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