ENLACE ENTRE LA ELECTRONICA DIGITAL Y LOS MICROPROCESADORES

J. VALVERDE, J.M. GARCIA
Departamento de Electrónica e Ingeniería Electromecánica. Universidad de Extremadura. España.

En este trabajo se muestra un enlace entre la electrónica digital clásica, cableada, y los sistemas digitales programables, tipo microprocesador. Se pretende que los alumnos vean como los conocimientos adquiridos en las asignaturas de electrónica digital, se pueden utilizar para la realización de un sistema de aplicación general.

1. Introducción

De forma general tanto las asignaturas denominadas, Electrónica Digital o Teoría de Conmutación, así como su bibliografía, terminan con en el estudio de los sistemas secuenciales síncronos, máquinas de estados, dispositivos lógicos programables etc. O bien con un tema de introducción a los microprocesadores o procesadores en general, en el que se ven conceptos como: buses, CPU, programa almacenado etc. Todo esto desde un punto de vista general y en la mayoría de los casos, desligado de los conocimientos de los temas anteriores, maquinas de estado, sumadores, contadores etc. En las asignaturas correspondientes a procesadores o microprocesadores, se comienza con el estudio de sus estructuras, desde el punto de vista funcional y no estructural. Como consecuencia el alumno, en la mayoría de los casos, no ve o la relación existente entre la Electrónica Digital y los sistemas programables, y en consecuencia, como a partir de los conocimientos adquiridos, en la asignatura de Electrónica Digital, se puede diseñar un sistema de estas características.

2. Realización

Detectado el problema se pensó en la realización, desde el punto de vista de la Electrónica Digital, el diseño de un sistema microprocesador sencillo, pero que a la vez contuviera los elementos mínimos necesarios para que, tanto su estructura como su capacidad de proceso, no se alejaran de un sistema real d y además que los alumnos pudiesen simular el diseño realizado mediante un programa de libre distribución.

Con estos planteamientos se optó por lo siguiente: Utilización del programa de simulación, de libre distribución, el mismo utilizado en las prácticas de la asignatura el TKGate 1.8, y un procesador del tipo RISC, el propuesto por XILINX en “The Practical XILINX Designer Lab Book”, con el nombre de GNOME, cuyas características básicas son las siguientes: operación con datos de 4 bits, 16 registros de datos, juego de 12 instrucciones espacio de direccionamiento de 128 posiciones.

Antes de comenzar con el diseño del sistema propiamente dicho, se debe hacer ver al alumno que los registros pueden servir, además de como elementos capaces de almacenar un determinado estado interno, realización de FSM, contadores o registros de desplazamiento, el de almacenar un dato, haciendo una clara distinción entre lo que es una señal de control del sistema y lo que es un dato sobre el que opera el sistema. Se estudian algunas estructuras simples de realización de transferencias con los datos de los registros y la utilización de registros operativos, y en concreto la estructura alu-acumulador, tal y como se puede observar en la figura 1.

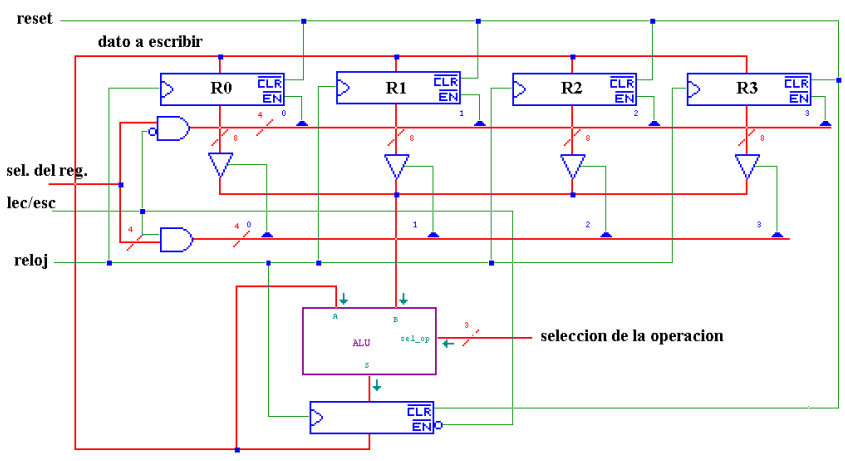

Figura 1 estructura típica alu-acumulador

A partir de la estructura de la figura 1 se puede hacer ver al alumno dos modificaciones importantes:

a) las señales pueden generarse de forma secuencial a partir de una FSM
b) los registros pueden sustituirse por una memoria

