

PRÁCTICA 2

ESTUDIO DEL RENDIMIENTO DEL PROCESADOR SEGMENTADO MIPS32

ARQUITECTURA DE COMPUTADORES. 2º CURSO

1. OBJETIVOS

El propósito de esta práctica es afianzar los conocimientos sobre el funcionamiento del encadenamiento básico, los bloqueos de datos, de control y estructurales (en especial los introducidos por instrucciones de larga duración) mediante el simulador del procesador MIPS segmentado VisualMips32.

2. PREPARACIÓN

Antes de acudir a la sesión de laboratorio el alumno debe:


- Leer y asimilar los contenidos teóricos del apartado 3.
- Realizar el cuestionario previo que se encuentra en un documento adicional en la enseñanza virtual en base a su turno (A o B) de forma manuscrita. Este cuestionario deberá ser entregado al profesor al comienzo de la sesión.

No olvide el enunciado de la práctica anterior, podría serle útil!

3. INTRODUCCIÓN TEÓRICA.


En clase teórica se ha estudiado el comportamiento de los procesadores MIPS y una serie de optimizaciones para evitar o reducir los bloqueos. El simulador VisualMips32 implementa muchas de estas características siendo algunas de ellas configurables.

Para que el alumno pueda interpretar correctamente los cronogramas del simulador, es conveniente que verifique la configuración, incluso tras arrancar el simulador ya que cargará por defecto la última configuración utilizada en anteriores ocasiones.

Para configurar el computador a simular, pulse el botón ‘Propiedades’  o directamente pulsando F12. En caso de duda, restaure la configuración por defecto accediendo a la sección ‘configuración’, en la misma ventana de propiedades.

La configuración por defecto en el simulador corresponde a la que hemos considerado también en clase como MIPS por defecto, esto es, un procesador Segmentado con adelantamientos (*Forwarding*), memoria separada para instrucciones y datos, sin saltos retardados y con instrucciones de larga duración en la multiplicación (EX de 5 ciclos) y la división (EX de 8 ciclos).

También puede configurarse el sistema total o parcialmente a través de directivas de configuración dentro de la sección de configuración (.config) del programa fuente. Esta solución puede resultar especialmente útil cuando se pretender realizar varias simulaciones con diferentes configuraciones. Durante la simulación, se utilizará la configuración base establecida en el simulador en la ventana ‘Propiedades’ pero con las modificaciones especificadas mediante directivas.

Puede consultar todas las directivas e instrucciones implementadas en el simulador en el botón  ‘Guía Rápida’ de la ventana ‘Editor’ o pulsando F1.

3.1. INSTRUCCIONES DE LARGA DURACIÓN

Hasta ahora se ha considerado que toda instrucción se ejecuta en una misma ALU, pero en la práctica el computador dispone de varias unidades de cálculo, unas integradas en el procesador y otras alojadas en un coprocesador independiente, cada una de ellas dedicadas a realizar ciertos tipos de operaciones. VisualMips32 simula una implementación MIPS con las siguientes unidades:

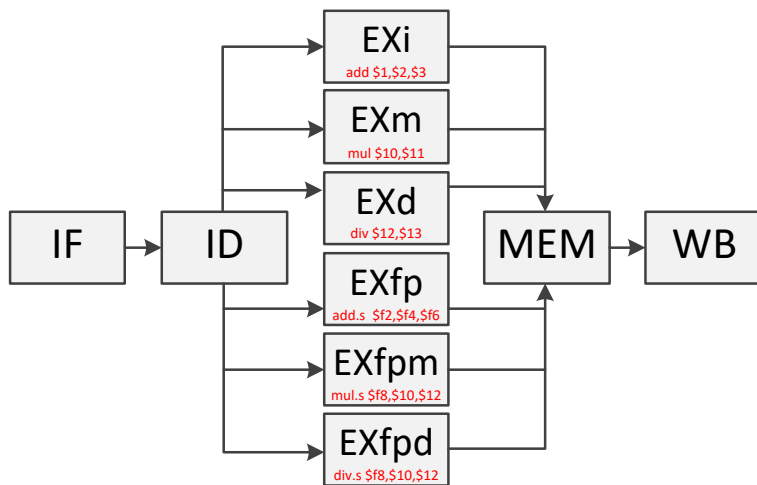


Figura 1. Pipeline con las unidades funcionales independientes

- **Unidad de enteros principal (EX_i):** encargada de las cargas, almacenamientos, saltos y las operaciones ALU enteras excepto las de multiplicación y división.
- **Unidad para multiplicación de enteros (EX_m)**
- **Unidad para la división de enteros (EX_d)**
- **Unidad de FP principal (EX_{fp}):** realiza todas las operaciones de coma flotante a excepción de la multiplicación y la división.
- **Unidad para la multiplicación en coma flotante (EX_{fpm}).**
- **Unidad para la división en coma flotante (EX_{fpd})**

