|  |
| --- |
|  |
| **计算机组成原理专题实践** |
| **(2024版)** |
|  |
| **任国林 编** |
|  |

**东南大学计算机科学与工程学院**

|  |
| --- |
|  |

**目 录**

[1 课程设计的目的和要求 1](#_Toc158756955)

[1.1 课程设计目标 1](#_Toc158756956)

[1.2 课程设计要求 1](#_Toc158756957)

[1.3 设计报告要求 1](#_Toc158756958)

[2 计算机组成原理回顾 3](#_Toc158756959)

[2.1 计算机组成及工作过程 3](#_Toc158756960)

[2.2 CPU组成及工作原理 5](#_Toc158756961)

[2.3 主存组成及其连接 10](#_Toc158756962)

[3 RISC-V指令系统介绍 12](#_Toc158756963)

[4 模型机主机的设计过程 16](#_Toc158756964)

[4.1 需求分析 16](#_Toc158756965)

[4.2 总体设计 17](#_Toc158756966)

[4.3 数据通路设计与实现 19](#_Toc158756967)

[4.4 控制单元设计与实现 21](#_Toc158756968)

[4.5 CPU封装 24](#_Toc158756969)

[4.6 主机实现与测试 24](#_Toc158756970)

[附录A RISC-V单周期CPU设计 27](#_Toc158756971)

[A.1 需求分析 27](#_Toc158756972)

[A.2 总体设计 28](#_Toc158756973)

[A.3 数据通路设计 29](#_Toc158756974)

[A.4 控制单元设计 36](#_Toc158756975)

[附录B 主存的设计与连接 40](#_Toc158756976)

[B.1 主存的设计 40](#_Toc158756977)

[B.2 主存的连接 40](#_Toc158756978)

[附录C 支持不同访存粒度的BIU设计 42](#_Toc158756979)

## 1 课程设计的目的和要求

计算机组成原理是计算机专业的核心基础课程之一，主要学习计算机硬件的基本组成及其工作原理。计算机组成原理专题实践是计算机组成原理的后续实践课程，目的是基于计算机组成原理的理论知识，通过设计并实现一个模型机主机，加深对计算机组成原理的掌握，巩固已建立的计算机整机概念，同时培养计算机硬件的分析及设计能力。

### 1.1 课程设计目标

本课程的目标是设计并实现支持RISC-V指令集的模型机主机。计算机的主机由CPU及主存构成，本课程的设计暂不涉及异常处理、存储管理、总线等内容，以突出核心内容、降低设计难度、减少工作量。

主机的实现及调试在EDA平台Quartus II上完成。

### 1.2 课程设计要求

本课程的总体要求是设计并实现支持RISC-V指令集的模型机主机，通过执行机器语言程序进行主机的测试。

本课程的基本设计要求如下：

（1）CPU支持15条RISC-V指令，不支持异常及中断处理，主存采用实地址存储管理方式。支持的指令分别为add、sub、addi、ori、sll、sra、slt、sltu、lui、lw、lb、sw、beq、blt、jal。

（2）CPU中，指令周期为一个时钟周期；数据通路采用专用结构；存储器采用哈佛结构；BIU的数据引脚为32位，可寻址空间为32位，控制引脚包含读/写/时钟3根信号线；CU采用硬布线方式实现，采用同步方式进行μOP定时。

（3）主存中，指令存储器、数据存储器分别由同步ROM、同步SRAM组成，配置容量都为4KB，都直接与CPU连接，传输周期＜1个时钟周期。

（4）采用原理图方式（非Verilog HDL编程方式）实现所设计的电路，采用功能仿真方式进行所设计电路的调试。

能力有富余的学生可以超出基本要求，设计功能更强、性能更好的CPU，如多周期CPU、CPU支持SB指令、PC初值不为零等，以便获得更好的成绩。

### 1.3 设计报告要求

本课程的考核包括平时考核、设计验收、设计报告3个环节。设计验收通过检查设计&实现结果，考核设计及实践能力；设计报告通过检查设计&实现的分析与设计过程，考核理论基础、分析解决问题能力、硬件设计能力。

设计报告的内容主要包括设计目标、设计成果、CPU设计及实现、主机实现与测试、设计总结，可以参考设计报告模板。相关要求如下：

（1）设计目标。指具体实现的内容，包括CPU功能及结构、主存组成及连接。可以复述/修改设计要求的（1）～（3）。

（2）设计成果。包括完成内容及设计特色。完成内容用一句话描述，如实现了支持15条RISC-V指令集的单周期主机；设计特色指超出基本要求的成果，仅完成基本要求可不写，如多周期CPU、PC初值不为零等。

（3）CPU设计及实现。包括需求分析、总体设计、数据通路设计与实现、CU设计与实现、CPU封装5个环节。需求分析、总体设计、电路设计、实现及仿真的叙述应该环环相扣。每个环节应该有明确的结果，分析及设计过程的叙述是考核重点之一。电路实现只需用电路图来表示结果，并基于设计方案稍加说明即可，杜绝无设计过程的电路图解释（设计成绩基本没有）。电路仿真应该重视数据组织、结果分析的叙述，这是考核重点之二，测试的目标是电路功能具有完备性（不是具有可能性）。

（4）主机实现与测试。包括主存设计与实现、主机实现（即CPU-主存的连接）、主机测试3个环节。主机测试中，测试程序的功能应该是测试硬件功能的完备性、而不是可能性，测试程序的执行应该注意执行环境与程序存放之间的关联，测试结果的分析应该关注程序执行过程，这是考核重点之三。

（5）设计总结包括工作小结、设计体会，工作小结中需包含小组成员的工作分工。

报告撰写时，内容组织应紧扣设计要求，文字尽量简洁，图表尽量紧凑（文字≥6号字、图表可拆分）。注意，叙述的重点是设计要求的需求分析结果、总体及电路设计方案、所实现电路、测试组织及结果分析，抄写讲义中的基本原理及设计方法毫无意义（这些内容的具体应用才是所需要的）。

## 2 计算机组成原理回顾

### 2.1 计算机组成及工作过程

计算机系统由计算机硬件、计算机软件组成，通过在计算机硬件上执行计算机软件，来实现用户所定制的功能。计算机硬件由多个部件按照计算机模型组织而成，计算机软件指用户编制的程序，程序由若干有序的指令组成。

现代计算机都采用冯·诺依曼计算机模型。冯·诺依曼计算机中，硬件由运算器、控制器、存储器、输入设备和输出设备组成；软件由指令序列组成，指令类型有顺序型、转移型2种，指令的执行顺序用指令地址表示、由指令类型决定；工作方式为存储程序方式，其基本思想是：程序和数据预先存放在存储器中，机器工作时，自动、逐条地从存储器中取出指令并执行。

因此，程序执行过程是循环的指令执行过程，指令执行过程可分为取指令、分析指令、执行指令3个阶段，循环通过改变指令地址（计算指令地址）来实现，如图2.1所示。



图2.1 冯·诺依曼计算机的程序执行过程

#### 1．计算机硬件的组成

现代计算机的结构大多采用冯·诺依曼计算机结构，并对其进行改进，以提高计算机的性能。改进主要体现在多种存储器共存、以存储器为中心两个方面。

采用多种存储器共存的存储器结构，目的是解决存储器的速度-容量-价格之间的矛盾，方法是用层次结构的存储系统代替单一的存储器。目前，存储系统至少由主存、辅存组成，CPU访存时只直接访问主存，即按主存地址访问存储器。

采用以存储器为中心的硬件结构，目的是实现数据加工与数据传送的并行，方法是存储系统中增设Cache、I/O系统中支持DMA方式传送数据。进而，CPU可通过访问Cache来执行程序，数据传送可在主存-外设间进行，两者可同时进行。

因此，计算机的部件包括CPU、主存、辅存、输入设备、输出设备，CPU与主存合称为主机，输入设备、输出设备、辅存合称为外设。计算机的部件通常采用总线方式进行互连，以提高可扩展性，如图2.2所示。



图2.2 采用总线方式互连的计算机硬件结构

为了协调不同部件间的差异，各个部件需要通过总线接口电路连接到总线上。总线接口电路的功能与部件密切相关，命名方法较多，如I/O接口、主存控制器（DRAMC）、BIU、总线桥等。

系统总线指连接CPU、主存等主要部件的总线，由地址总线（ABus）、数据总线（DBus）、控制总线（CBus）三类信号线组成。总线上的部件（常称为设备）有主设备、从设备2种类型，各个主设备通过总线仲裁器分时获得总线使用权，各个从设备通过设备地址来进行标识。使用总线进行操作（信息传送）时，主设备先发送地址及命令，各个从设备主动判断自己是否为本次操作的目标从设备，是则响应总线操作（否则无动作），然后主、从设备根据命令类型完成数据交换，最后释放所控制的信号线。

注意，主存的结构有冯·诺依曼结构、哈佛结构2种类型。哈佛结构的主存由指令存储器、数据存储器组成，分别存放指令和数据；冯·诺依曼结构的主存由单一的存储器组成，可以存放指令和数据。

#### 2．计算机的工作过程

计算机的工作过程主要是执行程序的过程，而程序执行过程是循环的指令执行过程，循环变量为指令地址。

CPU通常用寄存器PC保存用作循环变量的指令地址，用寄存器IR保存当前指令的内容。由于指令类型有顺序型、转移型2种，相应地，下条指令地址的计算方法也有两类，顺序型指令为PC←(PC)＋“1”，转移型指令为PC←转移目标地址，前者结果与指令字长有关，后者结果与指令内容及执行结果有关。

由图2.1可见，计算指令地址与指令执行过程重叠时，可以提高程序执行的性能。由于顺序型指令的下条指令地址计算与当前指令的内容无关，故PC←(PC)＋“1”可以放在取指、分析或执行阶段实现；由于转移型指令的下条指令地址计算与指令的数据操作无关，故可以放在执行阶段实现。因此，现代计算机的程序执行过程如图2.3所示，所有指令的指令地址计算都在指令执行过程中完成。



图2.3 现代计算机的程序执行过程

至于计算指令地址怎么完成、何时完成，不同类型的CPU有不同的实现方法。多周期CPU中，PC←(PC)＋“1”在取指阶段完成，PC←转移目标地址在执行阶段完成；单周期CPU中，计算指令地址在执行阶段结束前完成即可。

按照存储程序工作方式的要求，程序执行前，需要完成2个准备工作，一是将程序及数据预先装入主存，二是将程序入口地址写入PC；程序执行时，按照图2.3的流程不停地执行指令。

为了便于程序装入主存，程序中每个地址对应的信息长度应与主存单元长度相同，程序空间可抽象为一个存储器，称为程序MEM。程序MEM都从零开始编址，故程序地址又称为逻辑地址，相应地，主存地址称为物理地址。可见，程序中指令是按逻辑地址访问存储器的，所访问的指令/数据放在主存中，因此，按逻辑地址访问主存包含2个步骤：先根据指令装入主存的位置，将逻辑地址变换为物理地址；再按物理地址访问主存。这个地址变换是由存储器管理单元MMU来实现的。

### 2.2 CPU组成及工作原理

现代计算机都采用存储程序工作方式，它要求CPU循环地执行指令，而每条指令的功能都是由指令系统约定的，可见，CPU的基本任务是实现存储程序工作方式及指令系统约定功能。另外，CPU还需处理指令执行过程中发生的异常，以及外设产生的中断请求。

#### 1．CPU的基本组成

CPU的主要功能是循环地执行指令、检测及处理异常和中断。具体来说，CPU应具有指令控制、操作控制、时间控制、数据加工、外部访问、异常及中断处理6个功能。

CPU的所有功能都是通过相应部件来实现的。为了实现指令控制，需设置PC、IR、ID；为了实现数据加工，需设置运算部件（如ALU）、寄存器组、状态寄存器PSR；为了实现外部访问，需设置MAR、MDR、总线逻辑电路；为了实现按逻辑地址访问存储器，需设置MMU；为了实现异常及中断处理，需设置中断机构；为了实现操作控制及时间控制，需设置时序信号形成电路、μOP控制信号形成电路。

因此，CPU由指令部件、运算器、总线接口单元BIU、存储器管理单元MMU、控制单元CU及中断机构6个部分组成，基本结构如图2.4所示。注意，MMU属于存储系统部件，为提高其实现性能，通常封装在CPU芯片中。



图2.4 CPU的基本结构

为了便于实现操作控制，通常将CPU划分为数据通路（Datpath）、控制器两个部分，数据通路指指令执行过程中数据所经过的路径及路径上的部件，其余部分为控制器。由于ID与指令执行的数据路径无关，故常将其划入CU；由于MMU、中断机构的功能虽与指令功能无关，但与数据通路有交互，故常将其划入数据通路。可见，图2.4中，除CU外的都是数据通路。

注意，存储器采用哈佛结构时，BIU由IBIU（指令BIU）、DBIU（数据BIU）组成。

#### 2．CPU的工作流程

CPU的主要功能是循环地执行指令、检测并处理异常及中断。异常及中断的处理由响应、处理、返回3个环节组成，其中仅事件检测&响应环节由硬件实现（中断机构），其余环节通过软件实现（执行程序），因此，CPU的工作流程由循环的指令周期、中断周期组成，如图2.5所示。注意，中断事件的检测&响应放在2个指令周期之间进行，而异常事件的检测&响应则在指令周期之中立即进行（图中为标出）。



