CS 382 Fall 2024 — Computer Architecture and Organization — Homework 2

Read Before Start

For each of the tasks in this homework, you are provided two files. One is `<taskname>_data.s`, where `.data` and `.bss` segments are stored. You are free to change the **value** of the variables declared there, but you must not change the label names, and you must not add any new data/instruction there.

The other one is `<taskname>.s` where `.text` segment is provided, along with some starter code. Only write assembly instructions at the speicified place, and do not modify any existing code there. If you need to add new data in your program, feel free to declare another `.data` segment at the bottom of this file (**not** the `<taskname>_data.s` file!!).

When you are asked to print something out, **do not** use `printf()` or any functions from `stdio.h`, as the tester will not be able to capture your output, resulting in failing all test cases.

To test your code on your own, we take task 1 for an example, where `copystr.s` and `copystr_data.s` are provided:

```
$ aarch64-linux-gnu-as copystr.s -o copystr.o
$ aarch64-linux-gnu-as copystr_data.s -o copystr_data.o
$ aarch64-linux-gnu-ld copystr.o copystr_data.o -o copystr
$ qemu-aarch64 copystr
```

We also provided a tester file that can run multiple tests on your assembly program. This is also the tester we are going to use when grading your homework. Note that any violation to the conditions mentioned will likely crash the tester program, so please do follow the instructions.

> We will **not** be responsible for any points lost due to the violation of requirements above.

1 Task 1 (20 pts): Copy a String (Again)

In this task, you will write an assembly code that completes the same task as in previous homework, *i.e.,* copy string `src_str` to another string `dst_str`. You can assume `dst_str` is always large enough to store all characters copied there. After copying the string, please use system call to print the string `dst_str` out to terminal.

Requirements

- You must use loops or recursion. If you want to use recursion, you must follow calling conventions and manage stack frames;
- You must not declare, or hardcode variables that represent string length;
- You must not use any external libraries and functions;
- You must use system call to print, not `printf()`;
- Write your name and pledge at the top of the code.

2 Task 2 (50 pts): Binary Search

In this task, you'll implement a binary search algorithm in ARM assembly. An example of `.data` segment is provided to you in `bins_data.s`, which includes a double word array (sorted), the length of the array, the target value we want to find, and output messages. You can assume the numbers are signed, and the array is **already sorted**. Again, if you need to declare additional data, you must add another `.data` segment inside file `bins.s`, not `bins_data.s`.

After the search, you need to print the messages correctly with the target value using system calls. For example, for the array in `bins_data.s`, if `target` is -25, your output should look like:

```
Target is in the array.
```

If `target` is 20, your output should look like:

```
Target is not in the array.
```

You need to make sure the code will exit successfully without any errors after printing out the messages.

Requirements

- Your algorithm must be binary search, of course;
- You must not hard code array length in your code, so you should always use `length` in the `.data` segment;
- You must use loops or recursion. If you want to use recursion, you must follow calling conventions and manage stack frames;
- You must not use any external libraries and functions;
- You must use system call to print, not `printf()`;
- Write your name and pledge at the top of the code.

3 Task 3 (30 pts): Converting String to Integer

In this task, you'll write an assembly code to convert a string to an integer. For example, say a string is declared in the `.data` segment:

```
.data
numstr:  .string  "382"
number:  .quad    0
```

Then your program will convert the string into an integer `382`, and store it to `number`. You don't need to consider negative numbers.

Just a refresher: if the string is `"9082"`, the number can be calculated by $9 \times 10^3 + 0 \times 10^2 + 8 \times 10^1 + 2 \times 10^0$.

Be Careful...

- The characters in a string are stored as their ASCII values, not the real digit;
- When loading a character, or a byte, into a register, the command is `LDRB` or `LDRSB`, and the destination register is `Wt` not `Xt`.

Requirements

- You must use loops or recursion. If you want to use recursion, you must follow calling conventions and manage stack frames;
- You must not assume the length of the string `numstr`, so you must not declare and hardcode any variable representing string length in `.data` and `.text`;
- You must store the converted integer into variable `number`;
- You must not use any external libraries and functions;
- You must use system call to print, not `printf()`;
- Write your name and pledge at the top of the code.

4 Starter Code & Tester

To help you with testing, we provided a tester file `tester`. Put this tester file in the same directory as your assembly code, and go ahead and run the tester:

```
$ qemu-aarch64 -L /usr/aarch64-linux-gnu/ tester -t {copystr|bins|atoi|all}
```

5 Grading

The homework will be graded based on a total of 100 points.

- Task 1 (20 pts): 5 test cases in total, **4** points each;
- Task 2 (50 pts): 20 test cases in total, **2.5** points each;
- Task 3 (30 pts): 10 test cases in total, **3** points each.

After accumulating points from the testing above, we will inspect your code and apply deductibles listed below. The lowest score is 0, so no negative scores:

- Task 1 (20 pts):
  - **-20:** the code does not assemble, or executes with run-time error;
  - **-20:** the code is generated by compiler;
  - **-20:** no loop/recursion;
  - **-20:** used any external libraries and/or functions (*e.g.,* `printf()`);
  - **-15:** not managing stack frames and/or not following calling conventions if using recursion;
  - **-15:** declared/hardcoded string length;
  - **-5:** no pledge and/or name in assembly file;
- Task 2 (50 pts):
  - **-50:** the code does not assemble, or executes with run-time error;
  - **-50:** the code is generated by compiler;
  - **-50:** no loop/recursion;
  - **-45:** the algorithm is not binary search;
  - **-40:** not managing stack frames and/or not following calling conventions if using recursion;
  - **-30:** used any external libraries and/or functions (*e.g.,* `printf()`);
  - **-5:** no pledge and/or name in assembly file;
- Task 3 (30 pts):

- **-30:** the code does not assemble, or executes with run-time error;
- **-30:** the code is generated by compiler;
- **-30:** no loop/recursion;
- **-30:** used any external libraries and/or functions (*e.g.,* `printf()`);
- **-30:** the converted number is not stored in memory;
- **-20:** not managing stack frames and/or not following calling conventions if using recursion;
- **-20:** declared/hardcoded any data that represents the length of the string;
- **-5:** no pledge and/or name in assembly file.

2% of extra credit will be given if the homework is submitted two days ahead of deadline. If it's re-uploaded after the earlybird date, the deal is off.

Deliverable

Only submit three files: `copystr.s`, `bins.s`, and `atoi.s`. No need to zip.