



Datawhale 开源社区

DATAWHALE OPEN SOURCE COMMUNITY

深入理解计算机系统 (10)

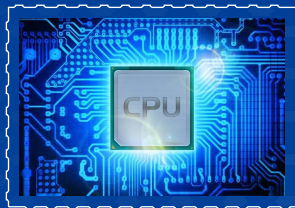
Computer Systems A Programmer's Perspective

CSAPP

李岳昆、易远哲

realjurk@gmail.com、yuanzhe.yi@outlook.com

2021 年 12 月 1 日



第 I 部分

处理器体系结构-II

Y86-64 处理器硬件结构

oooooooo

在取址阶段中，取址操作以程序计数器（PC）的值为起始地址。由于 Y86-64 指令集中最长的指令占 10 字节，为了保证每次取址操作至少能够获得一条完整的指令，取址操作每次从指令内存中读取 10 各字节。

接下来将这 10 字节分为两部分，第一部分占 1 字节，第二部分占 9 字节。例如编码为 30 f8 08 00 00 00 00 00 00 00 的指令，将被分成 30 和 f80800000000000000。

随后名为 Split 的硬件单元处理第一部分。它将这个字节又分成两部分，每部分占 4 个比特位，使这个字节分为两个字段，分别为指令代码和指令功能，用 icode 和 ifun 表示。

根据 icode 可以确定当前指令的状态信息，例如指令的合法性。如果 icode 在 0x0 到 0xB 之间，那么它就是一条合法指令。此外，通过 icode 还可以判断当前指令是否包含寄存器指示符字节和常数字节。

通过前面说的判断结果，就可以计算出当前指令的长度。例如一个指令既含寄存器指示符字节，又含常数字节，那它的长度就是 10 字节；如果既不含寄存器指示符字节，又不含常数字节，那它的长度就是 1 字节。

与此同时，还可以通过将 PC 值加上当前的指令长度来计算内存中下一条指令的地址，用于后续的更新阶段。

在前面我们处理了一条指令中的头一个字节，对于剩下的 9 个字节，我们通过名为 Align 的硬件单元来产生寄存器字段和常数字段。该硬件单元通过信号 need_regids 来判断该指令是否包含寄存器指示符字节。若 need_regids = 1，说明该指令包含寄存器指示符字节，那么第一个字节将被分成两部分，每部分占 4 个比特位，然后分别装入寄存器指示符 rA 和 rB 中；若 need_regids = 0，说明该指令不包含寄存器指示符字节，此时将 rA、rB 这两个字段设置为 0xF。

此外，若该指令含有常数，Align 单元还将产生常数字段 valC。当 need_regids = 1 时，valC 被设为指令的第 2 ~ 9 字节；当 need_regids = 0 时，valC 被设为指令的第 1 ~ 8 字节。

译码阶段的操作是从寄存器文件中读取数据，在 Y86-64 处理器中寄存器文件有两个读端口，它支持同时进行两个读操作，两个读端口的地址输入为 `srcA` 和 `srcB`，从寄存器文件中读出的数值通过 `valA` 和 `valB` 输出。

读端口的 `srcA` 和 `srcB` 用于产生寄存器的 ID，这需要寄存器指示符 `rA` 及 `rB`。

由于某些指令，例如 `push` 指令，该指令的寄存器指示符中只含有目的寄存器的 ID，但执行压栈操作时，还需要获得栈顶指针 `rsp` 的值。因此 `srcA` 和 `srcB` 不仅需要传入 `rA` 和 `rB`，还需要传入指令代码 `icode`。

执行阶段的核心部件 ALU 根据指令功能 ifun 来判断要对输入的操作数进行何种运算。每次运行时，ALU 都会产生三个与条件码相关的信号——零、符号、溢出。

我们只希望 ALU 在执行算术逻辑指令时才设置条件码，而计算内存引用地址以及栈操作时不要设置条件码。因此我们使用 Set_CC 信号根据指令代码 icode 来控制是否需要设置条件码。

此外，我们使用名为 Cond 的硬件单元来控制跳转操作。Cond 会根据指令功能和条件码寄存器来产生 Cnd 信号。对于跳转指令，如果 $Cnd = 1$ ，就执行跳转；如果 $Cnd = 0$ ，则不执行跳转。

该阶段的硬件设计主要包含以下四个控制块：

- ① 读控制块，用于进行读操作。
- ② 写控制块，用于进行写操作。
- ③ 内存地址控制块，用于产生内存地址。
- ④ 数据输入控制块，用于输入数据。

除此之外，访存阶段的最后还将根据 `icode` 判断出的指令有效性以及内存状况产生 `instr_valid` 和 `imem_error` 信号来计算状态码。

首先为寄存器文件系统添加 M 和 E，这两个写端口，对应的地址输入为 dstE 和 dstM。需要注意的是，当执行条件传送指令 (cmov) 时，写入操作还要根据执行阶段计算出的 cnd 信号，当条件不满足条件时，以将目的寄存器设置为 0xF 来禁止写入寄存器文件。

在更新阶段，PC 的值有以下三种情况：

- ① 当前执行的指令是函数调用指令 `call`，此使将 PC 值设为 `call` 指令的常数字段。
- ② 当前执行的指令是函数返回指令 `ret`，此使将 PC 值设为 `ret` 指令在访存阶段从内存中读出的返回地址。
- ③ 当前执行的指令是跳转指令 `jxx`，此使若满足跳转条件 ($cnd = 1$)，则新的 PC 值等于跳转指令的常数字段；若不满足跳转条件 ($cnd = 0$)，则新的 PC 值等于当前 PC 值加上当前指令长度。
- ④ 数据输入控制块，用于输入数据。