# ARCO Pràctica 2 - Informe

Ixent Cornella

## SESIÓN 1

**2) ¿En qué dirección de memoria están los vectores X, Y, Z? ¿Cuántos bytes de la memoria ocupa cada vector X, Y, Z?**

Los vectores X, Y y Z ocupan 16 bytes, ya que cuentan con 4 words (4 bytes/word) cada uno. Las posiciones son:

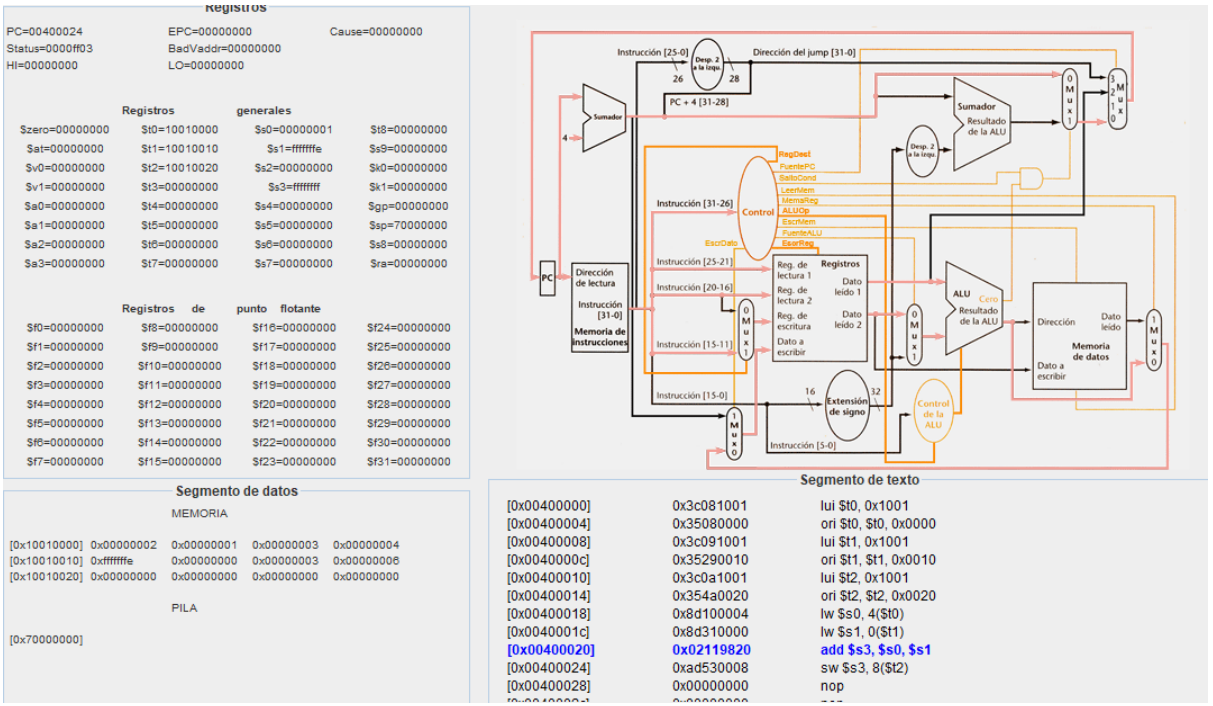
X: 0x10010000

Y: 0x10010010

Z: 0x10010020

**3) Ejecuta paso a paso (Ciclo siguiente). Haz una captura de pantalla del momento en que se hace la suma $add $s3, $s0, $s1, tanto del procesador como del valor que cambia en el banco de registros. ¿En qué posición de memoria y qué valor escribe la instrucción sw $s3, 8($t2)?**

Captura de pantalla:



La instrucción sw $s3, 8($t2) guarda en $t2 + 8 el valor de $s3, en este caso, $t2 + 8 equivale al tercer word del vector Z, y $s3 vale 0xffffffff, por lo que se guarda este valor en esa posición.

**4) Haz volver al editor, selecciona ahora el Camino de datos (tipo de procesador) Multiciclo. ¿Qué diferencias observas en la ejecución?**

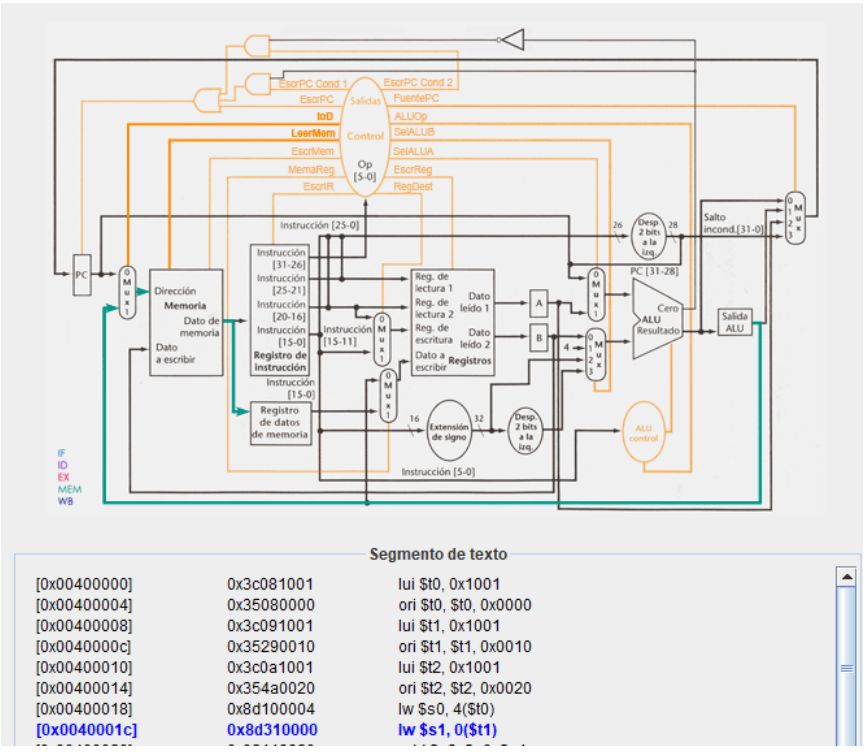
a) ¿Cuántos ciclos dura la ejecución de todo el código?

Tarda 42 ciclos, mientras que en Monociclo solo tarda 10.

b) ¿Cuántos ciclos duran las instrucciones? ¿Hay unas que tardan más que otras? Rellena la siguiente tabla con los ciclos que duran diferentes instrucciones.

| Tipo de instrucción | Número de ciclos |
|---|---|
| Aritmética | 4 |
| lw | 5 |
| sw | 4 |

c) Haz una captura de pantalla del ciclo donde la instrucción lw $s1, 0($t1) escribe en el banco de registros.



Si seguimos el recorrido cian, vemos como de la Salida de la ALU se escribe en el banco de registros.

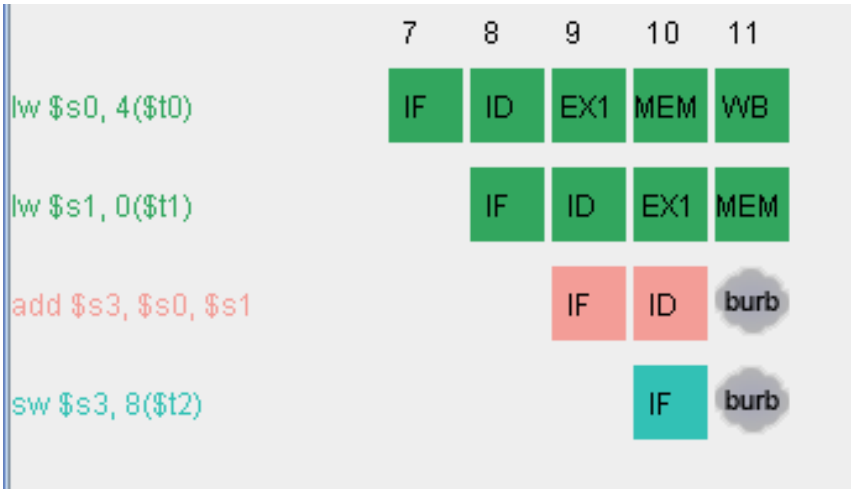
**5) Haz volver en el editor, y en configuración pon Camino de datos Segmentado Básico. Da a ensamblar y haz ejecutar.**

a) Realiza una ejecución paso a paso, ¿cuántos ciclos tarda cada instrucción ahora? ¿cuántos ciclos tarda ahora en ejecutarse el código entero?

El código entero tarda 15 ciclos en ejecutarse, mientras que cada instrucción dura lo mismo que con multiciclo, pero parece que se pueden ejecutar 5 ciclos de instrucciones diferentes a la vez, por lo que se reduce los ciclos totales de ejecución.

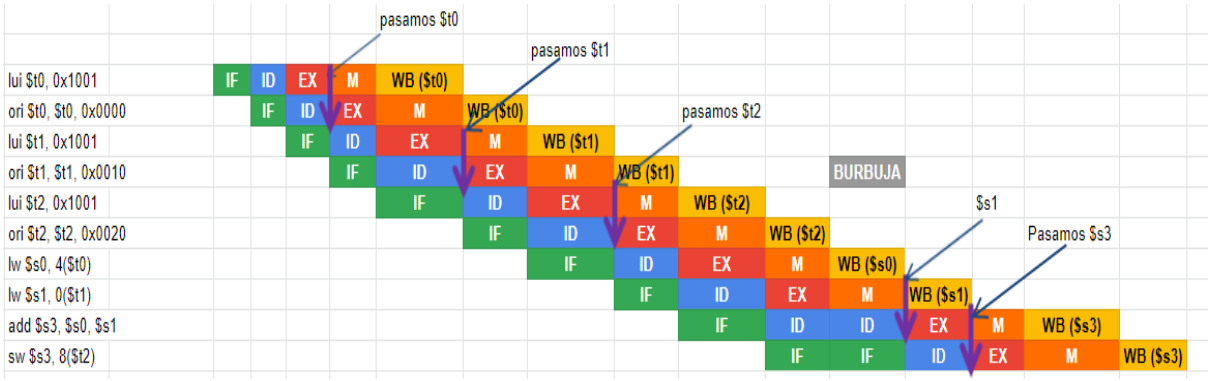
b) ¿Introduce alguna burbuja (nop hardware, mira el diagrama multiciclo) el compilador? ¿Dónde y por qué? Haz una captura.

Hay una burbuja hardware entre las instrucciones “lw $s1, 0($t1)” y “add $s3, $s0, $s1” , como se puede observar en la captura de pantalla. Esta instrucción nop se produce de forma “hardware”, ya que no la introducimos nosotros explícitamente. La instrucción nop está ahí para que el procesador pueda procesar las instrucciones correctamente.



## SESIÓN 2

**6) Haz un cronograma de la ejecución total del código anterior, marcando los lugares donde hay bloqueo hardware (burbuja) y donde hay cortocircuitos, puedes ir copiando lo que obtienes en el diagrama multiciclo.**




**7) Abre el archivo actividad2.s, pon como configuración el procesador segmentado, Camino de datos a Segmentado, y continua con salto fijo.**

a) Ensambla y dale a Ejecutar, ejecuta paso a paso. ¿Cuántas iteraciones hay en el bucle? ¿Cuántos ciclos dura toda la ejecución (hasta que el último bne llega a la última etapa)?

Tarda 55 ciclos a ejecutarse el código, y hay 4 iteraciones del bucle.

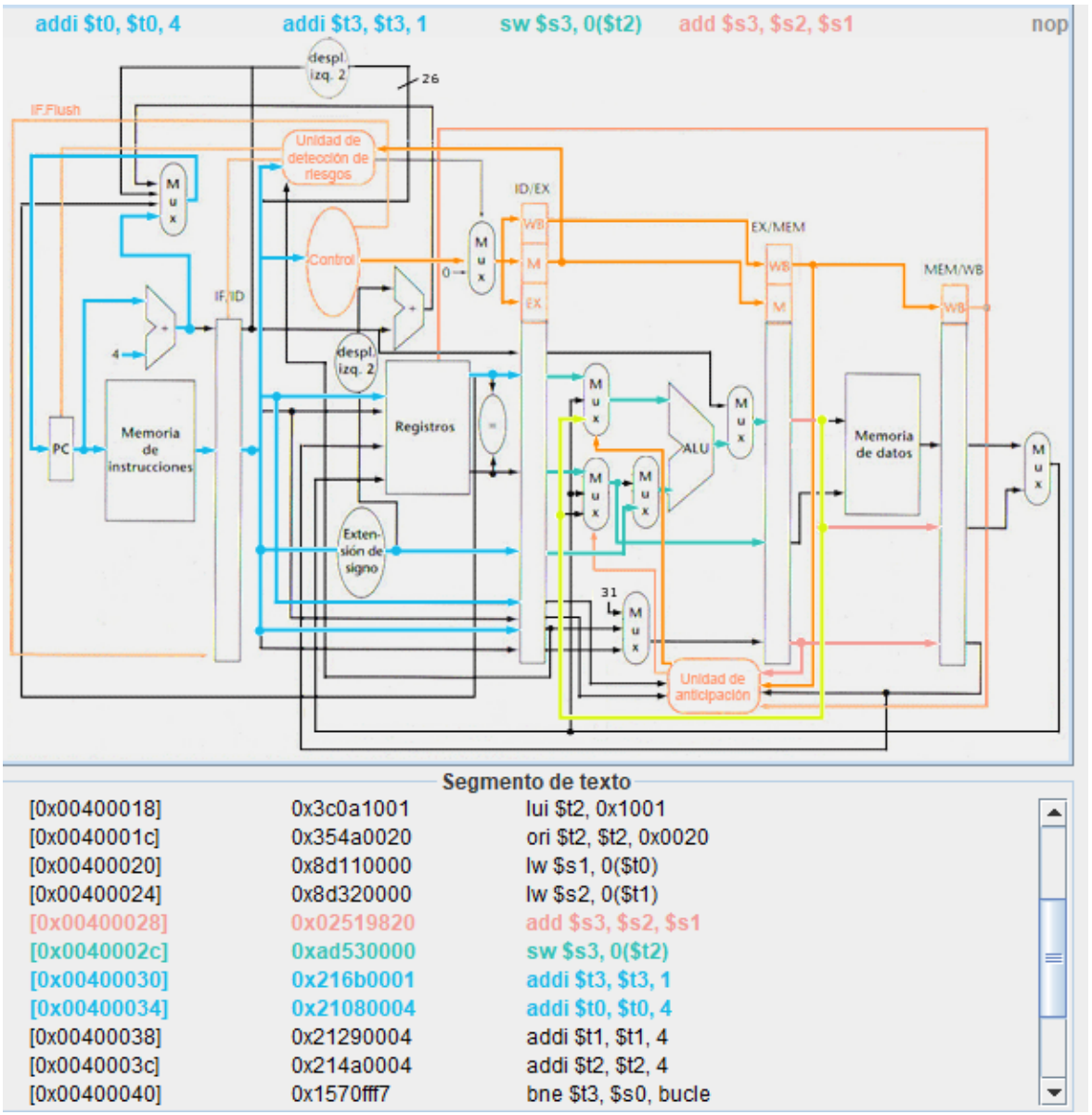
b) Con ayuda el diagrama multiciclo, realiza un cronograma de las 2 primeras iteraciones del bucle y deduce cuánto dura toda la ejecución del cronograma. En el cronograma ten en cuenta cuando se produce un bloqueo (nop hardware o burbuja) y donde se producen cortocircuitos.

Iteraciones = 8 + 11N + 3 = **11N + 11**

**El cronograma queda adjunto en la tarea, como un fichero tipo Excel.**

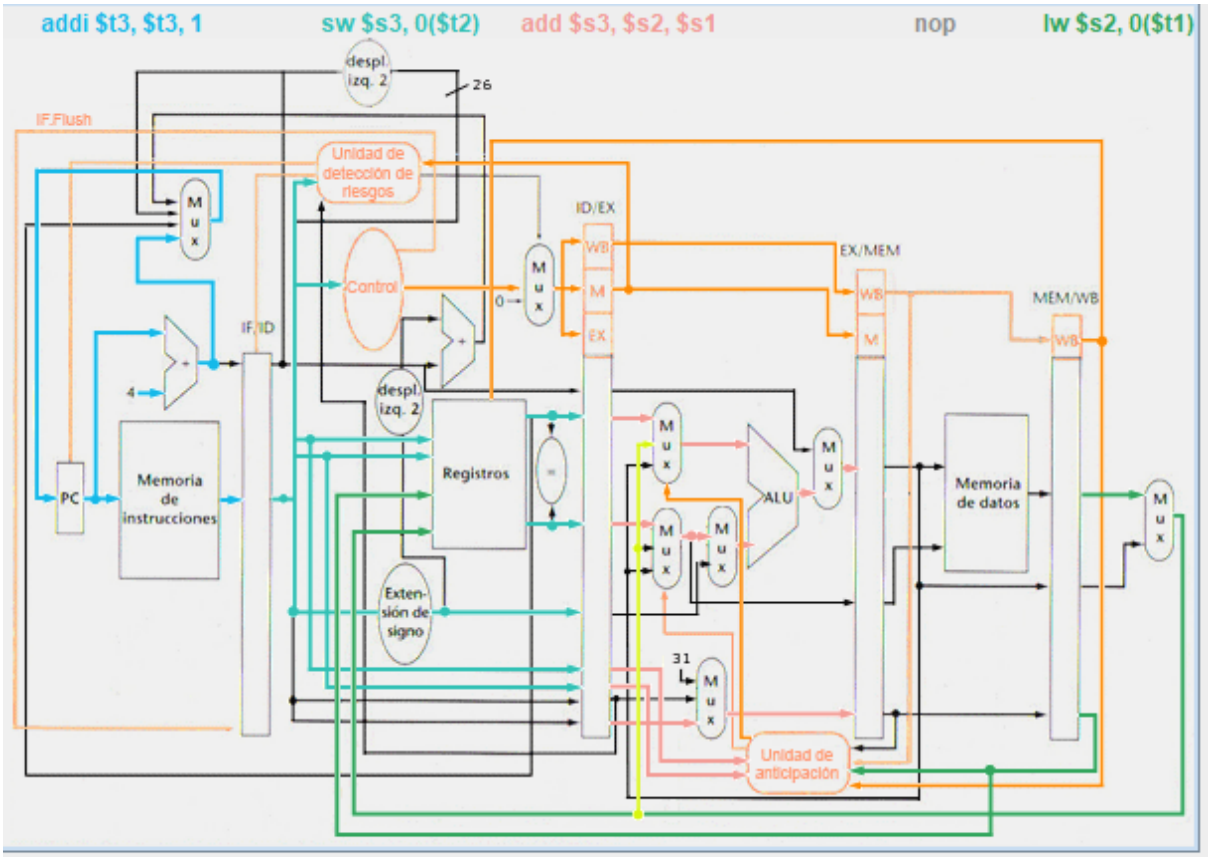
c) Haz una captura de pantalla de algún momento en que se produce un cortocircuito ALU-ALU y otro en el que el cortocircuito sea MEM-ALU.

Primero, cortocircuito ALU-ALU, donde se puede observar (en amarillo) como un dato recién calculado en ALU ($s2 + $s1) es llevado a la fase ALU para que la instrucción sw $s3 pueda tener el dato actualizado de $s3.




*Cortocircuito ALU-ALU.*

En la siguiente foto podemos ver como un dato recién sacado de la memoria es llevado a la fase ALU, por lo tanto tenemos un cortocircuito MEM-ALU:




**Segmento de texto**

| | | |
|---|---|---|
| [0x00400014] | 0x35290010 | ori $t1, $t1, 0x0010 |
| [0x00400018] | 0x3c0a1001 | lui $t2, 0x1001 |
| [0x0040001c] | 0x354a0020 | ori $t2, $t2, 0x0020 |
| [0x00400020] | 0x8d110000 | lw $s1, 0($t0) |
| **[0x00400024]** | **0x8d320000** | **lw $s2, 0($t1)** |
| **[0x00400028]** | **0x02519820** | **add $s3, $s2, $s1** |
| **[0x0040002c]** | **0xad530000** | **sw $s3, 0($t2)** |
| **[0x00400030]** | **0x216b0001** | **addi $t3, $t3, 1** |
| [0x00400034] | 0x21080004 | addi $t0, $t0, 4 |
| [0x00400038] | 0x21290004 | addi $t1, $t1, 4 |
| [0x0040003c] | 0x214a0004 | addi $t2, $t2, 4 |

*Cortocircuito MEM-ALU.*