ECE 220 Computer Systems & Programming

Lecture 4: Introduction to Stack Data Structures

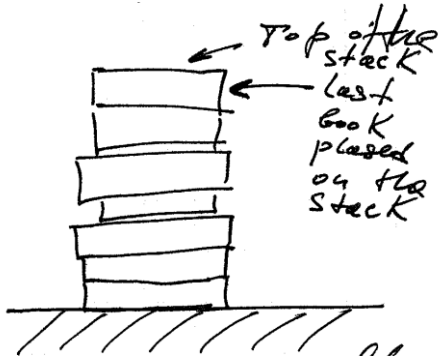
Jan 24th 2019

ECE ILLINOIS

ILLINOIS

Palindromes:

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



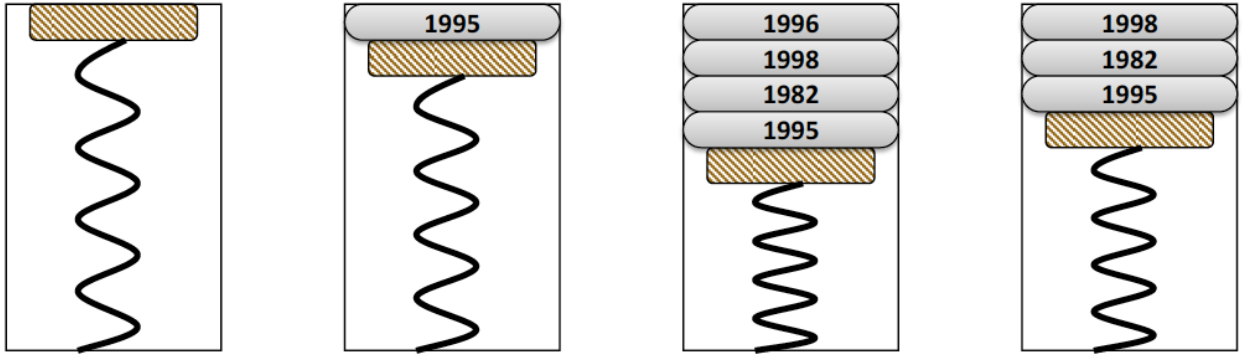
Outline

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

- Chapter 10 in textbook

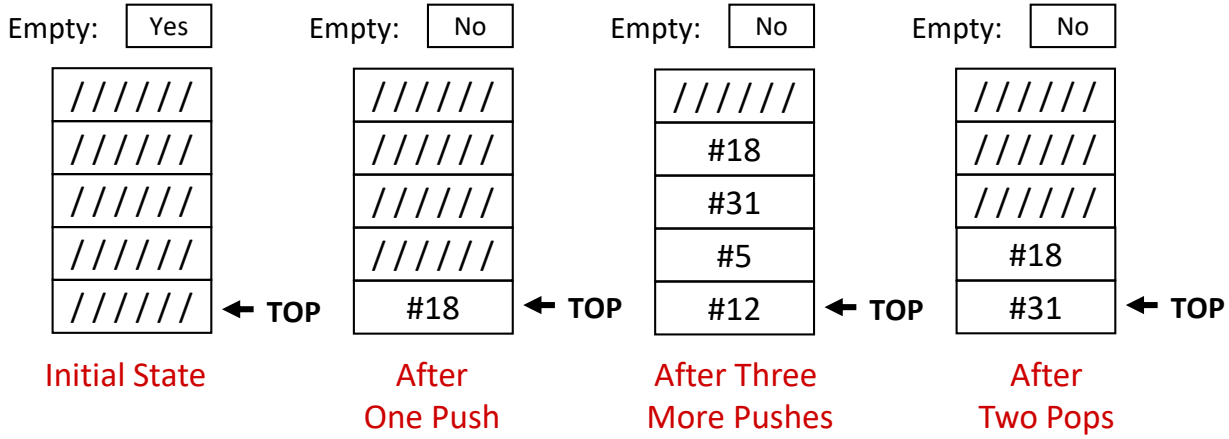
Coin Holder Example

- **First coin in is the last coin out**



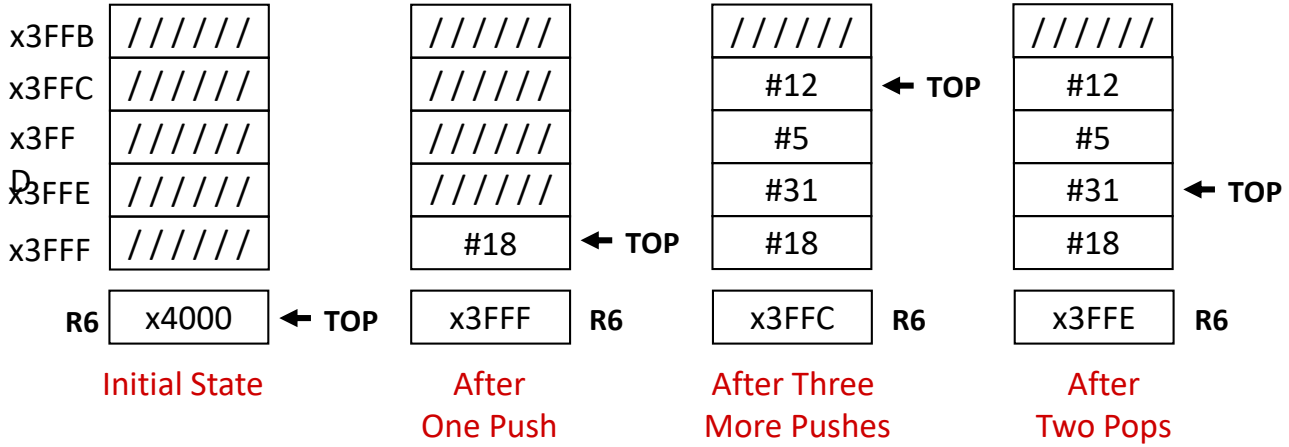
A Hardware Implementation

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



A Software Implementation

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



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

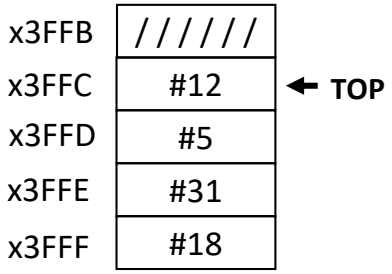
Why are Stack Data Structures useful?

