Cell: 98204 08217
BharatSir@hotmail.com

STACKS AND SUBROUTINES

STACKS

A **stack** is a **part of the Memory** that **operates in LIFO** (Last In First Out) manner.
In 8085 the **Stack** is **located anywhere within the 64 KB memory**.
The **top of the Stack** is **pointed by** the **SP** (Stack Pointer) register.
P.S.: The SP contains the **Address** of the top of the Stack and **not** the **top of the Stack**.

Instructions affecting the stack are:

- **LXI SP**, 16 bit value Eg LXI SP 4000.
- **SPHL**
- **INX SP**
- **DCX SP**
- **PUSH** Rp Eg PUSH B
- **POP** Rp Eg POP B
- **CALL** 16 bit address Eg CALL 2000
- **RSTn** Eg RST1
- **RET**
- **XTHL**

Usefulness of Stacks (Uses of Stacks)

Stacks are used in the following ways:

1) For Temporary Storage of data

Stack can be used for **storing data by** the **programmer** as the number of General Purpose registers is very less. The programmer, at any point of time during the program, can store data in the stack and then retrieve it later.
Data can be Pushed into the Stack using Push instruction as below:
Eg : **PUSH B** ; Contents of BC Pair are pushed onto the top of the stack.

	Before	Operation		**After**
		SP ← SP – 1		
		[SP] ← B	SP	C
		SP ← SP – 1		B
SP	XX	[SP] ← C		XX

Similarly data can also be removed from the stack using the POP operation as:
Eg : **POP B** ; Contents of the top of the stack are popped into BC Pair.

2) By the µP for storing the return address for CALL instruction

The **µP uses** the Stack **for storing** the **return address** during **CALL** instruction.
During a CALL instruction, **µP** first pushes the return address (i.e. the current contents of PC) into the stack and then loads the branch address into PC, to branch to the subroutine.
Inside the subroutine, when the **µP** gets a RET instruction, it pops back the return address from the stack, and thus successfully returns to the main program.

This is illustrated as follows:

Eg : **CALL 2000**

Before		**Operation**	**After**	
		SP ← SP – 1		
		[SP] ← PC_H	SP	PC_L
		SP ← SP – 1		PC_H
SP	XX	[SP] ← PC_L		XX
		PC ← 2000		

Eg : **RET**

Before		**Operation**	**After**	
		PC_L ← [SP]		
SP	PC_L	SP ← SP + 1		
	PC_H	PC_H ← [SP]		
	XX	SP ← SP + 1	SP	XX

3) To Read/Write the contents of the Flag register

The programmer can Read / Write the contents of the Flag register using the Program Status Word **(PSW)**. The PSW consists of the Accumulator as the higher byte and the Flag register as the lower byte. The PSW is available only for PUSH and POP instructions.

To Read the contents of Flag register:

- The programmer pushes PSW into the stack and then pops it into any register pair (BC).
- The contents of the flag register are thus available in the C register (lower register).

Eg:

PUSH PSW

SP	F	;Accumulator and Flag
	A	;Reg Pushed into Stack
	XX	

POP B

		;Contents of Flag Reg
		;come into C-Reg and
SP	XX	;can be read.

To Write into the Flag register:

- The programmer loads the appropriate byte into the lower register of a register pair (eg: C reg of BC pair).
- Then the register pair is pushed into the stack.
- These contents are then popped into the PSW, and thus the byte that was originally loaded into C register is now **written** into the Flag register.

Eg:

PUSH B

SP	C	;B and C Registers are
	B	;Pushed into Stack
	XX	

POP PSW

		;Contents of C Reg
		;come into Flag Reg
SP	XX	;

This is the **ONLY method** by which the programmer can **write into** the **Flag register**.

4) **To Pass parameters to a Sub-Routine**

The programmer can use the Stack to pass parameters to Sub-Routines.
Before calling the Sub-Routine the **parameter** is **Pushed** into the Stack and then **inside** the **Sub-Routine** the **parameter** is **Popped from** the **Stack**.

