



PRÁCTICA 5:

ESTRUCTURA DE COMPUTADORES

María González Herrero,
Santiago Molpeceres Díaz
Pablo de Francisco de la Torre
Titulación: Ingeniería Informática
Asignatura = Estructura de Computadores



UNIVERSIDAD
NEBRIJA

INDICE

Análisis del código	3
Código con el formato -O3	5
Métricas de uso	6
Sin formato -O3	6
Data caché	6
Instruction caché	9
Formato -O3	10
Data cache	10
Instrucciones caché	11
Respuesta de observación	14

Análisis del código

```
1 .data
2 arg: .word 7
3
4 .text
5 #llamamos al procedimiento
6 lw a0,arg
7 jal ra,fact
8
9 # print resultado
10 li a7,1
11 ecall
12
```

Imagen 1: Procedimientos e imprimir

En esta imagen vamos a ver los primeros pasos del código.

Primero nos fijamos en el .data en el cual vamos a declarar una variable llamada “arg” que va a ser del tipo Word con valor 7.

Después declaramos el cuerpo del código con el “.text”.

Cargamos el valor de la variable arg en a0, antes de llamar a la función..

Con el comando “jal” llamamos a la función “fact” y guardamos en ra el valor en ra que devuelve la dirección del salto.

Finalmente imprimimos el resultado por pantalla.

Imagen 2: Salto de línea e imprimir

```
12
13 end:
14 li a7,10
15 ecall
```

A continuación llamamos a la etiqueta “end” para que se termine de ejecutar el código.

Después hacemos un salto de línea y lo imprimimos por pantalla.

Imagen 3: Código sin reducir

```

16 fact:                                     # @fact(int)
17     addi    sp, sp, -32
18     sw      ra, 28(sp)                    # 4-byte Folded Spill
19     sw      s0, 24(sp)                    # 4-byte Folded Spill
20     addi    s0, sp, 32
21     sw      a0, -12(s0)
22     addi    a0, zero, 1
23     sw      a0, -20(s0)
24     sw      a0, -16(s0)
25     j       .LBB0_1
26 .LBB0_1:                                   # =>This Inner Loop Header: Depth=1
27     lw      a1, -16(s0)
28     lw      a0, -12(s0)
29     blt     a0, a1, .LBB0_4
30     j       .LBB0_2
31 .LBB0_2:                                   # in Loop: Header=BB0_1 Depth=1
32     lw      a0, -20(s0)
33     lw      a1, -16(s0)
34     mul     a0, a0, a1
35     sw      a0, -20(s0)
36     j       .LBB0_3
37 .LBB0_3:                                   # in Loop: Header=BB0_1 Depth=1
38     lw      a0, -16(s0)
39     addi    a0, a0, 1
40     sw      a0, -16(s0)
41     j       .LBB0_1
42 .LBB0_4:
43     lw      a0, -20(s0)
44     lw      s0, 24(sp)                    # 4-byte Folded Reload
45     lw      ra, 28(sp)                    # 4-byte Folded Reload
46     addi    sp, sp, 32
47     ret

```

Al principio del procedimiento hacemos uso de la pila y vamos almacenando el valor de a0 en la dirección de memoria de s0 -12. A continuación sumamos zero +1 y lo actualizamos en a0, en la siguiente instrucción almacenamos el valor de a0 en la dirección de memoria de s0 -20, después en la siguiente instrucción almacena el valor de memoria de s0 -16. Después hacemos un salto a .

LBBO_1.

En dicho salto cargamos los valores de a1 y en a0 en las distintas direcciones de memoria como se ve en el código. A continuación comparamos a1 y a0 y si es menor salta a

.LBBO_2 sino salta a .LBBO_4.

En este salto cargamos los valores de a1 y en a0 en las distintas direcciones de

memoria como se ve en el código. Después multiplicamos a0 y a1 y lo guardamos en a0 en el que vamos a almacenar la dirección de memoria de s0 -20 y saltamos a .LBBO_3.

En .LBBO_3 cargamos a0. También le sumamos la dirección de memoria obtenida +1 y lo guardamos en a0. Después en a0 almacenamos el valor de memoria de s0 - 16 y saltamos a .LBBO_1.

En .LBBO_4 cargamos los valores de a0 en S0 y con "ra" desapilamos.

Código con el formato -O3

```
16 fact:
17     blez    a0, .LBB0_4
18     mv      a1, a0
19     addi    a0, zero, 1
20     addi    a2, zero, 1
21     beq     a1, a2, .LBB0_3
22 .LBB0_2:                                     # =>This Inner Loop Header: Depth=1
23     addi    a2, a2, 1
24     mul     a0, a0, a2
25     bne     a1, a2, .LBB0_2
26 .LBB0_3:
27     ret
28 .LBB0_4:
29     addi    a0, zero, 1
30     ret
```

Imagen 4: Código con el formato -O3

En fact; primero hacemos un “for” según nuestro código (blez), “ mv a1,a0 copia el contenido del registro a0 en el registro a1. Después sumamos zero + 1 y lo guardamos en a0. A continuación sumamos zero más 1 y lo guardamos en a2. Después saltamos si a1 y a2 si es menor va a .LBB0_3.

