CHAPTER

8

Stack Control and Branching Group

8.1 Introduction to Stack

PTU - Dec. 2005

The stack is a reserved area of the memory in RAM where temporary information may be stored. An 8-bit stack pointer is used to hold the address of the most recent stack entry. This location which has the most recent entry is called as the **top of the stack.**

When the information is written on the stack, the operation is called **PUSH.** When the information is read from the stack, the operation is called **POP.** The stack works on the principle of **Last in First Out or First in Last Out.**

The microprocessor stores the information/data like stacking plates.

Fig. 8.1.1 shows stacked plates. If we want to remove the first stack plate, then we will have to remove all the plates above the first i.e. we have to remove the fourth plate, third plate, second plate and then finally the first plate. This indicates that the first plate pushed onto the stack is the last one to be popped from the stack. This operation is called as First In Last Out (FILO).

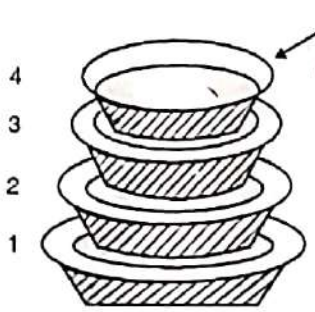

Fig. 8.1.1 : Stacked plates

The stack is implemented with the help of special memory pointer register called as the **stack pointer.** It is of 8 bit. The stack pointer's contents are automatically adjusted to point to the stack. The memory location that is currently pointed by the stack pointer is called as top of stack. As shown in Fig. 8.1.1 the 4th stacked plate represents top of stack.

When data is to be stored on the stack, the SP increments before storing the data on stack and when the data is to be retrieved/popped from the stack the SP decrements to point next available byte of stored data.

The stack array can reside anywhere in the on-chip RAM. However, its height is limited to the size of internal RAM.

Hence in short,

a) Stack area is usually defined in RAM ONLY.
b) Stack is used to store information (data) temporarily.
c) Stack is a part of memory defined in program by THE USER.
h) STACK, the name it self, informs about the way data is stored i.e. stack of data.

8.2 Stack Related Instructions

stack related instructions are:

PUSH R_P 2) POP R_P 3) SPHL 4) XTHL

us study these instructions.

8.2.1 PUSH R_p

Mnemonic	PUSH R_P
Operation	SP = SP – 1 SP = Higher order R_P SP = SP – 1 SP = Lower order R_P
No. of Bytes	1 byte
Machine Cycles	3 (OF + MW + MW)

Algorithm	SP ← SP – 1 (SP) ← Higher byte of R_P SP ← SP – 1 (SP) ← Lower byte of R_P
Flags	No flags are affected.
Addr. Mode	Register addressing mode
T-states	12 (6 + 3 + 3)

Description	**Push the contents of register pair on to the stack.** • This instruction is used to write 16 bit data on the stack. • When this instruction is executed the contents of the specified register pair are copied on the stack in the following sequence. (a) The stack pointer is decremented by one and the contents of higher order register of the specified register pair are copied to the memory location pointed the stack pointer. (b) The stack pointer is again decremented by 1 and the contents of lower order of register pair are copied to memory location pointed by the stack pointer. • The register pair R_P can be any 16 bit register pair like BC, DE, HL or PSW (program status word). • The PSW is program status word with A register as higher order register and flag register as low order register. • Only higher order register is to be specified within the instruction.
Example PUSH B	• Let BC = 3010H, and the stack pointer is initialized at FFFFH. Now if the instruction PUSH B is executed, then initially the stack pointer will decrement by 1 i.e. (SP = FFFE H). The contents of register B (30 H) will be copied to the memory location FFFE H. The stack pointer is then again decremented by one (SP = FFFD H). The contents of register C (10 H) will be copied to this memory location. Fig. 8.2.1 shows the PUSH B operation.

