CSE 220: Systems Fundamentals I

Homework #1

Fall 2017

Assignment Due: September 24, 2017 by 11:59 pm via Sparky

⚠ **READ THE WHOLE DOCUMENT BEFORE STARTING!**
⚠ DO **NOT** COPY/SHARE CODE! We will check your assignments against this semester and previous semesters!

ℹ Download the Stony Brook version of MARS posted on the PIAZZA website. **DO NOT USE** the MARS available on the official webpage. The Stony Brook version has a reduced instruction set, added tools, and additional system calls you will need to complete the homework assignments.

⚠ All assignments MUST be implemented in MIPS Assembly language.

⚠ All test cases MUST execute in 10,000 instructions or less. Efficiency is an important aspect of programming.

Introduction

The goal of this homework is to become familiar with basic MIPS instructions, syscalls, basic loops, conditional logic and memory representations.

In this assignment, you will read, validate, and set the fields of an IPv4 packet header stored in memory.

Part 1: Command-line arguments

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

To begin writing your program:

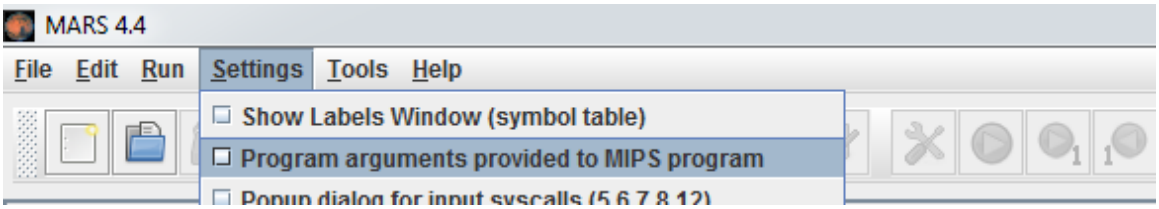
1. Open MARS and create a new MIPS program file.
2. At the top of your program add your name, Net ID and SBU ID# as comments. (Note that a single-line comment starts with a pound sign.).

```
# Homework #1
# Name: MY_FIRSTNAME MY_LASTNAME (e.g., John Smith)
# Net ID: MY NET ID (e.g., jsmith)
# SBU ID: MY_SBU_ID (e.g., 111999888)
```

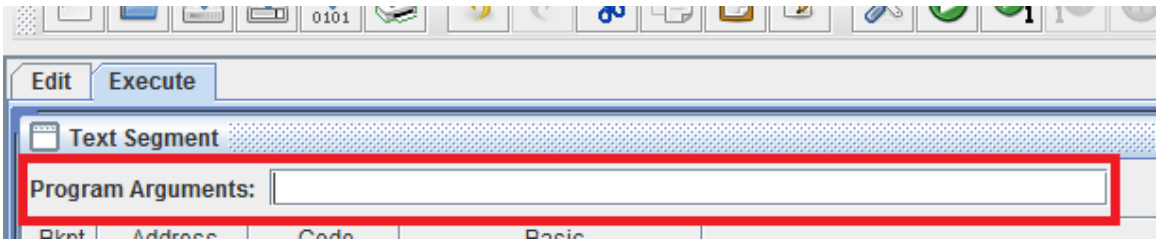
Configuring MARS for Command-line Arguments

Your program is going to accept command line arguments, which will be provided as input to the program. To tell MARS that we wish to accept command line arguments, we must go 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.

❗ Your program must ALWAYS be run with at least one command line argument! You can expect that command line arguments will always be given in the correct order for this assignment.

When your program is assembled and then run, the arguments to your program are placed in memory. Information about the arguments is then provided to your code using the argument registers, `$a0` and `$a1`. The `$a0` register contains the number of arguments passed to your program. The `$a1` register contains the starting address of an array of strings. Each element in the array is the starting address of the argument specified on the command line.

ℹ All arguments are saved in memory as ASCII character strings.

Your program will take the following command-line arguments:
`IPDest3 IPDest2 IPDest1 IPDest0 BytesSent Payload`

- `IPDest3-IPDest0`: Segments of the Destination IP address. Each segment value is in the inclusive range [0, 255]. Arguments map to the format: `IPDest3.IPDest2.IPDest1.IPDest0`
- `BytesSent`: Integer value in range [-1, 2^{13}-1]
- `Payload`: Any string of ASCII characters, any length. String WILL NOT contain spaces.

