MIPS SEGMENTADO Y GESTIÓN DE RIESGOS

ARQUITECTURA Y ORGANIZACIÓN DE COMPUTADORES 2

DANIEL RUEDA MACÍAS

NIP: 559207

Universidad Zaragoza

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

Índice

RESUMEN

En este proyecto se ha optimizado un MIPS segmentado añadiéndole instrucciones preincremento, paradas en la detección de riesgos de datos y un predictor de saltos.

Para las instrucciones preincremento, se ha añadido un nuevo puerto de escritura junto con una señal de control independiente a la que ya tenía; para las paradas de detección de riesgos de datos, se ha condicionado las señales de control del PC y del banco que separa las etapas de F y D si se detectaba alguno de estos riesgos; y por último, a la hora de añadir el detector de saltos, se ha ampliado el mux del PC a un mux de cuatro entradas con dos bits de control y se ha añadido el predictor correspondiente conectándole como entradas la información del último salto realizado y como salidas la información del salto predicho.

Todo esto con el fin de reducir los CPIs y en consecuencia, el tiempo de ejecución del procesador para así poder obtener uno mucho más eficiente.

INTRODUCCIÓN

Se nos ha proporcionado un MIPS segmentado de 32 bits como el visto en clase sin anticipación de operandos, que resuelve los saltos en la etapa D y escribe en el banco de registros en flanco de bajada. Todo descrito mediante VHDL, un lenguaje de descripción de hardware y ModelSim, un entorno de simulación basado en bancos de prueba y cronogramas.

OBJETIVOS

Se quiere reducir los CPIs y el número de instrucciones del MIPS, mediante el añadido de instrucciones preincremento, un módulo que detecte los riesgos de datos realizando las paradas necesarias y un predictor de saltos que diga si se va a saltar o no en la etapa F según lo que se ha hecho en el salto anterior.

METODOLOGÍA

LW y SW con preincremento

Para la realización de estas instrucciones preincremento, lo primero que se ha hecho ha sido modificar el banco de registros, añadiéndole un nuevo puerto de escritura, junto con un bus y una señal de control.

Lo segundo ha sido propagar la señal correspondiente al registro RS hasta la etapa MEM. En dicha etapa, se ha conectado la señal ALU_out_MEM, que es la que contiene la salida de la ALU, al nuevo bus del banco de registros. Lo mismo ha sucedido con RS, en dicha etapa, se ha conectado al nuevo puerto de escritura del banco.

Por último pero no menos importante, se ha añadido una nueva señal a la unidad de control, que es la que corresponde a la escritura en el banco de registros de las instrucciones preincremento, poniéndola a 1 en la unidad de control sólo cuando el código de operación sea el correspondiente a las instrucciones LW_pre o SW_pre.

Gestión de riesgos de datos

Mediante distintos ejemplos de código pensados en papel, se ha llegado a la detección de los siguientes riesgos de datos.

- Riesgos de datos de RS
    - Que en la etapa EX el futuro registro que vaya a escribir en el banco (RW_EX) coincida con el nuestro en D y que se vaya a realizar la escritura (RegWrite_EX = 1).
    - Que en la etapa MEM el futuro registro que vaya a escribir en el banco (RW_MEM) coincida con el nuestro en D y que se vaya a realizar la escritura (RegWrite_MEM = 1).
    - Que en la etapa EX el registro que vaya a escribir en el banco en modo preincremento (RS_EX) coincida con el nuestro y se vaya a realizar la escritura (Update_Rs_EX = 1).
- Riesgos de datos para RT
    - Que en la etapa EX el futuro registro que vaya a escribir en el banco (RW_EX) coincida con el nuestro en D, que se vaya a realizar la escritura (RegWrite_EX = 1) y que la instrucción que se encuentra en D no sea un LW o un LW_pre.

