## TRANSFER Flags

| Name      | Comment       | Code      | Operation                        | O | D | I | T | S | Z | A | P | C |
|-----------|---------------|-----------|----------------------------------|---|---|---|---|---|---|---|---|---|---|
| MOV       | Move (copy)   | MOV Source, Dest | Dest := Source                   |   |   |   |   |   |   |   |   |   |   |
| XCHG      | Exchange      | XCHG S0, S1 | S0 := S1, S1 := S0              |   |   |   |   |   |   |   |   |   |   |
| STC       | Set Carry     | STC       | CF := 1                          | 1 |   |   |   |   |   |   |   |   |   |
| CLC       | Clear Carry   | CLC       | CF := 0                          | 0 |   |   |   |   |   |   |   |   |   |
| CMC       | Complement Carry | CMC     | CF := ¬CF                        | ± |   |   |   |   |   |   |   |   |   |
| STD       | Set Direction | STD       | DF := 1 (string op's downwards)  | 1 |   |   |   |   |   |   |   |   |   |
| CLD       | Clear Direction | CLD     | DF := 0 (string op's upwards)    | 0 |   |   |   |   |   |   |   |   |   |
| STI       | Set Interrupt | STI       | IF := 1                          | 1 |   |   |   |   |   |   |   |   |   |
| CLI       | Clear Interrupt | CLI    | IF := 0                          | 0 |   |   |   |   |   |   |   |   |   |
| PUSH      | Push onto stack | PUSH Source | DEC SP, [SP] := Source       |   |   |   |   |   |   |   |   |   |   |
| PUSHF     | Push flags    | PUSHF O, D, I, T, S, Z, A, P, C | 286+: also NT, IOPL            | ± | ± | ± | ± | ± | ± | ± | ± | ± | ± |
| PUSHA     | Push all general registers | PUSHA | AX, CX, DX, BX, SP, BP, SI, DI |   |   |   |   |   |   |   |   |   |   |
| POPOF     | Pop flags     | POPOF O, D, I, T, S, Z, A, P, C | 286+: also NT, IOPL            | ± | ± | ± | ± | ± | ± | ± | ± | ± | ± |
| POPA      | Pop all general registers | POPA | DI, SI, BP, SP, BX, DX, CX, AX |   |   |   |   |   |   |   |   |   |   |
| CBW       | Convert byte to word | CBW | AX := AL (signed)             |   |   |   |   |   |   |   |   |   |   |
| CWD       | Convert word to double | CWD | DX:AX := AX (signed)           |   |   |   |   |   |   |   |   |   |   |
| CWDE      | Convert word extended double | CWDE | EAX := AX (signed)          |   |   |   |   |   |   |   |   |   |   |
| IN         | Input         | IN Dest, Port | AL/AX/EAX := byte/word/double of specified port |   |   |   |   |   |   |   |   |   |   |
| OUT        | Output        | OUT Port, Source | Byte/word/double of specified port := AL/AX/EAX |   |   |   |   |   |   |   |   |   |   |

