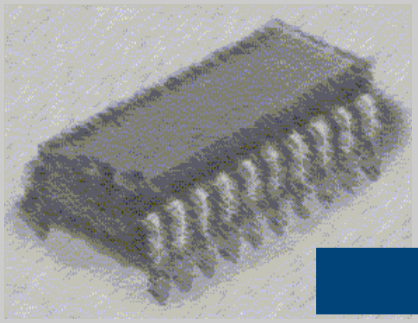
Soluciones a los problemas impares

Tema 3. La Unidad Aritmético-Lógica



Arquitectura de Computadores I

I. T. Informática de Sistemas

Curso 2009-2010

Base teórica

Al diseñar un computador, uno de los puntos a tener en cuenta es el diseño del camino de datos, y en definitiva, las operaciones y número de operandos con los que trabajará la ALU.

Suma entera

La suma es la operación aritmética más importante de todas las que realiza la ALU. Está presente en la actualización de contador de programa, y en todos los direccionamientos relativos, por lo que interesa que sea muy rápida. Para ello existen varias técnicas de aceleración de la misma.

Anticipador de acarreo puro

Se identifican las funciones de generación de acarreo y de propagación de acarreo lo que permite poder generar en paralelo el acarreo necesario para la siguiente suma:

Función generación ($g_i = a_i \cdot b_i$)

Función propagación ($p_i = a_i \oplus b_i$)

Acarreo siguiente ($c_{i+1} = g_i + p_i \cdot c_i$)

Anticipador de acarreo por bloques

Se tienen bloques sumadores de k bits y se calculan las funciones de propagación y de generación a nivel de bloque. Las siguientes ecuaciones son para el primer bloque.

$P0 = p3 \cdot p2 \cdot p1 \cdot p0$

$G0 = g3 + (g2p3) + (g1p3p2) + (g0p3p2p1)$

$C0 = G0 + P0 \cdot c0$

$C1 = G1 + (G0 \cdot P1) + (c0 \cdot P1 \cdot P0)$

Salto de acarreo

Dado que el cálculo de las Pi de bloque es más sencillo que el de las Gi, se emplean a la hora de anticipar el acarreo tal y como muestra la figura 1.

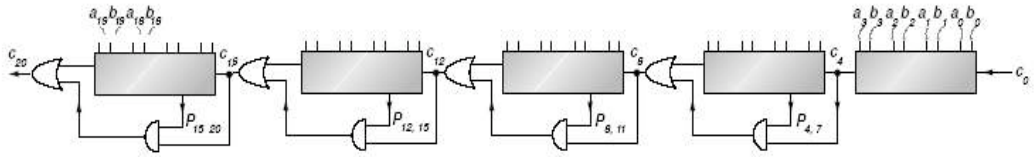


Figura 1. Sumador con salto de acarreo

Los niveles de puertas para obtener el acarreo de salida es de *4k + 2n/k -3*, siendo *k* el número de bloques necesario para sumar *n* bits.

Sumador con selección de acarreo

Consiste en duplicar el número de sumadores menos uno y realizar las sumas con acarreo de entrada 0 y acarreo de entrada 1. Las salidas irán a un multiplexor que se encargará de mostrar el resultado correcto, tal y como refleja la figura 2.

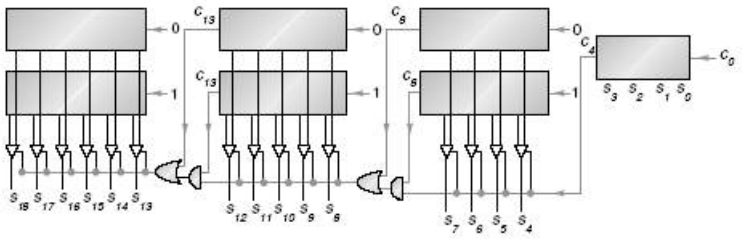


Figura 2. Sumador con selección de acarreo

Los niveles de puertas para obtener el acarreo de salida es de *2k + 2n/k*, siendo *k* el número de bloques necesario para sumar *n* bits.

Multiplicación entera

La multiplicación entera es una operación bastante costosa y, por ello, al principio, no todos los micros la tenían implementada debiéndose multiplicar por programa.

