指令集架构
Instruction Set Architecture (ISA)

刘祎禹

2023.6.19

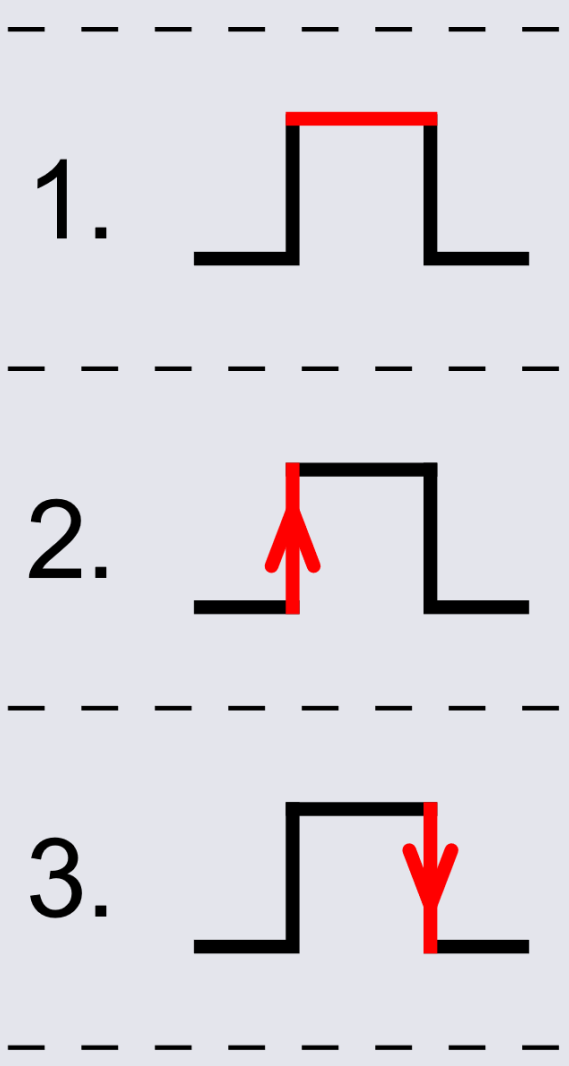
硬件

硬件 – Registers & Wires

- 寄存器 (Registers) 和线 (Wire)
- 寄存器用于储存状态
- 线本身无状态，用于连接元件

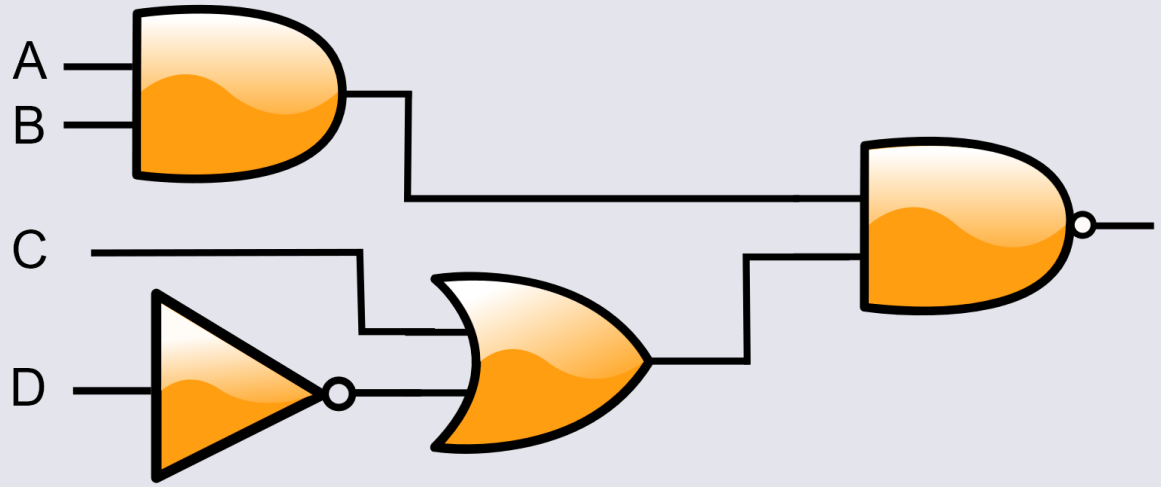
硬件 – 时钟

- 每个时钟周期都会执行一些操作
- 高电平、低电平
- 上升沿、下降沿



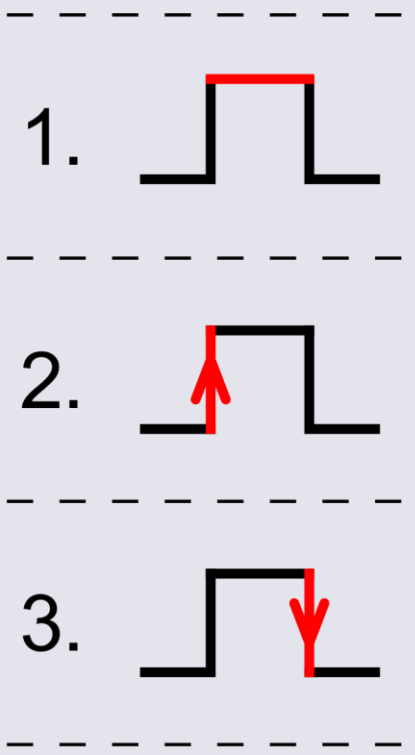
硬件 – 逻辑电路

- 门电路
- 硬件搭线
- 软件设计
  - Verilog
  - …



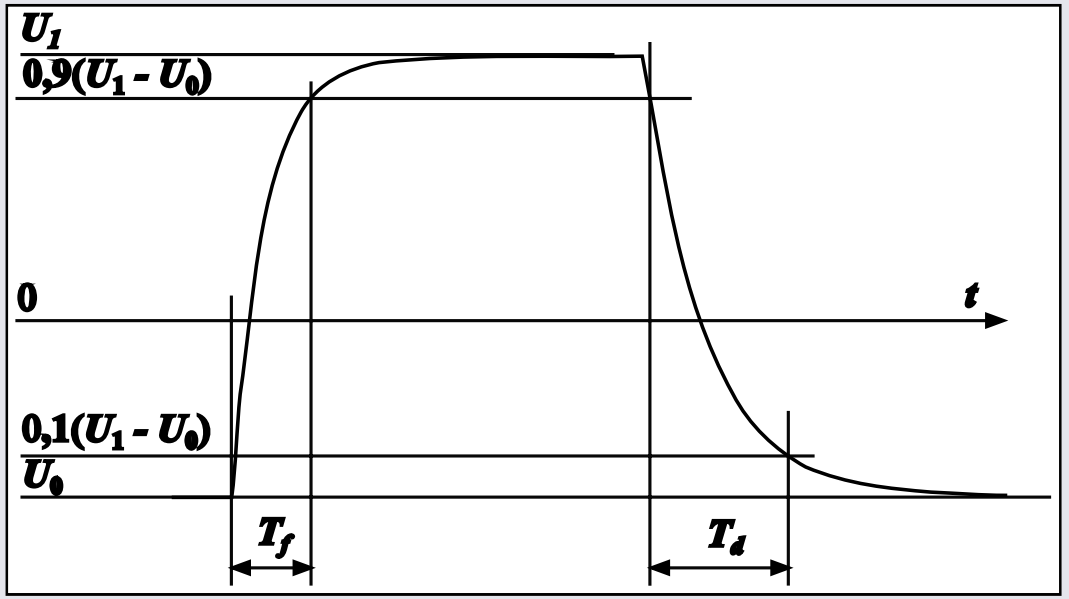
硬件 – 组合与时序

- 将硬件拆分成多个模块（模块也可组合成大模块）
- 每个模块有输入和输出
- 有组合和时序之分
- 组合逻辑：将逻辑电路组合起来
- 时序逻辑：当时钟信号的上升沿或下降沿触发
  - 严格遵循上一状态决定下一状态
  - 通常用于寄存器的更新



