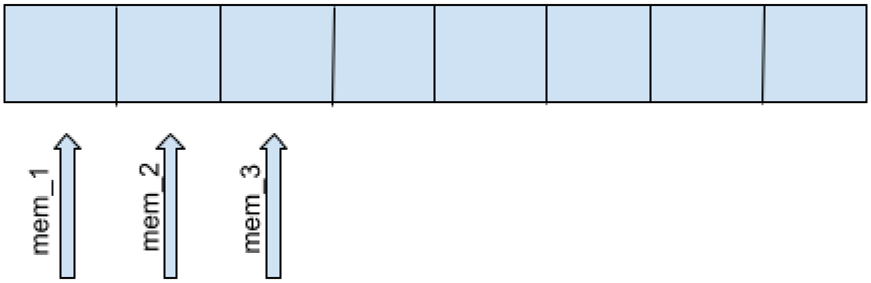
Documentation

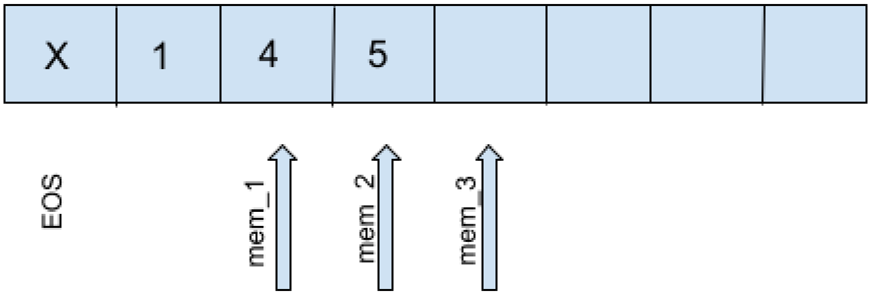
We have used C++ for writing the transpiler to convert the code in super stack to IITK Traveller.

As mentioned in the documentation of IITK-Traveller , we can conclude that the all three books represent the same array and the indexing/page number of each book represents pointers mem_1,mem_2 and mem_3 at respective default positions 0,1 and 2 And initially at each index is marked to zero.

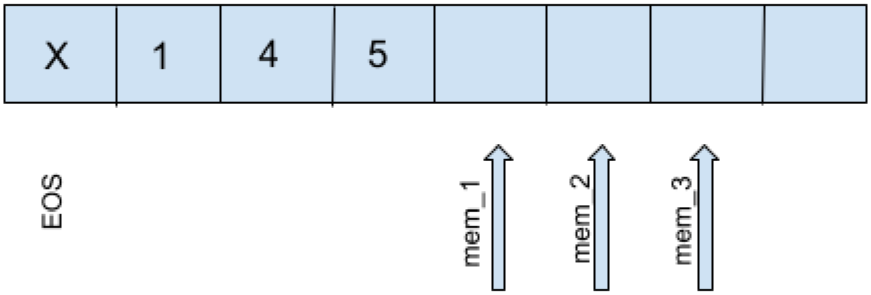


Pointer positions

Pointer positions were initially kept at the beginning of the memory array at values 0, 1, 2 respectively. At first, we tried to implement the pointer positions such that when at least three numbers are input into the array, then the pointers mem_1, mem_2, mem_3 would begin to shift right by one position. So, at any given time, mem_2 would point to the most recent input to the array .



But, later we changed the implementation so that, as numbers were added into the stack , the positions of mem_1, mem_2 and mem_3 were moved to the right ,keeping the same order in the memory array. So, now mem_1 would point to the position to the right of the most recent input. Functions were implemented based on manipulation of these pointers.



Functions

- Here, store in pointer x signifies that

Addition : ADD

For addition, take pointer 1,2 two positions backward and pointer 3 one position backwards and then take addition of mem_1, mem_2 and store in mem_3. Then copy the mem_3 to mem_1, remove data stored in mem_2,mem_3. Move pointer 1,2 one position forward. Hence the state of pointers is preserved.

Subtraction : SUB

For subtraction take pointer 1,2 two positions backward and pointer 3 one position backwards and then take subtraction of mem_2 from mem_1 and store in mem_3. Then copy the mem_3 to mem_1, remove data stored in mem_2,mem_3. Move pointer 1,2 one position forward.
Hence the state of pointers is preserved.

