Proyecto MIPS – Parte 1

Memoria

Eduardo Gimeno Soriano

721615

Curso 2017-2018

**Índice**

**1. Explicación del diseño**

**2. Hardware añadido**

**3. Medida de rendimiento**

**4. Pruebas**

**5. Conclusiones**

**6. Anexo**

**7. Correcciones**

**1. Explicación del diseño**

Partiendo de la base que se proporcionaba, se ha completado la unidad de detección de riesgos identificando un riesgo estructural, tres riesgos de datos y un riesgo de control. Esta unidad activará las señales de parada, las cuales se conectan a las respectivas señales load de los bancos que separan IF de ID e ID de EX, además del pc. Si el salto el tomado se activa la señal que anula la instrucción que ha entrado después del mismo. Las señales de parada también controlan tres mux que permiten que pasen las señales de control de la instrucción o 0. Estas señales son las que controlan la lectura de memoria y la escritura en banco de registros y memoria. Cada señal de parada controla los de su etapa.

A la unidad de control se ha añadido que proporcione a la nueva instrucción sus respectivas señales de control.

Se ha implementado una unidad de anticipación de operandos para disminuir las paradas en caso de dependencia. Compara los registros que va a utilizar la instrucción que se encuentra en la etapa ID con los destinos de las instrucciones que se encuentran en las etapas EX Y MEM. Según el caso que se de proporciona unas señales de control a los mux que están conectados a las ALU.

**2. Hardware añadido**

**Unidad de anticipación de operandos**

Mediante sentencias concurrentes se establecen valores para las siguientes señales internas:

(Reg\_Rt\_EX = RW\_WB) and RegWrite\_WB = 1 -> Corto\_B\_WB = 1

(Reg\_Rt\_EX = RW\_MEM) and RegWrite\_MEM = 1 -> Corto\_B\_MEM = 1

(Reg\_Rs\_EX = RW\_WB) and RegWrite\_WB = 1 -> Corto\_A\_WB = 1

(Reg\_Rs\_EX = RW\_MEM) and RegWrite\_MEM = 1 -> Corto\_A\_MEM = 1

Si no se cumple la sentencia las señales obtienen el valor 0.

Dicho de otra manera, si cualquiera de los dos registros de la instrucción que se encuentra en la etapa EX coincide con el destino de la instrucción que se encuentra en la etapa WB o MEM y esta escribe en registro, habrá que anticipar el resultado.

Acto seguido se define un process que determina los valores de las salidas de la unidad de anticipación utilizando las señales anteriores:

Corto\_B\_WB = 1 -> MUX\_ctrl\_B = 10

Corto\_A\_WB = 1 -> MUX\_ctrl\_A = 10

Corto\_B\_MEM = 1 -> MUX\_ctrl\_B = 01

Corto\_A\_MEM = 1 -> MUX\_ctrl\_A = 01

Si ninguna de las señales vale 1, MUX\_ctrl\_B y MUX\_ctrl\_A toman el valor 00.

Como en un process las sentencias son secuenciales se colocaban en primer lugar las correspondientes a la etapa WB y después las de la etapa MEM, ya que en caso de que el destino de la instrucción de la etapa WB y el de la etapa MEM coincidieran y en EX se usase debe prevalecer el resultado de la instrucción de MEM, porque es la inmediatamente anterior en el código.

**Unidad de control**

En la unidad de control se ha añadido un nuevo caso al switch de IR\_op\_code, para la instrucción addfp.

Las señales de control toman los siguientes valores:

Branch = 0 (no es un salto)

RegDst = 1 (el destino es Rd)

ALUSrc = 0 (el segundo operando proviene de busB, no el inmediato)

MemWrite = 0 (no escribe en memoria)

MemRead = 0 (no lee de memoria)

MemtoReg = 0 (el resultado no sale de memoria, si no de la ALU)

RegWrite = 1 (escribe en registro)

Update\_Rs = 0 (no se llega a usar nunca)

FP\_add = 1 (propia de la instrucción)

**Unidad de detección de riesgos**

En la unidad de detección de riesgos, se identifican tres riesgos de datos, uno estructural y otro de control.

El riesgo estructural vendría dado por la nueva instrucción addfp, la cual se detiene en EX varios ciclos hasta que obtiene el resultado. Esto implica detener las instrucciones que se encuentran en IF e ID respectivamente y propagar 0 a las etapas siguientes a EX.

Para identificar dicho riesgo, se define una señal interna, llamada Parar\_addfp a la cual se le asigna 0 o 1 de la siguiente manera:

FP\_done = 1 -> Parar\_addfp = 1

FP\_add\_EX = 1 -> Parar\_addfp = 0

En cualquier otro caso se le asigna valor 1.

|  |  |  |
| --- | --- | --- |
| FP\_done | FP\_add\_EX | Parar\_addfp |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

En el código, el primer valor que se observa es el FP\_done, ya que si FP\_add\_EX valiera 1 y FP\_done también Parar\_addfp no tomaría el valor correcto.

Los riesgos de datos vendrían dados por los siguientes casos: ld-uso, realizar un sw y que la instrucción anterior modifique el registro donde está el valor a guardar y realizar un beq y que la instrucción anterior modifique el contenido de los dos registros a comparar.

