

MIPS Assembly Assignment

Zakariyya Kurmally
2315839

Ihsaan Ramjaneer
2315007

April 1, 2024

Contents

1	Pseudocode	3
2	The Code	3
2.1	Data Section	4
2.2	Text Section	4
3	Subprograms	7
4	Learning resources used	8

1 Pseudocode

```
● ● ●  
  
DECLARE count: INTEGER = 0  
DECLARE currNum: INTEGER = 0  
DECLARE maximum: INTEGER = 0  
DECLARE minimum: INTEGER = 999999  
DECLARE sum: INTEGER = 0  
  
WHILE count < 9  
    INPUT currNum  
  
    IF currNum == -69 THEN  
        BREAKWHILE  
    ENDIF  
  
    IF currNum < minimum THEN  
        minimum = currNum  
    ELSE IF currNum > maximum THEN  
        maximum = currNum  
    ENDIF  
  
    sum += currNum  
    count += 1  
  
ENDWHILE  
  
OUTPUT "Maximum:" + maximum + "\n"  
OUTPUT "Minimum: " + minimum + "\n"  
OUTPUT "Sum: " + sum + "\n"
```

2 The Code

There are 2 main sections of code, the data and text section. The first one is similar to preprocessor defines in C. They can be considered as constants. The text part is where the most of the actual code is.

2.1 Data Section

The following contains all the constants and variables used throughout the program.

```
.data
prompt: .asciiz "Enter number (-69 to finish): "
newline: .asciiz "\n"
maxNumbers: .word 10
maximum: .word 0
# Biggest signed 32-bit integer
minimum: .word 2147483647
sum: .word 0
maximumStr: .asciiz "Maximum is: "
minimumStr: .asciiz "Minimum is: "
sumStr: .asciiz "Sum is: "
```

2.2 Text Section

We first initialised the pointer and counter variables:

```
main:
# Load address of array into $t0
la $t0, array
# Initialise input counter to 0
li $t1, 0
lw $t4, maxNumbers
```

Afterwards, the loop label takes over. It starts by prompting an input from the user. After storing user input in \$t2, it compares it with a rogue value to see if the user want to stop. If that is not the case, it will branch to the updateMaximum label if the new number is greater than the current maximum. Finally, it will jump to the checkMinimum label.

```

loop:
    # Prompt user
    li $v0, 4
    la $a0, prompt
    syscall

    # Read input from user and move it to $t2
    li $v0, 5
    syscall
    move $t2, $v0

    # Check if user wants to finish
    li $t3, -69
    beq $t2, $t3, exitLoop

    # Update the maximum number
    lw $t5, maximum
    bgt $t2, $t5, updateMaximum
    j checkMinimum

```

The updateMaximum label stores the contents of the \$t2 register into maximum and jumps to the checkMinimum label. The later will update the current minimum, if need be, via the updateMinimum label. Otherwise, it will jump to the sumAndLoop label.

```

updateMaximum:
    sw $t2, maximum
    j checkMinimum

checkMinimum:
    lw $t5, minimum
    blt $t2, $t5, updateMinimum
    j sumAndLoop

updateMinimum:
    sw $t2, minimum

```

The sumAndLoop label adds up all the numbers entered by the user, keeps track of the number of inputs and restarts the main loop.

```

sumAndLoop:
    lw $t6, sum
    add $t6, $t6, $t2
    sw $t6, sum

    # Increment input counter
    addi $t1, $t1, 1

    # Check if all 10 numbers have been read
    bge $t1, $t4, exitLoop

    # We go again
    j loop

```

After the loop is done, the exitLoop label will execute. It will output the maximum, minimum and sum of the inputs entered using the smolprintf subprogram before exiting the program.

```

exitLoop:
    # Output Maximum
    la $a0, maximumStr
    lw $a1, maximum
    jal smolprintf

    # Output Minimum
    la $a0, minimumStr
    lw $a1, minimum
    jal smolprintf

    # Output Sum
    la $a0, sumStr
    lw $a1, sum
    jal smolprintf

    # Exit program
    li $v0, 10
    syscall

```

Below is the subprogram responsible for printing the aforementioned values. It uses the \$a0 and \$a1 registers as parameters which store the message and value to output respectively.

```

# Subprogram for integer output
# Arguments:
#   $a0: String to print
#   $a1: Integer to print
smolprintf:
    li $v0, 4
    syscall

    li $v0, 1
    move $a0, $a1
    syscall

    li $v0, 4
    la $a0, newline
    syscall

    jr $ra

```

3 Subprograms

The `smolprintf` subprogram helped to significantly reduce the number of lines of code. Before, to output a single line, the following was required:

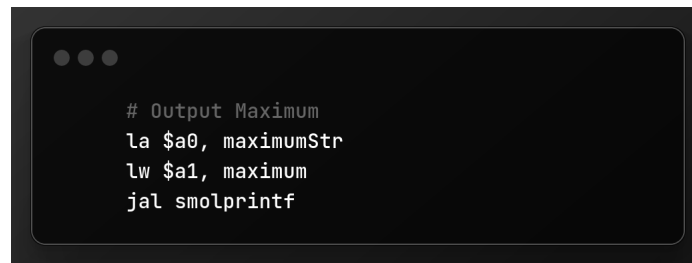
```

# Output new line
li $v0, 4
la $a0, newline
syscall

# Output maximum
li $v0, 4
la $a0, maximumStr
syscall

```

The subprogram `smolprintf` helps to significantly reduce this by about half:



```
# Output Maximum  
la $a0, maximumStr  
lw $a1, maximum  
jal smolprintf
```

4 Learning resources used

[MIPS Syscall table](#) - Syscall reference table

[Intro to MIPS](#) - Explanation of registers and hello world program