Web Aside ASM:IA32 IA32 programming

IA32, the 32-bit predecessor to x86-64, was introduced by Intel in 1985. It served as the machine language of choice for several decades. Most x86 microprocessors sold today, and most operating systems installed on these machines, are designed to run x86-64. However, they can also execute IA32 programs in a backward compatibility mode. As a result, many application programs are still based on IA32. In addition, many existing systems cannot execute x86-64, due to limitations of their hardware or system software. IA32 continues to be an important machine language. You will find that having a background in x86-64 will enable you to learn the IA32 machine language quite readily.

passing of data and control between procedures, as well as storage for local variables. Next, we consider how data structures such as arrays, structures, and unions are implemented at the machine level. With this background in machine-level programming, we can examine the problems of out-of-bounds memory references and the vulnerability of systems to buffer overflow attacks. We finish this part of the presentation with some tips on using the GDB debugger for examining the run-time behavior of a machine-level program. The chapter concludes with a presentation on machine-program representations of code involving floating-point data and operations.

The computer industry has recently made the transition from 32-bit to 64-bit machines. A 32-bit machine can only make use of around 4 gigabytes (2^{32} bytes) of random access memory, With memory prices dropping at dramatic rates, and our computational demands and data sizes increasing, it has become both economically feasible and technically desirable to go beyond this limitation. Current 64-bit machines can use up to 256 terabytes (2^{48} bytes) of memory, and could readily be extended to use up to 16 exabytes (2^{64} bytes). Although it is hard to imagine having a machine with that much memory, keep in mind that 4 gigabytes seemed like an extreme amount of memory when 32-bit machines became commonplace in the 1970s and 1980s.

Our presentation focuses on the types of machine-level programs generated when compiling C and similar programming languages targeting modern operating systems. As a consequence, we make no attempt to describe many of the features of x86-64 that arise out of its legacy support for the styles of programs written in the early days of microprocessors, when much of the code was written manually and where programmers had to struggle with the limited range of addresses allowed by 16-bit machines.

3.1 A Historical Perspective

The Intel processor line, colloquially referred to as *x86*, has followed a long evolutionary development. It started with one of the first single-chip 16-bit microprocessors, where many compromises had to be made due to the limited capabilities of integrated circuit technology at the time. Since then, it has grown to take ad-

vantage of technology improvements as well as to satisfy the demands for higher performance and for supporting more advanced operating systems.

The list that follows shows some models of Intel processors and some of their key features, especially those affecting machine-level programming. We use the number of transistors required to implement the processors as an indication of how they have evolved in complexity. In this table, "K" denotes 1,000 (10^3), "M" denotes 1,000,000 (10^6), and "G" denotes 1,000,000,000 (10^9).

8086 (1978, 29 K transistors). One of the first single-chip, 16-bit microprocessors. The 8088, a variant of the 8086 with an 8-bit external bus, formed the heart of the original IBM personal computers. IBM contracted with then-tiny Microsoft to develop the MS-DOS operating system. The original models came with 32,768 bytes of memory and two floppy drives (no hard drive). Architecturally, the machines were limited to a 655,360-byte address space—addresses were only 20 bits long (1,048,576 bytes addressable), and the operating system reserved 393,216 bytes for its own use. In 1980, Intel introduced the 8087 floating-point coprocessor (45 K transistors) to operate alongside an 8086 or 8088 processor, executing the floating-point instructions. The 8087 established the floating-point model for the x86 line, often referred to as "x87."

80286 (1982, 134 K transistors). Added more (and now obsolete) addressing modes. Formed the basis of the IBM PC-AT personal computer, the original platform for MS Windows.

i386 (1985, 275 K transistors). Expanded the architecture to 32 bits. Added the flat addressing model used by Linux and recent versions of the Windows operating system. This was the first machine in the series that could fully support a Unix operating system.

i486 (1989, 1.2 M transistors). Improved performance and integrated the floating-point unit onto the processor chip but did not significantly change the instruction set.

Pentium (1993, 3.1 M transistors). Improved performance but only added minor extensions to the instruction set.

PentiumPro (1995, 5.5 M transistors). Introduced a radically new processor design, internally known as the *P6* microarchitecture. Added a class of "conditional move" instructions to the instruction set.

Pentium/MMX (1997, 4.5 M transistors). Added new class of instructions to the Pentium processor for manipulating vectors of integers. Each datum can be 1, 2, or 4 bytes long. Each vector totals 64 bits.

Pentium II (1997, 7 M transistors). Continuation of the P6 microarchitecture.

Pentium III (1999, 8.2 M transistors). Introduced SSE, a class of instructions for manipulating vectors of integer or floating-point data. Each datum can be 1, 2, or 4 bytes, packed into vectors of 128 bits. Later versions of this chip

went up to 24 M transistors, due to the incorporation of the level-2 cache on chip.

