C.4 控制结构

微体系结构的控制结构由其状态机指定。如前所述，状态机(图 C.2 和图 C.7)确定每个时钟周期需要哪些控制信号来处理数据路径中的信息，以及每个时钟周期需要哪些控制信号来引导控制流从当前的活动状态到它的后续状态。

这里我们选择了一个简单的微程序实现。**控制结构的每个状态需要 42 位来控制数据路径中的处理，需要 10 位来帮助确定下一个状态是什么。这 52 位被统称为微指令。**每个微指令(即状态机的每个状态)都存储在一个称为控制存储的特殊内存的 52 位位置中。由于每个状态对应于控制存储器中的一条微指令，我们的微程序实现的控制存储器需要 6 位来指定每条微指令的地址。这六个位对应于与状态机中的每个状态相关联的状态号。例如，与状态 18 相关联的微指令是存储在控制存储器地址 18 中的 52 个控制信号的集合。

表 C.2 列出了 10 位控制信息的功能，这些信息有助于确定接下来的状态。**图 C.5 显示了微测序器的逻辑**。微序器的目的是确定控制存储器中对应下一个状态的地址，即下一个状态的 52 位控制信息存储的位置。

正如我们所说，状态机的状态 32(图 C.2)执行指令周期的 DECODE 阶段。它有 16 个“next”状态，这取决于当前指令周期中执行的 LC-3 指令。如果状态 32 对应的微指令中的 IRD 控制信号为 1，则微序列器的输出 MUX(图 C.5)将从 00 形成的 6 位与四个操作码位 IR[15：12]串联起来，获得其源。由于 IR[15：12]指定了当前正在处理的 LC-3 指令的操作码，所以控制存储的下一个地址将是 16 个地址中的一个，对应于这 15 个操作码加上一个未使用的操作码 IR[15：12] = 1101。也就是说，在状态 32 之后的 16 个状态，每一个都是在状态 32 被解码后要执行的第一个状态。例如，如果正在处理的指令是 ADD，下一个状态的地址是状态 1，它的微指令存储在位置 000001。回想一下，ADD 的 IR[15：12]是 0001。

如果指令无意中包含了未使用的操作码 IR[15：12] = 1101，则微架构将执行一系列微指令，从状态 13 开始。这些微指令会对获取了非法操作码的指令做出反应。C.7.3 节描述了这种情况下会发生什么。

控制数据路径和微序器所必需的几个信号不在表 C.1 和 C.2 中列出的那些信号中。它们是 DR、SR1、BEN、INT、ACV 和 r。图 C.6 显示了生成 DR、SR1、BEN 和 ACV 所需的附加逻辑。

INT 信号由正常指令处理外部的某个事件提供，表示正常指令处理应该被中断，并处理这个外部事件。中断机制将在第 9 章中描述。微体系结构中相应的控制流在 C.7 节中描述。

剩余的信号 R 是由内存产生的信号，以允许 LC-3 在需要多个时钟周期读取或存储一个值的内存中正确操作。

假设读取一个值需要 5 个内存周期。也就是说，一旦 MAR 包含了要读取的地址，并且微指令断言 read，那么在将内存中指定位置的内容加载到 MDR 中之前，将需要 5 个周期。(注意微指令通过两个控制信号来断言 READ： MIO。EN /YES，R.W / RD；参见图 C.3)。

回忆一下我们在 C.2 节中讨论的状态 28 的功能，它在每个指令周期的 FETCH 阶段从内存中访问一条指令。如果内存需要 5 个周期来读取一个值，LC-3 要正确操

作，状态 28 必须执行 5 次才能进入状态 30。也就是说，在 MDR 包含由 MAR 内容指定的内存位置的有效数据之前，我们希望状态 28 继续重新执行。在 5 个时钟周期之后，内存已经完成了“读取”，从而在 MDR 中产生了有效的数据，因此处理器可以继续运行到状态 30。如果微架构在状态 30 之前没有等待内存完成读取操作呢?由于 MDR 的内容仍然是垃圾，因此微架构将垃圾放入状态为 30 的 IR 中。

