CSC 230 Assignment 3
Fall 2017

This assignment is an AVR Assembly Language programming assignment needs to be tested on the Arduino 2560 boards in the ECS 249 lab. There are two parts to this assignment:

- Part 1: Called a formative submission, contains introductory activities that assist in the development of the final solution. The formative submission is due 1 week earlier than the final due date of the assignment. This submission is designed to ensure you start to work on the assignment early enough to seek assistance before the final due date. When your Part 1 is as complete as possible, submit it on Connex under the Assignments, Assignment 3 Part 1 link. If you feel there were any learning issues with this part of the assignment, follow up with the teaching team, during office hours, as early as possible during the week.
  Submit an assembly language file called `A3P1.asm` on conneX assignments (Assignment 3 Part 1) by Monday, October 2, 2017 well before 5 am.

- Part 2: Called a summative submission, this is the final submission of the assignment. This will be graded in the the week that follows the due date. When your program is complete and tested, including appropriate documentation, submit it on Connex under the Assignments, Assignment 3 Part 2 link.
  Submit a C file called `A3P2.asm` on conneX assignments (Assignment 3 Part 2) by Monday, October 9, 2017 well before 5 am.

Learning Objectives

Upon successful completion of this assignment, you should be able to write AVR assembly language programs that:

- ✓ Assemble and execute correctly on the 2560 boards
- ✓ Use Immediate, Register, Direct and Indexed address modes.
- ✓ Use loop structures
- ✓ Use index addressing to simulate arrays in memory
- ✓ Use the masks & shift instructors to write bytes of data to LEDs attached to a port.

Assignment 3 Part 1: Formative

Part 1: The program described in this part must be written, documented and submitted. Please follow up with any questions you may have with the teaching team during office hours, as early as possible during the following week. (This *formative* submission is intended for form your skills, rather than create a grade. The *summative evaluation*, which results in a grade, will be reserved for Part 2 below.)

Pseudo-code is a programming-like language that assists in program design, but is simpler or not as well defined as a programming language. Pseudo-code will be used throughout CSC 230 and will appear similar to statements in the C or Java programming

languages. It will be used to both define algorithms and serve as documentation throughout assembly language code.

Consider the following pseudo-code 'program':

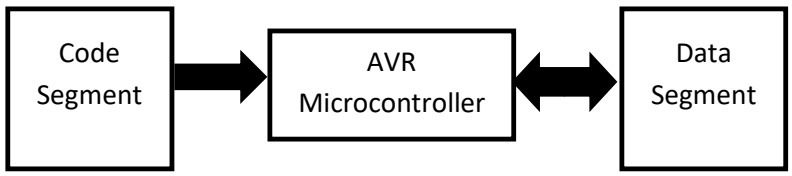
```
number1 = 27;
number2 = 41;
number3 = 15;

sum = 0;
sum += number1;
sum += number2;
sum += number3;

result = sum;
```

Essentially, this 'program' adds 3 constant values together and stores them in memory location called `result`. In Assignment 2 Part 1, this pseudo-code will be:

- Translated into AVR assembly language,
- Tested on the AVR studio simulator

While programs would normally be downloaded onto an AVR microcontroller and tested and used, that will be reserved for a future assignment.

First, recall that the AVR microcontroller uses the *Harvard architecture*:



The program that will be run by the microcontroller must be downloaded into the code segment, which is a *flash* memory, then the program is run by the microcontroller that can use the data segment, which is a *static random access memory* (SRAM), to store both temporary values and result. Observe that the code segment can be initialized, as it is when we download the program, but the data segment cannot. Thus, we must store the initial (hard coded) values used by the program in the code segment, along with the program.

Below is a 'shell' or starting assembly language program outline to be used in Assignment 3 Part 1. (It is also provided on the course site as A3P1.asm.)

```
.include "m2560def.inc"
;
;
*************************************************************
; * Program information goes here.                           *
; *  Examples include:  Course name, assignment number,      *
; *     program name, program description, program input,    *
; *     program output, programmer name, creation date,      *
; *     dates of program updates. *
; *************************************************************

; ************************
; * Code segment follows: *
; ************************
.cseg

;************************
; Your code starts here:
;




;
; Your code finishes here.
;************************

done:     jmp done

; ************************
; * Data segment follows: *
; ************************
.dseg
.org 0x200
```

Directions: (This will require the use of the AVR Instruction Set reference document.)

