Lab03 – Decisions, Decisions, Decisions! Reading Data and Making Choices

Jonathan Gusler
2/1/2019

Overview

This lab involved writing an assembly language program that reads simulated sensor data, copies it to a safe location in memory, performs some comparisons using the copied data, and depending on the comparison results, writes certain values to memory [1].

Background

This lab assumed knowledge of Atmel assembly language and familiarity with the architecture of the ATmega128A microcontroller.

Discussion

Given the starting code segment, seen in Figure 3 in Appendix A, we were to finish the code based off the program decision tree show in Figure 1.

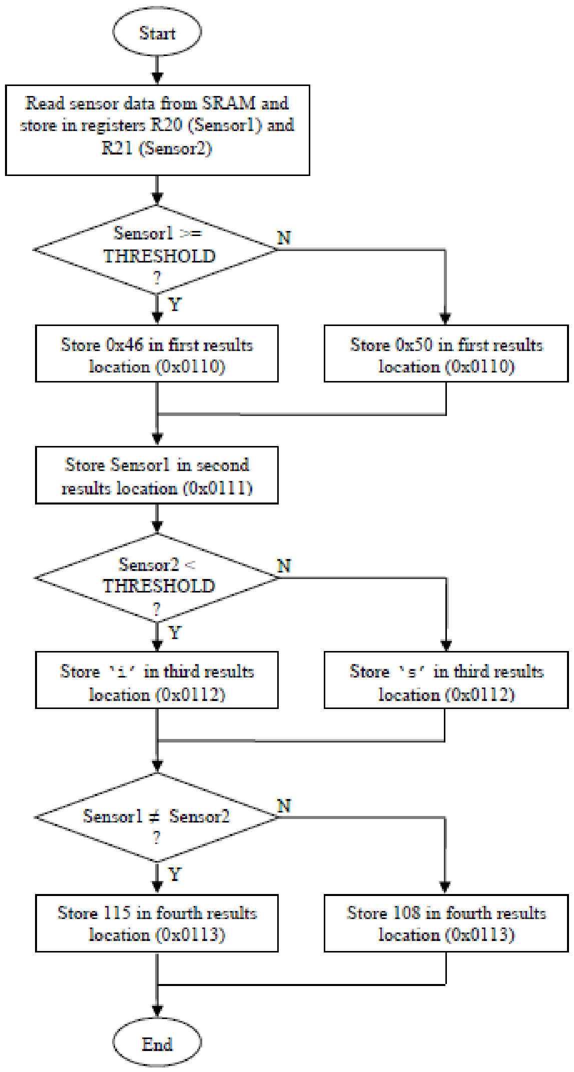

Start
Read sensor data from SRAM and store in registers R20 (Sensor1) and R21 (Sensor2)
Sensor1 >= THRESHOLD ?
N
Y
Store 0x46 in first results location (0x0110)
Store 0x50 in first results location (0x0110)
Store Sensor1 in second results location (0x0111)
Sensor2 < THRESHOLD ?
N
Y
Store 'i' in third results location (0x0112)
Store 's' in third results location (0x0112)
Sensor1 ≠ Sensor2 ?
N
Y
Store 115 in fourth results location (0x0113)
Store 108 in fourth results location (0x0113)
End

Figure 1: Decision Tree of program.

Starting at the end of the code we were given, I wrote the additional code to be added, shown in Figure 4 in Appendix A.

There were some caveats in writing the code [1]:

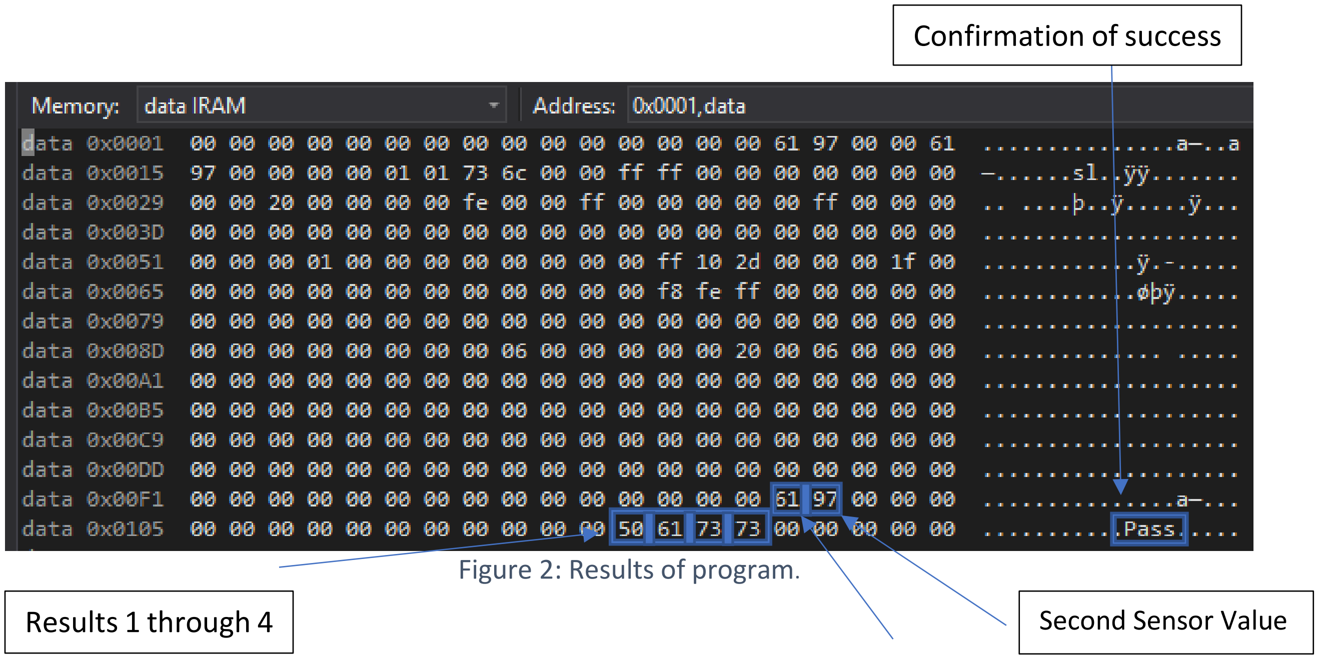
1. THRESHOLD has the value of 0x90 and should be treated as signed if comparing with signed values and unsigned if comparing with unsigned values.
2. Use the 16-bit Y register as a pointer when reading or writing SRAM. The Y register is R28 and R29.
3. Use the symbolic names Sensor1 and Sensor2 for registers R20 and R21.
4. Remember that Sensor1 data is unsigned and Sensor2 data is signed, so you may need different instruction when making comparisons. When comparing Sensor1 to Sensor2, treat both as unsigned.
5. Place a nop instruction as the last line of code in your program. This ensures you can single step through the last program instruction without having the debugger close prematurely.

Question 1: Register R16 could have easily been replaced with R25, but not R26. Explain why not.
Answer: R26 is part of the 16-bit register pair X which is intended to receive the low end of a 16-bit number. The value being loaded into it, 0x61, is not the low end of a 16-bit pair but it would be concatenated with the contents of R27 as if it were. Thus, a wrong number would be used in any calculations.

Question 2: Why is there no plus sign after the "X" in the ST X, R17 instruction.
Answer: The + after the X means to increment the value in X after the instruction. So, we first put something into 0x0100, then increment X so that it points to 0x0101 and we then put something into 0x0101. After that, we don't want to increment the value in X again, so we don't put a +.

Analysis and Results

Figure 2 below shows the final values in memory location 0x0100 and 0x0101 (sensor values), along with the results (locations 0x0110 through 0x0113). To the right of those values is a table showing, positionally corresponding, the ASCII equivalents of the values in the registers.



```
data 0x0001  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 61 97 00 00 61   ...............a—..a
data 0x0015  97 00 00 00 00 01 01 73 6c 00 00 ff ff 00 00 00 00 00 00 00   —......sl..ÿÿ.......
data 0x0029  00 00 20 00 00 00 00 fe 00 00 ff 00 00 00 00 00 ff 00 00 00   .. ....þ..ÿ.....ÿ...
data 0x003D  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ....................
data 0x0051  00 00 00 01 00 00 00 00 00 00 00 00 ff 10 2d 00 00 00 1f 00   ............ÿ.-.....
data 0x0065  00 00 00 00 00 00 00 00 00 00 00 00 f8 fe ff 00 00 00 00 00   ............øþÿ.....
data 0x0079  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ....................
data 0x008D  00 00 00 00 00 00 00 00 06 00 00 00 00 00 20 00 06 00 00 00   .............. .....
data 0x00A1  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ....................
data 0x00B5  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ....................
data 0x00C9  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ....................
data 0x00DD  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ....................
data 0x00F1  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 61 97 00 00 00   ...............a—...
data 0x0105  00 00 00 00 00 00 00 00 00 00 00 50 61 73 73 00 00 00 00 00   ...........Pass.....
```

Figure 2: Results of program.

