TEMA 4. OPERACIONES CON LOS DATOS

En este tema se realiza un estudio de las distintas operaciones básicas con los datos que un sistema computador pueden realizar.

Para ello, se parte del concepto de Unidad Operativa y la descripción de los elementos que la componen como son el circuito operador, el banco de registros y el registro de estado, del que algunos de sus biestables vienen definidos a partir de las operaciones realizadas por la Unidad Operativa.

A continuación, se repasan los distintos tipos de operaciones básicas que se pueden realizar clasificadas en operaciones lógicas, de desplazamiento y operaciones aritméticas. Dentro de las operaciones aritméticas, se analizarán técnicas de redondeo y se calculará el error cometido con cada una de las técnicas. Se verá cómo el signo y magnitud del error cometido con cada una de las técnicas de redondeo estudiadas dependen del rango del sistema en el que estemos trabajando (rango positivo o negativo) y de la resolución del mismo.

Finalmente, se presentan circuitos integrados empleados en el diseño de unidades operativas como son la ALU 74181, el procesador BIT-SLICE y los coprocesadores matemáticos. Para estos últimos, se comentarán tres posibles técnicas de conexión, indicando sus ventajas y desventajas.

Fuente de imagen: http://computersexplained.blogspot.com.es/2007/10/computers-understanding-arithmetic_14.html

4.1. INTRODUCCIÓN. UNIDAD OPERATIVA

La *Unidad Aritmética y Lógica* (*ALU*), también denominada *Unidad Operativa*, es la encargada de realizar en el computador las operaciones con los datos, de acuerdo con el programa en curso.

En un computador, una *operación* (aritmética, lógica, de desplazamiento, …) se puede realizar de las siguientes formas:

a) Mediante un CIRCUITO COMBINACIONAL. En este caso, la operación consta de una sola fase.
b) Mediante un CIRCUITO SECUENCIAL que genera sus propias fases. Es decir, el circuito que define el operador tiene su propia unidad de control.
c) Mediante un CIRCUITO SECUENCIAL cuyas fases las genera la *Unidad de Control* de la Unidad Central de Proceso.
d) Mediante un PROGRAMA. En este caso el usuario tiene que encargarse de definir, mediante un programa, la secuencia de operaciones a realizar por la Unidad Central de Proceso de nuestro sistema computador.

La estructura básica de una Unidad Operativa consta de los elementos mostrados en la Figura 4.1. El circuito secuenciador al que se hace referencia en dicha figura puede existir o no dependiendo de cómo se hagan las operaciones, según se ha mostrado más arriba.

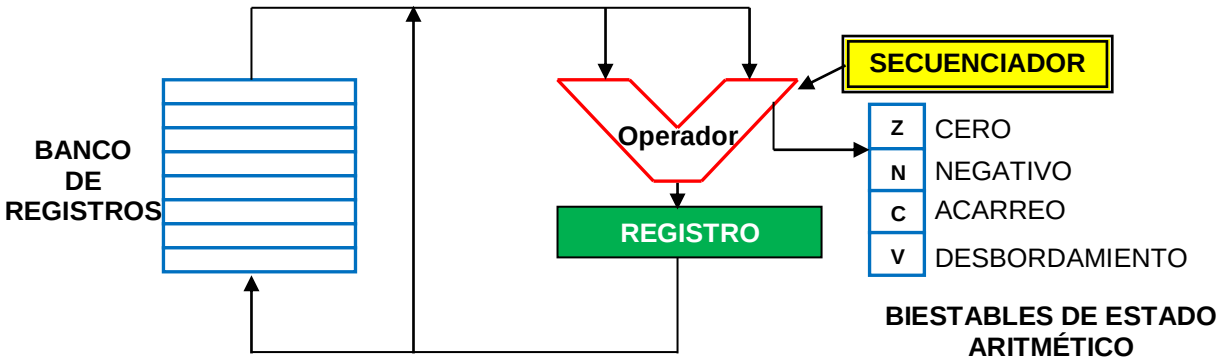


Figura. 4.1. *Estructura de la Unidad Aritmético-Lógica.*

Por OPERADOR entendemos todo circuito electrónico capaz de realizar una operación aritmética o lógica. Se pueden establecer distintas clasificaciones de operadores atendiendo a distintos criterios:

- Según el ***ámbito de aplicación***:
  - Operador general. Realizan varias clases de operaciones.
  - Operador especializado. Una única clase de operación.

- Según su ***realización***:
  - Operador combinacional. La operación se realiza en una única fase. Se construye con un circuito combinacional.
  - Operador secuencial. La operación se realiza según una secuencia de fases. Se construyen con circuitos secuenciales (necesitan elementos de memoria).
- Según el ***número de operandos***:
  - Operador monádico. Solamente emplean un operando.
  - Operador diádico. Requieren dos operandos.

    También se construyen operadores para múltiples sumas (admiten más de dos operandos); para obtener mayor velocidad.
- Según el ***paralelismo*** empleado:
  - Operador serie o de dígito. Trabajan dígito a dígito.
  - Operador paralelo o de vector. Realizan la operación sobre todos los dígitos de los operandos al mismo tiempo.
- Según la ***operación***:
  - Operador de desplazamiento. (Lógicos; Circulares; Aritméticos).
  - Operador lógico (NOT; AND; OR; XOR).
  - Operador aritmético (Negación; Suma; Resta; Multiplicación; División).
- Según la ***tecnología***:

  Los operadores se pueden construir empleando distintas tecnologías, teniendo cada una de ellas tiempos de retardo diferentes. De manera general, los tiempos de retardo de las distintas puertas lógicas se pueden poner en función del parámetro "**a**" de la siguiente forma:

PUERTAS NAND, NOR y NOT	→ a
PUERTAS AND y OR	→ 2a
PUERTAS XOR y XNOR	→ 3a

  El parámetro "a" según las distintas tecnologías tiene los siguientes valores:

  - Tecnología BIPOLAR:

TTL ESTÁNDAR	10 ns
TTL SCHOTTKY	1 ns
ECL	200 ps

  - Tecnología MOS:

CMOS	1 ns

  - Tecnología Ga As: 100 ps

Las operaciones típicas que realizan las distintas Unidades Operativas, según cuatro categorías de máquinas se muestran en la siguiente tabla:

	Potencia de cálculo			
	Mínima	**Baja**	**Media**	**Alta**
Suma/resta en binario	C	C	C	C
Suma/resta en coma flotante	P/A	P/UC	UC	S
Suma/resta en BCD	P	P	UC	S/UC
Multiplicación en binario	P/A	P/UC	UC	C
Multiplicación en coma flotante	P/A	P/UC	UC	S
División en binario	P/A	P/UC	UC	S
División en coma flotante	P/A	P/UC	UC	S
Operaciones lógicas	C	C	C	C
Desplazamiento unitario	C	C	C	C
Desplazamiento múltiple	P	P/UC	UC	C

C: Mediante Unidad Combinacional Específica
S: Mediante Unidad Secuencial Específica
UC: Gobernada mediante la Unidad de Control del Computador
P: Por programa
A: Mediante coprocesador aritmético opcional

Durante los años 90 se ha ido dotando a los microprocesadores de mayor número de operadores cada vez más potentes. Estos tienen tanto *unidades de coma fija* como *unidades de coma flotante*. En las Figuras 4.2 y 4.3 se muestran los diagramas esquemáticos de las dos unidades. En la Figura 4.4 se muestran, de forma esquemática, los operadores de un microprocesador potente comercial.

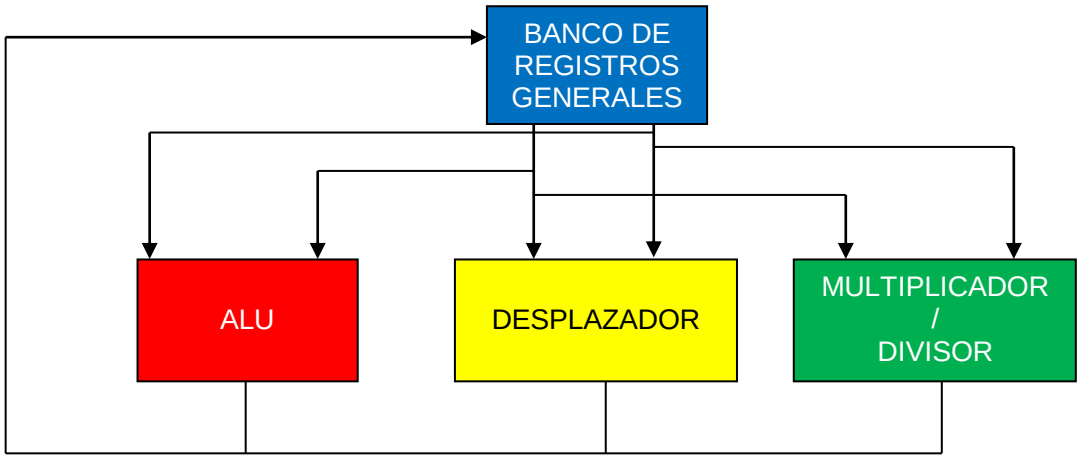


Figura. 4.2. *Unidades de coma fija.*

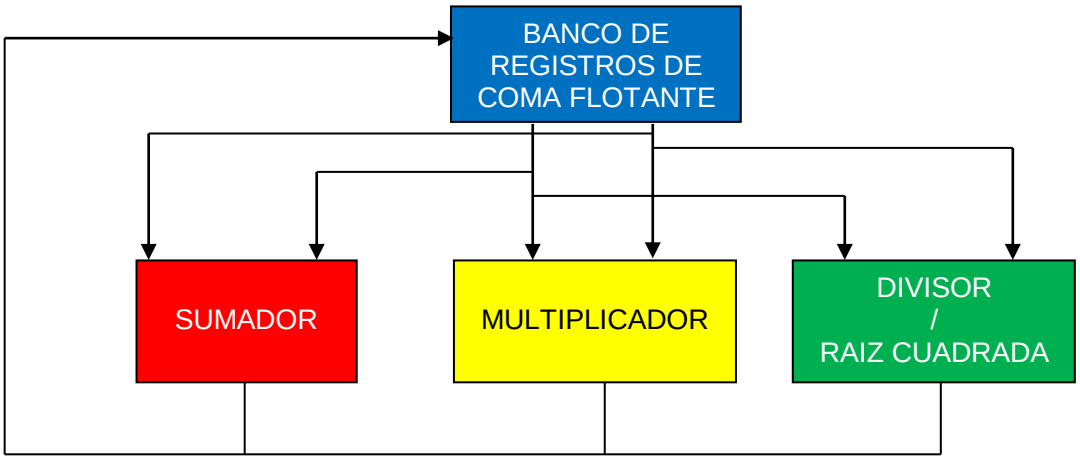


Figura. 4.3. *Unidades de coma flotante.*

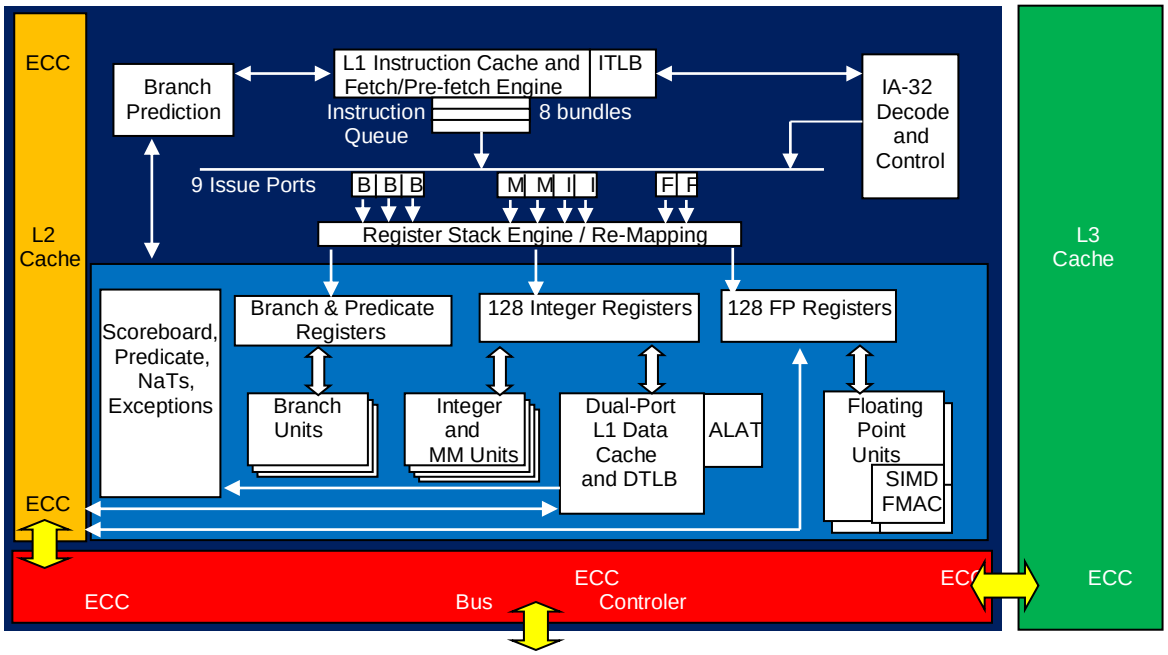


Figura. 4.4. *Operadores de un microprocesador potente.*

4.2. OPERACIONES DE DESPLAZAMIENTOS Y OPERACIONES LÓGICAS

Las ***operaciones de desplazamiento*** trasladan los bits de una palabra hacia la izquierda o derecha.

$$A\ (a_{n-1} \ldots\ldots\ldots a_1\ a_0) \longrightarrow D\ (d_{n-1} \ldots\ldots\ldots\ldots d_1\ d_0)$$

De forma general, para cualquier operación de desplazamiento se cumple: $\mathbf{d_{i+k} = a_i}$ **;** i = 0, 1, …, n-1.

k = n º de posiciones del desplazamiento

Si k es positivo → Desplazamiento a izquierda

Si k es negativo → Desplazamiento a derecha

La construcción de los *circuitos desplazadores* se puede realizar por ejemplo mediante circuitos multiplexores.

La forma en que se tratan los extremos del desplazamiento, en los que no se puede aplicar directamente la expresión $d_{i+k}=a_i$, permite diferenciar tres tipos de desplazamientos:

- **DEPLAZAMIENTOS LÓGICOS** → Los valores extremos se completan con ceros (o también a veces con unos).
- **DESPLAZAMIENTOS CIRCULARES** → Los bits de origen que sobran por un lado se insertan por el otro.
- **DESPLAZAMIENTOS ARITMÉTICOS** → Consiste en una multiplicación o división por una potencia de dos. Debe hacerse de forma diferente según la representación numérica empleada (debe conservarse el signo).

  En general, en los desplazamientos aritméticos, el desplazamiento a izquierda puede producir DESBORDAMIENTO. La detección de este desbordamiento se hará de una forma o de otra dependiendo del sistema de representación empleado. En la figura 4.5 se muestra cómo detectar el desbordamiento en los sistemas de representación de signo-magnitud y de complemento (C-2 y C-1).

