|  |
| --- |
|  |
| **计算机组成课程设计** |
| **(2021版)** |
|  |
| **任国林 编** |
|  |

**东南大学计算机科学与工程学院**

|  |
| --- |
|  |

**目 录**

[1 课程的目的和要求 1](#_Toc64544844)

[1.1 课程目标 1](#_Toc64544845)

[1.2 设计内容要求 1](#_Toc64544846)

[1.3 设计报告要求 1](#_Toc64544847)

[2 计算机组成原理回顾 3](#_Toc64544848)

[2.1 计算机组成及工作过程 3](#_Toc64544849)

[2.2 CPU组成及工作原理 5](#_Toc64544850)

[2.3 主存组成及其连接 10](#_Toc64544851)

[3 ARMv8 A64指令系统介绍 12](#_Toc64544852)

[4 模型机主机的设计过程 17](#_Toc64544853)

[4.1 需求分析 17](#_Toc64544854)

[4.2 总体设计 18](#_Toc64544855)

[4.3 数据通路设计与实现 20](#_Toc64544856)

[4.4 控制单元设计与实现 23](#_Toc64544857)

[4.5 CPU及主机实现 26](#_Toc64544858)

[4.6 主机测试 27](#_Toc64544859)

[附录A MIPS32单周期CPU设计 30](#_Toc64544860)

[A.1 需求分析 30](#_Toc64544861)

[A.2 总体设计 31](#_Toc64544862)

[A.3 数据通路设计 33](#_Toc64544863)

[A.4 控制单元设计 40](#_Toc64544864)

[附录B MIPS32多周期CPU的设计 44](#_Toc64544865)

[B.1 需求分析 44](#_Toc64544866)

[B.2 总体设计 44](#_Toc64544867)

[B.3 数据通路设计 46](#_Toc64544868)

[B.4 控制单元设计 50](#_Toc64544869)

[附录C 主存的设计与连接 55](#_Toc64544870)

[C.1 主存的设计 55](#_Toc64544871)

[C.2 主存的连接 56](#_Toc64544872)

[附录D 支持不同访存粒度的BIU设计 57](#_Toc64544873)

## 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ARMv8 A64指令集的模型机主机。计算机的主机由CPU及主存构成，本课程的设计不涉及异常处理、存储管理、外设及总线等内容，以强化核心内容、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上进行。

### 1.2 课程设计要求

本课程的总体要求是设计并实现支持ARMv8 A64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14条ARMv8 A64指令，MMU采用实地址存储管理模式，不支持异常及中断处理。支持的指令分别为ADD、SUB、ADDS、SUBS、ADDI、SUBI、AND、LSLV、ASRV、MOVZ、LDUR、STUR、CSEL、B.cond。

（2）CPU中，指令周期为一个时钟周期；存储器采用哈佛结构；数据通路采用专用结构（有称点点结构）；BIU的数据引脚为64位，可寻址空间为32位，控制引脚包含读/写/时钟3根信号线，访存时延＜0.5个时钟周期；CU采用硬布线方式实现，μOP采用同步方式定时。

（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。

（4）采用原理图方式（非Verilog HDL编程方式）实现所设计的电路。

（5）采用功能仿真方式进行所设计电路的调试。

有能力的学生可以超过基本要求，设计功能更强、性能更好的CPU，以便获得更好的成绩，如CPU的指令周期为多个时钟周期、支持STURB指令等。

### 1.3 设计报告要求

本课程的考核包括设计验收、设计报告2个环节，设计验收通过验收所实现的设计结果，考核学生的动手实践能力，设计报告通过叙述设计结果的分析与设计过程，考核学生的理论基础、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题目、设计目标、设计成果、CPU设计及实现、主机实现与测试、验收问题处理、设计总结等。相关要求如下：

（1）设计目标指准备实现的内容，包括CPU功能、CPU结构、主存组成的设计要求。

（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即可，如实现了支持14条ARMv8 A64指令的单周期CPU及主机；设计特色指超出基本要求的内容，如多周期CPU、PC初值不为零等，只完成基本要求的就不写。

（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4个环节。需求分析、总体设计、详细设计、电路实现的叙述应该环环相扣。每个环节的分析与设计需要有明确的结果，要重视分析及设计过程的叙述，这是报告的考核重点之一。各个环节的实现只需用电路图来反映结果，因为设计验收时已进行过考查，杜绝无设计过程的电路图解释。电路仿真要重视仿真数据组织、仿真结果分析的叙述，这是报告的考核重点之二。

（4）主机实现与测试包括CPU实现、主存设计与实现、主机实现、主机测试4个环节。主机测试中，应注意执行环境与测试程序、仿真结果间的关联，测试程序应测试正确实现、不是可以实现，测试结果的分析应关注程序执行过程，这是报告的考核重点之三。

（5）验收问题处理应先列出验收时的主要问题，每个问题的处理结果有未解决、有方案、已解决3种，有方案的问题应放在此处说明处理办法，已解决的问题应放在相应环节说明（相关结果为改正后的结果）。

（6）设计总结包括工作小结、设计体会，工作小结中应包含小组成员的工作分工。

报告撰写时，文字尽量简洁，图表尽量紧凑（字母能看清且小于正文字号、空白尽量少、信息尽量多）。注意，设计报告通过叙述如何完成设计要求来反映个人水平，抄写讲义中相关基本原理及设计方法的叙述毫无意义，重点是如何运用这些基本原理及设计方法，去分析问题、解决问题，完成设计任务。

## 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，循环可通过改变指令地址（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。改进主要有多种存储器共存、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，所需的支持是用层次结构的存储系统代替单一的存储器，存储系统至少由主存、辅存组成。因而，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（执行程序）与数据传送的并行，所需的支持是存储系统中增设Cache、I/O系统支持DMA方式传送数据。从而，CPU可以通过访问Cache来执行程序，而不与I/O设备争用主存。

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。计算机通常采用总线方式进行部件互连，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，命名方法也较多，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，主存的结构类型有冯·诺依曼结构、哈佛结构2种。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（其结果与指令内容有关）。

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，因而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，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，程序MEM（程序中所使用的存储器）的存储单元长度与主存单元长度相同，每个程序的程序MEM都从零开始编址，故程序地址又称为逻辑地址。由于程序中指令的执行顺序是在编程时约定的，即指令在程序MEM（非主存）中的执行顺序，因此，程序执行时必须按逻辑地址访问主存，才能保证程序执行的正确性。可见，按逻辑地址访问主存有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，而每条指令实现的功能又都是指令系统约定的指令功能，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、指令译码器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，其基本结构如图2.4所示。注意，MMU属于存储系统部件，非CPU功能部件，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、中断机构的功能与指令功能无关，但与数据通路有数据交互，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，指令部件需单独设置BIU，即BIU由IBIU（指令BIU）、DBIU（数据BIU）组成。并且，IBIU仅与PC、IR连接，IR不连接数据通路其他部件，PC还可连接ALU等部件，以实现指令地址计算时的部件复用。

#### 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，其中仅响应环节由硬件（中断机构）实现，其余环节通过执行程序实现，因此，CPU的工作流程由循环的指令周期、中断周期组成，中断周期仅在有中断请求时存在，如图2.5所示。另外，中断请求的检测也是由硬件实现的。

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，指令周期由若干个时钟周期组成，单周期CPU的指令周期为一个时钟周期，多周期CPU的指令周期为多个时钟周期。

#### 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）。

由于CPU内部与外部的接口是BIU，故CPU内部的数据通路部件还需包含BIU，不包含IMEM及DMEM。

BIU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存放方式、数据位数扩展（访存粒度＜数据宽度时）所需的格式转换，如大端/小端、A64中LDURB指令要求所取8位数据转换为64位数据；总线逻辑电路用于实现数据的传输过程，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外部实现，如电路放在BIU与数据通路结构之间，因此，图2.4的BIU中未包含数据转换电路。

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中μOP的功能组织、指令执行过程的数据路径有关。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（有称点点结构）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，部件的输入端可以直接与输出端连接，同时可以传送多个数据，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基于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

主存为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）、主存地址空间大小均由计算机系统结构确定。由于硬件通常支持多种数据类型，不同数据类型的长度有所不同，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较长数据类型的长度，即*m*×主存单元长度（*m*≥1）。

#### 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，当访问粒度≤数据引脚位数、访问粒度＞1种时，还需要设置数据掩码引脚。

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，则当*W*＝*w*时，主存可由单体存储器组成，存储体的参数为2*k*×*w*，或者由交叉访问方式的多体交叉存储器组成（性能会更好）；当*W*＝*m*×*w*（*m*＞1）时，主存须由并行访问方式的*m*体交叉存储器组成，存储体的参数为(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）。



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（写使能线和片选线）。

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 ARMv8 A64指令系统介绍

ARMv8-A是一种64位处理机架构，支持AArch64和AArch32两种执行状态。处理器的执行状态定义了处理器的执行环境，包括寄存器宽度，所支持指令集，异常模型、虚拟存储系统、编程模型等方面。AArch64和AArch32间的状态转换只能通过发生异常或系统复位来实现。

AArch64支持A64指令集，通用寄存器为64位；AArch32支持ARMv7的A32指令集和T32指令集，通用寄存器为32位。

AArch64支持4级异常等级（EL0～EL3），数字越大、执行特权越高。例如，应用程序在EL0级（非特权级）执行，操作系统内核程序在EL1级执行，虚拟机管理器在EL2级执行，安全监控程序在EL3级执行。不同异常等级间的转换，可在异常事件处理时实现，每个异常都有一个目标异常等级（处理时的异常等级）。

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A64指令系统的相关内容。

#### 1．数据表示

A64支持整数、浮点数、向量3种数据表示方法，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、64位。因此，机器字长为64位。

浮点数采用浮点格式表示，浮点格式采用IEEE 754标准，类型有半精度、单精度、双精度三种，数码长度分别为16位、32位、64位。

向量采用向量格式表示，元素长度有字节、半字、字、双字、四字5种类型，数据长度分别为64位、128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，以提高存储效率；实际上，由于算术运算和逻辑运算可以复用器件，多数系统就缺省了逻辑数的表示（用整数代替），逻辑运算通过对整数进行按位的逻辑操作来实现，这是一种简化约定。

#### 2．操作数存放

A64的操作数可存放在指令寄存器、寄存器、存储器中，指令只能存放在存储器中。

**（1）A64的存储器**

A64的存储器按字节编址，逻辑地址空间为48位（64位地址中的高16位未使用）。

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可采用小端/大端、对齐/非对齐方式。当前的端序类型由系统控制寄存器SCTLR\_EL1.E0E决定，默认为小端方式；系统要求采用对齐方式，系统控制寄存器SCTLR\_EL1.A决定，指令不按对齐方式存放/访问数据时是否会产生异常（对齐故障），LDXR/STXR等独占访问指令除外。应用中，整数、浮点数常采用小端、对齐方式存放，向量可采用小端/大端、对齐/非对齐方式存放（增强指令功能）。

**（2）A64的寄存器**

A64的寄存器有4类：通用寄存器、状态寄存器、特殊功能寄存器、系统寄存器组成。前3类中的重要寄存器如表3.1所示。

表3.1 ARMv8 A64的重要寄存器

|  |  |  |
| --- | --- | --- |
| 类型 | 位数 | 描述 |
| X0~X30、XZR | 64 | 通用寄存器，低32位可用作32bit寄存器（W0-W30、WZR）；  XZR（编号为31）恒为零，表示立即数0，常用硬连线实现；  X30又称过程链接寄存器（PLR或LR），用于保存过程调用返回地址 |
| V0~V31 | 128 | SIMD和浮点寄存器，用于浮点运算和向量操作，  低位可用作8位、16位、32位、64位寄存器 |
| PSTATE |  | 程序状态寄存器，是保存当前程序状态的一组寄存器统称，  可访问的寄存器有：PSTATE.{NZCV, DAIF, EL, SPSel, SS, IL, nRW等}  NZCV—条件标志，包含负(N)、零(Z)、进借位(C)、溢出(V)标志位  DAIF—异常屏蔽位，包含调试(D)、系统错误(A)、IRQ中断(I)、快速中断(F)屏蔽位，值为0时允许进行异常处理  EL—当前异常等级，SPSel—SP选择方案(0为SP\_EL0)  SS—单步执行标志，IL—非法执行状态标志(有非法异常时为1)  nRW—当前执行状态(0为A64) |
| PC | 64 | 程序计数器，保存下条指令地址 |
| SP\_Elx | 64 | 堆栈指针寄存器（x=0～3），保存当前异常等级使用的堆栈指针；  若PSTATE.SPSel=1，则每个ELx选择SP\_ELx，否则均选择SP\_EL0 |
| ELR\_ELx | 64 | 异常链接寄存器，保存当前异常等级异常的返回地址；  异常转入ELi级处理时，异常返回地址保存在ELR\_ELi中 |
| SPSR\_Elx | 32 | 备份程序状态寄存器，保存异常发生时的PSTATE信息，  异常转入ELi级处理时，PSTATE信息保存在SPSR\_ELi中 |