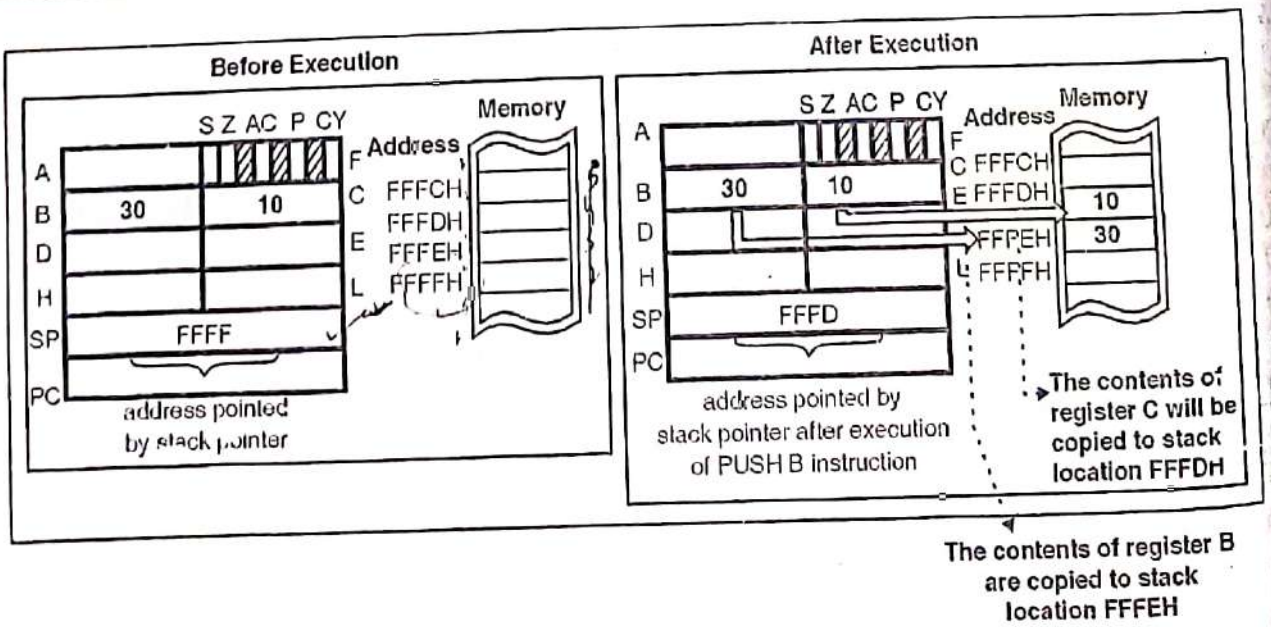

Fig. 8.2.1 : PUSH B

ote : In case of PUSH PSW the sequence of operation is

i) The stack pointer is decremented by one and the contents of higher order register i.e. register A are copied to that location.

ii) The stack pointer is again decremented by 1 and the contents of lower order register i.e. flag register F are copied to that location.

Fig.8.2.2 shows the timing diagram for the PUSH R_p instruction.

