FAU
DEPARTMENT OF COMPUTER & ELECTRICAL ENGINEERING & COMPUTER SCIENCE
College of Engineering & Computer Science
Florida Atlantic University

CDA3331C
Introduction to Microprocessor
Laboratory Manual

Fall
2018

Prepared by

Dr. Bassem A. Alhalabi

Super T.A. Umran Alrazzak

Department of Computer & Electrical Engineering & Computer Science

College of Engineering

Florida Atlantic University

FLORIDA ATLANTIC UNIVERSITY
CEECS DEPARTMENT
LOGIC DESIGN AND MICROPROCESSOR LAB

LD/MP Lab Rules and Regulations

1. **Use your student's ID card to admit ONLY yourself and KEEP the door always CLOSED.**
2. **No smoking / drinking / eating is allowed in the lab. Keep the entire lab CLEAN.**
3. **Do not leave your own documents and/or papers in the lab.**
4. **Lab documents may not be checked out.**
5. **Do not INSTALL / COPY / DELETE / MODIFY any software in the lab.**
6. **Keep in your possession only the chips you are using in your current (or next) experiment.**
7. **When your lab is graded, return the chips to the recycle bin and wires to the wires pan.**
8. **Be conservative on wires by using short runs (1", 2", 5", ..). Reuse used wires.**
9. **Teaching Assistants will be available in the lab at scheduled times, which are posted.**
10. **Completed experiments must be checked/graded by the TA's during scheduled lab time.**

Lab Usage Limitation

You must be enrolled in Logic Design or Microprocessor Courses to use this lab. Further, your experiment should be all thought of before taking time in the lab.

Security

You are continuously video taped for your security and against lab vandalism.

Problem Reporting

Any hardware/software failure and discovery of any lost or damaged item in the lab must be reported to the lab attendants or the CEECS technical support staff.

Penalties

Any violation and/or abuse of the Lab regulations and/or facilities may result in disciplinary actions and loss of lab access privileges.

Lab Access

You should aleady have access to the LD/MP lab via your student ID card. In using this access privilege, you indicate you read, understood, and accept all lab regulations.

FLORIDA ATLANTIC UNIVERSITY
CEECS DEPARTMENT
MICROPROCESSOR LAB

Experiments Grading Sheet

1. Students should maintain this sheet along with this whole lab manual for their own records.
2. TAs must sign and time/date next to every grade, and should indicate if the student cleaned up the workplace after grading each lab.

Lab#	TA Signature	Date	Time	Clean	Grade
0		/ / 2018	:	Y | N	/ **5**
	Special Remarks:				
1		/ / 2018	:	Y | N	/ 1**5**
	Special Remarks:				
2		/ / 2018	:	Y | N	/ **20**
	Special Remarks:				
3		/ / 2018	:	Y | N	/ **20**
	Special Remarks:				
4		/ / 2018	:	Y | N	/ **20**
	Special Remarks:				
5		/ / 2018	:	Y | N	/ **20**
	Special Remarks:				
6		/ / 2018	:	Y | N	/ **20**
	Special Remarks:				
Tota		/ / 2018	:	Y | N	/1**20**
	Special Remarks:				

Student Name: _____ **Z#:** _____ **Signature:** _____

• **CDA3331C • Intro to Microcomputers •** **Lab Assignment** **0**

Name/Semester: **Grade:** /5

[5] 0) This lab is designed to help students get acquainted with the MSP430 Launchpad microcontroller training kit. Type the following sample assembly language program which starts at address 0x0200 (&0200h), or simply $200. The program adds the contents of three consecutive memory locations starting at address $200. The sum is stored at location $206. In the following subsections, various commands are listed for you to explore.

In the Code Composer, create a new Assembly Project and insert the following code into section label "Main loop here". You can also copy the entire skeleton program from the text file provided on black board.

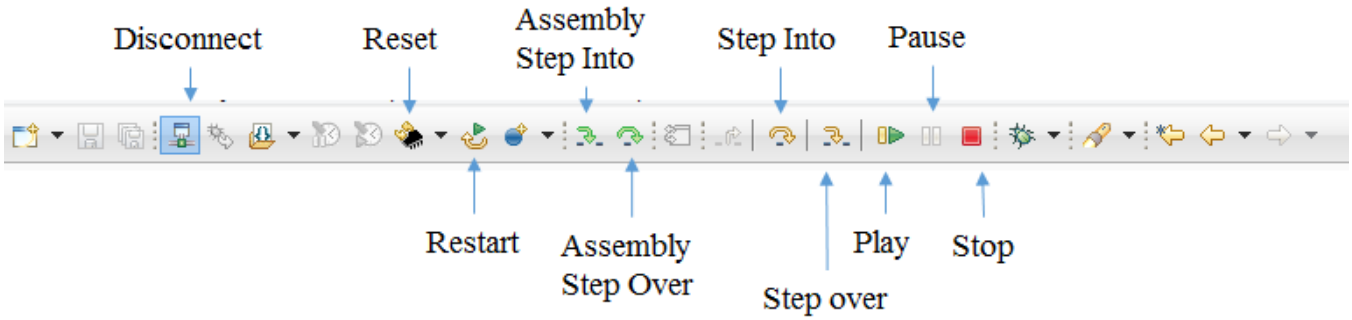
```
LAB1        mov.w  #01, &0200h        ;set a number on location $0200
            mov.w  #02, &0202h        ;set a number on location $0202
            mov.w  #03, &0204h        ;set a number on location $0204

LINEA       clr    R7                 ;clear the entire R7 register
            clr    R8                 ;clear the entire R8 register
            clr    R9                 ;clear the entire R9 register
            clr    R10                ;clear the entire R10 register

LINEB       mov.w  &0200h, R7         ;copy a word from &0200h to R7
            mov.w  &0202h, R8         ;copy a word from &0202h to R8
            mov.w  &0204h, R9         ;copy a word from &0204h to R9


LINEC       mov.b  R7, R10            ;start accumulator in R10 with value form R7
            add.b  R8, R10            ;add to it the content R8
            add.b  R9, R10            ;add to it the content R9
            mov.b  R10, &0206h        ;now store the sum back in memory

Mainloop    jmp     Mainloop          ;Infinite Loop
```



