



**MIDDLE EAST TECHNICAL UNIVERSITY NORTH CYPRUS  
CAMPUS**

**COMPUTER ENGINEERING PROGRAM**

**CNG 336 INTRODUCTION TO EMBEDDED SYSTEMS  
DEVELOPMENT**

**LAB MODULE #1**

**Fatma Erem Aksoy – 2315075**

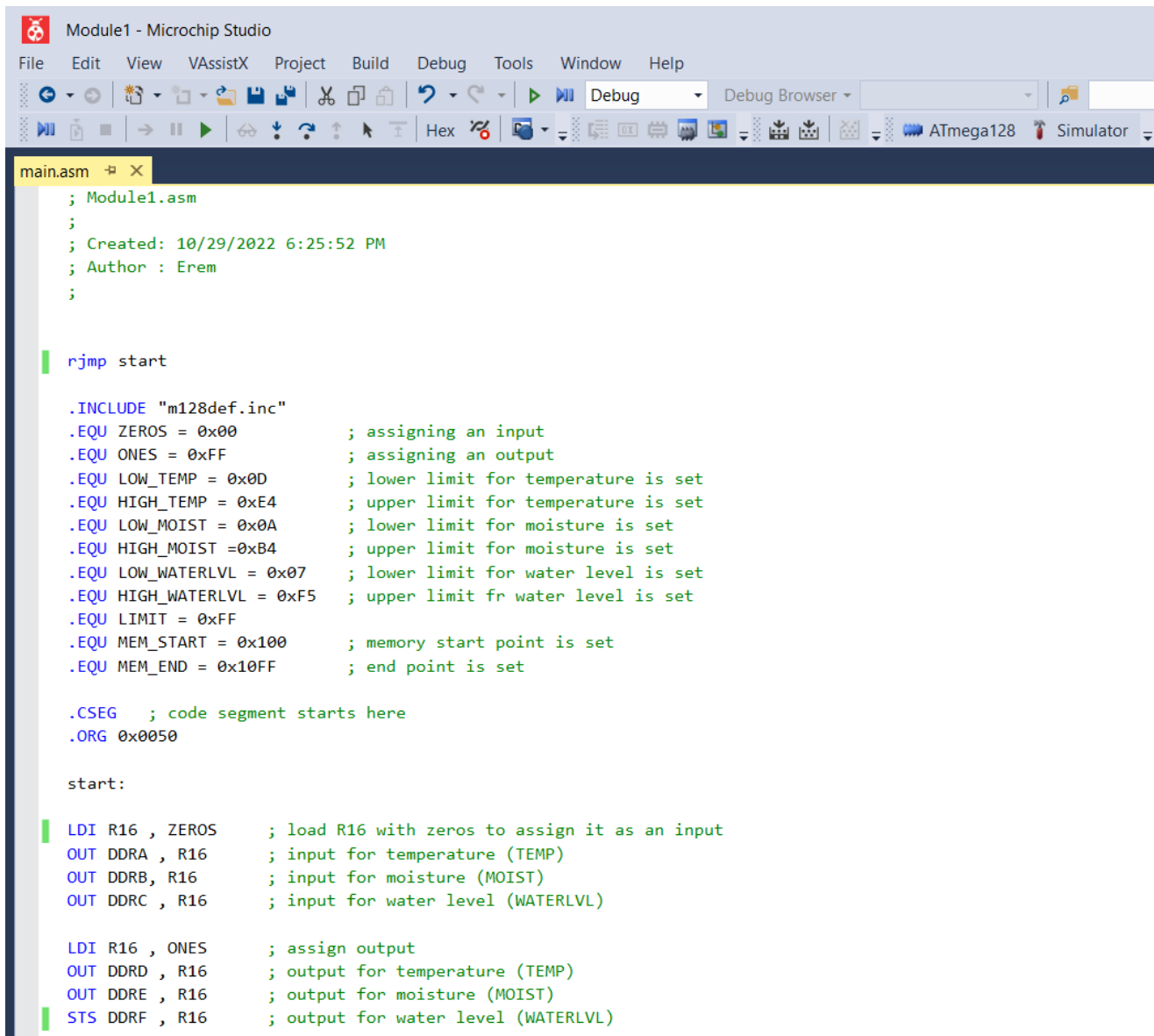
**Ece Erseven - 2385383**

# OBJECTIVE

The main purpose of this laboratory exercise is writing and executing a short assembly code segment for the given task by using the knowledge we have from the course learnings. The code is executed and debugged carefully on Microchip Studio, and Proteus is used for the schematics design of the experiment.