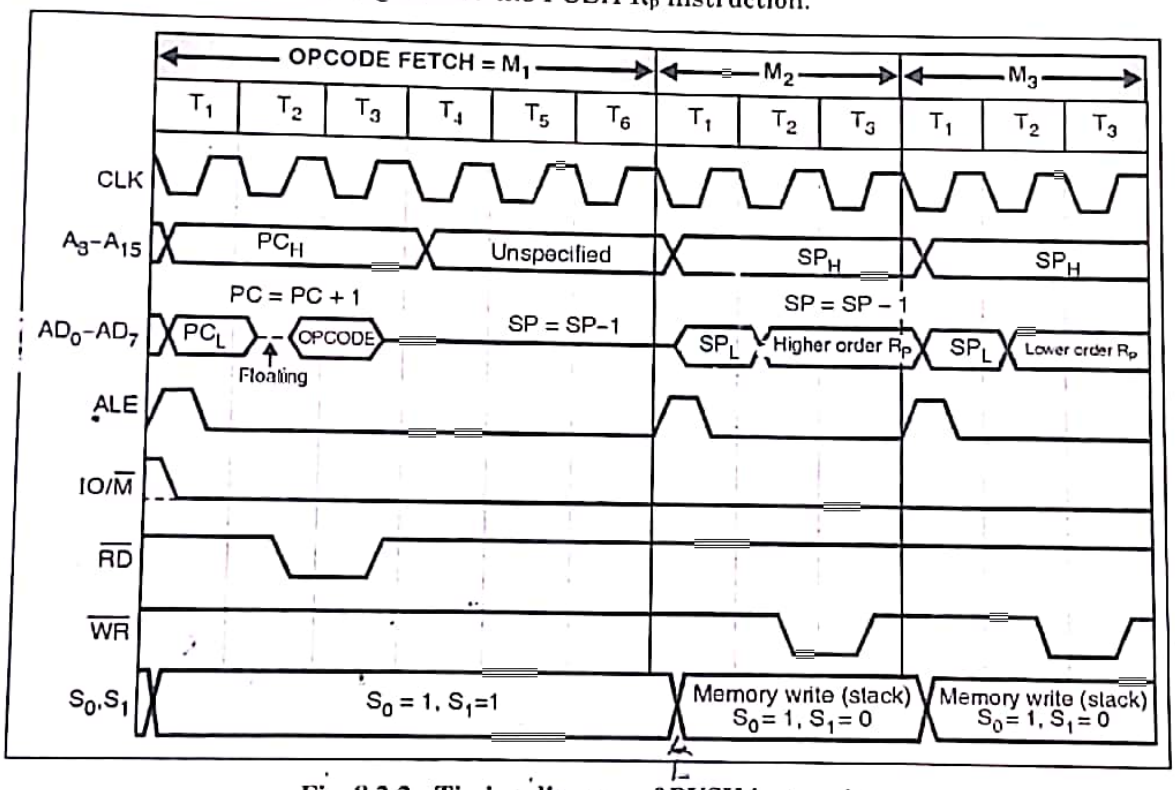

Fig. 8.2.2 : Timing diagram of PUSH instruction

The PUSH R_p instruction stores the contents of the specified register pair on to stack. To perform this 8085 requires two memory write machine cycles and an opcode fetch cycle. So in all 3 machine cycles are required.

(1) **OPCODE fetch :** The program counter places address on the lower order address bus and the higher order address bus. This cycle is used to read OPCODE of PUSH instruction. The address for this machine cycle is given by PC. The PC is then incremented by 1.This machine cycle requires 6 T-states. During T_5 and T_6 states the microprocessor performs internal operation of decrementing SP register contents by 1.

(2) **Memory write :** This machine cycle is used to store data from higher order register on to stack. Address for this is given by SP. The contents are stored and SP is again decremented by 1.

(3) **Memory write :** This machine cycle is used to store data from lower order register on to stack. Address for this is given by SP. The contents are stored and microprocessor will start OPCODE fetch machine cycle for next instruction.

2 POP R_p

mnonic	POP R_p
ration	Lower order R_p = (SP) SP = SP + 1 Higher order R_p = (SP) SP = SP + 1

Algorithm	Lower byte of $R_p \leftarrow$ (SP) SP $\leftarrow$ SP + 1 Higher byte of $R_p \leftarrow$ (SP) SP $\leftarrow$ SP + 1
Flags	No flags are modified.

No. of Bytes	1 byte
Machine Cycles	3 (OF + MR + MR)

Addr. Mode	Register addressing mode
T-states	10 (4 + 3 + 3)

Description	**POP off stack contents to register pair.** • When this instruction is executed, the contents of the memory location pointed by the stack pointer (SP) register are copied to the low order byte of register pair. • The SP is incremented by one and the contents of that memory location are copied to the higher order byte of register pair. • The SP is again incremented by 1. • The process of POP is exactly opposite to PUSH operation. So the contents stored by PUSH are taken back using POP instructions. • The examples of R_P are BC, DE, HL and PSW. • Only high order register is specified for register pair.
Example POP B	• Let BC = 3010 H, stack pointer = FFFD H at memory location FFFD H : 10 H is stored and at memory location FFFE H : 30 H is stored and the instruction POP B is executed. The contents of stack location FFFDH are popped off to the C register. The stack pointer will increment by one. The contents of this location i.e. FFFEH are popped off the B register. Then the stack pointer will again increment by 1 (i.e. SP = FFFF H). Fig. 8.2.3 shows this.

