CPE301 – SPRING 2019

Design Assignment 1, Part B

Student Name: James Skelly
Student #: 2000945485
Student Email: skellj1@unlv.nevada.edu
Primary Github address: https://github.com/skellj1/submission_da
Directory: skellj1/submission_da

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

- Components used for this assignment include Atmel Studio 7 Simulator (for programming in assembly, viewing register and memory contents after execution, and analyzing processor status, status register at termination of program, and cycle counter) and the online hexadecimal calculator (https://www.miniwebtool.com/hex-calculator/) in order to verify correct calculation. The sigma summation tool at https://www.mathsisfun.com/numbers/sigma-calculator.html was also used to verify the final sums in this assignment.

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

```
/*
 * Skelly_James_DA1.asm
 *
 *  Created: 2/3/2019 3:11:23 PM
 *  Author: James Skelly (CPE 301, Sp. 2019)
 */

 .ORG 0X00                      ; Sets the program to begin at memory location 0x00

 .EQU MULTIPLICAND = 0XFFFF     ; Initializes the multiplicand (16-bit value)
 .EQU MULTIPLIER = 0XFF         ; Initializes the multiplier (8-bit value)

 LDI R25, HIGH(MULTIPLICAND)    ; Places the higher 8-bits of the multiplicand into register 25
 LDI R24, LOW(MULTIPLICAND)     ; Places the lower 8-bits of the multiplicand into register 24
 LDI R22, MULTIPLIER            ; Loads the multiplier value into register 22
 LDI R21, MULTIPLIER            ; Keeps a copy of the multiplier in register 21 for review when
                                        ;program terminates
 LDI R16, 0x00                  ; Places the value zero into register 16

 LOOP:                          ; Loop label for repeated (iterative) addition
        ADD R18, R24            ; Begins repeated addition of the lower 8 bits of the multiplicand,
                                        ; places the value of the repeated addition in the first solution
                                        ; register, R18
        ADC R19, R25            ; Repeated addition of upper 8 bits, including addition of carry bit
                                        ; from SREG, places result in R19, the second solution register
        ADC R20, R16            ; Allocates bits 16-23 of the third solution register, initially adding
                                        ; the value 0 until further iterations where it will begin to sum
                                        ; up the carry values it receives from the SREG carry bit
        DEC R22                 ; Decrements R17, the multiplier, by 1 on each iteration
        BRNE LOOP               ; Branches back to the top of the LOOP subroutine until R17 is zero

        RJMP END                ; Jumps to the END label

END: RJMP END                   ; Program terminates
```

The above code is simply the first part of Design Assignment 1. This code does not directly have much to do with what was programmed and implemented in part B,.

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

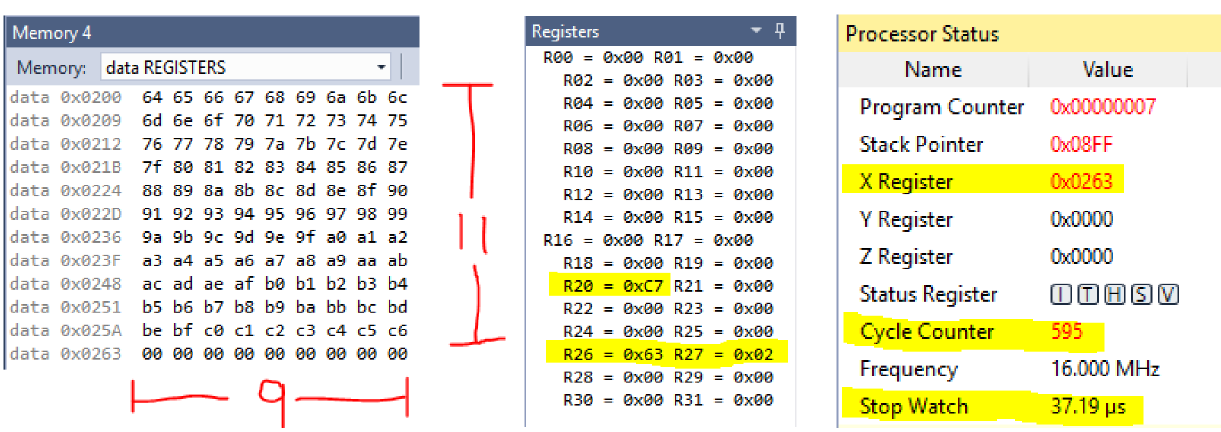
```
; Skellj1_DA1B.asm
;
; Store 99 numbers between 10 and 255, starting at memory location 0x0200, using the X/Y/Z registers.
;
; Created: 2/22/2019 4:22:18 PM
; Author : James Skelly

LDI xl, 0x00            // Load the lower bits of the start address into
                                // the lower 8 bits of the X index register.
LDI xh, 0x02            // Load the upper bits of the start address into
                                // the upper 8 bits of the X index register.
LDI R20, 100            // Load immediate value 100 into register 20.

LOOP:                           // Loop label for loop to populate memory
        st x+, R20              // Store the value of R20 into the memory location
                                // of the X index register, and then increment
                                // the value (memory address) stored in X by 1.
        INC R20                 // Increment the value in R20 by 1.
        CPI xl, 0x63    // Compare the lower bits of the X register (memory
                                // address) with the value 0x63. , or decimal 99.
        BRNE LOOP               // If 0x63 is not the value in the X reg, stay in loop.

END: RJMP END           // Terminates program
```

