

PRÁCTICA 5. Trabajo previo

Introducción al lenguaje máquina y ensamblador SISA. Implementación de algoritmos de multiplicación en el computador SISC.

Antes de realizar la práctica

Antes de preparar esta práctica se deben haber estudiado las secciones de la documentación señaladas.

Secciones de la documentación a estudiar antes de preparar la práctica
--

Capítulo 9 completo. Entrada/salida
Capítulo 10 completo. Unidad de control general
Capítulo 11 completo. Memoria
Capítulo 13 completo. Computador SISC Von Neumann

Algunos objetivos implícitos en estos capítulos de la documentación serán evaluados en el 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. Aunque las cinco primeras preguntas del informe previo sirven para repasar lo que se preguntará en la prueba previa, dejamos a vuestro criterio el número de apartados a contestar en cada pregunta. Sin embargo, sí que exigiremos que respondáis a las preguntas 6 a 10 del informe previo para que podáis realizar la práctica en el lab ya que es imprescindible para poder realizar el informe final, aunque sobre esto no preguntaremos nada directamente en la prueba previa.

5.1 Introducción

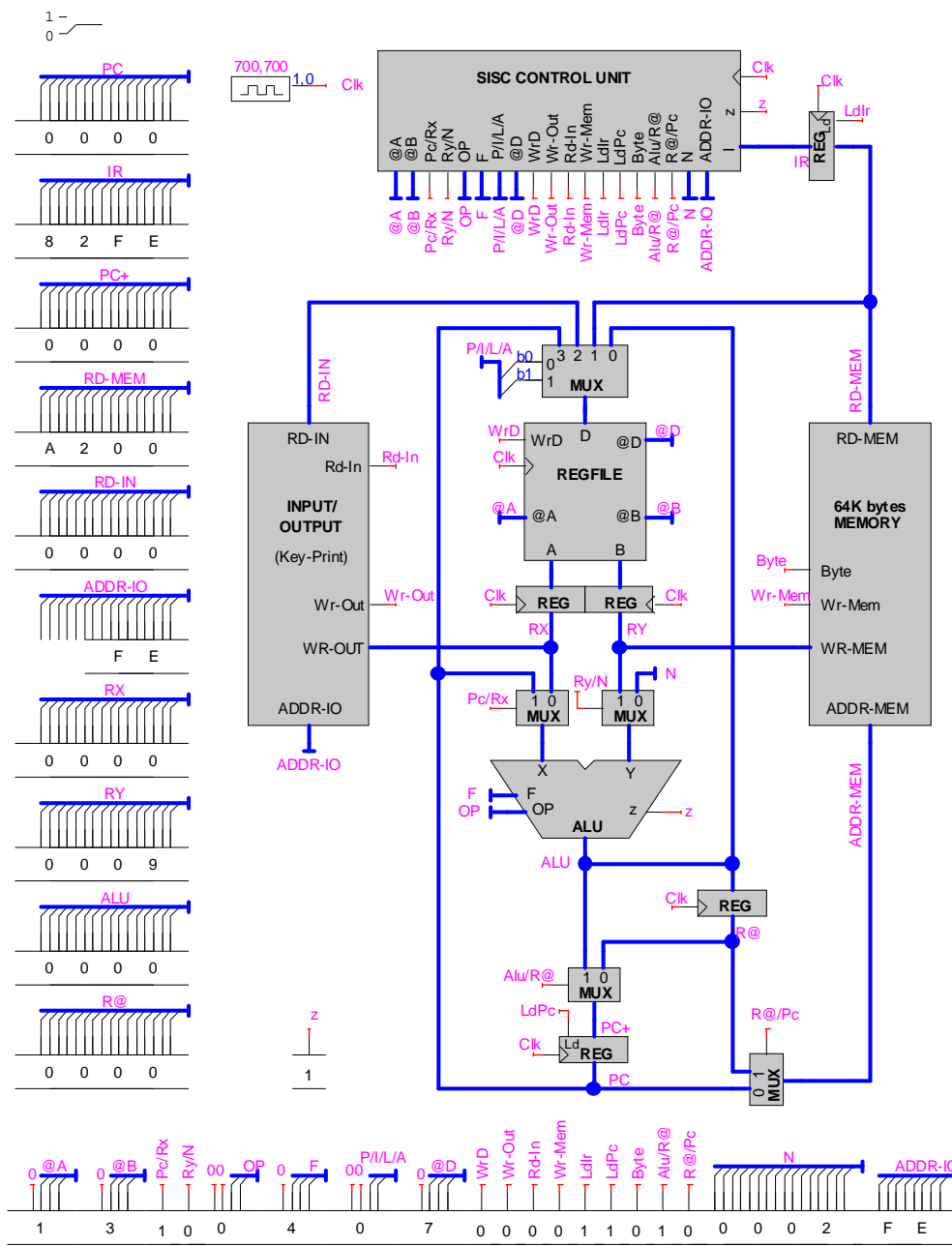
Ya hemos trabajado con la multiplicación durante las prácticas anteriores:

- En la práctica 3 realizasteis en hardware (con el simulador LogicWorks) un Procesador de Propósito Específico para multiplicar dos números naturales codificados en binario con 16 bits,
- En la práctica 4 realizasteis el multiplicador usando la unidad de proceso general, UPG, y diseñasteis la unidad de control específica para que funcionara como en la práctica 3.

En esta práctica os damos construido el computador SISC. Es un computador Von Neumann: tanto las instrucciones del programa a ejecutar como los datos están almacenados en una única memoria RAM. Ejecuta programas en lenguaje máquina SISA (la versión completa del lenguaje con sus 25 instrucciones de 16 bits). El computador SISC es multiciclo, las instrucciones de acceso a memoria tardan 4 ciclos en ejecutarse mientras que el resto tardan 3 ciclos, con un tiempo de ciclo de 1400 u.t.

