UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Politècnica Superior d'Enginyeria
de Vilanova i la Geltrú

P2 - Informe Final: El procesador segmentado

Arquitectura de Computadors

Daniel Aagaard Pérez i Pau Martín Nadal

Eva Marin Tordera
I4511

SESIÓN 1

1) Abre el simulador Simula3MS_v4, en el menú carga el archivo actividad1.s.

a) **En configuración elige el Camino de Datos (tipo de procesador) *Monociclo*; en salto asegúrate que esté en *Salto fijo***
b) **Dale al botón de ensamblar, si no hay ningún error de compilación se activará el botón de *Ejecutar*, dale a este botón.**
c) **Se te abrirá una nueva ventana con el procesador monociclo (1 instrucción = tarda 1 ciclo), y con el código en lenguaje máquina. Copia aquí el código en lenguaje máquina y comentas qué cambios ves respecto al código *.s:**

Texto en el editor (.s)	**Lenguaje a máquina**
la $t0, X	lui $t0, 0x1001
la $t1, Y	ori $t0, $t0, 0x0000
la $t2, Z	lui $t1, 0x1001
lw $s0, 4($t0)	ori $t1, $t1, 0x0010
lw $s1, 0($t1)	lui $t2, 0x1001
add $s3, $s0, $s1	ori $t2, $t2, 0x0020
sw $s3, 8($t2)	lw $s0, 4($t0)
	lw $s1, 0($t1)
	add $s3, $s0, $s1
	sw $s3, 8($t2)

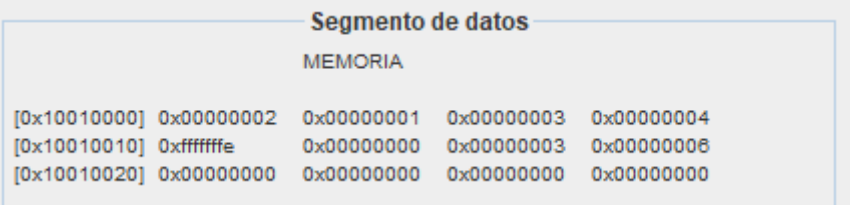
El cambio más significativo es la instrucción “la” - “load address” que se convierte en lui i ori.
La instrucción “lui” se encarga de cargar los 16 bits más significativos del valor hexadecimal que se indica, en el registro que queramos. Los bits menos significativos (los otros 16 bits) se rellenan con ceros.
Este ejemplo lui $t0, 0x1001 cargaría los 16 bits más significativos del valor hexadecimal 0x1001 en el registro $t0 y pondría ceros en los 16 bits restantes. Por tanto $t0 = 0x10010000.
Después a este mismo $t0 se le aplica la instrucción ori, que aplica una OR de la constante hexadecimal que queramos.
En este ejemplo ori $t0, $t0, 0x0000, guarda en $t0 el resultado de hacer una OR entre $t0 (0x1001000) i la constante hexadecimal 0x0000 por lo que el resultado en $t0 serà el mismo valor que antes, ya que la OR de 0x0000 no altera el resultado.

Esto sería útil para poner un valor en los 16 bits menos significativos de un registro, un ejemplo podría ser el siguiente:
lui $t2, 0x1001
ori $t2, $t2, 0x0020
Siguiendo la lógica anterior, en la primera instrucción $t2 = 0x10010000.
Después cuando hacemos la ori, $t2 = 0x10010020
De este modo hemos modificado los 16 bits menos significativos de $t2.

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

La dirección de memoria inicial de cada vector es la siguiente:

X = 0x10010000, Y = 0x10010010, Z = 0x10010020



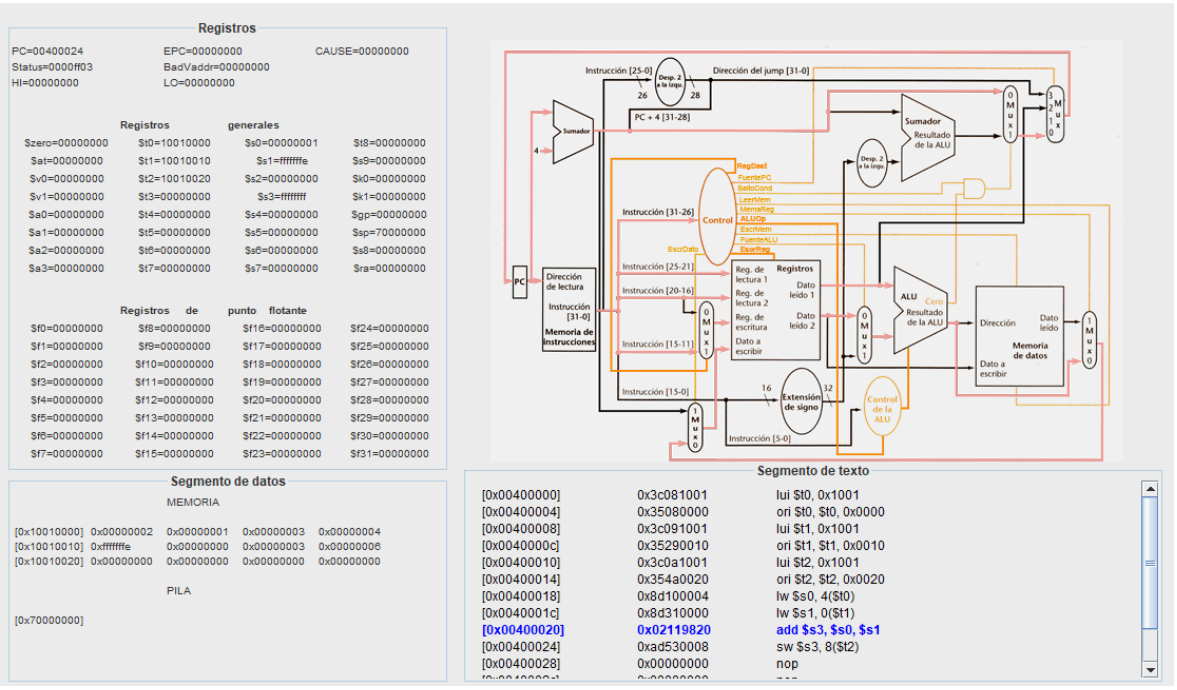
Segmento de datos

MEMORIA

[0x10010000]	0x00000002	0x00000001	0x00000003	0x00000004
[0x10010010]	0xfffffffe	0x00000000	0x00000003	0x00000006
[0x10010020]	0x00000000	0x00000000	0x00000000	0x00000000

Cada vector ocupa 16 bytes.

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)?



Registros

PC=00400024 EPC=00000000 CAUSE=00000000
Status=0000ff03 BadVaddr=00000000
HI=00000000 LO=00000000

Registros generales

$zero=00000000	$t0=10010000	$s0=00000001	$t8=00000000
$at=00000000	$t1=10010010	$s1=fffffffe	$s9=00000000
$v0=00000000	$t2=10010020	$s2=00000000	$k0=00000000
$v1=00000000	$t3=00000000	$s3=ffffffff	$k1=00000000
$a0=00000000	$t4=00000000	$s4=00000000	$gp=00000000
$a1=00000000	$t5=00000000	$s5=00000000	$sp=70000000
$a2=00000000	$t6=00000000	$s6=00000000	$s8=00000000
$a3=00000000	$t7=00000000	$s7=00000000	$ra=00000000

Registros de punto flotante

$f0=00000000	$f8=00000000	$f16=00000000	$f24=00000000
$f1=00000000	$f9=00000000	$f17=00000000	$f25=00000000
$f2=00000000	$f10=00000000	$f18=00000000	$f26=00000000
$f3=00000000	$f11=00000000	$f19=00000000	$f27=00000000
$f4=00000000	$f12=00000000	$f20=00000000	$f28=00000000
$f5=00000000	$f13=00000000	$f21=00000000	$f29=00000000
$f6=00000000	$f14=00000000	$f22=00000000	$f30=00000000
$f7=00000000	$f15=00000000	$f23=00000000	$f31=00000000

Segmento de datos

MEMORIA

[0x10010000]	0x00000002	0x00000001	0x00000003	0x00000004
[0x10010010]	0xfffffffe	0x00000000	0x00000003	0x00000006
[0x10010020]	0x00000000	0x00000000	0x00000000	0x00000000

