1 Overview

In this project, you will be using loops to iterate over arrays and strings. You will practice how to build loops, modeling both for() loops and while() loops. You’ll also practice with reading integers and characters, and learn about a new syscall.

As with Project 1, we will provide a set of testcases. Each testcase will include a set of variables, which your program will read; in every testcase, we’ll have the same variables, and the variables will have the same types - but the values of the variables, and their order in the file, will change.

1.1 Required Filename

Name your file proj02.s

2 Allowable Instructions

When writing MIPS assembly, the only instructions that you are allowed to use (so far) are:

- add, addi, sub
- beq, bne, j
- slt, slti
- lw, lh, lb, sw, sh, sb
- la
- syscall

While MIPS has many other useful instructions (and the assembler recognizes many pseudo-instructions), do not use them! We want you to learn the fundamentals of how assembly language works - you can use fancy tricks after this class is over.
3 A New Syscall: print_char

In Program 1, you probably used only syscalls 1 (print_int) and 4 (print_str). In this program, you will also need to use syscall 11 (print_char).

For syscall 11, $a0 must be the character to print (not the address of the character!). That is, if you want to print a newline, you can use this handy snippet:

```assembly
addi $a0, $zero, 0xa # ASCII '\n'
addi $v0, $zero, 11 # print_int
syscall
```

In your program, this will be necessary in order to implement the string printing routines. However, it is also useful (though not required) for printing newline characters. In fact, in this Project, it is possible (though not required) to write your code without creating a .data section at all!

4 The Loops

As in Project 1, the testcase will provide some control variables, which will tell you what to do. There are two control variables; each is either zero, or nonzero. Treat these as boolean variables. Thus, there are four different things that you might do: your program will do exactly one of them.

Each of the four things is a loop. Two are loops over arrays (when integers is nonzero); two are loops over a string (when integers is zero). Each has a forward variant (when forward is nonzero) and a backward variant (when forward is zero).

This program will not print out any blank lines; however, after each item that you print out, place a newline. If the array (or string) has zero elements, then your output will be empty.

4.1 Variants

When looping over integers, read each integer and print it out, one integer per line (but see below about filtering).

When looping over strings, read each character from the string and print it out, one character per line (but see below about filtering). Do not print out the null character at the end. Note that, while you will never have any blank lines in the output, if you print a space or tab, it will look like a blank line to a human. However, it is not actually blank; it is a line with a single character on it.

4.2 Filtering

In the two forward loops, you will filter the values that you print - meaning that you will simply not print some of them. You will not filter in either backward loop - you will always print every character.
In the forward integer loop, filter out any element that is zero; that is, only print out nonzero elements.

In the forward string loop, filter out any element that is an uppercase letter (that is, between 'A' and 'Z', inclusive). Print out any other character that you find.

In some testcases, you will find that all of the values are filtered out (or, that the array or string has zero length). In that case, your program will print nothing at all.

4.3 Testcase Format

Each testcase will provide the following variables. Note their types! Also, remember that you can’t assume anything about the order of the variables in memory (other than that the array and the string will be contiguous).

.data

integers: .byte 0 # nonzero means ints, zero means strings
forward: .byte 1 # nonzero means forwards, zero means backwards
numInts: .word 3

# this is an array, the length is given by numInts above
ints:
   .word 10
   .word -20
   .word 10

# this will always be a null-terminated string
str: .asciiz "The quick brown fox jumps over the lazy dog."

NOTE: The symbols numInts, ints, str will always be defined, whether or not they are needed.

4.4 C Code

The next page has (most of) the C code for your program. Feel free to copy this code into your comments if you desire. I highly recommend that you convert this to assembly without worrying about optimizing or reorganizing it.
// INTEGER-FORWARD
if (integers != 0 && forward != 0)
{
    for (int i=0; i<numInts; i++)
        if (ints[i] != 0)
            printf("%d\n", ints[i]);
}

// INTEGER-BACKWARD
if (integers != 0 && forward == 0)
{
    for (int i=numInts-1; i>=0; i--)
        printf("%d\n", ints[i]);
}

// STRING-FORWARD
if (integers == 0 && forward != 0)
{
    for (char *ptr = str; *ptr != '\0'; ptr++)
    {
        if (*ptr >= 'A' && *ptr <= 'Z')
            continue;
        printf("%c\n", *ptr);
    }
}

// STRING-BACKWARD
if (integers == 0 && forward == 0)
{
    char *ptr = str;

    // advance the pointer to the null terminator
    while (*ptr != '\0')
        ptr++;

    // back up by one
    ptr--;

    // when we get here, ptr points to the character *BEFORE* the null
    // terminator. If the string is empty, then this is actually before
    // str!
    for (; ptr >= str; ptr--)
        printf("%c\n", *ptr);
}
5  A Note About Grading

Your code will be tested automatically. While we will look at each program manually (and will grade you on style), the majority of your grade will be determined by an automatic script. Since your code is being tested automatically, you must match the required output exactly! Any extra spaces, blank lines (or missing lines), misspelled words, etc. will cause your testcase to fail.

(We will provide a set of testcases which you can use with the graing script; your final grade will include those testcases - and also some others, that we won’t be sharing.)

Thus, be careful to test your code thoroughly, and to read the spec carefully. Test your code with every testcase that we provide! While it’s not a perfect guarantee, if your code passes all of the testcases we provide, there’s a good chance that it will also pass the ones that we have not given to you.

6  Testing

You can find a set of testcases for this project at
http://www.cs.arizona.edu/classes/cs252/fall16/projects/proj02/

Each testcase has two parts: the .s file that contains the code we provide, and an .out text file which shows exactly what the output should be.

Please refer to the Project 1 spec to recall how to execute and test your code.

7  Rubric

In this project, your code will be graded by a script, and this will account for most of your grade. We’ll grade linearly: if you pass 50% of the testcases, then we will give you 50% of the points.

This is the point distribution:

- 70% - Passing testcases
- 10% - Good indentation (see the Coding Standards document)
- 10% - Sufficient comments (at least 50% of the lines)
- 10% - Style

8  Turning in Your Solution

You must turn in your program using turnin on lectura. Use the assignment name cs252_f16_proj02. Turn in only your program (for example: proj02.s); do not turn in any testcases, grading scripts, or other files.
If you need to use a late day, simply add `_late` to the end of the assignment name. Remember, both turnin folders automatically close - the first one at the due date, and the second one 24 hours later!