- Que en la etapa MEM el futuro registro que vaya a escribir en el banco (RW_MEM) coincida con el nuestro en D, que se vaya a realizar la escritura (RegWrite_MEM = 1) y que la instrucción que se encuentra en D no sea un LW o un LW_pre.
- Que en la etapa EX el registro que vaya a escribir en el banco en modo preincremento (RS_EX) coincida con el nuestro, que se vaya a realizar la escritura (Update_Rs_EX = 1) y que la instrucción que se encuentra en D no sea un LW o un LW_pre.

Estos riesgos se deben considerar para todas las instrucciones excepto para una NOP, una NOP no hace nada, aunque tenga operandos, por lo tanto no debemos hacer parada. En cuanto al comentario que se ha realizado respecto a que la instrucción no sea un LW o un LW_pre en los riesgos de RT, esto se debe a que en un LW en RT se carga el dato de memoria, no se lee de él, por lo tanto no tenemos que parar.

Gestión de riesgos de control

Lo primero que se hizo en esta última parte para gestionar los riesgos de control añadiendo un predictor, fue crear un nuevo fichero en el que se define el comportamiento de este.

Implementación del predictor

El predictor tiene como señales de entrada clk; reset; PC4, que corresponde al PC+4 en la etapa F; PC4_ID, que corresponde al PC+4 en la etapa D; branch_address_in, la cual es la dirección de salto calculada en la etapa D; prediction_in el cual es el bit de salto que indica si se ha saltado o no se ha saltado y update, que es el bit que indica si se debe actualizar el predictor o no.

Como señales de salida tiene branch_address_out y prediction_out que representan la predicción del predictor siendo la dirección y el bit de salto respectivamente.

Siguiendo por el comportamiento del predictor, se han definido las señales que representarán la memoria interna de este. Estas señales son:

- etiqueta: Nos permite identificar la instrucción de salto almacenada, se almacena como etiqueta PC4_ID.
- dirSalto: Se almacena la dirección de salto completa.
- prediccion: Indica que ocurrió la última vez, 0 de saltó y 1 no se saltó.
- validez: Indica si la información que contiene el registro es válida.

A continuación se ha definido el comportamiento mediante dos procesos que se ejecutan concurrentemente.

El primer proceso es el que se encarga de actualizar el registro. Tiene como lista de sensibilidad el clk. Si llega un evento de clk y este es igual a 1, entonces se mira si la señal reset está activada. En caso afirmativo, las señales del registro se ponen a 0, en caso contrario, se procede a comprobar si la señal de update está activada para actualizar la información del registro. En caso afirmativo se almacena la nueva información que viene por las señales de entrada.

El segundo proceso es el que se encarga de realizar de comparador, tiene como lista de sensibilidad las señales PC4, validez y prediccion. En caso de que alguna de estas cambie, se procederá a realizar la comparación. Si PC4 coincide con la etiqueta del registro, el dato del registro es válido, es decir, está a 1 y si predicción está a 1 (la última vez se saltó), entonces, se dirá que se salta (prediction_out a 1 y por branch_address out se sacará la dirección de salto almacenada en el registro). En caso contrario no se salta (prediction_out a 0).

Conexión del predictor y gestión de fallos

Una vez con el comportamiento del predictor programado toca conectarlo a nuestro MIPS.

Primero se ha ampliado el mux que conectaba la entrada al PC a un mux de 4 entradas con dos bits de control. En la entrada 0 se ha conectado el PC+4 en la etapa F, en la 1 la dirección de salto del predictor, en la 2 el PC+4 en la etapa D y por último en la entrada 3 la dirección de salto calculada.

En segundo lugar, viene la parte en la que hay que gestionar los bits de control según los riesgos o errores como yo los he llamado, que se den. Existen tres tipos de errores:

1. Que el predictor haya dicho en F que no se salta pero luego a la hora de calcular el salto en D sí se salta.
2. Que el predictor nos haya dicho en F que sí se salta pero luego a la hora de calcular el salto en D no se salta.
3. Que el predictor nos haya dicho en F que sí se salta, que en D al calcular el salto salga el mismo resultado pero luego las direcciones de salto del predictor y la calculada no coincidan.

