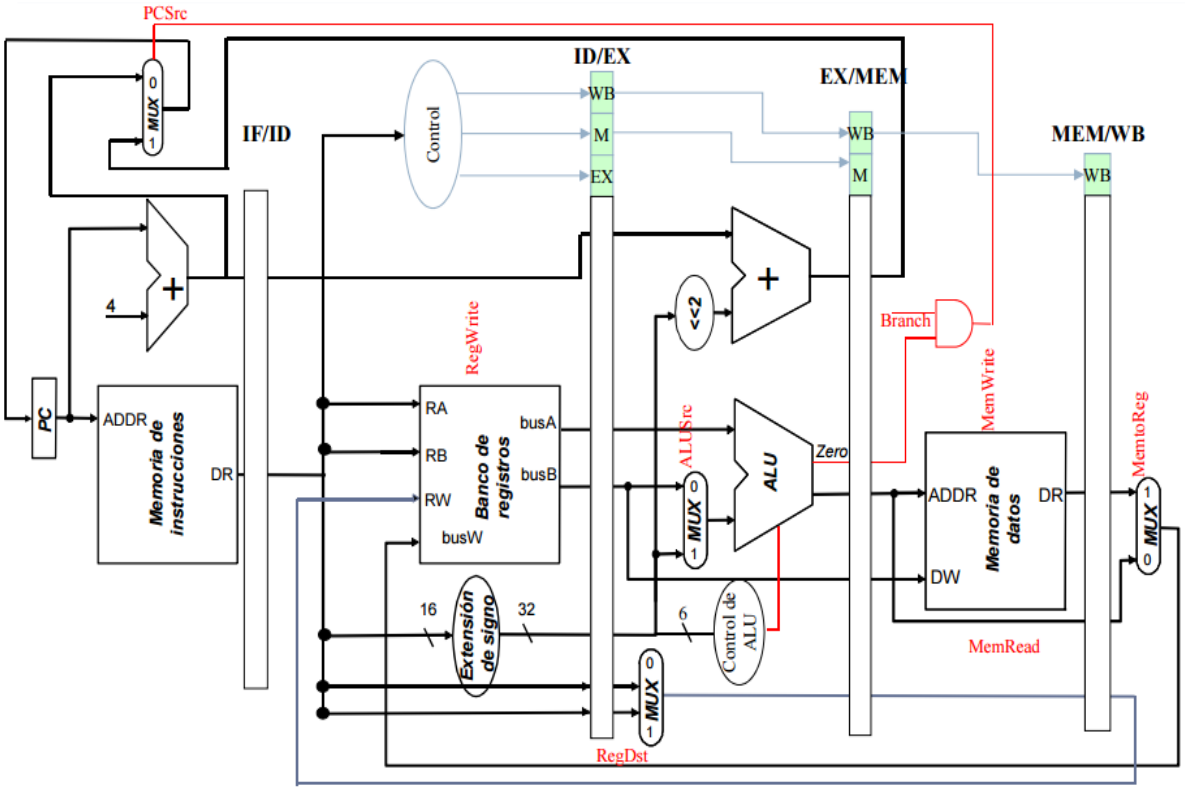
Proyecto Optativo 1:



Gestión de riesgos en MIPS segmentado con Cache de instrucciones

Miguel Allué Barón - 593599

Cristian Simón Moreno - 611487

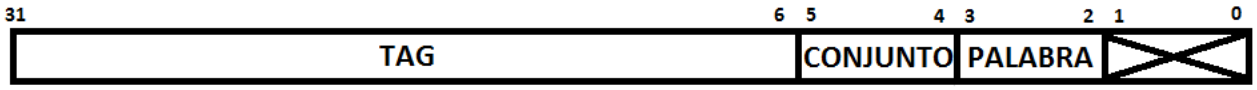
Paso 1: Señales de control a la UC

Operación	Codificación
NOP	000000
ARITMETICA	000001
LOAD	000010
STORE	000011
BEQ	000100

Op_code	Branch	RegDst	ALUSrc	MemWrite	MemRead	MemtoReg	RegWrite
NOP	0	0	0	0	0	0	0
ARIT	0	1	0	0	0	0	1
LW	0	0	1	0	1	1	1
SW	0	0	1	1	0	0	0
BEQ	1	0	0	0	0	0	0

Paso 2: Conexiones de la MC

Estructura de la dirección (ADDR):



- **dir_MI:** Lo usamos para acceder a la memoria principal de 128 palabras.

