

## 目录

<b>1 第四章 cpu 的组成</b>	<b>1</b>
1.1 概述 . . . . .	1
1.1.1 cpu 的构造法 . . . . .	1
1.1.2 PC . . . . .	1
1.1.3 ALU . . . . .	1
1.1.4 CACHE . . . . .	1
1.1.5 <b>TODO</b> Memory . . . . .	3
1.2 interlude . . . . .	3
1.2.1 Havard architecture and Princeton architecture . . . .	3
1.3 组件的硬件构成 . . . . .	3
1.3.1 Processor 的构成 . . . . .	3
1.3.2 Memory 的简单构成 . . . . .	7

## 1 第四章 cpu 的组成

### 1.1 概述

#### 1.1.1 cpu 的构造法

对于一个 cpu , 其中分为三个部分. 第一个部分是处理器, 第二部分是内存, 第三部分是总线.

1. 处理器, 里面由 ALU, 寄存器, PC
2. 内存, 就是内存相关的, Cache 缓存, memory 内存.
3. 总线, 就是 IO 相关的

#### 1.1.2 PC

PC 就是存储着指令, 其告诉 ALU 该如何处理寄存器.

### 1.1.3 ALU

ALU 就是运算单元, 其能够执行 add sub 等指令, 对寄存器进行操作.

### 1.1.4 CACHE

1. Cache 简介 Cache 是缓存, 是一种更为快速的内存. 速度和数量介于内存和寄存器之间.

Cache 用于提高效率. 我们有三种不同效率的存储单元, 将适合的任务分配给它们, 以此提高效率 (当然也是成本效率)

2. SRAM Cache memory operates between 10 to 100 times faster than RAM, requiring only a few nanoseconds to respond to a CPU request. The name of the actual hardware that is used for cache memory is high-speed static random access memory (SRAM).

实现的 Cache 的硬件为 SRAM, (high-speed static random access memory). 而 DRAM (dynamic random access memory) 用于构建 memory. SRAM 之中的数据, 只要没有断电 (power), 就能够一直保存, 不同于 DRAM 的数据那样, DRAM 之中的数据, 一把来说几秒钟后就会消失, 需要周期性地 refresh 一下. SRAM use latches and transistor to store data.

3. Cache 的多级建构 Cache 分为了三个 Level, 有 L1, L2, L3, 其中 L3 速度最慢, 是用来辅助 L1, L2 的.

Cache memory is a small amount of fast memory that is used to store frequently accessed data. It is located close to the CPU and is used to reduce the average time it takes to access data from the main memory.

There are three general levels of cache: L1 cache, L2 cache, and L3 cache. Each level is differentiated by its speed, size, and proximity to the CPU.

L1 cache, or primary cache, is extremely fast but relatively small. It is usually embedded in the processor chip as CPU caches.

L2 cache, or secondary cache, is often more capacious than L1 but slower. It is usually located on the processor chip or on a separate chip on the motherboards.

L3 cache, or tertiary cache, is larger than L2 but slower. It is usually located on a separate chip on the motherboards.

Is there anything else you would like to know about cache memory?

Computers, Explained -SearchStorage. <https://www.techtarget.com/searchstorage/definition/cache-memory> 访问时间 2023/4/1.

(2) Explainer: L1 vs. L2 vs. L3 Cache | TechSpot. <https://www.techspot.com/article/2066-cpu-l1-l2-l3-cache/> 访问时间 2023/4/1.

(3) How Does CPU Cache Work and What Are L1, L2, and L3 Cache? - MUO. <https://www.makeuseof.com/tag/what-is-cpu-cache/> 访问时间 2023/4/1.

#### 1.1.5 TODO Memory

### 1.2 interlude

#### 1.2.1 Havard architecture and Princeton architecture

程序有存储的地方, 数据也有存储的地方, 我们常将其抽象出来, 认为这两个部分是存在一个地方里的. 但实际上有区别.

数据的存储单元, 程序的存储单元, 在哈佛架构之中是分开的. 也就是说, 在哈佛架构中, 存在两个单元, 分别由两个总线进行单元和处理器之间的数据传输

### 1.3 组件的硬件构成

#### 1.3.1 Processor 的构成

##### 1. ALU Register PC Extender

(a) ALU 是算术运算单元, 即, 进行 and or add sub 运算的单元.

- (b) Register 我们很熟了, 支持读写操作. PC 我们也很熟了, 支持 +4, + imm 操作.
- (c) Extender 是 imm 生成单元也是扩展单元, 我们有的时候需要进行一些数据的符号扩展.

## 2. ALU (算术逻辑运算单元) [%]

□ 完善该标题

输入两个 32 位数据, 输出一个 32 位的数据. 进行位运算或者加减运算.

