PRACTICA 6 DE FUNDAMENTO DE LOS COMPUTADORES

JR

Conceptos previos:

La instrucción JR causa que el PC salte al contenido del primer registro de la fuente. Ej: JR $13

Al tener que saltar al contenido del primer registro, hemos de determinar en primer lugar cuál es dicho primer registro. En la arquitectura MIPS, las instrucciones que requieren de registros vienen codificadas de forma que se tienen tres posibles direcciones en la instrucción: $RS (source), $RT (target) y $RD (destiny).

Atendiendo al formato de instrucción del JR, *JR $RS*, hemos de usar por tanto el contenido del registro RS y enviarlo al PC.

En el proyecto DELUXE, que es un modelo básico de CPU MIPS, podemos acceder al valor de dicho registro RS en la señal REGA, que se corresponde con el componente U11 del esquemático.

También tenemos ya creado el multiplexor correspondiente a PCSource (FuentePC en el proyecto), lo cual nos ahorrará problemas a la hora de sumar elementos. Dicho multiplexor tiene la ventaja de disponer de una entrada libre, que aprovecharemos a posteriori.

Por último, al ser una función de tipo R, hay que recordar que este formato de instrucciones se diferencia por un código de función específico para cada 1.

Cambios a realizar en el camino de datos:

Por lo comentado en el párrafo de arriba, hemos de conseguir que se escriba en el PC el contenido del registro A y que éste salte.

Para ello:

1. Eliminamos la salida 0 del multiplexor con código U19. Acto seguido, buscamos la salida del REGA, que vienen indicado como U11, y conectamos un bus [31:0] desde la salida de este registro hasta la entrada 0 que hemos eliminado.
2. Hay que buscar alguna forma de indicar que estamos en nuestra instrucción. El formato R nos ofrece esa posibilidad, pues al diferenciarse las instrucciones por el código de función, se pueden reaprovechar los primeros 6 bits, que no empleamos para nada específico, aparte de indicar la instrucción. Para ello creamos un bus tipo [5:0] de 6 bits en alguna posición que nos resulte cómoda y fácil de ubicar. Lo nombramos como “FUNC” y añadimos a este bus un Terminal tipo Output.

Salimos del bloque del camino de datos (nos pedirá actualizar, cosa que hacemos), y si lo hemos hecho bien, nos ha de aparecer un pin nuevo para conectar.

Cambios a realizar en la MDE:

En la máquina de estados hemos de aplicar las siguientes modificaciones. Vamos a aprovechar un estado ya existentes, el 9, modificando el contenido de este estado nuevo puesto que tenemos que escribir el PC en el momento de salto.

Para ello modificamos las señales señaladas en rojo colocando los valores correspondientes. A continuación añadimos las señales FUNC de 6 bits y Q de 3 bits en la parte superior del esquemático de estados. Esta última nos será de utilidad a la hora de realizar la simulación.

IMPORTANTE: Si añadimos la señal Q en nuestro estado, hay que añadirla en <u>todos y cada uno</u> de los restantes con su consiguiente equivalente binario. P.Ej: S1 ➔ Q<="0001";

Una vez creado el estado, añadimos dos transiciones, una de S1 al nuevo estado, que contendrá los valores "I=0 and FUNC=8", y la otra del nuevo estado al S0.

Modificamos la transición de instrucciones tipo R la dejamos tal que quede indicando "I=0 and FUNC/=8" para que diferencie los dos subtipos de instrucción R.

```
EscrPC<='1';
EscrPCCond<='0';
IoD<='X';
LeerMem<='X';
EscrMem<='0';
EscrIR<='0';
MemaReg<='X';
FuentePC<="11";
ALUOp<="XX";
SelALUB<="XX";
SelALUA<='X';
EscrReg<='0';
RegDest<='X';
Q<="1010";
```

Sintetizamos el nuevo circuito lógico y a simular los estados.

Modificamos el código de instrucciones por este de abajo y a simular.

```
00:8c080024; lw $8, 0x24($0)
04:8c090028; lw $9, 0x28($0)
08:01095020; add $10, $8, $9
0c: 01495020; add $10, $10, $9
10:015f0008; jr $10

24:00000002; constante 2
28:00000001; constante
```

Señales clave de la simulación:

Para este caso las instrucciones clave que tenemos que tener presentes son, aparte del PC, Q y el código de instrucción, las siguientes:

- FUNC: Indica si entramos o no en el JR
- REGA: Indica qué es lo que se va a escribir en el PC
- ESCRPC: Indica si se escribe o no en el PC, debe aparecer activa en el estado 10 (A en hex)

Recomendación del autor: para mayor facilidad de visualización, es mejor poner estas señales en decimal.

