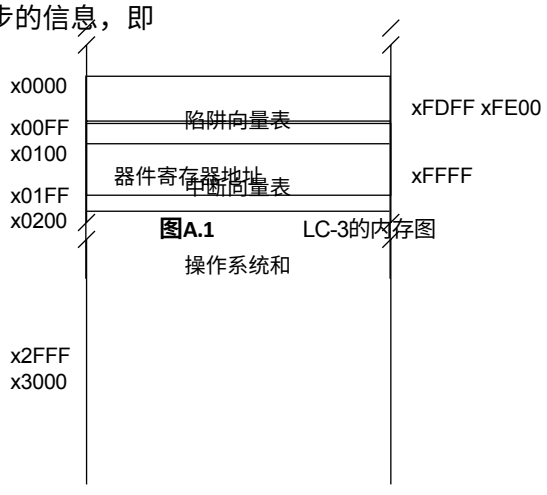
A.1 概述

LC-3的指令集结构（ISA）定义如下：

内存地址空间16位，对应于2个16，每个位置包含一个字（16位）。地址的编号从0（即x0000）到65,535（即xFFFF）。地址用于识别内存位置和内存映射的I/O设备寄存器。存储器的某些区域被保留用于特殊用途，如图A.1中所述。

位的编号 所有数量的位都有编号，从右到左，从0位开始。一个内存位置的内容的最左位是第15位。

指令指令的宽度为16位。位[15:12]指定操作码（要执行的操作），位[11:0]提供进一步的信息，即



图A.1 LC-3的内存图

主管堆栈

可用于用户程序

执行该指令所需的数据。每条LC-3指令的具体操作将在A.3节中描述。

非法操作码异常 比特[15:12]=1101没有被指定。如果一条指令在位[15:12]中包含1101，会发生非法操作码异常。A.4节解释了会发生什么。

程序计数器 一个16位的寄存器，包含下一条要处理的指令的地址。

通用寄存器 8个16位寄存器，编号从000到111。

条件代码 三个1位寄存器：N（负）、Z（零）和P（正）。加载指令（LD、LDI、LDR和LEA）和操作指令（ADD、AND和NOT）分别将一个结果加载到八个通用寄存器中的一个。条件代码被设置，基于该结果，作为一个16位2's补码的整数，是否为负数(N=1；Z，P=0），零（Z=1；N，P=0），或正（P=1；N，Z=0）。
所有其他LC-3指令都不改变条件代码。

内存映射的I/O 输入和输出由加载/存储（LDI/STI, LDR/STR）指令处理，使用内存地址来指定每个I/O设备寄存器。地址xFE00到xFFFF已经被分配来代表I/O设备的地址。见图A.1。另外，表A.3列出了迄今为止为LC-3确定的每个相关设备寄存器，以及它们在内存地址空间的相应分配地址。

中断处理 I/O设备具有中断处理器的能力。A.4节描述了这种机制。

优先级水平 LC-3支持八个优先级水平。优先级7（PL7）是最高的；PL0是最低的。当前执行的进程的优先级在位PSR[10:8]中指定。

处理器状态寄存器（PSR） 一个16位的寄存器，包含当前执行进程的状态信息。到目前为止，PSR的7位已经被定义。PSR[15]指定的特权模式是
正在执行的进程。PSR[10:8]指定当前执行的进程的优先级。PSR[2:0]包含条件代码。PSR[2]是N，PSR[1]是Z，而PSR[0]是P。

