Proyecto 1 – 2023 – Gestión de riesgos y excepciones en MIPS

Pablo Ernesto Angusto Delgado 842255

Miguel Aréjula Aísa 850068

sábado, 25 de marzo de 2023

ÍNDICE:

Proyecto 1 – 2023 – Gestión de riesgos y excepciones en MIPS

ÍNDICE: 1
Breve resumen 2
Verificación de los requisitos funcionales: 2
RF1 cambio a modo excepción 2
Figura 1: Código del multiplexor de entrada al registro PC 3
RF2 retorno a modo usuario RTE 4
RF3 WRO 5
RF4 Anticipación de operandos 6
- Test 5: Incremento y decremento de variables 6

RF5 Riesgos de datos 8
- Test 6: Búsqueda de un valor en un vector 9

RF6 Riesgos de control 12
RF7 contadores 12
Cuantificación de horas dedicadas 14
Conclusiones y Autoevaluación 14
ANEXO: 16
Imagen 1: Captura de pantalla del simulador ejecutando el primer test de prueba 16
Imagen 2: Captura de pantalla del simulador ejecutando el segundo test de prueba 16
Imagen 3: Captura de pantalla del simulador ejecutando el cuarto test de prueba 16
Imagen 4: Captura de pantalla del simulador ejecutando el quinto test de prueba 17
Imagen 5: Captura de pantalla del simulador ejecutando el quinto test de prueba mostrando los valores del PC y la señal corto_b_mem 17
Imagen 6: Captura pantalla del simulador ejecutando el sexto test de prueba mostrando 18
Imagen 7: Captura pantalla del simulador ejecutando el sexto test de prueba focalizando las señales lw_uso y wro_rs 19
Imagen 8: Captura pantalla del simulador ejecutando el sexto test de prueba focalizando el código de operacion y kill_if 20
Imagen 9: Captura pantalla del simulador ejecutando test_IRQ focalizando en las señales de incrementar los contadore 21

Breve resumen

En la versión base de MIPS es un procesador que no es capaz de anticipar operandos ni de detectar riesgos de datos y de control. Por lo tanto es el programador o el compilador el que se encarga de evitar estos riesgos ya sean modificando el orden de las instrucciones o utilizando instrucciones nops. Nuestro objetivo es añadir a esta versión estas funcionalidades. Además, hemos añadido dos instrucciones: RTE y Write Output. RTE se encarga de volver a la instrucción previa a la excepción. Write Output es una instrucción que escribe en la salida del MIPS el valor de un registro. Para ello hemos modificado señales y componentes del MIPS para poder realizarlo, nos hemos ayudado de la unidad de anticipación y de detención, las cuales comunicarán al MIPS mediante señales cuando detecten los riesgos correspondientes, para que MIPS actúe en consecuencia. Por último, hemos añadido una serie de contadores al procesador para poder monitorizar ciertos parámetros sobre el número de instrucciones y ciclos en unas situaciones determinadas. Para probar las mejores realizadas en MIPS hemos diseñado varios test que nos ayudarán a depurar y comprobar el correcto funcionamiento de este.

Verificación de los requisitos funcionales:

RF1 cambio a modo excepción

Descripción: Si el procesador está en modo usuario y recibe IRQ, ABORT o UNDEF pasa al modo correspondiente y ejecuta la rutina que indica la tabla de vectores de excepción. Si está en modo excepción ignora las señales hasta volver a modo usuario.

¿**Cómo**? En el registro PC se van introduciendo los valores de la instrucción a ejecutar, en el momento que se recibe una excepción y es aceptada, salta a la dirección de memoria predefinida para cada uno: Data_abort=x"00000008", UNDEF=x"0000000C" ,

IRQ=x"00000004”. En estas direcciones se encuentra un beq que saltará a la rutina de tratamiento de la excepción correspondiente.

```
PC_in <=    x"00000008"          when (Exception_accepted = '1') and (Data_abort = '1')  else

            x"0000000C"          when (UNDEF = '1') and (Exception_accepted = '1' ) else
            x"00000004"          when (Exception_accepted = '1') and (IRQ = '1') else
            Exception_LR_output   when (RTE_ID = '1') else
            Dirsalto_ID          when (salto_tomado = '1') else
            PC4; -- PC+4
```

Figura 1: Código del multiplexor de entrada al registro PC

Verificación:

- IRQ: banco de pruebas test_IRQ en el que se realizan diversas IRQs. Todas ellas se atienden si y sólo si el procesador está en modo usuario. Se ejecuta una RTI que contabiliza el número de IRQs. Se verifica su funcionamiento correcto.

  Como se puede ver en la *Imágen 1* nuestro MIPS ejecuta el programa principal del programa: Codigo_test_IRQ hasta que llega la interrupción IRQ, esta excepción es aceptada porque estamos en modo usuario. En ese momento, comienza a ejecutarse la rutina de excepción de IRQ. Realiza esta secuencia de instrucciones tres veces y durante la tercera repetición de la rutina de IRQ, la señal se vuelve a activar. Sin embargo, es ignorada porque aún se encuentra en el modo de excepción al seguir en la rutina de la anterior interrupción.

- Data Abort: en la memoria de instrucciones se incluyen dos bancos de pruebas en el que se realizan un acceso no alineado, y un acceso a una dirección fuera de rango. En ambos casos la ejecución salta a la rutina Data Abort (bucle infinito).

