Apellidos: ..... Nombre: .....

DNI: .....

1. ¿Qué es la memoria de un computador y cuál es su función?

   La memoria es el dispositivo principal de almacenamiento del ordenador. Su función es la de almacenar los datos y las instrucciones del programa —o de los programas— en ejecución.

2. Comenta brevemente cómo interactúan los distintos componentes de un computador para ejecutar la instrucción «`ldrb r3, [r2, #7]`».

   En primer lugar, como con todas las instrucciones, el procesador activa las señales de control necesarias y envía el contenido del contador de programa a la memoria. Esta le devuelve la instrucción a ejecutar y el procesador la almacena y decodifica.

   Posteriormente, el procesador lee el valor 7 de la propia instrucción y el contenido del registro `r2` y, mediante su ruta de datos, realiza la suma de estos valores.

   Envía el resultado de la suma como dirección a la memoria y activa las señales de control necesarias para leer el byte contenido en dicha dirección.

   Cuando recibe dicho byte, lo almacena en el registro `r3`, poniendo a 0 sus 24 bits de mayor peso.

   Además, durante alguna de las etapas anteriores, el procesador habrá actualizado el contenido del contador de programa para que guarde la dirección de la instrucción siguiente, y se pueda continuar con la ejecución del programa.

3. Codifica en binario y en hexadecimal la instrucción «**`bgt pc, #232`**» sabiendo que el formato de instrucción utilizado para dicha instrucción es el que aparece a continuación y que la condición «`gt`» se codifica como 1100_2:

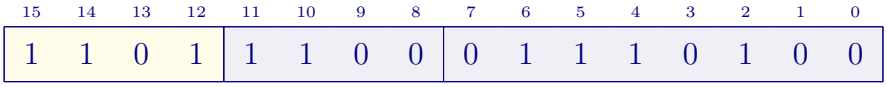
Cond Condición.

SOffset8 Dato inmediato con signo.

El contenido en binario de cada uno de los campos, de izquierda a derecha, será:

- `1101`, que es fijo y nos indica de qué tipo de instrucción se trata.
- `1100`, en el campo `Cond`, para indicar la condición «`gt`», tal y como se ha dicho en el enunciado.
- El campo `SOffsett8` codifica el desplazamiento, 232. Por tratarse de un salto, este número siempre será par, por lo tanto, se almacenará el número 232 dividido entre 2, que es 116. El número 116 se codifica en complemento a 2 con 8 bits como `01110100`.

Juntando los campos se tiene:



Que en hexadecimal es 0xDC74.

4. Describe el modo de direccionamiento de cada uno de los operandos de la instrucción «**`strh`** `r0, [r1, r2]`».

   La instrucción tiene dos modos de direccionamiento, uno para el operando fuente que se expresa como `r0` y otro para el operando destino, en memoria, que se expresa como `[r1, r2]`.

   El primer modo de direccionamiento es directo a registro, lo que indica que el valor se va a obtener de un registro, en particular, de `r0`.

   El segundo modo de direccionamiento es el relativo a registro con registro de desplazamiento. El registro `r1` es el registro base y el `r2`, el desplazamiento. Para formar la dirección efectiva, se suma el contenido de `r1` con el de `r2`. Esto nos da el valor de la dirección de memoria en la que se encuentra el operando.

5. Dado el siguiente programa en ensamblador Thumb de ARM, **describe su funcionamiento comentando cada una de sus líneas** y anota a continuación el contenido final de los registros `r0` y `r6` y los contenidos de las direcciones de memoria `0x2007 002C` (etiquetada como «`res`») y `0x2007 0040`.

```
1         .data
2  v1:    .word 3, 6, 2, 1, 4             @ Vector de repeticiones
3  v2:    .word 154, 89, -11, 609, 33     @ Vector de valores
4  tam:   .word 5                         @ Tamaño de ambos
5  res:   .space 1000                     @ Resultados: número de enteros escritos
6                                         @  y el vector creado con dichos enteros
7
8         .text
9  main:
10        ldr r5, =v1     @ r5 recorrerá el vector de repeticiones
11        ldr r6, =v2     @ y r6 el de valores.
12        ldr r7, =tam
13        ldr r2, [r7]    @ r2 guarda su tamaño y servirá de contador principal
14        add r7, #8      @ r7 apunta a la dirección para escribir los enteros
15        mov r3, #0      @ r3 es el contador de enteros escritos
16 b1:
17        ldr r0, [r6]    @ r0 guarda el valor
18        ldr r1, [r5]    @ r1 el número de veces a escribirlo
19 b2:
20        str r0, [r7]    @ Se escribe el valor
21        add r7, r7, #4  @   y se pasa a la dirección de escritura siguiente
22        add r3, r3, #1  @ Incrementa el contador de enteros
23        sub r1, r1, #1  @ Decrementa el de repeticiones
24        bne b2          @   hasta completar las escrituras
25        add r6, r6, #4  @ Pasa al valor siguiente
26        add r5, r5, #4  @   y al número de repeticiones siguiente
27        sub r2, r2, #1  @ Decrementa el número de elementos restantes
28        bne b1          @   hasta que llega al final
29        ldr r7, =res
30        str r3, [r7]    @ Escribe en res el número de enteros escritos
31        wfi
32
33        .end
```

El programa propuesto recorre los vectores v1 y v2, ambos del mismo tamaño —indicado por la variable tam— y escribe en el espacio reservado a partir de la dirección «`res + 4`» cada entero del vector v2 tantas veces como indique el entero del mismo índice en v1.

Para ello, `r5` recorre v1, `r6`, v2 y `r7`, el vector destino. El registro `r3` cuenta el número de enteros escritos, así que al terminar el programa, su valor será la suma de los elementos de v1, es decir, 16. Como este valor se almacena en «`res`», el contenido de la dirección `0x2007 002C` será 16.

`r0` almacena los elementos de v2, por lo que al terminar tendrá el valor del último elemento, es decir, 33.

`r6` recorre los elementos de v2 y se incrementa antes de verificar el final del bucle, por lo que al terminar tendrá la primera dirección fuera del vector, la de la etiqueta «`tam`», `0x2007 0028`.

Por último, los valores se almacenan a partir de «`res + 4`», así que en la dirección `0x2007 0040` se habrá escrito el quinto entero (la dirección dista 16 bytes, es decir, 4 enteros, de «`res + 4`»). Como 154 se escribe 3 veces y 89, 6, el quinto entero escrito es 89.

`r0`	33
`r6`	0x20070028
`0x2007 002C`	16
`0x2007 0040`	89