就绪信号(R)使内存读取能够正确执行。由于内存知道它需要 5 个时钟周期来完成读取，它在整个第五个时钟周期中断言一个就绪信号(R)。图 C.2 显示下一个状态是 28(即 011100)，如果在当前时钟周期内内存读取没有完成，则下一个状态是 30(即 011110)。正如我们所看到的，这是微序列器的工作(图 C.5)产生下一个状态地址。

状态 28 的 10 个微定序器控制信号为：

```
IRD/0       ; NO
COND/001    ; Memory Ready
J/011100
```

有了这些控制信号，微序器生成下一个状态地址是什么?对于状态 28 的前四次执行，由于 R = 0，下一个状态地址是 011100。这将导致状态 28 在下一个时钟周期中再次执行。在第五个时钟周期中，由于 R = 1，下一个状态地址是 011110，LC-3 移动到状态 30。请注意，为了使内存中的就绪信号(R)成为下一个状态地址的一部分，COND 必须设置为 001，这允许 R 通过它的 4 输入与门。

C.5 TRAP 指令

正如我们已经说过的，每条 LC-3 指令在其指令周期中从状态 32 到最终状态遵循自己的路径，然后返回状态 18 以开始处理下一条指令。以 TRAP 指令的指令周期为例，如图 C.7 所示。

回想一下，TRAP 指令将 PSR 和 PC 压入系统堆栈，用 TRAP 服务程序的起始地址加载 PC，然后从特权内存中执行服务程序。

从状态 32 开始，DECODE 之后的下一个状态是状态 15，与 TRAP 指令操作码 1111 一致。在状态 15 中，用于形成 trap 向量表项 MAR[15：8]的 Table 寄存器被加载了 x00， PC 被增加(我们将马上看到原因)，MDR 被加载了 PSR，准备将其推入系统堆栈。控制转到状态 47。

在状态 47 中，陷阱向量（IR[7：0]）被加载到八位寄存器向量中，由于陷阱服务例程在特权内存中执行，因此 PSR[15]被设置为监控模式，并且状态机分支到状态 37 或 45，这取决于执行陷阱指令的程序是处于用户模式还是监控模式。如果处于用户模式，状态 45 将用户堆栈指针保存在保存的 USP 中，从保存的 SSP 加载堆栈指针，然后继续到状态 37，处理器开始将 PSR 和 PC 推送到堆栈上。如果执行陷阱指令的程序已经处于特权模式，则不需要状态 45。

在状态 37 和 41 中，PSR 被推送到系统堆栈上。在状态 43、46 和 52 中，PC 被推到系统堆栈上。注意，在状态 43 中，PC 在被推到堆栈上之前被递减。这在处理中断和异常时是必要的，这将在第 C.7 节中解释。这对于处理陷阱指令是不必要的，这就是 PC 在状态 15 中递增的原因。

剩下的唯一一件事就是用陷阱服务例程的起始地址加载 PC。这是通过在陷阱向量表中加载带有正确条目地址的 MAR 来完成的，该地址是通过连接表和向量（状态 54）获得的，将起始地址从内存加载到 MDR（状态 53）中，并加载 PC（状态

55）。这就完成了陷阱指令的执行，控制返回到状态 18，开始处理下一条指令——在本例中是陷阱服务例程的第一条指令。

每个 trap 服务程序的最后一条指令是 RTI(从 trap 或中断返回)。从状态 32 的 DECODE 开始，RTI 的下一个状态是状态 8，与它的 8 位操作码 1000 一致。在状态 8、36 和 38 中，PC 从系统堆栈中弹出并加载到 PC 中。在状态 39、40、42 和 34 中，PSR 从系统堆栈中弹出并加载到 PSR 中。这将使 PC 和 PSR 返回到执行 trap 服务程序之前的值。最后，如果调用 TRAP 指令的程序处于用户模式，PSR[15]必须返回到 1，保存 Supervisor Stack Pointer，并将 User Stack Pointer 加载到 SP 中。这在状态 59 完成，完成 RTI 的指令周期。

