OBJECTIVES

- Gain familiarity with AVR® assembly programming and the use of *Atmel Studio* for creating, simulating, and emulating a program
- Understand basic microcontroller/microprocessor memory organization concepts

INTRODUCTION

In this lab, you will write your first AVR® assembly program. Based on a set of given conditions, your microcontroller will filter values from a predefined input table and store a subset of these filtered values into an output table. Overall, you will begin to leverage the AVR **I**nstruction **S**et **A**rchitecture (**ISA**) for assembly programming, while reinforcing several fundamental microcontroller concepts.

REQUIRED MATERIALS

- *Atmel Studio* Installation Instructions
- Create Simulate Emulate in *Atmel Studio* Tutorial
- AVR Instruction Set (doc0856)
- AVR Assembler Directives
- AVR Assembler User Guide
- *µPAD v2.0* with USB A to B cable
- **D**igilent (or National) **A**nalog **D**iscovery (**DAD**) with *Waveforms* software

SUPPLEMENTAL MATERIALS

- Assembly Language Conversion: GCPU to AVR
- *Atmel Studio* User Guide
- Utilizing *Watch* in *Atmel Studio*

PRE-LAB PROCEDURE

REMINDER OF LAB POLICY

As required, you must re-read the *Lab Rules & Policies* before submitting any pre-lab assignment and before attending any lab.

PRE-LAB EXERCISES

i. Which type of memory alignment is used for program memory in the ATXMEGA128A1U? Byte-alignment, or word-alignment? What about for data memory?
ii. Does the ATXMEGA128A1U reference memory locations in terms of words, or bytes? How about *Atmel Studio* (in the context of the ATXMEGA128A1U)?
iii. When using RAM (not EEPROM), what memory locations can be utilized for the data segment (`.dseg`)? Why?
iv. In which section of program memory is address 0xF0D0 located?
v. 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 constant values with a meaningful name?
vi. Which instructions can be used to read from (flash) program memory? For each instruction, which registers can be used as operands?
vii. When loading data from a given program memory address, does the address need to first be manipulated? If so, how, and why?
viii. List two methods in which the ATXMEGA128A1U accesses data from a memory address.

Below, you will write your first AVR® assembly language program, **`lab1.asm`**. This program will filter data from a predefined input table (see Table 1) based on a given set of conditions, and store a subset of these filtered values into an output table. The input table will consist of the series of 8-bit data provided in the first column of Table 1. Each byte of data should be interpreted as an **unsigned** value. Moreover, the input table should be placed in **program memory, starting at address 0xF0D0**, and the output table should be placed in **data memory, starting at address 0x3000**.

Additionally, it should be noted that the data in Table 1 is given in decimal, hexadecimal, binary, and ASCII formats; this is done to demonstrate that the assembler within *Atmel Studio* can interpret values in each of these given formats. (ASCII is a generally a 7-bit coded version of numbers, letters and symbols; an ASCII table can be found at www.asciitable.com. Am 8th bit is sometimes used to extend the code to other symbols.) In other words, you do **NOT** need to convert the given table values to a specific format, and you are expected to utilize the values exactly as given.

Separately, the second column of Table 1 provides the data in the ASCII format supported by *Atmel Studio*. This is done simply to allow ease of verification when debugging with a *Memory* view window in *Atmel Studio*; this set of values should **NOT** be used for your input table. (*Memory* debug views are available within *Atmel Studio* under *Debug* | *Windows* | *Memory*; for more information on *Memory* views, navigate to *Debugging* | *Memory View* within the *Atmel Studio* User Guide.)

Ultimately, your program, **`lab1.asm`**, must implement the following algorithm: starting at the first address of the input table, for each byte within the input table, if the **e**nd-**o**f-**t**able (**EOT**) value of NULL (0) is not found, filter the value based on the conditions given below (in the same order), without corrupting the original table. If a value is to be stored to the output table, place it within the first available address.

- If both bits 7 and 6 are set, divide the 8-bit value by 2; if that result is greater than 0x60, store it to the first available location within the output table.
- Else, if the byte is less than or equal to **84**, subtract four from the 8-bit value, and then store it to the first available location within the output table.
- Otherwise, do not store the 8-bit value to the output table.

Table 1: Memory Table

Data	Data (ASCII)[1]
0b11101010	ê
0x5E	^
0124	T
0x24	$
0b01011111	_ (underscore)
'B'	B
044	$
0x5D	]
0xC8	È
'P'	P
0b01100000	‘
0134	\
0x24	$
37	%
'W'	W
0x00	NULL

[1]ASCII format supported by *Atmel Studio*

Upon finding the EOT value in the input table, the output table should be terminated with a NULL character.

NOTES:

- If the given input table is successfully filtered, the output table should contain a readable message, when the data is viewed in terms of the ASCII format supported by *Atmel Studio*. Utilize a *Memory* debug window within *Atmel Studio* to view the output table data in this format.
- In order to make your code modular and re-locatable, utilize assembler directives. This will make it easy to change the location of both your input and output tables, the filter values, and the EOT value.
- A *Watch* window, available under *Debug* | *Windows* | *Watch* within *Atmel Studio*, is used to view memory locations while debugging a program; to learn more about *Watch* windows, navigate to *Debugging* | *Memory View* within the *Atmel Studio* User Guide, and additionally, read the Using *Watch* in *Atmel Studio* document located on the course website, under *Software/Docs.*

1. Make a flow chart or write pseudocode for the program that you will create.
2. Create the assembly language program, **`lab1.asm`**, as specified above.

3. Test your program using the *Atmel Studio* software simulator. Utilize debugging tools to verify that the program works as specified.
4. Emulate the program on your *µPAD* to verify that the program also works on your hardware. Utilize the same debugging tools.
5. Take a screenshot of a *Memory* view window, showing the entire output table at the appropriate memory location, as well as any registers that you used in the *Watch* window.

PRE-LAB PROCEDURE SUMMARY

1) Answer all of the pre-lab exercises.
2) Make a flowchart or write pseudocode **before** writing your program.
3) Write the assembly program, **`lab1.asm`**. Verify its correctness.
4) Take the necessary screenshot(s).