Salvo EX_i , estas unidades funcionales realizan operaciones complejas por lo que suelen precisar de varios ciclos de reloj para realizar el cálculo. Esto supone que las instrucciones que utilicen alguna de estas unidades más complejas requerirán varios ciclos en su fase EX. Las operaciones de coma flotante, de multiplicación y de división son las que requieren un mayor número de ciclos.

div \$1, \$2	IF	ID	EXd	EXd	EXd	EXd	EXd	EXd	EXd	EXd	EXd	MEM	WB
--------------	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	----

Figura 2. Ejecución de una instrucción de división entera de 8 ciclos

Puesto que la fase EX dispone de varias unidades funcionales independientes (figura 1) y que algunas de estas unidades son multiciclo, puede darse el caso de que varias instrucciones puedan estar ejecutando su fase EX simultáneamente. Véase como ejemplo el cronograma de la figura 3 en el que hay 3 instrucciones en EX durante el ciclo 6:

	1	2	3	4	5	6	7	8	9	10	11	12	13
1: [4194304] mult \$2, \$3	IF	ID	EXm0	EXm1	EXm2	EXm3	EXm4	MEM	WB				
2: [4194308] div \$1, \$2		IF	ID	EXd	EXd	EXd	EXd	EXd	EXd	EXd	EXd	MEM	WB
3: [4194312] add.s \$f1, \$f2, \$f3			IF	ID	EXfp0	EXfp1	EXfp2	EXfp3	MEM	WB			
4: [4194316] add \$4, \$5, \$6				IF	ID	EX	MEM	WB					

Figura 3. Ejecución de 4 instrucciones en la fase EX (cada instrucción se ejecuta en su unidad funcional según la operación que realiza)

Pero, además, cuando la implementación lo permite, estas unidades de cálculo multiciclo suelen estar a su vez **segmentadas**, esto es, que el cálculo de la operación puede estar dividido en varias subetapas (de un ciclo cada una). La segmentación de tales unidades va a permitir que en la misma unidad funcional pueda realizarse varias operaciones en paralelo siempre que estén en subetapas diferentes. Este hecho permitirá iniciar un nuevo cálculo sin haber terminado la anterior pudiéndose incluso llegar a finalizar una operación compleja en cada ciclo de reloj.

	1	2	3	4	5	6	7	8	9	10
mult \$1, \$2	IF	ID	EXm0	EXm1	EXm2	EXm3	EXm4	MEM	WB	
mult \$3, \$4		IF	ID	EXm0	EXm1	EXm2	EXm3	EXm4	MEM	WB
div \$13, \$14			IF	ID	EXd	EXd	EXd	EXd	EXd	EXd
add \$5, \$6, \$7				IF	ID	EX	MEM	WB		
mult \$8, \$9					IF	ID	EXm0	EXm1	EXm2	EXm3

Figura 4. Ejecución de instrucciones con una unidad funcional segmentada de 5 ciclos para el producto y una unidad no segmentada de 8 ciclos para la división.

En el ciclo 7 del cronograma de la figura 4 se puede apreciar 3 instrucciones de multiplicación ejecutando simultáneamente su fase *EX* dentro de la misma unidad funcional (*EXm*) aunque en subetapas distintas. El estado del pipeline durante dicho ciclo sería el siguiente:

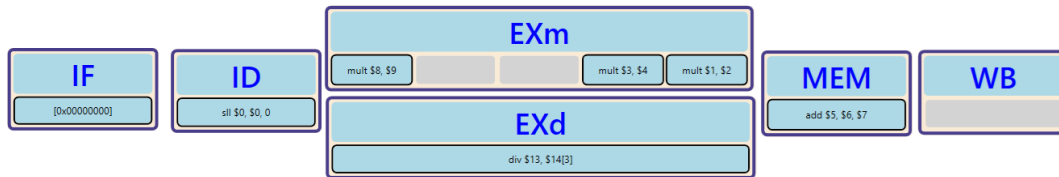


Figura 5. Estado del pipeline en el ciclo 7 de la figura 4. Por simplicidad sólo se están mostrando las unidades funcionales de interés.

