Lab report: Lab 2

By: Ofek Witkon

Part A

1. Introduction

The aim of this experiment was to develop an Assembly-language program for the 8051 that can interface with external input and output devices. The program reads user input from a 4x4 keypad matrix, identifies the key and presents the corresponding output on a multiplexed 4-digit 7-segment display. Because two of the I/O ports of the 8051 board are used for external memory, an 8255 I/O expansion board is used to connect the multiplexed display and the keypad to the 8051 microcontroller. Part A of this report aims to explain the hardware configuration, describe the design and operation of my assembly program, and discuss the performance and potential improvements.

2. Description of the hardware: keypad, display and 8255.

This section describes how the hardware used in the experiment is configured, with emphasis on the role of the 8255 I/O expansion device, the organization of the keypad matrix, and the multiplexed 4-digit seven-segment display.

2.1 The 8255 and port configuration

The 8255 is used in the experiment to provide additional I/O lines for the keypad matrix and the seven-segment display. The device is connected to the MCU via port 1 which acts as a data bus, and through additional 4 bits of port 3 for A0, A1 port selection signals and active low RD, WR control signals. The 8255 offers three 8-bit ports labeled A, B, C which are mapped into the 8051 external memory space. Port A is an output to the 4 digits segment lines- each bit of port A drives a specific segment high. Port B is configured as an output; each of its lower 4 bits activates one of the 4-digits. Port C is divided into 2 parts, the upper 4 bits of the port (C4-C7) are output connected to the rows of the keypad matrix and port C lower 4 bits (C0-C3) acting as the column inputs. This required configuration of Port A, B, and the upper part of port C as outputs and the lower part of port C as inputs is achieved by writing the control word 81h (10000001b) to the 8255 as specified in table 2 of the lab sheet.

2.2 Keypad scanning method

The keypad is a 4x4 matrix, all rows are connected via port C (C7-C4) and are driven as output, while all columns are connected via port C bits (C3-C0) and read as inputs. The columns are held down by a pull-down resistor, so they read logic 0 when not pressed. Each key is positioned at the intersection of a specific row and Column acting as a button. When a key is pressed, there is an electric path connecting the row and column of the specific key. If the row of the key is high, the press will drive the corresponding column high.

To detect if any key has been pressed, the program initially drives all rows high and reads the column input. If all column bits are low, no key has been pressed. If any of the column bits are high, at least one key is pressed. Once a key press has been detected, the program identifies the specific key by driving each of the rows high at a time and reading the columns input after each change. The combination of one active row with one active column is unique to each key.

2.3 Multiplexed display operation

The 4-digit-seven-segment display is connected to the 8255 through ports A and B, port A connects to the 8 segment lines of all the digits and Port B is used to activate each of the digits as shown in figure 4 of the lab sheet. Because only one pattern can be written on the segment bus at a time, the program cycles through the digits, activating each one for a short time while writing the appropriate pattern on port A, due to the fast switching between the digits this gives the appearance that all four digits are lit simultaneously. The program holds the value for each of the digits in registers R0-R3 corresponding to digits 0-3, the display routine writes each register value to port A and activates the corresponding digit through port B, by repeating this routine continuously inside the main loop the four digits appear continuously lit with minimal flickering.

3. Explanation of Assembly program

3.1 Structure Overview

The program is structured to run continuously, after initializing the 8255 and clearing the digit registers (R0-R3) the program enters an infinite main loop. At the beginning of this loop, the program checks for a key press. If no key is pressed, then the display is refreshed after which it jumps again to the start of the loop. If a key press is detected, the program scans the keypad row by row, to identify the exact key and decode its position into its seven-segment display value using a lookup table. The program then shifts the registers

values and stores the new detected value in register zero. The program then compares the detected value with the seven-segment value of the F key and clears the display registers if equal. Before updating the display, there is a debouncing routine meant to make sure that the key has been detected only one time and has been released. Overview flowchart is on the appendix

3.2 Initialization routine

This is the first part of the code, it is meant to configure the 8255 to ensure the keypad and the display are used correctly. This is achieved by setting up the appropriate control signals on port 3's A0, A1, RD, WR pins to program the 8255 control register and sending the control word 81h via Port 1 to configure Port A, Port B, and Port C as required by the hardware. After configuring the 8255, the program clears the digit registers (R0-R3) to make sure the display starts in a blank state before entering the main loop.

