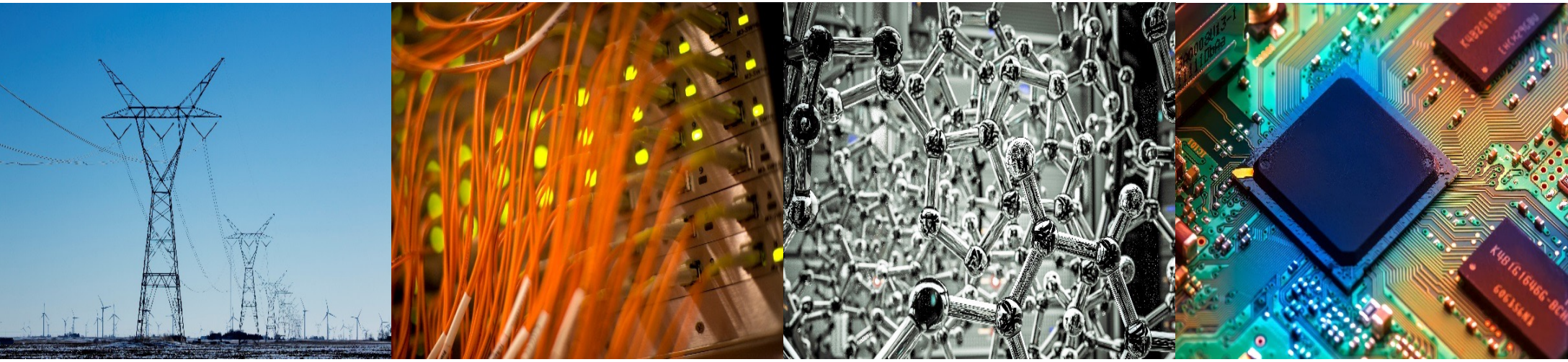
ECE 220 Computer Systems & Programming

Lecture 4 – Programming with Stack



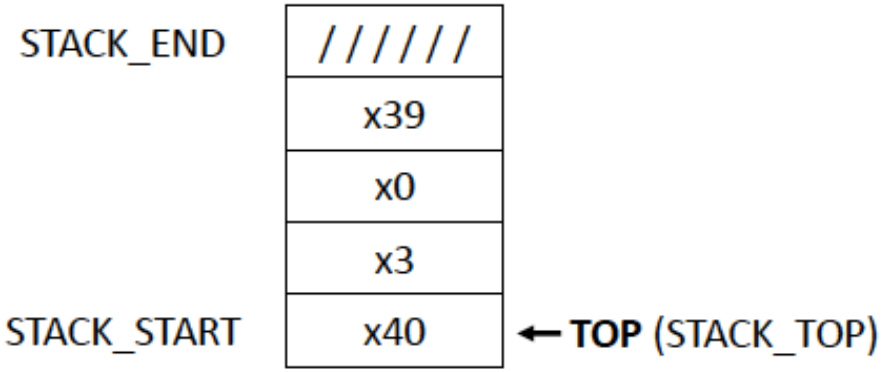
- LC-3 practice is available on PrairieLearn
- Mock quiz should be taken next week @ CBTF
- Quiz1 (LC-3 programming) is available for reservation

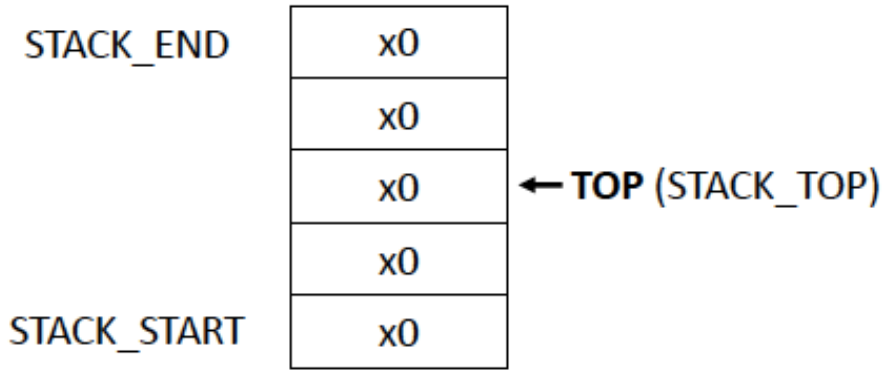
ILLINOIS
Electrical & Computer Engineering
GRAINGER COLLEGE OF ENGINEERING

Lecture 3 Review: Stack

- Order of Access
- Two Main Operations
- Overflow vs. Underflow
- Hardware vs. Software Implementation
- Top of Stack Pointer (stack pointer)

➢ In the following two figures, which stack is empty? (Note: STACK_TOP points to the next available spot.)



