ECE 220 Computer Systems & Programming

Lecture 4: Introduction to Stack Data Structures

September 5, 2019

ECE ILLINOIS

ILLINOIS

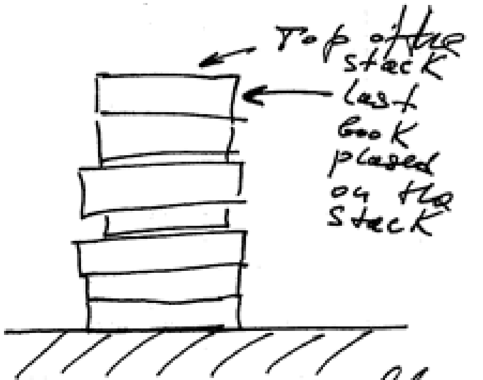
Outline

- What is a stack?
- How to implement a stack?
- POP and PUSH Subroutines in LC-3
- Overflow and Underflow in stack

- Chapter 10 in textbook

The Stack Abstraction

- Stack is an abstract data structure
- Stack of books example:
  - A new book always goes on top of the stack
  - We can only remove a book from the top of the stack

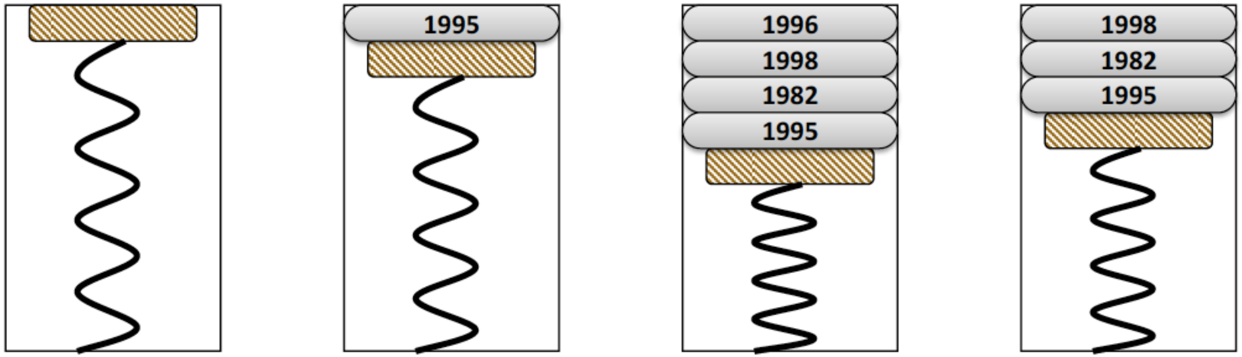


Palindromes:

- Examples of palindromes
  - Madam
  - Kayak
  - Was it a car or a cat I saw?
  - Aibohphobia
- How can we test for palindromes?

