REQUIREMENTS NOT MET

N/A

PROBLEMS ENCOUNTERED

The main problem that I encountered was when we were supposed to check the 6th bit, I checked the 5th (I started counting from 1 and not 0). This through me for a loop for literally HOURS. If you actually check the 5th bit instead and your program works fine, your output will be “202” which is close to the expected “2021!”. So, I didn’t think to check which bit until a long time later when reverse engineered how to get the “1!” at the end.

FUTURE WORK/APPLICATIONS

This lab helped me understand some of the basics of assembly which I really needed. Learning how to read from memory and write to memory will always be helpful, especially for future labs. This lab helped me learn simple things like how to use the RAMP registers and how to use directives. Overall, it was a great lab to start working with the XMEGA to get hands on experience. It is way easier to understand things said in lecture or understand what the manuals are saying now that I have physically worked with the XMEGA.

PRE-LAB EXERCISES

i. Go through your parts list (available at here). Identify any missing parts and report this to your PI and in your lab report. If all parts are there, indicate this in your lab report.

No parts were missing

ii. Which type of memory alignment is used for program memory in the ATxmega128A1U? Byte alignment, or word-alignment? What about for data memory?

Program memory: Word-alignment

Data memory: Byte-alignment

iii. In which section of program memory is address 0xF0D0 located?

0xF0D0 is located in the Application Table Section

iv. Which assembler directive places a byte of data in program memory? Which assembler directive allocates space within data memory? Which assembler directives allow you to provide expressions (either constant or variable) with a meaningful name?

Places a byte of data in program memory:	.db
Allocates space within data memory:	.dseg
provide expressions with a meaningful name:	.equ

v. Which assembly instructions can be used to read from (flash) program memory? For each instruction, list which registers can be used as an operand.

Instruction:	Operands:
LPM	Rd, Z
ELPM	Rd, Z

vi. Which assembly instructions can be used to load data indirectly from data memory within XMEGA AU microcontrollers? Which assembly instructions can be used to store data indirectly to data memory?

Load Indirectly:	Store Indirectly:
LD	SD
LDD	STD

vii. Which assembly instruction can be used to load data directly from the I/O memory of XMEGA AU microcontrollers? Which assembly instruction can be used to store data directly to the I/O memory?

Load Directly:	Store Directly:
LDS	STS
LDI	

viii. If you were to use the Memory debug window of Atmel Studio to verify that some datum was correctly stored at address 0xBADD within program memory of the ATxmega128A1U, which address would you specify within the debug window?

0x175BA

ix. When using the internal SRAM (not EEPROM), which memory locations can be utilized for the data segment (.dseg)? Why?

0x2000 – 0x3FFF
Because this is the address of SRAM

x. Which is the first (i.e., lowest) program memory address that would require the relevant RAMP register to be changed from its initial value of zero? Why?

0x10000
In order to access program memory or data memory above 64KB the address pointer must be larger than 16 bits. This is done by concatenating the RAMP register.

xi. In the context of pointing an index to a specific program memory address within an XMEGA AU architecture, explain why and how the address value should first be altered. Similarly, in the context of pointing an index to a specific data memory address, explain why the address value should not be altered

The address should first be doubled. Program memory is saved in 16 bits but the XMEGA can only read 8 bits at a time so it must read twice to get one word. However, data memory is saved in 8 bits so it only needs to be read once.

PSEUDOCODE/FLOWCHARTS

```
Read in value
if(valuebit6 = 1)
{
        value = value/2
        if(value > 55)
        {
                value = value - 8
                store value
        }
}
else
{
        value = value*2
        if(value <= 114)
        {
                value = value + 191
                store value
        }
}
```