Eg:

BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

```
LXI D 1200 ; Parameter 1020 Pushed        SUB:POP H ;Return address taken in HL
PUSH D     ; into the Stack                   POP B ;Parameter is accepted into
CALL SUB                                          .  ;BC pair.
ADD B                                             .
   .                                              .
                                              PCHL ;PC gets the return address
                                                   ; from HL
  HLT
```

#Please refer Bharat Sir's Lecture Notes for this ...

BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

SUBROUTINES

- SubRoutines are parts of a program, which can be re-executed by the programmer.
- SubRoutines are Called (invoked) using the CALL instruction; control returns to the main program using the RET instruction.
- SubRoutines generally perform tasks, which are used regularly by the program such as numerical calculations, Interrupt Service Routines (ISR) etc.
- SubRoutines are useful in the following ways:
  1. Causes **Code Re-Usability** hence reduces the **size** of the Program and also **saves time**.
  2. Makes the program easy to **Maintain** as it becomes **Modular**.

Passing Parameters to Sub-Routines

Passing parameters to SubRoutines is the procedure of passing some values from the main program to the SubRoutines. Passing parameters is very important as it **improves the flexibility of the SubRoutine**.
In 8085 there are 4 methods of passing parameters to SubRoutines.

1) Through Registers

- The parameter value to be passed is moved /loaded into the register before calling the SubRoutine.
- The SubRoutine accesses the value from the same register.

Eg:

```
MVI C,25     ; parameter value 25 moved into C
CALL SUB
ADD D
  .
  .
  .
HLT

SUB:  MOV A,C    ; Parameter value accessed from C-Reg
        .
        .
        .
      MOV D,A
      RET
```

This is the **simplest method** of passing parameters.
The only main **drawback** is that it **occupies** the **general-purpose registers** available to the programmer whish are already very few. **Also** the **number** of parameters passed is **restricted by** the number of **registers available**.

2) Through One or More Memory Locations

- The parameter value to be passed is stored into the Memory Location before calling the SubRoutine.
- The SubRoutine accesses the value from the same memory location.

Eg:

```
MVI A,25
STA 2000     ; parameter value 25 moved into location 2000
CALL SUB
ADD D
  .
  .
  .
HLT

SUB:  LDA 2000   ; Parameter value accessed from location 2000.
        .
        .
        .
      MOV D,A
      RET
```

If **more parameters** are to be passed, then **consecutive memory locations** can be used (which are **plenty**), thus **eliminating** the **drawbacks** of the previous method.
The **only drawback** here is that the programmer needs to **remember** the **memory locations** associated with each subroutine. When the number of subroutines is large, this can be troublesome.

3) Through Memory Pointer "M"

- The parameter value to be passed is moved into the Memory pointer **"M"** before calling the SubRoutine. ☺ In case of doubts, contact Bharat Sir: - 98204 08217.
- The HL pair at this point of time may contain any random value.
- The SubRoutine accesses the value from M.
- Care has to be taken that after the value is put into M the value of **HL pair** should not change until the SubRoutine accesses the value of M.

Eg

```
MVI M,25     ; parameter value 25      SUB:  MOV A,M   ; Parameter value
             ; moved into Memory                        ; accessed from location
             ; pointer "M"                              ; 2000.

CALL SUB                                     .
ADD D                                        .
   .                                         .
   .                                     MOV D,A
   .                                     RET
HLT
```

Thus the memory is used but without remembering the memory location.
i.e. In the above example, we still **do not know** the value of **HL** pair, and thus the **address** of the memory location used (thus avoiding the previous drawback).
But, when **more** than one values have to be passed, this method becomes **troublesome**.

4) Through Stack ※ *#Please refer Bharat Sir's Lecture Notes for this ...*

- The **parameter** value **to be passed** is **Pushed** into the Stack before calling the SubRoutine.
- The SubRoutine **Pops** the **passed parameter** from the Stack.
- When the **µP** calls the SubRoutine it **pushes** the **return address** into the stack.
- This address appears above the passed parameter in the stack.
- Due to the **LIFO** property of the stack we cannot access the passed parameter unless we pop the return address.
- Hence inside the SubRoutine **firstly**, the **return address** is **popped into** the **HL** pair.
- Then the **passed parameter** is **popped** into the BC pair.
- Now in this case the **RET** instruction **cannot be used** to come back to the main program as the top of the stack no longer contains the return address (as we had taken it out in the HL pair).
- Thus the return address is obtained from the HL pair using the **PCHL** instruction which **causes** the control to **return to the main program**.