Como podemos observar en la *Imágen 2* cuando MIPS ejecuta la instrucción con un acceso a memoria no alineado en los 65 ns, se acepta la interrupción Data_abort porque estamos en modo usuario y saltamos a la rutina de excepción en la que se realiza un bucle infinito.

- Undef: en la memoria de instrucciones se incluye un banco de pruebas en el que se ejecuta una instrucción con un código de instrucción desconocido. La ejecución salta a la rutina UNDEF.

  Como podemos observar en la *Imágen 3* cuando MIPS ejecuta la instrucción con un código de operación erróneo, se activa y se acepta la señal UNDEF. Y salta la rutina de excepción correspondiente en la que se realiza un bucle infinito.

RF2 retorno a modo usuario RTE

Descripción: Al terminar la excepción se vuelve a la ejecución original con la instrucción RTE

¿**Cómo**? Cuando entramos en la rutina de una de las excepciones ocurre lo descrito en el apartado anterior. En el momento en el que se acaba la rutina nos encontramos con una instrucción RTE. Como vemos en la *Figura 1*, PC pasa a tener exception_LR_output donde se guarda la dirección de memoria de la instrucción previa a la excepción, continuando con la ejecución del programa principal.

Verificación: Sólo se ha realizado para IRQs. Para el resto de excepciones se asume que es un fallo irresoluble y se debe resetear el sistema, pero la gestión es idéntica que para las IRQs. En el banco de pruebas test_IRQ se puede comprobar que se retorna a la ejecución sin alterarla siempre que se siga un esquema de prólogo y epílogo apropiado (guardando y restaurando los registros en pila). Para ello se utiliza el registro 31 como SP.

Como se puede observar en la *Imágen 1* en los 300 ns se produce una excepción y se acepta, por lo que se salta a su rutina, cuya última instrucción es un RTE. Por lo tanto, vuelve a la instrucción correspondiente del programa principal. Por ejemplo, en los 500 ns ocurre la situación descrita y vuelve al programa principal ya que en el registro 1 del banco de registros se continúa multiplicando por 2 su valor hasta que entra otra interrupción.

RF3 WRO

Descripción: la instrucción WRO permite escribir el contenido de un registro en la salida del MIPs

¿**Cómo**?

Write_output <= write_output_UC and valid_I_ID and (not parar_ID);

Cuando MIPS se encuentra con una instrucción write output, se activará la señal write_output si la instrucción es válida y no se produce una detención en la etapa decode, con el objetivo de no escribir en el registro output un dato erróneo. La señal write_output es la señal load del registro output que cargará el valor del registro correspondiente.

Verificación: En el banco de pruebas test_IRQ se realizan dos WROs (WRO R31, WRO R2).

Como vemos en la *Imágen 1* cuando se activa la señal write_output a los 100 ns se carga en el registro output el valor del registro 31, en este caso 256. Otro ejemplo, a los 450 ns también ocurre lo mismo pero con el registro 2.

RF4 Anticipación de operandos

Descripción: El procesador es capaz de anticipar los operandos para las instrucciones LW, SW y ARIT a distancia 1 (siempre que el dato a anticipar no venga de un lw) y 2.

¿**Cómo**? Cuando el MIPS detecta que el operando almacenado en A (rs) o en B (rt) es el mismo en el que se va a escribir en la etapa de Mem o en la etapa WB, se activarán las señales correspondientes(Corto_A_Mem, Corto_B_Mem,...) que se utilizarán para elegir la salida del multiplexor para poder escoger el dato directamente de la etapa en la que se encuentra la primera instrucción y no tener que esperar a que acabe.

Verificación:

- En el banco de pruebas test_IRQ se realizan las siguientes anticipaciones:
  - De LW R1, 8(R0) a ADD r31, R1, R31: anticipación de Rs distancia 2 (tras detención de un ciclo del ADD)
  - De SUB r31, R31,R1 a LW R1, 0(R31): anticipación de Rs distancia 1

- Test 5: Incremento y decremento de variables

```
int num1 = 10
int num2 = 0

void main(){
        while (num1 > 0){
                num2 = num2 + 5
                num1 = num1 - 1

        }
}
```

Memoria inicial = { 10 ,0, 1, 5 }

Valores finales:

r0 = 0 r1 = 0 r2 = 50 r3 = 1 r4 = 5

Mem(0) = 0 Mem(4) = 50

RESET	@0x0	10210003	beq r1,r1,INI	Salto incondicional al comienzo del programa @16

