Lab #3: Assembly Programming & Elementary Wiring

**Purpose**

This lab is an exercise in elementary assembly programming and basic wiring. It requires students to write short assembly code routines for statistical calculations on a sample population and to wire in an LED display and switches to the HC6812. If you do not have your 68HC12 development board completed and programmed do so at an earlier lab.

**Part I. Program Description**

We would like to convert an array of student grades to ASCII letter grades. See below for details:

1. Assume that your TA will download into your emulator memory a vector of student test grades. The starting address for the vector will be denoted as `score_addr` and an equate statement should be used at the top of your program to set this variable easily to a new address supplied by your TA.

2. The length or number of elements in the vector will be stored as a single byte at address `score_vector_len` which also will be given to you in lab.

3. The test scores or elements of the vector will be unsigned bytes that range from 0 to 100.

4. We would like you to create a new vector of ASCII letter grades that correspond to the vector of test scores. For example if the first element in the scores vector is from 90-100 then store the ASCII equivalent of ‘A’ in memory. Here is the breakdown for the grade computation:

   - 100-90 score => store ASCII ‘A’
   - 89 - 80 score => store ASCII ‘B’
   - 79 - 70 score => store ASCII ‘C’
   - 69 - 60 score => store ASCII ‘D’
   - 59 - 0 score => store ASCII ‘F’

   Thus if the second score in the vector is 73, you would store a ‘C’ in the second memory location of the new vector and repeat for all scores as required by `score_vector_len`. The starting address for this new vector will be called `grade_addr` and will again be given to you in lab.

5. As mentioned earlier, your TA will give you all required input/output addresses in lab. Thus you should use assembler equate statements for these constants in your program. You should also use equate statements to define origins for the program and data sections. This will enable your code to be easily moved (re-located) in memory.

6. Create a dummy score vector and length and test your program at home. Don’t try to write the code in lab… you won’t have time! Your TA will also be asking you questions on how your program works to see if you are the original author.

**Part I. Pre-Lab Work**

1. Create a program flow chart for your program.

2. Code each block in your flow chart in 68HC12 assembly.

3. Assemble and simulate your code outside of lab to verify proper function.

4. Bring the flow chart and program code (assembly and list files) to the lab on diskette and also on hard copy.

**Part I. In-Lab**

1. Your TA will not allow you in unless you have the pre-lab materials specified above.

2. A vector containing test scores will be given to you as well as the vector length. Re-assemble and test your code with this new information.

3. Verify your computed results with your TA.

4. Answer any questions they have regarding your code & program operation.

**Part II. Adding a 4 Bit LED Display in Lab**

We would like to connect 4 LEDs in a DIP type package to Port T of the 6812. This will enable us to write out binary values for visual information feedback during program execution.
Lab #3: Assembly Programming & Elementary Wiring

While this sounds quite elementary, it will turn out to be an extremely beneficial debug tool during run-time.

1. Connect Port T (T3:0) using wire-wrap wires to four series resistors (where the resistance should be in the range of ~220-680 ohms). It is suggested that you mount the resistors (DIP RPACK) in a wire-wrap socket such that the pins face down in your development board.

Note: The Port T header is defined as follows:

header pin1 => T0  header pin3 => T1  header pin5 => T2  header pin7 => T3
header pin9 => GND  header pin2 => T7  header pin4 => T6  header pin6 => T5
header pin8 => T4  header pin10 => GND

2. Connect the other side of the series DIP resistor to the Anode of an LED in the LED DIP package as shown below.

Note: LSB should be rightmost LED.

3. Once your binary display is connected write "1s" to DDRT (Data Direction Register Port T @ $00AF) for bits DDRT3:0 using the Monitor's Memory Modify (MM) command. This configures Port T's lower four bits as outputs. Note: Consult the online 6812 documentation for more information.

4. Using the Monitor again, write out a value to Port T ($00AE) and check if the proper LEDs light.

Note: If the proper LEDs do not light, check your wiring. If you find no errors, check the voltages at the actual 6812 Port T pins on the device (U4). Pin definitions for the 68HC12B32 can again be found in our online documentation.

Part III. Adding an Input Switch in Lab

We will now add a 4 bit DIP switch (using a 8 SPST switch DIP package) to Port T. This will enable us to manually input information to the 6812 while it is executing code.

1. Connect 4 DIP switches to pins 8, 6, 4, 2 of the Port T header.

Note: The DIP switch should be placed in a socket on top of the board with the wire-wrap pins facing down.

Note2: The rightmost switch from the top view should be connected to the T4 of Port T.

2. Connect the other side of the four DIP switches to GND.

3. Connect a SIP pull-up RPACK to the side of the 4 switches that are connected to Port T. Place the SIP RPACK in half of a DIP wire-wrap socket for easy connection.

4. Connect the common node of the RPACK to +5V. For clarification purposes, see the diagram that follows:
Lab #3: Assembly Programming & Elementary Wiring

5. Using the Monitor (MD command) read in the value of your switches. If you do not see the correct value, check your wiring. If your wiring is okay, check the voltages on the corresponding Port T pins of the 6812. You can also measure the resistance between your DIP switch and a Port T pin on the 6812 to verify that it is ~2 ohms or less.

6. Create a test program that reads Port T and writes the value out to Port T. The program should continuously scan the switches so that any change is reflected on the LEDs. Note: The code should be placed at $0900 which is an open area of internal SRAM.

7. With help from your TA, load the program into the development board and run your code. This process consists of typing "load" on the monitor command line, hitting enter, and then using the "Transfer" pull down menu in the Hyper Terminal application. You use Hyper Terminal to send your *.S19 file out the serial port of your PC to the development board by selecting "Send Text File" in the "Transfer" pull down menu and selecting your appropriate *.S19 file.

Note: You can use MiniIDE instead of Hyper Terminal to load your code onto your board as described above.

Point Break-Down

Pre-Lab Materials
A. flow chart for main program, 5%
B. flow chart for subroutines, 5%
C. list file print-out & on diskette, 5%

In-Lab Participation
D. ability to debug and find errors, 10%
E. flexible input/output variable addressing assignment, 10%
F. modular & relocatable code (extensive use of org and equ statements, clear concise program function) 10%
G. correct results for outputs from data given by TA & questions answered correctly, 25%
H. functional LED output port, 10%
I. functional input switches, 10%
J. program that echoes input switches to output display, 10%