Introduction to Computer System Organization

SZU Review

Chapter8

General

输入/输出设备

1. 基本原理:

   编址方式: 内存映射、独立编址

   传输时序控制: 同步、异步

   轮询与中断

2. LC-3 Keyboard and Display 的实现 **(Ban IN,OUT)**

   (1). KBSR/KBDR 轮询的代码原理

   (2). DSR/DDR 代码原理

   (3). 中断的工作原理 **(Concept)**

   ```
   中断通知

   中断响应:保存现场(硬件上多个状态机实现)

   中断号 --> 定位入口地址

   中断返回:  RET
   ```

   (4). 中断程序格式

输入与输出 (Variety)

1. 利用Register中的值进行计算
2. 将data从Memory装载到Register
3. 将data从Register存储到Memory

(I/O)

Question:

1. 内存中的数据是从哪里来的?
2. 人使用的数据是怎样通过计算机系统传出去的?

Answer: 外部设备(外设:e.g. 键盘，显示器)

Rate:

(How fast can the data transform.)

keyboard: 100 bytes/sec

U 盘 : 30 - 40 MB/s (2.0 480 mbps) 5GB/s - 10GB/s (3.1 gen1/gen2)

Internet: 1 MB/s - 1 GB/s - 10 GB/s **Obvious Difference (Variety)**

慢速的外部设备如何接到快速的总线上?

solu: 和CPU 内部总线连接

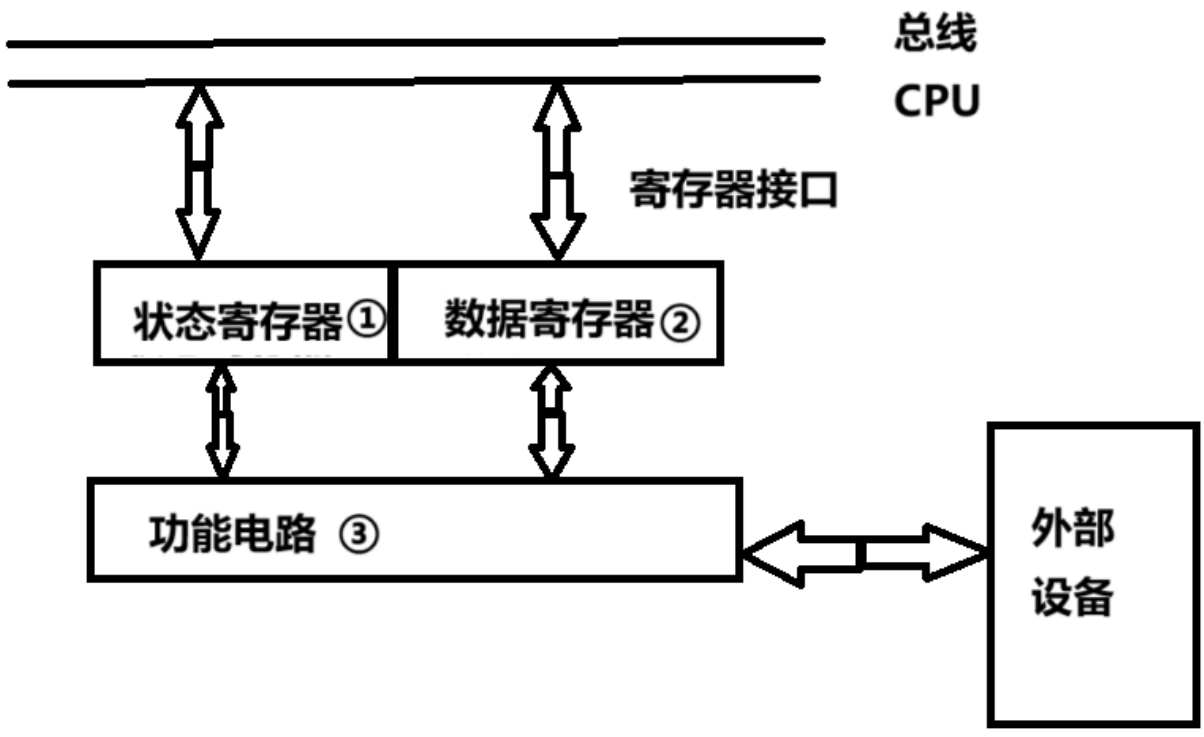
输入/输出控制器:

(与外部设备直接相连)

Function:

1. 格式转换(统一转换为寄存器接口格式的数据)
2. 缓存(不是来一个传一个,而是等达到一定数目一起传)(fit rate!)

