Intel opcodes

Basic building blocks of assembly code

Introduction

Good sources:

- https://en.wikibooks.org/wiki/X86_Assembly
- https://cs.brown.edu/courses/cs033/docs/guides/x64_cheatsheet.pdf

- Beware that not all x86 operations and opcodes are present in 64 bit mode.
- Remember sizes for q, l, w, and b are 8, 4, 2, and 1 byte respectively.

Optimization and Instruction Cycle Times

- Check out Agner Fog's awesome work:
- https://www.agner.org/optimize/

- The instruction times alone are worth the effort:
- https://www.agner.org/optimize/instruction_tables.pdf

Load Effective Address

Generic opcode	Examples / Notes
lea	leaq # know how to compute in the various modes as well as how to do fast math

Move Instructions

Generic opcode	Examples / Notes
mov	movq # know the modes as well!
movs	movswl # sign extend
movz	movzbl #zero extend

Conditional Move Instructions

Generic opcode	Examples / Notes
cmov	cmove # move if zero flag is set: ZF=1
	cmovge # greater or equal zero: SF=OF
	Many, many others. Source is register or memory, destination is register.

Function Related

Generic opcode	Examples / notes
call	call some_function # call a function by name call *%rax # %rax holds the address of a function call *(%rax) # memory at %rax holds the address # of a function
enter	# slower than the equivalent instruction sequence
leave	# might also be slower
ret	# %rsp had better point to the return address

Stack Related

Generic opcode	Examples / notes
push	pushq %rbp
pop	popq %rbp

Setting & Saving Condition Flags

Opcode	Examples / Notes
test	testl %eax %eax # AND
cmp	# subtract – beware of order!

Opcode	Examples / Notes
set	setnz %sil # put a zero in sil when the result was a zero, # otherwise a 1 setgt %r8b # put a 1 in the lower byte of r8 when the result # was greater than, otherwise put a zero

Some Jump Instructions

Opcode	Examples / Notes
jmp	jmp cleaup # unconditional (cleanup is a label somewhere in the code) # this is a direct jump jmp *%rax # jump to address held in rax (indirect) jmp *(%rax) # jump to address in memory at location held in rax
jne	jne loop_start # jump not equal (loop_start is a label in the code # somewhere) Must be direct.
jz	jz done_here # jump if zero (same as je) to label done_here. # all conditional jumps must be direct

Shifting

Opcode	Examples / Notes
shl	shlq $1, %rax #shift left. Note ALL shifts are by an immediate or by register cl (lowest 8 bits of rcx)
sal	# same as shl
shr	# shift in a zero
sar	# shift in the current sign bit

 # All single bit shifts put the bit shifted out into the C flag to let us know what got shifted out

Bitwise

Opcode	Examples / Notes
not	not %rax # bitwise complement
and	andq $1, %rsi # bitwise and
or	orq %rdi, %rax # bitwise or
xor	xorl %eax, %eax # exclusive or (can be used to selectively invert bits, also fastest way to clear a register)

Simple Math

Opcode	Examples / Notes
add	addq %rcx, %rax
adc	adcq $0, %rax # add zero plus the carry flag to %rax
sub	subq %rcx, %rdi
inc	incq %rsi #slightly faster than adding one to reg or mem
dec	decq %rcx # slightly faster than subtracting one
neg	neg %rax # arithmetic negation (2's complement)

Complex Math

Opcodes	Notes
mul, imul	# unsigned, signed. Some versions requires double size storage for the result; stored in %rdx:%rax
div , idiv	#unsigned, signed. Gives two results; quotient in %rax and remainder in %rdx

```
# We can use the "normal" version of imul and mul to scale subscripts:
imulq $24, %rsi        # turn rsi from a structure array subscript into a byte offset
                       # for a 24 byte structure
imulq $24, %rsi, %rax     #put 24 * rsi into rax
```

Supporting idiv and sign extension

Opcodes	Notes
cltd	# convert signed long to signed double long. Use the sign bit of %eax to set %edx. The two registers together (%edx:%eax which is %edx as the high 32 bits, %eax as the low 32 bits) are the 64 bit equivalent to the value in %eax
cwtd cqto	#convert word to doubleword(sign extend %ax into %dx:%ax) #convert quad to octword (sign extend %rax into %rdx:%rax)
cbtw cwtl cltq	#convert byte to word (sign extend %al into %ax) #convert word to long (sign extend %ax into %eax) #convert long to quad (sign extend %eax to %rax) #these 3 are faster than movs instructions (movsbw, movswl, movslq) but are wired to particular registers

An Enticing Throwback

loop

- Goes back to the 8088 chip
- The same as DEC CX followed by JNZ, only slower (to allow restart on page fault)
- There is an awesome discussion on Stack Overflow about this instruction at:

https://stackoverflow.com/questions/35742570/why-is-the-loop-instruction-slow-couldnt-intel-have-implemented-it-efficiently