ASSEMBLY LANGUAGE PROJECT

CUSTOMIZABLE DIGITAL ALARM CLOCK

LEARNING OBJECTIVES:

1. Apply knowledge about the 8051 Microcontroller and the instruction set of 8051 in programming with the EdSim51 simulator.
2. Demonstrate the Switch input interfaces.
3. Demonstrate LED and LCD output interfaces.
4. Demonstrate DC motor as counter or timer.
5. Demonstrate a Digital Clock through the combination of I/O interface, pre-programmed instructions, and 8051 microcontrollers and its instruction set as simulated in EdSim51 Simulator.

INTRODUCTORY DISCUSSION

Nowadays, we are immersed in a field of silicon. Every car has dozens of chips, and every office has dozens. The introduction of microelectronics has apparent benefits in many areas. The impact of computing technology on the disabled, the transformation of certain types of tedious and dangerous manual labor, and breakthroughs in medical technology and weather forecasting are all examples of this. The simple answer is that microprocessors have revolutionized modern society. They have an impact on how we work and play, as well as how we travel and communicate. They provide incredible processing power at a negligible cost. A device that conducts routine calculations automatically is a deceptively simple description of a computer. Calculation underpins a wide range of actions that aren't traditionally thought of as mathematical. For example, walking across a room necessitates very complicated, albeit subconscious, calculations. Computers, too, have proven capable of addressing a wide range of issues, from balancing a checkbook to even walking across a room in the form of robot navigation systems.

An analog clock is a time display device that shows the time by revolving hands on a dial. On this clock, there are three hands. The longest hand indicates the seconds, while another long hand indicates the minute, and the short hand indicates the hours.

In comparison, a digital clock is a time display device that displays the time in numerals. Unlike analog clocks, which use hands to tell the time, this clock employs electronic displays to show the time numerically. This clock has additional features, the type of which varies from one model to the next.

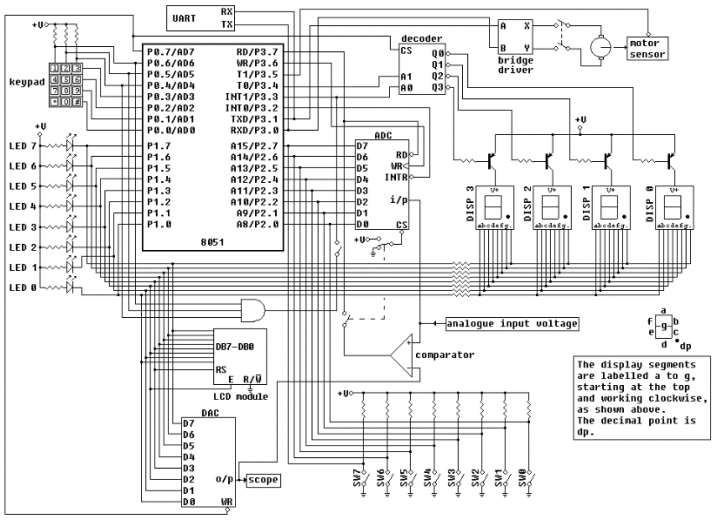


Figure 1: EdSim51 Logic Diagram

The Edsim51 simulator is used to build assembly language programs utilizing the 8051-instruction set. It allows the function of code debugging, with each line of code being examined for its effect on the internal memory and its peripherals. The simulator was connected to a variety of virtual I/O peripherals. Virtual peripherals that communicate with the 8051 Microcontroller are shown. LEDs, switches, seven-segment displays (SSD), a DOT matrix, a keypad, and a DC motor are just a few of the virtual peripherals available. Registers are used to store data temporarily. For arithmetic and logic operations, the accumulator ACC is required. During multiplication and division, register B is used as an additional hand for Acc. The general-purpose registers are R0 through R7. Any arithmetic operation cannot use these general-purpose registers as a destination. The data pointer register (DPTR) is a 16-bit register with two halves. The DPL is the lower byte of the DPTR, whereas the DPH is the upper byte. The Program Counter (PC) and the Instruction Pointer (IP) are both 16-bit registers.

