汇编语言

标号(label) 是指向内存单元的一个符号名，它可以在程序中直接引用。LC-3汇编语言中，一个标号可以包含1~20个字符(如大写或小写的字母或数字)，但首字符必须是字母。NOW、Under21、R2D2、C3PO等都是合法的标号例子。

.STRINGZ告诉汇编器**连续占用并初始化n+1个内存单元**，其参数(或操作数)是双引号括起来的n个字符。n+1个内存单元的前n个字的内容分别是**字符串对应字符的ASCII码的零扩展(zero-extend)值**，内存的最后一个字则被初始化为0。最后的这个字符x0000通常为ASCII码字符串的处理提供了哨兵机制。

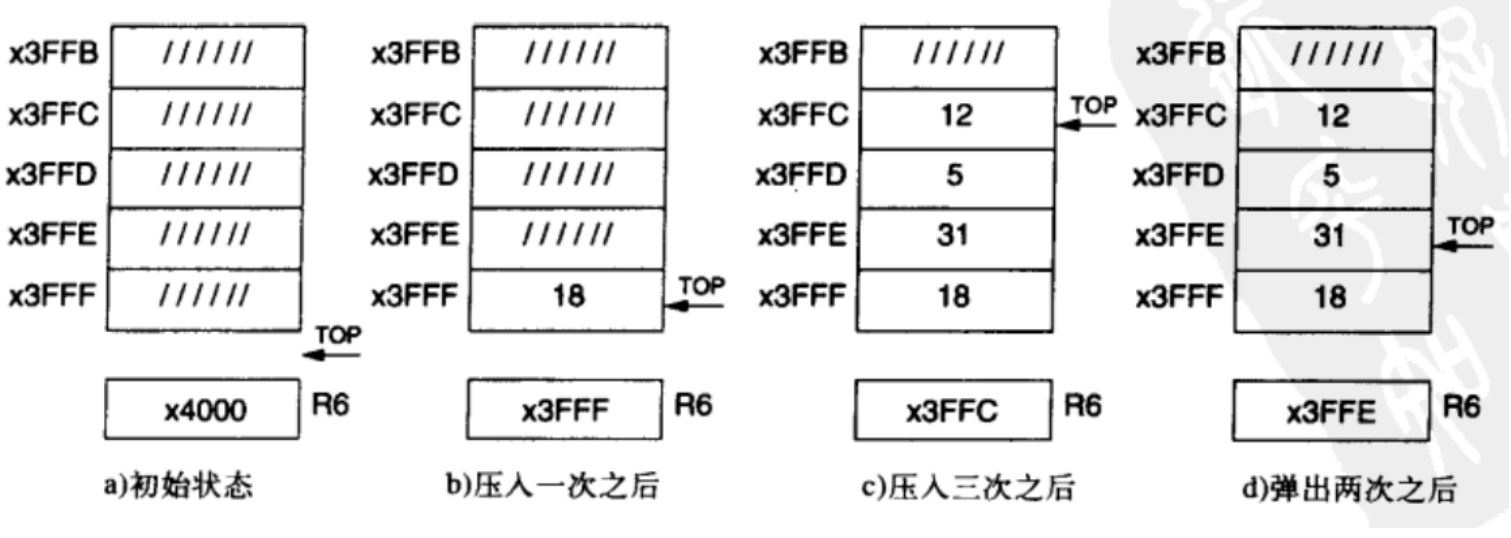
.END告诉汇编器"程序结束了"。出现在.END之后的任何字符都将被汇编器丢弃。注意.END并不会停止程序的执行。事实上，最后的程序中不会出现任何.END。它仅仅是一个分隔符(表示源程序结束了)。

子程序

调用机制的过程是，首先计算子程序的开始地址，并将其装入PC；然后，保存调用返回地址(下一条指令地址)。在返回机制中，将把返回地址再装入PC。

栈

在各种计算机系统中，最常见的栈实现方式如图所示，它是由一段连续内存空间和一个寄存器(栈指针)组成。所谓“ 栈指针”(stack pointer)，是一个寄存器，它的内容是一个地址值，始终指向栈的顶部(即最近被压入的元素)。每个被压入栈中的元素，在内存空间中都占据着一个独立的位置。但是，在入栈和出栈操作时，栈中的其他数据不需要再被移动。