En el lenguaje máquina SISA no hay ninguna instrucción para multiplicar dos números naturales. Las únicas operaciones aritméticas que se pueden realizar directamente ejecutando una sola instrucción son suma y resta, multiplicación y división por potencias de 2 y algunas comparaciones. Así pues, si se desea multiplicar dos números hay dos posibilidades: añadirle al computador un co-procesador hardware para multiplicar (que podría ser el PPE de las prácticas 3 o 4 con un controlador para conectarlo al SISC) o, lo que es más económico, escribir un pequeño programa que, usando las instrucciones disponibles en SISA, calcule la multiplicación de dos números. Esta segunda solución es la que desarrollamos en esta práctica.

Como objetivo principal de esta práctica escribiréis, en el informe previo, un algoritmo equivalente al que ya habéis diseñado en la práctica 3 (dibujando un grafo de estados para especificar la unidad de control específica que junto con una unidad de proceso específica formaban un PPE) y en la primera parte de la práctica 4 (con la unidad de proceso general, UPG). Luego, escribiréis y ejecutaréis en el laboratorio un algoritmo que es una optimización del primero que consigue, en muchos casos, multiplicar en menos ciclos. Este algoritmo, de terminación temprana, es el que usasteis para implementarlo en la UPG en la segunda parte de la práctica 4. Como veis, esta práctica es muy parecida a la 4 y muchos de los ejercicios son parecidos a los de la práctica 4. Aunque aquí, en vez de diseñar un grafo de estados para una unidad de control específica para la UPG, escribiréis un programa en el lenguaje SISA que luego cargaréis en la memoria del computador SISC para su ejecución. La siguiente figura muestra el esquema a bloques del computador SISC.



5.2 El lenguaje máquina y ensamblador del SISA

Los documentos de Lenguaje Máquina y Ensamblador SISA, que se encuentran en el tema 14 del curso de IC en Atenea describen el lenguaje máquina SISA y su ensamblador. Debéis estudiar en profundidad estos documentos y saber qué hace cada una de las instrucciones de este lenguaje.

Para escribir programas usaremos mnemotécnicos para especificar cada una de las instrucciones. Es mucho más claro de comprender y más difícil equivocarse si escribimos una instrucción con mnemotécnicos, por ejemplo, **ADD R3, R1, R2** que si la escribimos como una secuencia de ceros y unos, 0000 001 010 011 100, o incluso si la escribimos codificada en hexadecimal, que es más compacto, **0x029A**.

No obstante, aunque los humanos preferimos los mnemotécnicos, los computadores solo entienden las tiras de ceros y unos. Cada instrucción SISA es una palabra de 16 bits. Para ejecutar un programa, la secuencia de instrucciones que lo forman debe estar almacenada a partir de la dirección 0 de la memoria del SISC.

Un programa escrito con mnemotécnicos decimos que está escrito en lenguaje ensamblador, mientras que si está escrito con ceros y unos (o en hexadecimal, para que quede más compacto) decimos que está escrito en lenguaje máquina. El programa que pasa un código escrito en lenguaje ensamblador a uno en lenguaje máquina se denomina programa ensamblador (es una especie de compilador sencillo). El lenguaje ensamblador ofrece algunas ayudas para escribir programas cómodamente. Una de ellas la adelantamos a continuación, ya que resulta muy útil.

Usamos el siguiente fragmento de código, que realiza la entrada de un dato del teclado, a modo de ejemplo:

```
POLLING:      IN      R1, KEY-STATUS      ; Input port 1
              BZ      R1, POLLING        ; espera a teclado listo
              IN      R1, KEY-DATA       ; Input port 0
```

Cuando se escribe en lenguaje ensamblador una instrucción de salto, BZ R1, POLLING en el ejemplo, se indica mediante una etiqueta o dirección simbólica (POLLING) la dirección de la instrucción destino del salto tomado. El programa ensamblador se encarga de calcular el número de instrucciones desde la siguiente instrucción que hay a continuación de la instrucción de salto hasta la etiqueta, para codificar en complemento a dos este número de instrucciones (-2, en el ejemplo, ya que el salto es hacia arriba) y ponerlo en el campo N8, de 8 bits, de la instrucción en lenguaje máquina (N8=0x0xFE). A partir de ahora podemos usar las dos notaciones equivalentes para escribir en ensamblador un salto como el del ejemplo: BZ R1, POLLING o BZ R1, -2.

Tanto el formato y codificación en binario de cada instrucción como su notación con mnemotécnicos y su semántica se especifican en la hoja resumen *SISA Reference Data Sheet*. Usando este documento de consulta, contestad a las dos preguntas siguientes (las tablas de formato y codificación de las instrucciones no hay que saberlas de memoria, la semántica de las instrucciones sí).

5.2.1 Ensamblado: del lenguaje ensamblador al lenguaje máquina

➤ Informe previo

Pregunta 1

Traducid a lenguaje máquina cada una de las siguientes instrucciones SISA especificadas en lenguaje ensamblador en la siguiente tabla (las dos primeras filas os las damos traducidas). Indicad como “instrucción no válida” las instrucciones que no puedan codificarse en lenguaje máquina SISA, porque no exista esa funcionalidad o porque los valores numéricos indicados no puedan codificarse en los bits de que dispone el campo correspondiente de la instrucción. Lo siguiente no es un programa, cada fila de la tabla es un apartado del ejercicio.