为了便于实现指令执行过程，需要进一步细化指令周期的操作。指令由操作码、地址码组成，操作码用于指明操作类型、指令格式，地址码用于指明操作数地址及下条指令地址，操作数可存放在寄存器、存储器等部件中，因此，细化的指令执行过程有取指令、指令译码、取（源）操作数、数据运算、存（目的）操作数、计算指令地址6个基本步骤，如图2.6所示。其中，取操作数、存操作数步骤通常需要先计算操作数地址；指令地址计算步骤会与其它步骤并行（效果同图2.3），以提高性能。



图2.6 细化的指令执行过程

由于每个步骤都可由一个或几个操作实现，因此，指令执行过程由若干个有序的操作组成，取指令、指令译码的操作对所有指令是通用的，执行指令阶段的操作（取操作数/数据运算/存操作数）因指令功能而异，受指令中操作码、寻址方式等信息的影响。

同样地，中断响应过程也可通过若干有序的操作来实现。因此，CPU的工作流程由若干个有序的操作组成，这些操作由数据通路、中断机构来实现，由CU进行控制。可见，CPU的工作原理实际上就是在CU的控制下，有序地实现指令周期、中断周期的操作。

所有的操作都由基本操作构成，基本操作的定时都是通过主时钟脉冲来实现的，每个基本操作至少需要一个时钟周期。因此，指令周期可由若干个时钟周期组成，单周期CPU的指令周期由一个时钟周期组成，多周期CPU的指令周期由多个时钟周期组成。

#### 3．数据通路的组成

数据通路的功能是实现指令执行过程中的所有操作，因此，数据通路由功能部件、互连结构两个部分组成，互连结构又称为数据通路结构。

**（1）数据通路部件**

数据通路部件有操作部件（组合逻辑电路）、状态部件（时序逻辑电路）两种类型，分别实现数据加工、数据保存功能。

由指令执行过程的操作可知，取指令阶段的部件通常有PC、IR、指令存储器IMEM、加法器Adder（用于指令地址计算）；分析指令阶段的部件为ID，它不属于数据通路（没有数据操作）；执行指令阶段的部件通常有ALU（用于数据操作）、寄存器组、PSR、数据存储器DMEM。

注意，存储器采用哈佛结构时需设置IMEM与DMEM，采用冯·诺依曼结构时只需设置一个MEM；IMEM及DMEM可以是主存或Cache，它不属于CPU，但可以封装在CPU芯片中（如Cache）。

BIU是CPU内部与外部的接口电路，由内部端口、数据转换电路、总线逻辑电路（又称传输控制电路）组成，如图2.7所示。内部端口用于实现地址及数据的缓冲，如MAR、MDR；数据转换电路用于实现数据的存放方式、位数扩展所需的格式转换，如大端/小端、RISC-V中LB/LH指令所取8/16位数据转换为32位数据；总线逻辑电路用于实现外部访问的传输过程，如总线传输协议的实现。



图2.7 BIU的内部组成

BIU的数据转换电路功能也可以放在BIU与数据通路结构之间实现（如图2.4所示）。建议放在BIU中实现（如图2.7所示），总体结构相对简单。

多周期CPU中，同一部件可以在不同时钟周期被复用，故其它时钟周期需要使用的结果都必须保存在寄存器中，这些寄存器常称为附加寄存器。因此，多周期数据通路的部件还包含若干附加寄存器，附加寄存器的个数与数据通路的μOP功能组织有关。可见，PC、IR、BIU的内部端口都可用作附加寄存器。

可见，影响数据通路部件设置的因素主要有：

①指令的执行过程，如取指令阶段的操作是必须的；

②指令系统的约定功能，如执行指令阶段的操作是可变的；

③CPU的内部架构，如指令周期为单/多个时钟周期，存储器是/否采用哈佛结构等；

④数据通路结构，如总线方式互连与分散方式互连所需的连接部件不同。

**（2）数据通路结构**

数据通路结构有总线结构、专用结构两种类型。

总线结构通路中，同时只能传送一个数据，因此，每个部件的输出端必须通过三态门连接到总线上，每个部件的每个操作（组合逻辑操作）只能有一个未锁存信号线直接连接到总线上，其余输入端/输出端信号的前/后需要增设锁存器（寄存器），否则端口间信号会发生干扰。

专用结构通路中，同时可以传送多个数据，部件的输入端可以直接与其他部件的输出端连接，因此，当部件的输入端连接有多个输出端时，需要增设多路选择器。

对于指令执行过程来说，无论采用哪种数据通路结构，数据通路中的基本操作都主要有寄存器间传送、存储器读、存储器写、算逻运算4种类型，实现的功能分别为RD←(RS)、MDR←M[(MAR)]、M[(MAR)]←(MDR)、RD←(RS1) op (RS2)，其中，源操作数用存放部件的内容表示，目的操作数用存放部件本身表示，RS及RD表示寄存器，op表示运算类型，(*x*)表示寄存器*x*的内容，M[*y*]表示存储单元*y*的内容或存储单元本身。

可见，影响数据通路结构组织的因素主要有：

①数据通路的结构类型，如总线方式与点点方式的部件端口连接方法不同；

②取指令阶段的实现过程，如需要实现取指令阶段的数据路径；

③指令系统的约定功能，如不同指令执行阶段的数据路径有所不同。

**（3）数据通路的微操作及其控制**

微操作（μOP）指CPU内部的原子操作，可以通过给相应部件发送控制信号来实现，实现μOP的部件控制信号称为μOP控制信号（又称μOPCmd）。每个μOP都是一个独立操作，不依赖于其它μOP，因此，μOP的源数据、结果都必须放在状态部件中。

由指令执行过程的操作可知，数据通路中的μOP包括4个基本操作对应的μOP，及一些特殊功能的μOP，如PC←(PC)＋1、信号置位、信号复位等。

可见，影响μOP实现所需μOPCmd的因素主要有：

①数据通路的互连结构，如不同结构的连接部件不同，μOPCmd随之不同；

②部件的操作控制方式，如MEM读采用同步控制方式时需要2个μOPCmd（开始时和结束时），采用异步控制方式时只需1个μOPCmd。

**（4）数据通路的组织**

数据通路的组织就是指令系统的数据通路设计。数据通路组织的目标是实现指令系统中所有指令的指令执行过程的操作，这些操作受指令周期类型、存储器结构类型、数据通路结构类型、指令功能的影响，具体设计方法稍后介绍。

单周期CPU中，指令周期为1个时钟周期，故数据通路部件不能复用，存储器结构只能为哈佛结构，数据通路结构只能为专用结构。多周期CPU中，指令周期为*x*个时钟周期，不同指令的*x*可以不同，故数据通路部件可以复用，对存储器结构、数据通路结构的类型也没有限制。

因此，组织数据通路时，需要根据指令执行过程、所支持指令功能、指令周期类型、存储器结构类型、数据通路结构类型，来进行部件的配置和连接。

基于所组织的数据通路，就可以产生CPU工作流程中所有指令周期、中断周期对应的μOPCmd序列，以验证数据通路的组织是否正确。汇总所有指令周期、中断周期对应的μOPCmd序列，即可形成CPU工作流程的状态转换图，该状态转换图是控制器的设计需求（应用需求）。

#### 4．控制器的组成

控制器的功能是循环、有序地产生CPU工作流程所需的μOP控制信号。当前循环的μOP控制信号与指令类型、程序状态（如ZF/CF）、机器状态（如中断请求）有关，因此，控制器由指令部件、控制单元CU、中断机构组成，如图2.8所示。



图2.8 控制器的基本组成

控制单元CU是控制器的核心，由ID、时序信号形成电路、μOP控制信号形成电路组成。根据μOP控制信号的产生方法，控制器有硬布线、微程序两种类型。下面仅讨论硬布线控制器的组成。

硬布线控制器用有限状态机方法来描述CPU工作流程所需的μOP控制信号，时序信号用于表示工作流程的不同状态（步骤），用组合逻辑电路来产生当前状态的μOP控制信号。其中，所有的时序信号序列（由有效的时序信号组成）都是由时序系统确定的。

时序系统的组织基于CPU工作流程的状态转换图进行。现代计算机中，时序系统都采用节拍、工作脉冲2级时序，每个节拍可完成1个μOP。时序信号的个数需按最复杂的情况设置，可用来表示操作时刻或操作类型，通常选择后者；节拍信号的循环周期有定长、变长2种类型，对应的节拍信号序列有一种、多种，通常采用后者，工作脉冲信号序列只有一种；时序信号的定时方式有同步、异步、联合3种，通常采用联合方式。

**（1）时序信号形成电路**

时序信号形成电路的功能是循环地产生时序系统所需的时序信号序列。

时序信号形成电路由定序逻辑、定时逻辑组成，如图2.9所示。注意，时序信号个数＞2时，定序逻辑需要用环形信号发生器来实现，否则用门电路即可实现。



图2.9 时序信号形成电路的基本组成

定序逻辑用来产生所需的时序信号序列，由当前状态表示逻辑、下一状态产生逻辑两部分组成，第1级时序信号的下一状态产生逻辑中应包含复位逻辑、启动逻辑。定时逻辑用来控制节拍脉冲CP的时长（即当前节拍的时长），以实现μOP的定时。

μOP的定时方式有同步、异步、联合3种，异步方式基本不用。联合方式的基础是同步控制方式、支持异步控制方式，当前μOP所用的控制方式由μOPCmd指明，异步控制方式所需的时长由操作状态决定，因此，联合定时方式的定时逻辑中，除时钟脉冲信号CLK外，还需包含表示控制方式、操作状态的信号，如WMFC、mfc。

**（2）μOP控制信号形成电路**

μOP控制信号形成电路的功能是产生当前节拍的μOP控制信号。

由于不同节拍的μOP控制信号受指令类型、程序状态、机器状态的影响，因此，μOP控制信号形成电路的接口信号如图2.7所示。硬布线控制器通过组合逻辑电路形成μOP控制信号，因此，μOP控制信号形成电路就是一个编码器。

**（3）控制单元的组织**

控制单元的组织就是基于指令系统及数据通路的控制单元设计。CU组织的目标是实现CPU工作流程的状态转换图。状态转换的条件除时序信号外，还包括指令类型（由ID产生）、程序状态、机器状态。

组织CU时，设计次序依次为ID、时序系统、时序信号形成电路、μOP控制信号形成电路。ID根据指令系统的约定来组织，时序系统根据状态转换图来组织，具体设计方法稍后介绍。

基于所组织的数据通路、CU，增加MMU、中断机构后，就构成了完整的CPU。

### 2.3 主存组成及其连接

主存是由定长单元组成的一维线性空间，按地址进行访问，故主存容量＝主存单元长度×主存地址个数。CPU可以直接访问主存，且主存容量可以选配，故CPU可寻址空间的大小通常等于主存最大容量时的主存地址个数，即主存地址空间的大小。

主存单元长度（又称编址单位）由指令集结构确定，主存地址空间大小由系统设计者决定，配置的主存容量由用户决定。由于指令通常支持多种数据类型（含长度），为了提高存储效率，主存单元长度通常等于最短数据类型的长度；为了提高访问效率，CPU数据引脚的位数通常等于最长数据类型（整数）的长度，即*m*×主存单元长度（*m*≥1）。CPU的数据引脚位数≠主存单元长度、支持多种访问粒度的情况，增加了主存设计的复杂性。

#### 1．主存的组成与设计

主存由ROM及RAM组成，随机存取可以保证访存速度，ROM可以解决计算机启动时程序和数据的存放问题。

主存的设计方法是，对现有存储器芯片进行容量扩展，主要原因是ROM、SRAM或DRAM芯片的规格有限。存储器容量扩展的方法有位扩展、字扩展及字位扩展三种，SRAM与DRAM的字扩展、字位扩展方法有所不同。

主存引脚信号组织时，其数据引脚位数及信号方向都必须与CPU数据引脚相同。当CPU支持多种访存粒度时，还会设置数据掩码引脚。

主存内部电路设计时，假设CPU的数据引脚为*W*位，主存编址单位为*w*位，配置的主存容量为2*k*×*w*位，当仅支持一种访存粒度时，主存可由单体存储器组成，存储体参数为2*k’*×*W*，*k’*＝*k-*log2(*W/w*)；主存也可由交叉访问方式的*m*体交叉存储器组成，存储体参数为(2*k’*/*m*)×*W*。当支持多种访存粒度、关注访问效率时，主存必须由并行访问方式的*m*体交叉存储器组成，存储体参数为(2*k’*/*m*)×*W*。

并行访问方式的多体交叉存储器的组成示例如图2.10所示，需要设置*m*个数据掩码引脚，以实现写一个存储单元的功能，而读一个存储单元的功能通常放在CPU的BIU中实现（抽取速度更快），由其数据转换电路负责。



图2.10 并行访问方式的4体交叉存储器组成示例

#### 2.主存与CPU的连接

主存与CPU的连接线有3组：数据线、地址线、控制线（写使能、片选线等）。

数据线连接时，主存的数据线与CPU的数据线一一连接。

地址线连接时，主存的地址线与CPU地址线的低位信号线连接（主存空间常安排在CPU可寻址空间的低端），CPU地址线的高位信号线用于选择主存或外设。

控制线连接时，主存写使能线的有效逻辑为CPU的操作类型为写，主存片选线的有效逻辑为CPU的操作类型为存储器操作、目标地址在主存的地址范围之内。

## 3 RISC-V指令系统介绍

RISC-V是一种模块化结构、扩展灵活的指令集架构，由核心指令集、标准扩展指令集组成，核心指令集为32整数指令集RV32I，标准扩展指令集包括32乘除指令集RV32M、32位单精度浮点指令集RV32F、32位双精度浮点指令集RV32D、32位原子操作指令集RV32A、64位向量指令集RV64V，以及64位架构指令等。