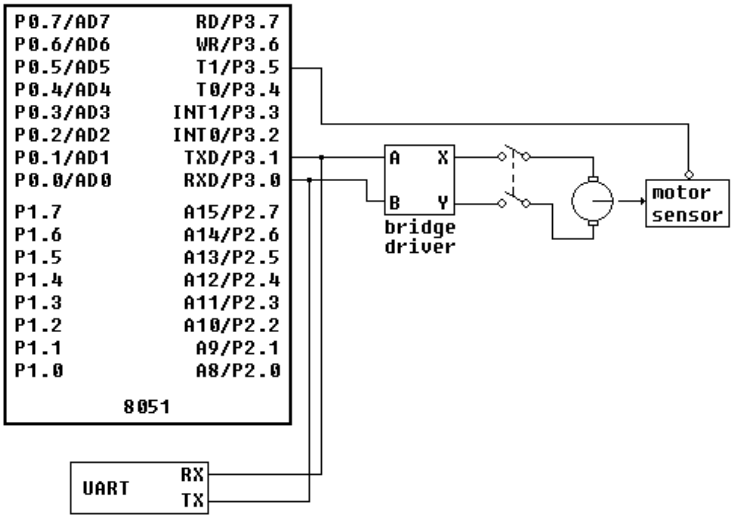


Figure 2: DC Motor Logic Diagram

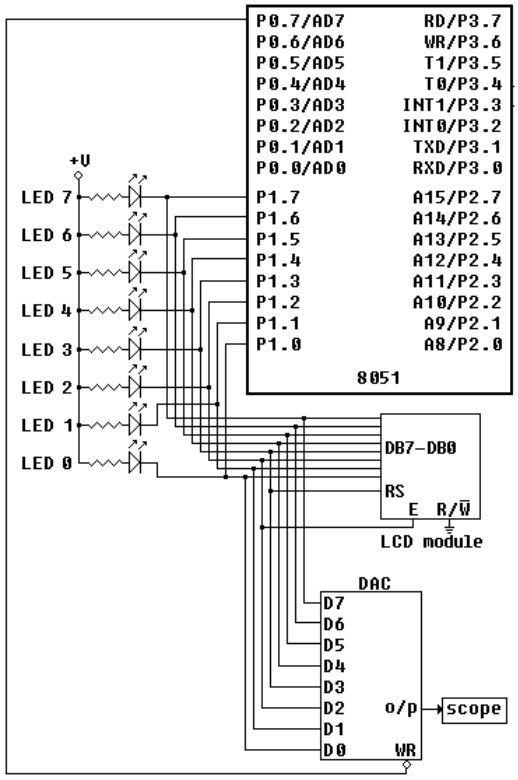


Figure 3: LCD module, Switch and LED banks logic diagram

The LCD module is a simulation that is interfaced to the 8051 in 4-bit mode. P1.7 through P1.4 are connected to DB7 through DB4, whereas P1.3 and P1.2 are attached to the register-select and enable pins, respectively. The read-write pin is grounded, indicating that the module can only be written to. As previously indicated, the module is interfaced in 4-bit mode by default. The lower four data bits (DB3 through DB0) are, however, also accessible (on P1.3 through P1.0). If the user wants to write to the module in 8-bit mode, the DI button on the peripheral panel should remap RS and E to other port pins. The ADC's outputs and the switch bank's outputs are applied to port 2.

As a result, it's important to remember that when the ADC is in operation, all switches in the switch bank should be open (in the simulator, the switches are grey when they are closed). When a switch is closed, no matter what the ADC tries to put on that line, it is held low since it is directly connected to the ground through the closed switch. Every turn, the motor sensor, connected to P3.5, drops low (in the simulator, whenever the motor shaft lines up with the sensor, the sensor changes from black to red, and P3.5 goes to logic 0). P3.5 is timer 1's external clock source. As a result, code may be written to count the motor's revolutions using timer 1. The motor's speed can be adjusted manually. The rev. counting programs will be more interesting as a result of this. P3.3 and P3.4 are used to select which LED interface should be enabled. These port pins are connected to a 2-to-4 line decoder, whose outputs are connected to the transistor bases that enable and disable the displays.

MATERIALS

The materials needed for the experiment are as follows: EdSim51 simulator and logic diagram, 8051 Microcontroller Instruction Set, and HD44780 LCD instruction set

Edsim51 simulator is a software that can simulate a basic 8051 Microcontroller functions. Its logic diagram includes LCD module, switches, LED bank, DC motor, and others, but only the indicated components will be used.

Available here: https://www.edsim51.com/index.html

In programming the microcontroller, the 8051 Microcontroller instruction set will be used in following program purposes: data transfer such as MOV, arithmetic operations such as ADD, INC and DEC, logical and boolean operations such as CLR, CPL, and SETB, and program branching to create simple loops includes the JMP, LJMP, CALL, LCALL, DJNZ, CJNE, and RET.

Available here: https://www.electronicshub.org/8051-microcontroller-instruction-set/

As part of the program, LCD will have to be configured. Thus, the HD44780 LCD instruction set will be used to configure the DDRAM Address for choosing which LCD line to display, the function set to determine the interface specifications, the cursor or display shift to set the cursor and shift movement of the display, the display control to enable display, cursor, and blinking properties of the LCD characters, and the entry mode set on how LCD reads from the DDRAM address.

Available here: https://www.sparkfun.com/datasheets/LCD/HD44780.pdf

PROCEDURE

PART I. - LCD INITIALIZATION

This part of the experiment will include setting the initial values stored in the memory addresses used in the program and programming a conditional LCD loop that will initialize the LCD to indicate the start of the digital clock program.

1) Launch the EdSim51DI Simulator
2) Click new and encode the following instruction at the IDE (integrated development environment).

```
ORG 0000H
MAIN:
        CALL Clear_RAM_0X
        CALL Clear_RAM_1X
        CALL Clear_RAM_2X
        CALL Clear_RAM_3X
        CALL Clear_RAM_4X
        CALL Clear_RAM_7X
        CALL Clear_Registers
        CALL Initial_RAM_LCD
        CALL Alarm_Time_Value
        CALL Initial_Current_Time_Value
        LJMP PROGRAM_START


;Switch Configurations
        ;SW 7 = Display Current Time Set Initially
        ;SW 6 = Display Alarm Time Set Initially
        ;SW 5 = Display Alarm Reminder/Phrase Set Initially
        ;SW 4
        ;SW 3
        ;SW 2 = Turn OFF Alarm
        ;SW 1 = For Starting the Clock Timer
        ;SW 0 = For Starting the Program

Clear_Registers:
        MOV R7, #0H ;For declaring the state of the clock (1:On, 0:Off)
        MOV R6, #0H ;For Temporary Memory of HH time value
        MOV R5, #0H ;For Temporary Memory of MM time value
        MOV R4, #0H ;For Temporary Memory of SS time value
        MOV R3, #0H ;For Starting the Program
 ;For LCD Line Indicator
        MOV R2, #0H ;For Short Loops
```

```
        MOV R1, #0H ;For LCD Reading the Current Memory Address
;For Decimal to 24HR Clock Cycle Loop
        MOV R0, #0H ;For Decimal to 24HR Clock Cycle Loop
        RET
```

