1. Stack

- Stack is a group of memory location in the R/W memory that is used for temporary storage of binary information during execution of a program.
- The starting memory location of the stack is defined in program and space is reserved usually at the high end of memory map.
- The beginning of the stack is defined in the program by using instruction **LXI SP, 16-bit memory address**. Which loads a 16-bit memory address in stack pointer register of microprocessor.
- Once stack location is defined storing of data bytes begins at the memory address that is one less then address in stack pointer register. LXI SP, 2099h the storing of data bytes begins at 2098H and continues in reversed numerical order.

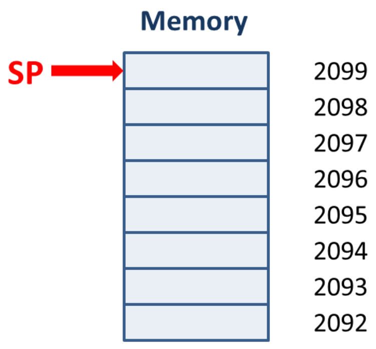


Fig. Stack

- Data bytes in register pair of microprocessor can be stored on the stack in reverse order by using the PUSH instruction.
- PUSH B instruction sore data of register pair BC on sack.

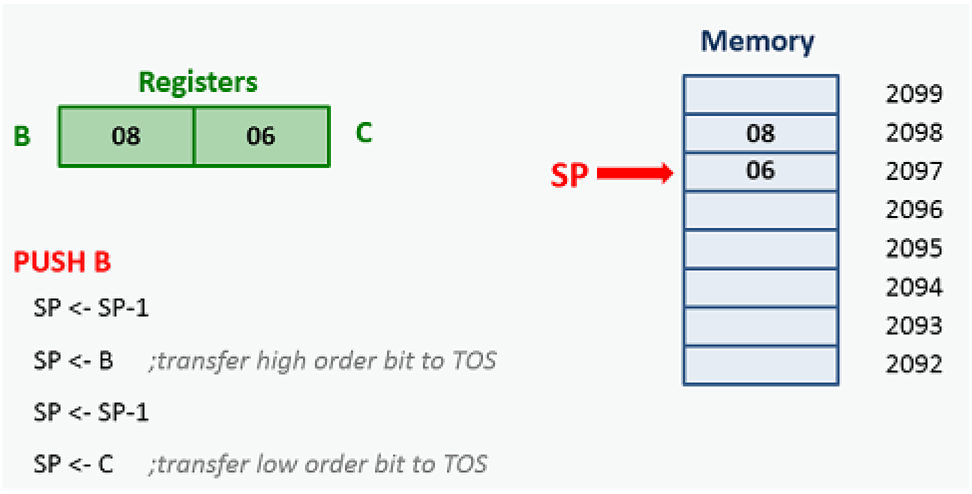


Fig. PUSH operation on stack

- Data bytes can be transferred from the stack to respective registers by using instruction POP.

