WINDLX

Álvarez Yanes, Rubén (alu2204@etsii.ull.es)
Darias Camacho, Yeray (alu2364@etsii.ull.es)
Monzón Hernández, Juan Ramón (alu2298@etsii.ull.es)

Índice.

1. Interfaz de usuario. 3

1.1. Componentes de la interfaz. 3

1.2. Carga de un ejemplo. 4

2. Dependencias de datos. 5

2.1. Dependencia de datos verdadera (RAW): 5

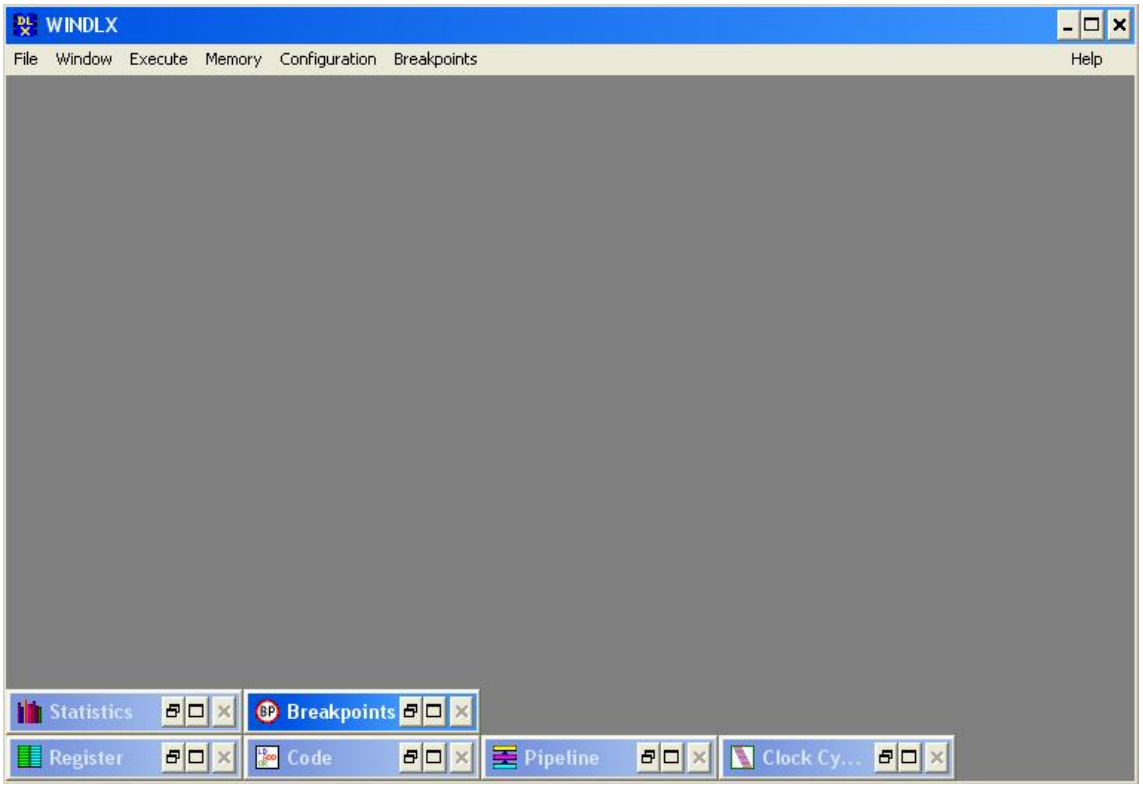
2.2. Antidependencia (WAR): 5

2.3. Dependencia de salida (WAW). 6

1. Interfaz de usuario.

1.1. Componentes de la interfaz.

Al abrir el archivo ejecutable del simulador windlx nos encontraremos con una ventana de apariencia similar a la siguiente que se compone de varias subventanas que se describirán a continuación.



Statistics: Se muestran datos relevantes del código que se simula como el número de ciclos transcurridos, el número de instrucciones en el pipeline o el número de burbujas que se producen entre otros datos de interés.