comment:先缓存数据再等待处理器读取



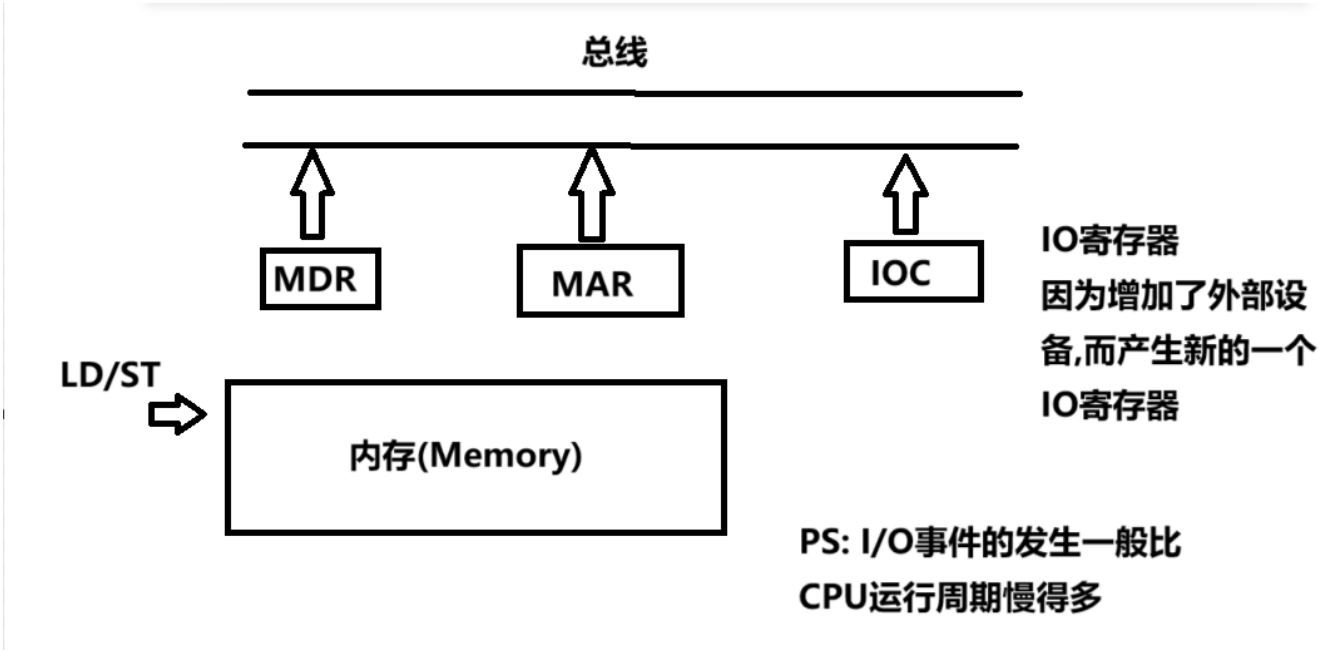
①. CPU 告诉我们设备做什么---Write (控制寄存器)

CPU check if tasks 've done --- Read (状态寄存器)

