UNIVERSIDAD NEBRIJA

Luis Ruiz Moreno
Estructura de computadores

Práctica 5

Análisis de código C/C++ y memoria caché

UNIVERSIDAD NEBRIJA

Ejercicio 1

Realizar el análisis del código ensamblador generado a partir del código C. Modifique el código con tantos comentarios y normas de estilo como sean necesarios para que se entienda claramente su funcionamiento.

```c
int arg = 7;

int fact(int n) {
    int c;
    int result = 1;

    for(c = 1 ; c <= n ; c++)
        result = result ^c;

    return result;
}
```

A partir del anterior código en C, generamos con la herramienta **https://godbolt.org/** el siguiente código ensamblador:

```
fact:                                   # @fact
        blez    a0, .LBB0_4
        neg     a1, a0
        li      a2, 1
        li      a0, 1
        li      a3, 1
.LBB0_2:                                # =>This Inner Loop Header: Depth=1
        xor     a0, a0, a3
        addi    a3, a3, 1
        add     a4, a1, a3
        bne     a4, a2, .LBB0_2
        ret
.LBB0_4:
        li      a0, 1
        ret
arg:
        .word   7                               # 0x7

.Ldebug_list_header_start0:
        .half   5                               # Version
        .byte   4                               # Address size
        .byte   0                               # Segment selector size
        .word   3                               # Offset entry count
        .word   .Ldebug_loc0-.Lloclists_table_base0
        .word   .Ldebug_loc1-.Lloclists_table_base0
```

```
        .word   .Ldebug_loc2-.Lloclists_table_base0
        .byte   4                               # DW_LLE_offset_pair
        .byte   1                               # Loc expr size
        .byte   90                              # DW_OP_reg10
        .byte   4                               # DW_LLE_offset_pair
        .byte   1                               # Loc expr size
        .byte   90                              # DW_OP_reg10
        .byte   0                               # DW_LLE_end_of_list
        .byte   4                               # DW_LLE_offset_pair
        .byte   3                               # Loc expr size
        .byte   17                              # DW_OP_consts
        .byte   1                               # 1
        .byte   159                             # DW_OP_stack_value
        .byte   4                               # DW_LLE_offset_pair
        .byte   1                               # Loc expr size
        .byte   90                              # DW_OP_reg10
        .byte   4                               # DW_LLE_offset_pair
        .byte   3                               # Loc expr size
        .byte   17                              # DW_OP_consts
        .byte   1                               # 1
        .byte   159                             # DW_OP_stack_value
        .byte   0                               # DW_LLE_end_of_list
        .byte   4                               # DW_LLE_offset_pair
        .byte   3                               # Loc expr size
        .byte   17                              # DW_OP_consts
        .byte   1                               # 1
        .byte   159                             # DW_OP_stack_value
        .byte   4                               # DW_LLE_offset_pair
        .byte   1                               # Loc expr size
        .byte   93                              # DW_OP_reg13
        .byte   4                               # DW_LLE_offset_pair
        .byte   3                               # Loc expr size
        .byte   17                              # DW_OP_consts
        .byte   1                               # 1
        .byte   159                             # DW_OP_stack_value
        .byte   0                               # DW_LLE_end_of_list
.Ldebug_list_header_end0:
```

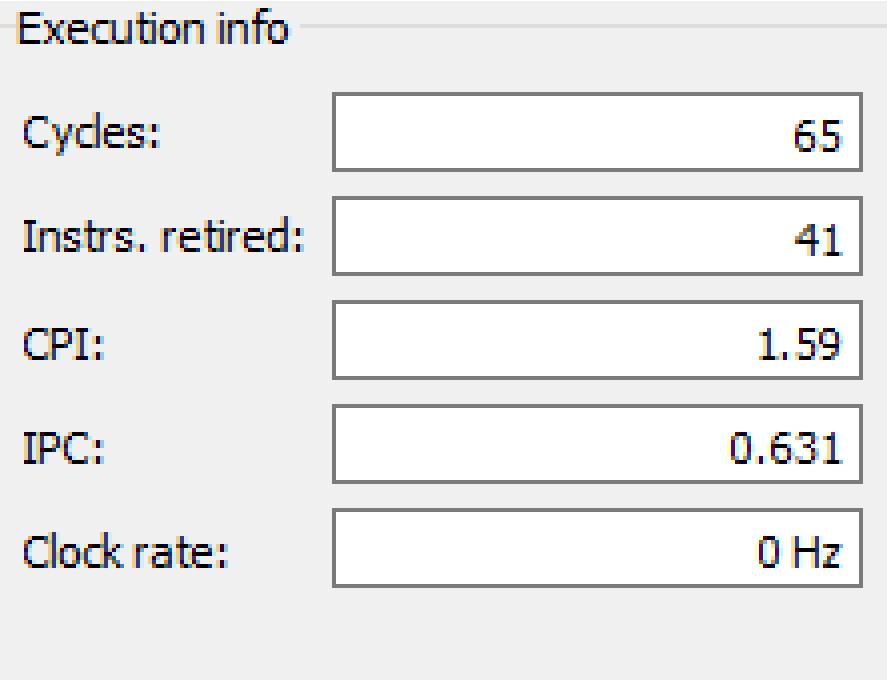
Para que funcione en el simulador Ripes, tengo que modificarlo ligeramente. Para ello distinguimos la sección **.data y .text.** Aparte escribimos el *main program* que llame al procedimiento.

```
.data
n:     .word 7

.text
lw a0, n     # call procedure
jal ra,fact
```

