Why RISC-V

Sparsh Mittal

Example of complexity of x86

- In x86, instructions can be 1B to 15B.
- Example of 1B instruction: **inc eax**, which assembles into (in hexadecimal) 40.
- Example of 15B instruction: **lock add dword ptr ds:[esi+ecx*4 +0x12345678], 0xefcdab89**.
- It assembles into (in hexadecimal): 67 66 f0 3e 81 84 8e 78 56 34 12 89 ab cd ef

- In RV32, all instructions are 32b long.

RISC-V instructions are typically at most one clock cycle per instruction (ignoring cache misses).

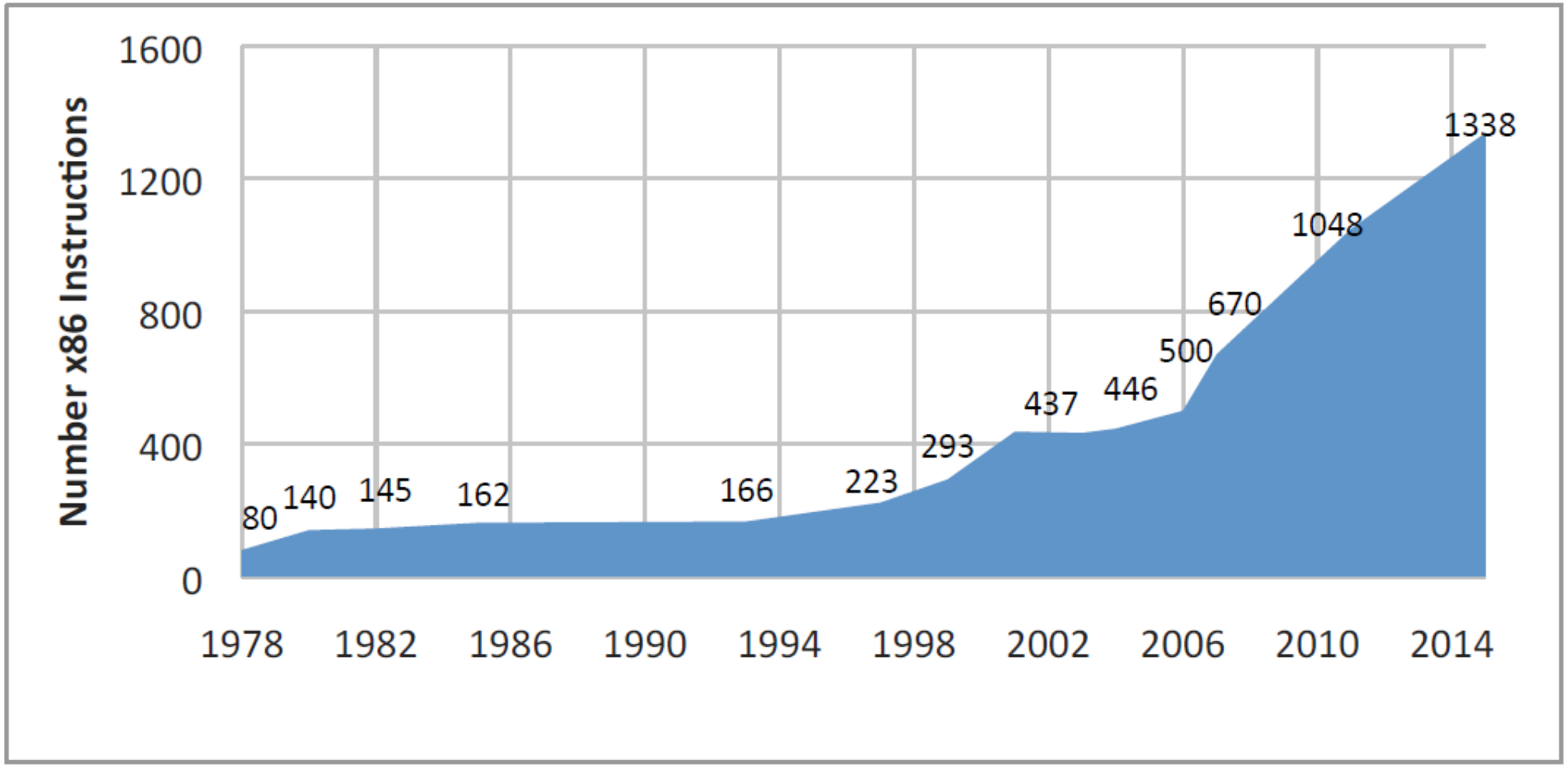
Both ARM-32 and x86-32 have instructions that take many clock cycles even when everything fits in the cache.

Unlike ARM-32 and RISC-V, x86-32 arithmetic instructions can have operands in memory instead of requiring all operands to be in registers.

Complex instructions and operands in memory make it difficult for processor designers to deliver performance predictability.

Growth of x86 instruction set over its lifetime

X86 is still growing!



