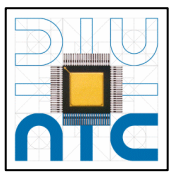
PRÁCTICAS DE PROCESADORES SUPERESCALARES con SuperDLX

Julio Ortega Lopera
Departamento de Arquitectura y Tecnología de Computadores



ÍNDICE

1. INTRODUCCIÓN

Estas prácticas permiten realizar un estudio experimental de un procesador superescalar que implementa el repertorio de instrucciones DLX. Para ello se utilizará el simulador SuperDLX, que utiliza emisión y ejecución fuera de orden, y con el que se puede analizar la influencia en las prestaciones de los distintos recursos y técnicas utilizadas en un procesador superescalar: el número de unidades funcionales y sus características; el número de instrucciones que se captan, decodifican, emiten, y retiran; el tamaño de las colas y ventanas de instrucciones; la técnica de predicción de saltos, el orden de ejecución de las instrucciones de acceso a memoria, etc.

El simulador SuperDLX ha sido desarrollado bajo la dirección del profesor Gao, en el laboratorio ACAPS (*Advanced Compiler, Architectures and Parallel Systems*) de la Universidad de McGill [MOU93]. Al implementar SuperDLX el repertorio DLX, es posible una comparación bastante directa entre las prestaciones que se obtienen en el procesador superescalar y las que se observan, para los mismos programas, en el procesador segmentado que se simula mediante WinDLX.

En esta introducción, se describirá en primer lugar el procesador superescalar. A continuación se presentan los comandos y la información que proporciona superDLX, para terminar con algún ejemplo de uso.

1.1 El procesador superescalar SuperDLX

En la Figura 1.1 se muestra un esquema del procesador superescalar simulado mediante SuperDLX (lo denominaremos procesador SuperDLX) indicando los elementos que definen su comportamiento desde el punto de vista de su comportamiento superescalar.

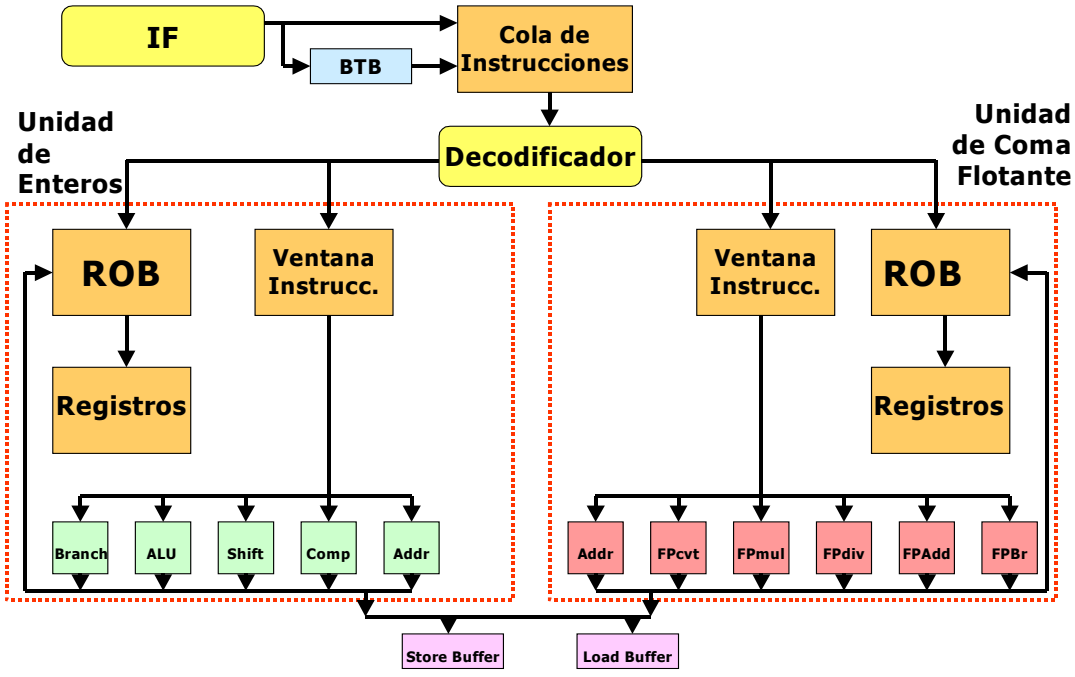


Figura 1.1 Elementos del Procesador SuperDLX

SuperDLX tiene una unidad funcional para enteros y otra para coma flotante. Cada una de esas unidades incluye una serie de circuitos aritmético-lógicos que implementan distintas operaciones, un banco de registros, un buffer de reorden y una ventana de instrucciones. Las instrucciones pasan a la unidad correspondiente desde el decodificador que, a su vez, las toma desde la cola donde han sido introducidas tras haber sido captadas. Cada unidad de datos, además, incluye los recursos necesarios para permitir la emisión y la ejecución de instrucciones fuera de orden o desordenada.

Así pues, las instrucciones se cargan de forma ordenada (según el orden del programa) en la ventana de instrucciones de la correspondiente unidad de datos. Desde la ventana, se emiten aquellas instrucciones que tienen sus operandos disponibles y para las que está libre el circuito aritmético-lógico que precisan. La emisión, por tanto, no tiene que ajustarse al orden en que las instrucciones llegaron a la ventana (emisión desordenada).

El buffer de reorden (ROB) almacena los resultados de las instrucciones una vez se han terminado de ejecutar, permite implementar el renombrado de registros, y resuelve los conflictos de almacenamiento ocasionados por la ejecución desordenada de instrucciones. Las instrucciones se retiran del buffer ordenadamente (según el orden del programa) una vez terminada su ejecución, y es precisamente cuando se retiran del ROB cuando se escriben los resultados en los registros de la arquitectura (ordenadamente).

Además de las Unidades de Datos, SuperDLX incluye un buffer de carga (load buffer), y otro de almacenamiento (store buffer) (Figura 1.1). Estos buffers son comunes a las dos unidades de datos y permiten que los cálculos de las direcciones de acceso a memoria para las cargas y/o almacenamientos se realicen de forma independiente a los accesos propiamente dichos. SuperDLX permite que las cargas adelanten a los almacenamientos (store bypass by loads) en el caso de no haya conflictos entre las direcciones de memoria.

En la Figura 1.1, BTB (Branch Target Buffer) hace referencia a un buffer que contiene direcciones de destino de salto correspondientes a instrucciones de salto previas. Este buffer permite acelerar el procesamiento de las instrucciones de salto de cara a implementar la técnica de predicción de salto utilizada para mejorar el comportamiento del procesador frente a las bifurcaciones.

SuperDLX es un procesador segmentado en el que se pueden distinguir cinco etapas:

- **Captación**: la instrucción entra en la cola de instrucciones.
- **Decodificación**: la instrucción pasa a la ventana de instrucciones de la unidad de datos correspondiente.
- **Emisión**: la instrucción deja la ventana de instrucciones cuando están disponibles sus operandos y la unidad funcional que necesita para ejecutarse.
- **Ejecución/Escritura**: se ejecuta la operación codificada en la instrucción y su resultado se escribe en la línea correspondiente del Buffer de Reorden (ROB).
- **Finalización** (Commited): La instrucción se retira del ROB y se escribe su resultado en el registro de la arquitectura correspondiente (se modifica el estado del procesador).

