Informe de la práctica 4

Introducción

En esta 4ª práctica de Introducció als Ordinadors encontraremos los objetivos, los ejercicios planteados y conclusiones:

- *Visualizar los diferentes pasos que debe hacer un microprocesador para ejecutar una instrucción.*

Ejercicio guiado

Debemos ejecutar los siguintes códigos, ciclo a ciclo (microinstrucción a microinstrucción):

Código 1

```
.data
resultat: .word 0
.text

main:
        add a3, zero, zero
        add a7, zero, zero
        addi a2, zero, 4
        add a3, a2, a3
        addi a7, a7, -1
        bgt a7, zero, salta

salta:
        la a0, resultat
        sw a3, 0(a0)
```

Código 2

```
.data
resultat: .word 0
.text

main:
        add a3, zero, zero
        add a7, zero, zero
        addi a2, zero, 4
        add a3, a2, a3
        addi a7, a7, -1
        la a0, resultat
        sw a3, 0(a0)
```

1) Antes de ejecutar los códigos trata de descubrir su función. ¿Hacen lo mismo? ¿Crees que tardará el mismo número de ciclos en ejecutarse?

A priori parece que los códigos sí hacen lo mismo debido a su similitud de código, pero no será así.

No, ya que el código 1 tardarás 1 ciclo más que el código 2 por la instrucción BGT.

2) Vete a la ventana del Ripes dedicada al procesador (Processor). Busca entre las opciones del Simulator control, la Pipeline table.

3) Ejecuta los dos códigos ciclo a ciclo fijandote como las instrucciones van pasando por las diferentes etapas del pipeline. Cuenta el número de ciclos que se necesitan para ejecutar cada uno de los códigos y compara las Pipeline tables.

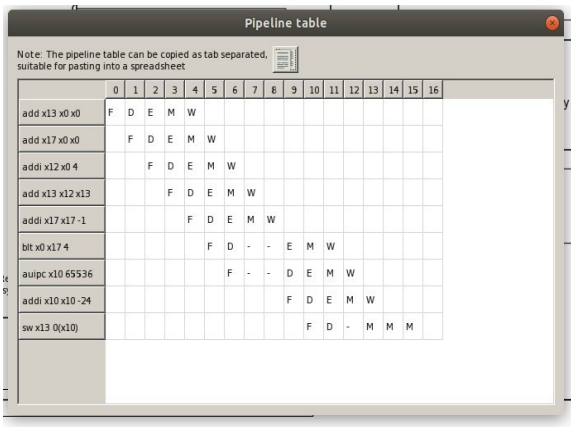

Ilustración 1: Pipeline del código 1

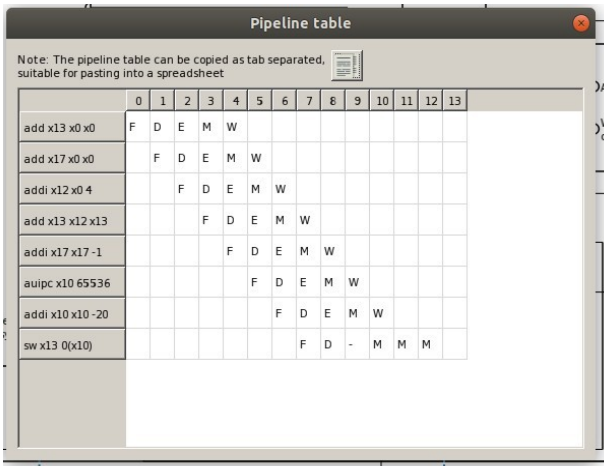

Ilustración 2: Pipeline del código 2

El código 1 tardará 16 ciclos y el código 2, 13 ciclos.

De primeras, ambos códigos aparentan ser muy similares, pero una vez hecho el pipeline observamos diferencias notables devidas a la condición de salto bgt a7, zero, salta. En Ripes, las condiciones de salto como pueden ser bgt o blt necesitan 2 ciclos de reloj nop(stall), que en el pipeline se identifican con el símbolo «-». Esto se debe a que debemos esperar a que se actualicen los datos necesarios para comprobar si se puede o no realizar el salto condicional.