Existen varios algoritmos de multiplicación: sumas y restas, Booth, grupos solapados. Dependiendo del algoritmo se pueden multiplicar números con signo o sin signo.

Algoritmo de sumas restas

Es un algoritmo que inicialmente se pensó para números en binario puro, aunque se puede modificar fácilmente para multiplicar operandos representados en complemento a 2 y a 1.

Consiste en ir explorando el bit menos significativo del registro de desplazamiento (que toma como valor inicial el multiplicador) y en el caso de ser 1 sumar el multiplicando y desplazar o simplemente desplazar. Recoge realmente el mecanismo de multiplicación a mano.

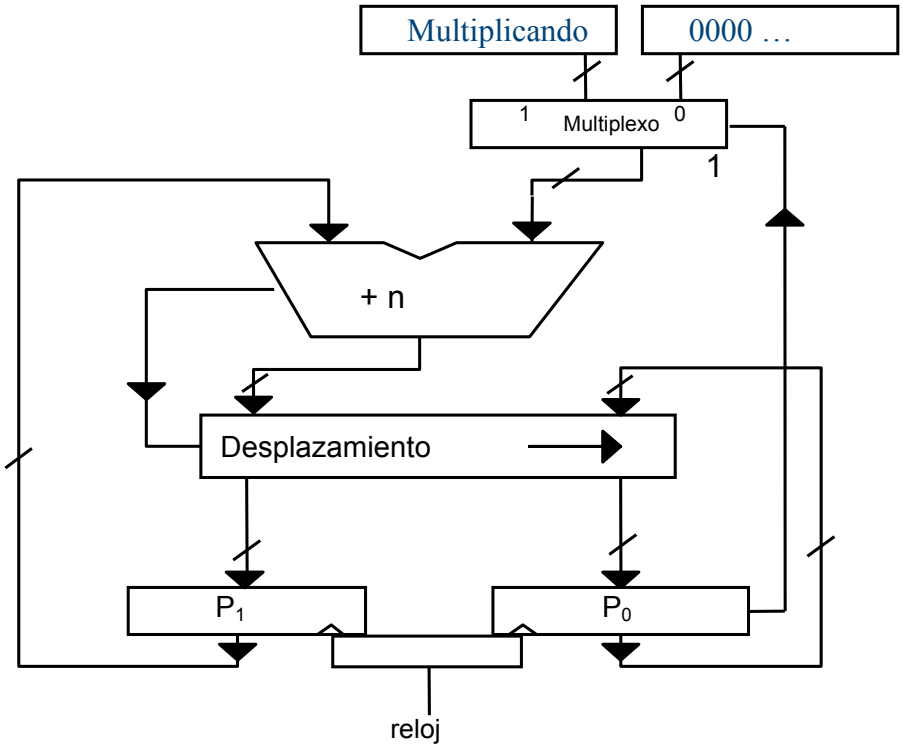


Figura 3. Multiplicador mediante el algoritmo de sumas-restas

Si se desea operar con valores representados en complemento a 2 deberemos tener en cuenta que si el multiplicador es negativo, cuando llegue el 1 del bit más significativo al multiplexor, entonces, se restará A.

En el caso de que los operandos estén expresados en complemento a 1, si el multiplicador es negativo, cuando llegue el 1 del bit más significativo al multiplexor, entonces, restaremos A. Y además, en la fase de inicialización P1 y el registro de desplazamiento toman el valor inicial del multiplicando.

Algoritmo de Booth

El algoritmo de Booth se ideo pensando en números representados en complemento a 2 y reduce el número de sumas, evitando las que son 0. Se basa en buscar cadenas de unos y se recodifica el multiplicador. Así, al llegar al primer uno de la cadena se resta el multiplicando afectado del peso del 1 y se sigue hasta encontrar un cero, en cuyo caso se suma el multiplicando afectado del peso del cero. Se siguen los pasos anteriores hasta terminar de recodificar el multiplicador.

Se debe tener en cuenta que al ser números expresados en complemento a 2, se debe realizar la extensión de signo de manera adecuada al operar.