PILA

[0x70000000]

Segmento de texto

[0x00400000]	0x3c081001	lui $t0, 0x1001
[0x00400004]	0x35080000	ori $t0, $t0, 0x0000
[0x00400008]	0x3c091001	lui $t1, 0x1001
[0x0040000c]	0x35290010	ori $t1, $t1, 0x0010
[0x00400010]	0x3c0a1001	lui $t2, 0x1001
[0x00400014]	0x354a0020	ori $t2, $t2, 0x0020
[0x00400018]	0x8d100004	lw $s0, 4($t0)
[0x0040001c]	0x8d310000	lw $s1, 0($t1)
[0x00400020]	0x02119820	add $s3, $s0, $s1
[0x00400024]	0xad530008	sw $s3, 8($t2)
[0x00400028]	0x00000000	nop

La instrucción add $s3, $s0, $s1 pone en $s3 el valor 0xFFFFFFFF.

La instrucción sw $s3, 8($t2), escribe el valor 0xFFFFFFFF en la posición de memoria 0x10010028, que pertenece a la tercera posición del vector Z.

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

Antes una instrucción tardaba un único ciclo, en cambio, con el tipo de procesador multiciclo tarda como mínimo 4 ciclos, excepto los lw que tardan 5 ciclos.

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

Todo el código dura 42 ciclos. 5*2 + 7*4 + 4 = 42

2 instrucciones lw de 5 ciclos **5*2**

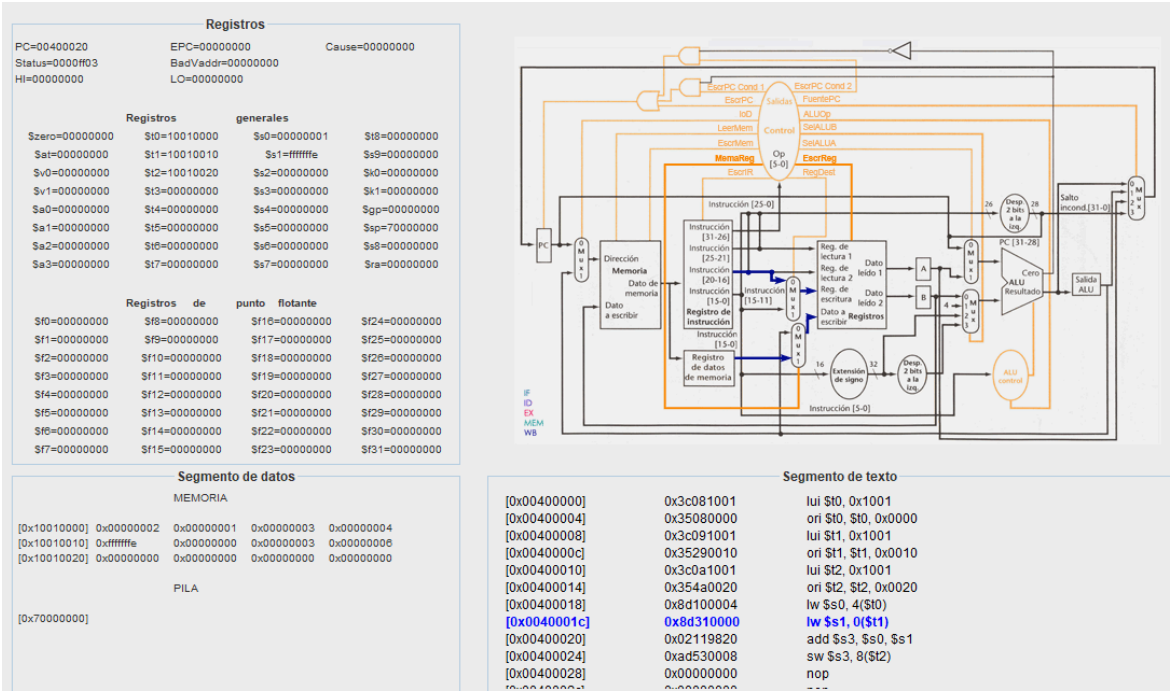
7 instrucciones aritméticas de 4 ciclos **7*4**

1 instrucción s2 de 4 ciclos **4**

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	Nº 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.



