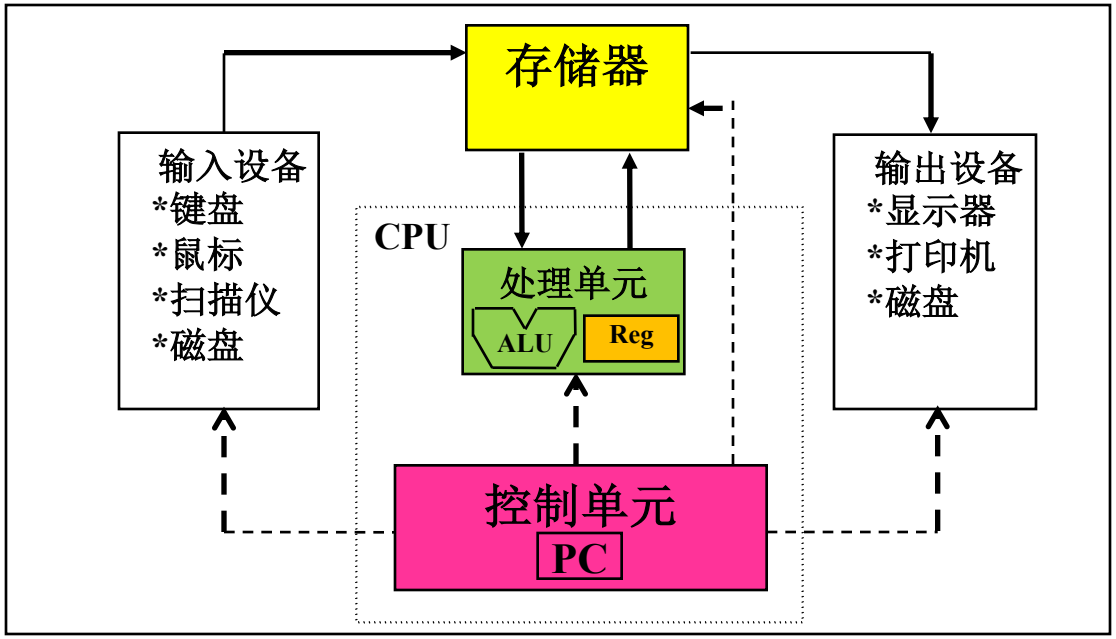
# 第十二章 输入和输出

# 输入和输出

- 可以通过执行TRAP指令来完成输入和输出（第9章）
- TRAP指令调用的是操作系统的I/O设备管理例程（第12、13章）

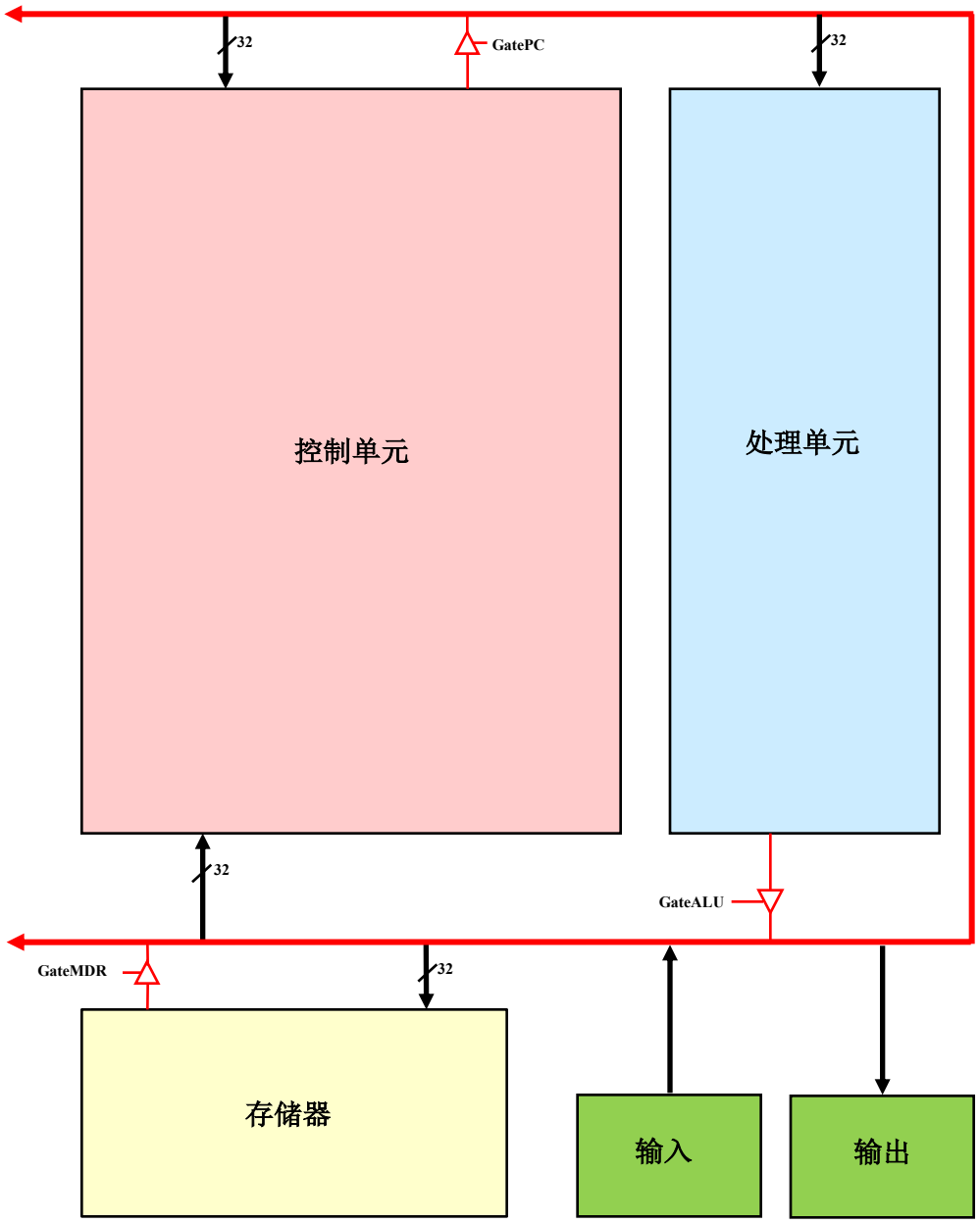
# 输入和输出

- 冯·诺依曼模型的重要组成部分



# 输入和输出

- 通过总线与CPU、存储器进行通信



# 键盘和显示器

- 字符设备
- 面向流的设备
  - 一个字符、一个字符的读写
