University of Arizona, Department of Computer Science

CSc 340 Foundations of Computer Systems — Assignment 1

Christian Collberg

Due Thursday, February 8, at 11:59pm

1 Assignment

One reason why one might want to write in assembly code is to optimize common or expensive operations. For example, applications that perform much sorting and searching on large data-sets might benefit from having QuickSort and hash table-routines implemented directly in machine code.

In this assignment we will implement a very simple string manipulation package in MIPS assembly code. You only need to write two routines:

length(A) computes the length of a null-terminated character string **A**,

compare(A,B) compares two null-terminated character strings **A** and **B** and returns -1 if **A<B**, 0 if **A=B**, and 1 if **A>B**.

The file `/home/cs340/spring01/prog1/string.s` contains a skeleton program to get you started. The `length` and `compare` routines have already been declared for you; all you have to do is fill in the missing bodies of these routines.[1]

2 Testing

The skeleton file also contains several test routines that call `length` and `compare`. Here's the output when I run the program using `spim`'s "batch" mode:

```
> spim -file string.s
length("")=0
length("HELL")=4
length("HELLO")=5
length("HELLNO")=6
compare("","")=0
compare("HELLO","HELLO")=0
compare("","HELLO")=-1
compare("HELLO","")=1
compare("HELL","HELLO")=-1
compare("HELLO","HELL")=1
compare("HELLO","HELLNO")=1
compare("HELLNO","HELLO")=-1
```

[1] My solution requires a total of 20 assembly code statements for the bodies of the two routines. Can you do better?

3 Honors section

Those enrolled in the honors section should do their work on `wonka.cs.arizona.edu`. You are allowed to collaborate in converting the skeleton file to *wonka-normal-form*.

4 Extension

For extra credit add one or both of the following two routines to the string manipulation package:

lowercase(A) converts any uppercase characters in the null-terminated string **A** to lower case. Other characters are unaffected.

index(A,B) returns the first position of the string **A** in **B**, or -1 if **A** does not occur in **B**. Both strings are null-terminated.

For example,

1. ⌜`lowercase("George W #2")`⌝ should convert the string `"George W #2"` to `"george w #2"`,
2. ⌜`index("ack","acker")`⌝ should return 0,
3. ⌜`index("ack","slacker")`⌝ should return 2,
4. ⌜`index("ack","slact")`⌝ should return -1, and
5. ⌜`index("ack","backpack")`⌝ should return 1.

You should add your own `testLowercase` and `testIndex` routines to `strings.s`.

Each routine will earn you an extra 10% of the total assignment grade.

5 Turnin and the Makefile

A sample makefile for your use can be found in file `/home/cs340/spring01/prog1/Makefile`. SPIM will run `string.s` directly, but the Makefile makes it easy for you to turn in your assignment when you're done.

When you have completed the program, submit your `string.s` file by typing ⌜`make turnin`⌝. Or, to do it manually type ⌜`turnin cs340prog1 string.s`⌝. You may turn in your `string.s` file as many times as you want; turnin will always replace the previously turned-in version with the new version.

This is an individual assignment. Don't show your code to anyone, don't read anyone's code, don't discuss the details of the code with anyone. If you need help with the assignment see the TA or the instructor.

A The Skeleton File

Each routine in the string package starts with a *prologue* and ends with an *epilogue*. These consist of instructions that are necessary to begin and end a function. For this assignment you do not need to modify the prologues and epilogues. The `length` routine looks like this:

```
length:
        # BEGIN  prologue
        subu     $sp,$sp, 24                  # Allocate stack frame
        sw       $fp, 0($sp)                  # Save $fp in frame
        addu     $fp, $sp, 24                 # Set up $fp
        sw       $ra, -20($fp)                # Save $ra
        # END prologue

        ##########################################
        #####     Add your own code here!    #####
        #####      Put the result in $v0     #####
        ##########################################

        # BEGIN epilogue
        lw       $ra, -20($fp)                # Restore $ra
        lw       $fp, -24($fp)                # Restore $fp
        addu     $sp, $sp, 24                 # Pop stack frame
        jr       $ra                          # Return
        # END epilogue
```

All routines take their arguments in registers `$a0`, `$a1`, `$a2`, and `$a3`. Function results are returned in register `$v0`. We will discuss this in detail later; for now all you need to know is that the code that you write for the `length` function can assume that the address of the input string is given in register `$a0`, and that `length` should put its result in register `$v0`.

The skeleton for the `compare` function is similar:

```
compare:
        # BEGIN  prologue
        subu     $sp,$sp, 24                  # Allocate stack frame
        sw       $fp, 0($sp)                  # Save $fp in frame
        addu     $fp, $sp, 24                 # Set up $fp
        sw       $ra, -20($fp)                # Save $ra
        # END prologue

        ##########################################
        #####     Add your own code here!    #####
        ##### Put the result (-1,0,1) in $v0 #####
        ##########################################

        # BEGIN epilogue ...
```

Here, the addresses of the two input strings `A` and `B` are given in registers `$a0` and `$a1`, and, again, `compare` should put its result in register `$v0`.

The rest of the skeleton file consists of routines for testing. You don't need to touch these, unless you want to test your routines with some different data (a good idea). The following strings are used by the test routines:

```
        .data
null:        .asciiz ""
HELL:        .asciiz "HELL"
HELLO:       .asciiz "HELLO"
HELLNO:      .asciiz "HELLNO"

nl:          .asciiz "\n"
str1:        .asciiz "compare(\""
str2:        .asciiz "\",\""
str3:        .asciiz "\")="
str4:        .asciiz "length(\""
```

Two routines `testLength` and `testCompare` call the `length` and `compare` functions, respectively, and print out their results:

```
        .text
# A routine for testing the length function.
testLength:
        # BEGIN  prologue
        subu     $sp,$sp, 24                # Allocate stack frame
        sw       $fp, 0($sp)                # Save $fp in frame
        addu     $fp, $sp, 24               # Set up $fp
        sw       $ra, -20($fp)              # Save $ra
        # END prologue

        move     $s0,$a0                    # save $a0

        jal      length                     # compute the length

        ....

# A routine for testing the compare function.
testCompare:
        # BEGIN  prologue
        subu     $sp,$sp, 24                # Allocate stack frame
        sw       $fp, 0($sp)                # Save $fp in frame
        addu     $fp, $sp, 24               # Set up $fp
        sw       $ra, -20($fp)              # Save $ra
        # END prologue

        ....
```

The `main` routine, finally, calls `testLength` and `testCompare` with the different test strings as input:

```
# The main program. Execution starts here.
main:
        # BEGIN  prologue
        subu      $sp,$sp, 24              # Allocate stack frame
        sw        $fp, 0($sp)              # Save $fp in frame
        addu      $fp, $sp, 24             # Set up $fp
        sw        $ra, -20($fp)            # Save $ra
        # END prologue

        la        $a0, HELLO
        jal       testLength

        la        $a0, HELL
        la        $a1, HELLO
        jal       testCompare

        ....

        # BEGIN epilogue
        lw        $ra, -20($fp)             # Restore $ra
        lw        $fp, -24($fp)             # Restore $fp
        addu      $sp, $sp, 24              # Pop stack frame
        jr        $ra                       # Return
        # END epilogue
```