Andrew Mackay 737069
Rubén Rodríguez 737215

Modificaciones

Para hacer el BEQ esperar a que se actualicen los registros correctamente en caso de que una instrucción anterior quiera escribir en uno de los registros que quiera utilizar el BEQ, se ha añadido una condición para parar la etapa ID: si el código de operación es el del BEQ y si la instrucción en la etapa EX o MEM quiere escribir en un registro que quiera utilizar el BEQ. Además, se ha cambiado la condición para activar *Kill_IF:* la etapa ID no debe estar parado. También se ha cambiado en el MIPS la condición para que se ponga a 1 el mux que controla la fuente del PC, solo si la etapa ID no está parada, además de que esté activada la señal *Z* y *branch.*

Por otro lado, para evitar paradas innecesarias cuando hay NOPs, se ha añadido en la UD otra condición para parar la etapa ID: solo se llegará a parar la etapa ID si no es una instrucción NOP, si lo es nunca parará.

Prueba 1

El contenido inicial de los registros es de 0 y el valor almacenado en (r0)4 es 1.

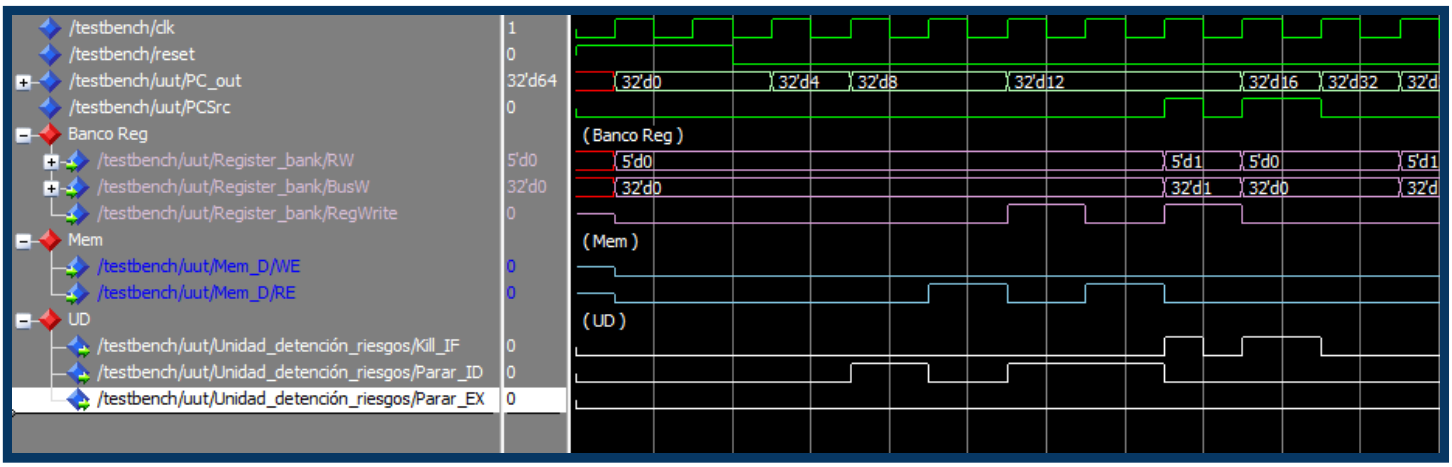
Programa con las instrucciones

Instrucción
LW r0, (r0)
LW r1, 4(r0)
BEQ r1, r0, #inm
BEQ r0, r0, #inm

A continuación se muestra la codificación de las instrucciones

Instrucción	Op	Rs	Rt	K	Hexadecimal
LW r0, (r0)	000010	00000	00000	0000000000000000	08000000
LW r1, 4(r0)	000010	00000	00001	0000000000000100	08010004

BEQ r1, r0, #inm	000100	00000	00001	111111111111111010	1001FFFE
BEQ r0, r0, #inm	000100	00000	00000	0000000000000100	10000004



En esta prueba cargamos en el registro r0 un 0 y en el registro r1 el contenido de r0 desplazado 4 unidades. Al ejecutar la instrucción primera no salta porque los registros r0 y r1 son distintos, mientras que al ejecutar la segunda, sí que lo hace porque compara r0 con r0.