  *dir_MI <= ADDR (**8** downto **4**)&palabra_UC*

  palabra_UC -> nos permite traernos 4 bloques (00, 01, 10, 11)

- **dir_palabra:** Lo usamos para elegir la instrucción solicitada de un determinado bloque.

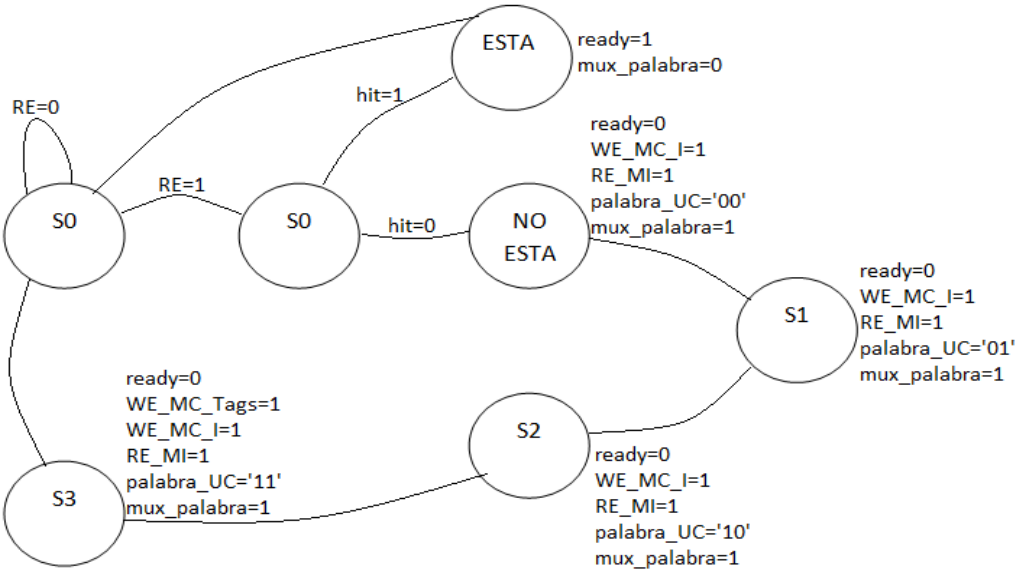
  *dir_palabra <= ADDR (**3** downto **2**)*

- **dir_cjto:** Lo usamos para elegir el conjunto al que se accede en la cache de instrucciones.

  *dir_cjto <= ADDR (**5** downto **4**)*

- **TAG:** *ADDR (**31** downto **6**)*

Unidad de control de la MC



Paso 3: Detención del segmentado

Tenemos 3 tipos de riesgos que debemos tratar:

1) Por riesgos de datos:

Para detectarlas deberemos comparar los operandos de la instrucción actual con los destinos de las instrucciones que tendremos en las etapas EX y MEM, es decir:

En la etapa EX:

- IR_ID(25 downto 21) = RW_EX
- IR_ID(20 downto 16) = RW_EX

En la etapa MEM:

- IR_ID(25 downto 21) = RW_MEM
- IR_ID(20 downto 16) = RW_MEM

Además de esta comparación deberemos comprobar el **tipo de instrucción** que tendremos delante, para evitar hacer paradas innecesarias. Para comprobar el tipo de

instrucción que tendremos en las etapas EX y MEM usaremos las señales **RegWrite** (en EX y MEM), **RegDst_EX**, **MemtoReg_MEM** y **RegWrite_MEM**.

Para realizar la gestión de estos riesgos lo primero que hemos hecho ha sido hacer un análisis de que instrucciones pueden presentar riesgos, y serán aquellas en las que se escriba en el banco de registros, ya que si la siguiente instrucción va a leer el destino de la operación anterior y aun no lo hemos escrito, obtendremos un valor erróneo.

F	**D**	**EX**	**MEM**	**WB**			
	F	**D**	**D**	**D**	**EX**	**MEM**	**WB**

Deberemos esperar a que se escriba en el banco de registros para poder leer el dato en la siguiente instrucción.

