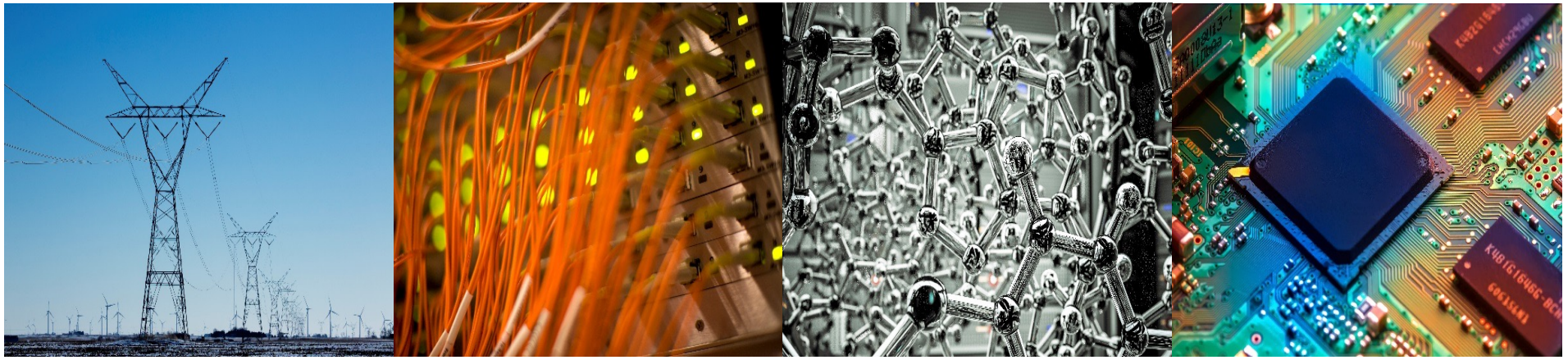
ECE 220 Computer Systems & Programming

Lecture 3 – Stack

January 27, 2026



- **MP1 is due on Thursday at 10pm CT**
- **Mock quiz (extra-credit) should be taken next week (2/3 to 2/5) at CBTF**

ILLINOIS
Electrical & Computer Engineering
GRAINGER COLLEGE OF ENGINEERING

Lecture 2 Review: Nested Subroutines

```
; Nested Subroutines Example
.ORIG x3000
AND R1, R1, #0    ;init R1
AND R2, R2, #0    ;init R2
ADD R1, R1, #5    ;set R1=5
ADD R2, R2, #2    ;set R2=2
;call SUBTR to calculate R1-R2
JSR SUBTR
;copy result to R6
ADD R6, R0, #0
HALT
```

```
;SUBTR - computes R1-R2
;IN: R1, R2
;OUT: R0 <- R1 - R2
SUBTR
    ADD R3, R2, #0
    JSR NEGATE      ;R3 = -R2
    ADD R0, R1, R3 ;R0 = R1-R2
    RET


;NEGATE - negates the input
;IN: R3
;OUT: R3
NEGATE
    NOT R3, R3
    ADD R3, R3, #1
    RET

.END
```

- Would this program work?

Stack – An Abstract Data Type

A **LIFO (last-in first-out)** storage structure

- The **first** thing you put in is the _____ thing you take out
- The **last** thing you put in is the _____ thing you take out

This **means of access** is what defines a stack, not the specific implementation.

Two main operations:

Push: add an item to the stack

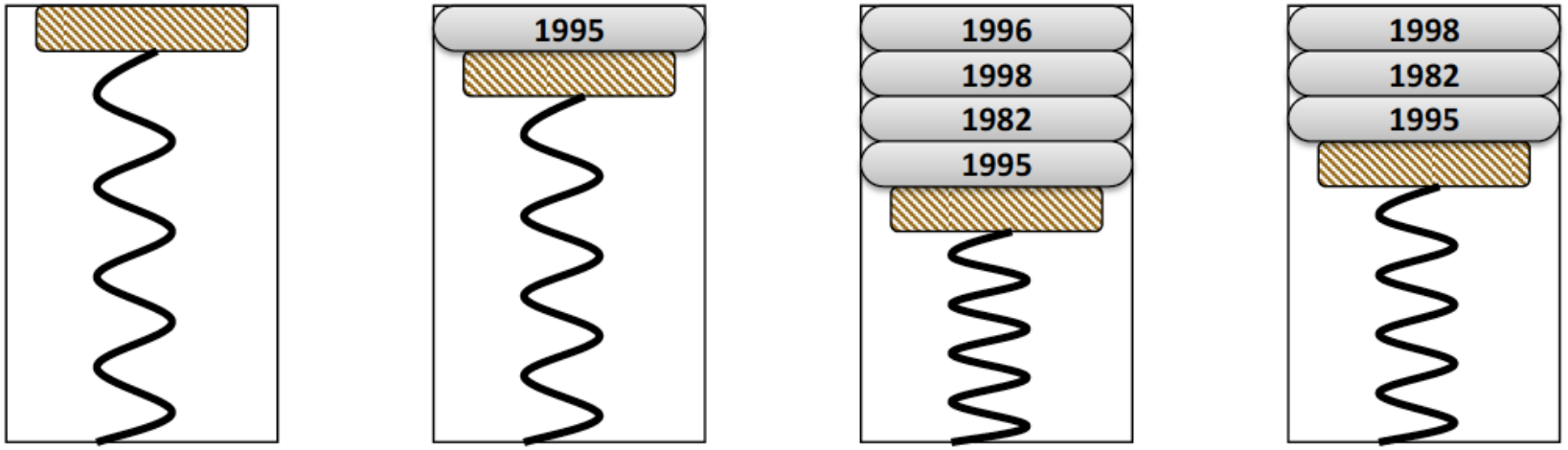
Pop: remove an item from the stack

IsFull: check whether the stack is full (_____)

IsEmpty: check whether the stack is empty (_____)

