Differenze tra la sintassi AT&T e la sintassi Intel

Differenze principali

Operandi immediati:
AT&T: preceduti da $
Intel: nessuna indicazione
es: AT&T: push $4 - Intel: push 4

Registri:
AT&T: preceduti da %
Intel: nessuna indicazione
es: AT&T: mov %eax, %ebx - Intel: mov ebx, eax

Operandi di salto assoluti (al contrario dei relativi rispetto al PC) per jump/call:
AT&T: preceduti da *
Intel: nessuna indicazione

Destinazione sorgente:
AT&T: sorgente, destinazione
Intel: destinazione, sorgente
es: AT&T: addl $4, %eax - Intel: add eax, 4

Determinazione della dimensione degli operandi:
AT&T: ultimo carattere dell'istruzione
b: byte, w: word, l: doubleword
Intel: prefisso all'operando
byte ptr: byte, word ptr: word, dword ptr: doubleword
es: AT&T: movb foo, %al - Intel: mov al, byte ptr foo

Salti lunghi:
AT&T: lcall/ljmp $section, $offset
Intel: call/jmp section:offset

Ritorni lunghi:
AT&T: lret $sistemazione-stack
Intel: ret far sistemazione-stack

Nome delle istruzioni

Il nome delle istruzioni ha come suffisso un carattere che indica la dimensione degli operandi.
Le lettere 'b', 'w' e 'l' specificano operandi di tipo byte, word e long.
Se non e' presente alcun suffisso, viene presunto dalla dimensione dell'operando di destinazione.
Ad es. "mov %ax, %bx" e' equivalente a "movw %ax, %bx" e "mov $1, %ax" e' equivalente "movw $1, %ax".
La maggior parte delle istruzioni hanno lo stesso nome nella sintassi AT&T e in quella Intel. Ci sono pero' alcune eccezioni.

Queste riguardano le *estensioni di segno e di zero*:
Nella sintassi AT&T si ha:
movs.. , movz..

con al posto dei .. i suffissi
bl: da byte a long
bw: da byte a word
wl: da word a long

Altra eccezione si ha per le *istruzioni di conversione*:
Nella sintassi Intel:
‘cbw’ byte in ‘%al’ to word in ‘%ax’, con estensione di segno.
‘cwde’ word in ‘%ax’ to long in ‘%eax’, con estensione di segno.
‘cwd’ word in ‘%ax’ to long in ‘%dx:%ax’, con estensione di segno.
‘cdq’ dword in ‘%eax’ to quad in ‘%edx:%eax’, con estensione di segno.
Nella sintassi AT&T si hanno le corrispondenti istruzioni:
‘cbtw’, ‘cwtl’, ‘cwtd’ e ‘cltd’

Nomi dei registri

I nomi dei registri sono sempre preceduti da ’%’.
I registri del 386 sono:
8 registri a 32 bit ‘%eax’ (accumulatore), ‘%ebx’, ‘%ecx’, ‘%edx’, ‘%edi’, ‘%esi’, ‘%ebp’ (base pointer) e ‘%esp’ (stack pointer).
8 registri 16-bit che sono la parte bassa dei precedenti: ‘%ax’, ‘%bx’, ‘%cx’, ‘%dx’, ‘%di’, ‘%si’, ‘%bp’ e ‘%sp’.
8 registri 8-bit: ‘%ah’, ‘%al’, ‘%bh’, ‘%bl’, ‘%ch’, ‘%cl’, ‘%dh’ e ‘%dl’ (questi sono la parte alta e bassa dei registri ‘%ax’, ‘%bx’, ‘%cx’ e ‘%dx’)
6 registri per le sezioni, ‘%cs’ (code section), ‘%ds’ (data section), ‘%ss’ (stack section), ‘%es’, ‘%fs’ e ‘%gs’.
3 registri per il controllo del processore ‘%cr0’, ‘%cr2’ e ‘%cr3’.
6 registri di debug ‘%db0’, ‘%db1’, ‘%db2’, ‘%db3’, ‘%db6’ e ‘%db7’.
2 registri di test ‘%tr6’ e ‘%tr7’.
8 registri floating point stack ‘%st’ o equivalentemente ‘%st(0)’, ‘%st(1)’, ‘%st(2)’, ‘%st(3)’, ‘%st(4)’, ‘%st(5)’, ‘%st(6)’ e ‘%st(7)’.

