Computation Structures
Project 1
Quicksort in β-assembly

October 11, 2016

General information

- **Deadline: November 1, 2016, 23:59**.
- Project must be done **individually**.
- There will be a penalty for late submission.
- Questions will no longer be answered 24 hours before the deadline.
- English is strongly encouraged.
- Contact: `r.mormont@ulg.ac.be`, Office I.128 (B28).

1 Introduction

The goal of this project is to make you more familiar with the β-assembly language by getting your hands dirty and writing code. Especially, you will implement the well-known Quicksort sorting algorithm.

2 Project

2.1 Quicksort

Quicksort is an algorithm created by C.A.R. Hoare in the late fifties for sorting arrays. Quicksort is a recursive procedure which works as follows:

1. **Base case**: if the array size is less than or equal to 1, the array is already sorted and nothing has to be done.
2. **Recursive case**:
   (a) Select a pivot value in the array (see below for the selection strategy)
   (b) Partition the array around this pivot value (see an example of partitioning in Figure 1)
   (c) Sort recursively the sub-arrays to the left and to the right of the pivot value using Quicksort

The pivot can be selected using various strategies:

- Picking a specific element (e.g. the first of the array)
- Picking an element at random
- Picking the median element among three (e.g. the first, the element at the middle of the array and the last element)

For more information about Quicksort, you can check online resources:

- Pierre Geurts's course on data structures and algorithms: `http://www.montefiore.ulg.ac.be/~geurts/Cours/sda/2015/sda-2015-2016.pdf` (from slide 121, in french)
- Explanation and interactive illustration: `http://me.dt.in.th/page/Quicksort/` (in english)
- Wikipedia: `https://en.wikipedia.org/wiki/Quicksort`
- ...

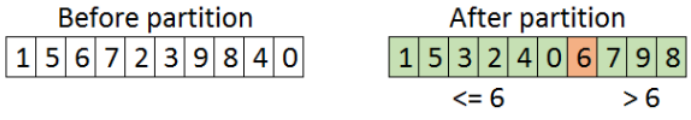

Figure 1: Example of array partitioning. Chosen pivot is value 6 at index 2 in the initial array.

2.2 Objective

You task is to implement a `quicksort` procedure in a file called `quicksort.asm`. This procedure should be callable using a label `quicksort` and should take two parameters:

1. `array`: the memory address of the first element of the array
2. `size` : the size of the array (`size` ≥ 0)

There will be a penalty for not conforming to this interface. Whatever you write in your `quicksort.asm` file, your procedure should be callable using the example code shown in Listing 1.

Listing 1: Example of call to the quicksort procedure

```
|; Reg[R1]: address of array[0]
|; Reg[R2]: size of the array
PUSH(R1)
PUSH(R2)
CALL(quicksort)
DEALLOCATE(2)
```

You are provided with a file `quicksort.c`, a C implementation of Quicksort, that you can use as basis for your implementation. While you are **not** bound to translate all implementation details contained in this file in your assembly code (see Section 2.3), you are still expected to implement the recursive Quicksort procedure presented in Section 2.1.

In addition to `quicksort.c`, you are provided with three other assembly files:

- `beta.uasm`: the definition of the β-assembly. You can check this file to see which macros you can use in your own code.
- `main.asm`: you can use this file to test your code. It does the following:
  1. It reserves some memory for an array of size n (starting at address `0x0008` in the DRAM).
  2. It initializes it with increasing values ranging from 1 to n.
  3. It shuffles the array.
  4. It calls the `quicksort` procedure that you must define.
- `util.asm`: contains some code needed by the `main.asm` file (for filling the array and shuffling it).

2.3 Bonus

You can choose any pivot selection strategy for your implementation, the simpler being picking the first element of the array. However, implementing the median of three strategy will earn you a bonus. Using this strategy induces the modifications in the C implementation presented in Listing 2.

Listing 2: Modifications to the C implementation of Quicksort for using the median of three pivot selection strategy.

```
/**
 * Return the address of the median of the values located at the given addresses
 *
 * @param first  int* Address of the first element
 * @param second int* Address of the second element
 * @param third  int* Address of the third element
 *
 * @return int* Address of the median element
 */
int* median_of_three(int* first, int* second, int* third) {
    if (*first < *second) {
        if (*third <= *first) {
            return first;
        } else if (*second < *third) {
            return second;
        } else {
            return third;
        }
```

```
    } else {
        if (*first <= *third) {
            return first;
        } else if (*second < *third) {
            return third;
        } else {
            return second;
        }
    }
}

void quicksort(int* array, size_t size) {
    // ...
    int* pivot_addr = median_of_three(
        array,
        array + (size / 2) - 1,
        array + size - 1
    );
    size_t pivot = pivot_addr - array;
    // ...
}
```

3 Additional guidelines

3.1 Practical organization

In order to learn β-assembly effectively, **this project will be done individually**. A report of **maximum two pages** can be provided if you want to explain things that are not easy to understand by just looking at the code and comments. Providing a report does not necessarily mean that you will earn a better grade; it should be provided only if it brings something that is not mentioned clearly elsewhere.

Plagiarism is of course not allowed and severely punished. Any detected attempt will result in the grade 0/20 for all who have participated in this practice.

You will include your completed `quicksort.asm` and your (optional) report (PDF only) in a ZIP archive named `sXXXXXX_NAME.zip` where `sXXXXXX` is your student `ID` and `NAME` your family name in uppercase. Insert your `quicksort.asm` in a ZIP archive even if you do not provide a report. Naming your files differently or submitting other files **will result in a penalty**.

Submit your archive to the **Montefiore Submission Platform**[1], after having created an account if necessary. If you encounter any problem with the platform, let me know. However problems that unexpectedly and mysteriously appear five minutes before the deadline will not be considered. **Do not send your work by e-mail; it will not be read**.

[1] `http://submit.run.montefiore.ulg.ac.be/`

3.2 Code guidelines

Choose a coding style and stick to it. You are advised to use the coding style used in `main.uasm` and `util.asm`. The goal here is not to write code which is as compact and efficient as possible, but to learn the concepts of β-assembly. However, your code should not be unreasonably long and inefficient: minimize the number of registers you use in your recursive procedures, as it impacts the growth rate of your stack.

3.3 Documentation

One of the challenges when writing assembly code is to write a program which is relatively easy to understand. Thus, the second most important element taken into account for your grade (after correctness) will be your code's readability. Use comments extensively (your comments can be larger than your code), but don't be verbose : explain the non obvious, not the immediately apparent.

In addition to the comments written alongside your code, all your procedures should be documented using pre- and post-conditions:

- Arguments have to be properly defined
- Any return value must be documented
- Any side effect (e.g. modification of the dynamic memory) must be documented

Your are free (and advised) to use macros to reduce the redundancy in your code. Those macros should be documented like your procedures (arguments, returned value and side effects).

Good luck and have fun !