Structured Lab 3 - Introduction to Instrument Remote Control

1 Introduction

This lab is intended to give you an introduction to remote control of test and measurement equipment. In this lab, you will be focusing on the use of the EDU36311A Triple Output Programmable DC Power Supply and the EDU34450A $5\frac{1}{2}$ Digit Multimeter.

The deliverable of this lab will be a working datalogger program that you can use to record the temperature from the LM335 temperature sensor used in the microcontroller lab. There are many snippets of code which you will use to follow along with the lab and complete the deliverable. These are all available in the git repository here:

```
git@github.com:svhum/ece295-sl3.git
```

You can clone this directory from the command line by typing:

```
git clone https://github.com/svhum/ece295-sl3.git
```

2 Command Protocols and Methods

The four instruments in the ECE295 testing setup are all digital instruments and capable of control either via the front panel or with commands sent to the instruments via a computer. There are many instances in which you may want to have the test instrument (or instruments) controlled remotely or automatically by a script instead of having to do so manually. Communication with test instruments can be done with several different ports and protocols.

2.1 IEEE-488: The GPIB Protocol

The first standard protocol created for remote control of instrumentation was published by the IEEE (Institute of Electrical and Electronics Engineers) in 1978 and labelled as the IEEE-488 standard [1]. The standard encompasses both a software communication protocol and a hardware signaling method and standard cable type, commonly referred to as a GPIB (General Purpose Interface Bus). This communication protocol was created and standardized to allow engineers to work with one instruction set and cable for numerous different types on instruments. An image of the GPIB port like you might see on a spectrum analyzer or other piece of test equipment is shown in Figure 1.

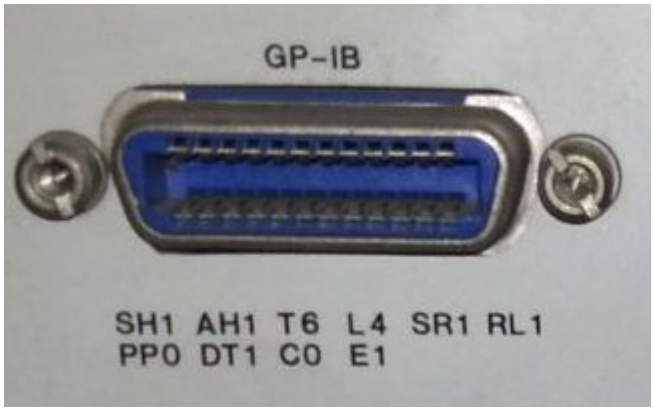

Figure 1: GPIB port on a spectrum analyzer (image from [2]).

Since the 1970s, the protocol has been revised and expanded several times to keep pace with technology developments.

2.1.1 IEEE-488 Commands

IEEE-488 standard defines a universal command set. These commands always start with an asterisk (*). Some important IEEE-488 commands are listed below:

- `*RST`

  This command resets instrument to factory default configuration.

- `*IDN?`

  This query returns instrument's identification string, which contains manufacturer name, model name and information on the firmware version

- `*CLS`

  This command clears event register and error queue.

2.2 Standard Commands for Programmable Instruments (SCPI)

IEEE-488 provided a standardized interface and communication protocol, but the commands used to control various instruments were still different from manufacturer to manufacturer. Standard Commands for Programmable Instruments (SCPI) were introduced in 1990 to provide a standard command set to go along with the GPIB standards. This allowed for a script written to take a voltage measurement on one manufacturer's multimeter, for example, to work identically if the piece of test hardware was switched out for another manufacturer's multimeter.

SCPI commands are defined as part of a hierarchy such that they are organized and easy to determine/find by a user with basic knowledge of what they want to do and a command manual or instrument programming guide.

2.2.1 SCPI Command Syntax

Note: Information in this section is taken from the SCPI Syntax and Style programming guide, Chapter 5 - Notation [3]. For more information on SCPI, you can check the Keysight EDU36311A Programming Guide, pages 11-15.

