第 1 章　可编程交换芯片概述

要学习 P4 语言，需要先掌握可编程交换芯片的基础知识。可编程交换芯片是交换芯片领域一个新的发展趋势，并且已经拓展到可编程网卡领域，进而组成一个端到端的可编程网络。本章将介绍可编程交换芯片的产生背景、发展过程、实现原理、特点优势和应用场景。

1.1 可编程交换芯片产生的背景

1.1.1　可编程交换芯片是 SDN 发展过程的自然产物

从互联网诞生的背景看，分布式架构是互联网天生的基因。分布式架构要求整个网络没有中心节点，在部分节点不能正常工作时，剩余的节点也能正常通信。这就要求各种网络设备（如交换机、路由器等）一旦配置完成，便能够独立工作，所以对网络设备制造商而言，只需要遵循标准的网络协议规范，便可以与其他厂商的网络设备正常通信，至于网络设备内部，各个厂商可以自己决定其实现细节。

交换机中最核心的部分就是交换芯片。交换芯片一般采用专用集成电路（Application Specific Integrated Circuit，ASIC）实现。ASIC 芯片的开发周期较长，一般需要 12 ~ 18 个月，设计一旦完成，就不能增加新的功能。

从 20 世纪 70 年代开始到 2000 年，这 20 多年网络的发展有两个特点：一是带宽越来越大，端口速率从百兆、千兆，提升到 1Gb/s、10Gb/s；二是网络协议标准的数量极速增长，从几百个增长到几千个。

网络设备制造商一方面不断提升交换芯片的集成度，不断提升端口速率；另一方面不断增加交换芯片的功能，支持新的网络协议。如此，经过长时间的发展，网络设备越来越固化，越来越封闭，控制面和数据面紧紧耦合在一起。网络研究者发现，不论是设备厂商还是网络管理员，都不愿意实验新的网络协议和网络技术，因为他们不想承担用生产环境的流量做实验的风险。

为了激发网络领域的创新和活力，斯坦福大学于 2006 年开展了一项名为 Clean Slate（重新开始）的研究计划，该计划的目标非常宏大，力图重塑互联网。受 Clean Slate 项目的启发和影响，2008 年，Nick McKeown 等提出了控制面和数据面分离的概念：OpenFlow。研究者发现，虽然各种网络设备形态各异，但是几乎都包含了相同的机制——三态内容寻址存储器（Ternary Content Addressable Memory，TCAM），用于三态匹配并执行特定动作。研究者通过使用 TCAM，匹配特定网络流量，然后执行特定动作。匹配什么

特征的流量，由控制面决定；报文匹配后执行什么样的操作，由数据面决定。如此，控制面与数据面的耦合度大幅降低，既可以将实验性流量与生产环境的流量进行分离，隔离线上风险，又可以尝试新的网络协议和网络技术，推动新技术的发展。

2009 年，Nick McKeown 等正式提出了软件定义网络（Software Defined Network，SDN）的概念，其中控制面与数据面分离是 SDN 的核心思想，OpenFlow 作为控制面与数据面的接口被学界和业界广泛接受。SDN 架构如图 1-1 所示。

