CSE 220: Systems Fundamentals I

Homework #1

Fall 2016

Assignment Due: Sept. 23, 2016 by 11:59 pm via Sparky

⚠ **PLEASE READ THE WHOLE DOCUMENT BEFORE STARTING!**

Introduction

The goal of this homework is to become familiar with basic MIPS instructions, syscalls, loops, conditional logic and memory representations. In the first part of the assignment you will read in 2 ASCII characters strings and print their different interpretations. In the second part, you will generate random integer values and collect statistics on the numbers.

⚠ You **MUST** download the SB version of MARS posted on the PIAZZA website. **DO NOT USE** the MARS available on the official webpage. The SB version has a reduced instruction set and additional system calls you will need to complete the homework assignments.

⚠ DO **NOT** COPY/SHARE CODE! We will check your assignments against this semester and previous semesters!

Part 1: Command-line arguments

In the first part of the assignment you will initialize your program and identify the different command line arguments.

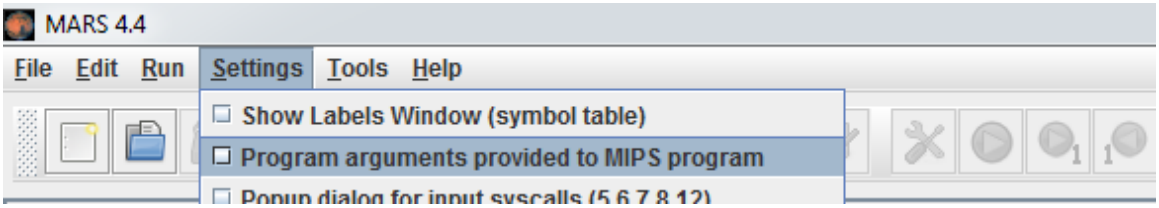
- Create a new MIPS program file.
- At the top of your program in comments put your name and id.

```
# Homework #1
# name: MY_FIRSTNAME_LASTNAME
# sbuid: MY_SBU_ID
...
```

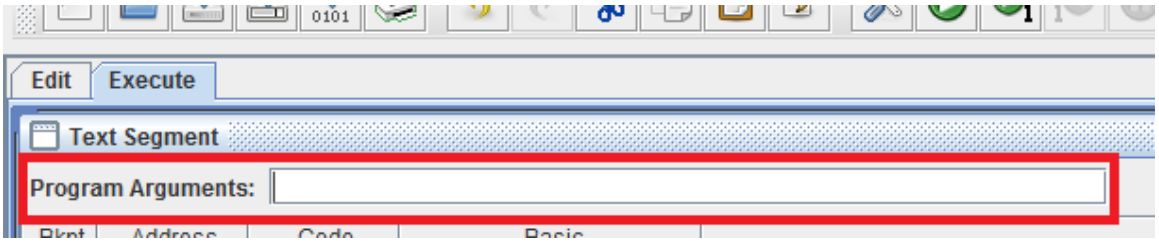
Configuring Mars for command line arguments

Your program is going to accept command line arguments which will provide input to the program and tell your program what operation to perform. To do this, we first need to tell Mars that we want to accept

command line arguments. To do this you need to open Mars, navigate to the **Settings** menu, and select `Program arguments provided to the MIPS program`.



After assembling your program, in the **Execute** tab you should see a text box where you can type in your command line arguments before running the program.



Each command line argument should be separated by a space.

❗ You can expect that command line arguments will always be given in the correct order for this assignment.

To use the arguments in your program, you will need to use the `$a0` and `$a1` registers. The `$a0` register will contain the number of arguments passed to your program. The `$a1` register contains the starting address of an array of strings. Each element in the array is an item that you specified on the command line.

In this assignment, we will be using two or three arguments. We will give you boilerplate code for obtaining these arguments further down in the assignment.

Setting up the .data section

Add the following directives to the `.data` section to define memory space for the assignment:

```
.data
.align 2
numargs: .word 0
arg1: .word 0
arg2: .word 0
arg3: .word 0
Err_string: .asciiz "ARGUMENT ERROR"

# Helper macro for grabbing command line arguments
.macro load_args
```

```
    sw $a0, numargs
    lw $t0, 0($a1)
    sw $t0, arg1
    lw $t0, 4($a1)
    sw $t0, arg2
    lw $t0, 8($a1)
    sw $t0, arg3
.end_macro
```

ℹ `load_args` is an assembler macro. Macros are different from functions. We use it to simplify access to the command line arguments for this first assignment.