SCPI commands can be one of two types, **Set** commands and **Query** commands. **Set** commands tell the instrument to do something, such as setting a channel voltage or turning an output on or off. **Query** commands request a value from the instrument, such as the current output current or output state. It is also possible for a command to function as a **Set** and **Query** simultaneously, such as telling an instrument to switch to power measurement mode and return the current measured signal power.

Each command is defined by a keyword, which can have several sub-keywords. When you want to use a command described by one of the sub-keywords, you need to provide the top-level keyword followed by any nested sub-keywords, separated by colons.

Some sub-keywords that are commonly used are defined as default behaviours and therefore implicit. These sub-keywords can be typed or omitted when a command is written, the instrument will respond the same way in either case. These default commands are often listed in programming guides with square brackets [ ] surrounding them. For example, the sub-keyword `[:CW]` of the `FREQuency` command allows the continuous-wave frequency to be queried, but an SCPI instrument will also interpret just the `FREQuency` keyword to mean the continuous-wave frequency should be queried/set, since `[:CW]` is the default sub-keyword.

Many commands (particularly **Set** commands), will also require a parameter to be included with them. Parameters are separated from keywords by a space and can take many forms, from Boolean On/Off values to numeric inputs.

To keep code shorter, many keywords with long names are listed in a combination upper-case and lower-case format (e.g. `FREQuency`). When a keyword is written like this, an SCPI instrument will interpret both the full keyword and shortened capitals-only version identically. For example,

an SCPI instrument will interpret both the keyword `FREQuency` and its shortened form `FREQ` identically.

Table 1: Example SCPI commands [3]

Keyword	Parameter
`:FREQuency`	
`[:CW]`	<numeric value>
`:AUTO`	<Boolean>
`:CENTer`	<numeric value>
`:SPAN`	<numeric value>

An example of the SCPI commands related to frequency (like might be used in a spectrum analyzer) are shown in Table 1 to give you an idea of what the structure of these commands look like.

2.2.2 Setting and Querying Commands

Most commands have both a **Set** and **Query** mode. To send the **Query** version of the command, a question mark should be appended to the end of the command. Generally **Query** commands do not include parameters to be passed to the instrument, but they will result in data being returned to the querying computer.

As an example, to set the centre frequency of an instrument to 20 kHz, the SCPI command `FREQ:CENT 20000` would be sent. To check what the centre frequency of the instrument is set to, you would send `FREQ:CENT?`, and parse the response.

2.3 Virtual Instrument Software Architecture (VISA)

Virtual Instrument Software Architecture (VISA) is an application programming interface used to interface testing instruments with computers. It makes it simple to develop scripts to automate and control instruments to conduct complex tests that would be onerous and time consuming to do manually.

In this course, you will primarily focus on using PyVISA, a VISA implementation in Python. PyVISA acts as a overlay layer that contains many useful commands to simplify much of the low-level communication tasks that need to be done to interface with an instrument via IEEE-488 and SCPI. This allows for the development of control scripting that is portable between devices (normally with minor revisions due to specific hardware restrictions) and is independent of the interface used connect to the instrument (GPIB, LAN, USB, etc.).

For more information on PyVISA, you can visit this link - `https://pyvisa.readthedocs.io/en/latest/`.

2.3.1 Activity: Script Setup with PyVISA

Here, we'll cover a bit of information on how to install and configure PyVISA. Then, you'll create a basic script that initializes PyVISA and connects to an instrument.

Installation Before beginning, you'll need to install the following software, preferably in this order:

- Python 3 (64-bit required, v3.10 recommended)
- Keysight IO Libraries Suite (v6.2020 recommended)
- PyVISA

The laptops given to you to work with the ECE295 test equipment should already have all of these installed, so the following information is provided for reference only. You should be able to skip to the **Adding Instruments** section.

Python 3 is easy to install from the Python downloads page. When installing, make sure that you check the box to add Python to your computer's PATH.

The Keysight IO library is only available on Windows and Linux, and must be downloaded from the Keysight website. You may have to go through a few pages to reach the download link, and you may also need to input your name and email address. Once the download is finished, you will need to install the software. You can install all three programs, but you only need the IO Libraries Suite for PyVISA to work. Note that you may also need to install the .NET 3.5 framework as part of the installation process.