El número máximo de instrucciones que se pueden procesar en cada etapa depende de las características del procesador (puertos de lectura/escritura, buses, unidades funcionales, etc.). El simulador SuperDLX permite cambiar la configuración del procesador para captar,

decodificar,... distintos números máximos de instrucciones por ciclo y realizar los correspondientes análisis experimentales. A continuación se proporcionan algunos detalles adicionales de cada una de las etapas enumeradas.

Etapa de Captación. Las instrucciones se captan de la cache y se sitúan en la cola de instrucciones. Las instrucciones de salto condicional afectan al mecanismo de captación de instrucciones debido a que (1) la instrucción a captar a continuación de la de salto depende del resultado de su ejecución, y (2) las instrucciones que van a continuación de la instrucción de salto en el bloque de instrucciones que se capta de memoria pueden no tenerse que ejecutar (no son válidas). En los procesadores superescalares, la primera dificultad se resuelve mediante la predicción de salto: se predice cual va a ser el resultado de la ejecución de la instrucción de salto a partir de la información (almacenada en el BTB) acerca del comportamiento pasado de la instrucción de salto. La segunda cuestión se aborda mediante la posibilidad de invalidar instrucciones captadas a lo largo de su paso a través del cauce, junto con el uso del ROB que permite implementar la finalización ordenada.

El esquema de predicción de salto utilizado es un esquema de predicción dinámico con dos bits de historia, para representar el comportamiento pasado de la instrucción de salto. El simulador utiliza una tabla denominada *Branch Target Buffer* (BTB) que se indexa utilizando los bits de orden inferior de la dirección de la instrucción de salto (la posición en la tabla es igual a la dirección de la instrucción módulo el tamaño de dicha tabla). El tamaño del BTB se puede establecer en el fichero de configuración utilizado por el simulador.

Cada vez que se capta una instrucción de salto, se consulta el BTB para determinar la predicción de salto. Cada línea del BTB tiene información de la dirección de la intrucción de salto (*address*), la dirección de la instrucción destino del salto (*target*), y dos bits de historia que determinan la predicción que se realiza (*predictionState*). Si los bits de historia son (11) o (10) se predice que el salto se va a producir, y si son (00) o (01) se predice que no. Inicialmente, (cuando la instrucción de salto se carga por primera vez) se predice que el salto se va a producir.
El estado de predicción se actualiza una vez se ha ejecutado la instrucción de salto, teniendo en cuenta si finalmente se produjo el salto (taken) o no (not taken). En la Figura 1.2 se muestra el diagrama de estados según el cual funciona la predicción de salto.

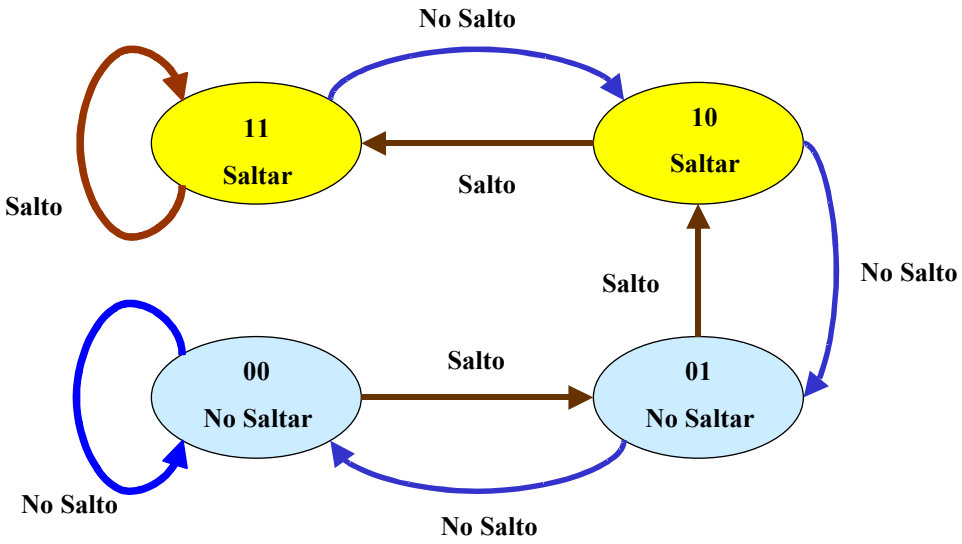


Figura 1.2. Diagrama de Estadps del Predictor de Saltos

Etapa de Decodificación. Las instrucciones se toman de la cola de instrucciones, se decodifican y se envían a la unidad de datos (enteros o coma flotante) apropiada. Cada instrucción decodificada se sitúa en el *ROB* de la unidad correspondiente. En cada línea del ROB hay espacio para almacenar el resultado de la instrucción cuando se produzca, y también para almacenar el identificador del registro de la arquitectura donde se escribirá el resultado de la instrucción cuando ésta se retire (en su etapa de *finalización*). El ROB está organizado como una cola FIFO circular, y sus líneas se van asignando a las instrucciones según el orden en que dichas instrucciones están en el programa.

Al mismo tiempo que las instrucciones pasan al ROB, también se almacenan en la ventana de instrucciones de la unidad de datos correspondiente. En esa ventana estarán hasta que se emitan. La emisión de una instrucción se produce cuando sus operandos, y la unidad funcional que se necesita para realizar la operación, están disponibles. Para obtener los operandos se mira primero en el ROB, y:

1. Si el operando se almacenará allí, pero no está todavía disponible (no se ha terminado de realizar el cálculo que lo producirá), se toma el número de la línea del ROB desde donde se tomará el operando cuando esté disponible. Siempre se tomará el valor, o el identificador de línea en su caso, de la línea del ROB más reciente (en el caso de que haya varias líneas del ROB que lo inluyan).
2. Si el operando no aparece en el ROB (en ninguna de las líneas del ROB que almacenan instrucciones se indica que se producirá el operando buscado), se toma del banco de registros correspondiente.

Aunque a las instrucciones se les asigne una línea del ROB y de la ventana de instrucciones de la unidad de datos en la que se procesarán, es posible que los operandos provengan de la otra unidad. En el caso de que se trate de una instrucción de almacenamiento o carga de memoria, también se almacenará en el buffer de almacenamiento o carga, respectivamente.

Etapa de Emisión. La lógica de emisión examina las instrucciones de la ventana y selecciona las que tienen los operandos disponibles y necesitan unidades funcionales que están libres. En el caso de que exista conflicto por el uso de una unidad funcional, o porque hay más instrucciones preparadas que el número máximo de instrucciones que se pueden emitir (debido a las limitaciones de buses y puertos de la unidad de datos), se elige la más antigua (se conoce por su posición en la ventana de instrucciones). Se implementa, por tanto, emisión desordenada con ventana deslizante (se van introduciendo instrucciones en la ventana, y se podrían emitir, conforme llegan).

La memoria se considera como otra unidad funcional más. Los almacenamientos y las cargas se emiten a la cache desde sus correspondientes buffers. No obstante, mientras que los almacenamientos (*stores*) deben modificar la memoria ordenadamente, las cargas (*loads*) pueden enviarse desordenadamente. Además, las cargas tienen más prioridad, y pueden adelantar (*bypass*) a los almacenamientos que no afecten a las mismas posiciones de memoria.