  - 按照先后顺序

# I/O控制器

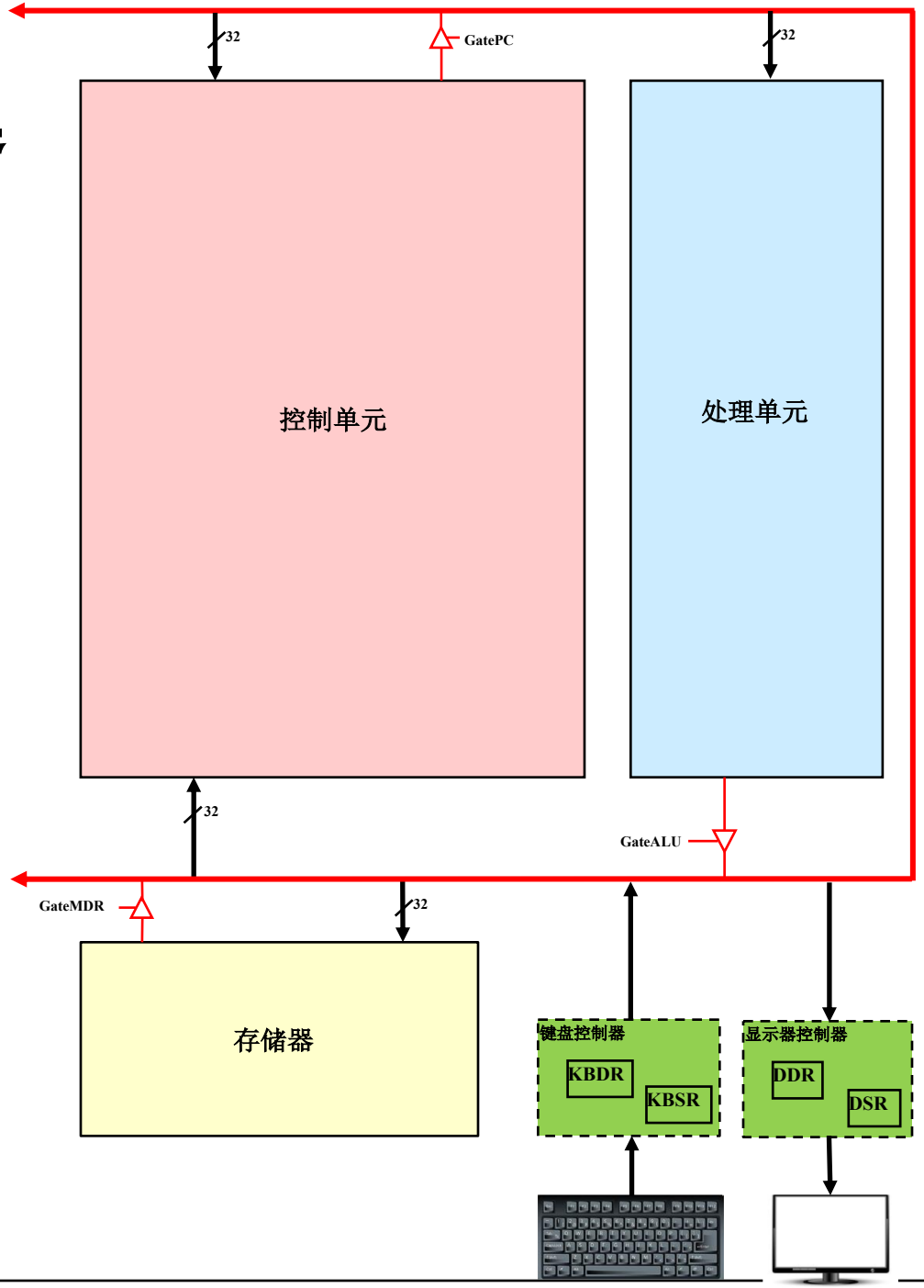
- 一种电子设备
- I/O设备与CPU通信的接口
- 以键盘控制器为例
  - 状态机
  - 解码器
  - 数据寄存器：保存数据
  - 状态寄存器：保存状态
  - ……

# I/O设备寄存器

- 最简单的I/O设备通常至少包含
  - 保存在计算机和设备之间进行传输的数据的寄存器
    - 键盘数据寄存器中存储的是用户输入的字符的ASCII码
  - 保存设备的状态信息的寄存器
