Universidad
Europea de Madrid

LAUREATE INTERNATIONAL UNIVERSITIES

FUNDAMENTOS DE COMPUTADORES

ALU

© Todos los derechos de propiedad intelectual de esta obra pertenecen en exclusiva a la Universidad Europea de Madrid, S.L.U. Queda terminantemente prohibida la reproducción, puesta a disposición del público y en general cualquier otra forma de explotación de toda o parte de la misma.

La utilización no autorizada de esta obra, así como los perjuicios ocasionados en los derechos de propiedad intelectual e industrial de la Universidad Europea de Madrid, S.L.U., darán lugar al ejercicio de las acciones que legalmente le correspondan y, en su caso, a las responsabilidades que de dicho ejercicio se deriven.

Índice

Presentación 4

Estructura y operaciones de una ALU 5

Operaciones de suma y resta I 7

Operaciones de suma y resta II 8

Multiplicación de binario puro 9

Multiplicación de enteros con signo 11

La división de enteros 13

División de enteros con signo 15

Multiplicación de rápidos 16

Operaciones con números reales - coma flotante 18

Resumen 21

Presentación

La Unidad Aritmético Lógica, es uno de los componentes que Von Neumann implementó en su primer diseño, como un componente fundamental para el computador.

La tarea de este componente, tal como su propio nombre indica, se encarga de realizar las operaciones aritméticas (+, -, *, etc.) y lógicas (and, or, not, etc.) sobre datos, mayoritariamente números.

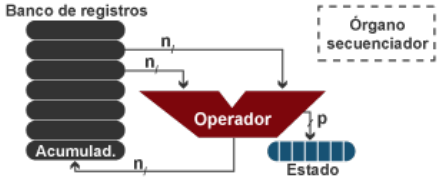
Hay muchas maneras de almacenar números en binario. Tenemos que diferenciar entre números positivos y negativos, enteros y reales, por lo que, para entender una Unidad Aritmético Lógica, hay que tener muy presente estos métodos de representación de la información.

Por tanto, debemos recordar que la representación de información numérica más estándar en los computadores actuales consiste en:

- Enteros
  - Enteros positivos: Binario puro. De 8, 16 ó 32 bits, son los formatos más extendidos.
  - Enteros con signo: complemento a dos.
- Reales
  - De simple precisión: 16 bits, de los cuales 8 para exponente y 8 para la mantisa.
  - Doble precisión: 32 ó 64 bits. 8 para el exponente y 24 para mantisa y 11 para exponente y 52 para mantisa respectivamente.

Para hacer un estudio de cómo se implementa una Unidad Aritmético Lógica (ALU a partir de ahora), estudiaremos las operaciones más importantes sobre cada uno de estos sistemas, así como las optimizaciones que podemos hacer sobre cada una de ellas.

Estructura y operaciones de una ALU



La imagen que aparece en pantalla representa la estructura de una unidad aritmética que, en general, se compone de uno o varios **operadores**, de un conjunto de **registros**, unos **biestables** para almacenar el estado y, en algunos casos, de un componente **secuenciador**.

Los operadores por los que está construido una ALU, pueden ser de dos tipos.

Operadores de tipo combinacional simple y/o de tipo paralelo	La unidad de control del computador se encarga de llevar a cabo el control de los algoritmos de las operaciones complejas. Es típica de dispositivos de bajo coste, donde se anteponen las condiciones económicas a la eficiencia.
Operadores con su propio órgano secuenciador	El órgano secuenciador se encarga de dirigir las fases necesarias para la realización de las operaciones que tenga encomendadas. Este órgano se compondrá de un contador de fases, de una lógica de decodificación de las fases y de un conjunto de registros para almacenamiento temporal de cálculos intermedios. Estos operadores suelen encontrarse en computadores muy potentes, a los que se les dota de una unidad aritmética compuesta por varios operadores que pueden funcionar en paralelo, anteponiendo la eficiencia a los demás factores.

Por su lado, el banco de registros de tipo general sirve para que el usuario almacene temporalmente datos y resultados intermedios. Suele constar del orden de 8 ó 16. En muchas máquinas, hay un registro en especial, llamado acumulador.

