Lec01 Input/Output of LC-3

复习LC-3架构

冯-诺伊曼模型

五个主要组成部分

内存、输入、输出、处理单元、控制单元

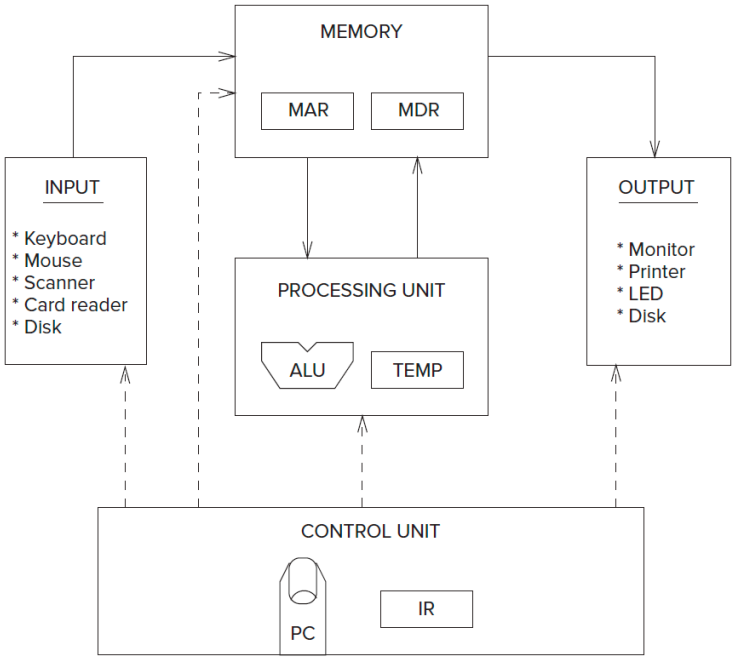

Figure 4.1 The von Neumann model, overall block diagram.

指令集架构（Instruction Set Architecture(ISA)）

- 八个寄存器：$R0, R1, \ldots, R7$
- 数据类型：16位二进制数
- 内存地址：x0000~xFFFF
- 寻址方式：
  - 立即数：ADD R1,R4,#-2
  - 寄存器寻址：NOT R3,R5
  - PC相对寻址：LD,ST
  - 基址+偏移：LDR,STR
  - 间接寻址：LDI,STI
  - 加载有效地址：LEA，地址是通过向递增的程序计数器添加一个偏移量来计算的。地址本身（而不是它的值）存储在寄存器中。

练习

```
.ORIG x3000
  LD R1, LABEL
  LDI R2, LABEL
  LDR R3, R2, #1
  LEA R4, LABEL
  LABEL .FILL x4001
.END

Address Data
; x4001 x6001
; ……
; x6001 x7001
; x6002 x7002
```

答案

```
R1 x4001 ;The value of "LABEL" itself.
R2,x6001 ;What stored in the address x4001
R3,x7002 ;Add 1 to x6001,and R3 is the value stored in the address x6002
R4,x3004 ;Where did we define LABEL? In x3004
```

I/O的内存映射

- 为每个设备寄存器分配一个内存地址
  - I/O设备寄存器被映射到一组分配给I/O设备寄存器的地址，而不是实际的内存位置
- 使用数据移动指令(LD/ST)来控制和转移数据
- **LC-3设备寄存器：**
  - KBDR(Keyboard Data Register)：用于存储从键盘输入的字符的ASCII值（A对应于`0x41`）
    - CPU可以从KBDR中读取数字，以获取用户输入的字符。
    - 但是读取之前，CPU需要确认有新的按键输入，避免读取无效的或旧的数据。这需要通过检查**KBSR**（键盘状态寄存器）来实现。
  - KBSR(Keyboard Status Register)：用于指示是否有新的键盘输入，以及KBDR中的数据是否准备好被读取。
    - KBSR的最高位（第15位）是一个状态标志：
      - 当用户按下一个键时，第15位被置为1，表示KBDR中有新的字符数据。
      - 当CPU从KBDR读取数据有，第15位会自动被清零，表示当前没有新的输入。
    - 这个标志就像一个“就绪信号”，告诉CPU是否可以安全地读取KBDR。
    - 在读取KBDR之前，CPU必须先检查KBSR的第15位：
      - 如果第 15 位是 1，表示有新数据，可以读取 KBDR。
      - 如果第 15 位是 0，表示没有新输入，CPU 不应读取 KBDR（否则可能得到无意义的数据）。
  - DDR(Display Data Register)：用于将ASCII码对应的字符显示到屏幕上。
    - CPU通过向DDR写入数据来输出字符到显示设备。

