

PRÁCTICA 2: EL PROCESADOR SEGMENTADO

SESIÓN 1

- 1) Abre el simulador Simula3MS_v4, en el menu carga el archive 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 (1instrucción=tarda 1 ciclo), y con el código en lenguaje máquina. Copia aquí el código en lenguaje máquina y comentas que cambios ves respecto al código *.s:

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

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)?

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?

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	
lw	
sw	

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.

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


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 → 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)?

b) Con ayuda del 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.

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 ALU-MEM. 

SESIÓN 3

8. Abre el archivo actividad3.s, asegúrate que la configuración de procesador es **Camino de datos** → **Segmentado** y el salto es **Salto fijo**, ensambla y dale a ejecutar. Después del salto bne hay 4 instrucciones que no tienen importancia, que están puestas como relleno para que se ejecute todo el bne

a) ¿Cuántas iteraciones hace el bucle? ¿Qué hace el código? ¿En qué posiciones de memoria escribe y qué valores?

b) Realiza un cronograma de las 2 primeras iteraciones del bucle, mostrando cuando se producen bloqueos (nops hardware o burbujas) y cuando se producen cortocircuitos de datos. A partir de estas dos primeras iteraciones, deduce ¿Cuántos ciclos tarda la ejecución total? ¿Coincide con lo que te da el simulador?

9. Vuelve al editor, ahora cambiar la configuración del procesador a **Camino de datos** → **Segmentado** pero el salto es **Salto retardado**.

Explicación del salto retardado:

Supongamos este código:

PC

36 sub \$t0, \$t4, \$8

40 \$beq \$t1, \$t3, 7 # si \$t1==\$t3 saltará a PC= PC+4+7*4=44+28=72

(suponemos que \$t1==\$t3)

44 and \$t2, \$t2, \$5

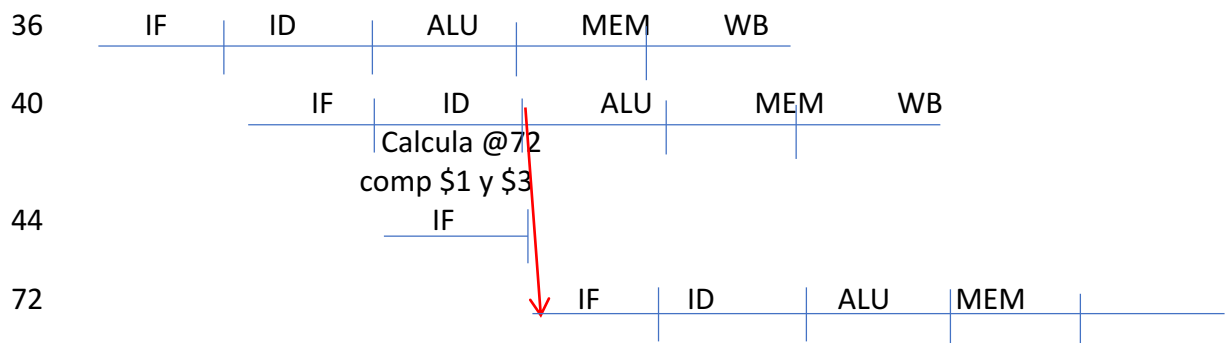
48 or \$t3, \$t2, \$5

52 add \$t4, \$t4, \$2

.....

72 lw \$t4, 50(\$t7)

CRONOGRAMA CON SALTO FIJO



En este caso se habría empezado a ejecutar la instrucción 44, y sólo esa se tendría que anular.

CRONOGRAMA CON SALTO RETARDADO

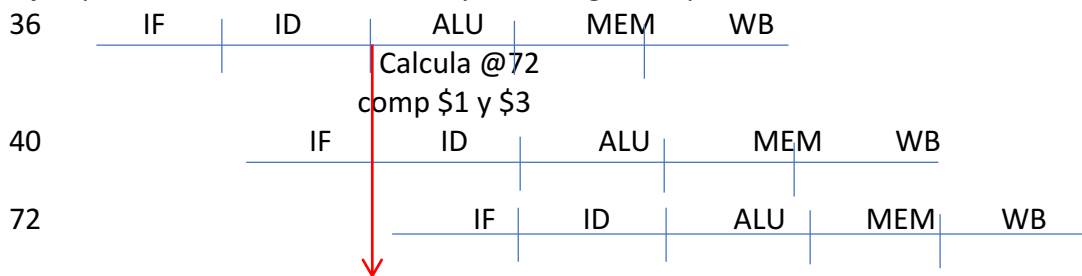
Otra opción es realizar el **salto retardado**, que quiere decir reordenar el código de manera que la instrucción de después del salto **Sí se tenga que ejecutar** (digamos que pone una instrucción de dentro del bucle aparentemente fuera, aunque en realidad sí que la ejecuta). En el ejemplo anterior quedaría:

```

PC
36 $beq $1, $3, 7    # si $1==$3 saltará a PC= PC+4+7*4=44+28=72
(suponemos que $1=$3)
40 sub $10, $4, $8
44 and $12, $2, $5
48 or $13, $2, $5
52 add $14, $4, $2
.....
72 lw $4, 50($7)

```

En este ejemplo 36 es un **salto retardado** y el cronograma quedaría:



36 es el salto BEQ y 40 es el sub que Sí se tiene que realizar, por eso no la anula cuando tiene la @ de salto y la comprobación de los registros, simplemente tiene que elegir si después de la 40 va la 44 o la 72, pero no anular la 40

a) ¿Qué ocurre cuando intentas ejecutar? ¿Por qué crees que ocurre? Para darte cuenta de qué pasa ejecuta paso a paso hasta llegar al salto bne, ¿qué ha cambiado respecto a la configuración anterior?

b) Para solucionar el problema pon una instrucción de dentro del bucle detrás de bne ¿Cuál es la única instrucción posible de poner para que el código haga lo mismo? Si es necesario modifica ligeramente la instrucción movida.

c) Vuelve a ejecutar ahora asegurándote que el resultado es correcto, ¿cuántos ciclos tarda? ¿Tarda menos? Si es así, ¿por qué tarda menos? Haz un cronograma de las dos primeras iteraciones