Etapa de Ejecución/Escritura. Se ejecutan las operaciones codificadas por las instrucciones emitidas. Los resultados obtenidos se almacenan en la línea correspondiente del ROB (donde se almacena la instrucción ejecutada). También se validan y se envían (forwarded) los resultados obtenidos a las líneas de las ventanas de instrucciones donde se necesitan los operandos

(apuntan a la línea del ROB donde se ha escrito el resultado). Se marcan las líneas del ROB que corresponden a operaciones terminadas.

En el caso de las instrucciones de salto, en la etapa de ejecución se determinará si se produce el salto o no. Si la predicción realizada es incorrecta, las instrucciones que están en el ROB se invalidan (*flush*) y se actualiza el BTB con la correspondiente información.

Etapa de Finalización. Los resultados se envían desde el ROB al banco de registros correspondiente. Las actualizaciones de los registros de la arquitectura se van realizado de forma ordenada, empezando desde la cabecera del ROB, hasta que se llega a una línea que almacena una instrucción cuyo resultado no se tienen todavía (o hasta que se han retirado el número máximo de líneas del ROB por ciclo que permiten los puertos y los buses del ROB). Las instrucciones que se han invalidado (*flush*) se retiran y sus resultados se descartan (no se actualizan los registros de la arquitectura).

En principio, no se precisa sincronizar los ROB de las unidades de enteros y coma flotante, dado que escriben en bancos de registros distintos y los almacenamientos (*stores*) en memoria se ordenan en su correspondiente buffer. Sin embargo, hay que establecer una sincronización para el caso de las instrucciones de salto, ya que ninguna instrucción de un ROB puede adelantar a una instrucción de salto de la otra unidad (retirarse antes y modificar los registros de su banco de registros), hasta que se haya completado la ejecución del salto. Por eso, a toda instrucción de salto se le asigna una línea en los ROBs de las dos unidades de datos.

1.1.1 Colas de almacenamiento temporal de SuperDLX

Para completar la descripción del procesador, en esta Sección se muestran las estructuras de la cola de instrucciones (Tabla 1.1), las ventanas (Tabla 1.2), y los buffers de reorden (Tabla 1.3) y de carga y almacenamiento (Tabla 1.4), de que dispone SuperDLX para el procesamiento superescalar las instrucciones. En cada tabla se indica el significado de los campos de cada una de las líneas de las distintas colas.

Tabla 1.1. Cola de Instrucciones Captadas

```
 ** Instruction Fetch Queue
Number of entries: 6
Maximum number of entries: 17
 ____________________________________________________________
|  #      icount  address    opcode    rs1  rs2   rd     extra |
|------------------------------------------------------------|
|  6           8  loop+0x14   SD         1    8    8        -8 |
|  5           7  loop+0x10   MULTD      6    2    8         0 |
|  4           6  loop+0xc    LD         1    6    6        -8 |
|  3           5  loop+0x8    SD         1    4    4         0 |
|  2           4  loop+0x4    MULTD      0    2    4         0 |
|  1           3  loop        LD         1    0    0         0 |
```

#	Número de posición (línea) en la cola
icount	Número de instrucción captada (cuenta de instrucciones dinámica)
address	Dirección de memoria donde se encuentra la instrucción
rs1	Registro fuente 1
rs2	Registro fuente 2
rd	registro destino
Extra	Dato inmediato u offset (en las instrucciones tipo I o J)

Tabla 1.2 Ventana de Instrucciones

```
** Floating Point Instruction Window
Number of entries: 2
Maximum number of entries: 20
 ____________________________________________________________________________
|  #  opcode  address     rb_entry  operand1 ok1 typ1  operand2 ok2 typ2 pred |
|----------------------------------------------------------------------------|
|  2  MULTD   loop+0x4           2         1   0  FPD          0   0  FPD   -  |
|  1  LD      loop               1         0   0  INT          0   1  IMM   -  |
+----------------------------------------------------------------------------
```

#	Número de posición (línea) en el buffer
opcode	Código de Operación
rb_entry	Línea del buffer de reorden en la que se encuentra la instrucción
operand1	Valor del operando o la línea del buffer de reorden desde donde se tomará cuando esté disponible
ok1	Si está a 0 indica que operand1 contiene la línea del buffer de rorden desde donde se toma el dato, si está a 1 indica que contiene el dato.
typ1	Indica el tipo de operando (INT, FPS, FPD, IMM). Permite saber si el operando se toma desde el buffer de reorden entero o flotante
pred	En el caso de las instrucciones de salto indica el tipo de predicción que se ha hecho (T o NT)

Tabla1.3 Buffer de Reorden

```
** Floating Point Reorder Buffer
Number of entries: 3
Maximum number of entries: 20
Head: 0 Tail: 2
 ____________________________________________________________________________
|  #  opcode     icount  reg#    unit        result    ok      ready  flsh |
|----------------------------------------------------------------------------|
|  0  LD             1    2    FP_DATA    0.000e+00     0          4     0 |
|  1  LD             3    0    FP_DATA        -         0          0     0 |
|  2  MULTD          4    4    FP_MULT        -         0          0     0 |
```

#	Número de posición (línea o registro) en el buffer
opcode	Código de operación
icount	Número de instrucción (dinámico)
reg#	Número de registro donde se debe almacenar el resultado de la operación
unit	Unidad Funcional donde se ejecuta la instrucción (desde donde se recibe el resultado)
result	Resultado de la instrucción si lo tiene (los STOREs no lo tienen)
ok	Indica si el resultado es válido (1) o no (0)
ready	Indica en qué ciclo de la ejecución estará disponible el resultado
flsh	Se pone a 1 cuando la instrucción sigue a un salto mal predicho (se descarta la línea al llegar a la cabecera del ROB)

Tabla 1.4 Buffers de Carga y Almacenamiento

```
 ** Load Buffer
Number of entries: 2
Maximum number of entries: 5
 _____________________________________________
|   #  opcode    icount   address  rel  flsh |
|---------------------------------------------|
|   2     LD          6        0     0     0 |
|   1     LD          3      10c0    0     0 |
+---------------------------------------------+
```

```
 ** Store Buffer
Number of entries: 1
Maximum number of entries: 5
 ____________________________________________________________________
|  #   opcode    icount   address  data to store  type ok rel  flsh |
|--------------------------------------------------------------------|
|   1     SD          5        0              2   FPD  0   0     0 |
```

#	Número de posición (línea o registro) en el buffer
opcode	Código de operación
icount	Número de instrucción (dinámico)
address	Dirección de memoria donde se escribe o de donde se lee
data	dato a escribir en memoria o línea del buffer de reorden desde donde se tomará
ok	Bit de validez del dato a almacenar
type	Indica el tipo de operando (INT, FPS, FPD, IMM) a almacenar (buffer Store)
rel	Indica si se permite que el dato se reescriba en memoria