Finally, to install PyVISA, just open a command or terminal window and type "pip install -U pyvisa". Note that this will only work if you added Python to your computer's PATH as described earlier. Otherwise, you will need to run "pip install -U pyvisa" from the folder where pip.exe is installed (probably in a subfolder of your Python installation).

After the installation is complete, you may need to restart your computer before everything will work properly.

Adding Instruments The next step for setup is to add the instruments to the Keysight IO Manager.

1. First, begin by making sure that the PC you're using is plugged into the ethernet switch on the test equipment rack. If using one of the course laptops, make sure to be careful when plugging in the laptop, as the port is very tight and the cables will sometimes stick. Call over your TA to assist you if you need help.
2. Next, open the Keysight IO Manager (also called Keysight Connection Expert), and press the **+ADD** button, then **LAN Instrument**. Note that the 3 instruments shown in the screenshot in Figure 2 were previously set up. If your PC shows 4 instruments there, it is already set up and you can skip to step 4. If there are no instruments, continue with this one.

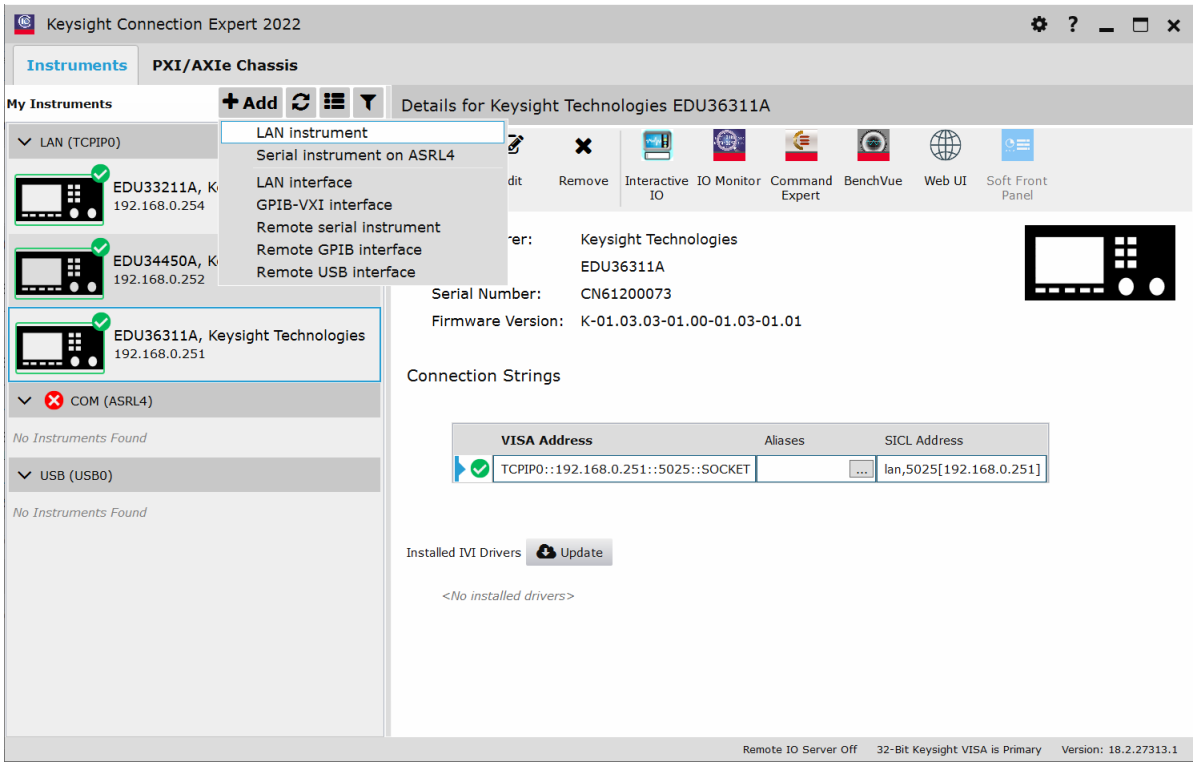

Figure 2: Keysight Connection Expert: adding a LAN instrument.