3.3 Keypad scanning

The program detects if a key is pressed by first driving all the keypad rows high and reading the column inputs from Port C. The lower 4 bits of Port C represent the columns, so the code masks the upper bits and checks if any column is high. If the column reads zero then no key is pressed, and the program returns to refresh the display. When one of the column bits is high, the program stores the value at the "col4bits" register (R5) and begins a row-by-row scan to identify which key was pressed. In this stage only one row is high at a time by writing a rotating bit pattern to the upper 4 bits of Port C. As each row is high, the program reads the columns input. When a row produces a non-zero column input, the combination of the exact row and column is used to identify the exact key that was pressed as each key has a unique row and column pattern. These patterns are stored in the corresponding "col4bits" and "row4bits" and they are then converted to an index value (0-15) which is later used to retrieve the correct seven segment code from a lookup table.

3.4 Key identification and lookup table

Once the program has identified the active row and column of the key that has been pressed, the 4 upper bits representing the rows are stored in the "row4bit" (R6) register and the lower 4 bits representing the column are stored in the "col4bit" (R5) register. The row and column patterns are first passed through the "convert2index" subroutine. This subroutine rotates the pattern into the carry bit and returns an index from 0 to 3 that represents the row or column number. Because the row pattern is read from the upper 4 bits of Port C, it is shifted 4 bits to the right before being passed to the "convert2index"

subroutine. The row index is then multiplied by four, and the column index is added to it resulting in a final key index between 0 to 15 that corresponds to the position of the key in the 4x4 matrix. This index is then used with the indexed addressing instruction, to retrieve the appropriate pattern for seven-segment display from a lookup table stored at labels KCODE0 to KCODE3. The lookup table matches the keypad layout, and it contains the pre-computed segment patterns for keys 0-9 and A-F. After the code for the pressed key has been retrieved and moved to R0, the program checks whether it corresponds to the 'F' key by comparing the segment value with the constant 71h. If the values are equal, the pressed key was the 'F', in that case a special subroutine is called to clear the display registers and update the screen.

3.5 Register Shifting Routine

After detecting a key press and before updating the value of R0 the "movReg" subroutine is called. The "movReg" subroutine shifts the digit registers to the left. It starts with copying the R2 register value to R3 after which R1 is copied to R2, and R0 is copied into R1. This order ensures that no important data is lost in the process. The original value of R3 is the oldest value, therefore it is redundant and overwritten. After the shifting of the old values, the newly decoded seven segment pattern is moved into R0 so that the most recent key press appears in the rightmost digit of the display. This mechanism ensures each key press updates the display in the correct order.

3.6 Display Routine

Once the digit registers have been updated, the next stage of the program is to update the multiplexed display. For each of the digits, the program writes the segment code from the digit register to Port A of the 8255. It then selects the appropriate digit by writing a bit sequence to port B, where only one of the lower 4 bits is set at a time. After activating each digit a short delay subroutine is called to allow the segment pattern to remain visible for a short period before moving to the next digit. This sequence is repeated for all four digits and ends by jumping to the start of the main loop. The display routine is executed repeatedly when no key is pressed making the display appear continuously lit due to the high refresh rate.

3.7 Performance Evaluation

The program performs reliably in simulation and on hardware. During testing, every key press was correctly detected and interpreted, including rapid and lightly pressed keys. The debouncing method used (combining delay with a release verification loop) proved

effective, as no false or repeated detection was observed. Even when several keys were pressed simultaneously, the program consistently captured the first valid key press. The display also behaved as expected, with no noticeable flicker and stable brightness across all digits. The display appeared smoothly and continuously illuminated, except for a moment when a key is pressed. The register shifting mechanism works correctly with new digits consistently appearing on the rightmost digit of the display and older values shifting left as intended. The 'F' key also operated reliably, clearing the display registers (R0-R3) every time it was pressed. Overall, the program functions exactly as designed, with stable keypad operation and correct display behavior.

