

Understand
address
computation

Use x86
Instructions to
do Transfer
Data

Use x86
Instructions to
do Arithmetic
operations



x86 Instructions

- ① Transfer Data
- ② Arithmetic Functions

x86-64 GPRS

%rax

%rbx

%rcx

%rdx

%rsi

%rdi

%rbp

%rsp

%r8

%r9

%r10

%r11

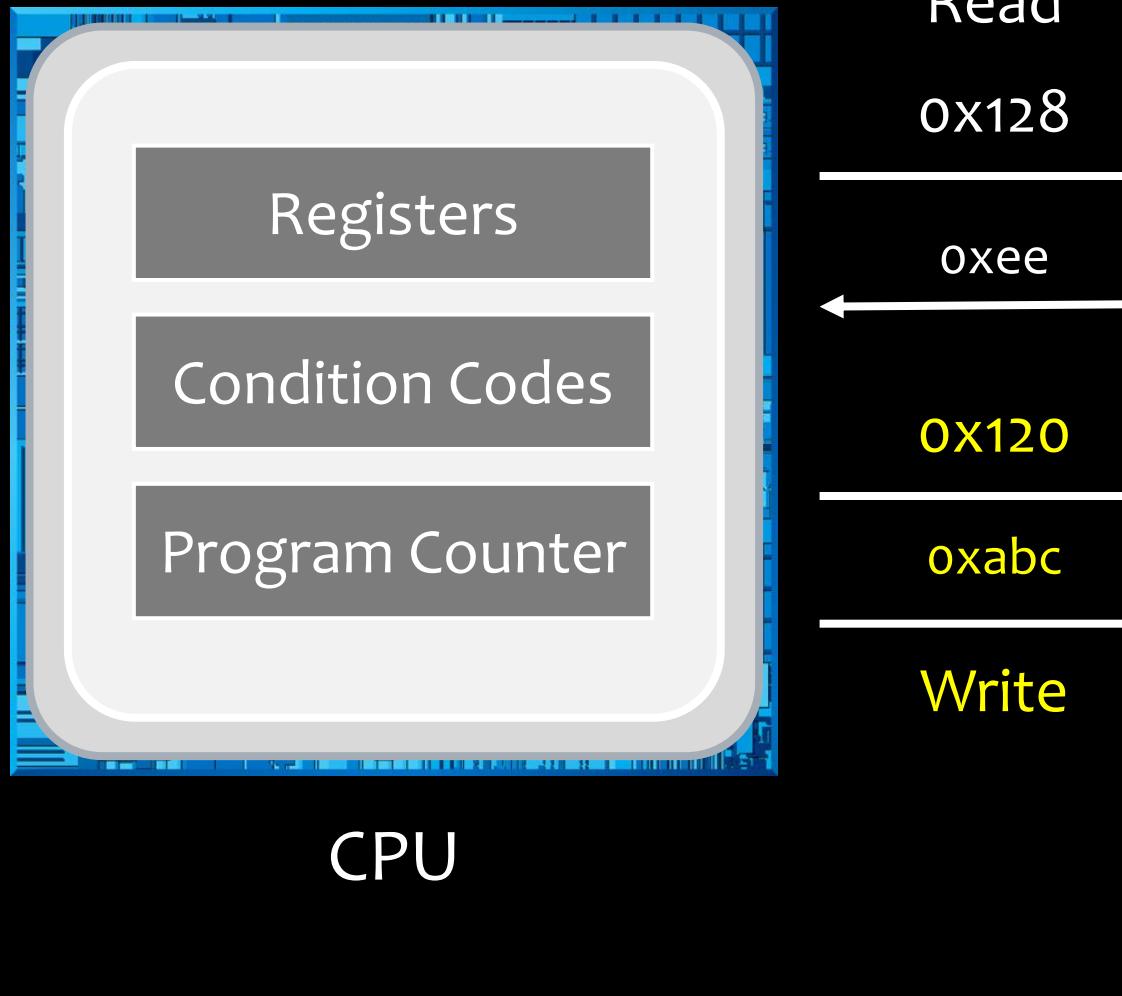
%r12

%r13

%r14

%r15

Programmer's View



Three Basic Kinds of Instructions

- Transfer data
 - MOV, LEA
- Arithmetic function
 - ADD, SUB, IMUL, SAL, SAR, SHR, XOR, AND, OR
 - INC, DEC, NEG, NOT
- Transfer control
 - JMP, JE, JNE, JS, JNS, JG, JGE, JL, JLE, JA, JB

