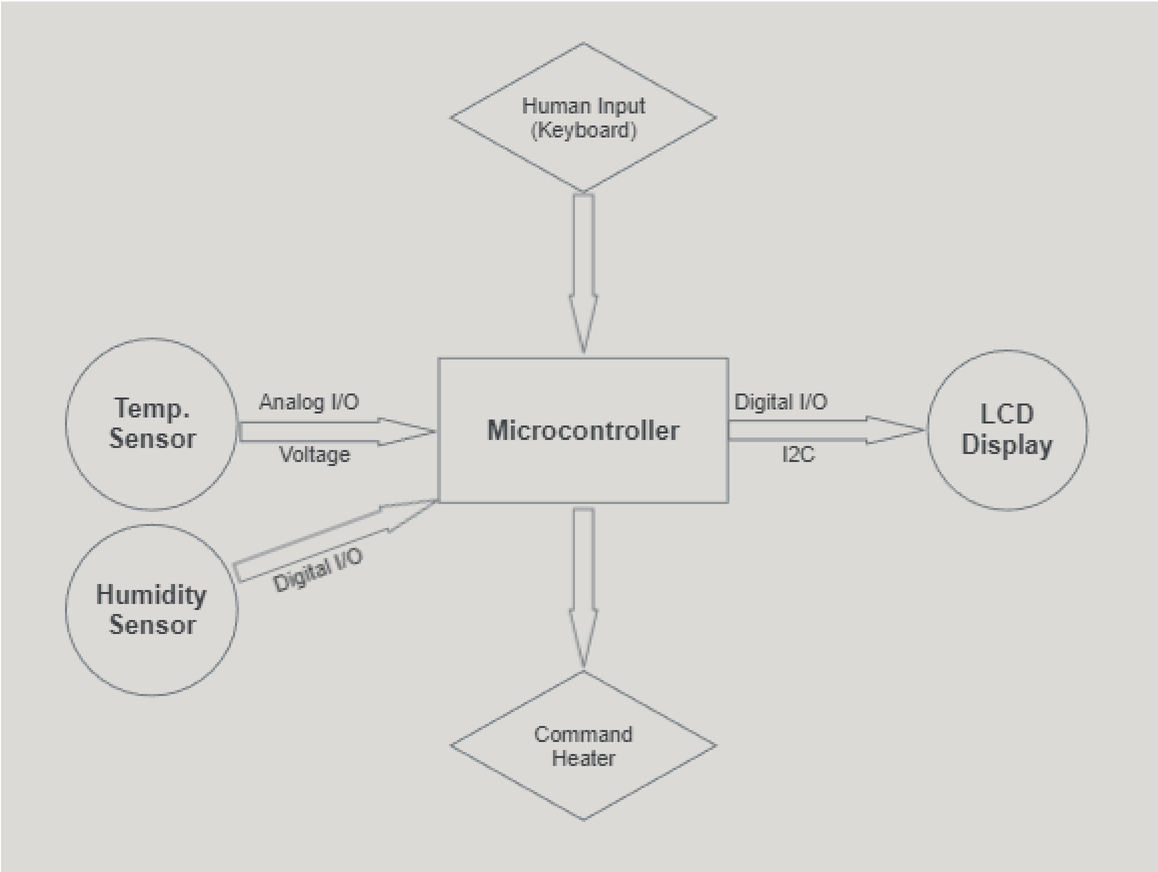
Microcontrollers Project

Thermostat

Axinte Octavian-Constantin

Table of Contents

1 Block Diagram and Working Principle

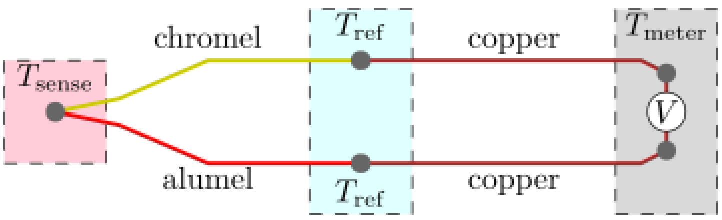


Firstly, the temperature sensor measures the temperature constantly and sends the information to the microcontroller. The microcontroller displays it on the LCD. If the user wants a greater or lower temperature, he will introduce it as input for the microcontroller and it will take a decision based on the current temperature and wanted one. The response will be sent to the heater.

2 Types of temperature sensors

2.1 Thermocouples

This sensor consists of two dissimilar metal wires, joined at one end, and connected to a thermocouple thermometer or other thermocouple-capable device at the other end. This causes a Seebeck Effect. The Seebeck Effect is a phenomenon in which a temperature difference of two dissimilar conductors produces a voltage difference between the two substances. It is this voltage difference that can be measured and used to calculate the temperature.



$$\nabla V = -S(T)\nabla T,$$

They offer **lower accuracy**, but they do work across **wider temperature ranges** than any of the other temperature sensors.

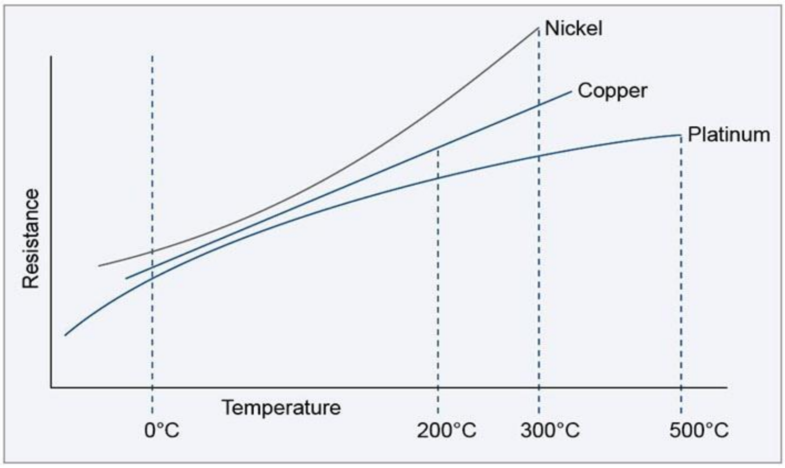
These sensors are **highly durable and cost-effective** and are important because they can work in **many different** applications - from an industrial usage thermocouple to a regular thermocouple found on utilities and regular appliances.

2.2 Resistance Temperature Detector (RTD)

An RTD (Resistance Temperature Detector) is a sensor whose resistance changes as its temperature changes. The resistance increases as the temperature of the sensor increases. An RTD is a resistor with well-defined resistance vs. temperature characteristics. Platinum is the most common and accurate material used to make RTDs.

They provide **the greatest accuracy** and are generally **the most expensive**. Resistance temperature detectors are best when high levels of accuracy are needed.

Two wire sensors are typically used in applications where accuracy is not critical. The two wire configuration allows for the simplest measurement technique, but suffers from an inherent inaccuracy due to the resistance of the sensor leads. For greater accuracy three wire sensors or four wire sensors are better.

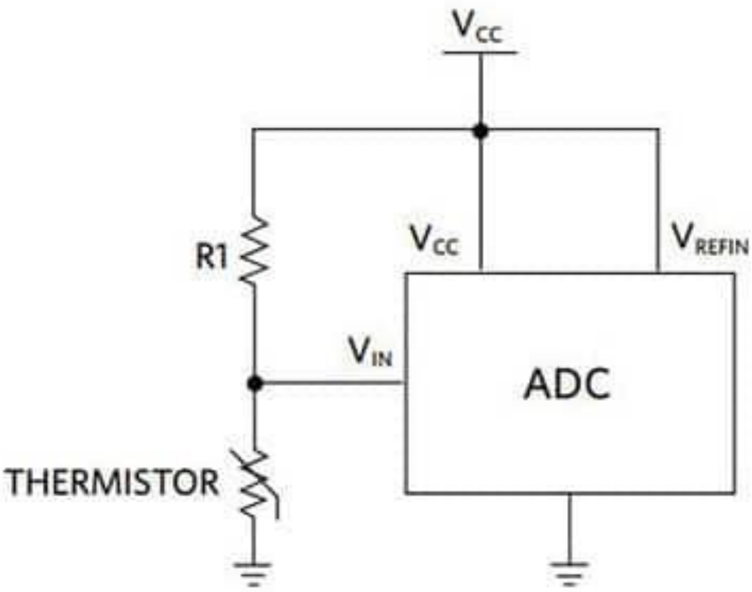


RTD Resistance versus Temperature

2.3 Thermistor

Thermistors are similar to RTDs in that temperature changes cause measurable resistance changes. Thermistors are usually made from a polymer or ceramic material. In most cases, thermistors are **cheaper** but are also **less accurate than RTDs**.

The NTC (Negative Temperature Coefficient) thermistor is the most commonly used thermistor for temperature measurement application. An NTC thermistor's resistance decreases as the temperature increases. Thermistors have a non-linear temperature resistance relationship. This requires a significant correction to interpret the data correctly. A common approach of using a thermistor, shown in the figure below, is where a thermistor and a fixed value resistor form a voltage divider with an output that is digitized by an ADC.



2.4 Integrated Circuit Temperature Sensor

An IC Temperature Sensor is a two terminal integrated circuit temperature transducer that produces an output current proportional to absolute temperature. The sensor package is small with a low thermal mass and a fast response time. It is the most **linear**, **small size** and **inexpensive**, but it requires **power supply** and it is **self-heating**, usually supports a **maximum of 200 Celsius** degrees.

3 Integrated Circuit Temperature Sensors

Name	Temperature Range [°C]	Price (Lei)	Availability: Mouser, Digi-key, Farnell	Supply Voltage [V]	Accuracy [°C]
TMP116*	-55 <-> 125	15-18	Yes Yes No	1.9 – 5.5	+/- 0.3
BME280*	-40 <-> 85	30 - 33	Yes Yes No	1.7 – 3.6	+/- 0.5
Winsen ZS05*	-20 <-> 65	-	No No No	3.3 – 5.5	+/- 1
TMP275AIDGKT	-40 <-> 125	22	No No No	2.7 – 5.5	+/- 1
BMP180*	0<->65	15	Cleste.ro	1.8 – 3.6	+/- 0.5 (at 25)
LMT87	-50 <-> 150	7.42	Yes Yes No	2.7 – 5.5	+/- 0.4

*TMP116 Interface I2C and SMBus. + EPROM

*Bosch BME280 Interface I2C and SPI + Humidity and Pressure

*BMP180 – Interface I2C

*ZS05 – Interface I2C

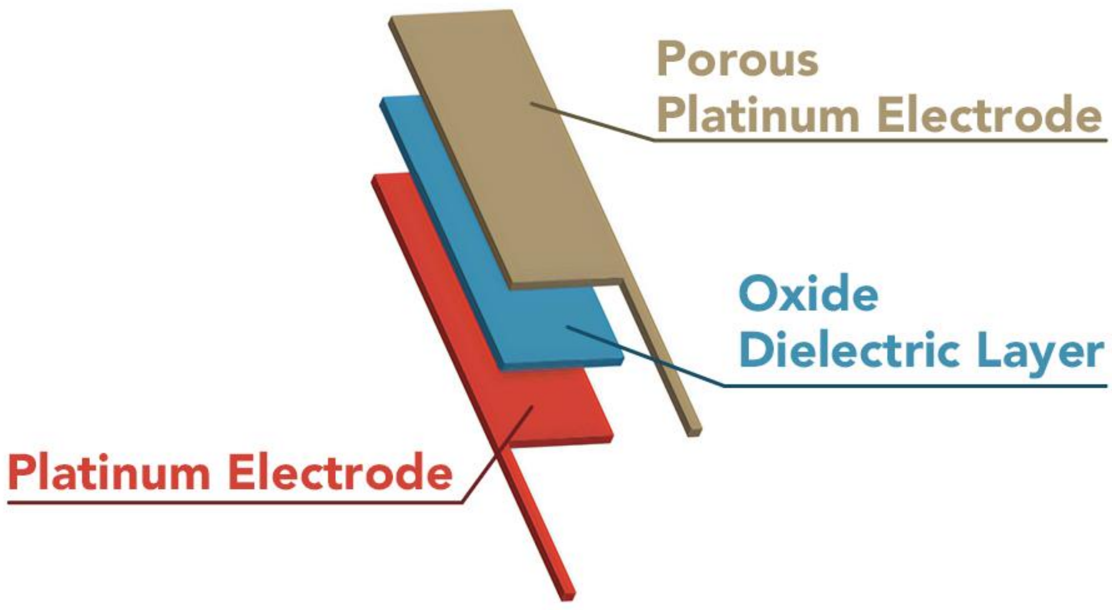
*LMT87 – Analogue output – Voltage output inversely proportional to temperature

I chose **LMT87** since it provides all the necessary conditions at the best price point.

