

Nombre:	
DNI:	Grupo:

Examen Test (3.0p)

Todas las preguntas son de elección simple sobre 4 alternativas.

Cada respuesta vale 3/30 si es correcta, 0 si está en blanco o claramente tachada, -1/30 si es errónea.

Anotar las respuestas (a, b, c o d) en la siguiente tabla.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

- Si queremos almacenar la palabra de 64bits 0x0000001f fffffe0 en una memoria de bytes según la convención little-endian a partir de la posición 0x0804913c, quedará
 - 0x00 en 0x0804913c y 0xe0 en 0x08049143
 - 0x1f en 0x0804913c y 0xe0 en 0x08049140
 - 0xe0 en 0x0804913c y 0x1f en 0x08049140
 - Todas las respuestas anteriores son incorrectas
- La diferencia entre el flag de acarreo y de overflow es que...
 - uno se activa cuando se opera con números con signo y otro cuando son sin signo
 - el flag de acarreo indica que ha habido acarreo en una operación con números enteros (ints), el de overflow indica que ha habido desbordamiento en una operación con números en punto flotante (p.f.)
 - ambos se recalculan tras cada operación aritmético-lógica con ints, correspondiendo al programador consultar uno u otro según piense que sus datos son con o sin signo
 - el de acarreo indica que el resultado es demasiado grande (para p.f.) o positivo (si se trata de ints) para poder almacenarse, el de overflow indica que es demasiado pequeño (p.f.) o negativo (ints)
- ¿Cuál de las siguientes secuencias de instrucciones multiplica el (contenido del) registro EAX por 18?
 - imull \$0x18, %eax
 - leal (%eax,%eax,8), %eax
leal (%eax,%eax), %eax
 - shll \$18, %eax
 - sarl \$1, %eax
imul \$9, %eax
- La instrucción IA32 test sirve para...
 - Testear el código de condición indicado, y poner un byte a 1 si se cumple
 - Mover el operando fuente al destino, pero sólo si se cumple la condición indicada
 - Realizar la operación resta (a-b) pero no guardar el resultado, sino simplemente ajustar los flags
 - Realizar la operación and lógico bit-a-bit (a&b) pero no guardar el resultado, sino simplemente ajustar los flags
- La instrucción leave equivale a:
 - movl %esp, %ebp; popl %esp
 - movl %ebp, %esp; popl %ebp
 - pushl %esp; movl %ebp, %esp
 - pushl %ebp; movl %esp, %ebp

6. alguna de estas opciones contiene algún elemento que no corresponde a los contenidos de un marco de pila GCC/Linux IA32.

- a. Argumentos de llamada a la función y Dirección de retorno
- b. Antiguo marco de pila y Registros salva-invocante
- c. Variables locales y Registros salva-invocado
- d. Variables globales y Valor de retorno de la función

7. Respecto a registros salva-invocante y salva-invocado en GCC/Linux IA32, ¿cuál de éstos es de distinto tipo que el resto?

- a. EAX
- b. EBX
- c. ECX
- d. EDX

8. La arquitectura x86-64 tiene:

- a. 8 registros de propósito general (RPG) de 64 bits (%rax, %rbx, ... %rsp, %rbp)
- b. 16 registros RPG de 64 bits
- c. 32 registros RPG de 64 bits
- d. 64 registros RPG de 64 bits

9. ¿Cuál de las siguientes listas menciona registros x86-64 del mismo tipo respecto a convenio de uso? (salva-invocante, invocado, etc)

- a. RAX, RBX, RCX, RDX
- b. RBX, RSI, RDI
- c. RSP, RBP
- d. CL, DX, R8d, R9

10. La zona roja en x86-64 Linux es...

- a. una zona de pila en donde no deben escribir las funciones invocadas (porque si se produjera una interrupción, el manejador de interrupción sobrescribiría los valores escritos en pila)
- b. una zona de pila en donde pueden escribir las funciones invocadas, pero teniendo en

cuenta que los valores escritos pueden verse alterados si se produce una interrupción

- c. una zona bajo (RSP) (adonde apunta RSP) que una función puede usar sin reservarla, pero sólo si no llama a ninguna otra función mientras la usa
- d. una zona bajo (RBP) (adonde apunta RBP) que una función puede usar sin reservarla, pero sólo si no llama a ninguna otra función mientras la usa

11. alguna de las siguientes *no* es una operación básica de la unidad de control:

- a. Transferir un registro a otro
- b. (Leer o escribir) un registro (de / a) memoria
- c. (Guardar o recuperar) un registro (en / de) la pila
- d. Realizar operación ALU y guardar resultado en registro