Para identificar el riesgo de ld-uso, se define la señal interna Parar\_ld\_uso, a la cual se le asigna 0 o 1 de la siguiente manera:

((Reg\_Rs\_ID = RW\_EX or Reg\_Rt\_ID = RW\_EX) and MemRead\_EX = 1) or ((Reg\_Rs\_ID = RW\_Mem or Reg\_Rt\_ID = RW\_Mem) and RegWrite\_Mem = 1) -> Parar\_ld\_uso = 1

En cualquier otro caso se le asigna el valor 0.

Principalmente lo que se hace es observar si cualquiera de los dos registros de la instrucción que está en ID coincide con el destino de la que esta en EX o MEM y si esta lee de memoria o escribe en registro respectivamente.

Para identificar el riesgo que provoca modificar en una de las dos instrucciones anteriores a un sw el dato que va a guardar, se define la señal interna Parar\_dep\_sw, a la cual se le asigna 0 o 1 de la siguiente manera:

(((Reg\_Rt\_ID = RW\_EX) and RegWrite\_EX = 1) or ((Reg\_Rt\_ID = RW\_Mem) and RegWrite\_Mem = 1)) and IR\_op\_code = 000011 -> Parar\_dep\_sw = 1

En cualquier otro caso se la asigna valor 0.

Se observa que el registro donde está el dato a guardar en memoria coincida con el destino de la instrucción que está en EX o MEM y que estas escriban en registro, además de que la instrucción que se encuentra en ID sea un sw.

Para identificar el riesgo provocado por modificar en una de las dos instrucciones anteriores a un salto cualquiera de los dos registros que se utilizan en el mismo, se define la señal interna Parar\_dep\_salto, a la cual se le asigna 0 o 1 de la siguiente manera:

(((Reg\_Rs\_ID = RW\_EX or Reg\_Rt\_ID = RW\_EX) and RegWrite\_EX = 1) or ((Reg\_Rs\_ID = RW\_Mem or Reg\_Rt\_ID = RW\_Mem) and RegWrite\_Mem = 1)) and IR\_op\_code = 000100

Se comprueba que cualquiera de los dos registros de la instrucción que esta en ID coincidan con o bien el destino de la que está en EX o bien con la que está en MEM y que estas escriban en registro, además de que la instrucción que se encuentra en ID sea un salto.

El riesgo de control que se puede dar es el propio salto, se presentan dos opciones: el salto es tomado o no. Si no es tomado no surge ningún problema, el programa sigue con su secuencia normal de PC + 4, pero si es tomado, la instrucción, en secuencia, siguiente al salto habrá entrado a realizar su proceso de ejecución, por tanto, hay que eliminarla. Para ello se utiliza la señal Kill\_IF, la cual toma valor 0 o 1 de la siguiente manera:

PCSrc = 1 -> Kill\_IF = 1

Kill\_IF toma el valor de PCSrc, que indica si se ha tomado el salto o no.

Finalmente, para que Parar\_ID y Parar\_EX, que junto con Kill\_IF son las señales de salida de la unidad detección de riesgos, tomen un valor se realiza lo siguiente:

Parar\_ID = Parar\_ld\_uso and Parar\_addfp and Parar\_dep\_salto and Parar\_dep\_sw

Parar\_EX = Parar\_addfp

Como aclaración, cabe añadir que como estas dos señales implican directamente que los bancos intermedios y pc carguen datos o no, se les asigna 0 en caso de parada, ya que la señal load de estos últimos está activa a 1. Kill\_IF implica el reset del banco que separa las etapas IF e ID, por ello se le asigna valor 1 en caso de salto.

**Mux intermedios, señales de load y reset**

En la etapa ID se añaden tres mux controlados por Parar\_ID, los cuales en caso de parada propagan 0 en vez de las señales de control de la instrucción que se queda detenida en esa etapa. Propagarán 0 cuando se den los casos de parada que activan Parar\_ID.

En la etapa EX se añaden otros tres mux controlados por Parar\_EX, los cuales realizan la misma función que los anteriores, salvo que estos propagarán 0 cuando se de el caso de parada que activa Parar\_EX.

Las tres señales que se propagan a través de estos mux son MemRead, MemWrite y RegWrite, ya que son las que permiten modificar partes importantes como son el banco de registros y la memoria.

Para implementar estos mux se ha creado un archivo vhd nuevo, en el quese ha definido un mux de dos entradas, ambas son de un bit.

ctrl = 1 -> Dout = Din1

ctrl = 0 -> Dout = Din0

La señal de control será Parar\_ID o Parar\_EX dependiendo de la etapa, como se ha mencionado antes. Estos mux seguirán la lógica de las señales de control, si Parar\_ID vale 1, quiere decir que no hay parada, por tanto, se deben propagar las señales de control. Si por el contrario vale 0, deben propagarse los 0.

Además, Parar\_ID y Parar\_EX, supondrán la señal load del banco que separa la etapa IF e ID, del pc y del banco que separa ID de EX.

load\_PC = Parar\_ID and Parar\_EX

load\_ID = Parar\_ID and Parar\_EX

load\_EX = Parar\_EX

Finalmente, el reset del banco que separa IF e ID estará controlado, además de por el reset normal, por Kill\_IF y load\_ID. Una situación de reset vendrá dada por un salto tomado.

reset\_ID = reset or (Kill\_IF and load\_ID)

Se ha definido así porque en caso de que hubiera un salto tomado y antes hubiera habido una parada el valor del pc nuevo del salto se machacaría con el pc + 4 del pc anterior, dando lugar a fallo y el salto no sería efectivo.