Esto puede observarse en la imagen ya que la señal *Parar_ID* está a 1 cuando ejecuta las dos instrucciones LOAD y el primer BEQ, ya que esta última instrucción debe pararse hasta que termine de cargar el registro r1. Como r0 y r1 no son iguales, el salto no se toma, la señal *PCSrc* vale 0 y la señal *Kill_IF* también está a 0. Sin embargo, al ejecutar el segundo BEQ no hay parada porque no hay dependencias. Como se compara r0 consigo mismo, se debe realizar el salto de la instrucción BEQ, activándose PCSrc a 1 y la señal Kill_IF a 1.

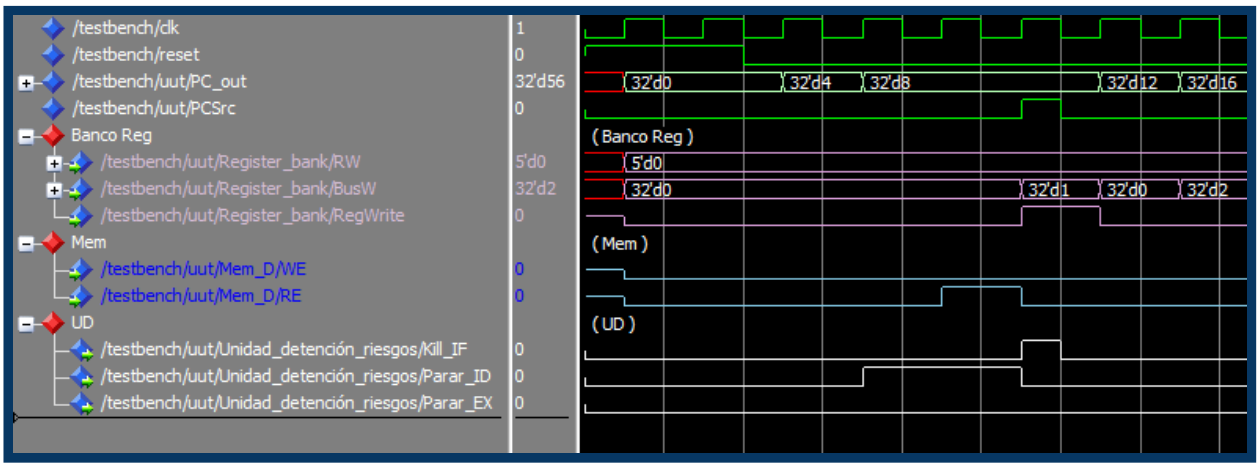
Prueba 2

El valor inicial de los registros es de 0 y el valor contenido en (r1) es 1.

Instrucción
LW r0, (r1)
BEQ r0, r1, #-1

A continuación se muestra la codificación de las instrucciones

Instrucción	Op	Rs	Rt	K	Hexa
LW r0, r1	000010	00001	00000	0000000000000000	08200000
BEQ r0 , r1, ,#-1	000100	00000	00001	1111111111111111	1001FFFF



En esta prueba se carga en r0 el contenido de la dirección de memoria apuntado por el registro r1. Posteriormente se ejecuta una instrucción de salto BEQ que compara el contenido de ambos registros. Al realizar la ejecución de LOAD, existe una dependencia con la instrucción BEQ ya que lee el dato almacenado en el registro r0, que es donde guarda el dato leído de memoria el LOAD. Por ello, se debe parar y activar la señal *Parar_ID* parando también el BEQ ya que necesita el valor obtenido del LOAD. Al escribir el valor 1 en r0, el BEQ compara r0 y r1, que son diferentes y por tanto no salta, tal y como se puede verificar en la señal *PC_out*.

Prueba 3

Todos los registros tienen como valor inicial 0 y el valor contenido en (r0)4 es 1.

