Estructura de Computadores

Tema 6. La unidad aritmética y lógica

- Operaciones típicas de la unidad aritmético-lógica.
- Algoritmos de multiplicación de Robertson y de Booth.
- Algoritmos de división con y sin reemplazamiento.

---

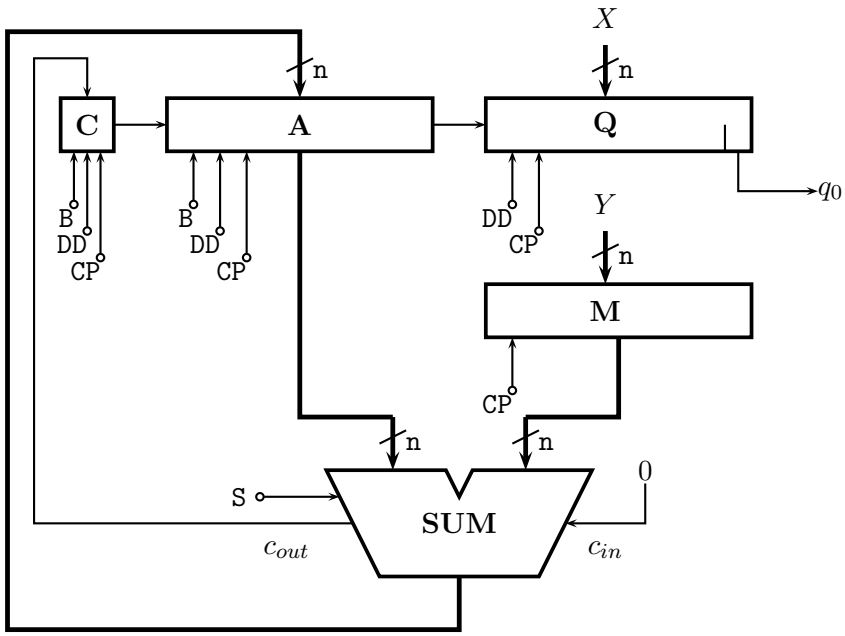
1. Ejercicios Resueltos

1.1.

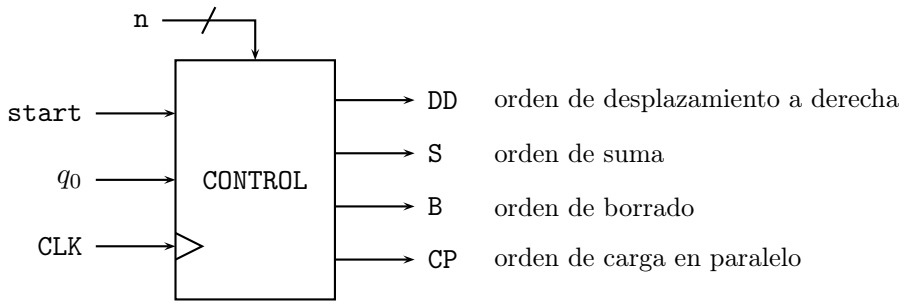
Realizar la traza de la multiplicación de los números 43 y 13 de un multiplicador secuencial sin signo, dibujando la estructura hardware correspondiente. Modificar la estructura anterior para que se puedan multiplicar números con signo representados en complemento a dos, detallando el algoritmo empleado.

Solución

El multiplicador secuencial sin signo emplea la siguiente estructura *hardware*:
En primer lugar el subsistema de datos.



y en segundo lugar el subsistema de control:



El algoritmo que realiza es:

0: $\rightarrow 0$ **if** $\overline{\texttt{start}}$;
1: $\texttt{Q} \leftarrow X \parallel \texttt{M} \leftarrow Y \parallel \texttt{C}\circ\texttt{A} \leftarrow 0 \parallel \texttt{P} \leftarrow n$;
2: **if** $(q_0 = 1)$ **then** $\texttt{A} \leftarrow \texttt{A} + \texttt{M} \parallel \texttt{C} \leftarrow c_{out}$;
3: $0\circ\texttt{C}\circ\texttt{A}\circ\texttt{Q} >> 1 \parallel \texttt{P}{-}{-}$;
4: $\rightarrow 2$ **if** $\texttt{P} > 0$;

Si $X = 43_d = 101011_{bin}$ e $Y = 13_d = 1101_{bin}$
se necesitan registros de 6 bits al menos: $n = 6$.

