9 Entrada/salida

9.1 Introducción

Seguimos en el camino que nos lleva, paso a paso, desde un diseño específico para resolver cada problema hasta el computador, que nos servirá para resolver cualquier problema. En este capítulo vamos a seguir diseñando procesadores de propósito específico (PPE), como en el anterior, pero ahora los vamos a conectar a dispositivos **periféricos de entrada/salida** (teclado, impresora, discos,...) como los que suele tener conectados el procesador de un computador. En concreto, los PPE de este capítulo serán como los del anterior en cuanto a que estarán formados por una unidad de control específica (UCE) y la unidad de proceso general (UPG), pero a diferencia de ellos ahora:

- Podrán entrar datos al PPE de distintos periféricos (en el capítulo anterior el bus RD-IN estaba conectado a un único circuito lógico secuencial que lo alimentaba). También podrán salir datos del PPE a diferentes periféricos (antes el bus WR-OUT estaba siempre conectado a un único circuito que recogía los resultados). Esto se trata en la sección 9.2 y 9.3.
- La comunicación entre el PPE y los periféricos de entrada/salida se realizará mediante un protocolo de comunicación **asíncrono** (antes el circuito que alimentaba con datos al PPE y el que recibía sus resultados tenían la misma señal de reloj que el PPE y por eso se podía usar un protocolo de comunicación síncrono, que hablaba de ciclos). Es necesario un protocolo asíncrono porque los periféricos están a relativamente alejados del procesador lo que hace inviable que compartan la misma señal de reloj que tiene el procesador, ya que suele ser de elevada frecuencia. Además, la variada velocidad de operación de los distintos periféricos nada tiene que ver con la del procesador. Esto se trata en la sección 9.4 y 9.5

El sistema de entrada/salida que terminamos diseñando en este capítulo está formado por un teclado y una impresora (sección 9.6) será usado en los diseños de PPE que hacemos en la sección 9.7. Este sistema es exactamente el mismo que conectaremos, en los capítulos siguientes, a los procesadores de propósito general que usaremos en el diseño final de nuestro computador.

9.2 Añadiendo un espacio de direcciones de entrada y otro de salida

Vamos a tener distintos periféricos de entrada/salida (teclado, ratón, pantalla, impresora, discos, etc.). El procesador debe ser capaz de leer (*input*, entrar) datos de estos periféricos y de escribir (*output*, sacar) datos hacia ellos.

9.2.1 Controladores y puertos de entrada/salida

Lo único que ve el procesador respecto de un periférico de entrada/salida es un conjunto de registros que el procesador debe ser capaz de leer y escribir. Estos registros, que sirven para comunicar el procesador con los periféricos de entrada/salida, están situados en un dispositivo que se denomina **controlador** del periférico. El controlador se conecta (se "cuelga") al bus de datos que entra (RD-IN) y que sale (WR-OUT) del procesador y hace de adaptador (*interface*) entre el periférico propiamente dicho (teclado, impresora,...) y el procesador (UPG más la unidad de control).

No nos vamos a ocupar aquí de cómo están construidos internamente los periféricos ni sus controladores para que, por ejemplo, cuando apretamos una tecla del teclado, los bits que codifican esa tecla en código ASCII (u otro código) se escriban en un registro concreto del controlador de teclado.

Únicamente vamos a ver el sistema de entrada/salida desde el punto de vista del procesador. Nos vamos a ocupar, por ejemplo, de cómo leer ese registro del controlador del teclado y escribirlo en uno de los registros de la UPG para, posteriormente, poder procesar el dato que el usuario tecleó.

Los **registros de entrada/salida**, que pertenecen a los diferentes controladores de los periféricos de entrada/salida, se suelen denominar **puertos de entrada/salida** (*input/output ports*). Con este nombre los diferenciamos de los registros del banco de registros del procesador, cuando pudiera haber duda. Un **controlador**, independientemente de si lo es de un **periférico de entrada** (teclado...), **de salida** (impresora...), como si lo es **de entrada/salida** (Disco duro, USB...) tiene registros/puertos tanto de entrada como de salida. Un **puerto de entrada**, por definición, lo es si lo lee el procesador y lo escribe el controlador, mientras que un **puerto de salida** lo escribe el procesador y lo lee el controlador.

A priori, no sabemos cuántos periféricos tendrá el computador y por lo tanto cuantos puertos de entrada y salida necesitamos. Además, un mismo procesador tiene que servir para construir distintos computadores, con un número distinto de periféricos en cada uno de ellos y el usuario final del computador tiene que poder variar el número y tipo de estos periféricos sin por ello tener que cambiar el procesador. Por lo tanto, tenemos que diseñar el procesador para que admita bastantes puertos, aunque para los ejercicios y problemas que haremos aquí sólo necesitemos unos pocos.

En la UPG sólo tenemos un bus de entrada de datos, RD-IN, y otro de salida, WR-OUT. Por ello, a continuación, vamos a dotar al procesador de la capacidad de poder conectar diferentes periféricos de entrada/salida: conectar diferentes puertos de entrada al bus RD-IN y conectar el bus WR-OUT a diferentes puertos de salida.

El procesador (en concreto su UC) deberá ser capaz de seleccionar, en cada ciclo en el que realice una acción de entrada de datos (`IN`, de *Input*) y/o de salida de datos (`OUT`, de *Output*):

- qué puerto concreto de entrada conecta a RD-IN para leer su contenido y almacenarlo, al final del ciclo, en uno de los registros de la UPG (acción IN) y
- en qué puerto de salida escribe el contenido de uno de los registros de la UPG (acción OUT).

Esto requiere que se modifique, en las siguientes secciones, la semántica y la sintaxis de las acciones `IN` y `OUT` tal como las vimos en el capítulo anterior.

9.2.2 Conexión de los puertos de entrada

Queremos poder conectar la salida de uno de entre varios puertos de entrada al bus de entrada de la UPG, RD-IN. Lo vamos a hacer de la única forma que sabemos hacerlo: mediante un multiplexor de buses, como se muestra en la figura 9.1. Los bits de selección de este multiplexor, que indican qué puerto se conecta en cada momento, son generados por el procesador y más concretamente por la unidad de control, formando parte de la palabra de control, como se ve en la figura 9.2. A este nuevo campo de la palabra de control lo denominamos ADDR-IO, al igual que el bus que sale de la UC y va a las entradas de selección del multiplexor. Para estandarizar la palabra de control hacemos que para todas las unidades de control que diseñemos el campo/bus ADDR-IO sea de ocho bits. Así, podemos conectar hasta 2^8 puertos de entrada a la UPG (que es un número suficientemente elevado.

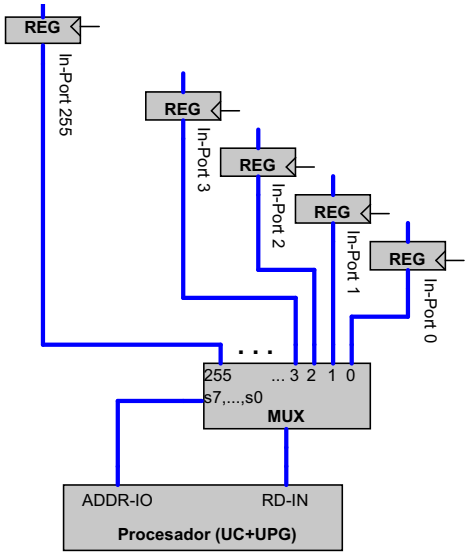


Fig. 9.1 Conexión de 256 registros de entrada al procesador mediante un multiplexor de buses.

Podemos decir ahora que nuestro procesador dispone de un **espacio de direcciones de entrada** de datos de 8 bits, o lo que es lo mismo, que puede direccionar 256 puertos de entrada de datos. A modo de resumen, recordamos que:

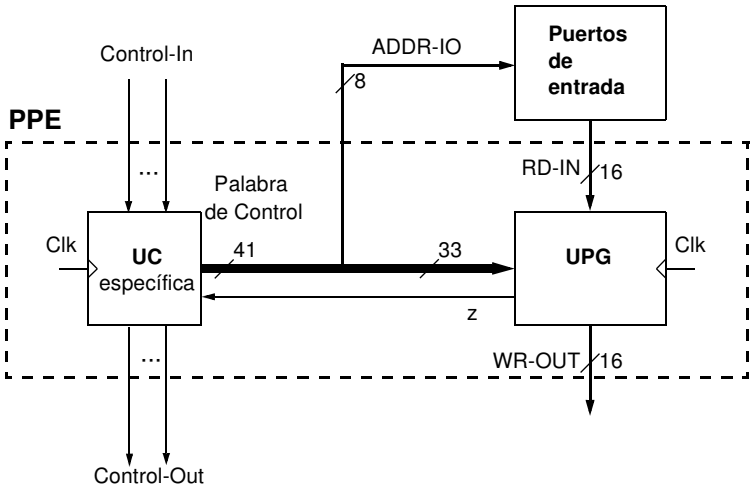


Fig. 9.2 Esquema general de interconexión del procesador de propósito específico (UCE +UPG) con los puertos de entrada y palabra de control de 41 bits

- El bus ADDR-IO sale del procesador (concretamente de la UC) y llega a los controladores de los periféricos para direccionar los puertos.
- Los **puertos de entrada**
    - son escritos por el controlador del periférico de entrada/salida al que pertenecen (la señal de reloj de los registros de entrada la genera el controlador y nada tiene que ver con la señal de reloj del procesador) y
    - son leídos por el procesador.

Si en una implementación concreta de un procesador sólo necesitamos unos pocos puertos de entrada, no implementaremos el multiplexor de buses con los 2^8 buses de entrada, sino uno lo menor posible, como se muestra en la figura 9.3 para 3 puertos con direcciones de entrada 0, 1 y 2.