Una vez con los errores descritos, se asignan los siguientes valores a los bits de control dados los siguientes casos:

- 00 – Si no hay ninguno de los errores citados anteriormente y el predictor dice que no hay que saltar.
- 01 – Si no hay ningún error y el predictor dice que hay que saltar.
- 10 – Si se da el segundo error.
- 11 – Si se da el primer o el tercer error.

En tercer y último lugar, se ha conectado el predictor al procesador. En las señales de entrada se han conectado el clk a clk, reset a reset, PC+4 en F bits 9 a 2 a PC4, PC+4 en D bits de 9 a 2 a PC4_ID, la dirección de salto calculada en ID a branch_address_in en el caso de que se produjera el primer o el tercer error, en cualquier otro caso el PC4_ID, en prediction_in se ha conectado el bit de salto y en update una señal auxiliar que se pone a 1 si el bit de salto y prediction_out son distintos, en cualquier otro caso 0.

En lo que respecta a las señales de salida, se han conectado unos puertos nuevos que se han creado en el banco de registros que separa la etapa F de D para así poder leer sus valores en D.

RESULTADOS

Para probar que el diseño funciona correctamente, se han realizado varios bancos de pruebas. Pero se va a destacar uno que contiene muchas situaciones típicas. El código del banco de pruebas es el siguiente:

```
dir 0   LW R5, 0(R0)        //R0=tamaño tabla

dir 1   LW_pre R30, 20(R2)  //Basura en R30 pero nos posicionamos al final de la tabla para
        empezar a mover (Es como hacer un mov a R2)

dir 2   LW_pre R29, 1(R4)   //Basura en R29 pero metemos un 1 en R4

dir 3   LW_pre R0, 4(R1)    //Cargamos el primer dato y movemos cursor

dir 4   NOP R0, 8(R1)       //Para engañar al procesador a ver si para

dir 5   ADD R3,R3,R4        //Incrementamos el contador

dir 6   SW_pre R0, 4(R2)    //Movemos el dato a otro lugar de memoria y movemos cursor

dir 7   BEQ R3, R5, dir9    //Si hemos recorrido toda la tabla, fin del programa

dir 8   BEQ R0, R0, dir3    //Saltamos al inicio del bucle

dir 9   BEQ R0, R0, dir9    //Bucle infinito de finalización
```

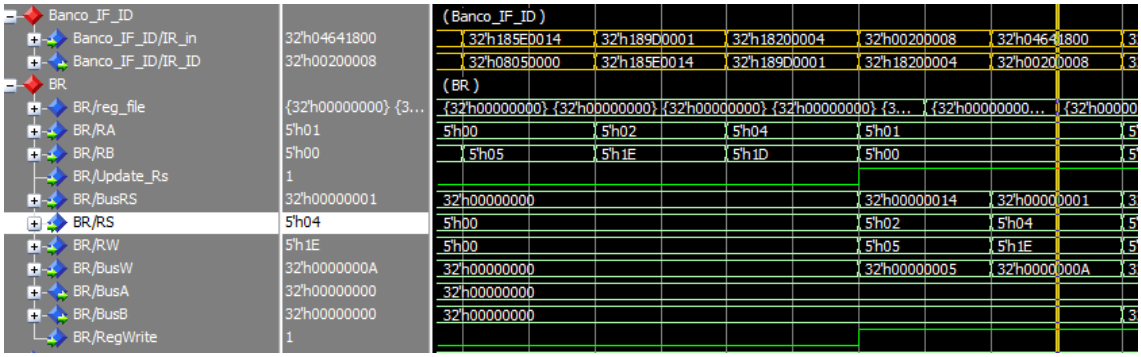
La codificación para las instrucciones es la siguiente: *08050000, 185E0014, 189D0001, 18200004, 00200008, 04641800, 1C400004, 10A30001, 1000FFFA y 1000FFFF* respectivamente.

El funcionamiento de este programa consiste en copiar una tabla de 5 elementos de una zona de memoria a otra y cuando ha terminado quedarse parado en un bucle infinito de finalización. Los elementos de la tabla son 5,4,9,2,A.