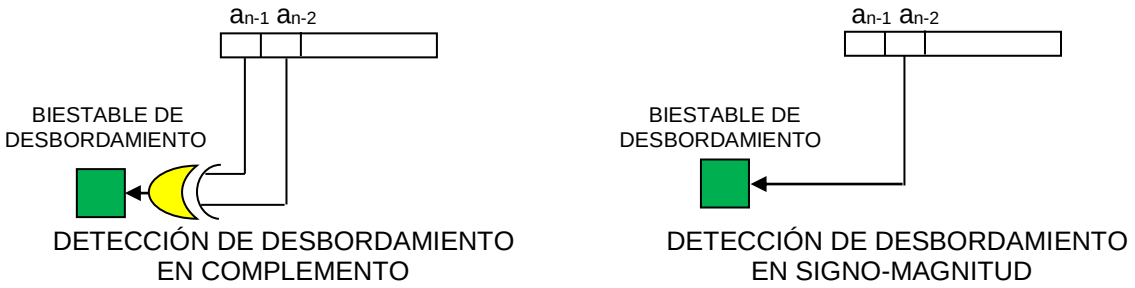
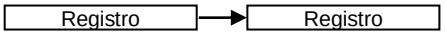
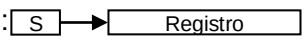


Figura. 4.5. *Detección de desbordamiento.*

- **DESPLAZAMIENTOS CONCATENADOS** → Afectan a un conjunto concatenado de dos o más elementos. Normalmente:

  - Dos registros concatenados: 

  - Un registro concatenado con el biestable de acarreo: 

  - Un registro concatenado con el biestable de signo: 

Las ***operaciones lógicas*** se aplican sobre bits independientemente. Generalmente, se encuentran en los computadores las siguientes operaciones lógicas:

* **NOT** → Negación.

* **OR** → Suma lógica.

* **AND** → Producto lógico.

* **XOR** → Suma lógica exclusiva.

4.3. OPERACIONES ARITMÉTICAS

En este apartado se estudiarán las operaciones de cambio y extensión de signo, de suma, de resta, de multiplicación, de división y las operaciones en precisión múltiple.

4.3.1. Operaciones de cambio de signo en los distintos sistemas de representación

La operación de cambio de signo se hará según el sistema de representación empleado:

- En **COMA FIJA SIN SIGNO**:

  No tiene sentido puesto que se trata de la representación de los números naturales (N).

- En **COMA FIJA CON SIGNO**:

  Simplemente hay que complementar o negar el bit de signo de la representación.

- En **COMA FIJA CON COMPLEMENTO A 2**:

  Hay que aplicar el complemento a 2.

  nº POSITIVO → $A \xrightarrow{\text{COMPLEMENTO A 2}} 2^n - A$

  nº NEGATIVO → $2^n - A \xrightarrow{\text{COMPLEMENTO A 2}} 2^n - (2^n - A) = A$

- En **COMA FIJA CON COMPLEMENTO A 1**:

  Hay que aplicar el complemento a 1.

  nº POSITIVO → $A \xrightarrow{\text{COMPLEMENTO A 1}} 2^n - 1 - A$

  nº NEGATIVO → $2^n - 1 - A \xrightarrow{\text{COMPLEMENTO A 1}} 2^n - 1 - (2^n - 1 - A) = A$

- En **COMA FIJA EN EXCESO Z**:

  Para cualquier exceso no resulta evidente. Sin embargo, si $Z = 2^{n-1}$, el cambio de signo consistirá en:

  1º- Cambiar el bit más significativo (0 a 1 ó 1 a 0)

  2º- Aplicar el complemento a 2.

  3º- Cambiar el bit más significativo (0 a 1 ó 1 a 0)

- En **COMA FLOTANTE**:

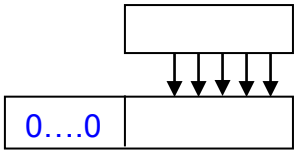
  Hay que cambiar el signo de la mantisa con cualquiera de los métodos anteriores (según la representación empleada para la mantisa).

4.3.2. Operación de extensión de signo en los diferentes sistemas de representación

La **operación de extensión de signo** consiste en: dada una representación con formato de n bits, pasar o otra, que siga teniendo el mismo valor que la anterior, con formato de m bits; siendo m > n. Según los distintos sistemas de representación, esta operación se realiza de la siguiente forma:

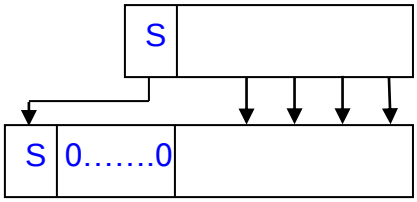
- **COMA FIJA SIN SIGNO**:

  Rellenar posiciones sobrantes a la izquierda con ceros.



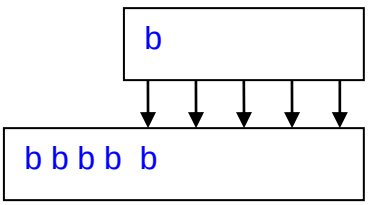
- **COMA FIJA CON SIGNO**:

  Mismo bit de signo y rellenar posiciones sobrantes a la izquierda del valor absoluto del número con ceros.



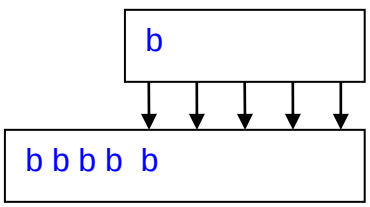
- **COMA FIJA CON COMPLEMENTO A 2**:

  Rellenar posiciones sobrantes a la izquierda con el valor del bit más significativo.



- **COMA FIJA CON COMPLEMENTO A 1**:

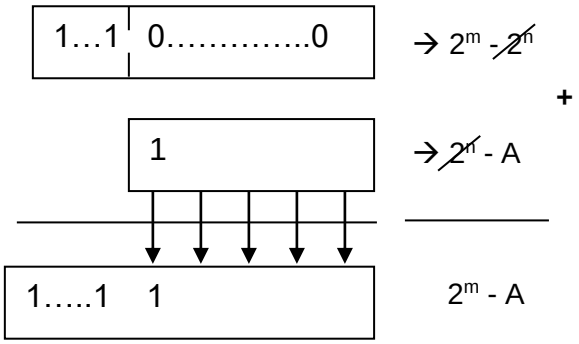
  Rellenar las posiciones sobrantes a la izquierda con el valor del bit más significativo.



JUSTIFICACIÓN DE LA EXTENSIÓN DE SIGNO EN COMPLEMENTO (Nos NEGATIVOS)

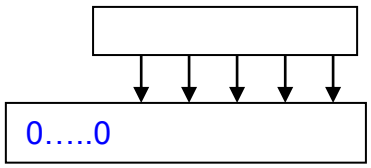
		Complemento a 2	Complemento a 1
n	1	→ 2^n - A	2^n – 1 - A
m		→ 2^m - A	2^m – 1 - A

$2^m - A + 2^n - 2^n = (2^m - 2^n) + (2^n - A)$ $\quad$ $2^m - 1 - A + 2^n - 2^n = (2^m - 2^n) + (2^n - 1 - A)$



- **COMA FIJA EN EXCESO Z**:

  Considerando el mismo exceso Z para los dos sistemas, rellenar posiciones sobrantes a la izquierda con ceros.



  Si por el contrario, al pasar de un formato con n bits a otro con m bits se pasa de exceso $Z = 2^{n-1}$ a exceso $Z' = 2^{m-1}$, hay que hacer $d_{m-1} = a_{n-1}$ y $d_i = \overline{a}_{n-1}$ para i = n-1, n, n+1, …, m-2. Esto viene justificado por la relación que existe entre los números en formato de coma fija en complemento a dos y los de coma fija en exceso 2^{n-1}.

- **COMA FLOTANTE**:

  Hay que realizar la extensión del signo independientemente para la mantisa y para el exponente, y según los sistemas de representación empleados respectivamente.

4.3.3. Operación de suma y resta

La operación de suma es el pilar sobre el que se construyen las unidades aritméticas de la mayoría de los computadores. El estudio de la suma y resta de enteros en los distintos sistemas de representación se ha venido realizando a lo largo del curso pasado, no siendo necesario volverla a tratar en éste. A la única suma y resta que se hará referencia en este tema, por no haberse tratado anteriormente, es a la suma en coma flotante.

En las Figuras 4.6, 4.7, 4.8 y 4.9 se muestran circuitos representativos de los operadores capaces de realizar las típicas operaciones aritméticas, lógicas y de desplazamiento.

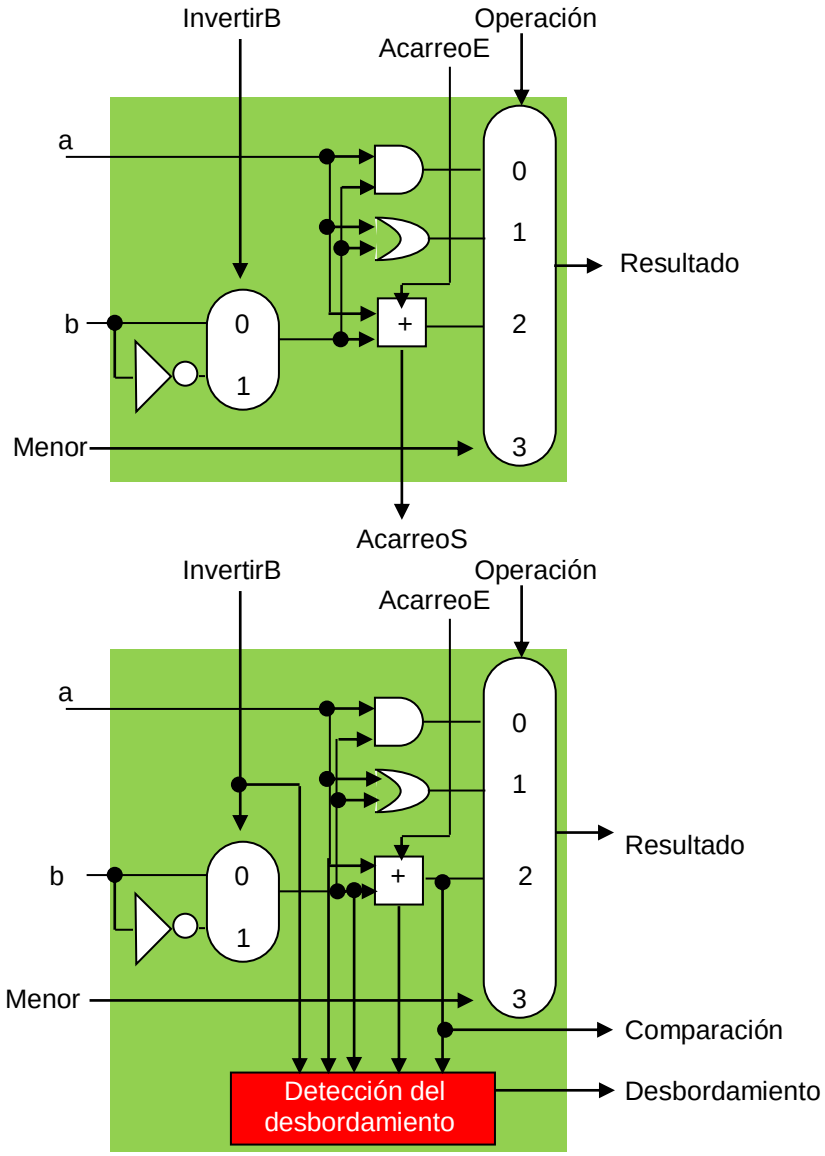


Fig. 4.6. *(Parte superior) ALU de 1 bit que realiza las operaciones AND, OR y suma de a y b o de $\overline{a}$ y b. (Parte inferior) ALU de 1 bit para el bit más significativo. El dibujo superior incluye la entrada directa que se conecta de forma que calcule la operación set on less than (véase Figura 4.7). El dibujo inferior tiene una salida directa desde el sumador para la comparación set on less than, llamada Comparación.*

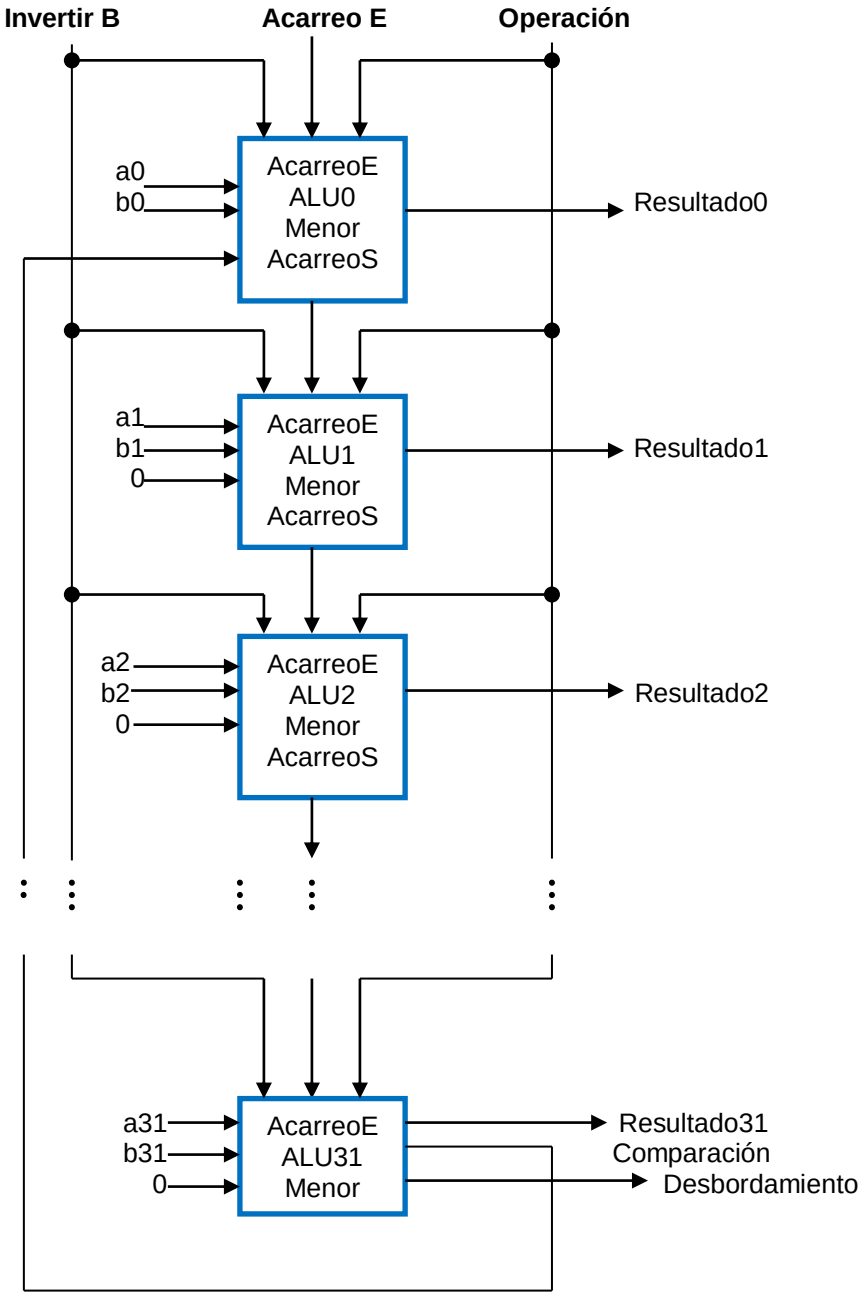


Figura 4.7. *ALU de 32 bits construida a partir de 31 copias de la ALU de 1 bit de la parte superior de la Figura 4.6 y una de la ALU de la parte inferior. Las entradas Menor se conectan a 0 excepto la del bit menos significativo, que se conecta a la salida Comparación del más significativo. Si la ALU realiza a – b y se selecciona la entrada 3 en los multiplexores de la figura 3.6 entonces Resultado = 0…001 si a < b y Resultado = 0…000 en cualquier otro caso.*