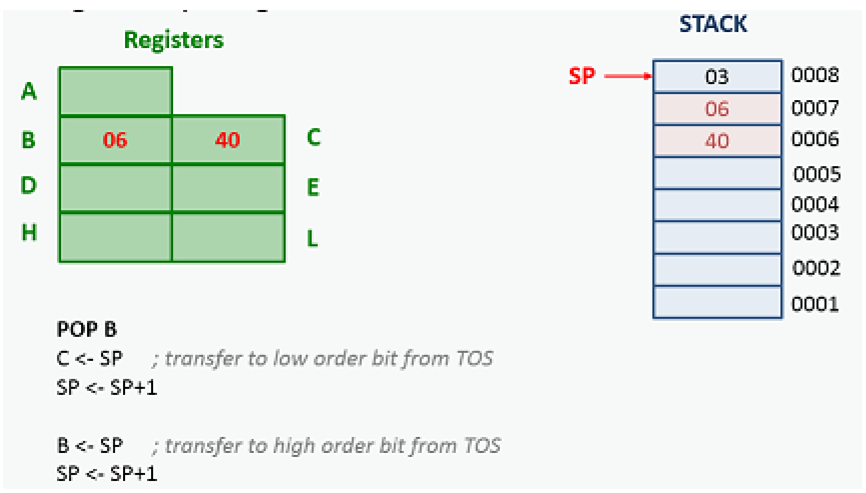


Fig. POP operation on stack

Instruction necessary for stack in 8085

LXI SP, 2095	Load the stack pointer register with a 16-bit address.
PUSH B/D/H	It copies contents of B-C/D-E/H-L register pair on the stack.
PUSH PSW	Operand PSW represents Program status word meaning contents of accumulator and flags.
POP B/D/H	It copies content of top two memory locations of the stack in to specified register pair.
POP PSW	It copies content of top two memory locations of the stack in to B-C accumulator and flags respectively.

2. Subroutine

- A subroutine is a group of instruction that performs a subtask of repeated occurrence.
- A subroutine can be used repeatedly in different locations of the program.

Advantage of using Subroutine

- Rather than repeat the same instructions several times, they can be grouped into a subroutine that is called from the different locations.

Where to write Subroutine?

- In Assembly language, a subroutine can exist anywhere in the code.
- However, it is customary to place subroutines separately from the main program.

Instructions for dealing with subroutines in 8085.

- The **CALL** instruction is used to redirect program execution to the subroutine.
  - When CALL instruction is fetched, the Microprocessor knows that the next two **new** Memory location contains 16bit subroutine address.
  - Microprocessor Reads the subroutine address from the next two memory location and stores the higher order 8bit of the address in the **W** register and stores the lower order 8bit of the address in the **Z** register.
  - Push the **Older** address of the instruction immediately following the CALL onto the stack [Return address]
  - Loads the program counter (**PC**) with the **new** 16-bit address supplied with the CALL instruction from **WZ** register.
- The **RET** instruction is used to return.

- Number of PUSH and POP instruction used in the subroutine must be same, otherwise, RET instruction will pick wrong value of the return address from the stack and program will fail.

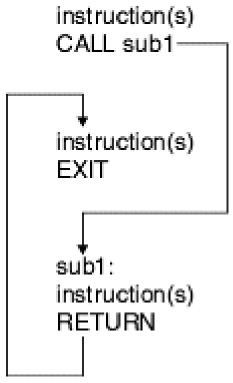


Fig. Subroutine

- Example: write ALP to add two numbers using call and subroutine.

```
LXI H 2000 ; Load memory address of operand
MOV B M ; Store first operand in register B
INX H ;Increment H-L pair
MOV A M ; Store second operand in register A
CALL ADDITION ; Call subroutine ADDITION
STA 3000 ; Store answer
HLT

ADDITION: ADD B ; Add A and B
RET ; Return
```

Conditional call and return instruction available in 8085

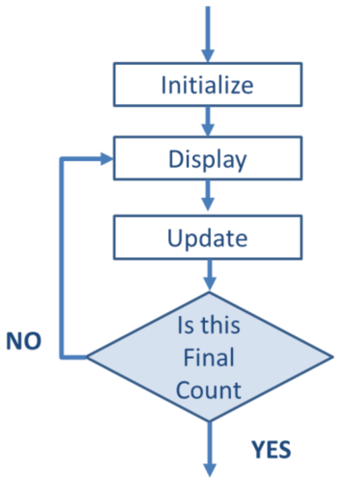
CC 16-bit address	Call on Carry, Flag Status: CY=1
CNC 16-bit address	Call on no Carry, Flag Status: CY=0
CP 16-bit address	Call on positive, Flag Status: S=0
CM 16-bit address	Call on minus, Flag Status: S=1
CZ 16-bit address	Call on zero, Flag Status: Z=1
CNZ 16-bit address	Call on no zero, Flag Status: Z=0
CPE 16-bit address	Call on parity even, Flag Status: P=1
CPO 16-bit address	Call on parity odd, Flag Status: P=0
RC	Return on Carry, Flag Status: CY=1
RNC	Return on no Carry, Flag Status: CY=0
RP	Return on positive, Flag Status: S=0
RM	Return on minus, Flag Status: S=1
RZ	Return on zero, Flag Status: Z=1
RNZ	Return on no zero, Flag Status: Z=0
RPE	Return on parity even, Flag Status: P=1
RPO	Return on parity odd, Flag Status: P=0

