CSCI 223 Computer Organisation and Assembly Language

Intel and AT&T Syntax.

Intel and AT&T syntax Assembly language are very different from each other in appearance, and this will lead to confusion when one first comes across AT&T syntax after having learnt Intel syntax first, or vice versa. So lets start with the basics.

Prefixes.

In Intel syntax there are no register prefixes or immed prefixes. In AT&T however registers are prefixed with a '%' and immed's are prefixed with a '$'. Intel syntax hexadecimal or binary immed data are suffixed with 'h' and 'b' respectively. Also if the first hexadecimal digit is a letter then the value is prefixed by a '0'.

Example:

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

Direction of Operands.

The direction of the operands in Intel syntax is opposite from that of AT&T syntax. In Intel syntax the first operand is the destination, and the second operand is the source whereas in AT&T syntax the first operand is the source and the second operand is the destination. The advantage of AT&T syntax in this situation is obvious. We read from left to right, we write from left to right, so this way is only natural.

Example:

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

Memory Operands.

Memory operands as seen above are different also. In Intel syntax the base register is enclosed in '[' and ']' whereas in AT&T syntax it is enclosed in '(' and ')'.

Example:

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

The AT&T form for instructions involving complex operations is very obscure compared to Intel syntax. The Intel syntax form of these is segreg:[base+index*scale+disp]. The AT&T syntax form is %segreg:disp(base,index,scale).

Index/scale/disp/segreg are all optional and can simply be left out. Scale, if not specified and index is specified, defaults to 1. Segreg depends on the instruction and whether the app is being run in real mode or pmode. In real mode it depends on the instruction whereas in pmode its unnecessary. Immediate data used should not '$' prefixed in AT&T when used for scale/disp.

Example:

Intel Syntax		AT&T Syntax	
instr	foo,segreg:[base+index*scale+disp]	instr	%segreg:disp(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).

Suffixes.

As you may have noticed, the AT&T syntax mnemonics have a suffix. The significance of this suffix is that of operand size. 'l' is for long, 'w' is for word, and 'b' is for byte. Intel syntax has similar directives for use with memory operands, i.e. byte ptr, word ptr, dword ptr. "dword" of course corresponding to "long". This is similar to type casting in C but it doesnt seem to be necessary since the size of registers used is the assumed datatype.

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`