3.2. BLOQUEOS DEBIDOS A INSTRUCCIONES DE LARGA DURACIÓN

En el apartado anterior se ha visto que hay instrucciones cuya ejecución de su fase *EX* requiere varios ciclos. La ejecución de estas instrucciones, denominadas instrucciones de larga duración, va a tener una serie de consecuencias sobre el rendimiento:

a) Incremento de los bloqueos de datos por dependencias RAW

Según se ha visto en teoría, los bloqueos por dependencia *RAW* (*read after write*) se producen para evitar que una instrucción posterior lea un operando antes de que otra anterior haya terminado de calcularlo. El aumento de este tipo de bloqueos se debe a que los datos van a estar disponibles en ciclos más tardíos. El uso de adelantamientos (*forwarding*) sólo consigue reducir ligeramente estos ciclos de bloqueo. La siguiente figura muestra este tipo de bloqueo producido por una multiplicación:

	1	2	3	4	5	6	7	8	9	10	11
1: [4194304] mul \$1, \$2, \$3	IF	ID	EXm0	EXm1	EXm2	EXm3	EXm4	MEM	WB		
2: [4194308] add \$4, \$1, \$5		IF	ID	*EX	*EX	*EX	*EX	EX	MEM	WB	
3: [4194312] sub \$6, \$4, \$7			IF	ID	ID	ID	ID	ID	EX	MEM	WB

Figura 6. Ejemplo de bloqueos RAW producidos por una instrucción de larga duración. En la figura, los bloqueos RAW aparecen en rojo, tachado y con un asterisco a su izquierda para simplificar su lectura.

En el ejemplo, puesto que la multiplicación requiere 5 ciclos en *EX* la instrucción de suma se bloquea 4 ciclos a la espera del resultado del producto.

b) Aparición de bloqueos de datos por dependencias WAW

Los bloqueos por dependencia de salida o *WAW* (*write after write*) evitan que se realicen las escrituras sobre un registro en un orden incorrecto.

En figura 7 hay 3 instrucciones que escriben en el mismo registro, si la instrucción suma no se bloqueara durante 4 ciclos, el registro \$1 quedaría actualizado finalmente por la multiplicación en vez de por la suma. En cambio, esto no ocurre con la instrucción resta pues la suma sólo requiere un ciclo en su fase *EX*.

	1	2	3	4	5	6	7	8	9	10	11
1: [4194304] mul \$1, \$2, \$3	IF	ID	EXm0	EXm1	EXm2	EXm3	EXm4	MEM	WB		
2: [4194308] add \$1, \$4, \$5		IF	ID	EX*	EX*	EX*	EX*	EX	MEM	WB	
3: [4194312] sub \$1, \$6, \$7			IF	ID	ID	ID	ID	ID	EX	MEM	WB

Figura 7. Ejemplo de bloqueos WAW. Los bloqueos WAW se muestran en verde, tachado y con un asterisco a su derecha para una mayor legibilidad.

c) Bloqueos estructurales

Aunque en clase de teoría los bloqueos estructurales del MIPS fueron resueltos, al considerar que varias unidades funcionales concurren en la fase *MEM* (véase figura 1), es posible que a varias instrucciones les corresponda avanzar simultáneamente desde la fase *EX* a la única *MEM* existente. Evidentemente sólo una de ellas podrá avanzar (la instrucción más antigua) por lo que el resto deberán ser bloqueadas. Este bloqueo se considera estructural al interpretarse que se producen por falta de recursos en *MEM*. La siguiente figura muestra un ejemplo de este tipo de bloqueos en el que a las instrucciones *mul* y *add* les corresponde abandonar la fase *EX* simultáneamente en el ciclo 7.

