Arquitectura y Organización de Computadores
Practica 4

Fermín Sola Bienzobas y Eduardo Ezponda Igea

1. Edita el fichero ejemplos/codigo.s y asegúrate de que las instrucciones del Ejercicio 1 no están comentadas:

multf f4, f0, f2

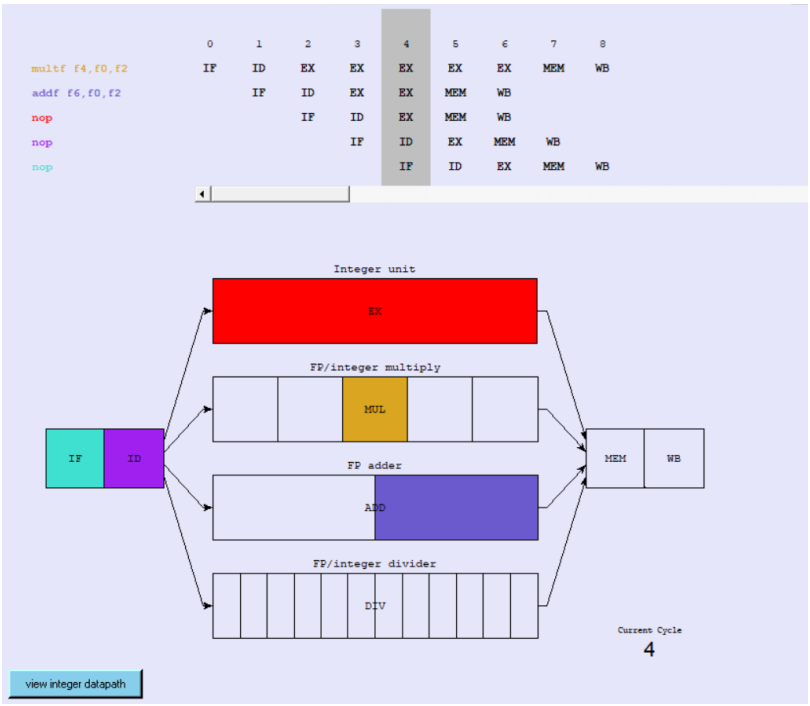
addf f6, f0, f2

Edita el fichero de configuración config.txt y asegúrate de que las unidades funcionales de suma y multiplicación NO están segmentadas.

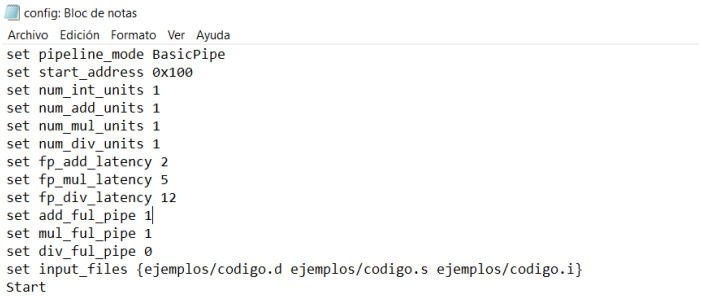
Ejecuta el programa ciclo a ciclo.

P1) ¿Cuántos ciclos tardan en completarse las dos instrucciones?

Las dos instrucciones tardan 8 ciclos en completarse.