Como se puede observar la ALU contiene además de las entradas A y B y la salida S de datos un conjunto de entradas de selección de la operación, si no se quiere modificar los datos que llegan del registro seleccionado bastará con seleccionar la operación de S=B. La entrada A de la ALU se encuentra conectada a la salida del registro auxiliar, el acumulador.

La estructura de la figura 1 es muy flexible y permite una gran variedad de operaciones, por ejemplo si se desea sumar los contenidos de los registros R0 y R2 y el resultado almacenarlo en R1 habría que realizar las siguientes operaciones:

1º escribir en el acc el contenido de R0
2º sumar el contenido de R2
3º guardar el contenido del acc en R1

Para realizar estas operaciones habría que hacer:

1º sel/reg =0001, lec/esc =1, sel_op= “S=B”
2º sel/reg =0100, lec/esc =1, sel_op= “S=A+B”
3º sel/reg =0010, lec/esc =0, sel_op=”XXX”

Sería interesante poder disponer de forma automática de los valores de sel/reg, lec/esc, sel_op por ejemplo en una unidad de memoria, de tal forma que en la primera posición, estuviesen escritos los valores correspondientes a la primera operación en la segunda la siguiente y así sucesivamente, en este caso tan solo haría falta un contador que hiciera las veces de puntero de direcciones de la memoria y que se fuese incrementando de forma adecuada para indicar la dirección de la operación a realizar. A partir de la estructura de la figura 1, se pueden diseñar sistemas digitales que operen con datos sin más que añadir una máquina de estados finita que suministre las señales oportunas a los registros. Como se vio en el apartado anterior, las acciones que deben realizarse pueden estar grabadas en una memoria, de tal forma que mediante un contador irlo incrementando de forma oportuna para que vaya de forma secuencial recorriendo todas las operaciones que deben realizarse, por lo tanto, se puede pensar en un sistema como el de la figura 2.

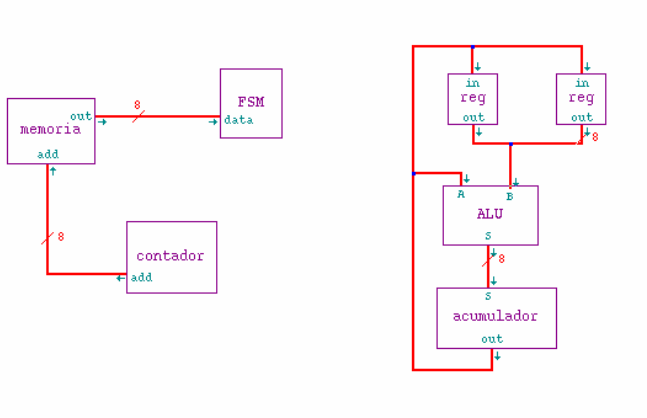

Figura 2 primera aproximación del sistema

Como se puede observar la parte de la derecha de la figura 2 corresponde al esquema de la figura 1, mientras que la parte de la derecha es la encargada de encontrar las acciones que hay que realizar sobre los datos.

