Lec02 Repeated Code: TRAPs and Subroutines

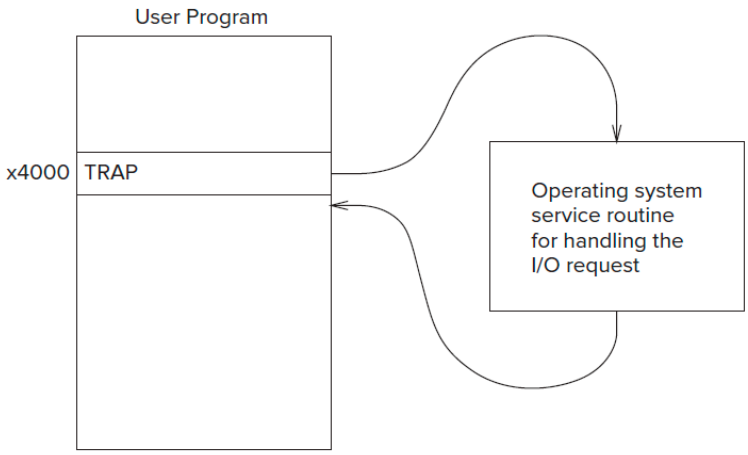
Lec1中I/O操作的缺点：

1. 需要硬件知识
2. 可能混淆硬件寄存器

解决方案：TRAP 服务程序

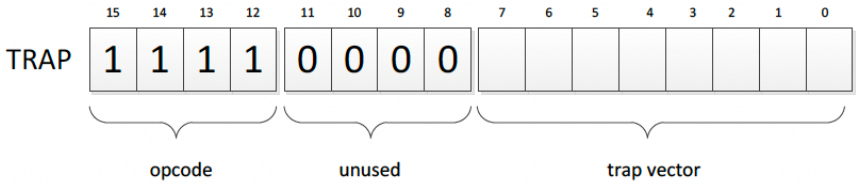
需要提供服务程序或者系统调用（操作系统的一部分）以安全方便地执行低级特权操作：

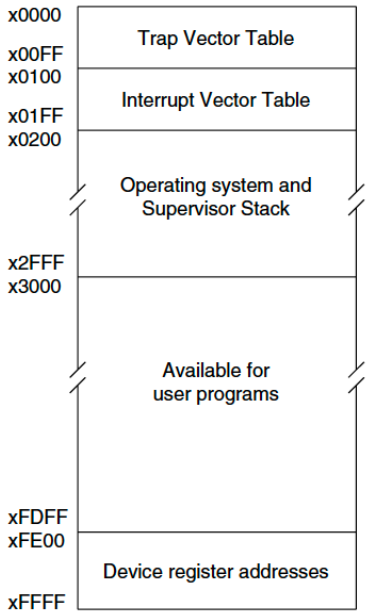
1. 用户程序调用系统调用
2. 操作系统代码执行操作
3. 将控制权返回给用户程序



TRAP的工作机制

- 服务程序地址存储在**Trap Vector Table**（内存地址为x0000~x00FF）
- TRAP指令（如TRAP x20）使用8位二进制数字定位服务地址





以TRAP x20为例子，在主程序中调用TRAP x20时依次发生以下操作：

1. **保存当前PC（程序计数器）到R7**
   - 假设用户程序在内存地址x3000处有一条指令`TRAP x20`
   - 当这条指令被取出来执行时，PC已经自增到0x3001（指向下一条指令）。
   - 执行TRAP x20时，PC的值0x3001被保存到R7，即 **R7 = 0x3001**。
2. **计算Trap Vector Table的地址**
   - LC-3直接使用x20作为偏移量，加上基地址x0000，得到地址x0020。
3. **从Trap Vector Table加载服务例程到PC**
   - LC-3会从刚刚计算出的地址（0x0020）读取一个值，这个值是服务例程的**起始地址**。
   - 这个地址被加载到**PC**中，程序就跳转到服务例程的代码开始执行。
   - 假设内存地址0x0020处存储的值是0x2000。执行这一步后，**PC = 0x2000**，程序跳转到0x2000处。
4. **执行服务例程**
   - 现在PC指向服务例程的起始地址（0x2000），LC-3开始执行这里的代码。
   - 对于TRAP x20，它通常是**GETC服务**，功能是等待用户输入一个字符，并将该字符的ASCII码存储到**寄存器R0**中。
5. **返回用户程序**
   - 服务例程以**RET指令**（相当于JMP R7）结束。
   - RET指令会将R7中保存的值（也就是之前的PC值）加载回PC。
   - 这样，程序就返回到用户程序的下一条指令继续执行。

常见的TRAP程序

Vector	Symbol	Routine
x20	GETC	read a single character into R0 (no echo)
x21	OUT	output a character in R0 to the monitor
x22	PUTS	write a string to the console (addr in R0)
x23	IN	print prompt to console, read and echo character from keyboard (R0)
x24	PUTSP	write a string to the console (2 characters per memory location) (addr in R0)
x25	HALT	halt the program

子程序(Subroutines)：用户定义代码复用

子程序是用户定义的可重复使用的代码块。

- 用户程序调用子程序。
- 子程序执行特定任务。
- 返回控制权，不改变其他状态。

JSR和JSRR

JSR:Junp to Subroutine

- 使用示例：

```
.ORIG x3000
LD R1, A      ; 加载A到R1
LD R2, B      ; 加载B到R2
JSR SUB       ; 调用子程序SUB
HALT
SUB           ; 子程序：计算R0 = R1 - R2
    NOT R2, R2
    ADD R2, R2, #1
    ADD R0, R1, R2
    RET
A .FILL #4
B .FILL #2
.END
```

- 工作原理：
    - 首先将主程序的PC存储到R7中。
    - 然后进入子程序，在子程序中最好不要对R7中的值有任何改变，如果需要改变，需要先ST，然后在子程序的末尾再LD。
    - 在子程序的末尾的RET中程序将R7中的值加载到PC中，返回主程序。

JSRR:Jump to Subroutine Register

- 使用示例：

```
.ORIG x3000
LD R1, A
LD R2, B
LEA R4, SUB   ; 将SUB地址加载到R4
JSRR R4       ; 调用子程序
HALT
SUB
    NOT R2, R2
    ADD R2, R2, #1
    ADD R0, R1, R2
    RET
A .FILL #4
B .FILL #2
.END
```

- 工作原理：同上，只不过进入的是寄存器中间内容所指向的地址

使用子程序的要求

- **程序员需了解**：

  子程序地址（或标签）、子程序功能、参数（输入数据）、返回值（输出数据）

- **示例**：
  - OUT：R0为要打印的字符。
  - PUTS：R0为字符串地址。
  - GETC：返回值在R0。

嵌套子程序问题（寄存器保存与恢复）

代码：

```
.ORIG x3000
LD R1, A
LD R2, B
JSR SUB
HALT
SUB
    NOT R2, R2
    ADD R2, R2, #1
    ADD R0, R1, R2
    ADD R3, R0, #0
    JSR ODD_EVEN  ; 嵌套调用
    RET
ODD_EVEN
    AND R4, R4, #0
    ADD R4, R4, #1
    AND R4, R3, R4
```

```
    RET
A .FILL #4
B .FILL #2
.END
```

问题：嵌套调用的时候R7被覆盖，返回地址丢失。