Assignment 3 (6%)
Strings

Deadline: Friday, 17 November 2023 at 23:59 on Submitty

Working individually, complete the assignment below. Submit your solution to Submitty (`https://submit.scss.tcd.ie`). Your mark will be a combination of the auto-graded mark assigned by Submitty (70 marks) and a manual grading mark (30 marks).

You are allowed to submit **six attempts** for the assignment without penalty. Subsequent attempts will attract a 5 mark penalty each, up to a maximum penalty of 30 marks.

Submitty will allow you **eight "late days"** for CSU11021 assignments. For example, you can submit one assignment late by eight days or four assignments late by two days each, without penalty. Once your "late days" are used up, you will receive zero marks for any late submissions. Note that even 1 second after the deadline counts as one full late day. Late days do not apply to exercises that are not for credit.

By submitting your solution, you are confirming that you have familiarised yourself with College's policy on academic integrity (`https://libguides.tcd.ie/academic-integrity`).

Instructions

When we write an ARM Assembly Language program, we can represent immediate operands in a number of different ways. For example, in the `MOV` instruction below, an immediate operand is represented in decimal (base 10) using the syntax `#74`.

```
MOV R1, #74
```

The same value can be represented in hexadecimal as `#0x4A` or in binary as `#0b1001010`. We can even express the same value in ASCII character form as `#'J'`. (Refer to an ASCII table to verify that `0x4A` is the ASCII code for upper-case 'J').

ARM Assembly Language programs are just strings of ASCII characters so the immediate operand `#74` is just an ASCII string containing the characters `'#'`, `'7'` and `'4'`.

Part 1: Decimal to Value

Write an ARM Assembly Language program that, given a NULL-terminated ASCII string representing an immediate operand in decimal form, will calculate the value of the operand and store it in `R0`. The input string starts in memory at the address in `R1`.

For example, given the string "`#74`", your program should store the value 74 in `R0`.

The string is terminated by the NULL character (i.e. ASCII character code 0). Assume (for now) that constant values are unsigned and are never prefixed with + or −.

Part 2: Decimal to Hexadecimal

Extending your program from Part 2, write an ARM Assembly Language program that, given a NULL-terminated ASCII string representing an immediate operand in decimal form, will create a new ASCII string in which the same value is represented in hexadecimal form.

For example, given the string "#74", your program should create the new string "#0x4A".

Your program should store the new string in memory at the address in R2. All alphabetic hexadecimal digits ('A' ... 'F') must be UPPERCASE and the 'x' in the prefix "0x" must be lowercase. Your program should also continue to store the value of the operand in R0. The new string should be NULL-terminated. The value represented in the new string should not contain leading zeros (e.g. the new string should be "#0x4A" instead of "#0x0000004A").

Part 3: Decimal to Hexadecimal in an ARM Instruction

Given a string containing a line from an ARM Assembly Language program, modify and extend your program from Part 2 to create a copy of the original string in which any occurrence of an immediate operand in decimal form is replaced with its equivalent in hexadecimal form.

For example, given the string ...

"ADD R8, R8, #90"

Your program should create the new string ...

"ADD R8, R8, #0x5A"

Note that the string might not end with the constant value operand and your program should be able to handle strings such as ...

"MOV R9, #180   @ y = 180"

You can make the simplifying assumption (for now) that an occurrence of the character '#' always represents the start of a valid immediate operand in decimal form. You can identify the end of a constant value by the occurrence of any character other than '0' ... '9'.

Store the new, modified copy of the string in memory at the address in R2.

Part 4: Binary to Hexadecimal

Modify and extend your program again to also convert immediate operands represented in binary form into their hexadecimal equivalent.

For example, the immediate operand "#0b10101000" should be converted to "#0xA8".

Immediate operands represented in binary form are denoted by the prefix "0b" whereas values represented in decimal form do not have any prefix. Your program should still also be able to convert immediate operands from decimal to hexadecimal form.

Part 5: ASCII Character to Hexadecimal

Modify and extend your program again to also convert immediate operands in the form of a single ASCII character into the corresponding ASCII character code. The character code should be in hexadecimal form.

For example, the immediate operand ...

```
#'Z'
```

should be replaced with the ASCII string "#0x5A".

The ASCII character code for the ' symbol used to denote an ASCII character is 0x27.

As before, your program should still also work for immediate operands in decimal and binary form.

Part 6: Signed Decimal to Value

Finally, extend your program to handle both unsigned and signed decimal immediate operands. Your program should recognise values such as "#+74", "#-74" and "#74". Negative values should be represented as 32-bit 2's complement values in R0 and replaced as before in the instruction with the hexadecimal representation of the raw binary value in R0.