# 1. Fundamentals

## 1.1 MOS Transistors

**Def. 1. nMOS**: ON when Gate=1, OFF when Gate=0. Passes strong 0, weak 1. **pMOS**: ON when Gate=0, OFF when Gate=1. Passes strong 1, weak 0.

**Def. 2. CMOS Logic**: Two complementary networks:

- **Pull-Up (PUN)**: pMOS connects output to $V_{DD}$ (1).
- **Pull-Down (PDN)**: nMOS connects output to GND (0).

Output is always strongly driven; no significant static power draw.

## 1.2 Logic Gates

**Def. 3. NOT**: $Y = \overline{A}$ (2 transistors). **NAND**: $Y = \overline{A \cdot B}$ (4T: 2 pMOS parallel, 2 nMOS series). **NOR**: $Y = \overline{A + B}$ (4T: 2 pMOS series, 2 nMOS parallel). AND/OR are NAND/NOR + inverter (inverting gates are natural in CMOS).

**Rem. 4. XOR**: $Y = A \oplus B$ (1 if different). **XNOR**: $Y = \overline{A \oplus B}$ (1 if same). **Buffer**: $Y = A$ (two NOTs in series, for amplification/delay).

## 1.3 Boolean Algebra

**Def. 5.** Key rules: Identity ($A \cdot 1 = A$, $A + 0 = A$), Complement ($A \cdot \overline{A} = 0$, $A + \overline{A} = 1$), Distributive ($A(B + C) = AB + AC$).

**Thm. 6. DeMorgan's**: $\overline{A \cdot B} = \overline{A} + \overline{B}$ and $\overline{A + B} = \overline{A} \cdot \overline{B}$.

**Imp. 7.** A gate set is **logically complete** if any Boolean function can be built from it. Complete sets: {AND, OR, NOT}, {NAND}, {NOR}.

**Def. 8. Minterm**: Product term with all variables. **SOP**: OR of minterms where output=1. **K-Map**: Group $2^n$ adjacent 1s to eliminate changing variables.

## 1.4 Combinational Logic

**Def. 9.** Output is a pure function of current inputs (no memory/state).

**Def. 10. Decoder** ($n$-to-$2^n$): Sets exactly one of $2^n$ outputs to 1 based on $n$-bit input. **Multiplexer**: $\log_2 N$ select lines choose which of $N$ data inputs reaches output. Can implement any logic function as a LUT.

**Def. 11. Full Adder**: $S = A \oplus B \oplus C_{in}$, $C_{out} = AB + C_{in}(A \oplus B)$. **Ripple Carry Adder**: Chain $n$ full adders; critical path is carry ripple from LSB to MSB.

**Lem. 12. Tri-state buffer**: Output can be 0, 1, or High-Z (disconnected). Enables shared buses. **PLA**: Programmable AND-plane + OR-plane = physical SOP implementation.

## 1.5 Sequential Logic

**Def. 13.** Output depends on current inputs **and past state**. Uses storage elements (feedback loops).

**Def. 14. R-S Latch**: Cross-coupled NANDs. Set ($S$=0) → $Q$=1, Reset ($R$=0) → $Q$=0. $S$=$R$=0 is **forbidden** (metastability). **D Latch**: Adds Write Enable; when WE=1, $Q$ follows $D$. **Transparent** while enabled.

**Imp. 15. D Flip-Flop**: Master-slave pair of D latches. **Edge-triggered**: captures $D$ only at rising clock edge. Solves transparency problem for synchronous design.

### 1.5.1 Timing and Clock

**Def. 16. Asynchronous**: Transitions on input change (no coordination). **Synchronous**: Transitions on clock edges (predictable, global). **Clock**: Periodic 0/1 signal; period must cover worst-case combinational delay.

### 1.5.2 Finite State Machines

**Def. 17. FSM**: States + transitions + inputs + outputs. **Moore**: Output depends only on state. **Mealy**: Output depends on state **and** inputs.

**Rem. 18.** FSM implementation: (1) State register (sequential), (2) Next-state logic (combinational), (3) Output logic (combinational). State encoding: Binary (min bits), One-Hot (one bit/state), Output (use outputs as state bits).