4) ¿Qué significan los signos '-' que aparecen a las Pipeline tables?

Significan «no operación» o nop(stall), es decir, el procesador ha mandado parar la ejecución porque no puede calcular de manera correcta la dirección de memoria a donde saltar. Estará parado hasta tener el valor correcto del registro que necesitas.

5) ¿Cómo afectaría al número total de ciclos de ejecución el siguiente cambio en el código:

Código 1

```
.data
Resultat: .word 0
.text
main:
        add a3, zero, zero
        add a7, zero, zero
        addi a2, zero, 4
        add a3, a2, a3
        addi a7, a7, -1
        bgt a7, zero, salta
salta:
        la a0, Resultat
        sw a3, 0(a0)
```

Código 3

```
.data
Resultat: .word 0
.text
main:
        la a0, Resultat
        add a3, zero, zero
        add a7, zero, zero
        addi a2, zero, 4
        add a3, a2, a3
        addi a7, a7, -1
        sw a3, 0(a0)
```

En el código 1 hacemos 16 ciclos, mientras que en el 3 sólo 13. Hay que tener en cuenta que en los últimos comando MMM es un bug del simulador Ripes. Entonces, suponemos que el resultado del pipeline es el mostrado en la ilustración 3:

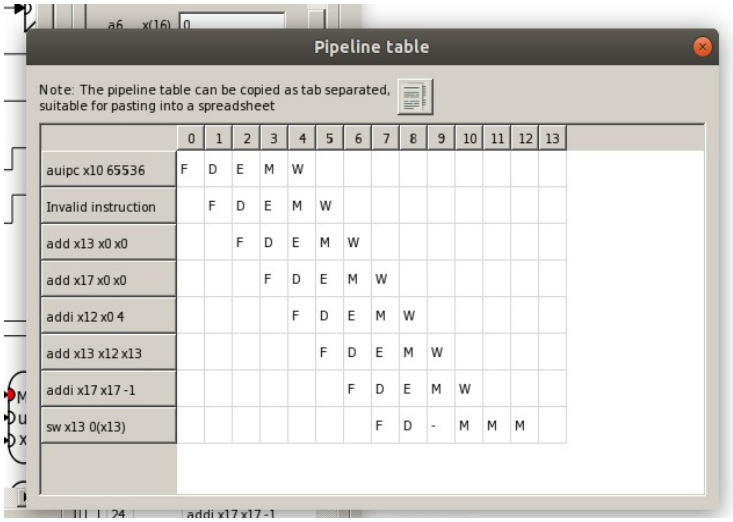

Ilustración 3: Pipeline del código 3

Realización de la práctica

Ripes: Ejecuta el siguiente programa microinstrucción a microinstrucción:

Código 4

```
.data
valorDada: .word 2          # valor que multiplicaremos por 9
guardaResultat: .word 0       # y donde guardaremos el resultado
.text
main:
        lw a7, valorDada           # cargamos el valor a7 => 2
        addi a2, zero, 9          # cargamos el valor a2 => 0+9 => 9
        add a3, zero, zero       # cargamos el valor a3 => 0+0 => 0
loop:
        add a3, a2, a3              # sumamos  a3 => a2 + a3
        addi a7, a7, -1             # restamos a7 => a7 - 1
        bgt a7, zero, loop        # mientras a7 sea mayor que cero, loop
        la a0, guardaResultat   # salimos del loop y guardamos el
        sw a3, 0(a0)          # resultado en la posicion 0(x10)
```

Preguntas sobre el simulador RIPES:

Para resolver estas preguntas es necesario mirar el estado del pipeline o utilizar la Pipeline table:

1) ¿Cuál es el estado de cada una de las cinco etapas del pipeline al ciclo 6? ¿Y al 8?

En el ciclo 6, tenemos los estados: Writeback (instrucción addi x12 x09), Memory Access (add x13 x0 x0), Execute (add x13 x12 x13), Decode (addi x17 x17 -1) y Fetch (blt x0 x17 -8, que se corresponde con el bgt).

