Software for Inverting and Non-inverting Operational Amplifiers

Anıl Karaca

Table of Content

VARIABLES

R1: This variable is a 5 bytes long array. It stores the value of the given input R1 resistor digit by digit.

R2: This variable is a 5 bytes long array. It stores the value of the given input R2 resistor digit by digit.

Gain: This variable is a 5 bytes long array. It stores the value of the given input gain(A) digit by digit.

Note: "R1", "R2" and "Gain" arrays may also contain "." and "$" characters, since "." represents float numbers and "$" indicates that the number is done.

GainMinusOne: This variable is 5 bytes long. It stores the value of the given input gain(A) minus one digit by digit.

lenR1: This variable is a word long. It stores the length of integer part of R1.

lenR2: This variable is a word long. It stores the length of integer part of R2.

lenA: This variable is a word long. It stores the length of integer part of gain(A).

dotFlag: This variable is a byte long flag. Its value is 1 if "." character is present and 0 if it is not.

invGainFlag: This variable is a byte long flag. Its value is 1 for gain calculations for inverting op-amp circuitry and 0 for non-inverting op-amp circuitry.

escFlag: This variable is a byte long flag. Its value is 1 if "escape" key is pressed and 0 if it is not.

result: This variable is a word long. It stores the integer part of the result.

remainder: This variable is a word long. It is an intermediate variable that holds the remainder of the division calculation made.

afterPoint: This variable is a byte long. It stores the 1st digit after the decimal point of the result.

afterPoint2: This variable is a byte long. It stores the 2nd digit after the decimal point of the result.

ten: This variable is a word long. I actually use it as a constant rather than a variable. It holds the value 10.

PROCEDURES AND MACROS

Macros

print(msg): This macro prints the given message(string) on the emulator screen by making use of INT 21h/09h.

printVideo(char): This macro prints the given character on the emulator screen(video mode) by making use of INT 10h/09h.

PUTC(char): This macro prints the given character on the emulator screen by making use of INT 10h/0Eh.

label(x, y, char1, char2): This macro prints the char1 and char2 on the given (x, y) coordinates of the screen(video mode) by making use of INT 10h/02h and INT 10h/09h.

labelA(x1, y1, x2, y2): This macro labels R1 and R2 resistors on the plotted circuitry for option a/A(non-inverting op-amp) by making use of INT 10h/02h. (x1, y1) is for the R1 resistor and (x2, y2) is for the R2 resistor. R1 resistor is always 1kohm, so we choose R2 accordingly.

labelB(x1, y1, x2, y2): This macro labels R1 and R2 resistors on the plotted circuitry for option b/B(inverting op-amp) by making use of INT 10h/02h. (x1, y1) is for the R1 resistor and (x2, y2) is for the R2 resistor. R1 resistor is always 1kohm, so we choose R2 accordingly.

labelCD(x1, y1, x2, y2): This macro labels R1 and R2 resistors on the plotted circuitry for option c/C(non-inverting op-amp) by making use of INT 10h/02h. (x1, y1) is for the R1 resistor and (x2, y2) is for the R2 resistor. R1 resistor is always 1kohm, so we choose R2 accordingly.

Procedures for general operations

resistanceNonInverting: This procedure handles the general operations for option a/A. It is basically the main procedure for the option a/A where I call all the other necessary procedures and macros.

resistanceInverting: This procedure handles the general operations for option b/B. It is basically the main procedure for the option b/B where I call all the other necessary procedures and macros.

gainNonInverting: This procedure handles the general operations for option c/C. It is basically the main procedure for the option c/C where I call all the other necessary procedures and macros.

gainInverting: This procedure handles the general operations for option d/D. It is basically the main procedure for the option d/D where I call all the other necessary procedures and macros.

Procedures for making calculations

calculateGain: Calculates the R2/R1 ratio(which is also equal to gain(A) for inverting op-amp) by using given R1 and R2 values, and then it stores the result in "result.afterPoint afterPoint2" format.

calculateRatio: Calculates R1/R2 ratio by using the given gain value(A), and then it stores the result in "result.afterPoint afterPoint2" format.