We have provided you boilerplate code for extracting each of the arguments from the array and storing their values in accessible labels in your `.data` section

Setting up the .data section

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

```
.data

# include the file with the test case information
.include "Header1.asm"  #change this line to test with other inputs

.align 2

    numargs: .word 0
    AddressOfIPDest3: .word 0
    AddressOfIPDest2: .word 0
    AddressOfIPDest1: .word 0
    AddressOfIPDest0: .word 0
    AddressOfBytesSent: .word 0
    AddressOfPayload: .word 0

    Err_string: .asciiz "ERROR\n"

    newline: .asciiz "\n"

# Helper macro for accessing command line arguments via Label
.macro load_args
    sw $a0, numargs
    lw $t0, 0($a1)
    sw $t0, AddressOfIPDest3
    lw $t0, 4($a1)
    sw $t0, AddressOfIPDest2
```

```
    lw $t0, 8($a1)
    sw $t0, AddressOfIPDest1
    lw $t0, 12($a1)
    sw $t0, AddressOfIPDest0
    lw $t0, 16($a1)
    sw $t0, AddressOfBytesSent
    lw $t0, 20($a1)
    sw $t0, AddressOfPayload
.end_macro
```

`load_args` will store the total number of arguments provided to the program at an address in memory labeled `numargs`. In addition, the starting address of each argument string that is provided to your program can be accessed using their labels (eg. `AddressOfIPDest3`, `AddressOfPayload`, etc).

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. You may implement macros in your own programs. However, it is not a requirement. We caution their use, as they can introduce non-obvious coding bugs if not used carefully!

ℹ You can declare more items in your `.data` section as you wish. Any specified items MUST appear exactly as defined in the above code section. DO NOT REMOVE or RENAME these labels.

Writing the program

Create the `.text` section in your homework file and create a label `main:` and add the `.globl` directive as shown below. This is the main entry to your program. Next, we will extract the command line arguments using the `load_args` macro. This must be executed before any other code to avoid losing values in `$a0` and `$a1`.

ℹ The `load_args` macro will crash if zero arguments are provided when you run the program. You must test with at least 1 argument specified.

In addition, NEVER call this macro again. Doing so could overwrite the command line arguments passed to your program.

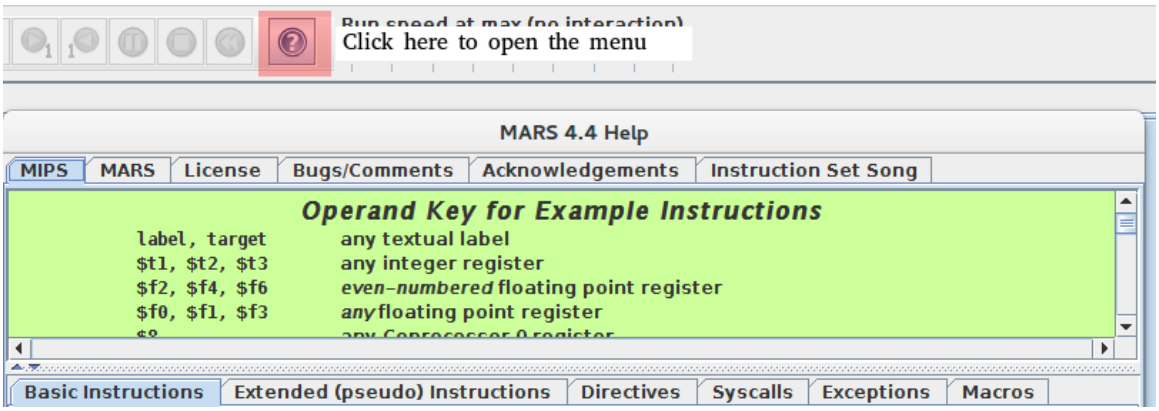
```
.text
.globl main
main:
    load_args()       # Only do this once
```

After the `load_args` macro is called, you will begin writing your program.

First add code to check the number of command line arguments provided. If the value string stored in memory at label `numargs` is not 6, the program should print out `Err_string` and exit the program (syscall 10). Note that the error string has already been defined in your `.data` section and is stored in memory at the address labeled `Err_string`.

ℹ 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 ? in the tool bar to open it.



Print out `Err_string` and exit the program (syscall 10) if any of the arguments is invalid.

If the number of arguments is valid, next check the strings for each of the `IPDest` arguments.

We have added a special `atoi` syscall to MARS to convert a string of digit characters to a 32-bit integer. You should use this syscall to convert each of the arguments, except `Payload`, to integer values.

Service	Code	Args	Result
atoi	84	$a0 = Starting address of the string to convert	Converts an ASCII string to an integer and places the result in $v0. Success: $v1 = 0, Fail: $v1 = -1

The syscall returns failure if the provided string contains any ASCII character not in the following set:
{ `'-', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'` }.

The arguments are valid if `atoi` returns success and the value returned is in the range [0,255].

