


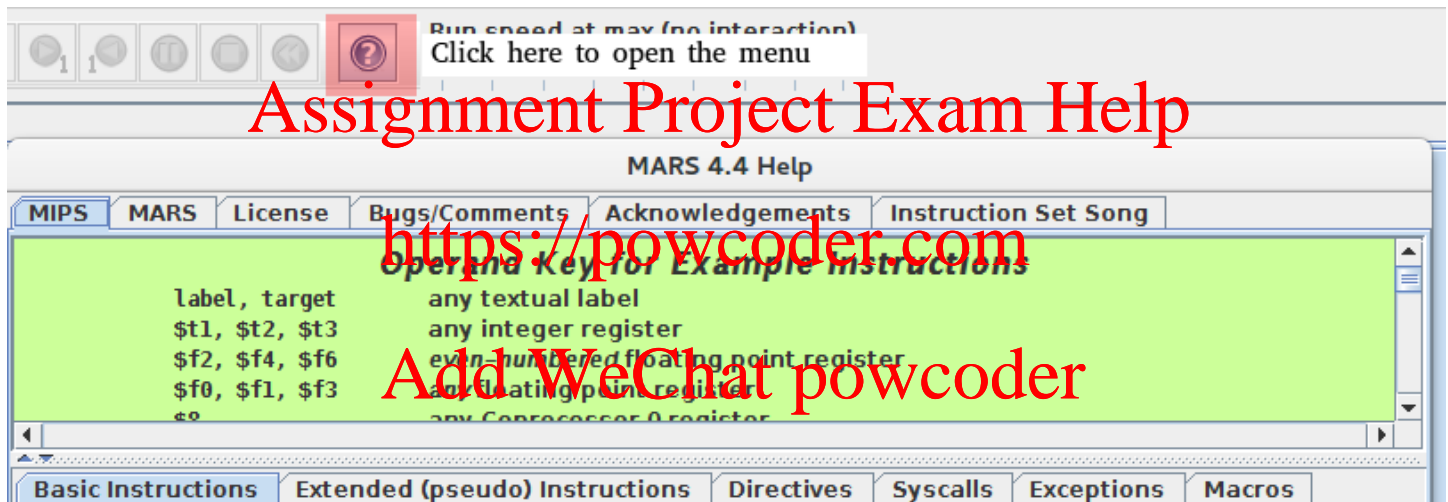
## Part I: Validate the First Command-line Argument and the Number of Command-line Arguments

For this assignment you will be implementing several operations that perform computations and do data manipulation.

In `hw1.asm`, begin writing your program immediately after the label called `start` coding here.

You may declare more items in the `.data` section after the provided code. Any code that has already been specified must appear exactly as defined in the provided file. Do not remove or rename these labels, as doing so will negatively impact grading.

The number of arguments is stored in memory at the label `num_args` by code already provided for you. You will need to use various system calls to print integers and strings that your code generates. For example, to print a string in MIPS, you need to use system call 4. You can find a listing of all the official MARS system calls on the [MARS website](https://mars.usf.edu/mars/). You can also find the documentation for all instructions and supported system calls within MARS itself. Click the  in the tool bar to open it:



The following is a list of the operations that your program will execute. Each operation is identified by a single character. The character is given by the first command-line argument (whose address is `addr_arg0`). The parameter(s) to each argument are given as the remaining command-line arguments (located at addresses `addr_arg1`, `addr_arg2`, etc.). Not all operations expect the same number of parameters. In this first part of the assignment your program must make sure that each operation is valid and has been given the correct number of parameters. Perform the validations in the following order:

1. The first command-line argument must be a string of length one that consists of one of the following characters: `F`, `C` or `2`. If the argument is a letter, it must be given in uppercase. If the argument is not one of these strings, print the string found at label `invalid operation error` and exit the program (via system call 10).
2. If the first command-line argument is `F` or `2`, then there must be exactly one other argument. If the total number of command-line arguments is not two, print the string found at label `invalid args error` and exit the program (via system call 10).
3. If the first command-line argument is `C`, then there must be exactly three other arguments. If the total number of command-line arguments is not four, print the string found at label `invalid args error` and exit the program (via system call 10).

