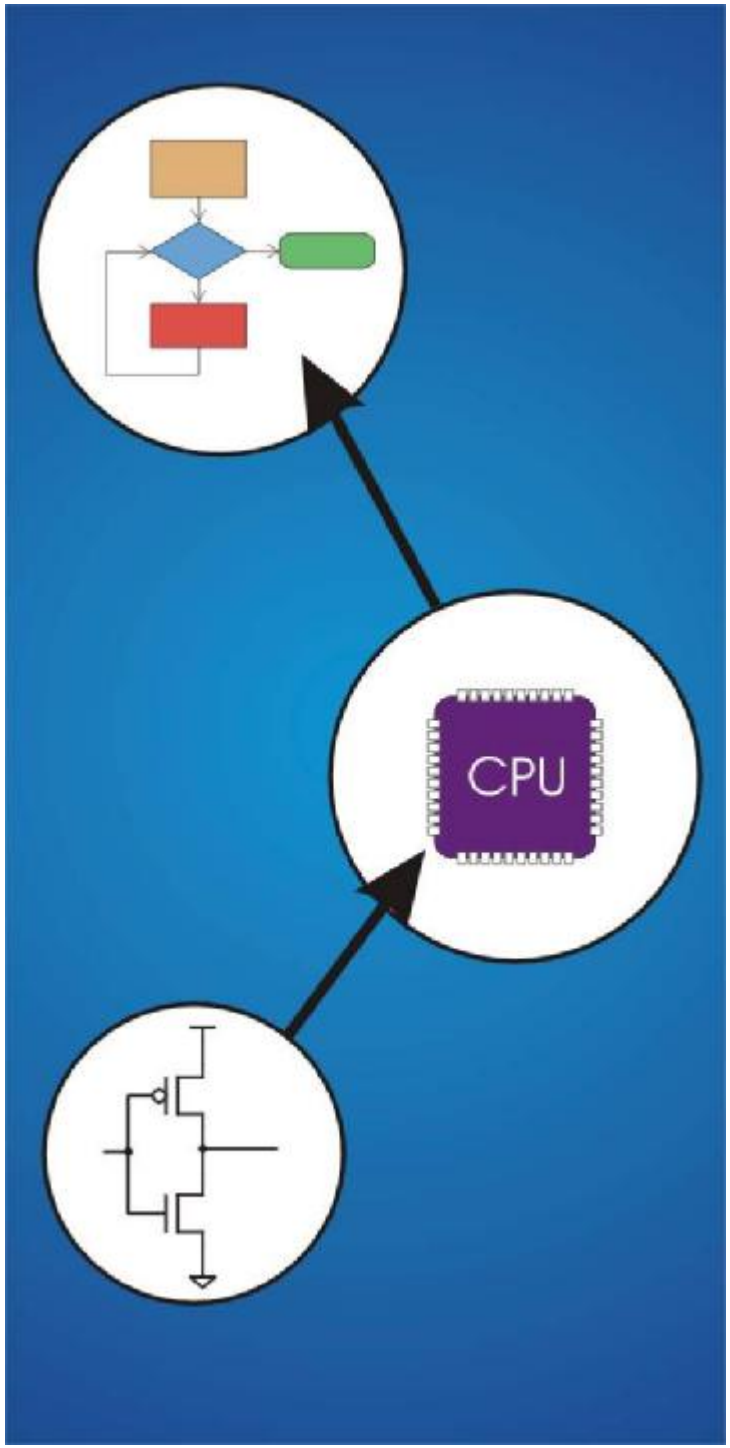


Chapter 9
TRAP 程序和子程序

系统调用

系统调用于实现特定操作，这些操作需要程序员具备某些不熟悉专业领域的知识，或者为了安全性而需要保护的操作。

- 专业性：比如对外部设备操作时需要了解外部设备的工作原理，内部的I/O寄存器以及对它们的操作顺序。这些都和具体硬件相关，不同的外部设备可能要求不一样。
- 安全性: 多个用户程序可能共享某个外部设备资源，用户编写程序直接操作外部设备，一个小错误可能会影响到很多其它的用户程序。

并不是每个程序员知道（或者想知道）这个层次的技术细节。

解决方法:

提供服务子程序或者系统调用(通常作为操作系统的一部分)，用户通过这些子程序或者系统调用来安全和方便的实现底层操作。

理解：

系统调用

1. 用户程序使用系统调用
2. 操作系统执行调用操作
3. 结束后将控制权返回给用户程序

在LC-3里，通过***TRAP*** 机制来实现系统调用。

LC-3 TRAP 机制

1.包括一组服务子程序.

- 操作系统的一部分 – 服务程序起始固定的内存地址
  LC-3中实现的服务子程序位于系统代码区（below x3000)
- 最多支持 256个服务子程序

2.起始地址表.

- 存放在 x0000 到 x00FF 的内存中(256x16bit) 。每16位存放一个系统服务子程序的起始地址。
- 在别的系统中可能称为为“系统控制块”，或 “陷入矢量表”

3. TRAP 指令.

- 用户程序通过TRAP指令来实现系统调用。操作系统将以用户程序身份执行某个特定的服务程序，并在执行结束后将控制权返回
- LC-3: TRAP指令通过指令中8-bit的 trap vector来指示调用256个服务子程序中的哪一个。

4.链接回到用户程序.

- 提供从服务程序返回到用户程序的机制

TRAP指令

Trap 向量

- 指示调用哪个系统服务程序（x00-xff）
- 通过8位trapvect8索引*起始地址表，获得对应系统调用的入口地址*

  ØLC-3实现的方法：

  起始地址表存放在内存的 0x0000 – 0x00FF处，8-bit trap vector 通过高位0扩展成16位的内存地址，该内存地址处存放的就是相应调用的入口地址

