

PRÁCTICA 5. Trabajo final

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

Objetivos que se deben alcanzar al realizar la práctica

Después de realizar esta práctica, además de haber mejorado el nivel de consecución de los objetivos necesarios para preparar la práctica y de los objetivos de la práctica anterior relativos al manejo del programa LogicWorks, el alumno será capaz de:

- 1) Crear y ejecutar programas en el computador SISC, que es una implementación Von Neuman (y multi-ciclo) en LogicWorks de la arquitectura SISA:
 - Cargar un programa en la memoria RAM del SISC en LogicWorks que contenga un programa SISA,
 - Ejecutar el programa (tanto ciclo a ciclo como todo seguido).
- 2) Escribir en lenguaje ensamblador y lenguaje máquina SISA los códigos MUL16, MUL y KEY_MUL_PRINT para la multiplicación de dos números naturales. MUL16 realiza la multiplicación en 16 iteraciones del bucle que multiplica el multiplicando por un bit del multiplicador y acumula el resultado parcial. MUL es un algoritmo de terminación temprana, en el que el número de iteraciones es igual a 16 menos el número de ceros consecutivos que tiene el multiplicador en sus bits de más peso. Tanto MUL16 como MUL presuponen que los operandos están almacenados en dos registros del procesador y el resultado de la multiplicación se deja en otro registro. El algoritmo KEY_MUL_PRINT se obtiene añadiendo instrucciones para entrar los operandos por un teclado y para escribir el resultado en una impresora. La lectura/escritura del puerto de datos del controlador del teclado/impresora tiene un efecto lateral: pone a cero el puerto de estado del controlador del teclado/impresora, para ahorrarnos parte del protocolo de handshaking de cuatro pasos.
- 3) Calcular el número de ciclos de ejecución de un algoritmo codificado en SISA y ejecutado en el SISC

Directorio de la práctica

I(Assignatures):/ic/PRAC5

6.1 Introducción

En este 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 visteis en el trabajo previo. Abrid, con el programa LogicWorks versión 4.1, el circuito SISC-w-Display-wo-Prog.cct y la librería LibPrac5.clf que habréis copiado previamente en vuestro escritorio. El circuito se muestra en la siguiente figura.

6.2 Carga y ejecución del programa KEY_MEM_PRINT

A continuación se muestra el programa KEY_MEM_PRINT 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
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

6.2.1 Carga de la memoria con el programa en lenguaje máquina

Ahora vais a escribir en la memoria RAM del SISC el programa en lenguaje máquina para pasar luego a ejecutarlo ciclo a ciclo.

Resumiendo y simplificando la realidad, lo que ocurre cuando un usuario quiere ejecutar un programa, que está escrito en lenguaje máquina en un fichero en el disco del computador es lo siguiente. El sistema operativo (un programa que está ya cargado en memoria ejecutándose, o al menos parte de él) se encarga de leer el programa del disco y copiarlo, cargarlo, en una zona de memoria que no esté ocupada por el sistema operativo y acto seguido pasar a ejecutar la primera instrucción del programa (cargando en el PC la dirección de memoria a partir de la que el sistema operativo cargó el programa en lenguaje máquina). Pero nuestro computador no tiene sistema operativo ni disco donde está el programa en lenguaje máquina. Por ello, lo que vamos a hacer a continuación para cargar el programa en memoria no es lo usual: usaremos las facilidades del simulador de circuitos, LogicWorks, que nos permite hacerlo con cierta comodidad.

Para cargar el programa en la memoria del SISC tenemos que tener el programa en lenguaje máquina (más en concreto, en formato "raw hex file" para que lo entienda el LogicWorks) repartido en dos ficheros con extensión .hex. Cada fichero se escribirá en uno de los dos módulos de memoria de 32KB que forman la memoria del computador. El fichero que contiene la secuencia de los bytes de menor peso de cada instrucción del programa (cada byte expresado por dos dígitos hexadecimales) se llama Byte-0-KEY-MEM-PRINT.hex y su contenido en hexadecimal es:

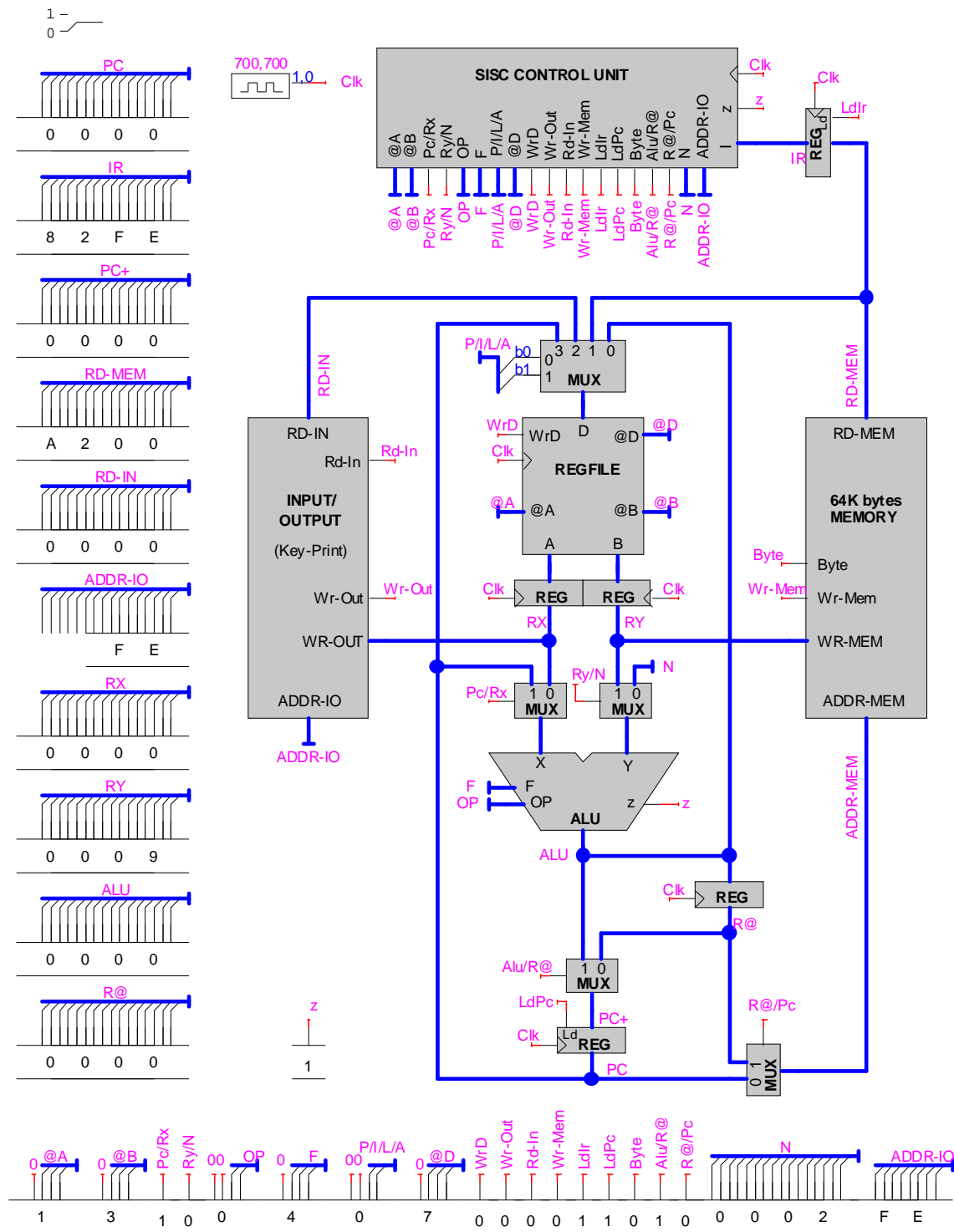
01 FE 00 00 54 06 01 FE 00 E0 01 F8 00 54 06 E0 02 FE 00 01 F8 EA

El fichero que contiene los bytes de mayor peso se llama Byte-1-KEY-MEM-PRINT.hex y su contenido es:
A2 82 A2 90 10 84 A6 86 A6 60 20 85 90 10 84 50 A8 88 A7 20 85 84

La separación entre cada pareja de dígitos hexadecimales puede ser un espacio, como se muestra, una coma, un cambio de línea...