e.g. check the state of extern equipment:显示器是否空闲

②. CPU通过数据寄存器从设备 read/write 数据

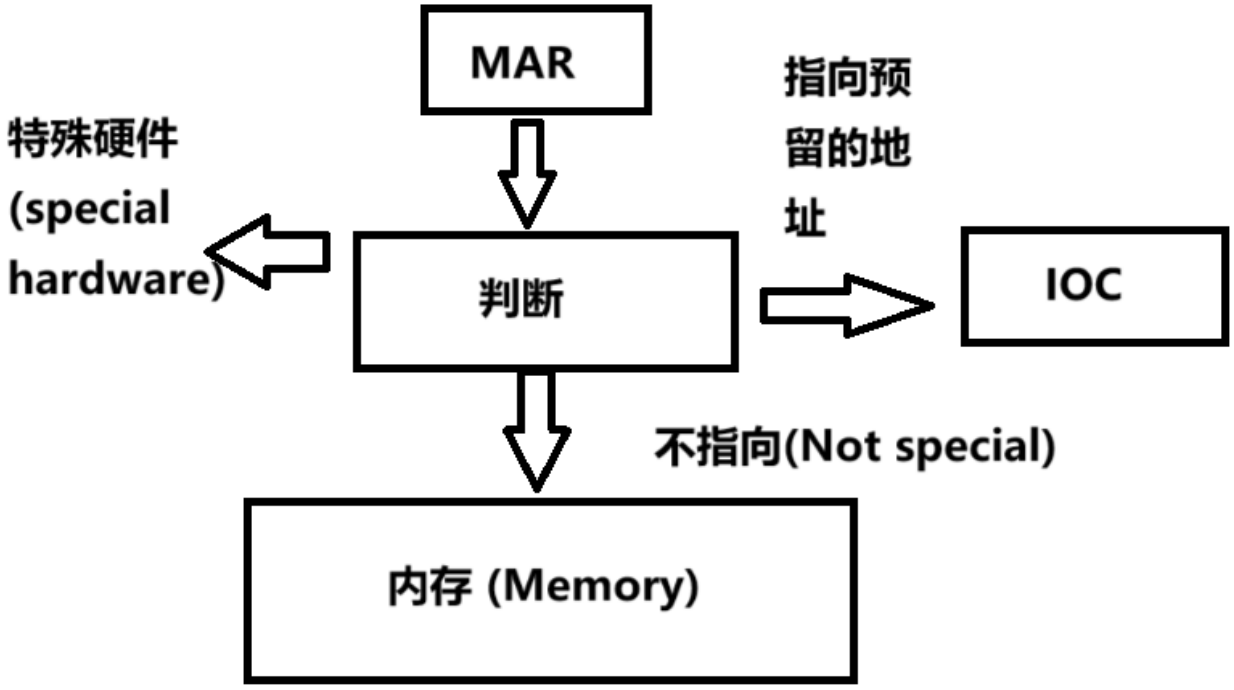
③. 执行实际操作(e.g. 输出像素带屏幕的字符流)



编址方式: 内存映射 VS 专用 I/O 指令

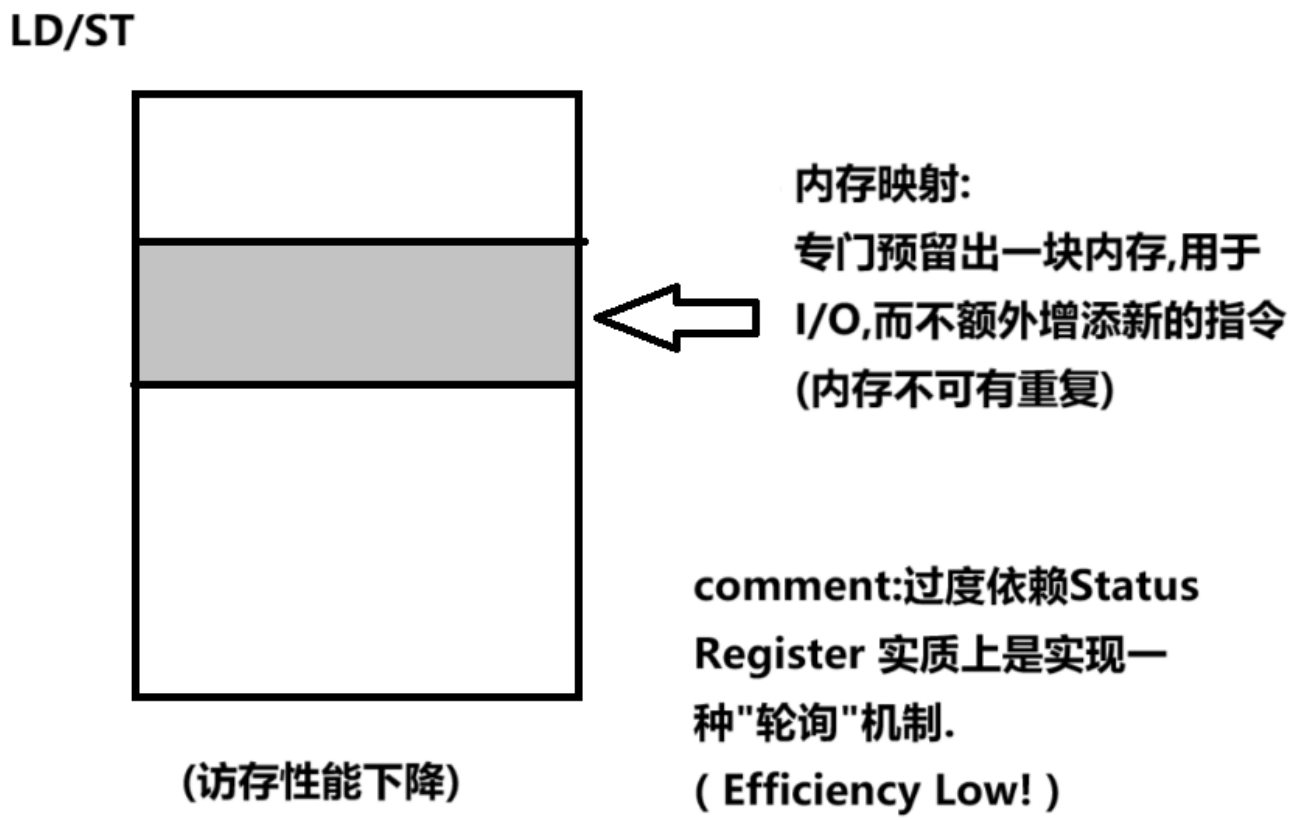
内存映射: e.g. ARM

专用I\O指令: e.g. X86 (IN / OUT) (专用IO指令是为访问外部设备而专门设计的指令)



