Foundations of Computer Science Architecture

Lorand Ștefan Szasz
Department of Computer Science
West University Timişoara
`lorand.szasz06@e-uvt.ro`

Abstract

This paper provides an in-depth survey of computer architecture, tracing its evolution from fundamental digital logic to advanced parallel and heterogeneous systems. We cover Boolean algebra, combinational and sequential circuits, instruction set architectures, pipeline and superscalar designs, memory hierarchies from registers to non-volatile storage, cache coherence, interconnection networks, multicore and manycore processors, GPUs, domain-specific accelerators, power and reliability considerations, and emerging paradigms such as neuromorphic and quantum architectures. The goal is a self-contained reference for advanced undergraduates and beginning graduate students.

Contents

1 Introduction

Computer architecture sits at the nexus of hardware and software, defining the structural blueprint by which programs execute atop silicon. It encompasses microarchitecture (the implementation of an ISA), system design (bus and I/O organization), and performance evaluation. Over decades, architectural innovation has enabled exponential growth in computing capabilities—Moore's Law scaling, Dennard scaling, and now the shift to parallelism and specialization. A deep understanding of architectural principles underpins future advances in performance-per-watt, security, and domain-specific computing.

2 Digital Logic

2.1 Boolean Algebra and Logic Gates

At the heart of any digital system lies Boolean algebra. Variables taking values 0/1 combine via logical operations: AND ($\wedge$), OR ($\vee$), NOT ($\neg$), NAND, NOR, XOR. Gate-level minimization techniques—Karnaugh maps and the Quine–McCluskey algorithm—reduce transistor count, critical for cost and power. CMOS transistor arrangements realize gates: pull-up networks (pMOS) and pull-down networks (nMOS) implement logic functions with static power advantages.

2.2 Combinational Circuits

Complex combinational blocks arise by connecting logic gates without memory elements. Ripple-carry adders, carry-lookahead adders, multipliers (array, Wallace tree), multiplexers, decoders, and arithmetic-logic units (ALUs) form the computational backbone. Performance hinges on gate delays and fan-out—factors that dictate critical paths and maximum clock frequency.

2.3 Sequential Circuits and FSMs

Sequential logic introduces state via latches and flip-flops. Edge-triggered D flip-flops synchronize data on clock transitions; master–slave configurations avoid race conditions. Finite state machines (FSMs) model control logic: Moore machines (output depends on state) versus Mealy machines (output depends on state and input). State minimization and encoding strategies (one-hot, binary, Gray code) balance simplicity against resource usage.

3 Processor Architectures

3.1 Instruction Set Architectures (ISA)

An ISA defines the programmer-visible contract: opcodes, register files, addressing modes, data types, and memory consistency semantics. RISC architectures (e.g. MIPS, ARM) favor fixed-length, load–store pipelines; CISC architectures (x86) integrate complex addressing and variable-length encodings. RISC-V's open ISA highlights extensibility and modular design.

3.2 Microarchitecture: Datapath and Control

The microarchitecture realizes the ISA via a datapath—ALUs, register files, multiplexers, and memory ports—and a control unit. Hardwired control uses fixed combinational logic for each opcode; microprogrammed control sequences micro-operations stored in a control store, easing ISA changes at cost of cycle overhead.

3.3 Pipelining and Hazard Mitigation

Pipelining overlaps instruction stages: fetch (IF), decode/register read (ID), execute (EX), memory access (MEM), write-back (WB). Hazards arise when dependencies break stage independence:

- Data hazards: RAW, WAR, WAW prevented via forwarding, pipeline stalls, and scoreboarding.
- Control hazards: branch mispredictions resolved with static (fixed not-taken/taken) or dynamic predictors (two-bit saturating counters, global history tables) and speculative execution.
- Structural hazards: multiple units contend for resources—resolved by duplication or scheduling.

3.4 Superscalar and Out-of-Order Execution

Superscalar cores issue multiple instructions per cycle. Tomasulo's algorithm enables register renaming and reservation stations to allow out-of-order execution, improving utilization of functional units. Reorder buffers ensure in-order retirement, preserving precise exceptions.

3.5 Vector and SIMD Extensions

Vector architectures (Cray, AVX-512) and SIMD units accelerate data-level parallelism. Strided memory access, alignment, and scatter/gather support affect performance. Compiler auto-vectorization and intrinsics expose these units to software.

4 Memory Hierarchies

4.1 Cache Architectures

Multi-level caches exploit locality:

- L1 caches: small, fastest (e.g. 32 KiB, 4–8 cycles)
- L2/L3 caches: larger, slower (hundreds of KiB to tens of MiB)

Cache parameters—block size, associativity, replacement (LRU, pseudo-LRU), write policy (write-back vs. write-through)—impact hit rates and bandwidth. Non-uniform cache architectures (NUCA) tune access times within large shared caches.

4.2 Virtual Memory and TLBs

Virtual memory provides protection and address space abstraction. Page tables map virtual to physical pages; multi-level and hashed page tables manage large address spaces. Translation lookaside buffers (TLBs) cache recent mappings; TLB misses incur page walking overhead. Page coloring optimizes cache–memory interaction.