En el ciclo 8, tenemos los estados: Writeback (instrucción add x13 x12 x13), Memory Access (addi x17 x17 -1), - (blt x0 x17 8) y otra vez - (auipc x10 65536).

2) ¿Qué señales de control se activan en el ciclo 4? ¿A qué instrucciones del código corresponden?

Se activan las señales del PC (0x14, corresponde al add x13 x12 x13), el Enable del Fetch/Decode (0xd6b3, add x13 x12 x13), el Write del Registers (0x1, add x13 x0 x0) y el Read del Data Memory (0x2, lw x17 0(x17)).

3) ¿Cuáles son los valores en las salidas de los multiplexores señalados a la figura en el ciclo 7?

Del multiplexor de arriba sale la señal 0x0 y del multiplexor de abajo 0x2.

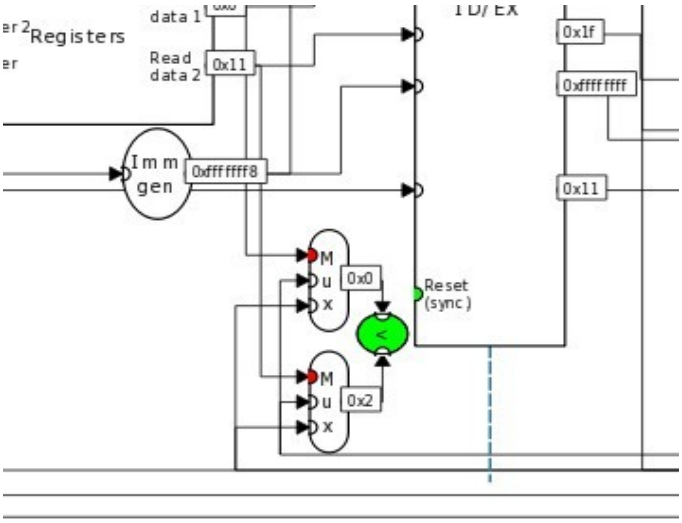

Ilustración 4: multiplexores en el ciclo 7

4) ¿Por qué los valores aparecen en este orden?

Porque, a parte de estar haciendo un blt para realizar el bgt (por ello, a nivel de ensamblador, en el multiplexor aparecen los valores cambiados de orden ya que solo permite hacer instrucciones blt), cada instrucción está pasando por un estado distinto. Entonces los resultados de los multiplexores siguientes (hacia la derecha en la máquina) aún están en proceso y necesitamos avanzar 2 ciclos más (hasta el ciclo 9) para que avancen los estados. Ahí ya veremos que aparecen 2 estados «nop (stall)», ya que debemos parar debido a la condición del bgt.

5) Le con cuidado la parte del código con la que se implementa el loop. Teniendo en cuenta el que has visto a la pregunta 3), justifica porque el pipeline al ciclo 9 presenta este estado:

Como no se cumple la condición del bgt para terminar el loop en el multiplexor, no haremos los siguientes estados, que serían Execute y Memory Access, directamente sino que aparecerán 2 «nop (stall)» con la finalidad de actualizar el valor del registrp a7 (el contador). A partir de ahí, continuaremos ejecutando el loop hasta que se cumpla la condición.

6) Cuando NO se produce el salto, ¿cuantos ciclos tarda en ejecutarse la instrucción bgt a7, zero, loop?

Tarda 7 ciclos en total. A diferencia del apartado 8, incluimos el Writeback, que nos confirma que se ha realizado el salto. Viendo el pipeline, la segunda comprobación del loop con el bgt empieza en el ciclo 12 con el Fetch (véase ilustración 5).