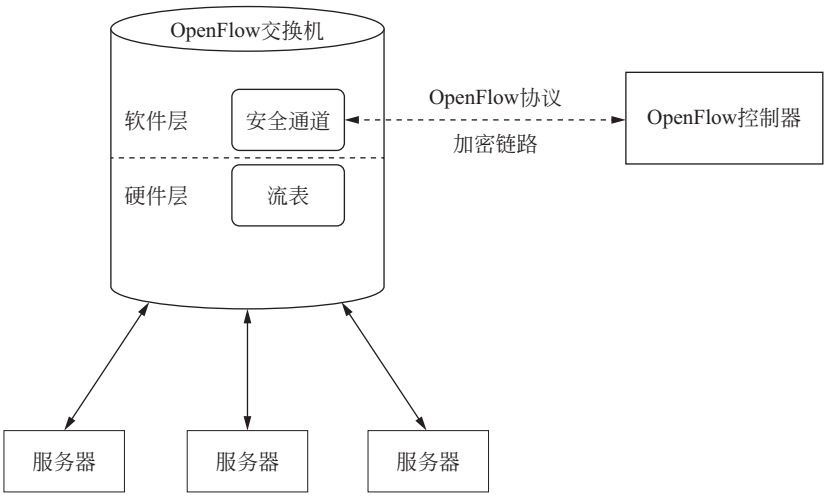

图 1-1　SDN 架构

但 SDN 技术还是做了妥协，它只解决了控制面与数据面分离的问题，数据面还是使用传统的商业交换机和路由器，新的网络协议和网络技术在数据面的创新工作仍然受到了很大程度的制约。

2013 年，Nick McKeown 等将注意力重新放到了数据面，引入了可重配置匹配表（Reconfigurable Match Table，RMT）模型，提出了可重配置交换芯片架构，在数据面引入可编程技术，极大地方便了新的网络技术和网络协议的开发和验证工作。

可重配置交换机具备以下特征。

（1）允许在不更换硬件的前提下，重新定义数据面的功能。

（2）允许对报文任意字段进行匹配和修改。

可重配置交换芯片后来被称为可编程交换芯片（Programmable Switch Chip），它的特点是数据面可以根据程序重新配置，这与传统交换芯片有本质的不同。可编程交换芯片，将交换芯片的数据面功能统一抽象为匹配 – 动作表（Match-Action Table），其中匹配字段（Key）、动作（Action）和表项容量由用户数据面程序定义，表项的内容由用户的控制面程序下发，极大地提升了交换芯片的可编程性，可以灵活地满足各种不同业务场景的功能需求。

之所以将这种芯片称为可编程交换芯片，是为了与传统的交换芯片进行区别。但是传统交换芯片在一定程度上也是可以编程的，只是灵活性差一些罢了。因此，准确地说，可重配置交换芯片应该被称为“可灵活编程的交换芯片”。本书依照惯例沿用“可编程交换芯片”的叫法，部分文章中也会出现“可编程芯片”的说法，它们的含义是相同的。

2013 年 5 月，Nick McKeown、Pat Bosshart 等联合成立了 Barefoot Networks 公司（2019 年被 Intel 收购），该公司主要从事可编程交换芯片以及相关软件的研发工作。2016 年 6 月，Barefoot Networks 公司推出业界第一款可编程交换芯片 Tofino，实现了协议无关的交换机架构（Protocol Independent Switch Architecture，PISA），最高支持 6.4Tb/s 的转发速率。后续发布的 Intel Tofino 2 芯片，最高支持 12.8Tb/s 的转发速率。

自诞生起，可编程交换芯片就凭借其数据面可灵活编程的特点，在学界和业界产生了巨大的影响。可编程交换芯片是 SDN 发展过程的自然结果，又推动了 SDN 概念的进一步发展。

Nick McKeown 在 2019 年的 Open Networking Foundation Connect 会议上做了题为“SDN Phase 3: Getting the humans out of the way”的演讲，他将 SDN 的发展分为三个阶段，如图 1-2 所示。

图 1-2　SDN 发展的三个阶段

（1）在阶段一中，网络拥有者掌控软件，标志性概念及协议是 OpenFlow。

（2）在阶段二中，网络拥有者也能掌控报文处理，标志是 P4 可编程交换芯片、P4 可编程网卡。

（3）阶段三是全可编程网络，网络进入可验证的闭环发展阶段，程序员可以观测并验证报文的行为，发现控制面和转发面的代码 Bug 并进行及时修复。

在演讲中，Nick McKeown 引用了《软件定义网络之旅》一书的作者 John Donovan 的一句话，阐述了 SDN 的宗旨：“The vendors have a stranglehold over me, my job is to break that stranglehold”。这句话翻译成中文就是“设备制造商给我施加了控制，我的工作就是打破这种控制。”