Por lo tanto tendremos riesgos si en la instrucción anterior detectamos un **LOAD** o una **ARITMETICA**. El código que hemos realizado para la detección y gestión de riesgos de datos es el siguiente:

```
DetectarRiesgos: process (IR_ID, RegWrite_EX, RegWrite_MEM, MemtoReg_MEM, RW_EX, RW_MEM, riesgo_rs_d1, riesgo_rs_d2, riesgo_rt_d1, riesgo_rt_d2)
begin
    riesgo_rs_d1 <= '0'; --riesgo de datos en rs con la instrucción que hay en EX.
    riesgo_rs_d2 <= '0'; --riesgo de datos en rs con la instrucción que hay en Mem
    riesgo_rt_d1 <= '0'; --riesgo de datos en rt con la instrucción que hay en EX
    riesgo_rt_d2 <= '0'; --riesgo de datos en rt con la instrucción que hay en Mem
    -- STORE, ARIT Y BEQ -> Riesgos con LOAD Y ARIT
    if (IR_ID(31 downto 26)= "000011" OR IR_ID(31 downto 26)= "000001" OR IR_ID(31 downto 26)= "000100") then
        --EX / Comprobamos si es un un load o aritmetica la de EX
        if(RegWrite_EX='1') then
            if(IR_ID(25 downto 21) = RW_EX) then
                riesgo_rs_d1 <= '1';
            end if;
            if(IR_ID(20 downto 16) = RW_EX) then
                riesgo_rt_d1 <= '1';
            end if;
        end if;
        --MEM / Comprobamos si es un un load o aritmetica la de MEM
        if(RegWrite_MEM='1') then
            if(IR_ID(25 downto 21) = RW_MEM) then
                riesgo_rs_d2 <= '1';
            end if;
            if(IR_ID(20 downto 16) = RW_MEM) then
                riesgo_rt_d2 <= '1';
            end if;
        end if;
```

```
    -- LOAD - Riesgos con ARIT
    elsif (IR_ID(31 downto 26)= "000010") then
        --EX / Comprobamos si es una aritmetica la de EX
        if(regDst_EX='1') then
            if(IR_ID(20 downto 16) = RW_EX) then
                riesgo_rt_d1 <= '1';
            end if;
        end if;
        --MEM / Comprobamos si es una aritmetica la de MEM
        if(MemtoReg_MEM='0' AND RegWrite_MEM='1') then
            if(IR_ID(20 downto 16) = RW_MEM) then
                riesgo_rt_d2 <= '1';
            end if;
        end if;
    end if;

    -- COMPROBAMOS SI HA HABIDO ALGUN RIESGO
    if (riesgo_rs_d1 = '1' OR riesgo_rs_d2 = '1' OR riesgo_rt_d1 = '1' OR  riesgo_rt_d2 = '1') then
        avanzar_ID <= '0';
    end if;
    if (riesgo_rs_d1 = '0' AND riesgo_rs_d2 = '0' AND riesgo_rt_d1 = '0' AND  riesgo_rt_d2 = '0') then
        avanzar_ID <= '1';
    end if;
end process;
```

Primero pondremos las señales (**riesgo_rs_d1, riesgo_rs_d2, riesgo_rt_d1** y **riesgo_rt_d2**) que nos servirán para indicar si hay riesgos a 0, después los hemos dividido en 2 partes:

- Si detectamos que la operación actual es un **Store**, **Aritmética** o un **Beq**, comprobaremos que si en la etapa EX o MEM hay un Load o una Aritmética y si la hay, y los operandos y los destinos son iguales pondremos la señal correspondiente de riesgo a 1.

- Si detectamos que la operación actual es un **Load**, comprobaremos si en la etapa EX o MEM hay una Aritmética y si la hay, y los operandos y los destinos son iguales pondremos la señal correspondiente de riesgo a 1.

Luego comprobaremos si hemos detectado algún riesgo, si hemos detectado alguno pondremos **avanzar_ID a 0**, por lo que detendremos la etapa Decode, y también deberemos detener la etapa Fetch, ya que si no la instrucción guardada en Decode la perderemos.

Un ejemplo de riesgo de control:

Tenemos las siguientes instrucciones (Operación - hexadecimal - binario - explicación):