## 1.2.3 IMPLEMENTATION

a)



```
Module1 - Microchip Studio
File Edit View VAssistX Project Build Debug Tools Window Help
Debug Debug Browser
Hex ATmega128 Simulator

main.asm
; Module1.asm
;
; Created: 10/29/2022 6:25:52 PM
; Author : Erem
;

rjmp start

.INCLUDE "m128def.inc"
.EQU ZEROS = 0x00      ; assigning an input
.EQU ONES = 0xFF      ; assigning an output
.EQU LOW_TEMP = 0x0D   ; lower limit for temperature is set
.EQU HIGH_TEMP = 0xE4  ; upper limit for temperature is set
.EQU LOW_MOIST = 0x0A   ; lower limit for moisture is set
.EQU HIGH_MOIST = 0xB4  ; upper limit for moisture is set
.EQU LOW_WATERLVL = 0x07 ; lower limit for water level is set
.EQU HIGH_WATERLVL = 0xF5 ; upper limit for water level is set
.EQU LIMIT = 0xFF
.EQU MEM_START = 0x100  ; memory start point is set
.EQU MEM_END = 0x10FF  ; end point is set

.CSEG ; code segment starts here
.ORG 0x0050

start:

LDI R16 , ZEROS      ; load R16 with zeros to assign it as an input
OUT DDRA , R16       ; input for temperature (TEMP)
OUT DDRB , R16       ; input for moisture (MOIST)
OUT DDRC , R16       ; input for water level (WATERLVL)

LDI R16 , ONES       ; assign output
OUT DDRD , R16       ; output for temperature (TEMP)
OUT DDRE , R16       ; output for moisture (MOIST)
STS DDRF , R16       ; output for water level (WATERLVL)
```

```

; Request - PUSH-BUTTON
LDI R16, 0x02      ; load R16 with 0x02
STS DDRG, R16      ; set DDRG to R16
LDI R17, 0x01      ; load R17 with 0x01

; Request ON/OFF Condition Check
REQUEST_CHECK:
    LDS R18, PING      ; load R18 with PING for the push button
    ANDI R18, 0x01     ; multiply R18 with 0x01 for checking the push button value
    SUB R18, R17        ; if their value is equal, then the result will be 0 and the program will jump to acknowledge
    BRNE REQUEST_CHECK ; continue checking the request till it is 1
    RJMP ACKNOWLEDGE

ACKNOWLEDGE:
    STS PORTG, R16      ; storing R16 in PORTG
    IN R17, PINA        ; getting TEMP input
    IN R18, PINB        ; getting MOIST input
    IN R19, PINC        ; getting WATERLVL input

; Capture TEMP
LDI R20, HIGH_TEMP    ; take the upper limit value for TEMP to compare
CP R20, R17            ; comparison for the upper bound
BRLT CONTROL_TEMP     ; if the value is lower than the upper limit, go to control_temp to check the lower bound as well
RJMP INVALID_TEMP     ; the TEMP value is not in the range, go to invalid_temp

INVALID_TEMP:
    LDI R21, ONES      ; loading R21 with ones
    OUT PORTD, R21     ; making the port D read to display ones as the value is out of range
    RJMP MOIST         ; go to moist to get the next value from the user

CONTROL_TEMP:
    LDI R21, LOW_TEMP   ; take the lower limit value for TEMP to compare
    CP R17, R21         ; comparison for the lower bound
    BRGE VALID_TEMP    ; if the value is lower than the upper limit, it means the value is in the required range
    RJMP INVALID_TEMP   ; jump to invalid_temp as the value is out of range

VALID_TEMP:
    OUT PORTD, R17      ; display TEMP value

; Capture MOIST
MOIST:
    LDI R22, HIGH_MOIST ; take the upper limit value for MOIST to compare
    CP R22, R18          ; comparison for the upper bound
    BRLT CONTROL_MOIST  ; if the value is lower than the upper limit, go to control_moist to check the lower bound too
    RJMP INVALID_MOIST  ; the MOIST value is not in the range, go to invalid_moist

INVALID_MOIST:
    LDI R23, ONES      ; loading R23 with ones
    OUT PORTE, R23     ; making the port E read to display ones as the value is out of range
    RJMP WATERLVL      ; go to waterlvl to get the next value from the user

CONTROL_MOIST:
    LDI R23, LOW_MOIST  ; take the lower limit value for MOIST to compare
    CP R18, R23         ; comparison for the lower bound
    BRGE VALID_MOIST    ; if the value is lower than the upper limit, it means the value is in the required range
    RJMP INVALID_MOIST  ; jump to invalid_moist as the value is out of range

VALID_MOIST:
    OUT PORTE, R18      ; display MOIST value

; Capture WATERLVL
WATERLVL:
    LDI R24, HIGH_WATERLVL ; take the upper limit value for WATERLVL to compare
    CP R24, R19           ; comparison for the upper bound
    BRLT CONTROL_WATERLVL ; if the value is lower than the upper limit, go to control_waterlvl to check the lower bound too
    RJMP INVALID_WATERLVL ; the WATERLVL value is not in the range, go to invalid_waterlvl

INVALID_WATERLVL:
    LDI R25, ONES      ; loading R25 with ones
    STS PORTF, R25     ; making the port F read to display ones as the value is out of range
    RJMP ACKNOWLEDGE   ; go to acknowledge to check if there is another request

```

```

CONTROL_WATERLVL:
    LDI R25 , LOW_WATERLVL ; take the lower limit value for WATERLVL to compare
    CP R19, R25             ; comparison for the lower bound
    BRGE VALID_WATERLVL    ; if the value is lower than the upper limit, it means the value is in the required range
    RJMP INVALID_WATERLVL  ; jump to invalid_waterlvl as the value is out of range

VALID_WATERLVL:
    STS PORTF , R19         ; display WATERLVL value
    RJMP ACKNOWLEDGE       ; go to ACKNOWLEDGE to check if there are more inputs requested
RJMP start

```

**b) i-ii)** When all the three inputs are within expected range;

Temperature = 45 = 0x2D (R17)

Moisture = 28 = 0x1C (R18)

Water level = 60 = 0x3C (R19)

The screenshot displays the Microchip Studio IDE in Advanced Mode. The main window shows the assembly code for the program, which includes initialization of registers, setting up I/O ports, and a loop that checks for button presses and updates the output ports based on sensor inputs. The Processor Status window on the left shows the current state of the registers, with R17 (Temperature) at 0x2D, R18 (Moisture) at 0x1C, and R19 (Water level) at 0x3C. The I/O window on the right shows the status of the I/O ports, with PORTG (0x02) being the active output. The Registers window at the bottom shows the current values of all registers, including R00 through R31.

Name	Value
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00
R16	0x02
R17	0x2D
R18	0x1C
R19	0x3C
R20	0xE4
R21	0x0D
R22	0xB4
R23	0x0A
R24	0xF5
R25	0x07
R26	0x00
R27	0x00
R28	0x00
R29	0x00

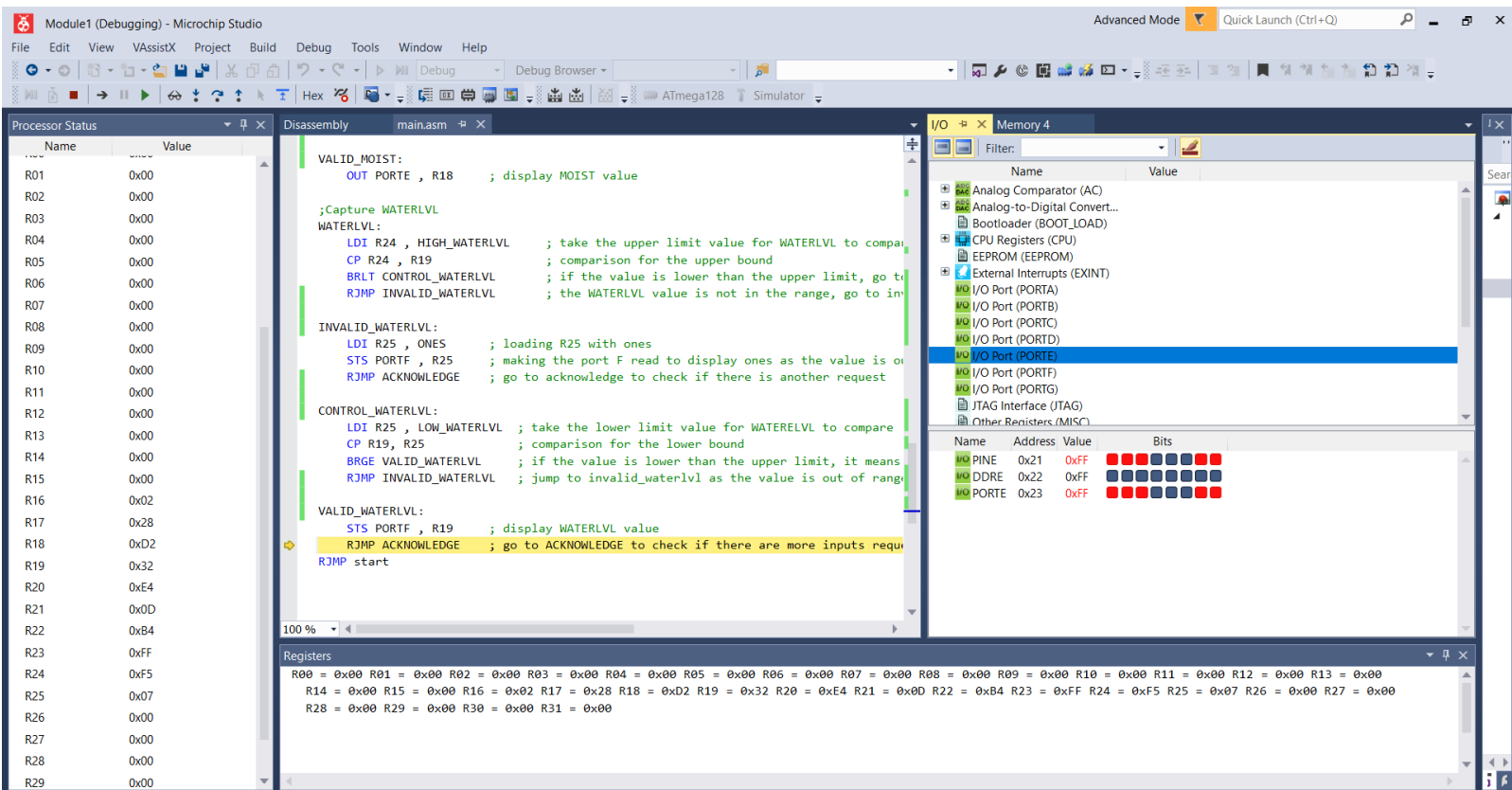
*The LEDs (in PORTD, E and F) will be turned on according to the input values as all the three inputs are within the specified range, and PORTG will be on as well as the acknowledge is set.*