硬件 – 组合

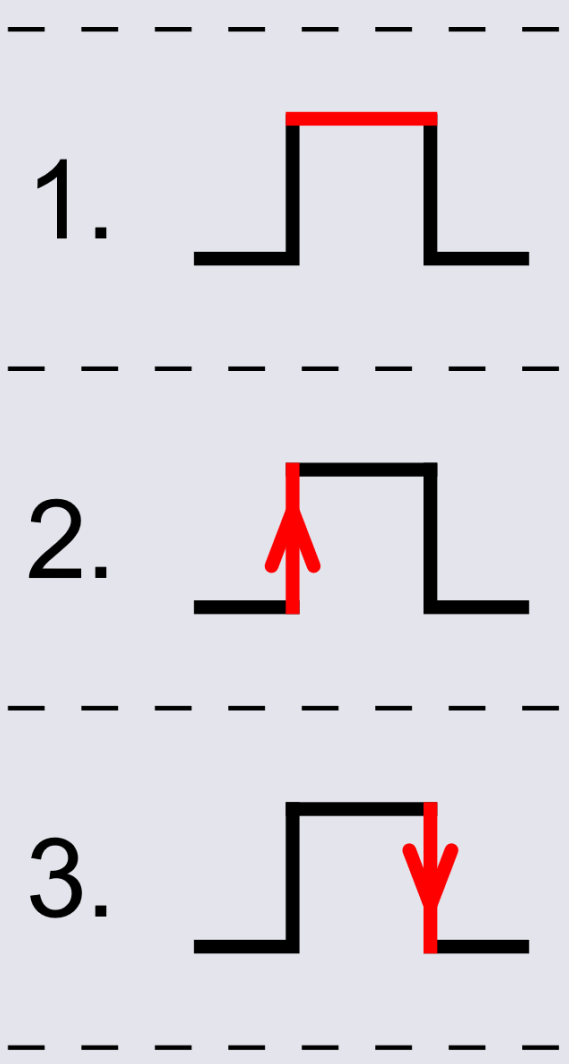
- 组合逻辑电路的时延较大
- 组合逻辑由多个小逻辑电路组成
- 小逻辑电路的时延会叠加



如何解决组合逻辑时延过大的问题?

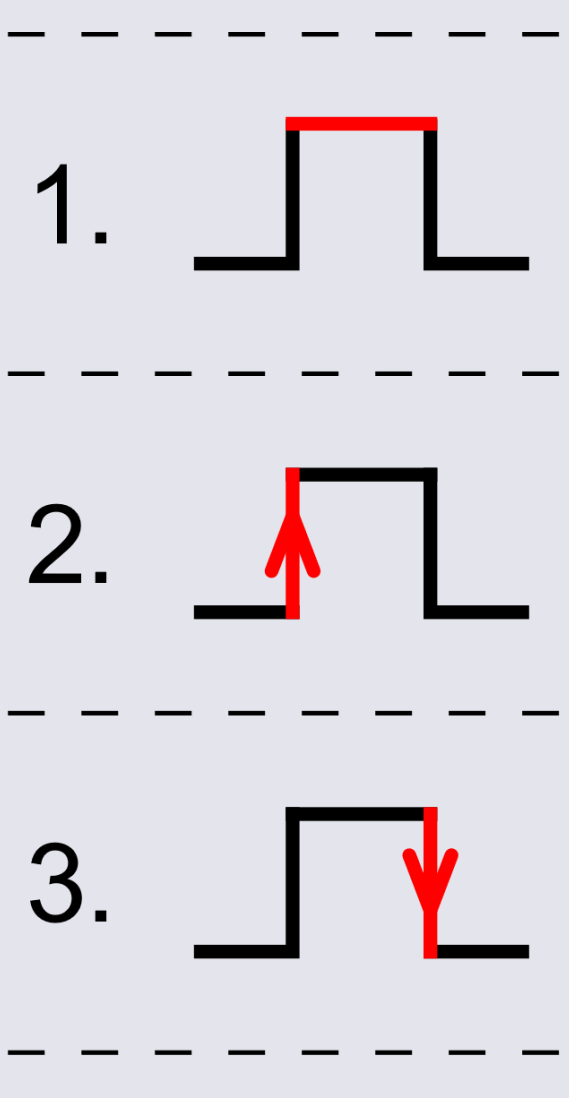
硬件 – 时序

- 将组合的电路划分成多个时钟周期
- 新的状态严格由上一状态决定



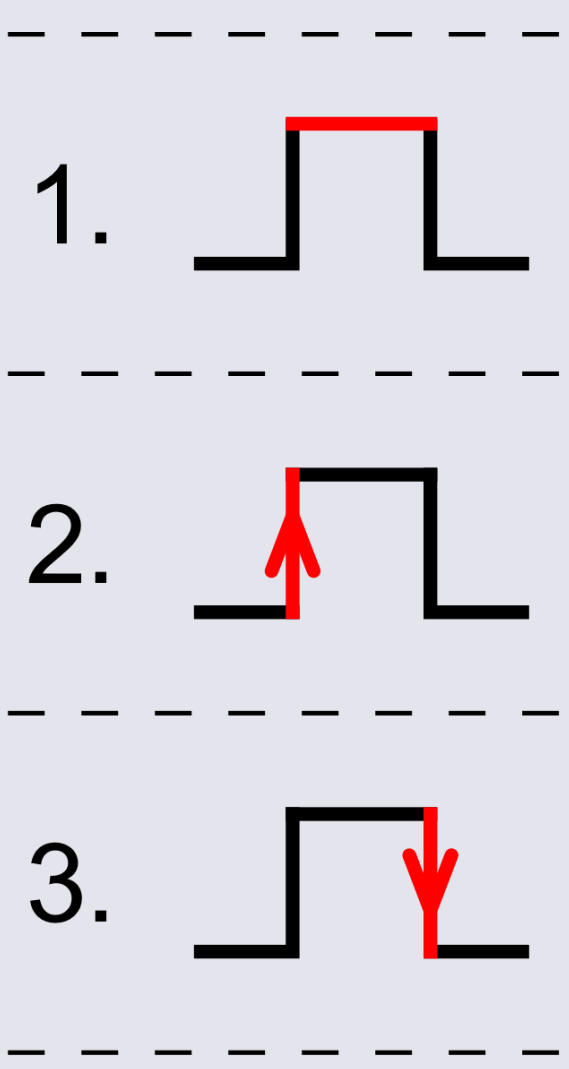
硬件 – 时序

- 例子
- 时序电路：在时钟上升沿执行
- Register a = b
- Register b = a
- 如果执行前 a = 1, b = 2
- 执行后 a, b 的值是多少?