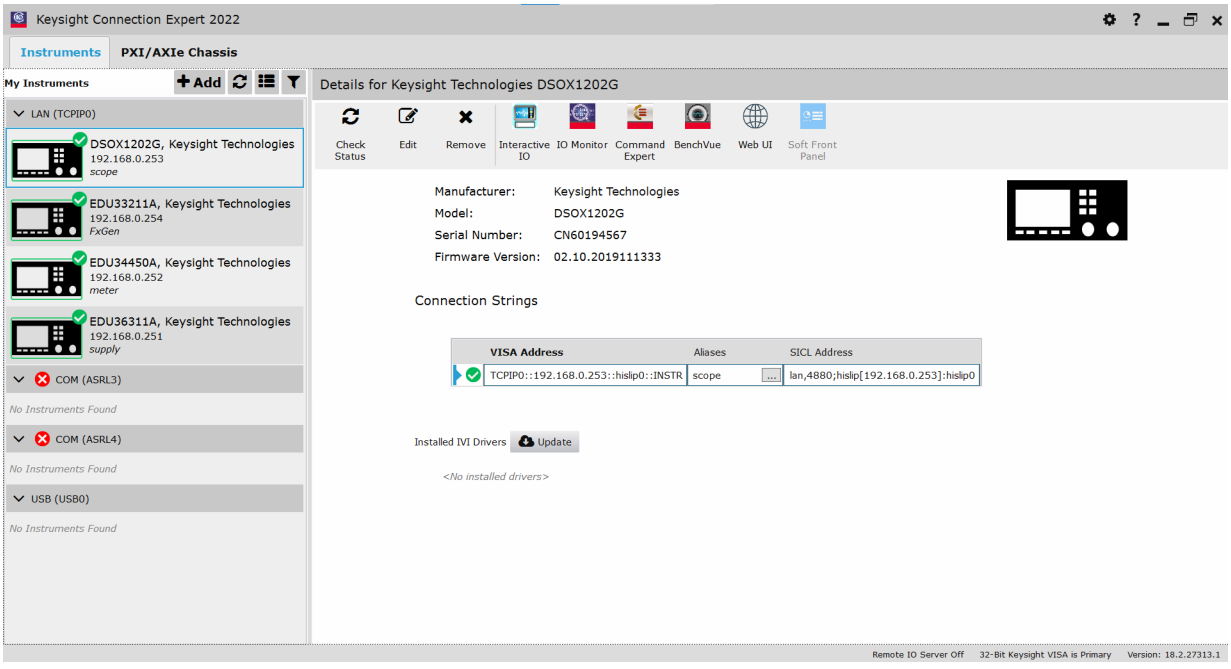

Figure 3: Keysight Connection Expert: four instrucments

3. You should see four instruments in the new window that pops up, as shown in Figure 3. Check the boxes next to each instrument and click the **Okay** button.

4. Finally, make note of the VISA Addresses of the four instruments. They should be identical aside from the IP address of each being different.

Testing your Installation First, choose a Python IDE of your choice. Visual Studio Code works well, but you can also use IDLE, SublimeText, PyCharm, or even a combination of Notepad++ and a terminal/command window.

Once your IDE is configured properly and open, create a new python file and copy the following code into it.

```
# This program makes sure that libraries are all installed correctly
import pyvisa as visa
rm = visa.ResourceManager(r'C:\WINDOWS\system32\visa64.dll')
print(rm)
print(rm.list_resources('TCPIP0::?*'))
```

Setting up an instrument Now that everything's working, you can create a script to connect to an instrument and execute a few basic commands. It is included in the files supplied in the git repository as "BasicScript.py".

3 Controlling the EDU3611A Power Supply

All four instruments you were introduced to in SL2 have both USB and Ethernet LAN ports for communication, and can be controlled via IEEE 488 instructions and SCPI with the help of the VISA API.