Ambos ficheros se encuentran en la carpeta de esta práctica (así no tenéis que crearlos vosotros y hay menos probabilidades de error).

Observad que cada instrucción se almacena en memoria en formato Little endian. Por ejemplo, la primera instrucción del programa (16 bits = 2 bytes), 1010001000000001 o A201 en hexadecimal, que se ha de cargar en la dirección 0 de la memoria, tiene el byte de menor peso, 01, en la primera posición del fichero cuyo contenido se deberá grabar en el módulo 0 de la memoria y el byte de más peso, A2, en la primera posición del fichero que ha de escribirse en el módulo 1.



Para escribir el programa en el bloque de memoria (MEM-64KB-32KW.cct) que se encuentra en el SISC, en el circuito LogicWorks denominado SISC-w-Display-wo-Prog.cct (SISC con displays y sin programa almacenado en la memoria) que está en la carpeta de esta práctica, se tiene que tener el circuito SISC y la librería LibPrac5, que también está en la carpeta, copiados en el escritorio para que se puedan modificar ya que el disco donde está la carpeta está protegido contra la escritura.

El procedimiento para escribir el programa en la memoria RAM del computador es:

1. Abrid el circuito SISC-w-Display-wo-Prog.cct y la librería LibPrac5.clf si no lo habéis hecho ya.
2. Doble clic en el bloque 64K bytes MEMORY, para abrirlo. Aparecerá su circuito interno formado, básicamente, por dos bloques: 32K bytes MEMORY (1) y (0).
3. Doble clic en el bloque 32K bytes MEMORY (0) para abrirlo y ver el dispositivo RAM que lo forma.
4. Seleccionar el bloque RAM-32KB-0
5. Clic en el botón PLA/PROM/RAM Construction Wizard de la barra de herramientas, . Se abre una ventana donde aparece seleccionado Edit Selected Device.
6. Clic en Siguiente. Aparece un nuevo contenido en la ventana.
7. Seleccionar (clic) en "Read data from a raw hex file" y clic en Siguiente.
8. Clic en "Select Raw Hex File". Navegar hasta seleccionar el fichero Byte-0-KEY-MEM-PRINT.hex que se encuentra en el escritorio. Clic en Abrir del navegador. Clic en Finalizar de la ventana.
9. Salvar el circuito (Clic en el icono "Save" de la barra o en File>Save).
10. Cerrar el circuito abierto.
11. Hacer lo mismo que los pasos 3 a 9 anteriores pero ahora entrando en 32K bytes MEMORY (1) para escribir Byte-1-KEY-MEM-PRINT.hex en el bloque el bloque RAM-32KB-1.
12. Cerrar los circuitos abiertos RAM-32KB-1 y 64K bytes MEMORY y aparece el circuito a bloques del SISC donde está el bloque de memoria 64K bytes MEMORY.
13. Salvar el circuito 64K bytes MEMORY.

Con esto ya tenemos el programa cargado a partir de la dirección 0 de la memoria del SISC. Como la memoria es RAM, es de lectura y de escritura, si hemos cometido un error y al ejecutarse alguna instrucción indeseada escribimos sobre el programa, este queda destruido y la ejecución será errónea.

6.2.2 Crear un dispositivo de la memoria del SISC que contenga el programa cargado

Para no tener que repetir el proceso anterior de escritura de los dos módulos de memoria con el programa vamos a salvar en la librería el bloque 64K bytes MEMORY. Se salva con el contenido actual de las memorias RAM, o sea, con el programa cargado. Así, si perdemos el programa solo tenemos que eliminar el bloque 64K bytes MEMORY del SISC y añadir el de la librería que contendrá el programa. Para ello hay que hacer:

1. En el circuito del computador SISC, con el bloque 64K bytes MEMORY seleccionado, hacer clic en el botón de la barra Schematic, y en el desplegable que se abre soltar en Save Part to Library...
2. Seleccionar:
 - a. Save internal circuit definition
 - b. Save all attributes in the selected instance
 - c. Save positions of all visible attributes in the selected instanceEscribir el nombre del dispositivo en Part Name, por ejemplo 64KBMEM-KEY_MEM_PRINT (que recuerda que este dispositivo es la memoria del computador con el programa ya cargado KEY-MEM-PRINT). Se selecciona la librería LibPrac5, donde se salvará (la que habéis copiado en el escritorio) y clic en Save.