• **CDA3331C • Intro to Microcomputers •** **Lab Assignment** **0**

Name/Semester: **Grade: /5**

[1] 0.a) Exercise 1: Default program execution

- Insert break point at the Mainloop line
- Build and execute program
- Record value of core registers when program stops at the breakpoint

R7_____, R8_____, R9_____, R10_____, SR_____, NZVC____

[2] 0.b) Exercise 2: Memory manipulation

- Soft Reset the micro
- Insert break point at LINEB label
- Insert break point at LINEC label
- Keep the break point at Mainloop line
- Run the program so it stops at LINEB
- Record Values of the following registers:

R7_____, R8_____, R9_____, R10_____, SR_____, NZVC____

- Using the memory browser, modify the content of following memory locations by manually typing the new values over the initial values (in decimal notation):
  - 0x0200 = **02**, 0x0202 = **03**, 0x0204 = **10**
- Run the code, and now it will stop at LINEC
- Record the updated values of the registers:

R7_____, R8_____, R9_____, R10_____, SR_____, NZVC____

- Run the code, and now it will stop at Mainloop
- Record the values again:

R7_____, R8_____, R9_____, R10_____, SR_____, NZVC____

[2] 0.c) Exercise 3: Register manipulation

- Soft Reset the micro
- Remove the break points at LINEB and keep the ones at LINEC and Mainloop
- Execute program to stop at LINEC
- Record the new values:

R7_____, R8_____, R9_____, R10_____, SR_____, NZVC____

- Manually modify the contents of Registers as follows:
  - R7 = **05**, R8 = **01**, R9 = **0** (decimal notation)
- Run the code till it stops at the last breakpoint, Mainloop
- Record the new values:

R7_____, R8_____, R9_____, R10_____, SR_____, NZVC____

• **CDA3331C • Intro to Microcomputers •** **Lab Assignment** **1**

Name: **Grade:** **/15**

[15] 1) Write an assembly program that add the content of Register R4,R5,R6 to register R7 then subtract the content of R10 from R7. Once calculation is done all values of registers must be preserve in memory starting at memory address 0x0200. R4 = 4, R5 = 3, R6 = 10, R10 = 15. The overall program structure should be as follows:

```
Setup       ......          ;clear all registers
            ......          ;Setup Register Values
            ......
Addition    ......          ;Add the content of registers
            ......          ;R4,R5,R6 into R7
            ......
Subtraction ......          ;subtract content of R10 from R7
            ......
Store       ......          ;Store the content of all Register used
            ......          ;into memory including results in the
            ......          ;Order R4, R5, R6, R10, R7.
            ......
            ......
Mainloop    jmp   Mainloop  ;Infinite Loop
```

[3] 1.a) complete the above assembly program.

[12] 1.b) Answer all the Following Questions

- [2] Record the values of Register prior to program execution
  - R4_____, R5_____, R6_____, R10_____, R7_____SR_____, NZVC____
- [2] Rrecord the values of memory location starting at location 0x0200
  - _____

    _____
- [2] Record the values of Register after program execution
  - R4_____, R5_____, R6_____, R10_____, R7_____SR_____, NZVC____
- [2] Record the values of memory location starting at location 0x0200
  - _____

    _____
- [2] Why do we need an infinite loop at the end of the program?
  - _____
- [2] Does the MSP430 Microcontroller support real-time clock?
  - _____

Name: **Grade:** /20

[20] 2) The program of this exercise deals with arrays of numbers and subroutines. Next page is the program outlines. At the beginning of your program you will allocate empty storage for two original arrays and their sorted versions. For the overall program layout, use the program skeleton file (lab2-v38-A-skl) available on Canvas.

[16] 2.a) Complete this MSP430 assembly language program where the **SORT1** section sets the R4/R5/R6 parameters, which are used by the **COPY** and **SORT** subroutines to copy and sort array **ARY1**. R4 holds the starting address of the array. R5 holds the length of the array. R6 holds the starting location of the sorted array. **COPY** subroutine copies the contents of array **ARY1** into **ARY1S**. **SORT** subroutine sorts the elements on **ARY1S** in place.

SORT2 section is similar to SORT1 above using same registers. Arrays are in decimal notation! Sort Arrays in ascending order from lowest to highest value.

Main Program: [6] for Program setup, and [10] for Sort Subroutine.

Use the following values for the array elements.

ARY1: (10, 29,16,-55,90,17,63,59,-35,27,55),

ARY2: (10, 43,84,29,-59,-51,77,79,69,77,64)

If the values in the skeleton code are different, use the values above.

[4] 2.b) Run your program and verify the results by using the *Memory Browser* window in the CCS Debug view. (Hex Values in order)

ARY1S:_____

ARY2S:_____

Name: **Grade:** /20

```
;-------------------------------------------------------------------------------
;-----  Your Sorting lab starts here ------------------------------------------

;Memory allocation of Arrays must be done before the RESET and StopWDT
ARY1        .set     0x0200       ;Memory allocation     ARY1
ARY1S       .set     0x0210       ;Memory allocation     ARYS
ARY2        .set     0x0220       ;Memory allocation     ARY2
ARY2S       .set     0x0230       ;Memory allocation     AR2S

            clr      R4           ;clearing all register being use is a good
            clr      R5           ;programming practice
            clr      R6



SORT1       mov.w    #ARY1, R4    ;initialize R4 as a pointer to array1
            mov.w    #ARY1S, R6   ;initialize R4 as a pointer to array1 sorted
            call     #ArraySetup1;then call subroutine ArraySetup1
            call     #COPY        ;Copy elements from ARY1 to ARY1S space
            call     #SORT        ;Sort elements in ARAY1

SORT2       mov.w    #ARY2, R4    ;initialize R4 as a pointer to array2
            mov.w    #ARY2S, R6   ;initialize R4 as a pointer to array2 sorted
            call     #ArraySetup2;then call subroutine ArraySetup2
            call     #COPY        ;Copy elements from ARY2 to ARY2S space
            call     #SORT        ;Sort elements in ARAY2

Mainloop    jmp      Mainloop     ;Infinite Loop



ArraySetup1 mov.b    #10, 0(R4)   ; Array element initialization Subroutine
            mov.b    #17, 1(R4)   ;First start with the number of elements
            mov.b    #55, 2(R4)   ;and then fill in the 10 elements.
            ......
            ret

ArraySetup2 ......                ;Similar to ArraySetup1 subroutine
            ret

COPY        ......                ;Copy original Array to allocated Array-
            ret                   ;Sorted space

SORT        ......                ;Subroutine SORT sorts array from
            ret                   ;lowest to highest value


;-----  Your Sorting lab ends here   ------------------------------------------
;-------------------------------------------------------------------------------
```

