

ARQUITECTURA E
 INGENIERÍA DE
 COMPUTADORES:

Pau Micó

Marcos del Amo Fernández.

Grado en Ingeniería Informática;
 UPV Campus de Alcoy

UD3: EVALUACIÓN DE PRESTACIONES DEL PROCESADOR SUPERESCALAR SDLX

Tabla de contenido

1. Introducción	3
1.1 Objetivos	3
1.2 Metodología	3
2. Simulador SuperDLX.....	3
2.1 Editor de archivos.....	4
2.2 Simulación	4
2.3 Configuración y estadísticas	5
2.3.1 Configuración básica.....	5
2.3.2 Planificación estática	5
2.3.3 Planificación dinámica	7
2.3.4 Extensiones de configuración.....	8
3. SESIÓN 1: Escalar estático vs. dinámico	9
3.1 Escalar estático	9
3.2 Escalar dinámico	11
3.3 Comparativa.....	12
4. SESIÓN 2: Superescalar estático vs. dinámico	15
4.1 Superescalar estático	15
4.2 Superescalar dinámico	23
4.3 Comparativa.....	27
5. Conclusión	28
6. Bibliografía.....	29
7. Anexo: ejemplos de SESCOs	29

1. Introducción

Mediante la resolución de esta práctica el alumno deberá consolidar los conocimientos teóricos adquiridos al respecto de la arquitectura de los microprocesadores superescalares y la mejora de prestaciones que esta tecnología supone respecto a la de la familia de los microprocesadores segmentados. Resulta imprescindible que el alumno haya trabajado previamente tanto los contenidos teóricos como los ejercicios relacionados con este tema.

1.1 Objetivos

- aprender a utilizar un simulador de procesador superescalar llamado SuperDLX
- entender la arquitectura de un procesador superescalar
- evaluar la mejora de prestaciones que un microprocesador superescalar supone respecto a las de un procesador segmentado simple

1.2 Metodología

La duración de la práctica es de 240 minutos, repartidos en dos sesiones de 120 minutos que se utilizarán para la resolución de las cuestiones planteadas en el boletín de la práctica. Al final de la misma (segunda sesión) el alumno deberá presentar (en formato pdf) una memoria justificativa del trabajo realizado estructurada en los siguientes apartados:

- portada identificativa (nombre alumno, curso, título práctica, bloque temático)
- detalle de los contenidos (índice)
- introducción y objetivos
- desarrollo
- conclusiones personales
- bibliografía

La nota obtenida en este acto de evaluación formará parte de la nota de la unidad didáctica dedicada al Procesamiento Superescalar.

2. Simulador SuperDLX

SuperDLX (SDLX) es un simulador de procesador superescalar, basado en el [set de instrucciones del procesador segmentado MIPS](#), que ya deberías conocer. En esta práctica vas a utilizar ensamblador como language de programación, por lo que sería muy interesante que repasaras lo aprendido en la asignatura de Estructura de Computadores.

El fichero instalable para Windows del simulador SuperDLX puedes descargarlo desde [aquí](#).

A continuación se presenta un pequeño tutorial para entender la configuración y funcionamiento del simulador. SDLX consta de tres menús accesibles desde la consola principal: el editor de archivos, el menú de simulación y el menú de configuración y estadísticas.

Para conseguir la simulación de un programa ensamblador sobre el SDLX configurado según especificaciones hay que seguir los siguientes pasos:

1. edición del programa (en el editor del menú de editor de archivos)
2. configuración el simulador (menú de configuración y estadísticas)
3. selección y carga del programa (otra vez, desde el editor de archivos)
4. simulación

2.1 Editor de archivos

En este menú se permite la apertura, edición y carga del programa ensamblador a ejecutar en el simulador. Los ficheros editados son ficheros de texto con extensión 'sdlx'. Para editar un programa en ensamblador podemos escribirlo directamente en el editor de archivos (guardándolo una vez terminado) o cargar un fichero ya existente desde el menú Archivo → Abrir. Una vez tenemos el programa disponible en el editor de SDLX, tenemos que seleccionarlo (botón 'Seleccionar') y carga en el simulador (botón 'Cargar'). La carga del fichero implica una compilación previa del programa en ensamblador. Si la programación no responde a la sintaxis MIPS, la carga del fichero no se completa, generándose un conjunto de mensajes donde se especifican los errores de compilación. Como se ha comentado en el apartado anterior, hay que tener en cuenta que, una vez cargado el programa en cuestión ya NO se puede modificar la configuración del microprocesador superescalar. Es decir, que si queremos configurar el microprocesador, deberemos acudir al apartado de configuración y estadísticas ANTES de realizar la carga (compilación y ensamblado) del programa.

2.2 Simulación

Este apartado sólo se habilita una vez compilado y cargado el código ensamblador a ejecutar. Consta de un cronograma en el que aparecen (en la columna de la izquierda) las distintas instrucciones seleccionadas (identificadas por el número de instrucción). El estado de ejecución de la instrucción aparece en el gráfico de la derecha, en forma de cronograma identificando las etapas con colores. El menú permite definir el número de ciclos de programa a ejecutar, así como su ejecución completa (del tirón). La simulación nos permite identificar las instrucciones, así como los ciclos y las etapas que cada una de ellas va completando con el transcurso del programa. En el momento de analizar la ejecución de un programa, resulta muy útil el activar la opción de rejilla, para poder seguir más fácilmente la instrucción analizada con su correspondencia en el cronograma.

Por otro lado, dentro de la simulación y desde el menú de Ver (o a partir de los botones de la barra), también resulta muy útil el poder inspeccionar tanto el estado de la memoria como el de los registros de propósito general. Al finalizar la simulación podremos comprobar si el resultado de la ejecución del programa ha sido o no el esperado. Además, si hemos configurado el SESC con planificación dinámica, desde el menú de simulación también se nos permite inspeccionar la evolución tanto del buffer de reordenación (ROB) como la de las estaciones de reserva (RS).

2.3 Configuración y estadísticas

A través de este menú se procede al detalle de la configuración de SDLX. El menú se divide en dos apartados claramente diferenciados. La ventana de la izquierda nos permite configurar manualmente las múltiples características del micro. La ventana de la derecha sólo permite consultar los resultados y estadísticos obtenidos al FINAL la ejecución del programa, una vez ha sido cargado en el menú de edición de archivos y ejecutado en la fase de simulación. Nos vamos a centrar en describir las distintas opciones de configuración de SDLX.

OJO: Hay que tener en cuenta que un cambio de configuración en el simulador implica la descarga de memoria del programa simulado hasta ese momento. Una vez reconfigurado el simulador ya se puede volver a cargar el programa e iniciar una nueva simulación.

