Opcodes

(2nd April 2010)

Ange Albertini
http://corkami.blogspot.com

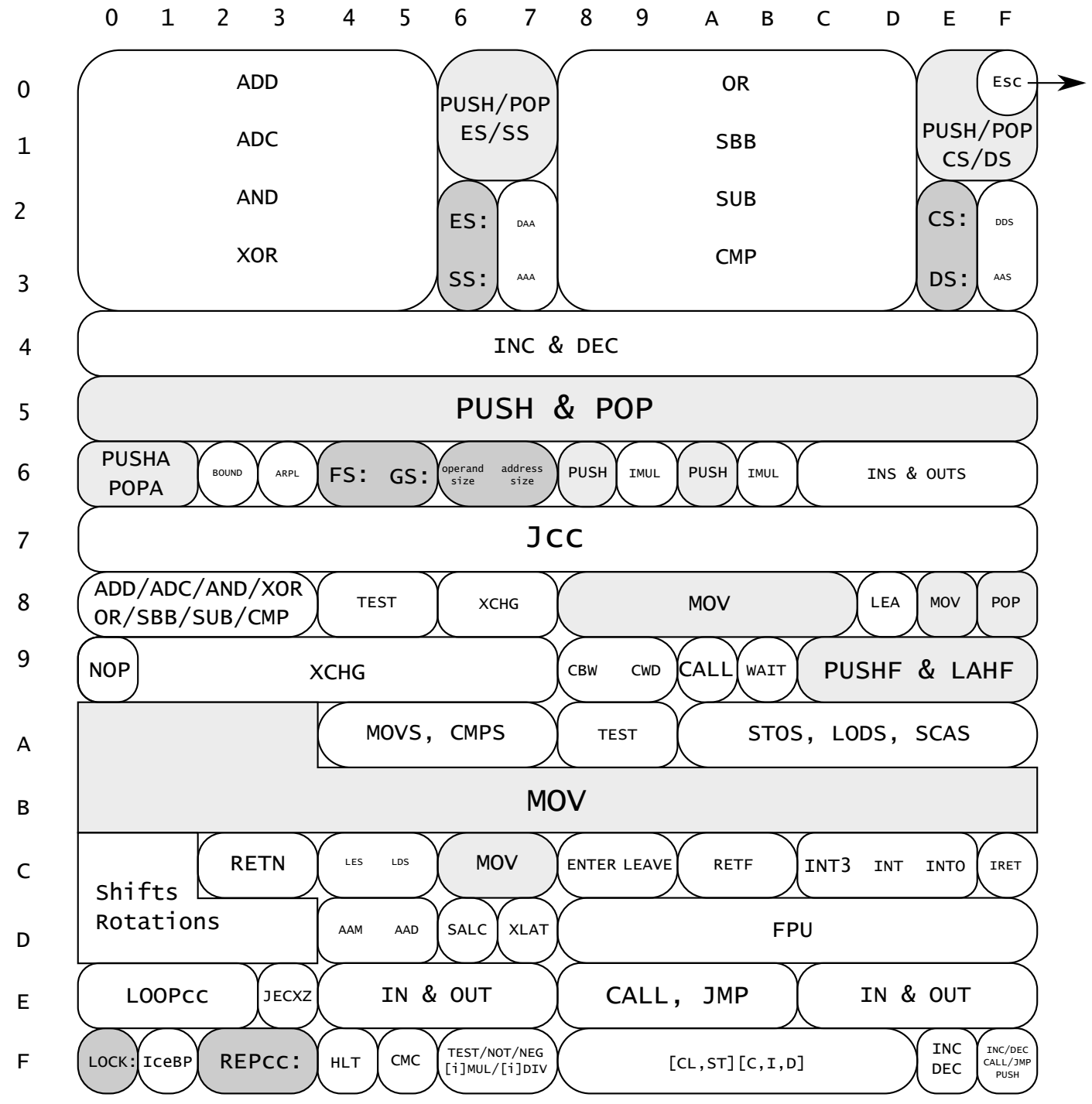
Table of contents

Changelog