Transfer Data

MOVX

movX

Source, Dest

1 byte **movb**
2 byte **movw**
4 byte **movl**
8 byte **movq**

Immediate

\$0x400
\$-533

Register

%eax
%rbx

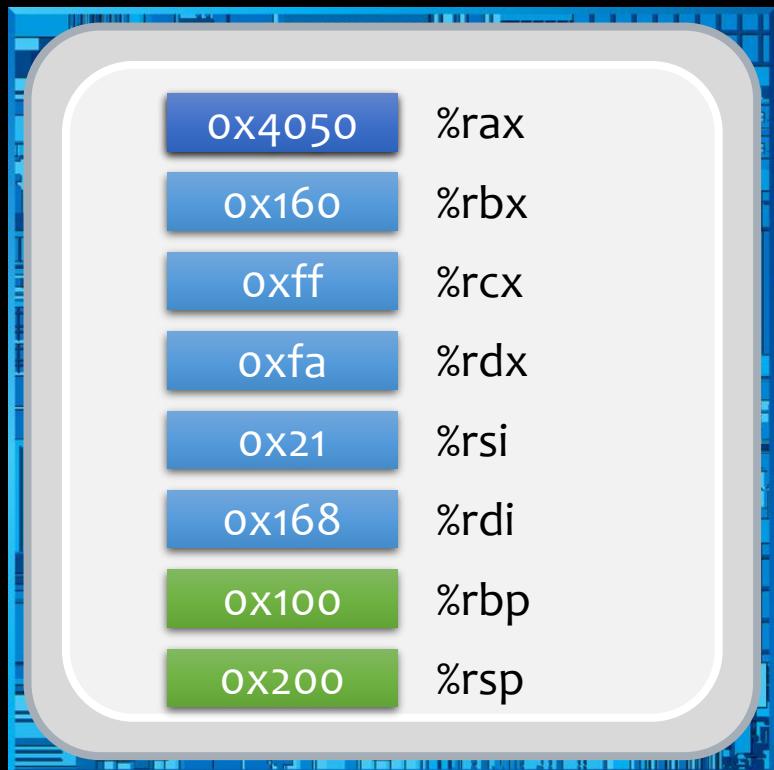
Memory

(%eax)
(%rbx)



Can't do memory-memory transfer with a single instruction.

MOV Example

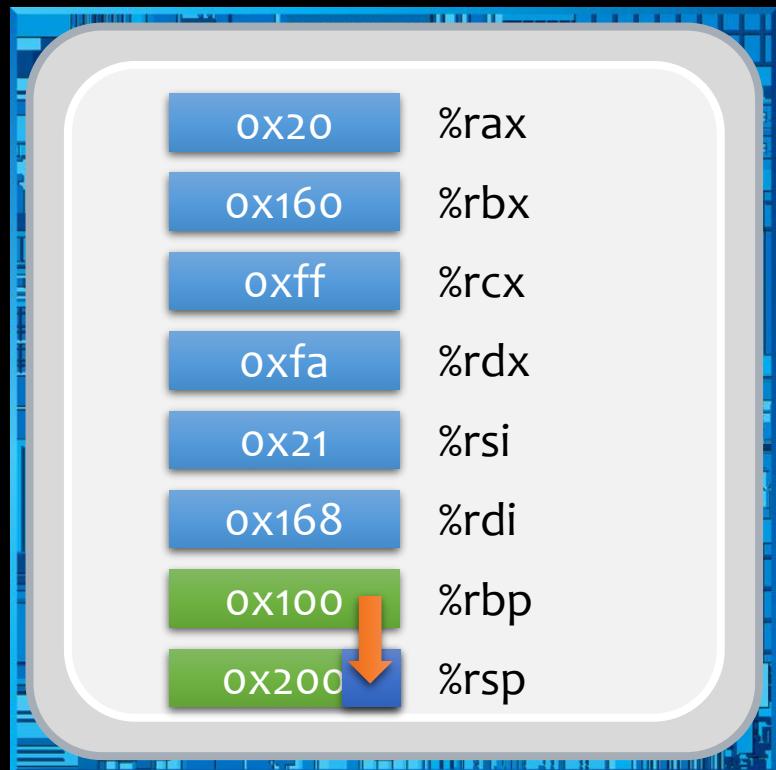


```
movl $0x4050, %eax
```

The diagram shows a stack dump with memory addresses on the left and their corresponding values on the right. The addresses decrease from top to bottom. Ellipses at the top and bottom indicate more entries.

