


Lecture-13

PROGRAM COUNTER:

This is a 16 bit register accessible to the user. It always contain the address of the next instruction to be executed in a program sequence. Thus the program counter keeps the track of the program execution up to which it is complete.

Whenever necessary, the address information available in PC shall be sent out through the address lines & BDB during T_1 timing slot. The higher order 8 bits PCH shall be sent out through $A_{15} - A_8$ address lines & the lower order 8 bits program counter shall be sent out through $AD_7 - AD_0$.lines during T_1 states since the BDB contains the lower order 8 bit address information during T_1 state an ALE pulse is also used by μp .

The above statement can be symbolically stated through macro RTL shown in the figure 22.

$$\begin{aligned} T_1 : (AD_7 - ASD_0) &\leftarrow , A_{15} - A_8 \leftarrow PCH , ALE = \text{_____} \\ T_2 : \overline{RD} &= 0, (PC) \leftarrow (PC) + 1 \end{aligned}$$
A timing diagram showing a single rectangular pulse for the ALE signal. The pulse starts at a low level, transitions to a high level at the beginning of the T1 state, and returns to a low level at the end of the T1 state. The pulse is labeled 'ALE =' in the preceding text.

Whenever the address information sent from the program counter to the external world during T_1 state, then the PC shall be incremented by 1 down the subsequence T_2 state so that program counter points to the next sequential byte. This also shown in fig.

Note: If the address information for PC has not been sent out during T_1 state to the external world, then the PC will not be incremented using T_2 state.

Whenever the address information is sent out from the program counter to the address bus (external world) then the PC is incremented by automatically during the subsequent T_2 state so that the PC points to the next sequential byte. If the data required provides instruction is of two bytes or three bytes long or it may be

the next instruction to be fetched and executed. If instructions are sequentially arranged in memory, this will guarantee that they will also be executed sequentially. Sometimes, program execution requires that non-sequential instructions executed e.g. JUMP or CALL type instructions. This requires the program counter to be loaded with an entirely new value. An 8-bit μp with a 16-bit program counter requires two data moves to completely modify the contents of the PC. If the address information from PC has not been sent out during T_1 state, the external world then PC will not be incremented during T_2 state.

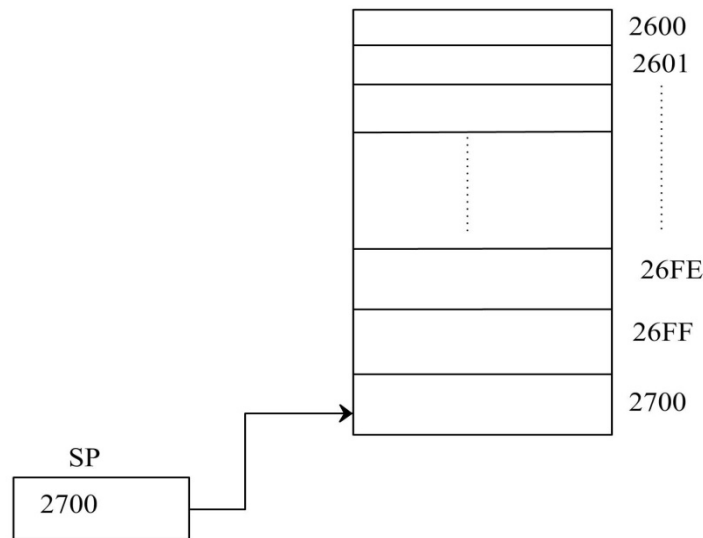
When the μp is RESET, the CPU initializes the PC to 0000 H. Therefore, the first instruction in the program should be at 0000 H in the memory address space of the CPU.

STACK POINTER REGISTER:

The stack is a storage structure. It consists of number of sequential and RAM locations in which a μp saves the internal register contents during subroutine calls and interrupts so that they will not be changed or destroyed by a subroutine.

8085 μp can address directly 64K memory locations. This is known as directly addressable memory space starting from the address 0000 H to FFFF H. this entire memory area is usually divided by the user into program area, data area and stack area. It is for the user to see that program area and data area do not overlap with that of stack memory area. The size of the stack memory area depends upon the application.

E.g. the user for a particular process control operation may decide to reserve memory space starting from 2600 to 2700 as the stack memory space. This is shown in fig.

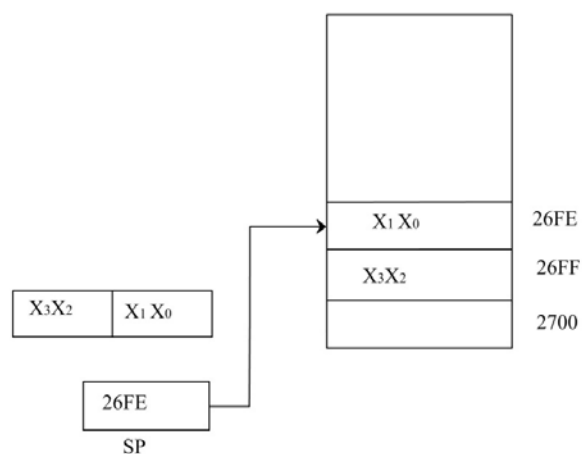


The stack pointer is a 16-bit register accessible to the user. It is required to address a memory location in the stack. It contains the address of the top of stack into which last data is written. Writing data into a stack is called a PUSH operation and reading data from a stack is called a POP operation. In the fig. shown 2700 H is known as the bottom of the stack. There is an instruction in the instruction set to initialize the stack pointer register to the bottom of the stack. This instruction is LXI SP, BADR.

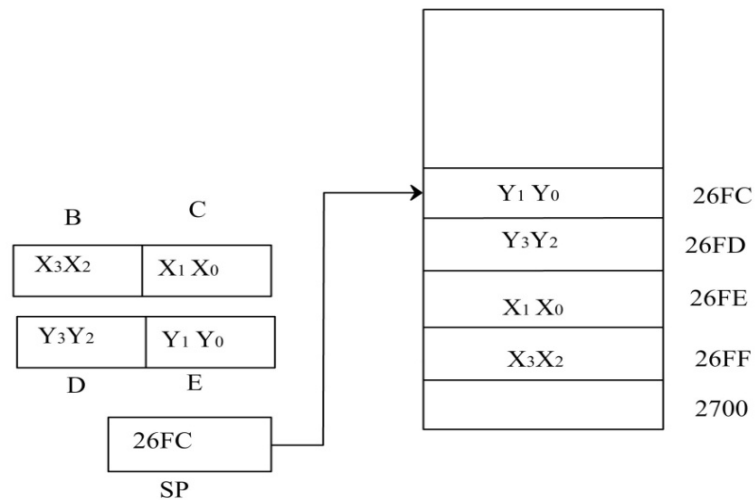
LXI is the numeric for Load immediate, BADR is a symbolic name given to the 16-bit address which is to be loaded into the stack pointer. The meaning of the instruction is to load the 16-bit of data immediately available in the instruction itself into the stack point. In this example, BADR equals 2700 H. when this instruction is executed

the situation is shown in fig. the stack pointer now points to the bottom of the stack.

Now, let us suppose that while calling a subroutine it becomes necessary to save the contents of (BC) reg pair and (DE) reg. pair as they are used in the subroutine. The process of saving the content of a reg. is known as push operation. The push operation is performed at the beginning of a subroutine to save reg. contents and the instruction for pushing the contents of the internal register is PUSH. E.g. PUSH B. The meaning of the instruction is to push the contents of BC reg. pair on to the stack so that it can be saved there till it is restored. PUSH B operation affects the stack and stack pointer as follows. Since the stack pointer always adds the address of the last byte of data pushed onto the stack, therefore when PUSH B instruction is executed, the stack pointer is decremented by 1 and the contents of the B register are copied onto the stack at that address. The stack pointer is decremented again, and the contents of the C register are copied to that address. Just after the execution of PUSH B, the situation is shown in fig.

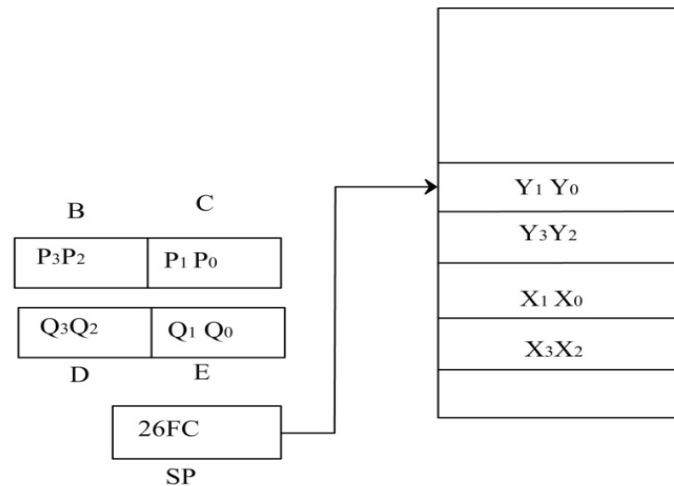


Similarly, to store the contents of (D, E) pair PUSH D instruction is used. The meaning of this instruction is push the contents of the (D,E) pair onto the stack to save them there as shown in fig. just after the execution.



Since the contents of (B, C) & (D, E) reg. pairs are stored at the top of the stack, these registers are now available for further computation in the subroutine. At a later stage of execution of the program after utilizing B, C, D, E registers, there may be a need to restore the original contents to the respective registers. E.g. at the end of the subroutine, the data is restored to the proper register.

The restoration of the contents is a READ operation from the stack and is known as POP operation. A POP register instruction copies the stored data from the stack back into the indicated register pair. Just before the execution of POP instruction, let us say the situation is as shown.



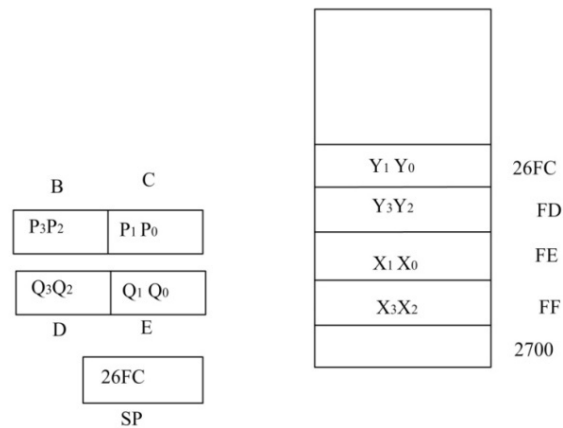
Note that regs B, C & D, E have same different contents because these registers are used in the subroutine.

To restore the contents of B, C reg pair POP B instruction is used. Whenever their instruction is executed, the contents of the top of the stack are read into the register (B, C) pair. To restore the contents of the top of the stack is read into the (D, E) reg pair. The question in which sequence these instructions are to be executed so that the contents are restored properly. The obvious sequence in POP D first & the POP B in the data must be popped off in the reverse order from which it was pushed. This type of stack is called last in first out (LIFO) memory.

Just after the execution of POP D & POP B instructions the situation is as shown in fig.

When POP D is executed the data from the top of the stack is copied to reg. E, data pointer is incremented by 1, then the next byte of the

saved data is copied from the stack to the reg. D, and SP is further incremented by 1.



This is similar to previous one but now some data has been stored in the stack area but these are irrelevant anyway. They will be destroyed during the next PUSH operation on the stack.

From the above discussion, following points emerge:

1. The stack pointer always points to the top of the stack up to which it is full with relevant data.
2. Storing or saving the data on stack is known as PUSH operation.
3. The restoring or reading data from the stack onto certain internal registers are known as POP instructions.
4. The stack operates on Last in first out basis.
5. The stack pointer can be initialized to the bottom of the stack but bottom of the stack cannot be utilized to store any useful data.
6. It is for the user to see that the program area does not overlap with stack area.