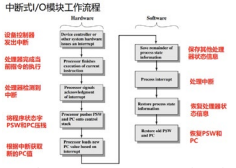
集中式仲裁• 链式询问方式• 计数器定时询问• 独立请求 - 分布式仲裁
替换算法 - 相联映射：可以采用多种替换方法（LRU）最近使用原则、(FIFO)先进先出原则、最少访问频率原则、随机原则
SRAM 和 DRAM - 随机读取存储器 - 易失性的存储器，需要持续供电
- DRAM• 构造简单，模拟信号，集成度高，价格便宜。需要定时刷新。主要用作内存 - SRAM• 构造复杂，数字信号，每一位的体积大，价格昂贵。不需要定时刷新。主要用作 cache

汉明码 $2^r-1>=M+K$
T = b/rN b 需要传送的字节数 N 一个磁道上的字节总数 r 旋转速度
• Ta = Ts + 1/2r + b/rN 总的时间=寻道时间+平均旋转延时+传送时间
• Ts = seek time 寻道时间
• 1/2r is the average Rotational latency 平均旋转延时
I/O 模块的功能：控制和定时，处理器通信，设备通信，数据缓冲，检错处理
编程式 I/O：处理器代表一个进程向 I/O 模块发出 I/O 命令；然后该进程忙等待操作完成后再继续执行。**中断驱动式 I/O：**处理器代表一个进程发出 I/O 命令，继续执行后续指令，并在 I/O 模块完成工作时被中断。如果该进程不需要等待 I/O 完成，后续指令可能在同一进程中。否则，进程挂起等待中断，同时执行其他工作。**DMA：**一个 DMA 模块控制着主存储器和 I/O 模块之间的数据交换。处理器向 DMA 模块发送一个传输一块数据的请求，并且只有在整块数据传输完成后才会被中断。



识别中断设备的方式 - 独立中断线 - 软件轮询 - 硬件轮询 - 总线仲裁
多重中断处理方式 - FIFO - 多个中断线，每个中断线都有优先级 - 硬件或软件轮询，轮询顺序决定了优先级 - 总线仲裁，仲裁方式决定了优先级
在**周期窃取**中，CPU 不需要保存上下文，仅仅是挂起一个周期，然后
可以继续访问总线，进行后续的操作。
操作系统的功能• 程序创建和执行• I/O 设备访问• 文件访问控制• 系统访问• 错误检测和处理• 统计和账务
交换实现对主内存的有效利用以支持进程的进行
虚拟内存 -存储在磁盘上的内存 多道程序设计的运行更加有效，并且避免了主存大小对程序的限制 **优点** 不需要将进程都加载到内存中 同时运行多个进程 运行效率提高
整数的无符号 unsign 表示法 -只有非负值 0 to 2n - 1
整数的符号一幅值表示法 -（2 n-1 - 1）~ （2 n-1 - 1） - 0：+0 和-0
整数的补码表示 -2 n-1 ~ （2 n-1 - 1）
IEEE -(2- 2 -23) x 2 128 and - 2 -127 负数 2-127 and (2- 2 -23) x 2128 正数
指令操作
指令类型 数据处理 数据存储 数据传送 程序流控制
指令集： CPU 能理解的完整的指令集合 • 机器指令典型要素 操作码 源操作数，目的操作 下一指令地址 • **指令设计的要素** { 操作指令表 操作数数量及类型 指令格式 寄存器 寻址方式}
操作数类型 地址、数字、字符、逻辑数据
操作类型 数据传输 算术运算 逻辑运算 转换 I/O 系统控制 控制转移
4 地址多一个下一指令地址 3 个地址：2 个源操作数地址，1 个目的操作数地址 - 2 个地址：一个地址为源操作数和目的操作数复用 - 1 个地址：隐含了第二个操作数地址 - 0 个地址：均为默认，通常使用栈来处理
控制转移指令 - **分支指令**：操作数中包含跳转后的指令地址 - **跳步指令**：跳过下一个需要执行的地址 - **过程调用**指令 它由调用指令调用，并由返回指令返回• 调用和返回需要匹配 • 过程调用可以嵌套 • 需要保存返回地址，还需要传递参数
可存放**操作数的位置** - 主存或虚拟内存 - 寄存器 - 立即数 - I/O
逻辑移位是指将一个字的比特向左或向右移动。在一端，移出的比特将丢失。在另一端，将移入一个 0。算术移位操作将数据视为有符号整数，并不移动符号位。在右**算术移位**中，符号位复制到其右边的比特位置。在左算术移位中，除了符号位之外的所有比特执行逻辑左移，而符号位保持不变。
生成条件 1. 大多数机器提供一个 1 位或多位条件码，这个码是某些操作的结果而设置的。2. 与三地址指令格式一起使用：执行比较并在同一指令中指定分支。
过程嵌套在一个过程中发生的过程调用。
过程返回地址 寄存器，过程的开始，栈顶。
指令格式中的**关键要素** - **操作码宽度**：决定了多少种操作 - **操作数宽度**：影响指令长度 - **寻址模式**：决定了复杂性和指令长度 - **指令长度**： • 操作码种类，操作数数量，寻址模式，地址域宽度，总线带宽，CPU 速度 - **位的分配** • 寻址模式，操作数数量，寄存器还是存储器，地址空间，地址粒度
寻址模式：确定怎么去获得指令中的操作数 • 可寻址的地址范围、寻址的灵活性、寻址的复杂度以及占用的存储单元数量之间进行平衡
影响**指令格式**的三个**关键因素**是哪些？ • 操作码，操作数，寻址模式