Name/Semester: **Grade:** /20

[20] 3) Implement the following arithmetic function using subroutines and develop the code to be as efficient as possible. The only input to the function is variable (a) that is initialized in register R4 at the beginning of the program to 5 and maintained thereafter. The X calculation result is stored in R5. The final answer, F, is stored in R7.

$$F = \left(\frac{X + 50}{4}\right) \qquad \text{where } X = \sum_{i=0}^{i=|a|} \left(2(i\,!)\right)$$

[12] 3.a) Write an MSP430 assembly language program that implements the above function using subroutines for Multiplication and Factorial. The overall program structure should be as follows. Please note that the multiply subroutine is included in the skeleton code. Please review how it works:

```
RESET      mov.w   #__STACK_END,SP         ;Initialize stack pointer
StopWDT    mov.w   #WDTPW+WDTHOLD,&WDTCTL  ;Stop WDT

LAB2       mov.w   #5, R4      ;Load "a" into R4

CLEAR      clr     R5          ;clear the entire register
           clr     R6          ;clear the entire register
           clr     R7          ;clear the entire register

XCALC      ...     ...         ;the X calculation part of your program
           ...     ...         ;taking value of R4 as an input
           ...     ...         ;and returning result X in R5

FCALC      ...     ...         ;the final part of your program
           ...     ...         ;taking inputs from R5
           ...     ...         ;and returning result F in R7

MainLoop jmp    Mainloop   ;Infinite Loop


MULT       ...   ...         ;Included in the skeleton code
```

[2] for Main Program – [1] XCALC, [1] FCALC

[10] for Factorial Subroutine

Name/Semester: **Grade: /20**

[2] 3.b) Run your program and verify the results by examining registers

R4 = (a) = **_____,** in decimal = **_5____**

R5 = (X) = **_____,** in decimal = **_____**

[2] 3.c) Manually change the contents of R4 to 5, a new input value for variable a. Run your program and observe the results

R4 = (a) = **_____,** in decimal = **_6____**

R5 = (X) = **_____,** in decimal = **_____**

[2] 3.d) Manually change the contents of R4 to -7, a new input value for variable a. Run your program and observe the results

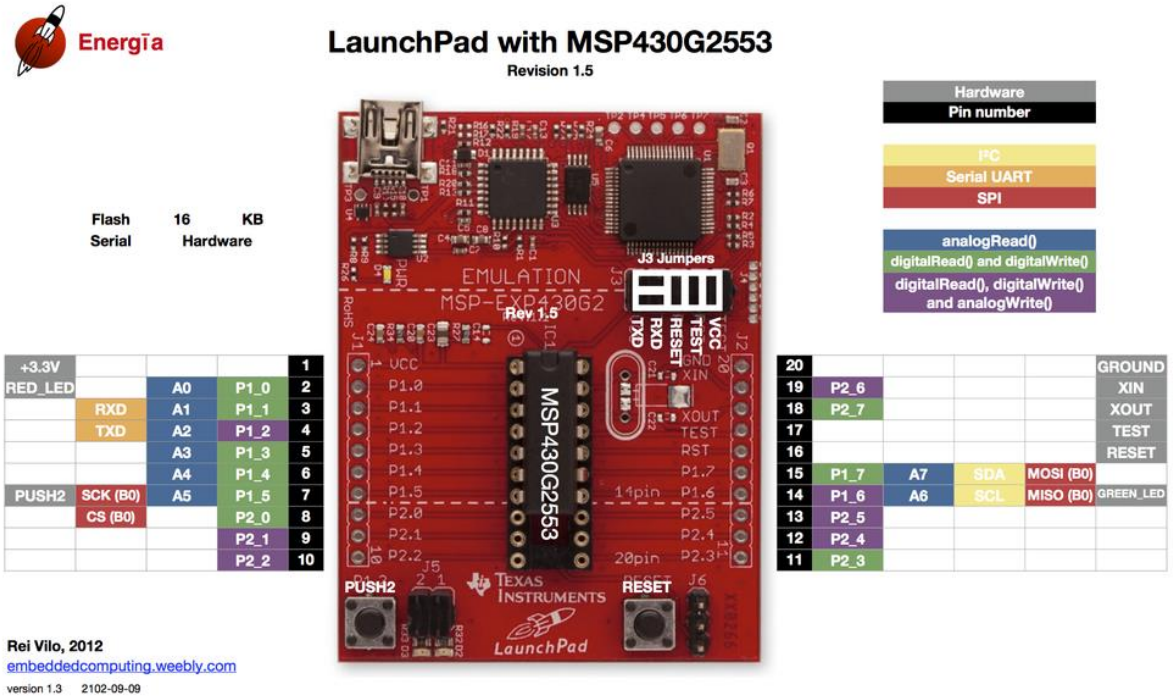
R4 = (a) = **_____,** in decimal = **_-7_____**

R5 = (X) = **_____,** in decimal = **_____**

[2] 3.e) what is the maximum value of "a" that the function can execute correctly and why?

Name/Semester: **Grade:** /20

The MSP430 microcontroller is designed to allow port pins to be individually configured as digital input or digital output and in the case of Port1, all pins P1.0-P1.7 could also be configured as analog inputs. Some pins can be used as analog outputs (analog write). Finally some pins have specific functions, such as serial communication SPI. See the figure below for comprehensive list of all pins functionalities. The labels in gray color are the actual hardware on the Launchpad.



