# 14. 상용 프로세서 분석

## 14.1 Intel x86/x64 구조 분석

### 🧠 기본 개요

| 항목 | 설명 |
|---|---|
| **x86** | Intel 8086부터 시작된 16/32비트 CISC 아키텍처 |
| **x86-64 (x64)** | AMD가 최초 제안한 64비트 확장 (AMD64) → Intel도 수용 (Intel 64) |
| **ISA 유형** | Complex Instruction Set Computing (CISC) |
| **호환성** | 하위 호환 극강: 16 → 32 → 64비트 실행 모두 가능 |

---

### 📜 역사적 흐름 요약

| CPU | 비트 | 특징 |
|---|---|---|
| 8086 | 16비트 | Real Mode |
| 80286 | 16비트 | Protected Mode 도입 |
| 80386 | 32비트 | Paging, Multitasking |
| Pentium | 32비트 | 파이프라인, MMX |
| x86-64 | 64비트 | 64비트 주소 공간, 레지스터 확장 |

---

### 🧩 x86 실행 모드 3가지

| 모드 | 설명 |
|---|---|
| **Real Mode** | 16비트 주소, 세그먼트 기반, BIOS 초기화 |
| **Protected Mode** | 32비트 주소, 세그먼트 + 페이지 보호, 링 구조 |
| **Long Mode** | 64비트 모드, 대부분 세그먼트 제거, 페이징 필수 |

> 모든 x86 시스템은 부팅 시 **Real Mode로 시작**하여 OS가 **Protected/Long Mode로 전환**함

---

## 📦 레지스터 체계

### ✅ x86 (32비트)

| 범용 레지스터 | 역할 |
|---|---|
| EAX, EBX, ECX, EDX | 연산용 |
| ESI, EDI | 메모리 복사, 문자열 처리 |
| EBP | Base Pointer |
| ESP | Stack Pointer |
| EIP | Instruction Pointer |
| EFLAGS | 상태 플래그 |

---

### ✅ x64 (64비트 확장)

- 기존 8개 → **총 16개** 레지스터로 확장 ( `RAX ~ R15` )
- 각 레지스터의 크기: 64비트 ( `RAX` ), 하위 접근: `EAX` , `AX` , `AL`

| 레지스터 | 크기 |
|---|---|
| RAX, RBX, RCX ... | 64bit |
| EAX, EBX ... | 32bit (하위) |
| AX, AL, AH | 16bit / 8bit |

---

## 🔐 링 구조 (Privilege Level)

| 링 | 권한 | 사용 |
|---|---|---|
| Ring 0 | 최고권한 | OS 커널 |
| Ring 1~2 | 중간 권한 | 미사용 (OS 내부 분할용) |
| Ring 3 | 사용자 | 일반 앱 |

→ CPL(Current Privilege Level): 코드 실행 시의 링 값
→ **시스템 콜 = Ring3 → Ring0 전환 + 보호 영역 접근**

---

## 🧠 세그먼트 모델 (Segmentation)

### ✅ Real Mode (20bit 물리주소 계산)

```
Physical Addr = Segment * 16 + Offset
```

→ 예: `0xFFFF:0x0001` → `0xFFFF0 + 0x1 = 0xFFFF1`

### ✅ Protected Mode

- 세그먼트 = 인덱스
- 실제 주소 = `Segment Selector → GDT/LDT → Base + Offset`

### ✅ Long Mode

- 거의 사용 안함
- 코드 세그먼트(CS), 데이터 세그먼트(DS)는 **flat memory model** 적용

---

## 🧠 페이징 (Paging)

| 항목 | 설명 |
| --- | --- |
| PAE | 32비트 CPU에서 36비트 주소 지원 (4-level paging) |
| x64 페이징 | CR3 → PML4 → PDPT → PDT → PT → 페이지 |

→ 모든 x64 OS는 **페이징 활성화 필수**

## 🧠 x86 vs x64 비교 정리

| 항목 | x86 (IA-32) | x86-64 (AMD64 / Intel64) |
| --- | --- | --- |
| 주소 크기 | 32비트 | 64비트 |
| 가상 주소 공간 | 4GB | 256TB (48~57bit) |
| 물리 주소 공간 | 4GB (PAE로 64GB) | 최대 52비트 (~4PB) |
| 레지스터 수 | 8개 | 16개 |
| 시스템 콜 | `int 0x80` | `syscall` (더 빠름) |
| 실행 모드 | Protected Mode | Long Mode |
| 세그먼트 사용 | 활발 | 거의 사용 안함 |

