6.4. Unidad aritmético-lógica (ALU)

El subsistema de cálculo del sistema microprocesador es una implementación ligera y algo modificada de una ALU cualquiera de un procesador RISC. Soporta un juego muy reducido de instrucciones típicas, incluyendo un par de instrucciones específicas para simplificar la implementación de las tareas de control.

La ALU dispone de dos registros de carga para operandos, un acumulador y un registro específico para carga de índices destinado al acceso indexado a memoria. Además posee internamente un registro con 4 bits de estado (Z, C, N y E) que se presentan directamente al control central para la decodificación de estados. El funcionamiento de dichas señales de estado se especifica en la Tabla 6. La información de su uso y las instrucciones que los modifican se encuentra junto con la especificación de las instrucciones del procesador.

Flag	Descripción
Z	'Zero', vale '1' si el resultado de la última operación fue cero o la última comparación fue verdadera
C	Bit de acarreo de la suma, vale '1' cuando la suma o la resta se desbordan
N	Bit de acarreo de 'nibble', vale '1' cuando existe acarreo entre los 'nibbles' (medio byte)
E	Error, vale '1' cuando se llega a algún resultado inesperado en la ALU

Tabla 6. Descripción de los flags del procesador

La ruta de datos de la arquitectura puede verse en la Figura 4.

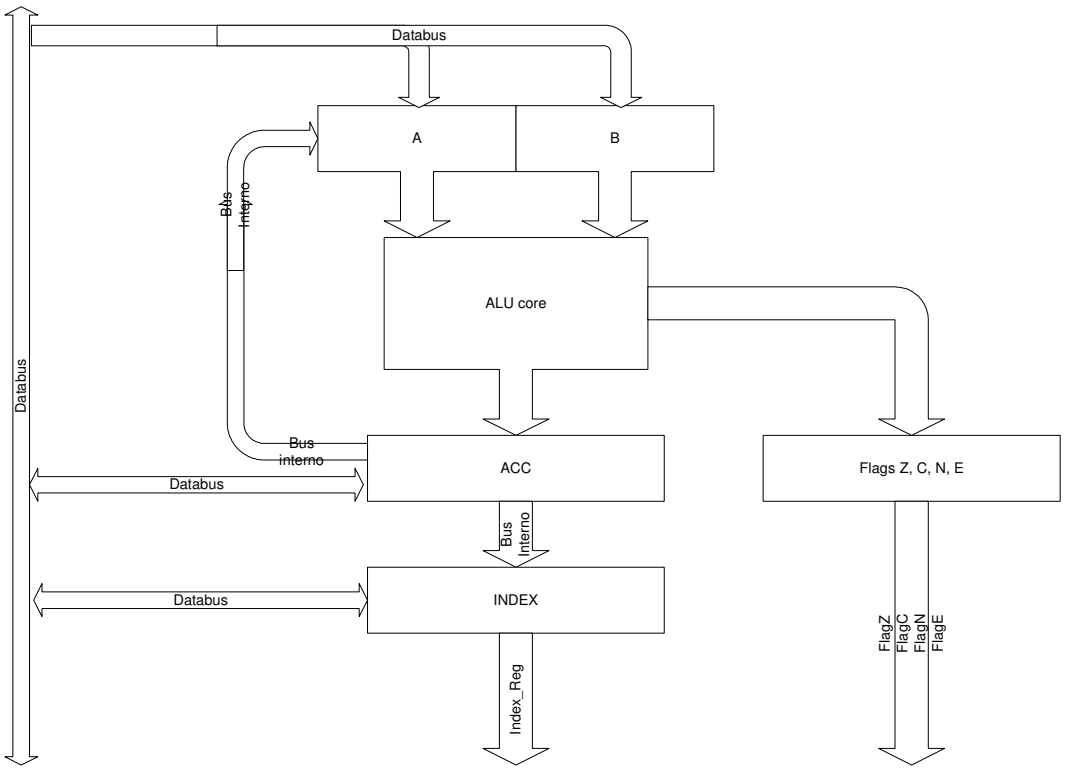

Figura 4. Ruta de datos de la ALU

Las operaciones que debe realizar la ALU se especifican más adelante, junto con el resto de instrucciones del procesador.

Los puertos de entrada y salida de la ALU son los indicados en la Tabla 7.

Línea	Sentido	Descripción
Reset	Entrada	Asíncrono y activo a nivel bajo. Inicializa todos los registros internos a 0x00
Clk	Entrada	Reloj principal del sistema, 20MHz
Alu_op[5..0]	Entrada	Bus de microinstrucciones del procesador. La ALU permite hasta 64 operaciones, aunque en este prototipo sólo se implementarán algunas
Databus	Bidireccional	Bus de datos del sistema
Index_Reg	Salida	Conexión directa del registro de índice a la unidad decodificadora de instrucciones
FlagZ	Salida	Flag de cero
FlagC	Salida	Flag de acarreo
FlagN	Salida	Flag de acarreo en 'nibble'
FlagE	Salida	Flag de error

Tabla 7. Líneas de entrada y salida de la ALU

Entre las microinstrucciones que recibe la ALU han de encontrarse aquéllas que permitan la salida de datos de los registros. Estas microinstrucciones no se corresponden con ninguna instrucción en particular del procesador (más bien con varias de ellas).

Respecto de la temporización, **todas las instrucciones de la ALU han de realizarse en un único ciclo de reloj**.

6.5. ROM de programa

El programa del microcontrolador se encuentra en este bloque. Dicho programa se encuentra grabado en una ROM que será accedida según las necesidades del control principal del procesador.

Las instrucciones del procesador son de longitud variable, 1 ó 2 palabras de 12 bits. En general, la primera palabra de la instrucción define la funcionalidad de la misma, reservándose la segunda para la definición de constantes o direcciones de memoria en aquellas instrucciones que lo requieran (como por ejemplo en una instrucción de salto).