Important: You must use the provided `invalid operation error` and `invalid args error`

strings when printing error messages. There are also five other strings marked with the comment # `Output strings`, which you must use for generating the output of your program. Do not create your own labels for printing output to the screen. If your output is marked as incorrect by the grading scripts because of typos, then it is your fault for not using the provided strings, and you will lose all credit for those test cases. See page 1 for more details about how your work will be graded.

### Examples:

<i>Command-line Arguments</i>	<i>Expected Output</i>
DB abc	INVALID_OPERATION
f xyz	INVALID_OPERATION
1	INVALID_OPERATION
F abc def	INVALID- ARGS
2	INVALID- ARGS
C 145 6842	INVALID- ARGS
C 1 2 3 4	INVALID- ARGS

## Review of Character Strings in MIPS Assembly

In assembly, a string is a one-dimensional array of unsigned bytes. Therefore, to read each character of the string we typically need to use a loop. Suppose that `$s0` contains the base address of the string (that is, the address of the first character of the string). We could use the instruction `lbu $t0, 0($s0)` to copy the first character of the string into `$t0`. To get the next character of the string, we have two options: (i) add 1 to the contents of `$s0` and then execute `lbu $t0, 0($s0)` again, or (ii) leave the contents of `$s0` alone and execute `lbu`

`$t0, 1($s0)`. Since most of the command-line arguments for this program can be of arbitrary length, you will probably find that you must use the first approach in most cases. Also note that syntax like `lbu $t0, $t1($s0)` is not valid; an immediate value (a constant) must be given outside the parentheses.

There is no `length()` function or method in assembly to tell us how long a string is. Therefore, we need a loop which traverses a string until it reads the null character, ASCII 0 (`'\0'`). The null character denotes the end of the string in memory.

If your program determines that the first command-line argument is a valid command and that it has been given a correct number of additional arguments, your program continues by executing the appropriate operation as specified below. Note that you are permitted to add code to the `.data` section as necessary to implement these operations.

## Part II: Interpret a String of 0s and 1s as a Two's Complement Number

### First Argument: 2

**Second Argument:** a two's complement value (given as a string of 0 and 1 characters)

The 2 operation treats the second command-line argument as a string of 0 and 1 characters that represent a two's-complement number. The leftmost character is the most significant bit of the integer. The argument might contain fewer than 32 characters. The operation converts the string into a two's-complement value and prints it in decimal to the screen, with a newline (`'\n'`) at the end.

## Input Validation:

The second command-line argument must consist only of at most 32 0 and 1 characters. If the argument contains invalid characters or more than 32 characters, print the string found at label `invalid args error` and exit the program (via system call 10).

You may assume that valid, converted values can fit within a single 32-bit register.

## Examples:

Command-line Arguments	Expected Output
2 11111111111111111111111111111111	INVALID ARGS
2 000000000000	0
2 0101	5
2 0101101	45
2 01111111111111111111111111111111	2147483647
2 1101110	-18
2 111111111	-1
2 10000000000000000000000000000000	-2147483648

## Part III: Interpret a String of Hexadecimal Digits as a 32-bit Floating-point Number

**First Argument:** F

**Second Argument:** a string of exactly 8 hexadecimal digits

The F operation treats the second command-line argument as a string of exactly 8 hexadecimal digit characters, 0–9, A–F (uppercase letters only), that represent a 32-bit floating-point number given in IEEE 754 notation. The first two characters together give the most significant byte of the 32-bit floating-point representation, whereas the last two characters provide the least significant byte. When the input is not a special value (zero, infinity or NaN), the operation extracts the sign bit, exponent and fraction from the input. The operation then prints the number in the following format:

$1.\text{fraction\_}2 \times 2^{\text{exponent}}$

The fraction is printed in binary using exactly 23 bits to the right of the radix point. The exponent is printed in decimal after subtracting the bias of 127. The program must not print any leading zeroes for the exponent.