## ⚙️ OS와의 연동

| 구조 | 설명 |
| --- | --- |
| GDT / LDT | 세그먼트 정보를 저장하는 테이블 |
| TSS (Task State Segment) | 태스크 전환 시 스택 포인터 보관 |
| IDT (Interrupt Descriptor Table) | 인터럽트/예외 핸들러 주소 저장 |
| CR0~CR4 | 제어 레지스터: 페이징, 보호모드 설정 |
| MSR | 시스템 콜, Timestamp 등 CPU 특수기능 제어 |

## 📌 요약 정리

| 항목 | 설명 |
| --- | --- |
| **x86** | 32비트, Protected Mode 중심 |
| **x64** | 64비트, Long Mode + 페이징 필수 |
| **레지스터** | RAX~R15, 총 16개, 고정폭 |
| **모드 전환** | Real → Protected → Long |
| **링 구조** | Ring0 (OS), Ring3 (User) |
| **페이징 구조** | PML4 기반 4~5단계 계층 주소 변환 |
| **세그먼트** | Long Mode에서는 Flat Model |

# 14.2 ARM Cortex 시리즈 구조

## 🧠 ARM 기본 개요

| 항목 | 설명 |
| --- | --- |
| ISA 종류 | **RISC (Reduced Instruction Set Computer)** |
| 설계자 | ARM Ltd (→ IP 라이선스 제공 방식) |
| 명령어 세트 | ARM, Thumb, Thumb-2, A64 (64bit) |
| 사용 분야 | 스마트폰, 임베디드, 서버, IoT, 자동차 등 |

## 🧩 Cortex 시리즈 분류

| 시리즈 | 용도 | 대표 제품 |
| --- | --- | --- |
| **Cortex-A** | **Application Processor** (OS 구동, 고성능) | 스마트폰, 리눅스 |

| 시리즈 | 용도 | 대표 제품 |
| --- | --- | --- |
| **Cortex-R** | **Real-time Processor** (고속, 예측 가능) | SSD, 자동차 ECU |
| **Cortex-M** | **Microcontroller Core** (초저전력, 간소화) | STM32, nRF, Arduino 등 |

## 🧠 명령어 아키텍처 요약

| 명령어 셋 | 비트 | 설명 |
| --- | --- | --- |
| **ARM** (A32) | 32bit | 고성능, 코드 크기 큼 |
| **Thumb** | 16bit | 코드 크기 작음, 효율성 ↑ |
| **Thumb-2** | 16/32bit 혼합 | Cortex-M의 기본 |
| **A64** | 64bit | ARMv8 이상에서 사용 (Cortex-A 전용) |

## 📦 Cortex-A 구조 (Application Core)

| 항목 | 설명 |
| --- | --- |
| 주소 공간 | 32비트 (ARMv7-A), 64비트 (ARMv8-A) |
| MMU | 지원 (가상 메모리, 페이징) |
| 실행 모드 | EL0~EL3 (ARMv8), 사용자/슈퍼바이저/IRQ 등 (ARMv7) |
| 예외 처리 | 인터럽트, Fast Interrupt (FIQ), Supervisor Call 등 |
| ISA | A32 (ARM), A64 (64비트) |
| 특성 | 멀티코어, OoO 가능, SIMD, NEON, TrustZone 지원 |
| 사용처 | Android/Linux 기반 모바일, 임베디드 서버 |

## ⚙️ Cortex-M 구조 (Microcontroller Core)

| 항목 | 설명 |
| --- | --- |
| 주소 공간 | 32비트 고정 |
| MMU | ❌ 없음, 대신 MPU (Memory Protection Unit) |
| ISA | Thumb / Thumb-2 전용 |
| 인터럽트 | Nested Vectored Interrupt Controller (NVIC) 내장 |
| 특성 | 파이프라인 단순, 실시간성 우수, 저전력 |
| 사용처 | STM32, nRF, IoT, 웨어러블, 센서 MCU 등 |