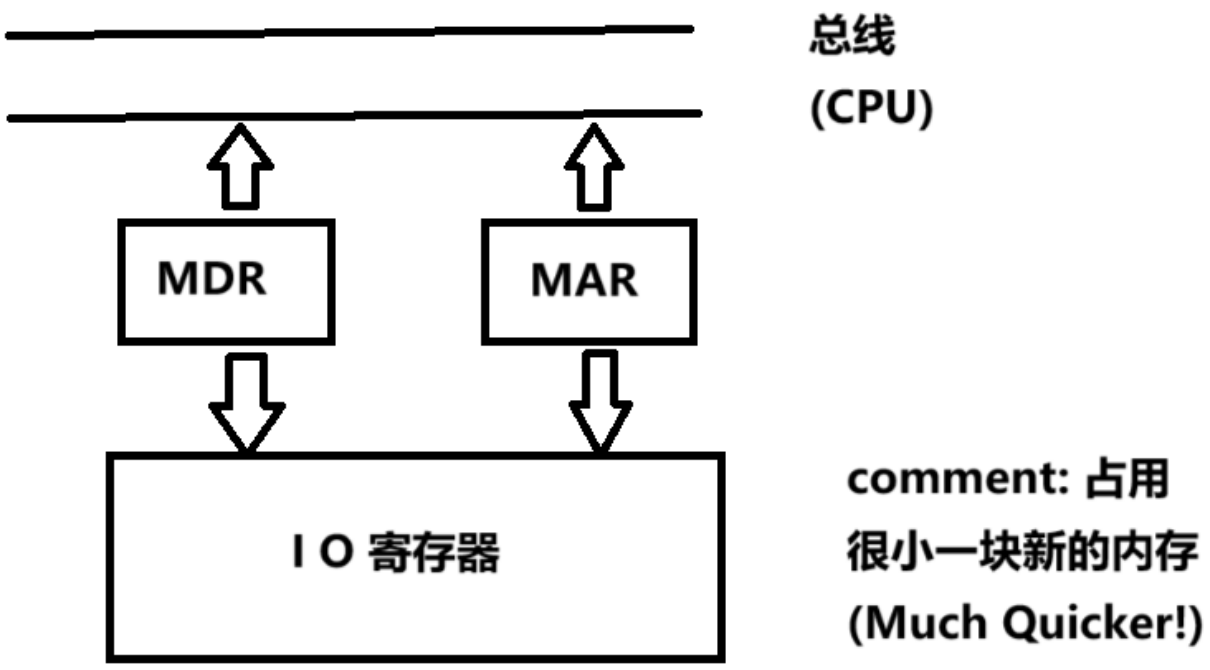
内存映射:

专门预留出一块内存,用于IO,而不额外增添新的指令(内存不可有重复)



专用 IN/OUT 指令:

仿照LD/ST设计新的专用IN/OUT指令.



同步与异步:

1. 同步: 数据以一种固定的,可预测的数据供给;CPU以某个固定的周期进行读/写操作.

2. 异步: 数据速率不可预测;CPU必须同步装置,以免遗失数据或者写入太快.

```
Method:
      设置状态寄存器(Status Register)或者标志位(signed bit),以满足/保证传输的正确性.
      Status Register:
            e.g.
            0   Wrong Message   --- 不读
            1   Correct Message --- 读
```