JM

Conceptos previos:

La instrucción JM causa que el PC salte a una dirección almacenada en memoria. Ej: JM 0x4. Su código de operación es el 6 (o el que nos den en el examen)

Al ser de formato tipo I, este tipo de funciones vienen dadas por un inmediato y un registro indicado en el código, normalmente el RS.

Atendiendo al formato de instrucción del JM,

PC <- DM[RF[rs] + SignExt(immed16)]

Se ve que hemos de usar por tanto el contenido del registro RS, sumarlo al extendido de signo, y enviarlo al PC y enviarlo al PC.

En el proyecto DELUXE, que es un modelo básico de CPU MIPS, podemos acceder al valor de dicho registro RS en la señal REGA, que se corresponde con el componente U11 del esquemático. El inmediato se almacena en dos ubicaciones, una de ellas es el MDR, que lo lleva al extensor de signo, y la otra es la propia de señal de instrucción. Para mayor comodidad, nos centramos en el MDR, simbolizado por el componente U5.

Al igual que en el JR, también tenemos ya creado el multiplexor correspondiente a PCSource (FuentePC en el proyecto), lo cual nos ahorrará problemas a la hora de sumar elementos. Dicho multiplexor tiene la ventaja de disponer de una entrada libre, que aprovecharemos a posteriori.

Cambios a realizar en el camino de datos:

Por lo comentado en el párrafo de arriba, hemos de conseguir que se escriba en el PC el contenido del inmediato y que salte hacia dicho inmediato. Necesitamos por tanto cargar el inmediato antes de hacer la suma y dar el salto.

Para ello:

1. Eliminamos la entrada 0 del multiplexor con código U19 (el de suma PC). Acto seguido, buscamos la salida del MDR, que viene referenciado por el U5_O31, desde la cual conectamos un cable directo hacia la entrada del 0 en el multiplexor U19.

Nota del autor: Si se preguntan porque hacemos esto, es tan simple como ver qué es lo que entra en el MDR y que es lo que sale. Si se fijan, en el MDR entran los bits [15:0], es decir, el inmediato, y salen en formato [31:0], o sea, a 32 bits. Dicho de otra forma, suponiendo que entra el inmediato 000F, saldrá el número 0000000F, que irá directo al PC y además en su formato correspondiente. Matamos dos pájaros de un tiro y nos ahorramos un estado al no tener que pasar por la ALU general para sumar con 0.

Salimos del bloque del camino de datos (nos pedirá actualizar, cosa que hacemos), y si lo hemos hecho bien, nos ha de aparecer un pin nuevo para conectar.

Cambios a realizar en la MDE:

Para este caso tenemos dos opciones:

1. Reutilizar los estados del LW, es decir S1 y S2, conectando a éste último un tercero S10 que nos permitirá la escritura del PC y ejecutar el salto, del mismo modo que hacíamos

con el JR. Esto nos fuerza a reescribir las transiciones de la ruta S1-S2-S10 para incluir el nuevo CodOP. Es la más eficiente.

2. Copiar dichos estados y añadirlos junto al estado 10, que ahora será el 12, en una nueva rama, de forma que pasamos a tener 3 estados extra (S10-$12). Es la menos eficiente**, pero ayuda a la visualización de la práctica.**

Nota del autor: es recomendable probar ambas soluciones, para una mejor comprensión del reciclaje de estados de cara al examen teórico final.

En la máquina de estados vamos a aplicar la segunda. Creamos 3 estados nuevos en cascada, S10, S11 Y S12. Copiamos las señales de S2 Y S3 y las ponemos en estricto orden en S10 y S11, a continuación, copiamos las señales del estado 9, modificando, al igual que hicimos con el JR, las señales *FuentePC* y *ESCRPC*, tal y como indica la tabla. También añadimos la señal Q, para controlar donde estamos en todo momento en la simulación.

