《RISC CPU原型设计 》

[《RISC CPU原型设计 》 1](#_Toc182564539)

[1. 第一节：认识CPU 3](#_Toc182564540)

[1.1CPU分类 4](#_Toc182564541)

[1.2指令 5](#_Toc182564542)

[1.2.1指令分类 6](#_Toc182564543)

[1.2.2汇编语言 6](#_Toc182564544)

[1.3算计算组成原理 7](#_Toc182564545)

[1.3.1基础组件 7](#_Toc182564546)

[第二节：RISC原型设计与验证 8](#_Toc182564547)

[2.1指令集设计 8](#_Toc182564548)

[2.2CPU总线设计 9](#_Toc182564549)

[2.3 RISC架构设计 10](#_Toc182564550)

[2.3.1程序计数器 11](#_Toc182564551)

[2.3.2指令寄存器 12](#_Toc182564552)

[2.3.3AUL 12](#_Toc182564553)

[2.3.4累加器 13](#_Toc182564554)

[2.3.5指令译码 14](#_Toc182564555)

# 第一节：认识CPU

在数字电路中有一些比较重要的电路设计，其中就有CPU，这些不同的电路设计在我们的生活中扮演着不可缺少的角色，下来我们就最常见的几种比较重要的数字电路设计分析。

|  |  |  |
| --- | --- | --- |
| 电路设计 | 应用目标 | 核心功能 |
| CPU | 数学计算领域 | **科学计算** |
| DSP | 信号处理领域 | 信号算法运算 |
| GPU | 视频图像处理领域 | 视频图像算法运算 |
| PLC | 工业控制 | 电路控制 |

以上数字电路都扮演者不同的角色，而**cpu是针对科学计算设计的专用的电路**，本质上CPU就是科学计算电路，尤其数学运算，它的重要性不言而喻。

数学运算

数学运算以四则运算为基础+，-，\*，/

主要是加法器电路，减法器电路，乘法器电路，除法器电路

**运算数是否支持小数运算取决于运算电路**

科学计算

逻辑运算

**逻辑运算以与，或，非，异或，同或，移位运算为基础**

**运算是否支持全部的逻辑运算取决于运算电路**

在cpu的电路设计中科学计算电路是整个电路的核心，如果我们把这个电路认为是一个独立的电路模块，我们把这个模块叫做alu（Arithmetic&logical Unit）

## 1.1CPU分类

CPU作为科学计算的重要的电路设计，按照设计我们可以它分为2种，如下：

1. Risc

CPU的设计思路概括起来有2种，第1种是以简代繁，即较少简单的电路实现复杂的功能。

优点：

* 电路结构精简，对于性能的提升的有很大的空间
* 由于电路结构精简，所以芯片可以做到小型化，既可以用于嵌入式设备，又可以用于PC，以及其它领域
* 便于软件人员开发

