Arquitecturas Avanzadas
Práctica 1

Cola de Instrucciones y Banco de Registros

Rafael Ubal
Mónica Serrano
José Luis March
Julio Sahuquillo

Objetivos

- Utilizar y familiarizarse con una herramienta de simulación de procesadores superescalares.
- Analizar el impacto de las principales estructuras del procesador (*reorder buffer*, cola de instrucciones y banco de registros físicos) en las prestaciones.
- Comprender la importancia de la generación de *scripts* para automatizar el lanzamiento de simulaciones y la preparación de gráficas o tablas de resultados.

1 La herramienta de simulación Multi2Sim

Multi2Sim (`www.multi2sim.org`) es una herramienta de libre distribución y código abierto destinada a la simulación de procesadores superescalares, multithread y multicore. En las prácticas de esta asignatura relacionadas con la arquitectura del procesador, nos servirá como soporte para llevar a cabo diferentes experimentos. Comenzaremos por descargar un archivo comprimido (`practicas_aav_p1.tgz`) que contiene todos los ficheros necesarios para la práctica 1. Podemos descargar dicho fichero del PoliformaT de la asignatura, descomprimirlo y compilar el simulador:

```
tar -xzvf practicas_aav_p1.tar.gz
cd multi2sim-3.0.1
./configure
make
cp ./src/m2s ../
cd ..
```

Dentro del directorio se encuentran cuatro programas de prueba ya compilados para la arquitectura x86. Estos programas los usaremos como pequeños *benchmarks* para estas prácticas. Hay que tener en cuenta que la carga que suponen estos trabajos es muy pequeña, con el objetivo de mantener tiempos de simulación reducidos. A continuación se muestra una breve descripción de los programas:

- `test-args`: muestra por pantalla los argumentos pasados al programa por la línea de órdenes.
- `test-printf`: hace llamadas a la función `printf` con muchas combinaciones de parámetros.
- `test-math`: hace llamadas a funciones de la librería matemática.
- `test-sort`: crea un vector relativamente grande y lo ordena utilizando el algoritmo *quick sort*.

Los archivos ejecutables asociados a los *benchmarks* son aquellos con la extensión `i386`, mientras que el archivo ejecutable asociado al simulador es `m2s`. Para hacer una primera prueba, vamos a lanzar la simulación del programa `test-args` con la siguiente orden:

```
./m2s test-args.i386 hola que tal
```

La información mostrada en pantalla se puede clasificar en dos partes. Por un lado, la información propia del simulador se vuelca en la salida estándar de error (`stderr`). Por otro, el texto generado por el programa simulado se obtiene en la salida estándar (`stdout`). Podemos separar la salida del simulador de la salida de la aplicación simulada mediante alguna de las siguientes órdenes:

```
./m2s test-args.i386 2> /dev/null
./m2s test-args.i386 > salida.txt 2> salida_error.txt
./m2s test-args.i386 2>&1
```

Con la primera orden, descartamos por completo la salida del simulador. Con la segunda, enviamos la salida del programa al fichero `salida.txt`, mientras que la salida del simulador la volcamos en el fichero `salida_error.txt`. Con la tercera, tanto la salida del simulador como la salida del programa se vuelcan en la salida estándar.

La salida del simulador tiene dos secciones. La primera, volcada antes de comenzar la simulación, contiene un listado de los posibles argumentos de entrada junto con los valores que han tomado en la ejecución actual. La segunda, volcada tras la simulación, está formada por una lista de estadísticas, que representan los resultados de la simulación.

1.1 Extracción de estadísticas

La metodología típicamente empleada en investigación para comparar distintas arquitecturas de procesadores es la basada en el uso de simuladores. El objetivo de las simulaciones es evaluar las prestaciones de un determinado diseño a partir de valores numéricos. Estos valores son los obtenidos en la sección de estadísticas, entre las cuales es importante identificar las de especial interés en función del estudio.

Para el manejo de la salida del simulador, utilizaremos tres herramientas clave de Linux: `mktemp`, `grep` y `awk`. La primera de ellas gestiona la creación de ficheros temporales; la segunda filtra la entrada estándar en base a expresiones regulares, mientras que la tercera descompone cadenas en campos y permite su procesamiento. Para una descripción detallada, consultar las páginas del manual asociadas (orden *man*).

