COMP15111 Lab 3 – Addressing

1.1 Aims

To practice using addresses with RISC-V assembly code.

1.2 Learning Outcomes

On successful completion of this exercise, you will:

- be familiar with some standard ways of using addresses.
- be able to operate with arrays of characters.
- Be able to correctly use register naming according to the RSIC-V ABI

1.3 Summary

Each part involves translating a different method into RISC-V instructions. By the time you have finished, you will have a program that can manipulate strings in various ways.

1.4 Deadline

Each lab exercise has the usual deadline which is the Friday of the week after the current one. All of our deadlines are listed on the unit's landing page on Blackboard. ***We encourage you to engage with this assignment long before the deadline***: a) this will allow you to use the lab sessions productively, b) you will be able to manage your time better (e.g. not having 2-3 deadlines at the same time), and c) you will be able to accommodate unexpected delays. You can submit earlier if you are ready.
Remember that you must tag your commits in the usual way to show that you completed your work by the deadline. The tag for this assignment is **Lab-3-Marking**
On Blackboard, there are specific instructions on how to use git and Bennett.

1.5 Description

As usual, we will use Bennett to run our programs.

There is a file, "lab3.s", that contains some starting RISC-V code, one or more incomplete methods as the starting point for each part described below, and various pieces of test code. You should edit this same file for each part. You should not have to discard any code as you progress from one part to the next.

Instead, you simply need to edit the first line of the file to select the correct piece of test code.

In all 3 parts, the programs you run will attempt to perform output. To view the output, you will need to open a "Terminal" window (as for lab 2).

Please note that every time you re-run your program, you need to close and reopen the terminal in order to reset it.

Part 1 - Calculating the length of a string

For this part, you need to edit a short method, "stringLength". Before you do make any changes, assemble, load, reset and run the program "lab3.s". The following output should appear in the Terminal window:

```
The length of string >> seven >> is 0
The length of string >> six >> is 0
The length of string >> five >> is 0
The length of string >> four >> is 0
The length of string >> three >> is 0
The length of string >> two >> is 0
The length of string >> one >> is 0
The length of string >> COMP15111 >> is 0
The length of string >> Fundamentals of Computer Architecture >> is 0
```

Task (2 points): The method "stringLength" should calculate the length of the string whose starting memory address is stored in register **a0**. In this context, register **a1** should store the length of the string. By default, **a1** contains the integer value zero. Your task is to write a piece of assembly code that calculates the length of the string addressed by **a0**. Essentially you need to write a loop with the following sequence:

- Get the first character from the string addressed by a0.
- Increment a1 by 1 if the character is not equal to 0.
- If the character is not equal to 0, branch back to the start of the loop to get the next character.
- If the character is zero, break the loop.

You will need to use ***postincrement*** addressing to get each character from the string in turn.

Edit the program as described above. assemble, load, reset and run your edited code, and you should get the following output in the Terminal window:

```
The length of string >> seven >> is 5
The length of string >> six >> is 3
The length of string >> five >> is 4
The length of string >> four >> is 4
The length of string >> three >> is 5
The length of string >> two >> is 3
The length of string >> one >> is 3
The length of string >> COMP15111 >> is 10
The length of string >> Fundamentals of Computer Architecture >> is 37
```

Part 2 - Printing string in reverse

Task (3 points): For this part, you need to modify the method, "printstringReverse" to print the given in reverse order. Before you do make any changes, replace the first line "j part1" with "j part2". Now assemble, load, reset and run the program "lab3.s". The following output should appear in the Terminal window:

```
seven
six
five
four
three
two
one
COMP15111
Fundamentals of Computer Architecture
```

The body of "stringReverse" should read characters from the string (addressed by **a0**) until it finds the "0" byte marking the end of the string. In the given code, the register **a0** contains the starting address of the string. To reach the end of the string, you can use the same logic as in the previous task.

Once you have the address of the last character, follow this sequence to print the string in reverse order **(assuming you have stored the address of the last character in a1)**:

- Get the character from the string addressed by a1.
- Print the character using ecall (remember to use the correct operation number! A description of this can be found in the handout of Video 11)
- Decrement a1 and branch back to the start of the loop to read the next character if a1 is not pointing to the first character.
- If a1 is pointing to the first character, exit the loop.

Also, think about which addressing mode(s) you can employ for this specific task.

Edit the program as described above. assemble, load, reset and run your edited code, and you should get the following output in the Terminal window:

```
neves
xis
evif
ruof
eerht
owt
eno
 11151PMOC
erutcetihcrA retupmoC fo slatnemadnuF
```

Part 3 - String copying and concatenation

Task (4 points): For this part, you need to write a method, “stringCopy”, which copies two strings to another empty string. Before you do make any changes, replace the first line “j part2” with “j part3”. Now assemble, load, reset and run the program “lab3.s”. The following output should appear in the Terminal window:

```
COMP15111
```

The body of “stringCopy” should consist of two loops. The first loop copies characters from one string (addressed by a0) to another string (addressed by a2) until it finds the “0” byte marking the end of the string.

- get the next character addressed by a0
- copy this character to the byte addressed by a2
- Increment a0 and a2
- if the character is not equal to 0, branch back to the start of the loop to get the next character.
- If the character is zero, break the loop.

The above loop will also copy the null character '\0', so you need to consider how to discard it before concatenating the second string.

The second loop is similar to the first one, except it starts concatenating characters from second string (addressed by a1) to the end of the existing string (addressed by a2), to make one long string containing both the original strings. (Thus, the first new character must overwrite the “0” byte originally marking the end of the string addressed by a2).

For example, if the string pointed at by a0 was “hello”, so it looked like this in memory:

’h’	’e’	’l’	’l’	’o’	0

and the string pointed at by a1 was “fred”, so it looked like this in memory:

’f’	’r’	’e’	’d’	0

then the string pointed at by a2 should end up being “hellofred”, and look like this in memory:

’h’	’e’	’l’	’l’	’o’	’f’	’r’	’e’	’d’	0

Edit the program “lab3.s” to insert RISC-V instructions equivalent to the loops described above. Edit the program as described above. assemble, load, reset and run your edited code, and you should get the following output in the Terminal window:

COMP15111 Fundamentals of Computer Architecture

Part 4 – Buffer overflow

Task (1 point): In parts of this exercise, you used a buffer (imaginatively labelled 'buffer') to store strings. You did not know in advance how big the strings would be. It is unlikely that you went to the trouble of determining if they were bigger than the buffer space (perhaps after concatenation).

What would happen if the buffer space

was not big enough? What effect might

that have on the program?

Buffer overflows are a common source of faults in software; be warned!

1.6 Completion, Feedback and Marking Process

The total mark for Lab 3 is 10 points. As soon as you have completed the exercise, you need to git commit and push to your repository. Make sure that any new files you might create are added to the repository (git add).

We will mark all submissions within 2 weeks of the deadline. We will provide detailed feedback, which you will receive by email when we mark your submission. We will also discuss this assignment in a live session after the extended submission deadline.

File(s) to be committed to Gitlab

1) lab3.s (as part of the solution to part 1, part 2 and part 3)
2) ans.txt (containing the answers to part 4)