ECE 220: Computer Systems & Programming

Lecture 4: Stack Data Structure and Stack Operations

Thomas Moon

January 25, 2024

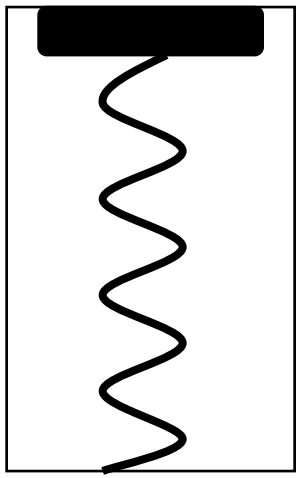
- MP1 due tonight 10pm
- Mock quiz next week 01/30 - 02/01

Previous lecture

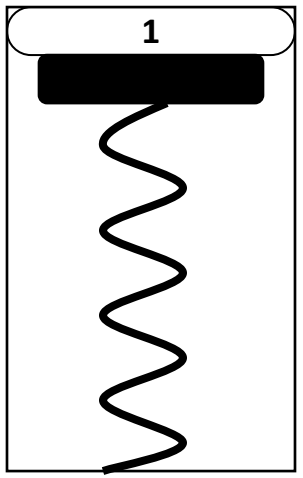
- TRAPs: `GETC, IN, OUT, PUTS, PUTSP, HALT`
- Subroutines: `JSR, JSRR`
- Demystify `R7`
- Caller-save, Callee-save

Today's lecture

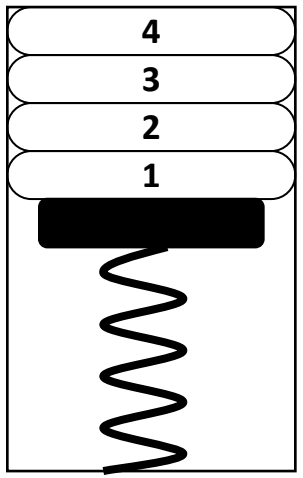
- Stack!



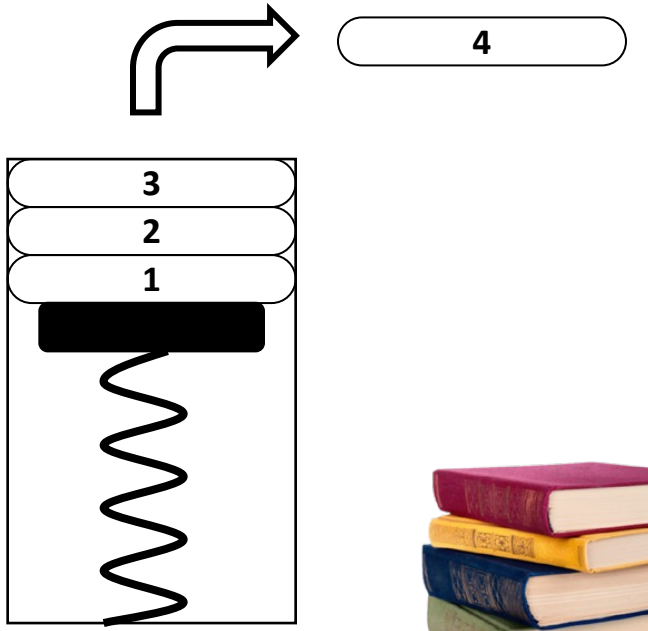
Initial State



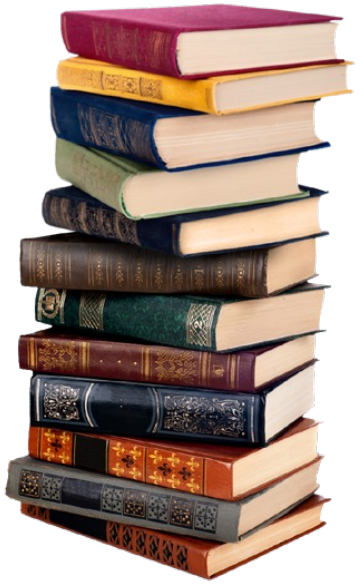
After
One Push



After Three
More Pushes



After
One Pop



Stack

The Last thing In is the First thing Out.

Stack – an Abstract Data Type

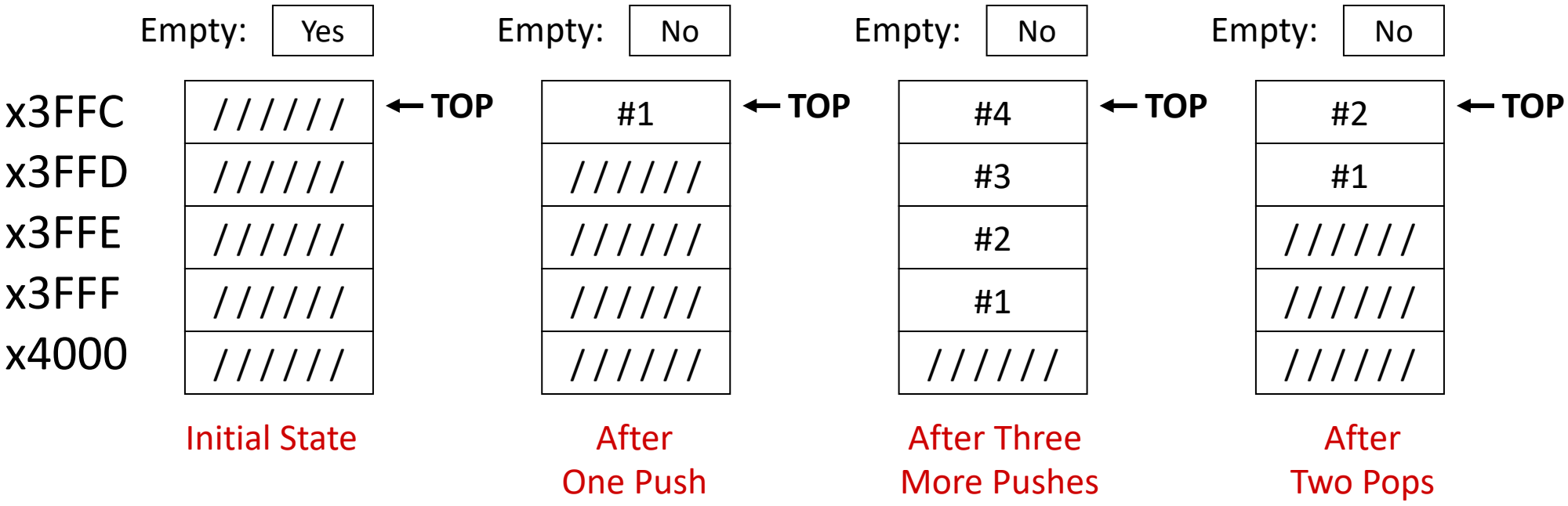
- Stack: A LIFO (Last-in First-out) storage structure
  - The first thing you put in is the last thing you take out.
  - The last thing you put in is the first thing you take out.
- This operation on the data is what defines a stack, not the specific implementation.