El tiempo de ejecución T_{ej} de un programa viene dado por la siguiente ecuación:

$$T_{ej} = I \times CPI \times t_{ciclo} = \frac{I \times t_{ciclo}}{IPC}$$

donde I es el número de instrucciones ejecutadas, CPI el número medio de ciclos por instrucción (IPC es su valor inverso) y t_{ciclo} el tiempo de ciclo. Por tanto, si para

un determinado diseño no varían los valores I y t_{ciclo}, el procesador que obtenga mejores prestaciones será aquel que alcance un mayor valor de IPC. Este valor viene dado por la estadística `sim.ipc` en el simulador, y se puede filtrar utilizando la siguiente secuencia de órdenes:

```
tempfile=$(mktemp)                              # Creamos un archivo temporal
./m2s test-args.i386 > /dev/null 2> $tempfile # Guardamos estadísticas en $tempfile
line=$(cat $tempfile | grep sim.ipc)            # Extraemos la línea con el IPC
ipc=$(echo $line | awk '{print $2}')            # Extraemos el valor numérico asociado
echo "El IPC es $ipc"                           # Mostramos IPC por pantalla
rm -f $tempfile                                 # Borramos el fichero temporal
```

De estas líneas, podemos destacar el uso de la sintaxis `variable=$(orden)`. Esta sintaxis es muy útil para ejecutar una determinada orden y almacenar su salida en una variable. Así ocurre, por ejemplo, en el caso de `mktemp`, que devuelve el nombre del fichero temporal creado. En el *script* anterior, se recoge este nombre en la variable `tempfile` para su posterior uso.

Para encapsular estas líneas en un *shell-script*, creamos un fichero, introducimos las órdenes anteriores, y lo almacenamos como `prueba.sh`. Es conveniente añadir esta línea al comienzo del *script*, para indicar cuál es el intérprete de órdenes a utilizar:

```
#!/bin/sh
```

Una vez creado el fichero, debemos asignarle permisos de ejecución, y podemos lanzarlo para probar su funcionamiento:

```
chmod +x prueba.sh
./prueba.sh
```

1.2 Configuración del procesador modelado

La herramienta Multi2Sim permite variar una amplia gama de parámetros que especifican las características del procesador modelado, como por ejemplo el tamaño de diferentes estructuras hardware. Estos parámetros pueden especificarse en la propia orden de ejecución del simulador o bien en un fichero de configuración. En esta práctica necesitamos variar un número reducido de parámetros, por lo que la primera opción es adecuada. Sin embargo, cuantos más parámetros varíen, más conveniente es utilizar ficheros de configuración.

Como primera prueba, vamos a observar el comportamiento del procesador al modificar el tamaño de una de sus principales estructuras hardware: el *reorder buffer* (ROB). Para ello, podemos introducir lo siguiente directamente en la línea de órdenes:

```
./m2s -rob_size  4 test-args.i386 2>&1 | grep sim.ipc
./m2s -rob_size  8 test-args.i386 2>&1 | grep sim.ipc
./m2s -rob_size 16 test-args.i386 2>&1 | grep sim.ipc
./m2s -rob_size 32 test-args.i386 2>&1 | grep sim.ipc
```

O de forma automática:

```
for size in 4 8 16 32
do
    ./m2s -rob_size $size test-args.i386 2>&1 | grep sim.ipc
done
```

El simulador permite intercalar cualquier número de opciones en la línea de órdenes. Las opciones de configuración del procesador modelado que no se especifiquen en la línea de órdenes toman un valor por defecto. Este valor automático puede consultarse en la lista de opciones inicial, volcada en la salida estándar de error al ejecutar el simulador (por ejemplo, el tamaño por defecto del ROB es 64). En la propia línea de órdenes, y después de las opciones, se debe especificar el nombre del archivo x86 ejecutable, opcionalmente seguido de sus argumentos.