2.3.1 Configuración básica

La configuración básica del microprocesador es la que va a marcar el resto de opciones. Será el tipo de planificación utilizado (dinámica o estática) el que determine la configuración final del simulador. Mediante el botón que aparece en el apartado de configuración básica podemos acceder a las siguientes características:

- **Grado de superescalaridad (m):** que define el nivel de paralelismo del simulador (Instruction Level Paralelism, ILP) indicando el número de instrucciones que pueden captarse por ciclo de reloj (IPC). Un grado de superescalaridad de 1 se refiere a un microprocesador escalar básico (con distintos cauces al replicar sus unidades funcionales) y que sólo capta UNA instrucción por ciclo (ver configuraciones predefinidas para el escalar con planificación estática y para el escalar con planificación dinámica). Cuando hablamos de un grado de superescalaridad de 2 nos estamos refiriendo a un microprocesador superescalar como tal, capaz de captar varias instrucciones por ciclo (tantas como grado de superescalaridad m)
- **Planificación:** que se refiere a la fase de emisión de las instrucciones y que puede ser estática o dinámica. Dependiendo del tipo de planificación la configuración del hardware en el micro cambia notablemente. En planificación estática el compilador optimiza el código antes de ser ejecutado. En planificación dinámica el hw se complica con una serie de estructuras que optimizan la ejecución de las instrucciones en tiempo real

2.3.2 Planificación estática

La emisión de las instrucciones es estática, o casi estática cuando se basa en las características de la propia instrucción y en ciertos bits del compilador. Es el compilador el que prepara las instrucciones a emitir de manera

ordenada (y rellena con NOP para evitar los posibles riesgos). En este caso de utilizar un microprocesador SESC con planificación estática se deben definir los siguientes parámetros:

- la política de **captación** de instrucciones, que viene determinada por el grado de superescalaridad del micro (número de instrucciones captadas por ciclo). Estas instrucciones quedan almacenadas en una cola (Instruction Buffer, I-Buffer) a la espera de su decodificación. Si la cola se llena el procesador deja de captar instrucciones
- la política de selección de las instrucciones que permanecen en el I-Buffer para proceder a su **decodificación**. Las instrucciones seleccionadas se mantienen en una ventana de instrucciones cuya gestión (política de llenado y vaciado) puede ser:
 - de **ventana fija** (o **alineada**) que se carga por completo con las instrucciones cargadas en el I-Buffer. La ventana fija se va vaciando tal y como se emiten sus instrucciones y no se recarga con nuevas instrucciones hasta que no se vacía por completo
 - **ventana deslizante** (o **desalineada**) que se carga parcialmente con las instrucciones del I-Buffer. La ventana deslizante se recarga tal y conforme se emiten sus instrucciones de manera que aparece llena en todo momento
- la política de **emisión** (o reglas de emisión) que determina el orden en que las instrucciones de la ventana se enviarán a la etapa de ejecución. Las reglas se aplican a cada uno de los slots de la ventana (número de reglas = tamaño de la ventana). Estas reglas de emisión pueden ser:
 - reglas de **emisión rígida u ordenada** es cuando las instrucciones se envían a las UFs en el mismo orden en el que se encuentran en el programa. En este caso las reglas se definen sobre cada uno de los slots de la cola de instrucciones, bien por UF, bien por tipo de instrucción. En cuanto los slots del mismo tipo se llenan y la siguiente instrucción requiere ese tipo de slot, se generan ciclos de parada hasta que se resuelve el riesgo estructural (se libera un slot de ese tipo). El motivo por el que los slots aparecen numerados es porque necesitamos conocer el orden de entrada de las instrucciones (para emitirlas ordenadamente)
 - reglas de **emisión flexible o desordenada** es cuando las instrucciones se emiten por disponibilidad de operandos y UFs, sin la necesidad de seguir el orden del programa. Las mismas consideraciones que en el caso anterior. La única diferencia es que, en este caso, los slots NO aparecen numerados (no es necesario emitir las instrucciones en orden)
- **Unidades Funcionales:** donde se nos permite configurar el número y latencias de las distintas UFs del SESC. En el caso de las unidades ADD/SUB y Div en FP, también podemos definir si están o no segmentadas
- **Bypasses:** donde se habilita el bypass (forwarding) entre las diferentes etapas del micro SESC. El forwarding minimiza el efecto (retardo) de los riesgos por dependencia de datos y, en este caso, todos los bypasses aparecen activados por defecto. El color del bypass correspondiente es el que luego aparece representado con una flecha (entre las etapas en las que se activa el bypass) en el menú de simulación

- **Puertos de escritura GPR-FPR:** donde se define el número de puertos de escritura del SESC. Hay dos tipos de puertos, los registros de propósito general (General Purpose Registers, GPRs) y los registros de coma flotante (Floating Point Registers, FPRs)

2.3.3 Planificación dinámica

En este caso, el procesador (ID) decide en tiempo de ejecución qué instrucciones pueden emitirse a la vez y qué instrucciones no. Es similar al [algoritmo de Tomasulo](#). Así, la configuración del SESC difiere completamente del caso anterior, por lo que los nuevos apartados a configurar serán los siguientes:

- la política de **captación** de instrucciones, que viene definido por el grado de superescalaridad del micro (número de instrucciones captadas por ciclo). Estas instrucciones quedan en la cola de instrucciones (Instruction Buffer, I-Buffer) a la espera de decodificación
- la política de selección de las instrucciones que permanecen en el I-Buffer para proceder a su **decodificación**. Las instrucciones seleccionadas se mantienen en una ventana de instrucciones (tamaño de la ventana = grado de superescalaridad tenga el micro). La gestión de esta ventana (política de llenado y vaciado) puede ser:
 - de **ventana fija** (o **alineada**): ver caso de planificación estática
 - **ventana deslizante** (o **desalineada**): ver caso de planificación estática
- en el caso de planificación dinámica la política de **emisión** (orden en que las instrucciones de las estaciones de reserva se envían a ejecución) será SIEMPRE **flexible** o **desordenada** (por eso no hace falta especificarlo en el simulador). En este caso el envío no se denomina ISSUE sino DISPATCH. Resulta importante destacar cómo, para la emisión de un SESC de planificación estática se deben configurar una serie de reglas asociadas a cada instrucción/UF. Esto NO ocurre cuando el SESC presenta planificación dinámica ya que es el mismo hw, en tiempo de ejecución, el que se encarga de encontrar las instrucciones que se pueden emitir
- hardware para la **ejecución**: aquí se determinan las características de las Estaciones de Reserva (RS), el número de Unidades Funcionales (UFs), las entradas del Buffer de Reordenamiento (ROB) o el número de Buses Comunes de Datos (BCD) presentes en el microprocesador.
 - **RS:** donde definimos el número de slots de cada estación (tamaño de la estación de reserva, es decir, cuántas instrucciones pueden esperar en ella) y la latencia de la UF asociada a esa estación de reserva. A tener en cuenta cómo, si modificamos la latencia de la RS, automáticamente queda modificada la latencia en la UF (lógicamente)
 - **UF:** donde definimos el número de unidades funcionales de cada tipo que presenta al SESC. Como hemos comentado, la latencia de la UF se puede determinar desde este apartado o desde el apartado anterior (RS)
 - **ROB:** para configurar el tamaño del buffer de reordenación y el número máximo de operaciones de commit (finalizaciones) a ejecutar por ciclo

- **BCD:** para configurar el ancho de banda del Bus Común de Datos que determinará el número de escrituras simultáneas que se pueden realizar sobre el banco de registros

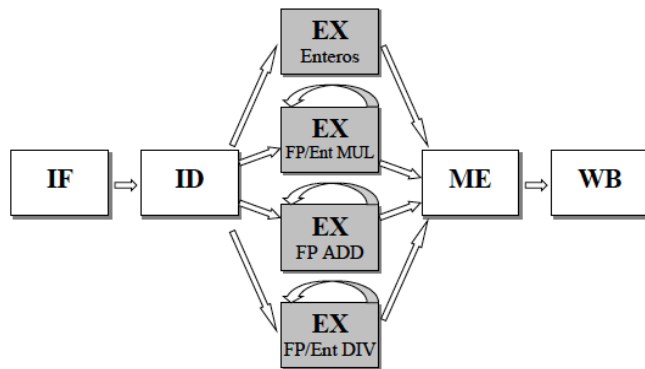
2.3.4 Extensiones de configuración

Finalmente cabe resaltar como punto muy interesante la opción que nos da el simulador de poder guardar las configuraciones realizadas (en ficheros con extensión .cps) o cargar configuraciones por defecto. Esta opción aparece (desde la opción de Configuración y Estadísticas) en la barra de menús como Configuración Procesador. En el caso que nos ocupa, existen cuatro configuraciones por defecto que responden a los procesadores escalar dinámico, escalar estático, superescalar dinámico y superescalar estático.