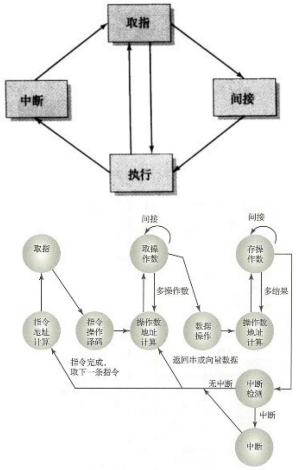
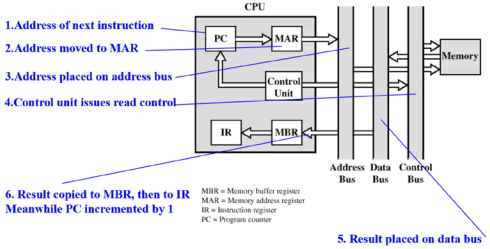


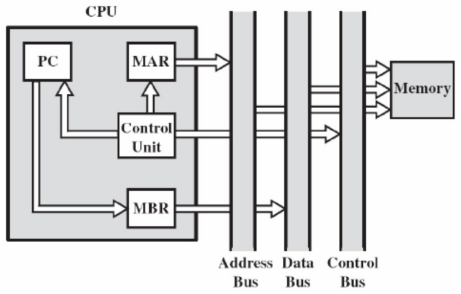
图 14-5 指令周期状态图

立即寻址：操作数的值在指令中。操作数=A
直接寻址：地址字段包含操作数的有效地址。EA=A
间接寻址：地址字段指向内存中的一个字，该字又包含操作数的有效地址。EA=(A)
寄存器寻址：地址字段指向包含操作数的寄存器。EA=R
寄存器间接寻址：地址字段指向一个寄存器，该寄存器又包含操作数的有效地址。
偏移寻址：指令有两个地址字段，至少有一个是明确的。一个地址字段中包含的值（值 = A）被直接使用。另一个地址字段指向一个寄存器，其内容加上 A 以产生有效地址。EA=A+(R)
相对寻址：隐式引用的寄存器是程序计数器（PC）。也就是说，当前指令地址加上地址字段以产生有效地址（EA）。
前变址寻址 - EA=（A+（R）） - 先变址，后间接
后变址寻址 - EA=（A）+（R） - 先间接，后变址
变长指令格式优点：提供一个大量不同操作码长度的操作码集合很容易。寻址可以更灵活，包括各种组合的寄存器和内存引用以及寻址模式。缺点：增加了 CPU 的复杂性。
用户可见寄存器：这些寄存器使机器或汇编语言程序员能够通过优化寄存器的使用来减少对主存储器的引用。（支持**数据类型**：通用；数据；地址；条件码）
控制和状态寄存器：这些寄存器被控制单元用来控制 CPU 的操作，以及被特权操作系统程序用来控制程序的执行。
条件码是由 CPU 硬件作为操作结果设置的位。例如，算术操作可能产生正数、负数、零或溢出结果。除了结果本身被存储在寄存器或内存中，还会设置一个条件码。这个码随后可能作为条件分支操作的一部分被测试。
所有 CPU 设计都包括一个寄存器或一组寄存器，通常称为**程序状态字**（PSW），其中包含状态信息。PSW 通常包含条件码和其他状态信息。
流水线
流水线冒险：数据依赖、控制依赖和资源依赖发生的时候，流水线会因此而暂停运行，称为流水线冒险 - 资源冒险：流水线中的两个指令都需要使用同一个资源而引起 **处理方法**：1. 增加资源；2. 串行执行 - 数据冒险：对同一个操作数的访问出现冲突（反相关（读后写）、输出相关（写后写） • 增加寄存器 • 用不同的寄存器，避免冲突 - 真相关（写后读） • 延迟 }- 控制冒险：经常称为**分支冒险**。程序中有分支转移指令，而流水 线在取指的时候，没有正确判断转移的结果，导致获取了错误的指令； **处理方法**： • 多指令流（有两个流水线给每个分支 在每个独立的流水线中预取一个分支的指令 最后根据分支条件确定使用哪一个流水线）• 预取分支目标（除了分支之后的指令外，还预取了分支的目标 预取之后并不执行，只是进行取指和解码 保留目标直到分支被执行 ） • 环形缓冲（快速的存储 包含 n 条最近顺 序取过来的指 在转移可能要发生的时候，先检查转移目标是否在缓冲器里面）• 分支预测（ - 静态预测 • 预测从不发生 • 预测总是发生 • 根据操作码预测 - 动态预测 • 根据条件转移指令的历史转移情况进行预测 • 基于关联的预测） • 延迟分支
流水线中断的原因 - 资源竞争 - 相关性 • 数据相关性 • 控制流 - 长延迟操作
指令流水线和**理想**的流水线之间存在一些**差距**，主要表现在以下几个方面：(1)执行时间通常会比取指时间长。执行将涉及读取和存储操作数以及某些操作的执行。因此，取指阶段可能必须等待一段时间才能清空其缓冲区。 (2)条件分支指令使得下一条要取指的指令地址变得未知。因此，取指阶段必须等到它从执行阶段接收到下一条指令的地址。然后，执行阶段可能必须等待，同时取指下一条指令。
CISC 指令操作码种类多 - 指令长度可 - 寻址模式多种多样
RISC 组织典型特征 (1) 使用固定格式的有限指令集，(2) 大量的寄存器或使用优化寄存器使用的编译器，以及 (3) 重视优化指令流水线。
RISC 机器上用于**减少寄存器存储器**操作：一种基于软件，另一种基于硬件。软件方法依赖于编译器来最大化寄存器使用。编译器将尝试将寄存器分配给在给定时间内将被最多使用的变量。这种方法需要使用复杂的程序分析算法。硬件方法简单地使用更多的寄存器，这样就可以让更多的变量在寄存器中保持更长时间。
RISC 采用**硬布线**的方式来实现
环形寄存器 管理全局变量(1) 在高级语言中声明为全局的变量可以由编译器分配内存位置，所有引用这些变量的机器指令将使用内存引用操作数。 (2) 在处理器中集成一组全局寄存器。这些寄存器数量固定，可供所有程序使用。
RISC 指令集体系结构的典型特征 每个周期一条指令。寄存器到寄存器的操作。简单的寻址模式。简单的指令格式。
完整的指令周期 取指周期 执行周期 间接周期 中断周期
指令预取 取指需要访问内存 执行一般不需要访问内存 在执行当期指令的时候，取下一个指
六阶段流水线 取指 指令译码 计算操作数地址 取操作数 执行指令 写结果
指令 4~7 已经完成的处理作废 • 需要重新开始指令 15 的取指 • 第 9 到第 12 时间片，没有指令 完成执行，称为**分支惩罚**
延迟分支 在分支指令后面安排一条有用的指令来代替仅为延迟的空操作
循环展开 - 复制循环体 - 降低迭代次数，减少循环开销 - 提高指令并行性 - 降低存储器访问次数
超标量
超标量处理器是指使用多个独立指令流水线的处理器。每个流水线包括多个阶段，因此每个流水线可以同时处理多条指令。多流水线引入了新的并行级别，使得可以同时处理多个指令流。
超级流水线利用了许多流水线阶段执行的任务需要不到半个时钟周期的事实。因此，内部时钟速度加倍可以在一个外部时钟周期内完成两项任务。
指令级并行性是指程序中的指令可以并行执行的程度
当序列中的指令相互独立并且可以通过重叠并行执行时，就存在**指令级并行性**。
机器并行性是处理器利用指令级并行性的能力的衡量。机器并行性由处理器可以同时获取和执行的指令数量（并行流水线的数量）以及处理器用于查找独立指令的机制的速度和复杂程度决定。
真实数据依赖：第二条指令需要第一条指令产生的数据。**程序依赖**：分支之后的指令（无论分支是否被执行）对分支有程序依赖，不能在执行分支之前执行。
资源冲突：资源冲突是两条或多条指令同时竞争相同资源的情况。**输出依赖**：两条指令更新同一个寄存器，因此后面的指令必须晚些更新。**反依赖**：第二条指令破坏了第一条指令使用的值。
按顺序发射与按顺序完成：按照顺序执行所能达到的确切顺序发射指令，并以同样的顺序写入结果。**按顺序发射与乱序完成**：按照顺序执行所能达到的确切顺序发射指令，但允许指令乱序完成。**乱序发射与乱序完成**：处理器具有前瞻能力，允许它识别可以带入执行阶段的独立指令。指令的发射几乎不考虑它们原始的程序顺序。指令也可能乱序完成
对于乱序发射策略，**指令窗口**是一个缓存区，用于保存解码后的指令。这些指令可以从指令窗口以最方便的顺序发射。
寄存器重命名寄存器由处理器硬件动态分配，并且与不同时刻指令所需的值相关联。当创建新的寄存器值时（即执行具有寄存器为目标操作数的指令时），将为该值分配新寄存器。
限制指令级并行的最重要的原因就是程序中指令之间的相关性 指令之间的相关性包括 真实数据相关 输出相关性反相关性 过程相关性（分支前和分支后的指令不能并行执行） 资源冲突
限制并行的关键因素 - 指令并行性 - 编译器能力 - 硬件能力
超标量处理器中提高性能的技术 资源复制 乱序发射 重命名
- **超标量**流水线的**本质**？从本质上来说，超标量流水线是在不同的流水线上独立执行指令的能力，一种真正的指令并行处理技术
- **超标量**会带来哪些**问题**？超标量实现带来了许多与 指令流水线相关的复杂设计问题。流水线本身的相关性问题依然存在；多流水线带来更复杂的相关性问题。需要编译器具有更复杂的优化技术，以达到更大程度的指令级并行性
- 简述**机器并行性、指令发射策略和重命名**之间的关系：机器并行性（处理器获取指令级并行性好处的能力）定义了处理器的并行能力，指令发射策略（指令发射就是寻找能够进入流水线并执行的指令的策略）决定了这些并行操作如何被组织和优化，而重命名技术（解决硬件级别假数据依赖（如输出依赖和反向依赖）的技术，通过创建额外的寄存器别名来实现）解决了并行执行中可能出现的数据依赖问题。
微程序控制
控制器实现有两种方式 -硬布线实现 -微程序控制实现
- 什么是**微操作**？ 计算机执行程序来完成用户指定的功能，程序包括若干指令，指令执行包括若干个周期，如取 指周期、执行周期、间接周期等，每个周期有一系列步骤，称为微操作，每一步做很小的操作
- **硬布线**方法的**设计视角**是什么？ 硬布线是从控制门的角度去设计
- **微程序**的**设计视角**是什么？从微操作的角度考虑，分析每个微操作相关的控制门
控制器的**硬布线**实现方式与**微程序**实现方式有何**不同?**硬布线控制单元是一种组合电路，其中输入逻辑信号被转换成一组输出逻辑信号，这些信号作为控制信号的功能。在微程序控制单元中，逻辑由一个微程序来指定。微程序由微编程语言中的一系列指令组成。这些都是非常简单的指令，用于指定微操作。
控制存储器有何作用？ 控制存储器包含一组微指令，这些微指令定义了控制单元的功能。
水平微指令与**垂直**不同？ 在水平微指令中，控制字段的每一位都连接到一个控制线上，存储字宽，高并发。在垂直微指令中，每个要执行的动作都使用一个代码，解码器将这个代码转换成单独的控制信号，存储字较短，高并有限，速度慢。
微程序控制器完成的**基本任务**是什么？ 微指令顺序：从控制存储器获取下一个微指令。微指令执行：生成执行微指令所需的控制信号。
指令和**微操作**的关系是什么？ 微操作是在一个时钟脉冲期间执行的基本 CPU 操作。一条指令由一系列微操作组成。
控制器要完成的**基本任务**有哪些？ 顺序控制：控制单元使处理器根据正在执行的程序，按正确的顺序逐步执行一系列微操作。 执行：控制单元使每个微操作得以执行
简要说明控制器的**硬布线**实现是什么意思？ 在硬布线实现中，控制单元本质上是一个组合电路。它的输入逻辑信号被转换成一组输出逻辑信号，这些信号就是控制信号。

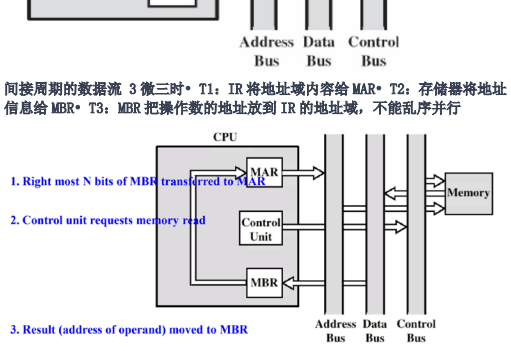
- 取指的数据流

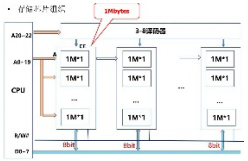


中断周期的数据流 4 微三时 - T1：PC 将下一个指令地址给 MBR - T2：操作系统将保存 PC 的存储器地址信息给 MAR。同时，中断处理程序把起始地址给 PC - T3：MBR 把 PC 之前给它的地址存入存储器



间接周期的数据流 3 微三时• T1：IR 将地址域内容给 MAR• T2：存储器将地址信息给 MBR• T3：MBR 把操作数的地址放到 IR 的地址域，不能乱序并行





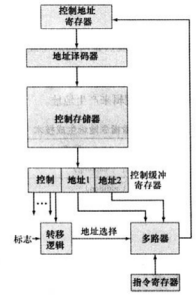


图 16-6　转移控制逻辑：双地址字段

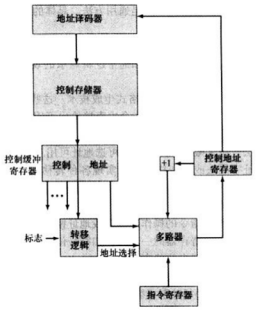


图 16-7　转移控制逻辑：单地址字段

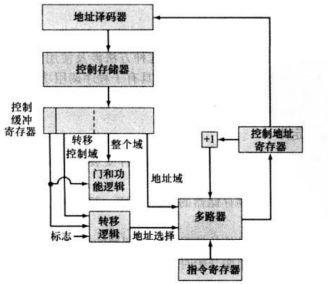


图 16-8　转移控制逻辑：可变格式