3. Applications of Counters and Time Delays

1. Traffic Signal
2. Digital Clocks
3. Process Control
4. Serial data transfer

4. Counters

- A counter is designed simply by loading appropriate number into one of the registers and using INR or DNR instructions.
- Loop is established to update the count.
- Each count is checked to determine whether it has reached final number; if not, the loop is repeated.



```
1. MVI C,05
2. LOOP: MOV A,C
3.       OUT 01
4.       DCR C
5.     JNZ LOOP
6. HLT
```

Fig. Counter

5. Time Delay

- Each instruction passes through different combinations of Fetch, Memory Read, and Memory Write cycles.
- Knowing the combinations of cycles, one can calculate how long such an instruction would require to complete.
- It is counted in terms of number of T–states required.
- Calculating this time we generate require software delay.

Time Delay Using Single Register

Label	Opcode	Operand	Comment	T-states
	MVI	C,05h	; Load Counter	**7**
LOOP:	DCR	C	; Decrement Counter	**4**
	JNZ	LOOP	; Jump back to Decr. C	**10/7**

MVI C 05 Mchine Cycle: F + R = 2 T-States: 4T + 3T = 7T	DCR C Mchine Cycle: F = 1 T-States: 4T = 4T	JNZ LOOP (true) Mchine Cycle: F + R + R = 3 T-States: 4T + 3T + 3T = 10T	JNZ LOOP (false) Mchine Cycle: F + R = 3 T-States: 4T + 3T = 7T

- Instruction MVI C, 05h requires 7 T-States to execute. Assuming, 8085 Microprocessor with 2MHz clock frequency. How much time it will take to execute above instruction?
  Clock frequency of the system (f) = 2 MHz
  Clock period (T) = 1/f = ½ * 10-6 = 0.5 µs
  Time to execute MVI = 7 T-states * 0.5 µs
  = 3.5 µs
- Now to calculate time delay in loop, we must account for the T-states required for each instruction, and for the number of times instructions are executed in the loop. There for the next two instructions:

DCR:	4 T-States
JNZ:	10 T-States
	14 T-States

- Here, the loop is repeated for 5 times.
- Time delay in loop T_L with 2MHz clock frequency is calculated as:
  T_L= T * Loop T-sates * N_{10} -----------------(1)
  T_L : Time Delay in Loop
  T : Clock Frequency
  N_{10} : Equivalent decimal number of hexadecimal count loaded in the delay register.
- Substituting value in equation (1)
  T_L= ($0.5 * 10^{-6} * 14 * 5$)
  = 35 µs
- If we want to calculate delay more accurately, we need to accurately calculate execution of JNZ instruction i.e
  If **JNZ = true**, then **T-States = 10**
  Else if **JNZ =false**, then **T-States = 7**
- Delay generated by last clock cycle:
  = 3T * Clock Period
  = 3T * ($1/2 * 10^{-6}$)
  = 1.5 µs
- Now, the accurate loop delay is:
  T_{LA}=T_L - Delay generated by last clock cycle
  T_{LA}= 35 µs - 1.5 µs
  T_{LA}= 33.5 µs
- Now, to calculate total time delay
  Total Delay = Time taken to execute instruction outside loop + Time taken to execute loop instructions
  $T_D = T_O + T_{LA}$
  = (7 * 0.5 µs) + 33.5 µs
  = 3.5 µs + 33.5 µs
  = 37 µs
- In most of the case we are given time delay and need to find value of the counter register which decide number of times loop execute.
- For example: write ALP to generate 37 µs delay given that clock frequency if 2 MHz.
- Single register loop can generate small delay only for large delay we use other technique.