特权模式 LC-3规定了两个级别的权限，即监督者模式（特权）和用户模式（非特权）。中断服务程序在监督者模式下执行。特权模式是

由 PSR[15]指定的。PSR[15]=0 表示监督员模式；PSR[15]=1 表示用户模式。

特权模式异常 RTI指令在监管者模式下执行。如果处理器在用户模式下试图执行RTI指令，就会发生一个特权模式异常。A.4节解释了会发生什么。

监督员堆栈 监督员空间的一个内存区域，可通过监督员堆栈指针（SSP）访问。当PSR[15]=0时，堆栈指针（R6）就是SSP。

用户**堆栈 用户**空间中的一个内存区域，可通过用户堆栈指针（USP）访问。当PSR[15]=1时，堆栈指针（R6）就是USP。

A.2 记号

表A.1中的符号将有助于理解LC-3指令的描述（A.3节）。

A.3 指令集

LC-3支持一个丰富但精简的指令集。每条16位指令由一个操作码（bits[15:12]）和12个额外的位组成，用于指定执行该指令工作所需的其他信息。图A.2总结了LC-3中的15个不同的操作码和每条指令剩余位的规格。第16个4位操作码没有规定，而是保留给将来使用。在下面几页中，将对这些指令进行更详细的描述。对于每条指令，我们显示了汇编语言的表示，16位指令的格式，指令的操作，对其操作的英文描述，以及一个或多个指令的例子。在相关的地方，我们还提供了关于该指令的额外说明。

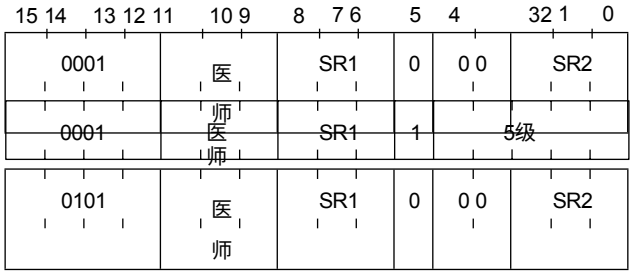
表A.1 记号性公约

符号	意义
xNumber	以十六进制表示的数字。#Number
以	十进制表示的数字。
A[I:r]	数据集A的左边位[I]和右边位[r]所划定的**字段**。例如，如果PC包含0011001100111111，那么PC[15:9]就是0011001。PC[2:2]是1。如果I和r是相同的位数，通常缩写为PC[2]。
基准R	基准寄存器；R0...R7之一，与六位偏移量一起使用，用于计算基准+偏移量地址。
DR	目标寄存器；R0...R7之一，指定指令的结果应该写到哪个寄存器。
imm5	A 5位即时值；当作为字面（即时）值使用时，指令的第[4:0]位。作为一个5位的2进制整数，它在使用前被符号扩展到16位。范围：-16..15.
标签	一种汇编语言结构，它以符号方式识别一个位置（即通过一个名称，而不是其16位地址）。
mem[地址]	表示给定地址处的内存内容。
offset6	一个6位值；指令的第[5:0]位；与Base+offset寻址模式一起使用。位[5:0]作为一个6位带符号的2's complement整数，符号扩展到16位，然后加到基数寄存器中，形成一个地址。范围：-32..31.
PC	程序计数器；16位寄存器，包含了下一条要取的指令的内存地址。例如，在执行地址A的指令时，PC包含地址A+1，表示下一条指令包含在A+1中。
PCoffset9	一个9位值；指令的第[8:0]位；与PC+offset寻址模式一起使用。位[8:0]作为一个9位有符号的2's complement整数，符号扩展到16位，然后加到增量的PC上，形成一个地址。范围为-256...255。
PCoffset11	一个11位的值；指令的第[10:0]位；与JSR操作码一起使用，计算子程序调用的目标地址。位[10:0]作为一个11位的2's complement整数，符号扩展到16位，然后加到增量的PC上，形成目标地址。范围为-1024...1023。
PSR	Processor Status Register；16位寄存器，包含正在运行的进程的状态信息。PSR[15] = 特权模式。PSR[2:0]包含条件代码。PSR[2]=N，PSR[1]=Z，PSR[0]=P。
setcc()	表示根据写入DR的结果值来设置条件代码N、Z和P。如果数值为负数，N=1，Z=0，P=0；如果数值为零，N=0，Z=1，P=0；如果数值为正数，N=0，Z=0，P=1。
SEXT(A)	A的最有意义的位被复制多少次，以使A扩展到16位。例如，如果A = 110000，那么SEXT(A) = 1111 1111 1111 0000。
SP	当前的堆栈指针。R6是当前的堆栈指针。有两个堆栈，每个特权模式都有一个。如果PSR[15]=0，SP为SSP；如果PSR[15]=1，SP为USP。
SR, SR1, SR2	S源寄存器；R0...R7中的一个，指定从哪个寄存器获得源操作数。
SSP	监督员堆栈指针。
trapvect8	一个8位的值；指令的第[7:0]位；与TRAP操作码一起使用，以确定一个陷阱服务