IRQ	@0x4	1021003E	beq r1,r1,RTI	Salto incondicional a la @64*4
DABORT	@0x8	1021005D	beq r1,r1,RT_ABORT	Salto incondicional a la @96*4
UNDEF	@0xC	1021006C	beq r1,r1,RT_UNDEF	Salto incondicional a la @112*4
INI:	@0x10	08010000	lw r1, 0(r0)	r1<=MEM(0) = 10
	@0x14	08020004	lw r2, 4(r0)	r2<=MEM(4) = 0
	@0x18	08030008	lw r3, 8(r0)	r3<=MEM(8) = 1
	@0x1C	0804000C	lw r4, 12(r0)	r4<=MEM(12) = 5
MAIN				
LOOP	@0x20	04441000	add r2, r2, r4	r2 = r2 + r4
	@0x24	0C020004	sw r2, 4(r0)	MEM(4) <= r2
	@0x28	04230801	sub r1,r1, r3	r1 = r1 - r3
	@0x2C	0C010000	sw r1, 0(r0)	MEM(0) <= r1
	@0x30	10200001	beq r1,r0, END	Si r1 = r0 salta a END
	@0x34	1000FFFA	beq r0, r0, LOOP	Salto incondicional a LOOP
END	@0x38	1000FFFF	beq r0, r0, END	Salto incondicional a END (bucle infinito)
RTI	@0x100	20000000	rte	Vuelta a la instrucción que se interrumpió
RT_ABORT	@0x180	1000FFFF	beq r0, r0, RT_ABORT	Salto incondicional a RT_ABORT (bucle infinito)
RT_UNDEF	@0x1C0	1000FFFF	beq r0, r0, RT_UNDEF	Salto incondicional a UNDEF (bucle infinito)

- Este test lo hemos realizado de tal forma que se realicen varias anticipaciones de operandos:
    - Entre add r2, r2, r4 y sw r2, 4(r0). El add realiza una suma que es guardada en el registro 2. La segunda instrucción se trata de un store por lo cual es necesario realizar una anticipación de los operandos. Como se puede ver en la quinta imagen mientras se ejecuta la instrucción sw salta la señal corto_b_mem. Esto significa que se ha detectado correctamente que el operando almacenado en B (rt) es el mismo que se encuentra en la etapa de memoria.
    - Ocurre lo mismo con sub r1,r1, r3 y sw r1, 0(r0), anticipación de rt distancia 1

RF5 Riesgos de datos

Descripción: El procesador es capaz de detener la ejecución para evitar los riesgos causados por las dependencias en instrucciones lw-uso (distancia 1), beq o WRO (distancias 1 o 2).

¿**Cómo**? El MIPS debe detectar cuando existe dependencia entre los datos utilizados en las instrucciones, es decir, una instrucción no puede utilizar un dato que va a ser modificado por la instrucción anterior. Para solucionar este problema hay dos tipos de soluciones: reorganizar las instrucciones para evitar las dependencias o como hace nuestro procesador, detecta estas dependencias y produce unas detenciones. MIPS activará parar_ID cuando se activan las señales de riesgo de datos y esa instrucción es válida en la etapa ID o cuando se da la orden de parar_EX.

Verificación:

- En el banco de pruebas test_IRQ se realizan las siguientes detenciones:

- Test1: 1 ciclo de detención por dependencia a distancia 2 entre LW R31, 0(R0) y WRO R31.
- Test 2: 2 ciclos de detención por dependencia a distancia 1 entre ADD R1, R1, R1 y beq R1, R1, main.
- Test 3: 1 ciclo de detención por dependencia lw-uso en varios casos. Por ejemplo: de LW R1, 8(R0) a ADD r31, R1, R31.
- Test 4: 2 ciclos de detención por dependencia a distancia 1 entre ADD R2, R1, R2 y WRO R2.

- Test 6: Búsqueda de un valor en un vector

```
int num1 = 10
int vector[num1]={3,6,4,2,7,1,5,8,0,9}
int i=0;
int obj=8;
int uno=1
int cuatro=4


Pseudocódigo:
void main(){
        while (i < num1){

                if(vector[i]==obj){
                        break()
                }
                i++
        }
        print(i)
}


Valores finales:
r1=A
r2=20
r3=8
r4=8
r5=1
r6=4
r7=8
```

r8=8
IO_output=8
Mem(0,1,2,...,14)=A,3,6,4,2,7,1,5,8,0,9,0,8,1,4

RESET	@0x0	10210003	beq r1,r1,INI	Salto incondicional al comienzo del programa @16
IRQ	@0x4	1021003E	beq r1,r1,RTI	Salto incondicional a la @64*4
DABORT	@0x8	1021005D	beq r1,r1,RT_ABORT	Salto incondicional a la @96*4
UNDEF	@0xC	1021006C	beq r1,r1,RT_UNDEF	Salto incondicional a la @112*4
INI:	@0x10	08010000	lw r1, 0(r0)	r1<=MEM(0) = 10
	@0x14	08020038	lw r2, 56(r0)	r2<=MEM(4) = vector
	@0x18	0803002C	lw r3, 44(r0)	r3<=MEM(44) = 0
	@0x1C	08040030	lw r4, 48(r0)	r4<=MEM(48) = 8
	@0x20	08050034	lw r5, 52(r0)	r5<=MEM(52) = 1
	@0x24	08060038	lw r6, 56(r0)	r6<=MEM(56) = 4
MAIN:	@0x28	10610008	beq r3, r1, fin	Si r3=r1 salta a un bucle infinito de fin
	@0x2C	08470000	lw r7, (r2)	Carga el siguiente dato del vector
	@0x30	10E40003	beq r7, r4, encontrado	Si r7=r4 salta a encontrado
	@0x34	04651800	add r3, r3, r5	r3=r3+1
	@0x38	04461000	add r2, r2, r6	r2=r2+4 Apunta al siguiente dato del vector
	@0x3C	1000FFFA	beq r0,r0, main	Vuelve al principio del bucle
encontrado:	@0x40	0C030000	sw r3, (r0)	MEM(0) = r3(índice

				del vector donde esta el dato encontrado)
	@0x44	08080000	lw r8 (r0)	r8<=MEM(0) = r3
	@0x48	81000000	wro r8	IO_output <= R1
fin:	@0x4C	1021FFFF	beq r1, r1, fin	Bucle infinito en el que solo se sale si entra una IRQ
RTI	@0x100	20000000	rte	Vuelta a la instrucción que se interrumpió
RT_ABORT	@0x180	1000FFFF	beq r0, r0, RT_ABORT	Salto incondicional a RT_ABORT (bucle infinito)
RT_UNDEF	@0x1C0	1000FFFF	beq r0, r0, RT_UNDEF	Salto incondicional a UNDEF (bucle infinito)