SPSR\_ELx的结构如图3.1所示，其中，RESO为保留位。响应异常时，PSTATE.NZCV、PSTATE.DAIF、PSTATE.SS、PSTATE.IL分别保存在SPSR\_Elx的相应位中，PSTATE.EL、PSTATE.SPSel、PSTATE.nRW分别保存在SPSR\_Elx的M[3..2]、M[0]、M[4]位中。



图3.1 SPSR\_ELx的结构

A64约定，通用寄存器X0～X30可以存放32位、64位整数，即可以对X0～X30进行32位、64位访问，短数据需进行位扩展后再存放；SIMD和浮点寄存器可以存放16位、32位、64位浮点数，或64位、128位向量，向量元素的长度可以为8位、16位、32位、64位、128位。

A64还约定，所有短数据（整数/浮点数/向量）在寄存器存放时，均采用寄存器低端方案，即只使用寄存器的低位部分。

#### 3．寻址方式

A64支持立即寻址、寄存器寻址、基址寻址、变址寻址、PC相对寻址5种寻址方式，基址寻址、变址寻址方式又包含多种变种，其地址形成方法如表3.2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3.2 A64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为无符号数 |
| 寄存器寻址 | | Rn | OPD＝(Rn) | Rn为寄存器号 |
| 基址  寻址 | 不可扩展 | Rn及imm  或  Rn及Rm | EA＝(Rn)＋imm | imm为有符号偏移量 |
| 可扩展 | EA＝(Rn)＋XExt(imm)或  EA＝(Rn)＋XExt((Rm)) | imm为有/无符号偏移量  XExt功能为加倍或扩展 |
| 变址  寻址 | 先变址 | Rn及imm | (Rn)＝(Rn)＋imm，EA＝(Rn) | imm为有符号数 |
| 后变址 | Rn及imm | EA＝(Rn)，(Rn)＝(Rn)＋imm | imm为有符号数 |
| PC相对寻址 | | imm | EA＝(PC)＋imm | imm为有符号数 |

由表3.2可见，PC相对寻址与相对寻址仅命名不同，imm运算前都需要进行位扩展。

A64中，操作数寻址支持除PC相对寻址外的所有寻址方式，指令寻址支持PC相对寻址、寄存器寻址、隐含寻址等方式。

#### 4．指令格式

A64采用32位定长指令字结构，指令格式有6种，如图3.2所示。

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 31 21 | | | | 20 16 | 15 10 | | 9 5 | 4 0 |
| R型 | opcode | | | | Rm | shamt | | Rn | Rd |
|  | 31 22 | | | 21 10 | | | | 9 5 | 4 0 |
| I型 | opcode | | | ALU\_Imme | | | | Rn | Rd |
|  | 31 21 | | | | 20 12 | | 11 10 | 9 5 | 4 0 |
| D型 | opcode | | | | DT\_Addr | | op2 | Rn | Rt |
|  | 31 26 | 25 0 | | | | | | | |
| B型 | opcode | BR\_Addr | | | | | | | |
|  | 31 24 | | 23 5 | | | | | | 4 0 |
| CB型 | opcode | | COND\_BR Addr | | | | | | Rt |
|  | 31 21 | | | | 20 5 | | | | 4 0 |
| IM型 | opcode | | | | MOV\_Imme | | | | Rd |

图3.3 ARMv8 A64的指令格式

其中，opcode为操作码，op2等为功能码，Rn、Rt、Rd为寄存器编号，shamt为移位位数，ALU\_Imme、DT\_Addr、BR\_Addr、COND\_Addr、MOV\_Imme为地址码的组合。

A64指令格式中，操作码采用扩展编码方式（如opcode及op2等），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。

R型格式主要用于算逻运算、移位运算、条件比较、条件选择指令，I型格式主要用于算逻运算、移位运算、数值传送、数据存取指令，D型格式主要用于存储器访问指令，B型格式主要用于无条件转移指令，CB型格式主要用于条件转移、地址传送指令，IM型格式主要用于异常产生、系统类指令。注意，调用、返回指令采用了B型、R型格式。

#### 5．指令功能

A64有150多条指令，可以分为5大类：数据处理、数据传送、分支、异常产生、系统寄存器操作，数据处理又可分为定点运算、浮点运算、SIMD运算。

表3.3列出了49条常用的A64指令功能及指令字格式。其中，ANDI、ORRI、EORI、ANDIS指令的Imm12定义与A64有所不同。

表3.3 A64常用指令字格式一览表

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 助记符 | 指令字格式 | | | | | | | | | 指令功能 |
| ADDI | 100**100**0100 | | | Imm12 | | | | Rn | Rd | Rd←(Rn) + ZExt(Imm12) |
| ADDIS | 101**100**0100 | | | Imm12 | | | | Rn | Rd | (Rd,PSTATE.NZCV)←(Rn) + ZExt(Imm12) |
| SUBI | 110**100**0100 | | | Imm12 | | | | Rn | Rd | Rd←(Rn) - ZExt(Imm12) |
| SUBIS | 111**100**0100 | | | Imm12 | | | | Rn | Rd | (Rd,PSTATE.NZCV)←(Rn) - ZExt(Imm12) |
| ANDI | 100**100**1000 | | | Imm12 | | | | Rn | Rd | Rd←(Rn) & ZExt(Imm12) |
| ORRI | 101**100**1000 | | | Imm12 | | | | Rn | Rd | Rd←(Rn) | ZExt(Imm12) |
| EORI | 110**100**1000 | | | Imm12 | | | | Rn | Rd | Rd←(Rn) ⊕ ZExt(Imm12) |
| ANDIS | 111**100**1000 | | | Imm12 | | | | Rn | Rd | (Rd,PSTATE.NZCV)←(Rn) & ZExt(Imm12) |
| LSL | 110**100**1101 | | | Immr:Imms | | | | Rn | Rd | Rd←(Rn)<<L Imm，Immr=64-Imm，Imms=63-Imm |
| LSR | 110**100**1101 | | | Immr:111111 | | | | Rn | Rd | Rd←(Rn)>>L Immr |
| ASR | 100**100**1101 | | | Immr:111111 | | | | Rn | Rd | Rd←(Rn)>>A Immr |
| MOVZ | 110**100**101:hw | | | | Imm16 | | | | Rd | Rd←ZExt(Imm16)<<(hw\*16)，hw=00~11 |
| MOVK | 111**100**101:hw | | | | Imm16 | | | | Rd | Rd←((Rd)&~(ZExt(FFFF)<<(hw\*16))) |(ZExt(Imm16)<<(hw\*16)) |
| ADD | 1000**101**1000 | | | | Rm | 000000 | | Rn | Rd | Rd←(Rn) + (Rm) |
| ADDS | 1010**101**1000 | | | | Rm | 000000 | | Rn | Rd | (Rd,PSTATE.NZCV)←(Rn) + (Rm) |
| SUB | 1100**101**1000 | | | | Rm | 000000 | | Rn | Rd | Rd←(Rn) - (Rm) |
| SUBS | 1110**101**1000 | | | | Rm | 000000 | | Rn | Rd | (Rd,PSTATE.NZCV)←(Rn) - (Rm) |
| SMULH | 1001**101**1010 | | | | Rm | 011111 | | Rn | Rd | Rd←((Rn) \* (Rm))127~64，有符号乘法 |
| UMULH | 1001**101**1110 | | | | Rm | 011111 | | Rn | Rd | Rd←((Rn) \* (Rm))127~64，无符号乘法 |
| MUL | 1001**101**1000 | | | | Rm | 011111 | | Rn | Rd | Rd←((Rn) \* (Rm))63~0，乘法 |
| UDIV | 1001**101**0110 | | | | Rm | 000010 | | Rn | Rd | Rd←(Rn) / (Rm)，无符号除法 |
| SDIV | 1001**101**0110 | | | | Rm | 000011 | | Rn | Rd | Rd←(Rn) / (Rm)，有符号除法 |
| AND | 1000**101**0000 | | | | Rm | 000000 | | Rn | Rd | Rd←(Rn) & (Rm) |
| ORR | 1010**101**0000 | | | | Rm | 000000 | | Rn | Rd | Rd←(Rn) | (Rm) |
| EOR | 1100**101**0000 | | | | Rm | 000000 | | Rn | Rd | Rd←(Rn) ⊕ (Rm) |
| ANDS | 1110**101**0000 | | | | Rm | 000000 | | Rn | Rd | (Rd,PSTATE.NZCV)←(Rn) & (Rm) |
| LSLV | 1001**101**0110 | | | | Rm | 001000 | | Rn | Rd | Rd←(Rn)<<L (Rm)，逻辑左移 |
| LSRV | 1001**101**0110 | | | | Rm | 001001 | | Rn | Rd | Rd←(Rn)>>L (Rm)，逻辑右移 |
| ASRV | 1001**101**0110 | | | | Rm | 001010 | | Rn | Rd | Rd←(Rn)>>A (Rm)，算术右移 |
| CMP | 1111**101**0010 | | | | Rm | cond:00 | | Rn | 0:nzcv | PSTATE.NZCV←(Hold(cond))?  (Rn)-(Rm) : nzcv，Hold()同B.cond |
| CMPI | 1111**101**0010 | | | | Imm5 | cond:10 | | Rn | 0:nzcv | PSTATE.NZCV←(Hold(cond))?  (Rn)-ZExt(Imm5) : nzcv |
| CSEL | 1001**101**0100 | | | | Rm | cond:00 | | Rn | Rd | Rd←(Hold(cond))? (Rn) : (Rm) |
| CSINC | 1001**101**0100 | | | | Rm | cond:01 | | Rn | Rd | Rd←(Hold(cond))? (Rn) : (Rm)+1 |
| LDXR | 1100**1**0**0**0010 | | | | 11111011111 | | | Rn | Rt | Rt←Mem[(Rn),64] |
| STXR | 1100**1**0**0**0000 | | | | Rs:011111 | | | Rn | Rt | Mem[(Rn),64]←(Rt)，Rs←(WrOK)? 1:0 |
| LDUR | 1111**1**0**0**0010 | | | | Imm9 | | 00 | Rn | Rt | Rt←Mem[(Rn)+SExt(Imm9),64] |
| LDURB | 0011**1**0**0**0010 | | | | Imm9 | | 00 | Rn | Rt | Rt←ZExt(Mem[(Rn)+SExt(Imm9),8]) |
| LDURH | 0111**1**0**0**0010 | | | | Imm9 | | 00 | Rn | Rt | Rt←ZExt(Mem[(Rn)+SExt(Imm9),16]) |
| LDURSW | 1011**1**0**0**0100 | | | | Imm9 | | 00 | Rn | Rt | Rt←SExt(Mem[(Rn)+SExt(Imm9),32]) |
| STUR | 1111**1**0**0**0000 | | | | Imm9 | | 00 | Rn | Rt | Mem[(Rn)+SExt(Imm9),64]←(Rt) |
| STURB | 0011**1**0**0**0000 | | | | Imm9 | | 00 | Rn | Rt | Mem[(Rn)+SExt(Imm9),8]←(Rt)7~0 |
| STURH | 0111**1**0**0**0000 | | | | Imm9 | | 00 | Rn | Rt | Mem[(Rn)+SExt(Imm9),16]←(Rt)15~0 |
| STURW | 1011**1**0**0**0000 | | | | Imm9 | | 00 | Rn | Rt | Mem[(Rn)+SExt(Imm9),32]←(Rt)31~0 |
| B | 000**101** | Imm26 | | | | | | | | PC←(PC)+SExt(Imm26:00) |
| BL | 100**101** | Imm26 | | | | | | | | R30←(PC)+4，PC←(PC)+SExt(Imm26:00) |
| BR | 110**101**10000 | | | | 11111 | 000000 | | Rn | 00000 | PC←(Rn) |
| CBZ | 101**101**00 | | Imm19 | | | | | | Rt | if ((Rt)=0) PC←(PC)+SExt(Imm19:00) |
| CBNZ | 101**101**01 | | Imm19 | | | | | | Rt | if ((Rt)≠0) PC←(PC)+SExt(Imm19:00) |
| B.cond | 010**101**00 | | Imm19 | | | | | | 0:cond | if (Hold(cond)) PC←(PC)+SExt(Imm19:00)  cond[3~1]表示Z/C/N/V/C·Z#/N=V/N=V·Z#/1  cond[0]=0/1表示是/否满足条件时Hold=1 |