上下溢出

常用的做法是，子程序将执行成功或失败的信息记录在某个寄存器中。比如在POP程序的流程中，将成功或失败的信息记录在R5中。那么，POP程序返回后，调用程序通过R5就可以获知IPOP执行是否成功。

> 子程序中的语句也会影响条件码

为避免R5在PUSH程序执行之前的内容丢失，调用程序要在JSR指令执行之前，保存R5的内容。

递归

每次JSR的时候要注意R7的状态

A new solution: using stack

```
FACT        ADD R6,R6,#-1
            STR R1,R6,#0    ; Push Caller's R1 on the stack, so we can use R1.

            ADD R1,R0,#-1               ; If n=1, we are done since 1! = 1
            BRz NO_RECURSE

            ADD R6,R6,#-1
            STR R7,R6,#0    ; Push return linkage onto stack
            ADD R6,R6,#-1
            STR R0,R6,#0    ; Push n on the stack

            ADD R0,R0,#-1               ; Form n-1, argument of JSR
B           JSR FACT
            LDR R1,R6,#0    ; Pop n from the stack
            ADD R6,R6,#1
            MUL R0,R0,R1    ; form n*(n-1)!

            LDR R7,R6,#0    ; Pop return linkage into R7
            ADD R6,R6,#1
NO_RECURSE  LDR R1,R6,#0    ; Pop caller's R1 back into R1
            ADD R6,R6,#1
            RET
```