	1	2	3	4	5	6	7	8	9	10
1: [4194304] mul \$1, \$2, \$3	IF	ID	EXm0	EXm1	EXm2	EXm3	EXm4	MEM	WB	
2: [4194308] sll \$0, \$0, 0		IF	ID	EX	MEM	WB				
3: [4194312] sll \$0, \$0, 0			IF	ID	EX	MEM	WB			
4: [4194316] sll \$0, \$0, 0				IF	ID	EX	MEM	WB		
5: [4194320] add \$10, \$8, \$9					IF	ID	EX	MEM	WB	

Figura 8. Ejemplo de bloqueo estructural. En el ciclo 7, la instrucción mul va a avanzar a la fase MEM, lo que impide que lo haga add. Los bloqueos estructurales aparecen en amarillo, tachado y con un acento circunflejo a su derecha para una mayor legibilidad.

3.3. CÁLCULO DEL RENDIMIENTO

La forma más habitual de medir el rendimiento de un procesador es mediante el cálculo del tiempo de CPU, esto es, el tiempo que tarda en ejecutar un programa sin tener en cuenta los accesos a la E/S, otros procesos, etc. Si no se considera la frecuencia del reloj, este tiempo puede definirse como:

$$TiempoCPU_{Ciclos} = NumInstruc \cdot CPI \rightarrow CPI = \frac{TiempoCPU_{Ciclos}}{NumInstruc}$$

Cabe recordar que cuando un programa es cargado en memoria, sus instrucciones pasan a formar parte del conjunto de instrucciones ya existentes en memoria (segmento de código) y que el procesador podrá ejecutarlas según especifique el contador de programa (PC) sin distinguir si forman parte o no del programa en cuestión. Por tanto, a la ejecución de las instrucciones del programa le antecedió y le sucederá la ejecución de otras instrucciones ajenas al programa (aunque no sea el caso, incluso podrían ejecutarse otras durante su ejecución: llamadas al sistema, interrupciones, concurrencia de procesos...). Lo anterior unido al hecho de que las instrucciones se ejecutan de forma segmentada (*pipelined*) hace que la tarea de calcular el tiempo de CPU del programa no siempre resulte fácil ya que en el comienzo y finalización de la ejecución del programa también se encuentran en ejecución otras instrucciones ajenas al programa.

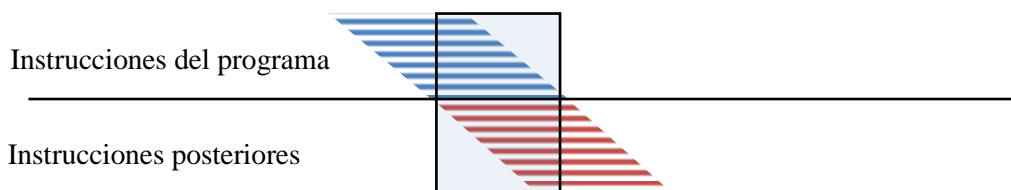


Figura 9. Ejecución segmentada de instrucciones del programa y posteriores.

Como se verá a continuación, la estadística del simulador es un ejemplo de ello: los resultados que muestra contabilizan todas las instrucciones (y ciclos) ejecutadas sean o no del programa a simular.

Una forma de medir este tiempo de CPU del programa a partir del cronograma consiste en contabilizar los ciclos transcurridos desde la etapa ID de la primera instrucción ejecutada del programa hasta la ID de la primera instrucción posterior al programa no inclusive. Esta forma de medir el tiempo, aunque no es infalible, es válida en la mayoría de los casos.

```
.config
    pipelined on      ; pipeline on (en algunas versiones)
    forwarding on
    delayed off
.data
datos: .word 1,3,5,7,9,11,13,15,17,19,21,0
.text
    la $10,datos
rep: lw $5,$0($10)
    beq $5,$0,fin
    sra $5,$5,1
    sw $5,$0($10)
    addiu $10,$10,4
    j rep
fin:
```

Figura 10. Código ejemplo para el cálculo del tiempo de ejecución.

Como ejemplo, véase las figuras 11a y 11b que muestran el cronograma generado por el código de la figura 10. Para medir el tiempo de CPU se toma desde el ID del ciclo 2 hasta el ID del ciclo 108 no incluido por lo que se obtiene: $108-2 = 106$ ciclos (frente a los 109 ciclos que muestra la estadística de la figura 11b).