C.6 内存映射 I/O

正如你在第 9 章中所知道的，LC-3 ISA 通过内存映射的 I/O 执行输入和输出，也就是说，使用相同的数据移动指令来读写内存。LC-3 通过给每个设备寄存器分配一个地址来实现这一点。输入由有效地址为输入设备寄存器地址的加载指令完成。输出由有效地址为输出设备寄存器地址的存储指令完成。例如，在图 C.2 的状态 25 中，如果 MAR 中的地址是 xFE02，则 MDR 由 KBDR 提供，数据输入将是输入的最后一个键盘字符。另一方面，如果 MAR 中的地址是合法的内存地址，则 MDR 由内存提供。

图 C.2 中的状态机不需要改变以适应内存映射的 I/O。然而，必须确定什么时候访问内存，什么时候访问 I/O 设备寄存器。这是地址控制逻辑(ADDR.CTL.LOGIC)的工作，如图 C.3 所示。

表 C.3 是地址控制逻辑的真值表，此表基于(1)MAR 的内容，(2)这个周期是否访问内存或 I/O (MIO)，显示产生了什么控制信号。(3)是否请求 load (R.W/Read)或 store (R.W/Write)。

注意，对于内存映射加载，数据可以从以下四个来源之一提供给 MDR：内存、KBDR、KBSR 或 DSR。地址控制逻辑为 INMUX 提供适当的选择信号。对于内存映射存储，可以将 MDR 提供的数据写入内存、KBSR、DDR 或 DSR。地址控制逻辑为相应的结构提供适当的使能信号。

C.7 中断和异常控制

最后的状态机需要完成 LC-3 故事是那些控制中断的初始化的状态，这些状态，控制从一个中断返回(RTI 指令)，和那些状态控制的三个异常的启动指定的 ISA。

中断和异常非常相似。两者都停止当前正在执行的程序。两者都将中断程序的 PSR 和 PC 推到系统堆栈上，从中断向量表中获取中断或异常服务程序的起始地址，并将该起始地址加载到程序计数器中。中断和异常之间的主要区别在于使正在执行的程序停止的事件。中断是通常与正在执行的程序无关的事件。异常是由于正在执行的程序中出现错误而直接导致的事件。**LC-3 规定了三种异常：特权模式违规、非法操作码和 ACV 异常。**图 C.7 显示了执行这些操作的状态机。图 C.8 显示了在图 C.3 中添加了使中断和异常处理工作所需的附加结构之后的数据路径。

章节 C.7.1 描述了启动中断所需的处理流程。7.3 节描述了发起异常所需的处理

流程。

C.7.1 启动中断

当程序执行时，一些外部事件可以请求中断，这样指令的正常处理就可以被抢占，控件就可以把注意力转向处理中断。外部事件通过断言它的中断请求信号来请求中断。记得从第 9 章，如果设备维护它的中断请求信号的优先级高于当前执行的程序的优先级和任何其他外部中断请求断言同时，INT 断言，INTV 加载外部事件的相对应的中断向量。微处理器通过启动中断响应 INT。也就是说，如果处理器不是处于 Supervisor 模式，它会将自己置于 Supervisor 模式，将中断进程的 PSR 和 PC 推入到 Supervisor 堆栈中，并将中断服务程序的起始地址加载到 PC 中。PSR 包含程序的特权模式 PSR[15]、优先级 PSR[10：8]和条件编码 PSR[2：0]。重要的是，当处理器恢复中断程序的执行时，特权模式、优先级和条件代码将恢复到中断发生时的状态。