Las líneas del componente son las mínimas necesarias para hacer que éste sea accesible únicamente por el control principal. Estas líneas se indican en la Tabla 8.

Línea	Sentido	Descripción
INS_Addr[11..0]	Entrada	Bus de direcciones de la ROM de programa
INS_Bus[11..0]	Salida	Bus de instrucciones

Tabla 8. Líneas de entrada y salida de la ROM

Dado que el bus de recogida de instrucciones es de sólo 12 bits, puede ser necesario acceder dos veces a la memoria de programa para la total ejecución de una instrucción. Esta decisión recae en manos del control principal del procesador.

Por último, nótese que sólo es posible direccionar un máximo de 4096 palabras de programa. Esto, aunque pueda parecer en exceso limitado, es más que suficiente para un primer prototipado de un sistema. En el momento en el que se desee añadir más memoria de programa, bastará con añadirle líneas de dirección al control central y a la ROM, sin variar en absoluto el resto del diseño.

6.6. Control principal

El núcleo de la ejecución del procesador se encuentra en este bloque. Esta máquina de control se encarga de recoger las instrucciones de la ROM de programa y ejecutarlas, generando las microinstrucciones necesarias.

Las tareas de este bloque pueden resumirse como sigue:

- Si durante el estado inicial de la máquina el controlador DMA está pidiendo los buses, concedérselos inmediatamente y detenerse hasta que el DMA acabe su acceso a memoria.
- Recoger una dirección de memoria y decodificar la instrucción contenida.
- Ejecutar la instrucción, para lo que puede ser necesario generar microinstrucciones, realizar un acceso a la memoria de datos (RAM), o incluso recoger una nueva palabra de la memoria de programa.
- Si la instrucción en concreto es del tipo 4 (SEND), puede ser necesario congelar ('*stall*') el procesador hasta que la instrucción se lleve a cabo.
- Regresar al estado inicial.

Las líneas de que este componente debe disponer son las indicadas en la Tabla 9.

Línea	Sentido	Descripción
Reset	Entrada	Asíncrono y activo a nivel bajo
Clk	Entrada	Reloj principal del sistema, 20MHz
ROM_Data[11..0]	Entrada	Bus de datos de la memoria del programa
ROM_Addr[11..0]	Salida	Bus de direcciones de la memoria de programa
RAM_Addr	Salida	Bus de direcciones de la memoria de datos
RAM_CS	Salida	'Chip select' de la RAM
RAM_Write	Salida	Microinstrucción para escritura en la RAM
RAM_OE	Salida	Microinstrucción para permitir la salida de datos de la RAM
Databus	Bidireccional	Bus de datos del sistema
DMA_RQ	Entrada	Línea de petición de buses del controlador DMA
DMA_ACK	Salida	Microinstrucción de entrega de los buses al controlador DMA
SEND_comm	Salida	Microinstrucción para iniciar una transmisión por la línea serie
DMA_READY	Entrada	Señal de estado del controlador DMA
ALU_op[5..0]	Salida	Microinstrucciones con la operación parar realizar en la ALU
Index_Reg	Entrada	Conexión directa al registro índice de la ALU
FlagZ	Entrada	Flag de cero de la ALU
FlagC	Entrada	Flag de acarreo de la ALU
FlagN	Entrada	Flag de acarreo de 'nibble' de la ALU
FlagE	Entrada	Flag de error de la ALU

Tabla 9. Líneas de entrada y salida del control principal

7. Instrucciones del procesador

El lenguaje ensamblador del microcontrolador está compuesto por cuatro tipos de instrucciones de longitud variable. A continuación se especifica cada uno de estos tipos y la funcionalidad de las instrucciones. La sintaxis del lenguaje propiamente dicho está desarrollada en el epígrafe 9. Lenguaje ensamblador.

Es de notar que, en las palabras de memoria que definen instrucciones, la información necesaria para decodificarlas se encuentra en los 8 bits menos significativos, siendo el resto cero. Esto es así para permitir la posterior ampliación del juego de instrucciones a nuevos tipos y nemónicos.

Para comprender el formato binario de estas instrucciones es necesario tener en cuenta las constantes definidas en el paquete `PIC_pkg` (fichero `PIC_pkg.vhd`). Esas constantes se usan posteriormente en el pequeño compilador asociado al sistema de modo que la ROM generada pueda ser comprendida.

7.1. Tipo 1, instrucciones relativas a la ALU

Nemónico	Descripción	Flags que modifica
ADD	A + B	Z, C, N

SUB	A – B	Z, C, N
SHIFTL	Gira hacia la izquierda el contenido del acumulador, introduciendo un cero	
SHIFTR	Gira hacia la izquierda el contenido del acumulador, introduciendo un cero	
AND	'and' lógico entre A y B	Z
OR	'or' lógico entre A y B	Z
XOR	'xor' lógico entre A y B	Z
CMPE	A = B	Z
CMPG	A > B	Z
CMPL	A < B	Z
ASCII2BIN	Convierte A del formato ASCII al binario (para números, devuelve FF si hay error)	E
BIN2ASCII	Convierte A del binario al ASCII (para números menores de 0x10, devuelve FF si hay error)	E

Tabla 10. Instrucciones de tipo 1

Todas estas instrucciones son de 1 palabra de longitud.

En las comparaciones (CMPE, CMPL, CMPG) se varía el bit de cero (Z), colocando un '1' si la comparación fue verdadera y un '0' en caso contrario. Por otra parte, el bit E se coloca a '1' cuando se produce un error en las funciones de conversión ASCII2BIN y BIN2ASCII. Los flags C, N y E no se utilizan para nada en este germen del procesador, pero se implementan para facilitar su posterior incorporación a las funciones del mismo.

El formato binario de esta instrucción es simple. Los 4 bits más significativos están a 0 y después tenemos 2 bits que indican el tipo de instrucción y el resto el código de operación. Haciendo uso de las constantes definidas en `PIC_pkg.vhd`, un ejemplo de instrucción de tipo 1 sería "0000" & TYPE_1 & ALU_ADD.