例程的起始地址。位[7:0]被当作无符号整数，并被零扩展到16位。这是包含相应服务例程的起始地址的内存位置的地址。范围为0...255。

USP	用户堆栈指针。
ZEXT(A)	将零扩展到A的最左位，将其扩展到16位。例如，如果A = 110000，那么ZEXT(A) = 0000 0000 0011 0000。

加+加



+和+

和+

BR

JMP

JSR

jsrr

ld+

ldi+

ldr+

lea+

not+

检索

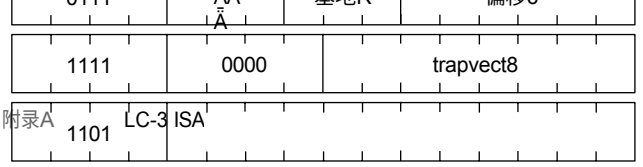
、测

试和

测试

陷阱

保留



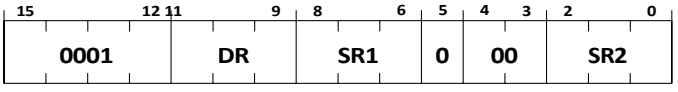
图A.2 整个LC-3指令集的**格式**。**注：**+表示修改条件代码的指令

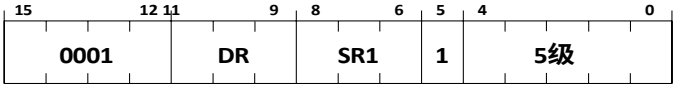
加法 加法

编译器格式

ADDDR , SR1, SR2
ADDDR , SR1,
imm5

编码





运作

```
如果(bit[5] == 0)
      dr = sr1 + sr2;
否则
      DR = SR1 + SEXT(imm5);
setcc();
```

描述

如果位[5]为0，第二源操作数从SR2获得。如果位[5]为1，第二源操作数通过将imm5字段的符号扩展到16位获得。在这两种情况下，第二源操作数与SR1的内容相加，其结果存储在DR中。根据结果是负数、零数还是正数，设置条件代码。

实例

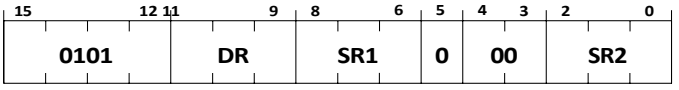
地址 R2, R3, R4 ; R2 ← R3 +
R4 添加 R2, R3, #7 ;
R2 ← R3 + 7

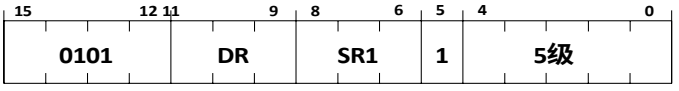
和 位智逻辑和

编译器格式

AND DR, SR1, SR2
AND DR, SR1, imm5

编码





运作

```
如果(bit[5] == 0)
      dr=sr1和sr2;
否则
      DR = SR1 AND SEXT(imm5);
setcc();
```

描述

如果位[5]为0，第二源操作数从SR2获得。如果位[5]为1，第二源操作数是通过将imm5字段的符号扩展到16位获得的。在任何一种情况下，第二源操作数和SR1的内容都要进行明智的位相加，并将结果存储在DR中。根据产生的二进制值（作为2的补数）是负数、零数还是正数，设置条件代码。