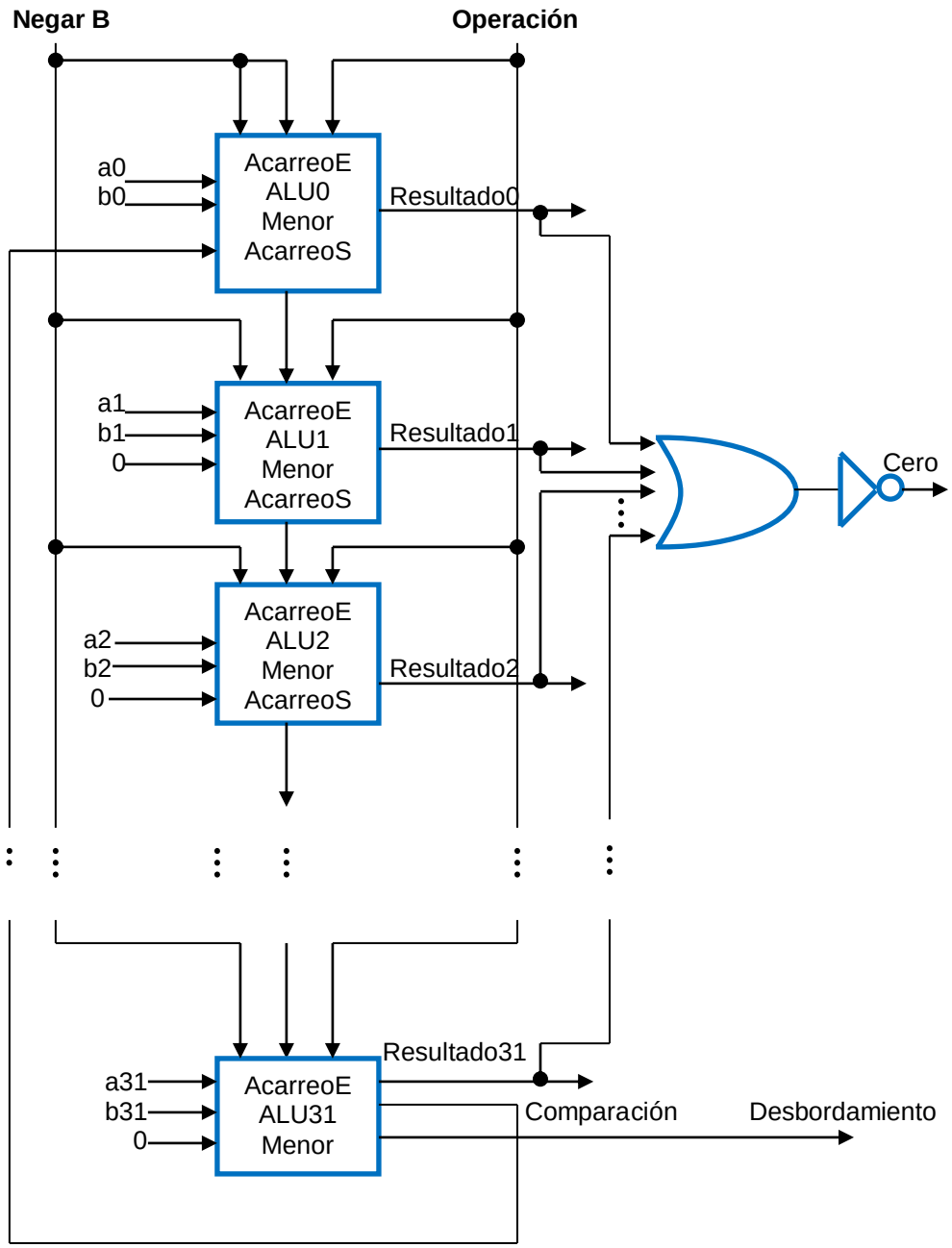


Figura 4.8. *La ALU de 32 bits resultante. Aquí se añade un detector de cero, respecto a la Figura 4.7.*

Líneas de control de la ALU	Función
000	and
001	or
010	suma
110	resta
111	*set on less than*

Figura 4.9. *Los valores de las tres líneas de control de la ALU, negarB y Operación, y las operaciones de la ALU correspondientes.*

En la Figura 4.10 se muestra el símbolo usado habitualmente para representar una ALU.

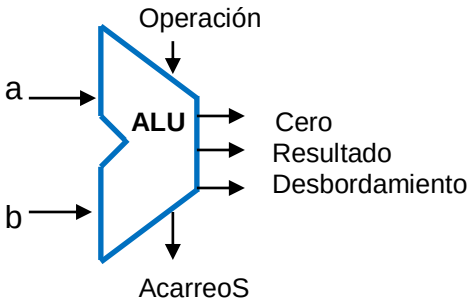


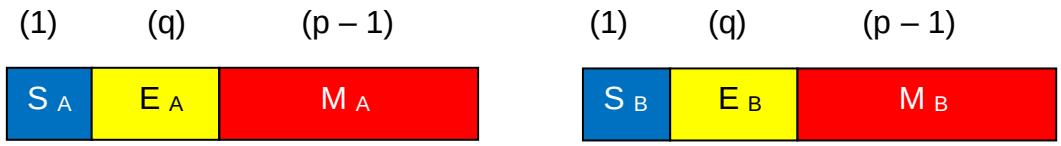
Figura. 4.10. *El símbolo usado normalmente para representar una ALU, como la de la Figura 4.8. Este símbolo se usa también para representar un sumador, etiquetándolo como ALU o como sumador, según el caso.*

- **SUMA Y RESTA EN COMA FLOTANTE:**

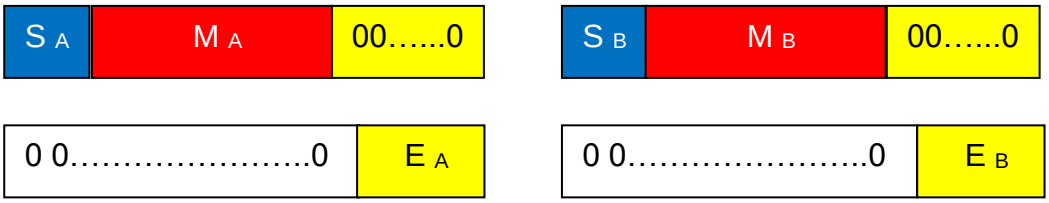
En el formato de coma flotante, los n bits de los operandos se dividen en **p** bits para la mantisa M y **q** bits para el exponente E. Se considera que la base del exponente es **r**. Supondremos que las mantisas están *normalizadas* y representación *signo-magnitud*, y el exponente viene dado en notación de *exceso*, por ser lo más frecuente. Dados dos operandos A y B, la operación de suma o resta requiere realizar los pasos que se indican a continuación.

$A \rightarrow M_A\, r^{E_A}$

$B \rightarrow M_B\, r^{E_B}$



1 er PASO → Separación de los exponentes y las mantisas. Este paso implica una operación de *extensión de signo* tanto para las mantisas como para los exponentes. Se necesitan cuatro registros de n bits que albergarán las siguientes informaciones:



2 º PASO → Resta de los exponentes. Con esta operación se obtienen las siguientes informaciones:

- ***Exponente Mayor → Exponente previo del resultado***.
- ***Nº de desplazamientos a realizar***.
- ***Mantisa a desplazar***. Esta mantisa será la de menor exponente.

3er PASO → Alineación de mantisas → Se desplaza a la derecha la mantisa con menor exponente, tantos dígitos como indique el resultado de la resta de exponentes. En esta operación se pierden dígitos significativos del operando menor; se perderán más o menos dígitos significativos dependiendo del número de desplazamientos a realizar. Esta pérdida de dígitos significativos implica una *operación de redondeo*. Una forma de evitar la pérdida de dígitos significativos es utilizando lo que se denominan ***dígitos de guarda*** (dígitos que se añaden al ancho de la palabra de la Unidad Operativa para completar, en formato de coma flotante, la mantisa del resultado); si no se pierden dígitos significativos, no se hace necesaria la operación de redondeo.
Con la alineación de mantisas se consigue que las mantisas de A y B tengan los dígitos del mismo peso en posiciones idénticas.

4º PASO → Realización de la suma o resta de las mantisas. Esta operación, como se ha considerado que las mantisas vienen en representación signo-magnitud, implica determinar si se hace una suma o una resta en función del análisis de los signos. La obtención de resultados que excedan de los p bits originales es muy frecuente por estar normalizadas las mantisas; aunque, como se ha realizado extensión de signo, este desbordamiento, una vez normalizado (5º PASO), no se va a reflejar en el operador de suma y resta puesto que opera con n > p bits. Esta operación raramente puede implicar redondeo.

5º PASO → Normalización del resultado. Esta operación, además de *desplazar los dígitos de la mantisa* hasta que el primer dígito sea distinto de cero, implica *modificar el exponente*, reflejando los desplazamientos realizados en la mantisa. Hay que decrementar el exponente si los desplazamientos son a izquierda de los dígitos de la mantisa y hay que incrementar el exponente si los desplazamientos son a derecha. Como se indica, ell exponente se incrementa cuando en la suma/resta de las mantisas se obtenga desbordamiento; en este caso, hay que desplazar los dígitos de la mantisa hacia la derecha, teniéndose que redondear si fuera necesario y, como consecuencia, teniendo que volver a comprobar si la mantisa resultado está normalizada o no.

Si como consecuencia de la normalización de las mantisas se produce desbordamiento del exponente hay que activar el biestable de desbordamiento y provocar una EXCEPCIÓN.

En la Figura 4.11 se muestra el organigrama que muestra gráficamente los pasos que hay que realizar, descritos anteriormente, para la obtención del resultado de la suma/resta de dos cantidades en formato de coma flotante.

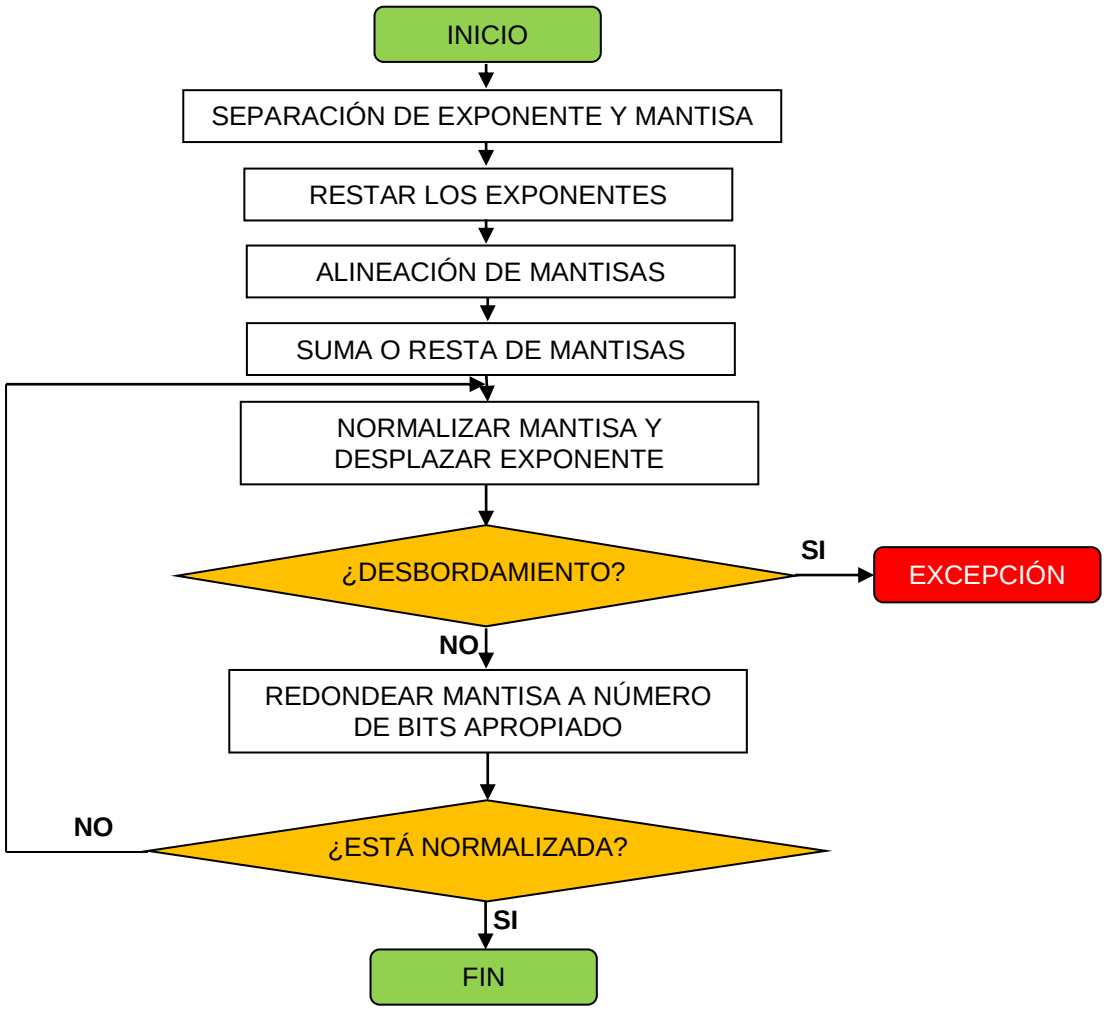


Figura. 4.11. *Organigrama representativo de la suma/resta en coma flotante.*

Aunque la operación de suma/resta en formato de coma flotante no es nada simple, los procesadores actuales de propósito general suelen incluir este tipo de operador. Estos operadores requieren, además de los registros para ir almacenando los datos, de los siguientes elementos:

- Un restador de exponentes, con un selector para el mayor.
- Un desplazador de mantisas, con una serie de multiplexores para seleccionar la mantisa a desplazar.
- Un sumador/restador de mantisas.