4 Types of Humidity Sensors

4.1 Capacitive Humidity Sensors

It is estimated that 75% of humidity sensors follow the capacitive technique. These humidity sensor types rely on electrical capacitance to provide the user with a humidity value.



Capacitive relative humidity (RH) sensors consist of two metal electrode layers between a dielectric (non-conductive) material, typically a polymer film with a dielectric constant of around 2-15. The dielectric film inside the capacitive humidity sensor attracts and absorbs moisture from the surrounding air. Once the moisture contacts the electrodes, a voltage change occurs.

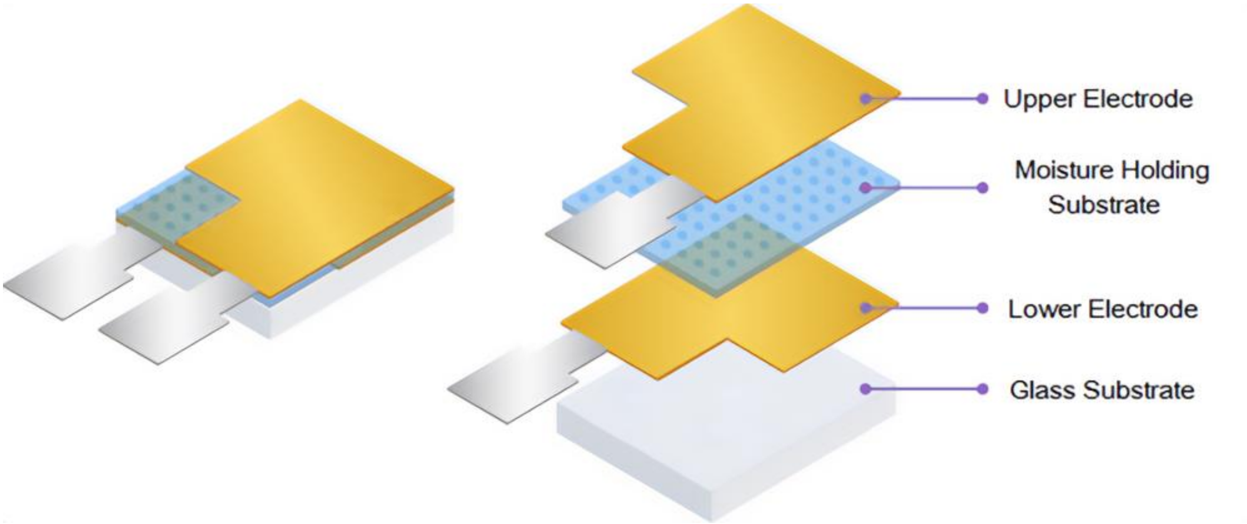
In capacitive humidity sensors, there is a direct relationship between the RH (relative humidity) of the surrounding air, the amount of moisture in the dielectric material, and the capacitance (dielectric constant) of the humidity sensor. The change in the dielectric constant is directly proportional to the RH, therefore, by measuring the dielectric constant, the RH can be calculated.

Pros: wide measurement range, almost linear output voltage, low cost, little maintenance

Cons: bad accuracy at low RH, limited distance between the uC and the sensor.

4.2 Resistive Humidity Sensors

Resistive humidity sensors, also known as electrical conductivity sensors, measure the change in resistivity between two electrodes inside a humidity probe (connected to the sensor) to establish relative humidity.



They have a similar principle to capacitive sensors; an electrical change is measured, producing an RH value. However, resistive humidity sensors use a moisture-absorbing (hygroscopic) material, so their operation principle is slightly different. The output voltage has an inverse exponential relationship to RH. As more water vapor is absorbed, the resistivity decreases due to an increase in the non-metallic conductivity material's conductivity.

Pros: low cost, small footprint, highly interchangeable, big distance between sensor and uC

Cons: sensitive to contaminants, bad accuracy at low RH

4.3 Thermal Conductivity Humidity Sensors

These types of sensors measure the absolute humidity (AH) of the surrounding air/environment by calculating the difference between thermal conductivity in dry air vs humid air. A thermal conductivity humidity sensor consists of two matched negative temperature coefficient (NTC) thermistor elements, suspended by thin wires, in a bridge circuit. One thermistor is located in an exposed chamber via several ventilation holes, exposing it to the surrounding environment. The second is hermetically encapsulated in dry nitrogen, and located in a different section within the humidity sensor.

An electrical circuit passes a current between the two thermistors, resulting in the thermistors self-heating; resistive heating increases the sensor's temperature. When one of the thermistors is exposed to humid air, the conductivity changes. The difference in resistance between the two thermistors (bridge circuit) is directly proportional to absolute humidity.

Pros: resistant, can be used in high-temperature, high-corrosive environments, great resolution

Cons: exposure to certain gases can affect humidity readings

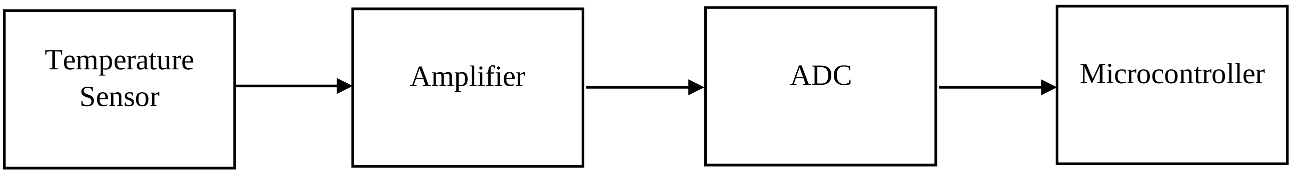
5 Capacitive Humidity Sensors:

Name	Rel. Humidity Range [°C]	Price (Lei)	Availability: Mouser, Digi-key, Farnell	Supply Voltage [V]	Accuracy [°C]
SHT21*	0 <-> 100	36 - 40	Yes Yes No	1.9 – 5.5	+/- 2
BME280*	0<-> 100	30 - 33	Yes Yes No	1.7 – 3.6	+/- 3
Winsen ZS05*	0 <-> 100	-	No No No	3.3 – 5.5	+/- 5
SHT40I-AD1B-R2	0 <-> 100	14.76	Yes Yes Yes	2.3 – 5.5	+/- 2
HPP845E031R4	0<-> 100	35-36	Yes Yes No	1.5 – 3.6	+/- 3

I chose SHT40I-AD1B-R2 since it is way cheaper than the rest, it is easy to find and has good accuracy. It is a digital output sensor, I2C communication channel.

6 Sensor to Microcontroller Connection

6.1 Block Diagram



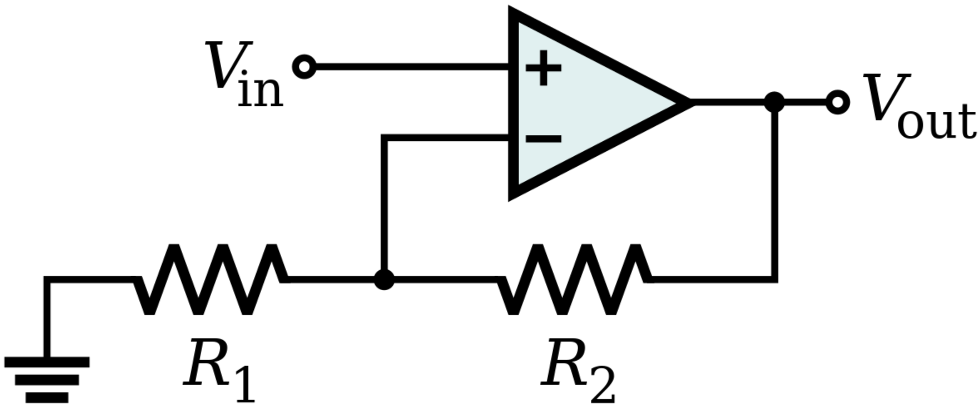
6.2 Temperature Sensor

The analogue temperature sensor has a voltage output depending on the temperature. I chose the LM35 temperature sensor for simulation due to Proteus' component availability. It is quite similar to the LMT87 I chose for implementation, although the main difference is that the output voltage is directly proportional to the temperature.

The temperature sensor LM35 has at the output 10mv/ ºC. For example, if the LM35 is measuring a temperature of 20°C, its output voltage will be 200 mV (20°C x 10 mV/°C = 200 mV). If the temperature changes to 21°C, the output voltage will change to 210 mV (21°C x 10 mV/°C = 210 mV), which represents a linear change in the output voltage in response to the change in temperature.

6.3 Amplifier

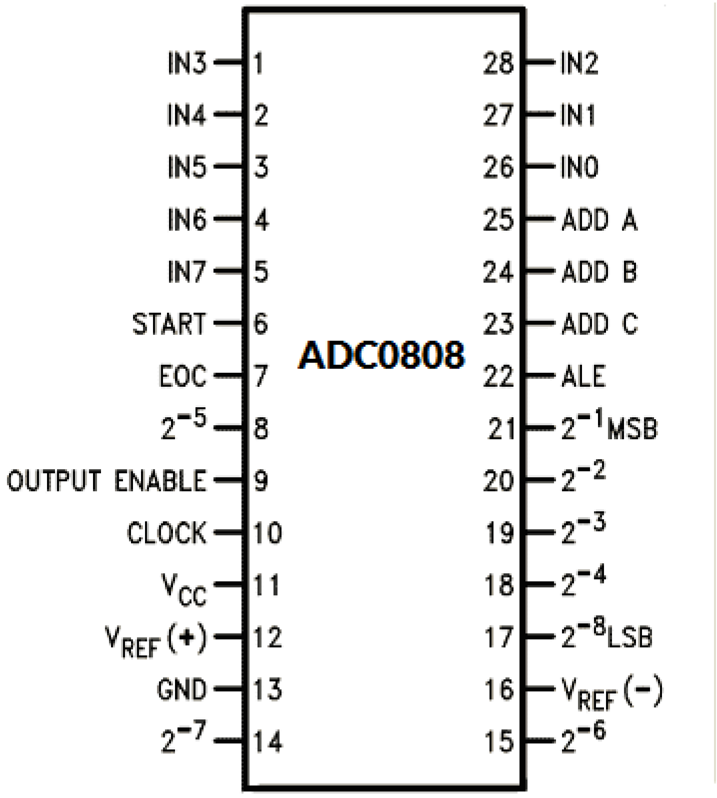
The amplifier is needed to increase the signal output of the sensor so that the ADC can offer distinct, reliable values for each temperature reading received. I implemented it using an operational amplifier in non-inverting configuration.