0 - LDI R0, [#0] -> 0800 0000 *000010 00000 00000 000...* *R0 <- 2*

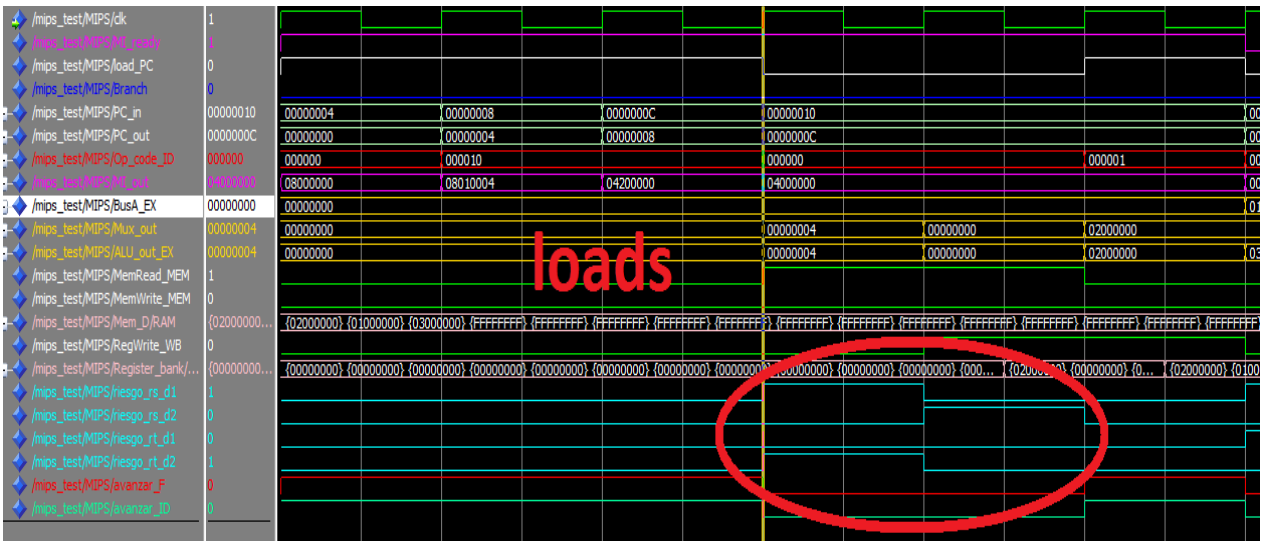
1 - LDI R1, [#4] -> 0801 0004 *000010 00000 00001 000...* *R1 <- 1*

2 - ADD R0, R1 -> 0420 0000 *000001 00001 00000 000...* *R0 <- R0+R1=2+1=3*

Por lo tanto al llegar a la instrucción 2 (*ADD R0, R1)* podemos tener riesgos en 2 etapas:

1. Cuando la instrucción 0 (*LDI R0, [#0])* este en la etapa EX y la instrucción 1 (*LDI R1, [#4])* este en la etapa MEM.
    a. Con la instrucción 0 -> rt (20..16) de 0 = rt (20..16) de 2 -> activamos **riesgo_rt_d2**
    b. Con la instrucción 1 -> rt (20..16) de 1 = rs (25..20) de 2 -> activamos **riesgo_rs_d1**
2. Cuando la instrucción 1 este en la etapa MEM.
    a. Con la instrucción 1 -> rt (20..16) de 1 = rs (25..20) de 2 -> activamos **riesgo_rs_d2**

Lo podemos ver mejor en el cronograma:



Como vemos cargamos los 2 loads, y se detectan los riesgos del add, primero cuando uno está en la etapa EX y el otro en la MEM, y en el siguiente ciclo cuando hay solo un load en la etapa MEM.

2) Por riesgos de control y fallos en la MC:

La realizaremos en la etapa Fetch, con el siguiente código:

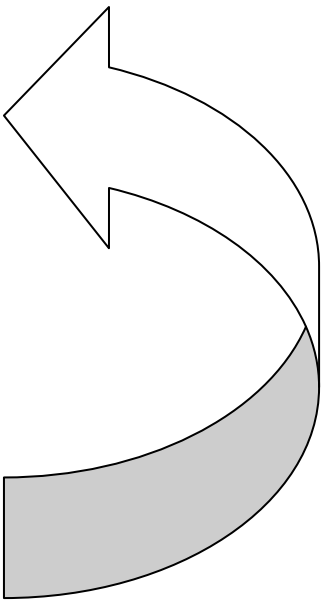
```
-- SI SALTO O M_I LISTA -> LOAD_PC = 1
actualizarPC: process (Branch, MI_ready, avanzar_ID)
begin
    if ((Branch = '1' OR MI_READY = '1') AND avanzar_ID = '1') then
        load_PC <= '1';
        avanzar_F <= '1';
    else
        load_PC <= '0';
        avanzar_F <= '0';
    end if;
end process;
```