En .LBB0_2, sumamos a2 más 1 y lo guardamos en a2. Después multiplicamos a0 por a2 y lo guardamos en a0. Comparamos a1 y a2 y si son distintos volvemos a repetir las mismas instrucciones.

En .LBB0_3 realiza un retorno de subrutina.

En .LBB0_4 sumamos zero y más 1 y lo guardamos en a0 y realiza un retorno de subrutina

Métricas de uso

Sin formato -O3

Data caché

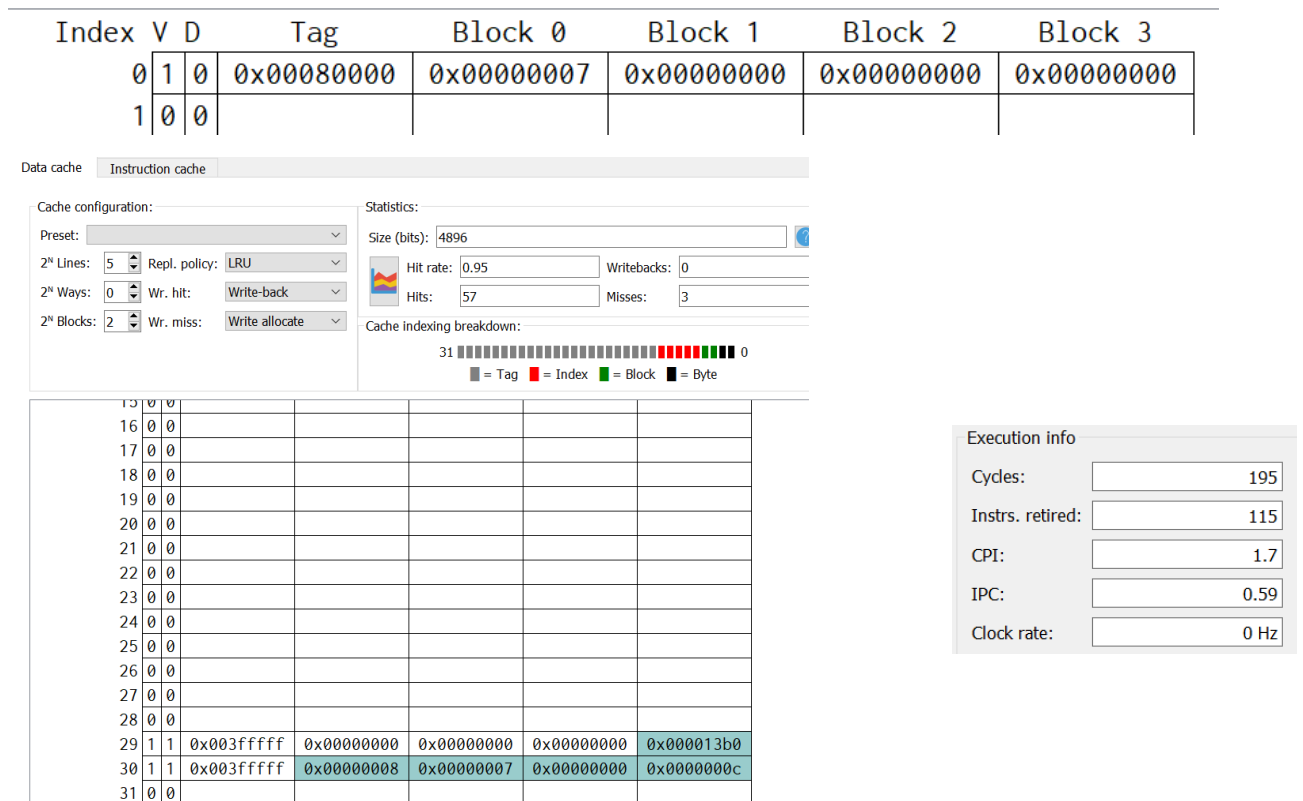


Imagen 5: Data caché

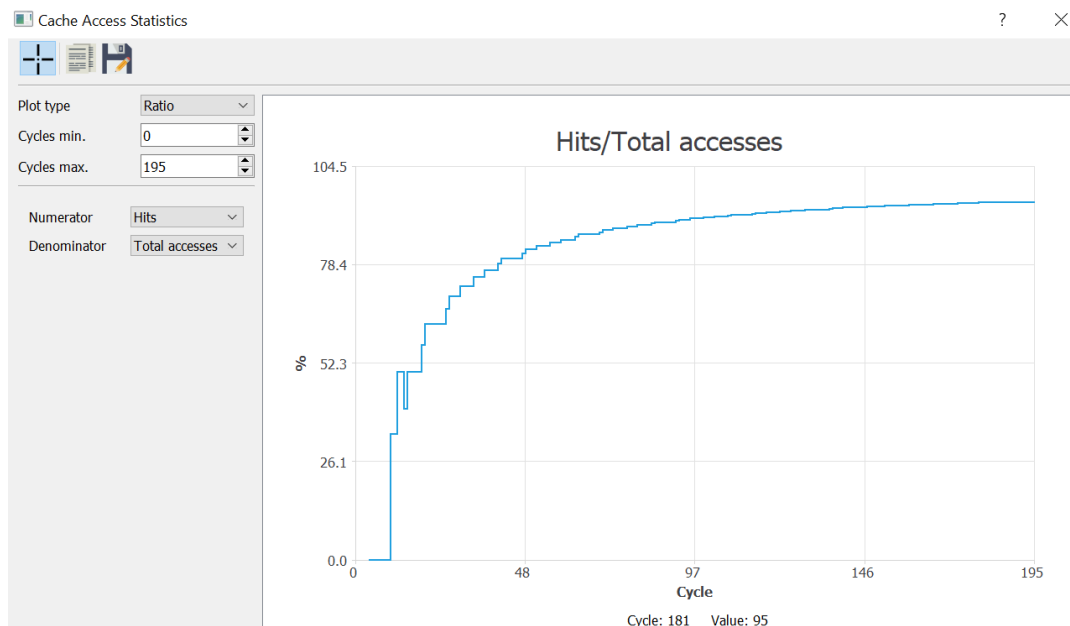


Imagen 6: Acceso a la caché

Como podemos ver en el estudio (*Imagen 5*) este programa se ejecuta en 195 ciclos de reloj, con un **CPI total de 1.7**, y un IPC de 0.59.

El número de instrucciones requeridas han sido 115, con 187 hits.