1.3 Estudio de prestaciones

En esta práctica y en las posteriores, se pide al alumno llevar a cabo varios estudios de prestaciones atendiendo a diferentes componentes del procesador. Los estudios de prestaciones pueden dividirse en tres partes o fases diferenciadas:

1. **Fase de simulación**. Esta fase debe estar completamente automatizada y encapsulada en un *shell-script*. De forma automática, se lanzan simulaciones con diferentes combinaciones de los parámetros del procesador modelado. Al finalizar cada simulación, se extraen las estadísticas oportunas y se almacenan en un fichero, que servirá de entrada para la siguiente fase.

2. **Fase de representación**. La representación gráfica debe estar también automatizada y encapsulada. En este caso, se analiza el fichero de resultados y, mediante una herramienta gráfica, como *gnuplot*, se generan directamente las figuras.

3. **Fase de análisis**. Esta fase consiste en describir los resultados observando su representación gráfica en las figuras generadas. Se deben destacar aquellos resultados significativos y citar las conclusiones a las que conducen. Por ejemplo, si se evalúa el impacto del tamaño de una determinada estructura hardware en las prestaciones, deberíamos explicar los motivos por los el IPC se ve afectado, e introducir comentarios sobre el coste hardware que dichas variaciones conllevan.

La automatización y distinción de las fases de simulación y representación es muy ventajosa a la hora de llevar a cabo experimentos complejos. De esta forma, una modificación en el aspecto de las figuras no requeriría relanzar las simulaciones, en ocasiones muy costosas. A su vez, diferentes simulaciones que extraigan resultados con un formato similar pueden reutilizar una fase de representación desarrollada con anterioridad.

2 Caso de estudio: evaluación del tamaño del ROB

A continuación se muestra el desarrollo y la solución de un caso de estudio de ejemplo. Se recomienda una lectura detallada del mismo, que debe servir como referencia para los casos de estudio requeridos más adelante. Todos los ficheros asociados a este ejemplo se pueden localizar en el paquete descargado. El enunciado es el siguiente:

Se pide un estudio de prestaciones en el que se modifique el tamaño del *reorder buffer* (ROB) tomando todos los valores potencia de 2 situados entre 4 y 256 entradas, inclusive. Como cargas computacionales se deben utilizar los cuatro programas de ejemplo incluidos en el paquete descargado (`test-args`, `test-printf`, `test-math` y `test-sort`). Se debe obtener una sola gráfica con cuatro curvas, cada una de ellas asociada a un *benchmark* y representando todos los tamaños del ROB evaluados.

2.1 Fase de simulación

El *script* asociado a la fase de simulación en este estudio es `rob.simul`, que se encarga de lanzar las cuatro cargas de prueba disponibles con los tamaños de ROB solicitados. Este *script* debe ejecutarse en el mismo directorio donde se encuentre el simulador `m2s` y los programas de prueba con extensión `i386`. Su contenido es el siguiente:

```
#!/bin/sh

resfile="rob.res"
tempfile=$(mktemp)
rm -f $resfile
for workload in test-args.i386 test-printf.i386 test-math.i386 test-sort.i386
do
    for robsize in 4 8 16 32 64 128 256
    do
        ./m2s -rob_size $robsize $workload > /dev/null 2> $tempfile
        line=$(cat $tempfile | grep sim.ipc)
        ipc=$(echo $line | awk '{print $2}')
        echo $ipc >> $resfile
        echo "$workload: simulacion con ROB de tamanyo $robsize terminada"
    done
    echo >> $resfile
    echo
done
echo
```

La ejecución de este *script* genera el fichero de resultados `rob.res`. Como el proceso completo puede tardar varios minutos, es conveniente volcar por pantalla alguna notificación de que la generación de resultados avanza. Esto nos ayuda a detectar rápidamente aquellos casos en los que una invocación incorrecta del simulador ha provocado una ejecución infinita. En este ejemplo, se ha decidido imprimir una frase por cada simulación finalizada.

El fichero `rob.res` tiene cuatro bloques de valores separados por una línea en blanco, que a su vez están formados por siete valores numéricos. Cada bloque corresponde a una carga computacional o programa de prueba, mientras que cada valor dentro de un mismo bloque corresponde a un tamaño determinado del ROB. Este formato es idóneo para la interpretación posterior por `gnuplot`, que puede incluir diferentes curvas en una sola gráfica, basándose en la separación por bloques de los datos.

