Understand address computation

Use x86 Instructions to do Transfer Data

Use x86 Instructions to do Arithmetic operations

x86 Instructions

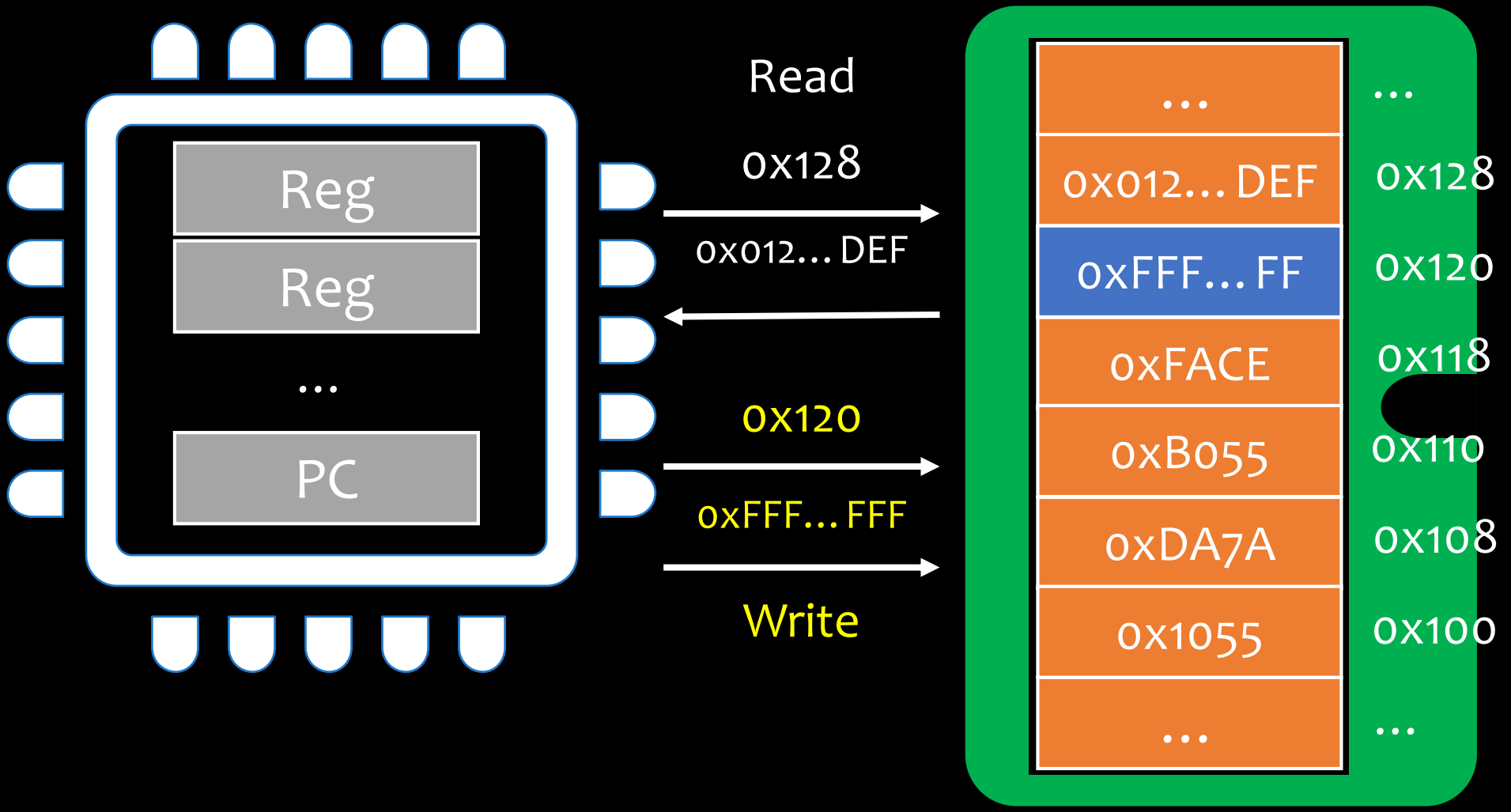
① Transfer Data

② Arithmetic Functions

x86-64 GPRS

%rax	%r8
%rbx	%r9
%rcx	%r10
%rdx	%r11
%rsi	%r12
%rdi	%r13
%rbp	%r14
%rsp	%r15

Assembly Programmer's View