4. SCHEMATICS

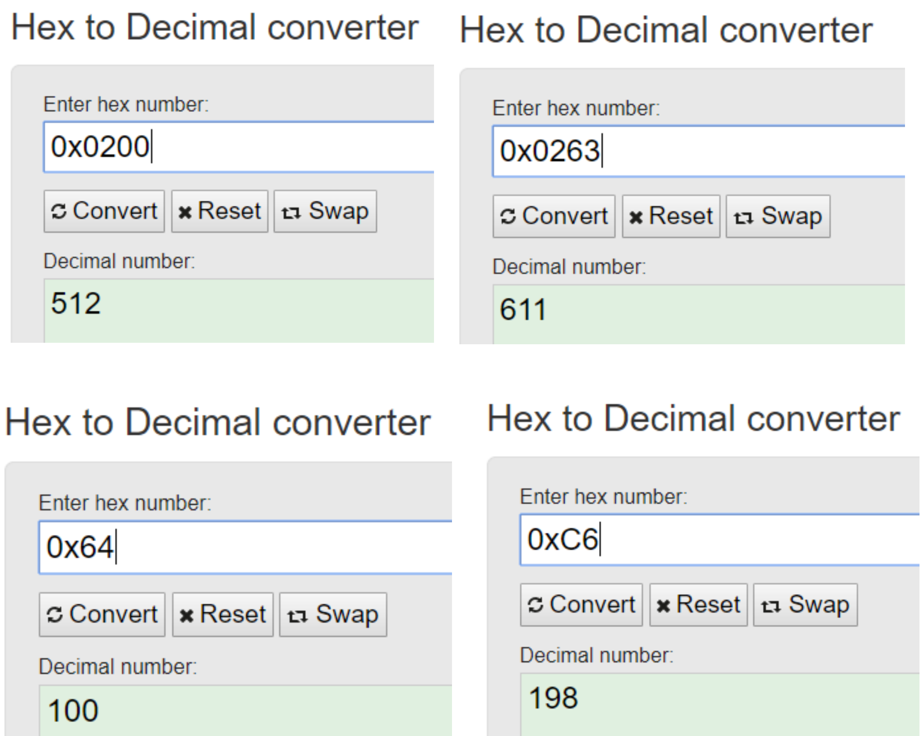
Not available for this assignment.

5. SCREENSHOTS OF EACH TASK OUTPUT (ATMEL STUDIO OUTPUT)

Part 1: Storing 99 Numbers Beginning at Memory Address 0x0200



From the above images, we see in the memory at the end of the program that 99 numbers beginning at memory address 0x0200 with the value 0x64 (decimal 100) and ending at memory address 0x02062 with the value 0xC6 (decimal 198) were correctly stored. We can also observe the registers at the end of the program. The memory address at which the program terminates is held in R26:R27, index register X, and is, as expected, 0x0263, which is 99 locations below the start address. The final value in R20 is 0xC7, one greater than the final value stored in 0x0262, which is 0xC6. The program increments R20 one final time before it terminates, but that value is not stored in the memory. Finally, we can observe that the total number of clock cycles taken for the program to execute is 595, which at 16 MHz, takes the program just 37.19 µs to execute completely.