缺点：

* 理论上不支持无限功能扩展
* 程序的运行效率保障

特点：

* **定长指令****（指令长度固定）**
* **指令少，用有限的指令实现复杂的功能**

1. Cisc

CPU的设计思路概括起来有2种，第2种通过功能扩展实现电路。

优点：

* 理论上支持功能的无限扩展，可实现很复杂的功能

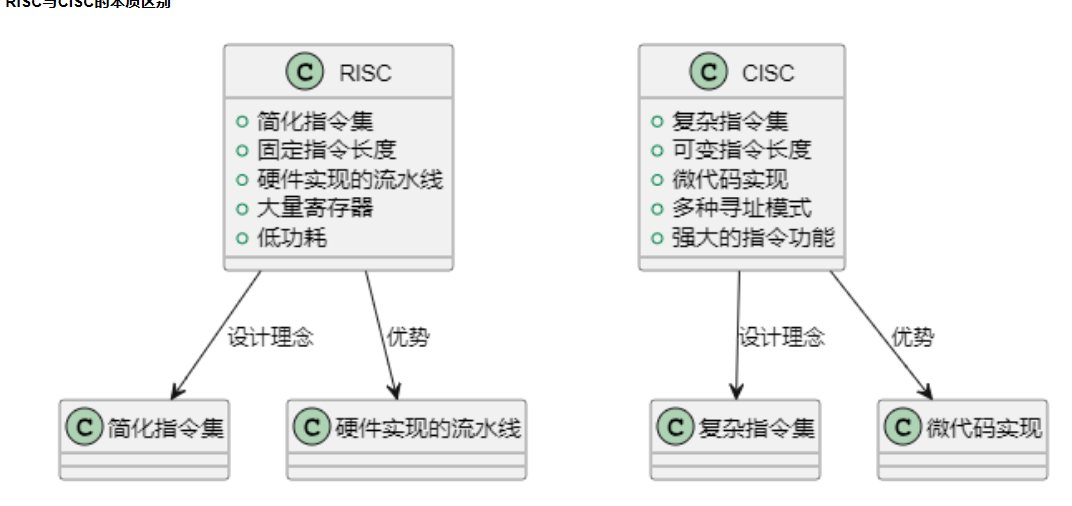
缺点：

电路复杂，不利于译码

特点：

* **变长指令（指令长度不固定）**
* **指令多，每一个功能设计一条或多条单独的指令，随着功能的扩展，指令会越来越多**。

这2种设计各有优缺点，总的来说RISC是相对更符合我们的生活需要，因为我们在**实际中用的功能也是有限的**，80%的应用都不需要太复杂的运算，将来risc是发展趋势，cisc逐渐将被淘汰。



## 1.2指令

CPU是科学运算电路，具体需要何种运算，是要按照我们的要求运算，通过人机交互实现，通过寄存组（存储器）参数解析实现交互，这个参数就是我们这里所谓的指令，这个参数指示CPU需要何种操作，换句话说指令时CPU中必不可少的设计，是人机交互的重要部分。

指令有定长指令和变长指令，RISC为定长指令，CISC为变长指令，指令由一串二进制数组成，有固定的格式：

指令 = 操作码 + 操作地址

操作码 ：指示该指令是何种操作

操作地址 ：有些指令需要读写数据等操作，参与运算数，甚至指令所在存储空间的地址。

### 1.2.1指令分类

1. 0地址指令

例如：NOP ;

1. 1地址指令

例如：ld 0x03

1. 2地址指令

例如:

Sub 0x03 rs

1. 3地址指令

例如:

xSub  0x03 0x05 rs

1. 4地址指令

Jsadd 0x03 0x05 rs 0x06

注意：一条多地址指令可以由多条少地址指令组合在一起实现该功能。

### 1.2.2汇编语言

在实际中如果我们记忆每一条指令，记忆二进制数较难，所以我们都用一种较为容易记忆的方式，**即指令 = 助记符 +指针**，也就是我们的汇编语言，每一个CPU都有自己的汇编语言，通过编译软件最终将汇编语言编译为二进制指令，即机器码。

助记符： 每条指令的操作码的符号表示

指针 ： 操作地址的表示，可以使一个数，也可以是寄存器名

### 1.3算计算组成原理

CPU的设计**遵循计算机组成原理**，按照计算机组成原理，CPU是由组件（能够独立完成某一功能的电路单元）构成，如下：

Cpu = 基础组件 + 扩展组件

基础组件 ：是构成CPU核心，是必不可少的组件，能够完成基本运算

扩展组件 ：在基础组件的基础上扩展的电路组件

### 1.3.1基础组件

一个cpu只要包含了基础组件就能够完成基本的操作，按照计算机组成原理它由5个组件构成。

1. 程序计数器

用于**产生下一条指令的指针**，CPU需要人机交互，交互的指令保存在存储器中，在实际中我们需要顺序读取指令。

1. 指令寄存器

指令寄存器（IR，Instruction Register），用于暂存当前正在执行的指令。指令寄存器的时钟信号是clk，在clk的[上升沿触发](https://baike.baidu.com/item/%E4%B8%8A%E5%8D%87%E6%B2%BF%E8%A7%A6%E5%8F%91/3859505?fromModule=lemma_inlink)。指令寄存器将数据总线送来的指令存入16位的寄存器中，但并不是每次数据总线上的数据都需要寄存，因为数据总线上有时传输指令，有时传输数据。由CPU状态控制器的 Ir\_ena信号控制数据是否需要寄存。复位时，指令寄存器被清零

