Stack and Subroutines

Stack is a set of memory locations in the Read/Write memory which is used for temporary storage of binary information during the execution of a program. It is implemented in the Last-in-first-out (LIFO) manner. i.e., the data written first can be accessed last, One can put the data on the top of the stack by a special operation known as PUSH. Data can be read or taken out from the top of the stack by another special instruction known as POP.

Stack is implemented in two ways. In the first case, a set of registers is arranged in a shift register organization. One can PUSH or POP data from the top register. The whole block of data moves up or down as a result of push and pop operations respectively. In the second case, a block of RAM area is allocated to the stack. A special purpose register known as stack pointer (SP) points to the top of the stack. Whenever the stack is empty, it points to the bottom address. If a PUSH operation is performed, the data are stored at the location pointed to by SP and it is

decremented by one. Similarly if the POP operation is performed, the data are taken out of the location pointed at by SP and SP is incremented by one. In this case the data do not move but SP is incremented or decremented as a result of push or pop operations respectively.

Application of Stack: Stack provides a powerful data structure which has applications in many situations. The main advantage of the stack is that,

We can store data (PUSH) in it with out destroying previously stored data. This is not true in the case of other registers and memory locations.

stack operations are also very fast

The stack may also be used for storing local variables of subroutine and for the transfer of parameter addresses to a subroutine. This facilitates the implementation of re-entrant subroutines which is a very important software property.

The disadvantage is, as the stack has no fixed address, it is difficult to debug and document a program that uses stack.

Stack operation: Operations on stack are performed using the two instructions namely PUSH and POP. The contents of the stack are moved to certain memory locations after PUSH instruction. Similarly, the contents of the memory are transferred back to registers by POP instruction.

For example let us consider a Stack whose stack top is 4506 H. This is stored in the 16-bit Stack pointer register as shown in Fig.29

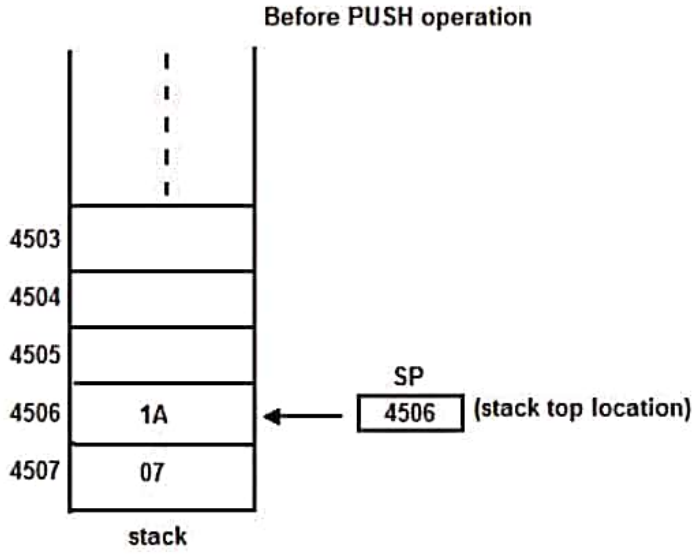
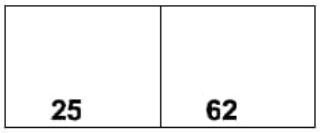


Figure.29 The PUSH operation of the Stack

Let us consider two registers (register pair) B & C whose contents are 25 & 62.

Reg. B Reg. C

After PUSH 

operation the status of the Stack is as shown in Fig 3.30

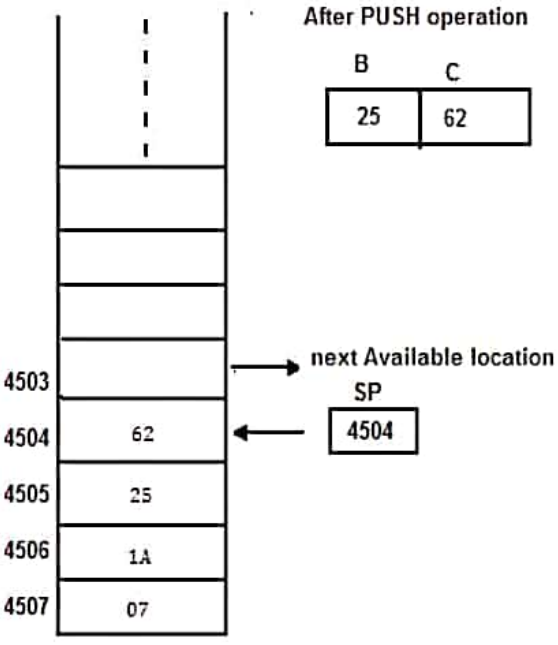


Figure .30 After PUSH operation the status of the stack

Let us now consider POP operation: The Figs 31 & 32 explains before and after the POP operation in detail