6.2.3 Ejecución ciclo a ciclo del programa

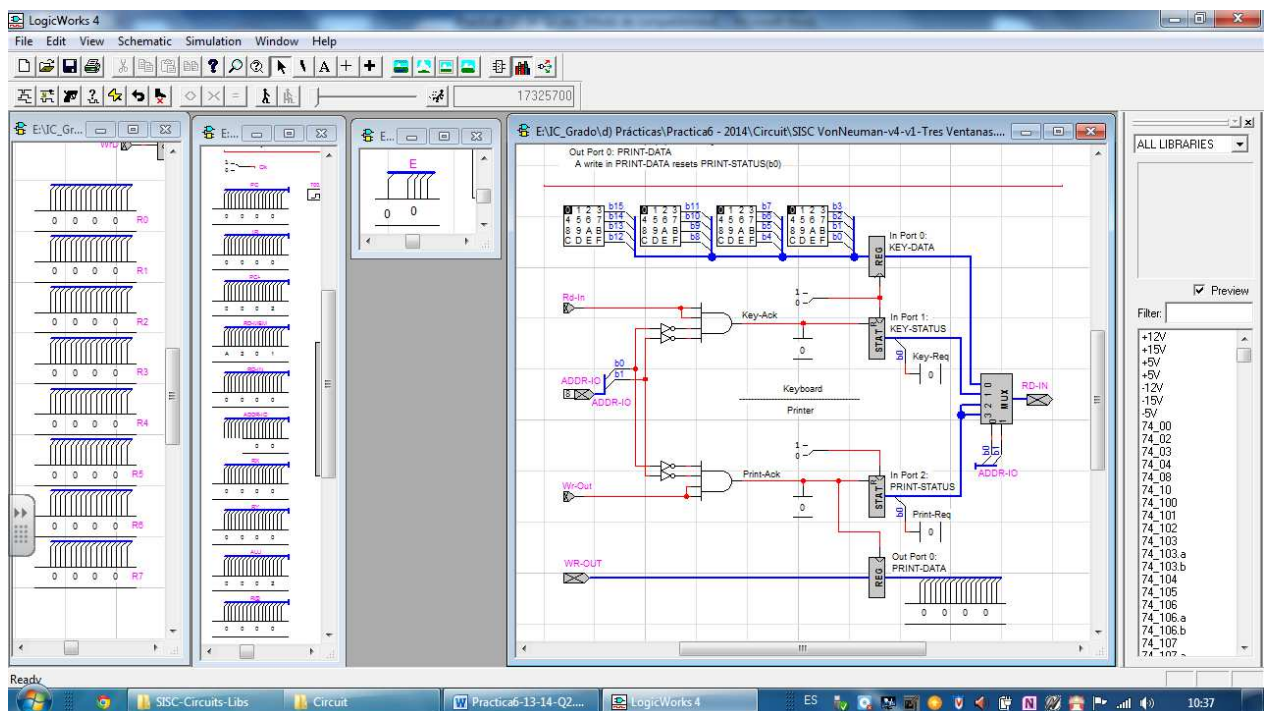
Para comprobar el funcionamiento del programa que acabamos de cargar en memoria vamos a efectuar una ejecución ciclo a ciclo observando y anotando, en una fila de la siguiente tabla por cada instrucción ejecutada el estado del computador durante el ciclo de búsqueda (Fetch) de cada instrucción.

Para ejecutar ciclo a ciclo y que dé tiempo de anotar la información en la tabla, debéis borrar el nombre Clk de la salida del dispositivo de Reloj del SISC y darle el nombre Clk a la señal de salida del binary switch que se encuentra en la parte superior izquierda del circuito (ver figura siguiente). Recordad que en

este programa hay instrucciones que tardan 3 ciclos y otras que tardan 4, aunque solo hay que apuntar contenido de los registros en el ciclo de Fetch de cada instrucción.