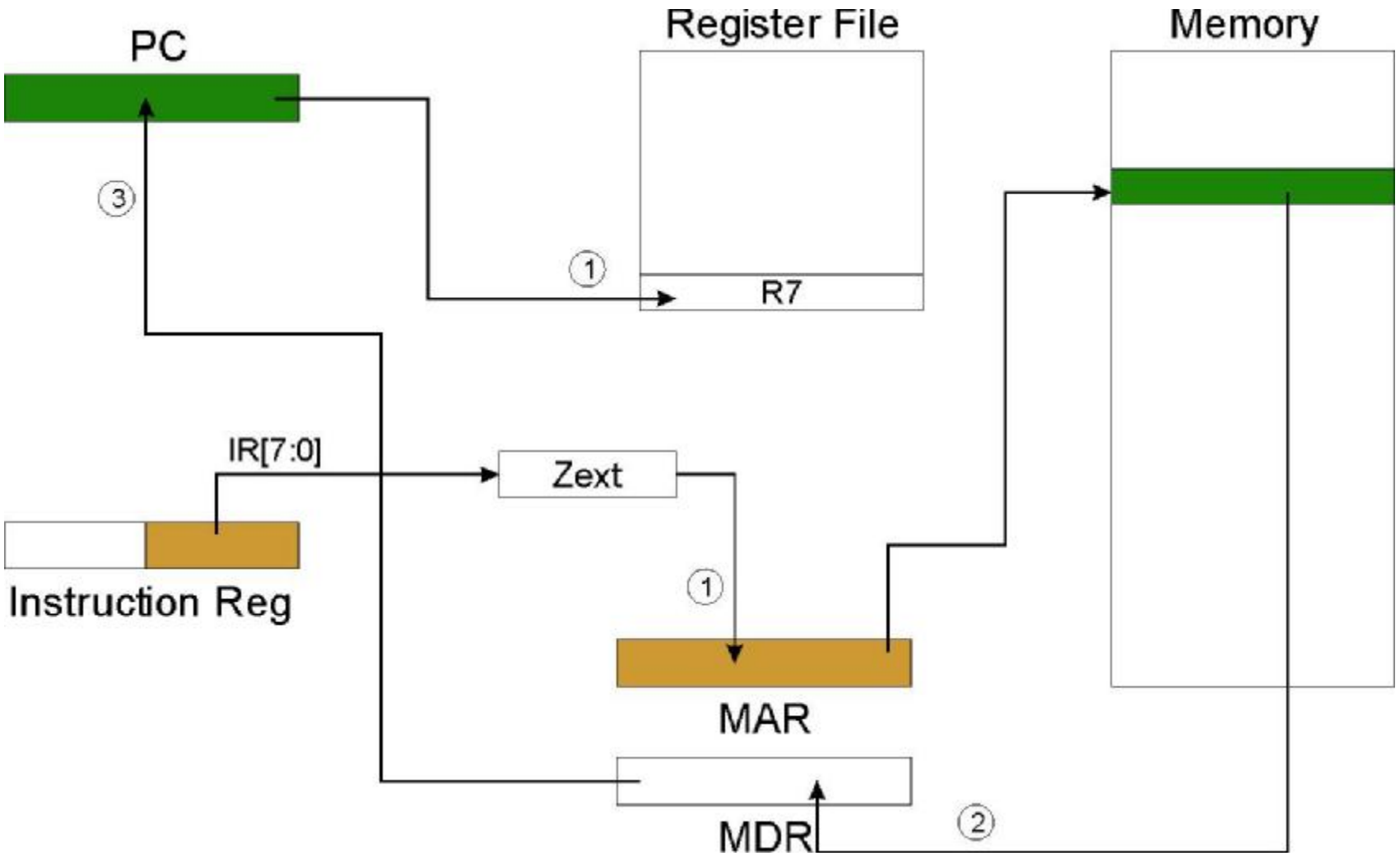
如何执行

- 从起始地址表查找服务程序地址，加载到PC中

如何返回

- 将下一条指令的地址（当前的PC值）保存到R7

TRAP



注: PC在指令获取阶段已经执行加1操作指向了当前指令的下一条指令

RET (JMP R7)

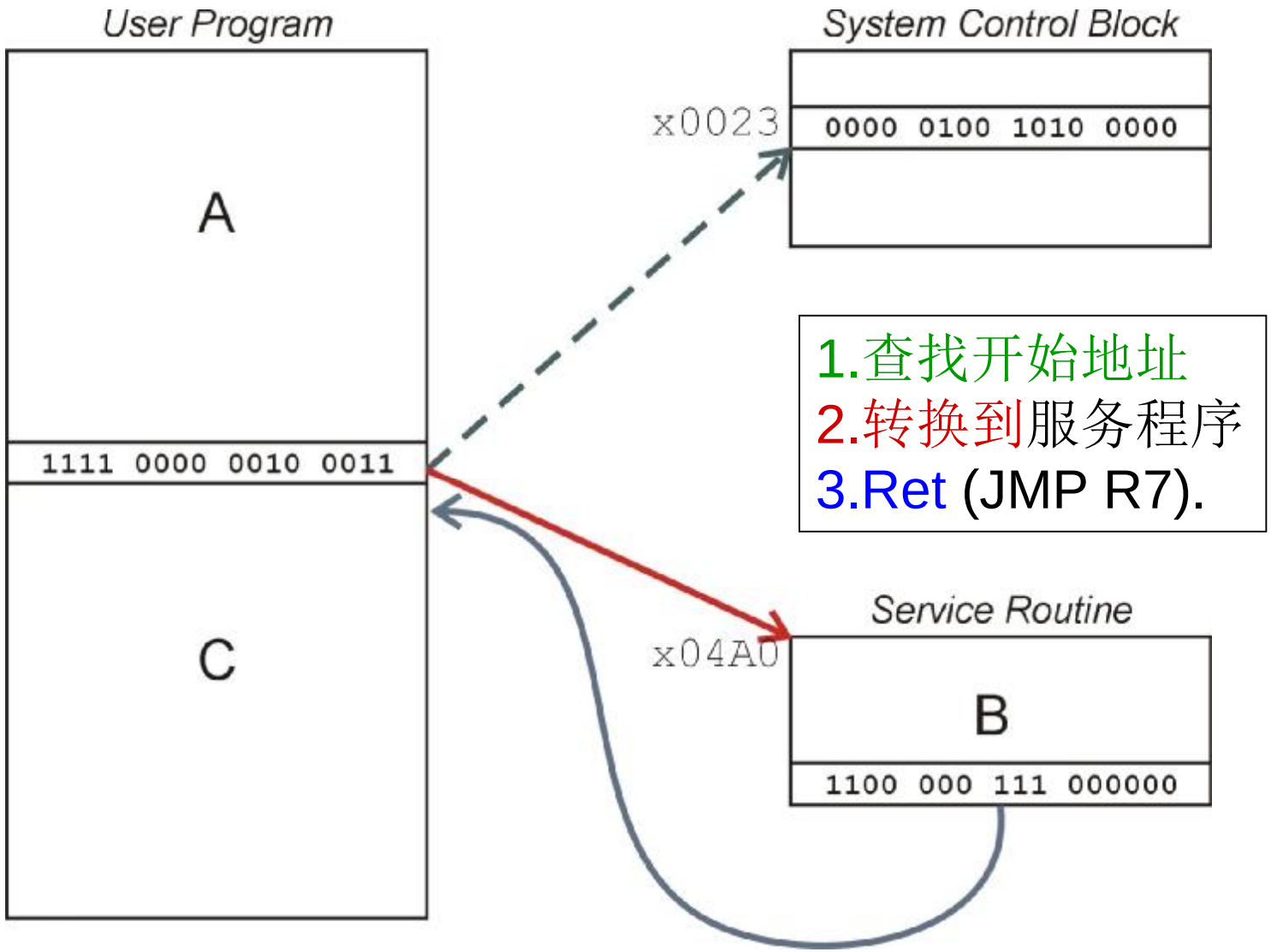
如何返回用户程序?即回到trap指令的下一条指令继续运行

执行trap指令时将PC保存在R7

- 服务程序使用JMP R7就可以返回到用户程序
- LC-3 汇编语言使用 RET (return) 助记符来取代 “JMP R7”.

因此：必须保证服务程序没有改变R7，否则无法返回.