For simulation I chose the AD822P, a single-supply, rail-to-rail low power FET-Input OpAmp. The main criteria are:

- Offset Voltage: typical 0.1mV
  It is very important so that at 0 ºC even after amplification the ADC will still transmit 0. This value for offset voltage is quite a small one.
- Input Bias Current and Input Offset Current: typical 2pA
  We want it to be as small as possible for the same reasons as before.
- Output Voltage Swing: typically extends to within 10mV of each rail.
  Really useful to choose a rail-to-rail OpAmp so that the supply can be as low as possible.

6.4 Analog to Digital Convertor



An analog-to-digital converter changes an analog signal that's continuous in terms of both time and amplitude to a digital signal that's discrete in terms of both time and amplitude. The conversion involves quantization of the input, so it necessarily introduces a small amount of error or noise. Furthermore, instead of continuously performing the conversion, an ADC does the conversion periodically, sampling the input, limiting the allowable bandwidth of the input signal.

The ADC used is an ADC0808 since it is a reliable choice, maybe even too complex for the needs of the project but is one of the only options in Proteus. The resolution is of 8 bits, conversion time of 100 us and single supply equal to 5V. The ADC0808 chip is designed with an 8-channel multiplexer, which allows it to select one of eight analog input channels to convert. It also has an internal clock that controls the sampling and conversion of the analog input signal. The chip uses a successive approximation technique for conversion, which involves comparing the input voltage with an internal reference voltage and gradually narrowing down the range until the output is an 8-bit digital value.

The output data of the ADC0808 is presented in parallel format, with each bit being represented by a corresponding output pin. The chip also includes a start conversion (SC) input pin, which initiates the conversion process, and an end of conversion (EOC) output pin, which indicates when the conversion is complete, and the digital output is available.

I chose to connect the output of the amplifier is to the IN0 (input 0) pin of the ADC0808 because it allows the analogue signal to be converted into a digital value that can be processed by a microcontroller or other digital device. IN0 is the first input channel of the multiplexer in the ADC0808, which selects the channel that will be converted.

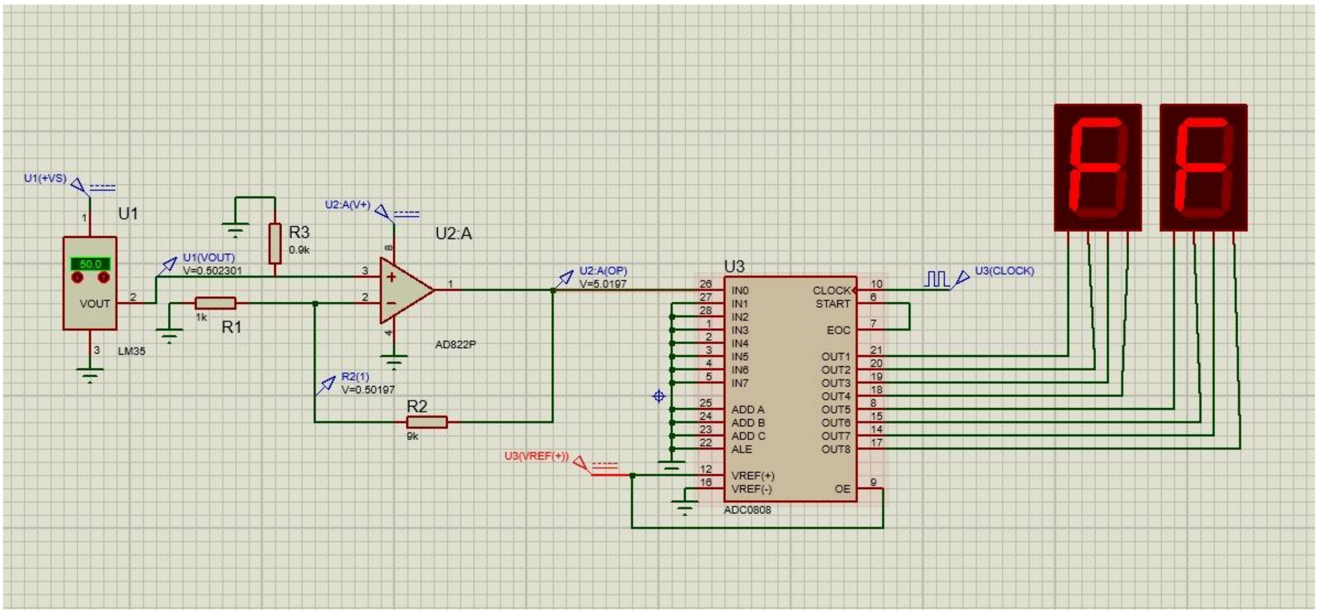
Pins 1-7 on the ADC0808 are not used in the basic mode of operation and are left unconnected (or "free") in most applications. In some advanced applications, these pins may be used for other functions, such as setting up a differential input mode, testing and calibration, or interfacing with other devices. However, in most cases, they are left unconnected to simplify the circuit design and reduce the potential for errors or interference.

By grounding the AD A, AD B, and AD C pins, the corresponding address bits are set to 0, which selects the default mode of operation. The default mode of operation for the ADC0808 is a single-ended, 8-channel multiplexed operation with an internal clock frequency of 640 kHz. In this mode, the ADC0808 sequentially samples and converts each of the eight analogue input channels, starting with channel 0 (IN0) and ending with channel 7 (IN7).

The OE(Output Enable) pin can be used to control the timing of the output data and to avoid any contention issues with other devices that may be connected to the same bus. For example, if multiple devices are connected to the same bus, the OE pin can be used to ensure that the ADC0808 only drives the bus when its data is needed and avoids any conflicts with other devices that may also be trying to drive the bus at the same time.

In summary, the OE pin of the ADC0808 is used to enable or disable the output data from the device and can be used to control the timing and avoid contention issues when multiple devices are connected to the same bus.

6.5 Electrical Schematic



The ADC working principle together with the temperature sensor will dictate the dimensioning of the amplifier's components.

$$N = \frac{V}{V_{max}} * N_{max}$$

N = number from the output of the ADC

V = analogue input voltage in ADC

V_{max}= $V_{max} = 5v - 1LSB$ (power supply for the ADC)

$V_{LSB} = \frac{V_{FS}}{2^n} \Rightarrow V_{LSB} = \frac{5}{2^8} = 20mV$

N_{max}=$2^n = 2^8 = 255$ (maximum number generated by the converter)

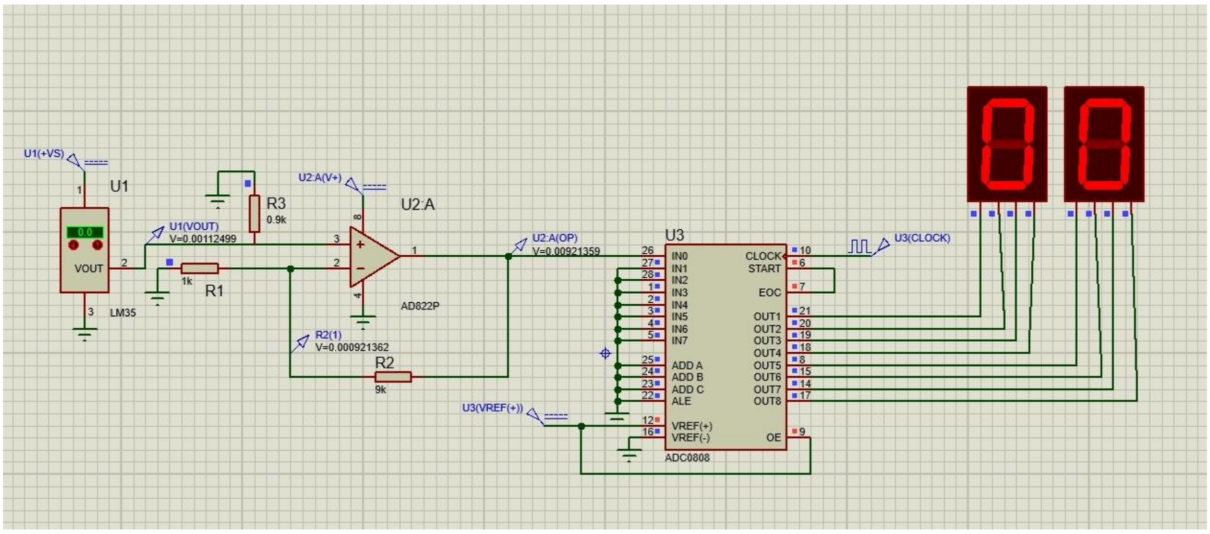
In order to obtain $N = N_{max}$ the analog input voltage of the ADC should be equal with the power supply. So at the maximum temperature of 50 ºC the input voltage must be 5V so that on the 7-segment display it will be shown FFh.

In order to obtain 5V at the input of the ADC, knowing that at the output of the sensor for a temperature of 50 ºC it will be 500mV it results that we need an amplification of 10.

The functioning equation of the non-inverting amplifier is: $V_{out} = (1 + \frac{R_2}{R_1}) * V_{in}$

It result that $\frac{R_2}{R_1} = 9$ and I have chosen $R_2 = 9k$ and $R_1 = 1k$

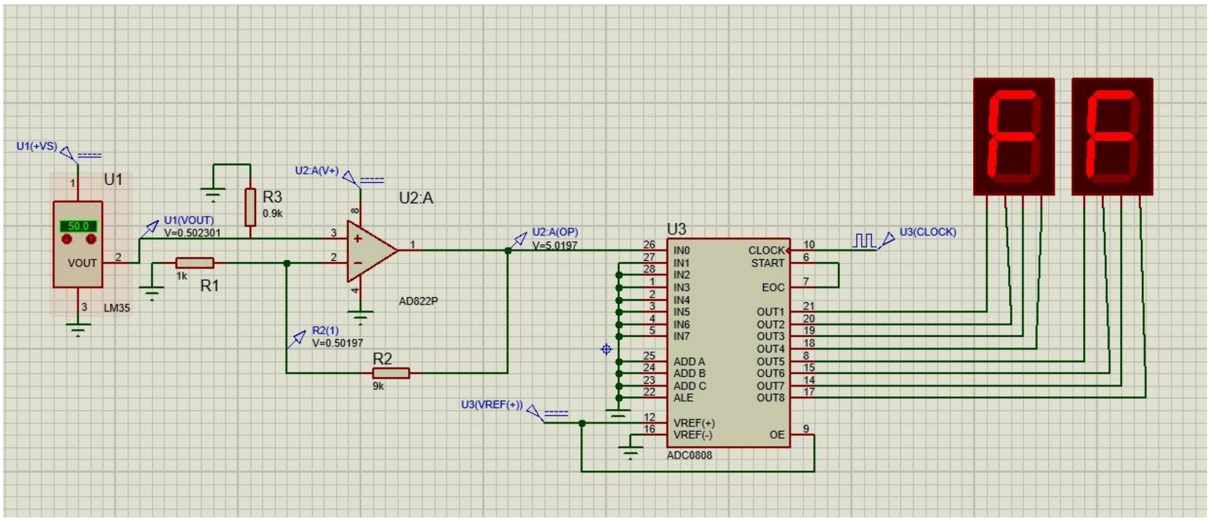
6.6 Simulation



For a temperature of 0 ºC the output of the ADC is 0.