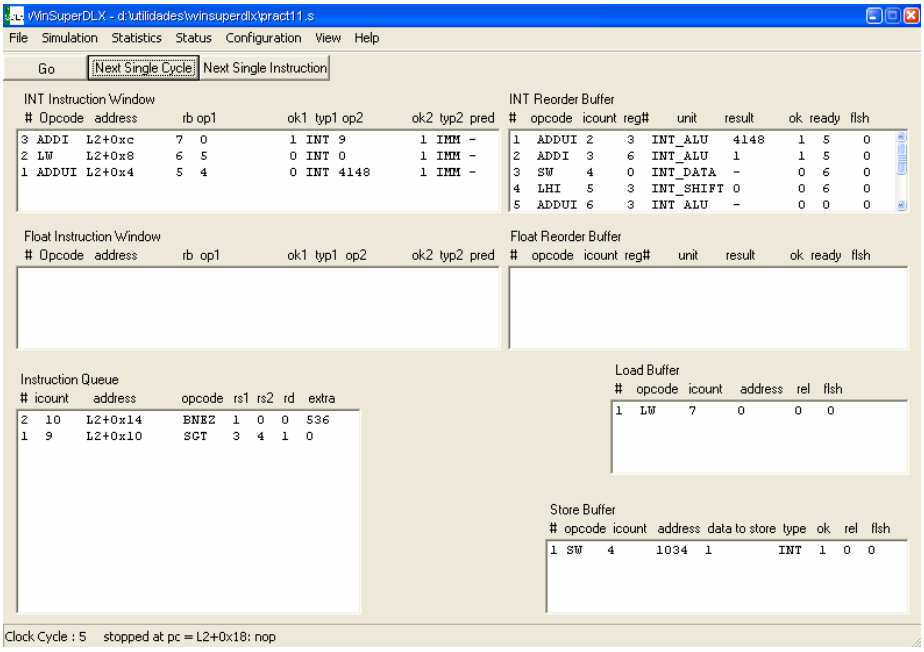
1.2 El simulador SuperDLX

En esta Sección se describen brevemente el simulador SuperDLX (Sección 1.2.1), y los datos que proporciona y se indica el significado de los datos proporcionados por el simulador (Sección 1.2.2)

1.2.1 Simulador SuperDLX

Desde 2012 existe una versión gráfica del simulador SuperDLX. Anteriormente, se empleaba una versión ejecutable por línea de comandos.

El programa debe copiarse a un directorio cuyo path no contenga espacios en blanco. Para iniciarlo se ejecuta Superdlx.exe.



Las opciones del menú son autodescriptivas, no obstante, a continuación se presenta una breve descripción de su funcionalidad.

File

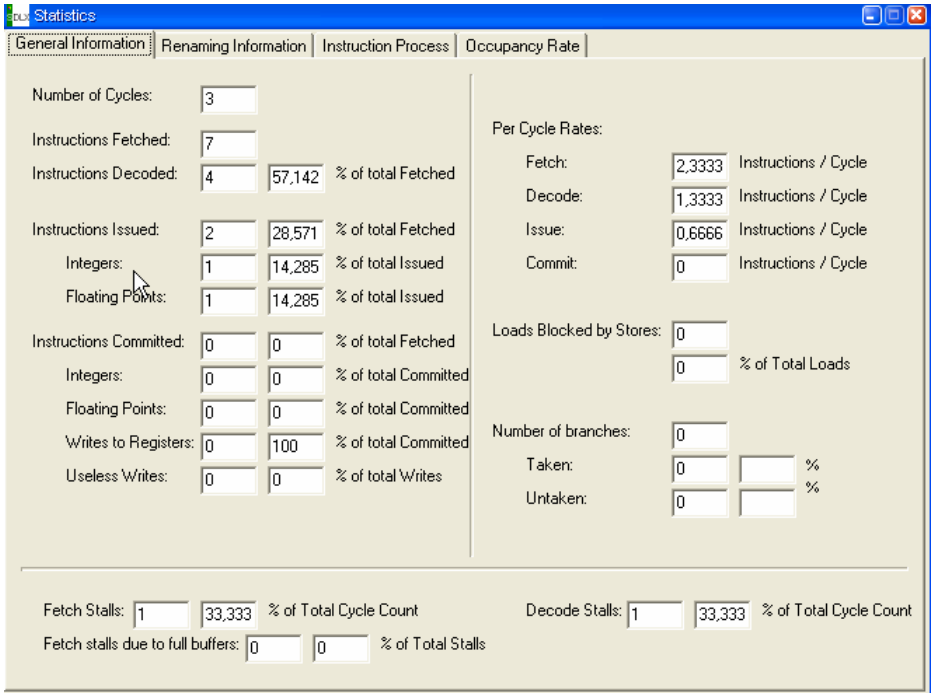
- Load Carga cada uno de los ficheros indicados. El texto(código) se suele cargar en 0x100 y los datos a partir de 0x1000.
- Reset Se utiliza para borrar la memoria y todos los elementos del simulador. Hay que volver a cargar un fichero para continuar

Simulation

- Go Ejecuta el programa hasta su finalización HALT
- Goto Se indica una dirección, sitúa el contador del programa en esa dirección y comienza a simular el programa cargado.
- Next Single Cicle Ejecuta un solo ciclo
- Next Single Instruction Ejecuta hasta que se haya finalizado la ejecución de una instrucción.

Statistics

Permite consultar las estadísticas descritas en el apartado 1.2.2 y salvarlas en un archivo.

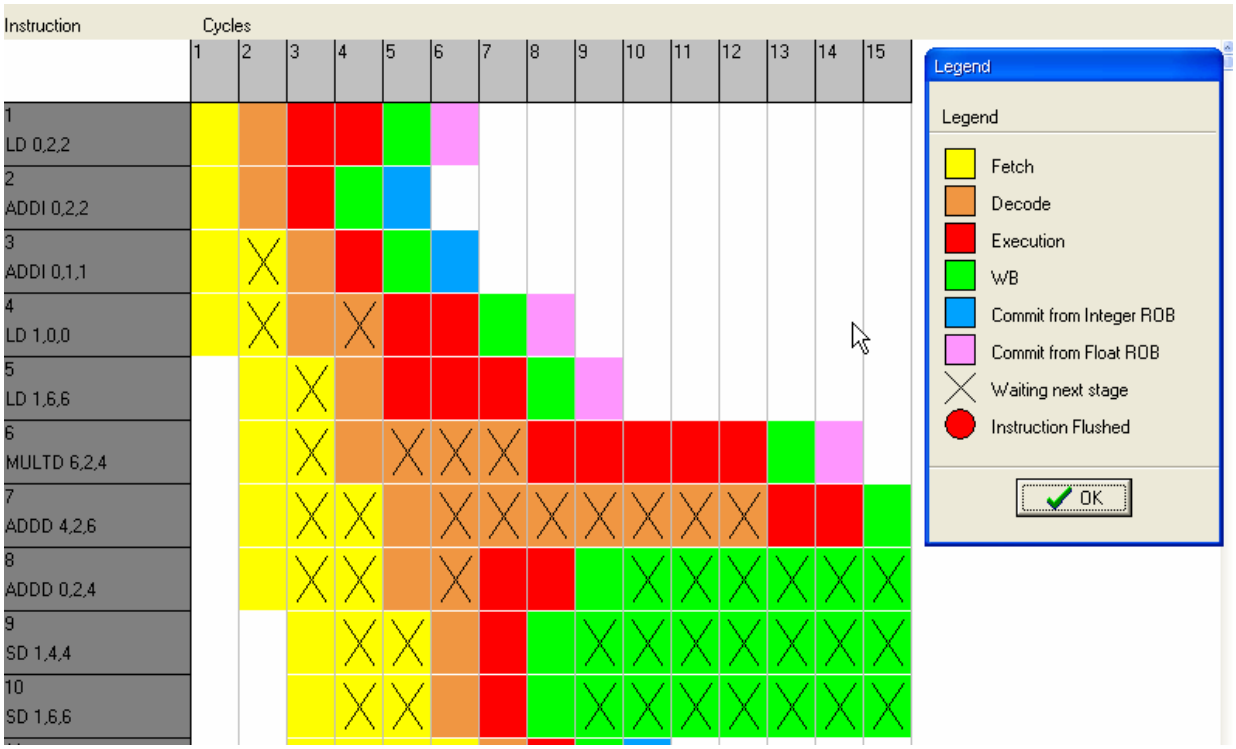


Configuration

Define las características del procesador SuperDLX. La configuración del procesador puede salvarse en un archivo y recuperarse para posteriores simulaciones.

