Floating-Point on x86-64

Sixteen registers: %xmm0 through %xmm15

• float or double arguments in %xmm0 – %xmm7
• float or double result in %xmm0
• %xmm8 – %xmm15 are temporaries (caller-saved)

Two operand sizes:
• single-precision = 32 bits = float
• double-precision = 64 bits = double
Arithmetic Instructions

```
addsx source, dest
subsx source, dest
mulsx source, dest
divsx source, dest
```

x is either s or d

Add doubles

```
addsd %xmm0, %xmm1
```

Multiply floats

```
mulss %xmm0, %xmm1
```
Conversion

cvtSx2Sx \ source, \ dest

cvtTTSx2Sx \ source, \ dest

x is either s, d, or i

With i, add an extra extension for l or q

Convert a long to a double

\texttt{cvtSI2SdQ} %rdi, %xmm0

Convert a float to a int

\texttt{cvTTSS2SIL} %xmm0, %eax
Example Floating-Point Compilation

double scale(double a, int b) {
    return b * a;
}

cvtsi2sd %edi, %xmm1
mulsd %xmm1, %xmm0
ret
SIMD Instructions

```
addpx source, dest
subpx source, dest
mulpx source, dest
divpx source, dest
```

Combine *pairs* of *doubles* or *floats*
... because registers are actually 128 bits wide

Add two pairs of *doubles*

```
addpd %xmm0, %xmm1
```

Multiply four pairs of *floats*

```
mulps %xmm0, %xmm1
```
Auto-Vectorization

```c
void mult_all(double a[4], double b[4]) {
    a[0] = a[0] * b[0];
    a[1] = a[1] * b[1];
}
```

- What if \(a\) and \(b\) are aliases?
- What if \(a\) or \(b\) is not 16-byte aligned?
void mult_all(double *__restrict__ ai,
        double *__restrict__ bi) {
    double *a = __builtin_assume_aligned(ai, 16);
    double *b = __builtin_assume_aligned(bi, 16);

    a[0] = a[0] * b[0];
    a[1] = a[1] * b[1];
}

gcc -O3
movapd 16(%rdi), %xmm0
movapd (%rdi), %xmm1
mulpd 16(%rsi), %xmm0
mulpd (%rsi), %xmm1
movapd %xmm0, 16(%rdi)
movapd %xmm1, (%rdi)
ret

Auto-Vectorization
History: Floating-Point Support in x86

8086

• No floating-point hardware

• Software can implement IEEE arithmetic by manipulating bits, but that’s slow

8087 (a.k.a. x87)

• Co-processor for 8086

• CPU handles instructions by deferring to the FPU
History: Floating-Point Support in x86

80386

• CPU + FPU together on one chip
• Some 80386 chips had FP support, some didn’t

80486

• FP support always available
x87 Features

80-bit floating-point numbers

• 63 bits for fraction part
• 15 bits for exponent
• 1 extra bit (not quite IEEE encoding)

Registers are arranged in a stack

• Leads to **FADD** vs. **FADDP**
On to SSE

To support parallel operations, Intel and AMD introduced alternative FP instructions as the MMX and 3DNow! extensions

By the Pentium 4 (ca. 2000): SSE2

• At this point, x87 is always available, and SSE2 is practically always available

• Still, to support old hardware and old libraries, x87 remains in widespread use for 32-bit x86
Floating-Point Support in x86-64

On x86-64, SSE2 is always available

So, x86-64 applications and ABIs use SSE2

    ADDSD, MOVD, CVTSD2SI, etc.

x87 is still around!

Scientific applications that benefit from 80 bits of FP precision sometimes still use it

    FADD, FLD, FILD, etc.
More Variants

- SSE (1999)
- SSE2 (2001) — new instructions
- SSE3 (2004) — new instructions
- SSSE3 & SSE4 — new instructions
- AVX (2011) — 256-bit registers
- AVX2 (2015) — new instructions