## 🛠️ Cortex-R 구조 (Real-time Core)

| 특징 | 내용 |
|---|---|
| 하버드 구조 기반 | 명령어/데이터 분리 |
| 고속 인터럽트 | 예측 가능성 최우선 |
| TCM 지원 | Tightly Coupled Memory (SRAM과 유사) |
| Dual CPU lockstep 지원 | 기능안전용 (ASIL-D, 항공우주 등) |
| 사용처 | SSD 컨트롤러, 자동차 실시간 ECU, 산업용 PLC |

## 📊 ARMv7 vs ARMv8 vs ARMv9

| 항목 | ARMv7-A | ARMv8-A | ARMv9-A |
|---|---|---|---|
| 주소 | 32bit | 64bit (AArch64), 호환 AArch32 | 64bit only |
| 모드 | 사용자, SVC, IRQ | EL0~EL3 | EL0~EL3 + Secure Realm |
| TrustZone | 일부 | 기본 포함 | Confidential Compute 확장 |
| SIMD | NEON | NEON, SVE | SVE2, AI 확장 |

## 🔁 실행 모드 (ARMv8-A 기준)

| 모드 | 설명 |
|---|---|
| **EL0** | 일반 사용자 앱 |
| **EL1** | OS 커널 (Linux 등) |
| **EL2** | Hypervisor |
| **EL3** | Secure Monitor (TrustZone) |

## 🔄 파이프라인 구조

| 시리즈 | 파이프라인 |
|---|---|
| Cortex-M0 | 3-stage |
| Cortex-M4 | 3-stage + FPU |
| Cortex-A53 | 8-stage in-order |
| Cortex-A76 | Out-of-Order, 11+ stage |

→ **Cortex-A는 고성능**, **Cortex-M은 저지연 + 간결성**, **Cortex-R은 실시간 예측성** 중심

## 🧠 MMU vs MPU

| 항목 | MMU (Cortex-A) | MPU (Cortex-M) |
|---|---|---|
| 주소 변환 | 가능 (가상→물리) | 불가능 (물리 주소 기반) |
| 보호 방식 | 페이지 단위 | 영역 단위 (region-based) |
| 캐시 | 페이지 단위 캐시 설정 | 제한적 |
| 쓰기 보호, 접근 권한 | 모두 지원 | 단순 형태 |
| 용도 | OS 기반 시스템 | RTOS, Baremetal MCU |

## 📌 요약 정리

| 항목 | Cortex-A | Cortex-R | Cortex-M |
|---|---|---|---|
| 용도 | 스마트폰, 리눅스 | 실시간 시스템 | MCU, IoT |
| 주소 공간 | 32/64비트 | 32비트 | 32비트 |
| 명령어 | A32/A64 | A32 | Thumb-2 |
| 가상 메모리 | MMU | 가능 | MPU |
| 인터럽트 | 고급 (GIC) | 고속 (FIQ 등) | NVIC |
| 아키텍처 모드 | EL0~EL3 | IRQ, SVC | Thread/Handler Mode |
| 실행 환경 | Linux, Android | RTOS | Baremetal, RTOS (FreeRTOS 등) |

# 14.3 RISC-V ISA와 오픈소스 아키텍처

## 🧠 RISC-V란?

**RISC-V (Risk-Five)**는
UC Berkeley에서 개발한 **완전 공개형(오픈소스) RISC ISA**로,
**특허 없이 누구나 사용/수정/배포 가능한 아키텍처 표준**이다.

| 항목 | 설명 |
|---|---|
| ISA 유형 | RISC (Reduced Instruction Set Computer) |
| 특징 | 오픈소스, 모듈화, 간결함, 확장성 |
| 라이선스 | BSD (상업적 사용 가능) |
| 공식 조직 | RISC-V International |

## 🧩 RISC-V ISA 구조 개요

RISC-V는 기본 **RV32 / RV64 / RV128** 구조 위에
**선택적 확장 모듈**을 덧붙이는 **모듈형 ISA** 구조이다.

