Computer Instruction Set Evolution

Dr.B.W.Stuck

MIT: SBEE 1968, SMEE 1969, EE 1970, PhD 1972

Google Scholar profile
https://scholar.google.com/citations?user=ykewuCAAAAAJ&hl=en

Outline

1960s

- IBM Instruction Set History
- Maurice Wilkes' Control Store Concept

1970s

- Complex Instruction Set Computing
- Reduced Instruction Set Computing
- Moore's Law, Denard's Law, Amdahl's Law

1980s

- Domain Specific Architectures: DSP, GPU, TPU, SDNU

2000s

- Concurrent Parallel Processing, Open Standards, Agile Hardware Development

IBM: Instruction Set History

Software interacts with digital hardware through an interface called an instruction set architecture (ISA)

By the early 1960s, IBM had four incompatible lines of computers, each with its own ISA, software stack, I/O subsystem and market niche

- Large business
- Small business
- Scientific
- Real time (electric power plants, rocket launches)

In the early 1960s, Fred Brooks and Gene Amdahl at IBM thought they could create a single ISA that would unify all four segments, with inexpensive 8-bit data paths on up to 64-bit data paths

Data paths are relatively easy to narrow or widen, the challenge is to control this hardware

The new product line, called System/360, was built with a hybrid hardware technology with ceramic substrates hosting resistor capacitors, inductors, diodes and transistors; System/370 and all subsequent products would replace this with integrated circuitry

Maurice Wilkes' Control Store

A control store is a two dimensional array

- Each column of the array corresponds to one control line
- Each row of the array is a microinstruction
- Writing microinstructions was called microprogramming

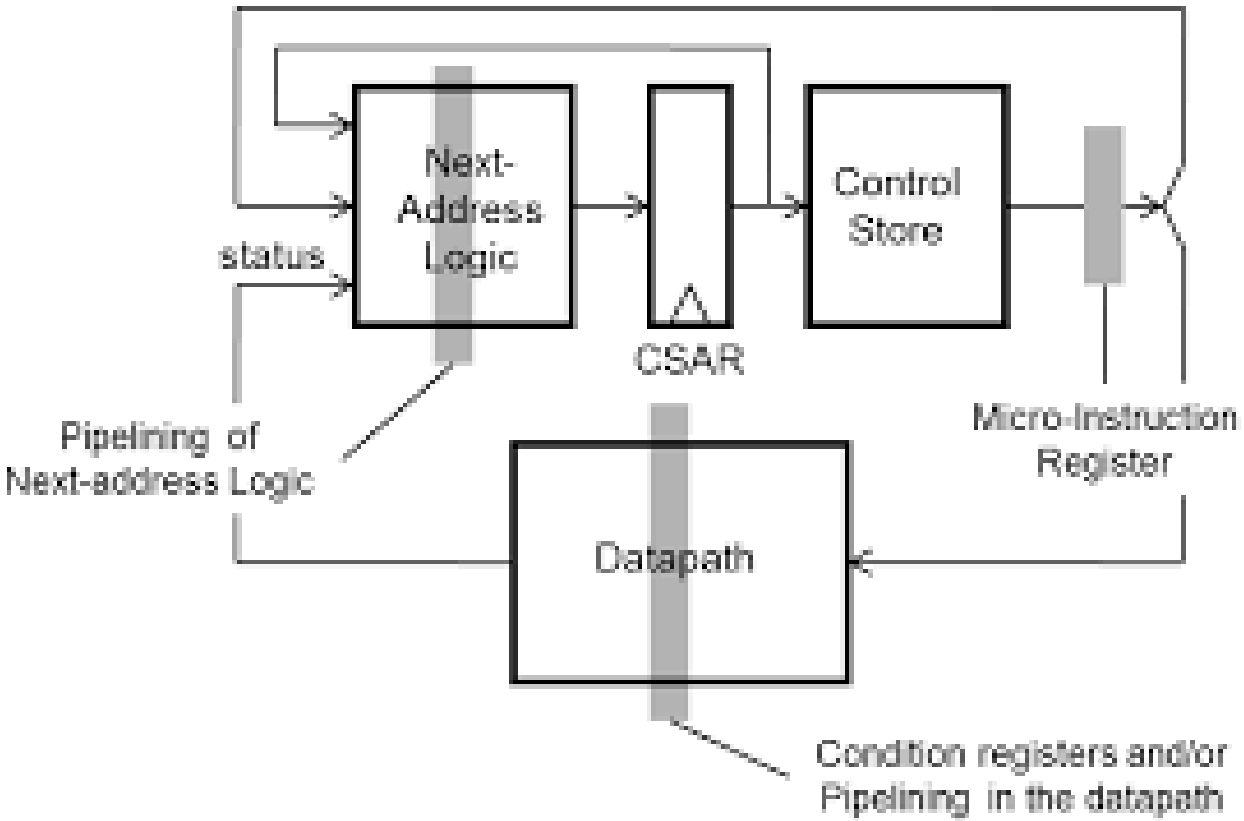
A control store contains an instruction set architecture interpreter written in microinstructions:

- Execution of a conventional instruction can take several microinstructions
- Control store was implemented in memory which was less costly than custom logic

Four Models IBM System/360

Model	M30	M40	M50	M65
Datapath Width	8 bits	16 bits	32 bits	64 bits
Control Store Size	4k x 50	4k x 52	2.75k x 85	2.75k x 87
Clock Rate	1.3 MHz	1.6 MHz	2 MHz	5 Mhz
Memory Capacity	8-64 KiB	16-256 KiB	64-512 KiB	128-1024 KiB
Commercial Perf	29K IPS	75K IPS	169K IPS	567 IPS
Scientific Perf	10.2K IPS	40K IPS	133K IPS	563K IPS
Price ($1964)	$192,000	$216,000	$460,000	$1,080,000

Control Store: Maurice Wilkes



Control Store: a two dimensional array, with each **column** of the array corresponding to one **control line**, each **row** was a **microinstruction**, and writing microinstructions was called **microprogramming.**

From Complex to Reduced Instruction Set Computers

The 1970s saw several investigations into complex instruction set computers (CISC)

UNIX demonstrated that even operating systems could use high level languages, so the question became What instruction should compilers generate rather than What assembly language would programmers use

John Cocke and IBM colleagues developed simpler ISAs and compilers for minicomputers, used only simple register-register operations and load-store data transfers of the IBM 360 ISA and found programs ran up to 3X faster on simpler subset

Emer and Clark found 20% of VAX instructions needed 60% of microcode in control store and represented 0.2% of execution time

These data points and the shift to higher level languages led to the opportunity to switch from CISC to RISC

1. RISC instructions typically as simple as microinstructions and could be directly executed by hardware
2. Fast memory formerly used as microcode interpreter of a CISC ISA was repurposed to be a cache of RISC instructions
3. Register allocators based on Gregory Chaitin's graph coloring scheme made it much easier for compilers to efficiently use registers, which benefits register to register ISAs
4. Moore's Law implied there would be enough transistors to include full 32 bit datapath, along with instruction and data caches, on a single chip

Reduced Instruction Set Computers: the 1960s

- A number of systems, going back to the 1960s, have been credited as the first Reduced Instruction Set (RISC) architecture, partly based on their use of the load–store approach. The term RISC was coined by David Patterson of the Berkeley RISC project, although somewhat similar concepts had appeared before.

- The CDC 6600 designed by Seymour Cray in 1964 used a load–store architecture with only two addressing modes (register+register, and register+immediate constant) and 74 operation codes, with the basic clock cycle being 10 times faster than the memory access time. Partly due to the optimized load–store architecture of the CDC 6600, Jack Dongarra says that it can be considered a forerunner of modern RISC systems, although a number of other technical barriers needed to be overcome for the development of a modern RISC system.

