Contents

IA32 & x64

```
mov  ; move
xchg ; exchange
add  ; add
sub  ; subtract
imul ; signed mul
mul  ; unsigned mul
inc  ; increment
dec  ; decrement
neg  ; negate
cmp  ; compare
lea  ; load effective address
test ; AND operands and set flags

and ; and
or  ; or
xor ; exclusive or
not ; not

push ; push onto stack
pop  ; pop from stack

sar ; shift arithmetic right
shl ; shift logical left
shr ; shift logical right

jmp                                     ; unconditional jump
j{e, ne, le, g, ge} signed              ; signed jump
; equal, not equal, less than or equal, greater, greater than or equal
j{b, be, a, ae} unsigned (below, above) ; unsigned jump
```

```
; e.g. jle

call ; call subroutine
ret  ; return from subroutine
```

IA32

Registers

- General purpose
  - eax - accumulator
  - ecx
  - edx
- Non-volatile
  - ebx
- Memory Addressing
  - esi
  - edi
  - ebp - frame pointer
  - esp - stack pointer
  - eflags - flags [status register]
  - eip - instruction pointers [pc]

Function Calling

```
; f(int i) {
;    int x;
;    int y;
;    ...
; }
f: push ebp      ; save ebp
   mov ebp, esp  ; epb = new stack frame
   sub esp, 8    ; allocate space for locals (x, y)
   ; function body
   mov esp, ebp  ; restore esp
   pop ebp       ; return previous stack frame
   ret
```

pointers	top of stack	
	i	pushed parameter

pointers	top of stack	
	return address	
ebp ->	saved ebp	
	x	local variable
esp ->	y	local variable

x64

Registers

- General purpose
  - rax - accumulator / return register
  - rcx - parameter 1 MS / parameter 4 GNU
  - rdx - parameter 2 MS / parameter 3 GNU / return reg 2 GNU
  - r8 - parameter 3 MS / paramter 5 GNU
  - r9 - parameter 4 MS / parameter 6 GNU
  - r10
  - r11
- Non volatile
  - rbx
  - r12
  - r13
  - r14
  - r15
- Memory Addressing
  - rsi - parameter 2 GNU
  - rdi - parameter 1 GNU
  - rbp - frame pointer
  - rsp - stack pointer
  - rflags - status register

Function Calling

Simple Function

```
; f(int i) {
;    int x;
```

```
;    int y;
;    ...
; }
f: push r12    ; save non-volatile registers for use
   push r13    ; save non-volatile registers for use
   ; Alternatively, use volatile registers
   ; function body
   pop r12
   pop r13
   ret
```

pointers	top of stack	
rbp ->		end of stack frame from previous function
	shadow r9	shadow space from previous function
	shadow r8	shadow space from previous function
	shadow rdx	shadow space from previous function
	shadow rcx	shadow space from previous function
rsp ->	return address	

Complex Function

```
; f(int i) {
;    int x;
;    int y;
;    f(i);
;    ...
; }
f: push rbp     ; save frame pointer
   mov rpb, rsp
   push r12     ; save non-volatile registers
   push r13     ; save non-volatile registers
   sub rsp, 32  ; allocate shadow space for function calling
   call g
   add rsp, 32  ; deallocate shadow space
   ; function body
   pop r12
   pop r13
   pop rbx
   ret
```

pointers	top of stack	
	shadow r9	shadow space from previous function
	shadow r8	shadow space from previous function
	shadow rdx	shadow space from previous function
	shadow rcx	shadow space from previous function
	return address	
rbp ->	saved rbp	
	saved r12	
	saved r13	
	shadow r9	
	shadow r8	
	shadow rdx	
rsp ->	shadow rcx	