Intel Assembly

**Arithmetic Operations:**
- Addition
- Subtraction
- Multiplication
- Division
- Comparison
- Negation
- Increment
- Decrement

**Logic Operations:**
- AND
- OR
- XOR
- NOT
- shift
- rotate
- compare (test)
Arithmetic, Logic and Control Instructions

Arithmetic Operations

Addition, Increment, Add-with-carry and Exchange-and-add:
Contents of the rightmost 8 bits of the FLAGS register can change (+ Overflow) for arithmetic and logic instructions.

Flags include:
- Z (result zero)
- C (carry out)
- A (half carry out)
- S (result positive)
- P (result has even parity)
- O (overflow occurred)

`add al, [ARRAY + esi]`

`inc byte [edi]` ; Adds 1 to any reg/mem except seg

`adc ecx, ebx` ; Adds registers + Carry flag.

`xadd ecx, ebx` ; ecx=ecx+ebx, ebx=original ecx.
Arithmetic Operations

Subtraction, Decrement and Subtract-with-borrow:

- `sub eax, ebx ; eax = eax - ebx`
- `dec edi`
- `sbb ecx, ebx ; Subs registers - Carry flag.

Comparison:

Changes only the flag bits.
Often followed with a conditional branch:

- `cmp al, 10H`
- `jae LABEL1 ; Jump if equal or above.
- `jbe LABEL2 ; Jump if equal or below.
- `cmpxchg ecx, edx ; if ecx == eax, eax = edx else eax = ecx`
Arithmetic Operations

Multiplication and Division:
- `imul/idiv`: **Signed** integer multiplication/division.
- `mul/div`: **Unsigned**.

`al` always holds the *multiplicand* (or `ax` or `eax`).
Result is placed in `ax` (or `dx` and `ax` or `edx` or `eax`).

```
mul  bl           ; ax=al*bl (unsigned)
imul bx           ; dx|ax=ax*bx (signed)
imul cx, dx, 12H   ; Special, cx=dx*12H (signed only)
mul ecx           ; edx|eax=eax*ecx
```

*C* and *O* bits are cleared if most significant 8 bits of the 16-bit product are zero
(result of an 8-bit multiplication is an 8-bit result).

Division by zero and overflow generate errors.
Overflow occurs when a small number divides a large dividend.

```
div  cl           ; ah|al=ax/cl, unsigned quotient
               ; in al, remainder in ah
idiv cx         ; dx|ax=(dx|ax)/cx
```
Logic Operations

Allow bits to be set, cleared and complemented.
   Commonly used to control I/O devices.

Logic operations always clear the carry and overflow flags.

\[ \text{\textbf{AND}}: 0 \text{ AND} \text{ anything is 0.} \]
   Commonly used with a MASK to clear bits:

\[
\begin{array}{ccc}
\text{Operand} & \text{Mask} & \text{Result} \\
xxxxxxxx & 00001111 & \text{and} \ al, \ bl \ ; al=al \ \text{AND} \ bl \\
0000xxxx & \text{Result} \\
\end{array}
\]

\[ \text{\textbf{OR}}: 1 \text{ OR} \text{ anything is 1.} \]
   Commonly used with a MASK to set bits:

\[
\begin{array}{ccc}
\text{Operand} & \text{Mask} & \text{Result} \\
xxxxxxxx & 00001111 & \text{or} \ eax, \ 10 \ ; eax=eax \ \text{OR} \ 0000000AH \\
xxxx1111 & \text{Result} \\
\end{array}
\]
Logic Operations

❖ **XOR**: Truth table: 0110.
   
   Commonly used with a MASK to complement bits:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>XXXX</td>
<td>XXXX</td>
<td>Operand</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000</td>
<td>1111</td>
<td>Mask</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>xor ah, ch ; ah=ah XOR ch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XXXX</td>
<td>XXXX</td>
<td>Result</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

❖ **TEST**: Operates like the AND but doesn't effect the destination.
   
   Sets the Z flag to the complement of the bit being tested:

   ```assembly
   test al, 4 ; Tests bit 2 in al -- 00000100
   jz LABEL ; Jump to LABEL if bit 2 is zero.
   ```

❖ **BT**: Test the bit, **BTC**: Tests and complements...

❖ **NOT** (logical one's complement)

❖ **NEG** (arithmetic two's complement - sign of number inverted)

   ```assembly
   not ebx
   neg TEMP
   ```
**Logic Operations**

**Shift**: Logical shifts insert 0, arithmetic right shifts insert sign bit.

- `shl` eax, 1 ; eax is logically shifted left 1 bit pos.
- `sar` esi, cl ; esi is arithmetically shifted right

Double precision shifts (80386 and up):

- `shdr` eax, ebx, 12 ; eax shifted right by 12 and filled from the left with the right 12 bits of ebx.
- `shdl` ax, bx, 14

**Rotate**: Rotates bits from one end to the other or through the carry flag.

- `rol` si, 14 ; si rotated left by 14 places.
- `rcr` bl, cl ; bl rotated right cl places through carry.

Commonly used to operate on numbers wider than 32-bits:

- `shl` ax, 1 ; Original 48-bit number in dx, bx and ax.
  ; Shift ax left 1 binary place.
- `rcl` bx, 1 ; Rotate carry bit from previous shl into low order bit of bx.
- `rcl` dx, 1 ; Rotate carry bit from previous rcl in dx.
Bit/String Scan

Bit Scan Instruction (80386 and up):

Scan through an operand searching for a 1 bit.
Zero flag is set if a 1 bit is found, position of bit is saved in destination register.

- `bsl ebx, eax` ; eax scanned from the left.
- `bsr bl, cl` ; cl scanned from the right.

String Scan Instructions:

- `scasb/w/d` compares the al/ax/eax register with a byte block of memory and sets the flags.
- Often used with `repe` and `repne`
- `cmpsb/w/d` compares 2 sections of memory data.
Program Control Instructions

Conditional and Unconditional Jumps, Calls, Returns, Interrupts

Unconditional Jumps

- **Short jump**: PC-relative using two bytes (+127/-128 bytes).
  
  (PC-relative: constant added to eip).

```
NEXT:   add  ax, bx
       jmp  short NEXT  ;short keyword is optional.
```

- **Near jump**:
  
  Within segment (max of +/- 2G).

```
jmp  near eax  ;Jump to address given by eax.
jmp  [eax]    ;Jump to address given by [ax].
```

- **Far jump**:
  
  Four bytes give the offset and two bytes give a new segment address.
  The segment value refers to a descriptor in protected mode.

```
jmp  far LABEL  ;Jump to address given by LABEL.
```
Flow-of-Control Instructions

**Conditional Jumps:**

Test flag bits S, Z, C, P and O.

For unsigned numbers:

- `ja` ;Jump if above \((Z=0 \text{ and } C=0)\)
- `jbe` ;Jump if below or equal \((Z=1 \text{ or } C=1)\)

For signed numbers:

- `jl` ;Jump if < \((S<>0)\)
- `jge` ;Jump if >= \((S=0)\)

For either signed or unsigned:

- `jne` ;Jump if != \((Z=0)\)
- `je` or `jz` ;Jump if ==; or jump if zero \((Z=1)\)
- `jc` ;Jump if carry set \((C=1)\)

Test cx instead of flags:

- `jcxz` ;Jump if cx==0
- `jecxz` ;Jump if ecx==0
Flow-of-Control Instructions

Conditional Set instructions:

Set a byte to either 01H or 00H, depending on the outcome of condition under test.

\[ \text{setg} \ \text{al} \quad ; \text{Set al=1 if greater than (test Z==0 AND S==0)} \]
\[ \quad ; \text{else set al to 0} \]

\textit{LOOP} Instruction:

Combination of decrement ecx and \textit{jnz} conditional jump.

- Decrement ecx
- If ecx $\neq 0$, jump to label
- else fall through.

Example

\[ \text{loop} \ \text{LABEL} \quad ; \text{Jump if ecx } \neq 0 \]
\[ \text{loope} \quad ; \text{Jump if (Z = 1 AND ecx } \neq 0) \]
\[ \text{loopne} \quad ; \text{Jump if (Z = 0 AND ecx } \neq 0) \]