Reduced Instruction Set Computers 1970s: IBM Project 801

- Michael J. Flynn views the first RISC system as the IBM 801 design, begun in 1975 by John Cocke and completed in 1980. The 801 developed out of an effort to build a 24-bit high-speed processor to use as the basis for a digital telephone switch. To reach their goal of switching 1 million calls per hour (300 per second) they calculated that the CPU required performance on the order of 12 million instructions per second (MIPS), compared to their fastest mainframe machine of the time, the 370/168, which performed at 3.5 MIPS.
- The design was based on a study of IBM's extensive collection of statistics gathered from their customers. This demonstrated that code in high-performance settings made extensive use of processor registers, and that they often ran out of them. This suggested that **additional registers would improve performance**.
- Additionally, they noticed that compilers generally ignored the vast majority of the available instructions, especially orthogonal addressing modes. Instead, they selected the fastest version of any given instruction and then constructed small routines using it. This suggested **that the majority of instructions could be removed without affecting the resulting code**. These two conclusions worked in concert; **removing instructions would allow the instruction opcodes to be shorter, freeing up bits in the instruction word which could then be used to select among a larger set of registers.**

Reduced Instruction Set Computers 1970s: IBM Project 801

- The telephone switch program was canceled in 1975, but by then the team had demonstrated that the same design would offer significant performance gains running just about any code. In simulations, they showed that a compiler tuned to use registers wherever possible would run code about three times as fast as traditional designs. Somewhat surprisingly, the same code would run about 50% faster even on existing machines due to the improved register use. In practice, their experimental PL/8 compiler, a slightly cut-down version of PL/I, consistently produced code that ran much faster on their existing mainframes.
- A 32-bit version of the 801 was eventually produced in a single-chip form as the IBM ROMP in 1981, which stood for 'Research OPD [Office Products Division] Micro Processor'. This CPU was designed for "mini" tasks, and found use in peripheral interfaces and channel controllers on later IBM computers. It was also used as the CPU in the IBM RT PC in 1986, which turned out to be a commercial failure. Although the 801 did not see widespread use in its original form, it inspired many research projects, including ones at IBM that would eventually lead to the IBM POWER architecture.

1970s: RISC and MIPS

- By the late 1970s, the 801 had become well known in the industry. This coincided with new fabrication techniques that were allowing more complex chips to come to market. The Zilog Z80 of 1976 had 8,000 transistors, whereas the 1979 Motorola 68000 (68k) had 68,000. These newer designs generally used their newfound complexity to expand the instruction set to make it more orthogonal. Most, like the 68k, used microcode to do this, reading instructions and re-implementing them as a sequence of simpler internal instructions. In the 68k, a full 1⁄3 of the transistors were used for this microcoding.
- In 1979, David Patterson was sent on a sabbatical from the University of California, Berkeley to help DEC's west-coast team improve the VAX microcode. Patterson was struck by the complexity of the coding process and concluded it was untenable.[15] He first wrote a paper on ways to improve microcoding, but later changed his mind and decided microcode itself was the problem. With funding from the DARPA VLSI Program, Patterson started the Berkeley RISC effort. The Program, practically unknown today, led to a huge number of advances in chip design, fabrication, and even computer graphics. Considering a variety of programs from their BSD Unix variant, the Berkeley team found, as had IBM, that most programs made no use of the large variety of instructions in the 68k.

1980s: RISC and MIPS

- Patterson's early work pointed out an important problem with the traditional "more is better" approach; even those instructions that were critical to overall performance were being delayed by their trip through the microcode. If the microcode was removed, the programs would run faster. And since the microcode ultimately took a complex instruction and broke it into steps, there was no reason the compiler couldn't do this instead. These studies suggested that, even with no other changes, one could make a chip with 1⁄3 fewer transistors that would run faster. In the original RISC-I paper they noted:
- **Skipping this extra level of interpretation appears to enhance performance while reducing chip size.**
- It was also discovered that, on microcoded implementations of certain architectures, complex operations tended to be slower than a sequence of simpler operations doing the same thing. This was in part an effect of the fact that many designs were rushed, with little time to optimize or tune every instruction; only those used most often were optimized, and a sequence of those instructions could be faster than a less-tuned instruction performing an equivalent operation as that sequence. One infamous example was the VAX's INDEX instruction.

1980s: RISC and MIPS

- The Berkeley work also turned up a number of additional points. Among these was the fact that programs spent a significant amount of time performing subroutine calls and returns, and it seemed there was the potential to improve overall performance by speeding these calls.
- This led the Berkeley design to select a method known as register windows which can significantly improve subroutine performance although at the cost of some complexity. They also noticed that the majority of mathematical instructions were simple assignments; only 1⁄3 of them actually performed an operation like addition or subtraction. But when those operations did occur, they tended to be slow. This led to far more emphasis on the underlying arithmetic data unit, as opposed to previous designs where the majority of the chip was dedicated to control and microcode.