Procedures for printing the results on the screen

printResult: This procedure prints the number stored in the AX register. Allowed values are from 0 to 65535(FFFFh), since 65535 is the greatest value a register can hold.

Procedures for getting inputs

getInputMenu: This procedures allows user to select between different operations such as option a/A, b/B, c/C or d/d.

It does not allow user to enter any invalid inputs or inputs that are longer than a single character and it also allows user to delete inputs as well.

getInputResistor: This procedure gets the R1 and R2 resistor values(kohm) (up to 5 digits since 65535 is 5 digits long) from the user by making use of INT 16h/00h.

It does not allow user to enter any invalid inputs(except “.” character, since it represents a float number) and it also allows user to delete inputs as well.

It also does not allows user to enter “0” or “.” as the first input since it may cause some trouble with dividing later on.

This procedure stores the R1 value in a array called "R1" and R2 value in a array called "R2" digit by digit. When the inputting process is done it puts "$" at the end (unless the input is already 5 digits long) to indicate that the array is done changing.

getInputGain: This procedure gets the gain(A) values (up to 5 digits since 65535 is 5 digits long) from the user by making use of INT 16h/00h.

It does not allow user to enter any invalid inputs(except "." character, since it represents a float number) and it also allows user to delete inputs as well.

It also does not allows user to enter "0" or "." as the first input since it may cause some trouble with dividing later on.

This procedure stores the gain(A) value in a array called "Gain" digit by digit. When the inputting process is done it puts "$" at the end (unless the input is already 5 digits long) to indicate that the array is done changing.

Procedures for formatting inputs

formatResistor: This procedure formats the input stored in the "R1" array such that its value is stored in the SI register, and it also formats the input stored in the "R2" array such that its value is stored in the DI register.

This operation is achieved by constantly multiplying the digit in question by the powers of 10.

formatGain: This procedure basically does the same operation as the previous procedure called "formatResistor" but it does this operation for the input stored in the "Gain" array.

formatResult: This procedure stores the result of the operation a/A in an array called "GainMinusOne" digit by digit so I can use it later to print it on the screen in the video mode.

Procedures for changing the colors of the pixels in different directions

All the following procedures basically does the same thing, which is looping until the end point SI is reached. You can see the names of this procedures and the directions they are responsible for below.

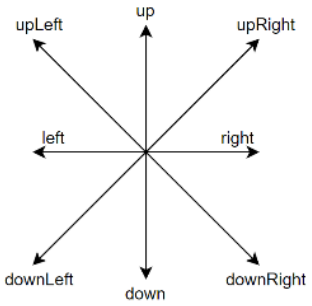


Figure 1

Procedures for displaying different components on the screen(video mode)

ground: This procedure creates a ground component consisting of 3 lines which are aligned vertically by making use of the direction procedures.

opamp: This procedure creates an op-amp component without "+" and "-" signs by making use of the direction procedures.

resistor: This procedure creates a resistor component by making use of the direction procedures.

Procedures for general operations(video mode)

drawInverting: This procedure handles the operation of displaying an inverting op-amp circuitry by making use of the procedures I have mentioned above. The way that this procedure works is basically just setting the endpoint(SI) and then calling the proper procedure until we are done printing every component and cable we need.

drawNonInverting: This procedure handles the operation of displaying a non-inverting op-amp circuitry by making use of the procedures I have mentioned before. The way that this procedure works is basically the same as the previous one(drawInverting).

HOW DIFFERENT OPERATIONS TAKE PLACE

In this part of the report I would like to point out more specifically how different operations(such as option a/A, b/B, c/C and d/D) take place.

a/A

Now let's say you want to calculate the R1/R2 ratio of a non-inverting op-amp circuitry by inputting the gain(A) of the system.

Firstly when you execute the program a welcome message and explanations of different operations appear on your screen, and then the program prompts you to select an operation. At that stage program only allows valid inputs to take place such as a/A, b/B, c/C or d/D. It also does not allow user to enter more than a character. But if user would like to delete the character on the input field, they could use the "backspace" key for this particular reason.

Once the user has pressed enter the program checks if the input was the "escape" key, if it is indeed the "escape" key the program terminates by printing a message on the emulator screen. If the input character was not the "escape" key the program directs the program according to the user input.

Let's say that the input was either "a" or "A", in that case the program is directed to the procedure called "resistanceNonInverting". In this procedure program firstly prompts user the input the gain(A) of the non-inverting op-amp circuitry by calling the "getInputGain" procedure. This "getInputGain" procedure only allows user to enter valid inputs. Some invalid input cases are listed below:

- Entering more than 5 digits.
- Entering "0" or "." as the first input character.
- Entering any character out of the (0-9) range, except the "." character.
- Entering "." character more than one time.

There are also some special I want to mention such as:

- If the user inputs "escape" key the program terminates by printing a message on the emulator screen.
- If the user inputs "backspace" key the program deletes the latest input (if there is any). This operation is done under the label called "delInput3".
- If the user inputs "enter" key the inputting process is done. This operation is done under the label called "endInputA".
- If the user inputs "." character the program updates the variable called "dotFlag" which indicates if the input contains "." character or not. This operation is done under the label called "dotFlag3".

Once the inputting process is done the "resistanceNonInverting" procedure formats the input stored in the array called "Gain" by calling the "formatGain".

<u>*Note:*</u> *This formatted gain value(A) is stored in the SI register.*

Now program can calculate the R1/R2 ratio by making use of the formula below.

$$\frac{R_1}{R_2} = \frac{1}{A-1}$$

But first we decrement the gain(A) by 1 and then format this result back to an array called "GainMinusOne" so that I can use this array later on for printing purposes. Finally program can obtain the R1/R2 ratio by calling the procedure called "calculateRatio". After the calculation is done "resistanceNonInverting" prints the result on the screen by calling "printResult" for the integer part of the result and by using "PUTC" for the part after the decimal point.

After this "calculation and printing the result on the screen part" is done the program suggests proper values for R1 and R2 resistor according to the calculations made earlier before prompting user to press any key(except the "escape" key of course) in order to begin the process of visually displaying the non-inverting op-amp circuitry with the obtained results.

This visually displaying the circuitry process is done by using the macros called "label", "labelA" and the procedure called "drawNonInverting".

<u>*Note:*</u> *Since the explanations for different operations would be too similar, from now on I will only point out the differences between operations with respect to the operation a/A.*

