X86 Assembly/GAS Syntax

General Information

Examples in this article are created using the AT&T assembly syntax used in GNU AS. The main advantage of using this syntax is its compatibility with the GCC inline assembly syntax. However, this is not the only syntax that is used to represent x86 operations. For example, NASM uses a different syntax to represent assembly mnemonics, operands and addressing modes, as do some High-Level Assemblers. The AT&T syntax is the standard on Unix-like systems but some assemblers use the Intel syntax, or can, like GAS itself, accept both.

GAS instructions generally have the form mnemonic source, destination. For instance, the following **mov** instruction:

```
movb $0x05, %al
```

will move the value 5 into the register al.

Operation Suffixes

GAS assembly instructions are generally suffixed with the letters "b", "s", "w", "l", "q" or "t" to determine what size operand is being manipulated.

- b = byte (8 bit)
- s = short (16 bit integer) or single (32-bit floating point)
- w = word (16 bit)
- l = long (32 bit integer or 64-bit floating point)
- q = quad (64 bit)
- t = ten bytes (80-bit floating point)

If the suffix is not specified, and there are no memory operands for the instruction, GAS infers the operand size from the size of the destination register operand (the final operand).

Prefixes

When referencing a register, the register needs to be prefixed with a "%". Constant numbers need to be prefixed with a "$".

Address operand syntax

The up to 4 parameters of an address operand are presented in the syntax `displacement(base register, offset register, scalar multiplier)`. This is equivalent to `[base register + displacement + offset register * scalar multiplier]` in Intel syntax. Either or both of the numeric, and either of the register parameters may be omitted:

```
movl    -4(%ebp, %edx, 4), %eax  # Full example: load *(ebp - 4 + (edx * 4)) into eax
movl    -4(%ebp), %eax           # Typical example: load a stack variable into eax
movl    (%ecx), %edx             # No offset: copy the target of a pointer into a register
```