G51CSA Assessed Coursework: Bubble Sort

Steven R. Bagley

Introduction

In this coursework, you are to write an ARM assembly program that iterates over an array of ten values and sorts them into numerical order. To do this, you will implement the classic **Bubble sort** algorithm. Bubble sort is not an efficient sorting algorithm by any stretch of the imagination, but it is simple to implement. This algorithm works by repeatedly switching pairs of numbers around in the array if they are out-of-order until the list is sorted. For example, consider this list:

```
10, 25, 13, 44, 9, 15, 6, 27, 36, 42
```

The algorithm would first compare the first pair of numbers in the list, `10` and `25`, to see if they are in order. In this case, they are (i.e. `10` is less than `25`), so they are left unchanged. The next two numbers are then compared, `25` and `13`. This pair is out of order (`13` comes before `25`) so their positions are swapped over in the array giving:

```
10, 13, 25, 44, 9, 15, 6, 27, 36, 42
```

The next pair is compared, `25` and `44`, in the same manner and they are already in order. Then the next pair, `44` and `9`, are compared and then swapped over since they are out of order, giving:

```
10, 13, 25, 9, 44, 15, 6, 27, 36, 42
```

This continues until the nineth and tenth values are compared and swapped. The result is the following list (the intermediate stages can be found in Appendix One):

```
10, 13, 25, 9, 15, 6, 27, 36, 42, 44
```

The result is that the highest value in the list (`44`) has 'bubbled up' to be the last element of the list. We then repeat the same process to continue sorting the list, although since we know that the highest value is now in the tenth (last) position in the list, we only need to compare the first nine values. This process continues until either we are only comparing the first element with itself, or we pass through the entire list without swapping any values.

For more information on Bubble sort, you could see the Wikipedia page for it: `http://en.wikipedia.org/wiki/Bubble_sort` or watch my former PhD student, Alex Pinkney, in a video for *Computerphile* explaining various sorting algorithms: `https://www.youtube.com/watch?v=kgBjXUE_Nwc` †

In the next session with your preceptor you will look at a C solution to this problem. The relevant parts of that solution can be found in Appendix Two.

Problem Descritpion

Your program should store the list of ten integers in memory using `DEFW` (see below). You can assume that there will only be complete lists, and that each list contains exactly ten integers. Your ARM program should loop through each pair of numbers in the list, read them from memory and if they are out of order (i.e. if the value at index n is greater than the value at index $n+1$), swap the values in the memory.

---

† If you'd prefer to dance Bubble Sort, `https://www.youtube.com/watch?v=lyZQPjUT5B4`

As you process each pair, you must keep track of the number of swaps you make as you go through the list. If at the end of the list, you have made zero swaps then you can stop processing otherwise you should continue processing the list to continue sorting it.

After your program has found the answer, it should print the sorted list using `SWI 4` (and others) like this:

```
6, 9, 10, 13, 15, 25, 27, 36, 42, 44
```

With the numbers replaced with the values calculated by your program.

ARM Implementation

Your program should be implemented as ARM machine code, with the list specified at the beginning of your source using `DEFW` like this:

```
            B main
list        DEFW 10,25,13,44,9,15,6,27,36,42
            Define any other variables you might need here…
main        ; Your code goes here
```

This lays each of the integers out in in memory one after the other. You can then access each of the ten values in the list individually by using the correct offset from the address of `list`, remembering that each integer value will take up four bytes in memory.

Before you implement your program, you will need to decide which register you are going to use to store what values (memory addresses, loop counters, values fetched from the list, results of temporary calculations, highest pair found etc.) and which, if any, you'll store in memory variables. You do **not** need to implement the routine as a function.

Testing

When marking your program, we will see that it produces the correct answer with different lists of numbers. Therefore, you probably want to check that your program works with a variety of input, such as each of the sample lists below:

```
10, 25, 13, 44,  9, 15,  6, 27, 36, 42
 4, 15, 47, 23, 18, 10, 22,  6, 37, 28
40, 19, 17, 23,  2, 43, 35, 21,  4, 34
 4, 25, 41, 48, 34, 20, 10, 19,  7, 16
39,  6, 10, 34,  8, 17, 23, 49, 38, 12
```

Hints and Tips

Even though the appendix contains a C solution to this problem, I recommend that you don't just attempt to convert it verbatim into assembler. You'll find it easier if you build it up bit by bit until you get a working program. When developing my own solution, I went through several stages before I had a working program. These included writing a program which printed out each of the values in the list (testing that my loops and memory accesses worked correctly), printing out each pair of values, printing out the pairs etc. This approach meant that *when* things didn't work as I expected I had a good idea of which area of the program the problem was in (because I knew that certain bits had already worked).