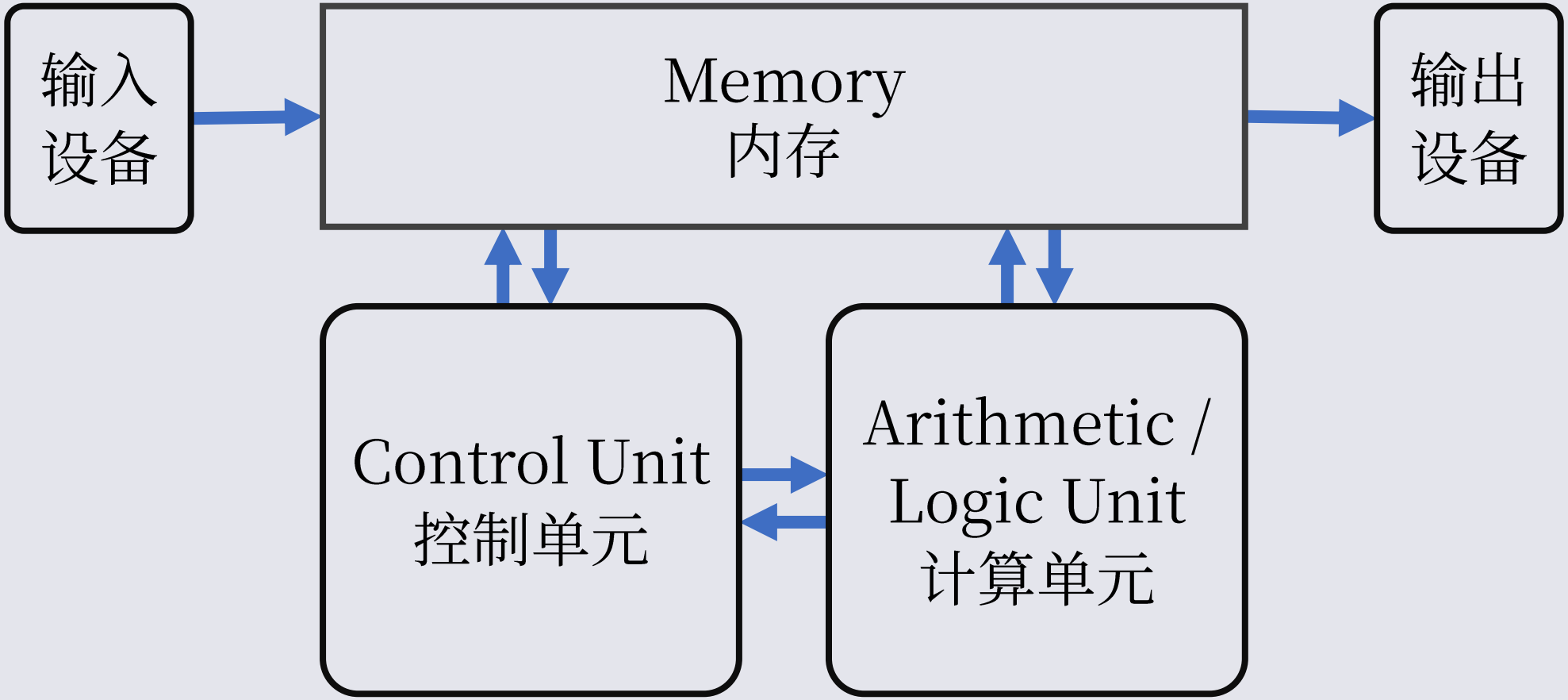
硬件 – 时序

- 将组合的电路划分成多个时钟周期
- 新的状态严格由上一状态决定
- 通常用于寄存器的更新

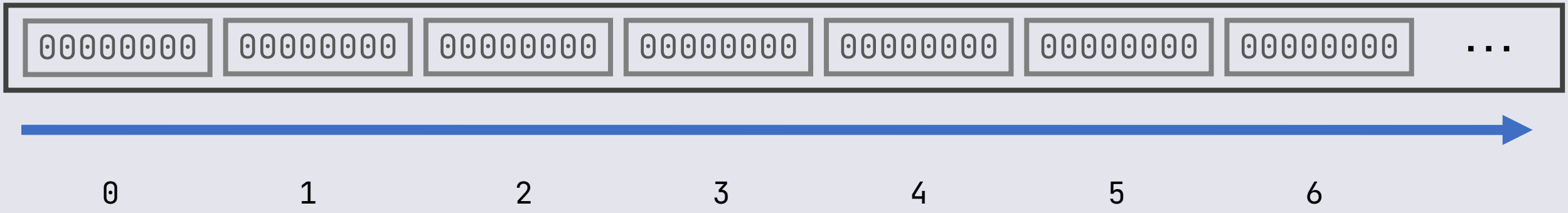


冯诺依曼架构

冯诺依曼 (Von Neumann) 架构



- Memory: 内存，用于储存信息
  - 数据
  - 程序（指令）
- 地址——确定信息所在的地方



Memory

```
0x100: 10000011 00100000
0x102: 00000000 01110000
0x104: 10000011 01000000
0x106: 01000000 01110000
0x108: 10000011 10000001
0x10A: 00100000 00000000
0x10C: 00100011 10100100
0x10E: 00000001 01110000
                ...
0x700: 01011101 11101101
0x702: 00110011 00000001
0x704: 10010001 11100101
0x706: 00100101 00000001
0x708: 11111111 11111111
0x70C: 11111111 11111111
0x70E: 11111111 11111111
0x710: 11111111 11111111
0x712: 11111111 11111111
0x714: 11111111 11111111
```

Chip

ALU

- ALU (arithmetic/logic unit)
- 算术逻辑单元
- 通常用于加减乘除和逻辑运算

Memory

```
0x100: 10000011 00100000
0x102: 00000000 01110000
0x104: 10000011 01000000
0x106: 01000000 01110000
0x108: 10000011 10000001
0x10A: 00100000 00000000
0x10C: 00100011 10100100
0x10E: 00000001 01110000
                ...
0x700: 01011101 11101101
0x702: 00110011 00000001
0x704: 10010001 11100101
0x706: 00100101 00000001
0x708: 11111111 11111111
0x70C: 11111111 11111111
0x70E: 11111111 11111111
0x710: 11111111 11111111
0x712: 11111111 11111111
0x714: 11111111 11111111
```