### 1.5.3 Memory and Registers

**Def. 19. Register**: Group of D flip-flops with common clock/WE. **Memory**: Array indexed by address. Address space = $2^n$ locations. Uses decoder + MUX.

## 1.6 Timing

### 1.6.1 Combinational Timing

**Def. 20. Propagation delay ($t_{pd}$)**: Max time from input change to output stable (longest path). **Contamination delay ($t_{cd}$)**: Min time from input change to output starts changing (shortest path).

**Rem. 21. Glitch**: Multiple output transitions from a single input change due to different path delays. Fixed by adding consensus terms in K-maps.

### 1.6.2 Sequential Timing

**Def. 22. Aperture time** $t_a = t_{setup} + t_{hold}$: $t_{setup}$: Data must be stable **before** clock edge. $t_{hold}$: Data must stay stable **after** clock edge. Violation → **metastability** (output stuck between 0/1).

**Def. 23.** $t_{pcq}$: Max time after clock edge until $Q$ stable. $t_{ccq}$: Min time after clock edge until $Q$ starts changing.

**Thm. 24. Setup constraint**: $T_c \geq t_{pcq} + t_{pd} + t_{setup}$ (determines $f_{max}$). **Hold constraint**: $t_{ccq} + t_{cd} > t_{hold}$ (cannot fix by slowing clock).

**Imp. 25. Clock skew** ($t_{skew}$) worsens both: Setup: $T_c \geq t_{pcq} + t_{pd} + t_{setup} + t_{skew}$. Hold: $t_{ccq} + t_{cd} > t_{hold} + t_{skew}$.

# 2. Processor Design

## 2.1 Von-Neumann Architecture

**Def. 26. Von Neumann Model**: Memory (stores program + data), Processing Unit (ALU + registers), Control Unit (PC + IR), Input, Output.

**Imp. 27. Stored Program**: Instructions and data share a linear memory. **Sequential Processing**: Instructions executed one at a time unless control flow changes PC.

**Rem. 28.** Instruction cycle: (1) Fetch → (2) Decode → (3) Evaluate Address → (4) Fetch Operands → (5) Execute → (6) Store Result.

**Def. 29. Address Space**: Total unique locations ($2^n$ for $n$ address bits). **Addressability**: Bits per address (byte=8, word=32). **Endianness**: Big (MSB at lowest addr) vs. Little (LSB at lowest addr).

## 2.2 Instruction Set Architecture (ISA)

**Def. 30. ISA**: Contract between software and hardware. Specifies memory organization, register set, instruction set, opcodes, data types, and addressing modes.

**Lem. 31. CISC**: Small semantic gap; complex instructions (x86). **RISC**: Large semantic gap; simple primitive instructions (MIPS, LC-3).

**Def. 32. Addressing Modes**: Immediate (constant in instr.), Register, PC-Relative ($PC + offset$), Indirect (pointer to address), Base+Offset ($Reg + offset$).

## 2.3 Assembly

**Imp. 33.** In RISC: operands must be in registers before processing. **Load (lw)**: Memory → Register. **Store (sw)**: Register → Memory. **Conditional branches**: Based on condition codes (LC-3) or register comparison (MIPS). **Unconditional jumps**: Always redirect PC.

## 2.4 Microarchitecture

**Def. 34. Microarchitecture**: The specific hardware implementation of an ISA. One ISA can have many different microarchitectures (single-cycle, multi-cycle, OoO). Software is unaware of the implementation.

**Rem. 35.** Design metrics: Performance (execution time, throughput), Cost (die area), Power/Energy, Reliability.

### 2.4.1 Single-Cycle

**Def. 36.** Every instruction completes in exactly **one clock cycle**. CPI = 1. Clock period = delay of **slowest** instruction (typically `lw`: fetch → decode → ALU → mem read → write-back). Fast instructions waste cycle time.

### 2.4.2 Multi-Cycle

**Def. 37.** Breaks execution into **multiple shorter cycles**. Variable CPI per instruction (e.g., `add`=3, `lw`=5). Enables functional unit reuse (one ALU for PC increment and arithmetic).

