12 Computadores SISC Harvard uniciclo y multiciclo

12.1 Introducción

En la primera mitad de este capítulo se termina de diseñar la lógica de control del computador SISC Harvard Uniciclo, considerando las restricciones temporales de las señales de permiso de modificación del estado. A continuación, se calcula el tiempo de ciclo mínimo del computador y se evalúa su rendimiento ejecutando programas. El tiempo de ciclo mínimo es el tiempo de propagación de las instrucciones que más tiempo necesitan para completar su ejecución (las instrucciones de acceso a memoria).

El computador uniciclo requiere un único ciclo para ejecutar cualquier instrucción, sea lenta (como las de acceso a memoria) o rápida (como el resto). Es ineficiente porque el resto de instrucciones que no son de acceso a memoria no necesitan tanto tiempo para completar su ejecución.

En la segunda mitad del capítulo, para solventar la falta de eficiencia, se propone el diseño de un computador multiciclo, en el que, con un tiempo de ciclo mucho menor, cada instrucción tarda varios ciclos en ejecutarse (el estado del computador se modifica en el último ciclo de ejecución de cada instrucción). La eficiencia del nuevo computador viene dada porque ahora no todas tardan el mismo número de ciclos en ejecutarse: las de acceso a memoria tardan más y el resto menos. El número de ciclos que tarda en ejecutarse cada conjunto de instrucciones (lentas o rápidas) se adapta, lo mejor posible, al tiempo de propagación que necesita para su ejecución correcta. El nuevo computador SISC Harvard multiciclo se construye a partir del uniciclo, efectuando unos pequeños cambios en la unidad de control, pero resulta ser bastante más eficiente. Se termina el capítulo evaluando su rendimiento y comparándolo con el uniciclo.

12.2 Completando el diseño del computador uniciclo.

Tal como hemos dejado diseñada la lógica de control en el capítulo anterior (ver figura 12.1), todas las señales de la palabra de control se generan correctamente desde un punto de vista lógico: en cada ciclo, cada señal debe tener el valor lógico que le corresponde para que se ejecute correctamente la instrucción en curso. Esto se ha conseguido mediante el esquema de la figura y escribiendo el contenido de la ROM adecuadamente.

El comportamiento temporal de cada señal de la palabra de control, excepto de las señales de permiso de modificación del estado del computador de las que hablaremos después, al igual que el de cualquier señal de un circuito secuencial con biestables y registros activados por flanco, es el siguiente: la señal tarda un tiempo, desde el flanco ascendente de la señal de reloj que indica el inicio del ciclo actual, en estabilizarse al valor lógico correcto. Durante este tiempo en la señal pueden producirse *glitches* (cambios indeseados de su valor lógico), pero pasado este tiempo, cuando se ha estabilizado, se mantiene estable hasta pasado el ciclo actual (hasta después de que se produzca el flanco ascendente de la señal de reloj que indica el final del ciclo y el inicio del siguiente). Y este comportamiento se repite ciclo a ciclo.

Además, el computador debe cumplir unas restricciones temporales para que funcione correctamente: el tiempo de ciclo debe ser mayor que el tiempo del camino crítico (tiempo de propagación del camino más largo, en tiempo, de la salida Q de un biestable a la entrada D de otro o del mismo biestable. Esta

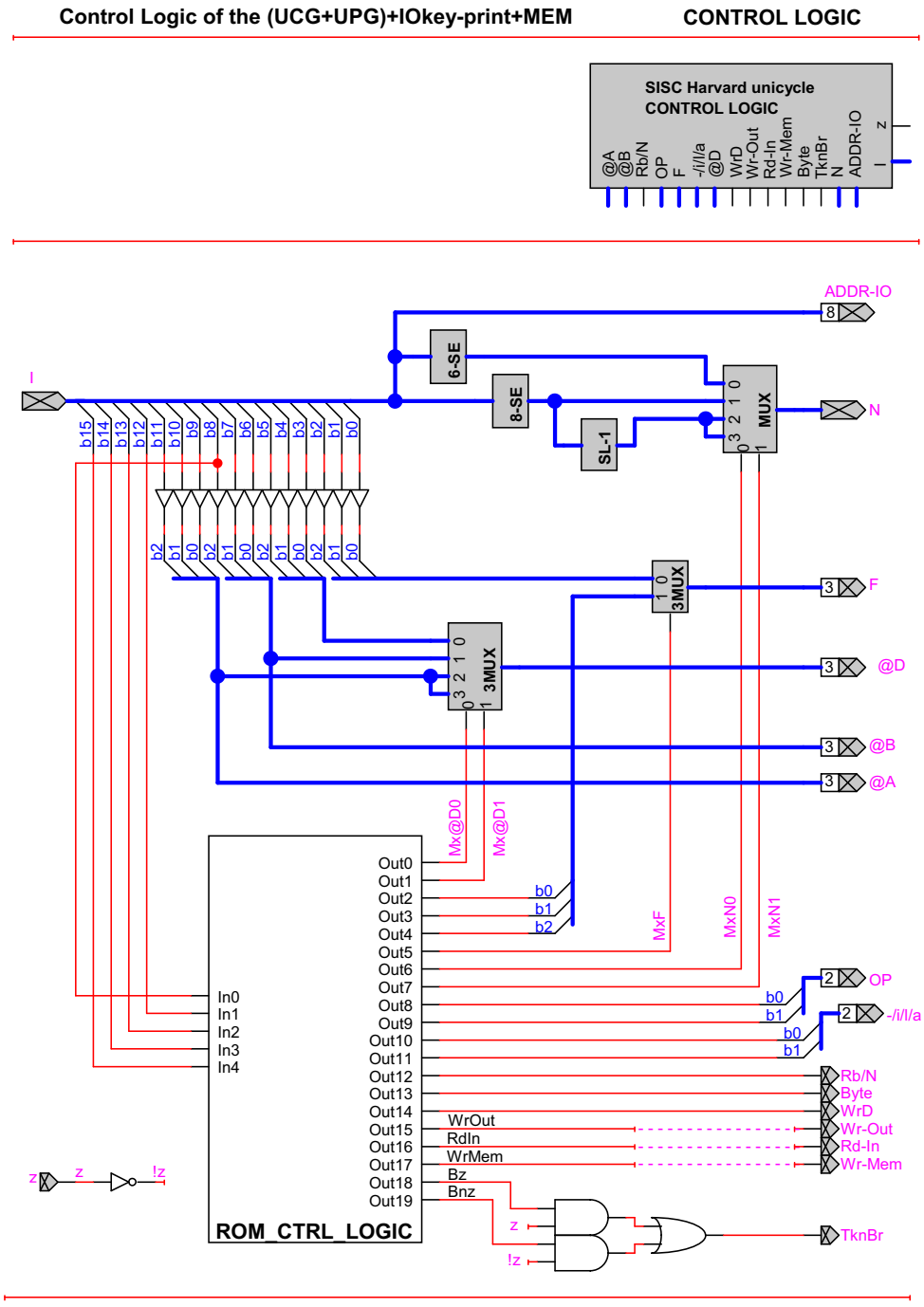

Fig. 12.1 Esquema de la lógica de control a la que le falta por especificar la formación de las señales de permiso de modificación del estado Wr-Mem, Wr-Out *y* Rd-In.

restricción es la misma que tiene cualquier circuito secuencial. En la sección siguiente calculamos el tiempo de ciclo mínimo del computador o el camino crítico, que es lo mismo.

Como se muestra en el cronograma de la figura 12.2, la señal WrD, aunque es una señal de permiso de modificación del estado del computador, no requiere ningún comportamiento temporal diferente del resto de señales que ya genera adecuadamente la lógica de control diseñada en el capítulo anterior. Y esto es así porque es una señal de permiso de escritura de registros activados por flanco: una vez que se estabiliza, dentro del tiempo del ciclo, se mantiene estable hasta después de final de ciclo. No importa que otras señales/buses (como @D o D) se estabilicen antes o después que WrD se estabilice a 1 dentro del ciclo, ni que dejen de estarlo antes o después de que WrD pase a valer cero ya en el ciclo siguiente.

Dicho de otra forma, no es ningún problema que la señal WrD valga 1 en momentos en los que el bus D no tiene el dato correcto, o en momentos en los que el bus @D no tiene la dirección del registro destino correcta.

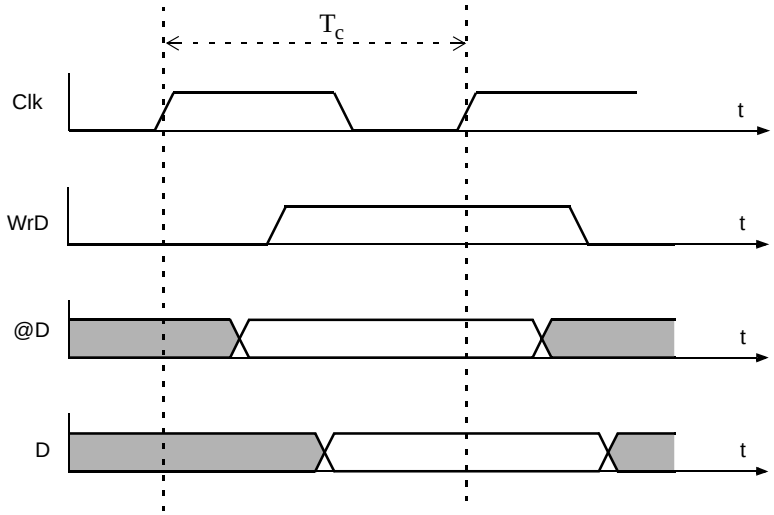

Fig. 12.2 Cronograma de una escritura en un registro del banco de registros

No ocurre lo mismo con el resto de señales de permiso de modificación del estado: escritura en la memoria de datos, Wr-Mem, escritura en un puerto de salida, Wr-Out, y lectura en uno de entrada (que puede poner a cero el puerto de estado de un controlador como efecto lateral de la lectura de su puerto de datos). Estas señales tienen restricciones temporales diferentes a las del resto del computador, y es así porque ni la escritura en memoria ni en un puerto ni tampoco la puesta a cero de un puerto por efecto lateral se realiza sincronizada con el "flanco" ascendente de la señal de reloj del procesador, se realiza sincronizada "por nivel".

Para que la escritura sea correcta, la señal de permiso de escritura (o de modificación del estado, Wr-Mem, Wr-Out o Rd-In) debe pasar a valer 1 al menos T_{su} u.t. (tiempo de *set-up*) después de que la dirección de memoria o del puerto y el dato a escribir permanecen estables en los buses de Memoria o

del controlador de E/S y además debe mantener el valor 1 durante al menos T_{pw} u.t. (tiempo de anchura de pulso, *pulse width*) antes de pasar a valer 0. Además, para que la escritura sea correcta la dirección de memoria o de entrada/salida y el dato a escribir deben seguir estables a sus valores correctos al menos T_h u.t. (tiempo de mantenimiento, *hold*) después de que la señal de permiso de escritura ha pasado a valer 0. Veamos en detalle estas restricciones y cómo hacer que se cumplan.

Wr-Mem

En la figura 12.3 se reproduce el cronograma de escritura de memoria para un funcionamiento correcto, que se explico en el capítulo anterior. La memoria se escribe durante el tiempo en que Wr-Mem vale 1. Para que la escritura se efectúe correctamente la señal Wr-Mem debe pasar a valer 1 después de Tsu u.t. de que queden estabilizada la dirección y el dato a escribir en memoria durante un tiempo de al menos T_{pw} u.t. y además el bus de direcciones de la entrada de la memoria ADDR-MEM y el bus del dato a escribir, WR-MEM deben estar estabilizados con los valores correctos T_{su} u.t. antes de que la señal Wr-Mem pase a valer 1 y mantenerse estables durante el tiempo que Wr-Mem valga 1 y durante el (ya que estamos suponiendo que el tiempo en que deben estar estables después del final de pulso, T_h, es 0).