TRAP 机制流程

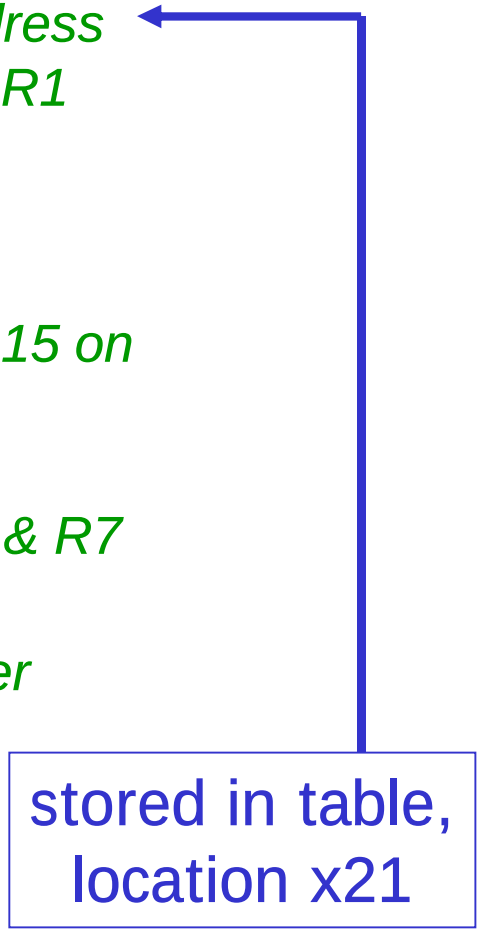

User Program
A
1111 0000 0010 0011
C
System Control Block
x0023
0000 0100 1010 0000
1.查找开始地址
2.转换到服务程序
3.Ret (JMP R7).
Service Routine
x04A0
B
1100 000 111 000000

Ex: 输入大写字母转换为小写字母，输入'7'结束

```
        .ORIG x3000
        LD   R2, TERM    ; Load negative ASCII '7'
        LD   R3, ASCII   ; Load ASCII difference
AGAIN   TRAP  x23        ; input character
        ADD  R1, R2, R0  ; Test for terminate
        BRz  EXIT        ; Exit if done
        ADD  R0, R0, R3  ; Change to lowercase
        TRAP  x21        ; Output to monitor...
        BRnzp  AGAIN     ; ... again and again...
TERM    .FILL xFFC9      ; -'7'
ASCII   .FILL x0020      ; lowercase bit
EXIT    TRAP  x25        ; halt
        .END
```

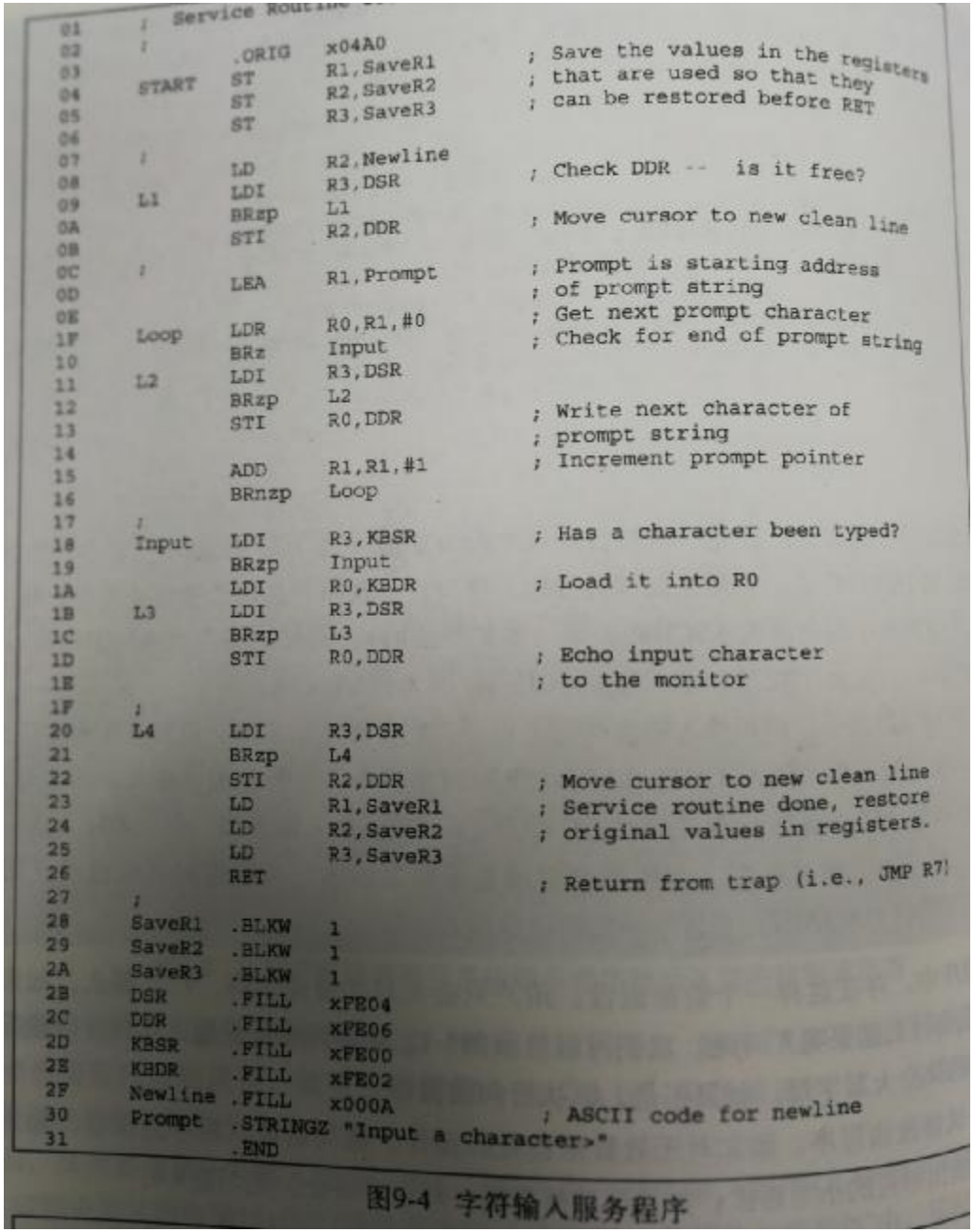
Example: Output Service Routine

```
            .ORIG x0430          ; syscall address
            ST    R7, SaveR7     ; save R7 & R1
            ST    R1, SaveR1
; ----- Write character
TryWrite    LDI   R1, CRTSR      ; get status
            BRzp  TryWrite       ; look for bit 15 on
WriteIt     STI   R0, CRTDR      ; write char
; ----- Return from TRAP
Return      LD    R1, SaveR1     ; restore R1 & R7
            LD    R7, SaveR7
            RET                  ; back to user
CRTSR       .FILL xF3FC
CRTDR       .FILL xF3FF
SaveR1      .FILL 0
SaveR7      .FILL 0
            .END
```