指令字格式中，第一列为操作码，其余为地址码，Rx为通用寄存器编号（占5位），Immx为立即数或偏移量（占x位），“：”表示编码结果为两部分内容的拼接。

指令功能中，源操作数用部件内容表示，目的操作数用部件名称表示，操作类型大多用C语言运算符表示，(Rd,PSTATE.NZCV)←表示结果同时写入Rd及PSTATE.NZCV；(Rx)表示寄存器*x*的内容，Mem[x,z]表示从地址x开始的z个存储单元内容或z个存储单元，(Rx)15~0表示寄存器Rx的最低16位，SExt、ZExt分别表示符号扩展、零扩展操作，WrOK表示访存操作是否正常完成，cond[3~1]表示了8种判断条件：结果相等、结果有进位/借位、结果为负、结果溢出、无符号大于、有符号小于、有符号大于、无条件执行，判断时与PSTATE.NZCV进行比较。

可见，指令ADDIS、SUBIS、ANDIS、ADDS、SUBS、ANDS的结果会影响状态寄存器PSTATE的值；指令LDXR、STXR采用的是可扩展基址寻址方式，即EA=(Rn)+(R31)，其余存取指令采用的是不可扩展的基址寻址方式。

由表3.3可见，不同类别指令的操作码编码有一定规则，如表3.4所示。

表3.4 A64操作码主编码表

|  |  |  |  |
| --- | --- | --- | --- |
| 指令分类 | 31-29 | 28-24 | 23-0 |
| 未定义 | x x x | 0 0 x x x | x ～ x |
| 数据处理指令（立即数型） | x x x | 1 0 0 x x | x ～ x |
| 分支/异常产生/系统指令 | x x x | 1 0 1 x x | x ～ x |
| 数据传送指令（存/取） | x x x | x 1 x 0 x | x ～ x |
| 数据处理指令（寄存器型） | x x x | x 1 0 1 x | x ～ x |
| 数据处理指令（SIMD及浮点类） | x x x | x 1 1 1 1 | x ～ x |

## 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ARMv8 A64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。由于主存的组成很简单，因此，课程的主要任务是设计、实现并测试模型机主机。

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6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、CPU及主机实现、主机测试。

注意：本课程的先修课程为计算机组成原理，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的叙述比较抽象，可以先通读一遍，设计过程中再细看，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、内部结构、外部接口需求，主存的设计需求包括操作需求、引脚需求。因此，需求分析的内容包括设计要求分析、指令系统分析。

**（1）分析设计要求**

本课程的设计要求包括CPU功能、CPU结构、主存组成三个方面。

CPU功能的基本要求为支持14条A64指令（可以增加指令条数），MMU采用实地址存储管理模式，不支持异常及中断处理。由于实地址管理模式的地址变换方法是物理地址＝逻辑地址，故MMU可以缺省；又由于CPU不需要支持异常及中断处理，故中断机构可以缺省。因此，通过分析A64指令集结构及14指令的约定功能，即可形成CPU的功能需求。

CPU结构的基本要求确定了CPU的内部结构需求、外部接口需求。例如，指令周期类型（单周期/多周期）、存储器结构类型、数据通路结构类型的要求，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定时方式的要求，决定了CU的基本结构。

CPU的外部接口需求由所支持访存指令类型、BIU接口参数的要求决定。例如，存储器采用哈佛结构，要求CPU的BIU由2套引脚。

主存组成的基本要求由主存容量、引脚需求、操作需求决定，操作需求包括操作类型、访问粒度、传输模式。例如，主存的数据引脚位数须与CPU数据引脚相同，数据引脚位数大于访问粒度时需要设置数据掩码引脚，主存的传输模式须满足CPU传输协议的要求。

**（2）分析指令系统**

分析指令系统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，以及需要支持的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，这些是CPU的功能需求之一。注意，寻址方式的地址计算方法与存储器的编址单位及逻辑地址空间有关。

通过分析所支持指令功能，可以获得CPU需要支持的指令操作的类型，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，系统总清是CPU的功能需求之三，即总清时对必要部件进行初始化，保证总清结束后，CPU能正确地开始工作。

CPU的设计需求（功能/内部结构/外部接口）必须明确表示，否则后续设计无法进行。例如，数据通路的宽度取决于整数的位数，ALU的功能取决于定点操作的类型，指令地址计算部件的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；数据通路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，新增指令只认可需要增加数据通路部件的指令。

注意2：建议先实现单周期CPU，条件许可时再改为多周期CPU。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；而CPU从单周期改为多周期时，数据通路部件、ID的电路可以直接使用，数据通路的电路需要修改（动作不大），时序信号形成电路、μOP控制信号形成电路需要重新设计，所浪费时间不多。

### 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，由功能模块划分、模块功能分配、模块接口设计3个环节组成。

主存通常组织成一个基本模块，因此，其设计过程无需再进行总体设计、详细设计划分，基于主存的设计需求，直接进行模块设计（引脚信号组织、内部电路设计）即可。

#### 1．CPU功能模块划分

由CPU基本功能可知，CPU由运算器、BIU、MMU、指令部件、CU及中断机构6个部分组成，如图2.4所示，其中，MMU是放在CPU芯片中的存储系统部件。而CPU的工作流程由循环的基本操作序列组成。

为了便于实现操作控制，CPU通常划分为数据通路、控制器2个部分，数据通路包括运算器、指令部件、BIU、中断机构，控制器只由CU组成。因此，CPU的功能模块可划分为3个模块：数据通路、MMU、CU，数据通路、MMU的操作均受CU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、中断机构4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及操作（读/写）。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信息的存放。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负责实现存储器及外设访问的操作中转，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。设计时，内部端口的组织与指令周期类型、μOP的功能组织有关，数据转换电路的功能与数据存放方式、访存粒度与数据宽度关系等有关，总线逻辑电路的功能与总线传输协议有关。

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

附加寄存器负责实现其他时钟周期需要使用的操作结果的存放。设计时，各个时钟周期所需的寄存器个数，取决于各个μOP的功能组织、指令执行过程的数据路径。

**（2）MMU**

MMU的主要功能是实现程序的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。

MMU的内部逻辑与存储管理模式有关。例如，实地址管理模式时的物理地址＝逻辑地址，MMU的内部逻辑仅为一条总线，对图2.7而言，有PA＝LA。

由于要求MMU采用实地址存储管理模式，因此，设计时，MMU可以缺省，只需将BIU的逻辑地址线LA的低位与物理地址线PA连接起来即可。据此，后续设计与实现中，将不再涉及MMU。

**（3）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。

时序系统的组成受限于指令周期类型，应先确定时序系统的结构特征，以减少详细设计的复杂度。单周期/多周期CPU中，时序系统采用一级/二级时序，状态的改变与指令周期/节拍周期同步，时序信号序列有1级/2级。

ID负责实现产生指令的指令类型（操作码＋寻址方式）信号。设计时，仅需输出所支持指令的信号，可能还需输出一个异常信号（需处理非法操作码异常时）。

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。

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。

**（4）CPU总清功能的组织**

CPU的总清功能是对必要部件进行初始化，例如，PC的初值为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、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。数据通路、CU的接口设计，放在详细设计时讨论，因为他们都受数据通路组织结果的影响。

数据通路的外部接口信号包括：复位信号，BIU外部接口信号，工作脉冲信号、μOP控制信号，程序状态信号。复位信号用于实现PC的初始化；BIU外部接口信号用于主存及外设的访问，工作脉冲、μOP控制信号用于μOP的实现；程序状态信号用于表明当前指令（分支指令）需要实现的功能。

CU的外部接口信号包括：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。

本课程的设计中，MMU及中断机构模块可以缺省，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三个子模块组成，CPU总体结构如图4.1所示。由于设计要求规定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几个功能部件，目标是尽量复用部件（降低硬件成本）、提高集成度（减少互连复杂度）。组织方法是：不兼容的功能、需同时使用的功能应该用不同的部件实现，尽量增加部件的功能（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（功能比较复杂），而寄存器组、BIU通常都组织成一个部件，指令部件、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：电路的设计是在纸上进行的，电路的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、OF），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），才进行数据通路的调试。这是因为，某部件的设计或实现错误，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。例如，找出ALU中的某个错误可能需要5分钟，在调试数据通路时想找出该错误起码需要20分钟。

①组织测试数据时，功能部件、数据通路调试的目标有所不同，测试数据的组织方法就有所不同。

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！）。测试数据应能够枚举电路功能表中每个操作（功能）的输出信号组合（数字信号只需一组），以提高使用该电路的上一级电路的调试效率。例如，ALU的有符号加法运算功能测试至少需要4组数据，以反映源操作数符号与OF标志的组合；基址寻址、相对寻址的偏移量应有正有负；有符号比较应考虑OF＝0及OF＝1的情况。

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，看数据通路能否实现指令的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。由于功能部件已调试过，因此，每条指令的测试数据只需包含一个（正确的）控制信号序列、一组（任意的）数据/地址信号，分支指令需要包含2组数据。

注意2：为了减少调试工作量，数据路径相同、仅部件控制信号不同的所有指令（如add/addu/sub/subu），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（时序仿真方式无需这一步）、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：编辑数据通路波形文件时，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）；若定时逻辑中CP＝CLK，则要求时钟周期开始于CLK上升沿，即控制信号在CLK上升沿（非下降沿）改变。

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，应该对每个输入信号的变化进行分析，判断输出信号及其时序的正确性。判断是基于理论结果进行的，理论结果由设计方案、仿真数据得到。例如，所组织的指令uOPCmd序列约定了每个时钟周期需完成的操作，因此，分析数据通路仿真结果时，首先需要判断给出（输入）的控制信号序列是否正确，然后需要根据输入数据，计算出每个时钟周期的相关输出信号，与仿真结果的输出信号相比，即可实现判断。

注意：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，而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”可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”可由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。

现代计算机的时序系统常采用节拍、工作脉冲两级时序。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节拍信号循环周期的影响。

工作脉冲信号为二级时序信号，其循环周期等于1个节拍周期，因此，工作脉冲信号序列只有1种，序列的组成是每个工作脉冲信号轮流有效。

节拍信号的循环周期有定长、变长两种。信号采用定长循环周期时，节拍信号序列只有1个，序列的组成是每个节拍信号轮流有效。信号采用变长循环周期时，节拍信号序列的个数≥状态转换图的路径长度种类数，每个序列的组成是状态转换图中所有路径的组合，同时可以得到每个时序信号序列的适用条件。

时序信号的定时方式即μOP定时方式，可采用联合、同步方式。

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，为了提高性能，复杂μOP可采用同步控制方式实现，即用间隔固定的2个μOPCmd来控制，因此，时钟周期＝max{*T*基本μOP*i*}。

联合定时方式中，时钟周期＝max{*T*基本μOP*j*}，复杂μOP操作采用异步控制方式实现，该μOP的节拍周期取决于操作状态信号的时延。

**（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。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：非法指令的处理方法有当作nop指令（空操作）、产生异常2种，前者的实现方法是，ID只输出合法指令的译码信号，或者时序信号形成电路、μOP控制信号形成电路只使用合法指令的译码信号，使得非法指令不能产生有效的μOPCmd（等价于nop指令）。

