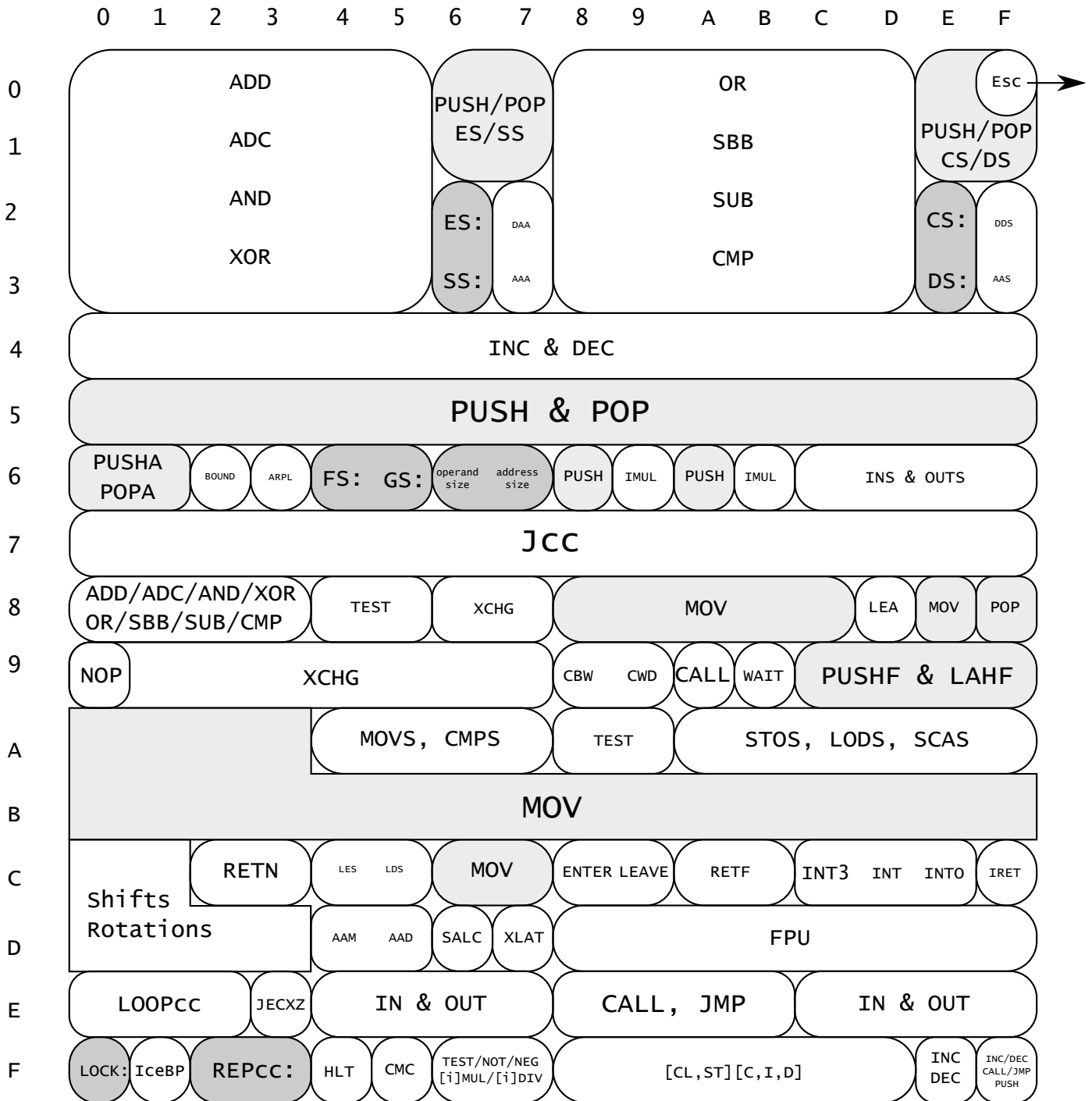


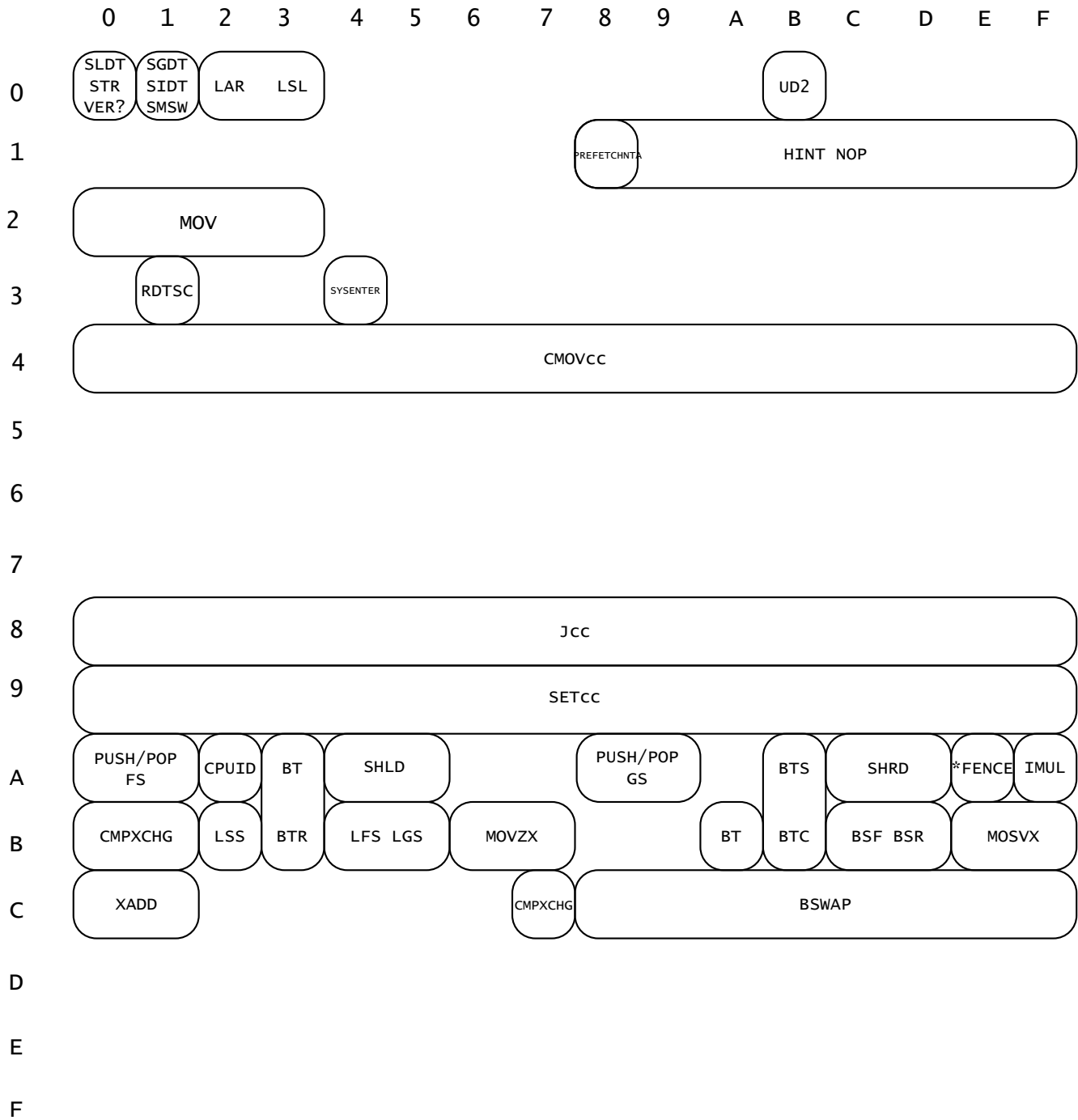
# Opcodes

## 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*): <i>mov eax, 3</i> → <i>eax = 3</i>
cmovcc	mov on condition: CF, <i>eax, ebx = 0, 0, 3</i> ; <i>cmovc eax, ebx</i> → <i>eax = 0</i>
lea	lea x, [y] = mov x,y: <i>eax = 3</i> ; <i>lea eax, [eax * 4 + 203A]</i> → <i>eax = 2046</i>
movzx	mov and extend with zeroes: <i>al = -1</i> ; <i>movzx ecx, al</i> → <i>ecx = ff</i>
movsx	mov and extend with the sign: <i>al = -3</i> ; <i>movsx ecx, al</i> → <i>ecx = -3</i>
xchg	swap contents: <i>al, bl = 1, 2</i> ; <i>xchg al, bl</i> → <i>al, bl = 2, 1</i>
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: <i>eax = 3</i> ; <i>add eax, 3</i> → <i>eax = 6</i>
adc	add, with carry: CF, <i>eax = 1, 3</i> ; <i>adc eax, 3</i> → CF, <i>eax = 0, 7</i>
xadd	add and exchange: <i>al, bl = 1, 2</i> ; <i>xadd al, bl</i> → <i>al, bl = 3, 1</i>
sub	subtraction: <i>eax = 6</i> ; <i>sub eax, 3</i> → <i>eax = 3</i>
sbb	sub, with carry: CF, <i>eax = 1, 6</i> ; <i>sbb eax, 3</i> → <i>eax = 2</i>
inc	=add 1: <i>eax = 0</i> ; <i>inc eax</i> → <i>eax = 1</i>
dec	=sub 1: <i>eax = 7</i> ; <i>dec eax</i> → <i>eax = 6</i>
