2º Grado Informática
Estructura de Computadores
28 Enero 2013

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

a. fich.c -> fich.o -> fich.s -> fich
b. fich -> fich.s -> fich.o -> fich.c
c. fich.c -> fich.s -> fich -> fich.o
d. fich.c -> fich.s -> fich.o -> fich

2. ¿Qué hace gcc -O?

a. Compilar con optimización suave
b. Compilar .c->.o (fuente C a objeto)
c. Compilar .s->.o (fuente ASM a objeto)
d. Ambas (b) y (c), según la extensión de los ficheros que se usen como argumentos

3. ¿Qué modificador (switch) de gcc hace falta para compilar .c->.s (de fuente C a fuente ASM)?

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

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

a. -m32
b. -32
c. -m elf_i386
d. No hace falta modificador, ld lo deduce del tipo de objeto a enlazar

5. Compilar .c->exe (de fuente C a ejecutable) usando sólo as y ld, sin gcc...

a. No se puede
b. Se puede, repartiendo entre as y ld los modificadores (switches) que corresponda
c. Se puede, repartiendo modificadores entre as y ld, y añadiendo al comando ld el runtime de C
d. Basta usar ld, con los modificadores de gcc que corresponda, y añadiéndole el runtime de C

6. ¿Cuál de las siguientes afirmaciones es falsa 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 inverso a como aparecen en la llamada de C, es decir, empezando por el último y acabando por el primero
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 dirección de memoria donde está almacenada la variable

---

7. El primer parámetro de printf:

a. es un entero
b. es un número en coma flotante
c. es un puntero
d. puede ser de cualquier tipo

---

8. Para averiguar la paridad de un número se puede usar la operación:

a. NOT
b. AND
c. OR
d. XOR

---

9. Utilizando la sentencia asm, las denominadas restricciones que se indican al final de dicha sentencia, involucran a:

a. Solamente las entradas
b. Solamente las salidas
c. Solamente los sobrescritos
d. Ninguna de las anteriores es cierta

---

10. Suponga la siguiente sentencia asm en un programa:

```
asm(“ add (%[a],%[i],4),%r”
    :[r] “+r” (result)
    :[i] “r” (i),
     [a] “r” (array) );
```

¿Cuál de las siguientes afirmaciones es correcta?

a. r es una posición de memoria de entrada/salida
b. a es una posición de memoria de entrada
c. i es un registro de entrada
d. la salida de la función se fuerza a que esté en la variable result

---

11. En una bomba como las estudiadas en la práctica 4, del tipo…

```
0x0804873f <main+207>: call   0x8048504 <scanf>
0x08048744 <main+212>: mov    0x24(%esp),%edx
0x08048748 <main+216>: mov    0x804a044,%eax
0x0804874d <main+221>: cmp    %eax,%edx
0x0804874f <main+223>:  je 0x8048756 <main+230>
0x08048751 <main+225>: call   0x8048604 <boom>
0x08048756 <main+230>:  ...
```

la contraseña es...

a. el entero 0x804a044
b. el entero almacenado a partir de la posición de memoria 0x804a044
c. el string almacenado a partir de la posición de memoria 0x24(%esp)
d. ninguna de las anteriores

---

12. En una bomba como las estudiadas en la práctica 4, del tipo…

```
0x08048705 <main+149>: call   0x80484c4
                                <gettimeofday>
...
0x08048718 <main+168>: cmp    $0x5,%eax
0x0804871b <main+171>: jle 0x8048722 <main+178>
0x0804871d <main+173>: call   0x8048604 <boom>
0x08048722 <main+178>: ...
```

ejecutada paso a paso con el depurador ddd, interesaría...

a. ejecutar hasta jle, ajustar %eax a 6, y continuar ejecutando paso a paso
b. ejecutar hasta jle, ajustar %eax a 4, y continuar ejecutando paso a paso
c. cambiar jle por jmp usando ddd o un editor hex, salvar el programa y reiniciar la depuración con el nuevo ejecutable
d. Ninguna de las opciones anteriores es de interés (bien porque no se pueda hacer eso o porque no sirva para evitar la bomba)

---

13. En una bomba como las estudiadas en la práctica 4, del tipo…

```
0x080486e8 <main+120>: call   0x8048524 <strncmp>
0x080486ed <main+125>: test   %eax,%eax
0x080486ef <main+127>: je 0x80486f6 <main+134>
0x080486f1 <main+129>: call   0x8048604 <boom>
0x080486f6 <main+134>: ...
```

la contraseña es...

a. el valor que tenga %eax
b. el string almacenado a partir de donde apunta %eax
c. el entero almacenado a partir de donde apunta %eax
d. ninguna de las anteriores

---

14. Respecto a las bombas estudiadas en la práctica 4, ¿en cuál de los siguientes tipos de bomba sería más difícil descubrir la contraseña? Se distingue entre strings definidos en el código fuente de la bomba, y strings solicitados al usuario mediante scanf(). Por "cifrar" podemos entender la cifra del César, por ejemplo. "Invertir" es darle la vuelta al string de manera que la primera letra se convierta en la última y viceversa.

a. 1 string del usuario se cifra, y se compara con el string del fuente
b. 2 strings del fuente se invierten, se concatenan, se cifra el resultado, y se compara con el string del usuario
c. 2 strings del fuente se concatenan, se invierte el resultado, se cifra, y se compara con el string del usuario
d. Las 2 (o 3) opciones más difíciles son de la misma dificultad, así que no se puede marcar ninguna como la más difícil

---

15. Respecto a las bombas estudiadas en la práctica 4, ¿en cuál de los siguientes tipos de bomba sería más difícil descubrir la(s) contraseña(s)? Se distingue entre enteros definidos en el código fuente de la bomba, y enteros solicitados al usuario mediante scanf(). Por "procesar" podemos entender calcular el n-ésimo elemento de la serie de Fibonacci, por ejemplo.

a. 1 entero del fuente se procesa, y se compara con el entero del usuario
b. 2 enteros del usuario se suman, se procesa la suma, y se compara con el entero del fuente
c. 2 enteros del usuario se procesan, se suman los resultados, y se compara con el entero del fuente
d. Las 2 (o 3) opciones más difíciles son de la misma dificultad, así que no se puede marcar ninguna como la más difícil

---

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

a. Todas las instrucciones deben realizar las cuatro etapas del cauce
b. Las etapas ID y MEM 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 3.8 ciclos en el procesador sin segmentar
d. ld f2,a en el procesador ssrDLX requiere 4.6 ciclos

---

17. Dado un programa en un procesador DLX, se puede aplicar la técnica que recibe el nombre de desenrollado de bucle, la cual repercute en:

a. realizar los cálculos de varias iteraciones en diferentes subrutinas
b. reducir el número de instrucciones de salto que se tienen que ejecutar
c. aumentar el número de instrucciones en el código para aumentar la probabilidad de que existan riesgos de control
d. reorganización de las instrucciones para reducir el efecto de las dependencias entre ellas

¿Cuál de las anteriores afirmaciones es cierta?

---

18. 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 256
b. La primera instrucción del programa se ubicará en la posición 0x100
c. Existe una etiqueta de salto denominada .text en la posición 256
d. Todas las anteriores afirmación son incorrectas

---

19. ¿Cómo se almacenaría como palabra de 32 bits el número -128 en un sistema que utilice el criterio del extremo menor (“little endian")?

a. posición 0: FF pos.1:FF pos.2: FF pos.3: 00
b. 0:00 1:FF 2:FF 3:FF
c. 0:00 1:01 2: 00 3:80
d. Ninguna de las anteriores

---

20. En un procesador de la familia 80x86 las posiciones de memoria que representan una variable entera contiene los bytes: F0 FF FF FF. ¿Cuánto vale dicha variable?

a. -16
b. 4043309055
c. 16
d. 4294967280

2º Grado Informática
Estructura de Computadores
9 Septiembre 2013

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 */
int i = 0;
int main() {
    func();
    return 0;
}
```

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

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

a. Error al compilar o enlazar, no se obtiene ejecutable
b. Escribe "0"
c. Escribe "1"
d. A veces escribe "0" y a veces "1"

2. ¿Qué hace gcc -S?

a. Compilar .c → .s (fuente C a fuente ASM)
b. Compilar .s → .o (fuente ASM a objeto)
c. Compilar optimizando tamaño (size), no tiempo
d. 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)?

a. Eso no se puede hacer con gcc
b. gcc -c
c. gcc -o
d. 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?

a. -m32
b. -32
c. -m elf_i386
d. 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?

a. -m32
b. -32
c. -m elf_i386
d. 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...

a. Se puede, repartiendo entre as y ld los modificadores (switches) que corresponda
b. Se puede, repartiendo modificadores entre as y ld, y añadiendo al comando ld el runtime de C
c. Basta usar ld, con los modificadores de gcc que corresponda, y añadiéndole el runtime de C
d. 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:

a. Produce el resultado correcto
b. Fallaría con array={0,1,2,3}
c. Fallaría con array={1,2,4,8}
d. 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:

a. Produce el resultado correcto
b. Fallaría con array={0,1,2,3}
c. Fallaría con array={1,2,4,8}
d. 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).

a. 1 string del fuente se cifra, y se compara con el string del usuario
b. 2 strings del usuario se concatenan, se cifra el resultado y se compara con el string del fuente
c. 2 strings del usuario se cifran, se concatenan los resultados, y se compara con el string del fuente
d. 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. ld 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

2º Grado Informática
Estructura de Computadores
14 Febrero 2014

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. En la convención cdecl estándar para arquitecturas x86 de 32 bits, cuál de las siguientes afirmaciones es cierta:

a. Los parámetros se pasan en pila, de derecha a izquierda; es decir, primero se pasa el último parámetro, después el penúltimo… y por fin el primero.
b. Solamente es necesario salvar el registro EAX
c. Los registros EBX, ESI y EDI son salva - invocante
d. Ninguna de las anteriores es cierta

2. En la convención cdecl estándar para arquitecturas x86 de 32 bits, el resultado de una función se devuelve usualmente en el registro:

a. EBX
b. EBP
c. EAX
d. ESI

3. Utilizando la sentencia asm, las denominadas restricciones que se indican al final de dicha sentencia, involucran a:

a. Solamente las entradas
b. Solamente las salidas
c. Solamente los sobrescritos
d. Ninguna de las anteriores es cierta

4. Suponga la siguiente sentencia asm en un programa:

```
asm(" add (%[a],%[i],4),%r"
:[r] "+r" (result)
:[i] "r" (i),
 [a] "r" (array)  );
```

¿Cuál de las siguientes afirmaciones es correcta?

a. r es una posición de memoria de entrada/salida
b. a es una posición de memoria de entrada
c. i es un registro de entrada
d. el código de retorno de la función asm se fuerza a que esté en la variable result

5. En la realización de la práctica de la bomba digital, una parte del código máquina es el siguiente:

```
0x080486e8 <main+120>:    call   0x8048524 <strncmp>
0x080486ed <main+125>:    test   %eax,%eax
0x080486ef <main+127>:    je  0x80486f6 <main+134>
0x080486f1 <main+129>:    call   0x8048604 <boom>
```

¿Cuál de las siguientes afirmaciones cambiaría el salto condicional por un salto incondicional?

a. set $0x080486ef=0xeb

b. set *(char*)0x080486ef=0xeb
c. set *(char*)0x080486f6=jmp
d. set %0x080486ef=0xeb

6. El punto de entrada de un programa ensamblador en gcc/as Linux x86 se llama

a. main
b. begin
c. _start
d. _init

7. Alguna de las siguientes líneas de código sirve para definir una variable entera llamada tam en gcc/as Linux x86. ¿Cuál?

a. var tam : integer;
b. tam: .int .-msg
c. _int tam = 0
d. int tam;

8. ¿Qué hace gcc –O1?

a. Compilar .c->.o (fuente C a objeto)
b. Compilar .s->.o (fuente ASM a objeto)
c. Ambas (a) y (b), según la extensión de los ficheros que se usen como argumentos
d. Compilar con optimización

9. ¿Cuál de las siguientes afirmaciones es falsa respecto al lenguaje C?

a. 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
b. Al llamar a una subrutina o función se introducen los parámetros en la pila y después se realiza una llamada a la subrutina
c. Los parámetros se introducen en la pila en el orden inverso a como aparecen en la llamada de C, es decir, empezando por el último y acabando por el primero
d. Pasar a una función un puntero a una variable se traduce en introducir en la pila el valor de la dirección de memoria donde está almacenada la variable

10. Para averiguar la paridad de un número se puede usar la operación:

a. AND
b. NAND
c. XOR
d. OR

11. En una bomba como las estudiadas en prácticas, del tipo…

```
0x0804873f <main+207>: call   0x8048504 <scanf>
0x08048744 <main+212>: mov    0x24(%esp),%edx
0x08048748 <main+216>: mov    0x804a044,%eax
0x0804874d <main+221>: cmp    %eax,%edx
0x0804874f <main+223>:  je 0x8048756 <main+230>
0x08048751 <main+225>: call   0x8048604 <boom>
0x08048756 <main+230>:  ...
```

…el código numérico (pin) es...

a. el entero 0x804a044
b. el entero cuya dirección está almacenada en la posición de memoria 0x804a044
c. el entero almacenado a partir de la posición de memoria 0x24(%esp)
d. el entero almacenado a partir de la posición de memoria 0x804a044

12. En una bomba como las estudiadas en prácticas, del tipo…

```
0x080486e8 <main+120>: call   0x8048524 <strncmp>
0x080486ed <main+125>: test   %eax,%eax
0x080486ef <main+127>:  je 0x80486f6 <main+134>
0x080486f1 <main+129>:  call   0x8048604 <boom>
0x080486f6 <main+134>:  ...
```

la contraseña es...

a. el valor que tenga %eax
b. el string almacenado a partir de 0x80486f6
c. el entero almacenado a partir de donde apunta %eax
d. ninguna de las anteriores

13. 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:

a. Produce el resultado correcto
b. Fallaría con array={0,1,2,3}
c. Fallaría con array={1,2,4,8}
d. No es correcta pero el error no se manifiesta en los ejemplos propuestos, o se manifiesta en ambos

---

14. 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:

a. Produce el resultado correcto
b. Fallaría con array={0,1,2,3}
c. Fallaría con array={1,2,4,8}
d. No es correcta pero el error no se manifiesta en los ejemplos propuestos, o se manifiesta en ambos

---

15. ¿Cuál de los siguientes registros tiene que ser salvaguardado (si va a modificarse) dentro de una subrutina según la convención cdecl para IA32?

a. eax
b. ebx
c. ecx
d. edx

---

16. Sea un computador de 32 bits con una memoria caché L1 para datos de 32 KB y líneas de 64 bytes asociativa por conjuntos de 2 vías. Dado el siguiente fragmento de código:

```
int v[262144];
for (i = 0; i < 262144; i += 2)
   v[i] = 9;
```

¿Cuál será la tasa de fallos aproximada que se obtiene en la primera ejecución del bucle anterior?

a. 0 (ningún fallo)
b. 1/2 (mitad aciertos, mitad fallos)
c. 1/8 (un fallo por cada 8 accesos)
d. 1 (todo son fallos)

---

17. El servidor de SWAD tiene dos procesadores Xeon E5540 con 4 núcleos cada uno. Cada procesador tiene 4 caches L1 de instrucciones de 32 KB, 4 caches L1 de datos de 32 KB, 4 caches unificadas L2 de 256 KB y una cache unificada L3 de 8MB. Suponga que un proceso swad, que se ejecuta en un núcleo, tiene que ordenar un vector de estudiantes accediendo repetidamente a sus elementos. Cada elemento es una estructura de datos de un estudiante y tiene un tamaño de 4KB. Si representamos en una gráfica las prestaciones en función del número de

estudiantes a ordenar, ¿para qué límites teóricos en el número de estudiantes se observarán saltos en las prestaciones debidos a accesos a la jerarquía de memoria?

a. 4 / 32 / 512 estudiantes
b. 8 / 64 / 2048 estudiantes
c. 16 / 32 / 64 estudiantes
d. 32 / 256 / 8192 estudiantes

18. En la práctica de cache hemos hecho una gráfica con el código size.cc ¿Qué forma tiene la gráfica que se debe obtener?

a. Forma de U (o V) con un tramo descendente y otro ascendente
b. Forma de U (o V) invertida, con un tramo ascendente y otro descendente
c. Forma de /, una gráfica siempre creciente y sin escalones
d. Una escalera con varios tramos horizontales

19. En la práctica de la cache, el código de line.cc incluye la sentencia

```
for (unsigned long long line=1;
line<=LINE; line<<=1) { … }
```

¿Qué objetivo tiene la expresión line<<=1?

a. salir del bucle si el tamaño de línea se volviera menor o igual que 1 para algún elemento del vector
b. duplicar el tamaño del salto en los accesos al vector respecto a la iteración anterior
c. volver al principio del vector cuando el índice exceda la longitud del vector
d. sacar un uno (1) por el stream line

20. En la práctica de la cache, el código de size.cc accede al vector saltando de 64 en 64. ¿Por qué?

a. Porque cada elemento del vector ocupa 64 bytes
b. Para recorrer el vector más rápidamente
c. Porque el tamaño de cache L1 de todos los procesadores actuales es de 64KB
d. Para anular los aciertos por localidad espacial, esto es, que sólo pueda haber aciertos por localidad temporal

2º Grado Informática
Estructura de Computadores
3 Septiembre 2014

Nombre:	
DNI:	**Grupo:**

Test 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. ¿Cuál es el valor mínimo (más negativo) que puede tomar un entero de 32bits en complemento a dos?

a. -2^{32}
b. $-2^{32} + 1$
c. -2^{31}
d. $-2^{31} + 1$

---

2. Después de ejecutar el siguiente código, ¿qué variables serán igual a 0? (Suponer ints de 32bits y longs de 64bits)

```
unsigned int  a = 0xffffffff;
unsigned int  b = 1;
unsigned int  c = a + b;
unsigned long d = a + b;
unsigned long e =
   (unsigned long)a + b;
```

a. Ninguna
b. c
c. c y d
d. c, d, y e

---

3. En la práctica 2 se pide sumar una lista de 32 enteros SIN signo de 32bits en una plataforma de 32bits sin perder precisión, esto es, evitando acarreos. ¿Cuál es el mínimo valor entero que repetido en toda la lista causaría acarreo con 32bits (sin signo)?

a. **0xfc00 0000**
b. **0xfbff ffff**
c. **0x0800 0000**
d. **0x07ff ffff**

---

4. En la práctica 2 se pide sumar una lista de 32 enteros CON signo de 32bits en una plataforma de 32bits sin perder precisión, esto es, evitando desbordamiento. ¿Cuál es el valor negativo más pequeño (en valor absoluto) que repetido en toda la lista causaría desbordamiento con 32bits (en complemento a 2)?

a. **0xfc00 0000**
b. **0xfbff ffff**
c. **0xf800 0000**
d. **0xf800 0001**

---

5. En la práctica 2 se pide calcular la media y resto de una lista de 32 enteros CON signo de 32bits en una plataforma de 32bits sin perder precisión, esto es, evitando desbordamiento. ¿Qué (media : resto) se debe obtener para una lista rellena a -1 salvo el primer elemento, que valiera -31?

a. (-1 :-30)
b. (-1 :-31)
c. (-2 : 1)
d. (-2 : 2)

---

6. ¿Cuál es el resultado de evaluar la expresión $1110)_2$ ^ $1010)_2$ en lenguaje C?

a. $1111)_2$
b. $1010)_2$
c. $0110)_2$
d. $0100)_2$

7. En la práctica 3 se pide calcular la suma de paridades de una lista de enteros sin signo. Suponer que un estudiante entrega la siguiente versión

```
int paridad5(unsigned* array,
             int len) {
  int i, k, result = 0;
  unsigned x;
  for (i = 0; i < len; i++) {
    x = array[i];
    for (k = 16; k == 1; k /= 2)
      x ^= x >> k;
    result += (x & 0x01);
  }
  return result;
}
```

Esta función:

a. es correcta
b. falla para **`array={0,1,2,3}`**
c. falla para **`array={1,2,3,4}`**
d. no se puede marcar una y sólo una de las opciones anteriores

---

8. Suponer una memoria cache con las siguientes propiedades: Tamaño: 512 bytes. Política de reemplazo: LRU. Estado inicial: vacía (todas las líneas inválidas). Suponer que para la siguiente secuencia de direcciones enviadas a la cache: 0, 2, 4, 8, 16, 32, la tasa de acierto es 0.33. ¿Cuál es el tamaño de bloque de la cache?

a. 4 bytes
b. 8 bytes
c. 16 bytes
d. Ninguno de los anteriores

---

9. Abajo se ofrece el listado de una función para multiplicar matrices C = A x B.

```
void mult_matr(   float A[N][N],
   float B[N][N], float C[N][N]){
   /* Se asume valor inicial C = {0,0...} */
int i,j,k;
for (i=0; i<N; i++)
 for (j=0; j<N; j++)
  for (k=0; k<N; k++)
   C[i][j] += A[i][k] * B[k][j];
}
```

Suponer que:

• El computador tiene una cache de datos de 8 MB, 16-vías, líneas de 64 bytes.
• N es grande, una fila o columna no cabe completa en cache.
• El tamaño de los tipos de datos es como en IA32.
• El compilador optimiza el acceso a C[i][j] en un registro.

Aproximadamente, ¿qué tasa de fallos se podría esperar de esta función para valores grandes de N?

a. 1/16
b. 1/8
c. 1/4
d. 1/2

---

10. Con los mismos supuestos, imaginar que se modifica la última sentencia (el cuerpo anidado) por esta otra

```
C[i][j] += A[i][k] * B[j][k];
```

de manera que se calcule C = A x B' (A por traspuesta de B). Aproximadamente, ¿qué tasa de fallos se podría esperar de esta nueva función para valores grandes de N?

a. 1/16
b. 1/8
c. 1/4
d. 1/2

---

11. La instrucción necesaria para cargar 0x07 en %eax es:

a. **`movb $0x07,%eax`**
b. **`movl $0x07,%eax`**
c. **`movb $0x07,%al`**
d. **`movl $0x07,%ah`**

---

12. La(s) instrucción(es) necesaria(s) para cargar el dividendo 0xa30bf18a en la pareja edx:eax como paso previo a una división sin signo son:

a. **`movl $0xf18a,%eax`**
   `movl $0xa30b,%edx`
b. **`movl $0xa30bf18a,%eax`**
   `xorl %edx,%edx`
c. **`movq $0xa30bf18a,%rax`**
d. **`movl $0xa30bf18a,%eax`**
   `cltd`

---

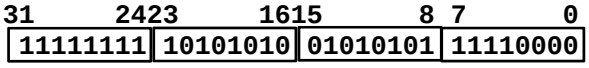
13. ¿Cuál es el efecto de la instrucción siguiente?

```
mov 8(%ebp),%ecx
```

a. Suma 8 al contenido de ebp y almacena la suma en ecx
b. Suma 8 al contenido de ebp, trata la suma como una dirección de memoria y almacena el contenido de esa dirección en ecx
c. Suma 8 al contenido de la posición de memoria cuya dirección está almacenada en ebp, y almacena la suma en ecx
d. Suma los contenidos de ebp y de la dirección de memoria 8 y almacena la suma en ecx

---

14. Si el registro eax contiene el siguiente valor binario:



¿Cuál será el valor de %eax tras ejecutar la instrucción **`xorb %al, %al`**?

a. **`11111111 10101010 01010101 11110000`**
b. **`00000000 10101010 01010101 11110000`**
c. **`00000000 00000000 00000000 00000000`**
d. **`11111111 10101010 01010101 00000000`**

---

15. ¿Qué valor contendrá el registro edx tras ejecutar las dos instrucciones siguientes?

```
movl $-1, %edx
movb $1, %dl
```

a. **`11111111 11111111 11111111 11111111`**
b. **`11111111 11111111 11111111 00000001`**
c. **`00000000 00000000 00000000 00000001`**
d. **`00000001 00000000 00000000 00000000`**

---

16. ¿Qué valor contendrá edx tras ejecutar las siguientes instrucciones?

```
xor %eax, %eax
sub $1, %eax
cltd
idiv %eax
```

a. 0
b. 1
c. -1
d. no puede saberse con los datos del enunciado

---

17. ¿Cuál de los siguientes fragmentos es correcto para comenzar un programa en ensamblador que conste de un solo archivo .s?

a.
```
.text
_start:
```
b.
```
.section .text
.local _start
_start:
```
c.
```
.section .text
.global _start
_start:
```
d.
```
section .text
.start _global
_start:
```

---

18. En un programa en ensamblador queremos crear espacio para una variable entera var inicializada a 1. La línea que hemos de escribir en la sección de datos es:

a. **`.int var 1`**
b. **`var: .int 1`**
c. **`.int: var 1`**
d. **`int var 1`**

---

19. ¿Cuántos operandos acompañan a la instrucción PUSH en IA32?

a. 0
b. 1
c. 2
d. 3

---

20. ¿Para qué se utiliza la función gettimeofday en la práctica de la "bomba digital"?

a. Para cronometrar y poder comparar las duraciones de las distintas soluciones del programa.
b. Para imprimir la hora en la pantalla.
c. Para cifrar la contraseña en función de la hora actual.
d. Para lanzar un error cuando el usuario tarde demasiado tiempo en introducir la contraseña o el PIN.

2º Grado Informática
Estructura de Computadores
2 Febrero 2015

Nombre:	
DNI:	**Grupo:**

Test 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. ¿De qué tipo son los procesadores Intel que usamos en los laboratorios?

a. little-endian
b. big-endian
c. puede ajustarse mediante un bit de control en el registro CR0 que funcionen como little- o big-endian
d. el concepto de endian no es aplicable a estas máquinas, ya que un registro del procesador no cabe en una posición de memoria

2. ¿En qué registro pasa el primer argumento a una función según el estándar _cdecl en una arquitectura IA32?

a. edi
b. esi
c. eax
d. Las anteriores respuestas son erróneas

3. ¿Cuál de las siguientes instrucciones es incorrecta en ensamblador GNU/as IA32?

a. pop %eip
b. pop %ebp
c. mov (%esp),%ebp
d. lea 0x10(%esp),%ebp

4. En un sistema Linux x86-64, ¿cuál de las siguientes variables ocupa más bytes en memoria?

a. char a[7]
b. short b[3]
c. int *c
d. float d

5. Si EBX=0x00002a00 y EDI=0x0000000a, ¿cuál es la dirección efectiva en la instrucción add 0x64(%ebx,%edi,2),%eax?

a. 0x00002a70
b. 0x00002a78
c. 0x00005478
d. 0x000054dc

6. Las instrucciones que asignan la dirección 0x78e00 a un puntero situado en la posición 0xff00 son:

a. mov 0x78e00,%eax; mov %eax,0xff00
b. mov 0x78e00,%eax; lea %eax,0xff00
c. mov $0x78e00,%eax; mov %eax,0xff00
d. mov $0x78e00,%eax; lea %eax,0xff00

7. Si ECX vale 0, la instrucción adc $-1,%ecx

a. Pone CF=1
b. Pone CF=0
c. Cambia CF
d. No cambia CF

8. Si el contenido de ESP es 0xAC00 antes de ejecutar la instrucción push %ebx. ¿Cuál será su contenido tras ejecutarla?

a. 0xABFC

b. 0xABFE
c. 0xAC02
d. 0xAC04

---

9. En IA32, tras dividir 0x640000 (64 bits) entre 0x8000 (32 bits), el resultado será:

a. 0x0 en AX y 0xC8 en DX.
b. 0x0 en EAX y 0xC8 en EDX.
c. 0xC8 en AX y 0xC8 en DX.
d. 0xC8 en EAX y 0x0 en EDX.

---

10. Si los contenidos de ESP y EBP son, respectivamente, 0x0008d040 y 0x00000000 antes de ejecutar una instrucción call, tras ejecutar el ret correspondiente, los contenidos serán:

a. 0x0008d040 y 0x00000000
b. 0x0008d040 y 0x0008d040
c. 0x0008d03c y 0x0008d03c
d. 0x0008d044 y 0x00000000

---

11. La instrucción leave hace exactamente lo mismo que

a. mov %ebp, %esp; pop %ebp
b. pop %eip
c. mov %esp, %ebp; pop %esp
d. ret

---

12. Sean un int*a y un int n. Si el valor de %ecx es a y el valor de %edx es n, ¿cuál de los siguientes fragmentos de ensamblador se corresponde mejor con la sentencia C return a[n]?

a. ret (%ecx,%edx,4)
b. leal (%ecx,%edx,4),%eax; ret
c. mov (%ecx,%edx,4),%eax; ret
d. mov (%ecx,%edx,1),%eax; ret

---

13. ¿Cuál de los siguientes fragmentos es correcto para comenzar un programa en ensamblador que conste de un solo archivo .s?

a.
```
.text
_start:
```
b.
```
.section .text
.global _start
_start:
```
c.
```
.section .text
.local _start
_start:
```
d.
```
section .text
.start _global
_start:
```

---

14. En la práctica 2 se pide sumar una lista de 32 enteros SIN signo de 32bits en una plataforma de 32bits sin perder precisión, esto es, evitando acarreos. ¿Cuál es el mínimo valor entero que repetido en toda la lista causaría acarreo con 32bits (sin signo)?

a. **0x07ff ffff**
b. **0x0800 0000**
c. **0xfbff ffff**
d. **0xfc00 0000**

---

15. 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"
        "setpe %%dl       \n\t"
        "movzx %%dl, %[x] \n\t"
        : [x] "+r" (x)
        :
        : "edx"
        );
    result += x;
  }
  return result;
}
```