PROGRAM CODE

Starting from MAIN

```
MAIN:
                                        ;point appropriate indices to input/output tables
        ldi ZL, low(IN_TABLE << 1)      ;load the first value in the table into the Z register
        ldi ZH, high(IN_TABLE << 1)

                                        ;To read program memory we must multiply the table address by 2
                                        ;0xABCD * 2 = 0x1579A
                                        ;ZL = 9A
                                        ;ZH = 57
                                        ;we still need the most significant 1
        ldi r20, 0x01                   ;so we load it into the RAMPZ register
        out CPU_RAMPZ, r20              ;we will need to extend load to use the RAMPZ register

        ldi YL, low(OUT_TABLE_START_ADDR)
        ldi YH, high(OUT_TABLE_START_ADDR)

        ldi r17, Table_Size

        clr r0


        ; loop through input table, performing filtering and storing conditions
LOOP:

        elpm r16, Z         ;load value from input table into an appropriate register
        dec r17
        breq DONE           ;determine if the end of table has been reached (perform general check)

        ; if end of table (EOT) has been reached, i.e., the NULL character was
        ; encountered, the program should branch to the relevant label used to
        ; terminate the program (i.e., LOOP_END)
        rjmp CHECK_1
        ; if EOT was not encountered, perform the first specified
        ; overall conditional check on loaded value (CONDITION_1)
CHECK_1:
        ; check if the CONDITION_1 is met (bit 6 is set); if not, branch to
        ; FAILED_CHECK1
        andi r16, 0b1000000         ;if bit is not set, this will = 0
        breq FAILED_CHECK1          ;if bit is not set, branch to FAILED_CHECK1

                                    ;if it is here, bit 6 is set, load Z back into r16
        elpm r16, Z+
        lsr r16                     ;divide by two
        ldi r18, 55
        cp r16, r18                 ;compare 55
        brlo loop                   ;if lower than 55 branch to loop

        ldi r18, 8
        sub r16, r18                ;otherwise, subtract 8

        st Y+, r16                  ;store in the table

        ; the program should jump back to the beginning of the relevant loop
        rjmp LOOP
```

```
FAILED_CHECK1:
	elpm r16, Z+

	lsl r16				;divide by 2
	ldi r18, 114
	cp r16, r18			;compare 114
	breq EQUAL_OR_LOWER		;there is not <= so check for == then <
	brlo EQUAL_OR_LOWER
					;if here, it was greater than 114 so just loop
	rjmp loop

EQUAL_OR_LOWER:
	ldi r18, 191			;now add that extra 191
	add r16, r18

	st Y+, r16			;store in the table

	; the program should jump back to the beginning of the relevant loop
	rjmp LOOP

; end of program (infinite loop)
DONE:
	rjmp DONE
;*****************************END OF MAIN PROGRAM ****************************
```

APPENDIX

Supporting code above MAIN:

```
;***********************************************************************
;  File name: lab1_skeleton.asm
;  Author:  Christopher Crary
;  Last Modified By: Koby Miller
;  Last Modified On: 25 May 2020
;  Purpose: To filter data stored within a predefined input table based on a
;                   set of given conditions and store a subset of filtered values
;                   into an output table.
;***********************************************************************
;*********************************INCLUDES*************************************
.include "ATxmega128a1udef.inc"
;*****************************END OF INCLUDES*********************************
;*********************************EQUATES**************************************
; potentially useful expressions
.equ NULL = 0
.equ OUT_TABLE_START_ADDR = 0x3700
.equ ThirtySeven = 3*7 + 37/3 - (3-7)  ; 21 + 12 + 4
;*****************************END OF EQUATES**********************************
;***************************MEMORY CONFIGURATION******************************
; program memory constants (if necessary)
.cseg
.org 0xABCD
IN_TABLE:
.db 99, 0x3B, 0164 ,0b00111111 ,0x44 ,0x55 ,'p' ,0112 ,'t' ,0b00111010 ,'<' ,0x39 ,0x57 ,49 ,100 ,NULL

.equ Table_size = 16; label used to calculate size of input table
IN_TABLE_END:
.db 16

; data memory allocation (if necessary)
.dseg
.org OUT_TABLE_START_ADDR
OUT_TABLE:
.byte (IN_TABLE_END - IN_TABLE)
;*************************END OF MEMORY CONFIGURATION*************************
;*******************************MAIN PROGRAM**********************************
.cseg
; configure the reset vector
;      (ignore meaning of "reset vector" for now)
.org 0x0
       rjmp MAIN

; place main program after interrupt vectors
;      (ignore meaning of "interrupt vectors" for now)
.org 0x100
```

