Flynn's Taxonomy: SISD, SIMD, MISD, and MIMD

Concepts, Real-World Applications, Diagrams, and Working Details

August 28, 2025

Abstract

Michael J. Flynn's taxonomy (1966) classifies computer organizations by the number of *instruction streams* and *data streams*. This document explains each class — SISD, SIMD, MISD, MIMD — with working details, processor–instruction–data viewpoints (no "processing element" abstraction), real-world examples, flowcharts, and block diagrams produced with TikZ. References to primary documentation and reputable sources are included at the end.

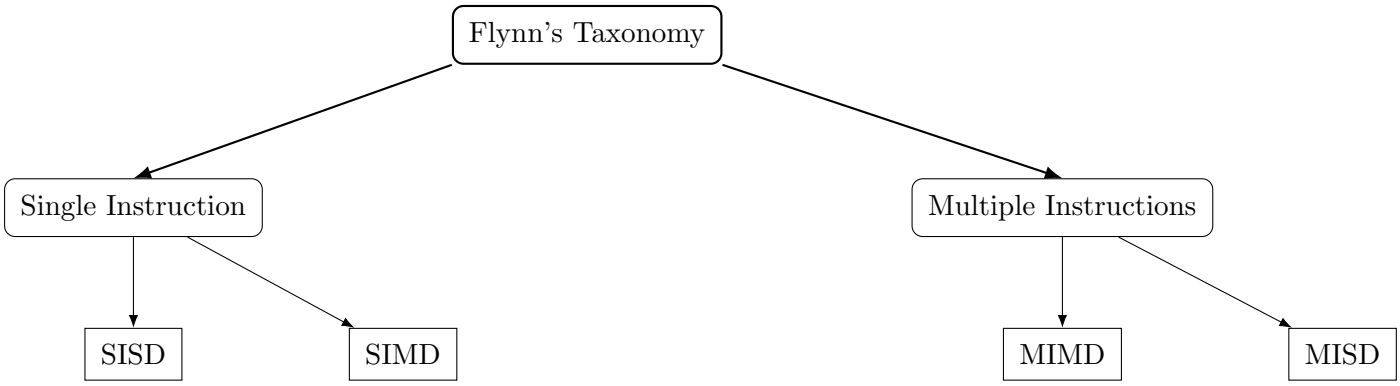
Contents

1 Overview

Flynn's taxonomy classifies machines by the cardinality of instruction and data streams:

- **SISD**: Single Instruction, Single Data
- **SIMD**: Single Instruction, Multiple Data
- **MISD**: Multiple Instruction, Single Data
- **MIMD**: Multiple Instruction, Multiple Data

Taxonomy at a glance

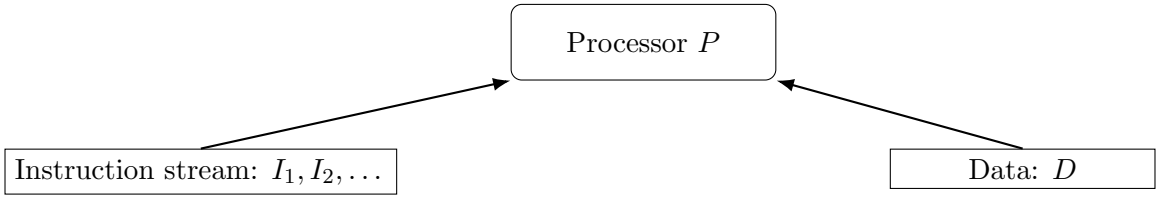


2 SISD: Single Instruction, Single Data

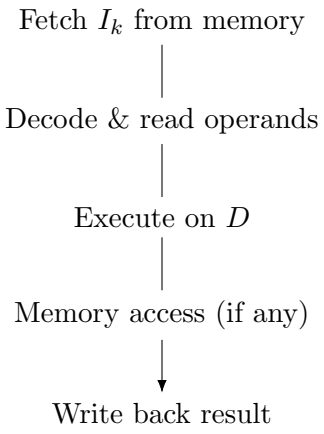
Definition

One processor executes a single instruction stream over a single data stream at any instant. This is the canonical von Neumann or Harvard single-core execution model.

Processor–Instruction–Data view



Typical pipeline (flowchart)



Real-world examples and uses

- **Microcontrollers** such as the 8-bit AVR ATmega328P (used on Arduino Uno) executing one instruction at a time over sensor/actuator data; typical in embedded control loops, appliance controllers, and simple IoT nodes (§[SISD1], §[SISD2]).

- **Classic CPUs** like Intel 8086 in single-tasking roles.[1]

Programming model

Ordinary serial code. Example: reading a sensor and updating a control output each loop iteration.

Listing 1: SISD-style control loop on a microcontroller

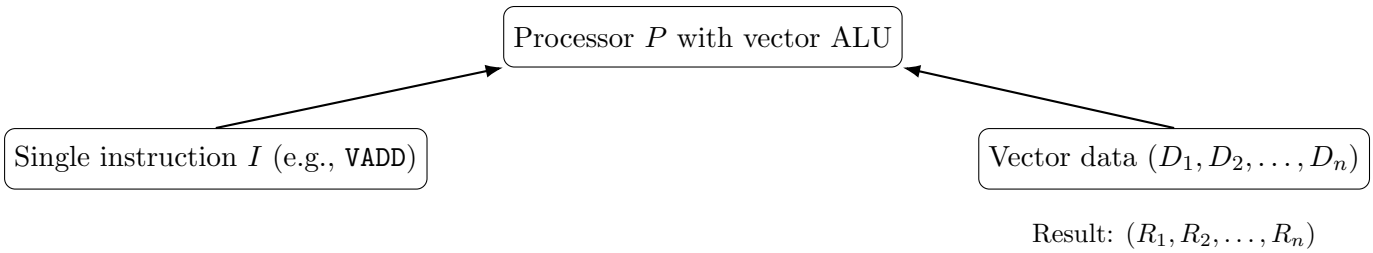
```
for (;;) {
  int d = read_sensor(); // data D
  int y = pid_update(d); // instructions I_k
  write_actuator(y);
}
```

3 SIMD: Single Instruction, Multiple Data

Definition

A processor issues *one* instruction that operates over multiple independent data items at once (vector/array style). On CPUs this appears as vector instructions (e.g., SSE/AVX). On GPUs the SIMT execution model issues the same instruction to many threads in lockstep.

Processor–Instruction–Data view (vector execution)

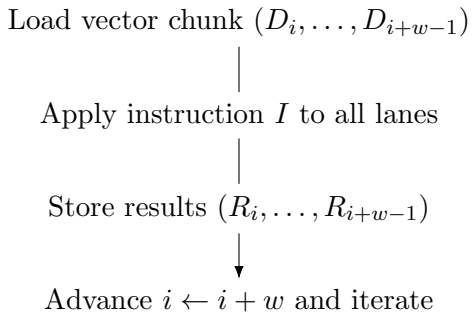


Working details

- **CPU vectors:** AVX/AVX2/AVX-512 extend registers to 256–512 bits and apply one op to many lanes; masking and fused multiply-add improve throughput (§[SIMD2], §[SIMD1], §[SIMD3]).
- **GPUs (SIMT):** kernels launch many threads that execute the same instruction sequence concurrently; hardware schedules them in warps on streaming multiprocessors (§[SIMD4], §[SIMD5], §[SIMD6]).

[1] While 8086 supports limited prefetch/pipelining and can be used in multi-processor modes, its basic single-core instruction/data execution exemplifies SISD for our purposes. See §[SISD3].

Flow of a vectorized loop (CPU)



Real-world examples

- **Image & video processing:** vectorized convolution, color transforms, codecs. CPU SIMD (SSE/AVX) can process 4–16 pixels per instruction; GPUs process whole tiles at once (§[SIMD2]).
- **Deep learning math:** matrix multiply (GEMM) and convolutions via CUDA/cuBLAS/cuBLASLt on GPUs; Tensor Cores accelerate mixed-precision GEMMs (§[SIMD7], §[SIMD8], §[SIMD9]).
- **Signal processing / scans:** vector scan, reductions, and packet-processing via AVX-512 (§**??**, §[SIMD3]).

Example: AVX2 add of eight floats

Listing 2: Single instruction applied to multiple data (AVX2)

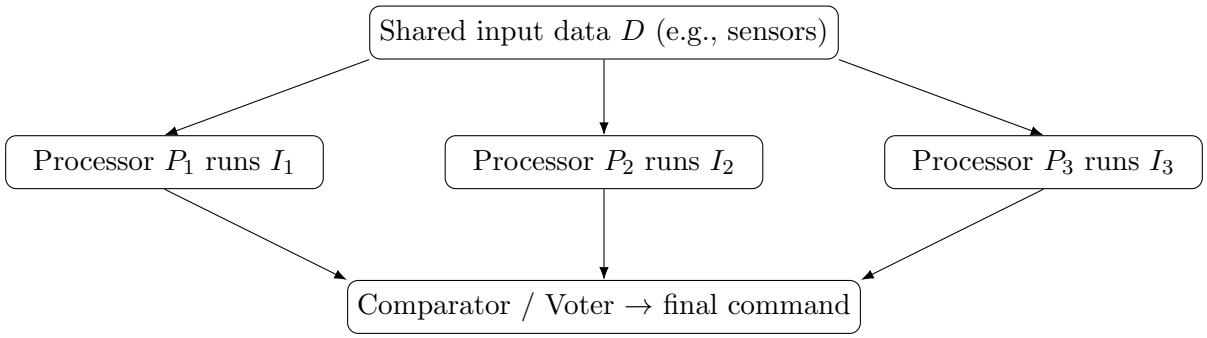
```
#include <immintrin.h>
void add8(const float *a, const float *b, float *c) {
  __m256 va = _mm256_loadu_ps(a);
  __m256 vb = _mm256_loadu_ps(b);
  __m256 vc = _mm256_add_ps(va, vb); // one instruction over 8 lanes
  _mm256_storeu_ps(c, vc);
}
```

4 MISD: Multiple Instruction, Single Data

Definition and note

Multiple processors execute *different* instruction streams on the *same* input data. This class is rare in commodity computing; it appears primarily in fault-tolerant systems where diverse algorithms process the same sensor data and compare/vote on outputs.

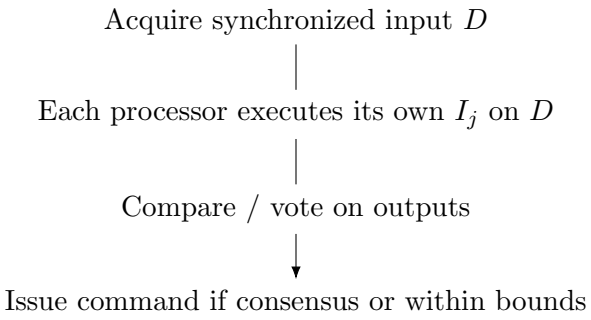
Processor–Instruction–Data view (diverse redundancy)



Where it shows up

- **Safety-critical avionics** using dissimilar redundancy: independent processors and independently developed software versions consume the same sensor data and their outputs are compared or voted before actuating flight controls ("N-version programming") (§[MISD1], §[MISD2], §[MISD3], §[MISD4]).
- **Space systems** with diverse backup computers (e.g., Shuttle BFS) developed independently to avoid common-mode faults; same inputs, different instruction sequences (§[MISD5]).

Working details (flowchart)



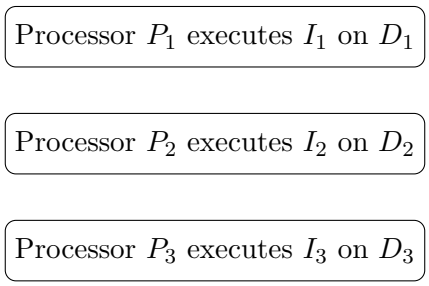
Caveat. Many pipelines or systolic arrays are *not* MISD in Flynn's strict sense; they often map to SIMD or MIMD depending on how instructions/data are organized. MISD in practice is mostly about diverse-redundant computation over the same input for reliability, not throughput.