División entera

Al igual que la multiplicación se trata de una operación compleja. Aunque existen muchos algoritmos, veremos únicamente un par de algoritmos que permiten operar con números sin signo.

División con restauración

1. **Dividendo parcial inicial**: tomar tantos bits del dividendo como tenga el divisor. Garantizar que tanto el dividendo como el divisor son positivos (si empiezan por 1 añadirle un cero a la izquierda)
2. Sumar al dividendo el complemento a 2 del divisor

   Si el resultado es positivo:

   - Bajar un nuevo bit del dividendo

- Añadir 1 al cociente

Si el resultado es negativo:

- Sumar de nuevo el divisor
- Bajar un nuevo bit del divisor
- Añadir un 0 al cociente

3. Repetir hasta que no queden más bits para bajar del dividendo

División sin restauración

Mejora el anterior evitando la fase de restauración si el resultado es negativo.

1. **Dividendo parcial inicial**: tomar tantos bits del dividendo como tenga el divisor. Garantizar que tanto el dividendo como el divisor son positivos (si empiezan por 1 añadirle un cero a la izquierda)
2. Sumar al dividendo el complemento a 2 del divisor

   Si el resultado es positivo:

   - Bajar un nuevo bit del dividendo
   - Añadir 1 al cociente

   Si el resultado es negativo:

   - Sumar el divisor en vez del complemento a 2 la próxima vez
   - Bajar un nuevo bit del divisor
   - Añadir un 0 al cociente

3. Repetir hasta que no queden más bits para bajar del dividendo

Coma flotante y normalización

La coma flotante es la manera que tenemos para poder representar números racionales (con decimales) Emplea un mecanismo similar al que estamos acostumbrados a ver cuando obtenemos un resultado demasiado grande como resultado en una calculadora: la notación exponencial. Para

ello se divide el espacio material con que cuente el computador en dos partes, la primera para expresar el exponente (representado en como fija) y la segunda la mantisa (expresanda de manera fraccionaria)

La normalización consiste en eliminar de la mantisa todos los bits que no sean significativos y ajustar el exponente de manera adecuada, de tal modo que no cambie ni el valor ni el signo del número. Dependerá del sistema de representación empleado el criterio de normalización

Signo-magnitud

A la derecha de la coma no habrá ningún cero, ya que no son significativos. Este criterio será válido tanto para positivos como negativos ya que existe un bit de signo independiente de la magnitud del número que se expresa en binario puro.

Un mantisa quedaría normalizada si está expresado como: ,1xxxx……xxxx

Complemento a 1 y complemento a 2

En estas representaciones los números no tienen un bit de signo independiente, sino que el bit más significativo del número indica el signo del mismo. Por ello, se deberá distinguir si el número es positivo (los dígitos no significativos serán los ceros) o negativo (los unos serán los dígitos que eliminaremos)

Una mantisa positiva normalizada quedaría como ,01 xxx … xxxxx

Una mantisa negativa normalizada quedaría como ,10 xxx … xxxxx

Técnica del bit implícito

La técnica del bit implícito consiste en “arañar” un bit más. A efectos prácticos es como si la mantisa contase con un bit extra para aumentar la precisión del número.

Esta técnica se aplica únicamente a números normalizados y consiste en no representar (por lo que tendremos un bit extra al final) el primer bit del criterio de normalización el 1 en signo-magnitud, el 0 en C2 y C1 si es

positivo o el 1 en C2 y C1 si es negativo. La ALU, sin embargo, lo tendrá en cuenta a la hora de operar.

Estándar IEEE 754 para la coma flotante

En dicho estándar se representan números en coma flotante en simple (32 bits) y doble precisión (64 bits)

Simple precisión

- Exponente de 8 bits expresados en exceso $2^{8-1}-1 = 127$
- Mantisa con 24 bits, expresada en signo-magnitud, fraccionaria, normalizada y con el bit implícito a la izquierda de la coma decimal

Doble precisión

- Exponente de 11 bits expresados en exceso $2^{11-1}-1 = 1023$
- Mantisa con 53 bits, expresada en signo-magnitud, fraccionaria, normalizada y con el bit implícito a la izquierda de la coma decimal

El estándar IEEE 754 contempla las combinaciones siguientes del exponente y de la mantisa para poder reflejar casos especiales

- Exponente = 0 y Mantisa = 0. El número representado es el ±0
- Exponente = 0 y Mantisa ≠ 0. El número representado no se encuentra normalizado.
- Exponente todo unos y Mantisa = 0. El numero representado es ±∞
- Exponente todo a unos y Mantisa ≠ 0. Es una condición de excepción, como resultado no válido, por ejemplo.