Instrucción
LW r1, (r0)4
ADD r2 , r1 , r0
BEQ r1,r2,#-2

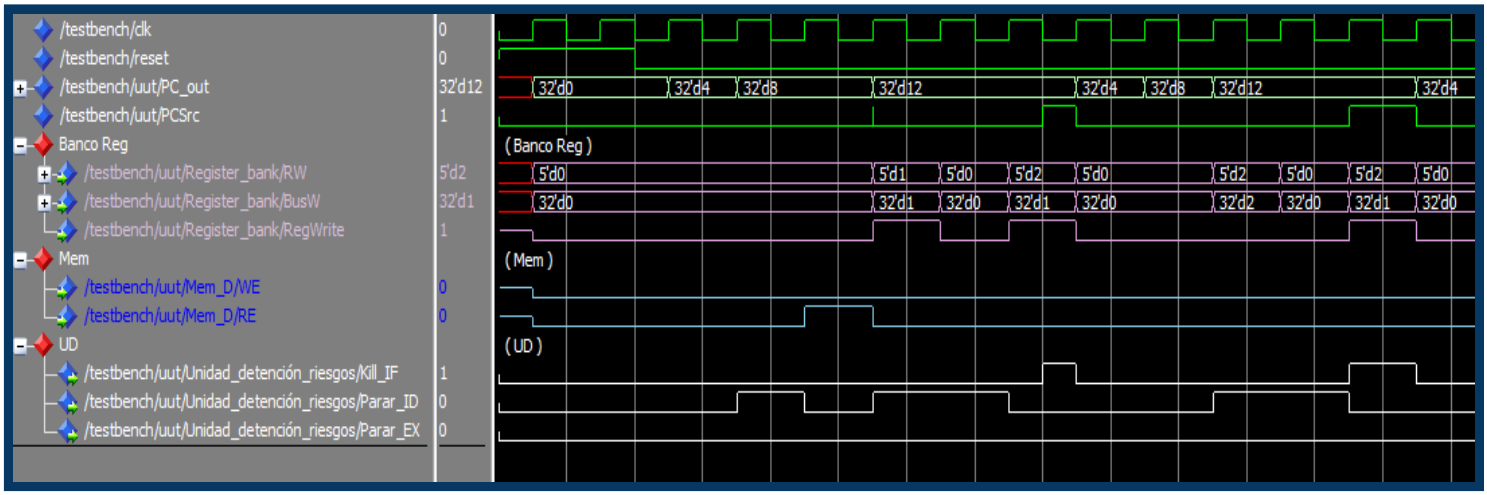
A continuación mostramos el código del programa y la codificación

Instrucciones de gestión de memoria y salto

Instrucción	Op	Rs	Rt	K	Hexa
LW r1, (r0)4	000010	00000	00001	0000000000000100	08010004
BEQ r1,r2,#-2	000100	00001	00010	1111111111111110	1001FFFE

Instrucción aritmética

Instrucción	Op	Rs	Rt	rd	Shamt	Funct	Hexa
ADD r2 , r1 , r0	000000	00001	00000	00010	00000	000000	04201000



Este programa de prueba consiste en un bucle donde se repite siempre el mismo ADD. Se mantiene en un bucle gracias al BEQ que compara r1 y r2, donde ambos tienen valor 1. Una vez entrado en el bucle repite siempre las mismas dos detenciones, una por esperar a que el ADD guarde el valor correcto en r2, ya que el BEQ quiere comparar el registro en el que se guarda el resultado del ADD, y otro por tomar el salto.

Se puede comprobar en la imagen que el salto se toma cada vez que llega al BEQ.

Prueba 4

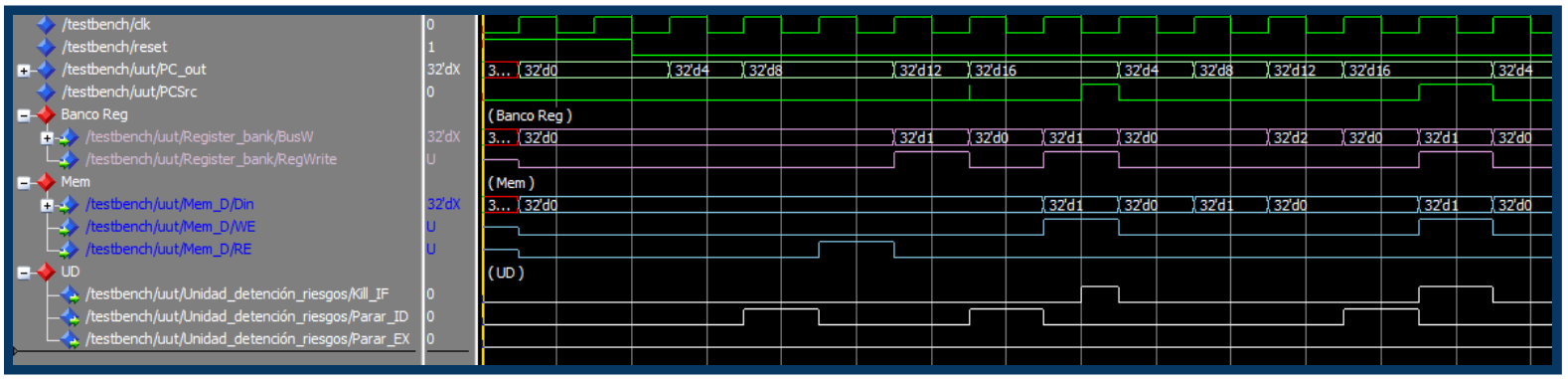
Todos los registros tienen como valor inicial 0 y en la dirección (r0)4 se encuentra el valor 1. Seguidamente se muestra la codificación del programa con las instrucciones.