Assemble the circuit below prior to coding making sure to follow diagram. For digital inputs we are using six (6) dip switches connected to port Port2 pins 0-5. Ignore the other four switches we are not using in the dip package. Please note the use of external pull-up resisters (1KΩ) with the switches as they are all configured as active-low. When a switch is turned on, it assets a logic zero (GND) on the port pin.

For digital outputs we are using eight (8) LEDs connected to Port 1 pins 0-7. Please note the use of current limiting resistors (330Ω) with the LEDs on the cathode (GND) side. Also note that LEDs are configured as active high, where a logic high signal (VCC) from the port pin will turn the LED on.

To connect the port pins to your breadboard, you need to use Female-Male jumper cables.

If you choose to take MSP430G2553 chip off the Launchpad board and insert it on your breadboard directly, make sure you use a 3.3VDC supply on VCC pin. Also a Reset resistor of 47KΩ must be used to connect Reset pin to VCC.

Name/Semester: **Grade:** /20

[20] 4) Write a program in C that reads a 3-bit desired light pattern from the 3 input switches connected to pins P2.3-P2.5 and displays the pattern on the 8 LEDs (pins P1.0-P1.7) with some "light playing" options based on the input from other 3 switches (pins P2.0-P2.2).

[8] 4.a) Switch on Port2.0 (Read/Rotate mode)

- On (logic 0) – Read mode: Your program continuously reads the switches and takes only the 3-bits (Port 2.3-2.5) as a desired display pattern and displays them on the LEDs (Port 1.3-1.5). All other LEDs on Port 1 are masked to zero (turned off).
- Off (logic 1) – Rotate mode: Once this switch is turned off, your program will rotate the pattern on the 8 LEDs connected to Port1 pins 0-7. The pattern is the 3-bit value you constantly updated and saved during the Read mode.

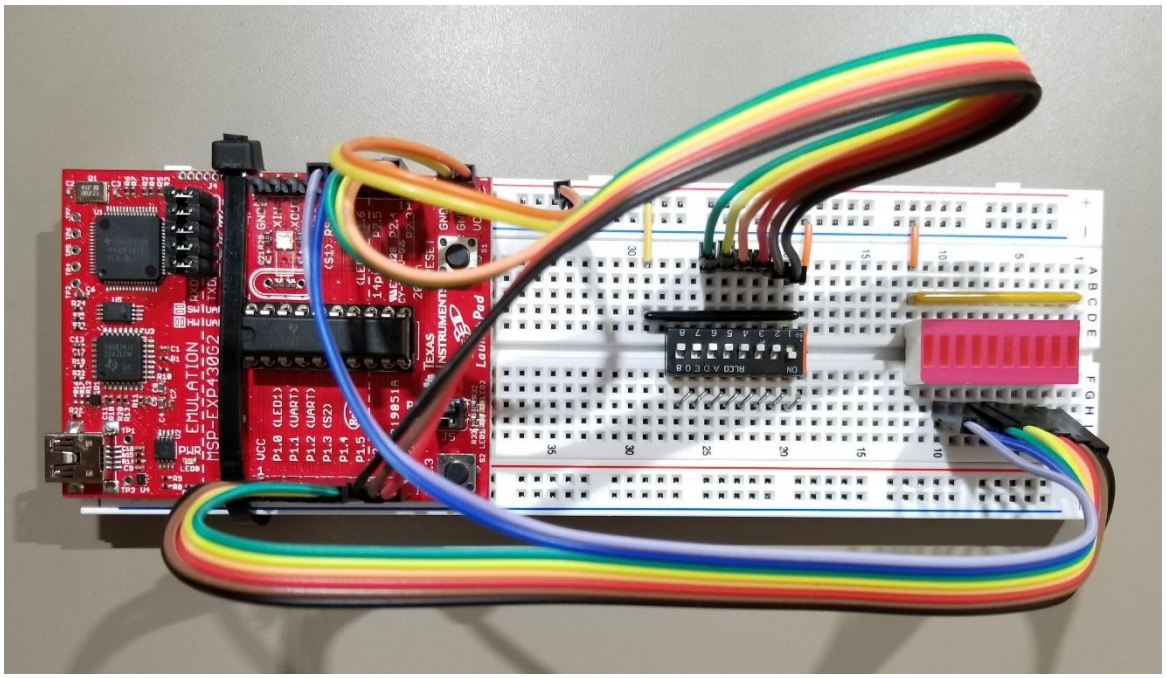
[6] 4.b) Switch on Port2.1 (Left/Right direction mode)

- On (logic 0) – The LED pattern rotates to the Left.
- Off (logic 1) – The LED pattern rotates to the Right.

[6] 4.c) Switch on Port2.2 (Fast/Slow speed mode) **Hint: Software Delay**

- On (logic 0) – The pattern rotation is Fast.
- Off (logic 1) – The pattern rotation is Slow.

[2] 4.d) **[BONUS]** Modify the program so that the two Switches on Port2.0-1 provide four different fun patterns which continuously display/move on the 8 LEDs. Keep Switch on Post 2.2 to change speed. ***Please be creative***.



• **CDA3331C • Intro to Microcomputers •** **Lab Assignment** **5**

Name/Semester: **Grade:** **/20**

[20] 5) Write a C program using the MSP430G2553 Launchpad which creates a counter from 00-99. Your kit may have 7-seg displays with different pinout than the one in the drawings. As always check datasheets of your hardware components before wiring. For input control we use three DIP switches SW-321 which are connected to pins P2.5-3 with pull-up resistors. The different combinations of these 3 switches determine the operation of the counter as explained below. The two 7-segment digits are multiplexed (turned on alternately) by two control signals coming from P2.0 and P2.1. The two 2N2222 transistors with 1K Ohm resistors on the gate are used as power drivers for the Common Anode pins of the displays. The 8 segments (a,b,c,d,e,f,g) of both displays are connected to Port 1 pins P1.0-P1.6 through a resister network (470-1000 Ohm). Please use the figure below as a sample how to organize components and wires on your breadboard. For exact pinout of the 2-digit 7-segment display, check the actual data sheet on the display part you are given in the lab kit. If the displays have Common Cathode instead of Common Anode, you need to use PNP transistors instead of NPN.