Acumulador

El acumulador recibe un trato privilegiado. Este registro se utiliza como almacén inicial/intermedio/final de los datos del operador y, sobre su contenido, se realizan muchas operaciones que no se pueden hacer sobre otros registros.

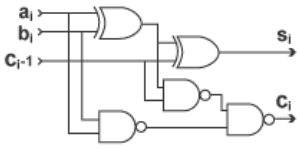
Finalmente, diremos que la unidad aritmética está generalmente dotada de unos biestables, que almacenan ciertas condiciones relativas a la última operación realizada por ella. Son muchas las condiciones que se pueden almacenar en este registro de estado, pero por su simplicidad y utilidad, destacamos los siguientes:

- Cero: se pone a 1 si el resultado es cero.
- Negativo: se pone a 1 si el resultado es negativo.
- Acarreo: se pone a 1 si el resultado tiene acarreo.
- Desbordamiento: se pone a 1 si el resultado tiene desbordamiento.

Denominación de los biestables de la unidad aritmética

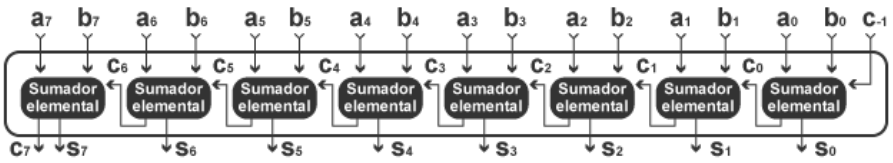
Los biestables de los que, generalmente, está dotada la unidad aritmética, se llaman **biestables de estado**.

Operaciones de suma y resta I



La suma/resta de números enteros es una de las operaciones simples que se realizan mediante lógica combinacional clásica. Y suele ser, en muchas ALU, la base de otras operaciones como la multiplicación o la división. La manera de construir un sumador, ya le hemos visto en otras asignaturas, por lo que sólo recordaremos a grandes rasgos cómo era su implementación.

Se basa en la combinación de sumadores elementales, es decir, sumadores de un solo bit (véase la imagen de la derecha) más acarreo, que se encadenan, para formar sumadores del número de bits que se desee (véase la imagen inferior).



Este método de construir un sumador es muy simple, pero el tiempo de suma, depende del número de bits que forman los números, ya que la suma tiene que ser secuencial. En la actualidad, hay sumadores con predicción de acarreo, que permiten hacer la suma de todos los bits de manera simultánea, acelerando considerablemente el tiempo de la suma. Estos sumadores se construyen gracias a una característica del acarreo, que puedes ser determinada solo, teniendo en cuenta los bits a_i,b_i de cada etapa.

Documentos Explicación del sumador con predicción de acarreo

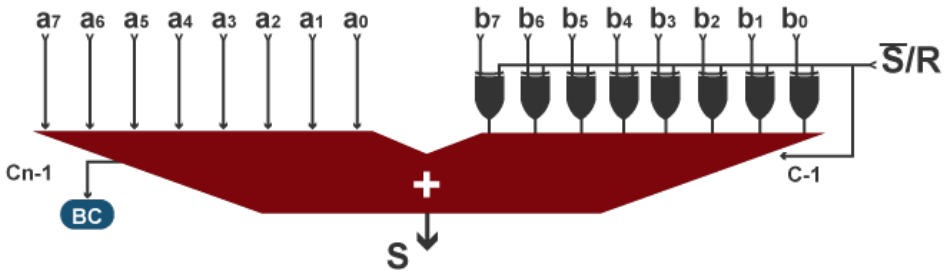
Operaciones de suma y resta II

Hasta ahora hemos hablado del sumador, pero ¿qué pasa con el restador? Como ya se ha mencionado, los números negativos en un computador actual, se implementan en complemento a dos. Este sistema tiene muchas ventajas, y una de ellas, quizás la más importante, es que **la conversión de positivo a negativo y viceversa** es muy simple (cambiar ceros por unos y unos por ceros, y después sumar uno). A priori, parece complicado, pero si se analiza bien y se tiene en cuenta que una operación de resta, se puede interpretar como:

X = A - B es lo mismo que X = A + (-B)

Todas las restas se convierten en sumas si podemos hacer el complemento a dos de B de manera rápida.