**（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，如图2.9所示，接口为节拍脉冲信号CP。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门电路输出信号的产生函数表示；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。同时，CP的占空比（高电平/脉冲周期）应满足数据通路中相关工作脉冲的要求。

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，由于μOPCmd使用时间表是一张二维表，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，因此，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及主机实现

CPU及主机实现的主要内容包括CPU实现、主存设计与实现、主机实现。这3个任务之间本来没有什么关联，由于都很简单，因此，放在同一个步骤中讨论。

**（1）CPU实现**

CPU实现的任务用EDA软件实现完整的CPU电路。

本课程的设计中，MMU及中断机构可以缺省，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。实现时，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（实现的错误概率很小），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、OF等调试信号，以便查看程序执行过程及指令执行结果。

**（2）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进行，主存的设计需求包括操作需求（操作类型/访问粒度/传输模式）、引脚需求（地址个数/编址单位/数据引脚方向）。

·主存引脚信号组织

主存的数据引脚位数必须与CPU数据引脚位数相同，当写操作的访问粒度＜数据引脚位数时，还需要设置数据掩码引脚；地址引脚位数基于配置容量、数据引脚位数进行计算可得到；控制引脚包含操作控制、片选、时钟（同步存储器才需要）引脚，当支持突发传输模式时，还需增加引脚，以支持突发长度设置。

CPU的数据引脚位数＝*m*×主存单元长度，通常*m*＞1，以提高访存效率。

·主存内部电路设计

主存仅支持常规传输方式时，若写操作的访问粒度＜数据引脚位数，则主存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），否则由单体存储器组成。

主存需支持突发传输方式时，主存须由交叉访问方式的多体交叉存储器组成，存储体可由单体存储器或并行访问方式的多体交叉存储器组成，这取决于写操作的访问粒度是否小于主存数据引脚位数。

·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：Quartus II的ROM及RAM都是同步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。

**（3）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。

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。

### 4.6 主机测试

主机测试的目的是测试、修改所实现的主机，直至正确地实现设计要求。主要内容包括组织测试环境、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。

**（1）组织测试环境**

组织测试环境的任务是组织执行环境、产生对测试程序的约束，组织执行环境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。

存储程序工作方式要求，程序执行前，需要将程序从辅存装入主存，并将程序入口地址（首条指令装入位置）写入到PC中。这些工作本来是由软件（操作系统）完成的，由于本课程不涉及操作系统，因此，这些工作只能由系统复位操作触发硬件来实现。

组织执行环境时，将测试程序入口地址写入到PC中，可以采用使PC内容等于PC初值（假设为*x*）的方法实现，可由系统复位操作触发；将测试程序装入主存，可以采用将测试程序保存在IMEM中的方法实现，首条指令的存放地址为*x*（与PC初值相同）。

产生对测试程序的约束时，由于本课程要求MMU采用实地址存储管理模式，而测试程序在IMEM中的入口地址为*x*，因此，测试程序的起始逻辑地址必须从*x*。

**（2）编写测试程序**

编写测试程序的任务是组织、编写并保存测试程序。

主机测试的目标是主机能否正确地实现复位功能、程序执行过程、所支持指令功能，以及主存的配置及连接。除复位功能外，其余都由测试程序来完成。

组织测试程序时，有如下要求：应包含支持的所有指令，以测试设计是否有遗漏；应包含不支持的指令，以测试非法指令的处理结果；每条分支指令应能够产生2种执行结果；存指令与取指令的位置应不连续，以防遗漏因操作地址相同而导致的错误；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，以便查看每条指令的执行结果。

编写测试程序时，按A64的指令字格式形成指令字，程序的起始逻辑地址为*x*（与PC初值相同）。

保存测试程序时，将测试程序保存到IMEM的初始化文件中，首条指令的存放地址为*x*（与程序的起始逻辑地址相同）。

**（3）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，在仿真波形文件中首先进行系统复位操作（初始化CPU），然后CPU将自动执行测试程序，执行结果（仿真结果）可立即显示或保存到文件中。

注意：为了便于查看仿真结果，输出信息应为十六进制（或二进制），禁止用ASCII字符或有符号数表示。

**（3）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++程序调试的过程完全相同。

判断仿真结果正确性是基于理论结果进行的，理论结果由设计方案、仿真数据得到。由于测试目标中包含指令执行过程，故需要判断每个时钟周期的测试结果，只要与理论结果不同，就有错误（理论结果计算错、设计方案错或者电路错）。由于时序系统约定了指令周期所含时钟周期数，指令的uOPCmd序列约定了每个时钟周期需完成的操作，因此，根据仿真波形的输入信号，即可得到每个时钟周期的理论结果。

注意：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；数据路径相同、仅部件控制信号不同的所有指令（如add/sub/addu/subu），只有一条指令需要对每个时钟周期的测试结果进行分析，其余指令只需进行目的OPD结果分析。

分析错误原因是基于实现电路、测试程序进行的，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所有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，即可获知是否有错、错误点。

指令执行结果错误的类型有理论结果错误、电路错误2种。分析方法是逐个时钟周期进行对比，重点是分析结果开始不同的那个时钟周期，有如下步骤：①重新计算理论值；②分析数据在设计方案、电路中的路径及结果；③用当前数据对数据通路→功能部件逐级进行仿真；④对主机→CPU→数据通路及CU→功能部件逐级增加调试输出信号，重新执行测试程序。很显然，第③、④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。

**（4）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，从原理上分析了一个主机的设计与实现过程，可以适用于不同的指令集。本课程的设计中，应该以上述步骤为指引，结合A64的应用需求，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 MIPS32单周期CPU设计

本课程可缺省CPU的MMU及中断机构设计，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7条MIPS32指令为例，介绍单周期CPU的设计过程。假设CPU结构参数除数据引脚外，与本课程的设计要求相同，数据引脚改为32位。

### 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7条MIPS32指令如表A.1所示。

表A.1 MIPS32的7条指令功能

|  |  |  |
| --- | --- | --- |
| 指令名 | 指令功能 | 功能说明 |
| 有符号加add | rd←(rs)＋(rt) | 操作数为有符号整数 |
| 有符号减sub | rd←(rs)－(rt) |
| 按位或ori | rt←(rs) | ZExt(imme) | ZExt表示零扩展，操作数为逻辑数 |
| 取数lw | rt←M[(rs)＋SExt(disp)] | SExt表示符号扩展，无符号加法 |
| 存数sw | M[(rs)＋SExt(disp)]←(rt) |
| 相等转移beq | if ((rs)=(rt)) PC←(PC)＋4＋SExt(disp) <<2 | 无符号减法 |
| 跳转j | PC←(PC)高4位 || addr <<2 | || 表示拼接，<<2表示左移2位 |

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缺省，BIU需划分为IBIU、DBIU，部件间互连需使用多路选择器实现，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的数据引脚为32位、可寻址空间（指物理地址空间）为32位、控制引脚包含读/写/时钟3根信号线。

分析MIPS32指令集结构及表A.1的7条指令功能，可以得到如下分析结果：

·数据类型包括32位的有/无符号整数、逻辑数，有/无符号整数采用定点格式的补码/无符号编码表示，逻辑数采用位向量格式的逻辑编码表示；

·指令功能操作支持32位的有符号加法、有符号减法、按位或、无符号减法，无符号减法需要产生ZF标志，有符号加法/减法需产生OF标志，ZF、OF标志都无需保存；

·数据寻址支持寄存器寻址、立即寻址、基址寻址方式，地址计算的操作包括32位的无符号加法、16位→32位的位扩展（零扩展及符号扩展）；

·指令寻址支持隐含寻址、PC相对寻址、伪直接寻址方式，地址计算的方法分别为：(PC)＋4、(PC)＋4＋SExt(disp)<<2、(PC)高4位||addr<<2，操作包括32位的无符号加法、16位→32位的符号扩展、左移2位、拼接；

·寄存器的长度为32位、个数为32个，一条指令中最多有2次读操作、1次写操作，只存放32位的数据或地址，0#寄存器恒为零；

·存储器按字节编址、逻辑地址空间为32位，采用大端、对齐方式存放数据及指令，存储器与外设采用统一编址方式；

·各条指令的数据路径放在数据通路部件互连设计时分析（与通路部件设计有关）；

总清功能的分析结果是：可用PC、时序信号形成电路具有复位功能来实现。

因此，CPU的功能需求为上述指令系统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可以缺省，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、BIU三个子模块组成，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。

由指令系统分析结果可见，寄存器组为通用寄存器组（记为GPRs），指令的功能与状态寄存器的内容无关（不使用ZF/OF标志），故数据通路设计时可透明化状态寄存器。因此，所设计的数据通路由运算部件、GPRs、指令部件、BIU四个子模块组成。

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操作及数据寻址操作（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，指令功能操作及数据寻址操作部分的功能需求为：实现32位的有符号加法、有符号减法、无符号加法、无符号减法、按位或运算，实现16位→32位的零扩展、符号扩展操作，有符号加/减法时产生OF标志，无符号减法时产生ZF标志。

可见，指令寻址操作部分的功能需求为：实现(PC)＋4、(PC)＋4＋SExt(disp)<<2、(PC)高4位||addr<<2，所含的操作包括32位的无符号加法、16位→32位的符号扩展、左移2位、拼接。

**（2）GPRs**

GPRs负责实现数据及地址在寄存器中的存放及操作（读/写）。

可见，GPRs的功能需求为：包含32个32位寄存器，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。

注意，数据通路采用专用结构时，GPRs的读端口数可以按指令要求设置。

**（3）指令部件**

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

由于是单周期CPU，每个指令周期内只有3个时钟信号边沿（CLK开始、中间、结束时），可用作时序逻辑操作的脉冲信号，指令周期中的读IMEM或写PC、DMEM操作、写GPRs操作已各占用一个，因此，指令部件仅由PC组成，IR须缺省。

PC的复位功能是置PC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：仅包含PC，复位时PC初值为零，指令字在指令周期结束前保持不变。

**（4）BIU**

BIU负责实现存储器及外设访问操作的中转。

由CPU设计要求分析结果可见，BIU需划分成IBIU、DBIU。

BIU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。内部端口用于实现地址及数据的缓冲，数据转换电路用于实现数据存放方式、数据位数扩展（访存粒度＜数据宽度时）所需的格式转换，总线逻辑电路用于实现数据的传输过程。

由于是单周期CPU，访存操作最多只能占用一个时钟信号边沿（与上述IR缺省类似），而IMEM、DMEM要求为同步RAM，故内部端口须全部缺省。本例中，所支持lw/sw指令的访存粒度为32位、数据长度为32位，故访存粒度＝数据宽度，数据位数无需扩展，数据转换电路只需实现大端存放方式所需的格式转换；由于访存粒度＝数据宽度，访存可以采用常规传输方式，而访存时延＜0.5个时钟周期，导致访存过程无需协议控制，故总线逻辑电路可以缺省。

可见，BIU的功能需求为：内部端口全部缺省，数据转换电路实现大端存放方式所需的数据格式转换，总线逻辑电路缺省。

#### 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。

单周期CPU中，指令执行过程只包含一个状态，因此，时序系统只需采用工作脉冲一级时序，状态的改变与时钟信号（指令周期）同步，时序信号序列只有工作脉冲信号序列一级。

由于MIPS32的指令类型仅由操作码及功能码确定，因此，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7条指令的指令类型信号。

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。

可见，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。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、BIU四个子模块组成，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操作、数据寻址操作、指令寻址操作。

由总体设计结果可知，运算部件由数据操作单元、指令地址计算单元2个部分组成，数据操作单元负责实现指令功能操作、数据寻址操作，指令地址计算单元负责实现指令寻址操作。可见，数据操作单元、指令地址计算单元的部件不能复用。

①数据操作单元的设计

由总体设计结果可知，数据操作单元的功能需求为：实现32位的有符号加法、有符号减法、无符号加法、无符号减法、按位或运算，实现16位→32位的零扩展、符号扩展操作，有符号加/减法时产生OF标志，无符号减法时产生ZF标志。

