Lab 1
Jineidy Mak
Secton 7F31

Pre-Lab Questions:

1. What registers can we use to read from program memory (flash)? We can use the Z register to read from program memory.

2. What is the difference between program and data memory? See section 7 in the ATxmega128A1U manual. The program memory is where the application and application tables are located. The data memory is where the data is being written to, externally.

3. When using RAM (not EEPROM), what memory location should be utilized for the .DSEG? Why? What .DSEG did you use in this lab and why? The internal SRAM begins at 0x2000 and that's the .ORG address we use use when using .DSEG. .DSEG specifies where data is being stored. This is the only way we can store to the SRAM otherwise we would not be able to store any data.

Problems Encountered:

The tutorial document needs to be updated.

Future Work/Applications:

We can use this method to filter all kinds of data in different ways. We can also display the data to an LCD.

Schematics:

Decoding Logic:

Pseudocode/Flowcharts:

```
Table[21] = {6E, 5C, 55, 2B, 68, 53, 6B, 79, 55, 67, 45, 53, 34, 59, 55, 37, 67, 6D, 22, 00};
Filtered_Table[21];
Table_Size = 21;
Table_Value;

For(int j = 0 to Table_Size
{
        For(int i = 0 to Table_Size)
        {
                Table_Value = Table[i];
                if(Table_Value < 6E and Table_Value >= 48)
                {
                        Filtered_Table[j] = Table_Value + 12;
                }


        }

        Filtered_Table[j + 1] = NULL;

}
```

Program Code:

```
;
; lab1.asm
;
; Created: 5/23/2016 5:32:12 PM
; Author : James Mak
;


; Replace with your application code

.nolist
.include "ATxmega128A1Udef.inc"

.equ twelve = 0x0C          ;Constant 12.
.equ null  = 0x00           ;Null character.
.equ test1 = 0x6E                           ;First test variable.
.equ test2 = 0x48                   ;Secon test variable.
.equ table_size = 21                        ;Intialize the table as 21.
.def counter = r16                          ;Counter will be stored in register 16.
.def new_counter = r17                      ;Counter will store the size of the new filtered table in r17.
.org 0x0000                                                 ;This is where our program will start.
            rjmp main                                       ;Jump to start of program.

.org 0x2A18                                                 ;Table location in memory.

table: .db 0x48, 0x6E, 0x5C, 0x55, 0x2B, 0x68, 0x53, 0x6B, 0x79, 0x55, 0x67, 0x45, 0x53, 0x34, 0x59, 0x55, 0x37,
0x67, 0x6D, 0x22, 0x00 ;Define the table, .db means bytes.

.dseg                                               ;Data Segment for variables.

.org 0x3701                                 ;This is where our new table will be located.

new_table: .byte table_size ;We reserve table size that is the same size as our original table for our new filtered
table.

.cseg                                               ;Code Segment.

.org 0x0200



MAIN:               ldi ZL, low(table <<1)                  ;Load the low byte of the table address.
                            ldi ZH, high(table <<1)                 ;Load the high byte of the table address.
                            ldi YL, low(new_table)      ;Load the low byte of the new table location (3701).
```

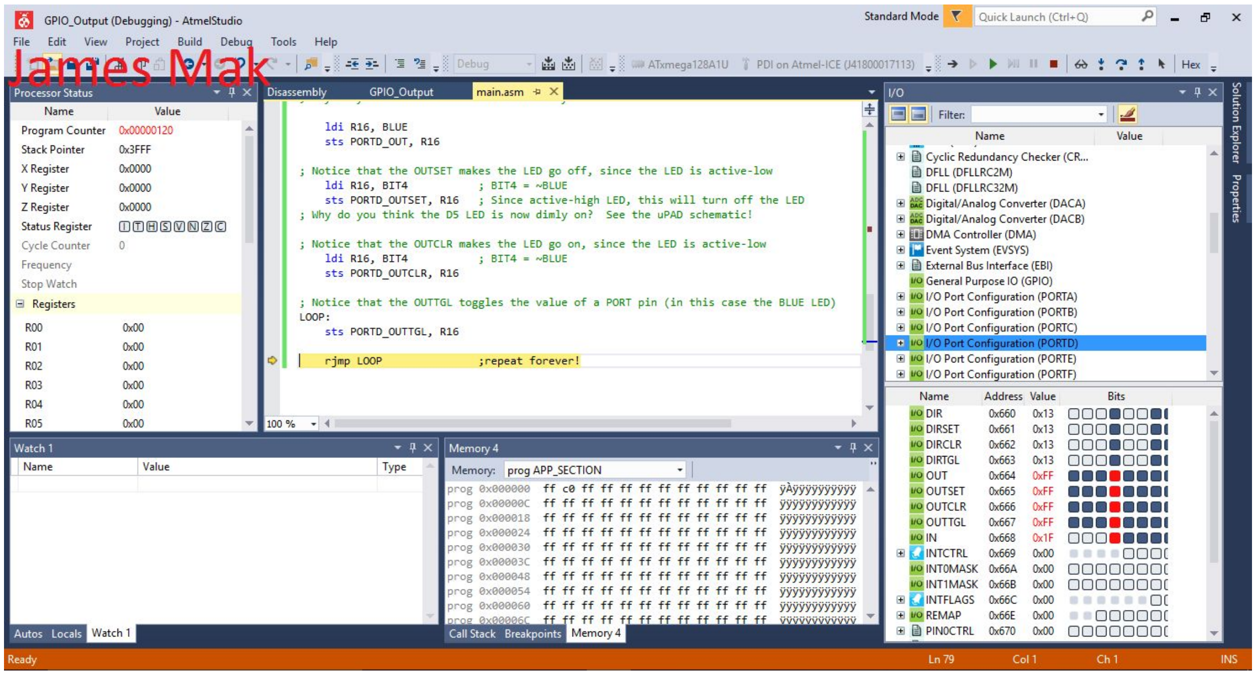
```
                    ldi YH, high(new_table)     ;Load the high byte of the new table location (3701).
                                                                        ;It is noteable that you
cannot use ldi with r0 - r15.

LOOP:          lpm r18, Z+           ;Loading the value at Z into r18 and then incrementing the pointer.
                    cpi r18, null          ;Compare r18 with the NULL character to see if the program should
end.
                    breq END_LOOP          ;Branch to the end of the LOOP if we are done.
                    cpi r18, test1         ;Compare r18 with test1(6E).
                    brsh LOOP              ;If it is not less than test1(6E) we repeat the process.
                    cpi r18, test2         ;Compare r18 with test2(48).
                    brlo LOOP              ;If it is not greaer than test2(48) we repeat the process.
                    ldi r19, twelve        ;Load the decimal value of 12 into register r19.
                    add  r18, r19                    ;Add the decimal value of 12 to the number.
                    st Y+, r18             ;Store the value where Y is pointed to.
                    inc new_counter        ;Add one to the new_counter since we tested the value is less
than 6E.
                    rjmp LOOP


END_LOOP:     inc new_counter                  ;Account for the null character.
                    st Y+, r18             ;Store the null character.
DONE:
                    rjmp DONE                             ;Infinite Loop.
```

Appendix:

Part A: Tutorial



James Mak

GPIO_Output (Debugging) - AtmelStudio

Processor Status

Name	Value
Program Counter	0x00000120
Stack Pointer	0x3FFF
X Register	0x0000
Y Register	0x0000
Z Register	0x0000
Status Register	I T H S V N Z C
Cycle Counter	0
Frequency	
Stop Watch	
Registers	
R00	0x00
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00

main.asm

```
    ldi R16, BLUE
    sts PORTD_OUT, R16

; Notice that the OUTSET makes the LED go off, since the LED is active-low
    ldi R16, BIT4            ; BIT4 = ~BLUE
    sts PORTD_OUTSET, R16    ; Since active-high LED, this will turn off the LED
; Why do you think the D5 LED is now dimly on?  See the uPAD schematic!

; Notice that the OUTCLR makes the LED go on, since the LED is active-low
    ldi R16, BIT4            ; BIT4 = ~BLUE
    sts PORTD_OUTCLR, R16

; Notice that the OUTTGL toggles the value of a PORT pin (in this case the BLUE LED)
LOOP:
    sts PORTD_OUTTGL, R16

    rjmp LOOP                ;repeat forever!
```

