浅谈CPU工作方式

CPU的工作方式分为：单周期、多周期、流水线、超标量。

内容介绍前，必须要明白一条指令从存储器中被取出 然后在CPU中执行的全过程。

PC指令地址寄存器将PC指令地址送入ROM指令存储器，存储器根据该PC地址将inst指令取出，送入至IR指令寄存器。之后，指令寄存器会将指令进行译码（解码），根据指令表的解释，得到多个控制信号，这些信号送往不同的模块里作为多路选择器的选择端筛选出指令所需要的数据。译码的同时，若指令译码后需要，会从Register寄存器堆里取出操作数，接下来进行运算，将运算的结果，根据控制信号送往译码后的写入位置，这可以是Register寄存器堆，也可以是RAM存储器等。（有的CPU是只有一个存储器，既存放指令，又存放数据，主要按照地址来区分取指还是取数）

当然，这里的叙述是最全的指令执行过程，包括大部分的指令运行过程，有的指令并不需要经历那么多的环节，例如跳转指令，在运算时根据得到的结果判断是否需要跳转PC地址还是直接执行下一条指令。

综上所述，（初学范围内）所有指令在CPU的活动阶段包括：1.取指；2.译码（解码）；3.执行（运算）；4.访存；5写回。

了解了一条指令的运行过程，下面我们就可以谈谈CPU的架构进化史了。

单周期CPU，一个时钟周期会执行一条指令。在一个时钟周期内，要经历指令运行的5个阶段，这就要求该周期的时间必须要足够长，促使这5个阶段能在该时间内能运行完。所以，我们经常看到的单周期CPU的时钟频率都不是很高（除了支持极少数量指令的CPU）。、

并且，单周期CPU的资源利用率很低。例如，当取指模块取指时，其他模块都在“吃灰”，即其他模块在“假”工作。此时正在取指，执行（运算）模块的运算的结果不能被使用，因为指令还未经过译码，且取出的操作数并不是有效的，故ALU的运算结果并不是可信的。

后来，出现了多周期CPU，一个时钟周期执行一条指令的一个阶段。看到此处，有的同学肯定会问了，单周期（CPU）还能够一个周期完成一条指令的五个阶段，多周期（CPU）完成一个阶段，这不会导致多周期CPU的性能下降吗？是的，如果保持原来的时钟频率不变，确实会出现这样的情况，所以关键是——升频，多周期CPU经过“切割”后可以提升时钟频率，导致多周期CPU在性能上高于单周期（CPU）。

具体是怎么回事呢？多周期CPU在控制模块里增加了状态机以及与其相关的其他的一些控制信号，这里我们主要说说状态机，每经过一个时钟周期就会跳转至下一个状态。除了复位后（防止复位后直接进入if取指态后，PC无法即使变化。例如，PC是在时钟下降沿变化，而复位也是在该下降沿无效，此时PC保存不变即错误的，可状态机此时处于取值状态）的第一条指令进入的第一个状态是init态外（后面的状态就是正常的if取指态），其他指令则是if取指态。

言归正传，划分了多个状态后，一个时钟周期所做的操作将从五个阶段的量变为一个阶段，故这才是CPU能升频的真正原因。

在单周期（CPU）里，不管是什么指令，都会经过5个阶段，即使有的指令并不需要经过其他的阶段，依然需要等待至下个时钟周期才能运送完毕。这会大大增加不必要的时间开销。例如，J直接跳转指令可在id译码态就直接在下个时钟周期跳转回if取指态，而不用在经历exe执行态、mem访存态、wb写回态。

在加入了状态机后，指令就不需要从if取指阶段一直执行到wb写回阶段，状态机根据指令的种类，可以控制状态的转换。这就是所谓的“把时间花在刀刃上”。

小结的话，多周期CPU较于单周期CPU，减小了不必要的时间开销，尽可能加快了时间频率。在性能上，优于单周期（CPU）。

但是，多周期CPU依然没有解决资源利用率低的问题，在CPU处于某一状态时，依然只有一部分的模块在工作，其他模块在“吃灰”。为了解决这个问题，流水线CPU应运而生，它延续着“切割”的思想，将阶段“流水”化。每一个阶段代表着一级流水。下面我们来好好了解它的设计思想。

