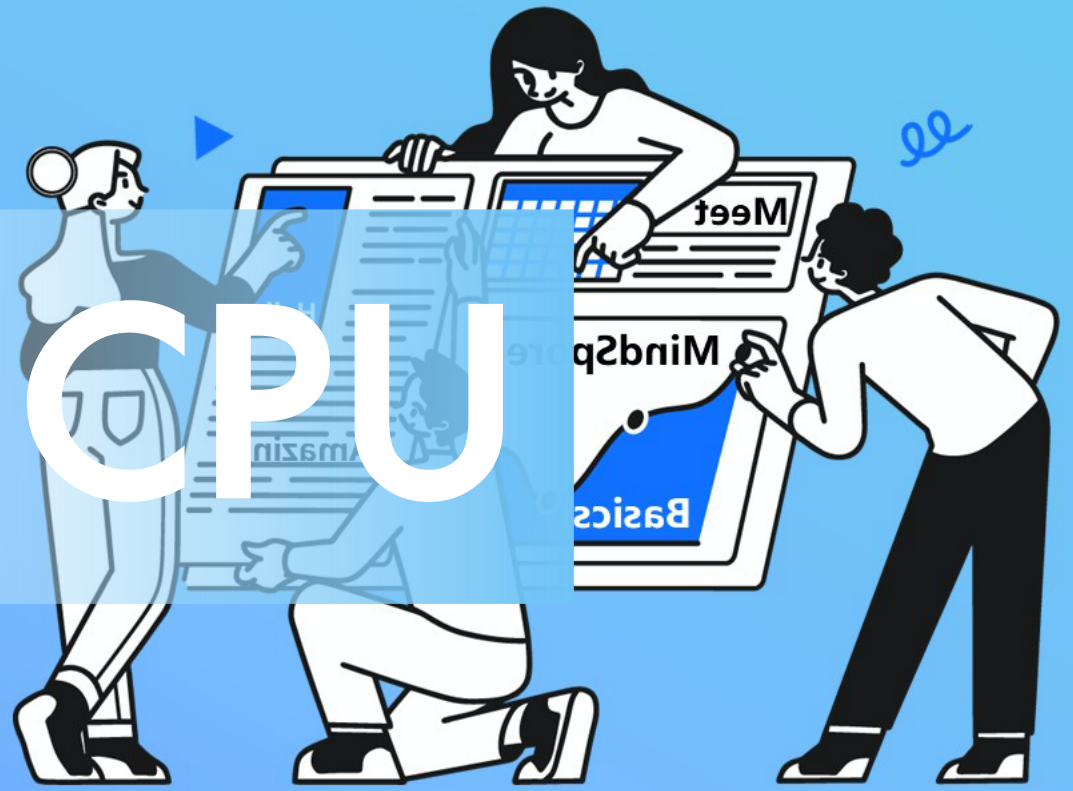
AI 芯片 – AI 芯片基础

通用处理器 CPU



ZOMI

Talk Overview

1. AI 计算体系

- 深度学习计算模式
- 计算体系与矩阵运算

2. AI 芯片基础

- 通用处理器 CPU
- 从数据看 CPU 计算
- 通用图形处理器 GPU
- AI专用处理器 NPU/TPU
- 计算体系架构的黄金10年

Talk Overview

1. 通用处理器 CPU

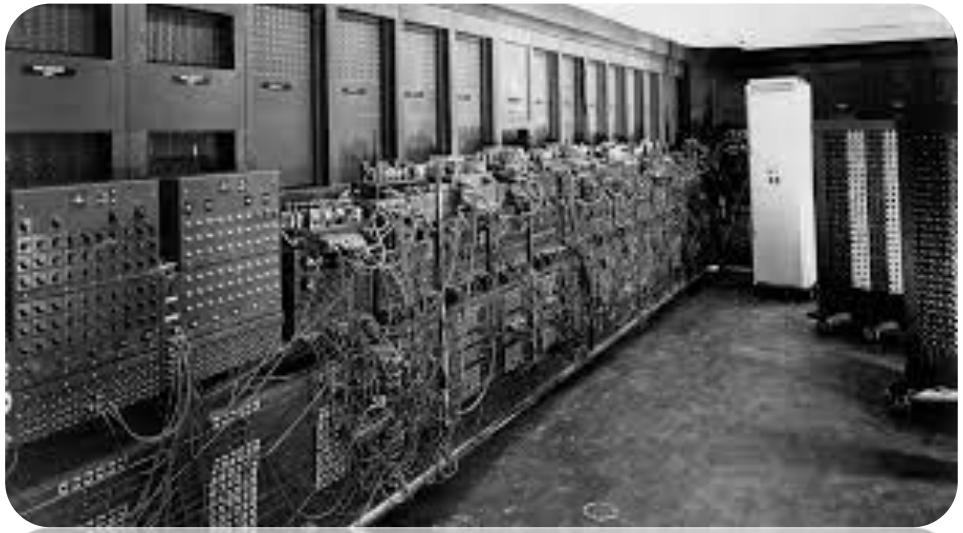
- The History – CPU 发展历史和组成
- Parallel Architecture – CPU 并行处理架构
- RoadMap of AI Chip – ISA指令集架构
- Application – CPU的应用场景

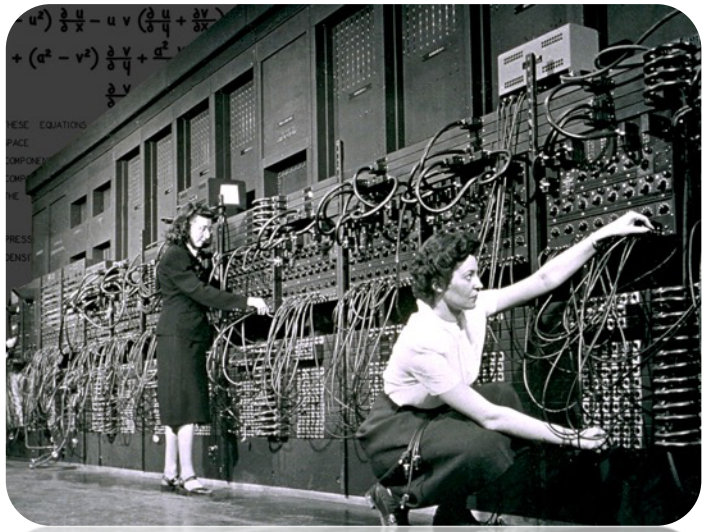
CPU

发展历史和组成

CPU 发展历史

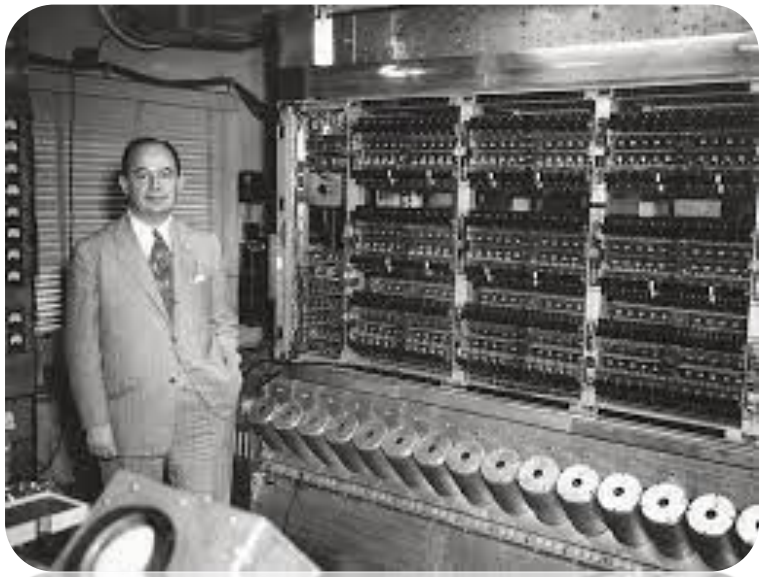
- **ENIAC**：(Electronic Numerical Integrator And Calculator，电子数字积分计算机）世界上第一台真正意义上的计算机，于1946年在美国宾夕法尼亚大学投入运行，采用十进制进行数据存储。ENIAC的发明，奠定了电子计算机的发展基础，开辟了信息时代的新纪元，是人类第三次产业革命开始的标志，具有重要的历史意义。





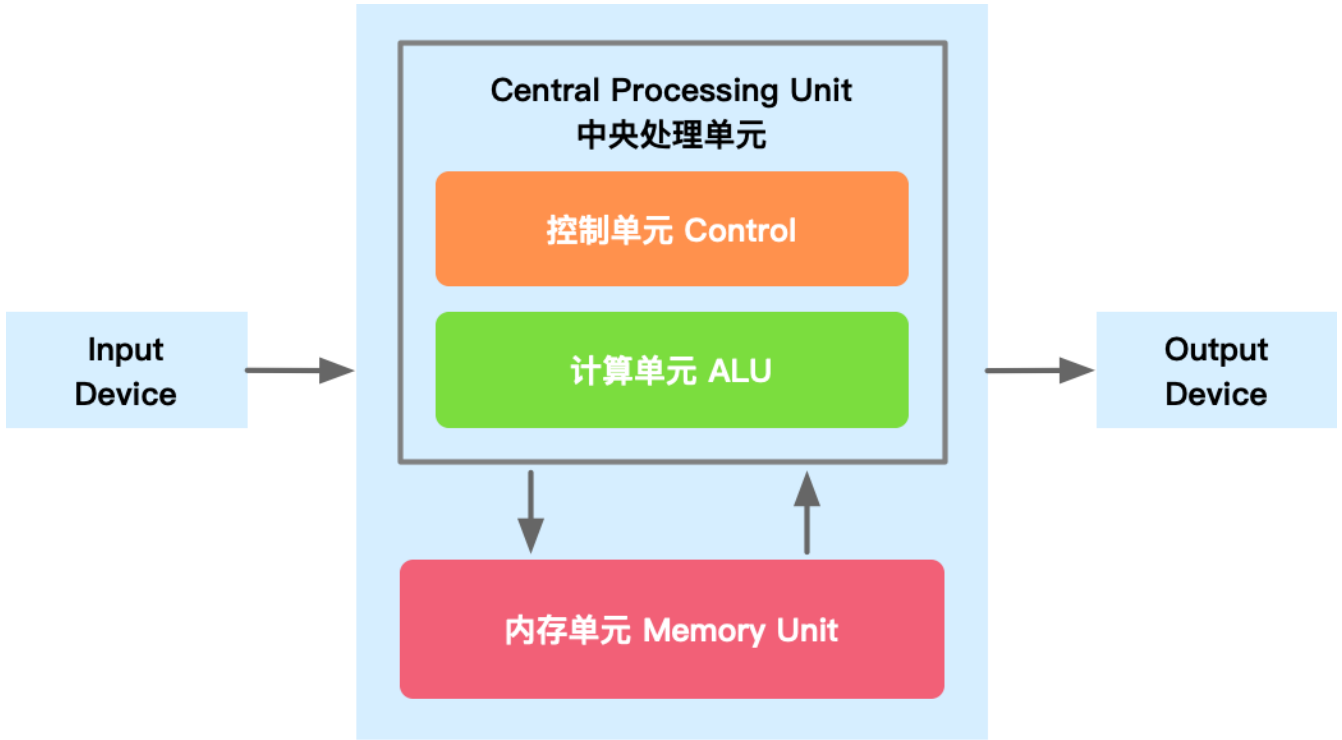
CPU 发展历史

- **EDVAC：**（Electronic Discrete Variable Automatic Computer，电子离散变量自动计算机）。世界上首次提出的第一台采用二进制的冯·诺依曼计算机，由运算器、控制器、存储器、输入和输出设备5部分组成。1945年3月由冯·诺伊曼本人与莫奇利、埃克特等提出，1951年最终完成。



