Целочисленные регисты (IA32)

Сравнение языков ассемблера Intel and AT&T

В синтаксисе Intel нет никаких префиксов регистра или префиксов констант. В AT&T, однако, регистры помечены '%', и константы помечены с '$'. В синтаксисе intel шестнадцатеричные или двоичные данные имеют suffix с 'h' и 'b' соответственно. Также, если первая шестнадцатеричная цифра - буква тогда, перед ней ставится '0'.

Пример:Example:

Intel Syntax		AT&T Syntax	
mov	eax,1	movl	$1,%eax
mov	ebx,0ffh	movl	$0xff,%ebx
int	80h	int	$0x80

Направление операндов в синтаксисе Intel противоположно от того из синтаксиса AT&T. В синтаксисе Intel первый операнд - место назначения, и второй операнд - источник, тогда как в синтаксисе AT&T первый операнд - источник, и второй операнд - место назначения.

Example:

Intel Syntax		AT&T Syntax	
instr	dest,source	instr	source,dest
mov	eax,[ecx]	movl	(%ecx),%eax

Memory Operands.

Операнды памяти, как замечено выше отличаются также. В синтаксисе Intel индексный регистр в '[' и']', тогда как в синтаксисе AT&T в '(' и')'.

Example:

Intel Syntax		AT&T Syntax	
mov	eax,[ebx]	movl	(%ebx),%eax
mov	eax,[ebx+3]	movl	3(%ebx),%eax

Форма AT&T для инструкций, включающих сложные операции, очень неясен по сравнению с синтаксисом Intel. Форма синтаксиса Intel - [base+index*scale+ offset]. Форма синтаксиса AT&T - смещение (база, индекс, масштаб).

Непосредственные используемые данные не должны , иметь префикс '$' в AT&T, когда используется для адресации

Example:

Intel Syntax		AT&T Syntax	
instr	[base+index*scale+offset]	instr	offset(base,index,scale),foo
mov	eax,[ebx+20h]	movl	0x20(%ebx),%eax
add	eax,[ebx+ecx*2h]	addl	(%ebx,%ecx,0x2),%eax
lea	eax,[ebx+ecx]	leal	(%ebx,%ecx),%eax
sub	eax,[ebx+ecx*4h-20h]	subl	-0x20(%ebx,%ecx,0x4),%eax

As you can see, AT&T is very obscure. [base+index*scale+disp] makes more sense at a glance than disp(base,index,scale).

Как Вы видите, AT&T очень неясен. [base+index*scale+disp] имеет больше смысла сразу, чем смещение (основа, индекс, масштаб).

Suffixes.

Как Вы, возможно, заметили, мнемоника синтаксиса AT&T имеет суффикс. Значение этого суффикса - значение размера операнда. 'l' long, 'w' word, и 'b' для байта. У синтаксиса intel есть подобные директивы для использования с операндами памяти, т.е. byte ptr, word ptr, dword ptr.

Example:

Intel Syntax	AT&T Syntax
`mov al,bl`	`movb %bl,%al`
`mov ax,bx`	`movw %bx,%ax`
`mov eax,ebx`	`movl %ebx,%eax`
`mov eax, dword ptr [ebx]`	`movl (%ebx),%eax`

Сравнение языков ассемблера Intel and AT&T

- **How does “mov (%ebx,%eHow does “mov (%ebx,%ecHow does “mov (%ebx,%ecx,4),%eax” work?**
- The complete memory addressing mode format in AT&T assembly is:
- offset(base, index, width)
- смещение (база, индекс,масштаб)
- offset = 0
- base = ebx
- index = ecx
- width = 4
- eax = *(uint32_t *)((uint8_t *)ebx + ecx * 4 + 0)
- **_ mov eax, [ebx+ecx*4] mov eax, [ebx+ecx*4] _ mov eax, [ebx+ecx*4] in NASM**

1. `x == (int) (float) x`

   Нет. Например, когда `x` — *TMax*.

2. `x == (int) (double) x`

   Да, поскольку `double` обладает большей точностью и диапазоном, нежели `int`.

3. `f == (float) (double) f`

   Да, поскольку `double` обладает большей точностью и диапазоном, нежели `float`.

4. `d == (float) d`

   Нет. Например, когда `d` равно `1e400`, в правой части получаем $+\infty$.

5. `f == - (-f)`

   Да, число с плавающей точкой отрицается простой инверсией знака.

6. `2/3 == 2/3.0`

   Нет, значение в левой части будет целым значением 0, тогда как значение в правой части будет приближением с плавающей точкой, равное $\frac{2}{3}$.

7. `(d >= 0.0) || ((d*2) < 0.0)`

   Да, поскольку умножение монотонно.

8. `(d+f) -d == f`

   Нет. Например, `d` равно $+\infty$, а `f` — 1, тогда левая часть будет *NaN*, а правая 1.