Pentium 4 (2000, 42 M transistors). Extended SSE to SSE2, adding new data types (including double-precision floating point), along with 144 new instructions for these formats. With these extensions, compilers can use SSE instructions, rather than x87 instructions, to compile floating-point code.

Pentium 4E (2004, 125 M transistors). Added *hyperthreading*, a method to run two programs simultaneously on a single processor, as well as EM64T, Intel's implementation of a 64-bit extension to IA32 developed by Advanced Micro Devices (AMD), which we refer to as x86-64.

Core 2 (2006, 291 M transistors). Returned to a microarchitecture similar to P6. First *multi-core* Intel microprocessor, where multiple processors are implemented on a single chip. Did not support hyperthreading.

Core i7, Nehalem (2008, 781 M transistors). Incorporated both hyperthreading and multi-core, with the initial version supporting two executing programs on each core and up to four cores on each chip.

Core i7, Sandy Bridge (2011, 1.17 G transistors). Introduced AVX, an extension of the SSE to support data packed into 256-bit vectors.

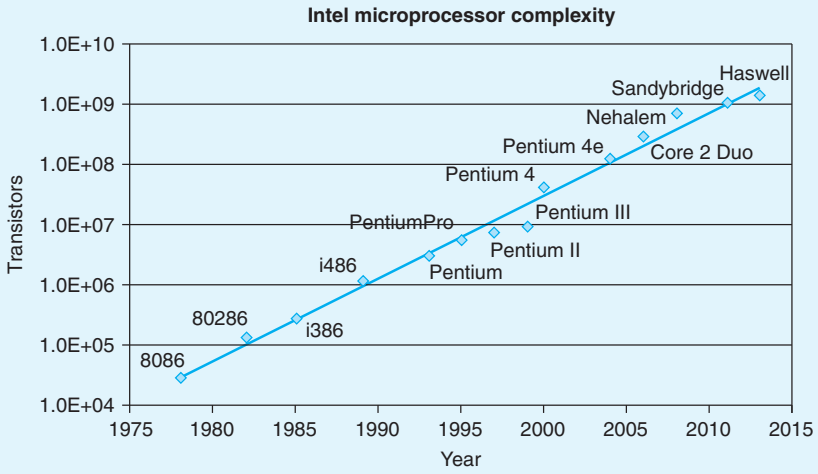
Core i7, Haswell (2013, 1.4 G transistors). Extended AVX to AVX2, adding more instructions and instruction formats.

Each successive processor has been designed to be backward compatible—able to run code compiled for any earlier version. As we will see, there are many strange artifacts in the instruction set due to this evolutionary heritage. Intel has had several names for their processor line, including *IA32*, for "Intel Architecture 32-bit" and most recently *Intel64*, the 64-bit extension to IA32, which we will refer to as *x86-64*. We will refer to the overall line by the commonly used colloquial name "x86," reflecting the processor naming conventions up through the i486.

Over the years, several companies have produced processors that are compatible with Intel processors, capable of running the exact same machine-level programs. Chief among these is Advanced Micro Devices (AMD). For years, AMD lagged just behind Intel in technology, forcing a marketing strategy where they produced processors that were less expensive although somewhat lower in performance. They became more competitive around 2002, being the first to break the 1-gigahertz clock-speed barrier for a commercially available microprocessor, and introducing x86-64, the widely adopted 64-bit extension to Intel's IA32. Although we will talk about Intel processors, our presentation holds just as well for the compatible processors produced by Intel's rivals.

Much of the complexity of x86 is not of concern to those interested in programs for the Linux operating system as generated by the GCC compiler. The memory model provided in the original 8086 and its extensions in the 80286 became obsolete with the i386. The original x87 floating-point instructions became obsolete

Aside Moore's Law



If we plot the number of transistors in the different Intel processors versus the year of introduction, and use a logarithmic scale for the *y*-axis, we can see that the growth has been phenomenal. Fitting a line through the data, we see that the number of transistors increases at an annual rate of approximately 37%, meaning that the number of transistors doubles about every 26 months. This growth has been sustained over the multiple-decade history of x86 microprocessors.

In 1965, Gordon Moore, a founder of Intel Corporation, extrapolated from the chip technology of the day (by which they could fabricate circuits with around 64 transistors on a single chip) to predict that the number of transistors per chip would double every year for the next 10 years. This prediction became known as *Moore's Law*. As it turns out, his prediction was just a little bit optimistic, but also too short-sighted. Over more than 50 years, the semiconductor industry has been able to double transistor counts on average every 18 months.

Similar exponential growth rates have occurred for other aspects of computer technology, including the storage capacities of magnetic disks and semiconductor memories. These remarkable growth rates have been the major driving forces of the computer revolution.

with the introduction of SSE2. Although we see vestiges of the historical evolution of x86 in x86-64 programs, many of the most arcane features of x86 do not appear.

3.2 Program Encodings

Suppose we write a C program as two files `p1.c` and `p2.c`. We can then compile this code using a Unix command line: