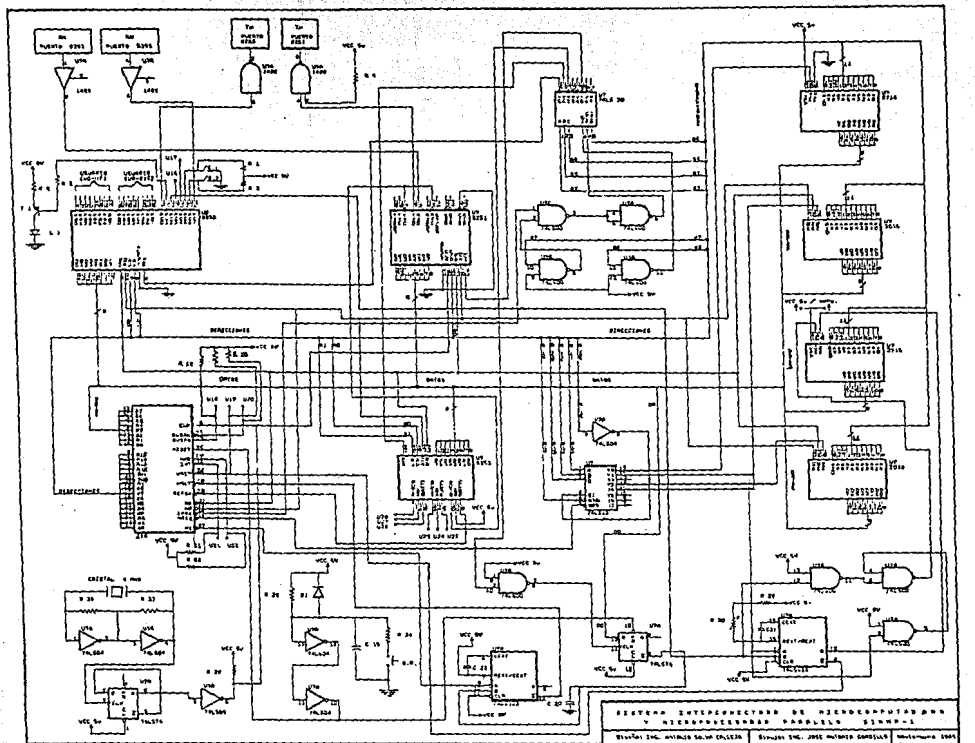


FIGURA A.1

figura A.1). Como se observa en la figura A.2 la zona de memoria RAM empleada por la pila (stack), va de las direcciones 2780H a la 27FFH; de la dirección 2700H a la 2780H se define una zona de memoria para uso del núcleo básico de entrada salida del sistema, esto es, tales localidades de memoria son usadas tanto para el firmware del sistema, como para el software de la microcomputadora que lo comanda. A este intervalo de direcciones se le denomina zona de servicio, y se le recomienda al usuario no modificar el contenido de cualquier localidad en este rango. Las direcciones de memoria comprendidas de la 2600H a la 26FFH son utilizadas para localizar rutinas de servicio tanto de interrupción no enmascarables como enmascarables de modo 1. Como en el caso de la zona de servicio se recomienda al usuario no modificar el contenido de las localidades de la zona de servicio de interrupciones; aunque hay que observar, que esto no es tan crítico como en la zona de servicio, ya que no todas las aplicaciones usarán interrupciones.

A.3 PAGINACIÓN DE PUERTOS DEL SIMMP-1.

En la figura A.3 se muestra el mapa de puertos del SIMMP-1. Con el hardware contenido en la tarjeta se pueden decodificar hasta 32 diferentes direcciones asociadas con puertos, de las cuales el usuario puede disponer de 16. Las otras 16 son usadas por el sistema en los puertos que el mismo contiene. A continuación se describe lo anterior.

1°. Las direcciones 14H y 15H están vinculadas con las direcciones de control y de datos del puerto serie 8255.

2°. Las direcciones 00H a 03H están vinculadas con el multipuerto paralelo 8255. Por lo tanto para los puertos A, B, y C de este chip, las direcciones respectivas son 00H, 01H y 02H, y la dirección 03H, la correspondiente al registro de control del chip.

3°. Las direcciones 04H a 07H están vinculadas con el temporizador programable 8253 de tres canales. La dirección 07H la correspondiente al registro de control del chip, siendo las direcciones 04H a 06H las asociadas con cada uno de los tres canales de temporización del 8253.

4°. La dirección 08H es empleada por un puerto de salida de 1 bit, que es parte de la lógica de ejecución de programas de paso a paso.

Las otras cuatro señales de habilitación de puerto provenientes del decodificador de puertos están disponibles para el usuario a modo de que éste pueda colocar más puertos al sistema.

A.4 SOFTWARE EN LA MICROCOMPUTADORA.

El software para el SIMMP-1 está escrito en un lenguaje de alto nivel (GWBASIC). El programa se denomina CCA y permite al usuario operar el sistema SIMMP-1 desde la microcomputadora, manejándose la información en notación hexadecimal. El programa se basa en menús, el principal se denomina de *comandos*, y genera ramificaciones a otros menús. *Comandos* consta de nueve opciones:

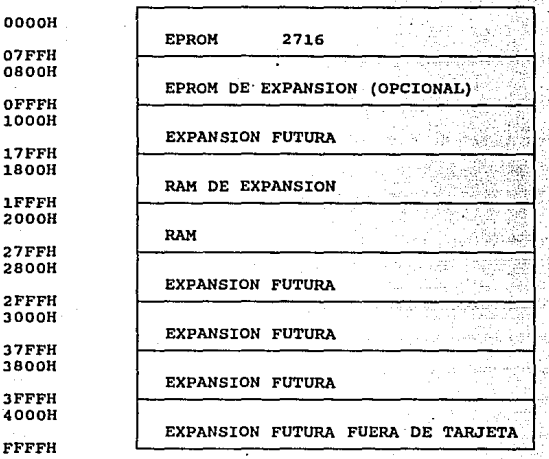

0000H
EPROM 2716
07FFH
0800H
EPROM DE EXPANSION (OPCIONAL)
0FFFH
1000H
EXPANSION FUTURA
17FFH
1800H
RAM DE EXPANSION
1FFFH
2000H
RAM
27FFH
2800H
EXPANSION FUTURA
2FFFH
3000H
EXPANSION FUTURA
37FFH
3800H
EXPANSION FUTURA
3FFFH
4000H
EXPANSION FUTURA FUERA DE TARJETA
FFFFH

Fig A.2

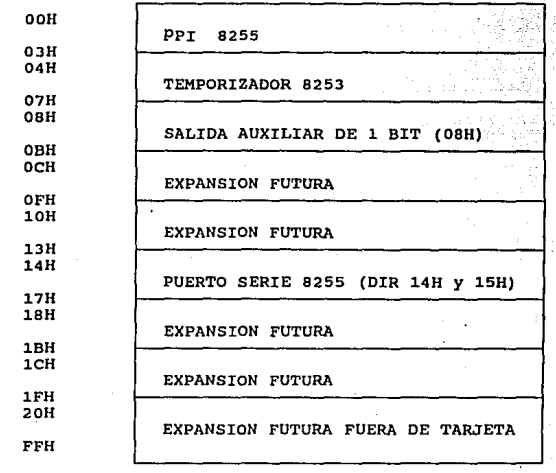

00H
PPI 8255
03H
04H
TEMPORIZADOR 8253
07H
08H
SALIDA AUXILIAR DE 1 BIT (08H)
0BH
0CH
EXPANSION FUTURA
0FH
10H
EXPANSION FUTURA
13H
14H
PUERTO SERIE 8255 (DIR 14H y 15H)
17H
18H
EXPANSION FUTURA
1BH
1CH
EXPANSION FUTURA
1FH
20H
EXPANSION FUTURA FUERA DE TARJETA
FFH

Fig A.3

1. Cargar un programa en lenguaje de máquina Z80.
2. Cargar de disco un programa para Z80.
3. Cargar en disco un programa para Z80.
4. Bajar a SIMMP-1 un programa que se autoejecute.
5. Editar un programa.
6. Manipulación de memoria.
7. Definir baudaje.
8. Manejo de disco.
9. Retornar al sistema operativo.

Opción 1. CCA pide direcciones inicial y final, y el nombre del programa que se va a cargar. Después despliega en la pantalla de la PC las direcciones en forma sucesiva, el usuario introduce un byte de información hexadecimal por dirección.

Opción 2. CCA pide el nombre de un programa y la unidad de disco. Después, la cadena de bytes que forman el programa se despliega en la pantalla.

Opción 3. CCA simplemente guarda en disco un programa y sus direcciones inicial y final.

Opción 4. CCA baja al SIMM-1 las direcciones inicial y final de un programa para Z80, así como la cadena de bytes que lo forman. Una vez que el programa es bajado, se autoejecuta.

Opción 5. CCA nos lleva al menú de edición, donde podemos editar un programa en lenguaje de máquina. Las opciones de este menú son:

a) Listar el programa. Permite listar todo ó, parte del programa.

b) Borrar bytes. Aquí, podemos borrar bytes del programa en lenguaje de máquina.

c) Insertar bytes. Inserta instrucciones en lenguaje de máquina de un programa en edición.

d) Cambiar bytes. Permite cambiar uno o más bytes de un programa en lenguaje de máquina.

e) Continuar. Con esta opción, el sistema pasa a un submenú que comprende las siguientes acciones:

1) Editar. Retornamos a la opción 5 de *comandos*.

2) Retornar a menú de *comandos*.