Time Delay Using a Register Pair

- Time delay can be considerably increased by setting a loop and using a register pair with a 16-bit number (FFFF h).
- A 16-bit is decremented by using DCX instruction.
- Problem with DCX instruction is DCX instruction doesn't set **Zero** flag.
- Without test flag, Jump instruction can't check desired conditions.
- Additional technique must be used to set Zero flag.

Label	Opcode	Operand	Comment	T-states
	LXI	B,2384 h	; Load BC with 16-bit counter	**10**
LOOP:	DCX	B	; Decrement BC by 1	**6**
	MOV	A, C	; Place contents of C in A	**4**
	ORA	B	; OR B with C to set Zero flag	**4**
	JNZ	LOOP	; if result not equal to 0, 10/7 jump back to loop	**10/7**

- Here the loop includes four instruction:
  Total T-States = 6T + 4T + 4T + 10T
  = 24 T-states
- The loop is repeated for 2384 h times.
- Converting $(2384)_{16}$ into decimal.
  2384 h = $(2 * 16^3) + (3 * 16^2) + (8 * 16^1) + (4 * 16^0)$
  = 8192 + 768 + 128 + 4 = **9092**
- Clock frequency of the system (f)= 2 MHz
- Clock period (T) = 1/f = ½ * 10^{-6} = 0.5 μs
- Now, to find delay in the loop
  T_L= T * Loop T-sates * N_{10}
  = 0.5 * 24 * 9092
  = 109104 μs = 109 ms (without adjusting last cycle)

Time Delay Using a LOOP within a LOOP

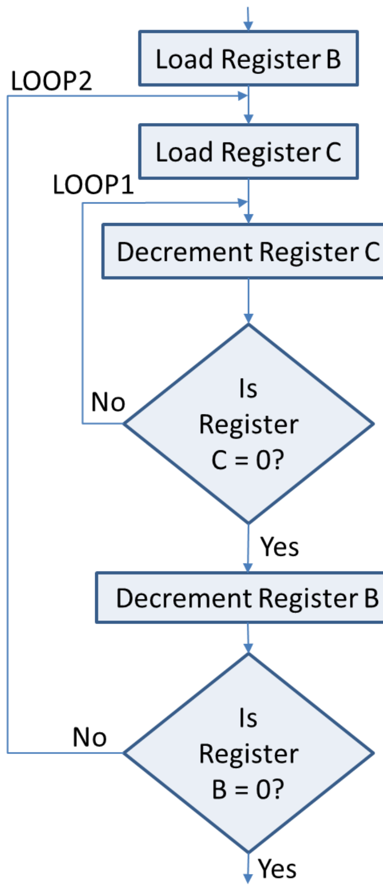


Fig. Time Delay Using a LOOP within a LOOP

Label	Opcode	Operand	T-states
	MVI	B,38h	7T
LOOP2:	MVI	C,FFh	7T
LOOP1:	DCR	C	4T
	JNZ	LOOP1	10/7 T
	DCR	B	4T
	JNZ	LOOP2	10/7 T

- Calculating delay of inner LOOP1: T_{L1}

  T_L= T * Loop T-states * N_{10}

  = 0.5 * 14* 255

  = 1785 µs = 1.8 ms

  T_{L1}= TL – (3T states* clock period)

  = 1785 – (3 * ½ * 10^{-6})

  = 1785-1.5=**1783.5 µs**
- Now, Calculating delay of outer LOOP2: T_{L2}
- Counter B : $(38)_{16}$ = **$(56)_{10}$** So loop2 is executed for 56 times.

