The 8086 microprocessor supports 8 types of instructions −

* Data Transfer Instructions
* Arithmetic Instructions
* Bit Manipulation Instructions
* String Instructions
* Program Execution Transfer Instructions (Branch & Loop Instructions)
* Processor Control Instructions
* Iteration Control Instructions
* Interrupt Instructions

**Instruction to transfer a word**

* **MOV** − Used to copy the byte or word from the provided source to the provided destination.

|  |  |  |
| --- | --- | --- |
| **Destination** | **source** | **example** |
| Register | register | mov ax,bx |
| Register | immediate | mov ax,10h |
| Register | memory | mov ax,es:[bx] |
| Memory | immediate | mov aNumber,10h |
| Memory | register | mov aDigit,ax |

* **PPUSH** − Used to put a word at the top of the stack.
* **POP** − Used to get a word from the top of the stack to the provided location.
* **PUSHA** − Used to put all the registers into the stack.
* **POPA** − Used to get words from the stack to all registers.
* **XCHG** − Used to exchange the data from two locations.
* **XLAT** − Used to translate a byte in AL using a table in the memory.

**Instructions for input and output port transfer**

* **IN** − Used to read a byte or word from the provided port to the accumulator.
* **OUT** − Used to send out a byte or word from the accumulator to the provided port.

**Instructions to transfer the address**

* **LEA** − Used to load the address of operand into the provided register.
* **LDS** − Used to load DS register and other provided register from the memory
* **LES** − Used to load ES register and other provided register from the memory.

**Instructions to transfer flag registers**

* **LAHF** − Used to load AH with the low byte of the flag register.
* **SAHF** − Used to store AH register to low byte of the flag register.
* **PUSHF** − Used to copy the flag register at the top of the stack.
* **POPF** − Used to copy a word at the top of the stack to the flag register.

**Arithmetic Instructions**

These instructions are used to perform arithmetic operations like addition, subtraction, multiplication, division, etc.

Following is the list of instructions under this group −

**Instructions to perform addition**

* **ADD** − Used to add the provided byte to byte/word to word.
* **ADC** − Used to add with carry.
* **INC** − Used to increment the provided byte/word by 1.
* **AAA** − Used to adjust ASCII after addition.
* **DAA** − Used to adjust the decimal after the addition/subtraction operation.

**Instructions to perform subtraction**

* **SUB** − Used to subtract the byte from byte/word from word.
* **SBB** − Used to perform subtraction with borrow.
* **DEC** − Used to decrement the provided byte/word by 1.
* **NPG** − Used to negate each bit of the provided byte/word and add 1/2’s complement.
* **CMP** − Used to compare 2 provided byte/word.
* **AAS** − Used to adjust ASCII codes after subtraction.
* **DAS** − Used to adjust decimal after subtraction.

**Instruction to perform multiplication**

* **MUL** − Used to multiply unsigned byte by byte/word by word.
* **IMUL** − Used to multiply signed byte by byte/word by word.
* **AAM** − Used to adjust ASCII codes after multiplication.

**Instructions to perform division**

* **DIV** − Used to divide the unsigned word by byte or unsigned double word by word.
* **IDIV** − Used to divide the signed word by byte or signed double word by word.
* **AAD** − Used to adjust ASCII codes after division.
* **CBW** − Used to fill the upper byte of the word with the copies of sign bit of the lower byte.
* **CWD** − Used to fill the upper word of the double word with the sign bit of the lower word.

**Bit Manipulation Instructions**

These instructions are used to perform operations where data bits are involved, i.e. operations like logical, shift, etc.

Following is the list of instructions under this group −

**Instructions to perform logical operation**

* **NOT** − Used to invert each bit of a byte or word.
* **AND** − Used for adding each bit in a byte/word with the corresponding bit in another byte/word.
* **OR** − Used to multiply each bit in a byte/word with the corresponding bit in another byte/word.
* **XOR** − Used to perform Exclusive-OR operation over each bit in a byte/word with the corresponding bit in another byte/word.
* **TEST** − Used to add operands to update flags, without affecting operands.

**Instructions to perform shift operations**

* **SHL/SAL** − Used to shift bits of a byte/word towards left and put zero(S) in LSBs.
* **SHR** − Used to shift bits of a byte/word towards the right and put zero(S) in MSBs.
* **SAR** − Used to shift bits of a byte/word towards the right and copy the old MSB into the new MSB.

**Instructions to perform rotate operations**

* **ROL** − Used to rotate bits of byte/word towards the left, i.e. MSB to LSB and to Carry Flag [CF].
* **ROR** − Used to rotate bits of byte/word towards the right, i.e. LSB to MSB and to Carry Flag [CF].
* **RCR** − Used to rotate bits of byte/word towards the right, i.e. LSB to CF and CF to MSB.
* **RCL** − Used to rotate bits of byte/word towards the left, i.e. MSB to CF and CF to LSB.

**String Instructions**

String is a group of bytes/words and their memory is always allocated in a sequential order.