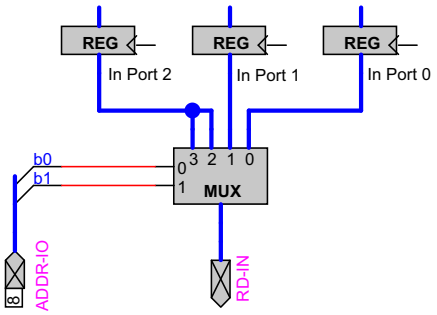


Fig. 9.3 Subsistema sencillo de entrada de datos. Conexión de los puertos de entrada 0, 1 y 2 a los buses del procesador. De los 8 bits del bus de direcciones de entrada/salida, ADDR-IO*, sólo se usan los dos de menor peso. Por ello, el puerto que se lee con la dirección 0 del bus* ADDR-IO *también se leería con las direcciones 4, 8, 12... del espacio de entrada de datos; el puerto 1 se puede leer con las direcciones 1, 5, 9... Como sólo tenemos 3 registros a conectar y para no dejar ninguna entrada del multiplexor al aire, el bus 3 de entrada al multiplexor se ha conectado al mismo registro que el bus 2. En casos como este, el los que se pueden usar muchas direcciones distintas para leer un mismo registro/puerto, usaremos siempre las direcciones más bajas.*

Por último comentar que, como todas las señales y buses de un circuito digital, en el bus ADDR-IO siempre hay bits con valores 0 y/o 1, aunque en la palabra de control el diseñador de la UC haya especificado x en los bits del campo ADDR-IO. No obstante, esto no es ningún problema, y ni siquiera es necesario que la UC genere una nueva señal para validar los bits de ADDR-IO cuando se ejecute la acción `IN`. No pasa nada incorrecto porque en todos los ciclos se esté leyendo algún puerto, ya que la lectura sobre un registro no destruye su contenido y además el valor leído no será escrito en ningún registro de la UPG excepto cuando lo indique la palabra de control al ejecutar la acción `IN` (bits In/ALU = 1 y WrD = 1).

9.2.3 Conexión de los puertos de salida

El bus de salida de datos de la UPG, bus WR-OUT, se debe conectar con la entrada de datos de cada uno de los registros/puertos de salida. Para saber qué puerto debe escribirse de entre todos los que están conectados al bus de salida, la unidad de control del procesador debe generar los 8 bits de dirección del puerto de salida (son 8 bits para poder direccionar tantos puertos de salida como de entrada). Estos 8 bits necesarios para realizar una salida mediante la acción `OUT` los vamos a sacar de la unidad de control por el mismo bus de direcciones (y el mismo campo de la palabra de control) que acabamos de definir para realizar una acción de entrada (`IN`): ADDR-IO; aunque podríamos haber usado un bus/campo diferente. Compartir un bus que sale del procesador con información distinta suele hacerse en

los procesadores integrados en un chip con el objetivo de minimizar el número de patillas (*pins*) del chip. Esto solo tiene un pequeño inconveniente: ahora ya no podremos hacer en un mismo ciclo una acción `IN` junto con una `OUT`, a no ser que las dos accedan a dos puertos, uno de entrada y otro de salida, que tengan la misma dirección, cosa poco usual.

Debido a que en todos los ciclos hay alguna combinación concreta de bits en el bus de direcciones de salida, incluso en los ciclos en los que no se desea escribir en ningún puerto de salida, hacemos que la unidad de control genere, además de los bits ADDR-IO, un bit de control que denominamos Wr-Out (*Write Output*), que valdrá 1 solamente en el ciclo en que se desee escribir en un puerto de salida mediante la acción `OUT`.

Un ejemplo de conexión de los puertos de salida con direcciones 0, 1 y 2 se muestra en la figura 9.4. Los bits del bus ADDR-IO y del bus de datos WR-OUT se deben considerar válidos solamente cuando Wr-Out vale 1. Por ello, la carga del registro concreto de salida se gestiona generando la señal de reloj de cada registro mediante un decodificador con el permiso de funcionamiento (enable) conectado a la señal Wr-Out que genera el procesador, y más concretamente la unidad de control.

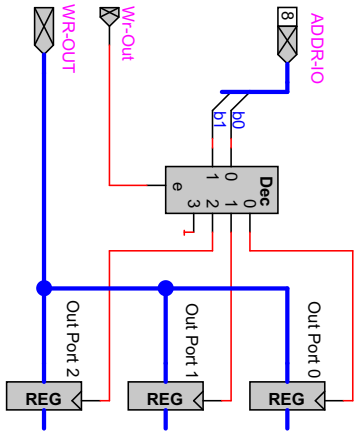


Fig. 9.4 Subsistema sencillo de salida de datos.Conexión de los puertos de salida 0, 1 y 2 a los buses del procesador. La salida 3 del decodificador no se ha usado. Sólo se ha usado un decodificador 2-4 al que le entran los dos bits de menor peso del bus de direcciones de entrada/salida, ADDR-IO*, porque sólo conectamos 3 puertos de salida.*

Para el buen funcionamiento del circuito debe asegurarse que la señal Wr-Out se pone a 1 después de que los buses ADDR-IO y WR-OUT están estables. De esta manera se produce el flanco ascendente en la entrada de reloj del registro concreto que se desea escribir, y no en otro, y el dato que se escribe es el correcto. Los detalles temporales de las señales los veremos cuando diseñemos la unidad de control general del procesador, en el capítulo 10, y más concretamente cuando implementemos, en el capítulo 12, el circuito interno de la lógica de control.

Por último indicamos que la señal de control Wr-Out debe valer 0 para cualquiera de las acciones que se pueden hacer en un ciclo en la UPG excepto cuando se desee sacar un dato (escribirlo en un puerto de salida). No sería correcto que especificáramos que Wr-Out vale x cuando no se realiza una salida ya que si esa x se implementa con un 1 se podría escribir un puerto sin ser eso lo que se desea hacer.

Acabamos de dotar al procesador de un **espacio de direcciones de salida** de 8 bits, para poder direccionar 2^8 puertos de salida de datos. Por último recordemos que los **puertos de salida**:

- son escritos por el procesador (la señal de reloj del registro de salida que se escribe se forma a partir de la decodificación del bus ADDR-IO y de la señal de permiso de escritura Wr-Out, generadas por el procesador (concretamente por la unidad de control) al ejecutarse una acción `OUT`) y
- son leídos por el controlador del periférico al que pertenece el registro.

9.3 El PPE con espacio de direcciones de entrada/salida (UCE+UPG+IOsim)

9.3.1 Palabra de control completa (de 42 bits)

Para soportar los espacios de direcciones de entrada y de salida que hemos diseñado, la unidad de control de propósito específico (UCE) del procesador debe generar, además de los 33 bits de la palabra de control que ya generaba en el capítulo anterior:

- 8 bits para indicar la dirección del puerto de entrada o de salida (ADDR-IO) y
- una señal de un bit (Wr-Out) para ordenar que, en el ciclo en que valga 1, que se escriban los 16 bits del bus de datos WR-OUT en el puerto cuya dirección indican los 8 bits del bus ADDR-IO.

Esto da lugar a la palabra de control completa de 42 bits cuyos nombres, tal como los solemos ordenar, se muestran en la figura 9.5 (se ha sombreado el nuevo bus y la nueva señal). Por falta de espacio y por claridad, codificamos con dos dígitos hexadecimales los 8 bits del bus ADDR-IO, al igual que ya hacíamos con los 4 dígitos hexadecimales del campo N.

Palabra de Control de 42 bits

@A			@B			Rb/N	OP		F			In/Alu	@D			WrD	Wr-Out	N (hexa)				ADDR-IO (hexa)	
x	x	x	x	x	x	x	x	x	x	x	x	0	x	x	x	x	x	X	X	X	X	X	X

Fig. 9.5 Palabra de control de 42 bits después de añadir, a la del capítulo anterior de 33 bits, el bus ADDR-IO de 8 bits y la señal Wr-Out. Estas nuevos campos se han sombreado. La disposición de los nuevos campos dentro de la palabra no es relevante.

Esquema general de interconexión. La figura 9.6 muestra el esquema general de interconexión entre la UC y la UPG, como el de la figura 9.2, pero ahora se han añadido las nuevas señales para acceder también a los puertos de salida (palabra de control de 42 bits).

9.3.2 Reformulación de las acciones de entrada y salida para añadir el direccionamiento

Las acciones `IN` y `OUT` se deben modificar para especificar la dirección del puerto de entrada o salida que se lee o escribe:

- **`IN`.** Lee el contenido de uno de los 2^8 puertos de entrada, cuya dirección genera la unidad de control por el bus ADDR-IO (campo ADDR-IO de la palabra de control), y escribe, al final del ciclo, el valor leído en uno de los 8 registros del banco de registros de la UPG. Por ejemplo, carga `R2` con el contenido del puerto 1 de entrada. Esto lo denotamos como:
  ```
  IN  R2, 1
  ```
- **`OUT`**. Escribe uno de los 2^8 puertos de salida con el contenido de uno de los 8 registros de la UPG (que será leído por el bus A del banco de registros para que esté presente en el bus de salida WR-OUT durante este ciclo). De los 5 tipos de acciones que es capaz de realizar la UPG, la de tipo `OUT`

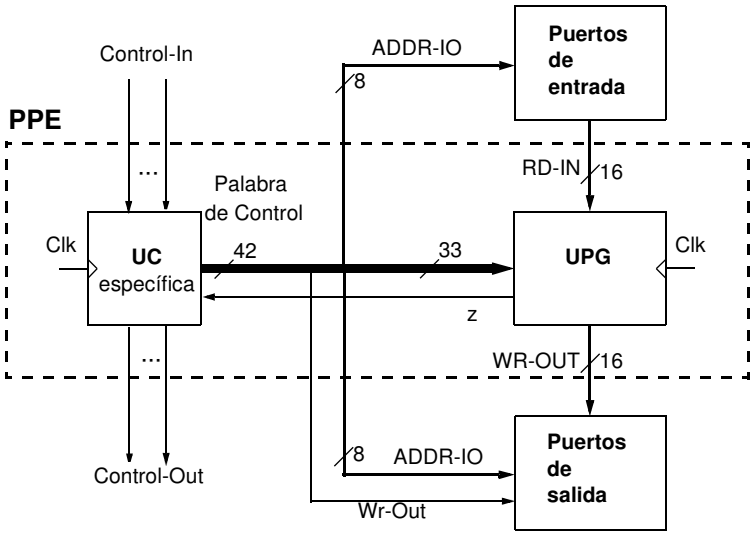


Fig. 9.6 Esquema general de interconexión de la UC con la UPG y con los puertos de entrada y salida (con la palabra de control de 42 bits)

es la única que no requiere que se escriba, al final del ciclo, ningún registro del banco de registros: es la única en la que WrD vale obligatoriamente 0. Por ejemplo, sacar el contenido de `R4` y escribirlo en el puerto 3 de salida. Esto lo denotamos como:

```
OUT  3, R4
```

Valores de los bits de la palabra de control. Para que la UPG haga una de estas acciones en un ciclo determinado, hay que proporcionar a la UPG la palabra de control adecuada durante ese ciclo. En la figura 9.7 se indican los bits de la palabra de control de 42 bits para las acciones que se mostraron en la sección 5.1 del capítulo anterior. En fondo gris de ven los cambios sobre la tabla del capítulo anterior: las nuevas acciones `IN` y `OUT` llevan en el mnemotécnico un campo que indica el puerto de donde se lee (1, para el `IN`) y escribe (3 para el `OUT`) así como las nueva señal Wr-Out y el campo ADDR-IO dispuestos donde solemos ponerlos, aunque el orden de los campos en la palabra de control es irrelevante. Ahora no se pueden hacer en paralelo (en un mismo ciclo) las acciones `IN` y OUT, a no ser que la dirección del puerto de entrada y de salida coincida, ya que comparten el mismo campo de la palabra de control: ADDR-IO.

