

Program 4: Printing Hex Values

Program due: Wednesday, April 16th, 8 p.m.

You are to complete a program by providing a **hex** function and a **printHex** function.

The **hex** function will print the hexadecimal equivalent of the 32-bit word passed as the first argument. The second argument to **hex** will be a **0** or **1** to indicate that **hex** should use lower- or upper-case letters, respectively.

The **printHex** function will print one hexadecimal character. It will be passed two parameters: a 4-bit binary number (bits 31 to 5 will be zeroes), and a **0** or **1** to indicate that the character should be printed as lower- or upper-case, respectively. **printHex** will print the hexadecimal digit. Values from **a** to **f** will be printed as indicated by the second parameter.

To make the output easier to read, **hex** will print a blank space after every pair of hexadecimal digits. The blank space will be printed by calling **printBlank**. We will provide the **printBlank** function. **printBlank** takes no arguments and will print one blank space.

Each test case will have **main** and **printBlank**. **main** will call **hex**. **printBlank** will be called by your **hex** procedure. Here is an example:

```
.data

mainNumElements:
    .word 5

mainValues:
    .word -257
    .word 16
    .word -53
    .word 0
    .word 0x777

.text
main:
    # Function prologue -- even main has one
    subu $sp, $sp, 24      # allocate stack space -- default of 24 here
    sw   $fp, 0($sp)      # save frame pointer of caller
    sw   $ra, 4($sp)      # save return address
    addiu $fp, $sp, 24    # setup frame pointer for main

    addi $s1, $zero, 0    # $s1 = i = 0
    la   $t0, mainNumElements
    lw   $s2, 0($t0)      # $s2 = mainNumElements
    la   $s3, mainValues  # $s3 = addr of mainValues[0]

mainLoopBegin:
    slt  $t0, $s1, $s2    # $t0 = (i < mainNumElements)
    beq  $t0, $zero, mainLoopEnd

    # call: hex(mainValues[i], upper-case)
```

```

sll  $t0, $s1, 2      # $t0 = 4 * i
add  $t0, $s3, $t0    # $t0 = addr of mainValues[i]
lw   $a0, 0($t0)      # $a0 = mainValues[i]
addi $a1, $zero, 1    # $a1 = upper-case
jal  hex
jal  printOriginal

# call: hex(mainValues[i], lower-case)
sll  $t0, $s1, 2      # $t0 = 4 * i
add  $t0, $s3, $t0    # $t0 = addr of mainValues[i]
lw   $a0, 0($t0)      # $a0 = mainValues[i]
addi $a1, $zero, 0    # $a1 = lower-case
jal  hex
jal  printOriginal

addi $s1, $s1, 1      # i++
j    mainLoopBegin

mainLoopEnd:

mainDone:
# Epilogue for main -- restore stack & frame pointers and return
lw   $ra, 4($sp)      # get return address from stack
lw   $fp, 0($sp)      # restore frame pointer of caller
addiu $sp, $sp, 24    # restore stack pointer of caller
jr   $ra              # return to caller

.data
printOriginalEqualStr:
    .asciiz "="
printOriginalNewline:
    .asciiz "\n"

.text
printOriginal:
# Function prologue -- printOriginal
subu $sp, $sp, 32     # allocate stack space -- default of 24 here
sw   $fp, 0($sp)      # save frame pointer of caller
sw   $ra, 4($sp)      # save return address
sw   $a0, 8($sp)      # save $a0, the number
sw   $s5, 12($sp)     # preserve $s5 so we can use it here
addiu $fp, $sp, 32    # setup frame pointer of printOriginal

# print original value
addi $s5, $a0, 0
la   $a0, printOriginalEqualStr
li   $v0, 4
syscall
addi $a0, $s5, 0
li   $v0, 1
syscall
la   $a0, printOriginalNewline
li   $v0, 4
syscall

printOriginalDone:

```

```

# Epilogue for printOriginal -- restore stack & frame pointers & return
lw   $fp, 0($sp)      # restore frame pointer of caller
lw   $ra, 4($sp)      # get return address from stack
lw   $a0, 8($sp)      # restore $a0, the number
lw   $s5, 12($sp)     # restore $st
addiu $sp, $sp, 32    # restore stack pointer of caller
jr   $ra              # return to caller

```

printBlank:

```

# Function prologue -- printBlank
subu  $sp, $sp, 24    # allocate stack space -- default of 24 here
sw    $fp, 0($sp)    # save frame pointer of caller
sw    $ra, 4($sp)    # save return address
sw    $a0, 8($sp)    # preserve $a0, since we need it to print
addiu $fp, $sp, 24   # setup frame pointer of printBlank

addi  $a0, $zero, 0x20 # 0x20 = ascii for blank space
li    $v0, 11
syscall

# Epilogue for printBlank -- restore stack & frame pointers & return
lw   $fp, 0($sp)    # restore frame pointer of caller
lw   $ra, 4($sp)    # get return address from stack
lw   $a0, 8($sp)    # restore $a0
addiu $sp, $sp, 24  # restore stack pointer of caller
jr   $ra            # return to caller