El contenido del fichero `rob.res` generado es el siguiente:

```
0.152
0.1608
0.1698
0.1804
0.1853
0.185
0.185

0.718
1.011
1.312
1.605
1.788
1.841
1.841

0.4614
0.569
0.6559
0.7299
0.7589
0.7653
0.7653

0.7759
1.127
1.504
1.852
2.018
2.012
2.012
```

2.2 Fase de representación

El *script* correspondiente a la fase de representación gráfica es `rob.plot`. Este *script* recoge el contenido de `rob.res` y llama a `gnuplot` para generar una gráfica en formato EPS, con el nombre `rob.png`. El código de `rob.plot` puede ser el siguiente:

```
#!/bin/sh

resfile="rob.res"
figfile="rob.png"
plotfile=$(mktemp)

cat << EOF > $plotfile
set term png
set key under
set xlabel 'ROB size'
set ylabel 'IPC'
set xrange [-0.5:6.5]
set xtics ( '4' 0, '8' 1, '16' 2, '32' 3, '64' 4, '128' 5, '256' 6)

plot '$resfile' every :::0::0 w linespoints t 'test-args',   \
     '$resfile' every :::1::1 w linespoints t 'test-printf', \
     '$resfile' every :::2::2 w linespoints t 'test-math',   \
     '$resfile' every :::3::3 w linespoints t 'test-sort'
EOF

gnuplot $plotfile > $figfile
rm -f $plotfile
```

Observando el código, vemos que `rob.plot` simplemente se encarga de preparar un fichero de entrada para `gnuplot`. Para ello, crea un archivo temporal mediante la orden `mktemp`, cuyo nombre almacena en la variable `plotfile`. La orden `cat << EOF > $plotfile` se utiliza para rellenar dicho fichero con todas las siguientes líneas hasta encontrar una línea que únicamente contenga la palabra `EOF`. La instrucción `plot` se encarga de leer los datos de un fichero de entrada (`rob.res` en el ejemplo), y representarlos. Su sintaxis es:

```
plot 'file' every {<point_incr>}
                  {:{<block_incr>}
                    {:{<start_point>}
                      {:{<start_block>}
                        {:{<end_point>}
                          {:<end_block>}}}}}
```

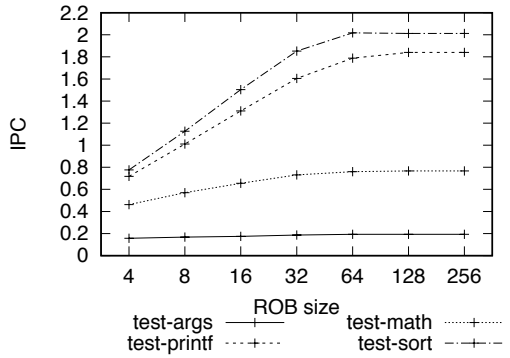
Los datos a representar se seleccionan según dos bucles: uno externo de bloques y uno interno de puntos dentro de un bloque. Un bloque es un conjunto de valores consecutivos separados por líneas en blanco. El primer bloque es `start_block` y el último `end_block` con incremencodetos de `block_incr`. El primer punto es `start_point` y el último `end_point` con incrementos de punto `point_incr`. Los datos y los bloques empiezan su numeración en '0'. Cualquier parámetro puede omitirse; por defecto, 1 es el incremento tanto para bloque como para puntos.

Ejemplo: `plot rob.res every :::3::3 # Selecciona todos los puntos del tercer bloque del fichero rob.res`

Se puede especificar el tipo de línea y la etiqueta de la misma con las opciones `with` o simplemente `w` y `title` o simplemente `t`. Así, en el ejemplo anterior se podría indicar que represente la línea con puntos y la etiquetase con el nombre `'text-args'`. El comando gnuplot quedaría de la forma:

```
plot rob.res every :::3::3 w linespoints t 'text-args'
```

Una vez preparado el fichero de entrada para `gnuplot`, llamamos a la propia herramienta gráfica, recogiendo su salida en el fichero `rob.png`, que es finalmente la representación gráfica perseguida. La figura generada es similar a ésta:



2.3 Fase de análisis

Las curvas asociadas a todos los *benchmarks* siguen una misma tendencia. Cuando aumenta el tamaño del ROB hasta un máximo de 64 entradas, las prestaciones van aumentando. Como el eje X tiene una escala exponencial (en cada posición se dobla el número de entradas del ROB), y los trazos que unen los puntos forman algo parecido a una recta, se puede decir que hay un crecimiento de las prestaciones logarítmico.