ℹ NOTE: Real computers do not have this type of syscall. This was created for your ease in this assignment.

ℹ Remember, the addresses of the arguments, which were null-terminated sequence of ASCII characters (strings), were stored at unique labels by the load_args() macro. You will need to use load instruction(s) to obtain the value(s) of these strings stored at the addresses.

Finally, check the `BytesSent` argument is valid. It is valid if it meets the following 3 criteria:

- `atoi` returns success,
- the value returned by `atoi` is in the decimal value range [-1, 2^{13}-1], and
- the value returned by `atoi` is a multiple of 8, except when it is -1.

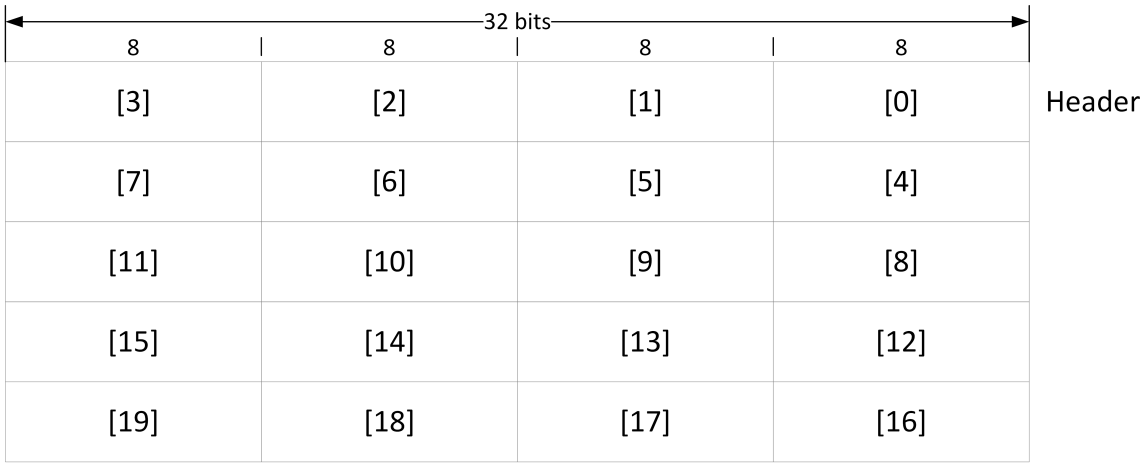
Print out `Err_string` and exit the program (syscall 10) if any of the arguments is invalid.

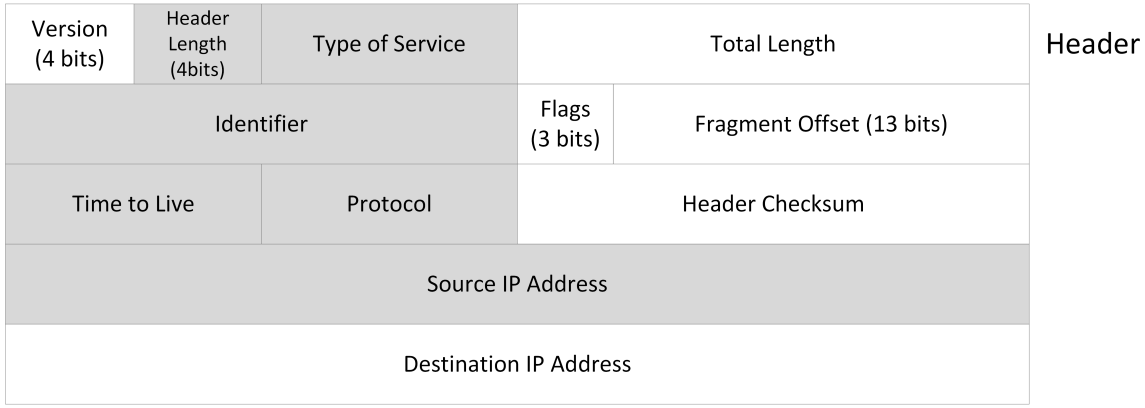
? Consider the difference between the instructions `la`, `lw` and `lb`.

At this point, your program should correctly handle and validate all of the command-line arguments passed to the program.

Part 2: Accessing the IPv4 Packet header fields

We have provided you with a set of packet headers in different files with names like `Header1.asm`, `Header2.asm`, etc. In each file is a label `Header:`, which is the address of the first byte of the packet stored in memory.





All Internet packets are communicated in Big Endian format. MARS stores data in memory in Little Endian format. We will learn more about these formats in lecture. To simplify the assignment, we will assume the IPv4 packets are stored in memory as viewed in the above figure. The numbers in brackets are the byte numbers. You can think of these as offsets from the address stored at the `Header` label.