neg	negative <i>al = 1</i> ; <i>neg al</i> → <i>al = -1</i>
div	divide ax/dx:ax/edx:eax by operand. Quo/Rem are in al:ah/ax:dx/eax:edx <i>ax, bl = 35, 11</i> ; <i>div bl</i> → <i>ax = 0203</i>
idiv	same, with sign
mul	multiply. same registers as div: <i>al, bl = 11, 3</i> ; <i>mul bl</i> → <i>ax = 33</i>
imul	signed mul. has a 3 operands version: <i>eax = 11</i> ; <i>imul eax, eax, 3</i> → <i>eax = 33</i>
aaa	ascii adjust after BCD addition: <i>ax, bx = 304, 307</i> ; <i>add ax, bx</i> ; <i>aaa</i> → <i>ax = 701</i>
aas	ascii adjust after subtraction: <i>ax, bx = 1, 4</i> ; <i>sub al, bl</i> ; <i>aas</i> → <i>ax = 7</i>
aam	decimal to BCD: <i>ax = 35</i> ; <i>aam</i> → <i>ax = 305</i>
aad	BCD to decimal: <i>ax = 305</i> ; <i>aad</i> → <i>ax = 35</i>
daa	decimal adjust after addition: <i>ax, bx = 1234, 537</i> ; <i>add ax, bx</i> ; <i>daa</i> → <i>ax = 1771</i>
das	decimal adjust after subtraction <i>ax, bx = 1771, 1234</i> ; <i>sub ax, bx</i> ; <i>das</i> → <i>ax = 537</i>
or	or: <i>eax = 1010b</i> ; <i>or eax, 0110b</i> → <i>eax = 1110b</i>
and	and: <i>eax = 1010b</i> ; <i>and eax, 0110b</i> → <i>eax = 0010b</i>
xor	exclusive or: <i>eax = 1010b</i> ; <i>xor eax, 0110b</i> → <i>eax = 1100b</i>
