Is Palindrome Project

cps2390

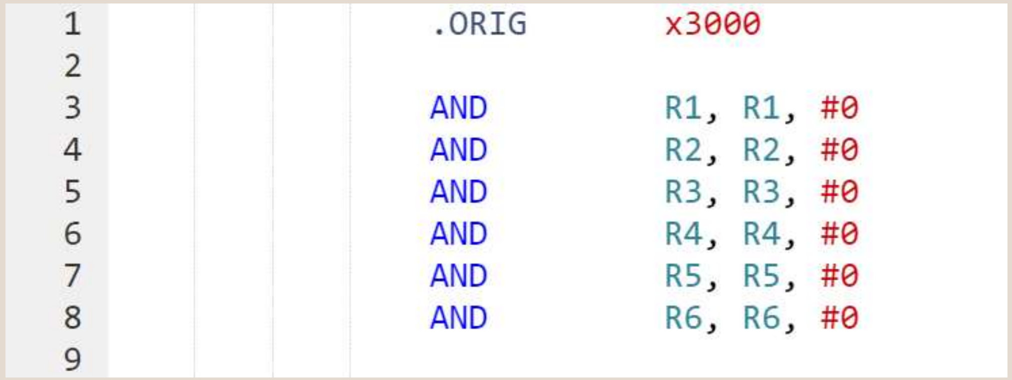
Introduction to theme selection:

The chosen topic here is about a palindrome checker implemented in assembly language. A palindrome is a sequence (such as a word or a number) that reads the same forward and backward.

Design decisions:

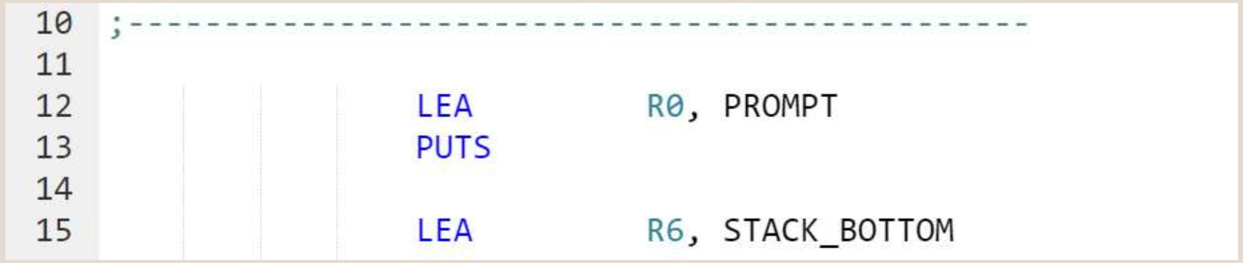
Design decisions include input handling through loops for string input until a new line, utilizing a stack for string storage considering register limitations, and error handling based on the stack pointer position for overflow or null input. Palindrome checking in the IS_PALINDROME subroutine involves two pointers, R3 and R6, comparing popped elements until a mismatch or reaching the bottom of the stack, determining the result. Output display varies based on palindrome status, overflow, or empty input.

Initialization: Registers R1 to R6 are cleared

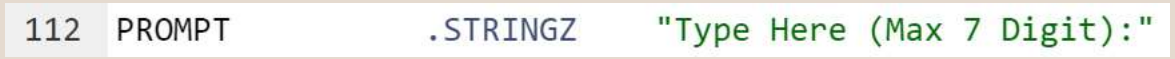


```
        .ORIG       x3000

        AND         R1, R1, #0
        AND         R2, R2, #0
        AND         R3, R3, #0
        AND         R4, R4, #0
        AND         R5, R5, #0
        AND         R6, R6, #0

```

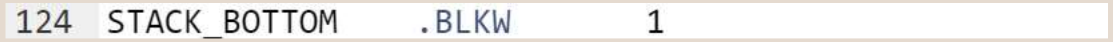
Input Handling:
The program prompts the user for input and reads characters one by one.



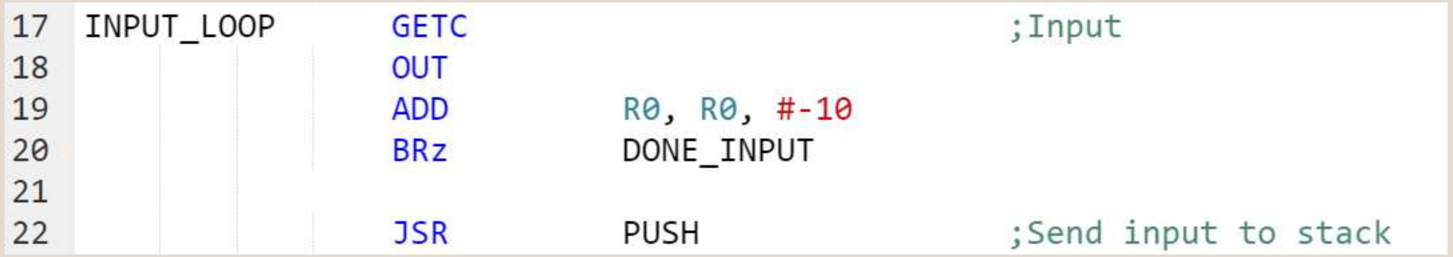
```
;----------------------------------------------

        LEA         R0, PROMPT
        PUTS

        LEA         R6, STACK_BOTTOM
```



```
PROMPT          .STRINGZ    "Type Here (Max 7 Digit):"
```

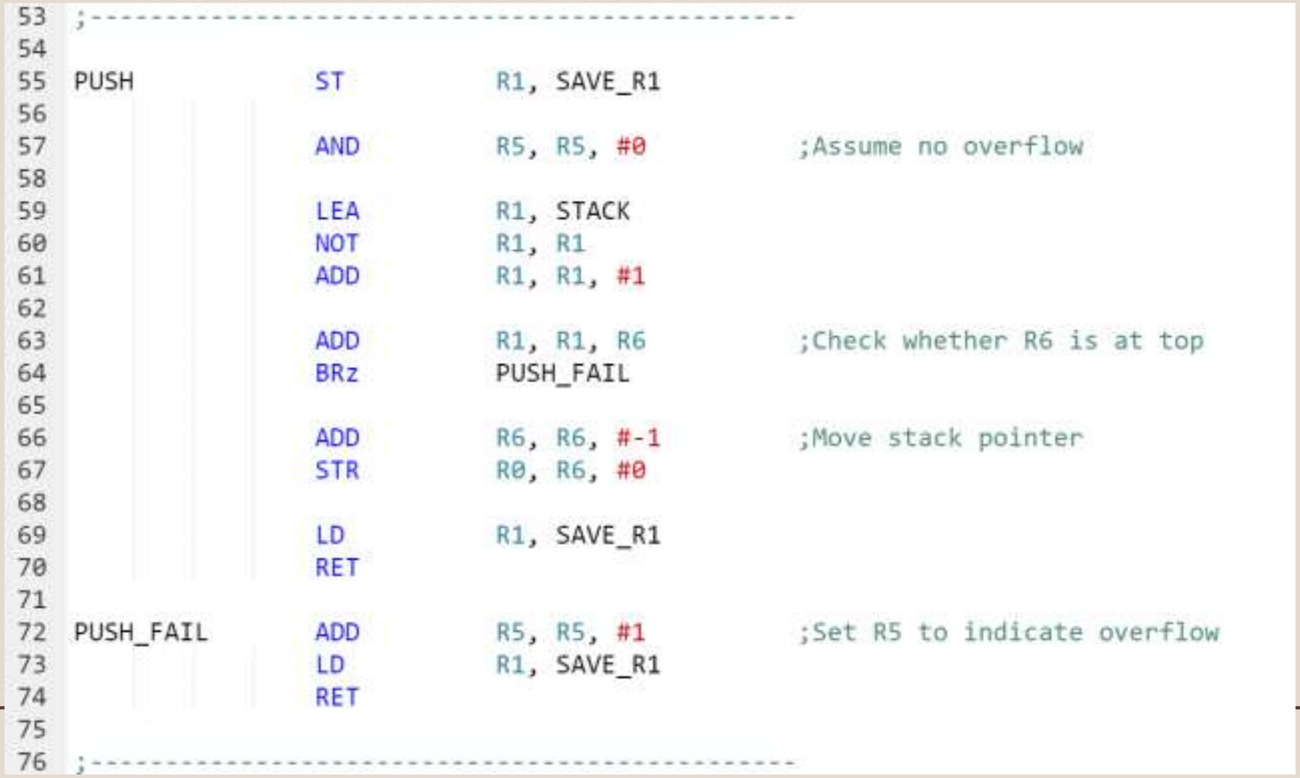


```
STACK_BOTTOM    .BLKW       1
```

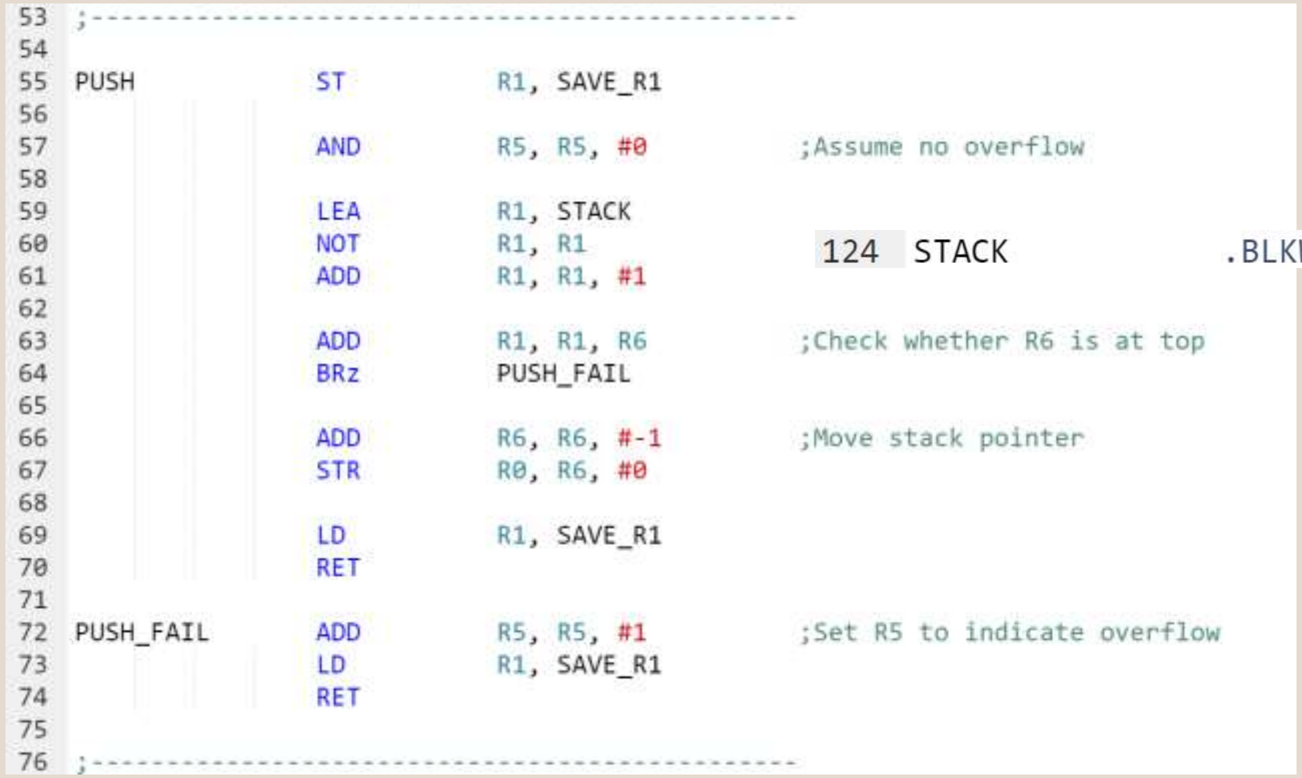


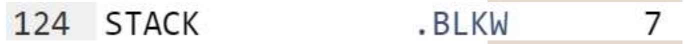
```
INPUT_LOOP      GETC                            ;Input
                OUT
                ADD         R0, R0, #-10
                BRz         DONE_INPUT

                JSR         PUSH                ;Send input to stack
```

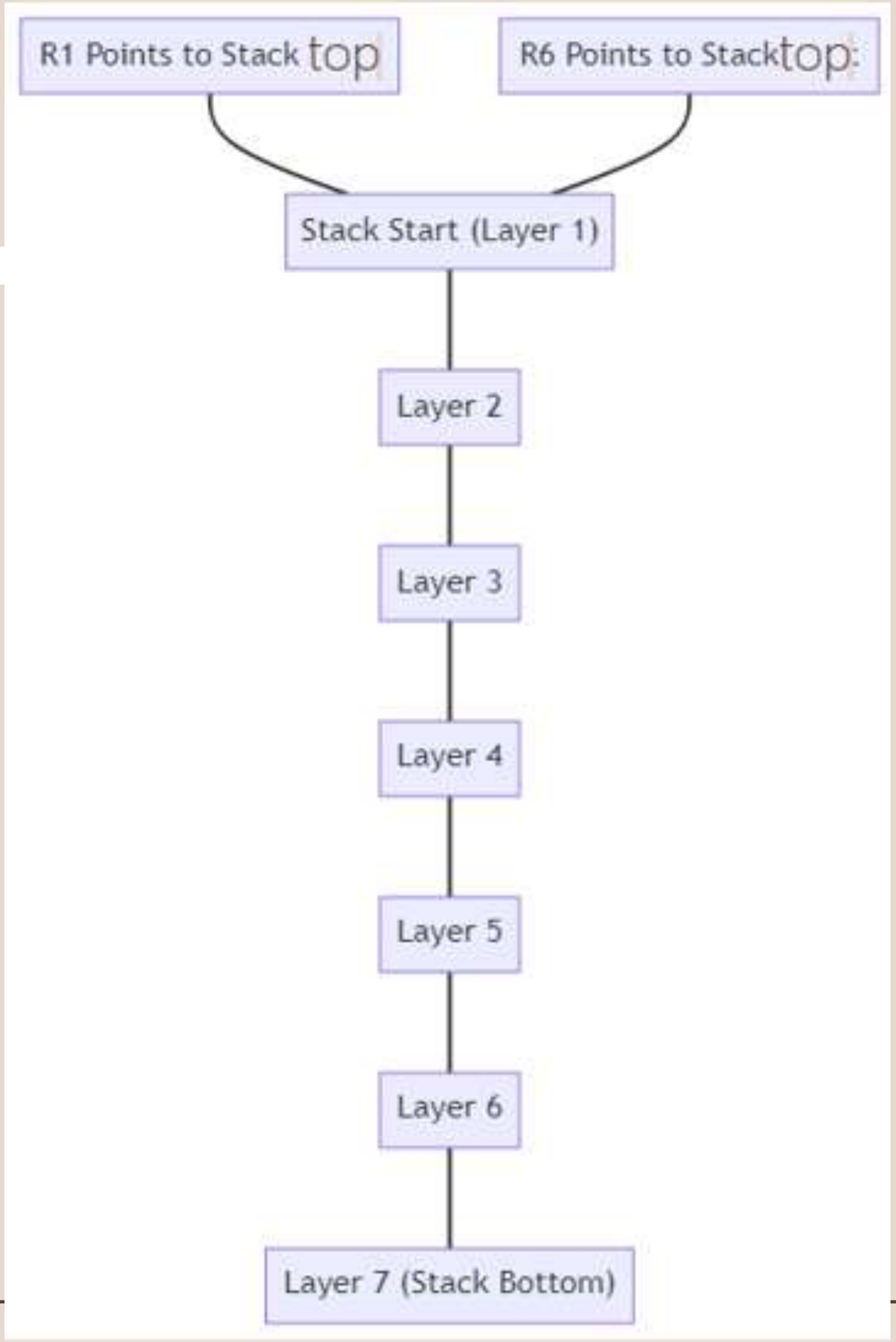
Jump to a PUSH subroutine, execute the code in PUSH, and then return to the code after the JSR instruction to continue execution



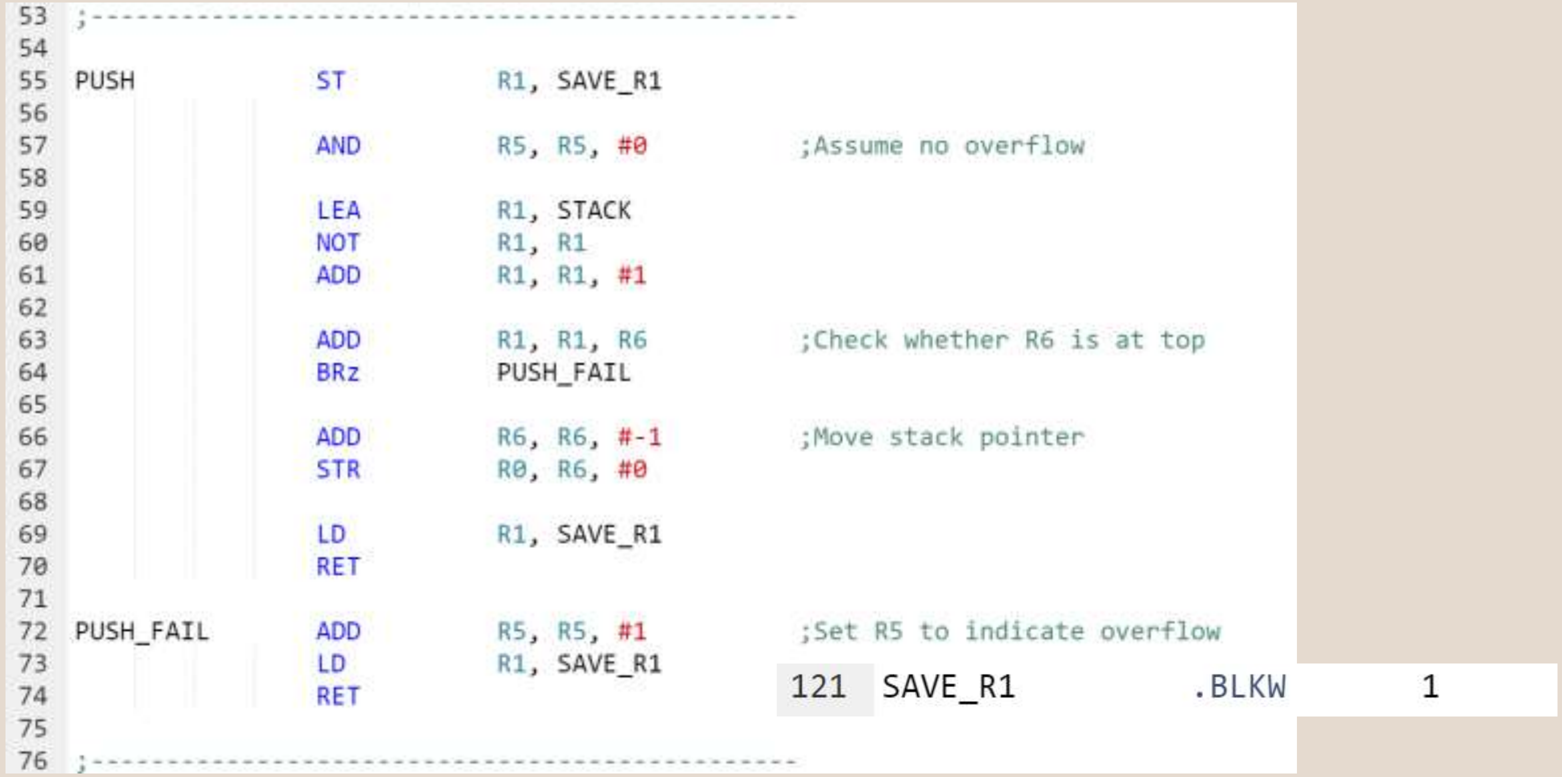
```
;-------------------------------------------

PUSH            ST          R1, SAVE_R1

                AND         R5, R5, #0          ;Assume no overflow

                LEA         R1, STACK
                NOT         R1, R1
                ADD         R1, R1, #1

                ADD         R1, R1, R6          ;Check whether R6 is at top
                BRz         PUSH_FAIL

                ADD         R6, R6, #-1         ;Move stack pointer
                STR         R0, R6, #0

                LD          R1, SAVE_R1
                RET

PUSH_FAIL       ADD         R5, R5, #1          ;Set R5 to indicate overflow
                LD          R1, SAVE_R1
                RET

;-------------------------------------------
```


```
;-----------------------------------------

PUSH            ST          R1, SAVE_R1

                AND         R5, R5, #0          ;Assume no overflow

                LEA         R1, STACK
                NOT         R1, R1
                ADD         R1, R1, #1

                ADD         R1, R1, R6          ;Check whether R6 is at top
                BRz         PUSH_FAIL

                ADD         R6, R6, #-1         ;Move stack pointer
                STR         R0, R6, #0

                LD          R1, SAVE_R1
                RET

PUSH_FAIL       ADD         R5, R5, #1          ;Set R5 to indicate overflow
                LD          R1, SAVE_R1
                RET

;-----------------------------------------
```


```
124 STACK                .BLKW       7
```

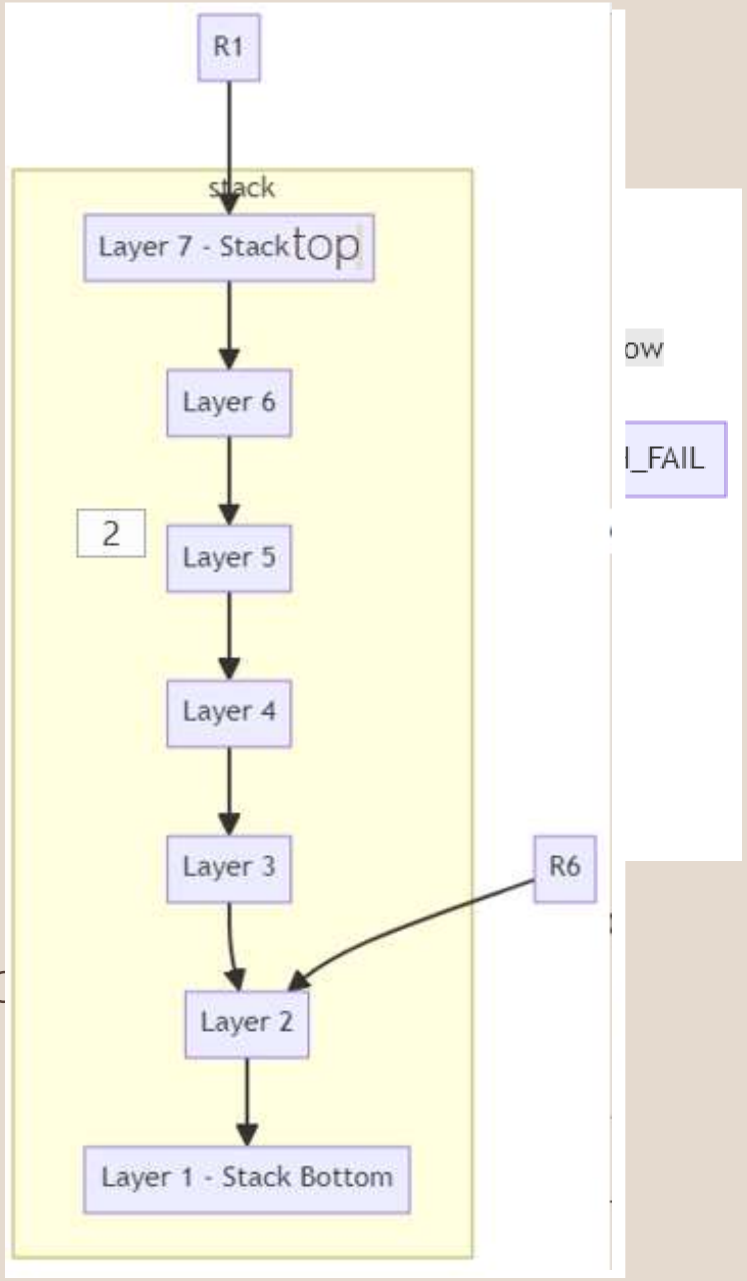


Save the current R1 value.
Check for stack overflow.
If there is no overflow, store the character on the stack and update the stack pointer R6.
Restore the value of the R1 register.
Use the RET instruction to return to the place where JSR PUSH was called to continue execution.

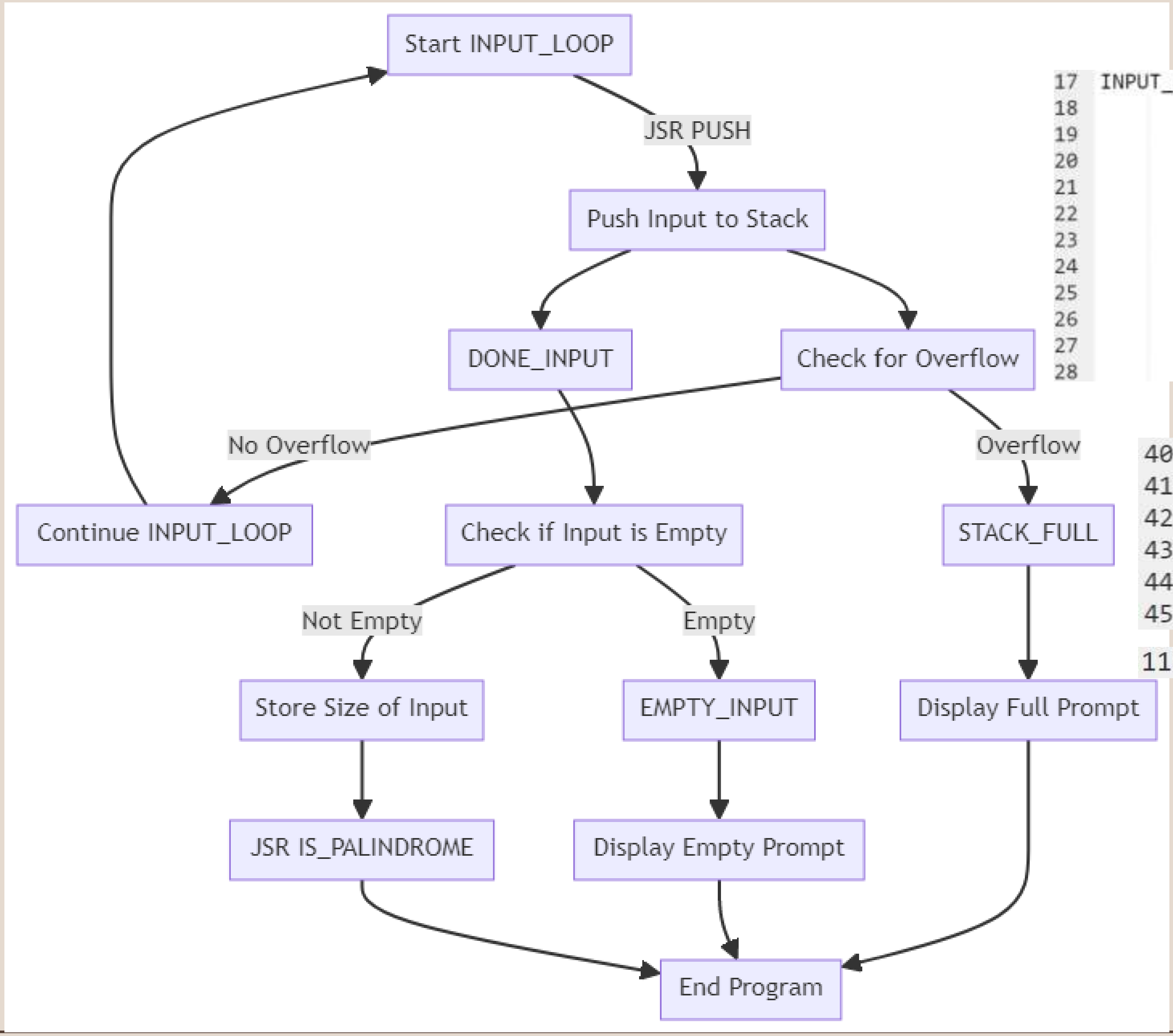


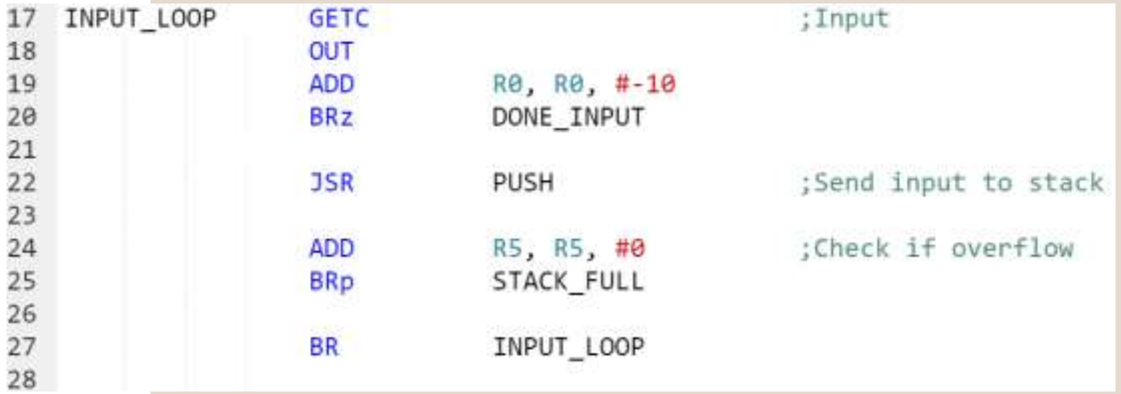
```
;-------------------------------------------

PUSH            ST          R1, SAVE_R1

                AND         R5, R5, #0          ;Assume no overflow

                LEA         R1, STACK
                NOT         R1, R1
                ADD         R1, R1, #1

                ADD         R1, R1, R6          ;Check whether R6 is at top
                BRz         PUSH_FAIL

                ADD         R6, R6, #-1         ;Move stack pointer
                STR         R0, R6, #0

                LD          R1, SAVE_R1
                RET

PUSH_FAIL       ADD         R5, R5, #1          ;Set R5 to indicate overflow
                LD          R1, SAVE_R1
                RET

;-------------------------------------------
```

```
121 SAVE_R1         .BLKW        1
```

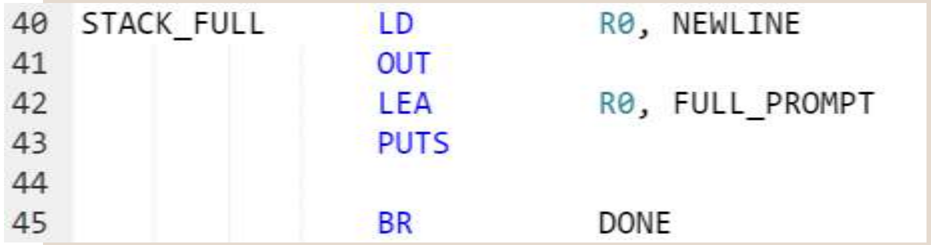


Save the current R1 value.
Check for stack overflow.
If there is no overflow, store the character on the stack and upc
stack pointer R6.
Restore the value of the R1 register.
Use the RET instruction to return to the place where JSR PUSH
to continue execution.

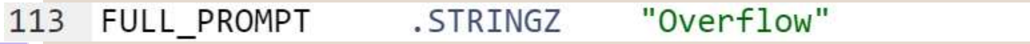




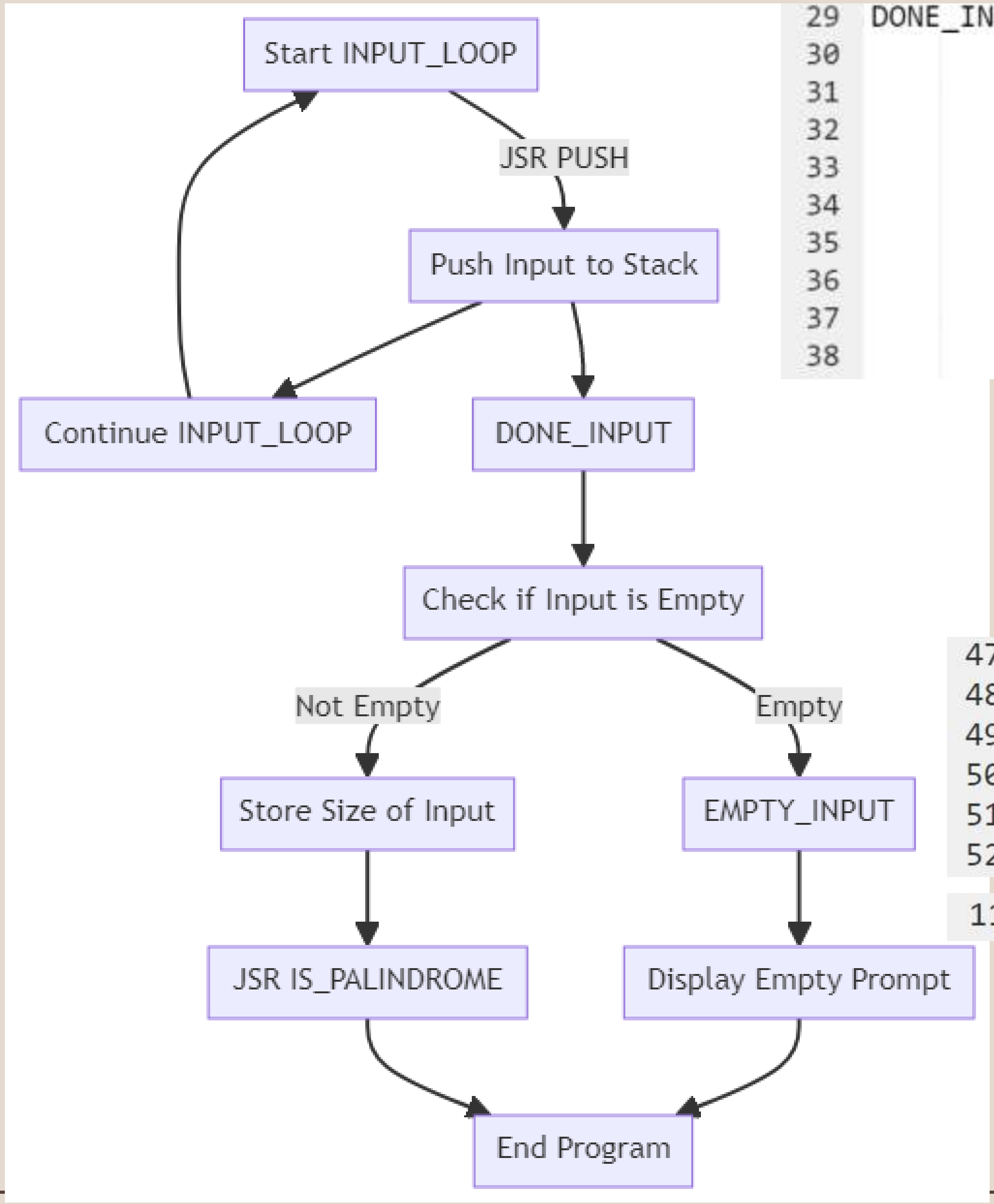
```
INPUT_LOOP      GETC                            ;Input
                OUT
                ADD         R0, R0, #-10
                BRz         DONE_INPUT

                JSR         PUSH                ;Send input to stack

                ADD         R5, R5, #0          ;Check if overflow
                BRp         STACK_FULL

                BR          INPUT_LOOP

```

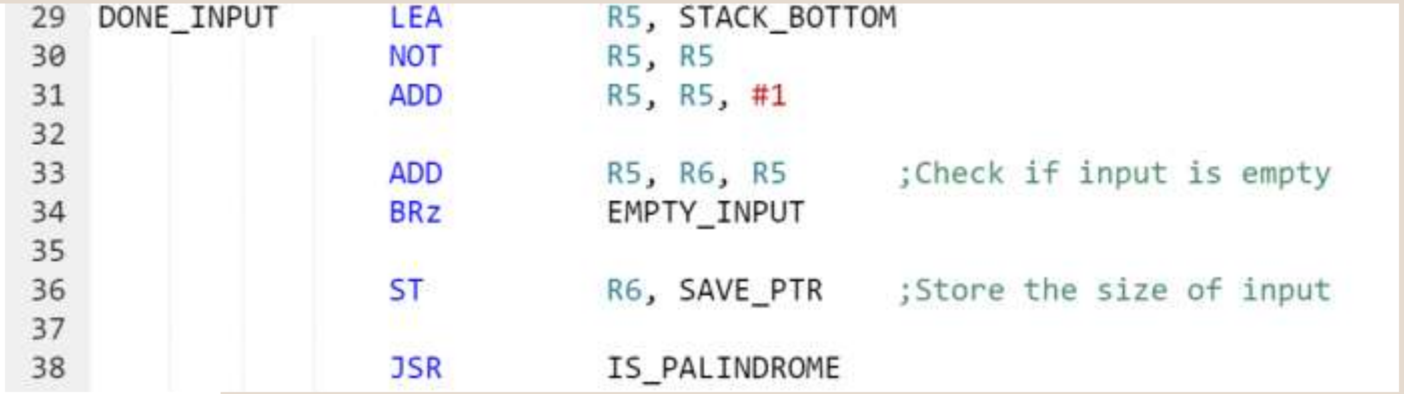


```
STACK_FULL      LD          R0, NEWLINE
                OUT
                LEA         R0, FULL_PROMPT
                PUTS

                BR          DONE
```

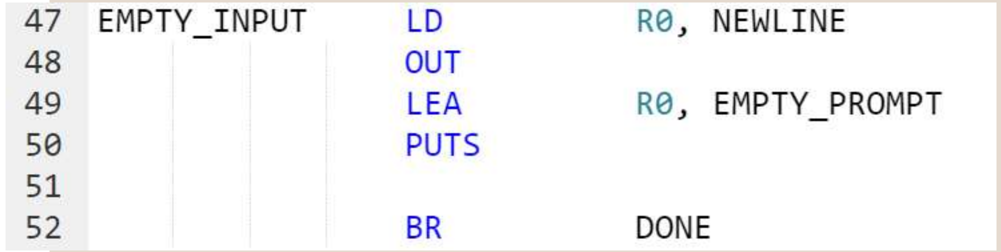


```
FULL_PROMPT     .STRINGZ    "Overflow"
```

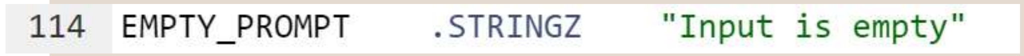




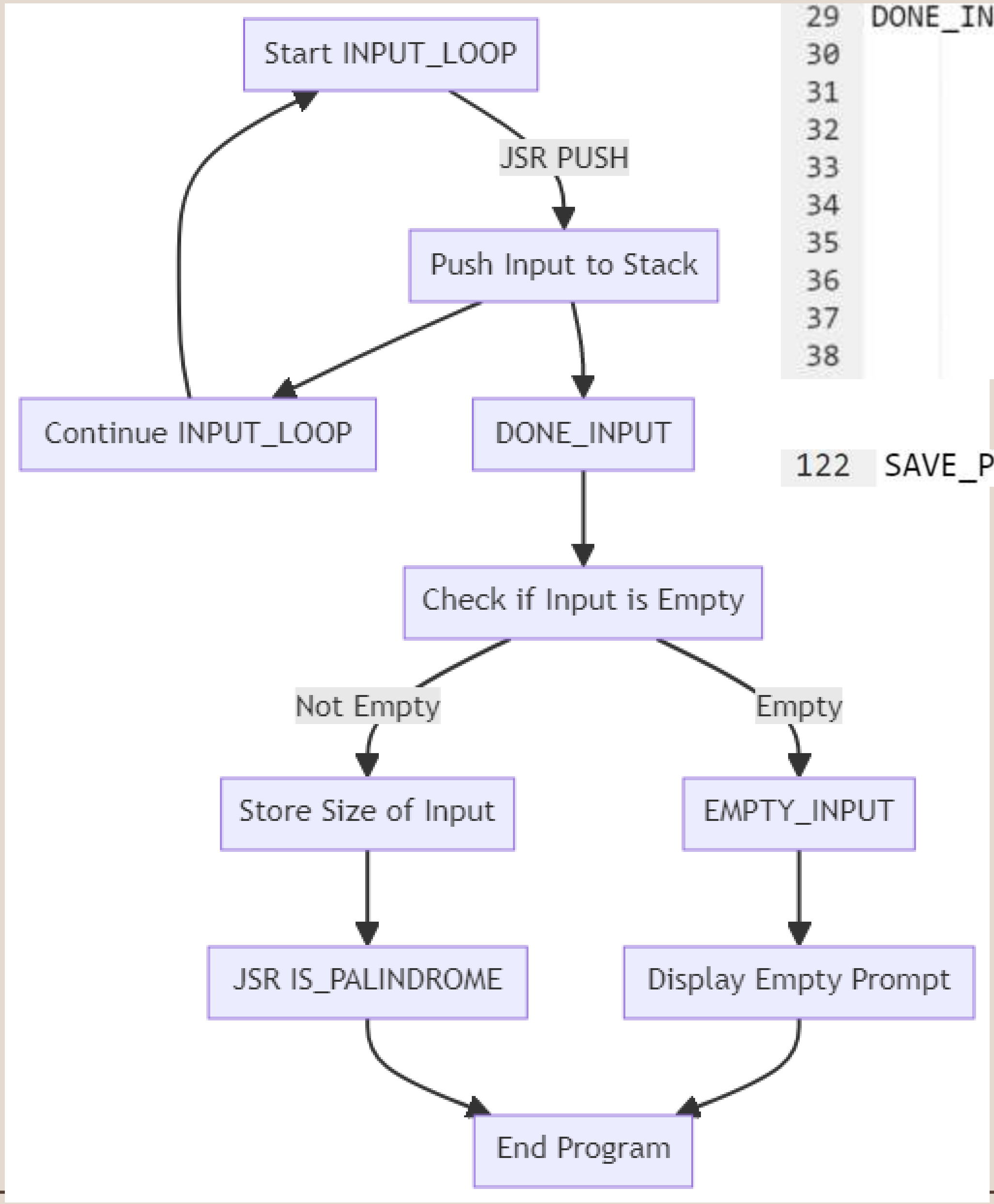
```
DONE_INPUT      LEA         R5, STACK_BOTTOM
                NOT         R5, R5
                ADD         R5, R5, #1

                ADD         R5, R6, R5      ;Check if input is empty
                BRz         EMPTY_INPUT

                ST          R6, SAVE_PTR    ;Store the size of input

                JSR         IS_PALINDROME
```

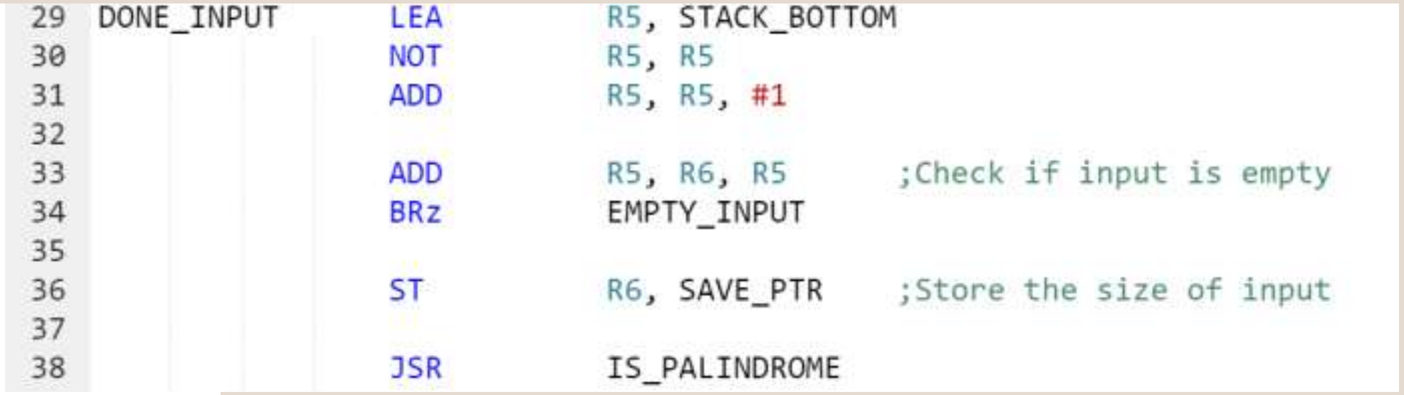


```
EMPTY_INPUT     LD          R0, NEWLINE
                OUT
                LEA         R0, EMPTY_PROMPT
                PUTS

                BR          DONE
```

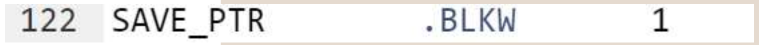


```
EMPTY_PROMPT    .STRINGZ    "Input is empty"
```

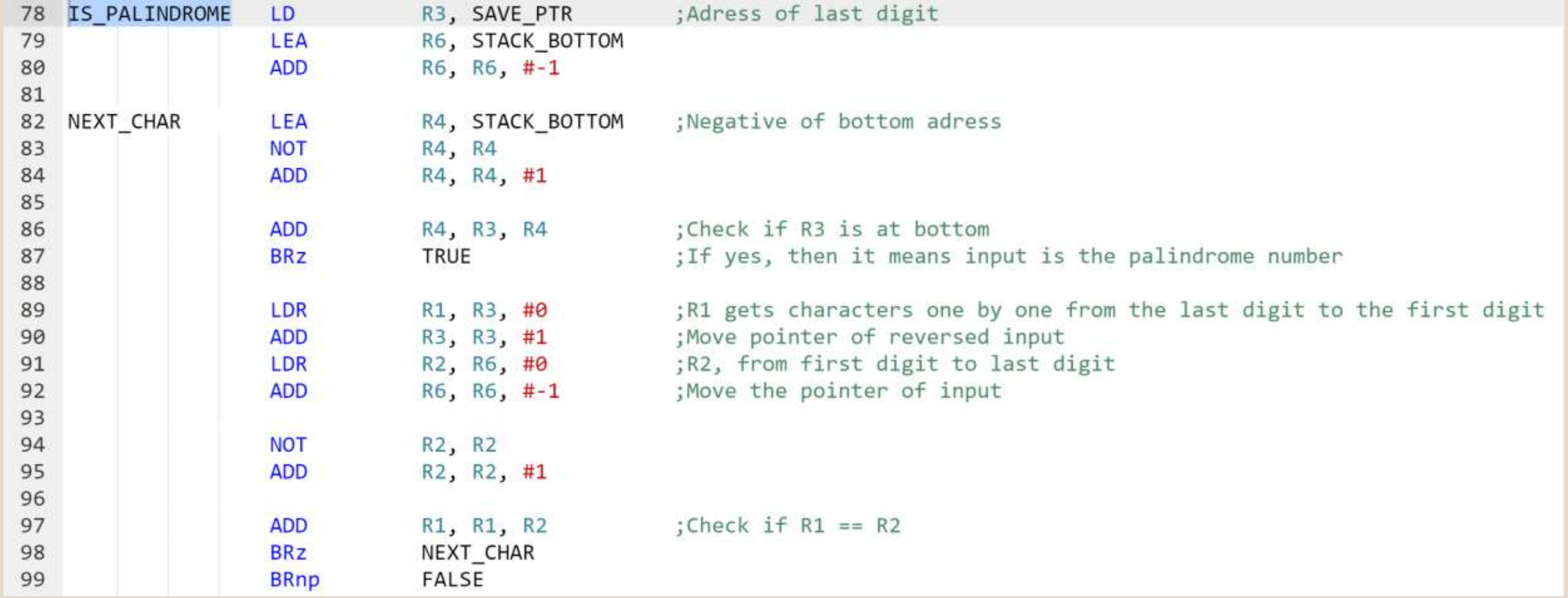




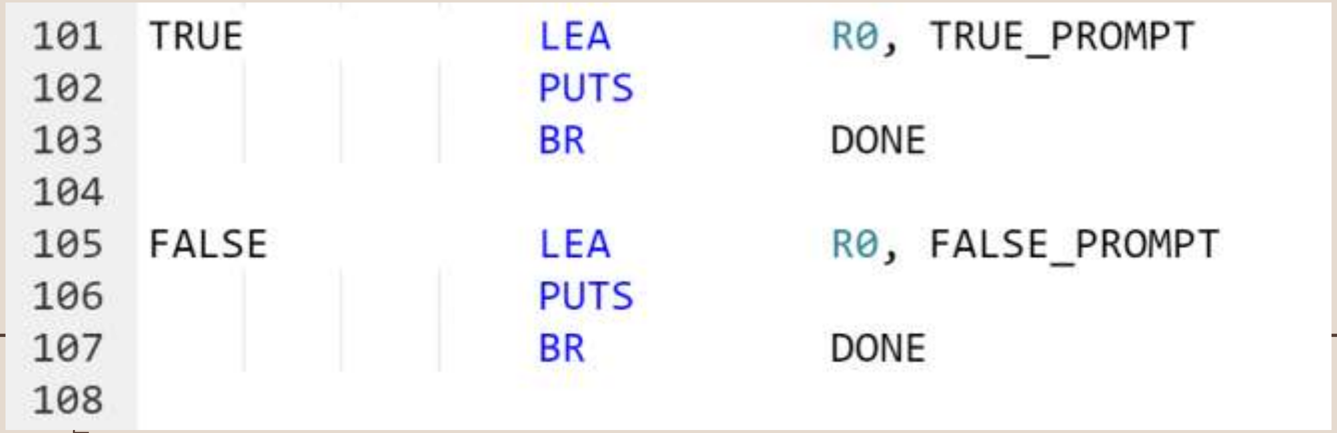
```
DONE_INPUT      LEA         R5, STACK_BOTTOM
                NOT         R5, R5
                ADD         R5, R5, #1

                ADD         R5, R6, R5      ;Check if input is empty
                BRz         EMPTY_INPUT

                ST          R6, SAVE_PTR    ;Store the size of input

                JSR         IS_PALINDROME
```



```
SAVE_PTR        .BLKW       1
```



```
IS_PALINDROME   LD      R3, SAVE_PTR        ;Adress of last digit
                LEA     R6, STACK_BOTTOM
                ADD     R6, R6, #-1

NEXT_CHAR       LEA     R4, STACK_BOTTOM    ;Negative of bottom adress
                NOT     R4, R4
                ADD     R4, R4, #1

                ADD     R4, R3, R4          ;Check if R3 is at bottom
                BRz     TRUE                ;If yes, then it means input is the palindrome number

                LDR     R1, R3, #0          ;R1 gets characters one by one from the last digit to the first digit
                ADD     R3, R3, #1          ;Move pointer of reversed input
                LDR     R2, R6, #0          ;R2, from first digit to last digit
                ADD     R6, R6, #-1         ;Move the pointer of input

                NOT     R2, R2
                ADD     R2, R2, #1

                ADD     R1, R1, R2          ;Check if R1 == R2
                BRz     NEXT_CHAR
                BRnp    FALSE
```



```
TRUE            LEA     R0, TRUE_PROMPT
                PUTS
                BR      DONE

FALSE           LEA     R0, FALSE_PROMPT
                PUTS
                BR      DONE

```