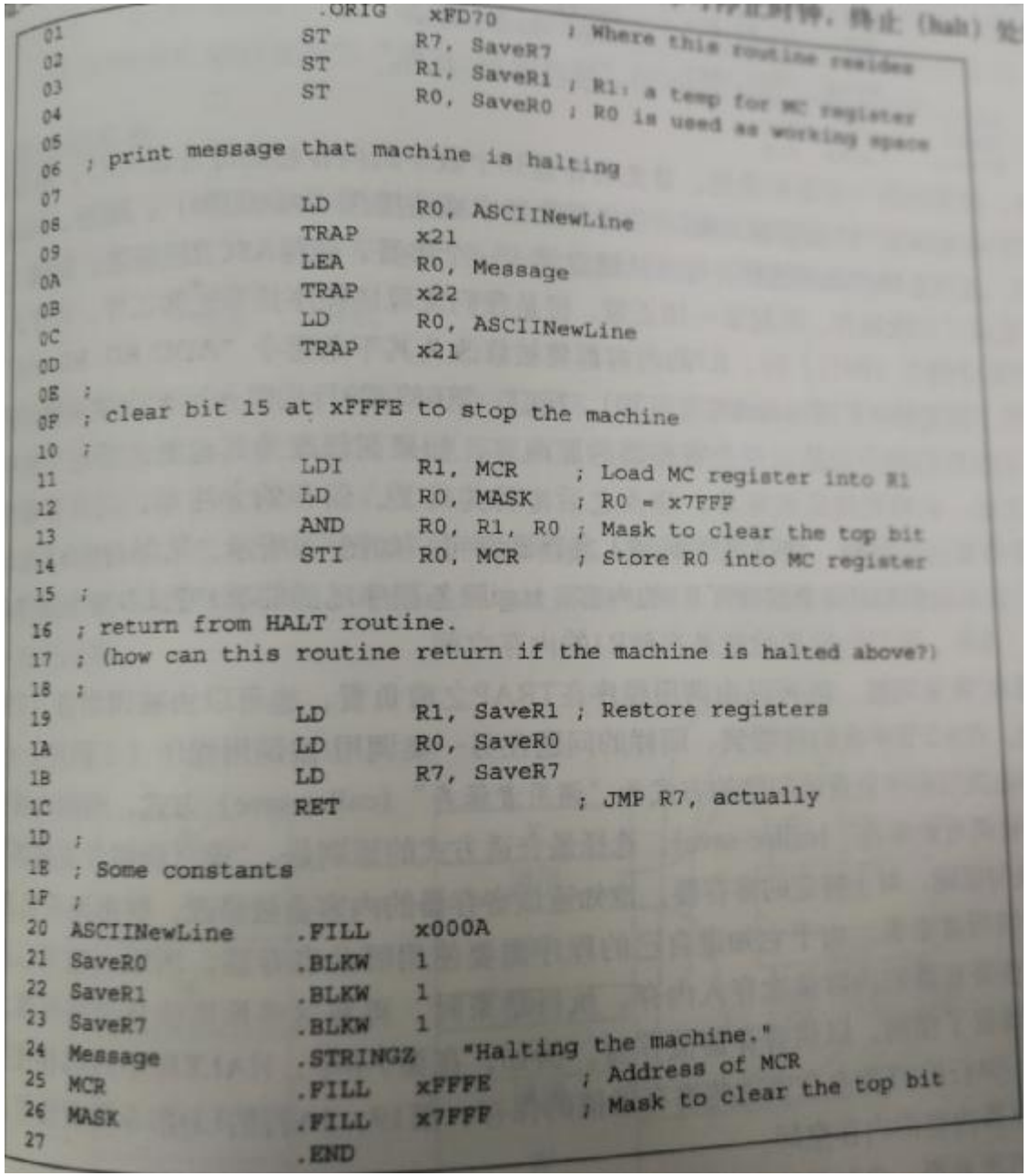
TRAP Routines and their Assembler Names

vector	*symbol*	*routine*
x20	**GETC**	read a single character (no echo)
x21	**OUT**	output a character to the monitor
x22	**PUTS**	write a string to the console
x23	**IN**	print prompt to console, read and echo character from keyboard
x25	**HALT**	halt the program



```
01      ; Service Routine
02      ;       .ORIG   x04A0
03              ST      R1,SaveR1       ; Save the values in the registers
04      START   ST      R2,SaveR2       ; that are used so that they
05              ST      R3,SaveR3       ; can be restored before RET
06
07      ;       LD      R2,Newline
08              LDI     R3,DSR          ; Check DDR -- is it free?
09      L1      BRzp    L1
0A              STI     R2,DDR          ; Move cursor to new clean line
0B
0C      ;       LEA     R1,Prompt       ; Prompt is starting address
0D                                      ; of prompt string
0E                                      ; Get next prompt character
1F      Loop    LDR     R0,R1,#0        ; Check for end of prompt string
10              BRz     Input
11      L2      LDI     R3,DSR
12              BRzp    L2
13              STI     R0,DDR          ; Write next character of
14                                      ; prompt string
15              ADD     R1,R1,#1        ; Increment prompt pointer
16              BRnzp   Loop
17      ;
18      Input   LDI     R3,KBSR         ; Has a character been typed?
19              BRzp    Input
1A              LDI     R0,KBDR         ; Load it into R0
1B      L3      LDI     R3,DSR
1C              BRzp    L3
1D              STI     R0,DDR          ; Echo input character
1E                                      ; to the monitor
1F      ;
20      L4      LDI     R3,DSR
21              BRzp    L4
22              STI     R2,DDR          ; Move cursor to new clean line
23              LD      R1,SaveR1       ; Service routine done, restore
24              LD      R2,SaveR2       ; original values in registers.
25              LD      R3,SaveR3
26              RET                     ; Return from trap (i.e., JMP R7)
27      ;
28      SaveR1  .BLKW   1
29      SaveR2  .BLKW   1
2A      SaveR3  .BLKW   1
2B      DSR     .FILL   xFE04
2C      DDR     .FILL   xFE06
2D      KBSR    .FILL   xFE00
2E      KBDR    .FILL   xFE02
2F      Newline .FILL   x000A           ; ASCII code for newline
30      Prompt  .STRINGZ "Input a character>"
31              .END
```

图9-4 字符输入服务程序



```
01                      .ORIG   xFD70           ; Where this routine resides
02                      ST      R7, SaveR7
03                      ST      R1, SaveR1      ; R1: a temp for MC register
04                      ST      R0, SaveR0      ; R0 is used as working space
05
06  ; print message that machine is halting
07
08                      LD      R0, ASCIINewLine
09                      TRAP    x21
0A                      LEA     R0, Message
0B                      TRAP    x22
0C                      LD      R0, ASCIINewLine
0D                      TRAP    x21
0E  ;
0F  ; clear bit 15 at xFFFE to stop the machine
10  ;
11                      LDI     R1, MCR         ; Load MC register into R1
12                      LD      R0, MASK        ; R0 = x7FFF
13                      AND     R0, R1, R0      ; Mask to clear the top bit
14                      STI     R0, MCR         ; Store R0 into MC register
15  ;
16  ; return from HALT routine.
17  ; (how can this routine return if the machine is halted above?)
18  ;
19                      LD      R1, SaveR1      ; Restore registers
1A                      LD      R0, SaveR0
1B                      LD      R7, SaveR7
1C                      RET                     ; JMP R7, actually
1D  ;
1E  ; Some constants
1F  ;
20  ASCIINewLine        .FILL   x000A
21  SaveR0              .BLKW   1
22  SaveR1              .BLKW   1
23  SaveR7              .BLKW   1
24  Message             .STRINGZ   "Halting the machine."
25  MCR                 .FILL   xFFFE           ; Address of MCR
26  MASK                .FILL   x7FFF           ; Mask to clear the top bit
27                      .END
```

```
01      ; This service routine writes a NULL-terminated string to the console.
02      ; It services the PUTS service call (TRAP x22).
03      ; Inputs: R0 is a pointer to the string to print.
04      ;
05                  .ORIG   x0450           ; Where this ISR resides
06                  ST      R7, SaveR7      ; Save R7 for later return
07                  ST      R0, SaveR0      ; Save other registers that
08                  ST      R1, SaveR1      ; are needed by this routine
09                  ST      R3, SaveR3      ;
0A      ;
0B      ; Loop through each character in the array
0C      ;
0D      Loop        LDR     R1, R0, #0      ; Retrieve the character(s)
0E                  BRz     Return          ; If it is 0, done
0F      L2          LDI     R3,DSR
10                  BRzp    L2
11                  STI     R1, DDR         ; Write the character
12                  ADD     R0, R0, #1      ; Increment pointer
13                  BRnzp   Loop            ; Do it all over again
14      ;
15      ; Return from the request for service call
16      Return      LD      R3, SaveR3
17                  LD      R1, SaveR1
18                  LD      R0, SaveR0
19                  LD      R7, SaveR7
1A                  RET
1B      ;
1C      ; Register locations
1D      DSR         .FILL   xFE04
1E      DDR         .FILL   xFE06
1F      SaveR0      .FILL   x0000
20      SaveR1      .FILL   x0000
21      SaveR3      .FILL   x0000
22      SaveR7      .FILL   x0000
23                  .END
```

图9-9 LC-3的PUTS服务程序（续）

寄存器内容的保存和恢复

在以下情况，必须要保存寄存器的内容:

- 如果该寄存器的内容会被系统调用使用
- 并在后续操作中将修改该寄存器.

