# GAS/Gnu Format - X86 Assembler Reference Card

**movl Src, Dest** Move 4-byte word to Destination

Source
- $Value, moves Value to Destination Immediate
- %Register, moves Value of Register to Destination
- (%Register), moves Value at Memory Address of %Register to Destination *

Destination
- %Register, moves the Source into the %Register
- (%Register), moves the Source to Memory Address in Register *

*Memory/Memory Operation with single Instruction not possible*
** Memory Addresses are calculated like leal, see leal for more Information*

**leal Src,Dest** Compute Memory Address without doing Memory referencing

Source
- Addressmode Expression,
- D(Rb,Ri,S) calculated as Dest = D + Rb + S*Ri where as Rb and Ri are Registers and all parameters are optional

Destination
- %Register, saves the calculated Address to Register

Two Operand Instructions

| | |
|---|---|
| **addl Src,Dest** | Adds two 4-byte words and stores it at Destination, *Dest = Dest + Src* |
| **subl Src,Dest** | Subtracts two 4-byte words and stores it at Destination, *Dest = Dest - Src* |
| **imull Src,Dest** | Multiplies two 4-byte words and stores it at Destination, *Dest = Dest * Src* |
| **sall Src,Dest** | Shifts Source by Destination left, *Dest = Dest << Src* Also called shll |
| **sarl Src,Dest** | Shifts Source by Destination arithmetic right, *Dest = Dest >> Src* |
| **shrl Src,Dest** | Shifts Source by Destination logical right, Dest = Dest >> Src |
| **xorl Src,Dest** | Computes bitwise XOR of Source and Destination, Dest = Dest ^ Src |
| **andl Src,Dest** | Computes bitwise AND of Source and Destination, Dest = Dest & Src |
| **orl Src,Dest** | Computes bitwise OR of Source and Destination, Dest = Dest \| Src |

One Operand Instructions

| | |
|---|---|
| **incl Dest** | Increments Destination by 1, *Dest = Dest + 1* |
| **decl Dest** | Decrements Destination by 1, *Dest = Dest - 1* |
| **negl Dest** | Negates Destination, *Dest = - Dest* |
| **notl Dest** | Negates Destination bitwise, *Dest = ~ Dest* |

Condition Codes:

Set by arithmetic operations and compare instructions, **CF** set if carry out from most significant bit, **ZF** set if result is Zero, **SF** set if result is negative, **OF** set if two's complement overflow,
Not set by leal!

| | |
|---|---|
| **cmpl b, a** | compares b with a like computing a-b without setting Destination. |
| **testl b, a** | computing a&b without setting Destination |

Jumps relative to setted Condition Codes

| | |
|---|---|
| **jmp Address** | Jumps to Address |
| **je / jne Address** | Jumps to Address when ZF is set / is not set, means b and a are Equal / Zero |
| **js Address** | Jumps to Address when SF is set / is not set |
| **jg Address** | Jumps to Address when ~(SF^OF)&~ZF is true, means b Greater a (Signed) |
| **jge Address** | Jumps to Address when ~(SF^OF) is true, means b Greater or Equal a (Signed) |
| **jl Address** | Jumps to Address when (SF^OF) is true, means b Less a (Signed) |
| **jle Address** | Jumps to Address when (SF^OF)\|ZF is true, means b Less or Equal a (Signed) |
| **ja Address** | Jumps to Address when ~CF&~ZF is true, means b Above a (unsigned) |
| **jb Address** | Jumps to Address when CF is true, means b Below a (unsigned) |