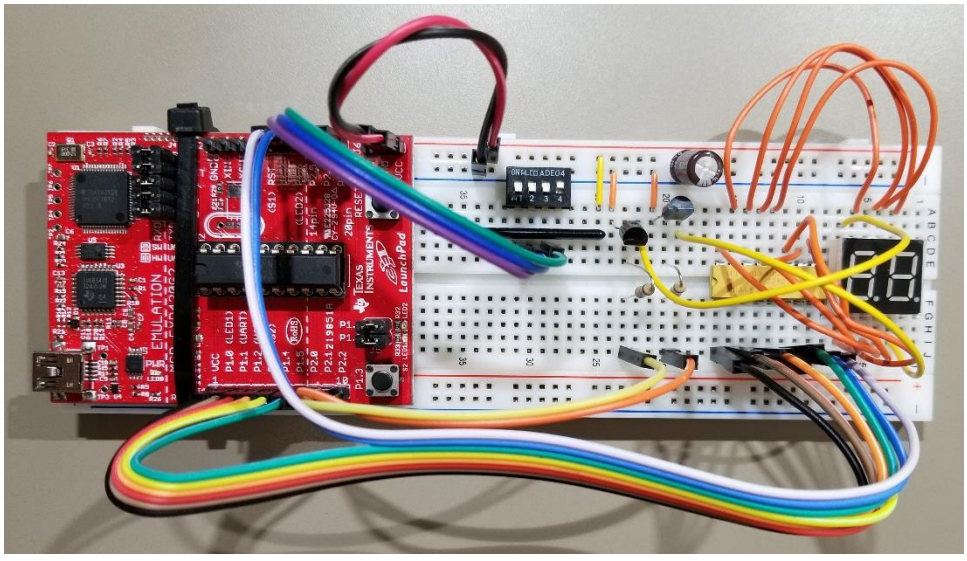
[4] 5.a) Build the circuit as described above and run the skeleton code.

[8] 5.b) Modify the code so that the counter initial number setup goes as follows. You may wish to slow down the display update so you see the changes better:

- SW-321 = 000: Counter resets to 00
- SW-321 = 001: Right digit cycles 0-9
- SW-321 = 010: Left digit cycles 0-9
- SW-321 = 011: Right and left digits both hold values (preset value)

[8] 5.c) Counter counts up or down

- SW-321 = 111: Counter cycles up from the preset value to 99
- SW-321 = 100: Counter cycles down from the preset value to 00
- SW-321 = 101: Pause Counting
- SW-321 = 110: Pause Counting



Name/Semester: **Grade:** **20**

[15] 6) In this lab, you will learn about reading analog values into the microcontroller. All pins of Port1 can be connected to analog signals which can vary form 0.0V-3.3V, and once converted by the internal ADC, the digital value for each port pin will be 0-255 for 8-bit resolution or 0-1023 for 10-bit resolution.

Assemble the following circuit with three analog signals as shown in the schematic below. For digital outputs we use the three LEDs 1, 2, and 3.

- The temperature sensor LM34DZ is connected to Port1.0 (A0). It is linear and it outputs 0.1 V for every degree, for example a room temperature of 78 F degrees, would show as a .78 V signal, which is translated into 10-b digital value as .68 x (1023/3.3V) = 210. The exact value will fluctuate due to the imprecision of the sensor and the noise around it.
- The touch switch is connected to Port1.1 (A1) and a 100K ohm pull-up resistor. When you touch the transistor base, your body acts like a ground sinking a very little current form the PNP transistor, which is enough to trigger it due to its high gain (10000). So when you touch the base and the transistor is turned on, the collector voltage will drop form VCC level (#1000 value) to almost ground (#50 value). To help your body to serve as a better GND, touch the GND as well when you touch the base.
- The photo cell is connected to Port1.2 (A2). With the 10K resistor in series working as a voltage divider, you read about .97 VDC on A2 at room light. When you block the light completely from the photo sensor, the reading goes higher to 3.1 V. When you apply direct light onto the sensor, the reading goes lower to 0.3 V.
- LEDs 123 are connected to Port pins 1.4, 1.5, 2.0. LEDs must have current limiting resistors in series with value between 100-470 ohms.
- Analog vales are very fluctuating in nature and the ADC converter inside the micro works very fast, so it is recommended to add a small capacitor on the analog wire. In the software, it is also recommended to read few samples and average the reading.

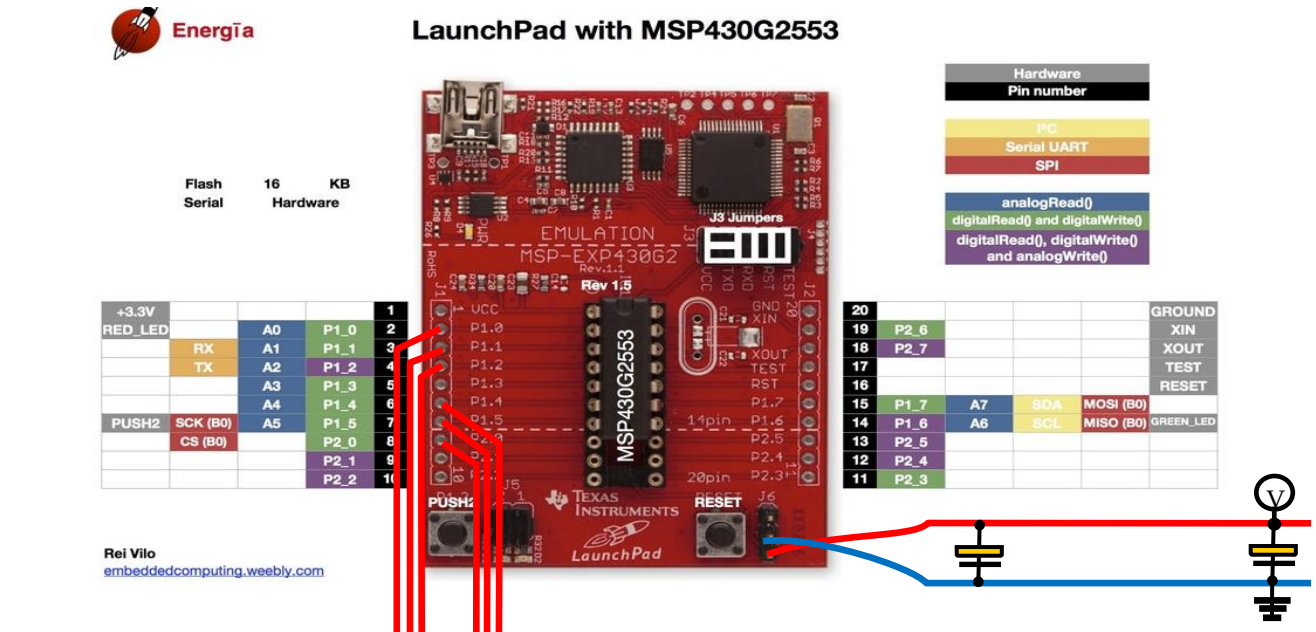
The skeleton software provided in C will run on this circuit and it will write the three analog values in the three variables, temp, touch and light. Run the software and check the values in the debugger. Note that the skeleton code takes the initial room readings from the three sensors and saves them as baseline values before the program enter the infinite loop, lightroom, touchroom, and temproom.

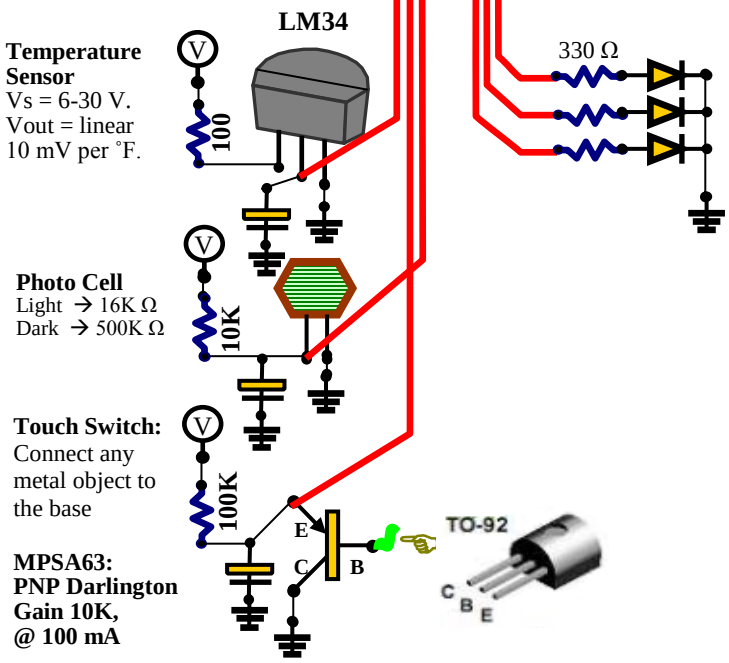
[8] 6.a) Observe how LED 1 changes on/off as you move your hand over the light sensor. Explain to the TA why LED1 is not flickering as it alternates between on and off, the hysteresis concept.

[6] 6.b) Modify the main infinite loop so that If the room temperature reading increases by 2% over the room value, LED 2 turns on (think of it as an Air Conditioner). If the temperature decreases to the original room temperature or below it, LED 2 will turn off. LED should not flicker!

[6] 6.c) Modify the main infinite loop so that every time you touch the touch switch, LED 3 will toggle between on and off. LED 3 should not flicker!

Name/Semester: **Grade:** 20



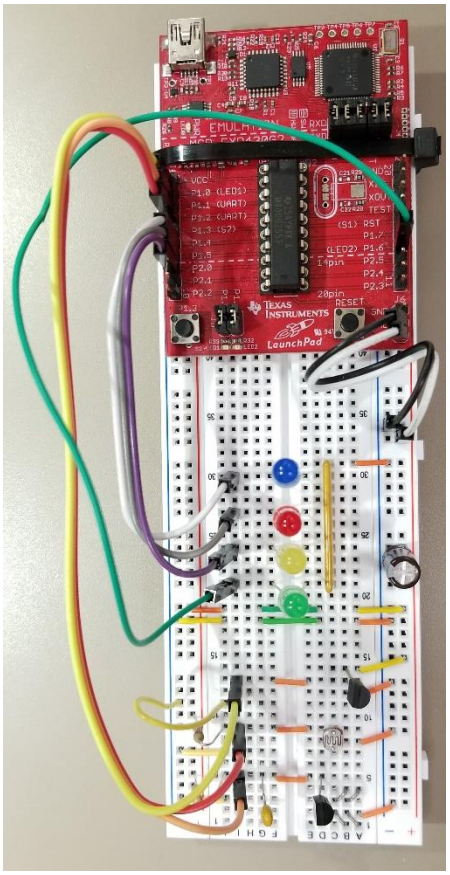


Temperature Sensor
Vs = 6-30 V.
Vout = linear
10 mV per ˚F.

Photo Cell
Light → 16K Ω
Dark → 500K Ω

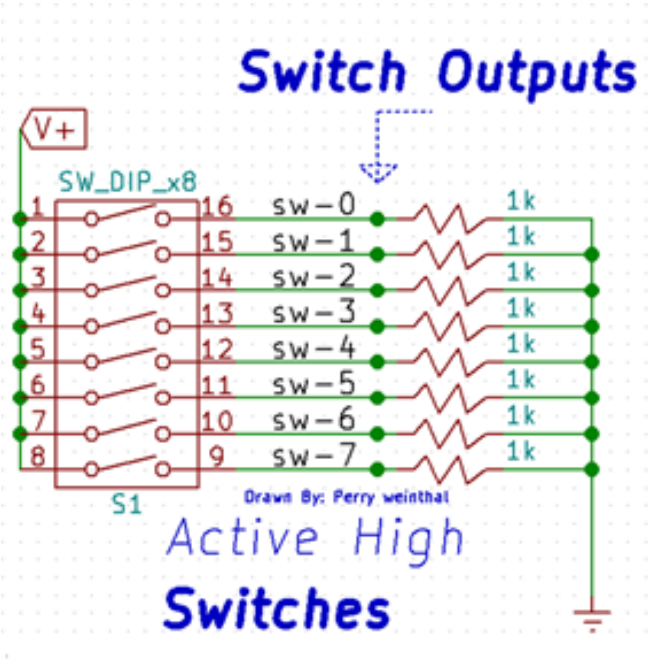
Touch Switch:
Connect any metal object to the base

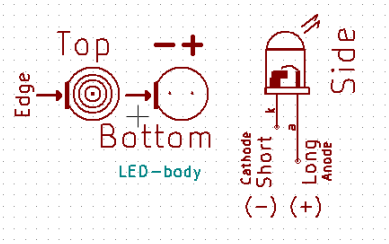
MPSA63:
PNP Darlington
Gain 10K,
@ 100 mA

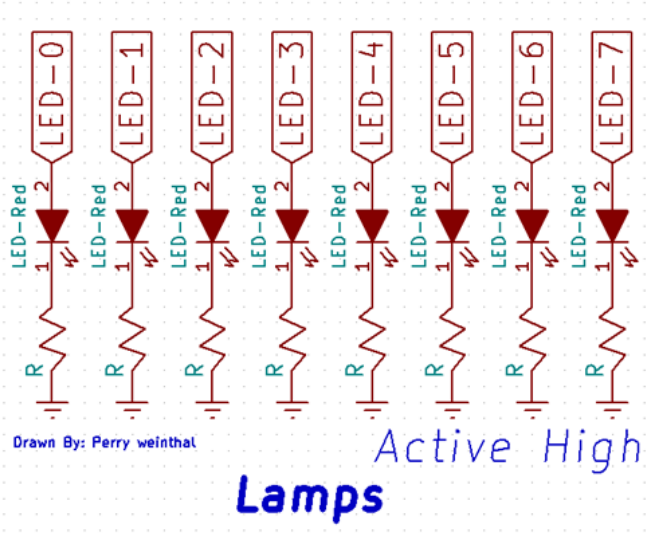


Additional Lab Hints, Notes and Schematics
Courtesy of Lab manager Perry Weinthal - EE205

Switches and LEDs connections



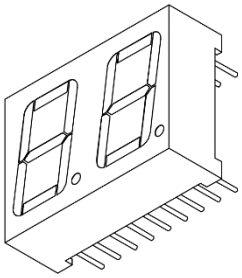


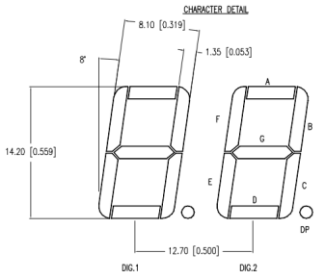


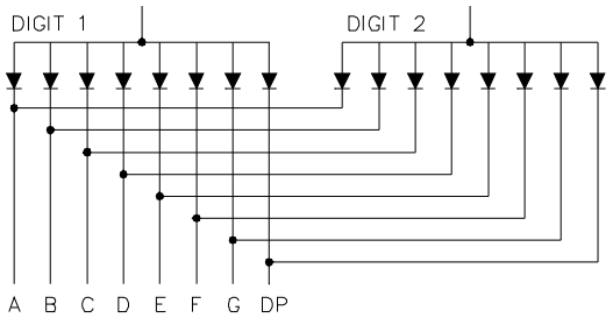
Seven-Segment Displays

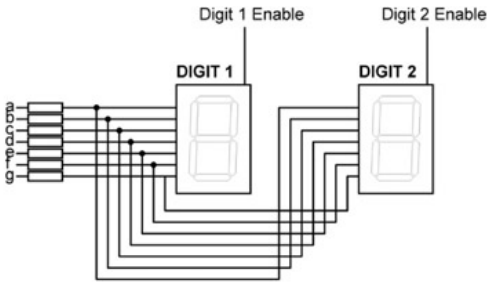
- ***See the Datasheets for the parts in your kits. Use Google:*** **part_number "datasheet .pdf"**
- **Some displays require each segment to be wired.**
- Others have a common pin for each segment (CA or CC), which is used as a digit selection.
- ***All LED segments require current limiting resistors:*** R=180 - 330 ohm, lower for brighter, until your microcontroller smokes.

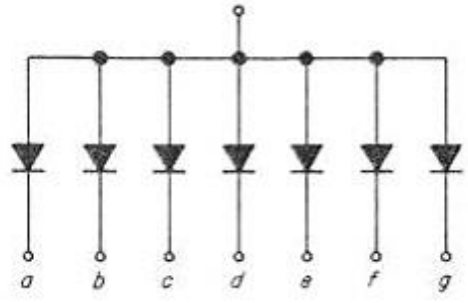
A Common Anode (CA) Display has the 7 LEDs connected together by their Anodes. A Common Cathode (CC) Display has the 7 LEDs connected together by their Cathodes. Some packages come with Joined Segments. There is a significant difference between these two types in how these LEDs are powered. "Common Anode segments draw current directly from the power source using the Vcc pin and sink it using Microcontroller pins, but Common Cathode LED's are sourced by pins of the microcontroller and sink by means of GND pin." Courtesy: Gadgetronicx