En forma de tabla vemos la evolución de los registros:

P	C	A	Q		M
6	0	000000	101011	$q_0 = 1$	001101
	0	001101	101011		
5	0	000110	110101	$q_0 = 1$	
	0	010011	110101		
4	0	001001	111010	$q_0 = 0$	
3	0	000100	111101	$q_0 = 1$	
	0	010001	111101		
2	0	001000	111110	$q_0 = 0$	
1	0	000100	011111	$q_0 = 1$	
	0	010001	011111		
0	0	001000	101111		

El resultado es correcto en $\boxed{\texttt{A}\circ\texttt{Q}} = 001000101111_{bin} = \texttt{22Fh} = 559_d$

Para multiplicar números con signo en representación de C2 veamos con detalle este tipo de representación:

Sea X un número de n bits en C2, $X = (x_{n-1}, x_{n-2}, \ldots, x_1, x_0)$, si x_{n-1} es el bit de signo entonces el valor del número es:

$$\text{si } x_{n-1} = 0 \text{ entonces es } v(X) = \sum_{i=0}^{n-2} x_i \cdot 2^i,$$

$$\begin{aligned}\text{si } x_{n-1} = 1 \text{ es } v(X) &= -\left[(1,\ldots,1) - (x_{n-1},\ldots,x_0) + 1\right] \\ &= -\left[(2^n - 1) - \sum_{i=0}^{n-1} x_i \cdot 2^i + 1\right]\end{aligned}$$

O sea, si $x_{n-1} = 1$

$$\begin{aligned} v(X) &= -2^n + 1 + \sum_{i=0}^{n-1} x_i \cdot 2^i - 1 \\ &= -2^n + 1 \cdot 2^{n-1} + \sum_{i=0}^{n-2} x_i \cdot 2^i \\ &= -2^{n-1} + \sum_{i=0}^{n-2} x_i \cdot 2^i \end{aligned}$$

O lo que es lo mismo:

$$\boxed{v(X) = -x_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} x_i \cdot 2^i}$$

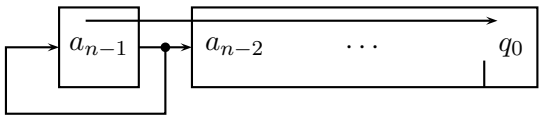
Lo único que se diferencia del caso de números positivos de n bits es que el bit más significativo es negativo. Por tanto, con el mismo esquema *hardware* anterior pero con un sumador–restador en lugar del sumador, y con un algoritmo semejante, con la única diferencia que es restar el producto parcial por el último bit, el multiplicador puede emplearse para números con signo en C2.

El algoritmo puede ser tal que así:

0: $\rightarrow 0$ **if** $\overline{\texttt{start}}$;
1: $\texttt{Q} \leftarrow X \parallel \texttt{M} \leftarrow Y \parallel \texttt{A} \leftarrow 0 \parallel \texttt{P} \leftarrow n-2$;
2: **if** $(q_0 = 1)$ **then** $\texttt{A} \leftarrow \texttt{A} + \texttt{M}$;
3: $a_{n-1} \circ \texttt{A} \circ \texttt{Q} >> 1 \parallel \texttt{P}--$;
/* desplazamiento aritmético a la derecha, con extensión del signo */
4: $\rightarrow 2$ **if** $\texttt{P} \geq 0$;
5: **if** $(q_0 = 1)$ **then** $\texttt{A} \leftarrow \texttt{A} - \texttt{M}$;
6: $a_{n-1} \circ \texttt{A} \circ \texttt{Q} >> 1$;

El bit de acarreo, C, ya no es necesario pues las sumas y resta de n bits se hacen en C2, y c_{out} se desprecia.

Los desplazamientos a la derecha se realizan con extensión del signo (desplazamientos aritméticos).



Los mismos números de antes pueden servir para realizar una traza.

Sean $X = 101011_{bin} = -21_d$ e $Y = 001101_{bin} = +13_d$, con $n = 6$.

En forma de tabla vemos la evolución de los registros:

P	A	Q		M
4	000000	101011	$q_0 = 1$	001101
	001101	101011		-M
3	000110	110101	$q_0 = 1$	110011
	010011	110101		
2	001001	111010	$q_0 = 0$	
1	000100	111101	$q_0 = 1$	
	010001	111101		
0	001000	111110	$q_0 = 0$	
-1	000100	011111	$q_0 = 1$	
	110111	011111		
	111011	101111		

El resultado es correcto en $\boxed{\texttt{A} \circ \texttt{Q}} = 111011101111_{bin}$, negativo, luego $\boxed{-\texttt{A} \circ \texttt{Q}} = 000100010001_{bin} = \texttt{111h} = 273_d$.
Producto = (-21)×13 = -273.

1.2.

Efectuar la siguiente multiplicación en complemento a 2: $110011 * 110110$ $(-13 * -10)$ utilizando

(a) el algoritmo de Booth con codificación de dos bits,

(b) el algoritmo de Booth con codificación de tres bits.

