XLAT instruction, the code of the pressed key obtained from the keyboard (i.e. the code to be translated) is moved in AL and the base address of the look up table containing the 7-segment codes is kept in BX. After the execution of the XLAT instruction, the 7-segment code corresponding to the pressed key is returned in AL, replacing the key code which was in AL prior to the execution of the XLAT instruction. To find out the exact address of the 7-segment code from the base address of look up table, the content of AL is added to BX internally, and the contents of the address pointed to by this new content of BX in DS are transferred to AL. The following sequence of instructions perform the task.

Example 2.22

```
MOV AX, SEG TABLE    ; Address of the segment containing look-up-table
MOV DS,AX            ; is transferred in DS
MOV AL, CODE         ; Code of the pressed key is transferred in AL
MOV BX, OFFSET TABLE; Offset of the code look-up-table in BX
XLAT                 ; Find the equivalent code and store in AL
```

Mnemonics & Description	*Instruction Code*			
Data Transfer				
MOV = Move	76543210	76543210	76543210	76543210
Register/Memory to/from Register	100010 dw	mod reg r/m		
Immediate to Register/Memory	1100011 w	mod 000 r/m	data	data if w = 1
Immediate to Register	1011 w reg	data	data if w = 1	
Memory to Accumulator	1010000 w	addr-low	addr-high	
Accumulator to Memory	1010001 w	addr-low	addr-high	
Register/Memory to Segment Register	10001110	mod 0 reg r/m		
Segment Register to Register/Memory	10001100	mod 0 reg r/m		
PUSH = Push:				
Register/Memory	11111111	mod 110 r/m		
Register	01010 reg			
Segment Register	000 reg 110			
POP = Pop:				
Register/Memory	10001111	mod 000 r/m		
Register	01011 reg			
Segment Register	000 reg 111			
XCHG = Exchange				
Register/Memory with Register	1000011 w	mod reg r/m		
Register with Accumulator	10010 reg			
IN = Input from:				
Fixed Port	1110010 w	port		
Variable Port	1110110 w			
OUT = Output to				
Fixed Port	1110011 w	port		
Variable Port	1110111 w			
XLAT = Translate Byte to AL	11010111			
LEA = Load EA to Register	10001101	mod reg r/m		
LDS = Load Pointer to DS	11000101	mod reg r/m		
LES = Load Pointer to ES	11000100	mod reg r/m		
LAHF = Load AH with Flags	10011111			
SAHF = Store AH into Flags	10011110			
PUSHF = Push Flags	10011100			
POPF = Pop Flags	10011101			
ARITHMETIC	**76543210**	**76543210**	**76543210**	**76543210**
ADD = Add:				
Reg/Memory with Register to Either	000000 dw	mod reg r/m		
Immediate to Register/Memory	100000 sw	mod 000 r/m	data	data if s w = 01

Mnemonics & Description	*Instruction Code*			
Immediate to Accumulator	0000010 w	data	data if w = 1	
ADC = **Add with Carry:**				
Reg/Memory with Register to Either	000100 dw	mod reg r/m		
Immediate to Register/Memory	100000 sw	mod 010 r/m	data	data if s w = 01
Immediate to Accumulator	0001010 w	data	data if w = 1	
INC = **Increment:**				
Register/Memory	1111111 w	mod 000 r/m		
Register	01000 reg			
AAA = ASCII Adjust for Addition	00110111			
DAA = Decimal Adjust for Addition	00100111			
SUB = Subtract				
Reg/Memory and Register to Either	001010 dw	mod reg r/m		
Immediate from Register/Memory	100000 sw	mod 101 r/m	data	data if s w = 01
Immediate from Accumulator	0010110 w	data	data if w = 1	
SBB = **Subtract with Borrow**				
Reg/Memory and Register to Either	000110 dw	mod reg r/m		
Immediate from Register/Memory	100000 sw	mod 011 r/m	data	data if s w = 01
Immediate from accumulator	0001110 w	data	data if w = 1	
DEC = **Decrement:**				
Register/Memory	1111111 w	mod 001 r/m		
Register	01001 reg			
NEG = Change sign	1111011 w	mod 011 r/m		
CMP = **Compare:**				
Register/Memory and Register	001110 dw	mod reg r/m		
Immediate with Register/Memory	100000 sw	mod 111 r/m	data	data if s w = 01
Immediate with Accumulator	0011110 w	data	data if w = 1	
AAS = ASCII Adjust for Subtract	00111111			
DAS = Decimal Adjust for Subtract	00101111			
MUL = Multiply (Unsigned)	1111011 w	mod 100 r/m		
IMUL = Integer Multiply (Signed)	1111011 w	mod 101 r/m		
AAM = ASCII Adjust Multiply	11010100	00001010		
DIV = Divide (Unsigned)	1111011 w	mod 110 r/m		
IDIV = Integer Divide (Signed)	1111011 w	mod 111 r/m		
AAD = ASCII Adjust for Divide	11010101	00001010		
CBW = Convert Byte to Word	10011000			
CWD = Convert Word to Double Word	10011001			
LOGICAL	76543210	76543210	76543210	76543210
NOT = Invert	1111011 w	mod 010 r/m		
SHL/SAL = Shift Logical/Arithmetic Left	110100 v w	mod 100 r/m		
SHR = Shift Logical Right	110100 v w	mod 101 r/m		
SAR = Shift Arithemtic Right	110100 v w	mod 111 r/m		
ROL = Rotate Left	110100 v w	mod 000 r/m		
ROR = Rotate Right	110100 v w	mod 001 r/m		
RCL = Rotate Through Carry Flag Left	110100 v w	mod 010 r/m		
RCR = Rotate Through Carry Right	110100 v w	mod 011 r/m		
AND = **And:**				
Reg/Memory and Register to Either	001000 dw	mod reg r/m		
Immediate to Register/Memory	1000000 w	mod 100 r/m	data	data if w = 1
Immediate to Accumulator	0010010 w	data	data if w = 1	
TEST = **And Function to Flags, No Result:**				
Register/Memory and Register	1000010 w	mod reg r/m		
Immediate Data and Register/Memory	1111011 w	mod 000 r/m	data	data if w = 1
Immediate Data and Accumulator	1010100 w	data	data if w = 1	
OR = **Or:**				
Reg/Memory and Register to Either	000010 dw	mod reg r/m		
Immediate to Register/Memory	1000000 w	mod 001 r/m	data	data if w = 1
Immediate to Accumulator	0000110 w	data	data if w = 1	

Mnemonics & Description	*Instruction Code*			
XOR = Exclusive or:				
Reg/Memory and Register to Either	001100 dw	mod reg r/m		
Immediate to Register/Memory	1000000 w	mod 110 r/m	data	data if w = 1
Immediate to Accumulator	0011010 w	data	data if w = 1	
STRING MANIPULATIONS				
REP = Repeat	1111001 z			
MOVS = Move Byte/Word	1010010 w			
CMPS = Compare Byte/Word	1010011 w			
SCAS = Scan Byte/Word	1010111 w			
LODS = Load byte/Wd to AL/AX	1010110 w			
STOS = Stor Byte/Wd from AL/A	1010101 w			
CONTROL TRANSFER				
CALL = Call:				
Direct Within Segment	11101000	disp-low	disp-high	
Indirect Within Segment	11111111	mod 010 r/m		
Direct Intersegment	10011010	offset-low	offset-high	
		seg-low	seg-high	
	76543210	76543210	76543210	
Indirect Intersegment	11111111	mod 011 r/m		
JMP = Unconditional Jump:				
Direct Within Segment	11101001	disp-low	disp-high	
Direct Within Segment-short	11101011	disp		
Indirect Within Segment	11111111	mod 100 r/m		
Direct Intersegment	11101010	offset-low	offset-high	
		seg-low	seg-high	
Indirect Intersegment	11111111	mod 101 r/m		
RET = Return from CALL:				
Within Segment	11000011			
Within Seg Adding Immediate to SP	11000010	data-low	data-high	
Intersegment	11001011			
Intersegment Adding Immediate to SP	11001010	data-low	data-high	
JE/JZ = Jump on Equal/Zero	01110100	disp		
JL/JNGE = Jump on Less/Not Greater or Equal	011111100	disp		
JLE/JNG = Jump on Less or Equal/Not Greater	01111110	disp		
JB/JNAE = Jump on Below/Not Above or Equal	01110010	disp		
JBE/JNA = Jump on Below or Equal/Not Above	01110110	disp		
JP/JPE = Jump on Parity/Parity Even	01111010	disp		
JO = Jump on Overflow	01110000	disp		
JS = Jump on Sign	01111000	disp		
JNE/JNZ = Jump on Not Equal/Not Zero	01110101	disp		
JNL/JGE = Jump on Not Less/Greater or Equal	01111101	disp		
JNLE/JG = Jump on Not Less or Equal/Greater	01111111	disp		
JNB/JAE = Jump on Not Below/Above or Equal	01110011	disp		
JNBE/JA = Jump on Not Below or Equal/Above	01110111	disp		
JNP/JPO = Jump on Not Par/Par Odd	01111011	disp		
JNO = Jump on Not Overflow	01110001	disp		
JNS = Jump on Not Sign	01111001	disp		
LOOP = Loop CX Times	11100010	disp		
LOOPZ/LOOPE = Loop While Zero/	11100001	disp		