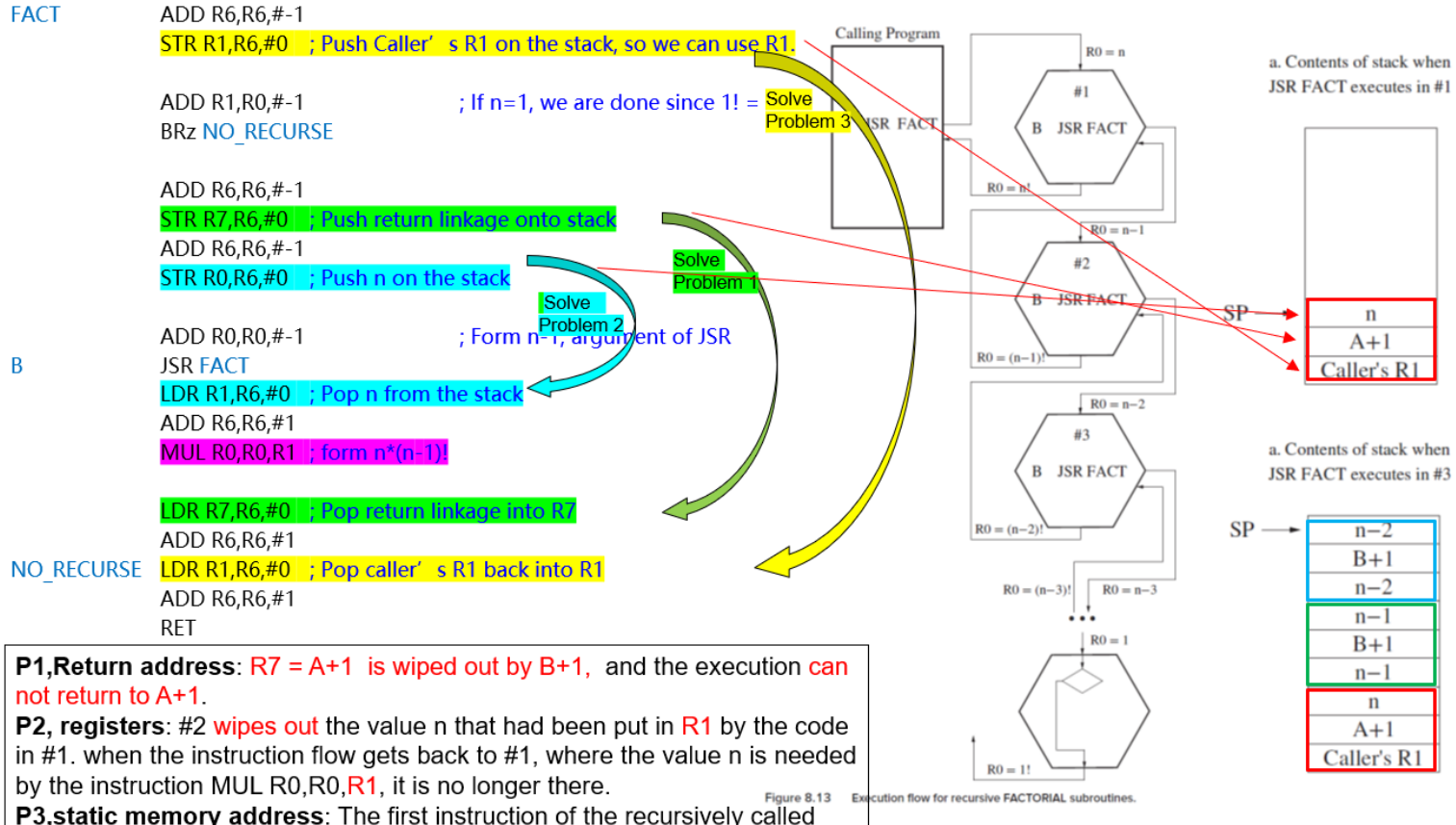


Figure 8.13 Execution flow for recursive FACTORIAL subroutines.

P1,Return address: R7 = A+1 is wiped out by B+1, and the execution can not return to A+1.
P2, registers: #2 wipes out the value n that had been put in R1 by the code in #1. when the instruction flow gets back to #1, where the value n is needed by the instruction MUL R0,R0,R1, it is no longer there.
P3,static memory address: The first instruction of the recursively called subroutine FACT (#2) will save that value to Save1, wiping out the value that the main program (#1) had stored in R1 when it called FACT.