Con esto lo que haremos será si tenemos salto (**Branch = 1**), o la memoria de instrucciones tiene la instrucción lista (**MI_READY = 1**) y en Decode avanzamos (**avanzar_ID = 1**), cargaremos PC y avanzaremos en Fetch. En caso contrario pararemos, si no hay que saltar y la memoria no está preparada paramos PC, ya que si no hemos cargado PC, no debemos avanzar a PC+4

Caso de prueba:

Instrucciones:

0 - LDI R0, [#0] -> 0800 0000	*R0 <- 2*
1 - LDI R1, [#4] -> 0801 0004	*R1 <- 1*
2 - ADD R0, R1 -> 0420 0000	*R0 <- R0+R1=2+1=3*
3 - ADD R0, R0 -> 0400 0000	*R0 <- R0+R0=3+3=6*
4 - LDI R1, [#8] -> 0801 0008	*R1 <- 3*
5 - ADD R0, R1 -> 0420 0000	*R0 <- R0+R1=6+3=9*
6 - STI R0, [#12] -> 0C00 000C	*#12 <- 9*
7 - BEQ RO, RO, #0 -> 1000 FFF8	*Salto a la #0*



Contenido de memoria:

#0 -> 2

#4 -> 1

#8 -> 3

Riesgos:

Este código nos presenta multitud de dependencias:

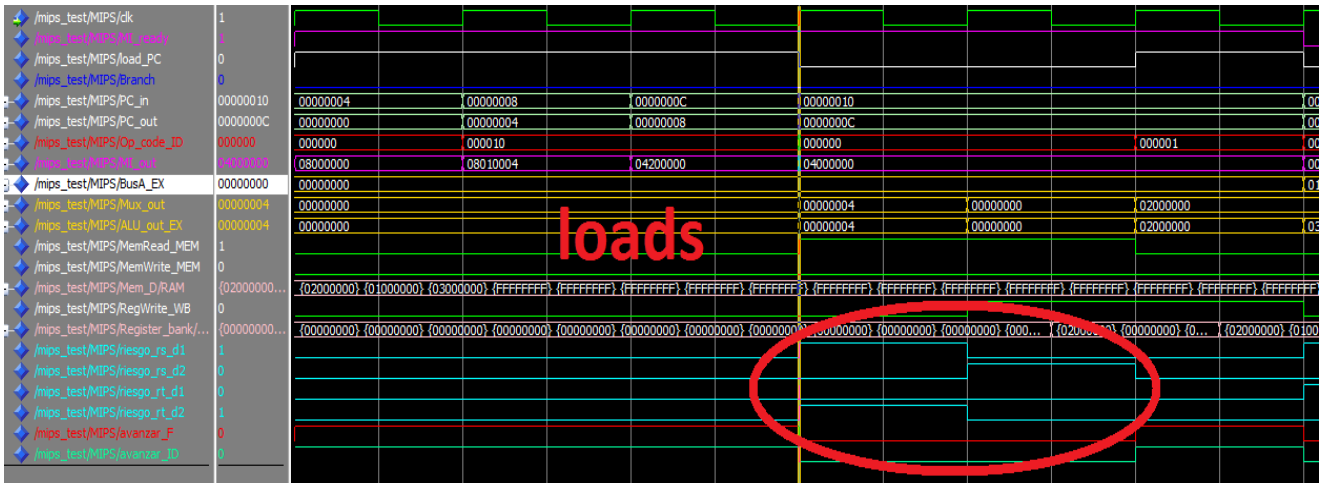
- Los 2 loads iniciales (Instrucciones 0 y 1) con la instrucción 2 (No podemos hacer la operación hasta que no tengamos los 2 registros con los valores correctos y listos para ser leidos).

- La instrucción 2 con la 3 (2 ADD que comparten operandos, por lo que hasta que no escribamos en el banco el resultado de la primera operación no podremos usarlo en la segunda).