5 MIMD: Multiple Instruction, Multiple Data

Definition

Multiple processors execute different instructions on different data, concurrently. This is the dominant general-purpose architecture today: multi-core CPUs, many-core servers, and distributed clusters.

Processor–Instruction–Data view



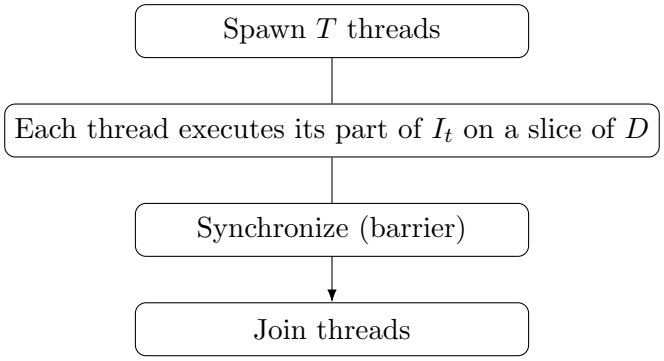
Working details

Two mainstream organizations:

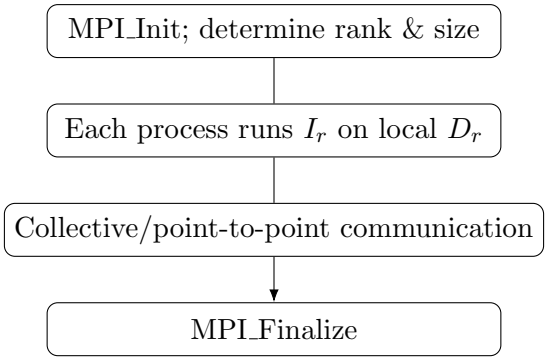
- **Shared memory (SMP/NUMA):** threads on the same multi-core processor cooperate via shared address space; typical programming with OpenMP pragmas.
- **Distributed memory (clusters):** processes exchange messages across nodes; typical programming with MPI.

Flowcharts

Shared-memory (OpenMP-style)



Distributed-memory (MPI-style)



Real-world examples

- **Web back-ends and microservices**: many processes (different instruction paths) handle distinct requests/data on a multi-core server or across a cluster.
- **HPC & scientific computing**: domain decomposed PDE solvers, linear algebra, and simulations where each rank owns a subdomain; MPI 4.x and OpenMP 5/6 are standard toolchains (§[MIMD1], §[MIMD2]).
- **Data engineering**: Spark/Hadoop clusters running independent tasks over partitioned datasets.

6 Choosing the right class for a workload

- Use **SISD** when the work is tiny, latency-bound, or on very small embedded targets.
- Use **SIMD** when the same mathematical kernel applies elementwise over large arrays (images, tensors, vectors).
- Consider **MISD** only for safety cases where diverse implementations must agree before actuating.
- Use **MIMD** for heterogeneous or task-parallel workloads, large simulations, and services handling many unrelated requests.

7 References

Below, "primary" means original standards, vendor docs, or peer-reviewed material.

Flynn and overview

[F1] M. J. Flynn, "Very high-speed computing systems," *Proceedings of the IEEE*, 54(12):1901–1909, 1966. Primary. DOI: https://doi.org/10.1109/PROC.1966.5273.

[F2] "Flynn's taxonomy" (overview), Wikipedia, citing Flynn 1966 and 1972. https://en.wikipedia.org/wiki/Flynn%27s_taxonomy.

SISD examples

[SISD1] Microchip, "AVR Instruction Set Manual," DS40002198. Primary. https://ww1.microchip.com/downloads/en/DeviceDoc/AVR-InstructionSet-Manual-DS40002198.pdf.

[SISD2] Arduino Uno R3 (ATmega328P) datasheets (e.g., Microchip ATmega328/P). Primary. https://docs.arduino.cc/resources/datasheets/Atmel-42735-8-bit-AVR-Microcontroller-ATmDatasheet.pdf.

