首先要写过程，同时注意课后习题重新做，指令重新背一遍

第一章-》

输入输出设备，运算器，控制器，储存器

计算机求解问题：

```
1. 描述问题
2. 分析问题 -- 算法
3. 语言实现算法
4. 机器结构 -- 指令集（二进制机器代码）
5. 微体系结构
6. 逻辑电路
7. 元器件
```

思想：通用，分层，软硬件结合

第二章-》Bit 数据类型及其运算

- 计算机里面数值就是补码形式
- 逻辑运算，算术运算
- IEEE754 浮点数定义以及转换
-

第三章-》数字逻辑

- CMOS（MOS管相当于开关，由N管和P管组成，CMOS只能组成非，与非和或非（记住其电路图））
- 组合逻辑电路（书上讲的如：译码器，复用器，全加器（工作原理，输入输出，真值表），分析和设计该电路，**先画出真值表**，对于可编程逻辑阵列n个输入则有power(n,2)个and门，真值表中的输出个数决定多少个or门）
- 时序电路（存储单元怎么工作的（如is锁存器什么时候置1，置0，静止，门控D锁存器，读和写，**状态转换图和真值表**）

第四章-》冯诺依曼体系

- 五大部件
- 指令周期（通常6个步骤或小于6步（取指令，译码，地址计算，取操作数，执行，回写）**地址计算是指内存的地址计算而不是寄存器**
- 停机（MCR寄存器第15位清零对应时钟的清零）
- 琐碎知识：
    - 字长：运算器支持的操作数的最大宽度/寄存器的宽度
    - 每个输入输出设备都有自己的访问接口，一般是一组专用寄存器类似MDR（16位）和MAR16（位）
    -
        - lc3中有KBDR，KBSR，DDR，DSR
        - 控制设备访问的程序称为设备驱动程序
        - 控制器（控制指令的执行过程）
        - PC，IR寄存器
    - LC3五大部件：

        - 内存（存储单元以及16位的MDR和MAR，容量为2^16*16）
        - 运算单元，包括ALU和8个16位寄存器
        - 控制单元：PC，IR寄存器
        - 输入输出单元：键盘和显示器
        - 部件的连接：总线（同一时间只允许同一个主设备使用）
    - 指令由操作数和操作码组成
    - 一个计算机系统所有指令和格式称为指令集（ISA）
    - 控制单元负责具体解释每条指令并产生控制信号协调其它部件来完成指令执行
    - LC3指令长度为16位
    - Fetch
        - 把PC中存放的内容拷贝到 内存的MAR寄存器中.
        - 给内存发读信号。
        - 拷贝MDR的内容到IR寄存器中。
        - PC=PC+1,PC指向下一条待执行的指令
    - Decode
        - 利用4-16的译码器译码（因为指令前4位表示操作码（最多16种可能）
    - 地址计算
        - 间接寻址（基址（放在寄存器中）+偏移）LDR
        - 相对寻址（PC+偏移）
        - 立即地址（偏移+0）
    - **注意在fetch阶段的最后PC=PC+1**
    - 指令和数据一样，都是放在内存中，取指令要用load指令取
    - **注意JMP指令的目标地址由寄存器给出**
    - 如果系统时钟不停，控制单元会重复不断的执行指令（系统指令或程序指令）停止时钟以停止状态机运行（S=1，R=0）低有效

第四章-》第七章

第五章（**LC3**）

- ISA是软件和硬件的分界面
- 什么是ISA：
    - ISA（指令集结构），一个计算机系统所有的指令和格式称为指令集。它向程序员提供有关控制机器所有的必要信息。包括内存组织方式（寻址空间，寻址能力），寄存器组（多少个，长度），指令集（操作码，数据类型，寻址模式）等信息。同时其实软件和硬件的分界面
- 数据类型（16位定点补码整数）
- 寻址方式：
    - 非内存寻址：
        - 直接寻址（操作数在指令中）
        - 寄存器寻址（操作数在寄存器中）
    - 内存寻址
        - pc-相对寻址
        - 间接寻址
        - 基址+偏移
- LC3的指令只有ADD，AND，NOT，**源操作数和目的操作数必须均为寄存器**
- 注意NOT的操作数 (源，目的) 不能是立即数
- ADD命令中为**5位补码立即数**，注意参与运算的立即数要符号扩展到16位

- LEA指令不访存，用立即数的方式给出操作数相对**PC**的偏移
- PC相对偏移中，立即数为9位
- 基址＋偏移中，偏移为6位
- LEA指令中立即数为9位，**计算PC+offset放到目标寄存器中**
- **任何一条写寄存器的指令都会改变条件码**
  (ADD, AND, NOT, LD, LDR, LDI, **LEA**)
- Store指令和控制指令不改变条件码，但注意LEA会改变
- 8位陷入矢量（共支持256个服务程序，使用时要0扩展）
- 总线：一组16位
- 指令（79页那个图考试提供，注意trap指令要自己背下来，如下）
  - trap x20 get char（获得一个字符）
  - trap x21 put char （输出一个字符）
  - trap x22 puts （输出一个字符串）
  - trap x23 in （输入一个字符，会回显（会显示你输入的是什么，但程序会多输出`prompt`这句话，一般不用）
  - trap x25 halt
  - **注意trap指令输出的是R0中的内容，使用GETC也是放到R0中，PUTS的首地址放到R0，使用lea r0, label**
  - **前往书中看如何书写指令的汇编形式**

第六章（编程）：

算法的特性：有限性，确定性，可计算性

第七章（汇编语言）

```
以点开头的都是伪指令如 .orig, .fill, .end
注意标号前不用加 .
lc3 中使用#表示10进制，x表示16进制，,分割操作数
lc3 中只有两种数据类型都占16位，定点整数（16位补码），字符（16位）
但实际上字符只占8bit，要高位0扩展，注意只是字符
如下图所示伪操作
指令不区分大小写
```

汇编过程：将`.asm`转为可执行目标文件`.obj`的过程,扫描两遍：

1. 第一遍：创建符号表
   - 扫描源程序文件，找到所有的标号，并计算相应的地址，创建符号表，**在第一遍计算地址**
2. 生成机器语言代码，利用符号表的信息将指令转化为机器语言代码

第八章-》输入输出

- **LDI，STI** 实现对设备寄存器的读写
- LC3采用内存映射而不是专门的指令，给每个设备寄存器分配一个内存地址
- 状态寄存器和数据寄存器

- 异步设备要先同步（**通过状态寄存器**），即设置状态寄存器，在操作之前检查设备是否准备好（**CPU轮询或设备中断**）
- 如下图所示：
- 当键盘就绪时（KBSR15位为1时：任何输入都将被忽略）
    - 检测KBSR15位是否为1，是则读取KBDR[7：0]
    - KBSR[15]设置为0
    - 键盘设置为使能，准备接收下一个字符
- 当显示器准备输出一个字符时（就绪位DSR[15]置为1）
    - 将数据写入DDR, 此时DSR[15]将会置为0
    - 写在DDR[7:0]的字符将会被输出
    - 在DSR[15]=0期间，所有写向DDR的数据都将悲忽律
    - 显示完成后，DSR[15]恢复至1
- 键盘与显示（怎么输入和怎么显示的三行代码，注意使用LDI，而不是LDR）
- 基于中断：
    - 因为轮询占有一定的处理器周期
    - **中断信号的产生要求中断使能位为1，且设备要就绪**
    - 指令具有优先级`0-7`级，7级最高
- CPU在指令执行的任何阶段检测是否有设备中断信号（**由硬件自动检测**），
- **中断信号的产生与检测**，15，14位，优先级，指令的原子性
- p144 图8-9

第九章-》trap程序和子程序

- trap指令可以有256个，在相应位置存放的是子程序的入口地址
- 使用trap的优点
    - 程序员与系统相关的细节隔离
    - 频繁使用的代码只需要写一次
    - 保护系统资源免受恶意/笨拙的程序员影响
- 子程序使用JSR或JSRR
    - JSR首先保存返回值到R7，跳到指定位置（PC相对寻址）
    - JSRR目标地址存放在寄存器中
- 子程序返回用RET
- 中断返回才用RTI
- 一般采用被调用者保存的模式
- 寄存器的保存与回复，注意一**些寄存器的特殊作用如R7作为返回地址，R0作为返回值**.....
- trap指令高位0扩展，进入程序
- tarp指令通过硬件将PC保存到R7
- 对于开发商开发的库，使用JSRR因为可能超出PC偏移所能表示的范围

第十章-》栈

- push和pop
- 中断具有随机性
- PSR[15]=0表示超级用户，1表示普通用户
- 注意栈顶指针R0存数据，R5是否溢出（1溢出），**R6栈顶指针**
- 栈运算（空栈或溢出）、ASCII与二进制的转换

- 中断的相应与返回，**只需要对PSR和PC的保存**，其他的由被调用者保存，**进入中断服务程序前保存**，保存在超级用户栈（内存中特殊的栈空间，仅供特权模式下的程序使用）
- 指向超级用户栈的指针保存在内部寄存器Saved.SSP中，指向普通用户栈的指针保存在内部寄存器Saved.USP中，所有程序都可以通过R6访问所有栈（即Saved.SSP或Saved.USP都可以赋值给R6
- 先压入PSR，然后压入PC，弹出时先弹出PC，在弹出PSR

```
保存现场
If Priv = 1 (用户模式), Saved.USP = R6, then R6 = Saved.SSP.
将PSR和 PC压入超级用户栈.
设置 PSR[15] = 0 (超级用户模式).
设置 PSR[10:8] = 中断服务的优先级
设置 PSR[2:0] = 0.
定位中断服务程序,通过中断矢量表（x0100-x01ff，支持256个外部设备中断 x00-xff）
中断矢量高位扩展，设置 MAR = x01vv，这里 vv 为对应于终端设备的8-bit的中断矢量 （INTV）
(例如，键盘 = x80).
将中断矢量表对应的位置 (M[x01vv]) 写入 MDR.
设置 PC = MDR，指向中断服务程序的起始地址。
注: 以上过程发生在用户程序被中断处


返回：
将 PC从超级用户栈弹出 (PC = M[R6]; R6 = R6 + 1)
将 PSR从超级用户栈弹出 (PSR = M[R6]; R6 = R6 + 1)
如果 PSR[15] = 1, R6 = Saved.USP. (如果返回用户模式，需要重新加载用户栈指针；否则不改变栈指针内容)
```

- RTI（中断返回，**注意区别于RET**）
- 基于栈的算数运算（了解即可，课后习题没有）

> 邢金明，21/1