1.1.2　可编程交换芯片的发展是学界与业界互相促进的结果

可编程交换芯片，是学界和业界互相合作、互相促进的典范。关于可编程交换芯片的学术研究，催生了可编程交换芯片的产品；可编程交换芯片产品的出现，又推动研究者将可编程交换芯片在更多领域进行应用，产生了更多创新的项目，发表了更多创新的论文。促使可编程芯片产生的里程碑事件列举如下。

（1）2008 年，Nick McKeown 等发表论文 *OpenFlow: Enabling Innovation in Campus Networks*，提出了交换机控制面和数据面分离的思想，标志着 SDN 的诞生。

（2）2013 年，Pat Bosshart 等发表论文 *Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN*，提出了可重配置匹配表（RMT）架构，为可编程交换芯片的产生奠定了理论基础，并提供了详细的实现参考。

（3）2013 年，Nick McKeown、Pat Bosshart 等在美国加利福尼亚州的圣克拉拉市联合成立了 Barefoot Networks 公司，专注于可编程交换芯片及相关软件的研发。

（4）2014 年，Pat Bosshart 等发表论文 *P4: Programming Protocol-Independent Packet Processors*，提出了与协议无关的网络数据面编程专用语言 P4。

（5）2015 年，Lavanya Jose 等发表论文 *Compiling Packet Programs to Reconfigurable Switches*，探讨了可编程交换芯片的编译器的设计问题。

（6）2016 年 6 月，Barefoot 推出 Tofino 芯片，实现了 PISA 架构，使用 16nm 芯片工艺，可以达到 6.4Tb/s 的吞吐量。

可编程交换芯片产生后，学界对它产生了浓厚的兴趣，一方面是围绕可编程交换芯片的开发平台提出新的方案，对解析器、编译器、验证工具等进行改进；另一方面是不断扩展它的使用边界，提出了很多应用场景。其中比较具有代表性的是 Rui Miao 等在 2017 年发表的论文 *SilkRoad: Making Stateful Layer-4 Load Balancing Fast and Cheap Using Switching ASICs*，在交换机上实现了有状态的 4 层负载均衡功能，可以存储千万级的连接信息。

从 2017 年开始，国内公有云厂商（如 Ucloud 和阿里云等）第一时间开始尝试在网关中使用可编程交换芯片技术，主要用在专线网关或者虚拟路由网关上。

2021 年，Tian Pan 等发表论文 *Sailfish: Accelerating Cloud-Scale Multi-Tenant Multi-Service Gateways with Programmable Switches*，详细介绍了阿里云基于可编程交换芯片的虚拟路由网关的设计和实现。

