AVR Assembly Programming Problems

Programming Problems

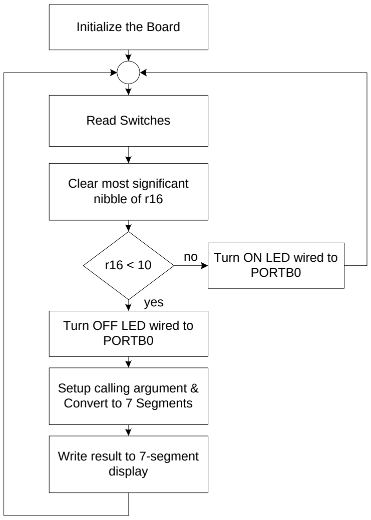
- The following programming problems are designed for the Arduino Uno (ATmega328P) with the CSULB Shield. Programs may be written in AVR Studio 4 or Atmel Studio 6.
- To display code, open in Chrome browser or download (Firefox).

1 In this programing problem you will write the assembly code needed to display a number between 0 and 9 as defined by the least significant 4 switches on your proto-shield (PINC). If the number is greater than 9 turn ON the discrete LED wired to PORTB bit 0, otherwise turn OFF the LED.

I have written much of the code, including calls to subroutines **InitShield, WriteDisplay** and **BCD_to_7SEG**. You should be familiar with the first two from your Lab work. The BCD_to_7SEG subroutine takes as input a number between 0 and 9 in **register r0**. The subroutine then converts the decimal number into its corresponding 7 segments and displays answer.

As you write your program remember that

- The least significant 4 switches are wired to **PINC**.
- The error LED is wired to **PORTB bit 0**
- BCD_to_7SEG's calling argument is in **register r0**
- **Do not modify r16** when you check to see if it is less than 10


Initialize the Board
Read Switches
Clear most significant nibble of r16
r16 < 10
no
Turn ON LED wired to PORTB0
yes
Turn OFF LED wired to PORTB0
Setup calling argument & Convert to 7 Segments
Write result to 7-segment display

```
.INCLUDE <m328pdef.inc>
RST_VECT:
    rjmp   reset
.ORG 0x0100
.INCLUDE "spi_shield.inc"
reset:
    ldi    r16,low(RAMEND)
    out    SPL,r16
    ldi    r16,high(RAMEND)
    out    SPH,r16
; Initialize Proto-shield
    call   InitShield
loop:
    _____    r16, _____   // Read Switches from GPIO Registers
    _____    r16, 0x___   // clear most significant nibble
    _____    r16, 0x___   // Is r16 less than $10_{10}$? (see notes)
    _____   ___________   // unsigned conditional branch
    _____   ______, ___   // error - turn on the LED
    rjmp    ___________    // see the flowchart
no_error:
    _____   ______, ___    // not an error - turn off the LED
display:
    _____   ____, _____    // send argument to subroutine
    call   BCD_to_7SEG     // (see notes)
    call   WriteDisplay
    rjmp    ____________
```

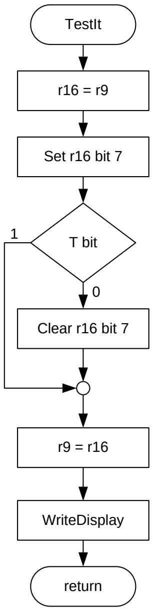
2 Write a subroutine named BlinkIt to complement a variable named blink and to then send the least significant bit (b0) of blink, in SREG bit T, to a subtoutine named TestIt. For the purpose of this exam you do not need to save registers on the stack for this question.

```
.DSEG
Blink:  .BYTE   1
.CSEG
```

```
BlinkIt:
    ______   _________      // load variable to register 16
    ______   _________      // do something
    ______   _________      // store register 16 back to variable
    ______   _________      // store bit 0 to SREG T bit
    ______   _________      // call TestIt using relative addressing mode
    ret
```

BlinkIt Project Code Simulation

3 The following flowchart defines a subroutine named TestIt which is called by BlinkIt. TestIt therefore accepts the T bit as an argument. We are working with the Arduino Proto-shield (see Proto-shield Schematic). Translate the following flowchart into its equivalent AVR code. Your code must implement the flowchart on the right. For the purpose of this exam you do not need to save any registers on the stack for this question.



```
.DEF  spiLEDS = r9

TestIt:
    ______   _________
    ______   _________
    ______   _________    ; guess bit is set
;execute next line only if t = 0
    ______   _________    ; guess is wrong
done:
    ______   _________
    ______   _________
    ret
```

TestIt Project Code Simulation

4 Given variables A and B, each holding an 8-bit signed 2's complement number. Write a program to find the maximum value and put into variable C. Example if A > B then C = A.