Es decir que hemos pedido el dato almacenado en el nivel i 187 veces. Como podemos ver la tasa de aciertos (hit ratio) que es la fracción de accesos encontrados en el nivel i, ha sido de un 95%.

Por lo que hemos tenido 3 miss (hemos tenido que solicitar el dato en un nivel i+1). Que es la representación de la gráfica (*Imagen 6*), por cada hit que ha sido correcto (no hemos tenido un miss) el hit rate ha subido. Cuando vemos que la gráfica ha bajado es por que hemos obtenido ese miss.

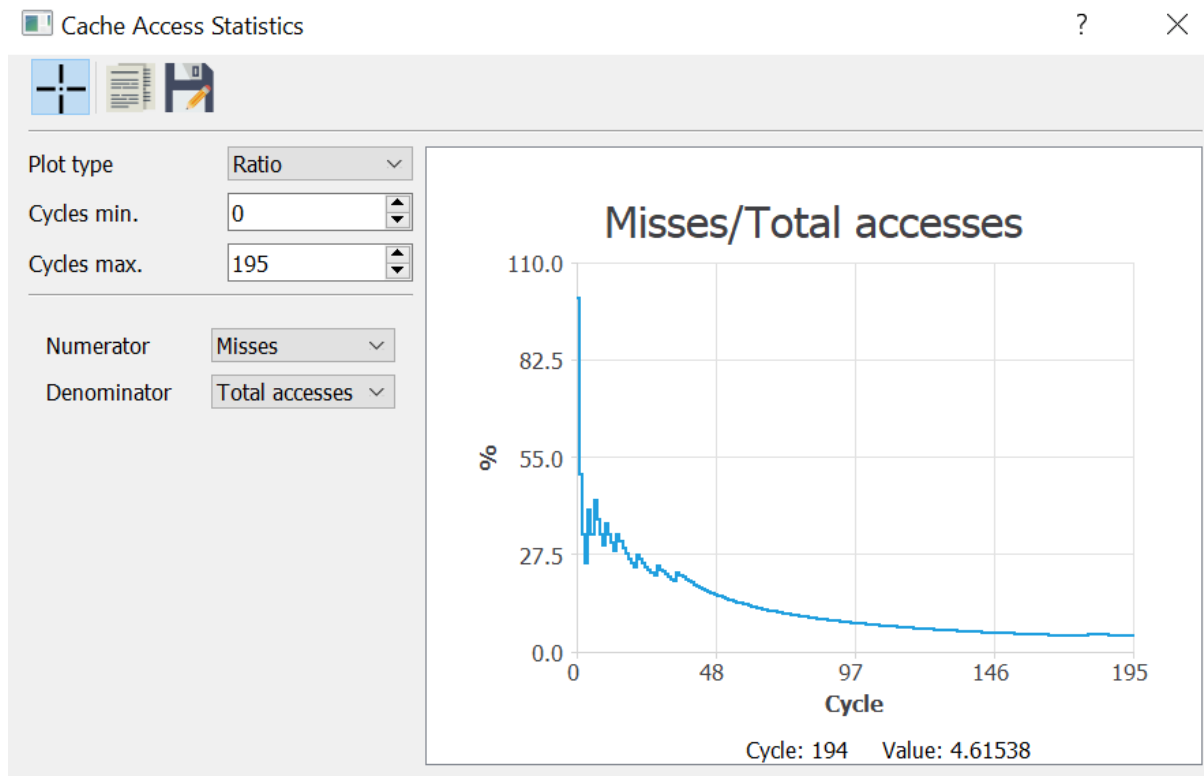


Imagen 7: Misses de un acceso a la caché sin formato -O3

Como podemos ver en el estudio (*Imagen 5*) este programa se ejecuta en 195 ciclos de reloj, con un **CPI total de 1.7**, y un IPC de 0.59.

