We know how to use `syscall` to print out an integer value. The following program shows one possible implementation of `syscall` for printing an integer. That is, the program extracts the digits of the integer, pushes them (in character form) on to a stack and then prints the digits out by repeatedly popping the stack. (Since the digits are extracted from right to left, a stack is used to print the digits in the correct order.)

It is assumed that a non-negative integer value is stored in $17$. Register $15$ is used as the stack pointer for the character stack. Registers $3$, $16$ and $19$ are used for scratch work.

```assembly
.data

# Space for the character stack. (20 bytes will do for any 32-bit integer.)
cstack: .byte 0:20
newline: .asciiz "\n"

# A character must be output as a string of length 1. So, we need
# two bytes including the '\0' character. (Note that '\0' has ascii code 0.)
outbuf: .byte 0:2

.text

print_val: sw $31, 0($sp)  # Save $31 on the system stack.
addi $sp, $sp, -4
la $15, cstack  # Initialize char stack pointer.

# Obtain successive digits and push them on stack.
li $19, 10  # For division.
push_dig: rem $3, $17, $19  # Get least significant digit in $3.
addi $3, $3, 48  # Convert digit to its ASCII code.
jal push  # Push char in $3 on stack.
div $17, $17, $19  # Eliminate least significant digit.
bgtz $17, push_dig
```

(over)
#All digits (in character form) are now in the stack. Pop and print the digits.

```
la $16, cstack  #To know when to stop popping.

print_dig:  beq $15, $16, over
            jal pop  #pop returns the character in $3.
            sb $3, outbuf  #Move the character to the output buffer.
            la $a0, outbuf
            li $v0, 4  #print_string command.
            syscall

            j  print_dig

over:  la $a0, newline  #Print newline.
            li $v0, 4
            syscall
            lw $31, 4($sp)  #Restore $31 and pop stack.
            addi $31, $31, 4  #return to caller.
```

#Procedure push: The character to be pushed is in the last byte of $3.
#Does not check for stack overflow.

```
push:  sb $3, 0($15)  #$15 used as stack pointer for char stack.
            addi $15, $15, 1  #Increment char stack pointer by 1.
            jr $31
```

#Procedure pop: The popped character is in the last byte of $3.
#Does not check for stack underflow.

```
pop:  addi $15, $15, -1
            lbu $3, 0($15)
            jr $31
```