1980s: RISC and MIPS

- The resulting Berkeley RISC was based on gaining performance through the use of pipelining and aggressive use of register windowing. In a traditional CPU, one has a small number of registers, and a program can use any register at any time. In a CPU with register windows, there are a huge number of registers, e.g., 128, but programs can only use a small number of them, e.g., eight, at any one time.
- A program that limits itself to eight registers per procedure can make very fast procedure calls: The call simply moves the window "down" by eight, to the set of eight registers used by that procedure, and the return moves the window back.
- The Berkeley RISC project delivered the RISC-I processor in 1982. Consisting of only 44,420 transistors (compared with averages of about 100,000 in newer CISC designs of the era), RISC-I had only 32 instructions, and yet completely outperformed any other single-chip design, with estimated performance being higher than the VAX. They followed this up with the 40,760-transistor, 39-instruction RISC-II in 1983, which ran over three times as fast as RISC-I.

1980s: RISC and MIPS

- As the RISC project began to become known in Silicon Valley, a similar project began at Stanford University in 1981. This MIPS project grew out of a graduate course by John L. Hennessy, produced a functioning system in 1983, and could run simple programs by 1984. The MIPS approach emphasized an aggressive clock cycle and the use of the pipeline, making sure it could be run as "full" as possible. The MIPS system was followed by the MIPS-X and in 1984 Hennessy and his colleagues formed MIPS Computer Systems to produce the design commercially. The venture resulted in a new architecture that was also called MIPS and the R2000 microprocessor in 1985.
- The overall philosophy of the RISC concept was widely understood by the second half of the 1980s, and led the designers of the MIPS-X to put it this way in 1987. The goal of any instruction format should be:

1. simple decode
2. simple decode
3. simple decode.

- Any attempts at improved code density at the expense of CPU performance should be ridiculed at every opportunity.

1980s: Commercialization

- In the early 1980s, significant uncertainties surrounded the RISC concept. One concern involved the use of memory; a single instruction from a traditional processor like the Motorola 68k may be written out as perhaps a half dozen of the simpler RISC instructions. In theory, this could slow the system down as it spent more time fetching instructions from memory. But by the mid-1980s, the concepts had matured enough to be seen as commercially viable.
- Commercial RISC designs began to emerge in the mid-1980s. The first MIPS R2000 appeared in January 1986, followed shortly thereafter by Hewlett-Packard's PA-RISC in some of their computers. In the meantime, the Berkeley effort had become so well known that it eventually became the name for the entire concept.
- In 1987 Sun Microsystems began shipping systems with the SPARC processor, directly based on the Berkeley RISC-II system. The US government Committee on Innovations in Computing and Communications credits the acceptance of the viability of the RISC concept to the success of the SPARC system. The success of SPARC renewed interest within IBM, which released new RISC systems by 1990 and by 1995 RISC processors were the foundation of a $15 billion server industry.

Intel: 8080->80x86 vs iAPX-432 The Road Not Taken

1970s microprocessors such as Intel 8080: 8 bit data path, assembly language programmed

Gordon Moore created 8800 project: 32 bit capability based addressing, objected oriented architecture, variable-bit-length instructions, and operating system written in Ada, renamed iAPX-432 and announced in 1981, discontinued in 1986 after Intel extended 16 bit 8086 to 32 bit 80386

1979: Intel launched 8086, extending 8080 8 bit registers and its instruction set to 16 bits

1980: IBM selects 8 bit version of 8086 for its Personal Computer, and Intel continually extended 8080 extension to 80286, 80386, 80486, higher clock rates and larger data paths

Marketplace chose 8086 rather than Motorola 68000 or iAPX-432

1980s: VLIW, EPIC, Itanium

The next ISA beyond RISC and CISC was Very Long Instruction Word (VLIW) and Explicitly Parallel Instruction Computer (EPIC)

If a single instruction could specify say six independent operations (two data transfers, two integer operations, and two floating point operations) and compiler technology could efficiently assign operations into six instruction slots, the hardware could be simpler, shifting work from the hardware to the compiler