    - 例如设备是处于可用的状态还是正忙于执行最近的I/O任务

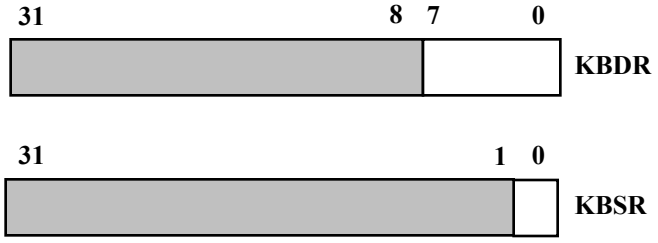
# DLX I/O控制器

- 键盘控制器
  - KBDR
  - KBSR
- 显示器控制器
  - DDR
  - DSR



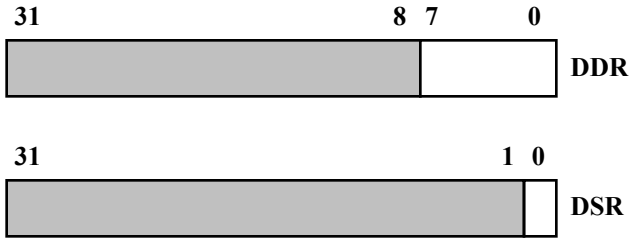
# 键盘设备寄存器

- 每一个寄存器32位（与DLX的通用寄存器一样），方便
- KBDR，[7:0]位用来存放数据，[31:8]位包含x000000
- KBSR，[0]位， 就绪位



# 显示器设备寄存器

- DDR，[7:0]位用于数据部分，[31:8]中包含x000000
- DSR，[0]位，就绪位



# 内存映射I/O

- 如何读取I/O设备寄存器中的数据？
- 如何向I/O设备寄存器加载数据？
- 两种机制
  - 专门的I/O指令
    - Intel x86指令集，in/out
      - 通用寄存器 ↔ I/O设备寄存器
  - 数据传送指令
    - 通用寄存器 ↔ 存储器
    - 内存映射I/O

# 内存映射I/O

- 问题：如何表示I/O设备寄存器？
- 采用内存映射的方式
  - 每一个I/O设备寄存器都被分配一个存储器地址空间中的地址
  - 这些地址被分配给I/O设备寄存器，而不再是存储单元

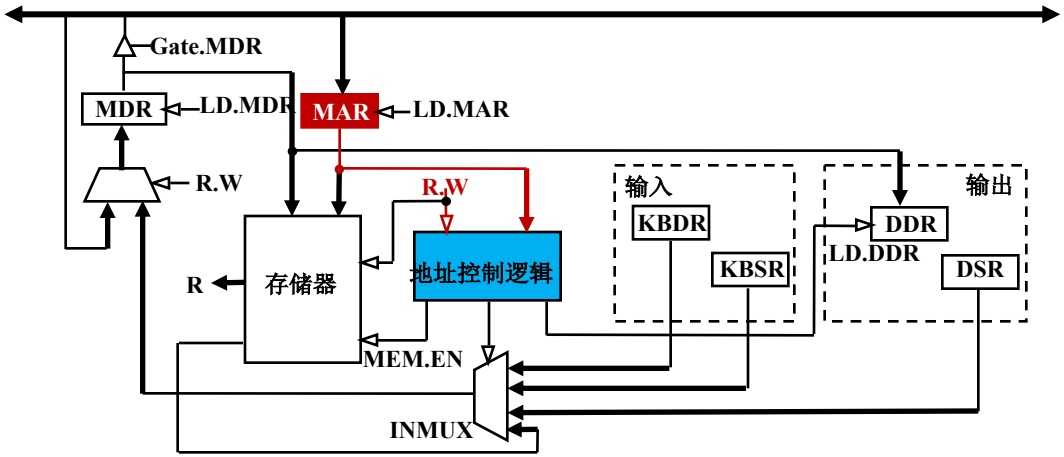
# DLX-内存映射的I/O

- 地址xFFFF 0000到xFFFF 00FF被分配给I/O设备寄存器，其他地址分配给存储单元
  - 每一个寄存器：32位（与通用寄存器一样）

| 地址 | I/O寄存器 |
| --- | --- |
| xFFFF 0000~ xFFFF 0003 | 键盘状态寄存器（KBSR） |
| xFFFF 0004~ xFFFF 0007 | 键盘数据寄存器（KBDR） |
| xFFFF 0008~ xFFFF 000B | 显示器状态寄存器（DSR） |
| xFFFF 000C ~ xFFFF 000F | 显示器数据寄存器（DDR） |
| xFFFF 00F8~ xFFFF 00FB | 机器控制寄存器（MCR） |