Vtemp = 0.0011 => VoutAMP = 0.011

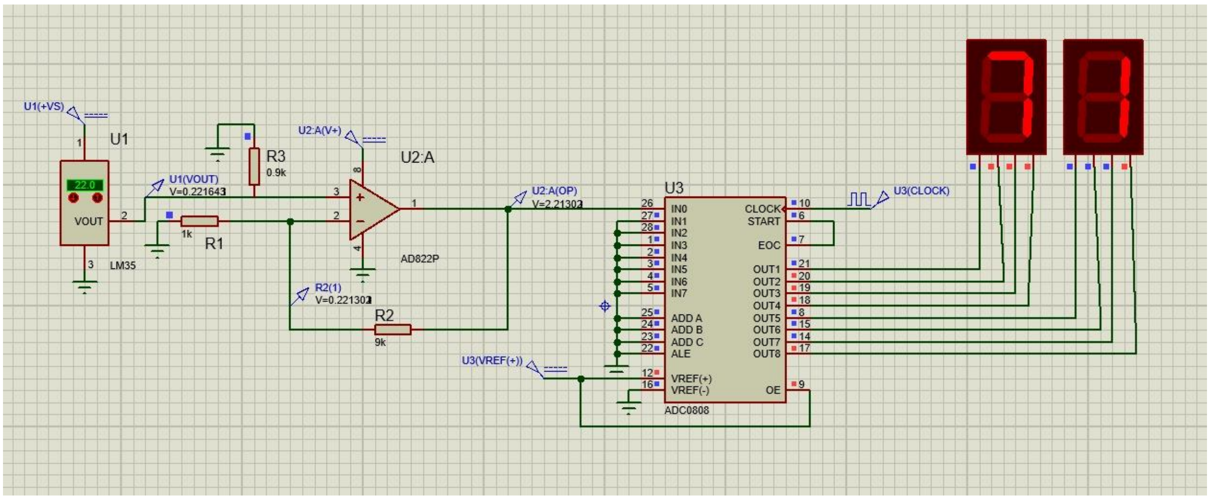
$$N = \frac{V}{V_{max}} * N_{max} = \frac{0.011}{4.98} * 256 = 0.56 => N = 0$$



For a temperature of 50 ºC the output of the ADC is 256 (FFh).

Vtemp = 0.502 => VoutAMP = 5.02

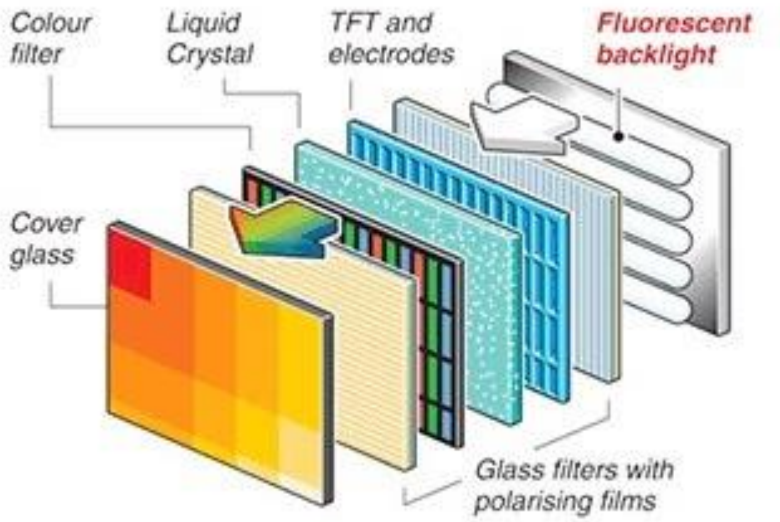
$$N = \frac{V}{V_{max}} * N_{max} = \frac{5.02}{4.98} * 256 = 257.08 => N = 256 \text{ (256 is maximum)}$$



For a typical room temperature of 22 ºC the output of the ADC is 113 (71h), almost in the middle.

7 LCD

A liquid-crystal display (LCD) is a flat-panel display or other electronically modulated optical device that uses the light-modulating properties of liquid crystals combined with polarizers. Liquid crystals do not emit light directly but instead use a backlight or reflector to produce images in colour or monochrome.



7.1 Working Principle

An LCD (Liquid Crystal Display) is a type of flat-panel display that uses liquid crystals to produce images. Here are the basic steps of how an LCD works:

1. Light source: The first step in the process is to provide a light source behind the LCD panel. This can be a backlight or a sidelight, depending on the type of LCD.

2. Polarization: The light then passes through a polarizing filter, which aligns the light waves in a single direction.

3. Liquid crystals: The light then passes through a layer of liquid crystals, which are molecules that can change the direction of the polarized light depending on the electrical charge applied to them.

4. Voltage applied: When a voltage is applied to the liquid crystals, they align themselves in a way that either allows the polarized light to pass through or blocks it.

5. Colour filter: The next step is to add a colour filter layer on top of the liquid crystals. This layer consists of tiny red, green, and blue filters that create the full-colour spectrum.

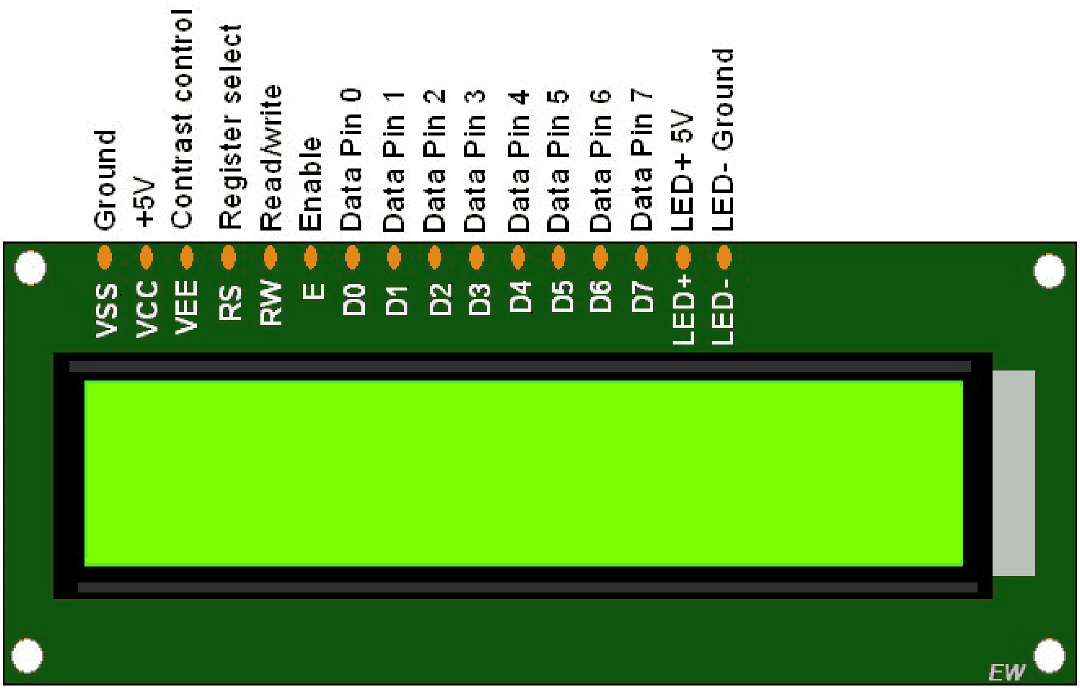
6. Displaying images: The final step is to control the voltage applied to each pixel to create the desired image. By varying the voltage to each pixel, different amounts of light are allowed to pass through, creating the illusion of colour and brightness.

Overall, the LCD works by manipulating the polarized light passing through the liquid crystals to create an image that is displayed on the screen.

7.2 LCD 16*2

An LCD 16x2 (also known as a 16 character by 2 line display) is a common type of alphanumeric LCD display module that is widely used in a variety of electronic devices, such as digital clocks, calculators, and consumer electronics.

The LCD 16x2 has a rectangular shape, typically measuring 80mm x 36mm, with a display area of 64mm x 16mm. It has 16 columns and 2 rows of characters, with each character consisting of 5x8 pixels. The module is typically controlled by a microcontroller or other digital device, which sends commands to the display to control the content and appearance of the characters.



Here's a brief description of the function of each pin:

- VSS: Ground pin
- VCC: Power supply pin (usually +5V)
- VEE: Contrast adjustment pin (can be used to adjust the contrast of the characters on the screen)
- RS: Register Select pin (used to select whether the data being sent to the display is a command or data)
- RW: Read/Write pin (used to select whether data is being written to or read from the display)
- E: Enable pin (used to enable the display for data transfer)
- D0-D7: Data pins (used to transfer data to and from the display)
- LED+: Anode pin (used to power the backlight)
- LED-: Cathode pin (ground for the backlight)

7.3 Types of LCD

There are several types of LCD (Liquid Crystal Display) technology that are commonly used in electronic devices. Here are some of the most common types:

- **TN** (Twisted Nematic)

TN displays are the most common type of LCD display. They are relatively inexpensive and offer fast response times and low power consumption. However, they have limited viewing angles and colour reproduction.

- **IPS** (In-Plane Switching)

IPS displays offer wider viewing angles and better colour reproduction than TN displays. They are commonly used in high-end smartphones, tablets, and computer monitors.

- **VA** (Vertical Alignment)

VA displays offer high contrast ratios and deep blacks, making them well-suited for use in televisions and monitors. However, they have relatively slow response times and limited viewing angles.

- **MVA** (Multi-Domain Vertical Alignment)

MVA displays offer improved viewing angles and response times compared to VA displays, making them well-suited for use in high-end monitors and televisions.

- **OLED** (Organic Light-Emitting Diode)

OLED displays use organic compounds to emit light when an electric current is applied. They offer deep blacks, wide viewing angles, and fast response times, making them well-suited for use in smartphones, televisions, and other devices that require high-quality displays.

- **AMOLED** (Active-Matrix Organic Light-Emitting Diode)

AMOLED displays are a type of OLED display that uses a thin-film transistor (TFT) to control the flow of current to each pixel. They offer faster response times and better colour reproduction than traditional OLED displays, making them well-suited for use in high-end smartphones and other devices.

Overall, the type of LCD technology used in a device depends on the specific application and desired characteristics of the display, such as viewing angles, colour reproduction, response time, and power consumption.

7.4 LM016L LCD

The LM016L is a common type of 16x2 character LCD display module that is widely used in a variety of electronic devices. This is the LCD used for simulation.

Here's an overview of its features and specifications:

- Display type: STN (Super Twisted Nematic)
- Display format: 16 characters x 2 lines
- Character size: 5x8 dots
- Display area: 64.5mm x 14.5mm
- Viewing angle: 6 o'clock
- Backlight type: LED
- Backlight colour: Blue or Green
- Operating voltage: 5V DC
- Operating temperature: -20°C to +70°C
- Controller: HD44780-compatible
- Interface: 4-bit or 8-bit parallel

The LM016L display module typically includes a built-in controller that is compatible with the HD44780 standard, which is a widely used standard for interfacing with LCD displays. This makes it relatively easy to control the display using a microcontroller or other digital device.

The LM016L display module can be interfaced with a microcontroller or other digital device using either a 4-bit or 8-bit parallel interface. This allows for flexible interfacing and easy integration into a wide range of electronic devices.

Overall, the LM016L display module is a versatile and widely used display solution that provides a simple and efficient way to display alphanumeric characters in a variety of electronic devices.

7.5 Display data on the LM016L LCD

These are the commands executed by the HD44780 controller in order to display data on the LA016L LCD.

Table 13 8-Bit Operation, 8-Digit × 2-Line Display Example with Internal Reset

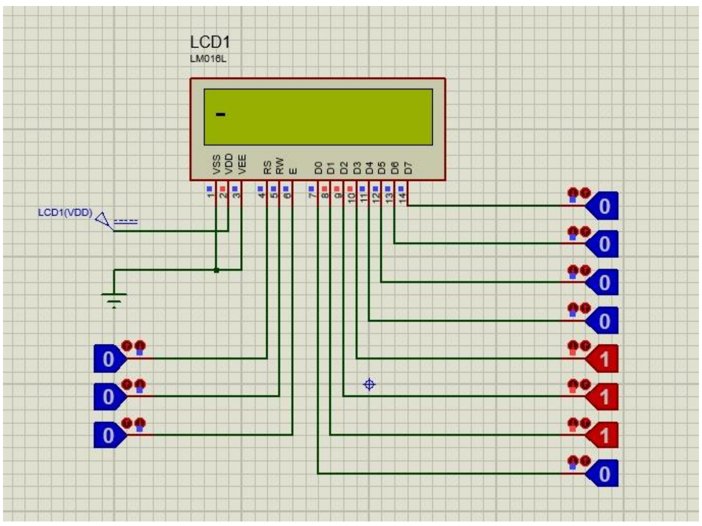
Step No.	Instruction RS	$R/\overline{W}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Display	Operation
1	Power supply on (the HD44780U is initialized by the internal reset circuit)											Initialized. No display.
2	Function set											Sets to 8-bit operation and selects 2-line display and 5 × 8 dot character font.
	0	0	0	0	1	1	1	0	*	*		
3	Display on/off control										_	Turns on display and cursor. All display is in space mode because of initialization.
	0	0	0	0	0	0	1	1	1	0		
4	Entry mode set										_	Sets mode to increment the address by one and to shift the cursor to the right at the time of write to the DD/CGRAM. Display is not shifted.
	0	0	0	0	0	0	0	1	1	0		
5	Write data to CGRAM/DDRAM										H_	Writes H. DDRAM has already been selected by initialization when the power was turned on. The cursor is incremented by one and shifted to the right.
	1	0	0	1	0	0	1	0	0	0		
6	. . . . .										. . . . .	
7	Write data to CGRAM/DDRAM										HITACHI_	Writes I.
	1	0	0	1	0	0	1	0	0	1		
8	Set DDRAM address										HITACHI _	Sets DDRAM address so that the cursor is positioned at the head of the second line.
	0	0	1	1	0	0	0	0	0	0		

HD44780U

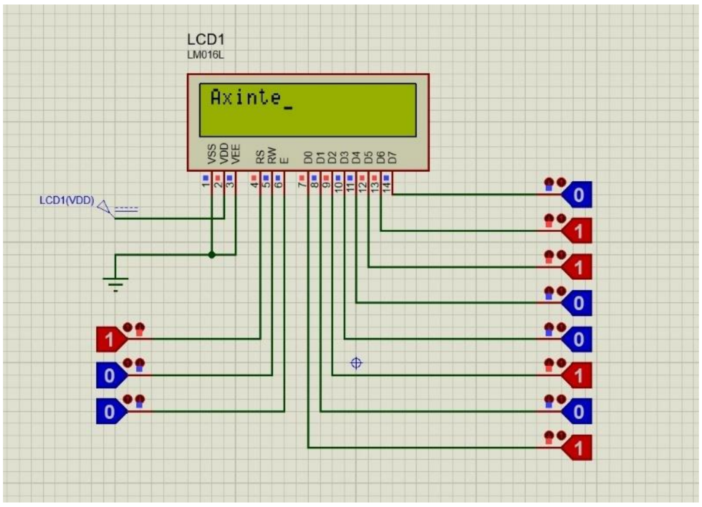
Table 13 8-Bit Operation, 8-Digit × 2-Line Display Example with Internal Reset (cont)

Step No.	RS	R/$\overline{W}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Display	Operation
9	Write data to CGRAM/DDRAM											
	1	0	0	1	0	0	1	1	0	1	HITACHI M_	Writes M.
10	⋮										⋮	
11	Write data to CGRAM/DDRAM											
	1	0	0	1	0	0	1	1	1	1	HITACHI MICROCO_	Writes O.
12	Entry mode set											
	0	0	0	0	0	0	0	1	1	1	HITACHI MICROCO_	Sets mode to shift display at the time of write.
13	Write data to CGRAM/DDRAM											
	1	0	0	1	0	0	1	1	0	1	ITACHI ICROCOM_	Writes M. Display is shifted to the left. The first and second lines both shift at the same time.
14	⋮										⋮	
15	Return home											
	0	0	0	0	0	0	0	0	1	0	HITACHI MICROCOM	Returns both display and cursor to the original position (address 0).

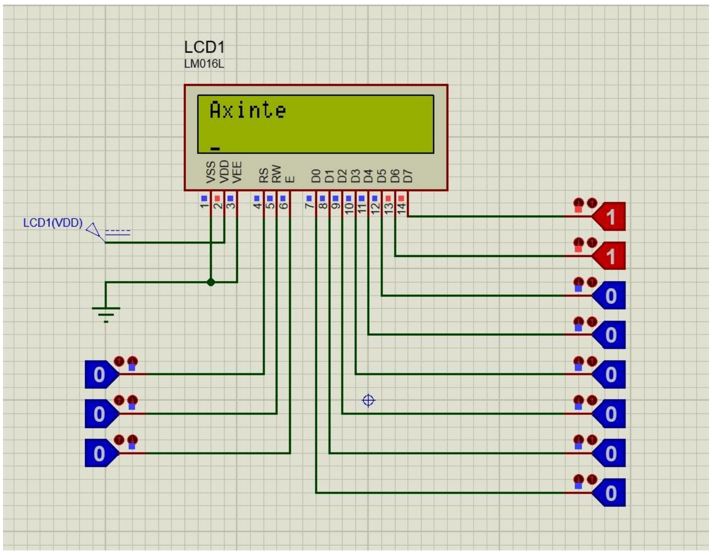
7.6 Simulation



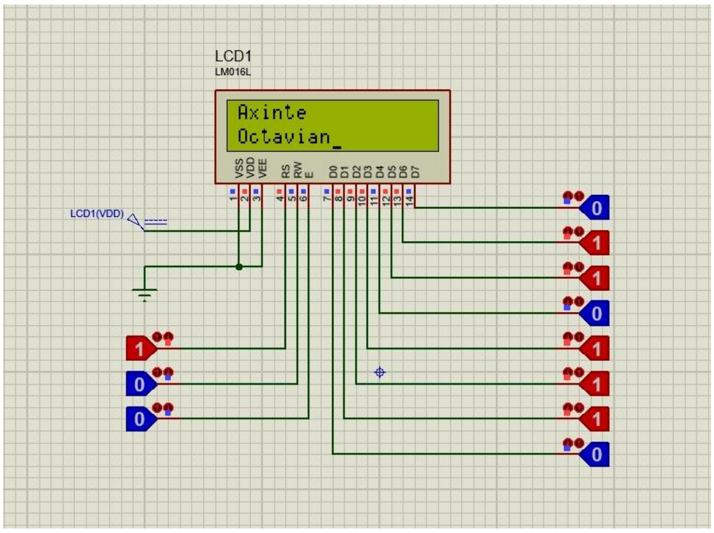
Cursor turned on ^



Last name on the first row ^



Cursor on the second row ^



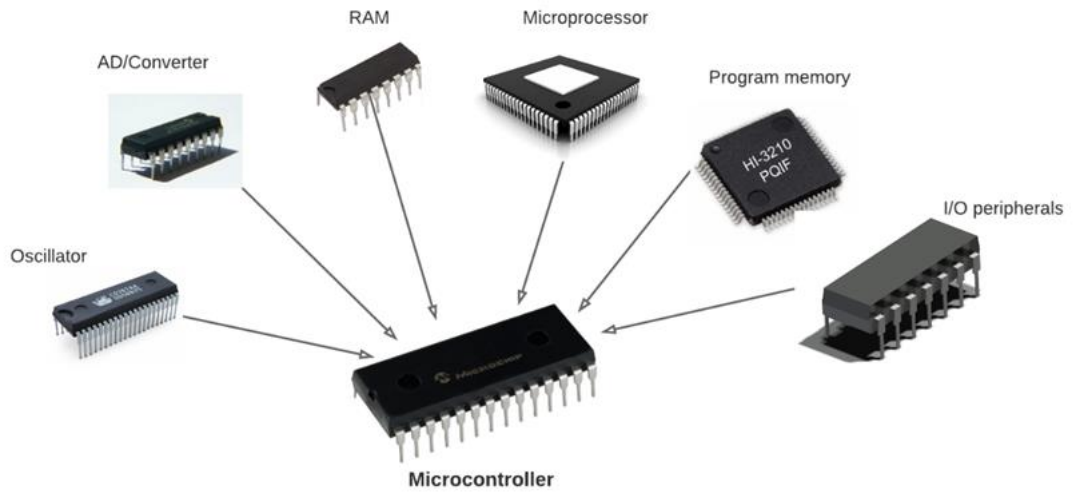
First name on the second row ^

8 Microcontroller

8.1 Definition

A microcontroller is a small computer on a single integrated circuit (IC) chip that contains a processor core, memory, and input/output peripherals. It is designed to control specific devices or systems and is used in a wide range of electronic devices, from home appliances to industrial machines.

8.2 Components



- Processor Core

The central processing unit (CPU) is the heart of the microcontroller, responsible for executing instructions and performing calculations.

- Memory

The microcontroller typically contains two types of memory - Random Access Memory (RAM) for data storage and Read-Only Memory (ROM) for storing the program code.