Chip

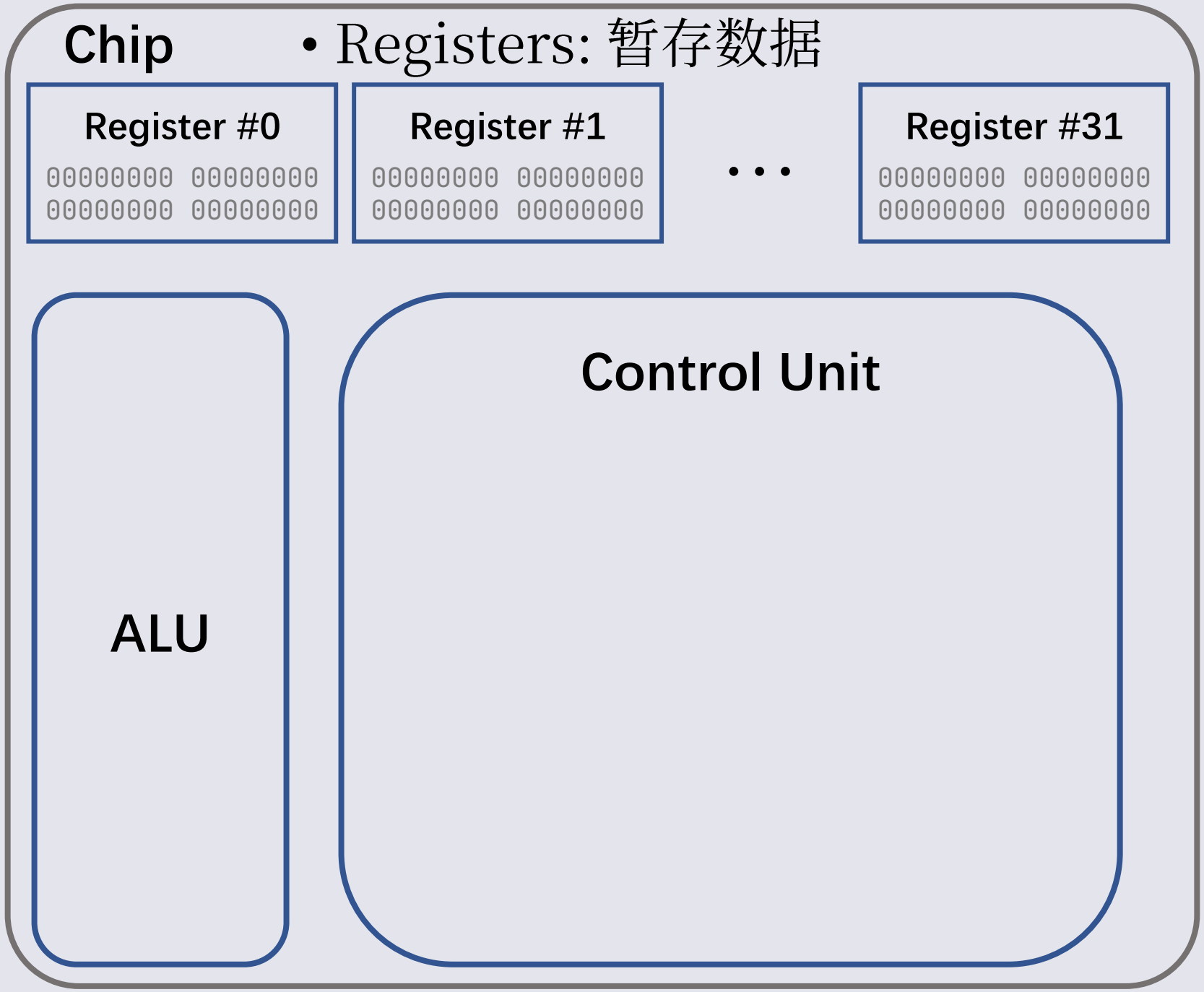
- Control Unit：控制单元

ALU

Control Unit

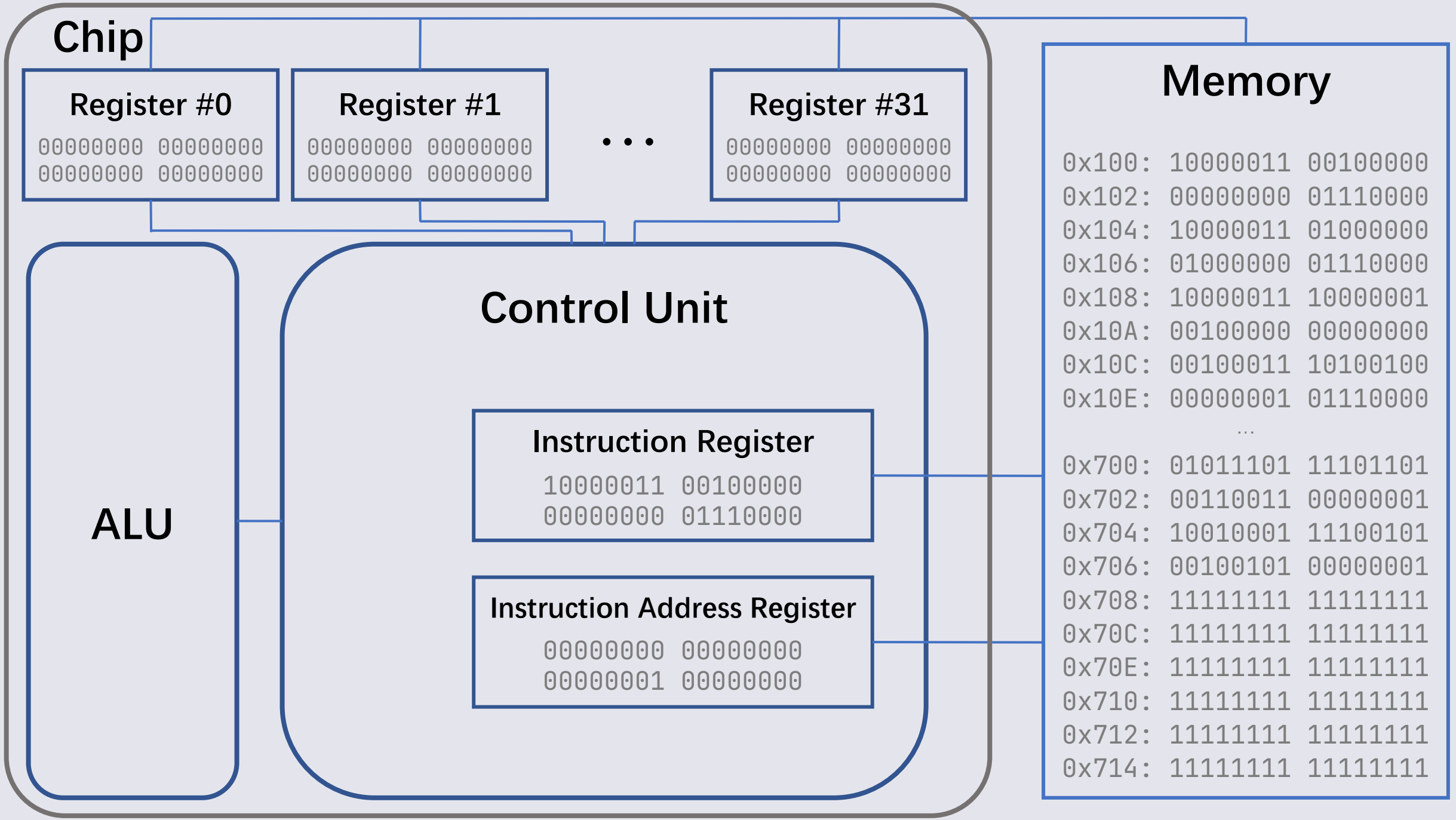
Memory

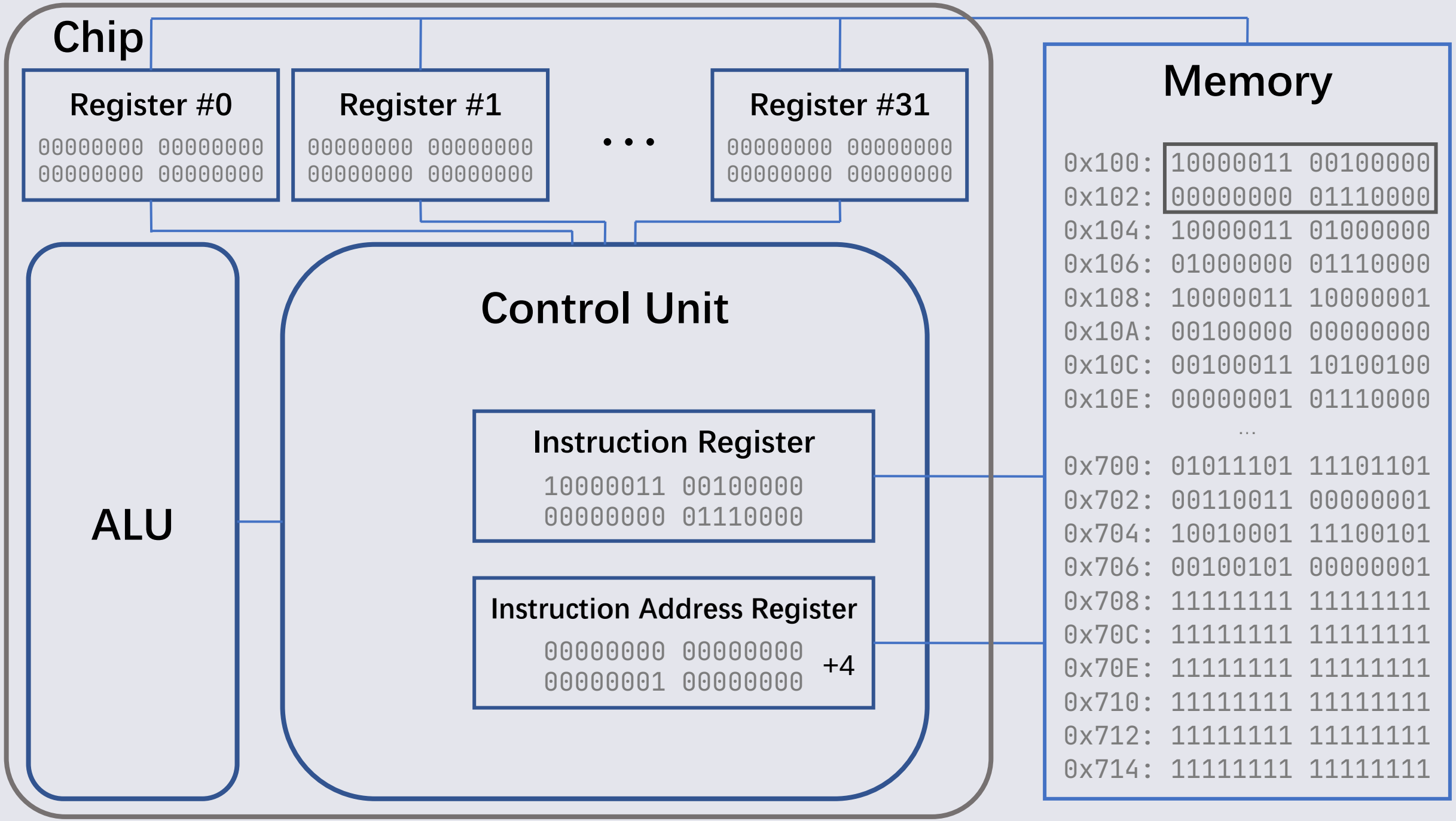
```
0x100: 10000011 00100000
0x102: 00000000 01110000
0x104: 10000011 01000000
0x106: 01000000 01110000
0x108: 10000011 10000001
0x10A: 00100000 00000000
0x10C: 00100011 10100100
0x10E: 00000001 01110000
                ...
0x700: 01011101 11101101
0x702: 00110011 00000001
0x704: 10010001 11100101
0x706: 00100101 00000001
0x708: 11111111 11111111
0x70C: 11111111 11111111
0x70E: 11111111 11111111
0x710: 11111111 11111111
0x712: 11111111 11111111
0x714: 11111111 11111111
```

