Stacks and Queues

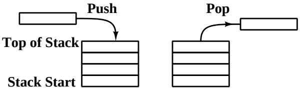
Dinesh Sharma

Electrical Engineering Department
IIT Bombay

March 13, 2021

Stacks

The stack is a data structure where the item last added to the collection of data is the first one to be retrieved.

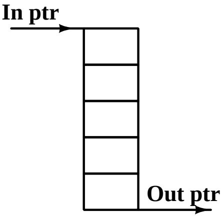


- A stack of items has this property – last added item is the first to be retrieved. Hence the name.
- One end of the stack is fixed. The other grows as we add items (push) and shrinks as we remove them (pop).
- The end where the addition or removal takes place is called the top of stack (often abbreviated to TOS).
- Since addition and removal takes place at the same end, we track the top of stack with a single pointer called the stack pointer.

The stack grows towards higher addresses in an 8051 and towards lower addresses for most other processors.

Queues

In a queue, the earliest item added to the collection of data is the first one to be retrieved.



- In a queue we insert items at one end and retrieve from the other end.
- Thus we need two pointers, one pointing to the address where a new item will be added and the other pointing to the address from where an item can be retrieved.

- A linear queue as shown is not a practical data structure because it will grow continuously at the entry end even though items are being removed from it continually.
- A more practical data structure is a circular queue.

Circular Queues

In a linear queue, the memory vacated from where an item has been removed is no more useful. this is corrected in a circular queue.

- The circular queue starts from a given location and the maximum size of the queue is pre-decided.
- The in-pointer is incremented as items are added. If the in pointer exceeds the maximum value, it is reset to the start location.
- The assumption is that the size of the queue is large enough, so that by the time the in-pointer reaches the maximum value, items would have been removed from the bottom end.
- The out-pointer also starts from the start location and is incremented as items are retrieved from the queue. Again, when the out-pointer reaches the maximum value, it is reset to the start location.

Efficient Circular Queues

Implementation of circular queues is efficient when the size of the queue is some power of 2.

- For an arbitrary sized queue, each addition and removal involves incrementing a pointer, comparing it to a max value and resetting it to the start value if it exceeds it.
- The overhead of comparison and resetting is considerable, because it involves a conditional jump (which may lead to a pipeline flush).
- This overhead can be minimized if the queue length is some power of two.
- In this case, we just increment the pointer and ignore (or zero out) all bits $\geq$ the n'th bit, where the queue size is 2^n.

Efficient Circular Queues

- Consider a queue of size 8. Here n is 3. Let us treat the queue as an array. The in and out indices will start with a zero value and be incremented at every insertion or removal.
- At each insertion or removal, we increment the corresponding index and unconditionally AND it with 07H. No comparison is required.
- Till the index reaches 7, ANDing will not change the value of the index.
- When the index is at 7, it will incremented to 8 or to ‘0000 1000’. ANDing with 07H will give ‘0000 0000’, which is what we want.
- This is true for any queue size which is a power of two. Notice that we need to use indices into the array and not pointers to memory.