View

- Source Code Presenta una nueva ventana con el código fuente cargado.
- Register File Permite consultar el estado del banco de registros.
- Clock Cicle Diagraman Presenta el diagrama de ciclos



1.2.2 Significado de los datos proporcionados por el simulador

Estos datos se obtienen cuando se consultan las opción **stats** del simulador. A continuación se indica el significado de cada uno de ellos a partir de un ejemplo. La primera línea indica el número total de ciclos que se ha simulado el programa. Si se ha simulado el programa hasta el final, nos permite determinar el tiempo que ha tardado en ejecutarse.

```
Number of cycles: 223   (se han consumido 223 ciclos)
```

Después se indica el número de instrucciones que se han captado (452 en el ejemplo), que se han decodificado (422 en el ejemplo), que se han emitido para su ejecución (417) distinguiendo entre instrucciones de enteros y de punto flotante, y que se han terminado de ejecutar (406) distinguiendo entre instrucciones enteras y de coma flotante, e indicando las escrituras que se han realizado. También se proporcionan los correspondientes porcentajes.

```
Instructions fetched            452
Instructions decoded            422      (93.36% of total fetched)

Instructions issued             417      (92.26% of total fetched)
-> integers                     322      (77.22% of total issued)
-> floating points               95      (22.78% of total issued)

Instructions committed          406      (89.82% of total fetched)
-> integers                     312      (76.85% of total committed)
-> floating points               94      (23.15% of total committed)
-> writes to registers          356      (87.68% of total committed)
-> useless writes               329      (92.42% of total writes)
```

A continuación se indican el número de lecturas de memoria que han tenido que esperar a que se realice una escritura debido a la dependencia entre ellas.

```
Loads Blocked by Stores:         10     (11.76% of total loads)
```

Finalmente se indican los números de instrucciones captadas, decodificadas, emitidas, y terminadas por unidad de ciclo. Para ello hay que dividir el número correspondiente a cada una de ellas que se ha ejecutado (dado más arriba) entre el número total de ciclos (223).

```
Per Cycle Rates
-> fetch    2.03 instructions/cycle
-> decode   1.89 instructions/cycle
-> issue    1.87 instructions/cycle
-> commit   1.82 instructions/cycle
```

La información acerca de los saltos indica el número de instrucciones de salto condicional ejecutadas distinguiendo entre las que han dado lugar a un salto ("taken") y las que no ("untaken"). Además se da información de cuántas predicciones de salto se han hecho correctamente y de los ciclos que se han perdido y las instrucciones que se han tenido que dejar de ejecutar ("flushed") debido a los saltos que no se han predicho correctamente. También se da información ("collisions in the BTB") de si ha habido que sacar datos de la BTB para nuevas instrucciones de salto que van apareciendo (en el ejemplo que se muestra no ha existido este problema).

```
Number of branches: 11, taken 1 (9.09%), untaken 10 (90.91%)
Correct predictions                9 (81.82% of total branches)
-> taken                           0 ( 0.00% of total taken branches)
-> not taken                       9 (90.00% of not taken branches)
Wrong predictions
-> lost cycles                     8 ( 3.59% of total cycles)
-> flushed instructions           14 ( 3.10% of total fetched)
Collisions in the BTB              0 ( 0.00% of total branches)
```

Esta parte del fichero indica si la unidad de captación y la de decodificación han tenido que estar detenidas durante cierto número de ciclos. En el caso de la unidad de captación, también se detallan los ciclos en los que ha estado detenida por estar los buffers llenos (en este ejemplo no ha existido este problema).

```
Fetch Stalls: 18 (8.07% of total cycle count)
Fetch stalls due to full buffers: 0 (0.00% of total stalls)
Decode Stalls: 11 (4.93% of total cycle count)
```

A partir de aquí, se da información del número de operandos que se han buscado, de la cantidad de renombramientos que ha habido, la distribución temporal de instrucciones emitidas y finalizadas, y los porcentajes de ocupación de los buffers (ventana de instrucciones y buffer de reordenamiento).

```
 **Operands Renaming (flow/anti-dependencies)

Renamed operands:  70.50%  of total operands
-> integers        65.25% of integer operands
-> floats          97.56% of floating point operands
```

En la tabla siguiente se indica durante cuantos ciclos se han renombrado 0,1,2, ó 3 operandos (obsérvese que el total de ciclos es de 223 en las tres columnas “total”).

	BOTH UNITS		INTEGER UNIT		FLOATING POINT UNIT	
NUM	total	%	total	%	total	%
0	21	9.42%	51	22.87%	178	79.82%
1	79	35.43%	84	37.67%	20	8.97%
2	92	41.26%	72	32.29%	15	6.73%
3	31	13.90%	16	7.17%	10	4.48%

("NUM": number of operands renamed ;"total": total number of clock cycles;
"%": percentage of clock cycles)

En la tabla que se da a continuación se indican los ciclos durante los cuales se ha estado buscando operandos (0, 1,2,3, ó 4 operandos en este caso). Obsérvese también que el total de ciclos es 223.

**Operands Searching Information

	BOTH UNITS		INTEGER UNIT		FLOATING POINT UNIT	
NUM	total	%	total	%	total	%
0	12	5.38%	22	9.87%	177	79.37%
1	0	0.00%	30	13.45%	20	8.97%
2	131	58.74%	122	54.71%	16	7.17%
3	77	34.53%	47	21.08%	10	4.48%
4	3	1.35%	2	0.90%	0	0.00%

("NUM": number of operands searched;"total": total number of clock cycles;
"%" : percentage of clock cycles)

En la tabla siguiente se indica la distribución de instrucciones emitidas. Es decir, se indica el número de ciclos en el que se emitieron 0, 1, 2, 3, y hasta 4 instrucciones. En las columnas dedicadas a enteros y punto flotante se recoge la información correspondiente a la emisión de instrucciones a la unidad de enteros y de punto flotante, respectivamente.

**Instruction Issue Distribution

	BOTH UNITS		INTEGER UNIT		FLOATING POINT UNIT	
NUM	total	%	total	%	total	%
0	11	4.93%	27	12.11%	149	66.82%
1	70	31.39%	112	50.22%	53	23.77%
2	100	44.84%	58	26.01%	21	9.42%
3	21	9.42%	10	4.48%	0	0.00%
4	21	9.42%	16	7.17%	0	0.00%

("NUM": number of instructions issued:"total": total number of clock cycles;
"%" : percentage of clock cycles

En la tabla siguiente se indica el número de ciclos que han ido necesitando las instrucciones para ser emitidas. En este caso, las instrucciones han necesitado desde un ciclo (como mínimo) hasta 7 ciclos. Obsérvese que el total de instrucciones es 417 en la columna “BOTH” (el total), 322 en la de enteros, y 95 en la de datos en coma flotante.