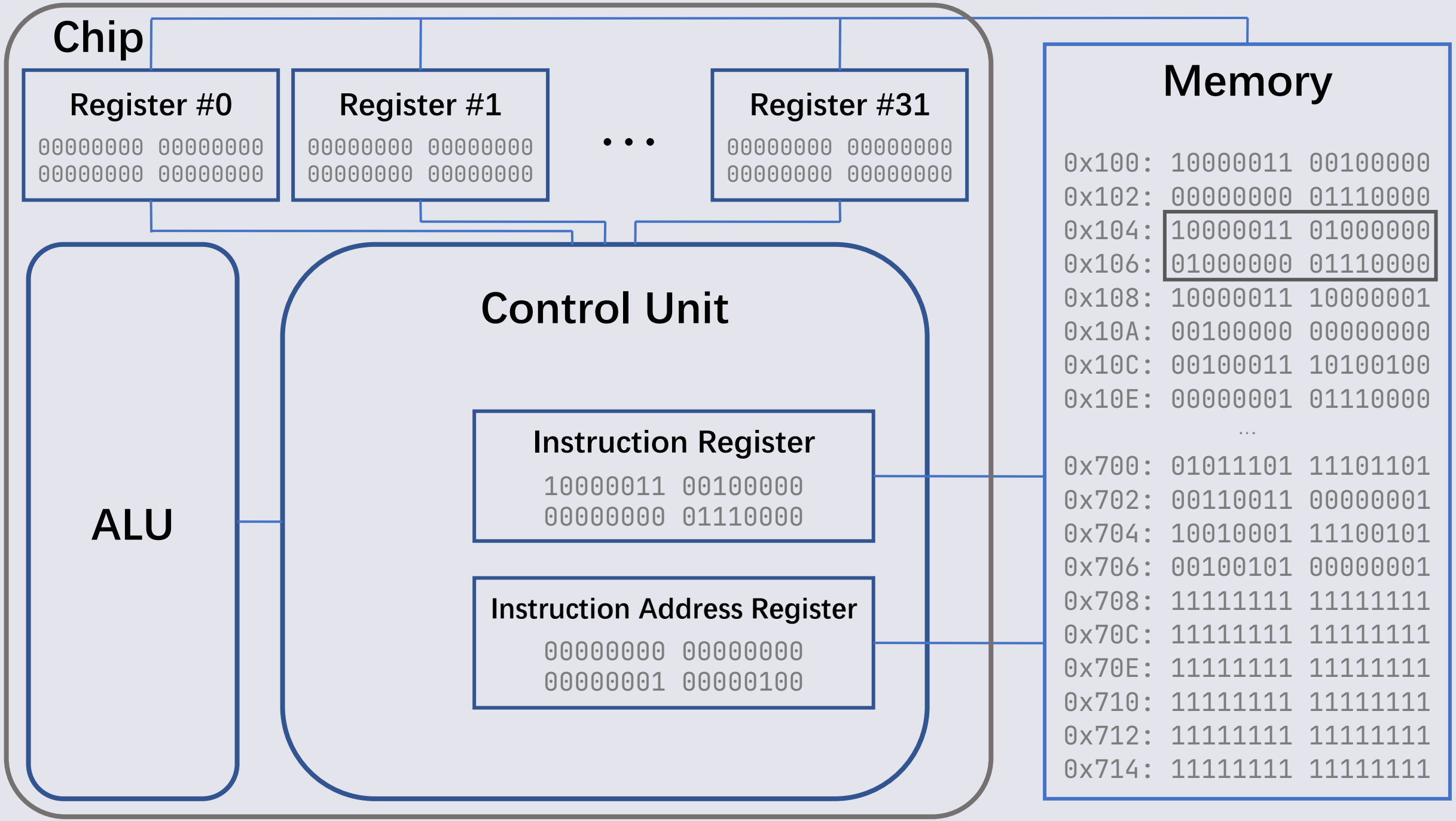


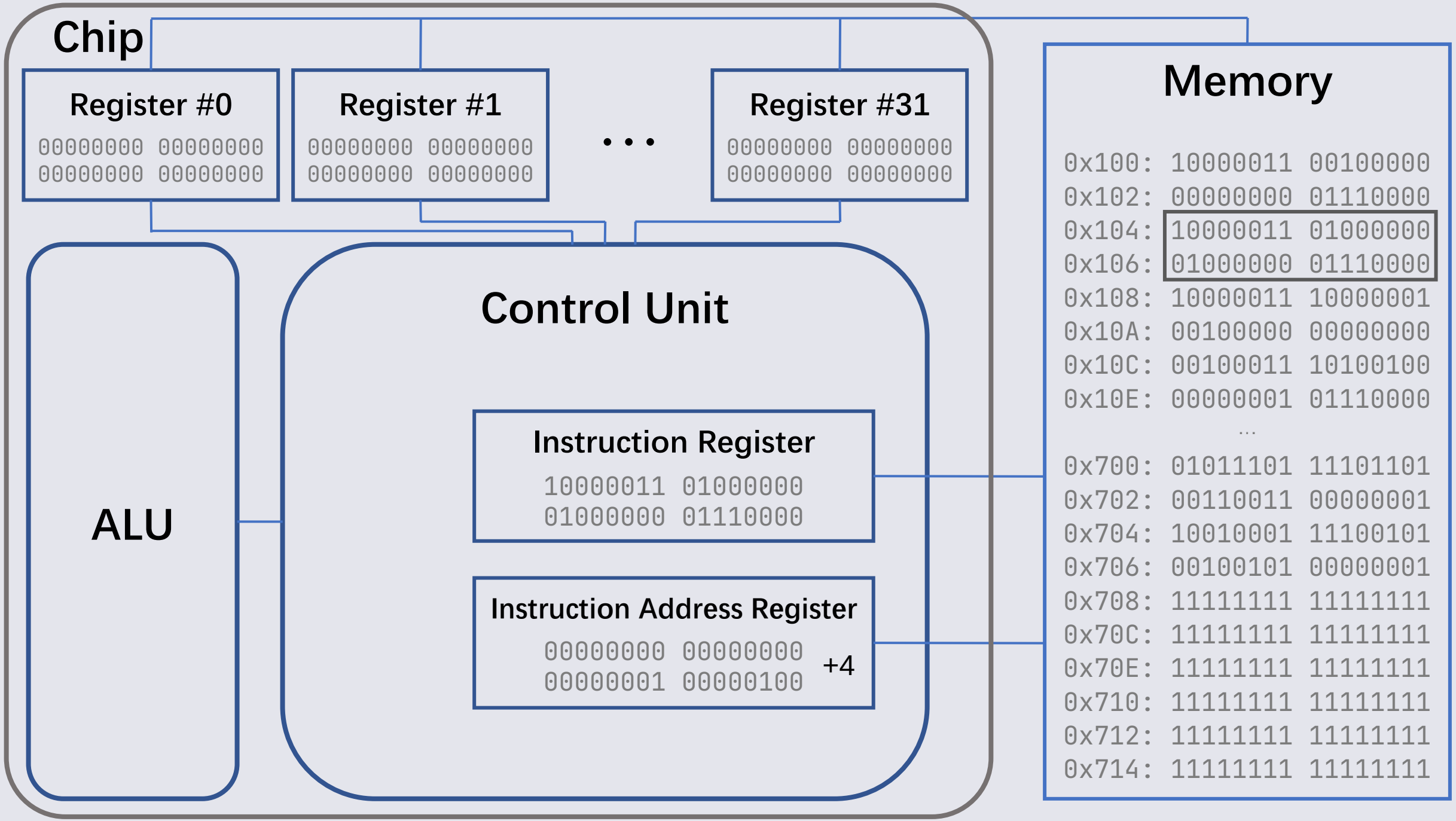
Memory

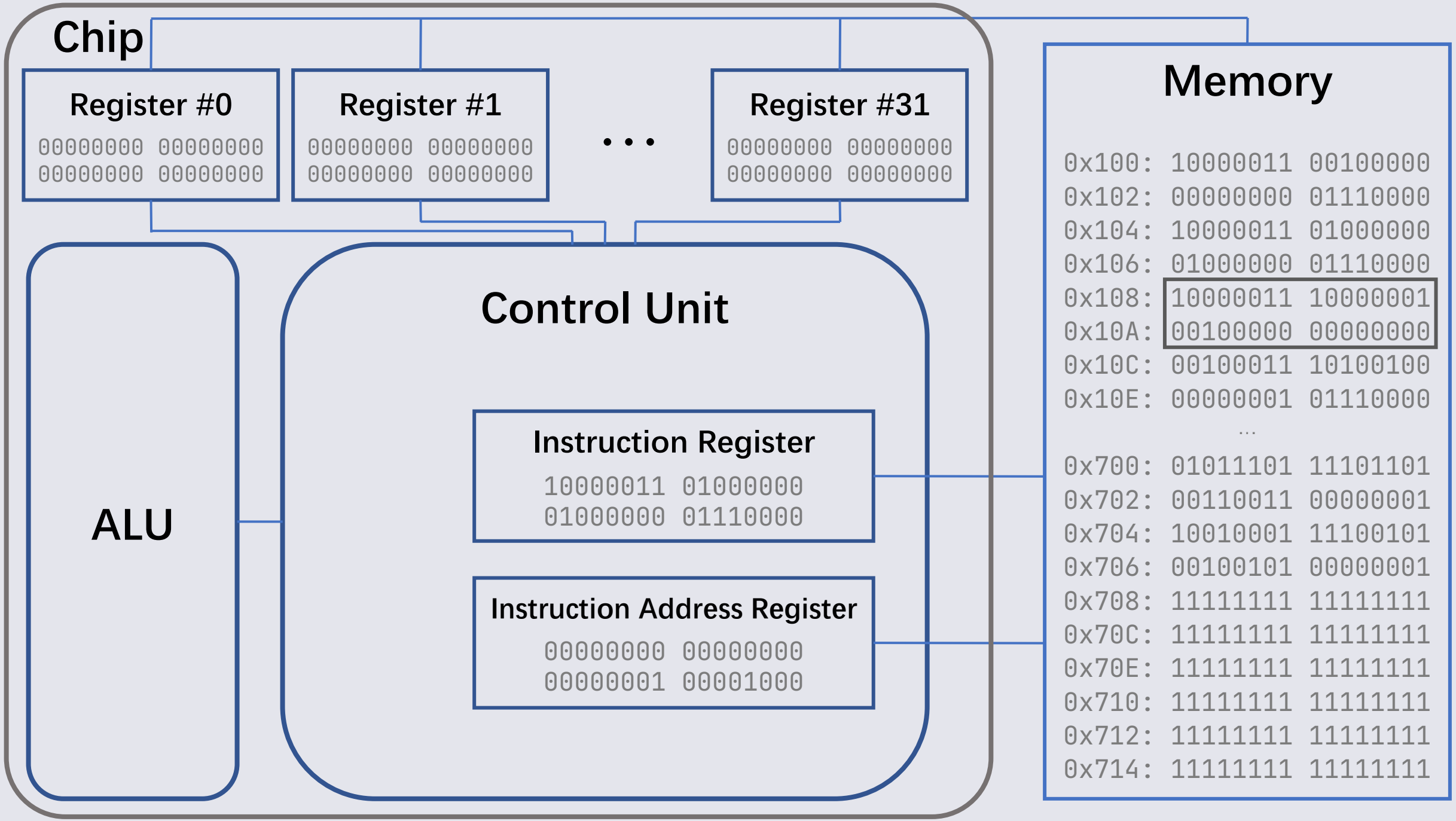
```
0x100: 10000011 00100000
0x102: 00000000 01110000
0x104: 10000011 01000000
0x106: 01000000 01110000
0x108: 10000011 10000001
0x10A: 00100000 00000000
0x10C: 00100011 10100100
0x10E: 00000001 01110000
                ...
0x700: 01011101 11101101
0x702: 00110011 00000001
0x704: 10010001 11100101
0x706: 00100101 00000001
0x708: 11111111 11111111
0x70C: 11111111 11111111
0x70E: 11111111 11111111
0x710: 11111111 11111111
0x712: 11111111 11111111
0x714: 11111111 11111111
```

