

Implementación de un multiplicador de números naturales con la unidad de proceso general (UPG) y una unidad de control de propósito específico.

Antes de preparar esta práctica se deben haber alcanzado los objetivos específicos que se indican en la siguiente tabla. Para ello, es recomendable haber estudiado las secciones de la documentación que se indican en la primera columna de la tabla.

Estos objetivos serán evaluados en parte del informe previo que debéis entregar al inicio de la sesión de laboratorio y en la prueba previa individual que se hará al inicio de la sesión.

En esta práctica implementamos un multiplicador secuencial de dos números naturales codificados en binario con 16 bits. Es equivalente al diseño de propósito específico que hicimos en la práctica 3 pero ahora usando la unidad de proceso general, UPG, que hemos estudiado en clase. Como la UPG ya está diseñada, nos centramos en el diseño de la unidad de control (UC). La comunicación de este multiplicador con el subsistema que le proporciona los datos y recoge el resultado es síncrona, como en la práctica 3. Por ello, os recomendamos que para hacer esta práctica repaséis profundamente la práctica 3 de este curso, así como la documentación sobre la unidad de proceso general UPG (Tema 8).

A modo de repaso de la UPG, en las siguientes figuras se muestra su estructura a nivel de bloques, la tabla que indica las operaciones que realiza la ALU en función del valor de los bits de selección OP y F de 2 y 3 bits respectivamente y por último los 33 bits que forman la palabra de control.



F			OP			
			11	10	01	00
0	0	0	---	X	CMPLT(X,Y)	AND(X,Y)
0	0	1	---	Y	CMPLE(X,Y)	OR(X,Y)
0	1	0	---	---	---	XOR(X,Y)
0	1	1	---	---	CMPEQ(X,Y)	NOT(X)
1	0	0	---	---	CMPLTU(X,Y)	ADD(X,Y)
1	0	1	---	---	CMPLEU(X,Y)	SUB(X,Y)
1	1	0	---	---	---	SHA(X,Y)
1	1	1	---	---	---	SHL(X,Y)

Funcionalidades de la ALU de la UPG

@A			@B			Rb/N	OP		F			In/Alu	@D			WrD	N (Hexa)			
b ₂	b ₁	b ₀	b ₂	b ₁	b ₀		b ₁	b ₀	b ₂	b ₁	b ₀		b ₂	b ₁	b ₀		D ₃	D ₂	D ₁	D ₀
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	X	X	X	X

Palabra de control de la UPG

Antes de pasar al diseño del multiplicador usando la UPG, vamos a hacer algunos ejercicios para familiarizarnos con ella y con el diseño de unidades de control específicas para controlar a la UPG.

➤ Informe previo

Pregunta 1

Cada fila de la siguiente tabla indica con mnemotécnicos una posible acción de la UPG durante un ciclo. Cuando varias acciones individuales se realizan en el mismo ciclo (están en la misma fila), se separan estas acciones por el símbolo // que indica que se realizan en paralelo. Se sigue la notación definida en la documentación (Tema 8):

Cada acción comienza por un mnemotécnico que indica la acción a realizar, por ejemplo ADD, SUB... A continuación se especifica el registro destino de la operación y los registros fuente, separados por comas (o del único registro fuente, en el caso de la operación NOT). En algunas acciones no se desea escribir el resultado de la operación en ningún registro destino, pues sólo se desea que la unidad de control vea el valor del bit z que le entrega la ALU para decidir en función de este bit cuál es el estado siguiente. En estos casos, en el lugar del registro destino pondremos un guión. El segundo operando de una operación de la ALU puede no ser el contenido de un registro sino un valor inmediato codificado por la unidad de control en el campo N. Este caso se especifica poniendo en el campo del segundo operando, en vez de un registro, el valor del campo N codificado en hexadecimal (lo que se indica comenzando por 0x el vector de dígitos) o directamente en decimal. En caso de que el segundo operando sea un valor inmediato, se añade la letra I (de *Immediate*) al mnemotécnico que indica la operación que realiza la ALU, por ejemplo ADDI R4, R4, -1.

Indicad la tira de bits que forma la palabra de control para cada una de las acciones (filas de la tabla). Indicad con una x los bits de la palabra de control que pueden valer tanto 1 como 0. Si la acción o acciones individuales que se especifican en una fila no pueden realizarse en la UPG en un único ciclo, indicadlo con una línea que tache los bits de la palabra de control y después explica brevemente por qué no puede hacerse esa acción o acciones individuales. Para ayudar a clarificar el ejercicio hemos rellenado las dos primeras filas de la tabla.

Notas a la solución: Las acciones ADD R1, R2, R3 y NOT R2, R1 no pueden realizarse en el mismo ciclo ya que la ALU sólo puede realizar una de las dos operaciones aritmético/lógicas en el mismo ciclo y además tampoco pueden escribirse al final del ciclo dos registros del banco de registros.

			@A			@B			Rb/N	OP		F			In/Alu	@D			WrD	N (Hexa)			
			b ₂	b ₁	b ₀	b ₂	b ₁	b ₀		b ₁	b ₀	b ₂	b ₁	b ₀		b ₂	b ₁	b ₀		D ₃	D ₂	D ₁	D ₀
AND	R3, R1, R5		0	0	1	1	0	1	1	0	0	0	0	0	0	0	1	1	1	X	X	X	X
ADD	R1, R2, R3 // NOT R2, R1																						
SHAI	R7, R7, -3																						
ADDI	R4, R7, -1																						
OUT	R5 // IN R6																						
IN	R1 // ADD R2, R3, R7																						
MOVEI	R3, 327																						
SHLI	R6, R6, 1																						
CMPEQ	-, R3, R2																						
SUBI	-, R2, 1																						

➤ **Informe previo**

Pregunta 2

Indicad el valor en hexadecimal de cada una de las palabras de control de la pregunta anterior, rellenando la siguiente tabla. Codificad como 0 los bits x de la palabra de control. Considerad los bits de la palabra de control ordenados como en la tabla anterior. Considerad, para la codificación en hexadecimal, que el bit b₀ de N es el bit de menor peso. Como la palabra de control tiene 33 bits hay que usar 9 dígitos hexadecimales, aunque del dígito de mayor peso sólo nos interesa el bit de menor peso (los 3 bits de más peso de este dígito los codificaremos como 0, ya que no se refieren a ninguna señal binaria de la palabra de control)

Mnemotécnico	Palabra de control hexadecimal
AND R3, R1, R5	06C070000
ADD R1, R2, R3 // NOT R2, R1	-----
SHAI R7, R7, -3	
ADDI R4, R7, -1	
OUT R5 // IN R6	
IN R1 // ADD R2, R3, R7	
MOVEI R3, 327	
SHLI R6, R6, 1	
CMPEQ -, R3, R2	
SUBI -, R2, 1	

➤ **Informe previo**

Pregunta 3

¿Cómo se modifican los registros de la UPG al final del ciclo en el cual la UPG ha estado controlada por la palabra de control que se especifica para cada uno de los siguientes apartados? Cada fila de la tabla es un ejercicio independiente. Debe suponerse, para cada fila/ejercicio, que desde el inicio del ciclo que estamos considerando, el estado de la UPG (el contenido de los registros de la UPG) y el valor presente en el bus de datos de entrada de la UPG es:

$$R_i \text{ vale } i+3 \text{ para } 0 \leq i \leq 7$$

$$RD-IN = 23.$$

Sólo tenéis que especificar los cambios ocurridos en el estado del computador tras la llegada del flanco de reloj que indica el final del ciclo. Los registros que no se modifiquen no deben especificarse. (Los dos primeros apartados los hemos completado nosotros, como muestra).