**Issue Delay Distribution

	BOTH UNITS		INTEGER UNIT		FLOATING POINT UNIT	
NUM	total	%	total	%	total	%
1	193	46.28%	178	42.69%	15	3.60%

```
|   2 |          127 | 30.46% |           97 | 23.26% |           30 |  7.19% |
|   3 |           26 |  6.24% |           11 |  2.64% |           15 |  3.60% |
|   4 |           26 |  6.24% |           11 |  2.64% |           15 |  3.60% |
|   5 |           20 |  4.80% |           10 |  2.40% |           10 |  2.40% |
|   6 |           20 |  4.80% |           15 |  3.60% |            5 |  1.20% |
|   7 |            5 |  1.20% |            0 |  0.00% |            5 |  1.20% |
!-----!--------------!--------!--------------!--------!--------------!--------!
("NUM": number of clock cycles;"total": total number of instructions;
 "%": percentage of issued instructions)
```

Instrucciones terminadas (“commited”). Se indica el número de ciclos (total) en el que se han retirado NUM instrucciones. Obsérvese que el producto de “NUM” por “total” es igual al número de instrucciones “commited” y que la suma de los totales es igual en las tres columnas (el número de ciclos en los que se retiran instrucciones). El porcentaje se refiere al porcentaje que supone cada cantidad de ciclos respecto al total de ciclos (223, en este caso). *En ese sentido, parece que hay un error en las indicaciones del significado de cada columna que proporciona el fichero*.

```
**Instruction Commit Distribution
 ____________________________________________________________________________
|      |      BOTH  UNITS      |      INTEGER UNIT     |  FLOATING POINT UNIT  |
| NUM |     total    |    %   |     total    |    %   |     total    |    %   |
|-----|--------------|--------|--------------|--------|--------------|--------|
|   0 |            0 |  0.00% |           13 |  5.83% |          114 | 51.12% |
|   1 |           66 | 29.60% |           95 | 42.60% |           32 | 14.35% |
|   2 |           42 | 18.83% |           29 | 13.00% |           31 | 13.90% |
|   3 |           30 | 13.45% |            1 |  0.45% |            0 |  0.00% |
|   4 |           34 | 15.25% |           39 | 17.49% |            0 |  0.00% |
|   6 |            5 |  2.24% |            0 |  0.00% |            0 |  0.00% |
!-----!--------------!--------!--------------!--------!--------------!--------!
("NUM": number of clock cycles;"total": total number of instructions;
 "%": percentage of clock cycles)
```

En las dos tablas que siguen, se indica la ocupación de los buffers en la unidad de enteros y de punto flotante respectivamente. Para cada caso se indica el número de ciclos (y el porcentaje correspondiente) durante el cual está ocupado un porcentaje determinado (desde 0% a 70%, en este caso) de los buffers de la ventana o del buffer de reorden.

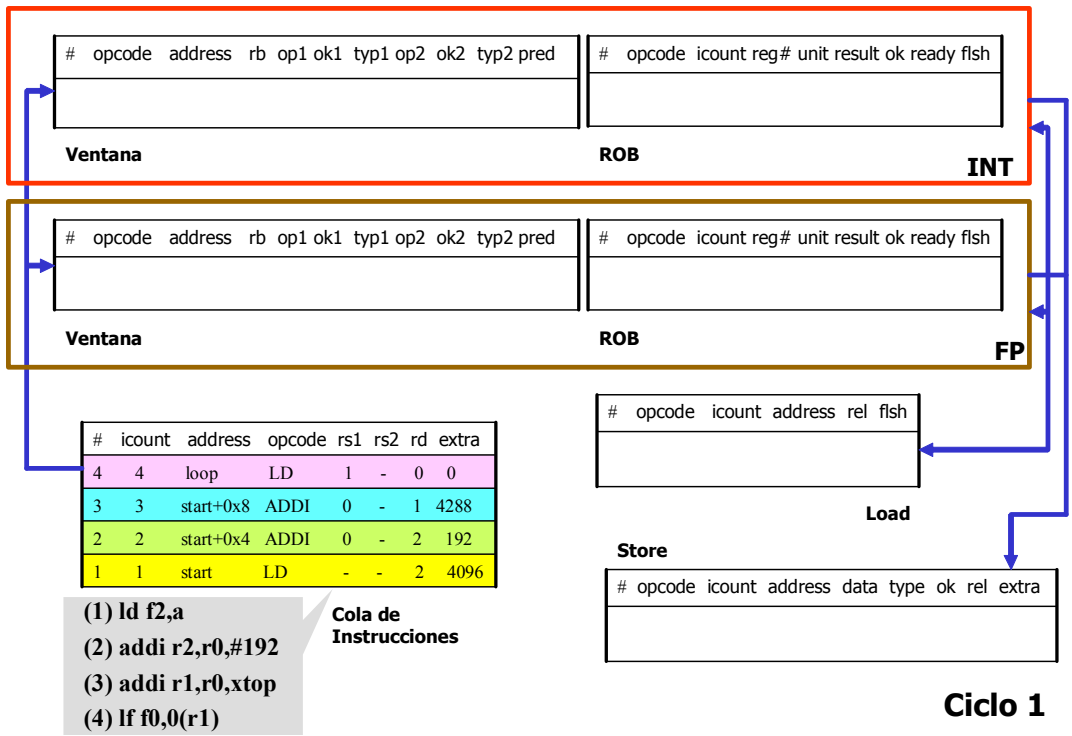
```
**Occupancy of the Integer Buffers
 ____________________________________________________________
|      |   INSTRUCTION WINDOW   |     REORDER BUFFER     |
| %OCC |   total cc   |    %cc  |    total cc  |   %cc   |
|------|--------------|---------|--------------|---------|
| + 0% |           71 |  31.84% |            7 |   3.14% |
| +10% |           64 |  28.70% |           33 |  14.80% |
| +20% |           83 |  37.22% |           29 |  13.00% |
| +30% |            5 |   2.24% |           42 |  18.83% |
| +40% |            0 |   0.00% |           23 |  10.31% |
| +50% |            0 |   0.00% |           35 |  15.70% |
| +60% |            0 |   0.00% |           39 |  17.49% |
| +70% |            0 |   0.00% |           15 |   6.73% |
!------!--------------!---------!--------------!---------!

**Occupancy of the Floating Point Buffers
 ____________________________________________________________
|      |   INSTRUCTION WINDOW   |     REORDER BUFFER     |
| %OCC |   total cc   |    %cc  |    total cc  |   %cc   |
|------|--------------|---------|--------------|---------|
| + 0% |          127 |  56.95% |           62 |  27.80% |
| +10% |           76 |  34.08% |           71 |  31.84% |
| +20% |           20 |   8.97% |           45 |  20.18% |
| +30% |            0 |   0.00% |           35 |  15.70% |
| +40% |            0 |   0.00% |           10 |   4.48% |
!------!--------------!---------!--------------!---------!
```