数据操作单元可组织成ALU、ExtU（位扩展部件）两个部件，因为算术运算与位扩展操作无法复用器件，这样做可简化设计、节约成本。

·ALU设计

ALU的功能组织为：实现32位的有符号加法、有符号减法、无符号加法、无符号减法、按位或运算，并产生OF（有符号加/减法时）、ZF（所有运算时）。该结果满足了数据操作单元的算术运算、逻辑运算需求，但放宽了ZF的产生条件。

放宽ZF产生条件的原因是可以简化实现，且不影响ZF的使用（beq指令时）；但OF产生条件不能放宽，因为指令系统约定OF有效将产生异常事件。

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），标志输出OF和ZF，操作控制Op（3位）。

ALU的功能表约定如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Op | 000 | 001 | 010 | 011 | 100 |
| Out | A＋B | A－B | A＋B | A－B | A | B |
| OF | 0（无符号） | 0（无符号） | 0/1（有符号） | 0/1（有符号） | 0 |
| ZF | 0/1 | 0/1 | 0/1 | 0/1 | 0/1 |

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，Op为操作控制信号，Subctr用于控制加减运算类型（假设0/1表示加/减），OFctr用于控制是否产生OF（假设0/1表示否/是），Outsrc用于选择运算结果（假设0/1表示或运算/算术运算）。

图中，MUX内部的□、■表示控制信号为0、1（或最大值）时所选择的入端。



因此，ALU内部的控制信号形成模块的逻辑为：Subctr在Op＝001或011时有效，即Subctr＝·Op[0]（缺省为加法）或Op[0]（缺省为减法）；OFctr在Op＝010、011时有效，即OFctr＝·Op[1]；Outsrc在Op＝000~011时有效，即Outsrc＝。

·ExtU设计

ExtU的功能组织为：实现16位→32位的零扩展、符号扩展操作。该结果满足了数据操作单元的位扩展操作需求。

ExtU的I/O信号组织为：数据输入D（＝d15…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 d15…d0；Op＝1时为符号扩展，即Q＝d15…d15 d15…d0。

基于ExtU的功能表，ExtU的内部逻辑为：q15…q0＝d15…d0，q31~q16＝d15·Op。

②指令地址计算单元的设计

总体设计结果约定，指令地址计算单元的功能需求为：实现(PC)＋4、(PC)＋4＋SExt(disp)<<2、(PC)高4位||addr<<2，操作包括32位的无符号加法、16位→32位的符号扩展、左移2位、拼接。

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)＋4＋SExt(disp)<<2、(PC)高4位||addr<<2。

ACU的I/O信号组织为：地址输入PC（32位）、addr（26位）、disp（16位），地址输出NPC（32位），操作控制Jmp、Brn及ZF。其中，Jmp、Brn表示当前指令是否为j、beq指令，ZF表示beq指令的比较结果是否相等。由于ACU连接时，disp、addr低16位的信号源相同，因此，I/O信号组织时，可以缺省disp信号。

基于所支持指令的寻址方式需求，ACU的功能表约定为： （×表示任意值）

|  |  |  |  |
| --- | --- | --- | --- |
| Jmp | Brn | ZF | NPC |
| 1 | × | × | (PC)高4位||addr<<2 |
| 0 | 1 | 1 | (PC)＋4＋SExt(disp)<<2 |
| 0 | 1 | 0 | (PC)＋4 |
| 0 | 0 | × |

注意，移动位数为常数时，移位操作可用拼接方法实现。例如，SExt(disp)<<2＝SExt(disp||00)，(PC)高4位||addr<<2＝(PC)高4位||addr||00。

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，||表示信号线拼接，左移2位用拼接来实现，SExtU为18位→32位的符号扩展逻辑（可不封装成器件）。由于部件不能复用，故需要使用两个Adder。



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及操作（读/写）。

由总体设计结果可知，GPRs的功能需求为：包含32个32位寄存器，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，复位控制Clr。

GPRs的功能表约定如下：（假设a#寄存器的内容用GPRs[a]表示）

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

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#寄存器恒为零采用了一直写0的方式来实现，也可以采用一直清零的方法实现。



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，包括指令字在指令周期结束前保持不变。

由总体设计结果可知，指令部件的功能需求为：仅包含PC，复位时PC初值为零，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，复位功能通过控制其清零引脚来实现。若PC初值不为零，则复位功能需通过额外的电路来实现。

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。而写PC的时机与IMEM是否为同步RAM有关。

单周期CPU中，每个指令周期只有3个时钟信号边沿，可用于时序逻辑操作。由lw指令的数据路径可知，DMEM操作、写GPRs操作需占用第2个、第3个时钟边沿。

当IMEM为同步RAM时，读IMEM应放在第1个时钟边沿进行，写PC须放在第2个时钟边沿进行，如图A.4(a)所示。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个时钟边沿（即下个指令周期的第1个时钟边沿）进行，则读IMEM无法取出下条指令，因为写PC有一定延迟，读IMEM时PC仍是旧值。



（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

当IMEM为异步RAM时，写PC须放在第3个时钟边沿（即下个指令周期的第1个时钟边沿）进行，读IMEM在第1个时钟边沿之后即可进行，如图A.4(b)所示。此后至指令周期结束前，PC一直是旧值，IMEM的输出不会改变，指令字会保持不变。若写PC放在第2个时钟边沿进行，则IMEM的输出在后半个时钟周期会变为下条指令的内容。

**（4）BIU设计**

BIU负责实现存储器及外设访问操作的中转。

由总体设计结果可知，BIU需划分成IBIU、DBIU，每个BIU的功能需求为：内部端口全部缺省，数据转换电路实现大端存放方式所需的数据格式转换，总线逻辑电路缺省。

每个BIU通常组织成一个部件，BIU的功能组织为：实现上述功能需求。

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；地址总线ABus（30位），数据总线DBusOut和DBusIn（32位），控制总线CBus（3位）。IBIU的I/O信号与DBIU基本相同，仅仅没有DataOut及DBusOut信号。

由于MIPS的逻辑地址空间为32位，故BIU的Addr为32位；由于MIPS的存储器按字节编址，要求BIU的数据引脚为32位、可寻址空间为32位，故BIU的ABus位数为log2(232×1B)/32bit＝30位。本例中，BIU的数据总线组织成两条单向总线，目的是便于与单向数据引脚的主存连接。

基于BIU的功能需求，IBIU及DBIU的内部结构如图A.5所示。其中，IBIU的IMRd、IMWr引脚分别连接1、0。



图A.5 支持lw/sw指令的IBIU及DBIU

大端存放方式的要求为：DBusOut7~0＝DataOut31~24、…、DBusOut31~24＝DataOut7~0，DataIn31~24＝DBusIn7~0、…、DataIn7~0＝DBusIn31~24，因此，数据转换电路的内部逻辑为：颠倒数据的字节次序。实现时，逆序连接字节信号线即可。

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、IBIU、DBIU。

数据通路中，假设μOP需要使用的工作脉冲为P0和P1，P0的上升沿在节拍周期开始/结束时，P1的上升沿在节拍周期的中部，不够用时再酌情增加信号数。单周期CPU中，1个指令周期＝1个节拍周期。

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指令的数据通路**

add/sub指令属于R-型指令，功能为rd←(rs)±(rt)，需产生标志OF。

add/sub指令数据操作的数据路径为：GPRs→ALU→GPRs。add/sub指令的数据通路如图A.6所示，其中，寄存器rs、rt的内容分别从GPRs的dA、dB读出，ALU的结果从GPRs的dW端写到寄存器rd中，ALUctr控制ALU的操作类型，RegWr控制是/否进行写GPRs操作。

注意，ALU的A端应该连接(rs)，因为ALU的功能是A±B；写GPRs应该安排在时钟周期结束时进行，因为lw指令有此需求（应统一处理），因此，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rt←(rs) | ZExt(imme)。

ori指令数据操作的数据路径为：GPRs及ExtU→ALU→GPRs，与add/sub指令的差别是一个源操作数的寻址方式不同。可见，I-型运算类指令与R-型指令的数据路径基本相同，图A.7的数据通路支持3条指令的执行，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ori指令的数据通路

与图A.6相比，图A.7增加了一个ExtU及两个MUX，用来处理R-型指令与I-型指令的格式差异。ExtU用于对imme进行零扩展（Extctr＝0），MUX1用于选择源操作数，MUX2用于选择目的寄存器（rd或rt）。

**（3）lw/sw指令的数据通路**

lw/sw指令属于I-型指令，功能是rt←M[(rs)＋SExt(disp)]、M[(rs)＋SExt(disp)]←(rt)。

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。可见，lw/sw的操作数地址计算与ori指令的数据操作的数据路径完全相同，图A.8的数据通路支持5条指令的执行，与图A.7的差别是增加了lw/sw指令的数据路径。



图A.8 支持lw/sw指令的数据通路

与图A.7相比，图A.8增加了DBIU及一个MUX，实现存储器读/写功能。MUX3用于选择目的操作数来源（ALU或DBIU）。

注意，DMEM不属于CPU内部的数据通路。教材中直接连接DMEM的原因，一是BIU的组成原理需要I/O接口、总线等知识的支持，当时尚未学到；二是DMEM虽不属于CPU，但可放在CPU芯片中（如Cache），直接连接并没有错。图中的DMEM、连接电路设计将在主机实现的时候再讨论。

由于DMEM为同步存储器，lw指令的读DMEM、写GPRs为有序的时序逻辑操作，单周期CPU中，读/写DMEM只能放在时钟周期中部，写GPRs放在时钟周期结束时，因此，DMEM的Clk应连接P1，GPRs的Clk应连接P0（add/sub指令中已实现）。

**（4）beq指令的数据通路**

beq指令属于I-型指令，功能是PC←((rs)=(rt)) ? (PC)＋4＋SExt(disp)<<2 : (PC)＋4。

beq指令数据操作的数据路径为：GPRs→ALU（产生ZF）。可见，beq指令数据操作的路径包含在add/sub指令数据操作的数据路径中，因此，图A.8的数据通路可以满足beq指令的数据操作需求，只是ALU需要产生标志ZF。

**（5）j指令的数据通路**

j指令属于J-型指令，功能是PC←(PC)高4位 || addr <<2。

j指令没有数据操作，因此，图A.8的数据通路也可满足j指令的需求。

**（6）取指令及指令寻址操作的数据通路**

由于缺省IR，取指令的操作为读出IMEM[(PC)]，并要求IMEM的输出在指令周期结束前保持不变。取指令操作的数据路径为：PC→IBIU→IMEM→IBIU→ID及GPRs和ExtU。

由于ACU实现了所有指令的指令寻址操作，地址参数来自于PC及IBIU，因此，所有指令寻址操作的数据路径为：PC及IBIU→ACU→PC。不同指令所需的控制信号（来自ID及ALU）有所不同。

由于IMEM为同步RAM，单周期CPU中，读IMEM应该放在时钟周期开始时进行，写PC只能放在时钟周期中部进行，因此，IMEM的Clk应连接P0，PC的Clk应连接P1。又由于每个时钟周期都要读IMEM、写PC，因此，读IMEM、写PC可只由工作脉冲控制，无需设置控制信号。

汇总取指令操作、各条指令数据操作及指令寻址操作的数据路径，可得到支持7条指令的完整的单周期数据通路，如图A.9所示。图A.9与图A.8的差别是增加了取指令及指令寻址操作的数据路径。



图A.9 支持7条指令的单周期数据通路

注意，IMEM、DMEM及其连接电路不属于CPU。

可见，数据通路的接口信号包括：复位信号，IBIU及DBIU的外部接口信号，输出到ID的指令操作码信号，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。

至此，数据通路设计全部完成。设计结果的正确性，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约定功能、指令执行过程的要求，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)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。其中，InsJ、InsB用于指令寻址操作的控制，其余用于指令功能操作、数据寻址操作的控制。

表A.2 指令执行过程的μOP控制信号组织

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令 | InsJ | InsB | Extctr | ALUBsrc | ALUctr | RegAsrc | RegDsrc | RegWr | MemRd | MemWr |
| add | 0 | 0 | × | 1 | 010 | 1 | 1 | 1 | 0 | 0 |
| sub | 0 | 0 | × | 1 | 011 | 1 | 1 | 1 | 0 | 0 |
| ori | 0 | 0 | 0 | 0 | 100 | 0 | 1 | 1 | 0 | 0 |
| lw | 0 | 0 | 1 | 0 | 000 | 0 | 0 | 1 | 1 | 0 |
| sw | 0 | 0 | 1 | 0 | 000 | × | × | 0 | 0 | 1 |
| beq | 0 | 1 | × | 1 | 001 | × | × | 0 | 0 | 0 |
| j | 1 | 0 | × | × | 000 | × | × | 0 | 0 | 0 |