Solución

(a)

A	110011	-13
X	×110110	-10
Y	0 -1 1 0 -1 0	multiplicador recodificado
Sólo desplaza	0000000	
Suma -A	+001101	
	0011010	
Desplaza	00011010	
Sólo desplaza	000011010	
Suma +A	+110011	
	110110010	
Desplaza	1110110010	
Suma -A	+001101	
	0010000010	
Desplaza	00010000010	
Sólo desplaza	000010000010	= 130

(b)

A	110011	-13
X	×110110	-10
Y	-1 2 -2	multiplicador recodificado
Sólo desplaza	0000000	
Suma -2A	+0011010	
Desplaza dos bits	000011010	
Suma +2A	+1100110	
	110110010	
Desplaza dos bits	11110110010	
Suma -A	+0001101	
	00010000010	
Desplaza dos bits	0000010000010	= 130

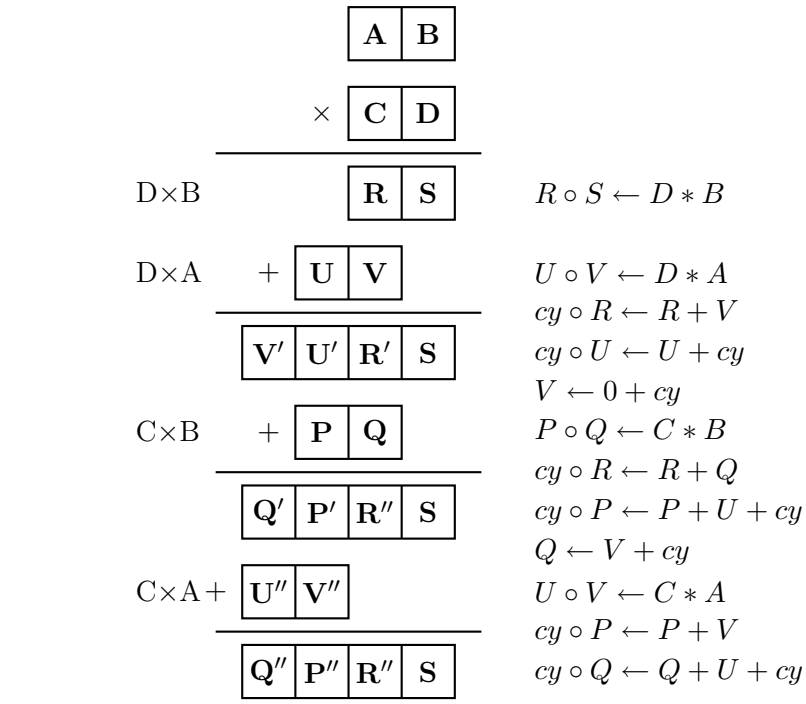
1.3.

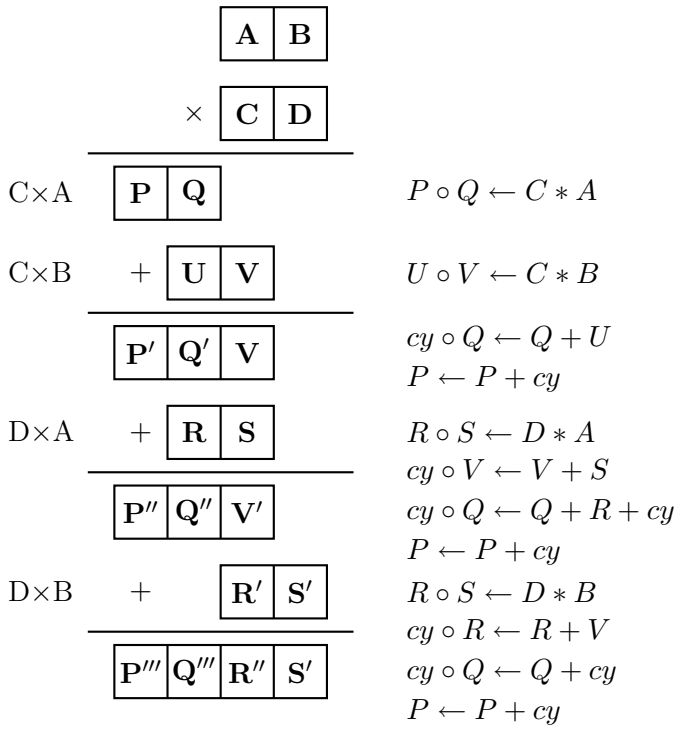
Una unidad aritmética de enteros que puede realizar sumas y productos de números sin signo de 16 bits, se quiere utilizar para multiplicar dos números sin signo de 32 bits. Los operandos, resultados intermedios y los resultados finales, se almacenan en registros de 16 bits etiquetados R_0 a R_{15}. El hardware del multiplicador obtiene el producto de R_i (multiplicando) por R_j (multiplicador) y memoriza el resultado de 32 bits en el par de registros R_j y R_{j+1}, con la mitad menos significativa en R_j. Cuando $j = i - 1$, el producto sobreescribe los dos operandos. El sumador hardware suma el contenido de los registros R_i y R_j y pone el resultado en R_j. El acarreo de entrada para una operación *Suma* es 0, y para una operación *Suma_ con_ acarreo* es el contenido de un indicador de acarreo C. El acarreo de salida del sumador se almacena siempre en C.

Especificar las etapas de un procedimiento para efectuar el producto de dos operandos de 32 bits almacenados en los registros R_1, R_0 y R_3, R_2, tratando primero las mitades de orden superior, y dejando el producto resultante, de 64 bits, en los registros R_{15}, R_{14}, R_{13} y R_{12}. Si es necesario, para los valores intermedios puede usarse cualquiera de los registros R_{11} a R_4. Cada paso del procedimiento puede ser una multiplicación, o una suma, o una operación de transferencia entre registros.

Solución

Los siguientes esquemas muestran la forma de realizar una multiplicación de doble precisión, utilizando parejas de registros para contener a los operandos de longitud doble. En el primer esquema se empieza multiplicando las partes menos significativas, en tanto que en el segundo esquema se empieza por las partes más significativas. El número de operaciones en cada caso es el mismo así que es indiferente elegir una u otra opción.





Las instrucciones que se describen en el enunciado son:

1. La instrucción de suma

$$\texttt{Suma}\quad \texttt{R}_i\texttt{,R}_j \quad ; C \circ R_j \leftarrow R_i + R_j$$

2. y de suma con acarreo

$$\texttt{Suma\_con\_acarreo} \quad \texttt{R}_i\texttt{,R}_j \quad ; C \circ R_j \leftarrow R_i + R_j + C$$

3. La de multiplicación la expresamos así:

$$\texttt{Multiplica} \quad \texttt{R}_i\texttt{,R}_j \quad ; R_{j+1} \circ R_j \leftarrow R_i \times R_j$$

4. Se necesitan también instrucciones de transferencia entre registros:

$$\texttt{Pasa} \quad \texttt{R}_i\texttt{,R}_j \quad ; R_j \leftarrow R_i$$

5. y por último alguna forma de poner a cero un registro, que puede ser así

$$\texttt{Borra} \quad \texttt{R}_i \quad ; R_i \leftarrow 0$$

   o bien así

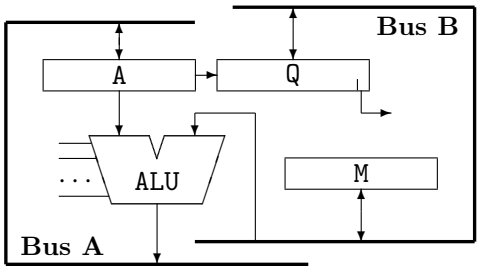
$$\texttt{Resta} \quad \texttt{R}_i\texttt{,R}_i \quad ; C \circ R_i \leftarrow R_i - R_i$$

Ahora se puede escribir el procedimiento de multiplicación de doble precisión:

```
1  Procedimiento ProductoDoblePrecision
2          Pasa   R3,R14              ;R14 ← R3
3          Multiplica   R1,R14        ;R15 ∘ R14 ← R1 * R14
4          Pasa   R3,R4               ;R4 ← R3
5          Multiplica   R0,R4         ;R5 ∘ R4 ← R0 * R4
6          Borra   R7                 ;R7 ← 0
7          Suma   R5,R14              ;C ∘ R14 ← R5 + R14
8          Suma_con_acarreo   R7,R15  ;C ∘ R15 ← R7 + R15 + C
9          Pasa   R2,R5               ;R5 ← R2
10         Multiplica   R1,R5         ;R6 ∘ R5 ← R1 * R5
11         Suma   R5,R4               ;C ∘ R4 ← R5 + R4
12         Suma_con_acarreo   R6,R14  ;C ∘ R14 ← R6 + R14 + C
13         Suma_con_acarreo   R7,R15  ;C ∘ R15 ← R7 + R15 + C
14         Pasa   R2,R12              ;R12 ← R2
15         Multiplica   R0,R12        ;R13 ∘ R12 ← R0 * R12
16         Suma   R4,R13              ;C ∘ R13 ← R4 + R13
17         Suma_con_acarreo   R7,R14  ;C ∘ R14 ← R7 + R14 + C
18         Suma_con_acarreo   R7,R15  ;C ∘ R15 ← R7 + R15 + C
19 FinProcedimiento
```

1.4.

Se quiere emplear al sistema de flujo de datos de la figura para realizar el algoritmo de *Robertson* de multiplicación de números binarios enteros con signo en representación de complemento a dos (C2). Los registros son de `n` bits.



(a) Especificar todas las señales de control que se precisen en el sistema.

(b) Expresar el algoritmo como secuencia de microinstrucciones.

(c) Realizar la traza para la operación $87 \times (-75)$ con `n`$= 8$ bits.

Solución

El algoritmo de multiplicación secuencial con la corrección del último paso se conoce como algoritmo de Robertson:

$$
\begin{array}{ll}
P^{(0)} = 0 \ ; & \#\ 1 \\
\textbf{for} \ \ j = 0 \ \ \textbf{to} \ \ n-2 \ \textbf{do} & \#\ 2 \\
\quad P^{(j+1)} = P^{(j)} + x_j \cdot M \ ; & \#\ 3 \\
\quad P^{(j+1)} = P^{(j+1)} \cdot 2^{-1} \ ; & \#\ 4 \\
P^{(n-1)} = P^{(n-1)} - x_{n-1} \cdot M \ ; & \#\ 5
\end{array}
$$

En (# 3) si $x_j = 1$ se suma el multiplicando M al producto parcial. La suma es entre dos números con signo de n bits, y el acarreo se desprecia.

En (# 4) se desplaza el producto parcial completo un bit a la derecha, desplazamiento aritmético, con arrastre del signo. En $P^{(j+1)}$ quedan n bits, los restantes bits que han ido saliendo por la derecha ya no participan en la suma, pero sí en los desplazamientos.

En (# 5) si $x_{n-1} = 1$ (multiplicador negativo) se corrige $P^{(n-1)}$ restándole M. Si $x_{n-1} = 0$ (multiplicador positivo) no hay nada que corregir.

El resultado queda de la siguiente forma: n bits en $P^{(n-1)}$, con el bit de signo correcto en el bit más significativo, y los restantes $n-1$ bits a la derecha de $P^{(n-1)}$. Lo habitual es trabajar con registros de n bits y por lo tanto los productos parciales $P^{(j)}$ y los bits que se han desplazado a la derecha (la cola) se guardarán en registro de n bits. En el enunciado de este ejercicio se especifican los registros `A` para el producto parcial de n bits (los $P^{(j)}$) y `Q` para la "cola". El multiplicando en su propio registro `M` de n bits, y el multiplicador se guarda inicialmente en `Q`.

Al final, en `A` están los n bits más significativos del producto y en `Q` los $n-1$ bits restantes alineados a la izquierda. Los sucesivos desplazamientos, $(n-1)$ en total, han ido empujando al multiplicador hacia fuera de `Q`, saliendo por su extremo derecho. En cada etapa el bit q_0, el bit menos significativo de `Q`, contiene al bit x_j, que decide si se suma o no `M` a `A`. Al final queda x_{n-1} en q_0. Lo habitual es realizar un desplazamiento adicional para alinear el producto

$$U = M \cdot X$$

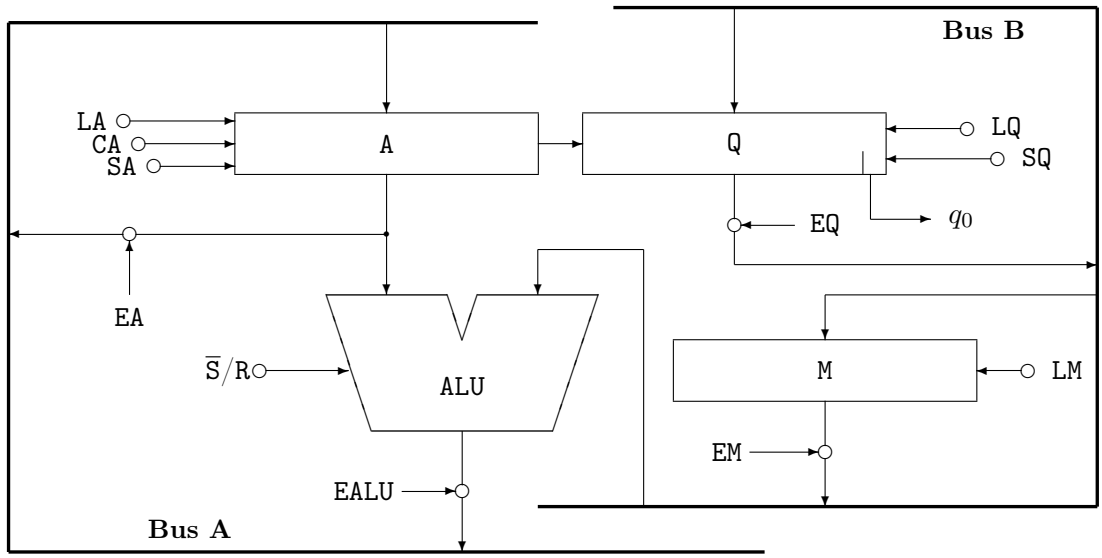
En lenguaje de transferencias de registro el algoritmo puede expresarse así:

```
M ← multiplicando (a través del bus B);
Q ← X (multiplicador, a través del bus B) ||
A ← 0; {se carga el 0 o el registro tiene una orden "CLEAR"}
Repetir n − 1 veces
      Si q0 entonces A ← A + M;
      A∘Q >> 1; {con extensión del signo}
Fin Repetir; {se empleará un contador para esto}
Si q0 entonces A ← A − M;
A∘Q >> 1; {con extensión del signo}
```

