354 (Lecture 9)

[C]

[x86] \rightarrow \begin{align*}
\text{review} & \quad \text{security, 64-bit x86 (vectors)} \\
\Rightarrow \left[ \text{"Systems"} \right] & \quad \text{start: next week}
\end{align*}
for (i = 0; i < size; i++)
{
    x[i] = i;
}

assume:
address of x (begin of array)
in %eax

know: size is 10

write 10 movl instructions that behave like

x[0] = 0;

movl $0, 0(%eax)

x[1] = 1;

movl $1, 4(%eax)

movl $2, 8(%eax)

x[9] = 9;

movl $9, 36(%eax)
for (i = 0; i < @size; i++)
X[i] = i;

Now: write loop in x86
assume: iX -> %eax
size -> %ecx

write loop
struct foo  
  int x;
  int y;
  int z;
3 4 bytes

struct foo2  
  int x;
  char y;
  int z;
3

movl $10, 0(%eax)
movb $97, 9(%eax)
movl $30, 8(%eax)
Floating Point

old way: \( \text{\(x87\)} \) was coprocessor

32-bit (and older)
FP on intel

new registers
\[ s0 \ldots s7 \]
up to 80 bits
32, 64, 80
single, double, fp

new instructions
\[
\begin{align*}
\text{fld} & \quad \text{fst} \\
\text{move fp} \quad \text{reg} \Rightarrow \text{mem} \\
\text{value from} \\
\text{mem} \Rightarrow \text{registers}
\end{align*}
\]

\[
\begin{align*}
\text{fadd, fmul, fdiv, fsub} \\
\text{fsqrt, fsin, fcose, …}
\end{align*}
\]
Security:

Stack smashing

```
mygets (char * s) {
    put stuff
    } => s
    } may overflow
    buffer
```

```
echo () {
    char buf[4];
    mygets (buf);
    }
```

```
main () {
    echo();
    }
```

Potential:
mix data/control on Stack
buffers => overflow (overwrite)
defenses

stack guard (canary)

some random value
x86 64-bit

motivation:

address space

32: refer to

$2^{32}$ bytes

64-bit: quite a bit larger

larger data items:

64-bit registers

32-bit addresses

$2^64 - 1$

most (unused)
x86: 64-bit (contrast 32-bit)

registers:

32-bit:
- eax, ebx, ...
- esp, ebp, esp

64-bit:
- rax, rbx, ...
- rip, rsp, rbp

64-bit

---

more registers:
- r8 ... r15

calling convention:
more "register" based

=> for first six args (size 64-bit or less)

use registers to pass values to functions

rdi, rsi, rdx, rcx, r8, r9

=> use stack for other (large) things or > 6 args
Extensions to instruction set:

**vector extensions**

=> instead of operating one data item @ time

=> vectors: operate on >1 (2, 4, 8) data items @ time

**new registers:**

xmm 0 ... 15 => ymm 0 ... 15

128 bits 256 bits

**new instructions:** use registers to do work in parallel

xmm0  
\[ \begin{array}{cccc}
  a & b & c & d \\
\end{array} \]

xmm1  
\[ \begin{array}{cccc}
  w & x & y & z \\
\end{array} \]

\[ \text{addps} \quad \text{packed} \Rightarrow \text{vector} \]