MIPS sprintf project

Abdelsalam ElTamawy

900170376

1 Running the Program

Simply write sprintf ("$< text >$ % $< data\ presentation >$", $< x >$, ...) into the console writing the desired string in the first field, making sure to have it bound in quotation marks, including a "%" with your choice of data presentation following it, choosing one from the below table. Following that add a comma and list the variables you wish to be printed in order of first to appear in the string. Make sure each variable is separated by a comma. Each entry chosen must be one of the 5 variables a, b, c, d and e. The final resulting string's initial memory address with be found in the register $v0.

Note: ensure there is a space between "*sprintf*" and the "("

Example: sprinf ("This prints a as a decimal %d. This prints b as an unsigned integer %u. This prints a as an octal %o.", a, b, a)

syntax	result
d	save as a signed integer
u	save as unsigned integer
b	save in binary representation
x	save in lowercase hexadecimal presentation
X	save in uppercase hexadecimal presentation
o	save in octal presentation
c	consider only the lower byte(8 bits), saving it as an ASCII value
s	append string to designated position

2 Design

2.1 Main

The main function is used to load the variables and addresses to be used into the \$t5 to \$t9 registers and loads the output buffer into \$a0 and the input string to parsed into \$a1.Then jumps using JAL to the sprinf function. That's it for a developer user writing this functions appropriate code in main.

2.2 Parser

The parser increments the input string address byte by byte until it reaches the first quotation mark; from there it saves all future bytes to 10 bytes prior to the current pointer (based on "sprintf ("" being 10 characters) until the pointer reaches the second quotation mark where it stops sending bytes 10 spaces back, ending with a 0 to ensure it's null terminated. Here, we parse the alphabetic characters through running each byte till one with a value greater than 45 is found, this is value is then run through a set of cases to know which value from a to e to consider; accessing it from its expected location in the stack. The correct value is then added to the stack and the number of arguments is incremented; this number is late used to keep track of which value is to be called from the stack. This process continues until the null terminator is found.

2.3 String Analyzer

The address of the start on the input string that now has the proper string of characters to be analyzed is passed onto a new part of the code. This section of code increments through it byte by byte saving it into the output buffer accordingly until it reaches a '%' sign; then it gets the character next to it and runs it through cases to evaluate which format the value is to be outputted. The value in $\$sp - 4 * argument\ counter$ is then forwarded to the appropriate case and the argument counter is decremented. This process is then looped until a null is encountered.

2.4 Cases

2.4.1 Decimal

First, the most significant bit is checked if it is a 1. If it is, then a "-" is added to the output buffer and the number multiplied by -1. This number is then divided by 10, storing the remainder in the stack, until the intermediary value is 0. The values in the stack are then converted to characters and saved into output buffer in a reversed order to that pushed to the stack.

2.4.2 Unsigned Integer

The number is divided by 10 using the unsigned variant of the div (divu). The remainder of this division is stored in the stack until the number is equal to 0. Then the values in the stack are converted to characters and saved to the output buffer.

2.4.3 Binary

The passed number is shifted from 31 to 0 being decremented by 1. When this number is greater than 0, this resulting number of each loop is anded with 1 and converted to ASCII characters and pushed to the output buffer.

2.4.4 Uppercase Hexadecimal

The number passed is shifted from 28 to 0 with decrements of 4. Once the number is greater than 0, it is anded with 0xf. This resulting number is checked whether it is greater than 10; if it is, then the ASCII value of 'A' - 10 is added to the number. otherwise, the integer is converted to an ASCII character through adding the ASCII value '0' to it. Then this resulting ASCII value is pushed to the output buffer.

2.4.5 lowercase Hexadecimal

The number passed is shifted from 28 to 0 with decrements of 4. Once the number is greater than 0, it is anded with 0xf. This resulting number is checked whether it is greater than 10; if it is, then the ASCII value of 'a' - 10 is added to the number. otherwise, the integer is converted to an ASCII character through adding the ASCII value '0' to it. Then this resulting ASCII value is pushed to the output buffer.

2.4.6 Octal

The number passed is first shifted to the right by 30 and anded with 3. This value is then outputted to the output buffer in its appropriate ASCII format. Then the number is shifted from 27 to 0 with decrements of 3. When the number is greater than 0, it is anded with 7. This resulting number is then converted to ASCII and added to the output buffer.

2.4.7 Low Byte Character

The passed value is anded with 0xff and added to the output buffer.

2.4.8 String

The address of the string is passed. The address is traversed byte by byte with each byte being saved into the output buffer.

3 Challenges

1. How to store the values to be used (a e) in a functional, usable format.
2. How to properly store a signed number using its remainder
3. General stack management.
4. overlap and overwriting of stack values.

4 Limitations

1. The output buffer cannot be dynamic and is a set amount of allocated space. In this case it has a maximum of 10000 characters.
2. To implement the optimization of using the same string and using only one register that acts as a pointer, there has to be a space between the "sprintf" and the "("
3. unable to print "\n" since its reserved for new lines