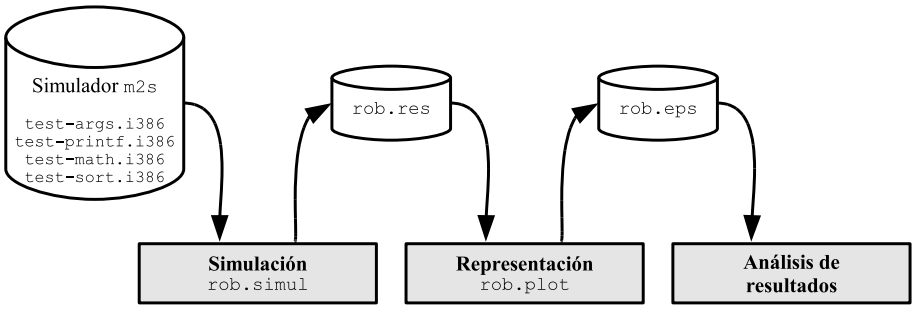
Sin embargo, una vez se aumenta el tamaño del ROB más allá de las 64 entradas, las prestaciones permanecen constantes. La razón es que el ROB deja de ser la estructura que limita la decodificación de instrucciones. A partir de ahí, son otras estructuras, como el banco de registros, las que se llenan antes de que pueda hacerlo el ROB, convirtiéndose en el nuevo cuello de botella para las prestaciones.

También se puede observar en la figura una gran diferencia de prestaciones entre los *benchmarks* para un tamaño de ROB determinado. Realizando pruebas individuales, se puede apreciar una relación entre las prestaciones obtenidas y el peso del *benchmark* en número de instrucciones. Concretamente, la estadística `sim.inst` nos devuelve 17K, 1.9M, 200K y 5.4M para `test-args`, `test-printf`, `test-math` y `test-sort`, respectivamente.

La razón es que una carga muy pequeña no ha tenido tiempo de estabilizarse trayendo a la memoria cache los datos y las instrucciones más frecuentes. Por ejemplo, `test-args` tiene una tasa de aciertos de 96% (`dl1-0.hitratio`) y 87% (`il1-0.hitratio`) en las caches L1 de datos e instrucciones, respectivamente, lo cual implica una porción considerable de accesos a L2 o memoria principal. En cambio, las tasas de acierto en `test-sort` rozan el 100% para ambas caches L1.

2.4 Resumen

A modo de resumen, la siguiente figura ilustra las fases de simulación, representación y análisis, junto con el trasiego de información entre ellas:



Trabajo a realizar

En esta práctica se plantea al alumno el desarrollo de un caso de estudio similar al que se da resuelto. A continuación se muestra el enunciado.

Ejercicio 1

En Multi2Sim la cola de instrucciones (IQ) tiene un tamaño por defecto de 40, mientras que el banco de registros físicos (RF) toma unos valores por defecto de 40 para el RF de enteros y de 40 para el RF de flotantes. Se pide analizar las prestaciones obtenidas y detectar las relaciones entre diferentes tamaños de IQ y del RF (mismo tamaño para enteros y flotantes). Para ello es conveniente representar en una misma gráfica el IPC alcanzando variando el tamaño del banco y de la IQ. Por ejemplo, se puede representar en el eje horizontal el tamaño del banco, y para cada valor obtener el IPC variando el tamaño de la IQ. Se aconseja empezar **realizando un script para la primera aplicación** `test-args` **y después modificarlo para cada una de las aplicaciones restantes (**`test-printf`**,** `test-math` **y** `test-sort`**).**

Los tamaños a explorar para ambas estructuras deben ser las potencias de 2 comprendidas entre 4 y 64, inclusive. En el caso del banco de registros, a estos tamaños se le debe sumar el valor 28, que es el número mínimo de registros físicos permitidos. La razón es que inicialmente, cada registro lógico está asociado (o renombrado) a un registro físico. Sólo los registros físicos libres en el estado inicial son aquéllos que pueden utilizarse para el renombrado de las instrucciones que tras decodificarse se insertan en el ROB. En el modelo de la arquitectura x86 implementado en Multi2Sim, el número mínimo de registros físicos impuesto garantiza que siempre pueda haber al menos una instrucción en vuelo.