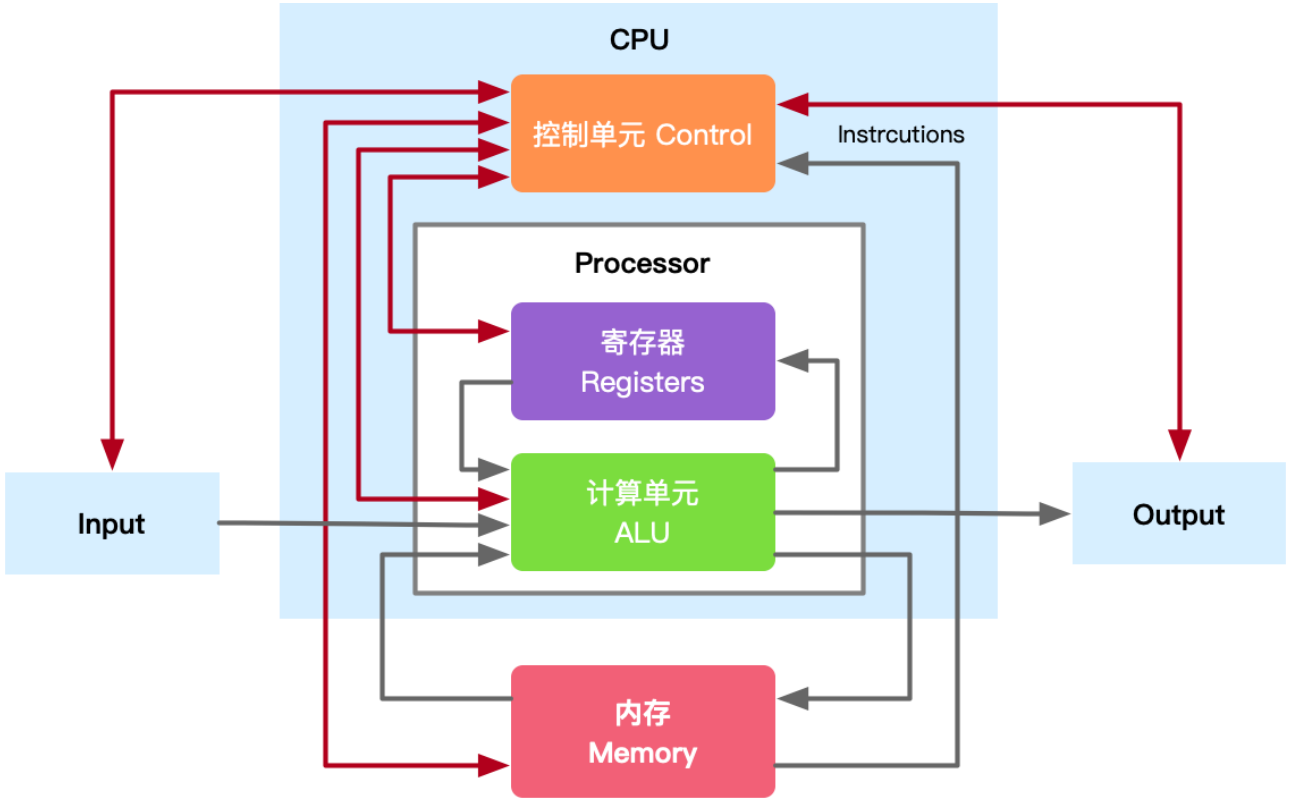
CPU 主要组成

- 从彼时至今，无论CPU的具体实现怎么变、晶体管数量翻多少番，它的构成始终由运算器、控制器和寄存器这三大部分组成。



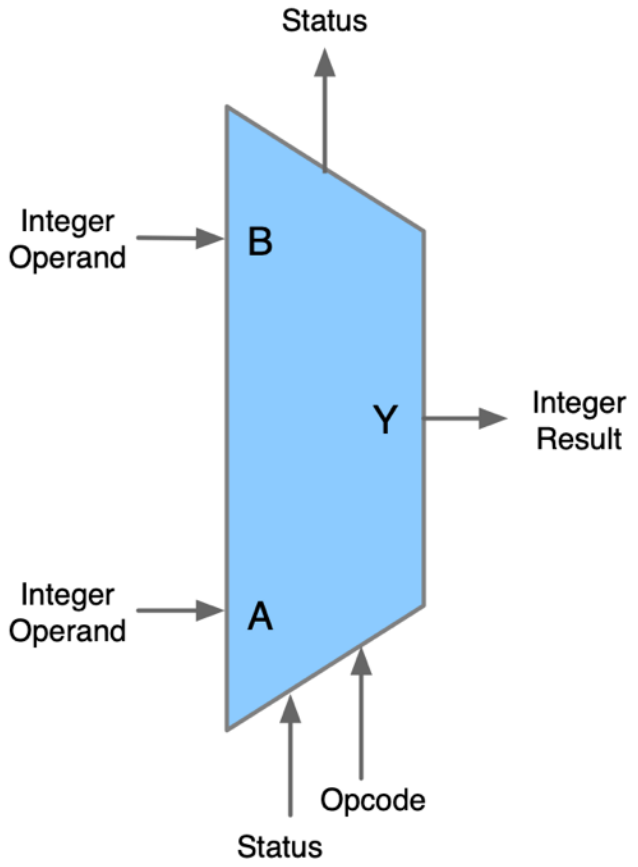
CPU 主要组成

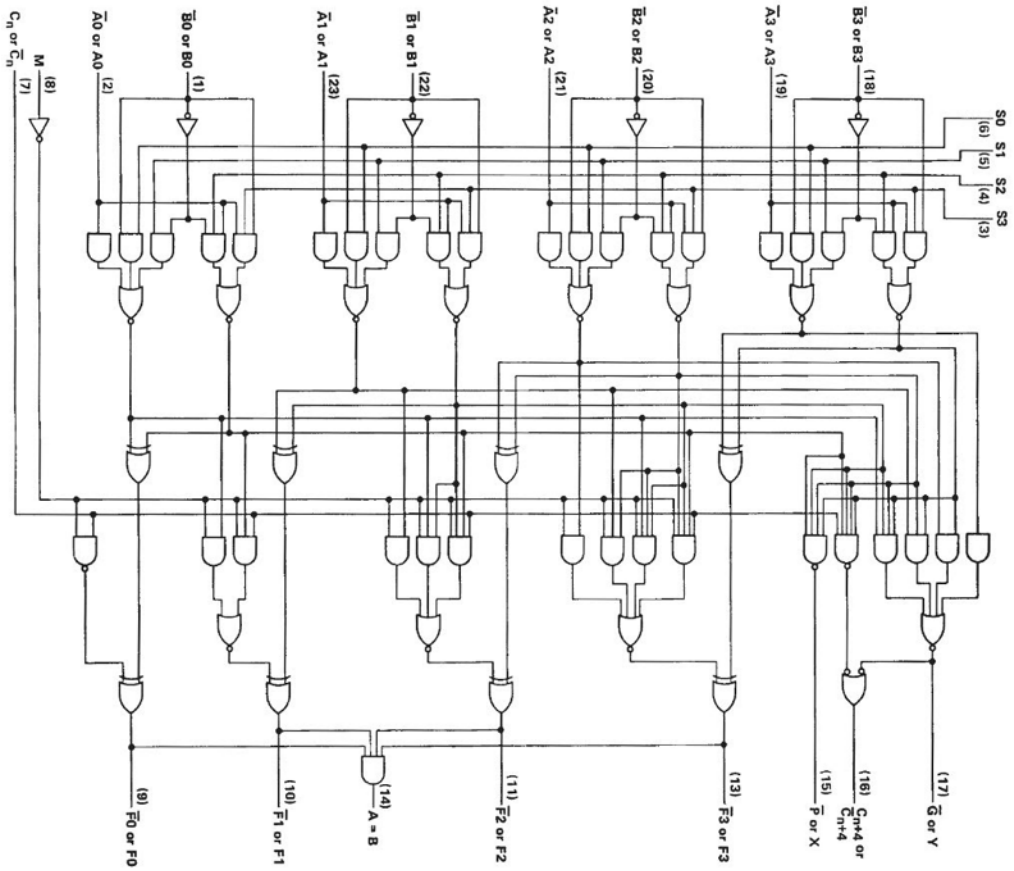
- 从彼时至今，无论CPU的具体实现怎么变、晶体管数量翻多少番，它的构成始终由运算器、控制器和寄存器这三大部分组成。



CPU 主要组成

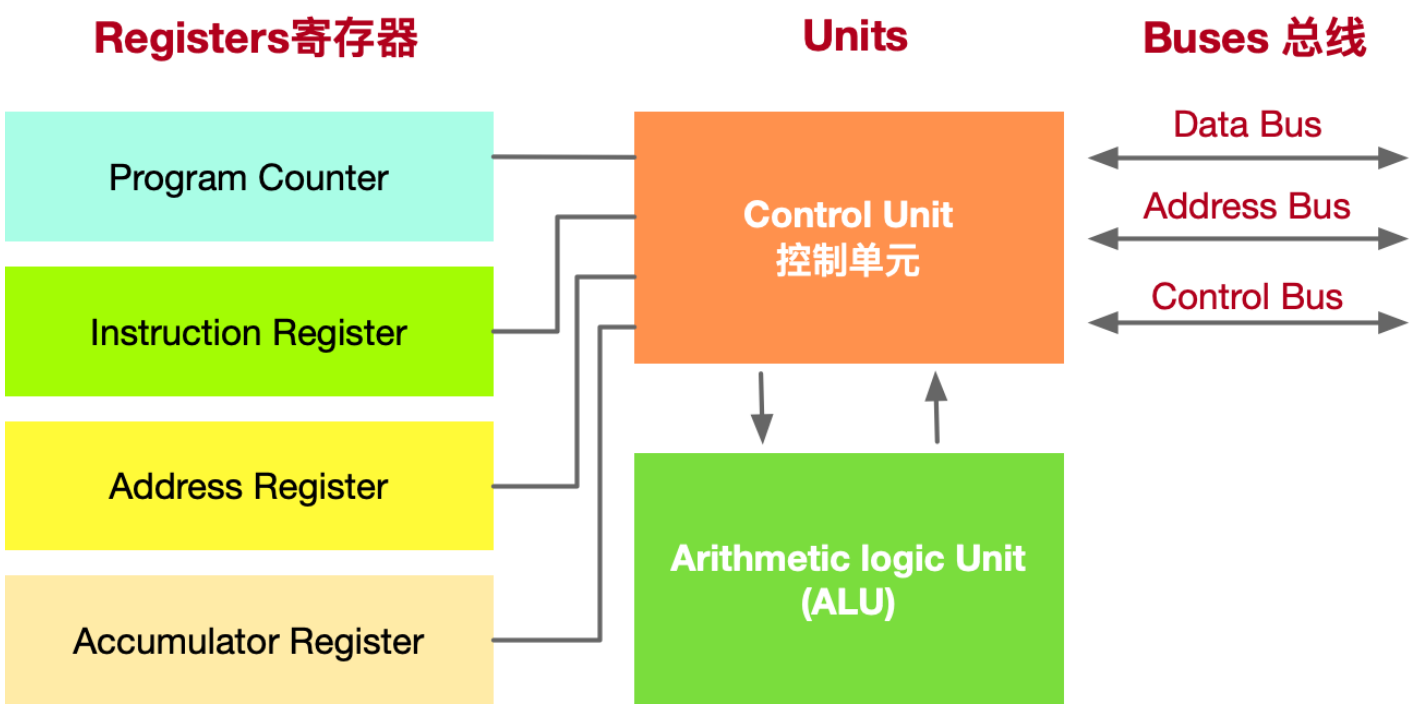
- **运算器**：也叫算术逻辑单元（ALU），负责算术运算和逻辑运算。





CPU 主要组成

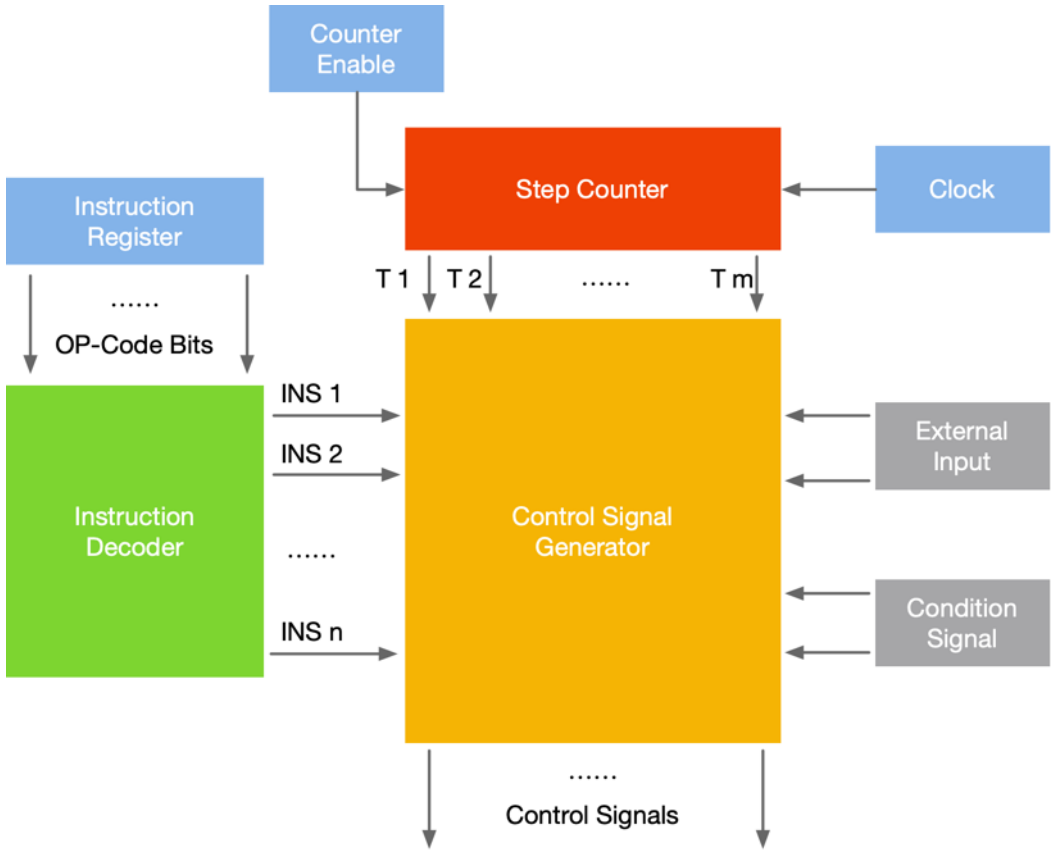
- **寄存器**：分为指令寄存器和数据寄存器，负责暂存指令、ALU所需操作数、ALU算出结果等。



Reg. Symbol	No. of Bits	Register	Function
DR	16	Data Register	Hold operand
AR	12	Address Register	Hold address for Mem
AC	16	Accumulator	Processor register
IR	16	Instruction Register	Hold instruction code
PC	12	Program Counter	Hold address of instruct
TR	16	Temp Register	Hold temp data
INPR	8	Input Register	Holds input char
OUTR	8	Output Register	Holds output char