### ✅ 기본 ISA 이름 구성

```
RV32IMAFDCSU
```

| 필드 | 의미 |
| --- | --- |
| RV32 / RV64 | 기본 워드 크기 (32bit / 64bit) |
| I | 정수 명령어 집합 (필수) |
| M | 곱셈/나눗셈 명령어 |
| A | Atomic 명령어 (lock-free 구조) |
| F | Single-precision 부동소수점 |
| D | Double-precision 부동소수점 |
| C | 압축 명령어 (16bit 인코딩) |
| S | Supervisor mode 지원 (OS 실행용) |
| U | User mode 지원 |

## 📦 레지스터 구조

| 종류 | 개수 | 설명 |
| --- | --- | --- |
| **x0~x31** | 32개 | 정수 레지스터 (x0 = 항상 0) |
| **f0~f31** | 32개 | 부동소수점 레지스터 (옵션) |
| **pc** | 1개 | Program Counter |
| **csr** | 다수 | Control & Status Registers (권한, 인터럽트 등) |

### 주요 레지스터 예

| 이름 | 번호 | 용도 |
| --- | --- | --- |
| zero | x0 | 항상 0 |
| ra | x1 | return address |
| sp | x2 | stack pointer |
| gp | x3 | global pointer |

| 이름 | 번호 | 용도 |
| --- | --- | --- |
| tp | x4 | thread pointer |
| a0~a7 | x10~x17 | 함수 인자, 반환값 |
| s0~s11 | x8~x9, x18~x27 | saved regs |
| t0~t6 | x5~x7, x28~x31 | temporary regs |

## 🧠 명령어 포맷

| 포맷 | 사용 예 | 필드 |
| --- | --- | --- |
| R-type | add, sub, mul | `opcode + rd + funct3 + rs1 + rs2 + funct7` |
| I-type | load, jalr | `opcode + rd + funct3 + rs1 + imm` |
| S-type | store | `opcode + imm + rs1 + rs2 + funct3` |
| B-type | branch | `opcode + offset + rs1 + rs2 + funct3` |
| U-type | lui, auipc | `opcode + rd + imm` |
| J-type | jal | `opcode + rd + offset` |

→ **명령어 길이 고정 (32bit)**, `C` 확장 시 16bit도 지원

## 🔐 실행 모드

| 모드 | 설명 |
| --- | --- |
| **User Mode (U-mode)** | 앱 실행 |
| **Supervisor Mode (S-mode)** | OS 커널 |
| **Machine Mode (M-mode)** | 부트로더, 하드웨어 제어 |
| **Hypervisor Mode (H-mode)** | 가상화 확장 시 (선택사항) |

→ 대부분의 MCU는 U/M만, Linux는 U/S/M 사용

## 📚 페이징 구조 (Sv32, Sv39, Sv48)

| 모드 | 가상 주소 비트 | 페이지 수준 | 주소 범위 |
| --- | --- | --- | --- |
| Sv32 | 32bit | 2-level page table | 4GB |
| Sv39 | 39bit | 3-level | 512GB |
| Sv48 | 48bit | 4-level | 256TB |

→ 페이지 크기 기본 4KB, L1~L3 Page Table 구조 → Linux fully supported

## 🛠️ 오픈소스 구현체 (RTL/IP)

| 구현체 | 언어 | 특징 |
|---|---|---|
| **RocketChip** | Chisel | Berkeley, Linux 가능 |
| **BOOM** | Chisel | Out-of-Order Core |
| **PicoRV32** | Verilog | 초소형, FPGA용 |
| **VexRiscv** | SpinalHDL | 소형 MCU에 최적 |
| **CV32E40P** | Verilog | OpenHW Group, MCU용 |
| **Shakti** | Bluespec | 인도 IIT-M 기반, 보안 확장 |
| **CVA6 (ex-Ariane)** | VHDL | Linux-capable, 고성능 Core |

## 🧪 상용 RISC-V SoC 예시

| 제품 | 용도 | 특징 |
|---|---|---|
| **SiFive U740** | 리눅스용 SoC | 4+1코어, 64bit, L2 캐시 |
| **Kendryte K210** | AI MCU | CNN 가속기 내장 |
| **ESP32-C3** | IoT MCU | RISC-V + WiFi/BT |
| **StarFive VisionFive** | 리눅스 SBC | StarFive JH7100, Fedora 지원 |
| **Pine64 Ox64** | AI+IoT | 64bit RISC-V + LPDDR4 |