从图 C.2 中可以看到，在状态 18 中，当 MAR 加载了 PC 的内容，并且 PC 被递增时，测试的是 INT。

状态 18 是处理器检查中断的唯一状态。测试状态 18 的原因是简单的：一旦一个 LC-3 指令开始处理，更容易让它完成其完整的指令周期(取回，解码等)比中间打断它，需要留意的是多么远当外部设备一个中断请求(例如，断言 INT)。如果 INT 只在状态 18 中被测试，那么当前的指令周期可以提前终止(甚至在指令被取走之前)，并且控制直接启动中断。

该测试是由控制信号，COND5， 101 只有在状态 18，允许值 INT 通过其 4 输入和门，如图 C.5 所示，贡献下一个状态的地址。因为 COND 信号在任何其他状态下都不是 101，所以 INT 在任何其他状态下都没有作用。

在状态 18 中，10 个微序器控制位如下：

```
IRD/0      ; NO
COND/101   ; Test for interrupts
J/100001
```

如果 INT = 1，与门的输出就会产生一个 1，这使得下一个状态地址不是 100001，对应状态 33，而是 110001，对应状态 49。这就开始了中断的启动(参见图 C.7)。

有几个函数在状态 49 中执行。PSR 包含被中断程序的特权模式、优先级和条件代码，它被加载到 MDR 中，以便将其推入管理器堆栈。PSR[15]被清除，反映到特权模式的改变，因为所有中断服务程序都在特权模式下执行。记录中断设备提供的 3 位优先级和 8 位中断向量(INTV)。PSR[10：8]加载中断设备的优先级。内部寄存器 Vector 装入 INTV， 8 位寄存器 Table 装入 x01，准备访问中断向量表以获得中断服务例程的起始地址。最后，处理器测试旧的 PSR[15]，以确定是否必须调整堆栈指针之前，压 PSR 和 PC 进栈。

如果旧的 PSR[15] = 0，则处理器已经在特权模式下运行。R6 是管理器堆栈指针(SSP)，因此处理器立即继续到状态 37 和 41，将被中断程序的 PSR 推入管理器堆栈。如果 PSR[15] = 1，则中断的程序处于 User 模式。在这种情况下，用户堆栈指针(USP)必须保存在已保存 USP 和 R6 必须加载已保存 SSP 的内容，然后移动到状态 37。这是在状态 45 做的。

从状态 49 到状态 37 或状态 45 的控制流由 10 个微序器控制位启用，如下所示：

```
IRD/0       ; NO
COND/100    ; Test PSR[15], privilege mode
J/100101
```

如果 PSR[15] = 0，控制转到状态 37 (100101)；如果 PSR[15] = 1， control 变为状态 45(101101)。

在状态 37 中，R6 (SSP)被减少(为推送做准备)，MAR 被加载了新的堆栈顶部的地址。

状态 41 时，内存使能 WRITE (MIO.EN /YES，R.W /WR)。当写操作完成时(以 R = 1 为信号)，PSR 被推到管理器堆栈上，流移动到状态 43。

状态 43 时，PC 被加载到 MDR 中。请注意，状态 43 说 MDR 装载了 PC-1。回想一下，在状态 18 中，在被中断指令的指令周期开始时，PC 被递增。用 PC-1 加载 MDR 将 PC 调整到中断程序的正确地址。

在状态 46 和状态 52 中，与状态 37 和状态 41 发生相同的顺序，只是这一次被中断程序的 PC 被推到管理器堆栈上。

完成中断启动的最后一个任务是将中断服务程序的起始地址加载到 PC 上。这是由状态 54、53 和 55 执行的。它的实现方式类似于用一个 TRAP 服务程序的起始地址加载 PC。引起 INT 请求的事件提供了与中断相关的 8 位中断向量 INTV，类似于 trap 指令中包含的 8 位 trap 向量。这个中断向量存储在 8 位寄存器 INTV 中，如图 C.8 所示的数据路径。

