# Program 3

## I. Problem Statement

This program is a number converter written in the MIPS assembly language with binary, decimal, and hexadecimal options. The input is read from a file which the user chooses and the output file is always ‘testout.txt’. The program accepts a file name input, then opens that file, reads it, and then produces the following output for each line: Input type, a colon and a space, the value input, a semicolon and a space, a output type, a colon and a space, and then finally the output value. The program then prints the output to both the console and the output file, which is a hardcoded ‘testout.txt’ file. This program is useful for teaching people more about the MIPS assembly language, such as file input and output, Strings, String addresses, bit masking, return addressing and procedure calling, number conversion between binary, decimal, and hexadecimal, as well as how integers are stored in registers.

## 2. Approach

The program was completed using file input/output MIPS assembly commands, procedure branching, bit masking, accessing String addresses, and ASCII character codes. A few of the important procedures include main, which does the job of opening and closing both the input and output files as well as calling the other procedures; loNew which reads a single character from the inputFile and records it into output buffer; printTheOutput which prints out the results of the conversion to both the console and the output file; prtany and printDecimal accomplish the jobs of printing the output value in binary, hexadecimal, and decimal. The program uses jump as well as jump and link commands to maneuver through these various procedures.

This program can open and read files thanks to the system service commands provided by the MIPS assembly language. By limiting the number of characters read by the syscall to 1, the program can read a file of infinite length, though you are only reading the file one character at a time, and you must call the loNew procedure and perform a syscall every single time you want to read a new character. loNew also saves the character read into a buffer with a size of 100000 so that the buffer can be used to write the output to a file.

I ran into multiple problems when writing this program that were difficult to wrap my head around. One of which was the fact that Windows text files insert a “Carriage Return” at the end of every line right before inserting a “New Line”. To solve this problem, I implemented a loop at the end of printing the output, which kept reading characters until it reached a “New Line” character, that way I was able to skip past the Carriage return and keep printing my output on the proper line. This loop also handled another problem, which was making sure the output file was replicable, meaning that you could use the output file as an input to produce the same output file, even though the input file for both runs were different, with one of them already containing the resultant output line.

For printing the resultant output in binary and hexadecimal I utilized an interesting technique called ‘bitmasking’ which is used to access specific bits in a byte of data. I utilized this to isolate digits in the inputted value, then get the ascii digit corresponding to that digit, record it, then slide the next number digit into the lower bits. This method is then repeated as needed.

## 3. Solution

The assigned tasks were to write a program in the MIPS assembly language which reads a inputed file and produces a number conversion output based on the request lines in the input file. By using the afformentioned techniques mentioned earlier in this report, I was able to produce the desired output to both the command line, as well as a new file titled “testout.txt”. It doesn’t matter whether you enter in a file with the results already calculated or one without, it will still produce the desirable results if the request lines follow the correct formating.

This program is executable via console by entering the command line: **“Java -jar mars.jar pgm3.asm”** into the console. Afterwards, enter in the desired input file name. Once the results are calculated, the output is displayed and then saved in a new file called “testout.txt”. If you’d like you can then repeat this process by restarting the program and then entering in “testout.txt”, and the program will produce the exact same “testout.txt” file. This can be done an infinite amount of times. Figure 1 shows how the program was built and executed. The sample text “inputPgm3.txt” was used for the example.

## Appendix

Text

Description automatically generated

*Figure 1 – Command line for running the program. As you can see here, the output is printed out in the command line as well as the file output. It is important to note that whatever file you use as an input, it must be placed in the same directory as your Mars.jar file. Failure to do so will result in an error message. The output filename is called testout.txt and is placed in the same directory as the Mars.jar file.*

Text

Description automatically generated

*Figure 2 – This is the file output titled, “testout.txt”. The inputted request lines are to the left of the “==” signs and have kept their original values. The output lines are printed to the right of the “==” signs and contain the requested information. Not shown in this figure is the fact that the program can actually read spaces in the requested input lines just fine. It looks a little bit ugly, but all the information is correct and most of the unaligned text values are due to the unaligned input values.*