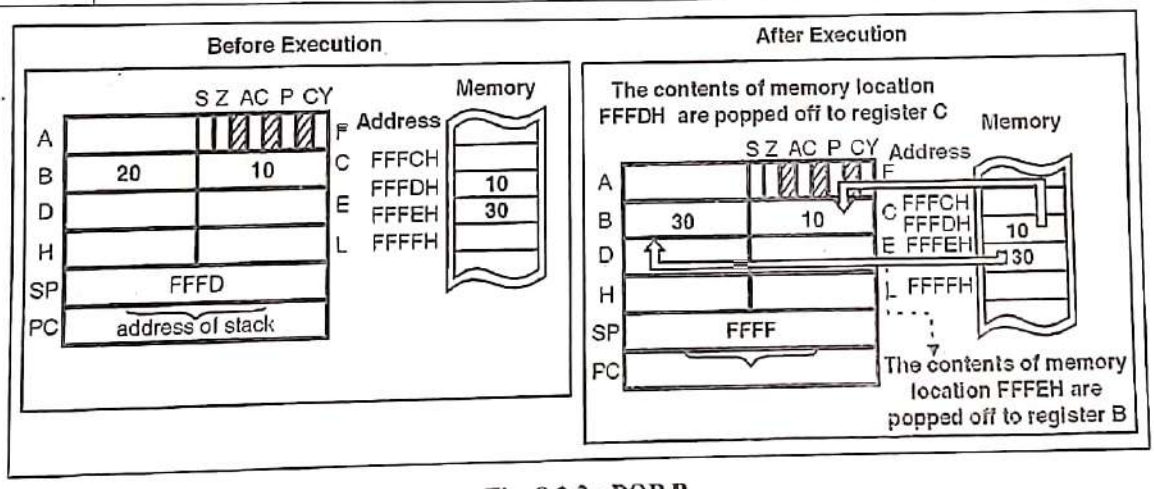

Fig. 8.2.3 : POP B

- To perform this operation 8085 requires one opcode fetch cycle and two memory read machine cycl So in all 3 machine cycles are required. Fig.8.2.4 shows the timing diagram of the POP R_p instructio

(1) **OPCODE fetch :** The program counter places address on the lower order address bus and the higher order address bus. This cycle is used to read OPCODE of POP instruction. The addr for this machine cycle is given by PC. The PC is then incremented by 1.

(2) **Memory read :** The stack pointer gives address on the lower order address bus and the hig order address bus. The lower order byte of the address specified is read into the speci register pair. The SP is then incremented by one.

(3) **Memory read :** The stack pointer gives address on the lower order address bus and the hi order address bus. The higher order byte of the address specified is read into the spec register pair. The SP is then incremented by one.

- After third cycle, microprocessor will start OPCODE fetch machine cycle for next instruction.

