# Data movement

mov_ src, dest

↳ suffix

reg → reg  imm → reg

imm → memory  reg → memory

memory → reg  ~~reg/mem → imm~~

~~memory → memory~~

movb → byte

movw → ~~—~~ word (2 bytes)

movl → long/double (4 bytes)

movsbw  sign extend  byte to word

movsbl  byte to long

movswl  word to long

---

movzbw  zero extend

movzbl

movzwl

%eax : 0xFF

%ebx : ~~0x0~~ 0xFF

%ecx : ~~0x0~~ 0xFF

%edx : ~~0x0~~ 0xFFFFFFFF

movl %eax, %ebx

movzbl ~~%eax~~ %al, %ecx

movsbl %al, %edx

two more insts...

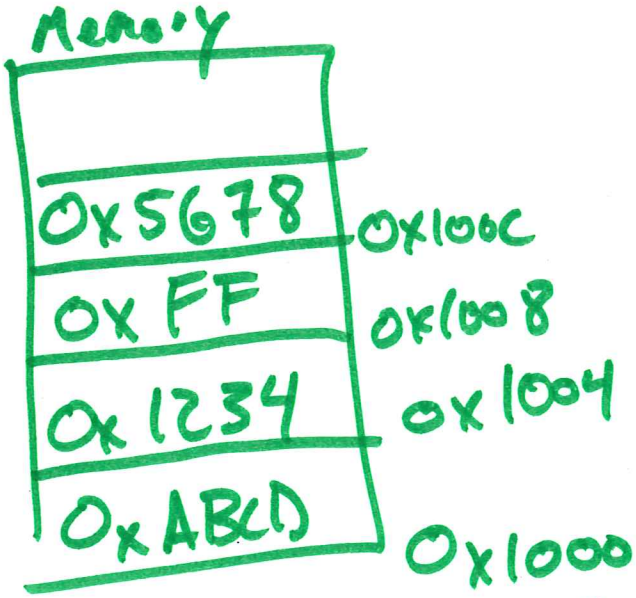
```
push src
pop  dst
```

push!

```
%esp ← %esp - 4
(%esp) ← src
```

pop

```
dest ← (%esp)
%esp ← %esp + 4
```




%eax: 0x1000
%ecx: 0x0
%esi: 0xFFAA
%edx: 0x4


%esi → 0xFFAA
%si → 0xFFAA
%ah → 0x10
%ax → 0x1000
%al → 0x00
(%eax) → 0xABCD
4(%eax) → 0x1234
4(%eax, %edx) → 0xFF

jpE:1 %eax
*i?
*(0x1000)
Addr: 0x1004

%eax + %edx · 1 + 4
0x1000 + 0x4 + 4
0x1008

from last time:

## operands

Immediate : $100

Register : %esi

Memory

~~Absolute/direct : 0x1000~~

~~Register indirect : %eax~~

- Absolute/direct: 0x1000
  - ↳ value @ 0x1000
- Register indirect: (%eax)
  - ↳ value @ addr. in %eax
- General form

C: 1, 2, 4, 8

N: a number

← ~~Form~~ N(R1, R2, C)