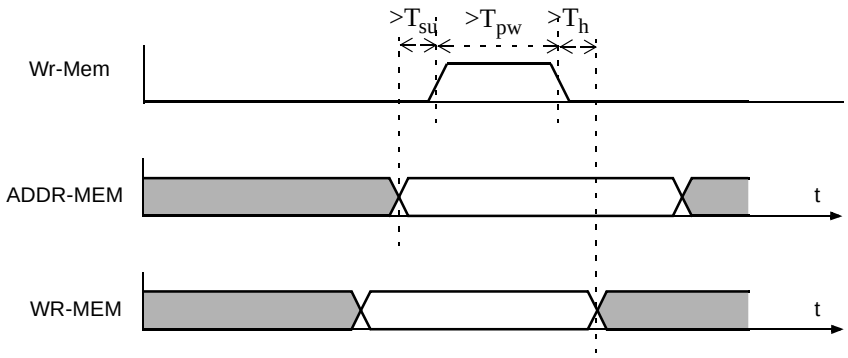

Fig. 12.3 Cronograma de una escritura correcta en memoria.

La figura 12.4 muestra el cronograma de como se estabilizan las señales en un ciclo del procesador en el que se está ejecutando una instrucción ST o STB. Si dejamos el diseño de la lógica de control como lo hicimos en la sección anterior, generando Wr-Mem directamente con la señal WrMem que genera la pequeña ROM de la lógica de control, la escritura podría ser incorrecta, ya que hay momentos en que Wr-Mem (que estamos suponiendo igual a WrMem) vale 1 y los buses de direcciones y datos no tienen el valor correcto ni durante el tiempo que Wr-Mem vale 1 ni durante T_{su} u.t. antes de que Wr-Mem pase a 1. Veamos esto en detalle:

- La señal WrMem y Wr-Mem, que de momento son la misma, pasa a valer 1 (iniciándose el pulso) pasadas **960** u.t. después del flanco ascendente de inicio de ciclo (100 del PC + 800 de I-MEM + 60 de la ROM) y, si al siguiente ciclo se ejecuta una instrucción que no escribe en memoria, pasará a valer 0 (finalizando el pulso) pasadas esas mismas unidades de tiempo después del final de ciclo. Esto es lo que se muestra en el cronograma. No obstante, la situación empeora si la siguiente instrucción es otra vez un ST o un STB, ya que en este caso no pasará a valer 0 y se mantendrá a 1 durante todo este ciclo y parte del siguiente, suponiendo que no haya más de dos instrucciones de escritura en memoria seguidas.

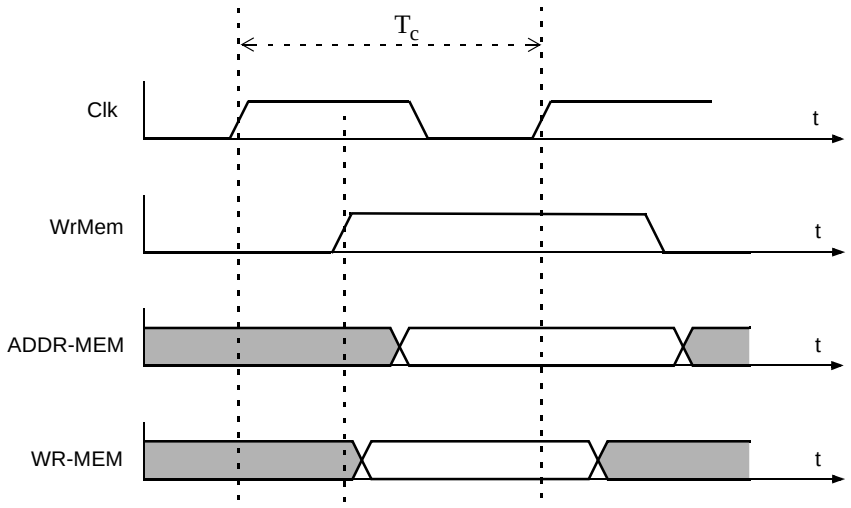

Fig. 12.4 Cronograma de una escritura en memoria que puede producir problemas.

- El bus ADDR-MEM se estabiliza a las **1950** u.t. del inicio de ciclo (los tiempos de cada bloque por el que pasa el camino crítico que estabiliza este bus son: 100 del PC + 800 de I-MEM + 60 de la ROM + 90 del MUX-4-1 entrando por MxN1 + 40 del MUX-2-1 entrando por N + 860 de la ALU para sumar; las 860 u.t. de la ALU para sumar vienen de: 660 del bloque ADD[1] que alimenta a la entrada 4 del MUX-8-1 que selecciona mediante la señal de control F la operación del bloque AL, más 120 u.t. de este MUX-8-1 más 80 u.t. del MUX-4-1 que selecciona mediante la señal OP la operación de la ALU).
- El bus WR-MEM se estabiliza pasadas las **1030** u.t. del inicio de ciclo (los tiempos de cada bloque por el que pasa el camino crítico que estabiliza este bus son:100 del PC + 800 de I-MEM + 0 de @B + 130 del MUX-8-1 de salida del bus B del banco de registros)

Para conseguir una señal Wr-Mem que produzca una escritura correcta aplicamos un truco. La señal de reloj vale 1 durante la primera mitad del ciclo y 0 durante la segunda mitad. Hasta ahora lo único importante respecto a esta señal Clk es el tiempo de ciclo, pero no importa cuando Clk pase de 1 a 0, pues el flanco descendente de reloj no se usa para nada. Vamos a hacer que la segunda mitad del tiempo de ciclo comience donde nos interese para generar una señal Wr-Mem correcta. Vamos a hacer que Wr-Mem sea la salida de una puerta And-2 en la que le entran la señal que genera la ROM, WrMem, y la señal de reloj negada, !Clk. La señal Wr-Mem así creada se muestra el la figura 12.5 y en la figura 12.6 se ve Wr-Mem junto con el resto de señales y buses involucrados en una escritura de memoria.

Así pues, a partir de ahora, a la lógica de control, que es un circuito combinacional, le entrará también la señal de reloj para generar Wr-Mem, definiendo T0, tiempo de la segunda mitad del ciclo, para que cumpla con las restricciones que se indican en el cronograma de la figura 12.6.

1. Observad que el bloque ADD está construido con 16 Full-adders con propagación del acarreo y que el acarreo que entra al Full-adder 0 (el de menor peso, c0) está fijado a cero. Por ello, el acarreo c2 está estable al cabo de 90 u.t. (50 de la Xor-2, 20 de la And-2 y 20 de la Or-2, todas ellas del Full-adder 1). Dicho de otra forma, el camino crítico no pasa por el Full-adder 0.

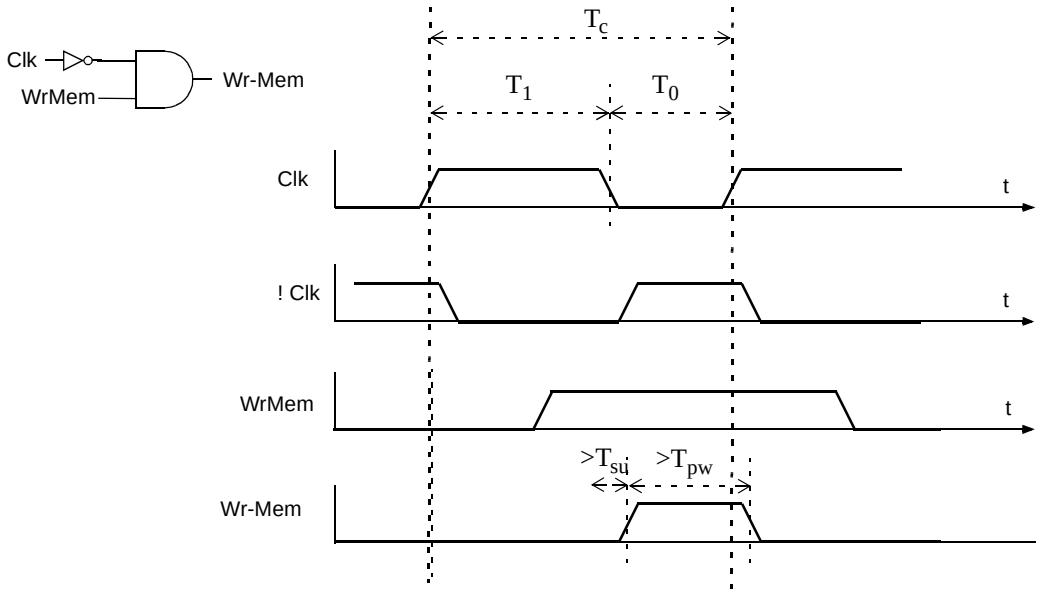

Fig. 12.5 Cronograma de una escritura en la memoria de datos.

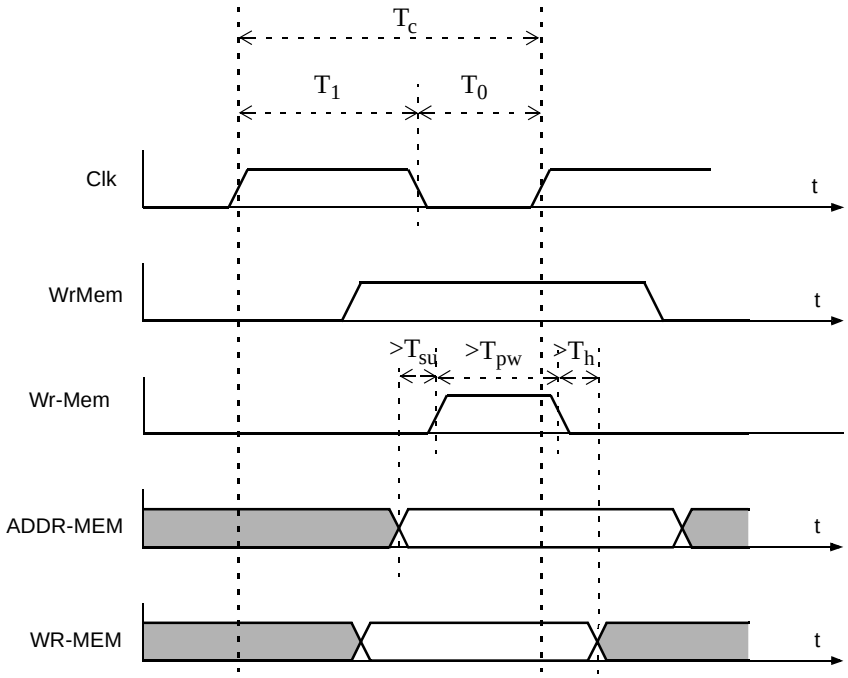

Fig. 12.6 Cronograma de una escritura en la memoria de datos del SISC Harvard Uniciclo.

Wr-Out y Rd-In

Ocurre algo muy parecido a lo analizado con la señal Wr-Mem con las señales Wr-Out y Rd-In: la escritura del puerto de salida o de entrada (por efecto lateral) se realiza "por nivel". Se debe recordar ahora la implementación que hicimos en el capítulo 9 del teclado y la impresora con efecto lateral.