另外，百度、腾讯、Ucloud、京东等互联网和云计算公司，也在不同的场合介绍了其可编程交换芯片大规模落地应用的情况，在网络领域掀起了 P4 旋风。

1.2 可编程交换芯片的实现原理

1.2.1 传统交换芯片存在的问题

传统交换芯片经过几十年的发展，具备了如下特点：市场规模大，产品线丰富，协议支持完整，系统稳定性强，生态成熟稳定。传统交换芯片流水线如图 1-3 所示。

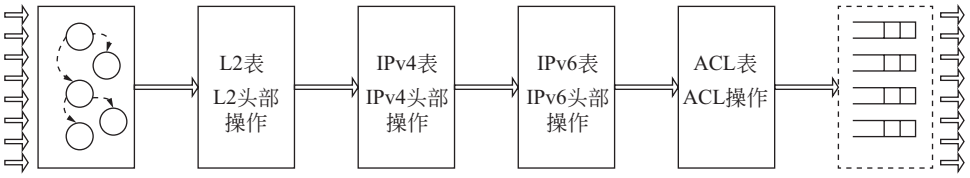

图 1-3　传统交换芯片流水线

传统交换芯片还有以下特点。

（1）功能固定：每个模块实现特定功能，不可更改。

（2）功能次序固定：每个模块前后次序固定，不可调整。

（3）资源基本固定：每个模块能够使用的资源是预先分配的，只能在很小的范围内调整。

以 Broadcom Tomahawk 交换芯片为例，它的编程接口如图 1-4 所示。

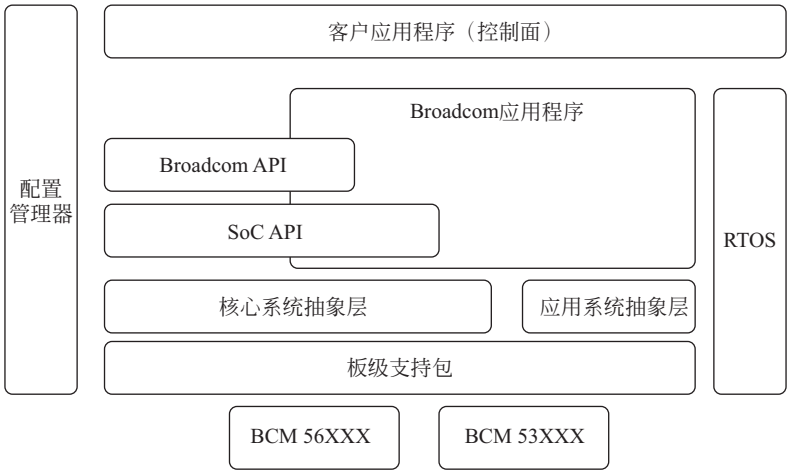

图 1-4　Broadcom Tomahawk 编程接口

Broadcom Tomahawk 交换芯片的流水线功能和顺序是固定的，用户只可以通过 SDK 提供的 API 接口进行配置。Broadcom SDK 已经集成了绝大部分交换机常用的功能，并实现了交换机需要支持的大部分网络协议，开发人员只需在原有框架的基础上，根据自己的实际需要，开启或关闭部分功能，调整某些表项的大小就可以了。对于开发功能完备的交换机来说，这种开发模式是非常高效的。

但是，如果开发者想要灵活地定义流水线的功能，或者想要灵活地分配存储资源，或者想要新增某些功能，这些都是无法轻易实现的。

具体来说，传统交换芯片在可编程性方面有以下两个主要的缺点。

1）传统交换芯片非常固化

传统交换芯片的设计是非常固化的，表的功能、匹配次序以及表的容量在芯片设计时就固定了，并且“匹配 – 动作”（Match-Action）操作只能在报文头部的特定字段上进行。这种方式严重缺乏灵活性。

以表项容量为例，核心路由器需要一个非常大的 32 位 IPv4 路由表，但是一个企业路由器需要一个非常大的 ACL 表，使用同一款传统交换芯片无法同时满足这两种需求。为满足不同需求设计生产不同的芯片是缺乏经济性的，因此设计交换芯片时只会考虑通用需求，不会为每种场景做定制和优化。

2）传统交换芯片支持的操作非常有限

传统交换芯片只支持固定的操作，如转发、丢弃报文、递减 TTL，增加 VLAN 头部、增加 GRE 封装等。这些操作抽象程度低，与协议耦合性强，可扩展性差。

设计并生产一款支持新功能的芯片，周期一般需要 12 ~ 18 个月，所以传统交换芯片如果要支持新的网络协议是需要很长时间的。例如，虚拟扩展局域网（Virtual eXtensible Local Area Network，VXLAN）协议是 VMware 和 Cisco 于 2011 年联合设计的，但是直到 4 年后，市场上才出现第一款支持 VXLAN 的交换机，因为支持 VXLAN 不仅需要软件上增加新的功能，还需要重新设计交换芯片，周期自然就会很长。

1.2.2 可编程交换芯片的设计目标

理想情况下，交换机硬件应能持续使用很多年。在此期间，必然会产生支持新的网络协议或者增加新功能的需求。研究者希望增加交换芯片的可编程性，具体来说，一款全新架构的交换芯片应能满足以下 4 方面的要求。

（1）允许在不重新设计芯片的前提下修改数据面的功能。

（2）在资源允许的范围内，程序员可以定义多个表，表的匹配字段、动作和表项容量都可以通过程序指定，表的数量与次序也可以通过程序指定。

（3）报文头部的定义可以被修改，可以增加新的字段。

（4）可以定义新的操作。

能够实现上述 4 方面要求的交换芯片就可以被称为可编程交换芯片。对于可编程交换芯片，如果需要支持新的网络协议，只需修改软件代码，然后将编译结果重新下发到可编程芯片中，数据面就可以支持新协议了，开发周期缩短到以周为单位，而不像传统交换芯片一样以年为单位。

可编程交换芯片更重要的意义在于全面实现 SDN，即软件定义网络，不仅可以定义网络的控制面，也可以定义网络的数据面。

传统交换芯片的设计是以协议为中心的，计算资源和存储资源都是围绕协议进行组织和分配的。例如，计算资源主要是对报文头部进行各种固定操作，如修改 MAC 地址、递减 TTL、查找路由表等；存储资源主要是 MAC 表、ARP 表、路由表、ACL 表等。

设计一款全新架构的可编程交换芯片，需要对传统交换芯片的架构进行更高层次的抽象，具体如下。

1）对交换芯片计算资源进行更高层次的抽象

从网络协议的角度来看报文，它是 MAC 头部、IPv4 头部、TCP 头部等协议数据，但是从更高层次的抽象角度来看，报文本身不过是二进制数据，于是交换芯片的计算操作便可以抽象成对二进制数据的位操作。二进制位的基本操作数量很少，也比较容易实现，只有与、或、非、异或、移位等。就像搭积木一样，二进制位的基本操作可以组合成稍微复杂一些的加、减、乘等较为复杂的操作，进而可以组合成具有协议处理含义的修改 MAC 地址、递减 TTL、进行 VXLAN 封装等更为复杂的操作。

2）对交换芯片存储资源进行更高层次的抽象

传统交换芯片的存储资源也是以协议为中心的，可划分为 MAC 表、ARP 表、路由表、ACL 表等，并且是固定切分的。但是，在实际场景中，需求各不相同，有些场景需要很大的 MAC 表，有些场景则需要很大的 ACL 表。存储资源的固定切分，势必造成同一款芯片无法灵活满足多个场景的需求。

对存储资源的更高层次的抽象，是将存储资源归一化，都看作是匹配 – 动作表，表的功能、匹配次序、表的容量以及表的数量都可以由程序指定，由编译器根据用户程序进行统一分配，从而实现资源的有效利用，满足不同场景的功能需求。

1.2.3　可编程交换芯片的参考实现——RMT 架构

可编程交换芯片有很多种，典型代表有 Intel 的 Tofino、Broadcom 的 Trident 和 Jericho、Juniper 的 Trio、Cisco 的 Silicon One 等。各个厂商实现可编程交换芯片的架构也各不相同。为了帮助读者深入理解可编程交换芯片的实现原理，本节以 2013 年 Pat Bosshart 等在 SIGCOMM 上发表的论文 *Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN* 中提出的 RMT 架构为例进行介绍。

RMT 虽然只是一个设计模型，并没有实际流片，但是介绍了很多交换芯片的设计细节，可以作为可编程交换芯片的一种有价值的设计参考。

为了介绍 RMT 架构，首先介绍几个基础概念。

1）阶段

流水线（pipeline）是提升芯片处理性能的一种重要技术。为了提升频率，流水线一般会根据指令执行步骤的计算复杂度划分为多个阶段（stage），分为物理阶段和逻辑阶段两个概念。与芯片实现相关的是物理阶段的概念，从程序员的角度看到的是逻辑阶段的概念。

2）包头部向量

在交换芯片中，阶段之间的数据传递，是通过包头部向量（Packet Header Vector，PHV）进行的。PHV 可以理解为 CPU 的寄存器，它有以下的特点。

