而其他一些代码则表示发生了某种类型的异常。代码 2，命名为 HLT，表示处理器执行了一条 halt 指令。代码 3，命名为 ADR，表示处理器试图从一个非法内存地址读或者向一个非法内存地址写，可能是当取指令的时候，也可能是当读或者写数据的时候。我们会限制最大的地址(确切的限定值因实现而异)，任何访问超出这个限定值的地址都会引发 ADR 异常。代码 4，命名为 INS，表示遇到了非法的指令代码。

值	名字	含义
1	AOK	正常操作
2	HLT	遇到器执行 halt 指令
3	ADR	遇到非法地址
4	INS	遇到非法指令

图 4-5　Y86-64 状态码。在我们的设计中，任何 AOK 以外的代码都会使处理器停止

对于 Y86-64，当遇到这些异常的时候，我们就简单地让处理器停止执行指令。在更完整的设计中，处理器通常会调用一个异常处理程序(exception handler)，这个过程被指定用来处理遇到的某种类型的异常。就像在第 8 章中讲述的，异常处理程序可以被配置成不同的结果，例如，中止程序或者调用一个用户自定义的信号处理程序(signal handler)。

4.1.5　Y86-64 程序

图 4-6 给出了下面这个 C 函数的 x86-64 和 Y86-64 汇编代码：

```
1   long sum(long *start, long count)
2   {
3       long sum = 0;
4       while (count) {
5           sum += *start;
6           start++;
7           count--;
8       }
9       return sum;
10  }
```

x86-64 code

```
   long sum(long *start, long count)
   start in %rdi, count in %rsi
1  sum:
2    movl    $0, %eax        sum = 0
3    jmp     .L2             Goto test
4  .L3:                    loop:
5    addq    (%rdi), %rax    Add *start to sum
6    addq    $8, %rdi        start++
7    subq    $1, %rsi        count--
8  .L2:                    test:
9    testq   %rsi, %rsi      Test sum
10   jne     .L3             If !=0, goto loop
11   rep; ret                Return
```

Y86-64 code

```
   long sum(long *start, long count)
   start in %rdi, count in %rsi
1  sum:
2    irmovq $8,%r8       Constant 8
3    irmovq $1,%r9       Constant 1
4    xorq %rax,%rax      sum = 0
5    andq %rsi,%rsi      Set CC
6    jmp     test        Goto test
7  loop:
8    mrmovq (%rdi),%r10  Get *start
9    addq %r10,%rax      Add to sum
10   addq %r8,%rdi       start++
11   subq %r9,%rsi       count--.  Set CC
12 test:
13   jne     loop        Stop when 0
14   ret                 Return
```

图 4-6　Y86-64 汇编程序与 x86-64 汇编程序比较。Sum 函数计算一个整数数组的和。Y86-64 代码与 x86-64 代码遵循了相同的通用模式