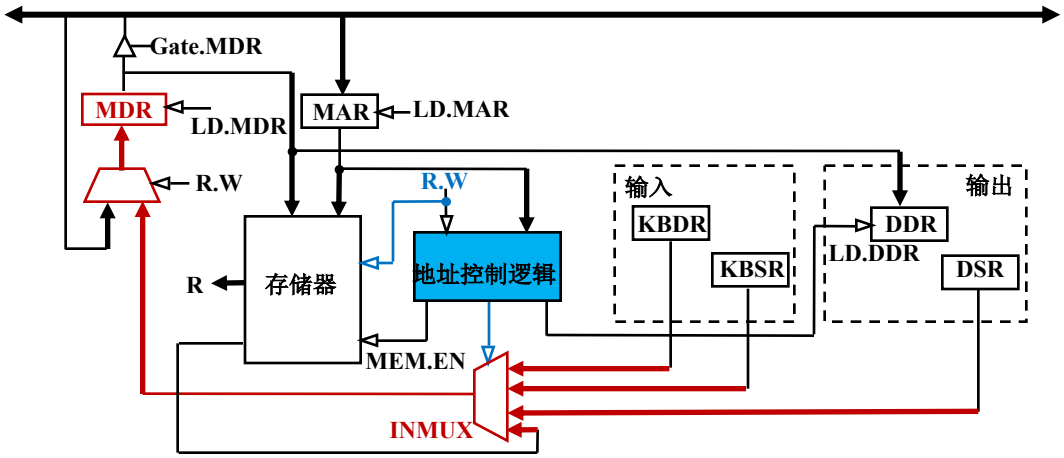
# 内存映射I/O的DLX数据通路

- **地址控制逻辑**
  - 控制输入/输出操作
  - 输入：**MAR，R.W**



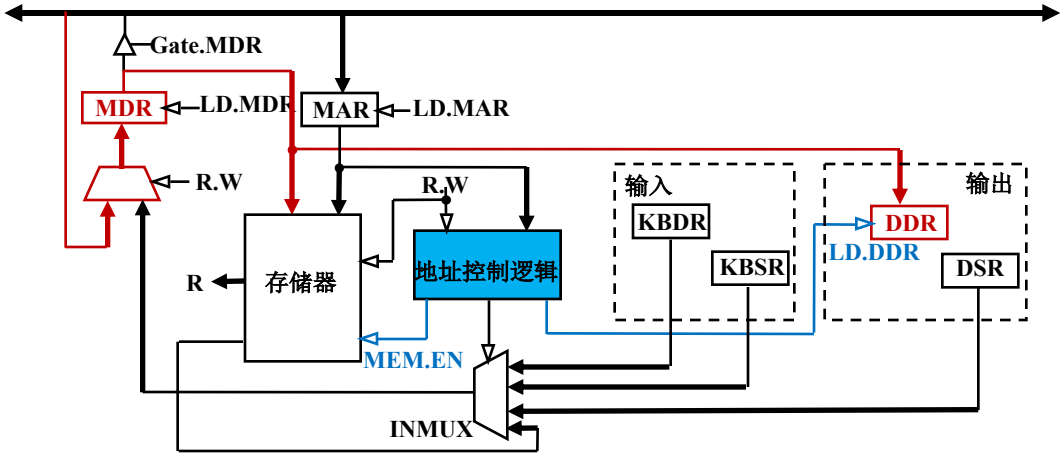
# R.W=0

- 加载 LOAD
  - INMUX：选择存储器/DSR/KBSR/KBDR→MDR



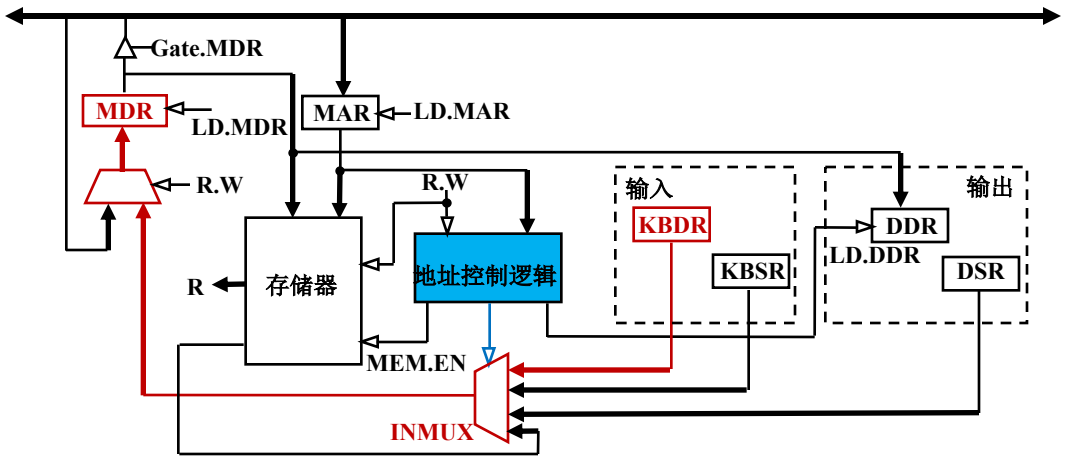
# R.W=1

- 存储 STORE
  - MDR → 存储器/DDR
  - 控制信号：MEM.EN，LD.DDR



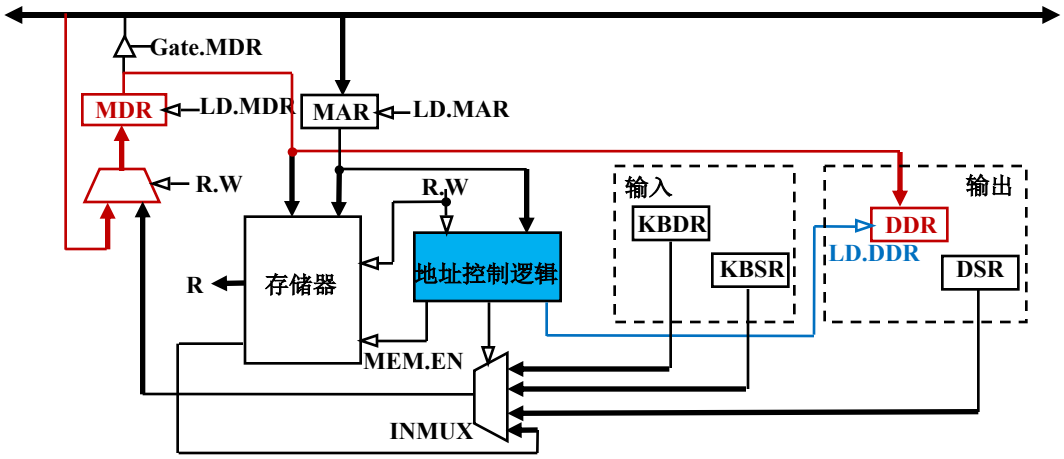
# 读KBDR的指令序列

```
KBDR: .word   xFFFF0004      ; KBDR的起始地址
……
        lw      r1, KBDR(r0)   ; R1= xFFFF 0004
        lw      r4, 0(r1)      ;将KBDR中的数据加载到R4中
```



# 写DDR的指令序列

```
DDR:    .word   xFFFF000C       ; DDR的起始地址
……
        lw      r1, DDR(r0)     ; R1= xFFFF 000C