Esta función parity6:

a. Produce el resultado correcto
b. No es correcta, fallaría por ejemplo con **array={0,1,2,3}**
c. No es correcta, fallaría por ejemplo con **array={1,2,4,8}**

d. No es correcta pero el error no se manifiesta en los ejemplos propuestos, o se manifiesta en ambos

---

16. Suponga la siguiente sentencia asm en un programa:

asm(“ add (%[a],%[i],4),%r”

:[r] “+r” (result)

:[i] “r” (i),

[a] “r” (array));

¿Cuál de las siguientes afirmaciones es incorrecta?

a. r es un registro de entrada/salida
b. a es un registro de entrada
c. i es un registro de salida
d. se desea que el valor calculado por la instrucción ensamblador quede almacenado en la variable C result

---

17. En una bomba como las estudiadas en prácticas, del tipo…

```
0x0804873f <main+207>:  call   0x8048504 <scanf>
0x08048744 <main+212>: mov    0x24(%esp),%edx
0x08048748 <main+216>: mov    0x804a044,%eax
0x0804874d <main+221>: cmp    %eax,%edx
0x0804874f <main+223>:  je   0x8048756<main+230>
0x08048751 <main+225>: call   0x8048604 <boom>
0x08048756 <main+230>:  ...
```

la contraseña es...

a. el entero 0x804a044
b. el entero almacenado a partir de la posición de memoria 0x804a044
c. el string almacenado a partir de la posición de memoria 0x24(%esp)
d. ninguna de las anteriores

---

18. En la práctica de la cache, el código de line.cc incluye la sentencia

```
for (unsigned long long line=1;
line<=LINE; line<<=1) { … }
```

¿Qué objetivo tiene la expresión line<<=1?

a. salir del bucle si el tamaño de línea se volviera menor o igual que 1 para algún elemento del vector
b. duplicar el tamaño del salto en los accesos al vector respecto a la iteración anterior
c. volver al principio del vector cuando el índice exceda la longitud del vector
d. sacar un uno (1) por el stream line

---

19. En la práctica de la cache, el código de size.cc accede al vector saltando de 64 en 64. ¿Por qué?

a. Porque cada elemento del vector ocupa 64 bytes
b. Para recorrer el vector más rápidamente
c. Porque el tamaño de cache L1 de todos los procesadores actuales es de 64KB
d. Para anular los aciertos por localidad espacial, esto es, que sólo pueda haber aciertos por localidad temporal

---

20. En el programa size de la práctica de la cache, si el primer escalón pasa de tiempo=2 para un tamaño de vector menor que 32KB a tiempo=8 para un tamaño mayor que 32KB, podemos asegurar que:

a. La cache L1 es cuatro veces más rápida que la cache L2.
b. La cache L1 es seis veces más rápida que la cache L2.
c. La cache L1 es al menos cuatro veces más de rápida que la cache L2.
d. La cache L1 es como mucho cuatro veces más rápida que la cache L2.

2º Grado Informática
Estructura de Computadores
3 de septiembre de 2015

Nombre:	
DNI:	**Grupo:**

Test 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. En contraposición a un ejecutable Linux ELF, un fichero objeto (obtenido con gcc -c)

a. no contiene tablas de símbolos, que sólo se añaden al ejecutable tras enlazar
b. ubica el código (y los datos) a partir de la posición 0x0, las direcciones definitivas sólo se calculan tras enlazar
c. contiene tablas de símbolos, pero sólo para los símbolos locales; p.ej., sólo al enlazar un programa que llame a printf se añadirá una entrada a la tabla con la dirección de printf
d. no reserva espacio para las direcciones de objetos desconocidos; p.ej., sólo al enlazar un CALL a printf se insertan los 4B de la dirección de printf entre el codop de CALL y el de la siguiente instrucción máquina

2. Tras ejecutar las tres instrucciones que se muestran desensambladas a continuación, el registro EAX toma el valor

```
08048074 <_start>:
8048074: be 74 80 04 08 mov $_start, %esi
8048079: 46             inc  %esi
804807a: 8b 06          mov (%esi), %eax
```

a. 0x08048074
b. 0x08048075
c. 0x08048079
d. 0x0804807a

3. Si %edx contiene 0xf000 y %ecx contiene 0x0100, el direccionamiento 0x80(%ecx,%edx,2) se refiere a la posición

a. 0xf182
b. 0xf280
c. 0x1e180
d. Ninguna de las respuestas anteriores es correcta

4. ¿Cuál de las siguientes secuencias de instrucciones multiplica %eax por 10?

a. leal(%eax,%eax,4), %eax
   sall $2, %eax
b. imull $0x10, %eax
c. addl %eax, %eax
   shll $5, %eax
d. Varias o ninguna de las respuestas anteriores son correctas, no se puede marcar una y sólo una

5. ¿Cuál de las siguientes secuencias de instrucciones calcula a=b-a, suponiendo que %eax es a y %ebx es b?

a. subl %eax, %ebx
b. subl %ebx, %eax
c. notl %eax
   addl %ebx, %eax
d. Varias o ninguna de las respuestas anteriores son correctas, no se puede marcar una y sólo una

6. Para desplazar %eax a la derecha un número variable de posiciones ≤ 32, indicado en %ebx, se puede hacer

a. sar %ebx, %eax

b. sar %bl, %eax

c. mov %ebx, %ecx
sar %cl, %eax

d. No se puede, sar sólo admite un número fijo de posiciones (debe ser un valor inmediato)

---

7. Indicar cuál es la dirección de salto (en qué dirección empieza la subrutina <main>) para esta instrucción call

```
0804854e: e8 3d 06 00 00 call <main>
08048553: 50             pushl %eax
```

a. 08048b90

b. 08048b8b

c. 450a854e

d. No se puede deducir de la información proporcionada, faltan datos

---

8. El ajuste de marco de pila que gcc (Linux/IA-32) prepara para todas las funciones consiste en las instrucciones

a. movl %ebp, %esp
pushl %ebp

b. movl %esp, %ebp
pushl %esp

c. pushl %ebp
movl %esp, %ebp

d. pushl %esp
movl %ebp, %esp

---

9. Al traducir de lenguaje C a ensamblador, gcc en máquinas Linux/IA-32 almacena (reserva espacio para) una variable "var" local a una función "fun" en una dirección de memoria referenciable (en lenguaje ensamblador) como

a. var (el nombre de la variable representa su posición de memoria)

b. -k(%ebp), siendo k un número constante positivo relativamente pequeño

c. k(%ebp), siendo k un número constante positivo relativamente pequeño

d. k(%esp), siendo k un número constante positivo relativamente pequeño

---

10. En x86_64 se pueden referenciar los registros

a. %rax, %eax, %ax, %ah, %al

b. %rsi, %esi, %si, %sih, %sil

c. %r8, %r8d, %r8w, %r8l

d. %r12q, %r12d, %r12w, %r12l

---

11. Comparando las convenciones de llamada de gcc Linux IA-32 con x86_64 respecto a registros

a. En IA-32 %ebx es salva-invocante, pero en x86_64 %rbx es salva-invocado

b. En IA-32 %ecx es salva-invocante, y en x86_64 %rcx es salva-invocante también

c. En IA-32 %esi es salva-invocado, y en x86_64 %rsi es salva-invocado también

d. En IA-32 %ebp es especial (marco de pila), y en x86_64 %rbp también

---

12. Si declaramos **`int val[5]={1,5,2,1,3};`** entonces

a. &val[2] es de tipo int* y vale lo mismo que val+8

b. val+4 es de tipo int* y se cumple que *(val+4)==5

c. val+1 es de tipo int y vale 2

d. val[5] es de tipo int y vale 3

---

13. Al traducir la sentencia C **`r->i = val;`** gcc genera el código ASM **`movl %edx, 12(%eax)`**. Se deduce que

a. r es un puntero que apunta a la posición de memoria 12

b. el desplazamiento de i en *r es 12

c. i es un entero que vale 12

d. val es un entero que vale 12

---

14. Una función C llamada **`get_el()`** genera el siguiente código ensamblador. Se puede adivinar que

```
movl 8(%ebp), %eax
leal (%eax,%eax,4), %eax
addl 12(%ebp), %eax
movl var(,%eax,4), %eax
```

a. var es un array multi-nivel (punteros a enteros) de cuatro filas
b. var es un array multi-nivel pero no se pueden adivinar las dimensiones
c. var es un array bidimensional de enteros, no se pueden adivinar dimensiones
d. var es un array bidimensional de enteros, con cinco columnas

---

15. En la práctica "suma" se pide sumar una lista de 32 enteros **con** signo de 32bits en una plataforma de 32bits sin perder precisión, esto es, evitando *overflow*. ¿Cuál es el menor valor positivo que repetido en toda la lista causaría *overflow* con 32bits?

a. 0x0400 0000
b. 0x0800 0000
c. 0x4000 0000
d. 0x8000 0000

---

16. En la práctica "suma" se pide sumar una lista de 32 enteros **con** signo de 32bits en una plataforma de 32bits sin perder precisión, esto es, evitando *overflow*. ¿Cuál es el mayor valor negativo (menor en valor absoluto) que repetido en toda la lista causaría *overflow* con 32bits?

a. 0xffff ffff
b. 0xfc00 0000
c. 0xfbff ffff
d. 0xf000 0000

---

17. La práctica "*parity*" debía calcular la suma de paridades impar (XOR de todos los bits) de los elementos de un array. La siguiente función contiene un único error (en realidad se han editado 2 líneas de código sobre la versión correcta) pero produce resultado **correcto** cuando se usa como test el array

```
int   parity4(unsigned* array,
                int len){
  int i;
  unsigned x;
  int val=0, result=0;

  for (i=0; i<len; i++){
    x = array[i];
    asm("\n"
"ini:                   \n\t"
    "xor %[x], %[v]   \n\t"
    "shr %[x]         \n\t"
    "jnz ini          \n\t"
  : [v]"+r" (val)
  : [x] "r" (x)
    );
    result += val & 0x1;
  }
  return result;
}
```

a. array={0, 1, 2, 3}
b. array={1, 2, 4, 8}
c. array={1, 16, 256, 1024}
d. array={5, 4, 3, 2}

---

18. En la práctica de la cache, el código de "line.cc" incluye la sentencia

```
for (unsigned long long line=1;
line<=LINE; line<<=1) { … }
```

¿Qué objetivo tiene la expresión line<<=1?

a. salir del bucle si el tamaño de línea se volviera menor o igual que 1 para algún elemento del vector
b. duplicar el tamaño del salto en los accesos al vector respecto a la iteración anterior
c. volver al principio del vector cuando el índice exceda la longitud del vector
d. sacar un uno (1) por el stream line

---

19. En la práctica de la cache, el código de "size.cc" accede al vector saltando de 64 en 64. ¿Por qué?

a. Porque cada elemento del vector ocupa 64 bytes
b. Para recorrer el vector más rápidamente
c. Porque el tamaño de cache L1 de todos los procesadores actuales es de 64KB
d. Para anular los aciertos por localidad espacial, esto es, que sólo pueda haber aciertos por localidad temporal

---

20. En el programa "size.cc" de la práctica de la cache, si el primer escalón pasa de tiempo=2 para un tamaño de vector

menor que 32KB a tiempo=8 para un tamaño mayor que 32KB, podemos asegurar que:

a. La cache L1 es cuatro veces más rápida que la cache L2.
b. La cache L1 es seis veces más rápida que la cache L2.
c. La cache L1 es al menos cuatro veces más de rápida que la cache L2.
d. La cache L1 es como mucho cuatro veces más rápida que la cache L2.