指令系统的内容主要包括数据表示、操作数存放、寻址方式、指令格式、指令功能等。下面仅介绍RISC-V的RV32I指令集。

#### 1．数据表示

RV32I仅支持整数数据表示方法，每种数据表示方法又支持几种数据类型，不同数据类型的数码长度不同。

整数采用二进制定点格式表示，有符号整数、无符号整数的编码方式分别为补码、无符号编码，数码长度可以为8位、16位、32位。因此，机器字长为32位。

注意，理论上整数不支持逻辑运算，逻辑数应采用位向量格式表示（提高存储效率）；实际上，由于算术运算和逻辑运算可以复用器件，多数系统就用整数表示代替了逻辑数表示，相应地整数操作支持算术运算和逻辑运算，这是一种简化约定。

#### 2．操作数存放

RV32I的操作数可存放在指令寄存器、寄存器、存储器中，指令仅存放在存储器中。

**（1）RV32I的存储器**

RV32I的存储器按字节编址，逻辑地址空间为32位。

指令在存储器中的存放采用小端、对齐方式，对齐单位为32位（等于指令字长）。

数据在存储器中的存放采用小端、非对齐方式。设计时常扩展为对齐方式，以提高访问效率。

**（2）RV32I的寄存器**

RV32I的程序可见寄存器为32个32位通用寄存器，无程序状态寄存器。

通用寄存器的使用约定中，0#恒为零，1#保存过程调用的返回地址，2#保存栈指针等；寄存器中仅保存32位操作数。因此，取数指令中，短数据需（如LB指令）位扩展后再放到寄存器中；存数指令中，寄存器中数据可能（如SB指令）需位截断后再存入存储器。

#### 3．寻址方式

RV32I支持立即寻址、寄存器寻址、基址寻址、PC相对寻址、寄存器相对寻址5种寻址方式，地址形成方法如表3.1所示。其中，OPD表示操作数的值，EA表示操作数或指令的有效地址，(*x*)表示寄存器*x*的内容。

表3.1 RV32I支持的寻址方式

|  |  |  |  |
| --- | --- | --- | --- |
| 寻址方式 | 地址码组成 | 地址计算方法 | 注释 |
| 立即寻址 | imm | OPD＝imm | imm为立即数 |
| 寄存器寻址 | Rx | OPD＝(Rx) | Rx为通用寄存器号 |
| 基址寻址 | Rx及imm | EA＝(Rx)＋imm | imm为有符号偏移地址 |
| PC相对寻址 | imm | EA＝(PC)＋imm<<1 | <<1表示左移1位 |
| 寄存器相对寻址 | Rx及imm | EA＝(Rx)＋imm<<1 |  |

由表3.1可见，PC相对寻址与教材中的相对寻址仅命名不同，imm形成地址前都需要进行位扩展操作。

#### 4．指令格式

RISC-V采用32位定长指令字结构，指令格式有6种，如图3.1所示。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 31 25 | 24 20 | 19 15 | 14 12 | 11 7 | 6 0 |
| R型 | funct7 | rs2 | rs1 | funct3 | rd | opcode |
| I型 | imm[11:0] | | rs1 | funct3 | rd | opcode |
| S型 | imm[11:5] | rs2 | rs1 | funct3 | imm[4:0] | opcode |
| B型 | imm[12|10:5] | rs2 | rs1 | funct3 | imm[4:1|11] | opcode |
| U型 | imm[31:12] | | | | rd | opcode |
| J型 | imm[20|10:1|11|19:12] | | | | rd | opcode |

图3.1 RISC-V的指令格式

其中，opcode、funct3及funct7为操作码，rs1、rs2为源通用寄存器编号，rd为目的通用寄存器编号，imm为立即数或偏移量。

RISC-V指令格式中，操作码采用扩展编码方式，分开存放可以提高指令格式的规整性。指令格式及目的OPD地址位置都由操作码指明，目的OPD地址位置约定为最后一个寄存器型地址码。

#### 5．指令功能

RV32I的指令共47条，包括整数运算、转移控制、访存、系统控制。其中，整数运算指令共有21条，包括算逻运算、移位运算、比较、赋值等，使用R型、I型、U型指令格式；转移控制指令共有8条，包括跳转(含调用/返回)、分支，使用J型、I型、B型指令格式；访存指令共8条，可存取8/16/32位，使用I型、S型指令格式；系统控制指令共10条，包括内存操作排序、环境变量、控制寄存器设置等，使用I型、S型指令格式。

表3.2列出了RV32I指令功能及指令字格式。

表3.2 RV32I指令字格式一览表

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 助记符 | 指令字格式 | | | | | | | | 指令功能 |
| lui | U | imm[31:12] | | | | | rd | 0110111 | rd←imm‖000H |
| auipc | U | imm[31:12] | | | | | rd | 0010111 | rd←PC＋(imm‖000H) |
| addi | I | imm[11:0] | | | rs1 | 000 | rd | 0010011 | rd←(rs1)＋SExt(imm) |
| slti | I | imm[11:0] | | | rs1 | 010 | rd | 0010011 | rd←(rs1)＜有SExt(imm)? 1:0 |
| sltiu | I | imm[11:0] | | | rs1 | 011 | rd | 0010011 | rd←(rs1)＜无ZExt(imm)? 1:0 |
| xori | I | imm[11:0] | | | rs1 | 100 | rd | 0010011 | rd←(rs1)⊕ZExt(imm) |
| ori | I | imm[11:0] | | | rs1 | 110 | rd | 0010011 | rd←(rs1)|ZExt(imm) |
| andi | I | imm[11:0] | | | rs1 | 111 | rd | 0010011 | rd←(rs1)&ZExt(imm) |
| slli | I | 0000000 | | shamt | rs1 | 001 | rd | 0010011 | rd←(rs1)<<shamt |
| lrli | I | 0000000 | | shamt | rs1 | 101 | rd | 0010011 | rd←(rs1)>>Lshamt |
| srai | I | 0100000 | | shamt | rs1 | 101 | rd | 0010011 | rd←(rs1)>>Ashamt |
| add | R | 0000000 | | rs2 | rs1 | 000 | rd | 0110011 | rd←(rs1)＋(rs2) |
| sub | R | 0100000 | | rs2 | rs1 | 000 | rd | 0110011 | rd←(rs1)－(rs2) |
| sll | R | 0000000 | | rs2 | rs1 | 001 | rd | 0110011 | rd←(rs1)<<(rs2) |
| slt | R | 0000000 | | rs2 | rs1 | 010 | rd | 0110011 | rd←(rs1)＜有(rs2)? 1:0 |
| sltu | R | 0000000 | | rs2 | rs1 | 011 | rd | 0110011 | rd←(rs1)＜无(rs2)? 1:0 |
| xor | R | 0000000 | | rs2 | rs1 | 100 | rd | 0110011 | rd←(rs1)⊕(rs2) |
| srl | R | 0000000 | | rs2 | rs1 | 101 | rd | 0110011 | rd←(rs1)>>L(rs2) |
| sra | R | 0100000 | | rs2 | rs1 | 101 | rd | 0110011 | rd←(rs1)>>A(rs2) |
| or | R | 0000000 | | rs2 | rs1 | 110 | rd | 0110011 | rd←(rs1)|(rs2) |
| and | R | 0000000 | | rs2 | rs1 | 111 | rd | 0110011 | rd←(rs1)&(rs2) |
| jal | J | imm[20|10:1|11|19:12] | | | | | rd | 1101111 | rd←(PC)+4,  PC←(PC)+SExt(imm<<1) |
| jalr | I | imm[11:0] | | | rs1 | 000 | rd | 1100111 | rd←(PC)+4,  PC←(rs1)+SExt(imm) |
| beq | B | imm[12|10:5] | | rs2 | rs1 | 000 | imm[4:1|11] | 1100011 | if((rs1)＝(rs2))  PC←(PC)+SExt(imm<<1) |
| bne | B | imm[12|10:5] | | rs2 | rs1 | 001 | imm[4:1|11] | 1100011 | if((rs1)≠(rs2))  PC←(PC)+SExt(imm<<1) |
| blt | B | imm[12|10:5] | | rs2 | rs1 | 100 | imm[4:1|11] | 1100011 | if((rs1)＜有(rs2))  PC←(PC)+SExt(imm<<1) |
| bge | B | imm[12|10:5] | | rs2 | rs1 | 101 | imm[4:1|11] | 1100011 | if((rs1)≥有(rs2))  PC←(PC)+SExt(imm<<1) |
| bltu | B | imm[12|10:5] | | rs2 | rs1 | 110 | imm[4:1|11] | 1100011 | if((rs1)＜无(rs2))  PC←(PC)+SExt(imm<<1) |
| bgeu | B | imm[12|10:5] | | rs2 | rs1 | 111 | imm[4:1|11] | 1100011 | if((rs1)≥无(rs2))  PC←(PC)+SExt(imm<<1) |
| lb | I | imm[11:0] | | | rs1 | 000 | rd | 0000011 | rd←SExt(M[(rs1)＋SExt(imm)]) |
| lh | I | imm[11:0] | | | rs1 | 001 | rd | 0000011 | rd←SExt(M[(rs1)＋SExt(imm)]) |
| lw | I | imm[11:0] | | | rs1 | 010 | rd | 0000011 | rd←M[(rs1)＋SExt(imm)] |
| lbu | I | imm[11:0] | | | rs1 | 100 | rd | 0000011 | rd←ZExt(M[(rs1)＋SExt(imm)]) |
| lhu | I | imm[11:0] | | | rs1 | 101 | rd | 0000011 | rd←ZExt(M[(rs1)＋SExt(imm)]) |
| sb | S | imm[11:5] | | rs2 | rs1 | 000 | imm[4:0] | 0100011 | M[(rs1)＋SExt(imm)]←(rs2)低8位 |
| sh | S | imm[11:5] | | rs2 | rs1 | 001 | imm[4:0] | 0100011 | M[(rs1)＋SExt(imm)]←(rs2)低16位 |
| sw | S | imm[11:5] | | rs2 | rs1 | 010 | imm[4:0] | 0100011 | M[(rs1)＋SExt(imm)]←(rs2) |
| fence | I | 0000 | pred | succ | 00000 | 000 | 00000 | 0001111 | 内存操作排序(设置存储模型) |
| fence.i | I | 0000 | 0000 | 0000 | 00000 | 001 | 00000 | 0001111 |  |
| ecall | I | 000000000000 | | | 00000 | 000 | 00000 | 1110011 | 环境调用(产生调用异常) |
| ebreak | I | 000000000000 | | | 00000 | 000 | 00000 | 1110011 | 环境断点(产生断点异常) |
| csrrw | I | csr | | | rs1 | 001 | rd | 1110011 | rd←(csr)，csr←(rs1) |
| csrrs | S | csr | | | rs1 | 010 | rd | 1110011 | rd←(csr)，csr←(csr)|(rs1) |
| csrrc | S | csr | | | rs1 | 011 | rd | 1110011 | rd←(csr)，csr←(csr)&(rs1) |
| csrrwi | S | csr | | | zimm | 101 | rd | 1110011 | rd←(csr)，csr←ZExt(zimm) |
| csrrsi | S | csr | | | zimm | 110 | rd | 1110011 | rd←(csr)，  csr←(csr)|ZExt(zimm) |
| csrrci | S | csr | | | zimm | 111 | rd | 1110011 | rd←(csr)，  csr←(csr)&ZExt(zimm) |

指令字格式中，第一列为指令格式类型，固定编码的其余列为操作码，其余为地址码，rs1、rs2为源通用寄存器编号（占5位），rd为目的通用寄存器编号，imm为立即数或偏移量（占12位或20位），shamt为移位位数。

指令功能中，操作功能采用教材中写法，ZExt、SExt表示零扩展、符号扩展，“‖”表示内容拼接，分支指令比较后直接转移（不使用结果标志），Mem[x]表示从地址x开始的存储单元内容（单元个数参考指令助记符）。

## 4 模型机主机的设计过程

本课程的目标是设计并实现支持RISC-V指令集的模型机主机，通过执行机器语言程序进行主机测试。

模型机主机的设计一般采用自顶向下方法，因此，设计过程主要有如下5个步骤：需求分析、总体设计、数据通路设计与实现、控制单元设计与实现、主机实现与测试。

注意：应预先温习CPU、主存相关知识，在理论指导下实践，以提高设计效率。

建议：各个步骤所含内容及组织方法影响因素较多、叙述比较抽象，可以先通读一遍，具体设计时再细看，理论与实践相互对照，以提高理解与掌握速度。

### 4.1 需求分析

需求分析的目标是形成CPU及主存的设计需求，CPU的设计需求包括功能、结构、接口需求，主存的设计需求包括支持操作、引脚需求，涉及内容包括设计要求分析、指令系统分析。

