8086/8088 Instruction Set and Assembler Directives 47

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 X LAT 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 X LAT 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 TABLEAddress of the segment contai ni ng l ook-up-tabl e

MOV DS ,AX  i s transferred i n DS

MOV AL , CODECode of the pressed key i s transferred i n AL

MOV BX , OFFSET TABLE; Offset of the code l ook- up-tabl e i n BX

X LATFi nd the equi val ent code and store i n AL

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Mnemonics & Description |  | Instruction Code | |  |
| Data Transfer  MOV = Move  Register/Memory to/from Register  Immediate to Register/Memory  Immediate to Register  Memory to Accumulator  Accumulator to Memory  Register/Memory to Segment Register Segment Register to Register/Memory PUSH = Push:  Register/Memory  Register  Segment Register POP = Pop:  Register/Memory  Register  Segment Register  XCHG = Exchange  Register/Memory with Register Register with Accumulator IN = Input from:  Fixed Port  Variable Port  OUT = Output to Fixed Port  Variable Port  XLAT = Translate Byte to AL LEA = Load EA to Register  LDS = Load Pointer to DS  LES = Load Pointer to ES  LAHF = Load AH with Flags SAHF = Store AH into Flags  PUSHF = Push Flags  POPF = Pop Flags ARITHMETIC ADD = Add:  Reg/Memory with Register to Either | 76543210  100010 dw    1011 w reg   1. w 2. w   10001110  10001100    01010 reg  000 reg 110    01011 reg  000 reg 111  1000011 w  10010 reg  1110010 w  1110110w  1 110011 w  1110111 w  11010111  10001101  11000101  11000100    10011100  10011101  76543210  000000 dw | 76543210 mod reg r/m mod 000 r/m data addr-low addr-low mod O reg r/m mod O reg r/m  mod 110 r/m  mod 000 r/m  mod reg r/m  port  port  mod reg r/m mod reg r/m mod reg r/m  76543210  mod reg r/m | 76543210  data data if w = 1 addr-high addr-high  76543210 | 76543210  data if w = 1  76543210 |

48 Advanced Mictoptocessors and Peripherals

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

8086/8088 Instruction Set and Assembler Directives 49

|  |  |  |  |
| --- | --- | --- | --- |
| Mnemonics & Description | Instruction Code | |  |
| XOR = Exclusive or:  Reg/Memory and Register to Either 001100 dw Immediate to Register/Memory 1000000 w  Immediate to Accumulator 001 1010 w  STRING MANIPULATIONS  REP = Repeat  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 1010110w  STOS = Stor Byte/Wd from AL/A 1010101 w CONTROL TRANSFER CALL = Call:  Direct Within Segment 11101000  Indirect Within Segment  Direct Intersegment 10011010  76543210 Indirect Intersegment  JMP = Unconditional Jump:  Direct Within Segment 11101001  Direct Within Segment-short 11101011  Indirect Within Segment  Direct Intersegment 11101010  Indirect Intersegment  RET = Return from CALL:  Within Segment 11000011  Within Seg Adding Immediate to SP 11000010  Intersegment 11001011  Intersegment Adding Immediate to SP 11001010  JEIJZ = Jump on Equal/Zero 01110100  JL/JNGE = Jump on Less/Not Greater or Equal  JLE/JNG = Jump on Less or  Equal/Not Greater  JBIJNAE = Jump on Below/Not Above 01110010 or Equal  JBE/JNA= Jump on Below or 01110110  Equal/Not Above  JPIJPE = Jump on Parity/Parity Even  JO = Jump on Overflow 01110000  JS = Jump on Sign  JNEIJNZ = Jump on Not Equal/Not 01110101  Zero  JNL/JGE= Jump on Not Less/Greater or Equal  JNLE/JG = Jump on Not Less or Equal/Greater  JNB/JAE= Jump on Not Below/Above 01110011 or Equal  JNBE/JA= Jump on Not Below or 01110111  Equal/Above  JNP/JPO= Jump on Not Par/Par Odd  JNO = Jump on Not Overflow 01110001  JNS = Jump on Not Sign  LOOP = Loop CX Times 11100010 LOOPZ/LOOPE = Loop While Zero/ 11100001 | mod reg r/m mod 110 r/m data  disp-low mod 010 r/m offset-low seg-low  76543210 mod 011 r/m  disp-low disp mod 100 r/m offset-low seg-low mod 101 r/m  data-low  data-low disp disp disp disp disp  disp disp disp disp  disp  disp  disp  disp  disp disp disp disp disp | data data if w = 1  disp-high  offset-high seg-high  76543210  disp-high  offset-high seg-high  data-high data-high | data if w = 1 |

50 Advanced Mictoptocessors and Peripherals

|  |  |  |
| --- | --- | --- |
| Mnemonics & Description |  | Instruction Code |
| Equal  LOOPNZ/LOOPNE = Loop While Not Zero/Equal  JCXZ = Jump on CX Zero  INT = Interrupt  Type Specified  Type 3  INTO = Interrupt on Overflow 'RET = Interrupt Return  PROCESSOR CONTROL  CLC = Clear carry  CMC = Complement Carry  STC = Set carry  CLD = Clear Direction  STD = Set Direction  CLI = Clear Interrupt  STI = Set Interrupt  HLT = Halt  WAIT = Wait  ESC = Escape (to External Device)  LOCK = Bus Lock Prefix | 1 1100000  11100011  11001101  11001100  11001110    76543210    10011011  1101 Ixxx | disp disp  type  76543210  mod xxx r/m |

\*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 Effecti ve address of Label ADR i . e . offset of ADR wi l l be transferred to Reg BX .

LEA S l , ADRCBx] ; offset of Label ADRwi 1 1 be added to content of Bxto formeffecti ve



address

and

it

will

be

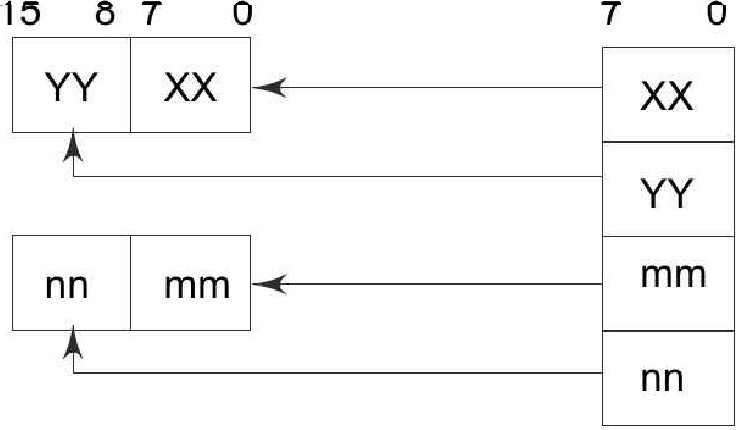
loaded

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.

LDS BX, 5000H/LES BX, 50001-1

BX5000

5001

DS/ES5002

5003

Fig. 2.5 LDS/LES Instruction Execution

MPI non-written