Note that the communication ports are both located on the back of the instruments, an example of which is shown in Figure 4. There is another USB-A port on the front of the instruments, but this is only usable with USB storage devices. The instruments cannot communicate with a computer through these front panel ports.

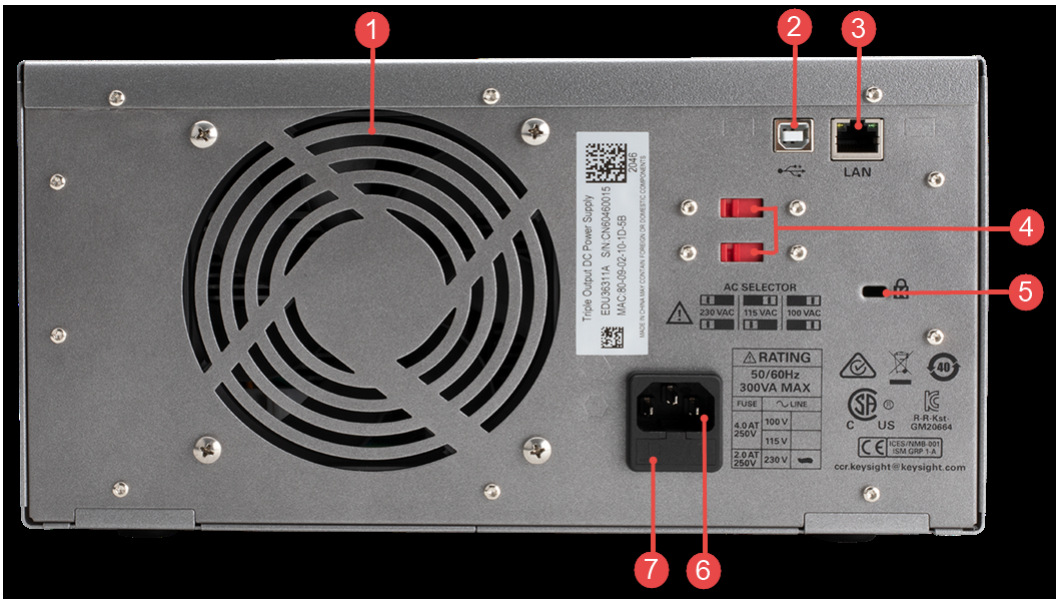

Figure 4: Communication USB-B and LAN ports on the back of the instruments (2 & 3, respectively).

In this course, we will focus on using the Ethernet LAN connections on the instruments to control them. In the test instrument rack, the four instruments and the laptop have all be manually assigned **static IP addresses**. This means that they will not change at any point unless you or another user does so manually. Static IP Addresses make it a bit easier to set up our Python scripts for the instruments. Note that since the instruments are hardwired into an Ethernet switch and not connected to the wider University of Toronto network, you will only be able to access them from a computer plugged into the same switch as the instruments. Table 2 lists the IP addresses of the four instruments as configured.

Table 2: IP Addresses of the four testing instruments.

Instrument	**IP Address**
Power Supply	192.168.0.251
Digital Multimeter	192.168.0.252
Oscilloscope	192.168.0.253
Function Generator	192.168.0.254

3.1 Selected Commands for the EDU3611A Power Supply

In this section, we will introduce `VOLT` and `CURR` commands. These two commands are useful to implement the CC and CV control for the power supply. Below are some commands, along with short descriptions, which may be useful for your project.

- `VOLT:PORT 5, (@1)`

  Set the level at which over-voltage protection trips to 5V for channel 1. The command for channel 2 is similar.

- `VOLT:PORT:CLE, (@1)`

  Clear the latch that disables channel 1 due to an occurrence of over-voltage protection.

- `VOLT 1, (@2)`

  Set the output voltage level to 1V on channel 2. The command for channel 1 is similar.

- `VOLT? (@2)`

  Return the output voltage level for channel 2.

- `CURR:PROT:STAT ON, (@1)`

  Enable the current protection for channel 1.

- `CURR 1, (@2)`

  Set the output current level to 1A on channel 2.

- `CURR? (@2)`

  Return the output current level for channel 2.

You can find more commands for the power supply from EDU36311A DC Power Supply's Programming Guide posted in Quercus, and also provided in the files cloned from the repository.

3.2 Sample code

Here is an example snippet which shows some of the commands mentioned above. It is included in the repository files as "PowerSupply.py".

4 Controlling the EDU34450A Digital Multimeter

You are likely quite familiar with the portable multimeter from previous electronics projects. They are quite handy for diagnosing problems with circuits. The bench-top multimeter performs many of the same functions but is more precise and feature-packed. Here, we will go over the main subsystems of the digital multimeter you'll need for the data logger project.

4.1 Selected Commands for EDU34450A Digital Multimeter

In this section, we will introduce `CONF` and `MEAS` commands for controlling digital multimeter. The following selected SCPI commands are used to configure and implement measurements. In the datalogger program, we will use these commands to record information from a temperature sensor.

- `CONF:VOLT:DC`

  Configure the digital multimeter for DC voltage measurements on primary display without actually performing a measurement.

- `CONF:SEC:CURR:DC`

  Configure the digital multimeter for DC current measurements on the secondary display without actually performing a measurement.

- `CONF:FREQ`

  Configure the digital multimeter for frequency measurements on the primary display without actually performing a measurement.

- `READ?`

  This usually comes after the `CONF` command to return readings for the configured measurement.

- `INIT`

  Set the digital multimeter to "wait for trigger" state.

- `FETC?`

  Return the readings for the configured measurement. `INIT` and `FETC?` commands usually come together after the `CONF` command to return readings for the configured measurement.

- `MEAS:VOLT:DC?`

  Measure DC voltage on the signal from the primary display.

- `MEAS:SEC:CURR:DC?`

  Measure DC current on the signal from the secondary display.

You can find more commands for the DMM from EDU34450A Digital Multimeter's Programming Guide posted on the course website.

4.2 Sample Code

There is an example snippet which configures and measures a DC voltage from primary display of the DMM included in the repository files as "Multimeter.py".

4.3 Exercise: Measure the Voltage Output from the Power Supply

Now that you're familiar with the SCPI commands for the multimeter, it is time to use them for something useful. Try to measure the voltage output from the power supply when it is set to 5V. Use the commands above to remotely configure the multimeter and print the resulting voltage measurement using the `print()` command in python. When you're done, show your TA.

5 Activity: Datalogger Implementation

Now let's work on the datalogger program that you can use to record information from the LM335 temperature sensor used in the microcontroller lab.

LM335 is a simple temperature sensor with an operating range of -40°C to 100°C. It is widely applied to different areas such as embedded systems and power supplies [4]. A diagram of LM335 is shown below:

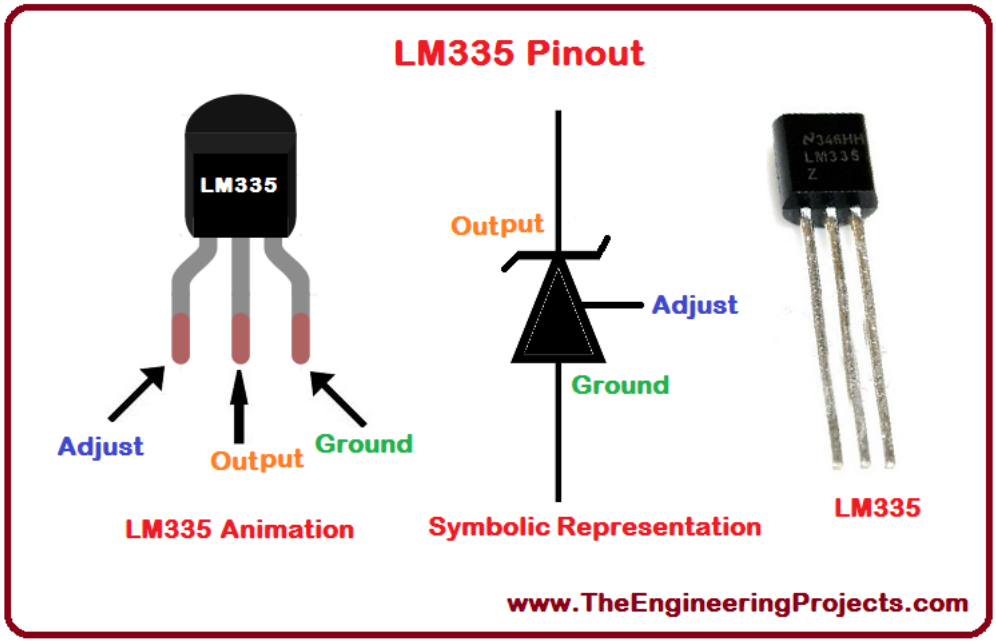

Figure 5: LM335 pinout [4]

As shown above, LM335 has three pins. The "adjust" pin is used to adjust calibration. Essentially, LM335 temperature senor is a Zener diode whose output voltage is related to the absolute temperature as follows [5]:

$$V_{out}(T) = V_{out}(T0) \times \frac{T}{T0} \tag{1}$$

where $T0$ is the reference temperature and nominal $V_{out}(T0)$ equals to $T0 \times 10$ mV/K. For example, at reference temperature of 25°C, nominal $V_{out}(298K) = 10$ mV/K $\times\, 298K = 2.98V$. Then the output voltage of LM335 at 40°C equals to 2.98 V $\times \frac{313\text{ K}}{298\text{ K}} = 3.13$ V.

Now let's see a simple temperature sensor circuit without calibration (i.e. "adjust" pin is kept floating) in Figure 6. Use a breadboard to construct this circuit. You can use the DC supply to provide the 5V supply voltage and the digital multimeter to check if you get the expected output voltage. Show your circuit setup to a TA before moving to the next step.

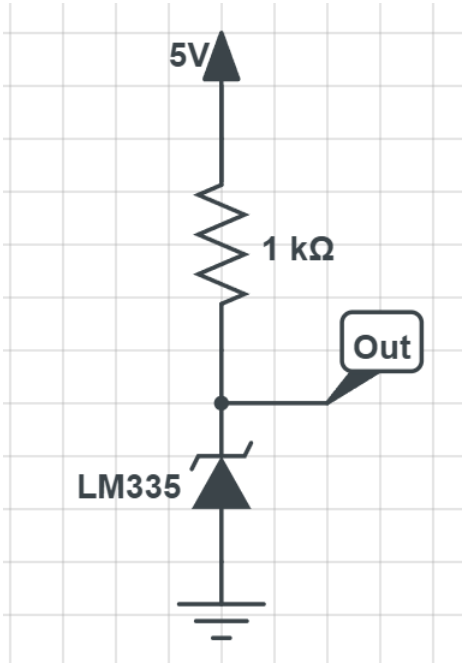

Figure 6: LM335 circuit

If you observe the multimeter display, you can see the output voltage continuously changes. Now let's record the temperature variations over a certain time period using the commands learned from previous sections. Some skeleton code for implementing the datalogger is included in the repository files as "DataloggerToComplete.py". However, there are spaces marked "TODO" in which you will have to fill in your own code.

At the end you will be able to plot the temperature in the room over 5s with 0.25s intervals. Show your TA the temperature versus time plot.

References

[1] "Introduction to IEEE-488: GPIB interface and communication protocol." [Online]. Available: https://www.omega.ca/en/resources/gpib-communication

[2] "Spectrum analyzer GPIB port (image)." [Online]. Available: https://upload.wikimedia.org/wikipedia/commons/8/85/IEEE488portcapabilities.jpg

[3] *Standard Commands for Programmable Instruments (SCPI), Volume 1: Syntax and Style*, Online, 1999.

[4] "Introduction to LM335." [Online]. Available: https://www.theengineeringprojects.com/2017/08/introduction-to-lm335.html

[5] "The LM335 temperature sensor." [Online]. Available: http://web.mit.edu/rec/www/workshop/lm335.html