REQUIREMENTS NOT MET

N/A

PROBLEMS ENCOUNTERED

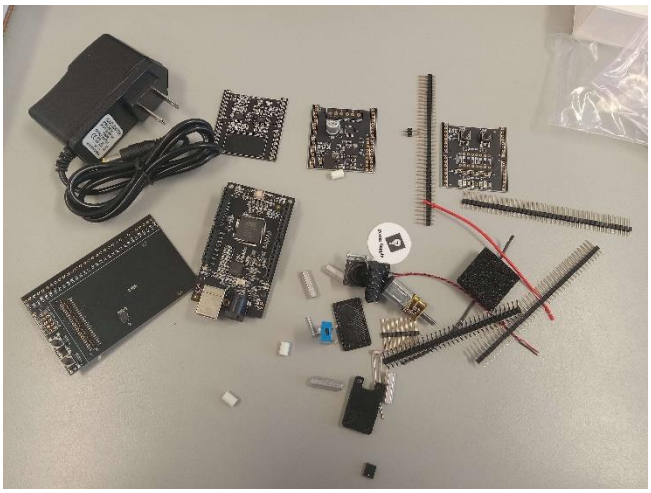
N/A

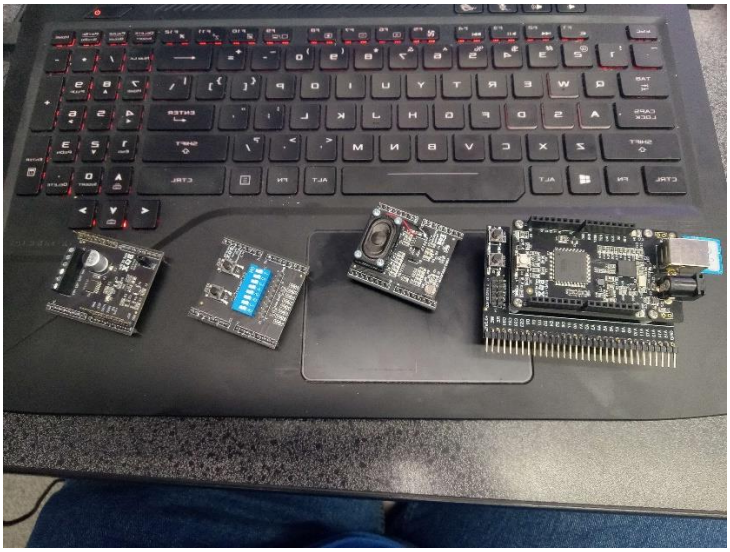
FUTURE WORK/APPLICATIONS

Storing tables to be used later in programs is a very useful feature of some chips. Some programs and chips rely on lookup tables for mathematical operations, such as finding sine or cosine.
In addition, learning how to perform loops may be useful for performing repeat operations, such as implementing software delays, performing multiplication (when its not available on some chips), and implementing timers.

PRE-LAB EXERCISES

i. As specified in Lab 0, you should have, upon receiving your kit, verified that it contained all of the parts listed on µPAD v2.0 Parts List (Excel or PDF). If it did not, you should have immediately notified the PI if any components were missing. For documentation purposes (and before any assembly), you should have taken pictures of all of the parts in your kit (each of the PCBs, the chips, etc.). Include these images in your Lab 1 Pre-Lab Report. Your Lab 1 report should also include images of all of your now completely constructed PCBs.





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 is word-aligned. Data memory is byte-aligned

iii. 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?

".DB" places a byte of data in program memory.
".BYTE" allocates a byte in data memory.
".EQU" allows you to label expressions with a name.
".DEF" allows you to nickname registers

iv. 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?

"LD" can be used with X,Y or Z to load indirectly from data memory.
"ST" can be used with X,Y or Z to store indirectly from data memory.

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

"LDS" can be used to load from I/O memory

"SDS" can be used to store directly to I/O memory

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

"LPM" and "ELPM" can be used to load from program memory.
Only the Z pointer can be used for program memory

viii. In which section of program memory is address 0xF123 located?

Application table

ix. 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?

Since program memory is stored word wise, we multiply oxbadd by 2 to get byte address 0x175ba

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

0x2000-0x3fff, this is because 0x4000 is reserved for external memory access via the EBI.

xi. Which is the first (i.e., lowest) program memory address (this is an address to the 16-bit wide program memory information) that would require the relevant RAMP register to be changed from its initial value of zero? Why?

Accessing byte memory above 11111111 11111111 (0xffff) requires the ramp z. This means that program memory can only access word address 0x8000 and above using ramp registers.

xii. 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 must first be multiplied by two in order to get the byte address in program memory. Since data memory is byte indexed, this isn't necessary.

PSEUDOCODE/FLOWCHARTS

SECTION 1

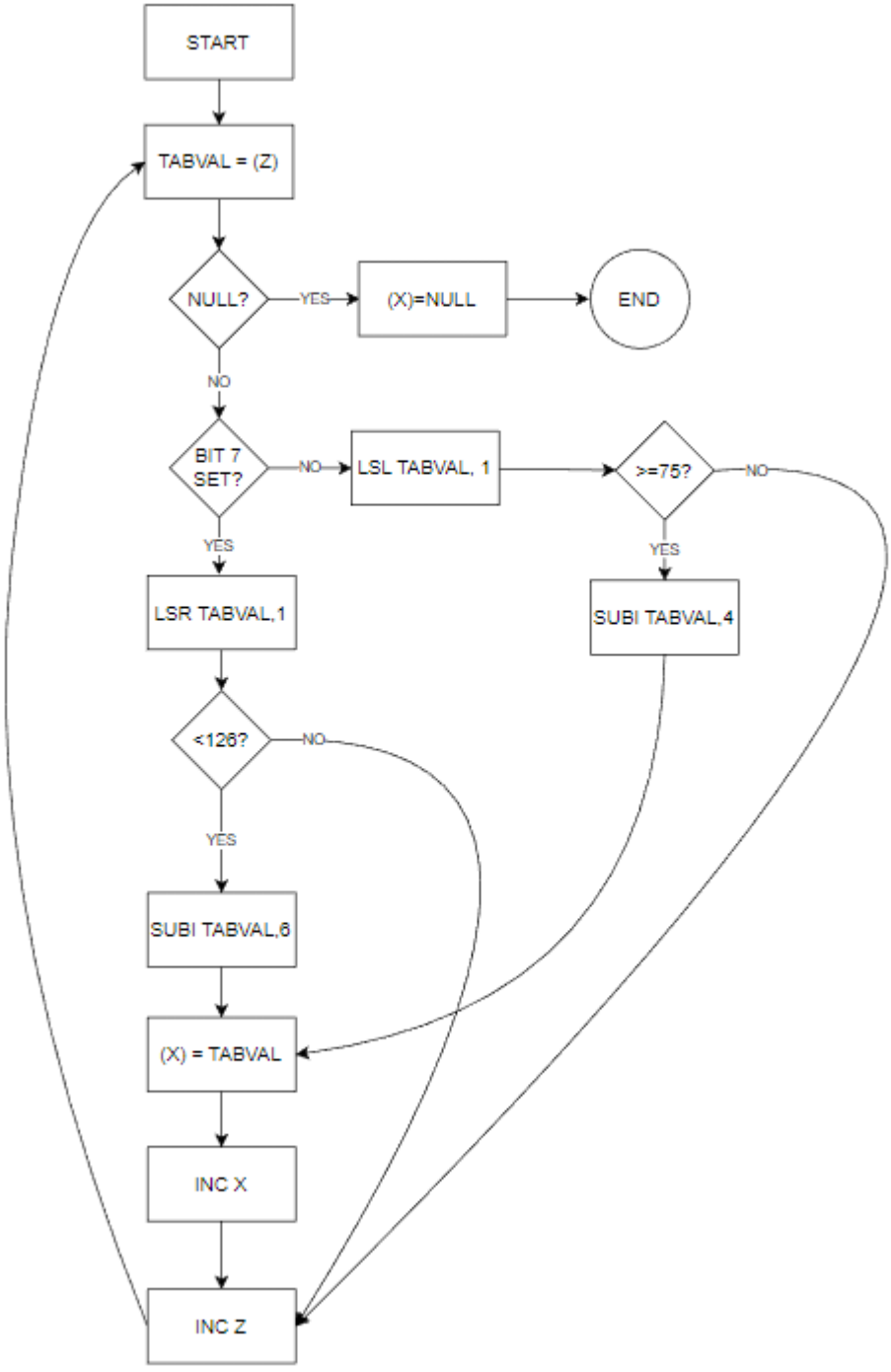


Figure 1: Flowchart for lab1.asm.