9.3.3 Un subsistema sencillo de entrada/salida

En la figura 9.6 todos los puertos de entrada se han juntado en un bloque y los de salida en otro. Esto no suele dibujarse así en el esquema a bloques de un computador: se dibuja un bloque con el subsistema de entrada/salida (que contiene a los periféricos y sus controladores y por lo tanto a los registros de entrada y de salida juntos). La figura 9.8 muestra la conexión de un subsistema sencillo de entrada/salida (bloque SIMPLE INPUT/OUTPUT) con la UPG y la figura 9.9 muestra el circuito interno de este bloque que contiene lo siguiente:

Mnemotécnicos	Palabra de Control de 33 bits: @A	@B	Rb/N	OP	F	In/Alu	@D	WrD	Wr-Out	N (hexa)	ADDR-IO (hexa)
`ADD R6, R3, R5`	0 1 1	1 0 1	1	0 0	1 0 0	0	1 1 0	1	0	X X X X	X X
`CMPLEU R3, R1, R5`	0 0 1	1 0 1	1	0 1	1 0 1	0	0 1 1	1	0	X X X X	X X
`ADDI R7, R1, -1`	0 0 1	x x x	0	0 0	1 0 0	0	1 1 1	1	0	F F F F	X X
`ANDI R2, R3, 0XFF00`	0 1 1	x x x	0	0 0	0 0 0	0	0 1 0	1	0	F F 0 0	X X
`NOT R4, R2`	0 1 0	x x x	x	0 0	0 1 1	0	1 0 0	1	0	X X X X	X X
`MOVE R1, R5`	1 0 1	x x x	x	1 0	0 0 0	0	0 0 1	1	0	X X X X	X X
`MOVEI R3, 0XFA02`	x x x	x x x	0	1 0	0 0 1	0	0 1 1	1	0	F A 0 2	X X
`IN R2, 1`	x x x	x x x	x	x x	x x x	1	0 1 0	1	0	X X X X	0 1
`OUT 3, R4`	1 0 0	x x x	x	x x	x x x	x	x x x	0	1	X X X X	0 3
`ANDI -, R3, 0x8000`	0 1 1	x x x	0	0 0	0 0 0	x	x x x	0	0	8 0 0 0	X X
`NOP`	x x x	x x x	x	x x	x x x	x	x x x	0	0	X X X X	X X
~~`IN R2,1 // OUT 3, R4`~~											

Fig. 9.7 Ejemplos de acciones que se pueden hacer en la UPG en un ciclo y las palabras de control completas asociadas a cada acción

- Tres registros/puertos de entrada (con direcciones 0, 1 y 2 del espacio de direcciones de entrada). El usuario del sistema puede cargar cada uno de estos tres registros (pulsando en el conmutador binario que genera la señal de reloj de cada registro para producir un flanco ascendente) con los 16 bits que se obtienen al pulsar los cuatro teclados hexadecimales que alimentan a cada registro.
- Tres registros/puertos de salida (con direcciones 0, 1, y 2 del espacio de direcciones de salida). El usuario del sistema puede ver en todo momento el contenido de cada uno de los tres registros mediante el visualizador de 4 dígitos hexadecimales conectado a los 16 bits de salida de cada registro.

Es importante recordar que el reloj del procesador nada tiene que ver con el reloj de los circuitos de los controladores ni de los periféricos. La lejanía física entre los periféricos de entrada/salida y el procesador hace que no sea posible usar el mismo reloj para todos ellos. La elevada frecuencia del reloj del procesador hace imposible usar el mismo reloj dentro del chip del procesador y fuera, donde se encontrarán los periféricos. Ademas, la diferente velocidad de los periféricos entre sí y con el procesador hace innecesario que usen el mismo reloj. En este sistema sencillo de entrada/salida se ve claro que la escritura de los puertos de salida (que realiza el procesador en el ciclo en que ejecuta una acción `OUT` nada tiene que ver con los momentos en que el usuario escribe un puerto de entrada. Esto, como vemos en la siguiente sección, imposibilita el uso de protocolos síncronos de comunicación entre el procesador y los periféricos.

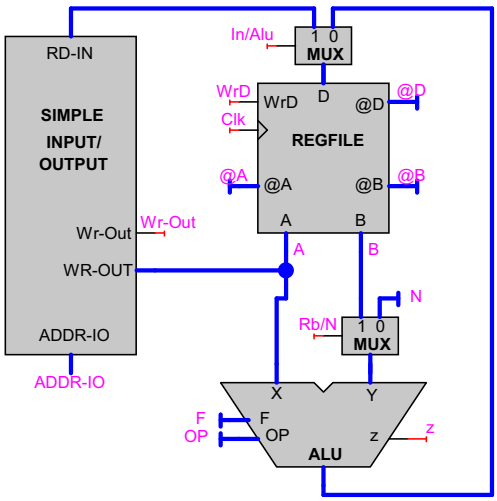


Fig. 9.8 Esquema de interconexión de un subsistema sencillo de entrada salida (bloque IOsim: SIMPLE INPUT/OUTPUT*) con la UPG (con sus bloques* REGFILE, ALU y MUX*)*

9.4 Protocolo de comunicación asíncrono

Vemos ahora cómo se sincroniza la comunicación entre el controlador de un periférico y el procesador. Hay un tipo de aplicaciones que no necesitan ninguna sincronización aunque son las menos usuales. Supongamos, por ejemplo, una aplicación el la que el PPE debe conocer la temperatura ambiente en determinados momentos del día. Para ello, el puerto 2 de entrada es alimentado, por ejemplo, por un termómetro digital que va escribiendo en el registro (puerto 2 de entrada) la temperatura ambiente cada minuto. El procesador, cuando desea saber la temperatura, sólo tiene que ejecutar la acción `IN` sobre el puerto 2 y leerá la última temperatura escrita por el termómetro. Si para la aplicación es suficiente precisión saber que la temperatura leída en el momento t es la temperatura ambiente obtenida en un instante del intervalo de tiempo que va de t menos un minuto a t, el sistema funcionará satisfactoriamente sin necesidad de ninguna sincronización.

No obstante, en las aplicaciones más usuales sí que hace falta alguna sincronización entre controlador y procesador. Por ejemplo, el PPE tiene que obtener de un puerto de entrada una secuencia de datos para, por ejemplo, sumarlos todos ellos. No es posible un tipo de protocolo síncrono, como los usados en los capítulos anteriores, que diga, por ejemplo: el ciclo en que en el puerto de entrada 0 haya un 1 significa que al ciclo siguiente en el puerto de entrada 1 se encontrará el primer dato de la secuencia y a partir de entonces cada dos ciclos llegará un nuevo dato. Esto no tiene sentido ya que el periférico y el PPE no tienen el mismo reloj, no pueden ponerse de acuerdo hablando de ciclos ya que para cada uno de ellos cada ciclo empieza en momentos distintos y dura también un tiempo distinto. Si el periférico es más rápido que el PPE se perderán datos de la secuencia y si es al revés el PPE tomara un mismo dato enviado pro el periférico como varios datos seguidos con el mismo valor. Para aplicaciones como esta

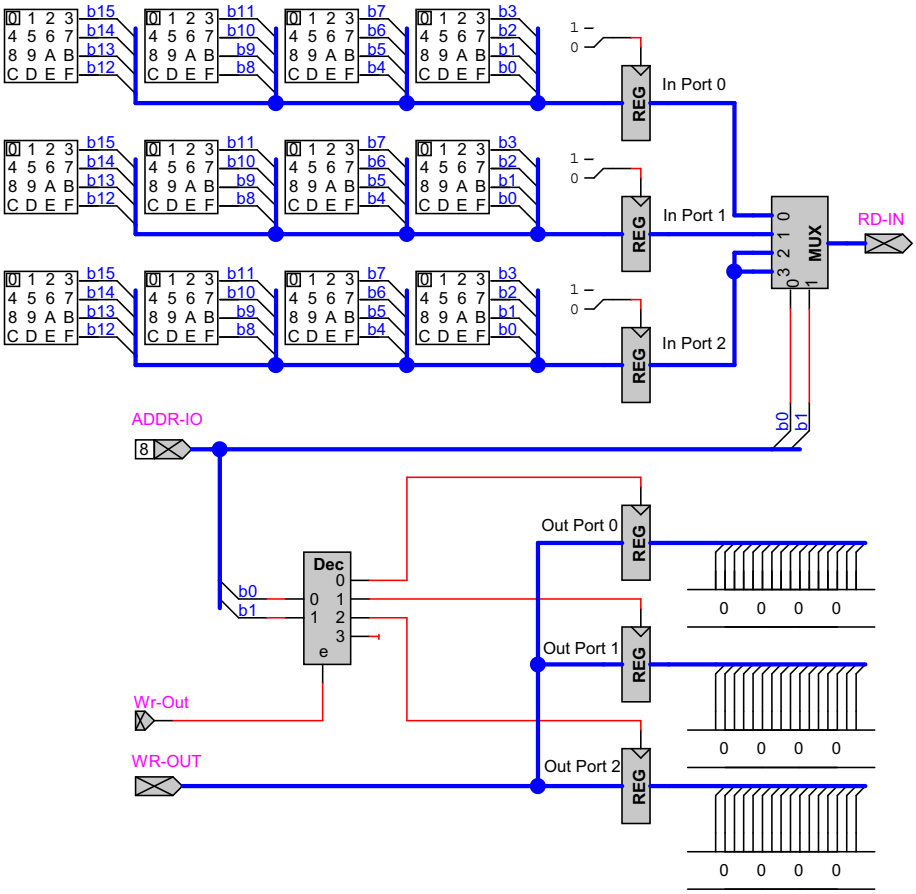


Fig. 9.9 Esquema lógico interno del subsistema sencillo de entrada salida (bloque IOsim: SIMPLE INPUT/OUTPUT*).*

hace falta un protocolo de entrada/salida asíncrono. Los protocolos asíncronos se usan para comunicar dos circuitos secuenciales que no tienen la misma señal de reloj y en ellos no se habla de ciclos.