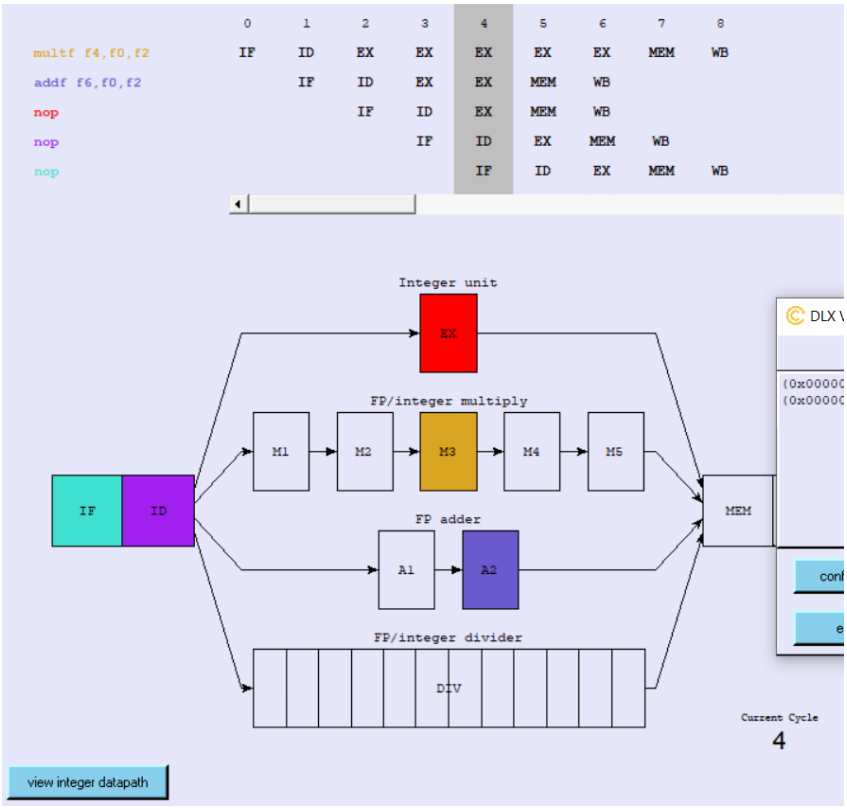


P2) Vuelve a editar el fichero de configuración y haz que las unidades funcionales de suma y de multiplicación SI estén segmentadas.



P3) ¿Se observa alguna mejora? Justifica la respuesta.

En este caso, el número de ciclos también es 8. Con lo cual, no se observa ninguna mejora entre estar segmentado o no estarlo.



2. Edita el fichero ejemplos/codigo.s y asegúrate de que las instrucciones del Ejercicio 2 no están comentadas:

multf f4, f0, f2

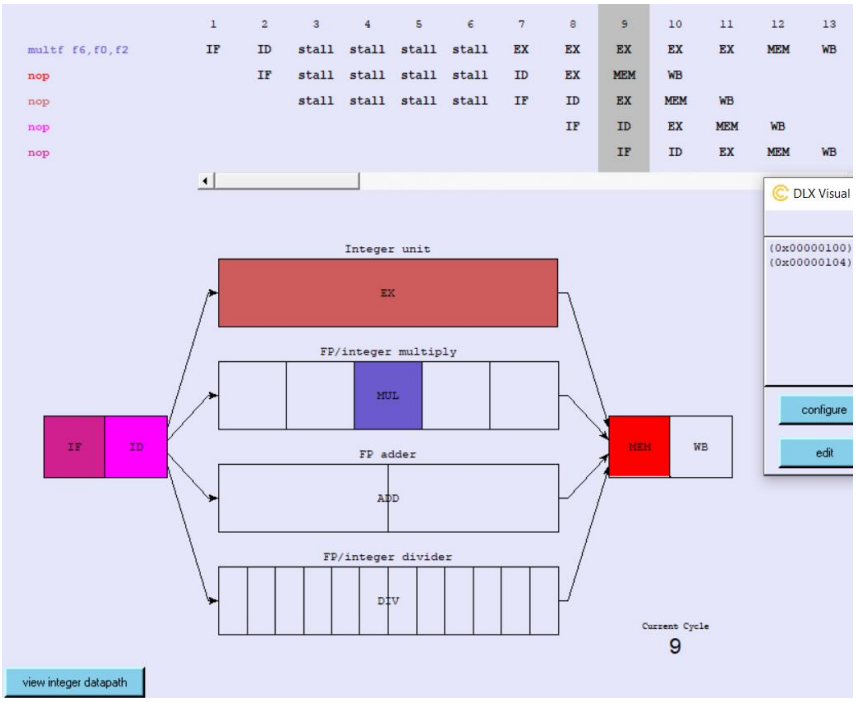
multf f6, f0, f2

Edita el fichero de configuración config.txt y asegúrate de que la unidad funcional de multiplicación NO está segmentada.

Ejecuta el programa ciclo a ciclo.

P1) ¿Cuántos ciclos tardan en completarse las dos instrucciones?

Las dos instrucciones tardan 13 ciclos en ejecutarse.



Vuelve a editar el fichero de configuración y haz que la unidad funcional de multiplicación SI esté segmentada.