**（1）设计要求分析**

本课程的设计要求包括CPU功能、CPU结构、主存组成及连接三个方面。

CPU功能的基本要求确定了CPU支持的指令功能、异常/中断类型、存储管理方式。例如，不支持异常及中断处理，即可缺省中断机构；主存采用实地址存储管理方式，即MMU的地址变换方法为：物理地址＝逻辑地址；指令功能需通过分析指令系统获得。

CPU结构的基本要求确定了CPU的结构、接口需求。例如，指令周期类型（单周期/多周期）、存储器结构类型、数据通路结构类型决定了数据通路的基本结构、部件组织方法；μOP控制信号形成方法、μOP定时方式决定了CU的基本结构；存储器结构类型、指令的访存位数及粒度决定了CPU的接口需求（如哈佛结构要求BIU有2组）。

主存组成的基本要求确定了主存的容量需求，而主存的引脚、支持操作需求则由CPU的接口所确定。例如，主存的数据引脚位数须与CPU数据引脚相同，主存的传输方式需满足CPU访存位数及粒度的要求。

**（2）指令系统分析**

指令系统分析包括指令集结构分析、所支持指令功能分析，目标是形成CPU的功能需求。指令集结构涉及的内容有数据表示、操作数存放、寻址方式、指令格式等。

通过分析指令集结构，可以获得CPU需要配置的寄存器位数及个数（含状态寄存器），以及所支持数据寻址方式的地址计算方法、指令寻址方式的地址计算方法等，这些是CPU的功能需求之一。

通过分析所支持指令功能，可以获得CPU所支持指令的操作类型，以及指令功能实现所需的数据路径，这些是CPU的功能需求之二。

另外，主存的存储管理方式实现逻辑地址-物理地址变换，是CPU的功能需求之三。系统总清实现必要部件的初始化，初始化PC、状态寄存器、时序信号形成电路，是CPU的功能需求之四。

CPU的设计需求（功能/结构/接口）需明确表示，否则后续设计无法进行。例如，数据通路宽度取决于整数的位数，ALU功能取决于定点操作的各种类型，指令地址计算部件功能取决于指令寻址方式的地址计算方法等；部件I/O信号的位数取决于各种操作的参数；功能部件的连接方式与数据通路结构类型、每条指令功能等有关。

注意1：只需分析拟支持指令的功能、寻址方式。增加拟支持的指令时，只认可需改变数据通路部件或数据路径的指令（否则背离课程目标）。

注意2：建议先实现单周期数据通路，感觉较好时再改为多周期数据通路，然后实现CU。因为多周期CPU的设计与实现有一定难度，必须保证按时提交设计成果；数据通路从单周期改为多周期的工作量不大，但CU的改动较大。

### 4.2 总体设计

总体设计的目标是组织CPU内部每个模块的功能及接口，包括功能模块划分、模块功能分配、模块接口设计3个环节。

主存通常组织成一个基本模块，其设计结果受限于CPU接口，因此，其需求分析、模块设计需放在CPU设计完成后进行。

#### 1．CPU功能模块划分

由CPU基本功能可知，CPU由运算器、指令部件、BIU、MMU、CU及中断机构6个部分组成，如图2.4所示。CPU通常划分为数据通路、控制器2个部分，以便于实现操作功能、操作控制，可见，数据通路包括运算器、指令部件、BIU、MMU、中断机构，控制器只由CU组成，数据通路的操作均由CU进行控制。

#### 2．CPU各个模块的功能分配

模块的功能分配包括子模块划分、各个子模块功能组织等内容。

**（1）数据通路**

数据通路的主要功能是实现指令执行过程中的所有操作，由运算器、指令部件、BIU、MMU、中断机构5个子模块组成，运算器通常由运算部件、寄存器组、状态寄存器组成，多周期数据通路中还包含若干附加寄存器。

运算部件负责实现所需的指令功能操作、数据寻址操作、指令寻址操作。设计时，所支持的功能由相应的指令系统需求分析结果决定。

寄存器组负责实现数据/地址在寄存器中的存放。设计时，寄存器个数取决于寄存器寻址方式的地址码位数，寄存器宽度取决于整数的最大位数，寄存器组的读端口、写端口个数取决于指令功能、数据通路结构类型的要求。

状态寄存器负责实现程序执行状态的存放。设计时，寄存器宽度、信息存放位置取决于指令系统中状态寄存器的使用约定。

指令部件负责实现指令控制功能，包括保存指令地址、使指令字在指令周期结束前保持不变，通常由PC、IR组成（ID已划入CU中）。设计时，必须包含PC，是否包含IR与指令周期类型（单周期/多周期）有关。

BIU负责实现外部访问（存储器/外设）的操作中转，由内部端口、数据转换电路、总线逻辑电路组成，如图2.7所示。设计时，内部端口的组织与指令周期类型、μOP功能组织有关，数据转换电路的功能与数据存放方式、访存位数及粒度等有关，总线逻辑电路的功能与访存位数及粒度、总线宽度有关。

MMU负责实现程序逻辑地址到物理地址的变换，地址变换失败时发出异常请求信号。设计时，其功能由存储管理方式决定，例如实地址管理方式功能为物理地址＝逻辑地址。

中断机构负责实现/异常中断请求的检测及响应。本课程要求CPU不支持异常及中断处理，设计时，中断机构可以缺省。

附加寄存器负责实现复用部件操作结果的存放。设计时，寄存器个数取决于指令执行过程的数据路径、相关μOP的功能组织。

**（2）CU**

CU的主要功能是循环、有序地产生CPU工作流程所需的μOP控制信号，由ID、时序信号形成电路、μOP控制信号形成电路组成，如图2.8所示。

ID负责实现产生指令的指令类型信号（操作码及寻址方式）。设计时，仅需输出所支持指令的指令类型信号，及一个非法指令信号（处理非法操作码异常时需要）。

时序信号形成电路负责产生各种时序信号序列，时序信号序列的组成由所组织的时序系统决定。设计时，先形成CPU工作流程的状态转换图，再基于状态转换图组织时序系统，最后用电路实现所组织的时序系统。时序系统的组成包括时序信号个数（含各级信号）、时序信号序列、时序信号定时3个方面。

μOP控制信号形成电路负责产生所有的μOP控制信号。设计时，μOP控制信号的个数由数据通路、μOP定时方式决定，信号产生的时机由状态转换图及时序系统决定。

**（3）CPU总清功能的组织**

CPU总清的功能是对必要部件进行初始化，例如，PC初值为开机后CPU执行的首条指令地址，时序信号形成电路的所有时序信号都为无效状态。

总清功能的实现方法是相关部件具有复位功能，受统一的复位信号（CPU引脚）控制。设计时，PC、状态寄存器、时序信号形成电路具有复位功能，操作控制由CPU的复位信号线（如Reset）触发即可。

#### 3．CPU各个模块的接口设计

数据通路及CU的接口组成，与其所分配的功能密切相关，受数据通路组织结果的影响，故数据通路、CU的接口设计，放在详细设计时讨论。

数据通路的接口信号包括：复位信号，BIU接口信号，工作脉冲信号、μOP控制信号，指令类型信号、程序状态信号。复位信号用于实现PC的初始化；BIU接口信号用于实现CPU的外部访问，工作脉冲、μOP控制信号用于实现μOP控制；指令类型信号、程序状态信号用于指明当前指令的约定功能。

CU的接口信号包括：复位信号，指令类型信号、程序状态信号、操作状态信号，工作脉冲信号、μOP控制信号。复位信号用于实现时序信号形成电路的初始化，指令类型信号、程序状态信号用于指明当前指令的约定功能，操作状态信号用于实现μOP的定时，工作脉冲信号、μOP控制信号用于控制数据通路实现当前指令的约定功能。

本课程设计中，中断机构模块缺省，因此，CPU由数据通路、CU两个模块组成，数据通路又由运算器、指令部件、BIU、MMU四个子模块组成，CPU总体结构如图4.1所示。由于要求存储器采用哈佛结构，故BIU需分为IBIU、DBIU。



图4.1 本课程要求的CPU总体结构

### 4.3 数据通路设计与实现

数据通路设计与实现的主要内容包括数据通路设计、数据通路实现、数据通路调试，数据通路设计指数据通路的详细设计，包含功能部件设计、部件互连设计2个环节。

数据通路由运算器、指令部件、BIU、MMU组成，运算器通常由运算部件、寄存器组、状态寄存器组成，多周期数据通路还包含若干附加寄存器。注意，每个模块的功能需求在总体设计时已经确定。

**（1）功能部件设计**

功能部件设计的任务是根据各模块的约定功能，设计每个功能部件。

①组织各个部件的功能

一个模块可以组织成一个或多个功能部件，目标是尽量复用部件（降低硬件成本）、简化互连与控制。组织方法是：不兼容、需同时使用的功能用不同部件实现，控制复杂性相当时尽量增加部件功能（减少互连复杂度）。

数据通路的各个模块中，运算部件通常组织成多个功能部件，寄存器组、BIU、MMU通常都组织成一个部件，指令部件、状态寄存器、附加寄存器等仅由寄存器组成的模块无需封装（既增加工作量、又增大出错概率）。

单周期CPU的运算部件中，指令寻址操作与数据操作（指令功能操作及数据寻址操作）都在同一时钟周期内进行，部件不能复用，数据操作的部件则可以供所有指令使用，故运算部件常划分为数据操作单元、指令地址计算单元2个部分。由于数据操作的类型较杂，数据操作单元常组织成多个功能部件，指令地址计算单元常组织成一个功能部件。

多周期CPU的运算部件中，三类操作的部件在不同时钟周期可以复用，根据拟定的部件复用方案，运算部件可组织成多个功能不同的功能部件，以降低成本。

②设计每个功能部件

部件的功能由总体设计结果约定，部件的设计包括I/O信号组织、内部逻辑设计2个环节。

I/O信号组织的目标是形成部件的功能表，表中枚举各种I/O信号取值所对应的功能。I/O信号有数据输入、数据输出、地址输入、功能选择、操作控制等类型，实现不同功能所需的信号有所不同。信号组织时，应尽量减少I/O信号的个数。

内部逻辑设计的目标是实现部件的功能表，需要选择适当的器件并进行连接。内部逻辑设计时，应尽量选择高效的器件、减少输出信号通过的器件个数。

**（2）部件互连设计**

部件互连设计的任务是为各条指令的数据路径建立部件连接。

部件互连的设计方法是，逐条增加指令所需的功能部件及信号线连接，直到所有指令处理完毕。由于部件互连的结果是相同的，故增加时的指令次序无所谓。

部件互连设计时应注意，①根据指令的数据路径（CPU功能需求之二），确定所需的功能部件，以及信号线连接方案；②尽量复用功能部件，以减少成本；③其他时钟周期需要使用的操作结果需保存到附加寄存器中，如指令字需用IR保存；④根据数据通路结构的类型，选用功能部件间的连接器件，如总线结构需使用三态门、锁存器，专用结构需使用多路选择器。

注意：电路设计是在纸上进行的，电路实现才是在计算机上进行的，没有设计的实现一定会效率低下、错误百出。

**（3）数据通路实现**

数据通路实现的任务是用EDA软件实现所设计的数据通路电路。

数据通路的实现方法是，分别实现每个功能部件的电路，最后根据部件互连设计结果连接成数据通路的电路。

电路实现方法由EDA软件的使用方法决定。例如，使用Quartus II进行电路实现的步骤如下：创建/打开工程文件、编辑原理图文件、设置顶层文件、编译工程文件，使用方法与使用VC++编程类似；每个原理图文件都可以生成一个符号文件，该符号文件可以作为元器件在其它电路中使用。

要求：除寄存器外的每个功能部件，都单独建立原理图文件，以节省调试时间，如ALU、ACU（指令地址计算单元）、ExtU（位扩展部件）、GPRs（寄存器组）、BIU等。数据通路的原理图文件中，除BIU接口信号外，还需输出一些调试信号（须包含指令字、GPRs读端口A、ALU输出端、GPRs写端口），以便于进行白盒测试。

注意1：当电路的I/O引脚数超过芯片支持的最大值时，需要将部分I/O引脚的属性设置为虚拟引脚，设置方法参见EDA使用说明书。

注意2：功能部件调试应紧随其实现进行，千万不要采用全部实现后再统一调试的愚蠢办法，这样会导致不同电路中的同一个错误需要分别修改。

**（4）数据通路调试**

数据通路调试的任务是用EDA软件调试所实现的数据通路电路。

数据通路的调试方法是，分别调试每个原理图文件，每个原理图文件的调试都包括组织测试数据、进行电路仿真、分析仿真结果、修正电路错误4个环节。

要求：每个原理图文件（功能部件）都需要单独进行调试。

注意1：应先完成各个功能部件的调试（每实现一个就调试一个），最后才进行数据通路的调试。因为某部件设计或实现的错误，会在使用该部件的电路中漫延，调试时发现错误源头十分困难，调试不充分时还发现不了。

①组织测试数据时，根据功能部件、数据通路调试的目标进行组织。

功能部件调试的目标是电路能否实现约定的操作功能（与指令功能无关！）。测试数据应枚举功能表中每个功能的输出信号组合（数字信号只需一组），以保证正确性。例如，ALU的加法运算测试应使各个结果标志均可为0和1。

数据通路调试的目标是电路能否实现每条指令的数据路径，检验方法是给出与指令μOPCmd序列对应的控制信号，观察数据通路能否正确实现指令执行过程及约定功能。组织测试数据时，首先测试复位功能，然后测试各条指令的执行过程及执行结果，不需要再测试功能部件的功能。