- Un normalizador, compuesto por un incrementador/decrementador de exponentes y un desplazador de mantisas.

En la Figura 4.12 se muestra el diagrama de bloques de una unidad operativa dedicada a la suma/resta en formato de coma flotante.

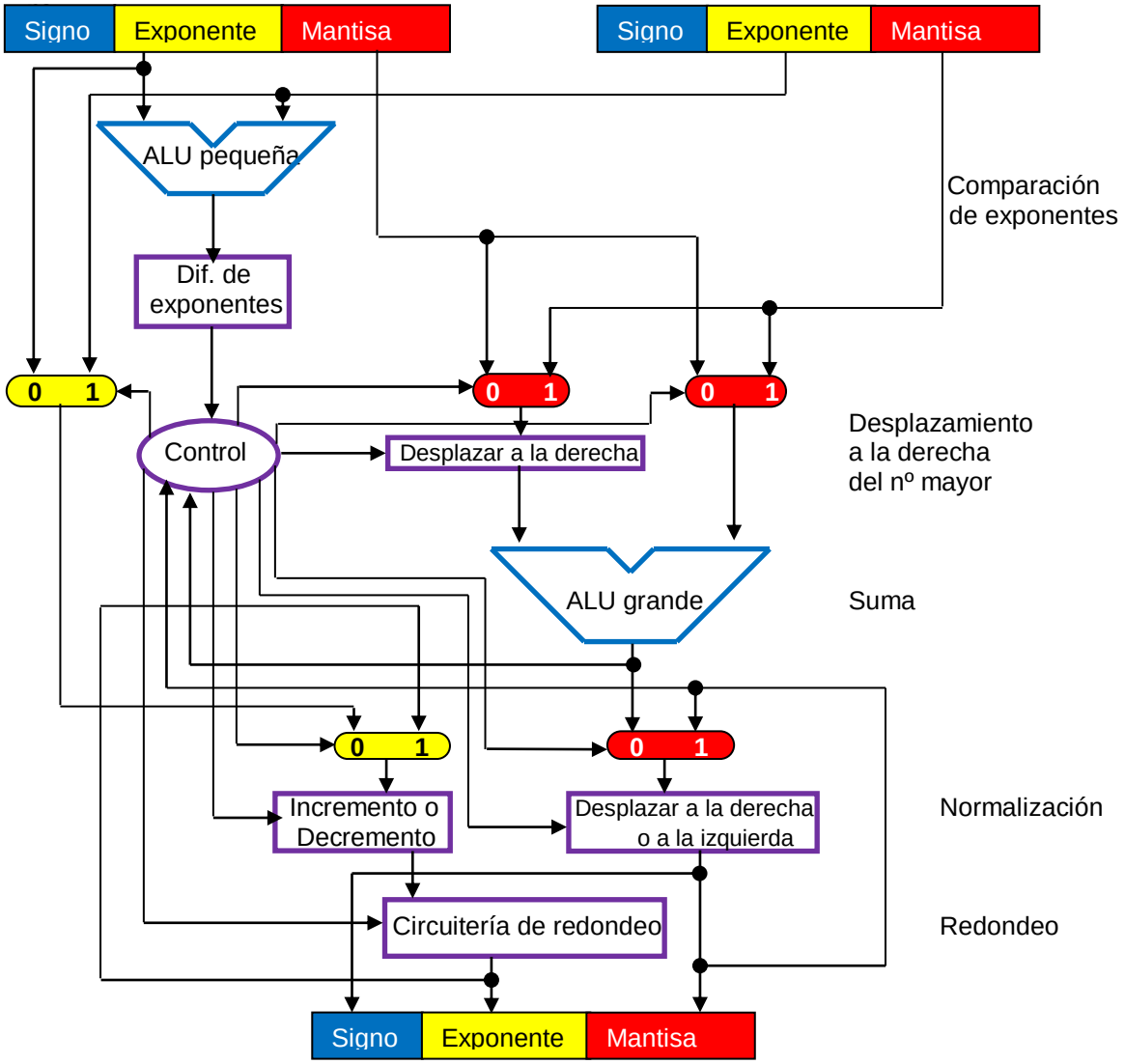


Figura. 4.12. *Diagrama de bloques de una unidad aritmética dedicada a la suma en coma flotante. Los pasos de la Figura 4.11 se corresponden con cada bloque, de arriba abajo. Primero se restan los exponentes en la ALU pequeña para determinar cuál es mayor y por cuánto. Esta diferencia controla los tres multiplexores: de izquierda a derecha, seleccionan el exponente mayor, la mantisa de menor exponente y la mantisa del mayor. La mantisa de menor exponente se desplaza a la derecha y se suman las mantisas en la ALU grande. El paso de normalización desplaza la suma a la izquierda o a la derecha e incrementa o decrementa el exponente. El último paso es el redondeo, que puede requerir una nueva normalización para obtener el resultado final.*

4.3.4. Operación de multiplicación

La operación se suele realizar utilizando un operador de suma/resta y un algoritmo adecuado. Solamente las máquinas muy potentes disponen de un operador combinacional para multiplicar.

En este apartado se describen algunos algoritmos para la realización de la operación de multiplicación en binario puro sin signo y en binario puro con signo (signo-magnitud).

- **Multiplicación binaria sin signo**

El algoritmo que se describe se denomina *algoritmo de suma desplazamiento*. Se considera que se desea multiplicar A · B, ambos números de n bits. El método consiste en observar el multiplicador B, bit a bit, y organizar una suma de n términos parciales que sean iguales a cero o al multiplicando A, dependiendo de que el correspondiente bit de B sea 0 ó 1. Estos términos parciales van desplazados a la izquierda un número de lugares igual al orden del correspondiente bit de B.

Llamando P_i a cada término parcial, se puede escribir que $P_i = A \cdot 2^i$ si $b_i = 1$ ó $P_i = 0$ si $b_i = 0$. De esta forma el producto resultado es $R = A \cdot B = P_{n-1} + P_{n-2} + \ldots + P_1 + P_0$. Si definimos la suma parcial S_i como $S_i = P_i + P_{i-1} + \ldots + P_1 + P_0$, se tiene que $R = S_{n-1}$, por lo que bastará con ir calculando la serie S_i.

La longitud máxima de estas sumas parciales S_i es de n+i+1, siendo la longitud del resultado final R de 2n bits como máximo.

Se pueden plantear distintos esquemas de circuitos capaces de realizar la operación de multiplicación; en las Figuras 4.13, 4.14, 4.15, 4.16, 4.17 y 4.18 se muestran tres versiones y los organigramas correspondientes para la realización de la operación.

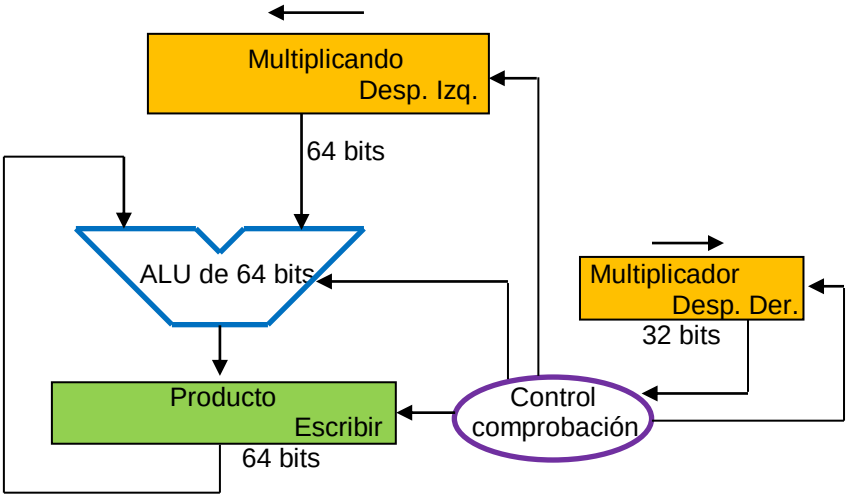


Figura. 4.13. *Primera versión de la circuitería de la multiplicación. El registro Multiplicando, la ALU y el registro Producto son todos de 64 bits. Sólo el registro Multiplicador contiene 32 bits. El multiplicando, de 32 bits, empieza en la mitad derecha del registro Multiplicando y se desplaza 1 bit a la izquierda en cada paso. El multiplicador se desplaza en la dirección contraria 1 bit en cada paso. El algoritmo empieza con el producto iniciado a 0. El control decide cuándo desplazar los registros Multiplicando y Multiplicador y cuándo escribir nuevos valores en el registro Producto.*

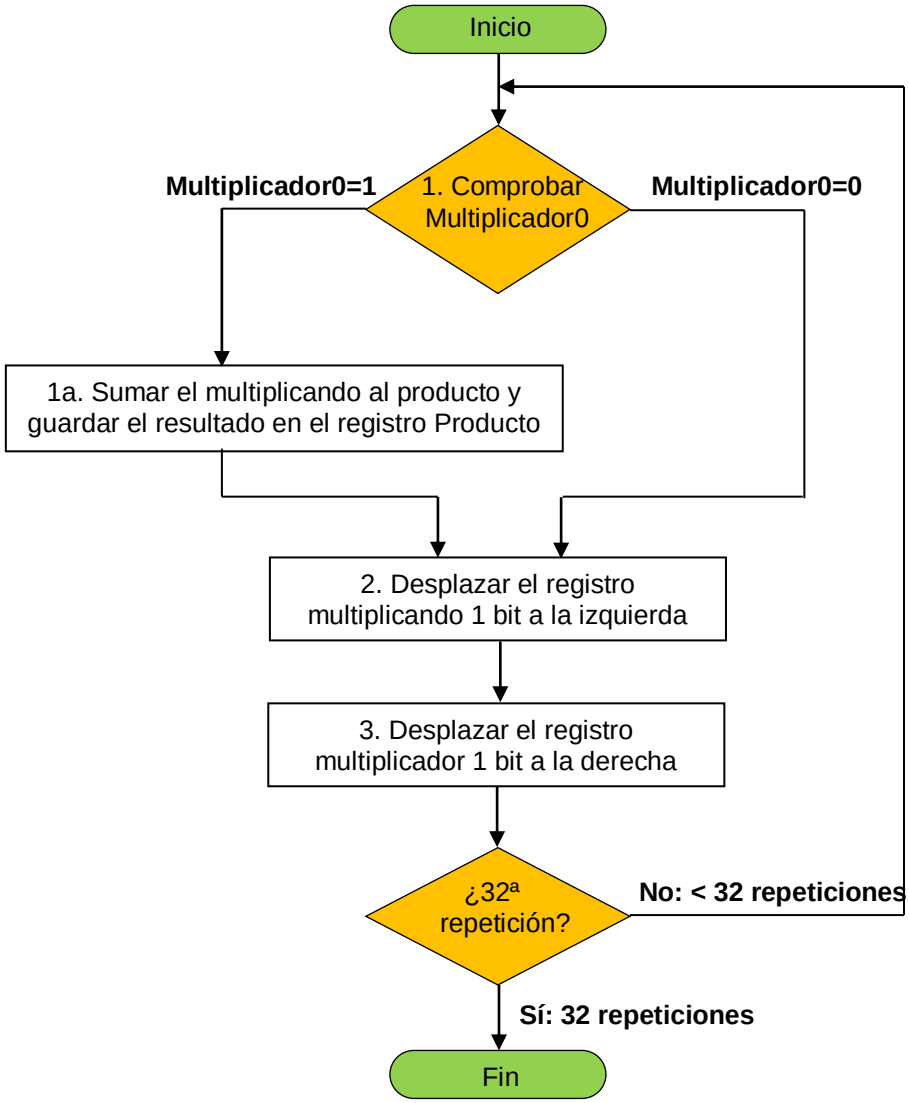


Figura. 4.14. *El primer algoritmo de multiplicación, que utiliza la circuitería de la figura 4.13. Si el bit menos significativo del multiplicador es 1, se suma el multiplicando al producto. Si no, se va al siguiente paso. En los siguientes dos pasos se desplaza el multiplicando a la izquierda y el multiplicador a la derecha. Estos tres pasos se repiten 32 veces.*

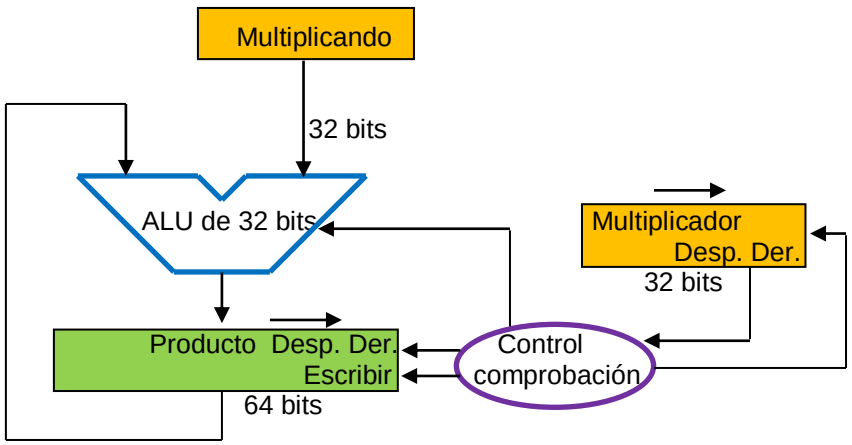


Figura. 4.15. *Segunda versión de la circuitería de la multiplicación. Compárese con la primera versión de la figura 3.13. El registro Multiplicando, la ALU y el registro Producto son todos de 32 bits. Sólo el registro Producto tiene 64 bits. Ahora el producto se desplaza a la derecha.*