        sw      0(r1) , r4      ;将R4中的数据写到DDR中
```



# 键盘与处理器

- 用户输入字符，KBDR←ASCII码

lw　　r4, 0(r1)

- 问题：如果执行这条指令时，用户还没有输入新的字符，会发生什么情况？
  - 将KBDR之前存储的数据读入R4中？

# 异步

- I/O的执行与处理器的执行不同步
  - 微处理器：时钟控制下执行指令
  - 用户键盘输入：不受时钟控制

# 显示器？

sw　0(r1) , r4

- 问题：如果执行该指令时，显示器还没有将上一个DDR中的字符显示完成，会发生什么情况？
  - 将DDR之前存储的数据覆盖？

# 异步

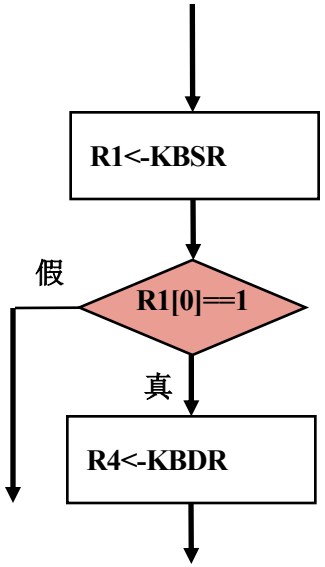
- 处理异步问题
  - 协议/握手机制
  - 键盘，1位的标志
    - 是否输入一个字符
  - 显示器，1位的标志
    - 被送给显示器的字符是否已被显示
  - 设备状态寄存器 [0]位
    - KBSR[0]
    - DSR[0]

# 键盘就绪位

- KBSR[0]
  - 每当用户输入一个字符时，键盘控制器就将就绪位设为1
    - 键盘不能用
  - 每当处理器读取该字符时，键盘控制器就将就绪位清空
    - 允许输入下一个字符