IMPORTANTE: Al igual que antes, si añadimos la señal Q en nuestro estado, hay que añadirla en todos y cada uno de los restantes con su consiguiente equivalente binario. P.Ej: S1 ➔ Q<="0001";

Asimismo, hace falta diferenciar la transición inicial para que el procesador pueda gestionar la instrucción. Conectamos con transiciones los estados en el orden indicado (1-10-11-12-0) y modificamos la transición 1-10, añadiendo el *codOP* como *I=6*.

Por último, sintetizamos, comprobamos que todo esté correcto, modificamos el código de instrucciones por este de abajo y a simular.

```
EscrPC<='1';
EscrPCCond<='0';
IoD<='X';
LeerMem<='X';
EscrMem<='0';
EscrIR<='0';
MemaReg<='X';
FuentePC<="11";
ALUOp<="XX";
SelALUB<="XX";
SelALUA<='X';
EscrReg<='0';
RegDest<='X';
Q<="1100";
```

Señales de S12

```
00:8c100020; lw $16, 0x20($0)
04:8e110020; lw $17, 0x20($16)
08:02118022; sub $16, $16, $17
0c:ae300020; sw $16, 0x20($17)
10:1a000020; jm 0x20($16)

20:00000008; constante 8
24:00000000; espacio reservado
28:00000004; constante 4
```

Señales clave de la simulación:

Para este caso las instrucciones clave que tenemos que tener presentes son, aparte del PC y Q, las siguientes:

- CodOP: Indica si entramos o no en el JM. Recuérdese que el CodOP es el 6, independientemente de que opción de implementación lógica se haya escogido.
- BUS del MDR: Indica qué es lo que se va a escribir en el PC.
- ESCRPC: Indica si se escribe o no en el PC, debe aparecer activa en el estado 12 (C en hex.)

Recomendación del autor: para mayor facilidad de visualización, es mejor poner estas señales en decimal.

ADDI

Conceptos previos:

La instrucción ADDI, realiza la suma de un inmediato cargado en memoria con el contenido de un registro en particular.

Al ser de formato Tipo I, este tipo de funciones vienen dadas por un inmediato y un registro indicado en el código, normalmente el RS.

Veamos el formato de instrucción del ADDI:

IM[PC]
RF[rt] <- RF[rs] + EXT(immed)
PC <- PC + 4

Como se ve, hemos de usar el contenido del registro RS y sumarlo al inmediato extendido de signo. El PC realizará la cuenta +4 automáticamente, sin nuestra intervención.

En el proyecto DELUXE, podemos acceder al valor de dicho registro RS en la señal REGA, que se corresponde con el componente U11 del esquemático. El inmediato se almacena en dos ubicaciones, una de ellas es el MDR, que lo lleva al extensor de signo, y la otra es la propia de señal de instrucción. El registro RT, se corresponde con la entrada RAAB del banco de registros y también con la entrada 0 del multiplexor de REGB. Podemos colocar el marcador en REGB para mayor comodidad. Por último, al ser una operación de escritura, hay que tener en cuenta dónde se almacena, por lo que necesitaremos controlar el RD (que ha ser igual que el RT), y el dato a escribir [1].

[1] WA del Banco de Registros, DATO_esc_RD en la simulación.

Cambios a realizar en el camino de datos:

Por lo comentado en el párrafo de arriba, hemos de conseguir que se escriba en el RD el contenido del inmediato sumado al RS. Necesitamos por tanto cargar el inmediato antes de hacer la suma y luego sumar.

Para este caso no hace falta realizar cambios en la estructura de datos puesto que ya tenemos implementada la carga de inmediatos, su extensión y además están conectadas a la ALU a través del multiplexor del REGB.

Cambios a realizar en la MDE:

Para este caso tenemos dos opciones:

1. Reutilizar los estados del LW, es decir S1 y S2, conectando a éste último un tercero S10 que nos permitirá la escritura del RD, del mismo modo que hacíamos con el JR y JM. Esto nos fuerza a reescribir las transiciones de la ruta S1-S2-S10 para incluir el nuevo CodOP. Es la más eficiente.
2. Copiar dichos estados y añadirlos junto al estado nuevo en una nueva rama, de forma que pasamos a tener 2 estados extra (S10-11). Es la menos eficiente**, pero ayuda a la visualización de la práctica.**

Nota del autor: es recomendable probar ambas soluciones, para una mejor comprensión del reciclaje de estados de cara al examen teórico final.