实例

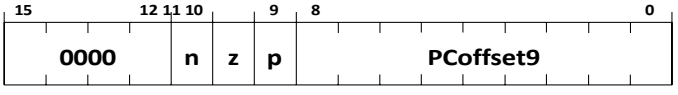
```
和      , r3, r4          ;r2← r3 和 r4 和
        , r3, #7          ;r2← r3 和 7
```

BR 条件性分支

编译器格式

ÄÄÄ	标签	BRzp	标签
BRz	标签	ÄÄÄ	标签
ÄÄÄ	标签	ÄÄÄ	标签
AAA	标签	BRnzp	标签

编码



运作

```
如果((n AND N) OR (z AND Z) OR (p AND P))
   PC = PC‡+ SEXT(PCoffset9);
```

描述

由位[11:9]的状态指定的条件代码被测试。如果位[11]被设置，N被测试；如果位[11]被清除，N不被测试。如果位[10]被设置，Z被测试，等等。如果任何一个被测试的条件代码被设置，程序就会分支到指定的位置，这个位置是将符号扩展的PCoffset9字段加到增量的PC上。

实例

```
BRzp  LOOP      ；如果最后一个结果是零或正数，则分支到LOOP
。BR† NEXT      ; 无条件分支到NEXT。
```

汇编语言操作码BR的解释与BRnzp相同；也就是说，总是分支到目标地址。

‡这是递增的PC。

JMP
RET

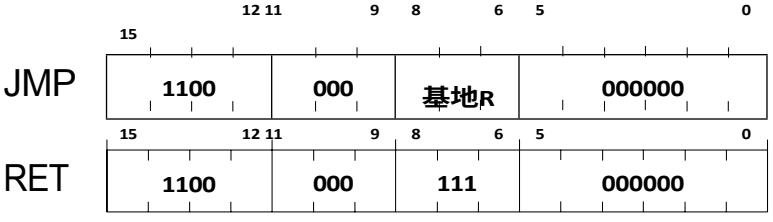
跳跃

从子程序返回

编译器格式

```
JMP    BaseR
RET
```

编码



运作

```
PC = BaseR;
```

描述

程序无条件地跳转到基数寄存器的内容所指定的位置。位[8:6]标识基数寄存器。

实例

```
JMP    r2        ; pc ← r2
ret              ; pc ← r7
```

注意事项

RET指令是JMP指令的一个特殊情况。PC被加载R7的内容，R7包含了返回子程序调用指令后的链接。

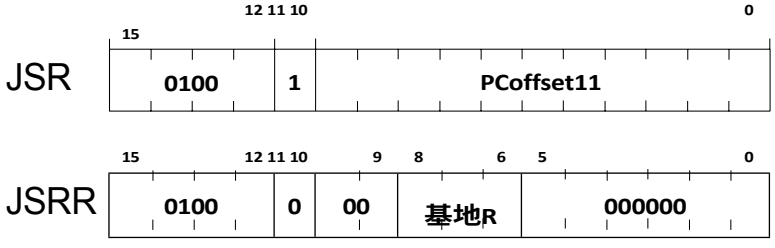
JSR
JSRR

跳转到子程序

编译器格式

JSR 标签
JSRR BaseR

编码



运作

```
R7 = PC;†
如果 (bit[11] == 0)
   PC = BaseR;
否则
   PC = PC† + SEXT(PCoffset11);
```

描述

首先，增量的PC被保存在R7中。这是与调用例程的联系。然后，PC被加载到子程序的第一条指令的地址，导致无条件跳转到该地址。子程序的地址从基础寄存器中获得（如果位[11]为0），或者通过符号扩展位[10:0]计算出地址，并将此值加入到增加的PC中（如果位[11]为1）。

实例

```
        JSRQUEUE ; 将JSR后面指令的地址放入R7；
                 ; 跳到QUEUE。
JSRR R3          ; 将JSRR后面的地址放入R7; 跳到
                 ； R3中包含的地址。
```