Analicemos este circuito:



Es un sumador en secuencial, al que al a cada operando $\mathbf{b_i}$ se le aplica una XOR junto con la señal S/R. Si S/R es 0, $\mathbf{b_i}$ no se modifica, pero si S/R es 1, $\mathbf{b_i}$ se invierte, además, si la señal S/R es 1, entonces el primer carry del sumador es 1, con lo que toda la suma se ve incrementado en 1.

En conclusión, si S/R es 0 el sumador se comporta como un sumador, mientras que si S/R es 0 el sumador se comporta como un restador.

Multiplicación de binario puro

Como ya sabemos, la tabla de multiplicar del 0 y del 1 en binario es la misma que en decimal. Por tanto, vamos a repasar una operación de multiplicar en binario.

Tal como se aprecia en la imagen, se realiza exactamente igual que en cualquier otro sistema de numeración (por ejemplo en decimal). Es decir:

- La multiplicación implica la generación de productos parciales, uno para cada bit del multiplicador.
- El producto total se obtiene sumando los productos parciales, desplazando cada uno un bit a la izquierda respecto al producto precedente.
- El producto de dos enteros sin signo de n bits da como resultado un número de hasta 2n bits (ej. 11 x 11 = 1001).

```
          1 0 1 1
        x 1 1 1 1
        ---------
          1 0 1 1
        0 0 0 0
      1 0 1 1
  1 0 1 1
---------------
1 0 0 0 1 1 1 1
```

Sería fácil escribir un programa que pudiese multiplicar dos números, usando sólo la suma. Este método lo utilizan algunos microcontroladores (microprocesadores de muy bajo coste), que para ahorrar costes, su ALU sólo implementa la suma/resta y falsea la multiplicación/división por software. Pero en este tema, nos interesa el caso contrario: cómo hacer un hardware que permita la multiplicación de dos números de manera óptima y rápida.

1/4

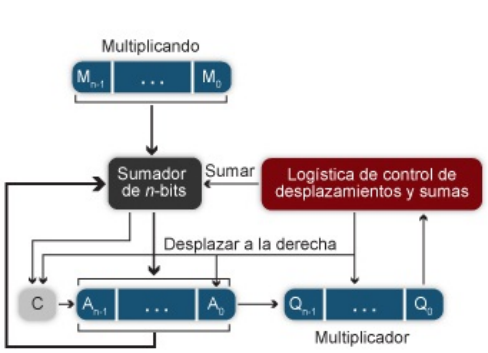


Diagrama de bloques del algoritmo de Booth

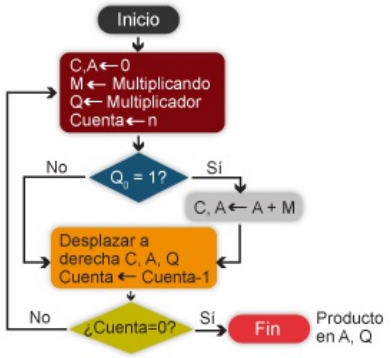


Diagrama de flujo para la multiplicación según el algoritmo de Booth

La solución a esto la implementa el algoritmo de Booth. El algoritmo funciona de la siguiente manera:
La lógica de control lee uno por uno los bits del multiplicador.