7.2. Tipo 2, instrucciones de salto

Nemónico	Parámetro	Descripción
JMP	Dirección Inmediata Etiqueta (#Nombre)	Salto incondicional
JMPTrue	Dirección Inmediata Etiqueta (#Nombre)	Salto si FlagZ = '1'

Tabla 11. Instrucciones de tipo 2

Todas estas instrucciones son de longitud fija, 2 palabras en este caso. La primera palabra es la instrucción en sí y la segunda la dirección de salto. Las etiquetas empleadas serán traducidas por el compilador en direcciones inmediatas.

El formato binario se define del siguiente modo:

- La primera palabra: los 4 bits más significativos están a 0 y después tenemos 2 bits que indican el tipo de instrucción y el resto el código de operación. Haciendo uso de las constantes definidas en `PIC_pkg.vhd`, un ejemplo de instrucción de tipo 2 sería "0000" & TYPE_2 & JMP_COND.
- La segunda palabra es la dirección de salto que ha sido traducida al formato adecuado por el compilador (el compilador las indica en formato hexadecimal).

7.3. Tipo 3, instrucciones de movimiento de datos

Nemónico	Parámetro Destino	Parámetro Origen	Descripción
LD LDI	.A .B .INDEX	.ACC	Movimiento entre registros
LD	.A .B .INDEX .ACC	Constante [Constante] [Dirección Inmediata]	Carga los registros de la ALU
LDI	.A .B .INDEX .ACC	[Constante] [Dirección Inmediata]	Carga los registros en la ALU, utiliza el registro .INDEX como índice para acceder a memoria
WR	Dirección inmediata Constante		Escribe datos en memoria
WRI	Dirección inmediata Constante		Escribe datos en memoria, utiliza el registro .INDEX como índice para acceder a memoria

Tabla 12. Instrucciones de tipo 3

Estas instrucciones pueden variar su longitud. Si la transferencia es entre registros (LD .A, .ACC) la instrucción tendrá una única palabra, en el resto de casos tendrá dos.

La instrucción LD entre registros puede utilizarse únicamente para cargar ACC en algún otro de los registros de la ALU; así instrucciones del tipo LD .ACC, .A son ilegales. Por otra parte es de notar que no es posible transferir una constante directamente desde la memoria de programa hasta la memoria de datos; para realizar esta operación son necesarias dos instrucciones del procesador:

```
LD .ACC, 45
WR .ACC, x40
```

Las instrucciones LDI y WRI tienen la particularidad de utilizar el registro .INDEX sumándolo a la dirección de memoria indicada de modo que, si se va a acceder a la memoria de datos, se puedan leer y escribir tablas de forma sencilla.

A la memoria principal se puede acceder por medio de constantes definidas con anterioridad. Nótese sin embargo que las constantes se pueden definir desde 0 a 4095, pero a la hora de hacer uso de estas para acceder a memoria o cargar los registros, sólo se usarán las 8 líneas más bajas de las 12 posibles.

El formato binario se define del siguiente modo:

- La primera palabra: los 4 bits más significativos están a 0 y después tenemos 2 bits que indican el tipo de instrucción, 1 bit que indica el tipo de operación, 2 que indican la fuente desde la que se carga y el resto (3 bits) el destino.
- La segunda palabra, en los casos en los que sea necesaria, tiene diversos significados según el tipo de instrucción, fuente y/o destino. En general define todo aquello que no pudo definirse en la primera palabra de la instrucción, esto es, direcciones de memoria o constantes.

Algunos ejemplos de definición de instrucciones junto con su código máquina y descripción pueden verse en la Tabla 13.

Código	Traducción	Interpretación
LD .A, .ACC	X"0" & TYPE_3 & LD & SRC_ACC & DST_A	Carga el acumulador en .A
LD .A, X65	X"0" & TYPE_3 & LD & SRC_CONSTANT & DST_A X"065"	Carga 0x65 en .A (8 bits menos significativos)
LDI .A, [X65]	X"0" & TYPE_3 & LD & SRC_INDXD_MEM & DST_A X"065"	Carga el contenido de la dirección (0x65 + .INDEX) en .A (8 bits menos significativos)
LD .A, C	X"0" & TYPE_3 & LD & SRC_CONSTANT & DST_A X"000" (según el valor de C)	Carga el valor de la constante C en .A
LD .A, [C]	X"0" & TYPE_3 & LD & SRC_MEM & DST_A X"000" (según el valor de C)	Carga el contenido de la dirección (C) en .A
WR X65	X"0" & TYPE_3 & WR & SRC_ACC & DST_MEM X"065"	Guarda el acumulador en la dirección 0x65
WRI C	X"0" & TYPE_3 & WR & SRC_ACC & DST_INDXD_MEM X"000" (según el valor de C)	Guarda el acumulador en la dirección 0x65 + .INDEX

Tabla 13. Ejemplos de instrucciones de tipo 3

Es necesario reiterar que, aunque las palabras de la memoria de programa tienen 12 bits, sólo los 8 menos significativos son empleados al hacer uso de constantes y direcciones de memoria.

7.4. Tipo 4, instrucciones especiales

Nemónico	Descripción
SEND	Para el procesador y ejecuta una petición de envío al controlador DMA

Tabla 14. Instrucciones de tipo 4

Mediante esta instrucción, de una sola palabra, se detiene el procesador y se le da la orden al controlador DMA para que transmita los datos almacenados en las posiciones de memoria DMA_TX_Buffer (2 bytes). Esta instrucción queda definida en binario del siguiente modo: X"0" & TYPE_4 & "000000".