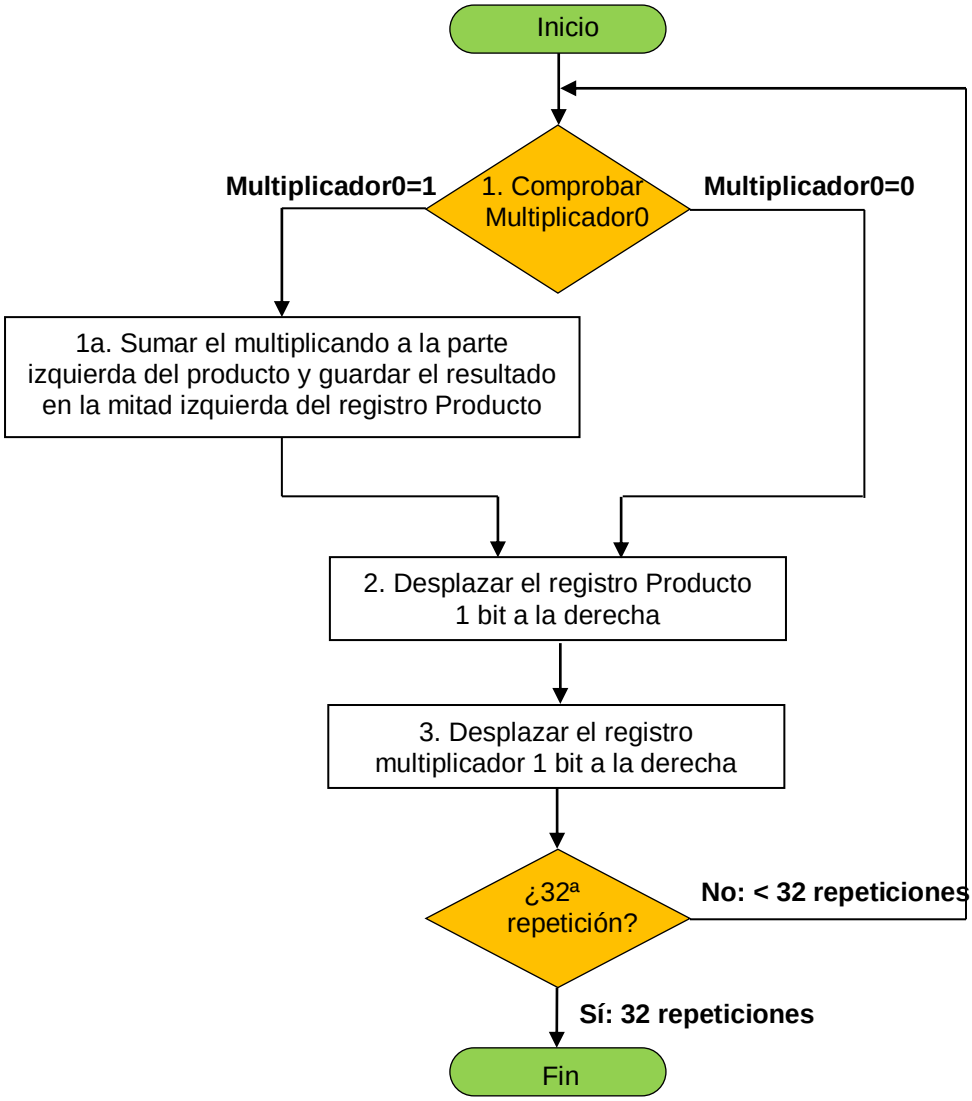


Figura. 4.16. *El segundo algoritmo de multiplicación, que utiliza la circuitería de la figura 3.15. En esta versión, el registro producto se desplaza a la derecha en lugar de desplazar el multiplicando.*

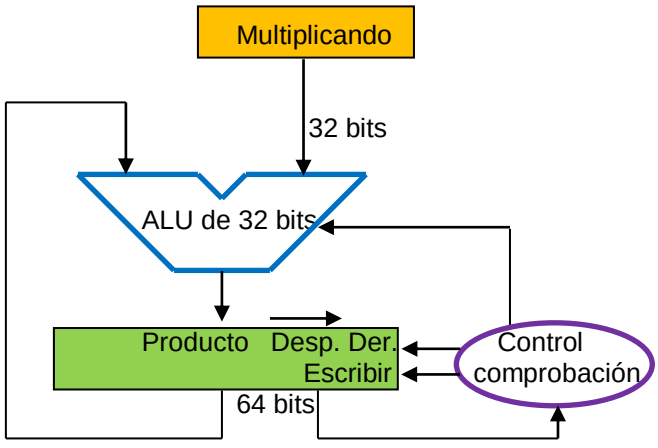


Figura. 4.17. *Tercera versión de la circuitería de la multiplicación. Comparando con la segunda versión de la figura 3.15, el registro Multiplicador separado ha desaparecido. El Multiplicador se sitúa en la mitad derecha del registro Producto.*

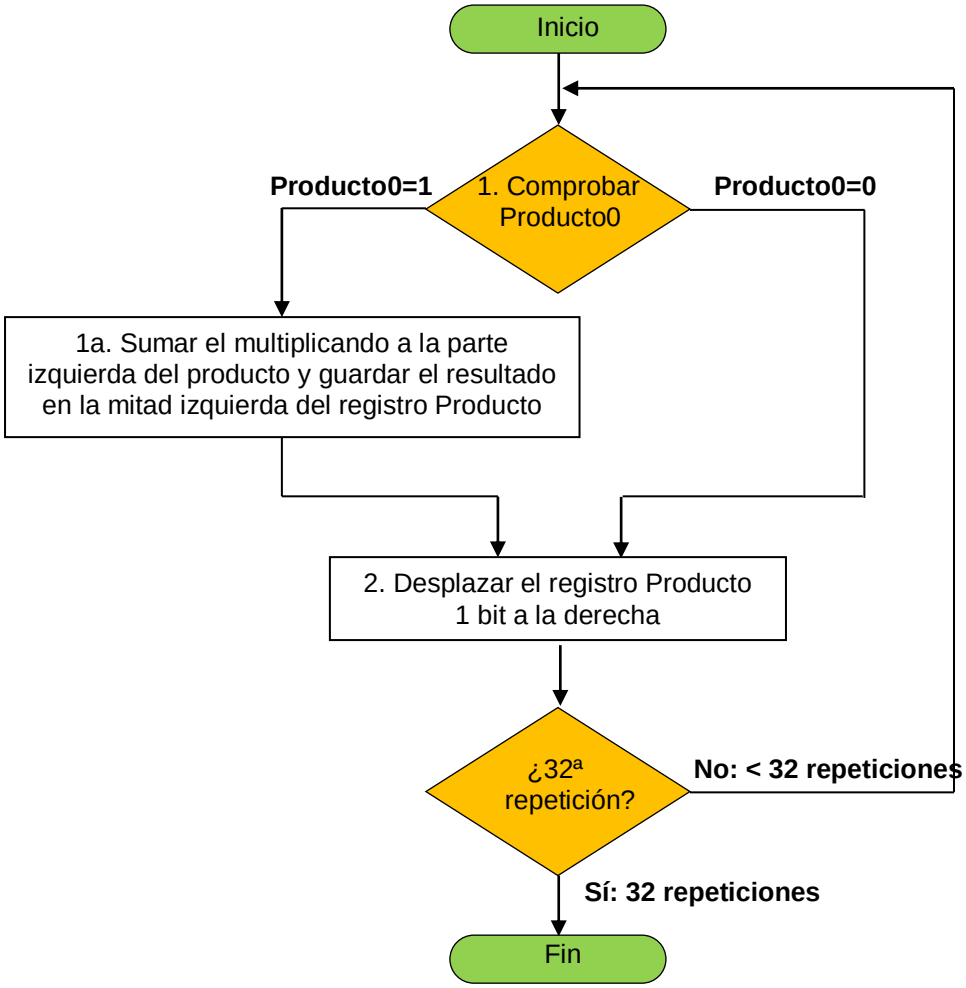


Figura. 4.18. *Tercer algoritmo de multiplicación. Necesita sólo dos pasos, ya que lo registros Producto y Multiplicador se han combinado en uno solo.*

- **Multiplicación binaria con signo**

  Pasos a realizar:

  - Análisis previo de los signos de los operandos, para determinar el signo del resultado.
  - Conversión a positivo de los operandos negativos.
  - Realización del producto mediante un procedimiento de multiplicación sin signo.
  - Cambio, en su caso, del signo de resultado.

Cuando los operandos están representados en signo-magnitud se simplifica mucho el proceso comentado. Si se intenta trabajar directamente con los números representados en complemento se deben tener en cuenta las siguientes consideraciones:

- Si el Multiplicando tiene un valor negativo: no representa ningún problema operar directamente en complemento ya que las sumas parciales se hacen

correctamente, obteniéndose el resultado negativo directamente en complemento, siempre que se hagan las correspondientes extensiones de signo en los productos parciales.

- Si el Multiplicador tiene una valor negativo: en este caso hay que realizar las siguientes correcciones:
  - Complemento a 2: El multiplicador tiene el siguiente valor:

$$B = -b_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} b_i \cdot 2^i$$

Implicando que todos los productos parciales son iguales al caso de multiplicador positivo con excepción del último producto parcial, correspondiente al bit b_{n-1}, que tiene signo negativo, lo que implica que hay que restar este producto parcial en vez de sumarlo.

  - Complemento a 1: El multiplicador tiene el siguiente valor:

$$B = -b_{n-1} \cdot (2^{n-1} - 1) + \sum_{i=0}^{n-2} b_i \cdot 2^i = [-b_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} b_i \cdot 2^i] + b_{n-1}$$

La expresión entre corchetes es la misma que en complemento a 2, por lo que se aplica lo dicho anteriormente. Adicionalmente está el término b_{n-1}, lo que implica una corrección consistente en sumar A. Aunque esta corrección corresponde al valor del bit multiplicador que se trata el último, lo más conveniente es hacerla al principio, puesto que en caso contrario habría que alinear el resultado parcial, que está desplazado a la izquierda, con el multiplicador A.

- **Multiplicación en coma flotante**

La multiplicación en coma flotante supone una suma de los exponentes y una multiplicación binaria de las mantisas. En la Figura 4.19 se muestra el organigrama correspondiente a la operación de multiplicación en formato de coma flotante.

4.3.5. Operación de división

Es algo más compleja que la multiplicación. Es raro que un computador disponga de un divisor combinacional, realizando la división normalmente mediante un operador sumador/restador y un algoritmo adecuado.

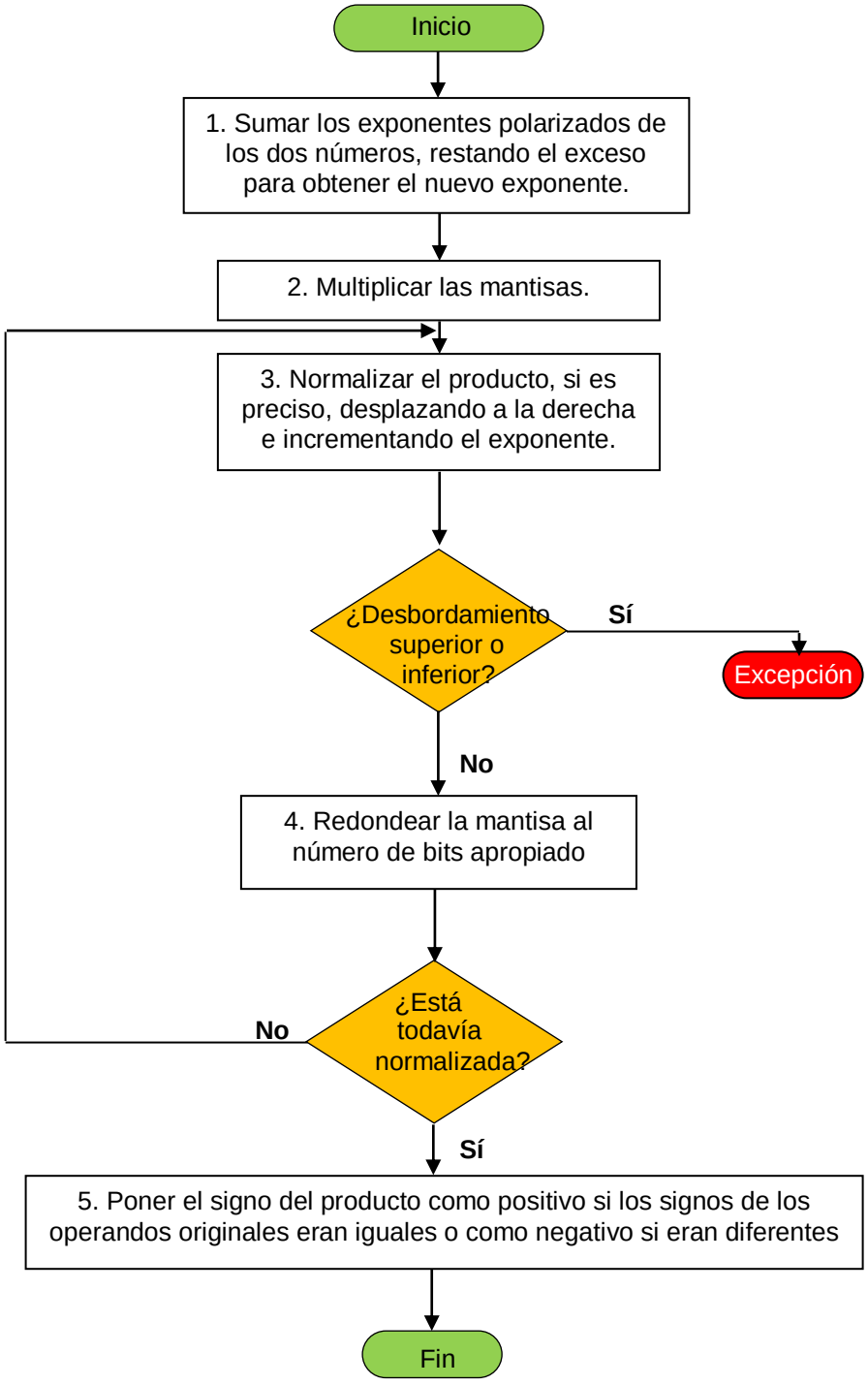


Figura. 4.19. *Multiplicación en coma flotante. El camino normal es ejecutar los pasos 3 y 4 una sola vez. Si el redondeo deja el resultado sin normalizar, se debe repetir el paso 3.*

- **División sin signo**

Al realizar la operación de división con el algoritmo de lápiz y papel vemos que, por realizarse la operación en binario, sólo existen dos alternativas: si el dividendo es mayor que el divisor, entonces “cabe a 1”, en caso contrario no cabe. En un computador, esta función puede realizarse mediante una resta (o mediante una comparación, que no es

más que una resta cuyo resultado no se almacena). Si el resultado de la resta es positivo, es que “cabe”, si es negativo, “no cabe”. Para realizar las restas correspondientes a los distintos dígitos del cociente, es necesario ir desplazando el divisor a la derecha. En el computador, para no perder dígitos significativos del divisor, se hace la operación inversa: se desplaza el dividendo residual a la izquierda. Esto es posible realizarlo sin producir desbordamiento porque las sucesivas restas van dejando ceros a su izquierda, que al desplazar son eliminados. Sin embargo, es necesario operar con n+1 bits para poder desplazar a la izquierda, cuando no ha cabido, sin perder bits significativos. Decir que la posición de la coma del cociente depende del número de desplazamientos efectuados en la normalización previa del divisor y del dividendo. Las Figuras 4.20, 4.21, 4.22, 4.23, 4.24 muestran diferentes organigramas para la realización de la operación de división.

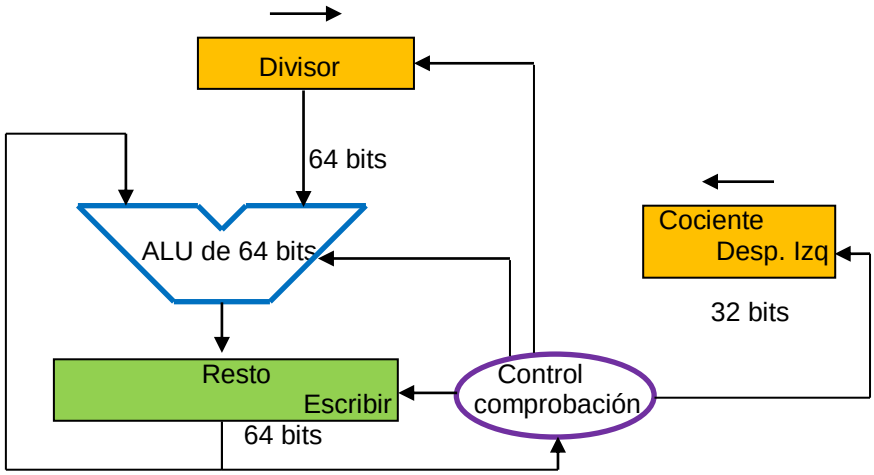


Figura. 4.20. *Primera versión de la circuitería de la división. El registro Divisor, la ALU y el registro Resto tienen todos 64 bits. Sólo el registro Cociente tiene 32 bits. El divisor, de 32 bits, empieza en la mitad izquierda del registro Divisor y se desplaza 1 bit a la derecha en cada paso. El registro Resto se inicia con el dividendo. El control decide cuándo se desplazan los registros Divisor y Cociente y cuándo se escribe el nuevo valor en el registro Resto.*

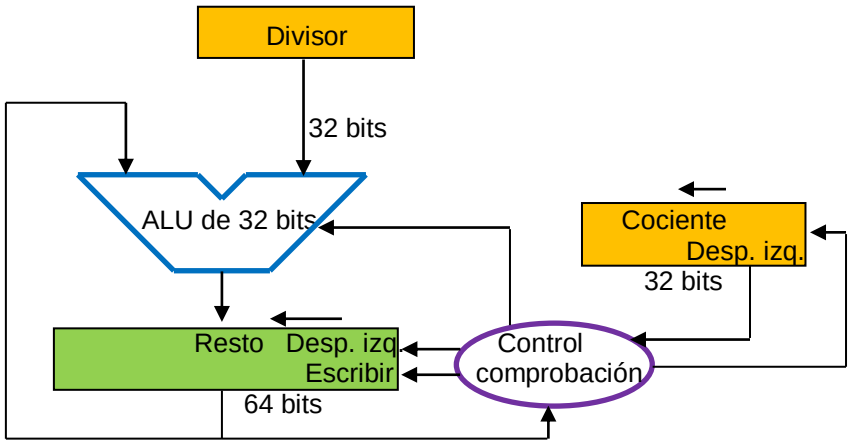


Figura. 4.21. *Segunda versión de la circuitería de división. El registro Divisor, la ALU y el registro Cociente tienen 32 bits: sólo el registro Resto continúa con 64 bits. Comparando con la figura 3.20, la ALU y el registro Divisor se han reducido a la mitad y el resto se desplaza a la derecha.*

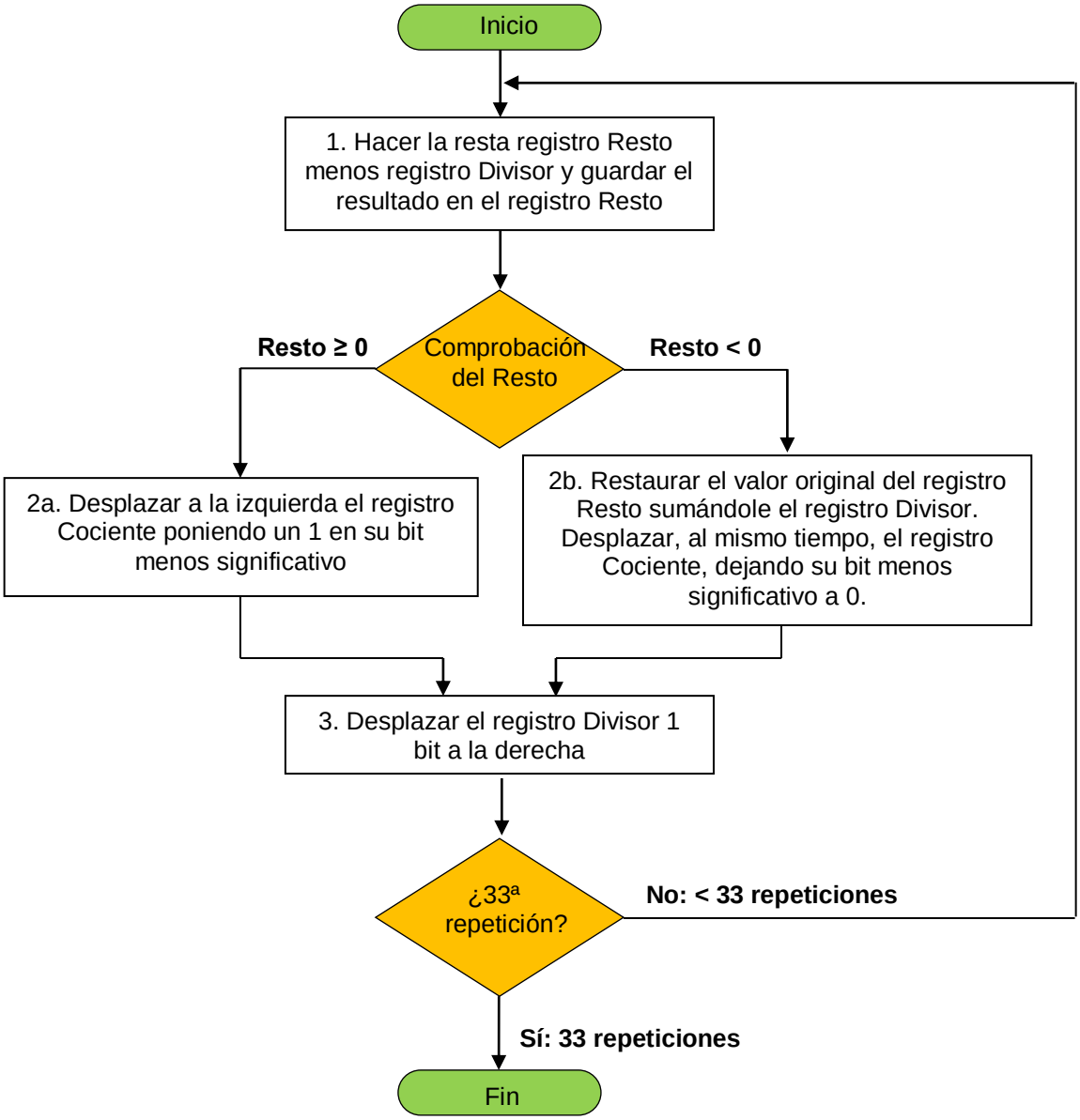


Figura. 4.22. *El primer algoritmo de la división, que utiliza la circuitería de la figura 4.20. Si el Resto es positivo, el divisor cabe en el dividendo, por lo que el paso 2a genera un 1 en el cociente. Un Resto negativo después del primer paso significa que el divisor no cabía en el dividendo, entonces el paso 2b genera un 0 en el cociente y suma el divisor al resto, deshaciendo la operación del paso 1. El desplazamiento final del paso 3, alinea el divisor adecuadamente, respecto al dividendo, para la próxima iteración. Estos pasos se repiten 33 veces, el motivo del paso aparentemente extra quedará claro en la siguiente versión del algoritmo.*

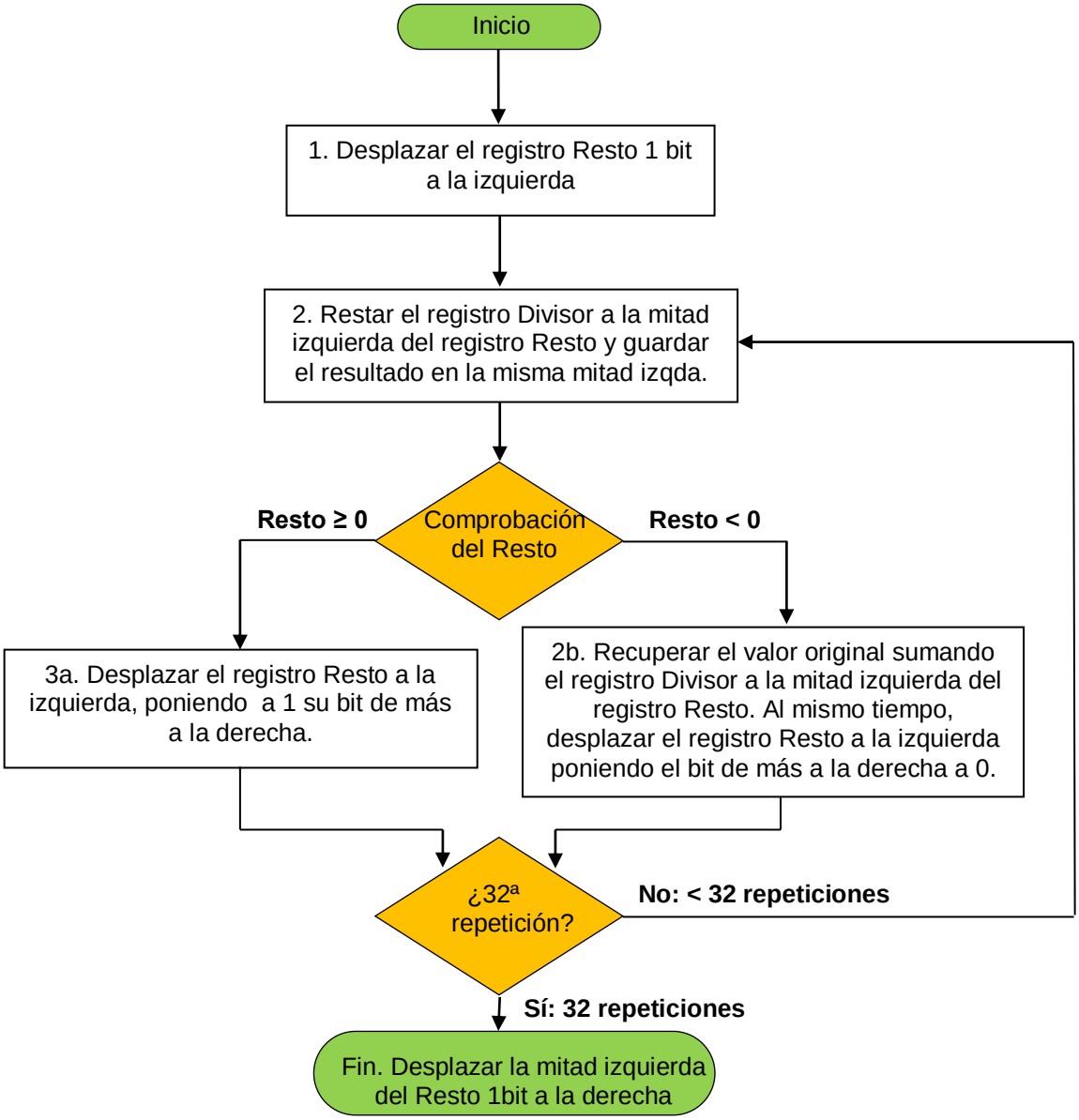


Figura. 4.23. *El segundo algoritmo de la división que utiliza el circuito de la figura 4.21. El registro Resto se desplaza a la izquierda.*

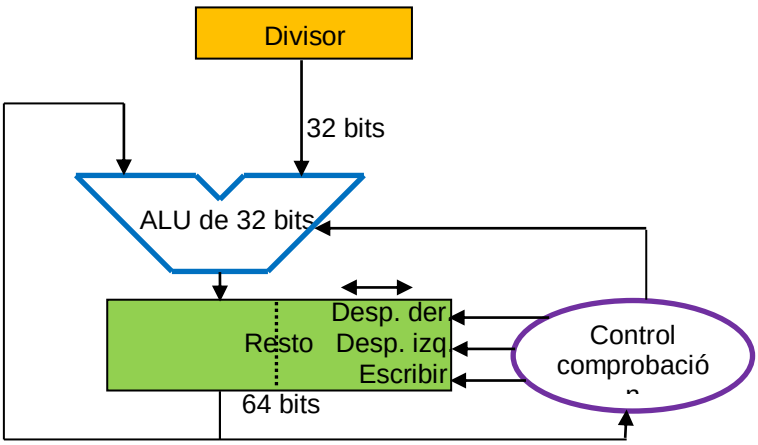


Figura. 4.24. *La tercera versión de la circuitería de la división. Esta versión sitúa el registro Cociente en la mitad izquierda del registro Resto.*

- **División con signo**.

Se muestra a continuación un algoritmo de división con signo considerando que los operandos vienen representados en el sistema de complemento a dos. Los números han de estar normalizados.

1º Inicialización: se pone a cero el contador de fases y periodos.

2º Se comparan los signos de dividendo y divisor. Si son iguales se pone un cero en el cociente y se hace la resta de dividendo menos el divisor, almacenando el resultado en el registro que alberga el dividendo. Si son distintos se pone un 1 en el cociente y se hace la suma del dividendo y el divisor, almacenando el resultado en el registro del dividendo.

3º Se comparan los signos del dividendo parcial y del divisor. En el caso de que el del dividendo parcial sea 0 se puede dar por terminada la división. Si, por el contrario, se desea seguir con el algoritmo, hay que considerar que el signo del dividendo parcial es el del dividendo inicial.

4º Se desplaza el dividendo un lugar a la izquierda.

5º Si la comparación de signo, del paso 2, dio igual, se pone un 1 en el cociente y se resta dividendo menos divisor, almacenando el resultado en el registro del dividendo.

6º Si la comparación de signo, del paso 2, dio distinto, se pone un 0 en el cociente y se suma el dividendo con el divisor, almacenando el resultado en el registro del dividendo.

7º Se incrementa el contador de fases. Si su valor es menor que n, se repite el ciclo a partir del paso 2. Si es igual a n, se termina con el paso 7.

8º Si el cociente es negativo, hay que corregirlo sumándole 000 … 0001.

- **División en coma flotante**.

La *división en coma flotante* supone una resta de los exponentes y una división binaria de las mantisas. En la Figura 4.25 se muestra el organigrama de la operación de división en formato de coma flotante.