Address	Value
0x198	...
0x190	0x11
0x188	0x22
0x180	0x33
0x178	0x44
0x170	0x55
0x168	0x66
0x160	0x77
0x158	0x88
0x150	0x99
0x148	0xaa
0x140	0xbb
0x138	0xcc
0x130	0xdd
0x128	0xee
0x120	0xff
0x118	0x12
0x110	0x34
0x108	0x56
0x100	0x78
...	...

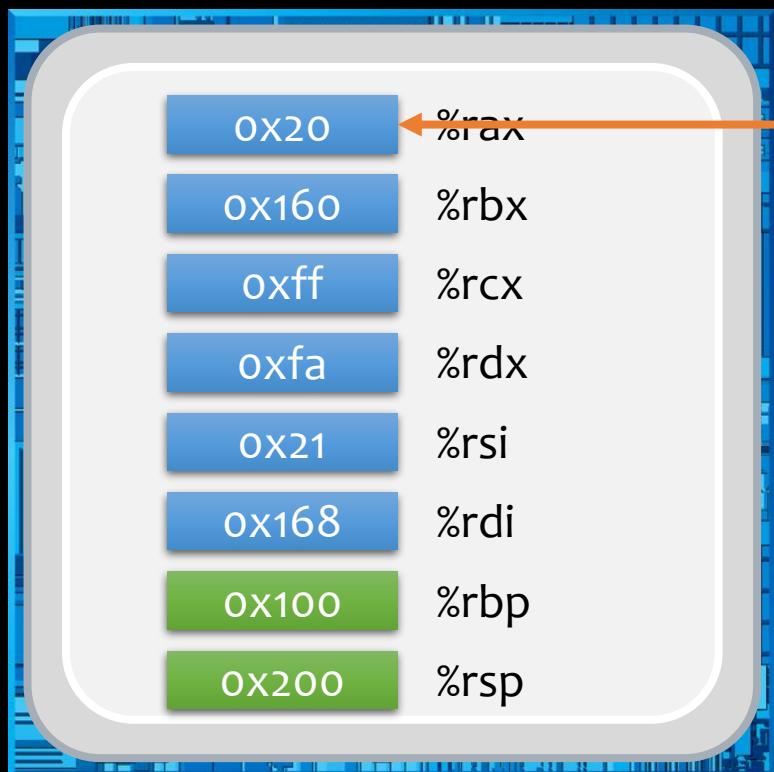
MOV Example



`movw %bp, %sp`

... 7

MOV Example

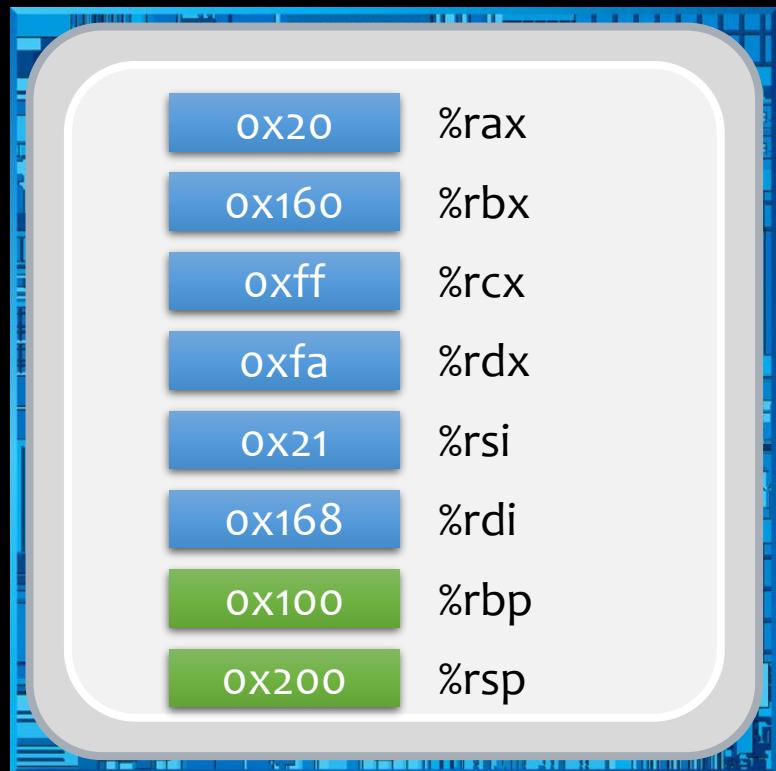


1 byte

...
0x198
0x190
0x188
0x180
0x178
0x170
0x168
0x160
0x158
0x150
0x148
0x140
0x138
0x130
0x128
0x120
0x118
0x110
0x108
0x100
...