注意，①为了防止lw/sw/beq指令产生OF标志，ALUctr应为无符号加/减运算；

②与指令的数据路径无关的组合逻辑部件的控制信号，可以为不会产生异常的任意值，如j指令的ALUctr≠010或011，其它为×；

③与指令的数据路径无关的时序逻辑部件的控制信号，必须为无效值，如RegWr＝MemWr＝MemRd＝0；

④单周期CPU中，无须设置End信号，因为它每个时钟周期都有效。

可见，基于图A.9的数据通路，7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。

通常将每一个不同的操作（指同时执行的μOP）看作是一个状态，对各条指令执行过程的状态进行汇总，就得到了一个状态转换图。

单周期CPU中，每条指令的执行过程只有一个状态（即μOPCmd序列只有一步），如表A.2中任意一行所示，汇总后得到的指令执行过程状态转换图如图A.10所示。每个状态中，有效的μOPCmd已全部列出，未列出的μOPCm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组成、时序信号的定时方式。

由总体设计结果可知，时序系统只采用工作脉冲一级时序，时序信号序列只有工作脉冲信号序列一级。

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，由于工作脉冲信号均采用定长循环周期（＝1个节拍周期），因此，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CPU设计要求可知，时序信号采用同步方式定时，即1个节拍周期＝1个时钟周期＝工作脉冲信号的循环周期。本例中，时钟周期的宽度为lw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。

由总体设计结果可知，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7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：操作码信号op（6位），功能码信号func（6位），指令类型信号Iadd/Isub/Iori/Ilw/Isw/Ibeq/Ij。

ID的内部逻辑为：由2个级联控制的6:64译码器组成。

#### 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。

由总体设计结果可知，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。

时序系统的组织结果是，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。

时序信号形成电路的功能组织为：复位时2个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。说明，工作脉冲信号的循环周期为节拍周期，单周期CPU中节拍周期＝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（假设复位后P0首先有效）。说明，用于定时的信号只有CLK的原因是，CPU设计要求为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，如图2.9所示，两者的接口为节拍脉冲信号CP。注意，节拍周期开始于CP上升沿。

组织定序逻辑时，工作脉冲的状态用2个门电路表示，复位时2个门电路的输出全部为0（无效）；2个下一状态产生函数都只能用门电路输出信号（当前状态）的产生函数表示，2个产生函数中都需包含复位逻辑（复位时使P0＝P1＝0），P0的产生函数中还需包含启动逻辑（复位后使P0首先有效），因此，2个当前状态的产生函数分别为：P0＝CP·、P1＝·（即·）。

组织定时逻辑时，由于工作脉冲信号的循环周期为1个时钟周期，因此，内部逻辑可设计为CP＝CLK。注意，CP＝CLK的逻辑要求时钟周期开始于CLK上升沿，μOP控制信号等都在CLK上升沿产生；CLK的占空比应满足工作脉冲P1的要求。

连接上述定时逻辑电路与定序逻辑电路，即可得到所设计的时序信号形成电路。

#### 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。

由总体设计结果可知，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/Iori/Ilw/Isw/Ibeq/Ij（7个），μOP控制信号InsJ/InsB/Extctr/ALUBsrc/ALUctr/RegAsrc/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。

形成各个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

由于指令周期＝1个节拍周期，故μOPCmd使用时间表可以组织为1行、10列；状态转换图中，每个状态的转入条件仅为指令类型、不含时间戳；在每个状态的各个有效μOPCmd对应的单元格中，填入该状态的指令类型（转入条件）后，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

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | InsJ | InsB | Extctr | ALUBsrc | ALUctr | RegAsrc | RegDsrc | RegWr | MemRd | MemWr |
| T0 | Ij | Ibeq | Ilw  +Isw | Iadd  +Isub  +Ibeq | bit2:Iori  bit1:Iadd+Isub  bit0:Isub+Ibeq | Iadd  +Isub | Iadd  +Isub  +Iori | Iadd+Isub  +Iori+Ilw | Ilw | Isw |

·形成各个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的逻辑表达式分别为：

InsJ＝Ij，InsB＝Ibeq；

Extctr＝Ilw＋Isw，ALUBsrc＝Iadd＋Isub＋Ibeq，

ALUctr[2]＝Iori，ALUctr[1]＝Iadd＋Isub，ALUctr[0]＝Isub＋Ibeq，

RegAsrc＝Iadd＋Isub，RegDsrc＝Iadd＋Isub＋Iori，RegWr＝Iadd＋Isub＋Iori＋Ilw，

MemRd＝Ilw，MemWr＝Isw。

使用组合逻辑电路，实现各个μOP控制信号的有效逻辑，即可得到所设计的μOP控制信号形成电路。

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7条指令的单周期CU

至此，CPU的设计全部完成。

## 附录B MIPS32多周期CPU的设计

多周期CPU的典型特征是，指令周期＝*x*个时钟周期，不同指令的*x*可以不同，因此，数据通路中的部件可以复用，对存储器结构、数据通路结构的类型也没有限制。

多周期CPU设计的基本思想是，将每条指令的执行过程分成多个时间上大致相等的阶段，每个阶段在一个时钟周期内完成，其它阶段需要使用的结果都必须保存在寄存器中，时钟周期的宽度为最复杂阶段所用的时间。

本课程可缺省CPU的MMU及中断机构设计，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，以附录A的单周期CPU所支持的7条MIPS32指令为例，介绍多周期CPU的设计过程。理解多周期CPU的设计方法后，设计支持15条MIPS32指令的多周期CPU就变得简单了。

### B.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。

由A.1节的需求分析过程可见，分析结果仅与CPU设计要求、所支持指令有关，本例的设计要求与附录A相同，因此，A.1节的需求分析结果适用于本例。

### B.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A.2节的功能模块划分可见，划分结果仅与CPU设计要求有关，与所支持的指令无关，而本例的CPU设计要求与附录A相同，因此，A.2节的功能模块划分结果适用于本例，即本例的CPU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、BIU三个子模块组成，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。

由A.2节的数据通路子模块划分可见，子模块划分与指令系统有关，与指令周期类型（单/多周期）无关，因此，本例的数据通路由运算部件、通用寄存器组GPRs、指令部件、BIU，以及附加寄存器组成。

由A.2节的数据通路子模块功能分配可见，子模块的功能组织与指令周期类型无关时，其功能需求与A.2节相同，因此，运算部件、GPRs的功能需求与A.2节相同，指令部件、BIU的功能需求与A.2节有所不同。下面分别进行讨论。

**·指令部件**

指令部件负责实现指令控制功能，包括指令字在指令周期结束前保持不变。

多周期CPU中，指令寻址操作可放在多个时钟周期实现（多次写PC），指令字在指令周期结束前保持不变的需求，不能通过在适当时候写PC来实现，最简单的实现方法是设置IR，指令周期内只写一次，因此，指令部件由PC、IR组成。

可见，指令部件的功能需求为：包含PC、IR，复位时PC初值为零，IR存放当前指令的内容。

**·BIU**

BIU负责实现存储器及外设访问操作的中转。

由于要求存储器采用哈佛结构，因此，BIU需划分成IBIU、DBIU，每个BIU由内部端口、数据转换电路、总线逻辑电路组成。

多周期CPU中，为了使访存μOP与其他μOP可以并行（数据路径不冲突），故BIU的内部端口应该包含MAR、MDR。使用“应该”字样的原因是，附加寄存器可以实现内部端口功能，访存μOP不与其他μOP并行时可以缺省内部端口。BIU的数据转换电路、总线逻辑电路组织与指令周期类型无关，故与A.2节相同。

可见，BIU的功能需求为：内部端口应该包含MAR、MDR，数据转换电路实现大端存放方式所需的数据格式转换，总线逻辑电路缺省。

**·附加寄存器**

附加寄存器负责实现其他时钟周期需要使用的操作结果的存放。

多周期CPU中，每条指令的执行过程都可以组织成μOP序列，其他步骤（其他时钟周期）需要使用的μOP结果都必须保存到附加寄存器中，保存（写入）在μOP结束的那个时钟周期进行。PC、IR、BIU内部端口都可以兼作附加寄存器。

可见，附加寄存器的功能需求为：保存所有其他时钟周期需要使用的μOP结果。

#### 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。

多周期CPU中，指令执行过程包含多个状态，因此，时序系统需采用节拍、工作脉冲两级时序，状态的改变与节拍信号同步，时序信号序列有节拍信号序列、工作脉冲信号序列2级。

由于ID的组织仅与指令字格式有关，因此，ID的功能需求与A.2节相同。

可见，时序信号形成电路的功能需求为：根据当前指令的类型，输出所组织时序系统的相应时序信号序列，复位时所有节拍信号全部无效。

可见，μOP控制信号形成电路的功能需求为：根据节拍信号的状态，输出当前指令执行过程中的当前节拍所需的所有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。

### B.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。

由于本例所支持的指令与单周期CPU相同，因此，可以通过改进单周期数据通路，来得到本例的多周期数据通路。

#### 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。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、通用寄存器组GPRs、指令部件、BIU、附加寄存器组成，各个子模块（不包含附加寄存器）的功能需求已确定。

数据通路的每个子模块都可以组织成一个或几个部件，其设计结果会影响μOP的功能组织、指令执行过程的数据路径。由于附加寄存器的设计与μOP的功能组织、指令执行过程的数据路径有关，因此，功能部件设计可划分为子模块部件设计、附加寄存器设计2个阶段。

**（1）子模块部件的设计**

多周期CPU中的部件可以复用，本例拟采用的部件复用方案是，运算部件中的指令寻址操作尽量复用指令功能操作及数据寻址操作的部件。

·运算部件设计

运算部件的功能需求与单周期CPU相同，故指令功能操作及数据寻址操作可使用A.3节的ALU、ExtU实现；指令寻址操作所需的无符号加法、16位→32位的符号扩展可复用ALU、ExtU实现，所需的左移2位、拼接操作需使用新的部件实现。

因此，运算部件由ALU、ExtU、SL2（左移2位）、Splice（拼接）四个部件组成。ALU、ExtU与A.3节相同，SL2、Splice请自行设计（功能很简单）。

·GPRs设计

GPRs的功能需求与单周期CPU相同，故GPRs可使用A.3节的设计结果。

·指令部件设计

指令部件的功能需求为：包含PC、IR，复位时PC初值为零，IR存放当前指令的内容。因此，指令部件可组织为：带清零功能的寄存器PC、寄存器IR。

·BIU设计

BIU由IBIU、DBIU组成，每个BIU的功能需求为：内部端口应该包含MAR、MDR，数据转换电路实现大端存放方式所需的数据格式转换，总线逻辑电路缺省。由于仅内部端口的功能需求与A.3节有所不同，因此，BIU与图A.5相同，内部端口除外。

由于PC的内容在读IMEM时会保持不变，故PC及IR可以实现IBIU内部端口的功能，因此，IBIU的内部端口可以缺省，IBIU的设计结果与图A.5相同。

由于图A.5中DBIU包含输入/输出分离的两条数据总线，DBIU的内部端口应该包含MAR、MDRr（存放所读数据）、MDRw（存放欲写数据）。由图A.8可见，GPRs的读端口B应连接到DBIU的MDRw端口，其内容在sw指令执行阶段会保持不变（地址未改变），故DBIU的MDRw端口可以缺省；ALU的出端应连接到DBIU的MAR端口，当ALU的出端设置有附加寄存器时，DBIU的MAR端口可以缺省；DBIU的MDRr端口应连接到GPRs的写端口，当某μOP功能为GPRs←DMEM时，DBIU的MDRr端口可以缺省。

本例拟采用DBIU内部端口全部缺省的方案，因此，DBIU的设计结果与图A.5相同。该方案的附加要求是：ALU的出端需设置附加寄存器，GPRs←DMEM功能需组织在同一个μOP中。后续设计中，应遵守这个约定。