1. Alu

CPU的核心组件，完成科学计算

1. 累加器

当我们ALU计算结果计算完成之后，我们需要把结果保存起来，需要一个寄存器，这个寄存器就是累加器。

在现在的CPU设计中除了累加器，都扩展了其它寄存器，比如标志寄存器，通用寄存器组。

1. 时序控制

核心是指令译码，它控制取指令，分析执行指令（指令译码）

# 第二节：RISC原型设计与验证

随着SOC的普及，掌握soc的开发基础对我们来说也是必备技能，通过RISC原型设计能够帮我们更好的理解SOC开发，换句话说是SOC开发的基础。

## 2.1指令集设计

这里我们规定指令采用16位指令，为了简化设计，我们只支持0地址和1地址类型的指令。

0地址类型

|  |  |
| --- | --- |
| Op[15:13] | NULL |

1地址类型

|  |  |
| --- | --- |
| Op[15:13] | 操作地址 |

|  |  |  |
| --- | --- | --- |
| 主机符 | 操作码 | 描述 |
| NOP | 3’B000 | 空操作 |
| LDA | 3’B001 | 从操作地址所指向的空间的数读出保存累加器中 |
| STO | 3’B010 | 把累加器数写入操作地址所指向的空间 |
| ADD | 3’B011 | 从操作地址所指向的空间的数读出与累加器数相加，结果保存累加器 |
| NXOR | 3’B100 | 从操作地址所指向的空间的数读出与累加器数相异或，结果保存累加器 |
| SFT | 3’B101 | 从操作地址所指向的空间的数读出,循环左移移位，结果保存累加器 |
| SKP | 3’B110 | 跳过下一条指令，执行下下一条指令 |
| NJMP | 3’B110 | 无条件跳转到操作地址所指向的空间 |

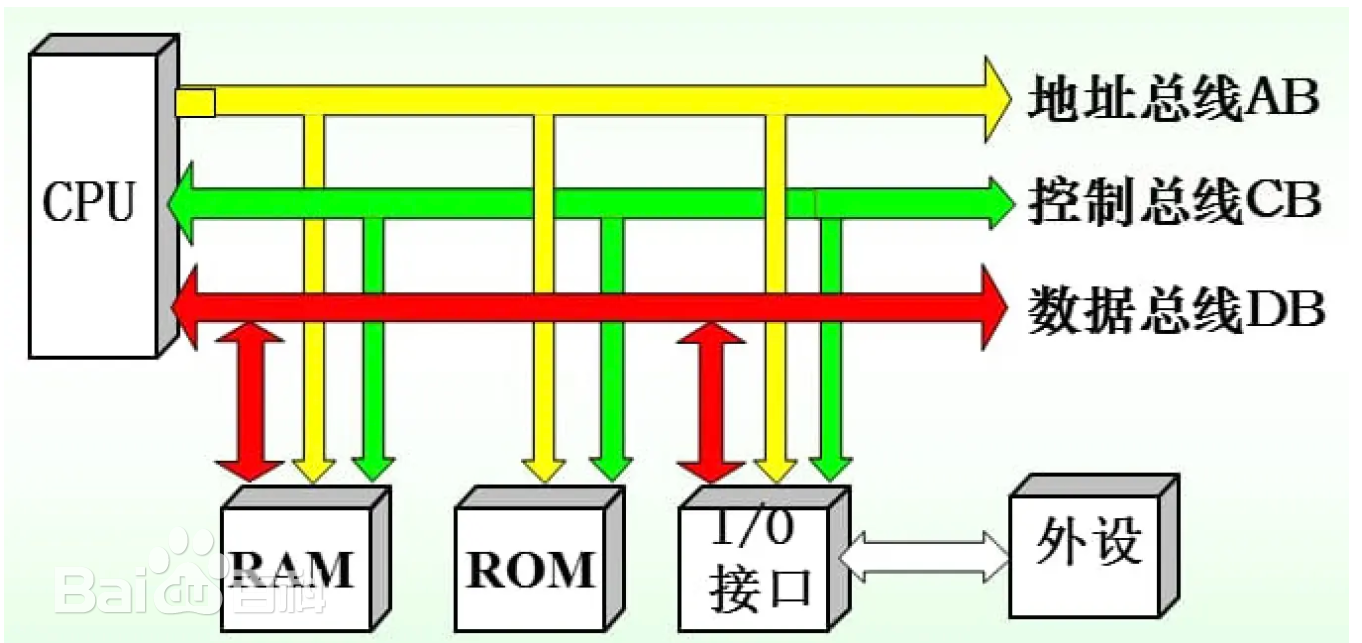
## 2.2CPU总线设计

CPU作为数字电路，独立的电路设计，与其它电路交互通过CPU总线进行交互，按照计算组成原理，CPU总线有3种：

数据总线：用来读写数据的信号线

地址总线：用来指示读写数据的指针

控制总线：读写指示信号，中断信号



这里我们设计如下：

|  |  |  |
| --- | --- | --- |
| 信号名 | 属性 | 描述数据总线 |
| bus\_data[7:0] | Inout | 数据总线 |
| Bus\_addr[12:0] | out | 地址总线 |
| bus\_wr | out | 控制总线，暂不支持中断  当wr==1表示写  当rd==1表示读 |
| bus\_rd | out |