Suma en coma flotante

Para poder sumar valores expresados en coma flotante, se deberán alinear las mantisas ajustando los exponentes de manera adecuada. El siguiente algoritmo expresa la manera de realizar la suma en coma flotante

1. Separar las mantisas de los exponentes
2. Comparar los exponentes y:

1. Guardar el exponente mayor que será el del resultado salvo que el número salga desnormalizado
2. Restar del exponente mayor el menor. Dicho número será el número de veces que se tendrá que desplazar a la derecha la mantisa menor

3. Desplazar la mantisa menor a la derecha para alinear las mantisas
4. Realizar la suma o la resta
5. Comprobar si el número está normalizado y en caso de que no lo esté, normalizarlo
6. Realizar el redondeo si es preciso

Multiplicación y división en coma flotante

Simplemente es emplear los algoritmos vistos en coma fija de multiplicación y división para las mantisas y luego sumar o restar los exponentes dependiendo de si se trata de una multiplicación o de una división

Dígitos de guarda y redondeo

Para mejorar la precisión de los resultados, la ALU, suele trabajar con más bits que el tamaño de los registros, por lo que una vez realizada las operaciones se deberá redondear le resultado y eliminar dichos bits extra.

Normalmente con dos bits de guarda y un bit retenedor se puede obtener una precisión más que aceptable.

La manera de expresar un número con dígitos de guarda es la siguiente:

b8b7b6b5b4b3b2b1b0 bg1 bg2 br

- Los bits b8 a b0 son un dato de 8 bits
- bg1es el primer bit de guarda que se emplea para la normalización
- bg2 es el segundo bit de guarda que se emplea para el redondeo
- br es el bit retendor que se emplea para no perder la precisión en la operación de resta y que se mantiene a 1 al pasar un 1 por él

El redondeo se realiza a la hora de eliminar esos bits. Las técnicas más empleadas son: truncación, forzar a 1 el bit menos significativo y redondear al más próximo.

Truncación

Consiste en eliminar los bits de guarda y el bit retenedor.

Forzar el bit menos significativo a 1

Consiste en eliminar los bits de guarda y el bit retenedor y poner a 1 el bit menos significativo del resultado. Si ya estaba a 1 quedará igual

Redondeo al más próximo

Es la más difícil de implementar pero la que mejor resultado proporciona. Si supera la mitad del valor de los bits de guarda se suma uno al resultado. Si no llega a la mitad, se trunca. En el caso de que sea la mitad, se fuerza a par.

Por ejemplo si contamos con 3 bits de guarda las combinaciones 000, 001, 010, 011 se truncarían. Con 101, 110 y 111 se truncaría y se sumaría 1 al resultado. La combinación 100 sumará 1 si el LSB es 1 ó truncará si el LSB es 0

1. Se cuenta con un sumador con anticipación de acarreo. Se desean sumar los dos números de 16 bits siguientes representados en binario.

   A = 0001 1010 0011 0011 y B = 1110 0101 1110 1011

   Calcúlense los valores de los gi y pi.

2. Se cuenta con un sumador con anticipación de acarreo. Se desean sumar los dos números de 16 bits siguientes representados en binario.

   A = 0111 1110 0000 1011 y B = 1111 1101 1100 1011

   Calcúlense los valores de los gi y pi.

3. Se cuenta con un sumador con anticipación de acarreo por pasos. Se desean sumar los dos números de 16 bits siguientes representados en binario.

   A = 0001 1010 0011 0011 y B = 1110 0101 1110 1011

   Calcúlense los valores de los gi, pi, Pi y Gi y del acarreo C4. Se supone un acarreo de entrada c0 = 1

4. Se cuenta con un sumador con anticipación de acarreo. Se desean sumar los dos números de 16 bits siguientes representados en binario.

