Computer Architectures **02LSEOV 02LSEOQ [AA-LZ]**	Delivery date: Thursday 6/12
Laboratory **7**	Expected delivery of lab_07.zip must include: - zipped project folder of the exercises 1 and 2 - this document compiled possibly in pdf format.

Solve the following 2 problems by starting from the *template.s* file.

Exercise 1) The owner of a shop is restocking shelves. He made a list with items to buy; the items are identified by means of a code. He wants to know the total amount of the expense according to the price list of his wholesaler. Write a program in ARM assembly language to compute the amount of the expense according to the following details.

The price list of the wholesaler is a table where each entry consists of two integer values: the identification code of the product (4 bytes) and the price (4 bytes). The example table is composed of 28 records (couples of 32-bit values) ordered according to the identification code. For example:

```
Price_list DCD 0x004, 120,  0x006, 315,  0x007, 1210, 0x00A, 245
           DCD 0x010, 228,  0x012, 7,    0x016, 722,  0x017, 1217
           DCD 0x018, 138,  0x01A, 2222, 0x01B, 34,   0x01E, 11
           DCD 0x022, 223,  0x023, 1249, 0x025, 240,  0x027, 112
           DCD 0x02C, 2245, 0x02D, 410,  0x031, 840,  0x033, 945
           DCD 0x036, 3211, 0x039, 112,  0x03C, 719,  0x03E, 661
           DCD 0x042, 230,  0x045, 1112, 0x047, 2627, 0x04A, 265
```

The list and the number of items to buy is stored in two more pools.
The Item_list is composed of two integer values for each entry: the identification of the product (4 bytes) and the desired quantity (4 bytes); they are not ordered. The Item_num is a 1 byte constant. For example:

```
Item_list  DCD 0x022, 14, 0x006, 431, 0x03E, 1210, 0x017, 56342
Item_num   DCB 4
```

A binary search should be implemented for searching the price of the items in the `Price_list` table, multiply by the requested quantity and accumulate the resulting value in register R10. The C code of the binary search that searches for `key` is:

```
first = 0
last = num_entries - 1;
while (first <= last)
      {
      middle = (first + last) / 2;
      if (key == table[middle])
      {
            /* element found */
            index = middle;
            break;
      }
      else
            if (key < table[middle])
                  last = middle - 1;
            else
                  first = middle + 1;
}
```

At the end of the program: register R10 should contain the amount of the expense if all products are available in the price list; otherwise, if at least one product is missing in the price list, R10 should be 0.
Respond to the following open questions.

Q1: are there criticalities of the program in terms of dimension of numbers?

In this case, there are no criticalities in terms of dimension of numbers: all the multiply operations can in fact be represented by 32-bits values, and so it can be also the total result (0x0424975F).

Q2: if yes, which counter-measurement can be taken with respect to a basic implementation?

If there were any criticalities, it would have been necessary to work with longer, 64-bits values. In order to achieve that on a 32-bits processor like the ARM Cortex M3, we would have needed two registers for storing the sum and (eventually) every partial result given by the multiplications; instructions to be used in this case include the UMULL for computing the intermediate result and the ADC to sum the upper words of the resulting double word.

Exercise 2) Create a new project by starting from the previous exercise and suppose that the entries in the price list are not sorted (unsort on purpose the list of values you have used in exercise 1); you have to implement first a sorting algorithm and then to perform the same calculation function of exercise 1.

Use the sorting algorithm you prefer to reorder the list of values (i.e., use algorithms you studied in previous courses or search on the web).

Remember that:

1. the sort routine must sort both the key and the data associated with each entry
2. it is not possible to reorder the content of a portion of memory in a read-only section, but a proper area needs to be allocated in a read-write section and used to store/order the list.

Report the selected values in the table below.

	Execution time	Code size	Data size
Exercise 1)	31.83 µs	564 B	RO: 204 B RW: 0 B
Exercise 2)	321.25 µs	564 B	RO: 204 B RW: 512 B

Respond to the following open questions.

Q1: what section have you used to store the ordered sequence?

The heap section was used to store the ordered sequence, which supports read and write operations.

Q2: which address range is assigned to this section and which memory of the system is used?

The assembler assigns to the heap 512 bytes (total heap dimension, only a smaller part of them are used to store the price list) starting from the address 0x10000000. This address space is mapped to the on-chip RAM memory.

Q3: describe what is the worst case of initial order (the one who takes the most to sort) and motivate.

For the selected sorting algorithm, which is a basic insertion sort, the worst-case scenario happens when the initial vector is sorted in the reverse order. In this case, every i-th value needs to be compared with all the $i-1$ previous ones, resulting in an overall $O(n^2)$ complexity.