队列

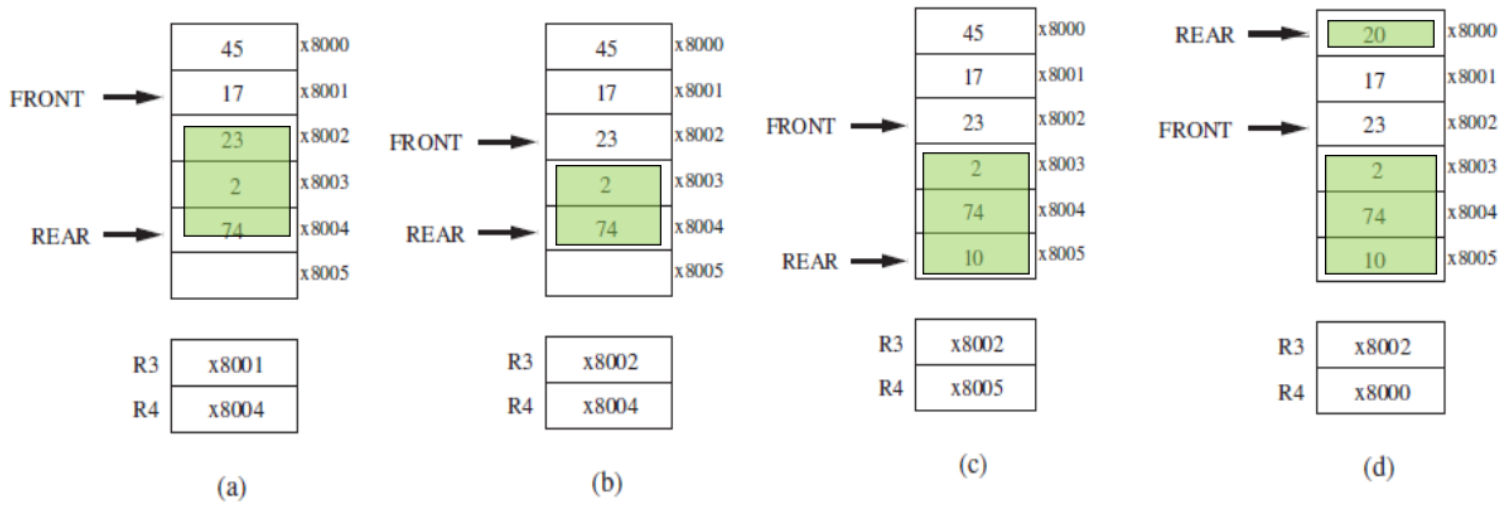


Figure 8.25 A queue allocated to memory locations x8000 to x8005.

The queue are allowed to store only n-1 (why?) elements for a queue with n locations.

因为需要区分队列空和队列满的状态，当头指针和尾指针相等时队列为空，当尾指针在头指针前一个位置时队列为满。尾指针所指向的位置必须空着，所以能用的元素个数为N-1。

元素从队列的前端移除或插入到队列的尾部，当其中一个指针到达最后一个元素时绕圈，并返回一个成功(R5 = 0)或失败(R5 = 1)的报告，这取决于访问是成功还是由于下溢或溢出条件导致访问失败。

我们为队列分配了位置x8000到x8005，x8000是FIRST位置，x8005是LAST位置。要插入，我们首先必须确保队列没有满。为此，我们增加REAR指针(R4)，然后测试REAR=FRONT。如果REAR指针最初是x8005，则通过将其设置为x8000来增加REAR；也就是说，我们需要绕过去。如果队列已满，我们需要设置后回到原来的值，并返回，报告失败(R5 = 1)。

如果队列不完整……

如果REAR=FRONT，则队列为空，因此返回，报告失败。如果REAR与FRONT不同，则队列不是空的，因此我们可以删除FRONT元素。为此，我们首先测试看看FRONT是否=x8005。如果是，则设置FRONT=x8000。如果不是，则增加FRONT。在这两种情况下，我们然后将该内存位置的值加载到R0中，并返回，报告成功。

TRAP

通常，我们说硬件寄存器是“有特权的”，是指它们只能被那些有合适权限的程序访问。

一种比较简单和安全的解决方法，是借助于TRAP指令和操作系统。所谓“操作系统”， 就是拥有特权权限的程序。

用户程序在x4000地址处，将要执行I/O任务。它请求操作系统以该用户程序的身份完成这个任务。操作系统接过控制权，分析并处理TRAP指令传递的服务要求，然后将控制权交还给x4001地址的指令。我们称这种用户程序的请求为“服务调用”(service call) 或系统调用”(system call)。

TRAP机制

(1) 服务程序(service routine) 集合：由操作系统提供，但以用户身份执行。这些服务程序是操作系统的组成部分，起始于各自固定的内存地址。LC-3最多可以支持256个服务程序。

(2) 起始地址表：包含256个服务程序的起始地址。该表位于内存地址x0000~x00FF。LC-3陷入矢量表给出了各矢量(服务程序)的起始地址。例如，地址x0021的内容是字符输出服务程序的起始地址(x0430)， 地址x0023的内容是键盘输入服务程序的起始地址(x04A0)， 地址x0025的内容是机器挂起服务程序的起始地址(xFD70)。

在执行服务程序之前，TRAP指令需要先完成两件事：

- 根据陷入矢量表项的内容，将PC值修改为对应于服务程序的起始地址。
- 提供一种机制，返回到调用TRAP指令的程序。我们称该“返回”机制为“链接”(linkage)。

“TRAP指令”由两部分组成：操作码1111和陷入矢量编号(bit[7:0])。 **位[11:8]必须为0。**

TRAP指令在执行时，要完成4件任务：

（1）将8-bit的陷入矢量零扩展（zero-extend）为16-bit地址，并装入MAR。例如对于陷入矢量x23来说，扩展后的地址就是x0023，它代表某个陷入矢量表项的地址。