- a) AND R3, R1, R5
Respuesta: R3 = 0
- b) ADD R1, R2, R3 // NOT R2, R1
Respuesta: -----

- c) SHAI R7, R7, -3
- d) ADDI R4, R7, -1
- e) OUT R5 // IN R6
- f) MOVEI R3, 327
- g) IN R1 // ADD R2, R3, R7
- h) SHLI R6, R6, 1
- i) CMPEQ -, R3, R2
- j) SUBI -, R2, 1

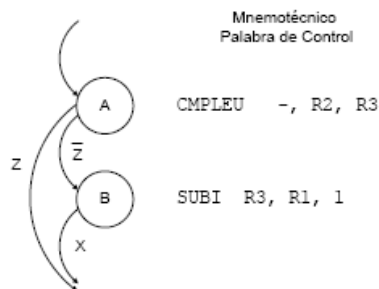
➤ Informe previo

Pregunta 4

Para cada uno de los apartados siguientes, dibujad el fragmento del grafo de estados de la unidad de control de propósito específico para que la UPG realice la funcionalidad que se indica. Indicad la palabra de control en hexadecimal que genera la unidad de control para cada nodo. Indicad con una flecha la llegada al primer nodo del fragmento y con otra flecha la salida del último nodo del fragmento. Consideramos que los contenidos de los registros son números naturales codificados en binario (*unsigned integers*). El apartado a) lo damos resuelto, escribiendo al lado de cada nodo la palabra de control con mnemotécnicos que debe generar la unidad de control cuando ejecuta el nodo.

- a) **if** (R2 ≤ R3)
R3 = R1 - 1;

Respuesta:



- b) **if** (R1 != 1)
R2 = R2 + R2;
else
R2 = R2 + 5;

- c) **for** (R2 = 3; R2 ≤ R5; R2 = R2+1)
R7 = R7 + 3;

- d) **if** (R1<3> = 1)
R2 = R2 + R5;

(Nota: R1<3> se refiere al bit 3 del registro R1. La acción ANDI de R3 con un valor inmediato adecuado da como resultado 0 si el bit 3 de R1 vale 0 y distinto de 0 si vale 1).

1.3 Multiplicación en la UPG

1.3.1 Repaso de la práctica 3

En la práctica 3 obtuvimos el siguiente algoritmo para multiplicar los números naturales X_u e Y_u :

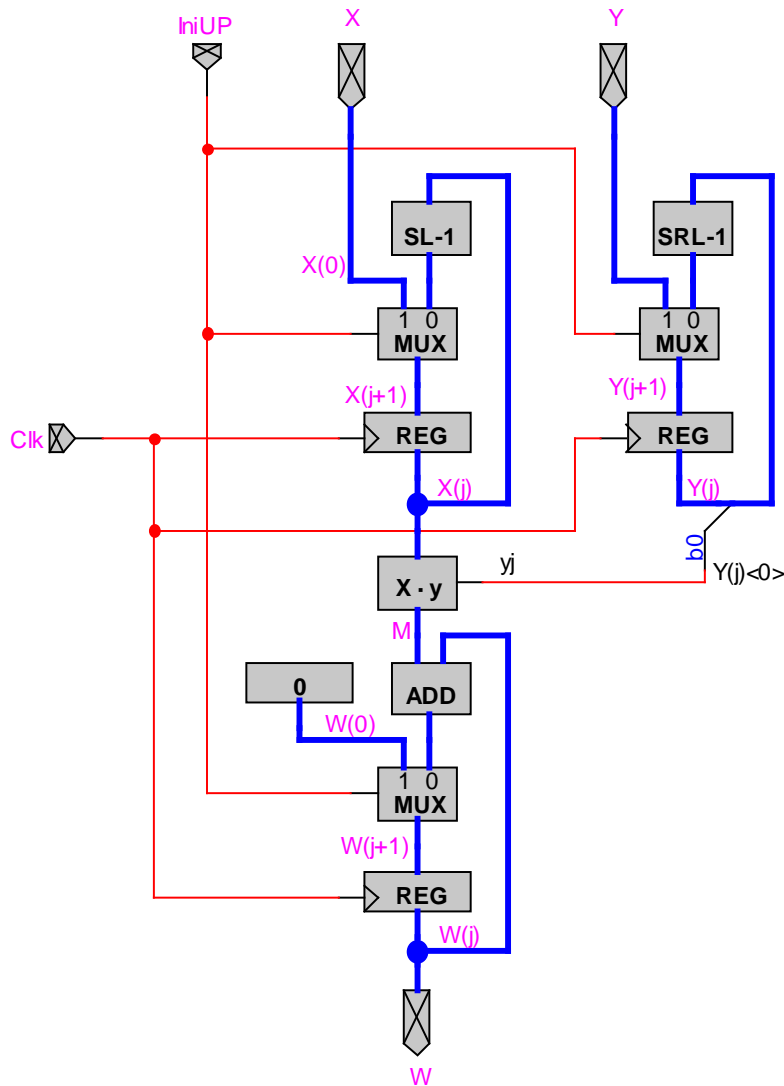
Algoritmo MUL

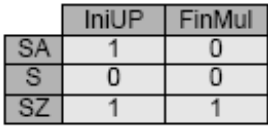
```

$$X_u(0) = X_u ;$$

$$W_u(0) = 0 ;$$
for ( $j = 0$ ;  $j < n$ ;  $j = j + 1$ ) {  
   $M_u = X_u(j) \times y_j$  ;  
   $W_u(j+1) = W_u(j) + M_u$  ;  
   $X_u(j+1) = X_u(j) \times 2$  ;  
}  
 $W_u = W_u(n)$  ;
```

También en la práctica 3 implementamos este algoritmo en LogicWorks usando una unidad de proceso y una unidad de control, ambas específicas para este algoritmo. A continuación mostramos la unidad de proceso y el grafo de la unidad de control que creamos en la práctica 3.