   A = 0111 1110 0000 1011 y B = 1111 1101 1100 1011

   Calcúlense los valores de los gi, pi, Pi y Gi y del acarreo C4. Se supone un acarreo de entrada c0 = 1

5. Se tiene un sumador/restador en coma flotante que opera con números de 16 bits representados con el siguiente formato:

   - Exponente: 8 bits en exceso 2^{8-1}.
   - Mantisa: 8 bits en complemento a 2, normalizada y fraccionaria.

   a) Realizar la suma de los números A y B, tal como lo haría dicho sumador. Suponiendo que A: 1000 0011 | 0110 0011 y B: 1000 0110 | 1001 1100 (exponente | mantisa)

   b) ¿Existe alguna diferencia entre el resultado obtenido por este sumador y el resultado real? ¿Por qué?

6. Considerando el sumador anterior y sabiendo que dicho sumador opera internamente con dos dígitos de guarda y un bit retenedor, y realiza el redondeo al más próximo:

   a) Realizar la suma de A: 1000 0110 | 0110 0010 y B: 1000 0010 | 1001 0000 (exponente | mantisa)

   b) ¿Existe alguna diferencia entre el resultado obtenido por este sumador y el resultado real? ¿Por qué?

7. Considerar un sumador/restador que opera con datos representados en coma flotante, con mantisa normalizada y fraccionaria expresada en Ca1 y exponente representado en exceso 2^{4-1}, además dicha unidad opera con dos dígitos de guarda y un bit retenedor. Emplea el redondeo al más próximo.

   Calcular la suma de A+B, siendo A: 1101 | 10110 y B: 1101 | 01011 (exponente | mantisa)

8. Calcular el producto de A x B, sabiendo que A: 0100010 y B: 1110010 están representados en C2.

9. Sabiendo que A: 110011 y B: 010001, están ambos representados en binario puro calcular el producto de A x B utilizando el algoritmo de suma-desplazamiento. Realizar paso a paso dicho producto, sabiendo que el resultado se almacenará en un registro P de 12 bits, formado por dos registros concatenados P1 y P0 de 6 bits.

10. Calcular el producto de A x B, sabiendo que A: 0100110 y B: 0110010 están representados en C2.

11. Calcular el producto de A x B, sabiendo que A: 111110 y B: 1111111 están representados en C2.

12. Calcular el cociente entre D: 0100100 y d: 1001, ambos representados en binario puro,
    a) Utilizando el algoritmo de división con restauración.
    b) Utilizando el algoritmo de división sin restauración.

13. Calcular el cociente entre D: 0100110 y d: 1110, ambos representados en binario puro,
    a) Utilizando el algoritmo de división con restauración.
    b) Utilizando el algoritmo de división sin restauración

14. Calcular el cociente entre D: 0100101 y d: 0111, ambos representados en binario puro,

a) Utilizando el algoritmo de división con restauración.

b) Utilizando el algoritmo de división sin restauración

---

15. Se tiene un sumador de coma flotante que opera con números de 16 bits representados en el formato siguiente:

Exponente	Mantisa

- Exponente: 8 bits, representado en exceso 2^{n-1}.
- Mantisa: 8 bits, representada en complemento a 2, fraccionaria, normalizada y **no emplea** bit implícito.

El sumador opera con un bit de guarda y un bit retenedor y se emplea como técnica de redondeo la de redondeo al más próximo.

Sean los números A y B siguientes

A	
1000 1010	0100 0011

B	
1000 0101	0111 0000

Se pide:

a) Realizar la suma de A y B tal y como lo haría el sumador.

b) Calcular el valor en decimal del resultado.

c) Calcular el valor real de la suma, también en decimal si se hubiese realizado a mano.

d) Si existe diferencia entre los apartados b) y c) explicar a qué es debido.

e) ¿Cambiaría el resultado si en lugar de redondeo al más próximo empleásemos la técnica de forzar el bit menos significativo a uno?

