Trabajo del tema 2: Ensamblador MIPS

EJ 6

Autor: Guillermo López García

Enunciado del Problema:

6. *printf con múltiples parámetros*

Consultar en el libro de Patterson y Hennessy cómo es el protocolo completo de llamada a función en MIPS. Ese protocolo permite pasar más de 4 parámetros usando la pila como lugar de almacenamiento.

Usando ese protocolo, programar una función printf similar a la que se hizo en clase (el problema está resuelto en el campus virtual) pero que acepte un número indeterminado de parámetros. Demostrarla llamándola desde el programa principal con al menos 6 parámetros.

Para empezar, he realizado la función en C con este resultado:

```c
#include <stdarg.h>
#include <stdio.h>
#include <string.h>

int myprintf(char s[], int count, ...)
{
    va_list list;
    int i = 0;
....
    va_start(list, count);
    for(i = 0; i < strlen(s); i++) {
        if(s[i] == '%' && count > 0) {
            switch (s[i+1]) {
                case 's': printf("%s", va_arg(list, char*)); break;
                case 'c': printf("%c", va_arg(list, int)); break;
                case 'i': printf("%i", va_arg(list, int)); break;
                case 'd': printf("%lf", va_arg(list, double)); break;
            }
            count--;
            i++;
        } else {
            printf("%c", s[i]);
        }
    }
    va_end(list);
....
    return count;
}

int main(int argc, const char *argv[])
{
    myprintf("abc%c %s %d %ajsdbjbdf", 3, 'd', "holi", 456.56);
    return 0;
}
```

Como es obvio, la traducción, no es literal, ya que, en ensamblador no tenemos forma de crear una lista como en un lenguaje de alto nivel como el C. En su defecto, hemos creado un pila.

Para empezar, cargamos los datos de memoria a registro, y posteriormente a pila. Así pues, queda:

```
1  .data
2  regexpr:      .asciiz "Yo soy un %s, un gran %s. Por lo t%cnto, hoy puede ser mi gran %s. La %iº de verdad."
3  parametro1:   .asciiz "trúan"
4  parametro2:   .asciiz "señor"
5  parametro3:   .asciiz "a"
6  parametro4:   .asciiz "noche"
7  parametro5:   .word   1
8
9  .text
10 .globl main
11 main:
12     # Se cargan todos los parametros en pila
13     la $s0, regexpr
14     addi $sp, $sp, -4
15     sw $s0, 0($sp)
16
17     la $s0, parametro1
18     addi $sp, $sp, -4
19     sw $s0, 0($sp)
20
21     la $s0, parametro2
22     addi $sp, $sp, -4
23     sw $s0, 0($sp)
24
25     la $s0, parametro3
26     addi $sp, $sp, -4
27     sw $s0, 0($sp)
28
29     la $s0, parametro4
30     addi $sp, $sp, -4
31     sw $s0, 0($sp)
32
33     la $s0, parametro5
34     addi $sp, $sp, -4
35     sw $s0, 0($sp)
```

Al no utilizar registros temporales en la función main, no guardo en pila los contenidos de los mismo. A continuación, paso los parámetros del puntero a la pila y el numero de parámetros pasados a través de la pila.

```
37     move $a0, $sp
38     li $a1, 6
```

Bien, el paso evidente es llamar a la función que corresponde.

```
40     jal printf
```

A continuación, veremos la función por partes.

Para empezar, inicializamos los registros que vamos a utilizar a sus valores por defecto.
Así pues, primero cargamos en memoria la cadena que vamos a enseñar con sus caracteres ‘%’ que simbolizan el cambio por el parámetro en cuestión.

```
45 printf:
46     move $t0, $a0        # copia del puntero a pila
47     move $t1, $a1        # max = numero de elementos
48
49     addi $t1, $t1, -1    # Se le quita uno al numero de parametros para despues hacerle el desplazamiento
50                          # De bits y sumarle al puntero de la pila para obtener la direccion de memoria del
51                          # primer parametro de la funcion
52     sll $t5, $t1, 2      # Se realiza el desplazamiento
53     add $t0, $t0, $t5    # Se le suma el desplazamiento al puntero de la pila
54
55     lw $t2, 0($t0)       # Se obtiene la direccion de memoria del primer parametro introducido en la pila
56                          # Asi pues, como tiene estructura LIFO, el primer entrar es el primero en salir
57
```