Following is the list of instructions under this group −

* **REP** − Used to repeat the given instruction till CX ≠ 0.
* **REPE/REPZ** − Used to repeat the given instruction until CX = 0 or zero flag ZF = 1.
* **REPNE/REPNZ** − Used to repeat the given instruction until CX = 0 or zero flag ZF = 1.
* **MOVS/MOVSB/MOVSW** − Used to move the byte/word from one string to another.
* **COMS/COMPSB/COMPSW** − Used to compare two string bytes/words.
* **INS/INSB/INSW** − Used as an input string/byte/word from the I/O port to the provided memory location.
* **OUTS/OUTSB/OUTSW** − Used as an output string/byte/word from the provided memory location to the I/O port.
* **SCAS/SCASB/SCASW** − Used to scan a string and compare its byte with a byte in AL or string word with a word in AX.
* **LODS/LODSB/LODSW** − Used to store the string byte into AL or string word into AX.

**Program Execution Transfer Instructions (Branch and Loop Instructions)**

These instructions are used to transfer/branch the instructions during an execution. It includes the following instructions −

Instructions to transfer the instruction during an execution without any condition −

* **CALL** − Used to call a procedure and save their return address to the stack.
* **RET** − Used to return from the procedure to the main program.
* **JMP** − Used to jump to the provided address to proceed to the next instruction.

Instructions to transfer the instruction during an execution with some conditions −

* **JA/JNBE** − Used to jump if above/not below/equal instruction satisfies.
* **JAE/JNB** − Used to jump if above/not below instruction satisfies.
* **JBE/JNA** − Used to jump if below/equal/ not above instruction satisfies.
* **JC** − Used to jump if carry flag CF = 1
* **JE/JZ** − Used to jump if equal/zero flag ZF = 1
* **JG/JNLE** − Used to jump if greater/not less than/equal instruction satisfies.
* **JGE/JNL** − Used to jump if greater than/equal/not less than instruction satisfies.
* **JL/JNGE** − Used to jump if less than/not greater than/equal instruction satisfies.
* **JLE/JNG** − Used to jump if less than/equal/if not greater than instruction satisfies.
* **JNC** − Used to jump if no carry flag (CF = 0)
* **JNE/JNZ** − Used to jump if not equal/zero flag ZF = 0
* **JNO** − Used to jump if no overflow flag OF = 0
* **JNP/JPO** − Used to jump if not parity/parity odd PF = 0
* **JNS** − Used to jump if not sign SF = 0
* **JO** − Used to jump if overflow flag OF = 1
* **JP/JPE** − Used to jump if parity/parity even PF = 1
* **JS** − Used to jump if sign flag SF = 1

**Processor Control Instructions**

These instructions are used to control the processor action by setting/resetting the flag values.

Following are the instructions under this group −

* **STC** − Used to set carry flag CF to 1
* **CLC** − Used to clear/reset carry flag CF to 0
* **CMC** − Used to put complement at the state of carry flag CF.
* **STD** − Used to set the direction flag DF to 1
* **CLD** − Used to clear/reset the direction flag DF to 0
* **STI** − Used to set the interrupt enable flag to 1, i.e., enable INTR input.
* **CLI** − Used to clear the interrupt enable flag to 0, i.e., disable INTR input.

**Iteration Control Instructions**

These instructions are used to execute the given instructions for number of times. Following is the list of instructions under this group −

* **LOOP** − Used to loop a group of instructions until the condition satisfies, i.e., CX = 0
* **LOOPE/LOOPZ** − Used to loop a group of instructions till it satisfies ZF = 1 & CX = 0
* **LOOPNE/LOOPNZ** − Used to loop a group of instructions till it satisfies ZF = 0 & CX = 0
* **JCXZ** − Used to jump to the provided address if CX = 0

**Interrupt Instructions**

These instructions are used to call the interrupt during program execution.

* **INT** − Used to interrupt the program during execution and calling service specified.
* **INTO** − Used to interrupt the program during execution if OF = 1
* **IRET** − Used to return from interrupt service to the main program

[adc](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#adc) Add with carry flag

[add](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#add) Add two numbers

[and](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#and) Bitwise logical AND

[call](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#call) Call procedure or function

[cbw](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#cbw) Convert byte to word (signed)

[cli](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#cli) Clear interrupt flag (disable interrupts)

[cwd](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#cwd) Convert word to doubleword (signed)

[cmp](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#cmp) Compare two operands

[dec](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#dec) Decrement by 1

[div](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#div) Unsigned divide

[idiv](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#idiv) Signed divide

[imul](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#imul) Signed multiply

[in](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#in) Input (read) from port

[inc](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#inc) Increment by 1

[int](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#int) Call to interrupt procedure

[iret](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#iret) Interrupt return

[j??](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#jcc) Jump if ?? condition met

[jmp](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#jmp) Unconditional jump

[lea](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#lea) Load effective address offset

[mov](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#mov) Move data

[mul](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#mul) Unsigned multiply

[neg](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#neg) Two's complement negate

[nop](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#nop) No operation

[not](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#not) One's complement negate

[or](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#or) Bitwise logical OR

[out](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#out) Output (write) to port

[pop](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#pop) Pop word from stack

[popf](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#popf) Pop flags from stack

[push](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#push) Push word onto stack

[pushf](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#pushf) Push flags onto stack

[ret](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#ret) Return from procedure or function

[sal](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#sal) Bitwise arithmetic left shift (same as shl)

[sar](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#sar) Bitwise arithmetic right shift (signed)

[sbb](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#sbb) Subtract with borrow

[shl](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#shl) Bitwise left shift (same as sal)

[shr](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#shr) Bitwise right shift (unsigned)

[sti](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#sti) Set interrupt flag (enable interrupts)

[sub](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#sub) Subtract two numbers

[test](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#test) Bitwise logical compare

[xor](http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html#xor) Bitwise logical XOR