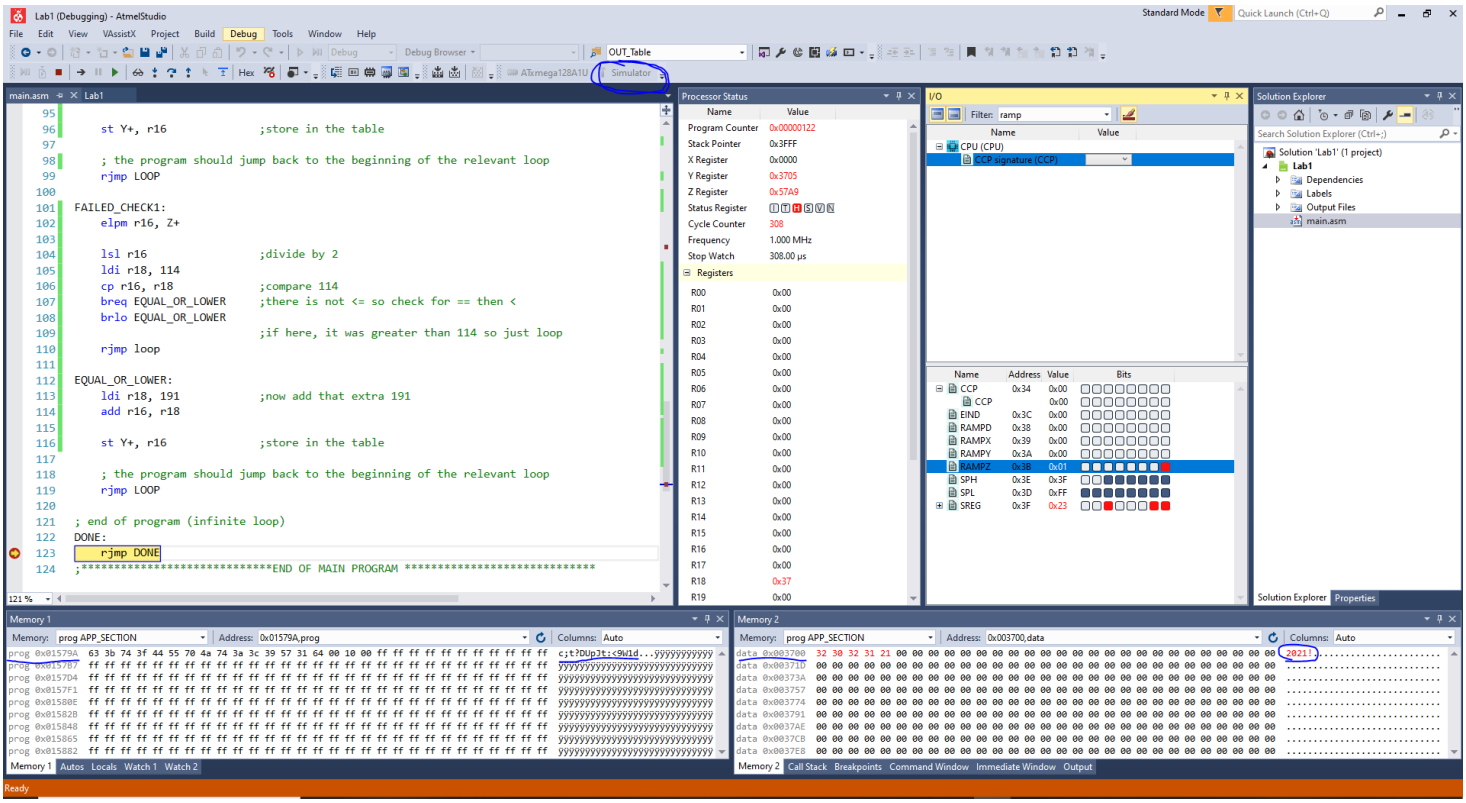

Figure 1 : Simulation Output

