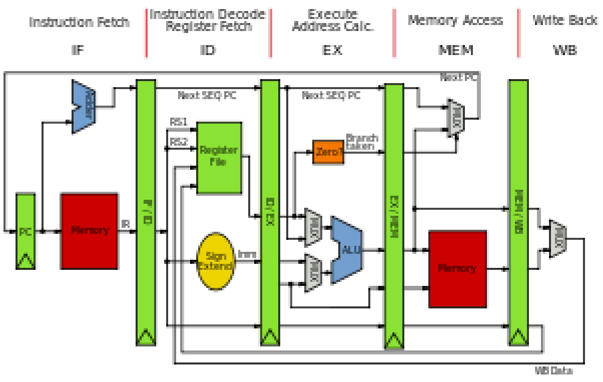
WIKIPEDIA

Computer architecture

In computer engineering, **computer architecture** is a set of rules and methods that describe the functionality, organization, and implementation of computer systems. Some definitions of architecture define it as describing the capabilities and programming model of a computer but not a particular implementation.[1] In other definitions computer architecture involves instruction set architecture design, microarchitecture design, logic design, and implementation.[2]



A pipelined implementation of the MIPS architecture. Pipelining is a key concept in computer architecture.

Contents

History

The first documented computer architecture was in the correspondence between Charles Babbage and Ada Lovelace, describing the analytical engine. When building the computer Z1 in 1936, Konrad Zuse described in two patent applications for his future projects that machine instructions could be stored in the same storage used for data, i.e. the stored-program concept.[3][4] Two other early and important examples are:

- John von Neumann's 1945 paper, First Draft of a Report on the EDVAC, which described an organization of logical elements;[5] and
- Alan Turing's more detailed *Proposed Electronic Calculator* for the Automatic Computing Engine, also 1945 and which cited John von Neumann's paper.[6]

The term “architecture” in computer literature can be traced to the work of Lyle R. Johnson, Frederick P. Brooks, Jr., and Mohammad Usman Khan, all members of the Machine Organization department in IBM’s main research center in 1959. Johnson had the opportunity to write a proprietary research communication about the Stretch, an IBM-developed supercomputer for Los Alamos

National Laboratory (at the time known as Los Alamos Scientific Laboratory). To describe the level of detail for discussing the luxuriously embellished computer, he noted that his description of formats, instruction types, hardware parameters, and speed enhancements were at the level of “system architecture” – a term that seemed more useful than “machine organization.”[7]

Subsequently, Brooks, a Stretch designer, started Chapter 2 of a book (Planning a Computer System: Project Stretch, ed. W. Buchholz, 1962) by writing,[8]

> Computer architecture, like other architecture, is the art of determining the needs of the user of a structure and then designing to meet those needs as effectively as possible within economic and technological constraints.

Brooks went on to help develop the IBM System/360 (now called the IBM zSeries) line of computers, in which “architecture” became a noun defining “what the user needs to know”.[9] Later, computer users came to use the term in many less-explicit ways.[10]

The earliest computer architectures were designed on paper and then directly built into the final hardware form.[11] Later, computer architecture prototypes were physically built in the form of a transistor–transistor logic (TTL) computer—such as the prototypes of the 6800 and the PA-RISC—tested, and tweaked, before committing to the final hardware form. As of the 1990s, new computer architectures are typically "built", tested, and tweaked—inside some other computer architecture in a computer architecture simulator; or inside a FPGA as a soft microprocessor; or both—before committing to the final hardware form.[12]

Subcategories

The discipline of computer architecture has three main subcategories:[13]

1. *Instruction Set Architecture*, or ISA. The ISA defines the machine code that a processor reads and acts upon as well as the word size, memory address modes, processor registers, and data type.
2. *Microarchitecture*, or *computer organization* describes how a particular processor will implement the ISA.[14] The size of a computer's CPU cache for instance, is an issue that generally has nothing to do with the ISA.
3. *System Design* includes all of the other hardware components within a computing system. These include:
    1. Data processing other than the CPU, such as direct memory access (DMA)
    2. Other issues such as virtualization, multiprocessing, and software features.

