ARM extra workshop - 2

Anas Shrinah

December 9, 2022

The goal of this worksheet

This workshop aims to help you write an ARM assembly program to capitalize the first letter of each word from a string.

1 The problem description

Write an ARM assembly program that capitalizes the first character of each word from a string. The characters of the string are stored in word-aligned memory addresses (four bytes each).

The end of the string is marked by the ASCII character Line Feed (lf) 0x0A. The first character of each word, apart from the first word, is a character that is preceded by a space; the ASCII code of the space is 0x20. The first character of the first word is the first character in the string.

In the following steps we will gradually write some of the required ARM assembly codes to solve this problem.

Use this template:

```
.section .text
.align 2
.global _start
_start:


@ your code goes here!



.section .data
_data:
  @Hi I know how to code with ARM assembly
.word 0x48, 0x69, 0x20, 0x49, 0x20, 0x6B, 0x6E, 0x6F, 0x77, 0x20, 0x68, 0x6F, 0x77,
 0x20, 0x74, 0x6F, 0x20, 0x63, 0x6F, 0x64, 0x65, 0x20, 0x77, 0x69, 0x74, 0x68, 0x20,
 0x41, 0x52, 0x4D, 0x20, 0x61, 0x73, 0x73, 0x65, 0x6D, 0x62, 0x6C, 0x79, 0x0A
```

2 Processing the first letter of the first word of the string

The first part of the program has to deal with the first character of the string. This character has a special condition because it has to be capitalized though it is not preceded by a space.

We have to load the first character (the ASCII code of the first character) of the string to the register r1. Then we have to test the ASCII code of this character to check if it is a lower case letter. If so, then we have to capitalize it.

To load the first character, we have to load its address to a register; use r0.

@ Write one instruction to load the address of the first character to the register r0.

Then, we have to load the first character to r1. The first character is located in the memory at the address specified by r0 (we loaded the address of the first character to r0 in the previous step).

To write an instruction to load r1 with the value addressed by r0, we have to answer the following questions:

1. Which instruction loads a single word from the memory to a register?
2. What type of addressing mode should we use pre-indexing or post-indexing?
3. Should we use the register write-back option?

@ Write one instruction to load r1 with the value addressed by r0.

After the first character is loaded, we have to test the ASCII code of this character is the Line Feed character (0x0A). If this character is the Line Feed character, then we have to branch to an infinite loop.

@ Write two instructions to test if r1 is equal to 0x0A, if so, branch to the label _end.
_end: b _end

Now we know the first character is not the Line Feed character, we have to test if ASCII code of this character represents a lower case letter. To do this task, answer the following questions:

1. What is the range of the ASCII codes of the lower case alphabet letters? Find this from an ASCII codes table.
2. How can we capitalize a letter by manipulating its ASCII code? Check previous ARM extra workshop.
3. How can we check two conditions in ARM. We need to check the ASCII code of the current character is more than the lower value and less than the upper value of the range of the the ASCII codes of the lower case alphabet letters.

Write ARM code to check if r1 value is greater than the ASCII code of the letter 'a' (lower case a) and r1 value is less than the ASCII code of the letter 'z' (lower case z). If the current character is a lower case letter, capitalize it.

@ Write three ARM instructions:
@ Check if r1 is greater than 0x60 (i.e. the ASCII code of the current character is 0x61 ('a') or more).
@ If r1 is greater than 0x60, move to label _ASCII_greater_than_0x60
@ If r1 is not greater than 0x60, move to label _ASCII_not_greater_than_60

Under the label _ASCII_greater_than_0x60:

@ Write three ARM instructions:
@ Check if r1 is less than 0x7B (i.e. the ASCII code of the current character is 0x7A ('z') or less).
@ If r1 is less than 0x7B, modify r1 to capitalize the lower case letter represented by r1.
@ If r1 is less than 0x7B, store the new value of r1 into the memory in the same address where r1 was loaded from.

Before we move to load and test the ASCII codes of the remaining characters, we have to check if the first character is a space. This is important because if the first character is a space and the second character is a lower case letter then we have to capitalize the second character.

Under the label _ASCII_not_greater_than_0x60:

@ Write three ARM instructions to check if r1 value is equal to the ASCII code of space (0x20). If r1 is a space, move 1 to r2, else move 0 to r2.
@Check if r1 (the current letter) is a space.
@ If r1 is equal to 0x20, move 1 to r2
@ If r1 is not equal to 0x20, move 0 to r2

3 Processing the remaining letters of the string

The second part of this code is concerned with checking the remaining characters of the string (starting from character number two). To do so, we have to code an infinite loop that first loads the second character. If this character is the Line Feed, branches to the label _end, otherwise loads a new character from the string and repeats the process.

@ Write four ARM instructions:
@ Load the second character to r1 using the address of the first character.
@ Check if the ASCII code of the second character is equal to the code of the Line Feed (0x0A).
@ If so, branch to the label _end
@ Branch unconditionally to the first instruction in this block.

Now we have the main loop of our program, we have to insert ARM code to test if the current character is a lower case letter and if the previous character is a space, then we have to capitalize the current character.
First check if the ASCII code of the current character is 0x61 ('a') or more).

@ Write three ARM instructions:
@ Check if r1 is greater than 0x60 (i.e. the ASCII code of the current character is 0x61 ('a') or more).
@ If r1 is greater than 0x60 move to label _ASCII_greater_than_0x60_in_loop
@ If r1 is not greater than 0x60 move to label _Skip_capitalization

Next, check if the ASCII code of the current character is is 0x7A ('z') or less.
Under the label _ASCII_greater_than_0x60_in_loop:

@ Write some ARM instructions to check if the ASCII code of the current character is 0x7A ('z') or less.
@ Check if r1 is less than 0x7B (i.e. the ASCII code of the current character is 0x7A ('z') or less).
@ If r1 is less than 0x7B move to label _ASCII_greater_than_0x60_and_less_than_0x7B
@ If r1 is not less than 0x7B move to label _Skip_capitalization; two instructions might be needed.

After that, if the current character is a lower case letter, check if the previous character is a space.
Under the label _ASCII_greater_than_0x60_and_less_than_0x7B:

@ Write three ARM instructions to check if the previous letter was a space.
@ Check if r2 is equal to 1 (i.e. the previous character was a space).
@ If r2 is equal to 1, move to label _ASCII_greater_than_0x60_and_less_than_0x7B_and_prev
@ If r2 is not equal to 1, move to label _Skip_capitalization.

Under the label _ASCII_greater_than_0x60_and_less_than_0x7B_and_previous_letter_is_space:

@ Write two ARM instructions to capitalize the value in r1 (the current letter) and store it in the memory (the same location of the second character).
@ Mathematically change the value of r1 so its new content represents the upper case letter of the lower case letter that was in r1.
@ Store the new value of r1 into the memory in the same address where r1 was loaded from.
@ If r2 is not equal to 1, move to label _Skip_capitalization.

Before we loop back to the beginning of the procedure, we have to check if the current character is a space. Under the label _Skip_capitalization:

@ Write three ARM instructions to check if r1 (the current letter) is a space and update r2 accordingly.
@ Check if r1 is equal to 0x20 (i.e. the current character is a space).
@ If r1 is equal to 0x20, move 1 to r2.
@ If r1 is not equal to 0x20, move 0 to r2.

4 GDB - Print the contents of multiple memory locations

Use the command x followed by /n address, where n is the number of locations, and the address is the address of the first location in the memory. For instance the command : x /40 0x8101C, prints 40 memory locations starting from the address 0x8101C.