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? Para poder ver la ejecución entera, hasta que sw $s3, 8($t2) llega a la última etapa, añade 4 nops al final del código.

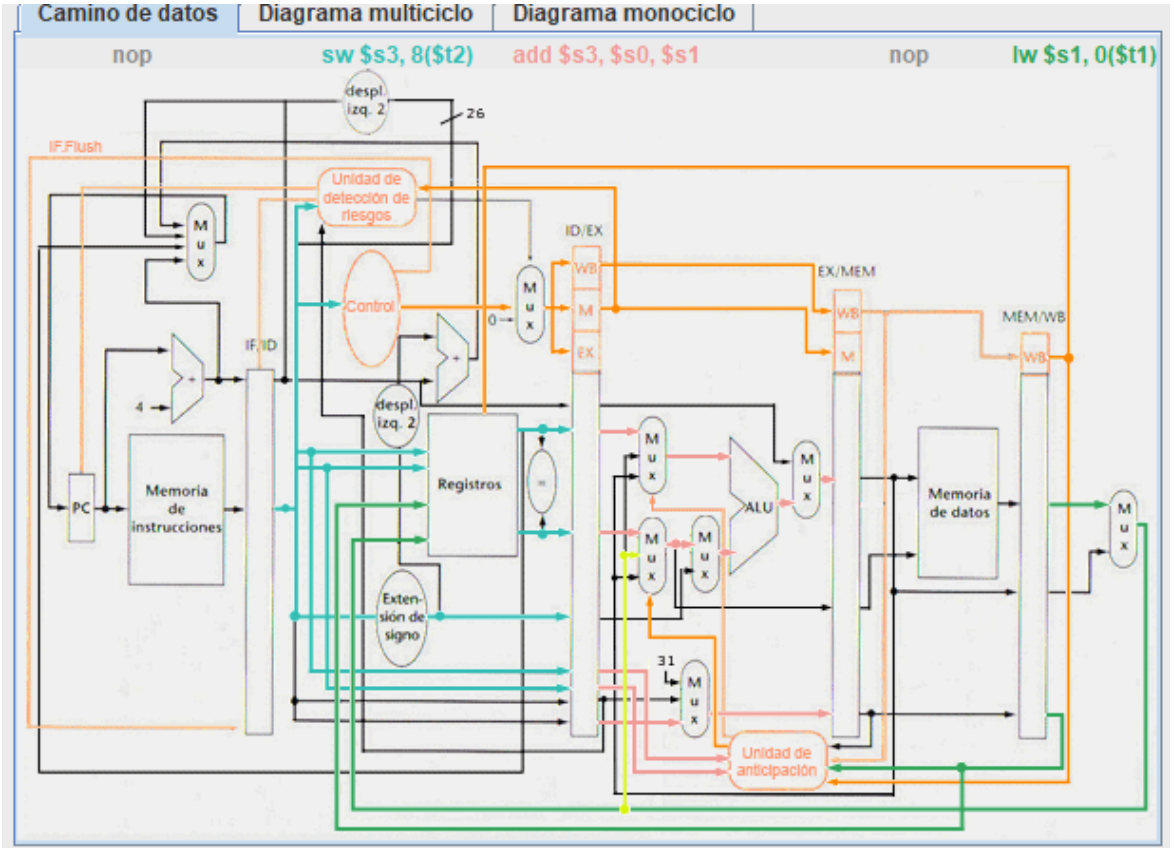
Cada instrucción tarda 5 ciclos.