El significado de la señal Wr-Out es el de validar el bus ADDR-IO y el bus WR-OUT. Tiene que pasar a valer 1 al menos T_{su} u.t. después de que estos buses estén estables, debe mantenerse a 1 durante al menos T_{pw} u.t. y estos buses tienen que estar estables al menos T_h u.t. después de que Wr-Out valga 1. Los valores numéricos concretos de T_{su}, T_{pw} y T_h para la escritura correcta de un puerto de salida no los damos, pero suponemos que creando la señal Wr-Out como hemos creado la señal Wr-Mem la escritura del puerto se realiza correctamente:

Wr-Out = !Clk · WrOut

La señal Rd-In valida el bus de direcciones ADDR-IO, cuando se está ejecutando una instrucción `IN`. Este acceso de lectura a un puerto de entrada puede producir un efecto lateral, según de que puerto se trate, que hace que se escriba, que se ponga a cero, otro puerto. Por ello, Rd-In no puede valer 1 mientras ADDR-IO no tenga el valor correcto estable, ya que se podría poner a cero un puerto no deseado. El caso es igual al de Wr-Out y por ello creamos Rd-In de la misma forma:

Rd-In = !Clk · RdIn

El circuito interno de la lógica de control del computador SISC Harvard Uniciclo, completo y definitivo, se muestra en la figura 12.7. En la sección 12.3 calculamos el tiempo de ciclo y elegimos T_1 y T_0 para que con los valores concretos de T_{su}, T_{pw} y T_h de nuestra memoria de datos el computador funcione correctamente a la máxima velocidad (los valores de T_{su}, T_{pw} y T_h para las E/S son mucho menos restrictivos).

12.3 Tiempo de ciclo del computador uniciclo

En el SISC Harvard Uniciclo todas las instrucciones tardan un ciclo en ejecutarse. ¿Cuál es el tiempo de ciclo mínimo? En el momento que llega el flanco ascendente de reloj, indicando que empieza (y termina) un ciclo, se carga el valor del PC con la dirección de la I-MEM que contiene la instrucción que debe ejecutarse en el ciclo que comienza. Las señales que salen del PC deben tener tiempo de leer la instrucción de la memoria de programa, viajar a través de los circuitos combinacionales de la unidad de control y de la unidad de proceso y llegar con tiempo suficiente a los registros, puertos y/o posiciones de memoria que deben escribirse, antes de que llegue el siguiente flanco ascendente de reloj, indicando que termina el ciclo (y empieza otro).

Para encontrar el camino crítico y calcular su tiempo, que será el tiempo de ciclo mínimo después de sumarle un pequeño valor por seguridad, usamos la implementación que hemos hecho del computador y los siguientes tiempos de propagación de los dispositivos:

- Tp(Not) = 10 u.t.
- Tp(And-2) = Tp(Or-2) = 20 u.t.

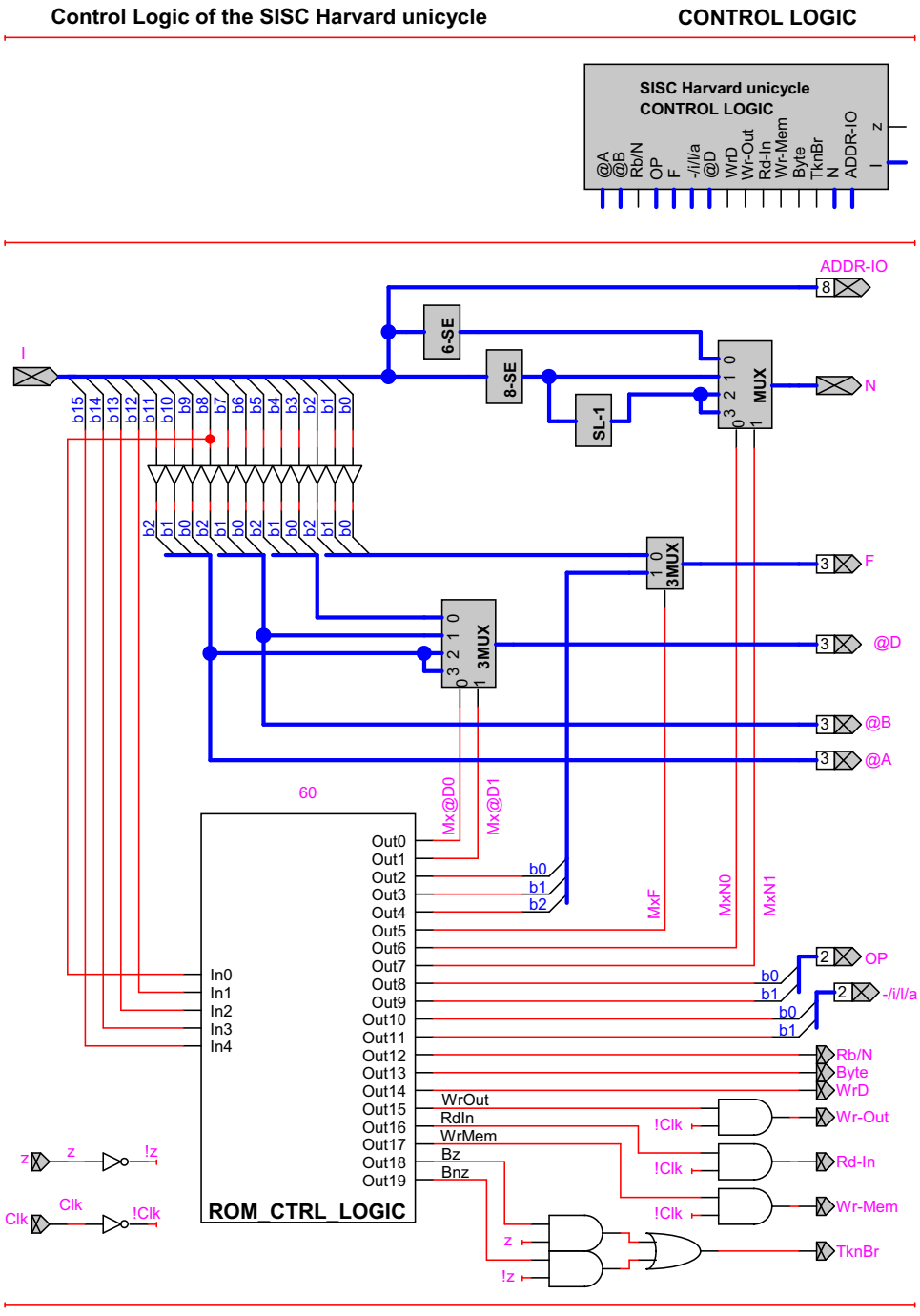

Fig. 12.7 Esquema definitivo de la lógica de control del SISC Harvard Uniciclo.

- Tp(FF) = 100 u.t. (Flip Flop o biestable D activado por flanco)
- Tp(ROM_CTRL_LOGIC) = 60 u.t. (ROM de la lógica de control)
- Tacc(I-MEM) = 800 u.t. (Tiempo de acceso de la memoria de instrucciones: ROM de 64KW)
- Tacc(32KB-RAM) = 800 u.t. (Tiempo de acceso de un módulo de memoria RAM de 32KB)

El camino crítico es el camino más largo atravesado por las señales que salen del PC y llegan a la entrada D de los biestables/registros. Este camino es el de parte de las señales significativas en el ciclo que se ejecuta la instrucción `LDB Rd, 0xN6(Ra)`. A groso modo, cuando se ejecuta una instrucción de lectura de memoria, las señales significativas, útiles para su ejecución, pasan por la I-MEM, por la lógica de control, la ALU para sumar, la memoria para leer el dato cuya dirección se calculo en la ALU y terminan en el banco de registros escribiendo el registro destino. El camino crítico cuando se ejecuta cualquier otra instrucción que no sea de acceso a memoria pasa por todos esos bloques, excepto por la memoria de datos (el camino de las instrucciones `IN` y `OUT` aún pasa por menos bloques, ya que no pasa por la ALU). Así que, cualquier instrucción que no sea de acceso a memoria tiene un camino crítico menor. Y de entre las de acceso a memoria las de lectura tienen un camino con mayor tiempo de propagación, porque además de usar la memoria, tienen que sacar el dato leído de memoria y escribirlo en uno de los registros del banco. De entre las dos instrucciones de *load* la `LDB` tiene un camino crítico un poco más largo (el tiempo de atravesar el MUX-2-1 que selecciona el dato leído de uno de los dos módulos y que la instrucción `LD` no tiene que atravesar).

A continuación indicamos los dispositivos por los que pasa el camino crítico cuando se ejecuta una instrucción `LDB` haciendo referencia al tiempo de propagación de las señales a través de los dispositivos y a las figuras en las que se muestran los bloques por los que pasa el camino crítico, que se ha marcado en oscuro semitransparente sobre las conexiones y bloques por los que pasa. Para cada bloque se ve por qué entrada entra el camino y porqué salida sale, ya que el tiempo de propagación de cada bloque es distinto, en general, para cada pareja entrada-salida. También se indica en las figuras el tiempo en el que se estabilizan cada una de las señales significativas durante la ejecución de la instrucción, desde el inicio del ciclo. La visión general del camino crítico se muestra en la figura 12.8 y el resto de las figuras muestra cómo pasa el camino crítico por el interior de algunos bloques (lógica de control, ALU, AL, 64KB-32KW-MEMORY, banco de registros y registro con señal Ld, de la figura 12.9 a la figura 12.14 respectivamente).

- El registro PC, donde comienza el camino. La salida del PC se estabiliza a las **100** u.t. –unidades de tiempo– que es el tiempo de propagación de un registro, desde que llega el flanco de reloj. Ver figura 12.8.
- La memoria de instrucciones I-MEM, para leer la instrucción a ejecutar (**800** u.t., tiempo de acceso a la I-MEM). La instrucción se encuentra estable a la salida de la memoria a las 900 u.t. desde el inicio del ciclo. Ver figura 12.8.
- El bloque CONTROL LOGIC que genera el bus N de la palabra de control (**60** u.t. de la memoria ROM que genera MxN0 más **90** u.t. de atravesar el MUX-4-1 desde su entrada de selección MxN0 a su salida, ya que está construido con dos niveles de multiplexores 2-1: 50 u.t. desde MxN0 a la salida de los multiplexores de primer nivel y 40 de atravesar el multiplexor de segundo nivel). La figura 12.9 muestra el camino crítico por el interior de la lógica de control e indica el tiempo en que se estabilizan todas las señales significativas en la ejecución de `LDB`. Así la señal N esta estable a las 1050 u.t. desde el inicio del ciclo.

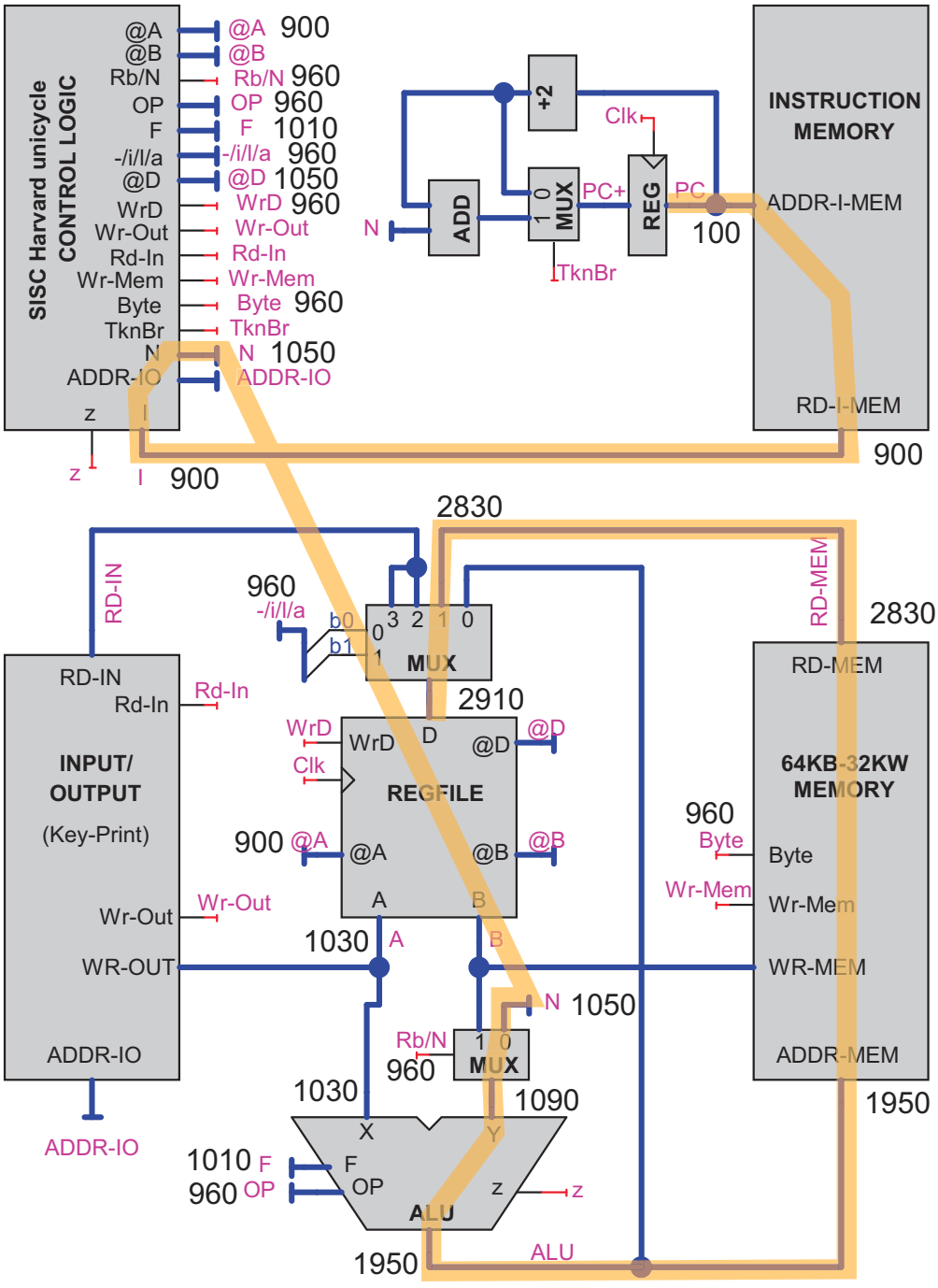

Fig. 12.8 Camino crítico en el computador uniciclo durante la ejecución de una instrucción LDB.

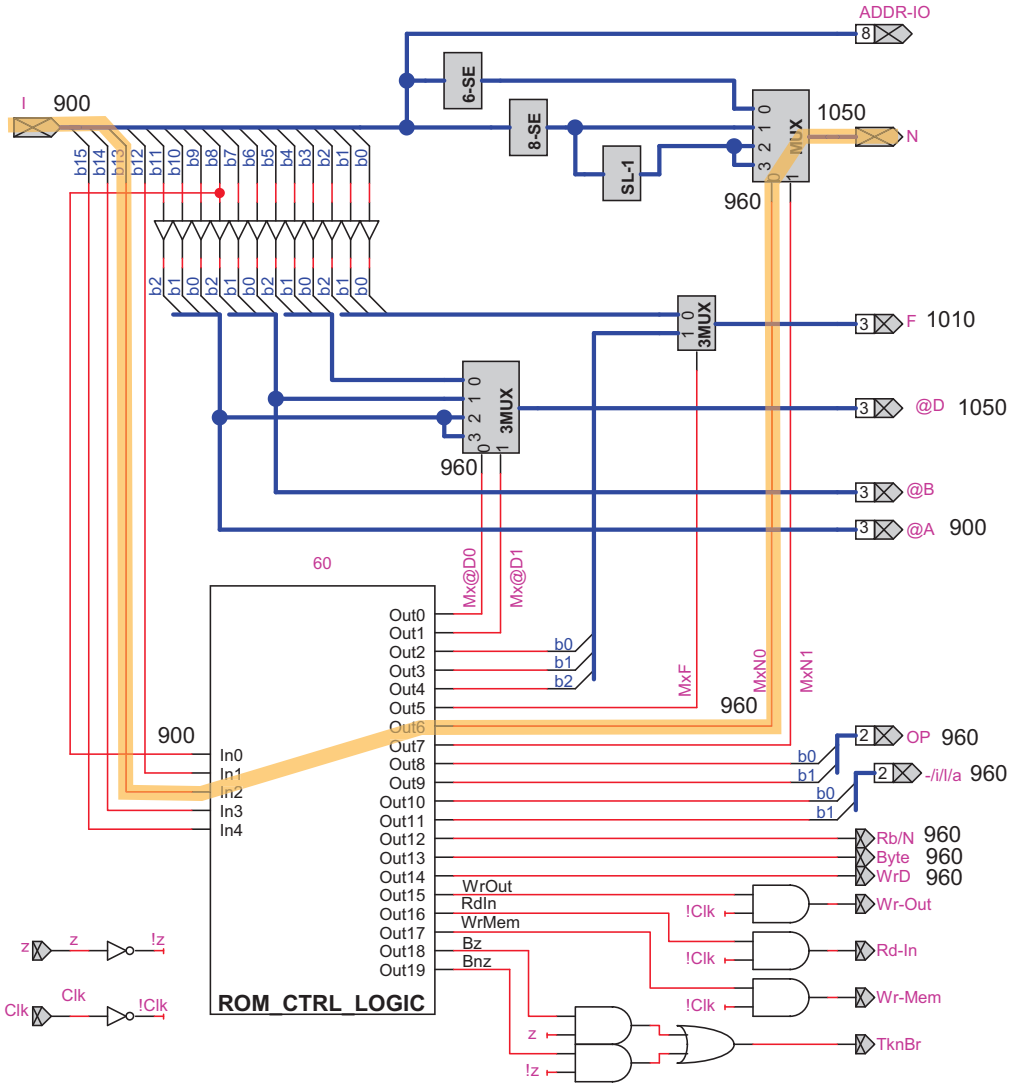

Fig. 12.9 Camino crítico en la lógica de control durante la ejecución de una instrucción LDB.

- El MUX-2-1 controlado por la señal Rb/N que encamina el bus N hacia la entrada Y de la ALU (**40** u.t. de atravesar el multiplexor de buses desde la entrada de datos donde está N). Ver figura 12.8. Por ello, la entrada Y de la ALU está estable a las 1.090 u.t. desde el inicio del ciclo.
- La ALU, para calcular la dirección de memoria sumando el contenido de Ra que ya se encuentra estable en la entrada X de la ALU con el valor N que llega por la entrada Y de la ALU. Ver figura 12.10 y figura 12.11. (**860** u.t: 660 del bloque ADD[1] que alimenta a la entrada 4 del MUX-8-1, que selecciona mediante la señal de control F la operación del bloque AL, más 120 u.t. de este MUX-

8-1 más 80 u.t. del MUX-4-1 que selecciona mediante la señal OP la operación de la ALU). Por ello, la dirección de memoria a la salida de la ALU y a la entrada de la memoria está estable a las 1.950 u.t. desde el inicio del ciclo.

- La memoria de datos, para leer la posición de memoria cuya dirección se calculó en la ALU (**880** u.t: 800 u.t. del tiempo de acceso a los dos módulos de memoria, que se acceden en paralelo, más 40 del MUX-2-1 que selecciona entre la salida de cada módulo, gobernado por el bit 0 de la dirección de memoria, más 40 u.t. del MUX-2-1 que selecciona entre el acceso a word y el acceso a byte controlado por la señal Byte). El bus RD-MEM se estabiliza con el dato leído a las 2.830 u.t. desde el inicio del ciclo. Ver figura 12.12.
- El MUX-4-1 controlado por la señal -/i/l/a, para encaminar el dato leído de memoria hasta el bus de escritura del banco de registros (**80** u.t: 40 de cada uno de los dos niveles de multiplexores). El dato se estabiliza en el bus D del banco de registros a las 2.910 u.t. desde el inicio del ciclo. Ver figura 12.8
- El banco de registros, para escribir el registro indicado por @D. Ver figura 12.13 y figura 12.14. (**40** u.t. del multiplexor interno del registro destino, que es un bloque REGwLd -observad que @D y WrD ya están estables, por lo que la señal Ld del registro destino Rd ya está estable cuando llega el dato a escribirse). Esto hace que a las 2.950 u.t. desde el inicio del ciclo el dato leído de memoria está estable en la entrada D de los biestables que forman el registro destino.

Con estos datos concluimos que el tiempo de ciclo debe ser superior a 2950 u.t. Usaremos un **tiempo de ciclo** de reloj de **3.000 u.t.** ya que es adecuado para el correcto funcionamiento del computador y da

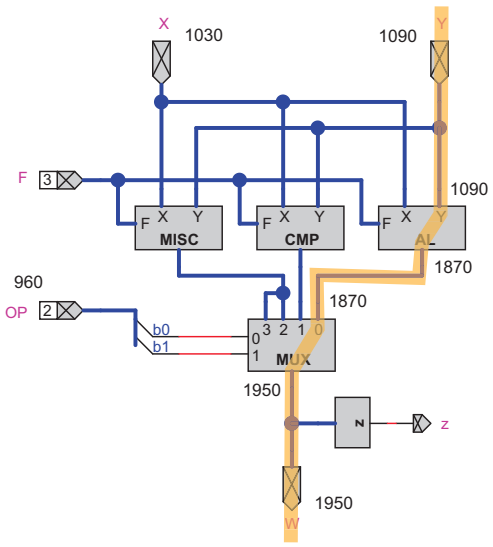

Fig. 12.10 Camino crítico en la ALU durante la ejecución de una instrucción LDB.

1. Observad que el bloque ADD está construido con 16 Full-adders con propagación del acarreo y que el acarreo que entra al Full-adder 0 (el de menor peso, c0) está fijado a cero. Por ello, el acarreo c2 está estable al cabo de 90 u.t. (50 de la Xor-2, 20 de la And-2 y 20 de la Or-2, todas ellas del Full-adder 1). Dicho de otra forma, el camino crítico no pasa por el Full-adder 0.

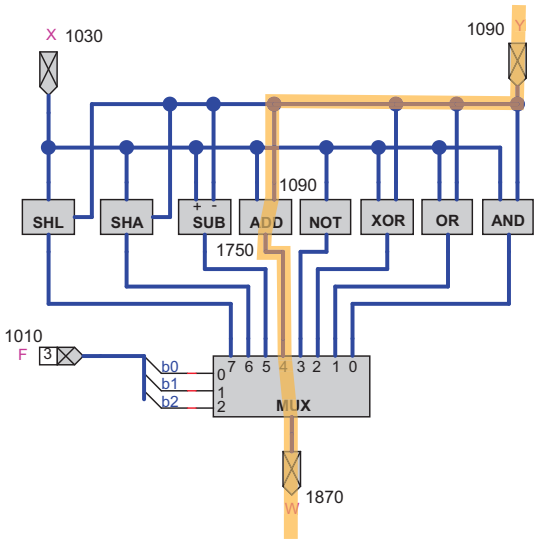

Fig. 12.11 Camino crítico en el bloque AL durante la ejecución de una instrucción LDB.