3.8 Possible Improvements

Although the program operates correctly, several improvements could be made to enhance clarity, efficiency and overall design. One improvement would be to restructure the code so that more functionality is organized into subroutines. This would make the program easier to read and modify. Some operations, such as detecting the 'F' key, could be optimized by checking its unique row and column bit pattern before calling the "convert2index" subroutine, which would save instructions and improve performance.

The current debouncing method works reliably on the hardware, but it relies on a blocking delay. By using a more complex debouncing method we can avoid blocking the program, making it more responsive while preventing bounce.

The display is stable, but its refresh rate depends on the timing of the main loop. Using a timer interrupt for the display update could ensure a consistent refresh frequency and remove the dimming that occurs during the processing of a key press.

Finally, the use of interrupts in general could make the program more efficient. A timer interrupt could handle the display refreshing, allowing the main loop to focus only on keypad input.

Part B

4.1 Overview of the Extended functionality

In this part, the original system is extended to implement a 2-digit decimal adder. The user can enter two numbers each in the range of 00-99, store each number, select an operation, and obtain a result in the range of 000-198. The keypad keys A-F are reassigned to provide control function for the simple calculator. Additional registers and lookup tables are required to convert the entered digits into numerical values and to perform the BCD addition.

4.2 Entering and storing the first Operand (Key A)

The first operand is entered by pressing one or two digit keys. If only one digit is pressed, it is treated as the ones digit. If a second digit is pressed the first digit becomes the tens and the second digit becomes the ones digit. The user cannot enter more than two digits, and any extra key press is ignored until the operand is stored or cleared. When the A key is pressed, the program detects that the first key should be stored. The segment codes in the display registers R1 and R0, which represent the tens and ones digits, are converted into their numerical BCD value using a lookup table. These values are then stored in register bank 2, with R1 holding the tens value and R0 holding the ones value of the first operand. After storing the first number, the display registers are cleared, so the user can begin entering the second operand.

4.3 Entering and storing the second Operand

The second operand is entered and stored the same way as the first number. The user can press one or two digit keys, where the first digit becomes the tens digit and the second becomes the ones digit. Only two digits are allowed at this stage, and any extra digit presses are ignored until the operand is stored or cleared. When the B key is pressed, the program converts the digit segment codes held in the display registers into their numerical BCD values and stores them in register bank 2, with R3 holding the tens digit and R2 holding the ones digit. After storing the second operand, the display registers are cleared.

4.4 Selecting the Operation

When the user presses the '+' key (which corresponds to key C on the keypad), the program detects that the addition operation has been selected and records this choice in the operation register. Although this step is not strictly required for the simple 2-digit adder in Task B, it is useful for expendability, as the same mechanism can support more operations in the future such as subtraction. The operation register used for this purpose is R5 in register bank 2, and a simple internal code is written into it to indicate that addition was chosen. As an optional user-interface feature, the program could also display a 'plus' on the display until the next operand is entered. Even though the system currently performs only addition, this operation-selection stage creates a clearer sequence for the user (first operand → operation → second operand → equals).

4.5 performing the Addition (the '=' key)

When the '=' key is pressed (key D), the program checks the operation register and performs the required operation between the two stored operands. In the case of addition, the calculation is carried out using BCD addition. The first operand is stored in register

bank 2 in R1, R0 (tens and ones), and the second operand is stored in R3, R2. The program first performs a BCD addition of the ones digits (R0 + R2) and handles any carry. It then adds the tens digits (R1 + R3) together with the carry, and if needed, handles the final carry to form the hundreds digit. Each of the resulting digits is then converted back into a seven-segment code using a lookup table and written to the display registers in bank 0 (R0 for the ones, R1 for the tens, and R2 for the hundreds if required).

4.6 displaying the result

After the BCD addition is completed, the result can contain up to three digits in the range 0–198. The result digits are converted to the corresponding segment values and written to the display registers in register bank 0. the ones digit in R0, the tens digit in R1, and the hundreds digit in R2 if needed, while R3 is left blank. If the result does not contain a tens or hundreds digit, those display registers are also left blank. The display routine itself remains the same as in Part A, only the values in the display registers are updated to present the result. After showing the result, the system waits for the user to clear the display and start a new calculation.

4.7 CLEAR function (the ‘F’ key)

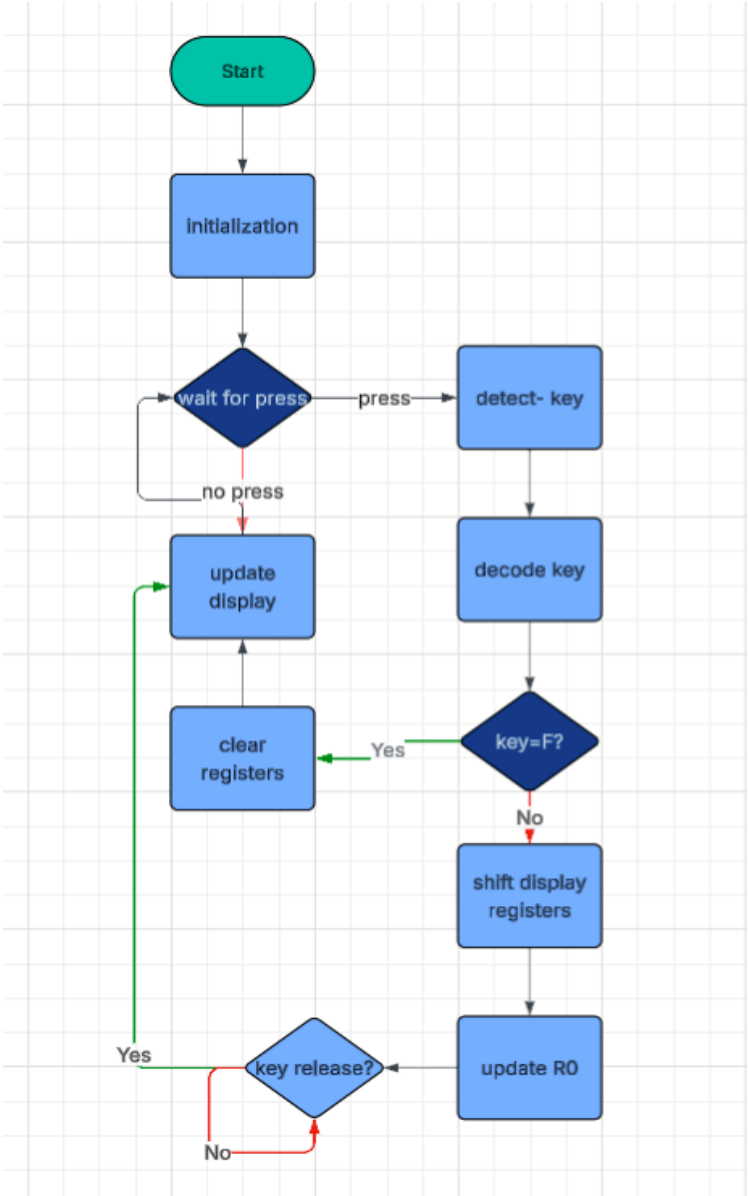
When the F key is pressed, the program clears all stored values and returns the calculator to its initial state. This includes resetting the display registers in bank 0 (R0–R3) so that the screen becomes blank and also clearing the operand registers in bank 2 where the two input numbers are stored. The operation register (R5) is also reset so that no previous operation remains active. After performing this full reset, the system is ready for the user to begin entering a new first operand. The clear function ensures that any incorrect input or completed calculation can be discarded quickly, allowing the user to start over without affecting the next operation.

4.8 summary of Program Modification

To support the 2-digit decimal adder, several modifications and additions are required on top of the original program from Part A. The main change is the introduction of register bank 2, which is used to store the BCD values of the first and second operands as well as the operation register. New lookup tables are added to convert the displayed seven-segment codes into numerical digits and to convert the BCD result back to segment patterns for the display. The main loop is expanded so it can detect the A, B, C, D and F keys and respond according to the current stage of the calculation (entering first operand → operation selection → entering second operand → performing the calculation). The display routine from Part A remains unchanged, as the only difference is which values are written to the display registers. Overall, these modifications reuse most of the original software

structure while adding the extra logic needed to store two operands, perform BCD addition, and display the final multi-digit result.

Appendix:



Structure overview flowchart