12. Una cola de precaptación sirve para:

- a. Reducir el efecto de los fallos de cache
- b. Disminuir el periodo de ciclo del cauce
- c. Aumentar el número de etapas del cauce
- d. Resolver ciertos problemas de dependencia de datos

13. Sólo una de las siguientes afirmaciones sobre memorias ROM es correcta. ¿Cuál?

- a. Para fabricar una ROM se deben conocer los datos que se desea que almacene
- b. Una PROM (Programmable ROM) se puede grabar usando un dispositivo programador que selectivamente funde contactos aplicándoles altas temperaturas mediante diminutas cabezas soldadoras ("equipo de puntas")
- c. Una EPROM (Electrically Progr. ROM) se puede grabar eléctricamente, sin fundir contactos, pero no se puede borrar
- d. Una EEPROM (Erasable EPROM) se puede grabar (eléctricamente), y borrar (usando rayos ultravioleta)

14. Una SRAM de 32Kx8bit (256Kbit) puede venir organizada en 512 filas, dedicando por tanto al decodificador de columnas...

- a. 6 bits
 - b. 7 bits
 - c. 8 bits
 - d. 9 bits
-

15. Para construir una DRAM de 4GB con pastillas de 512Mx4bit hacen falta

- a. 64 pastillas
 - b. 32 pastillas
 - c. 16 pastillas
 - d. 8 pastillas
-

16. Una sentencia en C del tipo `"while (test) body;"` puede transformarse en código "goto" como:

- a.

```
if (!test) goto done;
loop: body;
if (test) goto loop;
done:
```
 - b.

```
loop: body;
if (test) goto loop;
```
 - c.

```
if (test) goto true;
goto done;
true: body;
done:
```
 - d.

```
loop: if (test) goto done;
body;
goto loop;
done:
```
-

17. El cuerpo del siguiente código C:

```
unsigned copy(unsigned u) {return u;}
puede traducirse a ensamblador como:
```

- a. `movl 8(%ebp), %eax`
 - b. `movl %ebp, 8(%eax)`
 - c. `movl 8(%ebp), (%eax)`
 - d. `movl 8(%esp), %ebp`
-

18. Si representamos la fase Decode con una D, Execute con una E, Fetch con una F y Writeback con una W, el orden correcto de

las distintas fases de una instrucción máquina es:

- a. D E F W
 - b. F D E W
 - c. F W D E
 - d. D F E W
-

19. Si N es el número de instrucciones máquina de un programa, F es la frecuencia de reloj, y C el número promedio de ciclos por instrucción, el tiempo de ejecución del programa será:

- a. $N \cdot F / C$
 - b. $N \cdot F \cdot C$
 - c. $N \cdot C / F$
 - d. $N / (F \cdot C)$
-

20. ¿Cuál de las siguientes expresiones representa un direccionamiento inmediato?

- a. `%eax`
 - b. `$0x400`
 - c. `(%eax)`
 - d. `8(%ebp)`
-

21. La instrucción `movzbl %al, %eax`

- a. Pone a 0 el registro `%eax`
 - b. Copia en `%eax` el valor sin signo almacenado en `%al`, rellenando con ceros
 - c. Copia en `%eax` el valor del indicador de cero
 - d. Copia en `%eax` el valor de `%al` si el indicador de cero está activado
-

22. Dentro de una función declarada como `void swap(int *xp, int *yp)`, que intercambia los valores de los dos enteros cuyas direcciones de memoria (punteros) son pasadas como parámetros a la función, la instrucción `movl 12(%ebp), %ecx` copia en `%ecx`...

- a. el valor del entero apuntado por el puntero pasado como primer parámetro
- b. el valor del entero apuntado por el puntero pasado como segundo parámetro

- c. el valor del puntero pasado como primer parámetro
 - d. el valor del puntero pasado como segundo parámetro
-

23. Si la dirección del primer elemento de un vector de enteros z está almacenada en el registro `%edx` y la variable entera i está almacenada en el registro `%eax`, la instrucción máquina que realiza la operación $z[i]++$ es:

- a. `addl $1, (%edx,%eax,4)`
 - b. `addl $1, (%eax,%edx,4)`
 - c. `addl $1, (%edx,%eax)`
 - d. `addl $4, (%eax,%edx)`
-

24. En un camino de datos con un solo bus, para realizar la operación de copia de un registro $r1$ en un registro $r2$, es decir $r2 \leftarrow r1$, es necesario:

- a. Activar la carga del registro $r1$ y habilitar la salida triestado del registro $r2$
 - b. Habilitar la salida triestado del registro $r1$ y activar la carga del registro $r2$
 - c. Habilitar las salidas triestado de los registros $r1$ y $r2$ y activar la carga del registro $r2$
 - d. Habilitar la salida triestado del registro $r2$ y activar la carga de los registros $r1$ y $r2$
-

25. La salida de un campo del registro de microinstrucción que solapa dirección de salto y algunas señales de control han de conectarse a:

- a. una ROM o PLA
 - b. la memoria de control
 - c. el registro de instrucción
 - d. un demultiplexor controlado por el tipo de salto
-

26. Cuando dos o más instrucciones necesitan un recurso hardware en el mismo ciclo, se trata de un riesgo:

- a. estructural
- b. por dependencia de datos
- c. de control

- d. de salto
-

27. Una instrucción típica de entrada / salida tiene

- a. no tiene ningún parámetro
 - b. tiene un parámetro: un registro del procesador
 - c. tiene dos parámetros: un registro del procesador y una dirección de puerto de E/S
 - d. tiene tres parámetros: un registro del procesador, una dirección de puerto de E/S y una dirección de memoria
-

28. ¿Cuál de las siguientes afirmaciones acerca del daisy-chain es cierta?

- a. Todos los componentes se conectan directamente y con igual prioridad al procesador o gestor de interrupciones
 - b. Los componentes se comportan de forma cooperativa: sólo al de mayor prioridad se le concede la interrupción o se apodera del bus de comunicaciones
 - c. Los componentes están conectados todos con todos y un gestor centralizado decide la prioridad
 - d. Es incompatible con la técnica de sondeo o polling
-

29. La memoria DRAM:

- a. Se denomina dinámica porque para mantener almacenado un dato hay que recargarlo cada cierto tiempo en un ciclo de refresco
 - b. Es menos densa que la memoria SRAM
 - c. Se inventó en la década de los 90
 - d. Necesita 6 transistores por cada celda
-

30. Los módulos de memoria dinámica compactos que suelen usarse en los portátiles se denominan:

- a. SIMM
 - b. SODIMM
 - c. SLIM
 - d. MIN
-

Nombre:	
DNI:	Grupo:

Examen de Problemas (3.0p)

1. Bucles for (0.5 puntos). Una función, fun_c, tiene la siguiente estructura general:

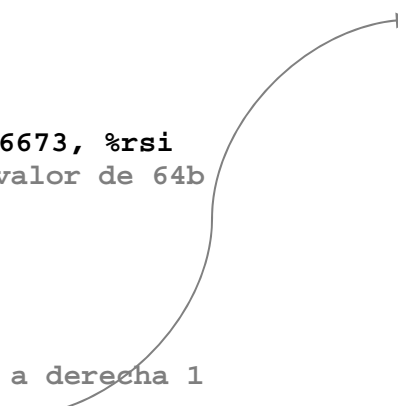
```
long fun_c(unsigned long x) {
    long val = 0;
    int i;
    for ( _____ ; _____ ; _____ ; ) {
        _____;
    }
    _____;
    return _____;
}
```

El compilador C GCC genera el siguiente código ensamblador:

```
# x en %rdi
fun_c:
    movl    $0, %ecx
    movl    $0, %edx
    movabsq $72340172838076673, %rsi
    # Mover un valor de 64b

.L2:
    movq    %rdi, %rax
    andq    %rsi, %rax
    addq    %rax, %rcx
    shrq    %rdi
    # Desplazar a derecha 1
    addl    $1, %edx

    cmpl    $8, %edx
    jne     .L2
    movq    %rcx, %rax
    sarq    $32, %rax
    addq    %rcx, %rax
    movq    %rax, %rdx
    sarq    $16, %rdx
    addq    %rax, %rdx
    movq    %rdx, %rax
    sarq    $8, %rax
    addq    %rdx, %rax
    andl    $255, %eax
    ret
```



Analizar el funcionamiento de este código y responder a las siguientes preguntas.
Resultará útil convertir la constante decimal de la línea 4 (`movabsq`) a hexadecimal.

- A. Usar la versión ensamblador para rellenar las partes que faltan del código C.
B. Describir en castellano qué calcula esta función.

2. Representación y acceso a estructuras (0.5 puntos). Para la declaración de estructura

```
struct {
    char      *a;
    short     b;
    double    c;
    char      d;
    float     e;
    char      f;
    long long g;
    void      *h;
} foo;
```

suponer que fue compilada en una máquina Windows (32b), donde cada tipo de datos primitivo de K bytes debe tener un desplazamiento múltiplo de K.

