# CPU设计前言综述

#### CPU的功能

与一般的加法器或者译码器不同，CPU不完成某种特别的功能，它的输入不是靠芯片上的管脚来实现的，虽然说CPU和加法器的功能差不多，原理却相去甚远。CPU的输入是靠读取内存来实现的，因此功能的需要也就建立在读取的内容之上。也就是说，CPU在于实现程序。

**运算器**

**执行运算指令**

辅助性的功能：**读取输入，存储输出**

由于CPU的输入来自内存，输出也指向内存，所以CPU与内存之间构成了一种逻辑上的联系，CPU中的某些部件就是专门为指定内存的某个区域而准备的。

CPU的主循环如下：

读取指令

分析指令

执行指令

输出结果

读取指令

为了使整个工作过程有序进行，CPU必须定义某些部件来完成某些工作。

读取指令：PC—>MAR->内存->MDR->IR

分析指令：

执行指令：运算器（逻辑运算，算术运算），寄存器

输出结果：MAR，MDR

我们可以假设只有当PC中的内容为00000000时整个CPU停止工作，即进入停机状态；只有当PC改变为有效值时，CPU读取指令开始工作。

### 指令集和指令格式

指令有以下几种：

1. 存数据指令
2. 取数据指令
3. 加法
4. 减法
5. 比较
6. 跳转指令

### 如何与存储器进行数据交换

只能说这个问题又是老生常谈的问题了，因为实际上存储器控制权不在我，而在其本身；我已经说过，一切存储器的访问，给出CS,WR,RD,BH,BL信号即可

# 牛刀小试：一个简单的存储器访问CPU

假设现在一个基于我们自己设计的CPU的程序，它在0000处存放了一段程序，现在让我们用CPU读取这段程序，然后执行它，看看最后的结果。

语言顺序描述：

在1000处取出数的低位

在1010处取出数的高位

两者合并，并使程序跳转到合并的地址处执行

**首先设计指令集**

通用寄存器：R1，R2，R3，R4

指令寄存器：IR

地址寄存器：PC

STA x , Ri 存储

LDA Ri , x 取数

JMP x 转移到x处

HLT 停止工作 //给cpu设计一个reset位即可

**根据指令集设计指令格式**

有4条指令，两位数即可完成对应关系如下：00 HLT；01 JMP;10 LDA;11 STA;

HLT无需操作数，JMP需要一个操作数，LDA需要两个操作数。由于有3个寄存器，所以寄存器地址需两位编码。假设内存地址有n位，则指令格式为n+4位，也就是说最终内存的格式为n\*(n+4)，地址线为n位，数据线为n+4位，寄存器也为n+4位。令n=4即可。但是这样就不可能完成取高位，低位的操作，为了不改变数据总线的位数，再增加一个限制：在LDA和STA的操作种，寄存器R1是参考寄存器，如果R1的低位是1，就是取低位；R1的高位是1，就取高位。

地址线4位，数据线8位。

**CPU到存储器的设计和管脚对应**

存储器具有以下信号：（主要是输入信号）

CS，WR，RD，BH，BL，Data（8位双向），Addr（4位输入）

CPU之所以支持高低位的操作，其原因在于存储器原生支持这种操作。

存储器为4\*8位的格式。

CPU设计如下信号：

输入：clk，reset

输出：（到储存器）CS，WR，RD，BH，BL，Data，Addr

内部信号：hlt regAddr(通用寄存器地址)

CPU的寄存器：

MAR：存放要到Addr的地址

MDR：存放Data的数据

PC：下一条指令地址

IR：当前指令

**CPU的状态机模型**

当CPU启动的时候就进入read\_instr的状态，并且next\_stage就是analyze\_instr；

其状态转换如下：(状态在改状态完毕的时候改变或者在一个时钟周期之后改变)

当前状态 输入htl,reset 下一个状态 输出

read\_instr x analyze\_instr

analyze\_instr exec\_instr

exec\_instr output\_res

output\_res read\_instr

异常状态

**CPU的部件**

寄存器阵列：为了实现寄存器自动选中，直接使用寄存器数组即可。

存储器

地址总线

CU

数据总线

**VHDL语言实现**

为了使cpu开始工作，cpu的什么信号应当作出改变?

当PC有内容装入时，即PC从ZZZZZZZZ变成一个已知的数工作

或者，当我按下开机键的时候，cpu开始工作，并且PC=0

**实践中遇到的问题**

双向数据总线的置ZZZZZZZZ的问题：原则上应当在内部和外部分别置ZZZZZZZZ。一方把data置位ZZZZZZZZ意味着，该方暂时不会对数据线写。一般而言，如果一方要写数据，另一方就应当先把总线置位ZZZZZZZZ。对于全自动的存储器而言，外界不必关心这些状态，但是同时也要多花费一个时钟的时间。但是考虑到如果全由存储器内部来写，则ZZZZZZZZ本身也是数据，所以他也不知道数据总线是不是空闲状态。尴尬。

我们是这样的认为数据总线的：它内部有I，O两条数据线，如下

Y

X

一条X->Y,一条Y->X，综合的数据线显示就是：如果一条线上为Z，就显示另一条；如果两条线上相同，显示正确；如果一个是0，一个是1，显示X（强制未知）

而每条线上赋值只影响该条线，即只影响输出，而不影响输入。但是外界从该总线读取时，使用的是综合的数据。