4.3 DRAM and Beyond

DRAM remains main memory: DDR standards increase throughput but face latency and energy bottlenecks. Emerging technologies—HBM, 3D-stacked memory, non-volatile memories (PCM, ReRAM)—promise higher density and persistence. Memory controllers schedule reads/writes (FR-FCFS, reordering) for efficiency.

4.4 Storage and I/O

Secondary storage—NAND-based SSDs, magnetic disks—presents orders-of-magnitude higher latency. RAID levels, wear leveling, and FTLs (flash translation layers) manage reliability and performance. Direct memory access (DMA), NVMe, and PCI Express interconnects deliver high I/O throughput. Remote Direct Memory Access (RDMA) and NVLink enable low-latency data transfer in clusters.

5 Parallel and Distributed Architectures

5.1 Flynn's Taxonomy

Flynn's classification:

- SISD: traditional uniprocessor
- SIMD: vector units and GPUs
- MISD: seldom used (fault-tolerant streams)
- MIMD: multicore CPUs, clusters

5.2 Shared-Memory Multiprocessors

Cache coherence protocols (MESI, MOESI, Dragon) maintain consistency. Directory-based schemes scale beyond small CMPs. Memory consistency models (SC, TSO, Release Consistency) define allowed reorderings—crucial for correctness and performance.

5.3 Distributed Memory and Message Passing

Clusters interconnect via Ethernet or specialized fabrics (InfiniBand). MPI libraries provide explicit communication primitives; PGAS languages (UPC, Chapel) offer global address spaces. Partitioning, topology-aware mapping, and collective algorithms optimize performance.

5.4 Interconnection Networks

NoCs and cluster networks adopt topologies—bus, ring, mesh/torus, hypercube, dragonfly. Routing algorithms (deterministic, adaptive) and flow control (wormhole, virtual cut-through) dictate latency and throughput. Network-on-Chip integrates routers and links on die to connect hundreds of cores.

5.5 Accelerator Architectures

GPUs feature thousands of lightweight threads with SIMT execution. Tensor cores and systolic arrays in TPUs and ML accelerators provide matrix-multiply throughput. FPGAs and coarse-grain reconfigurable fabrics allow custom pipelines for domain-specific tasks.

6 Power, Thermal, and Reliability

6.1 Power Modeling and Optimization

Dynamic voltage/frequency scaling (DVFS) and dynamic power management reduce active power. Clock gating and power gating minimize leakage. Heterogeneous designs (big.LITTLE, ARM DynamIQ) match cores to workload demands for energy efficiency.

6.2 Thermal Management

Thermal hotspots impair reliability. Thermal sensors and adaptive clock throttling prevent overheating. Floorplanning and heat-spreader designs aid uniform dissipation.

6.3 Fault Tolerance

Soft errors (SEUs) in memory and logic necessitate ECC, parity checks, and duplicated execution (DMR, TMR). Checkpoint/restart mechanisms enable recovery from detected faults. Wear-out in non-volatile memories demands monitoring and remapping.

7 Emerging and Future Paradigms

7.1 Neuromorphic Computing

Event-driven, spike-based architectures (Loihi, TrueNorth) model neural networks with low power. Memristive crossbars implement synaptic weights for both training and inference.

7.2 Quantum Architectures

Quantum processors use qubits, superposition, and entanglement. Architectures vary (superconducting, ion-trap), requiring cryogenic control electronics, quantum error correction, and qubit coherence management.

7.3 Optical and Photonic Interconnects

Photonic links on-chip and between chips promise high bandwidth and low latency with reduced energy per bit. Wavelength-division multiplexing (WDM) enables parallel channels over single waveguides.

7.4 3D Integration and Heterogeneous Systems

3D-stacked dies integrate logic, memory, and sensors vertically. Through-silicon vias (TSVs) and interposer technologies enable heterogeneous integration of CPU, GPU, FPGA, and memory in a single package.

8 Conclusion

Computer architecture has grown into a rich discipline balancing performance, efficiency, scalability, and reliability. From transistor-level logic to global-scale datacenter fabrics, each layer interacts with adjacent layers in complex trade-off spaces. As we confront the end of traditional scaling, domain-specific and heterogeneous architectures guided by deep architectural insight will drive the next computing revolution.

References

[1] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach*, 5th ed. Morgan Kaufmann, 2011.

[2] William Stallings. *Computer Organization and Architecture: Designing for Performance*, 10th ed. Pearson, 2016.

[3] David A. Patterson and John L. Hennessy. *Computer Organization and Design RISC-V Edition*, 2nd ed. Morgan Kaufmann, 2017.

[4] M. Müller-Olm, A. Tanter, and F. Børder. *Digital Memory: Fundamentals and Emerging Technologies.* Springer, 2004.

[5] Proceedings of the 53rd Annual IEEE/ACM International Symposium on Microarchitecture, 2020.