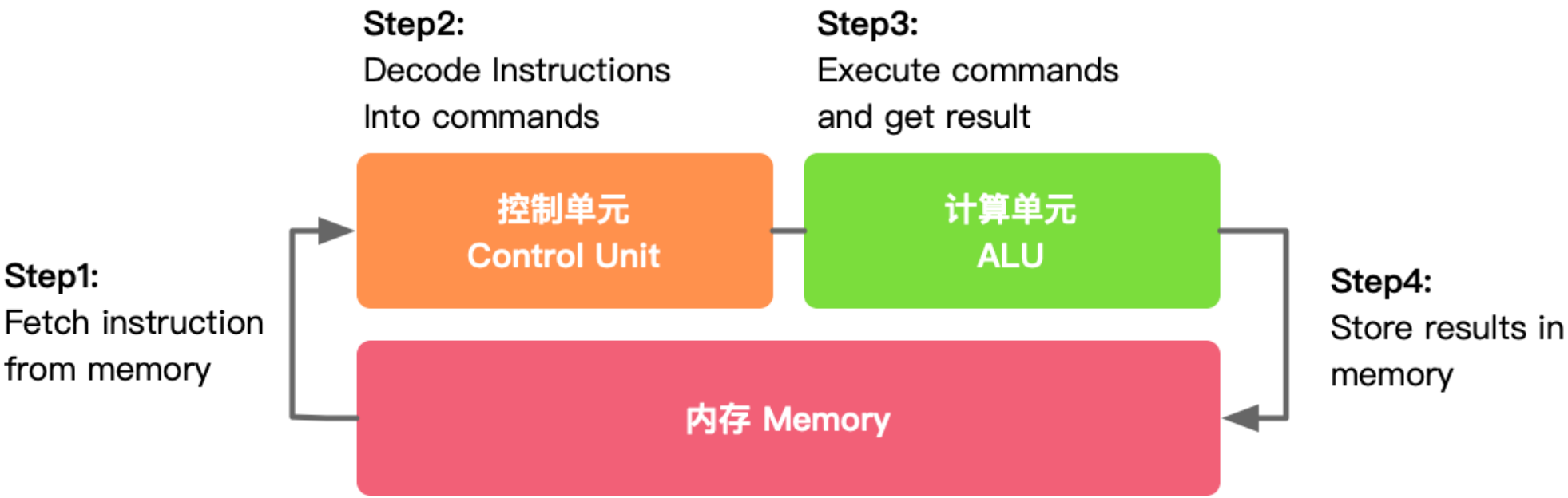
CPU 主要组成

- **控制器**：负责调度工作，包括对要执行的指令进行译码、从内存中调取数据给寄存器、向运算器和寄存器发出具体操作指令等。



CPU 工作流程

- 主要分为4步：1）从内存提取指令；2）解码；3）执行；4）写回。



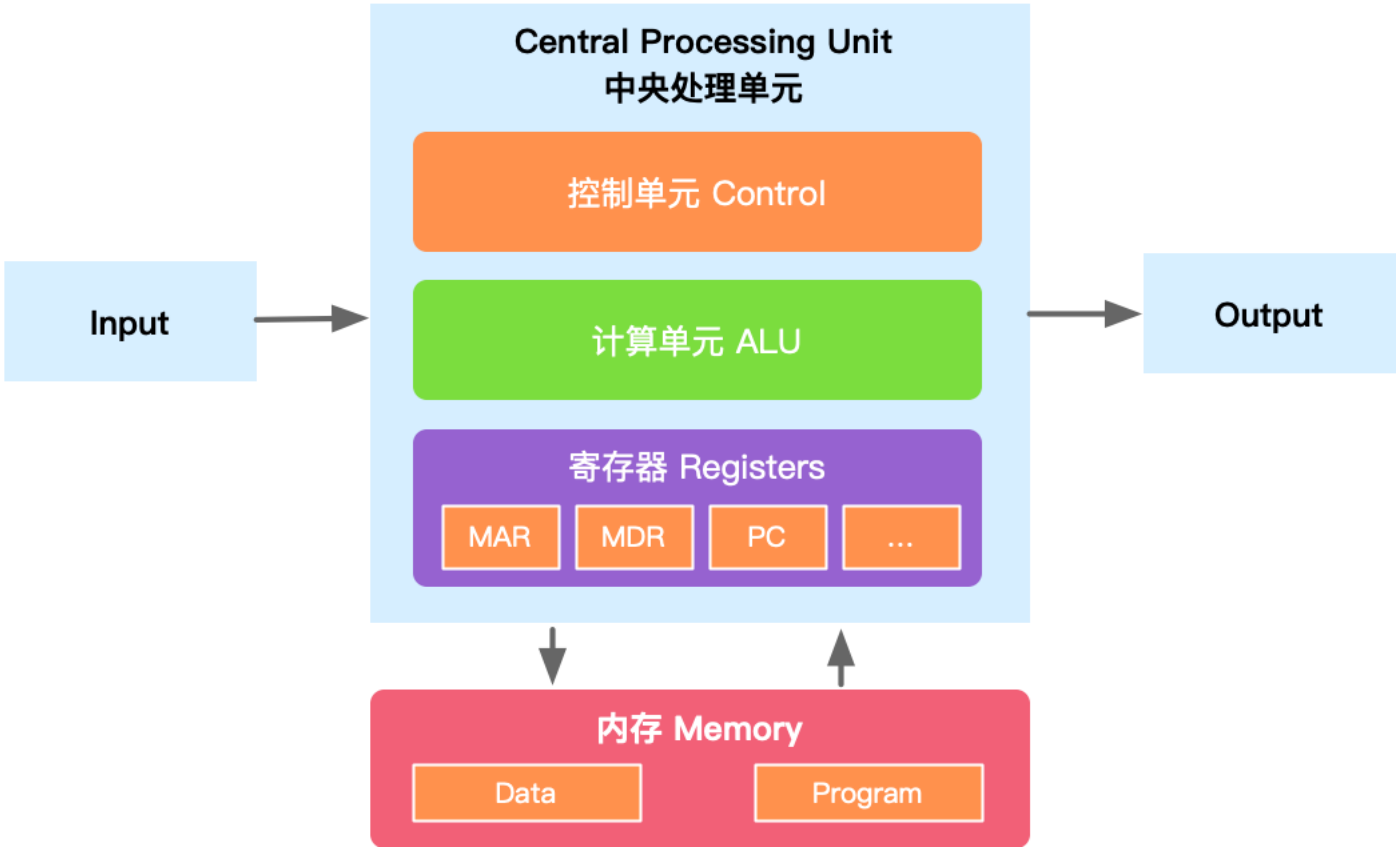
冯·诺依曼架构

- CPU三大组成的各自分工，控制器和寄存器负责的工作最多、要存的数据最多的两部分。

```
void demo(double alpha, double *x, double *y)
{
    int n = 2000;
    for(int i = 0; i < n; ++i)
    {
        y[i] = alpha * x[i] + y[i];
    }
}
```

