8086 Instruction Set Summary

Data Transfer Instructions.

MOV	Move byte or word to register or memory
IN, OUT	Input byte or word from port, output word to port
LEA	Load effective address
LDS, LES	Load pointer using data segment, extra segment
PUSH, POP	Push word onto stack, pop word off stack
XCHG	Exchange byte or word
XLAT	Translate byte using look-up table

Logical Instructions:

NOT	Logical NOT of byte or word (one's complement)
AND	Logical AND of byte or word
OR	Logical OR of byte or word
XOR	Logical exclusive-OR of byte or word
TEST	Test byte or word (AND without storing)

Shift and Rotate Instructions:

SHL, SHR	Logical shift left, right byte or word? by 1 or CL
SAL, SAR	Arithmetic shift left, right byte or word? by 1 or CL
ROL, ROR	Rotate left, right byte or word? by 1 or CL
RCL, RCR	Rotate left, right through carry byte or word? by 1 or CL

Arithmetic Instructions:

ADD, SUB	Add, subtract byte or word
ADC, SBB	Add, subtract byte or word and carry (borrow)
INC, DEC	Increment, decrement byte or word
NEG	Negate byte or word (two's complement)
CMP	Compare byte or word (subtract without storing)
MUL, DIV	Multiply, divide byte or word (unsigned)
IMUL, IDIV	Integer multiply, divide byte or word (signed)
CBW, CWD	Convert byte to word, word to double word (useful before multiply/divide)

Adjustments after arithmetic operations:

AAA, AAS, AAM, AAD	ASCII adjust for addition, subtraction, multiplication, division (ASCII codes 30-39)
DAA, DAS	Decimal adjust for addition, subtraction (binary coded decimal numbers)

Transfer Instructions

Conditional jumps:

JMP Unconditional jump (*short* ?127/8, *near* ?32K, *far* between segments)

JA (JNBE)	Jump if above (not below or equal)? +127, -128 range only
JAE (JNB)	Jump if above or equal(not below)? +127, -128 range only
JB (JNAE)	Jump if below (not above or equal)? +127, -128 range only
JBE (JNA)	Jump if below or equal (not above)? +127, -128 range only
JE (JZ)	Jump if equal (zero)? +127, -128 range only
JG (JNLE)	Jump if greater (not less or equal)? +127, -128 range only
JGE (JNL)	Jump if greater or equal (not less)? +127, -128 range only
JL (JNGE)	Jump if less (not greater nor equal)? +127, -128 range only
JLE (JNG)	Jump if less or equal (not greater)? +127, -128 range only
JC, JNC	Jump if carry set, carry not set? +127, -128 range only
JO, JNO	Jump if overflow, no overflow? +127, -128 range only
JS, JNS	Jump if sign, no sign? +127, -128 range only
JNP (JPO)	Jump if no parity (parity odd)? +127, -128 range only
JP (JPE)	Jump if parity (parity even)? +127, -128 range only

Loop control:

LOOP	Loop unconditional, count in CX, short jump to target address
LOOPE (LOOPZ)	Loop if equal (zero), count in CX, short jump to target address
LOOPNE (LOOPNZ)	Loop if not equal (not zero), count in CX, short jump to target address
JCXZ	Jump if CX equals zero (used to skip code in loop)

Subroutine and Interrupt Instructions:

CALL, RET	Call, return from procedure (inside or outside current segment)
INT, INTO	Software interrupt, interrupt if overflow
IRET	Return from interrupt

String Instructions:

MOVS	Move byte or word string
MOVSB, MOVSW	Move byte, word string
CMPS	Compare byte or word string
SCAS	Scan byte or word string (comparing to A or AX)
LODS, STOS	Load, store byte or word string to AL or AX

Repeat instructions (placed in front of other string operations):

REP	Repeat
REPE, REPZ	Repeat while equal, zero
REPNE, REPNZ	Repeat while not equal (zero)

Processor Control Instructions.

Flag manipulation:

STC, CLC, CMC	Set, clear, complement carry flag
STD, CLD	Set, clear direction flag
STI, CLI	Set, clear interrupt enable flag
LAHF, SAHF	Load AH from flags, store AH into flags
PUSHF, POPF	Push flags onto stack, pop flags off stack

Coprocessor, multiprocessor interface:

ESC	Escape to external processor interface
LOCK	Lock bus during next instruction

Inactive states:

NOP	No operation
WAIT	Wait for TEST pin activity
HLT	Halt processor

8086 Registers

General Purpose Registers

Accumulator	**AX**	**Multiply, divide, I/O**
Base	**BX**	**Pointer to base addresss (data)**
Count	**CX**	**Count for loops, shifts**
Data	**DX**	**Multiply, divide, I/O**

Pointer and Index Registers

Stack Pointer	**SP**	**Pointer to top of stack**
Base Pointer	**BP**	**Pointer to base address (stack)**
Source Index	**SI**	**Source string/index pointer**
Destination Index	**DI**	**Destination string/index pointer**

Segment Registers

Code Segment	**CS**
Data Segment	**DS**
Stack Segment	**SS**
Extra Segment	**ES**

Other Registers

Flags:

Status (Flags) Register

Nine individual **bits** of the **status** register are used as **control** flags (3 of them) and **status** flags (6 of them). The remaining 7 are not used.

