Computer Science 230
Computer Architecture and Assembly Language
Fall 2023

Assignment 1

Due: Sun, October 15, 11:59 pm by Brightspace submission
(Late submissions **not** accepted)

Programming environment

For this assignment you must ensure your work executes correctly on the simulator within Microchip Studio 7 as installed on workstations in ECS 249. If you have installed AVR Studio on your own, then you are welcome to do much of the programming work on your machine (The IDE is available at no charge from `https://bit.ly/3llENk7` but comes only in the Microsoft Windows flavour). **You must allow enough time** to ensure your solutions work on the lab machines. If your submission fails to work on a lab machine, the fault is very rarely that the lab workstations. “It worked on my machine!” will be treated as the equivalent of “The dog ate my homework!”

Individual work

This assignment is to be completed by each individual student (i.e., no group work). Naturally you will want to discuss aspects of the problem with fellow students, and such discussion is encouraged. **However, sharing of code fragments is strictly forbidden without the express written permission of the course instructor (Baharloo).** If you are still unsure regarding what is permitted or have other questions about what constitutes appropriate collaboration, please contact me as soon as possible. (Code-similarity analysis tools will be used to examine submitted work.) The URLs of significant code fragments you have found and used in your solution must be cited in comments just before where such code has been used.

Objectives of this assignment

- Understand short problem descriptions for which an assembly-language solution is required.
- Use AVR assembly language to write solutions to three such small problems.
- Use AVR Studio to implement, simulate and test your solution. (We will not need to use the Arduino mega2560 boards for this assignment.)
- Hand draw a flowchart corresponding to your solution to the second problem.
- **You are not to use AVR functions in this assignment.**

Part (a): Edit distance

One specialization in computer science and mathematics is *information theory*, invented in the 1940s when the possibilities for the digital representation of data and signals were first deeply investigated. An example of a concept in information theory is *edit distance*, which describes how much one data string differs from another data string. If we examine binary numbers instead of data strings, we can see an edit distance more clearly. For example, consider the 8-bit binary equivalents of 198 and 81:

```
11000110
01010001
```

The numbers are clearly different, and one measure of this is found by determining *the number of bit positions in which the two binary numbers differ*. Shown below are these two numbers again but with the different bits noted in bold:

110**00110**
010**10001**

That is, if we denote bit 7 as the left-most bit of each number, then bits 7, 4, 2, 1 and 0 (i.e., five bits) are different between each number. The edit distance of these two binary numbers is therefore five (5).

Your task for part (a) is to complete the code in `edit-distance.asm` within the project `csc230-a1-part-a` provided to you. Please read this file for more details on what is required.

- The values for which you will find the edit distance will be registers `r16` and `r17`
- The computed edit distance must be stored in `r25`.

Some test cases are provided to you in the provided assembly file.

Part (b): Resetting the right-most contiguous sequence of bits

Another common task when working with bit sequences is to identify and work with *contiguous set bits*. For example, consider the bit sequence shown below, with several of the set bits shown in a bold font:

010**111**00

We say that the bolded set bits constitute the *right-most contiguous set bits*. That is, bits 4, 3, and 2 are set. If we *reset the right-most contiguous set bits* of the example just given, the result is:

010**000**00

Your task for part (b) is to complete the code in `reset-rightmost.asm` in the project `csc230-a1-part-b` provided to you. Please read this file for more details on what is required.

- The bit sequence for which the right-most contiguous set bits must be reset is in `r16`.
- The result must be stored in `r25`.

Some test cases are provided to you in the provided assembly file.

You must also draw by hand the flowchart of your solution and submit a scan or smartphone photo of the diagram as part of your assignment submission; please ensure this is a JPEG or PDF named "`reset-rightmost-flowchart.jpg`" or "`reset-rightmost-flowchart.pdf`" depending on the file format you have chosen. (Please: No BMPs or Word files! We beg of you!)

Part (c): Addition of two packed BCDs numbers

In lectures we have seen that a number such as 72_{10} may be represented as an eight-bit two's complement number: `0b01001000`. Another form for representing decimal numbers that was once popular is called **binary-coded decimal (BCD)**. For a given number, each decimal digit (from 0 to 9) was encoded in a four-bit field. The BCD representation of 72_{10} is `0b01110010`, i.e., where the left nibble (`0111`) represents 7 and the right nibble (`0010`) represents 2. A larger decimal number would therefore require more groups of four bits, that is, four bits per decimal digit.

Put differently, one result of BCD is that the hexadecimal version of the number appears identical to the decimal, even though they are in different bases. That is, `0x72` in a BCD encoding has the same meaning as 72_{10}.

Performing arithmetic with BCD numbers is, however, definitely not the same as with the ordinary two's-complement encoding we have examined in class! For example, adding together `0x35` (53_{10}) and `0x49` (73_{10}) as two's complement numbers results in `0x6E` (126_{10}). However, as BCD, the addition `0x35` and `0x49` is `0x84`. This also suggest that the largest number which can be represented in a byte using BCD is `0x99`. (We will only concern ourselves in this assignment with positive numbers.)

Your task for part (c) is to complete the code in `bcd-addition.asm` in the project `csc230-a1-part-c` provided to you. The code is to add two numbers represented in BCD.

- The BCD operands for addition are in `r16` and `r17`.
- The right-most two BCD digits of addition result must be in `r25`.
- The carry (either 0 or 1) resulting from the addition must be in `r24`.

Please read the ASM file for more detail on what is required. In your solution you will wish to make use of the `cbr` and `swap` instructions (amongst others).

What you must submit

- Your completed work in the three source code files (`edit-distance.asm`, `reset-rightmost.asm`, `bcd-addition.asm`). Do not change the names of these files! **Please do not submit any other AVR 7 Studio project files. Please do not submit ZIP files.**
- Your work must use the provided skeleton files. Any other kinds of solutions will not be accepted. (Again: Do not submit ZIP files containing the Microchip projects!)
- The scan / smartphone photo of your hand-drawn flowchart with the name of "`reset-rightmost-flowchart.jpg`" or "`reset-rightmost-flowchart.pdf`" depending on the format you have chosen.

Evaluation

- 3 marks: Edit Distance solution is correct for a variety of byte pairs values (part a).
- 3 marks: Reset Rightmost is correct for different byte values (part b).
- 1 mark: Hand-drawn flowchart for Rest-rightmost solution is correctly prepared (part a).
- 2 marks: BCD addition solution is correct for different pairs of BCD values (part c).
- 1 mark: All submitted code is properly formatted (i.e., indenting and comments are suitably used), and files correctly named.

Total marks: 10