3. SESIÓN 1: Escalar estático vs. dinámico

En esta PRIMERA SESIÓN de la práctica vas a realizar una comparación entre las prestaciones obtenidas con un microprocesador escalar (con grado de superescalaridad $m = 1$) configurado para que resuelva las dependencias entre instrucciones de forma estática (las resuelve el compilador, en tiempo de compilación) y las prestaciones obtenidas por el mismo micro escalar configurado para resolver las dependencias dinámicamente (las resuelve el microprocesador mediante estructuras hardware, en tiempo de ejecución). Para la comparación se va a utilizar el programa de ejemplo saxpy.dlx (lo puedes cargar desde el menú de edición de archivos). Como podrás imaginar en este programa se codifica la operación vectorial clásica de $Y = a * X + Y$, donde X e Y son vectores de la misma longitud y a es un escalar. El programa utiliza un bucle para recalcular las componentes de Y guardándolas sobre el mismo vector.

DLX ENCADENADO CON VARIAS U.F.



Ejemplo de microprocesador escalar con cauces paralelos que incorpora varias unidades funcionales

3.1 Escalar estático

Para configurar un procesador escalar con planificación estática utiliza los siguientes parámetros (configuración por defecto de escalar estático):

- superescalaridad = 1
- planificación estática
- UFs: 2 INT, 2 Add/Sub FP (2 ciclos latencia), 2 Mult FP (5 ciclos) y 1 Div FP (19 ciclos, segmentada)
- Todos los bypasses activos
- 1 puerto GPR y 1 puerto FPR

Ejecuta ahora la simulación del programa saxpy.sdlx y contesta a las siguientes cuestiones:

- a partir de las instrucciones en ensamblador ([set de instrucciones del procesador segmentado MIPS](#)) explica el funcionamiento del programa. ¿Qué se calcula? ¿Dónde se guardan los resultados? Además compueba que no existen errores sintácticos compilando y cargando el programa en el simulador.

El programa `saxpy.sdlx` sobrescribe el vector `Y`, siendo el nuevo vector el resultado de la operación $Y = a * X + Y$. No existen ningún tipo de error sintáctico, el programa se carga, compila y ejecuta correctamente en el simulador.

- **segmento de datos:** ¿a partir de qué dirección de memoria se carga el vector `X`?, ¿y el vector `Y`? **Segmento de instrucciones:** ¿a partir de qué dirección de memoria se empiezan a cargar las instrucciones de programa? OJO, en el programa, los vectores de datos se cargan en direcciones de memoria especificadas en hexadecimal y los datos se muestran en coma flotante de doble precisión (Flotante D.P., 64 bits). En cambio, si queremos comprobar cómo se distribuyen las instrucciones a partir de su dirección de carga tenemos que seleccionar como tipo de dato Word (registros de instrucciones 32 bits). ¿Qué directivas se utilizan para especificar las direcciones de inicio del segmento de datos y el de instrucciones de programa?

Segmento de datos:

Vector `X`, a partir de la dirección de memoria `0x400`

Vector `Y`, a partir de la dirección de memoria `0x500`

Para especificar su inicio se usa la directiva `.data`

Segmento de instrucciones:

Instrucciones del programa, se cargan a partir de la dirección `0x64`.

Para especificar su inicio se usa la directiva `.text`

- el resultado obtenido, ¿es el correcto? (para ello comprueba las componentes del vector `Y`, a partir de la dirección de memoria `0x500`, se recomienda una inspección de la memoria con datos de tipo flotante de doble precisión, ver Figura)

El resultado obtenido es el correcto, tenemos que a partir de la dirección `0x500`, vemos los valores correspondientes al resultado de la operación anteriormente comentada.

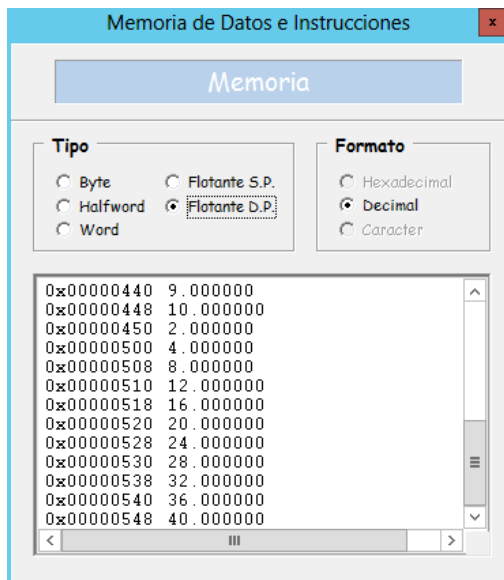
$Y = [4, 8, 12, 16, 20, 24, 28, 32, 36, 40]$

- ¿cuántos ciclos tarda en ejecutarse este programa, con la configuración indicada? y ¿qué CPI se obtiene?

El siguiente programa tarda en ejecutarse 163 ciclos.

El número de instrucciones ejecutadas es 109.

Por lo que el $CPI = 163/109 = 1,49$. Ciclos/instrucción.



Vista de la memoria a partir de la dirección de carga de Y, a la finalización de la ejecución del programa

3.2 Escalar dinámico

Para configurar un procesador escalar con planificación dinámica utiliza los siguientes parámetros (configuración por defecto de escalar dinámico):

- superescalaridad = 1
- planificación dinámica
- estaciones de reserva: 3 INT (1 ciclo latencia), 2 Add/Sub FP (2 ciclos), 3 Mult FP (5 ciclos), 3 Div FP (19 ciclos), 3 Load (1 ciclo), 3 Store (1 ciclo)
- ROB de 20 entradas con número máximo de reordenamientos por ciclo = 2
- 2 buses comunes de datos

Ejecuta ahora la simulación del programa saxpy.sdlx y contesta a las siguientes cuestiones:

- el resultado obtenido, ¿es el correcto?

Ejecutando la simulación del programa nuevamente obtenemos el resultado correcto.

- ¿qué CPI se obtiene? (míralo desde el cronograma, en el apartado de simulación, en este caso las estadísticas están desactivadas)

Observando el cronograma, obtenemos:

Ciclos ejecutados = 133

Instrucciones ejecutadas = 127

$CPI = 133/127 = 1,047$ ciclos/instrucción.

3.3 Comparativa

Una vez realizadas y entendidas las simulaciones del apartado anterior contesta a las siguientes cuestiones:

Recordemos los datos obtenidos de las cuestiones anteriores:

1. Planificación Estática datos obtenidos:

Ciclos ejecutados = 163 ciclos

Instrucciones ejecutadas = 109 instrucciones

El número de instrucciones ejecutadas es 109.

$CPI = 163/109 = 1,49$. Ciclos/instrucción.

Planificación Dinámica datos obtenidos:

Observando el cronograma, obtenemos:

Ciclos ejecutados = 133ciclos

Instrucciones ejecutadas = 127 instrucciones

$CPI = 133/127 = 1,047$ ciclos/instrucción.

- ¿cuántas instrucciones se ejecutan en el escalar estático?, ¿cuántas en el dinámico?, ¿por qué el número de instrucciones es distinto en uno y otro caso?

En el escalar estático se ejecutan 109, mientras que en el dinámico 127, esto se debe a que en planificación estática, el código se optimiza en tiempo de compilación, en cambio en el dinámico se realiza en tiempo de ejecución.

ejecución mediante estructuras hardware. Por lo que es el mismo número de instrucciones útiles pero las abortadas son distintas.

- *¿qué tipo de planificación ofrece unos mejores resultados de ejecución?, ¿por qué?*

El escalar dinámico nos ofrece unos mejores resultados de ejecución, pues este obtiene un CPI y tiempo de ejecución menor que el escalar estático. A pesar de tener más instrucciones.