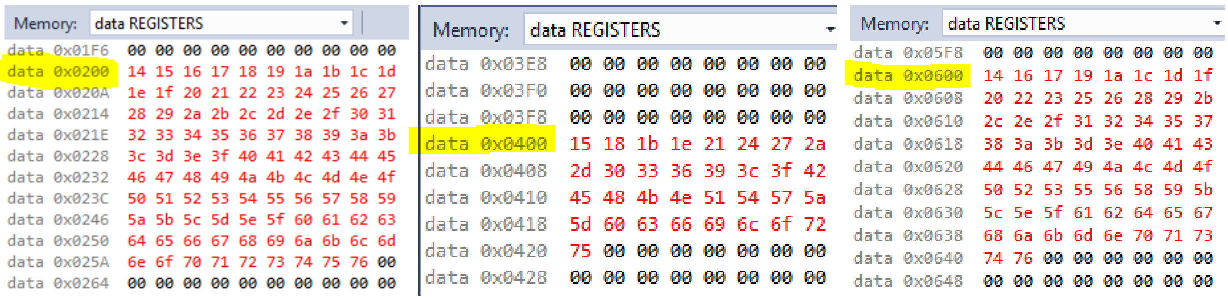
Verification of the correct operation of the program can be confirmed using the values found above from the online ***Hex to Decimal Converter*** found at [https://www.rapidtables.com/convert/number/hex-to-decimal.html?x=0xC6.](https://www.rapidtables.com/convert/number/hex-to-decimal.html?x=0xC6)