A. ¿Cuál es el desplazamiento en bytes de cada campo de la estructura?

B. ¿Cuál es el tamaño total de la estructura?

C. Reordenar los campos de la estructura para minimizar el espacio desperdiciado, y entonces mostrar los desplazamientos en bytes y tamaño total para la estructura reordenada.

Responder mediante una tabla con el siguiente formato:

(A)	nombre	n.campo2	total (B)
tamaño	tamaño campo1	t.campo2	tamaño
desplaz	desplaz. campo1	d.campo2	total
(C)	nombre	n.campo2	total
tamaño	tamaño campo1	t.campo2	tamaño
desplaz	desplaz. campo1	d.campo2	total

- Entrada/Salida** (0.5 puntos). Disponemos de un microprocesador de 8 bits (bus de datos de 8 bits y bus de direcciones de 16 bits) con E/S independiente. Diseñe un sistema de E/S que permita acceder a los siguientes puertos: puerto 0x0220 de salida y 0x0221 de entrada. Utilice lógica de decodificación distribuida. No emplee decodificadores.
- Diseño del sistema de memoria** (0.5 puntos). Diseñe un sistema de memoria para un procesador de 32 bits (cada dirección de memoria corresponde a una palabra de 32 bits) a partir de módulos SRAM 4K×8 y ROM 8K×16. La memoria SRAM debe ocupar las direcciones 0x0000 a 0x1FFF y la ROM las direcciones 0xC000 a 0xFFFF.
- Memoria cache** (0.5 puntos). Los parámetros que definen la memoria de un computador son los siguientes:
 - Tamaño de la memoria principal: 4 GB
 - Tamaño de la memoria cache: 64 KB
 - Tamaño de bloque: 256 B

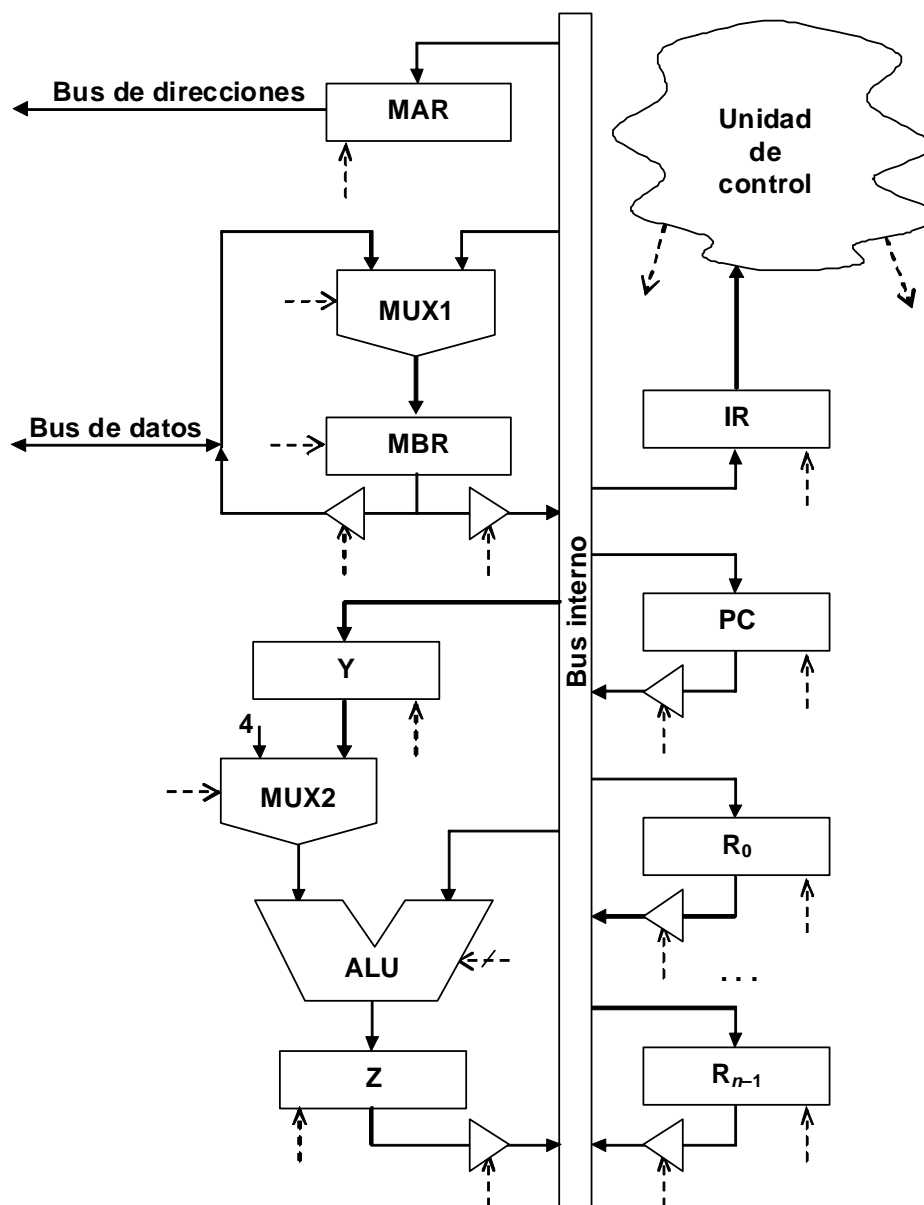
Determine el tamaño de los distintos campos de una dirección en las siguientes condiciones:

- Correspondencia completamente asociativa
- Correspondencia directa
- Correspondencia asociativa por conjuntos con 16 bloques por conjunto

6. **Unidad de control** (0.5 puntos). La figura muestra el camino de datos de un procesador de 32 bits que direcciona la memoria por bytes y en el que cada instrucción y cada dato ocupa una palabra completa (4 bytes). El multiplexor MUX2 selecciona o bien la salida del registro Y o un valor constante igual a 4 utilizado para incrementar el contenido del contador de programa. Las operaciones de lectura o escritura en memoria consumen un ciclo de reloj. Se desean implementar tres instrucciones de suma:

<i>Instrucción</i>	<i>Direccionamiento del operando fuente</i>	<i>Formato</i>
addr Rdst,Rsrc	registro	una palabra
addi Rdst,[Rsrc]	indirecto a memoria a través de registro	una palabra
addx Rdst,[Rsrc+desp]	indexado	dos palabras, la segunda contiene el desplazamiento

Escriba (en lenguaje de transferencia de registros o de alto nivel) un microprograma que incluya la fase de captación de instrucción y la fase de ejecución de cada una de esas tres instrucciones.



Nombre:	
DNI:	Grupo:

Examen de Prácticas (4.0p)

Todas las preguntas son de elección simple sobre 4 alternativas.

Cada respuesta vale 4/20 si es correcta, 0 si está en blanco o claramente tachada, -4/60 si es errónea.

Anotar las respuestas (a, b, c o d) en la siguiente tabla.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

1. Considerar los siguientes dos bloques de código almacenados en dos ficheros distintos:

```
/* main.c */      /* func.c */
int i = 0;          int i = 1;
int main() {        void func() {
    func();          printf("%d", i);
    return 0;        }
}
```

¿Qué sucederá cuando se compile, enlace y ejecute este código?

- Error al compilar o enlazar, no se obtiene ejecutable
- Escribe "0"
- Escribe "1"
- A veces escribe "0" y a veces "1"

2. ¿Qué hace gcc -S?

- Compilar .c → .s (fuente C a fuente ASM)
- Compilar .s → .o (fuente ASM a objeto)
- Compilar optimizando tamaño (size), no tiempo
- Compilar borrando del ejecutable la tabla de símbolos (strip)

3. ¿Qué modificador (switch) de gcc hace falta para compilar .c → .o (de fuente C a código objeto)?

- Eso no se puede hacer con gcc

- gcc -c
- gcc -o
- gcc -O

4. ¿Qué modificador (switch) de gcc hace falta para compilar una aplicación de 32bits (fuente C) en un sistema de 64bits en el que se ha instalado también el compilador de 32bits?

- m32
- 32
- m elf_i386

- No se puede conseguir ese efecto con modificadores de gcc, porque el gcc por defecto (/usr/bin/gcc) es el de 64bits, se debe usar el gcc de la instalación alternativa (de 32bits)

5. ¿Qué modificador (switch) de as hace falta para ensamblar una aplicación de 32bits en un sistema de 64bits en el que se ha instalado también el compilador de 32bits?