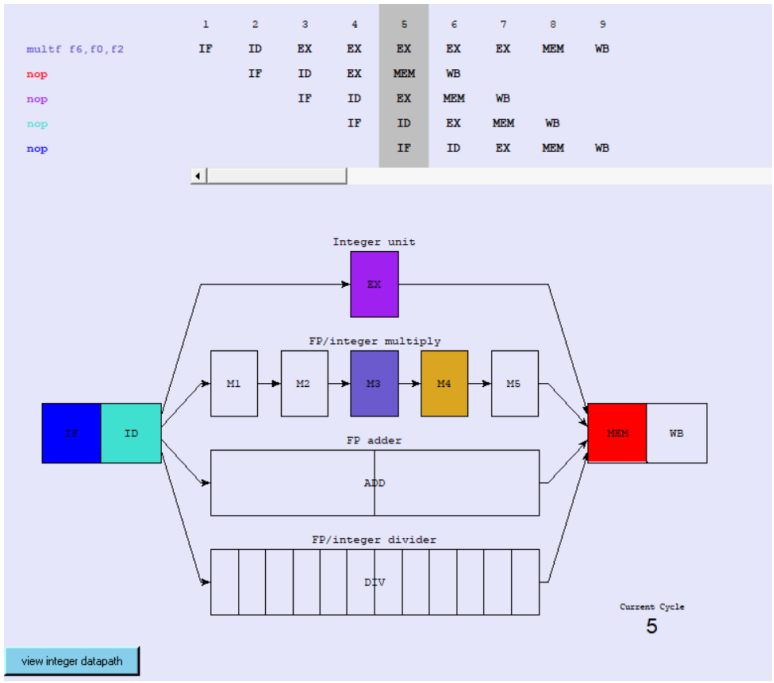
config: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
set pipeline_mode BasicPipe
set start_address 0x100
set num_int_units 1
set num_add_units 1
set num_mul_units 1
set num_div_units 1
set fp_add_latency 2
set fp_mul_latency 5
set fp_div_latency 12
set add_ful_pipe 0
set mul_ful_pipe 1
set div_ful_pipe 0
set input_files {ejemplos/codigo.d ejemplos/codigo.s ejemplos/codigo.i}
Start
```

P2) ¿Se observa alguna mejora? Justifica la respuesta.

Ahora el código tarda 9 ciclos en ejecutarse, por lo que la segmentación implicar una reducción de 4 ciclos (de 13 a 9 ciclos).



3. Edita el fichero ejemplos/codigo.s y asegúrate de que las instrucciones del Ejercicio 3 no están comentadas:

multf f4, f0, f0

addf f6, f0, f1

addf f8, f0, f2

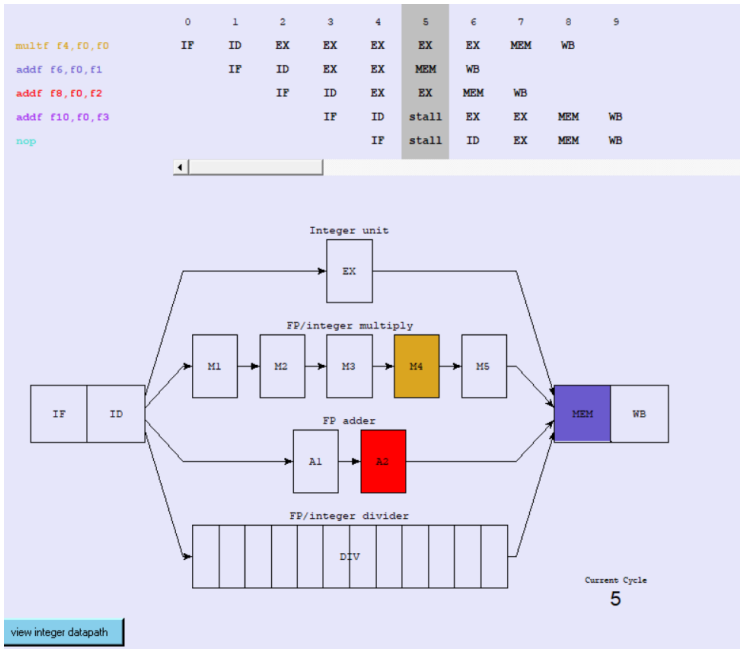
addf f10, f0, f3

Edita el fichero de configuración config.txt y asegúrate de que las unidades funcionales de suma y multiplicación SI están segmentadas (también para el resto de ejercicios en adelante).

Ejecuta el programa ciclo a ciclo.

P1) ¿Por qué se detiene la tercera suma?

Para que no coincida la fase WB de la instrucción 1 con la de la instrucción 3. Al detenerse, la fase WB de la instrucción 3 se hace un ciclo más tarde y no coincide. Esto ocurre porque no se puede hacer más de un acceso a memoria a la vez (en el mismo ciclo).



4. Edita el fichero ejemplos/codigo.s y asegúrate de que las instrucciones del Ejercicio 4 no están comentadas:

add r4, r0, r2

add r6, r0, r4

add r8, r0, r4

add r10, r0, r4

Ejecuta el programa ciclo a ciclo.

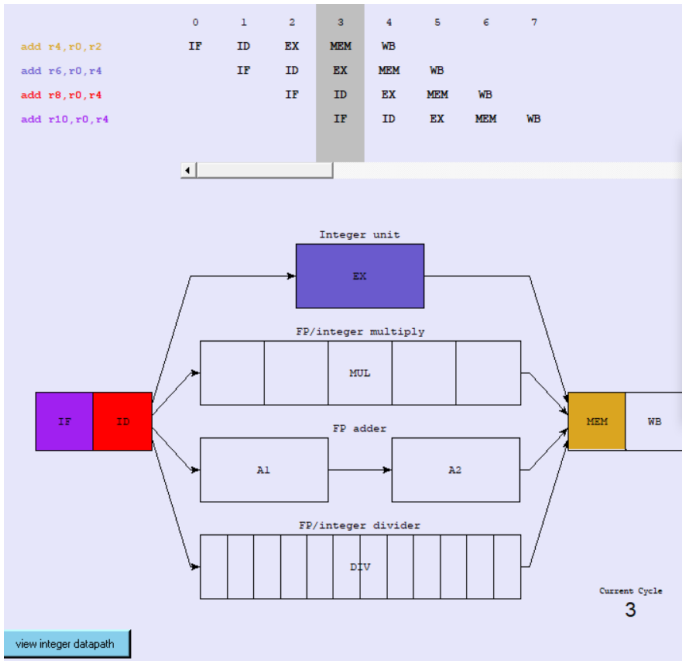
