CS061:
Machine Organization & Assembly Language
Lab 6

Agenda

1. Presentation:
    a. Sub-routine Review
    b. Stacks
    c. Reverse Polish Notation
    d. If - Statements
    e. Lab Descriptions

2. Work Time / Questions / Demos

Creating a Subroutine

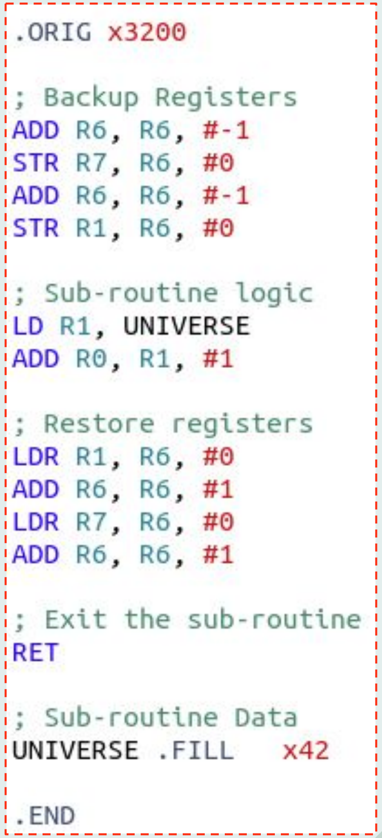
1. Define sub-routine at remote location (e.g. x3200, x3400, etc)!

2. Backup registers!
   a. Always backup R7!
   b. Backup registers **changed in the sub-routine**!
   c. Don’t backup registers that are return values.

3. Put in your sub-routine logic code.
   a. Code that does some task.
   b. E.g. Store input into an array, compute a value, etc

4. Restore registers!
   a. Restore registers in the reverse order there were backed up!

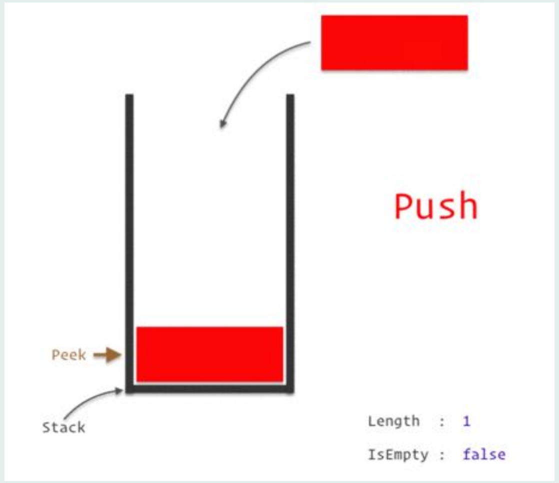
5. Exit out of the sub-routine (RET)!



```
.ORIG x3200

; Backup Registers
ADD R6, R6, #-1
STR R7, R6, #0
ADD R6, R6, #-1
STR R1, R6, #0

; Sub-routine logic
LD R1, UNIVERSE
ADD R0, R1, #1

; Restore registers
LDR R1, R6, #0
ADD R6, R6, #1
LDR R7, R6, #0
ADD R6, R6, #1

; Exit the sub-routine
RET

; Sub-routine Data
UNIVERSE .FILL   x42

.END
```

Stack'd

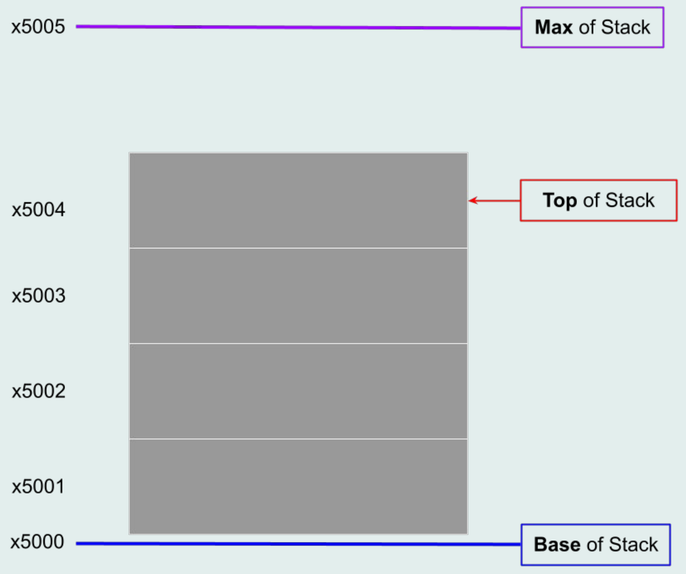
- A stack is an abstract data type with LIFO ordering!
  - Can only read/write to top of stack!
- Push Operation:
  - Push (add) a value onto the top of the stack!
  - Top of stack increases!
- Pop Operation:
  - Pop (take) a value from the top of the stack!
  - Top of stack decreases!