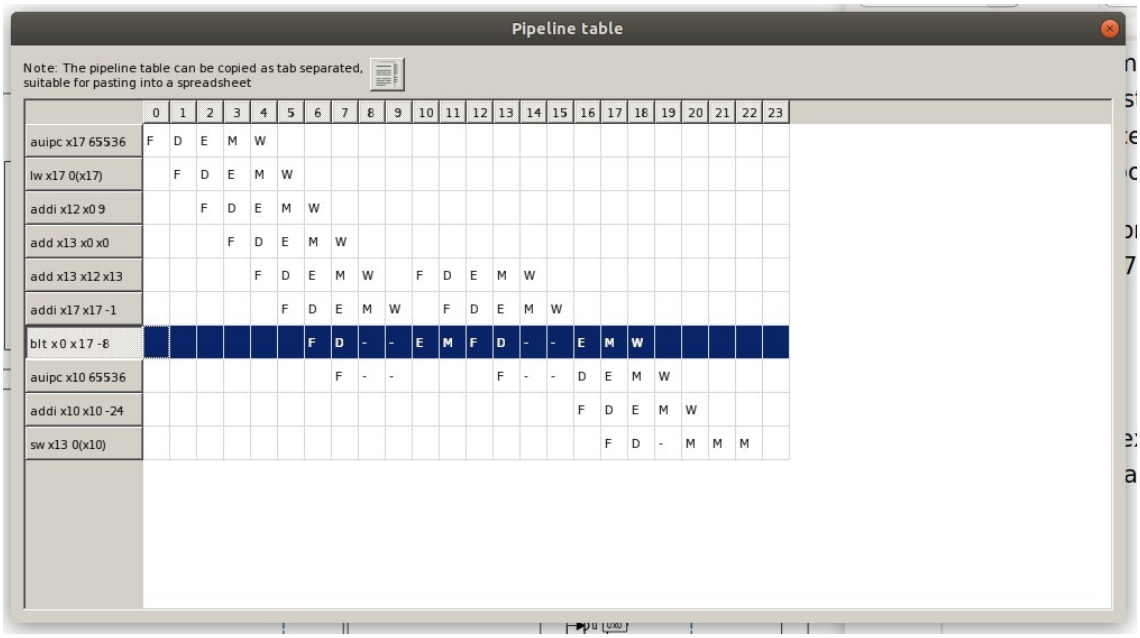

Ilustración 5: Pipeline del código 4

7) Cuando se está ejecutando la instrucción de salto, pero el salto NO se produce, ¿a qué posición apunta la memoria de instrucciones?

Apunta a la posición 0xfe850513 (estamos en el ciclo 16).
Corresponde con la intruccion addi x10 x10 -24 (que pasa por el Fetch) y el salto bgt está realizando el Execute.

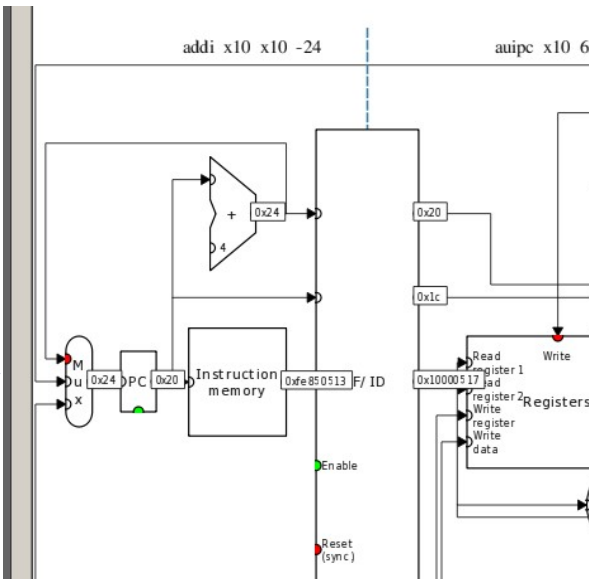

Ilustración 6: Instruction Memory en el ciclo 16

8) Cuando se produce el salto, ¿cuántos ciclos tarda en ejecutarse la instrucción bgt a7, zero, loop?

En total tarda 7 ciclos en ejecutarla. Esto se debe a que tarda 2 ciclos nop (stall) porque hay dependencia de datos, es decir, necesito saber el resultado de la instrucción addi anterior para poder realizar o no el salto. Además sí se hace el Writeback aunque no aparezca en el pipeline (véase ilustración 7), pero al escribir el pipeline se sobre escirben las letras (véase ilustración 5, fila subrayada, columna 12).