Option A: Basic implementation of if-then-else statement using load -> do something -> store structure

Max1 Project Code Simulation

Option B: Basic implementation of if-then-else statement. Structure modified to immediately store result.

Max2 Project Code Simulation

Option C: If-then-else statement restructured to if-then with guess. Result immediately stored in SRAM.

Max3 Project Code Simulation

5 Given variable A holds an 8-bit signed 2's complement number. Write a program to find the absolute value A. Save result back into variable A.

A = |A|

Abs Project Code Simulation

6 Write a program to add 8 bit variables A and B together. Store the sum into 8 bit variable C. For this programming problem you may assume that the sum is less than 255 if A and B are unsigned and between -128 and 127 if signed.

C = A + B

Adder88 Project Code Simulation

7 Write a program to find the sum of **unsigned** 8 bit variables A and B. For this programming problem the sum may be greater than 255 if A and B. Store the sum into 16 bit variable C using little endian byte ordering.

C = A + B

Adder816 Project Code Simulation

8 Write a program to find the sum of **signed** 8 bit variables A and B. For this programming problem the sum may be less than -128 and greater than 127. Store the sum into 16 bit variable C using little endian byte ordering.

C = A + B

Adder816s Project Code Simulation

9 Multiply 8-bit unsigned variables A and B placing the product into 16-bit variable C. Save the 16-bit product using little endian byte ordering.

C = A x B

Mul8x8_16 Project Code Simulation

10 Given 8-bit variables A and B, each holding an 8-bit **unsigned** number. Write a program to find the average of A and B. Place the result into variable C.

Hint: Shifting (or rotating) a binary number to the left divides the number by 2.

Avg Project Code Simulation

11 Given 8-bit variables A and B, each holding an 8-bit **signed** 2's complement number. Write a program to find the average of A and B. Place the result into variable C.

Hint: Shifting (or rotating) a binary number to the left divides the number by 2.

Avg8s2 Project Code Simulation

12 Write a function named Div8_8 to divide an unsigned 8 bit number by an unsigned 8 bit number. You can find this program in your textbook (Mazidi). Test your function by writing a program named Div8_8test to test the subroutine Div8_8 by dividing the 8-bit-number: 0xAA by the 8-bit-number 0x55.

Div8_8 Project Code Simulation

13 Write a function named Div16_8 to divide an unsigned 16 bit number by an unsigned 8 bit number. Test your function by writing a program named Div8_test to test the subroutine Div16_8 by dividing the 16-bit-number: 0xAAAA by the 8-bit-number 0x55.

Option A

Div8 Project Code Simulation

Option B
This solution is an extension of Div8_8
Div16_8B Project Code Simulation

14 Write a subroutine that convert a temperature reading in Fahrenheit (variable F)to Celsius (variable C).

ConvertCtoF Project Code Simulation

15 Write a subroutine that convert a temperature reading in Celsius (variable C) to Fahrenheit (variable F).

ConvertFtoC Project Code Simulation

16 Given variables A, B, and C; each holding an 8-bit unsigned number. Write a program to find the average of A to C, placing the result into variable D.

D = A + B + C / 3

Allow for a 16-bit interim sum and result.

AvgABC Project Code Simulation

17 Write a program to multiply a 16-bit unsigned number in the r25:r24 register pair by an 8-bit number in r26.return the answer in r4:r3:r2

Remember that when you multiply two numbers the result is in the r1:r0 register pair. Study the following multiplication example to see how it is done.

```
   Multiply
     r25:r24
      x   r26
   ----------
    0:r3:r2  = r16*r18  first byte
 + r1:r0     = r17*r18  second byte
   --------
   r4:r3:r2  = (r1 + c):(r3 + r0):r2
```

For additional help read "Binary hardware multiplication in AVR Assembler" at http://www.avr-asm-tutorial.net/avr_en/calc/HARDMULT.html

Test your hardware multiplication 16-by-8-bit subroutine.