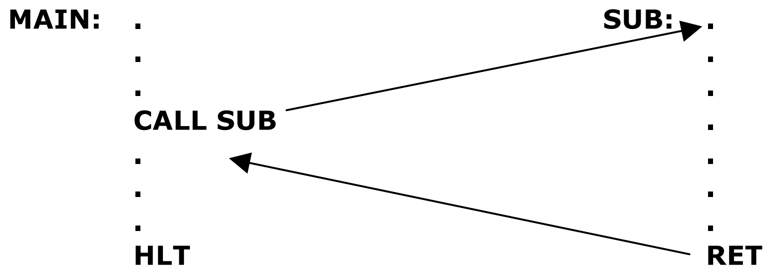
Eg:

```
LXI D 1200; Parameter 1020 Pushed        SUB:POP H;Return address taken in HL
PUSH D    ; into the Stack                   POP B ;Parameter is accepted into
CALL SUB                                         .  ;BC pair.
ADD B                                            .
   .                                             .
                                             PCHL ;PC gets the return address
                                                  ; from HL
  HLT;
```

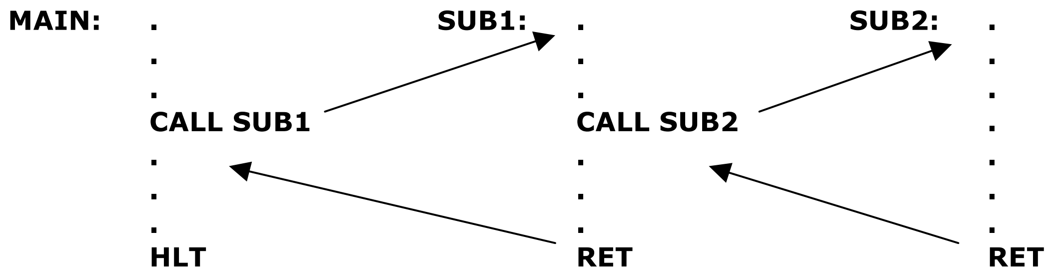
BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

TYPES OF SUB-ROUTINES

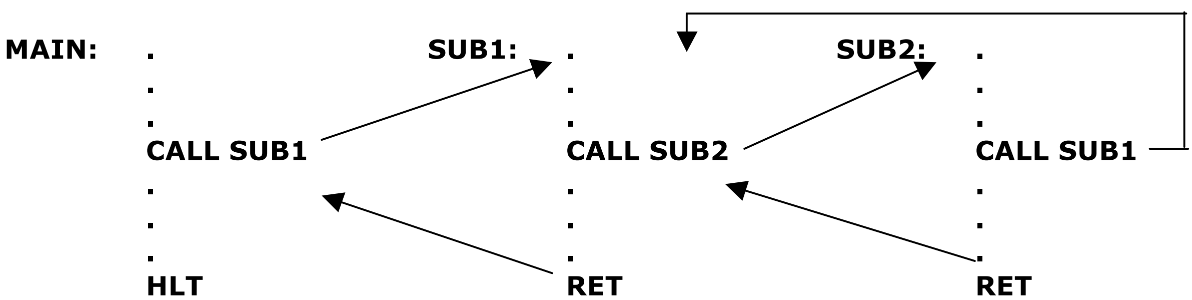
1. **Simple SubRoutines**
   When a SubRoutine does not call another SubRoutine it is called a Simple SubRoutine.



2. **Nested SubRoutines**
   When a SubRoutine calls another SubRoutine it is called a Nested SubRoutine.



3. **Re-entrant SubRoutines**
   When a SubRoutine is re-entered by another SubRoutine it is called a Re-entrant SubRoutine.



4. **Recursive SubRoutines**
   When a SubRoutine calls itself, it is called a Recursive SubRoutine.