Intel x86 Assembly Language Cheat Sheet (Intel Syntax*)

Instruction	Effect	Examples
Copying Data		

mov dest, src	Copy src to dest (dest = src)	mov eax, 10
		mov eax, ecx
		mov [eax], ecx
		mov eax, [arr+4*ecx]

lea dest, addr	"Load Effective Address." Places the address of a memory reference** into dest	lea eax, [arr+4*ecx]
Arithmetic		
add dest, src	dest = dest + src	add esi, 10 add esi, [eax+4*edx+8]
sub dest,src	dest = dest – src	sub eax,ebx
mul reg	edx:eax = eax * reg	mul esi
div reg	edx = edx:eax mod reg ; eax = edx:eax / reg	div edi
inc dest	Increment destination (dest++)	inc eax
dec dest	Decrement destination (dest--)	dec eax
Function Calls		
call label	Push current eip onto stack, transfer control to label	call format_disk
ret	Pop eip off stack and transfer control to new eip	ret
push item	Push item (constant or register) to stack	push 32 push eax
pop reg	Pop item from stack; optionally store to register	pop eax pop

Bitwise Operations		
and dest,src	dest = src & dest	and eax, ebx
or dest,src	dest = src | dest	or eax, 02000h
xor dest,src	dest = src ^ dest	xor ebx, 0FFFFFFFFh
shl dest,count	dest = dest << count	shl eax, 2
shl dest,count	dest = dest >> count	shr eax, 4
Conditionals and Jumps		
cmp arg1,arg2	Computes (arg1 - arg2) and sets flags appropriately. Typically precedes the conditional jump instructions	cmp eax, 0
je label	Jump to label if arg1 == arg2	je endloop
jne label	Jump to label if arg1 != arg2	jne loopstart
jg label	Jump to label if arg1 > arg2	jg exit
jge label	Jump to label if arg1 greater than/equal to arg2	jge format_disk
jl label	Jump to label if arg1 < arg2	jl error
jle label	Jump to label if arg1 less than/equal to arg2	jle finish
test reg, imm	Computes (reg & imm) and sets flags appropriately; typically immediately precedes the jz or jnz instructions	test eax, 0FFFFh
jz label	Jump to label if bits were not set (“zero”)	jz looparound
jnz label	Jump to label if bits were set (“not zero”)	jnz error
jmp label	Unconditional relative jump	jmp exit

jmp reg	Unconditional absolute jump; arg is a register	jmp eax
ljmp segment,offs	Unconditional absolute far jump	ljmp $0x10,$0
Miscellaneous		
nop	No-op (opcode 0x90)	nop
hlt	Halt the CPU	hlt

*x86 assembly has two syntaxes: AT&T syntax and Intel syntax. AT&T syntax is used by GCC and is hence more common in introductory courses; Intel syntax tends to be more readable and is used by assemblers such as masm and nasm, as well as IDA. The largest difference is that Intel syntax uses “dest,src” argument order, while AT&T syntax uses “src, dest”

** Many arguments can be memory references. A memory reference takes the form [offset+reg1+step*reg2], and is equivalent to writing *(offset+reg1+step*reg2) in C. “step” can be 1, 2, 4, or 8

Constant (decimal or hex): $10 or $0xff Fixed address: (2000) or (0x1000+53)
Register: %eax %bl Dynamic address: (%eax) or 16(%esp)

32-bit registers: eax, ebx, ecx, edx, esi, edi, esp, ebp
16-bit registers: ax, bx, cx, dx, si, di, sp, bp
8-bit registers: al, ah, bl, bh, cl, ch, dl, dh

Special registers:

esp is the stack pointer, and is incremented and decremented by the push and pop instructions.
eip is the instruction pointer, the address of the currently executing instruction