3 COMP15111 Lab 3 – Addressing

Duration: 1 lab session

3.1 Aims

To practise using addresses with ARM assembly code.

3.2 Learning Outcomes

On successful completion of this exercise a student will
– be familiar with some standard ways of using addresses.
– be able to operate with arrays of characters.

3.3 Summary

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

Each lab exercise has the usual deadline at the end of your scheduled lab session. If you attend the lab you will, if you need it, get an automatic 7-day extension (up to the usual starting time of the lab). This is so that you can get help in the lab and then have some time to make use of that help to complete the exercise. This will be the case for all your labs for this course-unit. Remember that you must use “submit” in the usual way to show that you completed your work by the (extended) deadline. If you decide to complete the exercise after our labs shut in the evening, it is your responsibility to make sure that you can “submit” remotely.

3.4 Description

As usual, you will need to copy the starting code to your lab directory and run KMD:

```sh
> cd COMP15111/ex3
> cp /opt/info/courses/COMP15111/Lab/ex3/* .
> start_komodo 15111&
```

There is a file, “ex3.s”, that contains some starting ARM 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 “Features” window (as for exercise 2). You probably need to use the “Clear” button each time before you run a program.

3.4.1 Part 1: Outputting the strings

For this part, you need to edit a short method, “printstring”, which outputs a string whose address is in R1, followed by a new line. It currently uses “SVC 3” to output the string and “SVC 0” to output the new line although names have been EQUated to these numbers (at the top of the file) to make the source code more legible. Thus “SVC print_str” outputs the string etc.

It is recommended that you start to develop this habit to improve readability.

Before you do make any changes, compile, load, reset and run the program “ex3.s”. The following output should appear in the Features window:

```
one
two
three
four
five
six
seven
twentytwo
twenty```
**TASK:** You have to replace the use of “SVC print_str” (SVC 3) in “printstring” by a loop, which gets each character in turn from the string and then uses “SVC print_char” (a.k.a. “SVC 0”) to output it, until it finds the “0” byte marking the end of the string:
- get the next character from the string addressed by R1
- if the character != 0
- then use “SVC 0” to output the character
- and branch back to the start of the loop to get the next character

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

Edit the program (e.g. using nedit) as described above. Compile, load, reset and run your edited code, and you should get the same output as before.

### 3.4.2 Part 2: String copying and concatenation

**TASK:** For this part, you need to write two methods, “strcpy” and “strcat”, which copy and concatenate strings respectively.

The body of “strcpy” should consist of a loop that copies characters from one string (addressed by R2) to another string (addressed by R1) until it finds the “0” byte marking the end of the string (which also needs to be copied):
- get the next character addressed by R2
- copy this character to the next byte addressed by R1
- if the character != 0
- then branch back to the start of the loop to get the next character

The body of “strcat” should consist of two loops. The first loop just looks for the “0” byte marking the end of the string addressed by R1:
- get the next character addressed by R1
- if the character != 0
- then branch back to the start of the loop to get the next character

The second loop is similar to “strcpy”, except it starts copying characters to the end of the existing string addressed by R1, to make one long string containing both the original strings. (Thus, the first new character has to overwrite the “0” byte originally marking the end of the string addressed by R1.)

For example, if the string pointed at by R1 was “hello”, so it looked like this in memory:

```
|h|e|l|l|o|0|
```

and the string pointed at by R2 was “fred”, so it looked like this in memory:

```
|f|r|e|d|0|
```

then the string pointed at by R1 should end up being “hellofred”, and look like this in memory:

```
|h|e|l|l|o|f|r|e|d|0|
```

Edit the program “ex3.s” (e.g. using nedit) to insert ARM instructions equivalent to the loops described above. Edit the first line to branch to “part2” instead of “part1”. Compile, load, reset and run your edited code. The following output should appear in the Features window:

```
one
onetwo
onetwothree
```

### 3.4.3 Part 3: Comparing two strings

**TASK:** For this part, you need to write a short method, “sorted”, which compares two strings to see if they are in alphabetical order.

When this method is called, the addresses of the two strings must be in R2 and R3. When it finishes and returns to where it was called from, we want to be able to test the result, so the method will finish by doing a final “CMP” instruction, that can be tested later.

The body of this method also consists mainly of a loop:
- get the next character from each string into a register (R4 and R5)
- if either character is zero, exit the loop
- if the two characters don’t match, exit the loop
- branch back to the start of the loop

The checking for alphabetical order relies on the ASCII code for ‘a’ being less than that for ‘b’ etc. Also note that the 0 terminator is the lowest possible code, so a short string “abc” will be ‘less than’ a longer one “abcde” because 0 is less than ‘d’.
When the loop is exited, but before returning from the method, you need to CMP the two characters so the result is available for testing. In this case you will have created a ‘function’ – a method which returns a value. (This function is slightly unusual in returning the value in the processor flags.)

Edit the program “ex3.s” (e.g. using nedit) to insert ARM instructions equivalent to the loop described above. Edit the first line to branch to “part3” instead of “part2”. Compile, load, reset and run your edited code. The following output should appear in the Features window:

one<two
two>three
three>four
four>five
five<six
six>seven
seven<twentytwo
twentytwo>twenty
twentytwo=twentytwo

3.5 Buffer overflows

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!

3.6 Completion, Feedback and Marking Process

There are 3 marks for part 1, 4 marks for part 2, and 3 marks for part 3. (You cannot get different parts marked on different days.)

As soon as you have completed the exercise, you need to run submit (making sure you are in the correct directory – COMP15111/ex3 for this exercise).

After this, run labprint (again, make sure you are in COMP15111/ex3), collect your marking sheet from a printer and then ask a demonstrator to mark your work (you may need to add your name and seat number to a list of those queueing to be marked).

IMPORTANT – University rules say that all work must be marked within 15 days, so you must get your work marked during your next scheduled lab (unless have a good excuse e.g. that you were ill). If a lab does not get marked by this deadline, it may get removed from Arcade and, hence, will not count towards the final mark. This will be the case for all of your labs for this course-unit.