中断向量表占用 x0100 到 x01FF 的内存位置。在状态 54 中，在状态 49 中被加载到 vector 中的中断向量与中断向量表的基址(x0100)相结合，被加载到 mar 中。在状态 53 中，内存为 READ。当 R = 1 时，读取已经完成，MDR 包含中断服务程序的起始地址。在状态 55 中，PC 被装入该起始地址，完成中断的启动。

需要强调的是，LC-3 支持两个堆栈，一个用于每个特权模式，两个堆栈指针(USP 和 SSP)，一个用于每个堆栈。R6 是栈指针，当权限从 User 模式切换到 Supervisor 模式时从 Saved SSP 加载，当权限从管理员模式切换到用户模式时从 Saved USP 加载。当特权模式更改时，R6 中的当前值必须存储在适当的“Saved”堆栈指针中，以便下次特权模式更改回来时可用。

C.7.2 从中断或陷阱服务程序返回，RTI

中断服务程序，就像前面描述的 trap 服务程序一样，以 RTI 指令的执行结束。RTI 指令的任务是将计算机恢复到中断或 trap 服务程序执行之前的状态。这意味着恢复 PSR(即特权模式、优先级和条件代码 N、Z、P 的值)并恢复 PC。这些值在中断启动或执行 TRAP 指令时被压入堆栈。因此，它们必须以相反的顺序弹出堆栈。

DECODE 之后的第一个状态是状态 8。在这里，我们用管理器堆栈顶部的地址来加载 MAR，它包含了最后一个被推送的东西(之后还没有被弹出)——当中断被启动时 PC 的状态。同时，我们测试了 PSR[15]，因为 RTI 是一个特权指令，只能在 Supervisor 模式下执行。如果 PSR[15] = 0，我们可以继续进行 RTI 的要求。

状态 36 和 38 恢复 PC 的值，当中断被启动时。在状态 36 中，读取内存。当读取完成时，MDR 包含了当中断发生时将要被处理的指令的地址。状态 38 将地址加载到 PC 中。

状态 39、40、42 和 34 将特权模式、优先级和条件码(N、Z、P)恢复到原来的值。在状态 39，管理器堆栈指针增加，以便在 PC 弹出后，它指向堆栈的顶部。MAR 是用新的堆栈顶部的地址加载的。状态 40 启动内存 READ；当 READ 完成时，MDR 包含中断的 PSR。状态 42 从 MDR 加载 PSR，状态 34 增加堆栈指针。

剩下的唯一一件事就是检查被中断程序的特权模式，看看是否必须切换堆栈指针。在状态 34 中，微序器控制位如下：

```
IRD/0       ; NO
COND/100    ; Test PSR[15], privilege mode
J/110011
```

如果 PSR[15] = 0，则控制流到状态 51(110011)在一个周期内不做任何操作。如果 PSR[15] = 1，控制流到状态 59，其中 R6 保存在 saved SSP 中，R6 从 saved USP 中加载。在这两种情况下，控制返回状态 18，开始处理下一条指令。

C.7.3 处理一个异常

LC-3 定义了三种情况，在这些情况下，由于执行程序中出现错误，处理不允许正常继续。我们将这些情况称为例外。它们的启动方式与中断的启动方式相同，通过将 PSR 和 PC 机推入系统堆栈，从中断向量表中获取异常服务例程的起始地址，并将该地址加载到 PC 中，启动异常服务例程。

在 LC-3 中声明的三个异常是(1)特权模式异常引起的程序试图在用户模式下执行 RTI 指令，(2)非法操作码异常引起的程序试图执行操作码为 1101 的指令，(3)访问控制违反(ACV)异常，由程序在用户模式下试图访问特权内存位置引起。

C.7.3.1 Privilege Mode Exception

如果处理器处于 User 模式(PSR[15] = 1)并试图执行 RTI，则会发生特权模式异常。处理器将 PSR 和 RTI 指令的地址推入管理器堆栈，并将处理特权模式违规的服务例程的起始地址装入 PC。图 C.7 显示了流程，如果 PSR[15] = 1，从状态 8 到状态 44 的分支开始。