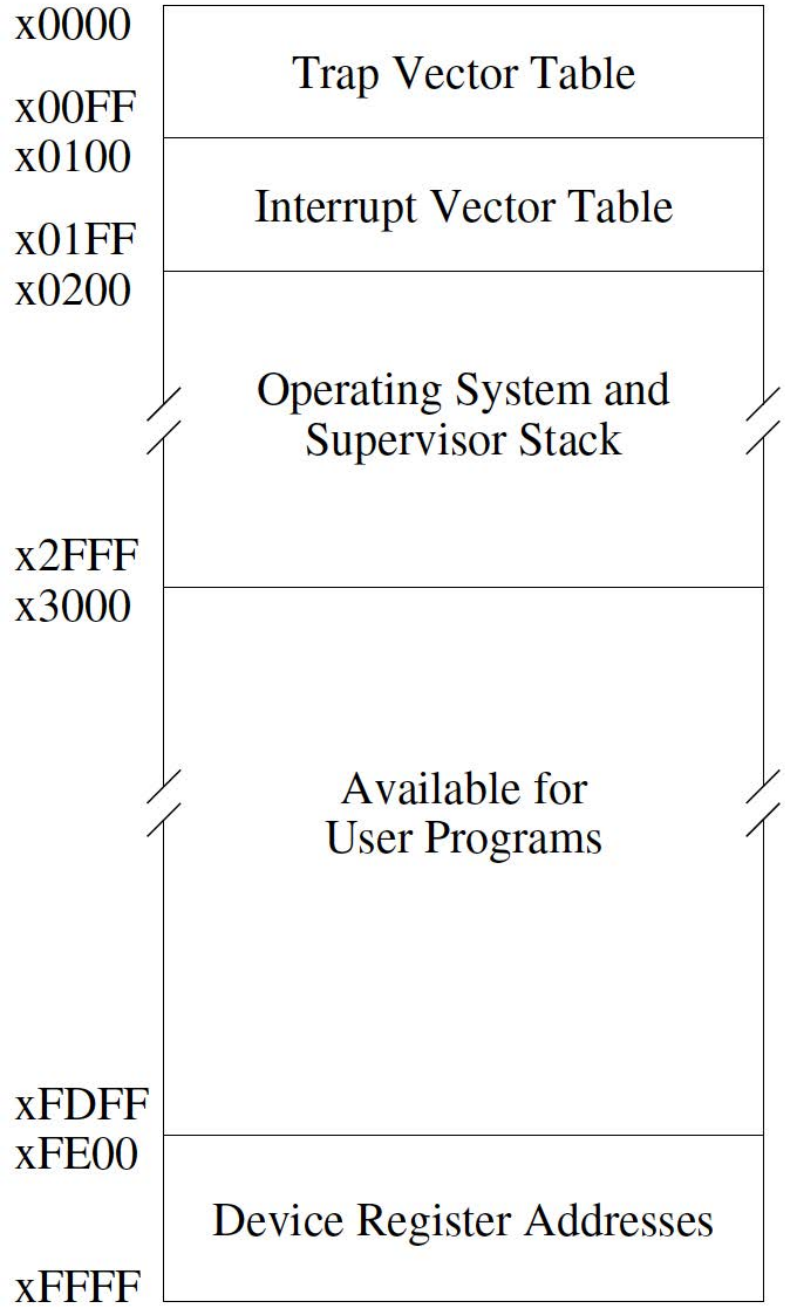


Run-Time Stack

- Information of an invoked function (subroutine) is stored in a memory template called the ***activation record*** or ***stack frame***.

- Functions' activation records are pushed onto the Run-Time Stack in the order they are invoked.

❖ **Supervisor Stack** is different from Run-Time Stack (more details at the end of the semester).