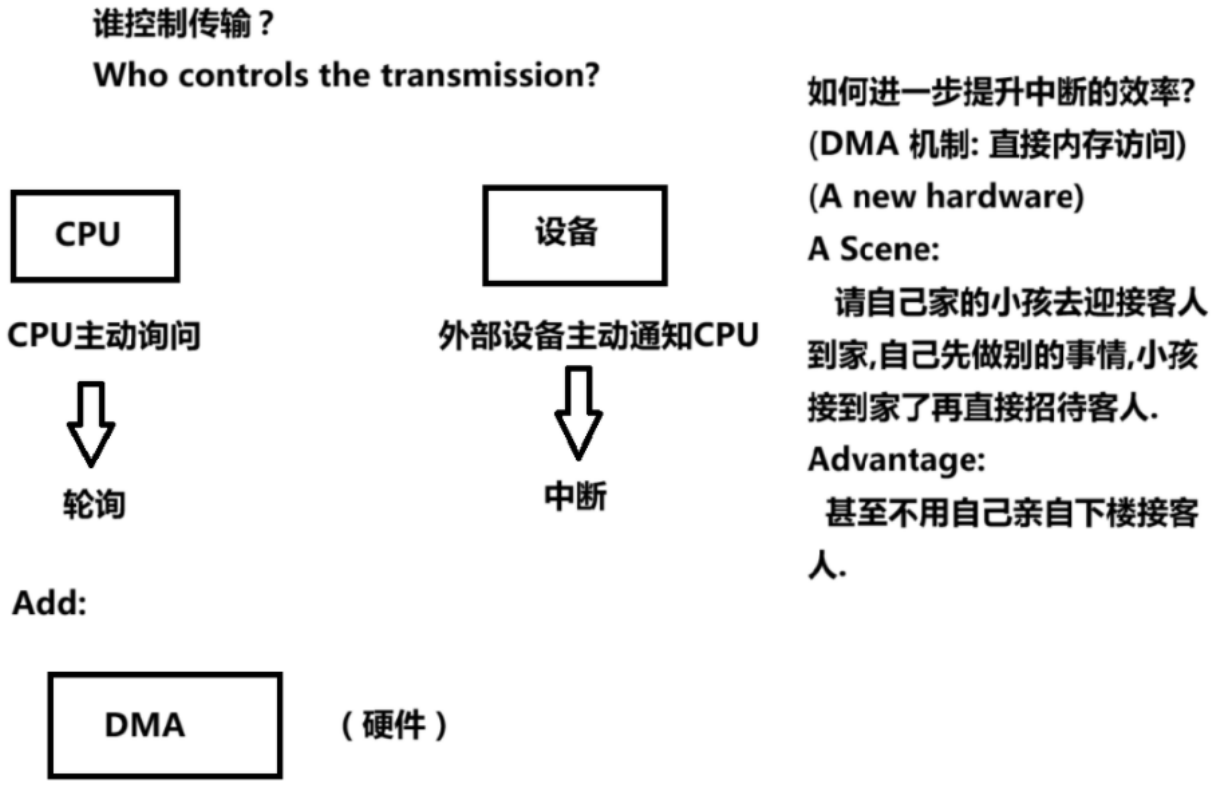
中断: 当新数据到达或者设备已经为下一个数据做好准备时,设备会发送一个特殊信号到CPU;CPU在此期间可以执行其他任务;而不是反复**轮询 (轮询机制-->中断机制)**

A Scene:

```
”客人到了没?客人到了没?客人到了没?“ ---> "当客人到达时请通知我"
```

CPU 和外设可并行工作,利用效率高,但需要专用中断硬件支持.

Difference: 谁控制传输?



an example for IO program

```
01  START   ST      R1,SaveR1   ; Save registers needed
02          ST      R2,SaveR2   ; by this routine
03          ST      R3,SaveR3
04  ;
05          LD      R2,Newline
06  L1      LDI     R3,DSR
07          BRzp    L1          ; Loop until monitor is ready
08          STI     R2,DDR      ; Move cursor to new clean line
09  ;
0A          LEA     R1,Prompt   ; Starting address of prompt string
0B  Loop    LDR     R0,R1,#0    ; Write the input prompt
0C          BRz     Input       ; End of prompt string
0D  L2      LDI     R3,DSR
0E          BRzp    L2          ; Loop until monitor is ready
0F          STI     R0,DDR      ; Write next prompt character
10          ADD     R1,R1,#1    ; Increment prompt pointer
11          BRnzp   Loop        ; Get next prompt character
12  ;
13  Input   LDI     R3,KBSR
14          BRzp    Input       ; Poll until a character is typed
15          LDI     R0,KBDR     ; Load input character into R0
16  L3      LDI     R3,DSR
17          BRzp    L3          ; Loop until monitor is ready
18          STI     R0,DDR      ; Echo input character
19  ;
1A  L4      LDI     R3,DSR
1B          BRzp    L4          ; Loop until monitor is ready
1C          STI     R2,DDR      ; Move cursor to new clean line
1D          LD      R1,SaveR1   ; Restore registers
1E          LD      R2,SaveR2   ; to original values
1F          LD      R3,SaveR3
20          BRnzp   NEXT_TASK   ; Do the program's next task
21  ;
22  SaveR1  .BKLW   1           ; Memory for registers saved
23  SaveR2  .BKLW   1
24  SaveR3  .BKLW   1
25  DSR     .FILL   xFE04
26  DDR     .FILL   xFE06
27  KBSR    .FILL   xFE00
28  KBDR    .FILL   xFE02
29  Newline .FILL   x000A       ; ASCII code for newline
2A  Prompt  .STRINGZ ''Input a character>''
```