Visualiza el camino de datos para datos enteros.

P1) ¿Qué tipo de dependencia existe entre la primera instrucción y las demás?

Las tres últimas instrucciones dependen del registro r4 al usarlo como operando en la suma. Como en la primera instrucción se modifica el valor del registro r4 usando la operación “add” con r4 como destino, el resto de las instrucciones tendrán que esperar al resultado de la instrucción. Dicha dependencia se conoce como **dependencia de datos (RAW)**.

P2) ¿Cómo es posible que la segunda y la tercera suma puedan ejecutarse antes de que la primera haya escrito el registro r4 y sin que se pierdan ciclos?

Es posible debido a que la primera instrucción escribe en el registro el resultado de la operación en el ciclo 3 (MEM). Además, en ese ciclo, justo la segunda instrucción captura el operando r4 (EXE) que acaba de escribirse, para poder ejecutar dicha instrucción sin perder ningún ciclo.



P3) ¿Es posible que sea un error en la implementación del simulador?

Es posible porque puede que el simulador esté dando por hecho que en el mismo ciclo en que se escribe un valor en el registro, se puede leer de ese registro el valor recién escrito. Sin embargo, puede que en la realidad hubiera que esperar un ciclo para poder leer el nuevo valor escrito y no hacer la lectura y escritura en el mismo ciclo.

5. Edita el fichero ejemplos/codigo.s y asegúrate de que las instrucciones del Ejercicio 5 no están comentadas:

add r4, r0, r0

add r4, r0, r1

add r4, r0, r2

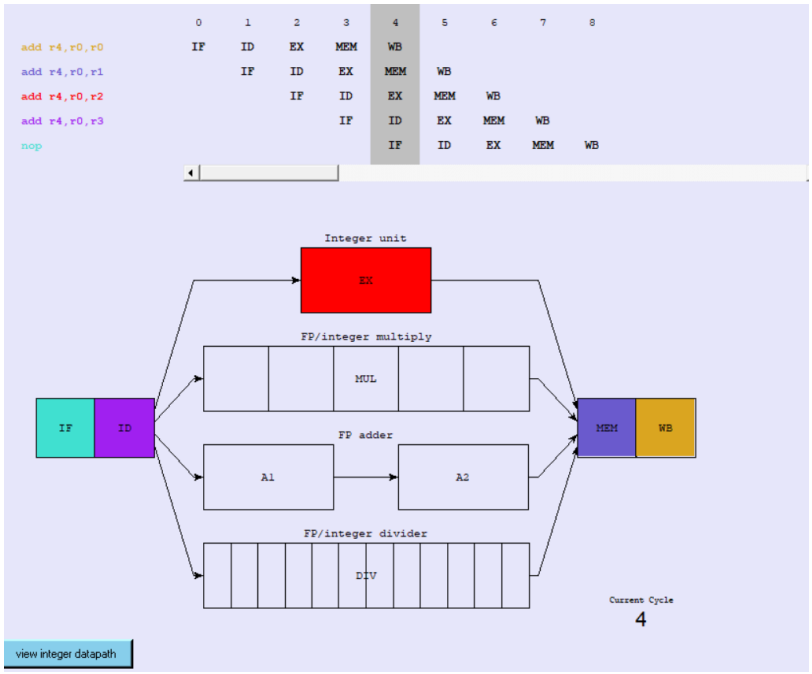
add r4, r0, r3

Ejecuta el programa ciclo a ciclo.

Visualiza el camino de datos para datos enteros.