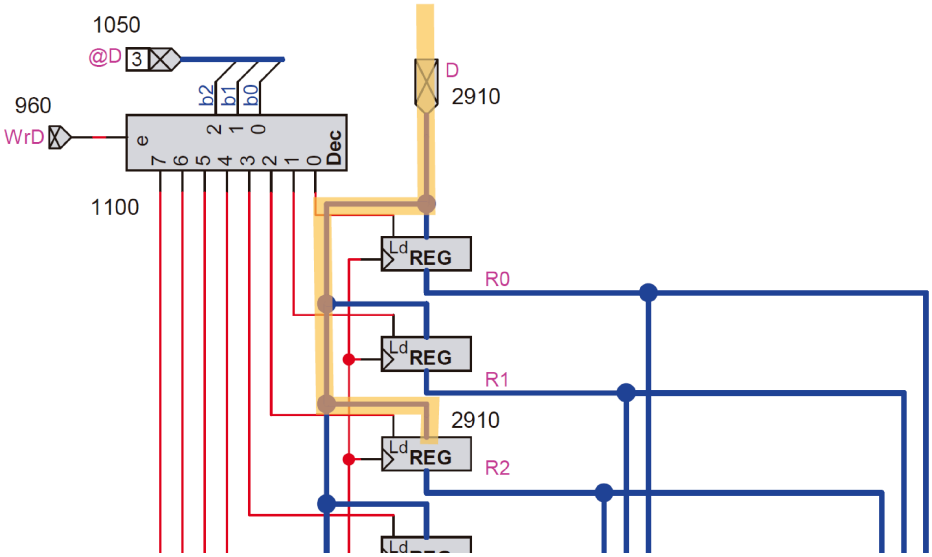

Fig. 12.13 Camino crítico en el banco de registros durante la ejecución de una instrucción LDB.

un pequeño margen de seguridad por si algún dispositivo tarda algo más de lo especificado en su tiempo de propagación.

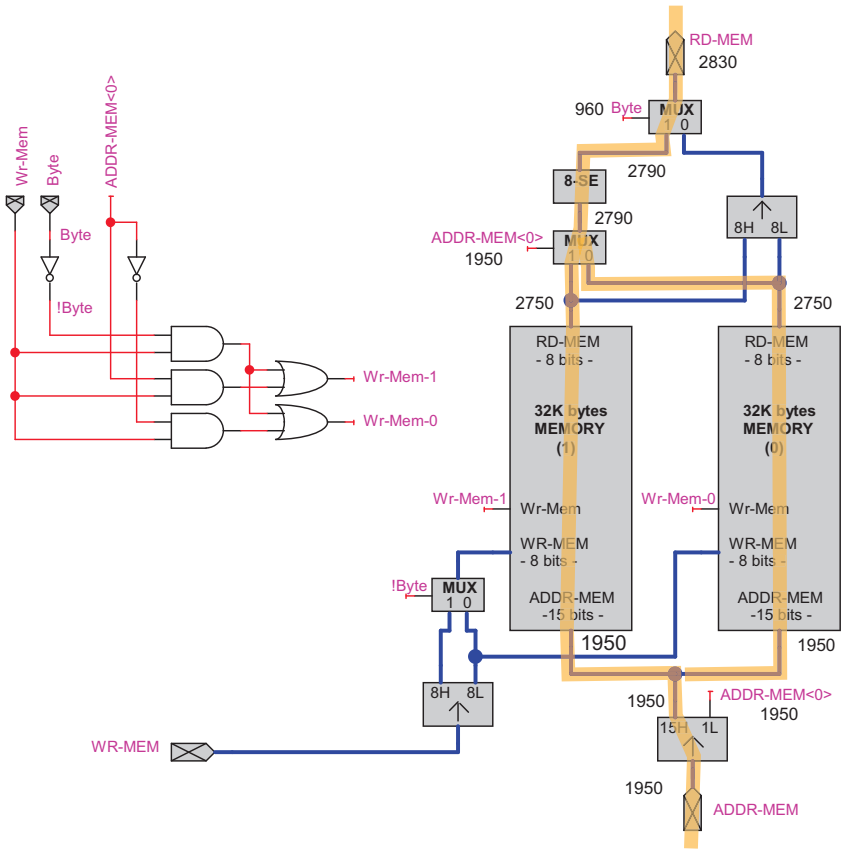

Fig. 12.12 Camino crítico en la memoria de datos durante la ejecución de una instrucción LDB.

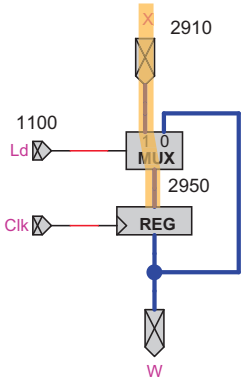

Fig. 12.14 Final del camino crítico en el registro destino del banco de registros durante la ejecución de una instrucción LDB. *El registro es un registro con señal* Ld

¿Funcionaría correctamente el computador con una señal de reloj simétrica, con la primera parte del ciclo de 1.500 u.t. con valor 1 y la segunda parte del ciclo de otras 1.500 u.t. con valor 0? Para contestar a esto debemos observar en la lógica de control (figura 12.7) que la señal de reloj invertida se usa para enmascarar con una puerta And-2 la señal WrMem y generar la señal Wr-Mem. Recordad que esto lo hicimos así para asegurarnos de que la entrada de permiso de escritura de memoria, Wr-Mem, valga 1 solamente en la segunda mitad del ciclo, cuando la dirección de memoria y el dato a escribir están estables a la entrada de la memoria. Observando la implementación del computador que hemos hecho y con los tiempos del camino crítico que acabamos de calcular para la instrucción LDB, podemos afirmar, ya que el cálculo de la dirección de memoria para LDB es igual que para ST y STB, que la dirección de memoria está estable a partir de 1.950 u.t. desde que se inicia el ciclo (2000 u.t. si lo redondeamos añadiéndole un pequeño margen de seguridad). El dato a escribir está estable en el bus de memoria bastante antes, 1.030 u.t. desde el inicio del ciclo (100 del PC más 800 de la memoria de instrucciones más 130 u.t. desde la entrada de selección @B a la salida del MUX-8-1 del bus B del banco de registros). Así pues, como la dirección debe estar estable antes de que llegue el inicio del pulso de permiso de escritura (Wr-Mem), vamos a explorar, en un principio estas dos posibilidades:

- Reloj simétrico con periodo 4.000 u.t. (T_1=2.000, T_0=2.000), o bien
- Reloj asimétrico con periodo 3000 u.t. (2.000, 1.000).

Vamos a profundizar en la segunda opción ya que los programas tardarán menos en ejecutarse que si implementáramos la primera. Suponemos que cada módulo de memoria de 32 KB con que hemos construido la memoria de datos del computador tiene los siguientes parámetros temporales: T_{acc}= 800 u.t., T_{su} = 60 u.t., T_{pw} = 600 u.t y T_h= 40 u.t. A continuación analizamos en detalle el comportamiento temporal del computador para concluir que con un reloj asimétrico de 2.000+1000 u.t., se cumplen sobradamente las restricciones temporales de la escritura de memoria.

En concreto, de los esquemas lógicos del computador se deduce que el pulso de Wr-Mem-0 y Wr-Mem-1 se inicia a las 2.070 u.t. del inicio de ciclo en el que se efectúa la escritura de memoria (a las 2.000 u.t. pasa de 1 a 0 Clk más 70 del camino que va de Clk a Wr-Mem-1 y/o Wr-Mem-0: 10 de la Not para obtener !Clk más 20 de la And-2 para generar Wr-Mem más 40 de la And-2 y Or-2 para generar Wr-Mem-1 y/o Wr-Mem-0), se mantiene a uno durante 1.000 u.t. (el tiempo de la segunda parte del ciclo, en que Clk vale 0) y pasa a valer cero 70 u.t. después de iniciarse el siguiente ciclo (tiempo del camino desde Clk a Wr-Mem1 o Wr-Mem-0).

Por otro lado, la dirección ADDR-MEM<15..1> está estable en el bus de entrada de direcciones de cada módulo de memoria desde 1.950 u.t. del inicio del ciclo y el dato a escribir en el bus de entrada del módulo bastante antes, desde 1.030 u.t. desde el inicio de ciclo. Esto hace que las restricciones temporales de una escritura en memoria se cumplan para los valores de los parámetros. De hecho, sería válida la escritura aunque los parámetros fueran más restrictivos: T_{su} podría llegar a ser hasta de 120 u.t. (120=2.070-1.950) y T_{pw} podría llegar a ser 1000 u.t. No hemos calculado el tiempo mínimo en el que podemos asegurar que ADDR-MEM y WR-DATA están estables desde que termina el ciclo del procesador en el que se ha escrito en memoria, pero seguro que no es menor de 110 u.t. (70 u.t. desde el inicio del siguiente ciclo en las que el pulso de escritura se mantiene a uno más 40 u.t. que requiere el T_h de nuestro módulo de memoria).

Si estudiáramos los requerimientos temporales de las señales Wr-Out y Rd-In veríamos que son menos restrictivas y que las de Wr-Mem y que por lo tanto el tiempo del reloj asimétrico que acabamos de calcular proporciona una ejecución correcta también para las instrucciones `IN` y `OUT`. Para el resto de instrucciones, para las que sólo importa el tiempo de ciclo total, el tiempo de ciclo calculado es más que suficiente, pues para todas ellas las señales significativas en su ejecución se estabilizan mucho antes de que llegue el flanco de fin de ciclo e inicio del siguiente.

Conclusión. Damos por válida la implementación de las señales Wr-Mem, Wr-Out y Rd-In mediante una And-2 con la señal de reloj negada y el uso de un reloj asimétrico con un **Tc = 3.000 u.t. (T1=2.000 y T0=1.000)**.