LC-3中的IO:

KBSR的第15位为 ready bit:

当 ready bit 为 0 时: 无效

当 ready bit 为 1 时: 有效

使用BRn指令来判断最高位的 0(zp正) 或 1(n负)

KBDR:前半部分内存不使用，后半部分为ASCII数据:如果检测到 KBSR 中的 ready bit 为 1(有效),则直接读走ASCII的数据.

(CPU do this!)

LC-3 中设有四个固定的寄存器":

(设计时就已经约定好了)

KBSR.FILL xFE00 (Keyboard Status Register)

KBDR.FILL xFE02 (Keyboard Data Register)

DSR.FILL xFE04 (Display Status Register)

DDR.FILL xFE06 (Display Data Register)

code example

```
LD  R1 , KBSRptr  ;   R1 = xFE00
LDR R0 , R1 ,#0   ;
```

```
LDI R0 , KBSRptr  ;
```

两部分的代码是等价的

comment:在I/O中,常使用LDI指令

Template:

```
POLL    LDI  R0 , KBSRptr
        BRzp POLL        ; 轮询
; CPU 主动了解外部设备的状态,BRn~1;BRzp~非1
        LDI  R0 , KBDRptr; 若轮询到了,就载入数据
```

comment:

基于中断,外部设备可以:

(1).强制当前处理器执行的程序停止;

(2).使处理器响应设备的需求;

(3).完成后,处理器恢复停止的程序就像没发生过.

中断时随机的,JSR(子程序)是确定的

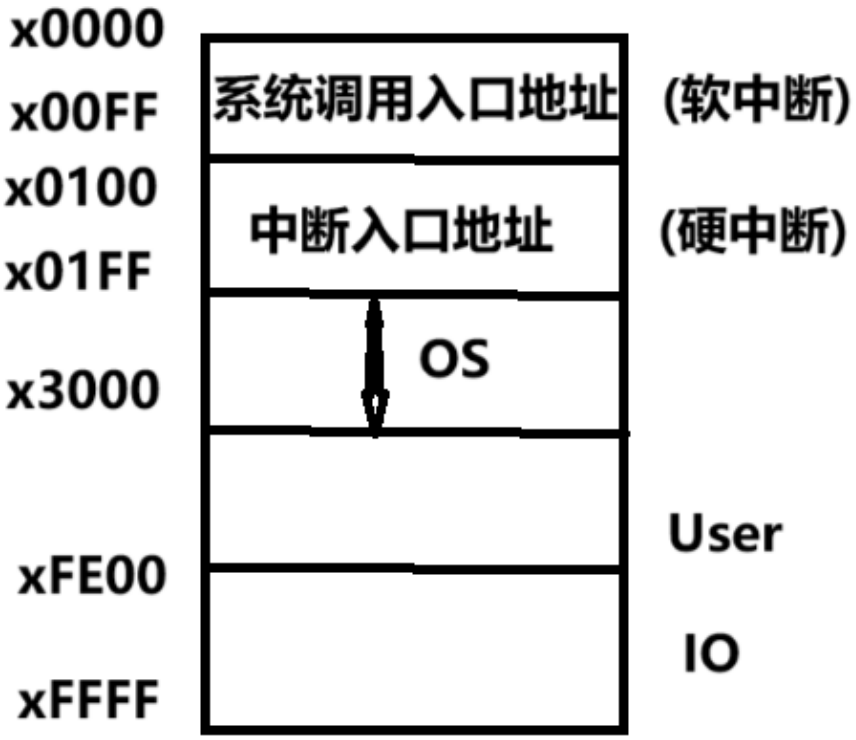
```
POLL1   LDI   R0,   KBSRptr
        BRzp    POLL1
        LDI   R0,   KBDRptr
POLL2   LDI   R1,   DSRptr
```