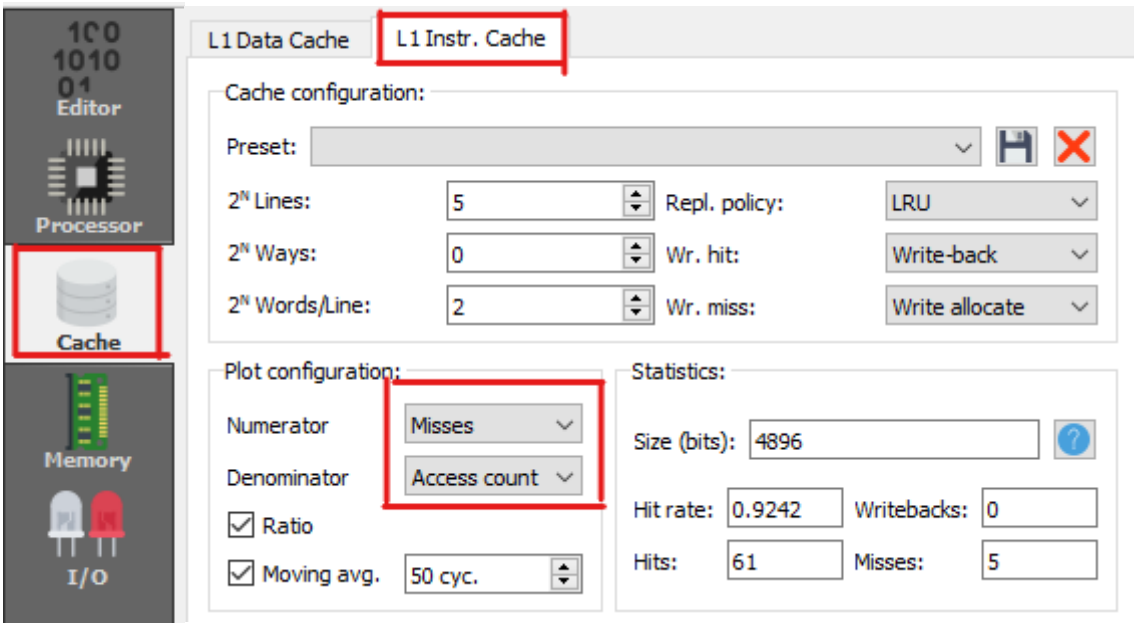
```
li a7,1     # print result
ecall

li a7,10     # exit
ecall

#-----------------------------------------------------------------
fact:    # a0 : n
         # a1 : return value
         # function: powers of 1 (exp=[1,n-1])
#-----------------------------------------------------------------
         blez    a0, LBB0_4      1
         neg     a1, a0
         li      a2, 1
         li      a0, 1
         li      a3, 1
LBB0_2:
Header: Depth=1
         xor     a0, a0, a3
         addi    a3, a3, 1
         add     a4, a1, a3
         bne     a4, a2, LBB0_2
         ret
LBB0_4:
         li      a0, 1          # return 1
         ret
```

Ejercicio 2

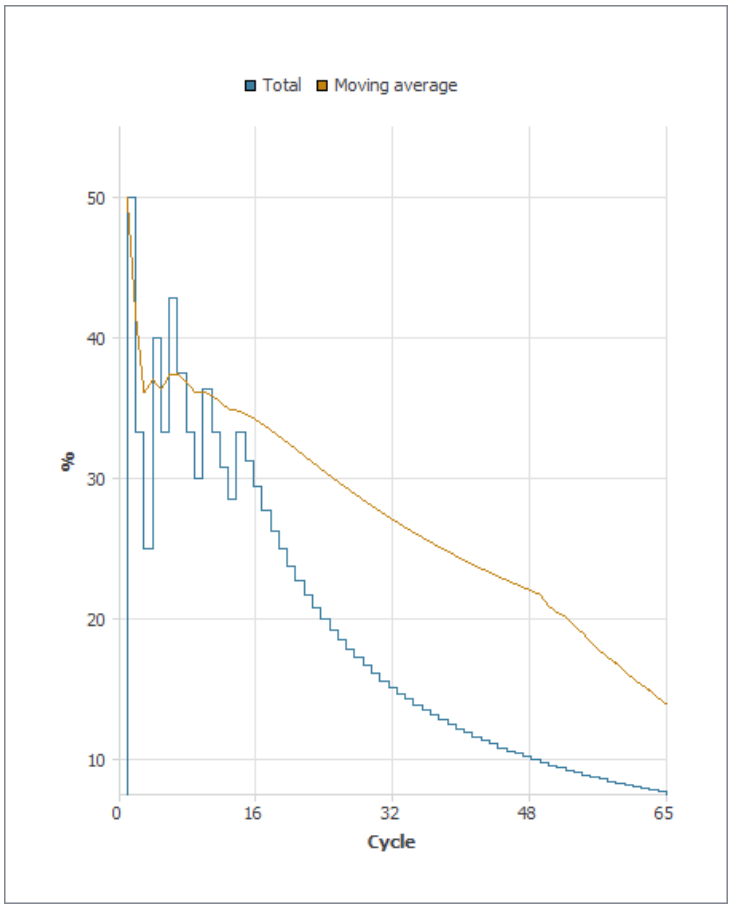
Primeramente, adjunto el número de ciclos ejecutados por el procesador, aparte de el número de instrucciones por ciclo (CPI).



Para la creación de una gráfica con el número de “misses” vs el número total de accesos a la memoria caché de instrucciones en Ripes, accedemos a la sección de caché. Posteriormente seleccionamos la pestaña de L1 Instr. Cache donde configuraremos la creación de la gráfica.



Dando el siguiente resultado:



Vemos que a medida que se ejecutan más ciclos de reloj hay una disminución en cuanto a los misses. Este comportamiento se puede deber a que al acceder repetidamente a los mismos datos o instrucciones se almacenan en la caché y reduce la probabilidad de futuros misses.