Balanced Parentheses Check Using a Stack

Examples of balanced parentheses:

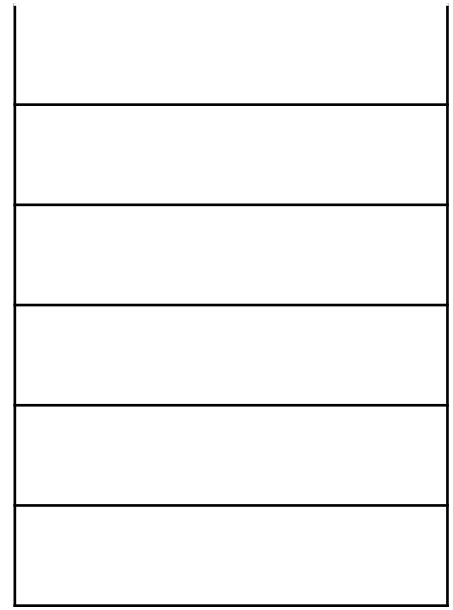
(( ) ( ) ( ) ( )) (((( )))) (( ) ((( )) ( )))

Examples of unbalanced parentheses:

((((((( )) ( ))) (( ) ( ) ((

Open parenthesis ‘ (’ – _____ to the Stack

Close parenthesis ‘) ’ – _____ from the stack



Assuming the expression would fit into the stack, unbalanced expression can be found under two situations:

1. At the end of the expression –
2. While entering expression –

Palindrome Check Using a Stack

A word, phrase, number or other sequence of characters which **reads the same forward or backward**.

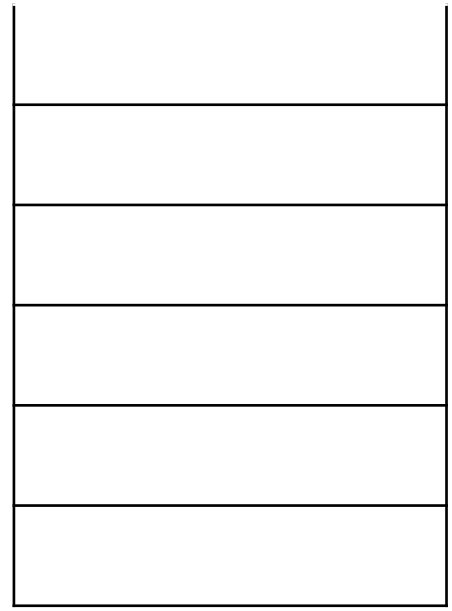
- Madam
- Kayak
- Was it a car or a cat I saw
- 123456654321

➢ How can we perform a palindrome check using a stack?

Postfix Expression (input is single digit operand)

Infix	Postfix
(3+4)-5	34+5-
2^(8-4)	
7+(9-6)/3	
	512+4*+3-

Note: ‘12-’ is 1-2 not 2-1



- Are these inputs valid postfix expressions? How would your program know?
- 46*-
- 13+57

Arithmetic Using a Stack

Compute (A+B)*(C+D) and store the result in R0

```
; Implementation using registers
LD R0, A
LD R1, B
ADD  R1, R0, R1
LD R2, C
LD R3, D
ADD  R3, R2, R3
JSR  MULT
HALT
```