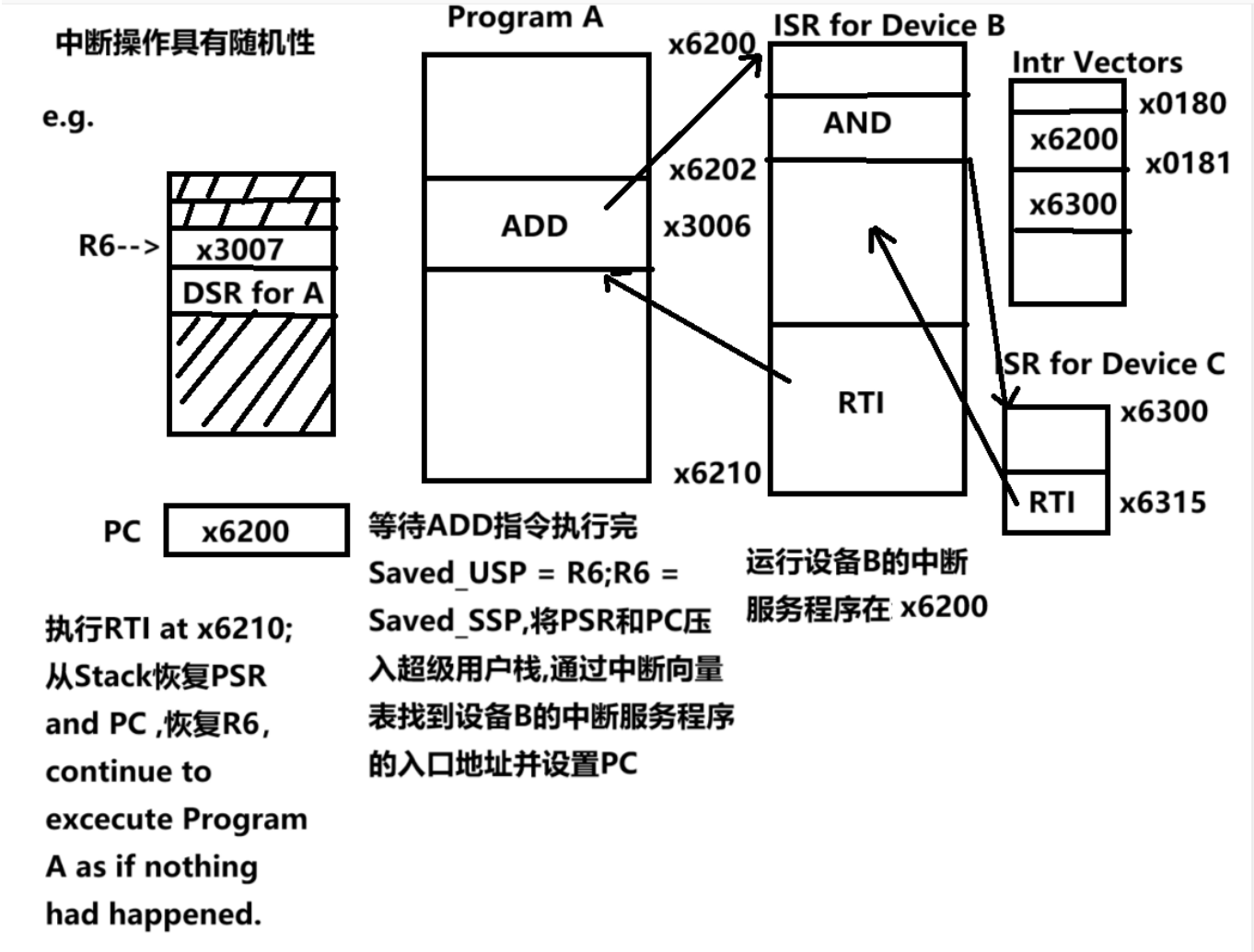
```
        BRzp    POLL2
        STI   R0,   DDRptr

KBSRptr.FILL    xFE00
KBDRptr.FILL    xFE02
DSRptr.FILL     xFE04
DDRptr.FILL     xFE06
```