（2）陷入矢量表位于内存x0000~x00FF。随后，表项x0023的内容被读入MDR，即x04A0（如图9-2所示）。

（3）将当前PC值存入寄存器R7，以实现返回用户程序的链接机制。

（4）将MDR的内容装入PC。至此，完成TRAP指令。

由于PC的内容目前为x04A0，因而程序将从地址x04A0开始，继续执行。

x04A0指向的是操作系统“键盘读入”服务程序的起始。我们说，该陷入矢量“指向”该TRAP程序的起始。所以，TRAP x23指令的作用是激活操作系统的“键盘读入”服务程序。

服务程序最后JMP R7即可返回原来的PC。

在trap服务程序中，JMP R7指令非常有用(返回用户程序)。为此，在LC-3汇编语言中，专为这个操作定义了一一个专用指令字RET。

中断

过RUN锁存门和晶振输出的“与” (AND)逻辑，可以控制计算机的运行。如果RUN锁存的位为0，则“与” 门输出为0，即传输给整个计算机系统的时钟停止。

在早期的指令集体系结构中，都有一个HALT指令(清零RUN的内容)，可用来停止时钟。但由于该指令的使用频率很低，为它分配一个操作符有些浪费(占用一个指令编码空间)。所以，在现代计算机中，改用TRAP指令方式来清除RUN门。在LC-3中，RUN的内容对应机器控制寄存器(映射内存地址xFFFE)的第15位。（例子程序在英文书P335页）

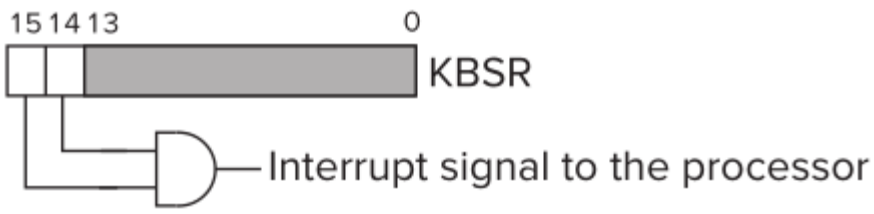
一个寄存器的原内容，如果在修改为其他值之后还要被使用，则在修改操作之前，必须要将原值保存，修改之后再将其恢复。**保存的方法是，将寄存器内容存入内存中的某个位置。**恢复时，将该值重新装入寄存器即可。

有关保存/恢复问题，既可以由调用程序在TRAP之前负责，也可以由被调用者(TRAP执行之后)来负责。我们称由调用程序负责该问题的方式为“调用者保存”(caller-save) 方式，而称由被调用者负责的方式为“被调用者保存”(allee-save)。选择最合适方式的原则：对于特定的寄存器，谁知道该寄存器的内容会被修改，则由谁来负责保存。

I/O设备要真正中断正在运行的程序，必须具备以下几个条件：1、I/O设备必须需要服务。2、设备必须有请求服务的权利。3、设备请求必须比处理器当前正在做的事情更紧急。如果这三个条件都满足，处理器就停止执行正在运行的程序，并处理中断。

如果I/O设备是键盘，那么如果有人输入了字符，它就需要服务。（当设置了相应的就绪位时，I/O设备需要服务。）

中断启用位，它可以由处理器(通常是由操作系统)设置或清除，这取决于处理器是否想要给I/O设备请求服务的权利。在大多数I/O设备中，这个中断启用(IE)位是设备状态寄存器的一部分。在图9.18的KBSR和DSR中，IE位为[14]位。I/O设备的中断请求信号是IE位和就绪位的逻辑AND。



中断可以在任何时候发生。它们与控制计算机的同步有限状态机是异步的。例如，中断信号可能发生在指令周期的FETCH OPERAND阶段。

> 中断相关的过程见中文书P140.