- 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



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**

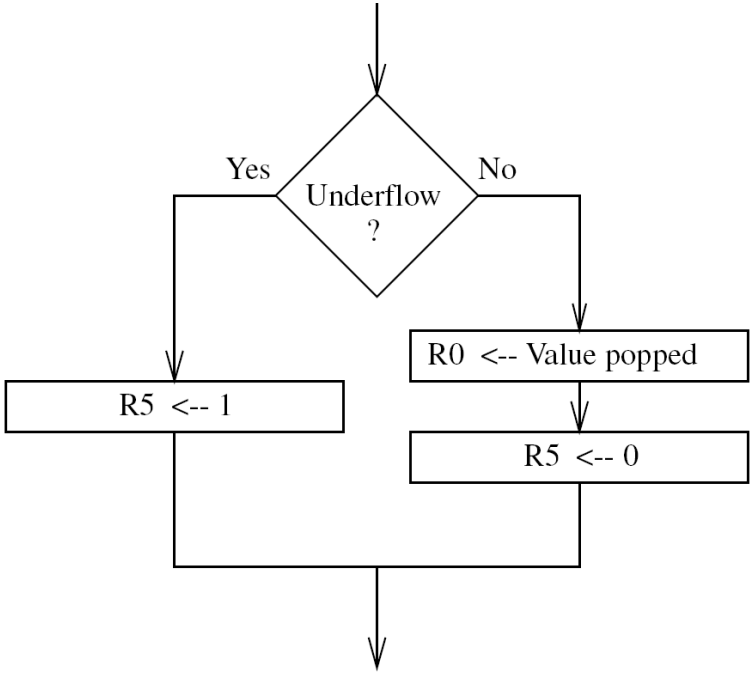
Overflow and Underflow

- Given STACK_TOP, STACK_END, STACK_START, how do we determine…
- Overflow?
- Underflow?

Label/address	
x3FEF	
x3FFB	XXXXXXXXXXX
…	XXXXXXXXXXX
x3FFD	XXXXXXXXXXX
x3FFE	XXXXXXXXXXX
x3FFF	XXXXXXXXXXX
x4000	XXXXXXXXXXX
STACK_TOP	x3FEB
STACK_START	x4000
STACK_END	x3FFB

A Full Stack

Figure 10.4 POP routine including the test for underflow



Underflow, Overflow detection:

```
POP       LD      R1,EMPTY
          ADD     R2,R6,R1
          BRz     Failure
          LDR     R0,R6,#0
          ADD     R6,R6,#1
          AND     R5,R5,#0
          RET
Failure   AND     R5,R5,#0
          ADD     R5,R5,#1
          RET
EMPTY     .FILL   xC000
                  ; EMPTY <-- -x4000
```