Veamos algunos ejemplos del funcionamiento de las mejoras que se han añadido al MIPS según el orden en el que se han ido añadiendo.

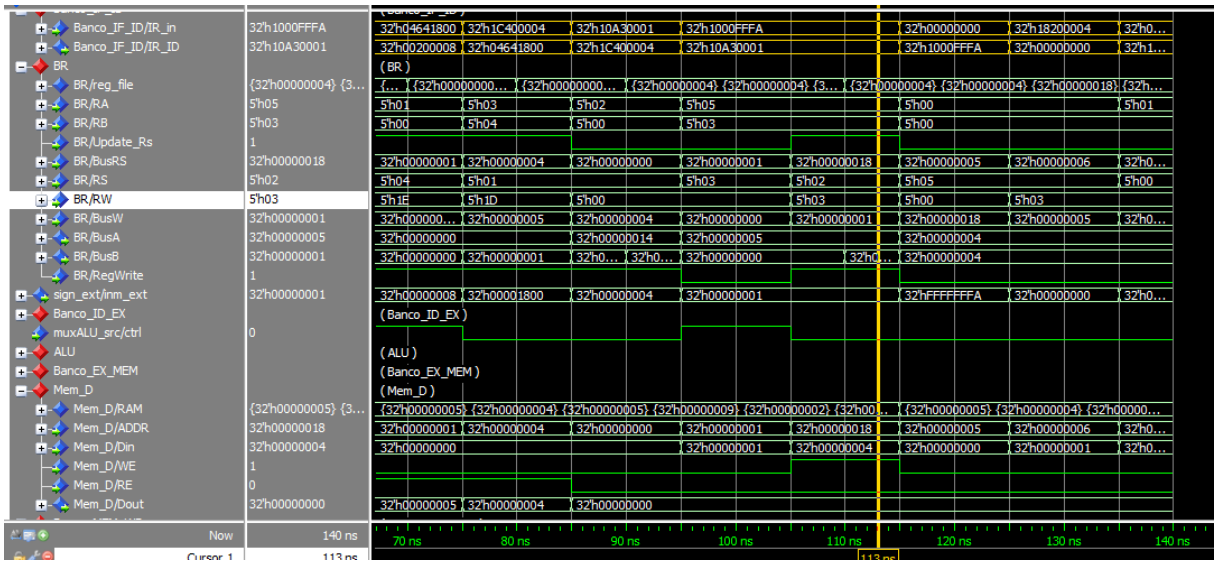
LW y SW preincremento

Como ejemplo de LW pre-incremento se puede poner la instrucción 1 del programa, que se usa a modo de mov (esto solo sirve al principio, ya que están todos los registros a 0). Cargamos basura en r30 pero en r2 cargamos la posición anterior a donde se empezará a escribir la tabla que vamos a mover.



Se puede ver el ciclo en el que instrucción LW (*185E0014)* se encuentra en F. Pasados dos ciclos esta se encuentra en MEM, se puede apreciar la señal Update_Rs a 1, el registro 2 en el puerto de escritura RS y el dato correspondiente en busRS. También en el ciclo siguiente se puede ver que carga correctamente el dato de memoria.