9.5节（英文书P351）的翻译：

回想一下我们对轮询的讨论：我们在相关的状态寄存器中持续测试就绪位，如果没有设置就绪位，我们就返回到再次测试就绪位。例如，假设我们正在向监视器写入一个字符串，并且我们正在使用轮询来确定监视器何时成功地写入了当前字符，以便我们可以分派下一个字符。我们认为三个指令序列LDI(加载DSR的就绪位)、BRzp(测试它并在设备就绪时失效)和STI(在DDR中存储下一个字符)作为一个原子单元是理所当然的。但是如果我们同时启用中断会怎么样呢?也就是说,如果一个中断发生在LDI，BRzp, STI序列(之前说,STI指令),它可以很容易的LDI指令表示DDR准备,BRzp指令没有分公司,但是当中断服务程序完成所以STI DDR可以写,DDR可能不再准备好了。计算机将执行STI，但写操作不会发生。一个简单但有些做作的例子:-)可以说明这个问题。假设执行了10次“for”循环，每次循环体都向监视器打印一个特定的字符。轮询用于在将下一个字符写入DDR之前确定监视器已经准备好。由于循环体执行了10次，这将导致字符在监视器上被打印10次。假设您还启用了键盘中断，并且键盘服务例程回显输入的字符。假设循环体执行如下:LDI加载就绪位，BRzp进入，因为监视器就绪，STI将字符存储在DDR中。在这个序列的中间，在STI执行之前，有人键入一个键。发生键盘中断，输入的字符被回送，即，写入DDR，然后键盘中断服务程序完成。然后，被中断的循环体接管并“知道”监视器已经就绪，因此它执行STI. ...除了监视器还没有准备好，因为它还没有完成键盘服务程序的编写！循环体的STI写入，但由于DDR还没有准备好，所以不会发生写入。最终的结果是：只写九个字，而不是十个。如果所写的字符串是用代码写的，而丢失的写会阻止代码被破译，那么这个问题就会变得更严重。处理此问题的一种简单方法是在进行轮询时禁用所有中断。但考虑一下后果。假设轮询需要很长时间。如果我们在轮询时禁用中断，中断将被禁用很长一段时间，这在一个关注高优先级中断发生和中断获得服务之间的时间间隔的环境中是不可接受的。

图9.22显示了一个更好的解决方案。我们想要使其不可中断的序列显示在0F到11行。我们首先在第09行加载R1，PSR在第0A行禁用中断，然后在R2中加载R1。PSR[14]是与此程序相关的所有中断的中断启用位。注意，PSR是映射到xFFFC的内存。我们通过将R1存储在PSR中(第0D行)来启用中断，然后立即通过将R2存储在PSR中(第0E行)来禁用中断。在中断被禁用的情况下，如果状态寄存器表明设备已经就绪，我们将执行三个指令序列LDI、BRzp和LDI(第0F、10和11行)。如果设备没有准备好，BRzp(第10行)将计算机带回第0D行，在那里再次启用中断。

通过这种方式，中断被一再禁用，但每次只足够执行三个指令序列LDI、BRzp、STI(在第0F行，10,0d行中)，然后再次启用中断。结果是：中断必须等待三个指令序列LDI、BRzp、STI执行，而不是等待整个轮询过程完成。

The Processor Status Register 处理器状态寄存器

包含分配给该程序的特权和优先级。Bit[15]表示特权，其中PSR[15]=0表示管理员特权，PSR[15]=1表示非特权。比特[10:8]指定程序的优先级级别(PL)。最高优先级为7（PL7），最低优先级为PL0。PSR还包含条件代码的当前值。

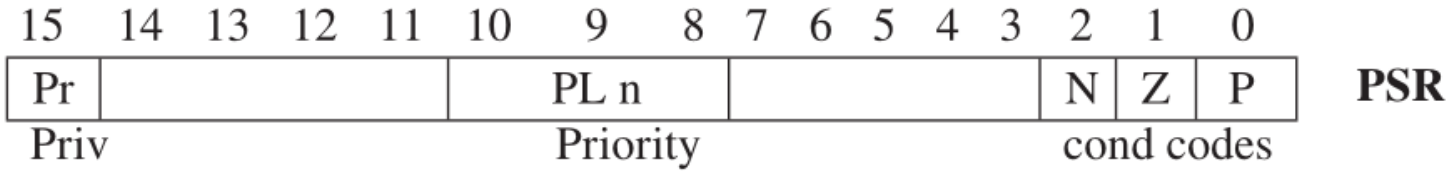


Figure 9.1 Processor status register (PSR).