Breakpoints: En esta ventana se muestran los puntos de ruptura que el usuario haya colocado a lo largo del código para realizar una observación de la traza de forma más sencilla.

Register: Se muestran los valores de los registros de ámbito general así como registros de uso interno de la arquitectura como el program counter u otros similares.

Code: Muestra el código ensamblador que se esta ejecutando.

Pipeline: Es posiblemente una de las ventanas más importantes y muestra el estado actual del pipeline en cada ciclo de reloj, indicando donde se encuentra cada instrucción o si han ocurrido burbujas entre otros detalles.

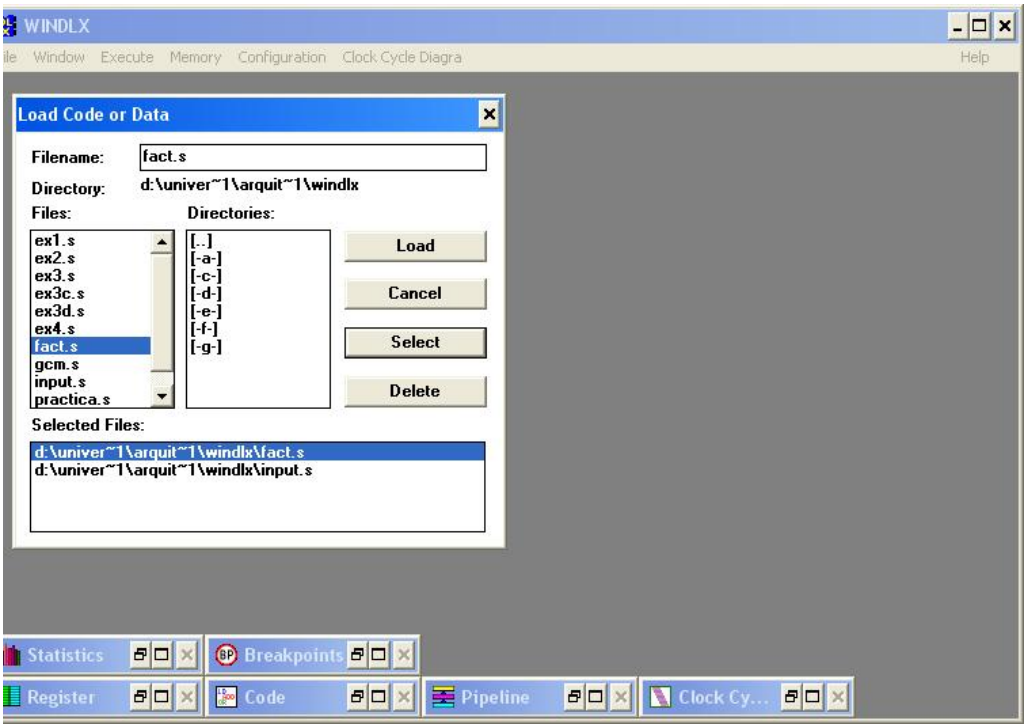
Clock cycle: De manera similar a la ventana pipeline muestra el momento en el que aparecen las instrucciones en el sistema y su paso a través del “procesador”, en cada

ciclo de reloj como indica su nombre. También tiene gran importancia debido a que se puede observar con cierta facilidad cual ha sido el comportamiento de cada instrucción tanto si se produce anticipación como los bloqueos del procesador.

1.2. Carga de un ejemplo.

Para comenzar a usar el simulador el primer paso consiste en reiniciar el programa para que se inicien todos los registros a su valor inicial, por si ha quedado rastro de una ejecución anterior. La opción más sencilla es pulsar resetear todo en la opción “Reset All” dentro del menú File.

Acto seguido se cargan los ficheros .s que se quieren ejecutar, para ello se selecciona la opción “Load code or data” en el mismo menú indicado anteriormente. Se debe tener en cuenta el orden en el que se cargan los ficheros ya que se ejecutaran en ese mismo orden. En el caso concreto que se ha analizado más en profundidad, tratamos los ficheros input.s y fact.s. Se debe notar que hay un botón llamado “Select” que es en el que se seleccionan los ficheros uno a uno. Cuando en los ficheros seleccionados están todos los que deseamos pulsamos “Load” para comenzar a simular el proceso.