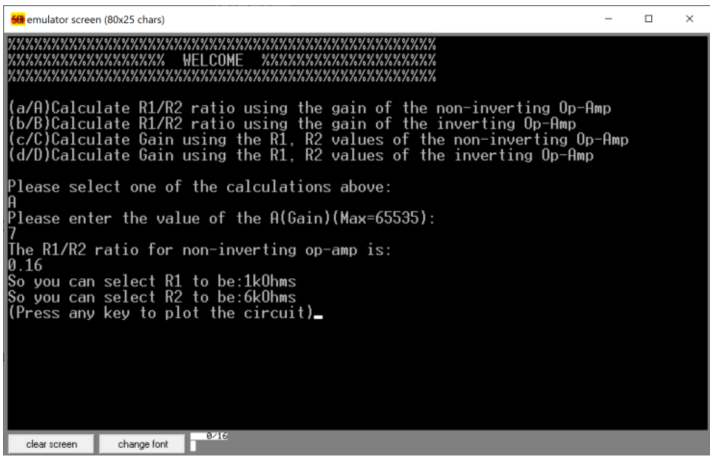


Figure 2

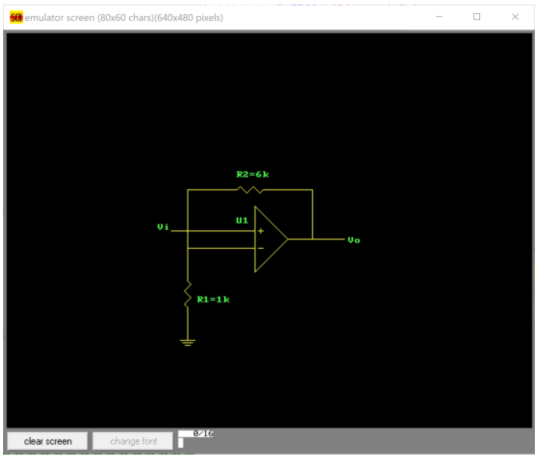


Figure 3

<u>**b/B**</u>

Now let's say you want to calculate the R1/R2 ratio of an inverting op-amp circuitry by inputting the gain(A) of the system.

<u>Differences with respect to operation a/A:</u>

- The general operation is handled by the procedure called "resistanceInverting".
- Even though we use the same procedure for getting the input, this time program prints a "-" character before taking the input since the gain(A) value of an inverting op-amp circuitry can not be a pozitive integer.

- This time program calculates the R1/R2 ratio without decrementing the gain(A) value by 1 since the formula is a bit different for inverting op-amp curcuits such as:

$$\frac{R_1}{R_2} = \frac{-1}{A}$$

- The last difference between the operations b/B and a/A is the way program visually displays the circuitry, since the inverting op-amp circuits and the non-inverting op-amp circuits are different. This time we use the macro called “labelB” instead of “labelA” and procedure called “drawInverting” instead of “drawNonInverting”.

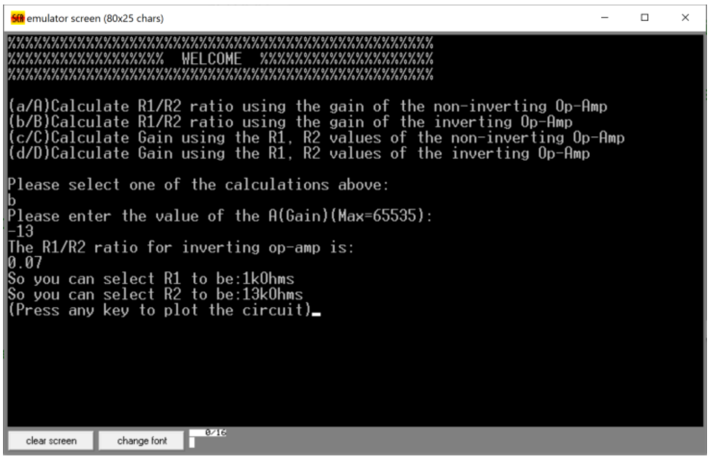


Figure 4

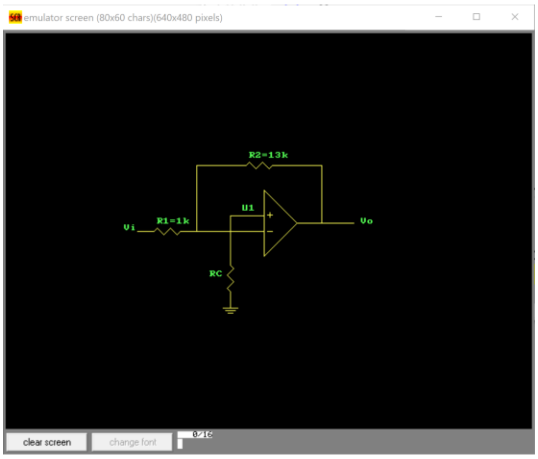


Figure 5

c/C

Now let’s say you want to calculate the gain of a non-inverting op-amp circuitry by inputting the R1 and R2 resistor values of the system.

