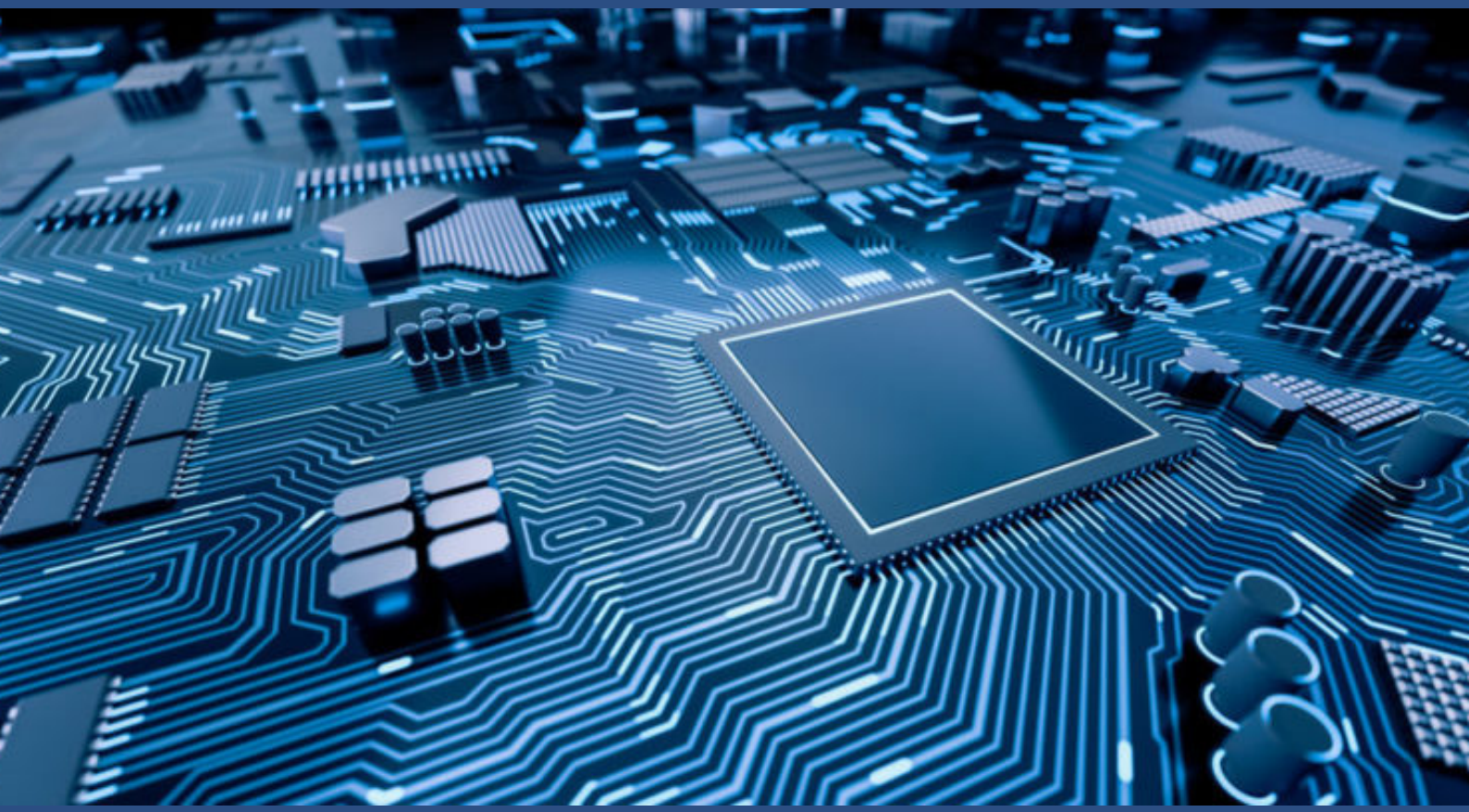
Universidad Zaragoza
1542

Gestión de riesgos en una arquitectura MIPS segmentada



Álvaro Pomar Martínez

ÍNDICE

Modificaciones realizadas

Adición de las nuevas instrucciones a la UC

El primer paso que se llevo a cabo, fue la modificación de la UC para añadir las nuevas instrucciones a utilizar en el MIPS. A continuación, se muestran las modificaciones realizadas.

```
– ADDFP
WHEN  "100001"  =>  Branch <= '0'; RegDst <= '1'; ALUSrc <= '0'; MemWrite <= '0'; MemRead <=
'0'; MemtoReg <= '0'; RegWrite <= '0'; FP_add <= '1'; FP_mem <= '0'; RegWrite_FP <= ‘1';

-- LWFP
WHEN  "100010"  =>  Branch <= '0'; RegDst <= '0'; ALUSrc <= '1'; MemWrite <= '0'; MemRead <=
'1'; MemtoReg <= '0'; RegWrite <= '0'; FP_add <= '0'; FP_mem <= '1'; RegWrite_FP <= '1';
            -- nuevo SWFP
WHEN  "100011"  =>  Branch <= '0'; RegDst <= '0'; ALUSrc <= '1'; MemWrite <= '1'; MemRead <=
'0'; MemtoReg <= '0'; RegWrite <= '0'; FP_add <= '0'; FP_mem <= '1'; RegWrite_FP <= ‘0’;
```

Igual que en las instrucciones que ya estaban creadas, para un código de operación, se especifican los valores de las distintas señales.

Riesgo Estructural

Como la instrucción addfp tiene una latencia variable y ocupa la etapa EX durante un número variable de ciclos, las instrucciones posteriores no deben avanzar mientras se esté realizando la operación.

Para detectar este riesgo, se ha añadido lo siguiente en la Unidad de Detección:

```
Process(FP_add_EX,FP_DONE)
    begin
      IF(FP_add_EX = '1' AND FP_DONE = '0') THEN parar_EX_FP_internal <= '0';
      ELSE parar_EX_FP_internal <= '1';
      end IF;
    end Process;

parar_EX_FP <= parar_EX_FP_internal;
```

Después, en el MIPS segmentado, se añadió el siguiente “multiplexor” para enviar 0’s a las etapas posteriores (MEM Y WB) por la ruta FP mientras la operación esté en marcha. El valor de salida se emplea en el Banco_EX_FP_MEM :