2º Grado Informática
Estructura de Computadores
8 Febrero 2016

Nombre:	
DNI:	**Grupo:**

Test 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. La dirección efectiva del primer parámetro de llamada a una función suele calcularse desde el código de la función como:

a. EBP+8
b. EBP-8
c. EBP+4
d. EBP-4

2. El comienzo de un procedimiento que siga la convención cdecl es:

a. mov %ebp,%esp; push %ebp
b. mov %esp,%ebp; push %ebp
c. push %ebp; mov %ebp,%esp
d. push %ebp; mov %esp,%ebp

3. Considere una función C declarada así:

```
void fun4arg (int a,int b,int c,int d);
```

Suponiendo que fun4arg se ha compilado para una máquina x86 IA-32 con enteros de 4 bytes, ¿cuál sería la dirección del argumento b relativa a %ebp, en el marco de pila de fun4arg?

a. %ebp + 8
b. %ebp + 12
c. %ebp + 16
d. %ebp + 20

4. ¿Cuál de las siguientes afirmaciones sobre las caches es ***FALSA***?

a. Casi ningún procesador actual tiene memoria cache L2
b. Las direcciones a las que accede un programa no son completamente aleatorias, sino que se rigen por ciertos patrones de localidad
c. Un procesador actual tiene varias cachés de nivel 1
d. La caché de nivel 3 no contiene toda la memoria que maneja el programa

5. En un sistema Linux x86-64, ¿cuál de las siguientes variables ocupa más bytes en memoria?

a. char a[7]
b. short b[3]
c. int *c
d. float d

6. En la práctica "suma" se pide sumar una lista de 32 enteros SIN signo de 32bits en una plataforma de 32bits sin perder precisión, esto es, evitando acarreos. ¿Cuál es el mínimo valor entero que repetido en toda la lista causaría acarreo con 32bits (sin signo)?

a. `0xfc00 0000`
b. `0xfbff ffff`
c. `0x0800 0000`
d. `0x07ff ffff`

7. En la práctica "suma" se pide sumar una lista de 32 enteros CON signo de 32bits en una plataforma de 32bits sin perder precisión, esto es, evitando

desbordamiento. ¿Cuál es el valor negativo más pequeño (en valor absoluto) que repetido en toda la lista causaría desbordamiento con 32bits (en complemento a 2)?

a. `0xfc00 0000`
b. `0xfbff ffff`
c. `0xf800 0000`
d. `0xf800 0001`

---

8. ¿Qué valor contendrá edx tras ejecutar las siguientes instrucciones?

```
xor %eax, %eax
sub $1, %eax
cltd
idiv %eax
```

a. 0
b. 1
c. -1
d. No puede saberse con los datos del enunciado

---

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

```
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:

a. produce el resultado correcto
b. fallaría con **`array={0,1,2,3}`**
c. fallaría con **`array={1,2,4,8}`**
d. no es correcta pero el error no se manifiesta en los ejemplos propuestos, o se manifiesta en ambos

---

10. 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:

a. produce el resultado correcto
b. fallaría con **`array={0,1,2,3}`**
c. fallaría con **`array={1,2,4,8}`**
d. no es correcta pero el error no se manifiesta en los ejemplos propuestos, o se manifiesta en ambos

---

11. En la práctica "paridad" se pide calcular la suma de paridades de una lista de enteros sin signo. Suponer que un estudiante entrega la siguiente versión:

```
int paridad5(unsigned* array,
             int len) {
  int i, k, result = 0;
  unsigned x;
  for (i = 0; i < len; i++) {
    x = array[i];
    for (k = 16; k == 1; k /= 2)
      x ^= x >> k;
    result += (x & 0x01);
  }
  return result;
}
```

Esta función:

a. es correcta

b. falla para `array={0,1,2,3}`

c. falla para `array={1,2,3,4}`

d. no se puede marcar una y sólo una de las opciones anteriores

---

12. Utilizando la sentencia asm, las denominadas restricciones que se indican al final de dicha sentencia, involucran a:

a. solamente las entradas

b. solamente las salidas

c. solamente los sobrescritos

d. Ninguna de las anteriores es cierta

---

13. En la realización de la práctica de la bomba digital, una parte del código máquina es el siguiente:

```
0x080486e8 <main+120>: call 0x8048524 <strncmp>
0x080486ed <main+125>: test %eax,%eax
0x080486ef <main+127>:  je   0x80486f6<main+134>
0x080486f1 <main+129>: call 0x8048604  <boom>
```

¿Cuál de los siguientes comandos cambiaría el salto condicional por un salto incondicional?

a. set $0x080486ef=0xeb

b. set *(char*)0x080486ef=0xeb

c. set *(char*)0x080486f6=jmp

d. set %0x080486ef=0xeb

---

14. En una bomba como las estudiadas en prácticas, del tipo…

```
0x0804873f <main+207>:  call     0x8048504 <scanf>
0x08048744 <main+212>: mov    0x24(%esp),%edx
0x08048748 <main+216>: mov    0x804a044,%eax
0x0804874d <main+221>: cmp    %eax,%edx
0x0804874f <main+223>:  je  0x8048756<main+230>
0x08048751 <main+225>: call   0x8048604 <boom>
0x08048756 <main+230>:  ...
```

la contraseña es...

a. el entero 0x804a044

b. el entero almacenado a partir de la posición de memoria 0x804a044

c. el string almacenado a partir de la posición de memoria 0x24(%esp)

d. ninguna de las anteriores

---

15. En una bomba como las estudiadas en prácticas, del tipo…

```
0x080486e8 <main+120>: call 0x8048524 <strncmp>
0x080486ed <main+125>: test   %eax,%eax
0x080486ef <main+127>:  je    0x80486f6 <main+134>
0x080486f1 <main+129>:  call  0x8048604 <boom>
0x080486f6 <main+134>:  ...
```

la contraseña es...

a. el valor que tenga %eax

b. el string almacenado a partir de donde apunta %eax

c. el entero almacenado a partir de donde apunta %eax

d. ninguna de las anteriores

---

16. El servidor de SWAD tiene dos procesadores Xeon E5540 con 4 núcleos cada uno. Cada procesador tiene 4 caches L1 de instrucciones de 32 KB, 4 caches L1 de datos de 32 KB, 4 caches unificadas L2 de 256 KB y una cache unificada L3 de 8MB. Suponga que un proceso swad, que se ejecuta en un núcleo, tiene que ordenar un vector de estudiantes accediendo repetidamente a sus elementos. Cada elemento es una estructura de datos de un estudiante y tiene un tamaño de 4KB. Si representamos en una gráfica las prestaciones en función del número de estudiantes a ordenar, ¿para qué límites teóricos en el número de estudiantes se observarán saltos en las prestaciones debidos a accesos a la jerarquía de memoria?

a. 4 / 32 / 512 estudiantes

b. 8 / 64 / 2048 estudiantes

c. 16 / 32 / 64 estudiantes

d. 32 / 256 / 8192 estudiantes

---

17. En la práctica de la cache, el código de line.cc incluye la sentencia

```
for (unsigned line=1;line<=MAXLINE;
                line<<=1) { … }
```

¿Qué objetivo tiene la expresión line<<=1?

a. Salir del bucle si el tamaño de línea se volviera menor o igual que 1 para algún elemento del vector
b. Duplicar el tamaño del salto en los accesos al vector respecto a la iteración anterior
c. Volver al principio del vector cuando el índice exceda la longitud del vector
d. Sacar un uno (1) por el stream line

---

18. Sea un computador de 32 bits con una memoria caché L1 para datos de 32 KB y líneas de 64 bytes asociativa por conjuntos de 2 vías. Dado el siguiente fragmento de código:

```
int v[262144];
for (i = 0; i < 262144; i += 2)
   v[i] = 9;
```

¿Cuál será la tasa de fallos aproximada que se obtiene en la primera ejecución del bucle anterior?

a. 0 (ningún fallo)
b. 1/2 (mitad aciertos, mitad fallos)
c. 1/8 (un fallo por cada 8 accesos)
d. 1 (todo son fallos)

---

19. Abajo se ofrece el listado de una función para multiplicar matrices C = A x B.

```
void mult_matr(   float A[N][N],
   float B[N][N], float C[N][N]){
   /* Se asume valor inicial C = {0,0...} */
int i,j,k;
for (i=0; i<N; i++)
 for (j=0; j<N; j++)
  for (k=0; k<N; k++)
   C[i][j] += A[i][k] * B[k][j];
}
```

Suponer que:

• El computador tiene una cache de datos de 8 MB, 16-vías, líneas de 64 bytes.
• N es grande, una fila o columna no cabe completa en cache.
• El tamaño de los tipos de datos es como en IA32.
• El compilador optimiza el acceso a C[i][j] en un registro.

Aproximadamente, ¿qué tasa de fallos se podría esperar de esta función para valores grandes de N?

a. 1/16
b. 1/8
c. 1/4
d. 1/2

---

20. Con los mismos supuestos, imaginar que se modifica la última sentencia (el cuerpo anidado) por esta otra

```
C[i][j] += A[i][k] * B[j][k];
```

de manera que se calcule C = A x B' (A por traspuesta de B). Aproximadamente, ¿qué tasa de fallos se podría esperar de esta nueva función para valores grandes de N?

a. 1/16
b. 1/8
c. 1/4
d. 1/2

2º Grado Informática
Estructura de Computadores
7 Septiembre 2016

Nombre:	
DNI:	**Grupo:**

Test 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. El switch de gcc para que únicamente compile de lenguaje C a ensamblador, y no realice ningún paso adicional (ensamblar, enlazar, etc), es...

a. -c
b. -S
c. -o
d. -g

2. Los switches --32 y --64 para trabajar en 32bit/64bit corresponden a la herramienta...

a. gcc
b. as
c. ld
d. nm

3. El switch -l para indicar librerías ***NO*** funciona con la herramienta...

a. gcc
b. as
c. ld
d. no se puede marcar una y solo una de las anteriores

4. ¿Cuál de las siguientes no es una sección de un fichero ELF?

a. .text
b. .static
c. .data
d. .bss

5. ¿Cuál de los siguientes contenidos no está incluido en un fichero ELF ejecutable?

a. código máquina
b. variables globales
c. pila del usuario
d. tabla de símbolos

6. En la práctica "media" se programa la suma de una lista de 32 enteros de 4 B para producir un resultado de 8 B, primero sin signo y luego con signo. Si la lista se rellena con el valor que se indica a continuación, ¿en qué caso ambos programas producen el mismo resultado?

a. `0x1111 1111`
b. `0x9999 9999`
c. `0xAAAA AAAA`
d. `0xFFFF FFFF`

7. En la práctica "media" se programa la suma de una lista de 32 enteros de 4 B para producir un resultado de 8 B, primero sin signo y luego con signo. Si la lista se rellena con el valor 0x0400 0000, ¿en qué se diferencian los resultados de ambos programas?

a. no se diferencian
b. en uno ocupa 32 bits, en otro 64 bits
c. en uno se interpreta como negativo, en otro como positivo
d. en uno los 32 bits superiores son 0xFFFF FFFF, en el otro no

8. En la práctica "media" se suma una lista de 32 enteros de 4 B con signo para producir una media y un resto usando la instrucción IDIV. ¿Cuál de las siguientes afirmaciones es falsa?

a. IDIV produce el mismo cociente que el operador / en lenguaje C
b. IDIV produce el mismo resto que el operador % en lenguaje C
c. La media se redondea al entero más próximo
d. El resto siempre tiene el mismo signo que la suma

---

9. En la práctica "media" un estudiante usa el siguiente bucle para acumular la suma en EBP:EDI antes de calcular la media y el resto

```
bucle:
        mov (%ebx,%esi,4), %eax
        cltd
        add %eax, %edi
        adc %edx, %ebp
        jnc nocarry
        inc %edx
nocarry:
        inc %esi
        cmp  %esi,%ecx
        jne bucle
```

Estando bien programado todo lo demás, este código

a. produce siempre el resultado correcto
b. fallaría con lista: .int 0,1,2,3
c. fallaría con lista: .int -1,-2,-4,-8
d. no siempre produce el resultado correcto, pero el error no se manifiesta en los ejemplos propuestos, o se manifiesta en ambos

---

10. Alguno de los siguientes no es un nombre de registro en una máquina IA-32 en modo 32 bits

a. ebp
b. ax
c. dh
d. sil

---

11. Alguno de los siguientes no es un nombre de registro en una máquina x86-64 en modo 64 bits

a. r8d
b. r12w
c. sih
d. spl

---

12. Para comprobar si el entero almacenado en EAX es cero (y posiblemente saltar a continuación usando JZ/JNZ), gcc genera el código

a. cmp %eax, $0
b. test %eax
c. cmp %eax
d. test %eax, %eax

---

13. 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 parity3:

```
int parity3(unsigned* array,
                        int len){
  int  i,res=0,val;
  unsigned x;
  for (i=0; i<len; i++){
    x=array[i];
    val=0;
    do {
      val += x;
      x >>= 1;
    } while (x);
    val &= 0x1;
    res+=val;
    }
  return res;
}
```

Esta función parity3:

a. produce siempre el resultado correcto
b. fallaría con array={0,1,2,3}
c. fallaría con array={1,2,4,8}
d. no siempre produce el resultado correcto, pero el error no se manifiesta en los ejemplos propuestos, o se manifiesta en ambos

14. Un estudiante entrega la siguiente versión de parity4:

```
int parity4(unsigned* array,
                        int len){
  int val,i,res=0;
  unsigned x;
  for (i=0; i<len; i++){
    x=array[i];
    val=0;
    asm("\n"
"ini3:              \n\t"
    "xor  %[x],%[v] \n\t"
    "shr  %[x]      \n\t"
    "test %[x], %[x]\n\t"
    "jne   ini3     \n\t"
    :[v]"+r" (val)
    :[x] "r" (x)
    );
    val = val & 0x1;
    res+=val;
  }
  return res;
}
```

Esta función parity4:

a. produce siempre el resultado correcto
b. fallaría con array={0,1,2,3}
c. fallaría con array={1,2,4,8}
d. no siempre produce el resultado correcto, pero el error no se manifiesta en los ejemplos propuestos, o se manifiesta en ambos

---

15. La sentencia asm() del listado anterior tiene las siguientes restricciones

a. ninguna
b. arquitectura de 32 bits
c. dos entradas y una salida
d. un registro y dos sobrescritos (clobber)

---

16. Un estudiante entrega la siguiente versión de parity5:

```
int parity5(unsigned* array,
                        int len){
  int  i,j,res=0;
  unsigned x;
  for (i=0; i<len; i++){
    x=array[i];
    for (j=sizeof(unsigned)*4;
         j>0; j=j/2){
      x^=x>>j;
    }
    x = x & 0x1;
    res+=x;
  }
  return res;
}
```

Esta función parity5:

a. produce siempre el resultado correcto
b. fallaría con array={0,1,2,3}
c. fallaría con array={1,2,4,8}
d. no siempre produce el resultado correcto, pero el error no se manifiesta en los ejemplos propuestos, o se manifiesta en ambos

---

17. Un estudiante entrega la siguiente versión de parity6:

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

Esta función parity6:

a. produce siempre el resultado correcto
b. fallaría con array={0,1,2,3}
c. fallaría con array={1,2,4,8}

d. no siempre produce el resultado correcto, pero el error no se manifiesta en los ejemplos propuestos, o se manifiesta en ambos

---

18. La sentencia asm() del listado anterior tiene las siguientes restricciones

a. ninguna
b. arquitectura de 32 bits
c. dos entradas y una salida
d. un registro y dos sobrescritos (clobber)

---

19. En el programa "size" de la práctica de la cache, si el primer escalón pasa de tiempo = 1 para todos los tamaños de vector menores o iguales que 32 KB a tiempo = 3 para los tamaños 64 KB y 128 KB, podemos asegurar que:

a. la cache L1 es al menos tres veces más rápida que la cache L2.
b. la cache L1 es como mucho tres veces más rápida que la cache L2.
c. la cache L2 es al menos el doble de rápida que la memoria principal.
d. la cache L2 es como mucho el doble de rápida que la memoria principal.

---

20. El código del programa "size" de la práctica de la cache accede al vector saltando...

a. de byte en byte.
b. de 64 en 64 bytes.
c. de 1 KB en 1 KB.
d. de 64 KB en 64 KB.

2º Grado Informática
Estructura de Computadores
23 de enero de 2017

Nombre:	
DNI:	**Grupo:**

Test de Prácticas (4.0p)

Todas las preguntas son de elección simple sobre 4 alternativas.
Cada respuesta vale 0.2p si es correcta, 0 si está en blanco o claramente tachada, -0.0$\widehat{6}$p si es errónea.
Anotar las respuestas (a, b, c ó 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. ¿Qué hace gcc -O0?

a. Compilar sin optimización
b. Compilar .c→.o (fuente C a objeto)
c. Compilar .s→.o (fuente ASM a objeto)
d. Compilar .c→.s (C→ASM sin generar objeto)

2. ¿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. -march32
d. -march64

3. La etiqueta del punto de entrada a un programa ensamblador en el entorno de las prácticas 1 a 4 (GNU/as Linux x86) es:

a. _main
b. .L0
c. _start
d. _init

4. La siguiente línea en la sección de datos de un programa en ensamblador de IA32

```
result: .int 0,0
```

a. Reserva espacio para un único entero, inicializado a 0,0
b. Reserva espacio para un entero, inicializado a 0, seguido de un dato de tamaño indefinido, también inicializado a 0
c. Reserva espacio para dos enteros, inicializados ambos a 0
d. Reserva espacio para un único entero, inicializado a 0, en la posición de memoria 0

5. El volcado mostrado abajo se ha obtenido con el comando...