En la máquina de estados vamos a aplicar la segunda opción. Creamos 2 estados nuevos en cascada, S10 y S11. Copiamos las señales de S2 en s10. A continuación, copiamos las señales del estado 10, modificándolas tal y como indica la tabla. También añadimos por comodidad la señal Q, para controlar donde estamos en todo momento en la simulación.

IMPORTANTE: Al igual que antes, si añadimos la señal Q en nuestro estado, hay que añadirla en todos y cada uno de los restantes con su consiguiente equivalente binario. P.Ej: S1 ➔ Q<="0001";

Asimismo, hace falta diferenciar la transición inicial para que el procesador pueda gestionar la instrucción. Conectamos con transiciones los estados en el orden indicado (1-10-11-0) y modificamos la transición 1-10, añadiendo el código de operación *I=8*.

Por último, sintetizamos, comprobamos que todo esté correcto, modificamos el código de instrucciones por este de abajo y a simular.

```
00:8c100020; lw $16, 0x20($0)
04:8e110020; lw $17, 0x20($16)
08:02314020; add $8, $17, $17
0c:21090001; addi $9, $8, 0x1
10:2129000f; addi $9, $9, 0Xf

20:00000004; constante 4
24:00000008; constante 8
```

```
EscrPC<='0';
EscrPCCond<='0';
IoD<='X';
LeerMem<='X';
EscrMem<='0';
EscrIR<='0';
MemaReg<='0';
FuentePC<="XX";
ALUOp<="00";
SelALUB<="10";
SelALUA<='1';
EscrReg<='1';
RegDest<='0';
Q<="1011";
```

Señales de S11. En azul, mantener estas en el mismo nivel que S10 para que se conserven los resultados hasta escritura.

Señales clave de la simulación:

Para este caso las instrucciones clave que tenemos que tener presentes son, aparte del PC y Q, las siguientes:

- CodOP: Indica si entramos o no en el ADDI. Recuérdese que el CodOP es el 6, independientemente de que opción de implementación lógica se haya escogido.
- RS: Indica el registro fuente desde donde se extrae el primer sumando.
- RT: Indica el registro donde se va a escribir.
- RD: Indica el registro destino de la escritura después de la suma.
- DATO_esc_RD: Indica el dato a escribir en el RD
- REGA: Primer sumando
- REGB: Segundo sumando. **Debe ser el inmediato.**
- ALUOP Y ALUOUT: Indican la suma y salida de ALU que va a escribirse

Nota del autor: No te enfules y revisa e interpreta el set de instrucciones indicado arriba. No importa si lo haces en papel, con Excel, tatuándotelo en los brazos o tallando las instrucciones en piedra pómez. Hazlo como tú quieras pero cerciórate de los resultados finales antes de simular. Te hará la comprensión de la práctica mucho más fácil. Ah, y recuerda ponerlas en decimal.

Recordatorios y consejos:

- A la hora de añadir la transición en la MDE, hay que seleccionar la opción *"State Action".*
- Revisar los cambios realizado en la MDE.
- Analizar el código que nos dan puede ser vital para aprobar.
- Más vale lento y bien, que rápido y mal. Muy recomendable hacer los cambios poco a poco y revisar todo.
- Para no jeringar el examen, guardar los estados de simulación y el archivo de comandos es **importante.** La ruta para que nuestro archivo de comandos modificado se exporte correctamente es *C:\xilinx\active\projects\nombredelproject\ram.*
- Recuerda, guardar es importante.
- La señal Q tiene que añadirse en la parte superior del esquemático de la MDE, estar en formato *Combinational* y con un rango de 3:0, al actualizar nos aparece un pin de salida en nuestro bloque de la MDE.
- Sobre el punto anterior, si tienes que pegare 10 minutos en el examen poniendo las cosas una por una, HAZLO. Mejor lento y bien que rápido y mal.
- El hecho de que este documento diga las señales que hay que tener presente no exime de añadir las restantes a la simulación.
- Guarda y Archiva después de realizar un bloque de cambios es obligatorio. Xilinx tiende a petar por cualquier minucia, desde un cambio en la MDE hasta una recarga de simulación.
- Que narices, guarda después de hacer cualquier cosa, Xilinx es más inestable que un psicótico puesto hasta arriba de coca.