El número de instrucciones requeridas han sido 115, con 9 hits.

Por lo que hemos tenido 3 miss (hemos tenido que solicitar el dato en un nivel i+1). Que es la representación de la gráfica (*Imagen 6*), por cada hit que ha sido correcto (no hemos tenido un miss) el hit rate ha subido. Cuando vemos que la gráfica ha bajado es por que hemos obtenido ese miss.

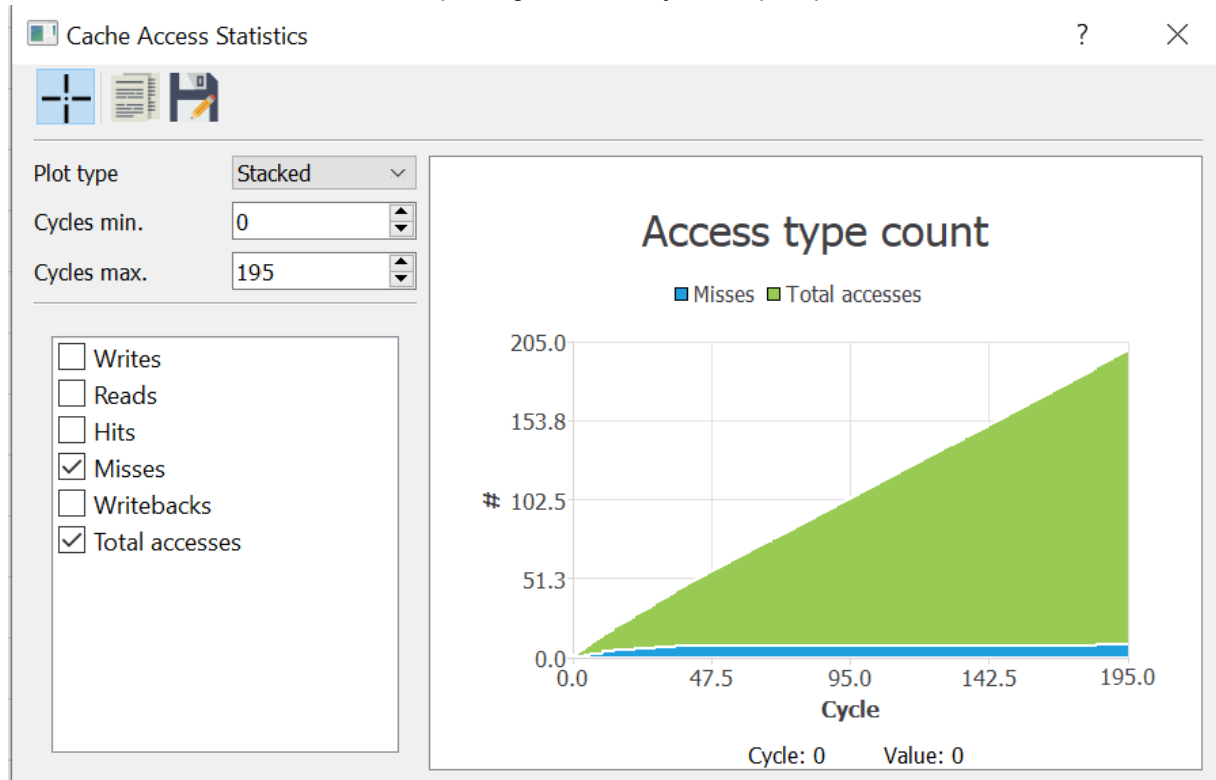


Imagen 8: Total de accesos

Tenemos 195 ciclos como máximo en la gráfica de los accesos de caché, con una pérdida mínima (misses) comparada con el total de accesos.

Instruction caché

En los 195 ciclos de reloj, se han ido almacenando las instrucciones que ha ido ejecutando el programa para poder volver a usarlas. Si ha visto que la instrucción no ha sido almacenada esta es guardada. ya sean las declaraciones , comparaciones , condiciones de salto etc.