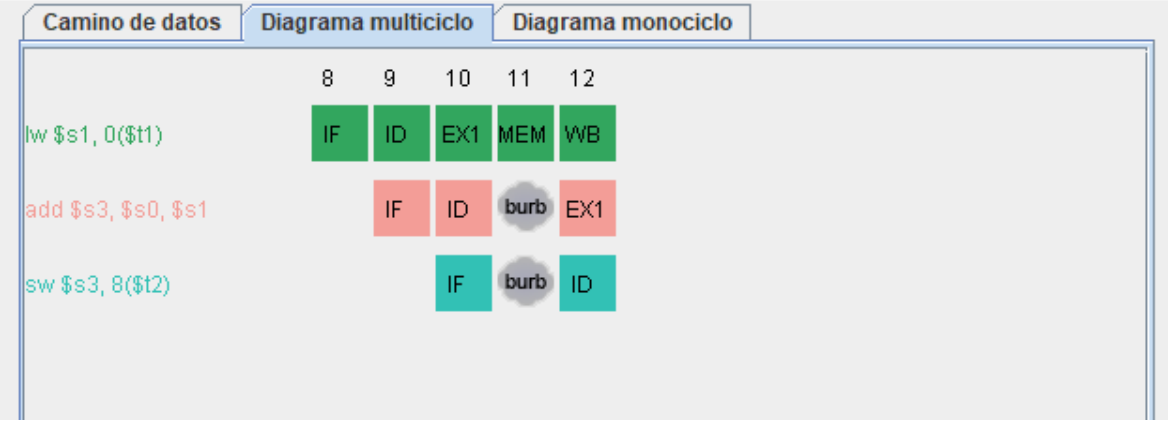
El código entero tarda en ejecutarse 15 ciclos.

Segmento de texto

[0x0040000c]	0x35290010	ori $t1, $t1, 0x0010
[0x00400010]	0x3c0a1001	lui $t2, 0x1001
[0x00400014]	0x354a0020	ori $t2, $t2, 0x0020
[0x00400018]	0x8d100004	lw $s0, 4($t0)
[0x0040001c]	0x8d310000	lw $s1, 0($t1)
[0x00400020]	0x02119820	add $s3, $s0, $s1
[0x00400024]	0xad530008	sw $s3, 8($t2)
[0x00400028]	0x00000000	nop
[0x0040002c]	0x00000000	nop
[0x00400030]	0x00000000	nop
[0x00400034]	0x00000000	nop

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



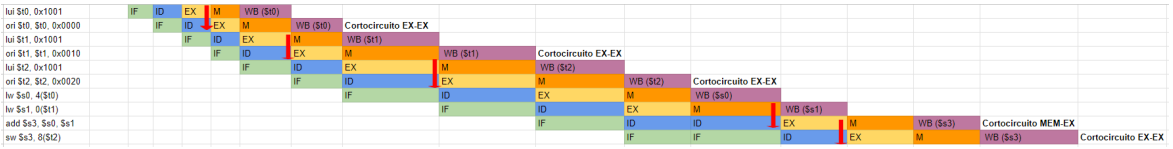


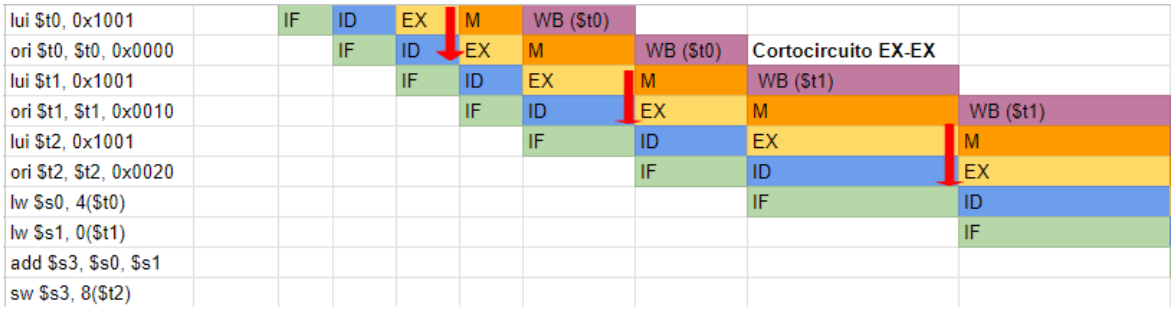
Introduce dos burbujas, una después del ID del ass $s3, $s0, $s1 y otra después del if de sw $s3, 8($t2). Las pone ya que se tiene que bloquear ya que se va a producir un cortocircuito MEM-EX. Esto pasa debido a que el load termina cuando acaba la etapa del MEM(ya que accede a memoria), como la siguiente instrucción (add $s3, $s0, $s1) depende del load de la anterior, aparece el bloqueo dicho anteriormente.