2010/04/02 +perspective of two-byte opcodes, +common opcodes description

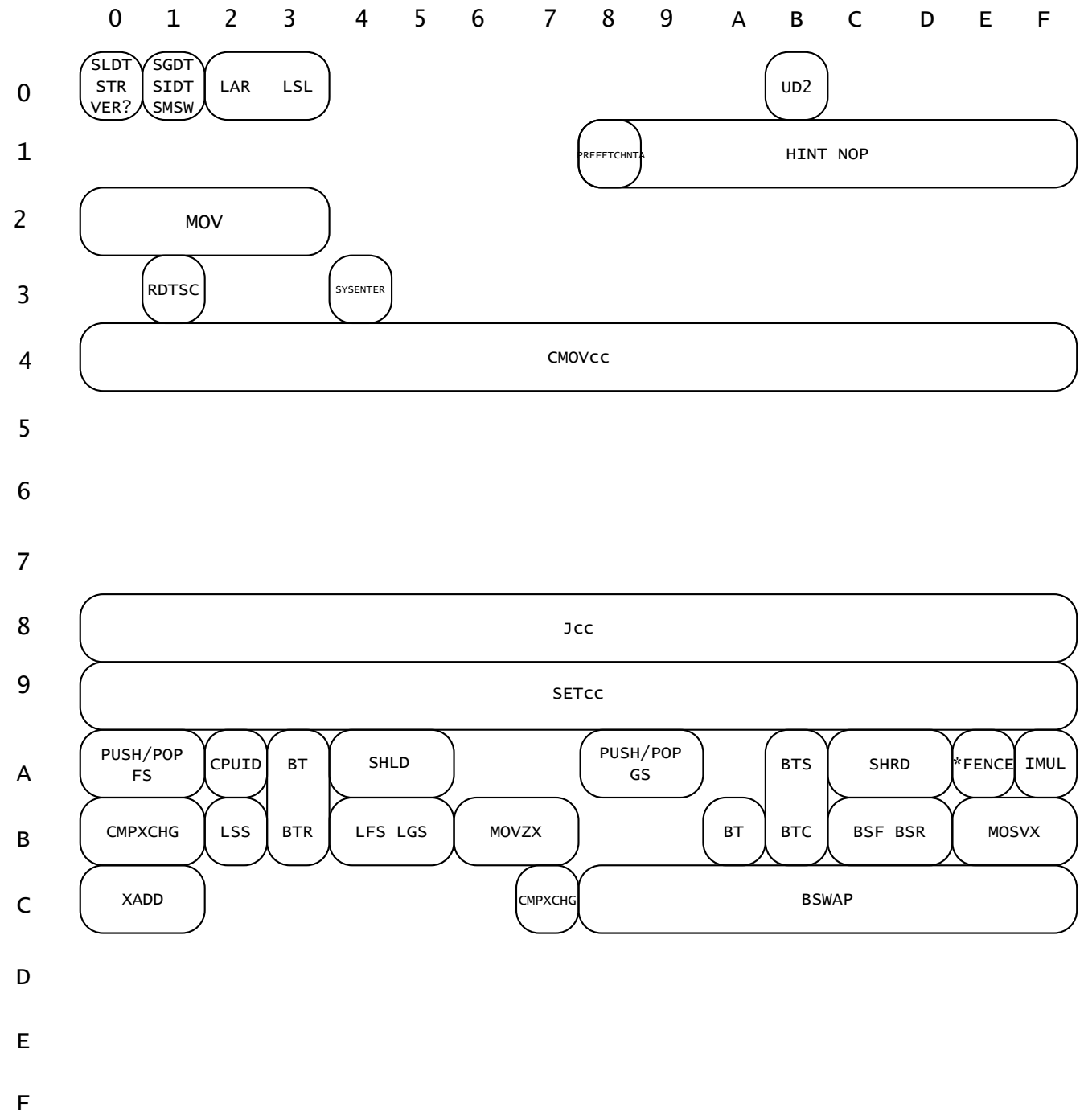
2010/03/28 +overview of one-byte opcodes

