ECE 220 Computer Systems & Programming

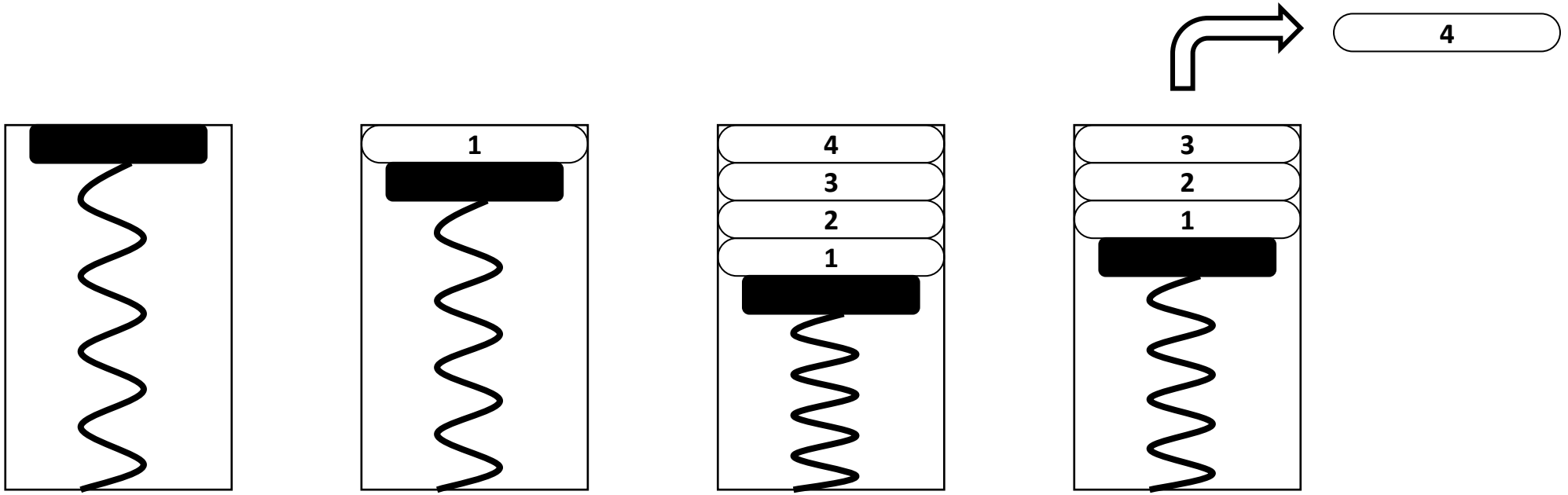
Lecture 3: Stack Data Structure and Stack Operations

January 23, 2024

ECE ILLINOIS

ILLINOIS

- MP1 due Thursday by 10pm
- Mock quiz next week.


4
1
4
3
2
1
3
2
1
Initial State
After
One Push
After Three
More Pushes
After
One Pop