```
00000000 <main>:
 0: 55              push %ebp
 1: 89 e5           mov  %esp,%ebp
 3: 83 e4 f0        and  $-16,%esp
 6: 83 ec 10        sub  $0x10,%esp
 9: c7 44 24 04 03 movl $3, 4(%esp)
 e: 00 00 00
11: c7 04 24 01 00 movl $0x1,(%esp)
16: 00 00
18: e8 fc ff ff ff call <main+0x19>
1d: c9             leave
1e: c3             ret
```

a. objdump -h p
b. objdump -d p
c. objdump -d p2.o
d. objdump -t p2.o

6. En la práctica "media" se desea invocar desde lenguaje ensamblador la función printf() de libC. Eso implica que este programa, como todo programa que use libC,

a. es obligatorio que contenga main (y entonces es más cómodo usar gcc para enlazar)
b. es obligatorio enlazarlo usando gcc (y entonces es más cómodo que contenga main)
c. es ventajoso para ensamblarlo que contenga main, y entonces es conveniente enlazarlo usando gcc (aunque ambas cosas son opcionales)

d. es ventajoso para enlazarlo usar gcc, y entonces es conveniente que contenga main (aunque ambas cosas son opcionales)

---

7. En la práctica "media" se pide sumar una lista de 32 enteros SIN signo de 32 bits en una plataforma de 32 bits sin perder precisión, esto es, evitando perder acarreos. Un estudiante entrega la siguiente versión

```
# $lista en EBX, longlista en ECX
suma:
   mov $0, %eax
   mov $0, %edx
   mov $0, %esi
bucle:
   add (%ebx,%edx,4), %eax
   jnc  seguir
   inc  %edx
seguir:
   inc  %esi
   cmp  %esi,%ecx
   jne bucle
   ret
```

Esta función presenta una única diferencia frente a la solución recomendada en clase, relativa al indexado en el array.

Esta función suma:

a. Produce siempre el resultado correcto
b. Fallaría con **lista: .int 1,1,1,1, 1,1,1,1, ...**
c. Fallaría con **lista: .int 1,2,3,4, 1,2,3,4, ...**
d. No es correcta pero el error no se manifiesta en los ejemplos propuestos, o se manifiesta en ambos

---

8. En la práctica "media" se pide sumar una lista de 32 enteros SIN signo de 32 bits en una plataforma de 32 bits sin perder precisión, esto es, evitando perder acarreos. De entre los siguientes, ¿cuál es el mínimo valor entero que repetido en toda la lista causaría acarreo con 32 bits (sin signo)? Se usa notación decimal y espacios como separadores de millares/millones/etc.

a. 10 000 000
b. 100 000 000
c. 1 000 000 000
d. 10 000 000 000

---

9. En la práctica "media" se pide sumar una lista de 32 enteros CON signo de 32 bits en una plataforma de 32 bits sin perder precisión, esto es, evitando desbordamiento. De entre los siguientes, ¿cuál es el valor negativo más pequeño (en valor absoluto) que repetido en toda la lista causaría desbordamiento con 32 bits (en complemento a 2)? Se usa notación decimal y espacios como separadores de millares/millones/etc.

a. -10 000 000
b. -100 000 000
c. -1 000 000 000
d. -10 000 000 000

---

10. ¿Cuál es el popcount (peso Hamming, nº de bits activados) del número 19?

a. 2
b. 3
c. 4
d. 5

---

11. La práctica "popcount" debía calcular la suma de bits (peso Hamming) de los elementos de un array. Un estudiante entrega la siguiente versión de popcount3:

```
int popcount3(unsigned* array,
              int len){
    int i, res = 0;
    unsigned x;
    for( i = 0; i < len; i++ ) {
      x = array[i];
      asm("ini3:        \n"
       "shr %[x]        \n"
       "adc $0, %[r]    \n"
       "add $0, %[x]    \n"
       "jne ini3        \n"
      : [r] "+r" (res)
      : [x] "r" (x) );
    }
    return res;
}
```

Esta función sólo tiene una diferencia con la versión "oficial" recomendada en clase. En concreto, una instrucción máquina en la sección **asm()** es distinta.

Esta función popcount3:

a. produce siempre el resultado correcto
b. fallaría con **array={0,1,2,3}**
c. fallaría con **array={1,2,4,8}**
d. no es correcta pero el error no se manifiesta en los ejemplos propuestos, o se manifiesta en ambos

---

12. La práctica "popcount" debía calcular la suma de bits (peso Hamming) de los elementos de un array. Un estudiante entrega la siguiente versión de popcount3:

```
int popcount3(int* array, int len){
    long val = 0;
    int i;
    unsigned x;
    for (i=0; i<len; i++){
      x= array[i];
      do{
          val += x & 0x1;
          x >>= 1;
      } while (x);
      val += (val >> 16);
      val += (val >> 8);
    }
    return val & 0xFF;
}
```

Esta función es una mezcla inexplicada de las versiones "oficiales" de **popcount2** y **popcount4**, incluyendo diferencias en 2 tipos de datos, la ausencia de la variable **res** y la diferente posición de la máscara **0xFF**.

Esta función popcount3:

a. produce siempre el resultado correcto
b. fallaría con **array={0,1,2,3}**
c. fallaría con **array={1,2,4,8}**
d. no es correcta pero el error no se manifiesta en los ejemplos propuestos, o se manifiesta en ambos

---

13. La práctica "popcount" debía calcular la suma de bits (peso Hamming) de los elementos de un array. Un estudiante entrega la siguiente versión de popcount4:

```
int popcount4(unsigned* array,
              int len){
  int i,j;
  unsigned x;
  int result = 0;
  for(i=0;i<len;i++){
    x=array[i];
    for(j=0;j<8;j++){
      result += x & 0x01010101;
      x>>=1;
    }
  }
  result += (result >> 16);
  result += (result >> 8);
  return result & 0xFF;
}
```

Esta función presenta varias diferencias con la versión "oficial" recomendada en clase, incluyendo la ausencia de una variable auxiliar **val** y la diferente posición de los desplazamientos >> y máscara **0xFF**.

Esta función popcount4:

a. produce siempre el resultado correcto
b. fallaría con **array={0,1,2,3}**
c. fallaría con **array={1,2,4,8}**
d. no es correcta pero el error no se manifiesta en los ejemplos propuestos, o se manifiesta en ambos

---

14. La práctica "parity" 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 parity5:

```
int parity5(unsigned * array,
            int len){
  int i,j, result = 0;
  unsigned x;
  for(i = 0; i<len; i++){
    x=array[i];
    for(j=1; j<8*sizeof(int); j*=2)
      x ^= x >> j;
    result += x & 0x1;
  }
  return result;
}
```

Esta función sólo se diferencia de la versión "oficial" recomendada en clase, en las condiciones del bucle **for** interno.

Esta función parity5:

a. Produce siempre el resultado correcto
b. Fallaría con **array={0,1,2,3}**
c. Fallaría con **array={1,2,4,8}**
d. No es correcta pero el error no se manifiesta en los ejemplos propuestos, o se manifiesta en ambos

---

15. La función **gettimeofday()** en la práctica de la "bomba digital" se utiliza para

a. Para comparar las duraciones de las distintas soluciones del programa
b. Para imprimir la fecha y hora
c. Para cifrar la contraseña en función de la hora actual
d. Para cronometrar lo que tarda el usuario en introducir la contraseña

---

16. Un fragmento de una "bomba" desensamblada es:

```
0x0804873f: call 0x8048504 <scanf>
0x08048744: mov 0x24(%esp),%edx
0x08048748: mov 0x804a044,%eax
0x0804874d: cmp %eax,%edx
0x0804874f: je 0x8048756 <main+230>
0x08048751: call 0x8048604 <boom>
0x08048756: ...
```

La contraseña/clave en este caso es...

a. el string almacenado a partir de la posición de memoria 0x804a044
b. el string almacenado a partir de la posición de memoria 0x24(%esp)
c. el entero almacenado a partir de la posición de memoria 0x804a044
d. el entero 0x804a044

---

17. Una de las "bombas" utiliza el siguiente bucle para cifrar la cadena con la contraseña introducida por el usuario:

```
80485bb: rolb $0x4,(%eax)
80485be: add  $0x1,%eax
80485c1: cmp  %edx,%eax
80485c3: jne 80485bb <encrypt+0x20>
```

La intrucción **rolb** rota el byte destino hacia la izquierda tantos bits como indica el operando fuente. Si inicialmente eax apunta a la cadena del usuario, que se compara con otra cadena "\x16\x26\x27\x16\x36\x16\x46\x16\x26\x27\x16", almacenada en el código, la contraseña es:

a. "\x61\x62\x72\x61\x63\x61\x64\x61\x62\x72\x61" ("abracadabra")
b. "\x61\x72\x62\x61\x64\x61\x63\x61\x72\x62\x61" ("arbadacarba")
c. "\x63\x61\x64\x61\x62\x72\x61\x61\x62\x72\x61" ("cadabraabra")
d. "\x61\x62\x72\x61\x61\x62\x72\x61\x63\x61\x64" ("abraabracad")

---

18. Una de las "bombas" utiliza el siguiente código para cifrar la clave numérica introducida por el usuario y ahora almacenada en eax:

```
804870d:    xor    $0xfff,%eax
8048712:    mov    $0x2,%ecx
8048717:    cltd
8048718:    idiv   %ecx
804871a:    cmp    %eax,0x804a034
```

Si el entero almacenado a partir de 0x804a034 es 0x7ff, la clave numérica puede ser:

a. 0x10094F97 (269045655)
b. 0xfff (2047)
c. 0x7ff (4095)
d. 1

---

19. En el programa line.cc de la práctica 5, si para cada tamaño de línea (line) recorremos una única vez el vector, la gráfica resultante es decreciente porque:

a. Cada vez que line aumenta al doble, el número de aciertos por localidad temporal aumenta, porque ya habíamos accedido a cada posición i del vector cuando lo recorrimos en el punto anterior del eje X.
b. Cada vez que line aumenta al doble, el número de aciertos por localidad espacial aumenta, porque ya habíamos accedido a cada posición i-1 del vector cuando lo recorrimos en el punto anterior del eje X.
c. Cada vez que line aumenta al doble, se accede con éxito a más posiciones del vector en niveles de la jerarquía de memoria más rápidos.
d. Cada vez que line aumenta al doble, realizamos la mitad de accesos al vector que para el valor anterior.

---

20. ¿Cuál de las siguientes afirmaciones sobre el programa size.cc de la práctica 5 es cierta?

a. La diferencia de velocidades entre L2 y L3 es mayor que la diferencia de velocidades entre L1 y L2.
b. Si continuáramos multiplicando por 2 el tamaño del vector en el eje X obteniendo más puntos de la gráfica, esta continuaría horizontal para cualquier valor más allá de 64 MB.
c. La gráfica tiene escalones hacia arriba porque en cada punto del eje X accedemos al mismo número de elementos del vector y el número de aciertos por localidad temporal disminuye bruscamente en ciertos puntos al aumentar el tamaño del vector.
d. La gráfica tiene tramos horizontales porque el hecho de realizar la mitad de accesos al vector en cada punto de un tramo horizontal respecto al anterior punto de ese mismo tramo horizontal es compensado por el número de fallos creciente en ese mismo tramo horizontal.

Nombre:	
DNI:	**Grupo:**

Test de Prácticas (4.0p)

Todas las preguntas son de elección simple sobre 4 alternativas.
Cada respuesta vale 0.2p si es correcta, 0 si está en blanco o claramente tachada, -0.0$\widehat{6}$p si es errónea.
Anotar las respuestas (a, b, c ó 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. ¿Cuál de las siguientes instrucciones máquina copia en EAX la dirección efectiva resultante de la operación EDX*4 + EBX?

a. leal 4(%edx, %edx), %eax
b. leal (%ebx, %edx, 4), %eax
c. movl 4(%edx, %edx), %eax
d. movl (%ebx, %edx, 4), %eax

2. ¿Cuál de las siguientes instrucciones máquina copia en EAX el entero almacenado en la posición de memoria cuya dirección efectiva es el resultado de la operación EDX*4 + EBX?

a. leal 4(%edx, %edx), %eax
b. leal (%ebx, %edx, 4), %eax
c. movl 4(%edx, %edx), %eax
d. movl (%ebx, %edx, 4), %eax

3. Los switches `-m elf_i386` y `-m elf_x86_64` para trabajar en 32 bits / 64 bits corresponden a la herramienta...

a. gcc
b. as
c. ld
d. nm

4. Si ECX vale 0, la instrucción adc $0,%ecx

a. Pone CF=1
b. Pone CF=0
c. Cambia CF
d. No cambia CF

5. Dada la siguiente definición de datos:

```
lista:  .int 0x10000000, 0x50000000,
             0x10000000, 0x20000000
longlista:   .int (.-lista)/4
resultado:   .quad 0x123456789ABCDEF
formato: .ascii "suma=%llu=%llx hex\n\0"
```

La instrucción para copiar la dirección de memoria donde comienza lista en el registro EBX es:

a. movl lista, %ebx
b. movl $lista, %ebx
c. movl (lista), %ebx
d. movl $lista, (%ebx)

6. Dada la siguiente definición de datos:

```
lista:  .int 0x10000000, 0x50000000,
             0x10000000, 0x20000000
longlista:   .int (.-lista)/4
resultado:   .quad 0x123456789ABCDEF
formato: .ascii "suma=%llu=%llx hex\n\0"
```

la instrucción `movl longlista, %ecx` copia el siguiente valor:

a. 4
b. 8
c. 16
d. 32

---

7. Dada la siguiente definición de datos:

```
lista:  .int 0x10000000, 0x50000000,
            0x10000000, 0x20000000
longlista:  .int (.-lista)/4
resultado:  .quad 0x123456789ABCDEF
formato: .ascii "suma=%llu=%llx hex\n\0"
```

y suponiendo que hemos llamado a una función *suma* que devuelve un número de 64 bits en la pareja EDX:EAX, las instrucciones que copian ese número en *resultado* son:

a. movl %eax, resultado
   movl %edx, resultado+4
b. movl (%eax), resultado
   movl (%edx), resultado+4
c. movl %eax, resultado+4
   movl %edx, resultado
d. movl (%eax), resultado+4
   movl (%edx), resultado

---

8. Dada la siguiente definición de datos:

```
lista:  .int 0x10000000, 0x50000000,
            0x10000000, 0x20000000
longlista:  .int (.-lista)/4
resultado:  .quad 0x123456789ABCDEF
formato: .ascii "suma=%llu=%llx hex\n\0"
```

la llamada correcta a *printf* será:

a. push resultado+4
   push resultado
   push $formato
   call printf
   add $12, %esp
b. push resultado+4
   push resultado
   push resultado+4
   push resultado
   push $formato
   call printf
   add $20, %esp
c. push resultado
   push resultado+4
   push $formato
   call printf
   add $12, %esp
d. push resultado
   push resultado+4
   push resultado
   push resultado+4
   push $formato
   call printf
   add $20, %esp

---

9. En la práctica "media" se pide sumar una lista de enteros ***con*** signo de 32 bits en una plataforma de 32 bits sin perder precisión, esto es, evitando overflow. ¿Cuál es el menor valor positivo que repetido en los ***dos*** primeros elementos de la lista causaría overflow con 32 bits al realizar la suma de ***esos dos*** primeros elementos de la lista?

a. 0x0400 0000
b. 0x0800 0000
c. 0x4000 0000
d. 0x8000 0000

---

10. ¿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

---

11. ¿Cuál de los siguientes registros tiene que ser salvaguardado (si va a modificarse) dentro de una subrutina según la convención cdecl para IA32?

a. eax

b. ebx

c. ecx

d. edx

---

12. ¿En qué registro se pasa el primer argumento a una función en Linux gcc x86-64?

a. ecx

b. edx

c. esi

d. edi

---

13. La práctica "popcount" debía calcular la suma de bits (peso Hamming) de los elementos de un array. Un estudiante entrega la siguiente versión de popcount3:

```
int popcount3(unsigned* array,
              int len){
    int i, res = 0;
    unsigned x;
    for( i=0; i<len; i++ ) {
      x = array[i];
      asm("ini3:       \n"
      "shr %[x]        \n"
      "adc $0, %[r]    \n"
      "add $0, %[x]    \n"
      "jne ini3        \n"
        : [r] "+r" (res)
        : [x] "r" (x) );
    }
    return res;
}
```

Esta función produce siempre el resultado correcto, a pesar de que una instrucción máquina en la sección **asm()** es distinta a la que se esperaba después de haber estudiado **`pcount_r`** en teoría. La instrucción distinta también se podría haber cambiado por…

a. sar %[x]

b. adc $0, %[x]

c. test %[x], %[x]

d. cmp %[x], %[r]

---

14. En la práctica de la bomba necesitamos estudiar el código máquina de la bomba del compañero. A veces dicho código no se visualiza directamente en el depurador ddd, y algunas de las técnicas que se pueden probar para conseguir visualizarlo son… (marcar la opción ***falsa***)

a. comprobar que está activado el panel *View → Machine Code Window*

b. escribir **`info line main`** en el panel de línea de comandos gdb

c. recompilar con información de depuración, por si se nos había olvidado, ya que sin –g el ejecutable no contiene información de depuración

d. asegurarse de que se ha escrito correctamente el nombre del ejecutable

---

15. En la práctica de la bomba, el primer ejercicio consistía en “saltarse” las “explosiones”, para lo cual se puede utilizar…

a. objdump o gdb
b. gdb o ddd
c. ddd o hexedit
d. hexedit u objdump

16. En la práctica de la bomba, el segundo ejercicio consistía en crear un ejecutable sin "explosiones", para lo cual se puede utilizar…

a. objdump o gdb
b. gdb o ddd
c. ddd o hexedit
d. hexedit u objdump

17. En la práctica de la bomba, el tercer ejercicio consistía en usar un editor hexadecimal para crear un ejecutable sin "explosiones". Para saber qué contenidos del fichero hay que modificar, se puede utilizar… (marcar la opción falsa)

a. objdump
b. gdb
c. ddd
d. hexedit

18. Suponer una memoria cache con las siguientes propiedades: Tamaño: 512 bytes. Política de reemplazo: LRU. Estado inicial: vacía (todas las líneas inválidas). Suponer que para la siguiente secuencia de direcciones enviadas a la cache: 0, 2, 4, 8, 16, 32, la tasa de acierto es 0,33. ¿Cuál es el tamaño de bloque de la cache?

a. 4 bytes
b. 8 bytes
c. 16 bytes
d. Ninguno de los anteriores

19. Abajo se ofrece el listado de una función para multiplicar matrices C = A x B.

```
void mult_matr(float A[N][N],
float B[N][N],float C[N][N]){
/*Asume val.ini. C={0,0...}*/
int i,j,k;
for (i=0; i<N; i++)
 for (j=0; j<N; j++)
  for (k=0; k<N; k++)
   C[i][j]+= A[i][k]*B[k][j];
}
```

Suponer que:

• El computador tiene una cache de datos de 8 MB, 16-vías, líneas de 64 bytes.

• N es grande, una fila o columna no cabe completa en cache.

• El tamaño de los tipos de datos es como en IA32.

• El compilador optimiza el acceso a C[i][j] en un registro.

Aproximadamente, ¿qué tasa de fallos se podría esperar de esta función para valores grandes de N?

a. 1/2
b. 1/4
c. 1/8
d. 1/16

20. Sea un computador de 32 bits con una memoria cache L1 para datos de 32 KB y líneas de 64 bytes asociativa por conjuntos de 2 vías. Dado el siguiente fragmento de código:

```
int v[262144];
for (i=0; i<262144; i+=8)
   v[i] = 9;
```

¿Cuál será la tasa de fallos aproximada que se obtiene en la ejecución del bucle anterior?

a. 0 (ningún fallo)
b. 1/2 (mitad aciertos, mitad fallos)
c. 1/8 (un fallo por cada 8 accesos)
d. 1 (todo son fallos)

2º Grado Informática
Estructura de Computadores
18 de enero de 2018

Nombre:	
DNI:	**Grupo:**

Test de Prácticas (4.0p)

Todas las preguntas son de elección simple sobre 4 alternativas.
Cada respuesta vale 0.2p si es correcta, 0 si está en blanco o claramente tachada, -0.0$\widehat{6}$p si es errónea.
Anotar las respuestas (a, b, c ó 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. ¿Cuál de los siguientes fragmentos es correcto para comenzar un programa en ensamblador que conste de un solo archivo .s?

a.
```
.text:
_start:
```
b.
```
.text
.local _start
_start:
```
c.
```
.text
.start _global
_start:
```
d.
```
.text
.global _start
_start:
```

---

2. Suponga una memoria cache con las siguientes propiedades: Tamaño: 512 bytes. Política de reemplazo: LRU. Estado inicial: vacía (todas las líneas inválidas). Suponga que para la siguiente secuencia de direcciones enviadas a la cache: 1, 2, 4, 8, 16, 32, la tasa de acierto es 0,333. ¿Cuál es el tamaño de línea de la cache?

a. 4 bytes
b. 8 bytes
c. 16 bytes
d. 32 bytes

---

3. En la convención cdecl estándar para arquitecturas x86 de 32 bits, cuál de las siguientes afirmaciones es cierta:

a. Los 6 primeros parámetros se pasan a través de registros
b. Solamente es necesario salvar el registro EAX
c. Los registros EBX, ESI y EDI son salva - invocante
d. Ninguna de las anteriores es cierta

---

4. ¿Cuál es el popcount (peso Hamming, nº de bits activados) del número 0x10101010?

a. 4
b. 8
c. 16
d. 32

---

5. Compilar de fuente C a ejecutable usando sólo as y ld, sin gcc...

a. Se puede, repartiendo entre as y ld los modificadores (switches) que corresponda
b. Basta usar ld, con los modificadores de gcc que corresponda, y añadiéndole el runtime de C
c. Se puede, repartiendo modificadores entre as y ld, y añadiendo al comando ld el runtime de C
d. No se puede

---

6. La función gettimeofday() en la práctica de popcount y parity se utiliza para

a. Comparar las duraciones de las distintas soluciones del programa
b. Imprimir la fecha y hora
c. Cifrar el código en función de la hora actual
d. Cronometrar lo que tarda el usuario en pulsar una tecla

---

7. ¿Cuál de los siguientes registros tiene que ser salvaguardado (si va a modificarse) dentro de

una subrutina según la convención cdecl para IA32?

a. ECX
b. EAX
c. EBP
d. EDX

---

8. ¿Qué hace gcc -O1?

a. Compilar .s→.o (fuente ASM a objeto)
b. Compilar con optimización
c. Compilar .c→.o (fuente C a objeto)
d. Compilar .c→.s (C→ASM sin generar objeto)

---

9. Dada la siguiente definición de datos:

```
lista: .int 0x10000000, 0x50000000,
            0x10000000, 0x20000000
longlista: .int (.-lista)/4
resultado: .quad 0x123456789ABCDEF
formato: .ascii "%llu=%llx hex\n\0"
```

La instrucción para copiar la dirección de memoria donde comienza lista en el registro EBX es:

a. movl lista, %ebx
b. movl $lista, %ebx
c. movl (lista), %ebx
d. movl $lista, (%ebx)

---

10. En la práctica "media" se programa la suma de una lista de 32 enteros de 4 B para producir un resultado de 8 B, primero sin signo y luego con signo. Si la lista se rellena con el valor 0x8000 0000, ¿en qué se diferencian los resultados de ambos programas?

a. no se diferencian
b. en uno ocupa 32 bits, en otro 64 bits
c. en uno los 16 bits superiores son 0xFFFF, en el otro no
d. en uno los 16 bits inferiores son 0x0000, en el otro no

---

11. En la práctica "media" se pide sumar una lista de 32 enteros CON signo de 32bits en una plataforma de 32bits sin perder precisión, esto es, evitando desbordamiento. Un estudiante entrega un programa que se diferencia de la versión recomendada en el siguiente bucle, en particular en la instrucción adc.

```
bucle:
   mov (%ebx,%esi,4), %eax
   cltd
   add %eax, %edi
   adc %eax, %ebp
   inc %esi
   cmp %esi, %ecx
   jne bucle
```

Esta versión de la suma CON signo

a. produce siempre el resultado correcto
b. fallaría con lista: .int 1, 1, 1, 1, ...
c. fallaría con lista: .int -1,-1,-1,-1, ...
d. no siempre produce el resultado correcto, pero el error no se manifiesta en los ejemplos propuestos, o se manifiesta en ambos

---

12. En la práctica "media" un estudiante usa el siguiente bucle para acumular la suma en EBP:EDI antes de calcular la media y el resto

```
bucle:
   mov (%ebx,%esi,4), %eax
   cltd
   add %eax, %edi
   adc %edx, %ebp
   jnc nocarry
   inc %edx
nocarry:
   inc %esi
   cmp  %esi,%ecx
   jne bucle
```

Este código es una mezcla de las soluciones recomendadas para suma sin signo y para suma con signo. Estando bien programado todo lo demás, este código

a. produce siempre el resultado correcto
b. fallaría con lista: .int 0,1,2,3
c. fallaría con lista: .int -1,-2,-4,-8
d. no siempre produce el resultado correcto, pero el error no se manifiesta en los ejemplos propuestos, o se manifiesta en ambos

---

13. ¿Cuál es el popcount (peso Hamming, nº de bits activados) de una lista de N números inicializada con los valores 0..N-1?

a. (N-1)*N/2
b. N*(N+1)/2
c. si N es par, N*(N/2)
d. si N es potencia de 2, log_2(N)*N/2

---

14. ¿Cuál es la paridad (XOR "lateral" de todos los bits) del número 199?

a. 0
b. 1
c. 2
d. 4

---

15. Comparando los popcounts (pop(199) vs. pop(99)) y paridades (par(199) vs. par(99)) de los números 199 y 99, se verifica que

a. pop(199) > pop(99)
b. par(199) < par(99)
c. pop(199) < par(99)
d. par(199) > pop(99)

---

16. ¿Cuál es la suma de paridades (suma de los XOR "laterales" de los bits de cada número) de una lista de N números inicializada con los valores 0..N-1?

a. (N-1)*N/2
b. N*(N+1)/2
c. si N es par, N/2
d. si N es potencia de 2, (log_2(N)*N)/2

---

17. En la práctica "popcount/paridad", para cronometrar sistemáticamente las diversas versiones necesitamos una función crono() a la que se le pueda pasar como argumento cuál versión queremos cronometrar. En lenguaje C esto se puede hacer con punteros a funciones. Sabiendo que todas las versiones devuelven un valor entero, el prototipo de la función crono() debería ser:

a. void crono(int * func (), char* msg);
b. void crono(int (* func)(), char* msg);
c. void crono((int *)func (), char* msg);
d. void crono(int * func , char* msg);

---

18. Para corregir la práctica "bomba digital", un profesor dispone de 26 ejecutables (y la lista de claves correspondientes) para asignar en el Laboratorio una bomba distinta a cada estudiante. Cuando un estudiante diga que la ha resuelto el profesor exigirá ver al estudiante ejecutando la bomba y tecleando la contraseña y el pin correctos para comprobar que no explota, para así anotarla como resuelta y que le puntúe al estudiante.

Un estudiante (usando ordenador del Laboratorio con Ubuntu 10.04) dice que ha resuelto su bomba, y cuando el profesor pide que se tecleen las claves, el ddd se "bloquea" y le empieza a "parpadear" al estudiante. Para poder puntuar, lo más recomendable para el estudiante sería…

a. teclear las claves (contraseña y pin), aunque ddd esté "bloqueado" y "parpadeando"
b. probar en orden los remedios básicos: pulsar <Ctrl>-C varias veces, pulsar <Enter> repetidamente, comprobar que está seleccionada "Machine Code Window", teclear "info line main", y si todo falla, ejecutar "**`rm –rf ~/ .ddd`**"
c. matar la ventana ddd, abrir un terminal con un shell, ejecutar la bomba desde línea de comandos y teclear las claves
d. reinstalar un paquete ddd más actualizado usando "sudo apt-get install"

---

19. Suponer el mismo contexto de la pregunta anterior, donde el profesor tiene una lista de las 26+26 claves (contraseñas y pines)

Un estudiante dice que ha resuelto su bomba, y cuando el profesor pide que se tecleen las claves, la contraseña coincide con la que tiene anotada el profesor, pero el pin no, y de todas formas la bomba no explota. Debería hacerse lo siguiente:

a. la bomba tiene que puntuarle al estudiante porque no ha explotado
b. el profesor puede pedirle que vuelva a descargar la bomba original e intente repetir con ese ejecutable las claves que acaba de teclear
c. la bomba debe marcarse como inválida y no hay que hacer más comprobaciones
d. no puede suceder lo que dice el enunciado, y en ningún caso el profesor tiene derecho a hacer comprobaciones adicionales como pedir que se vuelva a descargar la bomba

---

20. En la práctica de la cache, el código de "line.cc" incluye la sentencia

```
for (unsigned long long line=1;
        line<=LINE; line<<=1) { … }
```

¿Qué objetivo tiene la expresión line<<=1?

a. salir del bucle si el tamaño de línea se volviera menor o igual que 1 para algún elemento del vector
b. duplicar el tamaño del salto en los accesos al vector respecto a la iteración anterior
c. volver al principio del vector cuando el índice exceda la longitud del vector
d. sacar un uno (1) por el stream line

2º Grado Informática
Estructura de Computadores
9 de febrero de 2018

Nombre:	
DNI:	**Grupo:**

Test de Prácticas (4.0p)

Todas las preguntas son de elección simple sobre 4 alternativas.
Cada respuesta vale 0.2p si es correcta, 0 si está en blanco o claramente tachada, -0.0$\widehat{6}$p si es errónea.
Anotar las respuestas (a, b, c ó 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. El punto de entrada de un programa ensamblador en GNU/as Linux x86 se llama

a. main
b. begin
c. _start
d. _init

---

2. En un programa en ensamblador queremos crear espacio para una variable entera var inicializada a 1. La línea que hemos de escribir en la sección de datos es:

a. .int var 1
b. var: .int 1
c. .int: var 1
d. int var 1

---

3. En la práctica "media" se pide sumar una lista de 32 enteros SIN signo de 32 bits en una plataforma de 32 bits sin perder precisión, esto es, evitando perder acarreos. Un estudiante entrega la siguiente versión

```
# $lista en EBX, longlista en ECX
suma:
        mov $0, %eax
        mov $0, %edx
        mov $0, %esi
bucle:
        add (%ebx,%esi,4), %eax
        jne nocarry
        inc %edx
nocarry:
        inc %esi
        cmp %esi,%ecx
        jne bucle
        ret
```

Esta función presenta una única diferencia frente a la solución recomendada en clase, relativa al salto condicional.

Esta función suma:

a. produce siempre el resultado correcto
b. fallaría con lista: .int 1,1,1,1, 1,1,1,1, ...
c. fallaría con lista: .int 1,2,3,4, 1,2,3,4, ...
d. no es correcta pero el error no se manifiesta en los ejemplos propuestos, o se manifiesta en ambos

---

4. En la misma práctica "media" un estudiante entrega la siguiente versión de suma sin signo:

```
main: .global main
   ...
   call suma
   ...
   mov $1, %eax
   mov $0, %ebx
   int $0x80
suma:
   ...
bucle:
   ...
nocarry:
   inc %esi
   cmp %esi,%ecx
   jne bucle
```

Notar que falta la instrucción ret final. Al desensamblar el código ejecutable se obtiene

```
08048445 <nocarry>:
 8048445: 46    inc %esi
 8048446: 39 f1 cmp %esi,%ecx
 8048448: 75 f5 jne 804843f <bucle>
 804844a: 90    nop
 804844b: 90    nop
 .......  90    nop
 804844f: 90    nop
```

```
08048450 <__libc_csu_fini>:
 8048450: 55    push   %ebp
 8048451: 89 e5 mov    %esp,%ebp
 8048453: 5d    pop    %ebp
 8048454: c3    ret
```

Este programa:

a. está correctamente diseñado, la instrucción ret final es optativa, y no es concebible que quitar el ret cause algún error

b. produce un error "Segmentation fault" cuando empieza a acceder a memoria que no le corresponde (EIP)

c. funciona bien, pero si pusiéramos en el código fuente primero la definición de suma y luego la de main, el ejecutable terminaría accediendo a memoria que no le corresponde (ESP) y hará "Segmentation fault"

d. no se puede marcar ninguna de las opciones anteriores

---

5. ¿Cuál de las siguientes sumas con signo produce desbordamiento con 32 bits?

a. 0xFFFFFFFF + 0xFFFFFFFF

b. 0x7FFFFFFF + 0xFFFFFFFF

c. 0x7FFFFFFF + 0x000000001

d. 0xFFFFFFFF + 0x000000001

---

6. En la práctica “media” se pide sumar una lista de 32 enteros *con* signo de 32bits en una plataforma de 32bits sin perder precisión, esto es, evitando overflow. ¿Cuál es el menor valor positivo que repetido en toda la lista causaría overflow con 32bits?

a. 0x0400 0000

b. 0x0800 0000

c. 0x4000 0000

d. 0x8000 0000

---

7. En la práctica “media” se pide sumar una lista de 32 enteros *con* signo de 32bits en una plataforma de 32bits sin perder precisión, esto es, evitando overflow. ¿Cuál es el mayor valor negativo (menor en valor absoluto) que repetido en toda la lista causaría overflow con 32bits?

a. 0xffff ffff

b. 0xfc00 0000

c. 0xfbff ffff

d. 0xf000 0000

---

8. ¿Cuál es el popcount (peso Hamming, nº de bits activados) del número 29?

a. 2

b. 3

c. 4

d. 5

---

9. La práctica "popcount" debía calcular la suma de bits (peso Hamming) de los elementos de un array. Un estudiante entrega la siguiente versión de popcount4:

```
int popcount4(unsigned* array, int
len){
  int val = 0;
  int i, j;
  unsigned x;
  int res = 0;
    for (i=0; i<len; i++){
      x = array[i];
      val = 0;
      for (j=0; j<8; j++){
        val += x & 0x01010101;
        x >>= 1;
      }
      val += (val>>16);
      val += (val>>8);
      res += val;
    }
    return (res & 0XFF);
}
```

Esta función presenta varias diferencias con la versión "oficial" recomendada en clase, incluyendo la "doble inicialización" de val y la acumulación y retorno de res. Esta popcount4:

a. produce siempre el resultado correcto

b. fallaría con array={0,1,2,3}

c. fallaría con array={1,2,4,8}

d. no es correcta pero el error no se manifiesta en los ejemplos propuestos, o se manifiesta en ambos

---

10. En la misma práctica "popcount" un estudiante entrega la siguiente versión de popcount4:

```
int popcount4(unsigned* array, int
len){
  int i, j;
  unsigned x;
  int result = 0;
  long val;
    for (i=0; i<len; i++){
      x = array[i];
      val = 0;
      for (j=0; j<8*sizeof(int);j++){
        val += x & 0x01010101;
        x >>= 1;
      }
      val += (val>>16);
      val += (val>>8);
      result += val & 0xFF;
    }
    return result;
```

```
}
```

Esta función presenta varias diferencias con la versión "oficial" recomendada en clase, incluyendo el tipo de val y las condiciones del bucle for.

Esta función popcount4:

a. produce siempre el resultado correcto
b. fallaría con array={0,1,2,3}
c. fallaría con array={1,2,4,8}
d. no es correcta pero el error no se manifiesta en los ejemplos propuestos, o se manifiesta en ambos

---

11. Comparando los popcounts (pop(129) vs. pop(29)) y paridades (par(129) vs. par(29)) de los números 129 y 29, se verifica que

a. pop(129) > pop(29)
b. par(129) > par(29)
c. pop(129) < par(29)
d. par(129) < pop(29)

---

12. La práctica "parity" 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 parity4:

```
int parity4(unsigned* array, int
len){
  int val;
  int i;
  unsigned x;
  int result = 0;
    for (i=0; i<len; i++){
      x = array[i];
      val = 0;
      asm(\n"
      "ini3: \n\t"
        "shr $0x1, %[x] \n\t"
        "adc $0x0, %[r] \n\t"
        "test %[x], %[x] \n\t"
        "jnz ini3 "
      : [r]"+r"(val)
      : [x]"r"(x)
      );
      result += val & 0x1;
    }
    return result;
}
```

Esta función presenta una sentencia asm distinta de la versión "oficial" recomendada en clase. En concreto son distintas la etiqueta y las instrucciones adc/test.

Esta función parity4:

a. produce siempre el resultado correcto
b. fallaría con array={0,1,2,3}
c. fallaría con array={1,2,4,8}
d. no siempre produce el resultado correcto, pero el error no se manifiesta en los ejemplos propuestos, o se manifiesta en ambos

---

13. En la misma práctica "parity" un estudiante entrega la siguiente versión de parity4:

```
int parity4(unsigned* array, int
len){
  int i;
  unsigned x;
  int val, result = 0;
    for (i=0; i<len; i++){
      x = array[i];
      val = 0;
      asm("\n"
      "ini4: \n\t "
      " xor %[x], %[y] \n\t"
      " shr $1, %[x] \n\t "
      " cmpl $0, %[x] \n\t "
      " jnz ini4 \n\t "
      :[y] "+r" (val)
      :[x]  "r"  (x)
      );
      result +=val & 0x1;
    }
    return result;
}
```

Esta función presenta dos diferencias con la versión "oficial" recomendada en clase, relativas a la instrucción cmp y al nombre [y] escogido para la restricción val. Esta parity4:

a. produce siempre el resultado correcto
b. fallaría con array={0,1,2,3}
c. fallaría con array={1,2,4,8}
d. no siempre produce el resultado correcto, pero el error no se manifiesta en los ejemplos propuestos, o se manifiesta en ambos

---

14. En la misma práctica "parity" un estudiante entrega la siguiente versión de parity6:

```
int parity6(int *array, int len){
  int  i,  res=0;
  unsigned x;
    for (i = 0; i < len; i++){
      x = array[i];
      asm(
        "mov %[x], %%edx \n\t"
        "shr $16,  %[x]  \n\t"
        "xor %[x], %%edx \n\t"
        "xor   %%dh, %%dl\n\t"
        "setpo %%dl      \n\t"
        "movzx %%dl, %[x]\n\t"
      : [x] "+r" (x)
      :
      : "edx"
      );
      res += (x & 0x1);
    }
    return res;
```

```
}
```

Esta función presenta dos diferencias con la versión "oficial" recomendada en clase, relativas al tipo del array y a la máscara y paréntesis usados al acumular.

Esta función parity6:

a. produce siempre el resultado correcto
b. fallaría con array={0,1,2,3}
c. fallaría con array={1,2,4,8}
d. no siempre produce el resultado correcto, pero el error no se manifiesta en los ejemplos propuestos, o se manifiesta en ambos

---

15. En la práctica de la bomba, el primer ejercicio consistía en “saltarse” las “explosiones”, para lo cual se puede utilizar…

a. objdump o gdb
b. gdb o ddd
c. ddd o hexedit
d. hexedit u objdump

---

16. En la práctica de la bomba, el tercer ejercicio consistía en usar un editor hexadecimal para crear un ejecutable sin “explosiones”. Para saber qué contenidos del fichero hay que modificar, se puede utilizar… (marcar la opción ***FALSA***)

a. objdump
b. gdb
c. ddd
d. hexedit

---

17. Suponer una memoria cache con las siguientes propiedades: Tamaño: 512 bytes. Política de reemplazo: LRU. Estado inicial: vacía (todas las líneas inválidas). Suponer que para la siguiente secuencia de direcciones enviadas a la cache: 0, 2, 4, 8, 16, 32, la tasa de acierto es 0.33. ¿Cuál es el tamaño de bloque de la cache?

a. 4 bytes
b. 8 bytes
c. 16 bytes
d. Ninguno de los anteriores

---

18. Sea un computador de 32 bits con una memoria cache L1 para datos de 32 KB y líneas de 64 bytes asociativa por conjuntos de 2 vías. Dado el siguiente fragmento de código:

```
int v[262144];
for (i = 0; i < 262144; i += 8)
  v[i] = 9;
```

¿Cuál será la tasa de fallos aproximada que se obtiene en la ejecución del bucle anterior?

a. 0 (ningún fallo)
b. 1/2 (mitad aciertos, mitad fallos)
c. 1/8 (un fallo por cada 8 accesos)
d. 1 (todo son fallos)

---

19. En la práctica de la cache, el código de size.cc accede al vector saltando de 64 en 64. ¿Por qué?

a. Porque cada elemento del vector ocupa 64 bytes
b. Para recorrer el vector más rápidamente
c. Porque el tamaño de cache L1 de todos los procesadores actuales es de 64KB
d. Para anular los aciertos por localidad espacial, esto es, que sólo pueda haber aciertos por localidad temporal

---

20. ¿En qué unidades se suelen medir las capacidades de almacenamiento de los niveles de cache L1, L2 y L3 de un microprocesador actual (2017-2018)?

a. L1 en KB, L2 en KB o MB, L3 en MB.
b. L1 en MB, L2 en GB, L3 en GB o TB.
c. L1 en MB, L2 en MB, L3 en GB.
d. L1 en KB, L2 en MB, L3 en GB.

2º Grado Informática
Estructura de Computadores
14 de enero de 2019

Nombre:	
DNI:	**Grupo:**

Test de Prácticas (4.0p)

Todas las preguntas son de elección simple sobre 4 alternativas.
Cada respuesta vale 0.2p si es correcta, 0 si está en blanco o claramente tachada, -0.0$\widehat{6}$p si es errónea.
Anotar las respuestas (a, b, c ó 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
															c		d		

1. Habiendo definido en código fuente ASM **`longsal:.quad .-saludo`** justo detrás de un string **`saludo`** que ocupaba 28 bytes, si se comparan los comandos gdb siguientes: **`x/1xg &longsal`** frente a **`print (long) &longsal`**:

a. ambos nos muestran la longitud del string (que es/vale/equivale a 28)
b. el primero (x) nos muestra el string, y el segundo (print) nos muestra otro valor distinto
c. el segundo (print) nos muestra el string, y el primero (x) nos muestra otro valor distinto
d. alguno o ambos contienen algún error gramatical (falta o sobra algún &, algún typecast (char*) o (long), etc.)

2. En la práctica "media" se pide sumar una lista de 16 enteros SIN signo de 32 bits evitando acarreo. ¿Cuál es el menor valor que repetido en toda la lista causaría acarreo en 32 bits?

a. 0xFFFF FFFF
b. 0x7FFF FFFF
c. 0x1000 0000
d. 0x0FFF FFFF

3. En la práctica "media" se pide usar **`adc`** para sumar una lista de 16 enteros SIN signo de 32 bits en dos registros de 32 bits mediante extensión con ceros. Un estudiante entrega la siguiente versión:

```
…
resultado:    .quad   0
…
main: .global  main
   mov  $lista, %rbx
   mov  $16, %ecx
   call suma
   mov  %eax, resultado
   # código para printf …
   # código para _exit …
suma:
   mov  $0, %eax
   mov  $0, %rdx
bucle:
   adc  (%rbx, %rdx, 4), %eax
   inc   %rdx
   cmp   %rdx, %rcx
   jnle   bucle
   ret
```

Este programa no usa la variable **`longlista`**, guarda el resultado con una instrucción MOV, usa como índice RDX, no usa la instrucción ADD, y usa JNLE para el salto condicional.

Al empezar un programa CF no está activado. Esta versión de la suma SIN signo mediante extensión con ceros da resultado correcto:

a. con lista: .int 0x10000000, ... (16 elementos)
b. con lista: .int 200000000, ... (16 elementos)
c. con ambos ejemplos
d. con ninguno de los dos ejemplos

4. En la práctica "media" se pide usar **`cltd/cdq`** para sumar una lista de 16 enteros CON signo de 32 bits en dos registros de 32 bits mediante extensión de signo. Un estudiante entrega la siguiente versión:

```
…
main: .global  main
   mov     $lista, %rbx
   mov  longlista, %ecx
   call suma
   mov  %eax, resultado
   mov  %edx, resultado+4
```

```
   movq   $formato, %rdi
   movq   resultado,%rsi
   movq   resultado,%rdx
   movl          $0,%eax
   call  printf
   …
```

El programa produce la siguiente salida con el test #01 (16 elementos con valor -1):

```
__TEST01__-----------------------------
resultado   =                  -16 (sng)
            = 0x  fffffffffffffff0 (hex)
            = 0x 00000010 9f816d80
```

Recordar que todo el texto aparecía tal cual literalmente en el formato (ignorar la errata sng) y los números llevaban especificación de formato (%18ld, %18lx, etc). De esta versión de la suma CON signo mediante extensión de signo se puede afirmar que:

a. al inicio de main EAX vale 0 y R8 contiene un valor inferior a 0x7fffffff00000000
b. al llamar a suma RBX contiene un valor inferior a 0x600000 y RCX vale 16
c. al llamar a printf, ECX vale 16 y R8 contiene un valor superior a 0x80000000
d. tras volver de printf RAX contiene un valor superior a 60 y RDI superior a 0x600000

---

5. En la práctica "media" se pide usar **`cltq/cdqe`** y **`cqto/cqo`** para hallar la media y resto de una lista de 16 enteros CON signo de 32 bits usando registros de 64 bits. Un estudiante entrega la siguiente versión:

```
…
media:  .double 0
resto:  .double 0
formatoq:
 .ascii "media = %11d resto = %11d\n"
 .asciz "\t = 0x %08x \t = 0x %08x\n"
…
   mov     $lista, %rbx
   mov  longlista, %ecx
   call sumaq

   mov   $formatoq, %rdi
   mov   media,%rsi
   mov   resto,%rdx
   mov   $0,%eax
   call  printf
   …
sumaq:
   push  %rdx
   push   %rsi
   mov  $0, %rax
   mov  $0, %rsi
   mov  $0, %rdx
   mov  $0, %r8
bucleq:
   mov  (%rbx,%rsi,4), %eax
   cdqe             # EAX -> RAX
   add   %rax, %r8
   inc   %rsi
   cmp   %rsi,%rcx
   jne    bucleq

   mov   %r8, %rax
   cqo              # RAX -> RDX:RAX
   idivq %rsi
#  mov   %rdx, %r10
   mov   %rdx, resto
   mov   %rax, media
   pop   %rdx
   pop   %rsi
   ret
```

Este programa es muy diferente a la versión "oficial" recomendada en clase. Notar los **`push/pop`**, los **`mov $0`** adicionales, **`idiv %rsi`** en lugar de **`%rcx`**, los **`mov media/resto`** al final de la subrutina en lugar de tras la llamada en **`main`**, y el tipo de ambas variables.

¿Qué media y resto imprime esta versión para el test #03? (16 elementos con valor 0x7fffffff)

a.
```
media =  2147483647 resto =           0
      = 0x 7fffffff       = 0x 00000000
```

b.
```
media =          16 resto =         -16
      = 0x 7fffffff       = 0x 00000000
```

c.
```
media =  2147483647 resto =           0
      = 0x 00000010       = 0x fffffff0
```

d.
```
media =          16 resto =         -16
      = 0x 00000010       = 0x fffffff0
```

---

6. ¿Cuál expresión es cierta?

a. popcount(15) < popcount(51)
b. popcount(2) == popcount(64)
c. popcount(7) > popcount(60)
d. popcount(96) != popcount(3)

---

7. La práctica "popcount" debía calcular la suma de bits (peso Hamming) de los elementos de un array. Un estudiante entrega la siguiente versión de popcount1:

```
int pc1(unsigned* array, size_t len){
  size_t i,j;
  int res=0;
  unsigned x;
    for (i=0; i<len; i++){
      x = array[i];
      for (j=0; j<8*sizeof(int);j++){
        x >>= 1;
        unsigned bit = x & 0x1;
        res+=bit;
    } }
    return res;
}
```

Esta función se diferencia de la versión "oficial" recomendada en clase en el cuerpo del bucle interno. Esta función popcount1:

a. produce siempre el resultado correcto
b. fallaría con array={0,1,2,3}
c. fallaría con array={1,2,4,8}
d. no es correcta pero el error no se manifiesta en los ejemplos propuestos, o se manifiesta en ambos

---

8. En la misma práctica "popcount" un estudiante entrega la siguiente versión de popcount2:

```
int pcount2(int* array, size_t len){
  size_t i;
  int res=0;
  unsigned x;
  unsigned bit;
    for (i=0; i<len; i++){
      x = array[i];
      while(x){
        bit += x & 0x1;
        x >>= 1;
        res = res + bit;
    } }
    return res;
}
```

Esta función se diferencia de la versión "oficial" recomendada en clase en el tipo del **`array`**, la variable **`bit`** y el cuerpo del bucle interno. Esta función popcount1:

a. produce siempre el resultado correcto
b. fallaría con array={0,1,2,3}
c. fallaría con array={1,2,4,8}
d. no es correcta pero el error no se manifiesta en los ejemplos propuestos, o se manifiesta en ambos

---

9. En la misma práctica "popcount" un estudiante entrega la siguiente versión de popcount4:

```
int pc4(unsigned* array, size_t len){
  size_t  i;
  int res=0;
  unsigned x;
    for (i=0; i<len; i++){
      x = array[i];
     asm("\n\t"
        "clc            \n"
    "ini4:              \n\t"
        "adc  $0,  %[r] \n\t"
        "test %[x],%[x] \n\t"
        "shr  %[x]      \n\t"
        "jne  ini4      \n\t"
        "adc  $0,  %[r] \n\t"
        : [r]"+r" (res)
        : [x] "r" (x)   );
    }
    return res;
}
```

Esta función se diferencia de la versión "oficial" en que tiene una instrucción ensamblador adicional. Este popcount4:

a. produce siempre el resultado correcto
b. fallaría con array={0,1,2,3}
c. fallaría con array={1,2,4,8}
d. no es correcta pero el error no se manifiesta en los ejemplos propuestos, o se manifiesta en ambos

---

10. En la misma práctica "popcount" un estudiante entrega la siguiente versión de popcount4:

```
int pc4(unsigned* array, size_t len){
  size_t  i;
  int res=0;
  unsigned x;
    for (i=0; i<len; i++){
      x = array[i];
     asm("\n\t"
        "clc            \n"
    "ini4:              \n\t"
        "adc  $0,  %[r] \n"
    "fin4:              \n\t"
        "shr  %[x]      \n\t"
        "jne  ini3      \n\t"
        : [r]"+r" (res)
        : [x] "r" (x)   );
    }
    return res;
}
```

Esta función es muy diferente a la versión "oficial". Notar el salto condicional a "ini3" en la función popcount3 (en donde sí se hizo bien el **`ini3:/shr/adc/test/jne`** recomendado). Esta función popcount4:

a. produce siempre el resultado correcto
b. fallaría con array={0,1,2,3}
c. fallaría con array={1,2,4,8}
d. no es correcta pero el error no se manifiesta en los ejemplos propuestos, o se manifiesta en ambos

---

11. En la práctica de la bomba, el primer ejercicio consistía en saltarse las explosiones, para lo cual se puede utilizar… (marcar opción **falsa**)

a. objdump
b. gdb
c. ddd
d. eclipse

---

12. ¿Para qué se utiliza la función gettimeofday() en la práctica de la "bomba digital"?

a. Para cronometrar y poder comparar lo que tardan las distintas versiones del programa
b. Para imprimir la hora en la pantalla
c. Para cifrar la clave en función de la hora actual
d. Para lanzar un error cuando el usuario tarde demasiado tiempo en introducir la clave

---

13. ¿Para qué se utiliza la función scanf() en la práctica de la "bomba digital"?

a. Para escanear el fichero ejecutable “bomba” y asegurarse de que no contenga virus

b. Para leer la contraseña (clave alfanumérica)
c. Para leer el PIN (clave numérica)
d. Para lanzar un error cuando el usuario tarde demasiado tiempo en introducir la clave

14. Respecto a las bombas estudiadas en la práctica "bomba digital", ¿en cuál de los siguientes tipos de bomba sería más **difícil** descubrir la contraseña? Se distingue entre strings definidos en el código fuente de la bomba, y strings solicitados al usuario por teclado. Por "cifrar" podemos entender la cifra del César, por ejemplo.

a. 1 string del fuente se cifra, se invierte y se compara con el string del usuario
b. el string del usuario se cifra y se compara con 1 string del fuente
c. 2 strings del fuente se invierten, se concatenan, se cifra el resultado, y se compara con el string del usuario
d. el string del usuario se concatena con 1 string del fuente, luego se invierte 1 string del fuente, se cifra y se compara con el concatenado

15. En una bomba como las estudiadas en prácticas, del tipo...

```
0x40080f <main+180> lea 0xc(%rsp),%rsi
0x400814 <main+185> lea 0x1dd(%rip),%rdi
                                # 0x4009f8
0x40081b <main+192> mov    $0x0,%eax
0x400820 <main+197> call 0x400620<scanf>
0x400825 <main+202> mov    %eax,%ebx
0x400827 <main+204> test   %eax,%eax
0x400829 <main+206> jne 0x40083c <m+225>
0x40082b <main+208> lea 0x1c9(%rip),%rdi
                                # 0x4009fb
0x400832 <main+215> mov    $0x0,%eax
0x400837 <main+220> call 0x400620<scanf>
0x40083c <main+225> cmp    $0x1,%ebx
0x40083f <main+228> jne 0x4007f9 <m+158>
0x400841 <main+230> mov 0x200819(%rip),
                              %eax # 0x601060
0x400847 <main+236> cmp %eax,0xc(%rsp)
0x40084b <main+240> je  0x400852 <m+247>
0x40084d <main+242> call 0x400727 <boom>
0x400852 <main+247> lea 0x10(%rsp),%rdi
```

...el código numérico (pin) es...

a. el entero 0x601060
b. el entero cuya dirección está almacenada en la posición de memoria 0x4009f8
c. el entero almacenado a partir de la posición de memoria 0x4009fb
d. el entero almacenado a partir de la posición de memoria 0x200819+0x400847

16. La función **setup()** de Arduino es llamada:

a. Al principio de cada iteración de la función loop()
b. Cuando se sube desde el ~~kit~~ entorno de desarrollo o se pulsa el botón de reset, pero no cuando se conecta la alimentación
c. Cuando se conecta la alimentación a la placa, se pulsa el botón de reset, o se sube desde el ~~kit~~ entorno de desarrollo
d. Cuando se conecta la alimentación a la placa, se pulsa el botón de reset, pero no cuando se sube desde el ~~kit~~ entorno de desarrollo

17. ¿Qué sentencia usamos en el programa **blink** (led intermitente) para encender el led integrado en la placa Elegoo Mega2560?

a. digitalWrite (LED_BUILTIN, HIGH);
b. pinMode (LED_BUILTIN, OUTPUT);
c. analogWrite (LED_BUILTIN, HIGH);
d. pulseIn (LED_BUILTIN, HIGH);

18. Sobre el resultado devuelto por la función de Arduino **map(sensorValue, sensorLow, sensorHigh, 50, 4000);** usada en el programa del Theremín de luz:

a. Si sensorValue vale 50, devuelve sensorLow
b. Si sensorValue vale 50, devuelve sensorHigh
c. Si sensorValue vale ~~2520~~2025, devuelve la mitad entre sensorLow y sensorHigh
d. Si sensorValue vale la mitad entre sensorLow y sensorHigh, devuelve ~~2520~~2025

19. En el programa line.cc de la práctica de cache, si para cada tamaño de línea (`line`) recorremos una única vez el vector, la gráfica resultante es decreciente porque:

a. hay un mayor historial de accesos y es más probable que un nuevo acceso sea un acierto
b. cada vez los tamaños de línea escogidos van decreciendo y se tarda menos en leerlos
c. cada vez los tamaños de línea escogidos van decreciendo y hay menor localidad espacial
d. el vector se indexa con la variable de control del bucle, con un incremento o paso de `line`

20. En la práctica de la cache, en size.cc se accede al vector saltando de 64 en 64. ¿Por qué?

a. Para recorrer el vector más rápidamente
b. Porque con un salto menor que 64 habría aciertos por localidad espacial y haría menos clara la gráfica
c. Porque cada elemento del vector ocupa 64 B
d. Para evitar aciertos por localidad temporal y que sólo haya aciertos por localidad espacial