A large part of the growth is because the x86 ISA relies on SIMD instructions for data level parallelism

Cost: ARM vs RISC-V

$$cost \approx f(die\ area^2)$$

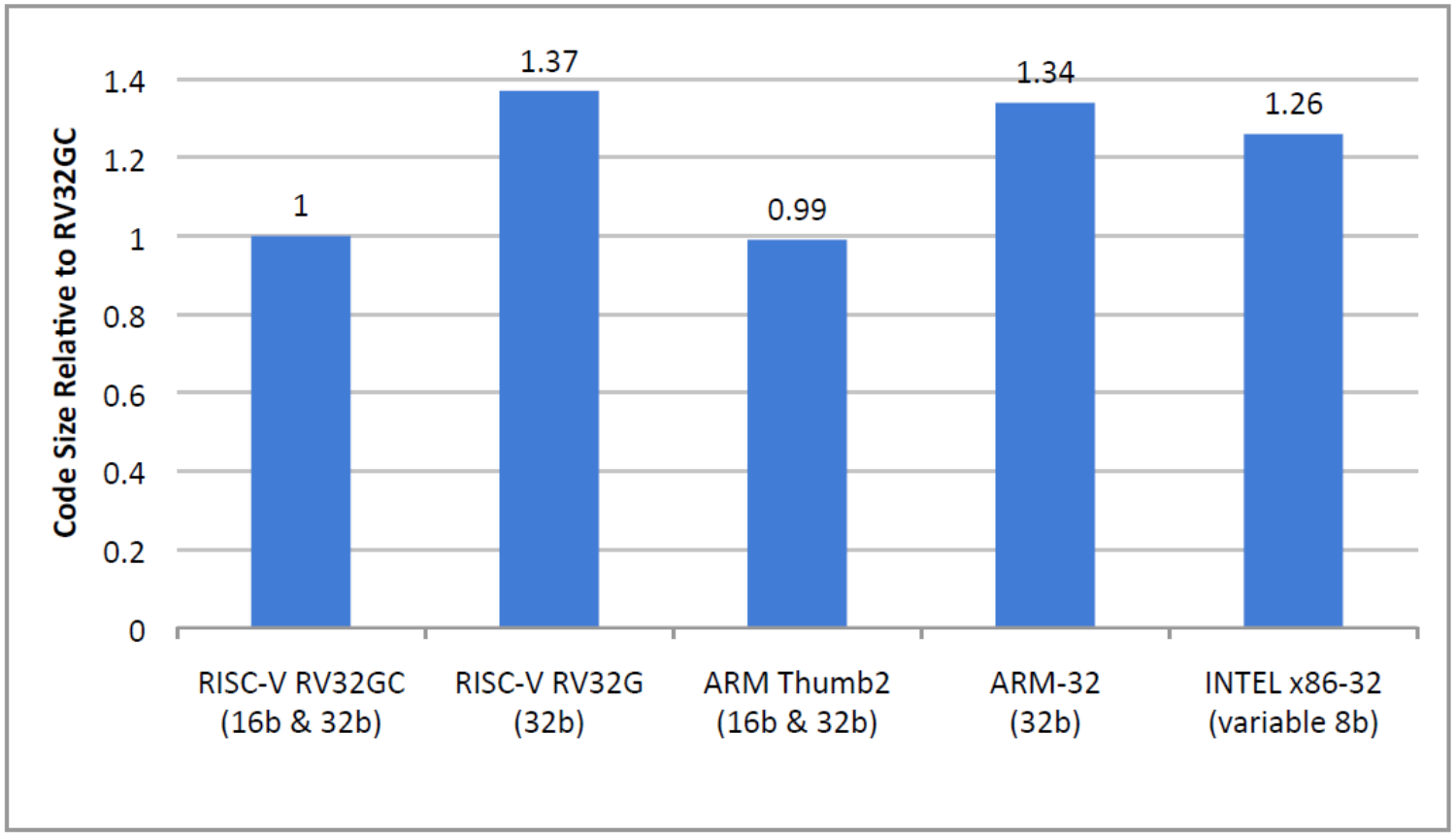
RISC-V is much simpler than ARM

Let's compare a RISC-V Rocket processor to an ARM-32 Cortex-A5 processor in the same technology (TSMC40GPLUS) using the same-sized caches (16 KiB).

The RISC-V die is 0.27mm2 versus 0.53mm2 for ARM-32.

Around twice the area, the ARM-32 Cortex-A5 die costs approximately 4X as much as RISC-V Rocket die.

Relative program sizes for RV32G, ARM-32, x86-32, RV32C, and Thumb-2.



RV32C and Thumb-2 are aimed at small code size. The programs were the SPEC CPU2006 benchmarks using the GCC compilers

More registers

- Since data in a register is so much faster to access than data in memory, the more the number of registers, the better.
- ARM-32 has 16 registers and x86-32 has only 8. Most modern ISAs, including RISC-V, have a relatively generous 32 integer registers.
- More registers make life easier for compilers and assembly language programmers.

Comparing RV32I, ARM-32, MIPS-32, and x86-32 using Insertion Sort

```
void insertion_sort(long a[], size_t n)
{
  for (size_t i = 1, j; i < n; i++) {
    long x = a[i];
    for (j = i; j > 0 && a[j-1] > x; j--) {
      a[j] = a[j-1];
    }
    a[j] = x;
  }
}
```

Insertion sort code in C

Number of instructions and code size for Insertion Sort for these ISAs.

ISA	ARM-32	ARM Thumb-2	MIPS-32	microMIPS	x86-32	RV32I	RV32I+RVC
Instructions	19	18	24	24	20	19	19
Bytes	76	46	96	56	45	76	52

Despite the emphasis on simplicity, the RISC-V version uses the same or fewer instructions, and the code sizes of the architectures are quite close.

In this example, the compare-and-execute branches of RISC-V save as many instructions as do the fancier address modes and the push and pop instructions of ARM-32 and x86-32