Stack

Stack – an Abstract Data Type

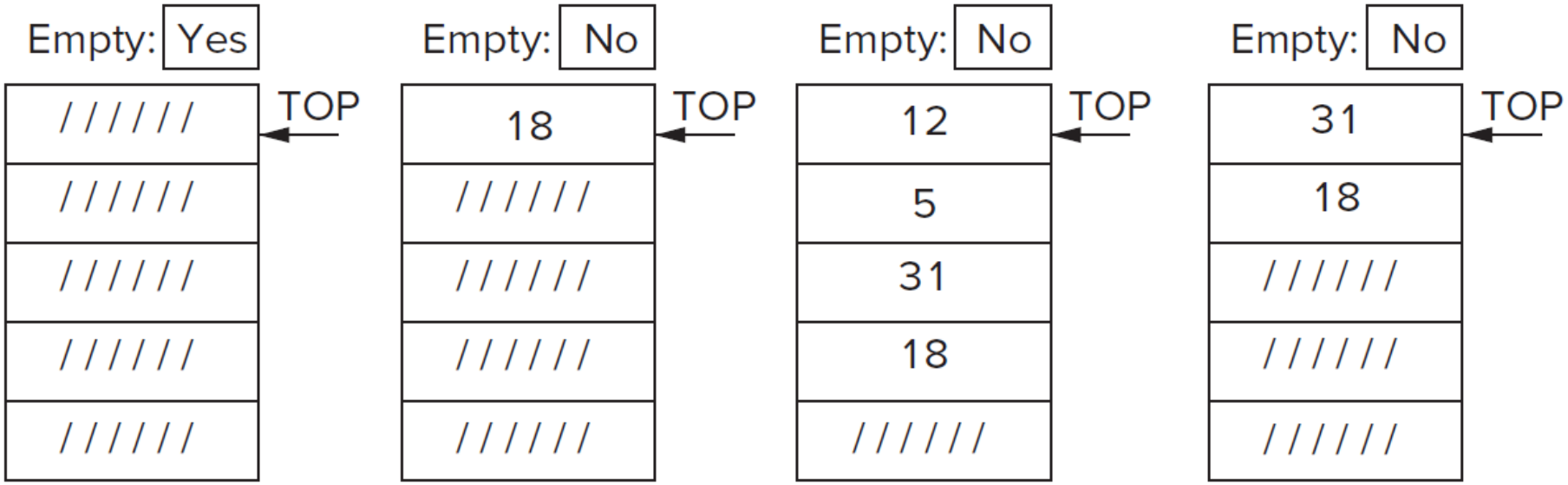
- Stack: A LIFO (Last-in First-out) storage structure
  - The first thing you put in is the last thing you take out.
  - The last thing you put in is the first thing you take out.
- This operation on the data is what defines a stack, not the specific implementation.

- Abstract Data Type (ADT): A storage mechanism defined by the operations performed on it.

  Example
  - Stack (LIFO)
  - Queue (FIFO: First-in First-out)
  - Linked list
  - Tree

Hardware Implementation of Stack

- Data items move between operations.



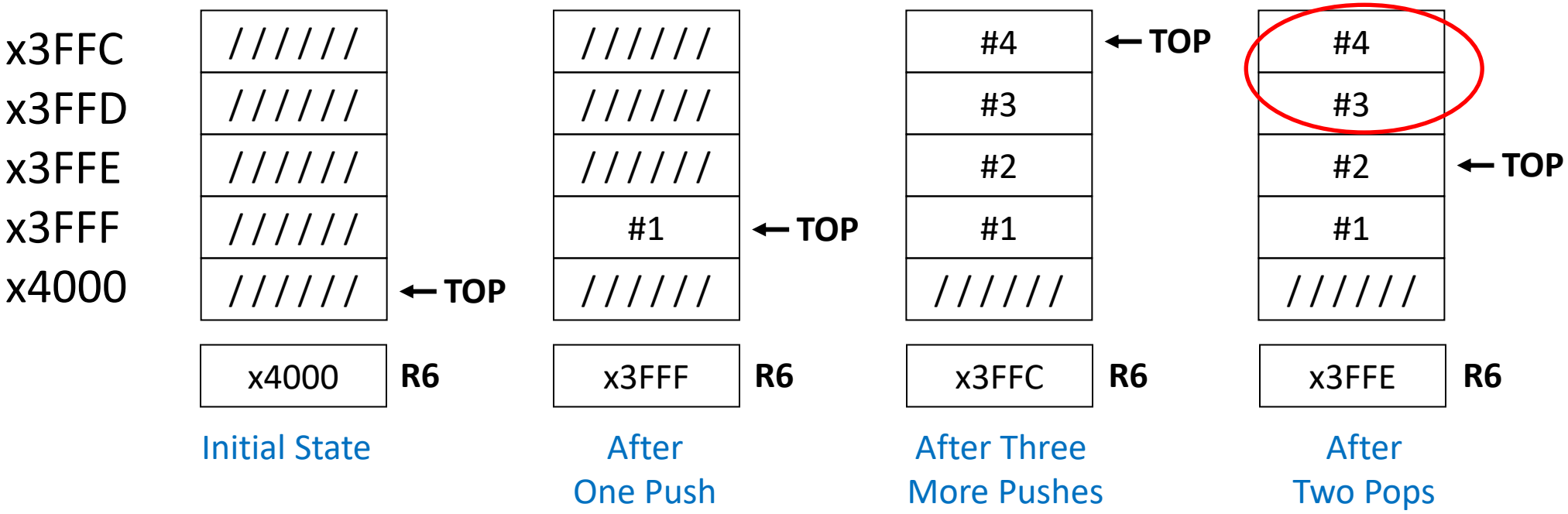
(a) Initial state (b) After one push (c) After three pushes (d) After two pops

problem?

Stack Implementation using memory– from textbook

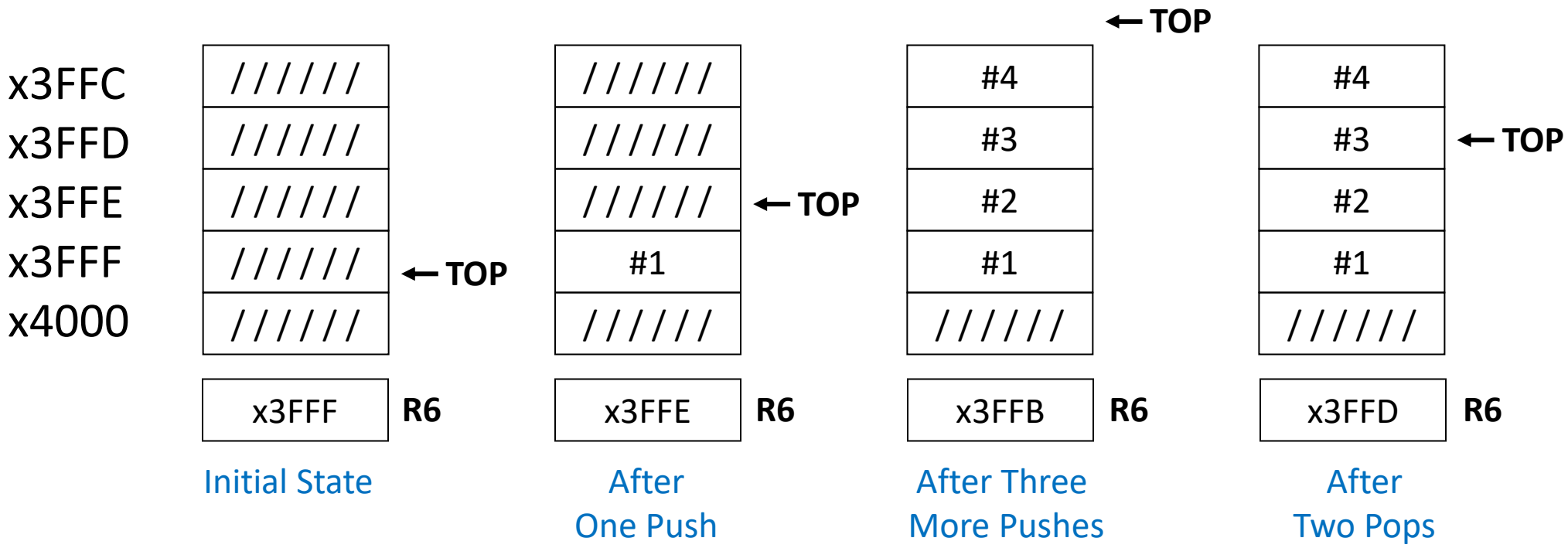
- Data items do NOT move in memory.
- Instead of moving the data, track the top of the stack.

They are still in memory
but cannot access
by stack anymore



- By convention, R6 holds the top of stack (TOS) pointer.
- When item added, TOS moves towards x0000

Another Implementation of stack - used in MP



TOS is pointing “Next available spot”

Exercise:

Worksheet

Stack Operation

1.

2.

3.

4.

*When item added, TOS moves closer to x0000.