ℹ You can declare more items in your `.data` section as you wish, but you must at minimum have the ones defined exactly as they appear in the above code section.

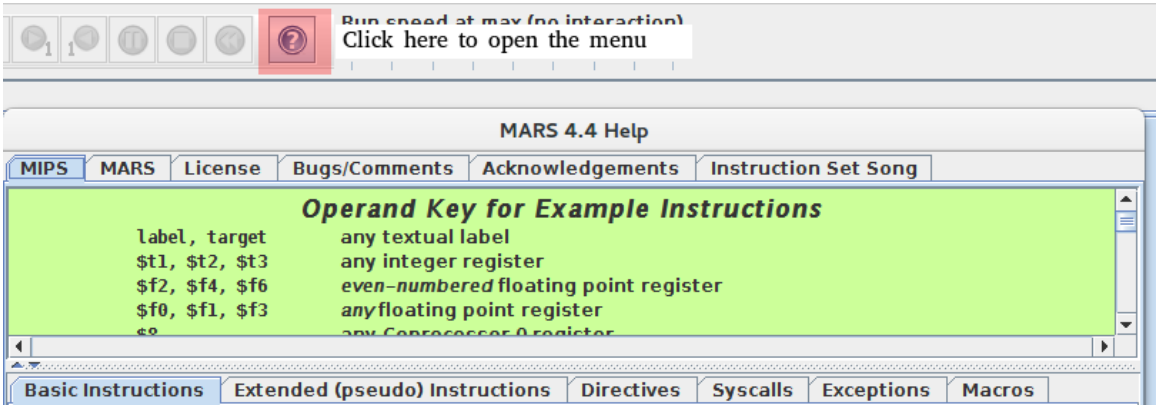
Writing the program

In your `.text` section, you will create a label `main:`. This is the main entry to your program. Next we will extract the command line arguments. You should do this before anything else to avoid losing the addresses to them. Also, never call this macro again, as you will overwrite the command line arguments passed to your program.

```
.text
.globl main
main:
    load_args()       # Only do this once
```

After the `load_args` macro is called, add code to check the number of command line arguments provided. If there are < 2 arguments or > 3 arguments, the program should print out the string "ARGUMENT ERROR" and exit the program. The number of arguments is stored in memory at the label `numargs` by the macro, but the value is also STILL in `$a0`, as the macro code does not modify the contents of `$a0`. Remember, values remain in registers until a new value is stored into the register, thereby overwriting it.

ℹ To print a string in MIPS, you need to use system call 4. You can find a listing of all the official Mars systems calls here. You can also find the documentation for all instructions and supported system calls within Mars itself. Click the ℹ in the tool bar to open it.



If the number of arguments is valid, check the first argument provided on the command line. The first argument is valid if it has both of the following properties:

- is only 1 character in length, and
- is upper or lower case ASCII characters for 'A' or 'R'

ℹ To use the values placed in `arg1`, `arg2`, and `arg3`, you will need to use load instruction(s) to get the value(s) stored in memory. The labels **arg1**, **arg2**, and **arg3** each store the starting address of a null-terminated sequence of ASCII characters stored in memory.

Some other questions to ask yourself are:

❓ Consider the difference between the instructions `lw` and `lb`.

❓ How are strings stored in memory? How would you know if the argument string has a length of only 1 character? How do you check this?

❓ What is the relationship between uppercase and lowercase ASCII characters in their binary representation?

If the first argument is not valid, print "ARGUMENT ERROR" and exit the program. If the first argument is valid, write code to check between the possible cases 'A' or 'R' (upper or lower case). If the argument is 'A', then you will perform the operation defined in Part 2 of the assignment. If the argument is 'R', then Part 3 of the assignment will be performed. For now, use the string syscall to print the string "Part 2: " or "Part 3: " along with the value of the first argument to the output. You must create these strings in your `.data` section of your program.

Sample Outputs:
Arguments: `abcd abcd abcd`
Output: `ARGUMENT ERROR`

Arguments: `a cse220`
Output: `Part 2:  a`

Arguments: `A abcd aBcD`
Output: `Part 2:  A`

Arguments: `1 1234 5678`
Output: `ARGUMENT ERROR`

Arguments: `z abracadabra opensesame`
Output: `ARGUMENT ERROR`

Arguments: `R 20`
Output: `Part 3:  R`

Arguments: `r rrrrr`
Output: `Part 3:  r`

Arguments: `C3-P0 R2D2`
Output: `ARGUMENT ERROR`

Part 2: ASCII Characters

In this part of the assignment, you will analyze the first 4 ASCII characters of the second and third command line arguments and print out information about these ASCII strings and then quit the program.