There are other types of computer architecture. The following types are used in bigger companies like Intel, and count for 1% of all of computer architecture

- Macroarchitecture: architectural layers more abstract than microarchitecture
- Assembly Instruction Set Architecture (ISA): A smart assembler may convert an abstract assembly language common to a group of machines into slightly different machine language for different implementations
- Programmer Visible Macroarchitecture: higher level language tools such as compilers may define a consistent interface or contract to programmers using them, abstracting differences between underlying ISA, UISA, and microarchitectures. E.g. the C, C++, or Java standards define different Programmer Visible Macroarchitecture.
- UISA (Microcode Instruction Set Architecture)—a group of machines with different hardware level microarchitectures may share a common microcode architecture, and hence a UISA.
- Pin Architecture: The hardware functions that a microprocessor should provide to a hardware platform, e.g., the x86 pins A20M, FERR/IGNNE or FLUSH. Also, messages that the processor should emit so that external caches can be invalidated (emptied). Pin architecture functions are more flexible than ISA functions because external hardware can adapt to new encodings, or change from a pin to a message. The term "architecture" fits, because the functions must be provided for compatible systems, even if the detailed method changes.

Roles

Definition

The purpose is to design a computer that maximizes performance while keeping power consumption in check, costs low relative to the amount of expected performance, and is also very reliable. For this, many aspects are to be considered which includes instruction set design, functional organization, logic design, and implementation. The implementation involves integrated circuit design, packaging, power, and cooling. Optimization of the design requires familiarity with compilers, operating systems to logic design, and packaging.[15]

Instruction set architecture

An instruction set architecture (ISA) is the interface between the computer's software and hardware and also can be viewed as the programmer's view of the machine. Computers do not understand high-level programming languages such as Java, C++, or most programming languages used. A processor only understands instructions encoded in some numerical fashion, usually as binary numbers. Software tools, such as compilers, translate those high level languages into instructions that the processor can understand.

Besides instructions, the ISA defines items in the computer that are available to a program—e.g. data types, registers, addressing modes, and memory. Instructions locate these available items with register indexes (or names) and memory addressing modes.

The ISA of a computer is usually described in a small instruction manual, which describes how the instructions are encoded. Also, it may define short (vaguely) mnemonic names for the instructions. The names can be recognized by a software development tool called an assembler. An assembler is a computer program that translates a human-readable form of the ISA into a computer-readable form. Disassemblers are also widely available, usually in debuggers and software programs to isolate and correct malfunctions in binary computer programs.

ISAs vary in quality and completeness. A good ISA compromises between programmer convenience (how easy the code is to understand), size of the code (how much code is required to do a specific action), cost of the computer to interpret the instructions (more complexity means more hardware needed to decode and execute the instructions), and speed of the computer (with more complex decoding hardware comes longer decode time). Memory organization defines how instructions interact with the memory, and how memory interacts with itself.

During design emulation software (emulators) can run programs written in a proposed instruction set. Modern emulators can measure size, cost, and speed to determine if a particular ISA is meeting its goals.

Computer organization

Computer organization helps optimize performance-based products. For example, software engineers need to know the processing power of processors. They may need to optimize software in order to gain the most performance for the lowest price. This can require quite detailed analysis of the computer's organization. For example, in a SD card, the designers might need to arrange the card so that the most data can be processed in the fastest possible way.

Computer organization also helps plan the selection of a processor for a particular project. Multimedia projects may need very rapid data access, while virtual machines may need fast interrupts. Sometimes certain tasks need additional components as well. For example, a computer capable of running a virtual machine needs virtual memory hardware so that the memory of different virtual computers can be kept separated. Computer organization and features also affect power consumption and processor cost.

Implementation

Once an instruction set and micro-architecture are designed, a practical machine must be developed. This design process is called the *implementation*. Implementation is usually not considered architectural design, but rather hardware design engineering. Implementation can be further broken down into several steps:

- **Logic Implementation** designs the circuits required at a logic gate level
- **Circuit Implementation** does transistor-level designs of basic elements (gates, multiplexers, latches etc.) as well as of some larger blocks (ALUs, caches etc.) that may be implemented at the log gate level, or even at the physical level if the design calls for it.

- **Physical Implementation** draws physical circuits. The different circuit components are placed in a chip floorplan or on a board and the wires connecting them are created.
- **Design Validation** tests the computer as a whole to see if it works in all situations and all timings. Once the design validation process starts, the design at the logic level are tested using logic emulators. However, this is usually too slow to run realistic test. So, after making corrections based on the first test, prototypes are constructed using Field-Programmable Gate-Arrays (FPGAs). Most hobby projects stop at this stage. The final step is to test prototype integrated circuits. Integrated circuits may require several redesigns to fix problems.

For CPUs, the entire implementation process is organized differently and is often referred to as CPU design.

Design goals

The exact form of a computer system depends on the constraints and goals. Computer architectures usually trade off standards, power versus performance, cost, memory capacity, latency (latency is the amount of time that it takes for information from one node to travel to the source) and throughput. Sometimes other considerations, such as features, size, weight, reliability, and expandability are also factors.

