COS284 Practical Assignment 2: Floating-Point String Conversion and Processing

University of Pretoria
Department of Computer Science

Online Publication: 23 August 2024

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Plagiarism Policy

- All work submitted must be your own. You may not copy from classmates, textbooks, or other resources unless explicitly allowed.
- You may discuss the problem with classmates, but you may not write or debug code with other classmates, other than those in your group.
- If you use any advanced material, you must cite these sources.

Assignment Overview

In this assignment, you will write assembly functions in YASM (Yet Another Assembler) that perform string-to-floating-point conversion, array processing, and integrate these functionalities into a single cohesive program. The tasks are designed to simulate real-world applications of low-level programming, such as memory management and data manipulation at the assembly level.

Assignment Objectives

1. Convert a string representation of a number into a floating-point value.
2. Parse and extract multiple floating-point numbers from a formatted string.
3. Process an array of floating-point numbers to compute a result based on specific mathematical operations.

4. Integrate the functions to create a complete program that takes user input, processes it, and outputs the correct result.

Mission Details

The experience will simulate a low-level data processing operation:

```
Enter values separated by whitespace and enclosed in pipes (|):
| 32.133 45.66 -21.255 |
Converted numbers:
32.132999
45.660000
-21.254999
The sum of the processed array is: 475.434491
```

Note that the above results are produced by using the provided test.c.
The following sections will break down the tasks and provide hints where needed.

Task 1: String to Floating-Point Conversion

Your first task is to write a function that converts a null-terminated string into a floating-point number. If the conversion is unsuccessful, your function should return `0.0`.

Note that this function does not have to interact with the console whatsoever, the method signature is as follows:

```
float convertStringToFloat(const char *str)
```

You will write the necessary code for this conversion in the file **convert_string_to_float.asm**. A partial skeleton has been provided.

Task 2: Extracting and Converting Floats from a String

The second task requires you to extract multiple floating-point numbers from a string formatted as follows:

```
| 32.133 45.66 -21.255 |
```

You will write the necessary code in the file **extract_and_convert_floats.asm**. This function will:

- Parse the input string.
- Convert the extracted substrings to floating-point numbers.
- Store the results in a dynamically allocated array.

A partial skeleton has been provided.

Note that this function has the following signature:

```
float* extractAndConvertFloats(int* num_floats);
```

And it does output the following message to the console before taking the user's input:

```
Enter values separated by whitespace and enclosed in pipes (|):
```

Also, do not assume that the size passed in will be correct. Rely on your own counting mechanism and also set it to the appropriate value before returning.

Task 3: Processing the Array of Floats

In this task, you will write a function that processes an array of floating-point numbers to produce a single result. The processing involves:

- Converting the float array to a double-precision array.
- Multiplying each element by the next in the array.
- Summing the resulting values.

You will write the necessary code for this in the file **process_array.asm**.
A partial skeleton has been provided.
Take note that the function signature is as follows:

```
double processArray(float *arr, int size)
```

Note that this function does not take from, nor have output to the console.

Task 4: Integrating and Testing the Functions

Finally, you will integrate the previous tasks into a single program that:

- Prompts the user to enter a string of numbers in the format:

  ```
  | 32.133 45.66 -21.255 |
  ```

- Uses the functions from Task 2 to parse and convert the string into an array of floats.
- Processes the array using the function from Task 3.
- Outputs the final computed result.

You will not need to write the integration code yourself. Instead, you will re-upload your tasks in one compressed file, which will then be used to test your implementation as a whole.

Assumptions

You may assume the following:

- The input string will always be correctly formatted with numbers separated by spaces and enclosed within pipe characters.
- The input will not exceed a reasonable length for processing in memory.

Submission

You will submit your assignment via **FitchFork** at ff.cs.up.ac.za. You will submit all of the files as an archive (.zip), and they will be graded individually. You will be graded on the following files:

```
convert_string_to_float.asm
extract_and_convert_floats.asm
process_array.asm
```

You need to submit all of these to be graded on all of the tasks. Partial marks are possible if some tasks are incomplete.

Marking

The total is 12 marks, with a breakdown as follows:

- Task 1: **2 marks**
- Task 2: **3 marks**
- Task 3: **3 marks**
- Task 4: **4 marks**

Advice

You will be splitting your logic into separate functions. Do not assume that a function will preserve the values in your registers when calling them. There are only 16 general-purpose registers, and although there is a calling convention to be respected, always maintain the stack as taught in the lectures.
You may use functions like `fgets`, `printf`, and `strtof` (to convert characters to floats).