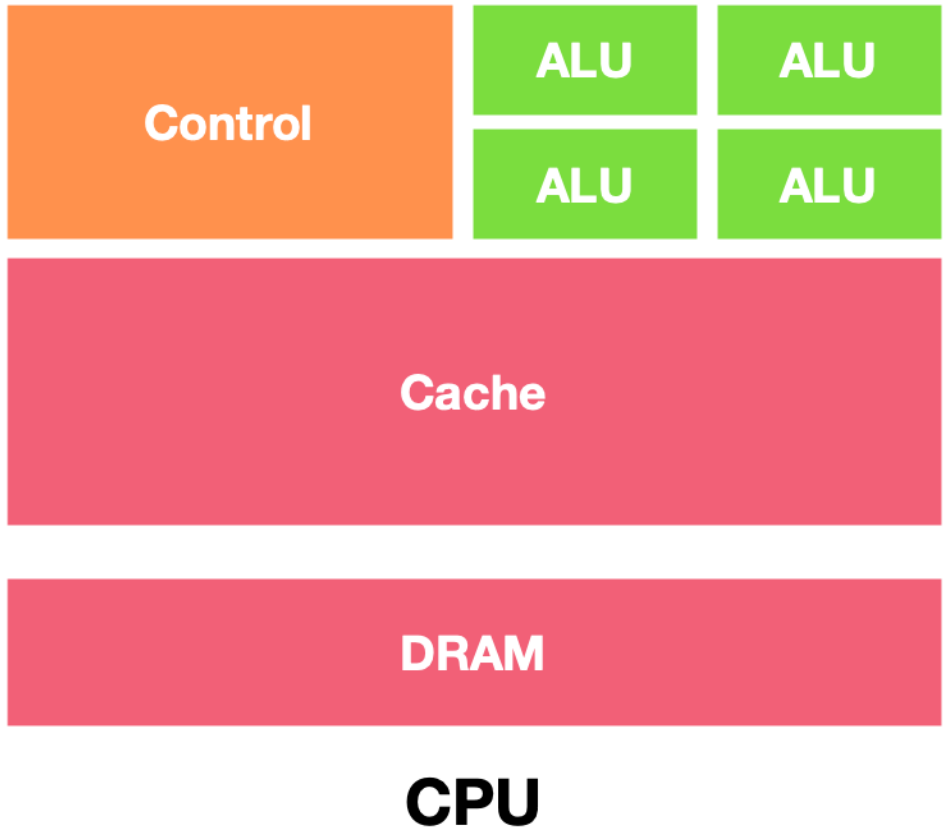
冯·诺依曼架构

- CPU三大组成的各自分工，控制器和寄存器负责的工作最多、要存的数据最多的两部分。

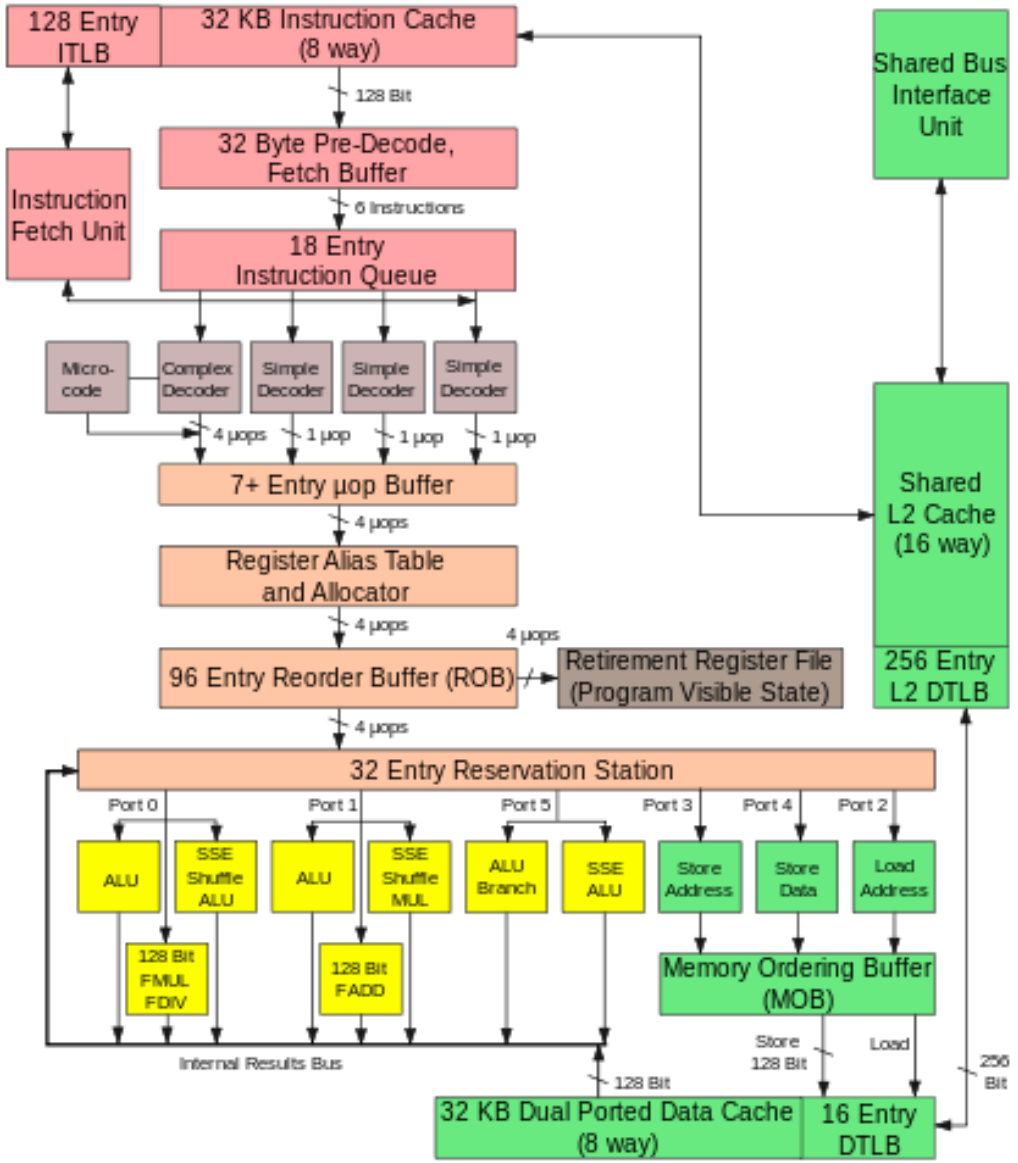


CPU 架构图

- CPU主要擅长是逻辑控制，而非计算。



现代 CPU 架构图



Intel Core 2 Architecture

约束与限制

- 实质上**ALU模块(逻辑运算单元)**是用来完成数据计算，其他各个模块的存在都是为了保证指令能够一条接一条的有序执行。这种通用性结构对于传统的编程计算模式非常适合，同时可以通过提升**CPU主频(提升单位时间内执行指令的条数)**来提升计算速度。

- 但是，依照冯·诺依曼架构针对指令的“顺序执行”的原则，CPU 只能执行完一条指令再来下一条，计算能力进一步受限。

Parallel Processing

并行处理架构

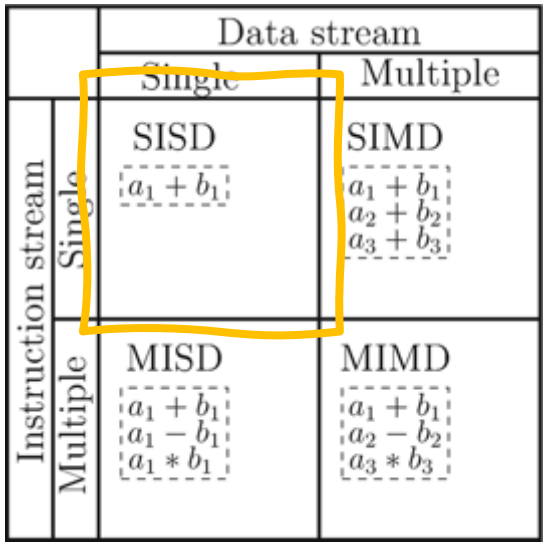
并行处理硬件架构

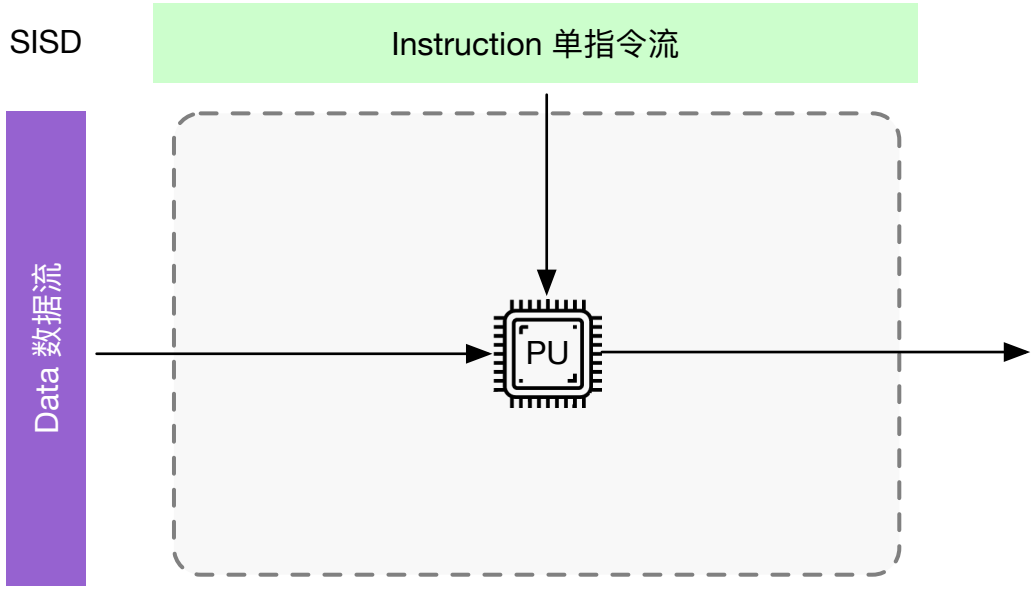
- 单指令流单数流（SISD）系统。
- 单指令流多数据流（SIMD）系统。
- 多指令流单数据流（MISD）系统。
- 多指令流多数据流（MIMD）系统。

		Data stream	
		Single	Multiple
Instruction stream	Single	SISD $a_1 + b_1$	SIMD $a_1 + b_1$ $a_2 + b_2$ $a_3 + b_3$
	Multiple	MISD $a_1 + b_1$ $a_1 - b_1$ $a_1 * b_1$	MIMD $a_1 + b_1$ $a_2 - b_2$ $a_3 * b_3$