- Input/Output Peripherals

These components are responsible for interfacing with external devices and sensors. Examples include digital and analogue input/output pins, timers, serial communication ports, and interrupt controllers.

- Clock

The microcontroller requires a clock to synchronize its operations. This clock is usually provided by an external crystal oscillator or an internal oscillator circuit.

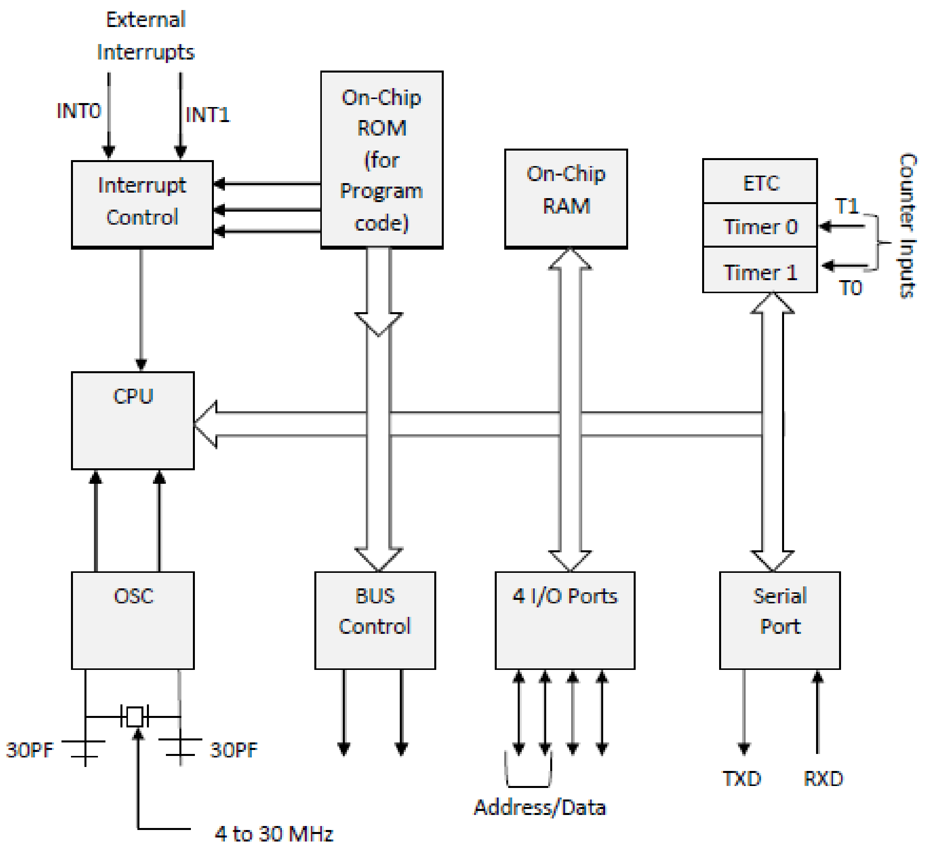
- Power Management Unit

This component regulates the power supply to the microcontroller, ensuring that it operates within safe voltage and current limits.

- Programming/Debugging Interface

This interface allows developers to program the microcontroller and debug the code running on it. It typically consists of a set of pins that connect to a programmer/debugger device.

8.3 8051 Microcontroller



The 8051 microcontroller is a family of microcontrollers that were first introduced by Intel in 1980. It is one of the most widely used microcontrollers in the world, and its popularity has led to a large number of manufacturers producing compatible chips.

The 8051 family microcontrollers have a simple architecture and are easy to program. They typically have a 8-bit data bus and 16-bit address bus, and run at speeds of up to 33 MHz.

Some of the key features of the 8051 microcontroller family include:

- On-chip memory:

The 8051 microcontrollers typically have on-chip memory, including RAM, ROM, and EEPROM, which makes them suitable for a wide range of applications.

- Interrupts:

The 8051 microcontrollers have a flexible interrupt structure, which allows for the handling of external events in real-time.

- Timers/counters:

The 8051 microcontrollers typically have a number of timers/counters that can be used for a range of applications, including generating PWM signals and measuring pulse widths.

- Serial communication:

The 8051 microcontrollers typically have one or more serial communication ports, which can be used to interface with other devices.

- GPIO pins:

The 8051 microcontrollers typically have a number of general-purpose input/output (GPIO) pins, which can be used for a range of applications, including controlling LEDs and reading switches.

8.4 Comparison of 5 8051 Microcontrollers

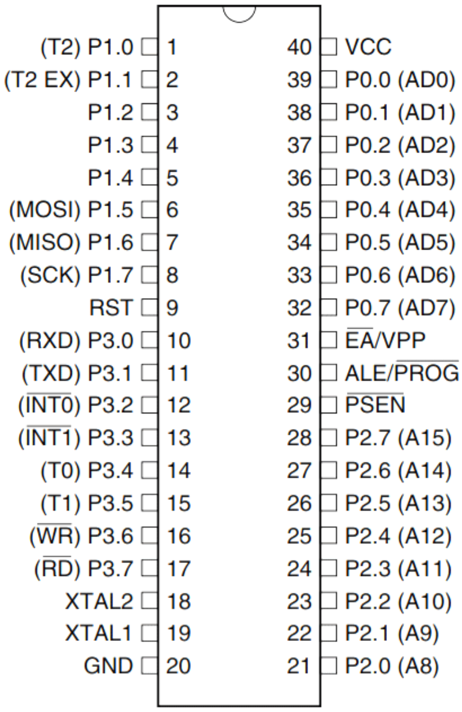
Microcontroller	**Manufacturer**	**Flash Memory**	**RAM**	**I/O Pins**	**Price Lei**	**Timer/Counters**
AT89S51	Atmel	4KB	128 bytes	32	18	2 (16-bit)
AT89S52	Atmel	8KB	256 bytes	32	20	2 (16-bit)
DS89C450	Maxim Integrated	64KB	2KB	32	92	3 (16-bit)
CY8C29466	Cypress Semiconductor	64KB	4KB	42	55	2 (16-bit)

STC89C52RC	STC Microelectronics	32KB	1KB	32	7	2 (16-bit)

The application requires small storage and no special functions so the decision is to use AT89S52 due to the low price and availability.

8.5 AT89S52

The AT89S52 microcontroller has a total of 40 pins, each of which serves a specific function. Here's a brief description of the various pins of the microcontroller:



P1.0 - P1.7: These are the eight bidirectional I/O pins of Port 1. They can be configured as input or output pins.

P2.0 - P2.7: These are the eight bidirectional I/O pins of Port 2. They can be configured as input or output pins.

P3.0 - P3.7: These are the eight bidirectional I/O pins of Port 3. They can be configured as input or output pins.

P0.0 - P0.7: These are the eight bidirectional I/O pins of Port 0. They can be configured as input or output pins.

XTAL1 and XTAL2: These are the input and output pins, respectively, of an external crystal oscillator or resonator. They provide the clock signal to the microcontroller.

RST: This is the reset pin of the microcontroller. When this pin is pulled low, the microcontroller resets and starts executing code from the beginning.

ALE: This is the Address Latch Enable pin. It is used to latch the address from the microcontroller onto an external latch.

EA/VPP: This pin is used to select the source of the program memory. When the pin is connected to VCC, the program memory is sourced from an external device. When the pin is connected to ground, the program memory is sourced from the internal flash memory. This pin is also used as the programming voltage input during in-system programming.

PSEN: This is the Program Store Enable pin. It is used to indicate that the microcontroller is accessing program memory.

INT0: This is the external interrupt 0 pin. It is used to trigger an interrupt when a signal is applied to the pin.

INT1: This is the external interrupt 1 pin. It is used to trigger an interrupt when a signal is applied to the pin.

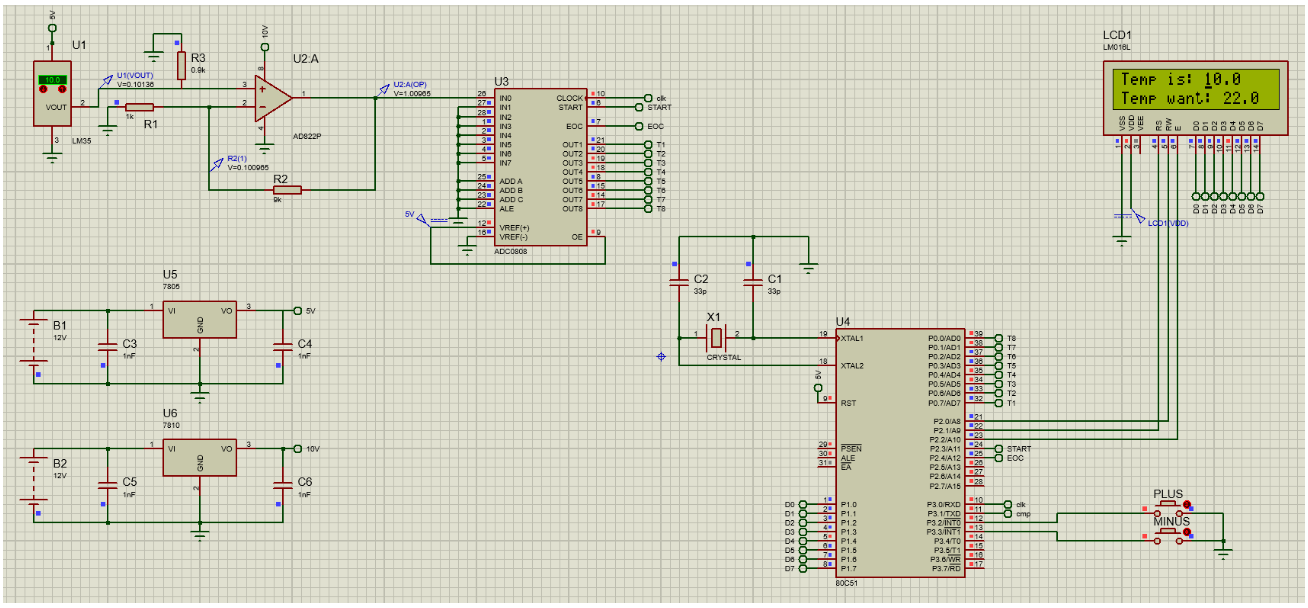
TXD: This is the transmit data pin of the built-in serial communication port.