- iii) When temperature and water level values are within expected range but moisture is out of range;

Temperature = 40 = 0x28 (R17)

Moisture = 210 = 0xD2 (R18)

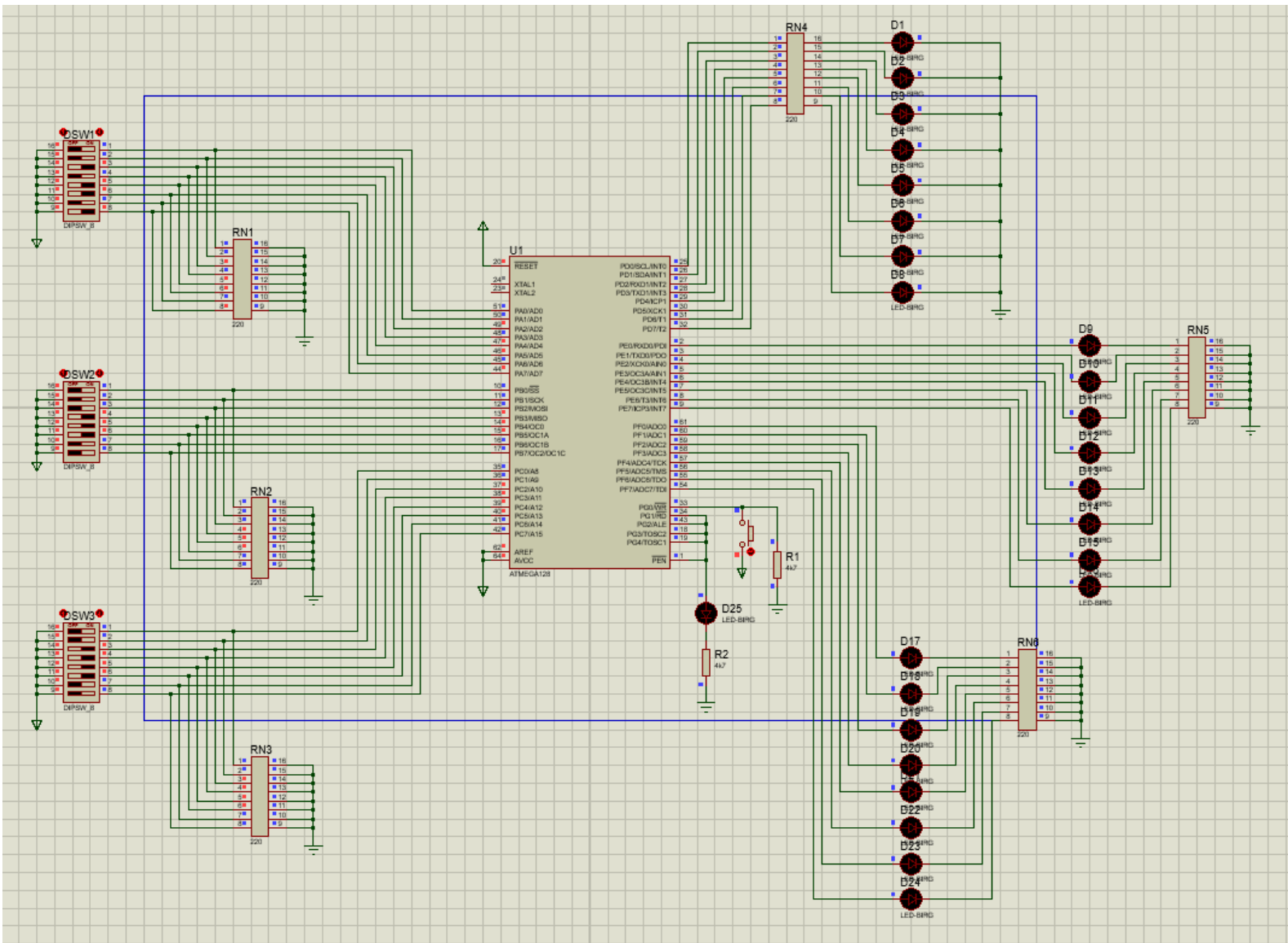
Water level = 50 = 0x32 (R19)