A flag can only take on the values 0 and 1. We say a flag is **set** if it has the value 1.
The status flags are used to record specific characteristics of arithmetic and of logical instructions.

Instruction Pointer:

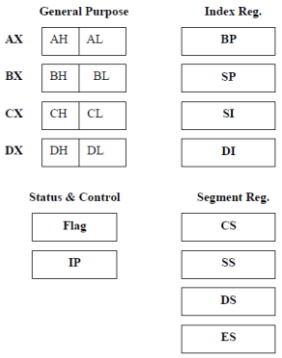
Instruction Pointer Register

This is a crucially important register which is used to control which instruction the CPU executes. The **ip**, or program counter, is used to store the memory location of the next instruction to be executed.

Instructions and Flags
MOV and XCHG - no flags are changed
ADD and SUB - all flags affected
INC and DEC - all except CF
NEG - all flags affected
CF=0 only if value is 0
OF=1 only if value is - MAXINT 80h or 8000h

Registers

✓ Registers are 8, 16, or 32-bit high speed storage locations directly inside the CPU, designed to be accessed at much higher speed than conventional memory.



Data Registers

✓ The general purpose registers, are used for arithmetic and data movement. Each register can be addressed as either 16-bit or 8 bit value.

✓ Example, AX register is a 16-bit register, its upper 8-bit is called AH, and its lower 8-bit is called AL. Bit 0 in AL corresponds to bit 0 in AX and bit 0 in AH corresponds to bit 8 in AX

✓ Instructions can address either 16-bit data register as AX, BX, CX, and DX or 8-bit register as AL, AH, BL, BH, CL, CH, Dl, and DH. If we move 126FH to AX then AL would immediately 6FH and AH = 12H.

Each general purpose register has special attributes:

1. **AX (Accumulator):** AX is the accumulator register because it is favored by the CPU for arithmetic operations. Other operations are also slightly more efficient when performed using AX.
2. **BX (Base):** the BX register can hold the address of a procedure or variable. Three other registers with this ability are SI, DI and BP. The BX register can also perform arithmetic and data movement.
3. **CX (Counter):** the CX register acts as a counter for repeating or looping instructions. These instructions automatically repeat and decrement CX.
4. **DX (Data):** the DX register has a special role in multiply and divide operation. When multiplying for example DX hold the high 16 bit of the product.

Segment Registers*: the CPU contain four segment registers, used as base location for program instruction, and for the stack.*

1. **Code Segment (CS):** The code segment register holds the base location of all executable instructions (code) in a program.
2. **Data Segment (DS):** the data segment register is the default base location for variables. The CPU calculates their location using the segment value in DS.
3. **Stack Segment (SS):** the stack segment register contain the base location of the stack.
4. **Extra Segment (ES):** The extra segment register is an additional base location for memory variables.

Index registers*: index registers contain the offset of data and instructions. The term offset refers to the distance of a variable, label, or instruction from its base segment. The index registers are:*

1. **Base Pointer (BP):** the BP register contain an assumed offset from the stack segment register, as does the stack pointer. The base pointer register is often used by a subroutine to locate variables that were passed on the stack by a calling program.
2. **Stack Pointer (SP):** the stack pointer register contain the offset of the top of the stack. The stack pointer and the stack segment register combine to form the complete address of the top of the stack.
3. **Source Index (SI):** This register takes its name from the string movement instruction, in which the source string is pointed to by the source index register.
4. **Destination Index (DI):** the DI register acts as the destination for string movement instruction.

Status and Control register:

1. **Instruction Pointer (IP):** The instruction pointer register always contain the offset of the next instruction to be executed within the current code segment. The instruction pointer and the code segment register combine to form the complete address of the next instruction.
2. **The Flag Register**: is a special register with individual bit positions assigned to show the status of the CPU or the result of arithmetic operations.

Flag Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	O	D	I	T	S	Z	X	A	X	P	X	C

O= Over Flow
D= Direction
I= Interrupt
T= trap
X= Undefined

S= sign
Z=Zero
A= Auxiliary
P= parity
C= carry

There two basic types of flags, (control flags and status flags):

Control Flags: *individual bits can be set in the flag register by the programmer to control the CPU operation, these are:*

- ✓ **Direction Flag (DF):** affects block data transfer instructions, such as MOVS and CMPS. The flag values are 0 = up and 1 = down.
- ✓ **Interrupt flag (IF):** dictates whether or not a system interrupt can occur. Such as keyboard, disk drive, and the system clock timer. A program will sometimes briefly disable the interrupt when performing a critical operation that cannot be interrupted. The flag values are 1 = enable, 0 = disable.
- ✓ **Trap flag (TF):** Determine whether or not the CPU is halted after each instruction. When this is set, a debugging program can let a programmer to enter single stepping (trace) through a program one instruction at a time. The flag values are 1 = on, 0 = off. The flag can be set by INT 3 instruction.