Se puede pensar en eliminar los registros que aparecen en la figura 2 ya que los datos que ellos contienen pueden estar almacenados en la unidad de memoria, en este caso se haría necesario que la memoria fuese del tipo RAM, lectura-escritura, o bien dos unidades una de lectura para almacenar las instrucciones a realizar y otra de lectura escritura para sustituir a los registros. Esto implica una mayor complejidad en la FSM ya que deberá acceder a los datos que se encuentran en la memoria por lo que de alguna manera deberá incluirse en el código de la operación, donde se encuentran los datos, para ir a buscarlos o donde se deben guardarse tras una operación, por lo tanto deberá existir un registro que almacene la operación, durante todo el tiempo que dure su ejecución. A este registro se le denomina normalmente “registro de instrucciones”, por otro lado, además la dirección con la que se va a acceder a la memoria dependerá por un lado de si lo que se quiere encontrar una dirección o se va a buscar o almacenar un dato.

Las instrucciones que deberá interpretar la FSM, que estarán almacenadas en la memoria, deben tener un formato como el mostrado a continuación:

operacion | direccion

En el campo destinado a la operación se indicará por ejemplo sumar, restar, leer, etc., mientras que, en el campo de la dirección se indicará donde se encuentra el dato o donde se debe almacenar el dato, por lo tanto se puede modificar el esquema de la figura 2 como se muestra en la figura 3.

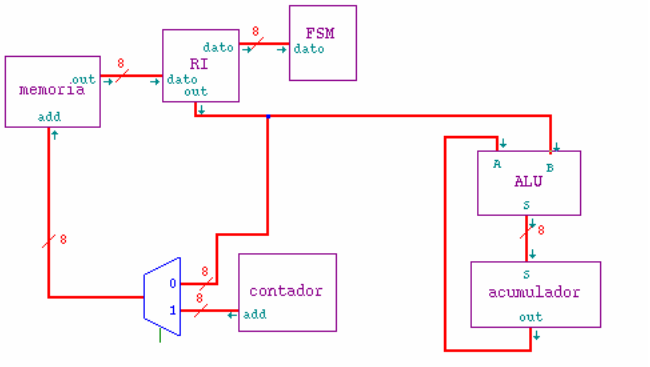

Figura 3 Esquema general del sistema

Observando el esquema de la figura 3, la entrada B de la ALU es suministrada por la memoria, a través del registro de instrucciones, con lo que una instrucción puede incluir además de la operación a realizar, el propio dato en lugar de la dirección en que se encuentre

De todo lo dicho anteriormente, se desprende que las operaciones que se quieran realizar deben de almacenarse en una memoria, en el mismo orden en que se quiere que se ejecuten, pero esto supone una rigidez muy grande. Por lo tanto, sería conveniente el poder realizar bucles o bifurcaciones en función de los datos obtenidos, en definitiva, poder modificar el contenido del contador que suministra las direcciones en que se encuentra la operación que se debe ejecutar.

Una vez justificada la estructura, como paso inicial se imponen una serie de características al sistema, operaciones a realizar, tipo y tamaño de los datos, codificación de las operaciones, en el caso que nos ocupa se han considerado las siguientes operaciones:

operación	**nemónico**	**codificación**
carga el registro en acumulador	load Rd	0100 d3 d2 d1 d0
carga el dato en el acumulador	load_#d	0001 d3 d2 d1 d0
almacena en el registro	store_Rd	0011 d3 d2 d1 d0
suma el contenido del registro	add_Rd	0101 d3 d2 d1 d0
suma el dato especificado	add_#d	0010 d3 d2 d1 d0
operación XOR con el registro	xor_Rd	0110 d3 d2 d1 d0
operación AND con el registro	and_Rd	0111 d3 d2 d1 d0
borra el acarreo	clc	00000000
pone al acarreo a 1	setc	00000001
saltar una instrucción si C=1	skip_c	00000010
saltar una instrucción si Z=1	skip_z	00000011
saltar a la dirección especificada	jump_#a	1 a6 a5 a4 a3 a2 a1 a0

Como se puede observar al realizar esta codificación se ha supuesto lo siguiente:

1º Las posiciones de memoria serán de 8 bits cada una

2º los registros con los que se puede operar vienen especificados por 4 bits, por lo tanto sólo se dispondrá de 16 registros los que van del 0000 al 1111