在状态 44 中，加载 8 位的 Table 寄存器 x01，表示中断向量表中一个表项的地址；加载 8 位的 vector 寄存器 x00，表示中断向量表中的第一个表项。x0100 的内容是处理特权模式异常的服务例程的起始地址。MDR 与导致异常的程序的 PSR 一起加载，以准备将其推入系统堆栈。最后，PSR[15]被设置为 0，因为服务例程将使用管理器特权执行。然后处理器移动到状态 45，它遵循与中断启动相同的流程。

这个流和中断起始流之间的主要区别是状态 54，其中 MAR 是用 x01’vector 加载的。在中断的情况下，用 INTV 加载状态为 49 的 Vector， INTV 由中断设备提供。在特权模式冲突的情况下，Vector 以 x00 的状态加载。

在状态 49 中执行的另外两个函数在状态 44 中没有执行。首先，根据中断设备的优先级改变优先级。在处理特权模式冲突时，我们不会改变优先级。服务例程的执行优先级与引起冲突的程序相同。其次，对中断执行一个测试，以确定特权模式。对于特权模式冲突，这是不必要的，因为处理器已经知道它是在 User 模式下执行的。

C.7.3.2 Illegal Opcode Exception

虽然这种情况很少见，但我们假设，用机器语言编写程序的程序员可能会错误地包含 opcode = 1101 的指令。由于在 LC-3 ISA 中没有这样的操作码，计算机不能处理该指令。状态 32 执行 DECODE，下一个状态是状态 13。

处理器采取的动作与特权模式异常非常相似。程序的 PSR 和 PC 被推到管理器堆栈上，PC 被加载非法 Opcode 异常服务程序的起始地址。

状态 13 与状态 44 非常相似，状态 44 开始启动特权模式异常。有两个区别：(1)Vector 是用 x01 加载的，因为非法的 opcode 异常的服务例程的起始地址在 x0101 中。(2)在特权模式异常的情况下，我们知道，当处理器试图执行 RTI 指令时，程序处于用户模式。在非法操作码的情况下，处理器可以处于任意一种模式，因此从状态 13 到状态 37 或状态 45，这取决于遇到非法操作码指令时，程序是在 Supervisor 模式还是 User 模式下执行。

与状态 44 一样，正在运行的程序的优先级不会改变，因为处理异常的紧迫性与执行包含该异常的程序的紧迫性相同。与状态 49 一样，状态 13 测试包含非法操作码的程序的特权模式，因为如果当前执行的程序处于用户模式，则需要按照章节 C.7.1 所述切换堆栈指针。和状态 49 一样，如果堆栈指针已经指向管理器堆栈，处理器就微分支到状态 37；如果堆栈指针必须切换到状态 45。从这里开始，初始化序列以 37、41、43 等状态继续进行，与中断被初始化(章节 C.7.1)或特权模式异常被初始化(章节 C.7.3.1)所发生的情况相同。PSR 和 PC 被推到管理器堆栈上，服务例程的起始地址被加载到 PC 上，完成异常的启动。

C.7.3.3 Access Control Violation (ACV) Exception

如果处理器试图在用户模式下访问特权内存，就会发生访问控制冲突(ACV)异常。状态机在处理器访问内存的每一种情况下(即状态为 17、19、23、33 和 35)都检查这一点。如果发生 ACV 冲突，下一个状态分别是 56、61、48、60 和 57(见图 C.2)。在所有五个状态中，处理器装载 Table (x01)， Vector (x02)， MDR (PSR)，将 PSR[15]设置为 0，与状态 44 完全相同，只有一个例外。Vector 被设置为 x02，因为 ACV 异常服务例程的起始地址在内存位置 x0102。处理过程与状态 44 完全一样，首先移动到状态 45 切换到系统堆栈，然后将 PSR 和 PC 推到堆栈上，并将服务例程的起始地址加载到 PC 上。