Autores: Angel Cañal Muniesa (NIA: 716205) y Abel Chils Trabanco (NIA: 718997).

En este trabajo se ha modificado el procesador para añadirle las dos nuevas instrucciones, la gestión de riesgos de datos y de controles, así como el apartado opcional de predecir el salto no tomado.

Nuevas instrucciones

Para lograr añadir las nuevas instrucciones, primero se ha modificado el banco de registros para que se pueda guardar en dos registos en el mismo ciclo. Por otro lado se ha añadido un multiplexor a la entrada de la dirección de la memoria de datos en el cual en las instrucciones con postincremento se selecciona como dirección rs y en las instrucciones sin postincremento se selecciona rt + inmediato. Por otro lado se propaga rs + inmediato así como la dirección de rs a las nuevas entradas del banco de registros para poder hacer el postincremento.
También se ha modificado la uc y los registros intermedios para propagar estas señales nuevas.

Gestión de riesgos de datos y de control

Para lograr gestionar los riesgos de datos y de control, hemos creados el módulo HDM.
Esta unidad detecta conflictos entre la instrucción que se encuentra en la etapa ID y las que se encuentran en la etapa EX y MEM.
Su función es notificar, al detectar un conflicto, la anticipación de operandos de las etapas posteriores o el bloqueo de la instrucción en etapa ID (evitar la entrada de nuevas instrucciones e introducir una instrucción-burbuja).
Dos multiplexores, uno a la salida del bus a y otro a la salida del bus b, los cuales permiten que se realicen los cortos (ambos reciben tanto la salida del banco de registros, la salida de la alu en la fase de EX, la salida de la alu de la fase anterior de la fase de MEM y la salida del banco de datos), ya que si el módulo HDM indica que hay que tomar un corto propagan este en vez de la salida del banco de registro. Y para poder obtener los registros y instrucción usadas en otras etapas se han modificado los registros intermedios, para poder propagar y saber en las etapas necesarias que instrucción se está ejecutando así como que registros usa.
Por otro lado se ha hecho que la señal de carga de PC y el registro IF_ID sea el negado de la señal de stop que lanza el módulo HDM, para de esta forma parar el procesador en el caso de que sea necesario, y se ha añadido un multiplexor que va tanto a la entrada de la unidad de control como al banco que separa la etapa de ID y EX de tal forma que si HDM manda stop la instrucción que se propaga es una nop y las señales de instrucción que salen de la unidad de control son las correspondientes a una nop.

Proporciona una ventaja clave para el procesador, reduce el número de instrucciones ejecutadas para realizar la misma tarea, ya que elimina la necesidad de añadir NOP.

Predicción de salto no tomado

Se ha añadido un multiplexor a la entrada de IR para tomar la instrucción leída o inyectar una NOP en el caso de que haya que tomar un salto, retrasando así 1 ciclo el procesador.

El procesador carga siempre la instrucción siguiente (predicción de salto no tomado), y cuando se resuelve el salto se actúa si es necesario anulando la instrucción anterior (la que se encuentra en IF).
La ventaja de esta decisión radica en códigos con muchos saltos no tomados, ya que no se desaprovecha el tiempo hasta que se resuelve el salto (1 ciclo) si no es tomado. La contra ocurre al tomar un salto, que obliga a actuar sobre la instrucción cargada erróneamente.

Pruebas realizadas
Para probar el correcto funcionamiento del procesador hemos creado los siguientes códigos de prueba los cuales incluimos en la carpeta bancos de prueba en las cuales se han comprobado todos los cortos posibles, incluyendo las paradas y la predicción de salto.
El código en ensamblador de estas es:

Prueba nº1

```
sw_pos r1, 1(r1)
sw_pos r1, 1(r1)
add r1, r1, r1
add r1, r1, r1
add r1, r1, r1
lw r1, 1(r1)
lw_pos r0, 1(r1)
lw_pos r0, 1(r1)
lw r1, 0(r1)
lw r1, 0(r0)
lw_pos r0, 1(r1)
lw r1, 0(r0)
sw_pos r0, 1(r1)
sw_pos r0, 1(r1)
lw r0, 0(r1)
```