Data cache

Instruction cache

Cache configuration:

Preset:


2nd Lines: 5 Repl. policy: LRU

2nd Ways: 0 Wr. hit: Write-back

2nd Blocks: 2 Wr. miss: Write allocate

Statistics:

Size (bits): 4896




Hit rate: 0.9541

Writebacks: 0

Hits: 187

Misses: 9

Cache indexing breakdown:

31 

= Tag

= Index

= Block

= Byte

Index	V	D	Tag	Block 0	Block 1	Block 2	Block 3
0	1	0	0x00000000	0x100000517	0x00052503	0x014000ef	0x00100893
1	1	0	0x00000000	0x00000073	0x00a00893	0x00000073	0xfe010113
2	1	0	0x00000000	0x00112e23	0x00812c23	0x02010413	0xfea42a23
3	1	0	0x00000000	0x00100513	0xfea42623	0xfea42823	0x0040006f
4	1	0	0x00000000	0xff042583	0xff442503	0x02b54663	0x0040006f
5	1	0	0x00000000	0xfec42503	0xff042583	0x02b50533	0xfea42623
6	1	0	0x00000000	0x0040006f	0xff042503	0x00150513	0xfea42823
7	1	0	0x00000000	0xfd1ff06f	0xfec42503	0x01812403	0x01c12083
8	1	0	0x00000000	0x02010113	0x00008067	0x00000000	0x00000000
9	0	0					
10	0	0					
11	0	0					
12	0	0					
13	0	0					
14	0	0					
15	0	0					

Imagen 9: Instrucciones de la caché

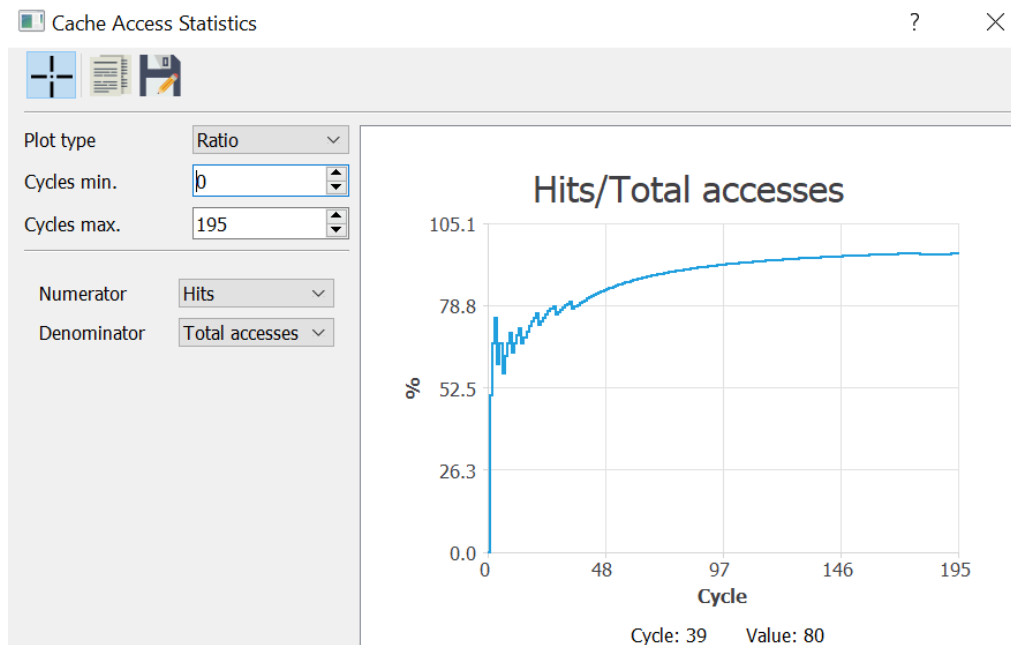


Imagen 10: Gráfico de acceso a la caché de instrucciones

Como podemos ver en el estudio (*Imagen 8*) este programa se ejecuta en 195 ciclos de reloj, con un **CPI total de 1.7**, y un IPC de 0.59.

El número de instrucciones requeridas han sido 115.con 57 hits.

Es decir que hemos pedido el dato almacenado en el nivel i 57 veces. Como podemos ver la tasa de aciertos (hit ratio) que es la fracción de accesos encontrados en el nivel i , ha sido de un 95%.

Por lo que hemos tenido 3 miss (hemos tenido que solicitar el dato en un nivel i+1). Que es la representación de la gráfica (*Imagen 8*), por cada hit que ha sido correcto (no hemos tenido un miss) el hit rate ha subido. Cuando vemos que la gráfica ha bajado es por que hemos obtenido ese miss.