Mul16 8 24 Project Code Simulation

18 Calculate A^2 where A is an 8-bit unsigned variable. The result is placed into 16-bit variable C. The 24-bit result is saved using little endian byte ordering.
$C = A^2$

A Squared Project Code Simulation

19 Calculate A^3 where A is an 8-bit unsigned variable. The result is placed into 24-bit variable C. The 24-bit result is saved using little endian byte ordering.
$C = A^3$

A Cubed Project Code Simulation

20 Calculate the factorial of the number held in variable A. The number in variable A must be greater than 0 and less than or equal to 6! = 720 (note: 5! = 120). Store factorial of A into 16 bit variable C. Byte ordering is little endian.

C = A!

Fact1 6 Project Code Simulation

21 Calculate the factorial of the number held in variable A. The number in variable A must be greater than 0 and less than or equal to 8! = 40,320

Store factorial of A into 16 bit variable C. Byte ordering is little endian.

C = A!

Fact1 8 Project Code Simulation

22 The previous programming problem limited the input to a number from 1 to 8. Modify the program to return 1 if the input is 0 and turn on the green LED wired to Port B bit 0 on the CSULB shield if the input is greater than 8.

Fact0 8 Project Code Simulation

23 Write a program to convert an 8-bit ASCII encoded number character ('0' to '9') into its BCD equivalent value (0 to 9). Assume the ASCII character is stored in SRAM variable ascii_char. The BCD result is saved SRAM variable bcd_value. Hint: Review Lecture 14 Logic and Shift Instructions.

ASCIItoBCD Project Code Simulation

24 Write a program to convert a BCD number (0 to 9) into its 8-bit ASCII encoded character ('0' to '9') equivalent. Assume the BCD value is stored in SRAM variable bcd_value. Return the ASCII character in SRAM variable ascii_char. Hint: Review Lecture 14 Logic and Shift Instructions.

BCDtoASCII8 Project Code Simulation

25 Write a function to unpack two BCD numbers in register r24 and then convert their binary equivalent values into two ASCII characters. Return ASCII characters in the r25:r24 register pair.

BCDtoASCII Project Code Simulation

26 Write a function to convert a packed BCD number in register r24 into its binary equivalent. Return 8-bit binary result in register r24.

For example if r24 = 32 then result r24 = 3 x 10 + 2

Source:

1. Beginners Introduction to the Assembly Language of ATMEL-AVR-Microprocessors http://www.ic.unicamp.br/~celio/mc404-2008/docs/beginner avr.pdf
2. AVR204: BCD Arithmetics http://www.atmel.com/images/doc0938.pdf

Solution:

BCDtoBINARY Project Code Simulation

27 Write a function to convert a 16-bit number into a packed BCD number. Return packed BCD result in register r24.

For example if r24 = 0x32 then result r24 = 0b0010 0000

```
Input:  r25:r24 a 16-bit binary number (0 to 65,535)
Output: r25:r24,r23:r22:r21:r20  (r25 = 0)
```

Source:

1. Beginners Introduction to the Assembly Language of ATMEL-AVR-Microprocessors http://www.ic.unicamp.br/~celio/mc404-2008/docs/beginner_avr.pdf

2. AVR204: BCD Arithmetics http://www.atmel.com/images/doc0938.pdf

Solution:

BINARYtoBCD Project Code Simulation

28 In the previous programming problem you wrote a function to convert a 16 bit number into its packed BCD equivalent. A more elegant solution uses the indirect addressing mode to accomplish the same task. Rewrite your solution using the indirect instruction lpm.

a. Begin by defining a constant table (16-bit with little endian byte ordering)

```
Bin2BCD_Table:

   DW 10,000, 1,000, 100, 10
```

b. Next read word-wise with the lpm instruction from the table and construct a loop for converting from binary to bcd.