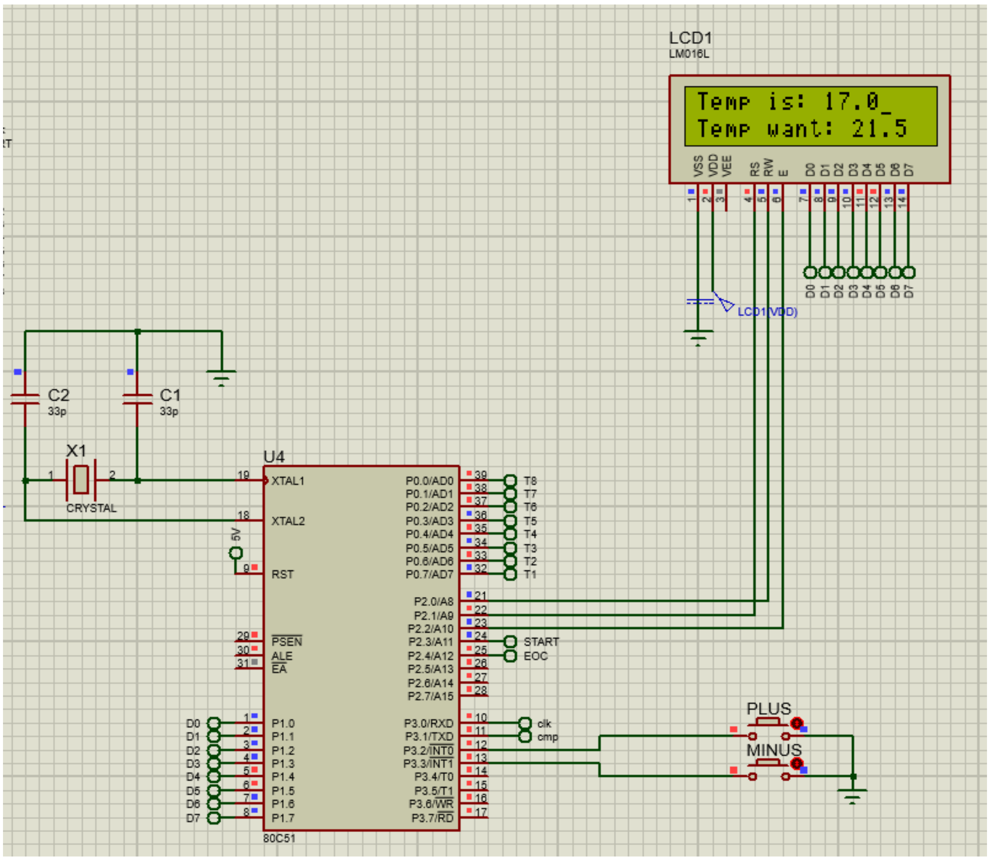
RXD: This is the receive data pin of the built-in serial communication port.

VCC and GND: These pins provide power supply and ground connections to the microcontroller.

9 Electrical Schematic

9.1 Proteus Implementation





The output pins of the ADC0808 are connected to the first port P0 of the microcontroller. The START and EOC pins are connected to P2.3 and P2.4. The LCD pins are controlled by the first port P1 of the microcontroller. The RS, R/W and E pins are connected to P2.0, P2.1, P2.2. The keyboard represented by the 2 buttons is connected to P3.2, P3.3 (INT0, INT1). The clock signal necessary for the ADC is given by P3.0. The turn on signal for the heating system can be found at P3.1 as "cmp".

Between XTAL1 and XTAL2 is connected a crystal oscillator with a frequency of 12MHz. Compared to an internal clock, the external crystal oscillator is more precise. The two capacitors have two purposes. The first one is to filter the unwanted frequencies that may be generated by the oscillator. Also, we use two capacitors in order to keep the phase shift unaltered.

The two buttons give the user the possibility to choose a certain wanted temperature. When a button is pressed the port pin will be activated (0 logic) and the required instruction will be executed. The two temperatures will always be compared. If the wanted temperature is greater than the present temperature than the heating system will be powered on.

I also used two LDOs to power on my circuit from a 12V battery. One has a 5V output and the other one has a 10V output.

9.2 Programming Logic

As previously mentioned, the ADC output will be between 00 and FFh for temperatures between 0 and 50 ºC. The question is how to display on the LCD the temperature from the sensor using the large domain (256 values) of the ADC output. These are three possible solutions, but only one was implemented.

- Changing the amplification so that at the output of the ADC the domain will be between 0 and 32h (50). In this way, no supplementary operations are needed.
- Using a look-up table assigning for every value from 00 to FFh a certain temperature.
- Dividing the ADC output in such a way that we obtain firstly the tens digit and then the unit digit

I chose the third method since it requires the least steps to apply, despite the more complicated logic.

```
LOAD:
ACALL READY_CONVERSION
MOV A, P3
MOV B, #33h
DIV AB
add A, #30h
acall display
mov A, B
mov B, #5
div AB
add A, #30h
acall display
RET
```

We observe that for the 10 ºC temperature the output from the ADC is 33h. Knowing that the temperature sensor has a linear characteristic it can also be computed: FFh : 5 = 33h.

In order to find the tens digit it is sufficient to divide the ADC output value to 33h. The quotient then is transformed in ASCII code and displayed. The remainder will have values between 0 and 32h (50). In order to have a unit between 0 and 9 then we need to normalise the interval by dividing with 5. The new quotient is transformed in ASCII code and displayed.

```
org 100h
STR: DB "Temperature is:", 0
```

```
;display on LCD the string
MOV DPTR ,#str
l1 : MOV A,#00H
MOVC A,@A+DPTR
JZ done
ACALL DISPLAY
INC DPTR
SJMP l1
```

```
done:
```

Another important part of the code is the one above. Instead of displaying the characters one by one in code I preferred to place a string in the memory and display it in a loop. The characters will still be displayed one by one on the LCD because it is the only possibility, but the speed will make it appear at the same time for the human eye. This is also done by checking the busy flag of the LCD instead of using a delay.

```
;verify if the LCD displayed the character in order to display the next one
READY_LCD:
CLR P2.2
SETB P1.7
CLR P2.1 ;RS=0
SETB P2.0 ; R/W = 1 => READ COMMAND REG
BACK:   CLR P2.2
        nop
        nop
        SETB P2.2
        JB P1.7, BACK
CLR P2.2
    RET

;verify if the conversion of the ADC is done
READY_CONVERSION:
SETB P2.4  ;eoc
NOP
SETB P2.3 ; start
NOP
NOP
CLR P2.3
BACK2: JNB P2.4, BACK2
    RET
```

For LCD we check if D7(busy flag) is one so that we can load the next character.

For the ADC we check the EOC bit so that we can start the next conversion.

```
TABLE:  DB "00.0","00.5", "01.0", "01.5", "02.0", "02.5", "03.0", "03.5", "04.0", "04.5", "05.0", "05.5", "06.0", "06.5", "07.0", "07.5", "08.0", "08.5", "09.0", "09.5"
        DB "10.0","10.5", "11.0", "11.5", "12.0", "12.5", "13.0", "13.5", "14.0", "14.5", "15.0", "15.5", "16.0", "16.5", "17.0", "17.5", "18.0", "18.5", "19.0", "19.5"
        DB "20.0","20.5", "21.0", "21.5", "22.0", "22.5", "23.0", "23.5", "24.0", "24.5", "25.0", "25.5", "26.0", "26.5", "27.0", "27.5", "28.0", "28.5", "29.0", "29.5"
        DB "30.0","30.5", "31.0", "31.5", "32.0", "32.5", "33.0", "33.5", "34.0", "34.5", "35.0", "35.5", "36.0", "36.5", "37.0", "37.5", "38.0", "38.5", "39.0", "39.5"
        DB "40.0","40.5", "41.0", "41.5", "42.0", "42.5", "43.0", "43.5", "44.0", "44.5", "45.0", "45.5", "46.0", "46.5", "47.0", "47.5", "48.0", "48.5", "49.0", "49.5"
```

```
MOV r0, #44
```

For the wanted temperature I wanted to use a different technique. A look-up table.

The offset value for the table is saved in R0. I encountered many problems, one of them being that the 8b r0 is not enough. We have 100 possible temperatures, on 4b each one of them => 400 bits, but we only have 256.

The solution was to save only the index of temperature (0-99) in R0 and then when displaying to multiply it with 4 so that we arrive at the real location in the table. The first button implements the “+” option whereas the second one implements the “–“ option. For this was necessary a careful computation of offset in the table. The buttons were implemented using interrupts. So the cursor always needed to be placed were it was when the interrupt signal appeared!

```
    butonl:


MOV A, #0C0h
ACALL COMMAND
mov a, #11
cursor:
push acc
mov a, #14h
ACALL COMMAND
pop acc
dec a
jz out
sjmp cursor
out:

   MOV DPTR, #TABLE
   MOV A, r0           ;
   mov b, #4
   mul AB
   push acc
   MOVC A, @A + DPTR
   acall display
   pop acc
   inc a
   push acc
   MOVC A, @A + DPTR
   acall display
   pop acc
   inc a
   push acc
   MOVC A, @A + DPTR
   acall display
   pop acc
   inc a
   push acc
   MOVC A, @A + DPTR
   acall display
   pop acc
   inc a
   mov b, #4
   div AB
   mov r0, a


MOV A, #80H   // force
ACALL COMMAND
mov a, #9
cursorl:
push acc
mov a, #14h
ACALL COMMAND
pop acc
dec a
jz init
sjmp cursorl
init:

reti
```

The most difficult part was implementing the comparison between the current temperature and wanted temperature.

For this I saved the digits from the current temperature in R3, R4, R5. Then I compared the MSB of the current temperature with the MSB of the wanted temperature. Since there are limited mnemonics in Keil, I did this by dividing them and assessing the quotient and reminder. If there was equality between them then I compared the next bit, till the LSB. I was always careful to convert from ASCII to numbers and vice-versa. If the wanted temperature is greater than the current temperature, P3.1 will be ‘1’, otherwise P3.0 will be ‘0’.

```
compare:

mov DPTR, #table
mov a, r0
mov b, #4
mul ab
movc a, @a+DPTR
subb a, #30h
push acc
mov a, r3
subb a, #30h

jnz merge
mov a, #1
merge:

mov b, a ; current temperatu
pop acc ; wanted temperature
div ab
    jz oprim
        cjne a, #1, pornim
        mov a, b
        cjne a, #0, pornim2
            ;continuam cu ur
            mov a, r0
            mov b, #4
            mul ab
            inc a
            movc a, @a+DPTR
            subb a, #30h
            push acc
            mov a, r4
            subb a, #30h

            jnz merge2
            mov a, #1
            merge2:

            mov b, a
            pop acc
            div ab
                jz oprim2
```

```
cjne a, #1, pornim3
mov a, b
cjne a, #0, pornim4
    mov a, r0
    mov b, #4
    mul ab
    inc a
    inc a
    inc a
    movc a, @a+DPTR
    subb a, #30h
    push acc
    mov a, r5
    subb a, #30h

    jnz merge3
    mov a, #1
    merge3:

    mov b, a
    pop acc
    div ab
        jz oprim3
            cjne a, #1, pornim5
            mov a, b
            cjne a, #0, pornim6
                sjmp oprim4
```

```
pornim:
pornim2:
pornim3:
pornim4:
pornim5:
pornim6:
SETB P3.1

sjmp sarim

oprim:
oprim2:
oprim3:
oprim4:
CLR P3.1

sarim:
ret


END
```

The clock signal was also implemented using interrupts. I used timer 1 in mode 2. TCON.0, TCON.2 are really important to be 1, so that the signal will be triggered on falling edge.

```
MOV IE,#10001101B
setb TCON.0
setb TCON.2
MOV r0, #44

; 5KHZ => 200us Period => 100us delay : 1.08507 = 92
MOV TMOD,#20H;Timer1,mod2
MOV TL1, -92
MOV TH1, #TL1
SETB TR1
```