- Follow the procedure described in the labs to use AVR studio to create a project called `A3P1.aps`.
- Type or copy and paste the shell above.
- Build and (if no errors) simulate the run (or debug) of the program. It really should not do anything, as there are no statements in the code segment.
- Type or copy and paste the first three lines of the pseudo-code above into the code segment, placing a semi-colon (;) at the beginning of each line, making them comments.

- After each line use an AVR load instruction, where one operand is a register and the other uses immediate addressing, to create these instructions in assembly language. Build and simulate program execution, checking for correctness.
- Build and (if no errors) simulate the run (or debug) of the program.
- Create a comment for the `sum = 0` instruction, then realize in assembly with a clear instruction.
- Create a comment for each of the lines that add the sum and realize with add instruction that use the add instruction, where both operands are registers.
- Build and (if no errors) simulate the run (or debug) of the program.
- (A short diversion from coding to set up the SRAM) Create a memory location for the `result`, which we will assume to be 1 byte long, in the Data Segment by typing `result .db 1` on the very last line, immediately following the `.dseg` and `.org 0x200` assembler directives. Observe that the SRAM memory address (in hexadecimal) `0x200` has been named `result` and 1 byte has been set aside for it.
- Build and (if no errors) simulate the run (or debug) of the program. Observe the SRAM at location `0x200` in the simulator.
- (Back to writing the code segment) Create a comment for the `result = sum` instruction. Use the store instruction with *direct* addressing to specify the location of `result`.
- Build and (if no errors) simulate the run (or debug) of the program. Observe the SRAM at location `0x200` in the simulator.
- Adjust the top of code comments to appropriately describe the program.

(Similar to page 68, #3 of the Margush textbook): You have just written an AVR assembly program that adds three (byte sized) numbers together. The inputs for the program are constant (hard coded) values, and the final output 'result' is stored in data memory segment, starting at location 0x200.

Make sure your code is fully documented, as described above. Observe that the AVR Studio created a file in your directory called A3P1.asm.

Submit your fully documented code file `A3P1.asm` to the CSC 230 conneX site well before 5 am on Monday, October 2, 2017. (Be careful to submit A3P1.asm, not something with .aps or .hex or other extension!!)

*******IMPORTANT*** Your file is *NOT* submitted until the conneX site has sent you an email confirming submission. Keep that email until you *KNOW* your assignment has been graded.***

Assignment 3 Part 2: Both Formative and Summative

Part 2: Follow the procedure described in the labs to use AVR studio to create a project called `A3P2.aps`.

1. Use a loop to write decrementing hexadecimal numbers into consecutive memory locations in the data memory, starting at memory address 0x200. The starting number, which is between 0x0 (exclusive) and 0x15 (inclusive) should be placed into a register (R16) before the subroutine begins. That starting number needs to be stored in the first data memory location. Then the number will be decremented by 1 and stored in the next consecutive memory location. This process of decrementing and storing will continue until the number 0 is the final value stored. After each number is stored in memory, output the binary equivalent of the number on the LEDs attached to Port L. Here is the pseudo-code:

```
number = /* choose a number in (0x00, 0x15] */ ;
count = 0;
while (number > 0) {
      dest[count++] = number;
      * Output number on LEDs *
      * delay 0.5 second *
      number --;
```

   Observe that your documentation *should* include the above pseudo code.

2. The third line from the end of the above pseudo code (`* Output number on LEDs *`) is described as follows: just after each number is placed in memory, display the number, in binary on the LEDs on the board that are attached to Port L.

   Recall from lab 4 that the number must be spread out on every other bit to output to the port. For example, to output the number 0x15 = 0b1111 on port L, each of those 4 bits must be followed by another bit, lets use 0, such that the binary number becomes 0b10101010. Observe that the 4 underlined bits were the original bits. The conversion from (for example) 0b1111 to 0b10101010 is accomplished using a 'mask' to separate out a single bit, shifting it to the proper location and then using an OR operation to combine the bits together.
3. The third line from the end of the above pseudo code (`* delay 0.5 second *`) can use the following code:

```
            ldi r24, 0x2A     ; approx. 0.5 second delay
outer:      ldi r23, 0xFF
middle:     ldi r22, 0xFF
inner:      dec r22
```

```
brne inner
dec r23
brne middle
dec r24
brne outer
```

Submit your fully documented code file, `A3P2.asm`, that was created by the AVR studio program to the CSC 230 conneX site well before 5 am on Monday, October 9, 2017.

****IMPORTANT*** Your file is *NOT* submitted until the conneX site has sent you an email confirming submission. Keep that email until you *KNOW* your assignment has been graded.***