  T-States = 7 + 4 + 10 = 21 T-States

  T_{L2} = 56 (T_{L1} + 21 T-States * 0.5)

  = 56(1783.5 µs + 10.5)

= 100464 µs

T_{L2} = 100.46 ms

Disadvantage of using software delay

- Accuracy of time delay depends on the accuracy of system clock.
- The Microprocessor is occupied simply in a waiting loop; otherwise it could be employed to perform other functions.
- The task of calculating accurate time delays is tedious.
- In real time applications timers (integrated timer circuit) are commonly used.
- Intel 8254 is a programmable timer chip that can be interfaced with microprocessor to provide timing accuracy.
- The disadvantage of using hardware chip include the additional expense and the need for extra chip in the system.

6. Counter design with time delay

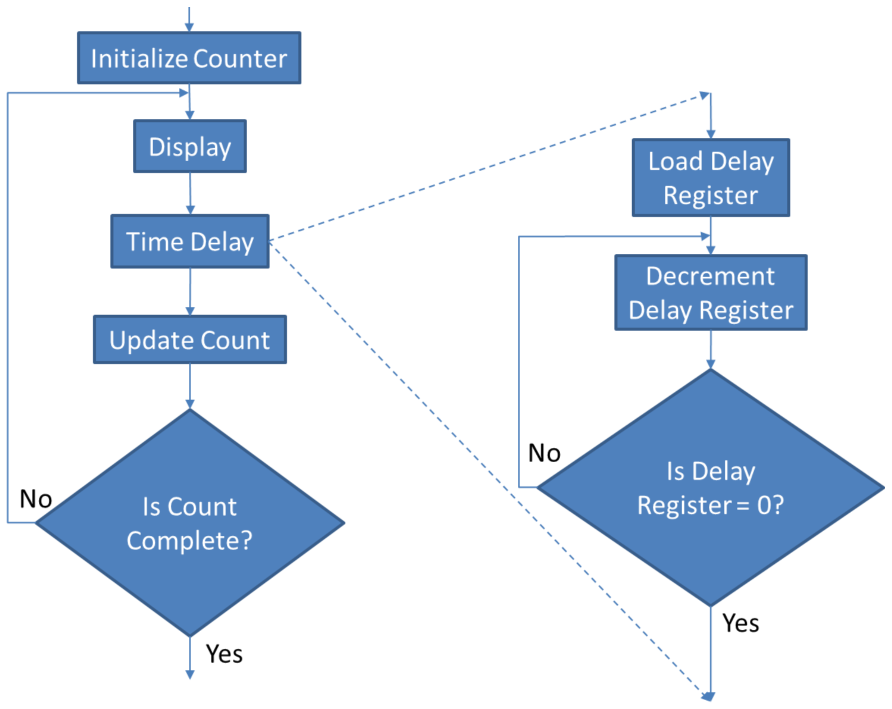


Fig. 6. Counter design with time delay

- It is combination of counter and time delay.
- I consist delay loop within counter program.

7. Hexadecimal counter program

- Write a program to count continuously in hexadecimal from **FFh** to **00h** with **0.5 µs** clock period. Use register C to set up 1 ms delay between each count and display the number at one of the output port.
- **Given:**
- Counter= FF h
- Clock Period T=0.5 µs

- Total Delay = 1ms
- **Output:**
- To find value of delay counter
- **Program**

```
MVI B,FF
LOOP:MOV A,B
    OUT 01
    MVI C, COUNT; need to calculate delay count
    DELAY: DCR C
            JNZ DELAY
    DCR B
    JNZ LOOP
HLT
```