**（2）附加寄存器的设计**

附加寄存器负责实现其他时钟周期需要使用的操作结果的存放。

由总体设计结果可知，附加寄存器的功能需求为：保存所有其他时钟周期需要使用的μOP结果。

附加寄存器的设计方法是，①确定时钟周期的长度，即一个时钟周期可完成哪些部件操作；②确定基于时钟周期的指令基本操作功能，即按时钟周期划分指令执行过程的操作；③确定所需的附加寄存器，即可在所有指令基本操作的末个部件出端设置。

确定时钟周期长度时，应使一个时钟周期可完成的部件操作时延大致相等，以提高系统效率，时延较长的操作可用多个时钟周期实现。

由图A.9可见，GPRs读、GPRs写、ALU操作的时延相近，因此，本例的方案为：时钟周期长度等于GPRs及ALU操作的时延最大值。

确定基于时钟周期的指令基本操作功能时，应基于指令的数据路径、时钟周期的长度来划分，指令基本操作的时延尽量组织成1个时钟周期。

由时钟周期长度、访存时延要求（＜0.5个时钟周期）可知，所有指令的数据路径中，时延＝1个时钟周期的基本操作只能包括GPRs读/GPRs写/ALU操作/IMEM读/DMEM读/DMEM写中的一个，以及个别微时延部件操作（如ExtU或SL2操作）。而BIU设计所要求的、功能为GPRs←DMEM的基本操作，操作时延＝2个时钟周期。

因此，本例指令基本操作的末个部件操作有：GPRs读、GPRs写、ALU操作、IMEM读、DMEM写。注意，DMEM读不在其中。

确定所需的附加寄存器时，应忽略末个部件操作为写操作、目标位置已存在寄存器的指令基本操作。

由基本操作的功能可知，IMEM、GPRs、ALU的出端都应设置附加寄存器。由于IR可以兼作IMEM出端的附加寄存器，因此，本例的方案为：在GPRs读端口设置附加寄存器A和B，在ALU出端设置附加寄存器ALUOut。

至此，功能部件设计全部完成，所设计的部件有ALU、ExtU、SL2（左移2位）、Splice（拼接）、GPRs、PC、IR、IBIU和DBIU，以及附加寄存器A、B和ALUOut。

#### 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，本例的功能部件有：ALU、ExtU、SL2、Splice、GPRs、PC、IR、IBIU和DBIU，及附加寄存器A、B和ALUOut。与单周期CPU相比，基于部件复用方案，缺省了ACU，增加了SL2、Splice；基于多周期CPU特征，增加了IR、附加寄存器。

由于本例的功能部件与单周期CPU基本相同，故每条指令的数据路径基本相同，因此，部件互连设计的任务主要是，在单周期数据通路基础上，修改与复用方案相关的数据路径，连接所设置的附加寄存器（包含IR）。

基于图A.9，删除ACU相关电路，增加部件SL2、Splice、IR、A、B和ALUOut，添加操作PC←(PC)＋4、PC←(PC)＋4＋SExt(disp)<<2、PC←(PC)高4位 || addr<<2对应的数据路径，得到的多周期数据通路如图B.1所示。其中，增设的部件已经用底纹标注出来，SL2、Splice分别实现左移2位、拼接功能。由于A和B每个时钟周期都可写入，故不需要控制信号。



图B.1 支持7条指令的多周期数据通路

多周期CPU中，指令寻址方式可通过多次操作实现，取指时实现PC←(PC)＋4，故写PC可能为2次，是否重写还与ZF值有关。ZF判断可以放在数据通路中实现，如图B.1所示，此时需2个μOPCmd，以区分写入条件，即PCWrB＝Ibeq**·**T*x*，其中，Ibeq为j指令译码信号，T*x*为节拍信号。ZF判断也可放在CU中实现，此时只需1个μOPCmd，即PCWr＝Ibeq**·**T*x***·**ZF＋…。感兴趣的同学，可以尝试写PC用1个μOPCmd控制。

可见，各条指令所需的μOP功能有：IR←IMEM[(PC)]，A←(GPRs[rs])、B←(GPRs[rt])，ALUOut←(A) op (B)、ALUOut←(A) op ExtU，GPRs[rt/rd]←(ALUOut)，GPRs[rt]←DMEM[(ALUOut)]、DMEM[(ALUOut)]←(B)，及PC←(PC)＋4、PC←(PC)＋SL2、PC←Splice。其中，GPRs[rt]←DMEM[(ALUOut)]为复杂μOP，其余为基本μOP。

为了减小μOP的时延，PC、IR、A、B、ALUOut、GPRs的写入时间都应安排在节拍周期结束时，相应地，IMEM、DMEM的操作时间应安排在节拍周期的中间。各个时序逻辑部件的时钟脉冲线连接如图B.1所示（P0及P1上升沿的假设同单周期数据通路）。

可见，多周期数据通路的接口信号包括：复位信号，IBIU及DBIU的外部接口信号，输出到ID的指令操作码信号，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15个μOP控制信号。

至此，数据通路设计全部完成。设计结果的正确性，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约定功能、指令执行过程的要求，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。

多周期CPU的指令执行过程中，取指令、分析指令、执行指令阶段的操作可以放在不同时钟周期实现，取指令、分析指令的操作对所有指令是通用的，取指令时还需要实现PC←(PC)＋4，指令寻址的剩余操作放在执行指令阶段实现。

组织指令执行过程的操作时，本例采用如下策略：

①尽量组织成基本μOP，其源数据、结果都放在状态部件中；

②复杂μOP采用同步控制方式实现，即在μOP开始、结束时的那个节拍各用1组μOPCmd来控制；

③IMEM、DMEM操作的所有信号须保持到μOP结束时，地址、数据信号除外（同步RAM自己会锁存）；

④μOP尽量提前安排，即只要部件有空闲就安排。

下面，开始组织各条指令执行过程的μOPCmd序列。注意，μOPCmd序列中，有效的μOPCmd需全部列出（无效的μOPCmd可不列出），没有End信号（CPU不含中断机构所致）。

取指令阶段的功能为：IR←M[(PC)]、PC←(PC)＋4。基于图B.1，其μOPCmd如下：

t1：IMRd、IRWr，ALUAsrc＝1、ALUBsrc＝3、ALUctr＝0、PCsrc＝1、PCWr

执行指令阶段的功能为：实现各条指令的约定功能及指令寻址剩余操作。基于图B.1，7条指令执行阶段的μOPCmd序列如下：

（1）add/sub指令。指令功能为rd←(rs)±(rt)，执行阶段的μOPCmd序列为

t2：无 ；μOP为A←(rs)、B←(rt)

t3：ALUAsrc＝0、ALUBsrc＝2、ALUctr＝2/3、ALUOWr

t4：RegAsrc＝1、RegDsrc＝1、RegWr

（2）ori指令。指令功能为rt←(rs)| ZExt(imme)，执行阶段的μOPCmd序列为

t2：无 ；μOP为A←(rs)

t3：ALUAsrc＝0、ALUBsrc＝1、ALUctr＝4、ALUOWr

t4：RegAsrc＝0、RegDsrc＝1、RegWr

（3）lw指令。指令功能为rt←M[(rs)＋SExt(disp)]，执行阶段的μOPCmd序列为

t2：无 ；μOP为A←(rs)

t3：Extctr、ALUAsrc＝0、ALUBsrc＝1、ALUctr＝0、ALUOWr

t4：MemRd

t5：MemRd、RegAsrc＝0、RegDsrc＝0、RegWr

说明，BIU设计时约定的μOP功能是GPRs[rt]←DMEM[(ALUOut)]，该复杂μOP采用同步控制方式实现时，需用2组μOPCmd（t4及t5）来控制；DMEM的操作控制信号需保持到μOP结束时，故MemRd在t4及t5都有效。

（4）sw指令。指令功能为M[(rs)＋SExt(disp)]←(rt)，执行阶段的μOPCmd序列为

t2：无 ；μOP为A←(rs)、B←(rt)

t3：Extctr、ALUAsrc＝0、ALUBsrc＝1、ALUctr＝0、ALUOWr

t4：MemWr

（5）beq指令。指令功能为if ((rs)=(rt)) PC←(PC)＋4＋SExt(disp)<<2，执行阶段需要实现if ((rs)=(rt)) PC←(PC)＋SExt(disp)<<2，执行阶段的μOPCmd序列为

t2：Extctr、ALUAsrc＝1、ALUBsrc＝0、ALUctr＝0、ALUOWr

；μOP还包括A←(rs)、B←(rt)

t3：ALUAsrc＝0、ALUBsrc＝2、ALUctr＝1、PCsrc＝0、PCWrB

（6）j指令。指令功能为PC←(PC)高4位 ||addr <<2，执行阶段的μOPCmd序列为

t2：无 ；为了与其他指令保持一致

t3：PCsrc＝2、PCWr

分析指令阶段的功能为：实现指令译码操作。有2种组织方法：在t1步与t2步之间增加一个步骤，或将指令译码操作合并到t2步中。

合并的方法有如下2个条件：合并不会影响t2步μOP的实现，合并不会增加t2步的节拍周期长度。前者要求t2步的μOP不使用节拍周期开始时的脉冲信号上升沿。

本例拟采用将指令译码操作合并到t2步的方案。各条指令μOPCmd序列中，t2步μOP的集合为：A←(rs)、B←(rt)、ALUOut←(PC)＋SExt(disp)<<2，由于部件操作都是组合逻辑操作，附加寄存器在节拍周期结束时写入，因此，指令译码操作合并到t2步后，不会影响t2步μOP的实现，本例的方案是可行的。

由于t2步的μOP结果都保存在附加寄存器中，为了简化操作控制，可以使每条指令t2步都包含上述3个μOP，结果不使用时忽略即可。例如，图B.1中的A、B未设置操作控制信号，每个节拍周期都实现A←(rs)、B←(rt)功能，包括t2步。

汇总所有指令的μOPCmd序列，可以发现，指令执行过程可划分为5个阶段（节拍）：取指令、译码/读GPRs、ALU操作、MEM操作、写GPRs，不同指令所含的阶段不同，前2个阶段对所有指令是通用的。

可见，基于图B.1的数据通路，7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。

### B.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。

多周期数据通路的设计结果如图B.1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，15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。

多周期数据通路中，每条指令执行过程的状态个数有所不同，汇总所有指令执行过程的μOPCmd序列，即可得到如图B.2所示的指令执行过程状态转换图，其中(a)表示控制信号的取值为a。



图B.2 多周期数据通路的指令执行过程状态转换图

本例中，图B.2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组成、时序信号的定时方式。

由总体设计结果可知，时序系统采用节拍、工作脉冲两级时序，时序信号序列有节拍信号序列、工作脉冲信号序列2级。

组织时序信号的个数时，由图B.2可见，节拍信号需要5个；由图B.1可见，工作脉冲需要2个。

组织节拍信号序列时，由图B.2可见，状态转换图的路径长度有3种，本例约定节拍信号用来表示操作类型，因此，节拍信号序列设计为4种，每种节拍信号序列的组成如图B.3所示。图中还根据状态转换图，标出了每种节拍信号序列的适用条件，其中，R-型指令的节拍信号序列也适用于ori等I-型运算类指令。



图B.3 多周期CU的时序信号序列

组织工作脉冲信号序列时，由于工作脉冲信号均采用定长循环周期（＝1个节拍周期），因此，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，如图B.3所示。

组织时序信号的定时方式时，由CPU设计要求可知，时序信号采用同步方式定时，即1个节拍周期＝1个时钟周期。本例中，时钟周期的宽度为ALU等μOP的时延。

#### 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。

ID的组成只与指令系统有关，因此，可以直接使用A.4节单周期CPU的ID。

#### 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。

由总体设计结果可知，时序信号形成电路的功能需求为：根据当前指令的类型，输出所组织时序系统的相应时序信号序列，复位时所有节拍信号全部无效。

时序系统的组织结果是，节拍信号有5个，工作脉冲信号有2个；节拍信号序列有4种，工作脉冲信号序列有1种，两级信号序列的组成如图B.3所示；1个节拍周期等于1个时钟周期。