注意2：为了减少数据通路的调试工作量，数据路径相同、仅部件控制信号不同的各条指令（如ADD与SUB），只需测试一条指令即可。

②进行电路仿真时，仿真方法由EDA软件的使用方法决定。例如，使用Quartus II进行电路仿真的步骤如下：编辑仿真波形文件、生成功能仿真网表、开始仿真，仿真结果可立即输出或保存在指定文件中。

注意3：控制信号的改变应发生在节拍周期开始时，即图2.9中的CP上升沿到来时（正脉冲均用上升沿来定时），因为CU在时钟周期CLK上升沿改变CP。

注意4：Quartus II功能仿真方式忽略了部件的操作时延，为了实现在时钟周期结束时写结果，仿真波形文件中，时钟信号的上升沿应比新的数据及控制信号早一些，否则会在时钟周期开始时写结果。

③分析仿真结果时，应该分析每个输入信号变化带来的输出信号及其时序的正确性。分析基于理论结果进行，理论结果由设计方案、仿真数据得到。

注意5：为了减少分析工作量，只有首条指令的取指令过程需要进行分析。

④修改电路错误时，应同时修改与错误点相关的所有电路。

### 4.4 控制单元设计与实现

CU设计与实现的主要内容包括CU设计（详细设计）、CU实现、CU调试，CU设计又包含状态转换图形成、时序系统组织、ID设计、时序信号形成电路设计、μOP控制信号形成电路设计5个环节。

CU由ID、时序信号形成电路、μOP控制信号形成电路组成，如图2.8所示。CU有硬布线、微程序2种实现方式，本课程要求采用硬布线方式。硬布线CU都基于有限状态机模型来实现，有限状态机模型如图4.2所示。



图4.2 有限状态机模型

可见，“当前状态”及“下一状态产生函数”由时序信号形成电路实现，其输入信号为脉冲信号CP；“输出信号产生函数”由μOP控制信号形成电路实现，其输入信号可为指令类型、程序状态、机器状态，输出信号为所有的μOP控制信号。另外，时序信号形成电路还需要实现状态的定时机制（确定CP时长），即μOP的定时方式。

**（1）状态转换图形成**

状态转换图形成的任务是得到CPU工作流程的所有状态及其转换条件。该状态转换图是CU的设计需求（应用需求），本课程只需形成指令执行过程的状态转换图。形成基于所设计的数据通路、指令的约定功能进行。

状态转换图的形成方法是，按照指令执行过程、各条指令约定功能的要求，基于所设计的数据通路，列出各条指令执行过程的μOPCmd序列，汇总所有μOPCmd序列后即可形成。每种状态由一种μOPCmd组合构成，状态转换条件可以为指令类型（操作类型及寻址方式）、程序状态（如ZF/CF）。若CPU支持中断或异常处理，还需将中断周期的μOPCmd序列并入状态转换图。

**（2）时序系统组织**

时序系统组织的任务是确定时序信号的个数、各种时序信号序列的组成、时序信号的定时方式。组织基于所形成的状态转换图进行，仅基于硬布线CU进行讨论。

时序信号的类型有节拍、工作脉冲2级。节拍信号的个数等于状态转换图中最长路径的状态数，单周期CPU中为0个；工作脉冲的个数等于数据通路中所有μOP所用信号边沿的不同时刻个数，通常为2个。

时序信号序列有节拍信号序列、工作脉冲信号序列2级。节拍信号序列的个数及每个序列的组成受序列组织方案（节拍循环周期为定长/变长型）、状态转换图的影响。工作脉冲信号序列只有1种，其组成是每个工作脉冲信号轮流有效。

时序信号的定时方式有同步方式、联合方式2种。同步定时方式中，节拍周期＝时钟周期，理论上时钟周期＝max{*T*所有μOP*i*}，实际上时钟周期＝max{*T*基本μOP*i*}，复杂μOP可通过多个时钟周期实现（用连续的多个μOPCmd来控制）。联合定时方式中，时钟周期＝max{*T*基本μOP*j*}，变长μOP操作采用异步控制方式实现（用特殊μOPCmd来控制）。

**（3）ID设计**

ID设计的任务是产生所有指令的操作类型及寻址方式信号。设计基于指令字格式的约定进行。

ID的内部逻辑很简单，由译码器及一些门电路组成即可。

注意：非法指令有当作nop指令（空操作）、产生异常2种处理方法。前者的实现方法是，ID输出nop指令信号，μOP控制信号形成电路产生相应的μOPCmd。后者的实现方法是，向中断机构发出异常信号。本课程要求采用当作nop指令方法。

**（4）时序信号形成电路设计**

时序信号形成电路设计的任务是实现时序系统。时序信号形成电路由定序逻辑、定时逻辑组成，接口为节拍脉冲信号CP，如图2.9所示。设计基于所组织的时序系统进行。

硬布线控制器中，定序逻辑由当前状态表示、下一状态产生函数组成，每个状态用一个时序信号表示，所有时序信号的下一状态产生函数值，同时只有一个有效，这些有效的时序信号组成了时序信号序列。

定序逻辑电路中，节拍信号、工作脉冲信号的状态可用触发器（信号数＞2时）或门电路（信号数≤2时）表示；各个下一状态产生函数用组合逻辑电路实现。当前状态用门电路表示时，下一状态产生函数只能用当前状态产生函数表示；当前状态用触发器表示时，下一状态产生函数可用触发器入端信号的产生函数表示（下个节拍输出）。

定序逻辑电路还应具有复位功能，由复位逻辑、启动逻辑2个部分组成，复位逻辑（复位时）使第1级时序信号的输出全部无效，启动逻辑（复位后）使某个第1级时序信号首先有效，对第2级时序信号则无此要求。

定时逻辑电路中，同步方式定时时，输入信号仅为CLK，内部逻辑为CP＝CLK；联合方式定时时，输入信号需包含控制方式信号、操作状态信号，如WMFC、mfc，内部逻辑为CP＝(＋WMFC·mfc)·CLK。

注意：节拍周期开始于CP上升沿，CP＝CLK要求时钟周期开始于CLK上升沿。

**（5）μOP控制信号形成电路设计**

μOP控制信号形成电路设计的任务是实现所有μOP控制信号的有效逻辑。硬布线CU的μOP控制信号形成电路就是一个编码器，接口信号如图2.8所示，设计基于所形成的状态转换图、所组织的时序系统进行。

μOP控制信号形成电路的设计方法是，填写μOPCmd使用时间表、获得各个μOPCmd的逻辑表达式、实现各个μOPCmd的有效逻辑。

①填写μOPCmd使用时间表时，μOPCmd使用时间表是一张二维表（单周期CPU为一维表），每个节拍信号占一列（或行），每个μOPCmd占一行（或列），每个单元格中内容为当前μOPCmd在当前节拍的有效条件。填表有两个步骤：

·给状态转换图的每个状态打上时间戳，由于每个状态的时间戳≥1个，故时间戳需与转入条件绑定，转入条件为指令操作类型及寻址方式、程序状态、机器状态；

·将每个状态的每个转入条件填到相应单元格（个数≥1）中，每个单元格所在的行与状态所含的μOPCmd相对应，单元格所在的列与转入条件所绑定的时间戳相对应。

②获得各个μOPCmd的逻辑表达式的方法是，对μOPCmd使用时间表按行（或列）进行汇总、逻辑化简即可。

③实现各个μOPCmd的有效逻辑的方法是，用组合逻辑电路实现其逻辑表达式，即可产生所有的μOP控制信号。

**（6）CU实现**

CU实现的任务是用EDA软件实现所设计的控制单元电路。

CU的实现方法是，依次实现ID、时序信号形成电路、μOP控制信号形成电路，最后连接成CU。

电路实现方法与数据通路所使用的方法相同。

要求：ID单独建立原理图文件，其余两个电路包含在CU的原理图文件中；多周期CPU的时序信号形成电路也需要单独建立原理图文件，以便于单独调试。

**（7）CU调试**

CU调试的任务是用EDA软件调试所实现的控制单元电路。

CU的调试方法是，逐个调试每个原理图文件，每个原理图文件的调试都包括准备测试数据、进行电路仿真、分析仿真结果、修正电路错误4个环节。CU调试的目标是电路能否产生各条指令的uOPCmd序列。测试数据组织的要求与数据通路部件相同。

电路仿真方法与数据通路的仿真方法相同。结果分析的要求也与数据通路部件相同。

### 4.5 CPU封装

CPU封装的任务用EDA软件实现完整的CPU电路。

本课程的设计中，中断机构可以缺省，CPU由数据通路、CU两个模块组成，CPU的I/O信号包括复位信号、时钟信号、IBIU及DBIU接口信号。CPU封装时，只需建立原理图文件，将数据通路、CU这2个模块连接起来即可。

要求1：CPU单独建立原理图文件，以加深对CPU组成的理解。CPU不需要单独调试（封装的错误概率很小），调试放在主机测试时一起进行（需输出CPU的调试信号），以减少工作量。

要求2：CPU原理图文件的I/O信号需包括CPU的I/O信号，以及当前指令字（或IR）、GPRs读端口A、ALU输出端、GPRs写端口等调试信号，以便查看程序执行过程及指令执行结果。

### 4.6 主机实现与测试

主机实现与测试的主要内容包括主存设计与实现、主机实现、主机测试。

**（1）主存设计与实现**

主存设计与实现的主要内容包括主存设计、主存实现、主存调试，主存设计包含引脚信号组织、内部电路设计2个环节。设计基于主存的设计需求、CPU的接口要求进行，CPU的接口要求包括引脚要求（数据&数据掩码）、操作要求（类型/访存粒度）。

·主存引脚信号组织

数据&数据掩码引脚位数必须与CPU相应引脚位数相同；地址引脚位数基于配置容量、数据引脚位数计算得到；控制引脚包含操作控制、片选、时钟等引脚。

·主存内部电路设计

当CPU访问粒度≤数据引脚位数时，主存仅需支持常规传输方式，否则需支持突发传输方式。当CPU只支持1种访问粒度时，主存可由单体存储器组成，否则需由并行访问方式的多体交叉存储器组成（示例如图2.10所示）。

·主存实现

主存的实现很简单，只需分别建立原理图文件，利用所选器件，实现所设计的IMEM、DMEM电路即可。

注意1：主存实现可以放在主机实现中进行，以避免无谓的封装。

注意2：Quartus II提供的都是同步ROM/RAM，默认带输出锁存功能（读操作需要2个时钟周期）。若希望存取周期≤1个时钟周期，则须取消ROM及RAM的输出锁存功能。

·主存调试

主存的仿真方法与数据通路部件的仿真方法相同。

注意：主存调试过程可以缺省（太简单了）。

**（2）主机实现**

主机实现的任务用EDA软件实现完整的主机电路。

主机由CPU及主存组成，实现时，只需建立原理图文件，将CPU、主存通过连接电路连接起来，即可实现完整的主机电路。

CPU-主存连接电路的组织，在计算机组成原理课程中已详细讲过，不再赘述。

注意：主存的时钟信号线应连接到BIU接口信号中CBus的时钟信号线，而不是输入到CPU的时钟信号线（两者的频率可能不同）。

要求：主机原理图文件的I/O信号需包括CPU的I/O信号及其调试信号，以便于查看CPU执行程序过程及指令执行结果。

**（3）主机测试**

主机测试的目的是调试所实现的主机，直至正确地满足设计要求，设计要求是主机能够正确地实现复位功能、程序执行过程、所支持指令功能。内容主要包括组织测试环境、编写测试程序、执行测试程序、分析执行结果、修正电路错误5个环节。

·组织测试环境

组织测试环境的任务是实现存储程序工作方式的执行准备，包括将测试程序装入主存、将测试程序入口地址写入到PC中两个方面。这些工作在计算机中是由操作系统及硬件完成的，本课程设计中均由系统复位操作来实现。

对于将测试程序入口地址写入到PC中的要求，可以采用使PC初值＝测试程序入口地址的方法实现，PC初值（假设为*x*）的设置由数据通路实现。

对于将测试程序装入主存的要求，可以采用将测试程序保存在IMEM中的方法实现，测试程序入口地址的存放位置需与PC初值相同。

·编写测试程序

编写并保存测试程序的任务是组织、编写测试程序。测试程序入口地址应与PC初值相同，以实现实地址存储管理方式。

测试程序需满足如下要求：应包含支持的所有指令，以测试设计是否有遗漏；每条分支指令应能够产生2种执行结果，以测试设计的正确性；相邻访存指令的MEM地址应不同，以防遗漏设计错误；访存指令的地址应包含越界情况（＞所配置主存的地址范围），以测试主存连接是否正确；当前指令的目的操作数应作为下条指令的源操作数，以便于查看每条指令的执行结果；应包含不支持的指令，以测试非法指令处理的正确性。

·**执行测试程序**

执行测试程序的任务是执行测试程序、显示/保存执行结果。

执行测试程序是通过对主机电路进行仿真来实现的，应先建立仿真波形文件，然后进行电路仿真，仿真结果（程序执行结果）可立即显示或保存到文件中。

仿真波形文件中，首先进行系统复位操作（初始化CPU），其余操作的控制信号将由CPU自动给出。

注意：输出信息应为十六进制（或二进制），以便于查看结果，禁止用ASCII字符或有符号数表示（害人害己）。

·**分析执行结果**

分析执行结果的任务是判断仿真结果正确性，分析错误原因。整个过程与C++程序的调试过程完全相同。

判断仿真结果正确性是基于理论结果进行的。理论结果在编写测试程序时，根据程序执行过程预先写好。判断时针对每个时钟周期的测试结果进行，只要与理论结果不同，就肯定有错误（理论结果错、设计方案错或实现电路错）。