The most common scheme does an in depth power analysis and figures out how to keep power consumption low, while maintaining adequate performance.

Performance

Modern computer performance is often described in IPC (instructions per cycle). This measures the efficiency of the architecture at any clock frequency. Since a faster rate can make a faster computer, this is a useful measurement. Older computers had IPC counts as low as 0.1 instructions per cycle. Simple modern processors easily reach near 1. Superscalar processors may reach three to five IPC by executing several instructions per clock cycle.

Counting machine language instructions would be misleading because they can do varying amounts of work in different ISAs. The "instruction" in the standard measurements is not a count of the ISA's actual machine language instructions, but a unit of measurement, usually based on the speed of the VAX computer architecture.

Many people used to measure a computer's speed by the clock rate (usually in MHz or GHz). This refers to the cycles per second of the main clock of the CPU. However, this metric is somewhat misleading, as a machine with a higher clock rate may not necessarily have greater performance. As a result, manufacturers have moved away from clock speed as a measure of performance.

Other factors influence speed, such as the mix of functional units, bus speeds, available memory, and the type and order of instructions in the programs.

There are two main types of speed: latency and throughput. Latency is the time between the start of a process and its completion. Throughput is the amount of work done per unit time. Interrupt latency is the guaranteed maximum response time of the system to an electronic event (like when the disk drive finishes moving some data).

Performance is affected by a very wide range of design choices — for example, pipelining a processor usually makes latency worse, but makes throughput better. Computers that control machinery usually need low interrupt latencies. These computers operate in a real-time environment and fail if an operation is not completed in a specified amount of time. For example, computer-controlled anti-lock brakes must begin braking within a predictable, short time after the brake pedal is sensed or else failure of the brake will occur

Benchmarking takes all these factors into account by measuring the time a computer takes to run through a series of test programs. Although benchmarking shows strengths, it shouldn't be how you choose a computer. Often the measured machines split on different measures. For example, one system might handle scientific applications quickly, while another might render video games more smoothly. Furthermore, designers may target and add special features to their products, through hardware or software, that permit a specific benchmark to execute quickly but don't offer similar advantages to general tasks.

Power efficiency

Power efficiency is another important measurement in modern computers. A higher power efficiency can often be traded for lower speed or higher cost. The typical measurement when referring to power consumption in computer architecture is MIPS/W (millions of instructions per second per watt).

Modern circuits have less power required per transistor as the number of transistors per chip grows.[16] This is because each transistor that is put in a new chip requires its own power supply and requires new pathways to be built to power it. However the number of transistors per chip is starting to increase at a slower rate. Therefore, power efficiency is starting to become as important, if not more important than fitting more and more transistors into a single chip. Recent processor designs have shown this emphasis as they put more focus on power efficiency rather than cramming as many transistors into a single chip as possible.[17] In the world of embedded computers, power eficiency has long been an important goal next to throughput and latency.

Shifts in market demand

Increases in publicly released refresh rates have grown slowly over the past few years, with respect to vast leaps in power consumption reduction and miniaturization demand. This has led to a new demand for longer battery life and reductions in size due t the mobile technology being produced at a greater rate. This change in focus from greater refresh rates to power consumption and miniaturization can be shown by the significant reductions in power consumption, as much as 50%, that were reported by Intel in their release of the Haswell microarchitecture; where they dropped their power consumption benchmark from 30-40 watts down to 10-20 watts.[18] Comparing this to the processing speed increase of 3 GHz to 4 GHz (2002 to 2006)[19] it can be seen that the focus in research and development are shifting away from refresh rates and moving towards consuming less power and taking up less space.

See also

- Comparison of CPU architectures
- Computer hardware
- CPU design
- Floating point
- Von Neumann
- Harvard (Modified)
- Dataflow
- TTA
- Reconfigurable computing
- Influence of the IBM PC on the personal computer market
- Orthogonal instruction set
- Software architecture
- von Neumann architecture
- Flynn's taxonomy

References

1. Clements, Alan. *Principles of Computer Hardware* (Fourth ed.). p. 1. "Architecture describes the internal organization of a computer in an abstract way; that is, it defines the capabilities of the computer and its programming model. You can have two computers that have been constructed in different ways with different technologies but with the same architecture."
2. Hennessy, John; Patterson, David. *Computer Architecture: A Quantitative Approach* (Fifth ed.). p. 11. "This task has many aspects, including instruction set design, functional organization, logic design, and implementation."
3. "Electronic Digital Computers" (http://www.computer50.org/kgill/mark1/natletter.html), *Nature*, **162**: 487, 25 September 1948, doi:10.1038/162487a0 (https://doi.org/10.1038%2F162487a0), retrieved 2009-04-10
4. Susanne Faber, "Konrad Zuses Bemuehungen um die Patentanmeldung der Z3", 2000
5. Neumann, John (1945). *First Draft of a Report on the EDVAC*. p. 9.