NOTE: Code at any order

```
Clear_RAM_0X:
MOV 00H, #0
MOV 01H, #0
MOV 02H, #0
MOV 03H, #0
MOV 04H, #0
MOV 05H, #0
MOV 06H, #0
MOV 07H, #0
MOV 09H, #0
MOV 0AH, #0
MOV 0BH, #0
MOV 0CH, #0
MOV 0DH, #0
MOV 0EH, #0
MOV 0FH, #0
RET

Clear_RAM_1X:
MOV 10H, #0
MOV 11H, #0
MOV 12H, #0
MOV 13H, #0
MOV 14H, #0
MOV 15H, #0
MOV 16H, #0
MOV 17H, #0
MOV 18H, #0
MOV 19H, #0
MOV 1AH, #0
MOV 1BH, #0
MOV 1CH, #0
MOV 1DH, #0
MOV 1EH, #0
MOV 1FH, #0
RET

Clear_RAM_2X:
MOV 20H, #0
MOV 21H, #0
MOV 22H, #0
MOV 23H, #0
MOV 24H, #0
MOV 25H, #0
MOV 26H, #0
MOV 27H, #0
MOV 28H, #0
MOV 29H, #0
MOV 2AH, #0
MOV 2BH, #0
MOV 2CH, #0
MOV 2DH, #0
MOV 2EH, #0
MOV 2FH, #0
RET

Clear_RAM_3X:
MOV 30H, #0
MOV 31H, #0
MOV 32H, #0
MOV 33H, #0
MOV 34H, #0
MOV 35H, #0
MOV 36H, #0
MOV 37H, #0
MOV 38H, #0
MOV 39H, #0
MOV 3AH, #0
MOV 3BH, #0
MOV 3CH, #0
MOV 3DH, #0
MOV 3EH, #0
MOV 3FH, #0
RET

Clear_RAM_4X:
MOV 40H, #0
MOV 41H, #0
MOV 42H, #0
MOV 43H, #0
MOV 44H, #0
MOV 45H, #0
MOV 46H, #0
MOV 47H, #0
MOV 48H, #0
MOV 49H, #0
MOV 4AH, #0
MOV 4BH, #0
MOV 4CH, #0
MOV 4DH, #0
MOV 4EH, #0
MOV 4FH, #0
RET

Clear_RAM_7X:
MOV 70H, #0
MOV 71H, #0
MOV 72H, #0
MOV 73H, #0
MOV 74H, #0
MOV 75H, #0
MOV 76H, #0
MOV 77H, #0
MOV 78H, #0
MOV 79H, #0
MOV 7AH, #0
MOV 7BH, #0
MOV 7CH, #0
MOV 7DH, #0
MOV 7EH, #0
MOV 7FH, #0
RET
```

```
Initial_RAM_LCD:
        MOV 50H, #'D'
        MOV 51H, #'I'
        MOV 52H, #'G'
        MOV 53H, #'I'
        MOV 54H, #'T'
        MOV 55H, #'A'
        MOV 56H, #'L'
        MOV 57H, #' '
        MOV 58H, #'C'
        MOV 59H, #'L'
        MOV 5AH, #'O'
        MOV 5BH, #'C'
        MOV 5CH, #'K'
        MOV 5DH, #' '
        MOV 5EH, #' '
        MOV 5FH, #0
        MOV 60H, #'G'
        MOV 61H, #'R'
        MOV 62H, #'P'
        MOV 63H, #' '
        MOV 64H, #'3'
        MOV 65H, #' '
        MOV 66H, #'E'
        MOV 67H, #'C'
        MOV 68H, #'E'
        MOV 69H, #'A'
        MOV 6AH, #'1'
        MOV 6BH, #'0'
        MOV 6CH, #'5'
        MOV 6DH, #'L'
        MOV 6EH, #' '
        MOV 6FH, #0
        RET
```

```
Alarm_Time_Value: ;24 Hour Clock Cycle
;Input Decimal Value #XX, ex. #09, #00, #00
;Hours Value (HH)
        MOV 76H, #17
;Minutes Value (MM)
        MOV 77H, #00
;Seconds Value (SS)
        MOV 78H, #00
        RET
```

```
Initial_Current_Time_Value: ;24 Hour Clock Cycle
;Input Decimal Value #XX, ex. #08, #30, #00
;Hours Value (HH)
        MOV 79H, #16
;Minutes Value (MM)
```

```
                MOV 7AH, #59
;Seconds Value (SS)
                MOV 7BH, #50
                RET


Alarm_Reminders: ;15 Character Phrase/String
;Input each character inside the #' '
;Example: #'T',#'i',#'m',#'e',#' ',#'t',#'o',#' ',#'W',#'a',#'k',#'e',#' ',#'u',#'p'
                MOV 60H, #'A'
                MOV 61H, #'B'
                MOV 62H, #'C'
                MOV 63H, #'D'
                MOV 64H, #'E'
                MOV 65H, #'F'
                MOV 66H, #'G'
                MOV 67H, #'H'
                MOV 68H, #'I'
                MOV 69H, #'J'
                MOV 6AH, #'K'
                MOV 6BH, #'L'
                MOV 6CH, #'M'
                MOV 6DH, #'N'
                MOV 6EH, #'O'
                MOV 6FH, #0
                RET
PROGRAM_START:
                MOV A, #0
                MOV A, P2
                MOV R3, #0
                CPL A
                MOV R3, A
                MOV R2, #100
                DJNZ R2, $
                CJNE R3, #1, PROGRAM_START
                LCALL LCD_New
                LJMP SWITCHES
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

3) To save the program. Click Save.
4) Answer Q1

NOTE: The program will not work yet since some labels are indicated without being specified which will be done in the following steps. After encoding the initial values of registers and memory addresses, the LCD instructions comes next.

5) Continue encoding the program with the codes below.