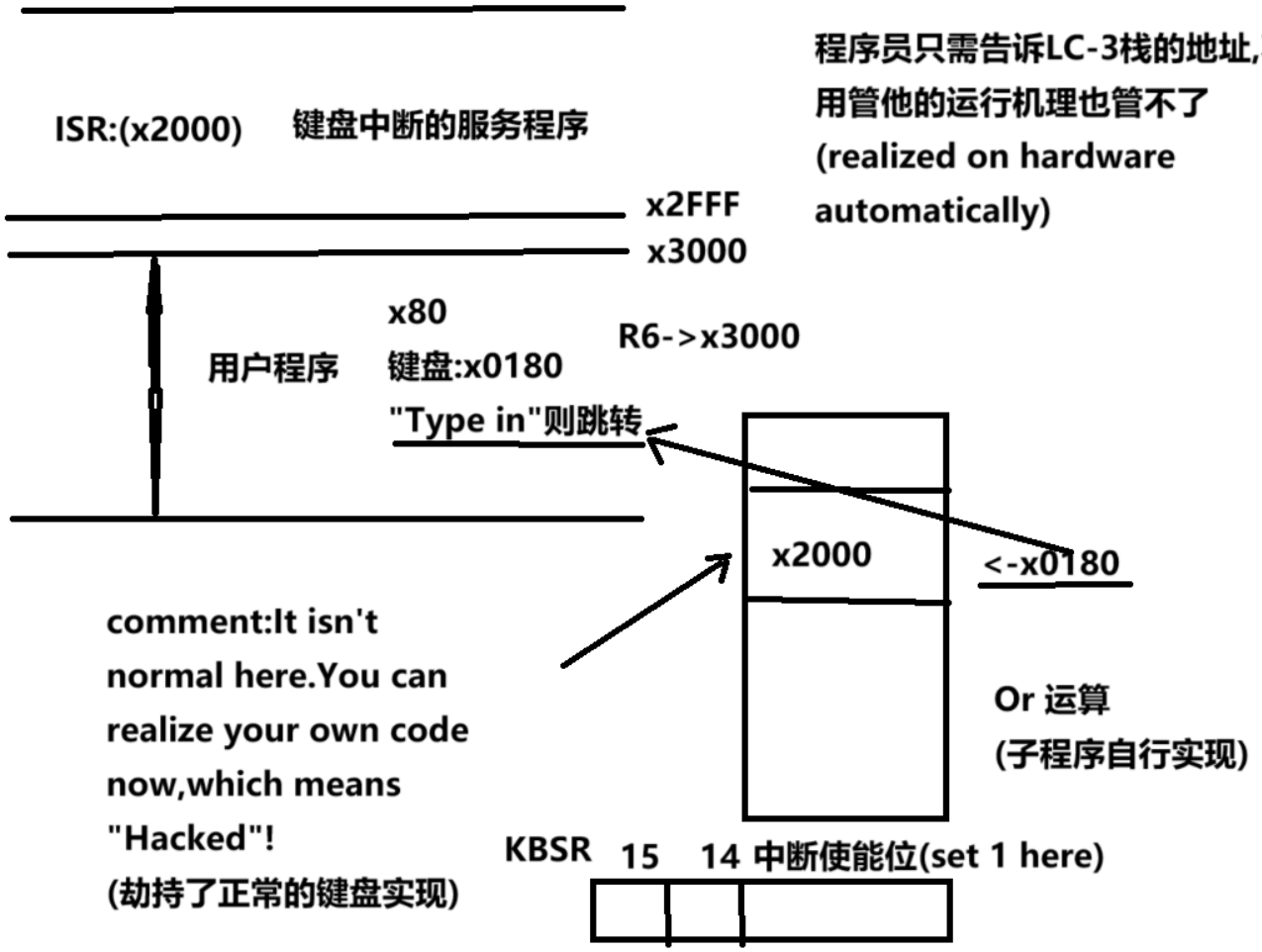
第15位:ready bit 1:就绪 0:不就绪

第14位:中断使能位 1:中断使能 0:不使用中断

Detailed explaination:


x0000
x00FF
系统调用入口地址
(软中断)
x0100
x01FF
中断入口地址
(硬中断)
x3000
OS
User
xFE00
IO
xFFFF


中断操作具有随机性
e.g.
R6-->
x3007
DSR for A
PC
x6200
Program A
x6200
ADD
x6202
x3006
x6210
ISR for Device B
AND
RTI
Intr Vectors
x0180
x6200
x0181
x6300
ISR for Device C
x6300
RTI
x6315
等待ADD指令执行完
Saved_USP = R6;R6 =
Saved_SSP,将PSR和PC压
入超级用户栈,通过中断向量
表找到设备B的中断服务程序
的入口地址并设置PC
运行设备B的中断
服务程序在 x6200
执行RTI at x6210;
从Stack恢复PSR
and PC ,恢复R6,
continue to
excecute Program
A as if nothing
had happened.


程序员只需告诉LC-3栈的地址,不用管他的运行机理也管不了
(realized on hardware automatically)
ISR:(x2000) 键盘中断的服务程序
x2FFF
x3000
x80
用户程序 键盘:x0180
R6->x3000
"Type in"则跳转
x2000
<-x0180
comment:It isn't normal here.You can realize your own code now,which means "Hacked"!
(劫持了正常的键盘实现)
Or 运算
(子程序自行实现)
KBSR 15 14 中断使能位(set 1 here)