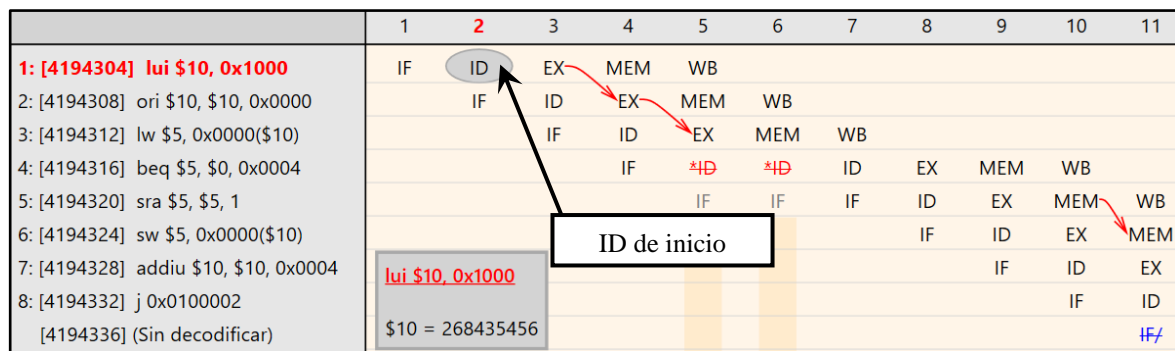


Figura 11a. Comienzo del cronograma del programa anterior

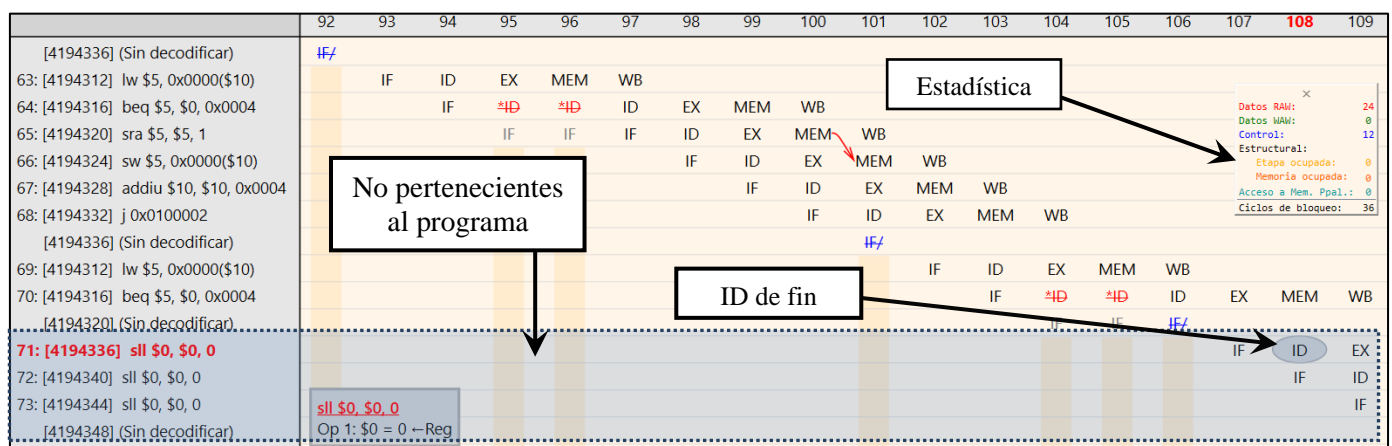


Figura 11b. Fin del cronograma del programa anterior. Las instrucciones posteriores al programa son instrucciones nop que corresponden al código de instrucción 0x00000000 y que se decodifica como sll \$0, \$0, 0.

Igualmente, al contabilizar las instrucciones del programa no ha de tenerse en cuenta las instrucciones canceladas (véase los bloqueos de control) pues en realidad no son instrucciones ejecutadas (evidentemente, la estadística del simulador tampoco las tiene en cuenta). Igualmente, no ha de contabilizarse las instrucciones que no pertenecen al programa como son los nops posteriores (que la estadística del simulador sí las contempla por ser instrucciones ejecutadas, aunque no pertenezcan al programa). En este ejemplo, según se aprecia en la figura 11b, se ejecutan 70 instrucciones del programa pues las instrucciones posteriores al último beq deben ser ignoradas (a pesar de que la estadística del simulador sí las contabilice).