时序信号形成电路的功能组织为：复位时5个节拍信号全部无效，工作时按序输出当前指令类型对应的节拍信号序列中的每个节拍信号，以及工作脉冲信号序列中的每个工作脉冲信号，节拍信号长度等于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，指令类型信号Iadd/Isub/Iori/Ilw/Isw/Ibeq/Ij，节拍信号T0～T4（假设复位后T0首先有效），工作脉冲信号P0及P1（假设复位后P0首先有效）。由于时序信号序列只有4种，故指令类型信号可以用2根信号线代替。

时序信号形成电路的功能表约定如下：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 指令类型 | Iadd | Isub | Iori | Ilw | Isw | Ibeq | Ij |
| 对应序列类型 | R-指令周期 | | | lw指令周期 | sw指令周期 | beq/j指令周期 | |

注意，设计要求为μOP采用同步方式定时，故定时逻辑的输入信号只有时钟信号，没有控制方式、操作状态信号。

时序信号形成电路的内部逻辑由定序逻辑、定时逻辑组成，如图2.9所示，两者的接口为节拍脉冲信号CP。注意，节拍周期开始于CP上升沿。

·定序逻辑的组织

对节拍信号而言，节拍的状态用5个触发器表示，复位时T0～T4的输出全部为0（无效）；5个下一状态产生函数都用触发器入端信号的产生函数表示（下个节拍输出），函数值取决于当前指令的当前节拍在节拍信号序列中的位置，T0的函数中需包含启动逻辑（复位后使T0首先有效），由图B.3可见，5个下一状态产生函数分别为：

T1＝T0，T2＝T1，T3＝(lw＋sw)⋅T2，T4＝(add＋sub＋ori)⋅T2＋lw⋅T3，

T0＝＋(add＋sub＋ori＋lw)⋅T4＋sw⋅T3＋(beq＋j)⋅T2。

其中，为启动逻辑，复位时T0的输入端为1，复位后CP的首个脉冲信号到来时，T0的输出变为1，T1～T4的输出还为0。

对工作脉冲信号而言，工作脉冲的状态用2个门电路表示，门电路中不包含复位逻辑（非第1级时序信号）；2个下一状态产生函数都只能用门电路输出信号（当前状态）的产生函数表示，P0的产生函数中需包含启动逻辑（复位后使P0首先有效），因此，2个当前状态产生函数分别为：P0＝CP、P1＝。

·定时逻辑的组织

由于节拍信号长度等于1个时钟周期，因此，内部逻辑可设计为CP＝CLK。注意，CP＝CLK的逻辑要求时钟周期开始于CLK上升沿，μOP控制信号等都在CLK上升沿产生；CLK的占空比应满足工作脉冲P1的要求。

连接上述定时逻辑电路与定序逻辑电路，即可得到本例所设计的时序信号形成电路，如图B.4所示。



图B.4 多周期CU的时序信号形成电路

#### 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。

由总体设计结果可知，μOP控制信号形成电路的功能需求为：根据节拍信号的状态，输出当前指令执行过程中的当前节拍所需的所有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/Iori/Ilw/Isw/Ibeq/Ij（7个），节拍信号T0～T4，μOP控制信号IMRd/IRWr/PCWr/PCWrB/PCsrc/Extctr/ALUAsrc/ ALUBsrc/ALUctr/ALUOWr/RegAsrc/RegDsrc/RegWr/MemRd/MemWr（15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。

形成各个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

μOPCmd使用时间表可以组织为15行、5列。根据时序系统的约定，状态转换图中，各个状态的时间戳如下：if为T0，id为T1，ex\_r、ex\_i、ex\_m、ex\_b、ex\_j为T2，mem\_r、mem\_w为T3，wb\_r、wb\_i、wb\_m为T4；各个状态的转入条件为指令类型，如ex\_b（时间戳为T2）的转入条件为：指令类型＝beq，又如wb\_（时间戳为T4）i的转入条件为：指令类型＝ori。

针对状态转换图中的每个状态，将转入条件填到该状态所含每个μOPCmd所在行的相应列（＝状态的时间戳）中，结果如表B.1所示。如ex\_b的转入条件填到其所含5个μOPCmd的T2列单元格中。

表中，All表示对所有指令通用，(*x*)表示μOPCmd的值为*x*，| 及＋都表示“或者”，| 还表示相关μOPCmd的取值相同，各个节拍中没有使用的信号都为无效值。

表B.1 所有μOPCmd使用时间表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 时间  命令 | T0 | T1 | T2 | T3 | T4 |
| PCsrc | All(1) |  | beq(0)＋j(2) |  |  |
| PCWr | All |  | j |  |  |
| PCWrB |  |  | beq |  |  |
| IMRd | All |  |  |  |  |
| IRWr | All |  |  |  |  |
| RegAsrc |  |  |  |  | add|sub(1)＋ori|lw(0) |
| RegDsrc |  |  |  |  | add|sub|ori(1)＋lw(0) |
| RegWr |  |  |  |  | add|sub|ori|lw |
| Extctr |  | All | lw|sw |  |  |
| ALUAsrc | All(1) | All(1) | add|sub|ori|lw|sw|beq(0) |  |  |
| ALUBsrc | All(3) | All(0) | add|sub|beq(2)＋ori|lw|sw(1) |  |  |
| ALUctr | All(0) | All(0) | add(2)＋sub(3)＋lw|sw(0)＋beq(1)＋ori(4) |  |  |
| ALUOWr |  | All | add|sub|ori|lw|sw |  |  |
| MemRd |  |  |  | lw | lw |
| MemWr |  |  |  | sw |  |

·形成各个μOPCmd的逻辑表达式

对表B.1的每一行进行汇总、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，如：

PCsrc[1]＝T2⋅j，PCsrc[0]＝T0，PCWr＝T0＋T2⋅j，

RegWr＝T4⋅(add＋sub＋ori＋lw)，ALUAsrc＝T0＋T1，MemRd＝(T3＋T4)⋅lw，…

然后，使用组合逻辑电路，实现各个μOP控制信号的有效逻辑，即可得到所设计的μOP控制信号形成电路。

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B.5所示。其中，指令译码器同单周期CPU，时序信号形成电路如图B.4所示。



图B.5 支持7条指令的多周期CU

至此，CPU的设计全部完成。

## 附录C 主存的设计与连接

主机由CPU及主存组成，CPU设计完成后，主机设计的剩余任务就是设计主存、连接主存。

### C.1 主存的设计

主存通常组织为一个基本模块，因此，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，包括参数需求、操作需求，分析的内容为主存的设计要求、主存连接部件（如CPU或总线）的外部接口及传输协议。

本课程的主存设计要求为：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。

所设计的CPU外部接口及传输协议为：接口有2组，分别访问指令存储器、数据存储器，存储器按字节编址；每组接口中，地址引脚为30位，数据引脚（单向）为32位，控制引脚包含读/写/时钟3根信号线；每组接口中，访存操作采用常规传输方式，操作时延＜0.5个时钟周期（无需协议控制）。

由于CPU的数据引脚为32位、未设置数据掩码引脚、采用常规传输方式，故CPU的访存粒度为32位。由于主存直接与CPU连接，主存的数据引脚应为32位。

可见，主存的设计需求如下：

IMEM为4KB同步存储器，数据引脚为32位（单向）；支持常规传输方式的读操作，访问粒度为32位，访问时间＜0.5个时钟周期。

DMEM为4KB同步存储器，数据引脚为32位（单向）×2；支持常规传输方式的读、写操作，访问粒度为32位，访问时间＜0.5个时钟周期。

#### 2．模块设计

主存的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**

IMEM/DMEM的容量为4KB，数据引脚为32位，因此，地址引脚为log2(4KB/32b)＝10位；由于访问粒度＝数据引脚位数，因此，无需设置数据掩码引脚。

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、读使能引脚RD（可以缺省）、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。当访问粒度支持8位/16位时，还应设置数据掩码引脚DM3~DM0。

注意，控制信号可约定为高/低电平有效，上述的CS、WE为高电平有效。

**（2）内部电路设计**

IMEM/DMEM仅支持常规传输模式，且主存单元长度（8位）≠访问粒度（32位），因此，IMEM、DMEM应由并行访问方式的4体交叉存储器组成，每个存储体为1K×1B同步ROM、1K×1B同步SRAM。由于要求访问时间＜0.5个时钟周期，IMEM/DMEM的输出不应该与时钟信号同步（不锁存/直接输出）。

由于IMEM/DMEM未设置数据掩码引脚（访问粒度只有一种），因此，4体交叉存储器可以用1K×4B的单体存储器代替。注意，存储单元（长度＝4B）中存放数据/指令时的字节顺序，应满足大端/小端方式的要求。如01020304H在小端方式时存为01020304H，在大端方式时存为04030201H。

### C.2 主存的连接

主存需要通过连接电路与CPU连接，连接电路的设计需要基于CPU的外部接口及传输协议、主存的引脚信号进行。假设主存安排在CPU可寻址空间的低端。

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C.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C.1所示。



图C.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。

至此，模型机主机的设计全部完成。

## 附录D 支持不同访存粒度的BIU设计

BIU是CPU内部与外部的接口，是数据通路的子模块。

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、数据位数扩展（访存粒度＜数据宽度时）所需的格式转换，如大端/小端、所取8位/32位数据转换为32位数据；总线逻辑电路用于实现数据的传输过程，如总线传输协议的实现。

因此，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，其组成如图D.1所示。



图D.1 支持2种访存粒度的BIU

其中，EndianT用于实现数据存放方式所需的数据格式转换，ReadT、WriteT用于实现访存粒度＜数据宽度的读操作、写操作所需的数据格式转换。

**（1）EndianT的组织**

数据存放有大端、小端2种方式，对于每种方式而言，存、取时的字节顺序转换要求是相同的。因此，BIU中，读、写操作的数据端序处理，可以使用功能相同的EndianT来实现，如图D.1所示。

由大端、小端方式的存放要求可见，大端方式的EndianT内部逻辑为：颠倒输入信号的字节次序，小端方式的EndianT内部逻辑为：输出信号＝输入信号。

**（2）WriteT的组织**

访存粒度与数据宽度的关系有3种：小于、等于、大于。第3种情况下，WriteT及ReadT内部需设置缓冲队列，操作时的数据传送需采用突发传输方式，其组成较为复杂，暂不讨论。下面仅讨论访存粒度≤数据宽度时，WriteT及ReadT的组织方法。

对于写操作而言，当访存粒度＜数据宽度时，只能写入数据引脚上的一部分信号（数据）；为了只写入合法信号，BIU必须设置数据掩码引脚（如图D.1中的DataMask），以通知存储器信号线上哪些是合法信号。当访存粒度＝数据宽度时，数据掩码引脚上的信号全部有效，否则部分有效。

假设图D.1为支持MIPS32指令的BIU，DataL&S＝00/01/1\*表示数据为8位无符号数/8位有符号数/32位数，WriteT的输出引脚为DataTemp，故DataMask为4位。可见，WriteT的内部逻辑为：当DataL&S＝1\*时，DataMask＝1111，DataTemp＝DataOut；当DataL&S＝0\*时，DataMask＝0001<<*x*，DataTemp第*x*字节＝DataOut7~0，其中*x*＝PA1~0。

**（3）ReadT的组织**

对于读操作而言，当访存粒度＜数据宽度时，数据引脚上只有一部分为所读数据；为了提高操作速度，数据位置的指示可用BIU的DataL&S及PA1~0实现，故读操作可以不使用数据掩码引脚上的信号。

假设图D.1为支持MIPS32指令的BIU，DataL&S＝00/01/1\*表示数据为8位无符号数/8位有符号数/32位数，连接ReadT的EndianT输出引脚为DataTemp，ReadT的内部逻辑为：当DataLen＝1\*时，DataIn＝DataTemp；当DataLen＝00时，DataIn7~0＝DataTemp第*x*字节，DataIn31~8＝0，其中*x*＝PA1~0；当DataLen＝01时，DataIn7~0＝DataTemp第*x*字节，DataIn31~8＝DataIn7，其中*x*＝PA1~0。

BIU中，内部端口、总线逻辑电路的设计方法，与附录A、附录B完全相同。至此，支持不同访存粒度的BIU设计全部完成。