UNIVERSIDAD DE GRANADA — ETSIIT Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

2º Grado Informática
Estructura de Computadores
20 de julio de 2017

ATC Departamento de Arquitectura y Tecnología de Computadores UNIVERSIDAD DE GRANADA

Nombre:	
DNI:	**Grupo:**

Examen de Problemas (3.0 p)

1. Ensamblador (0.3 puntos). Escriba la instrucción máquina (una sola) necesaria para llevar a cabo cada una de las siguientes operaciones:

a. (0.05) Quitar un entero de la cabecera de la pila y guardarlo en el reg. EAX:

b. (0.05) Sumar el valor del indicador de acarreo (CF) al registro ECX:

c. (0.05) Llamar a la función *f*:

d. (0.05) Copiar el carácter (byte) situado en la cabecera de la pila en el registro CL sin alterar la pila:

e. (0.05) Multiplicar el contenido del registro EAX por 5, dejando el resultado en el mismo registro:

f. (0.05) Poner el registro EDX a cero sin usar la instrucción mov: xorl %edx,%edx

2. Estructuras (0.8 puntos). Considerar las siguientes declaraciones de estructuras en lenguaje C:

```
struct a {
    float* f;
    char c;
    int i;
    char
    z[4];
    double d;
    short s;
};
```

```
struct b {
    struct a a1;
    int j;
    struct a a2;
};
```

a. (0.2) Indicar cuántos bytes ocupa struct a en Linux gcc x86:

b. (0.2) Indicar cuántos bytes ocupa struct a en Linux gcc x86-64:

c. (0.2) Indicar cuántos bytes ocupa struct b en Linux gcc x86:

d. (0.2) Indicar cuántos bytes ocupa struct b en Linux gcc x86-64:

3. Depuración (0.3 puntos). 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:

```
#define NBITS 20
#define SIZE ( 1<<NBITS )
unsigned lista[SIZE];
#define RESULT (NBITS*(1<<(NBITS-1)))

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;
}
```

Aunque para popcount3 se pedía una implementación **asm()**, esta función produce el resultado correcto con los tres ejemplos "pequeños" (4, 156 y 116), pero falla con el ejemplo "grande" (`lista={0,1, … 1048575}`) con el cual produce el resultado 237. Es posible razonar que esta función produce el resultado correcto (**NBITS**$\cdot 2^{\mathbf{NBITS-1}}$) siempre que queden sin efecto las partes erróneas del código (`val>>16`, `val>>8` y `val&0xFF`), lo cual sucede para valores de NBITS menores que o iguales a…

NBITS <=

4. Unidad de Control (0.4 puntos). El contenido de la memoria de control de una unidad de control microprogramada es:

Dirección	Contenido
0000	0000 0010
0001	0011 0111
0010	0010 0100
0011	0100 1010
0100	0000 0011
0101	0000 0010
0110	0011 0111
0111	0100 1010
1000	0100 1010
1001	0011 0111
1010	0100 1010
1011	0011 0111
1100	0000 0011
1101	0011 0111

Calcule:

a. (0.1) El tamaño en bits total de dicha memoria de control:

b. (0.2) El tamaño en bits total que requeriría un diseño nanoprogramado:

c. (0.1) El ahorro en bits expresado en porcentaje:

5. Entrada/Salida (0.5 puntos). Complete el código ensamblador de una función

```
void write_value (unsigned char value)
```

que realice una operación de salida con consulta de estado. El puerto de estado, en la dirección 0x22C, puede leerse con la instrucción **`inb $0x22C, %al`**. El puerto de datos de salida, en la misma dirección 0x22C, puede escribirse con la instrucción **`outb %al, $0x22C`**. La consulta de estado consistirá en leer del puerto de estado mientras el bit 7 valga 1, o sea, leer el puerto hasta que el bit 7 valga 0. Sólo entonces puede procederse a escribir el dato pasado a la función en el puerto de datos de salida.

```
write_value:
        pushl %ebp
        movl %esp, %ebp

        ; Escribir código de salida programada
        ; con consulta de estado desde aquí...









        ; ...hasta aquí

        popl %ebp
        ret
```

6. Configuración de memoria (0.4 puntos). Disponemos de circuitos SRAM de 32 K x 4 + EPROM de 16 K x 4 y queremos construir una memoria direccionable por bytes con un bus de datos de 8 bits y la siguiente configuración: SRAM de 64 K x 8 a partir de la dirección 0xE0000, EPROM de 32 K x 8 a partir de la dirección 0x00000. A cada chip SRAM se conectan las patillas de dirección A_{14}-A_0. A cada chip EPROM se conectan las patillas del bus de direcciones A_{13}-A_0.

a. (0.1) ¿Cuántos circuitos SRAM necesitamos?

b. (0.1) ¿Cuántos circuitos EPROM necesitamos?

c. (0.1) ¿Cuál es la última dirección de la SRAM?

d. (0.1) ¿Cuál es la última dirección de la EPROM?

7. Memoria cache (0.3 puntos). La siguiente tabla indica la frecuencia de fallos de varias caches:

Tamaño	Cache sólo instrucciones	Cache sólo datos	Cache unificada
8 KB	5.8%	6.8%	8.3%
16 KB	3.6%	5.3%	5.9%
32 KB	2.2%	4.0%	4.3%

El porcentaje de referencias a instrucciones es 53%. Se desea conocer cuál de estas dos configuraciones tiene una frecuencia de fallos menor:

• Configuración 1: una caché de instrucciones de 16 KB + una caché de datos de 16 KB

• Configuración 2: una caché unificada de 32 KB.

a. (0.1) Frecuencia de fallos para configuración 1:

b. (0.1) Frecuencia de fallos para configuración 2:

c. (0.1) ¿Cuál tiene una frecuencia de fallos menor, separada o unificada?