movb (%rdi), %al

MOV Example

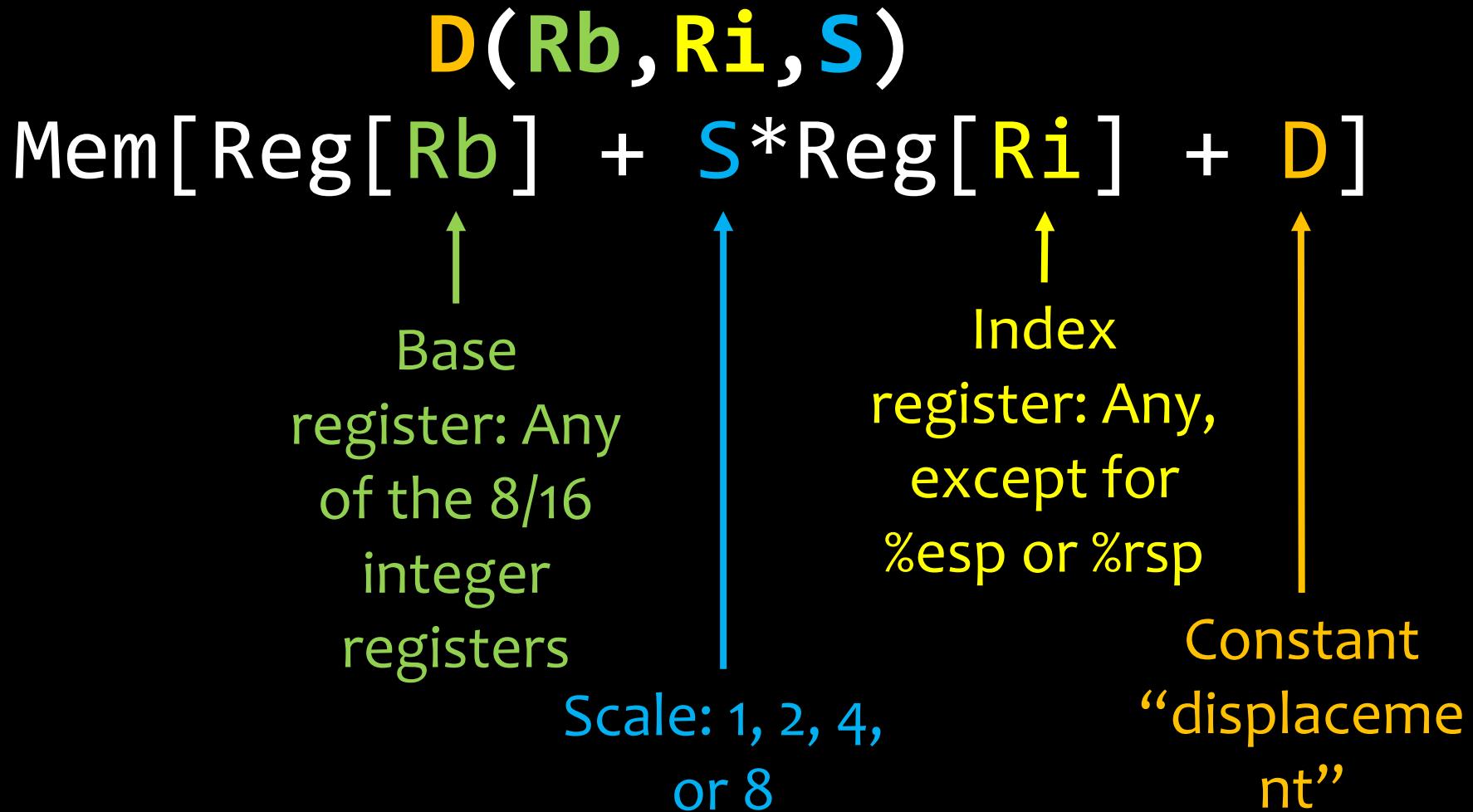


```
movb $-17, (%rdi)
```

The diagram shows a memory dump with addresses from 0x100 to 0x198. The value at address 0x168 is highlighted in blue, corresponding to the %rdi register value in the register dump.

...	0x198
0x190	0x190
0x188	0x188
0x180	0x180
0x178	0x178
0x170	0x170
0x168	0x168
0x160	0x160
0x158	0x158
0x150	0x150
0x148	0x148
0x140	0x140
0x138	0x138
0x130	0x130
0x128	0x128
0x120	0x120
0x118	0x118
0x110	0x110
0x108	0x108
0x100	0x100
...	0x100

Memory Addressing Modes



Memory Addressing Modes

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

Register	Value
%rax	0x100
%rcx	0x1
%rdx	0x3

Operand	Value
%rax	
0x104	
\$0x108	
(%rax)	
4(%rax)	
9(%rax,%rdx)	
260(%rcx,%rdx)	
0xFC(,%rcx,4)	
(%rax,%rdx,4)	

Practice Problem

Explain what is wrong?

```
movb    $0xF, (%ebx)
movl    %rax, (%rsp)
movw    (%rax), 4(%rsp)
movb    %al, %sl
movq    %rax, $0x123
movl    %eax, %rdx
movb    %si, 8(%rbp)
```

Swap

```
void swap_l  
  (long int *xp, long int *yp)  
{  
    long int t0 = *xp;  
    long int t1 = *yp;  
    *xp = t1;  
    *yp = t0;  
}
```

```
swap_l:  
    movq (%rdi), %rdx  
    movq (%rsi), %rax  
    movq %rax, (%rdi)  
    movq %rdx, (%rsi)  
    retq
```

Address Computation

LEAX load effective address
leax Source, Dest

leal (%edx,%ecx,4), %eax



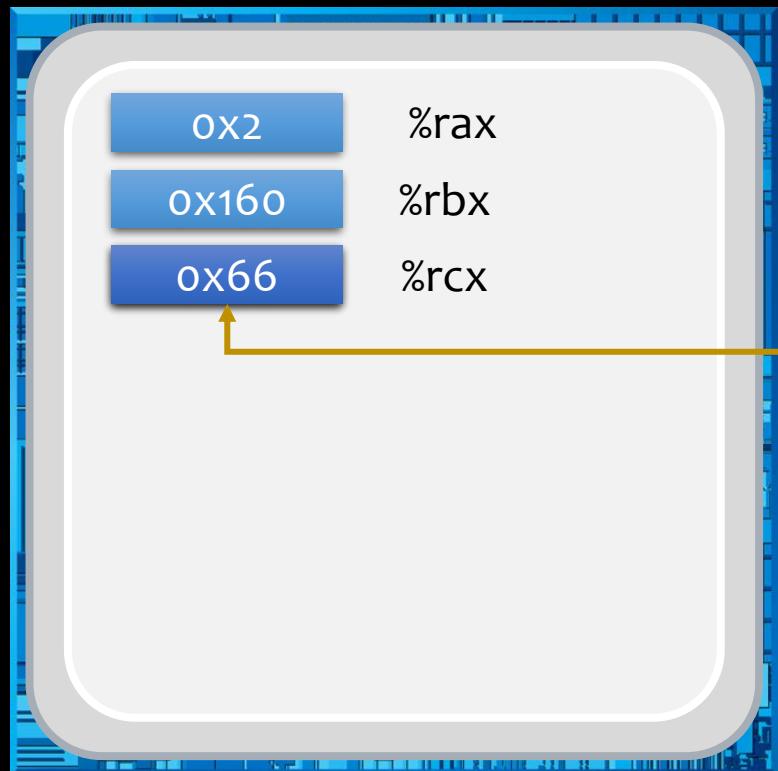
lea (%rbx,%rax,4), %rcx
Compute address of value

mov (%rbx,%rax,4), %rcx
Load value at that address

Suppose register %eax holds value x and %ecx holds value y. Fill in the table below:

Instruction	Result
leal 6(%eax), %edx	?
leal (%eax,%ecx), %edx	?
leal (%eax,%ecx,4), %edx	?
leal 7(%eax,%eax,8), %edx	?
leal 0xA(%ecx,4), %edx	?
leal 9(%eax,%ecx,2), %edx	?

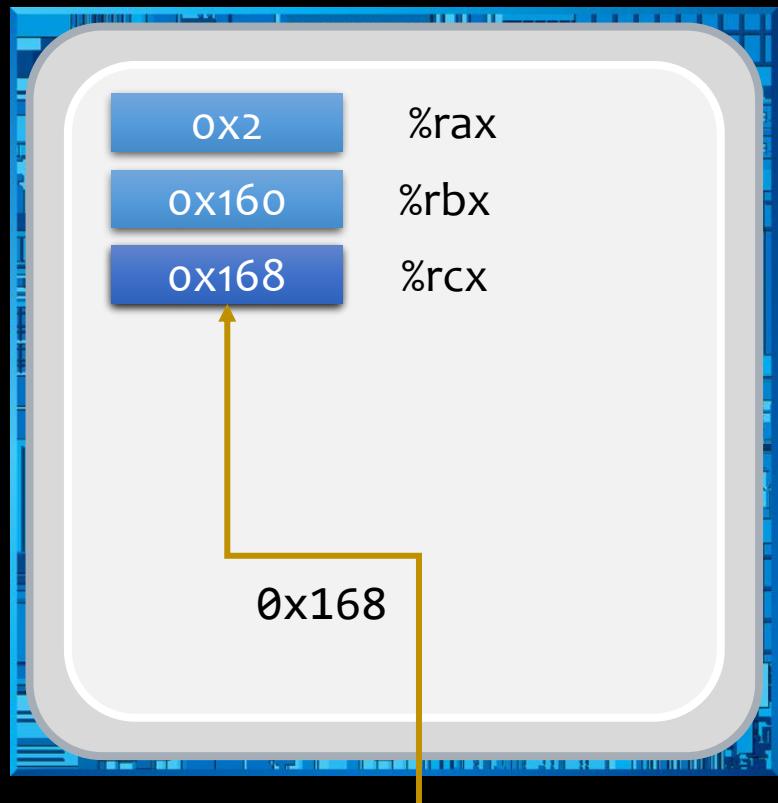
MOV vs. LEA



`mov (%rbx,%rax,4), %rcx`

...	...
0x198	0x198
0x190	0x190
0x188	0x188
0x180	0x180
0x178	0x178
0x170	0x170
0x168	0x168
0x160	0x160
0x158	0x158
0x150	0x150
0x148	0x148
0x140	0x140
0x138	0x138
0x130	0x130
0x128	0x128
0x120	0x120
0x118	0x118
0x110	0x110
0x108	0x108
0x100	0x100
...	...

MOV vs. LEA



lea (%rbx,%rax,4), %rcx

...	...
0x198	0x190
0x188	0x180
0x178	0x170
0x168	0x160
0x158	0x150
0x150	0x148
0x148	0x140
0x140	0x138
0x138	0x130
0x130	0x128
0x128	0x120
0x120	0x118
0x118	0x110
0x110	0x108
0x108	0x100
...	...

Arithmetic Operations

Format	Computation
add Src, Dest	$Dest = Dest + Src$
sub Src, Dest	$Dest = Dest - Src$
imul Src, Dest	$Dest = Dest * Src$
sal Src, Dest	$Dest = Dest \ll Src$
sar Src, Dest	$Dest = Dest \gg Src$
shr Src, Dest	$Dest = Dest \gg Src$
xor Src, Dest	$Dest = Dest \wedge Src$
and Src, Dest	$Dest = Dest \& Src$
or Src, Dest	$Dest = Dest Src$

Arithmetic Operations

Format	Computation
inc Dest	$\text{Dest} = \text{Dest} + 1$
dec Dest	$\text{Dest} = \text{Dest} - 1$
neg Dest	$\text{Dest} = -\text{Dest}$
not Dest	$\text{Dest} = \sim \text{Dest}$

Assume the following values are stored at the indicated memory addresses and registers, fill in the table below:

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

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

Instruction	Destination	Value
addl %ecx,(%eax)	?	?
subl %edx,4(%eax)	?	?
imull \$16,(%eax,%edx,4)	?	?
incl 8(%eax)	?	?
decl %ecx	?	?
subl %edx,%eax	?	?

Using LEA for arithmetic exps

```
int arith  
    (int x, int y, int z)  
{  
    int t1 = x+y;  
    int t2 = z+t1;  
    int t3 = x+4;  
    int t4 = y * 48;  
    int t5 = t3 + t4;  
    int r = t2 * t5;  
    return r;  
}
```

x in %rdi
y in %rsi
z in %rdx

```
arith:  
leal (%rsi,%rsi,2),%ecx  
sall $4,%ecx  
leal 4(%rdi,%rcx),%eax  
addl %edi, %esi  
addl %esi, %edx  
imull %edx,%eax  
ret
```

Summary

- x86 data transfer instructions
- x86 arithmetic instructions