Como vemos en *Imágen 6* e *Imágen 7* hay dos tipos de dependencias. En los 900 ns se produce 1 ciclo de detención por una dependencia entre load y beq a distancia 1, ya que en la etapa EX se está ejecutando un load y en la etapa Fetch una instrucción beq. Por otro lado en los 950 ns se produce una dependencia wro a distancia 1, porque la instrucción write output utiliza un registro que la instrucción anterior va a modificar.

RF6 Riesgos de control

Descripción: El procesador es capaz eliminar los riesgos de control causados por las instrucciones de salto: BEQ y RTE.

¿**Cómo**? Explicar brevemente la gestión de riesgos de control.

Cuando se ejecuta una instrucción de salto, la instrucción siguiente se encuentra en la etapa Fetch. Por lo que si no se produce el salto continuará la ejecución de la siguiente instrucción. Sin embargo, si se produce el salto, se activará la señal salto_tomado y por lo tanto, habrá que dejar de ejecutar la instrucción siguiente(PC+4) mediante Kill_If y ejecutar la instrucción del salto.

Verificación:

- En el banco de pruebas test_IRQ se realizan varios saltos tomados en los que sólo se ejecutan las instrucciones adecuadas:
  - beq R1, R1, main
  - rte
- En el banco de pruebas test 6 se puede comprobar que no siempre que se ejecute una instrucción BEQ, saltará y por lo tanto no se activará la señal Kill_If. Como observamos en la *Imagen 8*, cuando el código de operación es el del beq, en el siguiente ciclo se sigue ejecutando PC+4 y no se salta, por lo que no se activa la señal Kill_If.

RF7 contadores

Descripción: los contadores nos dan información de la ejecución del código

¿**Cómo**? Los contadores se encargan de almacenar el número de instrucciones o ciclos para monitorizar un parámetro. Para ello cada señal de incrementar el contador se activa cada vez que su condición se cumple. Estas condiciones se describen en la verificación de este apartado.

Verificación: En el banco de pruebas test_IRQ se ha comprobado que:

- Por cada instrucción válida que hace su etapa WB se incrementa el contador de instrucciones Ins. Si la instrucción no es válida no se incrementa.
- Data_stalls contabilizan bien los ciclos debidos a los riesgos de datos incrementando el número indicado en los test mencionados en el RF5.
- Control_stalls contabilizan bien los ciclos debidos a los riesgos de control, incrementando un ciclo en cada salto tomado (ver pruebas en el RF6).
- Exception_accepted contabiliza las excepciones aceptadas. Su cuenta coincide con la cuenta que realiza el propio código.
- Exception_cycles se actualiza cada ciclo en el que el procesador está en modo excepción.

Como se puede ver en la *Imagen 9*, la señal de incrementar se activa en la situación correspondiente. Por ejemplo, en los 300 ns se produce una excepción por lo que se activan las señales de incremento del contador Exception_accepted y Exception_cycles. Además durante toda la ejecución de la rutina de la excepción la señal Exception_cycles está activa, contando los ciclos que se producen mientras el procesador estan modo excepción.

Cuantificación de horas dedicadas

- Estudio del MIPS, VHDL, entorno, instalación…: 5 h 30 min Miguel y 5 h Pablo
- Adición de las nuevas instrucciones: 1 h 30 min Ambos
- Gestión de excepciones: 1 h Ambos
- Gestión de riesgos: 2 h Pablo y 1 h y 30 min Miguel
- Depuración, verificación y programas de prueba: 13 h Ambos
- Memoria: 3 h Ambos

Todo el trabajo lo hemos realizado juntos en todo momento, salvo la parte de estudio del MIPS, VHDL, entorno y la instalación y en la gestión de riesgos. Miguel tuvo ciertos problemas debido a que su ordenador tiene sistema operativo MacOs y la compilación de ficheros no se ejecutaba correctamente, por lo que intentó realizar el proyecto en un contenedor. Esta solución tampoco obtuvo resultado, por lo que finalmente decidió utilizar una máquina virtual.

Conclusiones y Autoevaluación

Tras realizar este trabajo hemos podido terminar de entender lo explicado en clase sobre MIPS, al haber podido analizar cada detalle del procesador y poder ver en un simulador el funcionamiento de este. Además durante el proceso de realizar el trabajo hemos tenido que trabajar con muchas señales distantes y asimilar su funcionamiento e importancia para nuestro procesador. Es cierto que al principio al tener que trabajar con un entorno de trabajo nuevo y un lenguaje muy distinto a lo que estamos acostumbrados se nos hizo más difícil poder comprender del todo. Con el paso del tiempo hemos sido capaces de acostumbrarnos a VHDL y comprender cada modificación que realizabamos.