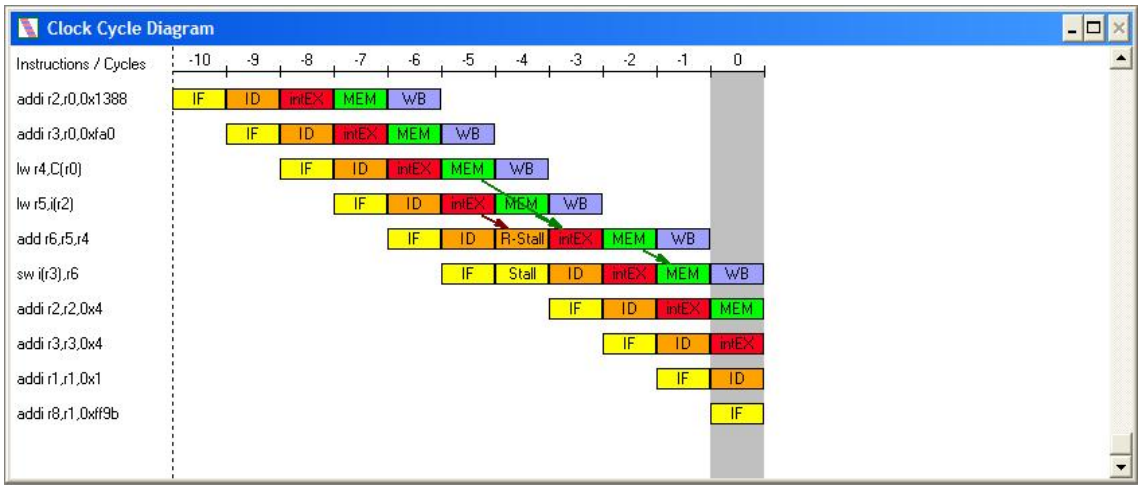


Una vez cargados los ficheros deseados, principalmente se pueden hacer dos cosas (aunque hay más opciones). Se puede pulsar F5 o seleccionar la opción “Run” del menú execute para que se ejecute todo el código. O también se puede pulsar F7 o la opción “Single cycle” para poder seguir la simulación ciclo a ciclo.

2. Dependencias de datos.

2.1. Dependencia de datos verdadera (RAW):

Para explicar mejor este tipo de dependencia utilizaremos el ejemplo ex2.s en el que se pueden observar varias situaciones de riesgo.

