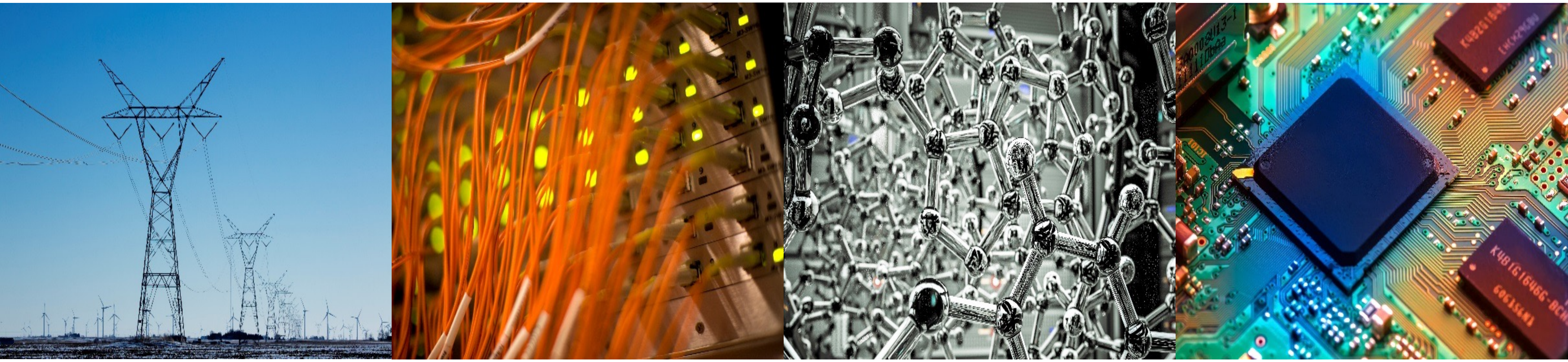
ECE 220 Computer Systems & Programming

Lecture 5 – Programming with Stack

September 6, 2022



- Mock quiz this week
- Quiz 1 next week (LC-3 Concepts)

ILLINOIS
Electrical & Computer Engineering
GRAINGER COLLEGE OF ENGINEERING

Lecture 4 Review: Stack

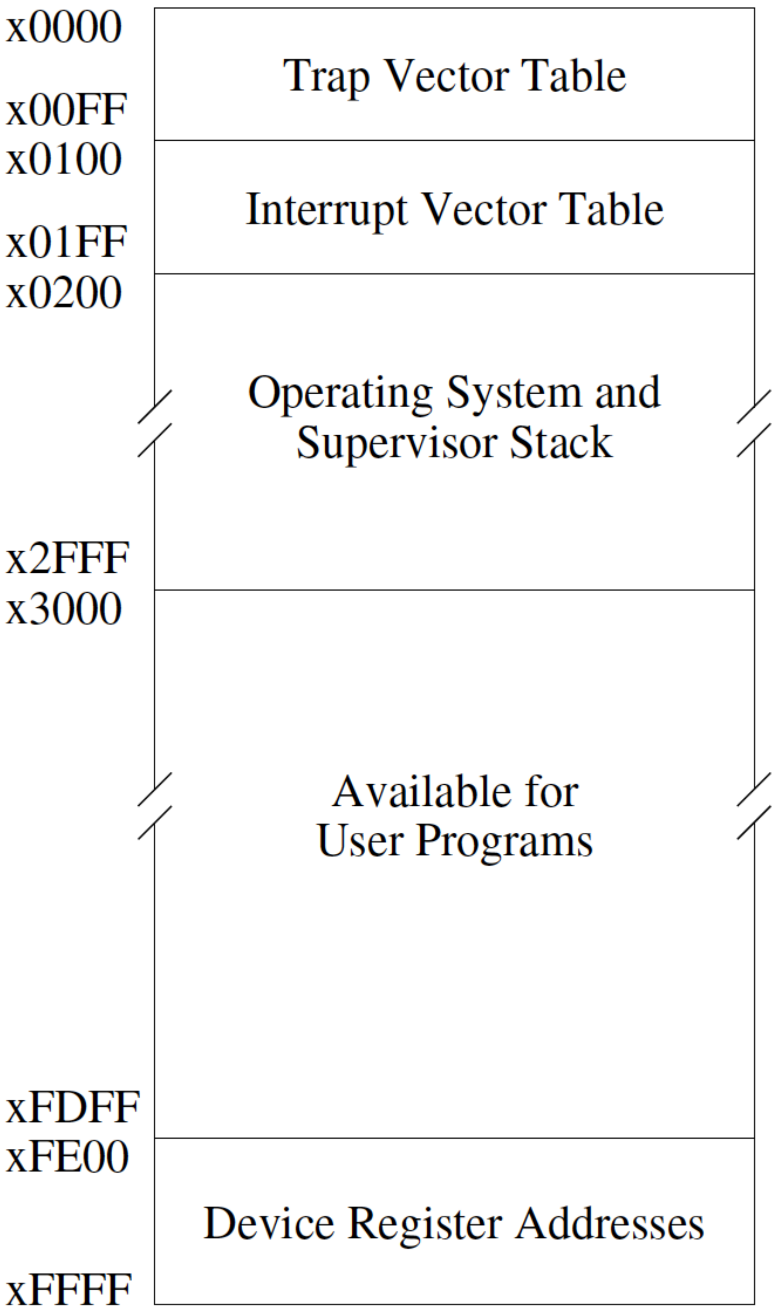
- Order of Access
- Two Main Operations
- Overflow vs. Underflow
- Hardware vs. Software Implementation
- Top of Stack Pointer (stack pointer)