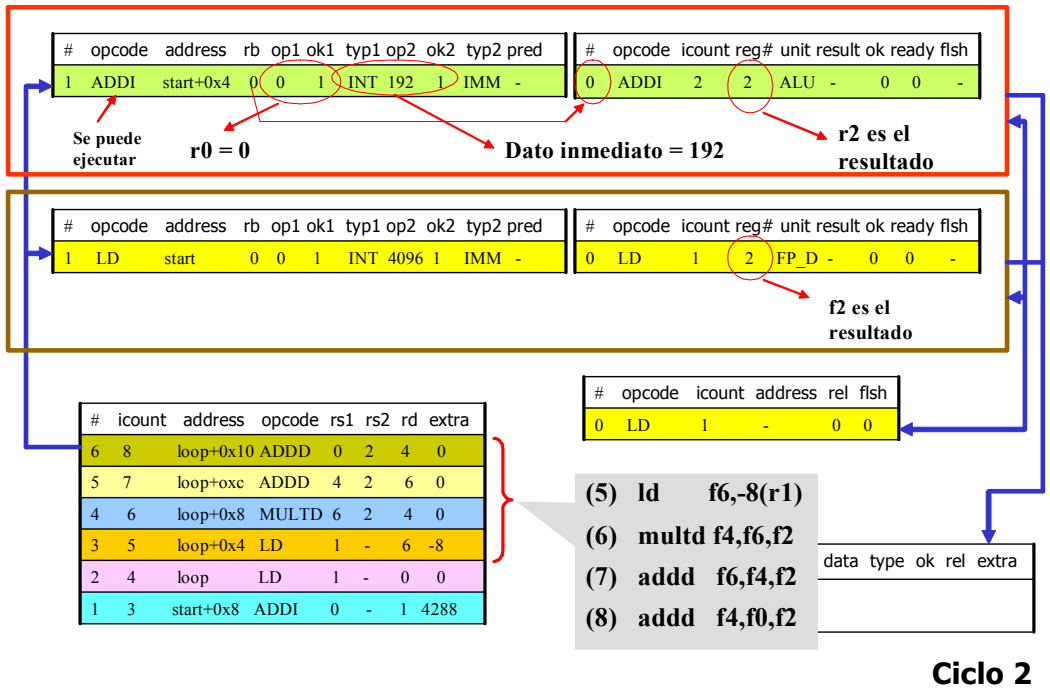
1.3 Ejemplo de ejecución de programa en SuperDLX

En esta sección se muestra la evolución de la cola de instrucciones, las ventanas, los buffers de reorden, y los buffer de carga y almacenamiento de memoria al ejecutar un programa. En concreto, se consideran los cinco primeros ciclos del programa:

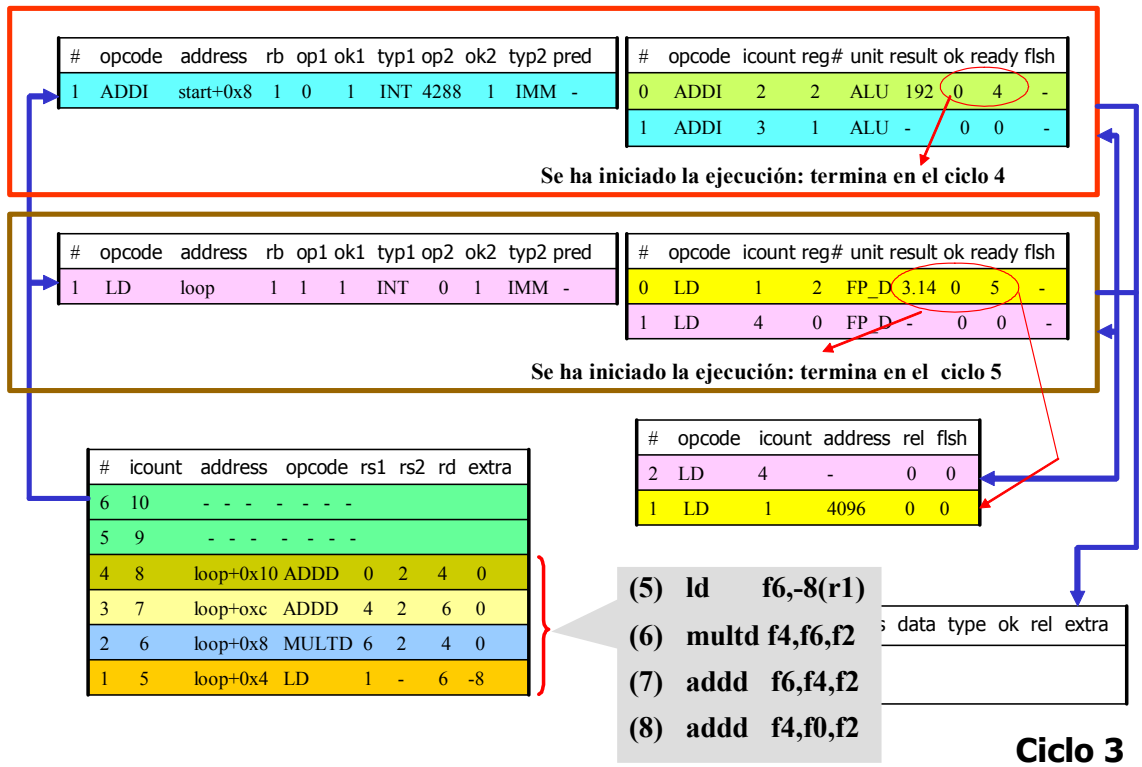
```
start:
(1)    ld         f2,a
(2)    addi     r2,r0,#192
(3)    add      r1,r0,xtop
loop:
(4)    ld         f0,0(r1)
(5)    ld         f6,-8(r1)
(6)    multd  f4,f6,f2
(7)    addd    f6,f4,f2
(8)    addd    f4,f0,f2
(9)    sd          0(r1),f4
(10)   sd        -8(r1),f6
(11)   sub      r2,r2,#16
(12)   bnez    r2,loop
(13)   nop
(14)   trap      #0
```

A continuación se muestran los contenidos de las distintas estructuras del procesador superescalar para los primeros cinco ciclos:

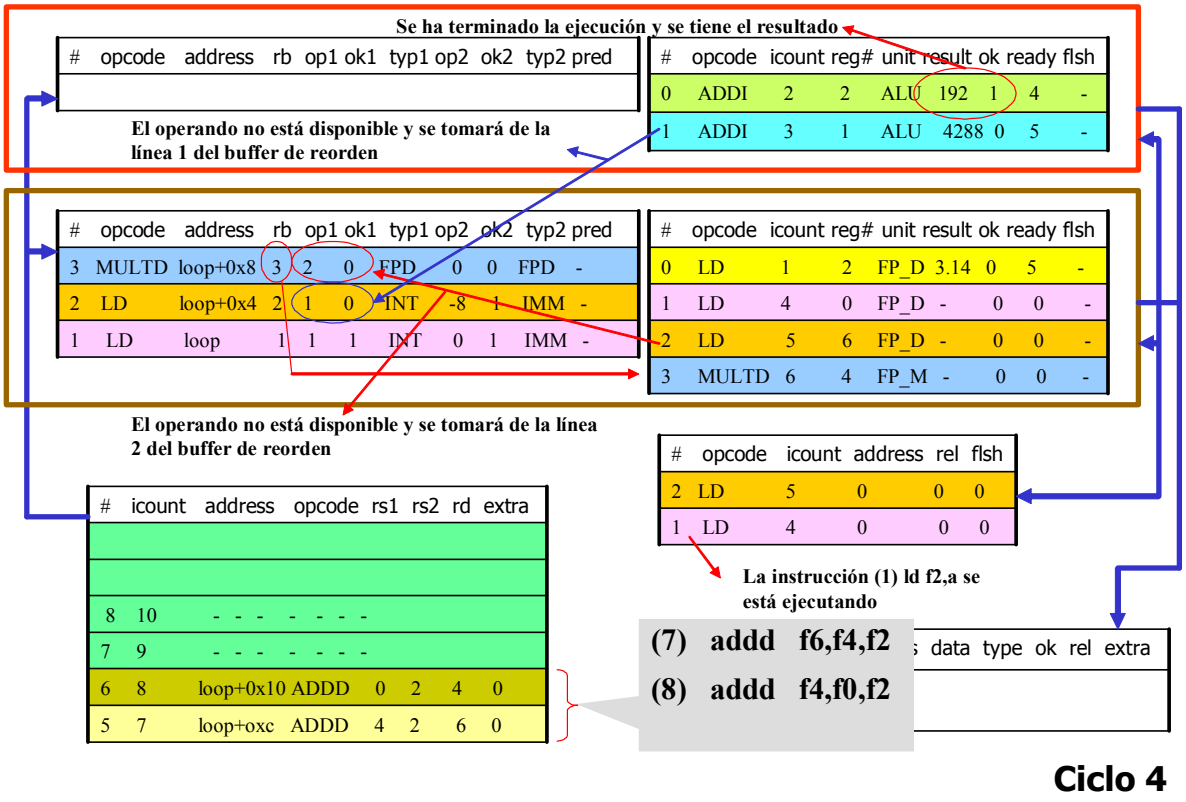




Ciclo 2

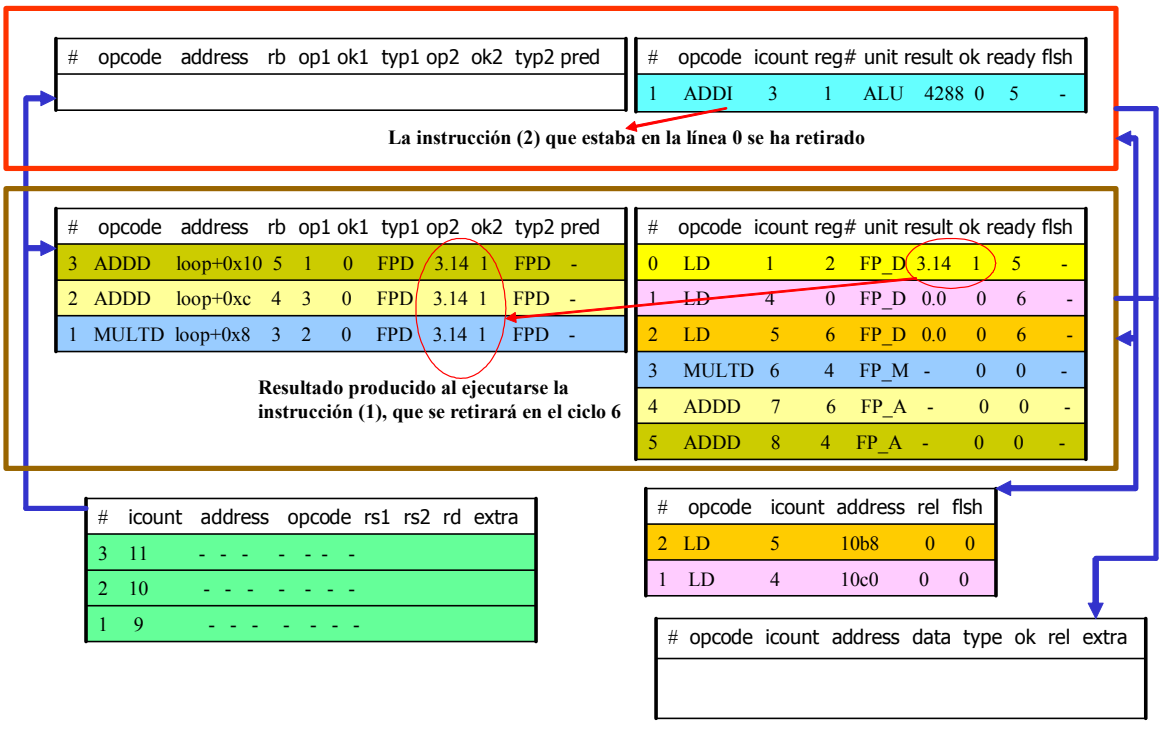


Ciclo 3



Ciclo 4

EJEMPLO DE EJECUCIÓN DE UN PROGRAMA EN EL PROCESADOR SUPERDLX (VI)



Ciclo 5

2. Ejercicos prácticos con SuperDLX

Para las **mejores versiones** de los programas **pr1_1.s** desarrollados en los **ejercicios 2 y** 5 de la **práctica 1** de procesadores segmentados con WinDLX, analice las características de su procesamiento en la arquitectura superescalar que implementa el simulador SuperDLX:

(1) Evalúe y justifique los cambios en los tiempos de ejecución cuando se modifican las características del procesador superescalar:
   - Número de instrucciones que captan, decodifican o finalizan ('commit') por ciclo de reloj.
   - Tamaño de los buffers de reorden, ventanas de instrucciones, y cola de instrucciones.
   - Predicción/No Predicción de Saltos y número de bits de predicción.

   Evidentemente, la prueba sistemática de todas las combinaciones posibles de los parámetros de configuración del simulador llevaría a un numero inviable de test. Por ello, el alumno debe justificar las pruebas que decide realizar (o las que descarta) en base a las dependencias entre los componentes funcionales del procesador. Por ejemplo, una tasa de decodificación superior a la de captación difícilmente aportará ventajas.

(2) Determine, a partir de los resultados obtenidos en el apartado anterior, la configuración de menor costo que permita obtener los mejores tiempos de ejecución en cada programa. Para ello, debe demostrar que el aumento de las capacidades de del procesador, no repercute en una reducción de ciclos o que la relación entre el aumento del coste y la reducción de tiempos no se justifica.

(3) Compare los resultados obtenidos para el procesador superescalar y los que se obtuvieron para el procesador escalar con el simulador winDLX. Para esta comparación utilice el mismo número de unidades funcionales con los mismos retardos en los dos procesadores.

(4) Para los programas *inic1.s*, *inic2.s*, *inic3.s*, *inic4.s*, e *inic5.s* obtenga los valores de las ganancias de velocidad del procesador superescalar con respecto al procesador sin segmentar de referencia ssrDLX y compárelas con las obtenidas para el procesador segmentado con WinDLX. ¿Que conclusiones se pueden extraer? (NOTA: Utilice una configuración de unidades funcionales en WinDLX similar a la del procesador superescalar. Considere tamaños suficientemente grandes para las colas, ventanas de instrucciones, y ROBs, de forma que no supongan una limitación en la velocidad de ejecución de instrucciones).

Documentación a presentar:

Resultados experimentales obtenidos, prestando especial atención a la claridad en la presentación (uso de tablas, representaciones gráficas, etc.) y a la justificación de los resultados en términos de las características propias de los procesadores superescalares.

Referencias

[MOU93] Moura, C.:"SuperDLX. A Generic Superscalar Simulator". ACAPS Technical Memo 64, 1993.