not	logical not: <i>al = 1010b</i> ; <i>not al</i> → <i>al = 11110101b</i>
rol	left rotation: <i>eax = 1010b</i> ; <i>rol eax, 3</i> → <i>eax = 1010000b</i>
ror	right rotation: <i>al = 1010b</i> ; <i>ror al, 3</i> <i>ra al = 1000001b</i>
rcl	rol over carry: CF, <i>al = 1, 1010b</i> ; <i>rol al, 3</i> → <i>al = 1010100b</i>
rcr	ror over carry: CF, <i>al = 1, 1010b, 1</i> ; <i>rcr al, 3</i> → <i>al = 10100001b</i>
shl	shift left (=sal): <i>al = 1010b</i> ; <i>shl al, 2</i> → <i>al = 101000b</i>
shr	shift right: <i>al = 1010b</i> ; <i>shr al, 2</i> → <i>al = 10b</i>
sar	arithmetic shr (propagates sign): <i>al = -8</i> ; <i>sar al, 2</i> → <i>al = -2</i>
shld	shift and concatenate: <i>ax, bx = 1111b, 010...0b</i> ; <i>shld ax, bx, 3</i> → <i>ax = 1111010b</i>
shrd	<i>ax, bx = 1101001b, 101b</i> ; <i>shrd ax, bx, 3</i> → <i>ax = 10100...001101b</i>
lds	loads register and segment: <i>[ebx] = 12345678, 0</i> ; <i>lds eax, [ebx]</i> → <i>ds = 0</i> ; <i>eax = 12345678</i> same with lss/les/lfs/lgs and the other segments
loopcc	dec ecx. jump if ecx is 0 and extra condition
repcc:	repeat operation, decrease counter. stop if condition met or counter is 0
jecz	jump if *cx is null
jmp	eip = operand
jmpf	cs:eip = operands
jcc	jump on condition

