Ejercicio 1

1. Dado el siguiente código, basado en el código fuente de la aplicación **ffmpeg** (utilizada para [de]codificar/reproducir/manipular archivos multimedia), se pide:
    - Indicar el tamaño en bytes de las 3 estructuras: `ogg_stream`, `ogg_state` y `ogg`, considerando que los datos deben cumplir con restricciones de alineamiento.
    - Escribir el código assembly x86_64 correspondiente a las funciones:
        - `int ogg_find_stream (struct ogg * ogg, int32_t serial)`,
        - `struct ogg_state * ogg_find_state (struct ogg * ogg)`.
    - En caso de haber *padding*, muestre claramente su tamaño y ubicación.

```
 1: struct ogg_stream {
 2:     uint8_t *buf;
 3:     uint32_t pduration;
 4:     uint32_t serial;
 5:     uint64_t granule;
 6:     uint64_t start_granule;
 7:     int32_t header;
 8:     int32_t nsegs;
 9:     uint8_t segments[27];
10:     int32_t incomplete;
11:     int32_t page_end;
12:     int32_t start_trimming;
13:     int32_t end_trimming;
14:     uint8_t *new_metadata;
15:     int32_t new_metadata_size;
16:     void *private;
17:     const struct ogg_codec *codec;
18: };
19:
20: struct ogg_state {
21:     uint64_t pos;
22:     int32_t curidx;
23:     struct ogg_state *next;
24:     int32_t nstreams;
25:     struct ogg_stream streams[1];
26: };
27:
28: struct ogg {
29:     struct ogg_stream *streams;
30:     int32_t nstreams;
31:     int32_t headers;
32:     int32_t curidx;
33:     int64_t page_pos;
34:     struct ogg_state *state;
35: };
36:
37: int ogg_find_stream (struct ogg * ogg, int32_t serial)
38: {
```

```
39:     for (int i = 0; i < ogg->nstreams; i++)
40:         if (ogg->streams[i].serial == serial)
41:             return i;
42:
43:     return -1;
44: }
45:
46: struct ogg_state * ogg_find_state (struct ogg * ogg)
47: {
48:    struct ogg_state * state = ogg->state;
49:    while (state != NULL && state->curidx != ogg->curidx) {
50:       state = state->next;
51:    }
52:    return (state != NULL && state->curidx == ogg->curidx) ? state
: NULL;
53: }
```

Ejercicio 2

1. ¿Qué se entiende por especulación en el contexto de la emisión múltiple dinámica?
2. ¿Qué es una *Very Long Instruction Word* y cuál es su relación con el *scheduling* que realiza el compilador?
3. ¿A qué se debe la separación de la memoria caché L1 en L1-datos y L1-instrucciones?
4. Responda si son verdaderas o falsas las siguientes afirmaciones y **justifique** cada respuesta (las respuestas sin justificar serán consideradas incorrectas):

   a. La ejecución fuera de orden es un problema que surje al hacer predicciones de salto incorrectas.

   b. En assembly x86_64 nunca es necesario pushear el registro `%rax` ya que es usado para retornar los valores de las funciones.

   c. El *stack* que utilizan los procesos que se ejecutan en la PC se define en la ISA.

   d. Las tablas de paginación multinivel se llaman así porque un nivel de la tabla se aloja siempre en la memoria cache SRAM y los demás en la DRAM.

   e. No hay representaciones numéricas que tengan doble cero, es decir, cero positivo y cero negativo.

   f. Al ejecutar múltiples procesos en una computadora, la MMU se encarga de evitar que un proceso acceda a los datos de otro.

   g. Si no se encuentra un dato en la TLB, se produce un *page fault*.

   h. El *page fault* el proceso mediante el cual el procesador carga en el disco datos que antes no estaban y originaron un *page miss*.

   i. La relación entre cache *hits* y *misses* es una propiedad del diseño del procesador.

j. Dado un *datapath* secuencial tiene un *throughput* de 1 GIPS, si se lo segmenta en 5 etapas el *throughput* resultante es de 5 GIPS (si no se considera el retardo que agregan los registros de pipeline)

k. Si la frecuencia de operación de una arquitectura secuencial es 400 MHz y la misma se divide en 5 etapas, la frecuencia de operación resultante es 1.93 GHz.

l. Siempre resulta conveniente el uso de las política *write-back/write-allocate* cuando se diseñan memorias caché.

m. Nunca es posible reducir el CPI a un número menor a 1, aunque es ideal hacerlo.

n. Dada una representación de números similar a la estándar IEEE-754, con 2 bits de exponente y 4 de fracción, el número 5.6250 se puede representar en forma exacta.

o. Usando la representación en punto flotante dada por el estándar IEEE-754, los números normalizados siempre se encuentran “lejos” del 0.

p. Todo número representable en el formato del estándar IEEE-754 se puede representar también en una representación similar con una mayor cantidad de bits en su exponente.