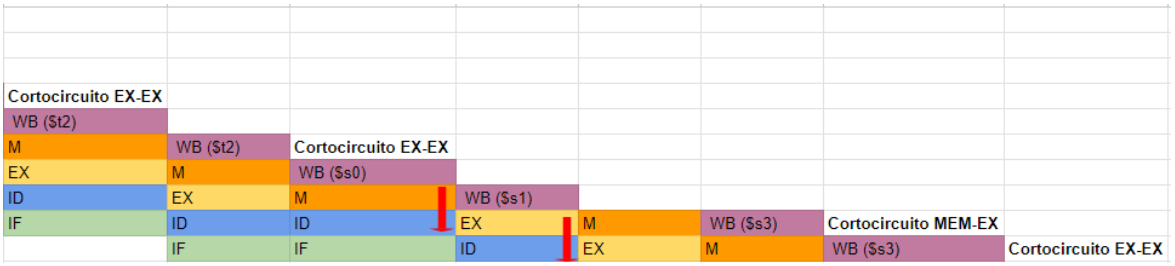
Teoria: Instrucción que produce el dato en memoria (lw) y lo 'consume' en otra instrucción en EX.

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.







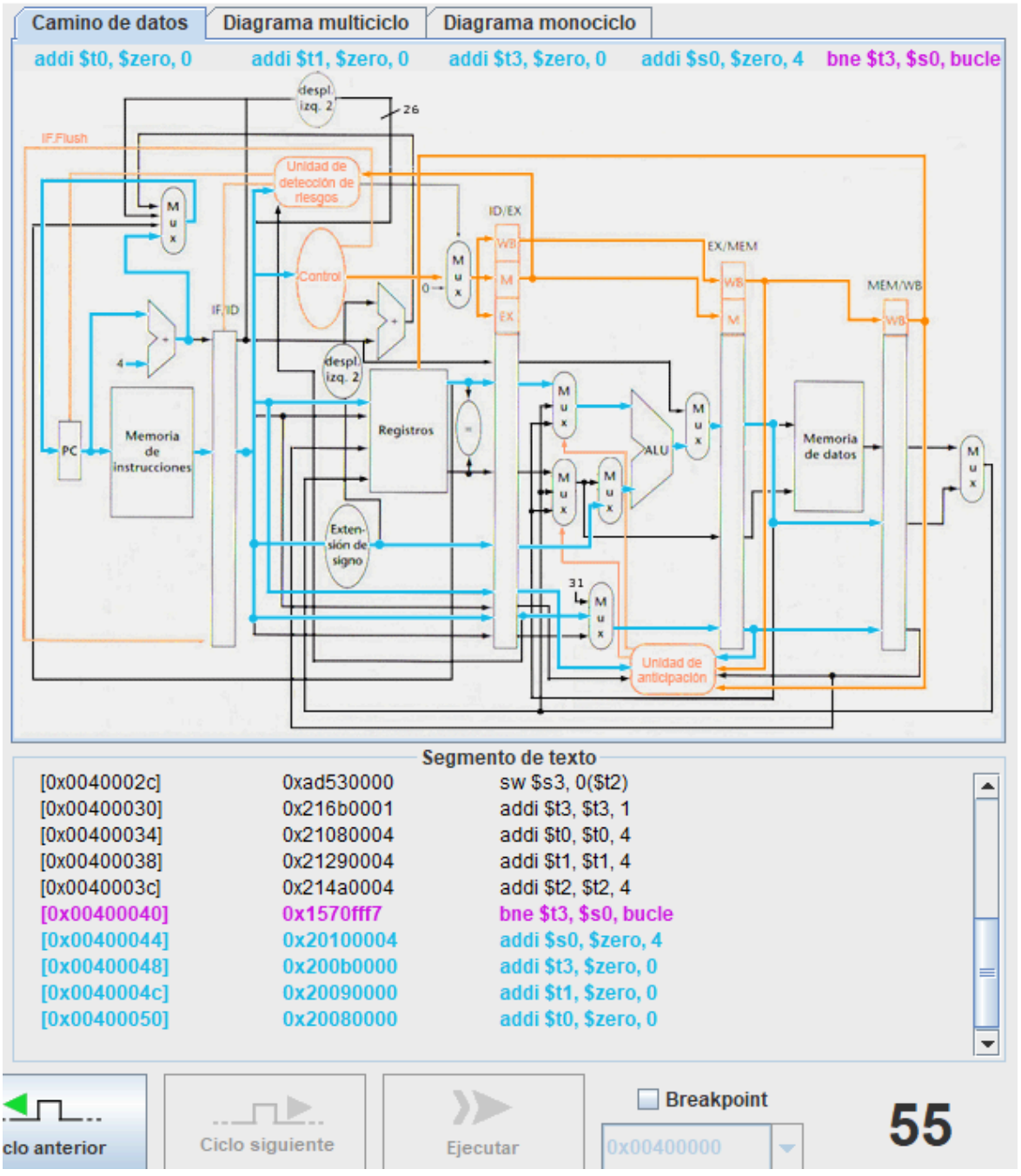
Aquí está el cronograma para verlo mejor. **Cronograma ex 6**