Multiplication : MUL

For multiplication take pointer 1,2 two positions backward and pointer 3 one position backwards and then take multiplication of mem_1, mem_2 and store in mem_3. Then copy the mem_3 to mem_1, remove data stored in mem_2, mem_3. Move pointer 1,2 one position forward.
Hence the state of pointers is preserved.

Division : DIV

For division, take pointer 1,2 two positions backward and pointer 3 one position backwards and then take division of mem_1, mem_2 and store in mem_3. Then copy the mem_3 to mem_1, remove data stored in mem_2,mem_3. Move pointer 1,2 one position forward.
Hence the state of pointers is preserved.

Input: input

For taking a single number as input and pushing it to stack we take input using iit_gate_in_1 at the position where mem_1 is pointing and then move the pointers mem_1 , mem_2 and mem_3 each one position forward.
Hence the state of pointers is preserved.

Output: output

In order to pop the top number and output it with a space we first move all the pointers mem_1 , mem_2 and mem_3 one position backward. Hence the state of pointers is preserved. Then output the

number at mem_1 using iit_gate_out_1 (iit_gate_out_1 prints the value with a space automatically).
And then set the value stored at mem_1 to be zero.

If and fi

If and fi are written such that when the if is executed the condition variable's value is pushed in the vector maintained in the transpiler' stack. Then the lecture_hall_eq function is executed, since the value stored in mem2 is 0 it checks the equality of mem1 with 0 and goes to lecture_hall_eq_t of equal and lecture_hall_eq_f if not. Similar to what is in super-stack.

Debug

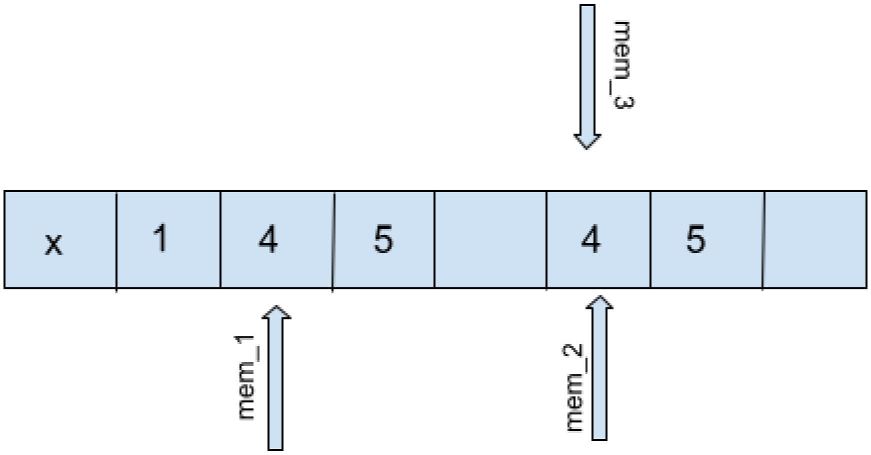
For debug function firstly insert an EOS at pointer 1 and move pointer 1 one position backwards and then run a loop checking equality of pointer 1 with EOS if EOS is not present then move pointer 1 backwards until it hits EOS. Once the pointer 1 is at EOS the state will be something like this.

Now move the pointer 1 forward by 1 position. And go to iit_gate_out_1 to print mem1 until it hits EOS.

Then remove the EOS at mem1 initially added.
The final pointer positions and memory state will be preserver and it will output the desired value.

Swap

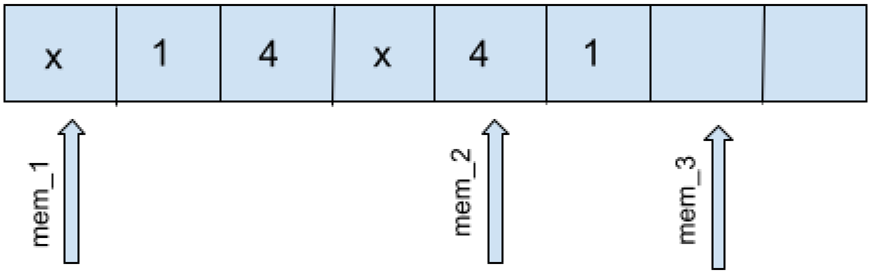
For swapping the top two numbers in the stack move the pointer 1 backwards by 1 position and copy it in mem3. Move the pointer 3 and pointer 1 backwards and again copy the mem1 in mem3. The memory will look like this.