**3. Medida de rendimiento**

Programa 1 Programa 2

lw r1, 0(r0) lw r1, 0(r0)

nop; nop; addfp r2, r1, r1

addfp r2, r1, r1 sw r2, 4(r0)

nop; nop;

sw r2, 4(r0)

Estado del MIPS en el programa 1: funcionalidades básicas para que funcione correctamente la instrucción addfp.

Estado del MIPS en el programa 2: MIPS completo, todas unidades implementadas y todas conexiones realizadas.

Programa 1

B D A M E

B D A M E

B D A M E

B D A A A A A M E

B D D D D D A M E

B D A M E

B D A M E

Programa 2

B D A M E

B D D A A A A A M E

B D D D D D D D A M E

CPI(1) = 15 ciclos / 7 inst = 2,1

CPI(2) = 14 ciclos / 3 inst = 4,6

Speedup = 7 \* 2,1 \* Tc / 3 \* 4,6 \* Tc = 1,06

**4. Pruebas**

**1. FP con nops**

Riesgos: addfp

lw r1, 0(r0) r1 = 0,2

nop; nop;

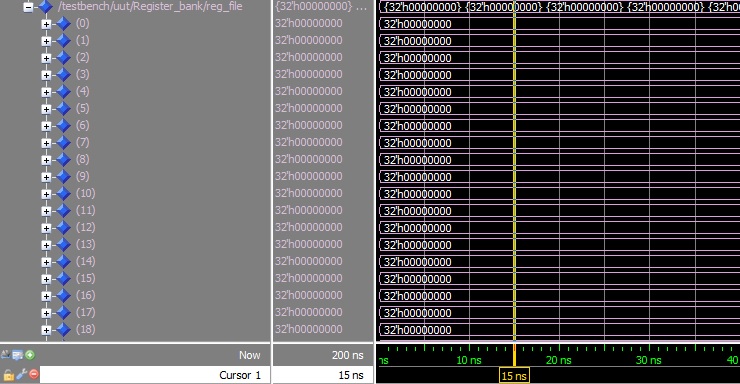
addfp r2, r1, r1 r2 = 0,4

nop; nop;

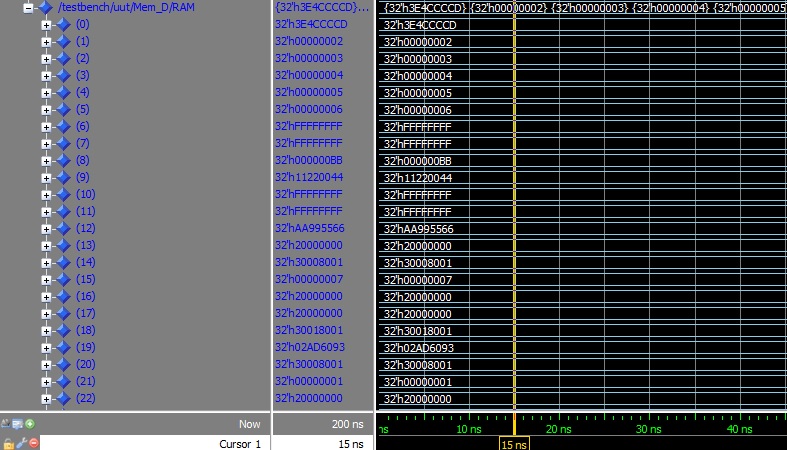
sw r2, 4(r0) Mem(1) = 0,4

nop; nop; nop;

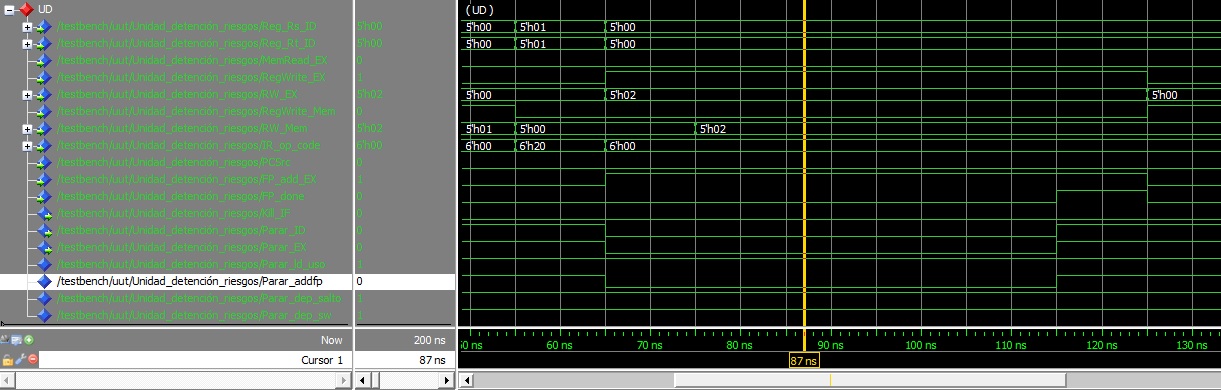
nop; nop;



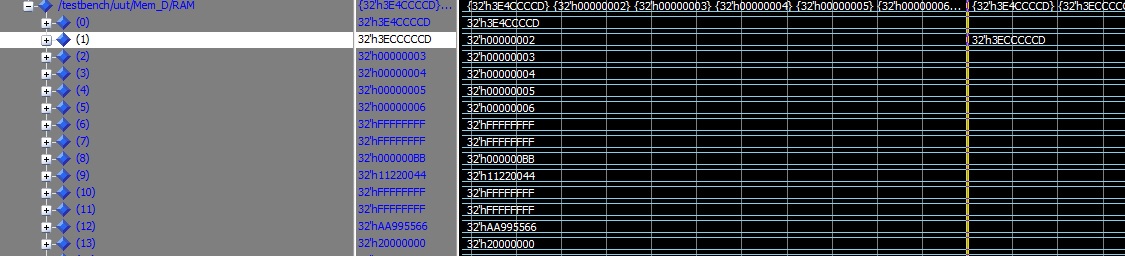
1. Estado inicial del banco de registros



