Project : sprintf

Overview

You may choose to do this project by yourself or with a partner. This project is intent to make you familiar with MIPS coding and also binary representation of numbers. It will be your task to change from the computer representation to the human readable form, ie character. Note that the number 0 is not the same as char ‘0’ (look at their ASCII numbers). If you are not familiar with sprintf, then write a small C program where instead of using printf, you will use sprintf. It is basically the same behavior except the whole output string is put into a buffer (you supply) instead of the console output. Since printf/sprintf can take variable amount of arguments (ie more than 4) we need to use stack-argument passing protocol. The following section will explain how it works so read it before you get started.

Argument passing using the Stack

The MIPS procedure calling convention uses registers $a0-$a3 for passing arguments down to procedures. If there are more than four, the remaining arguments are passed on the stack. Each argument gets one word of stack space. Suppose we are trying to write in MIPS assembler a program like this:

```
int foo (int x, int y, int quux, int bar, int baz) {

     int a, b;
     ...
     a = y;
     ...
}

int main () {
     int c, d;
     ...
     foo (3, 4, 43, 62, 1);
     ...
}
```

Procedure foo has five integer arguments. The space for those arguments is allocated on the stack as part of the caller's stack frame. In other words, main , not foo , must allocate the space. The arguments go at the bottom of main's stack frame. That is, main will use 0($sp) to hold the argument x and 4($sp) to hold the argument y. The first argument is always at the top of the stack. you have to be consistent about this so that foo knows which argument is which. **N.B.:** *Stack space is allocated for **all** arguments, even those which are passed in the $a* registers. Those values are not put on the stack, but stack space is nonetheless reserved for them.*

```
main:
        ...
        addi $t0, $0, 3      # assume this holds the value of c
        addi $t1, $0, 4      # assume this holds the value of d
        ...
        addi $sp, $sp, -32   # make stack space for 8 words:
                             # $ra, c, d, and
                             # the args x, y, quux, bar, baz
```

```
        sw   $ra, 28($sp)    # save $ra
        sw   $t0, 24($sp)    # save c before calling foo
        sw   $t1, 20($sp)    # save d before calling foo
        add  $a0, $0, $t0    # arg x = c
        add  $a1, $0, $t1    # arg y = d
        addi $a2, $0, 43     # arg quux = 43
        addi $a3, $0, 62     # bar = 62
        addi $t0, $0, 1      # baz = 1, but no more registers
        sw   $t0, 16($sp)    # so pass on the stack
        jal  foo
        ...
        addi $sp, $sp, 32    # restore stack space
        lw   $ra, -4($sp)    # reload return address
        jr   $ra             # return to caller

foo:    addi $sp, $sp, -12   # make stack space for 3 words:
                             # $ra, a, b
        sw   $ra, 8($sp)     # save $ra
        ...
        add  $t0, $0, $a1    # get argument y
        lw   $t1, 28($sp)    # *** (see below)
                             # 12 (foo's frame) + 16 = 28 up on stack
                             # fetched argument baz
        ...
        addi $sp, $sp, 12    # restore stack space
        lw   $ra, -4($sp)    # reload return address
        jr   $ra             # return to caller
```
The instruction indicated by "***" is the key to understanding the stack method of argument passing. Procedure foo is referring to a word of stack memory that is from the caller's stack frame. Its own frame includes only the three words 0($sp) , 4($sp) , and 8($sp).

Project Details

Write a MIPS assembly language implementation of a function very similar to the C function sprintf:

```
int sprintf (char *outbuf, char *format, ...)
```

`sprintf` works like `printf`, except that it writes to the string outbuf instead of to standard output. outbuf is assumed already to point to allocated memory sufficient to hold the generated characters--see `man sprintf` for more information. Your function must accept any number of arguments, passed according to MIPS standard conventions: Stack space is allocated for all the arguments, but the first four arguments are passed in registers anyway; their stack space is unused. The first four arguments are passed in the $a0-$a3 registers, and the rest are passed on the stack.

The first argument is the address of a character array into which your procedure will put its results. The second argument is the address of a format string in which each occurrence of a percent sign (%) indicates where one of the subsequent arguments is to be substituted and how it is to be formatted. The remaining arguments are values that are to be converted to printable character form according to the format instructions. `sprintf` returns the number of characters in its output string not including the null at the end.

You do not have to do any error checking (e.g. comparing the number of arguments to the number of % specifications). You also do not have to implement all of the formatting options of the real `sprintf`. Here are the ones you are to implement:

- **%b**: unsigned integer argument to unsigned binary (treat the next argument as an unsigned integer and output in base 2)
- **%d**: signed integer argument to signed decimal (treat the next argument as an signed integer and output in decimal)
- **%u**: unsigned integer argument to unsigned decimal (treat the next argument as an unsigned integer and output in decimal)
- **%x**: unsigned integer argument to unsigned hexadecimal (treat the next argument as an unsigned integer and output in base 16)
- **%o**: unsigned integer argument to unsigned octal (treat the next argument as an unsigned integer and output in base 8)
- **%c**: character argument copied to output (treat the low byte of the argument word as a character and copy it to the output)
- **%s**: include a string of characters in output (treat the argument as a pointer to a null-terminated string to be copied to the output)
- **%%**: print a literal '%' character, with no conversion (no argument required)

Expected Outputs

We supplied the following test files for you :

spf-main.s :

```
97 characters:
string: thirty-nine, unsigned dec: 255, hex: 0xff, char: o, dec: -255,
bin: 11111111, percent: %
```

test1.s :

```
71 characters:
string: thirty-nine, unsigned dec: 255, hex: 0xff, char: o, dec: -255.
```

test2.s :

```
59 characters:
binary: 0b11111111, percent: %, octal: 0377, decimal: 255.
```

You should create your own tests to make sure your project works. You can look into spf-main.s where it is printing out how many characters got returned in $v0. That will give you a good starting point of how to implement your own sprintf for fields with numbers. Do not **FORGET** to put ‘\0’ or NULL termination at the end of your string buffer. These tests are not complete (missing corner cases) so be sure to test your solution throughly with your own cases by modifying spf-main.s. To create one file to run using a simulator that takes one input, use the following command :

```
cat spf-main.s sprintf.s > runtest.s
spim -file runtest.s
```

Submission

Just attach sprintf.s and put your parter’s name (if any) in the text submission box.