- m32
- 32
- m elf_i386

- Ninguna de las anteriores respuestas es correcta

6. Como parte del proceso de compilación de una aplicación en lenguaje C, enlazar .o → .exe (de objeto proveniente de fuente C a ejecutable) usando sólo as y ld, sin gcc...
- Se puede, repartiendo entre as y ld los modificadores (switches) que corresponda
 - Se puede, repartiendo modificadores entre as y ld, y añadiendo al comando ld el runtime de C
 - Basta usar ld, con los modificadores de gcc que corresponda, y añadiéndole el runtime de C
 - Ninguna de las anteriores respuestas es correcta

7. La práctica "popcount" debía calcular la suma de bits de los elementos de un array. Un estudiante entrega la siguiente versión de popcount4:

```
int popcount4(unsigned* array, int len)
{
    int i, j, res = 0;
    for(i = 0; i < len; ++i) {
        unsigned x = array[i];
        int n = 0;
        do {
            n += x & 0x01010101L;
            x >>= 1;
        } while(x);
        for(j = 16; j == 1; j /= 2){
            n ^= (n >>= j);
        }
        res += n & 0xff;
    }
    return res;
}
```

Esta función popcount4:

- Produce el resultado correcto
 - Fallaría con array={0,1,2,3}
 - Fallaría con array={1,2,4,8}
 - No es correcta pero el error no se manifiesta en los ejemplos propuestos, o se manifiesta en ambos
-
8. La práctica "paridad" debía calcular la suma de paridades impar (XOR de todos los bits) de los elementos de un array. Un estudiante entrega la siguiente versión de parity6:

```
int parity6(unsigned * array, int len)
{
    int i, result = 0;
    unsigned x;
    for (i=0; i<len; i++){
        x = array[i];
        asm( "mov %[x], %%edx \n\t"
            "shr $16, %%edx \n\t"
            "shr $8, %%edx \n\t"
            "xor %%edx,%%edx \n\t"
            "setp %%dl \n\t"
            "movzx %%dl, %[x] \n\t"
            : [x] "+r" (x)
            : "edx"
            );
        result += x;
    }
    return result;
}
```

Esta función parity6:

- Produce el resultado correcto
 - Fallaría con array={0,1,2,3}
 - Fallaría con array={1,2,4,8}
 - No es correcta pero el error no se manifiesta en los ejemplos propuestos, o se manifiesta en ambos
-
9. Respecto a las bombas estudiadas en la práctica 3, ¿en cuál de los siguientes tipos de bomba sería más **fácil** descubrir la(s) contraseña(s)? Se distingue entre strings definidos en el código fuente de la bomba, y strings solicitados al usuario mediante scanf(). Por "cifrar" se entiende aplicar la cifra del César (sumar o restar una constante fija a los códigos ASCII).
- 1 string del fuente se cifra, y se compara con el string del usuario
 - 2 strings del usuario se concatenan, se cifra el resultado y se compara con el string del fuente
 - 2 strings del usuario se cifran, se concatenan los resultados, y se compara con el string del fuente
 - Las opciones más fáciles son de la misma dificultad, así que no se puede marcar ninguna como la más fácil

10. Respecto a las bombas estudiadas en la práctica 3, ¿en cuál de los siguientes tipos de bomba sería más **difícil** descubrir la contraseña? Se distingue entre enteros definidos en el código fuente de la bomba, y enteros solicitados al usuario mediante `scanf()`. Por "procesar" se entiende calcular el n-ésimo elemento de la serie de Fibonacci.

- a. 1 entero del usuario se procesa, y se compara con el entero del fuente
- b. 2 enteros del fuente se suman, se procesa la suma, y se compara el resultado con el entero del usuario
- c. 2 enteros del fuente se procesan, se suman los resultados, y se compara la suma con el entero del usuario
- d. Las opciones más difíciles son de la misma dificultad, así que no se puede marcar ninguna como la más difícil

11. ¿Cuál de los siguientes es el orden correcto en el ciclo de compilación de un programa escrito en lenguaje C? (el fichero sin extensión es un ejecutable):

- a. `file.s` → `file.c` → `file` → `file.o`
- b. `file.c` → `file.s` → `file.o` → `file`
- c. `file.c` → `file.s` → `file` → `file.o`
- d. `file.s` → `file.c` → `file.o` → `file`

12. ¿Qué hace `gcc -O`?

- a. Compilar sin optimización, igual que `-O0`
- b. Compilar `.c` → `.o`
- c. Compilar `.s` → `.o`
- d. Compilar con optimización, igual que `-O1`

13. ¿Qué modificador (switch) de `gcc` hace falta para compilar `.s` → `.o` sin llamar al enlazador?

- a. Eso no se puede hacer con `gcc`
- b. `gcc -c`
- c. `gcc -s`
- d. `gcc -S`

14. ¿Qué modificador (switch) de `gcc` hace falta para compilar una aplicación de 32 bits en un sistema de 64 bits?

- a. `-m32`
- b. `-m64`
- c. `-32`
- d. `-64`

15. Compilar `.s` → ejecutable, usando sólo `as` y `ld`, sin `gcc`...

- a. No se puede
- b. Se puede, usando en `as` y `ld` los modificadores (switches) que corresponda
- c. Basta usar `as`, con los modificadores que corresponda
- d. Basta usar `ld`, con los modificadores que corresponda

16. ¿Cuál de las siguientes afirmaciones es cierta respecto al lenguaje C?

- a. En lenguaje C, al llamar a una subrutina o función se introducen los parámetros en la pila y después se realiza una llamada a la subrutina
- b. Los parámetros se introducen en la pila en el orden en el que aparecen en la llamada de C, es decir, empezando por el primero y acabando por el último
- c. Antes de volver de la rutina llamada, el programa en C se encarga de quitar de la pila los parámetros de llamada realizando varios `pop`
- d. Pasar a una función un puntero a una variable se traduce en introducir en la pila el valor de la variable

17. El primer parámetro de `printf`:

- a. es un `char`
- b. es un entero
- c. es un puntero
- d. puede ser de cualquier tipo, incluso no existir

18. Respecto al procesador que denominamos `ssrDLX` (procesador `DLX` sin segmentar), ¿cuál de las siguientes afirmaciones es falsa?

- a. Todas las instrucciones deben realizar las cuatro etapas del cauce

- b. Las etapas ID y WB duran un 80% del tiempo de ciclo de reloj
 - c. Una operación de suma de enteros (operandos y resultado en el banco de registros) necesitará 3.6 ciclos en el procesador sin segmentar
 - d. $f2,a$ en el procesador $ssrDLX$ requiere 4.6 ciclos
-

19. La directiva `.text` en el simulador DLX con la dirección 256, (`.text 256`), indica en un programa que:

- a. la variable `text` tiene el valor 0x100
 - b. Existe una etiqueta de salto denominada `.text` en la posición 0x100
 - c. La primera instrucción del programa se ubicará en la posición 0x100
 - d. Todas las anteriores afirmaciones son incorrectas
-

20. En un procesador de la familia 80x86 una variable de 32 bits, entera con signo, almacenada a partir de la dirección n contiene: 0xFF en la dirección n , 0xFF en la dirección $n+1$, 0xFF en la dirección $n+2$ y 0xF0 en la dirección $n+3$. ¿Cuánto vale dicha variable?

- a. -16
 - b. -251658241
 - c. 16
 - d. 4294967280
-

Examen Test (3.0p)

Todas las preguntas son de elección simple sobre 4 alternativas.
Cada respuesta vale 3/30 si es correcta, 0 si está en blanco o claramente tachada, -1/30 si es errónea.
Anotar las respuestas (a, b, c o d) en la siguiente tabla.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
c	c	b	d	b	d	b	b	d	c	c	a	a	a	c	a	a	b	c	b	b	d	a	b	d	a	c	b	a	b

Examen de Prácticas (4.0p)

Todas las preguntas son de elección simple sobre 4 alternativas.
Cada respuesta vale 4/20 si es correcta, 0 si está en blanco o claramente tachada, -4/60 si es errónea.
Anotar las respuestas (a, b, c o d) en la siguiente tabla.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
a	a	b	a	b	c	a	b	a	a	b	d	b	a	b	a	c	a	c	b

Examen de Problemas (3.0p)

1. Bucles for (0.5 puntos).

A.

```
long fun_c(unsigned long x) {
    long val = 0;
    int i;
    for (i = 0; i < 8; i++) {
        val += x & 0x0101010101010101;
        x >>= 1;
    }
    val += (val >> 32);
    val += (val >> 16);
    val += (val >> 8);
    return val & 0xFF;
}
```

B. Calcula el nº bits a 1 (popcount) de x.

Va haciendo 8 sumas parciales en paralelo en los 8 bytes de val.

Luego se suman las 4 sumas superiores (7-4) con las 4 inferiores (3-0),

se siguen acumulando 2 superiores (3-2) con 2 inferiores (1-0),

y por último, la suma del byte (1) con la del byte inferior (0).

Se retorna sólo ese byte, en donde están sumados todos los bits.

La ventaja es que hace sólo 8 iteraciones, 3 sumas y una máscara en lugar de 64 iteraciones.