Intel and Hewlett-Packard designed a 64 bit processor to replace the 32 bits x86 called Itanium

The EPIC approach worked well for highly structured floating point program, but could not achieve high performance for integer programs that had less predictable cache misses or less predictable branches

Donald Knuth noted: The Itanium approach was supposed to be so terrific until it turned out that the compilers were impossible to write

Marketplace abandoned Itanium in favor of 64 bit version of x86

VLIW still matches a set of applications with small programs and simpler branches and omits caches, such as digital signal processing

1980s: Commercialization

- By the later 1980s, the new RISC designs were easily outperforming all traditional designs by a wide margin. At that point, all of the other vendors began RISC efforts of their own. Among these were the DEC Alpha, AMD Am29000, Intel i860 and i960, Motorola 88000, IBM POWER, and, slightly later, the IBM/Apple/Motorola PowerPC.
- Many of these have since disappeared due to them often offering no competitive advantage over others of the same era. Those that remain are often used only in niche markets or as parts of other systems; of the designs from these traditional vendors, only SPARC and POWER have any significant remaining market.

1980s: Advanced RISC Machines (ARM) Commercialization

- The ARM architecture is illustrative of the adaptations made by RISC vendors to respond to changing competitive circumstances, being first introduced to deliver higher performance in desktop computers such as the Acorn Archimedes, but also being introduced in embedded applications such as laser printer raster image processing.
- ARM, in partnership with Apple, developed a low-power design and then specialized in that market, which at the time was a niche.
- With the rise in mobile computing, especially after the introduction of the iPhone, ARM became the most widely used high-end CPU design in the market.
- Competition between RISC and conventional CISC approaches was also the subject of theoretical analysis in the early 1980s, leading, for example, to the iron law of processor performance.

RISC vs CISC in PC and post PC Era

AMD and Intel used 500 person design teams and superior semiconductor technology to close the performance gap between x86 and RISC

Performance advantages of pipelining simple vs complex instructions, the instruction decoder translated the complex x86 instructions into internal RISC-like microinstructions on the fly

AMD and Intel then pipelined the execution of RISC microinstructions

Any ideas RISC designers used to improve performance could be incorporated into x86

- Separate instruction and data caches
- Second level caches on chip
- Deep pipelines
- Fetching and executing several instructions simultaneously

Mobile smart phones from 2007 on and Internet of things (IoT) placed value on die area, performance and energy efficiency to the disadvantage of CISC ISA, typically using Systems on a Chip (SoC) based on RISC processors from ARM: 99% of all 32 bit and 64 bit processors today are RISC

CISC won the later stages of the PC era, RISC is winning the post PC era, no new commercially successful CISC ISAs in decades, RISC has won 35 years after introduction

Current Semiconductor Technology Challenges

Since the late 1970s, the semiconductor technology of choice has been metal oxide semiconductor (MOS), first n-type metal oxide (nMOS), then complementary metal oxide (CMOS)

Gordon Moore predicted in 1965 a doubling of transistor density per unit area per year, and in 1975 projected it to doubling every two years, and around 2000 this began to slow

In 1974 Robert Denard showed that the MOSFET scaling law states roughly that, as transistors get smaller, their power density stays constant, so that the power use stays in proportion with area; both voltage and current scale with length; this began to slow significantly in 2007

Between 1986 and 2002 instruction level parallelism (ILP) was exploited to gain performance, leading to an annual performance gain of 50%

ILP caused greater inefficiency. Consider an illustrative example of a processor with a 15 stage pipeline, it can issue four instructions every clock cycle, so 60 instructions are in the pipeline on average; roughly 25% of the instructions include branches, so to keep the pipeline full, branches are predicted and code is speculatively placed into the pipeline for execution, and this is the source of performance gain and also inefficiency, because when the branches are mispredicted, the processor must throw away the incorrectly speculated instructions, wasting computation work and energy, and the internal processor state must be restored to the state that existed before the mispredicted branch, expending more time and energy

Amdahl's Law: if we have 64 processors but one processor must execute one particular piece of code before the other processors can execute in parallel, only (1/64)=1.5% of the time is serial, the speedup is 35x, but the power needed is proportional to 64 processors, so approximately (64-35)/64=45% of the energy is wasted

Multiple Processor Core vs Single Core

Multicore shifted responsibility for identifying parallelism and deciding how to exploit it to the programmer and the language

Multicore does not resolve the energy-efficient computation need exacerbated by the end of Denard scaling

Increasing the number of cores on a chip meant power is also increasing and must be removed as heat

Managing heat dissipation led to slowing down processors and turning off idle cores

1980s: The Iron Law of Processor Performance

- In computer architecture, the iron law of processor performance (or simply iron law of performance) describes the performance trade-off between complexity and the number of primitive instructions that processors use to perform calculations. This formulation of the trade-off spurred the development of **Reduced Instruction Set Computers (RISC)** whose instruction set architectures (ISAs) leverage a smaller set of core instructions to improve performance. The term was coined by Douglas Clark based on research performed by Clark and Joel Emer in the 1980s.
- **Time/Program=(Instructions/Program)X(Clock Cycles/Instruction)X(Time/Clock Cycle)**
- While the iron law is credited for sparking the development of RISC architectures, it does not imply that a simpler ISA is always faster. If that were the case, the fastest ISA would consist of simple binary logic. A single CISC instruction can be faster than the equivalent set of RISC instructions when it enables multiple micro-operations to be performed in a single clock cycle. In practice, however, the regularity of RISC instructions allowed a pipelined implementation where the total execution time of an instruction was (typically) ~5 clock cycles, but each instruction followed the previous instruction ~1 clock cycle later. CISC processors can also achieve higher performance using techniques such as modular extensions, predictive logic, compressed instructions, and macro-operation fusion.

New Directions: Compiler Technology

What other approaches might be promising to improve performance and enable new software capabilities?

First, existing techniques for building software make extensive use of high level languages with dynamic typing and storage management. Typically these languages are interpreted and execute very inefficiently. Leiserson used a small example-performing a matrix multiplication-to illustrate this inefficiency, suggesting factors of 100 to 1000 speedups might be achievable in practice possibly with new compiler technology and architecture

Implementation	Speedup
Python	1
C	47
C+parallel loops	366
C+parallel loops+cache optimization	6727
C+parallel loops+memory optimization+SIMD	62806

New Directions: Accelerators

Second, a more hardware centric approach is to design architectures tailored to a specific problem domain and offer significant performance and efficiency gains for that domain (Domain Specific Architecture or DSA)

DSAs effectively accelerate an application compared to executing the application on a general purpose processor

- graphics processing units (GPUs)
- neural network processor used for deep learning, and
- processors for software defined networks

DSAs exploit a more efficient form of parallelism for the specific domain

- Single instruction multiple data (SIMD) parallelism is more efficient than multiple instruction multiple data (MIMD) because it needs to fetch only one instruction stream and processing units operate in lockstep
- DSAs may use VLIW approaches to ILP rather than speculative out of order mechanisms, since the control mechanisms are much simpler
- DSAs can make more effective use of memory hierarchy (e.g., accessing a block in a 32KB cache involves an energy cost 200x higher than a 32-bit integer add, so optimizing memory accesses is critical; using multilevel caches bandwidth and hides latency in relatively slow, off chip DRAMs
- DSAs can use less precision when it is adequate: in deep neural networks, inference regularly uses 4-,8-,or 16-bit integers, improving both data and computational throughput
- DSAs benefit from targeting programs written in domain specific languages (DSLs) that expose more parallelism, improve the structure and representation of memory access, and make it easier to map the application efficiently to a domain specific processor

Domain Specific Languages

DSLs can make vector, dense matrix, and sparse matrix operations explicit, enabling the DSL compiler to map operations efficiently to the processor

Examples include

- MatLab (a language for operating on matrices)
- TensorFlow (a language used for programming DNNs)
- P4 (a language for programming SDNs)
- Halide (a language for image processing specifying high level transformations)

When using DSLs how can you retain enough architecture independence that software written in a DSL can be ported to different architectures while also achieving high efficiency, e.g., XLA system translates TensorFlow to heterogeneous processors

Example: Google TPU v1, designed to accelerate neural net inference. The main computational unit is a matrix, providing 256x256 multiply-accumulates every clock cycle. Instead of caches, a local memory of 24MB is used. Both activation memory and weight memory are linked through user controlled high bit rate memory channel. TPU is 29x faster than general purpose CPU, requiring less than half the power->80x gain in performance

Domain Specific Architectures

DSPs: digital signal processor optimized for communications functions

GPUs: Nvidia GPUs use many cores, each with a large register file, many hardware threads, and caches

TPUs: Google TPUs rely on large two-dimensional systolic multipliers and software-controlled on-chip memory

FPGAs: MicroSoft deploys field programmable gate arrays (FPGAs) in its data centers it tailors to neural network applications

CPUs: Intel offers CPUs with many cores enhanced by large multi-level caches and one dimensional SIMD instructions, the kinds of FPGAs used by MicroSoft, and a new neural networkprocessor that is closer to a TPU than to a CPU

Dozens of startups are pursuing their own approaches

Open Architectures

Inspired by the success of open source software, and open source knowledge repositories such as Wikipedia, why not open ISAs?

There is a need for industry-standard open ISAs so the community can create open source cores, to complement proprietary ISAs

RISC-V: the fifth RISC architecture developed at the University of California at Berkeley has a community that maintains the architecture under the RISC-V Foundation, with a modular instruction set:

- Small base of instructions run the full open source software stack
- Multiple optional standard extensions designs can include or omit
- 32-bit and 64-bit address versions
- Base software stack runs fine even if new extensions are NOT employed
- Standard extensions: M (Integer multiply/divide), A (Atomic memory operations), F/D (single/double precision floating point), and C (Compressed instructions)

Nvidia Deep Learning

Open simple architectures are synergistic with security

Agile Hardware Development

Modern electronic computer aided design (ECAD) tools raise the level of abstraction, enabling agile development, and this higher level of abstraction increases reuse across designs

A software simulator is the easiest and quickest place to make changes, if the software simulator could satisfy rapid iteration every month or so

FPGAs can run hundreds of times faster than a detailed software simulator, so the FPGAs can run operating systems and full benchmarks like those from the Standard Performance Evaluation Corporation (SPEC); Amazon Web Services offers FPGAs in the cloud, so architects can use FPGAs without having to first buy hardware and set up a lab

ECAD tools can generate a chip layout, including die area and electric power consumption

Fabricate chips!

Security

Computer security was overlooked in all the discussion so far!

Large software systems continue to have many security flaws, with risk amplified due to the vast increasing amount of personal information available on line and the use of cloud software which shares physical hardware among potential adversaries

Hardware support for virtual machines and encryption began to appear, but this in turn led to new security flaws and new vulnerabilities

Meltdown and Spectre use so called side channel attacks whereby information is leadked by observing the time taken for a task and converting information invisible at the ISA level into a timing visible attribute; one variant of Spectre allows leakage of information without the attacker loading code onto the target processor

New vulnerabilities are being discovered every day; as one example, it is possible to have an Apple iPhone powered off and yet take it over and use the microphone

2010s: Open Source Instruction Set Architecture

- Since 2010, a new open source instruction set architecture (ISA), RISC-V, has been under development at the University of California, Berkeley, for research purposes and as a free alternative to proprietary ISAs.
- As of 2014, version 2 of the user space ISA is fixed. The ISA is designed to be extensible from a barebones core sufficient for a small embedded processor to supercomputer and cloud computing use with standard and chip designer–defined extensions and coprocessors. It has been tested in silicon design with the ROCKET SoC, which is also available as an open-source processor generator in the CHISEL language.

B.W.Stuck Biography

- MIT, 1964-1972: SBEE 1968, SMEE 1969, EE 1970, PhD 1972
- Bell Laboratories 1972-1984:
  - Lectured at over 50 universities and research institutes in North America, Western Europe, Soviet Union, Japan.
  - Worked on 20 UNIX application systems, consulted on another 80, created a class for a masters level computer science program, which was taught at Columbia University three times, leading to publishing a book, A Computer and Communication Network Performance Analysis Primer, Prentice Hall, 1985
- Independent consulting, 1984-1998
- Venture capital, 1998-today