```
RegWrite_FP_EX_mux_out <= (RegWrite_FP_EX AND Parar_EX_FP);

Banco_EX_FP_MEM: Banco_MEM_FP PORT MAP ( ADD_FP_out => ADD_FP_out, ADD_FP_out_MEM =>
ADD_FP_out_MEM, clk => clk, reset => reset, load => '1',
RegWrite_FP_EX => RegWrite_FP_EX_mux_out, RegWrite_FP_MEM => RegWrite_FP_MEM, FP_mem_EX
=> FP_mem_EX, FP_mem_MEM => FP_mem_MEM, RW_FP_EX => RW_FP_EX, RW_FP_MEM => RW_FP_MEM);
```

Además, para que las instrucciones de las etapas anteriores no avancen, se puso como señal de load de los bancos de registros la señal Parar_EX_FP así como en el load del PC para que no se sigan leyendo instrucciones.

Riesgo de Datos

En primer lugar, se desarrolló la Unidad de Anticipación (UA) para la ruta entera. La función de la UA es anticipar los operandos a las instrucciones que los necesitan mediante dos buses, uno que va de la etapa WB a EX y otro de MEM a EX.

```
MUX_ctrl_A <= "01" WHEN (Corto_A_Mem = '1')
         ELSE "10" WHEN (Corto_A_WB = '1' AND Corto_A_Mem = '0')
         ELSE "00";
Corto_A_WB <= '1' WHEN (RegWrite_WB = '1' AND RW_WB = Reg_Rs_EX) ELSE '0';
Corto_A_Mem <= '1' WHEN (RegWrite_MEM = '1' AND RW_MEM = Reg_Rs_EX) ELSE '0';

MUX_ctrl_B <= "01" WHEN (Corto_B_Mem = '1')
         ELSE "10" WHEN (Corto_B_WB = '1' AND Corto_B_Mem = '0')
         ELSE "00";
Corto_B_WB <= '1' WHEN (RegWrite_WB = '1') AND (RW_WB = Reg_Rt_EX) ELSE '0';
Corto_B_Mem <= '1' WHEN (RegWrite_MEM = '1') AND (RW_MEM = Reg_Rt_EX) ELSE '0';
```

Luego, estas señales son las encargadas de controlar los multiplexores Mux_A y Mux_B como se puede comprobar en la imagen.

```
Mux_A: mux4_1_32bits port map  ( DIn0 => BusA_EX, DIn1 => ALU_out_MEM, DIn2 => busW,
DIn3 => cero, ctrl => MUX_ctrl_A, Dout => Mux_A_out);

Mux_B: mux4_1_32bits port map  ( DIn0 => BusB_EX, DIn1 => ALU_out_MEM, DIn2 => busW,
DIn3 => cero, ctrl => MUX_ctrl_B, Dout => Mux_B_out);
```