Part 2: Separating Numbers Divisible and Not Divisible by 3

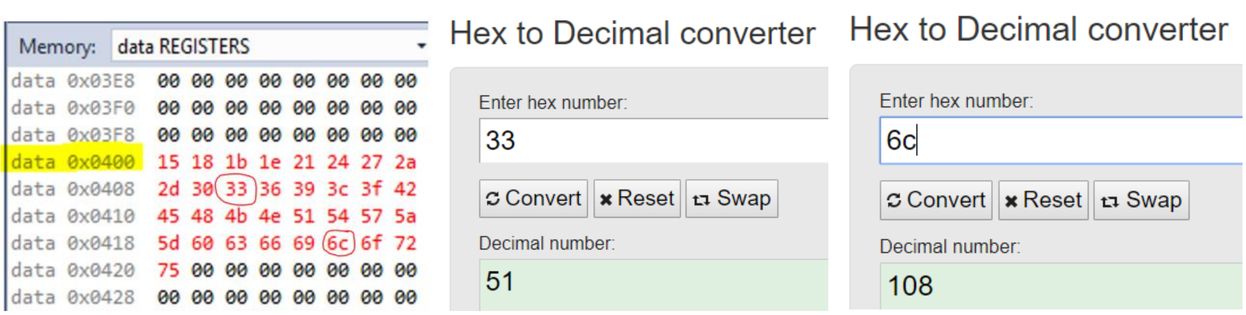
```
; Skellj1_DA1B_part2.asm
; Created: 2/22/2019 4:22:18 PM
; Author : James Skelly

// PART 1 CODE
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

LDI xl, 0x00            // Load the lower bits of the start address into
                                // the lower 8 bits of the X index register.
LDI xh, 0x02            // Load the upper bits of the start address into
                                // the upper 8 bits of the X index register.
LDI R20, 20             // Load immediate value 20 into register 20.

LOOP1:                          // Loop label for loop to populate memory
        ST x+, R20              // Store the value of R20 into the memory location
                                        // of the X index register, and then increment
                                        // the value (memory address) stored in X by 1.
        INC R20                 // Increment the value in R20 by 1.
        CPI xl, 0x63    // Compare the lower bits of the X register (memory
                                // address) with the value 0x63. , or decimal 99.
        BRNE LOOP1              // If 0x63 is not the value in the X reg, stay in loop.
```

```
// PART 2 CODE: Store values that are divisible by 3 starting at memory location 0x0400, and values not
// divisible by 3 starting at memory location 0x0600.
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

LDI xl, 0x00            // Load memory address 0x0200 into index register X.
LDI xh, 0x02

LDI yl, 0x00            // Load memory address 0x0400 into index register Y.
LDI yh, 0x04

LDI zl, 0x00            // Load memory address 0x0600 into index register Z.
LDI zh, 0x06

LOP2:                           // Loop label for loop that will separate numbers that
                                        // are divisible by 3 and numbers that are not divisible
                                        // by 3.
        LD R22, x+              // Load the value stored in the memory location currently
                                        // stored in the X index register, then increment X.
        MOV     R25, R22        // Store a copy of the value pulled from memory in R25
        LDI R23, 3              // Place the number to be divided (repeatedly subtracted)
                                        // into a separate register, R23.
        RCALL mod_divide        // Call the modulus division function.
        CPI R22, 0                      // Compare the value in R22, the result of the mod division,
                                        // with the value zero to test for divisibility by 3.
        BREQ Y_REG              // Branch to Y_REG subroutine if the result of mod division is 0.
        ST z+, R25              // Store the value from R25, the copy of the number pulled from
                                // memory, into the memory location currently stored in the
                                // Z index register, and then increment the value stored in Z.
        CPI xl, 0x63    // Compare the value in the lower X index register with the maximum
                                // memory location, 0x63.
        BREQ END        // If the X register has reached its maximum memory location, branch
                                // to the END label.
        RJMP LOOP2      // Stay in the loop until all of the values have been pulled from the
                                // memory locations held by the X index register.

Y_REG:                  // Subroutine to store numbers divisible by 3 in memory starting at 0x0400.
        ST y+, R25              // Store the value from R25 in the memory location currently stored in
                                // the Y index register, and then increment the value stored in Y.
        CPI xl, 0x63    // Check to see if X has reached the maximum memory location.
        BREQ END        // Branch to the END label if X has reached its maximum memory location.
        RJMP LOOP2      // Jump back to the loop if all the values have not been tested for divisibility.

mod_divide:                             // Modulus division function to check for divisibility by 3.
        again:                          // Again subroutine within mod_divide to repeatedly subtract.
                MOV R24, R22                    // Copies the remaining value into R24 after subraction.
                SUB R22, R23                    // Subtracts 3 from the value pulled from memory
                BRLT LessThanZero    // If the difference is less than zero, branch to less than zero sub.
                CPI  R22, 0                     // Compare the remaining number with zero.
                BREQ EqualToZero        // Branch to equal to zero sub if the number remaining is zero.
                RJMP again              // Jump back to the top of the again sub if the number remaining
                                                // is greater than zero.

        LessThanZero:                   // Label for less than zero subroutine.
                MOV R22, R24            // Move the result of the mod division into R22.
                RJMP DONE               // Jump to DONE

        EqualToZero:                    // Equal to zero sub, R22 will be holding the value 0
                RJMP DONE               // Jump to DONE

        DONE:
                RET                             // Return to the next line to be executed after RCALL

END:                                            // Terminate the program
        RJMP END
```

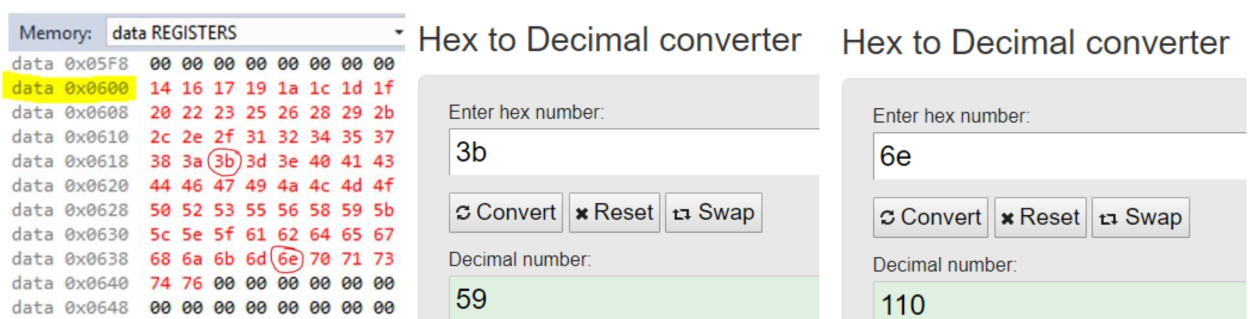
Results from Part 2:



The images above were taken from the *Data Registers* window after the program terminates. As we can see, values starting at 0x14 (decimal 20 was the starting value for this run) all the way up to 0x76 are stored in the first 99 memory locations starting at 0x0200, from part one of the code. Part two of the code separates values that **are divisible by 3** from values that **are not divisible by 3** and places those divisible by 3 in consecutive memory addresses beginning at address 0x0400, and places those not divisible by 3 in consecutive memory addresses beginning at address 0x0600. Testing some random values from the numbers in red up above, we can test for divisibility by 3 using the hex calculator.



- **51/3 = 17 , 108/3 = 36**
- **These numbers are evenly divisible by 3.**



- **59/3 = 19.67, 110/3 = 36.67**
- **These numbers are not evenly divisible by 3**

Processor Status	
Name	Value
Program Counter	0x00000025
Stack Pointer	0x08FF
X Register	0x0263
Y Register	0x0421
Z Register	0x0642
Status Register	I T H S V N Z C
Cycle Counter	18717
Frequency	16.000 MHz
Stop Watch	1,169.81 µs

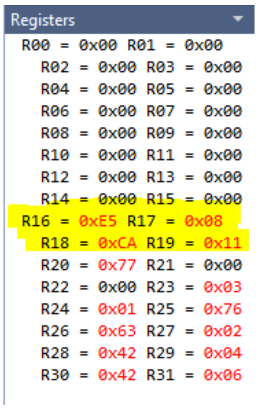
Here we can observe the processor status when the program finishes. The total execution time at 16 MHz was 1.169 ms, and or 18,717 clock cycles. The final value in the X register is 0x0263, where 0x63 is 99 in decimal. This represents the 99 numbers that were stored from part 1. The Y register has a final value of 0x0421, where 0x21 is 33 in decimal. This represents the 33 numbers that are divisible by 3. The Z register has a final value of 0x0642, where 0x42 is 66 in decimal. This represents the 66 numbers that are not divisible by 3.

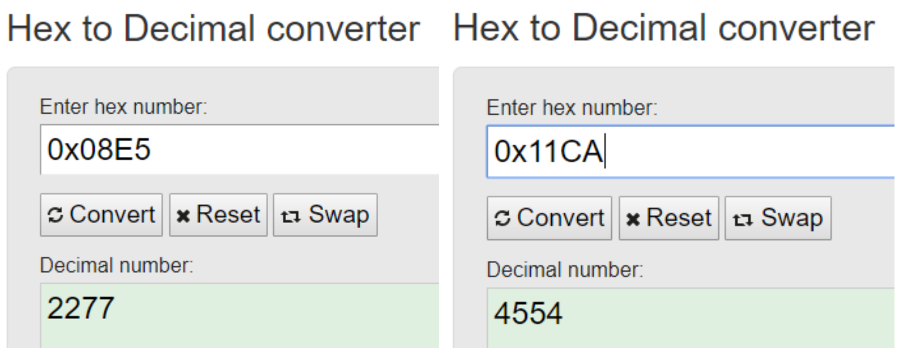
Part 3: Simultaneously Adding Numbers from 0x0400 and 0x0600

```
// PART 3 CODE: Add up all the values that are divisible by 3 (store result in R16:R17) and the values
that are not divisible by 3 (store result in R18:R19).
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

PART3:

        LDI yl, 0x00            // Load memory address 0x0400 back into Y.
        LDI yh, 0x04
        LDI zl, 0x00            // Load memory address 0x0600 back into Z.
        LDI zh, 0x06
        LDI R16, 0x0            // Initialize the divisible by 3 sum reg to 0.
        LDI R18, 0x0            // Initialize the not div. by 3 sum reg to 0.

LOOP3:

        LD R22, y+              // Load the value from the current mem. address of Y, then increment Y.
        ADD R16, R22            // Add the value to the running sum for div by 3 values.
        ADC R17, R0             // Push carry into R17
        LD R25, z+              // Load the value from the current mem. address of Z, then increment Z.
        ADD R18, R25            // Add the value to the running sum for not div by 3 values.
        ADC R19, R0             // Push carry into R19
        CPI zl, 0x42            // Compare low value of Z to upper bound of Z memory
        BREQ END                // Final branch to end
        RJMP LOOP3              // Stay in loop if the sum is not completed

END: RJMP END
```