```
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

;LCD SEGMENT

LCD_New:
        LCALL LCD_Initialize
        LCALL LCD_Function
        LCALL LCD_Display_Control
        LCALL LCD_Line_1
        LCALL LCD_Entry_Mode
        LCALL LCD_Data_Read_1
        RET

LCD_Initialize:
        CLR P1.3
        RET
```

NOTE: Code at any order

```
LCD_Function:
        CLR P1.7
        CLR P1.6
        SETB P1.5
        CLR P1.4

        SETB P1.2
        CLR P1.2
        CALL LCD_Delay
        SETB P1.2
        CLR P1.2

        SETB P1.7
        CLR P1.6
        CLR P1.5
        CLR P1.4

        SETB P1.2
        CLR P1.2

        CALL LCD_Delay
        RET

LCD_Display_Control:
        CLR P1.7
        CLR P1.6
        CLR P1.5
        CLR P1.4

        SETB P1.2
        CLR P1.2

        SETB P1.7
        SETB P1.6
        SETB P1.5
        SETB P1.4

        SETB P1.2
        CLR P1.2

        CALL LCD_Delay
        RET

LCD_Entry_Mode:
        CLR P1.7
        CLR P1.6
        CLR P1.5
        CLR P1.4

        SETB P1.2
        CLR P1.2

        CLR P1.7
        SETB P1.6
        SETB P1.5
        CLR P1.4

        SETB P1.2
        CLR P1.2

        CALL LCD_Delay
        RET

LCD_Data_Read_1:
        SETB P1.3
        MOV R1, #50H
        MOV R3, #0
        CALL LCD_Loop
        RET

LCD_Line_1:
        SETB P1.7
        CLR P1.6
        CLR P1.5
        CLR P1.4
        SETB P1.2
        CLR P1.2

        CLR P1.7
        CLR P1.6
```

```
        CLR P1.5
        CLR P1.4

        SETB P1.2
        CLR P1.2

        CALL LCD_Delay

        RET

LCD_Data_Read_2:
        CALL LCD_Line_2
        SETB P1.3
        MOV R3, #1
        MOV R1, #60H
        JMP LCD_Loop

LCD_Line_2:
        CLR P1.3

        SETB P1.7
        SETB P1.6
        CLR P1.5
        CLR P1.4

        SETB P1.2
        CLR P1.2

        CLR P1.7
        CLR P1.6
        CLR P1.5
        CLR P1.4

        SETB P1.2
        CLR P1.2

        CALL LCD_Delay

        RET

LCD_Loop:
        MOV A, @R1
        CALL LCD_Send_Character
        INC R1
        CJNE A, #0, LCD_Loop
        CJNE R3, #1, LCD_Data_Read_2
        MOV R3, #0
        RET

LCD_Send_Character:
        MOV C, ACC.7
        MOV P1.7, C
        MOV C, ACC.6
        MOV P1.6, C
        MOV C, ACC.5
        MOV P1.5, C
        MOV C, ACC.4
        MOV P1.4, C

        SETB P1.2
        CLR P1.2

        MOV C, ACC.3
        MOV P1.7, C
        MOV C, ACC.2
        MOV P1.6, C
        MOV C, ACC.1
        MOV P1.5, C
        MOV C, ACC.0
        MOV P1.4, C

        SETB P1.2
        CLR P1.2

        CALL LCD_Delay

LCD_Delay:
        MOV R2, #25
        DJNZ R2, $
        RET
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

6) Assemle the program to check for errors. Click Assm.

NOTE: Make the LJMP SWITCH a comment for now or create a temporary SWITCH label to run without an error

7) To observe the output. Click Run.

8) To speed up the output. Click Update Frequency (Update Freq.). For this part of the program, choose 100.
9) Press switch 0 (SW 0), the right most switch labelled “0”.
10) Answer Q2, Q3, Q4, and Q5.
11) To pause the execution. Click Pause.
12) To stop the program execution. Click RST.

PART II. - SWITCH MODE SELECTION

This part of the experiment includes codes that utilizes the memory addresses, LCD output display and switches input interfaces to program specific mode configuration for different cases of pressed switches. The LCD will continuously display current mode selected together with the initial inputs at the memory addresses.