N + R1 + R2 · C

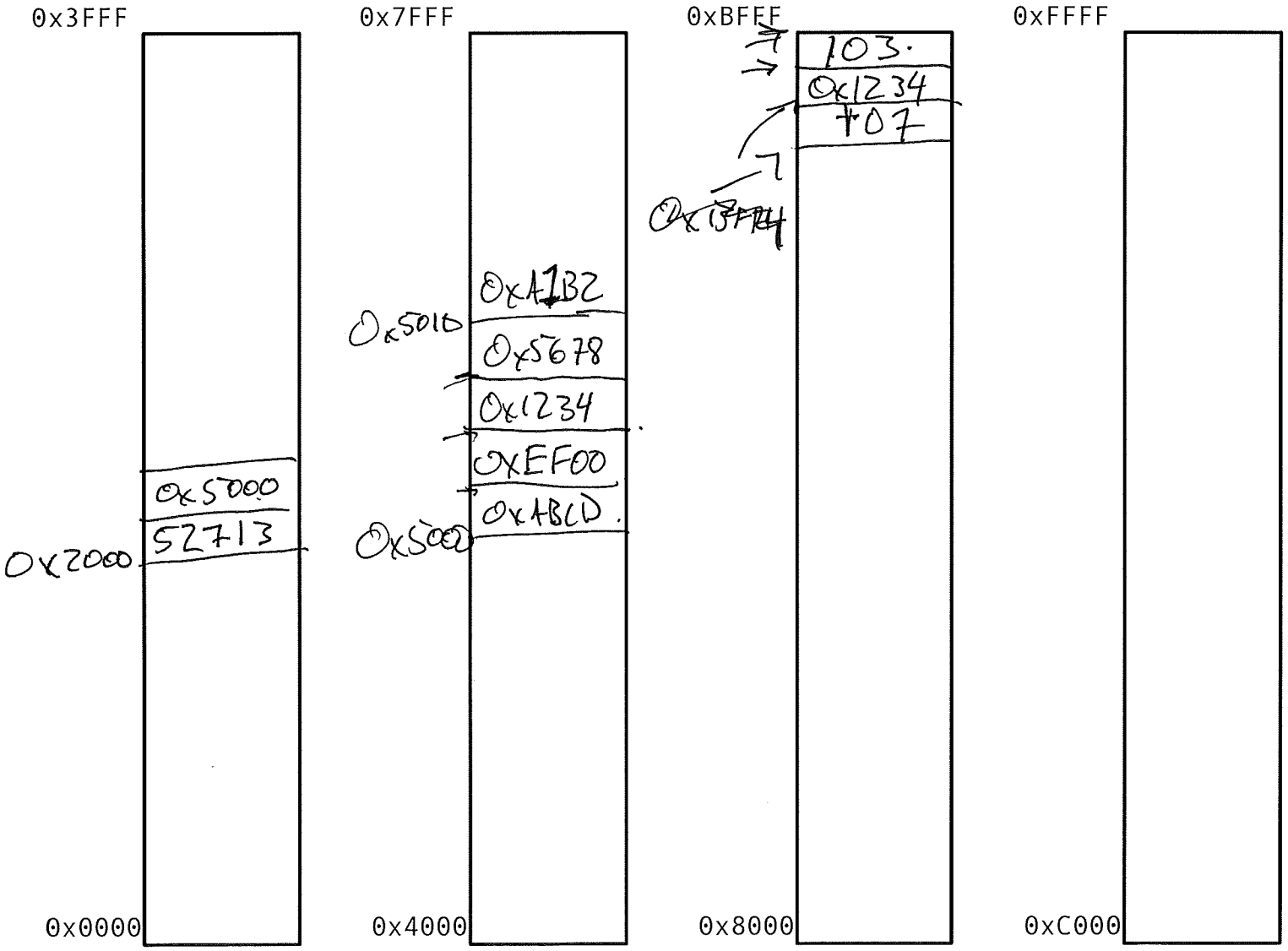
↳ value @ this location

Defaults:

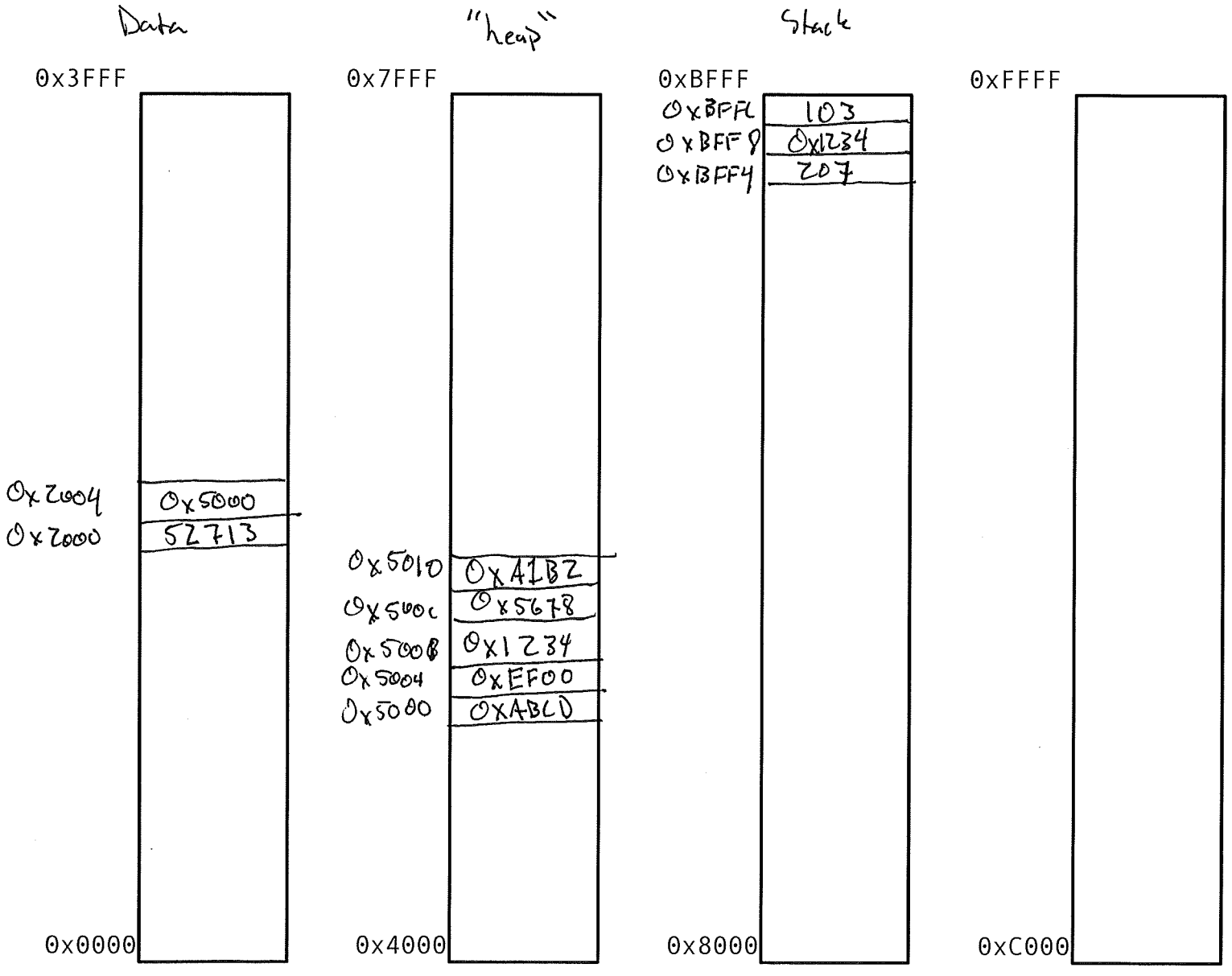
C=1 R2=0

N=0

| | |
|---|---|
| %eax | ~~0x0~~ ~~0x103~~ 0x5000 |
| %ecx | ~~0x0~~ ~~0xBFFC~~ 3 |
| %edx | 0x0 ~~0x1234~~ 0x5678 |
| %ebx | ~~0x0~~ 52713 |
| %esi | ~~0x0~~ 103 |
| %edi | ~~0x0~~ 107 |
| %esp | ~~0xBFFC~~ ~~0xBFF8~~ ~~0xBFF4~~ ~~0xBFF4~~ BFF8 |
| %ebp | 0xBFFC |



| | |
|---|---|
| %eax | ~~0x0~~ ~~103~~ 0x5000 |
| %ecx | ~~0x0~~ ~~0xBFFC~~ 3 |
| %edx | ~~0x0~~ ~~0x1234~~ 0x5678 |
| %ebx | ~~0x0~~ 52713 |
| %esi | ~~0x0~~ 103 |
| %edi | ~~0x0~~ 207 |
| %esp | ~~0x7FFF~~ ~~0xBFF8~~ ~~0xBFF4~~ 0xBFF8 |
| %ebp | 0x7FFF |



# Arithmetic instructions

leal → load effective address

- does address calculation
- looks like mov, but instead of performing the load just computes the address.
- compilers may cleverly use this inst.

leal src, dest → (%eax → 0x1000, %ebx → 0x10)

leal 8(%eax), %edx → %edx → 0x1008

leal 16(%eax, %ebx, 4), %edx → %edx → 0x1000 + 4·0x10 + 0x10 → 0x1050

like & in C (int *p = &array[100];)

---

incl dest → - increment dest register/memory
- dest = dest + 1
- ++ in C

decl dest → decrement
-- in C

shll → shift left
shrl → shift right
(sarl, sall) → arithmetic shifts

addl/subl src, dest → dest = dest +/- src

andl/orl/xorl src, dest → dest = dest &/|/^ src

imull dest, src

---

dest + src can be memory locations (but not both)