谁保存?

- 调用服务程序的程序（调用者保存）?
  - Ø需要知道系统调用会修改或使用那些寄存器什么，但实际可能不知道服务程序会使用和修改哪些寄存器
- 被调用的服务程序（被调用者保存）?
  - Ø知道自己修改的内容，但不知道调用程序后面需要什么
  - Ø全部保存

Example

```
            LEA   R3, Binary
            LD    R6, ASCII   ; char->digit template
            LD    R7, COUNT   ; initialize to 10
AGAIN       TRAP  x23         ; Get char
            ADD   R0, R0, R6  ; convert to number
            STR   R0, R3, #0  ; store number
            ADD   R3, R3, #1  ; incr pointer
            ADD   R7, R7, -1  ; decr counter
            BRp   AGAIN       ; more?
            BRnzp NEXT
ASCII       .FILL xFFD0
COUNT       .FILL #10
Binary      .BLKW #10
```

What's wrong with this routine?
What happens to R7?

寄存器内容的保存和恢复

被调用者保存-- *“callee-save”*

- 在开始之前，保存可能被修改的寄存器内容（除非调用程序预先知道会被修改-**EX.** 参数）
- 在返回前，恢复这些寄存器内容

调用者保存-- *“caller-save”*

- 针对将被调用程序或被调用程序（如果知道的话）修改的寄存器，如果其内容后面需要，将其内容保存下来
  - Ø在**TRAP**之前保存**R7**
  - Ø在**TRAP x23 (input character)** 之前保存**R0**
- 或者避免使用那些寄存器

保存在哪里：内存

Question

一个服务程序能否调用另一个服务程序？

如果可以的话，调用程序需要做什么？

用户代码

服务程序提供3个主要功能：

1. 程序员与系统相关的细节隔离
2. 频繁使用的代码只需要写一次
3. 保护系统资源免受恶意/笨拙的程序员影响

对用户程序是否可以提供类似的功能？

子程序

一个子程序是一个满足以下条件的程序片段：

- 在用户空间运行 **lives in user space**
- 执行一个预定义好的任务
- 被另一个用户程序调用
- 执行完毕后，将控制权返回给调用程序

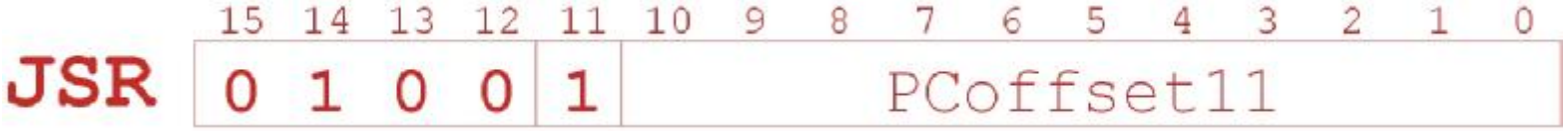
与服务程序类似，但不是操作系统的一部分

- 不与受保护的硬件资源打交道
- 不需要特权

为什么需要子程序？

- 重用有用的（调试好的！）代码
- 在多个程序员之间划分任务
- 使用供应商提供的代码库

JSR指令

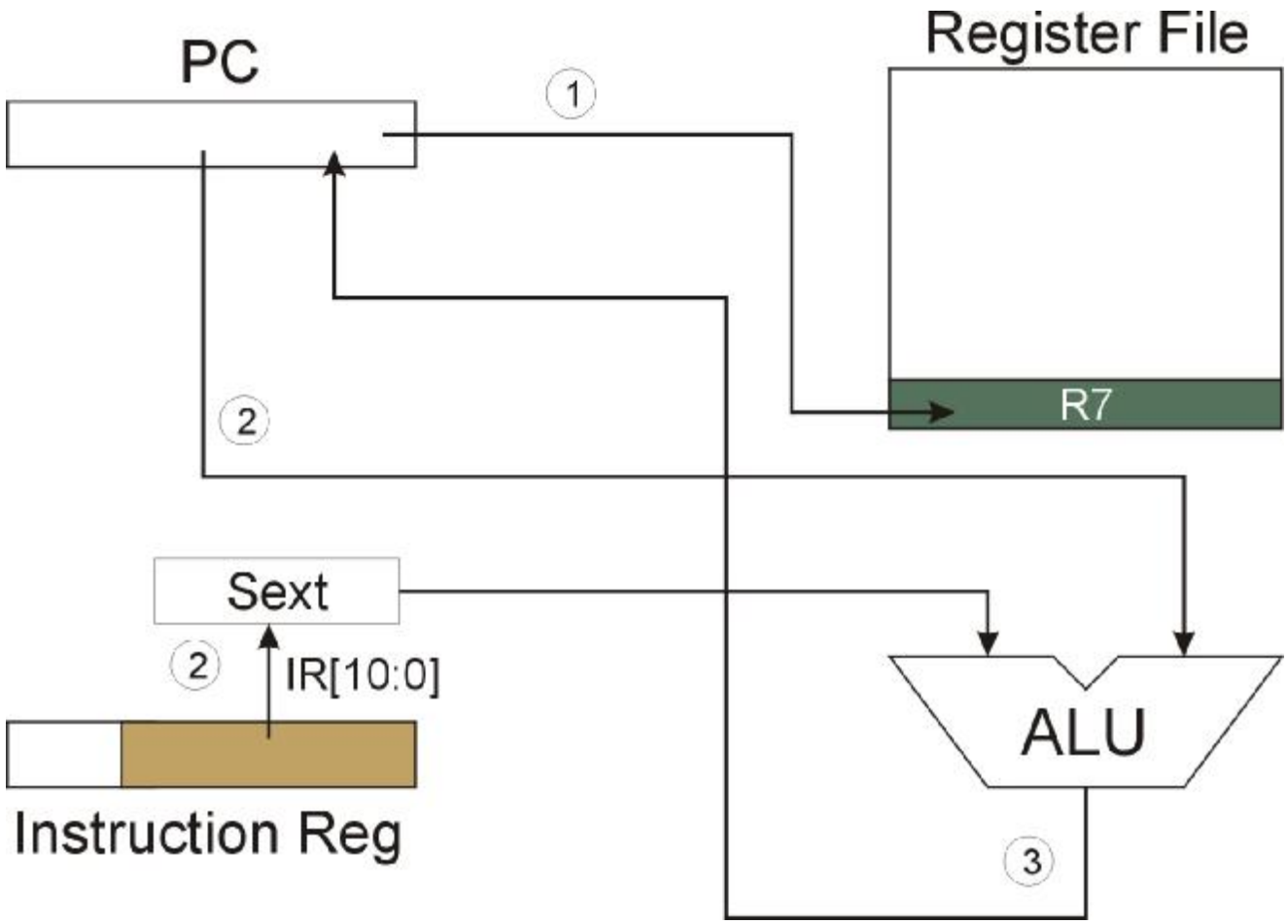


保存当前的PC（下一条指令的地址）到R7，跳到一个指定的位置（PC相对寻址）。

- 保存返回地址叫做 “linking”
- 目标地址是 PC-relative (PC + Sext(IR[10:0])) -1024~+1023
- 第11个bit位定义寻址模式：
  - Ø如果为1，PC相对寻址：目标地址 = PC + Sext(IR[10:0])
  - Ø如果为0，寄存器寻址：目标地址 = 寄存器内容IR[8:6]

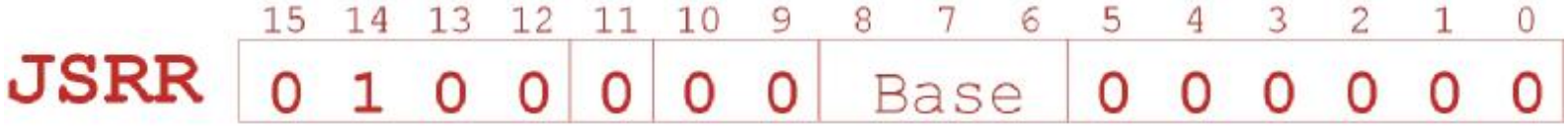
能用BRnzp 指令取代吗？

JSR



NOTE: PC has already been incremented during instruction fetch stage.