Observad con atención los displays hexadecimales que hemos dispuesto a la izquierda del circuito del SISC, para que podáis ver el valor del PC, del IR etc. Para ver el contenido de los registros debéis abrir el bloque que implementa el banco de registros. También en este circuito hemos puesto displays para visualizar el contenido de los registros. Podéis hacer que las ventanas no ocupen toda la pantalla y así ver abiertas varias ventanas a la vez.

Organizad la pantalla para que os queden tres ventanas como se muestra en la siguiente figura. Una ventana fina y alargada para ver el contenido de los registros, otra del mismo tipo para los displays del SISC y el binary switch que sustituye al reloj (que habréis dispuesto encima del display del PC, Una pantalla pequeña del circuito interno del bloque SISC CONTROL UNIT visualizando el estado en el que esta la unidad de control y la última, más grande, para el teclado e impresora. Así os será muy cómodo trabajar. Podéis modificar el Zomm con que se ve cada ventana: Hacer clic en la ventana y en la pestaña View hecer Zoom In o Zoom Out.



❑ Informe final

Pregunta 1:

Vamos a ir ejecutando ciclo a ciclo el programa KEY_MEM_PRINT haciendo clic en el binary switch que hace de reloj y además vamos a teclear los datos en el teclado y a hacer manualmente la parte del protocolo del controlador del teclado y la impresora. Rellenad la siguiente tabla con el resultado de la ejecución del código instrucción a instrucción, para las 10 primeras instrucciones que se ejecutan. Solo hay que apuntar en la tabla el contenido de los registros en el ciclo de Fetch de cada instrucción (al estilo de lo que hicisteis en las preguntas 7 y 9 del informe previo). Aunque esté código sí que modifica algunos puertos de E/S y algunos bytes de memoria no incluimos estos elementos del estado del computador dentro de la tabla que deberéis rellenar. Explicamos, a continuación, los primeros ciclos de ejecución del programa que se encuentra almacenado a partir de la dirección 0x0000 de memoria.

Ciclo 0. Haced reset del sistema, pues es necesario ya que hemos cambiado la memoria de instrucciones. Poned a correr la simulación a máxima velocidad. Observad que el contenido de todos los registros se ha puesto a 0 debido al reset. Además, en los displays del SISC podemos ver que el PC vale 0 y que efectivamente se está leyendo la instrucción 0 de nuestro programa, 0xA201 (IN R1, 1), lo que se ve en el display del bus RD-MEM. Este es el ciclo de Fetch, F (se ve en la ventana de la unidad de control en el visualizador del estado E=00), de la primera

instrucción del programa, por lo que deberíais apuntar los contenidos de los registros en la primera fila de la tabla, pero en este caso no hace falta porque ya lo hemos hecho nosotros.

Podéis aprovechar esta ejecución ciclo a ciclo para ver otras cosas que no hay que anotar en la tabla pero que ocurren en cada ciclo de la ejecución de cada instrucción. Por ejemplo, como este es el ciclo de Fetch de una instrucción, sea cual sea, pues todavía no está decodificada, en la ALU se calcula $PC+2$ y por eso la salida de la ALU vale 0x0002, cosa que se ve en el visor del bus ALU.

Ciclo 1. Este es el ciclo de decodificación de la instrucción 0xA201 (E=01, D). No hay que escribir nada en la tabla. En este ciclo ya se ve en el IR la instrucción que estaba en RD-MEM en el ciclo anterior. **IMPORTANTE para este ejercicio:** para no perder tiempo en el primer bucle de la encuesta del teclado debéis cargar el valor de N en el registro de datos de la impresora en este momento para que cuando se lea el registro de estado en el ciclo siguiente ya esté a 1 y salgamos del bucle del polling a la primera iteración. Para ello, poned en el teclado hexadecimal del bloque de E/S 0x0002 (entraremos una secuencia de dos datos, para tardar poco en ejecutar todo el programa) y haced clic en el switch del teclado para se consiga cargar 0x0002 en el puerto 0 (KEY-DATA) y poner a 1 uno el puerto 1 (KEY-STATUS). El bit de menor peso de la salida de KEY-STATUS se ve en el visor, y ahora ya se ve a 1. Volved a hacer clic en el switch para dejarlo a 0.