并行计算处理硬件架构（I） ： SISD 系统

- 每个指令部件每次仅译码一条指令，而且在执行时仅为操作部件提供一份数据
- 串行计算，硬件不支持并行计算；在时钟周期内，CPU只能处理一个数据流。

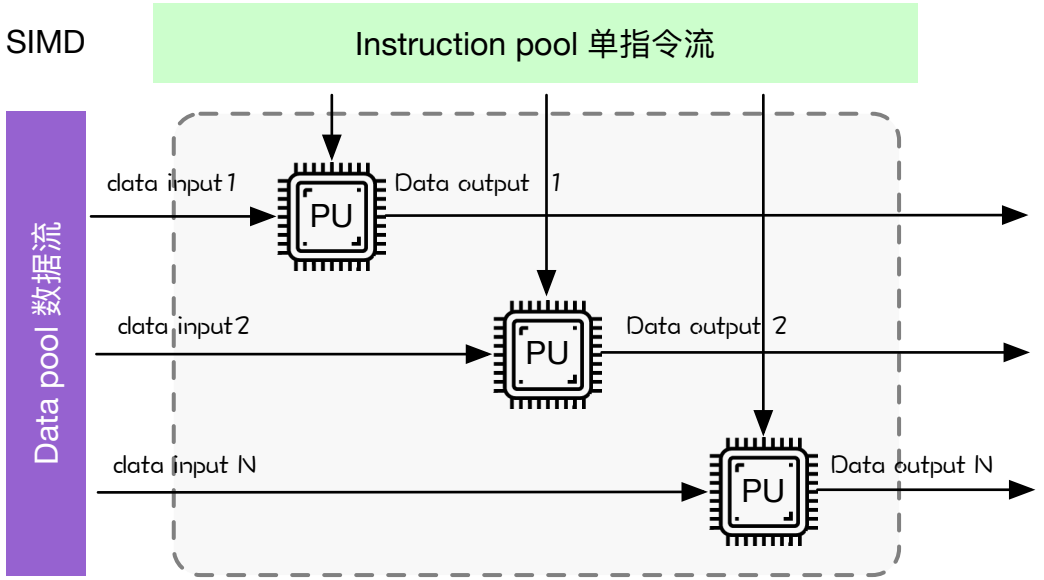




并行计算处理硬件架构（II）：SIMD 系统

- 一个控制器控制多个处理器，同时对一组数据中每一个分别执行相同操作
- SIMD主要执行向量、矩阵等数组运算，处理单元数目固定，适用于科学计算
- 特点是处理单元数量很多， 但处理单元速度受计算机通讯带宽传递速率的限制

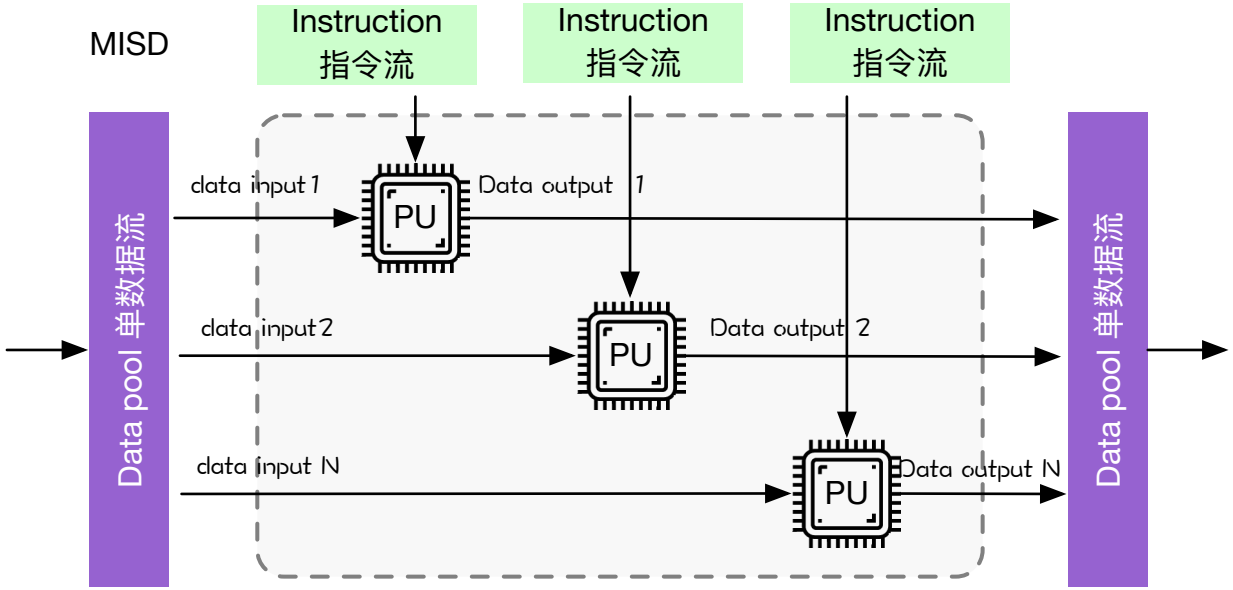
		Data stream: Single	Data stream: Multiple
Instruction stream: Single		SISD $a_1 + b_1$	SIMD $a_1 + b_1$ $a_2 + b_2$ $a_3 + b_3$
Instruction stream: Multiple		MISD $a_1 + b_1$ $a_1 - b_1$ $a_1 * b_1$	MIMD $a_1 + b_1$ $a_2 - b_2$ $a_3 * b_3$



并行计算处理硬件架构（I）：MISD 系统

- 多指令流单数据流机器，采用多个指令流来处理单个数据流
- 作为理论模型出现，没有投入到实际应用之中

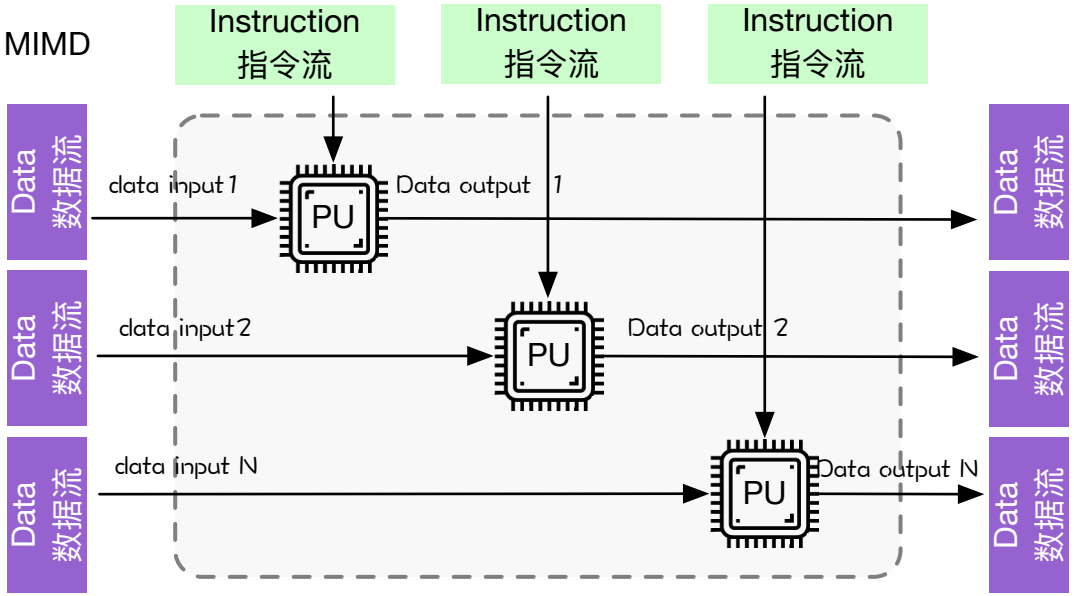
		Data stream	
		Single	Multiple
Instruction stream	Single	SISD $a_1 + b_1$	SIMD $a_1 + b_1$ $a_2 + b_2$ $a_3 + b_3$
	Multiple	MISD $a_1 + b_1$ $a_1 - b_1$ $a_1 * b_1$	MIMD $a_1 + b_1$ $a_2 - b_2$ $a_3 * b_3$