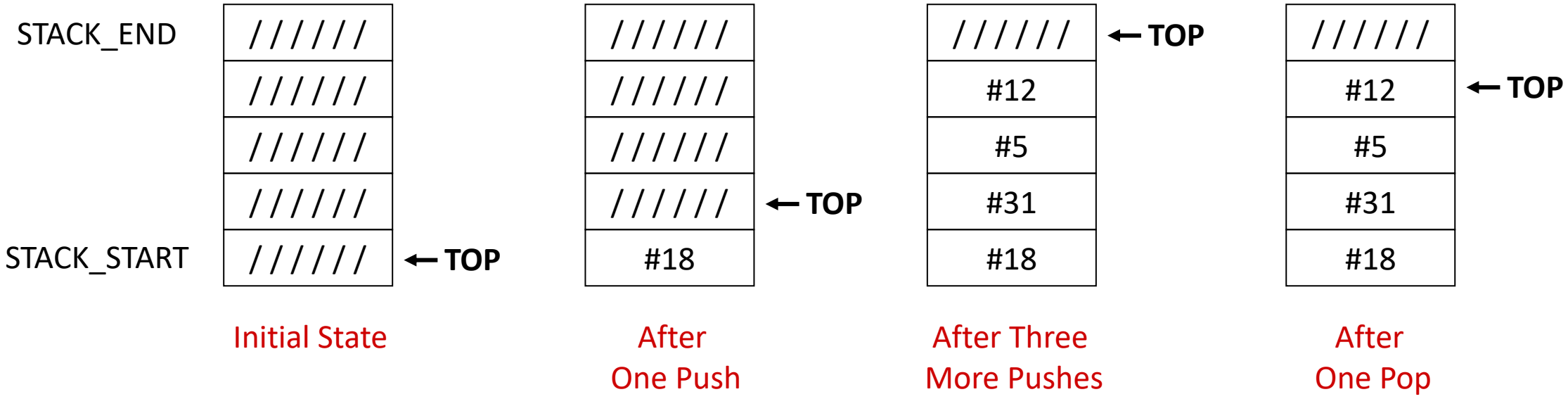
Basic PUSH and POP code

R0: input data

```
PUSH    STR R0, R6, #0 ; store data to TOS
        ADD R6, R6, #-1 ; decrement TOS pointer
```

R0: output data

```
POP     ADD R6, R6, #1 ; increment TOS pointer
        LDR R0, R6, #0 ; load data from TOS
```



Stack Implementation

- We label two memory locations
  - STACK_START to indicate the first memory location available for our stack
    - Stack is empty if value stored in STACK_TOP is the same as the value stored in STACK_START
  - STACK_END to indicate the last memory location available for our stack
    - Stack is full if the value stored in STACK_TOP is the same as the value stored in STACK_END decremented by 1
  - Example:

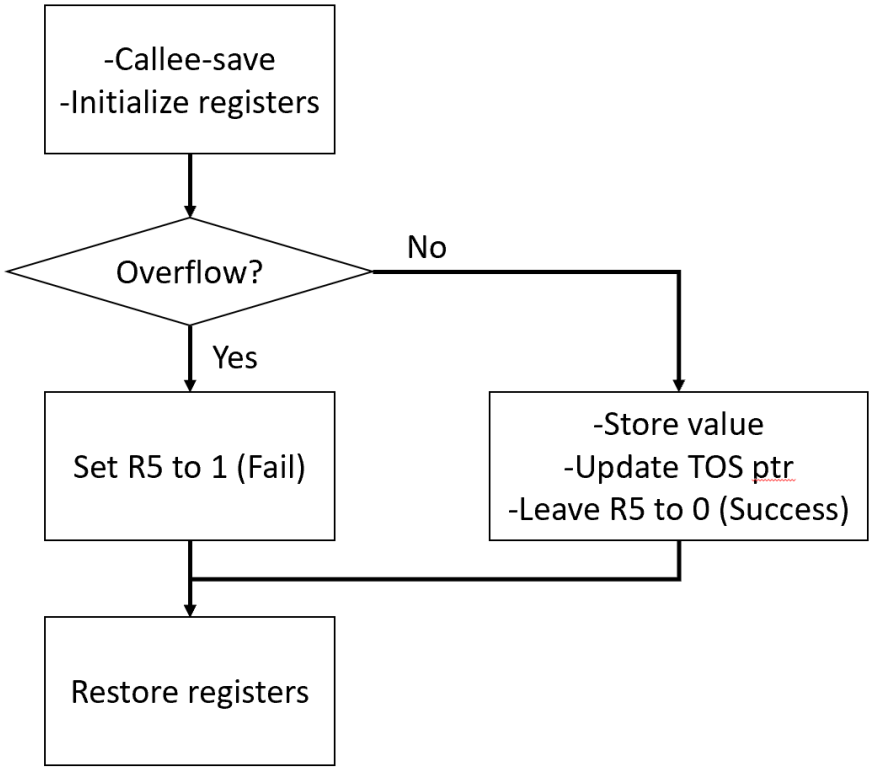
```
STACK_TOP   .FILL x4000
STACK_START .FILL x4000
STACK_END   .FILL x3FF0
```