Instrucción
LW r1,(r0)4
ADD r2,r1,r0
STR r2,(r0)
BEQ r1,r2,#-3

Instrucción	Op	Rs	Rt	K	Hexa
LW r1, (r0)4	000010	00000	00001	0000000000000100	08010004
STR r2,(r0)	000011	00000	00010	0000000000000000	0C020000

BEQ r1,r2,#-2	000100	00001	00010	1111111111111110	1001FFFE

Instrucción	Op	Rs	Rt	rd	Shamt	Funct	Hexa
ADD r2 , r1 , r0	000000	00001	00000	00010	00000	000000	04201000



El programa realiza un bucle repitiendo la instrucción ADD y STR, parecida a la prueba anterior pero con una instrucción entre el ADD y el BEQ. El salto se debe realizar ya que el resultado de la suma, guardada en r2, debe ser igual a r1, porque lo suma con un 0. La unidad de detención realiza dos detenciones de la etapa ID en la primera iteración del bucle, la primera por el LW y el segundo por el BEQ, ya que debe esperar a que se guarde el resultado de la suma en r2 para poder comparar los valores correctos. También se activa la señal *Kill_IF* por el BEQ. Luego, se mantiene en un bucle donde repite siempre la misma detenciones del BEQ.

Viendo en la imagen las direcciones guardadas en *PC_out* se puede comprobar que funciona correctamente el programa.

Prueba 5

Los registros están inicializados a 0 y en la dirección (r0) está guardado un 1.

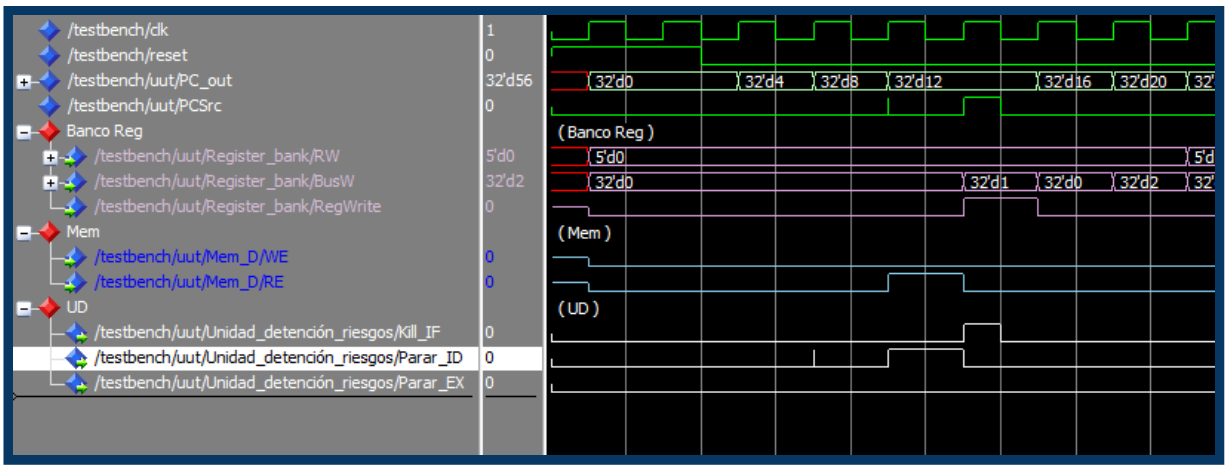
Instrucciones del programa

Instrucción
LW r0,(r1)
NOP
BEQ r0,r1,#16

Codificación de la instrucciones

Instrucción	Op	Rs	Rt	K	Hexa
LW r0, (r1)	000010	00001	00000	0000000000000000	08200000
BEQ r0, r1, 16	000100	00000	00001	0000000000010000	10010010

La instrucción NOP son todo ceros



Como podemos observar, solo se realiza una detención del ID. Esto es debido a que el BEQ tiene que esperar a que se guarde el valor del LW, pero solo espera un ciclo ya que al NOP (que utiliza el registro r0) no lo espera. El LW tampoco detiene ID aunque el NOP utiliza r0, ya que no debe hacer esperar al NOP puesto que no debe hacer nada.