2. Representación y acceso a estructuras (0.5 puntos).

(A)	a	b	c	d	e	f	g	h	total (B)
tamaño	4	2	8	1	4	1	8	4	(4B relleno)
desplaz	0	4	8	16	20	24	32	40	48

(C)	c	g	a	e	h	b	d	f	total (C)
tamaño	8	8	4	4	4	2	1	1	(sin relleno)
desplaz	0	8	16	20	24	28	30	31	32

3. Entrada/Salida (0.5 puntos).

Se podrían usar 2 (N)ANDs para decodificar: E/S 0x220 activa IO/M, A9 y A5, 0x221 activa además A0
 Drivers triestado E/S (E obligatorio) desde-hacia terminales Q-D de los registros (tipo D, 8bits)
 Segundo nivel (N)AND según R/W para CLK-Load Salida o driver Entrada (driver Salida activado decodificador)
 Se espera que temporización R/W \subset IO/M-Addr, para que CLK-Load pulse con datos estables

4. Diseño del sistema de memoria (0.5 puntos).

SRAM: módulos 4Kx8 \rightarrow 12bits Addr, 8bits Data
 espacio 0x0-0x1fff=13bits \rightarrow 8K
 $8K \times 32 = (4K \times 8) \times (2 \times 4) \rightarrow 2 \times 4 = 8$ módulos
 ROM: módulos 8Kx16 \rightarrow 13bits Addr, 16bits Data
 espacio 0xc000-0xffff como 0x0-3fff=14bits \rightarrow 16K
 $16K \times 32 = (8K \times 16) \times (2 \times 2) \rightarrow 2 \times 2 = 4$ módulos
 Dibujar: SRAM: dos hileras de 4 módulos, 0x0-0xfff y 0x1000-0x1fff,
 datos D0-7, D8-15, D16-23, D24-31, direcciones A0-11, a todos los módulos
 dirs A12-31 para decodificadores NAND (todas invertidas 1ª hilera, A12 no invertida 2ª hilera)
 ROM: dos hileras de 2 módulos, 0xc000-0xdfff y 0xe000-0xffff,
 datos D0-15, D16-31, direcciones A0-12, a todos los módulos, A13-31 para decods NAND
 1ª hilera todas invertidas salvo A15-14 (0xc*, 0xd*), 2ª hilera A15-13 (0xe*, 0xf*)
 R/W a todas las SRAM (R/W) y ROM (OE) (R/W invertida si OE activa baja)

5. Memoria cache (0.5 puntos).

- A. Asociativa: $32 = 24 + 8$. etiqueta +palabra.
 B. Directa: $32 = 16 + 8 + 8$. etiqueta+marco+palabra. $2^{16} \text{ B/cach} / 2^8 \text{ B/bloq} = 2^8 \text{ bloq/cache}$
 C. Asociativa Conjuntos 16vías: $32 = 20 + 4 + 8$. etiqueta+conjnt+palabra. $2^8 \text{ bloq/cach} / 2^4 \text{ blq/conj} = 2^4 \text{ conj/cache}$

6. Unidad de control (0.5 puntos).

FETCH: MAR := PC, Z := PC+4
 Read, PC := Z
 MBR := Mem
 IR := MBR
 goto f(IR)
 ADDR: Y := Rsrc
 Z := Y + Rdst
 Rdst := Z, goto FETCH
 ADDI: MAR := Rsrc
 ADDI_rd: Read
 MBR := Mem, Y := Rdst
 Z := Y + MBR
 Rdst := Z, goto FETCH
 ADDX: MAR := PC, Z := PC+4
 Read, PC := Z
 MBR := Mem, Y := Rsrc
 Z := Y + MBR
 MAR := Z, goto ADDI_rd

Detallar señales de control no se pide
 (EN_{PC}, LD_{MAR}), (EN_{PC}, MUX2₄, ALU₊, LD_Z)
 (ciclo lectura), (EN_Z, LD_{PC})
 (MUX1_{MEM}, LD_{MBR})
 (L_{IR}, EN_{MBR-BUS})
 No sabemos cómo obtener codop de IR, cómo va goto, suponer f(IR)
 (EN_{Rsrc}, LD_Y) obtener src-dst de IR? suponer que vale decir Rsrc
 (MUX2_Y, ALU₊, EN_{Rdst}, LD_Z)
 (EN_Z, LD_{Rdst}) No sabemos cómo va goto, suponer OK en paralelo
 (EN_{Rsrc}, LD_{MAR})
 (MUX1_{MEM}, LD_{MBR}), (EN_{Rdst}, LD_Y)
 (MUX2_Y, ALU₊, EN_{MBR-BUS}, LD_Z)
 (EN_Z, LD_{Rdst})
 (EN_{PC}, LD_{MAR}), (EN_{PC}, MUX2₄, ALU₊, LD_Z)
 (como fetch), (EN_Z, LD_{PC})
 (MUX1_{MEM}, LD_{MBR}), (EN_{Rsrc}, LD_Y)
 (MUX2_Y, ALU₊, EN_{MBR-BUS}, LD_Z)
 (EN_Z, LD_{MAR}), resto como addi