- *a partir de las latencias de las diferentes unidades funcionales, ¿que mejora de velocidad de procesamiento se obtiene para cada uno de los casos (escalar estático/dinámico) respecto de la ejecución del mismo código en un microprocesador segmentado que minimice los riesgos por dependencia de datos y de control? Sugerencia: trata de ejecutar el código en el simulador DLX y ajusta el tiempo de ciclo al peor caso (latencia de la unidad funcional más lenta)*

En el escalar estático:

$S = T(\text{sin mejorar}) / T(\text{mejorado}).$

$$S = (TLI + (N^{\circ}\text{ciclos} - 5) * 19) / 163 = (95 + 158 * 19) / 163 = 19$$

En el escalar dinámico:

$S = T(\text{sin mejorar}) / T(\text{mejorado}).$

$$S = (TLI + (N^{\circ}\text{ciclos} - 5) * 19) / 163 = (95 + 128 * 19) / 133 = 19$$

- *del apartado anterior, ¿has podido ejecutar el código saxpy.dlx sobre el micropocesador segmentado DLX?, ¿aparecen todas las instrucciones del programa en el set de instrucciones del DLX?*

No se ha podido ejecutar, hay instrucciones que no se encuentran en el set instrucciones del DLX. Por ejemplo, las de double.

- *a la vista del resultado del apartado anterior y desde el punto de vista de las prestaciones obtenidas en la ejecución del programa de ejemplo ¿es mejor una planificación estática o una dinámica? Justifica la respuesta*

A la vista del resultado del apartado anterior y desde el punto de vista de las prestaciones obtenidas en la ejecución del programa de ejemplo, la planificación dinámica es una mejor opción.

- *¿qué mejora de velocidad se obtendría en cada caso respecto de la ejecución del mismo código en un procesador NO segmentado?*

Para calcular la mejora o el speed up respecto de un NO segmentado, primeramente calculamos cuantos ciclos tardaría el programa.

Para la configuración estática tenemos 109 instrucciones, y para la dinámica 127, asumiendo que el mismo tipo de instrucción tarda lo mismo en ambos procesadores y cuando el simulador no especifique el tipo de instrucción, asumiremos que tarda un ciclo.

CONFIGURACIÓN ESTÁTICA

LOAD	21*1=21
ADD	33*2=66
MULT	10*5=50
STORE	10*1=10
OTROS	35*1
TOTAL	182 ciclos

Speed Up estático: $182/163 = 1,1165$

CONFIGURACIÓN DINÁMICA

LOAD	21*1=21
ADD	33*2=66
MULT	10*5=50
STORE	10*1=10
OTROS	53*1
TOTAL	200 ciclos

Speed Up dinámico: $200/133=1,502$

4. SESIÓN 2: Superescalar estático vs. dinámico

En esta SEGUNDA SESIÓN de la práctica vas a realizar una comparación entre las prestaciones obtenidas de la ejecución del código utilizado en la sesión anterior (*saxpy.dlx*) sobre un microprocesador superescalar con planificación estática versus el mismo superescalar configurado con planificación dinámica.

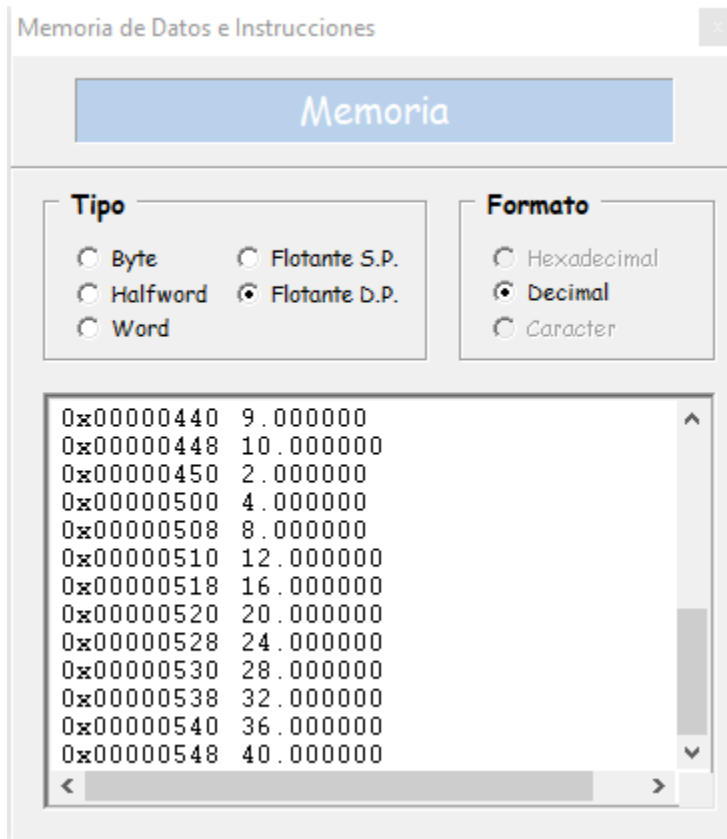
4.1 Superescalar estático

Para configurar un procesador superescalar con planificación estática utiliza los siguientes parámetros (configuración por defecto de superescalar estático):

- superescalaridad = 2
- planificación estática
- emisión alineada y ordenada
- tubería 1 asignada a las UFs: INT, Add/Sub FP, Mult FP y Div FP
- tubería 2 asignada a las UFs: INT, Add/Sub FP, Mult FP y Div FP
- UFs: 2 INT, 2 Add/Sub FP (2 ciclos latencia), 2 Mult FP (5 ciclos) y 1 Div FP (19 ciclos, segmentada)
- Todos los bypasses activos
- 1 puerto GPR y 1 puerto FPR

Ejecuta ahora la simulación y contesta a las siguientes cuestiones:

- el resultado obtenido, ¿es el correcto? (para ello comprueba las componentes del vector Y, a partir de la dirección de memoria 0x500, se recomienda una inspección de la memoria con datos de tipo flotante de doble precisión)



El resultado obtenido es el correcto, como se puede observar a partir de la dirección 0x500

- *¿qué CPI se obtiene?*

Ciclos ejecutados = 139 ciclos

Instrucciones ejecutadas = 109 instrucciones

$CPI = 139/109 = 1,275$ ciclos/instrucción

- ¿cuántos bloqueos aparecen y de qué tipo? (de datos, de control y/o estructurales)

```

• CICLO ACTUAL: 139
• NÚMERO DE INSTRUCCIONES DECODIFICADAS (ID/IS): 109
• CONTEO DE INSTRUCCIONES:
  - Número de instrucciones LOAD: 21 (19.27%)
  - Número de instrucciones STORE: 10 (9.17%)
  - Número de instrucciones INT: 48 (44.04%)
    de las cuales ADD INT: 23 (47.92%)
    de las cuales LÓGICAS INT: 10 (20.83%)
  - Número de instrucciones PUNTO FLOTANTE: 20 (18.35%)
    de las cuales Add FP: 10 (50.00%)
    de las cuales MULT FP: 10 (50.00%)
    de las cuales DIV FP: 0 (0.00%)
    de las cuales LÓGICAS FP: 0 (0.00%)
  - Número de instrucciones de SALTO: 10 (9.17%)
    de las cuales son SALTOS CONDICIONALES: 10 (100.00%)
    de las cuales son SALTOS INCONDICIONALES: 0 (0.00%)
  - Número de instrucciones ABORTADAS: 0 (0.00%)
• BLOQUEOS DE DATOS:
  - Bloqueos RAW: 0 (0.00%)
  - Bloqueos WAW: 0 (0.00%)
• BLOQUEOS ESTRUCTURALES: 0 (0.00%)
• FALLOS DE SALTO: 0 (0.00%)
    
```

No existen bloqueos, 0 de datos y 0 estructurales.

- ¿por qué I7-I8 no entran en IF hasta el ciclo 6, en lugar de hacerlo en el ciclo 4?

Instrucciones\Ciclos	1	2	3	4	5	6	7	8
#1(1) ld d0,0x450(r0)	IF	ID	EX 1	ME	WB			
#2(2) addi r1,r0,1024	IF	ID	EX 1	ME	WB			
#1(3) addi r2,r0,1280		IF	ID	EX 1	ME	WB		
#2(4) addi r4,r0,1104		IF	ID	EX 1	ME	ME	WB	
#1(5) inicio: LD d1,0x0(r1)			IF	ID	EX 1	ME	WB	
#1(6) multd d2,d1,d0			IF	ID	ID	ID	EX 1	EX 2
#1(7) ld d3,0x0(r2)						IF	ID	EX 1
#1(8) addd d3,d2,d3						IF	ID	ID
#1(9) sd 0x0(r2),d3								
#2(10) addi r1,r1,8								

Dado que la ventana fija es de 2, la instrucción 7 debe esperar, hasta que se decodifique la instrucción 6.

- ¿por qué I8 no se emite hasta el ciclo 12, si ha sido decodificada en el ciclo 7?

Instrucciones\Ciclos	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
#1(1) ld d0,0x450(r0)	IF	ID	EX 1	ME	WB											
#2(2) addi r1,r0,1024	IF	ID	EX 1	ME	WB											
#1(3) addi r2,r0,1280		IF	ID	EX 1	ME	WB										
#2(4) addi r4,r0,1104		IF	ID	EX 1	ME	ME	WB									
#1(5) inicio: LD d1,0x0(r1)			IF	ID	EX 1	ME	WB									
#1(6) multd d2,d1,d0			IF	ID	ID	ID	EX 1	EX 2	EX 3	EX 4	EX 5	ME	WB			
#1(7) ld d3,0x0(r2)				IF	ID	EX 1	ME	WB								
#1(8) addd d3,d2,d3				IF	ID	ID	ID	ID	ID	ID	EX 1	EX 2	ME	WB		
#1(9) sd 0x0(r2),d3										IF	ID	EX 1	ME	WB		

La I8 no se emite hasta el ciclo 12 porque debe esperar la ejecución de la I6, I8 utiliza elementos de la I6, por lo que hasta que no termine de ejecutar mantendrá a I8 en la ventana de instrucciones.

- ¿por qué se aborta I14?

La instrucción se aborta porque es un nop y no tiene nada que ejecutar, es una instrucción no operate.

- ¿qué significa que, por ejemplo, para la I13 en el ciclo 18, la etapa de WB aparezca de color más pálido? (también sucede en I19, ciclo 28, etc).

En la I13, la etapa WB aparece más pálido porque una vez que se resuelve el salto y al haber forwarding, las dos otras etapas no hacen nada.

- ¿qué diferencia hay (en etapas de color más pálido) entre las instrucciones de LOAD y una de STORE y?, ¿por qué?

La diferencia reside en la etapa WB, que en el caso de la instrucción LOAD se carga la variable en memoria en un registro y en la instrucción STORE se almacena la variable de un registro en memoria, por lo que no hay escritura.

Sin cambiar el tipo de planificación ni el grado de superescalaridad, ¿se puede configurar el microprocesador para conseguir mejor CPI? (sugerencia: prueba emisión no alineada y desordenada).

```

• CICLO ACTUAL: 138
• NÚMERO DE INSTRUCCIONES DECODIFICADAS (ID/IS): 109
• CONTEO DE INSTRUCCIONES:
  - Número de instrucciones LOAD: 21 (19.27%)
  - Número de instrucciones STORE: 10 (9.17%)
  - Número de instrucciones INT: 48 (44.04%)
    de las cuales ADD INT: 23 (47.92%)
    de las cuales LÓGICAS INT: 10 (20.83%)
  - Número de instrucciones PUNTO FLOTANTE: 20 (18.35%)
    de las cuales ADD FP: 10 (50.00%)
    de las cuales MULT FP: 10 (50.00%)
    de las cuales DIV FP: 0 (0.00%)
    de las cuales LÓGICAS FP: 0 (0.00%)
  - Número de instrucciones de SALTO: 10 (9.17%)
    de las cuales son SALTOS CONDICIONALES: 10 (100.00%)
    de las cuales son SALTOS INCONDICIONALES: 0 (0.00%)
  - Número de instrucciones ABORTADAS: 0 (0.00%)
• BLOQUEOS DE DATOS:
  - Bloqueos RAW: 0 (0.00%)
  - Bloqueos WAW: 0 (0.00%)
• BLOQUEOS ESTRUCTURALES: 0 (0.00%)
• FALLOS DE SALTO: 0 (0.00%)
    
```

- indica la nueva configuración y el CPI obtenido

La nueva configuración es con emisión no alineada y desordenada:

Ciclos ejecutados: 138

Instrucciones ejecutadas: 127

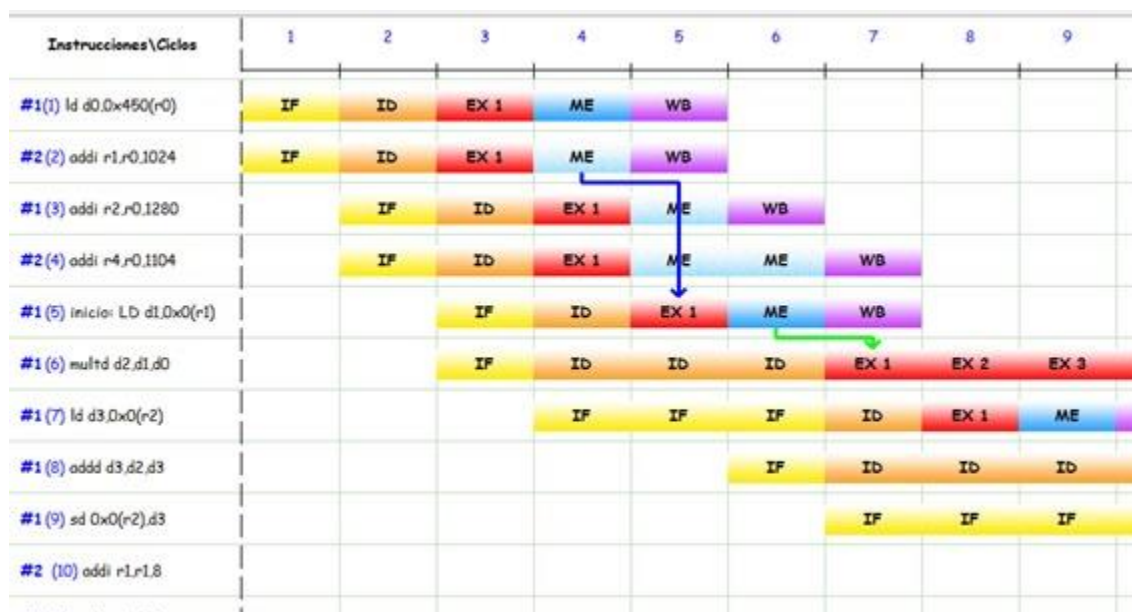
$CPI = 138/127 = 1,0866$ ciclos/instrucción.

- ¿cuántos bloqueos aparecen y de qué tipo? (de datos, de control y/o estructurales).

Nuevamente no aparecen ni bloqueos de datos ni bloqueos de control y/o estructurales.

- ¿por qué las I7 e I8 no entran en la etapa ID hasta el ciclo 7?

I7 e I8 entran en etapa de decodificación en el ciclo 7, ya que acaba la etapa ID de la instrucción anterior I6, es decir, se emite, ya que I6 solo puede entrar en ejecución una vez que I5 ha terminado la ejecución, por dependencia.



- ¿por qué se abortan las I14, I15 e I16?