# 同步机制

- 读取KBDR之前，检查就绪位



# 键盘：同步的指令序列

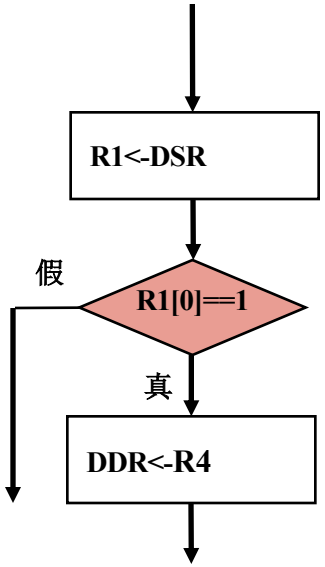
```
……
01 KBSR :    .word   xFFFF0000     ; KBSR的起始地址
02 KBDR :    .word   xFFFF0004     ; KBDR的起始地址
……
03           lw      r1, KBSR(r0)
04           lw      r2, 0(r1)     ; 测试是否有字符被输入
05           andi    r3, r2, #1
06           beqz    r3, XXX
07           lw      r1, KBDR(r0)
08           lw      r4, 0(r1)
09           j       NEXT_TASK     ; 执行下一个任务
```

# 显示器就绪位

- DSR[0]
  - 每当显示器完成了一个字符的显示，显示器控制器就将DSR[0]设为1
  - 每当处理器向DDR写字符时，显示器控制器就将DSR[0]清空

# 同步机制

- 写DDR之前，检查就绪位



# 显示器的同步指令序列

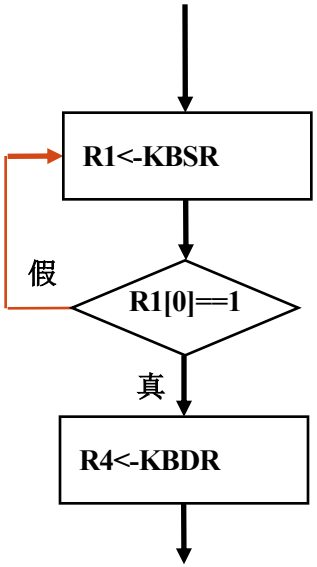
```
……
01 DSR :      .word   xFFFF0008      ; DSR的起始地址
02 DDR :      .word   xFFFF000C      ; DDR的起始地址
……
03            lw      r1, DSR(r0)
04            lw      r2, 0(r1)      ; 测试输出寄存器是否就绪
05            andi    r3, r2, #1
06            beqz    r3, XXX
07            lw      r1, DDR(r0)
08            sw      0(r1), r4
09            j       NEXT_TASK      ; 执行下一个任务
```

# 打印机

- 数据寄存器
  - 存储要打印的字符
- 状态寄存器
  - Done Bit：表示上一个字符是否打印完成
  - Error Bit：表示打印机出错，如卡纸或缺纸

# 轮询

- **周期性的检查**状态位，判断是否执行I/O操作的方法
  - **最简单的方式**
  - 由处理器完全控制和执行通信工作



# 输入服务例程-轮询

```
……
01 KBSR :      .word   xFFFF0000      ; KBSR的起始地址
02 KBDR :      .word   xFFFF0004      ; KBDR的起始地址
……
03             lw      r1, KBSR(r0)
04 INPOLL:     lw      r2, 0(r1)      ; 测试是否有字符被输入
05             andi    r3, r2, #1
06             beqz    r3, INPOLL
07             lw      r1, KBDR(r0)
08             lw      r4, 0(r1)
09             j       NEXT_TASKX     ; 执行下一个任务
```

# 输出服务例程-轮询

```
……
01 DSR :          .word   xFFFF0008       ; DSR的起始地址
02 DDR :          .word   xFFFF000C       ; DDR的起始地址
……
03                lw      r1, DSR(r0)
04 OUTPOLL:       lw      r2, 0(r1)       ; 测试输出寄存器是否就绪
05                andi    r3, r2, #1
06                beqz    r3, OUTPOLL
07                lw      r1, DDR(r0)
08                sw      0(r1), r4
09                j       NEXT_TASKY      ; 执行下一个任务
```