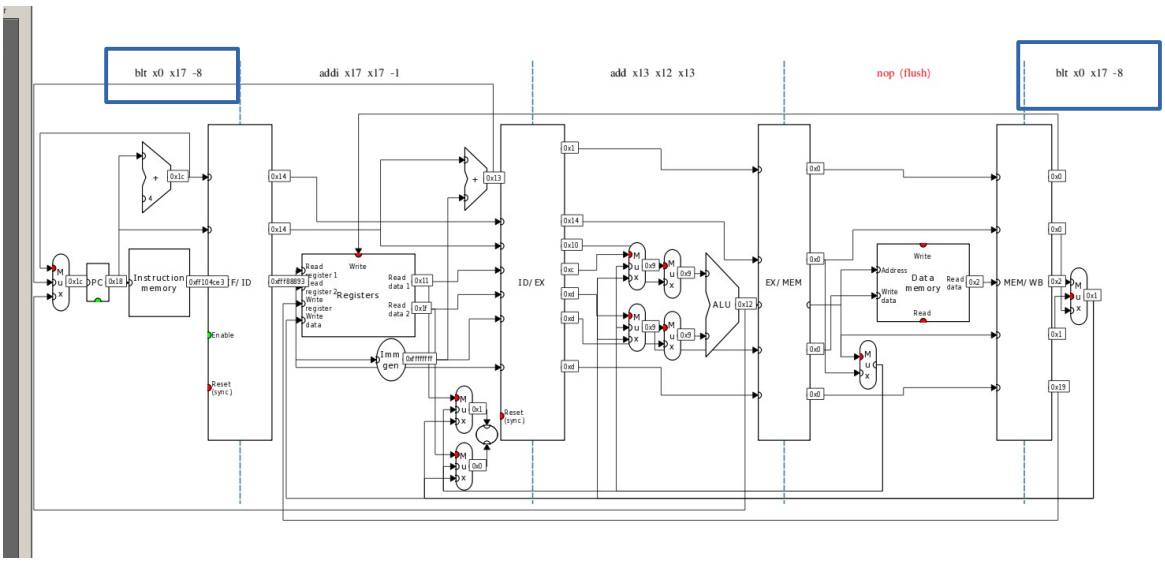

Ilustración 7: instrucción BLT duplicada, véase Fetch y Writeback

9) Cuando se está ejecutando la instrucción de salto, y el salto se produce,¿a qué posición apunta la memoria de instrucciones?

Apunta a la posición 0xd606b3 (estamos en el ciclo 10). Corresponde con la intruccion add x13 x12 x13 (que pasa por el Fetch) y el salto bgt está realizando el Execute.

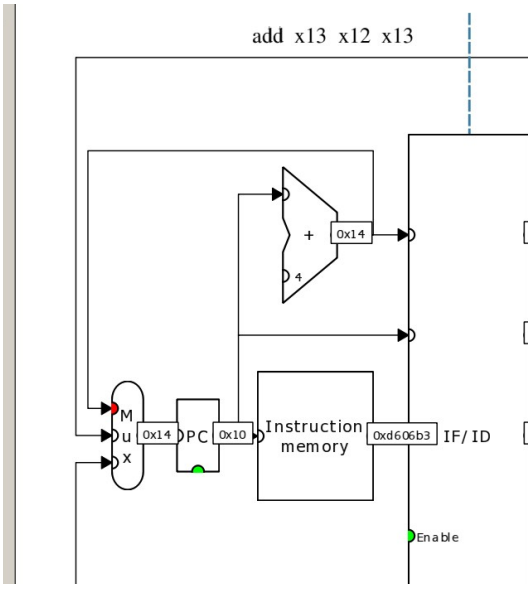

Ilustración 8: Instruction Memory ciclo 10

SiMR: Traduce el código anterior (código 4) de ensamblador de RISC-V a ensamblador de máquina rudimentaria.

Código 4

```
.data
valorDada: .word 2          # valor que multiplicaremos por 9
guardaResultat: .word 0     # y donde guardaremos el resultado
.text
main:
        lw a7, valorDada          # cargamos el valor a7 => 2
        addi a2, zero, 9          # cargamos el valor a2 => 0+9 => 9
        add a3, zero, zero        # cargamos el valor a3 => 0+0 => 0
loop:
        add a3, a2, a3            # sumamos  a3 => a2 + a3
        addi a7, a7, -1           # restamos a7 => a7 - 1
        bgt a7, zero, loop        # mientras a7 sea mayor que cero, loop
        la a0, guardaResultat     # salimos del loop y guardamos el
        sw a3, 0(a0)              # resultado en la posicion 0(x10)
```

Para hacer la conversión, utiliza el siguiente convenio de selección de registros de la Tabla 1. Nota que no es necesario ningún registro equivalente a a0 para realizar este código en SiMR. Esto se simboliza en la tabla con una ‘x’.

Simulador	Registros				
Ripes	a0	a2	a3	a7	zero
SiMR	X	R2	R3	R7	R0

Tabla 1. Convenio de selección de registros para pasar el código de Ripes a SiMR.

Ejecuta el código que hagas en SiMR microinstrucción a microinstrucción.

```
Dades:      .DW 2
.begin inici
inici:
        LOAD Dades(R0), R7    ;cargamos los datos en R7, R7 => 2
        ADDI R0, #9, R2         ; sumamos 0+9 y lo guardamos R2 => 9
        ADD R0,R0, R3           ; inicializamos R3 a 0
loop:
        ADD R2, R3, R3          ; sumamos R2+R3 y lo guardamos R3
        SUBI R7, #1, R7         ; restamos 1 al contenido de R7
        BG loop                 ; mientras R7 sea mayor que 0
        STORE R3, 10(R0)      ; guardamos el contenido de R3 en 0Ah
.end
```

Preguntas sobre el simulador SiMR:

10) Cuando se produce el salto, ¿cuántos ciclos tarda en ejecutarse la instrucción BG loop?

Tarda 5 ciclos, siendo estos FETCH,DECO, ACS, ADR2 y BRANCH (ciclos del 19 al 23). Una vez veamos DECO otra vez, ésta pertecene a la instrucción add R2, R3, R3.

11) Cuando NO se produce el salto, ¿cuántos ciclos tarda en ejecutarse la instrucción BG loop?

Tarda 3 ciclos, siendo estos FETCH, DECO y ACS (ciclos del 31 al 33). Una vez veamos el siguiente FETCH, estaremos en la instrucción store R3, 10(R0).

12) ¿Qué bits del bus de control se activan en el primer ciclo de la instrucción ADD R2, R3, R3?

Se activan los siguientes bits a 1: Ld_IR y Ld_PC. Luego, se activaran o no según se necesite (ya que estan a X), los siguientes bits: Crf y OPERAR. El resto están a 0, es decir, desactivados.

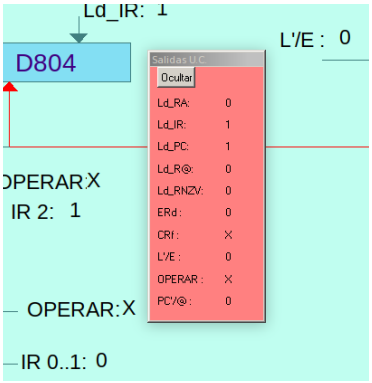

Ilustración 9: salidas de UC

13) ¿Es igual la respuesta del procesador en el primer ciclo de la instrucción ADD R2, R3, R3 que en el primer ciclo de LOAD valorDada(R0), R7?

No, la primera respuesta del procesador para LOAD Dades(R0), R7 es DECO (véase la ilustración 10, estamos en el ciclo 0) mientras que para la intrucción ADD R2, R3, R3 es FETCH (véase ilustración 11 , estamos en el ciclo 11). Además nótese que los bits del bus de control cambian entre el primer ciclo una instrucción y el de la otra.