（1）每个报文有自己独立的 PHV。

（2）PHV 通过一条总线，在流水线的各个阶段之间传递数据。

（3）PHV 的位宽比较大，一般可以达到 4096bit。

（4）PHV 既可以存储报文头部数据，也可以保存元数据。

（5）计算单元可以从 PHV 中读取数据，也可以将计算结果写回 PHV。

本节将从 6 方面来详细介绍 RMT 可编程交换芯片架构。

1. RMT 流水线设计

RMT 是一个单流水线架构，设计运行频率为 1GHz，支持 64 个 10Gb/s 的端口，支持 960Mpps 的转发性能。

RMT 流水线报文的处理过程可以分为 4 个步骤，如图 1-5 所示。

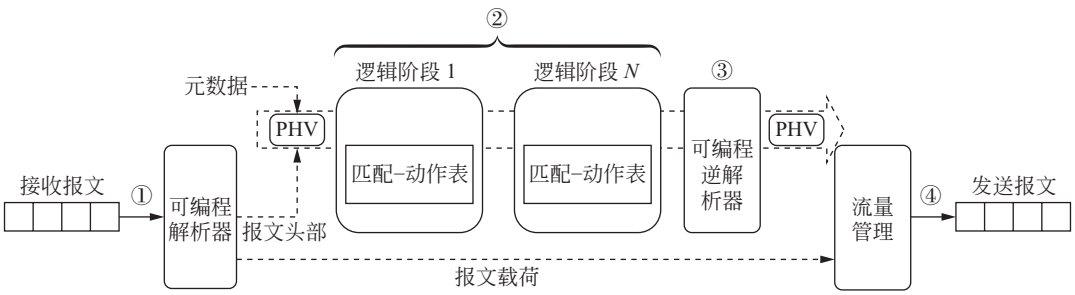

图 1-5　RMT 流水线报文处理过程

（1）报文进入可编程解析器（parser），报文拆分成报文头部和报文载荷两部分。解析器对报文进行解析和过滤，并把特定报文头部数据提取出来，保存到 PHV 中。报文头部的修改，主要是通过程序修改 PHV 中的字段实现的。报文载荷不经过流水线处理，直接到达流量管理模块。

（2）报文头部经过可编程的多级逻辑阶段处理。逻辑阶段是匹配 – 动作表的载体，负责进行报文处理操作。

（3）报文头部经过可编程逆解析器（deparser）处理，生成新的报文头部。

（4）报文头部到达流量管理模块，与报文载荷拼接产生完整的报文，然后发送出去。

RMT 中每一级流水线都可以被看作一个逻辑的匹配 – 动作阶段，每个阶段由表和动作组成，表的匹配字段、动作和表项容量都可以由程序指定。

为了进一步增加流水线的阶段数量，从而增加更多的逻辑，并且为了方便处理多播报文，RMT 在逻辑上将流水线分为入口流水线（Ingress Pipeline）和出口流水线（Egress Pipeline）两部分，各有 32 个阶段。从逻辑上看，在不回环的情况下，一个报文最多可以经过 64 级流水线的处理，从而支持对报文进行复杂的逻辑判断和操作。但是在物理实现上，为了节省硬件资源，这两种流水线共用统一的计算资源和存储资源。编译器会给计算资源和存储资源增加一个标记，来区分是由入口流水线使用还是由出口流水线使用。RMT 芯片架构如图 1-6 所示。