---

16. Sea un computador que trabaja con el siguiente sistema de representación de la información para coma flotante:
    - Exponente expresado en exceso 2^{n-1} con 8 bits.
    - Mantisa en expresada en complemento a 1, normalizada y sin bit implícito con 8 bits.

Sabiendo que la ALU de dicho computador permite la suma de números en coma flotante, que emplea internamente dos bits de guarda y un bit retenedor y que emplea la técnica de redondeo al más próximo.

Se pide realizar la suma de los números (exponente || mantisa):

- A = 1000 0111 **||** 1010 0000
- B = 1000 0010 **||** 0110 0000

---

17. Sea un computador que trabaja con el siguiente sistema de representación de la información para coma flotante:
    - Exponente expresado en exceso 2^{n-1} con 8 bits.
    - Mantisa en expresada en complemento a 1, normalizada y sin bit implícito con 8 bits.

Sabiendo que la ALU de dicho computador permite la suma de números en coma flotante, que emplea internamente dos bits de guarda y un bit retenedor y que emplea la técnica de redondeo al más próximo.

Se pide realizar la suma de los números (exponente || mantisa):

- A = 1000 0111 **||** 1000 0000
- B = 1000 0010 **||** 0100 0000

Solución ejercicio 1

Si alineamos los valores será fácil el cálculo ya que gi = aixbi y pi = ai $\oplus$ bi

ai = 0001 1010 0011 0011

bi = 1110 0101 1110 1011

gi = 0000 0000 0010 0011

pi = 1111 1111 1101 1000

Solución ejercicio 3

Si alineamos los valores será fácil el cálculo ya que gi = aixbi y pi = ai $\oplus$ bi

ai = 0001 1010 0011 0011

bi = 1110 0101 1110 1011

gi = 0000 0000 0010 0011

pi = 1111 1111 1101 1000

P0 = p3xp2xp1xp0 = 1 x 0 x 0 x 0 = 0

P1 = p7xp6xp5xp4 = 1 x 1 x 0 x 1 = 0

P2 = p11xp10xp9xp8 = 1 x 1 x 1 x 1 = 1

P3 = p15xp14xp13xp12 = 1 x 1 x 1 x 1 = 1

G0 = g3 + (p3xg2) + (p3xp2xg1) + (p3xp2xp1xg0) = 0

G1 = g7 + (p7xg6) + (p7xp6xg5) + (p7xp6xp5xg4) =1

G2 = g11 + (p11xg10) + (p11xp10xg9) + (p11xp10xp9xg8) = 0

G3 = g15 + (p15xg14) + (p15xp14xg13) + (p15xp14xp13xg12) = 0

Con lo que C4 será:

C4 = G3 + (P3XG2) + (P3XP2XG1) + (P3XP2XP1XG0) +
+(P3xP2xP1xPp0xc0) = 1

Solución ejercicio 5

Apartado a)

1. Primero separamos mantisas y exponentes:

	A	B
Exponente	1000 0011	1000 0110
Mantisa	0110 0011	1001 1100

2. Comparamos los exponentes

   Exponente A = 3
   Exponente B = 6

   Con lo que el exponente para el resultado salvo que haya que normalizar, será el exponente de B

3. Alineamos las mantisas y realizamos la suma

Para alinear, desplazaremos hacia la derecha la mantisa afectada del menor exponente. La desplazamos 3 veces (Exponente B – Exponente A) y realizaremos la suma

   1001 1100
   0000 1100 ~~011~~ + (los bits tachados se pierden al desplazar)
   1010 1000

4. Normalizar el resultado

El valor del resultado se encuentra normalizado ya que la mantisa está expresada en complemento a 2

Por lo tanto el resultado es:

	Resultado
Exponente	1000 0110
Mantisa	1010 1000

Apartado b)

Calculamos el valor de A

	A
Exponente	1000 0011
Mantisa	0110 0011
Valor	$(2^{-2} + 2^{-3} + 2^{-7} + 2^{-8}) \cdot 2^{3} = 3{,}09375$

Calculamos el valor de B

	B
Exponente	1000 0110
Mantisa	1001 1100
Valor	$-(2^{-2}+2^{-3}+2^{-6})\cdot 2^{6} = -25$

Calculamos el valor del resultado anterior

	Resultado
Exponente	1000 0110
Mantisa	1010 1000
Valor	$-(2^{-2} + 2^{-4} + 2^{-5})\cdot 2^{6} = -22$

Realizando la suma de ambos valores

A + B = -25 + 3,09375 = -21,90625

Como apreciamos, el resultado tiene un error con respecto al obtenido debido a que al deplazar a la derecha la mantisa se pierden bits con información significativa.