JSRR Instruction

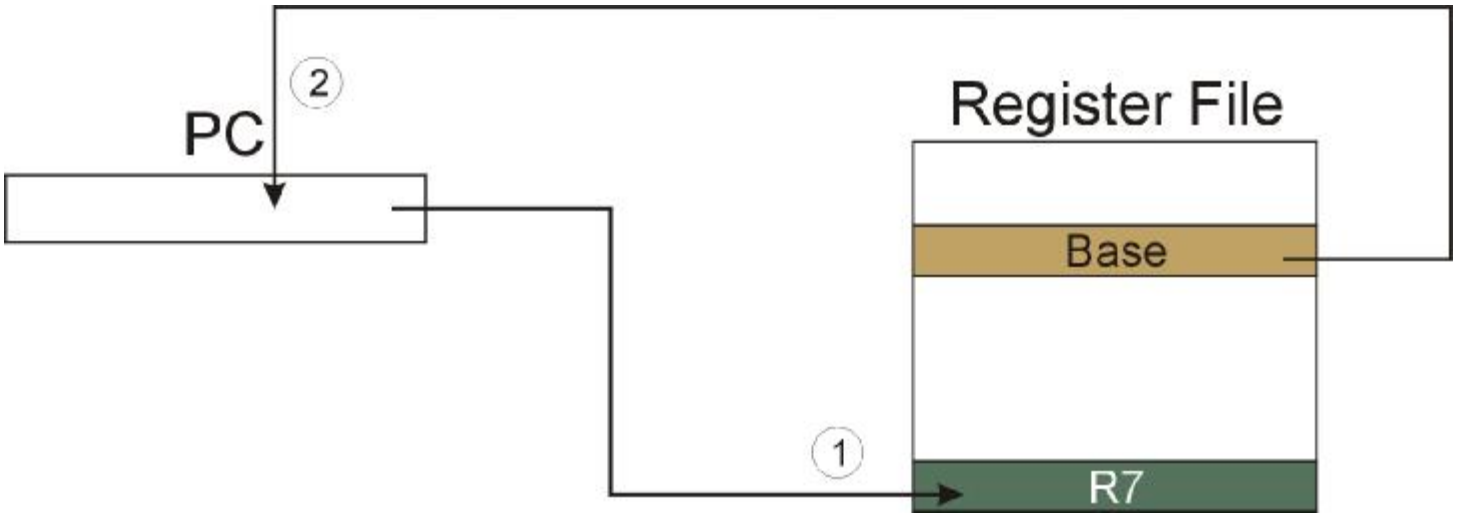


和JSR相似，除了寻址模式不同

- 目标地址是寄存器基址
- 第11个bit位定义了寻址模式

思考：JSSR具备的什么特征是JSR指令没有的？

JSRR



NOTE: PC has already been incremented during instruction fetch stage.

子程序返回

和系统调用类似：

RET (JMP R7)

Example: 对R0的数求反+1

```
2sComp    NOT   R0, R0       ; flip bits
          ADD   R0, R0, #1    ; add one
          RET                 ; return to caller
```

To call from a program (within 1024 instructions):

```
; need to compute R4 = R1 - R3
          ADD   R0, R3, #0    ; copy R3 to R0
          JSR   2sComp        ; negate
          ADD   R4, R1, R0    ; add to R1
          ...
```

Note: R0是否需要保存？

思考：子程序定义的位置？

```
01                .ORIG   x04A0
02  START         ST      R7,SaveR7
03                JSR     SaveReg
04                LD      R2,Newline
05                JSR     WriteChar
06                LEA     R1,PROMPT
07  ;
08  ;
09  Loop          LDR     R2,R1,#0    ; Get next prompt char
0A                BRz     Input
0B                JSR     WriteChar
0C                ADD     R1,R1,#1
0D                BRnzp   Loop
0E  ;
0F  Input         JSR     ReadChar
10                ADD     R2,R0,#0    ; Move char to R2 for writing
11                JSR     WriteChar   ; Echo to monitor
12  ;
13                LD      R2, Newline
14                JSR     WriteChar
15                JSR     RestoreReg
16                LD      R7,SaveR7
17                RET                 ; JMP R7 terminates
18  ;                                   the TRAP routine
19  SaveR7        .FILL   x0000
1A  Newline       .FILL   x000A
1B  Prompt        .STRINGZ  "Input a character>"
1C  ;
1D  WriteChar     LDI     R3,DSR
1E                BRzp    WriteChar
1F                STI     R2,DDR
20                RET                 ; JMP R7 terminates subrcutine
```

```
21  DSR           .FILL   xFE06
22  DDR
23  ;
24  ReadChar      LDI     R3,KBSR
25                BRzp    ReadChar
26                LDI     R0,KBDR
27                RET
28  KBSR          .FILL   xFE00
29  KBDR          .FILL   xFE02
2A  ;
2B  SaveReg       ST      R1,SaveR1
2C                ST      R2,SaveR2
2D                ST      R3,SaveR3
2E                ST      R4,SaveR4
2F                ST      R5,SaveR5
30                ST      R6,SaveR6
31                RET
32  ;
33  RestoreReg    LD      R1,SaveR1
34                LD      R2,SaveR2
35                LD      R3,SaveR3
36                LD      R4,SaveR4
37                LD      R5,SaveR5
38                LD      R6,SaveR6
39                RET
3A  SaveR1        .FILL   x0000
3B  SaveR2        .FILL   x0000
3C  SaveR3        .FILL   x0000
3D  SaveR4        .FILL   x0000
3E  SaveR5        .FILL   x0000
3F  SaveR6        .FILL   x0000
40                .END
```

子程序的参数及返回值

参数

- 传递给子程序的值，称为参数
- 子程序需要利用该值来完成任务
- **Examples:**
  - Ø在 **2sComp** 程序中, **R0**存放需要取负**+1**的数
  - Ø在输出服务程序中，**R0**存放需要被打印的字符
  - Ø在 **PUTS**服务程序中, **R0**存放打印字符串的起始地址

返回值

- 由子程序输出的值，称为子程序的返回值，
- 是子程序的计算结果
- **Examples:**
  - Ø在**2sComp**程序中，取负**+1**的结果保存在 **R0**
  - Ø在**GETC**服务程序中，由键盘输入的字符保存在 **R0**

如何使用子程序

为了使用子程序，程序员必须知道：

- 子程序调用地址(子程序标号：入口指令的标号)
- 功能(what does it do?)
  - Ø注：程序员不需要知道子程序如何工作，但需要知道子程序运行后机器的状态会发生什么变化
- 参数(怎么传递数据给子程序，如果需要)
- 返回值 (怎么获取子程序的计算结果，如果有的话)