- Si Q0 es 1:
  - A = A + M
  - oC guarda el acarreo

2/4

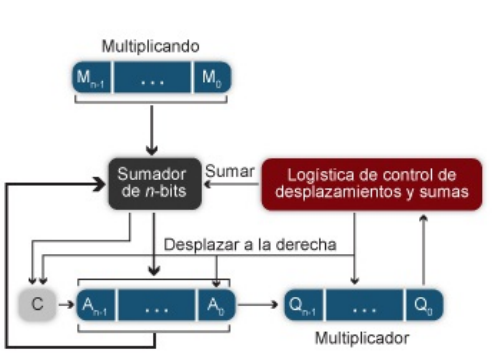


Diagrama de bloques del algoritmo de Booth

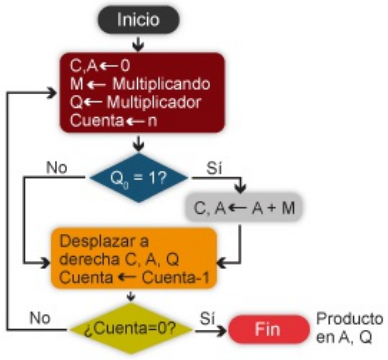


Diagrama de flujo para la multiplicación según el algoritmo de Booth

- Desplazamos una posición a la derecha C, A y Q, de manera que:
  - An-1 = C
  - Qn-a = A0
  - Q0 se pierde
- Repetimos el proceso para cada bit del multiplicando original
- El producto final queda en los registros A y Q.

3/4

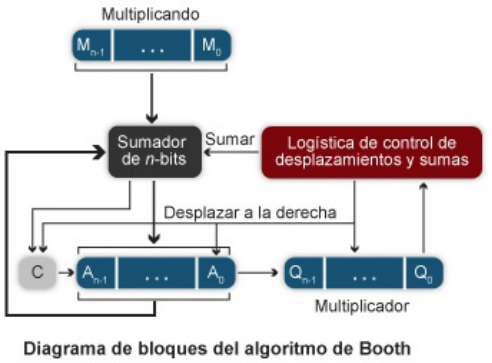


Diagrama de bloques del algoritmo de Booth

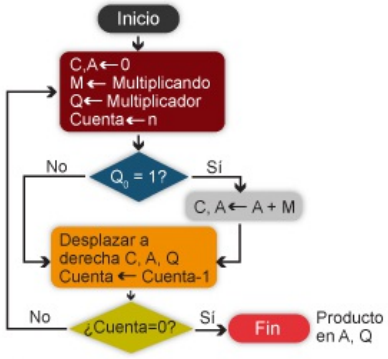


Diagrama de flujo para la multiplicación según el algoritmo de Booth

Si se analiza con detalle, se puede ver que este método es la implementación hardware del método clásico (manual) de la multiplicación. Pero ahorrando registros, de manera que el multiplicador y el resultado temporal (la suma parcial) comparten registro.
Si nos fijamos, lo que hace es ir sumando el multiplicando a la suma parcial (que inicialmente vale 0) cada ved que en el multiplicador hay un 1 y luego desplazar. Esto es justamente lo que hacemos en la multiplicación manual.

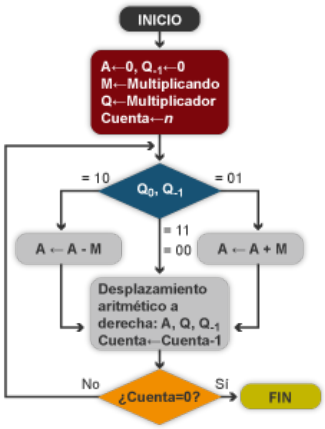
4/4

Documentos

Ejemplo de multiplicación de dos números sin signo

Multiplicación de enteros con signo

Para la representación de enteros negativos/positivos se utiliza complemento a dos. Debemos comprobar que el método de multiplicación de números positivos, no es válido para los negativos. No obstante, Booth ha simplificado esta tarea para desarrollar un método que permite la multiplicación de números positivos y negativos sin tener en cuenta el signo de los operandos. En la imagen,observamos que Booth ha escrito un algoritmo que tiene una traducción directa a hardware y cuyo principio de funcionamiento consiste en:



Complemento a dos

El complemento a dos lo repasamos en la pantallas de sumador/restador.