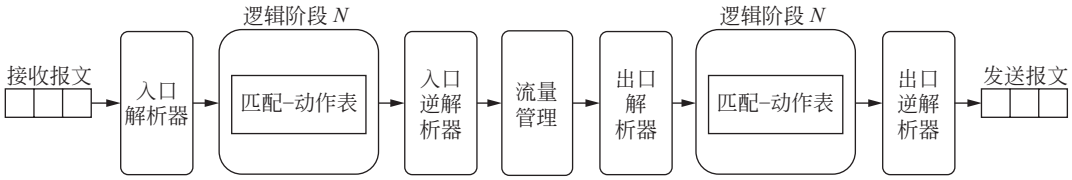

图 1-6　RMT 芯片架构

2. RMT 存储模块设计

为了实现对存储资源的更高层次的抽象，RMT 将存储资源（TCAM、SRAM）归一化，设计成与协议无关的匹配 – 动作表，表的匹配字段、动作和表项容量都可以由程序指定。

TCAM，即三态内容寻址存储器，所谓三态，是指查找结果分为命中状态（hit）、不命中状态（miss）以及不关心状态（don’t care）。TCAM 在交换芯片中一般容量较少，主要用于存储需要最长前缀匹配或者三态匹配的表，如路由表、ACL 表等。

SRAM（Static Random Access Memory），即静态随机存取存储器，SRAM 在交换芯片中一般容量较大，主要用于存储需要精确匹配的表，如 MAC 表、ARP 表等。

存储资源由入口流水线和出口流水线共同使用。每个表都会增加一个标识，用于表示该资源是被入口流水线使用，还是被出口流水线使用。

RMT 设计了 32 个物理阶段，每个物理阶段包含 106 个 SRAM 块，每个 SRAM 块由 1K 个表项组成，每个表项宽度是 112bit。每个物理阶段包含 16 个 TCAM 块，每个 TCAM 块由 2K 个表项组成，每个表项宽度是 40bit。存储资源（TCAM、SRAM）平均分配在 32 个物理阶段上。

为了灵活分配存储资源，在流水线物理阶段的基础上，RMT 抽象出逻辑阶段的概念。

逻辑阶段是面向程序员的概念，它是匹配 – 动作表实现的载体。

假设有两个表，分别将它们命名为表 1 和表 2，其匹配字段分别为 key1 和 key2，动作则分别为 action1 和 action2。这两个表能否存储在同一个阶段的存储空间中，以及能否并行执行，取决于它们之间是否有依赖关系。

RMT 架构中一共分为 4 种依赖关系，如图 1-7 所示。

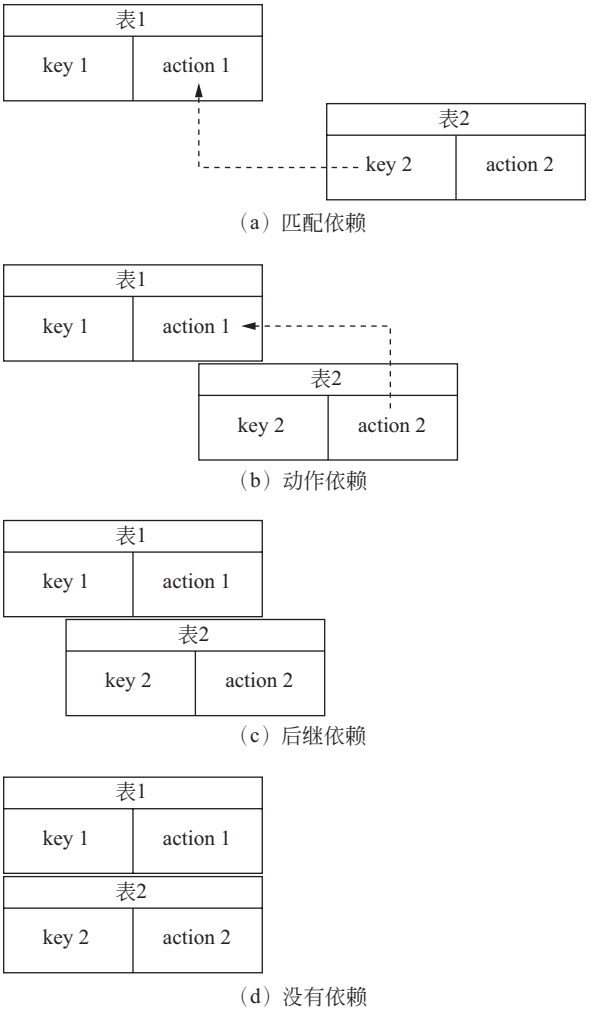

图 1-7　匹配 – 动作依赖关系

（1）匹配依赖。如图 1-7（a）所示，即上一个表的动作会修改下一个表的匹配字段，这两个动作不能并行执行，需要放到两个不同的阶段上执行。假设 action1 会修改 key2 的数据，则表 2 与表 1 之间就产生了匹配依赖。