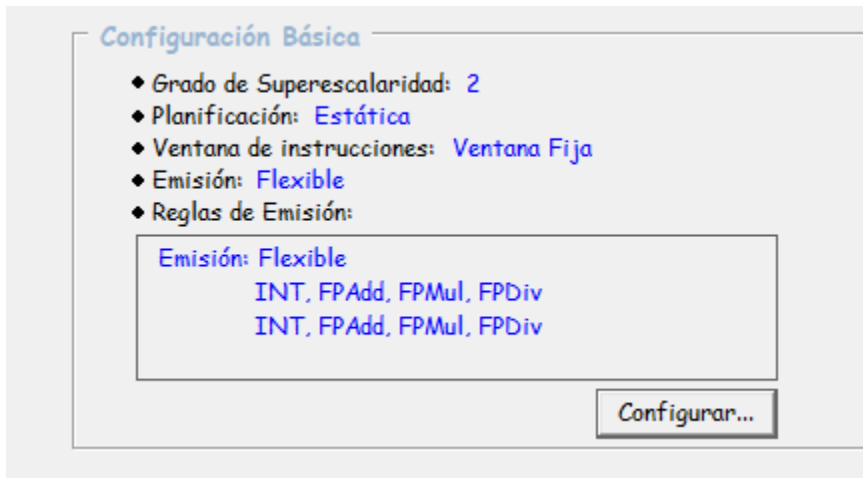
Porque son instrucciones nop.

- a la vista del número de instrucciones abortadas en este caso y en el anterior justifica, en términos de eficiencia, cuál de las dos configuraciones te parece óptima.

Dado que las dos configuraciones nos proporcionan una cantidad de bloqueos igual y al esta terminar un ciclo antes, diríamos que esta configuración es más óptima.

- ¿cómo aparecen configuradas las reglas de emisión? ¿Qué sucede si asignamos la #1 a la UF de enteros (INT) y la #2 al resto de UFs?

Las reglas de emisión aparecen configuradas de la misma forma que la anterior, en las dos tuberías esta cada una de las UFs. Si asignamos la #1 a la UF de enteros y la #2 el resto, observamos que pierde capacidad de ejecutar dos instrucciones iguales al mismo tiempo, y por consiguiente, obtenemos un tiempo de ejecución mayor, de 163 ciclos:



The screenshot shows a window titled 'Configuración Básica' with the following settings:

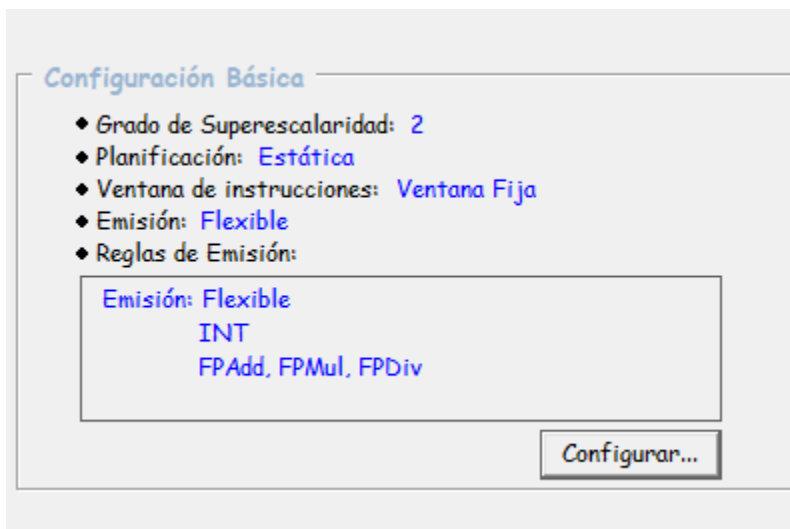
- ◆ Grado de Superescalaridad: 2
- ◆ Planificación: Estática
- ◆ Ventana de instrucciones: Ventana Fija
- ◆ Emisión: Flexible
- ◆ Reglas de Emisión:

Below the 'Reglas de Emisión' label, there is a box containing the text:

Emisión: Flexible
INT, FPAdd, FPMul, FPDiv
INT, FPAdd, FPMul, FPDiv

A 'Configurar...' button is located at the bottom right of the window.

Así es como está actualmente, después del cambio sugerido tenemos que:



The screenshot shows the same 'Configuración Básica' window, but with updated emission rules:

- ◆ Grado de Superescalaridad: 2
- ◆ Planificación: Estática
- ◆ Ventana de instrucciones: Ventana Fija
- ◆ Emisión: Flexible
- ◆ Reglas de Emisión:

Below the 'Reglas de Emisión' label, there is a box containing the text:

Emisión: Flexible
INT
FPAdd, FPMul, FPDiv

A 'Configurar...' button is located at the bottom right of the window.

Ahora con esta configuración obtenemos el siguiente resultado:

```

• CICLO ACTUAL: 163
• NÚMERO DE INSTRUCCIONES DECODIFICADAS (ID/IS): 109
• CONTEO DE INSTRUCCIONES:
  - Número de instrucciones LOAD: 21 (19.27%)
  - Número de instrucciones STORE: 10 (9.17%)
  - Número de instrucciones INT: 48 (44.04%)
    de las cuales ADD INT: 23 (47.92%)
    de las cuales LÓGICAS INT: 10 (20.83%)
  - Número de instrucciones PUNTO FLOTANTE: 20 (18.35%)
    de las cuales ADD FP: 10 (50.00%)
    de las cuales MULT FP: 10 (50.00%)
    de las cuales DIV FP: 0 (0.00%)
    de las cuales LÓGICAS FP: 0 (0.00%)
  - Número de instrucciones de SALTO: 10 (9.17%)
    de las cuales son SALTOS CONDICIONALES: 10 (100.00%)
    de las cuales son SALTOS INCONDICIONALES: 0 (0.00%)
  - Número de instrucciones ABORTADAS: 0 (0.00%)
• BLOQUEOS DE DATOS:
  - Bloqueos RAW: 0 (0.00%)
  - Bloqueos WAW: 0 (0.00%)
• BLOQUEOS ESTRUCTURALES: 0 (0.00%)
• FALLOS DE SALTO: 0 (0.00%)
    
```

- desde tu punto de vista, ¿es adecuada la traza que presenta el simulador SDLX en cuanto a la cola de instrucciones se refiere?

La traza que presenta el simulador SDLX no es adecuada en cuanto a la cola de instrucciones se refiere, esta tiene problemas en la captación de instrucciones (etapa IF).

- si nos referimos al grado de superescalaridad, ¿qué diferencias presenta el simulador SDLX con el comportamiento de un SESC real?, el grado de superescalaridad del SDLX ¿se refiere al número de instrucciones captadas por ciclo (IF), o también se puede configurar independientemente el número de instrucciones decodificadas por ciclo (ID) como en un superescalar real?

Atendiendo al grado de superescalaridad en el SDLX, no podemos configurar de forma independiente el número de instrucciones decodificadas por ciclo (ID), esto causa los problemas comentados en el apartado anterior. En un superescalar real podemos hacer este tipo de configuraciones y obtener resultados distintos. El grado de superescalaridad del SDLX se refiere al número de instrucciones captadas por ciclo IF.

