	FUNDAMENTOS DE COMPUTADORES **Examen - 1 de febrero de 2018 FINAL**
	Nombre_____DNI_____

1. (1,5 puntos) Dados los siguientes números: A=+44, B= -73, C=-85 y D=70

a) Exprese los cuatro números con el mínimo número de bits en representación en complemento a dos.
b) Efectúe las operaciones A-B y C-D en complemento a dos, indicando si existe o no desbordamiento o acarreo.

2. (3 puntos) Se quiere diseñar un circuito digital que controle el funcionamiento de una máquina expendedora de caramelos. Dicho controlador recibe dos señales de entrada: *S* procedente de un sensor que toma el valor cero cuando se introduce una moneda de 5 céntimos y toma el valor uno si la moneda es de 10 céntimos, y R procedente de un pulsador que permite reiniciar el sistema quedando a la espera de otro cliente. Para expender un caramelo el controlador deberá activar la señal de salida *z*. La máquina se comporta de la siguiente manera:

- Cada caramelo cuesta 15 céntimos.
- El cliente puede ir introduciendo monedas en el orden que quiera.
- Cuando el saldo introducido alcanza o supera los 15 céntimos la máquina expende un caramelo, quedando almacenado el saldo restante por si el cliente quiere comprar otro caramelo. Por ejemplo, si un cliente introduce dos monedas de 10 céntimos seguidas, al introducir la segunda moneda la máquina expende un caramelo y deja almacenados los 5 céntimos sobrantes por si el cliente quiere seguir comprando.
- El cliente puede pulsar en cualquier momento un botón de reinicio y la máquina quedará a la espera de que algún nuevo cliente comience a usar la máquina.

Se pide:

a) Especificar el sistema mediante un diagrama de estados como máquina de Moore.
b) Implementar el sistema mediante un registro contador y puertas lógicas, indicando claramente las tablas de verdad que especifican las funciones de salida, transición de estados del sistema y señal load; y dibujando todo el sistema.
c) Especificar cómo se implementa el reinicio.

3. (3,5 puntos) Tenemos que realizar un programa en ensamblador que lee todos los componentes de un vector de 8 palabras de 32 bits que está inicializado en memoria a partir de la dirección VEC. Si el valor del componente es mayor o igual que 5 y menor de 10 guarda en una palabra de memoria a partir de la dirección SAL el índice de ese elemento del vector y en la siguiente palabra el valor de ese elemento.

A continuación se muestra la estructura del programa con la inicialización de variables de entrada y la reserva de espacio para las de salida.

```
        .global start

        .data
VEC: .word 0x00000006, 0x697A204E, 0x00000008, 0x00000005, 0x61632020, 0x00002120,
0x61611121, 0xFFFFFFFF

        .bss
SAL:    .space 64
        .text
start:
```

Código a realizar

a. Realizar el programa con bucles ascendentes
b. Realizar el programa usando un método que reciba el elemento del vector VEC que se quiere analizar y devuelva 0 si no cumple la condición y 1 si sí la cumple. Nota: Todos los registros que se modifiquen en el método deben salvarse en pila.
c. Si el SP tiene el valor 0x0000FF00 indicar cuál es el contenido de la pila (indicando claramente que valores se encuentran en ella y en qué direcciones) cuando se está ejecutando el método justo después de salvar todos los registros necesarios en la misma.
d. Suponiendo que se carga el programa en la dirección de memoria 0X00003000 indicar dónde se encuentra la primera instrucción del programa.

4. (2 puntos) Supongamos que el estado de un procesador ARM es el siguiente:

PC=0x00000100
Registro de estado: N=0, Z=1, C=0, V=0

Banco de registros:

Registro	Contenido
R0	0x FFFF 10FF
R1	0x 0000 0004
R2	0x 1111 203F
…	

La memoria contiene en cada palabra de dirección ***dir*** el valor ***dir***, es decir la palabra de dirección 0x00001000 contiene el valor 0x00001000, la palabra de dirección 0x000010FC contiene el valor 0x000010FC, etc.

a. Mostrar cómo varía el estado del procesador tras la ejecución de la instrucción:

```
UMULL R4, R3, R0, R1
```

b. Mostrar cómo varía el estado del procesador tras la ejecución de la instrucción:

```
ASRS R9, R2, #7
```

c. Mostrar cómo varía el estado del procesador tras la ejecución de la instrucción:

```
ORREQ R5, R0, R1
```

d. Mostrar cómo varía el estado del procesador tras la ejecución de la instrucción:

```
STR R5, [R2, R1, #8]
```