Ejercicio 1

Sea la definición de una estructura `struct s`, de la cuál conocemos la existencia pero ningún otro atributo.

1. Utilice el siguiente código assembly x86-64 para recuperar la definición de la estructura mencionada. Observe que todas las funciones reciben como primer parámetro un puntero una estructura `struct s`.

2. Indique los prototipos de las funciones presentes en el código.

3. Indique claramente los espacios donde cree que puede haber padding. Justifique.

```
_f1:
        movsbl  128(%rdi), %eax
        ret
_f2:
        movl    (%rdi), %eax
        ret
_f3:
        movq    24(%rdi), %rax
        movl    $0, %esi
        jmp .L4
.L6:
        leaq    2(%rdx,%rsi,4), %rcx
        movq    8(%rdi,%rcx,8), %rcx
        cmpq    %rcx, %rax
        cmovl   %rcx, %rax
        addq    $1, %rdx
        jmp .L7
.L9:
        movl    $0, %edx
.L7:
        cmpq    $3, %rdx
        jbe .L6
        addq    $1, %rsi
.L4:
        cmpq    $2, %rsi
        jbe .L9
        rep ret
_f4:
        movq    %rsi, %rax
        movq    %rsi, 16(%rdi)
        ret
_f5:    # unsigned long _f5 (void);
        movl    $136, %eax       # return sizeof (struct s)
        ret
_f6:
        movsd   8(%rdi), %xmm0
        ret
_f7:
        movsbq  128(%rdi), %rax
        cmpq    %rax, 120(%rdi)
        setb    %al
        ret
_f8:
        addq    %rsi, %rsi
        addq    16(%rdi), %rsi
        movzwl  (%rsi), %eax
        ret
_f9:
        movq    %rsi, 120(%rdi)
        ret
```

Ejercicio 2

En el siguiente ejercicio se pide hacer un seguimiento de pedidos de datos hechos por la CPU, utilizando memoria virtual. El sistema cuenta con una TLB, paginado de 1 nivel (1-level page table) y cache L1 (no dispone de cache L2 ni L3).

Las direcciones virtuales son de 20 bits y las direcciones físicas de 16 bits. La TLB es asociativa de 8 vías con un total de 32 entradas (se muestran 16 en la tabla). La memoria cache se muestra completa. Se muestran las primeras 32 entradas de la tabla de paginación. El *page size* es de 4096 bytes. Los accesos a memoria se realizan de a bytes.

Index	Tag	PPN	V
0	03	B	1
	07	6	0
	28	3	1
	01	F	0
1	31	0	1
	12	3	0
	07	E	1
	0B	1	1
2	2A	A	0
	11	1	0
	1F	8	1
	07	5	1
3	07	3	1
	3F	F	0
	10	D	0
	32	0	0

(a) TLB

VPN	PPN	V	VPN	PPN	V
00	6	1	10	0	1
01	5	0	11	5	0
02	3	1	12	2	1
03	4	1	13	4	0
04	2	0	14	6	0
05	7	1	15	2	0
06	1	0	16	4	0
07	3	0	17	6	0
08	1	1	18	1	1
09	4	0	19	2	0
0A	3	0	1A	5	0
0B	2	0	1B	2	0
0C	5	0	1C	6	0
0D	6	0	1D	2	0
0E	1	1	1E	3	0
0F	0	0	1F	1	0

(b) Page Table

Index	Tag	V	Byte 0	Byte 1	Byte 2	Byte 3	Tag	V	Byte 0	Byte 1	Byte 2	Byte 3
0	19	1	99	11	23	11	00	0	99	11	23	11
1	15	0	4F	22	EC	11	2F	1	55	59	0B	41
2	1B	1	00	02	04	08	0B	1	01	03	05	07
3	06	0	84	06	B2	9C	12	0	84	06	B2	9C
4	C7	1	43	6D	8F	09	CD	1	8F	09	84	06
5	0D	1	36	32	00	78	1E	1	A1	B2	C4	DE
6	11	0	A2	37	68	31	00	1	BB	77	33	00
7	16	1	11	C2	11	33	F2	1	00	C0	0F	00

(c) Cache L1

Cuadro 1: Tablas de memoria

1. Utilize un esquema con los bits de la memoria para indicar claramente los bits utilizados para el *offset* de la página física, el número de página física, el número de página virtual, el *offset* de la página virtual, el *offset* de cache, el índice de cache, el *tag* de cache, el *tag* de TLB, y el índice de TLB.

2. Complete la siguiente tabla. La columna VA indica las direcciones virtuales que pide la CPU:

VA	VPN	VPO	TLBi	TLBt	PA	PPN	PPO	Co	Ci	Ct	¿TLB Hit?	¿Cache Hit?	¿Page Fault?	Dato
59E5D														
1B5B4														
088F3														

Ejercicio 3

Dada la siguiente secuencias de instrucciones x86_64, ejecutadas en una arquitectura de 5 etapas (IF, ID, EX, MEM, WB) equivalente a la vista en clase para Y86_64:

```
v:
        leaq    (%rdi,%rdi,2), %rdx
        leaq    0(,%rdx,8), %rax
        addq    m(%rax), %rdi
        movq    %rdi, %rax
        ret
func:
        pushq   %rbp
        pushq   %rbx
        movl    $0, %ebx
        movl    $0, %ebp
        jmp .L3
.L4:
        movq    %rbx, %rdi
        call    v
        addq    %rax, %rbp
        addq    $1, %rbx
.L3:
        cmpq    $1, %rbx
        jbe .L4
        movq    %rbp, %rax
        popq    %rbx
        popq    %rbp
        ret
```

1. Complete un cuadro de evolución del estado del *pipeline* para una llamada completa a la función `func`, considerando que `m` es una matriz en memoria (`extern unsigned long matriz[][]`).

   La arquitectura cuenta con *data forwarding*, un predictor de saltos *BTFN*, y registros de pipeline con capacidad de demorar la ejecución e insertar burbujas si fuese necesario.

   Indique claramente si se insertan burbujas y dónde.