El trabajo nos ha parecido muy interesante y beneficioso, ya que nos ha permitido trabajar con un lenguaje más profesional y terminar de comprender el funcionamiento del MIPS. Como autocrítica, nos hemos dado cuenta que nuestra organización es bastante mejorable

y debemos recurrir a las tutorías, ya que los problemas y preguntas que nos han ido surgiendo, podríamos haberlos solucionado de una forma más rápida y eficiente haciendo uso de estas.

Viendo nuestro trabajo realizado estamos muy satisfechos, creemos que es un poco mejorable por lo tanto consideramos que nuestra calificación es un 8,5.

ANEXO:

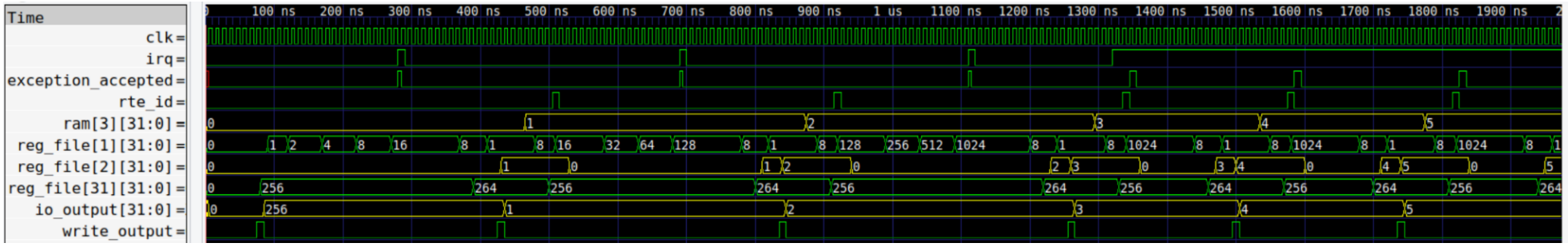

Imagen 1: Captura de pantalla del simulador ejecutando el primer test de prueba