Now move pointer 3 forward by 1 position and copy mem_3 in mem1. Move mem_1 forward by 1 position, copy mem_2 in mem_3 and copy mem_3 in mem_1. Clear mem_2, mem_3 & move the mem_1 pointer forward.

<u>Cycle</u>

Add EOS at pointer 1, move pointer 1,3 backward then go to events_1 and until mem_1 is at EOS copy mem1 in mem3 and move pointer 1 back and pointer 3 forward. The memory will look like this.

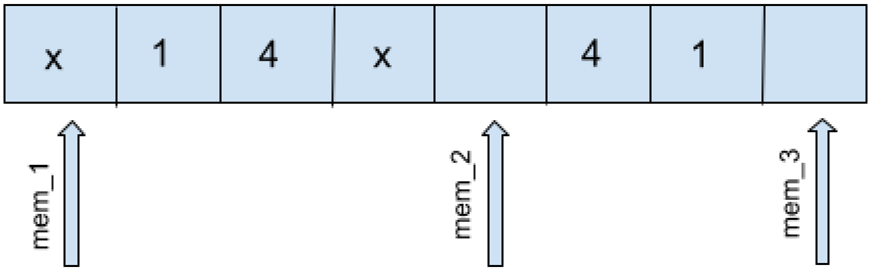


Move the pointer 1 forward and copy mem2 in mem3 and mem3 in mem1 and go to events_1 to copy mem3 in mem1, empty mem3 by hall_13_3 and move pointer 1 forward and 3 backward until pointer 1 is at EOS.

Then change the pointer 1,2,3 to initial state.

Rcycle

Add EOS at pointer 1, move pointer 1 backward the go to events_1 and until mem1 is at EOS copy mem1 in mem3 and move pointer 1 back and pointer 3 forward. The memory will look like this.



Move the pointer 1 forward and pointer 3 backwards and go to events_1 to copy mem3 in mem1 , empty mem3 by hall_13_3 and move pointer 1 forward and 3 backward until pointer 1 is at EOS.

Then change the pointer 1,2,3 to initial state.

Outputascii

To output the ascii character we moved each of the pointer mem_1 , mem_2 , mem_3 one position backward and output the character pointed by mem_1 using nankari_gate_out_1 and then set the value to 0 pointed by mem_1. Hence preserving the initial order of the pointers.

Bitwise Operators:
A separate class has been created to handle the bitwise operations separately.

Not

To implement it, we move mem_1 one position backward and decrease the value at mem_2 by 1. Then we multiply values at mem_1 and mem_2 and store it at mem_3, then we store the value at mem_3 in mem_1. Then we reduce the value at mem_1 by 1, reinitialise the values at mem_2, mem_3 to 0 and move mem_1 forward by one space. Hence preserving the initial order of pointers.

And

To implement this, first of all we extract the bits of each number using internal while loops in the IITK-Traveller language.
First of all, if the top number is smaller than the second number, we swap the two numbers. Then we store the bits extracted from the top number, then add an EOS literal to mark the ending position. Then further store the bits extracted from the second number.

Once we have extracted the bits, we calculate the bitwise “and” simply by multiplying the bits. This is because “and” operation results in 1 when both bits are 1 and results 0 if atleast one of the bits is 0. Since the same output is produced when we multiply the bits, we use this to calculate the bitwise “and”.

Now to generate the number back, we again use an internal while loop that does the following:

1. Multiply the bit by current power of 2 and add it to the result
2. Multiply the current power of 2 by 2

With result initialized to 0.

OR

Calculated using the “add” and “and” operations by using the following identity.

a+b = (a|b) + (a&b)

XOR

Calculated using the “add” and “and” operations by using the following identity.

a^b = (a|b) - (a&b)

NAND

Calculate by simply using the function “and” followed by “not” operation.

Pop

To pop the number we moved each of the pointers mem_1 , mem_2 , mem_3 one position backward and set the value to 0 pointed by mem_1.

Dup

To duplicate the number we moved the pointer mem_1 and mem_2 one position backward and copied the value pointed by mem_1 to mem_3 and then again copied the value from mem_3 to mem_2 and set the value to 0 pointed by mem_3. And then we shifted pointers 1 and 2 by 2 units forward and then pointer 3 to one position forward to maintain the state of pointers.

