Topic 17 RISC vs CISC

©2025 Chris Nielsen – *www.nielsenedu.com*

Vocabulary

stored-program computer (n): a computer where the program instructions are stored in memory – whether the same memory as data, or separate memory.

Von Neumann architecture (n): a computer in which instructions are stored in memory with the data. Modern computers use this architecture at the level of RAM.

Harvard architecture (n): a computer in which instructions are stored in memory, but not in the same memory as the data. Modern high-performance computers use this architecture for level 1 cache.

complex instruction set computer (**CISC**) (n): a computer where a single instruction may perform multi-step operations. Most early processors, including the x86 architecture, follow this design philosophy.

reduced instruction set computer (**RISC**) (n): a computer with few, simple, fixed-length instructions. More complex tasks require multiple instructions to accomplish.

load-store architecture (n): an architecture where the processor can only perform arithmetic and logic operations on values held in the CPU registers, not data in main memory. Memory access is implemented in separate LOAD and STORE instructions. This aligns with RISC principles.

vertical integration (n): a business strategy where a company owns or controls multiple stages of its supply chain, from raw materials to final production and distribution, reducing reliance on external suppliers.

History and Concepts

The First Computers

Early precursors to computers were mechanical and not programmable – only able to perform a single, fixed computation, such as addition.

In 1935, Alan Turing published a paper, *On Computable Numbers, with an Application to the Entscheidungsproblem*, that defined a simple abstract model of a computer, now called a ***Turing machine***, that could compute anything that is computable, and a ***Universal Turing Machine (UTM)*** that could simulate any other *Turing machine* and thus be a general-purpose computer. Turing's work provided the mathematical and theoretical foundation, but was not a blueprint for building a computer.

In 1943, during the second world war, Colossus, the first electronic digital computer, became operational. It was made to help break the German Lorenz cipher (secret code). Although this computer was not general purpose, it was reconfigurable by rewiring plug boards and setting switches. Turing worked on developing this machine.

In 1944, the Harvard Mark I computer used a long, continuous paper tape to store the program instructions, and data was input into either electromechanical registers or punch cards. A computer that stores the instruction and data in different memory is now called a ***Harvard architecture***.

In 1945, after having worked on another reconfigurable computer called ENIAC, John von Neumann published a paper, *First Draft of a Report on the EDVAC*, a report outlining the design for a new machine where data and instructions reside in the same memory. A computer that stores the instructions and data in the same memory is now called a ***von Neumann architecture***. Although EDVAC only became operational in 1951, von Neumann's report inspired others to implement stored-program computers that completed as early as 1948.

The other major innovation proposed in von Neumann's paper was for the use of binary rather than denary for computations. Previous computers performed calculations in denary and avoided the computations involved with a final conversion to decimal for or by the user. Computer logic implements binary computation more simply than it does denary computation.

Complex Instruction Set Computers

With the initial stored-program computers, memory was limited and computers were programmed using machine code – assembly language and assemblers weren't even developed. In this environment, computers were developed such that one machine code instruction would perform multi-step operations. Let's look at a simple example.

Although the x86 instruction set was unveiled in 1978 with the release of the Intel 8086 processor, it is a good example of a *complex instruction set computer* (CISC) instruction set. Consider the following x86 instruction, given in assembly language and machine code:

x86 assembly language instruction	x86 machine code
`movl 0x100(%ebx, %ecx, 4), %eax`	`8B 84 8B 00 01 00 00`

The above instruction moves a 32-bit value to register `EAX` from the memory address that is calculated by: `EBX + (ECX × 4) + 0x100`. That single instruction would be implemented in RISC-V assembly as three separate machine language instructions:

RISC-V assembly language instruction	RISC-V machine code

`slli t0, x12, 2`
`add t0, t0, x11`
`lw x10, 0x100(t0)`
	`00 C2 92 93`
	`00 52 82 B3`
	`10 02 A5 03`

Note also that the x86 machine code is 7 bytes long. With variable-length instructions, it is more complicated for the processor to determine where each instruction starts and ends before it can even begin to decode it.

The IBM Personal Computers and Clones

In the early 1980s, a number of popular home computers, including the Apple II, Commodore 64, and Atari 800, as well as game consoles, including Atari 2600 and Nintendo used a MOS 6502 processor, while other personal computers, including the TRS-80 used a Zilog Z80 processor. These processors were all now what is considered CISC architecture.

In 1981, under pressure to get into the exploding personal computer market, developed the IBM Personal Computer. In order to keep development times short, they had a policy of: buy, don't build. They used standard parts from outside vendors and published the specifications in the now-famous *IBM PC Technical Reference Manual*, including the complete BIOS source code. IBM's contracts with Microsoft and Intel did not forbid them from selling the same parts to other companies.

Because technical details were public and key components (Intel CPU, Microsoft OS) were available to all, other companies started building IBM PC clones, leading to competition and an accidental standardization as "IBM Compatible".

Emergence of the Reduced Instruction Set Computer (RISC)

Although the concept of a minimalist instruction set was researched and developed in the 1970s, the term *reduced instruction set computer* (RISC) was not coined until 1980. Mainstream processors of the time were not considered to have a "complex" instruction set; they just followed the philosophy that hardware should close the "semantic gap" between hardware and high-level programming languages. However, even as the explosion of personal computers was in its infancy, it became apparent that RISC processors were often technically superior. Here we consider why, and how the x86 architecture was still able to win despite its technical drawbacks.

The more complex instructions of CISC processors require complex hardware, and the instructions may be difficult to optimize during real-time execution. Complex hardware in the x86 architecture provides for out-of-order execution, branch prediction, etc. In contrast, with RISC the compiler and assembler is able to examine and optimize source code to produce machine code that has been optimized prior to execution.

The simplified and fixed-length instruction set of a RISC architecture means less decoding logic, most computation done in registers, efficient pipelining, and smaller cache requirements. RISC processors are more power efficient and run cooler – important features for mobile computing in such devices as laptops and cellular phones.

Despite the advantages of a RISC architecture, computers designed with RISC philosophy were generally sold in niche markets such as expensive high-end servers and workstations for engineering work (Such as SPARC computers running the Solaris UNIX operating system). The “Wintel” ecosystem – inexpensive IBM PC clones and the vast library of affordable software for the x86 was the “killer app”. Over time, even those niche high-end markets were invaded and ruled by the x86 architecture.

Modern x86 CISC interface, RISC core

The Intel Pentium (P5), released in 1993, was the last Intel processor to have the execution units inside the processor core to directly handle x86 instructions. There was no out-of-order execution of instruction, and only simple branch prediction.

The Intel Pentium Pro (P6), released in 1995, was the first x86 processor to implement hardware translation of the CISC instruction set to internal micro-operations (μops). All modern x86 processors continue this design. At the heart, the execution units work on simple, RISC-like fix-format operations with a load-store model. After decoding the x86 instructions into μops, the processor is able to reorder the instructions to optimize execution.

Thus, modern x86 computers are able to take a complex, variable-length CISC instruction set and run as fast as a simple RISC one.

What was Apple Computer Doing?

In the late 1970s and early 1980s, prior to the dominance of the IBM PC, Commodore, Tandy, and Apple computers were very popular. Commodore and Tandy competed as low-margin consumer electronics and failed against countless IBM PC clones. Apple positioned themselves as a more premium product using *vertical integration* – controlling the hardware, operating system, and core software – providing a seamless, cohesive, pleasant user experience, and fostering customer loyalty (or one might say creating a sticky, high-margin business). IBM PCs were cheap and flexible, while Apple computers, although more expensive, were easy to use and dependable.

Because of the close architecture and vertical integration, Apple computers were able to seamlessly change their underlying architecture while only slowly evolving the user interface to improve the user experience.

The first MacOS ran on Apple Macintosh computers that used the Motorola 68000 series processors, CISC architecture processors.

For performance, Apple transitioned to the PowerPC processor – a RISC architecture developed by IBM, Motorola, and Apple. However, the existing software that ran on Macintosh computers was compiled to machine code for Motorola 68000 processors. To allow this software to run on the PowerPC, Apple developed software, named Rosetta, to dynamically translate the Motorola 68000 machine code into PowerPC machine code. This had a performance hit, but by translating blocks of code and caching translated code, it could near the performance of native PowerPC code. Over time, older Motorola 68000 software would be replaced by software compiled into PowerPC machine code. Such a move to a new processor architecture would not be practical for the IBM PC clone computers as it would be difficult to convince and organize so many different manufacturers of the hardware to agree to transition in any one direction.

After PowerPC, Apple moved its computers to x86 processors (CISC), and most recently to its own version of ARM processor (RISC), the Apple M-Series processors.

The End of x86 Dominance

Despite the advantages of RISC-based processors, x86 has maintained its dominance of the personal computer market – largely due to its vast software library. Disruption of this near monopoly has not come from an innovative computer manufacturer, but from smart phones.

Although high-performance, x86 was not power-efficient, so not suitable for use in smart phones. Apple released the first iPhone in 2007, which used an ARM processor (which is RISC-based). The Linux operating system kernel had been ported to work on ARM processors by 1996, and Google released a modified Linux operating system, called the Android operating system, as an open-source smart phone operating system in late 2007, and the first Android phones became available late 2008. Just as the open architecture of the initial IBM PC aided widespread adoption, the open sourcing of the Android operating system also largely united the what was until then fragmented proprietary systems.

In 2009, Apple started to develop its own ARM-based processors, which they used in their 2010 iPhone 4. and in 2020, they announce that they would transition all Mac computers from Intel x86 processors to its own Apple silicon – the ARM-based M-Series processors. These Apple processors have been able to compete on performance with x86 processors with much better power efficiency.

The Windows operating system announced it would begin to support ARM processors by releasing Windows on ARM in 2011. This operating system requires ARM-specific device drivers, but applications may be either recompiled for ARM, or run thorough x86 emulation software similar to the Rosetta software for Apple computers.

In 2000, x86 held an estimated 70% of compute output. By 2020, ARM-based processors accounted for an estimated 82% of global output. However this is largely due to the smart phone and embedded systems markets. For personal computers, prior to 2020, ARM's market share was around 2%. By 2022, due to Apple moving to ARM, the market share increased to about 12.8%. The forecast for 2027 is that ARM will have over 25% of the market share – due to Apple computers as well as Windows on Arm.

RISC-V

RISC-V is a modern, open-source instruction set architecture (ISA) based on RISC principles, which may prove to be a significant challenger to the long-established x86 and ARM architectures. The architecture was created in 2010 at UC Berkley and, unlike 86 or ARM, is free to both use and customize without licensing fees.

Due to low cost and customizability, RISC-V processors have achieved significant market penetration in embedded systems and the internet of things (IoT), and is growing in AI, data center, and automotive applications. It has achieved its first 100 billion core shipments in just 12 years, significantly faster than the decades it took for x86 or ARM.

The challenges to the system is a fragmented software ecosystem. However, there are numerous Linux distributions that will run on RISC-V processors, including Fedora (since 2018), Alpine (since 2019), Ubuntu (since 2020). Alpine Linux is a RISC-V favorite due to its small footprint, security focus, and adaptability.