Recordad que y_j es el bit de peso 2^j del vector de bits Y que codifica al multiplicador Y_u . Otra forma de escribir el algoritmo de multiplicación anterior con más detalle, y de acuerdo con la implementación realizada en la práctica 3, consiste en especificar cómo se obtiene y_j . Primero, antes de entrar en el bucle del algoritmo, se carga Y en el registro superior derecha. Para que, en cada una de las iteraciones j , el bit 0 (bit de peso 2^0) de $Y(j)$ sea y_j , en cada iteración del bucle hay que desplazar a la derecha el contenido del registro superior derecho (división entera por 2). El algoritmo que se obtiene especificando la obtención de y_j es el siguiente. Notad que la expresión $Y_u(j) \bmod 2$, que aparece en la primera línea del cuerpo del bucle, es una forma de referirnos al bit 0 de $Y(j)$ usando el valor del número, $Y_u(j)$, representado por $Y(j)$.

$$\begin{aligned} X_u(0) &= X_u ; \\ Y_u(0) &= Y_u ; \\ W_u &= 0 ; \\ \text{for } (j = 0; j < n; j = j + 1) \{ \\ &M_u = X_u(j) \times (Y_u(j) \bmod 2) ; \\ &W_u(j+1) = W_u(j) + M_u ; \\ &X_u(j+1) = X_u(j) \times 2 ; \\ &Y_u(j+1) = Y_u(j) / 2 ; \\ &\} \\ W_u &= W_u(n) ; \end{aligned}$$

1.3.2 Enunciado del problema

Suponiendo que el vector de bits X que representa al multiplicando X_u se encuentra inicialmente en el registro R6 de la UPG y que el vector de bits Y que representa al multiplicador Y_u se encuentra en R7, dibujad el grafo de la unidad de control, especificando la palabra de control con mnemotécnicos, que deje los 16 bits de menor peso del resultado de la multiplicación de X_u por Y_u en el registro R5. No hay que detectar el caso de desbordamiento, esto es, el caso en el que el resultado de la multiplicación no puede representarse correctamente en binario con 16 bits.

1.3.3 Obtención del grafo de estados, sentencia por sentencia

Para resolver este problema vamos a utilizar el algoritmo MUL (segunda versión). De momento, y motivados por el enunciado del problema, asociamos los siguientes registros del banco de registros de la UPG con las variables del algoritmo MUL:

- R6 contiene el vector de bits $X(j)$ (que representa al valor $X_u(j)$)
- R7 contiene el vector de bits $Y(j)$ (que representa al valor $Y_u(j)$)
- R5 contiene el vector de bits $W(j)$ (que representa al valor $W_u(j)$)

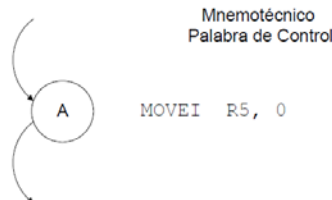
a) Sentencias anteriores al bucle

Dada esta asociación las dos primeras líneas del algoritmo MUL ($X_u(0) = X_u$ e $Y_u(0) = Y_u$) no necesitan ningún nodo del grafo para su implementación. La tercera línea del algoritmo ($W_u(0) = 0$) se puede implementar con la acción:

`MOVEI R5, 0`

(También puede hacerse, por ejemplo con `XOR R5, R5, R5`; o con `SUB R5, R5, R5`)

El nodo del grafo que especifica esta acción es el siguiente:



b) Implementación del bucle

En el diseño específico de la práctica 3 realizamos las 4 sentencias de cada iteración del bucle del algoritmo MUL en un solo ciclo, ya que la unidad de proceso específica fue diseñada para ello y se dispuso del hardware necesario para que pudiera actuar en paralelo (en el mismo ciclo). Ahora, en la UPG, tenemos una única ALU que sólo puede efectuar una operación por ciclo. Por ello, lo que en la práctica 3 hacíamos en un ciclo (un nodo de la unidad de control) ahora lo tendremos que hacer en varios ciclos (varios nodos de la unidad de control).

También, en la práctica 3 implementamos las 16 iteraciones del algoritmo MUL mediante una secuencia de 16 nodos del grafo de estados, uno por iteración del bucle `FOR` del algoritmo. De momento pensemos en hacer lo mismo, con la salvedad de que será una secuencia de 16 subgrafos, cada uno de ellos de varios nodos, ya que cada iteración del bucle necesita varios ciclos para ejecutarse.

Sentencias del cuerpo del bucle

Veamos ahora cómo implementamos cada una de las sentencias del cuerpo del bucle del algoritmo MUL.

Primera sentencia del cuerpo del bucle. La primera sentencia tiene cierta dificultad:

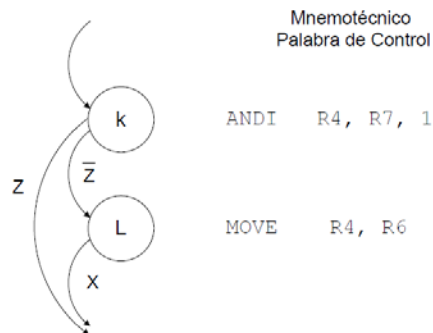
$$M_u = X_u(j) \times (Y_u(j) \bmod 2)$$

Supongamos que deseamos el valor de M_u en el registro R4, por ejemplo. En R4 debe quedar el contenido de R6, $X_u(j)$, si el bit de menor peso de R7, de la representación de $Y_u(j)$, es 1 y debe quedar el valor 0 si el bit de menor peso de R7 es 0.

Podemos escribir en R4 el bit de menor peso de R7, $(Y_u(j) \bmod 2)$, mediante la acción,

```
ANDI R4, R7, 1
```

El valor 1 (representado en binario como todo ceros excepto el bit de menor peso que vale 1) hace el papel de una máscara que sólo nos deja ver el bit 0 de R7. Así, después de ejecutar la acción `ANDI`, el registro R4 valdrá 1 o 0 según el bit de menor peso de R7 sea 1 o 0 respectivamente. Dicho esto, podemos comprobar que el siguiente fragmento de grafo implementa eficientemente la primera sentencia del cuerpo del bucle (carga en R4 un 0 si el bit 0 de R7 vale 0 y si no carga el valor de R6):



El resto de sentencias del cuerpo del bucle son muy fáciles de implementar en la UPG pues sólo requieren un nodo (un ciclo de reloj) para su ejecución.

Segunda sentencia del cuerpo del bucle. Con el uso de los registros que hemos establecido, la sentencia,

$$W_u(j+1) = W_u(j) + M_u$$

se implementa con la acción

```
ADD R5, R5, R4
```

Tercera sentencia del cuerpo del bucle. La sentencia

$$X_u(j+1) = X_u(j) \times 2$$

se implementa con

```
SHLI R6, R6, 1
```

(también puede hacerse con `ADD R6, R6, R6` o con `SHAI R6, R6, 1`).

Cuarta sentencia del cuerpo del bucle. La sentencia

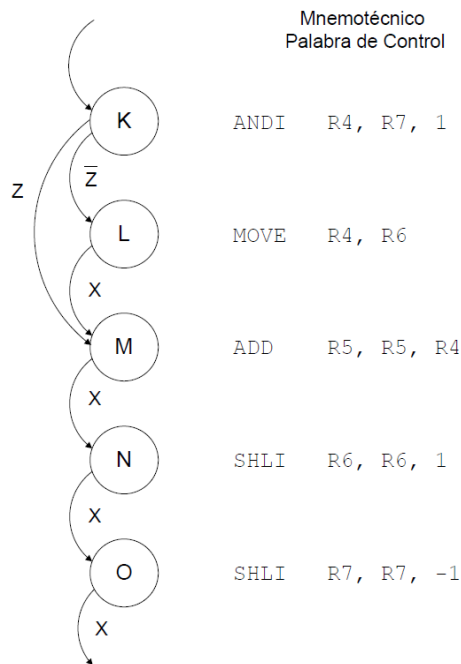
$$Y_u(j+1) = Y_u(j) / 2$$

se implementa con

```
SHLI R7, R7, -1
```

Juntando y optimizando las sentencias del cuerpo del bucle

Juntando los 5 nodos que acabamos de definir obtenemos los nodos del cuerpo del bucle, tal como muestra la siguiente figura.