## 📦 상용 OS 호환

| OS | 지원 여부 |
|---|---|
| **Linux (mainline)** | ✅ Sv39 기반 |
| **Zephyr RTOS** | ✅ MCU (U/M 모드 기반) |
| **FreeRTOS** | ✅ U/M 기반 |
| **Debian, Fedora, Arch** | ✅ RV64 Linux 배포판 |
| **QEMU** | ✅ 모든 모드 에뮬레이션 지원 |

## 📌 요약 정리

| 항목 | 설명 |
|---|---|
| **RISC-V 특징** | 오픈소스, 모듈형 ISA, 특허 무료 |
| **기본 구조** | RV32/RV64 + 확장모듈(IMACFD...) |

| 항목 | 설명 |
| --- | --- |
| **명령어 포맷** | 고정폭(32bit), 구조 단순 |
| **모드** | U/S/M (+H) |
| **페이징** | Sv32~Sv48 → Linux 지원 |
| **실제 칩** | SiFive, StarFive, Kendryte, Espressif |
| **소프트웨어 스택** | Linux, RTOS, QEMU, GCC/LLVM toolchain 등 활발 |

# 14.4 Apple Silicon (M1, M2) 구조 분석

## 🧠 Apple Silicon이란?

**Apple Silicon**은 Apple이 자체 설계한 **ARM 기반 SoC(System on a Chip)** 시리즈로,
2020년부터 Mac 및 iPad 제품군에 **x86(Intel) 대신 탑재**되기 시작했다.

| 항목 | 설명 |
| --- | --- |
| 아키텍처 | ARMv8.4-A (M1), ARMv8.5/9-A (M2 이후) |
| 제조 공정 | TSMC N5 / N4 / N3 |
| 설계 방식 | Apple 독자 설계 + ARM ISA 라이선스 |
| 운영체제 | macOS, iPadOS (Darwin 기반) |

## 🧩 전체 시스템 구성 요약 (M1 기준)

```
+--------------------------------------+
| Performance Cores (Firestorm, 4x)     |
| Efficiency Cores (Icestorm, 4x)       |
| Unified L2/L3 Cache                  |
| Unified Memory Architecture (UMA)    |
| GPU (8~10코어)                        |
| Neural Engine (16-core NPU)          |
| Secure Enclave                        |
| ISP, Media Engine (H.264/HEVC Pro HW)|
| Thunderbolt/PCIe Interface           |
+--------------------------------------+
```

## 💡 big.LITTLE 설계 (Asymmetric Multiprocessing)

| 구성 | 설명 |
| --- | --- |
| **P-Core (Firestorm)** | 고성능 코어 (고속, Out-of-Order, 큰 L1/L2) |
| **E-Core (Icestorm)** | 저전력 코어 (In-order, 작은 L1/L2, 스케줄 오프로드) |

| 구성 | 설명 |
| --- | --- |
| **Task Scheduler** | macOS가 코어 특성에 따라 자동 분배 (Energy-aware Scheduling) |

→ P-core는 무거운 앱 처리, E-core는 백그라운드/시스템 유지

## 🧠 M1 / M2 주요 사양 비교

| 항목 | M1 | M2 |
| --- | --- | --- |
| 공정 | 5nm | 5nm+ (N5P) |
| CPU 구성 | 4P + 4E | 4P + 4E |
| GPU | 7~8코어 | 8~10코어 |
| NPU (Neural Engine) | 16코어 | 16코어, 성능 향상 |
| 메모리 대역폭 | 68 GB/s | 100 GB/s |
| 최대 메모리 | 16GB | 24GB |
| Transistor 수 | 160억 | 200억+ |

## 🧠 CPU 코어 구조 (M1 Firestorm)

| 구조 | 설명 |
| --- | --- |
| Out-of-Order 파이프라인 | 고도로 깊은 파이프라인 (16+ stages) |
| 넓은 fetch/decode (8-wide) | ARM 코어 중 최대 수준 |
| 별도 MOP 캐시 | micro-op 캐시로 명령어 재디코딩 감소 |
| 192KB L1I / 128KB L1D | 대형 L1 캐시 |
| L2 12MB (공유) | 4 P-core 공유 |
| Pre-decode queue + branch prediction | 성능 극대화 설계 |