Lenguaje ensamblador	Lenguaje máquina (L.M.) (binario)	L.M. (hexa)
ADDI R2, R0, -1	0010000010111111	0x20BF
ADDI R5, R0, -120	Instrucción no válida	---
BNZ R2, -6		
SHL R7, R7, R3		
ADD R6, R6, R6		
MOVI R0, -100		
BZ R4, 2		
CMPLT R2, R2, R3		
CMPLEU R4, R7, R1		
MOVHI R5, 0xA4		

5.2.2 Desensamblado: del lenguaje máquina al lenguaje ensamblador

➤ Informe previo

Pregunta 2

Especificad en lenguaje ensamblador cada una de las siguientes instrucciones que os damos en lenguaje máquina SISA (notación hexadecimal). Indicad como “instrucción no válida” las

instrucciones cuyo código no corresponda con ninguna instrucción del SISA (os damos ya contestadas las dos primeras filas de la tabla).

Lenguaje máquina (hexa)	Lenguaje máquina (L.M.) (binario)	Lenguaje ensamblador
0x20C3	0010000011000011	ADDI R3, R0, 3
0x1052	0001000001010010	Instrucción no válida
0x0FCF		
0x7000		
0x4200		
0x6282		
0xA4B2		
0x9DF8		
0x80AF		
0x1FF4		

5.2.3 Ejecución: modificación del estado del computador

Se define el estado del computador como el contenido de todas las posiciones de la memoria (de datos e instrucciones en el SISC), de los 8 registros generales del computador (R0..., R7), de todos los puertos (registros) de entrada/salida y del contador de programa (PC). El estado del computador es “una fotografía” de la memoria del computador. La ejecución de una instrucción produce una modificación del estado del computador. La ejecución de un programa puede verse como una secuencia de modificaciones del estado del computador.

Con esta definición queda claro que no forman parte del estado del computador los registros intermedios que mantienen información entre ciclos/fases de la ejecución de una instrucción en el computador SISC. Así, el contenido de IR, RX, RY o R@ no forman parte del estado del computador ya que cuando empieza la ejecución de una nueva instrucción, al inicio de la fase F, no importa el valor que tengan estos registros: no pasan información de una instrucción a otra.

➤ Informe previo

Pregunta 3

Suponemos que el estado del computador antes de iniciar la ejecución de cada una de las instrucciones (cada uno de los apartados) de esta pregunta es el siguiente:

- La palabra (16 bits) de memoria cuya dirección del byte de menos peso es k contiene el valor k , $MEM_w[k] = k$, para $0 \leq k \leq 2^{15}-1$.
- El contenido del registro general R_k vale $k \% 2$ (el resto de dividir el número natural k entre 2) para $0 \leq k \leq 7$. Así, R0, R2, R4 y R6 contienen 0x0000 y R1 y R3, R5 y R7 contienen 0x0001.
- El contenido del registro PC vale 0x00AE.
- El valor de los puertos de entrada/salida no lo especificamos, pues no es necesario en el ejercicio que vamos a hacer.

Suponiendo que la instrucción que aparece en cada uno de los apartados de este ejercicio se encuentra en la dirección 0x00AE de la memoria y que el estado del computador antes de la ejecución de la instrucción es el que se ha especificado anteriormente, ¿cuánto vale el estado del computador después de la ejecución de la instrucción? Solo tenéis que especificar los cambios ocurridos en el estado del computador tras la ejecución de la instrucción. Los registros y posiciones de memoria que no se modifiquen no deben especificarse (los dos primeros apartados los hemos completado nosotros, como muestra). Para especificar el valor del byte de memoria cuya dirección es k usad la notación $MEM_b[k] = \dots$

- a) ADDI R3, R1, 7
Respuesta: R3 = 8 // PC = 0x00B0
- b) ADD R3, R4, R5
Respuesta: R3 = 1 // PC = 0x00B0
- c) BNZ R3, -6
- d) SHL R7, R7, R2
- e) SHA R7, R7, R2
- f) CMPLEU R5, R7, R3

g) CMPEQ R5, R7, R3
h) BZ R1, -1
i) ADDI R3, R3, -3
j) AND R5, R1, R7
k) LD R2, 30(R5)
l) STB 3(R0), R2
m) ST -26(R5), R4

5.2.4 Programación: del grafo de estados al programa SISA

En la práctica 5 os acostumbrasteis a especificar la unidad de control de propósito específico que gestiona las acciones sobre la UPG mediante un grafo de estados. En este grafo el estado siguiente (secuenciamiento) se indica con los arcos que salen de cada nodo y la palabra de control mediante mnemotécnicos.



En el tema 10 creamos la unidad de control general, que ejecuta instrucciones SISA, de 16 bits cada una. Después creamos los computadores Harvard unicolor y multicolor y ahora el SISC Von Neumann. La única forma de especificar lo que debe hacer nuestro computador es mediante un programa en lenguaje máquina SISA. La unidad de control del SISC se encargará de generar en cada uno de los ciclos de ejecución de cada instrucción la palabra de control a partir de la instrucción SISA que se está ejecutando.



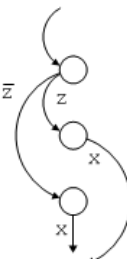
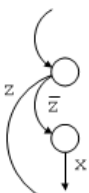
El hecho de haber tenido que compactar la información de cada nodo del grafo de estados (secuenciamiento y palabra de control) en solo 16 bits hace que lo que antes se podía hacer en un solo ciclo, en un nodo, ahora requiera, en algunos casos, más de una instrucción SISA (y más de tres o cuatro ciclos que es lo que tarda en ejecutarse una instrucción SISA en el SISC). Es muy importante que tengáis claro esto, ya que, para implementar el algoritmo de multiplicación en esta práctica vamos a partir del grafo que hicisteis en la práctica anterior. Vamos a ejercitar, en general, este paso, antes de programar la multiplicación.

➤ Informe previo

Pregunta 4

Escribid en lenguaje ensamblador SISA, para cada apartado/fila de la siguiente tabla, un pequeño fragmento de programa que realice la misma funcionalidad que el fragmento de grafo que se especifica. El grafo es el grafo de estados de la unidad de control de propósito específico que genera la palabra de control para la UPG + IO_{key-Print} + Mem. El grafo se especifica mediante nodos y arcos (para el secuenciamiento) y mnemotécnicos (para la palabra de control que indica las acciones sobre la UPG + IO_{key-Print} + Mem). A la derecha de cada fila de la tabla se han reservado unas líneas (sombreadadas) para poner las instrucciones SISA, pero esto no quiere decir que cada fragmento de programa deba ocupar todas las líneas reservadas.