2. Estadio inicial de la memoria



3. Detección riesgo addfp



4. Estado final de la memoria

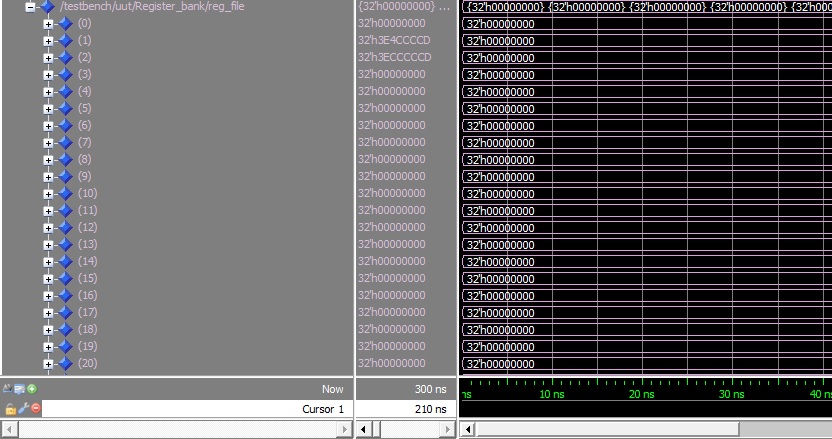
**2. FP sin nops**

Riesgos: ld-uso, addfp, modificar registro dato sw + anticipación

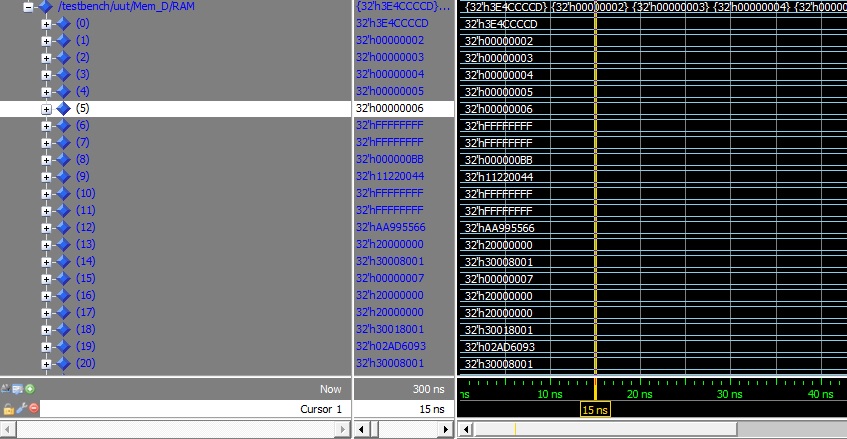
lw r1, 0(r0) r1 = 0,2

addfp r2, r1, r1 r2 = 0,4

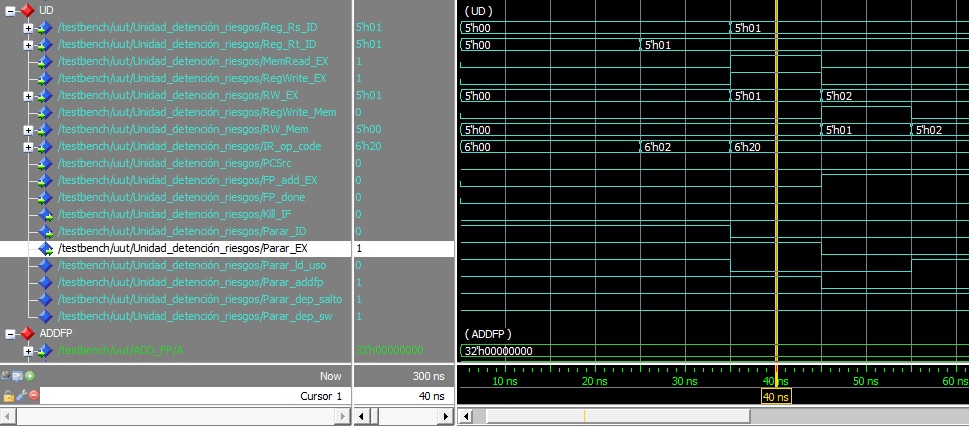
sw r2, 4(r0) Mem(1) = 0,4



1. Estado inicial del banco de registros



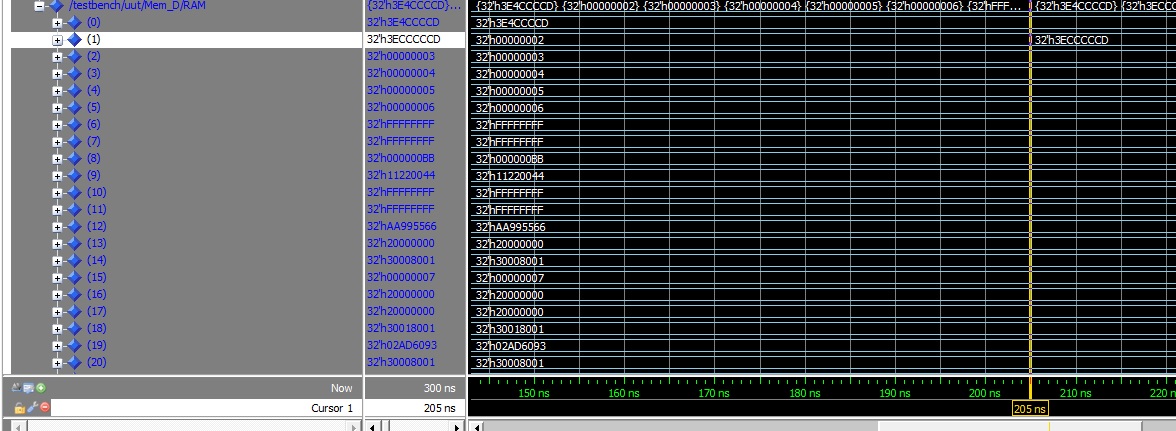
2. Estadio inicial de la memoria



3. Detección del riesgo ld-uso



4. Detección del riesgo provocado por modificar el dato a guardar del sw



5. Estado final de memoria

**3. Salto 1 (salto tomado)**

Riesgos: modificar registro del salto, salto, ld-uso + anticipación