```

Your code goes below this line

This is the output from the above code:

```

FF FF FE FF = -257
ff ff fe ff = -257
00 00 00 10 = 16
00 00 00 10 = 16
FF FF FF CB = -53
ff ff ff cb = -53
00 00 00 00 = 0
00 00 00 00 = 0
00 00 07 77 = 1911
00 00 07 77 = 1911

```

Input:

The two values passed to **hex** will be correct values. The first argument will be a 32-bit value in **\$a0**. The second argument will be **0** or **1** in **\$a1**.

Your **hex** procedure will call your **printHex** procedure. Pass two arguments to **printHex**. The first argument will be a 4-bit value in **\$a0**. The second argument will be **0** or **1** in **\$a1**.

Output:

printHex will print one hexadecimal digit. When the digit is **a** to **f**, **printHex** will use lower- or upper-case as indicated by the second argument being **0** or **1**, respectively.

hex will print a blank space after every pair of hexadecimal digits.

Note that some test cases that we provide may call **printHex** directly, without calling **hex**. This makes it easier for us to test if **printHex** is following the calling conventions. See, for example, `test05.s`.

Label naming conventions:

Labels in MIPS programs are global. This presents a problem when you want to use a label that is already in use elsewhere in the program. To avoid conflicts in labels between your code and the test cases, we will adopt the following practice:

Labels will have the name of the procedure or function at the start of the label. For example, if I want to use the label **LoopBegin:** in **main**, then I will name the label **mainLoopBegin:**. If you also want to use the label **LoopBegin:** in your average procedure, you will name it **hexLoopBegin:**.

We will check that you are following these labeling conventions when we grade your program!

What will be provided:

Each test case will contain **main**, and **printBlank**. We may use other procedures in our test cases to simplify the design of the test case. For example, several of the test cases include a **printOriginal** function that is called from **main**.

Your program must have the comment:

```
# Your code goes below this line
```

Everything that you provide must go below this required comment line. This includes all comments that you put in, all **.text** code segments that you add, and any **.data** segments that you might need.

We will provide some sample files for you to copy. They will be available to copy on the Unix machines (lectura and the Fedora machines, fd01 to fd08). They will also be on the Windows machines. On the Windows systems, look in the shared Rotis drive: **Rotis->csc252->shared->prog4**. On the Unix machines, look in the directory: **~cs252/spring08/prog4**. In both cases, the files will be named: **test01.s**, **test02.s**, etc. Secure ftp can also be used to access the test cases on lectura from your home system. Please check with us if you have problems accessing these test cases.

The required comment line:

```
# Your code goes below this line
```

must be present in the **prog4.s** file that you turnin! The **prog4.s** that you turnin may contain one of the test cases above the required comment, or the required comment can be the very first line (no test case data above it). Either is acceptable.

Turnin:

Note: We are asking for submission of programs via D2L only.

- Name your program: **prog4.s**

- Using a web browser, go to: <http://d2l.arizona.edu/>
- Login using the “UA NetID Login” (upper-left corner of the web page).
- You should now be at “My Home” on D2L. At the center bottom of the screen, under the heading “My Academic Courses”, you should find “C SC252 SP08 001-002H Homer” listed under 2008 Spring. Click on this link.
- You will now be at the CSc 252 page for Spring 08.
- There is a row of links just underneath the Wildcat. In this row, you will find the link “Dropbox”. Click on this link.
- You will be at a page that shows three Dropbox Folders: **Program4**, **Program4-Late**, **Program4-Regrade**. Only one of these will be active. The **Program4** folder is for on-time submissions. The **Program4-Late** folder will be available for late submissions. The **Program4-Regrade** folder is used for requesting a re-grade of the program after the grades have been returned. Click on “Program4”. If you are doing a late turnin, click on “Program4-Late”.
- You should now be at the page: “Submit Files - Program4”. Click on the “Add a File” button. A pop-up window will appear. Click on the “Choose File” button. Use the file browser that will appear to select your **prog4.s** file. Click on the “Upload” button in the lower-right corner of the pop-up window.
- You are now back at the “Submit Files - Program4”. Click on “Upload” in the lower-right of this window. You should now be at the “File Upload Results” page, and should see the message **File Submission Successful**.
- You can repeat this process as often as necessary. Each submission will be time-stamped. When we grade programs, we will grade only the most recent submission.