Lab 0 Summary

Schuster, Mark

Section 1540

08/24/2017

Pre-Lab Answers:

1. The Lab is dropped, but you should complete the lab on your own time.
2. a) Technically yes only if you do not need a second drop. If a second drop is needed documentation will be needed for both the first and second excuse and in that case “Oversleeping” is not a valid excuse.

b) Again, technically yes only if you do not need a second drop. If a second drop is needed documentation will be needed for both the first and second excuse and in that case another class’s project is not a valid excuse. A good rule of thumb is an emergency on my part doesn’t constitute an emergency on your part.

In short, do not miss labs.

1. 65%!
2. 20 minutes late maximum to still be admitted, but only 10 minutes late maximum to be allowed to take the quiz with no extra time!
3. The iron should contact only the wire and the pin and the un-melted solder should also only the wire and the pin. The key is to avoid contacting the iron and the un-melted solder directly.
4. “Executable code can reside only in the program memory, while data can be stored in the program memory and the data memory. The data memory includes the internal SRAM, and EEPROM for nonvolatile data storage.” Basically, program memory is for the application and limited data, and the data memory is for more general data.
5. LPM or “Load Program Memory”. You may only use one register to store the data, and the Z pointer which points to the address in program memory the programmer is trying to access.
6. The internal SRAM address space ranges from 0x2000 to 0x3FFF. This range has this size as the device has 8k of space. We used location 0x3744 in this lab because it is the CRN of this course.

Problems encountered:

Initially had trouble with the getting the program to place the input table at address 0x9000. This was solved after I realized that the address in the assembly was being shifted left by one. Thus, I set the org to 0x4800 and the simulator populated 0x9000 in program memory accordingly.

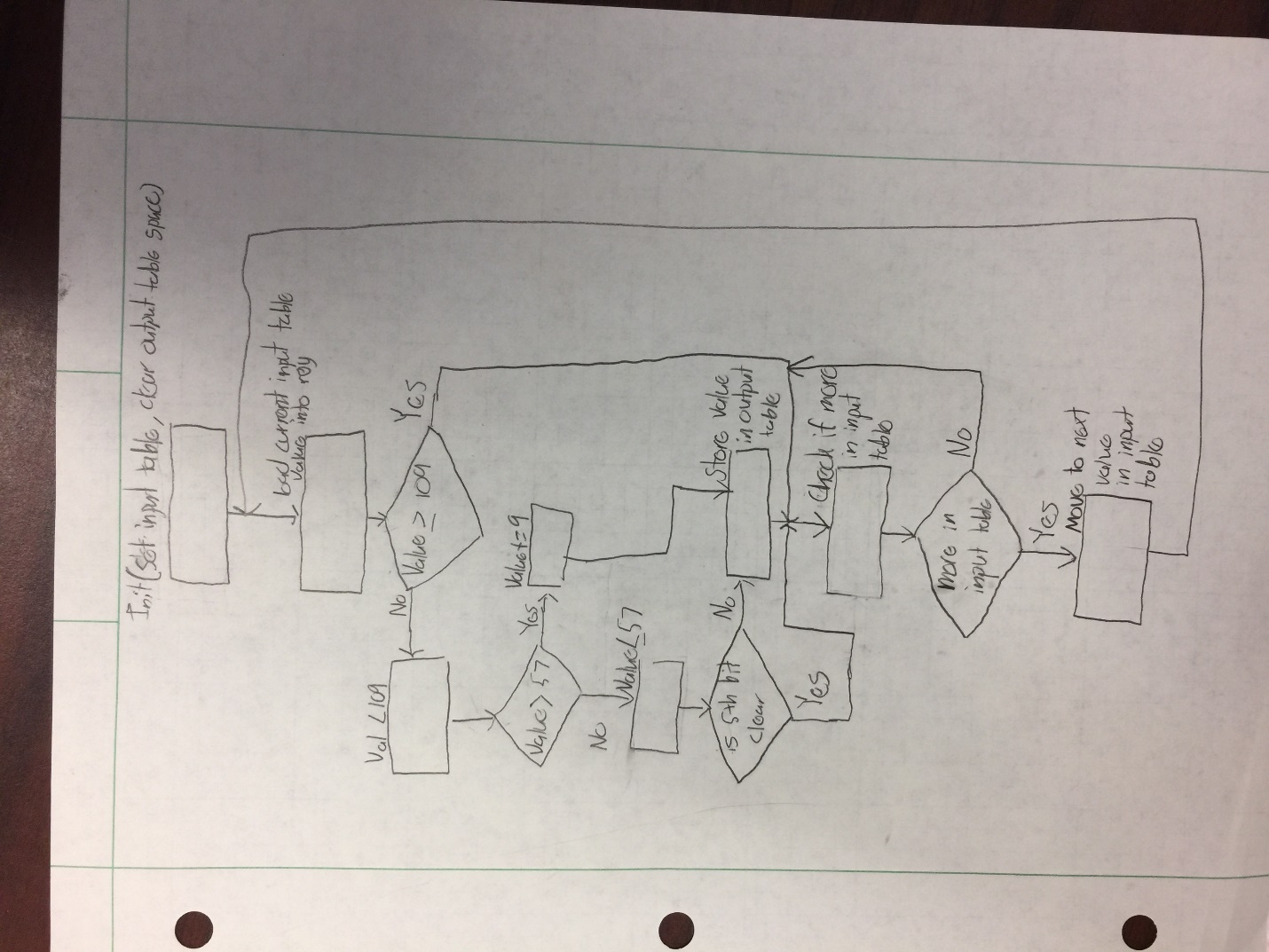
Future Work/Applications:

I have already applied these concepts to TI’s CC2640R2 wireless MCU, but I look forward to developing a more diverse embedded record.

Schematics:

Not applicable for this lab.

Pseudocode/Flowcharts:



Program Code:

Part B:

;

; GPIO\_Output.asm

;

; Created: 8/24/2017 10:22:33 PM

; Author : Mark

;

/\*\*\*\*

\* GPIO\_Output.asm

\*

\* Modified: 4 May 17

\* Authors: Dr. Schwartz, Colin, Milan

This program shows how to initialize a GPIO port on the Atmel

(Port D for this example) and demonstrates various ways to write to

a GPIO port. The output will blink LEDs at the bottom left of the

uPAD, labeled D5. PortD4, PortD5, and PortD6 are the red, green,

and blue LEDs, respectively. Note that these LEDs are active-low.

\*\*\*\*/

;Definitions for all the registers in the processor. ALWAYS REQUIRED.

;View the contents of this file in the Processor "Solution Explorer"

; window under "Dependencies"

.include "ATxmega128A1Udef.inc"

.equ BIT4 = 0x10

.equ RED = ~(BIT4)

.equ BIT5 = 0x20

.equ GREEN = ~(BIT5)

.equ BIT6 = 0x40

.equ BLUE = ~(BIT6)

.equ BIT456 = 0x70

.equ WHITE = ~(BIT456)

.equ BIT64 = 0x50

.equ PINK = ~(BIT64)

.equ BLACK = 0xFF

.ORG 0x0000 ;Code starts running from address 0x0000.

rjmp MAIN ;Relative jump to start of program.

.ORG 0x0100 ;Start program at 0x0100 so we don't overwrite

; vectors that are at 0x0000-0x00FD

MAIN:

ldi R16, BIT456 ;load a four bit value (PORTD is only four bits)

sts PORTD\_DIRSET, R16 ;set all the GPIO's in the four bit PORTD as outputs

; Notice that the 3 LEDs (RED, GREEN, and BLUE) are all now on, creating white

; The following code shows different ways to write to the GPIO pins.

; Turn on each of the primary colored LEDs in turn, then use some combinations

; These instructions sends the value in R16 to the PORTD pins.