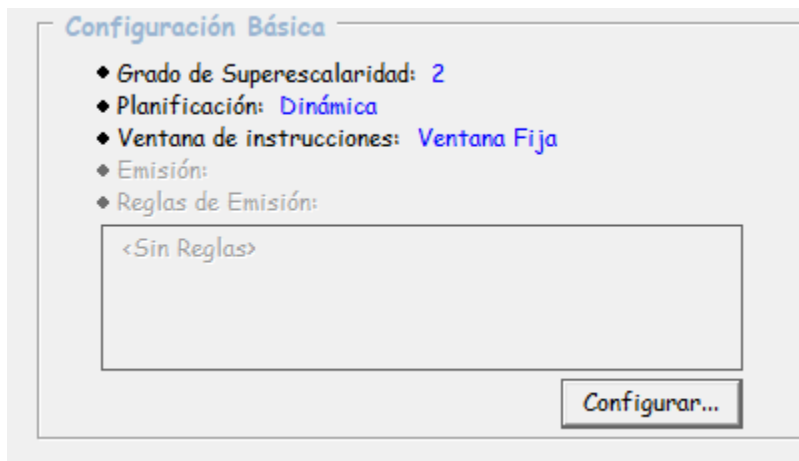
4.2 Superescalar dinámico

Ahora vas a realizar una simulación de la ejecución del mismo código pero sobre un microprocesador superescalar con planificación dinámica (resuelve las dependencias en tiempo de ejecución mediante las estructuras hardware correspondientes), con la siguiente configuración (configuración por defecto de superescalar dinámico):

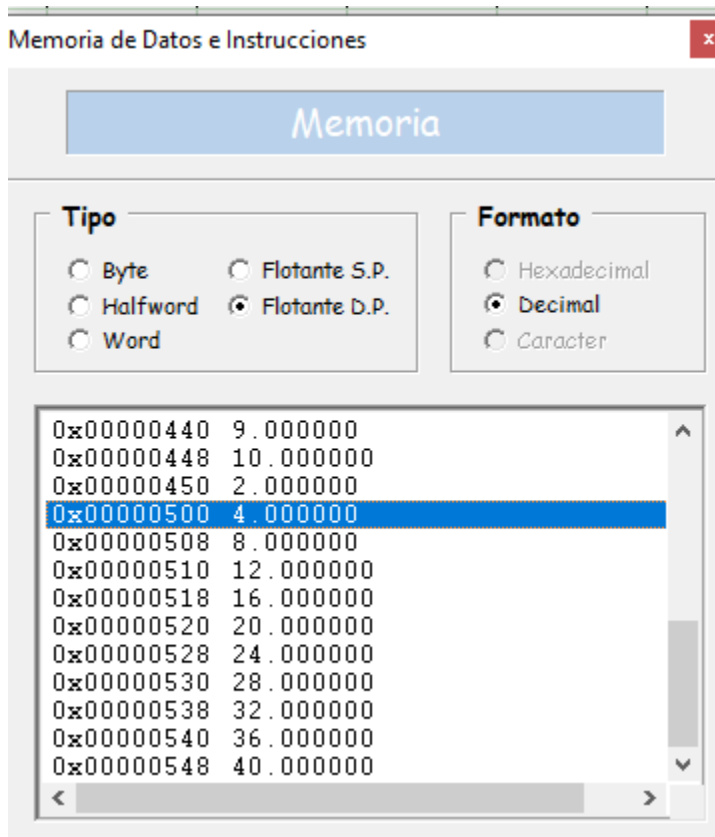
- superescalaridad = 2
- planificación dinámica con ventana de instrucciones fija
- estaciones de reserva: 3 INT (1 ciclo latencia), 2 Add/Sub FP (2 ciclos), 3 Mult FP (5 ciclos), 3 Div FP (19 ciclos), 3 Load (1 ciclo), 3 Store (1 ciclo)
- ROB de 20 entradas y número máximo de reordenamientos por ciclo = 2
- 2 buses comunes de datos

Ejecuta ahora la simulación y contesta a las siguientes cuestiones:

Adjunto configuración correspondiente:



- el resultado obtenido, ¿es el correcto?



El resultado obtenido es nuevamente el correcto, se aprecia a partir de la dirección de memoria 0x0500, el resultado de las componentes del vector Y.

- ¿qué CPI se obtiene? (míralo desde el cronograma, en el apartado de simulación, en este caso las estadísticas están desactivadas)

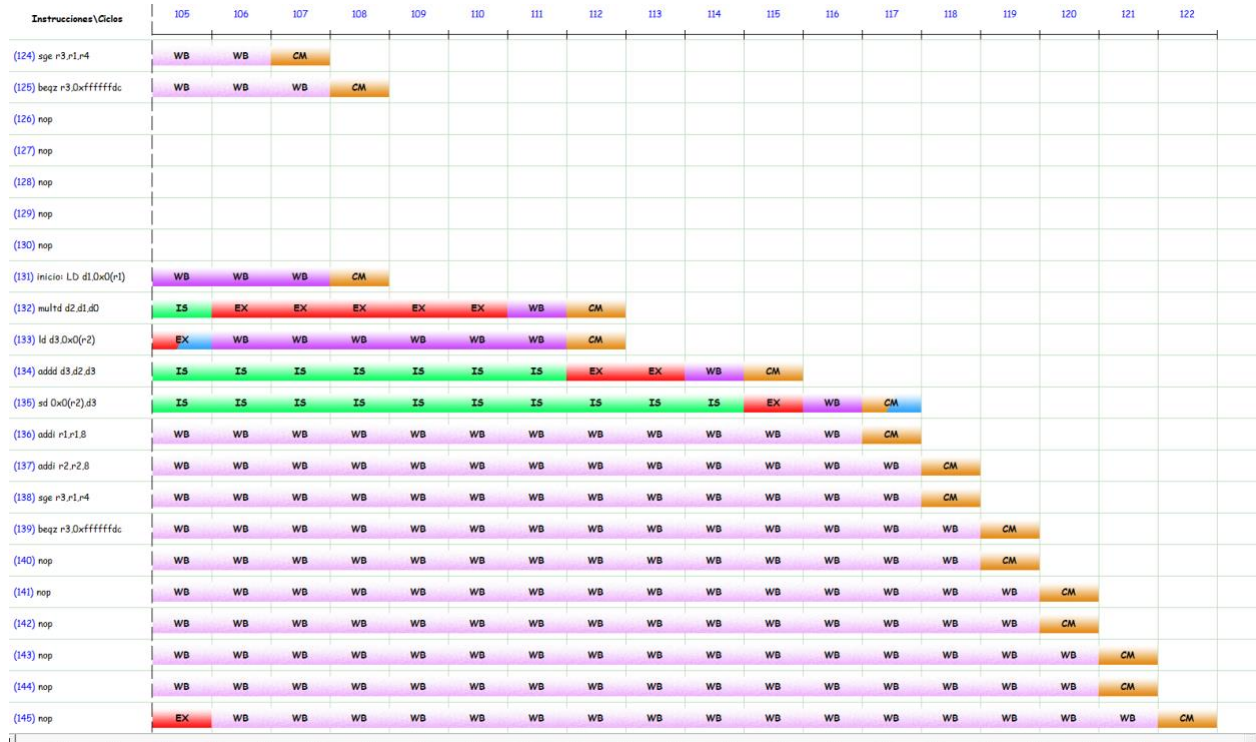
Ciclos ejecutados: 122 ciclos

Instrucciones ejecutadas: 145 instrucciones

$CPI = 122/145 = 0,841 \text{ ciclos/instrucción}$

Obtenido a partir del siguiente cronograma:

AlCo - Procesadores Superscalares



- la planificación dinámica, ¿mejora las prestaciones de la estática, para el programa de ejemplo? Justifica la respuesta.

En términos de prestaciones, podríamos decir que la planificación dinámica presenta un tiempo de ejecución menor, ya que ejecuta 122 ciclos, así como disminuye el CPI a 0,841. Por lo que diremos que aumentan las prestaciones.

- ¿por qué algunas instrucciones esperan tantos ciclos en la etapa de WB (write back)?

Algunas instrucciones esperan tantos ciclos en la etapa WB porque en la configuración dinámica tenemos una política de emisión que hace que siempre sea ordenada y se crean ciclos WB (o ROB) necesarios para que esto ocurra.