1) Save the current program as Part 1.
2) Encode the following instruction at the IDE (integrated development environment).

```
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;SWITCH SEGMENT
SWITCHES:
        CALL Switch_Delay
        CALL Switch_Delay
        CALL Switch_Delay
        JMP Mode_1

Switch_Delay:
        MOV B, #10 ;Adjust Switch Delay
        MOV 7CH, B
        MOV 7DH, B
        MOV 7EH, B
        MOV 7FH, B
        CALL Switch_Delay_Loop
        RET

Switch_Delay_Loop:
        DEC 7FH
        MOV R0, 7FH
        CJNE R0, #0, Switch_Delay_Loop
        MOV 7FH, B
        DEC 7EH
        MOV R0, 7EH
        CJNE R0, #0, Switch_Delay_Loop

        MOV 7EH, B
        DEC 7DH
        MOV R0, 7DH
        CJNE R0, #0, Switch_Delay_Loop

        MOV 7DH, B
        DEC 7CH
        MOV R0, 7CH
        CJNE R0, #0, Switch_Delay_Loop
        MOV 7CH, B

        RET

SW_7_Current_Time:
        MOV A, #0
        MOV C, P2.7
        CPL C
        MOV ACC.0, C
        CLR C
        CJNE A, #1, No_SW_7_Current_Time
        JMP Mode_1

No_SW_7_Current_Time:
```

```
        RET

SW_6_Alarm_Time:
        MOV A, #0
        MOV C, P2.6
        CPL C
        MOV ACC.0, C
        CLR C
        CJNE A, #1, No_SW_6_Alarm_Time
        JMP Mode_2

No_SW_6_Alarm_Time:
        RET

SW_5_Alarm_Reminder:
        MOV A, #0
        MOV C, P2.5
        CPL C
        MOV ACC.0, C
        CLR C
        CJNE A, #1, No_SW_5_Alarm_Reminder
        JMP Mode_3

No_SW_5_Alarm_Reminder:
        RET
MODE_0:
        CALL Current_Time_Line_1
        CALL Current_Time_Update
        CALL LCD_New
        RET

Mode_1:
        CALL Current_Time_Line_1
        CALL Current_Time_Line_2
        CALL LCD_New
        CALL SW_6_Alarm_Time
        CALL SW_5_Alarm_Reminder
        LCALL SW_1_Start_Timer
        CALL Switch_Delay
        JMP Mode_1

Mode_2:
        CALL Alarm_Time_Line_1
        CALL Alarm_Time_Line_2
        CALL LCD_New
        CALL SW_7_Current_Time
        CALL SW_5_Alarm_Reminder
        LCALL SW_1_Start_Timer
        CALL Switch_Delay
        JMP Mode_2

Mode_3:
        CALL Alarm_Reminder_Line_1
        LCALL Alarm_Reminders
        CALL LCD_New
        CALL SW_7_Current_Time
        CALL SW_6_Alarm_Time
        LCALL SW_1_Start_Timer
        CALL Switch_Delay
        JMP Mode_3

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

NOTE: Uncomment the LJMP SWITCHES from the part I. Assembling the program will result in errors for now since labels used were still not yet specified which will be done in the succeeding steps.

3) Answer Q1, and Q2.

4) Continue encoding the following instructions for the program.

NOTE: Codes can be configured in any order since most subprograms are configured to be called and return once all instructions are done inside the subprogram but, following the left-to-right column is still preferred in the order of commands.

```
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
Current_Time_Line_1:
        MOV 50H, #'C'
        MOV 51H, #'U'
        MOV 52H, #'R'
        MOV 53H, #'R'
        MOV 54H, #'E'
        MOV 55H, #'N'
        MOV 56H, #'T'
        MOV 57H, #' '
        MOV 58H, #'T'
        MOV 59H, #'I'
        MOV 5AH, #'M'
        MOV 5BH, #'E'
        MOV 5CH, #58
        MOV 5DH, #' '
        MOV 5EH, #' '
        MOV 5FH, #0
        RET
```

```
Current_Time_Line_2:
        MOV 60H, #'2'
        MOV 61H, #'4'
        MOV 62H, #'H'
        MOV 63H, #'R'
        MOV 64H, #127
        MOV 67H, #58
        MOV 6AH, #58
        MOV 6DH, #' '
        MOV 6EH, #' '
        MOV 6FH, #0

        MOV R4, 79H
        MOV R5, 7AH
        MOV R6, 7BH

        LCALL Decimal_24_Clock_Cycle

        MOV A, 70H
        MOV 65H, A
        MOV A, 71H
        MOV 66H, A

        MOV A, 72H
        MOV 68H, A
        MOV A, 73H
        MOV 69H, A

        MOV A, 74H
        MOV 6BH, A
        MOV A, 75H
        MOV 6CH, A

        RET
```

```
Alarm_Time_Line_1:
        MOV 50H, #'A'
        MOV 51H, #'L'
        MOV 52H, #'A'
        MOV 53H, #'R'
        MOV 54H, #'M'
        MOV 55H, #' '
        MOV 56H, #'T'
        MOV 57H, #'I'
        MOV 58H, #'M'
        MOV 59H, #'E'
        MOV 5AH, #58
        MOV 5BH, #' '
        MOV 5CH, #' '
        MOV 5DH, #' '
        MOV 5EH, #' '
        MOV 5FH, #0
        RET
```

```
Alarm_Time_Line_2:
        MOV 60H, #'2'
        MOV 61H, #'4'
        MOV 62H, #'H'
        MOV 63H, #'R'
        MOV 64H, #127
        MOV 67H, #58
        MOV 6AH, #58
        MOV 6DH, #' '
        MOV 6EH, #' '
        MOV 6FH, #0

        MOV R4, 76H
        MOV R5, 77H
        MOV R6, 78H

        LCALL Decimal_24_Clock_Cycle

        MOV A, 70H
        MOV 65H, A
        MOV A, 71H
        MOV 66H, A

        MOV A, 72H
        MOV 68H, A
        MOV A, 73H
        MOV 69H, A

        MOV A, 74H
        MOV 6BH, A
        MOV A, 75H
        MOV 6CH, A

        RET
```

```
Alarm_Reminder_Line_1:
        MOV 50H, #'A'
        MOV 51H, #'L'
        MOV 52H, #'A'
        MOV 53H, #'R'
        MOV 54H, #'M'
        MOV 55H, #' '
        MOV 56H, #'R'
        MOV 57H, #'E'
        MOV 58H, #'M'
        MOV 59H, #'I'
        MOV 5AH, #'N'
        MOV 5BH, #'D'
        MOV 5CH, #'E'
        MOV 5DH, #'R'
        MOV 5EH, #58
        MOV 5FH, #0
        RET
```

```
Current_Time_Update:
        MOV 60H, #' '
        MOV 61H, #' '
        MOV 62H, #58
        MOV 63H, #' '
        MOV 64H, #' '
        MOV 65H, #58
        MOV 66H, #' '
        MOV 67H, #' '
        MOV 68H, #' '
        MOV 69H, #' '
        MOV 6AH, #' '
        MOV 6BH, #' '
        MOV 6CH, #' '
        MOV 6DH, #' '
        MOV 6EH, #' '
        MOV 6FH, #0