Sources:

1. Beginners Introduction to the Assembly Language of ATMEL-AVR-Microprocessors http://www.ic.unicamp.br/~celio/mc404-2008/docs/beginner_avr.pdf

2. AVR Addressing Indirect http://www.csulb.edu/~hill/ee346/Lectures/13%20AVR%20Addressing%20Indirect.pdf

Solution:

BINARYtoBCD2 Project Code Simulation

29 Write a subroutine to convert a BCD number into an 8-bit value ready to be sent to the 7-segment display on the CSULB shield.

Source:

1. AVR Addressing Indirect http://www.csulb.edu/~hill/ee346/Lectures/13%20AVR%20Addressing%20Indirect.pdf

Solution:

BCDto7SEG Project Code Simulation

Implement the following Taylor Series Expansions (Programs 15 to 21)
Source:

1. Taylor Series http://en.wikipedia.org/wiki/Taylor_series
2. AVR204: BCD Arithmetics http://www.atmel.com/images/doc0938.pdf

30 sin(x) to the third order

$$\sin(x) \approx x - \frac{x^3}{3!}$$

TaylorSine3 Project Code Simulation

31 cos(x) to the fourth order

$$\cos x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!}$$

TaylorCosine4 Project Code Simulation

32 e^x to the second order

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} \cdot$$

TaylorExpX2 Project Code Simulation

33 e^x to the fourth order

TaylorExpX4 Project Code Simulation

34 (1+x)^0.5 to the second order

$$(1+x)^{0.5} = 1 + \tfrac{1}{2}x - \tfrac{1}{8}x^2$$

Taylor1pX2 Project Code Simulation

35 log(1+x) to the third order

$$\log(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} \cdot$$

TaylorLog1pX3 Project Code Simulation

36 e^x / cos(x) to the second order

$$\frac{e^x}{\cos x} = 1 + x + x^2 \cdot$$

TaylorEX_cosX2 Project Code Simulation

37 Using the indirect addressing mode instruction st clear registers initialize registers r0 to r20 with values 0x15 to 0x01

Hint: SRAM address 0x0000 to 0x0001F map to registers r0 to r31

PreReg0 20 Project Code Simulation

38 Using the indirect addressing mode instruction st clear registers r0 to r15. To test your subroutine, begin by adding and then calling subroutine PreReg0_20 in the setup section of your code.

Hint: SRAM address 0x0000 to 0x0001F map to registers r0 to r31

Clr0 15 Project Code Simulation

39 Using the indirect addressing mode instructions ld and st, push registers r0 to r15 onto a stack buffer starting at SRAM address 0x04FF. To test your subroutine, begin by adding and then calling subroutine PreReg0_20, in the setup section of your code.

Push0_15 Project Code Simulation

40 Using the indirect addressing mode instruction ld and st, pop registers r15 to r0 from a stack buffer starting at SRAM address 0x4FF. To test your subroutine, begin by adding and then calling subroutines PreReg0_20, Push0_15, Clr0_15, Pop15_0 in the setup section of your code.

Pop15 0 Project Code Simulation

41 Write a program to test the following subroutine.

Source:

1. AVR Instruction Set, page 73
   http://www.csulb.edu/~hill/ee346/Reference/AVR%20Instruction%20Set doc0856.pdf

```
/* Signed multiply of two 16-bit numbers with 32-bit result.
 * Usage
 *  c3: c2: c1: c0 =  ah:al  *  bh:bl
 * r19:r18:r17:r16 = r23:r22 * r21:r20
*/
muls16x16_32:
   clr   r2
   muls  r23, r21  ; (signed)ah * (signed)bh
   movw  r19:r18, r1:r0
   mul   r22, r20  ; al * bl
   movw  r17:r16, r1:r0
   mulsu r23, r20  ; (signed)ah * bl
   sbc   r19, r2
   add   r17, r0
   adc   r18, r1
   adc   r19, r2
   mulsu r21, r22  ; (signed)bh * al
   sbc   r19, r2
   add   r17, r0
   adc   r18, r1
   adc   r19, r2
   ret
```

muls16x16_32 Project Code Simulation

42 Assume three signed numbers are formatted using (1.7) notation, a fractional number (N.Q) with N binary digits left of the radix point and Q binary digits right of the radix point. Write a program to find the average of 5 numbers (array A[5]) putting the result into variable C.

