CPE301 – SPRING 2019

Design Assignment 1b

Student Name: Daniel Senda

Student #: 5002362633

Student Email: sendad1@unlv.nevada.edu

Primary Github address: https://github.com/dsenda/Smiles

Directory: DA1b

Submit the following for all Labs:

1. In the document, for each task submit the modified or included code (only) with highlights and justifications of the modifications. Also, include the comments.
2. Use the previously create a Github repository with a random name (no CPE/301, Lastname, Firstname). Place all labs under the root folder ESD301/DA, sub-folder named LABXX, with one document and one video link file for each lab, place modified asm/c files named as LabXX-TYY.asm/c.
3. If multiple asm/c files or other libraries are used, create a folder LabXX-TYY and place these files inside the folder.
4. The folder should have a) Word document (see template), b) source code file(s) and other include files, c) text file with youtube video links (see template).

1. **COMPONENTS LIST AND CONNECTION BLOCK DIAGRAM w/ PINS**

Atmel Studio 7 and Summation Calculator.

1. **INITIAL/MODIFIED/DEVELOPED CODE OF TASK 1/A**

The following code was not modified for part 1b of this assignment. It was included either way.

; DA1a.asm

; Daniel Senda

.org 0x000

start:

ldi r21, 0x0 ;Set register r23 to zero, "Zero register".

ldi r18, 0x0 ;Set register r24 to zero. "Low byte of final answer".

ldi r19, 0x0 ;Set register r24 to zero. "Mid byte of final answer".

ldi r20, 0x0 ;Set register r24 to zero. "High byte of final answer".

ldi r23, 0x0 ;Set register r23 to zero. "Overflow sum of multiplicand."

ldi r24, 0b00100011 ;Load low byte of multiplicand.

ldi r25, 0b00000000 ;Load high byte of multiplicand.

ldi r22, 0b00010111 ; Load 8-bit multiplier.

main\_loop:

lsr r22 ;Logical shift right of multiplier.

brcs lsb\_multiplier ;Branch to lsb\_multiplier if carry is set.

add r22, r21 ;add "zero register" and multiplier.

breq end ;If last result is zero, branch to end.

lsl r24 ;Logical shift left low byte of multiplicand.

rol r25 ;Rotate left with carry high byte of multiplicand.

rol r23 ;Rotate left with carry overflow sum of multiplicand.

jmp main\_loop ;Jump to main loop.

lsb\_multiplier:

add r18, r24 ;Add low byte of multiplicand with low byte of answer register.

adc r19, r25 ;Add with carry high byte of multiplicand with mid byte of answer register.

adc r20, r23 ;Add overflow sum of multiplicand with the high byte of answer register.

lsl r24 ;Logical shift left low byte of multiplicand.

rol r25 ;Rotate left with carry high byte of multiplicand.

rol r23 ;Rotate left with carry overflow sum of multiplicand.

jmp main\_loop ;Jump to main loop.

end: jmp end ;End loop, signifying that it finished.

1. **DEVELOPED MODIFIED CODE OF TASK 2/A from TASK 1/A**

The following is the working code of DA1b.

; DA1b.asm

; Daniel Senda

.org 0x000 ;Sets program origin to 0x000.

start:

ldi r20, 11 ;Loads 11 into r20.

ldi r21, 99 ;Loads 99 into r21.

ldi xh, 0x02 ;Loads 0x02 into high byte of x register.

ldi xl, 0x00 ;Loads 0x00 onto low byte of x register.

;Sets the address pointer to 0x0200.

loop1:

st x+,r20 ;Store value of r20 into address pointed by x register, then moves pointer by 1.

inc r20 ;Increments value of r20.

dec r21 ;Decrements r21 to keep track of stored values.

brne loop1 ;Branch to first\_loop if r17 not zero.

ldi r21, 100 ;Loads 100 into r21.

ldi yh, 0x04 ;Loads 0x04 into high byte of y register.

ldi yl, 0x00 ;Loads 0x00 into low byte of y register.

ldi zh, 0x06 ;Loads 0x06 into high byte of z register.

ldi zl, 0x00 ;Loads 0x00 into low byte of z register.