Status Flags: *The status flags reflect the outcomes of arithmetic and logical operations performed by the CPU, these are:*

- ✓ **Carry Flag (CF):** is set when the result of an unsigned arithmetic operation is too large to fit into the destination for example, if the sum of 71 and 99 where stored in the 8-bit register AL, the result cause the carry flag to be 1. The flag values = 1 = carry, 0 = no carry.
- ✓ **Overflow Flag (OF)**: is set when the result of a signed arithmetic operation is too wide (too many bits) to fit into destination. 1 = overflow, 0 = no overflow.
- ✓ **Sign Flag (SF):** is set when the result of an arithmetic or logical operation generates a negative result, 1= negative, 0 = positive.
- ✓ **Zero Flag (ZF):** is set when the result of an arithmetic or logical operation generates a result of zero, the flag is used primarily by jump or loop instructions to allow branching to a new location in a program based on the comparison of two values. The flag value = 1 = zero, & 0= not zero.
- ✓ **Auxiliary Flag**: is set when an operation causes a carry from bit 3 to bit 4 (or borrow from bit 4 to bit 3) of an operand. The flag value = 1 = carry, 0 = no carry.
- ✓ **Parity Flag**: reflect the number of 1 bits in the result of an operation. If there is an even number of bit, the parity is even. If there are an odd number of bits, parity is odd. This flag is used by the OS to verify memory integrity and by communication software to verify the correct transmission of data.

Flags and instructions.

Control flags

- DF - Direction flag
  1. STD: direction = down
  2. CLD: direction = up
- IF - Interrupt enable
  1. STI: enable external interrupts
  2. CLI: disable maskable external interrupts
- TF - Trace flag
  1. Interrupt 1 after executing instruction, if set

Status Flags

- Carry
  1. carry or borrow at MSB in add or subtract
  2. last bit shifted out
- Parity
  1. low byte of result has even parity
- Auxiliary
  1. carry or borrow at bit 3
- Zero
  1. result is 0
- Sign
  1. result is negative
- Overflow
  1. signed overflow occurred during add or subtract

The Carry Flag (CF)

- CF = 1 if there is a carry out from the msb (most significant bit) on addition, or there is a borrow into the msb on subtraction
- CF = 0 otherwise
- CF is also affected by shift and rotate instructions

The Parity Flag (PF)

- PF = 1 if the low byte of a result has an even number of one bits (even parity)
- PF = 0 otherwise (odd parity)

The Auxiliary Carry Flag (AF)

- AF = 1 if there is a carry out from bit 3 on addition, or there is a borrow into the bit 3 on subtraction
- AF = 0 otherwise
- AF is used in binary-coded decimal (BCD) operations

The Zero Flag (ZF)

- ZF = 1 for a zero result
- ZF = 0 for a non-zero result

The Sign Flag (SF)

- SF = 1 if the msb of a result is 1; it means the result is negative if you are giving a signed interpretation
- SF = 0 if the msb is 0

The Overflow Flag (OF)

- OF = 1 if signed overflow occurred
- OF = 0 otherwise

(Signed) Overflow

- Can only occur when adding numbers of the same sign (subtracting with different signs)
- Detected when carry into MSB is not equal to carry out of MSB
  - Easily detected because this implies the result has a different sign than the sign of the operands
- Programs can ignore the Flags!

Signed Overflow Example

```
  10010110
+ 10100011
  00111001
```

Carry in = 0, Carry out = 1
Neg+Neg=Pos
Signed overflow occurred
OF = 1 (set)

```
  00110110
+ 01100011
  10011001
```

Carry in = 1, Carry out = 0
Pos+Pos=Neg
Signed overflow occurred
OF = 1 (set)

Examples of No Signed Overflow

```
  10010110
+ 01100011
  11111001
```

Carry in = 0, Carry out = 0
Neg+Pos=Neg
No Signed overflow occurred
OF = 0 (clear)

```
  10010110
+ 11110011
  10001001
```

Carry in = 1, Carry out = 1
Neg+Neg=Neg
No Signed overflow occurred
OF = 0 (clear)

Unsigned Overflow

- The carry flag is used to indicate if an unsigned operation overflowed
- The processor only adds or subtracts - it does not care if the data is signed or unsigned!

```
  10010110
+ 11110011
  10001001
```

Carry out = 1
Unsigned overflow occurred
CF = 1 (set)