➤ **Informe previo**
Pregunta 5

¿Se os ocurre alguna forma alternativa de escribir el algoritmo para ahorrarnos una instrucción en el cuerpo del bucle?

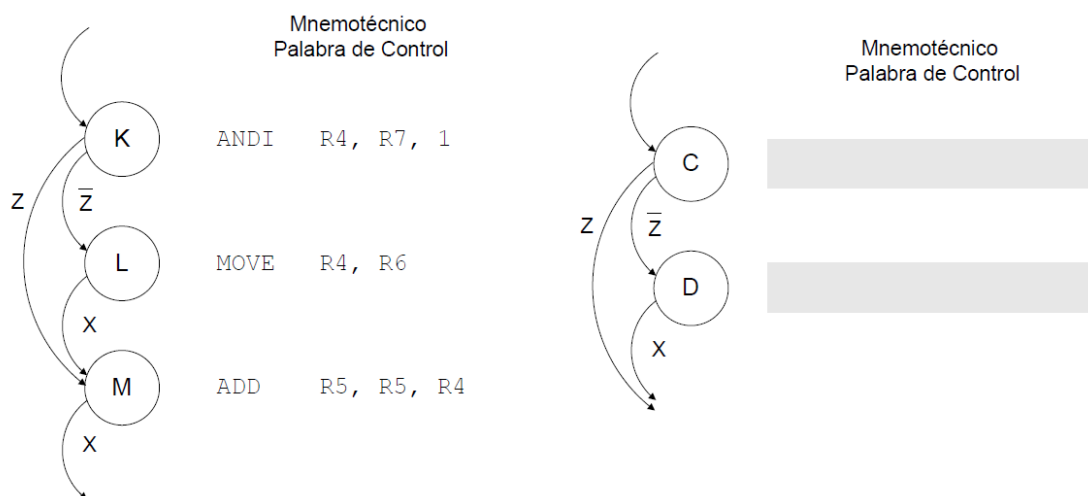
Observad la implementación conjunta que ha resultado de las sentencias de alto nivel

$$M_u = X_u(j) \times (Y_u(j) \bmod 2)$$

$$W_u(j+1) = W_u(j) + M_u.$$

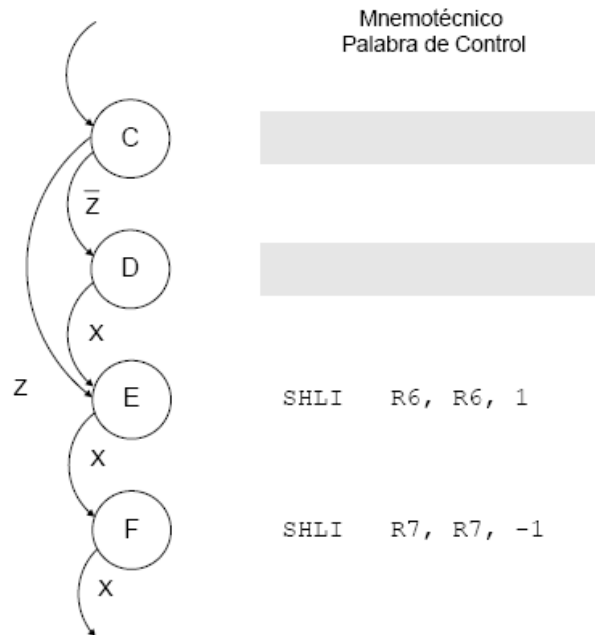
Como hemos usado R4 temporalmente tanto para $Y_u(j) \bmod 2$ como para M_u y como sumar 0 a $W_u(j)$ es lo mismo que no sumarle nada, podemos implementar estas dos sentencias con 2 nodos del grafo, en vez de con los 3 que hemos usado y sin requerir el registro temporal R4.

Completad la tabla de las palabras de control del nuevo grafo optimizado. A la izquierda os damos el original, para que os sirva de ayuda.



Con esta optimización que acabáis de hacer, completad la implementación definitiva del cuerpo del bucle con 4 nodos que se encuentra en la siguiente figura (simplemente volved a copiar los

mnemotécnicos de los nodos C y D ya que los antiguos nodos N y O de la implementación con 5 nodos son iguales a los que ahora se llaman E y F en la implementación con 4 nodos).



Veamos ahora la implementación del control del bucle.

Control del bucle

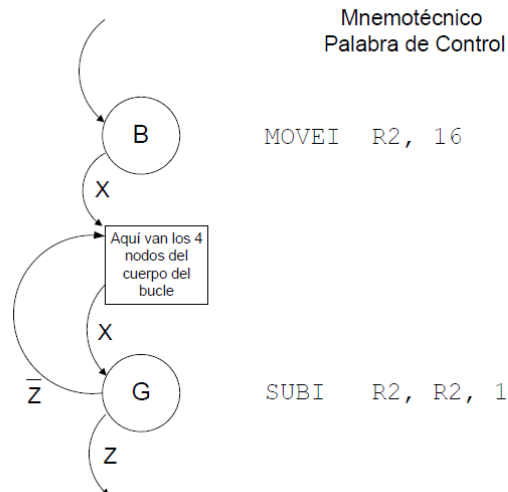
En la práctica 3 implementamos una versión del algoritmo MUL con el bucle totalmente desenroscado usando un total de 16+2 nodos, uno para cada iteración del bucle más el estado inicial y final. Ahora, con la UPG, cada iteración del bucle requiere 4 nodos por lo que una implementación desenroscada requeriría al menos 64 nodos (4x16), y esto ya empieza a ser excesivo. Por ello, en esta práctica vamos a hacerlo de otra forma. En la unidad de control tendremos solamente una instancia de los 4 nodos que implementan el cuerpo del bucle de forma que la ejecución del algoritmo suponga reutilizar estos pocos nodos, pasar por ellos, 16 veces, una por iteración del bucle. Se requerirá un registro en la UPG para contar el número de iteraciones hechas. Cada vez que se ejecute una iteración la unidad de control ordenará incrementar este contador. Vamos a ver esto en detalle a continuación.

Es muy importante observar que lo que en la unidad de proceso específica de la práctica 3 se hacía en un ciclo (un nodo), ahora requiere 3 o 4 ciclos, dependiendo del valor del bit del multiplicador que se esté tratando. Al implementar la unidad de control con un bucle, reutilizando los 4 nodos del cuerpo del bucle 16 veces, se usan poco más de 4 nodos en total (muchos menos de 64) pero se necesita más tiempo de ejecución, pues a los 3 o 4 ciclos por iteración del cuerpo del bucle habrá que añadirles algún nodo extra (ciclo extra) para ordenar a la UPG que actualice el registro contador y que compruebe si se han realizado las 16 iteraciones para saber cuándo terminar.

