Nombre:	
DNI:	**Grupo:**

Examen de Problemas (3,0 p)

1. Ensamblador. (0.7 puntos). Compilando con gcc -Os la siguiente función C:

```
unsigned ndof (unsigned y) {
    return fórmula-oculta;
}
```

se obtiene este código ensamblador de x86-64:

```
ndof:
    movl $28, %eax
    testb $3, %dil
    jne .L1
    movl $100, %ecx
    movl %edi, %eax
    xorl %edx, %edx
    divl %ecx
    movl $29, %eax
    testl %edx, %edx
    jne .L1
    xorl %edx, %edx
    movl $400, %ecx
    movl %edi, %eax     ; Pista: como la instrucción sbb src, dst
    divl %ecx           ;                (Subtract with Borrow)
    cmpl $1, %edx       ;          calcula dst = dst – (src+CF),
    sbbl %eax, %eax     ; el objetivo de las 3 instr. cmp/sbb/not
    notl %eax           ; es que EAX termine valiendo o 0 o -1
    addl $29, %eax
.L1:
    ret
```

a) (0.1p) ¿Qué valor devuelve la función ndof al usar como argumento y=2020?

b) (0.5p) Escriba en C el cuerpo completo de la función ndof (sustituya "return *fórmula-oculta;*" por una sentencia o varias sentencias en C que realicen la misma función que el código ensamblador anterior).

c) (0.1p) ¿Qué calcula la función?

2. Ensamblador (0.4 puntos). La siguiente función realiza una serie de operaciones y cálculos sobre la siguiente estructura C.

```
struct rec {
     num_t1 i;
     num_t2 a[M][N];
     num_t1 j;
};

int fun(struct rec* r){
     return r->a[r->i][r->j];
}
```

num_t1 y num_t2 son tipos declarados con typedef. M y N son constantes enteras positivas declaradas con #define.

La traducción a ensamblador con gcc -Og de esta función es:

```
fun:
     movslq    16(%rdi), %rdx
     movslq    (%rdi), %rax
     leaq      (%rdi,%rax,4), %rax
     movsbl    4(%rdx,%rax), %eax
     ret
```

a) (0.2p) ¿Qué tipos son num_t1 y num_t2?

b) (0.2p) ¿Qué valores tienen las constantes M y N?

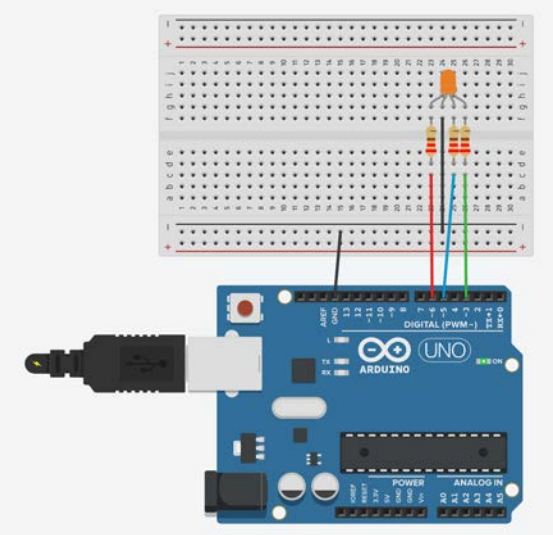
3. **Unidad de control** (0.6 puntos). La máquina de Tanenbaum estudiada en clase se implementó con un diseño horizontal de 79 microinstrucciones de 32 bits, y con un diseño vertical de 160 microinstrucciones de 12 bits. A ambos diseños se les podría haber aplicado nanoprogramación, pero un simple vistazo al microcódigo del diseño horizontal hace pensar que habrá del orden de 10-20 microinstrucciones repetidas como mucho. En el diseño vertical se ven más repeticiones, pero tampoco se acercan ni por asomo al 50% de microinstrucciones repetidas. Para fijar conceptos y entender por qué Tanenbaum no se planteó realizar nanoprogramación en su máquina:

   a) (0.3p) calcular qué porcentaje de microinstrucciones del diseño horizontal deberían estar repetidas para que la nanoprogramación fuera más ventajosa que el diseño vertical (en el sentido de ahorrar memoria de control)

   b) (0.3p) calcular qué porcentaje de microinstrucciones del diseño vertical deberían estar repetidas para que resultara ventajoso aplicarle nanoprogramación (en el mismo sentido)

   Notar que incluso si se diera ese porcentaje de repetición, la nanoprogramación implicaría una lectura adicional a la nanomemoria que retardaría la máquina aún más.

4. **Entrada/Salida** (0.4 puntos). Disponemos del siguiente circuito que conecta una placa Arduino con un led RGB. El led RGB dispone de tres patillas R, G y B conectadas a los pines 6, 3 y 5 de la placa Arduino. Complete el programa de Arduino para que el led RGB pase gradualmente de rojo a verde en 10 segundos aproximadamente, luego de nuevo de rojo a verde en otros 10 segundos, y así sucesivamente.




```
#define RED   6
#define GREEN 3
#define BLUE  5

void setup() {}

void loop() {

/* Escriba aquí el código */

}
```

Ayuda:

`analogWrite(pin, value)`

Escribe un valor analógico entre 0 y 255 en un pin. Puede usarse para variar el brillo de un led. No es necesario llamar a pinMode() para fijar un pin como salida antes de llamar a analogWrite().

`delay(ms)`

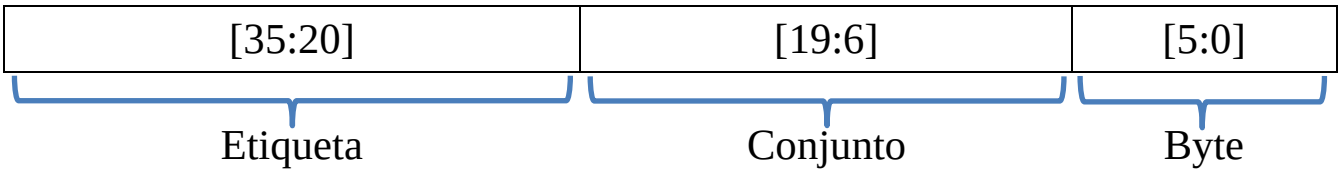
Pausa el programa la cantidad de milisegundos especificada como parámetro.

5. Configuración de memoria (0.5 puntos).

a) (0.4p) Partiendo de circuitos SRAM de 2Mx8, dibuje un sistema de memoria direccionable por palabras de 32 bits, con un bus de direcciones de 22 bits, un bus de datos de 32 bits, una señal Write y una señal Read.

b) (0.1p) Indique la primera y la última dirección en hexadecimal de cada fila de circuitos.

6. Cache (0.4 puntos). Las direcciones físicas de memoria principal del procesador Intel Core i7-10710U tienen el siguiente formato desde el punto de vista de la cache L3 de 12 MB:



Calcule:

a) (0.1p) Tamaño de memoria principal en bytes

b) (0.1p) Tamaño de línea en bytes

c) (0.1p) Tamaño de conjunto en bytes

d) (0.1p) Número de vías de la cache L3