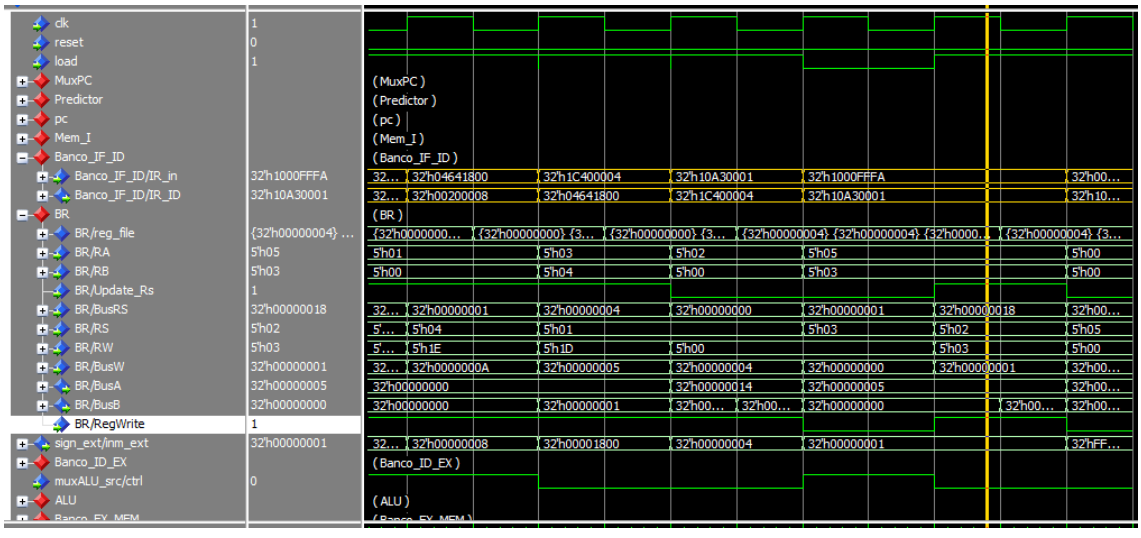
Por último como ejemplo del SW preincremento, se puede poner la instrucción número 6 del programa la primera vez que se ejecuta.



Puede verse cuando nuestra instrucción (1C400004) está en F, pasados dos ciclos se encuentra en MEM. En el mismo ciclo actualiza el registro preincremento (Update_RS=1, busRS=24 y RS=2) y se escribe el dato correspondiente en memoria (WE=1).

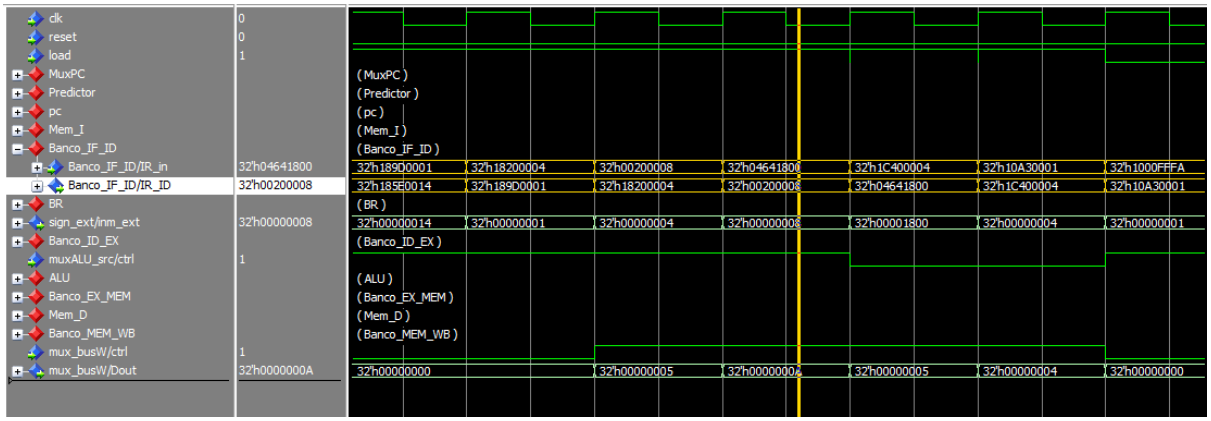
Paradas en riesgos de datos

Como ejemplo de parada al detectar un riesgo de datos, se puede poner la instrucción 7 la primera vez que se ejecuta.



Existe una dependencia de la instrucción 7 con la instrucción 5. Hay que fijarse en el ciclo en que la instrucción 5 (*04641800*) y en el ciclo en que la 7 (*10A30001*) se encuentra en F. En el ciclo siguiente en el que la instrucción 7 se encuentra en D, la instrucción 5 se encuentra en M y por tanto aún no ha escrito el valor del R3 y la instrucción 7 tiene que detenerse a la espera de los operandos hasta que los escriba en el ciclo siguiente. En dicho ciclo, puede verse que la instrucción 5 escribe el valor de R3 y la instrucción 7 ya puede continuar.