（2）动作依赖。如图 1-7（b）所示，即两个表的动作会修改同一个 PHV 字段，这两个动作允许部分并行。假设 action1 和 action2 都会修改 PHV 的同一个字段，则表 2 与表 1 之间就产生了动作依赖。

（3）后继依赖。如图 1-7（c）所示，当前匹配阶段的执行依赖于前一阶段的执行结果，就说明两个表存在后继依赖。例如，只有在表 1 命中的情况下才进行表 2 的执行，这样表 2

与表 1 之间就产生了后继依赖。此时表 2 和表 1 可以部分并行执行，但是需要妥善处理分支预测失败后的撤销问题。

（4）没有依赖。如图 1-7（d）所示，表 1 和表 2 之间没有任何关系，可以放到同一个逻辑阶段上并行执行。

逻辑阶段最终需要映射到物理阶段上。一个逻辑阶段可以映射到多个物理阶段上，从而占用更多的存储资源，实现更大的表项；多个逻辑阶段也可以在同一个物理阶段上并行处理，从而节省计算资源。从逻辑上看，相邻的物理阶段资源可以合并，组成更大的表。极端情况下，32 个物理阶段的存储资源可以组成一张最大的表。逻辑阶段到物理阶段的映射如图 1-8 所示。

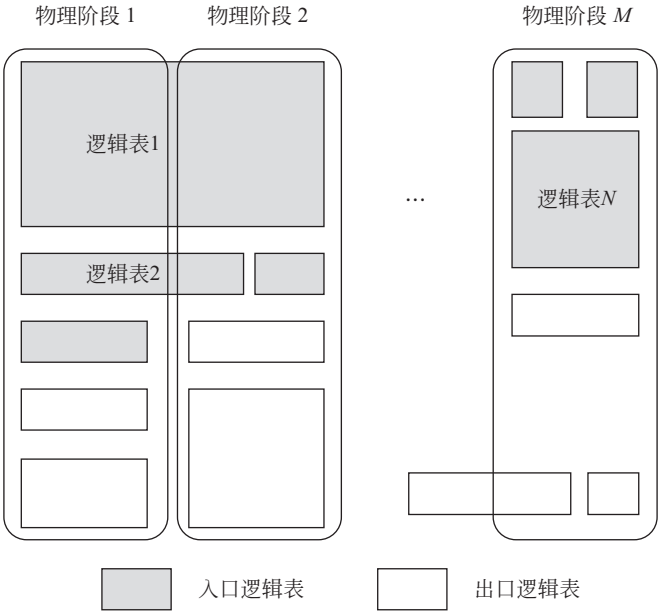

图 1-8　逻辑阶段到物理阶段的映射

每个存储表项都包含一个指针，指向动作指令和动作数据。动作指令定义要执行的动作，由程序指定，动作指令使用单独的存储空间。动作数据包含动作的参数，由控制面指定；动作数据是从每个阶段的 106 个 SRAM 块中分配的，占用 8 个 SRAM 块。

3. RMT 计算模块设计

为了做到协议无关，并对计算资源进行更高层次的抽象，RMT 设计了以位操作为主的指令集。

RMT 支持的指令集如表 1-1 所示，其中 S_i 表示源操作数，V_x 表示 x 是合法的（valid）。

表 1-1　RMT 支持的指令集

指 令 分 类	描　　述
逻辑指令	and、or、xor、not
移位指令	signed/unsigned shift