Lab 1
Pengzhao Zhu
Section: 112D

B) Prelab Questions

1. What are the key differences between program and data memory? See section 7 in the ATxmega128A1U manual.
    - **Executable Code can reside only in the program memory, while data can be stored in the program memory and data memory.**
    - **Program memory is in terms of “word (16 bits)”, while data memory is in terms of “byte (8 bits)”.**
    - **One example of data memory is SRAM, one example of program memory is flash.**

2. What instruction(s) can be used to read from program memory (flash)? Can you use any registers with this instruction?
   I used the ‘ldi’ instruction to point the Z register to get access to program memory. No, you cannot ‘ldi’ for any registers. You can only use it for r16-r31, and you cannot use it for r0-r15. I also used ‘elpm’ to load value from the concatenated Z register to a general purpose register. The ‘elpm’ instruction can only be used with concatenated pointer registers (X,Y,Z). The Z register (R30-R31) can be used as a pointer to point to program memory.

3. When using RAM (not EEPROM), what memory locations can be utilized for the .dseg? Why? What .dseg did you use in this lab and why?

   For internal SRAM, memory location 2000 to 3FFF can be utilized for the .dseg. The reason for this memory range is that our board is set up this way (see Figure1 below). For this lab, I used .dseg to set up a place (.byte) where I can refer to in code segment in order to write to data memory. I used address 0x3744 for the start of .dseg because the lab instruction asked me to do so.

Byte Address	ATxmega128A1U
0	I/O Registers (4KB)
FFF	
1000	EEPROM (2K)
17FF	
	RESERVED
2000	Internal SRAM (8K)
3FFF	
3000	External Memory (0 to 16MB)
FFFFFF	

Figure 1

C) Problems Encountered

Initially, I did not know how to grab one byte at a time from program memory (which is in terms of 16 bit "word"). It took me a couple of office hours/lectures to figure out that since our processor thinks in terms of byte, I had to double the address to grab the correct data out of the word address.

Another program that I ran into was that I could not write data to the data memory address I wanted. I fixed it by declaring the variable in data segment (use .dseg) and refer back to the variable in code segment.

D) Future Work/Application

I can take the concepts I learned in this lab and apply it to different microprocessors. The only difference would be the difference in instructions set. By learning these simple concepts (how to use instructions and how the processor reads input), I can also use it to write more complicated programs in future lab/work.

E) Schematics

No hardware was modified on this lab.

F) Pseudocode/Flowcharts

*The Pseudocode uses assembly branching format

Initialize starting address at 0x0000
Put table values at program memory 0xC000

Reserve a byte in data memory that starts a 0x3744

Start:
Y pointer point to data memory at 0x3744
Concatenated Z pointer to look at first table value (0x18000)

TOP:
Store the value at the address pointed by Z to register 16. Post Increment Z pointer
Load decimal 0 into register 20. Compare register 20 and 16 to see if they are equal.
If they are equal, we go down to **BOTTOM** and prepare to stop the program

If they are not equal, load bit 6 of register 16 into the T-Flag.
Use the T-Flag to check if the bit is set.
If set, branch to **BITSET**.

Use the T-Flag to check if the bit is clear.
If clear, branch to **TCLEAR.**

BITSET:
Compare value in register 16 to 0x79
If value in register 16 is less than 0x79, branch to **SUBTRACT**
Otherwise, back to **TOP**.

SUBTRACT:
subtract 3 from value in r16.
Jump to **STORE**

TCLEAR:
Check if value in r16 is less than 37.
If it is, go to **STORE**.
If not, back to **TOP**.

STORE:
Store value in r16 to address pointed by Y pointer. Post increment Y pointer.
Back to **TOP**

BOTTOM:
Store value at register 16 (which is 0) to the address pointed by Y.
Finally, infinite loop to end the program.

