Operands

Kinds → Immediate (constant)
→ Store it in a register
→ Store it in memory.

Immediate

Syntax → $ Imm

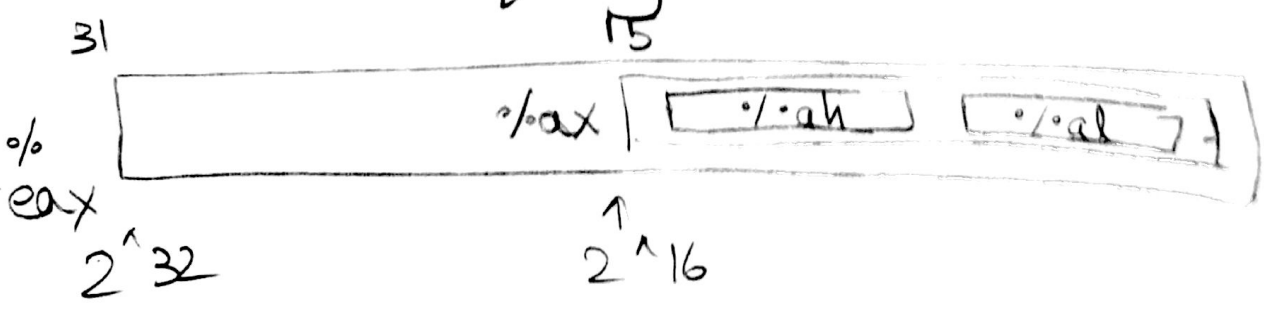
Value → Imm

Eg: $67, $-400
$0x7b.

Register → denotes the value stored in a register.

Syntax → E_a (%R)
%eax

Value → R[E_a]



③ Memory → We access some memory location according to some computed value

a) Immediate

Syntax → Imm.

Value → M [Imm]

↑

Since we view the memory as a large array of bytes.

b) Indirect.

Syntax → (Ea)

Value → M [[R (Ea)]]

Value stored in the address stored in the register.

c) Base + Displacement

Syntax → Imm (Ea)

Value → M [Imm + R[Ea]]

Value is stored in the address

Imm + R[Ea].

Syntax → Imm (Ea, Eb)

Value → M [Imm + R(Ea) + R(Eb)]

9 (%eax, %edx)

M [9 + 0x100 + 0x3]

M [0x10C] → 0x11

---

d) Scaled Indexing

(, Ea, S) → M [R[Ea] * S]

Imm (, Ea, S) → M (Imm + R[Ea] * S)

Imm (Ea, Ei, S) → M (Ea + Ei * S)

Address	Value
0x100	0xFF
0x104	0xAB
0x108	0x13
0x10C	0x11

Registers	Value
%eax	0x100
%ecx	0x1
%edx	0x3

① Immediate

Operand	Value
$0x108	0x108

② Register

Operand	Value
%eax	0x100
%ax	0x100
%al	0x0

③ Immediate (Memory).

Operand	Value
0x104	0xAB

④ Indirect

Operand	Value
(%eax)	0xFF

M[0x100]

⑤ Base + Displacement

4(%eax)

M[4 + 0x100]

M[0x104] → 0xAB

Data Movement

Syntax

movW S, D.

⇒ Move a 16-bit word from S to D.

movb , movl

Source → Immediate, Stored in a register
Stored in some memory location

Destination → Register, Memory ~~data~~ location

5 possible combinations

Imm to Register	movl	$0x4, %eax
Reg to Reg.	movl	%eax, %edx
Imm to Memory	movl	$-20, (%ecx)
Memory to Register		
Register to Memory.		

(! Cannot move from memory location to memory location !)

Why? RAM modes

Arithmetic & Logical Operations

Unary → w, l, b

INC	D	D ← D + 1
DEC	D	D ← D − 1
NEG	D	D ← −D
NOT	D	D ← ~D

Binary

ADD	S, D	D ← D + S
SUB	S, D	D ← D − S
IMUL		D ← D * S
XOR		D ← D ^ S
OR		D ← D | S
AND		D ← D & S

Address	Value
0x100	0xFF
0x104	0xAB

addl %ecx, (%eax)

Destination → 0x100

Register	Value
%eax	0x100
%ecx	0x1
%edx	0x3

Value → 0xFF + 0x1

→ 0x100

subl %edx, 4(%eax)

Destination → 0x104

Value → 0xAB − 0x3

→ 0xA8

Load Effective Address

leal S, D

Clue → Basically it's in the name!

Contrast with

movl 4 (%ebx), %eax

↓

M[4 + 0x100]

M[0x104]

Copies the value at 0x104 to %eax.

%eax ⇒ 0x6

Address	Value
0x100	0x5
0x104	0x6

Register	Value
%eax	0
%ebx	0x100

leal 4 (%ebx), %eax

↓

M[0x104]

Copies 0x104 into %eax.

%eax ⇒ 0x104.

Stack operations.

- pushl S → push double word S onto stack.
- popl D → pop double word off stack onto some location D.

pushl %.eax

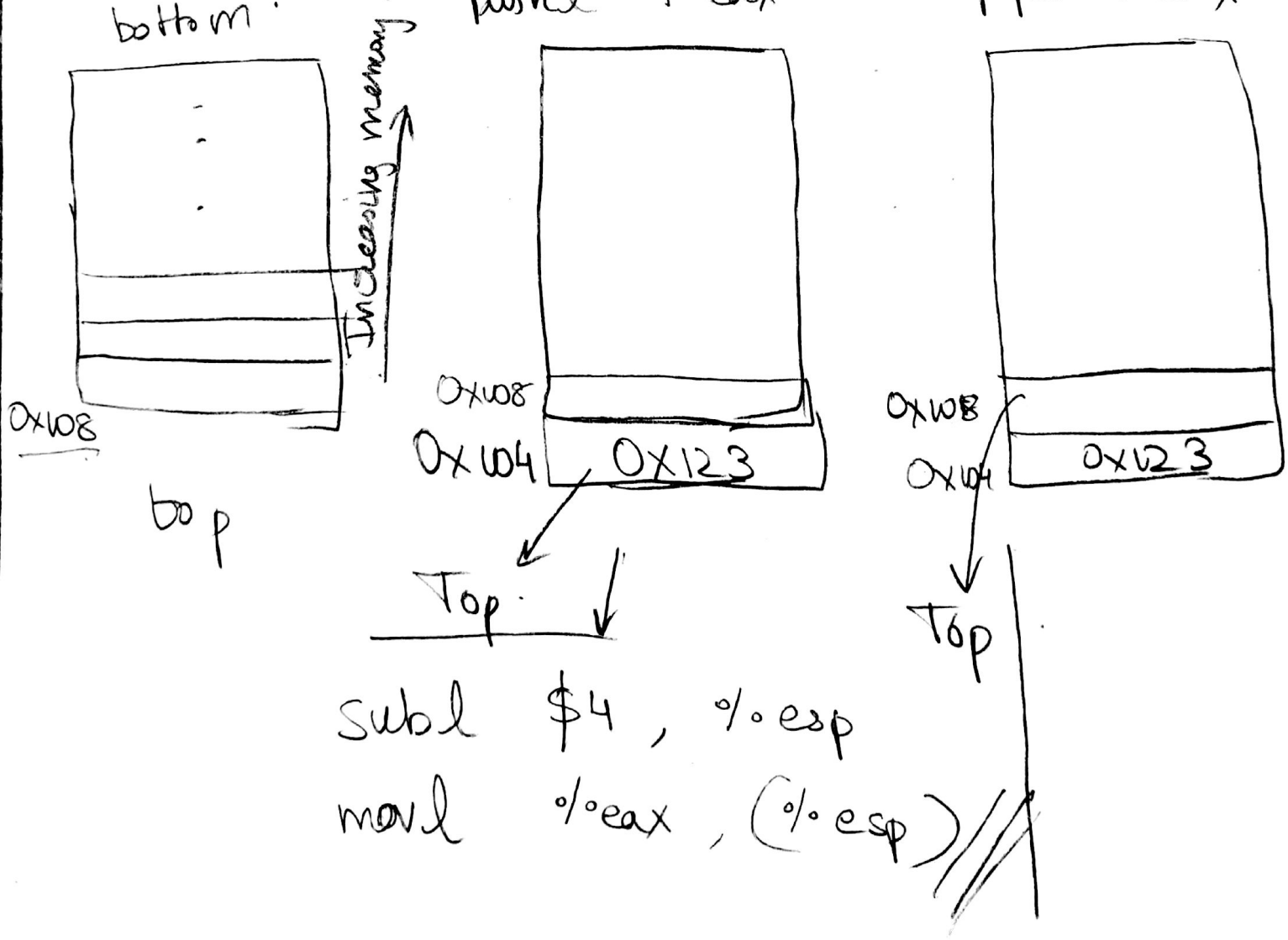
%.eax	0x123
%.edx	0
%.esp	0x108

pushl %.eax

%.eax	0x123
%.edx	0
%.esp	0x104

popl %.edx

%.eax	0x123
%.edx	0x123
%.esp	0x108



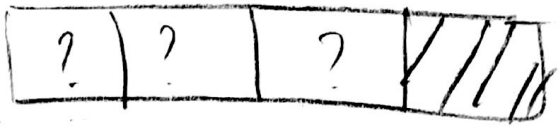
subl $4, %.esp

movl %.eax, (%.esp)

What if you want to move a byte to a 4 byte location?

→ 2 byte.

movsbw S, D.



movsbl D → 4 bytes MSB.

movswl D → 4 bytes → For signed numbers

Zero extension!

movzbw

movzwl

movzbl → For unsigned numbers

Shift Operators

Left Shift Arithmetic — SAL k, D

Left Shift Logical — SHL k, D

Right Shift Arithmetic — SAR k, D →

Right Shift Logical — SHR k, D →

k → a byte since it ranges only from (0 to 31)

SAL / SHL: Fills the left with the MSB bit

SAR: Fills the right with the MSB bit

SHR: Fill the left with zeros.