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.
