Computer Science 230
Computer Architecture and Assembly Language
Spring 2018

Assignment 1

Due: Monday February 5th, 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 as installed on workstations in ECS 249. If you have installed AVR Studio on your own computer then you are welcome to do much of the programming work on your machine. If this is the case, however, then 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 *even parity*

Digital-communication hardware and software often include circuity and code for detecting errors in data transmission. The possible kinds of errors are large in number, and their detection and correction is sometimes the topic of upper-level Computer Science courses. In a nutshell, however, the concern is that bits sent by a data transmitter are sometimes not the same bits that arrive at the data receiver.

That is, if the data transmitter sends this byte:

```
0b10101011
```

but this is received instead:

```
0b00101011
```

then the data receiver has the wrong data, and would need some way to determine that this was the case. In response to the error, the same receiver might discard the byte, or ask the sender to retransmit it, or take some other combination of steps.

A technique for detecting one-bit errors is to associate with each byte a *parity bit*. If an *even parity bit* is transmitted along with the byte, then *the number of all set bits in the byte* plus *the value of the parity bit* must sum to *an even number*. In our example above involving the data transmitter, the chosen parity bit would be 1 (i.e., five set bits in 0xAB, plus one set parity bit, equals six set bits). The data receiver has the corrupted byte as shown (0x2B) plus the parity bit value, and determines that the number of set bits is five (i.e., four bits in the byte, plus the set parity bit, resulting in five set bits, which is an odd number). Given that five is not an even number, the receiver can conclude there was an error in transmission. Note that the receiver can only detect an error with the parity bit; more is needed to correct the error.

Your main task for part (a) is to complete the code in the file named `parity.asm` that has been provided for you. Please read the 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 "`parity_flowchart.jpg`" or "`parity_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` that has been provided for you. Please read this 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): An implementation of *modulus*

You are already familiar with modulus operations from first-year programming courses. Usually these are presented as the "`%`" binary operation and explained to you as the equivalent of finding the remainder from integer division operations.

It can also be explained as the result of successive subtractions. For example, consider the expression `M % N` where `M = 370` and `N = 120`. If we repeatedly subtract until we have a value less than `N`, then that value must be `M % N`.

- `370 – 120 => 250` (subtraction #1)
- `250 – 120 => 130` (subtraction #2)
- `130 – 120 => 10` (subtraction #3)

The value of `10` obtained from the last subtraction is therefore `370 % 120`.

Your task for part (a) is to complete the code in `modulo.asm` that has been provided for you. In that file there are some comments giving several simplifying assumptions. For example, the value `M` will always be a positive two's-complement number; the value `N` will always be a two's complement positive number less than 128. One challenge you might face, however, is how to determine when to stop subtractions. The AVR instruction-set architecture does not have a "less than" or "greater than" instruction. (Hint: If you were to do one more subtraction than needed, the carry bit would be set to indicate a "borrow" would be needed in order to ripple up the subtraction of more-significant bytes.)

Amongst other AVR machine instructions you will need to use bit shift, logical AND, arithmetic ADD, and branching. ***Do not use functions***. You need not worry about error checking of M and N values – you may assume that only those values satisfying the constraints given in the skeleton ASM will be used to test your solution.

What you must submit

- Your completed work in the three source code files (`parity.asm`, `reverse.asm`, `modulo.asm`). Do not change the names of these 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 "`parity_flowchart.jpg`" or "`parity_flowchart.pdf`" depending on the format you have chosen.

Evaluation

- 2 marks: Parity solution is correct for different byte values (part a).
- 1 mark: Hand-drawn flowchart for parity solution is correctly prepared (part a).
- 3 marks: Reverse solution is correct for different word values (part b).
- 3 marks: Modulo solution is correct for different values of M and N (part c).
- 1 mark: All submitted code properly formatted (i.e., indenting and comment are correctly used), and files correctly named.

Total marks: 10