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

DNI: .....

1. ¿Cuál es la función del procesador de un computador?

   El procesador es el encargado de ejecutar las instrucciones almacenadas en la memoria principal y de generar las señales de control que rigen el funcionamiento del ordenador.

2. Comenta brevemente cómo interactúan los distintos componentes de un computador para ejecutar la instrucción «`strh r5, [r1, r2]`».

   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 contenido de los registros `r1` y `r2` y, mediante su ruta de datos, realiza la suma de estos valores.

   A continuación, envía el resultado de la suma como dirección a la memoria y activa las señales de control necesarias para escribir en dicha dirección la media palabra de menor peso contenida en el registro `r5`.

   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 «**`strh`** `r2, [r5,#8]`» sabiendo que el formato de instrucción utilizado para dicha instrucción es el que aparece a continuación:

L Load/Store bit: 0, almacenar en memoria; 1, cargar.

Offset5 Dato inmediato.

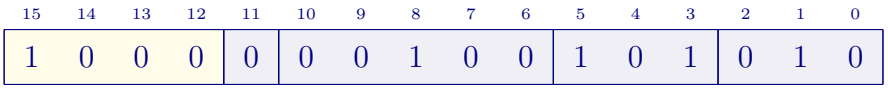
Rb Registro base.

Rd Registro fuente/destino.

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

- `1000`, que es fijo y nos indica de qué tipo de instrucción se trata.
- El campo `L` es un 0, pues se trata de una instrucción de almacenamiento.
- El campo `Offset5` codifica el desplazamiento, 8. Por tratarse de una instrucción de almacenamiento de una media palabra, la dirección de la media palabra siempre será par, por lo tanto, en el campo correspondiente al desplazamiento se almacenará el número 8 dividido entre 2, que es 4. El número 4 se codifica en complemento a 2 con 5 bits como `00100`.
- El registro base es `r5`, `101`.
- El registro fuente es `r2`, `010`.

Juntando los campos se tiene:



Que en hexadecimal es 0x812A.

4. Describe el modo de direccionamiento de cada uno de los operandos de la instrucción «**`ldr`** `r0, [r1, #8]`».

   La instrucción tiene dos modos de direccionamiento, uno para el operando destino, que se expresa como `r0`, y otro para el operando fuente, en memoria, que se expresa como «`[r1, #8]`».

   El primer modo de direccionamiento es directo a registro, lo que indica que el operando destino es un registro, en particular, el `r0`.

   El segundo modo de direccionamiento es el relativo a registro con desplazamiento. El registro `r1` es el registro base y el valor inmediato `8`, el desplazamiento. Para formar la dirección efectiva, se suma el contenido de `r1` y `8`. Esto nos da el valor de la dirección de memoria en la que se encuentra el operando fuente.

5. Dada la siguiente subrutina en ensamblador Thumb de ARM:

   - Describe su funcionamiento comentando adecuadamente cada una de sus líneas.
   - Indica qué valores habrá en el momento de ejecutar la instrucción «`mov pc, lr`» en los registros y posiciones de memoria indicados si cuando se ha llamado a la subrutina: en `r0` estaba la dirección de comienzo de un vector de palabras V = [1, 2, 3, 4, 5] (`0x2007 0000`), en `r1`, su tamaño (5) y en `r4`, el número 42.

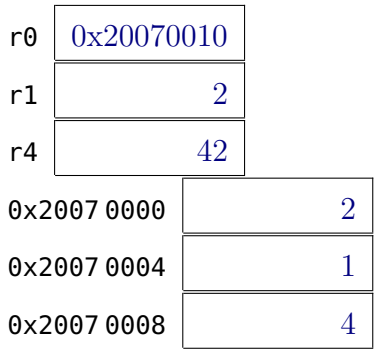
```
1  subr:   push {r4}         @ Apila el registro r4
2          mov r2, #0        @ r2 <- 0 (índice i)
3          asr r1, #1        @ r1 <- r1>>1 (divide r1 entre 2)
4  sw:     cmp r2, r1        @ r2 - r1
5          bge swe           @ si r2 >= r1, salta a swe
6          ldr r3, [r0]      @ r3 <- V[i]
7          ldr r4, [r0, #4]  @ r4 <- V[i+1]
8          str r3, [r0, #4]  @ V[i+4] <- r3
9          str r4, [r0]      @ V[i] <- r4
10         add r0, r0, #8    @ r0 <- r0 + 8, direción de V[i+2]
11         add r2, r2, #1    @ r2 <- r2 + 1
12         b sw              @ Salta a sw
13 swe:    pop {r4}          @ Desapila r4
14         mov pc, lr        @ Retorna de la subrutina
```

La subrutina propuesta recorre los elementos de un vector V de dos en dos, intercambiando sus valores.

El registro `r1` indica inicialmente el número de elementos del vector, que se desplaza un bit a la izquierda al comienzo de la subrutina, por lo que acabará valiendo $5 >> 1 = 2$. Esta operación también se puede ver como la división entera de 5 entre 2. El contenido de este registro se utilizará para determinar cuántas veces se tiene que ejecutar el bucle etiquetado con «`sw`».

El registro `r0` mantiene la dirección de comienzo del siguiente par de elementos que se van a intercambiar. Como el bucle «`sw`» se ejecutará 2 veces, acabará tomando el valor `0x2007 0010` (`0x2007 0000` + 8 + 8).

El registro `r4` contiene el valor 42 al comenzar la ejecución de la subrutina, este valor se guarda en la pila en la primera instrucción de la subrutina y se recupera justo antes de la instrucción de retorno de la subrutina, por lo que independientemente de para qué se haya utilizado en la subrutina, acabará valiendo 42.


r0
0x20070010
r1
2
r4
42
0x2007 0000
2
0x2007 0004
1
0x2007 0008
4