## RISC架构设计

Ir[12:0]

en\_inc(当en\_inc==1pc自加1)

a\_sel

ld\_acc

en\_alu

ld\_ir

ld\_pc

en\_inc

ir\_regester

pc\_gen

alu

ld\_pc（ld\_pc==1pc更新为ir[12:0]）

pc[12:0]

ld\_ir（当ld\_ir==1从bus\_data上加载指令）

Ir[15:13]

**bus\_data[7:0]**

Ir[15:13]

**bus\_data[7:0]**

Ir[15:0]

en\_alu（en\_alu==1表示out\_alu输出结果）

out\_acc[7:0]

out\_acc[7:0]

out\_alu[7:0]

out\_alu[7:0]

bus\_data\_link

bus\_data双向数据线三态门控制信号

a\_sel

0:bus-addr输出pc

1:bus\_addr输出ir[12:0]

ld\_acc(当ld\_acc==1将out\_alu保存在out\_acc寄存器中)

bus\_ctl

ui

时序控制信号（人机交互控制）

acc

**wr**

**rd**

bus\_data\_link

risc

**bus\_data[7:0]**

out\_acc[7:0]

pc[12:0]

ir12:0]

**bus\_addr[12:0]**

pc[12:0] :指令地址

ir[15:0] :指令

out\_acc[7；0] :累加器值

output\_alu[7:0] ：alu结果

### 2.3.1程序计数器

功能：产生存放指令的存储器地址

rst\_n

clk

pc\_gen

en\_inc(当en\_inc==1pc自加1)

pc[12:0]

Ir[12:0]

ld\_pc（ld\_pc==1pc更新为ir[12:0]）

分析：程序计数器的地址计算分2种情况

1：正常模式 ：地址自动加，相当于计数器

2：跳转 ：当我们某一时刻希望从新的地址读取指令，这情况下需要跳转

端口说明

|  |  |
| --- | --- |
| 信号名 | 描述 |
| 系统相关 | |
| clk | RISC核心工作时钟 |
| rst\_n | RISC复位，低电平有效 |
| 指令时序控制相关 | |
| en\_inc | (当en\_inc==1pc自加1) |
| ld\_pc | (ld\_pc==1pc更新为ir[12:0]) |
| 操作地址 | |
| Ir[12:0] | 操作地址 |
| 输出指令指针 | |
| pc[12:0] | 指令指针 |

### 2.3.2指令寄存器

功能：用于加载保存当前指令的寄存器

rst\_n

clk

ld\_ir（当ld\_ir==1从bus\_data上加载指令）

Ir[15:0]

ir\_regester

**bus\_data[7:0]**

端口描述

|  |  |
| --- | --- |
| 信号名 | 描述 |
| 系统相关 | |
| clk | RISC核心工作时钟 |
| rst\_n | RISC复位，电平有效 |
| 指令时序控制相关 | |
| ld\_ir | 当ld\_ir==1从bus\_data上加载指令 |
| CPU总线相关 | |
| bus\_data[7:0] | 数据总线 |
| 输出指令 | |
| ir[15:0] | 当前指令 |

### 2.3.3AUL

功能：核心电路，是RISC的关键，按照指令完成科学计算

rst\_n

clk

out\_acc[7:0]

**bus\_data[7:0]**

Ir[15:13]

out\_alu[7:0]

alu

时序控制信号（人机交互控制）

en\_alu（en\_alu==1表示out\_alu输出结果）

端口描述

|  |  |
| --- | --- |
| 信号名 | 描述 |
| 系统相关 | |
| clk | RISC核心工作时钟 |
| rst\_n | RISC复位，低电平有效 |
| 指令时序控制相关 | |
| en\_alu | en\_alu==1表示out\_alu输出结果 |
| 操作码 | |
| Ir[15:13] | 指令操作码 |
| 参与运算的数相关 | |
| bus\_data[7:0] | RISC的数据总线 |
| out\_acc[7:0] | 累加器中的数据 |
| 运算结果输出 | |
| out\_alu[7:0] | 运算结果 |

### 2.3.4累加器

功能：保存ALU计算结果

rst\_n

clk

ld\_acc

out\_alu[7:0]

out\_acc[7:0]

acc

端口说明

|  |  |
| --- | --- |
| 信号名 | 描述 |
| 系统相关 | |
| clk | RISC核心工作时钟 |
| rst\_n | RISC复位，低电平有效 |
| 指令时序控制相关 | |
| ld\_acc | 当ld\_acc==1保存alu计算结果 |
| ALU输出 | |
| out\_alu[7:0] | ALU输出 |
| ACC输出 | |
| out\_acc[7:0] | ACC输出 |