保存和恢复寄存器

由于子程序和服务程序类似，如果有必要的话，也需要保存和恢复寄存器内容

一般情况使用“被调用者模式”，除了返回值

- 保存任何子程序内部将修改，但子程序返回时不应该改变的值
- 一般情况下，返回后恢复输入的参数为初始值（除非被返回值修改）

注意: 如果调用任何子程序或服务程序，必须保存**R7**

- 否则，将不能返回调用程序

Example

(1)编写一个子程序 FirstChar ,实现以下功能:

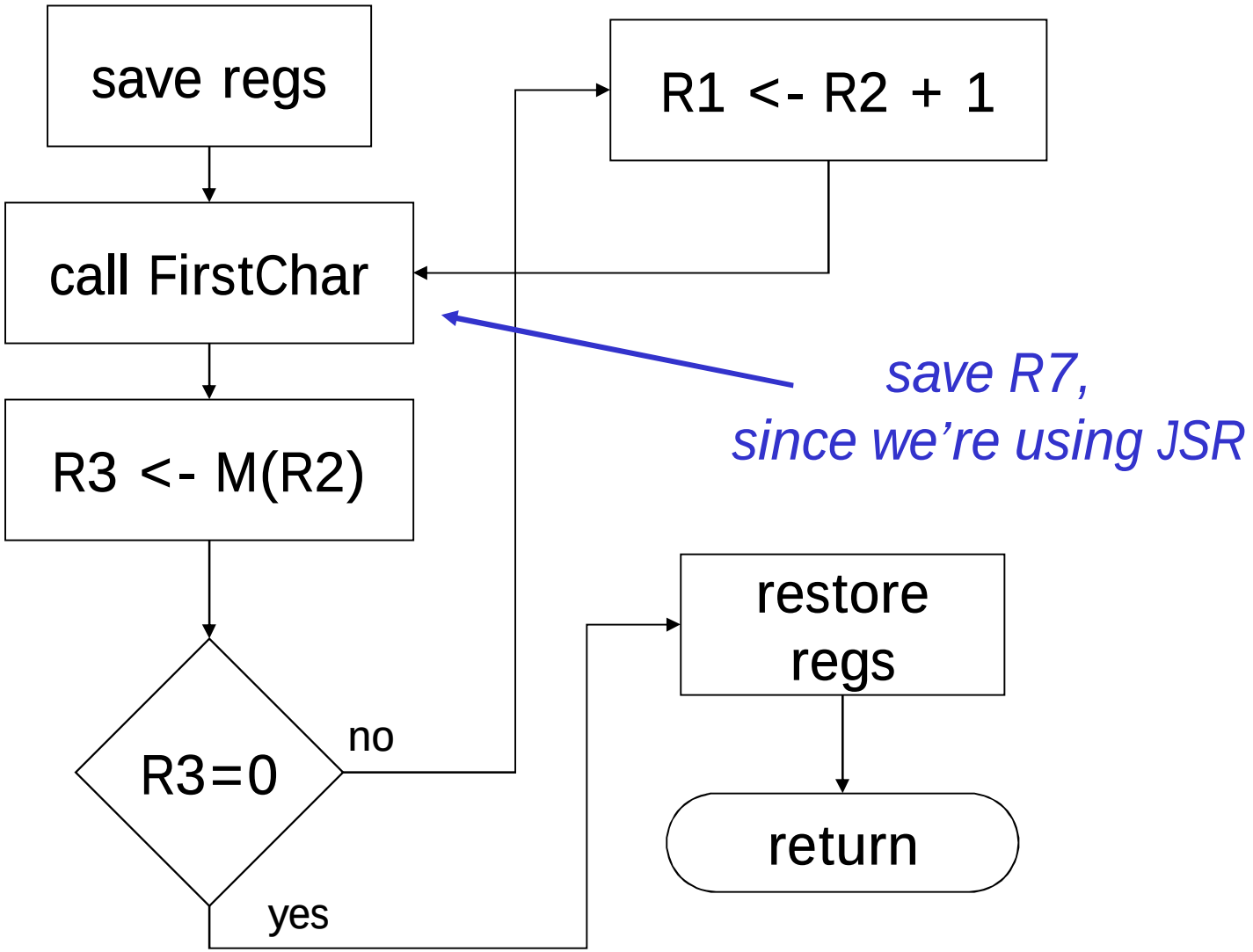
找到一个给定字符（字符ASCII码保存在R0 ）在一个字符串（ R1 指向字符串首地址）中第一次出现的位置；返回所在内存地址信息在R2 （内存位置如果指向NULL表示没有该字符出现）。

(2)使用FirstChar来实现算法CountChar，实现以下功能:

计算一个给定字符（保存在R0 ）在一个字符串（ R1 指向）中的出现次数；结果保存在在R2。

可以在不知道FirstChar实现的情况下写CountChar算法。

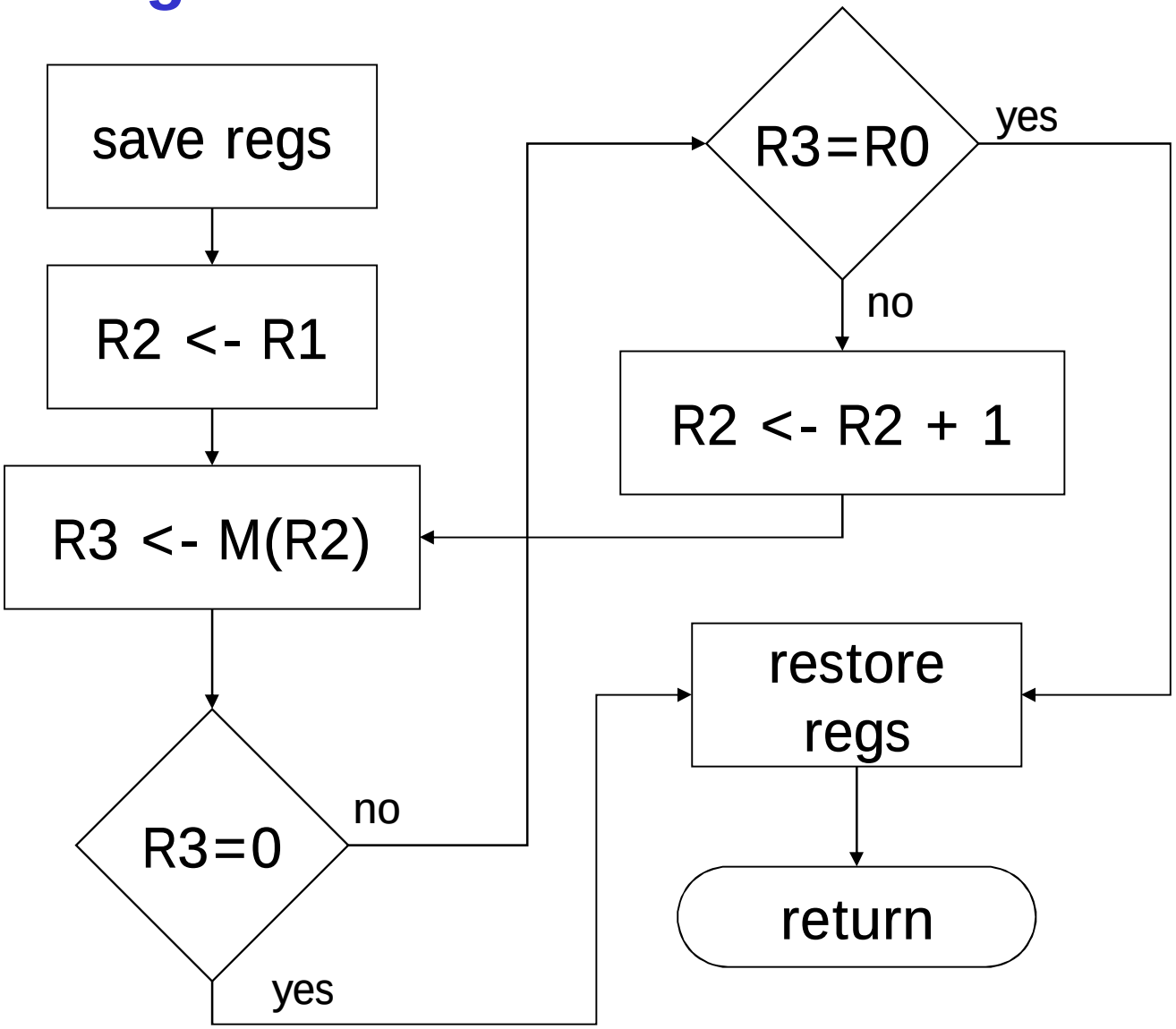
CountChar Algorithm (using FirstChar)