Vamos a usar, para la comunicación entre controlador y procesador el protocolo de ***Handshaking*** (literalmente, darse la mano) de cuatro pasos, que es, en general, usado para comunicar dos circuitos secuenciales con distinta señal de reloj y por lo tanto en casos en los que no se puede usar un protocolo síncrono: el protocolo no puede hablar de ciclos. Este mismo protocolo se usa en otros ámbitos en los que se cambia el nombre de emisor/receptor por *Master/Slave*, Cliente/Servidor, etc.

9.4.1 Handshaking entre dos circuitos secuenciales con distinto reloj

Vamos a definir primero el protocolo asíncrono de cuatro fases entre un circuito secuencial emisor y otro receptor que no tienen la misma señal de reloj (y que el emisor no sabe nada de la velocidad del receptor y este también desconoce la velocidad del emisor). En este apartado la explicación es general

y no suponemos que ni el emisor ni el receptor son uno de ellos un PPE y el otro un controlador de un periférico de entrada/salida; ya consideraremos esto en el siguiente apartado.

Tal como muestra la figura 9.10, la comunicación entre emisor y receptor se establece mediante dos señales de control Req (*Request*, petición) y Ack (*Acknowledgement*, reconocimiento, acuse de recibo), de un bit cada una, y un bus, BUS, de n bits, por donde se comunicará el dato.

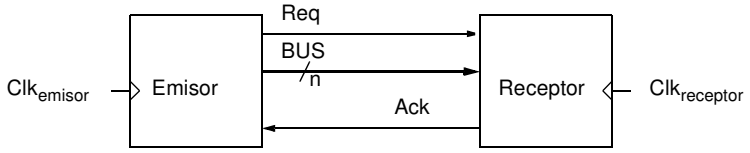


Fig. 9.10 Conexión entre emisor y receptor para el protocolo de handshaking de cuatro pasos.

La señal Req es generada por el emisor (va del emisor al receptor) mientras que Ack va al revés, es generada por el receptor. El caso contrario, en el que Req va del receptor al emisor y Ack del emisor al receptor es igualmente posible, pero no lo tratamos aquí. En cualquier caso el bus de datos va del emisor al receptor.

El protocolo tiene cuatro pasos para comunicar cada dato (por eso se denomina *Four-Cycle Handshaking*). La comunicación la inicia el subsistema que genera la señal Req, en nuestro caso el emisor. El emisor realiza el paso 1, terminado este, el receptor realiza el paso 2, después el emisor realiza el 3 y por último el receptor el 4. A partir de este momento el emisor ya puede comenzar por el paso 1 para iniciar el envío de otro dato. La transacción correcta de los datos la proporciona el hecho de que con este protocolo ninguno de los dos subsistemas avanza al paso siguiente hasta que el otro subsistema no le ha asegurado que ya ha terminado el paso anterior. Por ello, en este tipo de protocolos de Handshaking no es necesario hacer ninguna suposición previa sobre el tiempo que tarda cada subsistema en actuar. En la figura 9.11 puede verse un cronograma con los cuatro pasos del protocolo. Empezamos la explicación suponiendo que las dos señales de control, Req y Ack, están a 0.

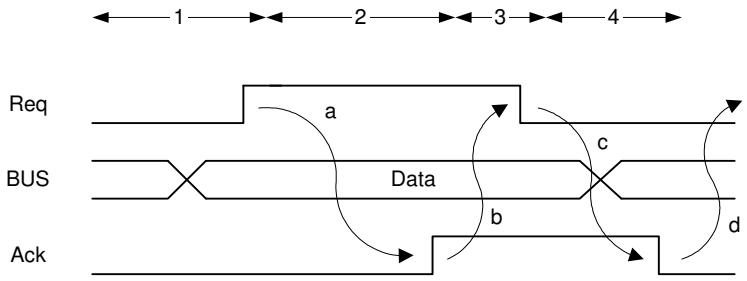


Fig. 9.11 Cronograma de las señales del protocolo de handshaking de cuatro pasos.

El emisor, comprueba que Ack valga 0 y si no lo vale se espera hasta que pase a valer 0 (la necesidad de esto se entiende mejor después de explicado el protocolo completo). Alimenta el bus de datos con el dato que desea enviar. Cuando las señales del bus están estables pone a 1 la señal Req. Esta señal le indica al receptor que tiene un nuevo dato válido en el bus de datos. El emisor no puede presuponer nada al respecto del tiempo que tardará el receptor en enterarse de que Req vale 1 ni del tiempo que tardará en leer el dato. Por ello, tiene que seguir alimentando la señal Req a 1 y el dato en el bus hasta el paso 3, que es cuando recibe notificación del receptor de que ha leído el dato (acuse de recibo).

El receptor, al ver Req a 1 se entera de que hay un nuevo dato válido en el bus, lee el dato (posiblemente lo almacene en un registro) y cuando ya no necesita que el emisor mantenga el dato en el bus, pone a 1 la señal Ack (acuse de recibo). El receptor tampoco puede presuponer el tiempo que tardará el emisor en enterarse de que la señal Ack vale 1, por lo que debe mantenerla hasta que el emisor le indique que ya la ha visto, lo que ocurre en el paso 4.

El emisor, al ver la señal Ack a 1 pone a 0 la señal de Req. Poniendo a 0 la señal Req, el emisor le notifica al receptor que ha visto Ack a 1 y que ya no necesita que la mantenga a 1 por más tiempo. Además, como el emisor ha visto Ack a 1, sabe que puede dejar de mantener los datos en el bus, pues el receptor ya los ha leído.

El receptor, al ver que el emisor ha puesto a 0 la señal Req, sabe que el emisor ya ha visto la señal Ack activada y que no necesita mantenerla por más tiempo, por lo que la pone a 0.

Ahora las dos señales de control están a 0: el emisor puede iniciar otra comunicación poniendo Req a 1, y se repite el proceso...; el receptor mira Req esperando que valga 1 para leer un nuevo dato, y se repite el proceso,...

Finalizada la explicación del protocolo se ve más claro que el emisor, antes de empezar una nueva comunicación debe esperar a que Ack valga 0. Si no lo hace y después de poner Req a 0 (paso 3) pone un nuevo dato en el bus y pone a 1 Req sin asegurarse de que Ack está a 0 (paso 1 mal iniciado), al pasar al paso 3 puede ver Ack a 1 (si el receptor todavía no la ha puesto a 0) y creer que el receptor ya ha leído el nuevo dato, cosa que no es cierta, ya que este Ack es del dato anterior.

La figura 9.12 muestra los fragmentos de los grafos de estados de los circuitos secuenciales emisor y del receptor. Los grafos son parciales en el sentido que sólo muestran las señales Req y Ack del protocolo que acabamos de ver. En estos dos grafos no se especifican las acciones del emisor y del receptor con el dato a transmitir. Por último, antes de pasar a la explicación detallada del protocolo que si el sistema estuviera en un bucle, transmitiendo un dato detrás de otro, el estado E2 podría juntarse con el E0 y formar un bucle de dos estados. En este caso haría falta una señal de entrada, proveniente por ejemplo de un contador, para salir del bucle por el estado E0 cuando ya se hubieran enviado todos los datos. Algo similar pasa con el grafo del receptor: R2 podría juntarse con R0. No obstante, cualquier sistema emisor (o receptor) suele requerir más estados, pues tiene que generar los datos a enviar, cargar los datos a enviar en el registro de salida, etc. (cargar el dato recibido en un registro, procesarlo, etc.).

Emisor. Cuando la señal Req pasa de 0 a 1 el dato a enviar debe estar estable en el bus, puesto que el receptor puede ser rapidísimo y leer el dato inmediatamente después de ver la señal Req a 1. Por otro lado, el registro fuente se puede cargar con el dato a enviar varios ciclos antes de activar la señal de

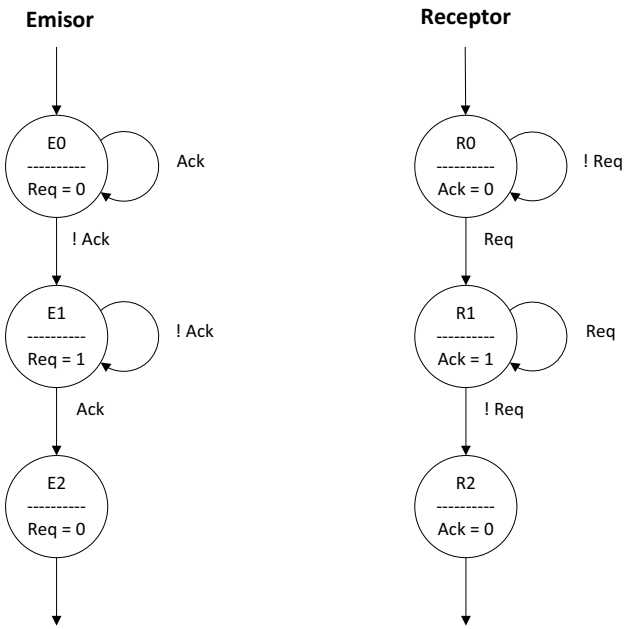


Fig. 9.12 Fragmentos de los grafos de la UC del Emisor y del Receptor para realizar la comunicación de un dato mediante el protocolo de cuatro pasos

Req, esto no da ningún problema. Incluso puede cargarse cuando Ack vale 1. El dato debe estar en el registro fuente desde que Req pasa de 0 a 1 hasta que Ack pasa de 0 a 1. Cuando Ack pasa de 0 a 1 ya se puede cargar otro dato en el registro fuente (que puede ser tanto el próximo dato a enviar como otro dato que no vamos a enviar). También se puede dejar el dato ya enviado durante los ciclos que se desee en el registro fuente, esto tampoco da problemas, el caso es que antes de iniciar un nuevo envío (poniendo Req a 1) se haya cargado el nuevo dato.