- 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

Stack Implementation 1

- Data items move between operations.

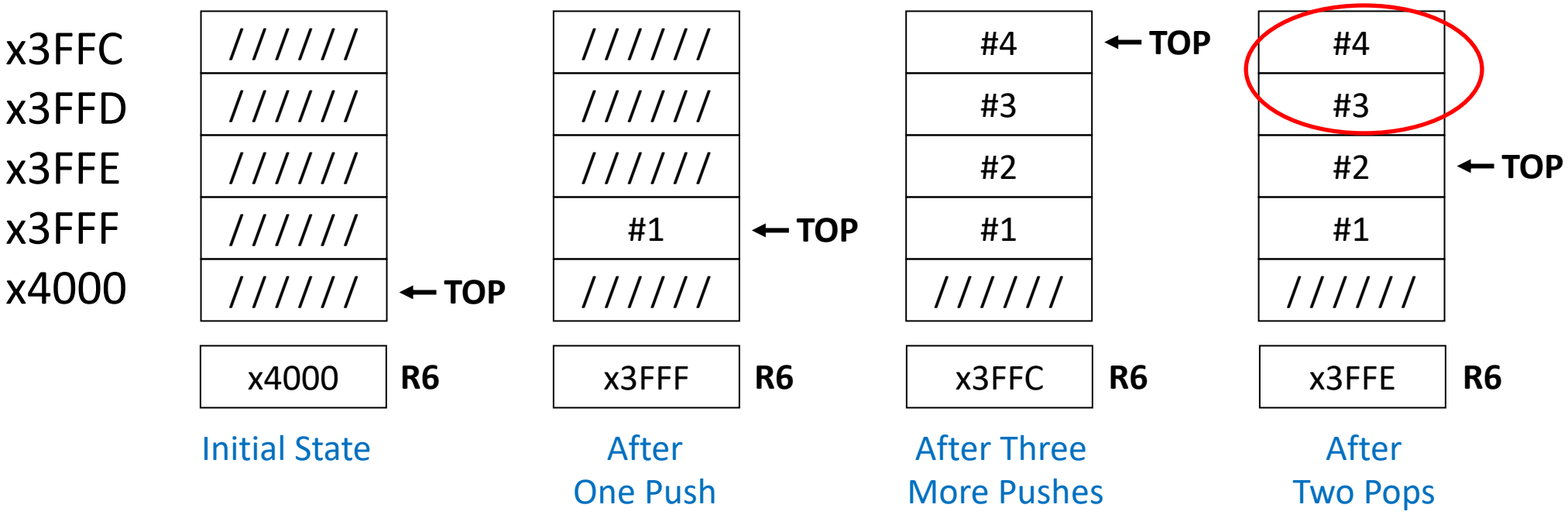


problem?

Stack Implementation 2 – from textbook

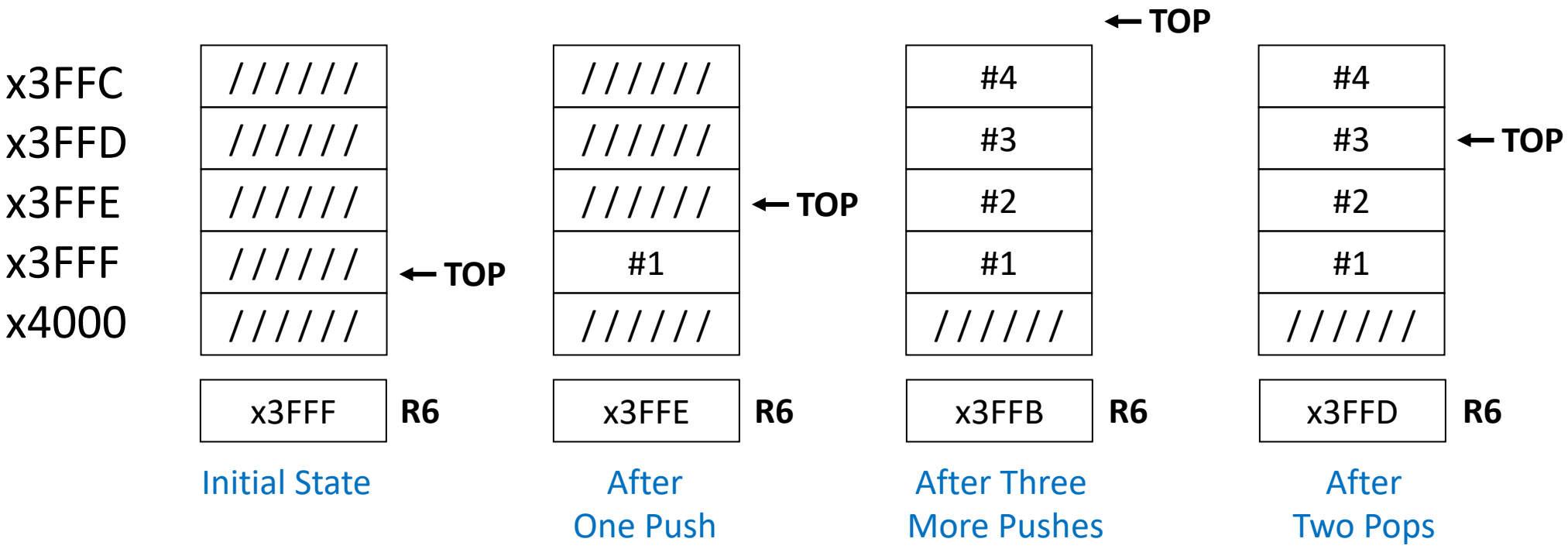
- Data items do NOT move in memory.
- Instead of moving the data, track the top of the stack.

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 closer to x0000

Still Stack? – from MP2



TOS is pointing “Next available spot”

Basic Stack Operations

PUSH

POP

Overflow detection

(Is it full?)

Underflow detection

(Is it empty?)

*When item added, TOS moves closer to x0000.

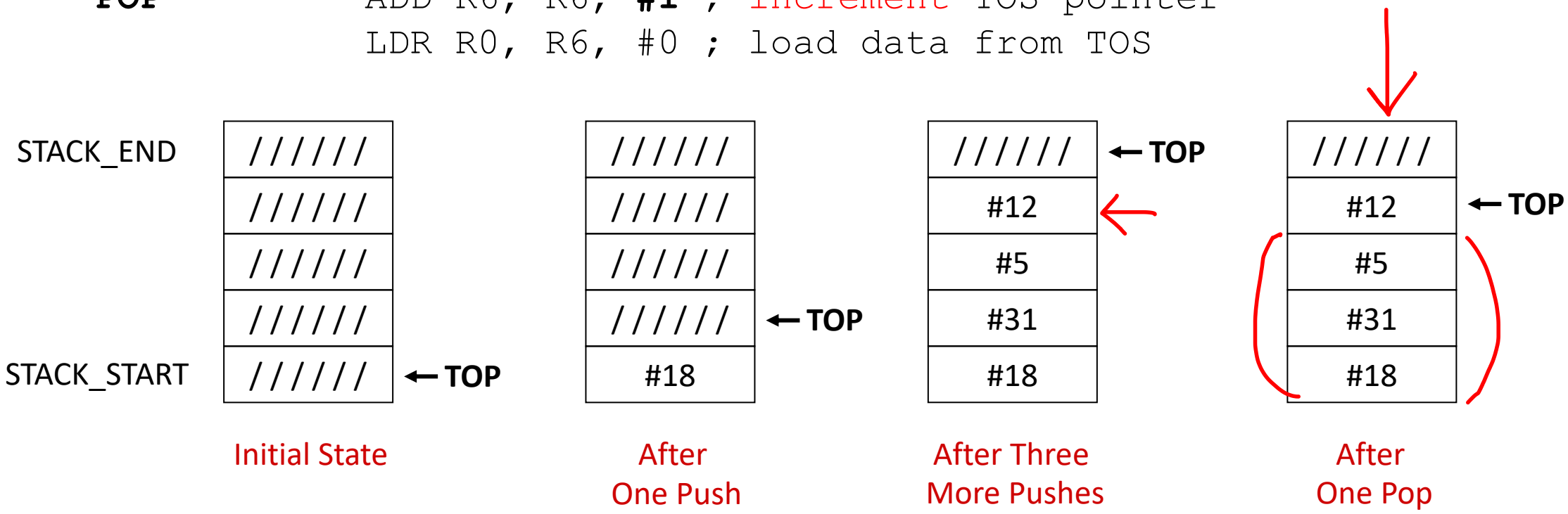
Basic PUSH and POP code

R0: input data

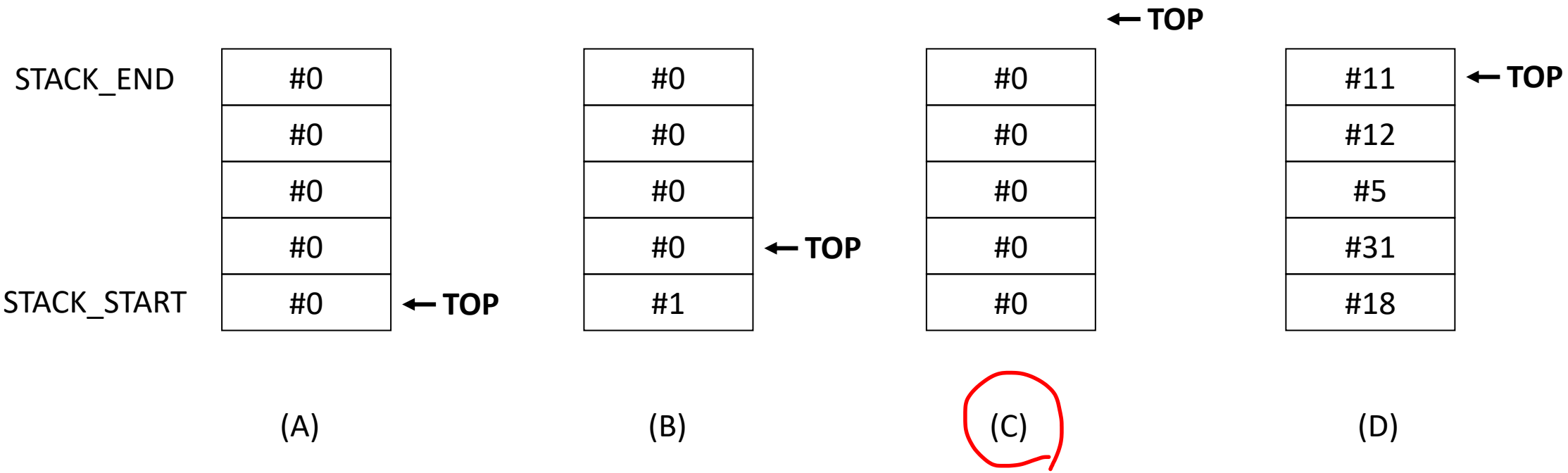
```
PUSH    STR R0, R6, #0 ; store data to TOS
        ADD R6, R6, #-1 ; decrement TOS pointer
```

R0: output data

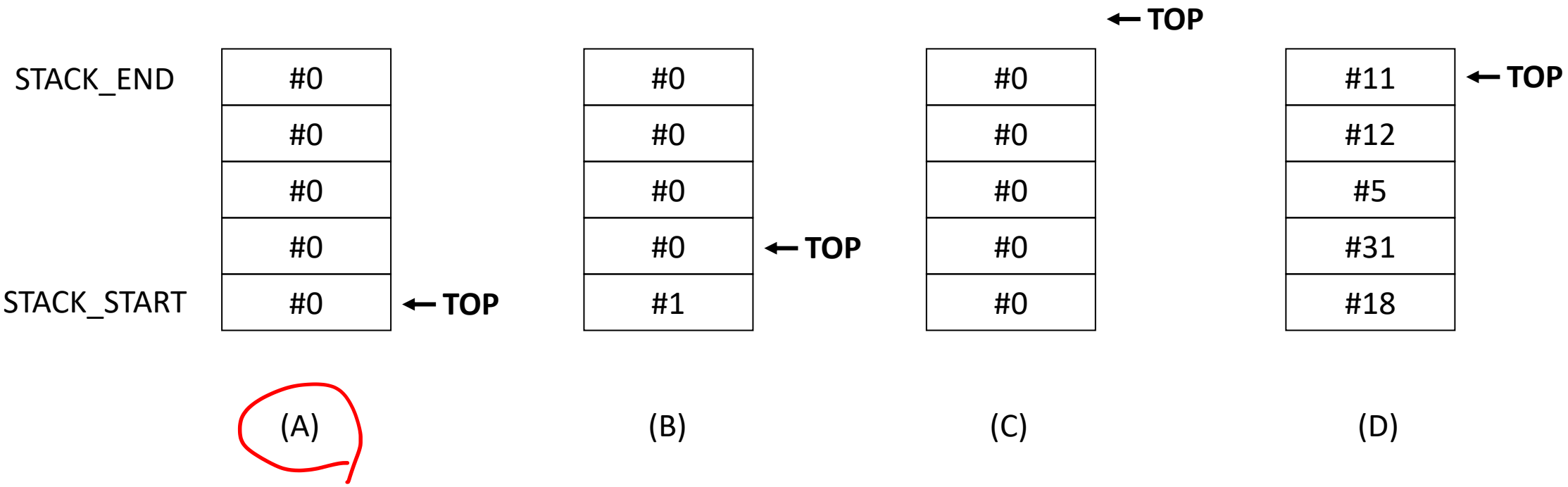
```
POP     ADD R6, R6, #1 ; increment TOS pointer
        LDR R0, R6, #0 ; load data from TOS
```



Q. Which of the following stack is Full?
(TOS is pointing the next available spot)

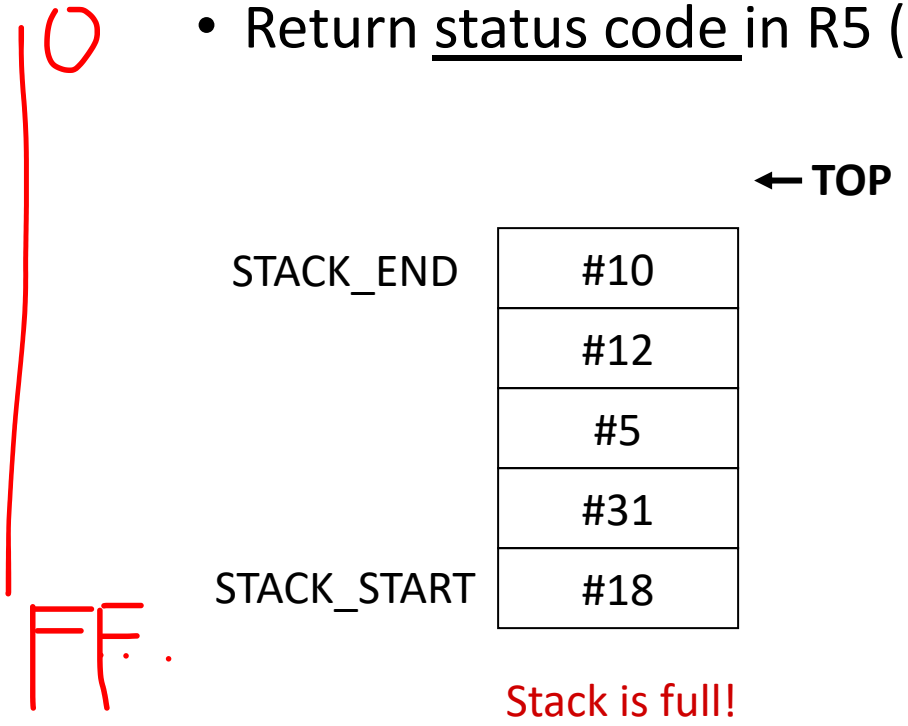


Q. Which of the following stack is Empty? (TOS is pointing the next available spot)



PUSH with Overflow detection

- If we try to **push** too many items onto the stack, an **overflow** condition occurs.
  - Check overflow before adding data.
  - Return status code in R5 (0: success, 1: overflow)

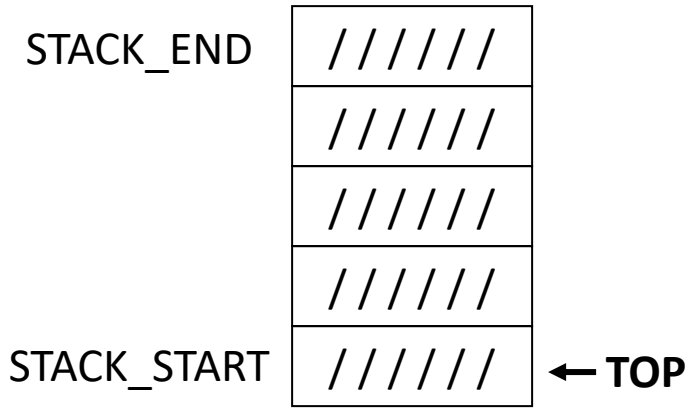


Q. Stack is full, if TOS is

A. STACK_END + 1
B. STACK_END – 1
C. STACK_END
D. STACK_START

POP with Underflow detection

- If we try to **pop** too many items onto the stack, an **underflow** condition occurs.
  - Check underflow before removing data.
  - Return status code in R5 (0: success, 1: underflow)



Stack is empty!

Q. Stack is empty, if TOS is

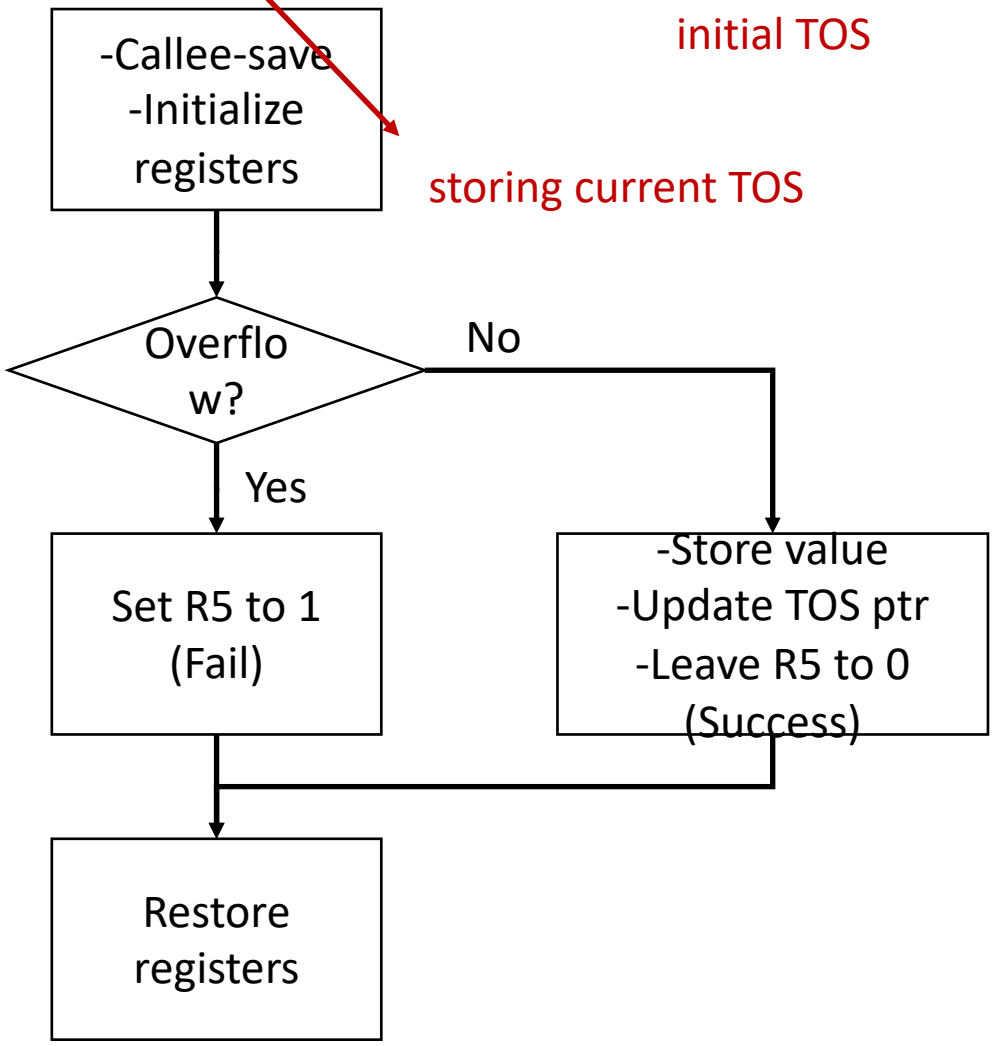
A. STACK_START + 1
B. STACK_START – 1
C. STACK_END
D. STACK_START

PUSH Subroutine

```
;PUSH subroutine
;IN: R0 (value)
;OUT: R5(0-success, 1-fail)
;R3: STACK_END
;R6: STACK_TOP
PUSH
;callee-save & initialize registers
ST     R3, SAVE_R3
ST     R6, SAVE_R6
LD     R3, STACK_END
LD     R6, STACK_TOP
AND    R5, R5, #0

;overflow?
;Check if STACK_TOP = STACK_END - 1
;Or check if STACK_TOP - (STACK_END - 1) = 0
ADD    R3, R3, #-1   ; STACK_END - 1
NOT    R3, R3
ADD    R3, R3, #1    ;- (STACK_END - 1) bc 2'scomp
ADD    R3, R6, R3    ; STACK_TOP - (STACK_END - 1)
BRz    OVERFLOW
```

```
STACK_END     .FILL   x3FFD
STACK_START   .FILL   x4000
STACK_TOP     .FILL   x4000
```

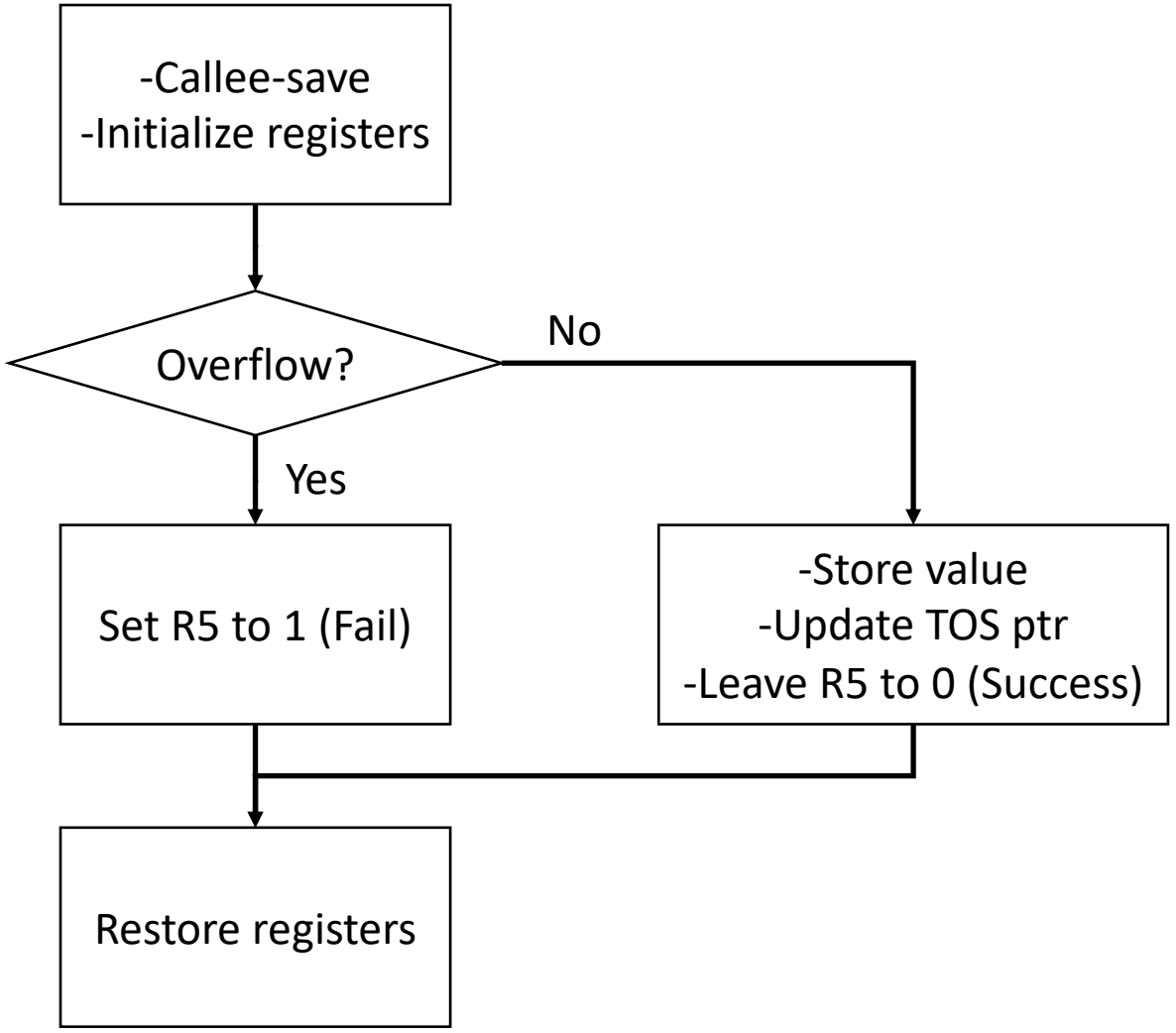
initial TOS



PUSH Subroutine

```
STACK_END       .FILL   x3FFD
STACK_START     .FILL   x4000
STACK_TOP       .FILL   x4000
```

```
;it is NOT overflow
STR     R0, R6, #0
ADD     R6, R6, #-1
ST      R6, STACK_TOP
BR      DONE_PUSH

;it is overflow
OVERFLOW
ADD     R5, R5, #1


;restore registers
DONE_PUSH
LD      R3, SAVE_R3
LD      R6, SAVE_R6
RET
```



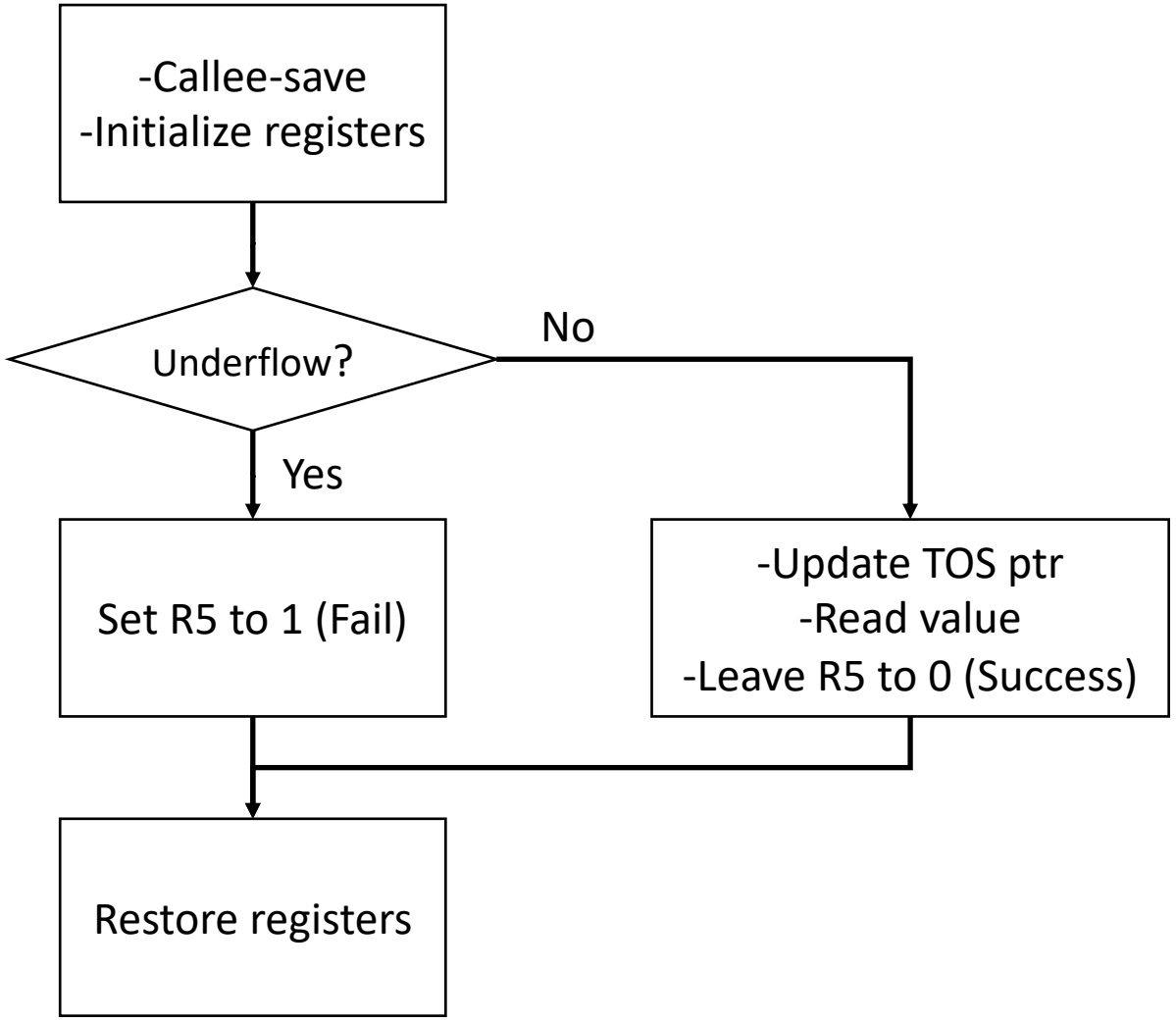
How to use PUSH?

1. Load the data in R0
2. JSR PUSH
3. Check R5

POP Subroutine

```
;POP subroutine
;IN: none
;OUT: R0 (value)
;OUT: R5 (0: success, 1: fail)
;R3: STACK_START
;R6: STACK_TOP
POP
;callee-save & initialize registers
 ST      R3,SaveR3
 ST      R6,SaveR6
 LD      R3,STACK_START
 LD      R6,STACK_TOP
 AND     R5,R5,#0
;underflow?
;Check if STACK_TOP = STACK_START
;Or check if STACK_TOP - STACK_START = 0
 NOT     R3,R3
 ADD     R3,R3,#1
 ADD     R3,R3,R6
 BRz     UNDERFLOW
```

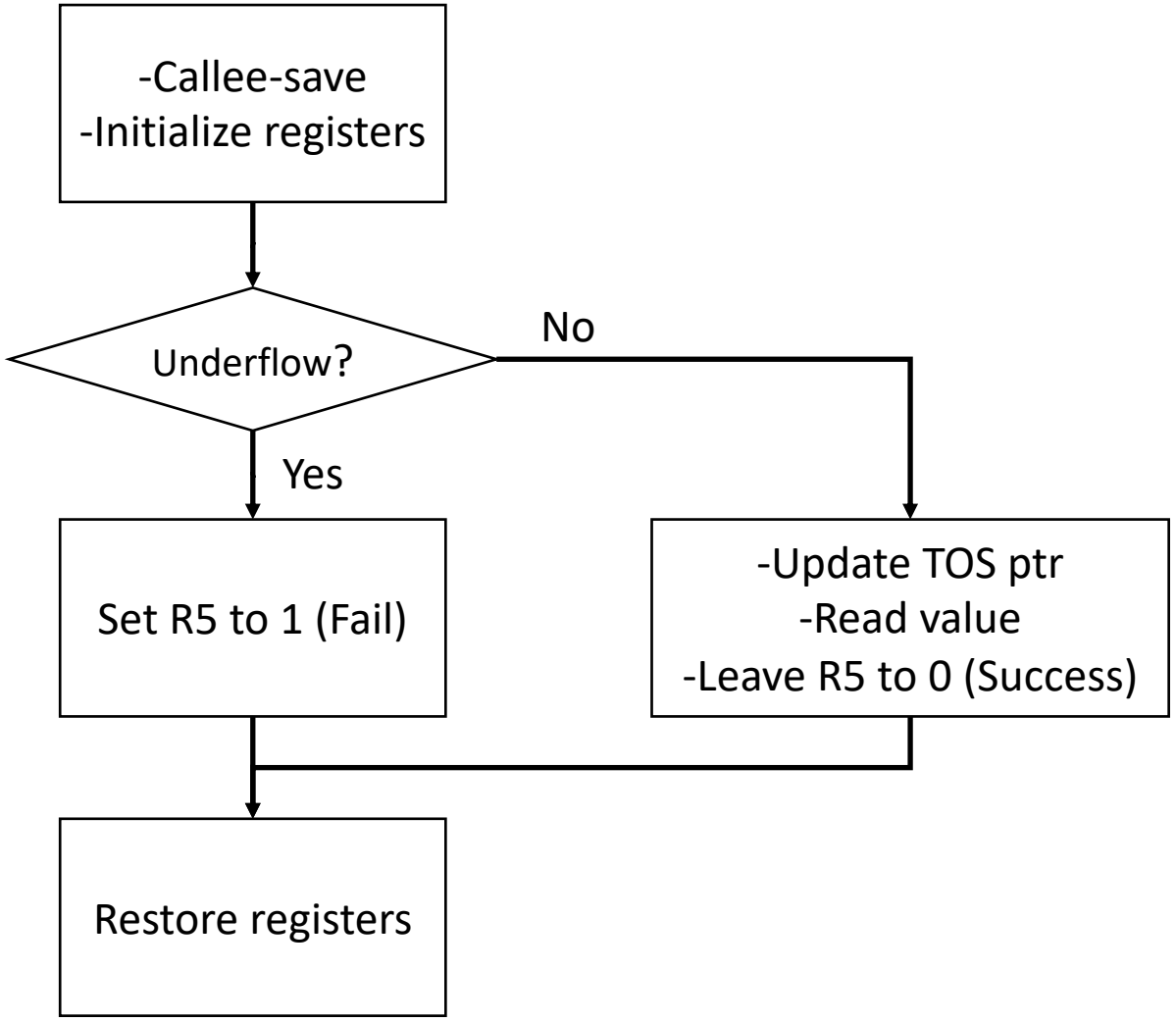
```
STACK_END       .FILL   x3FFD
STACK_START     .FILL   x4000
STACK_TOP       .FILL   x4000
```



POP Subroutine

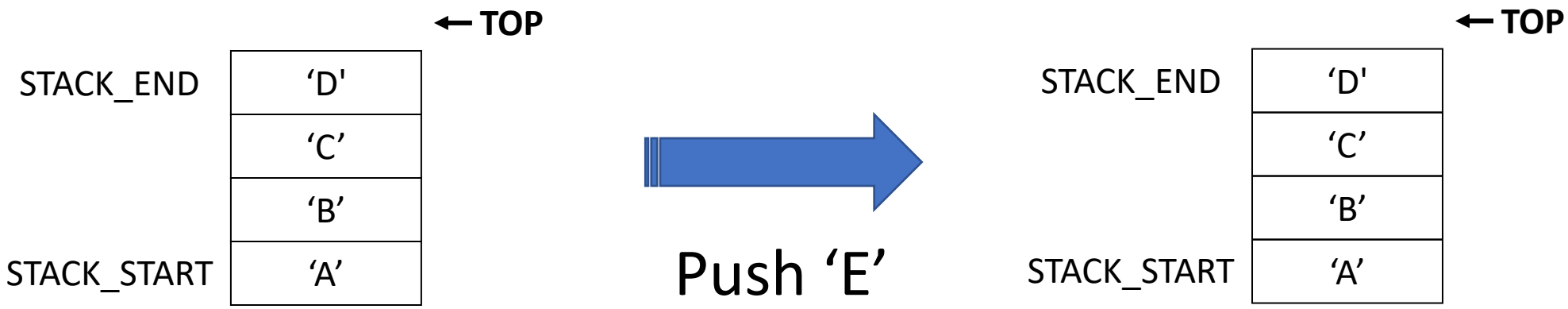
```
;it is NOT underflow
 ADD     R6,R6,#1
 LDR     R0,R6,#0
 ST      R6,STACK_TOP
 BRnzp   DONE_POP

;it is underflow
UNDERFLOW
  ADD     R5,R5,#1

;restore registers
DONE_POP
 LD      R3,SaveR3
 LD      R6,SaveR6
 RET
```

```
STACK_END       .FILL   x3FFD
STACK_START     .FILL   x4000
STACK_TOP       .FILL   x4000
```

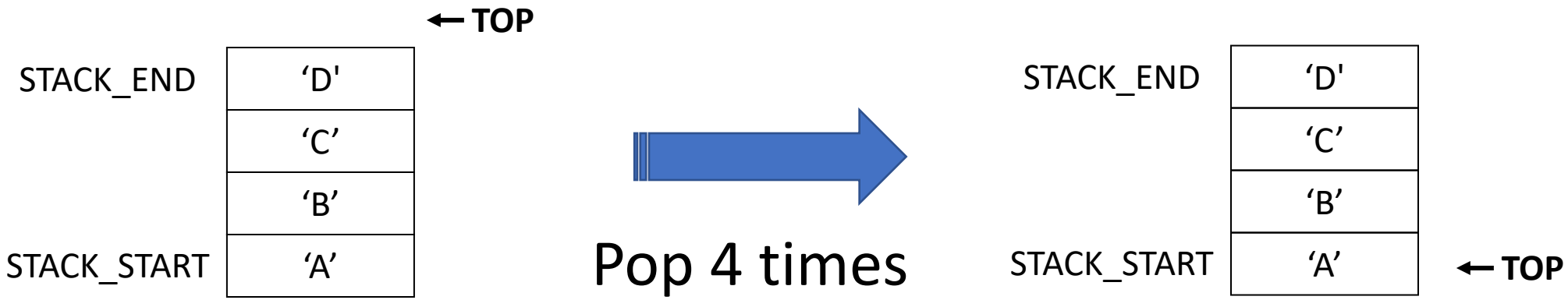


How to use POP?

1. JSR POP
2. Check R5
3. (if R5 is 0), use R0


TOP
STACK_END
‘D'
‘C’
‘B’
STACK_START
‘A’
Push ‘E’
TOP
STACK_END
‘D'
‘C’
‘B’
STACK_START
‘A’

R5: 0→1



The memory data are NOT removed!!

But, the stack is EMPTY.

```
.ORIG x3000

LOOP
        GETC
        OUT
        JSR PUSH
        ADD   R5, R5, #0
        BRp   PUSH_FAIL
        BRnzp LOOP

        PUSH_FAIL
        HALT

; PUSH subroutine omitted
```

See the full code in https://github.com/tmoon-illinois/ece220_sp24