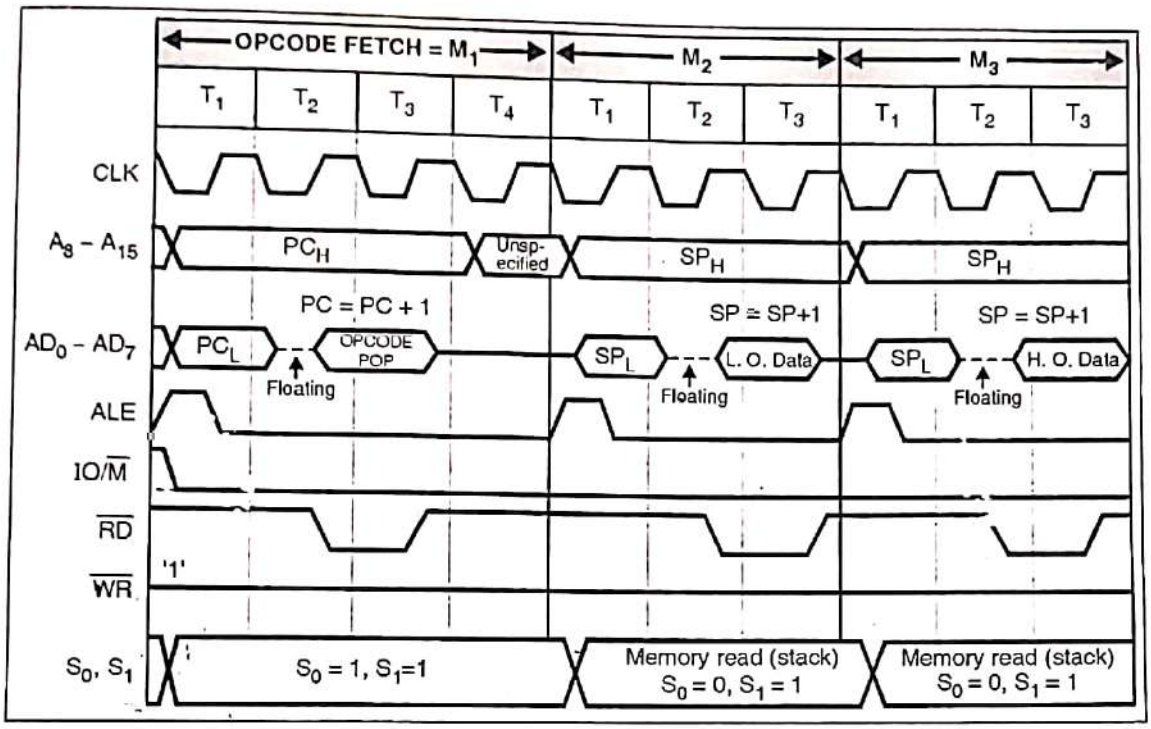

Fig. 8.2.4 : Timing diagram of POP instruction

3 SPHL

stack pointer HL

emonic	SPHL.
eration	SP = HL
of Bytes	1 byte
chine Cycles	1 (OF)

Algorithm	SP ← HL
Flags	No flags are affected.
Addr. Mode	Implied addressing mode
T-states	6

scription	**Load stack pointer with HL register pair contents.** • When this instruction is executed, the contents of HL register pair are transferred to the stack pointer. • The contents of H register are copied to higher order byte of stack pointer and contents of L register are copied to the lower byte of stack pointer.
mple	• Let HL = ABCD H, Stack Pointer = FFFE H and the instruction SPHL is executed. • The contents of stack pointer after the execution of this instruction will be SP = ABCD H as shown in Fig. 8.2.5.

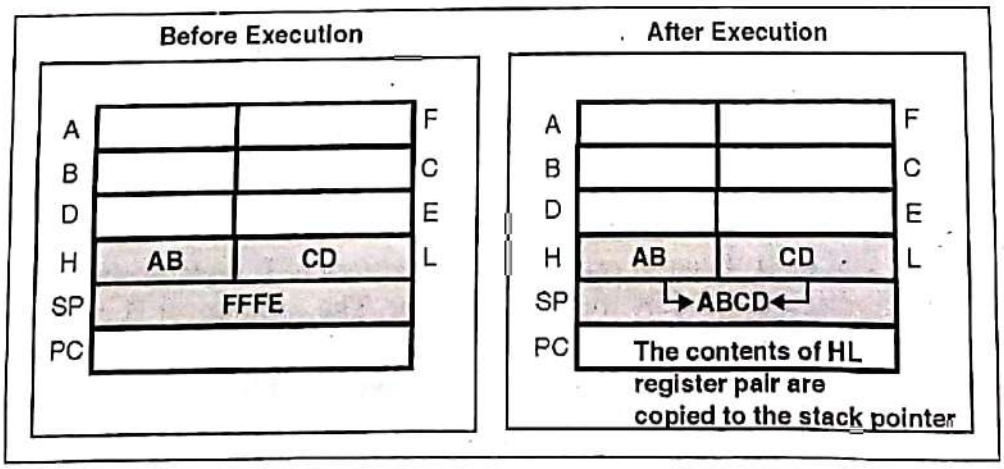

Fig. 8.2.5 : SPHL

- This instruction also requires only **opcode fetch machine cycle.**
- The microprocessor will read the opcode of the instruction from the memory location that is addre - ed and then decodes it.
- The program counter holds the address on the low and high order address bus. The PC is increme - ed by 1.
- The opcode fetch machine cycle requires 6T states.

Fig. 8.2.6 shows the timing diagram of SPHL instruction.