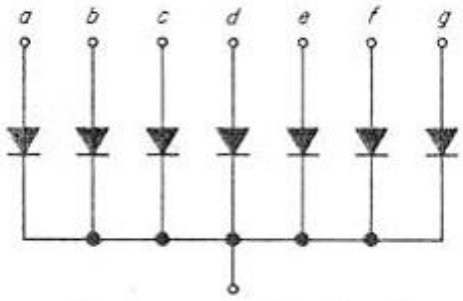








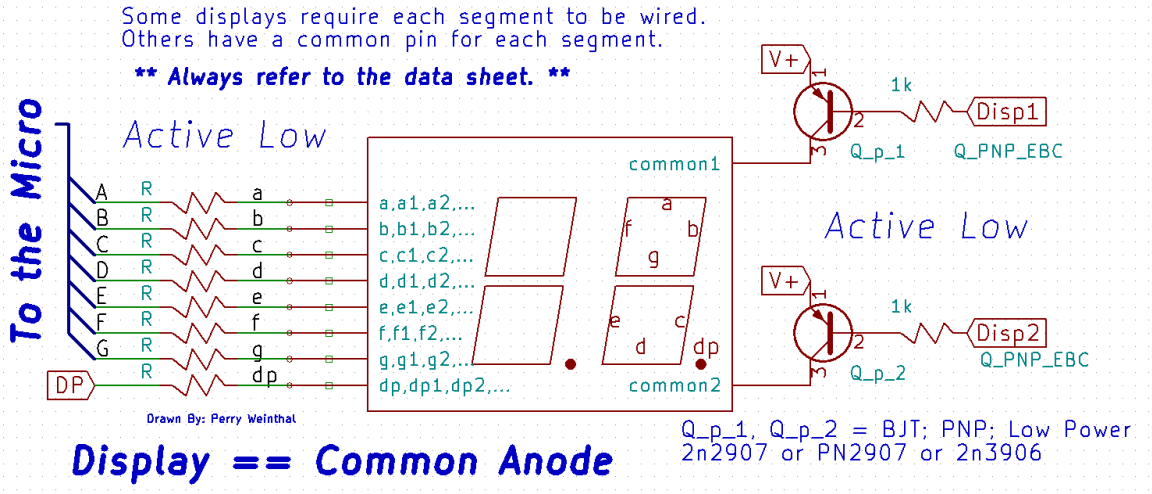
Common Anode



Common Cathode

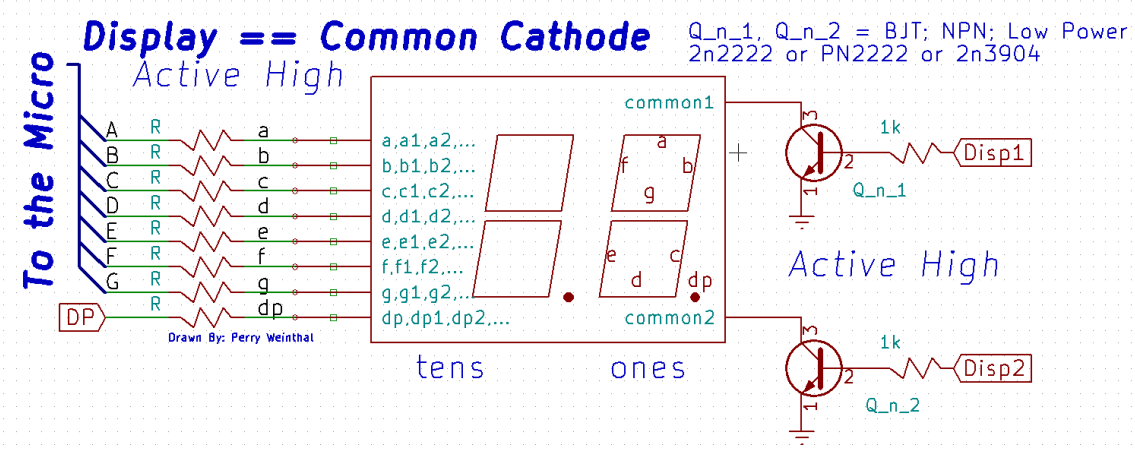
For the Common Anode:

- A PNP will connect the Common Anode to the High (3.3V or 5V)
- 2n2907 or PN2907 or 2n3906 are PNP, ~500mA in a TO-92 type package (The “D” package)
- Emitter (Arrow Pointing IN) is connected to the “HIGH Rail”
- Base is an **Active LOW Control**
- Collector is to the Display Common



For the Common Cathode:

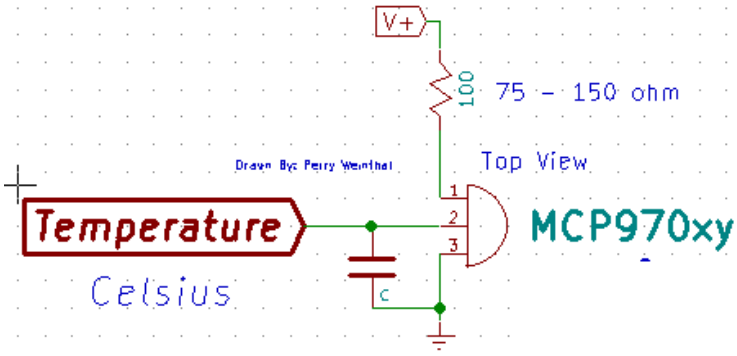
- A NPN will connect the Common Cathode to the Low (GND).
- 2n2222a or PN2222a or 2n3904 are NPN, 500mA in a TO-92 type package (The “D” package)
- Emitter (Arrow Pointing OUT) is connected to the “LOW Rail”
- Base is an **Active HIGH Control**
- Collector is to the Display Common



Sensors

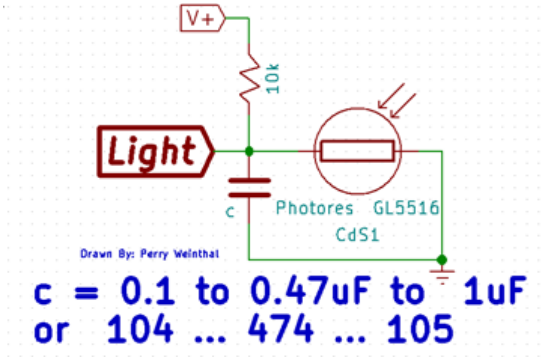
Temperature Sensor:

Your kit may not contain the LM34/LM35/LM36.
Instead it may contain the MCP9700 or MCP9701, **you need to look up the slope / curve of this device.**



Light Sensor:

It is basically a light-sensitive resistor, so along with another resistor, it work like a voltage divider.



Touch Sensor:

It is a high gain PNP transistor, and your body touching the base acts like a ground.