Instrucciones\Ciclos	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
(6) multd d2,d1,d0	EX	EX	EX	EX	EX	WB	CM								
(7) ld d3,0x0(r2)	WB	WB	WB	WB	WB	WB	CM								
(8) addd d3,d2,d3	IS	IS	IS	IS	IS	IS	EX	EX	WB	CM					
(9) sd 0x0(r2),d3	IS	IS	IS	IS	IS	IS	IS	IS	IS	EX	WB	CM			
(10) addi r1,r1,8	EX	WB	WB	WB	WB	WB	WB	WB	WB	WB	WB	CM			
(11) addi r2,r2,8	IS	EX	WB	WB	WB	WB	WB	WB	WB	WB	WB	WB	CM		
(12) sge r3,r1,r4	IS	IS	EX	WB	WB	WB	WB	WB	WB	WB	WB	WB	CM		
(13) beqz r3,0xfffffddc	IF	IS	IS	IS	EX	WB	WB	WB	WB	WB	WB	WB	WB	CM	
(14) nop	IF	IF	IS	EX	WB	Abort!									
(15) nop			IF	IS	EX	Abort!									
(16) nop			IF	IF	IS	Abort!									

- ¿qué significa la etapa CM con que finalizan todas las instrucciones? (como ayuda se te sugiere que abras la ventana del ROB en una ejecución paso a paso y compruebes qué sucede).

La etapa CM es cuando realmente se escribe, después de tomar acción el ROB a la hora de reordenar y mantener consistencia.

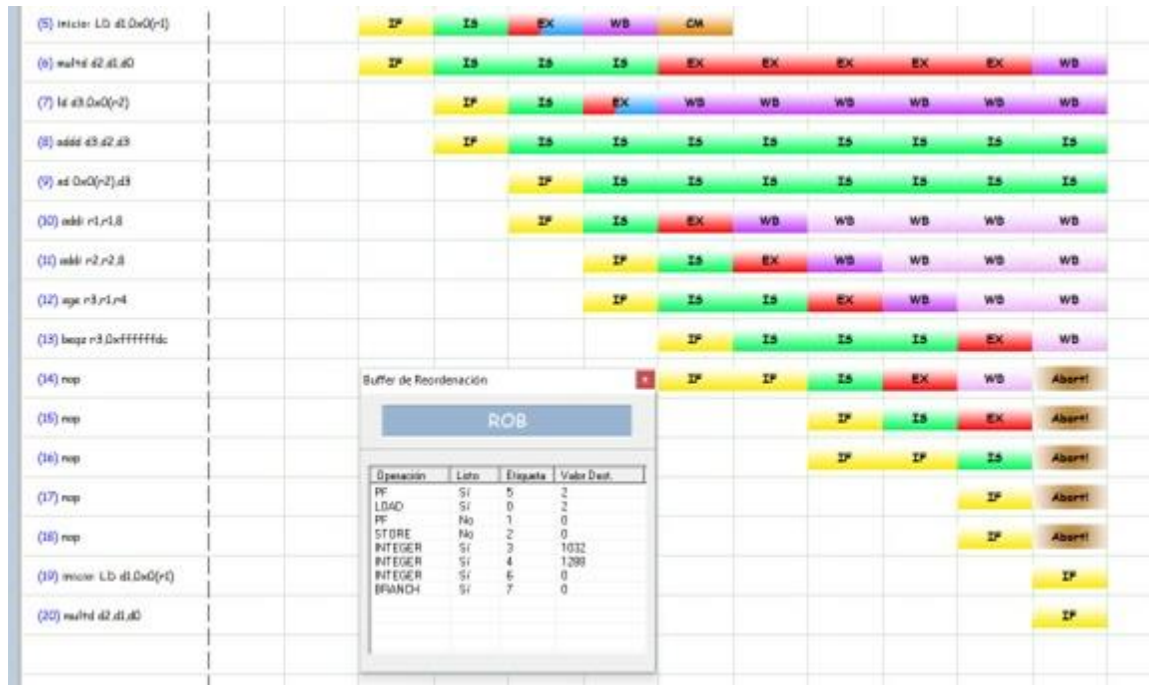
- ¿qué significa el doble color (marrón/azul) que la etapa CM muestra en algunos casos?

El color marrón/azul en la etapa CM indica que está escribiendo en memoria, estas son operaciones STOREs.

- ¿qué significa el doble color (rojo/azul) que la etapa EX muestra en algunos casos?

El color rojo/azul en la etapa EX indica que se está escribiendo en memoria, pero en este caso por ser operaciones LOAD y siguen cargando datos en memoria.

- durante la ejecución de los primeros ciclos del programa, ¿qué instrucción corresponde a la etiqueta 5 del ROB?, ¿por qué bloquea a las instrucciones posteriores en el ROB, ¿por qué desaparecen 3 entradas al pasar del ciclo 11 al ciclo 12?



Se trata de la instrucción “multd d2, d1, d0”, que al ser una multiplicación tarda 5 ciclos en ejecutarse, ciclos en los que al ser un ROB, en el que las instrucciones se retiran en el orden correspondiente al código, el resto de instrucciones deberá esperar aunque estén listas. Respecto a las 3 entradas que desaparecen al pasar del ciclo 11 al ciclo 12, esto ocurre porque estas 3 entradas corresponden a instrucciones nop, y en el momento en que la instrucción de salto I13 se ejecuta y pasa a estar lista, las nop mencionadas que habían entrado en el ROB abortan.

4.3 Comparativa

Una vez realizadas las simulaciones del apartado anterior contesta a las siguientes cuestiones:

- ¿qué diferencia hay entre un microprocesador segmentado (DLX), un microprocesador escalar (SDLX de grado 1) y un micro superescalar (SDLX con grado > 1)?

Entre las diferencias de los microprocesadores mencionados, cabe destacar que a mayor grado de superescalaridad, mayor rendimiento, siendo posible un $CPI < 1$ únicamente con grados de superescalaridad iguales o mayores a 2.

5. Conclusión

La realización de esta práctica ha supuesto una inmersión profunda en la arquitectura y el funcionamiento de los procesadores superescalares, una tecnología que se encuentra en el corazón del procesamiento de alto rendimiento en la computación moderna. A través del simulador SuperDLX, he podido trasladar los conceptos teóricos aprendidos en clase a un entorno práctico y experimental que simula de cerca el comportamiento de un procesador real.

La experiencia de utilizar el simulador me ha permitido entender con claridad cómo la planificación estática y dinámica influye en la ejecución de las instrucciones. He observado que la planificación dinámica, con su capacidad de adaptación en tiempo real y optimización de ejecución de instrucciones, supera significativamente a la planificación estática en términos de rendimiento. Esto se debe a su habilidad para manejar de manera eficiente las dependencias de datos y los conflictos de recursos, al tiempo que maximiza el paralelismo a nivel de instrucciones.

6. Bibliografía

- J. Ortega, M. Anguita, A. Prieto, *Arquitectura de Computadores*, Ed. Thomson
- P. de Miguel, *Fundamentos de Computadores*, Ed. Paraninfo
- C. Hamacher, Z. Vranesic, S.G. Zaky, *Computer Organization*
- José María Rodríguez Alcázar, "Simulador de la Versión Superscalar Dinámica del DLX", TFG, U. Sevilla, [online](#)
- Pau Micó Tormos, *Apuntes disponibles en Poliformat, UPV, 2023. Última consulta 17 de Diciembre de 2023.*

7. Anexo: ejemplos de SESCs

tipo	forma de emisión (ISS)	detección de riesgos	planificación	ejemplos
superscalar estático	dinámica	por hardware	estática	Embedded MIPS, ARM
superscalar especulativo	dinámica	por hardware	dinámica con especulación	Pentium IV, Core2, Power5 y Power7, Sparc VI, Sparc VII
VLIW	estática	por software	estática	TI C6x, Itanium