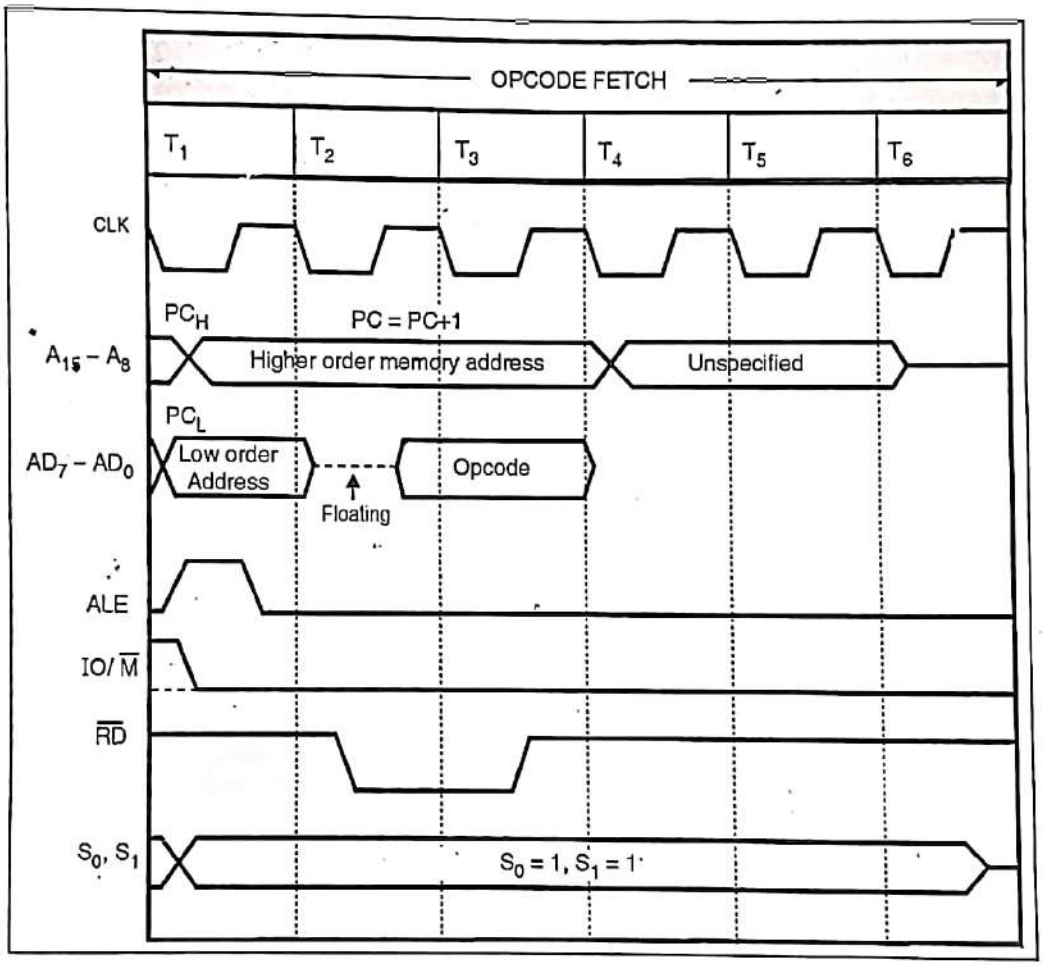

Fig. 8.2.6 : Timing diagram of SPHL

8.2.4 XTHL

Mnemonic	XTHL.
Operation	L ⇌ (SP) H ⇌ (SP + 1)
No. of Bytes	1 byte
Machine Cycles	5 (OF + MR + MR + MW + MW)

Algorithm	L ← SP H ← (SP + 1)
Flags	No flags are modified.
Addr. Mode	Implied addressing mode
T-states	16 (4 + 3 + 3 + 3 + 3)

Description	**Exchange HL with top of stack** • When this instruction is executed the contents of L register are exchanged wit - the stack location pointed by the stack pointer. The contents of H register are exch - ged with the next stack location. • The contents of the stack pointer register are not altered.

Example XTHL

- Let H = 01 H, L = 20 H, SP = FFFD H and at memory location FFFD H : 05 H is stored. At memory location FFFEH : 06 H is stored. After the execution of instruction XTHL, the contents of L register (20) H will be exchanged with contents of memory location FFFDH and the contents of H register (01H) will be exchanged with the contents of memory location FFFEH.
- Fig 8.2.7 shows the timing diagram of the XTHL instruction.

To perform this, microprocessor has to first read top of stack contents in to temporary registers W and Z, then it stores contents of H and L at top of stack. It then transfer W and Z contents to H and L. It has 5 machine cycles.

(1) **OPCODE fetch :** It is used to read OPCODE of XTHL instruction. Address is given by PC and then PC is incremented by 1. T_5 and T_6 states are not required.

(2) **Memory read :** It is used to read contents of stack. Address for this is given by SP register. The SP is incremented by 1.

(3) **Memory read :** It is used to read contents of stack. Address for this is given by SP register The stack pointer is incremented by 1.