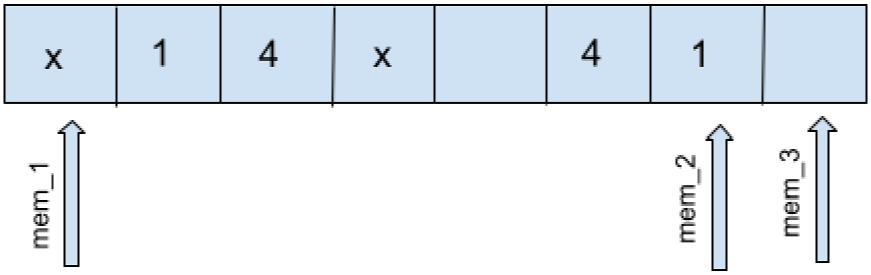
Rev

For rev, 2 approaches can be used. One similar to cycle, rcycle and the other can be to copy the reel in reversed order and add EOS in end of the initial reel and move the pointers forward thus making 2 copies of reel but ignoring the previous one through EOS.

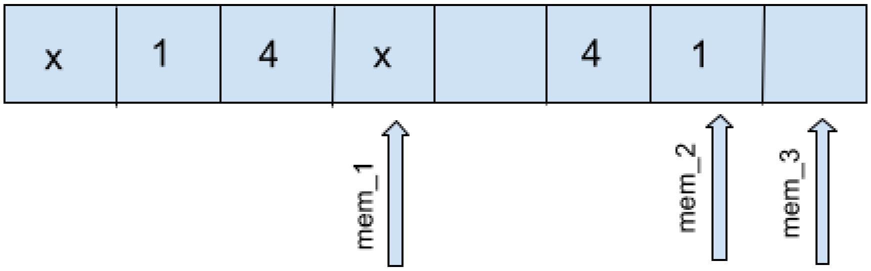
In this function a lot of memory is being wasted but since the goal is to write minimum lines of code in IITK-Traveller the second approach is used.

Approach:

Add EOS at pointer 1, move pointer 1 backward to go to events_1 and until mem1 is at EOS copy mem1 in mem3 and move pointer 1 back and pointer 3,2 forward. The memory will look like this.



Move mem_1 forward and go to events_1 for entering loop to move pointer 1 forward until pointer 1 is at EOS.
Memory state will be like this.



Now move pointer 1,2,3 forward, add EOS at pointer 1,2.
Go to events_1 to loop moving pointer 1 forward until it hits EOS.

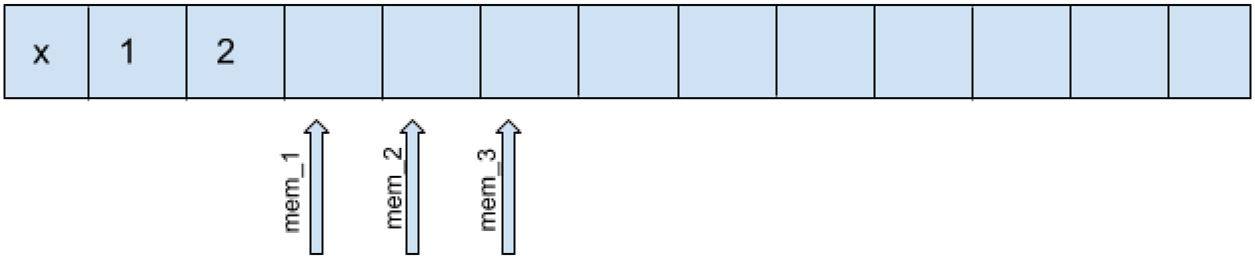
After exiting the loop, remove the EOS at 1.

<u>Inputascii</u>

For implementing this function in IITK-Traveller. Add eos at mem1, Take input of the ascii string using Airstrip_land and then move the pointer 1 to EOS, remove the EOS stored at start of the string and copy the reverse order of string to the array(memory reel) while removing the copied array from pointer 3. And move the pointers to respective initial positions.

Memory representation of function:

*Ascii value converted to letters in representing in dig.



Submitted by- Hall2 Kshatriyas

- Rishi Agarwal (210849)
- Rudradeep Datta (220919)
- Ritvik Goyal(220898)
- Akhil Agarwal(200076)