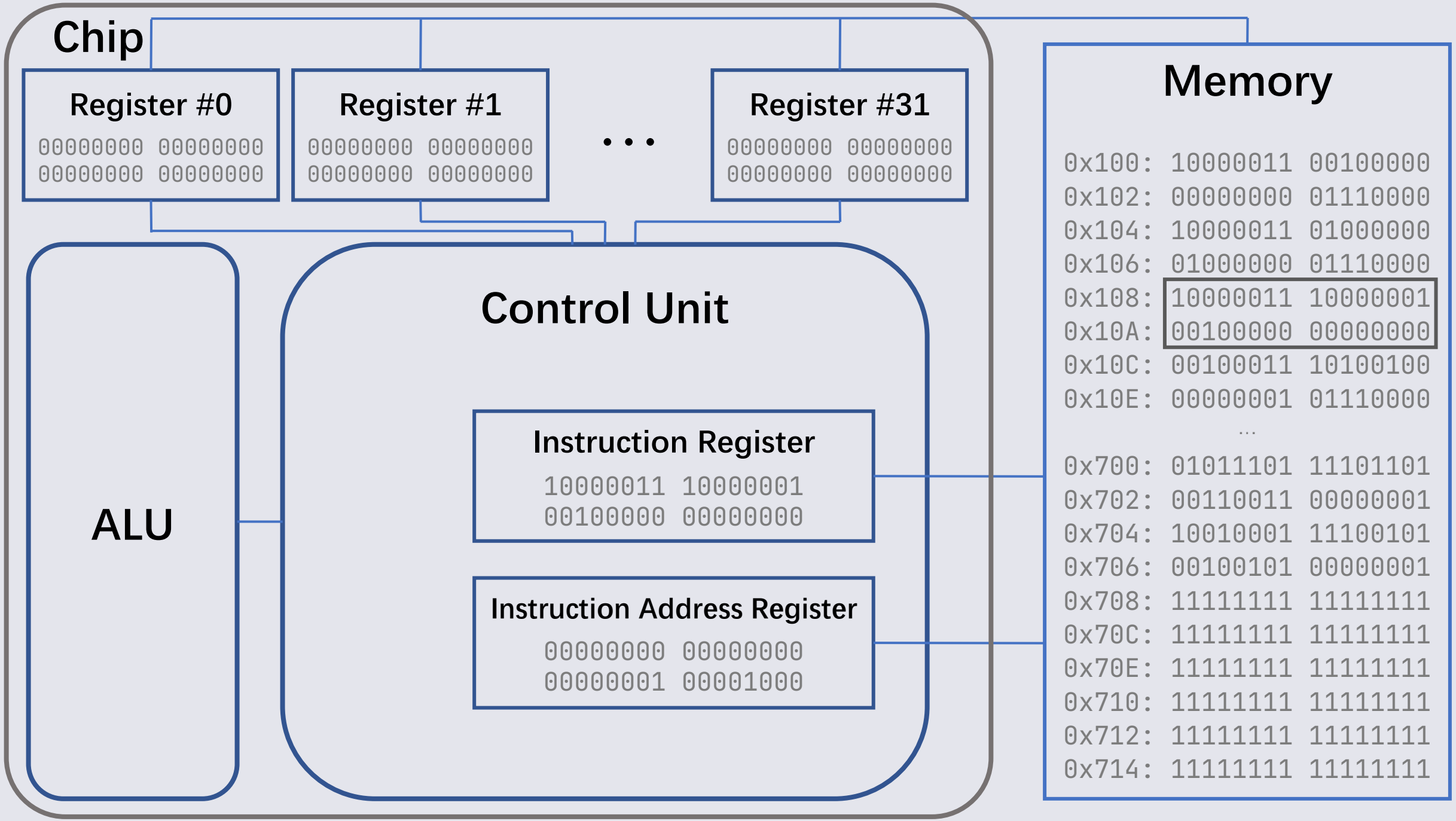

Chip
Register #0
00000000 00000000
00000000 00000000
Register #1
00000000 00000000
00000000 00000000
• • •
Register #31
00000000 00000000
00000000 00000000
ALU
Control Unit
Instruction Register
10000011 00100000
00000000 01110000
Instruction Address Register
00000000 00000000
00000001 00000000
Memory
0x100: 10000011 00100000
0x102: 00000000 01110000
0x104: 10000011 01000000
0x106: 01000000 01110000
0x108: 10000011 10000001
0x10A: 00100000 00000000
0x10C: 00100011 10100100
0x10E: 00000001 01110000
...
0x700: 01011101 11101101
0x702: 00110011 00000001
0x704: 10010001 11100101
0x706: 00100101 00000001
0x708: 11111111 11111111
0x70C: 11111111 11111111
0x70E: 11111111 11111111
0x710: 11111111 11111111
0x712: 11111111 11111111
0x714: 11111111 11111111


Chip
Register #0
00000000 00000000
00000000 00000000
Register #1
00000000 00000000
00000000 00000000
• • •
Register #31
00000000 00000000
00000000 00000000
ALU
Control Unit
Instruction Register
10000011 00100000
00000000 01110000
Instruction Address Register
00000000 00000000
00000001 00000000
+4
Memory
0x100: 10000011 00100000
0x102: 00000000 01110000
0x104: 10000011 01000000
0x106: 01000000 01110000
0x108: 10000011 10000001
0x10A: 00100000 00000000
0x10C: 00100011 10100100
0x10E: 00000001 01110000
...
0x700: 01011101 11101101
0x702: 00110011 00000001
0x704: 10010001 11100101
0x706: 00100101 00000001
0x708: 11111111 11111111
0x70C: 11111111 11111111
0x70E: 11111111 11111111
0x710: 11111111 11111111
0x712: 11111111 11111111
0x714: 11111111 11111111


Chip
Register #0
00000000 00000000
00000000 00000000
Register #1
00000000 00000000
00000000 00000000
• • •
Register #31
00000000 00000000
00000000 00000000
ALU
Control Unit
Instruction Register
10000011 00100000
00000000 01110000
Instruction Address Register
00000000 00000000
00000001 00000100
Memory
0x100: 10000011 00100000
0x102: 00000000 01110000
0x104: 10000011 01000000
0x106: 01000000 01110000
0x108: 10000011 10000001
0x10A: 00100000 00000000
0x10C: 00100011 10100100
0x10E: 00000001 01110000
...
0x700: 01011101 11101101
0x702: 00110011 00000001
0x704: 10010001 11100101
0x706: 00100101 00000001
0x708: 11111111 11111111
0x70C: 11111111 11111111
0x70E: 11111111 11111111
0x710: 11111111 11111111
0x712: 11111111 11111111
0x714: 11111111 11111111