Por último, un ejemplo de que el MIPS no hace paradas innecesarias viene dado por la instrucción 4. La instrucción 4 es una instrucción NOP que tiene como operandos los registros R0 y R1 y estos dependen de la instrucción anterior.

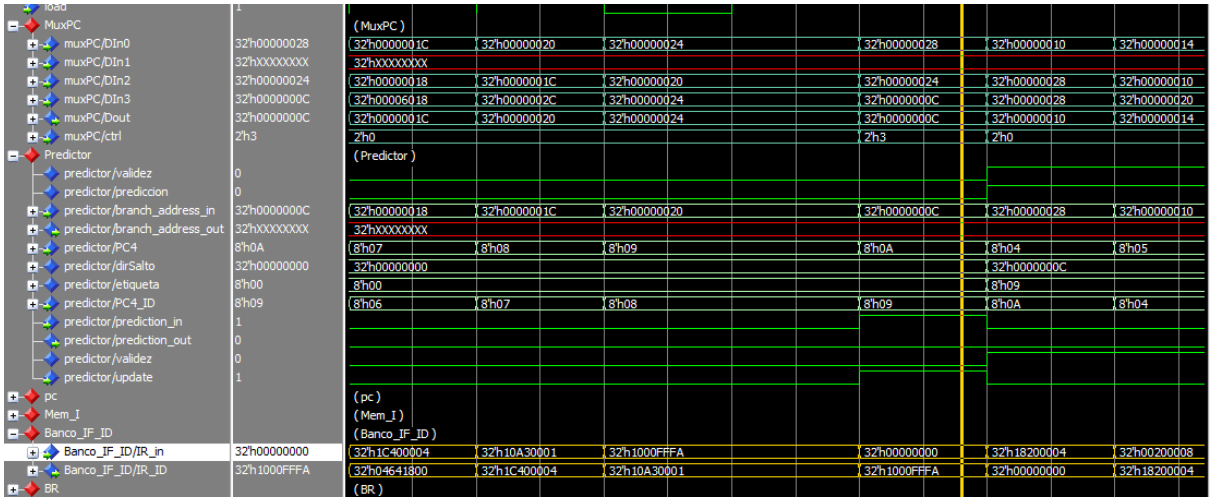


Puede verse en la imagen que en el ciclo D de la instrucción NOP (*00200008*) no se produce ninguna parada a la espera de operandos.

Gestión de riesgos de control con el predictor

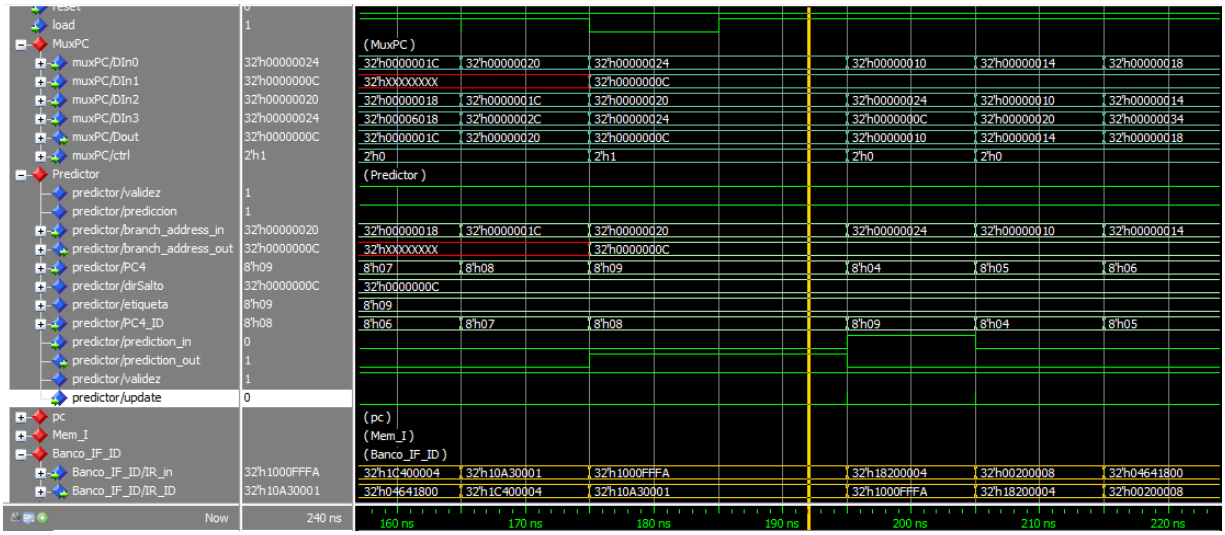
Para demostrar el buen funcionamiento del predictor, se van a mostrar los siguientes casos.