Avg5 Project Code Simulation

43 Write a subroutine to count the number of 'space' characters in a buffer located in SRAM. Return the count in register r0. Inspiration for this programming problem came from problem 1 located here.

Note: In the ASCII character set, the code for a space is 0x20 (base 16).

CountSpaces Project Code Simulation

44 You are probably familiar with the formula $\sum_{i=0}^{n} i = \frac{n^2+n}{2}$

Suppose that you were skeptical of this formula and no one showed you a proof. You try it for a few cases and it works, but you are having difficulty finding a proof yourself. Before looking harder for a proof you'd like to verify it for a few hundred cases or more, by computer.

Write a program which verifies this equation (i.e. that the two sides are indeed equal), for 0 <= n < N.

The answer should be in r0 when your program ends. If overflow occurs at any point in your calculation, your program should exit with -2 in r0. Otherwise, if it finds a counterexample to this formula, it should exit with the n counterexample in r0 and with carry set (and overflow clear). If it verifies the formula for all n, 0 <= n < N, it will exit with -1 in R0.

Your program needn't be especially well-commented but you should explain your use of registers and anything else particularly unclear.

Source: http://www.cdf.toronto.edu/~ajr/258/probs/velma/ problem 3

Solution:

FormProof Project Code Simulation

45 Write a program to count how many integers in a certain range are prime. An integer greater than one is prime if and only if it has only two factors: itself and 1. The limits of the range will be stored in registers R3 and R4 before your program is started, and the range is inclusive. For example, if R3 contains 11_{10} and R4 contains 19_{10}, the answer would be 4, because the prime numbers in that range are (only) 11, 13, 17, and 19. Your program should end with the count in R0. If R3 > R4, the count is zero.

Source: http://www.cdf.toronto.edu/~ajr/258/probs/velma/ problem 8

Primes Project Code Simulation

46 Write a program to find the square root of a 16-bit number.
Author: Nicholas Lombardo
Inputs: N, 16-bit number with arbitrary radix point, between R and R+1 bit
Output: S, 16-bit sqrt(N) radix point shifted left to be between (R+R/2) and (R+R/2)+1
EX. for R=0, N is integer. SH will have an integer, SL will be values right of radix

1. guess a square root starting at 0x8000
2. increase/decrease next bit according to compare
3. repeat up to 16 times

```
ex. guess up:      0x8000 --> 0xC000
ex. guess down:    0x8000 --> 0x4000
```

Sqrt Project Code Simulation

Extra Credit

Extra credit may be earned by helping me add the missing links (Project Folder, Simulation). Simulating an existing program is the simplest method for earning extra credit. Here are a few things to remember when simulating a program.

- Explain how simulation was run, input conditions tested along with expected outputs.
- Show variable values in decimal and hexadecimal.
- If branch is present include two tests with both paths taken.
- Do not modify code unless there is an error. In this case please let me know how code was fixed.
- To initialize variables and/or registers, add to watch window and enter values at the start of the simulation (double click).
- Based on nature of the problem, display values in hexadecimal or decimal (right-click in watch window, select "Display all Values as Hex.")
- Provide your simulation document in Microsoft Word and PDF formats.
- Save to your Public Drop Box folder and send link to me in an email. Please send to my gmail address.
- Submissions are handled on a first come first serve basis. Extra credit is given only when your simulation document is posted.
- Be prepared for at least 3 revisions.

Extra credit may also be earned by writing the solution to a programming problem, where none is provided or creating your own programming problem. You may also submit programming problems found on the web or a textbook. In these cases you must provide a link to your source.

Earning extra credit is a very time intensive activity for you and me. In most cases I will request that you revise your simulation or program. This should be expected and is not a problem. However, you may only submit 2 written clarification questions[1] and submit only 2 revisions before the exercise is ended. Please choose another programming problem at this point.

Extra credit earned is at the discretion of the instructor and will be applied to your third Quiz grade, up the maximum number of points assigned to the quiz.

[1] This includes a drop box folder, which I cannot directly open by clicking on the link – no log-in required.