12.3.1 Tiempo de ejecución de un programa

A modo de ejemplo, veamos el tiempo de ejecución de dos programas en el computador SISC Harvard uniciclo. Los dos programas copian el contenido de los 1.000 (0x03E8) elementos de la tabla (vector), que llamamos A, que se encuentra almacenada en la memoria de datos, en posiciones consecutivas, a partir de la dirección 0x5000 y la copia en la tabla, que llamamos B, que se encuentra a partir de la dirección 0x8000. Cada elemento es un carácter que se almacena en un byte. El primer código es el siguiente.

```
            MOVI   R1, 0x00
            MOVHI  R1, 0x50       ; R1: puntero al primer elemento de A
            MOVI   R2, 0x00
            MOVHI  R2, 0x80       ; R2: puntero al primer elemento de B
            MOVI   R3, 0xE8
            MOVHI  R3, 0x03       ; R3: Número elementos, número de bytes
            ADD    R3, R3, R1     ; R3: punt. al último elem. de A, + 1
Bucle:      LDB    R4, 0(R1)      ; Lectura del elemento de A
            STB    0(R2), R4      ; Escritura en B
            ADDI   R1, R1, 1      ;Incremento puntero a A
            ADDI   R2, R2, 1      ; Incremento puntero a B
            CMPLTU R5, R1, R3     ; ¿Se copió el último elemento?
            BNZ    R5, -6 ; Si no se copio, ir a bucle
```

En este programa se ejecutan 7 instrucciones antes de entrar en el bucle y 6 instrucciones en cada iteración del bucle. Como el bucle se ejecuta `N` veces, el número total de instrucciones ejecutadas es de

$$\text{NumTotalInst} = 7 + 6 \times N \approx 6000 \qquad \textbf{(EQ 1)}$$

Como cada instrucción tarda un ciclo en ejecutarse y el tiempo de ciclo de reloj es de 3000 unidades de tiempo, el tiempo de ejecución del programa es:

$$\text{TiempoTotalEjec} = \text{NumTotalInst} \times 3000 = 18 \times 10^6 \text{u.t.} \qquad \textbf{(EQ 2)}$$

A continuación mostramos el segundo código que usamos como ejemplo. Realiza la misma función que el primero pero con distinto código (el código es correcto porque N es múltiplo de 4, si no fuera así habría que efectuar algún cambio). Este segundo código tiene las 7 primeras instrucciones iguales al primero, por lo que no se muestran.

```
        …
Bucle4: LDB    R4, 0(R1)      ;
        STB    0(R2), R4      ; Primer elemento del bucle
        LDB    R4, 1(R1)      ;
        STB    1(R2), R4      ; Segundo elemento del bucle
        LDB    R4, 2(R1)      ;
        STB    2(R2), R4      ; Tercer elemento del bucle
        LDB    R4, 3(R1)      ;
        STB    3(R2), R4      ; Cuarto elemento del bucle
        ADDI   R1, R1, 4      ; Incremento puntero a A
        ADDI   R2, R2, 4      ; Incremento puntero a B
        CMPLTU R5, R1, R3     ;¿Se movió el último elemento?
        BNZ    R5, -12        ; Si no se copio, ir a bucle4
```

Las medidas para este código son:

$$\text{NumTotalInst} = 7 + 12 \times \frac{N}{4} \approx 3000 \qquad \textbf{(EQ 3)}$$

$$\text{TiempoTotalEjec} = \text{NumTotalInst} \times 3000 \approx 9 \times 10^6 \qquad \textbf{(EQ 4)}$$

Este código se ejecuta el doble de rápido que el primero, en la mitad del tiempo.

Es interesante notar que de las 6 instrucciones ejecutadas dentro del bucle del primer código, sólo dos de ellas hacen directamente lo que queremos hacer en el programa, copiar un elemento de A a B. Las 4 instrucciones restantes se encargan de gestionar esta copia, incrementar los punteros para dejarlos apuntando al siguiente elemento y comprobar si se han movido ya todos los elementos. Sin embargo, el segundo código hace lo mismo pero con menos burocracia. Ahora hacemos en una sola iteración lo que el código anterior hacía en 4 (la profundidad del desenrosque, 4, es arbitraria, aunque múltiplo de N). En esta segunda versión del programa, de las 12 instrucciones de cada iteración hay 8 de movimiento y 4 de gestión. Se ha reducido la sobrecarga de la gestión respecto al trabajo útil. Esta técnica se denomina “desenroscado” de bucle.

12.4 Ventajas de la implementación multiciclo frente a la uniciclo

12.4.1 Ineficiencia del procesador uniciclo

En el computador uniciclo todas las instrucciones tardan un ciclo en ejecutarse y como la señal de reloj es periódica, el tiempo de ciclo es el mismo sea cual sea la instrucción que se ejecute. Para que el computador funcione correctamente se ha ajustado el tiempo de ciclo al tiempo de propagación de la instrucción que más tiempo necesita para estabilizar las señales a la entrada del registro destino, que es la Instrucción LDB. Esta instrucción requiere (ver sección 11.3 del capítulo anterior) algo menos de 3.000 u.t. El testo de instrucciones de acceso a memoria LD, STB y ST requieren un tiempo de ciclo solamente un poco menor que la instrucción LDB.

No obstante, hay instrucciones que, para su ejecución correcta, requieren bastante menos tiempo que las instrucciones de acceso a memoria. Para simplificar, agrupamos el resto de instrucciones bajo el nombre de instrucciones **rápidas**. Esto no quiere decir que todas ellas requieran el mismo tiempo, sino que requieren bastante menos que las de acceso a memoria, que denominaremos **lentas**.

De entre las instrucciones rápidas, las que requieren más tiempo son las de comparación, en concreto la CMPLE, que necesita 2210 u.t., tal como justificamos a continuación.

El camino crítico para una instrucción CMPLE pasa por los siguientes dispositivos (entre paréntesis especificamos el tiempo de propagación de las señales por los dispositivos del camino crítico):

- El registro PC, donde comienza el camino (**100** u.t., que es el tiempo de propagación de un registro, tiempo que tarda en estabilizarse la salida del registro desde que llega el flanco de reloj).
- La memoria de programa, para leer la instrucción a ejecutar (**800** u.t.)
- La Lógica de Control, para generar las señales @A y @B (**0** u.t. ya que se generan directamente y siempre de los mismos campos de la instrucción, cualquiera que sea).
- El banco de registros, para leer el registro Ra por el bus A y Rb por el bus B (**130** u.t. desde la entrada de selección del MUX-8-1 que selecciona el registro que se lee hasta su salida, que es el bus A o B de salida del banco de registros).
- El MUX-2-1 con señal de selección Rb/N que deja pasar el bus B a la entrada Y del la ALU (**40** u.t., ya que la señal de selección de este multiplexor está estable mucho más de 10 u.t. antes que el bus B).
- La ALU, para comparar el contenido de Ra y de Rb que se encuentran en sus entradas X e Y, respectivamente. (**1020** u.t: 940 del bloque CMP más 80 del MUX-4-1 controlado por OP que selecciona entre cuatro diferentes tipos de operaciones de la ALU: Aritmético lógicas, de comparación o miscelánea). Las 940 u.t. del bloque CMP se obtienen de sumar 710 para generar la salida W del bloque SUB más 90 del bloque z (o el camino equivalente de 750 de la señal de overflow v, más 50 de la Xor-2) más 20 de la Or-2 más 120 del Mx-8-1 de salida del bloque CMP. Los 710 del SUB se obtienen de sumar 10 de la puerta Not más 700 del ADD cuando el acarreo de entrada es 1. Estos 700 se obtienen de sumar 90 (del acarreo de salida del Full-adder 0 cuando el acarreo de entrada es 1) más 14 por 40 (14 eses el número de Full-adders intermedios y 40 es el tiempo de propagar el acarreo en cada Full-adder ya que pasa por una puerta And-2 y una Or-2) más 50 de la Xor-2 de salida del bit de más peso del ADD.
- El MUX-4-1 controlado por la señal -/i/l/a, para encaminar el resultado hasta el bus de escritura D del banco de registros (**80** u.t.).
- El banco de registros, para escribir el registro que indica @D (**40** u.t. del multiplexor interno del registro destino, que es un bloque REGwLd, ya que @D y WrD ya están estables, por lo que la señal Ld del registro destino Rd ya es estable cuando llega el dato a escribirse).

Por ello, el tiempo de propagación de una instrucción CMPLE es de **2.210** u.t. Por lo tanto, en el ciclo en que se ejecuta una instrucción rápida el computador no es eficiente: de las 3.000 u.t. que dura el ciclo estará 790 u.t. esperando a que llegue el final de ciclo sin hacer nada útil.

12.4.2 Idea: reloj con taquicardia

Para solucionar esta baja eficiencia se podría construir un reloj que ajustara el tiempo de cada ciclo al tiempo de propagación de la instrucción que se está ejecutando en ese ciclo. Podríamos llamar a esta solución procesador uniciclo con taquicardia (el reloj no late a un ritmo constante, a veces va más rápido y a veces más lento). Pero esta no es la solución que suelen usar los arquitectos de computadores que usan una señal de reloj periódica, de periodo constante: el tiempo de ciclo.

12.4.3 Idea: procesador multiciclo

Hay una solución que con un reloj periódico consigue un rendimiento parecido al procesador uniciclo con taquicardia. Esta es la solución que adoptamos para el procesador multiciclo que vamos a diseñar a continuación.

La idea para reducir el tiempo medio de ejecución de una instrucción consiste en hacer el tiempo de ciclo más pequeño y descomponer la ejecución de una instrucción en varios ciclos. Ahora las instrucciones lentas, las que tardan **más tiempo** en estabilizar sus señales en el computador uniciclo, tardarán **más ciclos** en ejecutarse que las instrucciones rápidas, que tardarán menos ciclos.

12.5 Computador Harvard multiciclo

A la vista de los tiempos del camino crítico de las instrucciones lentas (2.950 u.t. según lo calculado en la sección 12.3) y rápidas (2.210 u.t. según acabamos de calcular) tomamos la decisión de usar una señal de reloj con tiempo de ciclo de 750 u.t. de forma que las instrucciones lentas tarden cuatro ciclos en ejecutarse (tardan 4*750 = 3000 u.t. que es más que su tiempo del camino crítico, 2.950) mientras que las rápidas (el resto) lo hagan en tres ciclos (tardan 3*750 = 2.250 u.t. que es más que el tiempo de su camino crítico, 2.210). Por lo tanto el computador funcionará perfectamente y será bastante más eficiente. Otra decisión es que vamos a diseñar el nuevo computador multiciclo efectuando los mínimos cambios posibles a partir del diseño uniciclo. Este computador lo llamamos SISC Harvard multiciclo ya que ejecuta el mismo juego de instrucciones SISA que el uniciclo, por eso es SISC, y mantiene la estructura Harvard (tiene separadas las memorias de instrucciones y de datos).

12.5.1 Unidad de proceso

La unidad de proceso del multiciclo es casi idéntica a la del uniciclo. En el uniciclo el registro contador de programa, PC, se cargaba con la dirección de la siguiente instrucción a ejecutar al final de todos los ciclos, por lo que se implementaba con un dispositivo REG. Ahora habrá que implementarlo con un registro con señal de carga (REGwLd) ya que el PC no se tiene que cargar en todos los ciclos. Se tiene que cargar en el tercer y último ciclo de ejecución de las instrucciones rápidas y en el cuarto y último ciclo de las lentas. Por ello, en el multiciclo la palabra de control tiene un bit más que en el uniciclo, el bit LdPc, que controla la carga del PC.

La figura 12.15 muestra el computador SISC Harvard multiciclo. Los subsistemas de memoria y de entrada/salida son los mismos que los el computador uniciclo. Las diferencias están en el procesador, y en concreto en la unidad de control.

12.5.2 Unidad de control

El objetivo de diseño que seguimos para la unidad de control es que se parezca lo más posible a la del computador uniciclo. Tanto el computador uniciclo como en el multiciclo, en el instante en que se carga el PC y comienza la ejecución de una nueva instrucción (y un nuevo ciclo), las señales fluyen desde el PC a la memoria de instrucciones, se obtiene la instrucción a ejecutar, pasan por la unidad de control, se aplican las señales de la palabra de control a la unidad de proceso, memoria y entradas/

salidas y finalmente se modifica el estado del computador con el resultado de la instrucción (incluyendo la carga del PC) y se inicia la ejecución de la nueva instrucción. La diferencia es que en el uniciclo, desde que se inicia la ejecución de una instrucción hasta que se modifica el estado del computador pasa siempre un único ciclo, mientras que en el multiciclo, pasan tres o cuatro ciclos.