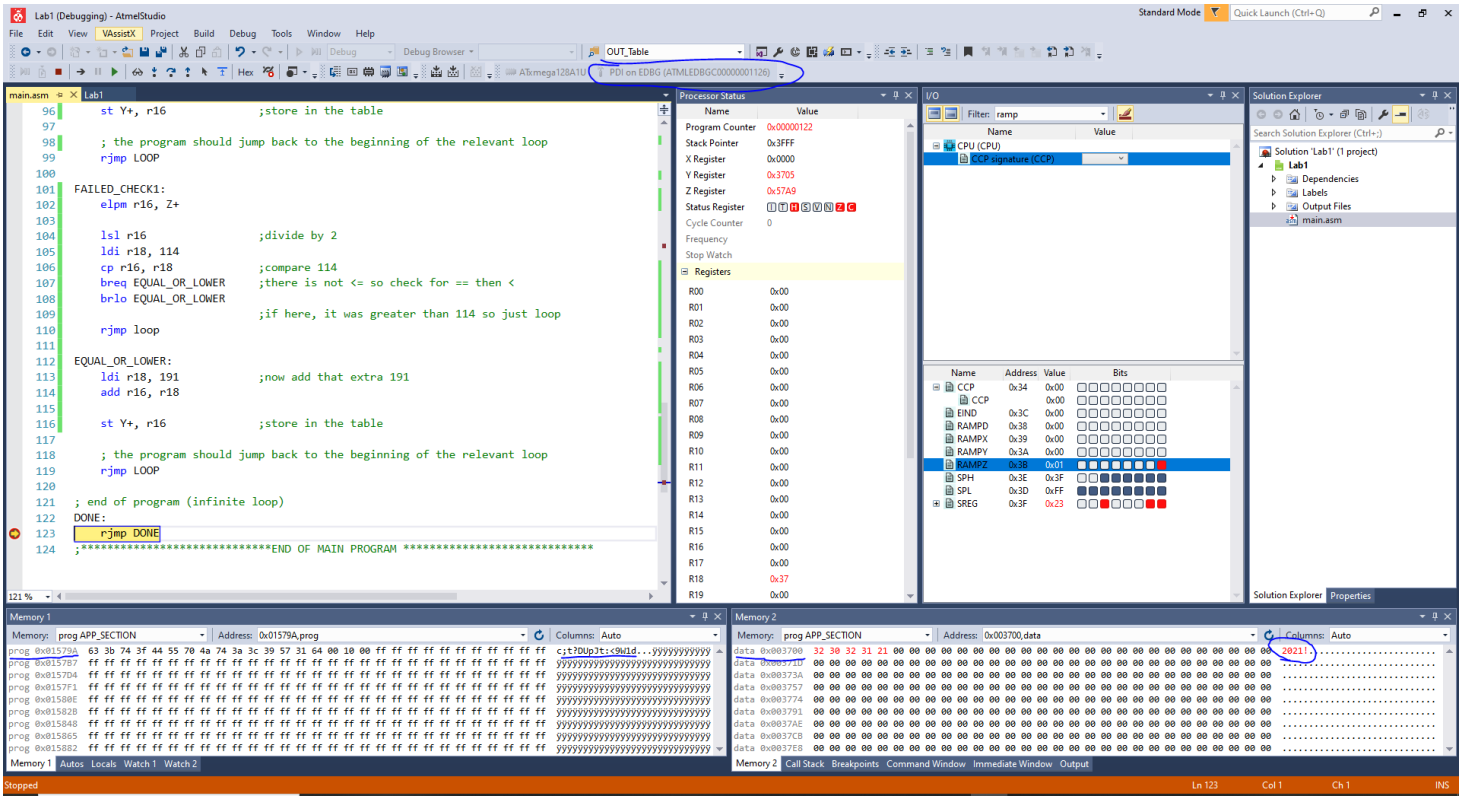

Figure 2 : Emulation Output