并行计算处理硬件架构（III）：MIMD 系统

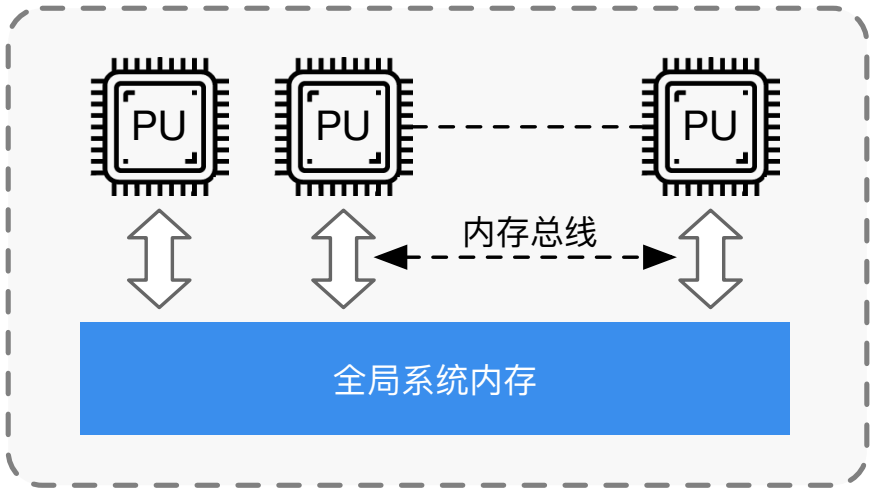
- 在多个数据集上执行多个指令的多处理器机器
- 共享内存 MIMD 和分布式内存

		Data stream	
		Single	Multiple
Instruction stream	Single	SISD $a_1 + b_1$	SIMD $a_1 + b_1$ $a_2 + b_2$ $a_3 + b_3$
	Multiple	MISD $a_1 + b_1$ $a_1 - b_1$ $a_1 * b_1$	MIMD $a_1 + b_1$ $a_2 - b_2$ $a_3 * b_3$

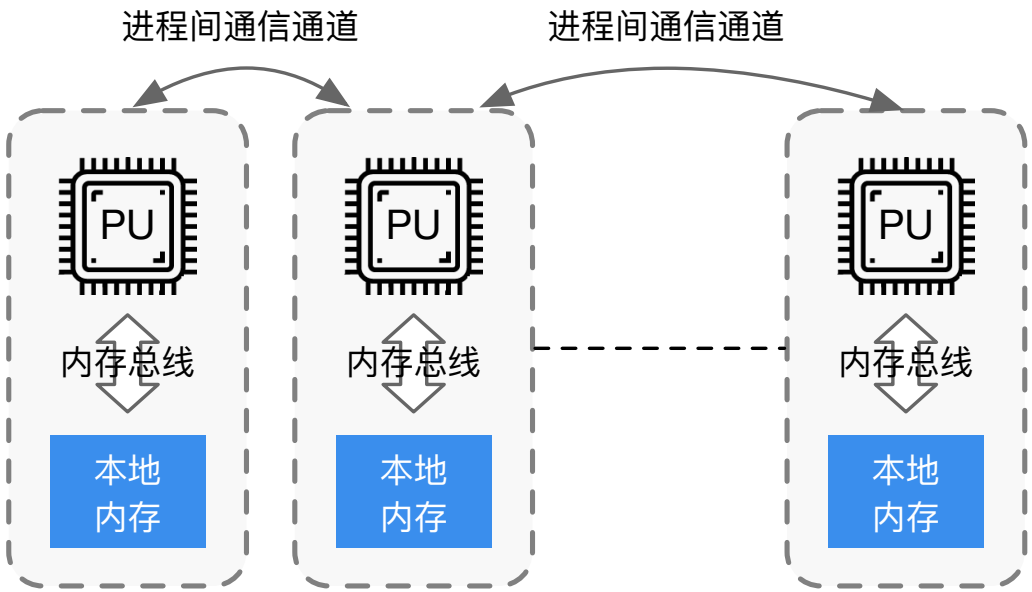


并行计算处理硬件架构（III）：MIMD 系统

- 在多个数据集上执行多个指令的多处理器机器
- 共享内存 MIMD 和分布式内存 MIMD



共享内存 MIMD

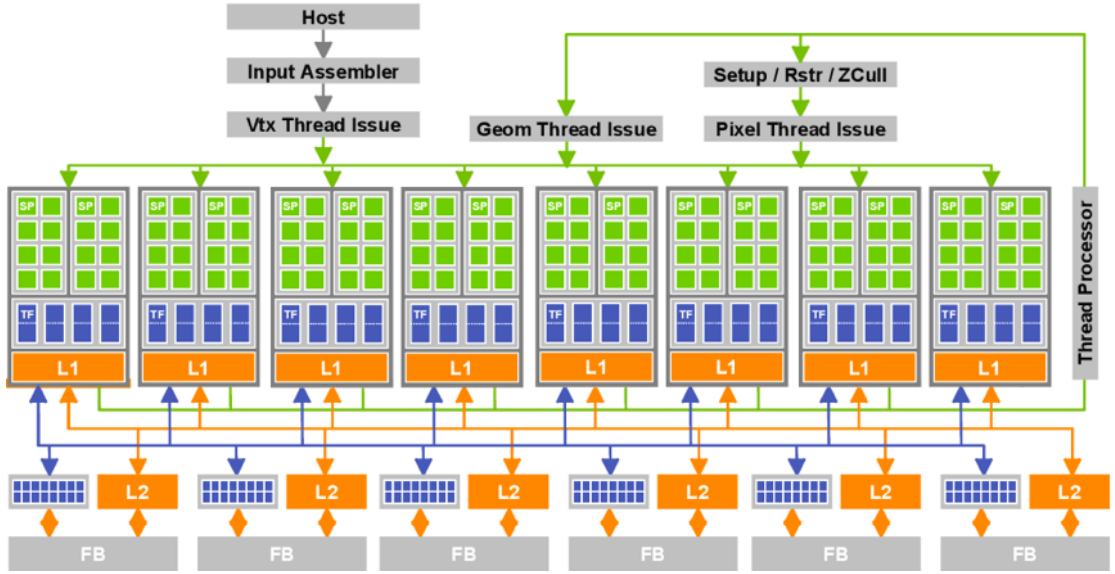


分布式内存 MIMD

并行计算处理硬件架构（IV）： SIMT 系统

- 单指令多线程，有效地管理和执行多个单线程，允许一条指令的多数据分开寻址
- 无需开发者把数据凑成合适的矢量长度，并且SIMT允许每个线程有不同的分支
- 条件跳转会根据输入数据不同在不同的线程中有不同表现

Instruction stream	Data stream: Single	Data stream: Multiple
Single	SISD $a_1 + b_1$	SIMD $a_1 + b_1$ $a_2 + b_2$ $a_3 + b_3$
Multiple	MISD $a_1 + b_1$ $a_1 - b_1$ $a_1 * b_1$	MIMD $a_1 + b_1$ $a_2 - b_2$ $a_3 * b_3$



引用

1. Processor Architecture Design Practices: survey & Issues