(a) A la vista del algoritmo planteado se requieren las siguientes operaciones (*micro-operaciones*) sobre los registros y la ALU del esquema:

- sobre M: `LOAD` desde el bus B
  `READ` hacia el bus B y de allí al operando 2 de la ALU
- sobre A: `CLEAR` puesta a cero
  o bien `LOAD` la constante 0
  `LOAD` desde la salida de la ALU a través del bus A
  `SHR` desplaza un bit a la derecha y no se pierde ningún bit, duplicando el signo
  `READ` hacia la ALU y hacia el bus A
- sobre Q: `LOAD` desde el bus B
  `READ` hacia el bus B
  `SHR` desplaza un bit a la derecha
  El bit q_0 siempre visible.
- sobre la ALU: operaciones de `SUMA` y `RESTA`, poniendo el resultado en el bus A.

A continuación se añaden al esquema original las señales de control.



En la unidad de control se admite la presencia de un contador capaz de contar $n-1$ pasos al menos, es decir con $k = \lceil \log_2(n-1) \rceil$ bits. Este contador debe tener una salida `CO` de cuenta a cero, unas entradas en paralelo de k bits para iniciar la cuenta, la señal de reloj y la de habilitación de cuenta.

Un total de 11 señales de control:

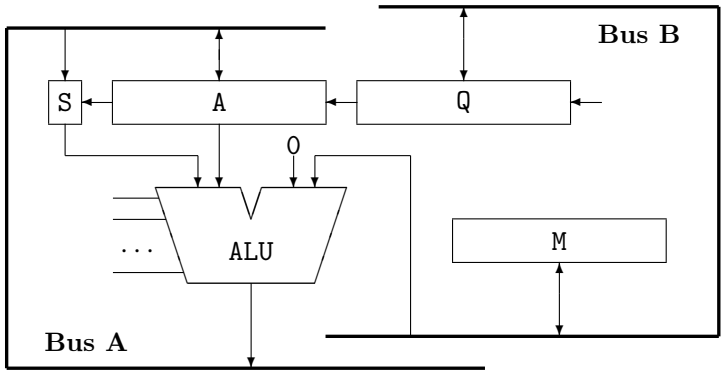
LA	CA	SA	EA	LQ	SQ	EQ	LM	EM	$\overline{\text{S}}$/R	EALU

(b) El algoritmo queda así (se muestran las señales activas):

1:	M $\leftarrow$ multiplicando $\parallel$ COUNT $\leftarrow (n-1)$;	LM
2:	Q $\leftarrow$ multiplicador $\parallel$ A $\leftarrow 0$;	LQ, CA
3:	**if** q_0 $\mid$ A $\leftarrow$ A + M ;	EM, $\overline{\text{S}}$/R $= 0$, EALU, LA
4:	A$\circ$Q $>> 1$ $\parallel$ COUNT$--$ $\parallel$**if** $\overline{\text{CO}} \rightarrow 3$;	SA, SQ
5:	**if** q_0 $\mid$ A $\leftarrow$ A − M ;	EM, $\overline{\text{S}}$/R $= 1$, EALU, LA
6:	A$\circ$Q $>> 1$;	SA, SQ

1.5.

Se quiere emplear al sistema de flujo de datos de la figura para realizar el algoritmo de división sin restauración de números binarios enteros sin signo. Los registros A, Q y M son de n bits, ampliándose A con un bit de signo adicional S. La ALU realiza las operaciones sobre números con signo de $n+1$ bits.



(a) Especificar todas las señales de control que se precisen en el sistema.
(b) Expresar el algoritmo como secuencia de microinstrucciones.