https://fullyunderstood.com/stack/

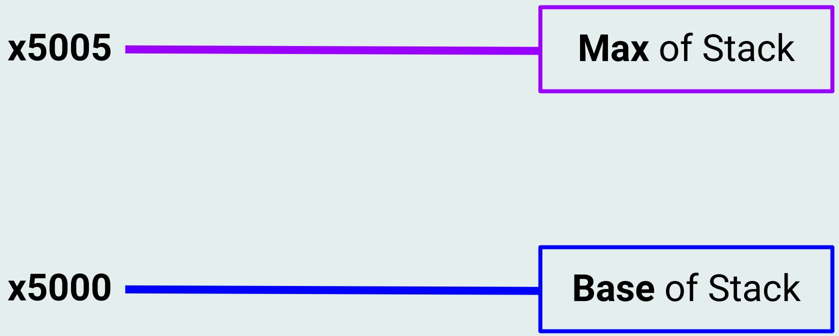
Stack Terms

- **Top of Stack** Address:
  - A memory address that refers to the element at the top of the stack!
- **Base** of Stack Address:
  - Pointer to the bottom of the stack!
  - Think of this like the floor!
  - Bottom of the stack **is not an element.**
  - Top of stack address can't be than the base address!
- **Max** of Stack Address:
  - Pointer to the maximum of the stack!
  - Think of this like the ceiling!
  - Max of stack **is not an element**.
  - Top of stack address can't be greater than the max address!



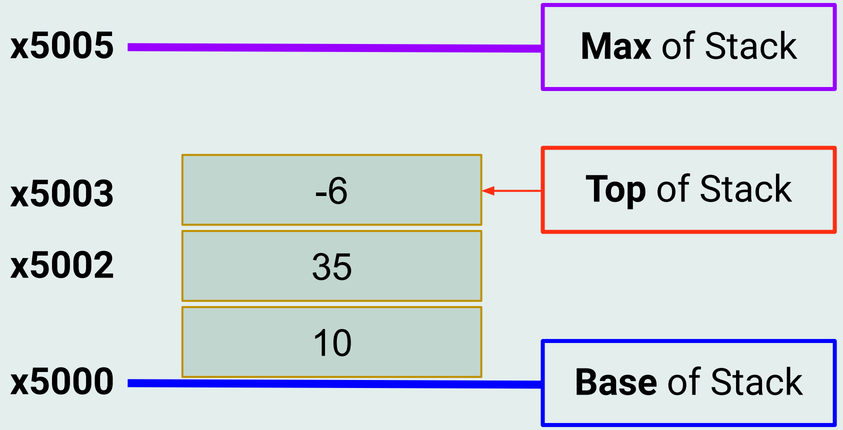
Constrained Stack

- Our stack is constrained by some bounds!
- **Overflow Error**: Trying to push elements past the max address of the stack!
- **Underflow Error**: Trying to pop elements below the base address of the stack!



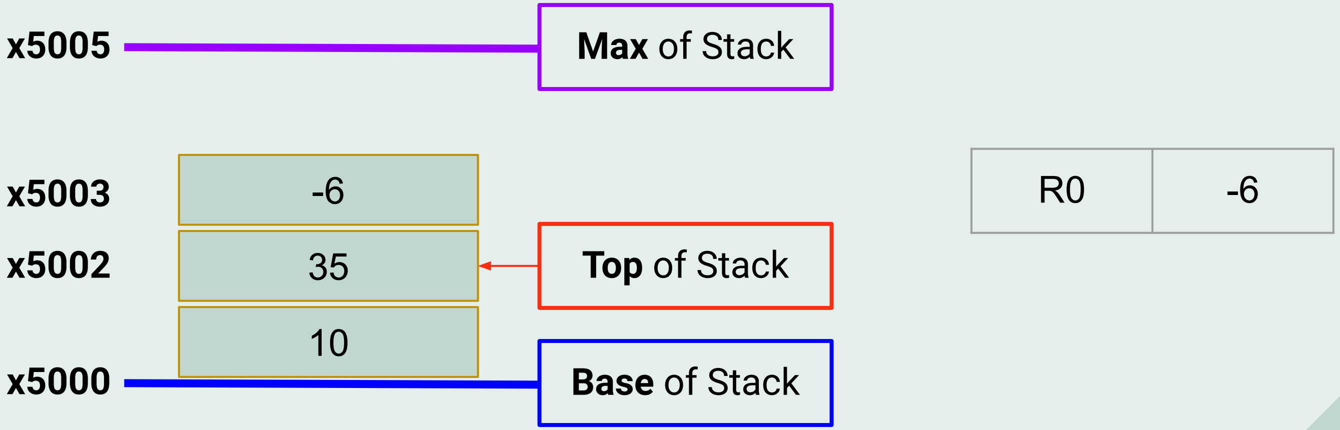
Implementing Push

1. Verify that the TOS (top of stack) address < MAX address!
   a. If it is above/equal to the MAX address, print out a Stack Overflow error message and exit!
2. Increment the TOS address!
3. Write the element to the TOS address!



POP!

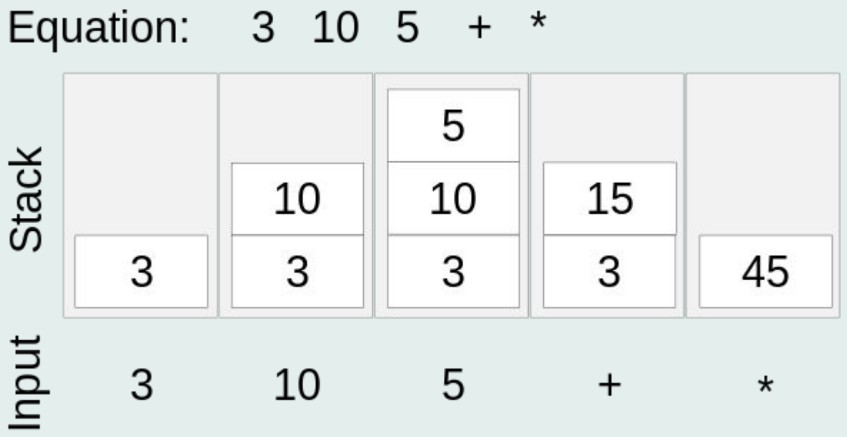
1. Verify that the TOS address > BASE address!
    a. If it's not, print out an underflow error message and exit!
2. Copy the value at the top of the stack to some destination register.
3. Decrement the TOS address.



Reverse Polish Notation

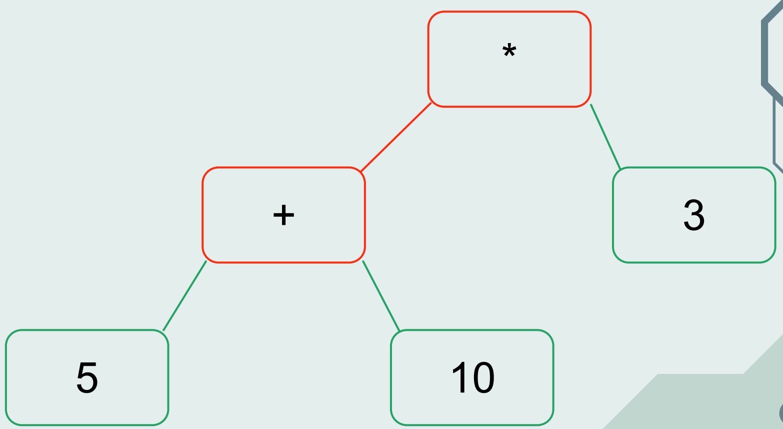
- Reverse Polish Notation (Post-fix)
  - Method of representing expressions!
  - Operator goes after numeric operands!

Original Equation: (5 + 10) * 3



https://upload.wikimedia.org/wikipedia/commons/thumb/5/53/CPT-RPN-example1.svg/600px-CPT-RPN-example1.svg.png

