nasm x86 Assembly Quick Reference ("Cheat Sheet")

Instructions

Mnemonic	Purpose	Examples
mov *dest,src*	Move data between registers, load immediate data into registers, move data between registers and memory.	mov eax,4 ; Load constant into eax mov ebx,eax ; Copy eax into ebx mov ebx,[123] ; Copy ebx to memory address 123
push *src*	Insert a value onto the stack. Useful for passing arguments, saving registers, etc.	push ebp
pop *dest*	Remove topmost value from the stack. Equivalent to "mov *dest,*[esp] add esp,4"	pop ebp
call *func*	Push the address of the next instruction and start executing func.	call print_int
ret	Pop the return program counter, and jump there. Ends a subroutine.	ret
add *src,dest*	*dest=dest+src*	add eax,ebx # Add ebx to eax
mul *src*	Multiply eax and *src* as unsigned integers, and put the result in eax. High 32 bits of product go into eax.	mul ebx #Multiply eax by ebx
jmp *label*	Goto the instruction *label*:. Skips anything else in the way.	jmp post_mem ...

Stack Frame

(example without ebp or local variables)

Contents	**off esp**
caller's variables	[esp+12]
Argument 2	[esp+8]
Argument 1	[esp+4]
Caller Return Address	[esp]

```
my_sub: # Returns first argument
  mov eax,[esp+4]
  ret
```

(example when using ebp and two local variables)

Contents	**off ebp**	**off esp**
caller's variables	[ebp+16]	[esp+24]
Argument 2	[ebp+12]	[esp+20]
Argument 1	[ebp+8]	[esp+16]
Caller Return Address	[ebp+4]	[esp+12]
Saved ebp	[ebp]	[esp+8]
Local variable 1	[ebp-4]	[esp+4]
Local variable 2	[ebp-8]	[esp]

```
my_sub2: # Returns first argument
  push ebp  # Prologue
  mov ebp, esp
  mov eax, [ebp+8]
  mov esp, ebp  # Epilogue
```

		post_mem:
cmp *a,b*	Compare two values. Sets flags that are used by the conditional jumps (below).	cmp eax,10
jl *label*	Goto *label* if previous comparison came out as less-than. Other conditionals available are: jle (<=), je (==), jge (>=), jg (>), jne (!=), and many others.	jl loop_start ; Jump if eax<10

```
 pop ebp
 ret
```

Constants, Registers, Memory

"12" means decimal 12; "0xF0" is hex.
"some_function" is the address of the first instruction of a label.
Memory access (use register as pointer): "[esp]". Same as C "*esp".
Memory access with offset (use register + offset as pointer): "[esp+4]". Same as C "*(esp+4)".
Memory access with scaled index (register + another register * scale): "[eax + 4*ebx]". Same as C "*(eax+ebx*4)".

Registers

esp is the stack pointer
ebp is the stack frame pointer
Return value in eax
Arguments are on the stack
Free for use (no save needed):
   eax, ebx, ecx, edx
Must be saved:
   esi, edi, ebp, esp
8 bit: ah (high 8 bits) and al (low 8 bits)
16 bit: ax
32 bit: eax
64 bit: rax

The Intel Software Developer's Manuals are incredibly long, boring, and complete--they give all the nitty-gritty details. Volume 1 lists the processor registers in Section 3.4.1. Volume 2 lists all the x86 instructions in Section 3.2. Volume 3 gives the performance monitoring registers in Section. For Linux, the System V ABI gives the calling convention on page 39. Also see the Intel hall of fame for historical info. Sandpile.org has a good opcode table. Ralph Brown's Interrupt List is the definitive reference for all interrupt functions. See just the BIOS interrupts for interrupt-time code.

O. Lawlor, ffosl@uaf.edu
Up to: Class Site, CS, UAF