Formato -O3

Data cache

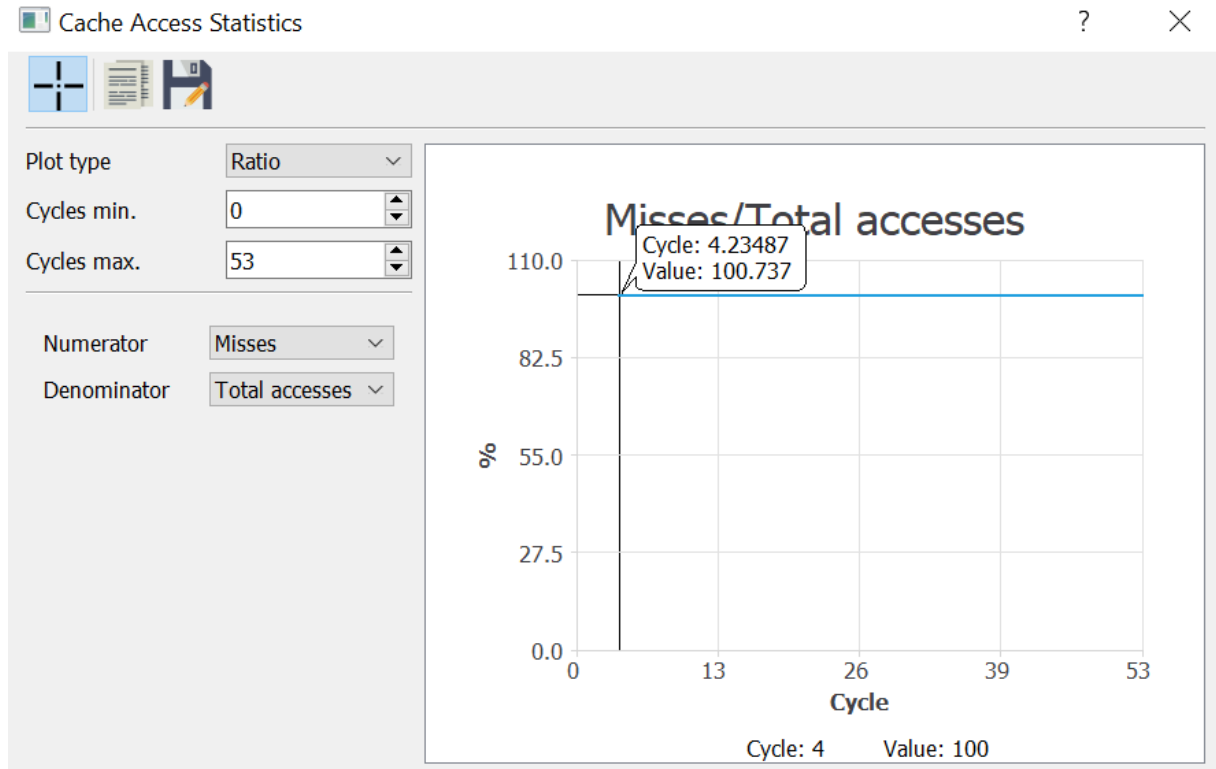


Imagen 11: Gráficos de misses de la memoria caché

Como se puede ver en la imagen 12, tenemos un máximo de 53 ciclos, que hace que haya un total de 100 accesos aproximadamente de la caché, haciendo que se cree una línea uniforme a lo largo de 53 ciclos.

Imagen 13:

En los 53 ciclos de reloj, se han ido almacenando las instrucciones que ha ido ejecutando el programa para poder volver a usarlas. Si ha visto que la instrucción no ha sido almacenada esta es guardada. ya sean las declaraciones , comparaciones , condiciones de salto etc.

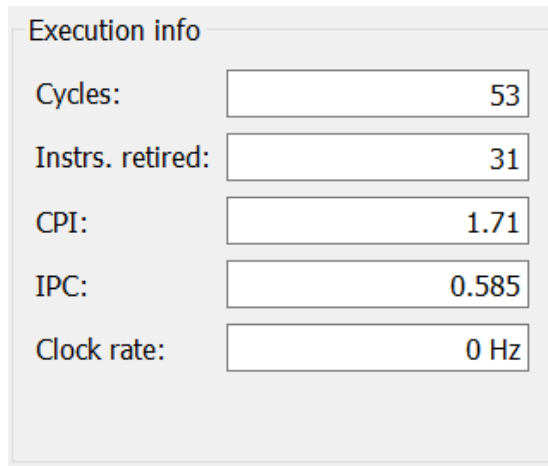


Imagen 14

Como podemos ver en el estudio (Imagen 14) este programa se ejecuta en 53 ciclos de reloj, con un **CPI total de 1.7**, y un IPC de 0.585