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.



Check whether parentheses in an expression are balanced

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. While entering expression –
2. At the end of the 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
- aibophobia

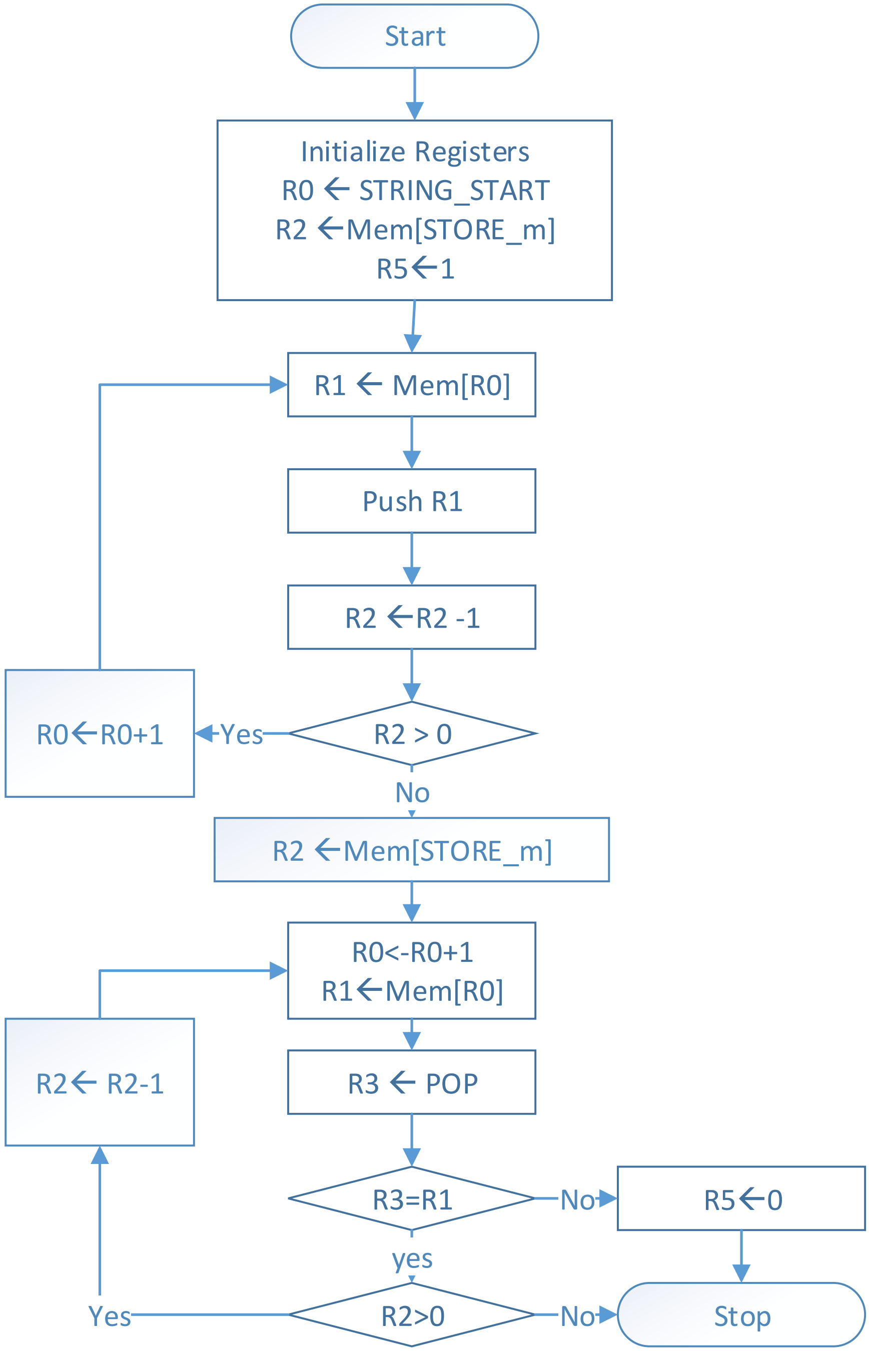
➢ How can we perform a palindrome check using a stack?