7) Abre el archivo actividad2.s, pon como configuración el procesador segmentado, Camino de datos → Segmentado, y continua con salto fijo. Después del salto bne hay 4 instrucciones que no tienen importancia, que están puestas como relleno para que se ejecute todo el bne.

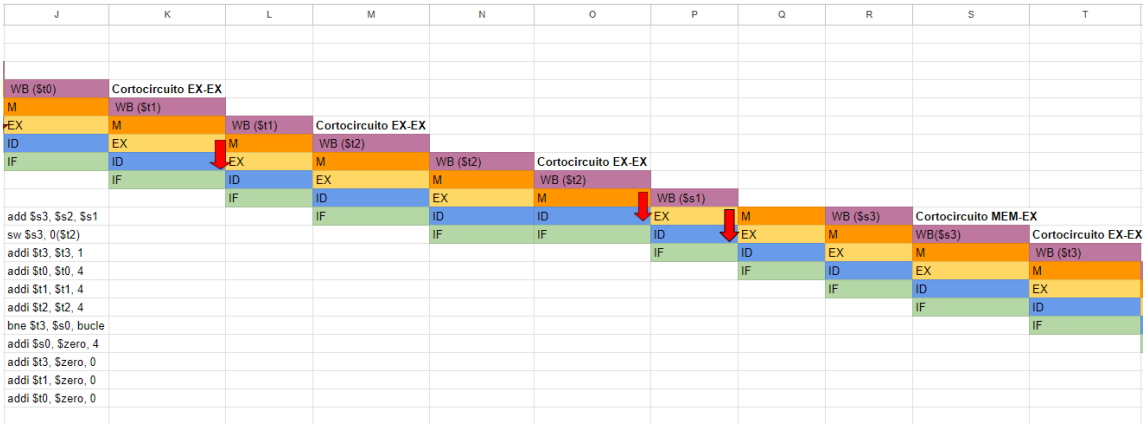
a) Ensambla y dale a Ejecutar, ejecuta paso a paso. ¿Cuántas iteraciones hay en el bucle? ¿Cuántos ciclos dura toda la ejecución (hasta que el último bne llega a la última etapa)?