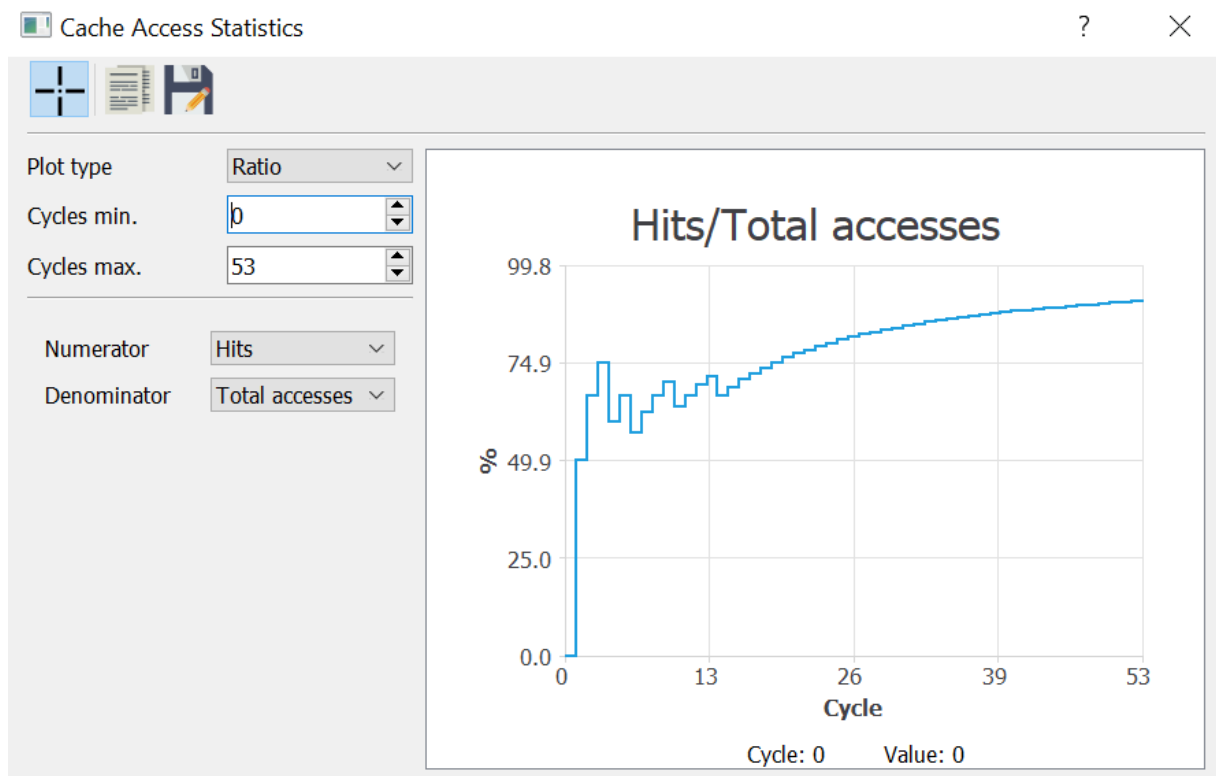


Imagen 13: Gráfico de acceso a la caché de instrucciones

Como se puede observar en la imagen 13, tenemos un máximo de 53 ciclos, que hace que haya un total de 100 accesos aproximadamente de la caché, haciendo que se cree una línea uniforme a lo largo de 53 ciclos, en total hay 49 hits.

