

ECS 50: Homework #4 - Assembly functions

Prof. Joël Porquet-Lupine

UC Davis, Fall Quarter 2023


程序代写代做 CS编程辅导

Change log

NOTE

The specifications for this project are subject to change at anytime for additional clarification. Make sure to always refer to the **latest** version.

- v1: First publication



Introduction

November 19, 2023

Programming assignment

This assignment is for the CSIF, and the RISC-V simulator `rvrv/qvrv`

Assignment

The assignment are:

- Writing a complex high-quality assembly program.
- Using functions with arguments, and return values.
- Programming a recursive function.
- Implementing two particular algorithms for sorting and searching.

WeChat: cstutorcs

Assessment

Your grade for this assignment will be broken down in several scores:

Assignment Project Exam Help

Autograding: ~75% of grade

Running an auto-grading script that tests your program and checks the output against various inputs

- Manual review: ~25% of grade

The manual review is itself broken down into different rubrics:

- Quality of implementation: ~10%
- Coding style: ~5%

Program

Sorting and searching

Introduction

There are two most important algorithms used in almost all significant programs are *sorting*, that is organizing a collection of data items in a defined order, and *searching*, that is finding a certain data item in a collection.

https://tutorcs.com

Sorting

There exist plenty of sorting algorithm (including some with very intriguing names, such as “cocktail shaker sort”) but one of the simplest, and yet quite efficient for small collections, is the *insertion sort*.

Below is the pseudo-code corresponding to the insertion sort algorithm on an array `A`.

```
i := 1
while i < length(A)
  j := i
  while j > 0 and A[j-1] > A[j]
    swap A[j] and A[j-1]
    j := j - 1
  end while
  i := i + 1
end while
```

Searching

The most naive searching algorithm consists in looking at *all* the data items in the collection to find the requested one (complexity of $O(N)$ for each search). However, if the collection is already in sorted order, then a *binary search* can leverage this characteristic by recursively dividing the search interval in half until it finds the requested item (complexity of $O(\log(N))$).

Below is the pseudo-code corresponding to the binary search algorithm on an array `A` with a key `key`.

```
position := BinarySearch(A, 0, length(A) - 1, key)

function BinarySearch(A, low, high, key):
  if high < low
    return not_found

  mid := (low + high) / 2

  if A[mid] > key
    return BinarySearch(A, low, mid - 1, key)
  else if A[mid] < key
    return BinarySearch(A, mid + 1, high, key)
  else
    return mid
```

Assembly program

Write program `sort_search.s`, which reads in a variable-size collection of positive 32-bit integers from the user, then sorts them in ascending order, and finally provides the user with the ability to search certain integers within this sorted collection.

The following trace shows an example of the interaction between the user and this program. The special value `-1` is used to indicate the end of the variable-size of input integers, and the end of the search requests.

```
$ rvrv -q sort_search.s
75
84
37
15
47
41
28
76
82
-1
41
Found at index 3
0
Not found
82
Found at index 7
42
Not found
15
Found at index 0
-1
$
```

A compiled C reference runnable on Linux is available on CSIF at `/home/cs50jp/public/hw4/sort_search`. It should show what the expected output is for different inputs, so that you can compare with your assembly implementation.

Your program must be composed of 5 functions, as defined in the following subsections.

Function `load`

Function `load` has the equivalent C prototype: `int load(int arr[], int max_len);`.

It reads integers from the user, one at a time, and stores them in the argument array `arr`. The function stops reading and returns when the user inputs a negative integer (which should not be part of the array) or when the maximum amount of integers, specified by argument `max_len`, has been read. The function returns the number of integers that have been actually read in the array.

Function `sort`

Function `sort` has the equivalent C prototype: `void sort(int arr[], int len);`.

It sorts the argument array `arr` of length `len` in ascending order, using the insertion sort algorithm.

Function `bsearch`

Function `bsearch` has the equivalent C prototype: `int bsearch(int arr[], int low, int high, int key);`.

It recursively finds the integer specified in argument `key` into the argument array `arr` using the binary search algorithm. It returns a negative integer if the search was unsuccessful, or the index of the key in the array.

Function `find`

Function `find` has the equivalent C prototype: `void find(int arr[], int len);`.

It reads integers from the user, one at a time, and launches a search into the argument array `arr` of length `len` using function `bsearch()` and displays the result of the search. The function stops reading integers from the user once the user inputs a negative number.

Function `main`

The main function calls the other functions in the following order:

- `load` to read the integer array from the user
- `sort` to sort the array
- `find` to find different keys in the array

Additional information and constraints

- Although the array itself should be declared in the data section as a global variable, only the main function is allowed to access it directly. All the other functions should receive the array’s address via their first function argument.
- The maximum number of integers that the array can contain is `10`.
- The RISC-V calling convention, as seen in class and discussion, must be respected.
 - Non-leaf functions must use the stack, and should have one prologue, one body, and one epilogue.
 - Leaf functions are allowed to optimize the use of stack and allocate registers instead.
 - When writing a function, you should not assume that you know how registers are used in other functions; this means that:
 - If you use a temporary register, you should not assume that it should retain its value across function calls.
 - If you use a saved register, you need to save its value in the stack before using it.
 - When writing a function that necessitates a local variable (if you were to imagine the equivalent C code), you have a couple options:
 - If the local variable doesn’t need to be persistent across function calls, you can probably optimize it away with a temporary register.
 - If the local variable needs to be persistent across function calls, you can either allocate a saved register for it (if there aren’t too many local variables in the function) or you can allocate the space for an actual local variable in the stack.
- For this program, you should only use the RV32I instructions and relevant pseudo-instructions. You should not have to use the instructions from the M or F extensions.
- Your code should be properly commented, and the different blocks of code should be well-organized and spaced, in order to improve its readability.

Testing

When debugging your code, the easiest is to simply interact with the program manually, by inputting the numbers directly.

Once your code starts working and you want to be able to quickly test it, it can be useful to automate the testing. For example, you can build a input file that can be fed directly to the program upon execution:

```
$ cat input_test.txt
75
84
37
15
47
41
28
76
82
-1
41
0
82
42
15
-1
$ cat input_test.txt | rvrv -q sort_search.s
Found at index 3
Not found
Found at index 7
Not found
Found at index 0
$
```

Submission

Gradescope will be opened for submission on Thursday, November 16 at 0:00. At that time, you will be able to submit your programs and have them be autograded.

Academic integrity

Novelty

You are expected to write this program **from scratch**.

Therefore, you cannot use any existing source code available on the Internet, or even reuse your own code if you took this class before.

Authorship

You are also expected to write this program **yourself**.

Asking anyone someone else to write your code (e.g., a “tutor” on a website such as Chegg.com, or some AI assistant) is not acceptable and will result in severe sanctions.

Sources

You must specify in your report any sources that you have viewed to help you complete this assignment. All of the submissions will be compared with MOSS to determine if students have excessively collaborated, or have used the work of past students.

Violation

Any failure to respect the class rules, both as explained above and in the syllabus, or the [UC Davis Code of Conduct](#) will automatically result in the matter being transferred to Student Judicial Affairs.