Coin Holder Example

- First coin in is the last coin out



Initial State

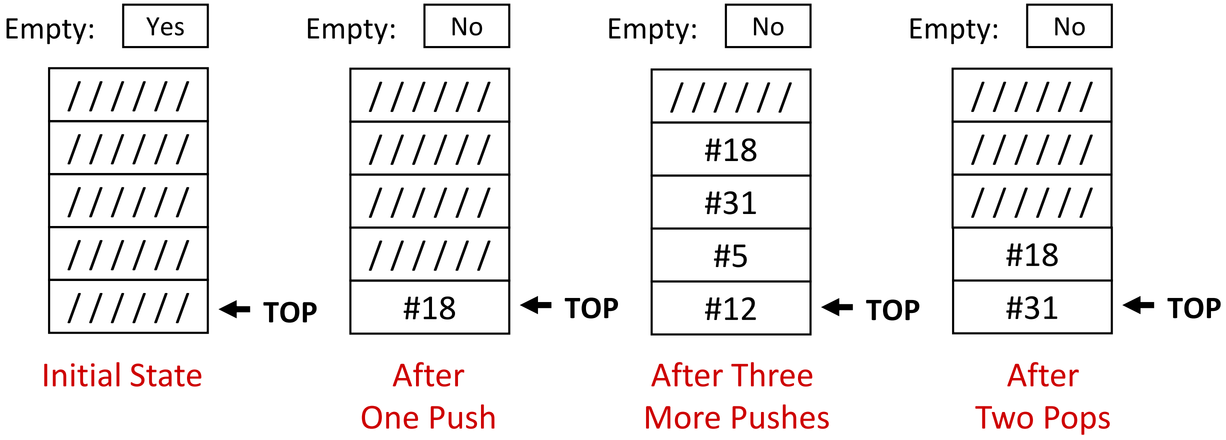
After
One Push

After Three
More Pushes

After
One Pop

A Hardware Implementation

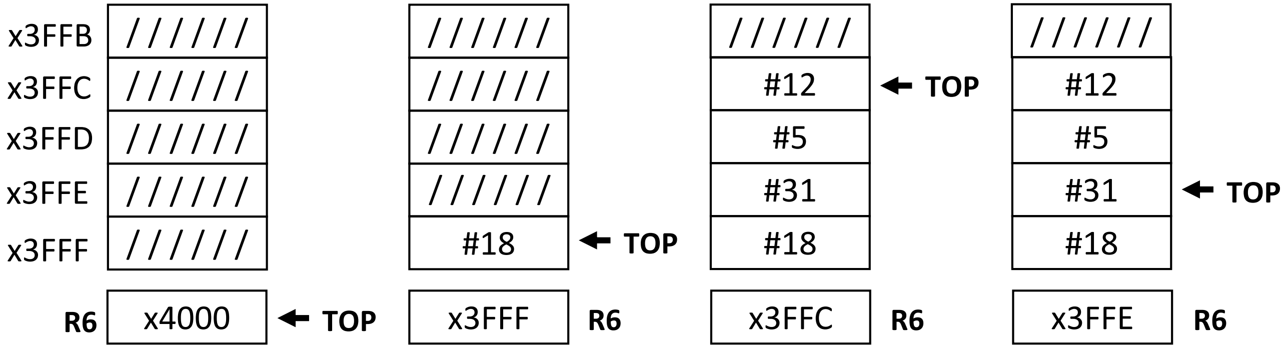
- **Data items move in memory, top of stack is fixed**



Initial State	After One Push	After Three More Pushes	After Two Pops
Empty: Yes	Empty: No	Empty: No	Empty: No
//////	//////	//////	//////
//////	//////	#18	//////
//////	//////	#31	//////
//////	//////	#5	#18
////// ← TOP	#18 ← TOP	#12 ← TOP	#31 ← TOP

A Software Implementation

- **Data items don’t move in memory, just our idea about where the top of the stack is.**



	Initial State	After One Push	After Three More Pushes	After Two Pops
x3FFB	//////	//////	//////	//////
x3FFC	//////	//////	#12 ← TOP	#12
x3FFD	//////	//////	#5	#5
x3FFE	//////	//////	#31	#31 ← TOP
x3FFF	//////	#18 ← TOP	#18	#18
R6	x4000 ← TOP	x3FFF	x3FFC	x3FFE

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

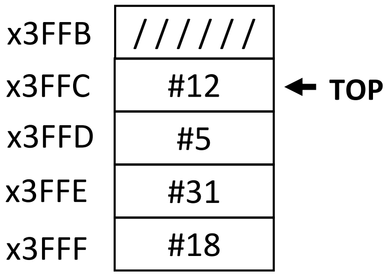
Why are Stack Data Structures useful?

- Saving and Restoring of registers when we call a subroutine
  - PUSH to save when we enter
  - POP to restore when before we return

- Stacks enable subroutines (and functions and methods) to be ***re-entrant****
  - They can be interrupted
  - They can call other subroutines, and have control return back to them, possibly ***recursively****
  - Part of the foundation for ***multi-threading****

*****These are big new concepts for many of you, and you'll be exposed to them in more detail later in this course and in others

Basic Push and Pop Code



x3FFB	//////	
x3FFC	#12	← TOP
x3FFD	#5	
x3FFE	#31	
x3FFF	#18	

Using Software Implementation of Stack

- **Push (R0 contains the data to be pushed)**

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

- **Pop (R0 contains the data after popped)**

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

- What if we Push when the stack is full? **Overflow**
- What if we Pop when the stack is empty? **Underflow**

Our implementation

- BASE: beginning of stack in memory (initial state)
- MAX: end of stack in memory
- START: Location of most recent element pushed (R6)

Address/Label	
x3FFB	;end of stack
…	
x3FFF	; Base of the stack
X4000	; start of stack
MAX	.FILL x3FFB
BASE	.FILL x4000
START	….