- El multiplicador y el multiplicando se ubican en los registros Q y M respectivamente.
- Se crea un registro de un bit con una ubicación lógica a la derecha del bit menos significativo (Q_0,) del registro Q, y que denominamos Q_{-1}.
- A y Q_{-1}, se fijan inicialmente a 0.
- Se comprueban Q_0 y Q_{-1}:
  - Si los dos son iguales (1-1 ó 0-0), todos los bits de los registros A, Q, y Q_{-1} se desplazan un bit a la derecha.
  - Si dichos bits difieren, el multiplicando se suma o se resta al registro A, según que los dos bits sean 0-1 ó 1-0. A continuación de la suma o resta se realiza un desplazamiento a la derecha.
- Debemos tener en cuenta queel desplazamiento a derecha es un desplazamiento aritmético,es decir, el bit más significativo de A (A_{n-1}) se copia en A_{n-2}, pero A_{n-1} mantiene su valor. A este efecto se le conoce como **rellenar con el bit de sigo**, y se utiliza para preservar el signo del número contenido en la pareja de registros A y Q.
- Se repite esta secuencia hasta que se hayan procesado todos los bits de multiplicador.

Solución para multiplicar números negativos

Para solucionar este problema, podemos tomar una solución muy similar al caso del Sumador/restador. Que consistiría en pasar los números negativos a positivos, realizar la multiplicación y luego corregir el signo del resultado. Este método, aunque poco eficiente, es utilizado por multitud de microcontroladores.

Ejemplo de algoritmo de Booth para números con signo

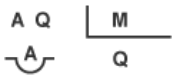
La división de enteros

Existe una simple traducción del método de división manual a un algoritmo, y de él, a un hardware que lo implementa. El método utilizado tiene adecuaciones a una traducción a hardware. Consisten en una lógica de control para hacer las distintas iteraciones así como un desplazamiento a la izquierda, para simular los desplazamientos a la derecha de las distintas restas.

Este algoritmo funciona de la siguiente manera:

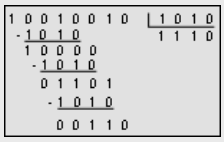
- Ponemos el divisor en M y el dividendo en Q e inicializamos A a ceros.
- Las iteraciones comienzan con, A y Q desplazándose 1 bit a la izquierda.
- Restamos A = A - M para ver si A divide el resto parcial (resta de enteros sin signo).
  - Si se cumple: A mayor o igual a 0, $Q_0 = 1$
  - Si no se cumple: A negativo, $Q_0 = 0$ y A = A + M(se restaura al valor anterior a la resta)
- Reducimos la cuenta y el proceso se repite n pasos.
- Terminadas las iteraciones, el cociente queda en Q y el resto en A.

Por tanto, la operación de división se realiza según los registros:



Método de división manual

Vamos a recordar la división manual:



Primero se ajusta mentalmente el divisor con el dividendo, comprobando al mismo tiempo si cabe. En caso afirmativo, se resta, se pone un 1 en el cociente y se baja el siguiente dígito. En caso negativo, se pone un 0 en el cociente y se corre mentalmente el divisor un lugar a la derecha hasta que se agotan los dígitos del dividendo. Dado que en el sistema binario sólo existen dos alternativas, si el dividendo es mayor que el divisor, entonces cabe a uno, en caso contrario, no cabe. En un computador, esta función se puede hacer mediante una resta, si el resultado de la resta es positivo, es que cabe, si es negativo, no cabe. Obsérvese que al contrario que en la multiplicación, un dividendo de 2n bits, generan un cociente y un resto de n bits.

Documentos Ejemplo de división sin signo

division_de_signo.pdf

Crear un documento pdf a partir del documento division_de_signo.docx que se encuentra en la carpeta de documentos. Hay que rehacer a estilo UEM la imagen dg_0012.png que se encuentra en el directorio de imágenes

Desplazamiento de las restas

Si no movemos la resta y desplazamos los operadores a la izquierda, es lo mismo que no mover los operadores y desplazar la resta a la derecha.