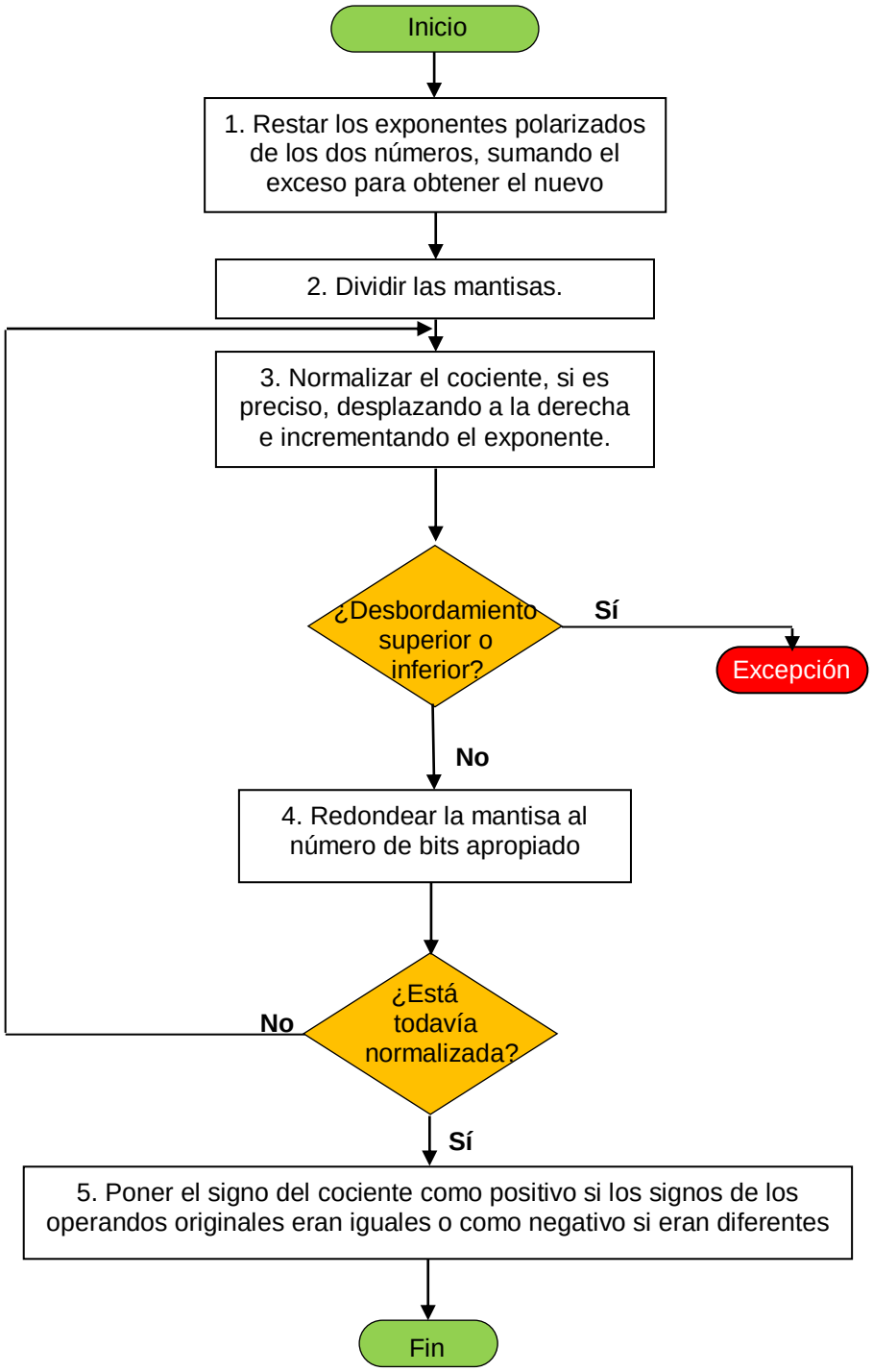


Figura. 4.25. *División en coma flotante. El camino normal es ejecutar los pasos 3 y 4 una sola vez. Si el redondeo deja el resultado sin normalizar, se debe repetir el paso 3.*

4.3.6. Operaciones en precisión múltiple

Se dice que un computador trabaja en precisión múltiple cuando los datos emplean para su representación un número α de bits mayor que el ancho de su palabra (n bits). Normalmente α = k · n con k natural.

La precisión múltiple presenta dos problemas:

1. Hay que almacenar cada dato en más de una posición de memoria o en más de un registro.
2. Hay que tratar los datos en trozos ya que la Unidad Aritmético–Lógica opera en paralelo con datos de n bits.

Supongamos que se va a realizar la operación de suma en doble precisión. El sumando A se supone que se encuentra almacenado en los registros A_1 y A_2, con los bits menos significativos en el registro A_1. De la misma forma, el sumando B se encuentra almacenado en los registros B_1 y B_2.

$$\left.\begin{array}{l} A \rightarrow A_2A_1 \\ B \rightarrow B_2B_1 \end{array}\right\} \quad A + B = A_2 \cdot 2^N + A_1 + B_2 \cdot 2^N + B_1 \rightarrow \left\{\begin{array}{l} A_1 + B_1 \\ A_2 + B_2 + C_{n-1} \end{array}\right\}$$

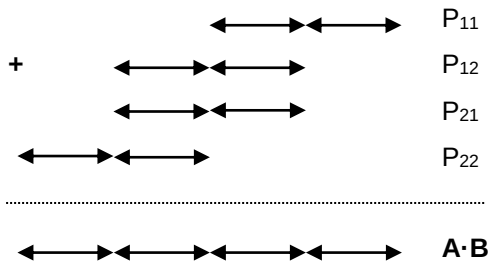
Las condiciones de SIGNO y DESBORDAMIENTO de toda la suma se obtienen de la última operación. Sin embargo, la condición de RESULTADO NULO exige tener en cuenta los resultados parciales de las dos operaciones.

Supongamos ahora que se va a realizar la operación de multiplicación de los dos datos A y B en doble precisión.

$$\left.\begin{array}{l} A \rightarrow A_2A_1 \\ B \rightarrow B_2B_1 \end{array}\right\} \quad A \cdot B = (A_2 \cdot 2^n + A_1) \cdot (B_2 \cdot 2^n + B_1) \rightarrow \text{Hay que realizar:}$$

$$\left\{\begin{array}{l} A_1 \cdot B_1 \rightarrow P_{11} \\ A_1 \cdot B_2 \rightarrow P_{12} \\ A_2 \cdot B_1 \rightarrow P_{21} \\ A_2 \cdot B_2 \rightarrow P_{22} \end{array}\right\} \quad A \cdot B = P_{11} + (P_{12} + P_{21})2^n + P_{22} \cdot 2^{2n}$$

Hay que tener en cuenta que cada producto parcial ocupa dos registros, por lo que hay que realizar las distintas sumas de la siguiente forma y teniendo en cuenta los acarreos correspondientes.



Los procedimientos de suma, resta y multiplicación son fácilmente extensibles al caso de precisión triple o mayor. En cambio, la operación de división exige que las restas y sumas parciales, se hagan en la precisión correspondiente.

Una dificultad que presentan las operaciones en precisión múltiple es que exigen poder introducir el acarreo inicial y corregir el acarreo de salida en el caso de resta. Por lo tanto, hay que realizar una modificación en el esquema del sumador-restador, como se muestra en la Figura 4.26.

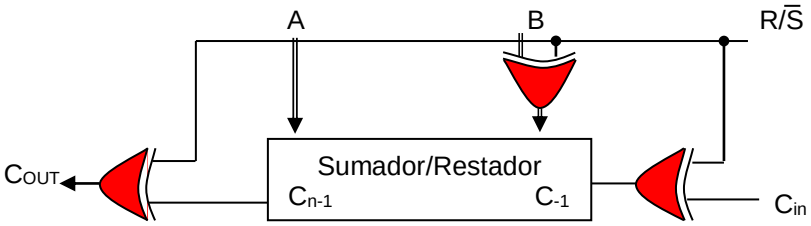


Figura. 4.26. *Sumador/Restador con acarreo de entrada.*

4.3.7. Biestables de estado aritmético

El objetivo fundamental de los biestables de estado aritméticos es guardar constancia de algunas particularidades del resultado de la última operación realizada por la Unidad Aritmética. Esta información se utilizará para tomar las acciones oportunas. (Se verá en el tema correspondiente a la UNIDAD DE CONTROL). Los más corrientes son:

- CERO
- SIGNO
- DESBORDAMIENTO
- ACARREO

La interpretación de los valores de los biestables de estado aritmético debe hacerse de acuerdo a la representación numérica empleada.

4.3.8. Técnicas de redondeo

Hay que aplicar alguna ***técnica de redondeo*** cuando como consecuencia de las operaciones se producen más bits de los que puede albergar el destino del resultado.

Sistema con las representaciones $A_0, A_1, A_2, \ldots\ldots A_r \ldots$ $\Rightarrow$
Cantidad a representar: **α** / $A_i > \alpha > A_{i-1}$

$\Rightarrow$ El objetivo de la operación de redondeo es seleccionar A_i ó A_{i-1}, como representación de α.

Las *técnicas de redondeo* más frecuentes son:

- **Truncamiento** → Consiste en eliminar los bits de la derecha que no caben en el formato de la representación. En definitiva, para cantidades positivas consiste en aproximar al número A_{i-1}, y para cantidades negativas se aproxima al A_i. Esta técnica tiene como ventaja el ser muy fácil de ejecutar; y tiene como inconveniente el hacer que el error introducido sea siempre por defecto, para cantidades positivas, o siempre por exceso para cantidades negativas. Luego se acumulan ERRORES POR DEFECTO, para cantidades exclusivamente positivas, y se acumulan ERRORES POR EXCESO para cantidades exclusivamente negativas. Luego, para cantidades positivas **- RES < ERROR ≤ 0**, y para cantidades negativas **0 ≤ ERROR < RES**.

- **Redondeo propiamente dicho o redondeo al más próximo** → Toma el valor más próximo al que se quiere representar; es decir:

  Si $|Ai-\alpha| > |A_{i-1}-\alpha|$ → α se aproxima a A_{i-1}

  Si $|Ai-\alpha| \leq |A_{i-1}-\alpha|$ → α se aproxima a A_i

  En este caso no se acumulan ERRORES ni por DEFECTO ni por EXCESO. El inconveniente que tiene es el coste; el coste de la operación de redondeo es alto puesto que supone una operación de truncamiento y otra de incremento (que equivale a una suma). En este caso tenemos que se cumple **- RES/2 < ERROR ≤ RES/2.**

- **Truncar y bit menos significativo a uno** → Es un método tan rápido como el truncamiento simple y presenta la ventaja de que sus ERRORES son por DEFECTO y por EXCESO, por lo que tiende a limitar su acumulación. En este caso se cumple que **- RES < ERROR < RES**.

4.4. CIRCUITOS INTEGRADOS PARA EL DISEÑO DE UNIDADES OPERATIVAS

La función que asocia al bloque que se denomina Unidad Operativa es la de realizar las distintas operaciones posibles con los datos. Para el diseño de la misma se cuenta con los siguientes tipos de circuitos integrados:

4.4.1. Unidad Aritmética-Lógica 74181

Se trata de un operador combinacional de cuatro bits, que realiza 16 operaciones aritméticas sencillas y otras 16 operaciones lógicas. Todos los computadores incluyen al menos un operador multifunción de coma fija, parecido a este circuito.

Para la definición de una unidad operativa con este circuito será necesario asociarle registros donde hacer permanecer los datos de entrada sin modificación durante el tiempo que dure la operación correspondiente.

4.4.2. Procesadores BIT-SLICE o unidades operativas universales

Son los dispositivos denominados *Bit-Slice Processor*, Unidades Operativas Expansibles. Esta expansión se puede realizar de dos formas: con propagación de acarreo en serie y con propagación de acarreo en paralelo; en este último caso es necesario incluir en el esquema un bloque funcional generador de acarreos. Las unidades operativas universales deben poseer un conjunto de características mínimas:

- Un adecuado juego de operaciones en los operadores: suma, producto, inversión, etc. En el caso de que la unidad operativa no posea un operador producto aritmético, la unidad operativa debe poseer un circuito combinacional desplazador que permita realizar la multiplicación mediante una sucesión de sumas.
- Una cierta capacidad de operación en paralelo, ampliable mediante el acoplamiento de bloques idénticos.

El diagrama de bloques representativo de las unidades operativas expansibles se representa en la Figura 4.27.

Esta unidad está compuesta por los siguientes elementos:

a) Unidad de memoria de acceso aleatorio, de acceso múltiple en lectura y escritura simultánea en una de las posiciones, constituida por un conjunto de biestables activados por nivel. La combinación de la memoria con los dos registros colocados en las salidas A y B, formada por biestables

activados por niveles opuestos a los de la memoria, constituyen en realidad una unidad de memoria activada por flancos.

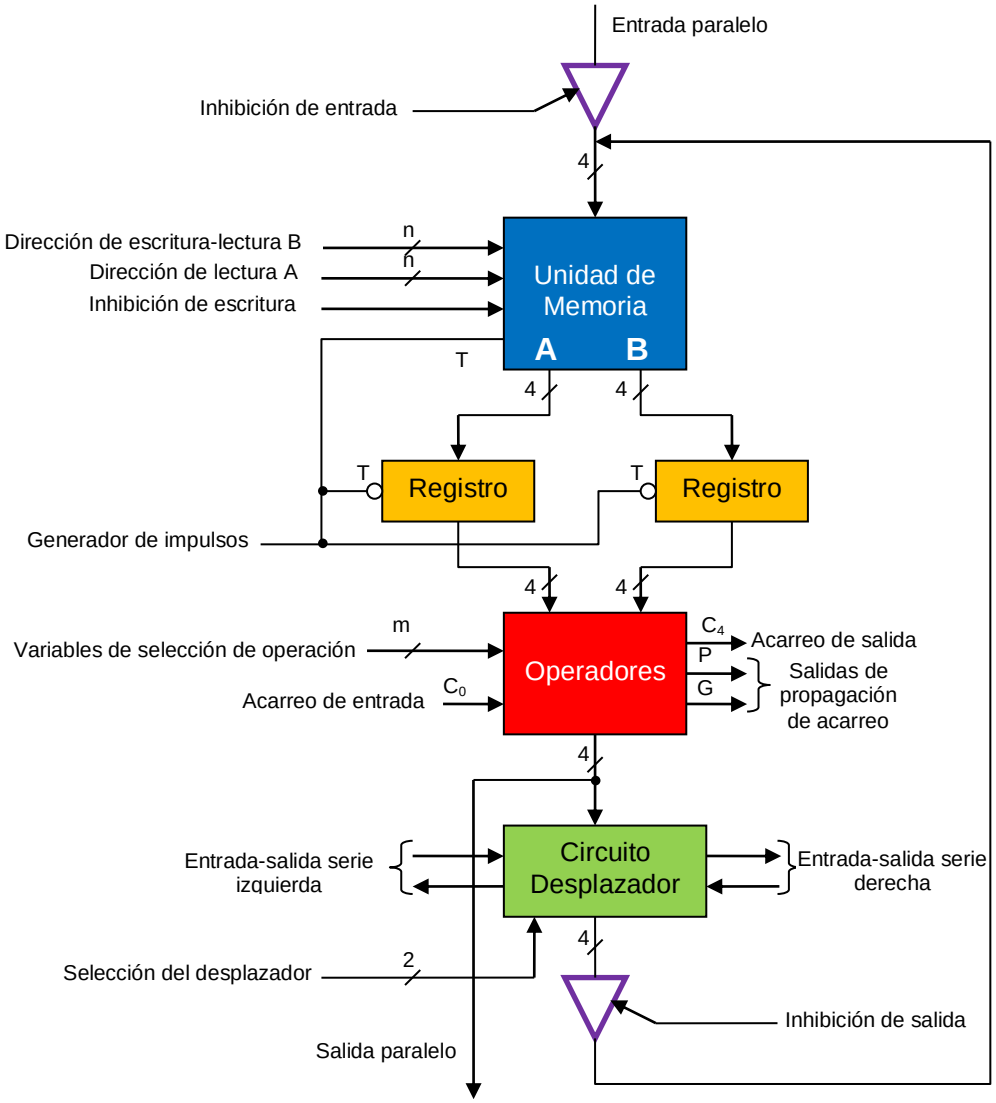


Figura. 4.27. *Esquema básico de una unidad operativa de 4 bits expansible.*

b) Un operador que posee variables de selección de operación, entrada y salida de acarreo, y salidas de generación y propagación de acarreo G y P respectivamente.

c) Un circuito desplazador combinacional cuyas entradas se conectan a las salidas del operador, y que tiene como misión desplazar el resultado una posición hacia la derecha o hacia la izquierda. Un circuito desplazador combinacional está formado por un conjunto de multiplexores (tantos como bits posea la combinación binaria que se procesa. Las entradas serie permiten desplazar introduciendo ceros o unos o la información procedente de otro bloque funcional idéntico. La salida del circuito

desplazador se conecta a las entradas de la unidad de memoria, a través de un triestado, para poder memorizar el resultado.