8. Consideraciones adicionales para la implementación

8.1. Memoria RAM

La memoria RAM del procesador puede dividirse en dos bloques de cara a su implementación. En primer lugar la batería de posiciones de memoria que poseen algún significado específico (las posiciones 0x00 a 0x3F), y por otro lado la memoria de propósito general (posiciones 0x40 a 0xFF). Atendiendo a su descripción funcional, la única diferencia apreciable entre una zona de memoria y la otra es la existencia de una señal de Reset para la primera. Esta señal se provee como ayuda para agilizar la inicialización del sistema y reducir la memoria de programa, pero podría eliminarse y realizar esta función con software. Para la implementación de este bloque se ofrece como ejemplo el archivo `RAM.vhd`, que constituye una memoria de propósito general de 16 bytes.

La tarea del alumno con este bloque es crear una entidad que contenga una memoria RAM de propósito general como la ofrecida, pero del tamaño adecuado, y una segunda memoria que contenga la

señal de Reset. La memoria de carácter general deberá construirse en una entidad aparte (bastará con modificar adecuadamente el fichero `RAM.vhd`), siendo luego instanciada en el bloque principal de la memoria.

La ventaja de trabajar con la memoria RAM genérica por separado es que la herramienta de síntesis la reconocerá como tal, y será capaz de optimizarla haciendo un mejor uso del hardware. Esto no ocurre sin embargo con la zona de memoria con señal de Reset, para la que el sintetizador deberá hacer uso de flip-flops discretos hasta completarla. Evidentemente, el alumno debe ser capaz de decodificar la dirección especificada para acceder a uno u otro banco de memoria.

La RAM dispone de líneas de salida destinadas a mostrar el estado de alguno de los registros de propósito específico. Las líneas SWITCHES muestran el estado de los interruptores, para lo que basta que cada línea esté directamente asociada con el bit menos significativo de cada registro de estado de interruptor. Las líneas TEMP_L y TEMP_H deben convertir el '*nibble*' bajo y alto del registro T_STAT (dirección 0x31) a un formato legible en un display de 7 segmentos. Esto puede conseguirse de forma sencilla construyendo una ROM con una instrucción del tipo with-select (aunque existen otras maneras más o menos elegantes de conseguir el mismo resultado). Para la construcción de estas funciones de conversión, téngase en cuenta que las líneas de los visualizadores están 'mapeados' con respecto a las líneas de salida según lo indicado en Figura 5.

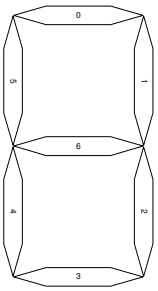

Figura 5. Líneas de los displays de 7 segmentos

Una última consideración a tener en cuenta es el hecho de que **el bus de datos debe permanecer en estado de alta impedancia mientras que la señal OE esté a nivel bajo** (véase la especificación de las señales en la Tabla 4).

Como apoyo para la codificación de la RAM se han definido algunos tipos de datos en el paquete `PIC_pkg` (contenido en el archivo `PIC_pkg.vhd`). Este paquete contiene, aparte de los ya mencionados tipos, otras funciones y tipos destinados a facilitar el desarrollo de la práctica.

8.2. ALU

La implementación de la ALU resulta bastante inmediata. Basta con seguir la ruta de datos indicada anteriormente. Para desarrollar de una forma cómoda el conjunto de las microinstrucciones que debe recibir la ALU, se ofrece el tipo `alu_op` como parte del paquete `PIC_pkg.vhd`. Este tipo no está completo, pero da una idea bastante adecuada de cómo debe procederse para hacer uso de un tipo enumerado. Si se hiciera uso de este tipo, la cabecera de la entidad de la ALU sería:

```
entity ALU is
  port (
    Reset         : in    std_logic;    -- asynnchronous, active low
    Clk           : in    std_logic;    -- Sys clock, 20MHz, rising_edge
    u_instruction : in    alu_op;       -- u-instructions from CPU
    FlagZ         : out   std_logic;    -- Zero flag
    FlagC         : out   std_logic;    -- Carry flag
    FlagN         : out   std_logic;    -- Nibble carry bit
    FlagE         : out   std_logic;    -- Error flag
    Index_Reg     : out   std_logic_vector(7 downto 0);   -- Index register
    Databus       : inout std_logic_vector(7 downto 0)    -- System Data bus
  );
end ALU;
```

Nótese que **este bloque debe mantener el bus de datos (Databus) en alta impedancia siempre que no le sea imprescindible su uso**.

8.3. Controlador DMA

La primera de las máquinas de control que forman parte del sistema es la encargada de gestionar la carga de los datos recibidos por la línea serie en la memoria. Su tarea básica es "escuchar" las líneas de EMPTY del receptor RS232 y Send_Comm procedente del control principal del procesador.

Para el diseño de este bloque pueden tomarse dos aproximaciones igualmente válidas, bien puede desmembrarse la máquina de control en dos mitades (recepción y transmisión) y añadir algunas señales de negociación entre ellas; o bien se puede optar por la codificación de una única máquina de estados que englobe ambos procesos.

Un esquema de la conexión se muestra en la Figura 6. Las líneas CS, OE, Write_En, Address y Databus son de uso compartido con el procesador principal (en el caso de Databus también con la ALU), por lo que **deberán mantenerse en alta impedancia siempre que no sea necesario usarlas**.

El bus de control (CS, OE y Write_en) es un aspecto crítico del diseño. El control de estas señales recae principalmente sobre la unidad principal del procesador, debiendo cederlas (y quedarse en alta impedancia) únicamente cuando el control DMA así lo requiera.

La funcionalidad de este bloque ha quedado ya suficientemente especificada, por lo que basta en principio codificar la máquina de control que gestione los procesos que se han explicado.

El controlador DMA debe constar de al menos un fichero vhdl, el `DMA.vhd`. De todos modos, el alumno es libre de subdividir las tareas que ejecuta el componente y luego agruparlas en el fichero `DMA.vhd` por medio de componentes.