Memory 4 — Memory: prog APP_SECTION

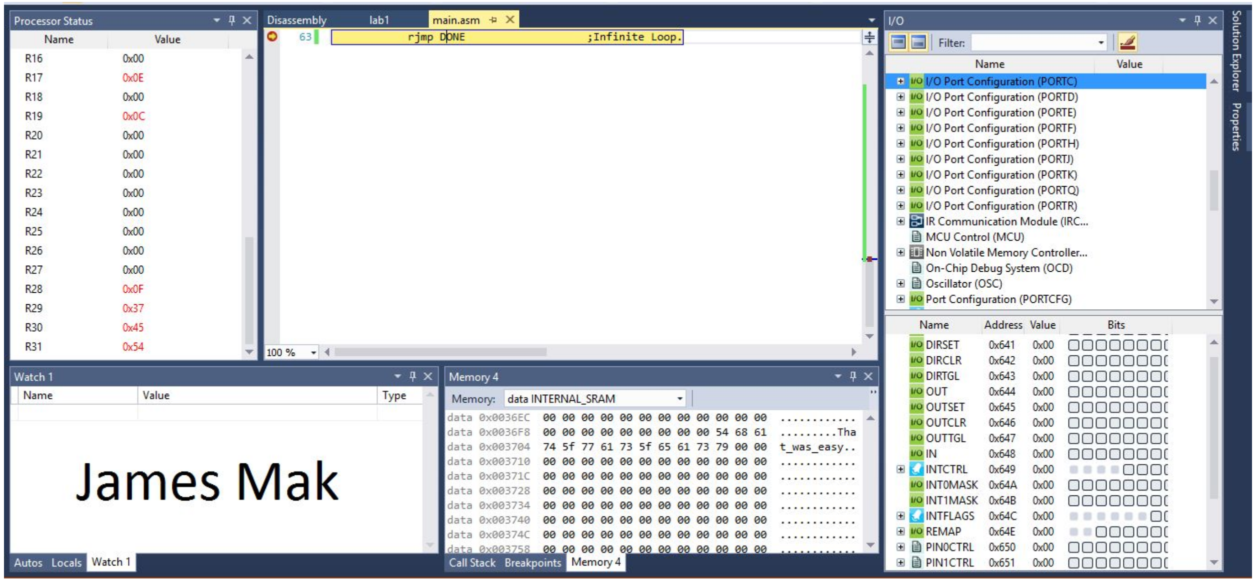
```
prog 0x000000  ff c0 ff ff ff ff ff ff ff ff ff ff  ÿÀÿÿÿÿÿÿÿÿÿÿ
prog 0x00000C  ff ff ff ff ff ff ff ff ff ff ff ff  ÿÿÿÿÿÿÿÿÿÿÿÿ
prog 0x000018  ff ff ff ff ff ff ff ff ff ff ff ff  ÿÿÿÿÿÿÿÿÿÿÿÿ
prog 0x000024  ff ff ff ff ff ff ff ff ff ff ff ff  ÿÿÿÿÿÿÿÿÿÿÿÿ
prog 0x000030  ff ff ff ff ff ff ff ff ff ff ff ff  ÿÿÿÿÿÿÿÿÿÿÿÿ
prog 0x00003C  ff ff ff ff ff ff ff ff ff ff ff ff  ÿÿÿÿÿÿÿÿÿÿÿÿ
prog 0x000048  ff ff ff ff ff ff ff ff ff ff ff ff  ÿÿÿÿÿÿÿÿÿÿÿÿ
prog 0x000054  ff ff ff ff ff ff ff ff ff ff ff ff  ÿÿÿÿÿÿÿÿÿÿÿÿ
prog 0x000060  ff ff ff ff ff ff ff ff ff ff ff ff  ÿÿÿÿÿÿÿÿÿÿÿÿ
prog 0x00006C  ff ff ff ff ff ff ff ff ff ff ff ff  ÿÿÿÿÿÿÿÿÿÿÿÿ
```

I/O — I/O Port Configuration (PORTD)

Name	Address	Value
DIR	0x660	0x13
DIRSET	0x661	0x13
DIRCLR	0x662	0x13
DIRTGL	0x663	0x13
OUT	0x664	0xFF
OUTSET	0x665	0xFF
OUTCLR	0x666	0xFF
OUTTGL	0x667	0xFF
IN	0x668	0x1F
INTCTRL	0x669	0x00
INT0MASK	0x66A	0x00
INT1MASK	0x66B	0x00
INTFLAGS	0x66C	0x00
REMAP	0x66E	0x00
PIN0CTRL	0x670	0x00

Ready — Ln 79 Col 1 Ch 1 INS

Part B: Simulation



Processor Status

Name	Value
R16	0x00
R17	0x0E
R18	0x00
R19	0x0C
R20	0x00
R21	0x00
R22	0x00
R23	0x00
R24	0x00
R25	0x00
R26	0x00
R27	0x00
R28	0x0F
R29	0x37
R30	0x45
R31	0x54