    - Stack is located in memory at address x4000 – x3FF0 inclusive
    - First memory location available to add to the stack is at x4000

Implementation of PUSH Subroutine

- Argument
  - Value to be pushed onto the stack
  - Passed to the subroutine in R0
- Result
  - To indicate if push was successful
  - Will be returned in R5 (0 – success, 1 – fail)

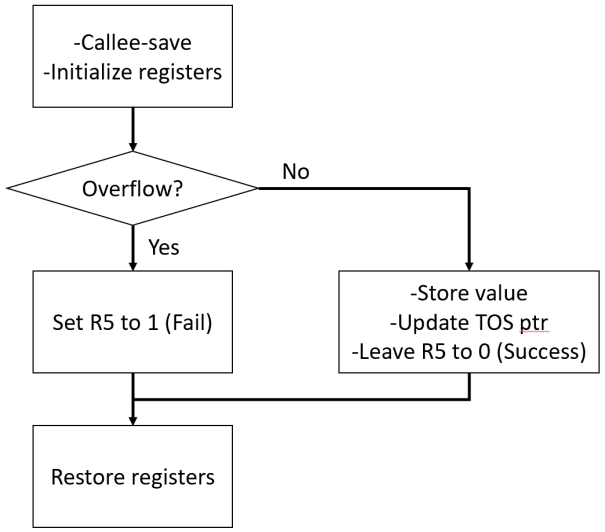
```
; IN: R0 (value)
; OUT: R5 (0 - success, 1 - fail)
; R3: STACK_END
; R6: STACK_TOP
;
PUSH
;
; prepare registers/Callee Save
    ST R3, PUSH_SaveR3   ; save R3
    ST R6, PUSH_SaveR6   ; save R6
    AND R5, R5, #0 ; clear R5, indicates success
    LD R3, STACK_END
    LD R6, STACK_TOP
```



```
; check for overflow (when stack is full)
```

;R3: STACK_END
;R6: STACK_TOP

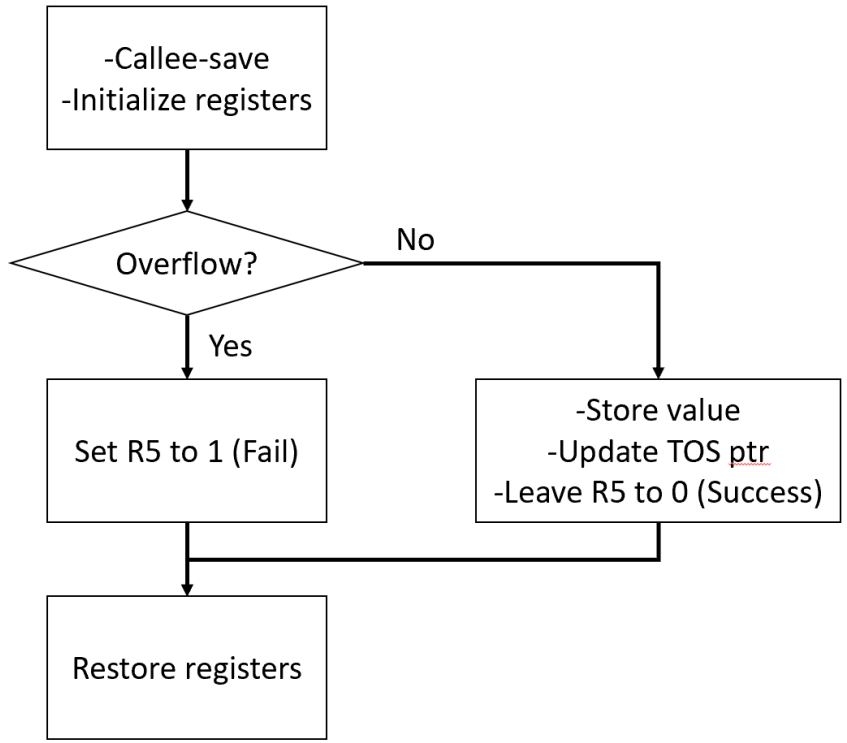
;overflow?
;Check if STACK_TOP = STACK_END - 1
;Or check if STACK_TOP - (STACK_END - 1) = 0




```
    BRz OVERFLOW ; stack is full
;
; store value in the stack
                    ; push onto the stack
                ; move top of the stack
     ST R6, STACK_TOP ; store top of stack pointer
    BRnzp DONE_PUSH
;
; indicate the overflow condition on return
OVERFLOW
    ADD R5, R5, #1
```

```
restore modified registers and return
DONE_PUSH
      LD  R3,  PUSH_SaveR3
      LD  R6,  PUSH_SaveR6
      RET
;
PUSH_SaveR3 .BLKW #1
PUSH_SaveR6 .BLKW #1

STACK_TOP   .FILL x4000
STACK_START .FILL x4000
STACK_END   .FILL x3FF0
```

