Stack

PUSH

R6 is the stack pointer that always points to the top of the stack. When we push a value onto the stack, the stack pointer is first decremented and then the value is stored. The following code pushes the value stored in R0 onto the stack.

We assume that the stack starts at $F000 and grows downward, so R6 must first be initialized with $F000.

```
PUSH    ADD   R6, R6, #-1
        STR   R0, R6, #0
```

POP

The value on the top of the stack is first read, and then the stack pointer is incremented. The following code pops the value stored on the top of the stack and loads it into R0.

```
POP     LDR   R0, R6, #0
        ADD   R6, R6, #1
```

If the stack pointer is stored in a memory location SP instead of in R6, then we need to load R6 from the memory location SP, and the memory location SP is initialized with $F000. Furthermore, since these two functions modify R6, we need to first save R6 to a known memory location, and restore it before returning to the caller.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
PUSH    ST    R6, saveR6     ; save register R6 to memory
        LD    R6, SP         ; get stack pointer
        ADD   R6, R6, #-1    ; decrement pointer
        STR   R0, R6, #0     ; store value to stack
        ST    R6, SP         ; save stack pointer
        LD    R6, saveR6     ; restore register R6 from memory
        RET

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
POP     ST    R6, saveR6     ; save register R6 to memory
        LD    R6, SP         ; get stack pointer
        LDR   R0, R6, #0     ; read value from stack
        ADD   R6, R6, #1     ; increment pointer
        ST    R6, SP         ; save stack pointer
        LD    R6, saveR6     ; restore register R6 from memory
        RET

saveR6  .BLKW #1
SP      .FILL xF000
```

Underflow – check when the stack is empty

To check for when the stack is empty, we can add the following check before doing the pop operation

```
POP       LD     R1, EMPTY
          ADD    R2, R6, R1      ; compare stack
          BRz    UNDERFLOW
          LDR    R0, R6, #0
          ADD    R6, R6, #1
          RET
UNDERFLOW       …                ; handle underflow situation

EMPTY     .FILL x1000           ; x1000 is -xF000
```