ARM assembler doesn't have an equivalent of `printf()` so you will need to print the output manually by using a combination of `SWI 3`, and `SWI 4` to print the string and integer parts respectively.

APPENDIX ONE

A full pass of the bubble sort algorithm for a sample array. The pair being compared and swapped is highlighted in bold.

Pass One

10, 25, 13, 44, 9, 15, 6, 27, 36, 42,
10, **25, 13,** 44, 9, 15, 6, 27, 36, 42,
10, 13, **25, 44,** 9, 15, 6, 27, 36, 42,
10, 13, 25, **44, 9,** 15, 6, 27, 36, 42,
10, 13, 25, 9, **44, 15,** 6, 27, 36, 42,
10, 13, 25, 9, 15, **44, 6,** 27, 36, 42,
10, 13, 25, 9, 15, 6, **44, 27,** 36, 42,
10, 13, 25, 9, 15, 6, 27, **44, 36,** 42,
10, 13, 25, 9, 15, 6, 27, 36, **44, 42,**
10, 13, 25, 9, 15, 6, 27, 36, 42, 44,

Pass Two

10, 13, 25, 9, 15, 6, 27, 36, 42, 44,
10, **13, 25,** 9, 15, 6, 27, 36, 42, 44,
10, 13, **25, 9,** 15, 6, 27, 36, 42, 44,
10, 13, 9, **25, 15,** 6, 27, 36, 42, 44,
10, 13, 9, 15, **25, 6,** 27, 36, 42, 44,
10, 13, 9, 15, 6, **25, 27,** 36, 42, 44,
10, 13, 9, 15, 6, 25, **27, 36,** 42, 44,
10, 13, 9, 15, 6, 25, 27, **36, 42,** 44,
10, 13, 9, 15, 6, 25, 27, 36, 42, 44,

Pass Three

10, 13, 9, 15, 6, 25, 27, 36, 42, 44,
10, **13, 9,** 15, 6, 25, 27, 36, 42, 44,
10, 9, **13, 15,** 6, 25, 27, 36, 42, 44,
10, 9, 13, **15, 6,** 25, 27, 36, 42, 44,
10, 9, 13, 6, **15, 25,** 27, 36, 42, 44,
10, 9, 13, 6, 15, **25, 27,** 36, 42, 44,
10, 9, 13, 6, 15, 25, **27, 36,** 42, 44,
10, 9, 13, 6, 15, 25, 27, 36, 42, 44,

Pass Four

10, 9, 13, 6, 15, 25, 27, 36, 42, 44,
9, **10, 13,** 6, 15, 25, 27, 36, 42, 44,
9, 10, **13, 6,** 15, 25, 27, 36, 42, 44,
9, 10, 6, **13, 15,** 25, 27, 36, 42, 44,
9, 10, 6, 13, **15, 25,** 27, 36, 42, 44,
9, 10, 6, 13, 15, **25, 27,** 36, 42, 44,
9, 10, 6, 13, 15, 25, 27, 36, 42, 44,

Pass Five

9, 10, 6, 13, 15, 25, 27, 36, 42, 44,
9, **10, 6,** 13, 15, 25, 27, 36, 42, 44,
9, 6, **10, 13,** 15, 25, 27, 36, 42, 44,
9, 6, 10, **13, 15,** 25, 27, 36, 42, 44,
9, 6, 10, 13, **15, 25,** 27, 36, 42, 44,
9, 6, 10, 13, 15, 25, 27, 36, 42, 44,

Pass Six

9, **6**, 10, 13, 15, 25, 27, 36, 42, 44,
6, **9**, **10**, 13, 15, 25, 27, 36, 42, 44,
6, 9, **10**, **13**, 15, 25, 27, 36, 42, 44,
6, 9, 10, **13**, **15**, 25, 27, 36, 42, 44,
6, 9, 10, 13, 15, 25, 27, 36, 42, 44,

Pass Seven

6, **9**, 10, 13, 15, 25, 27, 36, 42, 44,
6, **9**, **10**, 13, 15, 25, 27, 36, 42, 44,
6, 9, **10**, **13**, 15, 25, 27, 36, 42, 44,

APPENDIX TWO

```
void bubbles(int array[], int numElems)
{
    int max, i;
    int swap;
    int n;

    for(max = numElems - 1; max > 0; max--)
    {
        n = 0;
        for(i = 0; i < max; i++)
        {
            if(array[i] > array[i+1])
            {
                swap = array[i];
                array[i] = array[i+1];
                array[i+1] = swap;
                n++;
            }
        }
        if(n == 0)
        {
            break;
        }
    }
}
```