Solución

El algoritmo de la división sin restauración es el siguiente:

1:	$R \leftarrow X$;
2:	$R \leftarrow 2 \cdot R - D^*$;
3:	**for** $j = 1$ **to** $n-1$ **do**
4:	**if** $R \geq 0$ **then** $\{q_{n-j} \leftarrow 1 \parallel R \leftarrow 2 \cdot R - D^*\}$
	else $\{q_{n-j} \leftarrow 0 \parallel R \leftarrow 2 \cdot R + D^*\}$;
5:	**if** $R < 0$ **then** $\{q_0 \leftarrow 0 \parallel R \leftarrow R + D^*\}$;
	else $\{q_0 \leftarrow 1\}$;

donde X es el dividendo, de $2n$ bits, y D es el divisor de n bits; $D^* = 2^n \cdot D$, es el dividendo alineado a la izquierda. El algoritmo permite obtener bit a bit al cociente $Q = q_{n-1}q_{n-2}\ldots q_1q_0$, secuencialmente, los $n-1$ bits más significativos en la iteración de los pasos (3:) y (4:), y el

último bit, q_0, en el paso (5:). El resto parcial, R, que se inicializa con el valor del dividendo en el paso (1:) contiene al final el resto definitivo de la división, compliéndose

$$X = D \cdot Q + R, \qquad R < D.$$

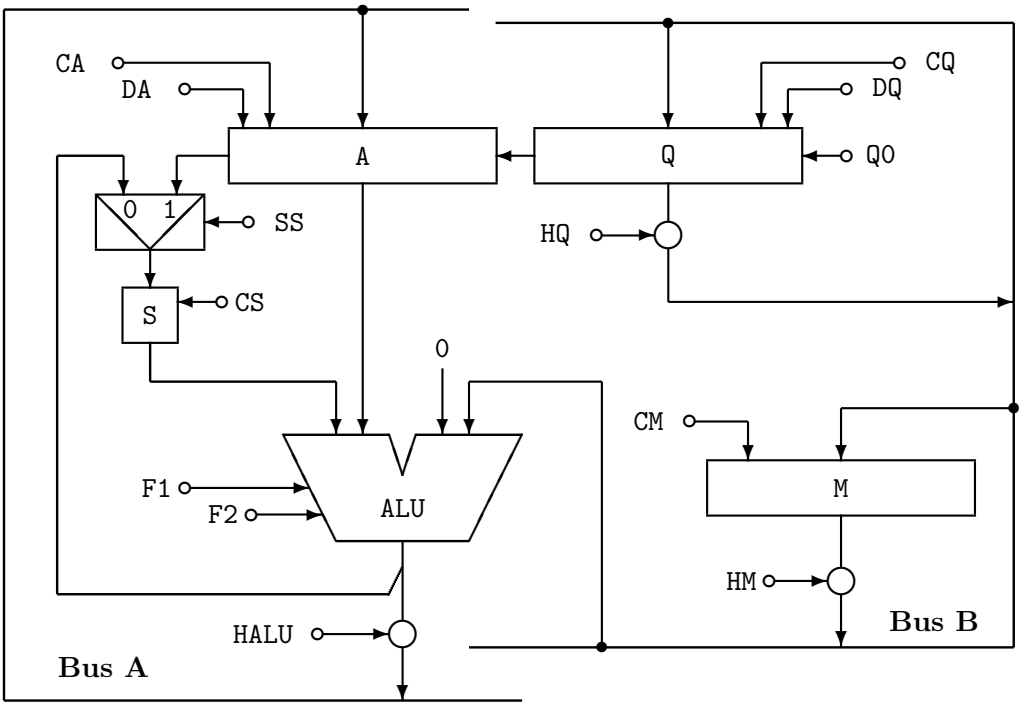
En el camino de datos propuesto en el enunciado del ejercicio los registros `A` y `Q` son registros de desplazamiento a la izquierda, cada uno de n bits y concatenados entre sí. En esta pareja de registros entra inicialmente el dividendo X, que hace de resto parcial inicial. Las operaciones del tipo $R \leftarrow 2 \cdot R \pm D^*$ que se hacen en los pasos (2:) y (4:) del algoritmo exigen desplazar a la izquierda un bit y sumar o restar el divisor alineado con los n bits más significativos. A continuación hay que comprobar el signo del resto parcial resultante, lo que quiere decir que, aunque los operandos sean números sin signo, esta operación exige hacerla con signo. Por eso se añade un bit adicional, el `S`, que amplía a `A` hasta los $n+1$ bits. La ALU, capaz de sumar y restar, debe ser también de $n+1$ bits.

El divisor D se guarda en el registro `M`, de n bits, para que esté disponible a la entrada de la ALU, donde se completa con el bit de signo adicional a cero.