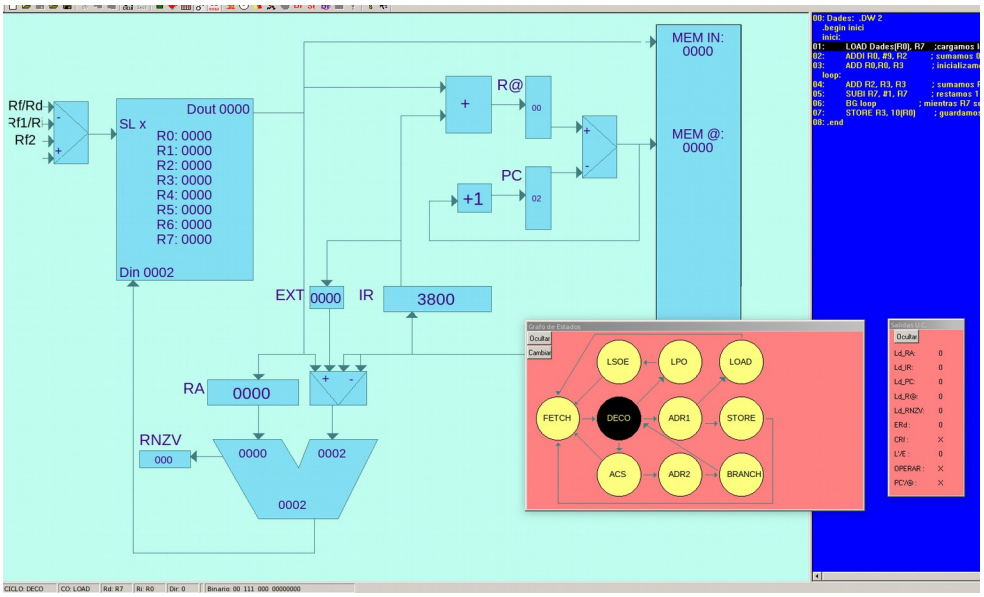

Ilustración 10: 1º ciclo de la instrucción LOAD Dades(R0), R7

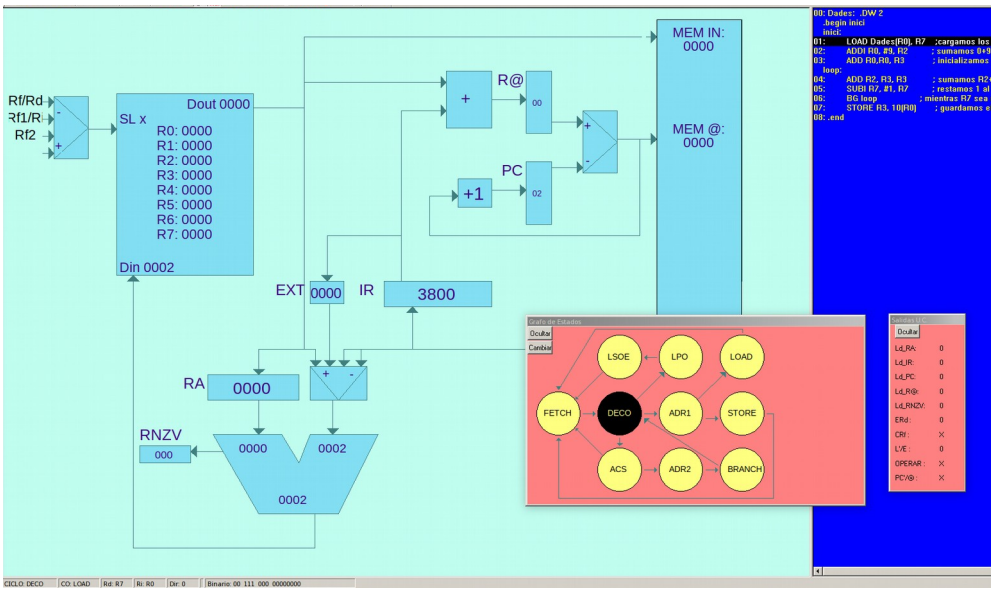

Ilustración 11: 1º ciclo de la instrucción ADD R2,R3,R3

14) Cuando se produce el salto (instrucción BG loop), ¿qué entrada del multiplexor se activa como salida para apuntar a una determinada posición de la memoria? ¿A qué posición apunta?