El número de instrucciones puede obtenerse teóricamente mediante el siguiente cálculo: 8 instrucciones de la iteración inicial (1a es una pseudoinstrucción de 2 instrucciones), 6 instrucciones por cada una de las 10 iteraciones intermedias y 2 instrucciones en la iteración final: $8+6*10+2=70$. Respecto al tiempo de CPU, 70 ciclos por la ejecución de las 70 instrucciones, 2 ciclos de bloqueo RAW de lw-beq en cada una de las 12 iteraciones, 1 bloqueo de control por j en cada una de las 11 iteraciones primeras y 1 bloqueo de control por el beq de la última iteración: $70+2*12+1*11+1*1 = 106$

4. SIMULACIÓN MEDIANTE VISUALMIPS32.

Tras abordar el funcionamiento básico del simulador en la práctica anterior a continuación se estudiarán las ventanas relacionadas con la simulación: Pipeline y Cronograma.

La figura 12 muestra la ventana Pipeline durante un ejemplo de simulación. Esta ventana muestra, mediante un esquema similar al mostrado en la figura 1, el estado de cada etapa (o subetapa en el caso de etapas segmentadas) y las instrucciones que se encuentran en ellas durante un determinado ciclo. Cada etapa o subetapa coloreará su borde de un color según el resultado de la ejecución de la instrucción alojada en dicha etapa. Un borde de color negro especifica que la ejecución se ha realizado con éxito y otro color especifica el tipo del bloqueo según la configuración de colores especificada en la ventana configuración. Cada etapa puede moverse y cambiar de tamaño para adaptarse a las necesidades del usuario. También dispone de botones que simplifican la distribución de las etapas en el espacio de trabajo.

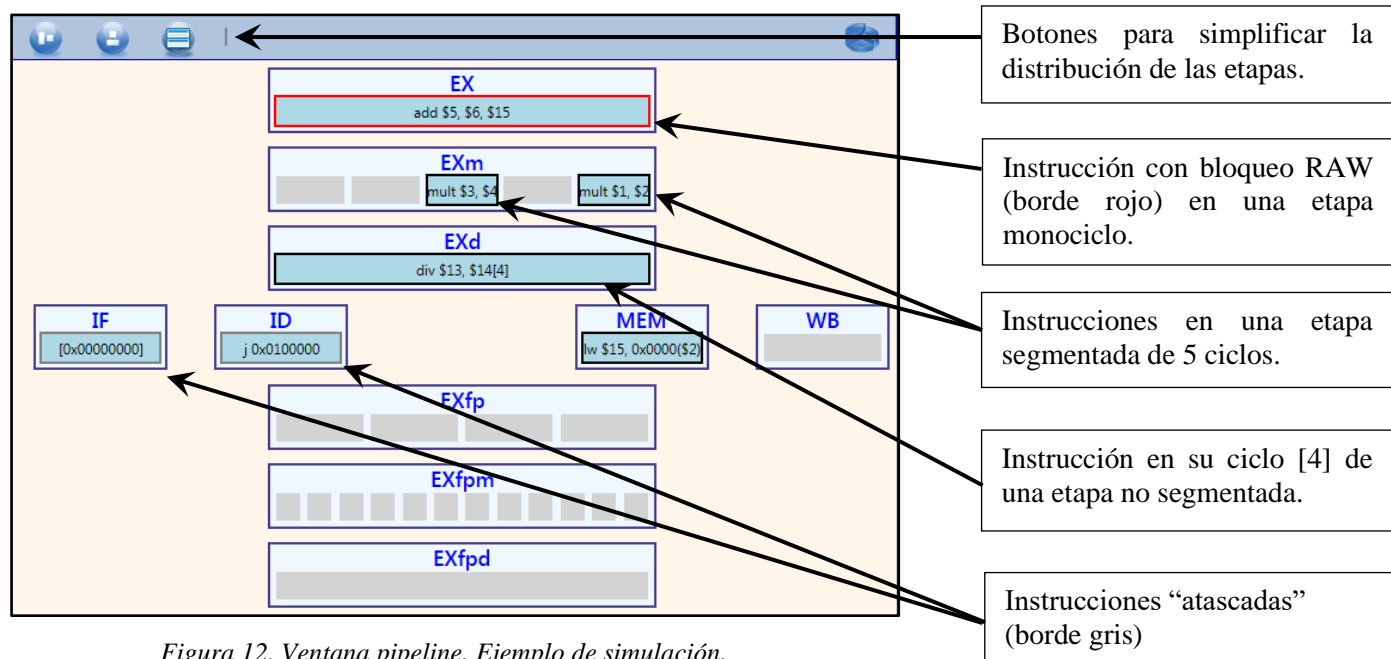


Figura 12. Ventana pipeline. Ejemplo de simulación.