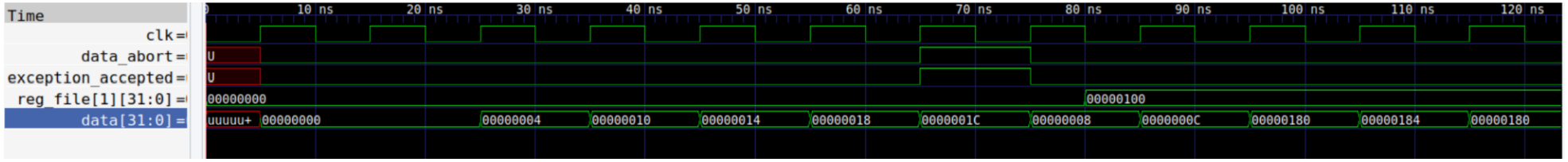

Imagen 2: Captura de pantalla del simulador ejecutando el segundo test de prueba

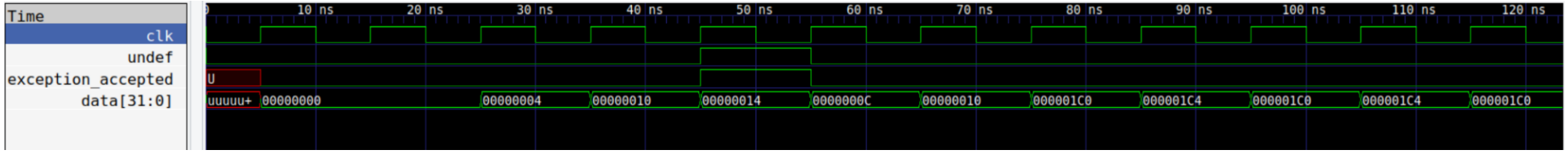

Imagen 3: Captura de pantalla del simulador ejecutando el cuarto test de prueba

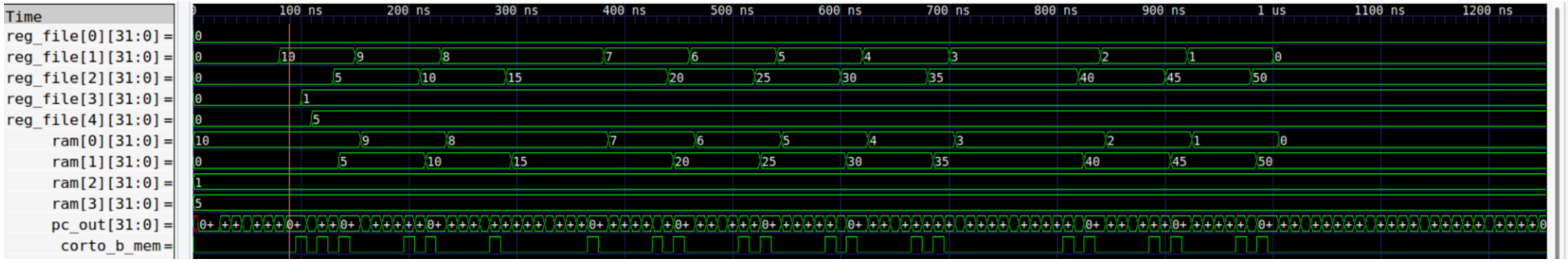

Imagen 4: Captura de pantalla del simulador ejecutando el quinto test de prueba

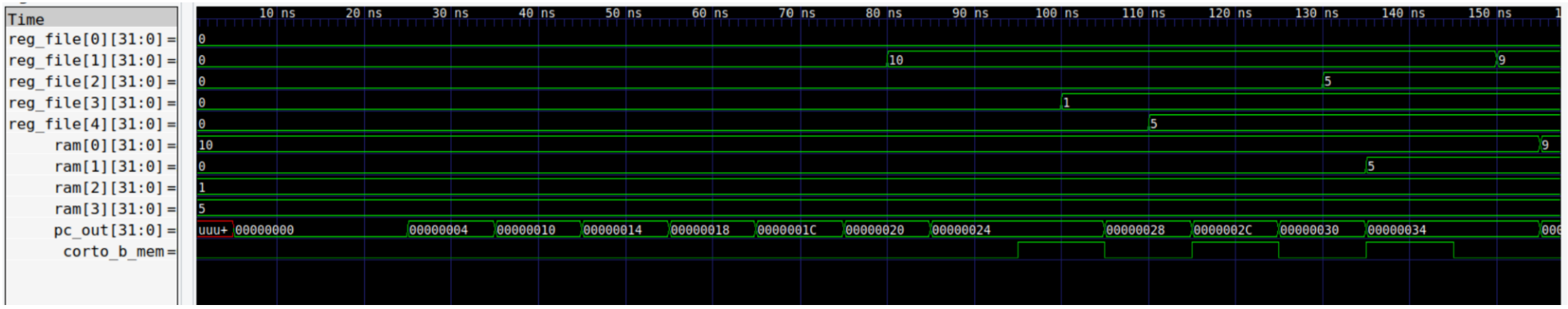

Imagen 5: Captura de pantalla del simulador ejecutando el quinto test de prueba mostrando los valores del PC y la señal corto_b_mem

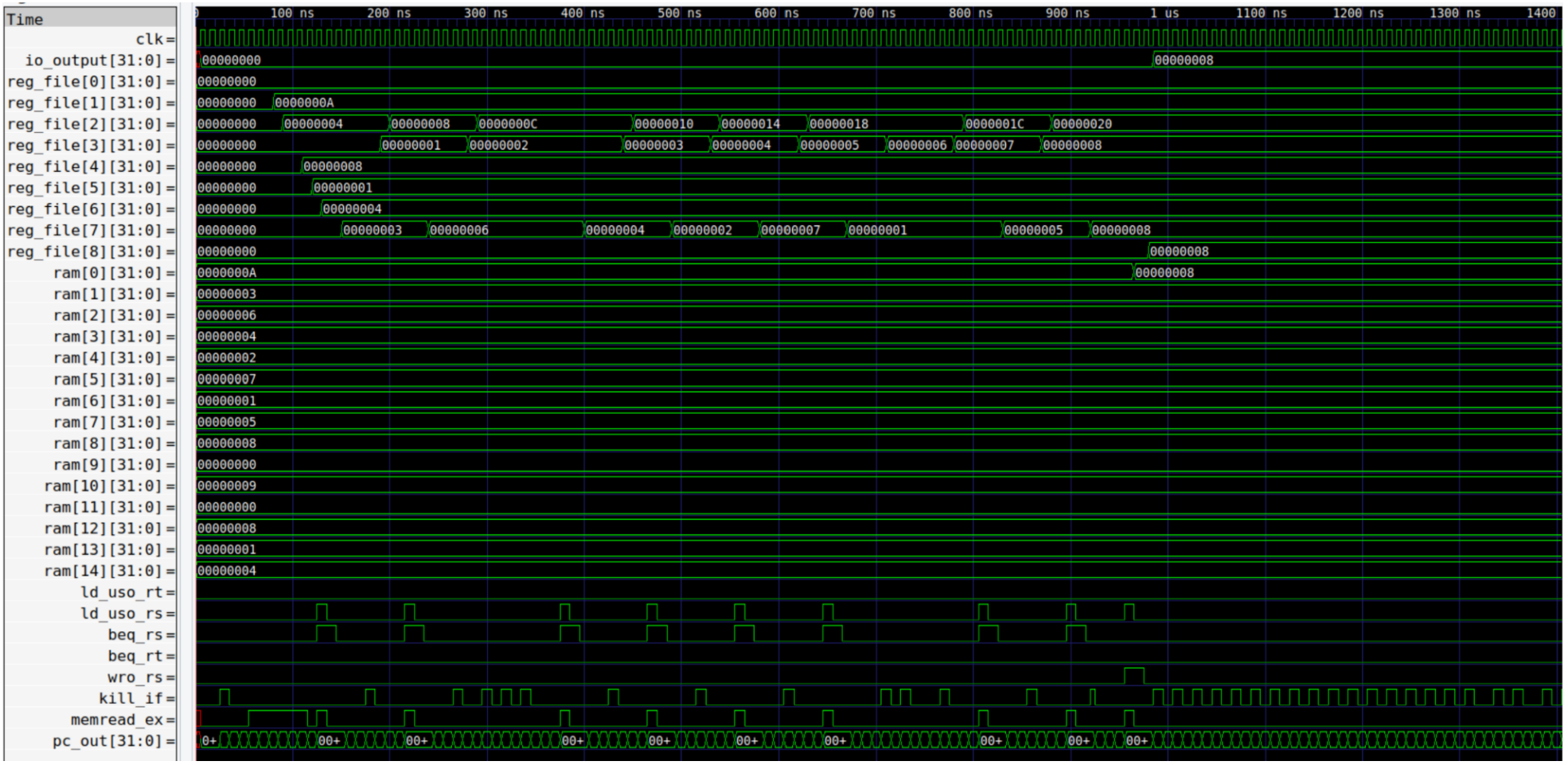

Imagen 6: Captura pantalla del simulador ejecutando el sexto test de prueba mostrando