注意：为了减少分析工作量，取指令过程分析只需要针对首条指令进行；指令执行过程分析（针对每个时钟周期）只需要针对各个数据路径不同的指令进行，数据路径相同、部件控制信号不同的所有指令，只需要分析目的OPD的正确性。

分析错误原因是基于实现电路、测试结果进行的。关键是找到错误点，错误类型有指令执行过程错误、指令执行结果错误2种。

指令执行过程错误的特征是同类指令的错误现象相同，分析方法是对比图2.6、图2.3、设计方案、仿真结果，即可获知错误点。

指令执行结果错误的类型有理论结果错、实现电路错2种。分析方法是逐个时钟周期进行对比，关键是结果开始不同的那个时钟周期，分析步骤如下：①重新计算理论结果；②理论结果正确时，根据输入信息，从主机→CPU→数据通路→功能部件逐级进行结果分析；③发现错误部件时，对该部件进行调试（可增加调试信号），直至发现错误点或下级部件错误（需重复步骤③）。第③步除了浪费时间不说，部件/电路的仿真一个都没省掉，可见，当初认真调试每个部件/电路有多么重要。

·**修正电路错误**

修正错误很简单，修正后需要重新进行部件→数据通路或CU→主机的测试，直至结果正确。

上述的模型机主机设计过程，是从方法上阐述的，可以适用于不同的指令集。本课程的具体设计，应该以上述方法为指引，结合设计要求，完成所有的设计与实现过程，从而加深对计算机组成原理的理解与掌握。

## 附录A RISC-V单周期CPU设计

本课程可缺省CPU的中断机构设计，故CPU的设计过程包括需求分析、总体设计、数据通路设计、控制单元设计4个阶段。

下面，以8条RV32I指令为例，介绍单周期CPU的设计过程。假设设计要求与本课程的设计要求相同。

### A.1 需求分析

需求分析的目标是形成CPU的设计需求，包括功能需求、内部结构需求、外部接口需求，分析的内容主要为CPU设计要求、指令系统。

本例支持的8条RV32I指令如表A.1所示。

表A.1 RV32I的7条指令功能

|  |  |  |
| --- | --- | --- |
| 指令名 | 指令功能 | 功能说明 |
| 加法add | rd←(rs1)＋(rs2) |  |
| 减法sub | rd←(rs1)－(rs2) |
| 比较slt | rd←(rs1)＜有(rs2)? 1:0 | 1表示值1、不是全1 |
| 按位或ori | rd←(rs1)|ZExt(imm) | imm为12位立即数 |
| 取数lw | rd←M[(rs1)＋SExt(imm)] | SExt、ZExt表示符号扩展、零扩展  imm为12位偏移量 |
| 存数sw | M[(rs1)＋SExt(imm)]←(rs2) |
| 分支beq | if((rs1)＝(rs2)) PC←(PC)+SExt(imm<<1) | imm为12位偏移量 |
| 跳转jal | rd←(PC)+4, PC←(PC)+SExt(imm<<1) | imm为20位偏移量 |

（1）分析CPU设计要求，可以得到如下分析结果：

·CPU的内部结构中，中断机构缺省，MMU采用实地址存储管理方式，BIU需划分为IBIU、DBIU，部件间互连采用专用结构，CU为硬布线控制器。

·CPU的外部接口中，包括复位信号、时钟信号，IBIU及DBIU的数据引脚均为32位、地址引脚≤32位（数据位数＞8位）、控制引脚包含读/写/时钟3根信号线。

（2）分析RV32I指令集结构及表A.1的指令功能，可以得到如下分析结果：

·数据类型支持32位有/无符号整数，采用定点格式的补码/无符号编码表示；

·指令功能操作支持32位加法、减法、按位或、有符号小于比较、等于比较运算，支持GPRs、MEM的读/写操作；

·数据寻址支持寄存器寻址、立即寻址、基址寻址方式，地址计算操作包括32位加法运算、12位→32位符号扩展及零扩展；

·指令寻址支持隐含寻址、PC相对寻址方式，地址计算方法分别为：(PC)＋4、(PC)＋SExt(imm<<1)，操作包括32位加法、左移1位、12位及20位→32位符号扩展；

·寄存器仅含GPR，GPRs的位数为32位、个数为32个，0#GPR恒为零，一个指令周期中最多有2次32位读操作、1次32位写操作；

·存储器按字节编址、逻辑地址空间为32位，采用小端、对齐方式存放数据及指令，一个指令周期中最多有1次32位读/写操作；

·各条指令的数据路径与通路部件设计有关，数据通路部件互连时再分析。

（3）存储管理方式的分析结果是：物理地址PA＝逻辑地址LA，可直接连线实现。

（4）总清功能的分析结果是：初始化PC、时序信号形成电路，假设PC初值为零。

因此，CPU的功能需求为上述指令系统、MMU及总清功能分析结果所示的功能；CPU的内部结构及外部接口需求满足上述CPU设计要求分析结果的规定。

### A.2 总体设计

CPU总体设计的目标是组织每个CPU模块的功能及接口，包含模块划分、模块功能分配、模块接口设计3个环节。所有的设计都基于需求分析的结果进行。

#### 1．功能模块划分

由CPU设计要求分析的结果可见，中断机构缺省，MMU功能纳入数据通路，因此，所设计的CPU可划分成数据通路、CU两个模块，如图4.1所示。

#### 2．数据通路总体设计

数据通路由运算器、指令部件、MMU、BIU四个子模块组成，运算器通常由运算部件、寄存器组、状态寄存器组成，如图4.1所示。RISC-V的程序可见寄存器中不包含状态寄存器，设计时可缺省。

下面，仅分别讨论各个子模块的功能组织，其接口设计放在详细设计时讨论。

**（1）运算部件**

运算部件负责实现所需的指令功能操作、数据寻址操作、指令寻址操作。

每条指令执行过程中都需要形成下条指令地址，而单周期CPU中的部件不能复用，因此，运算部件的功能需划分为指令功能及数据寻址操作（常合称为数据操作）、指令寻址操作2个部分，不同部分的部件不能复用。

可见，指令功能及数据寻址操作的功能需求为：实现32位加法、减法、按位或、有符号小于比较、等于比较运算，实现32位加法运算、12位→32位符号扩展及零扩展。且同一指令的指令功能与数据寻址中，32位加法运算不会同时进行。

可见，指令寻址操作的功能需求为：实现(PC)＋4、(PC)＋SExt(imm<<1)，操作包括32位加法、左移1位、12位及20位→32位符号扩展。

**（2）寄存器组GPRs**

GPRs负责实现数据及地址在寄存器中的存放。

可见，GPRs的功能需求为：由32个32位寄存器组成，0#寄存器恒为零，具有2个32位读端口、1个32位写端口，按地址进行访问。其中，设置2个读端口的原因是受限于单周期CPU的一个时钟周期完成指令功能。

**（3）指令部件**

指令部件负责实现指令控制功能，包括保存指令地址、指令字在指令周期结束前保持不变。指令部件通常由PC、IR组成，ID划入CU中，如图4.1所示。

单周期CPU的每个指令周期中，通常只有3个时钟信号边沿可用作时序逻辑操作的脉冲信号，而IMEM、DMEM均为同步RAM，指令周期中的读IMEM、读/写DMEM或写PC、写GPRs已各占用一个，因此，指令部件仅由PC组成，IR须缺省。

PC的复位功能是置初值，PC的初值通常不为零，本例为零（实现简单）。

可见，指令部件的功能需求为：仅由PC组成，复位时PC值为零，指令字在指令周期结束前保持不变。

**（4）MMU**

MMU负责实现程序逻辑地址到物理地址变换，地址变换失败时发出异常请求信号。

由于MMU采用实地址存储管理方式，中断机构缺省，MEM采用哈弗结构，故MMU的功能需求为：由IMMU、DMMU组成，物理地址PA＝逻辑地址LA。

**（5）BIU**

BIU负责实现存储器及外设访问操作的中转，通常由内部端口、数据转换电路、总线逻辑电路组成，如图2.7所示。

单周期CPU的访存操作最多包含1次时序逻辑操作，MEM为同步ROM/RAM时，要求BIU的数据引脚位数＝数据通路宽度，内部端口无时序逻辑操作，总线逻辑电路的传输周期＜1个时钟周期（只能采用常规传输方式），因此，BIU的内部端口须全部缺省（类似于IR缺省），总线逻辑电路缺省。

对于BIU的数据转换电路，由于lw/sw指令只有一种访存粒度（32位）、访存粒度＝数据通路宽度，故无需变换数据位数，只需实现小端存放方式所需的格式转换。

可见，BIU的功能需求为：BIU由IBIU、DBIU组成，数据引脚均为32位，内部端口、总线逻辑电路缺省，数据转换电路实现小端存放方式所需的格式转换。

#### 3．控制单元总体设计

控制单元CU由指令译码器ID、时序信号形成电路、μOP控制信号形成电路三个子模块组成，如图2.8所示。各子模块的接口设计放在详细设计时讨论。

由表3.2的指令功能可知，ID的功能需求为：根据指令字格式中的操作码及功能码，输出8条指令的指令类型信号。

由于单周期CPU的指令周期只包含一个状态，因此，时序系统的时序信号仅包含工作脉冲，工作脉冲的状态改变与时钟信号同步，可见，时序信号形成电路的功能需求为：输出所组织时序系统的工作脉冲信号序列，复位时所有工作脉冲信号全部无效。

而μOP控制信号形成电路的功能需求为：输出当前指令执行过程所需的所有μOP控制信号。

至此，CPU的模块划分及每个模块的功能分配均已完成。

### A.3 数据通路设计

数据通路设计指数据通路的详细设计，包含功能部件设计、部件互连设计2个环节。设计基于总体设计结果、需求分析结果、CPU设计要求进行。

#### 1．功能部件设计

功能部件设计的任务是根据各模块的约定功能，设计每个功能部件（电路）。部件设计包括I/O信号组织、内部逻辑设计2个环节。

总体设计的结果是，数据通路由运算部件、GPRs、指令部件、MMU、BIU五个子模块组成，每个子模块的功能需求也已确定。

**（1）运算部件设计**

运算部件负责实现所需的指令功能及数据寻址操作、指令寻址操作。由总体设计结果可知，指令功能及数据寻址操作、指令寻址操作的部件不能复用，分别记为数据操作单元、指令地址计算单元。

1. 数据操作单元的设计

总体设计约定，数据操作单元的功能需求为：实现32位加法、减法、按位或、有符号小于比较、等于比较运算，实现32位加法运算、12位→32位符号扩展及零扩展。且同一指令的指令功能与数据寻址中，32位加法运算不会同时进行。

根据不同功能可复用器件就合并（节约成本），不能复用就分离（简化控制）的经验，数据操作单元可组织成ALU、ExtU（位扩展部件）2个部件，比较运算用ALU内部减法+判断标志来实现。

·ALU设计

ALU的功能组织为：实现32位加法、减法、按位或、有符号比较、相等比较运算，相等比较的结果用ALU输出的最低位表示。相等比较结果也可以用ALU输出ZF标志来实现，本例拟使用ALU输出的最低位表示。

ALU的I/O信号组织为：数据输入A和B（各32位），数据输出Out（32位），操作控制Op（3位）。ALU的功能表约定如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Op | 000 | 001 | 010 | 100 | 101 |
| Out | A＋B | A－B | A|B | A＜有B？1 : 0 | A＝B？1 : 0 |

基于ALU的功能表，ALU的内部逻辑组织结果如图A.1所示。其中，Subctr用于控制加减运算类型（假设0/1表示加/减），Cmpsrc用于选择比较结果（□/■表示选择信号最小值/最大值），Outsrc用于选择运算结果。



图A.1 支持8条RV32I指令的ALU组成

由功能表可见，Subctr在Op＝001、100及101时有效，Cmpsrc在Op＝101时有效， Outsrc在Op＝000或001、010、100或101时为0、1、2，逻辑函数略。

·ExtU设计

ExtU的功能组织为：12位→32位符号扩展及零扩展操作。

ExtU的I/O信号组织为：数据输入D（＝d11…d0），数据输出Q（＝q31~q0），操作控制Op（1位）。ExtU的功能表约定为：Op＝0时为零扩展，即Q＝0…0 d11…d0；Op＝1时为符号扩展，即Q＝d11…d11 d11…d0。

基于ExtU的功能表，ExtU的内部逻辑为：逻辑函数略。

1. 指令地址计算单元的设计

总体设计约定，指令地址计算单元的功能需求为：实现(PC)＋4、(PC)＋SExt(imm<<1)，操作包括32位加法、左移1位、12位及20位→32位符号扩展。

指令地址计算单元通常组织成一个部件（记为ACU），因此，ACU的功能组织为：实现(PC)＋4、(PC)＋SExt(imm<<1)。

ACU的I/O信号组织为：地址输入PC（32位）、偏移量imm（25位），地址输出NPC（32位），操作控制Jmp、Brn、Cond（共3位）。imm 为25位的原因是分支及跳转指令格式中12位imm及20位imm的位置不完全重合，需取并集。ACU的功能表约定为：Jmp=1时NPC＝(PC)＋SExt(imm20<<1)，Brn=1且ZF=1时NPC＝(PC)＋SExt(imm12<<1)，否则NPC＝(PC)＋4。