Solución ejercicio 7

Apartado a)

1. Primero separamos mantisas y exponentes:

	A	B
Exponente	1101	1101
Mantisa	10110	01011

2. Comparamos los exponentes

   Exponente A = 5
   Exponente B = 5

Con lo que el exponente para el resultado salvo que haya que normalizar, será 5

3. Alineamos las mantisas y realizamos la suma

Como los exponentes son iguales no es necesario desplazar ninguna mantisa, pero al tener dos bits de guarda y un bit retenedor tendremos que inicializarlos adecuadamente. Los hemos resaltado en negrita

```
 10110 1 1 1
 01011 0 0 0 +
100001 1 1 1
```

Resulta que se produce un acarreo de salida en la suma. Como los números A y B se encuentran expresados en complemento a 1 ese acarreo debe sumarse al resultado para corregir el resultado

```
 00001 1 1 1
 00000 0 0 1 +
 00010 0 0 0
```

4. Normalizar el resultado

El resultado no se encuentra normalizado ya que los primeros bits no son significativos por lo que habrá que desplazar la mantisa hacia la izquierda dos veces con lo que deberemos decrementar en 2 el exponente.

	Resultado
Exponente	1011
Mantisa	01000 **0 0 0**

5. Redondeo del resultado

Como los bits de guarda y el bit retenedor son cero, el redondeo al más próximo coincide con la truncación del resultado quedando:

	Resultado
Exponente	1011
Mantisa	01000

Solución ejercicio 9

Estado inicial:

A: 110011,
P_0 = B =010001
P_1: 000000
Registro de desplazamiento: 000000 000000

Registro de desplazamiento	P		Operación
	P_1	P_0	
000000 000000	000000	010001	Fase inicial
110011 010001	000000	010001	Suma
011001 101000	011001	101000	Desplazamiento
011001 101000	011001	101000	Suma
001100 110100	001100	110100	Desplazamiento
001100 110100	001100	110100	Suma
000110 011010	000110	011010	Desplazamiento
000110 011010	000110	011010	Suma
000011 001101	000011	001101	Desplazamiento
110110 001101	000011	001101	Suma
011011 000110	011011	000110	Desplazamiento
011011 000110	011011	000110	Suma
001101 100011	001101	100011	Desplazamiento

Solución ejercicio 11

Si A y B están representados en C2, la forma más eficiente de realizar la multiplicación es utilizando un algoritmo de multiplicación con signo: el algoritmo de Booth:

	Representación
A	1 1 1 1 1 0
-A Complemento a 2 (A)	0 0 0 0 1 0
B	1 1 1 1 1 1

Recodificamos el multiplicador:

Cadena de unos, primer uno afectado del peso 0, entonces restar A
Ya o hay ceros, hemos terminado de recodificar el multiplicador por lo que el resultado sera:

$A \times B = -A \times 2^0 = 0\ 0\ 0\ 0\ 1\ 0$

Solución ejercicio 13

Ambos algoritmos operan con operandos sin signo. En primer lugar, hay que considerar que el dividendo y el divisor deben comenzar con un bit 0. Además se emplea el complemento a dos del divisor para reducir a sumar el complemento en vez de restar.