El control de la ejecución de una instrucción se realiza en el bloque SISC Harvard Multicycle CONTROL, que genera la palabra de control de forma parecida a como lo hacía la lógica de control del Harvard Unicycle. Ahora, diferenciamos dos tipos de señales de la palabra de control:

- Las señales de permiso de modificación del estado,
  - WrD para escribir el registro destino del banco de registros,
  - Wr-Out para escribir un puerto de salida de datos,
  - Rd-In para escribir el registro de estado de un periférico al leer su registro de datos,

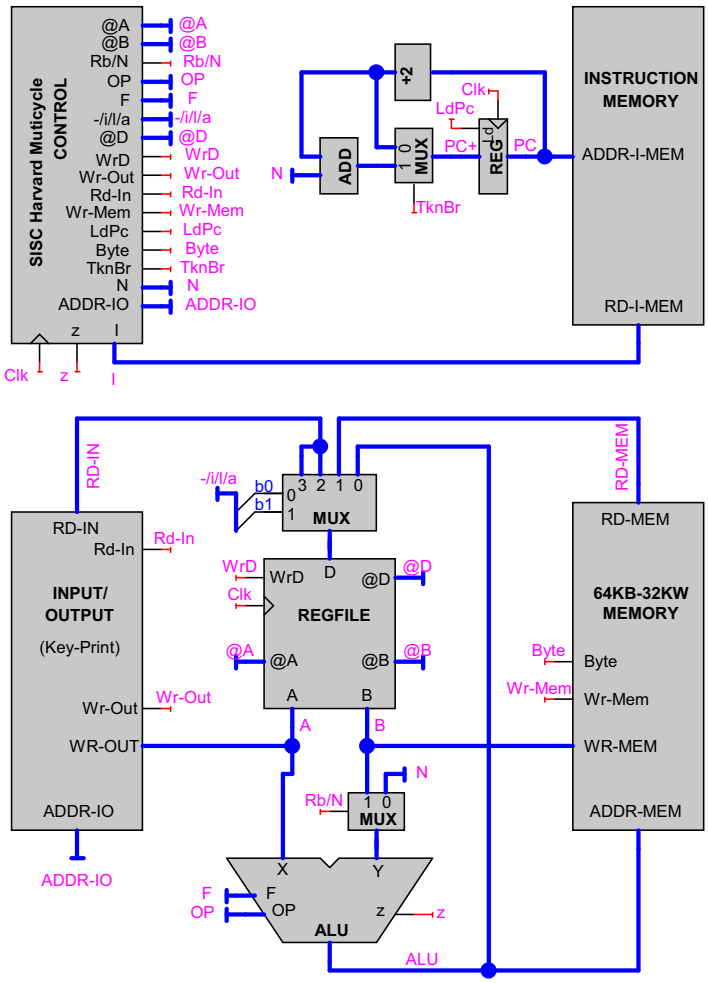

Fig. 12.15 Esquema del computador SISC Harvard Multicycle.

- Wr-Mem para escribir la memoria de datos, y
- LdPc para escribir el PC, en el último ciclo de ejecución de cada instrucción, con la dirección de la siguiente instrucción a ejecutar.

De entre estas señales, las que deben activarse (lo que depende del tipo de instrucción que se esté ejecutando) deben hacerlo en el último ciclo de ejecución de la instrucción, el tercero o el cuarto. Si se activaran antes se modificaría el estado del computador erróneamente.

- El resto de señales de la palabra de control, las que controlan a los multiplexores para encaminar los datos por la UPG, controlan la funcionalidad de la ALU y codifican la dirección de los registros del REGFILE y de los puertos de entrada y salida.

Diferenciamos dos partes en el circuito interno del SISC Harvard Multicycle CONTROL (ver figura 12.16):

- El bloque SISC Harvard Multicycle CONTROL LOGIC, que genera las mismas señales y de la misma forma que el SISC Harvard Unicycle CONTROL LOGIC excepto:
  - La nueva señal, Mem. Esta señal valdrá 1 cuando se ejecute una instrucción lenta (LD, LDB, ST o STB) y 0 para el resto de instrucciones.
  - Las señales de permiso de modificación del estado WrD, Wr-Mem, Wr-Out y Rd-In que generaba el bloque uniciclo ahora el multiciclo las genera sin guion, WrMem, WrOut y RdIn, y en vez de WrD genera WrD1. Las señales sin guion las genera directamente como salen de la ROM interna del bloque, sin que atraviesen una puerta And-2 con el reloj negado. La funcionalidad de generar el pulso de permiso de escritura adecuadamente lo hace ahora el bloque SISC Harvard Multicycle CONTROL a partir de estas señales.
- La parte encargada de generar las señales de permiso de modificación del estado del computador formada por dos circuitos:
  - El circuito secuencial encargado de generar la señal LdPc a partir de la señal Mem que indica si se está ejecutando una instrucción rápida (Mem = 0) o lenta (Mem = 1). Se ve en la parte superior del esquema del circuito de la figura 12.16, implementado con una ROM para el estado siguiente y una And-2 para la salida, LdPc (que vale 1 en el tercer ciclo de ejecución de la instrucción, si se está ejecutando una instrucción rápida, o en el cuarto, si se ejecuta una lenta. El grafo de estados de este circuito se muestra en la figura 12.17 y es muy sencillo: la señal LdPc vale 1 solamente en el último ciclo/estado de ejecución de cada instrucción, las instrucciones rápidas llegan a este estado al tercer ciclo de ejecución y las lentas al cuatro.
  - El circuito encargado de generar las señales de la palabra de control de permiso de modificación del estado (WrD, Wr-Out, Rd-In, y Wr-Mem) en función de las señales equivalentes que genera el bloque SISC Harvard Unicycle CONTROL LOGIC (WrD1, WrOut, RdIn, y WrMem), que se estabilizan antes del tercer ciclo, y de la señal LdPc, que se activa solamente en el último ciclo de ejecución de la instrucción. Este usa cuatro puertas And-2 para que las señales de modificación del estado se activen, si tienen que hacerlo, en el último ciclo de ejecución de la instrucción:
    WrD = WrD1 · LdPc
    Wr-Out = WrOut · LdPc
    Rd-In = RdIn · LdPc
    Wr-Mem = WrMem · LdPc

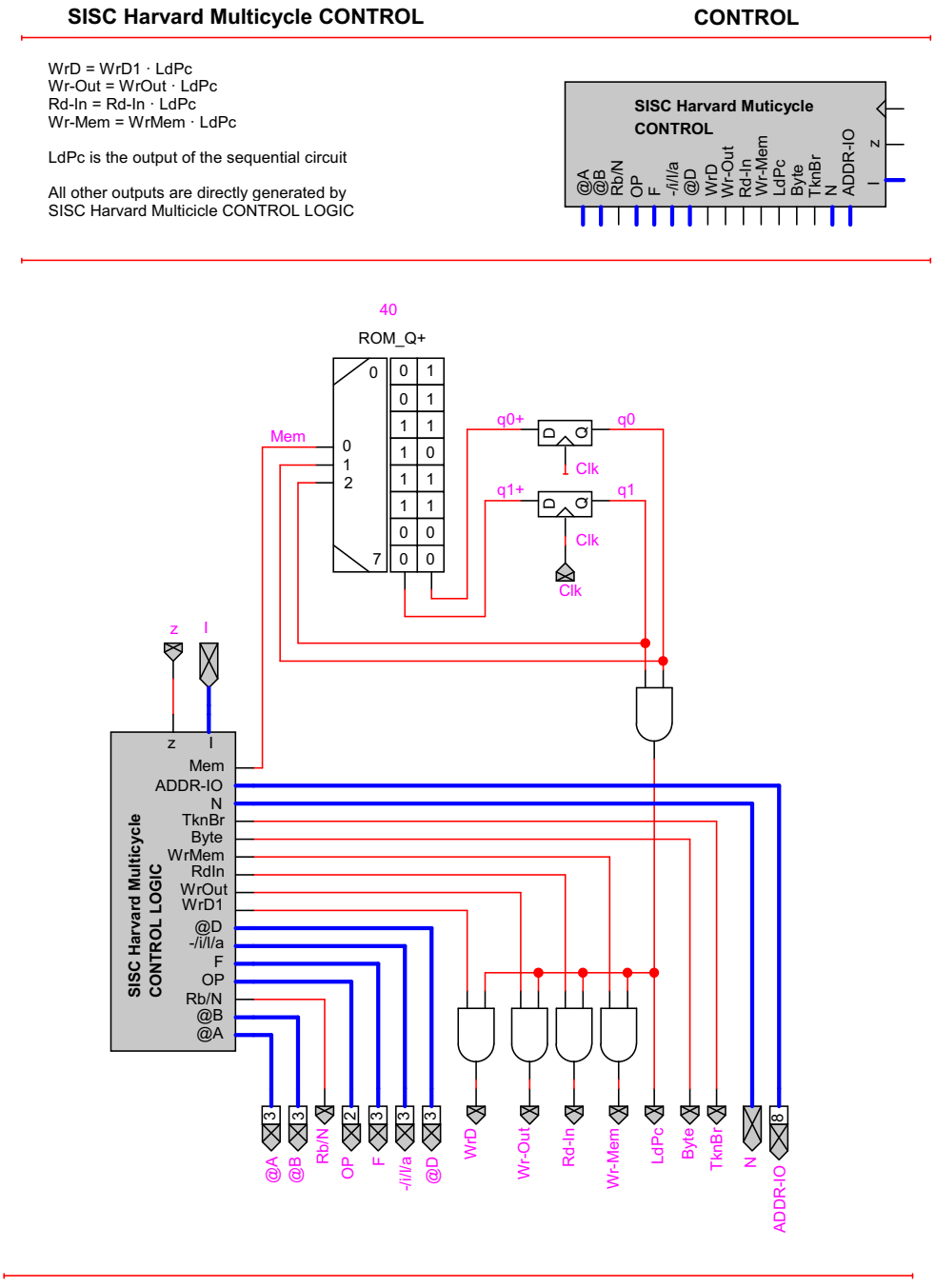

Fig. 12.16 Esquema lógico interno del bloque secuencial SISC Harvard Multicycle CONTROL.

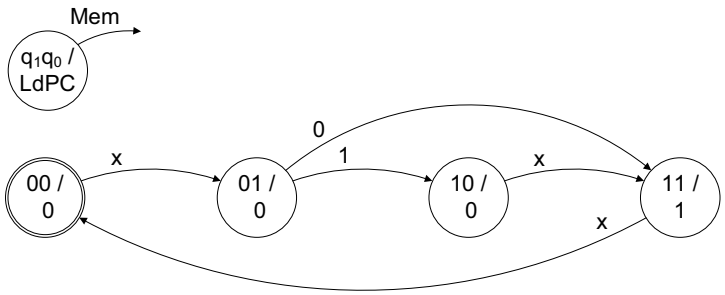

Fig. 12.17 Grafo de estados del secuencial que calcula LdPc *dentro en el bloque* CONTROL.

La implementación interna del bloque SISC Harvard Multicycle CONTROL LOGIC se muestra en la figura 12.18 y el contenido de la ROM que utiliza se especifica en la figura 12.19 mediante una tabla compactada. Observando el contenido de la ROM se ve que es idéntico que el de la ROM de la lógica de control del uniciclo que especificamos en el capítulo anterior, excepto para la nueva señal Mem (considerando el cambio de nombre de WrD a WrD1). El contenido de esta ROM se ha preparado para pasarlo a notación en hexadecimal, separando cada cuatro bits, empezando por la izquierda, y haciendo que las x valgan 0. En la figura 12.20 se muestra el contenido de la ROM en notación LogicWorks.