- Calculate Delay for Internal Loop
  TI = T-States * Clock Period * COUNT
  = 14 * 0.5 * 10-6 * COUNT
  TI = (7.0 * 10-6)* COUNT
- Calculate Delay for Outer Loop:
  TO = T-States * Clock Period
  = 35 * 0.5 * 10-6
  TO = 17.5 ?s
- Calculate Total Time Delay:
  TD = TO + TL
  1 ms = 17.5 * 10-6 + (7.0 * 10-6)* COUNT
  1 * 10-3 = 17.5 * 10-6 + (7.0 * 10-6)* COUNT
  COUNT="1 ∗ 10−3 − 17.5 ∗ 10−6" /"7.0 ∗ 10−6"
  COUNT= $(140)_{10}$ = $(8C)_{16}$

8. 0-9 up/down counter program

- Write an 8085 assembly language program to generate a decimal counter (which counts 0 to 9 continuously) with a one second delay in between. The counter should reset itself to zero and repeat continuously. Assume a crystal frequency of 1MHz.
- **Program**

```
    START: MVI B,00H
    DISPLAY: OUT 01
            LXI H, COUNT
    LOOP: DCX H
            MOV A, L
            ORA H
            JNZ LOOP
            INR B
            MOV A,B
            CPI 0A
            JNZ DISPLAY
```

```
JZ START
```

9. Code Conversion

Two Digit BCD Number to Binary Number

1. Initialize memory pointer to given address (2000).
2. Get the Most Significant Digit (MSD).
3. Multiply the MSD by ten using repeated addition.
4. Add the Least Significant Digit (LSD) to the result obtained in previous step.
5. Store the HEX data in Memory.

- **Program**

```
LXI H 2000
MOV C M
MOV A C
ANI 0F ; AND operation with 0F (00001111)
MOV E A
MOV A C
ANI F0 ; AND operation with F0 (11110000)
JZ SB1 ; If zero skip further process and directly add LSD
RRC ; Rotate 4 times right
RRC
RRC
RRC
MOV D A
MVI A 00
L1: ADI 0A ; Loop L1 multiply MSD with 10
DCR D
JNZ L1
SB1: ADD E
STA 3000 ; Store result
HLT
```

8-bit Binary Number to Decimal Number

1. Load the binary data in accumulator
2. Compare ‘A’ with 64 (Dicimal 100) if cy = 01, go step 5 otherwise next step
3. Subtract 64H from ‘A’ register
4. Increment counter 1 register
5. Go to step 2
6. Compare the register ‘A’ with ‘0A’ (Dicimal 10), if cy=1, go to step 10, otherwise next step
7. Subtract 0AH from ‘A’ register
8. Increment Counter 2 register
9. Go to step 6
10. Combine the units and tens to from 8 bit result
11. Save the units, tens and hundred’s in memory
12. Stop the program execution

- **Program**

```
MVI B 00
```

```
LDA 2000
LOOP1: CPI 64 ; Compare with 64H
JC NEXT1 : If A is less than 64H then jump on NEXT1
SUI 64 ; subtract 64H
INR B
JMP LOOP1
NEXT1:  LXI H 2001
MOV M B ; Store MSD into memory
MVI B 00
LOOP2: CPI 0A ; Compare with 0AH
JC NEXT2 ; If A is less than 0AH then jump on NEXT2
SUI 0A ; subtract 0AH
INR B
JMP LOOP2
NEXT2:  MOV D A
MOV A B
RLC
RLC
RLC
RLC
ADD D
STA 2002 ; Store packed number formed with two leas significant digit
HLT
```

Binary Number to ASCII Number

- Load the given data in A - register and move to B - register
- Mask the upper nibble of the Binary decimal number in A - register
- Call subroutine to get ASCII of lower nibble
- Store it in memory
- Move B - register to A - register and mask the lower nibble
- Rotate the upper nibble to lower nibble position
- Call subroutine to get ASCII of upper nibble
- Store it in memory
- Terminate the program.

```
LDA 5000 Get Binary Data
    MOV B, A
    ANI 0F          ; Mask Upper Nibble
    CALL SUB1       ; Get ASCII code for upper nibble
    STA 5001
    MOV A, B
    ANI F0          ; Mask Lower Nibble
    RLC
    RLC
    RLC
    RLC
    CALL SUB1       ; Get ASCII code for lower nibble
    STA 5002
```

```
        HLT         ; Halt the program.

SUB1:   CPI 0A
        JC SKIP
        ADI 07
SKIP:   ADI 30
        RET         ; Return Subroutine
```

ASCII Character to Hexadecimal Number

1. Load the given data in A - register
2. Subtract 30H from A - register
3. Compare the content of A - register with 0AH
4. If A < 0AH, jump to step6. Else proceed to next step
5. Subtract 07H from A - register
6. Store the result
7. Terminate the program

- **Program**

```
LDA 2000
CALL ASCTOHEX
STA 2001
HLT

ASCTOHEX: SUI 30 ; This block Convert ASCII to Hexadecimal.
CPI 0A
RC
SUI 07
RET
```

10. BCD Arithmetic

Add 2 8-bit BCD Numbers

1. Load firs number into accumulator.
2. Add second number.
3. Apply decimal adjustment to accumulator.
4. Store result.

- **Program**

```
LXI H, 2000H
MOV A, M
INX H
ADD M
DAA
INX H
MOV M, A
HLT
```

Subtract the BCD number stored in E register from the number stored in the D register

1. Find 99’s complement of data of register E
2. Add 1 to find 100’s complement of data of register E
3. Add Data of Register D
4. Apply decimal adjustment

- **Program**

```
MVI A, 99H
SUB E        : Find the 99's complement of subtrahend
INR A        : Find 100's complement of subtrahend
ADD D        : Add minuend to 100's complement of subtrahend
DAA          : Adjust for BCD
HLT          : Terminate program execution
```

11. 16-Bit Data operations

Add Two 16 Bit Numbers

1. Initialize register C for using it as a counter for storing carry value.
2. Load data into HL register pair from one memory address (9000H).
3. Exchange contents of register pair HL with DE.
4. Load second data into HL register pair (from 9002H).
5. Add register pair DE with HL and store the result in HL.
6. If carry is present, go to 7 else go to 8.
7. Increment register C by 1.
8. Store value present in register pair HL to 9004H.
9. Move content of register C to accumulator A.
10. Store value present in accumulator (carry) into memory (9006H).
11. Terminate the program.

- **Program**

```
MVI C, 00H
LHLD 9000H
XCHG ; Exchange contents of register pair HL with DE
LHLD 9002H
DAD D ; Add register pair DE with HL and store the result in HL
JNC AHEAD ; If carry is present, go to AHEAD
INR C
AHEAD: SHLD 9004H ; Store value present in register pair HL to 9004H
MOV A, C
STA 9006H ; Store value present in accumulator (carry) into memory (9006H)
HLT
```

Subtract Two 16 Bit Numbers

1. Load first data from Memory (9000H) directly into register pair HL.
2. Exchange contents of register pair DE and HL.
3. Load second data from memory location (9002H) directly into register pair HL.

4. Move contents of register E into accumulator A.
5. Subtract content of register L from A.
6. Move contents of accumulator A into register L.
7. Move contents of register D into accumulator A.
8. Subtract with borrow contents of register H from accumulator A.
9. Move contents of accumulator A into register H.
10. Store data contained in HL register pair into memory (9004H).
11. Terminate the program.

- **Program**

```
LHLD 9000H ; Load first data from Memory (9000H) directly into register pair HL
XCHG ; Exchange contents of register pair DE and HL.
LHLD 9002H ; Load second data from memory location (9002H) directly into register pair HL
MOV A, E
SUB L
MOV L, A
MOV A, D
SBB H ; Subtract with borrow contents of register H from accumulator A
MOV H, A
SHLD 9004H ; Store data contained in HL register pair into memory (9004H)
HLT
```