流水线CPU，一个时钟周期可执行多条指令的一个阶段。注意这里的“多条指令”的指令条数就是流水线的级数，这里的“一个阶段”指的是这么多条指令的不同阶段。

拿经典的五级流水线CPU来说，它没有在多周期CPU的结构上进行修改，而是重写了逻辑，将5个阶段作为5级流水，用组合逻辑对每级流水输入的数据进行处理并将下级流水所需的数据进行输出，每级流水之间是通过时序逻辑的D触发器进行数据传输的。

首先PC\_reg寄存器将PC送入ROM指令存储器里，取出inst指令后，同时把inst和PC送往if\_id（取指阶段 -> 译码阶段）模块在时钟上升沿（是上升沿还是下降沿，具体看设计者设计）时把数据传输给id（译码）模块。

id模块将inst指令解码，同时向寄存器堆模块取出操作数（如果需要的话），将数据传送至id\_ex（译码阶段 -> 执行阶段）模块在时钟上升沿时把数据传输给ex（执行）模块

ex模块相当于ALU，将根据指令的具体需要，将操作数进行逻辑、算数或移位等运算，将数据传送至ex\_mem（执行阶段 -> 访存阶段）模块……

mem模块是针对load、store指令（如lw、sw），load指令根据ex阶段计算出的地址，从RAM存储器里读出数据，store则是往地址写入数据。load指令会返回出一个数据在wb阶段写进寄存器堆里，而store则没有数据返回……

wb模块是将运算的结果或者是从RAM存储器里得到的数据写入寄存器堆里。实际上，Register寄存器堆模块就是wb模块，一般我们就直接将写回的操作放在了寄存器堆里了。

小结一下，每个阶段的模块都是使用组合逻辑处理数据（除了if取值时，PC\_reg寄存器模块需要用到时钟），而阶段转换的模块都是“清一色”地使用时序逻辑（D触发器）将数据送往下一个阶段模块。

所以，在同一个时间，五个阶段都在同时工作。

例如，CPU顺序执行以下指令：A，B，C，D，E，F……（NOP空指令，CPU不作任何操作）

开始时，第一个时钟周期，if阶段执行A指令，id阶段、ex阶段、mem阶段、wb阶段均执行NOP空指令；

第二个时钟周期，if阶段执行B指令，id阶段执行A指令，ex阶段、mem阶段、wb阶段均执行NOP空指令；

第三个时钟周期，if阶段执行C指令，id阶段执行B指令，ex阶段执行A指令，mem阶段、wb阶段均执行NOP空指令；

第四个时钟周期，if阶段执行D指令，id阶段执行C指令，ex阶段执行B指令，mem阶段执行A指令，wb阶段均执行NOP空指令；

第五个时钟周期，if阶段执行E指令，id阶段执行D指令，ex阶段执行C指令，mem阶段执行B指令，wb阶段执行A指令；

第六个时钟周期，if阶段执行F指令，id阶段执行E指令，ex阶段执行D指令，mem阶段执行C指令，wb阶段执行B指令；

……

当然，这里省略了很多结构细节，包括流水线暂停设计、数据前递、数据相关等等。

与多周期CPU相比，流水线CPU从原来只能执行一条指令的一个阶段，到现在能执行多条指令的一个阶段，资源利用率增大，指令执行并行度提高。

但是，这就是CPU的极限了吗？

不，超标量CPU打破了这种单发射流水线CPU的极限。

超标量CPU，拿最简单的双发射顺序执行流水线CPU来说，可以同时支持发射两条CPU进入……

（未完待续）