Computation Structures Project 1

Sorting Algorithm in Beta-assembly

Pagano Florian s183627 Vinders Adrien s194594

Macros

For this project, we used macros in order to simplify the code

- .macro ADDR(Ra, Ri, Ro)
  - Compute the address of the element at index Ri in array Ra, return it in Ro
- .macro LDARR(Ra, Ri, Ro)
  - Load the value from index Ri of Array Ra in Ro
- .macro SWAP(Ra, Rb, RTMP1, RTMP2)
  - Swap the content of two addresses (puts in register Ra and Rb) in memory
- .macro SWAPARR(Ra, Rb, Rc, RTMP1, RTMP2)
  - Swap the content of Rb and Rc elements of array Ra in memory
  - Restore the registers used

Bubble Sort

In the beginning we push the values of the registers we will modify, and at the end we restore them.

For the implementation of the bubble sort algorithm, we use two different loops.

- Loop i runs through the array, and calls loop j
- Loop j runs through the array, and stop at the size – 1– i cell. It perform a swap of the addresses of array[j] and array[j+1] in memory. If the value of array[j] is greater than the value of array[j+1]

Loops are denoted by the labels bubble_sort_loop_i and bubble_sort_loop_j. bubble_sort_loop_i_2 is used to increment i and to go back at the start of loop i.

We have a condition (if (array[j] > array[j+1])). If this condition is met, we branch to the bubble_sort_swap label, this piece of code will perform the swap of the addresses of array[j] and array[j+1] in memory, and come back to the start of loop j.

Quick Sort

In the beginning we push the values of the registers we will modify, and at the end we restore them.

For the implementation of the bubble sort algorithm, we use two different functions.

- quick_sort calls partition, and calls quicksort on left and right subarrays
- quick_sort_partition does the main job of sorting the array (not an exact sort on only one call, but the fact that partition is called on each subarray make it sorts the whole array. It also computes the new pivot position, by adding one to the index of the last cell of the array that have been exchanged with the current pivot (small).

So the most important function is quick_sort_partition. It is composed of a main loop that ends when current arrives at the end of the array(quick_sort_partition_loop). It increases small and swap the address of the cell small and the cell current if the value of the current cell of the array is smaller than the value of the pivot cell.

At the end of the function we swap the cell of small + 1 with the last cell of the array, and we return the new position of the pivot (small + 1) in R0