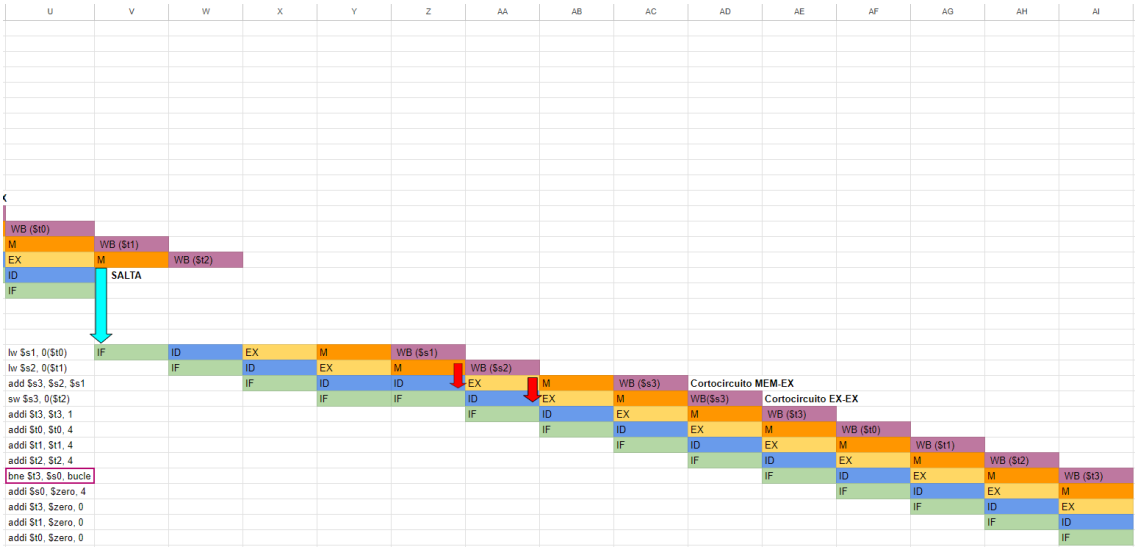
Hay 4 iteraciones en el bucle porque $s0 es 4 i $t3 empieza en 0 i suma 1 cada iteración del bucle. 4/1 = 4 iteraciones.

Toda la ejecución del código dura 55 ciclos.



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.

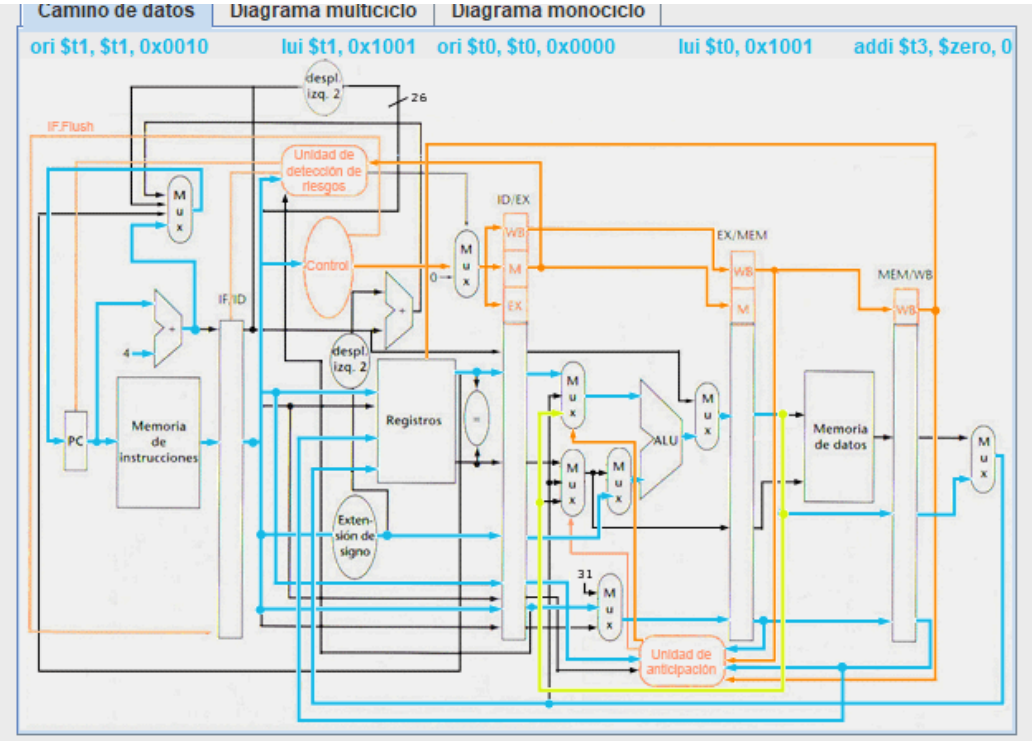




Aquí está el cronograma para verlo mejor. **Cronograma ex 7**

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.

ALU - ALU



MEM - ALU