P1) ¿Qué tipo de dependencia existe entre las instrucciones?

Entre estas instrucciones existe la **dependencia de datos de tipo WAW**, puesto que, al escribirse en todos los casos en el mismo registro, puede pasar que una escritura de más abajo termine antes que una de arriba, y se quede en el registro un valor anterior.



P2) ¿Cómo es posible que las instrucciones puedan ejecutarse sin que se pierdan ciclos?

Anteriormente hemos comentado la dependencia de datos que sufría el código. Sin embargo, como todas las instrucciones tardan 4 ciclos, no llega a ocurrir lo anteriormente comentado, y no se llegan a producir dependencias (no tienen que esperar).

P3) ¿Es posible que sea un error en la implementación del simulador?

En este caso no lo parece, por lo anteriormente comentado. Todas las instrucciones tardan lo mismo y no se producen problemas dependencia WAW.

6. Edita el fichero ejemplos/codigo.s y asegúrate de que las instrucciones del Ejercicio 6 no están comentadas:

multf f4, f0, f0

addf f4, f0, f1

addf f4, f0, f2

addf f4, f0, f3

Configura la unidad funcional de suma para que SI esté segmentada. Observa que este código es similar al anterior, pero para números reales.

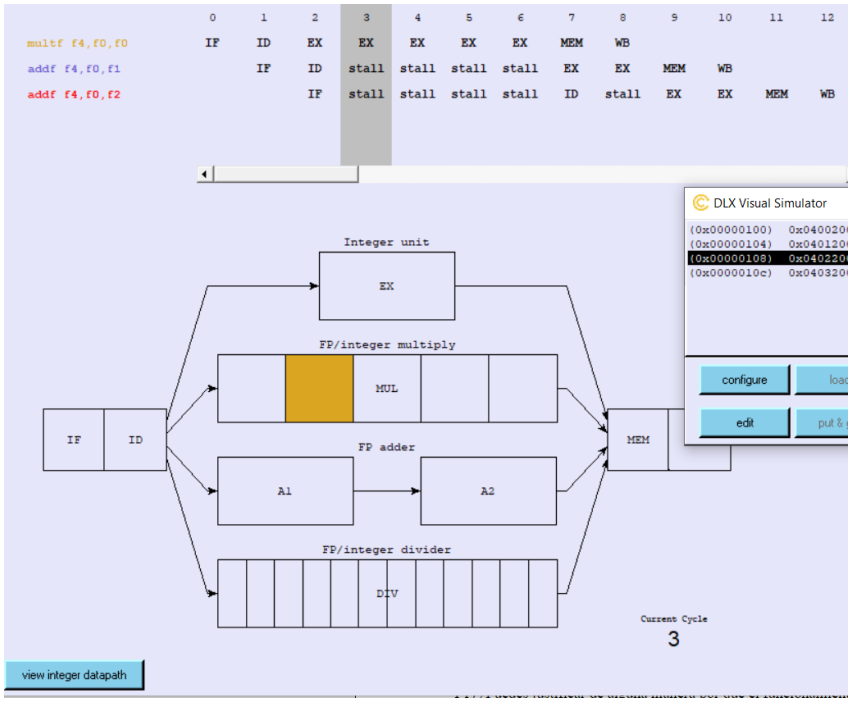
config: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
set pipeline_mode BasicPipe
set start_address 0x100
set num_int_units 1
set num_add_units 1
set num_mul_units 1
set num_div_units 1
set fp_add_latency 2
set fp_mul_latency 5
set fp_div_latency 12
set add_ful_pipe 1
set mul_ful_pipe 0
set div_ful_pipe 0
set input_files {ejemplos/codigo.d ejemplos/codigo.s ejemplos/codigo.i}
Start
```