### ARITHMETIC Flags

| Name      | Comment       | Code      | Operation                        | O | D | I | T | S | Z | A | P | C |
|-----------|---------------|-----------|----------------------------------|---|---|---|---|---|---|---|---|---|---|
| ADD       | Add           | ADD Source, Dest | Dest := Dest + Source             | ± | ± | ± | ± | ± | ± | ± | ± | ± | ± |
| ADC       | Add with Carry | ADC Source, Dest | Dest := Dest + Source + CF         | ± | ± | ± | ± | ± | ± | ± | ± | ± | ± |
| SUB       | Subtract      | SUB Source, Dest | Dest := Dest - Source             | ± | ± | ± | ± | ± | ± | ± | ± | ± | ± |
| SBB       | Subtract with borrow | SBB Source, Dest | Dest := Dest - (Source + CF)      | ± | ± | ± | ± | ± | ± | ± | ± | ± | ± |
| MUL       | Multiply (unsigned) | MUL Op | Op := byte: AX := AL * Op  | if AH=0 then CF := 0 else CF := 1 | ± | ± | ± | ± | ± | ± | ± | ± | ± | ± |
| MUL       | Multiply (unsigned) | MUL Op | Op := word: DX:AX := AX * Op | if DX=0 then CF := 0 else CF := 1 | ± | ± | ± | ± | ± | ± | ± | ± | ± | ± |
| MUL       | Multiply (unsigned) | MUL Op | Op := double: EDX:EAX := EAX * Op | if EDX=0 then CF := 0 else CF := 1 | ± | ± | ± | ± | ± | ± | ± | ± | ± | ± |
| IMUL      | Signed Integer Multiply | IMUL Op | Op := byte: AL := AL * Op  | if AL insufficient then CF := 0 else CF := 1 | ± | ± | ± | ± | ± | ± | ± | ± | ± | ± |
| IMUL      | Signed Integer Multiply | IMUL Op | Op := word: DX:AX := AX * Op | if AX insufficient then CF := 0 else CF := 1 | ± | ± | ± | ± | ± | ± | ± | ± | ± | ± |
| IMUL      | Signed Integer Multiply | IMUL Op | Op := double: EDX:EAX := EAX * Op | if EAX insufficient then CF := 0 else CF := 1 | ± | ± | ± | ± | ± | ± | ± | ± | ± | ± |
| INC       | Increment      | INC Op | Op := Op + 1 (Carry not affected) | if CF := 0, OF := 0 else CF := 1, OF := 1 | ± | ± | ± | ± | ± | ± | ± | ± | ± | ± |
| DEC       | Decrement      | DEC Op | Op := Op - 1 (Carry not affected) | ± | ± | ± | ± | ± | ± | ± | ± | ± | ± |
| CMP       | Compare        | CMP S0, S1 | S0 := S1                          | ± | ± | ± | ± | ± | ± | ± | ± | ± | ± |
| SAL       | Shift arithmetic left (≡ SAL) | SAL Quantity, Dest | ± | ± | ± | ± | ± | ± | ± | ± | ± | ± |
| SAR       | Shift arithmetic right (≡ SAR) | SAR Quantity, Dest | ± | ± | ± | ± | ± | ± | ± | ± | ± | ± |
| RCL       | Rotate left through Carry | RCL Quantity, Dest | ± | ± | ± | ± | ± | ± | ± | ± | ± | ± |
| RCR       | Rotate right through Carry | RCR Quantity, Dest | ± | ± | ± | ± | ± | ± | ± | ± | ± | ± |
| ROL       | Rotate left | ROL Quantity, Dest | ± | ± | ± | ± | ± | ± | ± | ± | ± | ± |
| ROR       | Rotate right | ROR Quantity, Dest | ± | ± | ± | ± | ± | ± | ± | ± | ± | ± |

### LOGIC Flags

| Name      | Comment       | Code      | Operation                        | O | D | I | T | S | Z | A | P | C |
|-----------|---------------|-----------|----------------------------------|---|---|---|---|---|---|---|---|---|---|
| NEG       | Negate (two-complement) | NEG Op | Op := 0 - Op  | if Op = 0 then CF := 0 else CF := 1 | ± | ± | ± | ± | ± | ± | ± | ± | ± | ± |
| NOT       | Invert each bit | NOT Op | Op := ¬Op (invert each bit)   | ± | ± | ± | ± | ± | ± | ± | ± | ± | ± |
| AND       | Logical and   | AND Source, Dest | Dest := Dest ∧ Source       | 0 | ± | ± | ± | ± | ± | ± | ± | ± | ± |
| OR        | Logical or    | OR Source, Dest | Dest := Dest ∨ Source          | 0 | ± | ± | ± | ± | ± | ± | ± | ± | ± |
| XOR       | Logical exclusive or | XOR Source, Dest | Dest := Dest (xor) Source     | 0 | ± | ± | ± | ± | ± | ± | ± | ± | ± |
| SHL       | Shift logical left (≡ SAL) | SHL Quantity, Dest | ± | ± | ± | ± | ± | ± | ± | ± | ± | ± |
| SHR       | Shift logical right | SHR Quantity, Dest | ± | ± | ± | ± | ± | ± | ± | ± | ± | ± |

For more information see instruction specifications

Download latest version free of charge from www.jegerlehner.ch/intel This page may be freely distributed without cost provided it is not changed. All rights reserved
## Intel Assembler