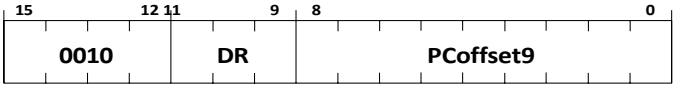
†这是增量的PC。

债务 负载

编译器格式

LDDR , LABEL

编码



运作

```
DR = mem[PC† + SEXT(PCoffset9)];
setcc();
```

描述

一个地址的计算方法是将位[8:0]的符号扩展到16位，并将这个值加到增量的PC上。这个地址的内存内容被加载到DR中。根据加载的值是负的、零的还是正的，条件代码被设置。

例子

LDR4 , VALUE ; R4 ← mem[VALUE]

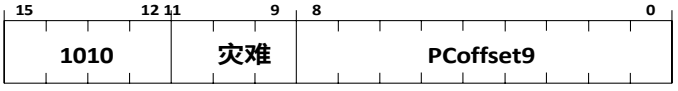
†这是增量的PC。

LDI 负载间接

编译器格式

LDI DR, LABEL

编码



运作

```
DR = mem[mem[PC† + SEXT(PCoffset9)]];
setcc();
```

描述

一个地址的计算方法是将位[8:0]的符号扩展到16位，并将这个值加到增量的PC上。在这个地址上存储的是要载入DR的数据的地址。条件代码被设置，基于加载的值是负的，零的，还是正的。

例子

LDIR4 , ONEMORE ; R4 ← mem[mem[ONEMORE]]

†这是增量的PC。

LDR 负载基数+偏移量

编译器格式

LDR DR, BaseR, offset6

编码

运作

```
DR = mem[BaseR + SEXT(offset6)];
setcc();
```

描述

地址的计算方法是将位[5:0]的符号扩展到16位，然后将这个值加到位[8:6]指定的寄存器的内容中。这个地址的内存内容被加载到DR中。根据加载的值是负的、零的还是正的，条件代码被设置。

例子

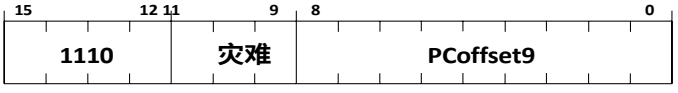
LDRR4 , R2, #-5 ; R4 ← mem[R2 - 5]

LEA 负载有效地址

编译器格式

铅笔 ，标签

编码



运作

```
DR = PC† + SEXT(PCoffset9); setcc()
;
```

描述

通过将位[8:0]的符号扩展到16位，并将此值加到增量的PC上，计算出一个地址。这个地址被加载到DR.‡的条件代码被设置，基于加载的值是负的、零的还是正的。

例子

LEAR4 , TARGET ; R4 ← TARGET的地址。

†这是增量的PC。

‡LEA指令不读取内存以获得载入DR的信息。地址本身被加载到DR中。

不是 比特-明智补码

编译器格式

いいいい いいいいいいいいいいいいいいいいいいいいいいいいいいいい

编码

15–12	11–9	8–6	5	4–0
1001 星	驱动器	卫	1	11111

运作

```
DR = NOT(SR);
setcc();
```

描述

SR内容的位数补码被存储在DR中。根据产生的二进制值（作为2的补码的整数）是负数、零数还是正数，设置条 件 代码。

例子

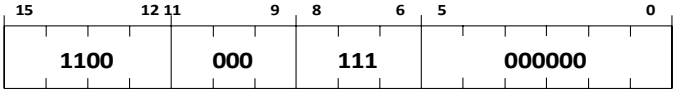
notr4 , r2 ; r4 ← not(r2)

RET† 从子程序返回

编译器格式

RET

编码



运作

```
PC=R7;
```

描述

PC被加载R7中的值。这将导致之前的JSR指令的返回。

例子

```
RET    ; PC ← R7
```

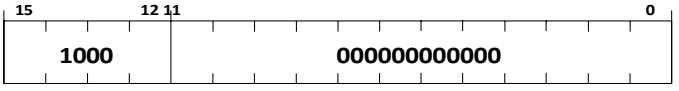
† RET指令是JMP指令的一个特定编码。参见JMP。

RTI 从中断中返回

编译器格式

RTI

编码



运作

如果(PSR[15] == 0)

