

Project # 1

Due: March 4, 2024, by 11:59pm

CS 2160: Assembly Programming and Computer Organization
Spring 2024
Instructor: Keith Paarporn

- This project is worth 100 points
- This project will be completed individually. You are expected to *write your solutions and submit it individually*.
- Please read each task carefully and that you address what is being asked.
- Please read the submission instructions at the end of this document carefully.

Part 1: Getting started (10 pts)

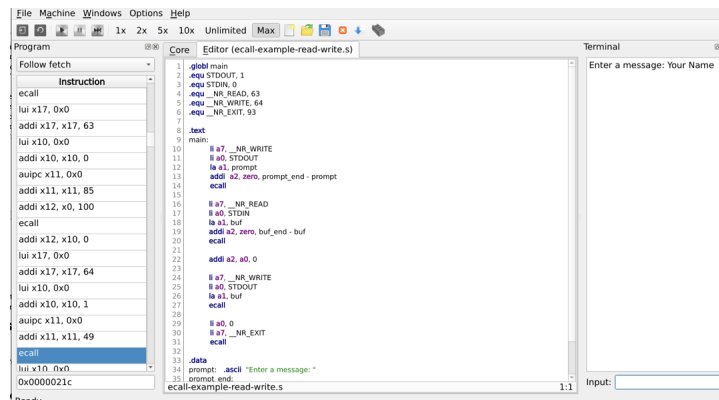
Familiarize yourself with the QtRVSim tool. You can access the browser-based simulator here

<https://comparch.edu.cvut.cz/qtrvsim/app/>

and the documentation can be found here

<https://github.com/cvut/qtrvsim>.

Open the QtRVSim Simulator, “Start Empty” with “No Pipeline no cache” option selected. Navigate to “Open Source” and open the provided project1.S file. You should see the source assembly code in a text editor. Under the “Windows” tab, open the “Terminal” window. Now “Compile Source” and “Run”. You may like to change the simulator speed from 1x to Max for now. You should see “Enter a message:” appear in the terminal window. In the “Input:” box at the bottom of the Terminal window, type in your name. Now “Run” the program to continue its execution. Your typed name should disappear from the Input box—it has been consumed as input. Now “Run” the program to continue its execution again. It should output your name in the Terminal Window. Take a screenshot of your window. It should look approximately like the following:



Clicking “Run” again causes the program to terminate with an `UnsupportedInstruction` exception when `main()` returns. Submit the screenshot as part of your `readme.pdf`, and explain any difficulties you encountered in your `readme.pdf` file with this Part.

Part 2: Refactoring Functions (40 pts)

The given source file contains a single `main()` function in the `.text` section. Between this function’s prolog and epilog are three distinct sequences of code each ending with an `ecall`.

Your task in this part is to refactor the three basic blocks ending in `ecall` to turn them into function calls. It is recommended that you compile often and log your progress. For this task, write these two functions:

```
// reads @a bytes from stdin into the string @a buf.
// Returns the number of bytes read
int read_string(char *buf, int bytes);
```

```
// writes @a bytes of the string @a buf to stdout
void write_string(char *buf, int bytes);
```

Where the body of `write_string` will look similar to the first and third sequences of code, and the body of `read_string` will look like the second sequence of code. The behavior of this program should be identical to the initial behavior

Part 3: libc (40 pts)

Using the results of the previous part, your task in this part is to implement the following four libc functions, and then write a new program `project1-libc.S` that implements:

```
char *str = "Enter a message: " ;
void main(void) {
    char buf[100];
    puts(str);
    gets(buf);
    puts(buf);
}
```

Note that this program is very similar to the first part of the project, but with a few important differences in how the `gets/puts` functions work compared to the `read_string/write_string` functions. The main differences are due to using single character I/O at a time, rather than the bulk I/O interface provided by the `ecall` interface.

For this task, write these four functions. Only the `getchar/putchar` functions should use `ecall`, while the `gets/puts` functions should make iterated calls to `getchar/putchar`.

```
// reads the next character from stdin and returns it as an unsigned char cast to
// an int, or -1 on end of file or error.
int getchar(void);
```

```
// writes the character c, cast to an unsigned char, to stdout
// returns the same input unsigned char
int putchar(int c);
```

```
// reads a line from stdin into the buffer pointed to by s until a terminating
// newline.
// Returns the number of bytes read into the buffer, or -1 on error or end-of-file
// reached
// before reading a newline.
char *gets(char *s);
```

```
// writes the string s appended with a trailing newline to stdout.
// Returns a nonnegative number on success, or -1 on error
int puts(const char *s);
```

To help you out, a C code implementation of `project1-libc` is provided. A good starting point for your assembly file `project1-libc.S` is your assembly file from the previous part.

Part 4: Reporting and Logging (10 pts)


Create the following documents for your submission.

1. A `readme.pdf` document that includes
 - (a) Your name, class, an explanation of how to run your program, a brief description of the pieces of your assignment, any notes, and identification of resources used.
 - (b) Test cases that you used to test your program. Include: A description of what is being tested; the input; the expected output; the actual output; and any known problems of your program, which will help you earn partial credit.
 - (c) Any resources you used for this assignment.
 - (d) Challenges you faced and whether you overcame them.
 - (e) Approximately how much time you think you spent on the project.
 - (f) Documentation of functionality, bugs, etc. of your final submitted version.

- (g) Embed any screenshots you took in the readme.pdf with a brief description.
 - (h) Any other miscellaneous notes.
2. Documentation of the versioning of your code. It is recommended that you use git source version control with a **private repository**. You may use any method available to you for hosting a private git repository, such as github, gitlab, or self-hosted.
For example if you use github, create a directory called {USERNAME}-project1/. Initialize your git repository with the provided “project1.S” file, using a descriptive commit message (e.g. “import provided files”). As you code, you should save your work frequently by committing significant iterations in this repository. By the time you finalize your source code, I would like you to take a screenshot of the commit logs in your repository, and a screenshot of a side-by-side diff change within one of your commits - see figures below. Please have a separate file that contains these logs, e.g. logs.pdf or logs.doc.
 3. Your final source code, formatted as {USERNAME}-project1.S. It should be able to be loaded into QtRVSim by clicking “Open Source”.

Submission Instructions

Please submit into Canvas a single compressed file (zip or tar) named {USERNAME}-project1 that contains one directory that contains your readme.pdf, your source code, and your logs. Your USERNAME should just be your UCCS username (email address without the @uccs.edu part).

 **kpaarporn** Merge pull request [#1](#) from kpaarporn/kpaarporn-patch-1 xxx

3847f7d · yesterday 🕒 5 Commits

project1.s

Update project1.s 3

yesterday

Commits

main

Commits on Feb 11, 2024

Merge pull request [#1](#) from kpaarporn/kpaarporn-patch-1 xxx

 kpaarporn committed yesterday

Update project1.s 3

 kpaarporn committed yesterday


Update project1.s 2

 kpaarporn committed yesterday

Update project1.s xxx

 kpaarporn committed yesterday

Add files via upload xxx

 kpaarporn committed yesterday

Showing 1 changed file with 1 addition and 1 deletion.

project1.s		
↑...	@@ -11,7 +11,7 @@ main:	
11	addi sp, sp, -104	11
12	sw ra, 100(sp)	12
13		13
14	- # main() body	14 +
15		15
16	# Write the prompt to the terminal (stdout)	16
17	li a7, __NR_WRITE	17