**Thm. 38.**

$$\text{Time} = \text{Instr. Count} \times \text{Avg. CPI} \times T_c$$

Higher CPI than single-cycle, but shorter $T_c$ (only slowest *stage*, not entire instruction).

**Rem. 39.** Control unit is an FSM tracking current step. Microarchitectural registers (IR, MDR, ALUOut) store intermediate results between cycles. **Sequencing overhead**: register setup/clk-to-Q adds per-cycle penalty; too many cycles can negate benefits.

# 3. Instruction Level Parallelism

## 3.1 Pipelining

**Def. 40. Pipelining**: Overlap multiple instructions in execution (assembly line). Increases **throughput**, not single-instruction latency. Ideal $k$-stage pipeline → $k$-fold speedup.

**Def. 41. 5-Stage RISC Pipeline**: (1) IF: Fetch instruction, update PC. (2) ID: Decode, read registers. (3) EX: ALU / address calc. (4) MEM: Data memory access. (5) WB: Write result to register file. Pipeline registers (IF/ID, ID/EX, EX/MEM, MEM/WB) carry data between stages.

### 3.1.1 Pipeline Hazards

**Def. 42. Structural**: Hardware resource conflict. **Data (RAW/WAR/WAW)**: Instruction depends on incomplete prior result. Only **RAW** (flow dependence) needs HW intervention in 5-stage in-order. **Control**: Branch target unknown; wrong instructions may be fetched.

**Lem. 43. Forwarding (Bypassing)**: Route result from EX/MEM stage directly to ALU input, bypassing register file. Resolves most RAW hazards.

**Imp. 44. Load-Use Hazard**: Forwarding cannot resolve (data available only after MEM). Requires a **1-cycle stall**: disable PC and IF/ID update, flush ID/EX (insert bubble/NOP).

**Rem. 45. Branch handling**: Static prediction (always not-taken), Dynamic prediction (Branch History Table), Early resolution (move comparison to Decode stage).

### 3.1.2 Precise Exceptions

**Def. 46. Exceptions**: Internal problems. **Interrupts**: External events. Both require: retire all prior instructions, flush younger ones, save PC+state, redirect to handler.

**Def. 47. Reorder Buffer (ROB)**: Stores results as instructions complete OoO. Retires **in-order**, checking for exceptions at retirement. Uses valid bits in registers pointing to ROB entries.

## 3.2 Out-of-Order Execution

**Def. 48. OoO**: Execute instructions based on operand availability (dataflow), not program order. Fetched/decoded in-order → executed OoO → retired in-order (via ROB).

**Rem. 49.** OoO tolerates long latencies by executing independent instructions while dependent ones wait. Improves IPC at cost of complexity and power.

### 3.2.1 OoO Mechanisms

**Def. 50. Register Renaming**: Eliminates false dependencies (WAW, WAR) by mapping architectural registers to physical storage. **Reservation Stations (RS)**: Buffers holding instructions waiting for operands. **Tag Broadcast**: Results broadcast via bus; RS entries with matching tags capture values.

**Def. 51. Tomasulo's Algorithm**: Decode → allocate RS → rename via Register Alias Table (RAT) → if operands ready: execute + broadcast; else: watch bus for tags. Instruction window size limits scalability.

### 3.2.2 Memory Dependencies

**Def. 52. Memory Disambiguation**: Determine if load/store depends on prior load/store. **Conservative**: Stall all (safe, slow). **Aggressive**: Assume independent (fast, may need recovery). **Intelligent**: Predict dependencies.

**Def. 53. Load/Store Queue (LSQ)**: Tracks uncommitted memory ops. **Store-to-Load Forwarding**: Search LSQ for matching address, copy data directly to load.

## 3.3 Superscalar Execution

**Def. 54.** $N$**-Superscalar**: Issue $N$ instructions per cycle. Orthogonal to OoO (can be in-order superscalar or OoO superscalar). Main challenge: finding $N$ independent instructions.

## 3.4 Branch Prediction

**Rem. 55.** Mispredictions drastically reduce IPC in deep pipelines. Modern processors target 95%+ accuracy.

### 3.4.1 Static Prediction

**Def. 56. Always Taken / Not-Taken**: Simple, low accuracy. **BTFN**: Backward Taken, Forward Not-Taken (exploits loops). **Profile/Program/Programmer-based**: Compile-time heuristics or hints (`likely()`/`unlikely()`).