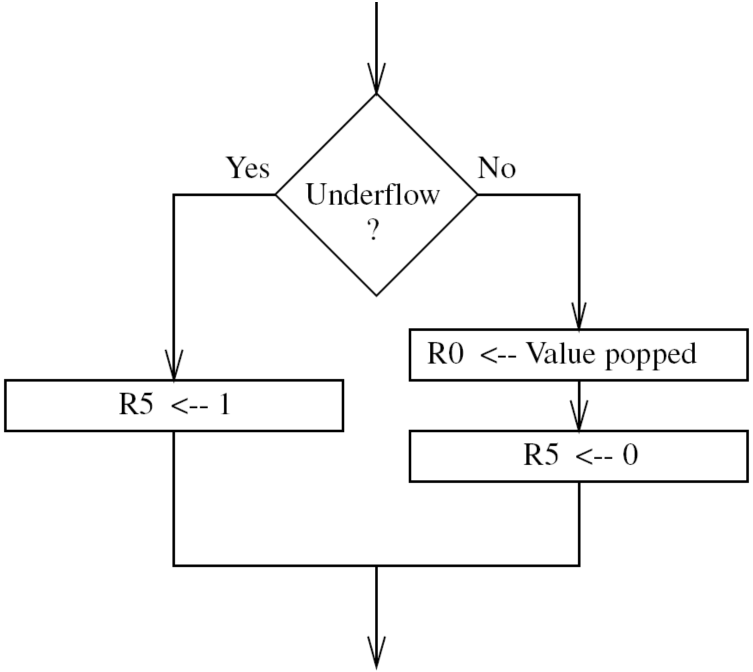
Overflow and Underflow

- Given MAX, BASE, START, how do we determine…
- Overflow?
- Underflow?

Label/address	
x3FEF	
x3FFB	XXXXXXXXXXXX
…	XXXXXXXXXXXX
x3FFD	XXXXXXXXXXXX
x3FFE	XXXXXXXXXXXX
x3FFF	XXXXXXXXXXXX
x4000	XXXXXXXXXXXX
MAX	.FILL x3FFB
BASE	.FILL x4000
START	…….

A Full Stack

Figure 10.4 POP routine including the test for underflow



Our implementation

- BASE: beginning of stack in memory (initial state)
- MAX: end of stack in memory
- START: Location of most recent element pushed (R6)

Address/Label	
x3FFB	;end of stack
…	
x3FFF	; Base of the stack
X4000	; start of stack
MAX	.FILL x3FFB
BASE	.FILL x4000
START	….

Push 18

Address/Label	
x3FFB	;end of stack
…	
x3FFF	18
x4000	
MAX	.FILL x3FFB
BASE	.FILL x4000
START	x3FFF

Push 31

Address/Label	
x3FFB	;end of stack
…	
x3FFE	31
x3FFF	18
X4000	
MAX	.FILL x3FFB
BASE	.FILL x4000
START	x3FFE

Push 5

Address/Label	
x3FFB	;end of stack
…..	
x3FFD	5
x3FFE	31
x3FFF	18
X4000	
MAX	.FILL x3FFB
BASE	.FILL x4000
START	x3FF3D

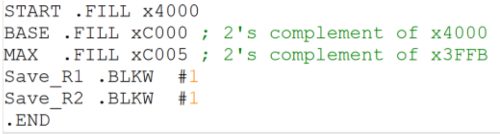
Pop (return 5)

Address/Label	
x3FFB	;end of stack
…..	
x3FFD	5
x3FFE	31
x3FFF	18
X4000	
MAX	.FILL x3FFB
BASE	.FILL x4000
START	x3FF3E