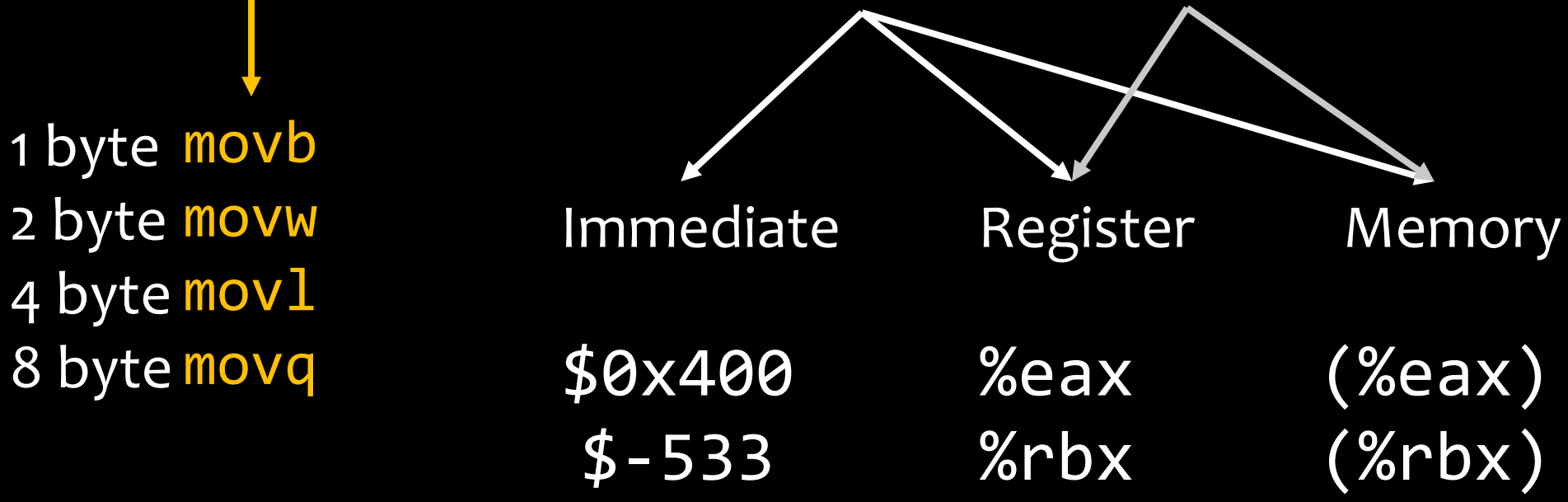
Three Basic Kinds of Instructions

- 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

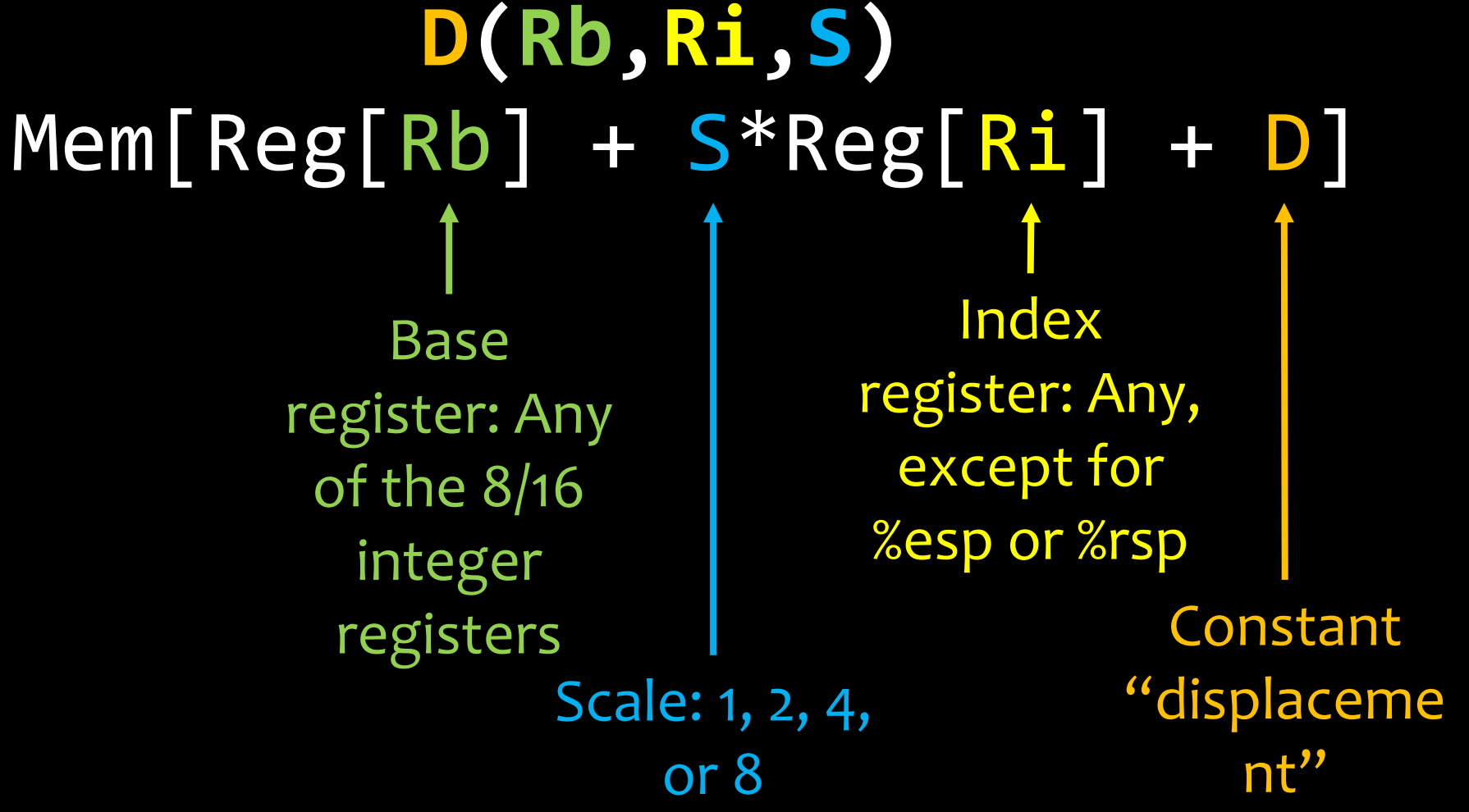
MOVX

`movx` `Source, Dest`



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

Memory Addressing Modes



Memory Addressing Modes

%edx	0xf000
%ecx	0x100

(Rb,Ri)	Mem[Reg[Rb]+Reg[Ri]]
D(,Ri,S)	Mem[S*Reg[Ri]+D]
(Rb,Ri,S)	Mem[Reg[Rb]+S*Reg[Ri]]
D(Rb)	Mem[Reg[Rb] +D]

Expression	**Address Computation**	**Address**
0x8(%edx)	0xf000 + 0x8	0xf008
(%edx,%ecx)	0xf000 + 0x100	0xf100
(%edx,%ecx,4)	0xf000 + 4*0x100	0xf400
0x80(,%edx,2)	2*0xf000 + 0x80	0x1e080

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
```

x86-64

Address Computation

LEAX load effective address

```
leax     Source, Dest
```

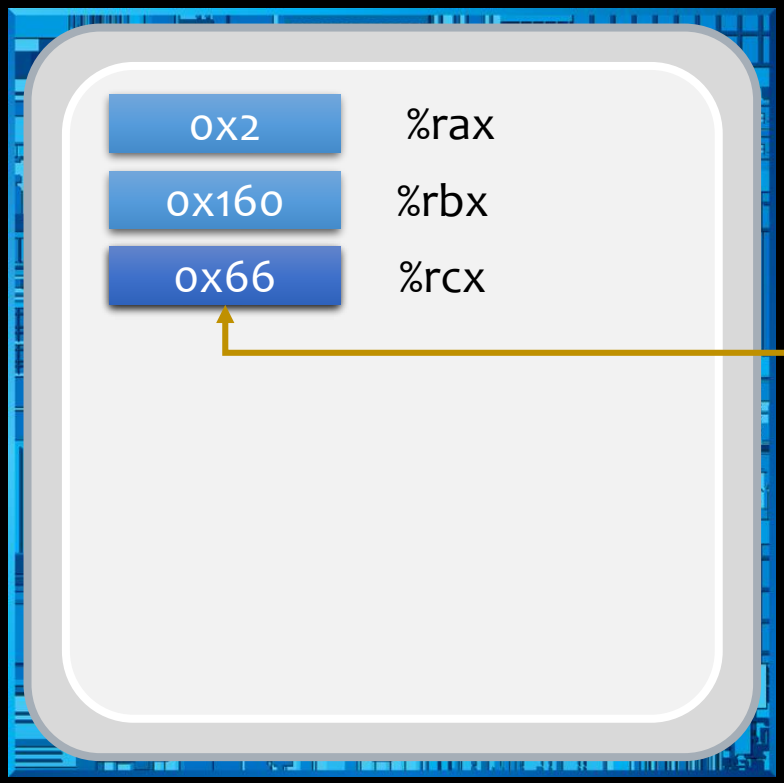
```
leal (%edx,%ecx,4), %eax
```

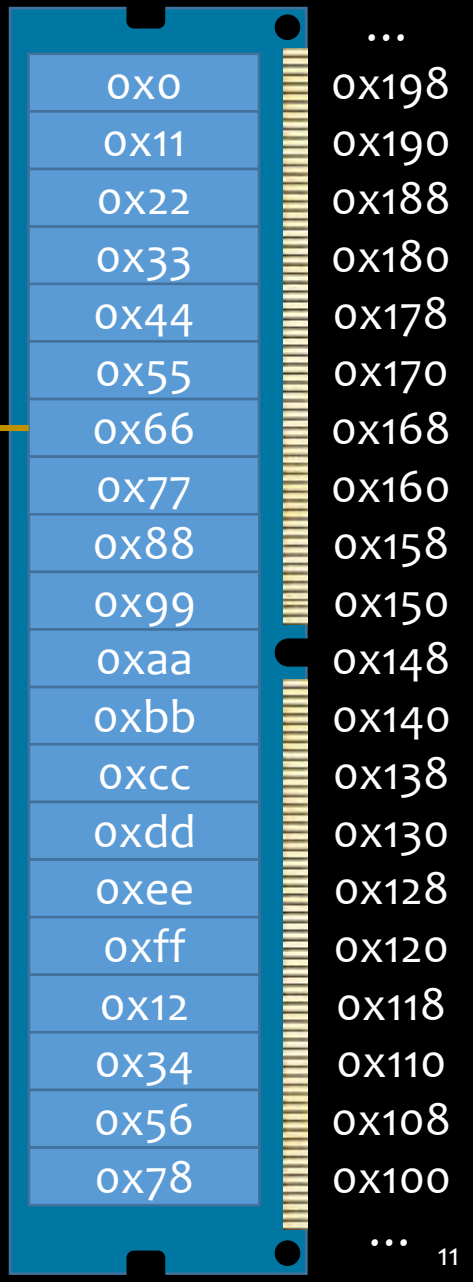
```
lea (%rbx,%rax,4), %rcx
   Compute address of value

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`	6+x
`leal (%eax,%ecx), %edx`	x+y
`leal (%eax,%ecx,4), %edx`	x+4y
`leal 7(%eax,%eax,8), %edx`	7+9x
`leal 0xA(,%ecx,4), %edx`	10+4y
`leal 9(%eax,%ecx,2), %edx`	9+x+2y

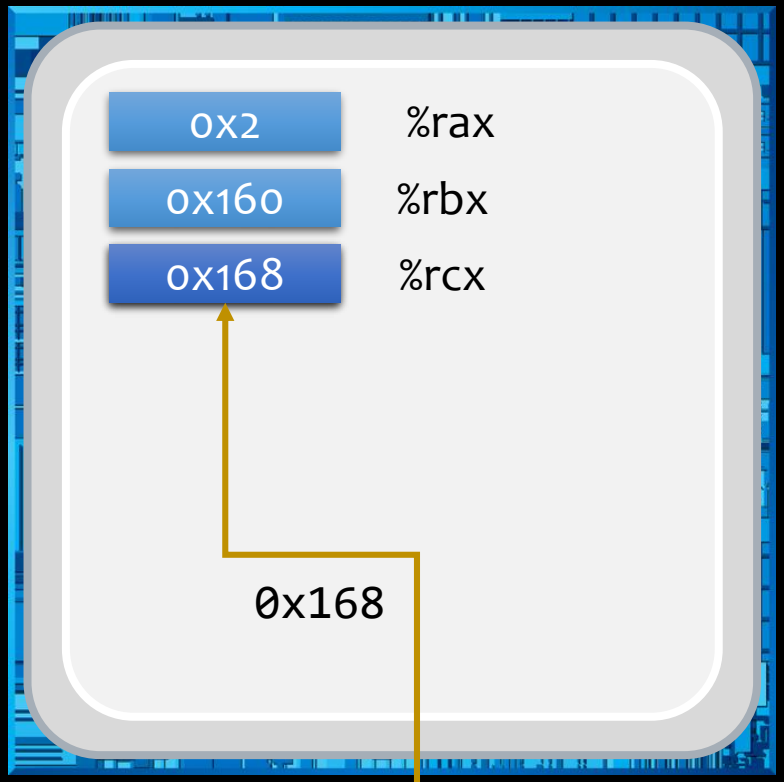
MOV vs. LEA



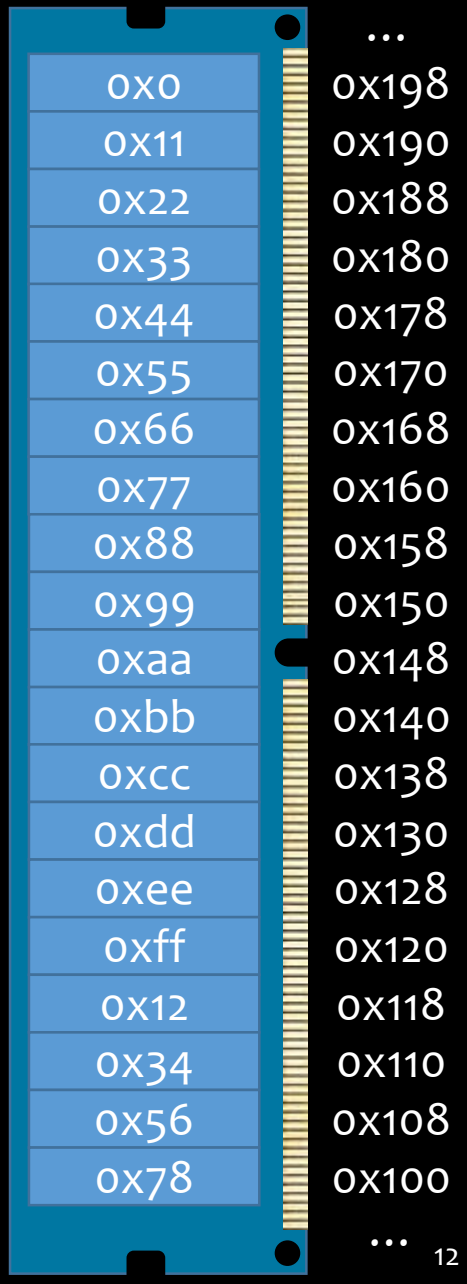



```
mov (%rbx,%rax,4), %rcx
```

MOV vs. LEA




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



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 << Src`
`sar Src, Dest`	`Dest = Dest >> Src`
`shr Src, Dest`	`Dest = Dest >> Src`
`xor Src, Dest`	`Dest = Dest ^ Src`
`and Src, Dest`	`Dest = Dest & Src`
`or Src, Dest`	`Dest = Dest | Src`

Arithmetic Operations

Format	Computation
`inc Dest`	`Dest = Dest + 1`
`dec Dest`	`Dest = Dest - 1`
`neg Dest`	`Dest = -Dest`
`not Dest`	`Dest = ~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)`	0x100	0x100
`subl %edx,4(%eax)`	0x104	0xA8
`imull $16,(%eax,%edx,4)`	0x10C	0x110
`incl 8(%eax)`	0x108	0x14
`decl %ecx`	%ecx	0x0
`subl %edx,%eax`	%eax	0xFD

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

Kenneth Harry Olsen

Founder of Digital Equipment Corp

> “There is no reason for any individual to have a computer in his home.”