2º Grado Informática
Estructura de Computadores
31 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

1. Habiendo definido en código fuente ensamblador **`saludo: .ascii "Hola mundo\n"`**, si comparamos los comandos gdb siguientes: **`x/s &saludo`** frente a **`print (char*) &saludo`**, se puede decir que:

a. ambos nos muestran el string (el mensaje "Hola mundo")
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*) ó (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 **mayor** valor que repetido en toda la lista **no** 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 **`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: .int    0
resto: .int    0
…
main: .global  main
   mov     $lista, %rbx
   mov  longlista, %ecx
   call suma
   mov  %rax, media
   mov  %edx, resto
   …
suma:
   mov  $0, %rax
   mov  $0, %rsi
   mov  $0, %edx
   mov  $0, %rdi
   mov  $0, %ebp
bucle:
   movl  (%rbx,%rsi,4), %eax
   cltq                 # EAX -> RAX
   add  %rax, %rdi
   #adc  %edx, %ebp
   inc  %rsi
   cmp  %rsi,%rcx
   jne  bucle
   mov  %rdi, %rax
   #mov  %ebp, %edx
   idiv %rcx
   ret
```

Este programa se diferencia de la versión "oficial" recomendada en clase en que se guarda como media todo RAX, la subrutina no se llama **sumaq**, se inicializan a 0 más registros y no se usa CQTO.

¿Qué media y resto calcula este programa para el test #02? (16 elems. con valor -1,-2,-1,-2…)

a. media = -1, resto = -8
b. media = -2, resto = 8
c. media = -1, resto = 8
d. media = -2, resto = -8

---

4. En la misma práctica "media" (**`cltq/cdqe`** y **`cqto/cqo`**), un estudiante entrega la siguiente versión:

```
sumaq:
   mov   $0, %edi
   mov   $0, %esi
```

```
bucleq:
   mov   (%rbx,%rsi,4), %eax
   cltq              # EAX -> RAX
   add   %rax, %rdi
   inc   %esi
   cmp   %esi, %ecx
   jne   bucleq

   mov   %rdi, %rax
   cqto              # RAX -> RDX:RAX
#idiv/idivq %rcx no hace la division,
#queda todo como resto
   idiv  %ecx
   ret
```

Esta subrutina se diferencia de la versión "oficial" recomendada en clase en que se nombran EDI, ESI y ECX. El propio estudiante comentó alguna de las diferencias. ¿Qué media y resto calcula este programa para el test #03? (16 elementos con valor 0x7fffffff)

a. media = 0x0fffffff, resto = 0
b. media = 0x7fffffff, resto = 0
c. media = 0xffffffff, resto = 0
d. resto distinto de cero

---

5. En la misma práctica "media" (**`cltq/cdqe`** y **`cqto/cqo`**), un estudiante entrega la siguiente versión:

```
…
media:         .int 0
resto:         .int 0
formato:       .asciz "…"
formatoq:      .asciz "…"
…
   mov     $lista, %rbx
   mov  longlista, %ecx
   call sumaq
   mov   %rax, media
   mov   %rdx, resto
   …
sumaq:
   mov  $0, %rax
   mov  $0, %rsi
   mov  $0, %rdx
   mov  $0, %rdi
   mov  $0, %rbp
bucleq:
   mov  (%rbx,%rsi,4), %rax
   cqo               # RAX -> RDX:RAX
   add   %rax, %rdi
   adc   %rdx, %rdi

   inc   %rsi
   cmp   %rsi,%rcx
   jne    bucleq

   mov   %rdi, %rax
#  mov   %rbp, %rdx
   mov  $0, %ebp
   mov  %ecx,%ebp
   idiv %ebp
   ret
```

Este programa es muy diferente a la versión "oficial" recomendada en clase, destacando especialmente los **`mov media/resto`** tras la llamada a **`sumaq`**, los **`mov $0`** adicionales superfluos, mover elementos a **`%rax`**, usar **`cqo`** en lugar de **`cltq`** y no usarlo antes de **`idiv %ebp`**, que tampoco está bien, y además sobra el **`adc`**.

Este programa calcula el resultado correcto:

a. para lista: .int 1, 2, 1, 2, … (16 elementos)
b. para lista: .int -1,-2,-1,-2, … (16 elementos)
c. para ambos ejemplos
d. para ninguno de los dos ejemplos

---

6. 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 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"
          "shr  %[x]        \n\t"
          "adc  $0,  %[r]   \n\t"
          "test %[x],%[x] \n\t"
          "jne  ini4        \n"
     "fin4:                 \n\t"
          "clc              \n\t"
          : [r]"+r" (res)
          : [x] "r" (x)     );
    }
    return res;
}
```

Esta función es una mezcla de las versiones "oficiales" recomendadas en clase para popcount3 y popcount4, con alguna instrucción cambiada. 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

---

7. 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"
          "shr  %[x]        \n\t"
          "jnz  ini4        \n\t"
```

```
        : [r]"+r" (res)
        : [x] "r" (x)     );
    }
    return res;
}
```

Esta función se diferencia de la versión "oficial" recomendada en clase en que le falta una instrucción ensamblador final (última).

Esta función popcount4 (pensar bien qué pasa con el elemento 0 y con el elemento 1):

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 popcount5:

```
int pc5(unsigned* array, int len){
  size_t i, k, x;
  int  res=0;
    for (i=0; i<len; i++){
      long val = 0;
      x = array[i];
      for (k=0; k<8; k++){
         val+= x& 0x0101010101010101L;
         x >>= 1;
      }
      val += (val >> 32);
      val += (val >> 16);
      val += (val >>  8);
      res += (val & 0xFF);
   }
   return res;
}
```

Esta función no sigue la adaptación a 32 bits recomendada en clase como versión "oficial", y conserva muchos vestigios de la versión de 64 bits, como el tipo de **`val`**, **`len`** y **`x`**, la constante **`0x01…01L`** y el desplazamiento **`>>32`**. Esta función popcount5:

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 popcount5:

```
int pc5(unsigned* array, size_t len){
  size_t i,j;
  long val=0;
  unsigned long x;
    for (i=0; i<len; i++){
      x = array[i];
      for (j=0; j<8; j++){
         val+= x& 0x0101010101010101L;
         x >>= 1;
    } }
    val += (val >> 16);
    val += (val >> 8);
    val += (val >> 4);
    return val & 0xFF;
}
```

Esta función es muy diferente a la versión "oficial" recomendada en clase, destacando especialmente el tipo de **`val`** y **`x`**, la inicialización de **`val`**, la variable **`res`**, la constante **`0x01…01L`**, el anidamiento de los desplazamientos y el desplazamiento **`>>4`**.

Esta función popcount4 da resultado correcto:

a. con array={0,1,2,3}
b. con array={1,2,4,8}
c. con ambos ejemplos
d. con ninguno de los dos ejemplos

---

10. En la misma práctica "popcount" un estudiante entrega la siguiente versión de popcount7:

```
int pc7(unsigned* array, size_t len){
  size_t i;
  unsigned long x1,x2;
  int res=0;
const unsigned long m1 = 0x5555555555555555;
const unsigned long m2 = 0x3333333333333333;
const unsigned long m4 = 0x0f0f0f0f0f0f0f0f;
const unsigned long m8 = 0x00ff00ff00ff00ff;
const unsigned long m16= 0x0000ffff0000ffff;
const unsigned long m32= 0x00000000ffffffff;

  if (len & 0x3)
    printf("len no múltiplo de 4\n");

  for (i=0; i<len; i+=4){
    x1= *(unsigned long*)&array[i  ];
    x2= *(unsigned long*)&array[i+2];

    x1 = (x1 & m1 ) + ((x1 >> 1) & m1 );
    x1 = (x1 & m2 ) + ((x1 >> 2) & m2 );
    x1 = (x1 & m4 ) + ((x1 >> 4) & m4 );
    x1 = (x1 & m8 ) + ((x1 >> 8) & m8 );
    x1 = (x1 & m16) + ((x1 >>16) & m16);
    x1 = (x1 & m32) + ((x1 >>32) & m32);

    x2 = (x2 & m1 ) + ((x2 >> 1) & m1 );
    x2 = (x2 & m2 ) + ((x2 >> 2) & m2 );
    x2 = (x2 & m4 ) + ((x2 >> 4) & m4 );
    x2 = (x2 & m8 ) + ((x2 >> 8) & m8 );
    x2 = (x2 & m16) + ((x2 >>16) & m16);
    x1 = (x1 & m32) + ((x1 >>32) & m32);