### Flags

<table>
<thead>
<tr>
<th>Name</th>
<th>Comment</th>
<th>Code</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOP</td>
<td>No operation</td>
<td>NOP</td>
<td>No Operation</td>
</tr>
<tr>
<td>LEA</td>
<td>Load effective address</td>
<td>LEA</td>
<td>Dest := address of Source</td>
</tr>
<tr>
<td>INT</td>
<td>Interrupt</td>
<td>INT</td>
<td>Interrupts current program, runs spec. int-program</td>
</tr>
</tbody>
</table>

### MISC

<table>
<thead>
<tr>
<th>Name</th>
<th>Comment</th>
<th>Code</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOP</td>
<td>No operation</td>
<td>NOP</td>
<td>No Operation</td>
</tr>
<tr>
<td>LEA</td>
<td>Load effective address</td>
<td>LEA</td>
<td>Dest := address of Source</td>
</tr>
<tr>
<td>INT</td>
<td>Interrupt</td>
<td>INT</td>
<td>Interrupts current program, runs spec. int-program</td>
</tr>
</tbody>
</table>

### JUMPS (flags remain unchanged)

<table>
<thead>
<tr>
<th>Name</th>
<th>Comment</th>
<th>Code</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL</td>
<td>Call subroutine</td>
<td>CALL</td>
<td>Proc RET Return from subroutine</td>
</tr>
<tr>
<td>JMP</td>
<td>Jump</td>
<td>JMP</td>
<td>Dest</td>
</tr>
<tr>
<td>JE</td>
<td>Jump if Equal</td>
<td>JE</td>
<td>Dest (= JZ)</td>
</tr>
<tr>
<td>JZ</td>
<td>Jump if Zero</td>
<td>JZ</td>
<td>Dest (= JE)</td>
</tr>
<tr>
<td>JCXZ</td>
<td>Jump if CX Zero</td>
<td>JCXZ</td>
<td>Dest (= JCX)</td>
</tr>
<tr>
<td>JF</td>
<td>Jump if Parity (Parity Even)</td>
<td>JF</td>
<td>Dest (= JPF)</td>
</tr>
<tr>
<td>JFPE</td>
<td>Jump if Parity Even</td>
<td>JFPE</td>
<td>Dest (= JPF)</td>
</tr>
</tbody>
</table>

### JUMPS Unsigned (Cardinal)

<table>
<thead>
<tr>
<th>Name</th>
<th>Comment</th>
<th>Code</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>JA</td>
<td>Jump if Above</td>
<td>JA</td>
<td>Dest (= JNA)</td>
</tr>
<tr>
<td>JAE</td>
<td>Jump if Above or Equal</td>
<td>JAE</td>
<td>Dest (= JNAE)</td>
</tr>
<tr>
<td>JB</td>
<td>Jump if Below</td>
<td>JB</td>
<td>Dest (= JNE)</td>
</tr>
<tr>
<td>JBE</td>
<td>Jump if Below or Equal</td>
<td>JBE</td>
<td>Dest (= JNBE)</td>
</tr>
<tr>
<td>JNA</td>
<td>Jump if not Above</td>
<td>JNA</td>
<td>Dest (= JIC)</td>
</tr>
<tr>
<td>JNBE</td>
<td>Jump if not Above or Equal</td>
<td>JNBE</td>
<td>Dest (= JINB)</td>
</tr>
<tr>
<td>JN</td>
<td>Jump if not Below</td>
<td>JN</td>
<td>Dest (= JIC)</td>
</tr>
<tr>
<td>JNBE</td>
<td>Jump if not Below or Equal</td>
<td>JNBE</td>
<td>Dest (= JINB)</td>
</tr>
<tr>
<td>JC</td>
<td>Jump if Carry</td>
<td>JC</td>
<td>Dest</td>
</tr>
<tr>
<td>JNC</td>
<td>Jump if no Carry</td>
<td>JNC</td>
<td>Dest</td>
</tr>
</tbody>
</table>