3º Las posiciones de memoria para almacenar instrucciones en principio serán 127 si se quieren alcanzar todas las posiciones de memoria desde la instrucción jump_#a

Pasaremos a continuación al diseño de cada uno de los bloques:

2.1 ALU

A la vista de las operaciones que se quieren realizar con los datos, se puede diseñar una unidad aritmético lógica, que permita realizar las mismas. Esta unidad deberá ser capaz de: sumar dos datos, realizar las operaciones lógicas AND y XOR, en las operaciones deberá determinar si se ha producido acarreo o la operación ha dado como resultado “0”, así mismo como se debe almacenar en el acumulador el contenido de la línea de datos proveniente del registro de instrucciones sin alterarlo deberá poder pasar el dato a su entrada y modificar, poner a 1 o a 0 el contenido del registro correspondiente al acarreo, cada operación a realizar será determinada por el decodificador de instrucciones, a través de la línea sel_op (selección de la operación), por ejemplo se puede codificar cada operación de la siguiente manera:

PASS	sel_op=001
ADD	sel_op=010
XOR	sel_op=011
AND	sel_op=100
SET_CARRY	sel_op=101
CLR_CARRY	sel_op=110

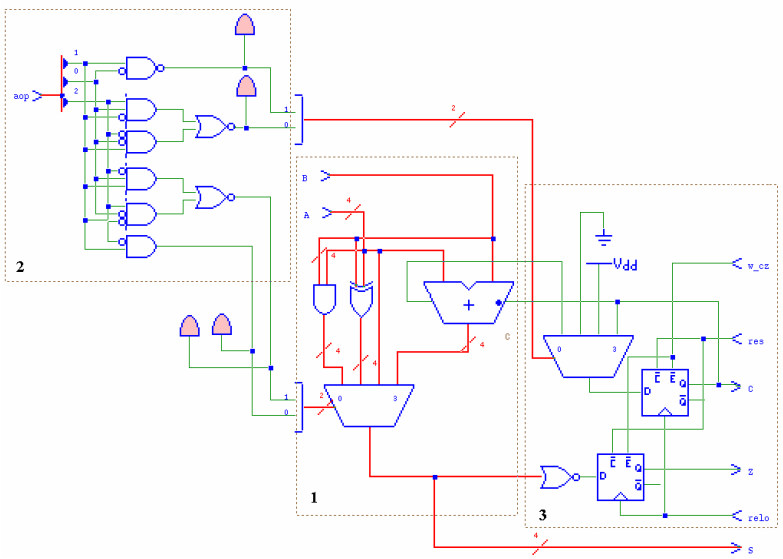

Figura 4 esquema detallado de la ALU

Como se puede observar existen dentro de la ALU tres zonas diferenciadas, la marcada con el número 1 corresponde a la zona de operación con los datos, se puede observar que se realizan las operaciones AND, XOR y SUMA de las entradas A y B, así mismo existe un multiplexor en el que entran los valores de las operaciones anteriores junto a la entrada A de la ALU, para pasar el dato presente a la salida.

La zona 2 corresponde a dos decodificadores, uno de ellos se encarga del gobierno del multiplexor de salida de datos seleccionando, según el código de operación las distintas opciones, AND, XOR, ADD o PASS. Según los códigos de operación sean 101, 100, 011, 010 o 001 respectivamente. El otro se encarga del gobierno del registro del acarreo, según la operación sea ADD, CLR_CARRY, SET_CARRY, u otra, se selecciona la entrada correspondiente al acarrero de salida del sumador, 0, 1, o la propia salida del acarreo, para que permanezca invariable, respectivamente.

La zona 3 corresponde a los registros de acarreo y de cero, estos registros de deben modificar cada vez que se produzca una operación en la ALU, por ello necesitarán una señal que valide la operación, w_cz, suministrada por FSM cuando sea oportuno.