Comment out the lines of code which print "Part 2: " and the value of the first argument. We will be replacing this functionality in this section.

First, since this part of the assignment requires 3 arguments, you must first check that 3 arguments were provided. If only 2 arguments were provided, print "ARGUMENT ERROR" and exit the program.

Output Sample:
Arguments: `a cse220`
Output: `ARGUMENT ERROR`

Next, load the first 4 ASCII characters of the second command line argument into a register. Remember, these ASCII characters are encoded binary which are interpreted when printing. Therefore, the 1's and 0's of their representation can be interpreted in different ways. To illustrate this, you will print the different representations of the binary in different number formats.

ℹ Assume that the strings for `arg2` and `arg3` will always be 4+ ASCII characters in length. The assignment WILL ONLY be tested for these cases.

⚠ Memory alignment of strings will make loading the 4 ASCII characters into a register more difficult than just using the `lw` instruction. How can you load these characters from memory and place them into a register just as they are stored in memory?

⚠ Endianess matters! MIPS is Little Endian, therefore the first character of the string should be in the Least Significant Byte (LSB) of the register!

Print the value in your register in the following format:

```
ARG2: BINARY_VALUE 0xHEX_VALUE TWOS_COMPLEMENT ONES_COMPLEMENT SIGNED_MAGNITUDE
```

Print the label string "ARG2: ", the binary representation of the characters, the hexadecimal representation of characters, the 2's complement integer value of the number, the one's complement value of the number, and the signed magnitude value of the number.

Begin by defining the label string for `ARG2` in your .data section.

You will need to:

1. Print the label string (syscall 4)
2. Print the binary representation of the characters, `BINARY_VALUE` (syscall 35)
3. Print the hexadecimal value of the characters, `0xHEX_VALUE` (syscall 34)
4. Print the value of the binary as if it were a 2's complement number `TWOS_COMPLEMENT` (syscall 1)
5. Print the value of the binary as if it were a 1's complement number `ONES_COMPLEMENT` (syscall 100)
6. Print the value of the binary as if it were a Signed Magnitude number `SIGNED_MAGNITUDE` (syscall 101)

To print sign magnitude and 1's complement values using MARS, we created additional syscalls. These syscalls are not standard and will not appear in the official MARS documentation.

description	syscall	arguments	result
print 1's comp	100	$a0 = value to print	
print SM	101	$a0 = value to print	

Sample Outputs: Arguments: `A abcd aBcD`
Output: `ARG2:  01100100011000110110001001100001 0x64636261 1684234849 1684234849 1684234849`

Arguments: `a 1234 7890`
Output: `ARG2:  00110100001100110011001000110001 0x34333231 875770417 875770417 875770417`

Repeat this process for the third argument.

ℹ It is acceptable to "copy and paste" and modify the registers to do this, as we have not covered functions yet.

⚠ Make sure to keep the 4 characters of `arg2` in a register. Do NOT overwrite them as we will use them again.

Sample Outputs:
Arguments: `A abcd aBcD`
Output:
```
ARG2:  01100100011000110110001001100001 0x64636261 1684234849 1684234849 1684234849
ARG3:  01000100011000110100001001100001 0x44634261 1147355745 1147355745 1147355745
```

Arguments: `a 1234 7890`
Output:

```
ARG2:  00110100001100110011001000110001 0x34333231 875770417 875770417 875770417
ARG3:  00110000001110010011100000110111 0x30393837 809056311 809056311 809056311
```

The last thing you will do is compare the hamming distance between the binary representations of these arguments. The Hamming distance between two binary representations of equal length is the number of bits which differ. In another way, it measures the minimum number of bit flips required to change one representation into the other. For example, for 7-bit binary numbers 1011101 and 1001001 the hamming distance is 2.

Calculate the hamming distance between the binary representations of `arg2` and `arg3` and print it to the output as shown in the example. After printing, the program should exit (syscall 10).

ℹ Hint: Which logical operation will tell you if two bits have different values?

