07/26

*know diff. btwn lea vs mov

Intel Load Effective Address: ≠ mov instruction

Putting the address into the register.

- Leal Src, DST
- LeaQ Src, DST

*Src has to be addr.
*DST has to be register!

() means go to memory

- Leal (%eax), %ebx = movl %eax, %ebx
  ↳ %ebx gets contents of %eax register

But this Does not go to memory

- movl (%eax), %ebx
  ↳ %ebx get 4 byte Long from memory at addr in %eax

- Leal 4o(%eax), %ebx
  ↳ %ebx gets 4o + contents of %eax
- Leal (%eax, %esx), %ebx
  ↳ %ebx gets sum of %eax, %esx contents

*Registers = numbers
↓
Memory = addr.
*think of it as being similar to pointers

```
int sum (int A, int B)
    return A+B;
```

```
Leal (%edi, %esi), %eax
Ret.
```

```
int FCN (int i){
    int *p;
    p = &i;
    *p = 3;
    return 0;
}
```

get addr. of local var i

```
LeaQ -4(%rbp), %rbx
movl $3, (%rbx)
*p = 3
```

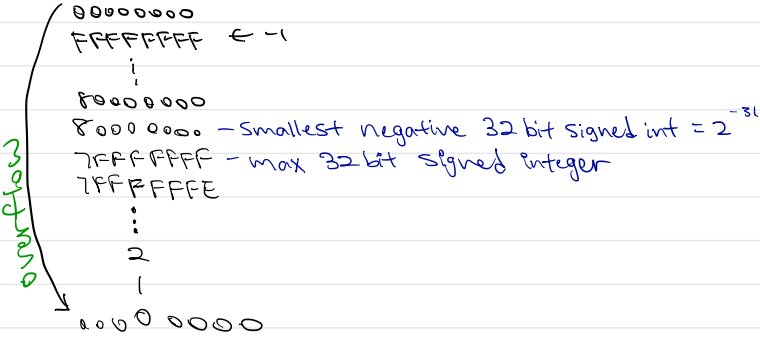
pointer dereference

Lea not access memory

Overflow:

- occurs when you try to represent a number that does not fit

*LSB gets stored and MSB gets lost



- Register size of processor determines range of values that can be represented.
  - Unsigned examples:
    - 8 bit: $0 \ldots 2^8 - 1$
    - 16 bit: $0 \ldots 2^{16} - 1$
    - 32 bit: $0 \ldots 2^{32} - 1$
    - 64 bit: $0 \ldots 2^{64} - 1$

- An arithmetic result produces a result larger than N bits.
- Overflow reduces result to modulo N, retaining only the least significant N bits, causing a wrap around.

addl src, dst DST += SRC

(src > 0 && dst > 0 && sum < 0)
|| (src < 0 && dst < 0 && sum > 0)

overflow!
$$\begin{array}{r} +\text{int} \\ +\ +\text{int} \\ \hline -\text{int} \end{array}$$

$$\begin{array}{r} -\text{int} \\ +\ -\text{int} \\ \hline +\text{int} \end{array}$$
overflow!

ex.

$$\begin{array}{r} \text{7FFFFFFF} \\ +\qquad\quad 1 \\ \hline \text{80000000} \end{array}$$

OVERFLOW

subl src, dst DST -= SRC

(src < 0 && dst > 0 && difference < 0)
|| (src > 0 && dst < 0 && difference > 0)

+ int		− int
− − int		− + int
− int	overflow!	+ int

− int		+ int
− + int		− − int
+ int	overflow!	− int

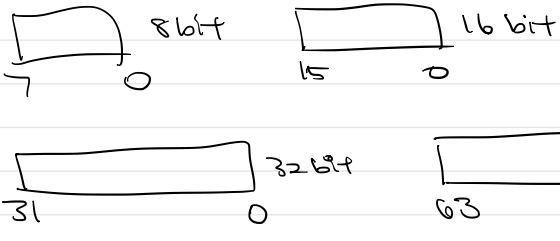
imull src, dst DST *= SRC src − DST

(src > 0 && dst > 0 && product < 0)
|| (src < 0 && dst < 0 && product < 0)
|| (src < 0 && dst > 0 && product > 0)
|| (src > 0 && dst < 0 && p roduct > 0)

+ int		− int		− int		+ int
x + int		* − int		* + int		* − int
− int	of!	− int	of!	+ int	of!	+ int

Bit Ordering:

A convention independent of byte ordering (Endianness) ✱



Bitwise operations: (And, or, Xor)

Bitwise And (C operator: &)

```
int a,b,c;
c = A & B;
```

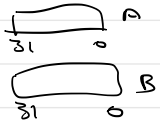
two opnds


```
 1100
&1010
------
 1000
```

Bitwise or (C operator: |)

```
int a,b,c;
c = A | B;
```

two opnds




```
 1100
|1010
------
 1110
```

Bitwise Xor (C operator: ^)

```
int a,b,c;
c = A ^ B;
```

two opnds


```
 1100
^1010
------
 0110
```

Bitwise Not (C operator: ~)

- flip the bits

```
int a,b;
a = ~b;
```


```
~01
---
 10
```

Logical operations: (And, or)

Logical And (operator: &&)

- result is 0 or 1

```
if (f() && g())
```

if left is 0, then right is not reached

Logical Or (operator: ||)

- result is 0 or 1

```
if (f() || g())
```

if left is 1, then right is not reached