### 3.4.2 Dynamic Prediction

**Def. 57. Branch Target Buffer (BTB)**: Cache of recent branch target addresses. **Return Address Stack (RAS)**: Push on call, pop on ret (>95% accuracy with 8 entries).

**Def. 58. Last-Time**: Predict same as last outcome (1-bit). Good for long loops, poor for short/alternating. **Two-Bit Counter (Bimodal)**: Changes prediction only after 2 consecutive mispredictions (hysteresis). ∼85–90% accuracy.

**Def. 59. Two-Level Prediction**: Use history to correlate outcomes. **GHR** (Global History Register): Recent outcomes of different branches. **LHR** (Local History Register): Past outcomes of same branch. **Gshare**: XOR GHR with PC to index Pattern History Table (reduces interference).

**Imp. 60. Tournament Predictor**: Multiple predictors run in parallel; meta-predictor selects the most accurate one. **Biased branch filtering**: Route heavily biased branches to simple predictors, save complex predictor capacity.

### 3.4.3 Advanced Predictors

**Def. 61. Perceptron**: Single-layer neural net (adder tree + sign). 95%+ accuracy, but only learns linearly separable functions. **TAGE**: Multiple tables with geometric history lengths. State-of-the-art (97%+).

### 3.4.4 Alternative Control Flow

**Def. 62. Delayed Branching**: $N$ delay-slot instructions execute unconditionally. Keeps pipeline full but ties ISA to pipeline depth. **Predicated Execution (CMOV)**: Execute both paths, commit only if predicate true. Eliminates mispredictions but wastes energy.

# 4. Advanced Architectures

## 4.1 VLIW Architectures

**Def. 63. VLIW**: Multiple independent operations packed into one long instruction word, executed by multiple PEs. **Compiler** ensures independence (no HW dependency checking). Reduces portability (binary tied to HW).

**Def. 64. Trace Scheduling**: Find most frequent path, schedule to maximize parallelism. **Superblocks**: Single entry, multiple exits via tail duplication.

## 4.2 SIMD Architectures

**Def. 65. SIMD**: Same instruction on multiple data. **Array Processor**: Multiple PEs simultaneously. **Vector Processor**: Same PE, consecutive time steps (pipelined). Needs regular data and high memory bandwidth.

**Def. 66. Vector Registers**: Store multiple elements. **Stride**: Distance between consecutive addresses. **Memory Banks**: Concurrent access to independent addresses.

## 4.3 Systolic Arrays

**Def. 67.** Grid of PEs with rhythmic data flow. Each PE processes and passes to neighbors. Avoids reloading; efficient for regular computations (matrix multiply, convolution). Not a standard pipeline: non-linear, multi-dimensional flow.

## 4.4 GPU Architectures

### 4.4.1 SIMT

**Def. 68. SIMT**: One instruction issued to thread group, each with own data/registers/PC. Threads can diverge (HW serializes branches, worst case 1/32 throughput).

### 4.4.2 GPU Organization

**Def. 69. SM**: CUDA cores + registers + shared memory + warp schedulers. **Warp**: 32 threads in lock-step. **Thread Block**: Warps on same SM sharing shared memory. **Grid**: All thread blocks for a kernel.

**Thm. 70. Warp advantages**: Amortized control, latency hiding, memory coalescing. **Warp disadvantages**: Divergence, uncoalesced access, register pressure, tail effects.

### 4.4.3 Latency Hiding

**Def. 71. FGMT**: Switch warps every cycle. To hide latency $L$: $N_{warps} \geq L/(\text{arith. ops between mem accesses})$. GPU tolerates latency through parallelism.

### 4.4.4 GPU Memory

**Rem. 72.** Registers (∼1cy), Shared/L1 (∼20–30cy), L2 (∼100–200cy), Global/HBM (∼400–800cy). **Coalescing**: Contiguous warp accesses → single wide DRAM transaction. **Bank Conflicts**: Same-bank accesses serialized; avoid by padding.