### JUMPS Signed (Integer)

<table>
<thead>
<tr>
<th>Name</th>
<th>Comment</th>
<th>Code</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>JG</td>
<td>Jump if Greater</td>
<td>JG</td>
<td>Dest (= JNG)</td>
</tr>
<tr>
<td>JGE</td>
<td>Jump if Greater or Equal</td>
<td>JGE</td>
<td>Dest (= JNGE)</td>
</tr>
<tr>
<td>JL</td>
<td>Jump if Less</td>
<td>JL</td>
<td>Dest (= JNG)</td>
</tr>
<tr>
<td>JLE</td>
<td>Jump if Less or Equal</td>
<td>JLE</td>
<td>Dest (= JNLE)</td>
</tr>
<tr>
<td>JNS</td>
<td>Jump if no Sign</td>
<td>JNS</td>
<td>Dest (= JNLE)</td>
</tr>
<tr>
<td>JNO</td>
<td>Jump if no Overflow</td>
<td>JNO</td>
<td>Dest (= JNLE)</td>
</tr>
</tbody>
</table>

### General Registers:

- **EAX 386**
  
<table>
<thead>
<tr>
<th>EAX</th>
<th>AH</th>
<th>AL</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>24</td>
<td>16</td>
</tr>
<tr>
<td>15</td>
<td>8</td>
<td>0</td>
</tr>
</tbody>
</table>

- **EDX 386**
  
<table>
<thead>
<tr>
<th>EDX</th>
<th>DX</th>
<th>DL</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>24</td>
<td>16</td>
</tr>
<tr>
<td>15</td>
<td>8</td>
<td>0</td>
</tr>
</tbody>
</table>

- **ECX 386**
  
<table>
<thead>
<tr>
<th>ECX</th>
<th>CX</th>
<th>CL</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>24</td>
<td>16</td>
</tr>
<tr>
<td>15</td>
<td>8</td>
<td>0</td>
</tr>
</tbody>
</table>

- **EBX 386**
  
<table>
<thead>
<tr>
<th>EBX</th>
<th>BH</th>
<th>BL</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>24</td>
<td>16</td>
</tr>
<tr>
<td>15</td>
<td>8</td>
<td>0</td>
</tr>
</tbody>
</table>

### Example:

```
.DOSSEG ; Demo program
.MODELS SMALL
.STACK 1024

DATA
  VarW DB ? ; define Byte, any value
  VarW DW 1010b ; define Word, binary
  VarW2 DW 257 ; define Word, decimal
  VarD DD 0AFFFFh ; define Doubleword, hex
  S DB "Hello !", 0 ; define String

.CODE
  main: MOV AX, DGROUP ; resolved by linker
      MOV AX, DS ; AX ; init datasegment reg
      MOV AX, [VarB] ; init VarB
      MOV [VarD], -7 ; set VarD
      MOV AX, [VarW] ; addr of "H" of "Hello !"
      ADD AX, [VarW2] ; add VarW2 to AX
      MOV AX, 4C00h ; store AX in VarW2
      INT 21h ; back to system
      END main
```

### Flags

- **O**: Overflow result of signed op. is too large or small. 1 = overflow/underflow
- **S**: Sign sign of result. Reasonable for Integer only. 1 = neg. / 0 = pos.
- **Z**: Zero result of operation is zero. 1 = zero
- **A**: Aux. carry similar to Carry but restricted to the low nibble only
- **P**: Parity 1 = result has even number of set bits

### Status Flags (result of operations):

- **C**: Carry result of unsigned op. is too large or below zero. 1 = carry/borrow
- **O**: Overflow result of signed op. is too large or small. 1 = overflow/underflow
- **S**: Sign sign of result. Reasonable for Integer only. 1 = neg. / 0 = pos.
- **Z**: Zero result of operation is zero. 1 = zero
- **A**: Aux. carry similar to Carry but restricted to the low nibble only
- **P**: Parity 1 = result has even number of set bits

### Control Flags (how instructions are carried out):

- **D**: Direction 1 = string op's process down from high to low address
- **I**: Interrupt whether interrupts can occur. 1 = enabled
- **T**: Trap single step for debugging

Download latest version free of charge from [www.jegerlehner.ch/intel](http://www.jegerlehner.ch/intel)

This page may be freely distributed without cost provided it is not changed. All rights reserved