Se activa la entrada de R@, que apunta a la posición de memoria 0004.

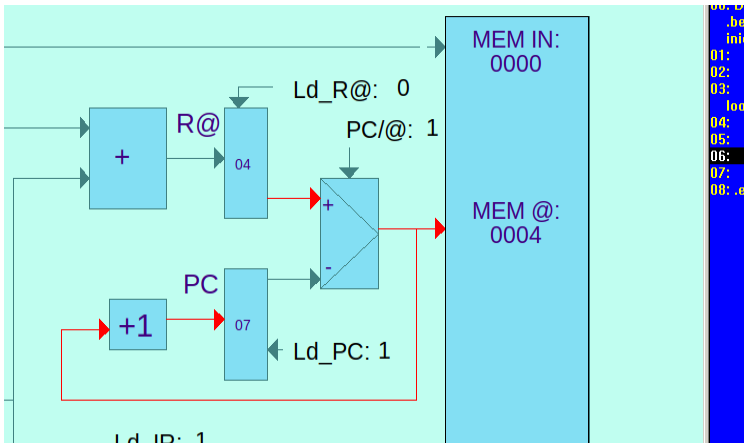

Ilustración 12: multiplexor en la primera comprobación del BG loop

15) Cuando NO se produce el salto (instrucció BG loop), ¿qué entrada del multiplexor se activa como salida para apuntar a una determinada posición de la memoria? ¿A qué posición apunta?

Se activa la entrada de PC, que apunta a la posición de memoria 0007.

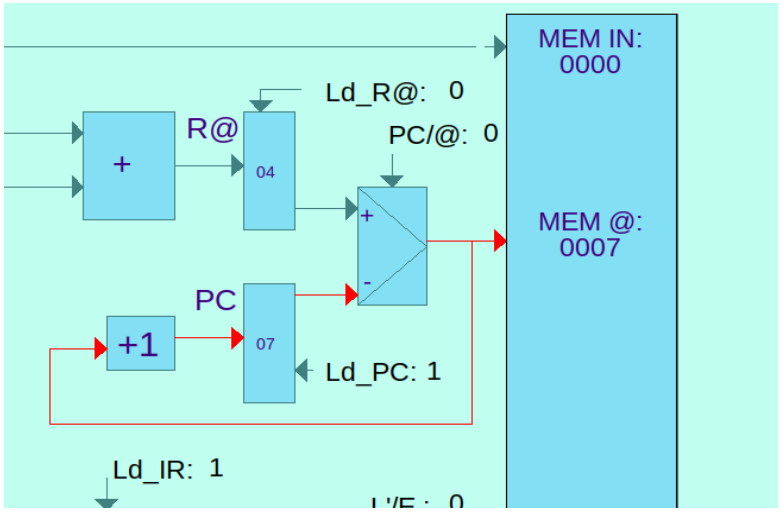

Ilustración 13: multiplexor en la segunda comprobación del BG loop

Nótese el cambio de la señal PC/@: en la ilustración 12 dicha señal está activada, es decir, la condición de BG loop se cumple (el bit de control PC/@ es 1) y por tanto cogeremos la posición de memeoria a donde apunta la instrucción de salto. En la ilustración 13, observamso que la señal de control PC/@ es cero, es decir, cogeremos la posición de memoria a donde apunta el PC, debido a que NO se ha producido el salto.

Conclusiones

En esta práctica hemos consolidado conocimientos tanto en Ripes como SiMR, como saber el funcionamiento del Pipeline mediante :

- Los ejercicios planteados, tanto guiados en los directos de youtube como los que son a realizar en casa junto con el informe.
- Entendiendo el funcionamiento de los programas: sus multiplexores, como van los ciclos y las instrucciones en cada máquina, etc. En este apartado si que he tenido más dificultades ya que con el video no me queda del todo claro ciertos aspectos, sobretodo del SiMR.
- Comparando un mismo planteamiento de código en Ripes como SiMR para ver como funciona el microprocesador en ambas máquinas.

En resumen, doy por cumplidos los objetivos propuestos más arriba, pero con dudas no resueltas.