Vamos a implementar el bucle

```
for (j = 0; j < n; j = j + 1) {
    acciones del cuerpo del bucle
}
```

Una forma eficiente de hacerlo, ya que sabemos que hay que ejecutar al menos 1 vez las acciones del cuerpo del bucle (de hecho se tienen que ejecutar 16 veces), consiste en usar la estructura de control `do while` que ejecuta el cuerpo del bucle y después pregunta si hay que volver al ejecutar otra iteración. Consiste en inicializar un registro contador con el valor 16 y decrementarlo una unidad en cada iteración. En el ciclo en el que se decrementa el registro, la unidad de control puede saber si vale cero mirando el valor del bit z que llega de la ALU. Si $z = 1$ ya se ha terminado y si vale 0 hay que volver a ejecutar otra vez el bucle. El grafo resultante es el siguiente:



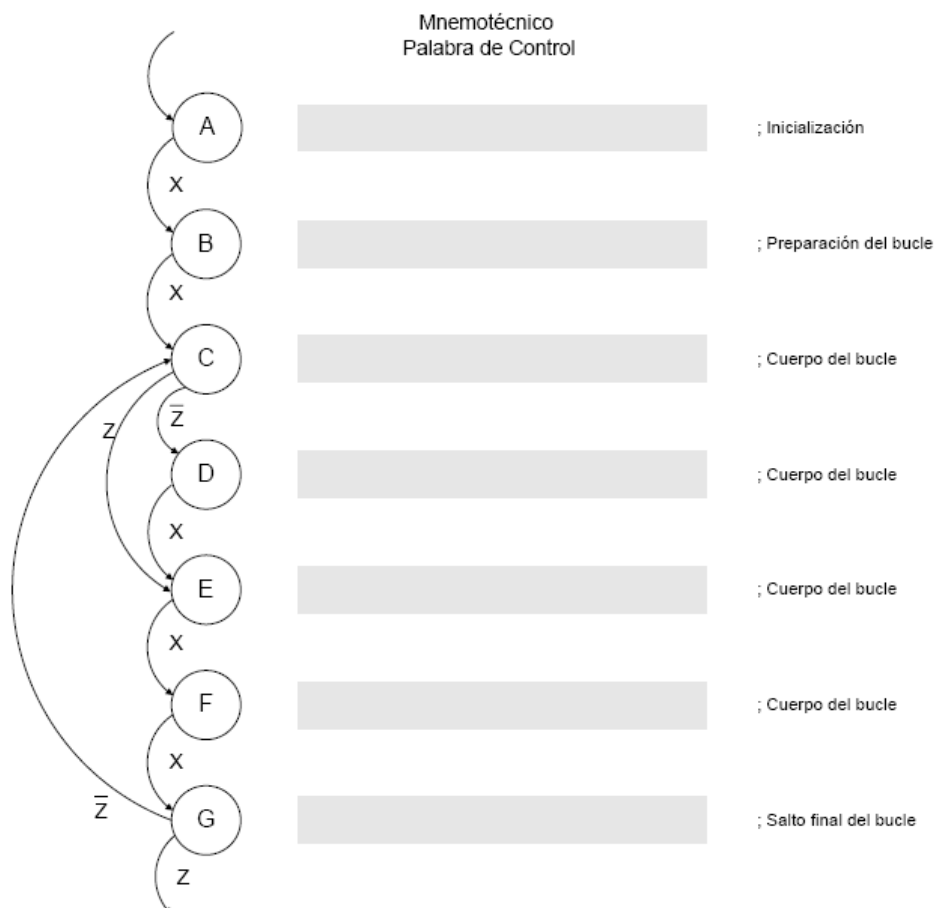
c) Grafo completo de la Unidad de Control del algoritmo MUL

Antes de terminar el grafo completo de la unidad de control decir que la última sentencia del algoritmo MUL, que se encuentra fuera del bucle, no necesita ningún nodo para su implementación, simplemente debemos saber que el resultado de la multiplicación se encuentra en el registro R5 de la UPG.

➤ Informe previo

Pregunta 6

Completad las palabras de control de cada nodo usando mnemotécnicos, para la unidad de control que implementa el algoritmo MUL, tal como la hemos diseñado en este apartado.



1.3.4 Seguimiento del algoritmo

➤ Informe previo

Pregunta 7

Seguid la ejecución del algoritmo ciclo a ciclo rellenando la siguiente tabla. Suponed que los registros son de 4 bits (para que el algoritmo sea correcto con registros de 4 bits la acción del nodo B debe ser `MOVEI R2, 4` en vez de `MOVEI R2, 16`). En la tabla se da el valor inicial de los registros en el ciclo 0. Se desea multiplicar los números 3 por 5, por lo que R6 contiene inicialmente el valor 3 y R7 el 5. El resto de registros involucrados en el algoritmo no importa el valor inicial que tengan. Hemos rellenado nosotros las 3 primeras filas, mostrando el resultado de los registros después del ciclo 1, para mostrar como deseamos que se complete la tabla. Por claridad los registros que no se modifican no se escriben en la tabla. Así, el valor final de cada registro será el de la última fila en que se visualiza un valor. Sólo es necesario rellenar los 20 primeros ciclos.

Ciclo	Mnemotécnico	Estado actual de los registros			
		R2	R5	R6	R7
0	<code>MOVEI R5, 0</code>	X X X X	X X X X	0 0 1 1	0 1 0 1
1	<code>MOVEI R2, 4</code>		0 0 0 0		
2	<code>ANDI -, R7, 1</code>	0 1 0 0			
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					
17					
18					
19					

Además de rellenar la tabla, responded a las siguientes preguntas:

- ¿Cuántos ciclos tarda en ejecutarse el algoritmo?
- ¿Cuál es el estado de la UPG (el valor de los registros de la UPG) después de ejecutarse el algoritmo?

1.4 Un algoritmo de multiplicación con terminación temprana

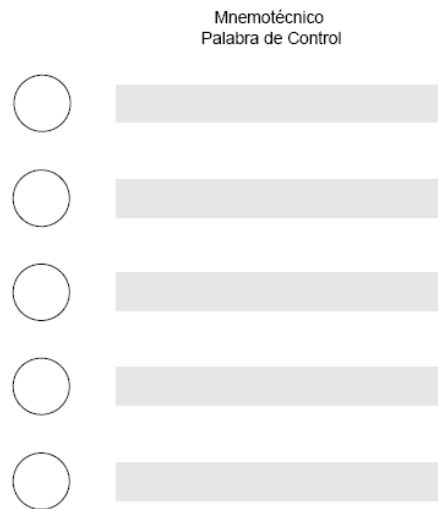
➤ Informe previo

Pregunta 8

Dibujad el grafo de estados del algoritmo de multiplicación de la pregunta 6 del informe previo modificado adecuadamente para que no tengan que ejecutarse siempre 16 iteraciones del bucle. Cuando la representación en binario del multiplicador, Y, tenga k ceros en las posiciones de mayor peso, debemos ahorrarnos las últimas k iteraciones del algoritmo. Esto es posible porque el resultado parcial de la multiplicación que se encuentra en R5 al final de la iteración $n-1-k$ ya es el definitivo. Si continuáramos con el algoritmo sumaríamos un 0 a R5 en cada iteración.