位置x0000到x2FFF是特权内存位置。它们包含操作系统的各种数据结构和代码。他们需要管理员的权限才能进入。它们被称为系统空间。

位置x3000到xFDFF是非特权内存位置。访问这些内存位置不需要管理器特权。所有的用户程序和数据都使用这个内存区域。该区域通常称为用户空间。

地址xFE00到xFFFF根本不对应内存位置。也就是说，内存位置的最后一个地址是xFDFF。地址xFE00到xFFFF用于标识参与输入和输出函数的寄存器和一些与处理器相关的特殊寄存器。

例如，给PSR分配地址xFFFC，给处理器的主控制寄存器(MCR)分配地址xFFFE。从xFE00到xFFFF的地址集合通常被称为I/O页，因为大多数地址用于标识参与输入或输出功能的寄存器。访问这些寄存器需要管理员权限。

内存区域（英文书P316）

系统空间中的管理器堆栈和用户空间中的用户堆栈。每个堆栈都有一个堆栈指针，管理器堆栈指针(SSP)和用户堆栈指针(USP)，用来指示堆栈的顶部。由于一个程序在任何时候只能在管理器模式或用户模式下执行，因此在任何时候只有两个堆栈中的一个是活动的。**寄存器6通常用作活动堆栈的堆栈指针**(SP)。两个寄存器，Saved SSP和Saved USP，用于保存不使用的SP。当权限发生变化时，例如从Supervisor模式切换到User模式，SP会保存在Saved SSP中，SP会从Saved USP中加载。

输入输出

内存映射

大多数计算机设计者不喜欢指定一套额外的指令来处理输入和输出。它们使用相同的数据移动指令，用于在内存和通用寄存器之间加载和存储数据。(如LD，ST)

由于程序员使用与内存相同的数据移动指令，每个输入设备寄存器和每个输出设备寄存器都必须惟一地标识，就像唯一地标识内存位置一样。因此，每个设备寄存器都从ISA的内存地址空间分配一个地址。也就是说，I/O设备寄存器映射到一组分配给I/O设备寄存器而不是分配给内存位置的地址。因此得名内存映射I/O。

原来的PDP-11 ISA有一个16位的地址空间。所有比特[15:13]= 111被分配到I/O设备寄存器的地址。也就是说，在2^{16}个地址中，只有57344个地址对应于内存位置。其余的2^{13}个是内存映射的I/O地址。

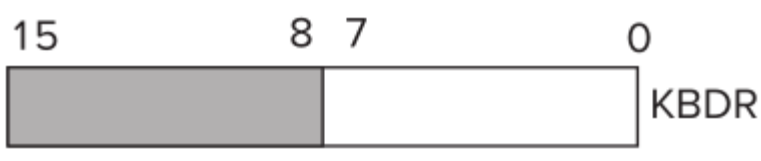
LC-3使用内存映射I/O。x0000到xFDFF地址指的是实际的内存位置。**地址xFE00到xFFFF预留给输入/输出设备寄存器**。表A.3列出了到目前为止已经分配的LC-3设备寄存器的内存映射地址。

同步/异步

处理器和I/O之间的大多数交互是异步的。在异步世界中控制处理需要一些协议或握手机制。我们的键盘和显示器也是如此。**在键盘的情况下，我们将需要一个位状态寄存器，称为标记，以指示某人是否输入了字符**。在监视器的情况下，我们将需要一个位状态寄存器来指示最近发送到监视器的字符是否已经被显示，因此可以给监视器另一个要显示的字符。

键盘输入/回显

为了处理来自键盘的字符输入，需要两个东西：储存要输入字符的数据寄存器 和 让处理器知道输入已经发生的同步机制。同步机制包含在与键盘相关联的状态寄存器中。这两个寄存器称为键盘数据寄存器(KBDR)和键盘状态寄存器(KBSR)。它们从内存地址空间中被分配地址。如表A.3所示，将地址xFE02分配给KBDR；地址xFE00被分配给KBSR。



