OBJECTIVES

In this lab, you will write your first program for the Atmel processor. You will gain practice in programming the processor, understanding how to simulate your program **before** programming the board, and how to debug/emulate your program **after** programming the board.

REQUIRED MATERIALS

- As stated in the *Lab Rules and Policies*, submit your entire pre-lab report and **YOUR** asm file(s) through Canvas **BEFORE** entering the lab (as specified)..
- Read/save the following documents:
  - *Atmel Studio Installation Instructions*
  - *Create Simulate Emulate on Atmel* tutorial
  - *AVR instruction set (doc 0856)*
  - *Assembly Language Conversion: GCPU to Atmel Assembly*
- Toolbox including: uPAD kit and DAD/NAD

PRELAB PROCEDURE

You **must** make a **flowchart** or write pseudo-code **before** writing **any** program in this course. This will help you formulate a plan of attack for the code. The flowchart or pseudocode for part C should be included (as stated in the *Lab Rules and Policies*, part 16f), in your pre-lab report. Your assembly language program must also be included in the report (see part 16 g) and also submitted as a separate file (see part18a, item 2).

Note: Pre-lab requirements **MUST** be accomplished **PRIOR** to coming to your lab.

PART A - ATMEL Studio Installation

Go through the *Atmel Studio 7.0 Installation Instructions* to install the necessary software on your laptop (or tablet) computer.

PART B - ATMEL Studio Tutorial

1. Go through the *Create, Simulate, and Emulate a Project* tutorial found on the website. Obtain a screen shot on your laptop of the results of step 11 (the simulation) that **also shows your name** in big letters on the same screen. To do a screen shot in Windows, press Ctrl-PrtScrn (i.e., select Ctrl and PrtScrn at the same time). **Note:** The built-in *Snipping Tool* program in Windows is another great feature to print your screen. Copy your screen shot into MS Word (or a similar program) and include this in the Appendix (see section 16h) of your pre-lab report that you will submit to Canvas.
2. Go through steps 12-14 (the emulation), and again obtain a screen shot that **shows your name** in big letters on the same screen. Screenshots in **ALL LABS** should be put in the **Appendix** of your pre-lab reports.

PART C – Write, Debug/Simulate a Project

In this part, you will write an **assembly language program** (`lab1.asm`) that filters data from a given table that you will place in program memory. (It is required that you make a flowchart or write pseudo-code **before** writing **any** program in this course. This will help you formulate a plan of attack for the code.) The data is given in decimal, hexadecimal, binary, octal, or ASCII. This is to demonstrate that your assembler can read and interpret all these given formats. You do NOT need to convert these values to hex. For each byte in the data table, if the byte matches the filter condition, write the filtered data to the specified memory locations. Include this program in your pre-lab report (as stated in the *Lab Rules and Policies*, part 16 g). Your program will assume that the data is already placed in **program memory** prior to execution, i.e., you will use assembler directives like ".db" to put the constant table values into memory. See the following webpage for more information on using assembler directives:
http://www.avr-tutorials.com/assembly/avr-assembler-directives

Also see Section 4.5 of the below document:
http://mil.ufl.edu/3744/docs/XMEGA/doc1022_Assembler_Directives.pdf

The input table is shown in Table 1; you **MUST** place this table in **program memory** starting at **word address 0xC000** (using assembler directives). Make sure you understand key differences between program memory and data memory. Do our processor's instructions reference memory locations in terms of words, or bytes? **Note:** ASCII is an 8-bit coded version of numbers, letters and symbols. An ASCII table can be found at www.asciitable.com.

Your program should filter the data given in the Table 1 based on the given criteria, and potentially store each of the resulting values in successive **data memory** locations, starting at address **0x3744**. You must use pointers to accomplish this task. You should assume that the table values are **unsigned**.

Table 1: Memory Table

Data	Data (ASCII)[1]
0b01111000	x
0x79	y
0123	S
0b00100000	space
108	l
'7'	7
'v'	v
0b01111110	~
040	space
0x69	i
'9'	9
0x78	x
0b01110001	q
122	z
0	NULL

1. If bit 6 is set AND the value is LESS THAN 0x79, subtract 3 and then add the result to the output table.
2. If bit 6 is cleared, then check if the value is less than 37. If it is, store it directly to the output table.
3. If both conditions are false, do NOT add the value to the output table, i.e., skip it.
4. Your filtering should end after your program detects the NULL character (0x00).

The data in the original table should **not** be corrupted. The new table should end by adding the NULL character (0x00).

In order to make your code modular and re-locatable, use assembler directives. This will make it easy to change the location of both your input and output tables, the filter values, and the end of table (EOT) value. Use the `.byte` assembler directive within a data segment (after a `.dseg` assembler directive) to reserve enough space for the output table.

A summary of the steps required for this lab are as follows.

1. Make a flow chart or write pseudo-code for the program that you will create. (It is required that you make a flowchart or write pseudo-code **before** writing **any** program in this course. This will help you formulate a plan of attack for the code.)
2. Create the assembly language program (`lab1.asm`).
3. Test your program using the Atmel Studio Simulator. Verify that the program works as specified. You must also emulate the program to make sure it works on your board.
4. Take a screen shot of a memory view window that shows the filtered values located at the correct addresses in data memory. Also, include any registers you used in the ***watch window*** (see https://mil.ufl.edu/3744/docs/Watch_Atmel_Studio.pdf) A watch window allows you to see the memory locations and registers change when stepping through the code. Make sure the screenshot **displays your name** in big letters on the same screen. Save this screen shot in the Appendix of your pre-lab report (as stated in the *Lab Rules and Policies*, part 16h).

PRELAB QUESTIONS

1. What are the key differences between program and data memory? See section 7 in the ATxmega128A1U manual.
2. What instruction(s) can be used to read from program memory (flash)? Can you use any registers with this instruction?
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?

PRELAB REQUIREMENTS SUMMARY

1. Answer all pre-lab questions.
2. Take the necessary screenshots described in prelab part B.
3. Make a flowchart or write pseudocode before writing your program. This will help you formulate a plan of attack for the code.
4. Create your program for prelab part C (in a file on your computer) using the Atmel Instruction Set (doc8096). Include this program in your pre-lab report (as stated in the *Lab Rules and Policies*, part 16h).
5. Take the necessary screenshots described in prelab part C.

Note: All pre-lab requirements **MUST** be accomplished **PRIOR** to coming to your lab.

IN-LAB PROCEDURE

Quiz:
After introductions, your TA will give your first lab quiz. Prepare by understanding the concepts used in this lab, do your own lab preparation, and read through your processor's entire Instruction Set (doc8096).

Simulation Demo:
Demonstrate (to your TA) that your program from Part C creates the correct table. The TA will ask you to change the data values and/or filtering conditions and then re-simulate your program. Your TA will also ask you to single step through your program and use breakpoints. Be prepared to answer questions about your program and the simulation/debug procedures.

Emulation Demo:
After simulating, you will now run your program on your processor.

Remember that debugging with hardware is called **emulation**. Whenever possible, you should simulate your design first, before emulating; this eliminates any chance of mistaking hardware bugs for software bugs.

Load the program that you created for prelab Part C onto your board and demonstrate this emulation to your TA.

REMINDER OF LAB POLICY

Re-read the *Lab Rules and Policies* so that you are sure of the deliverables that are due **prior** to the start of your lab.