Opcodes
Overview of one-byte opcodes



Opcodes
a perspective of two-byte opcodes

most privileged and MMX/SSE opcodes
are intentionally missing



Common opcodes description

most privileged and FPU/MMX/SSE opcodes are intentionally missing

name	description
nop	does nothing. or maybe exchange *ax with *ax :)
fwait	waits for fpu stuff to be finished. junk nop usually
sfence	serializing stuff. used as nop (same for mfence, lfence)
prefetchnta	cpu hint - used as nop
hint nop	cpu hint, nop with operand - never triggers exceptions
mov	move (privileged when with dr* and cr*): *mov* `eax, 3` → `eax = 3`
cmov*cc*	mov on condition: `CF, eax, ebx = 0, 0, 3;` *cmovc* `eax, ebx` → `eax = 0`
lea	lea x, [y] = mov x,y: `eax = 3;` *lea* `eax, [eax * 4 + 203A]` → `eax = 2046`
movzx	mov and extend with zeroes: `al = -1;` *movzx* `ecx, al` → `ecx = ff`
movsx	mov and extend with the sign: `al = -3;` *movsx* `ecx, al` → `ecx = -3`
xchg	swap contents: `al, bl = 1, 2;` *xchg* `al, bl` → `al, bl = 2, 1`
movs	mov ds:[edi], es:[esi] , and inc (or dec) esi and edi
lods	mov *ax, es:[esi], and inc (or dec) esi
stos	mov ds:[edi], *ax and inc (or dec) edi
add	addition: `eax = 3;` *add* `eax, 3` → `eax = 6`
adc	add, with carry: `CF, eax = 1, 3;` *adc* `eax, 3` → `CF, eax = 0, 7`
xadd	add and exchange: `al, bl = 1, 2;` *xadd* `al, bl` → `al, bl = 3, 1`
sub	substraction: `eax = 6 ;` *sub* `eax, 3` → `eax = 3`
sbb	sub, with carry: `CF, eax = 1, 6 ;` *sbb* `eax, 3` → `eax = 2`
inc	=add 1: `eax = 0;` *inc* `eax` → `eax = 1`
dec	=sub 1: `eax = 7;` *dec* `eax` → `eax = 6`
neg	negative `al = 1;` *neg* `al` → `al = -1`
div	divide ax/dx:ax/edx:eax by operand. Quo/Rem are in al:ah/ax:dx/eax:edx
	`ax, bl = 35, 11;` *div* `bl` → `ax = 0203`
idiv	same, with sign
mul	multiply. same registers as div: `al, bl = 11, 3;` *mul* `bl` → `ax = 33`
imul	signed mul. has a 3 operands version: `eax = 11;` *imul* `eax, eax, 3` → `eax = 33`
aaa	ascii adjust after BCD addition: `ax, bx = 304, 307; add ax, bx;` *aaa* → `ax = 701`
aas	ascii adjust after substraction: `ax, bx = 1, 4; sub al, bl;` *aas* → `ax = 7`
aam	decimal to BCD: `ax = 35;` *aam* → `ax = 305`
aad	BCD to decimal: `ax = 305;` *aad* → `ax = 35`
daa	decimal adjust after addition: `ax, bx = 1234, 537; add ax, bx;` *daa* → `ax = 1771`
das	decimal adjust after substraction `ax, bx = 1771, 1234; sub ax, bx;` *das* → `ax = 537`
or	or: `eax = 1010b;` *or* `eax, 0110b` → `eax = 1110b`
and	and: `eax = 1010b;` *and* `eax, 0110b` → `eax = 0010b`
xor	exclusive or: `eax = 1010b;` *xor* `eax, 0110b` → `eax = 1100b`
not	logical not: `al = 1010b;` *not* `al` → `al = 11110101b`
rol	left rotation: `eax = 1010b;` *rol* `eax, 3` → `eax = 1010000b`
ror	right rotation: `al= 1010b;` *ror* `al, 3  ra al = 1000001b`
rcl	rol over carry: `CF, al = 1, 1010b;` *rol* `al, 3` → `al = 1010100b`
rcr	ror over carry: `CF, al = 1, 1010b, 1;` *rcr* `al, 3` → `al = 10100001b`
shl	shift left (=sal): `al = 1010b;` *shl* `al, 2` → `al = 101000b`
shr	shift right: `al = 1010b;` *shr* `al, 2` → `al = 10b`
sar	arithmetic shr (propagates sign): `al = -8;` *sar* `al, 2` → `al = -2`
shld	shift and concatenate: `ax, bx = 1111b, 010...0b;` *shld* `ax, bx, 3` → `ax = 1111010b`
shrd	`ax, bx = 1101001b, 101b;` *shrd* `ax, bx, 3` → `ax = 10100...001101b`
lds	loads register and segment: `[ebx] = 12345678, 0;` *lds* `eax, [ebx]` → `ds = 0; eax = 12345678`
	same with lss/les/lfs/lgs and the other segments
loop*cc*	dec ecx. jump if ecx is 0 and extra condition
rep*cc*:	repeat operation, decrease counter. stop if condition met or counter is 0
jecxz	jump if *cx is null
jmp	eip = operand
jmpf	cs:eip = operands
j*cc*	jump on condition