Dibujad el grafo de la unidad de control para implementar este algoritmo de multiplicación que denominamos “**con terminación temprana**”. Indicad claramente las etiquetas de los nodos, la palabra de control y las etiquetas en los arcos. **Una pista:** Os podéis hacer dos preguntas que os ayudarán a encontrar el nuevo grafo, ¿Cómo se modifican los arcos del grafo para poder detectar

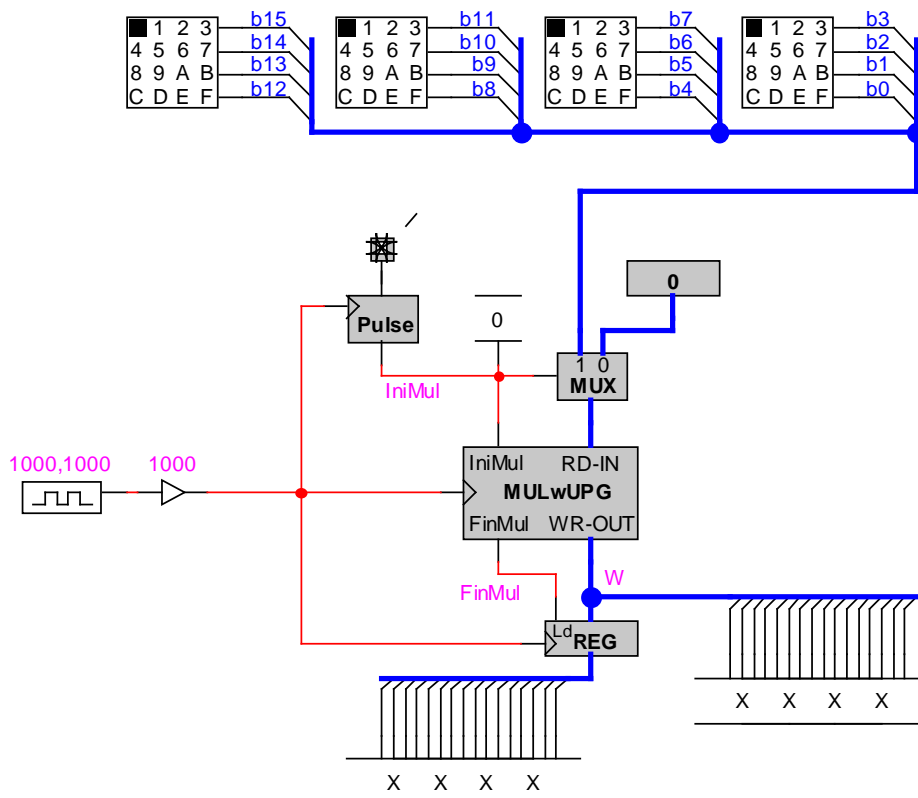
la terminación temprana? Una vez hecha esta modificación preguntaos ¿Qué nodos del grafo son innecesarios ahora? Después de responder a estas preguntas veréis que el grafo resultante tiene solamente 5 nodos en vez de los 7 nodos que tiene el de la pregunta 6. Completad el grafo siguiente, añadiendo los arcos con sus etiquetas e indicando la palabra de control de cada nodo usando mnemotécnicos.




1.5 Añadiendo Entradas/Salidas síncronas

El grafo que implementa el algoritmo de multiplicación con terminación temprana usando la UPG puede incrustarse en un grafo más complejo. Cada vez que se desea multiplicar dos números, sólo hay que cargar los números en R6 y R7 antes de pasar a ejecutar el grafo que hemos diseñado y después recoger el resultado de R5.

En esta práctica vamos a usar el multiplicador en un entorno equivalente al de la práctica 3. Entramos los operandos de forma síncrona desde un teclado hexadecimal de 4 dígitos y vemos el resultado en un display hexadecimal de 4 dígitos. El circuito se muestra a continuación.



El funcionamiento del sistema es como sigue. Como el multiplicador sólo tiene una entrada de datos (RD-IN), a diferencia de lo que ocurría en la práctica 3, ahora tenemos que entrar primero un operando, por ejemplo el multiplicando, y después el multiplicador.

Cuando hacemos clic en el pulsador  el bloque Pulse genera un pulso sincronizado con el reloj del multiplicador. El multiplicador sabe que en el primer ciclo que IniMul vale 1, el primer operando está disponible en el bus RD-IN. También sabe que después de este pulso, llegará otro para indicar que el segundo operando está disponible en el bus. Lo que no está definido es cuántos ciclos pasarán desde el primer pulso al segundo, ya que esto depende de lo rápido o lento que sea el usuario en pulsar el teclado, pero seguro que pasarán varios ciclos. Sólo se puede asegurar que un operando está un ciclo en la entrada, el ciclo en el que IniMul vale 1.

Cuando el multiplicador tenga el resultado correcto en el bus de salida, lo indicará poniendo a 1 durante un ciclo la señal de control FinMul. El resultado sólo estará presente en el bus de salida del multiplicador durante ese ciclo. Por ello, para retener el resultado hemos dispuesto un registro con su señal de carga alimentada por FinMul.

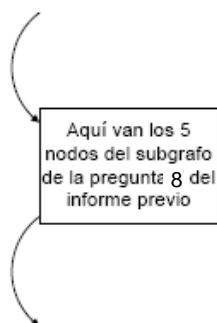
Si mientras se está multiplicando un par de números (después del segundo ciclo en el que IniMul vale 1 y antes de que FinMul valga 1) se recibe por la entrada IniMul un 1, el multiplicador hará caso omiso de esta petición y continuará sin inmutarse hasta que ponga a 1 FinMul.

En el último ciclo, en el que FinMul vale 1, si IniMul vale 1 comenzará otra multiplicación. Por ello, en este último ciclo la unidad de control debe generar la palabra de control adecuada para que al final de ese ciclo se cargue en el registro adecuado de la UPG el valor que se encuentra en RD-IN. Si IniMul vale 1, este valor es el primer operando de una nueva multiplicación. Mientras que si IniMul vale 0, lo que se cargue en el registro al final del ciclo no importa. En este caso, al ciclo siguiente se debe pasar al estado inicial, donde se volverá a cargar el registro con RD-IN. Tanto si se está en el estado inicial, como en el último estado, si IniMul vale 1 se deberá pasar al estado en el que se carga en otro registro de la UPG el segundo operando. En este segundo estado debe permanecer hasta que IniMul valga 1 por segunda vez, validando el segundo operando. Después de este estado ya puede comenzar el cálculo de la multiplicación.

➤ Informe previo

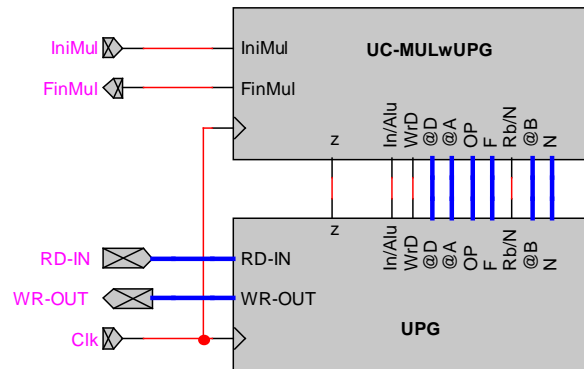
Pregunta 9

Dibujad el grafo de estados de la unidad de control del multiplicador con terminación temprana para que funcione como acabamos de indicar. Para ello añadid a los 5 nodos del grafo de la pregunta 8 del informe previo (representado en el nuevo grafo mediante un cuadrado) los 2 nuevos nodos necesarios para realizar la entrada de los dos operandos y el nuevo nodo necesario para la salida del resultado. Indicad claramente las nuevas transiciones (arcos con sus etiquetas) y las salidas de cada nuevo nodo, completando la siguiente figura.

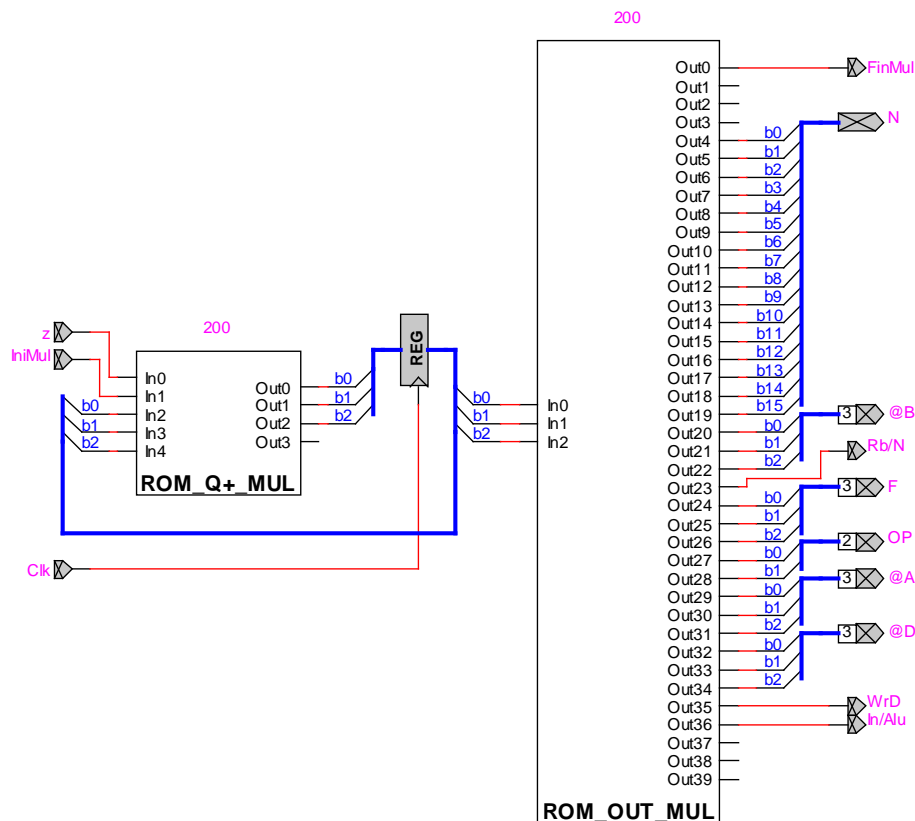


1.6 Implementación del multiplicador en LogicWorks

En esta sección vamos a implementar en LogicWorks el multiplicador con terminación temprana usando la UPG. La siguiente figura muestra el esquema interno del multiplicador en dos bloques, la unidad de control específica y la unidad de proceso general UPG.



En la siguiente figura mostramos el esquema interno de la unidad de control específica para la multiplicación. La hemos diseñado con dos memorias ROM, la ROM_Q+_MUL para calcular el estado siguiente, en función de las entradas a la unidad de control y del estado actual (implementa la tabla de transiciones que se obtiene del grafo de estados), y la ROM_OUT_MUL para calcular las salidas de la unidad de control en función del estado actual (tabla de salidas que se obtiene del grafo de estados). Observad que de los 4 bits (un dígito hexadecimal) de menor peso de la salida de la ROM, solamente usamos el bit de menor peso, para codificar la señal FinMul. Hemos usado un dígito en vez de un bit para codificar FinMul, para que sea más fácil escribir el contenido de la ROM en hexadecimal, aunque por esto se necesite una ROM con 3 bits más por palabra. Además hemos usado 9 dígitos hexadecimales para la palabra de control, de 33 bits, aunque no se usen los 3 bits de más peso de la ROM.



➤ Informe previo

Pregunta 10

Indicad el contenido de cada una de las dos memorias ROM, para que el multiplicador funcione como se ha especificado (y de acuerdo con el grafo de 8 estados que habéis diseñado en la pregunta 9 del informe previo). Los bits de salida de las ROM que no se usan haced que sean siempre 0. Expresad el contenido de cada una de las dos ROM en hexadecimal, separando cada palabra por un espacio o un cambio de línea acordando que la primera palabra corresponde a la palabra con dirección 0, la siguiente a la de dirección 1, etc. Usad el espacio siguiente.

ROM_Q+_MUL

ROM_OUT_MUL

Informe previo Práctica-4

Apellidos y nombre: Grupo:

Pregunta 1

		@A			@B			Rb/N	OP		F			In/Alu	@D			WrD	N (Hexa)			
		b ₂	b ₁	b ₀	b ₂	b ₁	b ₀		b ₁	b ₀	b ₂	b ₁	b ₀		b ₂	b ₁	b ₀		D ₃	D ₂	D ₁	D ₀
AND	R3, R1, R5	0	0	1	1	0	1	1	0	0	0	0	0	0	0	1	1	1	X	X	X	X
ADD	R1, R2, R3 // NOT R2, R1																					
SHAI	R7, R7, -3																					
ADDI	R4, R7, -1																					
OUT	R5 // IN R6																					
IN	R1 // ADD R2, R3, R7																					
MOVEI	R3, 327																					
SHLI	R6, R6, 1																					
CMPEQ	-, R3, R2																					
SUBI	-, R2, 1																					

Pregunta 2

Mnemotécnico	Palabra de control hexadecimal
AND R3, R1, R5	06C070000
ADD R1, R2, R3 // NOT R2, R1	-----
SHAI R7, R7, -3	
ADDI R4, R7, -1	
OUT R5 // IN R6	
IN R1 // ADD R2, R3, R7	
MOVEI R3, 327	
SHLI R6, R6, 1	
CMPEQ -, R3, R2	
SUBI -, R2, 1	

Pregunta 3

- a) AND R3, R1, R5
Respuesta: R3 = 0
- b) ADD R1, R2, R3 // NOT R2, R1
Respuesta: -----