If the number is negative, then print a minus sign in front of the value. Here's an example:

$-1.00110111101000010100000_2 \times 2^{15}$

Likewise, if the exponent is negative, print a minus sign in front of it:

$1.00110111101000010100000_2 \times 2^{-15}$

If the entire number and/or exponent is positive, do not print a plus sign.

After printing the number in this format, print a newline character (`\n`) on the output. A string at label `n1` has been provided in the `.data` section for you that contains the newline character.

When the input is a special value, print the corresponding string as indicated in the table below. Do not create your own strings in the `.data` section to print these outputs. Rather, use the strings found at the labels `zero str`, `neg infinity str`, `pos infinity str`, `NaN str` and `floating point str` at

the top of the provided .asm file.

To implement this operation you can (and must) use only integer registers. MIPS instructions that use the floating-point registers have been disabled in the CSE 220 version of MARS.

### Input Validation:

The second command-line argument must consist of exactly 8 hexadecimal digit characters (0–9, A–F), with uppercase letters only for digits A–F. If the argument contains invalid characters and/or is of the wrong length, print the string found at label `invalid args error` and exit the program (via system call 10).

### Examples:

Command-line Arguments	Expected Output
F 1234567	INVALID ARGS
F G1231234	INVALID ARGS
F 6318675309	INVALID ARGS
F 00000000 or F 80000000	Zero
F FF800000	-Inf
F 7F800000	+Inf
F 7F800001 thru F 7FFFFFFF	NaN
F FF800001 thru F FFFFFFFF	NaN
F 42864000	$1.00001100100000000000000000_2 \cdot 2^6$
F 14483B47	$1.10010000011101101000111_2 \cdot 2^{-87}$
F 94483B47	$-1.10010000011101101000111_2 \cdot 2^{-87}$
F C46AB32C	$-1.11010101011001100101100_2 \cdot 2^9$

## Part IV: Convert a Positive Integer from One Base to Another Base

**First Argument:** C

**Second Argument:** a string of decimal digits representing a number in a base

**Third Argument:** a string of decimal digits representing the base we are converting from

**Fourth Argument:** a string of decimal digits representing the base we are converting to

This operation takes a number given in one base (the “from” base), converts it to another base (the “to” base) and prints out the number in the other base, with a newline (‘\n’) at the end. For example, C 165 9 7 means that we want the computer to print the value of 165, as it would be represented in base 7 (i.e., 260).

The SBU version of MARS contains a special system call (#84) for converting a string of ASCII digit characters into a decimal integer. You are welcome to use that system call for converting the third and fourth arguments to the “C” operation into integers. For example, suppose \$s0 contained the starting address of a string of digit characters we wanted to convert into an integer. We would write this code to convert it into a number:

```
move $a0, $s0
li $v0, 84
```

syscall

The resulting integer would be available in `$v0`.

### Input Validation:

You may assume that only digits appear in the second, third and fourth command-line arguments. You may assume that the largest value we might need to convert is  $2^{31} - 1$ . You may assume that the third and fourth command-line arguments represent integers in the range  $[2, 10]$ . However, the program must check that the input number (second argument) is a valid representation of a number in the “from” base (third argument). If this is not the case, the operation prints the string found at label `invalid_args_error` and exits the program (via system call #10).

### Examples:

<i>Command-line Arguments</i>	<i>Expected Output</i>
C 8154 8 7	INVALID ARGS
C 000 5 10	0
C 23568 9 10	15776
C 0101 3 8	12
C 4123 5 5	4123
C 111102365 7 2	110110100110000110000

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder