Binary, Hexadecimal and Decimal Conversions

I. Problem Statement

Batch Number Conversion:

1. Prompt the user for the file name, then input and open it.
2. Each request is a single line as follow:
   1. Input type (one character, b: binary, d: decimal, or h: hexadecimal)
   2. Input length (1 to 32, in decimal)
   3. Output type (one character, B: binary, D: decimal, or H: hexadecimal)
   4. Colon (:) and space
   5. Input value in the appropriate format including optional sign. Whitespace may be used for character grouping. If used, it does not count as part of the input length.
3. Output the data as follows:
   1. Input type (one character, b: binary, d: decimal, or h: hexadecimal)
   2. Colon (:) and space
   3. Value input
   4. Semicolon (;) and space
   5. Output type (one character, B: binary, D: decimal, or H: hexadecimal)
   6. Colon (:) and space
   7. Output value
4. Clean up after all input lines have been processed.

Your test file, of arbitrary length, must include all combinations of conversions including implicitly and explicitly signed values. Binary and hexadecimal output should always include thirty-two bits.

Decimal output must not include leading zeroes.

Upload code, your test file, and report here. MARS is not required as part of the submission.
Bonus points:

5 points for insertion of spaces between every grouping of four characters in binary and hexadecimal outputs.

5 points for insertion of commas between grouping of three characters, for decimal outputs, counting from the right.

II. Approach

As part of the Course 5330 Computer Architecture, the Professor Dr. Richard Goodrum decided to test the skills of the students taking the course. In this Assignment conversions between Binary, Hexadecimal and Decimal needed to be coded in Mips assembly language. An open source simulator/IDE for MIPS assembly language called MARS (version 4.5) was used to develop the assignment. Mars needed a Java 8+ JRE to be installed. I have used Java 11.0.12.

The problem statement required the filename to not be hardcoded. This was achieved by asking the user to input the filename from the user. A maximum of 128 character length was allowed for the filename. Also the file needs to be placed at the location of Mars application for it to be accessible from Mars.

The Problem statement says that the input type may use white space for character grouping. We define whitespace as space, tab, vertical tab, new line, form feed and Carriage return. Even though the usage of whitespace is touted to be for character grouping, technically the input value could become very large. To avoid allocating space in memory to store input value as it is, the input value is printed onto console as soon as it is read. Provisions have been made to print on to console in proper format as stated in the problem statement

III. Limitations

The file name is limited to a length of 128 characters. The Binary value is taken as 32 bits in length. This means even if binary value of less than 32 bits is provided in input, sign extension is done and the input binary value is converted to 32 bits. Binary input cannot exceed 32 characters. Hexadecimal input cannot exceed 8 characters. Decimal input should lie in the range of 2147483647 to -2147483648

[Note: If facing difficulties opening the file, you may use sudo to run the Mars application as root user in linux or Mac systems. In windows, the Mars may be opened with Administrative privileges.]

IV. Solution

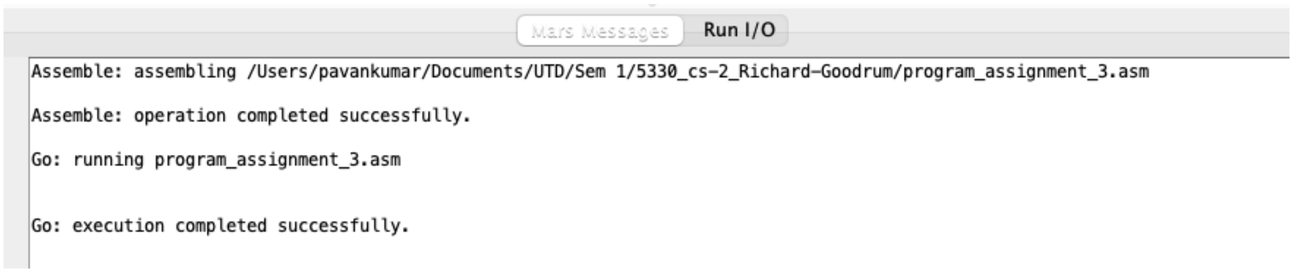
The program starts with a Descriptive prompt for the user's edification on how to interact with the program. We obtain a file name from the user. Clean up the name of any unwanted newline characters and open the file. We read 1 character at a time from the file.

As we know the data to be in certain format, we can start discerning the data while being read and print the data onto console conveniently avoiding the need to store the input value as it is.
Our main goal is to convert the input value given in one base to another base. In order to achieve this, we convert the input to binary first and then convert the binary to necessary base.

The Binary and Hexadecimal representations are following Two's complement so if the input value has a negative sign, the binary value obtained in the middle step must be negated so as to have proper sign field. In order to print the Two's complement representation of a negative number in binary or hexadecimal, we take the modulus of the input value, subtract it by 1. Then we divide this value by the base of output required to get the digits from least significant to most significant as remainder. This value is subtracted from the base -1 to obtain the complement of the value which when printed will yield the Two's compliment initially intended