Fragmento de grafo con mnemotécnicos para la palabra de control	Fragmento de programa en lenguaje ensamblador SISA
<p>a)</p>  <p>AND R1, R2, R3</p>	<div style="background-color: #e0e0e0; height: 20px; width: 100%;"></div> <div style="background-color: #e0e0e0; height: 20px; width: 100%;"></div> <div style="background-color: #e0e0e0; height: 20px; width: 100%;"></div>
<p>b)</p>  <p>SHAI R7, R7, -3</p>	<div style="background-color: #e0e0e0; height: 20px; width: 100%;"></div> <div style="background-color: #e0e0e0; height: 20px; width: 100%;"></div> <div style="background-color: #e0e0e0; height: 20px; width: 100%;"></div>

<p>c)</p>  <p>MOVI R3, 327</p>	<div></div> <div></div> <div></div>
<p>d)</p>  <p>MOVI R1, -22</p>	<div></div> <div></div> <div></div>
<p>e)</p>  <p>SUBI -, R2, 1</p> <p>ADD R3, R5, R5</p> <p>SUB R3, R4, R5</p>	<div></div> <div></div> <div></div> <div></div>
<p>f)</p>  <p>CMPLTUI -, R2, 250</p> <p>SHL R4, R1, 4</p>	<div></div> <div></div> <div></div> <div></div> <div></div>

5.2.5 Programación: del programa en alto nivel al programa SISA

Otra forma de plantearse la escritura de un programa SISA es partir de un programa en un lenguaje de alto nivel, como por ejemplo el C, y traducirlo a ensamblador. En la práctica 5 ya planteamos el paso de fragmentos de programas de C a grafos de estados. Ahora podemos hacer directamente el paso de programas de C a ensamblador SISA. Vamos a practicar esto antes de pasar a la multiplicación.

➤ Informe previo

Pregunta 5

Escribid un fragmento de código ensamblador SISA que implemente cada una de las sentencias siguientes, que se especifican en pseudo C (considerad que los tipos de datos que hay en los registros y memoria son números enteros codificados en complemento a 2 con 16 bits) y que el vector de números enteros V está almacenado a partir de la dirección de memoria 0x1000. Recordad que cada índice i del vector V hace referencia a un elemento del vector y cada elemento ocupa dos bytes en direcciones consecutivas de memoria y formato *Little Endian*. Así, el byte de menor peso de $v[3]$ está almacenado en la dirección de memoria 0x1006 y el de más peso en la 0x1007.

```

a) R4 = 0;
b) V[R2] = R3 * 2;
c) V[10] = V[R2 + 3];
d) if (R3 <= R1) R3 = R1 - 1;
e) if (R1 >= 320) R2 = R2 + R2;
    else R5 = R2 + R5;
f) for (R2 = 3; R2 <= R5; R2 = R2 + 1) {
    V[R1 + R2 + 25] = 0;
}
g) for (R3 = 0; R3 < 16; R3 = R3 + 1) {
    V[R3 + R2] = 0;
}

```

Ahora que ya hemos practicado el lenguaje SISA, vamos a programar el computador basado en el SISC, con un algoritmo para multiplicar dos números naturales.

5.3 Un algoritmo de multiplicación con n iteraciones

Enunciado del problema

Suponiendo que el vector de bits X_u , que representa el multiplicando X_u , se encuentra inicialmente en el registro R6 del procesador y que el vector de bits Y_u , que representa al multiplicador, Y_u , se encuentra en R7, escribir un programa en ensamblador del SISA 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 16 bits.

5.3.1 Algoritmo en lenguaje de alto nivel

En la práctica 3 obtuvimos un algoritmo que calcula en 16 ciclos la multiplicación de los números naturales X_u e Y_u y lo implementamos en LogicWorks como un procesador de propósito específico. Este algoritmo, expresado en un lenguaje de alto nivel, suponiendo que los operandos X_u e Y_u se encuentran almacenados en los registros R6 y R7 respectivamente y el resultado se obtiene en R5 se muestra a continuación. Se han omitido los subíndices u en los registros que indicarían que interpretamos los vectores de bits en ellos contenidos como números naturales codificados en binario. Recordad que $R7<0>$ es el bit 0, el de menor peso, de R7.

Algoritmo MUL16 en alto nivel

```

R5 = 0;
for (j = 0; j < 16; j = j + 1) {
    if (R7<0> == 1) R5 = R5 + R6;
    R6 = R6 * 2;
    R7 = R7 / 2;
}

```

5.3.2 Grafo de la UC para la ejecución de MUL16 en la UPG

En la práctica 4, a partir del algoritmo anterior, diseñamos la unidad de control de propósito específico para que la UPG ejecutara el algoritmo MUL16. El grafo de estados resultante se muestra a continuación. La palabra de control que genera en cada ciclo la unidad de control para que la UPG realice las acciones pertinentes se especifica con mnemotécnicos. La variable j del algoritmo, que hace de contador del número de iteraciones del bucle, se ha implementado con el registro R2 de la UPG. Como el bucle se ejecuta más de una vez, de hecho se ejecuta 16 veces, se ha implementado el algoritmo inicializando R2 a 16, entrando en el bucle sin preguntar si hay que ejecutarlo, haciendo el cuerpo del bucle, decrementando en una unidad R2 y preguntando si hay que volver a ejecutar el bucle una vez más. Esta forma de hacer es un poco más eficiente, y se puede escribir en un lenguaje de alto nivel así:

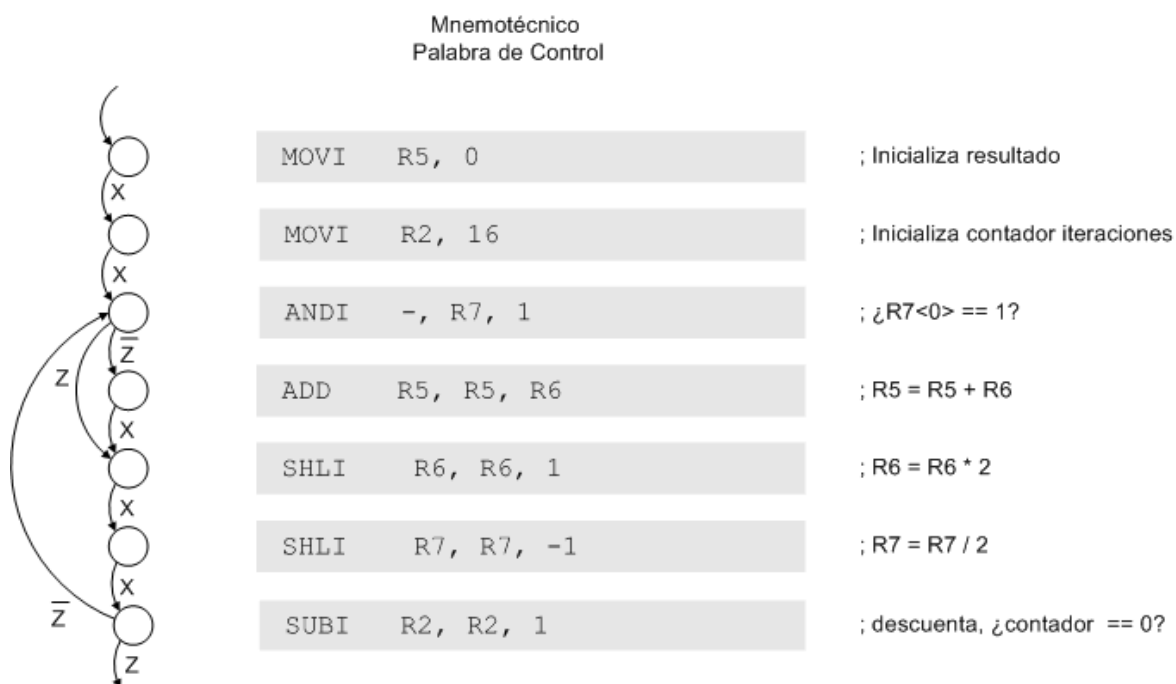
Algoritmo MUL16 en alto nivel con do_while

```

R5 = 0;
R2 = 16;
do {
    if (R7<0> == 1) R5 = R5 + R6;
    R6 = R6 * 2;
    R7 = R7 / 2;
    R2 = R2 - 1;
} while (R2 > 0);

```

Grafo de estados para ejecutar en la UPG el algoritmo MUL16



5.3.3 Obtención del algoritmo MUL16 en SISA

Ahora con la experiencia de los ejercicios que hemos hecho hasta ahora en el informe previo, podemos escribir el algoritmo MUL16 en el lenguaje ensamblador SISA. Para ello, podemos partir del algoritmo en alto nivel o del grafo de estados que hemos visto en los apartados anteriores.

➤ Informe previo

Pregunta 6

Completad el siguiente código escrito en lenguaje ensamblador SISA para que implemente el algoritmo MUL16. Se muestran algunas instrucciones o parte de ellas así como los comentarios, para ayudar en el proceso.

Algoritmo MUL16 en SISA

MOVI R5, 0	; Inicializa resultado
	; Inicializa contador iteraciones
MOVI R1, 1	; Mascara bit 0
	; R3= Constante para dividir por 2
for: AND R4, ,	; ¿R7<0> == 1?
	; si no ir a endif
	; R5 = R5 + R6
endif: R6, R6,	; R6 = R6 * 2
R7, ,	; R7 = R7 / 2
ADDI R2, R2, -1	; R2 = R2 - 1
R2, for	; if (R2 != 0) goto for

5.3.4 Seguimiento del algoritmo MUL16

➤ Informe previo

Pregunta 7

Seguid la ejecución del algoritmo **MUL16** instrucción a instrucción rellenando la siguiente tabla (para las primeras 20 instrucciones que se ejecuten), una fila por instrucción ejecutada. Solo anotaremos el estado del computador en el ciclo de Fetch de cada instrucción.

Suponed que en el ciclo 0 el registro R6 contiene el vector de bits 0x0003 y R7 el 0x0005. Como el resto de registros involucrados en el algoritmo no importa el valor inicial que tengan, hemos puesto en la tabla valores iniciales 0xFFFF. Suponemos que la primera instrucción del algoritmo empieza su ejecución (se hace el Fetch, F) en el ciclo 0 y que el código se encuentra almacenado en memoria a partir de la dirección 0x000C (por eso el valor del PC en la primera fila de la tabla, en el ciclo 0, es 0x000C). Los registros que no se modifican no se deben escribir en la tabla. Así, el valor final de cada registro será el de la última fila en que se modificó. Nosotros hemos rellenado las tres primeras filas para que veáis como se hace. Recordad que en el SISC las instrucciones de acceso a memoria tardan 4 ciclos en ejecutarse y el resto 3.

Ciclo Fetch	Instrucción en ensamblador que se va a ejecutar	Estado de los registros, en el ciclo en que se hace el Fetch de la instrucción (en hexadecimal)								
		PC	R0	R1	R2	R3	R4	R5	R6	R7
0	MOVI R5, 0	000C	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	0003	0005
3	MOVI R2, 16	000E						0000		
6	MOVI R1, 1	0010			0010					

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

- ¿Cuántos ciclos tarda en ejecutarse el código completo en el computador SISC?
- ¿Cuál es el estado del computador (el valor de los registros del procesador que se han modificado) después de ejecutarse el código completo?