Como ejemplo ilustrativo se presenta en la Figura 4.28 el diagrama de bloques de la unidad operativa Am 2901.

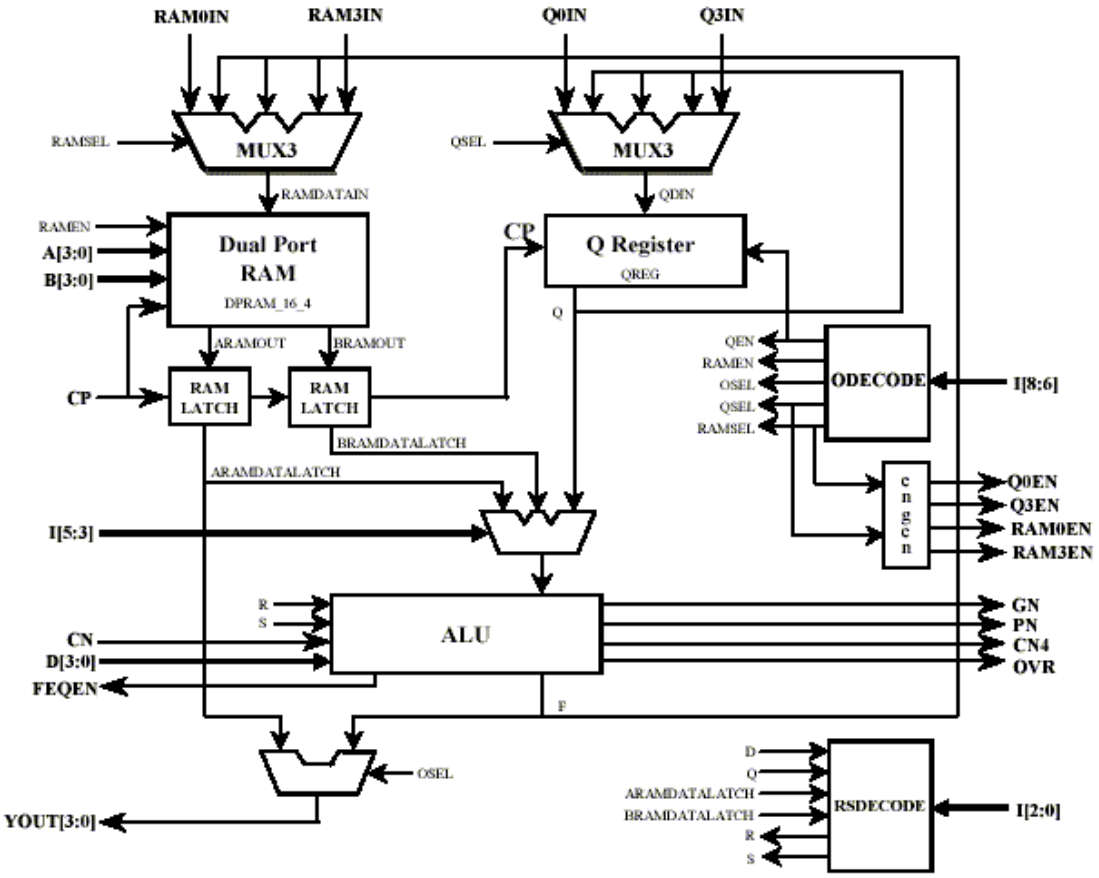


Figura 4.28. *Esquema de bloques de la Unidad Operativa Am 2901.*

4.4.3. Coprocesadores matemáticos

En los microprocesadores de 8 bits típicos, como el INTEL 8085 y el Z-80, la tecnología imponía un límite al área del chip. Como consecuencia, estas CPUs no disponían de hardware o firmware para ejecutar cálculos científicos, como aritmética en coma flotante, manipulación de matrices o tratamientos gráficos. Por tanto, los usuarios de estos sistemas tenían que escribir estos programas. Desgraciadamente, esta aproximación es inaceptable en aplicaciones de alta velocidad, ya que la ejecución del programa ocupaba un intervalo de tiempo importante. Para eliminar este problema se empleaban los COPROCESADORES.

Se utiliza un chip independiente para cálculos científicos a alta velocidad. Sin embargo, este chip se consideraba como un acompañante de la CPU original (también llamada HOST). Normalmente cada operación especial se codifica como una operación

que sólo puede interpretar la CPU dependiente. Cuando la CPU dependiente se encuentra con una de estas instrucciones, la asume de forma independiente con respecto a la CPU. Un procesador que opera de esta forma se denomina ***COPROCESADOR***.

Funcionalmente, un COPROCESADOR proporciona una extensión lógica del modelo del programador en cuanto a instrucciones, registros y tipos de operando. Esta extensión es transparente al programador.

Es importante distinguir entre el hardware periférico que habitualmente rodea a una CPU, y un coprocesador. Un coprocesador es un dispositivo capaz de comunicarse con el procesador principal a través de un protocolo definido en el *interfaz del coprocesador*.

En el caso del hardware periférico, el procesador accede a éste a través de registros con direcciones en el espacio de memoria del procesador principal. El programador utiliza instrucciones normales del procesador para acceder a los registros de interfaz de los periféricos y utilizar así los servicios que proporcionan éstos.

Existen tres técnicas generales para pasar órdenes a un coprocesador:

- Utilizar un *INTERFAZ INTELIGENTE DE CONTROL*
- Utilizar un *COPROCESADOR CON SEÑALES E INSTRUCCIONES ESPECIALES.*
- Utilizar un *COPROCESADOR CON INSTRUCCIONES ESPECIALES*.

  - **Interfaz Inteligente de Control.** Esta forma de comunicación de la CPU con el coprocesador es empleada por INTEL, con el coprocesador 8087. El esquema representativo de esta forma de conexión se representa gráficamente en la Figura 4.29.

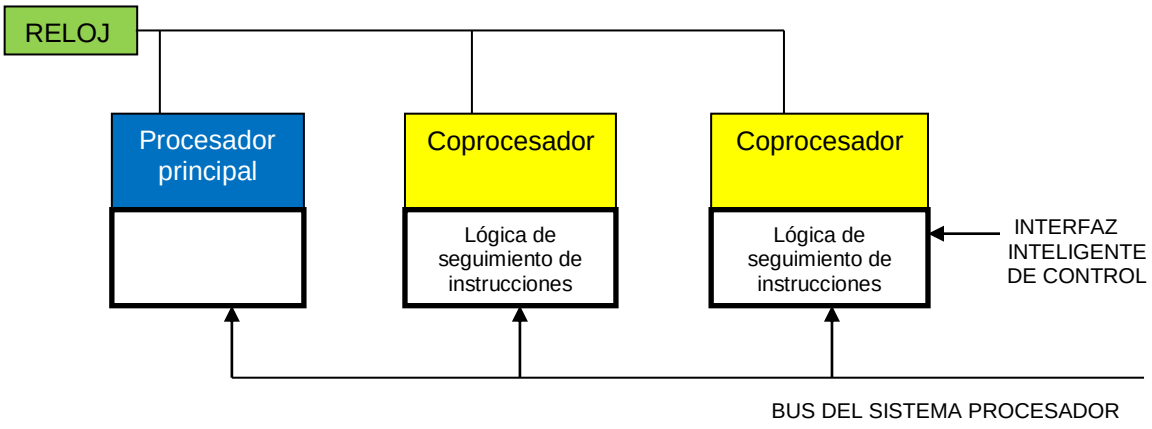


Figura. 4.29. *Coprocesador 8087 de INTEL.*

En este caso, el coprocesador debe controlar la secuencia de instrucciones tomándolas directamente del bus y al mismo tiempo que el procesador principal.

VENTAJAS:

- No se necesitan ciclos de instrucción adicionales par pasar el contenido de la palabra de instrucciones al coprocesador.

DESVENTAJAS:

- Cada coprocesador del sistema debe duplicar la circuitería de control y la cola de instrucciones, seguir las bifurcaciones, estados de espera y cargas (fetches) de operandos e instrucciones.
- Las técnicas de control simultáneo del bus exigen que el par procesador/coprocesador opere no más deprisa que la máxima velocidad a la que puede operar el elemento más lento del conjunto.

- **Interfaz utilizando un bus con señales e instrucciones especiales.** Esta conexión la emplea el coprocesador NS 16081 de NATIONAL. En la Figura 4.30 se muestra este tipo de conexión.

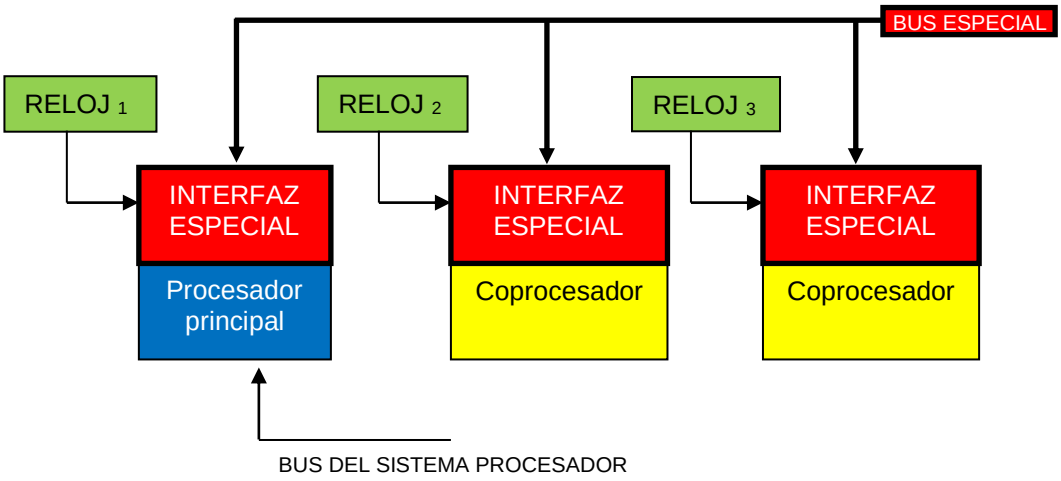


Figura. 4.30. *Coprocesador NS16081 de NATIONAL.*

Con esta conexión el coprocesador puede direccionarse mediante algunas instrucciones que utilicen un protocolo especial. Cada procesador principal debe conocer previamente el tipo de coprocesador al que está conectado si ambos deben trabajar de forma eficiente.

VENTAJAS:

- El coprocesador puede ser más rápido que el procesador principal, puesto que las transferencias de datos se producen de forma asíncrona mediante handshakes.

DESVENTAJAS:

- Sobrecarga de la CPU. El procesador principal debe reconocer que una instrucción se dirige a un coprocesador y dirigirla al coprocesador adecuado.

- **Interfaz utilizando instrucciones especiales.** Esta conexión la emplea el coprocesador 68881 de MOTOROLA. En la Figura 4.31 se muestra gráficamente un esquema para este tipo de interfaz. En este caso, el coprocesador puede direccionarse explícitamente utilizando instrucciones especiales para el procesador principal, que inician una secuencia especial de microinstrucciones en el procesador principal para ejecutar las instrucciones y la transferencia de operandos.

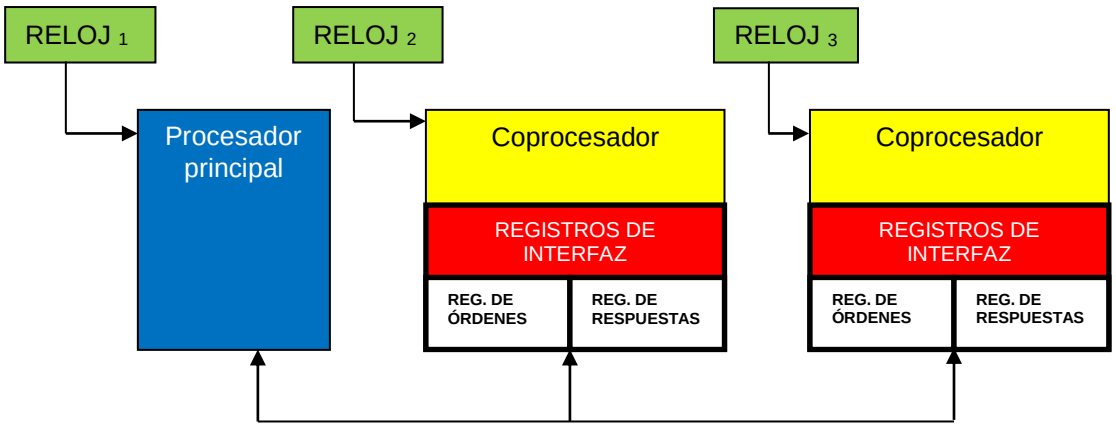


Figura. 4.31. *Coprocesador 68881 de MOTOROLA.*

Cuando el procesador principal efectúa una instrucción del coprocesador, decodifica la instrucción y deja una orden en el ***REGISTRO DE ÓRDENES*** (uno de los registros de interfaz) especificando la operación que debe realizar el coprocesador. Como respuesta, el coprocesador escribe su respuesta en el ***REGISTRO DE RESPUESTA*** (otro de los registros de interfaz). El procesador principal puede entonces leer estos datos y comunicarle al coprocesador dónde se encuentra la información adicional que éste puede necesitar para efectuar las operaciones necesarias. Si el coprocesador necesita estos datos procede a leerlos; en caso contrario, efectúa la operación de forma concurrente con el procesador principal y le indica los resultados obtenidos.

VENTAJAS:

- No se necesitan señales especiales.
- No se limita de forma inherente el número de coprocesadores.

DESVENTAJAS:

- Una vez que el procesador principal detecta una instrucción para el coprocesador, utiliza el ancho de banda del bus y temporizaciones para transmitir.