First Sensor Value

Given the variables shown in the decision tree of the program and the values initialized in the code given to us, the proper results 50, 61, 73, and 73 did show up where they were supposed to. These valued, when translated into ASCII spelled out “Pass” as a confirmation that the right values showed up. In the event that every wrong move was made, the word “Fail” would be the result instead. These outcomes were hinted at in the lab, “did each test properly “pass” or “fail”?” [1].

Summary and Conclusions

This lab was a good introduction to decision making using the Atmel assembly language, introducing the step by step debugging feature in the Atmel studio, and showing Atmel’s feature of being able to see the live contents of the registers, plus the ability to edit that contents.

Appendix A

```
.equ THRESHOLD = 0x90       ; Create a constant  10010000
.def Sensor1 = R20          ; Define a nickname for R20
.def Sensor2 = R21          ; Define a nickname for R21

; next instruction will be written to address 0x0000, the location of
; the reset vector
.org 0x0000

; set reset vector to point to the main code entry point
rjmp main

main:                       ; jump here on reset

; initialize the stack (RAMEND = 0x10FF by default for the ATmega128A)
ldi R16, HIGH(RAMEND)
out SPH, R16
ldi R16, low(RAMEND)
out SPL, R16

;-----------------------------------------------------------------
; Write simulated sensor data into locations 0x0100 and 0x0101
;-----------------------------------------------------------------
; load the simulated sensor values into registers R16 and R17
ldi R16, 0x61               ; simulated sensor 1 value UNSIGNED
ldi R17, 0x97               ; simulated sensor 2 value SIGNED

; initialize 16-bit X-register to point to SRAM location 0x0100
ldi XH, high(0x0100)        ; put high byte of 0x0100 into XH
ldi XL, low(0x0100)         ; put low byte of 0x0100 into XL

; store simulated sensor values at SRAM locations 0x0100 and 0x0101
st X+, R16                  ; (0x0100) <- (R16)
st X,  R17                  ; (0x0101) <- (R17)
```

Figure 3: Code given to us.

```
ldi XH, high(0x0100)       ; put high byte of 0x0100 into XH
ldi XL, low(0x0100)        ; put low byte of 0x0100 into XL

; read sensor data from SRAM and load into registers Sensor1 and
Sensor2
ld Sensor1, X+
; Load data at SRAM location 0x0100 into Sensor1 (Sensor1) -> (X+)
ld Sensor2, X
; Load data at SRAM location 0x0101 into Sensor2 (Sensor2) -> (X)


/*FIRST COMPARISON AND SUBSEQUENT STEP
 load 0x46 and 0x50 into register Y

 if Sensor1 is greater than or equal to THRESHOLD
           store 0x46 in 0x0110, first results location
 else  store 0x50 in 0x0110

  Store Sensor1 in second results location*/
ldi R28, 0x46
ldi R29, 0x50

cpi Sensor1, THRESHOLD
; UNSIGNED OPERATION, branch when Sensor1 >= THRESHOLD, is 61 >= 90?
brsh Sensor1Check
; didn't branch, Sensor1 < THRESHOLD, store 50 in 0x0110
sts 0x0110, R29
rjmp Continue1

; did branch, Sensor1 >= THRESHOLD, store 46 in 0x0110
Sensor1Check:
sts 0x0110, R28

Continue1:
sts 0x0111, Sensor1


/*SECOND COMPARISON
 load 's' and 'i' into register Y
 if Sensor2 is less than THRESHOLD
           store 'i' in third results location
 else store 's' in third results location */
ldi R28, 'i'
ldi R29, 's'
```

```
cpi Sensor2, THRESHOLD
; SIGNED OPERATION, branch when Sensor2 < THRESHOLD, is 97 < 90?
brlt Sensor2Check
; didn't branch, Sensor2 is >= THRESHOLD, 97 >= 90
sts 0x0112,      R29
rjmp Continue2

Sensor2Check:
sts 0x0112, R28            ; 97 <= 90

/*THIRD COMPARISON
 load 115 and 108 into register Y
 if Sensor1 != Sensor2
           store 115 in fourth results location
 else store 108 in fourth results location*/

Continue2:
ldi R28, 115
ldi R29, 108

cp Sensor1, Sensor2        ; treat both as unsigned
brne ThirdCompare          ; branch when not equal
sts 0x0113, R29            ; didnt branch, Sensor1 = Sensor2

ThirdCompare:
sts 0x0113, R28

nop
```

Figure 4: My code, inserted after the code given.

Works Cited

[1] D. G. Nordstrom, "EECE4253Lab03," [Online]. Available: https://drive.google.com/drive/folders/1Av3dayhOQwMNdSrBdB2UJXu8f_4GPd57. [Accessed 5 February 2019].