Apartado a)

Utilizando el algoritmo de división con restauración:

	0	1	0	0	1	1	0	0 1 1 1 0
+	1	0	0	1	0			**0 1 0**
	1	1	0	1	1			
	0	**1**	**0**	**0**	**1**	**1**		← restauramos
+	1	1	0	0	1	0		
~~1~~	0	0	0	1	0	1	**0**	
+	1	1	1	0	0	1	0	
	1	1	1	1	1	0	0	
	0	**0**	**0**	**1**	**0**	**1**	**0**	← restauramos el resto

Apartado b)

Utilizando el algoritmo de división sin restauración:

	0	1	0	0	1	1	0	0 1 1 1 0
+	1	0	0	1	0			**0 1 0**
	1	1	0	1	1	**1**		
+	0	0	1	1	1	0		
~~1~~	0	0	0	1	0	1	**0**	
+	1	1	1	0	0	1	0	
	1	1	1	1	1	0	0	
	0	**0**	**0**	**1**	**0**	**1**	**0**	← restauramos el resto

Solución ejercicio 15

Apartado a)

5. Primero separamos mantisas y exponentes:

	A	B
Exponente	1000 1010	1000 0101
Mantisa	0100 0011	0111 0000

6. Comparamos los exponentes

   Exponente A = 10
   Exponente B = 5

   Con lo que el exponente para el resultado salvo que haya que normalizar, será el exponente de A

7. Alineamos las mantisas y realizamos la suma

Para alinear, desplazaremos hacia la derecha la mantisa afectada del menor exponente. La desplazamos 5 veces (Exponente A – Exponente B) y realizaremos la suma

0100 0011 **00**
0000 0011 **10** + (bits en negrita son el bit de guarda y el retenedor)
0100 0110 **10**

8. Normalizar el resultado

El valor del resultado se encuentra normalizado ya que la mantisa está expresada en complemento a 2

9. Redondear el resultado

Al ser redondeo al más próximo y contar con los bits de guarda y retenedor con el valor 10, se deberá comprobar si el bit menos significativo del resultado es cero o uno, para sumar cero o uno respectivamente al mismo.

Al ser 0 (0100 0110) se le sumaría cero y pr lo tanto el resultado es:

	Resultado
Exponente	1000 1010
Mantisa	0100 0110

Apartado b)

Calculamos el valor del resultado anterior

	Resultado
Exponente	1000 1010
Mantisa	0100 0110
Valor	$-(2^{-2} + 2^{-6} + 2^{-7})\cdot 2^{10} = 280,0$

Apartado c)

Calculamos el valor de A

	A
Exponente	1000 1010
Mantisa	0100 0011
Valor	$(2^{-2} + 2^{-7} + 2^{-8}) \cdot 2^{10} = 268,0$

Calculamos el valor de B

	B
Exponente	1000 0101
Mantisa	0111 0000
Valor	$(2^{-2}+2^{-3}+2^{-4})\cdot 2^{5} = 14,0$

Por lo tanto tendríamos 268,0 + 14,0 = 282,0

Apartado d)

Es debido a los desplazamientos para alinear las mantisas y a la pérdida de precisión que sufren los números fraccionarios al pasarlos a coma flotante.

Apartado e)

Al forzar el bit menos significativo a uno, el resultado sería el siguiente:

	Resultado
Exponente	1000 1010
Mantisa	0100 0111
Valor	$-(2^{-2} + 2^{-6} + 2^{-7} + 2^{-8})\cdot 2^{10} = 284,0$

Solución ejercicio 17

Apartado a)

1. Primero separamos mantisas y exponentes:

	A	B
Exponente	1000 0111	1000 0010
Mantisa	1000 0000	0100 0000

2. Comparamos los exponentes

   Exponente A = 7
   Exponente B = 2

   Con lo que el exponente para el resultado salvo que haya que normalizar, será el exponente de A

3. Alineamos las mantisas y realizamos la suma

Para alinear, desplazaremos hacia la derecha la mantisa afectada del menor exponente. La desplazamos 5 veces (Exponente A – Exponente B) y realizaremos la suma

1000 0000 **111**
0000 0010 **000** + (los bits en negrita son los 2 bits de guarda y el retenedor)
1000 0010 111

4. Redondear al más proximo

Como los bits de guarda son 111 deberemos truncar y sumar uno a la mantisa con lo que queda:

1000 0010
00000 001 +
1000 0011

5. Normalizar el resultado

El valor del resultado se encuentra normalizado ya que la mantisa está expresada en complemento a 1

Por lo tanto el resultado es:

	Resultado
Exponente	1000 0111
Mantisa	1000 0011