Lecture-7 (Stack Machines, Subroutine Calls, allocation and evaluation)

29 January 2024 14:57

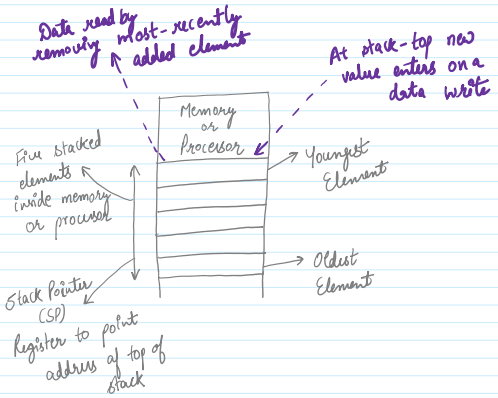
Stacks and Queues Addressing :-

Stack :-

last-in-first-out (LIFO)

→ set of locations.

→ each location can hold one word of data.



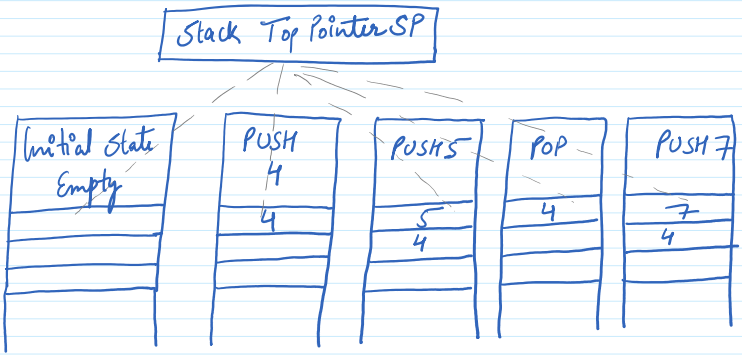
A Stack.

① Addition → All data moves down one location.

② Removal → " " up " "

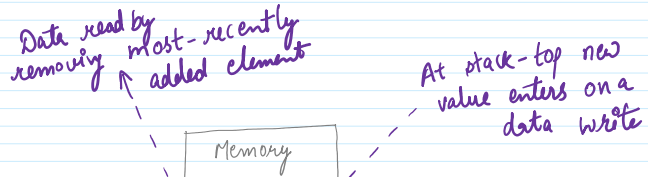
③ In general, data cannot be read from a stack without disturbing the stack.

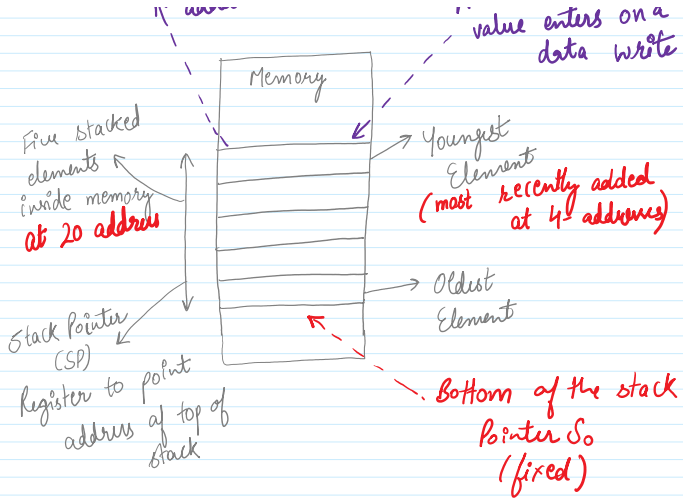
Basic operations : PUSH and POP



Set of PUSH and POP operations affecting a Stack.

Using Computer memory for implementing Stack Operations :-





A stack of 32-bit words implementation at the Memory.

S_0 is the highest memory address available

Example :- Assume 32-bit words in memory.

Use r_{13} as stack pointer. Move immediate # 0x00FFFF, an address in the memory for stack top.

Now place r_0 to r_2 on to stack.

What is the address of top of the stack?

MOV r_{13}, # 0x00FFFF

PUSH r_0; r_{13} will be $r_{13} - 4 =$ 0x00FFFB

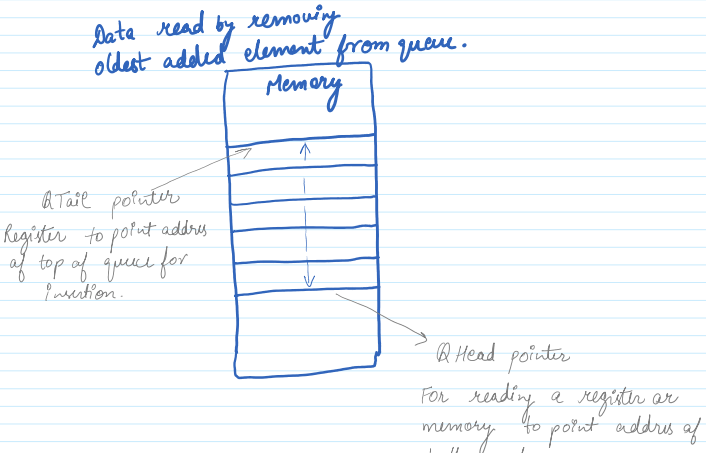
PUSH r_1; r_{13} will be $r_{13} - 4 =$ 0x00FFF7

PUSH r_2; r_{13} will be $r_{13} - 4 =$ 0x00FFF3

$\Rightarrow$ r_{13} will have the address 0x00FFF3 as SP.

Queue :-

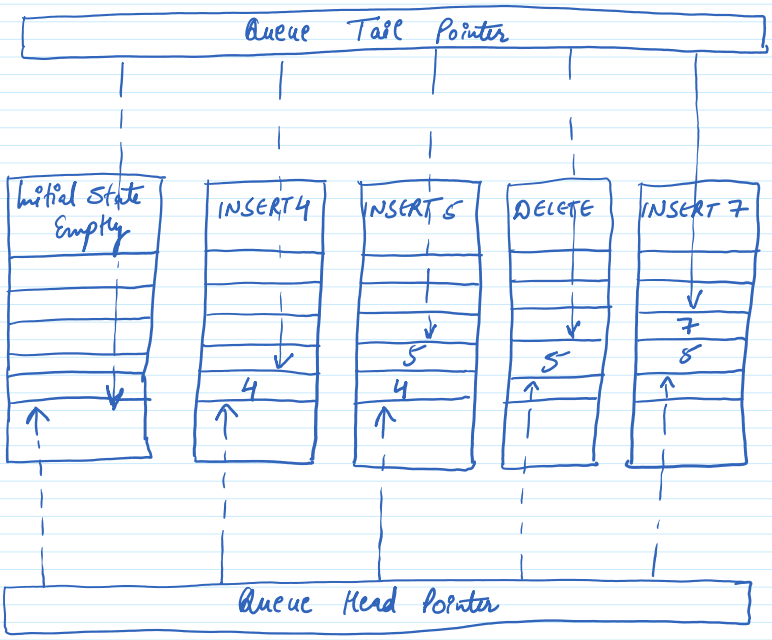
First-in-first-out (FIFO)



bottom of queue

A Queue.

Basic INSERT and DELETE operations.



Example:- Use r_7 and r_8 as queue tail and queue head pointers.
Assuming 32-bit memory.
Use 0x001000 as address in the memory for queue tail and head at start.
Insert r_0 to r_2 in the memory.

MOV r_7, #0x001000 (Tail Pointer)
MOV r_8, #0x001000 (Head Pointer)

INSERT r_0, r_7 will be $r_7+4 = 0x001004$
INSERT r_1, — — — $r_7+4 = 0x001008$
INSERT r_2, — — — $r_7+4 = 0x00100C$

r_7 ⟶ 0x00100C (Tail pointer)
r_8 ⟶ 0x001000 (Head pointer)

Subroutine Nesting using STACKS.
(Subroutine Calls)

A subroutine is a set of instructions or a sub-program or function (in C/C++).

Main program :

```
MOV r4, #1
MOV r3, #0d2000
MOV r2, #0d100000
MOV r1, #0x200000
MOV r0, #0d100000
Step1:
    CALL subroutine A  f()
Step2:
    CALL subroutine B  g()
```

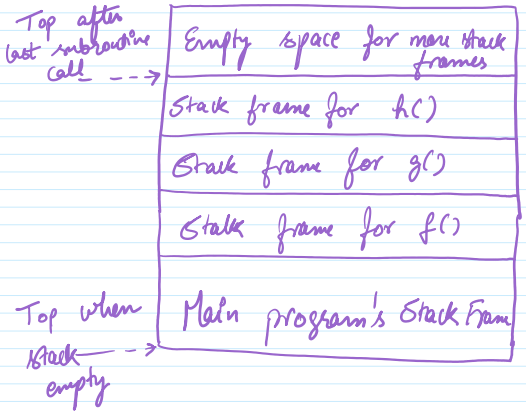
Subroutine A: f()

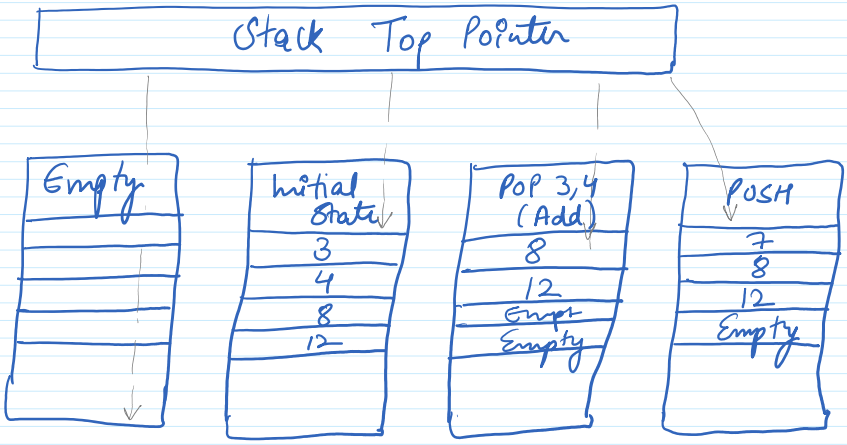
```
ST(r1), r3
ST(r1), r2
RET
```

Subroutine B: g()

```
ADD r2, r0
ADD r3, r4
CALL f()
RET
```

Sub-routines using Stacks :-





Memory operations in ADD instruction:

(a) fetch the instruction

(b) fetch the first operand from the stack

(c) fetch the second operand from the stack

(b) Write the result back onto the stack.

a → STACK

b → STACK

STACK ← a+b