Tree Representation

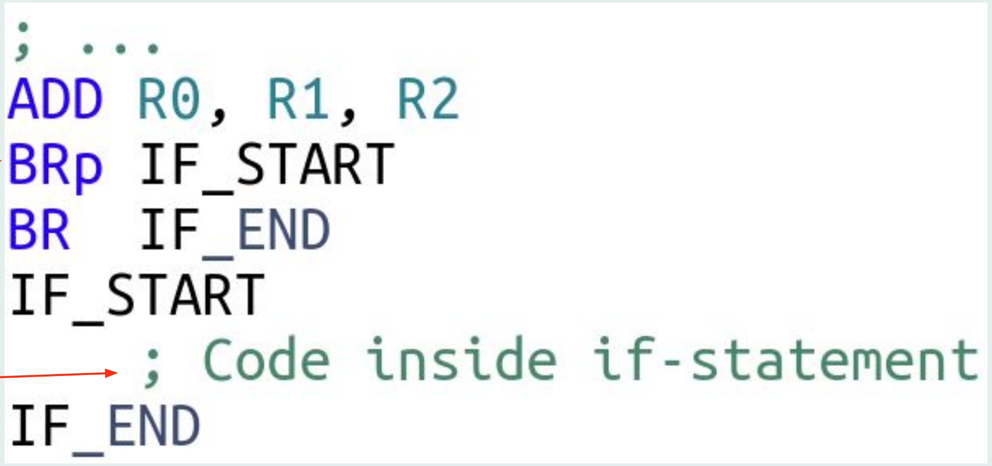


If Statement

- Goal: Execute IF statement if R0 is positive!

- If R0 is positive, jump to IF_START
- Otherwise, jump to IF_END

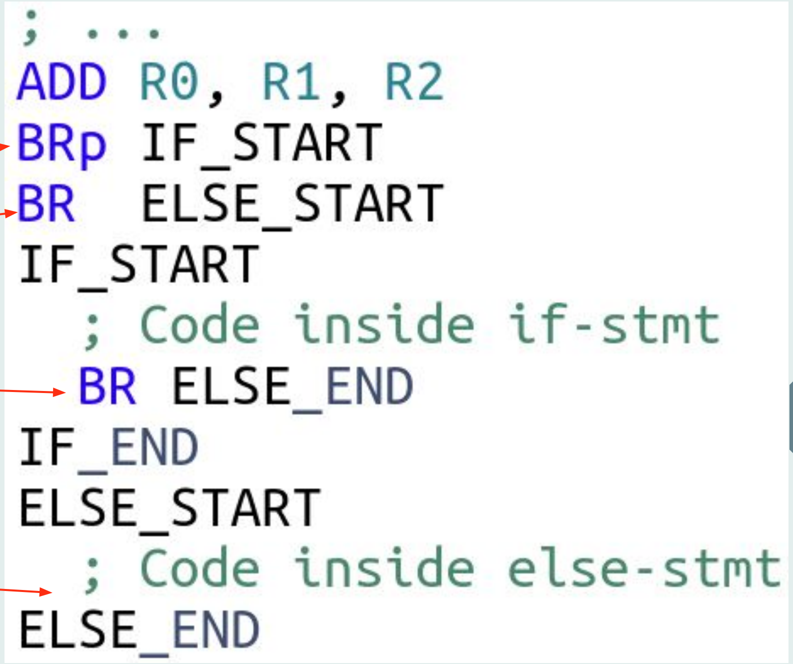
- Code only executes if R0 is positive.



```
; ...
ADD R0, R1, R2
BRp IF_START
BR  IF_END
IF_START
    ; Code inside if-statement
IF_END
```

If-Else

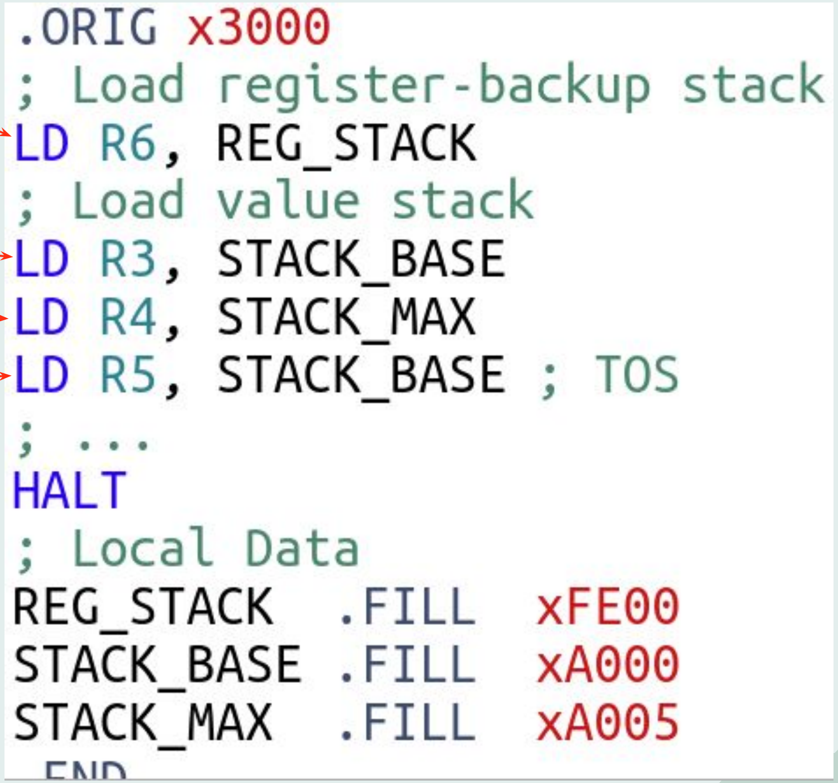
- Goal: Execute IF statement if R0 is positive, otherwise execute ELSE statement!
- If R0 is positive, jump to IF_START
- Otherwise, jump to ELSE_START
- **Skip the else statement**
- *Why no BR over here?*



```
; ...
ADD R0, R1, R2
BRp IF_START
BR  ELSE_START
IF_START
   ; Code inside if-stmt
   BR ELSE_END
IF_END
ELSE_START
   ; Code inside else-stmt
ELSE_END
```

Starter Code

- Separate stack used for backing up / restoring registers.
- Base of Stack Address
- Max of Stack Address
- Top of Stack Address
    - Starts out = BASE



```
.ORIG x3000
; Load register-backup stack
LD R6, REG_STACK
; Load value stack
LD R3, STACK_BASE
LD R4, STACK_MAX
LD R5, STACK_BASE ; TOS
; ...
HALT
; Local Data
REG_STACK  .FILL  xFE00
STACK_BASE .FILL  xA000
STACK_MAX  .FILL  xA005
```

Exercise 1

- Implement the stack push sub-routine!
- Test Harness: Push values (e.g. hard-coded, user-input, etc) to stack!
  - Demonstrate overflow error!

```
;-----------------------------------------------------------------------------------------
; Subroutine: SUB_STACK_PUSH
; Parameter (R1): The value to push onto the stack
; Parameter (R3): BASE: A pointer to the base (one less than the lowest available address) of
the stack
; Parameter (R4): MAX: The "highest" available address in the stack
; Parameter (R5): TOS (Top of Stack): A pointer to the current top of the stack
; Postcondition: The subroutine has pushed (R1) onto the stack (i.e to address TOS+1).
;               If the stack was already full (TOS = MAX), the subroutine has printed an
;               overflow error message and terminated.
; Return Value: R5 ← updated TOS
;-----------------------------------------------------------------------------------------
```

Exercise 2

- Implement stack pop sub-routine!
    - Make sure to put the value popped into R0!

- Test Harness: Pop values (e.g. hard-coded or from user-input)
    - Demonstrate underflow error!

```
;--------------------------------------------------------------------------------------------
; Subroutine: SUB_STACK_POP
; Parameter (R3): BASE: A pointer to the base (one less than the lowest available address) of the
stack
; Parameter (R4): MAX: The "highest" available address in the stack
; Parameter (R5): TOS (Top of Stack): A pointer to the current top of the stack
; Postcondition: The subroutine has popped MEM[TOS] off of the stack and copied it to R0.
;               If the stack was already empty (TOS = BASE), the subroutine has printed
;                an underflow error message and terminated.
; Return Values: R0 ← value popped off the stack
;               R5 ← updated TOS
;--------------------------------------------------------------------------------------------
```

Exercise 3

1. Copy your push and pop sub-routine to the exercise 3 file!

2. Implement a sub-routine SUB_RPN_ADDITION
   a. Pop two values off the stack.
   b. Add the values together.
   c. Push result onto the stack.

3. Main Program:
   a. Prompt user to enter a single-digit number.
      i. Convert ASCII character to number.
      ii. Push it onto stack (via the PUSH sub-routine)
   b. Repeat the step above to get the second number.
   c. Prompt user for an operator (the “+”)
      i. Can discard this input!
   d. Call your new sub-routine SUB_RPN_ADDITION .
   e. Pop result off from stack (via POP sub-routine)
   f. Print out result to console (via a helper sub-routine PRINT_DIGIT)

Demo Info

- **Lab Grade Breakdown:**
    - 3 points for attendance.
    - 7 points for demoing (+1 bonus point demo’d before/during Friday).
    - 3 point penalty if lab is demo’d during the next lab session.

- **Tips before you demo:**
    - ***Understand your code!*** (Know what each line does & the input/output)
    - ***Test your code!*** (Check for correct output and that there are no errors)