<u>Differences with respect to operation a/A:</u>

- The general operation is handled by the procedure called “gainNonInverting”.
- This time we use a different procedure called “getInputResistor” for getting the R1 and R2 resistor values from the user. This procedure pretty much does the same thing as the “getInputGain” procedure but this time the inputting process takes place 2 times in a row for R1 and R2. And these inputs are storedin arrays called “R1” and “R2” respectively.
- This time we use a different procedure called “formatResistor” for formating the R1 and R2 resistor values. This procedure pretty much does the same thing as the “formatGain” procedure but this time the formating process takes place 2 times in a row for R1 and R2.

<u>Note:</u> Formatted R1, R2 values are stored in SI and DI registers respectively.

- This time program calculates the gain value(A) by using the formula below:

$$A = 1 + \frac{R_2}{R_1}$$

- Since both options a/A and c/C deal with operations of non-inverting op-amp circuitries the procedures we use for them are the same but one of the macros used is different. This time we use the macro called “labelCD” instead of “labelA”.

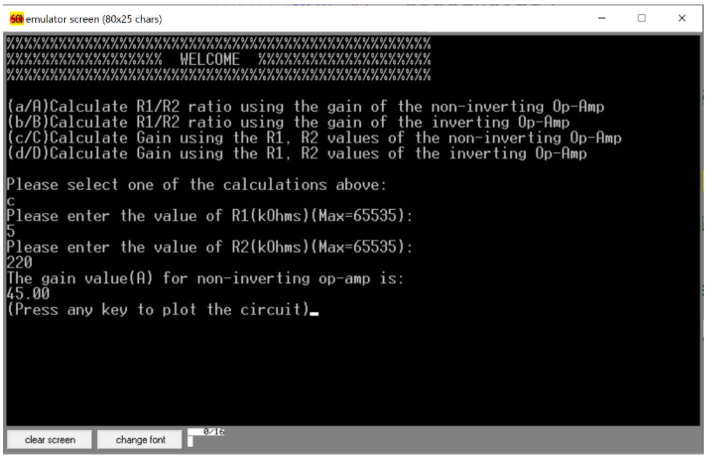


Figure 6

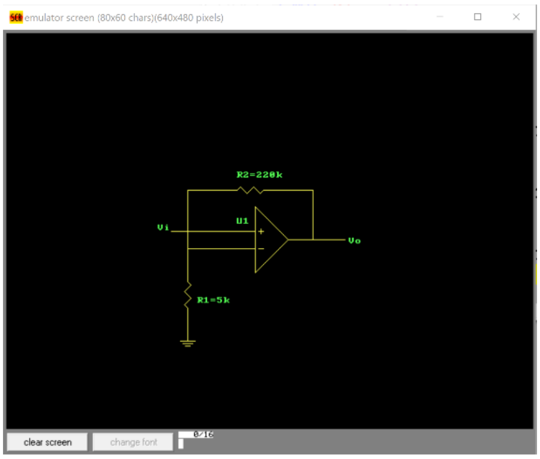


Figure 7

d/D

Now let's say you want to calculate the gain of an inverting op-amp circuitry by inputting the R1 and R2 resistor values of the system.

Differences with respect to operation a/A:

- The general operation is handled by the procedure called "gainInverting".
- This time we use a different procedure called "getInputResistor" for getting the R1 and R2 resistor values from the user, which is the same procedure we use to get inputs for option c/C.
- This time we use a different procedure called "formatResistor" for formating the R1 and R2 resistor values, which is the same procedure we use to get inputs for option c/C.

Note: Formatted R1, R2 values are stored in SI and DI registers respectively just like in option c/C.

- This time program calculates the gain value(A) by using the formula below:

$$A = -\frac{R_2}{R_1}$$

- The last difference between the operations d/D and a/A is the way program visually displays the circuitry, since the inverting op-amp circuits and the non-inverting op-amp circuits are different. This time we use the macro called "labelCD" instead of "labelA" and procedure called "drawInverting" instead of "drawNonInverting".

