Administrative details

Submit a solution to this project by noon on Wednesday 9/30. The project should be done with your partner(s). Everything you turn in should include the names and lab section times of everyone in your partnership. Put a file of program source code named mips1.s in a directory named mips1 and submit it as you did for earlier assignments. Any notes you wish to share with the readers can be placed in comments in the program.

NOTE: THIS IS due BEFORE the C Project 2 which you may have already looked at.

Project description

Write a MAL implementation of a string-formatting routine inspired by the C function sprintf: int sprintf (char *outbuf, char *format, ...)

sprintf turns the characters that would be printed by a corresponding printf into a string. (You may have already encountered C++'s string streams, which provide a nicer version of this facility. Java's string buffers provide equivalent functionality. There is an especially elaborate version of formatted output available in Lisp, a facility that I suspect was put together in an attempt to mock all such formatting routines. It includes Roman numerals, spelled-out numbers like "one trillion five billion seventy-five", and the plural facility indicated below. You can look on-line in chapter 22 of the standard book on Common Lisp if you are curious about what other weird stuff might be included in format strings. A network browser pointed to

`http://www.cs.cmu.edu/Groups/AI/html/cltl/clm/node1.html`

will get you to this reference.) Your function must accept any number of arguments, all passed on the stack as described below. The first argument is the address of a character array into which your procedure will put its results. The second argument is 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).

Here are the format specifications you must implement. You might add some others if you have time on your hands. We suggest you not do any floating-point, because it is tedious:

%d convert integer to decimal

%x convert integer to hexadecimal

%c include one character argument in result

%s include string of characters in result

%% include a percent sign in result

%p (plural-normal): includes "s" if arg is 0 or >1 else nothing

%P (plural-y) includes "ies" if arg is 0 or >1 else includes "y"

Don't implement width or precision modifiers (e.g., %6d).

The plurals modifiers can be used to conveniently print format strings like "7 tries and 1 win" by sprintf ("%d tr%P and %d win%p",7,7,1,1).

Background

The procedure-calling convention we've been using up to now uses four registers ($a0-$a3) for passing arguments down to procedures.

What if there are more than four arguments? That approach won't work. For the sprintf project you will use an alternative convention, in which all arguments are passed on the stack, not in registers at all. Each argument gets one word of stack space. (One of the versions of the swap function you worked with in lab assignment 5 did this.) Suppose we are trying to write in MIPS assembler a program like this:

```
int foo (int x, int y) {
    int a, b;
    ...
    a = y;
    ...
}
int main () {
    int c, d;
    ...
    foo (3, 4);
    ...
}
```

Procedure foo has two 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.)

```
main:
      addi    $sp, $sp, -20 # five wds: $ra, c, d, arg x, arg y
      sw      $ra, 20($sp)  # save $ra
      ...
      addi    $t0, $0, 3     # first argument value is 3
      sw      $t0, 0($sp)    # save on stack
      addi    $t0, $0, 4     # second argument value is 4
      sw      $t0, 4($sp)    # save on stack
```

```
      jal     foo
      ...
      addi    $sp, $sp, 20
      jr      $ra

foo:
      addi    $sp, $sp, -12 # three wds: $ra, a, b
      sw      $ra, 8($sp)  # save $ra
      ...
      lw      $t0, 16($sp)   # get argument y *** (see below)
      sw      $t0, 4($sp)    # store as a
      ...
      addi    $sp, $sp, 12
      jr      $ra
```

*** This instruction is the key to understanding the stack method of argument passing. Procedure foo is referring to a word of stack memory that's beyond the boundary of its own stack frame. (Its own frame includes only the three words 0($sp), 4($sp), and 8($sp).) It thereby refers to the stack frame of its caller. This is legal only to the extent that the caller's stack frame contains foo's arguments! (foo doesn't know what's where on the rest of its caller's stack frame; it doesn't even know which procedure called it.)

For this project, although the arguments are passed on the stack, the return value should still be in $v0.

Miscellaneous requirements

Your sprintf procedure should work with the following main program. (Of course, it should not have any assumptions about this particular main program built into it!)

```
        .data
buffer: .space  200
format: .asciiz "%d%% of all %ss say %d %c %x!\n"
str:    .asciiz "American"
chrs:   .asciiz " characters:\n"

      .text
__start:
      addi    $sp, $sp, -32
      la      $t0, buffer     # first arg: place to put formatted
version
      sw      $t0, 0($sp)
      la      $t0, format     # second arg: format string
      sw      $t0, 4($sp)
      addi    $t0, $0, 87     # third arg: 87
      sw      $t0, 8($sp)
      la      $t0, str  # fourth arg: "American"
      sw      $t0, 12($sp)
      addi    $t0, $0, -5002 # fifth arg: -5002
      sw      $t0, 16($sp)
      addi    $t0, $0, 60     # sixth arg: '<'
      sw      $t0, 20($sp)
      addi    $t0, $0, 3840 # seventh arg: 0xf00
      sw      $t0, 24($sp)
      jal     sprintf
      add     $a0, $v0, $0    # should now contain 38
      jal     putint
      puts    chrs
      puts    buffer
      addi    $sp, $sp, 32
      done
putint:
      addi    $sp, $sp, -8    # void Print (int value) {
      sw      $ra, 0($sp)     #   if (value/10 != 0) {
      rem     $t0, $a0, 10    #     Print (value/10);
      addi    $t0, $t0, '0' #   }
      div     $a0, $a0, 10    #   print value%10;
      beqz    $a0, onedig     # }
      sw      $t0, 4($sp)
      jal     putint
      lw      $t0, 4($sp)
onedig:
      putc    $t0
      lw      $ra, 0($sp)
      addi    $sp, $sp, 8
      jr      $ra
```

This program is online in `~cs61c/lib/spf-main.s`.

P.S. The real MIPS argument passing convention is a combination of the two we've used. Stack space is allocated for all the arguments, but the first four arguments are passed in registers anyway; their stack space is unused.