Coin Holder Example

First coin in is the last coin out

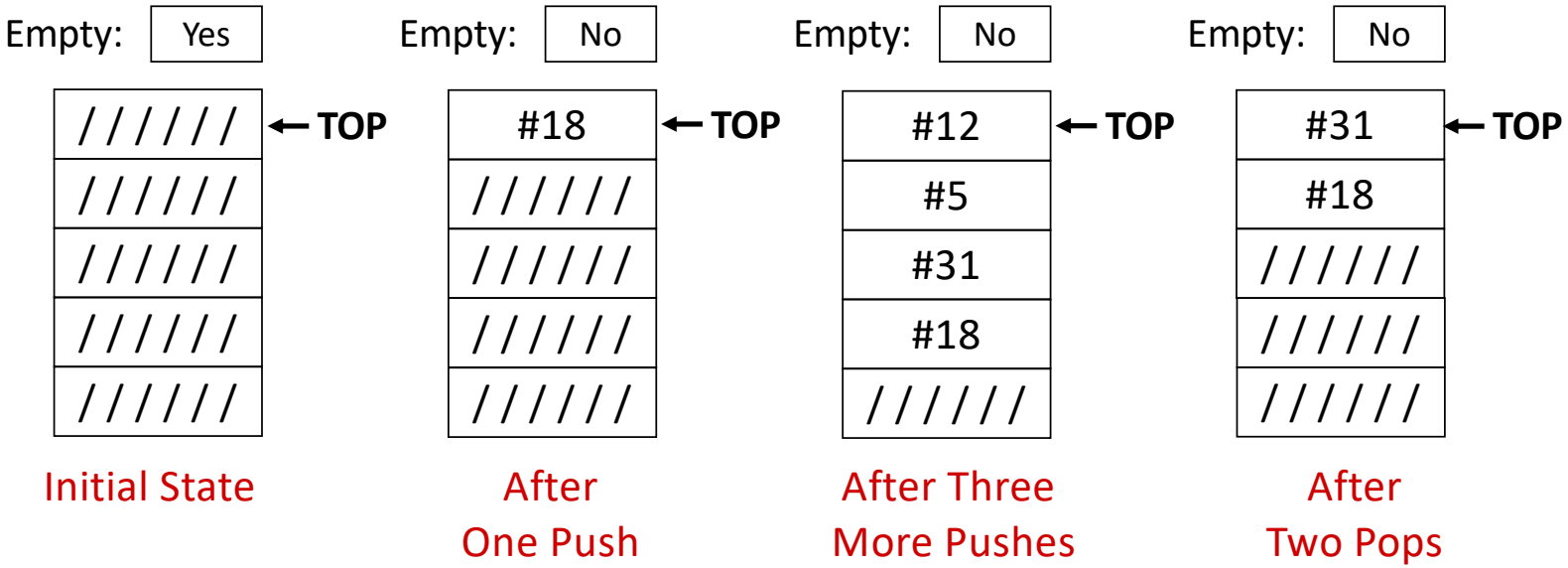


Initial State | After One Push | After Three More Pushes | After One Pop

- Can you think of anything else that is implemented using a stack ADT?

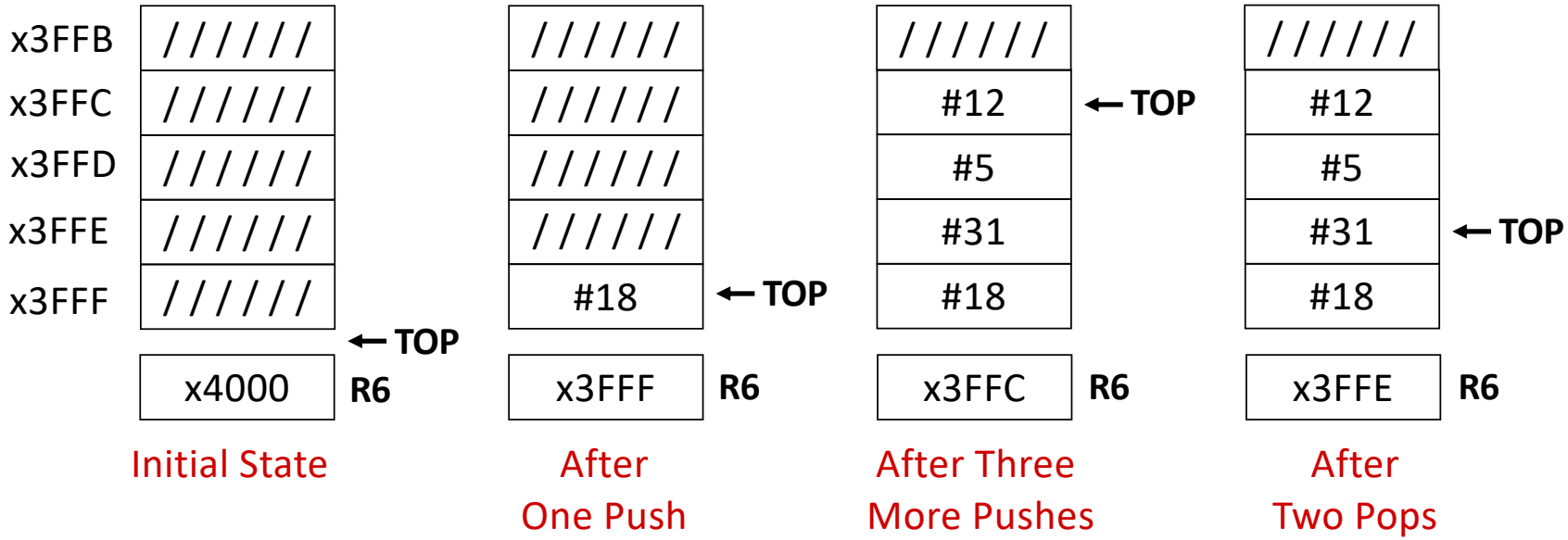
Implementation in Hardware

Data items move, top of stack is fixed



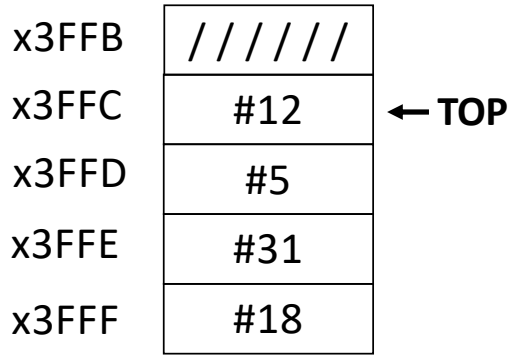
Implementation in Memory

Data items don't move, just our idea about where the top of the stack is



- By convention, **R6** holds the Top of Stack (TOS) pointer

Basic Push and Pop Code



Using Software Implementation of Stack

- **Push**

```
ADD R6, R6, #-1 ;decrement stack ptr
STR R0, R6, #0  ;store data (to Top of Stack)
```

- **Pop**

```
LDR R0, R6, #0  ;load data from stack ptr
ADD R6, R6, #1  ;increment stack ptr
```

Implement PUSH Subroutine

x3FF0	
x3FF1	
x3FF2	
x3FF3	
x3FF4	
x3FF5	
x3FF6	
x3FF7	
x3FF8	
x3FF9	
x3FFA	
x3FFB	
x3FFC	
x3FFD	
x3FFE	
x3FFF	
x4000	

← STACK_END

← STACK_START

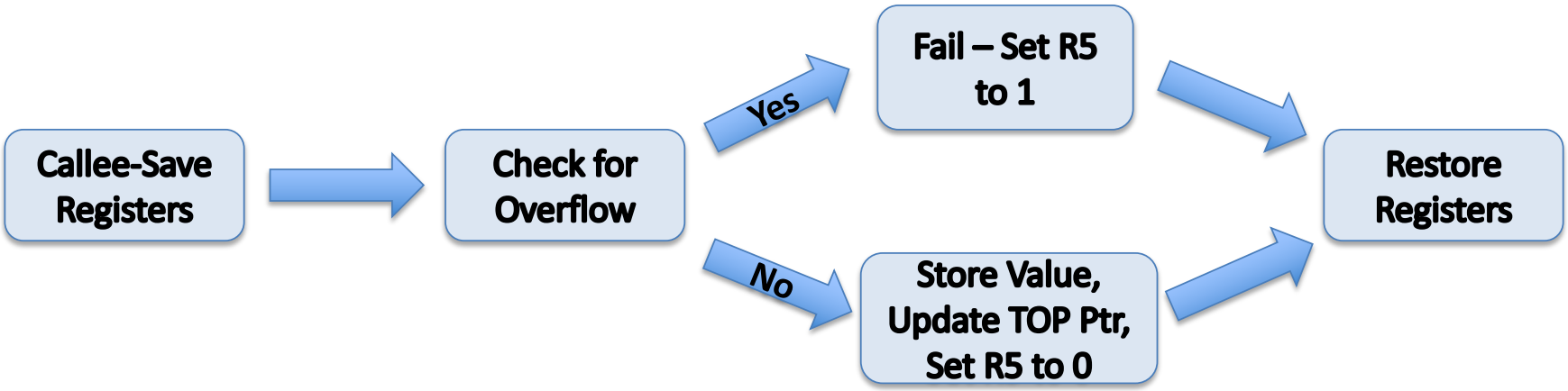
```
.ORIG x3000
…
HALT

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

PUSH_SAVER3   .BLKW #1
PUSH_SAVER6   .BLKW #1

.END
```

← STACK_TOP (next available spot)




```
; PUSH subroutine
; IN: R0 (value)
; OUT: R5 (0 – success, 1 – fail)
; R3: STACK_END
; R6: STACK_TOP
PUSH
; save original values of R3 and R6, init R5 to 0



; load R3 with STACK_END, R6 with STACK_TOP
```

```
; check for overflow (when stack is full: STACK_TOP < STACK_END)




; store value (in R0) to stack, update STACK_TOP




; indicate the overflow condition on return
OVERFLOW

; restore modified registers and return
DONE_PUSH
```