*The LEDs (in PORTD and F) will be turned on according to the input values as all their inputs are within the specified range, but all the LEDs will be turned on in PORTE as the moisture value is out of range. PORTG will be on as well as the acknowledge is set.*

- iv) The lines with the OUT instructions don't affect the machine state at all because it only stores data from the specified register in the Register File to I/O Space which are represented with the ddr word.

### c) Proteus Design Screenshots

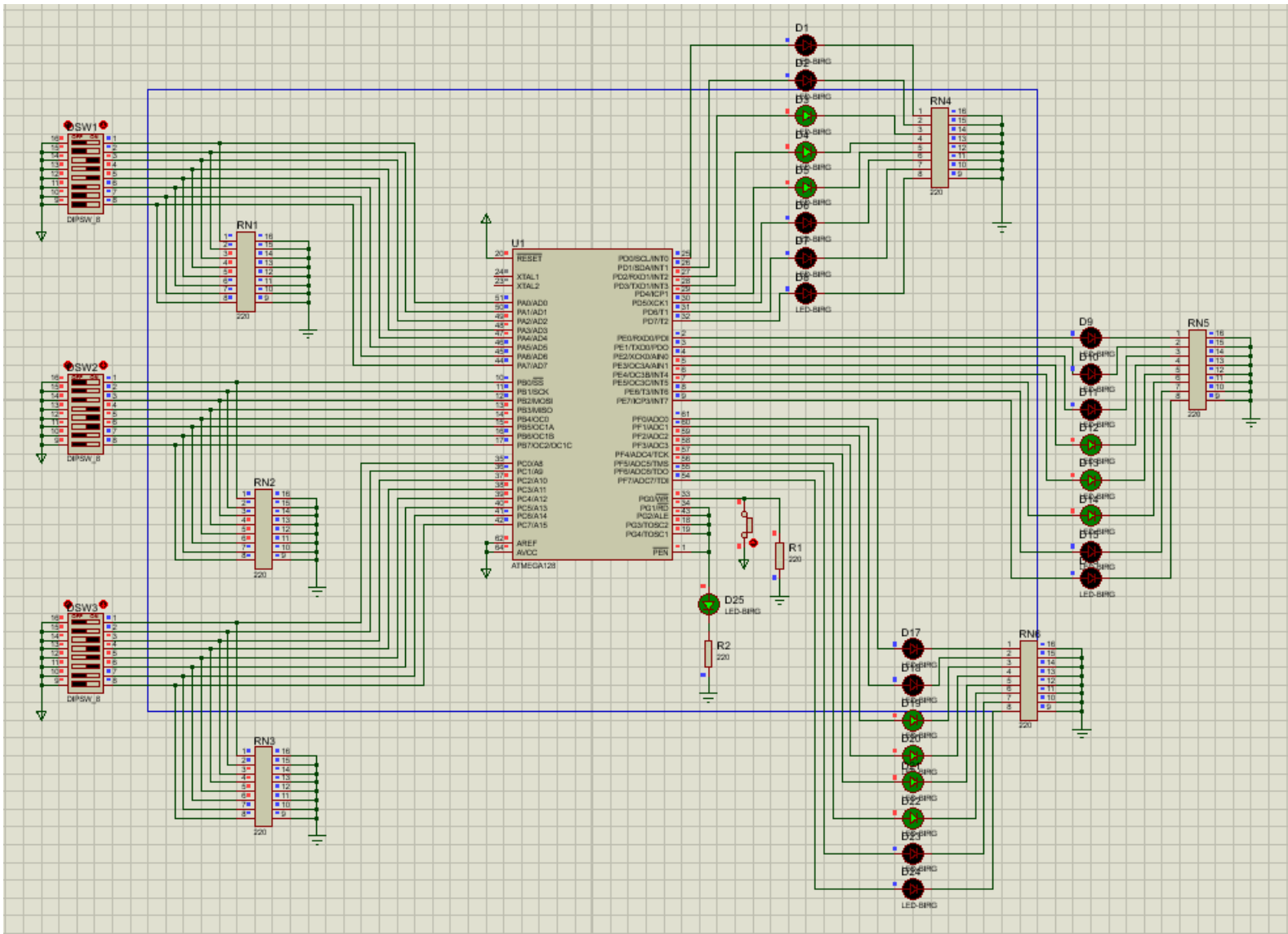


All the three inputs are within the expected range but as the push button is off, no LEDs will be on in this scenario.

Temperature = 45 = 0x2D = 00101101

Moisture = 28 = 0x1C = 00011100

Water level = 60 = 0x3C = 00111100



In this scenario, the inputs are all within the expected range and the request (push button) is on, so the LEDs for outputs will be turned on according to their corresponding values. The values are:

Temperature = 56 = 0x38 = 00111000 (PORTD output)

Moisture = 28 = 0x1C = 00011100 (PORTE output)

Water level = 60 = 0x3C = 00111100 (PORTF output)

Water level = 50 = 0x32 = 00111100 (PORTF output)



#### **1.2.4** What is the difference between RAM and ROM? Why is there a need for both a Flash Memory and an EEPROM?

*- Random Access Memory (RAM) is a form of computer memory that can be read and changed in any order, typically used to store working data and machine code. RAM is used to store the programs and data being used by the CPU in real-time. The data on the random access memory can be read, written, and erased any number of times. RAM is a hardware element where the data being currently used is stored. It is a volatile memory. Read Only Memory (ROM) is a type of memory where the data has been prerecorded. Data stored in ROM is retained even after the computer is turned off ie, non-volatile.*

*- There are fundamental differences in the operating characteristics of EEPROM and Flash which makes them appropriate for different use cases. EEPROM memory is based on NOR gates. Flash is based on NAND gates. NOR-based memory is much faster than NAND-based, and it's much more expensive. EEPROM isn't really designed to be written a lot so it's mostly used to store bits of data that don't change much. On the other hand, NAND is not only cheaper, it has a completely different method of erasing/writing memory. NOR does this one byte at a time, and NAND does this a page at a time, while erasure of NAND can only happen on a whole block of pages at a time. NAND is designed to be written a lot more than NOR.*

#### **1.2.5** What is the function of the control unit? How does the control unit get the instruction that it must execute? How does it know the number of bytes in an instruction?

*- The control unit fetches (gets) the instruction from memory and decodes the instruction (decides what it means) and directs that the necessary data be moved from memory to the arithmetic/logic unit.*

*- It doesn't know how many bytes in an instruction in advance. The simple way to know it is just to read one byte, decode it and then determine if it's a complete instruction. If not read another byte, decode it if necessary and then determine if a complete instruction has been read. If not continue reading/decoding bytes until the complete instruction is read.*

## **CONCLUSION**

With this lab practice, a short assembly code segment is written and executed to solve a simple problem by doing minimum number of computations, and also moving data to and from the memory, and microcontroller digital I/O ports (parallel I/O interface) were studied with its necessary details. To debug and test the code by checking each register and pins/ports, Microchip Studio is used, and Proteus is used to design and simulate the entire system. After all these are tested, the observations for the experiment are shown with the details including code and the design with different inputs for several scenarios.