Podéis ver, aunque no hay que apuntarlo todavía en la tabla, que en este ciclo ya se ve que el PC vale 0x0002 ya que al final del ciclo anterior se cargó el PC con este valor que había calculado la ALU. También se ve que en este ciclo el IR contiene la primera instrucción del programa (0x A201) que se fue a buscar a memoria en el ciclo anterior.

Ciclo 2. En este ciclo se ejecuta propiamente la instrucción IN R1, KEY STATUS (Estado E=0F, In). Ya puede verse el 0x0001 del KEY-STATUS que se está leyendo en este ciclo en el display del bus RD-IN, que al final del ciclo se cargará en R1

Podéis ver que como en todos los ciclos de decodificación se calcula en la ALU el valor $PC+SE(N8)*2$ que se almacenará al final de este ciclo en R@, para ser usado solo si estamos ejecutando una instrucción de salto y el salto es tomado. Este valor calculado, que se muestra en el visor del bus ALU no se usará al ciclo siguiente porque no se está ejecutando una instrucción de salto. No obstante podéis comprobar que el cálculo que hace la ALU es el correcto.

Ciclo 3. Ciclo de búsqueda (E=00, F) de la segunda instrucción del programa (0x82FE, BZ R1, POLLING-1). Ya se ve el 0x0001 en R1 que cargo la instrucción anterior al final del ciclo anterior. Como es un ciclo de Fetch hay que apuntar en la tabla el valor de los registros en la tabla: PC = 0x0002 y R1 = 0x0001.

Ciclo 4. Ciclo de decodificación (E=01, D) de la instrucción de salto. No hay que anotar nada.

Ahora sí que podría usarse al ciclo siguiente lo que calcula la ALU, que se cargará al final del ciclo en R@, ya que sí que se está ejecutando una instrucción de salto.

Ciclo 5. Ciclo de ejecución del BZ R1, POLLING-1 (E=0B, Bz). No hay que apuntar nada en la tabla.

Pasando a pantalla completa la ventana del SISC se ve, en la parte inferior encima de los visores de la palabra de control que z vale 1 en este ciclo. Esto es así porque R1 no vale cero y la ALU deja pasar R1 que vale 0x0001 y esto hace que z valga 0 y por tanto LdPc vale 0 (como se ve en la palabra de control, en la parte inferior del circuito SISC) para que no se cargue el PC al final de este ciclo con la dirección de salto tomado que se encuentra en R@ (que como se ve es 0x000) si no que se vaya a buscar al ciclo siguiente la que indica el PC en este ciclo, que como puede verse es la 0x0004: No se rompe la secuencia, por lo que PC = 0x0004, como muestra el visor.

Ciclo 6. Búsqueda de la instrucción que indica el PC. La instrucción que está en la dirección contenida en el PC, 0x0004, ya se ve en el bus RD-MEM y es la 0xA200.

Continuad vosotros solos ciclo a ciclo, anotando solo el contenido del PC y del registro modificado por la instrucción anterior (si se modifica alguno) en cada ciclo de Fetch de cada instrucción.

Cuando se ejecute una instrucción IN Rd, KEY-STATUS, escribid el dato en el teclado hexadecimal (por ejemplo 0x000F) y haced clic en el binary switch (para producir un flanco ascendente, cargar el dato en el KEY-DATA y poner a 1 KEY-STATUS) en el ciclo de Decodificación de la instrucción IN, como se hizo en el ciclo 1, para que el bucle de la encuesta se pase en la primera iteración.

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
0	IN R1, KEY-STATUS	0000	0000	0000	0000	0000	0000
3	BZ R1, POLLING-1						
6	IN R1, KEY-DATA						
	MOVI R0, 0						
	CMPLTU R2, R0, R1						
	BZ R2, FI-WHILE-1						
	IN R3 KEY-STATUS						
	BZ R3 POLLING-2						
	IN R3-KEY-DATA						
	STB 32(R0), R3						

Aunque sólo hay que apuntar hasta las 10 primeras instrucciones en la tabla, podéis seguir ejecutando paso a paso el algoritmo completo, hasta sacar por la impresora al menos el primer dato entrado y almacenado en memoria.

