程序(exception handler)，这是操作系统的一部分，但是实现异常处理的这部分超出了本书讲述的范围。

在一个流水线化的系统中，异常处理包括一些细节问题。首先，可能同时有多条指令会引起异常。例如，在一个流水线操作的周期内，取指阶段中有 halt 指令，而数据内存会报告访存阶段中的指令数据地址越界。我们必须确定处理器应该向操作系统报告哪个异常。基本原则是：由流水线中最深的指令引起的异常，优先级最高。在上面那个例子中，应该报告访存阶段中指令的地址越界。就机器语言程序来说，访存阶段中的指令本来应该在取指阶段中的指令开始之前就结束的，所以，只应该向操作系统报告这个异常。

第二个细节问题是，当首先取出一条指令，开始执行时，导致了一个异常，而后来由于分支预测错误，取消了该指令。下面就是一个程序示例的目标代码：

```
0x000: 6300                  |    xorq %rax,%rax
0x002: 741600000000000000   |    jne  target       # Not taken
0x00b: 30f00100000000000000 |    irmovq $1, %rax   # Fall through
0x015: 00                    |    halt
0x016:                       | target:
0x016: ff                    |    .byte 0xFF        # Invalid instruction code
```

在这个程序中，流水线会预测选择分支，因此它会取出并以一个值为 0xFF 的字节作为指令(由汇编代码中.byte 伪指令产生的)。译码阶段会因此发现一个非法指令异常。稍后，流水线会发现不应该选择分支，因此根本就不应该取出位于地址 0x016 的指令。流水线控制逻辑会取消该指令，但是我们想要避免出现异常。

第三个细节问题的产生是因为流水线化的处理器会在不同的阶段更新系统状态的不同部分。有可能会出现这样的情况，一条指令导致了一个异常，它后面的指令在异常指令完成之前改变了部分状态。比如说，考虑下面的代码序列，其中假设不允许用户程序访问 64 位范围的高端地址：

```
1    irmovq $1,%rax
2    xorq %rsp,%rsp    # Set stack pointer to 0 and CC to 100
3    pushq %rax        # Attempt to write to 0xffffffffffffff8
4    addq  %rax,%rax   # (Should not be executed) Would set CC to 000
```

pushq 指令导致一个地址异常，因为减小栈指针会导致它绕回到 0xfffffffffffffff8。访存阶段中会发现这个异常。在同一周期中，addq 指令处于执行阶段，而它会将条件码设置成新的值。这就会违反异常指令之后的所有指令都不能影响系统状态的要求。

一般地，通过在流水线结构中加入异常处理逻辑，我们既能够从各个异常中做出正确的选择，也能够避免出现由于分支预测错误取出的指令造成的异常。这就是为什么我们会在每个流水线寄存器中包括一个状态码 stat(图 4-41 和图 4-52)。如果一条指令在其处理中于某个阶段产生了一个异常，这个状态字段就被设置成指示异常的种类。异常状态和该指令的其他信息一起沿着流水线传播，直到它到达写回阶段。在此，流水线控制逻辑发现出现了异常，并停止执行。

为了避免异常指令之后的指令更新任何程序员可见的状态，当处于访存或写回阶段中的指令导致异常时，流水线控制逻辑必须禁止更新条件码寄存器或是数据内存。在上面的示例程序中，控制逻辑会发现访存阶段中的 pushq 导致了异常，因此应该禁止 addq 指令更新条件码寄存器。