lw r1, 0(r0) r1 = 1

lw r2, 4(r0) r2 = 2

lw r3, 0(r0) r3 = 1

add r1, r3, r1 r1 = 2

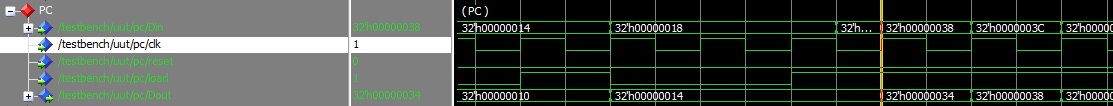
beq r2, r1, #8 r1 = r2

…

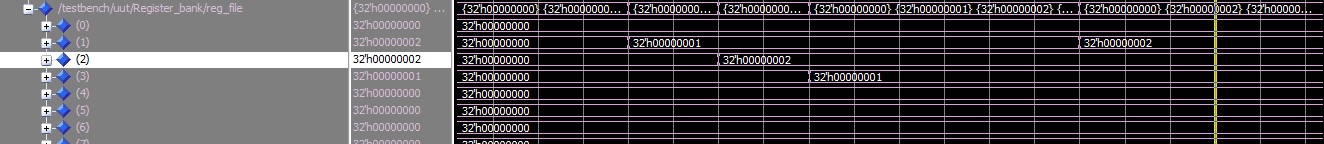
add r3, r2, r1 r3 = 4



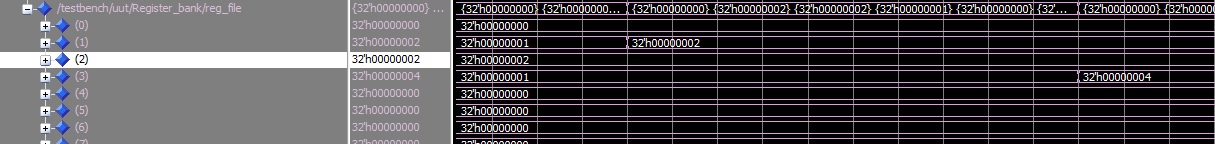
1. Detección de los tres riesgos



2. Cambio en el PC



3. Evolución del banco de registro (1)



4. Evolución del banco de registros (2)

**4. Salto 2 (salto tomado + salto no tomado)**

lw r1, 0(r0) r1 = 1

lw r2, 4(r0) r2 = 2

lw r3, 0(r0) r3 = 1

add r1, r3, r1 r1 = 2

lw r4, 8(r0) r4 = 0,2

addfp r5, r4, r4 r5 = 0,4

beq r2, r1, #8 r2 = r1

…

add r2, r1, r1 r2 = 4

beq r2, r3, #8 r2 != r3

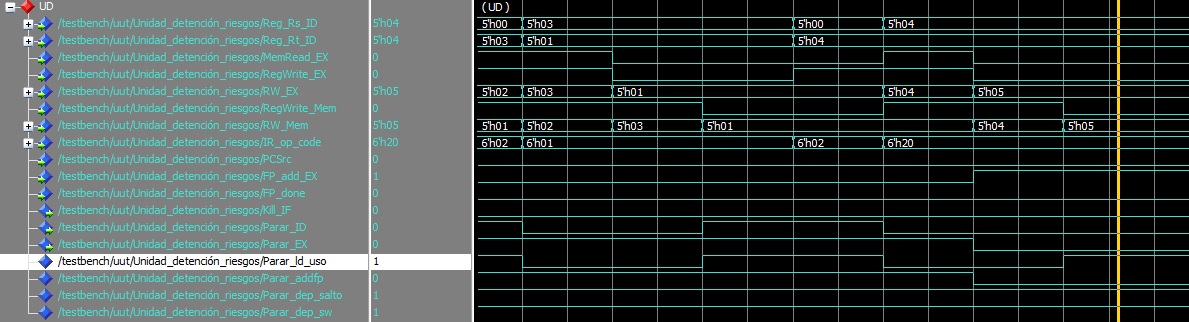
sw r1, 0(r0) Mem(0) = 2

sw r2, 4(r0) Mem(1) = 4