Ejercicio 1

Este es un tema importante que os dejamos como ejercicio. Justificad numéricamente, analizando los tiempos de propagación de los dispositivos, por qué en el computador multiciclo las señales Wr-Mem y Wr-Out y Rd-In que genera el bloque SISC Harvard Multicycle CONTROL no necesitan pasar por ninguna puerta And-2 controlada por la señal de reloj invertida. Recordad que estas puertas son necesarias en el computador uniciclo para que se activen las señales de escritura de memoria y de los puertos solamente cuando la dirección de memoria o del puerto y el dato a escribir están estables en el bus de direcciones y de datos correspondiente.

```
005000 005000 005100 005100 005031 005031 104431 104431

120030 120030 106431 106431 122030 122030 000000 000000

0402A0 0802A0 004266 00426A 014802 000000 000000 000000

000000 000000 000000 000000 000000 000000 000000 000000
```

Fig. 12.20 Contenido de la ROM del bloque SISC Harvard Multicycle CONTROL LOGIC *en formato LogicWorks.*

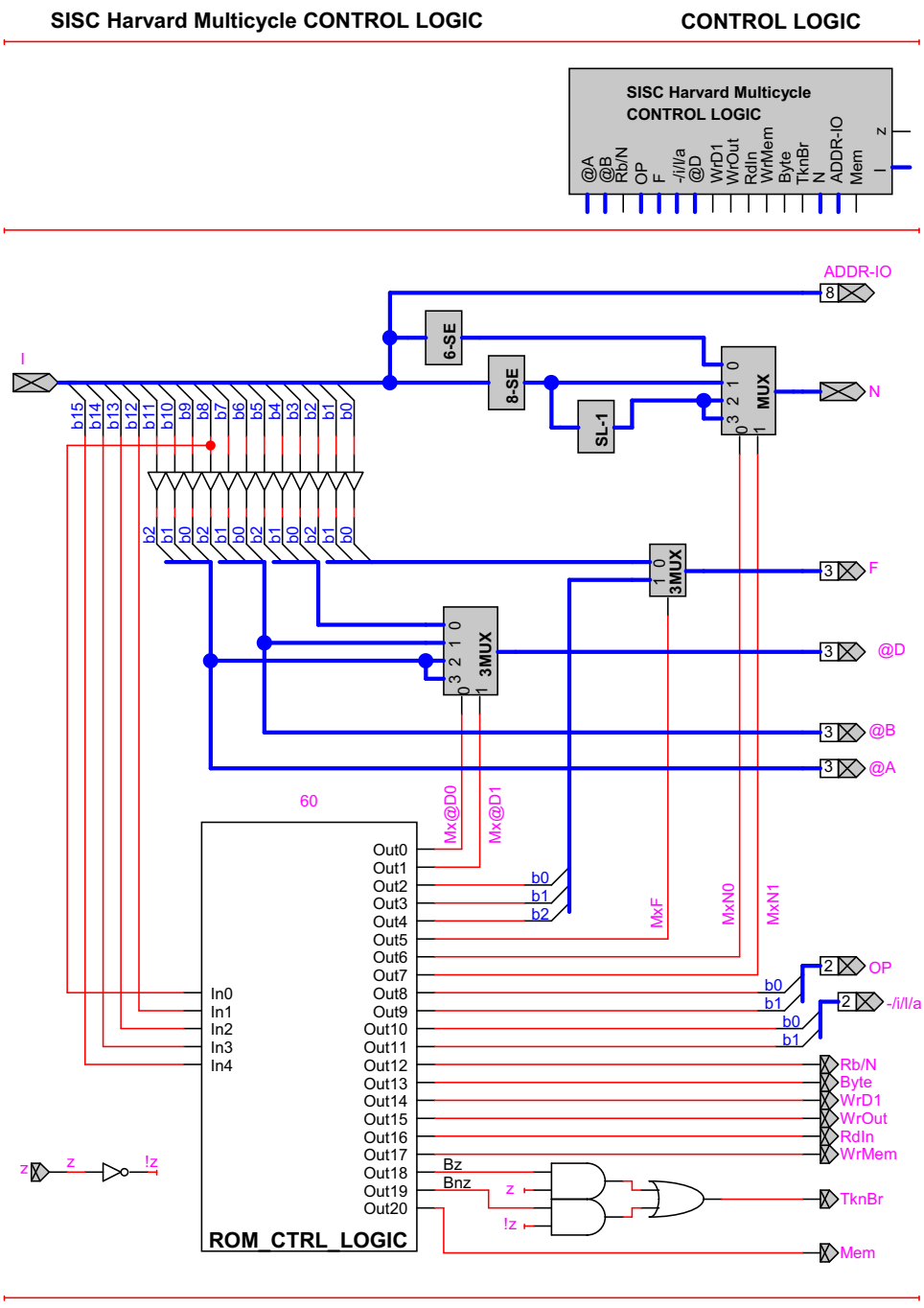

Fig. 12.18 Esquema del bloque combinacional SISC Harvard Multicycle CONTROL LOGIC.

Dirección					Contenido																					
In4	In3	In2	In1	In0	Out20	Out19	Out18	Out17	Out16	Out15	Out14	Out13	Out12	Out11	Out10	Out9	Out8	Out7	Out6	Out5	Out4	Out3	Out2	Out1	Out0	
I<15>	**I<14>**	**I<13>**	**I<12>**	**I<8>**	Mem	Bnz	Bz	Wr-Mem	RdIn	WrOut	WrD1	Byte	Rb/N	-/i/I/a1	-/i/I/a0	OP1	OP0	MxN1	MxN0	MxF	F2	F1	F0	Mx@D1	Mx@D0	
0	0	0	0	x	0	0	0	0	0	0	1	x	1	0	0	0	0	x	x	0	x	x	x	0	0	AL
0	0	0	1	x	0	0	0	0	0	0	1	x	1	0	0	0	1	x	x	0	x	x	x	0	0	CMP
0	0	1	0	x	0	0	0	0	0	0	1	x	0	0	0	0	0	0	0	1	1	0	0	0	1	ADDI
0	0	1	1	x	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	1	1	0	0	0	1	LD
0	1	0	0	x	1	0	0	1	0	0	0	0	0	x	x	0	0	0	0	1	1	0	0	x	x	ST
0	1	0	1	x	1	0	0	0	0	0	1	1	0	0	1	0	0	0	0	1	1	0	0	0	1	LDB
0	1	1	0	x	1	0	0	1	0	0	0	1	0	x	x	0	0	0	0	1	1	0	0	x	x	STB
0	1	1	1	x	0	0	0	0	0	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	(NOP)
1	0	0	0	0	0	0	1	0	0	0	0	x	x	x	x	1	0	1	0	1	0	0	0	x	x	BZ
1	0	0	0	1	0	1	0	0	0	0	0	x	x	x	x	1	0	1	0	1	0	0	0	x	x	BNZ
1	0	0	1	0	0	0	0	0	0	0	1	x	0	0	0	1	0	0	1	1	0	0	1	1	0	MOVI
1	0	0	1	1	0	0	0	0	0	0	1	x	0	0	0	1	0	0	1	1	0	1	0	1	0	MOVHI
1	0	1	0	0	0	0	0	0	1	0	1	x	x	1	0	x	x	x	x	x	x	x	x	1	0	IN
1	0	1	0	1	0	0	0	0	0	1	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	OUT
1	0	1	1	x	0	0	0	0	0	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	(NOP)
1	1	x	x	x	0	0	0	0	0	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	(NOP)

Fig. 12.19 Tabla de verdad compacta del contenido de la ROM-CTRL-LOGIC de la lógica de control.

12.6 Evaluación del rendimiento

12.6.1 Tiempo de ejecución de un programa

Ahora, en el computador multiciclo, el tiempo de ejecución de una instrucción varía según el tipo de instrucción, rápida o lenta:

- La instrucciones rápidas tardan $3 \times 750 = 2250$ u.t., que es menos que las 3000 u.t. que tardan en el uniciclo.
- Las instrucciones lentas, `LD`, `LSB`, `LS` y `STB`, tardan $4 \times 750 = 3000$ u.t., que es lo mismo que tardan en el uniciclo.

Por ello, cualquier programa se ejecutará más rápido en el nuevo procesador que en el anterior. No obstante, si queremos saber cuánto más rápido se ejecuta debemos saber el porcentaje de instrucciones de cada tipo (acceso a memoria frente al resto) que se ejecutan al ejecutar el programa.

Para el primer programa que hemos escrito en la sección 12.3.1, que mueve el contenido de una tabla y que en el cuerpo del bucle tiene 2 instrucciones lentas frente a 4 instrucciones rápidas, el tiempo de ejecución en el computador multiciclo es, aproximadamente, de (para N = 1000):

$$\text{TiempoTotalEjec} = (2 \times 4 + 4 \times 3) \times 750 \times N = 15 \times 10^6 \text{ u.t.}$$

Este tiempo es menor que en el uniciclo, que es de 18×10^6 u.t.

En el programa optimizado que presentamos también en la misma sección del capítulo anterior, se desenrosca el bucle para tener menos instrucciones de control, consiguiéndose que en el cuerpo del nuevo bucle se ejecuten 8 instrucciones lentas y 4 rápidas y que el bucle se ejecute N/4 veces. El tiempo de ejecución de este programa optimizado en el computador multiciclo es:

$$\text{TiempoTotalEjec} = (8 \times 4 + 4 \times 3) \times 750 \times \frac{N}{4} = 8,25 \times 10^6 \text{ u.t.}$$

Este tiempo es menor al tiempo de ejecución en el uniciclo, que es de 9×10^6 u.t.

12.6.2 Tiempo medio por instrucción en el computador multiciclo frente al uniciclo

Supongamos que una distribución típica de tipos de instrucciones ejecutadas para un conjunto de programas compilados al lenguaje máquina SISA es:

- 30% de instrucciones de acceso a memoria y
- 70% del resto de instrucciones.

Con estos porcentajes de tipos de instrucciones ejecutadas, el **tiempo medio por instrucción en el multiciclo es de** $(0.3 \times 4 + 0.7 \times 3)750 = 2475$ u.t. **frente a** 3000 u.t. **en el uniciclo.**

Esto es, ejecutando programas con esta distribución de instrucciones **el multiciclo es un 21% más rápido que el uniciclo, aproximadamente** ($21 \approx \left(\frac{3000}{2475} - 1\right) \times 100$).

Ejercicio 2

Para finalizar os proponemos el siguiente ejercicio que es muy completo, aunque su realización no es trivial. Se trata de repetir prácticamente todos los pasos que aquí hemos seguido para implementar y evaluar el computador multiciclo pero partiendo de otras decisiones de diseño. Hay muchas soluciones válidas, pero unas conseguirán mejores resultados que otras. No se trata de encontrar la más eficiente, pues ese es un problema muy complejo, sino de hacer propuestas razonables.

Diseñad el bloque de control (grafo de estados e implementación con el número mínimo de biestables y dos pequeñas ROMs más el resto de lógica que puede incluir una modificación (o no) del bloque de la lógica de control) de un computador multiciclo con la unidad de proceso del multiciclo en el que tanto el tiempo de acceso a la memoria de instrucciones como a un módulo de la de datos es el doble del tiempo que en el computador que hemos diseñado (1.600 u.t. = 800x2). Tenéis que decidir el tiempo de ciclo de reloj y el número de ciclos en que se ejecuta cada tipo de instrucción (no tenéis porque hacer una clasificación tan simple como hemos hecho antes que hemos considerado solamente instrucciones lentas y rápidas) para que se consiga la máxima velocidad en la ejecución de los programas.

Por último, calculad el tiempo de ejecución medio por instrucción para esta nueva implementación multiciclo cuando ejecuta programas con la siguiente distribución de tipo de instrucciones ejecutadas:

AL:	20%	LD y LDB:	20%
CMP:	10%	ST y STB:	10%
MOVI y MOVHI:	5%	BZ y BNZ:	15%
IN:	10%	ADDI:	5%
OUT:	5%		