# 键盘输入回显

- 可以通过简单的组合前面讨论过的两个例程得到，不需要添加任何复杂的电路

```
……
01 KBSR:        .word    xFFFF0000        ; KBSR的起始地址
02 KBDR:        .word    xFFFF0004        ; KBDR的起始地址
03 DSR:  .word  xFFFF0008          ; DSR的起始地址
04 DDR:  .word  xFFFF000C          ; DDR的起始地址

……
05        lw       r1, KBSR(r0)
06 INPOLL:        lw       r2,0(r1)         ; 测试是否有字符被输入
07        andi     r3,r2,#1
08        beqz     r3, INPOLL
```

```
09          lw      r1, KBDR(r0)
0A          lw      r4,0(r1)
0B          lw      r1, DSR (r0)
0C    ECHO: lw      r2,0(r1)        ; 测试输出寄存器是否就绪
0D          andi    r3,r2,#1
0E          beqz    r3,ECHO
0F          lw      r1, DDR(r0)
10          sw      0(r1),r4
11          j       NEXT_TASK       ; 执行下一个任务
```

# 键盘输入——提示符

- 为了让用户知道该例程正在等待键盘的输入，应该在显示器上输出一些信息——提示符

# 提示符（数据区）

```
01      ……
02      KBSR:       .word    xFFFF0000      ; KBSR的起始地址
03      KBDR:       .word    xFFFF0004      ; KBDR的起始地址
04      DSR:        .word    xFFFF0008      ; DSR的起始地址
05      DDR:        .word    xFFFF000C      ; DDR的起始地址
06      ……
07      Newline:    .byte    x0A            ; 新行的ASCII码
08      Prompt:     .asciiz  " Input  a  character>"
09      ……
```

# 输出新行（代码区）

```
0A          lb      r2, Newline(r0)
0B          lw      r5, DSR(r0)  ; 测试输出寄存器是否就绪
0C    L1:   lw      r3, 0(r5)
0D          andi    r3, r3, #1
0E          beqz    r3, L1       ; 循环直到显示器就绪
0F          lw      r5, DDR(r0)
10          sw      0(r5), r2    ; 光标移到新的一行
11    ;
```

# 输出提示符

```
12          addi    r1, r0, Prompt   ; 提示符字符串的起始地址
13    LOOP: lb      r2, 0(r1)        ; 获取提示符字符
14          beqz    r2, Input        ; 提示符字符串结束？
15          lw      r5, DSR(r0)
16    L2:   lw      r3, 0(r5)
17          andi    r3, r3, #1
18          beqz    r3, L2           ; 循环直到显示器就绪
19          lw      r5, DDR(r0)
1A          sw      0(r5), r2        ; 输出下一个提示符字符
1B          addi    r1, r1, #1       ; 提示符的指针加1
1C          j       LOOP             ; 下一个提示符字符
1D    ;
```

# 输入回显

```
1E      Input:  lw      r5, KBSR(r0)
1F      L3:     lw      r3, 0(r5)
20              andi    r3, r3, #1
21              beqz    r3, L3          ; 轮询直到一个字符被键入
22              lw      r5, KBDR(r0)
23              lw      r4, 0(r5)       ; 将输入的字符加载到R4
24              lw      r5, DSR(r0)
25      L4:     lw      r3, 0(r5)
26              andi    r3, r3, #1
27              beqz    r3, L4          ; 循环直到显示器就绪
28              lw      r5, DDR(r0)
29              sw      0(r5), r4       ; 将输入的字符回显
2A      ;
```

# 轮询缺点

- 读取键盘数据时，状态未就绪，假设
  - 微处理器的时钟频率：300MHz
    - 一个时钟周期：3.3纳秒
  - 处理执行一条指令平均需要10个时钟周期
    - 执行一条指令33纳秒
  - 用户在1秒钟之后输入一个字符
  - 重复指令序列10条指令
    - 处理器执行指令序列300万次，才能读取到该字符
- 浪费了大量的处理器时间

# 中断

- 让处理器一直做它自己的事直到被从键盘发来的信号打断：“已经输入了一个新字符，其ASCII码位于输入设备寄存器里，你需要读取它”——中断驱动的I/O
- 由I/O设备控制器来控制交互

# 习题

- 12.6
- 12.7
- 12.8

  订正：ANDI　　R5, R4, #1

  　　BEQZ　　R5, START