Los parámetros del simulador para modificar el tamaño de la IQ y del RF de enteros y de flotantes son `-iq_size`, `-rf_int_size` y `-rf_fp_size`, respectivamente. Todos los demás parámetros del procesador deben permanecer con sus valores por defecto. Debe realizarse cuatro figuras, una figura para cada carga computacional (`test-args`, `test-printf`, `test-math` y `test-sort`).

3 Depurador del pipeline

El depurador del pipeline o *Pipeline debugger* es una herramienta incluida en el directorio `tools/m2s-pipeline` del paquete Multi2Sim. Proporciona diagramas de tiempo para simulaciones detalladas, en las que el estado del pipeline se puede examinar ciclo a ciclo con una representación de las microinstrucciones ejecutadas (muchas de las instrucciones definidas en el ISA x86 se pueden decodificar internamente en varias microinstrucciones, también denominadas uops). Esta herramienta se basa en la librería `ncurses`, que permite representaciones gráficas en terminales mediante caracteres de texto plano.

También se pueden comparar varias ejeccuciones usando más de un fichero con información de depuración como argumentos en la línea de comandos.

3.1 Obtener los diagramas de tiempo

Para mostrar el funcionamiento del depurador del pipeline, consideraremos un ejemplo. Ejecutamos el simulador con el benchmark `test-args`. Para obtener la información de simulación que se necesita para generar posteriormente el diagrama de tiempo, debe usarse la opción `-debug:pipeline`, seguida del nombre de un archivo donde se volcará esta información. Después se compila y se ejecuta el depurador del pipeline (ejecutable `m2s-pipeline`) usando este archivo como argumento. Para todo ello, deben utilizarse los siguientes comandos:

```
./m2s -debug:pipeline pipeline.txt test-args.i386
cd multi2sim-3.0.1/tools/m2s-pipeline
make
./m2s-pipeline ../../../pipeline.txt
```

La primera orden ejecuta el programa `test-args` en Multi2Sim, y guarda la información de depuración en el fichero `pipeline.txt`. Este fichero tiene un formato de texto plano, y se puede ver con cualquier editor. La información almacenada incluye eventos de las uops creadas, eliminadas (por fallo en la predicción de salto) o actualizadas, así como de su ubicación en el pipeline del procesador en cada ciclo. El último comando llama al depurador del pipeline, que interpreta los contenidos volcados por Multi2Sim en el fichero de depuración.

3.2 Elementos del depurador del pipeline

Los elementos que se muestran en pantalla al ejecutar el depurador son el número de secuencia de la uop y su descripción, los ciclos de ejecución, la etapa de ejecución en la que la que se encuentra la uop y la barra de estado. La primera uop de un grupo de uops pertenecientes a la misma macroinstrucción se representa en negrita. La Figura 1 muestra una captura de pantalla de la ejecución del depurador.