Prefissi alle istruzioni

I prefissi sono utilizzati per modificare l’istruzione successiva.
Sono usati per ripetere le istruzioni su stringa, per poter distinguere le sezioni, per effettuare operazioni bus lock, e per determinare la dimensione dell’istruzione e degli operandi.
I prefissi devono essere subito prima delle istruzioni. Per esempio l’istruzione scas (scan string) viene ripetuta con repne scas.
Ecco una lista dei prefissi:
Per distinguere le sezioni:
‘cs’, ‘ds’, ‘ss’, ‘es’, ‘fs’, ‘gs’. Sono usati direttamente facendo sezione:locazione_di_memoria.
Per determinare le dimensioni:
‘data16’ e ‘addr16’ che cambiano le istruzioni/indirizzi a 32 bit in indirizzi a 16 bit.
Per effettuare operazioni bus lock:
‘lock’ disabilita gli interrupt durante l’esecuzione dell’istruzione. Si puo’ usare solo con certe istruzioni.
Per ripetere piu’ volte l’istruzione:
‘rep’, ‘repe’, ‘repne’ ripetono l’istruzione tante volte quanto e’ il valore contenuto in %ecx.

Indirizzamenti di memoria

L’indirizzamento indiretto nella sintassi Intel:
section:[base + index*scale + disp]
Viene tradotto nella sintassi AT&T in:
section:disp(base, index, scale)
Dove base e index sono i registri a 32 bit base e indice, disp e’ lo spiazzamento e scale, assume i valori 1, 2, 4, 8, moltiplicando l’indice per calcolare l’indirizzo dell’operando. Se non e’ specificato alcun scale viene assunto che sia 1. section specifica la sezione in cui si trova l’operando.
Ecco alcuni esempi dalla sintassi Intel a quella AT&T:
AT&T: ‘-4(%ebp)’, Intel: ‘[ebp - 4]’
la base e’ ‘%ebp’; disp e’ ‘-4’. La section non e’ specificata, ed e’ usata la sezione predefinita (‘%ss’ per gli indirizzi che hanno %ebp come registro base; anche index e scale sono omessi.
AT&T: ‘foo(,%eax,4)’, Intel: ‘[foo + eax*4]’
L’index e’ ‘%eax’ (moltiplicato per uno scale di 4); disp e’ ‘foo’. Tutti gli altri campi non sono stati usati. Il registro section predefinito qui e’ ‘%ds’.
AT&T: ‘foo(,1)’; Intel ‘[foo]’
Questa usa il valore puntato da foo come operando di memoria. Notare che la base e l’index sono stati omessi e c’e’ solo la ‘,’. Questa e’ un’eccezione sintattica.
AT&T: ‘%gs:foo’; Intel ‘gs:foo’
Questa seleziona il contenuto della variabile foo con section ‘%gs’.

Gestione delle istruzioni di salto

Le istruzioni di salto sono sempre ottimizzate per avere il minor spiazzamento possibile.
Questo e’ ottenuto usando salti con spiazzamenti a 8 bit. Se lo spiazzamento a 8 bit e’ insufficiente viene usato uno spiazzamento a 32 bit.
Notare che le istruzioni ‘jcxz’, ‘jecxz’, ‘loop’, ‘loopz’, ‘loope’, ‘loopnz’ e ‘loopne’ possono avere solo spiazzamenti a 8 bit, cosi’ se vengono usate possono dare origine a messaggi di errore.
L’assemblatore AT&T 80386 cerca di aggirare questo problema espandendo l’istruzione ‘jcxz foo’ in
jcxz cx_zero
jmp cx_nonzero
cx_zero: jmp foo
cx_nonzero:

Testo tradotto e arrangiato da syscalo

21-06-2000