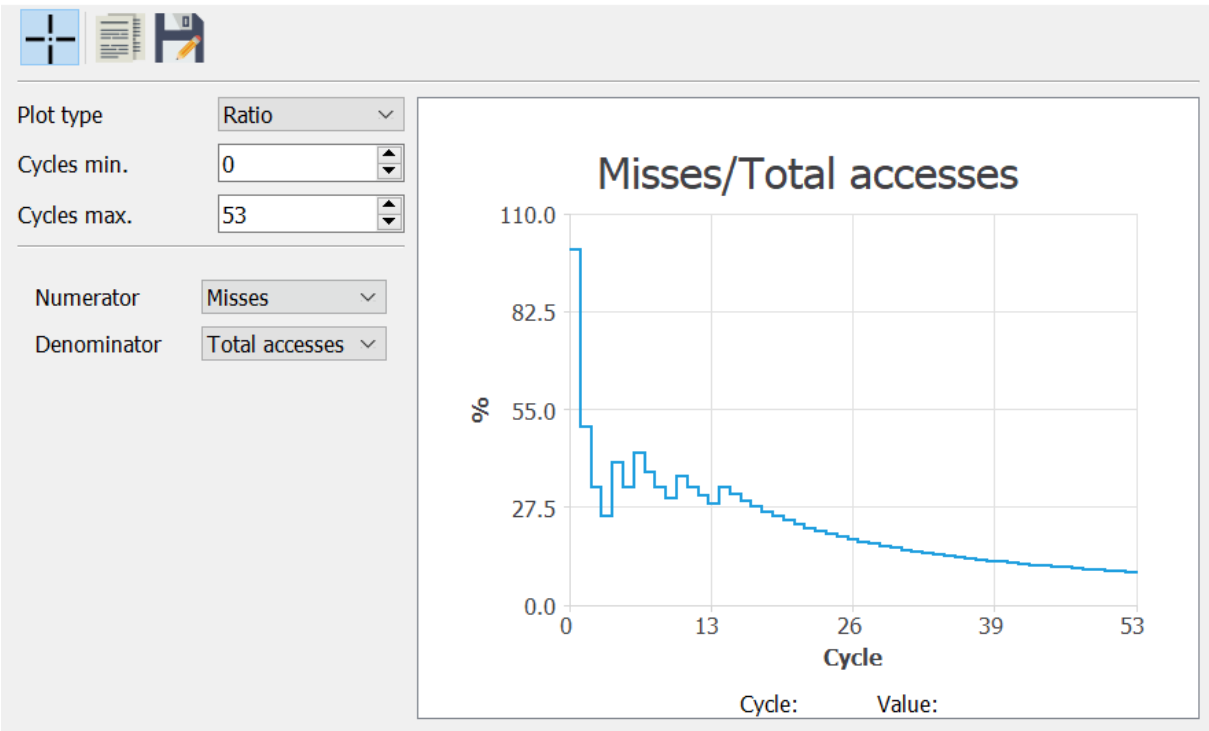


Imagen 14: Gráfico de acceso a la caché de instrucciones

Como se puede observar en la imagen 1, tenemos un máximo de 53 ciclos, que hace que haya un total de 100 accesos aproximadamente de la caché, haciendo que se cree una línea uniforme a lo largo de 53 ciclos, en total hay 5 misses.

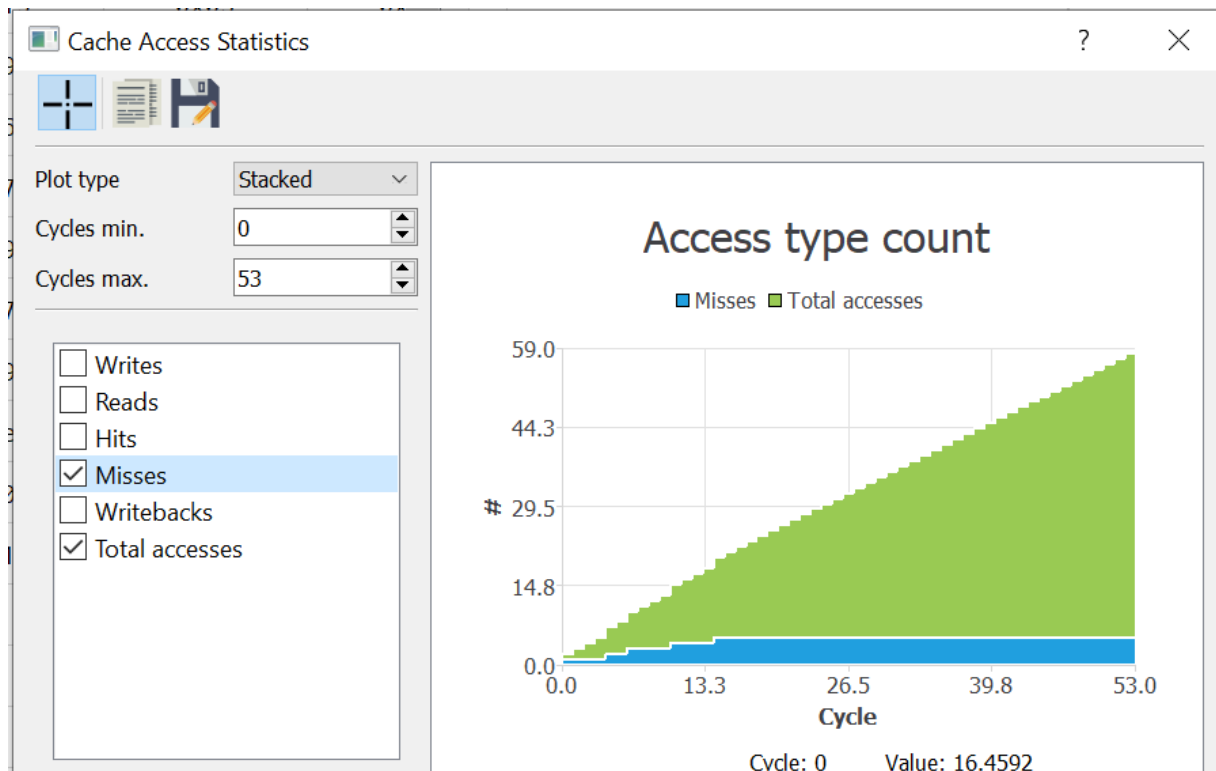


Imagen 15: Contador de tipo de accesos

Tenemos 53 ciclos como máximo en la gráfica de los accesos de caché, con una pérdida mínima (misses) comparada con el total de accesos.

Respuesta de observación

En el formato -O3, la gráfica de data cache nos sale vacía ya que lo único que hacemos es asignar un valor a la variable fija de retorno a0.

El número de hits que nos está otorgando la gráfica de Instructions caché, es debido a que lo único que estamos guardando en la memoria RAM, y haciendo accesos rápidos desde caché es a las instrucciones. Es decir, guardamos en la memoria RAM las instrucciones, y los accesos a memoria RAM para encontrar las instrucciones guardadas en la memoria caché para no tener que bajar a la memoria RAM. Como solo actuamos con las instrucciones, no existen datos en la memoria caché.