Con la UA implementada, los únicos riesgos de datos que pueden darse en la ruta entera son el load uso y cuando hay una instrucción productora seguida de un BEQ en el que rs o rt son consumidores. Para detectar estos casos, se ha empleado las señales internas de la UD ld_uso_rt, ld_uso_rs, beq_rt y beq_rs como se puede ver a continuación.

```
Process(MemRead_Ex,RegWrite_EX,Reg_Rs_ID,RW_EX,IR_op_code)
begin
   IF(MemRead_Ex = '1' AND RegWrite_EX = '1' AND Reg_Rs_ID = RW_EX)
   THEN ld_uso_rs <= '1';
   ELSE ld_uso_rs <= '0';
   end IF;
end Process;

Process(IR_op_code,RegWrite_EX,RegWrite_Mem,Reg_Rs_ID,RW_EX,RW_Mem)
begin
   IF((IR_op_code = BEQ AND RegWrite_EX = '1' AND Reg_Rs_ID = RW_EX) OR
     (IR_op_code = BEQ AND RegWrite_Mem = '1' AND Reg_Rs_ID = RW_Mem))
   THEN BEQ_rs <= '1';
   ELSE BEQ_rs <= '0';
   end IF;
end Process;
```

Los proceso de ld_uso_rt y beq_rt son iguales pero empleando la señal Reg_Rs_Rt

Por otra parte, la ruta de punto flotante, al no tener UA, todas las dependencias de datos se convierten en riesgos. Para detectar estos riesgos, se ha desarrollado el siguiente código.

```
Process(RegWrite_FP_EX,Reg_Rs_ID,RW_FP_EX,FP_inst)
begin
   IF(FP_inst = '1' AND RegWrite_FP_EX = '1' AND Reg_Rs_ID = RW_FP_EX)
   THEN dep_rs_EX_FP <= '1';
   ELSE dep_rs_EX_FP <= '0';
   end IF;
end Process;


Process(RegWrite_FP_MEM,Reg_Rs_ID,RW_FP_MEM,FP_inst)
begin
   IF(FP_inst = '1' AND RegWrite_FP_MEM = '1' AND Reg_Rs_ID = RW_FP_MEM)
   THEN dep_rs_MEM_FP <= '1';
   ELSE dep_rs_MEM_FP <= '0';
   end IF;
end Process;

riesgo_rs_FP <= '1' WHEN (dep_rs_EX_FP = '1' OR dep_rs_MEM_FP = '1') ELSE '0';
riesgo_rt_FP <= '1' WHEN (dep_rt_EX_FP = '1' OR dep_rt_MEM_FP = '1') ELSE '0';
```

Los proceso de dep_rt_EX_FP y dep_rt_MEM_FP son iguales pero empleando la señal Reg_Rs_Rt

Por último, se le da valor a la señal Para_ID que es la que saldrá de la UD. Es decir, la señal se activará (se activa a 0) cuando se de cualquiera de los riesgos anteriores y no haya un riesgo estructural, esto último se ha hecho para que no estén activas las dos señales a la vez innecesariamente.

```
Parar_ID <= '0' WHEN ((ld_uso_rs = '1' OR ld_uso_rt = '1' OR BEQ_rs = '1' OR BEQ_rt =
'1' OR riesgo_rs_FP = '1' OR riesgo_rt_FP = '1') AND parar_EX_FP_internal = '1')
ELSE '1';
```

Una vez detectados los riesgos, para inyectar a la etapa EX la burbuja, se ha realizado lo siguiente:

```
RegWrite_FP_ID  <= RegWrite_FP and Parar_ID;
RegWrite_ID <= RegWrite and Parar_ID;
FP_add_ID   <= FP_add and Parar_ID;
MemWrite_ID <= MemWrite and Parar_ID;
MemRead_ID  <= MemRead and Parar_ID;
```

Además, la señal Parar_ID se ha empleado como load junto con Parar_EX_FP en el banco de registros Banco_IF_ID y en el pc . Para ello se ha tenido que definir una nueva señal, load_ID_EX_FP.

```
load_ID_EX_FP <= '0' WHEN (Parar_ID = '0' OR Parar_EX_FP = '0') ELSE '1';
load_PC <= load_ID_EX_FP
```

Riesgo de control

Al asumir que los saltos son no tomados, cuando un salto es tomado, se debe matar la última instrucción leída de la memoria de instrucciones y sustituirla por una nop. Para ello, en primer lugar se debe de detectar en la UD que el salto ha sido tomado. Esto puede comprobarse de la siguiente forma (Kill_IF se activa a 0).

```
Kill_IF <= '0' WHEN (PCSrc = '1' AND IR_op_code = BEQ AND BEQ_rs = '0' AND BEQ_rt = '0')
           ELSE '1';
```

Una vez detectado que el salto ha sido tomado, para matar la última instrucción leída, se ha puesto la señal Kill_IF en la señal RE de la memoria de instrucciones, esto se ha hecho puesto que en la especificación del componente *memoriaRAM_I* se explica que si RE es 0 la salida serán todo 0's.

```
Mem_I: memoriaRAM_I PORT MAP (CLK => CLK, ADDR => PC_out, Din => cero, WE => '0',
                              RE => Kill_IF, Dout => IR_in);
```

Programas de prueba

Para la comprobación del correcto funcionamiento del MIP's, se decidió realizar pequeños programas que comprobaran las distintas dependencias por separado. Primero se verificó la parte entera y posteriormente la flotante.

Una vez se obtuvieron los resultados deseados con dichos programas, se realizaron unas pruebas más complejas, intercalando la parte entera y flotante junto con todo tipo de dependencias/riesgos.

Pruebas para la ruta entera

Programas que comprueban el correcto funcionamiento de la Unidad de Anticipación:

- Funcionamiento del Corto **W -> E** y **M -> E**

```
@0x0    LW  R1, 0(R0)  ; r1 <- 0x8        F D E M W
@0x4    LW  R2, 4(R0)  ; r2 <- 0xC          F D E M W
@0x8    nop                                   - - - - -
@0xC    ADD R3, R1, R2 ; r3 <- 0x14             F D E M W
@0x10   ADD R4, R3, R3 ; r4 <- 0x28               F D E M W
```

- Comprobación de la precedencia de operandos. El último add emplea el valor de R3 proporcionado por el add anterior (@0x10) y no por el primero (@0xC).

```
@0x0    LW  R1, 0(R0)  ; r1 <- 0x8        F D E M W
@0x4    LW  R2, 4(R0)  ; r2 <- 0xC          F D E M W
@0x8    nop                                   - - - - -
@0xC    ADD R3, R1, R2 ; r3 <- 0x14             F D E M W
@0x10   ADD R3, R3, R3 ; r4 <- 0x28               F D E M W
@0x14   ADD R3, R3, R3 ; r4 <- 0x50                 F D E M W
```

A continuación, se va a mostrar el programa que se ha empleado para comprobar si la señal **Kill_IF** se activa correctamente. En este caso, la instrucción de después del BEQ, es decir, ADD R4, R3, R3 se lee siempre después del BEQ pero nunca llega a ejecutarse debido a que el salto es tomado.

```
@0x0    LW  R1, 0(R0)  ; r1 <- 0x8        F D E M W
@0x4    LW  R2, 4(R0)  ; r2 <- 0xC          F D E M W
@0x8    nop                                   - - - - -
@0xC    ADD R3, R1, R2 ; r3 <- 0x14             F D E M W
@0x10   ADD R3, R3, R3 ; r4 <- 0x28               F D E M W
@0x14   BEQ R0, R0, -2                              F D E M W
@0x18   ADD R3, R3, R3 ; r4 <- 0x60                   F - - - -
```

Tal y como se ha mencionado anteriormente, la señal **Parar_ID** en la ruta entera solo se debe activar en dos casos, cuando hay una dependencia load-uso y cuando hay una instrucción productora seguida de un BEQ en el que rs o rt son consumidores.

- Prueba load-uso:

```
@0x0    LW  R1, 0(R0)  ; r1 <- 0x8        F D E M W
@0x4    LW  R2, 4(R0)  ; r2 <- 0xC          F D E M W
@0x8    ADD R3, R1, R2 ; r3 <- 0x14           F D D E M W
```

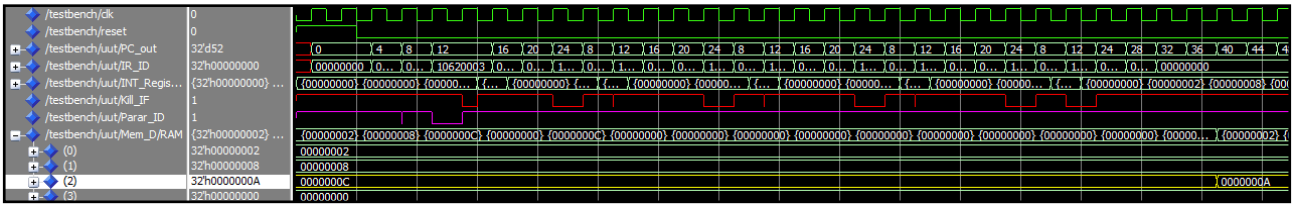
- Instrucción productora seguida de un BEQ en el que rs o rt son consumidores + load-uso.

```
@0x0    LW  R1, 0(R0)  ; r1 <- 0x8        F D E M W
@0x4    LW  R2, 4(R0)  ; r2 <- 0xC          F D E M W
@0x8    ADD R3, R1, R2 ; r3 <- 0x14           F D D E M W
@0xC    ADD R4, R1, R2 ; r3 <- 0x14             F F D E M W
@0x10   BEQ R3, R4, -2                              F D D D E M W
```

Por último, se muestra una programa en el que se comprueba el correcto funcionamiento de todo lo mencionado anteriormente, es decir, UA+ Kill_if + Parar_ID. Para facilitar la corrección, este es el único programa de los hasta ahora mostrados que se ha entregado debido a que resumen todas las pruebas previas (disponible en el fichero *memoriaRAM_I_test1.vhd*).

```
@0x0    LW  R2, 4(R0)   ; r2 <- 0x8         F D E M W
@0x4    LW  R1, 0(R0)   ; r1<- 0x2            F D E M W
@0x8    beq R3, R2, 3                            F D D M W
@0xC    ADD R3, R3, R1  ; r3 <- r3 + 2             F D E M W
@0x10   ADD R4, R3, R1  ; r4 <- r3 + 2               F D E M W
@0x14   beq R0, R0, -4                                 F D E M W
@0x18   SW  R4, 8(R0)   ; M[2] <- 0xA                    F - - - -
                                                 2ª iter     F D E M W
                                                                .. .. .. ..
                                                 última iter    F D E M W
```

A continuación, se muestra una captura de ModelSim en la que se comprueba el correcto funcionamiento del programa. Se puede observar que el resultado almacenado en memoria es el esperado y que las señales Kill_IF y Parar_ID se activan cuando es necesario.



Pruebas para la ruta entera + FP

En este programa se comprueba el correcto funcionamiento de la señal **Parar_EX_FP** (se activa cuando hay una instrucción addfp) así como de la señal Parar_ID.

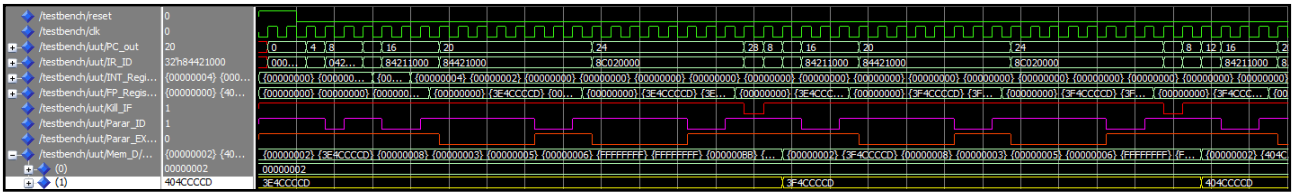
```
@0x0 LWfp R1, 0(R0)          ; R1<- 0.2
@0x4 ADDfp R2fp, R1fp, R1fp  ; R2fp <- 0.4
@0x8 SWfp R2fp, 4(R0)        ; M[4] <- 0.4
```

Por último, este programa (*memoriaRAM_I_FP_test2.vhd*) comprueba el funcionamiento correcto del MIPS segmentado cuando se intercalan instrucciones enteras con fp. Mediante este programa, se ha reafirmado que funcionan adecuadamente la Unidad de Anticipación así como todas las señales de la Unidad de Detección.

```
@0x0 LW R1, 0(R0) ; R1<- 2
@0x4 ADD R0, R1, R1 ; r0 <- 4
@0x8 LWfp R1FP, 0(R0) ; R1FP<- 0.2
@0xc ADDfp R2fp, R1fp, R1fp ; r2fp <- 0.4
@0x10 ADDfp R2fp, R2fp, R2fp ; r2fp <- 0.8
@0x14 SWfp R2fp, 0(R0) ; M[4] <- 0.8
@0x18 beq r0, r0, inm
@0x1C SWfp R1fp, 8(R0) ; M[4] <- 0.2
```

```
F D E M W
  F D D E M W
    F F D E M W
      F D E E E E E M W
        F D D D D D D E E E E E M W
          F F F F F F F D D D D D D D E M W
                        F F F F F F F D E M W
                                    F - - - -
```