** MULT subroutine*

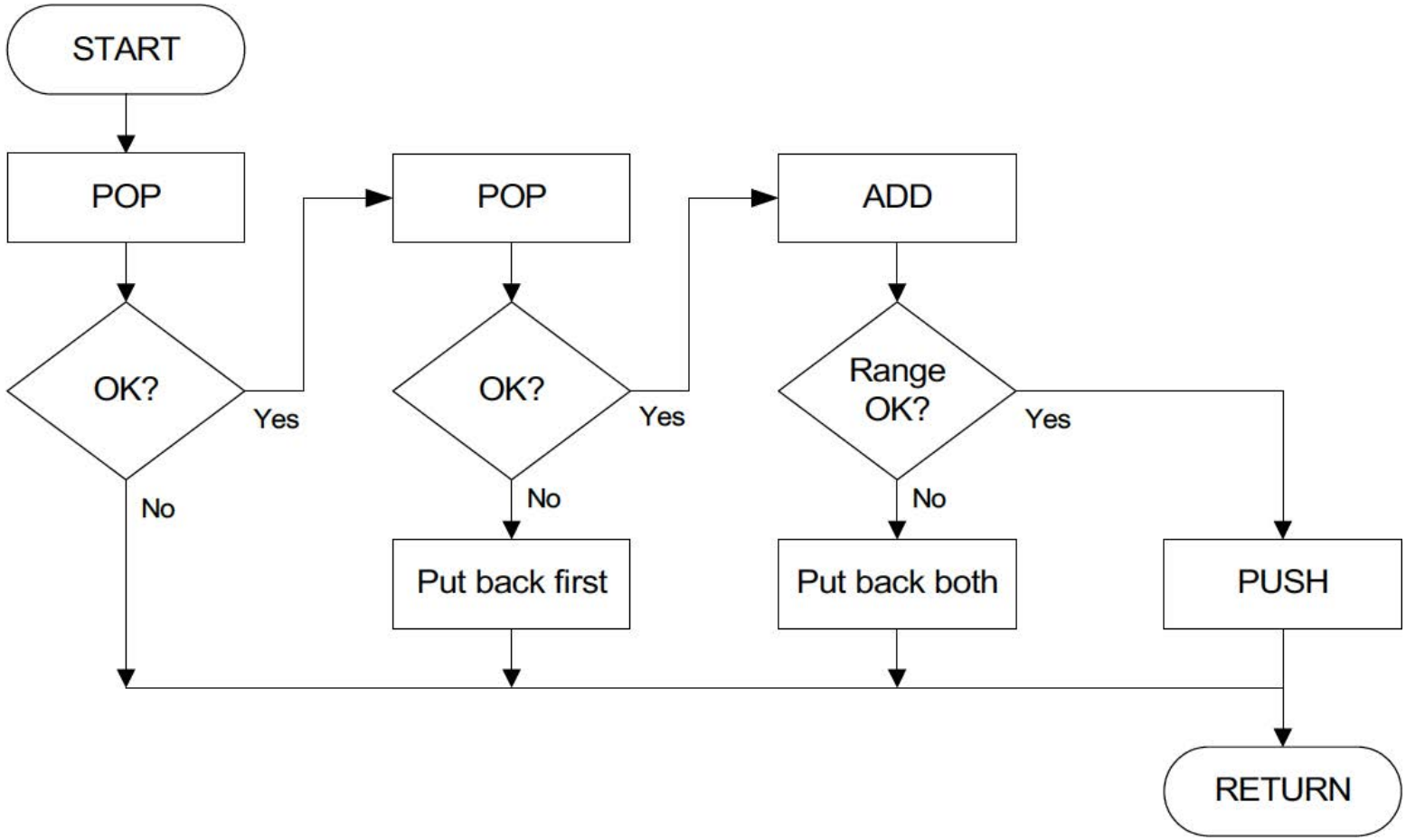
(Input: R1, R3; Output: R0)

```
; Implementation using a Stack
; PUSH, POP, ADD & MULT subroutines are given
LD R0, A
JSR PUSH
```

***PUSH: from R0 to stack; POP: from stack to R0**
***ADD: POP 2 numbers, compute and then PUSH result back**
***MULT: POP 2 numbers, compute and then PUSH result back**

Arithmetic Using a Stack

Implement an ADD subroutine that pops two numbers from a stack and perform the add operation (see flowchart below).



Implement ADD Subroutine

- **R6** should be used as stack pointer (points to the **next available spot** on the stack)
- Assume **PUSH**, **POP** and **CHECK_RANGE** subroutines are given & callee-saved

```
; PUSH
; Input: R0 (value to be stored on stack)
; Output: R5 (0 – success, 1 – failure)

; POP
; Output: R0 (value to be loaded from stack)
; Output: R5 (0 – success, 1 – failure)

; CHECK_RANGE: return 0 if value is within -100 to 100 decimal,
; otherwise return 1
; Input: R0 (value to be checked)
; Output: R5 (0 – success, 1 – failure)
```

➢ **What do we need to consider when implementing the ADD subroutine?**

```
; ADD subroutine – pop two numbers from stack,
; perform '+' operation and then push result back to the stack
; Output: R5 (0 – success, 1 – failure)

; save registers









; Initialize R5

; first pop


; check return value of first pop, go to EXIT if failed (R5 = 1)
```

```
; second pop

; check result of second pop, go to RESTORE_1 if it failed


; add two numbers

; check range of sum, go to RESTORE_2 if it failed


; everything is good, push sum to stack
```

```
RESTORE_1
; put back first number




RESTORE_2
; put back both numbers




EXIT
; restore registers




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