        MOV R4, 79H
        MOV R5, 7AH
        MOV R6, 7BH
        LCALL Decimal_24_Clock_Cycle

        MOV A, 70H
        MOV 60H, A
        MOV A, 71H
        MOV 61H, A
        MOV A, 72H
        MOV 63H, A
        MOV A, 73H
        MOV 64H, A

        MOV A, 74H
        MOV 66H, A
        MOV A, 75H
        MOV 67H, A

        RET
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Decimal_24_Clock_Cycle:
        CALL S_Loop
        CALL M_Loop
        CALL H_Loop
        RET

Decrement_By_10:
        CPL A
        ADD A, #10
        CPL A
        INC R1
        JMP T0

S_Loop:
        MOV A, R6
        MOV R1, #0
        CALL T0
        MOV 75H, R0

        MOV A, R1
        CALL T0
        MOV 74H, R0

        RET

M_Loop:
        MOV A, R5
        MOV R1, #0
        CALL T0
        MOV 73H, R0

        MOV A, R1
        CALL T0
        MOV 72H, R0

        RET

H_Loop:
        MOV A, R4
        MOV R1, #0
        CALL T0
        MOV 71H, R0

        MOV A, R1
        CALL T0
        MOV 70H, R0

        RET

T0: CJNE A, #0, T1
    MOV R0, #'0'
    RET

T1: CJNE A, #1, T2
    MOV R0, #'1'
    RET

T2: CJNE A, #2, T3
    MOV R0, #'2'
    RET

T3: CJNE A, #3, T4
    MOV R0, #'3'
    RET

T4: CJNE A, #4, T5
    MOV R0, #'4'
    RET

T5: CJNE A, #5, T6
    MOV R0, #'5'
    RET

T6: CJNE A, #6, T7
    MOV R0, #'6'
    RET

T7: CJNE A, #7, T8
    MOV R0, #'7'
```

```
RET          T8: CJNE A, #8, T9          RET          T9: CJNE A, #9, Decrement_By_10          MOV R0, #'9'
                 MOV R0, #'8'                                                                   RET
```

```
SW_1_Start_Timer:
        CJNE R7, #0, $
        MOV A, #0
        MOV C, P2.1
        CPL C
        MOV ACC.0, C
        CLR C
        CJNE A, #1, No_SW_1_Start_Timer
        MOV R7, #1
        LJMP TIMER

No_SW_1_Start_Timer:
        RET
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

NOTE: Create another temporary label named TIMER to assemble the program without error.

5) Run the program.
6) Speed up the program to 10000 Update Frequency.
7) Answer Q3, Q4, and Q5.
8) Save the program as Part II

PART III. - DC MOTOR TIMER

This part of the experiment continues after observing the configured values and proceeds to enabling the clock timer to cycle in a 24-hour or 12-hour clock cycle. This part also makes use of loops to increment the value of seconds which increments minutes after 60 seconds and also increments the hours after 60 minutes. For every change in time, the program will display the current time in the LCD and compares it to the set alarm time.

1) Continue encoding the program with the following instructions or commands.

```
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;TIMER SEGMENT
TIMER:
        JMP Timer_Start

Increment:
        INC 7BH
        MOV A, 7BH
        CJNE A, #60, No_Increment
        MOV 7BH, #0
        INC 7AH
        MOV A, 7AH
        CJNE A, #60, No_Increment
        MOV 7AH, #0
```

```
        INC 79H
        MOV A, 79H
        CJNE A, #24, No_Increment
        ;Set Clock Cycle (#24 = 24Hr, #12 =
12Hr)
        MOV 79H, #0
        RET

No_Increment:
        RET

Time_Compare:
        CLR A
        MOV A, 7BH
        CJNE A, 78H, Alarm_Off
        CLR A

        MOV A, 7AH
        CJNE A, 77H, Alarm_Off
        CLR A

        MOV A, 79H
        CJNE A, 76H, Alarm_Off
        CLR A

        LJMP Alarm_On

Alarm_Off:
        RET

Alarm_On:
        LCALL ALARM_MODE
        LJMP PROGRAM_RESET

Timer_Count:
        MOV B, #10 ; Adjust Clock Delay
        MOV 7CH, B
        MOV 7DH, B
        MOV 7EH, B
        MOV 7FH, B

Timer_Start:
        SETB C
        MOV P3.0, C
        CPL C
        MOV P3.1, C
        CLR C

        MOV TMOD, #50H
        SETB TR1

Timer_Loop:
        MOV A, TL1
        DEC 7FH
        MOV R0, 7FH
        CJNE R0, #0, Timer_Loop

        MOV 7FH, B
        DEC 7EH
        MOV R0, 7EH
        CJNE R0, #0, Timer_Loop

        MOV 7EH, B
        DEC 7DH
        MOV R0, 7DH

        CJNE R0, #0, Timer_Loop

        MOV 7DH, B
        DEC 7CH
        MOV R0, 7CH
        CJNE R0, #0, Timer_Loop
        MOV 7CH, B

Timer_Reset:
        CLR A
        CLR TR1

        CALL Increment
        LCALL MODE_0
        CALL Time_Compare

        MOV TL1, #0
        SETB TR1

        JMP Timer_Loop
```

;~~~~~~

NOTE: Create a temporary label named ALARM_MODE and PROGRAM_RESET to run the program without error.