Ejecuta el programa ciclo a ciclo y observa que en este caso SI se pierden ciclos entre instrucciones.

Como se puede observar, se pierden ciclos (stall), en las dos instrucciones de suma.



Sustituye el anterior código por:

multf f4, f0, f0

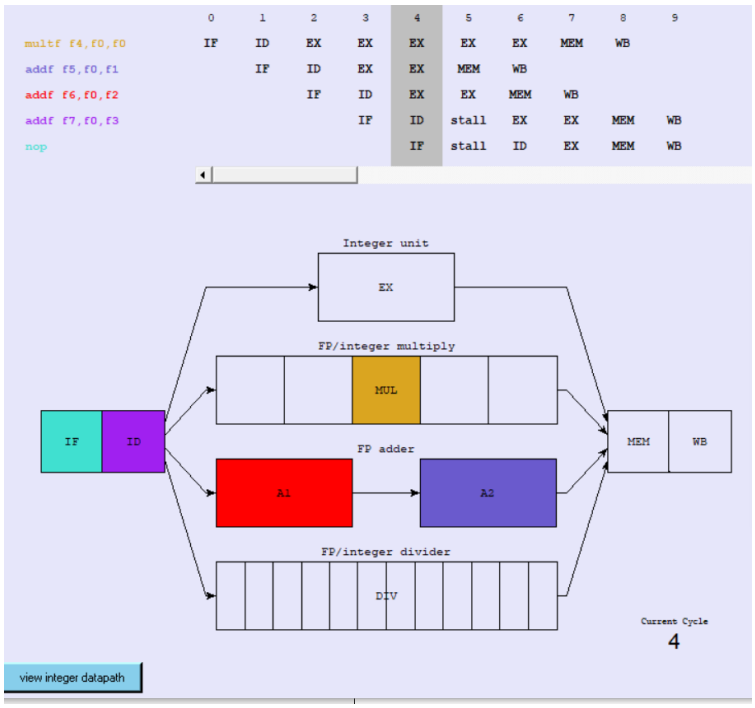
addf f5, f0, f1

addf f6, f0, f2

addf f7, f0, f3

Ejecuta el programa ciclo a ciclo y observa que ahora NO se pierden ciclos entre instrucciones.

En este caso se perdería un ciclo porque la primera instrucción haría el WB en el octavo ciclo, que sería el mismo ciclo en el que escribiría la última instrucción. Por lo tanto, se debe esperar un ciclo por ello. Sin embrago, no se perderían ciclos entre instrucciones.



P1) ¿Puedes justificar de alguna manera por qué el funcionamiento de este simulador es diferente para ambos ejemplos? ¿O crees que es un error en la implementación del simulador?

El funcionamiento es distinto debido a que en las instrucciones del ejercicio 6 hay una dependencia WAW utilizando el registro f4. Por lo tanto, habrá que hacer esperas hasta resolver el resultado.

Sin embargo, sustituyendo el código por las nuevas instrucciones, deja de existir dicha dependencia.

P2) ¿Se te ocurre alguna manera para corregir o mejorar el simulador para que no se pierdan ciclos en casos como el del primer ejemplo?

Para no perder ciclos, se podrían repartir las primeras 4 instrucciones entre las 4 siguientes para que se distribuyan los ciclos y que no se necesite el valor del registro r4 en instrucciones consecutivas (WAW).

7. Edita el fichero ejemplos/codigo.s y asegúrate de que las instrucciones del Ejercicio 7 no están comentadas:

.data 0

a: .double 10

b: .double 20

.text

lf f6, a(r2)

lf f2, b(r3)

multf f0, f2, f4

multf f8, f0, f6

multf f10, f0, f6

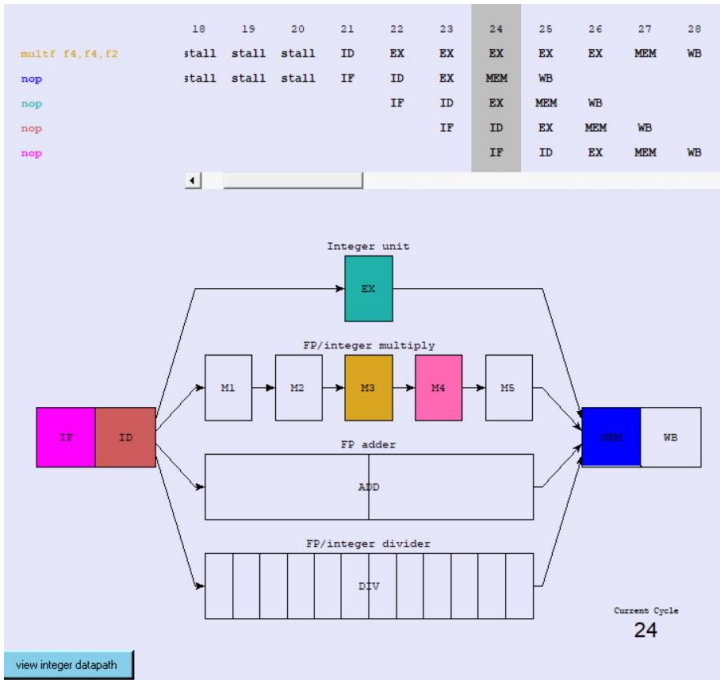
multf f8, f6, f2

multf f6, f8, f2

multf f4, f4, f2

P1) Ejecuta el programa ciclo a ciclo y averigua cuantos ciclos son necesarios para completar todas las instrucciones del código.