G) Program Code

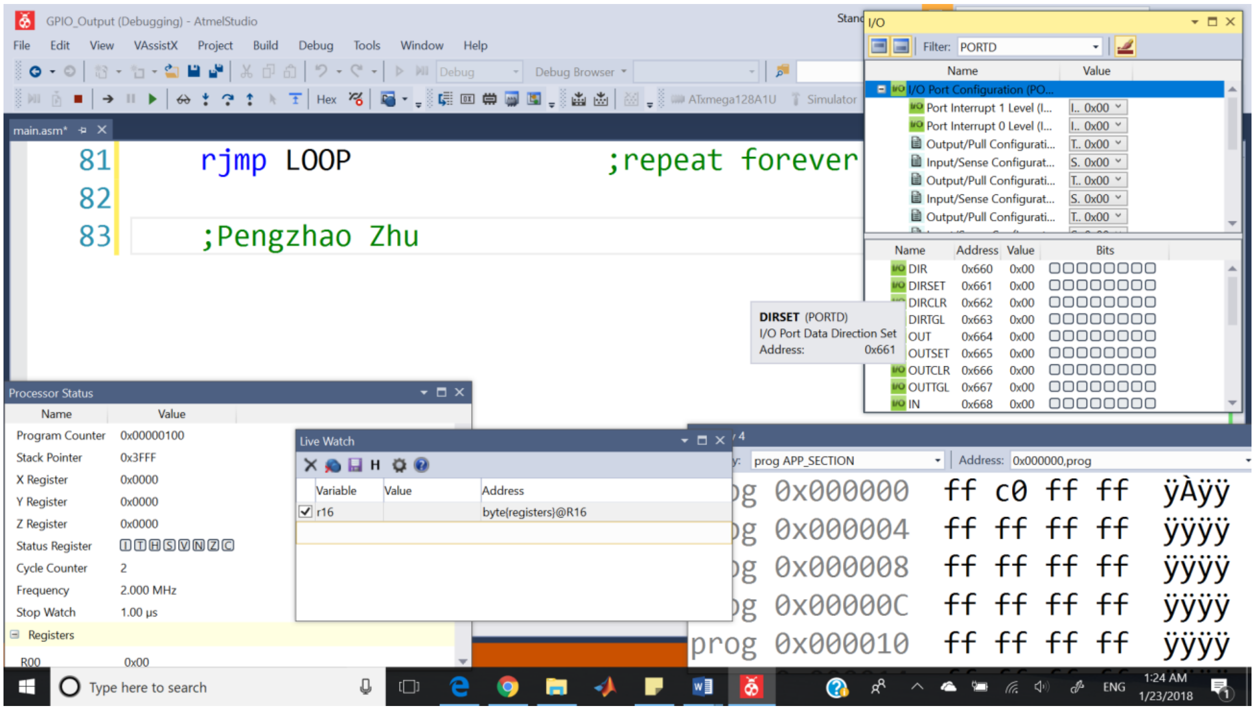
```
/* Lab 1 Part C
   Name: Pengzhao Zhu
   Section#: 112D
   TA Name: Chris Crary
   Description: This program filters a table in program memory according the several
conditions and put the filtered
                data into data memory
*/

.include "ATxmega128A1Udef.inc"  ;according the Schwartz. The new board/program doesn't
need this anymore
.list

.org 0x0000   ;start at address 0x000
       rjmp MAIN ;jump to main code

.org 0xC000  ;location to put the table

Table: .db 0b01111000, 0x79, 0123, 0b00100000, 108, '7', 'v', 0b01111110, 040, 0x69,
'9' ,0x78, 0b01110001, 122, 0 ; initialize table

.dseg  ;changing to data memory because this is the place where we will put our filtered
table
.org  0x3744   ;where we will start putting the table

filtable: .byte 15   ;reserve a byte. will use pointer to increment this table.

.cseg    ;node the code segment
.org 0x200  ;a place where we will put our executable code
MAIN:

ldi YL, low(filtable)  ;Y pointer point to where the table will start
ldi YH, high(filtable) ;low and high bytes

;look at the byte 0x18000
ldi ZL, byte3(Table<<1) ;look at 0x01.  load byte3 into ZL
out CPU_RAMPZ, ZL       ;put address pointed by ZL into CPU_RAMPZ
ldi ZL, low(Table << 1)        ; look at 0x00
ldi ZH, high(Table << 1)       ; look at 0x80

THIRD:
elpm r16, Z+                ;store the value at concavated RAMPZ + Z pointer to r16, then
increment z point

ldi r20, 0  ;load r10 with 0. The null character is 0
cp r16, r20 ;check if r16 is 0
breq DOWN  ; branch to DOWN if equal. In other word, that is the null character, we need
to stop

bst r16, 6 ; store bit 6 of r16 into the T flag
brts BITSET  ; branch if the bit is set
brtc TCLEAR  ; branch if the bit is clear

BITSET:
```

```
 cpi r16, 0x79 ;compare value in r16 with 0x79
 brlo SUBTRACT  ;branch if value in r16 is less than 0x79
 rjmp THIRD     ;back on top

 SUBTRACT:
 subi r16,3   ;subtract 3 from r16
  rjmp STOR   ;jump

TCLEAR:
cpi r16, 37          ; check is r16 is less than 37. if it is, go to store
brlo STOR    ;branch to STOR
rjmp THIRD   ;back to THIRD


  STOR:
  st Y+, r16   ;store value in r16 to address pointed by Y pointer. post increment Y
pointer
  rjmp THIRD   ;back on top

  DOWN:
  st Y, r16 ;store the value in r16 to the place pointed by Y pointer
  DONE:
      rjmp DONE    ;infinite loop to end the program
```

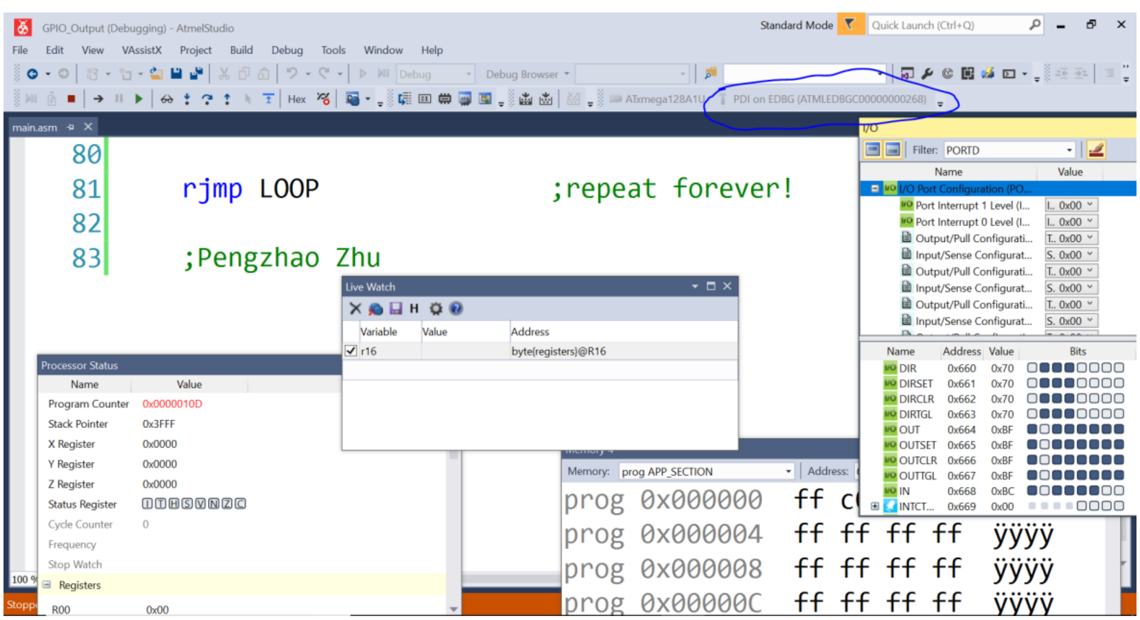
H) Appendix

Part A- N/A

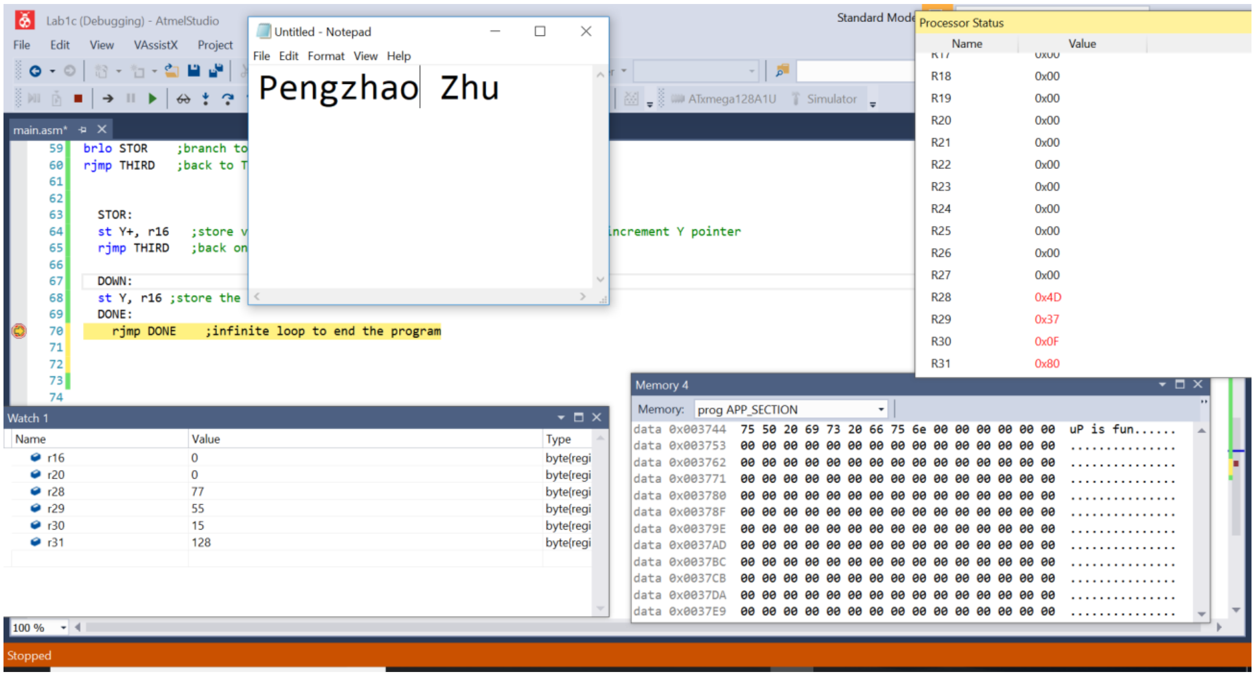
Appendix A: Lab 1, Part B Simulation



Appendix B: Lab 1, Part B Emulation



Appendix C: Lab1, Part C Simulation



Appendix D: Lab1, Part C Emulation