2) Run the program
3) Answer Q1, Q2, and Q3.
4) Save the program as part III

PART IV. - DIGITAL CLOCK ALARM THROUGH LED's

The last part of the experiment is to set the result when the current time is equal to the alarm time. This alarm system could be represented using a sound however, since no speaker are available in the simulator, the program uses the LED bank to indicate that the alarm is on when it is blinking.

1) Continue encoding the remaining parts of the program code.

```
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;ALARM
ALARM_MODE:
        MOV 50H, #' '
        MOV 51H, #' '
        MOV 52H, #58
        MOV 53H, #' '
        MOV 54H, #' '
        MOV 55H, #58
        MOV 56H, #' '
        MOV 57H, #' '
        MOV 58H, #' '
        MOV 59H, #' '
        MOV 5AH, #' '
        MOV 5BH, #' '
        MOV 5CH, #' '
        MOV 5DH, #' '
        MOV 5EH, #' '
        MOV 5FH, #0

        MOV R4, 76H
        MOV R5, 77H
        MOV R6, 78H

        LCALL Decimal_24_Clock_Cycle

        MOV A, 70H
        MOV 50H, A
        MOV A, 71H
        MOV 51H, A

        MOV A, 72H
        MOV 53H, A
        MOV A, 73H
        MOV 54H, A

        MOV A, 74H
        MOV 56H, A
        MOV A, 75H
        MOV 57H, A

        LCALL Alarm_Reminders

SW_2:
        LCALL LCD_New

        CALL LED_Blink
        MOV A, #0
        MOV C, P2.2
        CPL C
        MOV ACC.0, C
        CLR C
        CJNE A, #1, SW_2
        RET

LED_Blink:
        MOV A, #0
        MOV P1, A
        CALL LED_Delay
        CPL A
        MOV P1, A
        CALL LED_Delay

LED_Delay:
LD2:MOV R2, #75
LD1:MOV R1, #75
        DJNZ R1, $
        DJNZ R2, LD1
        RET

PROGRAM_RESET:
        LCALL Initial_RAM_LCD
        LCALL LCD_New
        END
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

NOTE: Replace the temporary labels created earlier with the continuation of the program code.

2) Answer Q1, Q2, and Q3.
3) Save the program as DOE - Customizable Digital Alarm Clock

ANSWER SHEETS RESULTS

PART ONE - LCD INITIALIZATION

Q1. What is the purpose of switch 0 in the start of the program?

Switch 0 is used as a condition for the program to proceed to the next instruction. When SW0 is pressed, the program proceeds to the LCD_New label otherwise it will only loop at the PROGRAM_START label until it is pressed.

Q2. What is the output of the program? Illustrate using a picture or a drawing with an explanation.

The output of the current program is displayed in the LCD. The characters shown in each cell was from the initialization of values stored in the memory addresses. The output is “DIGITAL CLOCK” at the first line and “HELLO WORLD!” at the second line.

Q3. What happens when switch 0 is unpressed while the program is displaying the initial display?

Nothing happens. The program will still display the initial display and proceed to the succeeding instructions. Unpressing switch 0 was not configured to affect the program after LCD was initialized.

Q4. What is the function of LCD_Line_1 and LCD_Line_2 labels?

The LCD_Line_1 label sets the DDRAM address to its starting position which is 00H thus, whenever we call LCD_New, the displays start again from the initial position or cell which is the left-most cell of the first line. The LCD_Line_2 label also set DDRAM address, however, it sets the value to 40H which the left-most cell of the second line.

Q5. What is the importance of including a value of “0” for the end of each line?

The program is configured so that whenever the current value being read is 0, the display stops and return back to the instruction that the LCD_New was called. For both line 1 and 2, a value 0 is stored to the memory address it is designated at.

PART TWO - SWITCH MODE SELECTION

Q1. What switches were configured for a specific purpose and what purpose?

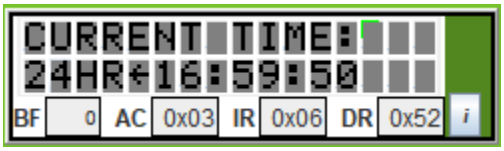
Switches 7 (SW 7), switch 6 (SW 6), and switch 5 (SW 5) were configured as a condition that the next instruction will proceed to different subroutines. These subprograms are labeled Mode_1, Mode_2, and Mode_3. Switch 7 can be used to jump in the Mode_1 subprogram while switch 6 for Mode_2 and switch 5 for Mode_3. However, when we are already in the selected mode, pressing the same switch that directs to that subprogram will have no effect. This loop will continue for as long as the SW_1_Start_Timer do nothing.

Q2. What is the value count of switch delay?

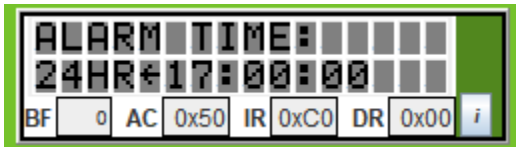
The switch delay can be computed as the fourth power of the value stored in the B register. This value is the minimum count since steps and instruction bytes increments for all the instructions used even in data transfer, arithmetic operation, or logical operations. By calculation, the switch delay is at least 10000 steps.

Q3. What is the output of the program? Illustrate the outputs at different modes with explanation.

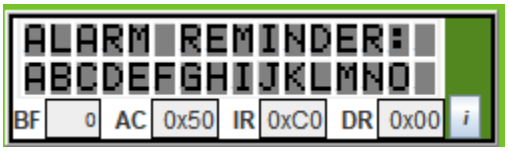
The outputs of the program are in three different modes for the current program. The current time mode, the alarm time mode, and the alarm reminder mode. The values displayed for each mode was initially set from the part I of the procedure.



The initial mode selected is the current time. This mode displays a mode title at the LCD line 1 while the time values at the line 2 together with an indicator that it is a 24-hr cycle of time. At this mode, we can select the alarm time by pressing switch 6 or the alarm reminder by pressing switch 5.



The second mode is the alarm time. This mode displays a mode title at the LCD line 1 while the time values at the line 2 together with an indicator that it is a 24-hr cycle of time. At this mode, we can select again the current time by pressing switch 7 or the alarm reminder by pressing switch 5.



The third mode is the alarm reminder. This mode displays a mode title at the LCD line 1 while the reminder phrase consisting of up to 15 characters at the LCD line 2. At this mode, we can reselect again the current time by pressing switch 7 or the alarm time by pressing switch 6.