main.asm

```
rjmp DONE           ;Infinite Loop.
```

James Mak

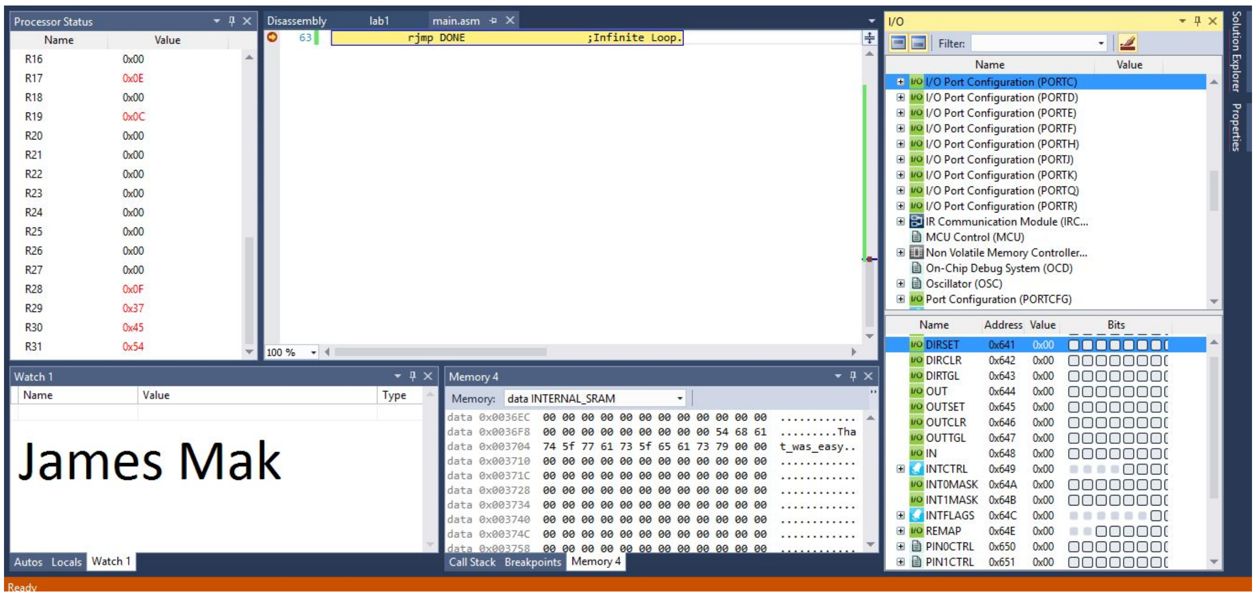
Memory 4 — Memory: data INTERNAL_SRAM

```
data 0x0036EC  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x0036F8  00 00 00 00 00 00 00 00 00 54 68 61  .........Tha
data 0x003704  74 5f 77 61 73 5f 65 61 73 79 00 00  t_was_easy..
data 0x003710  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x00371C  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x003728  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x003734  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x003740  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x00374C  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x003758  00 00 00 00 00 00 00 00 00 00 00 00  ............
```

I/O

Name	Address	Value
DIRSET	0x641	0x00
DIRCLR	0x642	0x00
DIRTGL	0x643	0x00
OUT	0x644	0x00
OUTSET	0x645	0x00
OUTCLR	0x646	0x00
OUTTGL	0x647	0x00
IN	0x648	0x00
INTCTRL	0x649	0x00
INT0MASK	0x64A	0x00
INT1MASK	0x64B	0x00
INTFLAGS	0x64C	0x00
REMAP	0x64E	0x00
PIN0CTRL	0x650	0x00
PIN1CTRL	0x651	0x00

Part C: Emulation



Processor Status

Name	Value
R16	0x00
R17	0x0E
R18	0x00
R19	0x0C
R20	0x00
R21	0x00
R22	0x00
R23	0x00
R24	0x00
R25	0x00
R26	0x00
R27	0x00
R28	0x0F
R29	0x37
R30	0x45
R31	0x54

Disassembly | lab1 | main.asm

```
rjmp DONE              ;Infinite Loop.
```

Watch 1

Name	Value	Type

James Mak

Memory 4

Memory: data INTERNAL_SRAM

```
data 0x0036EC  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x0036F8  00 00 00 00 00 00 00 00 00 54 68 61  .........Tha
data 0x003704  74 5f 77 61 73 5f 65 61 73 79 00 00  t_was_easy..
data 0x003710  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x00371C  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x003728  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x003734  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x003740  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x00374C  00 00 00 00 00 00 00 00 00 00 00 00  ............
data 0x003758  00 00 00 00 00 00 00 00 00 00 00 00  ............
```

I/O

Name	Value
I/O Port Configuration (PORTC)	
I/O Port Configuration (PORTD)	
I/O Port Configuration (PORTE)	
I/O Port Configuration (PORTF)	
I/O Port Configuration (PORTH)	
I/O Port Configuration (PORTJ)	
I/O Port Configuration (PORTK)	
I/O Port Configuration (PORTQ)	
I/O Port Configuration (PORTR)	
IR Communication Module (IRC...	
MCU Control (MCU)	
Non Volatile Memory Controller...	
On-Chip Debug System (OCD)	
Oscillator (OSC)	
Port Configuration (PORTCFG)	

Name	Address	Value
DIRSET	0x641	0x00
DIRCLR	0x642	0x00
DIRTGL	0x643	0x00
OUT	0x644	0x00
OUTSET	0x645	0x00
OUTCLR	0x646	0x00
OUTTGL	0x647	0x00
IN	0x648	0x00
INTCTRL	0x649	0x00
INT0MASK	0x64A	0x00
INT1MASK	0x64B	0x00
INTFLAGS	0x64C	0x00
REMAP	0x64E	0x00
PIN0CTRL	0x650	0x00
PIN1CTRL	0x651	0x00