9.3 Assembly Code

```
org 300h

str: DB "Temp is: ", 0
str2: DB "Temp want: ", 0
TABLE:   DB "00.0","00.5", "01.0", "01.5", "02.0", "02.5", "03.0", "03.5", "04.0", "04.5", "05.0", "05.5", "06.0", "06.5", "07.0", "07.5", "08.0", "08.5", "09.0", "09.5"
         DB "10.0","10.5", "11.0", "11.5", "12.0", "12.5", "13.0", "13.5", "14.0", "14.5", "15.0", "15.5", "16.0", "16.5", "17.0", "17.5", "18.0", "18.5", "19.0", "19.5"
         DB "20.0","20.5", "21.0", "21.5", "22.0", "22.5", "23.0", "23.5", "24.0", "24.5", "25.0", "25.5", "26.0", "26.5", "27.0", "27.5", "28.0", "28.5", "29.0", "29.5"
         DB "30.0","30.5", "31.0", "31.5", "32.0", "32.5", "33.0", "33.5", "34.0", "34.5", "35.0", "35.5", "36.0", "36.5", "37.0", "37.5", "38.0", "38.5", "39.0", "39.5"
         DB "40.0","40.5", "41.0", "41.5", "42.0", "42.5", "43.0", "43.5", "44.0", "44.5", "45.0", "45.5", "46.0", "46.5", "47.0", "47.5", "48.0", "48.5", "49.0", "49.5"

ORG 0000H
LJMP main

org 0003h
    LJMP buton1

org 0013h
    LJMP buton2

ORG 001BH
    cpl P3.0
    retiS

org 0030h
    main:
MOV A, #38H     // use 2 lines and 5*7
ACALL COMMAND
MOV A, #0EH   //cursor blinking off
ACALL COMMAND
MOV A, #01H     //clr screen
ACALL COMMAND

MOV IE,#10001101B
setb TCON.0
setb TCON.2
MOV r0, #44

; 5KHZ => 200us Period => 100us delay : 1.08507 = 92
MOV TMOD,#20H;Timer1,mod2
MOV TL1, -92
MOV TH1, #TL1
SETB TR1

mov A, #0C0h
ACALL COMMAND
MOV DPTR ,#str2

l3 : MOV A,#00H
MOVC A,@A+DPTR
JZ done3
ACALL DISPLAY
INC DPTR
```

```
SJMP l3
done3:

MOV A, #80H   // force cursor to first line
ACALL COMMAND
;display on LCD the string
MOV DPTR ,#str
l1 : MOV A,#00H
MOVC A,@A+DPTR
JZ done
ACALL DISPLAY
INC DPTR
SJMP l1
done:


;read and display the temperature on the LCD
temp:
ACALL LOAD
mov a, #10h
ACALL COMMAND
mov a, #10h
ACALL COMMAND
mov a, #10h
ACALL COMMAND
mov a, #10h
ACALL COMMAND
SJMP temp
;convert the output from the ADC to ASCII for display
LOAD:
ACALL READY_CONVERSION
;tens
MOV A, P0
MOV B, #33h
DIV AB
add A, #30h
mov r3, A; pt comparare temperaturi
acall display
;units
mov A, B
mov B, #5
div AB
add A, #30h
mov r4, A; pt comparare temperaturi
acall display
;first decimal
mov A,#"."
acall display
mov A, B
mov B, #3
div AB
cjne A, #0, mare
mov A, #"0"
mov r5, A; pt comparare temperaturi
acall display
sjmp gata
mare:
mov A, #"5"
mov r5, A ; pt comparare temperaturi
acall display
gata:

acall compare

RET
```

```
;execute commands for LCD configuration
COMMAND:
ACALL READY_LCD
MOV P1, A
CLR P2.1
CLR P2.0
SETB P2.2
NOP
NOP
CLR P2.2
RET

;display character on LCD
DISPLAY:
ACALL READY_LCD
MOV P1, A
SETB P2.1
CLR P2.0
SETB P2.2
NOP
NOP
CLR P2.2
RET

;verify if the LCD displayed the character in order to display the next one
READY_LCD:
CLR P2.2
SETB P1.7
CLR P2.1 ;RS=0
SETB P2.0 ; R/W = 1 => READ COMMAND REG
BACK:   CLR P2.2
        nop
        nop
        SETB P2.2
        JB P1.7, BACK
CLR P2.2
    RET

;verify if the conversion of the ADC is done
READY_CONVERSION:
SETB P2.4  ;eoc
NOP
SETB P2.3 ; start
NOP
NOP
CLR P2.3
BACK2: JNB P2.4, BACK2
    RET
```

```
    buton1:


MOV A, #0C0h
ACALL COMMAND
mov a, #11
cursor:
push acc
mov a, #14h
ACALL COMMAND
pop acc
dec a
jz out
sjmp cursor
out:

   MOV DPTR, #TABLE
   MOV A, r0
   mov b, #4
   mul AB
   push acc
   MOVC A, @A + DPTR
   acall display
   pop acc
   inc a
   push acc
   MOVC A, @A + DPTR
   acall display
   pop acc
   inc a
   push acc
   MOVC A, @A + DPTR
   acall display
   pop acc
   inc a
   push acc
   MOVC A, @A + DPTR
   acall display
   pop acc
   inc a
   mov b, #4
   div AB
   mov r0, a


MOV A, #80H   // force cursor
ACALL COMMAND
mov a, #9
cursor1:
push acc
mov a, #14h
ACALL COMMAND
pop acc
dec a
jz init
sjmp cursor1
init:

reti
```

```
    buton2:

MOV A, #0C0h
ACALL COMMAND
mov a, #11
cursor2:
push acc
mov a, #14h
ACALL COMMAND
pop acc
dec a
jz out2
sjmp cursor2
out2:

   MOV DPTR, #TABLE
   MOV A, r0          ;
   dec a
   dec a
   mov B, #4
   mul AB
   push acc
   MOVC A, @A + DPTR
   acall display
   pop acc
   inc a
   push acc
   MOVC A, @A + DPTR
   acall display
   pop acc
   inc a
   push acc
   MOVC A, @A + DPTR
   acall display
   pop acc
   inc a
   push acc
```

```
   push acc
   MOVC A, @A + DPTR
   acall display
   pop acc
   inc a
   mov b, #4
   div AB
   mov r0, a


MOV A, #80H   // force cu
ACALL COMMAND
mov a, #9
cursor3:
push acc
mov a, #14h
ACALL COMMAND
pop acc
dec a
jz init2
sjmp cursor3
init2:
reti
```

```
compare:

mov DPTR, #table
mov a, r0
mov b, #4
mul ab
movc a, @a+DPTR
subb a, #30h
push acc
mov a, r3
subb a, #30h

jnz merge
mov a, #1
merge:

mov b, a ; current temperatu
pop acc ; wanted temperature
div ab
    jz oprim
        cjne a, #1, pornim
        mov a, b
        cjne a, #0, pornim2
            ;continuam cu ur
            mov a, r0
            mov b, #4
            mul ab
            inc a
            movc a, @a+DPTR
            subb a, #30h
            push acc
            mov a, r4
            subb a, #30h

            jnz merge2
            mov a, #1
            merge2:

            mov b, a
            pop acc
            div ab
                jz oprim2
cjne a, #1, pornim3
mov a, b
cjne a, #0, pornim4
    mov a, r0
    mov b, #4
    mul ab
    inc a
    inc a
    inc a
    movc a, @a+DPTR
    subb a, #30h
    push acc
    mov a, r5
    subb a, #30h

    jnz merge3
    mov a, #1
    merge3:

    mov b, a
    pop acc
    div ab
        jz oprim3
            cjne a, #1, pornim5
            mov a, b
            cjne a, #0, pornim6
                sjmp oprim4
pornim:
pornim2:
pornim3:
pornim4:
pornim5:
pornim6:
SETB P3.1

sjmp sarim

oprim:
oprim2:
oprim3:
oprim4:
CLR P3.1

sarim:
ret


END
```

9.4 C Code

```
#include <reg51.h>

sbit D7 = P1^7;
sbit START = P2^3;
sbit EOC = P2^4;
sbit R_W = P2^0;
sbit RS = P2^1;
sbit EN = P2^2;
sbit CLK  = P3^0;
sbit CMP = P3^1;
sbit BUTP = P3^2;
sbit BUTM = P3^3;

void display_temperature();
void command_LCD(unsigned char x);
void display_LCD(unsigned char x);
void ready_LCD();
void ready_conversion() ;
void display_string(unsigned char *x);
void compare();

void timer1() interrupt 3{
   CLK =~ CLK;
}

void btn1() interrupt 0{

}

void btn0() interrupt 2{

}

```

```c
code unsigned char str[] = {"Temp is:"};
code unsigned char str2[] = {"Temp want:"};
unsigned char r0, r3, r4, r5;

void main() {

    command_LCD(0x38);
    command_LCD(0x0E);
    command_LCD(0x01);

    IE = 0x8D;
    TCON = 0x05;
    r0 = 44;

    TMOD = 0x20;
    TL1 = -92;
    TH1 = TL1;
    TR1 = 1;

    command_LCD(0xC0);

    int i = 0;
    while (str2[i] != '\0') {
        display_LCD(str2[i]);
        i++;
    }

    command_LCD(0x80);

    char str[] = "Temperature: ";
    i = 0;
    while (str[i] != '\0') {
        display_LCD(str[i]);
        i++;
    }
```

```c
    while (1) {
        display_temperature();
        command(0x10);
        command(0x10);
        command(0x10);
        command(0x10);
    }
}

void display_temperature(){
    ready_conversion();

    // tens
    unsigned char A, B;
    A = P0;
    B = 0x33;
    A = A / B;
    A = A + 0x30;
    r3 = A;
    display(r3);

    // units
    A = B;
    B = 0x05;
    A = A / B;
    A = A + 0x30;
    r4 = A;
    display(r4);

    // first decimal
    A = '.';
    display(A);
    A = B;
    B = 0x03;
    A = A / B;
```

```
    if (A == 0) {
        A = '0';
    } else {
        A = '5';
    }
    r5 = A;
    display(r5);

    compare();
}

void command_LCD(unsigned char x){
  ready_LCD();
  P1 = x;
  RS = 0;
  R_W = 0;
  EN = 1;
  {}{}
  EN = 0;
}

void display_LCD(unsigned char x) {
  ready_LCD();
  P1 = x;
  RS = 1;
  R_W = 0;
  EN = 1;
  {}{}
  EN = 0;
}
```

```
void ready_LCD(){
  EN = 0;
  D7 = 1;
  RS  = 0;
  R_W = 1;
  while(D7==0){
  EN = 0;
  {}{}
  EN = 1;}
  EN = 0;
}

void ready_conversion(){
  EOC = 1;
  {}
  START = 1;
  {}{}
  START = 0;
  while(EOC!=0){}
}

void display_string(unsigned char *x){
  while(*x!=0){
    display_LCD(*x);
    x++;
  }
}

void compare(){

}
```

10 References:

https://www.geeksforgeeks.org/temperature-sensor-types-of-sensor/

https://www.digikey.com/en/blog/types-of-temperature-sensors

https://atlas-scientific.com/blog/humidity-sensor-types/

https://ro.mouser.com/

https://www.digikey.ro/

https://ro.farnell.com/