Se puede observar en el cronograma como al no haber anticipación, la señal Parar_ID se activa muchas mas veces que en un programa solo de instrucciones enteras.Además, Parar_EX_FP dura un número X de ciclos en principio desconocidos. Se adjunta una captura de las señales obtenidos en ModelSim para una parte de la ejecución de este último programa.



Impacto en rendimiento

Para evaluar el impacto en rendimiento de la gestión de riesgos diseñada, se va a calcular el speedup al ejecutarse el programa de prueba mostrado en la imagen con anticipación y sin anticipación. El programa se puede encontrar en la entrega bajo el nombre *memoriaRAM_I_FP_test3.vhd* .

```
@0x0     LW R2, 4(R0)     ; R2 <- 0x8
@0x4     LWFP R1FP, 0(R2) ; R1FP <- 0.2
@0x8     LW R1, 0(R0)     ; R1<- 0x2
@0xC     beq R3, R2, 4
@0x10    ADD R3, R3, R1   ; R2 <- R3 + 2
@0x14    ADD R4, R3, R1   ; R4 <-R3 + 2
@0x18    ADD R4, R3, R4   ; R4 <-2 * (R3 + 2)
@0x1C    beq R0, R0, -5
@0x20    SW R4, 8(R0)     ; M[2] <- 0x14
@0x24    SWFP R1FP, 0(R4)
```

Cronograma para el MIPS con anticipación:

```
F D E M W
  F D D E M W
    F F D E M W
        F D E M W
          F D E M W
            F D E M W
              F D E M W
                F - - - -
2a iter           F D E M W

                      .. .. .. ..

últimas 2 instrucciones   F D E M W
                            F D E M W
```

Cronograma para el MIPS sin anticipación:

```
F D E M W
  F D D D E M W
    F F F D E M W
          F D E M W
            F D D E M W
              F F D D D E M W
                  F F F D D D E M W
                        F F F D E M W
                              F - - - -
2a iter                         F D E M W

                                   .. .. .. ..

últimas 2 instrucciones                F D D E M W
                                         F F D E M W
```

$$CPI = \frac{43\ ciclos}{21\ instrucciones} = 2{,}05\ (CPI\ sin\ anticipacion)$$

$$CPI' = \frac{25\ ciclos}{21\ instrucciones} = 1{,}19\ (CPI\ con\ anticipacion)$$

$$Sup(Tex) = \frac{Tex}{Tex'} = \frac{I * CPI * Tc}{I' * CPI' * Tc'} = \frac{2{,}05}{1{,}19} = 1{,}72$$

Se puede observar la cercanía a un CPI = 1 cuando se emplea la UA ya que se reducen el número de dependencias que se convierten en riesgos. Sin embargo, el CPI se desvía en gran medida de 1 cuanado se quita la UA, ya que todos las dependencias se convierten en riesgos y se retrasa la ejecución del programa significativamente.

Esfuerzos invertidos

Tarea	Raúl López	Álvaro Pomar
Estudio enunciado, simulación etc	3h	3h
Añadir nuevas instrucciones y riesgo estructural	1h 30min	1h 30min
Riesgo datos y control	1h 30min	1h 30min
Depuración (pruebas) y ajustes	16h	16h
Memoria	3h	3h

Conclusiones y autoevaluación

A pesar de que ya se había ahecho un estudio previo al proyecto del MPS segmentado para entender su funcionamiento de forma adecuada, este proyecto a servido para asentar estos conocimientos y para que ya no quede ningún tipo de duda acerca de como funciona. Consideramos que ha sido un proyecto muy útil para terminar la primera parte de la asignatura ya que engloba varios de los aspectos vistos en las clases teóricas.

Con respecto a los miembros de este equipo, se ha trabajado de manera correcta, metódica, continua y no ha surgido ningún tipo de conflicto durante la realización del proyecto. Ambos integrantes han dedicado el mismo número de horas y el MIPS segmentado ha sido comprendido igual de bien por los dos.