sw r3, 8(r0) Mem(2) = 1

sw r5, 12(r0) Mem(3) = 0,4

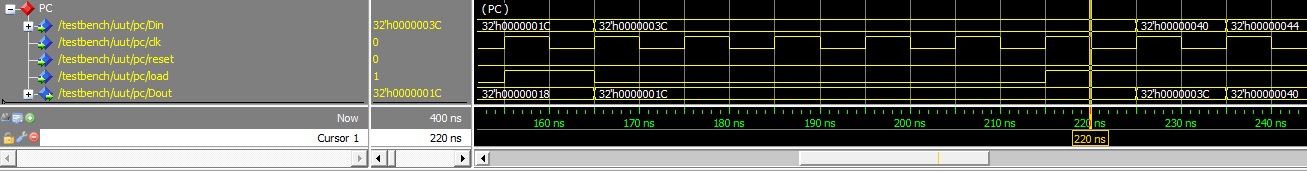
Riesgos: salto, modificar registro del salto, ld-uso, addfp



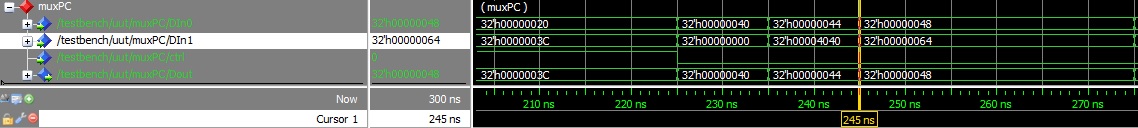
1. Detección riesgo ld-uso



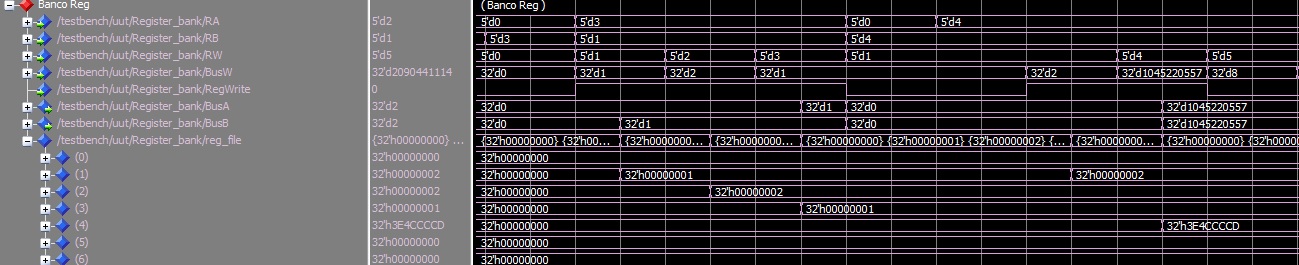
2. Detección riesgo addfp y salto



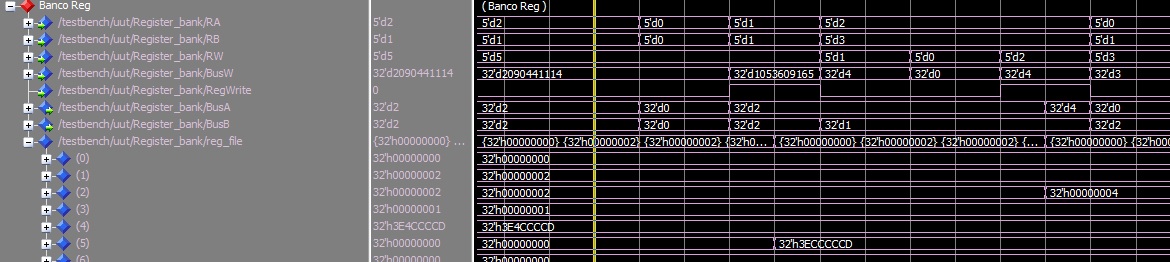
3. Cambio en el PC por salto tomado



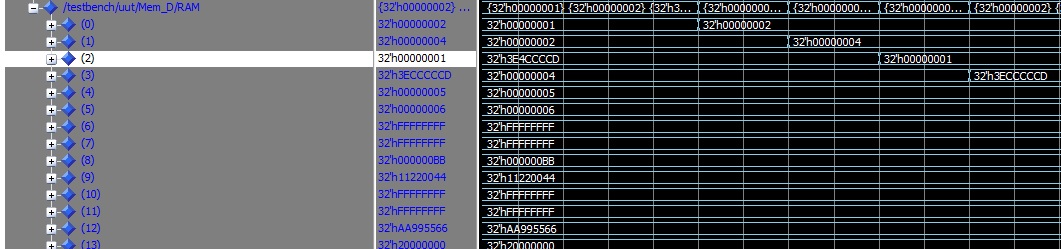
4. Salto no tomado



5. Evolución del banco de registros (1)



6. Evolución del banco de registros (2)



7. Estadio final de memoria

**5. Conclusiones**

El proyecto lo he realizado en toda su dimensión solo, lo cual no me ha permitido desarrollar algunas partes como me hubiera gustado por la falta de tiempo.