Receptor. El receptor sabe que el dato que llega por el bus es estable a partir de que Req vale 1 (aunque realmente el dato pudo estar estable desde mucho antes, esto no lo puede saber el receptor) y que permanecerá estable al menos hasta que él ponga Ack a 1. En este tiempo el receptor puede cargar el valor del bus en un registro destino o puede usarlo directamente para alimentar algún circuito combinacional, cuya salida finalmente se cargará en un registro destino. Mientras el receptor necesite que el dato esté estable en el bus no pondrá la señal Ack a 1.

Si emisor y receptor acuerdan seguir este protocolo, independientemente que uno sea más o menos rápido que el otro, seguro que no se darán problemas de perdida de datos o de recibir un mismo dato varias veces creyendo que son distintos.

Es importante poner de manifiesto que el mismo protocolo funciona correctamente cambiando la dirección del bus de datos (y cambiando los nombres de los subsistemas emisor y receptor, respectivamente). Esto es, el protocolo puede diseñarse para que sea el receptor el que inicie la comunicación, haciendo que Req vaya del receptor al emisor y Ack del emisor al receptor.

9.4.2 Handshaking entre un controlador de un periférico de entrada/salida y el PPE

En esta sección vamos a aplicar el protocolo de handshaking de cuatro fases a la comunicación asíncrona entre un controlador de un dispositivo de entrada/salida y un PPE. Elegimos la versión del protocolo en que siempre es el controlador del periférico el que inicia la comunicación, tanto si es un periférico de entrada de datos (teclado, por ejemplo) como si es de salida (impresora) o de entrada y salida (disco). Así, la señal Req siempre va del controlador al PPE mientras que Ack va en el sentido contrario.

Los dos circuitos secuenciales que tenemos que comunicar ahora son un controlador que dispone de registros/puertos de entrada y de salida y un PPE. Parece evidente que si el dispositivo periférico es de entrada, como lo es un teclado, por ejemplo, el bus de datos del protocolo, que hemos llamado BUS en la sección anterior, por donde el teclado enviará un dato al PPE es el bus RD-IN que entra en la UPG. El teclado escribirá el dato tecleado por el usuario en un registro/puerto del controlador de teclado (no nos ocuparemos de cómo se hace esto) y el PPE leerá este puerto y guardará el dato leído en un registro del banco de registros de la UPG, por ejemplo en `R1` (de esto sí que nos ocuparemos aquí). El puerto donde el teclado escribe el dato debe ser un puerto de entrada ya que es escrito por el controlador y leído por el procesador. Supongamos que es el puerto de entrada con dirección 2. Para copiar el dato tecleado desde el puerto 2 de entrada al registro `R1`, el PPE ejecutará la acción

```
IN R1, 2
```

Así es como se copia el dato, pero ¿cuando se debe ejecutar esta acción? Para responder a esa pregunta hacen falta las señales Req y Ack del protocolo asíncrono. Req debe ir del controlador al PPE y Ack al revés. Parece también evidente, tal como hemos actuado hasta ahora en el diseño de PPE, que Req debe entrar a la UCE del PPE y Ack debe salir de la UCG, ya que son señales de un bit. En una UCE podría hacerse así y no habría mucho problema porque a la UCE entraran varias señales Req de diferentes dispositivos de entrada/salida. No obstante, ahora que vamos hacia una unidad de control de propósito general, UCG (que construiremos en el siguiente capítulo), y que sirve para ejecutar cualquier algoritmo, queremos que los grafos de la unidad de control sean lo más sencillos posible.

La solución más sencilla y más general a esta cuestión es la que han tomado los arquitectos de computadores desde que se diseñaron los primeros computadores: que a la unidad de control no entrará ni saldrá **ninguna** de estas señales de control de los protocolos asíncronos de E/S. Parece extraña esta solución, ¿no? ¿Cómo controlaremos el protocolo asíncrono? Las señales de tipo Req y Ack tienen que existir para que el protocolo funcione, por lo tanto ¿por dónde entrarán y saldrán estas señales? La respuesta es: por el único sitio que nos queda, por el bus de datos de entrada RD-IN y por el bus de datos de salida WR-OUT de la UPG.

En nuestro computador, que es una arquitectura de 16 bits, todos los buses de datos de entrada y de salida son de 16 bits así como los puertos de entrada/salida a los que están conectados. No obstante, para algunos puertos sólo algunos de estos 16 bits tendrán significado. Por ejemplo, podemos convenir que la señal Req que envía un controlador al procesador se implementa a través del bit 0 del puerto que está conectado en la dirección 1 del espacio de entrada/salida y que el resto de bits de este puerto siempre tendrán el valor 0.

Así pues, la comunicación de las señales Req y Ack entre el controlador y el procesador se realizará mediante los puertos de entrada/salida del controlador, conectados al procesador como hemos visto en la sección anterior y usando las acciones `IN` y `OUT` sobre puertos de entrada y salida, al igual que hacemos con la entrada y salida de los datos.

Después de tomar esta decisión, podemos decir que en un controlador de entrada/salida hay tres tipos de registros (puertos) según la función que cumplan en la conexión controlador-procesador:

- de estado (registro de entrada) indican al procesador el estado del controlador, por ejemplo un registro que almacena en uno de sus bits la señal Req,
- de control (registro de salida) a través del cual el procesador informa y le da ordenes al controlador, por ejemplo un registro que almacena en uno de sus bits la señal Ack, y
- de datos (de entrada o de salida) para comunicar datos de 16 bits entre controlador y procesador.

> En muy importante resaltar que, con la decisión que hemos tomado de que las señales de control del protocolo asíncrono de comunicación (Req y Ack) entren y salgan al procesador por los buses de entrada y salida de datos (RD-IN y WR-OUT) en vez de que vayan directamente a la unidad de control, a partir de ahora **la unidad de control sólo tiene un bit de entrada, el bit z, que le llega de la ALU de la UPG.** Con esto, los grafos de la unidad de control son muy simples. De cada nodo sólo salen dos arcos, uno para el valor z igual a 1 y otro para z igual a 0. Hemos conseguido tener grafos simples y generales, eso sí, a costa de requerir más ciclos para resolver un problema que con otros diseños específicos.

Encuesta (*Polling*)

Veamos primero el fragmento de grafo de la UCE del PPE encargado de esperar hasta que la señal Req asociada a un determinado puerto de entrada de datos vale 1. Supongamos que la señal Req que envía un periférico al procesador es el bit 0 del registro que está conectado en la dirección 1 del espacio de entrada. Se sabe también que el resto de bits del puerto 1 siempre vale 0.

La figura 9.13a muestra el estado 0 del receptor que se encarga de esperar a que Req valga 1, según el grafo de la figura 9.12. Pero ahora, como hemos decidido que la señal Req entra por el bus RD-IN proveniente del puerto 1 de entrada, hacen falta al menos dos ciclos (dos nodos del grafo de la unidad de control). El primer ciclo para leer el puerto `1` (registro de entrada que almacena la señal Req) y escribirlo en un registro del banco de registros del procesador, por ejemplo en `R1`. El segundo ciclo para que la unidad de control sepa si el valor de Req que hemos leído es 0 o 1. En este segundo ciclo se compara en la ALU el contenido de `R1` con el valor `1`, de forma que

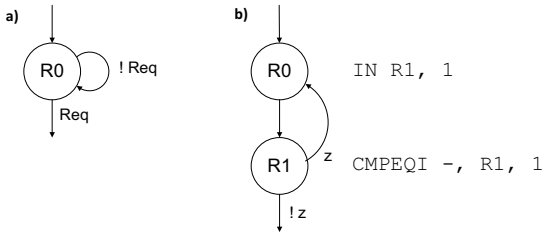


Fig. 9.13 Grafos para esperar a que la señal Req valga 1. a) Nodo 0 del receptor del grafo de la figura 9.13. b) Caso en el que la señal Req es el bit 0 del puerto de entrada 1.

la unidad de control, examinando el bit z que le llega de la ALU, sabe si Req valía 1 cuando se leyó el puerto (resultado de la comparación verdadero, z = 0) o 0 (resultado falso, z = 1). La figura 9.13b muestra esta situación. La comparación se ha realizado mediante la acción

```
CMPEQI -, R1, 1
```

A este tipo de procesos de espera hasta que un determinado bit de un puerto de entrada tiene un valor concreto se les denomina **encuesta** o espera activa (*polling*).

Grafo de estados de la UCE para la entrada de un dato

Ahora vamos a dibujar un fragmento de grafo de la unidad de control para entrar un dato y dejarlo almacenado en un registro de la UPG usando el protocolo de handshaking de 4 pasos.

Es aconsejable ahora recordar los grafos del emisor y receptor de dos circuitos secuenciales, relativos a las señales Req y Ack, que nos ayudaron a entender el protocolo de handshaking de cuatro pasos (ver figura 9.12). Vamos a implementar ahora el grafo del receptor, que es el de la UCE del PPE.

La diferencia con lo visto en la sección 9.4.1es que ahora el periférico de entrada y el procesador de propósito general se comunican a través de los registros de entrada/salida, en vez de mediante las señales directas de un bit Req y Ack. Esta limitación y las propias de la UPG harán que se requieran más nodos en el grafo de la unidad de control del procesador que los 3 nodos del grafo del receptor de la figura 9.12.

El periférico tiene los siguientes registros para comunicarse con el procesador (damos sólo su dirección simbólica ya que no nos interesa la dirección específica del espacio de entrada/salida con las que está asociado cada registro) (ver figura 9.14):

- DATA-IN, registro de entrada de datos, donde el periférico escribe los datos a entrar y de donde el procesador los lee.
- REQ, registro de entrada del estado (es escrito por el periférico y leído por el procesador). El bit 0 de este registro (que denotamos como REQ<0>) es la señal Req del protocolo de entrada del dato. El resto de bits valen siempre 0.
- ACK, registro de salida de control (es escrito por el procesador y leído por el periférico). El bit 0 de este registro (que denotamos como ACK<0>) es la señal Ack del protocolo de entrada del dato. El resto de bits no se usan y los escribiremos con el valor 0.

El fragmento de grafo de la unidad de control del procesador encargado de gestionar la entrada del dato se muestra en la figura 9.15. Los dos primeros nodos realizan la encuesta sobre REQ, de forma equivalente a como se hace en la figura 9.13b. El dato leído se carga en el registro R3 del procesador en el tercer nodo.

El cuarto nodo del grafo, acción OUT ACK, R2, sirve para poner a 1 el puerto ACK ya que al salir del bucle de la encuesta a REQ, el registro del procesador R2 valdrá siempre 1. De forma equivalente ponemos un 0 en ACK mediante la última acción OUT ACK, R2, ya que al salir del bucle de la segunda encuesta a REQ, R2 valdrá siempre 0.

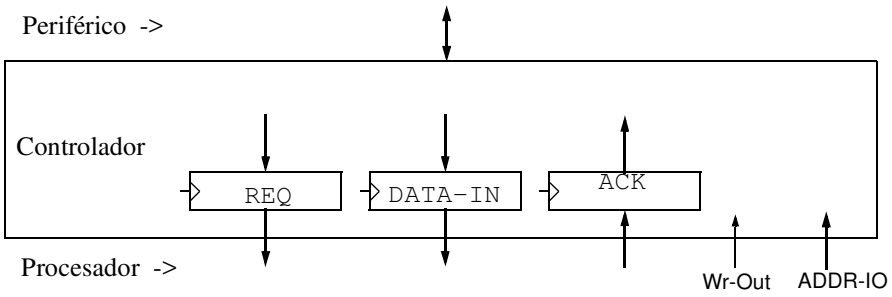


Fig. 9.14 Puertos de entrada/salida para leer un dato de un periférico de entrada con un protocolo de 4 pasos en el que el periférico inicia la comunicación.

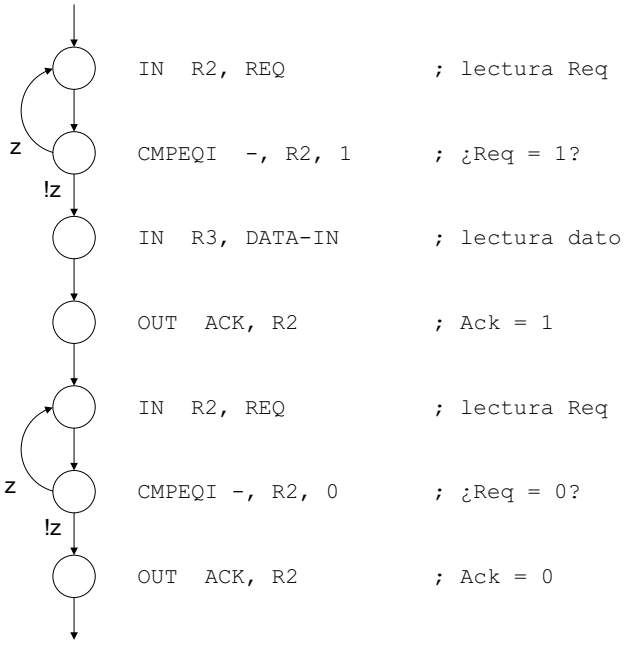


Fig. 9.15 Fragmento de grafo de la unidad de control para leer un dato de un periférico de entrada con el protocolo de 4 pasos.

Grafo de estados de la UCE para la salida de un dato

Para que el protocolo de comunicación de un dato desde el procesador al puerto de datos del controlador de un periférico de salida, como por ejemplo una impresora, sea muy parecido al de entrada se decide que en este caso también sea el periférico el que inicie la comunicación. Por ello, la señal Req del protocolo va del periférico al procesador, y cuando vale 1 el controlador le indica al procesador que está listo para recibir otro dato.

En el caso de usar tres puertos sin efectos laterales para implementar el protocolo de cuatro pasos (puertos REQ, DATA-OUT y ACK), el procesador lee el puerto REQ y escribe el ACK, como en la entrada de datos. La diferencia es que en la entrada lee DATA-IN y ahora, en la salida, escribe DATA-OUT. Por ello, el fragmento del grafo de estados de la unidad de control para sacar un dato es como el de la figura 9.15, cambiando la acción `IN R3, DATA-IN` por `OUT DATA-OUT, R3` (si el dato a sacar se encuentra en el registro `R3` del procesador).

9.5 Efecto lateral al leer/escribir el puerto de datos del teclado/impresora

Vamos a crear el esquema lógico de un modelo sencillo de sistema de entrada/salida que se comporta como un teclado y una impresora. La ventaja de este sistema de entrada/salida es que para leer un dato del teclado (ocurre lo mismo al escribir un dato en la impresora) solamente hacen falta tres acciones, tres nodos en el grafo de la UCE del PPE, para implementar el protocolo de handshaking de cuatro pasos, que en la sección anterior vimos que necesita 7 acciones/nodos. Esto se consigue porque el controlador del teclado y de la impresora realizan en hardware parte del protocolo. Para ello vamos a construir unos registros especiales de entrada/salida que hagan que las cuatro últimas acciones del grafo de la figura 9.15 se efectúen de forma automática por el hardware del controlador, sin requerir ninguna acción por parte del procesador. De esta forma, la entrada o salida de un dato sólo requerirá las tres primeras acciones/nodos del grafo (y además no hará falta tener un puerto para la señal Ack). ¿Cómo será posible? Veamos primero el caso de una entrada de datos de un teclado, aunque sería exactamente lo mismo para cualquier otro periférico de entrada.

9.5.1 Teclado

Para poder eliminar las cuatro acciones últimas del protocolo, las que manejan la señal Ack, construiremos el controlador de forma que la acción de lectura del puerto `DATA-IN` produzca un efecto lateral además de la propia lectura. Este efecto consiste en que el controlador se entere de que el procesador está leyendo su puerto de datos y antes de que el procesador termine la acción de lectura del puerto, el hardware del controlador ponga a 0 la señal Req (el bit 0 del registro `REQ`). Además, el controlador se tiene que enterar de cuándo ha terminado la lectura del puerto de datos, para poder iniciar la entrada de otro dato del teclado al procesador, si el dato está preparado. Para ello, hace falta que el procesador genere una nueva señal, que denominamos Rd-In, que esté activa durante el ciclo en que el procesador esté haciendo la acción

```
IN Rd, AddrPortIn
```

La nueva señal Rd-In, que forma parte de la palabra de control que genera la UCE, sale del PPE y va al controlador del teclado (y de los otros controladores de periféricos de entrada, si los hay). La figura 9.16 muestra la nueva palabra de control con 43 bits, donde se ha resaltado la señal Rd-In.

Veamos un ejemplo concreto de cómo se pueden construir puertos con este efecto lateral, ya que así es como funcionan los controladores de casi todos los dispositivos de entrada/salida. La figura 9.17 muestra un modelo muy simple de teclado hexadecimal y su controlador, todo junto. Como es usual en los controladores, el registro que tiene la información del bit Req del protocolo se le denomina, en general, **registro de estado** (*status register*). En nuestro controlador de teclado el registro de datos está

conectado a la dirección 0 del espacio de entrada y se le ha dado la dirección simbólica, o el nombre, KEY-DATA, mientras que el registro de estado, KEY-STATUS, está conectado a la dirección 1 de entrada. Para leer estos registros se usa un multiplexor de buses que selecciona el registro a leer usando solamente los dos bits de menor peso del bus de direcciones de entrada, ADDR-IO. Esto es así porque, finalmente, en todo el computador sólo se usan las direcciones de 0 a 3 del espacio de entrada.

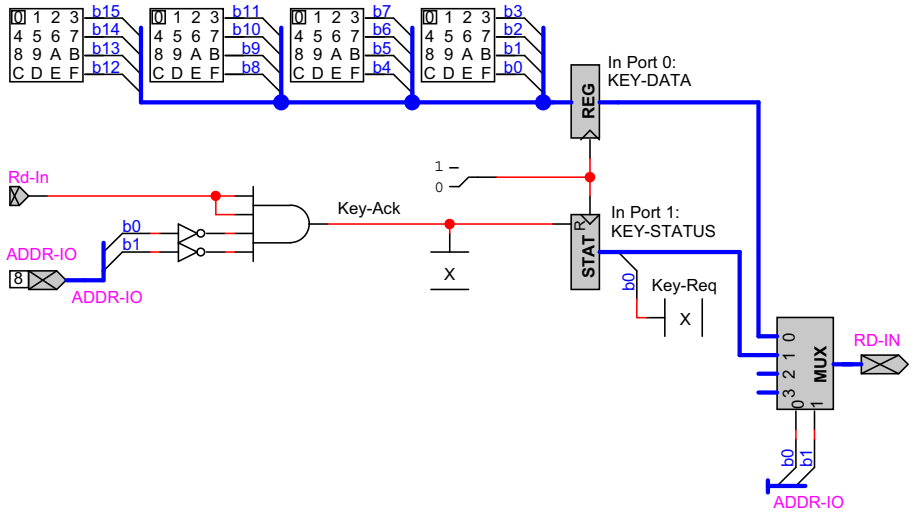


Fig. 9.17 Hardware de un modelo simplificado de teclado y su controlador. Como efecto lateral de la lectura del puerto de datos KEY-DATA *se pone a 0 el bit 0 del puerto* KEY-STATUS.

Veamos la parte del circuito del controlador encargada de detectar que se está leyendo el puerto de datos, KEY-DATA (la acción de lectura sobre este puerto la realizará el procesador después de haber detectado que Req vale 1). Para ello el controlador decodifica la dirección del puerto KEY-DATA, dirección 0, mediante una puerta And con los dos bits de menor peso de ADDR-IO negados. Para validar la decodificación de la dirección se entra a esta misma puerta And la señal Rd-In, que vale 1 solamente cuando el procesador está realizando la acción `IN` y los bits de ADDR-IO están estables. La señal de salida de esta puerta And la denominamos Key-Ack, ya que es equivalente a la señal Ack del protocolo de cuatro pasos. Esta señal se activa durante el ciclo de procesador en que se está leyendo el puerto

Palabra de Control de 43 bits

@A			@B			Rb/N	OP		F			In/Alu	@D			WrD	Wr-Out	Rd-In	N (hexa)				ADDR-IO (hexa)	
x	x	x	x	x	x	x	x	x	x	x	x	0	x	x	x	x	x	x	X	X	X	X	X	X

Fig. 9.16 Palabra de control a la que se le ha añadido la señal Rd-In *para poder tener efectos laterales con la lectura de un puerto.*

KEY-DATA. Cuando esta señal toma el valor 1, el hardware del controlador sabe que el procesador ha visto a 1 el registro de estado y de hecho sabe que ha iniciado la lectura del registro de datos. Por eso el controlador (como veremos a continuación en detalle) pone a cero el bit Req de su registro de estado (REQ) para que el procesador no lea el mismo dato otra vez en ciclos siguientes. Además, cuando la señal Key-Ack pasa a 0 el controlador sabe que ya ha terminado la lectura del puerto de datos y si lo desea puede iniciar el envío de otro dato. Y así se repite el proceso.

Antes de analizar cómo funciona el sistema completo, veamos cómo está construido internamente el registro de estado. KEY-STATUS es un registro especial (que hemos etiquetado como STAT) cuya implementación interna se muestra en la figura 9.18. Los 16 bits de salida del registro siempre valen 0 excepto el bit de menor peso que se pone a 1 cada vez que llega un flanco ascendente en su señal de entrada de reloj, Clk. Para implementar el bit 0 de este registro se ha usado un biestable D activado por flanco con entradas asíncronas de puesta a cero, R, *Reset*, y puesta a 1, S, *Set*. Este tipo de biestable no se ha usado hasta ahora en este curso, pero se comentó su funcionamiento cuando se introdujeron los biestables. Cuando la entrada S vale 1, independientemente de la señal Clk, el biestable se pone a 1 (pasado el tiempo de propagación). Lo mismo ocurre con la entrada R, que pone el biestable a 0. Se dice que S y R son entradas asíncronas, porque modifican el valor del biestable en el momento en que se activan, sin que tenga que llegar un flanco ascendente de reloj.[1]

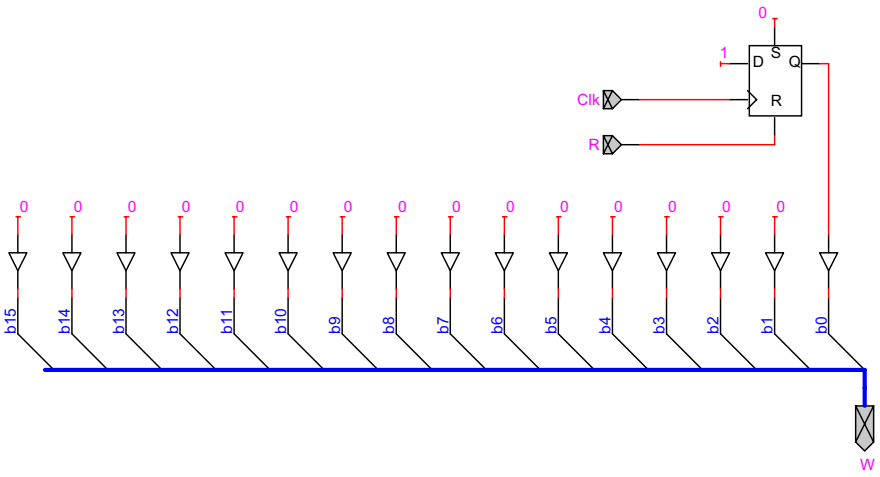


Fig. 9.18 Hardware interno del registro STAT *usado para implementar el puerto* KEY-STATUS.

De cara al procesador este simple teclado funciona como uno de verdad. Otra cosa es que nosotros haciendo clic en el teclado hexadecimal y en el conmutador binario y mirando en el visor binario que nos muestra el valor lógico de la señal Key-Ack, vamos a seguir el protocolo adecuadamente como si fuéramos un controlador de teclado real.

1. En general, se usan biestables de este tipo cuando se requiere que el biestable tome un valor inicial determinado al encender el sistema y antes de que se produzca el primer flanco ascendente de reloj. Los biestables que almacenan el estado de una unidad de control tienen que tener entradas asíncronas de puesta a cero y a uno, pues necesitan tener un estado inicial, cuando empiece a funcionar el sistema. Nosotros, para simplificar los esquemas lógicos no dibujamos normalmente las entradas asíncronas ni los cables que las conectan.

Veamos ahora en detalle cómo funciona el protocolo completo. Por un lado el procesador está ejecutando el fragmento de grafo de la figura 9.19, que contiene las dos acciones para hacer la encuesta sobre KEY-STATUS, y la posterior lectura del dato almacenado en KEY-DATA. Inicialmente el registro de estado vale 0 y en el registro de datos está almacenado el último dato que se entró. El procesador está haciendo encuesta sobre el registro de estado (cada dos ciclos de procesador lo lee). Nosotros, que hacemos de usuarios y de controlador de teclado, hacemos clic en los 4 teclados hexadecimales para codificar los 16 bits que deseamos entrar al procesador. A continuación producimos un flanco ascendente en la señal de reloj que entra en los registros KEY-DATA y KEY-STATUS haciendo clic en el conmutador binario. Este flanco hace que se cargue el dato tecleado en KEY-DATA y que se ponga a 1 el bit de menor peso de KEY-STATUS, indicándole al procesador que puede leer el dato (podemos hacer otra vez clic en el conmutador binario para dejarlo como estaba ya que el flanco descendente de esta señal no tiene efectos).

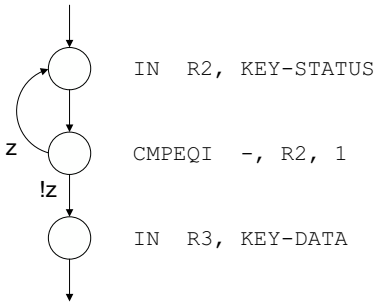


Fig. 9.19 Fragmento del grafo de estados para la entrada de un dato desde el teclado.

El procesador, que está haciendo una encuesta sobre el registro de estado, tan pronto como se da cuenta de que KEY-STATUS vale 1 (al ejecutar `BZ` y ver que `R2` no vale 0), pasa a ejecutar la siguiente acción, `IN R3, KEY-DATA`. En el ciclo en que se ejecuta esta acción, la unidad de control del procesador pone en el bus ADDR-IO la dirección 0 y un poco más tarde, cuando está seguro de que los bits del bus ADDR-IO están estables, pone en la señal Rd-In el valor 1. Como se ve, observando la figura 9.17, esto provoca que la señal que hemos denominado Key-Ack se ponga a 1 y al llegar a la entrada asíncrona de reset, R, del registro de estado, este se ponga a 0.

Nosotros, que además de usuarios hacemos parte del trabajo de controlador del teclado, después de producir el flaco ascendente de reloj de los registros de estado y datos pasamos a mirar el visor binario que nos muestra el valor de la señal Key-Ack (como si hiciéramos una encuesta sobre ella)[1]. El hecho de que Key-Ack se pone a 1 nos indica (le indica al controlador) que el procesador ha visto nuestro registro de estado a 1 (y de hecho sabemos que ya ha iniciado la lectura del puerto de datos). Al ser activada la entrada R del registro de estado, el hardware pone a 0 el registro de estado. Nosotros debemos seguir observando Key-Ack. Cuando pase a 0, cosa que ocurre muy rápido (menos del tiempo de un ciclo de procesador), nos indica que ya podemos iniciar el envío de otro dato. De hecho, el protocolo por parte del controlador comienza mirando y esperando que Key-Ack valga 0, aunque en esta explicación lo hemos dejado para el final, que es el principio de la comunicación de otro dato.

1. No obstante, y aunque no forma parte del protocolo, a modo de curiosidad podemos mirar el visor binario del bit de menor peso del registro de estado, Req. Al hacer clic en el conmutador vemos cómo este visor se pone a 1 y pasado un tiempo muy corto, pues normalmente el procesador va muy rápido y está haciendo una encuesta sobre ese bit, vemos cómo este bit pasa a valer 0. Eso nos indica que se está leyendo el puerto de datos.

Así pues, en nuestro computador tendremos un teclado como el visto en la figura 9.17 y con sólo las tres acciones del fragmento de grafo de la figura 9.19 podremos leer un dato con el protocolo asíncrono de 4 pasos. Esta simplificación del grafo supone un pequeño coste:

- la unidad de control del procesador debe generar una señal binaria Rd-In que se activa en el ciclo que se efectúa la acción `IN` y
- el controlador debe crearse él mismo la señal Key-Ack con una puerta And e implementar por hardware el efecto lateral de puesta a cero del puerto de control.

9.5.2 Impresora

Al igual que los de entrada, los controladores de periféricos de salida tienen puertos especiales con efectos laterales. La diferencia con la entrada es que ahora, en la salida, se tiene que desencadenar el efecto lateral cuando el procesador escribe en el puerto de datos del controlador (y no al leer un puerto). El efecto lateral consiste en que el controlador se entere de cuándo se inicia la escritura de su puerto de datos (mediante la implementación interna de la señal que hemos denominado Print-Ack, que pasará a valer 1), ponga a 0 el registro de estado (PRINT-STATUS) y se entere de cuándo ha terminado la escritura (cuando Print-Ack pase a valer 0) para prepararse a recibir un nuevo dato, iniciando otra comunicación, poniendo Req a 1.

La figura 9.20 muestra la construcción de un modelo muy simple de impresora y su controlador con efectos laterales. La impresión consiste en ver el dato representado en hexadecimal por el visor. El registro de datos, PRINT-DATA se ha conectado en la dirección 0 del espacio de salida y el de estado PRINT-STATUS, a la dirección 2 del espacio de entrada (los puertos 0 y 1 de entrada están ocupados por los registros de datos y estado del teclado). Como en el caso del teclado nosotros haremos la parte del protocolo de comunicación de cuatro pasos haciendo clic en el conmutador binario y mirando en el visor binario de la señal Print-Ack.

El controlador construye la señal Print-Ack de forma similar a como lo hace un controlador de entrada pero usando la señal Wr-Out que sale del procesador y que se activa cuando los bits del bus de direcciones ADDR-IO están estables en el ciclo en que el procesador está ejecutando la acción `OUT`.

La figura 9.21 muestra las tres acciones que forman el fragmento de grafo para escribir el contenido de `R3` por la impresora.

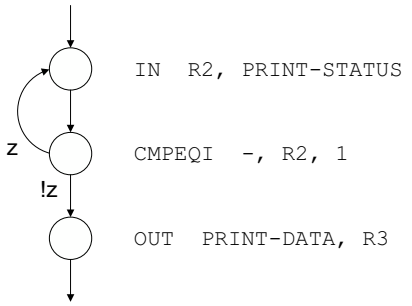


Fig. 9.21 Fragmento de grafo de estados para la salida de un dato por la impresora.

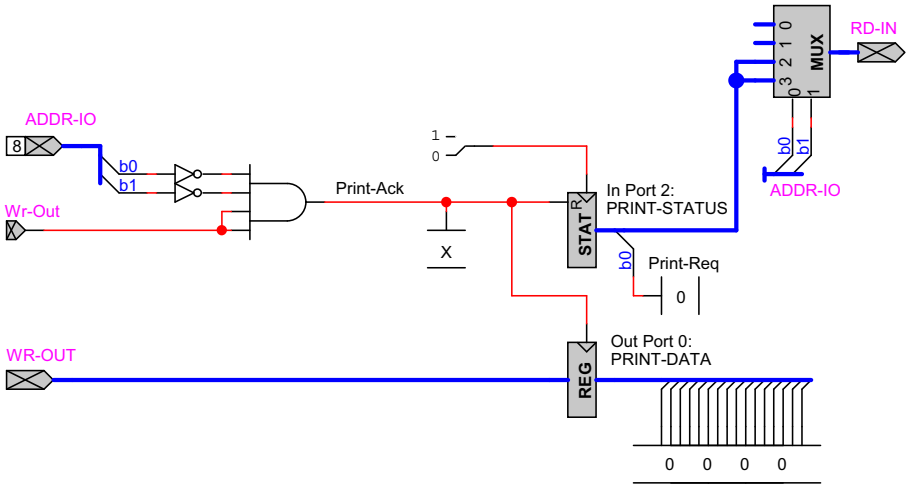
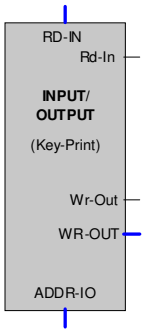


Fig. 9.20 Hardware de un modelo simplificado de impresora y su controlador. Como efecto lateral de la escritura del puerto de datos PRINT-DATA *se pone a 0 el bit 0 del puerto de estado* PRINT-STATUS.

9.6 Subsistema de entrada/salida con teclado e impresora



A la derecha de este párrafo se muestra el símbolo del subsistema de entrada/salida con un teclado y una impresora (bloque IO: INPUT/OUTPUT (Key-Print)) cuyo circuito interno se detalla en la figura 9.22. Este circuito es la unión de los modelos simplificados del teclado (figura figura 9.17) y la impresora (figura 9.20) junto con sus controladores. De ahora en adelante, supondremos conectado este sistema de entrada/salida a todos los procesadores de diseñemos, incluidos los de propósito general que formarán el computador en los capítulos siguientes.

Teclado. Una lectura del registro KEY-DATA (con dirección 0 del espacio de entrada) provoca la puesta a cero del registro de estado KEY-STATUS (con dirección 1 del espacio de entrada).

Impresora. Una escritura en el registro PRINT-DATA (con dirección 0 del espacio de salida) provoca la puesta a cero del registro PRINT-STATUS (con dirección 2 del espacio de entrada). ,

No obstante lo dicho, de cara a hacer ejercicios definiremos otros controladores de dispositivos de entrada/salida con efecto lateral sobre la lectura/escritura del registro de datos de entrada/salida como podría ser un disco.

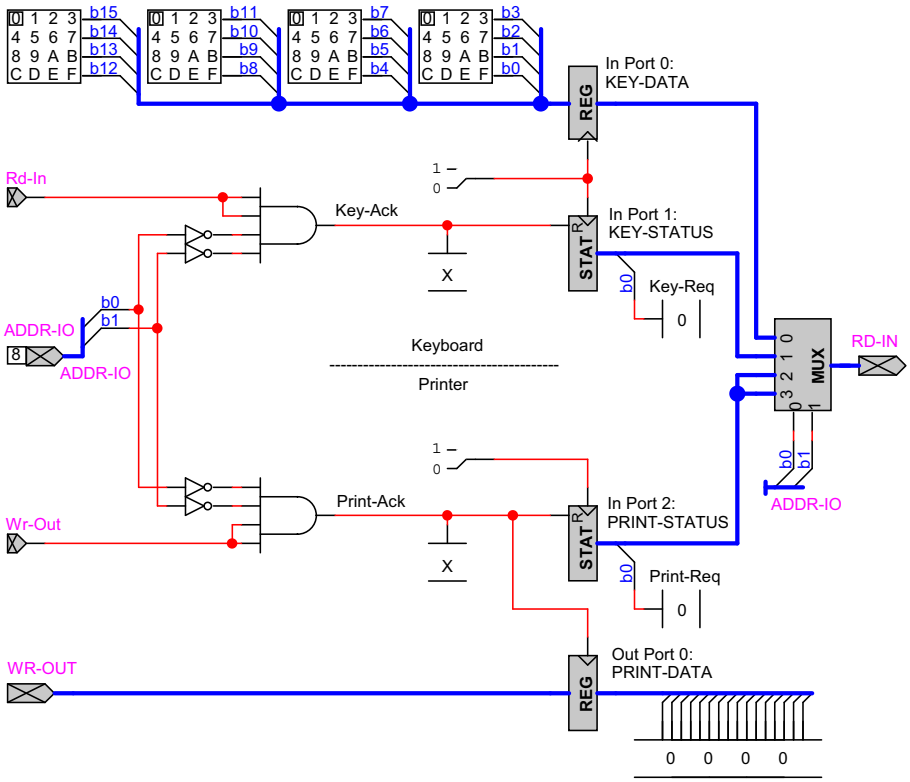


Fig. 9.22 Circuito interno del subsistema de entrada/salida con un teclado y una impresora (bloque IO: INPUT/OUTPUT (Key-Print)*).*

9.7 Ejemplo de PPE con entrada/salida por teclado/impresora

9.7.1 Máximo común divisor

La figura 9.23muestra el grafo de estados de la UCG que junto con la UPG forman un PPE que calcula el máximo común divisor de dos los dos números que recibe por el teclado y una vez calculado envía el resultado a la impresora, repitiendo este proceso indefinidamente.

El grafo es como el de la figura 8.10 del capítulo anterior, que calcula el MCD de dos números almacenados en R0 y R1 y deja el resultado en R1, al que se le han añadido al principio lo 6 nodos para entrar los dos operandos del teclado y escribirlos en R0 y R1 y al final se han añadido los 3 nodos para imprimir el resultado. Del último nodo se pasa al primero incondicionalmente para repetir el proceso indefinidamente.

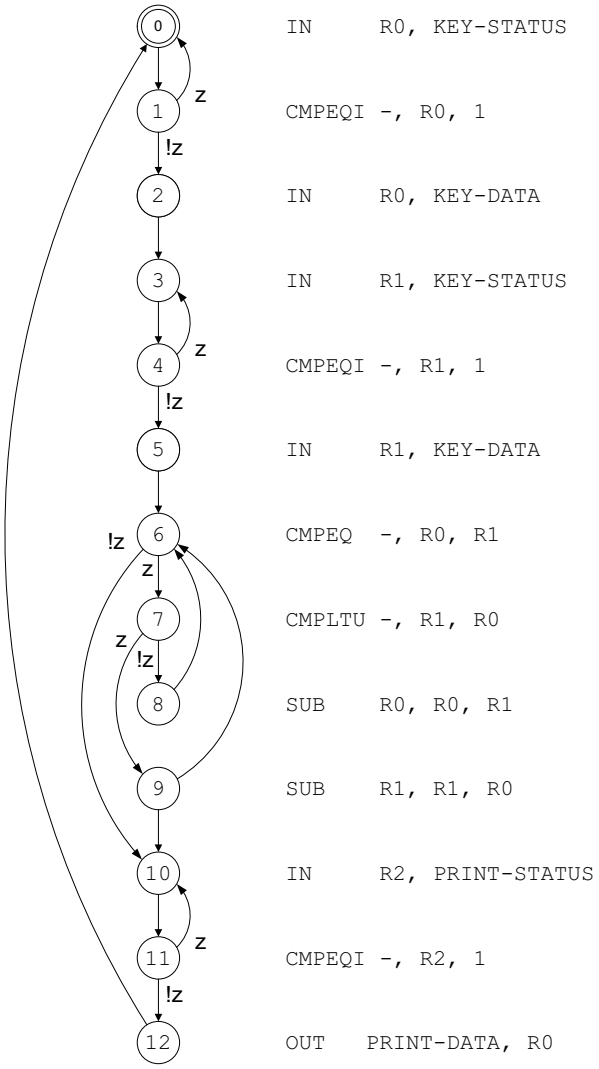


Fig. 9.23 Grafo de estados para calcular indefinidamente el MCD. Entrada de datos y salida de resultado por el teclado y la impresora respectivamente