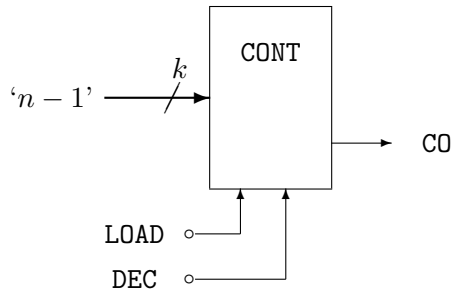
El registro `S`, de 1 bit, recibe datos desde `A`, cuando se multiplica R por 2, y también desde la ALU, tras sumar o restar; habrá que poner un multiplexor adicional a la entrada de `S`.

Se admite que la unidad de control tiene un contador capaz de contar los $n-1$ pasos de la iteración.

(a) Con las señales de control necesarias el sistema de datos queda así:



En la unidad de control se admite la presencia de un contador capaz de contar $n-1$ pasos al menos, es decir con $k = \lceil \log_2(n-1) \rceil$ bits. Este contador debe tener una salida `CO` de cuenta a cero, unas entradas en paralelo de k bits para iniciar la cuenta, la señal de cargar la cuenta inicial y la de decrementar el contador.



Un total de 13 señales de control:

DA	CA	SS	CS	CQ	DQ	HQ	Q0	CM	HM	F1	F2	HALU

(b) El algoritmo queda así (se muestran las señales activas):

1:	A $\leftarrow X_H$ ‖ Q $\leftarrow X_L$; (partes alta y baja del dividendo)	CA, CQ
2:	M $\leftarrow D$ (divisor) ‖ COUNT $\leftarrow (n-1)$;	CM, LOAD
3:	S∘A∘Q $\ll 1$;	DA, DQ, SS, CS
4:	S∘A $\leftarrow$ S∘A $-$ 0∘M ;	$\overline{\text{SS}}$, CS, F1F2=11(resta), HALU, CA, HM
5:	S∘A∘Q $\ll 1$ ‖ CONT$--$ ‖ Q0 $\leftarrow \overline{\text{S}}$;	DA, DQ, DEC
6:	S∘A $\leftarrow$ S∘A $-$ 0∘M **if** Q0 | S∘A $\leftarrow$ S∘A $+$ 0∘M **if** $\overline{\text{Q0}}$ ‖ $\rightarrow 5$ **if** $\overline{\text{C0}}$;	$\overline{\text{SS}}$, CS, F1F2=10 ó 11(suma ó resta), Q0, HALU, CA, HM
7:	S∘A $\leftarrow$ S∘A $-$ 0∘M **if** $\overline{\text{Q0}}$ ‖ Q $\ll 1$ ‖ Q0 $\leftarrow \overline{\text{S}}$ ‖ $\rightarrow final$;	$\overline{\text{SS}}$, CS, F1F2=11(resta), HALU, CA, HM, Q0, DQ

1.6.

¿Cual de las dos representaciones, signo-magnitud y complemento a dos, sería la más adecuada en las siguientes situaciones?

a) Cuando es crítico que el *hardware* niegue un número de la forma más sencilla posible.

b) Cuando la mayoría de las operaciones matemáticas realizadas son sumas y restas.

c) Cuando la mayoría de las operaciones matemáticas realizadas son multiplicaciones y divisiones.

d) Cuando es esencial que se detecte si un número es positivo o negativo de la forma más sencilla posible.

Solución

a) En este caso, la representación en signo y magnitud sería la mejor, ya que para negar un número simplemente hay que invertir el bit de signo.

b) Los números en complemento a dos requieren un *hardware* más sencillo para la suma y la resta que los números en signo y magnitud. Con los números en complemento a dos no se necesita *hardware* adicional para sumar números positivos y negativos sino que se tratan a los números como cantidades sin signo y sumándolas se obtiene el resultado correcto en complemento a dos. La resta se puede implementar negando el segundo operando y luego sumando.

Por el contrario, la representación en signo-magnitud requiere un *hardware* diferenciado para realizar restas o sumas de números positivos y negativos, haciendo a esta representación más cara (costosa en tiempo) si la mayoría de los cálculos que se van a realizar son sumas y/o restas.

c) La representación en signo y magnitud es mejor en este caso, ya que la multiplicación y la división se pueden implementar tratando las magnitudes de los números como enteros sin signo y determinando el signo del resultado examinando los bits de signo de los dos valores.

d) En este caso, las dos representaciones son muy parecidas. En general, el signo de un número en cualquiera de las representaciones se puede determinar analizando el bit más significativo del número (si este bit es 1, el número es negativo). La excepción a esto es cuando el valor del número es 0. La representación en signo-magnitud tiene dos representaciones para el 0, una cuando el bit de signo es 1 y otra cuando es 0, mientras que en complemento a dos sólo hay una representación del cero.

En resumen, las dos representaciones son equivalentes si no es importante detectar si el cero es positivo o negativo. Si importa determinar si el nómero es positivo, negativo o cero, los números en complemento a dos son ligeramente mejores.