Prueba nº2

```
lw r1, 0(r0)
beq r1, r0, 0
beq r1, r0, 0
lw_pos r1, 4(r0)
beq r0, r1, 0
beq r0, r1, 0
```

Prueba nº3

```
lw r1, 0(r0)
nop
sw r1, 8(r0)
lw r1, 4(r0)
sw r1, 4(r1)
lw_pos r2, 1(r3)
nop
```

```
    sw r2, 0(r3)
    lw_pos r2, 1(r3)
    sw r2, 0(r3)
```

Prueba nº4

```
    add r1, r1, r1
    nop
    lw_pos r0, 1(r1)
    add r1, r1, r1
    lw_pos r0, 2(r1)
    lw r0, 4(r3)
    lw_pos r3, 4(r0)
    sw_pos r4, 3(r4)
    nop
    lw_pos r3, 3(r4)
    sw_pos r0, 2(r1)
    lw_pos r0, 3(r1)
```

Prueba nº5

```
    lw r0, 0(r1)
    sw_pos r0, 3(r0)
    sw_pos r0, 3(r0)
    nop
    sw_pos r0, 3(r0)
    lw_pos r0, 2(r1)
    nop
    sw_pos r0, 2(r3)
    lw_pos r0, 2(r1)
    nop
    sw_pos r0, 3(r0)
```

Prueba nº6

```
    lw r1, 0(r0)
    lw r3, 4(r4)
b:  add r2, r2, r1
    add r0, r0, r3
    beq r0, r1, f
    beq r0, r0, b
f:  sw r2, 0(r1)
```

Almacena en una dirección leída de la posición 0x0 de la RAM el cuadrado de esa dirección por 4.

Prueba nº7

```
    lw_pos r1, r(r0)
    lw_pos r3, 4(r0)
```

```
b:  lw r4, 0(r0)
    add r2, r2, r4
    add r0, r0, r3
    beq r0, r1, f
    beq r0, r0, b
f:  sw r2, 0(r1)
```

Calcula la suma del vector cuya primera componente se encuentra en la posición 0x8 de memoria,
y ocupa tantos bytes como se lean de la posición 0x0 de la RAM. Esta suma la guarda en la palabra
que se encuentra a continuación del vector.

Prueba nº8

```
    lw_pos r1, 4(r0)
b:  lw_pos r3, 4(r0)
    add r2, r2, r3
    beq r0, r1, f
    beq r0, r0, b
f:  sw_pos r2, 0(r0)
```

Realiza lo mismo que el código anterior, pero aprovechando la instrucción nueva.

Prueba nº9

```
    lw_pos r1, 4(r0)
    lw_pos r2, 4(r0)
    lw_pos r3, 4(r0)
    add r4, r2, r3
b:  lw_pos r5, 4(r1)
    sw_pos r5, 4(r2)
    beq r2, r4, f
    beq r2, r2, b
f:  sw r2, 0(r0)
```

Copia un vector v (en la posición 0x0 de la RAM hay un puntero a dicho vector) a
un segundo vector w (su puntero se encuentra en la segunda palabra de la RAM).

Reparto del trabajo:

Primeramente los dos integrantes del grupo realizamos el diseño del procesador sobre papel, una vez que ya se había diseñado Ángel se dedicó a realizar el módulo HDM y Abel se dedicó a modificar y cablear el módulo Mips_segmentado_2017 así como de modificar el módulo UC_Mips.
Las pruebas y la memoria se realizaron entre los dos.
El tiempo total dedicado al trabajo ha sido de 22.5 horas.
Al diseño en papel se dedicaron 3 horas.

A la implementación se dedicaron 8 horas (durante la implementación se fueron realizando pruebas).
Al diseño de pruebas y su comprobación se dedicaron 8 horas.
A la corrección de errores se dedicó 1 hora.
A la realización de la memoria se dedicó 2 horas y media.