PC = mem[R6]; R6是SSP R6 =

R6+1;

TEMP =

mem[R6];

R6=R6+1;

PSR = TEMP; 被中断进程的特权模式和条件代码被恢复。

否则

启动特权模式例外;

描述

如果处理器在监督者模式下运行，监督者堆栈的前两个元素被弹出并加载到PC，PSR中。如果处理器在用户模式下运行，就会发生特权模式违反的异常。

例子

RTI ; PC, PSR ← 从堆栈中弹出前两个值。

注意事项

在外部中断或内部异常时，启动序列首先将特权模式改为监督者模式（PSR[15] = 0）。然后，在用中断或异常服务例程的起始地址加载PC之前，被中断程序的PSR和PC被推到Supervisor堆栈。中断和异常服务例程以Supervisor的权限运行。服务例程的最后一条指令是RTI，它通过从Supervisor堆栈中弹出两个值来恢复PC和PSR，将控制权返回到被中断的

程序。在中断的情况下，PC被恢复到中断开始时即将处理的指令的地址。在发生异常的情况下，PC被恢复到引起异常的指令的地址或下一条指令的地址，这取决于引起异常的指令是否要被重新执行。在中断的情况下，PSR被恢复到中断发生时的值。在发生异常的情况下，PSR被恢复到异常发生时的值，或者恢复到一些修改后的值，这取决于异常的情况。参见A.4节。

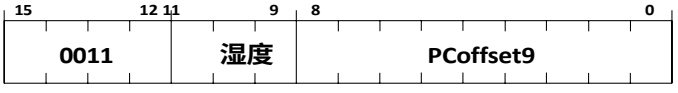
如果处理器在用户模式下运行，就会发生特权模式违反的异常。A.4节描述了在这种情况下会发生什么。

淘宝网 仓库

编译器格式

STSR , LABEL

编码



运作

```
mem[PC† + SEXT(PCoffset9)] = SR;
```

描述

SR指定的寄存器的内容被保存在内存位置，其地址是通过符号扩展位[8:0]到16位，并将此值添加到增加的PC上计算出来的。

例子

```
STR4, HERE    ; mem[HERE] ← R4
```

†这是增量的PC。

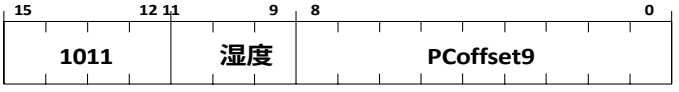
STI

仓库 间接

编译器格式

STI SR，标签

编码



运作

```
mem[mem[PC† + SEXT(PCoffset9)]] = SR;
```

描述

由SR指定的寄存器的内容被存储在内存位置，其地址是按以下方式获得的：位[8:0]被符号扩展为16位，并加到增加的PC上。在这个地址的内存中的内容就是SR中的数据被存储到的位置的地址。

例子

```
STIR4                    , NOT_HERE     ; mem[mem[NOT_HERE]]← R4
```

†这是增量的PC。

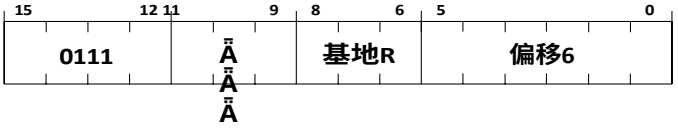
STR

存储基数+偏移量

编译器格式

STR SR, BaseR, offset6

编码



运作

```
mem[BaseR + SEXT(offset6)] = SR;
```

描述

SR指定的寄存器的内容被存储在内存位置，其地址是通过符号扩展位[5:0]到16位，并将此值加到位[8:6]指定的寄存器的内容中计算出来的。

例子

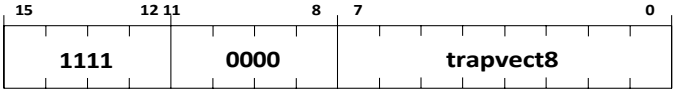
STRR4 , R2, #5 ; mem[R2 + 5] ← R4

TRAP 系统调用

编译器格式

TRAP trapvector8

编码



运作

```
R7 = PC;†
PC = mem[ZEXT(trapvect8)];
```

描述

首先，R7被加载了增量的PC。(这使得在服务程序执行完毕后，可以返回到原始程序中TRAP指令的物理位置）。然后，PC被装入由trapvector8指定的系统调用的起始地址。这个起始地址包含在内存位置中，这个位置的地址是通过将trapvector8扩展到16位而得到的。