2.2 **ACUMULADOR**

El acumulador es un registro, en este caso de 4 bits que recibe datos de la salida de la ALU, deberá disponer de una entrada de control W_acc, procedente de la FSM que permita, cuando proceda, la escritura del mismo. La figura 5 muestra el esquema del mismo.

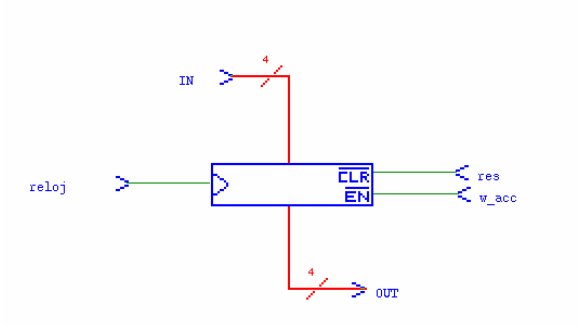

Figura 5 esquema del acumulador

2.3 **REGISTRO DE INSTRUCCIONES**

Como se puede observar en la figura 6 el registro de instrucciones debe almacenar el contenido de la memoria donde se encuentra la instrucción que se está ejecutando pero a la vez debe suministrar el dato con el que operar tanto en las instrucciones. Teniendo en cuenta este funcionamiento el diseño del registro puede obedecer al mostrado en la figura 6, en donde se puede observar que está constituido por dos registros de 4 bits, gobernadas por dos señales externas suministradas por la FSM una que almacena los ocho bits, correspondientes a la memoria donde se encuentra la instrucción, ld_ir, y otra señal que solamente graba los cuatro bits menos significativos. ld_ir_lsn

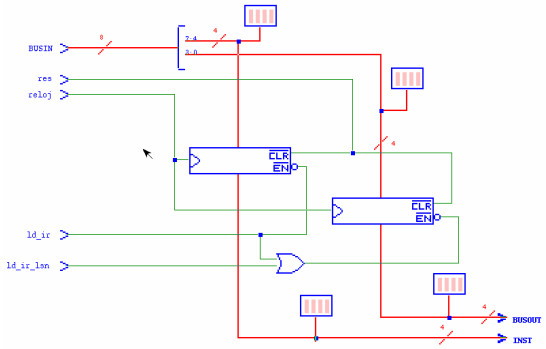

Figura 6

2.4 **CONTADOR DE PROGRAMA**

En este bloque se incluye además del contador de programa, que es el encargado de suministrar la dirección a la memoria de donde se debe leer una instrucción y que debe incrementarse cada vez que se accede a una instrucción, el suministrar la dirección de memoria para escribir o leer un dato. Dado que el direccionamiento de la instrucción jump_#a es de 7 bits se puede destinar en una memoria RAM de 256 posiciones, y por lo tanto 8 bits de direccionamiento, y dentro de ella las primeras 128 posiciones, de 00000000 a 01111111 de acceso a instrucciones y utilizar las 16 últimas posiciones, de 11110000 a 11111111 para colocar los registros operativos, de tal forma que, las posiciones comprendidas entre 10000000 a 11101111 quedan inutilizadas.

El contenido de la dirección por lo tanto vendrá dada por un registro que almacenará la dirección de la siguiente instrucción a ejecutar, bien sea por el funcionamiento normal o, por modificación del mismo, como consecuencia de una instrucción skip o jump, o por la dirección que indique el registro de instrucciones, cuando se haya que acceder a un dato o por.

Por lo tanto se necesitarán unas señales que actúen sobre la salida del circuito, a saber:

Una señal que incremente en 1 el contador de programa con la lectura de cada instrucción o por cumplirse una condición de skip

Una señal que indique que se va a alterar el contenido del contador de programa como consecuencia de una instrucción jump

Una señal que indique que se va a acceder a un registro para leer o escribir en el.

Además deberá tener acceso a los datos o direcciones propias de cada instrucción.

Teniendo en cuenta lo anterior el esquema de contador de programa puede venir dada por la figura 7.

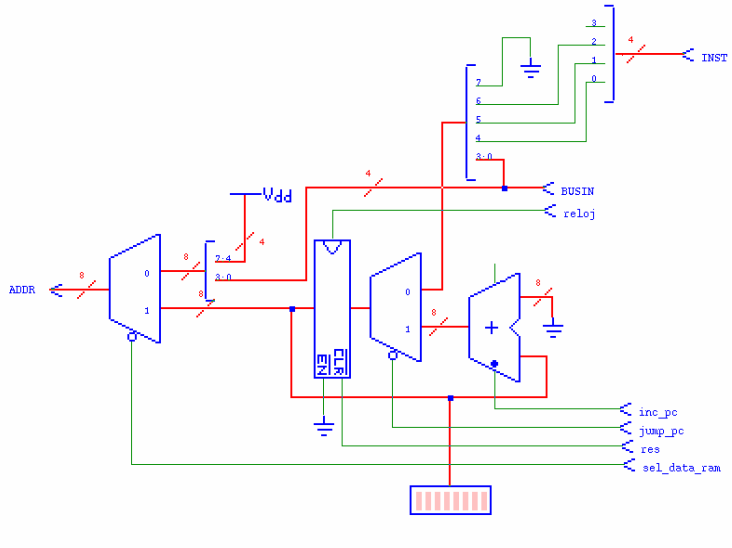

Figura 7 contador de programa

Como se puede observar cuando se activa inc_pc se pone a 1 el acarreo del sumador, por lo que se incrementa en una unidad con el próximo pulso de reloj, esta señal se activará cada vez que se lea una instrucción o se cumpla una instrucción de skip.

Cuando se activa la señal jump_pc se seleccionan los siete bit menos significativos del registro de instrucciones, estando a 0 el más significativo, esta señal se deberá activar cuando se ejecute una instrucción jump_#a.

La señal sel_data_ram se activará cada vez que se ejecute una señal que opere con registros y pondrá la dirección especificada en los cuatro bits menos significativos de la instrucción, BUSIN, con los 4 bits más significativos a 1.

2.5 MEMORIA

Como se ha comentado se utilizará una memoria RAM de 256x8 bits con entrada de “chip select” activa en bajo, que se mantendrá siempre activa. Señal de “ouput enable” activa en bajo que deberá desactivarse cada vez que se escriba en ella. Seal de “write enable” activa en bajo que deberá activarse en un ciclo diferente al de colocar la dirección de acceso y de desactivación del “ouput enable”.

2.6 DECODIFICADOR DE INSTRUCCIONES

Con este nombre se denomina a la máquina de estados, junto a toda la lógica combinacional asociada a la misma, para suministrar las señales necesarias, en los momentos oportunos, para que se ejecuten las acciones que se quieren realizar.

Conviene hacer las siguientes consideraciones sobre el funcionamiento del mismo:

1º El sistema deberá comenzar colocando el contenido del contador de programa, en los terminales de dirección de la memoria, para acceder a la instrucción que se debe ejecutar.

2º Dependiendo de la instrucción que se haya leído desde la memoria podrán ocurrir dos casos: Que el dato, con el que hay que operar se encuentre en la memoria exterior o que no, en el primer caso, se deberá buscar el dato, nótese que en el resto de las operaciones; o el dato, o dirección, va incluido en la propia instrucción, o va implícito en la misma instrucción caso de o no operan con ninguna dirección o dato

3º Una vez que se ha determinado la instrucción a ejecutar y, se dispone de los datos necesarios, se deberán desencadenar las acciones para que se lleve a lugar.

Estas tres consideraciones anteriores llevan a los tres pasos fundamentales que se deben ejecutar para llevar a cabo la tarea especificada en la instrucción y son denominadas:

Busqueda (Fetch) → Decodificación (Decode) → Ejecución (Execute)

Por lo tanto, lo primero que se debe diseñar es una máquina de estados que realice esta secuencia, hay que tener en cuenta que cada una de estas tres fases, no tiene por que ser un solo estado, por ejemplo se podría haber realizado una codificación con instrucciones de distinta longitud, con lo que cada operación podría contenerse en más de una posición de memoria o, por ejemplo como es el caso que nos ocupa se va a añadir un estado más para la instrucción store_Rd, que permita la escritura en la memoria, debido a las particularidades de la memoria seleccionada.

Así en primer lugar, se diseñará una maquina de estados que evolucione tal y como se especifica en la figura 8.

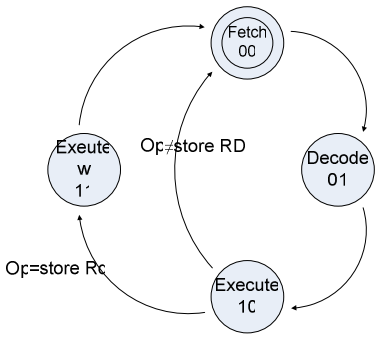

Figura 8 flujograma de la maquina de estados

El circuito obtenido se muestra en la figura 9.

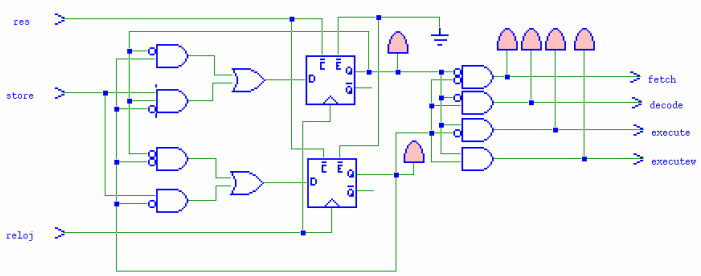

Figura 9 maquina de estados

Como se puede observar, se necesita la señal que identifique cuando la operación a realizar es store_Rd. Al igual que, se debe identificar esta operación será, necesario el hacerlo todas las

instrucciones para en cada caso realizar las operaciones oportunas, por lo tanto se hace necesario un decodificador, que tiene a la entrada la instrucción y una salida activa para cada tipo de instrucción. Como se puede observar en la figura 10 se dispone de dos entradas denominadas BUSIN y INST que corresponden a los 4 bits menos significativos y a los 4 bits mas significativos respectivamente, de la instrucción almacenada en el registro de instrucciones y que corresponden a los 8 bits de la posición de memoria, donde se encuentra la instrucción a ejecutar.

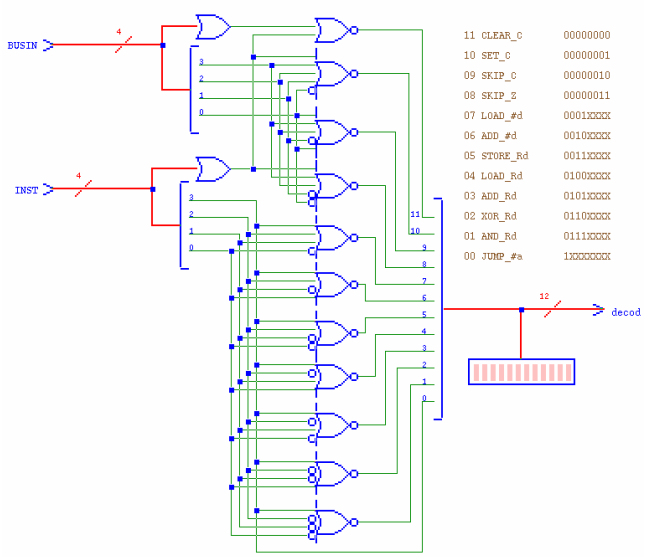

Figura 10 decodificador

Las señales a controlar serán por lo tanto:

sel_data_ram. Controla el multiplexor que ataca al bus de direcciones de la memoria

jump_pc . Controla el multiplexor que ataca a la entrada del registro del contador de programa

inc_pc Es la encargada de incrementar el contador de programa en una unidad

aop Compuesta por tres líneas que son las encargadas de suministrar la operación que debe realizar la ALU