; Since the LEDs are wired as active-low, an R16 = RED = 0xFE = 0b1111 1110

; will turn the RED LED on.

ldi R16, RED

sts PORTD\_OUT, R16 ;send the value in R16 to the PORTD pins

ldi R16, GREEN

sts PORTD\_OUT, R16

ldi R16, BLUE

sts PORTD\_OUT, R16

ldi R16, WHITE

sts PORTD\_OUT, R16

ldi R16, PINK

sts PORTD\_OUT, R16

ldi R16, BLACK

sts PORTD\_OUT, R16

; Why do you think the D5 LED is now dimly on? See the uPAD schematic!

ldi R16, BLUE

sts PORTD\_OUT, R16

; Notice that the OUTSET makes the LED go off, since the LED is active-low

ldi R16, BIT6 ; BIT6 = ~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, BIT6 ; BIT6 = ~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!

Part C:

; Lab 0 part C

; Name: Mark L. Schuster

; Section #: 1540

; TA Name: Christopher Crary

; Description: This assembly application filters data from a given input table

; based on the constraints listed in Lab0's lab doc.

.nolist ; Included for fun.

.include "ATxmega128A1Udef.inc" ;

.list ;

.equ tableSize = 14

.def currentVal = r16 ; Set register 16 to be referred to as "currentVal".

.def temp = r17 ; Set register 17 to be referred to as "temp".

.org 0x0000 ; Start address at 0x0000.

rjmp START ; Jump to the main segment of code.

.org 0x9000 ; Store input table at 0x9000 as specified.

inputTable: .db 0x00, 108 ; Store input table byte-wise two at a time to avoid zero padding.

.db 'G', 0b00100000

.db 'I', 0106

.db ':', 0x42

.db 0b11110001, 74

.db '!', 0xF7

.db 0b00001101, 0x0A

.dseg ; Switch to data memory.

.org 0x3744 ; Move to address 0x3744 as specified.

outputTable: .byte tableSize ; Free space for the output table.

.cseg ; Switch back to program memory.

.org 0x200 ; Place our program at 0x200.

START:

clr currentVal ; Initialize the register holding each table value to zero.

ldi ZL, byte3(inputTable << 1); Temporarily store the most significant byte on the input table's address in ZL.

out CPU\_RAMPZ, ZL ; Load the upper third byte into the left most byte of the Z pointer.

ldi ZH, byte2(inputTable << 1); Load the high byte of the input table into the Z pointer.

ldi ZL, byte1(inputTable << 1); Load the low byte of the input table into the Z pointer.

ldi YL, low(outputTable) ; Point the Y pointer to the output

ldi YH, high(outputTable) ; table.

BEGIN:

elpm currentVal, Z+ ; Begin by loading the current input table value into currentVal and incrementing the Z pointer.

cpi currentVal, 0x0A ; Check if the program has reached the end of the input table.

breq FINAL ; If so, end the program.

cpi currentVal, 109 ; Compare the current table value to 109.

brlo LESSTHANONEZERONINE ; If the current value is less than 109, continue with the algorithm.

rjmp BEGIN ; If not, move to the next value in the input table.

LESSTHANONEZERONINE:

cpi currentVal, 57 ; Compare the current value to 57.

breq CHECKFIFTHBIT ; If the current value is equal to 57, continue with the algorithm.

brlo CHECKFIFTHBIT ; If the current value is greater than 57, also continue with the algorithm.

ldi temp, 9 ; If not, load 9 into temp and proceed to increment the current value by 9.

add currentVal, temp ;

rjmp STORE ; Then jump to store the current value.

CHECKFIFTHBIT:

mov temp, currentVal ; Copy the current value into "temp".

andi temp, 0x20 ; Check is the 5th bit of the current value is set by ANDing it with 0b00100000.

cpi temp, 0x20 ; Check if current value's 5th bit was set.

breq STORE ; If so, jump to store the value.

rjmp BEGIN ; If not, ignore the value.

STORE:

st Y+, currentVal ; Store the current value in the output table, then increment the Y pointer.

rjmp BEGIN ; Then process the next table value.

FINAL:

ldi currentVal, 0x00 ; "Cap" the output table by setting the last value to NULL (0x00).

st Y, currentVal ;

rjmp END ; Jump to the Finish

END:

rjmp END ; Infinite loop :-)

Appendix:

Files:

lab0.asm