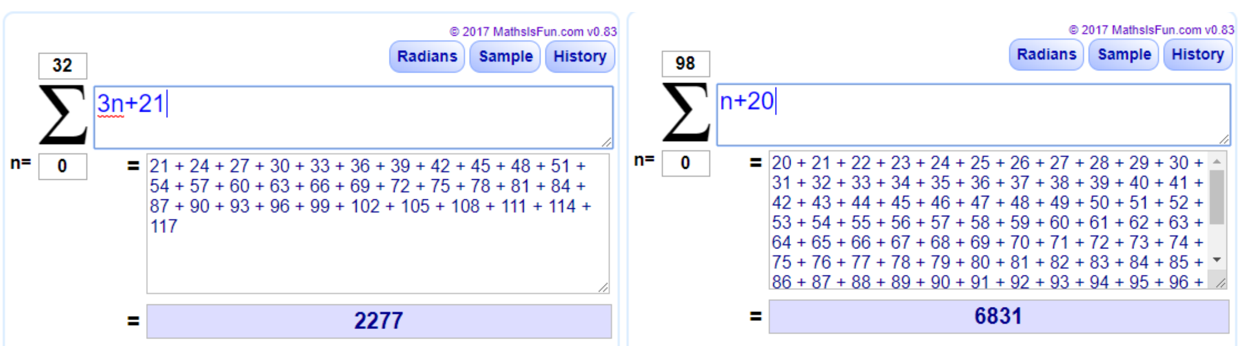
The above code was executed using previously stored values from part 2. For brevity and a clean report, the code for part 2 was not copied into this section. The values that are divisible by 3 begin at memory location 0x0400, so the Y register needed to be reset to 0x0400 in order to pull the first value into the loop to begin the sum. The same goes for the Z register.

Results

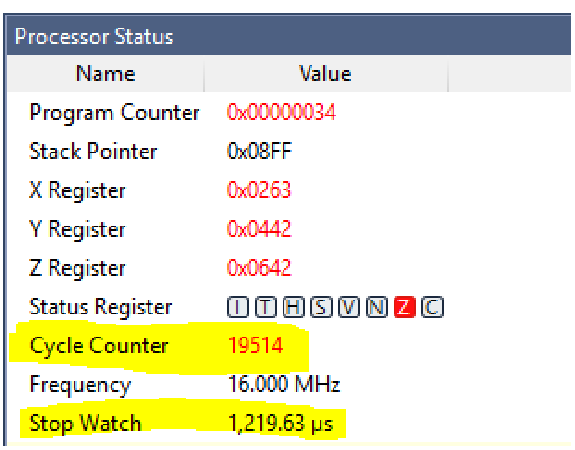




The sum of the values divisible by three can be verified using a summation over 33 values (0 to 32) beginning at 21, the first value after 20 that is divisible by 3. The result of the summation matches the hexadecimal value in the registers shown above.



In order to verify the sum of the values that are not divisible by 3, all 99 of the numbers, 20 through 118, were summed together to obtain a grand total sum of all the numbers. Next, the sum of the divisible by 3 values, 2277, was subtracted from the grand total sum, to give a value of 6831 – 2277 = 4554. This value matches the hexadecimal value in the registers shown up above.



The processor status window shows us that the final execution time to complete all three parts of the code is 1.219 ms, or 19,514 clock cycles at 16 MHz. Notice that the Y register and the Z register low bits both finish at 0x42. This is because there were twice as many values stored for not divisible by three than there were for divisible by three. The Y register needed to increment and add an extra 33 zeros in order for all of the Z values to be pulled out of memory and added.

6. SCREENSHOT OF EACH DEMO (BOARD SETUP)

- No board was used in this assignment.

7. VIDEO LINKS OF EACH DEMO

- No videos were required for this assignment.

8. GITHUB LINK OF THIS DA

- https://github.com/skellj1/submission_da

Student Academic Misconduct Policy
http://studentconduct.unlv.edu/misconduct/policy.html

"*This assignment submission is my own, original work*".
James W. Skelly