KBSR[15]控制慢键盘和快处理器的同步。当敲击键盘上的一个键时，该键的ASCII码被载入KBDR[7:0]中，与键盘相关的电子电路自动将KBSR[15]设置为1。当LC-3读取KBDR时，与键盘相关的电子电路自动清除KBSR[15]，允许敲击另一个键。如果KBSR[15] = 1，则最后一个击键对应的ASCII码还没有被读取，因此键盘被禁用。

实现内存映射输入所需的额外数据路径（英文书P321）。

在内存映射输入的情况下，MAR不是用内存位置的地址加载，而是用设备寄存器的地址加载。地址控制逻辑选择相应的设备寄存器来为MDR提供输入，而不是使用地址控制逻辑来读取内存。

输出的工作方式与输入非常相似，DDR和DSR分别取代了KBDR和KBSR的角色。DDR代表显示数据寄存器，它驱动监视器显示。DSR代表显示状态寄存器。在LC-3中，DDR被分配的地址是xFE06。DSR被分配地址为xFE04。

这一部分的示例LC3程序在英文书P323附近。

```
01     START   LDI     R1, KBSR    ; Test for character input
02             BRzp    START
03             LDI     R0, KBDR
04     ECHO    LDI     R1, DSR     ; Test output register ready
05             BRzp    ECHO
06             STI     R0, DDR
07             BRnzp   NEXT_TASK
08     KBSR    .FILL   xFE00       ; Address of KBSR
09     KBDR    .FILL   xFE02       ; Address of KBDR
0A     DSR     .FILL   xFE04       ; Address of DSR
0B     DDR     .FILL   xFE06       ; Address of DDR
```

思考

假设我们被要求编写一个程序，它接受在键盘上输入的100个字符的序列，并处理这100个字符中包含的信息。假设字符以每分钟80个字的速度输入，相当于每0.125秒输入一个字符。假设处理100个字符的序列需要12.49999秒，我们的程序将对1000个连续序列执行此处理。我们的项目完成这项任务需要多长时间?

我们可以通过轮询获得每个字符输入。如果我们这样做，我们将浪费大量时间等待“下一个”字符被输入。获得一个100个字符的序列需要12.5秒。

如果我们使用中断驱动的I/O，处理器在等待输入字符时不会浪费任何时间重新执行LDI和BR指令。相反，处理器可以忙于处理之前输入的100个字符序列，除了被I/O设备打断读取下一个输入字符的那一小段时间。假设读取下一个输入的字符需要执行10条指令程序，每条指令平均执行时间为0.00000001秒。这意味着对于输入的每个字符，0.0000001秒，或者对于整个100个字符的序列，0.00001秒。也就是说，使用中断驱动的I/O，因为处理器只在实际读取字符时才需要，所以每100个字符序列所需的时间是0.00001秒，而不是12.50000秒。剩下的12.50000秒中的12.49999秒，处理器可以做有用的工作。

TRAP服务程序执行细节

陷入矢量	汇编器名	描　述
x20	GETC	从键盘读入一个字符。但该字符并不在屏幕上回显(echo)。该字符的ASCII码值被拷贝入R0（R0的高8位被清零）
x21	OUT	将R0[7:0]的字符输出在屏幕上显示
x22	PUTS	向屏幕写一个字符串。所有字符在内存中的存放是连续的，且每个内存单元一个字符。起始地址由R0指定，结束判断由当前内存单元是否为x0000确定
x23	IN	先打印提示（prompt）在屏幕上，然后等待键盘输入一个字符。读入字符回显在屏幕上，读入的ASCII码装入R0，R0高8位清零
x24	PUTSP	向屏幕写一个字符串。所有字符在内存中的存放是连续的，但每个内存单元存放两个字符。输出显示时，先将bit[7:0]输出，然后再将bit[15:8]输出（如果一个字符串的字符个数是奇数个，则最后一个内存单元的内容bit[15:8]为x00）。起始地址由R0指定，结束判断由当前内存单元是否为x0000确定
x25	HALT	停止执行，并在屏幕上输出信息