$\overline{WE}$ Es la encargada de escribir en la memoria,

$\overline{OE}$ Es la encargada de habilitar la lectura en la memoria RAM debe estar a 0 un cliclo de reloj antes de la escritura

ld_ir Controla la escritura en los 8 bits del registro de instrucciones, debe activarse en la fase de “fetch”

ld_ir_lsn Carga los cuatro bits menos significativos de el registro de instrucciones en la fase de “decode” de las señales que buscan un dato en el registro.

w_acc Se activa cuando se deba modificar el contenido del acumulador, es decir cada vez que se ejecute una operación en la ALU, es decir: load_Rn, add_Rn, xor_Rn, and_Rn, load_#d, add_#d, en la fase de “execute”

w_cz Es la encargada de almacenar los resultados internos del acarreo y de la coincidencia con 0 del resultado de la ALU, debe actuar además de en los casos de w_acc cuando de ejecuten instrucciones de skip_c y skip_z.

El esquema general del sistema es el mostrado en la figura 11

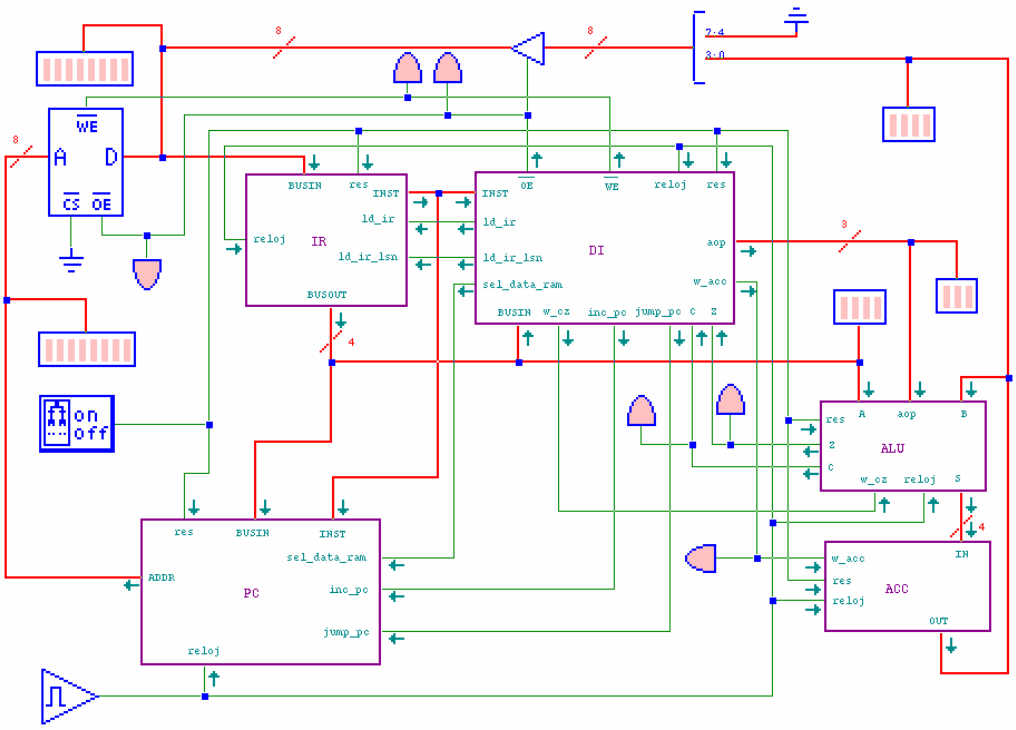

Figura 11 esquema general del sistema

El sistema diseñado anteriormente es de aplicación general y permite la realización de aplicaciones diversas.

3. Conclusiones

Con el presente trabajo se pretende plantear un tema para enlazar la Electrónica Digital con la Estructura de Microprocesadores de tal forma que conociendo los principios de funcionamiento a bajo nivel de un procesador, los alumnos sean capaces de comprender, de forma adecuada, su funcionamiento.