Son necesarios 28 ciclos.



P2) Edita el fichero de configuración config.txt y configura el modo de segmentación con el algoritmo de planificación dinámica de Tomasulo y configura el número de estaciones de reserva de la unidad de multiplicación a 4.

NOTA: En la configuración del algoritmo de Tomasulo las variables del fichero de configuración num_xxx_units representan el número de estaciones de reserva para cada tipo de unidad funcional NO el número de unidades funcionales. Se considera una única unidad funcional segmentada para cada tipo. Es igual de eficiente una unidad segmentada con N estaciones de reserva que N unidades no segmentadas (aunque la diferencia de coste es importante).

Ejecuta el programa ciclo a ciclo.

Escribe la secuencia de ejecución de las instrucciones, o lo que es lo mismo, ordena las instrucciones según se ejecutan y comprueba que el orden es diferente al original (se ha realizado una planificación dinámica).

lf f6, a(r2)

lf f2, b(r3)

multf f0, f2, f4

multf f8, f6, f2 --> Esta instrucción por ejemplo se ejecutaba en la prueba anterior más abajo

multf f8, f0, f6

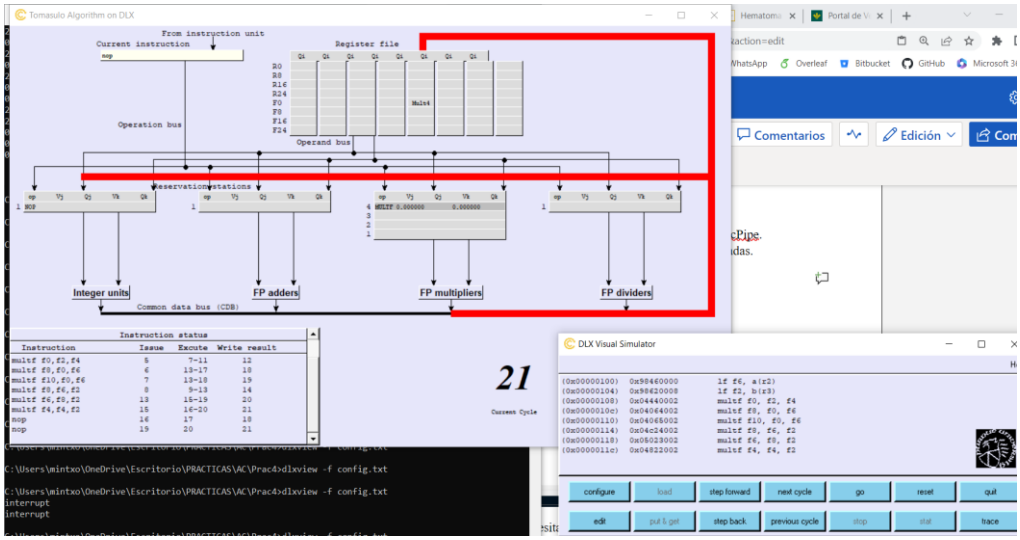
multf f10, f0, f6

multf f6, f8, f2

multf f4, f4, f2

P3) ¿Cuál es la mejora en ciclos obtenida respecto al caso sin planificación dinámica?

Con la planificación de Tomasulo se tardan 21 ciclos, mientras que con la Básica son 28 ciclos. Esto supone una mejora de 7 ciclos.



8. Edita el fichero ejemplos/codigo.s y asegúrate de que las instrucciones del Ejercicio 8 no están comentadas:

```
.data 0
a: .double 10, 11, 12, 13, 14, 15
.text 0x100
loop: lf f0, a(r1)
divd f4, f0, f2
multf f6, f4, f0
multf f10, f2, f0
addf f8, f0, f10
addf f8, f8, f6
sf a(r1), f8
subi r1, r1, #8
bnez r1, loop
nop
nop
nop
```

nop

nop

nop

nop

trap #0

Edita el fichero ejemplos/codigo.i y asegúrate de que las instrucciones de inicialización no están comentadas:

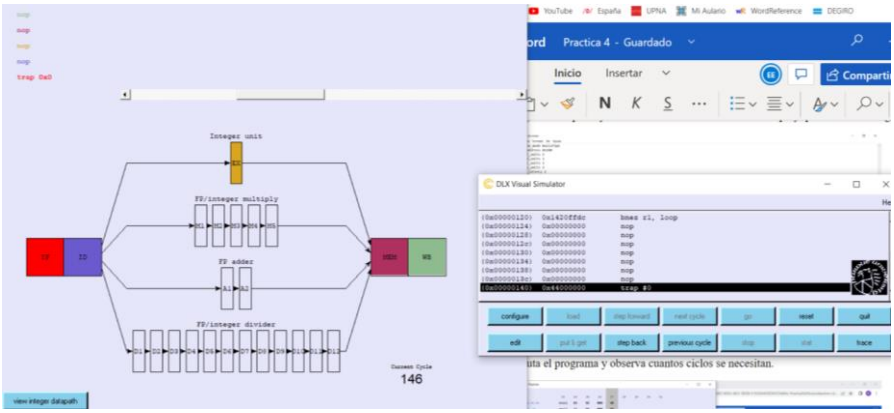
put r1 40

fput f2 2.5 d

Edita el fichero de configuración config.txt y configura el modo de segmentación en BasicPipe. Asegúrate de que hay una única unidad funcional de cada tipo y que todas están segmentadas.

Ejecuta el programa y observa cuantos ciclos se necesitan.

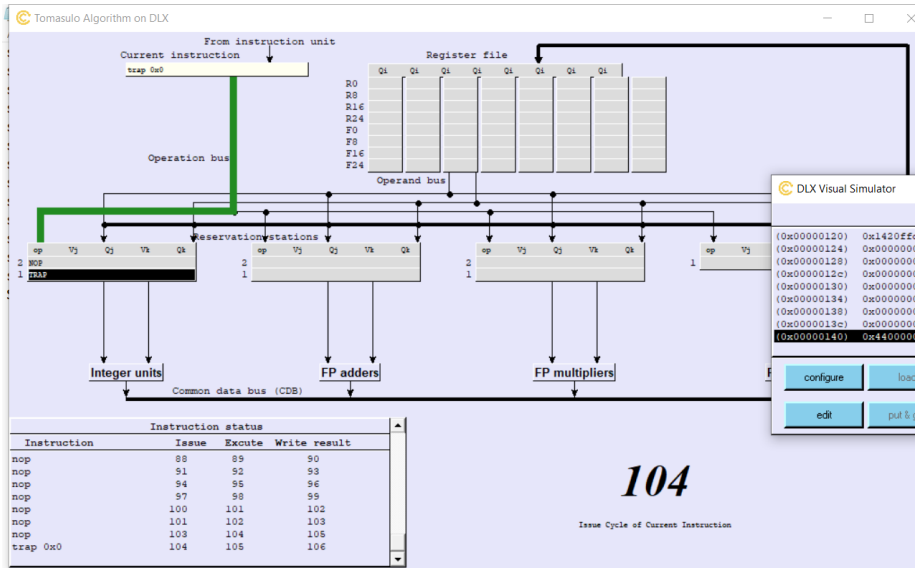
Se necesitan 146 ciclos.



Edita el fichero de configuración config.txt y configura el modo de segmentación con el algoritmo de planificación dinámica de Tomasulo, con 2 estaciones de reserva para la unidad de enteros, con 2 estaciones para la de suma real y con 2 estaciones para la unidad de multiplicación real.

Ejecuta el programa y observa cuantos ciclos se necesitan.

Se necesitan 104 ciclos de ejecución.



config: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
set pipeline_mode Tomasulo
set start_address 0x100
set num_int_units 2
set num_add_units 2
set num_mul_units 2
set num_div_units 1
set fp_add_latency 2
set fp_mul_latency 5
set fp_div_latency 12
set add_ful_pipe 1
set mul_ful_pipe 1
set div_ful_pipe 1
set input_files {ejemplos/codigo.d ejemplos/codigo.s ejemplos/codigo.i}
Start
```

P1) A la vista de los resultados, explica por qué hay tanta diferencia de tiempo de una ejecución a la otra en este para este código en concreto.

Con Tomasulo, la planificación de las instrucciones es dinámica (se reordenan). Esto hace que el total de instrucciones termine antes que con la planificación estándar.

ENTREGA: Un documento “Gxx-P04-DosApellidosA-DosApellidosB.doc” correspondiente al documento tipo Word o OpenOffice con los resultados de las tareas realizadas.