PROGRAM CODE

SECTION 1

```
;*********************************************************************
;Lab 1, Section 1
;Name: Steven Miller
;Class #: 11318
;PI Name: Anthony Stross
;Description: Iterates through an input table located in program memory,
;filters it, and stores it in data memory
;*********************************************************************

;*********************************INCLUDES*****************************
.include "ATxmega128a1udef.inc"
;***********END OF INCLUDES*****************************

;*********************************EQUATES*****************************
; potentially useful expressions
.equ NULL = 0
.equ ThirtySeven = 3*7 + 37/3 - (3-7)  ; 21 + 12 + 4
;***********END OF EQUATES*****************************

;*********************************DEFS*****************************
.def tabval = r16
;***********END OF DEFS*****************************

;***********MEMORY CONFIGURATION*************************
; program memory constants (if necessary)
.cseg
.org 0xf123 ;keep in mind this is a word
IN_TABLE:
.db 178, 0xd2, '#',041,'6',043,0x24,0b11111010,0xce,073,'<',0b00111111,0x00,0b00100100

; label below is used to calculate size of input table
IN_TABLE_END:

; data memory allocation (if necessary)
.dseg
; initialize the output table starting address
.org 0x3737
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 0x0000
        rjmp MAIN

; place main program after interrupt vectors
;       (ignore meaning of "interrupt vectors" for now)
.org 0x0100
MAIN:
; point appropriate indices to input/output tables (is RAMP needed?)

;initialize Z pointer using ramp register
ldi ZH, byte3(in_table<<1)
out CPU_RAMPZ, ZH
ldi Zl, byte1(in_table<<1)
ldi ZH, byte2(in_table<<1)

;initialize x pointer
ldi XH, high(out_table)
ldi XL, low(out_table)

;were golden
```

```
; loop through input table, performing filtering and storing conditions
LOOP:
        ; load value from input table into an appropriate register
        elpm tabval, z+
        ; determine if the end of table has been reached (perform general check)
        cpi tabval, null
        ; 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 (e.g., DONE)
        st x, tabval
        breq done
        ; 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 7 of # is set);
        sbrs tabval, 7
        ;   if not, branch to FAILED_CHECK1
        rjmp failed_check1
        ; since the CONDITION_1 is met, perform the specified operation
        ;   (divide # by 2)
        lsr tabval
        ; check if CONDITION_1a is met (result < 126); if so, then
        ;   jump to LESS_THAN_126; else store nothing and go back to LOOP
        cpi tabval, 126 ;tabval - 126
        ;if negative bit clear, then tabval is greater
        ;if negative bit set, then tabval is lesser
        brlo less_than_126
        rjmp LOOP

LESS_THAN_126:
        ; subtract 6 and store the result
        subi tabval, 6
        st x+, tabval
        rjmp LOOP

FAILED_CHECK1:
        ; since the CONDITION_1 is NOT met (bit 7 of # is not set,
        ;    i.e., clear), perform the second specified operation
        ;    (multiply by 2 [unsigned])
        lsl tabval
        ; check if CONDITION_2b is met (result >= 75); if so, jump to
        ;    GREATER_EQUAL_75 (and do the next specified operation);
        ;    else store nothing and go back to LOOP
        cpi tabval, 75; tabval-75
        ;if n flag is clear, then tabval is greater
        ;if n flag is set, then tabval is lesser
        brsh GREATER_EQUAL_75
        rjmp LOOP

GREATER_EQUAL_75:
        ; subtract 4 and store the result
        subi tabval, 4
        st x+, tabval
        ;go back to LOOP
        rjmp LOOP

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

APPENDIX

SECTION 1

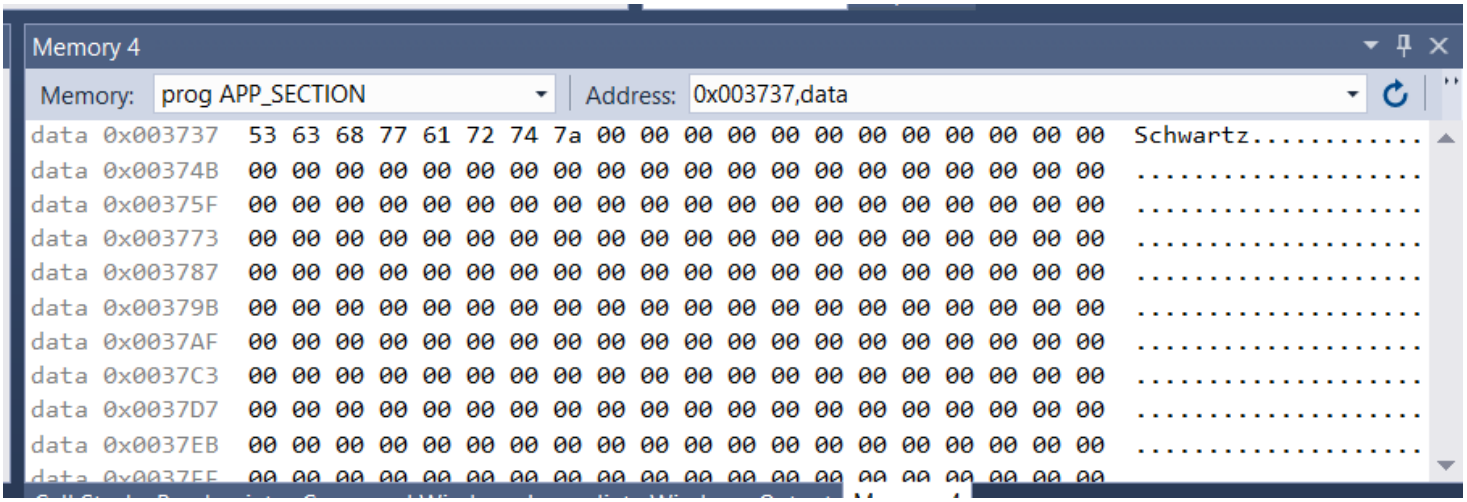


Figure 2: Memory window after running lab1.asm. The entries from the program memory have been filtered and stored in data memory.