ℹ Note the shaded fields of the packet MUST not be modified in memory.

Complete the following steps in the specified order:

1. Check Packet Version

   - Load byte 3 of `Header` from memory. Use bitwise and/or shifting operations to obtain the Version number. If the field is 4, print to console "IPv4\n" else print "Unsupported: IPv_\n" where the _ is the value of the field. Note that there are NO SPACES in the output.
   - If the version was not 4, set the version to 4. Use bitwise and/or shifting operations to set the 4 bits of the field. Make sure you do not modify the header length field.

2. Load the values of the following fields and print them to the console in the specified format. All values should be printed as POSITIVE decimal integers (use syscall 1), separated by commas, with NO SPACES in between. Do not print any `+` symbols. A single newline must be printed at the end of the line.

   `Type_of_Service,Identifier,Time_To_Live,Protocol\n`

3. Print the value of the `Source IP Address` field in the format. Print each decimal integer value with periods separating the segments.

   `IPSrc3.IPSrc2.IPSrc1.IPSrc0\n`

4. Set the value of the `Destination IP Address` field to the value specified by the IPDest arguments, `IPDest3.IPDest2.IPDest1.IPDest0`

   - Once the value is stored in memory. Use syscall 34 to print the Dest address in hexadecimal to the screen. For example, IP address 130.100.10.28 would print as

     `0x82640A1C\n`

5. Use a loop to calculate the number of bytes for the `Payload` argument (i.e., count the number of chars in memory until a NULL byte ('\0') is reached.).

   - Add the number of bytes for the `Payload` to the value in bytes of the `Header Length` field and store the resultant value in the `Total Length` field in units of bytes. The `Header Length` field is the number of 32-bit words (4-byte words) present in the header. The value of this is typically, but not limited to, 5 or 6.

6. `Flags` and `Fragment Offset` fields

   - Print both the `Flags` field and the `Fragment_Offset` field in binary using sycall 35. Note: The maximum possible decimal value for the flags field is 7, as the field is only 3 bits wide. Use a combination of load and bitwise instructions to determine the value of each field.

     `Flags,Fragment_Offset\n`

   - Set these fields based on the `BytesSent` argument, using the following specification.
     - If `BytesSent == 0`, the packet is the first packet. Set the bits of the flag field to `000` (decimal value 0) and set the `Fragment Offset` field to 0.
     - If `BytesSent == -1`, the packet is NOT fragmented. Set the bits of the flag field to `010` (decimal value 2) and set the `Fragment Offset` field to 0.

- If `BytesSent > 0`, set the bits of the flag field to `100` (decimal value 4) and set the fragment offset to `BytesSent`.

> ℹ Note: The flag field bits for this homework are ordered in Little Endian. In the real IPv4 packet header they are stored in Big Endian.

7. Save the `Payload` Argument into memory after the packet header. The IPv4 packet header can also contain one word of optional bytes after the `Destination IP Address` (Not shown in figure). The value of the `Header Length` field includes the optional bytes, if used.

   Use a loop to store each byte of the `Payload` argument into memory of the `Header` IPv4 packet starting at the address (`Header + Header Length*4`). Remember, the `Header Length` field is the number of 32-bit *words* (4-byte words) present in the header.

   > ℹ It is guaranteed there is enough space allocated in memory to store the full Payload argument after the `Header`.

8. Print the HEX values for the starting address of the header and the address of the last byte of the IPv4 packet (ie. the last character of the `Payload` argument once stored in memory).

   ```
   Starting_Addr_of_Packet,Ending_Addr_of_Packet\n
   ```

   The absolute difference between these 2 addresses should be equal to `Total Length-1`. You can use this calculation to check your work!

9. Calculate and update the `Checksum` field for the current modified `Header`. The size of the header is based on the Header Length field. Refer to IPv4 header checksum via Wikipedia for the 1's complement algorithm which is performed on half-words.

   > ℹ Remember: When calculating the checksum the bytes for this field of the header are skipped.

   > ℹ The Wikipedia article shows you a method to check that the checksum value you calculated is correct!

10. Terminate the program using Syscall 10.

Sample files are provided (with expected outputs in the comments of the files) on PIAZZA. These files must be placed in the same directory as your Mars executable! If they are in a different location, you need to change the `.include "Header1.asm"` to contain the full path of the file.

> ℹ We HIGHLY encourage you to create your own samples to test and verify your programs on different packet values.

The provided sample files MAY BE used in grading of the assignment. Additional tests WILL BE used for grading.

Hand-in instructions

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 provided examples.

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.

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 having trouble submitting, stop by office hours prior to the deadline for assistance.