□ **Informe final**

Pregunta 2:

Cuando estéis seguros de que sabéis hacer la parte del protocolo de la impresora y del teclado, poned otra vez el reloj original, haced reset y ejecutad el programa KEY_MEM_PRINT a toda velocidad haciendo vosotros el clic sobre el teclado y los binary switch del teclado e impresora. Cuando estéis seguros de hacerlo bien, avisad al profesor de laboratorio y pedidle que revise vuestro trabajo y firme en el informe final.

6.3 Carga y ejecución del programa KEY_MUL_PRINT

En esta segunda y última parte del trabajo final vamos a hacer lo mismo que acabáis de hacer para el programa KEY_MEM_PRINT pero para el programa KEY_MUL_PRINT, que escribisteis en ensamblador y en lenguaje máquina en la pregunta 10 del informe previo.

En concreto:

1. Cargad el programa KEY_MUL_PRINT en la memoria del SISC (en el dispositivo 64K bytes MEMORY). Para ello:
 - a. Cread dos ficheros .hex uno para cargar en el módulo-1 de la memoria y otro para el módulo-0. (con el formato "Raw Hex File" de LogicWorks).
 - b. Cargar estos dos ficheros en la memoria siguiendo los mismos pasos que ya habéis seguido para el programa KEY_MEM-PRINT pero ahora para el KEY_MUL_PRINT (ver sección "Cargar la memoria RAM del SISC con el programa en Lenguaje Máquina".
 - c. Salvar la memoria del SISC (el dispositivo 64K bytes MEMORY con el programa ya cargado) en la librería LibPrac5 nombrando el dispositivo 64KMEM_KEY_MUL_PRINT. Para ello seguid los mismos pasos que ya habéis seguido con el programa KEY_MEM_PRINT pero ahora para el KEY_MUL_PRINT y que se encuentran explicados en la sección "Crear un dispositivo con la memoria del SISC que contenga el programa cargado".
2. Haced las siguientes dos preguntas del informe final, equivalentes a la 2 y 2 pero ahora con el programa KEY_MUL_PRINT.

□ **Informe final**

Pregunta 3:

Para comprobar el funcionamiento del programa KEY_MUL_PRINT vamos a ejecutar ciclo a ciclo del código cargado en memoria y a anotar, en cada ciclo de Fetch de las 20 primeras instrucciones que se ejecuten, la parte del estado del computador que nos piden en la siguiente tabla, como hicisteis con el código KEY_MEM_PRINT en la pregunta 1 del informe final.

Si el código no funciona correctamente a la primera, con este seguimiento podéis detectar los errores. Posiblemente hayáis codificado mal alguna instrucción. El paso de ensamblador a lenguaje máquina es muy engorroso y es fácil cometer errores. Arreglad el error en los ficheros .hex, volver a cargar el programa en la memoria y volvedlo a intentar.

[illegible]

- ❑ Informe final

Quando estéis seguros del correcto funcionamiento del programa **KeyPrintMUL**, avisad al profesor de laboratorio y pedidle que revise vuestro trabajo y firme en el informe final.

Pregunta 5:

8

Informe final Práctica-5

Apellidos y nombre: Grupo:

Pregunta 1 (1.5 puntos)

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
0	IN R1, KEY-STATUS	0000	0000	0000	0000	0000	0000
3	BZ R1, POLLING-1						
6	IN R1, KEY-DATA						
	MOVI R0, 0						
	CMPLTU R2, R0, R1						
	BZ R2, FI-WHILE-1						
	IN R3 KEY-STATUS						
	BZ R3 POLLING-2						
	IN R3-KEY-DATA						
	STB 32(R0), R3						

Pregunta 2 (2 puntos)

Programa **KEY_MEM_PRINT**

Comentario del profesor:

Firma del profesor:

Pregunta 3 (2.5 puntos)

(Tabla de respuesta en la siguiente página)