        - 但写入之前，CPU需要确保显示器已经准备好接受新字符，这需要检查**DSR**（显示状态寄存器）
    - DSR(Display Status Register)：用于指示显示器是否准备好接受新的字符进行显示。
        - DSR 的最高位（第 15 位）是一个状态标志：
            - 当显示器准备好接受新字符时，第 15 位被置为 1。
            - 当 CPU 向 DDR 写入数据后，第 15 位会被置为 0，直到显示器再次准备好。
        - 这个标志就像一个“就绪信号”，告诉 CPU 是否可以向 DDR 写入数据。
        - 在向 DDR 写入数据之前，CPU 必须先检查 DSR 的第 15 位：
            - 如果第 15 位是 1，表示显示器就绪，可以向 DDR 写入字符。
            - 如果第 15 位是 0，表示显示器忙碌，CPU 需要等待（通常通过循环检查 DSR）。
- **输入流程（键盘）**
    1. 用户按键，KBSR 的第15位置 1，KBDR 存储字符的 ASCII 值。
    2. CPU 检查 KBSR，确认第15位为 1。
    3. CPU 从 KBDR 读取字符，第15位自动清零。
- **输出流程（显示器）**
    1. CPU 检查 DSR，确认第 15 位为 1（显示器就绪）。
    2. CPU 向 DDR 写入字符的 ASCII 值，字符显示在屏幕上。
    3. DSR 的第 15 位清零，直到显示器再次就绪。
- 在 LC-3 架构中，这四个寄存器是内存映射的，意味着它们有固定的内存地址：
    - **KBSR**: 通常在 0xFE00
    - **KBDR**: 通常在 0xFE02
    - **DSR**: 通常在 0xFE04
    - **DDR**: 通常在 0xFE06
- CPU通过加载(`LDI`)和存储指令(`STI`)访问这些地址。
    - 从键盘读取字符：

```
LOOP    LDI R0, KBSR_ADDR    ; 检查 KBSR
        BRzp LOOP        ; 如果第 15 位为 0，继续等待
        LDI R0, KBDR_ADDR    ; 读取 KBDR 中的字符
```

    - 向显示器输出字符：

```
WAIT    LDI R1, DSR_ADDR     ; 检查 DSR
        BRzp WAIT        ; 如果第 15 位为 0，继续等待
        STI R0, DDR_ADDR     ; 将 R0 中的字符写入 DDR
```

    - 汇总：

```
.ORIG x3000
```

```
KPOLL LDI R0, KBSR     ; 循环检测键盘输入状态
      BRzp KPOLL       ; 如果KBSR[15]=0（无新字符），继续等待
      LDI R0, KBDR     ; 读取键盘数据寄存器中的字符
; 以上是TRAP x20 = GETC的工作原理
DPOLL
      LDI R1, DSR.     ; 循环检测显示器状态
      BRzp DPOLL.      ; 如果DSR[15]=0（显示器忙碌），继续等待
      STI R0, DDR      ; 将字符写入显示数据寄存器
; 以上是TRAP x21 = OUT的工作原理
      HALT
KBSR .FILL xFE00       ; KBSR地址
KBDR .FILL xFE02       ; KBDR地址
DSR  .FILL xFE04       ; DSR地址
DDR  .FILL xFE06       ; DDR地址
.END
```

补充：Polling I/O 和 Interrupt-Driven I/O

中断驱动I/O

1. **核心思想**：中断驱动I/O的核心目标是**让CPU在等待I/O设备就绪时不必忙等（busy-wait）**，而是允许CPU去执行其他任务，直到设备主动通知CPU“数据已准备好”或“需要处理”。
   这种机制通过**硬件中断信号（Interrupt Signal）**触发，CPU收到中断后，会暂停当前程序，跳转到预设的中断处理程序（ISR, Interrupt Service Routine）处理I/O，完成后恢复原程序执行。
2. **工作流程**：
   1. **设备准备就绪**：当I/O设备（如键盘、显示器）准备好数据或需要CPU处理时，设备会通过硬件电路向CPU发送一个**中断请求信号（Interrupt Request, IRQ）**
      - 例如：用户按下键盘时，键盘控制器会将按键的ASCII码存入 `KBDR`（键盘数据寄存器），并设置 `KBSR`（键盘状态寄存器）的最高位为 `1`，表示“数据就绪”。
   2. **CPU响应中断**：CPU在每个指令周期的末尾检查是否有中断请求。如果满足中断条件，则：
      1. 完成当前指令的执行。
      2. 保存当前程序状态（PC和PSW，即程序计数器和处理器状态字）
         - 在LC-3中，PC和PSW会被压入系统栈(Supevisor Stack)
      3. 关闭中断(CLI)，防止嵌套中断。
      4. 跳转到中断服务例程（ISR）：
         - 根据设备的中断向量号（Interrupt Vector），从**中断向量表（Interrupt Vector Table, IVT）** 中读取ISR的入口地址。
         - 在LC-3中，中断向量表存储在内存地址 `x0100` 到 `x01FF` 之间，每个中断向量占1个字（16位）。
   3. **执行中断服务例程（ISR）**
      - ISR是程序员编写的处理I/O的代码，按下不表。
      - 关键操作：
        - 从设备寄存器（如 `KBDR` 或 `DDR`）读取数据，或向设备写入数据。

        - 清除设备的中断标志（如将 `KBSR` 最高位置 `0`）。
        - 可能需要保存/恢复寄存器（LC-3的ISR中需手动保存使用的寄存器）。
4. **返回原程序**
    - 执行 `RET` 指令后：
        - 从系统栈中恢复PC和PSW。
        - 重新打开中断（如果之前关闭）。
        - CPU继续执行被中断的程序，就像什么都没发生过一样。

中断驱动I/O vs. 轮询I/O

	Polling	**Interrupt-Driven**
CPU利用率	忙等期间CPU被完全占用，无法执行其他任务。	CPU仅在中断发生时处理I/O，其余之间可自由工作
响应延迟	取决于轮询频率(高频率浪费CPU，低频率延迟高)	设备就绪后立即响应
实现复杂度	简单（只需循环检查状态位）	较复杂（需设置中断向量表、编写ISR）
适用场景	高频设备或实时性要求极高的场景	大多数慢速设备（键盘、鼠标）