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

```
PUSH      LD      R1,MAX
          ADD     R2,R6,R1
          BRz     Failure
          ADD     R6,R6,#-1
          STR     R0,R6,#0
          AND     R5,R5,#0
          RET
Failure   AND     R5,R5,#0
          ADD     R5,R5,#1
          RET
MAX       .FILL   xC005
                  ; MAX <-- -3FFB
```

Stack_End x3FFB
Top of the Stack – R6 (Stack Pointer)
Load – R0 (value to be popped)
Output – R5 (success / fail)

Our implementation

- STACK_START: beginning of stack in memory
- STACK_END: end of stack in memory
- STACK_TOP: Location of most recent element pushed

Address/Label	
x3FFB	;end of stack
…	
x3FFF	; Base of the stack
X4000	; start of stack
STACK_END	.FILL x3FFB
STACK_START	.FILL x4000
STACK_TOP	….

Push 18

Address/Label	
x3FFB	;end of stack
…	
x3FFF	18
x4000	
STACK_END	.FILL x3FFB
STACK_START	.FILL x4000
STACK_TOP	x3FFF

Push 31

Address/Label	
x3FFB	;end of stack
…	
x3FFE	31
x3FFF	18
X4000	
STACK_END	.FILL x3FFB
STACK_START	.FILL x4000
STACK_TOP	x3FFE

Push 5

Address/Label	
x3FFB	;end of stack
…..	
x3FFD	5
x3FFE	31
x3FFF	18
X4000	
STACK_END	.FILL x3FFB
STACK_START	.FILL x4000
STACK_TOP	x3FF3D

Pop (return 5)

Address/Label	
x3FFB	;end of stack
…..	
x3FFD	5
x3FFE	31
x3FFF	18
X4000	
STACK_END	.FILL x3FFB
STACK_START	.FILL x4000
STACK_TOP	x3FF3E

Simple example

```
1   .ORIG X3000
2   ;ITEM1 X18
3   ;ITEM2 X31
4   ;ITEM3 X5
5   ;MAIN PROGRAM:
6       LD R6,STACK_START    ;LOAD R6 WITH STACK_START
7       LD R0,ITEM1          ;load the ITEM1 into R0
8       JSR PUSH
9       LD R0,ITEM2          ;load the ITEM2 into R0
10      JSR PUSH
11      LD R0,ITEM3          ;load the ITEM3 into R0
12      JSR PUSH
13      JSR POP              ;POP ITEM3 INTO R0
14
15      HALT
```

Pushes three values into the stack, and pops one value from the stack.

```
52  ;Values to be pushed into the stack
53  ITEM1          .FILL   x18
54  ITEM2          .FILL   x31
55  ITEM3          .FILL   x5
56  STACK_START    .FILL   x4000
```

```
17  ; Subroutines for carrying out the PUSH and POP functions.  This
18  ; program works with a stack consisting of memory locations x3FFF
19  ; (BASE) through x3FFB (MAX).  R6 is the stack pointer.
20  ;
21  POP             ST      R2,Save2        ; are needed by POP.
22                  ST      R1,Save1
23                  LD      R1,BASE         ; BASE contains -x3FFF.
24                  ADD     R1,R1,#-1       ; R1 contains -x4000.
25                  ADD     R2,R6,R1        ; Compare stack pointer to x4000
26                  BRz     fail_exit       ; Branch if stack is empty.
27                  LDR     R0,R6,#0        ; The actual "pop."
28                  ADD     R6,R6,#1        ; Adjust stack pointer
29                  BRnzp   success_exit
30  PUSH            ST      R2,Save2        ; Save registers that
31                  ST      R1,Save1        ; are needed by PUSH.
32                  LD      R1,MAX          ; MAX contains -x3FFB
33                  ADD     R2,R6,R1        ; Compare stack pointer to -x3FFB
34                  BRz     fail_exit       ; Branch if stack is full.
35                  ADD     R6,R6,#-1       ; Adjust stack pointer
36                  STR     R0,R6,#0        ; The actual "push"
37  success_exit    LD      R1,Save1        ; Restore original
38                  LD      R2,Save2        ; register values.
39                  AND     R5,R5,#0        ; R5 <-- success.
40                  RET
41  fail_exit       LD      R1,Save1        ; Restore original
42                  LD      R2,Save2        ; register values.
43                  AND     R5,R5,#0
44                  ADD     R5,R5,#1        ; R5 <-- failure.
45                  RET
```

```
47  ; BASE contains -x3FFF.
48  BASE            .FILL   xC001
49  MAX             .FILL   xC005
50  Save1           .FILL   x0000
51  Save2           .FILL   x0000
52
53  ;Values to be pushed into the stack
54  ITEM1           .FILL   x18
55  ITEM2           .FILL   x31
56  ITEM3           .FILL   x5
57  STACK_START     .FILL   x4000
58  .END
```