INFO0012-2 : Beta Assembly Project

Romain LAMBERMONT - s190931
Arthur LOUIS - s191230

December 17, 2021

1 Introduction

In this project, we implemented sorting algorithms using Beta Assembly and running our code on BSim. Our main goal was to make our code quite efficient. We tried to achieve this by reducing the number of registers used by our functions. We implemented the assembly version of the algorithms by smartly translate the functions from **.c** files by comparing and branching to loop on our array.

2 Bubblesort

To make our lives easier while implementing the Bubblesort, we defined some macros to make actions on the arrays :

- **SWAP** : swaps two elements in the array by using two temporary registers
- **ADDR** : returns the address of an element in an array using it's index and puts it in a specified register
- **LDR** : returns the value of an element in an array using its index and puts it in a specified register

We simply implemted our two loops and the condition using 3 comparaisons and branching pairs. We also created a function using the label **swap** to call the SWAP macro separetly. The rest of the code naively translates the **bubblesort.c** function by smartly using our macros and thus doesn't need further explainations.

3 Quicksort

To implement the Quicksort, we used the same macros and **swap** function defined earlier. To correctly implement the sorting algorithm, we had to create another function, **partition** that finds the pivot and puts values smaller than the pivot in a sub-array on the left and the values greater than the pivot in a sub-array on the right. Our function **quicksort** translated from the **quicksort.c** file recursively calls himself and the partition function to sort the array as predicted.

4 Comparaison of sorting algorithms

When testing our sorts on some large arrays (e.g 4096 elements), we distinguish a big leap in performance between the the sorts. Indeed, we can analyze, the time complexity of our algorithms.

- Bubblesort : The bubblesort goes through the entire array for each iterations (n steps for an array of length n) and it does n iterations. The time complexity of our algorithm is then : $\mathcal{O}(n^2)$ (best and worst case).
- Quicksort : Although the worst case is in $\mathcal{O}(n^2)$ (array sorted in reverse). In average, the time complexity for this sort is $\mathcal{O}(n\log(n))$ which is better than the bubblesort, as we can see in our tests.