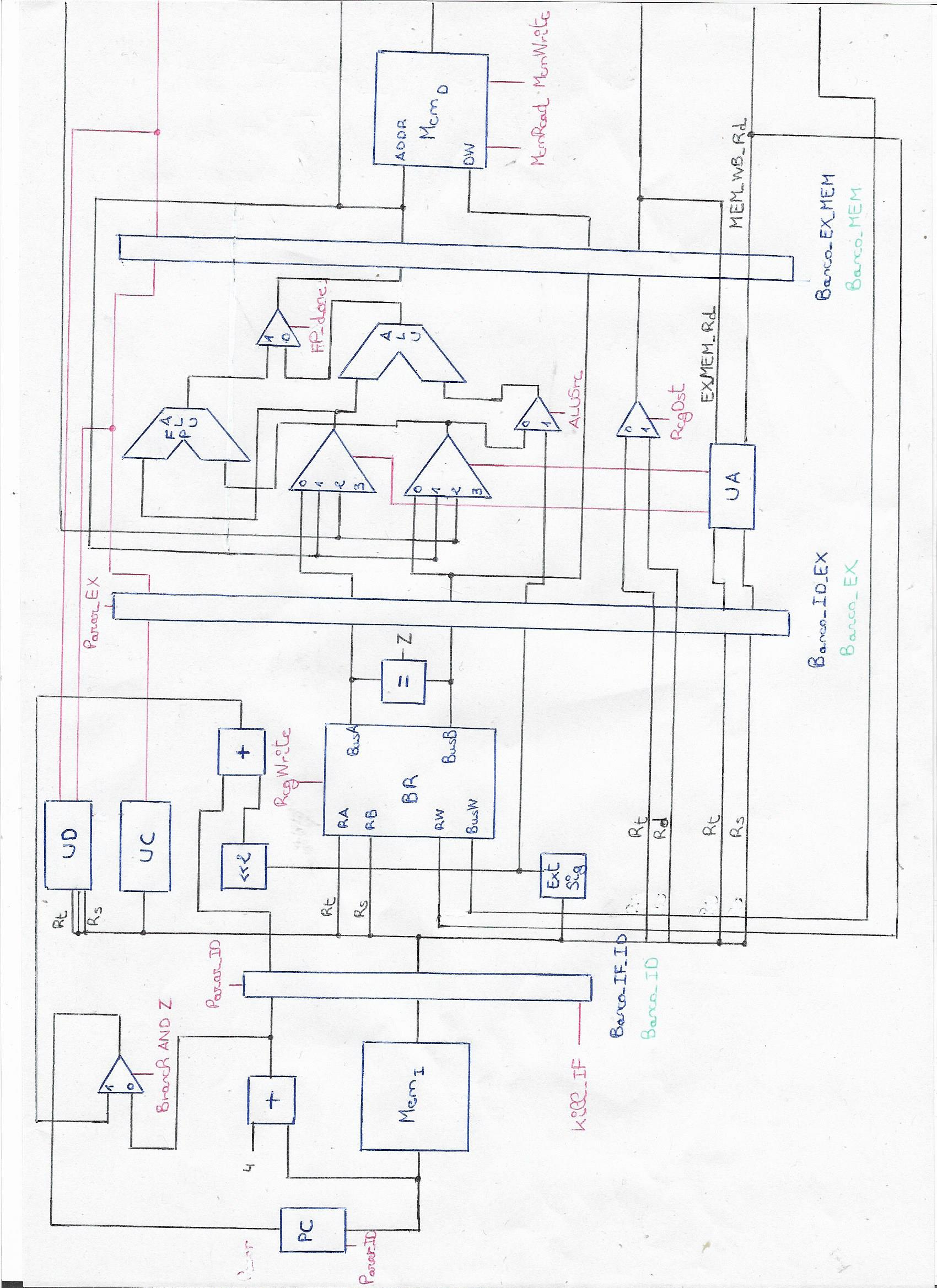
Entender el funcionamiento del simulador y cómo funcionaba el MIPS dentro de él me llevo más tiempo de lo esperado, unas 6 horas. Realizar esquemas en papel para entender que faltaba por completar u añadir, otras 2 horas.