ldi r16, 0x00 ;Set r16 to zero. "Low byte" of sum1 (numbers divisible by 3).

ldi r17, 0x00 ;Set r17 to zero. "High byte" of sum1 (numbers divisible by 3).

ldi r18, 0x00 ;Set r18 to zero. "Low byte" of sum2 (numbers not divisible by 3).

ldi r19, 0x00 ;Set r19 to zero. "High byte" of sum2 (numbers not divisible by 3).

ldi r22, 0x00 ;Set register r23 to zero, "Zero register".

div\_test: ;Loop that sorts out numbers that are divisible by 3.

ld r20, -X ;Loads value stored at the address indicated by register X into r20.

dec r21 ;Decrements r21 to keep track of values sorted.

breq end ;Goes to end once all numbers are sorted.

loop2:

subi r20, 3 ;Subtracts 3 from value stored in r20

cpi r20, 9 ;Compares r20 with 9.

breq div\_by3 ;Branches if r20 equals 9.

cpi r20, 9 ;Compares r20 with 9.

brmi ndiv\_by3 ;Branches if r20 is less than 9.

rjmp loop2 ;Jumps back to loop2.

div\_by3:

ld r20, X ;Loads value stored at the address indicated by register X into r20.

st y+, r20 ;Store value of r20 into address pointed by y, then moves pointer by 1.

add r16, r20 ;Add value in r20 with value in r16. Stores result in r16(low byte of sum1).

adc r17, r22 ;Add zero with carry to value in r17. Stores result in r17(high byte of sum1).

rjmp div\_test ;Jumps back to div\_test.

ndiv\_by3:

ld r20, X ;Loads value stored at the address indicated by register X into r20.

st z+, r20 ;Store value of r20 into address pointed by z, then moves pointer by 1.

add r18, r20 ;Add value in r20 with value in r18. Stores result in r18(low byte of sum2).

adc r19, r22 ;Add zero with carry to value in r19. Stores result in r19(high byte of sum2).

rjmp div\_test ;Jumps back to div\_test.

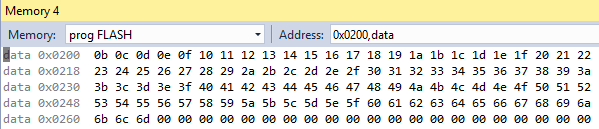
end: rjmp end ;End loop.

1. **SCHEMATICS**

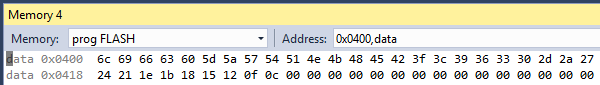
Not applicable.

1. **SCREENSHOTS OF EACH TASK OUTPUT (ATMEL STUDIO OUTPUT)**

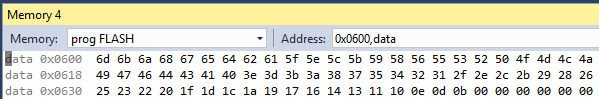
The first requirement if storing 99 values is demonstrated here. The values stored are 11-109.



The next requirement was to sort the values into to two groups. The first group is numbers divisible by three and are stored in the addresses 0x0400 onward. (33 numbers)



The second group is numbers that were not divisible by three and are stored in the addresses 0x0600 onward. (66 numbers)



The third requirement was to add the numbers in group 1 to get a total. (0x07BC = 1980)

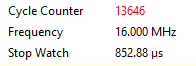


Also had to add the numbers in group 2 to get a total. (0x0F78 = 3960)



The fourth requirement was to verify the code. The code was verified by using a summation calculator to verify that the code was outputting the correct values. The numbers matched, thus confirming that the code is correct.

The fifth requirement was to determine the excecution time running at 16MHz.



1. **SCREENSHOT OF EACH DEMO (BOARD SETUP)**

Not applicable.

1. **VIDEO LINKS OF EACH DEMO**

Not applicable

1. **GITHUB LINK OF THIS DA**

https://github.com/dsenda/Smiles/tree/master/DA1b

**Student Academic Misconduct Policy**

<http://studentconduct.unlv.edu/misconduct/policy.html>

“This assignment submission is my own, original work”.

Daniel Senda