基于ACU的功能表，ACU的内部逻辑组织结果如图A.2所示。其中，Adder为32位加法器，SL1&SExt为<<1位后符号扩展逻辑（无需封装成器件），选择逻辑略。



图A.2 指令地址计算单元的组成

**（2）GPRs设计**

GPRs负责实现数据及地址在寄存器中的存放。

总体设计约定，GPRs的功能需求为：由32个32位寄存器组成，0#寄存器恒为零，具有2个32位读端口、1个32位写端口，按地址进行访问。

GPRs通常组织成一个部件。GPRs的功能组织为：实现上述功能需求。其中，寄存器通常由D触发器组成，读操作为组合逻辑操作，仅写操作为时序逻辑操作。

GPRs的I/O信号组织为：读地址rA和rB（5位），数据输出dA和dB（32位），写地址rW（5位），数据输入dW（32位），写操作控制Wr、Clk。GPRs的功能表约定如下：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Wr | Clk | rA | rB | rW | dW | dA | dB |
| 0 | × | a | b | × | × | GPRs[a] | GPRs[b] |
| 1 |  | c | d | g | D | GPRs[c]① | GPRs[d]① |
| 1 |  | e | f | × | × | GPRs[e] | GPRs[f] |
| 1. ：当g＝c或d时，GPRs[c]或GPRs[d]＝D | | | | | | | |

基于GPRs的功能表，GPRs的内部逻辑组织结果如图A.3所示。其中，写操作控制用控制译码器是否译码来实现，0#寄存器恒为零可采用直接接地的方式来实现。



图A.3 通用寄存器组GPRs的组成

**（3）指令部件设计**

指令部件负责实现指令控制功能，包括保存指令地址、使指令字在指令周期结束前保持不变。

总体设计约定，指令部件的功能需求为：仅包含PC，复位时PC值为零，指令字在指令周期结束前保持不变。

由于PC初值为零，PC可组织为：带清零功能的寄存器，复位功能通过使其清零来实现。若PC初值不为零，则复位功能需通过写入非零值来实现。

指令字在指令周期结束前保持不变的实现方法可以为：在适当时候写PC。单周期CPU中，写PC的时机与IMEM、DMEM是否为同步RAM有关。

单周期CPU中，指令周期通常只有3个时序信号边沿可用于时序逻辑操作，本例中IMEM、DMEM均为同步RAM，由lw指令功能可知，读IMEM、读DMEM、写GPRs操作分别占用3个时序信号边沿，写PC须放在第2个时序信号边沿进行，如图A.4(a)所示。因为此后至指令周期结束前，PC虽是新值，但读IMEM不会发生，指令字会保持不变；若写PC放在第3个时序信号边沿进行（即下条指令读IMEM时），会导致读IMEM取出的仍是当前指令（共用Clk+写PC有延迟所致）。



（a）IMEM为同步存储器时 （b）IMEM为异步存储器时

图A.4 单周期CPU中写PC的时机

相应地，IMEM为异步RAM时，写PC须放在第3个时序信号边沿（即第1个时序信号边沿）进行，读IMEM结果在写PC完成之后立即改变，如图A.4(b)所示。此后至指令周期结束前，PC一直是旧值（当前指令地址），IMEM输出的一直是当前指令，满足指令字在指令周期结束前保持不变的需求。

**（4）MMU设计**

MMU负责实现程序逻辑地址到物理地址变换，地址变换失败时发出异常请求信号。

总体设计约定，MMU的功能需求为：MMU由IMMU、DMMU组成，物理地址PA＝逻辑地址LA。

MMU的功能组织为：物理地址PA＝逻辑地址LA。

MMU的I/O信号组织为：逻辑地址LA（32位），物理地址PA（32位）。

MMU的内部逻辑组织为：PA＝LA。

**（5）BIU设计**

BIU负责实现存储器及外设访问操作的中转。

总体设计约定，BIU的功能需求为：由IBIU、DBIU组成，数据引脚均为32位，内部端口、总线逻辑电路缺省，数据转换电路实现小端存放方式所需的格式转换。

由于BIU的数据引脚均为32位，可寻址空间为32位，存储器按字节编址，故32位地址空间所需的地址引脚均为32-log232/8＝30位。且DBIU数据总线组织成单向总线，便于与Quartus支持的RAM（单向数据引脚）连接。

DBIU的I/O信号组织为：地址输入Addr（32位），数据输入DataIn（32位）、数据输出DataOut（32位），操作控制MemRd、MemWr及Clk；数据总线DBusOut和DBusIn（32位），地址总线ABus（30位），控制总线CBus（3位）。

IBIU的I/O信号组织为：数据通路侧为Addr（32位）、DataIn（32位）、IMRd及Clk，系统总线侧为DBusIn（32位）、ABus（30位）及CBus（3位）。

基于BIU的功能需求，IBIU及DBIU的内部结构如图A.5所示。



图A.5 支持lw/sw指令的IBIU及DBIU

数据转换电路中，小端存放方式的要求为：DBusOut7~0＝DataOut7~0、…、DBusOut31~24＝DataOut31~24，DataIn7~0＝DBusIn7~0、…、DataIn31~24＝DBusIn31~24，因此，其内部逻辑为：信号线直接连接。相应地，大端存放方式通过信号线按字节交叉连接来实现。

至此，功能部件设计全部完成。

#### 2．部件互连设计

部件互连设计的任务是为每条指令的数据路径建立部件连接。设计方法是逐条地增加指令所需的功能部件及信号线连接，直到所有指令处理完毕。

由功能部件设计结果可知，本例所设计的功能部件有：ALU、ExtU、ACU、GPRs、PC、IMMU、DMMU、IBIU、DBIU。

数据通路中，假设μOP需要使用的工作脉冲为P0和P1，P0的上升沿在指令周期开始/结束时，P1的上升沿在指令周期的中部，不够用时再酌情增加信号数。

专用结构中，指令的数据路径与指令功能有关。下面，就逐条地为每条指令建立数据通路，同时给出部件控制信号的名称。为了便于理解，先建立各条指令数据操作所需的数据通路，最后建立取指令及指令寻址操作所需的数据通路。

**（1）add/sub/slt指令的数据通路**

add/sub指令属于R型指令，功能为rd←(rs1)±(rs2)、rd←(rs1)＜有(rs2)? 1 : 0。

add/sub指令数据操作的数据路径为：GPRs→ALU→GPRs，数据通路如图A.6所示，其中，ALUctr控制ALU的操作类型，RegWr控制是/否进行写GPRs操作。

信号线连接时，ALU的A端应连接(rs1)，因为ALU的功能是A±B；写GPRs应安排在时钟周期结束时进行，以与lw指令统一处理，故GPRs的Clk引脚应连接P0。

**（2）ori指令的数据通路**

ori指令属于I型指令，功能是rd←(rs1) | ZExt(imm)。

ori指令数据操作的数据路径为：GPRs及ExtU→ALU→GPRs，其数据通路与add/sub指令的差别是一个源操作数不同。图A.7是支持3条指令的数据通路，在图A.6基础上增加了ori指令的数据路径。



图A.6 支持add/sub指令的数据通路 图A.7 支持3条指令的数据通路

**（3）lw/sw指令的数据通路**

lw/sw指令属于I型/S型指令，功能是rd←Mem[(rs1)＋SExt(imm)]、Mem[(rs1)＋SExt(imm)]←(rs2)，访存粒度为32位（＝GPR位数）。

lw指令数据操作的数据路径为：GPRs及ExtU→ALU→DBIU→DMEM→DBIU→GPRs，sw指令数据操作的数据路径为：GPRs及ExtU→ALU、ALU及GPRs→DBIU→DMEM，其操作数地址计算的数据路径与ori指令完全相同。图A.8是支持5条指令的数据通路，在图A.7基础上增加了DMEM操作的数据路径。



图A.8 支持lw/sw指令的数据通路

信号线连接时，由于DMEM为同步RAM，lw指令的写GPRs应该安排在时钟周期结束时进行，故读/写DMEM只能放在时钟周期中部，DMEM的Clk应连接P1。

注意，DMEM不属于CPU内部的数据通路，图中画出的目的是便于理解指令功能。

**（4）beq指令的数据通路**

beq指令属于B型指令，指令功能是PC←((rs1)＝(rs2))? (PC)＋SExt(imm<<1) : (PC)＋4。

beq指令数据操作的数据路径为：GPRs→ALU，其数据通路与add/sub指令完全相同。beq指令指令寻址操作的数据路径为：PC及ALU→ACU→PC，其数据通路增设PC、ACU部件即可。

**（5）jal指令的数据通路**

jal指令属于J型指令，指令功能是rd←(PC)+4，PC←(PC)+SExt(imm<<1)。

jal指令数据操作的数据路径为：PC→ALU→GPRs，其数据通路与add/sub指令的差别是源操作数不同（为PC及4）。beq指令指令寻址操作的数据路径为：PC及IBIU→ACU→PC，其数据通路与beq指令完全相同。

**（6）取指令及指令寻址操作的数据通路**

取指令及指令寻址操作是每条指令执行都必需的，故单独进行讨论。

取指令操作的功能为读出IMEM[(PC)]，由于缺省IR，IMEM的输出在指令周期结束前需保持不变。取指令操作的数据路径为：PC→IBIU→IMEM→IBIU，其数据通路需增设IBIU及PC。

信号线连接时，读IMEM应放在指令周期开始时进行、写PC应放在指令周期中部进行，这是由IMEM为同步RAM、CPU为单周期所决定的，故IBIU的Clk应连接P0，PC的Clk应连接P1；写PC无需设置控制信号，因为每个时钟周期都要写PC；IBIU的读IMEM信号一直使能，因为每个时钟周期都要写读IMEM。

指令寻址操作的数据路径为：PC及IBIU、ALU→ACU→PC，其数据通路仅需增设ACU，并连接相应信号源。

在图A.8基础上，增加beq、jal指令的数据路径、取指令操作及指令寻址操作的数据路径，可得到支持8条指令的完整的单周期数据通路，如图A.9所示。



图A.9 支持6条指令的单周期数据通路

至此，数据通路设计全部完成。可见，数据通路的接口信号包括：IBIU及DBIU的外部接口信号，复位信号，时序信号形成电路产生的2个工作脉冲信号，μOP控制信号形成电路产生的10个μOP控制信号，输出到ID的指令操作码信号。

数据通路设计的正确性，应通过组织各条指令的执行过程来进行验证。

#### 3．指令执行过程的组织

指令执行过程组织的任务是验证所设计的数据通路能否满足指令执行过程、指令约定功能的要求，组织结果可用作控制单元的设计需求。组织方法是给出每条指令执行过程所需的μOP控制信号序列，若能够给出，则说明所设计的数据通路是正确的。

基于图A.9的数据通路，单周期CPU的指令执行过程组织如下：

①时钟周期开始时，用(PC)作为地址从IMEM中取出当前指令；

②ID自动进行指令译码，CU自动产生当前指令的所有μOP控制信号；

③数据通路部件根据μOP控制信号实现相应功能，形成下条指令地址并写入PC；

④时钟周期结束时（即下个时钟周期开始时），转①。

由于单周期CPU中的部件不能复用，因此，每条指令执行过程中的所有μOP控制信号（μOPCmd）只有一种状态，即每条指令执行过程的μOPCmd序列只有一个步骤。

基于图A.9的数据通路，可以画出每条指令的数据路径；在此基础上，根据相关部件的功能表，可以得到各条指令执行过程所需的μOP控制信号，如表A.2所示。

表A.2 指令执行过程的μOP控制信号组织

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令 | IsJmp | IsBrn | Extctr | ALUAsrc | ALUBsrc | ALUctr | RegDsrc | RegWr | MemRd | MemWr |
| add | 0 | 0 | × | 0 | 1 | 000 | 1 | 1 | 0 | 0 |
| sub | 0 | 0 | × | 0 | 1 | 001 | 1 | 1 | 0 | 0 |
| slt | 0 | 0 | × | 0 | 1 | 100 | 1 | 1 | 0 | 0 |
| ori | 0 | 0 | 0 | 0 | 0 | 010 | 1 | 1 | 0 | 0 |
| lw | 0 | 0 | 1 | 0 | 0 | 000 | 0 | 1 | 1 | 0 |
| sw | 0 | 0 | 1 | 0 | 0 | 000 | × | 0 | 0 | 1 |
| beq | 0 | 1 | × | 0 | 1 | 101 | × | 0 | 0 | 0 |
| jal | 1 | 0 | × | 1 | 2 | 000 | 1 | 1 | 0 | 0 |

注意，①与指令数据路径无关的μOP控制信号中，组合逻辑部件控制信号可以为任意值(×)，时序逻辑部件控制信号必须为无效值，如RegWr、MemWr、MemRd；②单周期CPU中，无须设置End信号，因为它每个时钟周期都有效。

由表A.2可见，基于图A.9的数据通路，8条指令执行过程中的所有操作都可以实现，也就是说，所设计的数据通路满足了所有指令执行过程的操作需求。

注意，数据通路电路、指令执行过程组织的正确性还需要通过电路仿真来验证。

### A.4 控制单元设计

控制单元设计指控制单元的详细设计，包含状态转换图形成、时序系统组织、ID设计、时序信号形成电路设计、μOP控制信号形成电路设计5个环节。设计基于总体设计结果、所设计的数据通路、需求分析结果进行。

单周期数据通路的设计结果如图A.9所示，与CU相关的接口信号包括：指令操作码信号，2个工作脉冲信号，10个μOP控制信号。

#### 1．状态转换图形成

状态转换图形成的任务是得到CPU工作流程的所有状态及其转换条件。

每个不同的操作（指同时执行的μOP）可以看作为一个状态，对各条指令执行过程的状态进行汇总，即可得到状态转换图。

本例中各条指令的所有状态如表A.2所示，每行为一个状态，汇总后得到的指令执行过程状态转换图如图A.10所示。其中，每个状态中未列出的μOPCmd都是无效的。



图A.10 单周期CPU的指令执行过程状态转换图

本例中，由于不支持异常及中断处理，图A.10就是CPU工作流程的状态转换图，即控制器的设计需求（应用需求）。

#### 2．时序系统组织

时序系统组织的任务是确定时序信号的个数、时序信号序列的个数及其组成、时序信号的定时方式。

总体设计结果约定，时序系统的时序信号只包含工作脉冲，时序信号序列只有一种，即输出所有的工作脉冲信号信号。

组织时序信号的个数时，由图A.9可见，工作脉冲信号需要2个。

组织时序信号序列时，工作脉冲信号序列只有1种，工作脉冲信号序列的组成为：2个工作脉冲信号轮流有效。

组织时序信号的定时方式时，由于设计要求为μOP采用同步方式定时，即工作脉冲信号的循环周期＝1个时钟周期，本例的时钟周期宽度＝LDUR指令的数据通路时延。

#### 3．ID设计

ID设计的任务是产生所有指令的操作类型及寻址方式信号。

总体设计结果约定，ID的功能需求为：根据指令字格式中的操作码及功能码，输出8条指令的指令类型信号。

ID的功能组织为：实现上述功能需求。

ID的I/O信号组织为：输入为操作码op（17位），输出为指令类型Iadd/Isub/Islt/Iori/ Ilw/Isw/Ibeq/Ijal。

ID的内部逻辑为：由译码器+门电路组成，或直接由门电路组成。

#### 4．时序信号形成电路设计

时序信号形成电路设计的任务是实现时序系统。

本例的时序系统组织结果是，工作脉冲信号有2个；工作脉冲信号序列有1种，信号序列的组成为：2个工作脉冲信号轮流有效。

总体设计结果约定，时序信号形成电路的功能需求为：输出所组织时序系统的工作脉冲信号序列，复位时所有工作脉冲信号全部无效。

时序信号形成电路的功能组织为：复位时工作脉冲信号全部无效，工作时按序输出工作脉冲信号序列中的每个工作脉冲信号，工作脉冲信号的循环周期为1个时钟周期。

时序信号形成电路的I/O信号组织为：复位信号Clr，时钟信号CLK，工作脉冲信号P0及P1。由于要求μOP采用同步方式定时，故图2.9种的控制方式、操作状态信号缺省。

时序信号形成电路的内部逻辑由定序逻辑、定时逻辑组成，两者的接口信号为节拍脉冲信号CP。组织定序逻辑时，工作脉冲的状态用2个门电路表示；工作脉冲的下一状态产生函数用当前状态函数表示。当前状态函数中需包含复位及启动逻辑，即复位时全部为无效，复位后P0首先有效，因此，其逻辑表达式分别为：P0＝CP·、P1＝·。组织定时逻辑时，由于μOP采用同步方式定时，故其内部逻辑可为CP＝CLK。

注意，通常用CLK的上升沿表示指令周期的开始；为使Clr的改变与CP同步，Clr应从触发器输出，触发器由CP控制。

#### 5．μOP控制信号形成电路设计

μOP控制信号形成电路设计的任务是实现所有μOP控制信号的有效逻辑。

总体设计结果约定，μOP控制信号形成电路的功能需求为：输出当前指令执行过程所需的所有μOP控制信号。

μOP控制信号形成电路的功能组织为：实现上述功能需求。

μOP控制信号形成电路的I/O信号组织为：指令类型信号Iadd/Isub/Islt/Iori/Ilw/Isw/ Ibeq/Ijal（8个），μOP控制信号IsJmp/IsBrn/Extctr/ALUAsrc/ALUBsrc/ALUctr/RegDsrc/ RegWr/MemRd/ MemWr（10个）。

μOP控制信号形成电路的内部逻辑由各个μOP控制信号的有效逻辑组成，不同μOP控制信号之间没有关联。

形成所有μOP控制信号的有效逻辑需要2个步骤：填写μOPCmd使用时间表、形成各个μOPCmd的逻辑表达式。

·填写μOPCmd使用时间表

由于状态转换图中的状态转入条件仅为指令类型、不含时间戳，故μOPCmd使用时间表可以组织为1行、10列，将每个状态的转入条件（指令类型），填入该状态的各个有效μOPCmd对应的单元格中，得到的结果如表A.3所示，表中的+表示“或者”。

表A.3 所有μOPCmd的使用时间表

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | IsJmp | IsBrn | Extctr | ALUAsrc | ALUBsrc | ALUctr | RegDsrc | RegWr | MemRd | MemWr |
| T0 | Ijal | Ibeq | Ilw+Isw | Ijal | b1:Ijal  b0:Iadd  +Isub+Islt+Ibeq | b2:Islt+Ibeq  b1:Iori  b0:Isub | Iadd+Isub+Islt+Iori  +Ijal | Iadd+Isub  +Islt+Iori  +Ilw+Ijal | Ilw | Isw |

·形成所有μOPCmd的逻辑表达式

对表A.3的每一列进行汇总、逻辑化简，即可得到各个μOPCmd的逻辑表达式，即各个μOP控制信号的有效逻辑。本例的各个μOPCmd的逻辑表达式如表A.3所示。

最后，将所设计的指令译码器ID、时序信号形成电路、μOP控制信号形成电路连接起来，就形成了完整的CU，如图A.11所示。



图A.11 支持8条RV32I指令的单周期CU

最后，将数据通路和CU连接起来，CPU的设计全部完成。

## 附录B 主存的设计与连接

### B.1 主存的设计

主存通常组织为一个基本模块，主存的设计包括需求分析、模块设计2个环节。

#### 1．需求分析

需求分析的目标是形成主存的设计需求，包括数据引脚及操作需求，分析内容主要为CPU/总线的接口要求（引脚设置及传输协议）。

附录A所设计CPU的外部接口及传输协议为：IBIU的地址引脚为30位，数据引脚为单向32位，控制引脚包含读/写/时钟3根信号线；DBIU地址引脚为30位，数据引脚为2×单向32位，控制引脚包含读/写/时钟3根信号线；访存过程＜1个时钟周期，采用常规传输方式（1个数据/次）。

由于主存直接与CPU连接，主存的数据引脚位数须与CPU数据引脚位数相同。由于CPU访存粒度≤数据引脚，CPU只需采用常规传输方式。因此，主存的设计需求如下：IMEM为4KB同步ROM，数据引脚为单向32位；DMEM为4KB同步RAM，数据引脚为2×单向32位；IMEM及DMEM均采用常规传输方式，访存过程＜1个时钟周期。

#### 2．模块设计

模块设计包含引脚信号组织、内部电路设计2个环节，设计基于主存的设计需求进行。

**（1）引脚信号组织**

由主存的设计需求可见，IMEM/DMEM的容量为4KB，数据引脚均为32位，采用常规传输方式，访存过程＜1个时钟周期，因此，地址引脚均为log2(4KB/32b)＝10位。

IMEM的引脚组织为：数据引脚为D31~D0、地址引脚为A9~A0、控制引脚包括片选引脚为CS、时钟引脚Clk。

DMEM的引脚组织为：数据引脚为DI31~DI0及DO31~DO0、地址引脚为A9~A0、控制引脚包括片选引脚为CS、写使能引脚WE、时钟引脚Clk。

**（2）内部电路设计**

由于IMEM/DMEM采用常规传输模式，访存过程＜1个时钟周期，因此，IMEM、DMEM可由单体存储器组成。当支持多种访问粒度时，需由并行访问方式的多体交叉存储器组成，并设置数据掩码引脚。

### B.2 主存的连接

由于主存容量可以选配，主存与其他部件的连接，通常需要通过连接电路来实现。连接电路的设计需要基于CPU外部接口及传输协议、主存引脚信号进行。假设主存安排在CPU可寻址空间的低端。

本课程中，主存-CPU连接电路设计时，CPU侧的信号如图A.5所示，主存侧的信号如B.1节所示，其内部逻辑包括数据线、地址线、控制线的连接逻辑，计算机组织与结构课程中已详细讨论过，不再赘述。

将所设计的CPU、主存，通过主存-CPU连接电路连接起来，就完成了模型机主机的设计，如图B.1所示。



图B.1 模型机主机的基本组成

注意，主存片选信号的有效逻辑应由连接电路实现，主存的时钟信号应由CPU（连接主存的部件）提供。

## 附录C 支持不同访存粒度的BIU设计

BIU由内部端口、数据转换电路、总线逻辑电路组成，如图2.7所示。其中，内部端口用于实现地址及数据的缓冲，如MAR、MDR；数据转换电路用于实现数据存放方式、不同访存粒度所需的格式转换，如大端/小端、访存粒度为8/16/32位；总线逻辑电路用于实现数据的传输控制，如总线传输协议的实现。

访存粒度可以小于、等于、大于数据总线宽度，当访存粒度＞数据总线宽度时，BIU的数据转换电路内部需设置缓冲队列，总线传输需采用突发传输方式，数据转换电路的组成较为复杂，此处暂不讨论，仅讨论访存粒度≤数据总线宽度的BIU设计。

支持不同访存粒度的BIU中，只有数据转换电路的组成与访存粒度有关，其基本组成如图C.1所示。其中，EndianT用于实现数据存放方式所需的数据格式转换，ReadT、WriteT用于实现多种访存粒度（≤数据总线宽度）所需的数据格式转换。



图C.1 支持多种访存粒度的BIU

BIU中，内部端口、总线逻辑电路的设计方法，与附录A完全相同，不再赘述。下面仅讨论数据转换电路的设计内容，且假设数据采用对齐方式存放。

**（1）EndianT的组织**

数据存放有大端、小端2种方式，每种方式存、取时的字节顺序转换要求是相同的。因此，读、写操作的数据端序处理，可以用功能相同的EndianT来实现，如图C.1所示。

分析大端、小端存放方式可见，大端方式的EndianT内部逻辑为：输出信号＝输入信号字节次序的倒序，小端方式的EndianT内部逻辑为：输出信号＝输入信号。

**（2）WriteT的组织**

支持多种访存粒度时，BIU外部须设置数据掩码引脚（如图C.1中的DataMask），指明数据线上哪些信号有效；存储器需由多体交叉存储器组成，同样需设置数据掩码引脚，以实现存储体的写控制，如图2.10所示。

图C.1中DataLen表示访存粒度，如00/01/10表示长度为8/16/32位，则DBus为32位，DataMask为4位。DataOut＝aabbccdd时的WriteT真值表如下：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| DataLen | 00 | | | | 01 | | 10 |
| PA1~0 | 00 | 01 | 10 | 11 | 00 | 10 | 00 |
| WrData | \*\*\*\*\*\*dd | \*\*\*\*dd\*\* | \*\*dd\*\*\*\* | dd\*\*\*\*\*\* | \*\*\*\*ccdd | ccdd\*\*\*\* | aabbccdd |
| DataMask | 0001 | 0010 | 0100 | 1000 | 0011 | 1100 | 1111 |

即PA1~0＝*x*时，WriteT的功能可表示为：

DataLen＝00时，DataMask＝0001<<*x*，WrData8\**x*+7 ~8\**x*＝DataOut7~0；

DataLen＝01时，DataMask＝0011<<*x*，WrData8\**x*+15 ~8\**x*＝DataOut15~0；

DataLen＝10时，DataMask＝1111<<*x*，WrData31 ~0＝DataOut31~0。

WriteT的内部逻辑组织时，可通过选择器实现，或通过选择器＋移位器实现。

**（3）ReadT的组织**

支持多种访存粒度时，BIU外部无需设置数据掩码引脚，通过访存地址PA1~0可提取有效信息，并进行与数据通路宽度匹配的数据位数扩展；存储器无需设置数据掩码引脚。

图C.1中DataLen表示访存粒度，例如00/01/10表示长度为8/16/32位，则DBus为32位。若RdData＝aabbccdd，则ReadT的真值表如下：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| DataLen | 00 | | | | 01 | | 10 |
| PA1~0 | 00 | 01 | 10 | 11 | 00 | 10 | 00 |
| DataIn | 0000dd | 000000cc | 000000bb | 000000aa | 0000ccdd | 0000aabb | aabbccdd |

即PA1~0＝*x*时，ReadT的功能可表示为：

DataLen＝00时，DataIn7~0＝RdData8\**x*+7 ~8\**x*，DataIn31~8＝000000；

DataLen＝01时，DataIn15~0＝RdData8\**x*+15 ~8\**x*，DataIn31~16＝0000；

DataLen＝10时，DataIn31~0＝RdData31~0。

ReadT的内部逻辑组织时，可通过选择器实现。