### 2.3.5指令译码

功能：主要完成指令译码，根据当前指令（存放在指令寄存器中），解析该指令，控制CPU的运行状态。

程序计数器

指令地址

分析指令

取指令

执行指令

累加器

Alu

指令寄存器

指令译码

通过分析CPU按照指令运算，整个过程就是取指令，分析执行指令，由于每条指令复杂程序不一样，所以需要的时间可能不同，如果我们给每条指令分配不同的时间，这样电路控制难以控制，所以CPU一律给所有的指令分配的是同样的时间，我们把这个时间叫做指令周期，该时间是由设计者根据实际情况决定，比如8051单片是8个机器周期（核心工作时钟的周期）。

CPU运行可以理解为周期性的取指令，分析执行指令，这里规定咱们的设计指令周期为8个机器周期。

**rd**

**wr**

bus\_data\_link

a\_sel

Ir[15:13]

ld\_acc

en\_alu

ld\_ir

ld\_pc

en\_inc

ui

risc

端口说明

|  |  |
| --- | --- |
| 信号名 | 描述 |
| 系统相关 | |
| clk | RISC核心工作时钟 |
| rst\_n | RISC复位，低电平有效 |
| 操作码 | |
| Ir[15:13] | 指令操作码 |
| 指令时序控制相关 | |
| en\_inc | 当en\_inc==1指令地址自加1 |
| ld\_pc | 当ld\_pc==1指令地址更新为操作地址 |
| ld\_ir | 当ld\_ir==1从data\_bus加载指令 |
| en\_alu | 当en\_alu==1输出ALU计算结果 |
| ld\_acc | 当ld\_acc==1保存ALU结算结果到累加器 |
| a\_sel | CPU总线 地址总线选择  0：指令地址  1：操作地址 |
| bus\_data\_link | CPU总线数据总线三态控制  1：输出  0：输入 |
| CPU控制总线相关 | |
| wr | CPU控制总线写有效信号 |
| rd | CPU控制总线读有效信号 |

### 2.3.6CPU总线控制程序设计

功能： 由于CPU内部的地址有指令地址，还有操作地址，所以地址总线我们需要控制输出，

有时需要将结算结果写入外部存储空间，因此我们需要通过三态控制数据总线

本模块主要针对CPU总线的数据总线控制，地址总线控制

|  |  |  |
| --- | --- | --- |
| CPU总线 | 总线控制信号 | 说明 |
| Bus\_addr[12:0] | a\_sel | A\_sel 选择  0 指令地址  1 操作地址 |
| **Bus\_data[7:0]** | Bus\_data\_link | Bus\_data\_link 选择  1 写数据  0 读数据 |

bus\_data\_link

bus\_data双向数据线三态门控制信号

**bus\_data[7:0]**

out\_acc[7:0]

a\_sel

0:bus-addr输出pc

1:bus\_addr输出ir[12:0]

**bus\_addr[12:0]**

ir12:0]

pc[12:0]

bus\_ctl

端口说明

|  |  |
| --- | --- |
| 信号名 | 描述 |
| CPU地址总线选择有关 | |
| bus\_addr[12:0] | CPU地址总线 |
| a\_sel | A\_sel 选择  0 指令地址  1 操作地址 |
| pc[12:0] | 指令地址 |
| Ir[12:0] | 操作地址 |
| CPU数据总线选择有关 | |
| bus\_data\_link | Bus\_data\_link 选择  1 写数据  0 读数据 |
| bus\_data | CPU数据总线 |
| out\_acc[7:0] | 累加器 |
| …. |  |

### 2.3.7最小系统

CPU和其它的电路交互，都是通过寄存器参数解析，即命令解析，完成运算，并且在运算的过程中需要读取外部数据参与运算，并且有时我们希望将运算结果保存在外部存储存储器中。

综合所述，不管是读取命令还是读写数据，都是通过访问存储空间的形式体与其他电路进行交互。

在CPU系统中，我们搭建一个系统，这个包含了其运行的最低环境，我们把这个系统叫做最小系统。

risc

mem\_porcess

（rom）

wr

bus\_addr[12:0]

bus\_data[7:0]

rd

最小系统样例

led[7:0]

Pio

mem\_run

(ram)