5.4 MUL: un algoritmo de multiplicación con terminación temprana

➤ Informe previo

Pregunta 8

Modificad adecuadamente el algoritmo **MUL16** en SISA que obtuvimos en la sección 6.3.3 (pregunta 6 del informe previo) para que no tengan que ejecutarse siempre 16 iteraciones del bucle. Cuando Y , la representación en binario del multiplicador, 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 (numerando las iteraciones de 0 a $n-1$ con $n = 16$). Si continuáramos con el algoritmo sumariamos un 0 a $R5$ en cada nueva iteración y esto no modifica $R5$. El algoritmo resultante lo denominamos “con terminación temprana” y lo denotamos como **MUL**. Es equivalente al grafo de estados de la unidad de control de propósito específico que obtuvimos en la práctica 5 para multiplicar con la UPG.

Completad el siguiente código escrito en lenguaje ensamblador SISA para que implemente el algoritmo **MUL**. Se muestran algunas instrucciones o parte de ellas así como los comentarios o parte de ellos, para ayudar en el proceso.

Algoritmo MUL en ensamblador SISA

	MOVI R1, 1	; Mascara bit 0
for:	AND R4, ,	; ¿R7<0> == 1?
		; si no ir a endif
		; R5 = R5 + R6
endif:	R6, R6,	; R6 = R6 * 2
	, for	

➤ Informe previo

Pregunta 9

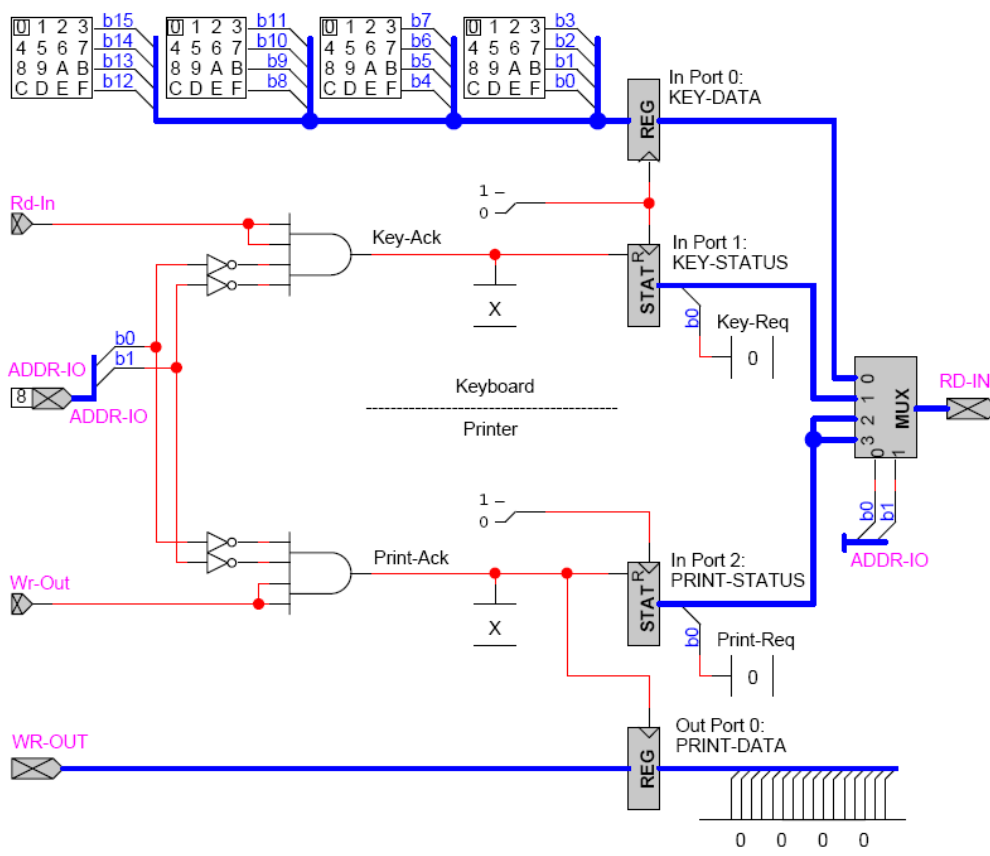
Seguid la ejecución del algoritmo **MUL** con las mismas suposiciones usadas en la pregunta 7 del informe previo para el algoritmo **MUL16** excepto que ahora en el ciclo 0 el registro $R6$ contiene el vector de bits 0x0081 y $R7$ el 0x0005. En este caso nosotros hemos rellenado completamente solo la primera fila.

Los N datos se almacenan en un vector que comienza en la dirección 0xFFE0 de memoria (podría ser cualquier otra dirección, con un pequeño cambio del código). Una vez entrados todos los datos se imprimen en el mismo orden en que se entraron. Se considera que N es un número natural y cada uno de los N datos son enteros codificados en 8 bits. En el display de la impresora se verá cada dato (byte) en 16 bits, previa extensión del bit de signo (bit 7). Terminada la impresión el programa vuelve al inicio repitiéndose el proceso indefinidamente.

El código se cargará en memoria, para su ejecución, a partir de la dirección 0x0000, para que al hacer *reset* del computador se comience su ejecución, ya que el PC se inicializa con el valor 0x0000. Por ello, si N es mayor que 31 (0x001F) el programa dejará de funcionar ya que al almacenar, por ejemplo, el dato número 32 (0x0020), lo almacenaría en la dirección 0x000 de la memoria, donde se encuentra el byte de menor peso de la primera instrucción del programa (IN R1, KEY-STATUS), destruyendo la instrucción, de forma que cuando se fuera a ejecutar otra vez ejecutaría otra instrucción o una instrucción ilegal.

Repasad el código que os damos a continuación para comprenderlo, o lo que sería mejor para vuestro aprendizaje: escribid vosotros el programa sin mirar nuestra propuesta. Los programas para hacer esto correctamente son muchos, más o menos eficientes, usando unos registros u otros...

Veamos nuestra propuesta de programa. En las tres instrucciones a partir de la etiqueta POLLING-1 (IN, BZ e IN) se entra N , en las de la etiqueta POLLING-2 se entra un dato y el las tres de la etiqueta POLLING-3 se imprime un dato. Fijaos que hemos usado una forma un tanto extraña de acceder a los elementos del vector mediante la instrucción STB -32(R0),R3 en el cuerpo del bucle en el que se escribe un dato por iteración y LDB R3,-32(R0) en el bucle de lectura, donde R0 es el contador del bucle que se inicializa a 0 y se incrementa en uno para acceder al siguiente dato. Hemos usado esta forma de acceso por hacerlo diferente a lo usual y que veáis que también es correcto.



```

BEGIN:
POLLING-1:  IN      R1, KEY-STATUS      ; Input port 1
              BZ      R1, POLLING-1    ; espera a teclado listo
              IN      R1, KEY-DATA      ; Input port 0
                                              ; en R1 Numero datos a entrar
                                              ; Comienza la entrada de datos y
                                              ; su almacenamiento
              MOVI    R0, 0             ; inicializa indice de bucle y vector
WHILE-1:    CMPLTU   R2, R0, R1         ; while (r0 < r1) {
              BZ      R2, FI-WHILE-1

```

```

POLLING-2:  IN      R3, KEY-STATUS
            BZ      R3, POLLING-2
            IN      R3, KEY-DATA      ; dato leído del teclado

            STB      -32(R0), R3      ; almacena a partir de la dir. 0xFFE0

            ADDI     R0, R0, 1        ; incrementa índice
            BNZ      R2, WHILE-1      ; }
                                           ; Comienza la lectura de datos de
FI-WHILE-1:                                ; memoria y su impresión
            MOVI     R0, 0            ; inicializa índice de bucle y vector

WHILE-2:    CMPLTU   R2, R0, R1        ; while (r0 < r1) {
            BZ      R2, FI-WHILE-2

            LDB      R3, -32(R0)      ; lee dato de memoria

POLLING-3:  IN      R4, PRINT-STATUS  ; Input port 2
            BZ      R4, POLLING-3    ; espera a printer lista
            OUT      PRINT-DATA, R3   ; Output port 0

            ADDI     R0, R0, 1        ; incrementa índice
            BNZ      R2, WHILE-2      ; }

FI-WHILE-2:                                ;
            BZ      R2, BEGIN         ; ir a BEGIN, bucle infinito

```

Como en el trabajo final, en el laboratorio, vamos a ejecutar este código en el SISC implementado en LogicWorks, tenemos que pasarlo a lenguaje máquina SISA. Esto lo hemos hecho nosotros pero es muy conveniente que lo hagáis vosotros a modo de ejercicio. Mostramos en tres columnas: el código ensamblador, el código binario (en lenguaje máquina) pero separando los campos de la instrucción según su formato (3R, 2R o 1R) para ver con más claridad su codificación y el código en hexadecimal.

Ensamblador SISA		binario SISA	Hexa
BEGIN:			
POLLING-1:	IN R1, KEY-STATUS	1010 001 0 00000001	A201
	BZ R1, POLLING-1	1000 001 0 11111110	82FE
	IN R1, KEY-DATA	1010 001 0 00000000	A200
	MOVI R0, 0	1001 000 0 00000000	9000
WHILE-1:	CMPLTU R2, R0, R1	0001 000 001 010 100	1054
	BZ R2, FI-WHILE-1	1000 010 0 00000110	8406
POLLING-2:	IN R3, KEY-STATUS	1010 011 0 00000001	A601
	BZ R3, POLLING-2	1000 011 0 11111110	86FE
	IN R3, KEY-DATA	1010 011 0 00000000	A600
	STB 32(R0), R3	0110 000 011 100000	60E0
	ADDI R0, R0, 1	0010 000 000 000001	2001
	BNZ R2, WHILE-1	1000 010 1 11111000	85F8
FI-WHILE-1:	MOVI R0, 0	1001 000 0 00000000	9000
WHILE-2:	CMPLTU R2, R0, R1	0001 000 001 010 100	1054
	BZ R2, FI-WHILE-2	1000 010 0 00000110	8406
	LDB R3, 32(R0)	0101 000 011 100000	50E0
POLLING-3:	IN R4, PRINT-STATUS	1010 100 0 00000010	A802
	BZ R4, POLLING-3	1000 100 0 11111110	88FE
	OUT PRINT-DATA, R3	1010 011 1 00000000	A700
	ADDI R0, R0, 1	0010 000 000 000001	2001
	BNZ R2, WHILE-2	1000 010 1 11111000	85F8
FI-WHILE-2:	BZ R2, BEGIN	1000 010 0 11101010	84EA

5.6 Añadiendo Entrada/Salida al algoritmo MUL: KEY_MUL_PRINT

➤ Informe previo

Pregunta 10

Escribid un programa en SISA, que denominamos **KEY_MUL_PRINT**, que haga lo siguiente:

- 1) lea el valor del multiplicando por el teclado del computador,
- 2) lea el valor del multiplicador por el teclado del computador,
- 3) multiplique los dos valores leídos en los pasos 1 y 2 usando el algoritmo MUL que diseñasteis en la pregunta 8 del informe previo,
- 4) muestre el resultado de la multiplicación por la impresora del computador y
- 5) vaya otra vez a realizar el paso 1), y así indefinidamente.

Por lo tanto, el programa **KEY_MUL_PRINT** se obtiene añadiendo, a las 9 instrucciones del programa **MUL**, 6 instrucciones al principio para entrar los 2 operandos, 3 después de MUL para imprimir el resultado y una última para romper la secuencia incondicionalmente al principio del programa (aprovecharemos que R7 vale 0 al salir de MUL para hacer esta ruptura de secuencia incondicional). En total, el programa tiene 19 instrucciones.

Escribid el código del algoritmo **KEY_MUL_PRINT** en ensamblador SISA, traducirlo a lenguaje máquina, código binario SISA, y representadlo en notación hexadecimal separando el byte de más peso (byte-1) y el de menor peso (byte-0), ya que esto os será útil para cargar el programa en el módulo-1 y módulo-0 de la memoria del SISC. Para ello rellenad la siguiente tabla. A modo de ejemplo, hemos completado la primera fila de la tabla.

PC (Hexa)	Lenguaje Ensamblador	Lenguaje Máquina (L.M.) (binario)	L.M. Byte-1 (Hexa)	L.M. Byte-0 (Hexa)
0000	Begin: IN R6, KEY-STATUS	1010 110 0 00000001	AC	01
0002				
0004				
0006				
0008				
000A				
000C				
000E				
0010				
0012				
0014				
0016				
0018				
001A				
001C				
001E				
0020				
0022				
0024				

En el trabajo final, en el laboratorio, vamos a cargar en la memoria del SISC y a ejecutar los códigos en lenguaje máquina de los programas **KEY_MEM_PRINT** y **KEY_MUL_PRINT** que acabamos de ver en este trabajo previo.

Informe previo Práctica-5

Apellidos y nombre: Grupo:

Apellidos y nombre: Grupo:

(por orden alfabético)

Pregunta 1

(Contesta solo a los apartados que consideres oportunos para mejorar tu aprendizaje)

Lenguaje ensamblador	Lenguaje máquina (L.M.) (binario)	L.M. (hexa)
ADDI R2, R0, -1	0010000010111111	0x20BF
ADDI R5, R0, -120	Instrucción no válida	---
BNZ R2, -6		
SHL R7, R7, R3		
ADD R6, R6, R6		
MOVI R0, -100		
BZ R4, 2		
CMPLT R2, R2, R3		
CMPLEU R4, R7, R1		
MOVHI R5, 0xA4		

Pregunta 2

(Contesta solo a los apartados que consideres oportunos para mejorar tu aprendizaje)

Lenguaje máquina (hexa)	Lenguaje máquina (L.M.) (binario)	Lenguaje ensamblador
0x20C3		ADDI R3, R0, 3
0x1052		Instrucción no válida
0x0FCF		
0x7000		
0x4200		
0x6282		
0xA4B2		
0x9DF8		
0x80AF		
0x1FF4		



Pregunta 3



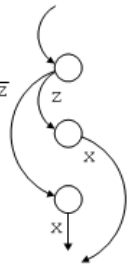
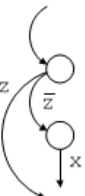
(Contesta solo a los apartados que consideres oportunos para mejorar tu aprendizaje)

- a) ADDI R3, R1, 7
Respuesta: R3 = 8 // PC = 0x00B0
- b) ADD R3, R4, R5
Respuesta: R3 = 1 // PC = 0x00B0
- c) BNZ R3, -6
- d) SHL R7, R7, R2
- e) SHA R7, R7, R2
- f) CMPEU R5, R7, R3
- g) CMPEQ R5, R7, R3
- h) BZ R1, -1
- i) ADDI R3, R3, -3
- j) AND R5, R1, R7
- k) LD R2, 30(R5)
- l) STB 3(R0), R2
- m) ST -26(R5), R4

Pregunta 4

(Contesta solo a los apartados que consideres oportunos para mejorar tu aprendizaje)

Fragmento de grafo con mnemotécnicos para la palabra de control	Fragmento de programa en lenguaje ensamblador SISA
a)  AND R1, R2, R3	<div></div> <div></div> <div></div>
b)  SHAI R7, R7, -3	<div></div> <div></div> <div></div>

<p>c)</p>  <pre>MOVI R3, 327</pre>	<div></div> <div></div> <div></div>
<p>d)</p>  <pre>MOVI R1, -22</pre>	<div></div> <div></div> <div></div>
<p>e)</p>  <pre>SUBI -, R2, 1 ADD R3, R5, R5 SUB R3, R4, R5</pre>	<div></div> <div></div> <div></div> <div></div>
<p>f)</p>  <pre>CMPLTUI -, R2, 250 SHL R4, R1, 4</pre>	<div></div> <div></div> <div></div> <div></div> <div></div>

Pregunta 5

(Contesta solo a los apartados que consideres oportunos para mejorar tu aprendizaje)

a) $R4 = 0;$

b) $V[R2] = R3 * 2;$

c) `V[10] = V[R2 + 3];`

d) `if (R3 <= R1) R3 = R1 - 1;`

e) `if (R1 >= 320) R2 = R2 + R2;
 else R5 = R2 + R5;`

f) `for (R2 = 3; R2 <= R5; R2 = R2 + 1) {
 V[R1 + R2 + 25] = 0;
}`

g) `for (R3 = 0; R3 < 16; R3 = R3 + 1) {
 V[R3 + R2] = 0;
}`

Pregunta_6

Algoritmo MUL16 en SISA

MOV R5, 0	; Inicializa resultado
	; Inicializa contador iteraciones
MOV R1, 1	; Mascara bit 0
	; R3= Constante para dividir por 2
for: AND R4, ,	; ¿R7<0> == 1?
	; si no ir a endif
	; R5 = R5 + R6
endif: R6, R6,	; R6 = R6 * 2
	; R7 = R7 / 2
ADDI R2, R2, -1	; R2 = R2 - 1
	; if (R2 != 0) goto for

Pregunta 7

[illegible]

a) ¿Cuántos ciclos tarda en ejecutarse el código completo en el computador SISC?