6. Reproduced in B. J. Copeland (Ed.), "Alan Turing's Automatic Computing Engine", OUP, 2005, pp. 369-454.
7. Johnson, Lyle (1960). "A Description of Stretch" (http://archive.computerhistory.org/resources/text/IBM/Stretch/pdfs/05-10/102634114.pdf) (PDF). p. 1. Retrieved 7 October 2017.
8. Buchholz, Werner (1962). *Planning a Computer System*. p. 5.
9. "System 360, From Computers to Computer Systems" (http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/system360/). *IBM100*. Retrieved 11 May 2017.
10. Hellige, Hans Dieter (2004). "Die Genese von Wissenschaftskonzeptionen der Computerarchitektur: Vom "system of organs" zum Schichtmodell des Designraums". *Geschichten der Informatik: Visionen, Paradigmen, Leitmotive*. pp. 411–472.
11. ACE underwent seven paper designs in one year, before a prototype was initiated in 1948. [B. J. Copeland (Ed.), "Alan Turing's Automatic Computing Engine", OUP, 2005, p. 57]
12. Schmalz. "Organization of Computer Systems" (https://www.cise.ufl.edu/~mssz/CompOrg/CDAintro.html). *UF CISE*. Retrieved 11 May 2017.
13. John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach* (Third ed.). Morgan Kaufmann Publishers.
14. Laplante, Phillip A. (2001). *Dictionary of Computer Science, Engineering, and Technology*. CRC Press. pp. 94–95. ISBN 0-8493-2691-5.
15. Martin, Milo. "What is computer architecture?" (https://www.cis.upenn.edu/~milom/cis501-Fall11/lectures/00_intro.pdf) (PDF). *UPENN*. Retrieved 11 May 2017.
16. "Integrated circuits and fabrication" (http://eacharya.inflibnet.ac.in/data-server/eacharya-documents/53e0c6cbe413016f23443704_INFIEP_33/192/ET/33-192-ET-V1-S1__ssed_unit_4_module_10_integrated_circuits_and_fabrication_e-text.pdf) (PDF). Retrieved 8 May 2017.
17. "Exynos 9 Series (8895)" (http://www.samsung.com/semiconductor/minisite/Exynos/w/solution/mod_ap/8895/?CID=AFL-hq-mul-0813-11000170). *Samsung*. Retrieved 8 May 2017.
18. "Measuring Processor Power TDP vs ACP" (http://www.intel.com/content/dam/doc/whitepaper/resources-xeon-measuring-processor-power-paper.pdf) (PDF). *Intel*. April 2011. Retrieved 5 May 2017.
19. "History of Processor Performance" (http://www.cs.columbia.edu/~sedwards/classes/2012/3827-spring/advanced-arch-2011.pdf) (PDF). *cs.columbia.edu*. 24 April 2012. Retrieved 5 May 2017.

Sources

- John L. Hennessy and David Patterson (2006). *Computer Architecture: A Quantitative Approach* (Fourth ed.). Morgan Kaufmann. ISBN 978-0-12-370490-0.
- Barton, Robert S., "Functional Design of Computers", *Communications of the ACM* 4(9): 405 (1961).
- Barton, Robert S., "A New Approach to the Functional Design of a Digital Computer", *Proceedings of the Western Joint Computer Conference*, May 1961, pp. 393–396. About the design of the Burroughs B5000 computer.
- Bell, C. Gordon; and Newell, Allen (1971). "Computer Structures: Readings and Examples", McGraw-Hill.
- Blaauw, G.A., and Brooks, F.P., Jr., "The Structure of System/360, Part I-Outline of the Logical Structure", *IBM Systems Journal*, vol. 3, no. 2, pp. 119–135, 1964.
- Tanenbaum, Andrew S. (1979). *Structured Computer Organization*. Englewood Cliffs, New Jersey: Prentice-Hall. ISBN 0-13-148521-0.

External links

- ISCA: Proceedings of the International Symposium on Computer Architecture
- Micro: IEEE/ACM International Symposium on Microarchitecture
- HPCA: International Symposium on High Performance Computer Architecture
- ASPLOS: International Conference on Architectural Support for Programming Languages and Operating Systems
- ACM Transactions on Architecture and Code Optimization
- IEEE Transactions on Computers
- The von Neumann Architecture of Computer Systems