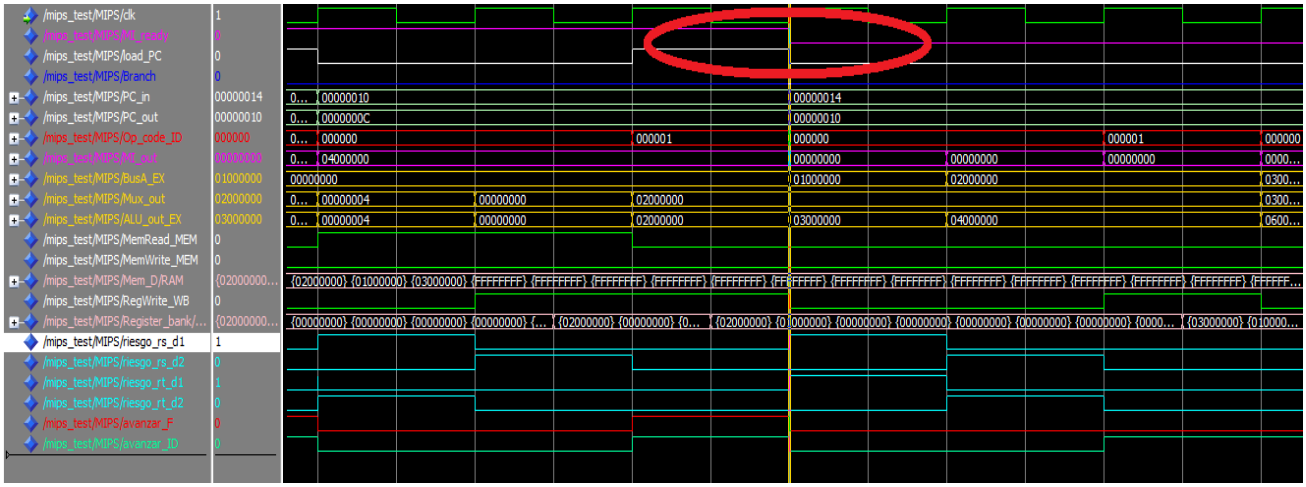
- La instrucción 4 con la 5 (No podemos hacer la operación hasta que no tengamos los 2 registros con los valores correctos y listos para ser leídos).

- La 5 con la 6 (Debemos esperar a que el dato este escrito en el banco antes de cogerlo y almacenarlo en memoria.

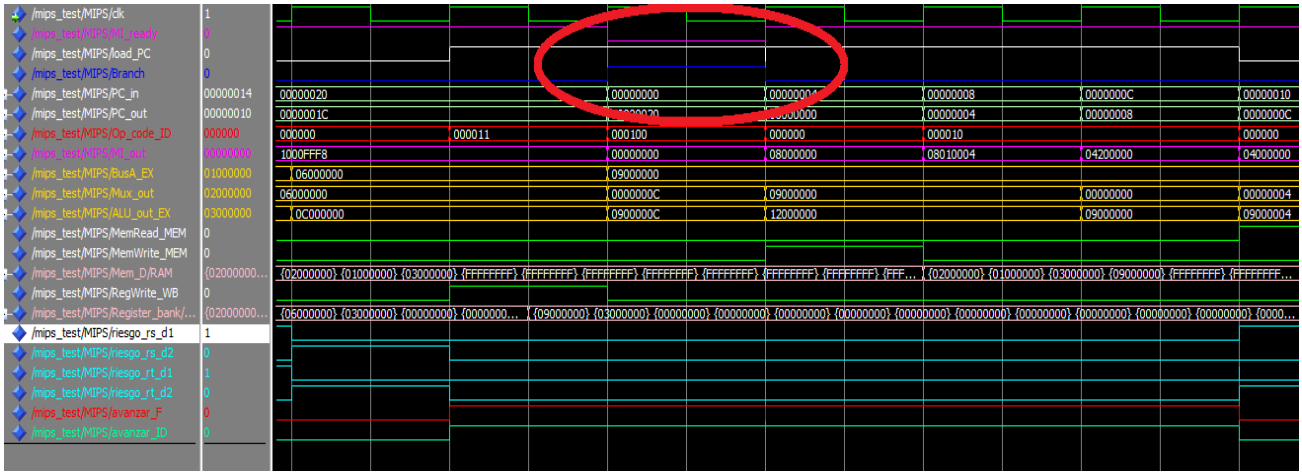
Con este caso de prueba comprobaremos casi todos los posibles riesgos existentes. Por riesgo de datos, por ejemplo tendremos el de los load con las aritméticas:



Podemos ver cómo tras ejecutar los 2 loads iniciales del código (0800 0000 y 0801 0000), se detectan riesgos para la ejecución de la aritmética (0420 0000) por lo que se detiene la etapa Decode mandando NOP, deteniendo también la etapa Fetch. Una vez desaparecidos los riesgos se hace la aritmética.



Si no hay que saltar o la memoria no está preparada paramos PC, ya que si no hemos cargado PC, no debemos avanzar a PC+4. Mientras no esté lista la memoria mandaremos NOPS.



Si hay que saltar actualizamos el PC, en este caso la memoria casi no tarda en tener listos los datos porque ya están en la cache.