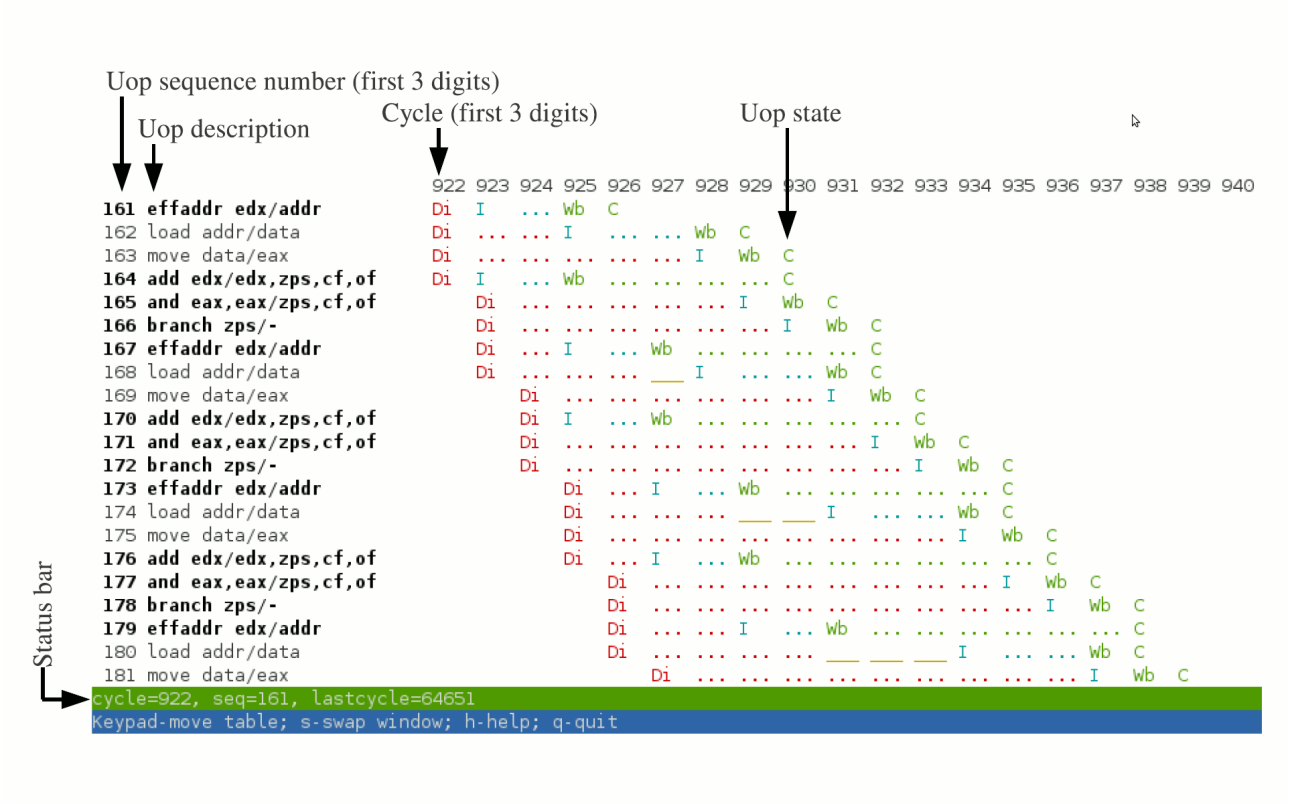


Figure 1: Captura de pantalla mostrando los distintos elementos del depurador.

3.3 Navegación sobre el diagrama de tiempo

Para movernos por el diagrama, podemos usar las teclas del cursor o bien las teclas `n`, `b`, `PgUp` y `PgDown`, que hacen avanzar o retroceder el diagrama varios ciclos, además de alinearlo correctamente para que la primera uop enviada se muestre en la esquina superior izquierda. También se puede ver el diagrama en un ciclo concreto tecleando el número de ciclo y pulsando `g`.

Cuestiones

A partir del diagrama obtenido en la sección 3.1 responde a las siguientes preguntas:

- ¿En qué ciclo finaliza el acceso a la cache la primera load?
- Respecto a la microinstrucción de salto con código de secuencia 55, ¿acierta el predictor de saltos?
- Indica el código de secuencia de la primera microinstrucción de salto (el nemotécnico utilizado es branch) en la que la predicción es incorrecta.
- ¿Por qué el diagrama no empieza en el ciclo 1, sino que tarda más de 100 ciclos en empezar?

Uso del Multi2sim con cargas de prueba científicas de uso generalizado

El simulador se ha utilizado en muchos artículos de investigación. Normalmente, se utilizan cargas representativas que utilizan gran cantidad de datos y cuyo tiempo de ejecución es mucho mayor que las pequeñas cargas utilizadas como muestra.

Entre las cargas más utilizadas se encuentran las SPEC-CPU2006 y SPLASH. A continuación se describe como lanzarlas en Multi2Sim.