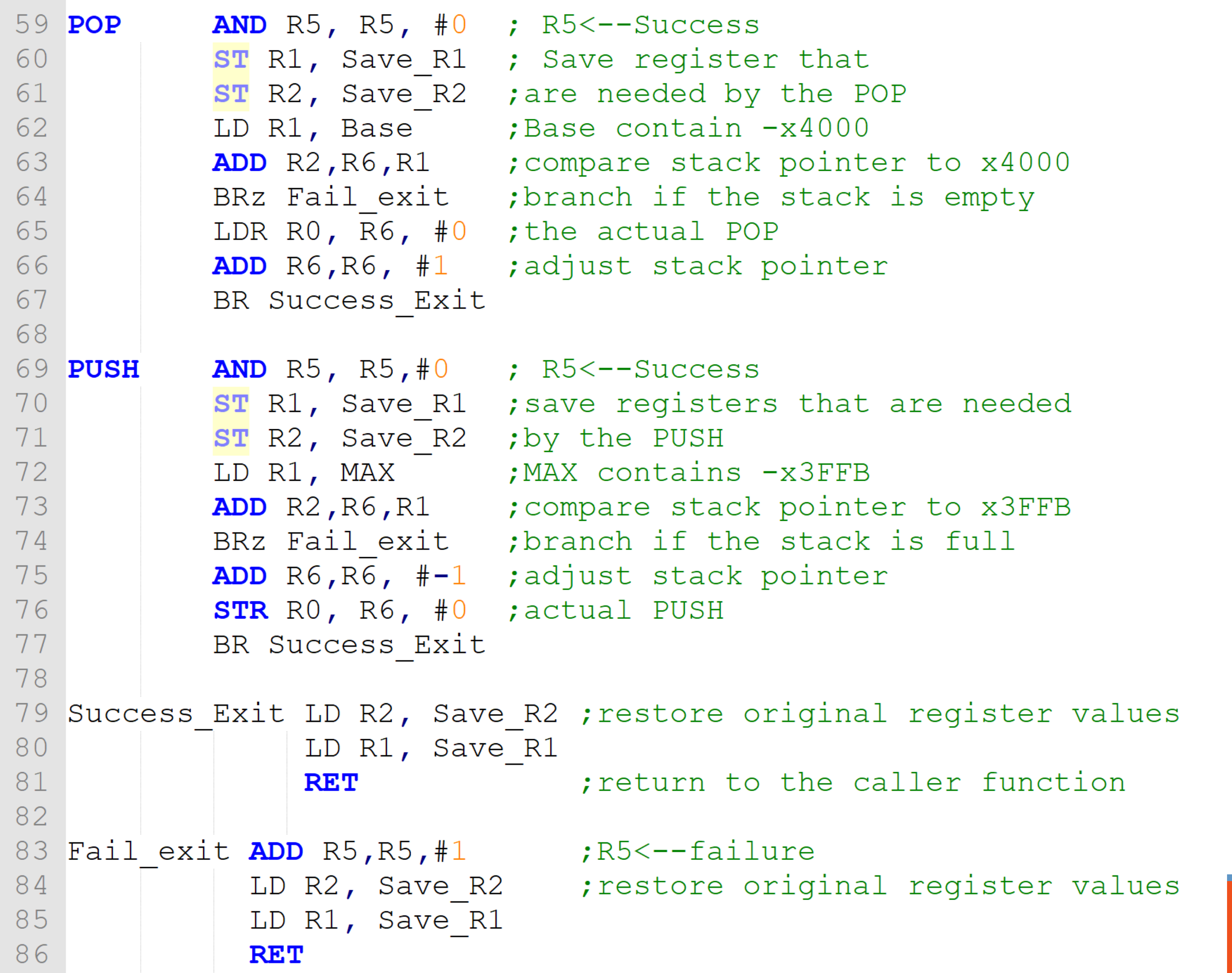
PUSH/POP implementation

BASE x4000
Top of the Stack – R6 (Stack Pointer)
Load – R0 (value to be popped)
Output – R5 (success / fail)

MAX x3FFB
Top of the Stack – R6 (Stack Pointer)
Store – R0 (value to be Pushed)
Output – R5 (success / fail)



```
START .FILL x4000
BASE .FILL xC000 ; 2's complement of x4000
MAX  .FILL xC005 ; 2's complement of x3FFB
Save_R1 .BLKW   #1
Save_R2 .BLKW   #1
.END
```



```
POP     AND R5, R5, #0   ; R5<--Success
        ST R1, Save_R1   ; Save register that
        ST R2, Save_R2   ;are needed by the POP
        LD R1, Base      ;Base contain -x4000
        ADD R2,R6,R1     ;compare stack pointer to x4000
        BRz Fail_exit    ;branch if the stack is empty
        LDR R0, R6, #0   ;the actual POP
        ADD R6,R6, #1    ;adjust stack pointer
        BR Success_Exit

PUSH    AND R5, R5,#0    ; R5<--Success
        ST R1, Save_R1   ;save registers that are needed
        ST R2, Save_R2   ;by the PUSH
        LD R1, MAX       ;MAX contains -x3FFB
        ADD R2,R6,R1     ;compare stack pointer to x3FFB
        BRz Fail_exit    ;branch if the stack is full
        ADD R6,R6, #-1   ;adjust stack pointer
        STR R0, R6, #0   ;actual PUSH
        BR Success_Exit

Success_Exit LD R2, Save_R2 ;restore original register values
             LD R1, Save_R1
             RET            ;return to the caller function

Fail_exit ADD R5,R5,#1      ;R5<--failure
          LD R2, Save_R2    ;restore original register values
          LD R1, Save_R1
          RET
```

Exercise:

> Push two values into the Stack
> Pop those values from the stack
> Add the poped Values and
> put the result back into the stack