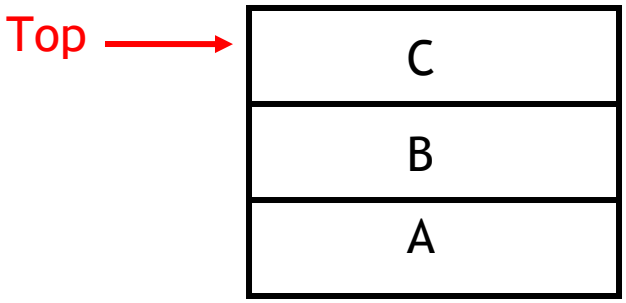
## The general Stack

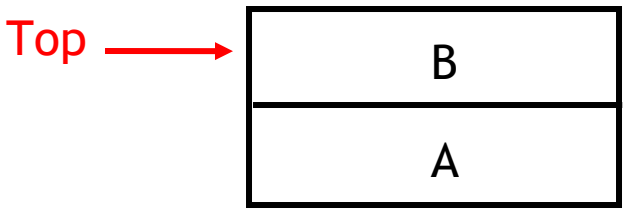
A stack is a data structure that bears the Last In First Out (LIFO) property for data retrievals. Data can only be inserted to or retrieved from one end of the stack. The location where data can be inserted/retrieved is known as the "top" of the stack, which is usually pointed to by a pointer. Normally two operations are possible in a stack, the Pop operation and the Push operation. The Pop operation retrieves an element from the top of the stack and then removes it. The Push operation, on the other hand, stores data to the stack. The operations of pop/push are illustrated by the figures below:

Original stack with 1 element:


After performing the push(B) and push(C) operations, we get the following stack:

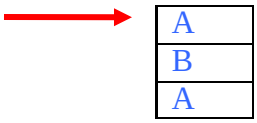


Then after performing pop() for once we get




Example: Draw the resulting stack after performing following sequence of operations, assume the stack is empty initially (the operations must be performed in the exact order as listed ):

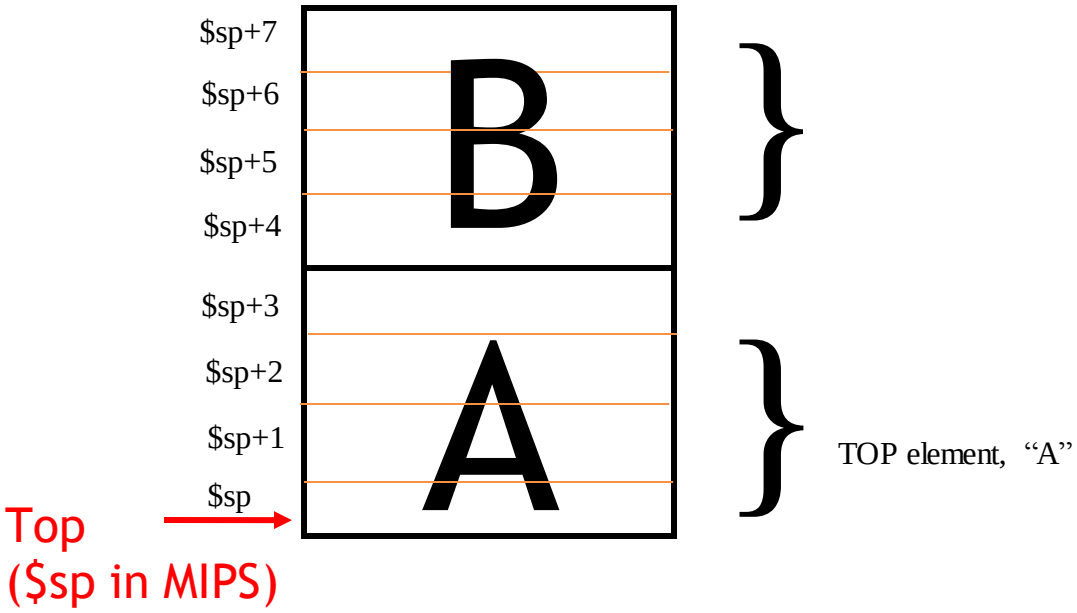
Push(A), Push(C), Pop(), Push(D), Pop(), Push(1), Push(3), Pop(), Pop()



## MIPS Stack

For MIPS architecture, the stack data structure is supported through the use of the stack pointer $sp. $sp holds the address of the TOP element of the stack. Assume each stack element is 4-byte in size, then the 4-byte TOP element occupies the 4 bytes at the addresses $sp, $sp+1, $sp+2, $sp+3 (see the picture below). $sp holds the smallest address among these 4 bytes. Note that by providing $sp, the lw/sw instructions will be able to access the TOP element (the bytes being loaded/stored will be at $sp, $sp+1, $sp+2, $sp+3). For example if you want to load the top element to the $s0 register, you can issue "lw $s0, ($sp)".

**Figure** to illustrate the structure of a stack under MIPS




**The MIPS PUSH operation**

Under MIPS, the push action (i.e. store an element and make it the new TOP of the stack using the $sp register), is usually done through:

```
addi $sp, $sp, -4  # update the $sp to allocate 4 bytes for the new top
sw $s0, ($sp)      # store the 32-bit data from $s0 to the TOP( $sp,$sp+1,$sp+2,$sp+3)
```

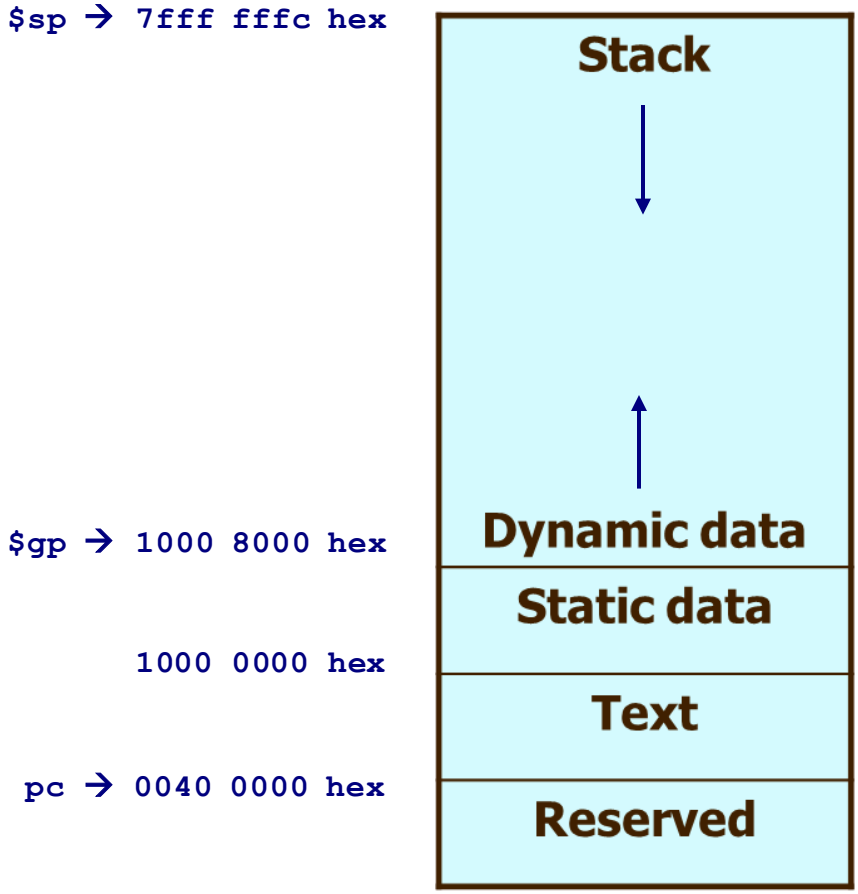
Mind that when you enlarge the stack under MIPS, you are decreasing the $sp value, because as the MIPS stack grows, it grows to the lower part of the memory space as being shown on the picture in slide 28 of the note set at http://course.cse.ust.hk/comp2611/note/comp2611_ISA_Spring2016_part2.pdf, the picture is copied below for your quick reference.

**The MIPS POP operation**

Under MIPS, the pop action (i.e. read the top element and remove it from the stack by updating the $sp register) is done through:

```
lw $s0, ($sp)      # load the top element to $s0, assume the top element is 32-bit
addi $sp, $sp, 4   # update the $sp to reflect the new TOP, and exclude the older TOP from the
                   # stack
```

**Figure** to show MIPS memory layout




If a procedure "pushes" *n* element(s) to the stack at the beginning of it, and then "pops" for *n* times at the end of it (which is generally the case for us), then the item(s) the procedure pushed to the stack will disappear from the stack when the procedure exits (LIFO property of stack). For the caller of that procedure, it looks as if the procedure has never done anything to the stack, and the stack looks same to the caller before and after calling the procedure.