name	description
enter	= (push ebp/mov ebp, esp) op2+1 times, then esp -= op1: <i>enter 4,1</i> = push ebp/mov ebp, esp/ push ebp/ mov ebp, esp/ esp -= 4
leave	= mov esp, ebp/pop ebp
cmp	comparison by <i>sub</i> , 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 <i>and</i> , discard result
push	push on stack: <i>push 12345678</i> → 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 ; <i>pop eax</i> → 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: <i>al = 3; cbw</i> → ax = 3
cwd	extend signed value from ax to dx:ax: <i>ax = 3; cwd</i> → dx = 0
cwde	extend signed value from ax to eax: <i>ax = 3; cwde</i> → eax = 3
bsf	scan for the first bit set: <i>eax = 0010100b; bsf ebx, eax</i> → ebx = 2
bsr	same but from highest bit to lowest bit
bt	copy a specific bit to CF: <i>ax, bx = 00100b, 2; bt ax,bx</i> → 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]: <i>al, [ebx + 35] = 35, 75; xlatb</i> → al = 75
bound	int5 if op1 <[op2] && op1 > [op2 + size]: <i>eax, [ebx] = 136, [135, 143]; bound eax, ebx</i> → nothing
opsiz:	turns dword operand into word: <i>ecx = -1; 66: inc ecx (=inc cx)</i> → ecx = ffff0000
addsiz:	use 16b addressing mode: <i>67:add [eax], eax</i> → add [bx + si], eax
bswap	endian swapping <i>eax = 12345678h; bswap eax</i> → eax = 78563412
cmpxchg	if (op1 == *ax) op1 = op2 else *ax = op1: <i>al = 3; bl = 6; cmpxchg bl,cl</i> → al = bl
rdtsc	edx:eax = timestamp counter. for timing and anti-debug
sidt	store idt - used in anti-vmware: <i>sidt [eax]</i> → [eax] = 7fff
sgdt	store gdt: <i>sgdt [eax]</i> → [eax] = 3fff
sldt	store ldt (always 0?): <i>sldt eax</i> → eax = 0
cpuid	get cpu info (brand, features, ...)
lsl	get segment limit: <i>cx = cs;lsl eax, ecx</i> → eax= -1
str	store task register: <i>str ax</i> → ax = 28 (XP) / 4000 (vmware)
arpl	compares lower 2 bits and copy if inferior: <i>ax, bx = 1100b, 11b;arpl ax,bx</i> → ax = 1111b
lar	check descriptor and get some parameter if existing: <i>cx = cs; lar eax, ecx</i> → eax = cff300
ver*	check segment accessibility (and readability or writability): <i>cx = cs; verr cx</i> → cf = 1
sysenter	gateway to kernel: <i>eax, [esp], edx = 0, @return, esp; sysenter</i> → eip = return;...

---

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

---

Ange Albertini, 2010, cc by 3.0 [http://corkami.googlecode.com/files/opcodes\\_desc.pdf](http://corkami.googlecode.com/files/opcodes_desc.pdf) source