Chip
Register #0
00000000 00000000
00000000 00000000
Register #1
00000000 00000000
00000000 00000000
• • •
Register #31
00000000 00000000
00000000 00000000
ALU
Control Unit
Instruction Register
10000011 01000000
01000000 01110000
Instruction Address Register
00000000 00000000
00000001 00000100
+4
Memory
0x100: 10000011 00100000
0x102: 00000000 01110000
0x104: 10000011 01000000
0x106: 01000000 01110000
0x108: 10000011 10000001
0x10A: 00100000 00000000
0x10C: 00100011 10100100
0x10E: 00000001 01110000
...
0x700: 01011101 11101101
0x702: 00110011 00000001
0x704: 10010001 11100101
0x706: 00100101 00000001
0x708: 11111111 11111111
0x70C: 11111111 11111111
0x70E: 11111111 11111111
0x710: 11111111 11111111
0x712: 11111111 11111111
0x714: 11111111 11111111


Chip
Register #0
00000000 00000000
00000000 00000000
Register #1
00000000 00000000
00000000 00000000
• • •
Register #31
00000000 00000000
00000000 00000000
ALU
Control Unit
Instruction Register
10000011 01000000
01000000 01110000
Instruction Address Register
00000000 00000000
00000001 00001000
Memory
0x100: 10000011 00100000
0x102: 00000000 01110000
0x104: 10000011 01000000
0x106: 01000000 01110000
0x108: 10000011 10000001
0x10A: 00100000 00000000
0x10C: 00100011 10100100
0x10E: 00000001 01110000
...
0x700: 01011101 11101101
0x702: 00110011 00000001
0x704: 10010001 11100101
0x706: 00100101 00000001
0x708: 11111111 11111111
0x70C: 11111111 11111111
0x70E: 11111111 11111111
0x710: 11111111 11111111
0x712: 11111111 11111111
0x714: 11111111 11111111


Chip
Register #0
00000000 00000000
00000000 00000000
Register #1
00000000 00000000
00000000 00000000
• • •
Register #31
00000000 00000000
00000000 00000000
ALU
Control Unit
Instruction Register
10000011 10000001
00100000 00000000
Instruction Address Register
00000000 00000000
00000001 00001000
Memory
0x100: 10000011 00100000
0x102: 00000000 01110000
0x104: 10000011 01000000
0x106: 01000000 01110000
0x108: 10000011 10000001
0x10A: 00100000 00000000
0x10C: 00100011 10100100
0x10E: 00000001 01110000
...
0x700: 01011101 11101101
0x702: 00110011 00000001
0x704: 10010001 11100101
0x706: 00100101 00000001
0x708: 11111111 11111111
0x70C: 11111111 11111111
0x70E: 11111111 11111111
0x710: 11111111 11111111
0x712: 11111111 11111111
0x714: 11111111 11111111

指令集架构

什么是指令集架构?

- 统一标准
- 规定指令的格式和行为
- 通常需要有运算指令、读写内存指令、分支指令
- 规定寄存器的数量、名称和用途
- 复杂指令集 (CISC) 和精简指令集 (RISC)

CISC v.s. RISC

	RISC	CISC
Instruction width	Fixed(despite some 16bits compacted extensions)	Variable
Memory access style	Load, store with memory compute in registers	One single instruction can access both registers and memory
Code size and cycles	Low cycles per second, large code sizes	Small code sizes, high cycles per second
Memory access efficiency	Heavy use of RAM	More efficient of RAM
Instruction Number	Small	Large number

RISC-V RV32I

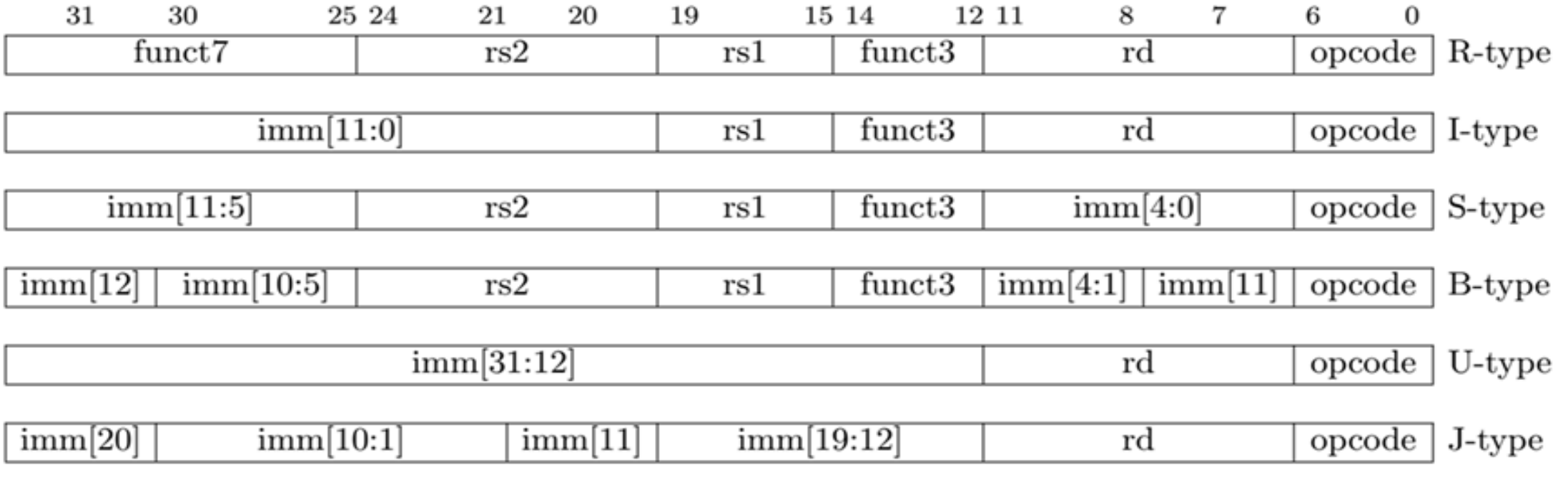
RISC-V RV32I

- RISC-V（发音为“risk-five”）是一个基于精简指令集（RISC）原则的开源指令集架构（ISA）。该项目 2010 年始于加州大学柏克莱分校，但许多贡献者是该大学以外的志愿者和行业工作者。
- 作业里涉及的指令是 RV32I 的一部分。32 位基础整数指令集，它支持 32 位寻址空间，支持字节地址访问，寄存器也是 32 位整数寄存器。

RISC-V RV32I – 寄存器

- 32 个通用寄存器
  - 虽然通用，但是有一些的用途较为明确
  - SP (Stack Pointer): 用于指向栈
  - RA (Return Address): 用于储存返回地址
- 1 个指令地址寄存器: PC (Program Counter)

RISC-V RV32I – 以指令形式分类



RISC-V RV32I

- 具体的指令请阅读手册
- 也可以从网上找一些资料来辅助了解
- 关于 CPU 执行的原理，可以参考 https://www.bilibili.com/video/BV1EW411u7th 第 5-9 集

谢谢