Lanzamiento de las SPEC

Un ejemplo de lanzamiento para el benchmark calculix es:

```
./m2s -config procesador.txt -cacheconfig memoria.txt -ctxconfig ctxconfig.txt >
calculix.out 2> calculix.err
```

Donde los archivos `procesador.txt`, `memoria.txt` y `ctxconfig.txt`, contienen la configuración del procesador (tamaño del ROB, banco de registros, número de cores, unidades funcionales, etc), la configuración de la memoria (principal y cache) y la configuración del contexto, respectivamente. El archivo de configuración de contexto `ctxconfig.txt` es el que contiene el nombre del ejecutable y sus argumentos. A continuación se presentan ejemplos de estos ficheros.

Ejemplo de configuración del procesador

```
-cores 2
-decode_width 2  -dispatch_width 2  -issue_width 2  -commit_width 2
-fu:intadd  4 1 1 -fu:intsub  4 1 1 -fu:intmult  1 1 1 -fu:intdiv    1 1 1
-fu:effaddr 4 1 1 -fu:logical 4 1 1 -fu:fpsimple 2 1 1 -fu:fpadd     2 3 3
-fu:fpcomp  2 1 1 -fu:fpmult  1 5 5 -fu:fpdiv    1 7 7 -fu:fpcomplex 1 1 1
-bpred perfect
-lsq_size 128
-iq_size  128
-rob_size 128
-rf_int_size 160
-rf_fp_size  160
-max_inst 1500000
```

La variable `max_inst` limita el número de instrucciones a ejecutar (a 15M en el ejemplo). El objetivo es reducir el tiempo de ejecución del benchmark. Por supuesto, el número debe ser elevado para que los resultados sean representativos.

Los parámetros `nombre-etapa_width` indican el ancho de dicha etapa. Por ejemplo, `-decode_width 2` indica que se pueden decodificar 2 instrucciones por ciclo.

Para cada operador, etiquetado como fu (functional unit) se indican 3 parámetros. El primero es el número de operadores, el segundo es la latencia del operador y el tercero el *issue_rate* o tasa de repetición que nos indica cuantos ciclos deben pasar antes de poder lanzar otra operación a dicho operador. Por ejemplo, solo hay un multiplicador de coma flotante (`-fu:fpmult`) que tiene latencia 5 y no se encuentra segmentado.

Ejemplo de configuración del fichero de contexto

```
[ Context 0 ]
exe  = calculix_base.i386
args = -i hyperviscoplastic
cwd  = spec2006/454.calculix
```

Los ejemplos de configuración de memoria se detallarán en prácticas posteriores, así pues en esta práctica se omitirá el fichero para que lanzarlo en la configuración por defecto.

Ejercicio 2

Ejecuta el benchmark `calculix` durante 1.5M de ciclos con los parámetros mostrados arriba. Obtén los ciclos de ejecución, las instrucciones que han hecho commit, el IPC y los fallos de cache de L1.

Cambia los anchos del procesador para convertir el procesador superescalar en escalar. Obtén de nuevo los resultados.

Indica el speedup alcanzado por el procesador superescalar.

Realización y entrega de la memoria de la práctica

La entrega correspondiente a esta práctica consiste en una breve memoria del estudio de prestaciones realizado. Para este caso de estudio y los presentados en las siguientes prácticas se debe adjuntar en la propia memoria el contenido de todos los archivos implementados, incluyendo los archivos intermedios de resultados y las figuras finales.

Para esta práctica en concreto la memoria debe contener:

- *Script* para la obtención de resultados.
- *Script* para la generación de la gráfica. Unicamente debe presentarse la generación de una de las aplicaciones.
- Gráficas obtenidas. Una por aplicación.
- Análisis de las gráficas obtenidas.
- Cuestiones sobre el depurador del pipeline.
- Resultados del ejercicio 2 y valores de los anchos (`decode_width`, `dispatch_width`, `issue_width` y `commit_width`) con que se han lanzado las ejecuciones.

La entrega de las memorias de las prácticas relativas a la arquitectura del procesador se realizará individualmente por parte de cada alumno el primer día que empiece la siguiente práctica.