El predictor dijo que no se saltaba y si se salta



Se está en el punto en el que ha terminado la primera iteración del bucle y se va a producir el salto para realizar la segunda. En el ciclo en el que la instrucción 8 (*1000FFFA)* se encontraba en F, el predictor dijo que no se saltaba (prediction_out=0), pero resulta que en D al hacer la comprobación del salto sí que se salta. Por tanto hay que actualizar la información del predictor (prediction_in=1 y branch_address_in=0xC), introducir el dato correcto al PC y cambiar por una NOP la instrucción que se encuentra en F.

El predictor dijo que se saltaba y sí se salta



Nos encontramos al final de la segunda iteración del bucle. Como en la anterior se guardó en el predictor la información del salto, en el ciclo en el que la instrucción 8 (*1000FFFA)* se encuentra en F, se puede ver que como el PC4 coincide con la etiqueta, hay validez y la última vez se saltó, se produce el salto en F. En D se comprueba si habíamos acertado en F, y como es así no se tiene que actualizar el predictor.

CONCLUSIONES

En este proyecto se han aprendido distintas estrategias para optimizar el rendimiento de un procesador segmentado. Teniendo en cuenta la fórmula del tiempo de ejecución: $T_{ex} = I \times CPI \times T_c$ se pueden obtener las siguientes conclusiones:

- Mediante las instrucciones preincremento, se han conseguido fusionar dos instrucciones en una. Instrucciones aritméticas con instrucciones de acceso a memoria, por tanto, se reduce el número de instrucciones y en consecuencia el tiempo de ejecución.
- Mediante la parada ante la detección de riesgos de datos, se ha conseguido reducir el número de instrucciones ya que así no se introducen NOPs para evitar los riesgos, aunque se aumentan los CPI debido a las paradas. Esto lo único que hace es que el programador evite tener que hacer la gestión de riesgos el mismo. Con una unidad de anticipación de operandos, evitaríamos tener que poner NOPs, realizar tantas paradas y así reduciríamos el número de instrucciones lo que haría el mismo efecto en el tiempo de ejecución.
- Gracias a la predicción dinámica de saltos mediante el predictor, se pueden llegar a predecir los saltos en un solo ciclo, por tanto en situaciones como un bucle en el que se va a ejecutar el mismo salto varias veces, los CPIs se reducen ya que se van a producir muchos aciertos en el predictor y lógicamente el tiempo de ejecución es menor.

Aún con todas las mejoras añadidas sigue habiendo aspectos que optimizar, ya que, tal y como cita la primera ley del software: “El software es un gas, se expande hasta ocupar todo el recipiente que lo contiene”.

NOTAS

Horas dedicadas al proyecto:

- Entendimiento del código y familiarización con el entorno: 6 horas.
- Instrucciones preincremento: 12 horas.
- Riesgos de datos: 12 horas.
- Riesgos de control: 18 horas.
- Escritura de la memoria: 10 horas.

Total de horas dedicadas al proyecto: 58 horas.

Editor de texto usado: Sublime Text 2 con paquete instalado del lenguaje VHDL.

Puede verse la evolución del proyecto en el siguiente enlace:
https://github.com/danirueda/AOC2-Projects.git