例子

```
TRAP    x23    ; 指示操作系统执行IN系统调用。
               ;这个系统调用的起始地址包含在
               ； 内存位置x0023。
```

注意事项

内存位置x0000到x00FF，共256个，可用于包含系统调用的起始地址，由其相应的陷阱向量指定。这个内存区域被称为陷阱向量表。表A.2描述了对应于陷阱向量x20到x25的服务例程所执行的功能。

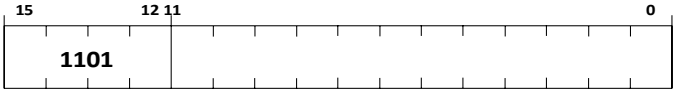
†这是增量的PC。

未使用的操作码

编译器格式

没有

编码



运作

发起一个非法操作码异常。

描述

如果遇到一个非法的操作码，就会发生一个非法操作码异常。

注意事项

操作码1101已经被保留给未来使用。它目前没有被定义。如果当前执行的指令的位[15:12]=1101，就会发生一个非法操作码异常。A.4节描述了发生的情况。

表A.2 陷阱服务程序

陷阱 矢量	汇编器名称	描述
x20	GETC	从键盘上读取一个字符。该字符不会被反馈到控制台。它的ASCII代码被复制到R0中。R0的高八位被清除。
x21	OUT	将R0[7:0]中的一个字符写到控制台显示器上。
x22	PUTS	将一串ASCII字符写入控制台显示。这些字符包含从R0中指定的地址开始，在连续的内存位置写入一个字符，每个内存位置一个字符。在一个内存位置上出现x0000时，写入就结束了。
x23	IN	在屏幕上打印一个提示，并从键盘上读取一个字符。该字符被应答到控制台监视器上，其ASCII码被复制到R0中。R0的高八位被清零。
x24	PUTSP	向控制台写入一串ASCII字符。这些字符包含在连续的内存位置，每个内存位置两个字符，从R0中指定的地址开始。存储器位置的位[7:0]中包含的ASCII码首先被写到控制台。然后将该内存位置的位[15:8]中包含的ASCII码写到控制台。(一个由奇数个字符组成的字符串将在包含最后一个字符的内存位置的位[15:8]中出现x00。）当内存位置上出现x0000时，写入就结束了。
x25	HALT	停止执行并在控制台打印一条信息。

表A.3 器件寄存器的分配

地址I/O寄存器		名称I/O寄存器功能
xFE00	键盘状态寄存器	也被称为KBSR。准备就绪位（位[15]）指示是否xFE02键盘数据寄存器也被称为KBDR。第[7:0]位包含最后的键盘上输入的字符。
xFE04	显示状态寄存器	也被称为DSR。准备就绪位（位[15]）指示是否显示设备已经准备好接收另一个字符打印在屏幕上。
xFE06	显示数据寄存器	也被称为DDR。在低字节中写入一个字符
xFFFE	Machine control register	也叫MCR，该寄存器的值将显示在屏幕上。位[15]是时钟使能位。当清零时，指令处理停止。