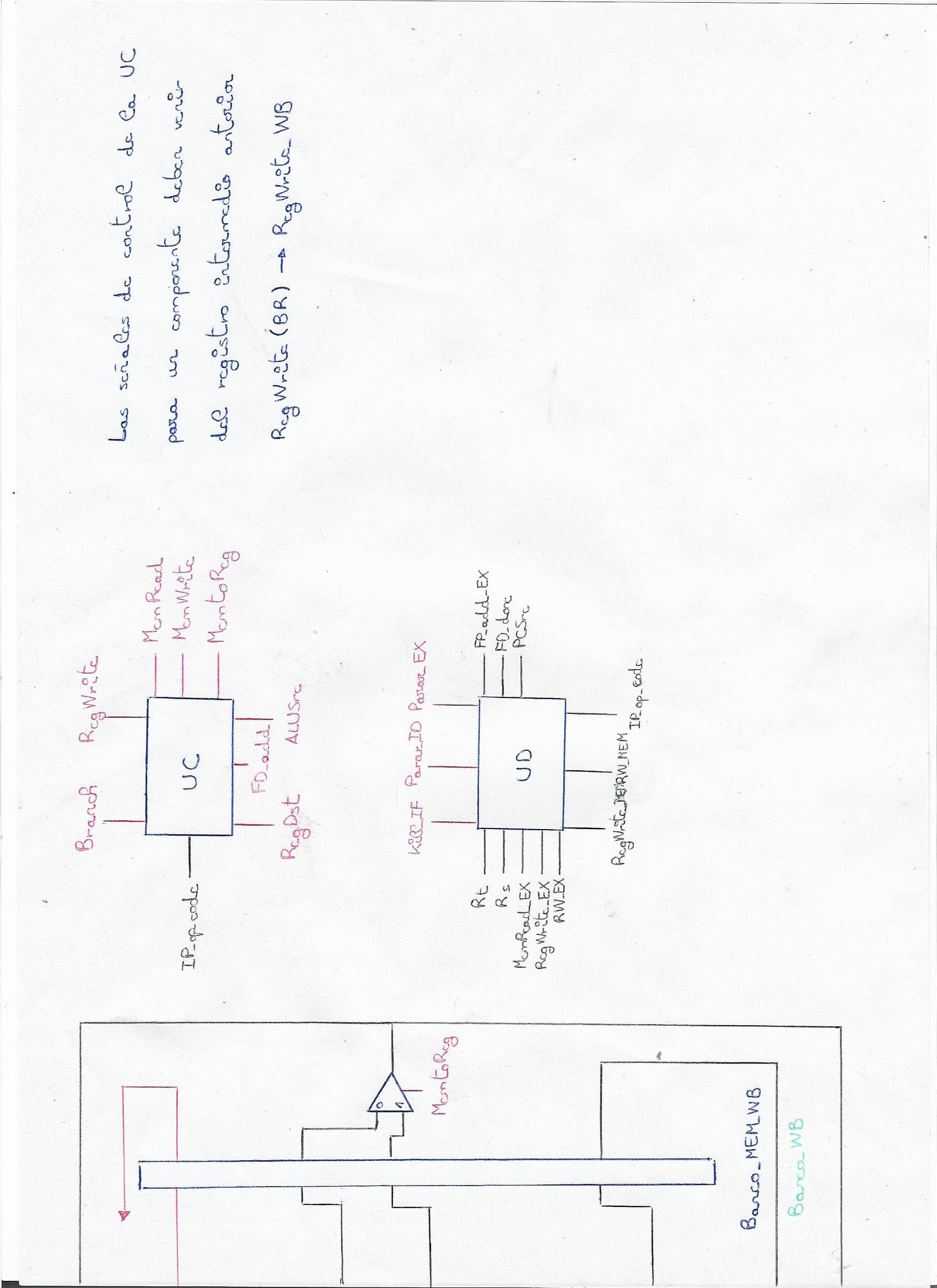
Realizar el código necesario (unidad de anticipación, de detección de riesgos, de control y conexiones) me ha llevado en torno a 2 horas, contando correcciones realizadas.

Una vez estaba hecho el código pasaba a probarlo con las pruebas incluidas en está memoria, repetidas veces. Lo más costoso ha sido, una vez encontrado un fallo, hallar el foco del problema. Algunos casos han costado más que otros, por ejemplo, cuando se tomaba un salto, muxPC permitía el paso del PC calculado en el mismo y si estaba activada la señal de parada Parar\_ID, el PC + 4 de la instrucción anterior al salto lo machacaba (muxPC le permitía pasar). Este fallo era debido a que la señal de reset de banco\_IF\_ID era directamente Kill\_IF. Encontrar ha que era debido esto me llevo unas 4 horas. En total habré dedicado unas 12 horas a pruebas (realización de las mismas + encontrar fallos + correcciones). A la memoria se le habrán dedicado unas 3 horas aproximadamente.

En definitiva, que algunas partes me hayan llevado más tiempo de lo esperado, me ha lastrado en otras, provocando que no muestren un resultado óptimo.

**6. Anexo**

****

****

**7. Correcciones**

**Unidad de anticipación de operandos**

Se ha sustituido el process que daba valores a las salidas Mux\_ctrl\_B y Mux\_ctrl\_A por dos acciones concurrentes.

MUX\_ctrl\_B <= "01" when Corto\_B\_Mem = '1' else

"10" when Corto\_B\_WB = '1' else

"00";

MUX\_ctrl\_A <= "01" when Corto\_A\_Mem = '1' else

"10" when Corto\_A\_WB = '1' else

"00";

**Unidad de detección de riesgos**

La parada por un riesgo de load-uso se ha cambiado teniendo ahora solo en cuenta la instrucción que se encuentra en la etapa EX y además que la instrucción que se encuentra en la etapa ID no sea ni una nop ni un lw.

Parar\_ld\_uso <= '0' when ((Reg\_Rs\_ID = RW\_EX or Reg\_Rt\_ID = RW\_EX) and MemRead\_EX = '1') and (IR\_op\_code /= "000000" and IR\_op\_code /= "000010") else

'1';

Se ha añadido un nuevo riesgo, si en la etapa EX y en la etapa D hay un lw y el segundo utiliza como dirección de memoria el contenido del registro sobre el cual el anterior lw ha cargado un dato hay que detener. Este riesgo no se detecta como un load-uso ya que se mira que la operación que está en la etapa ID no sea un lw, para evitar falsos positivos.

Parar\_ld\_ld <= '0' when (Reg\_Rs\_ID = RW\_EX and MemRead\_EX = '1') and IR\_op\_code = "000010" else

'1';

Por último, sea ha simplificado la señal Kill\_IF.

Kill\_IF <= PCSrc;

**Circuito general**

Se ha añadido un nuevo mux en la etapa ID para la señal FP\_add, en caso de parada se propagarán 0, si no pasará el valor de la señal.

Se ha cambiado la entrada del banco que separa las etapas EX y MEM que se dirige a memoria con el dato a guardar. Ahora en vez de conectarse el bus B, se conecta la salida del mux de anticipación B.

Se ha añadido al banco que separa las etapas ID y EX una nueva entrada para el registro Rs.