Finalmente, la figura 13 muestra la ventana Cronograma durante un ejemplo de simulación. Esta ventana muestra la ejecución ciclo a ciclo de las instrucciones del código a lo largo del tiempo. Como suele ser lo habitual, el eje de ordenadas especifica las instrucciones y las abscisas el ciclo de ejecución.

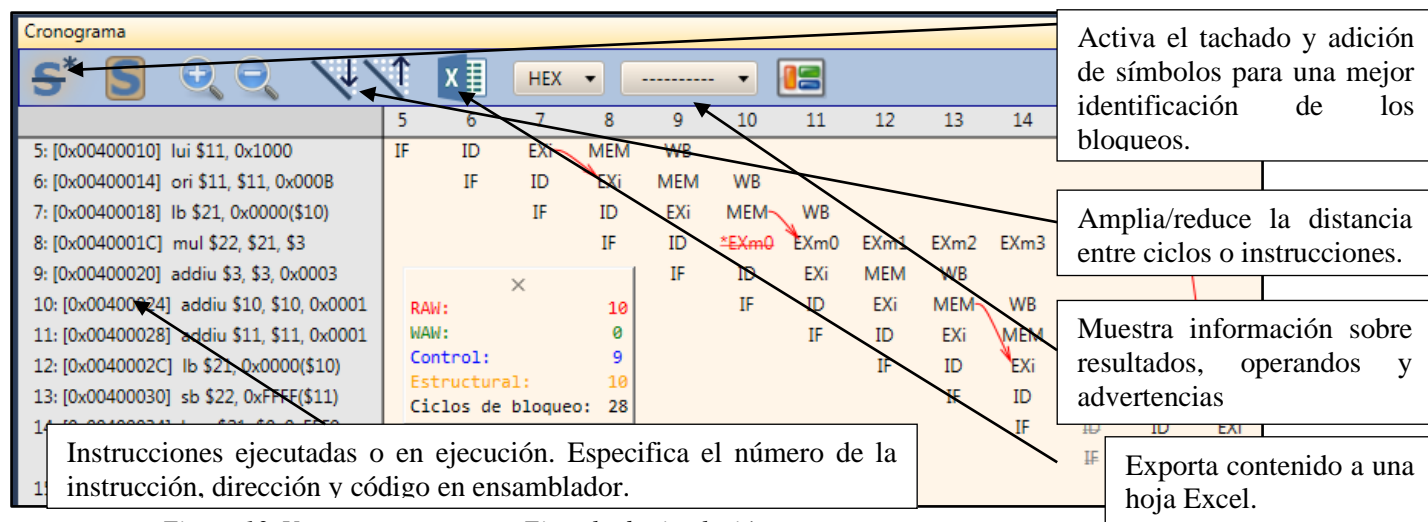


Figura 13. Ventana cronograma. Ejemplo de simulación.

En cada Instrucción/Ciclo se especifica el nombre de la etapa que se está ejecutando y el estado de la ejecución. Si la etapa está segmentada, añade un número al nombre para distinguir la subetapa en la que se encuentra. Por ejemplo, EXm3 se refiere a la subetapa 3 de la unidad funcional de la multiplicación. Respecto al estado, el color de la etapa informa del tipo de bloqueo que se produjo según los colores definidos en la ventana de configuración. Si se activa el botón identificado como S* en la figura 13, los bloqueos son también identificados por símbolos que se añaden al nombre de la etapa. El significado de estos símbolos es:

*EXi	Bloqueo RAW en la fase EXi
EXi*	Bloqueo WAW en la fase EXi
EXi^	Bloqueo estructural en la fase EXi
IF/	Bloqueo de control en IF

A la izquierda de la figura 13 se encuentran las instrucciones enumeradas según su orden de ejecución y con su dirección de memoria. Las instrucciones que finalmente son canceladas no son tenidas en cuenta en esta numeración.

5. EJERCICIOS A REALIZAR EN EL LABORATORIO:

Durante la sesión de laboratorio, el alumno deberá realizar la hoja de ejercicios que le entregue el profesor, rellenarla de forma manuscrita y entregarla al profesor antes de finalizar la práctica. Después de realizar cada ejercicio, el alumno debe mostrar al profesor el funcionamiento de su programa en el simulador.