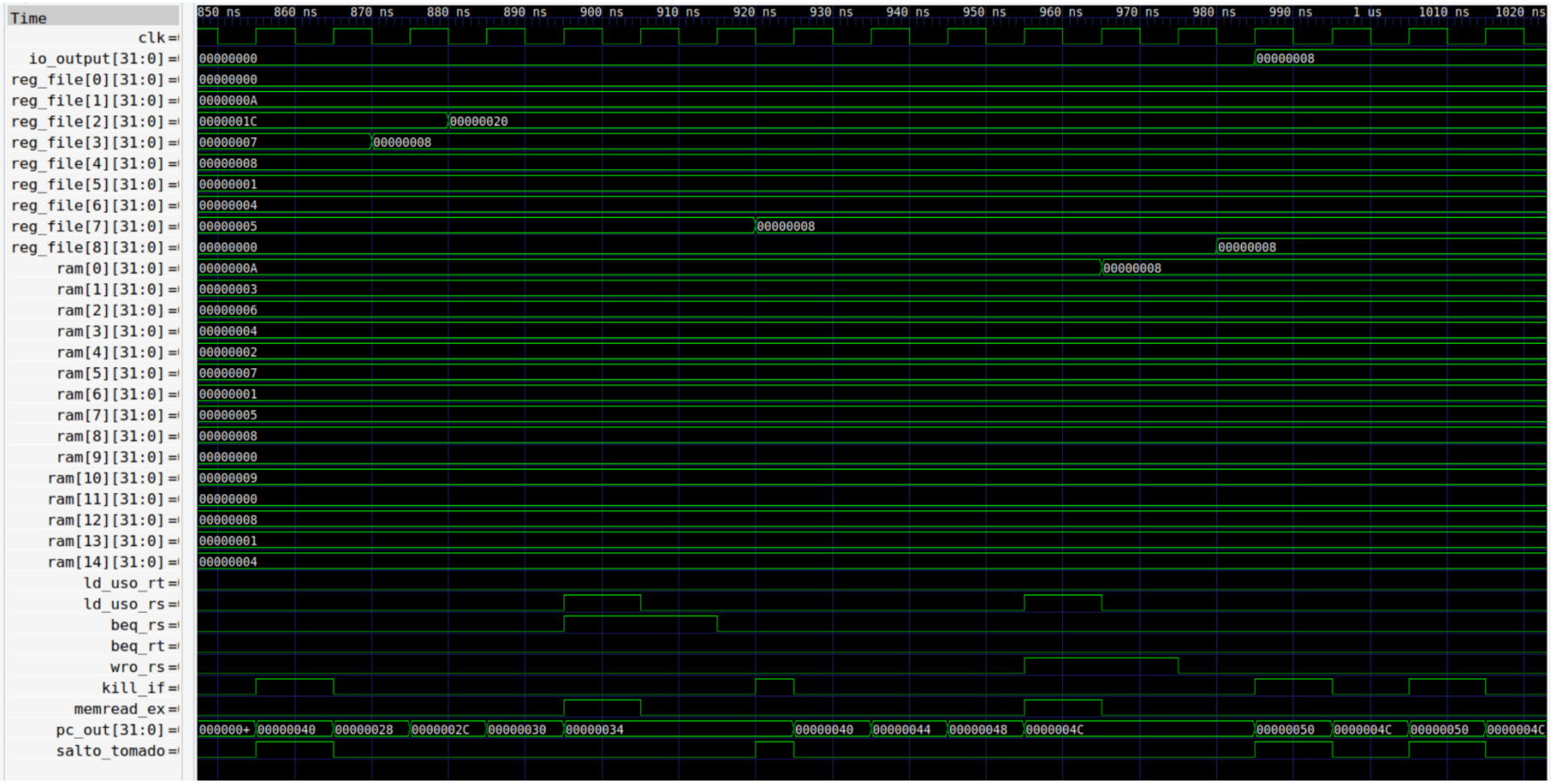

Imagen 7: Captura pantalla del simulador ejecutando el sexto test de prueba focalizando las señales lw_uso y wro_rs