# 5. Memory

**Rem. 73.** Memory is the **limiting factor** in computation (time, space, energy). Robustness decreases with capacity (e.g., RowHammer).

**Def. 74. Memory types** (cost/performance trade-offs): FlipFlops (tens of T/bit), SRAM (6T/bit, fast, no refresh), DRAM (1T+1C, cheap, needs refresh), Storage (1T, non-volatile, slow).

## 5.1 Organization of Memory

**Def. 75.** Memory is an array of cells with addressing and output logic, divided into **banks** for concurrent access.

**Def. 76. DRAM Hierarchy**: Channels → DIMMs → Ranks → Chips → Banks → Rows/-Columns. A **row-buffer** holds an activated row; **row hit** (fast) vs. **row miss** (slow, needs precharge + activate).

## 5.2 Memory Technologies

**Def. 77. SRAM**: 6 transistors (cross-coupled inverters), fast, no refresh. **DRAM**: 1 transistor + capacitor, cheap, needs periodic refresh. **PCM**: Chalcogenide glass (amorphous/crystalline), non-volatile, dense, but slow, energy hungry, limited endurance.

## 5.3 Memory Hierarchy

**Rem. 78.** Can't have fast *and* large memory → hierarchy: small fast caches + large slow RAM. Intelligent data movement gives near-ideal performance.

**Def. 79. Data Locality**: **Temporal**: Recently accessed → likely accessed again. **Spatial**: Nearby data → likely accessed soon.

## 5.4 Cache

**Def. 80. Cache mapping**: Direct Mapped (1 location per address), Set Associative (one of $N$ ways per set), Fully Associative (any location).

**Def. 81. Replacement**: Random, FIFO, LRU, LFU. **Belady's OPT**: Evict line used furthest in future (optimal misses, not realizable).

**Def. 82. Write Policy**: Write-Back (cache only, dirty bit), Write-Through (cache + main memory). **Cache Compression**: Store base address + offsets to reduce space.

### 5.4.1 Prefetching

**Def. 83. Prefetching**: Fetch data before needed to hide latency. Mispredictions don't affect correctness. **Metrics**: Accuracy (used/sent), Coverage (prefetched misses/all misses), Timeliness, Cache Pollution.

**Def. 84. Hardware Prefetchers**: Stride (per-PC tracking), Memory-Region-Based, Stream Buffers (stride=1). **Software Prefetching**: Compiler-inserted instructions (limited portability). **Runahead Execution**: On long-latency miss, checkpoint state and speculatively execute ahead to generate prefetches. Enables memory-level parallelism without large ROB. Cannot handle pointer-chasing (→ AVD Prediction).

**Rem. 85. Modern**: Pythia (RL-based offset selection), Hermes (Perceptron predicts off-chip loads). **Multi-core**: Prefetching can cause cache/bus/DRAM contention across cores; needs throttling.

## 5.5 Virtual Memory

**Def. 86. Virtual Memory**: Each process gets illusion of private, large address space. VA → PA translation via page tables. Provides isolation between processes.

**Lem. 87. Address Translation**: VA split into VPN + Page Offset. Only VPN translated to PPN; offset unchanged.

**Def. 88. Page Table**: Per-process, stored in physical memory. **PTE**: Contains PPN + status bits (Present, R/W, User/Supervisor). **Multi-level**: Only first-level permanently in memory (x86-64: 4 levels).

**Def. 89. TLB**: Hardware cache of recent PTEs. Speeds up translation. **TLB miss**: Hardware walker (x86) or OS exception (MIPS).

**Def. 90. CLOCK Replacement**: Circular list with reference (R) bit. Clears R-bits during traversal, replaces first page with R=0 (approximates LRU).

### 5.5.1 Memory Protection

**Rem. 91. Access Control**: Permission bits in PTE. x86 uses 4 protection rings (Ring 0: kernel, Ring 3: user). Violations → hardware exception. **RowHammer**: Induce bit flips in adjacent rows; can flip PTE bits to gain full memory access.

**Lem. 92. Cache-VM Interaction**: **Virtually Addressed**: Fast but has Homonyms (same VA, different PA) and Synonyms (different VA, same PA). **Physically Addressed**: No aliasing but needs TLB lookup before cache access.