Checking palindromes

Problem:

- INPUTS: String starting from memory location STRING_START
- Length of the string is 2m, m is stored in memory location STORE_m.
- OUTPUT: R5 = 1 if palindrome; 0 otherwise
- Assume that the string is NUL terminated; no spaces and punctuations

- R0: address of character being read
- R1: current character being read
- R3: 'mirror' character
- R2: (m - #characters read)



Postfix Expression (input is single digit operand)

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

Note: ‘34-’ is 3-4 not 4-3

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

Arithmetic Using a Stack

- Compute (A+B)*(C+D) and store 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*

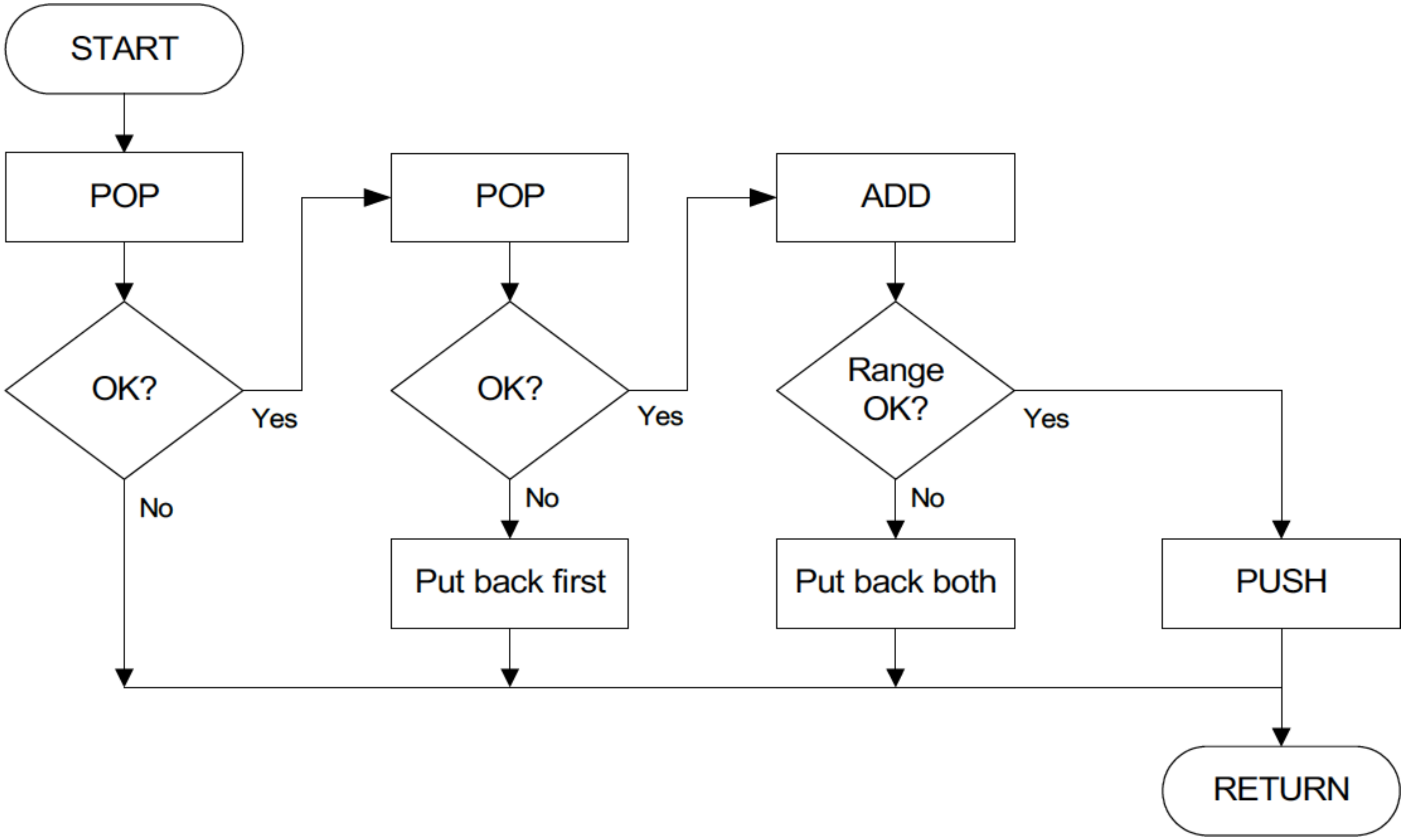
(Input: R1, R3; Output: R0)

```
; 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

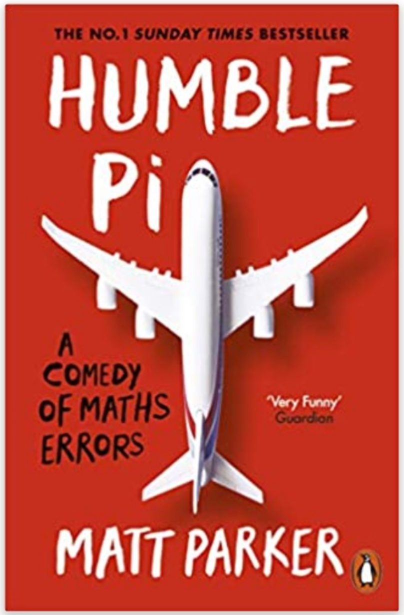
Arithmetic Using a Stack

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



Why Range Check

- Overflow!
- Sept 14 2004, Los Angeles Air Traffic Control lost voice radio with 800 aircraft over SoCal for 3 hours, causing widespread panic, because of a counter that had to be reset every 49 days, 17 hours and 2 mins
- Software counter in generators of Boeing 787 will overflow after 248 days, 13 hours, 13 minutes, 56.47 seconds (before Boeing fixed it), and all GCU may go into failsafe mode after 248 days of continuous operation resulting in loss of AC power regardless of flight phase
- What are these mysterious numbers?
- 4,294,967,295
- 2,147,483,647



Implement ADD Subroutine

- R6 is 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 you need to pay attention 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 and load STACK_TOP to R6









; 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
; update STACK_TOP and restore registers

RET
```