Q4. Explain the function of the subprogram Decimal_24_Clock_Cycle.

The subprogram converts any two-digit decimal value to its corresponding two LCD character data value. The initial two-digit decimal values from part I were stored at memory addresses 76H to 7BH, three for the alarm time and the other for the current time. The data is called and stored to registers R4, R5, or R6 depending on the use. R4 is temporary data storage for the hour value while R5 for the minute and R6 for the seconds. The subprogram compares each value from 0 to 9 and stores a specific characater data to R0. If the value is not among the following, the Decrement_By_10 subprogram is called to subtract 10 to the current value by complimenting and addition operation of binary bits. If the data is decremented by 10, R3 is incremented which will be used as the second digit value. This looping process continues until all three decimal values are converted in to its corresponding two-digit LCD characters which will be temporarily stored from 70H to 75H memory addresses.

Q5. What happens when switch 1 is pressed?

Mode selection will stop and that the program will proceed to the subprogram labelled TIMER. The commands have not been encoded yet for this part but, it indicates the start of the DC motor that will be used as a clock timer of the Digital Clock program.

PART THREE - DC MOTOR TIMER

Q1. What is the use of the DC Motor?

The DC motor is used as a timer in the digital clock program. Included in the loop is to increment the seconds value of current time and compares the current time to the alarm time.

Q2. What is the output of the current program? Illustrate and explain.

The output of the program changes for every cycle of the DC motor. The time is incremented until it reaches the alarm time set. Afterwards the time will no longer increments because as configured in the program, it now proceed to the alarm reminder which is not yet part of the program code.

Q3. How does it perform like a digital clock?

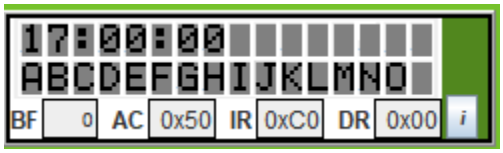
Just like any digital clock, the program displays the current time which we can observe as it updates every change in the seconds. The incrementing subprogram allows the time to only be in the 0 to 59 in range for minutes and seconds while 24 for the hours. The hour cycle can be adjusted to 12-hr clock cycle depending on the value.

PART FOUR - DIGITAL CLOCK ALARM THROUGH LED's

Q1. What happens when the current time is equal to the alarm time set?

The program will proceed to the ALARM_MODE subprogram which indicates that the alarm is on and that it will now display the alarm reminder. At this point, it will continue to loop in displaying the alarm reminder and another indicator of the alarm which is the LEDs blinking periodically until it is turned off.

Q2. What is the output of the program? Illustrate.



The output can both be observed in the LED bank and the LCD. The LED blinks periodically which signifies that the alarm is on. The LCD displays the alarm time and the alarm reminder. Once the alarm is turned off, the LCD displays the initial introductory display of the program.

Q3. How can the alarm be turned off?

The program is configured so that when the switch 2 (SW 2) is pressed, the blinking and displaying of the alarm indicators will stop. After the alarm is turned off, the program ends.

DISCUSSION AND INTERPRETATION OF RESULTS

In this activity, it explains the applications and understanding of the 8051 Microcontroller and edsim51 simulator. The simulator allows everyone to program in assembly language using the 8051's set of instructions. Debugging codes and viewing output of each line to the external peripheral and internal memory are all part of it. The simulator was connected to a variety of virtual I/O peripherals. LEDs, seven-segment display, switches, keypad, and other components are included. The scripts were run and constructed after input, resulting in the output on the simulator's upper left side. There are carried out line by line by the simulator. The instructions were used to apply the edsim51 simulator's basic operating directives CLR, MOV, CPL, CLR were only the few of the basic commands provided. To deal with the carious I/O interfaces, the 8051-instruction set was utilized and understood. The 8051-microcontroller speaks a machine language called opcodes. These opcodes specify which functions should be carried out. The bytes size is also specified, which indicate how much code memory is used. By utilizing such commands, the students were able to display different outputs of working alarm, character display and current time through LED interface. The program creates a timer or a counter system employing the use of dc motor so that in can increase a value after a defined amount of time with from minutes to hours that will be displayed through the LCD interface. By using the different instructions of SW, it can be used to alter between modes and values for a different functionality ranging from current date, time, and reminders and the characters displayed in each cell were created by initializing variables stored in memory addresses. By running the digital alarm clock, the alarm is turned on when the LED blinks frequently and then the alarm time and the alarm reminder will be displayed on the LCD. When it is turned off, the LCD will display then the program's initial introductory display.

CONCLUSION

- Understand and apply commands of the 8051 Microcontroller instruction set.
- Understand, perform, and configure LCD module according to the HD44780U instruction set.
- The program has successfully integrated the principles of different components such as LED bank, LCD module, Switch bank, and DC motor present in the EdSim51 Simulator that represents a basic 8051 Microcontroller. LED bank
- The program has properly demonstrated the application and function of a digital alarm clock.
- Modification in the program can be made so that it can be a normal counter or timer such as day/month/year count.