A.4 中断和异常处理

正在运行的程序的外部事件可以中断处理器。外部事件的一个常见例子是中断驱动的I/O。还有一种情况是，处理器可以被程序运行时发生的、

由程序本身引起的特殊事件打断。这种 "内部 "事件的一个例子是正在运行的计算机程序中出现了一个未使用的操作码。

与每个可以中断处理器的事件相关的是一个8位的向量，它提供了一个进入256条*中断向量表的*入口。中断向量表的起始地址是x0100。就是说，中断向量表

占据内存位置x0100至x01FF。中断向量表中的每个条目都包含处理相应事件需求的服务例程的起始地址。这些服务例程在监督员模式下执行。

这些条目的一半（128），位置x0100到x017F，提供了为运行中的程序本身引起的事件提供服务的例程的起始地址。这些例程被称为*异常服务例程*，因为它们处理异常事件，即阻止程序正常执行的事件。另一半条目，位置x0180到x01FF，提供了为正在运行的程序的外部事件提供服务的例程的起始地址，例如来自I/O设备的请求。这些程序被称为*中断服务程序*。

A.4.1 中断

目前，LC-3计算机系统只提供一个可以中断处理器的I/O设备。这个设备就是键盘。它的中断优先级为PL4，提供中断向量x80。

如果一个I/O设备想要得到服务，如果它的中断使能（IE）位被设置，并且其请求的优先级高于正在运行的程序的优先级，那么它就可以中断处理器。

假设一个程序正在以低于4的优先级运行，有人在键盘上敲了一个键。如果KBSR的IE位为1，当前执行的程序在当前指令周期结束时被中断。中断服务程序的**启动方式**如下：

1. 处理器将特权模式设置为监督者模式（PSR[15]=0）。
2. 处理器将优先级设置为PL4，即优先级的中断设备（PSR[10:8]=100）。
3. 如果R6还没有包含SSP的话，它就会被加载到Supervisor Stack Pointer（SSP）。
4. 被中断的进程的PSR和PC被推到监督者堆栈上。
5. 键盘提供它的8位中断向量，在这种情况下是x80。
6. 处理器将该向量扩展到x0180，即中断向量表中相应的16位地址。
7. PC被加载内存位置x0180的内容，这是键盘中断服务例程中第一条指令的地址。

然后，处理器开始执行中断服务程序。

中断服务例程中执行的最后一条指令是RTI。Supervisor堆栈的前两个元素被弹出并加载到PC和PSR寄存器中。R6被加载适当的堆栈指针，这取决于PSR[15]的新值。然后继续处理被中断的程序。

A.4.2 例外情况

此时，LC-3 ISA规定了两种异常条件：特权模式违反和非法操作码。特权模式违反发生在处理器

在用户模式下运行时遇到RTI指令。如果处理器在正在处理的指令中遇到未使用的操作码（位[15:12] = 1101），就会发生非法操作码异常。

异常一旦被检测到就会立即被处理。它们的*启动*非常像中断的启动，也就是说：

1. 处理器将特权模式设置为监督者模式（PSR[15]=0）。
2. 如果R6还没有包含SSP的话，它就会被加载到Supervisor Stack Pointer（SSP）。
3. 被中断的进程的PSR和PC被推到监督者堆栈上。
4. 异常提供它的8位向量。在特权模式异常的情况下，这个向量是x00。在非法操作码的情况下，这个向量是x01。
5. 处理器将该向量扩展到x0100或x0101，即中断向量表中相应的16位地址。
6. PC被加载内存位置x0100或x0101的内容，即相应的异常服务例程中第一条指令的地址。

然后，处理器开始执行异常服务程序。

异常服务程序的细节取决于异常和操作系统希望处理该异常的方式。

在许多情况下，异常服务例程可以纠正由异常事件引起的任何问题，然后继续处理原始程序。在这些情况下，异常服务例程的最后一条指令是RTI，它从监督器堆栈中弹出前两个元素，并将其加载到PC和PSR寄存器中。然后程序恢复执行，问题得到纠正。

在某些情况下，异常事件的原因是如此灾难性的，以至于异常服务例程将程序从进一步处理中移除。

处理中断和处理异常的另一个区别是在执行服务例程时处理器的优先级别。在处理异常的情况下，我们通常不会在服务异常时改变优先级别。一个程序的优先级是指它需要被执行的紧迫性。在LC-3 ISA规定的两种异常的情况下，程序的紧迫性不会因为发生特权模式违反或程序中存在非法操作码而改变。