División de enteros con signo

La división con signo, no es muy diferente de la división sin signo. Si algoritmo es el que se ve en la figura y se interpreta de la siguiente manera:

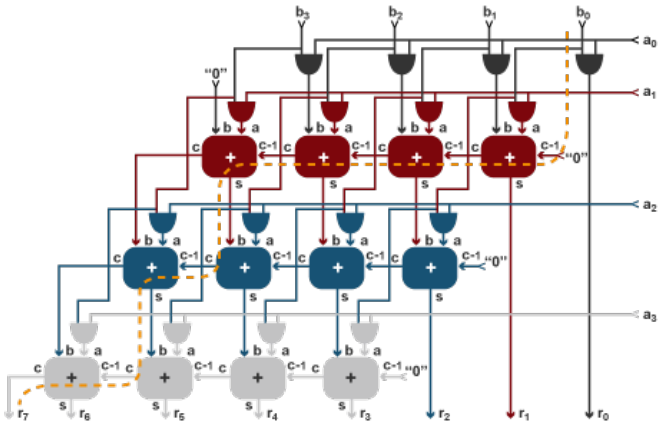
- Cargar el divisor en M y el dividendo en A y Q. El dividendo debe estar expresado en complemento a dos de 2n bits.
- Desplazar A-Q una posición a la izquierda.
- Comparar el signo de A y M:
  - Si M y A tienen el mismo signo: A ← A - M.
  - Si M y A tienen distinto signo: A ← A + M.
- La operación anterior tiene éxito si el signo de A es igual antes y después de la operación o A es cero (0).
  - Si la operación tiene éxito, entonces hacer Q_0= 1.
  - Si la operación no tiene éxito, entonces Q_0 = 0 y restablecer el valor inicial de A.
- Repetir los pasos 2 a 4 tantas veces como bits tenga Q.
- La final del algoritmo:
  - El resto está en A.
  - Si los signos del dividendo y el divisor eran iguales, el cociente está en Q; si no, el cociente correcto es el complemento a dos de Q.

El **fallo de este método se encuentra en el cociente**, cuyo signo debemos corregir debido a que siempre es positivo, de manera que su dividendo y divisor son de signos opuestos. Por el contrario, el signo del resto es correcto.

Documentos Ejemplo de división con signo

Multiplicación de rápidos

Dado que los algoritmos que hemos visto son circuitos combinacional multi-ciclo, el tiempo de ejecución de pende del reloj de la ALU, por lo que puede hacer lentas estas operaciones.



Existen modificaciones de estos circuitos, conocidos como multiplicadores o divisores rápidos. Se caracterizan por construir un circuito puramente combinacional no iterativo que realice directamente la suma de los productos parciales.

Un ejemplo de uno de estos circuitos lo podemos ver en la siguiente imagen.

En este tipo de multiplicadores, los productos parciales son inmediatos. El problema reside con los sumadores de estos productos, que a pesar de que sólo necesiten 2n sumadores (siendo n el numero de bits de los operandos), lo que es un número muy aceptable para una implementación física, estos sumadores son los que provocan el retraso del cálculo de la multiplicación.

El retardo viene dado porque los sumadores están colocados en cascada. No se puede calcular la suma de un elemento si antes no se ha calculado la suma parcial anterior. Extrapolando esta idea, el retardo total de estos sumadores es considerable, ya que viene determinado por el número de sumadores en cascada (en la imagen se representa mediante una línea discontinua), que para el ejemplo es 8 niveles.

La fórmula general es n+2(n-2). Así, para un multiplicador de 8 bits es de 8+2·(8-2)= 20 niveles.

Productos parciales inmediatos

Los productos parciales inmediatos se obtienen haciendo una AND entre cada uno de los dígitos de B con cada uno de los dígitos de A en una matriz de AND's.

Operaciones con números reales - coma flotante

A continuación puedes ver algoritmos para suma/resta y multiplicación/división en operaciones en punto flotante.

Documentos Algoritmos

En este tema no vamos a tratar estos algoritmos, pero sí es importante destacar con qué problemas se han encontrado los diseñadores para implementar una unidad Aritmética en punto flotante, y por qué hasta no hace muchos años, esta venía como un coprocesador matemático externo al microprocesador.

La doble representación del cero, la representación de números especiales cono NaN (Not a Number = no es un número), el +∞(infinito) el -∞, etc., son condiciones que hay que procesar por separado, por lo que antes de empezar a operar, hay que determinar casos especiales.

Un caso especial es la precisión. Este error lo conoce bien Intel, ya que su microprocesador insignia, el Pentium (año 1994) tenía un error de precisión en su unidad de división de punto flotante.

Un ejemplo de precisión lo podemos encontrar siempre que trabajamos en física con magnitudes reales.

Según la norma IEEE 754, el estándar de punto flotante utiliza 2 bits de guarda a la derecha para los resultados intermedios. Ejemplo: A= 12 y B = 2.75.

En punto flotante con 1 bit de signo, 3 para el exponente y 3 para la mantisa se escriben	• A = 1100 = $1.100 \cdot 2^3$ = 0 110 100 • B = 10.11 = $1.011 \cdot 2^1$ = 0 100 011. Igualamos exponentes: B = 0 110 010

Suponiendo que no hay bits de guarda ni redondeo	• A + B = 0 110 110 = $1.110 \cdot 2^3 = 1110_2 = 14_{10}$. Error acumulado de 0.75_{10}
Si ponemos 2 bits de guarda	• A = 1100 = $1.100 \cdot 2^3$ = 0 110 10000 (los subrayados son los bits de guarda) • B = 10.11 = $1.011 \cdot 2^1$ = 0 100 01100 Normalizamos exponentes: B = 0 110 01011 • A+B = 0 110 11011. • Redondeamos para eliminar los bits de guarda: A + B = 0 110 111 • A+B = 0 110 111 = $1.111 \cdot 2^3 = 1111_2 = 15_{10}$. Error de 0.25_{10}.

Algoritmos para operaciones en punto flotante

Esta información no es un tema a tratar en esta unidad, ya que las implementaciones hardware que lo resuelven son complejas. Tanto, que realmente se construyen como una máquina secuencia y no como un circuito combinacional como los vistos hasta ahora.

Curiosidad sobre los errores de precisión de Intel

Intel nunca admitió el error, pero si el usuario demostraba que su microprocesador fallaba (cosa fácil, ya que había software que chequeaba este error en detalle), Intel se lo reemplazaba sin coste alguno.

Ejemplo de precisión en física

Estamos acostumbrados a utilizar una precisión más alta en los valores intermedios, para que el error acumulado con las consiguientes operaciones, tenga un error despreciable.Si no hacemos eso, los errores por truncamiento se van propagando y después de varias operaciones, el error acumulado es grande. Así, por ejemplo:

- Supongamos que queremos calcular $2.56 \cdot 10^0 * 2.34 \cdot 10^2$, utilizando solo tres cifras significativas.
  - Resultado: $2.56 \cdot 10^0 + 2.34 \cdot 10^2 = 0.02 \cdot 10^2 + 2.34 \cdot 10^2 = 2.58 \cdot 10^2$
- Pero en vez de tres dígitos, ampliamos con solo dos dígitos más, cinco dígitos:
  - Resultado: $0.0256 \cdot 10^2 + 2.34 \cdot 10^2 = 2.3656 \cdot 10^2$. Redondeamos a tres dígitos: $\approx 2.37 \cdot 10^2$.

Resumen

En este tema, hemos podido ver:

- Cuáles son las funciones más importantes de una **Unidad Aritmético Lógica**. Durante el desarrollo del tema hemos pasado por alto las acciones lógicas, ya que por su simplicidad apenas tienen interés, y además, porque se han estudiado en temarios anteriores.
- En cambio, hemos prestado especial interés a las acciones más complejas, como la suma/resta o la multiplicación/división de números enteros, con y sin signo.
- Dejamos planteado el problema de las operaciones aritméticas sobre números reales y las complejidades que esta tendría.
- Hemos visto desde los algoritmos utilizados, hasta la implementación hardware de los elementos combinacionales (secuenciales en los casos más complicados) para poder realizar una Unidad Aritmética de altas prestaciones. Esta Unidad Aritmética es igual a la que implementan hoy en día la mayor parte de los microprocesadores más utilizados en los computadores actuales.