name	description
enter	= (push ebp/mov ebp, esp) op2+1 times, then esp -= op1:
	enter `4,1 = push ebp/mov ebp, esp/ push ebp/ mov ebp, esp/ esp -= 4`
leave	= mov esp, ebp/pop ebp
cmp	comparison by *sub*, discard result
cmps	cmp es:[esi], ds:[edi] and inc (or dec) esi and edi
scas	cmp *ax, es:[edi] and inc (or dec) edi
test	comparison by *and*, discard result
push	push on stack: *push* `12345678` → `esp -= 4 ; [esp] = 12345678`
pushf	push EFLAGS on stack
pusha	push eax/ecx/edx/ebx/(original) esp/ebp/esi/edi
pop	pop from stack: `[esp] = 12345678 ;` *pop* `eax` → `esp += 4 ; eax = 12345678`
popf	pop EFLAGS from stack
popa	pop edi/esi/ebp/.../ebx/edx/ecx/eax
smsw	eax=cr0 (non privileged)
lahf	ah=flag (CPAZS)
sahf	flag=ah
in	read port - privileged - vmware backdoor
ins	in es:[edi], dx; inc (or dec) edi
out	write port - privileged
outsd	out dx, [esi]; inc (or dec) esi
call	push eip of next instruction/eip = <operand>
callf	push cs, eip of next instruction/cs:eip = <operands>
ret	pop eip / esp += <operand>
retf	pop eip / pop cs
iret	pop eip / pop cs + pop eflags
cbw	extend signed value from al to ax: `al = 3;` *cbw* → `ax = 3`
cwd	extend signed value from ax to dx:ax: `ax = 3;` *cwd* → `dx = 0`
cwde	extend signed value from ax to eax: `ax = 3;` *cwde* → `eax = 3`
bsf	scan for the first bit set: `eax = 0010100b;` *bsf* `ebx, eax` → `ebx = 2`
bsr	same but from highest bit to lowest bit
bt	copy a specific bit to CF: `ax, bx = 00100b, 2;` *bt* `ax,bx` → `CF = 1`
	bts/btr/btc the same + set/reset/complement that bit
stc/d/i	set CF/DF (rep prefix)/IF (privileged)
clc/d/i	clear those flags
cmc	complement CF: `CF = !CF`
int	trigger interrupt <operand>
into	trigger interrupt 4 if OF is set
int3	trigger interrupt 3
xlat	al = [ebx + al]: `al, [ebx + 35] = 35, 75;` *xlatb* → `al = 75`
bound	int5 if op1 <[op2] && op1 > [op2 + size]: `eax, [ebx] = 136, [135, 143];` *bound* `eax, ebx` → `nothing`
opsize:	turns dword operand into word: `ecx = -1; 66 : inc ecx (=inc cx)` → `ecx = ffff0000`
addsize:	use 16b addressing mode: `67:add [eax], eax` → `add [bx + si], eax`
bswap	endian swapping `eax = 12345678h;` *bswap* `eax` → `eax = 78563412`
cmpxchg	if (op1 == *ax) op1 = op2 else *ax = op1: `al = 3; bl = 6;` *cmpxchg* `bl,cl` → `al = bl`
rdtsc	edx:eax = timestamp counter. for timing and anti-debug
sidt	store idt - used in anti-vmware: *sidt* `[eax]` → `[eax] = 7fff`
sgdt	store gdt: *sgdt* `[eax]` → `[eax] = 3fff`
sldt	store ldt (always 0?): *sldt* `eax` → `eax = 0`
cpuid	get cpu info (brand, features, ...)
lsl	get segment limit: `cx = cs;`*lsl* `eax, ecx` → `eax= -1`
str	store task register: *str* `ax` → `ax = 28 (XP) / 4000 (vmware)`
arpl	compares lower 2 bits and copy if inferior: `ax, bx = 1100b, 11b;`*arpl* `ax,bx` → `ax = 1111b`
lar	check descriptor and get some parameter if existing: `cx = cs;` *lar* `eax, ecx` → `eax = cff300`
ver*	check segment accessibility (and readability or writability): `cx = cs;` *verr* `cx` → `cf = 1`
sysenter	gateway to kernel: `eax, [esp], edx = 0, @return, esp;` *sysenter* → `eip = return;...`

name	description
set*cc*	operand = condition ? 1 : 0 `CF = 1;` *setc* `al` → `al = 1`
setalc	al = cf ? -1 : 0: `cf = 1;` *setalc* → `al = ff`
hlt	stops cpu. usually used to trigger PRIVILEGED_INSTRUCTION
IceBp	triggers SINGLE_STEP exception
ud1-ud2	invalid opcodes. used as exceptions triggers for ILLEGAL_INSTRUCTION
lock:	preserve memory content. picky prefix, mostly used to trigger exceptions