    res += x1+x2;
  }
  return res;
}
```

Esta función sólo se diferencia de la versión "oficial" recomendada en clase en la última máscara, suma y desplazamiento **`>>32`**, que se realizan sobre una variable distinta. Esta función popcount4 da resultado correcto:

a. con array={3,2,1,0}
b. con array={8,4,2,1}
c. con ambos ejemplos

d. con ninguno de los dos ejemplos

11. Invocando a printf de libC (SystemV AMD64) desde ensamblador…

a. el primer argumento debe ponerse en %rax
b. el segundo argumento es el formato
c. si se desean imprimir tres **long**, el tercero debe ponerse en %rdx
d. si se desean imprimir cuatro **int**, el cuarto debe ponerse en %r8d

12. Recordando que los argumentos de `_start` (argc /argv y variables de entorno) se pasan en pila en SysV AMD64, si en el Ejemplo 1 de la Práctica 3 se hace **gdb -tui --args suma_01 uno dos tres**, se lanza con **br _start** y **run**, y se teclea **p * (char**)($rsp+40)**, ¿qué obtenemos?

a. tres
b. 0x0
c. una variable de entorno
d. un error de gdb

13. 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 eclipse
c. eclipse o ghex
d. ghex u objdump

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

```
0x40079b <main+64> lea  0x30(%rsp),%rdi
0x4007a0 <main+69> mov  0x2008d9(%rip),
                        %rdx # 0x601080
0x4007a7 <main+76> mov  $0x64,%esi
0x4007ac <main+81> call 0x400600 <fgets>
0x4007b1 <main+86> test %rax,%rax
0x4007b4 <main+89> je 0x400785 <main+42>
0x4007b6 <main+91> lea  0x30(%rsp),%rdi
0x4007bb <main+96> mov  $0xd,%edx
0x4007c0 <main+101>lea  0x2008a1(%rip),
                        %rsi # 0x601068
0x4007c7 <main+108>call 0x4005d0<strncmp>
0x4007cc <main+113>test %eax,%eax
0x4007ce <main+115>je 0x4007d5<main+122>
0x4007d0 <main+117>call 0x400727 <boom>
0x4007d5 <main+122>lea  0x20(%rsp),%rdi
```

...la contraseña (alfanumérica) es...

a. el string almacenado a partir de 0x601068
b. el string alm. a partir de 0x2008a1+0x4007c7
c. el string almacenado a partir de 0x30(%rsp)
d. no se puede marcar una y sólo una de las respuestas anteriores

15. En la práctica de E/S en Arduino, la instrucción CBI del repertorio del microcontrolador realiza…

a. una comparación
b. un complemento a uno
c. una operación de bits
d. una llamada a subrutina

16. En la práctica de E/S en Arduino, la instrucción SBI del repertorio del microcontrolador realiza…

a. una suma de un valor inmediato
b. una resta de un valor inmediato
c. una operación de bits
d. un salto incondicional

17. En la práctica de E/S en Arduino, DDRB es

a. el segundo canal de memoria DDR
b. el registro de dirección de datos del puerto B
c. el registro de datos y direcciones B
d. el registro buffer de datos D

18. En la práctica de E/S en Arduino, la instrucción SBIW del repertorio del microcontrolador realiza…

a. una suma de un valor inmediato
b. una resta de un valor inmediato
c. una operación de bits
d. un salto incondicional

19. En el programa line.cc de la práctica de cache, para cada tamaño de línea (`line`) recorremos una única vez el vector, pero podríamos haber realizado un número fijo y grande de accesos. Al dibujar la gráfica del tiempo de iteración en función del tamaño de línea…

a. de ambas formas sale gráfica creciente
b. recorriendo una vez sale gráfica creciente
c. con núm. fijo accesos sale gráfica creciente
d. de ninguna de las dos formas sale gr. creciente

20. En la práctica de la cache, en size.cc se realiza un número fijo y grande de accesos, pero podríamos haber recorrido una única vez el vector (saltando también de 64 en 64). Al dibujar la gráfica del tiempo de bucle en función del tamaño del vector…

a. de ambas formas sale gráfica creciente
b. recorriendo una vez sale gráfica creciente
c. con núm. fijo accesos sale gráfica creciente
d. de ninguna de las dos formas sale gr. creciente