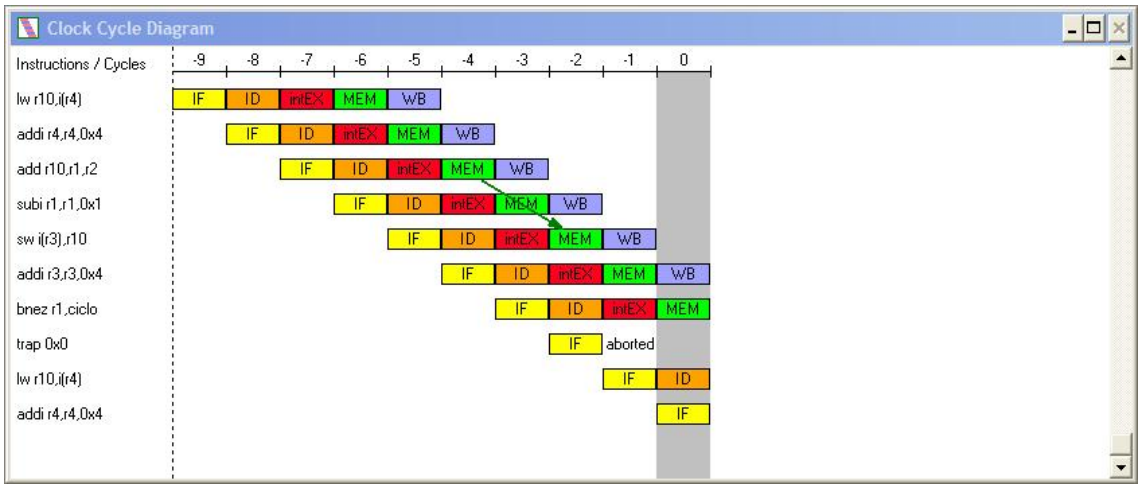


En el diagrama anterior podemos ver que la tercera instrucción carga en el registro 4 que es necesario para la quinta instrucción que suma los registros 4 y 5. En este caso se adelanta el dato sin ningún problema ya que hay una instrucción de por medio y cuando la quinta instrucción requiere el dato en la etapa de ejecución ya ha pasado por la etapa de memoria.

En cambio, en la quinta instrucción se requiere un dato de la instrucción inmediatamente anterior. Esto se soluciona añadiendo una no operación (burbuja) para que la instrucción anterior pueda llegar a la fase de memoria y adelantar el dato correspondiente en la etapa de ejecución. En caso de no existir adelanto de datos esto habría requerido dos no operaciones en vez de una, influyendo negativamente en el rendimiento de la CPU.

2.2. Antidependencia (WAR):

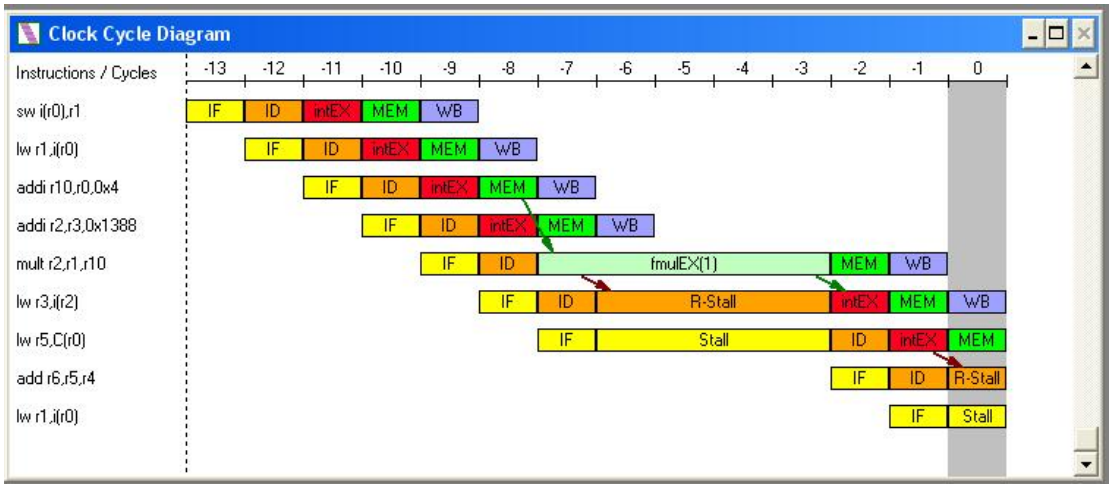
Para comprobar el funcionamiento del pipeline en caso de producirse una antidependecia, hemos modificado el ejemplo ex3.s



Vemos como en este caso, en las instrucciones 3 y 4 se escribe después de haber leído el dato (r1). Esto no ocasiona ningún comportamiento destacable o anómalo en el pipeline ya que no se trata de una dependencia verdadera en la que se necesite los datos antes de que estén actualizados correctamente.

2.3. Dependencia de salida (WAW).

Para comprobar esta dependencia hemos modificado el ejemplo ex1.s para que nos produzca la siguiente salida:



En la cuarta instrucción guardamos en el registro r2 el valor de la instrucción addi, y seguidamente guardamos en el mismo registro el valor de la instrucción mult, provocándose la dependencia WAW. Acto seguido al requerir de nuevo el valor del registro r2 que aún no es accesible porque la operación de multiplicación sigue en la etapa de ejecución (al necesitar varios ciclos de reloj), se produce una parada en el pipeline añadiendo las burbujas necesarias hasta que el valor de la multiplicación sea anticipado a la instrucción de carga. Si así no fuese, el valor del registro r2 sería el de la suma, provocándose un error.