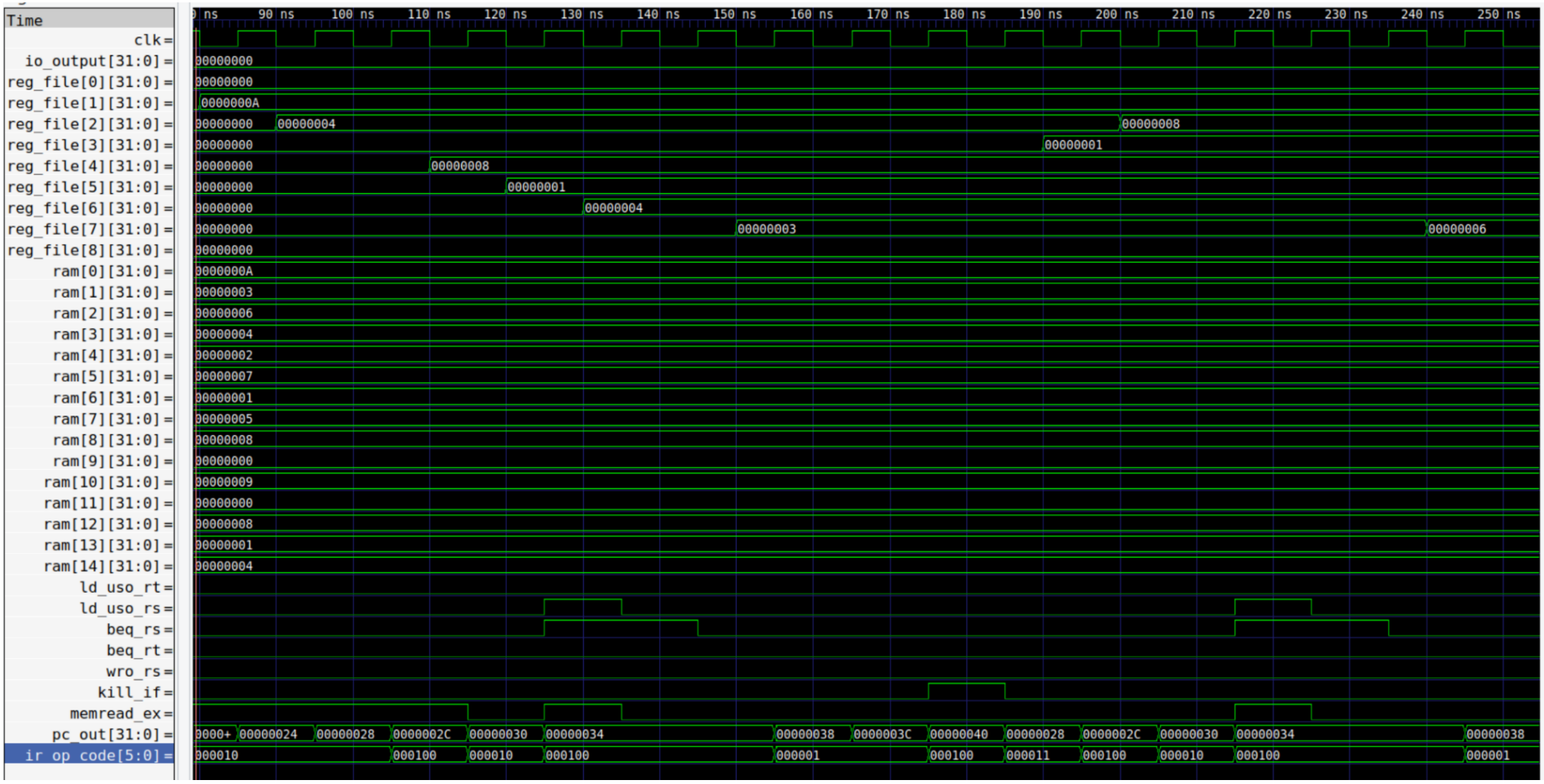

Imagen 8: Captura pantalla del simulador ejecutando el sexto test de prueba focalizando el código de operacion y kill_if

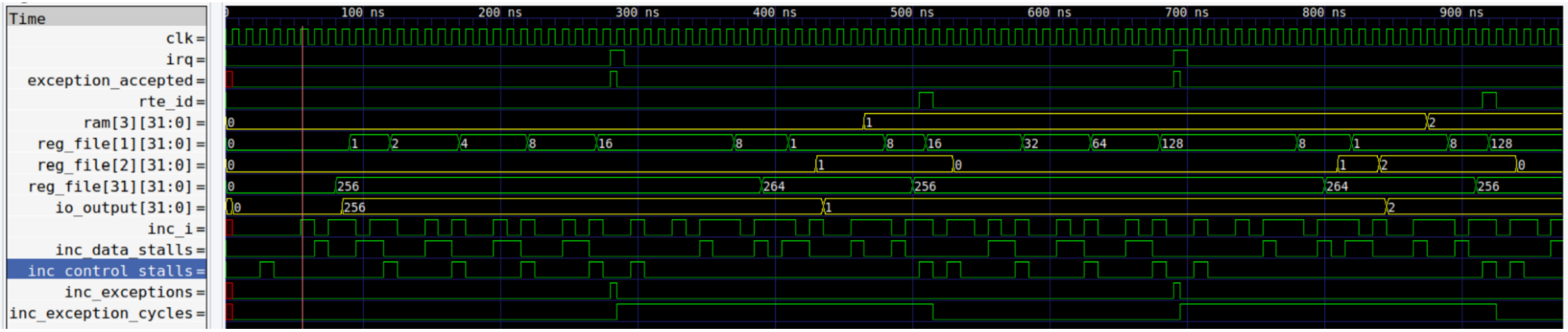

Imagen 9: Captura pantalla del simulador ejecutando test_IRQ focalizando en las señales de incrementar los contadore