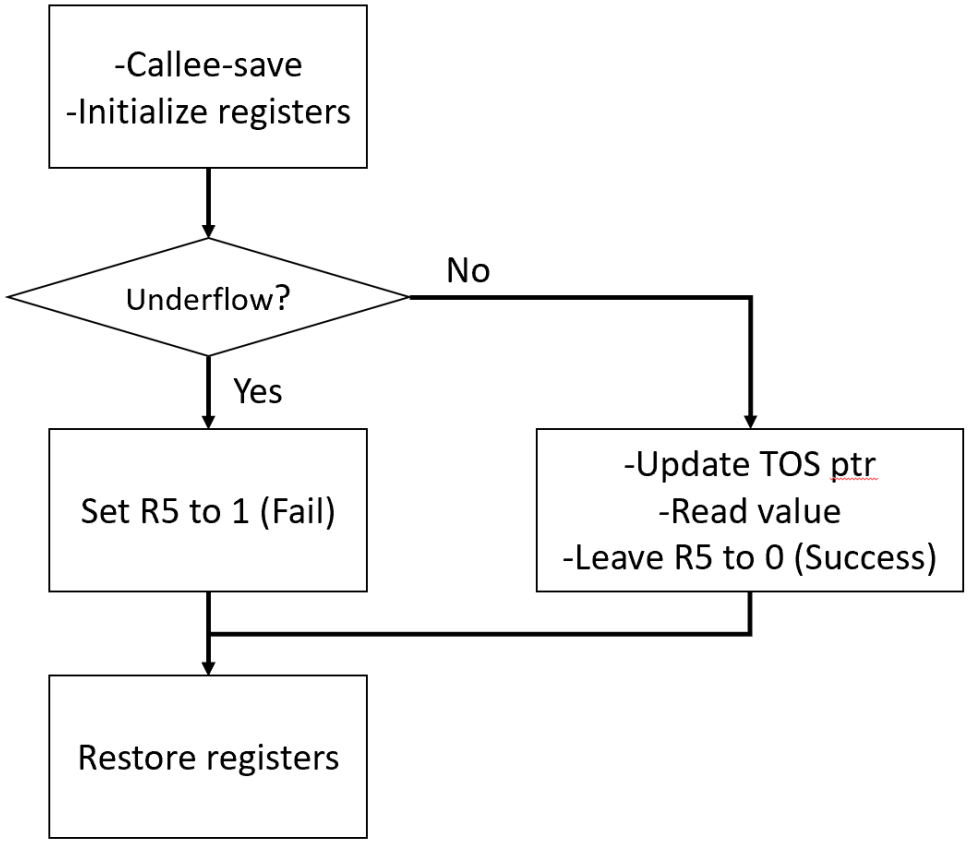


```
;----------------------PUSH/POP----------------------------
;IN:R0, OUT:R5 (0-success, 1-fail/overflow)
;R3: STACK_END R6: STACK_TOP
;
PUSH
    ST R3, PUSH_SaveR3   ;save R3
    ST R6, PUSH_SaveR6   ;save R6
    AND R5, R5, #0       ;
    LD R3, STACK_END     ;
    LD R6, STACK_TOP     ;
    ADD R3, R3, #-1      ; Stack End Decremented by 1
    NOT R3, R3       ;
    ADD R3, R3, #1       ;
    ADD R3, R3, R6       ;
    BRz OVERFLOW         ;stack is full
    STR R0, R6, #0       ;no overflow, store value in the stack
    ADD R6, R6, #-1      ;move top of the stack
    ST R6, STACK_TOP     ;store top of stack pointer
    BRnzp DONE_PUSH      ;
OVERFLOW
    ADD R5, R5, #1       ;
DONE_PUSH
    LD R3, PUSH_SaveR3   ;
    LD R6, PUSH_SaveR6   ;
    RET


PUSH_SaveR3 .BLKW #1     ;
PUSH_SaveR6 .BLKW #1     ;
```