## 💾 Unified Memory Architecture (UMA)

> CPU, GPU, NPU가 **하나의 물리 메모리를 공유**하는 구조

| 장점 | 설명 |
| --- | --- |
| Zero-copy 처리 | GPU ↔ CPU 간 버퍼 복사 필요 없음 |
| 전력 효율성 | 중복 데이터 없음 |
| 성능 향상 | Bandwidth, cache coherency 관리 최적화 |

## 🎮 GPU 아키텍처

| 항목 | M1 GPU | M2 GPU |
|---|---|---|
| 아키텍처 | Apple Custom Tile GPU | 성능/클럭 증가 |
| 코어 수 | 7~8 | 8~10 |
| Metal 최적화 | macOS와 완전 통합 | |
| Tiled-based Renderer | ARM계 GPU 전통 방식 | |

→ GPU는 자체 고안된 **Tile-Based Deferred Rendering** 방식으로 전력당 성능 우수

## 🧠 Neural Engine (NPU)

| 항목 | 사양 |
|---|---|
| 구성 | 16-core NPU |
| 성능 | 11 TOPS (M1), 15.8 TOPS (M2) |
| 용도 | Face ID, Live Text, 음성 인식, 머신러닝 가속 |
| API | CoreML, Accelerate, MPS, TensorFlow-Metal 등 |

## 🔐 Secure Enclave & 보안 구조

| 구성 | 설명 |
|---|---|
| Secure Enclave | 자체 SoC 내 보안코어 (ARMv8-M 계열) |
| SEP OS | 별도 마이크로커널 기반 보안 OS |
| 키 관리 | Face ID, Touch ID, Apple Pay 보안 처리 |
| Boot Chain | Root-of-Trust → Secure Boot 단계별 서명 검증 |

## 🔁 x86 대비 아키텍처적 장점

| 항목 | Apple Silicon (ARM) | Intel x86 |
|---|---|---|
| 명령어 | RISC (단순, 고효율) | CISC (복잡, 해석 부담) |
| 파워 효율 | 우수 (P+E코어 조합) | 상대적으로 불리 |
| 메모리 구조 | UMA (CPU/GPU 공유) | DRAM + VRAM 분리 |
| 통합성 | SoC (CPU+GPU+IO+ML+보안) | CPU + 외장 구성 |
| 네이티브 OS 설계 | macOS에 맞춤 최적화 | 범용 구조 |

### 🧪 개발자 관점에서의 차이

| 항목 | 설명 |
|---|---|
| 명령어 호환성 | Rosetta 2를 통한 x86 → ARM 변환 |
| 네이티브 빌드 | Xcode로 `arm64` 타깃 가능 |
| Docker / VM | Apple Hypervisor Framework 기반 (`UTM`, `Parallels`) |
| 성능 추적 | `Instruments`, `Activity Monitor` ARM-aware |

### 📌 요약 정리

| 항목 | 내용 |
|---|---|
| 아키텍처 | ARMv8.4~9 기반 자체 설계 |
| 코어 구성 | 4P + 4E big.LITTLE |
| 메모리 | UMA (CPU-GPU-NPU 공유) |
| GPU | Apple Tile-based GPU, Metal 최적화 |
| NPU | Neural Engine, 16-core ML 가속 |
| 보안 | Secure Enclave, Secure Boot, SEP |
| 전력 효율 | 매우 우수 (배터리 + 성능 균형 최적) |
| OS 연계 | macOS, iPadOS, CoreML, Metal 등 완전 통합 설계 |

## 14.5 GPU와 SIMD 구조

### 🧠 SIMD란?

**SIMD (Single Instruction, Multiple Data)**는
**하나의 명령어로 여러 데이터 요소에 동일한 연산을 동시에 수행하는 구조**이다.

| 예시 |

```
// 일반 반복문
for (int i = 0; i < 4; i++) C[i] = A[i] + B[i];

// SIMD 예 (1명령어로 4개 처리)
C = A + B  // A, B는 벡터
```

## 🧩 SIMD 명령어 예시

| 플랫폼 | 명령어 세트 | 설명 |
|---|---|---|
| Intel | **SSE**, **AVX** | 128~512bit, 벡터 연산 |
| ARM | **NEON** | 128bit SIMD |
| RISC-V | **V Extension** | 가변 길이 벡터 |
| GPU | CUDA warp = SIMT | SIMD 비슷한 흐름, 스레드마다 데이터 분기 가능 |