CountChar Implementation

```
;  CountChar: subroutine to count occurrences of a char
CountChar
        ST     R3, CCR3      ;  save registers
        ST     R4, CCR4
        ST     R7, CCR7      ;  JSR alters R7
        ST     R1, CCR1      ;  save original string ptr
        AND    R4, R4, #0    ;  initialize count to zero
CC1     JSR    FirstChar     ;  find next occurrence (ptr in R2)
        LDR    R3, R2, #0    ;  see if char or null
        BRz    CC2           ;  if null, no more chars
        ADD    R4, R4, #1    ;  increment count
        ADD    R1, R2, #1    ;  point to next char in string
        BRnzp  CC1
CC2     ADD    R2, R4, #0    ;  move return val (count) to R2
        LD     R3, CCR3      ;  restore regs
        LD     R4, CCR4
        LD     R1, CCR1
        LD     R7, CCR7
        RET                  ;  and return
```

FirstChar Algorithm



FirstChar Implementation

```
;  FirstChar: subroutine to find first occurrence of a char
FirstChar
        ST     R3, FCR3      ;  save registers
        ST     R4, FCR4      ;  save original char
        NOT    R4, R0        ;  negate R0 for comparisons
        ADD    R4, R4, #1
        ADD    R2, R1, #0    ;  initialize ptr to beginning of string
FC1     LDR    R3, R2, #0    ;  read character
        BRz    FC2           ;  if null, we're done
        ADD    R3, R3, R4    ;  see if matches input char
        BRz    FC2           ;  if yes, we're done
        ADD    R2, R2, #1    ;  increment pointer
        BRnzp  FC1
FC2     LD     R3, FCR3      ;  restore registers
        LD     R4, FCR4      ;
        RET                  ;  and return
```

例子：计算标量积

标量积的定义：**DP=sum (X[i] * Y[i]) for i=0 to n-1**

调用子程序**PROD**完成：**Mutliply R4 by R5 and add the result to R0**

数据区定义；

```
COUNT    .FILL #6
X        .FILL #31     ; X[0]
         .FILL #-12    ; X[1]
         .FILL #65     ; X[2]
         .FILL #27     ; X[3]
         .FILL #34     ; X[4]
         .FILL #-43    ; X[5]
Y        .FILL #22     ; Y[0]
         .FILL #-8     ; Y[1]
         .FILL #-57    ; Y[2]
         .FILL #33     ; Y[3]
         .FILL #70     ; Y[4]
         .FILL #-53    ; Y[5]
DOTPROD  .FILL #0
```

Example code:

```
        .orig x3000
        AND R0,R0,#0 ; Zero dot product
        LEA R2,X
        LEA R3,Y
        LD  R1,COUNT
        ADD R1,R1, #-1 ; Start i at n-1
        BRN DONE ; Check for n=0
        ADD R2,R2,R1
        ADD R3,R3,R1
LOOP    LDR R4,R2, #0
        LDR R5,R3, #0
        JSR PROD
        ADD R2,R2, #-1
        ADD R3,R3, #-1
        ADD R1,R1, #-1
        BRZP LOOP
DONE    ST  R0,DOTPROD
        HALT
```

课后研究以下乘法代码，适合负数乘法吗，自己仿真运行下

```
PROD    ST R7,PSAVER7
        ST R6,PSAVER6
        AND R6,R6,#0
        ADD R6,R6, #1 ; R6 is mask reg
PLOOP   AND R7,R6,R4
        BRZ PZERO
        ADD R0,R0,R5
PZERO   ADD R5,R5,R5 ; Double R5 (shift left 1)
        ADD R6,R6,R6 ; Double R6 (shift left 1)
        BRNP PLOOP
        LD R6,PSAVER6
        LD R7,PSAVER7
        RET
PSAVER6         .FILL #0
PSAVER7         .FILL #0
```

求平均-分析以下代码，完成子程序的设计

```
        .orig x3000
        LEA R0,VECTOR
        LD  R1,COUNT
        JSR SUMVEC
        LD R1,COUNT
        JSR DIVIDE
        ST R4,AVG
        HALT
COUNT   .FILL   #14
VECTOR  .FILL   #63     ; VEC[0]
        .FILL   #21     ; VEC[1]
        .FILL   #-90    ; VEC[2]
        .FILL   #32     ; VEC[3]
        .FILL   #312    ; VEC[4]
        .FILL   #114    ; VEC[5]
        .FILL   #20     ; VEC[6]
        .FILL   #-3     ; VEC[7]
        .FILL   #201    ; VEC[8]
        .FILL   #34     ; VEC[9]
        .FILL   #21     ; VEC[10]
        .FILL   #111    ; VEC[11]
        .FILL   #53     ; VEC[12]
        .FILL   #601    ; VEC[13]
        AVG     .FILL   #0
        .end
```

分析：

SUMVEC：

入口参数 **R0 à** 数组指针

R1->数组大小

返回参数：自定义（**R2**），提供给**DIVIDE**子程序作为入口参数

DIVIDE: 假设除数和被除数都为正，商为整数，余数不处理

入口参数 **R2 à** 除数，**R2 > 0**

R1 - >被除数

返回参数：**R4**

入口参数　　R0 à 数组指针

　　R1->数组大小

返回参数：　　R2->数组和

```
SUMVEC      ST R3, PSAVER3
            AND R2,R2,#0
SUMLOOP       LDR R3,R0,0
              ADD R2,R2,R3
             ADD R0,R0,1
             ADD R1,R1,-1
             BRP SUMLOOP
            LD R3, PSAVER3
             RET
 PSAVER3    .FILL 0
```

```
DIVIDE      AND R4,R4,#0
            NOT R1,R1
            ADD R1,R1,1
            ADD R2,R2,#0
            BRNZ ENDDIV
DIVLOOP     ADD R4,R4,1
            ADD R2,R2,R1
            BRP DIVLOOP; P 继续
            BRZ ENDDIV ;  Z  整除
            ADD R4,R4,-1 ;  N 不够除
ENDDIV      RET
```

入口参数
R2除R1
返回参数：
R4 商

库程序

开发者可能提供包含有用的子程序的目标文件

- 不想提供源代码– 知识产权的问题
- 汇编器/链接器必须支持外部符号(或程序起始地址必须给用户)

```
            ...
            .EXTERNAL SQRT
            ...
            LD   R2, SQAddr    ; load SQRT addr
            JSRR R2
            ...
SQAddr      .FILL SQRT
```

用**JSRR**, 因为不知道**SQRT**是否在**1024**条指令范围内。

作业

Ex 9.2, 9.4, 9.5, 9.11

Ex 9.17, 9.18

Ex 9.8, 9.12, 9.13