Mnemonics & Description		*Instruction Code*
Equal		
LOOPNZ/LOOPNE = Loop While Not Zero/Equal	11100000	disp
JCXZ = Jump on CX Zero	11100011	disp
INT = **Interrupt**		
Type Specified	11001101	type
Type 3	11001100	
INTO = Interrupt on Overflow	11001110	
IRET = Interrupt Return	11001111	
	76543210	76543210
PROCESSOR CONTROL		
CLC = Clear Carry	11111000	
CMC = Complement Carry	11110101	
STC = Set Carry	11111001	
CLD = Clear Direction	11111100	
STD = Set Direction	11111101	
CLI = Clear Interrupt	11111010	
STI = Set Interrupt	11111011	
HLT = Halt	11110100	
WAIT = Wait	10011011	
ESC = Escape (to External Device)	11011xxx	mod xxx r/m
LOCK = Bus Lock Prefix	11110000	

*The v, w, d, s and z bits and the mod, reg, r/m fields are discussed in the addressing modes' section.

Fig. 2.4 *8086/8088 Instruction Set Summary*

LEA: Load Effective Address The load effective address instruction loads the effective address formed by destination operand into the specified source register. This instruction is more useful for assembly language rather than for machine language. The examples are given below.

Example 2.23

```
LEA BX,ADR      ; Effective address of Label ADR i.e. offset of ADR will be
                  transferred to Reg ; BX.
LEA SI, ADR[Bx]; offset of Label ADR will be added to content of Bx to form effective
                ; address and it will be loaded in SI
```

LDS/LES: Load Pointer to DS/ES This instruction loads the DS or ES register and the specified destination register in the instruction with the content of memory location specified as source in the instruction. The example in Fig. 2.5 explains the operation.

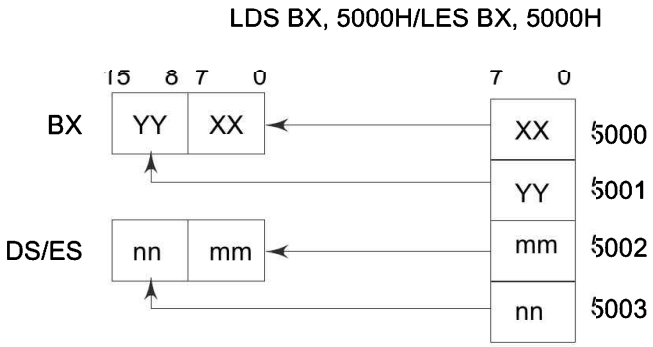

Fig. 2.5 *LDS/LES Instruction Execution*