Instruction Set of 8086

Gursharan Singh Tatla
professorgstatla@gmail.com

Instruction Set of 8086

- An instruction is a binary pattern designed inside a microprocessor to perform a specific function.
- The entire group of instructions that a microprocessor supports is called Instruction Set.
- 8086 has more than 20,000 instructions.

Classification of Instruction Set

- Data Transfer Instructions
- Arithmetic Instructions
- Bit Manipulation Instructions
- Program Execution Transfer Instructions
- String Instructions
- Processor Control Instructions

Data Transfer Instructions

These instructions are used to transfer data from source to destination.

The operand can be a constant, memory location, register or I/O port address.

Data Transfer Instructions

MOV Des, Src:
- Src operand can be register, memory location or immediate operand.
- Des can be register or memory operand.
- Both Src and Des cannot be memory location at the same time.
- E.g.:
  - MOV CX, 037AH
  - MOV AL, BL
  - MOV BX, [030A H]

PUSH Operand:
- It pushes the operand into top of stack.
- E.g.: PUSH BX

POP Des:
- It pops the operand from top of stack to Des.
- Des can be a general purpose register, segment register (except CS) or memory location.
- E.g.: POP AX
Data Transfer Instructions

- **XCHG Des, Src:**
  - This instruction exchanges Src with Des.
  - It cannot exchange two memory locations directly.
  - E.g.: XCHG DX, AX

- **IN Accumulator, Port Address:**
  - It transfers the operand from specified port to accumulator register.
  - E.g.: IN AX, 0028 H

- **OUT Port Address, Accumulator:**
  - It transfers the operand from accumulator to specified port.
  - E.g.: OUT 0028 H, AX

- **LEA Register, Src:**
  - It loads a 16-bit register with the offset address of the data specified by the Src.
  - E.g.: LEA BX, [DI]
  - This instruction loads the contents of DI (offset) into the BX register.

- **LDS Des, Src:**
  - It loads 32-bit pointer from memory source to destination register and DS.
  - The offset is placed in the destination register and the segment is placed in DS.
  - To use this instruction the word at the lower memory address must contain the offset and the word at the higher address must contain the segment.
  - E.g.: LDS BX, [0301 H]

- **LES Des, Src:**
  - It loads 32-bit pointer from memory source to destination register and ES.
  - The offset is placed in the destination register and the segment is placed in ES.
  - This instruction is very similar to LDS except that it initializes ES instead of DS.
  - E.g.: LES BX, [0301 H]

- **LAHF:**
  - It copies the lower byte of flag register to AH.

- **SAHF:**
  - It copies the contents of AH to lower byte of flag register.

- **PUSHF:**
  - Pushes flag register to top of stack.

- **POPF:**
  - Pops the stack top to flag register.
Arithmetic Instructions

**ADD Des, Src:**
- It adds a byte to byte or a word to word.
- It effects AF, CF, OF, PF, SF, ZF flags.
- E.g.:
  - ADD AL, 74H
  - ADD DX, AX
  - ADD AX, [BX]

**ADC Des, Src:**
- It adds the two operands with CF.
- It effects AF, CF, OF, PF, SF, ZF flags.
- E.g.:
  - ADC AL, 74H
  - ADC DX, AX
  - ADC AX, [BX]

**SUB Des, Src:**
- It subtracts a byte from byte or a word from word.
- It effects AF, CF, OF, PF, SF, ZF flags.
- For subtraction, CF acts as borrow flag.
- E.g.:
  - SUB AL, 74H
  - SUB DX, AX
  - SUB AX, [BX]

**SBB Des, Src:**
- It subtracts the two operands and also the borrow from the result.
- It effects AF, CF, OF, PF, SF, ZF flags.
- E.g.:
  - SBB AL, 74H
  - SBB DX, AX
  - SBB AX, [BX]

**INC Src:**
- It increments the byte or word by one.
- The operand can be a register or memory location.
- It effects AF, OF, PF, SF, ZF flags.
- CF is not effected.
- E.g.: INC AX

**DEC Src:**
- It decrements the byte or word by one.
- The operand can be a register or memory location.
- It effects AF, OF, PF, SF, ZF flags.
- CF is not effected.
- E.g.: DEC AX
Arithmetic Instructions

- **AAA (ASCII Adjust after Addition):**
  - The data entered from the terminal is in ASCII format.
  - In ASCII, 0 – 9 are represented by 30H – 39H.
  - This instruction allows us to add the ASCII codes.
  - This instruction does not have any operand.

- **Other ASCII Instructions:**
  - AAS (ASCII Adjust after Subtraction)
  - AAM (ASCII Adjust after Multiplication)
  - AAD (ASCII Adjust Before Division)

- **DAA (Decimal Adjust after Addition):**
  - It is used to make sure that the result of adding two BCD numbers is adjusted to be a correct BCD number.
  - It only works on AL register.

- **DAS (Decimal Adjust after Subtraction):**
  - It is used to make sure that the result of subtracting two BCD numbers is adjusted to be a correct BCD number.
  - It only works on AL register.

- **NEG Src:**
  - It creates 2's complement of a given number.
  - That means, it changes the sign of a number.

- **CMP Des, Src:**
  - It compares two specified bytes or words.
  - The Src and Des can be a constant, register or memory location.
  - Both operands cannot be a memory location at the same time.
  - The comparison is done simply by internally subtracting the source from destination.
  - The value of source and destination does not change, but the flags are modified to indicate the result.

- **MUL Src:**
  - It is an unsigned multiplication instruction.
  - It multiplies two bytes to produce a word or two words to produce a double word.
  - AX = AL * Src
  - DX : AX = AX * Src
  - This instruction assumes one of the operand in AL or AX.
  - Src can be a register or memory location.

- **IMUL Src:**
  - It is a signed multiplication instruction.

- **DIV Src:**
  - It is an unsigned division instruction.
  - It divides word by byte or double word by word.
  - The operand is stored in AX, divisor is Src and the result is stored as:
    - AH = remainder
    - AL = quotient

- **IDIV Src:**
  - It is a signed division instruction.
Arithmetic Instructions

- **CBW (Convert Byte to Word):**
  - This instruction converts byte in AL to word in AX.
  - The conversion is done by extending the sign bit of AL throughout AH.
- **CWD (Convert Word to Double Word):**
  - This instruction converts word in AX to double word in DX:AX.
  - The conversion is done by extending the sign bit of AX throughout DX.

Bit Manipulation Instructions

- These instructions are used at the bit level.
- These instructions can be used for:
  - Testing a zero bit
  - Set or reset a bit
  - Shift bits across registers

**NOT Src:**
- It complements each bit of Src to produce its complement of the specified operand.
- The operand can be a register or memory location.

**AND Des, Src:**
- It performs AND operation of Des and Src.
- Src can be immediate number, register or memory location.
- Des can be register or memory location.
- Both operands cannot be memory locations at the same time.
- CF and OF become zero after the operation.
- PF, SF and ZF are updated.

**OR Des, Src:**
- It performs OR operation of Des and Src.
- Src can be immediate number, register or memory location.
- Des can be register or memory location.
- Both operands cannot be memory locations at the same time.
- CF and OF become zero after the operation.
- PF, SF and ZF are updated.

**XOR Des, Src:**
- It performs XOR operation of Des and Src.
- Src can be immediate number, register or memory location.
- Des can be register or memory location.
- Both operands cannot be memory locations at the same time.
- CF and OF become zero after the operation.
- PF, SF and ZF are updated.
Bit Manipulation Instructions

- **SHL Des, Count:**
  - It shifts bits of byte or word left, by count.
  - It puts zero(s) in LSBs.
  - MSB is shifted into carry flag.
  - If the number of bits desired to be shifted is 1, then the immediate number 1 can be written in Count.
  - However, if the number of bits to be shifted is more than 1, then the count is put in CL register.

- **SHR Des, Count:**
  - It shifts bits of byte or word right, by count.
  - It puts zero(s) in MSBs.
  - LSB is shifted into carry flag.
  - If the number of bits desired to be shifted is 1, then the immediate number 1 can be written in Count.
  - However, if the number of bits to be shifted is more than 1, then the count is put in CL register.

- **ROL Des, Count:**
  - It rotates bits of byte or word left, by count.
  - MSB is transferred to LSB and also to CF.
  - If the number of bits desired to be shifted is 1, then the immediate number 1 can be written in Count.
  - However, if the number of bits to be shifted is more than 1, then the count is put in CL register.

- **ROR Des, Count:**
  - It rotates bits of byte or word right, by count.
  - LSB is transferred to MSB and also to CF.
  - If the number of bits desired to be shifted is 1, then the immediate number 1 can be written in Count.
  - However, if the number of bits to be shifted is more than 1, then the count is put in CL register.

Program Execution Transfer Instructions

- These instructions cause change in the sequence of the execution of instruction.
- This change can be through a condition or sometimes unconditional.
- The conditions are represented by flags.

- **CALL Des:**
  - This instruction is used to call a subroutine or function or procedure.
  - The address of next instruction after CALL is saved onto stack.

- **RET:**
  - It returns the control from procedure to calling program.
  - Every CALL instruction should have a RET.
Program Execution Transfer Instructions

- **JMP Des:**
  - This instruction is used for unconditional jump from one place to another.

- **Jxx Des (Conditional Jump):**
  - All the conditional jumps follow some conditional statements or any instruction that affects the flag.

Conditional Jump Table

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Meaning</th>
<th>Jump Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>JA</td>
<td>Jump if Above</td>
<td>CF = 0 and ZF = 0</td>
</tr>
<tr>
<td>JAE</td>
<td>Jump if Above or Equal</td>
<td>CF = 0</td>
</tr>
<tr>
<td>JB</td>
<td>Jump if Below</td>
<td>CF = 1</td>
</tr>
<tr>
<td>JBE</td>
<td>Jump if Below or Equal</td>
<td>CF = 1 or ZF = 1</td>
</tr>
<tr>
<td>JC</td>
<td>Jump if Carry</td>
<td>CF = 1</td>
</tr>
<tr>
<td>JE</td>
<td>Jump if Equal</td>
<td>ZF = 1</td>
</tr>
<tr>
<td>JNC</td>
<td>Jump if Not Carry</td>
<td>CF = 0</td>
</tr>
<tr>
<td>JNE</td>
<td>Jump if Not Equal</td>
<td>ZF = 0</td>
</tr>
<tr>
<td>JNZ</td>
<td>Jump if Not Zero</td>
<td>ZF = 0</td>
</tr>
<tr>
<td>JPE</td>
<td>Jump if Parity Even</td>
<td>PF = 1</td>
</tr>
<tr>
<td>JPO</td>
<td>Jump if Parity Odd</td>
<td>PF = 0</td>
</tr>
<tr>
<td>JZ</td>
<td>Jump if Zero</td>
<td>ZF = 1</td>
</tr>
</tbody>
</table>

String Instructions

- **Loop Des:**
  - This is a looping instruction.
  - The number of times looping is required is placed in the CX register.
  - With each iteration, the contents of CX are decremented.
  - ZF is checked whether to loop again or not.

String Instructions

- **CMPS Des, Src:**
  - It compares the string bytes or words.

- **SCAS String:**
  - It scans a string.
  - It compares the String with byte in AL or with word in AX.

String Instructions

- **MOV Des, MOVSB / MOVSW:**
  - It causes moving of byte or word from one string to another.
  - In this instruction, the source string is in Data Segment and destination string is in Extra Segment.
  - SI and DI store the offset values for source and destination index.
String Instructions

- **REP (Repeat):**
  - This is an instruction prefix.
  - It causes the repetition of the instruction until CX becomes zero.
  - E.g.: REP MOVSB STR1, STR2
    - It copies byte by byte contents.
    - REP repeats the operation MOVSB until CX becomes zero.

Processor Control Instructions

- These instructions control the processor itself.
- 8086 allows to control certain control flags that:
  - causes the processing in a certain direction
  - processor synchronization if more than one microprocessor attached.

- **STC:**
  - It sets the carry flag to 1.

- **CLC:**
  - It clears the carry flag to 0.

- **CMC:**
  - It complements the carry flag.

- **STD:**
  - It sets the direction flag to 1.
    - If it is set, string bytes are accessed from higher memory address to lower memory address.

- **CLD:**
  - It clears the direction flag to 0.
    - If it is reset, the string bytes are accessed from lower memory address to higher memory address.

Thank You 😊
Have a Nice Day 😊