- (a) ALU 的接线 一个一位 ALU 单元应该
  - i. 根据 ALU control 的值, 决定 ALU 该执行什么运算
  - ii. 根据输入 A B 给出结果 R
  - iii. 判断是否溢出, 接出一根线: Overflow, 其值为 1 当且仅当发生溢出
  - iv. 判断结果是否为 0, 接出线: Zero, 其值为 1 当且仅当结果为 0
  - v. 接出一根线, Carry Out. 用于串联的进位

### (b) ALUop 和 ALU control

ALUop 是一个二位的信号, 其和 funct field 结合在一起, 通过 ALU control 单元, 生成一个 ALU control 信号.

这里使用的是多级<sup>1</sup>的控制信号生成器. 多级, 但是每一级的规模很小, 这使得信号生成的延迟降低了. 控制信号的延迟是非常重要的参数. CPU 的时钟周期就取决于这个参数.

但是出于不明原因, 我们这里居然没有提及 ALU control 信号, 将其和 ALUop 混为一谈, 着实离谱. 下面有一些 ALUop 的出现, 他们实际上指的是 ALU control 信号

---

<sup>1</sup>这种多级译码的方式---主控制单元生成 ALUop 位作用 ALU 的输入控制信号, 在生成实际信号来控制 ALU---是一种常见的方式. 多级控制可以减小主控制单元的规模. 多个小的控制单元可以潜在地减小控制单元的延迟.

ALUop 是指令码的一个字段, 其和 funct 字段经过 ALU 控制单元生成实际的控制信号. 见下表:

ALUop	funct7	funct3	操作
00	XXXXXXX	XXX	0010
X1	XXXXXXX	XXX	0110
1X	0000000	000	0010
1X	0100000	000	0110
1X	0000000	111	0000
1X	0000000	110	0001

能够看出, 实际上控制单元的输入可以简化, 比如说

(c) ALU control 信号的功能

ALU con	操作
0000	and
0001	or
0010	add
0110	sub subtraction
0111	slt set on less than
1100	nor

(d) 构建简单的 ALU

以 Multiplexer 为基础, 而后构建 and or add 操作 and 使用 and 门, or 使用 or 门, add 使用一个一位 Full adder. 构建是简单的.

ALU con 的后两位是用于 Multiplexer 的

- 00 代表 Multiplexer 的第 0 个输入: A and B
- 01 代表 Multiplexer 的第 1 个输入: A or B
- 10 代表 Multiplexer 的第 2 个输入: A + (B)
- 11 代表 Multiplexer 的第 3 个输入: Less

其中 (B) 代表对 B 进行一定处理之后的数据. Less 是 slt 的实现之中会稍微提及的一个信号.

## (e) one bit ALUs 的串联 [5/5]

在串联之中我们要实现

- ☒ sub 操作
- ☒ slt 操作
- ☒ nor 操作
- ☒ Overflow 判断
- ☒ Zero 判断

## i. sub 操作

设 ALUop 的第三位为 Binvert. 通过等式  $R = A - B = A + B'$  来实现减法.

A. 在 one bit ALU 之中, 通过一个 Mux 和反相器, 使 B 取反

B. 接入末位 ALU viz., ALU0 的 CarryIn. 使得结果 +1

这就有  $R = A + B' + 1$ . 也就有  $R = A - B$ .

## ii. slt 操作

$$R = (A < B)$$

true 代表 1, false 代表 0. 我们用 Less 作为 one bit ALU 的输入信号. 我们只需要计算出  $A - B$  的值, 然后  $A - B$  的符号传回 Less 的末位 (Less 在其他位的值均为 0), 最后 result 直接等于 Less.

## iii. nor 操作

ALUop = 1100 的时候, 其为 ALU 为 nor 操作. 观察后两位, 这个时候 Multiplexer 选择第 0 位数据, 只需要让 ALUop 的最高位为 Ainvert. 其为 1 的时候, A 的数据反相. 由

$$\overline{A + B} = \overline{A} * \overline{B}$$

$$\text{就有 } R = \overline{A + B}$$

## iv. Overflow 判断

设输入的两个符号位为 s1 s2, 结果的符号位为 s3, 那么有

$$\text{Overflow} = (\text{s1 and s2}) \text{ xor s3}$$

就有, 当溢出发生的时候, Overflow 为 1.

v. Zero 判断

每一位结果取 nor 即为结果.

3. PC

(a) PC 的功能

4. Register 我们应该有这些功能:

(a) 根据 Register 编号 Rw 将 busW 写入到寄存器之中

(b) 根据 Register 编号 Ra Rb 将寄存器的值输出到 busA, busB 上

并且读操作不应收到时钟控制.

### 1.3.2 Memory 的简单构成

1. 数据存储器 应当采用时序逻辑设计. 其应做到, 将指定的数据 DataIn 写入到 Addr 指定的内存位置里, 并且能够根据指定的 Addr 将内存中对应的数据写到输出 DataOut 上. 这就是读写操作, 但其中读的操作不应该受到时钟的控制 (至少是可以不受到时钟的控制)

2. 指令存储器 一个程序运行的之前, 程序装载机将程序装载起来, 在程序运行过程中, 不能对指令存储器进行写入的操作.

其应做到

(a) 根据对应的 Addr 给出对应位置存储的指令.

(b) 不能在程序运行过程对其进行写操作.