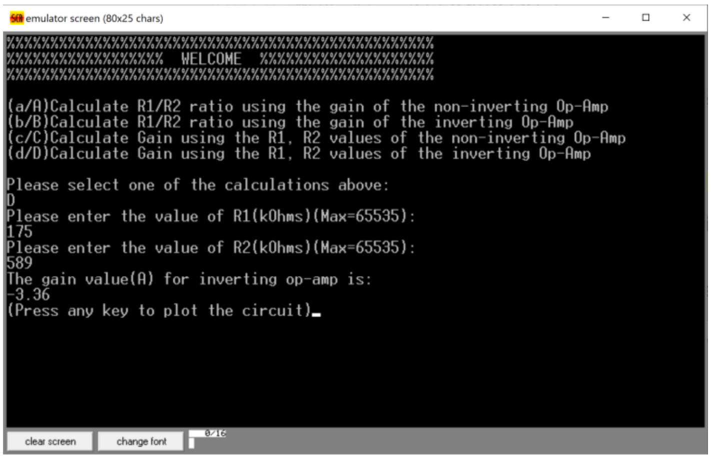


Figure 8

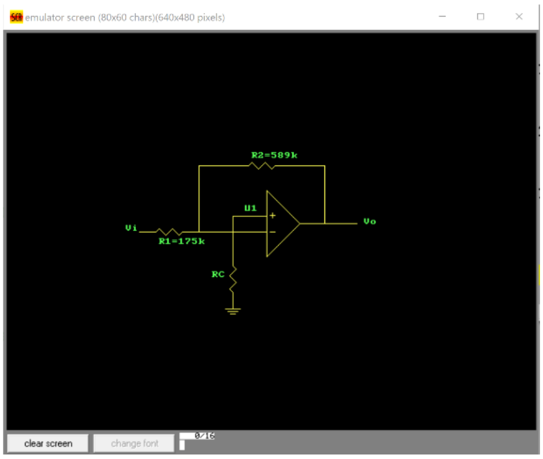
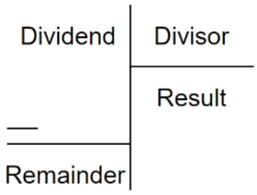


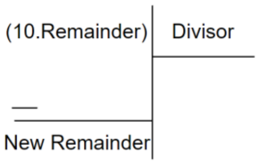
Figure 9

IMPORTANT THINGS TO POINT OUT

- Whenever the user presses the “escape” key, the program terminates.
- Even though the program allows user to input float number, the program only works with its integer part to approximate the result.
- For operations a/A and b/B we always select R1 to be 1kohms and then calculate R2 value accordingly.
- I would like to point out the math behind the way I implemented division operation used in “calculateGain” and “calculateRatio” procedures.
  As you can see on the figure below we can directly obtain the integer part of the result but in order to obtain the 2 figures(“afterPoint” and “afterPoint2”) after the decimal point needs some math such as:



$$afterPoint = \frac{10.Remainder}{Divisor}$$

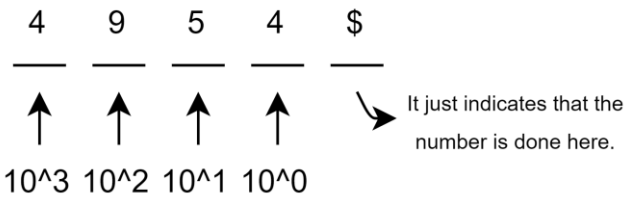


$$afterPoint2 = \frac{10.(New\ Remainder)}{Divisor}$$

Note: I could carry on this calculations but I thought 2 digits after the decimal point would be sufficient enough.

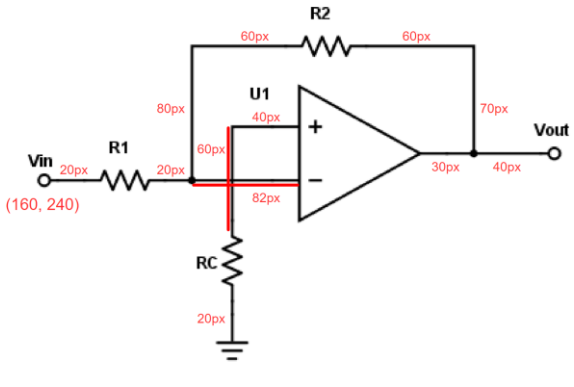
- I also would like to point out the logic behind deleting an input form the emulator screen. Whenever user presses the “backspace” key the program firstly moves cursor to the left by one. And then, it prints an empty space(“ ”) to overwrite the previous input. Finally the program moves the cursor back by one again to allow user to continue from that point on. You can see this implementation under different labels such as “delInput”, “delInput1”, “delInput2” and “delInput3”.

- The way that “formatResistor” and “formatGain” procedures format the input stored in “R1”, “R2” and “Gain” arrays is done by maintaining the different exponentials of 10. For example let’s say we are formatting the “R1” array that stores a arbitrary but valid input.



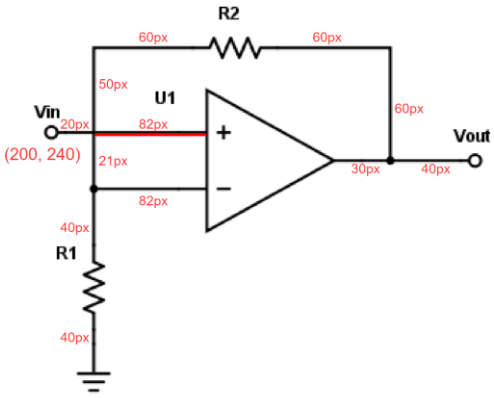
- One other thing to point out is the way the program prints the circuits pixel by pixel, which is done in the procedures called “drawInverting” and “drawNonInverting”. I would like to point it out because with all the coordinates and decimal values it can be hard to read the code of these procedures.
  In both of this procedures we set the end point SI and then call one of the direction procedures to change the color of the pixels until we reach the end point SI. Program does this process for every single connection on the circuit.
  I could also use macros instead of procedures for different directions but, since we use this operation very frequently, declaring them as macros would significantly increase the compilation time and it would reduce the overall performance.

  The pixel values for both inverting and non-inverting op-amp circuitries are as follows:



Inverting Op-amp

Figure 10

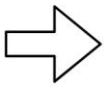


Non-inverting Op-amp

Figure 11

- The very last thing I want to point out is the arguments of the macros called “label”, “labelA”, “labelB”, “labelCD”. I call these macros in order to label the printed circuits. When you look at the way I call them, you will see that I call them with some numbers as arguments. These numbers are actually meaningful because the resolution of the screen of the emulator screen in video mode is (640x480 pixels) or (80x60 characters). This means that each character the program prints on the screen takes about (8x8 pixels). I have used this logic to determine the arguments of these macros.
  For example let me demonstrate this by using the code under the “resistanceNonInverting” procedure:

```
label 23, 29, "V", "i"
label 29, 40, "R", "1"
label 35, 21, "R", "2"
label 52, 31, "V", "o"
label 35, 28, "U", "1"
labelA 31, 40, 37, 21
```



(23, 29) in characters = (184, 232) in pixels

(29, 40) in characters = (232, 320) in pixels

(35, 21) in characters = (280, 168) in pixels

(52, 31) in characters = (416, 248) in pixels

(35, 28) in characters = (280, 224) in pixels

(31, 40) in characters = (248, 320) in pixels

(37, 21) in characters = (296, 168) in pixels

REFERANCES

- https://stackoverflow.com/
- https://jbwyatt.com/253/emu/asm_tutorial_01.html
- https://jbwyatt.com/253/emu/8086_instruction_set.html
- https://jbwyatt.com/253/emu/8086_bios_and_dos_interrupts.html#top1