V. Program Screenshots

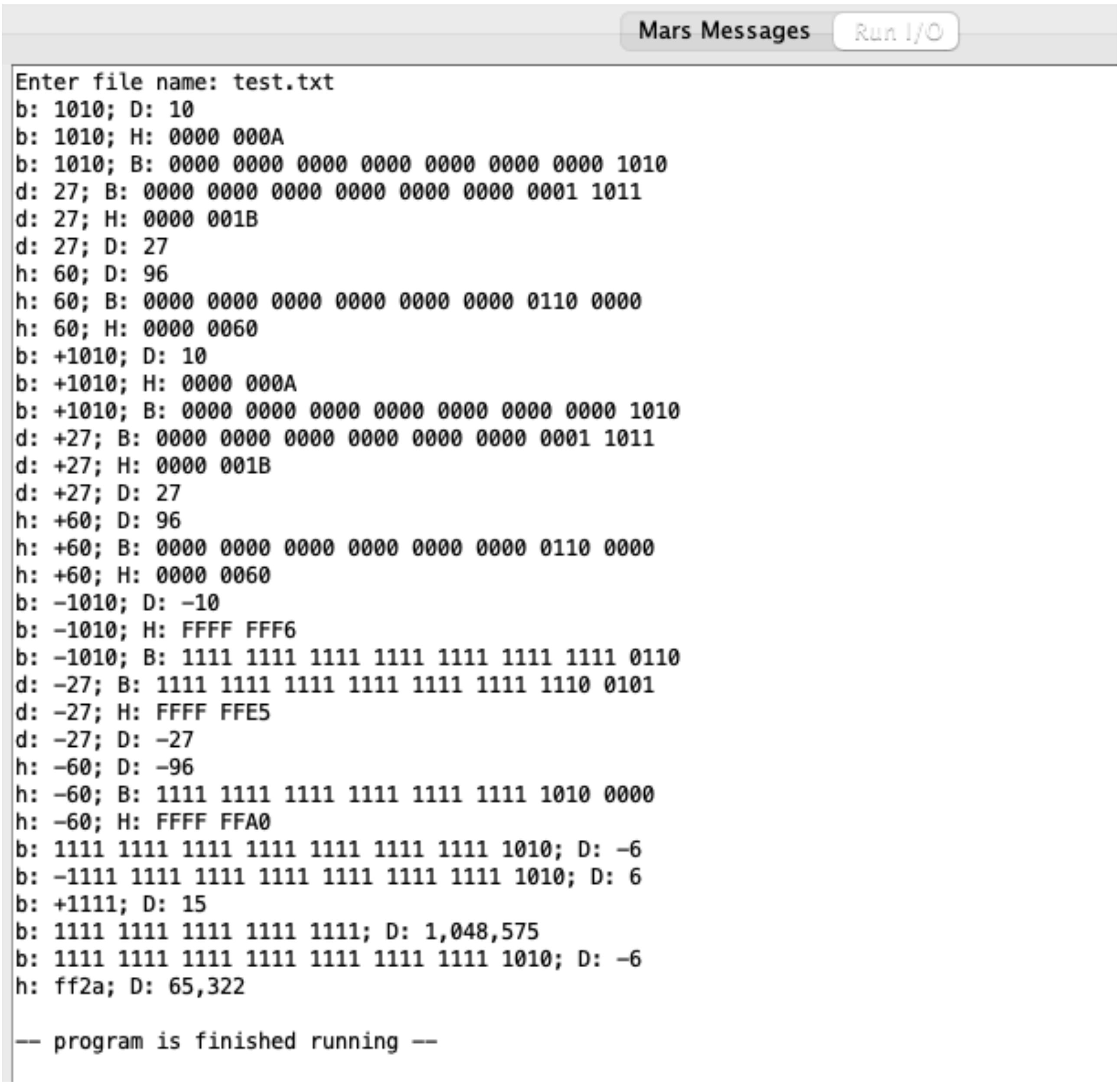
Figure 1: Assembler output of the program is provided in this image. As seen it has successfully compiled else we can see errors at this position. At this point the program is ready to run. In the next image, we'll discuss the output of the program.



```
Assemble: assembling /Users/pavankumar/Documents/UTD/Sem 1/5330_cs-2_Richard-Goodrum/program_assignment_3.asm

Assemble: operation completed successfully.

Go: running program_assignment_3.asm


Go: execution completed successfully.
```

Figure 1: Mars Assembler Output

Figure 2: This is the console output as obtained from running the program in the MARS IDE. It shows the initial prompt to the user so that proper filename can be provided. In this trial run, 33 different combination of inputs are used. They include all 27 permutations that have to be compulsorily provided as per problem statement. The test file used "test.txt" will also be submitted.



```
Enter file name: test.txt
b: 1010; D: 10
b: 1010; H: 0000 000A
b: 1010; B: 0000 0000 0000 0000 0000 0000 0000 1010
d: 27; B: 0000 0000 0000 0000 0000 0000 0001 1011
d: 27; H: 0000 001B
d: 27; D: 27
h: 60; D: 96
h: 60; B: 0000 0000 0000 0000 0000 0000 0110 0000
h: 60; H: 0000 0060
b: +1010; D: 10
b: +1010; H: 0000 000A
b: +1010; B: 0000 0000 0000 0000 0000 0000 0000 1010
d: +27; B: 0000 0000 0000 0000 0000 0000 0001 1011
d: +27; H: 0000 001B
d: +27; D: 27
h: +60; D: 96
h: +60; B: 0000 0000 0000 0000 0000 0000 0110 0000
h: +60; H: 0000 0060
b: -1010; D: -10
b: -1010; H: FFFF FFF6
b: -1010; B: 1111 1111 1111 1111 1111 1111 1111 0110
d: -27; B: 1111 1111 1111 1111 1111 1111 1110 0101
d: -27; H: FFFF FFE5
d: -27; D: -27
h: -60; D: -96
h: -60; B: 1111 1111 1111 1111 1111 1111 1010 0000
h: -60; H: FFFF FFA0
b: 1111 1111 1111 1111 1111 1111 1111 1010; D: -6
b: -1111 1111 1111 1111 1111 1111 1111 1010; D: 6
b: +1111; D: 15
b: 1111 1111 1111 1111 1111; D: 1,048,575
b: 1111 1111 1111 1111 1111 1111 1111 1010; D: -6
h: ff2a; D: 65,322

-- program is finished running --
```

Figure 2: Program execution and output