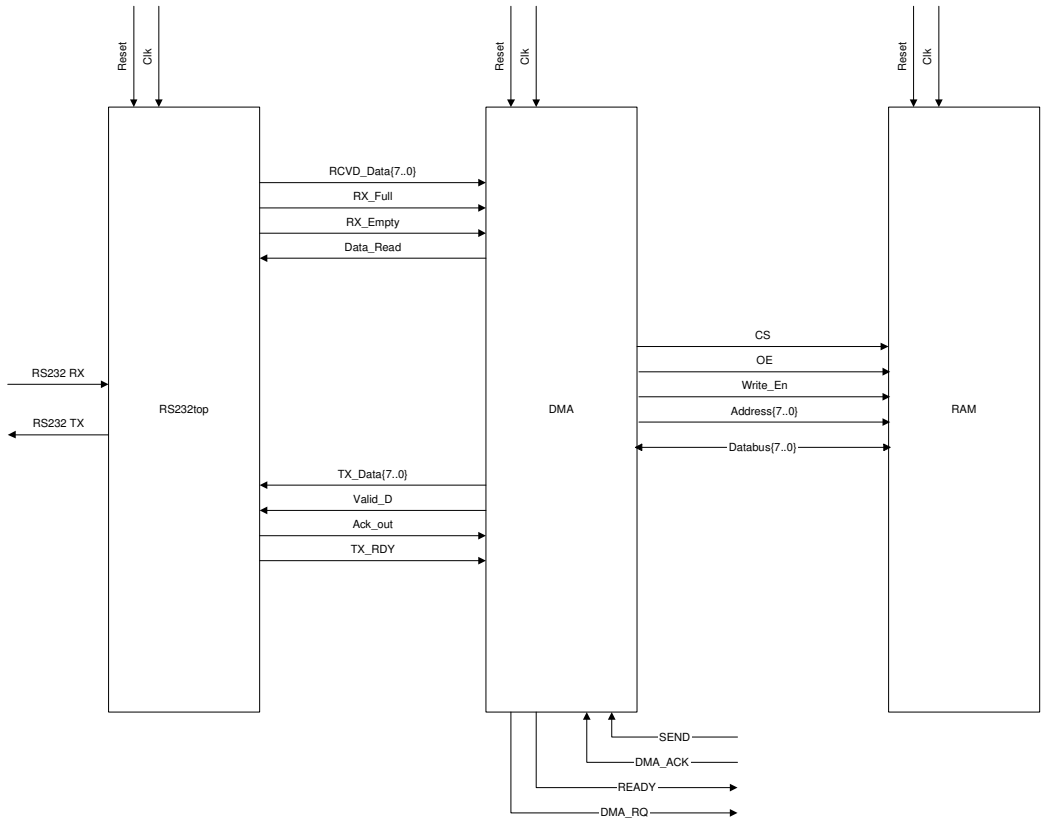

Figura 6. Conexión RS232-DMA-RAM

8.4. Control principal

El control principal es el encargado de interpretar el código del programa guardado en la ROM de instrucciones. De forma resumida su tarea es la del núcleo de todo procesador: recoger la instrucción de memoria (‘*fetch’*), decodificarla (‘*decode’*) y ejecutarla (‘*execute*’). La ejecución de dicha instrucción pasa por la generación de las microinstrucciones necesarias y/o la recogida de una nueva palabra de la memoria de programa (caso de las instrucciones de salto o de movimiento de datos). Además debe tenerse en cuenta que se debe controlar el acceso directo a memoria y ofrecer los buses al controlador DMA siempre que no se esté en medio de una ejecución de instrucción.

Para facilitar la implementación de la decodificación de instrucciones, se ofrecen las constantes que definen una instrucción como parte del paquete `PIC_pkg.vhd`. Para la correcta implementación de esta unidad es conveniente recordar cómo es la estructura básica de una CPU. Una CPU contiene internamente diversos registros que le ayudan en su función ejecutora. Algunos de ellos (los mínimos para que la máquina de control sea realmente efectiva) son:

- Un *contador de programa*: que controla el acceso secuencial o los saltos que se produzcan en la lectura del programa.
- Un *registro de instrucciones*: recuérdese que la primera fase de ejecución de una instrucción es su lectura de la memoria de programa. Mediante este registro se puede mantener internamente una copia de la instrucción que se está tratando de ejecutar.
- Un *registro de almacenamiento temporal*: que no resulta imprescindible, pero que facilita las operaciones en las que sea necesario volcar un contenido de la memoria de programa a otro registro, sea éste interno o externo.

El aspecto más especial de esta CPU es que debe detenerse durante los accesos directos a memoria, así como durante los ciclos de envío de datos por la línea serie. Los accesos directos a memoria no pueden limitarse directamente por parte de la CPU, pero siempre es recomendable que el diseñador del *software* limite al máximo el número de envíos por línea serie. Un envío por línea serie a 115200 bps es siempre mucho más lento que el procesador, cuya frecuencia de funcionamiento hemos fijado en 20 MHz.

Teniendo todo lo anterior en cuenta, el diseño de la CPU es una máquina de control que reproduce el comportamiento especificado con la ayuda de los registros internos indicados y que haga especial hincapié en el exhaustivo control del bus de control de la memoria de datos.

8.5. Memoria de programa

La memoria de programa definitiva que debe emplearse para comprobar la funcionalidad del sistema se proporcionará en un fichero `ROM.vhd`. Sin embargo, y a fin de facilitar la labor de depurado del código, existe también un pequeño compilador al que se le da como argumento un fichero escrito en el lenguaje ensamblador de este procesador. Este compilador se encarga de generar código binario (o su representación según las constantes definidas en el paquete `PIC_pkg.vhd`) en un fichero intermedio y, si no se han producido errores, se genera una ROM en vhdl.

El compilador en cuestión, `pic_compiler`, acepta uno o dos parámetros. El primero de ellos es el fichero que contiene el código fuente y el segundo el archivo en el que se copiará la ROM generada. Si no se especifica el segundo parámetro, el compilador redirige la salida directamente al flujo de salida estándar ‘*stdout*’. Si se producen errores durante la compilación del código, se puede acceder al fichero `output_tmp_file`, creado automáticamente por el compilador y que contiene las instrucciones que se han leído hasta el momento. Nótese que el lenguaje ensamblador definido es ‘*case sensitive*’, por lo que es necesario tener cuidado con las mayúsculas al escribir un programa.