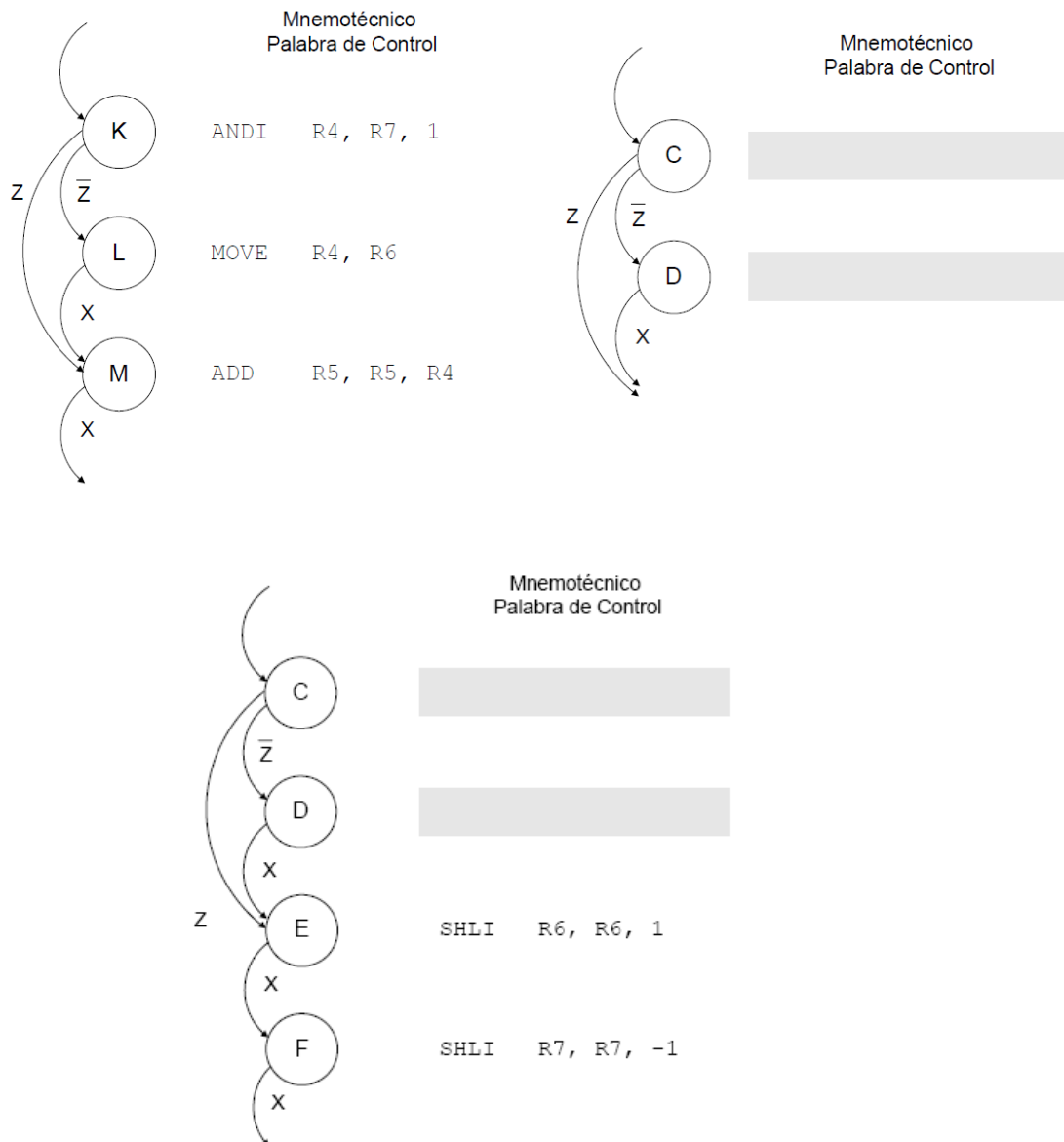
- c) SHAI R7, R7, -3
- d) ADDI R4, R7, -1
- e) OUT R5 // IN R6
- f) MOVEI R3, 327
- g) IN R1 // ADD R2, R3, R7
- h) SHLI R6, R6, 1
- i) CMPEQ -, R3, R2
- j) SUBI -, R2, 1

Pregunta 4

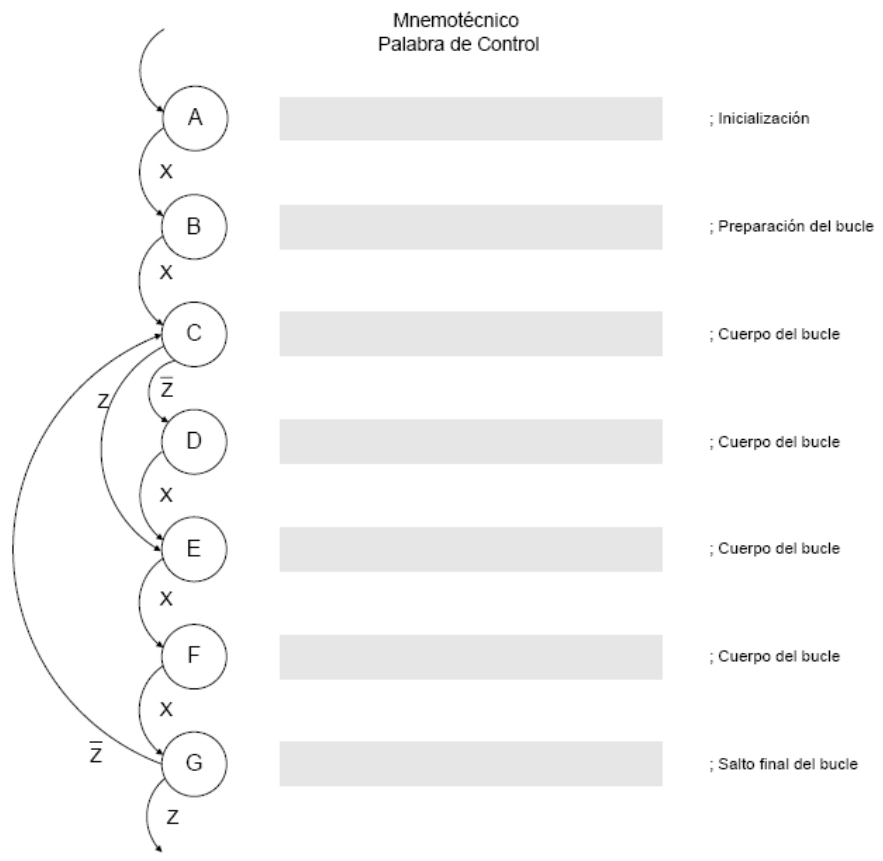
- a) **Ya está resuelto en el enunciado**
- b) **if** (R1 != 1)
 R2 = R2 + R2;
else
 R2 = R2 + 5;
- c) **for** (R2 = 3; R2 <= R5; R2 = R2+1)
 R7 = R7 + 3;

- d) **if** (R1<3> = 1)
 R2 = R2 + R5;
 (Nota: R1<3> se refiere al bit 3 del registro R1. La acción ANDI de R3 con un valor inmediato adecuado da como resultado 0 si el bit 3 de R1 vale 0 y distinto de 0 si vale 1).

Pregunta 5



Pregunta 6



Pregunta 7

Ciclo	Mnemotécnico	Estado actual de los registros			
		R2	R5	R6	R7
0	MOVEI R5, 0	X X X X	X X X X	0 0 1 1	0 1 0 1
1	MOVEI R2, 4		0 0 0 0		
2	ANDI -, R7, 1	0 1 0 0			
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					
17					
18					
19					

a) ¿Cuántos ciclos tarda en ejecutarse el algoritmo?

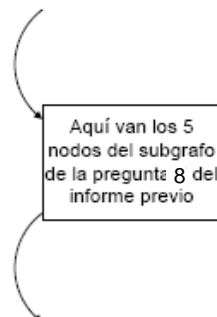
- b) ¿Cuál es el estado de la UPG (el valor de los registros de la UPG) después de ejecutarse el algoritmo?

Pregunta 8

Mnemotécnico
Palabra de Control

<input type="radio"/>	
<input type="radio"/>	
<input type="radio"/>	
<input type="radio"/>	
<input type="radio"/>	

Pregunta 9



Pregunta 10

ROM_Q+_MUL

ROM_OUT_MUL