(4) **Memory write :** The stack pointer is decremented by 1. It is used to store data from H register on to stack. Address for this is given by SP register.

(5) **Memory write :** The stack pointer is decremented by 1. It is used to store contents of L register on to stack. Address for this is given by SP register.

After memory write the contents of temporary register are transferred to H and L registers and microprocessor will start OPCODE fetch machine cycle for next instruction.

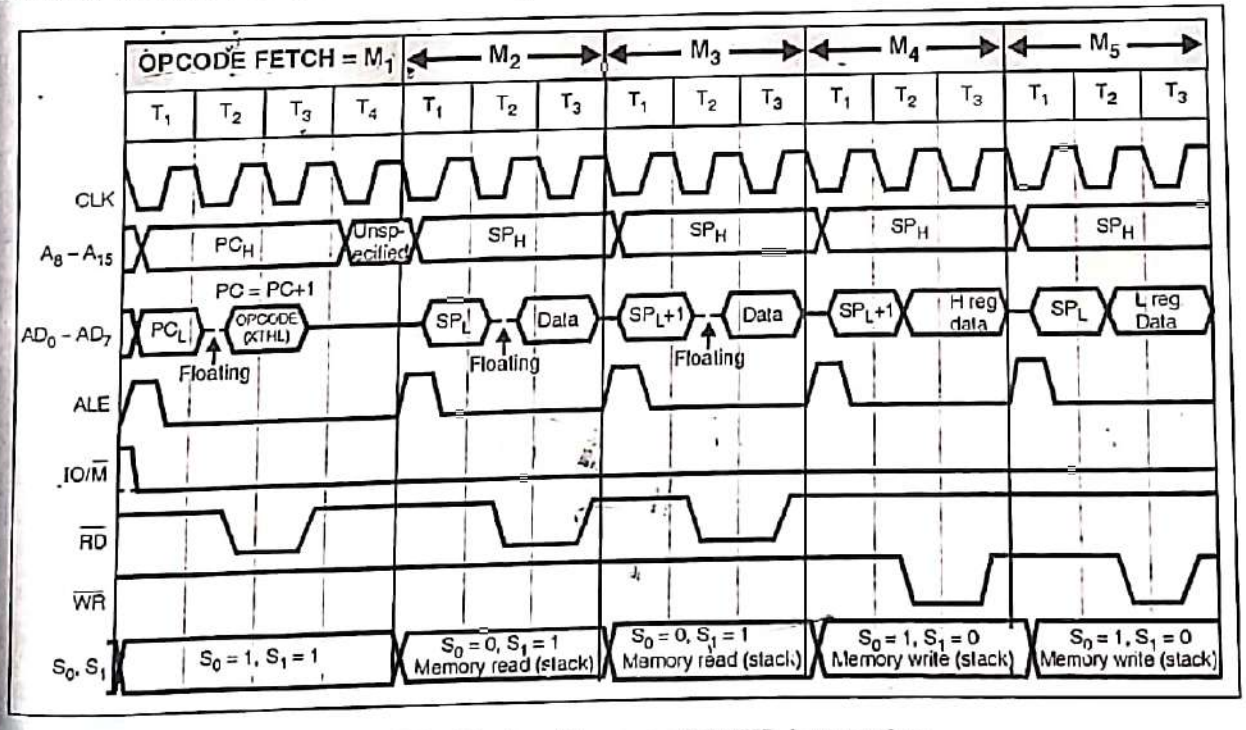

Fig. 8.2.7 : Timing diagram of XTHL instruction

8.5 LXI SP, data

This instruction initializes the stack pointer with 16 bit address.

The stack point can be initialized by two method :

(i) Direct method : LXI SP, data (16 bit).

(ii) Indirect method : LXI H, data (16 bit).
SPHL

Generally the stack pointer is initialized by direct method. If the user wants to set the stack pointer to a value that is found out from the program, then in such cases the indirect method is used.

The timing diagram is shown in Fig. 5.7.3.

8.2.6 INX SP

This instruction increments the Stack Pointer by 1. (For details refer Chapter 5)

8.2.7 DAD SP

This instruction adds the contents of Stack Pointer with the contents of HL register pair. The resu
addition is store into the HL register pair.

8.2.8 DCX SP

This instruction decrements the stack pointer by 1 (For details refer Chapter 5).