Computer Science 230
Computer Architecture and Assembly Language
Summer 2019

Assignment 1

Due: Thursday June 6th, 11:55 pm by conneX submission
(Late submissions **not** accepted)

Programming environment

For this assignment you must ensure your work executes correctly on the simulator within AVR 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/2VIArVA` but comes only in the Windows 10 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 (Zastre).** 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 problem.
- 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 first problem.

Part (a): Computing *Hamming distance*

Digital-communication hardware and software often includes circuity and code for detecting errors in data transmission. In fact, nearly all dynamic-memory devices include an error-correcting code called a *Hamming code*. While it may be a little premature to implement such a code in our course, we can investigate a related concept called *Hamming distance*.

Consider two different byte values, first:

```
0b10101100
```

and then:

```
0b11101010
```

In order to convert one of the byte values into the other, we could flip bits in specific positions, i.e., those positions where the bits are different between the two bytes. In our example, the bits in three positions are different: positions 6, 2 and 1 (i.e., recall that bit position 7 is the leftmost bit, while bit position 0 is always the rightmost bit). We can therefore state that the *Hamming distance* of these two byte values is equal to three.

Your task for part (a) is to complete the code in the file named `hamming_distance.asm` in the project `csc230_a1_part_a` provided to you. Please read the ASM file for more detail on what is required. Amongst other AVR machine instructions, you will need to use bit shift, logical AND, arithmetic ADD, and branching. ***Do not use functions***. 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 "`hamming_distance_flowchart.jpg`" or "`hamming_distance_flowchart.pdf`" depending on the file format you have chosen.

Part (b): Reversing the order of bits in a word

Recall that in our course we define a word to be a 16-bit sequence (i.e., two consecutive bytes). For some algorithms it is useful to have a reversed version of that 16-bit sequence. (The deeply curious can read a brief description about such use in Fast Fourier Transform algorithm implementations by visiting Wikipedia at this link: `http://bit.ly/2rnvwz6`).

Your task for part (b) is to complete the code in `reverse.asm` in the project `csc230_a1_part_b` provided to you. Please read the ASM file for more detail on what is required. Amongst other AVR machine instructions, you will need to use bit shift, logical AND, arithmetic ADD, and branching. ***Do not use functions***.

Part (c): Binary Coded Decimal

In lectures we have seen that a number such as 72_{10} may be represented in binary as `0b01001000`. Another form of representing decimal numbers that was once popular is called **binary-coded decimal (BCD)** where 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.

Although it would be fun write code that converts a binary number into its BCD equivalent, for this assignment you will instead implement the other direction (BCD into its binary equivalent) as it is a bit easier to code. That is, when given a BCD number such as `0b01110010`, your program will produce the binary-number equivalent of `0b01001000`.

Your task for part (c) is to complete the code in `bcd2binary.asm` in the project `csc230_a1_part_c` provided to you. In that file there are some comments giving several simplifying assumptions. Please read the ASM file for more detail on what is required. Amongst other AVR machine instructions you will need to use bit shift, logical AND, arithmetic ADD, and branching. ***Do not use functions***. You may assume that all byte values used to test your submission will be made up of properly formed BCD numbers.

What you must submit

- Your completed work in the three source code files (`hamming_distance.asm`, `reverse.asm`, `bcd2binary.asm`). Do not change the names of these files! Please do not submit any other AVR 7 Studio project files.
- Your work must use the provided skeleton files. Any other kinds of solutions will not be accepted.
- The scan / smartphone photo of your hand-drawn flowchart with the name of "`hamming_distance_flowchart.jpg`" or "`hamming_distance_flowchart.pdf`" depending on the format you have chosen.

Evaluation

- 3 marks: Hamming-distance solution is correct for a variety of byte pairs values (part a).
- 1 mark: Hand-drawn flowchart for Hamming-distance solution is correctly prepared (part a).
- 3 marks: Reverse solution is correct for different word values (part b).
- 2 marks: BCD2binary solution is correct for different values 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