[SISD3] Intel, *8086 Family User's Manual*, Oct. 1979 (historical). https://bitsavers.org/components/intel/8086/9800722-03_The_8086_Family_Users_Manual_Oct79.pdf.

[SISD4] "Von Neumann architecture," Wikipedia (background). https://en.wikipedia.org/wiki/Von_Neumann_architecture.

SIMD (CPU vectors and GPUs)

[SIMD1] Intel, *Intel Intrinsics Guide* (SSE/AVX/AVX-512). Primary. https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html.

[SIMD2] C. Lomont, "Introduction to Intel AVX," LLNL HPC (tutorial/notes). https://hpc.llnl.gov/sites/default/files/intelAVXintro.pdf.

[SIMD3] Intel, "Intel AVX-512 instruction set for packet processing," Technology Guide. Primary. https://builders.intel.com/docs/networkbuilders/intel-avx-512-instruction-set-for-packpdf.

[SIMD4] NVIDIA, *CUDA C++ Programming Guide* (SIMT architecture). Primary. https://docs.nvidia.com/cuda/cuda-c-programming-guide/.

[SIMD5] NVIDIA, *Parallel Thread Execution ISA*, Sec. 3.1 (Set of SIMT multiprocessors). Primary. https://docs.nvidia.com/cuda/parallel-thread-execution/.

[SIMD6] NASA NAS, "Basics on NVIDIA GPU Hardware Architecture" (SM, SIMT). https://www.nas.nasa.gov/hecc/support/kb/basics-on-nvidia-gpu-hardware-architecture_704.html.

[SIMD7] NVIDIA, *cuBLAS Library Guide*. Primary. https://docs.nvidia.com/cuda/cublas/.

[SIMD8] NVIDIA, *cuBLAS Library (PDF, Aug 2025)* (cuBLASLt, GEMM). Primary. https://docs.nvidia.com/cuda/pdf/CUBLAS_Library.pdf.

[SIMD9] NVIDIA, "Matrix Multiplication Background User's Guide," 2023. Primary. https://docs.nvidia.com/deeplearning/performance/dl-performance-matrix-multiplication/.

MISD (fault tolerance and dissimilar redundancy)

[MISD1] M. Lyu (ed.), "The Methodology of N-Version Programming," book chapter (tutorial). https://www.cse.cuhk.edu.hk/~lyu/book/sft/pdf/chap2.pdf.

[MISD2] W. Torres-Pomales, "Software Fault Tolerance: A Tutorial," NASA/TM-2000-210616. Primary. https://ntrs.nasa.gov/api/citations/20000120144/downloads/20000120144.pdf.

[MISD3] Airbus, "Airbus fly-by-wire: a process toward total dependability," ICAS 2006. Primary. https://www.icas.org/icas_archive/ICAS2006/PAPERS/050.PDF.

[MISD4] Airbus Safety First, "Computer mixability – an important issue," detailing A330/A340 FCPC/SEC roles. Primary. https://safetyfirst.airbus.com/app/themes/mh_newsdesk/documents/archives/computer-mixability-an-important-issue.pdf.

[MISD5] NASA, "Space Shuttle Avionics System" and related PASS/BFS documentation (independent backup software). Primary. https://ntrs.nasa.gov/api/citations/19900015844/downloads/19900015844.pdf.

MIMD and programming models

[MIMD1] Message Passing Interface Forum, *MPI 4.1 Standard*, Nov. 2023. Primary. https://www.mpi-forum.org/docs/mpi-4.1/mpi41-report.pdf.

[MIMD2] OpenMP Architecture Review Board, *OpenMP API Specification* (v5.2 and v6.0). Primary. https://www.openmp.org/specifications/.

Notes on classification: GPUs expose SIMT rather than classic SIMD terminology; under Flynn, SIMT machines still fall under SIMD because one logical instruction stream is applied over many independent data elements concurrently (§[SIMD4], §[SIMD5]). MISD examples are intentionally conservative and focused on safety literature.