---

## 🔄 CPU와 GPU의 병렬성 차이

| 구조 | 설명 |
|---|---|
| **CPU** | 소수의 강력한 코어 → **MIMD** 방식 |
| **GPU** | 수천 개의 약한 코어 → **SIMT (SIMD 유사)** 방식 |

---

## 🎮 GPU의 내부 구조 (NVIDIA 기준)

```
+-----------------------------------+
| Streaming Multiprocessor (SM)     |
|  ├── Warp Scheduler (32 threads)  |
|  ├── ALU × N (FP32, INT32, SFU)   |
|  ├── Shared Memory (Fast SRAM)    |
|  └── Registers                    |
+-----------------------------------+

한 SM은 여러 워프(warp)를 스케줄링
한 워프 = 32개의 스레드
```

---

## 📦 SIMD vs GPU 구조 비교

| 항목 | SIMD (CPU) | GPU |
|---|---|---|
| 연산 단위 | 벡터 단위 (128~512bit) | 워프 (32스레드) |
| 제어 흐름 | 단일 명령어 | 워프 내 divergence 가능 |
| 연산 유닛 | ALU + SIMD | 다수의 ALU 병렬 |
| 메모리 | 계층적 cache | Shared + Global + Local |

---

## 🔧 SIMD 명령어 예 (Intel AVX2, ARM NEON)

```
#include <immintrin.h> // Intel AVX2
__m256 a = _mm256_loadu_ps(A);   // 8 x float
__m256 b = _mm256_loadu_ps(B);
__m256 c = _mm256_add_ps(a, b);  // SIMD 덧셈

#include <arm_neon.h> // ARM NEON
float32x4_t a = vld1q_f32(A);
float32x4_t b = vld1q_f32(B);
float32x4_t c = vaddq_f32(a, b);
```

---

## 🧠 GPU 메모리 계층 구조

| 계층 | 설명 |
| --- | --- |
| **Register** | 각 스레드 전용 |
| **Shared Memory** | SM 내부 고속 공유 메모리 |
| **Global Memory** | GPU 전체 공유 DRAM |
| **Texture/Constant Memory** | 특수 목적, 캐싱 최적화됨 |
| **Host Memory** | CPU와 공유 (DMA 필요, Unified Memory는 예외) |

---

## ⚙️ CUDA / Metal / OpenCL에서의 SIMD 활용

### ✅ CUDA

```
__global__ void vec_add(float* A, float* B, float* C) {
    int i = threadIdx.x + blockIdx.x * blockDim.x;
    C[i] = A[i] + B[i];
}
```

- CUDA에서는 스레드 그룹 = block, 워프 = 32스레드 단위로 동작
- 내부는 SIMT 구조 (분기 있는 SIMD)

---

### ✅ Metal (Apple GPU)

```
kernel void add(device float* A, device float* B, device float* C,
                uint id [[thread_position_in_grid]]) {
    C[id] = A[id] + B[id];
}
```

- Metal에서도 **벡터화 및 SIMD 최적화 자동 적용**
- Apple GPU는 **Tile-based Rendering + SIMD pipeline**

## 📈 SIMD / GPU 최적화 포인트

| 전략 | 설명 |
|---|---|
| 데이터 정렬 | 16~64byte 경계 정렬 필요 (`aligned_alloc`) |
| 벡터 길이 패딩 | 4, 8, 16 등 고정 벡터 크기에 맞추기 |
| 분기 제거 | `if` 대신 `select` 사용, warp divergence 방지 |
| Coalesced access | GPU에서 연속 메모리 접근으로 DRAM 효율↑ |
| Loop unrolling | SIMD와 GPU 모두 성능 증가 |

## 📌 요약 정리

| 항목 | 설명 |
|---|---|
| **SIMD** | 1명령 → 여러 데이터 처리 (CPU 내부 벡터) |
| **SIMT (GPU)** | 1명령 → 32스레드 병렬 (워크그룹 내 스레드) |
| **CPU SIMD** | AVX2, NEON, SSE 등 |
| **GPU 구조** | SM, 워프, ALU, Shared Memory 구조 |
| **프로그래밍 모델** | CUDA, Metal, OpenCL, SYCL 등 |
| **최적화 기법** | 데이터 정렬, 분기 제거, 메모리 접근 패턴 개선 |