Opción 6. CCA nos lleva a un menú con los siguientes puntos.

a) Examinar memoria del SIMMP-1. Permite examinar el contenido de las localidades de memoria del sistema.

b) Cargar memoria del SIMMP-1. Permite cargar bytes en memoria RAM del sistema.

c) Ejecutar un programa cargado previamente en memoria RAM del sistema. El usuario puede escoger entre ejecución paso a paso ó, velocidad plena.

d) Transferir bloques en memoria de SIMMP-1. Aquí el usuario, puede mover bloques contenidos en memoria ROM o RAM a memoria RAM.

e) Acomodar bloque provenientes de disco en memoria RAM del SIMMP-1. Mediante ésta opción el usuario puede bajar a memoria

RAM una cadena de bloques contenidos originalmente en disco.

f) Programar EPROM de expansión del SIMMP-1. Con esta opción, el usuario podrá programar la EPROM auxiliar del sistema, cuya fuente de datos puede venir de: disco, memoria del SIMMP-1 ó, datos tecleados en la PC.

g) Retornar a menú de *comandos*.

Opción 7. CCA permite al usuario programar el baudaje de cualquiera de los dos puertos serie del sistema, los baudajes permisibles son: 300, 600, 1200, y 2400 bauds.

Opción 8. CCA nos lleva a un menú con los siguientes puntos.

a) Examinar directorio activo en alguna unidad de disco.

b) Cambiar directorio activo en alguna unidad de disco.

c) Borrar un programa en código para Z80 en alguna unidad de disco.

d) Retornar a menú de *comandos*.

Opción 9. Mediante esta opción retornamos al sistema operativo.

A.5 EMSAMBLADOR CRUZADO.

El software del lado de la microcomputadora asociado con el SIMMP-1 cuenta con un programa emsamblador escrito previamente para el sistema. El emsamblador es de dos pasadas, admitiendo cualquier nemónico del Z80. En la primera pasada se genera un código de máquina para todas las instrucciones a excepción de las que involucren saltos o llamadas a etiquetas, colocando al emsamblador

en tales casos, banderas reconocibles. En la segunda pasada, se coloca el código correcto correspondiente a las instrucciones de salto o llamada. El manejo del emsamblador por parte del usuario es sumamente sencillo, se maneja al igual que el programa CCA (en base a menús). El principal o de comandos, consta de las siguientes opciones:

1. Cargar un programa en lenguaje emsamblador para Z80.
2. Tomar de disco un programa fuente en emsamblador.
3. Cargar en disco un programa fuente en emsamblador.
4. Editar un programa fuente.
5. Guardar en disco el código objeto correspondiente a un programa fuente recién ensamblado.
6. Ensamblar un programa fuente, especificando el usuario la dirección inicial a partir de la cual se desea cargar el programa objeto en la memoria del SIMMP-1. Una vez que un programa está ensamblado, el usuario podrá bajarlo al SIMMP-1 para su ejecución, a modo de poderle hacer una prueba al programa, sin la necesidad de almacenarlo en disco. Si es necesario hacer algún cambio, el usuario puede retornar al editor para hacer las modificaciones necesarias, a fin de ensamblarlo y probarlo de nuevo. Una vez que el programa funciona correctamente, se puede cargar su código objeto en disco, siendo tal archivo, compatible con el programa CCA de manejo hexadecimal del SIMMP-1.

APÉNDICE B

PROGRAMAS BASIC.

Los programas que presentamos aquí, fueron creados con el objetivo de contar con una herramienta para el desarrollo eficaz de nuestra unidad aritmética y de su aplicación. Después de que se programaba un bloque de la unidad aritmética y se le hacían pruebas, era necesario a mano y a calculadora, crear datos específicos y, una vez obtenido resultados de los bloques de la unidad, había que volverlos a su equivalente decimal para su apreciación y exacta interpretación. Esta tarea no es muy ardua cuando se trata de transformar algunos datos y algunos resultados, pero cuando lo que hay que verificar son realmente muchos datos o resultados, entonces la tarea sí se vuelve muy complicada.

Los programas IEEE.BAS y CAMBIO.BAS son las herramientas que nos ayudaron a ésta tarea complicada. IEEE.BAS transforma datos decimales al formato estándar de punto flotante de la IEEE para 32 bits; y CAMBIO.BAS transforma un número representado en el estándar de la IEEE para punto flotante 32 bits, a decimal. Con un ejemplo explicaremos la estructura general de los programas.

Tenemos un número decimal: 45.5

Transformándolo a binario: 101101.1000

El mismo número en notación científica: $.455 \times 10^2$

Igualando bases decimal y binaria:

$$10^n = 2^x$$

$$n = x \log 2$$

$$x = n / \log 2$$

para el ejemplo: $x = 2 / 0.30103 = 6.6438562$

El número en base 2 notación científica : 0.7109375×2^6

Cambiando la mantisa a binario: 0.101101100×2^6

Si queremos volver de base 2 a base 10 sería, igualando bases

$$2^x = 10^n$$

$$n = x \log 2$$

para el ejemplo: $n = 6 \log 2 = 1.80618$

Así: $0.7109375 \times 10^{1.80618}$

ó: $0.7109375 \times 10^{.80618} \times 10^1$

ó: 4.55×10^1

B.1. IEEE.BAS.

En IEEE.BAS, una vez que se tiene el número en base 2, se procede a cambiar a binario la fracción resultante, como lo vimos en el capitulo I; después, normalizamos ésta fracción o mantisa y transformamos el exponente decimal a binario, aplicamos el sesgo y verificamos sobreflujo. Una vez que tenemos todos los unos y ceros de que está compuesto el número, desde A(1) hasta A(32), preguntamos por grupos de cuatro bits y les asignamos un valor hexadecimal; terminando así IEEE.BAS. La mayoría de los PRINT's aparecen con el objetivo de observar los resultados parciales durante las conversiones. El resultado será una cadena de valores hexadecimales, siendo el primer valor donde está el signo.

```
100 REM     IEEE.BAS

120 INPUT " CUAL ES EL NÚMERO EN BASE 10 "; AD:PRINT " AD= ";AD
125 SIGNO = SGN(AD): AD = ABS(AD)
130 IF AD= 0 THEN DIM A(32) : GOTO 2000
140 IF AD= 1 THEN DIM A(32) : GOTO 901
150 IF AD> 1 THEN 240
153 REM
155 REM           CAMBIO DE BASE 10  A  2
157 REM
160 N = 0
170 AD= AD* 10
180 N = N - 1
190 IF AD<1 GOTO 170
200 AD= AD/ 10 : N = N + 1
210 GOTO 300
240 N = 0
250 AD= AD/ 10
260 N = N + 1
270 IF AD < 1 THEN 300
275 GOTO 250
300 PRINT " NOTACION CIENTIFICA DE  A =";AD:PRINT"N=";N
310 S = SGN(N)
320 X = N / .30103    :PRINT "EXP BASE 2=";X
330 X = ABS (X)
```

```
340 XFR = X - INT (X)
350 FR2= AD* 2 ^ (S*XFR)
360 X = S * INT(X)
362 IF FR2 < 1 THEN 370
364    FR2 = FR2 / 2 : X = X + 1  : GOTO 362
370 PRINT "FRBIN";FR2:PRINT "EXPBIN";X
372 REM
375 REM     TRANSFORMACION DE FRACCION DECIMAL A BINARIA
377 REM
379 DIM A(32)
380 IF SIGNO =  1 THEN A(1) = 0 : GOTO 398
382                    A(1) = 1
398 FOR J = 10 TO 32
399     FR2 = FR2 * 2
400     A(J)= INT (FR2)
410     FR2 = FR2 - A(J)
420 NEXT J
600 PRINT A(10),A(11),A(12),A(13),A(14),A(15),A(16),A(17),
        A(18),A(19),A(20),A(21),A(22),A(23),A(24),A(25),
        A(26),A(27), A(28),A(29),A(30),A(31),A(32)
605 PRINT
606 PRINT
608 REM   NORMALIZACION DE LA MANTISA
609 REM
610 I = 10
620 IF A(I) = 1 THEN 800
625 X=X-1
630 A(I) = A(I+1) : I = I+1
640 IF I< 32 THEN 630
650 GOTO 610
800 PRINT A(10),A(11),A(12),A(13),A(14), A(15),A(16),A(17),
        A(18),A(19),A(20),A(21),A(22),A(23), A(24),A(25),
        A(26),A(27),A(28),A(29),A(30),A(31),A(32)
802 REM
804 REM TRANSFORMACION DEL EXPONENTE DECIMAL A BINARIO
806 REM
810 PRINT "EXP BASE2 =";X : XBIAS = X + 128: PRINT "EXPBIAS=";XBIAS
820 IF XBIAS > 255 THEN PRINT "SOBREFLUJO DEL EXPONENTE":GOTO 2396
830 IF XBIAS <  0  THEN PRINT "  SUBFLUJO DEL EXPONENTE":GOTO 2396
835 N = XBIAS
840 FOR J = 9 TO 2  STEP -1
850     N = N / 2 : NF = N -  INT (N)
860     IF NF = 0  THEN A(J) = 0 : GOTO  880
870                    A(J) = 1
880     N = INT (N)
890 NEXT J
900 PRINT A(2),A(3),A(4),A(5),A(6),A(7),A(8),A(9) : GOTO 2000
901 IF SIGNO = 1 THEN A(1) = 0 : GOTO 903
902                   A(1) = 1
903 A(2) = 1 : A(9) = 1 : A(10)= 1
```

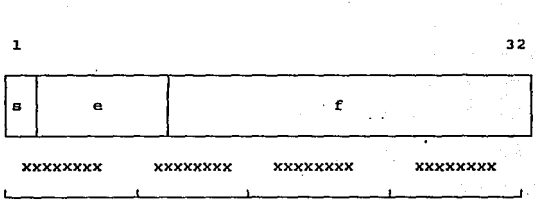
```
2000 REM          TRANSFORMACION AL ESTANDAR DE LA IEEE

2040 I = 1
2050 FOR K = 1 TO 8
2100      IF A(I) = 1 THEN 2220
2110         IF A(I+1)=1 THEN 2170
2120            IF A(I+2)=1 THEN 2150
2130               IF A(I+3)=1 THEN HEXA$(K)="1" : GOTO 2340
2140                               HEXA$(K)="0" : GOTO 2340
2150               IF A(I+3)=1 THEN HEXA$(K)="3" : GOTO 2340
2160                               HEXA$(K)="2" : GOTO 2340
2170            IF A(I+2)=1 THEN 2200
2180               IF A(I+3)=1 THEN HEXA$(K)="5" : GOTO 2340
2190                               HEXA$(K)="4" : GOTO 2340
2200               IF A(I+3)=1 THEN HEXA$(K)="7" : GOTO 2340
2210                               HEXA$(K)="6" : GOTO 2340
2220         IF A(I+1)=1 THEN 2280
2230            IF A(I+2)=1 THEN 2260
2240               IF A(I+3)=1 THEN HEXA$(K)="9" : GOTO 2340
2250                               HEXA$(K)="8" : GOTO 2340
2260               IF A(I+3)=1 THEN HEXA$(K)="B" : GOTO 2340
2270                               HEXA$(K)="A" : GOTO 2340
2280            IF A(I+2)=1 THEN 2310
2290               IF A(I+3)=1 THEN HEXA$(K)="D" : GOTO 2340
2300                               HEXA$(K)="C" : GOTO 2340
2310               IF A(I+3)=1 THEN HEXA$(K)="F" : GOTO 2340
2320                               HEXA$(K)="E"
2340    I = I + 4
2355 NEXT K
2360 PRINT " EL NUMERO EN PUNTO FLOTANTE SIGUIENDO EL "
2370 PRINT " FORMATO ESTANDAR DE 32 BITS DE LA  IEEE  ES:"
2390 PRINT   HEXA$(1),HEXA$(2),HEXA$(3),HEXA$(4)
2395 PRINT   HEXA$(5),HEXA$(6),HEXA$(7),HEXA$(8)
2396 END
```

B.2. CAMBIO.BAS.

En CAMBIO.BAS se lee una cadena de valores hexadecimales que representan al número en el estándar de la IEEE, se les asigna su equivalente en binario desde C(1) hasta C(32), luego se agrupan según los que pertenezcan a la mantisa, al exponente y el bit de signo. El bit de signo es C(1), el exponente va de C(2) a C(9) y la fracción va de C(10) a C(32), se hacen las multiplicaciones

pertinentes para obtener los valores decimales respectivos y, por último, una multiplicación del valor decimal de la mantisa por el valor decimal del exponente y por el signo, genera el valor decimal buscado. Comenzamos dando los valores HH$ (hexadecimales) de izquierda a derecha.




```
100 REM     CAMBIO.BAS

105 DIM C(32):DIM A$(8)
110 J=1
200 FOR I= 1 TO 8
300 INPUT " DIGITO EN HEXADECIMAL";A$(I)
400 IF A$(I)="0" THEN C(J)=0:C(J+1)=0:C(J+2)=0:C(J+3)=0:GOTO 1500
500 IF A$(I)="1" THEN C(J)=0:C(J+1)=0:C(J+2)=0:C(J+3)=1:GOTO 1500
600 IF A$(I)="2" THEN C(J)=0:C(J+1)=0:C(J+2)=1:C(J+3)=0:GOTO 1500
700 IF A$(I)="3" THEN C(J)=0:C(J+1)=0:C(J+2)=1:C(J+3)=1:GOTO 1500
800 IF A$(I)="4" THEN C(J)=0:C(J+1)=1:C(J+2)=0:C(J+3)=0:GOTO 1500
900 IF A$(I)="5" THEN C(J)=0:C(J+1)=1:C(J+2)=0:C(J+3)=1:GOTO 1500
910 IF A$(I)="6" THEN C(J)=0:C(J+1)=1:C(J+2)=1:C(J+3)=0:GOTO 1500
920 IF A$(I)="7" THEN C(J)=0:C(J+1)=1:C(J+2)=1:C(J+3)=1:GOTO 1500
930 IF A$(I)="8" THEN C(J)=1:C(J+1)=0:C(J+2)=0:C(J+3)=0:GOTO 1500
940 IF A$(I)="9" THEN C(J)=1:C(J+1)=0:C(J+2)=0:C(J+3)=1:GOTO 1500
950 IF A$(I)="A" THEN C(J)=1:C(J+1)=0:C(J+2)=1:C(J+3)=0:GOTO 1500
960 IF A$(I)="B" THEN C(J)=1:C(J+1)=0:C(J+2)=1:C(J+3)=1:GOTO 1500
970 IF A$(I)="C" THEN C(J)=1:C(J+1)=1:C(J+2)=0:C(J+3)=0:GOTO 1500
980 IF A$(I)="D" THEN C(J)=1:C(J+1)=1:C(J+2)=0:C(J+3)=1:GOTO 1500
990 IF A$(I)="E" THEN C(J)=1:C(J+1)=1:C(J+2)=1:C(J+3)=0:GOTO 1500
999 IF A$(I)="F" THEN C(J)=1:C(J+1)=1:C(J+2)=1:C(J+3)=1:GOTO 1500
```

```
1500 J=J+4
1550 NEXT I
1560 CLS
1570 PRINT "HEXADECIMAL A BINARIO"
1580 PRINT
1600 PRINT A$(1),C(1),C(2),C(3),C(4)
1700 PRINT A$(2),C(5),C(6),C(7),C(8)
1800 PRINT A$(3),C(9),C(10),C(11),C(12)
1900 PRINT A$(4),C(13),C(14),C(15),C(16)
2000 PRINT A$(5),C(17),C(18),C(19),C(20)
2100 PRINT A$(6),C(21),C(22),C(23),C(24)
2200 PRINT A$(7),C(25),C(26),C(27),C(28)
2300 PRINT A$(8),C(29),C(30),C(31),C(32)
2500 IF C(1)=0 THEN 2800
2600 S#=-1
2700 GOTO 2900
2800 S#=1
2900 E#= (C(2)*2^7)+(C(3)*2^6)+(C(4)*2^5)+(C(5)*2^4)+
        (C(6)*2^3)+(C(7)*2^2)+(C(8)*2^1)+(C(9)*2^0):E#=E#-128
2910 PRINT "EXPONENTE";E#
2920 F#=0
3000 FOR J=10 TO 32
3100      F#=F#+C(J)*2^(9-J)
3200 NEXT J
3210 PRINT " FRACCION=";F#
3400 R#=S#*2^E#*F#
3500 PRINT "EL NUMERO EN DECIMAL ES    ";R#
3600 END
```

APÉNDICE C

PUNTO FLOTANTE : MUNDO REAL.

Un aprovechamiento singular de la unidad aritmética, estriba en la posibilidad de que sus capacidades se comuniquen con el mundo real; no estamos hablando de interfases, sino más bien de formatos de registros que representan a números o valores.

En este apéndice presentamos dos programas en emsamblador que nos ayudarán a adecuar datos provenientes del mundo real a la unidad aritmética y, adecuar resultados provenientes de la unidad aritmética para el mundo real. Como ya sabemos, los números representados en el formato de punto flotante en nuestra unidad aritmética constan de 32 bits, de los cuales 23 son para

representar a la mantisa, 8 bits para representar al exponente y 1 bit para representar al signo. Ahora, el problema es, de estos 32 bits, hacer un número equivalente de 8 bits, y éste, sirva como entrada a un convertidor digital analógico; tal nivel analógico representará al número o valor original en punto flotante. De forma similar, necesitaremos convertir un valor analógico, a nuestro formato numérico de punto flotante.

Los programas CHANGE y TURN nos permiten realizar lo anterior. Algo importante que hay que tener en cuenta es, que el hecho de convertir un número de 32 bits a otro equivalente de 8 bits, trae consigo una posible e irremediable perdida de precisión.

Los límites en la magnitud de los números en nuestro formato de punto flotante es de aproximadamente 1.7×10^{38} para los positivos y, de 3.8×10^{38} para los negativos, cualquier número fuera de éste rango se le considera como un sobreflujo. Por otro lado, los límites para un número de 8 bits son de -128 a +127, es decir que, para empezar la diferencia en rangos de magnitud son abismales. Esta última comparación, es del lado binario, pasándonos a lado analógico (mundo real), los niveles de voltaje (que pueden representar una infinidad de variables físicas, cuyas magnitudes oscilen en valores y rangos variados) sólo manejarán rangos específicos, según el convertidor y la aplicación. Con todo esto, nos enfrentamos a un problema, no sólo relativo a magnitudes numéricas y reales, sino también a la interpretación que se tenga de ellos.

La solución que proponemos a éste problema, es limitar el rango de valores permisibles en punto flotante de [-128 a 127]. Con esto los único que estamos estableciendo, es una correspondencia biunivoca (hasta donde se pueda) entre los valores de 32 bits con los de 8 bits y, viceversa. De esta forma obligamos al usuario, a que sus datos que desee sacar o meter del o al sistema, caigan en éste rango.

C.1 CHANGE.

El programa CHANGE realiza uno de estos cambios, el de 8 bits a 32 bits. El registro del dato de 8 bits está en la dirección (1B7AH) y el dato de 32 bits lo tendremos en (OP1). Primeramente cargamos el dato de 8 bits en el acumulador, verificamos si es cero, si es cero, cargamos en (OP1) puros ceros; si no es cero, preguntamos por el signo del número, recordemos que el número está en complemento a dos. Si el número es positivo vamos a DATPOS y si es negativo a DATNEG, cuando es positivo, cargamos a B con 7, interpretándose con esto que nuestro entero es ahora fracción y B contiene la posición del punto binario. Después forzamos a que esté normalizada ésta fracción de 8 bits; en este momento B representa el exponente, una vez esto, hacemos una carga de C con lo de A, cargamos a HL con ceros, que serán los bytes faltantes de la fracción, por último recorremos desde el acarreo (CY=0) a la izquierda CY →B→C, se sesga el exponente encendiendo en bit 6 de B y formamos así el nuevo números en punto flotante. Cargamos BCHL EN

(OP1) terminado CHANGE. Cuando el dato es negativo, primero se complementa a dos el dato y, se le aplica el mismo procedimiento que cuando es positivo, salvo la excepción de incluir un signo negativo. Lo siguiente, es el programa emsamblador CHANGE.

```
OP CODE    ETIQUETAS   NEMÓNICOS       COMENTARIOS

                       CHANGE.ASM
3A 7A 1B               LD A,(1B7AH) ; A ← x
C6 00                  ADD A,00H    ; x ← x + 0
28 21                  JR Z,CERO    ; Si x = 0, →CERO
CB 27                  SLA A        ;
38 25                  JR C,DATNEG  ; Signo (-) ir a DATNEG
06 07                  LD B,07H     ; Lo volvemos de entero a
CB 27        DATPOS    SLA A        ; fracción,
38 02                  JR C,NONORM  ; y normalizo.
10 FA                  DJNZ DATPOS  ;
CB 1F        NONORM    RR A         ; Dato normalizado.
4F                     LD C,A       ; C ← x[parte MSB]
21 00 00               LD HL,0000H  ;
CB 38                  SRL B        ; Espacio al signo(+)
CB 19                  RR C         ;
CB F0                  SET 6,B      ;
22 70 1B     WR        LD (1B70H),HL; (OP1) ← BCHL.
ED 43 72 1B            LD (1B72H),BC;
```

```
OP CODE    ETIQUETAS    NEMÓNICOS       COMENTARIOS
18 25                   JR ENDCHG       ; Termina operación.
21 00 00   CERO         LD HL,0000H     ; Si dato de 8 bits fue 0,
01 00 00                LD BC,0000H     ; (OP1) ← 0.
18 EF                   JR WR           ; Puente.
CB 1F      DATNEG       RR A            ; Le devuelvo su signo(-)
2F                      CPL             ; Sacamos el complemento
C6 01                   ADD A,01H       ; a dos al dato.
06 07                   LD B,07H        ; Exponente.
CB 27      BOSS         SLA A           ; Lo volvemos de entero
38 02                   JR C,NONOR      ; a fracción normalizada.
10 FA                   DJNZ BOSS       ;
CB 1F      NONOR        RR A            ; Le devolvemos su MSB.
4F                      LD C,A          ; C ← x[parte MSB]
21 00 00                LD HL,0000H     ; bytes a 0.
CB 38                   SRL B           ; Espacio al signo(-)
CB 19                   RR C            ;
CB F0                   SET 6,B         ; sesgo del exponente.
CB F8                   SET 7,B         ; signo negativo.
18 D2                   JR WR           ; Vamos a cargar a (OP1).
C9         ENDCHG       RET             ; Termina Operación.
                        .END            ;
```

C.2 TURN.

El programa TURN realiza el proceso inverso de transformación de CHANGE, esto es, que realiza la transformación de un números en punto flotante de 32 bits según el estándar de la IEEE. Dado un dato de 4 bytes, cargamos en sólo dos bytes el número, un byte que contiene los bits más significativos de la fracción y otro conteniendo al exponente y al signo. Si el número es negativo, nos vamos a DATNEG, si es positivo, verificamos si es cero, sino no es cero, quitamos el sesgo al exponente y verificamos que este no exceda de 7 puesto que como máxima magnitud absoluta posible tenemos 128. Si el exponente es menor o igual a 7, recorremos a la derecha la fracción, decrementando en cada corrimiento al exponente hasta que sea 1; después de esto ya tenemos nuestro nuevo número de 8 bits, que bastara colocarlo en el registro (1B7AH). Cuando el dato es negativo, se le aplica el mismo procedimiento que si es positivo, salvo la excepción de que cuando se tiene el dato final, se le complementa a dos; termina así TURN. Lo siguiente es el programa emsamblador TURN.

```
OP CODE     ETIQUETAS  NEMONICOS          COMENTARIOS

                       TURN
ED 4B 72 1B            LD BC,(1B72H); Se toman signo,
                                   ; exponente y fracción.
CB 21                  SLA C       ; Se pregunta por signo.
```

```
OP CODE    ETIQUETAS   NEMÓNICOS        COMENTARIOS
CB 10                  RL B             ;
38 19                  JR C,DATNEG      ; Si es (-) ir a DATNEG.
78           DATPOS    LD A,B           ;
C6 80                  ADD A,80H        ; Quitar sesgo.
17                     RLA              ;
38 2E                  JR C,CERO        ; Verificar cero.
1F                     RRA              ;
47                     LD B,A           ;
3E 07                  LD A,07H         ; Verificar exponente
90                     SUB B            ; no pase de 7,si se pasa
38 2E                  JR C,OVF         ; ir a OVF.
47                     LD B,A           ;
04                     INC B            ; Recorrer fracción hasta
CB 39        AQUI      SRL C            ; que sea entero (exp=1).
10 FC                  DJNZ AQUI        ;
79                     LD A,C           ;
32 7A 1B               LD (1B7AH),A     ; Cargar resultado.
18 2A                  JR ENDTURN       ; Termina operación.
78           DATNEG    LD A,B           ; Si es negativo el dato.
C6 80                  ADD A,80H        ; Quitar sesgo.
17                     RLA              ;
38 15                  JR C,CERO        ; Verificar cero.
1F                     RRA              ;
47                     LD B,A           ;
```

```
OP CODE   ETIQUETAS   NEMÓNICOS        COMENTARIOS
3E 07                 LD A,07H      ; Verificar exponente
90                    SUB B         ; no pase de 7.
38 15                 JR C,OVF      ;
47                    LD B,A        ;
04                    INC B         ;
CB 39       AHI       SRL C         ; Recorrer fracción hasta
10 FC                 DJNZ AHI      ; que sea entero (exp=1).
79                    LD A,C        ;
ED 44                 NEG           ; Complementar en dos.
32 7A 1B              LD (1B7AH),A  ; Termina operación.
18 0F                 JR ENDTURN    ;
3E 00       CERO      LD A,00H      ; Si el dato es cero.
32 7A 1B              LD (1B7AH),A  ; cargar (1B7AH) ← 00H.
18 08                 JR ENDTURN    ; Termina operación.
3A 7B 1B    OVF       LD A,(1B7BH)  ; Si el exp > 7, encender
CB EF                 SET 5,A       ; bit de sobreflujo en la
32 7B 1B              LD (1B7BH),A  ; conversión.
C9          ENDTURN   .END          ;
```

Si se quisiera una correspondencia exacta entre números en punto flotante y valores reales tendríamos que escalarlos; v.g. si tuviéramos valores reales entre -5 volts y +5 volts y quisiéramos procesarlos como datos en punto flotante, haríamos lo siguiente: (ver figura C.1).

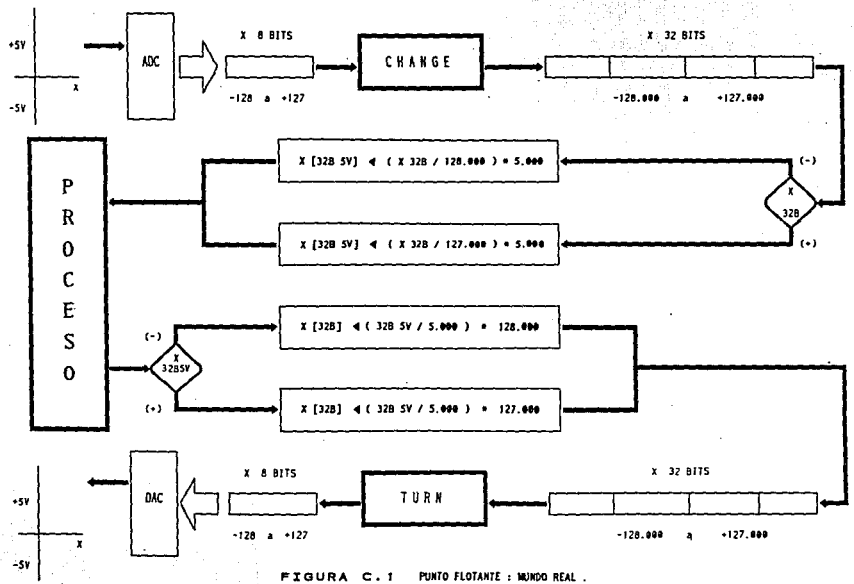


FIGURA C. 1 PUNTO FLOTANTE : MUNDO REAL .

BIBLIOGRAFÍA

Digital Computer Arithmetic
desing and implementation.
Joseph J.F. Cavanagh
McGraw Hill

Digital Integrated Electronics.
Taub / Shilling
McGraw Hill

Computer Systems Architecture.
M. Morris Mano
Prentice Hall

Computer Architecture and Organization.
Hayes
McGraw Hill

Lógica Digital y diseño de Computadores.
M. Morris Mano
Prentice Hall

Construya una Microcomputadora Basado en el Z80.
Steve Ciarcia
McGraw Hill

Programming The Z80.
Rodnay Zaks
SYBEX Computer Books

Microprocesador Z80.
Nichols / Nichols
Marcombo

Z80 Assembly Language Subrutines.
Leventhal / Salville
Osborne-McGraw Hill

Programación BASIC
Kemeny / kurtz
CECSA.