Poco después, empezamos un bucle donde recorremos la cadena inicial hasta que nos encontremos con el carácter terminador (‘\0’).

```
58      while:
59          lbu $t3, 0($t2)      # char aux = cad[i]
60          beqz $t3, end_while  # while(aux != '\0') {
```

También, a continuación, comprobamos con dos saltos condicionales si ya el carácter a enseñar es distinto del carácter especial ‘%’ o bien ya no quedan mas parámetros que enseñar. En este caso, se sigue enseñando los caracteres de forma normal.

```
62          bne $t3, '%', print  # if(aux != '%' ||
63          blez $t1, print      #    max < 0) {
```

```
111         print:
112             move $a0, $t3
113             li $v0, 11
114             syscall
```

Si bien, por el contrario, indica que es el carácter especial ‘%’ y que ademas quedan parametros por enseñar, así pues, entra en el switch:

```
65      switch:
66          addi $t1, $t1, -1  # max -= 1
67          addi $t0, $t0, -4  # puntero a pila decrementado, siguiendo la estructura contraria a lifo para enseñar los datos en orden
68
69          lbu $t3, 1($t2)    # aux = cad[i + 1]
70
71          beq $t3, 's', case_s  # caso para enseñar cadenas
72          beq $t3, 'c', case_c  # caso para enseñar un caracter
73          beq $t3, 'i', case_i  # caso para enseñar un numero entero con signo
74          beq $t3, 'f', case_f  # caso para enseñar un numero en precision simple con signo
75          beq $t3, 'd', case_d  # caso para enseñar un numero en precision doble con signo
```

Como podemos apreciar, tenemos el switch con los distintos case arriba del todo, donde posteriormente se verán los casos y las distintas rama por las que puede seguir el programa.

Así pues, el programa sigue una de las siguientes ramas:

```
77          case_s:
78              lw $t3, 0($t0)
79              move $a0, $t3
80              li $v0, 4
81              syscall
82              j end_switch
83          case_c:
84              lw $t3, 0($t0)
85              lbu $a0, 0($t3)
86              li $v0, 11
87              syscall
88              j end_switch
89          case_i:
90              lw $t3, 0($t0)
91              lw $a0, 0($t3)
92              li $v0, 1
93              syscall
94              j end_switch
95          case_f:
96              lw $t3, 0($t0)
97              lw $a0, 0($t3)
98              li $v0, 2
99              syscall
100             j end_switch
101         case_d:
102             lw $t3, 0($t0)
103             lw $a0, 0($t3)
104             li $v0, 3
105             syscall
106         end_switch:
107
108         addi $t2, $t2, 1
109         j end_print
```

Como se puede apreciar, toma la rama que precise y después salta al fin del switch, donde incrementa en uno el puntero y posteriormente, salta a “end_print” donde vuelve a incrementar otra vez el puntero, para eliminar el caso de ‘%c’, que solo sobre escriba el ‘%’, pero deje la ‘c’.

Para ir terminando, el fin de la función, con su correspondiente jr $ra y el incremento por segunda vez del puntero a la cadena, para evitar el caso erróneo mencionado en el párrafo anterior.

```
116         end_print:
117
118         addi $t2, $t2, 1          # *cad = (*cad + 1) Propio de las caracteristicas del C al bajo nivel
119         j while
120
121     end_while:
122
123     jr $ra
```

Para finalizar el programa, se carga en $v0 la instrucción 10 para finalizar el programa.

```
42      li $v0, 10
43      syscall
```

Por último, la mayor dificultad fue darme cuenta de que los parámetros en la pila se incluían en estructura LIFO y que debía modificar el puntero a pila añadiéndole el desplazamiento total para obtener el primer elemento y después, ir bajando. Es decir, no seguir la estructura LIFO con una pila.