Sample Output:
Arguments: `A abcd aBcD`
Output:

```
ARG2:  01100100011000110110001001100001 0x64636261 1684234849 1684234849 1684234849
ARG3:  01000100011000110100001001100001 0x44634261 1147355745 1147355745 1147355745
Hamming Distance:  2
```

Arguments: `a 1234 7890`
Output:

```
ARG2:  00110100001100110011001000110001 0x34333231 875770417 875770417 875770417
ARG3:  00110000001110010011100000110111 0x30393837 809056311 809056311 809056311
Hamming Distance:  7
```

❓ Why are the 2's complement, 1's complement, and Signed Magnitude values always the same? Will they ever be different?

Part 3: Random Numbers

In this part of the assignment, you will use the random number generator in MARS to draw numbers based on the value of `arg2`, the seed. For the numbers drawn you will keep track of statistics about these values. You will stop drawing numbers when the value is a power of 2 and the value is less than 64.

Comment out the lines of code which print "Part 3: " and the value of the first argument. We will be replacing this functionality in this section.

First, since this part of the assignment requires only 2 arguments, you must first check that 2 arguments were provided. If 3 arguments were provided, print "ARGUMENT ERROR" and exit the program.

Next, you will need to convert the string in `arg2` to an integer encoded in two's complement. Pseudo code of the algorithm for converting the string to a number is presented below (positive numbers only). If an invalid character in the string is found, conversion should stop and the part of the number converted

should be used. If the first character of `arg2` is an invalid character, the algorithm will result in the `sum` of 0.

```
var sum = 0
# Convert every character until the end of the string is reached
# or an invalid character is reached
for each character in the input:
    if character >= '0' and character <= '9':
        sum = (sum * 10) + (char - '0')
    else:
        break
```

Sample Conversion:
arg2: `"123"`
sum: 123

arg2: `"1a3"`
sum: 1

arg2: `"a"`
sum: 0

arg2: `"23-4"`
sum: 23

arg2: `"-10"`
sum: 0

Once `arg2` is converted into a positive integer value (stored in register as 2's complement), set the seed of the random number generator using syscall 40 (Set `$a0` to 0).

Details of the syscalls for drawing random numbers can be found in greater detail in the MARS documentation. We have included an abbreviated version for reference.

description	syscall	arguments	result
set seed	40	`$a0` = i.d. of pseudorandom number generator (any int). `$a1` = seed for corresponding pseudorandom number generator.	None
random int range	42	`$a0` = i.d. of pseudorandom number generator (any int). `$a1` = upper bound of range of returned values.	`$a0` next value

For the random numbers which you will be drawing, you will need to collect the following statistics:

- Total number of values drawn

- Total number of odd values drawn
- Total number of values which are a power of 2
- Total number of values which were a multiple of 2
- Total number of values which were a multiple of 4
- Total number of values which were a multiple of 8

Create a while loop to continue drawing random values in the range of 1-1024 (ie. 1+ rand()%1023), using syscall 41, until the value is a power of 2 AND is less than 64. Given the value drawn, update all statistics. When the while loop terminates, print out all the statistics and the last random number drawn. After printing, the program should exit (sycall 10).

Sample Output:

Arguments: `r 0`
Output:

```
Last value drawn:  2
Total values:  28
# of Even:  12
# of Odd:  16
Power of 2:  1
Multiple of 2:  12
Multiple of 4:  9
Multiple of 8:  3
```

Arguments: `r 10`
Output:

```
Last value drawn:  1
Total values:  564
# of Even:  286
# of Odd:  278
Power of 2:  3
Multiple of 2:  286
Multiple of 4:  140
Multiple of 8:  62
```

Arguments: `R 220`
Output:

```
Last value drawn:  16
Total values:  342
# of Even:  155
```

```
# of Odd:  187
Power of 2:  4
Multiple of 2:  155
Multiple of 4:  76
Multiple of 8:  39
```

Hand-in instructions

If you completed all parts of the assignment your program should support the `A` and `R` flags only. Please do not add any miscellaneous printouts, as this will probably make the grading script give you a zero. Please print out the text exactly as it is displayed in the examples.

See Sparky Submission Instructions on piazza for hand-in instructions. There is no tolerance for homework submission via email. They must be submitted through Sparky. Please do not wait until the last minute to submit your homework. If you are struggling, stop by office hours.

When writing your program try to comment as much as possible. Try to stay consistent with your formatting. It is much easier for your TA and the professor to help you if we can figure out what your code does quickly.