Para asegurar el correcto funcionamiento del compilador, los ficheros de código deben acabar con una línea en blanco. Como ya se comentado, la especificación del lenguaje ensamblador se da en el Apartado 9. Lenguaje ensamblador.

8.6. Fichero 'top' del sistema

Como interfaz estándar para todos los grupos de prácticas se ofrece el fichero `PICtop.vhd`, que inicialmente define los puertos de entrada y salida generales del procesador. En esta entidad serán instanciados los diversos componentes del sistema según éstos sean implementados.

```
entity PICtop is
  port (
    Reset    : in  std_logic;           -- Asynchronous, active low
    Clk      : in  std_logic;           -- System clock, 20 MHz, rising_edge
    RS232_RX : in  std_logic;           -- RS232 RX line
    RS232_TX : out std_logic;           -- RS232 TX line
    Switches : out std_logic_vector(7 downto 0);  -- switch status bargraph
    Temp_L   : out std_logic_vector(6 downto 0);  -- Less significant figure of T_STAT
    Temp_H   : out std_logic_vector(6 downto 0)   -- Most significant figure of T_STAT
  );
end PICtop;
```

8.7. Ficheros de pruebas

A fin de facilitar la construcción de ficheros de pruebas se ofrece un nuevo paquete definido en el fichero `RS232_test.vhd`. Este paquete contiene código VHDL no sintetizable, válido únicamente para simulación. El contenido inicial de este paquete es el procedimiento `Transmit`, que admite un byte de datos y lo envía por la línea que se especifique en formato RS232.

```
procedure Transmit (
  signal   TX   : out std_logic;                    -- serial line
  constant DATA : in  std_logic_vector(7 downto 0)  -- Byte to send
);
```

Como parte del desarrollo de la práctica **se requiere la implementación de al menos los siguientes bancos de pruebas**:

- Un fichero `tb_DMA.vhd` que contenga pruebas del subistema compuesto por el nivel físico RS232, el controlador de DMA y la RAM.
- Un fichero `tb_ALU.vhd` que contenga pruebas de la unidad aritmético-lógica por separado del resto del procesador.
- Un fichero `tb_PICtop.vhd` que contenga pruebas del procesador completo y que simule el envío de algunos comandos de usuario.

9. Lenguaje ensamblador

Para definir completamente el lenguaje ensamblador es necesario añadir ciertas consideraciones. Para facilitar la comprensión de la sintaxis del lenguaje ensamblador se incluye el código que al finalizar la práctica el procesador debe ejecutar con éxito.

9.1. Formato de los datos

El compilador acepta datos en formato hexadecimal, decimal y binario, siempre y cuando estén dentro de los límites definidos por el ancho de la palabra. Esto limita los valores válidos a números sin signo con un valor máximo de 4095.

- Los valores en hexadecimal van precedidos del carácter 'x' o del carácter 'X'.
- Los valores en decimal se escriben por su valor, sin ningún prefijo.
- Los valores en binario van precedidos del carácter 'b' o del carácter 'B'.

9.2. Definición de constantes

Las constantes se pueden definir en cualquier momento a lo largo del programa y son cadenas de letras y números iniciadas siempre por una letra. Para definir una constante se debe seguir el siguiente formato:

NOMBRECONSTANTE : VALOR

Donde VALOR puede ser un dato en cualquiera de los formatos aceptados.

9.3. Definición de etiquetas

Las instrucciones de salto aceptan únicamente valores en hexadecimal o etiquetas como direcciones válidas de salto.

Una etiqueta puede definirse en cualquier línea del programa excepto en una que contenga una instrucción de salto (para evitar incoherencias en la construcción de un programa). Para ello, basta seguir el siguiente formato:

#NOMBREETIQUETA

Donde NOMBREETIQUETA es una cadena de letras y números.

El compilador se encargará de traducir las etiquetas en direcciones efectivas.

9.4. Registros

Los registros definidos en el procesador pueden accederse según se indica en Tabla 15.

Registro del procesador	Indicador para el compilador
A	.A
B	.B
Acumulador	.ACC
Índice	.INDEX

Tabla 15. Indicadores de registros para el compilador

9.5. Comentarios

Se pueden insertar comentarios en líneas nuevas o al final de una instrucción. Los comentarios van siempre precedidos del carácter ';'.

10. Desarrollo de la práctica y metodología

Los grupos de laboratorio deberán entregar un sistema que cumpla los requisitos arriba especificados. Además deberá hacerse entrega del código correspondiente a los bancos de pruebas ('*testbenches*') que se hayan desarrollado para verificar el funcionamiento de cada una de las partes que componen el sistema.

A modo de guía se sugiere que se proceda del siguiente modo:

- Escritura de la memoria RAM del sistema (fichero `RAM.vhd`) y de un banco de pruebas que verifique su funcionamiento con detalle.
- Escritura del controlador DMA (fichero `DMA.vhd`).
- Desarrollo de un banco de pruebas que haga uso del procedimiento `Transmit` incluido en el paquete `RS232_test` para comprobar el funcionamiento de la terna RS232-DMA-RAM.

- Implementación de la unidad aritmético-lógica (fichero `ALU.vhd`) y de un banco de pruebas que verifique su funcionamiento por separado del resto del sistema.
- Implementación del control principal del procesador (`MAIN_CONTROL.vhd`).
- Desarrollo de pequeños programas para verificación de que la máquina de control funciona correctamente con las diversas instrucciones.
- Llegados a este punto es posible unir todos los subsistemas hasta ahora generados y realizar una síntesis a fin de comprobar su correcta interconexión.
- Generación de ficheros de pruebas para comprobar el funcionamiento del sistema completo con programas simples.
- Realización de simulaciones adicionales a fin de comprobar el funcionamiento del procesador con el programa completo.
- Repetición de las pruebas anteriores en simulación post-PAR.
- Síntesis final del sistema y comprobación de funcionalidad sobre la FPGA.

Hay que recordar que, durante la escritura de cada módulo, debe realizarse el proceso de síntesis periódicamente para asegurar que el código generado es sintetizable.

La secuencia propuesta para la realización de la práctica se indica en la Tabla 16.

	Sistemas desarrollados
Etapa 1	RAM y decodificación de direcciones, pruebas
Etapa 2	DMA integrado con RS232 y RAM, pruebas
Etapa 3	ALU, pruebas
Etapa 4	Máquina de control integrada con el resto, pruebas
Etapa 5	Simulación post-PAR, pruebas en placa

Tabla 16. Secuencia de desarrollo propuesta

11. Posibles mejoras del sistema

A fin de ofrecer una idea de la clase de mejoras que se pueden realizar en el desarrollo de esta práctica, se subrayan aspectos susceptibles de mejora:

- Experimentación sobre la interfaz RS232: se propone que los alumnos investiguen otros modos de transmisión, como un menor número de bits de datos, bits de paridad, diferentes velocidades de transmisión, etc.
- Investigación de posibles paralelismos en la CPU, es decir, la posibilidad de solapar ciclos de recogida de instrucciones con la ejecución de las anteriores. Se valorará que se estudien con atención los casos de problemas de paralelismo, como son las instrucciones de salto condicional con condición sin resolver.
- Estudio del paralelismo de la unidad DMA, de forma que las transmisiones y las recepciones sean simultáneas. Esto implica dividir la máquina de control DMA y construir un protocolo de negociación para la posesión del bus de datos entre los subsistemas receptor y transmisor.

Nótese que las mejoras aquí propuestas han de servir como ejemplo para que el alumno investigue e implemente cualquier otra mejora que añada una funcionalidad adicional significativa al sistema.

12. Bibliografía recomendada

- [1] Christopher E. Strangio, *The RS232 Standard. A Tutorial with Signal Names and Definitions, CAMI Research Inc*, 1993.
- [2] Sundar Rajan, *Essential VHDL – RTL Synthesis Done Right, S&G Publishing*, 1998.
- [3] Xilinx, `www.xilinx.com`
- [4] Xilinx, *Spartan-IIE 1.8V FPGA Family*, 2003.
- [5] Xess Corporation, *XSB Board V1.0 Manual*, 2003.

13. Apéndice 1. Recomendaciones finales

Finalmente, es importante recordar que en la realización de todo diseño existen una serie de recomendaciones en el estilo de descripción, provenientes tanto del propio grupo de trabajo como de los fabricantes de las herramientas de diseño. En este apartado se incluyen algunas de las recomendaciones típicas que deben ser tenidas en cuenta por cualquier diseñador de circuitos:

- Recomendaciones referentes a la determinación y evaluación de la arquitectura:
    - Cualquier diseño está compuesto por un conjunto de elementos interconectados, los cuales han de realizar la funcionalidad requerida por la entidad (especificaciones de funcionamiento). La cantidad de elementos que sea posible utilizar estará relacionada con las restricciones de área del sistema.
    - Para cada uno de esos elementos, el diseñador debe especificar el momento en que se han de realizar las operaciones (especificaciones de velocidad y temporización de cada uno de los componentes).
    - **Es importante definir cómo realizar la comunicación entre los distintos componentes o bloques del sistema** (especificaciones de interfaz). Dentro de este grupo de especificaciones debe quedar completamente detallado el formato que utiliza cada una de las señales: si es activa nivel alto o bajo, el número de bits enteros y totales, etc., y en qué momento su valor es válido.
    - Es importante recordar también que el sistema ha de superar una serie de pasos de verificación (especificaciones relativas al test del circuito), por lo que éste debe ser fácil de entender, testear y modificar en el momento que se necesite.
- Recomendaciones referentes a la descripción VHDL del circuito:
    - El estilo de descripción debe ser fácilmente legible y mantenible. En este sentido, en la escritura de los ficheros VHDL se recomienda, entre otras consideraciones, que se distinga la utilización de palabras clave de los nombres definidos por el usuario mediante el uso de mayúsculas y minúsculas, que se utilicen nombres significativos según la función que realicen los distintos elementos, que los ficheros contengan una correcta estructura y comentarios, y que el número de caracteres utilizados en cada línea no exceda de 80.
    - Utilización de las funciones estándar de librería. Además de hacer más simple la lectura y mantenimiento del código, contienen elementos estructurados de uso común que han sido ya verificados.
    - En diseños en los que tanto el tiempo como el área son parámetros críticos es necesario recurrir a los componentes del fabricante. La descripción inicial requiere algo más de tiempo, pero los elementos están ya optimizados y verificados, por lo que las prestaciones obtenidas son bastante mayores.
- Recomendaciones referentes al proceso de síntesis:
    - Es importante conocer y distinguir cuándo utilizar señales y cuándo variables. Las primeras requieren mayor cantidad de memoria, y por tanto ralentizan las simulaciones. Las segundas, por el contrario, dejan una mayor libertad a la herramienta de desarrollo, pero son menos cercanas a la síntesis, y por tanto suelen producir más problemas al utilizar estas herramientas.
    - **Genera todos los componentes del sistema utilizando la misma señal y flanco de reloj.** De esta forma se hace más fácil la verificación por parte de la herramienta de desarrollo y se evitan posibles fallos en la temporización de los componentes en las etapas finales, más complejas, de la síntesis.
    - Para concluir, recuerda que aunque las herramientas CAD facilitan y reducen el tiempo de diseño de los distintos componentes, lo hacen a costa de cierta cantidad de área. Por esta razón, **es importante tener siempre en mente la arquitectura que hay detrás de cada descripción, y comprobar que las inferencias que realiza la herramienta de diseño corresponden con las deseadas.**

14. Apéndice 2. Código completo del programa ensamblador

```
; ZONA DE CONSTANTES ASCII

I          :       X49 ; CODIGO ASCII PARA LA I
A          :       X41 ; CODIGO ASCII PARA LA A
C          :       X43 ; CODIGO ASCII PARA LA C
T          :       X54 ; CODIGO ASCII PARA LA T
S          :       X53 ; CODIGO ASCII PARA LA S
E          :       X45 ; CODIGO ASCII PARA LA E
R          :       X52 ; CODIGO ASCII PARA LA R
O          :       X4F ; CODIGO ASCII PARA LA O
N          :       X4E ; CODIGO ASCII PARA LA N
K          :       X4B ; CODIGO ASCII PARA LA K
F          :       X46 ; CODIGO ASCII PARA LA F

; ZONA DE CONSTANTES PARA MEMORIA

RCBUF0     :       X00 ; DIRECCIONES BUFFER RECEPCIÓN
RCBUF1     :       X01
RCBUF2     :       X02
NINST      :       X03

TXBUF0     :       X04 ; DIRECCIONES BUFFER TRANSMISIÓN
TXBUF1     :       X05

SWBASE     :       X10 ; BASE PARA LA ZONA DE SWITCHES

LEVBASE    :       X20 ; BASE PARA LA ZONA DE ACTUADORES

GPMEM      :       X40 ; BASE DE LA MEMORIA GENERAL
TMP        :       X41 ; UN BYTE PARA CALCULOS

; COMIENZO DEL CÓDIGO, ESPERA ACTIVA POR
; VIGILANCIA DE NINST

#INIT      LD          .A, [NINST]
           LD          .B, XFF
           CMPL
           JMPT        #INIT

; LLEGA UNA NUEVA INSTRUCCIÓN POR LÍNEA
; SERIE, DECODIFICADOR

           LD          .ACC, X00
           WR          NINST
           LD          .A, [RCBUF0]
           LD          .B, A
           CMPE
           JMPT        #LEVER
           LD          .B, I
           CMPE
           JMPT        #SWITCH
           LD          .B, T
           CMPE
           JMPT        #TEMP
           LD          .B, S
           CMPE
           JMPT        #SND
           JMP         #ERR

; COMPROBACION DEL DECODIFICADOR

#OUT       LD          .ACC, O
           WR          TXBUF0
           LD          .ACC, K
           WR          TXBUF1
           SEND
           JMP         #INIT
```

```
; RUTINA DE ATENCIÓN A LOS SWITCHES

#SWITCH    LD         .A, [RCBUF1]
           ASCII2BIN
           LD         .INDEX, .ACC
           LD         .A, .ACC
           LD         .B, X07
           CMPG
           JMPT       #ERR ; debería saltar a ERR
           LD         .A, [RCBUF2]
           ASCII2BIN
           LD         .A, .ACC
           LD         .B, X01
           CMPG
           JMPT       #ERR ; debería saltar a ERR
           WRI        SWBASE
           JMP        #OUT

; RUTINA DE ATENCION A LOS ACTUADORES

#LEVER     LD         .A, [RCBUF1]
           ASCII2BIN
           LD         .A, .ACC
           LD         .INDEX, .ACC
           LD         .B, XFF
           CMPE
           JMPT       #ERR
           LD         .A, [RCBUF2]
           ASCII2BIN
           LD         .A, .ACC
           LD         .B, X09
           CMPG
           JMPT       #ERR
           WRI        LEVBASE
           JMP        #OUT

; RUTINA DE ATENCION AL TERMOSTATO

#TEMP      LD         .A, [RCBUF1]
           ASCII2BIN
           LD         .A, .ACC
           LD         .B, X02
           CMPG
           JMPT       #ERR
           LD         .B, X00
           ADD
           SHIFTL
           SHIFTL
           SHIFTL
           SHIFTL
           WR         TMP
           LD         .A, [RCBUF2]
           ASCII2BIN
           LD         .A, .ACC
           LD         .B, XFF
           CMPE
           JMPT       #ERR
           LD         .B, [TMP]
           ADD
           WR         TSTAT
           JMP        #OUT

; ENVÍO DE INFORMACIÓN

#SND       LD         .A, [RCBUF1]
           LD         .B, A
           CMPE
           JMPT       #LEVSND
           LD         .B, I
           CMPE
           JMPT       #SWSND
```

```
          LD        .B, T
          CMPE
          JMPT      #TSND
          JMP       #ERR

; ENVIANDO INFO DE UN ACTUADOR

#LEVSND   LD        .A, [RCBUF2]
          ASCII2BIN
          LD        .A, .ACC
          LD        .INDEX, .ACC
          LD        .B, X09
          CMPG
          JMPT      #ERR
          LDI       .A, [LEVBASE]
          BIN2ASCII
          WR        TXBUF1
          LD        .ACC, A
          WR        TXBUF0
          SEND
          JMP       #INIT

; ENVIANDO INFO DE UN SWITCH

#SWSND    LD        .A, [RCBUF2]
          ASCII2BIN
          LD        .A, .ACC
          LD        .INDEX, .ACC
          LD        .B, X07
          CMPG
          JMPT      #ERR
          LDI       .A, [SWBASE]
          BIN2ASCII
          WR        TXBUF1
          LD        .ACC, S
          WR        TXBUF0
          SEND
          JMP       #INIT

; ENVIANDO INFO DEL TERMOSTATO

#TSND     LD        .A, [TSTAT]
          LD        .B, B11110000
          AND
          SHIFTR
          SHIFTR
          SHIFTR
          SHIFTR
          LD        .A, .ACC
          BIN2ASCII
          WR        TXBUF0
          LD        .A, [TSTAT]
          LD        .B, B00001111
          AND
          LD        .A, .ACC
          BIN2ASCII
          WR        TXBUF1
          SEND
          JMP       #INIT

#ERR      LD        .ACC, E
          WR        TXBUF0
          LD        .ACC, R
          WR        TXBUF1
          SEND
          JMP       #INIT
```