取指令(IF)

CPU 在取指令阶段（IF 阶段）时，先向一级指令缓存要指令，要到指令后我们将程序计数器（PC）自增 1（1 表示移动一条指令的宽度，如果数据单位是 32 位，那么就自增 1，如果数据单位是 8 位 1 字节，那么就自增 4）。这样我们在下次取指令的时候就能取到下一条指令了。同时如果你实现了分支预测，那么在这里则需要做另外的处理。

解码(ID)

注意到我们有这些指令对：add 和 addi，addu 和 addui，or 和 ori 等，这些指令对的功能是一样的，只是取操作数的方式不一样，如果我们能用某种方式统一这些指令对，那么我们在实现这些指令的执行将会变得更简单，因为对于同一类的指令我们能做同一个操作。解码器就是做这个事情的。

CPU 在解码阶段（ID 阶段）时还需要根据解码器得到的操作数寄存器的编号，从相应的寄存器中取出 ALU 所需要的操作数，因此我们也将寄存器归到解码阶段中。

执行(EX)

执行阶段（EX 阶段）是我们调用 ALU 进行真正的计算过程。由于乘法和除法的速度比较慢，如果 1 个周期能完成加法的计算，那么乘法和除法就需要超过 1 个周期的时间，也就是说乘法器和除法器在多周期 CPU 里是多周期的。同时，所有的浮点运算也都是多周期的，比如浮点加是 4 个周期（另外在浮点运算单元中同样存在流水线，把浮点加法分成 4 个阶段计算）。

同时对于 syscall 指令，我们也要在执行阶段完成操作。

因此执行阶段 CPU 一共会有以下执行模块，分别是 ALU、整数的乘除运算单元、浮点运算单元（内部仍存在流水线，而且加减法和乘除法可以并行执行）和其他的一些处理电路。

【ALU】

ALU 在执行阶段将会按照给定的 ALU 的微指令，对两个操作数进行运算（无论是加减运算、位运算、比较运算以及跳转指令需要的比较运算），并得到结果输出（输出包括运算结果、是否跳转等信息）。对于内存操作的指令，我们也需要 ALU 计算出我们要读取的内存的真实地址（因为指令规范中给定的是某个寄存器的偏移，或者当前指令的偏移）。

内存读写(MEM)

对于 lb、lbu、lh、lhu、lw、sb、sbu、sh、shu、sw 等内存操作的指令，这个阶段将被启用。这个阶段将会与一级的数据缓存交互。

寄存器写(WB)

这个阶段（WB 阶段）有结果的指令将会把数据存到对应的寄存器里。