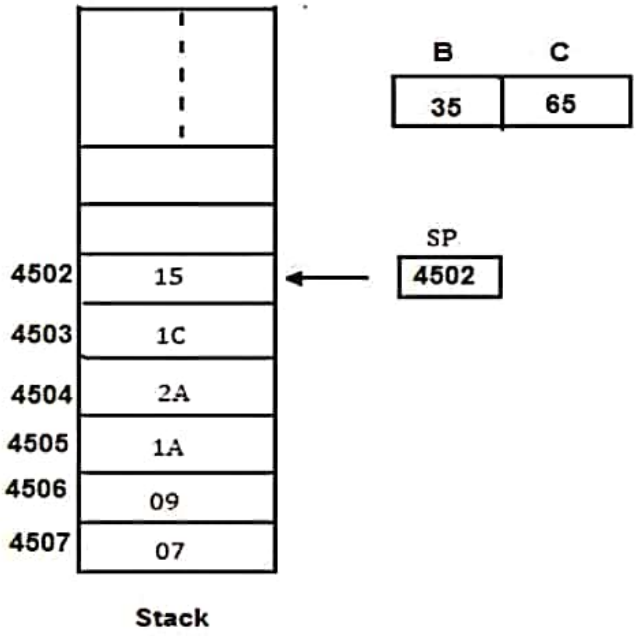


Figure 31 The POP operation of the Stack

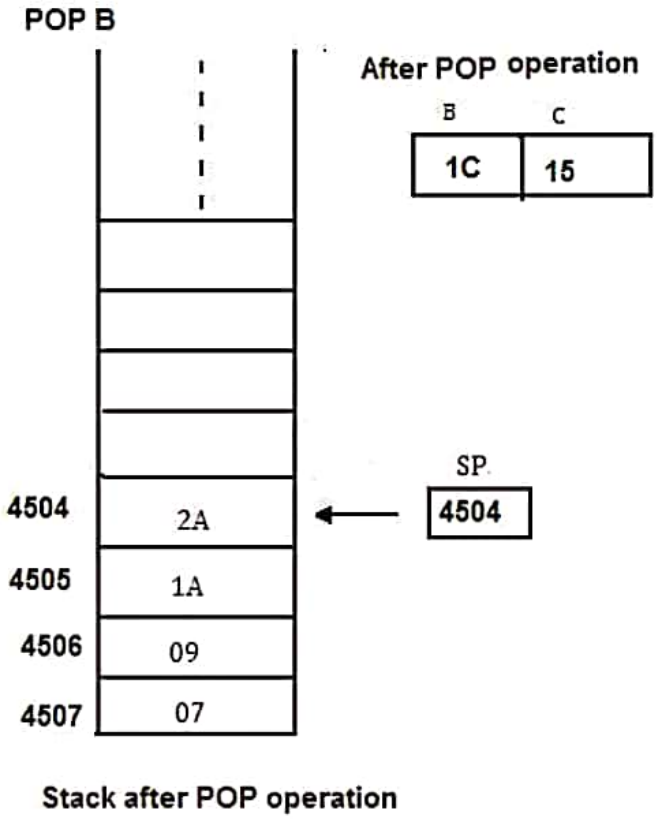


Figure 32 After POP operation the status of the stack

Before the operation the data 15 and 1C are in the locations 4502 & 4503 and after the pop operation the data is copied to B-C pair and now the SP register points to 4504 location. This is shown in Fig.3.32

Programming Example FOR PUSH & POP

Write a program to initialize the stack pointer (SP) and store the contents of the register pair H-L on stack by using PUSH instruction. Use the contents of the register pair for delay counter and at the end of the delay retrieve the contents of H-L using POP.

Memory Location	Label	Mnemonics	Operand	Comments
8000		LXI	SP, 4506 H	Initialize Stack pointer
8003		LXI	H,2565 H	
8006		PUSH	H	
8007				
.		DELAY	CALL	Push the

. . . 8.00A		. . . POP	. . . H	contents.

Subroutine: It is a set of instructions written separately from the main program to execute a function that occurs repeatedly in the main program.

For example, let us assume that a delay is needed three times in a program. Writing delay programs for three times in a main program is nothing but repetition. So, we can write a subroutine program called 'delay' and can be called any number of times we need

Similarly, in 8085 microprocessor we do not find the instructions for multiplication and division. For this purpose we write separate programs. So, in any main program if these operations are needed more than once, the entire program will become lengthy and complex. So, we write subroutine programs MUL & DIV separately from main program and use the instruction CALL MUL (or) CALL DIV in the main program. This can be done any number of times. At the end of every subroutine program there must be an instruction called 'RET'. This will take the control back to main program.

The 8085 microprocessor has two instructions to implement the subroutines. They are CALL and RET. The CALL instruction is used in the main program to call a subroutine and RET instruction is used at the end of the subroutine to return to the main program. When a subroutine is called, the contents of the program counter, which is the address of the instruction following the CALL instruction is stored on the stack and the program execution is transferred to the subroutine address. When the RET instruction is executed at the end of the subroutine, the memory address stored on the stack is retrieved and the sequence of execution is resumed in the main program.

Diagrammatic representation

Let us assume that the execution of the main program started at 8000 H. It continues until a CALL subroutine instruction at 8020 H is encountered. Then the program execution transfers to 8070 H. At the end of the subroutine 807B H. The RET instruction is present. After executing this RET, it comes back to main program at 8021 H as shown in the following Fig. 33