Implementation of POP Subroutine

- Argument - none
- Result
  - Value to be popped of the stack
    - Passed from the subroutine in R0
  - Indicator if pop was successful
    - Will be returned in R5 (0 – success, 1 – fail)



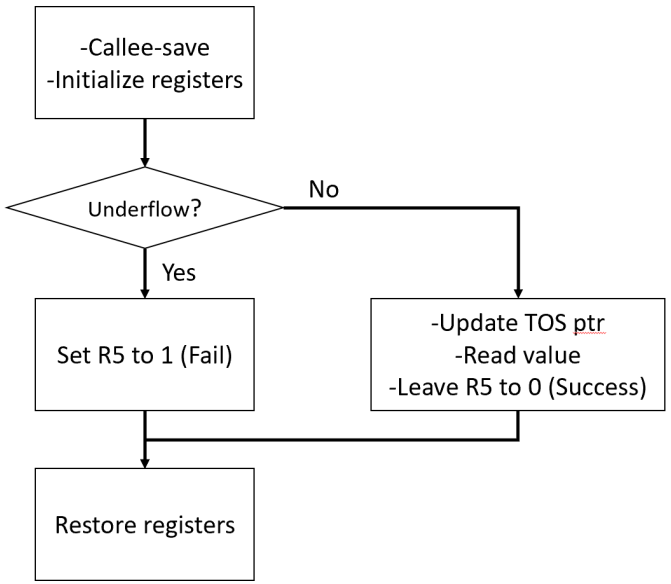

```
; OUT: R0 (value)
; OUT: R5 (0 - success, 1 - fail)
; R3: STACK_START
; R6: STACK_TOP
POP
; prepare registers/callee-save
    ST R3, POP_SaveR3  ; save R3
    ST R6, POP_SaveR6  ; save R6
    AND R5, R5, #0 ; clear R5, indicates success
    LD R3, STACK_START
    LD R6, STACK_TOP
```

```
; check for underflow (when stack is empty)
```

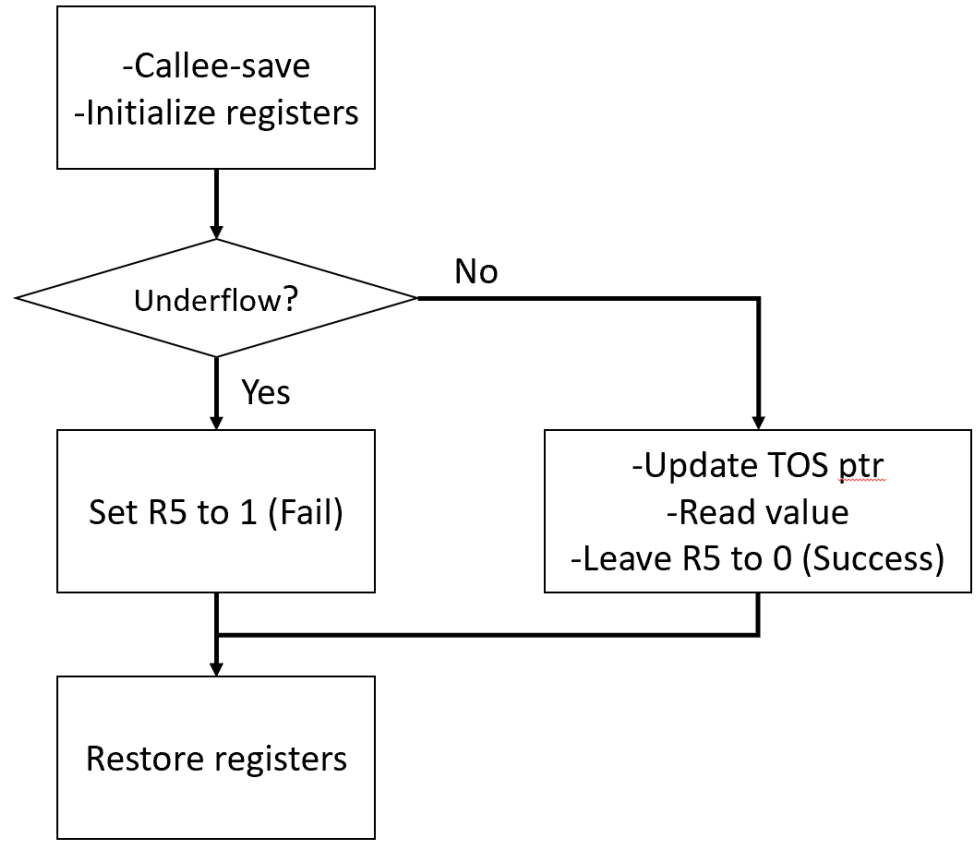
;R3: STACK_START
;R6: STACK_TOP

;underflow?
;Check if STACK_TOP = STACK_START
;Or check if STACK_TOP - STACK_START = 0

```
    BRz UNDERFLOW ; stack is empty, nothing to pop
;
; remove value from the stack
                    ; move top of the stack
                     ; read value from the stack
    ST R6, STACK_TOP ; store top of stack pointer
    BRnzp DONE_POP
;
; indicate the underflow condition on return
UNDERFLOW
    ADD R5, R5, #1
```



```
; restore modified registers and return
DONE_POP
     LD  R3,  POP_SaveR3
     LD  R6,  POP_SaveR6
     RET
;
POP_SaveR3 .BLKW #1
POP_SaveR6 .BLKW #1
```



POP Subroutine

```
;OUT: R0, OUT R5 (0-success, 1-fail/underflow)
;R3: STACK_START, R6: STACK_TOP
;
POP
    ST R3, POP_SaveR3    ;save R3
    ST R6, POP_SaveR6    ;save R6
    AND R5, R5, #0       ;clear R5
    LD R3, STACK_START   ;
    LD R6, STACK_TOP     ;
    NOT R3, R3       ;
    ADD R3, R3, #1       ;
    ADD R3, R3, R6       ;
    BRz UNDERFLOW        ;
    ADD R6, R6, #1       ;
    LDR R0, R6, #0       ;
    ST R6, STACK_TOP     ;
    BRnzp DONE_POP       ;
UNDERFLOW
    ADD R5, R5, #1       ;
DONE_POP
    LD R3, POP_SaveR3    ;
    LD R6, POP_SaveR6    ;
    RET


POP_SaveR3   .BLKW #1    ;
POP_SaveR6   .BLKW #1    ;
STACK_END    .FILL x3FFE ;
STACK_START .FILL x4000  ;
STACK_TOP    .FILL x4000 ;
```

Exercise 1:

Write a program that reads the contents of a sequence of memory locations and prints out in reverse order (but not changing the original memory contents). The starting and ending address is stored in R1 and R2.

- Use PUSH and POP subroutines. Assume the subroutines are provided in the code.
- You do not have to check the overflow condition.
- Use the underflow detection (R5) by POP to break LOOP_POP.
- *Example*

Address	Value
x4000 (starting addr)	x0
x4001	x1
x4002	x2
x4003 (ending addr)	x3

Result: 3210