华中科技大学

硕士学位论文

基于MIPS指令集的RISC微处理器数据通路的设计与实现

姓名：刘宁

申请学位级别：硕士

专业：计算机应用技术

指导教师：曹计昌

20080603

摘 要

随着集成电路技术的发展，SOC 的设计方法越来越流行。基于 SOC 的方法进行嵌入式系统设计，可以显著降低开发成本，并且易于维护。

嵌入式系统中，微处理器是核心，决定了整个嵌入式系统的性能。MIPS 指令格式清晰、紧凑，采用 MIPS 指令设计微处理器可以简化体系结构的设计，并得到比较好的性能。最终的设计以 MIPS 指令集为基础，实现了五级流水 RISC 微处理器。

根据对 MIPS 指令集的研究，选取了待实现的指令，及指令寻址方式。通过对指令执行过程的详细分析，设计和实现了各逻辑功能模块,包括：存储指令和数据的存储模块、高速提供运算操作数的寄存器堆、完成操作数运算的算术逻辑单元、对 16 位操作数进行扩展的符号扩展单元。在完成各模块设计后，设计和实现了单周期 RISC 数据通路。

流水线是提高微处理器性能的重要方法，在对单周期数据通路设计的基础上，对流水线数据通路进行详细分析，设计了流水线寄存器，最终建立了基于 MIPS 指令集的五级流水线数据通路。

设计完成后，对流水线数据通路进行仿真，然后下载到 FPGA 开发板进行验证。最终的数据通路支持 34 条指令，主频达 40 M HZ。

关键字： 片上系统，精简指令集微处理器，流水线，数据通路

Abstract

With the development of integrated circuit technology, SOC design becomes more and more popular. The SOC-based approach to embedded system design reduces the cost of development significantly, it also makes maintainance easier.

The microprocessor is the core of embedded systems, it determines the performance of the entire embedded system. The MIPS instruction format is clear and compact; it can simplify the design of the microprocessor architecture, and achieves relatively good performance. The ultimate design is based on the MIPS instruction set, and it has five stages pipeline.

Based on the research of MIPS instruction set, instructions and instruction addressing has been chosen to implement. The detailed analysis of execution, the functional unit designed includes: the memory unit of instruction and data, the register files providing high-speed operation, the arithmetic logical unit, the sign extend unit. We implement the data path of single cycle risk processor after all the modules are designed.

As we know, the pipeline is the most important method to improve the performance of the embedded microprocessor. After completion of single cycle processor data path, we carefully analyze and design the pipeline processor data path and registers, and eventually realized the five stages pipeline processor data path.

After the design is completed, we simulate all the modules, and download it into the FPGA development board. The final data path of the design supports 34 instructions in total, and reaches its highest frequency 40 M HZ.

Key Words: SOC, RISC processor, pipeline, datapath

独创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除文中已经标明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到，本声明的法律结果由本人承担。

学位论文作者签名：

日期：　年　月　日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，即：学校有权保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权华中科技大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本论文属于　保密□，在_____年解密后适用本授权书。
　　不保密□。

（请在以上方框内打“√”）

学位论文作者签名：　　指导教师签名：

日期：　年　月　日　　日期：　年　月　日

1 绪论

1.1 课题背景、目的及意义

本课题是实验室预研课题‘ 可动态重构的双内核多片内操作系统的智能卡体系结构研究 ” 中的一个子课题，课题的目的是设计一种以 8 位微处理器为主内核，以 32 位微处理器为安全协处理器的双内核、多片内操作系统智能卡体系结构。论文完成了多片内操作系统智能卡安全协处理器数据通路的设计与实现。

随着集成电路生产工艺的不断提高，FPGA(Field Programmable Gate Array，现场可编程逻辑门阵列)芯片的集成度越来越高，整个嵌入式系统的芯片和微处理器能够下载集成到单片 FPGA 芯片上，芯片与微处理器的界限越来越模糊，为某些特定应用设计和开发嵌入式微处理器变得非常流行。设计从嵌入式微处理器体系结构出发，分析和给出了待实现的指令集，在此基础上设计了基于 MIPS（Microprocessor without interlocked piped stages，无内部互锁流水级的微处理器)指令集的单周期数据通路，并进一步对流水线寄存器和流水线数据通路进行分析与设计。最终设计和实现的嵌入式微处理器，能够运行常用 MIPS 指令，可在 FPGA 开发板上下载和验证，并能正确执行裁剪后的全部指令。

课题的研究目的在于：基于 MIPS 指令集，对微处理器数据通路进行分析与设计；对嵌入式微处理器流水线进行分析，设计微处理器的流水线数据通路；将实现的嵌入式微处理器在 FPGA 上下载和功能验证；为基于 FPGA 的多片内操作系统智能卡安全体系结构的研究提供基本的方法。

课题的研究意义是：有利于对精简指令集系统计算机体系结构的研究,由于并行性强，便于扩展和维护，精简指令集系统微处理器被广泛的应用；有利于掌握嵌入式产品的快速开发,基于嵌入式系统的电子消费品越来越流行，按照片上系统的设计方法，开展以嵌入式微处理器为核心的应用研究，具有一定的价值；对基于 FPGA 的嵌入式系统开发具有重要意义,FPGA 在各行各业中发挥着越来越大的作用,使用软核处理器能够大大降低设计和开发的难度，在某些场合，软核处理器可以完全替代传统的硬核处理器，从而能够将整个系统的大部分功能，集成到单片 FPGA 芯片上，系统更加紧凑，开发维护更加简单；基于 FPGA 的嵌入式系统开发对于本实验室是全新的课题，为实验室嵌入式系统的研究提供了一个新的方法。

1.2 国内外研究概况

本课题是设计与实现基于 MIPS 指令集的嵌入式微处理器，该处理器采用 RISC 架构，基于 FPGA 进行设计与实现。以下将从三个方面阐述相关的国内外研究现状：精简指令集系统计算机研究概况；嵌入式微处理器的研究现状；基于 FPGA 的嵌入式系统设计方法研究现状。

1.2.1 精简指令集计算机的研究概况

CPU 按照指令集可以分为两类：CISC[1](Complex Instruction Set Computer，复杂指令集计算机) 和 RISC[2](Reduced Instruction Set Computer，精简指令集计算机)。

CISC 处理器的主要特点是支持多种寻址模式，不同的数据类型，指令的长度各不相同，并频繁的访问外部存储。因为控制复杂，CISC 通常使用微程序的控制方法。CISC 处理器指令与高级语言语义上的差别小，需要的程序存储空间相对较少，但在设计时，需要增加大量额外硬件来标记不同类型的指令，控制起来也相对复杂一些，基于 x86 体系结构的 Intel 微处理器就是典型的 CISC 处理器，但新的 Intel 处理器也融合了大量 RISC 的设计思想。

计算机发展初期，CISC 的设计思想占据统治地位，CISC 指令系统意味着大量的复杂指令、不同的指令长度、繁杂的寻址方式、复杂的指令解码，随着计算机性能要求的提高，复杂指令在解码上浪费的时间越来越多，大部分的复杂指令很少被使用，却耗费了研发人员的大部分时间，计算机整体性能不高。在 CISC 处理器设计中，新的处理器必须向下兼容以前开发的软件，指令系统会越来越复杂。庞大的指令集延长了处理器的开发周期，调试和维护也非常的复杂[3]。

人们通过多年的研究发现，处理器运行时，多数时间都在执行简单指令，大部分的复杂指令都可以用一组简单指令来代替[4]，专注于简单指令的高效实现的 RISC 设计思想由此提出，设计一个简单的 RISC 处理器来满足特定应用变得越来越简单[5]。从 RISC 处理器的提出可以看出，RISC 注重简单指令的高效实现，和处理器整体性能的提高。

通过计算机专家的统计分析，计算机大部分时间运行的指令集中在 20%指令集之中，20%的指令承担了计算机 80%左右的工作，占用芯片面积最多的大部分复杂指令多数情况下被闲置，因此利用率不高，设计低效，基于 RISC 的设计思想[6,7]，可以简化硬件的设计，让硬件高效的执行简单指令，复杂指令依赖编译技术，由简单指令实现。

RISC 这一命名由加州大学伯克利分校 RISC 项目负责人 David A. Paterrson 提出，并随着 RISC 处理器研究的进一步深入，逐渐成为流行的处理器架构。RISC 简单而统一的指令格式优化了指令系统，程序的编译和执行速度更高，少的寻址方式使得处理器的结构更加清晰紧凑，设计和维护起来也更加容易。比较有代表性的 RISC 处理器有 IBM 与摩托罗拉等公司合作研发的 PowePC[8]，HP 公司的 PA-RISC[9]以及 SUN 公司的 SPARC[10]，其中影响最大的要数 MIPS 公司的 MIPS 架构微处理器。

相对于 CISC 来说，RISC 处理器有以下特点[11,12]：

1．将主要的精力放在那些经常使用的指令上，尽量简单高效的实现，对于某些指令，使用硬件实现。

2．采用等长指令，控制器设计简单高效。

3．采用大量的寄存器，使得大部分的操作都在寄存器间进行，提高了运算速度。

4．指令寻址方式相对较少。

5．由于指令集简单，采用硬布线控制代替微程序控制，速度更快，效率更高。

6．编译器更加容易产生高效代码。

7．采用流水线提高指令执行的效率。

8．RISC 对存储器操作进行了严格限制，使其操作尽量简单，只有指令 load 和 store 访问内存，而 CISC 访存类型指令很多。

9．RISC 微处理器结构简单，设计开发周期短，易于采用最新的技术，便于调试和维护。

对于其中的 MIPS 微处理器，有人评价其为高效的 RISC 体系结构中最优雅的一种体系结构，其总能保持最简捷的设计的同时取得最快的速度。MIPS 关注于通过流水线来提高处理器效率，其体系结构清晰简洁，可以很好的利用流水线来提高处理器的速度。

MIPS 体系结构对外开放，它只提出体系结构规范，而不具体去规定每条指令是如何的实现，任何人都可以根据 MIPS 的体系结构，设计和开发自身需要的 MIPS 微处理器，因此 MIPS 非常易于实现，效率很高，并且易于作高性能优化。这是设计选择 MIPS 指令集进行研究与实现的原因。

1.2.2 嵌入式微处理器研究现状

当前，随着嵌入式应用的流行，国内外在嵌入式微处理器体系结构设计方面的研究越来越多，形成了比较有代表性的几款微处理器，有 SUN 公司的 SPARC 系列微处理器，MIPS 公司的 MIPS 系列微处理器，ARM 公司(Advanced RISC Machines Limited，简称 ARM)的 ARM 系列微处理器和摩托罗拉公司的 PowerPC 等，这些微处理器都基于

RISC 体系结构，功耗低，设计简单，被大量应用到移动通信、汽车电子、航空航天等之中。

复旦大学专用集成电路与系统国家重点实验室开发了一款高速、低功耗的 32 位 RISC 处理器(FDU32)，该处理器实现了与 ARM7TDMI 指令和接口的兼容，采用 5 级流水线，整个元器件的消耗降低 11%，主频提高 67%，指令执行速度提高 87%，功耗降低 46%，该处理器在 FPGA 上进行了验证[13]。ECOMIPS 是一款基于 MIPS 的架构的微处理器，为 32 位结构，整个处理器占用资源很少，并且留下了很大的扩展空间用于控制和处理，多达 4GB 的寻址空间对嵌入式微处理器来说，绰绰有余，实现了比较少见的微程序控制，微指令存储在赛灵思自带的 Block RAM 上，最大的时钟频率达 64 M HZ[14]。为了高效的进行图像的处理，有人设计了一款 32 位的 RISC 处理器，支持大约 25 条指令，处理器核与一个在 200 k 门的开发板上实现的图像处理单元相连，该图像处理单元能对屏幕进行捕捉，对固定不变的噪声和黑点进行处理，对于键盘控制器、存储访问控制和 LED 和 LCD 以及相关接口使用软件或专用 IC 芯片实现[15]。为了更快，更好的满足市场的需要，今天的处理器设计必须具有更加灵活的处理器核，易于修改，并能快速的推向市场[16]。基于人脸识别的门禁系统被大量的应用，有人设计了基于人脸识别的处理器核，该系统能对面部区域进行很好的分割，使用一种更高效的类 C 硬件描述语言 Handel-C 描述，处理器核基于 MIPS 架构，按照 MIPS 五级流水线进行组织，整个系统由 MIPS 处理器核子系统、视频输入子系统和颜色识别及图像输出子系统组成，图像的去噪处理由 MIPS 程序实现，其他的全由硬件接口完成[17]。NPU1750A 是一款 16 位嵌入式微处理器核，该处理器核可以实现定、浮点运算，集成了中断控制、和定时器功能，定义了 16 种硬件表决中断器，支持 4 种数据类型，6 种指令格式，在 ATERA 的 FPGA 开发板上实现，仅占用 16 万逻辑门[18]。文献[19]提到在 FPGA 平台上开发了一款 32 位的整型微处理器，根据 CPU“ 软核 ”的思想按实际的应用进行了裁剪，除了必须的数据通路部件外，实现了指令缓冲，执行时钟频率可达 30 M HZ，支持的指令多达 150 条，占用 7%的系统逻辑资源，被应用到网络分类系统中。为了保证嵌入式微处理器设计的正确，有必要建立专门的处理器验证平台，在进行模块的测试和回归测试同时，进行更全面的系统级验证。有人设计了以 PowerPC 架构嵌入式微处理器“ 龙腾 R2 ”和嵌入式操作系统 VxWorks 为基础的复杂验证平台，该验证平台能够快速和有效的对微处理器核进行调试，极大的提高了验证的效率，并能发现 RTL 仿真不能发现的 corner case 问题[20]。在高性能嵌入式微处理器研究方向，国防科技大学计算机学院自行设计了 32 位的 RISC 高性能嵌入式微处理器“ 银河 TS-1 ”，在 ALTERA 公司的 FPGA 芯片上进行了功能仿真，主频可达

36.7 M HZ，除了具有嵌入式微处理器的计算功能外，提供了对向量运算的支持，并通过指令的动态调度技术，降低了阻碍流水线性能的数据冒险和控制冒险问题[21]。

随着人们对嵌入式应用性能要求的不断提高，芯片和微处理器的界线越来越模糊，对于某些特定的应用，将微处理器同专用芯片整合在一起很有必要，嵌入式系统设计将进入软核时代[22,23,24]。科技的发展和应用的要求的不断提高，软件编程已经不能满足嵌入式系统的要求，而硬件编程将事先描述好的 HDL 语言程序写到芯片内部，提高了性能[25]，符合嵌入式应用的特点。通过在原有硬件的基础上，加入软核处理器，使 FPGA 具有软可编程性。

1.2.3 基于 FPGA 的嵌入式系统设计方法

从 20 世纪 60 年代至今，数字集成电路的集成度不断提高，目前已经发展到超大规模集成电路阶段，集成度最高的电路规模已经达到上千万门。Xilinx 作为世界上最大的 FPGA/CPLD 生产商之一[26]，于 1985 年推出了现场可编程逻辑门阵列器件，它是一种新型的高密度 PLD，采用标准的 CMOS 工艺制造，具有密度高、编程速度快、设计灵活和设计可再配置等优点。随着 FPGA 芯片规模的增大和功能的增强，通过 HDL 进行描述，在 FPGA 上进行嵌入式微处理器的开发成为可能[27]。

以应用为中心的嵌入式系统设计得到飞速的发展，因为其快速和高效，在各行各业中得到了广泛的应用，与传统的 IC 设计相比，其功能更加强大，集成度也更高，可以按照需要，将数据采集，处理，信号输入，输出，集成到一起，作为一个特定功能部件完成整个系统功能。系统的开发周期缩短，调试和维护更加简单，开发成本较低。传统的嵌入式开发过程中，软件和硬件同时开发，然后到嵌入式系统中进行集成和测试，效率低下，往往到最后，才会发现设计的问题。而新的嵌入式设计以软硬件协同设计和协同验证为前提，极大的缩短了开发周期[28]。

对于大规模的嵌入式设计，自底向上的设计方法不再可取，必须在更高的层次进行设计，向下逐步细化，取精，实现，验证，并灵活的使用各种 EDA 开发辅助工具提高设计效率。

根据片上系统开发流程，嵌入式系统设计，可以按照以下层次进行[29,30,31]设计和开发，在每个层次分别进行仿真和验证，整个设计分为五个层次。

1．系统总体设计

设计开始时，必须有一个写好的设计规范说明，包括功能，时序，硅片面积，功耗，可测试性，故障覆盖率以及其他的指导设计的规范和标准。

2．RTL 设计

RTL 是介于逻辑级与算法级的一种描述方法，并没有严格的定义和界限，相对来

说，RTL 描述比逻辑级描述简单清晰，比算法级描述具体，在实际的系统设计中应用广泛。

3．逻辑设计

将功能需求进一步细化，也称为结构化设计，在嵌入式设计中，通常以 VHDL[32]、或 Verilog 进行开发，或者以硬件原理图或逻辑功能图等进行描述。

4．物理设计

当前面的语法和功能错误都被解决后，综合工具会去创建一个优化的布尔门级描述，一般来讲综合工具会除去大量的多余逻辑，对芯片面积进行优化。

5．测试交付

对实现的整个系统进行下载验证，与系统设计说明进行比较，当其完全满足规范和标准时，就能交付完成。

这种自顶向下的设计方法可以用图 1.1 进行表示。

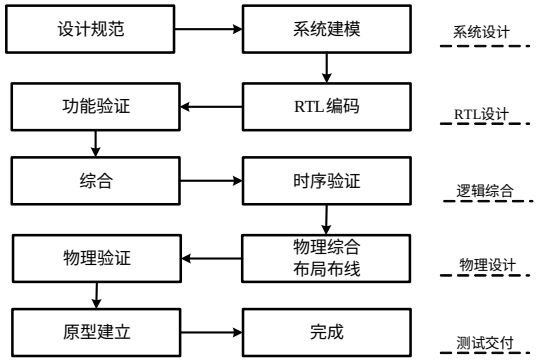


图 1.1　FPGA 开发过程

对于复杂的嵌入式系统，首先必须进行详细的设计，然后进行 RTL 级实现，经过一系列的验证后，在 FPGA 开发板上下载实现。对于设计中出现的问题，如设计方案不可行等，RTL 设计者必须将设计返还给系统设计阶段，进行重新设计；如若物理设计不能满足时序要求，就必须退回到 RTL 设计阶段，重新进行设计。

采用自顶向下的设计方法，使得初始设计的模块接口不需要大的修改，在设计的初期就充分考虑扩展性，并预留足够的扩展空间。这种新的开发方法，可以减少设计的重复工作，提高开发效率。

1.3　研究目标与研究内容

根据前面的分析和设计可知，课题的研究集中于基于 MIPS 指令集 RISC 处理器核的研究与设计，研究目标以及研究内容如下：

1．研究目标

在深入研究微处理器体系结构及 FPGA 设计的关键技术的前提下，对 RISC 处理器数据通路部件进行行为描述和仿真验证，在完成单周期微处理器数据通路的基础上，设计流水线数据通路，增加流水线寄存器，最终实现支持五级流水的微处理器数据通路，并支持特定指令集。

2．研究内容

（1）根据设计需求，确定要实现的指令集，作为微处理器数据通路的设计依据。

（2）根据裁剪的指令集设计与之对应的数据通路。

（3）对数据通路部件进行分析与设计。

（4）设计与实现五级流水微处理器数据通路。

（5）下载验证。将设计的完成的嵌入式处理器数据通路进行综合和实现，生成 bit 文件，下载到 FPGA 开发板进行验证。

设计参考了 MIPS32[33]指令集，在对 RISC 微处理器数据通路分析的基础上，实现了基于 MIPS 指令集的 32 位的 RISC 微处理器的流水线数据通路。

2 微处理器体系结构分析

本章主要对基于 MIPS 指令集的嵌入式微处理器进行总体分析与设计，从 MIPS 指令类型出发，分析了指令的寻址方式，选取待实现的指令，对 MIPS 指令的执行过程进行详细的分析。

2.1 嵌入式微处理指令类型及寻址方式

设计参照 MIPS 指令格式，所有指令均为等长指令，不同功能的指令格式不同，按照类型来分，可分为 R 型指令，I 型指令，J 型指令。其指令格式如图 2.1 所示。

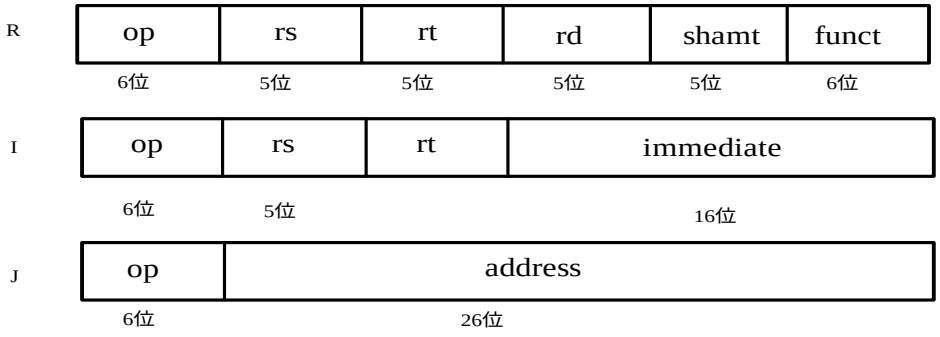


图 2.1 MIPS 三种不同指令类型

其字段意义如表 2.1 所示。

表 2.1 指令字段意义说明

字段名	字段意义说明
op	指令操作码
rs	源操作数 1
rt	源操作数 2 或目的操作数
rd	目的操作数
shamt	位移量，在移位运算中存储位移量
funct	功能码，当指令为 R 类型时，根据功能码判断指令的操作类型
immediate	当其为 I 型指令时为立即数，分支指令时，为跳转偏移量
address	跳转绝对地址，为指令 J 的指令格式

根据分析的嵌入式处理器指令类型，至少需要实现三种寻址方式：寄存器寻址，基址加偏移寻址，PC 相对寻址。

寄存器寻址：源操作数，或目的操作数为寄存器的指令，主要有 R 型指令，I 型指令，通过指明寄存器号来读取操作数。

基址加偏移寻址：数据传送类型指令 LW，SW，目的地址的产生通过寄存器与偏移量相加得到。

PC 相对寻址：目的地址通过 PC 与地址偏移相加得到，主要有分支指令。其原理图如图 2.2 所示。

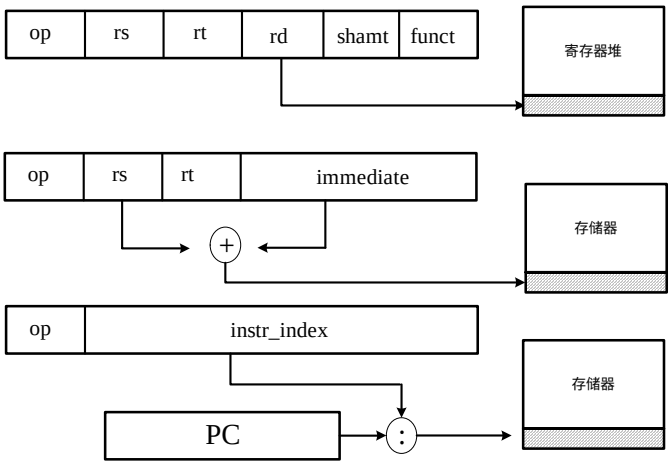


图 2.2　指令的寻址方式

2.2　嵌入式处理器指令集分析

指令集结构包括指令的长度，寻址方式等，设计的指令集采用 32-bit 编码，因为指令长度相同，能够降低译码的复杂度，减少译码延迟，并能很容易的使用流水线来提高处理器的效率。设计的嵌入式处理器只有 load 和 store 指令对存储系统进行访问。简单的指令寻址方式简化了嵌入式微处理器控制器和数据通路的设计。

处理器采用通用寄存器组结构。在这种结构中，指令可以从寄存器堆读出操作数，并将执行的结果回写到寄存器堆中，其中 R 类型和 I 类型都可能回写寄存器堆。

MIPS 指令集可以分为以下几类：

1．计算指令，对操作数进行计算，可能是 R 型或 I 型指令。

2．存取指令，用于存取存储器，指令格式与 I 型指令相同，意义不同。

3．分支和跳转指令，它可能改变程序的执行过程，可能是 I 型，由当前地址加当前 PC 得到新地址，也可能是 J 型，绝对地址由指令低 26 位给出。其中分支是进行程序的循环执行必须的，实现的指令集包含 6 种分支跳转，包含了绝大部分分支的情况，有利于提高程序的编译执行效率。

4．协处理器及其 CP0 指令，和专用指令，未予实现。

由于时间有限，根据实际情况，设计实现 R 型指令 18 条，I 型指令 15 条，J 型 1 条，总共 34 条，其中包含无符号指令 5 条，分支指令 6 条，涵盖了常用的算术运算与逻辑运算以及程序的分支和跳转，待实现的指令集如表 2.2 所示。

表 2.2　待实现的指令集

style	[31..26]	[25..21]	[20..16]	[15..11]	[10..6]	[5..0]	功能描述
R	op	rs	rt	rd	shamt	function	
nop	000000	00000	00000	00000	00000	000000	无操作
add	000000	rs	rt	rd	00000	100000	加
addu	000000	rs	rt	td	00000	100001	无符号加
sub	000000	rs	rt	rd	00000	100010	减
subu	000000	rs	rt	td	00000	100011	无符号减
and	000000	rs	rt	rd	00000	100100	与
or	000000	rs	rt	rd	00000	100101	或
xor	000000	rs	rt	rd	00000	100110	异或
nor	000000	rs	rt	rd	00000	100111	或非
slt	000000	rs	rt	rd	00000	101010	小于比较
sltu	000000	rs	rt	rd	00000	101011	无符号小比较
sll	000000	00000	rt	rd	sa	000000	逻辑左移
srl	000000	00000	rt	rd	sa	000010	逻辑右移
sra	000000	00000	rt	rd	sa	000011	算术右移
sllv	000000	rs	rt	rd	00000	000100	逻辑左移
srlv	000000	rs	rt	rd	00000	000110	逻辑右移
srav	000000	rs	rt	rd	00000	000111	算术右移
jr	000000	rs	0000000000		00000	001000	跳转至 rs 处
style-I	op	rs		imm			
bltz	000001	rs	00000	imm			小于分支
bgez	000001	rs	00001	imm			大于等于分支
blez	000110	rs	00000	imm			小于等于分支
bgtz	000111	rs	00000	imm			大于分支
beq	000100	rs	rt	imm			相等分支
bne	000101	rs	rt	imm			不等分支
addi	001000	rs	rt	imm			加
addiu	001001	rs	rt	imm			无符号加
andi	001100	rs	rt	imm			与
ori	001101	rs	rt	imm			或
xori	001110	rs	rt	imm			异或

表 2.2 （续）

slti	001010	rs	rt	imm	小于比较
sltiu	001011	rs	rt	imm	无符号小于比较
lw	100011	base	rt	offset	加载数据存储器
sw	101011	base	rt	offset	写数据存储器
J	op	instr_index			
j	000010	address			跳转至 address 处

2.3 流水线组织与分析

流水线概念来自工业生产，通过让单个工人熟练掌握单一的工作任务，来提高生产的效率。显然，每个工人单位时间内处理的任务并没有增多，但工人的熟练程度提高了，平均效率显著提高。

通过观察数据通路，很容易发现，每条指令在执行时，很多部件都处于空闲状态，因此，如何对每一时刻处于空闲状态的部件充分利用就是处理器的流水线。将多条指令送入数据通路，让它们并行执行。显然单个指令的执行过程和执行时间没有减少，而大部分的数据通路部件都在并行执行，单位时间内执行的指令数增加，处理器的效率提高[34,35,36]。

对于非流水线状态下，如果一条指令的执行分为 m 个阶段，那么执行一条指令就需要 m 个时钟周期，执行 n 条指令就需要 m×n 个时钟周期，而如果采用流水线，理想情况下只需要 m+n-1(假设 n>m)个时钟周期完成。当 n 远大于 m 时，流水线条件下，执行效率和吞吐量提高了近 m 倍[37,38]。

流水线处理器把一条指令的执行分为以下几个步骤，也称为级(stage)，每一级需要在一个周期内完成。每个时钟周期，流水线执行一条指令，m 级流水线，则可重叠执行的指令条数为 m 个，理想情况下的流水线的加速比接近 m，其计算如式(2.1)所示。

$$t = \frac{\text{非流水执行时间}}{\text{流水化执行时间}} = \frac{\mathrm{m} \times \mathrm{n}}{\mathrm{m} + \mathrm{n} - 1} \approx \mathrm{m} \tag{2.1}$$

同一时刻不同指令的不同阶段在流水线中被处理，如图 2.2 所示。流水线的设计需要根据执行特点平衡流水线每一级的执行时间，理想情况下流水线的加速比为流水线级数，但流水线级与级很难得到平衡，流水线的控制也需要一些额外的时间花销。设计按照 MIPS 指令的执行特点，对流水线进行了五级划分，理想情况下每一级的执行时间大体相同。

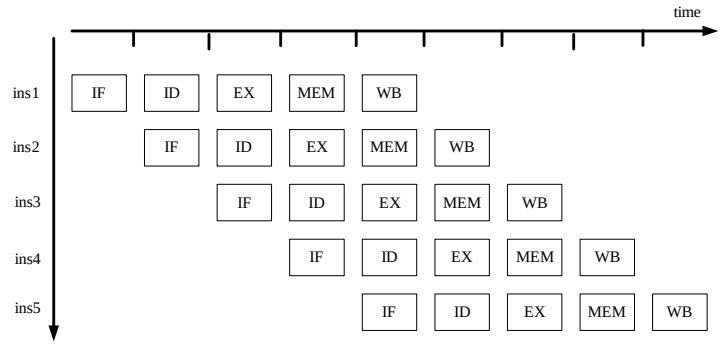


图 2.2 流水线指令的执行

根据 RISC 处理器的特点，可以将指令的分为以下几个基本操作：

Step1 – IF 取值

INS←Mem[PC] 根据 PC 从指令存储器取指令

New PC←PC+4 计算 PC 的新值

…

Step2 – ID 译码

A←Regs[$INS_{10..6}$] 取寄存器操作数 A

B←Regs[$INS_{16..11}$] 取寄存器操作数 B

Imm←extend(($INS_{15..0}$)$_{32}$)

控制信号←控制器

…

每次把两个操作数取出，可以不用，后面的阶段不处理

Step3 – EX 指令

Mem 指令：

ALUOutput←A+extend(($INS_{15..0}$)$_{32}$) 结果作为地址

Mem[ALUOutput]←Data 数据存入 Mem

R 型指令：

ALUOutput←A op B 对寄存器 A 和 B 进行计算

I 型指令:

ALUOutput←A op extend(($INS_{15..0}$)$_{32}$) 寄存器 A 和立即数扩展后的值计算

J 型指令：

ALUOutput←PC+(extend($INS_{15..0}$)$_{32}$)

if condition is true

New PC←ALUOutput

Step4 – MEM 访存

Mem 指令:

if load

MDB←Mem[ALUOutput]

else store

Mem[ALUOutput]←MDB

跳转指令：

if (conditon) then

PC←New PC

else

PC←PC+4

Step5 – WB 回写

Reg 指令：

Regs[$INS_{20..16}$]←ALUOutput

Imm 指令：

Regs[$INS_{15..11}$]←ALUOutput

Mem 指令：

Regs[$INS_{15..11}$]←MDB

不同类型指令的详细执行过程可用图 2.3 进行说明。

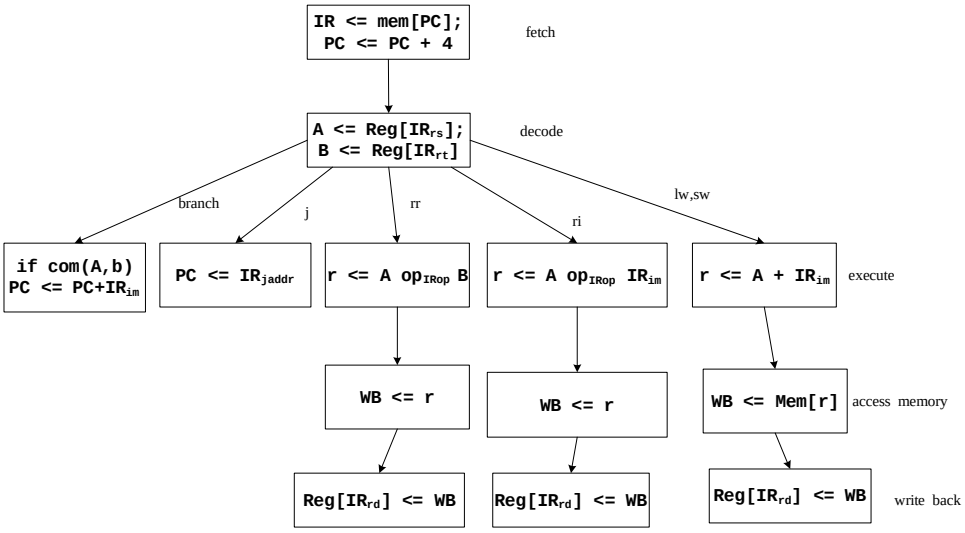


图 2.3 各指令操作执行过程

因此将流水线定为 5 级，既考虑到实际数据通路资源的合理分配，又让各个流

水线阶段的执行时间比较平衡，对于流水线来说，以下问题在流水线建立阶段必须认真考虑：

1．流水线技术不能提高单个任务的执行效率，它可以提高整个系统的吞吐率。
2．流水线中的处理速度的瓶颈在最慢的那一级。
3．多个任务同时并行执行，但同一时刻使用不同的数据通路资源。
4．其潜在的理想情况下的加速比等于流水线级数。
5．各个流水阶段所需时间不均衡将降低流水线的加速比。
6．流水线存在装入时间和清空时间延迟，其加速比比理想情况下低。
7．存在一些问题，会导致流水线阻塞。
8．必须解决数据冒险和控制冒险问题。

2.4 小结

本章对基于 MIPS 指令集的嵌入式微处理器体系结构进行了初步分析，在对指令集进行研究的基础上，给出了待实现的三种指令寻址方式，并对指令执行的详细过程，流水线的设计进行了分析，是流水线数据通路设计的基础。

3 单周期数据通路及部件设计

本章在对微处理器数据通路基本知识分析的基础上，对处理器中的各个功能部件进行了详细的需求分析和设计，并根据设计的数据通路部件建立了微处理器的单周期数据通路。

3.1 简单数据通路分析

要建立微处理器数据通路，首先需要一个存储部件存储程序指令，并能根据不同的地址，读出不同的程序指令。还需要将地址暂存，根据暂存的当前程序地址计算下一条指令的地址，在下一个时钟周期读出下一条指令。

读出的指令有不同的类型，要执行的操作也各不相同，必须对指令进行译码，得到指令的控制信号。因为嵌入式 RISC 处理器为通用处理器结构，需要一个寄存器堆存储参与运算的数据。有些参与运算的数据由低 16 位提供，需要对其进行位扩展，即为符号扩展单元的功能。

除此之外，还需要对操作数进行运算的算术运算单元和对数据进行存储的数据存储器。

为了更具代表性，包含尽可能多的数据通路部件，以 LW 为例，给出了 LW 指令的数据通路简图如图 3.1 所示。

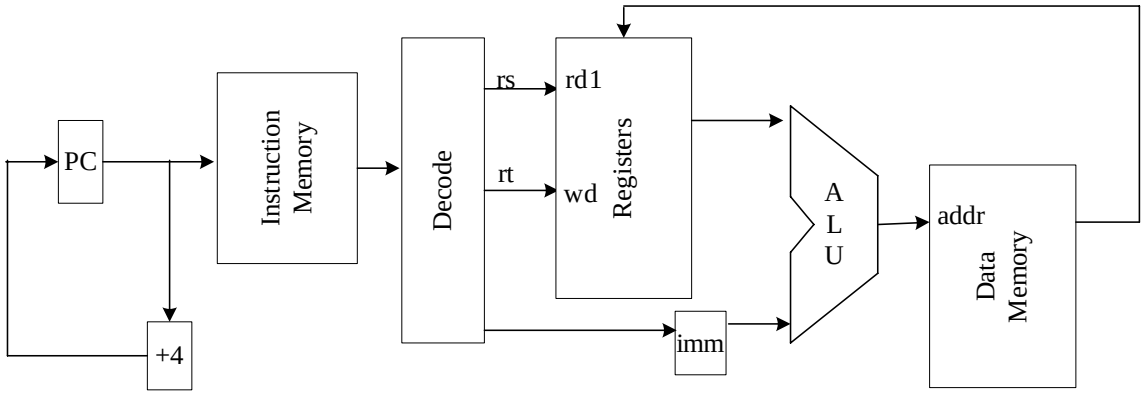


图 3.1 LW 指令的数据通路简图

下面将对数据通路的各个部件进行详细的分析与设计，并在上图的基础上，建立最终的、支持特定指令集的完整数据通路。

从图中可以看出，需要设计的功能部件有：程序地址寄存器、指令和数据存储器、译码控制器、寄存器堆和算术逻辑运算单元等。

3.2 算术逻辑单元设计

算术逻辑单元是微处理器中执行算术运算和逻辑运算的部件，其功能为按照程序指令，对操作数执行相应的运算。

3.2.1 加法器设计

加法器是算术逻辑单元中的重要逻辑部件，由全加器和进位信号产生部件组成。通过实例化两个半加器和一个或门可以实现一位全加器的功能。而进位信号可以采用并行进位的方法，也可以采用串行进位的方法产生，串行进位的由于效率很低已经很少被使用。

1．串行进位

通过将单个的全加器部件重复，可以产生多位加法器。进位信号一级一级传递，要组成 n(n>1)位的加法器，产生最终的结果需要 2n 个逻辑门延迟，速度不高，对于高速的加法运算，无法满足要求。

2．超前进位

为了减少延迟，实现更高的效率，需要缩短进位产生的时间，对于 1 位加法运算其进位如式(3.1)。

$$C_{out}=xy+(x|y)C_{in} \tag{3.1}$$

当 xy 为 1 的时，必然产生进位信号，一般称为进位产生函数，而当（x|y）为 1，则向高位传输低位的进位信号，称为进位传输函数，可以看成低位进位信号不经过计算，直接向高位进位，其中“|”表示或运算。对于 4 位超前进位函数关系来说，其进位的产生如式（3.2）~式（3.5)。

$$\begin{aligned} C_1&=G_0+P_0C_0 \\ &=G_0+P_0C_{in} \end{aligned} \tag{3.2}$$

$$\begin{aligned} C_2&=G_1+P_1C_1 \\ &=G_1+P_1G_0+P_1P_0C_{in} \end{aligned} \tag{3.3}$$

$$\begin{aligned} C_3&=G_2+P_2C_2 \\ &=G_2+P_2G_1+P_2P_1G_0+P_2P_1P_0C_{in} \end{aligned} \tag{3.4}$$

$$\begin{aligned} C_4&=G_3+P_3C_3 \\ &=G_3+P_3G_2+P_3P_2G_1+P_3P_2P_1G_0+P_3P_2P_1P_0C_{-1} \end{aligned} \tag{3.5}$$

其中 $P_n=X_nY_n$，$G_n=X_n|Y_n$，与进位产生函数 G_i的定义类似，根据四位一组，可以计算进位函数 G*和 P*及加法的第 n 位结果为 F_n，其中“^”表示异或运算，计算方法如式(3.6)~式(3.8)所示。

$$P^*=X_3X_2X_1X_0 \tag{3.6}$$

$$G^*=Y_3+Y_2X_3+Y_1X_2X_3+Y_0X_1X_2X_3 \tag{3.7}$$

$$F_n=X_n \wedge Y_n \wedge C_n \tag{3.8}$$

4 位超前链的逻辑电路图如图 3.2 所示，C_{-1}为低位的进位信号，根据式（3.2）~式（3.8），。相对其他逻辑芯片来说其功能更简单，占用逻辑门更少。

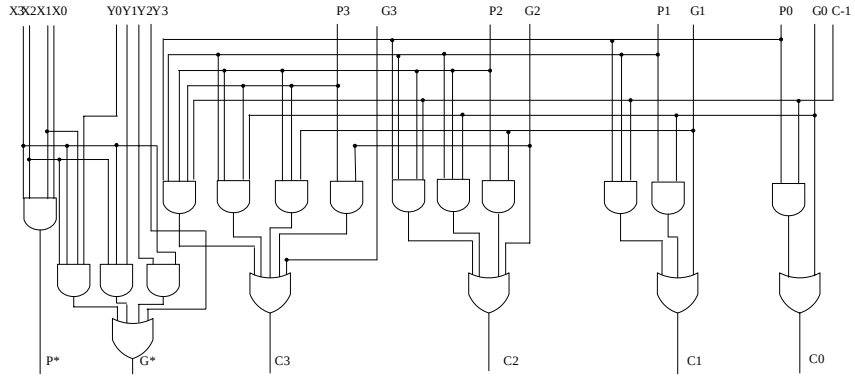


图 3.2　4 位超前进位逻辑电路图

以 16 位加法器为例，将 4 片图 3.2 逻辑部件串联，就可以组成 16 位先行进位加法器，其组间进位函数如式（3.9）~式（3.13）所示。

$$P^*=P_0\&P_1\&P_2\&P_3 \tag{3.9}$$

$$G^*=G_3+G_2P_3+G1P_2P_3+G_0P_1P_2P_3 \tag{3.10}$$

$$C_{n+3}=G_0+P_0C_{-1} \tag{3.11}$$

$$C_{n+7}=G_1+G_0P_1+P_0P_1C_{-1} \tag{3.12}$$

$$C_{n+11}=G_2+G_1P_2+G_0P_1P_2+P_0P_1P_2C_{-1} \tag{3.13}$$

根据上述运算，可以画出其逻辑电路图如图 3.3 所示。

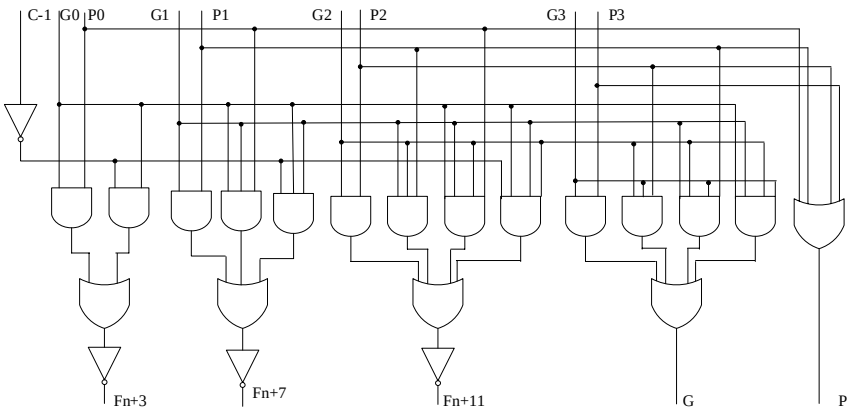


图 3.3　超前进位部件逻辑电路图

当处理器性能要求比较高的时，必须对相关的运算部件进行专门设计，按照逻辑电路图进行 VHDL 设计，从半加器、全加器出发，设计最终的加法器。由于时间关系，实际加法器的设计采用综合工具自带的先行进位加法器完成加法运算，设计将主要的精力集中在体系结构的逻辑实现上，而没有对数据的处理细节进行深入分析。加法器的设计可作为优化设计在后续工作中实现。

3.2.2 算术逻辑单元功能需求说明与实现

设计的微处理器需要实现 34 条指令，主要包括算术运算、逻辑运算、移位运算和比较运算。

对于单周期 RISC 的实现，分支指令 beq 和 bne 的处理信号在 ALU 中运算产生，而在流水线中，分支的处理提前到 ID 阶段完成。

没有设计专门的除法部件，因为除法器占用面积大，使用率不高，采用软件实现。为了实现 ALU，定义的操作码的如表 3.1 所示。

表 3.1 ALU 操作码功能定义

操作码	功能描述	操作码	功能描述	操作码	功能描述
00000	**输出高阻**	01001	A≠B	00110	srl
00001	A and B	01010	A = B	00111	sra
00010	A or B	01011	A + B	01000	slt
00011	A xor B	01100	A－B	01111	srav
00100	sll	01101	sllv	10000	sltu
00101	A nor B	01110	srlv	**others**	**输出高阻**

设计的 ALU 有确定的操作 17 种和一个默认运算，ALU 的操作码为 5-bit 宽，最多支持 32 个操作。操作码的信号可以直接从控制器得到，也可以从 ALU 控制器得到，为了简化译码控制器的设计，使整个数据通路更加紧凑和便于功能扩展，设计的 ALU 的操作码来自于 ALU 控制器。ALU 部件有两个输入(即 ALU 的两个操作数)，两个输出(运算的结果输出，和运算异常输出)以及 ALU 操作码输入。表中操作 sll，srl，sra 分别表示逻辑逻辑左移、右移、算术右移，而 sllv，srlv，srav 分别表示寄存器移位，即位移量存储在通用寄存器中，slt 和 sltu 分别代表有符号小于比较和无符号小于比较，nor 表示或非运算，由于 5-bit 操作码编码未用完，对于全部的默认操作，ALU 输出为高阻态。

x86 有一个专门的标志寄存器，存储零标志位、奇偶标志位、符号标志位、进位标志位和溢出标志位等，而 MIPS 与 x86 的一个重要的区别之一就是 MIPS 没有设置标志位寄存器，而是直接对寄存器的值进行比较来进行跳转的判断，比较的结果存

放在通用寄存器中。根据分析，可以得到 ALU 的实体描述如图 3.4 所示。

```
entity ALU is
    port(
        pOpCode         :in std_logic_vector(4 downto 0);          --ALU 操作码
        pIn1            :in std_logic_vector(31 downto 0);         --输入操作数 1
        pIn2            :in std_logic_vector(31 downto 0);         --输入操作数 2
        pException_out  :out std_logic;                            --异常输出
        pOut            :out std_logic_vector(31 downto 0));       --结果输出
end ALU;
```

图 3.4 ALU 的实体描述

ALU 根据输入的 pOpCode，对输入的操作数 pIn1 和 pIn2 进行指定的操作，结果通过 pOut 输出，当计算中出现异常的时，pException 把异常输出。表 3.1 的结构适合 switch case 语句实现。

可以看出，移位运算未采用专门的移位部件实现，由自带的综合工具实现，首先使用 switch case 语句，对 ALU 的操作码及其对应的运算进行枚举。sllv 操作码为 01110 处，移位运算的所移的位数由操作数 1 的低 5 位提供，所以最多可以移 32 位，其他的 sll，srl，sra 实现的方法基本相同，对加法运算来说，其操作码为 01011，当 2 个操作数的符号位相同，且结果的符号位与其中任一操作数的符号位不同时，即 pIn1(31) xor pIn2(31))='0' and (sO(31) xor pIn1(31))='1'，运算的结果出错，异常输出。其设计流程图如图 3.5 所示，按照不同的操作码选择对应的操作，最后检测是否发生运算溢出。

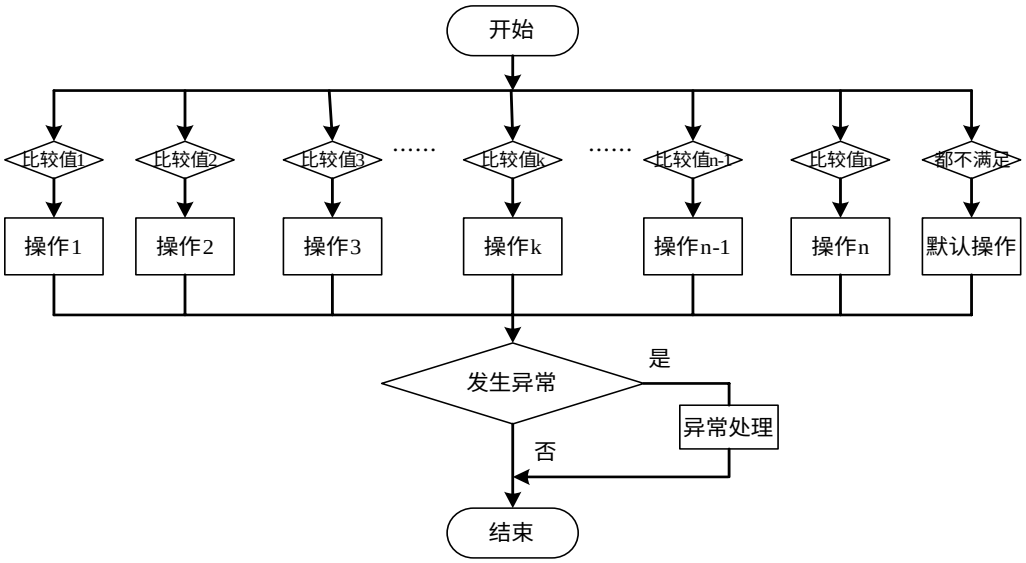


图 3.5 ALU 的设计流程图

3.3 寄存器堆

在 RISC 体系结构中，寄存器堆是数据通路的重要部件，为了提高指令执行的速度，将大部分的操作数放入寄存器中，只让少量的指令访问存储器，大部分的指令都从寄存器堆中读取操作数，然后进行运算，并将运算的结果回写入寄存器堆。寄存器堆的设计中，处理器的 32 个 32 位的通用寄存器通过指定相应的寄存器序号来进行读写。

R 型指令指明 3 个寄存器号，对于每一条指令都要从寄存器堆读取 2 个数据字，再写入一个数据字。为了实现从寄存器堆中读出一个数据字，寄存器堆需要一个输入信号指定要读的寄存器号和输出数据信号表示从寄存器堆读出的结果。为写入一个数据字，寄存器堆要有两个输入：一个指定要写的寄存器号，另一个提供要写的数据。寄存器堆总是根据输入的寄存器号输出相应寄存器的内容，而写操作由写控制信号控制，在写操作发生的时钟边沿，写控制信号必须是有效的。根据前面的分析，要完成寄存器堆的功能一共需要 4 个输入(3 个寄存器号和 1 个数据)和 2 个数据输出，其实体描述如图 3.6，省略了时钟信号和复位信号。

寄存器编号从 0 到 31，对于寄存器 0，不论如何进行操作，其保存的数据总是为 0，1 号寄存器用于保存编译器的暂时变量，31 号寄存器保存着 jal 指令的返回地址，其他寄存器都可用作指令的通用寄存器。

当 pReset 信号为 1 时，将寄存器中保存的数据全部清空。在时钟的上升沿读数据，在时钟的下降沿写入数据。

寄存器堆设计中，必须按照上升沿写，下降沿读，否则读出的数据可能为脏数据。

```
entity RegFile is
    port(
        pRegWE          : in std_logic;                          --寄存器写使能信号
        pReadRegister1  : in std_logic_vector(4 downto 0);       --读寄存器号 1
        pReadRegister2  : in std_logic_vector(4 downto 0);       --读寄存器号 2
        pWriteRegister  : in std_logic_vector(4 downto 0);       --写寄存器号
        pWriteData      : in std_logic_vector(31 downto 0);      --写寄存器数据
        pReadData1      : out std_logic_vector(31 downto 0);     --读数据 1
        pReadData2      : out std_logic_vector(31 downto 0));    --读数据 2
end RegFile;
```

图 3.6 寄存器堆的实体描述

设计时，使用一个 32×32 的寄存器组数据类型存储寄存器值，在时钟的上升沿

首先对寄存器初始化，每个寄存器存储的初始值为其寄存器号。当写信号有效，并且目的寄存器不是 0 号寄存器时，数据写入。而当是读寄存器时，如果要读的寄存器恰好是要写入的寄存器时，直接将写入的数值输出，否则，读取相应寄存器的值输出，寄存器堆设计流程图如图 3.7 所示。

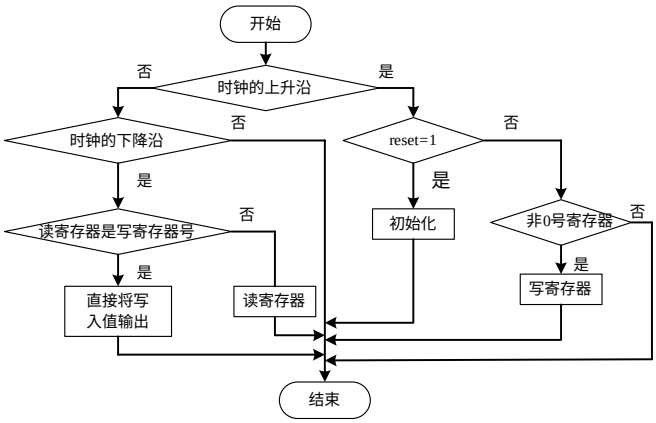


图 3.7　寄存器设计流程图

3.4　存储系统设计

本节主要是设计与实现数据通路的存储系统，它需要一个指令存储器，用于存储程序的指令，一个加法器来实现将地址寄存器的指令地址加 4，指向下一条指令的地址。对于 RISC 的微处理器设计，多采用哈佛结构，数据和指令分开存储，可以同时访问，保证了运行数据的快速和高效。在设计中，有待对指令缓存进行进一步的研究。

3.4.1　指令存储器

指令存储器的实现比较简单，因为指令存储器的数据在综合下载实现前，通过 coe 文件初始化进去，所以只需要 ROM 就可以实现其功能，设计使用 Xilinx 自带的 Coregen[39]生成，地址信号的长度由程序指令的长度决定，其实体描述如图 3.8 所示。

```
entity Imem is
    port (
        pReadaddr    : in  std_logic_vector(31 downto 0);    --地址信号
        pDataout     : out std_logic_vector(31 downto 0);    --输出数据信号
        pClock       : in  std_logic);                       --时钟信号
end Imem;
```

图 3.8　指令存储器实体描述

对于指令存储器的实现，需要用到 Xilinx 自带的 Block RAM 资源，按照 coe 文件的大小定义其读取深度，读取宽度定义为指令的长度 32 位，其输入有一个时钟信号，一个地址信号。该方法灵活性不高，但对于刚开始的开发者来说，灵活使用 Block RAM 可以降低设计的难度，因为指令存储器按照 32-bit 进行组织，因此，将 PC 的第 k(k<31)位到第 2 位作为指令存储器的读取地址，k 的大小为数据存储器的读取深度，在设计的过程中，可以根据存储指令的数量进行选择。并使用一个加法器来实现将地址寄存器的指令的地址加 4，指向下一条程序指令的地址。

3.4.2 数据存储器

除了指令存储器实现指令的存取之外，还需要数据存储器存储程序数据。数据存储器有两个使能输入端，读使能和写使能，当写使能有效时，数据输入，当读使能有效时，数据读出，一个数据输出端，和一个数据输入端，以及时钟信号输入和地址信号输入。其信号实体描述如图 3.9 所示。

数据存储器一般也被称为随机存储数据存储器，在物理和逻辑上分为内部数据存储空间和外部数据存储空间，用于中间结果和过程数据的存储。

由于设计使用 Coregen 生成的内部 RAM，没有使用到外部存储空间，采用字寻址的方式来简化设计的问题，因此没有实现字节和半字指令。

ISE 自带的 Coregen 生成工具生成的 ram，只能进行写使能控制，为了保证数据的安全，设计的数据存储器包括读使能控制，设计了一个专门的顶层模块，对其进行封装，在顶层模块中实现数据的安全访问控制。

```
entity Dmem is
    port(
        pAddress     :in     std_logic_vector(7 downto 0);     --地址信号
        pWriteData   :in     std_logic_vector(31 downto 0);    --写数据信号
        pOE          :in     std_logic;                        --读使能信号
        pWE          :in     std_logic;                        --写使能信号
        pReadData    :out    std_logic_vector(31 downto 0);    --输出数据信号
        pClock       :in     std_logic);                       --系统时钟信号
end Dmem;
```

图 3.9 数据存储器实体描述

当读使能有效时，数据读出，而当写使能有效时，数据写入，并且读写必须按照 ram 的特性进行，上升沿到来时，写数据，下降沿到来的时读数据。设计时，可以使用存储字的一个数组对数据进行存储，数据的读写都是在时钟的上升沿进行，当 pOE

为 1 时，读出数据，否则在 pWE 为 1 时，写入数据。其实现流程图可用图 3.10 予以说明。

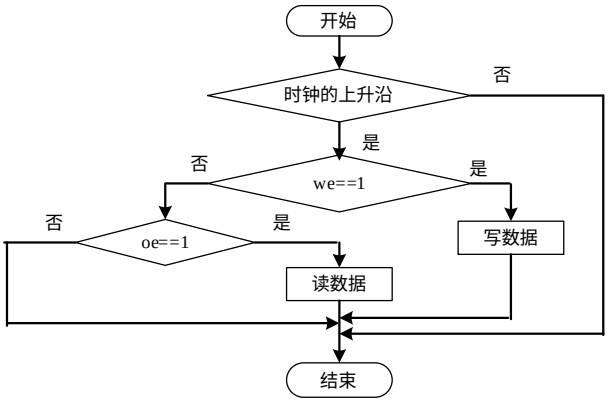


图 3.10　数据存储器设计流程图

3.4.3　层次化存储器组织

Cache 主要用于匹配外部存储器和 CPU 之间的速度的差异，现代的 CPU 系统也常常带有指令 Cache 和数据 Cache。Cache 能够实现期望的大容量，高速度以及低成本，设计只对 CPU 的一级 Cache 进行分析。

研究表明，处理器当前执行的程序往往局限在一小段程序范围内，即在一个较短的时间段内，CPU 对一小段程序进行频繁的访问，而对这一小段之外的程序访问较少，这称为程序的局部性原理，因此，设想可以将一小段程序预先调入高速缓存，减少处理器的访存次数，显著提高 CPU 的执行速度。

Cache 位于 CPU 与主存之间，以页或块为单位与主存进行数据交换，Cache 每个时刻只保存主存中最活跃的数据信息，是主存部分信息的一个副本，Cache 的基本结构如图 3.11 所示。

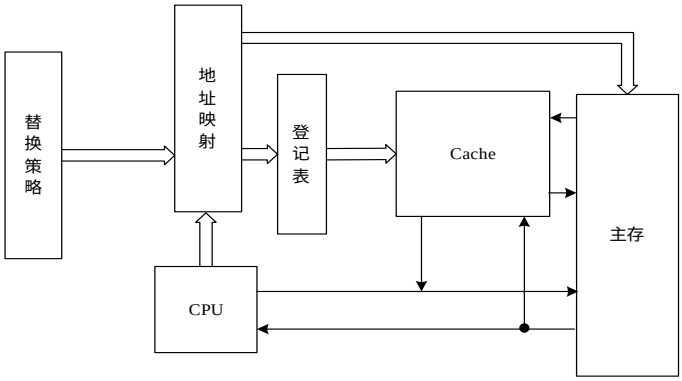


图 3.11　Cache 存储系统原理框图

将映射后的地址与 Cache 登记表进行比较，当其相等，标记位有效时，说明 Cache

命中，不相等时，Cache 失效，要访问的数据不在 Cache 中，需要从主存调入。当 Cache 命中时，不需要对内存执行操作，当 Cache 不命中时，CPU 从内存中读取需要的信息，同时将信息存入 Cache，如果 Cache 已满，就需要根据替换算法，用新的块替换掉 Cache 原有的块，传统的 Cache 块替换算法有 LFU(least frequency used，最不常用)和 LRU(least recently used，最近最少使用)，LFU 利用数据的访问频率作为替换思路，有利于数据的总体命中率，但不利于数据的突发访问，LRU 利用数据最后一次的访问时间，能很好的适应数据访问的变化，但只是访问时间的局部优化，并不能代表数据的长期访问特征。

任意时刻都必须保持 Cache 数据与主存数据的一致，常用的方法有通写法和回写法。通写法就是发生数据修改时对内存和 Cache 同时写，当对同一个内存单元进行多次的修改时，存在不必要的多次写内存，降低了访问的速度。而回写法在 Cache 中的内容被替换时才将修改写回内存，操作数据较快，可能因为内存中的数据未及时的修改而产生数据的不一致。

Cache 与主存的映射方式主要分为直接相联映射方式，全相联映射方式和组相联映射方式。

1．直接相联

就是将主存中的一个数据块，唯一映射到 Cache 中的一个位置，假设 Cache 的大小为 b，主存大小为 c，则主存中的第 i 块数据映射到 Cache 中的位置 p 由式（3.14）算出。

$$p=c \bmod b \tag{3.14}$$

可以理解为将大小为 c 的主存按照 Cache 的大小分成大块，每个大块的第 n 个小块，映射到 Cache 的第 n 小块处。

2．全相联

全相联就是主存的所有块可以映射到 Cache 中任意块上，这种方法使得 Cache 发生冲突的几率变小，因此 Cache 的利用率比较高，但 Cache 登记表的规模过大，价格过于昂贵，也降低了判断命中的速度。

3．组相联

组相联是直接相联和全相联的折中，将主存按照 Cache 的大小进行分区，Cache 和主存按同样的大小分成相同的组，组内进一步划分为块，主存的组与 Cache 的组之间进行直接相联，而组内使用全相联连接。对于下一阶段工作中的 Cache 的实现，拟采用组相联的方式实现，因为该方案设计简单，性价比高。

地址按照标志位，索引和块内偏移进行划分，因此 Cache 的大小也必须为 2 的倍

数，对于 32 位的指令系统计算机，一个字大小为 32 个比特，因此，当块以 32 比特组织时，外部存储器以字节编址时，字节偏移占 2 位，寻址四个字节。假设 Cache 的大小位 2^k 块，则，索引位为第 2 到 k+1 位，k+2 到 31 位为标记位，标记位用来比较 Cache 块是否命中，还需要一个位表示 Cache 是否有效，因此对于上述的分析，就可以算出 Cache 的位数为 $2^k \times (1+(31-k-2+1)+32)$位。

可以很容易发现，数据 Cache 和指令 Cache 存在访问主存的冲突，因此在主存和 Cache 之间需要加入一个读写的接口如图 3.12 所示。

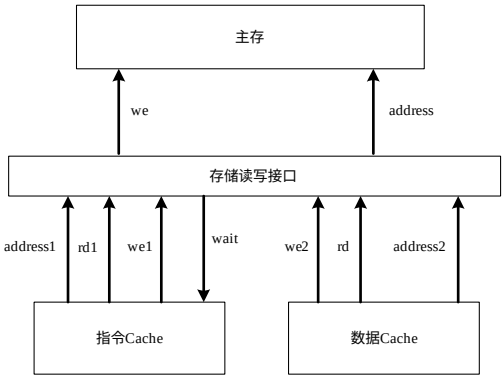


图 3.12 Cache 与主存的存储接口

设计中，数据 Cache 访问内存的优先级高于指令 Cache，在发生内存冲突时，首先满足数据 Cache 的读写，指令 Cache 的 wait 信号有效，等待数据 Cache 读写完成。

数据 Cache 和指令 Cache 一样，都是为了提高存取数据的速度。数据 Cache 有两个使能输入端，即读使能和写使能，当写使能有效时，数据写入 Cache，当读使能有效时，数据读出，一个数据输出端，和一个数据输入端，以及时钟信号输入和地址信号输入等。在设计中，为了简化问题，所有存储器都按照字进行存储。设计对层次化存储器及其存储 IO 控制系统进行了简单的分析，有待下一步设计予以实现，来达到指令和数据缓冲的目的，就可以很容易的实现对字节的读写，实现 lb，lh 等字节和半字指令。

3.5 程序地址寄存器

MIPS 没有 x86 意义上的程序计数器 PC，因为 MIPS 是基于流水线结构的微处理器，程序计数器在同一时刻有多个值。

程序地址寄存器负责暂存指令地址，同时在时钟的下降沿到来时，将指令送 ALU 控制器，控制器和寄存器堆等。

指令的地址必须保存在程序地址寄存器（PC）中。同时使用一个加法器来增加

PC 的值(+4)，使其指向下一个指令的地址，实现指令对齐。由以上分析，可以知道为了准备好下一条要执行的指令，必须将程序地址寄存器的值增加，使它指向 4 个字节后的下一条指令的地址。其实体描述如图 3.13 所示。

```
entity PC is
    port(
        pReset    :in std_logic;                              --复位信号
        pClock    :in std_logic;                              --时钟信号
        pHazard :in std_logic;                                --冒险检测信号
        pIn       :in std_logic_vector(31 downto 0);          --输入程序地址
        pOut      :out std_logic_vector(31 downto 0));        --输出程序地址
end PC;
```

图 3.13 程序地址寄存器实体描述

pReset 为复位信号，当其有效时，pOut 输出 0，当时钟的上升沿到来时，将输入的地址信号 pIn 输出到 pOut，pHazard 为冒险检测信号，当其为 1 时，保持 pOut 输出不变。其实现流程图如图 3.14 所示。

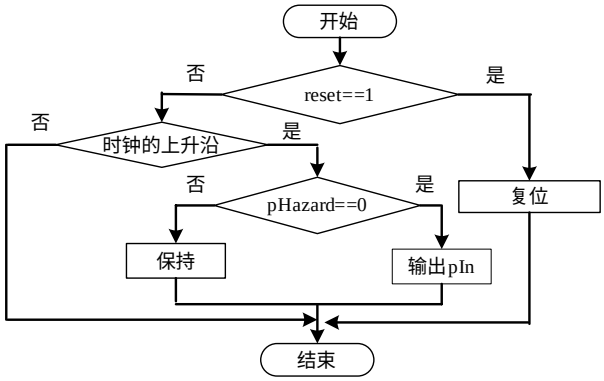


图 3.14 程序地址寄存器实现流程图

对于寄存器数据，一般由 D 锁存器锁存，32 位地址寄存器需要 32 个 D 锁存器，每个锁存器保存一位数据。D 锁存器在时钟的上升沿将输入端数据输出，并将数据保持到下一个上升沿到来时，它是触发器的一种，能够将数据延迟一个周期输出。用 32 位锁存器构成程序地址寄存器如图 3.15 所示。

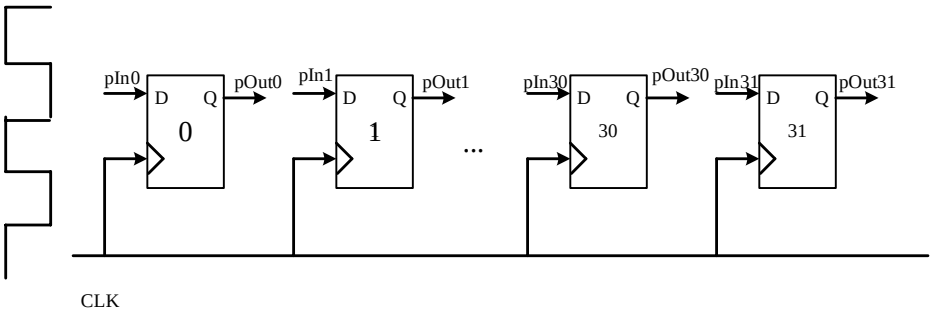


图 3.15　32 位程序地址寄存器结构图

3.6　符号扩展单元

MIPS 指令分为 R 类型指令，I 类型指令和 J 类型指令，对于 I 类型指令，其中的一个操作数来自于寄存器堆，另一个数来自于指令的低 16 位，ALU 为 32 位的运算单元，需要对 16 操作数进行位扩展，MIPS 指令分为无符号指令和有符号指令两种，符号扩展单元需要根据指令的类型选择符号扩展或无符号扩展，其实体描述如图 3.16 所示。

```
entity SignExtend is
    port(
        pSext       :in std_logic;                          --符号扩展选择信号
        pIn         :in std_logic_vector(15 downto 0);      --16 位立即数输入
        pOut        :out std_logic_vector(31 downto 0));    --扩展结果输出
end SignExtend;
```

图 3.16　符号扩展单元实体描述

pSext 为符号扩展选择信号，当其为 1 时，进行无符号扩展，当其为 0 时，进行符号扩展，pIn 和 pOut 分别为输入输出信号。以立即数 0x8001 为例，其有符号扩展的结果为 0xffff8001，无符号扩展结果为 0x00008001。

3.7　单周期 RISC 设计与实现

对于单周期的数据通路，所有指令都在时钟的边沿被取出和执行，在一个时钟周期内完成。在实际的处理器设计中很少被用到，因为其时钟频率低，单位时间的硬件花销大。但对于处理器设计来说，单周期处理器设计是理解处理器设计的基础。

下面将按照不同的指令类型，逐级建立最终的单周期处理器的数据通路，最终的

数据通路能够支持全部的 R、I、J 类型的指令。

3.7.1 R 型指令的单周期数据通路

以 R 类型指令 add $4,$2,$3 为例,该指令表示将寄存器$2 和$3 相加，结果送寄存器$4，以此为例，可以得到 R 类型指令的单周期数据通路，R 类型指令执行步骤如下：

1．从指令存储器中取出该加法指令：0x00641020。

2．指令译码，得到指令要执行的操作为 add $4，$2，$3。

3．读取寄存器$2，$3，得到操作数 1 和操作数 2。

4．执行指令，即将寄存器$2，$3 存储的操作数相加。

5．回写结果到寄存器$4。

从上面的分析可知，需要将译码的得到的 rs，rt，rd 分别送寄存器堆的读端口 1、读端口 2 和写寄存器端口，读出的寄存器数据送到 ALU 参与运算，根据分析，可以组成 R 类型指令的简单数据通路简图如图 3.17 所示。

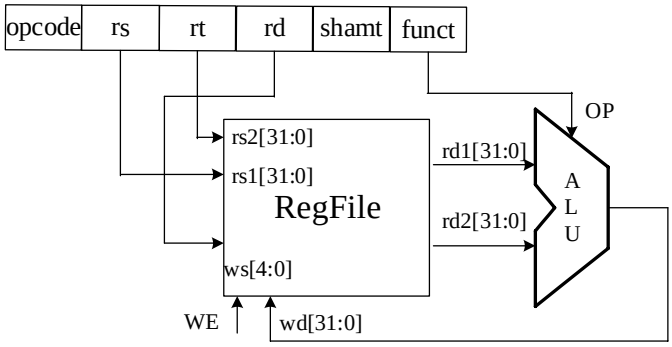


图 3.17 R 类型指令单周期数据通路简图

R 型指令中，包含一组特殊的指令，就是 srl，sra，sll 等移位指令，对于移位指令来说，移位的位数由指令的 shamt 字段得到，因此，ALU 进行运算时，需要根据指令的不同类型，选择指令的不同的字段，可以在 ALU 的操作数 1 处加上一个多路器，用于实现寄存器堆读数据 1 和 shamt 字段的切换，该部分设计会在最终的数据通路图中说明。

3.7.2 I 型指令单周期数据通路

以指令 addi $4，$2，100 为例，这条指令为 I 型指令，寄存器$4 为 I 型指令的 rt 字段，$2 为 I 型指令的 rs 字段，无 rd 字段，运算的结果送 rt 寄存器，因为 I 指令带 16 位的立即数字段，而 ALU 执行 32 位的运算，需要对 16 位的指令立即数进行位扩展，扩展的方法有两种，有无符号扩展和有符号扩展，需要根据指令的类型

选择位扩展的方法，因此可以组织 I 类型指令数据通路简图如图 3.18 所示。

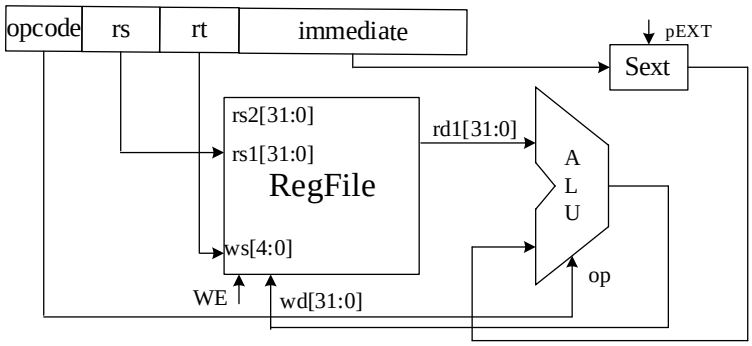


图 3.18 I 型指令单周期简单数据通路图简图

上面分别讨论了 R 类型指令的单周期数据通路和 I 型指令的单周期数据通路，但对于一个处理器设计，不可能为每种类型指令单独的设计数据通路，既耗费大量的资源，通用性也较差，整合后的数据通路会在后面图中说明，解决的办法就是增加多路选择器。

3.7.3 M 型指令单周期数据通路

对于 M 型指令，有存字指令 SW 和取字指令 LW，分别实现对数据存储器的写入和读取，下面以 LW 为例进行 M 型指令的数据通路分析和设计。对于 M 类型指令 LW $4，100($3) 来说，其操作可按照以下步骤进行：

1．从指令存储器中读出指令。

2．进行指令译码，M 型与 I 型指令格式类似，都由 op，rs，rt 和低 16 位字段组成，对于 I 型指令，低 16 位为立即数字段，对于 M 型，为地址偏移。在前面的指令集中将其归于一类，但由于涉及的数据通路部件不同，在此分开讨论。

3．读寄存器$3 的值。

4．计算访存地址：100+$3。

5．将对应地址的内容写入寄存器$4。

根据 M 类型指令格式和 M 类型指令的操作序列，我们可以在 I 类型指令的基础上实现 M 类型指令的数据通路简图如图 3.19 所示。

如图 3.29 所示，首先将指令的 rs 字段和 rt 字段送到寄存器堆的读端口 2 和写端口处，并将偏移量字段使用 Ext 部件进行 32 位扩展，扩展的结果通过一个多路器送到 ALU 的操作数 2 处，ALU 计算得到的地址值直接送数据存储器的地址端口，数据存储器写入的数据由寄存器数据 2 提供。数据存储器将读取的数据回写到寄存器堆中去。

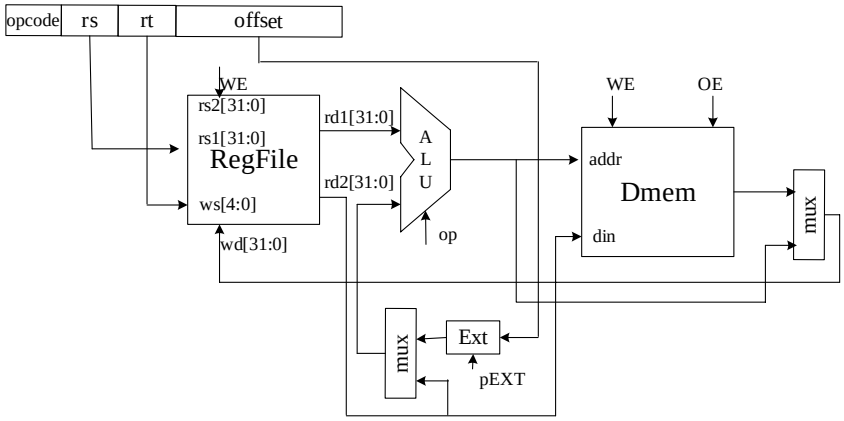


图 3.19 M 类型指令数据通路简图

3.7.4 完整单周期数据通路

前面重点分析了单周期 R 型指令，I 型指令以及 M 型指令数据通路。在现有创建的 R 类型，I 类型和 M 类型指令的数据通路的基础上，加入指令预取单元，就可以组成比较完整的单周期数据通路。

对于三种类型的指令，R，I，J 指令中的 J 类型指令，上一节未进行讨论，因为这些跳转指令不涉及到运算通路，只对新指令地址的计算和取指有影响，将在最终的数据通路中反映。当未发生分支与跳转时，取指部分将当前的 PC 值加 4，作为新的程序地址，存入程序地址寄存器。对于分支与跳转指令则各不相同，其新地址的计算过程如下：

1．无分支或跳转发生时，新的地址由当前 PC 加 4 得到。

2．分支指令，在条件得到满足时，跳转到新的地址，新的地址的由当前的 PC 值与跳转指令的低 16 位即偏移量字段进行移位加得到。

3．对于 J 类型指令来说，当其被执行时，进行无条件跳转，新的地址通过低 26 位的偏移量左移 2 位，用当前 PC 的高 4 位补齐得到。

4．而对于 JR 类型的跳转指令，新的地址直接由寄存器堆的读数据端口 1 得到。

对于四种不同的新地址产生方法，需要根据指令执行的不同情况进行选择，对于 PC+4 产生的地址，为默认新地址，当无分支和跳转及异常发生时，将其作为新的地址。而分支指令只在条件满足时，才改变执行过程，因此可以将比较和计算的结果作为该地址的选择信号。对于 J 和 JR 指令，当其执行时，进行无条件的跳转，因此可以通过控制器引出跳转控制信号，作为 J 类新地址的选择信号。多路器就可以满足上述的功能，通过给上述的数据通路，加入适当的多路器，就可以得到最终支持 R，I，J 全部类型指令的数据通路。

上面已经对不同类型指令的数据通路进行了简要的分析，根据上面的分析，我们可以进一步细化，得到最终的完整的单周期数据通路的详细设计图。通过给上述的指令的执行通路加入取指和译码部分，就是完整的数据通路图，在单周期数据通路设计中，每条指令只需要一个时钟周期执行，控制单元只需要通过指令译码，给各部件发操作控制信号，就可以完成整个数据通路的执行。并在当前指令执行完成后，让指令地址寄存器指向下一条指令。因此，可以根据前面的分析，设计完整的单周期数据通路图，控制信号由控制器给出，由控制模块设计部分完成，完成的数据通路图如图 3.20 所示[11]。

所有的指令从程序地址寄存器指向的指令存储器处开始执行，由指令的字段指明指令的操作数。

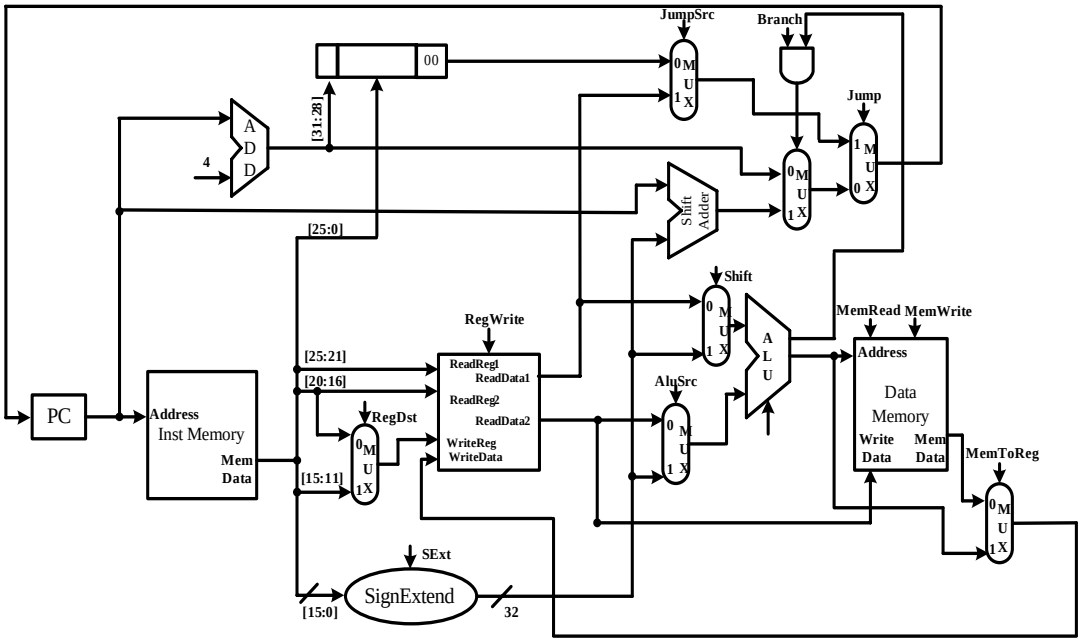


图 3.20　单周期不带控制信号完整数据通路图

整个数据通路通过程序地址寄存器提供指令的地址，从指令存储器中读取指令，然后对指令进行分析，根据指令不同字段从寄存器堆中读取操作数作为运算单元的输入，运算完成后，将计算的结果送回寄存器堆或将计算的结果作为数据存储器访问的地址，对数据存储器进行读写。对于下条指令地址，源操作数，目的寄存器号等，都可能来自于不同的指令字段或部件，需要给相关的数据通路加入多路选择器，进行多路选择，多路选择器的控制信号由控制部件、或运算部件给出。对图 3.20 的单周期的数据通路部件说明如表 3.2 所示。

表 3.2 单周期数据通路部件说明

部件名称	部件说明
程序地址寄存器	用于暂存程序地址
指令存储器	保存程序的指令，由 Coregen 生成的 ROM 实现，其详细的设计说明参见存储系统设计
跳转地址计算器	由当前 PC 的高 4 位与跳转指令的低 26 位左移 2 位组合而成
加法器	固定每次自动+4，用于在下一个时钟周期，产生新的程序地址
符号扩展单元	将指令低 16 位扩展为 32 位，sEXT 决定进行符号或无符号扩展
寄存器堆	取运算的操作数及保存回写的运算结果
移位加法器	分支需要进行移位加法得到跳转的新地址
ALU	按照不同的操作指令需要，执行不同的运算，输出结果
数据存储器	保存程序运行的数据，load 和 store 指令需要读写数据数据存储器，其详细的设计说明参见存储系统设计
Branch 多路器	选择将移位加得到跳转地址还是 PC+4 作为新的地址
JumpSrc 多路器	选择将寄存器读出值，还是 J 指令的跳转地址作为新地址
Jump 多路器	选择是否将跳转地址计算器的值作为新的地址
RegDst 多路器	选择目的寄存器取自指令的 rt 字段还是指令的 rd 字段
Shift 多路器	选择操作数从寄存器读端口 1 读取，还是取自指令低字节扩展
AluSrc 多路器	选择操作数从寄存器读端口 2 读取，还是来自指令低字节扩展
MemToReg 多路器	选择回写数据来自存储器的读取数据，还是运算器的计算结果

对于不同类型的指令，其在数据通路中的执行不同，控制也各异，控制部分设计不予考虑。设计只讲述对于不同类型指令其数据通路的执行。

下面以指令为例，对不同类型指令的详细执行过程，进行分析：

1．以 add 指令为例，其 OpCode 为 000000，FuncCode 为 100000，以 add $4，$2，$3 为例。

（1）取指和 PC 增加

INS=IMEM[PC]；

PC=PC+4；

（2）读寄存器

RegData1=Reg[rs:=2]；

RegData2=Reg[rt:=3]；

（3）执行 ALU 运算

Result=RegData1+RegData2；

（4）将计算的结果写回寄存器堆

Reg[rd:=4]←Result；

2．对于 lw，其 OpCode 为 100011，如 lw $5，100($4)

其执行过程如下：

（1）取指和 PC 增加

INS=IMEM[PC]；

PC=PC+4；

（2）读寄存器

RegData1=Reg[base:=4]；只有一个寄存器被读取

（3）执行 ALU 运算

Result=RegData1+zero_extend(INS[15..0]:=100)；

（4）读存储器

Data=DMEM[Result]；

（5）将数据写回寄存器堆

Reg[rt:=5]=Data；

3．sw，beq 与 lw 格式类似，sw 完成 lw 相反的功能，不在此一一阐述，下面以 beq 指令 beq $4，$5，offset 为例，对其进行分析，其操作码 000100。

（1）取指和 PC 增加

INS=IMEM[PC];

PC=PC+4

（2）读寄存器及计算分支地址

RegData1=Reg[base:=4]；读取 4 号寄存器值

RegData2=Reg[rt:=5]；读取 5 号寄存器值

Result=PC+(zero-extend)(INS[15..0]<<2)；由移位加法器计算

（3）执行 ALU 运算

Result=RegData1-RegData2；

if Result==0 then

branch=1；

（4）根据 branch 信号修改 PC 的当前值

if branch==1 then

pc=pc+Result；

else PC=PC+4；

4．对于 J 类型，对于指令 j，其 OpCode 为 000010，如 j offset， J 指令使用 PC 的低 26-bit 位左移 2 位替换 PC 的低 28 位，作为新的程序地址，其计算示意图如图 3.21 所示。

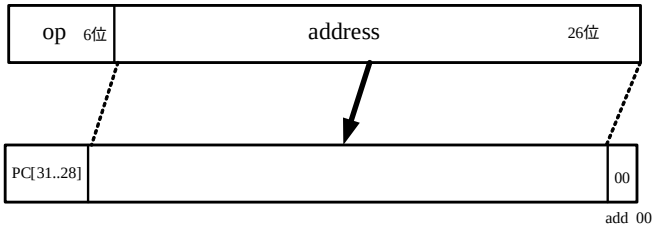


图 3.21 J 指令地址计算示意图

在现实的生活中，单周期的微处理器数据通路很少被使用，因为所有的指令与最慢的指令采用相同的时钟周期，处理器速度不高，数据通路部件使用率较低，每个时钟周期，每个部件只能被使用一次。最终实现的带控制信号的完整单周期数据通路图见附录 1，设计和完成了图中白色模块及实线部分。

3.8 小结

本章对数据通路部件进行了详细的设计，对 RISC 微处理器各个部件进行了详细的逻辑功能分析，在详细分析的基础上，对各个部件予以实现。在完成各个部件设计之后，对各类指令的数据通路进行了详细的分析，并以此为基础，设计了最终的单周期 RISC 嵌入式微处理器的完整数据通路，支持前面分析的 34 条 RISC 常用指令。

4 流水线数据通路设计

在单周期数据通路的执行中，每个执行中的指令独占数据通路资源，所有的数据通路资源按照顺序进行执行，每个数据通路部件的输入都直接来自于其他部件。流水线设计是 RISC 处理器设计的基础，也决定了处理器速度和效率，本章对流水线数据通路进行分析和设计。

4.1 建立流水线数据通路

通过在处理部件之间增加暂存数据的寄存器，使指令的执行可以共享数据通路部件，将数据通路进行分割，在指令的各个执行阶段，保持各条指令在各个阶段的执行值。方法是在图 3.1 的虚线部分加入寄存器，这种寄存器通常称为流水寄存器，增加流水线寄存器的数据通路简图如图 4.1 所示。

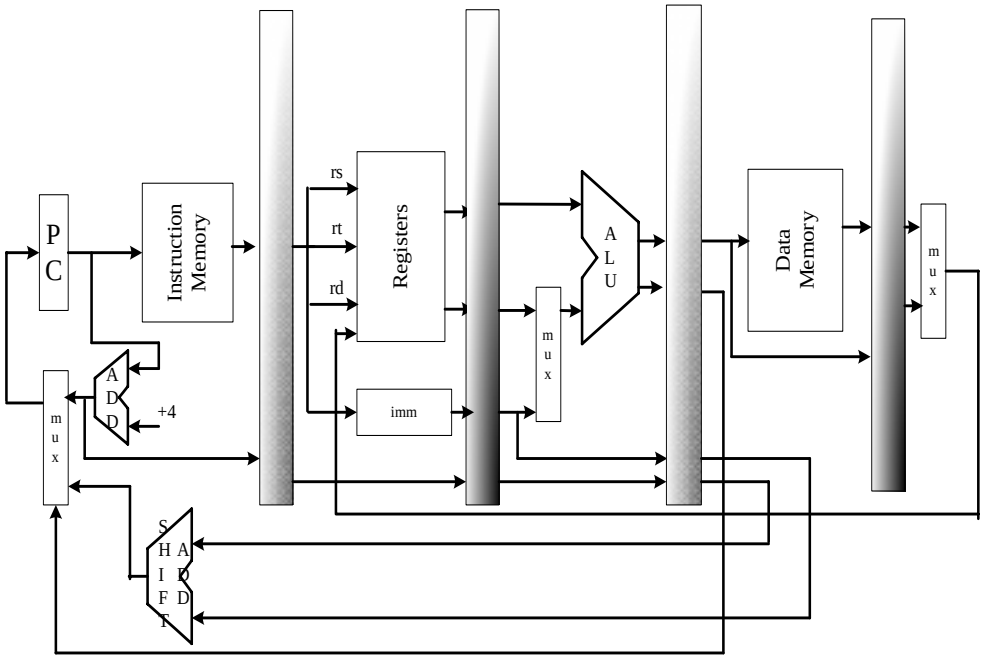


图 4.1 简化的流水线数据通路图

图 4.1 中，垂直放置的浅灰色长条即为流水寄存器，在每个时钟周期中，数据会从上一个流水线寄存器流入到下一个流水线寄存器或处理部件，流水线寄存器按照指令执行的不同阶段进行命名，如取指与译码阶段之间的流水寄存器就称为 IFID 流水寄存器。

每条指令的五个执行阶段就被严格的划分出来，通过流水线寄存器隔开，使得原本直接相连的部分变得相互独立。因为该处理器可以分为五个执行步骤，所以中间需要加入 4 个流水寄存器：

1．将取指与译码分开的 IFID 流水寄存器。

2．将译码与执行分开的 IDEX 流水寄存器。

3．将执行与访存分开的 EXMEM 流水寄存器。

4．将执行与回写分开的 MEMWB 流水寄存器。

在流水线数据通路中，一条指令从读取到执行都需要五个时钟周期，单个指令的执行过程并没有减少，通过流水线寄存器隔开，指令不占用整条数据通路，只占用数据通路的一级。五条指令可以分别占用流水线的一级，这样就大大提高了数据通路的利用率。指令在数据通路中重叠执行，从整个流水线来看，我们可以发现每一个时钟周期都会有一条指令执行完毕，由于单位时间内每一级的执行次数比单周期多，因此，流水线整个数据通路的执行速度比单周期高很多，可以有效提高处理器的效率。

流水线都在时钟的上升沿来临的时候把上一级处理器的数据发送给下一级，这样不同的指令的执行延迟被隔离开来。

为了使结构清晰明了，并将每个信号进一步明确，使用 IF 表示取指部件，ID 表示译码部件，EX 表示执行部件，MEM 表示存储部件，WB 表示回写部件，带完整数据信号的流水线数据通路顶层框图见图 4.2，其中控制信号未给出，由控制模块设计部分完成。

流水线寄存器与程序地址寄存器一样，由 D 锁存器组成，主要考虑的问题是需要传递哪些控制信号，将在下一节中进行详细讨论。对各个执行模块的组成分析如下：

IF：取指令模块，完成指令的读取，并在下一个时钟周期读取新的程序指令，整个模块主要由一些多路器、地址加四加法器、程序 PC 和指令存储器组成。

ID：译码模块，完成指令译码和读取操作数，主要由译码单元、寄存器堆、移位加法器、符号扩展单元组成。

EX：执行模块，完成指令的执行，由算术逻辑单元、ALU 控制器及相关的控制模块组成。

MEM：存储模块，实现程序数据的存储，由数据存储器组成，并不是每条指令都经历此阶段。

WB：回写模块，比较简单，只是选择要回写的操作数，使用多路器就可以完成其功能。

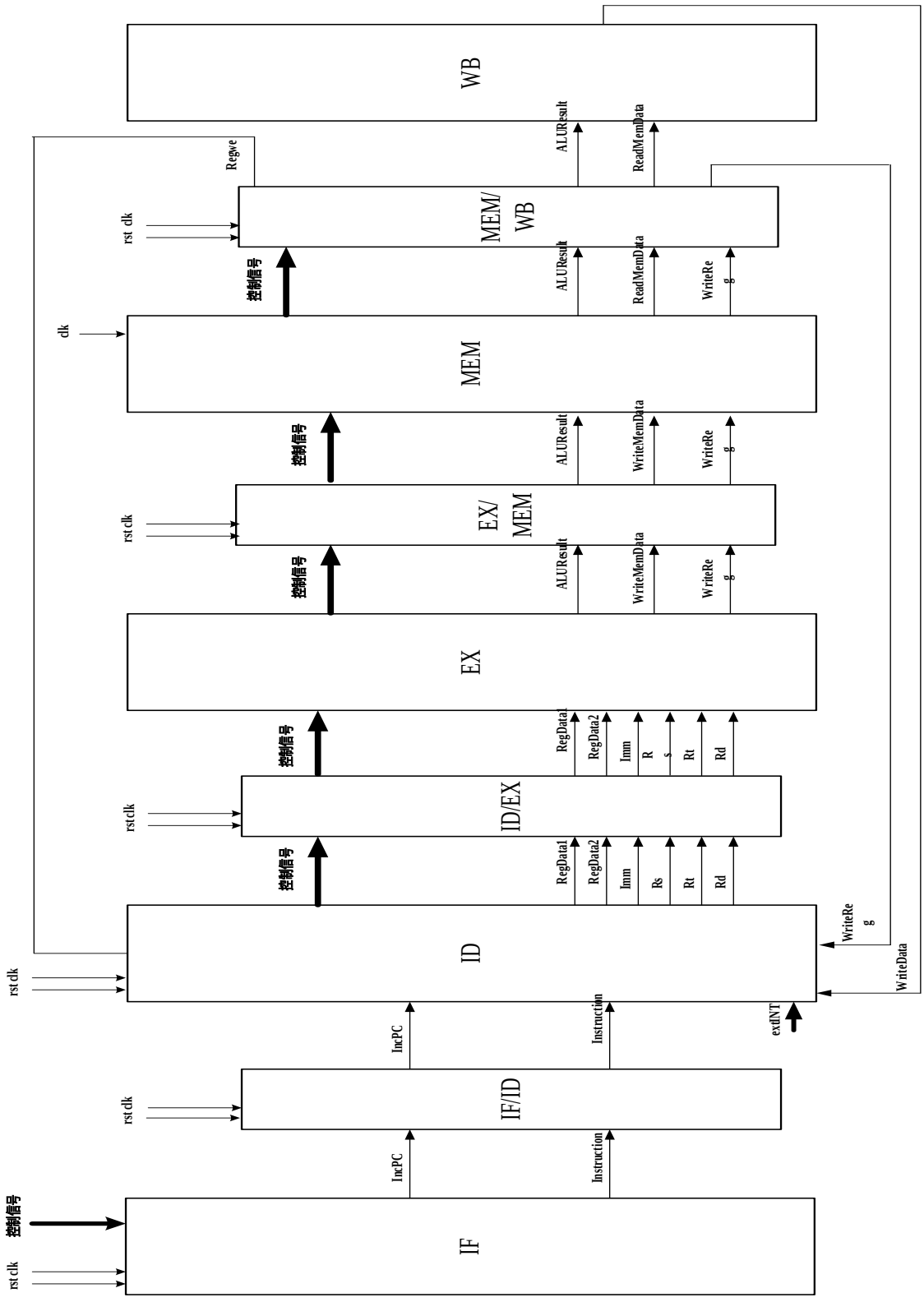


图 4.2　无控制信号流水线数据通路顶层图

4.2 流水线寄存器设计

根据前面的分析和设计的顶层框图，就可以对顶层框图进一步详细分析和设计，本节将按照数据的流向，对流水线寄存器进行详细设计，考虑的主要问题是流水线需要传递的数据，以及数据在哪一个阶段被使用。

4.2.1 IFID 流水寄存器

要设计 IFID 寄存器，就必须对 IF 模块的数据信号进行分析。IF 主要完成指令的预取，要完成指令的预取，需要一个程序地址寄存器单元，一个存储指令的指令存储器和一个用于计算新地址的加法器，取指令模块数据通路局部图如图 4.3 所示。

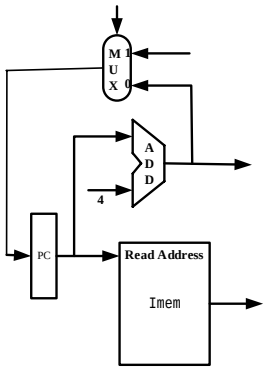


图 4.3 IF 模块数据通路局部图

因此与 IF 模块相连的 IFID 寄存器需要对增加的 PC 和取出的指令进行暂存，流水线按照输入和输出成对出现，一个为输入信号，一个为输出信号，因为控制部分和冒险检测和转发单元由设计的其他人员完成，不考虑控制信号，IFID 流水寄存器实体描述如图 4.4，所有流水线寄存器都包含时钟信号和复位信号，在实体描述中省略。

```
entity IFID is
    port(
    pIncPC_in           : in std_logic_vector(31 downto 0);--输入的地址信号
    pIncPC_out          : out std_logic_vector(31 downto 0);--输出地址信号
    pInstruction_in     : in std_logic_vector(31 downto 0); --输入指令信号
    pInstruction_out    : out std_logic_vector(31 downto 0));--输出指令信号
end IFID
```

图 4.4 IFID 流水寄存器信号实体描述

该流水线寄存器主要是将 PC+4 的值进行暂存，数据通路处理时，并不知道下一

条指令是什么类型，所以必须为下一个周期可能出现的分支或跳转指令做准备，作为 ID 阶段进行转发的新的程序地址。控制信号 pIF_Flush 用于将流水线寄存器清空，当其为 1 时，需要将增加后的地址，和取出的指令清空，而当发生冒险时，输出数据保持不变。IFID 流水线寄存器设计流程图如图 4.5 所示。

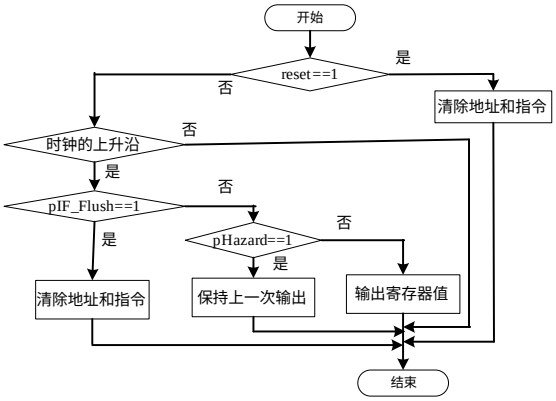


图 4.5 IFID 流水寄存器设计流程图

当 reset 信号有效，清除 IFID 的地址和指令，否则在时钟的上升沿进行判断，当 pIF_Flush 有效时，清空流水线寄存器的 PC 增加值和存储的程序指令，如果没有冒险发生，将 IFID 寄存器保存的 PC 增加值和程序指令输出，否则，保持输出不变。

4.2.2 IDEX 流水寄存器

对于 ID 模块来说，指令的译码和操作数的预取在此级产生，其中指令译码控制器由控制模块设计完成。ID 模块由寄存器堆，移位加法器，符号扩展单元组成，寄存器堆用于取运算的操作数，符号扩展单元对立即数或偏移量进行位扩展，移位加法器用于计算分支跳转的地址，取指令的模块图如图 4.6 所示。

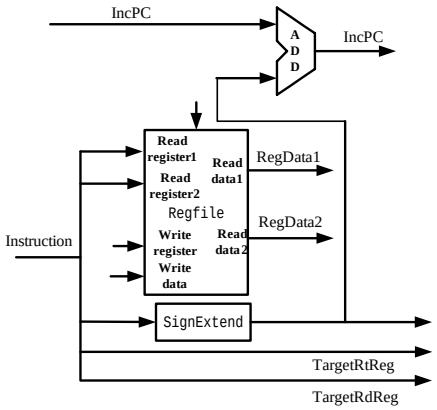


图 4.6 ID 模块部分数据通路局部图

IDEX 流水线寄存器位于译码模块与执行模块之间，传递经过位扩展的立即数，

寄存器堆读出的寄存器数据，以及将上一阶段的增加后的 PC 值向下传递等，其传递信号如表 4.1 所示，未列控制信号与时钟和复位信号。

表 4.1 IDEX 流水线寄存器信号分析

输入信号	输出信号	位宽	信号类型说明
pRegData1_in	pRegData1_out	32	指令 rs 字段读出的寄存器数据 1
pRegData2_in	pRegData2_out	32	指令 rt 字段读出的寄存器数据 2
pImm_in	pImm_out	32	将指令的低 16 位进行 32 位的符号扩展
pTargetRtReg_in	pTargetRtReg_out	5	用于数据转发和回写阶段选择寄存器号
pTargetRdReg_in	pTargetRdReg_out	5	用于数据转发和回写阶段选择寄存器号
pRsReg_in	pRsReg_out	5	Rs 寄存器，用于数据转发

由于不知道要执行的指令类型，回写阶段的目的寄存器可能为指令的 rt 字段，也可能为指令的 rd 字段，因为流水线长度一般来说是固定的，在暂存数据时，应考虑所有的可能，所以 IDEX 流水线寄存器将 rt 和 rd 都进行了暂存，在后面的阶段到底选择哪一个作为回写的寄存器号，由传递的控制信号进行多路选择。IDEX 同时要传递 rs，用于数据的转发。两个寄存器端口读出的操作数，有些操作数可能在后面的部分不会被使用，但为了保证设计的完整性与兼容型，设计时必须将寄存器的两个值都进行传递，除此之外，还有扩展后的立即数，及译码控制信号。

当复位信号有效时,将流水线寄存器 IDEX 清空,否则在时钟的上升沿将所有 IDEX 寄存器的数据输出至下一级。

4.2.3 EXMEM 流水寄存器

对于 EX 模块来说，除了完成上一级传递的相关控制信号之外，主要完成算术运算，由 ALU 和一些多路器组成，EX 模块局部数据通路图如图 4.7 所示。

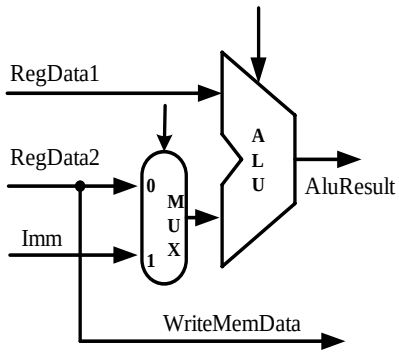


图 4.7 EX 模块分数据通路局部图

EXMEM 流水线寄存器位于执行部件与存储访问部件之间，按照控制信号，对 ID 阶段取到的操作数进行运算，并将相关的数据和控制信号传递到下一级，其实体描述如图 4.8 所示，时钟、复位及控制信号未列出。

```
entity EXMEM is
    port(
    pAluResult_in       : in std_logic_vector(31 downto 0); --输入 ALU 计算结果
    pAluResult_out      : out std_logic_vector(31 downto 0);--输出 ALU 计算结果
    pWriteMemData_in    : in std_logic_vector(31 downto 0); --输入写存储器数据
    pWriteMemData_out   : out std_logic_vector(31 downto 0);--输出的写存储器数据
    pWriteReg_in        : in std_logic_vector(4 downto 0);  --输入的写寄存器号
    pWriteReg_out       : out std_logic_vector(4 downto 0)  );--输出的写寄存器号
end EXMEM;
```

图 4.8　EXMEM 流水寄存器信号实体描述

对于不同的指令，其 MEM 不尽相同，为了将操作简化，所有指令都必须经过 MEM 阶段，保证指令的执行按照一级一级的往下进行，虽然部分指令在当前级不做任何操作，但也必须将数据存入流水线寄存器，等回写阶段做完再执行该条指令后的下一条指令，所以需要传递的有 ALU 的计算结果，作为回写的数据或作为 MEM 阶段访存的地址，因为需要回写结果，所以必须将回写的寄存器号暂存。

其实现的功能比较简单，当复位信号有效时，将流水线 EXMEM 寄存器清空，否则在时钟的上升沿将所有 EXMEM 寄存器的数据输出。

4.2.4　MEMWB 流水寄存器

MEM 阶段完成数据的存储和读取，主要由数据存储器组成，同时要传递部分回写阶段的控制信号。

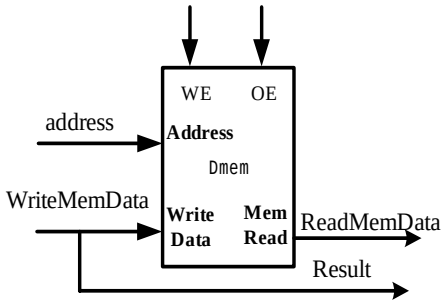


图 4.9　MEM 模块局部数据通路图

MEMWB 流水寄存器处于 MEM 模块与 WB 模块之间，主要对回写的寄存器号，回写

的两个数据进行暂存，其信号实体描述如图 4.10 所示，控制信号，时钟、复位信号未列出。

```
entity MEMWB is
    port(
    pReadMemData_in : in std_logic_vector(31 downto 0); --数据存储器读出数据输入
    pReadMemData_out: out std_logic_vector(31 downto 0);--数据存储器读出数据输出
    pAluResult_in   : in std_logic_vector(31 downto 0); --alu 数据输入
    pAluResult_out  : out std_logic_vector(31 downto 0);--alu 数据输出
    pWriteReg_in    : in std_logic_vector(4 downto 0);  --要写入的寄存器号输入
    pWriteReg_out   : out std_logic_vector(4 downto 0); --要写入的寄存器号输出
end MEMWB;
```

图 4.10　MEMWB 流水寄存器信号实体描述

回写阶段的流水寄存器需要指明流水寄存器号，送出两个回写数据，即 ALU 的运算结果数据，数据存储器的读出数据，见图 4.10 的实体描述。

当复位信号有效,将流水线寄存器 MEMWB 清空,否则在时钟的上升沿将所有 MEMWB 寄存器的数据输出。

每一个周期都会有最多五条指令在流水线中执行，为了方便对流水线的理解，设计画出了完整的流水线数据通路设计图见附录 2,设计和完成了图中白色模块及实线部分，控制器和数据转发和冒险模块等灰色模块由课题组由控制设计模块完成。

4.3　流水线数据通路简析

如何在单周期的基础上提高其执行效率，思路就是通过流水线提高处理器的效率，程序执行的效率，即单位时间执行的程序数，可用式(4.1)表示。

$$\frac{\text{sec}onds}{program}=\frac{instructions}{program}\times\frac{cycle}{instruction}\times\frac{\text{sec}onds}{cycle} \tag{4.1}$$

对于一个特定的程序，可以通过下面的方法提高执行的效率。

1．通过编译器的优化可以减少单个程序的指令数，即减少第一个乘数。

2．第二项表示每个周期的指令数，即 CPI，理想 CPI 为 1。

3．对于第三项，可以发现，其值越大，执行的速度越高。

对于单周期数据通路而言，大多数时间，ALU 要么等待数据的输入，要么保持数据的输出，所以最理想的状态就是让处理器的所有部件一直都在工作。使用流水线，

5 嵌入式微处理数据通路验证

完成了嵌入式微处理器各个模块的设计后，就需要从底层开始对各个设计的模块进行仿真验证，最后对整体进行仿真验证。本章以实现的 34 条指令为基础，在流水线数据通路中，对各个模块的进行仿真验证和分析，并以此为依据验证整体正确性。

5.1 算术逻辑单元仿真验证

算术逻辑单元需要完成的操作主要分为算术运算，逻辑运算，移位运算，以及比较运算，为了对算术逻辑单元进行验证，需要建立相应的测试指令[40]，并对算术逻辑单元进行仿真分析，指令序列如表 5.1 所示。

表 5.1 算术逻辑单元测试指令序列

输入指令	期望的 ALU 输出	结果分析
addi $2, $0, 10	0x0000000a	ALU 执行加操作，输出为 a
addi $3, $0, 5	0x00000005	ALU 执行加操作，输出为 5
sub $4, $2, $3	0x00000005	ALU 执行减操作，输出为 5
add $5, $2, $3	0x0000000f	ALU 执行加操作，输出为 f
and $6, $2, $3	0x00000000	ALU 执行与操作，输出为 0
or $7, $2, $3	0x0000000f	ALU 执行或操作，输出为 f
xor $8, $2, $3	0x0000000f	ALU 执行异或操作，输出为 f
nor $9, $2, $3	0xfffffff0	ALU 执行或非操作，输出为 0xfffffff0
sll $10, $2,28	0xa0000000	ALU 执行左移操作
srl $11, $2,1	0x00000005	a 右移 1 位，结果为 5
sra $12,$10,1	0xd0000000	0xa0000000 算术右移一位为 0xd0000000

ALU 是处理器的算术逻辑运算部件，如图 5.1~5.3 所示，本测试指令序列主要完成了加运算、减运算、与运算、或运算、异或运算，或非运算，逻辑左移，逻辑右移，算术右移的测试。对于 ALU 操作，该验证代码重点验证算术运算，逻辑运算和移位运算的正确性。可以使用 ModelSim[41]或 ISE 自带的 simulator[42]进行仿真。

通过上述指令序列，可以测试 ALU 的功能实现，仿真结果如图 5.1~5.3 所示。ALU 仿真波形(一)为从第三个时钟周期的开始的波形图，因为按照五级流水线，指令从第三个周期开始进入执行阶段，使用 ALU 进行运算，包括的四条指令分别对应指令

addi $2，$0，10；addi $3，$0，5；sub $4，$2，$3；add $5，$2，$3；指令 1 为立即数指令，即将 0 号寄存器的值加上 10 送 2 号寄存器，0 号寄存器默认为 0，所以运算后 2 号寄存器值为 10，同理，指令 2 的操作也是为给 5 号寄存器赋初值 5，指令 1 与指令 2 执行的操作为加法操作，对应的操作码为 01011，从图中可以看出，前两个周期操作码都为 01011，两数执行加法运算的得到了预期的仿真结果。

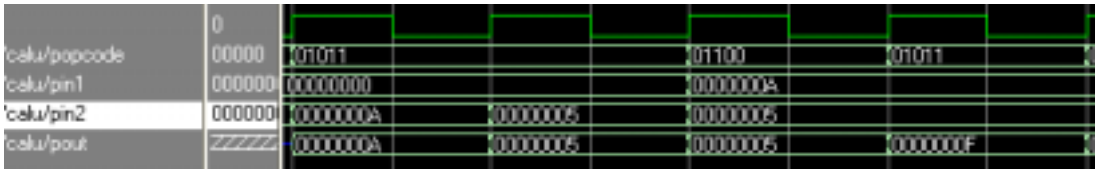


图 5.1 ALU 仿真波形(一)

对于指令 3，其操作为让 2 号寄存器的值减 3 号寄存器的值，减法运算对应的操作码为 01100，减法运算的结果应为 0x0000000a 减 0x00000005 为 0x00000005，指令 3 为 R 类指令 add，对应的操作码为 01011，将 2 号寄存器的值与 3 号寄存器值相加，结果应该为 0x0000000f，如图 5.1 所示 ALU 得到了预期的仿真结果。

指令 and $6，$2，$3；or $7，$2，$3；xor $8，$2，$3；nor $9，$2，$3 对应的仿真图如图 5.2 所示。

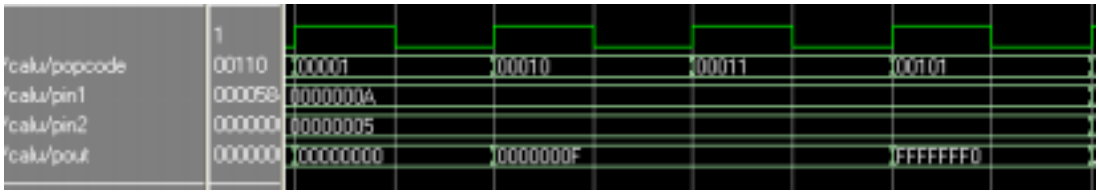


图 5.2 ALU 仿真波形(二)

对应的 ALU 操作分别为与，或，异或，或非，对应的操作码为 00001，00010，00011，00101，10 对应的二进制为 1010(高 28 位略)，5 对应的二进制为 0101(高 28 位略),进行与运算的结果应该 0000(高位略),或运算和异或的结果为 1111(高位略)，或非的结果为 0xfffffff0，通过仿真检验，ALU 为预期的仿真结果。指令 sll $10，$2，28，srl $11， $2，1，sra $12，$10，1 对应的仿真图如图 5.3 所示。

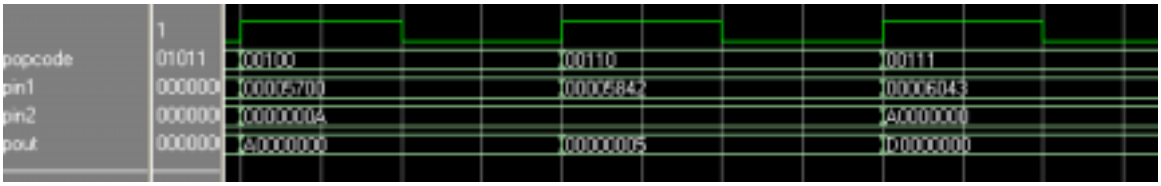


图 5.3 ALU 仿真波形(三)

对应的 ALU 操作分别为左移，右移，算术右移，对应的操作码为 00100，00110，00111，对于 sll 指令，必须取其指令的 sa 字段的值，将其指令的低 16 位进行扩展，

参与 ALU 运算，sll 指令的低 16 位为 0x5700，符号扩展后的结果为即为 pin1 的输入 0x00005700，同理，srl 的低 16 位作为符号扩展后的为 0x00005842，srl 的低 16 位经过符号扩展后的 0x00006043，ALU 执行运算时取其第 10 到 6 位参与运算，通过仿真验证，结果与预期结果相同，0x0000000a 逻辑左移 28 位结果位 0xa0000000，逻辑右移 1 位的结果位 0x00000005，而对于 0xa0000000 进行算术右移的结果为高位补符号位右移，其结果位 1010 算术右移的结果 1101，为 d，如图 5.3 所示。

5.2 寄存器堆仿真验证

寄存器堆作为数据通路的重要部件，几乎所有的指令都要对其进行访问，每条指令至少要读或写一次寄存器堆。读寄存器堆，是为了得到运算操作数，写寄存器是将运算的结果或读取的数据回写到寄存器堆中。为了对寄存器堆进行测试，本验证设计了三条指令，并对寄存器堆的读写信号进行分析，设置的测试指令如表 5.2。

表 5.2 寄存器堆测试指令序列

输入指令	寄存器读操作	寄存器写操作
addi $2, $0, 10	T1:readreg1:=0 readreg2:=2	T4: regwe:=1 写使能有效 writereg:=2 writedata:=10
addi $3, $0, 5	T1:readreg1:=0 readreg2:=3	T4: regwe:=1 写使能有效 writereg:=3 writedata:=5
add $4, $2, $3	T1:readreg1:=2 readreg2:=3	T4: regwe:=1 写使能有效 writereg:=4 writedata:=15

寄存器部分仿真波形如图 5.4 所示。

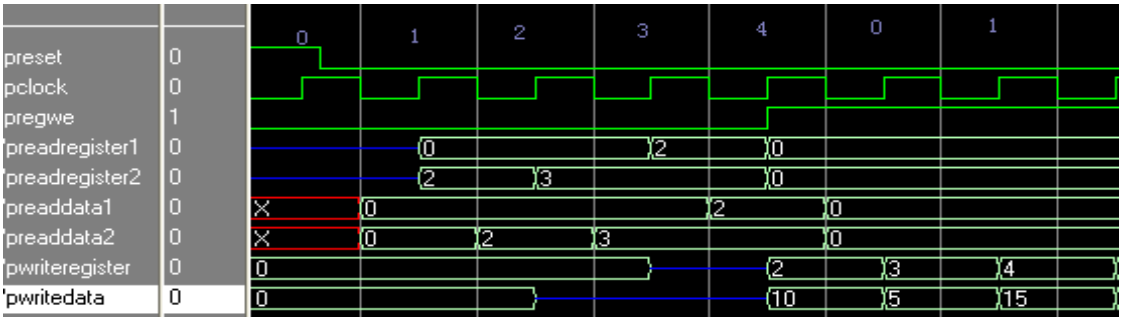


图 5.4 寄存器堆仿真图

对于流水线数据通路来说，从第 2 个时钟周期开始，即第一条指令进入 ID 阶段，第一条指令在第 2 个时钟周期读 0 号寄存器和 2 号寄存器，并在第 5 个时钟周期，即第一条指令的写回阶段，对 2 号寄存器回写入数据 10。第二条指令从第 3 个时钟周期读寄存器 3 和寄存器 0，并在第 6 个时钟周期，即第二条指令的回写阶段，对 3 号寄存器写入数据 5。第三条指令在第 4 个时钟周期读 2 号寄存器和 3 号寄存器，并在第 7 个时钟周期写入寄存器 4 运算结果 15。

5.3 指令存储器及地址寄存器仿真验证

指令存储器和地址寄存器是取指的重要部件，验证将其放一起讨论。地址寄存器是一个简单的 32 位数据锁存器，由一些简单的触发器组成，数据在时钟的上升沿输出，并将加 4 后的值存入地址寄存器，指令存储器将地址寄存器的输出值作为其地址进行寻址，读取相应的指令，其仿真图如图 5.5。

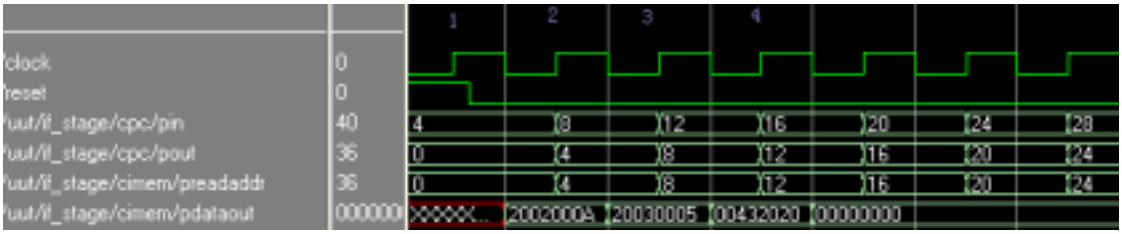


图 5.5 指令存储器仿真波形

如上图所示，指令地址寄存器的在每个时钟周期的上升沿将暂存的地址 pin 值在输出端口 pout 输出，作为指令存储器的地址输入，将其值加 4 从 pin 输入地址寄存器暂存。从仿真图可以看出，指令的 pin 总保持 pout+4 的值，并在每个时钟周期自动+4，符合设计的要求，指令存储器初始化指令有 addi $2, $0, 10，addi $3, $0, 5，add $4, $2, $3，nop，……对应指令编码为 0x200200a，0x2003005，0x00432020，0x00000000……，如仿真图所示，pdataout 在每个时钟的下降沿从指令存储器按照 preadaddr 读出，仿真结果满足设计的需要。因为可综合的指令存储器无法进行仿真，因此需要设计专门的仿真指令存储器进行仿真，要求与 Xilinx 自带的 Block RAM 功能完全相同。

5.4 数据存储器仿真验证

数据存储器是存储程序数据的地方，通过指令 lw，sw 进行读写，如指令 sw $5,100($4)，就是将 5 号寄存器的值保存到数据存储器 M[100+reg:4]，而对于 lw

$6,100($4)则相反，将数据数据存储器 M[100+reg:4]，读出，写入 6 号寄存器，为了对数据存储器进行仿真分析，设置指令序列如表 5.3 所示，为了验证 lw 指令的正确，最后对取到的值进行了加法运算。

表 5.3 数据存储器测试指令序列

输入指令	操作	结果分析
addi $2, $0, 10	无存储器操作	给 2 号寄存器赋值 10
addi $3, $0, 20	无存储器操作	给 3 号寄存器赋值 20
sw $2, 100($3)	写数据 Cache	将寄存器 2 值 10 保存到 100+20 处
lw $4, 100($3)	读数据 Cache	将 100+20 处数据读出，存 4 号寄存器
addi $5, $4, 30	将 4 号寄存器加 30，送 5 号	将 4 号寄存器的值加 30，结果为 40

其仿真图如图 5.6 所示。

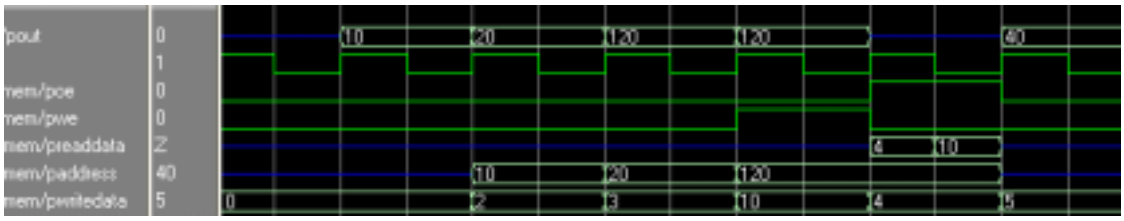


图 5.6 数据存储器仿真图

从上图可以看出在第 4 个时钟周期，ALU 计算出偏移地址 pout 为 120，保存一个周期，作为的下一个周期写数据存储器地址 paddress，在第 5 个时钟周期，写使能 pwe 有效，写入数据 10，在第 6 个时钟周期，执行指令 lw，在第 6 个时钟周期，读出数据 10，并写入寄存器 4，最后执行一个 addi 指令，对寄存器 4 加 30，得到结果 40。

FPGA 都提供了 Block RAM 供用户使用，但使用起来不够灵活，移植性差，设计仿真采用了 VHDL 描述的 RAM，以便仿真。实际综合时，使用 Coregen 的 Block RAM 生成。

5.5 符号扩展单元仿真验证

RISC 执行立即数指令时，ALU 执行运算的两个操作数，一个来自于读寄存器 rs 的值，另外一个来自指令的低 16 位的 32 位扩展，因为指令分为无符号类型和有符号类型，符号扩展单元应该能根据不同的指令分别进行符号扩展和无符号扩展。当执行无符号指令时，进行无符号的扩展，16 位立即数的高 16 位全部补 0，当执行有符号指令时，当 16 位立即数的最高位为 1 时，新的 32 位的扩展结果高位全部补 1，

否则补 0。

表 5.4 符号扩展测试指令序列

输入指令	符号单元输入	结果分析
addi $2,$0,-1	0xffff	符号扩展后输出为 0xffffffff
addiu $3,$0,-1	0xffff	无符号扩展后的输出为 0x0000ffff
addiu $3,$0,10	0x000a	无符号扩展后的输出为 0x0000000a
nop	0x0000	0x00000000

符号扩展单元仿真图如图 5.7 所示。

图 5.7 符号扩展单元仿真图

当到第 2 个时钟周期，即到达 addi 指令的 ID 阶段时，进行符号扩展，addi 为有符号加，此时 pin 输入为-1，即为 0xffff，pext 为 0，即进行有符号扩展，结果应该为 0xffffffff。在第 3 个时钟周期，进入 addiu 的 ID 阶段，即为无符号加，此时的 pin 仍为为-1，pext 为 1，进行无符号扩展，结果为 0x0000ffff。在第 4 个时钟周期，进入 addiu 的 ID 阶段，此时 pin 输入为 10，即 0x000a，pext 为 1，进行无符号扩展，扩展后结果输出 pout 为 0x0000000a，而对于后面的 nop 指令，立即数字段全为 0，执行符号扩展后的结果输出仍为 0。

5.6 小结

本章对流水线的整个数据通路部件进行了验证，为了提高验证的效率，并能从整体上把握设计的正确性，仿真分析都是基于指令序列的流水线数据通路部件的验证，按照不同的指令序列，观察各数据通路部件的运行情况，分析指令序列在各个部件中的运行，进行对比分析，并能更进一步判断整个数据通路在执行特定测试程序时是否正确。经过前面的分析和验证，可判断流水线处理器数据通路是否设计正确。

6 总结与展望

6.1 全文总结

论文重点对嵌入式微处理器的数据通路进行分析，对算术逻辑单元 ALU，寄存器堆，指令存储器、数据存储器进行了设计，并对带 Cache 的存储系统进行了简单分析，在完成各个部件的基础上设计了单周期数据通路。在对单周期数据通路分析的基础上，通过加入流水线寄存器，实现了五级流水嵌入式微处理器数据通路，最终设计的嵌入式微处理器解决了流水线数据冒险和控制冒险问题，由控制器模块设计完成。最终实现了五级流水，基于 MIPS 指令集的软核处理器，支持 34 条 MIPS32 指令，并全部在数据通路中验证通过。

设计基于 MIPS 指令集，通过最简洁的设计来取得最快的速度，因为其结构清晰简单，可以很容易的利用流水线来提高处理器的执行效率。

设计主要完成的工作有：

1．设计了单周期不流水数据通路。

2．选取了需要实现的指令集，并根据要完成的指令，对数据通路进行设计，以支持预先设定的指令集。

3．对各个数据通路部件分别进行设计和实现，使其支持待实现的指令集。

4．设计了流水线数据通路，并在对其仔细分析的基础上，设计了流水线寄存器。

5．在控制器设计部分实现数据转发和冒险检测的基础上，基于整个系统进行了功能模块仿真。

6．将设计的单周期嵌入式处理器和流水线处理器下载到 FPGA 开发板上，运行预先设计的二进制程序代码，得到了预想的结果。

6.2 研究展望

设计实现了多片内操作系统智能卡中嵌入式五级流水处理器的开发，对整个嵌入式五级流水处理器的设计过程和设计方法进行了详细的分析，限于时间和个人的能力，论文尚有很多需要完善和进一步研究的地方，主要包括以及下一些方面：

1．实现指令 cache 和数据 cache 的功能，对缓存的替换算法进行分析和比较，并在此基础上实现字节和半字指令，如 lh，sh，lb，sb 指令等。

2．在此基础上，对多发射处理器数据通路进行分析和设计。

3．设计多基于行为级的描述，有待对各个部件进一步研究，在 RTL 级实现更高效的数据通路部件，并建立嵌入式系统的基准性能测试平台。

4．设计的处理器的设计的外部接口不够充分，整个处理器与串口模块，加密模块的集成和协同工作有待更进一步的研究。

致 谢

在整篇论文结束前，我要特别感谢我的导师曹计昌教授！两年来，曹老师在学习和科研中对我严格要求，给予我大量的指导和帮助，在生活上，对我的关怀无微不至。曹老师高尚的品德，严谨的治学态度，敏锐的眼光和执着的科研精神，令我深感钦佩，他对事业的崇高追求和永远饱满的工作热情，令我折服，必将是我终生学习的榜样。在此，我要再一次向曹老师献上我诚挚的敬意和衷心的感谢！

感谢舒林博士，他在工作、学习方面给予了我很多无私的帮助，并给我树立了极好的榜样。感谢同届的邹志斌、刘畅、段天，在我学习和生活上的给予了我很多的关怀，和他们交流，让我学到了很多。也感谢我的 07 级师弟程新国、张凡、刘涛、胡剑方，08 级申风有、陈斌，和他们相处很快乐，也学到了很多东西。

在这里，我还要感谢我的父母，亲戚和朋友，他们一直都默默给予我支持和关爱，祝愿他们幸福安康！

最后，衷心感谢在百忙之中抽出时间审阅本论文的专家教授！

参考文献

[1] Kitajima, A., Sasaki, T. Design of application specific CISC using PEAS-III. in: 13th IEEE International Workshop on Rapid System Prototyping 2002, Osaka: IEEE Computer Society, 2002. 12~18

[2] Stallings, W. Reduced instruction set computer architecture. in: Proceedings of the IEEE . New York : IEEE Computer Society,1988. 38~55

[3] Q. Zhang, G. Theodoropoulos. Modeling SAMIPS: A Synthesisable Asynchronous MIPS Processor. in: ACM Special Interest Group on Simulation and Modeling. Washington, DC: IEEE Computer Society, 2004. 205~212

[4] Sivarama P.D. Fundamentals of Computer Organization and design. New York:Springer-Verlag, 2003

[5] M. A. S.D. Poz, J. E. A. Cobo. A Simple RISC Microprocessor Core Designed for Digital Set-Top-Box Applications. in: IEEE International Conference on Application-Specific Systems, Architectures, and Processors. Sao Paulo: IEEE Computer Society, 2000, 35~44

[6] Patterson, D.A, Seccombe, S. Complex versus reduced instruction set computers. in: Solid-State Circuits Conference. Digest of Technical Papers. Washington, DC: IEEE Computer Society, 1983. 218~219

[7] Hennessy, J.L. VLSI Processor Architecture. IEEE Transactions on Computers, 1984,C-33(4):1221~1246

[8] Mateosian, R. The PowerPC Architecture-A Specification of a New Family of RISC Processors. Micro, 1994,14(5):2

[9] DeLano, E., Walker, W. A high speed superscalar PA-RISC processor. in: Thirty-Seventh IEEE Computer Society International Conference, Digest of Papers. San Francisco: IEEE Computer Society, 1992. 116~121

[10] Brown, E.W., Agrawal, A. Implementing Sparc in ECL. Micro, 1990,10(1):10~22

[11] Patterson D.A, Hennessy J.L. Computer Organization and Design: The Hardware /Software Interface. Third Edition. New York:Morgan Kaufmann Publishers, 2007

[12] Hennessy J.L, Patterson D.A. Computer Architecture:A Quantitative Approach. Fourth Edition. New York:Morgan Kaufmann Publishers, 2007

[13] Xiao Li, Longwei Ji. VLSI implementation of a high-performance 32-bit RISC microprocessor. in: Circuits and Systems and West Sino Expositions, IEEE 2002 International Conference. Chengdu: IEEE Computer Society, 2002. 1458~1461

[14] Xizhi Li, Tiecai Li. ECOMIPS: An Economic MIPS CPU design on FPGA. in: System-on-Chip for Real-Time Applications, 4th IEEE International Workshop on. New Zealand: IEEE Computer Society, 2004. 291~294

[15] C.J.Chang, C.W.Huang. FPGA platform for CPU design and applications. in: 5th IEEE Conference on Nanotechnology. Nagoya: IEEE Nanotechnology Council, 2005(1). 187~190

[16] Manjikian,Naraig, Roth,Jonathan. Performance enhancement and high-level specification of a pipelined processor in programmable logic. in: IEEE Northeast Workshop on Circuits and Systems, 2007. Montreal: IEEE Computer Society, 2007. 1340~1343

[17] Ramdas T., Li-minn Ang. FPGA implementation of an integer MIPS processor in Handel-C and its application to human face detection. in: IEEE TENCON 2004,2004. Chang Mai:IEEE Computer Society, 2004(1). 36~39

[18] 李瑛，高德远．16 位定、浮点微处理器的设计．计算机工程与应用，2004(4):10~12,28

[19] 冯海涛，王永纲．基于 FPGA 的 32 位整数微处理器的设计与实现．小型微型计算机系统, 2005, 26(6):1113~1117

[20] 庄伟，樊晓桠．嵌入式微处理器的系统验证平台设计．计算机应用研究，2007, 24(10):240~242

[21] 陆洪毅，沈立．一种高性能的嵌入式微处理器-银河 TS-1．电子学报，2002, 30(10):1668~1671

[22] 夏军，袁丽霞．参数化软 IP 核的速度、面积和功耗估算方法．电子学报, 2004, 32(5):47~49

[23] Wilton, S.J.E., Kafafi, N. Design considerations for soft embedded programmable logic cores. IEEE Journal of Solid-State Circuits, 2005,40(2):485~497

[24] Biswas, P., Banerjee, S. Performance and energy benefits of instruction set extensions in an FPGA soft core. in: 5th International Conference on Embedded Systems and Design. Washington, DC:IEEE Computer Society, 2006(1). 6

[25] Reaz, M.B.I., Islam, M.S. A single clock cycle MIPS RISC processor design using VHDL. in: IEEE International Conference on Semiconductor Electronics. Penang: IEEE Computer Society, 2002. 199~203

[26] 王诚. FPGA/CPLD 设计工具 Xilinx ISE 使用详解. 北京:人民邮电出版社, 2004

[27] Gschwind, M., Salapura, V. FPGA prototyping of a RISC processor core for embedded applications. IEEE Transactions on Very Large Scale Integration (VLSI) Systems,2001,9(2):241~250

[28] 李建成，庄钊文．SOC 设计的软硬件协同验证研究．半导体技术，2007, 23(10):904~908

[29] 张艳，胡桂. SOC 设计中的核心技术．微计算机信息, 2007, 23(10-2):110~112

[30] 冉峰，李润光. IP 复用的 FSPLC 微处理器 SOC 设计．微电子学与计算机, 2007, 24(11):111~113

[31] 张楷，汤志忠．VHDL 语言设计可综合的微处理器内核．计算机应用研究, 2004(6):123~124,173

[32] 齐京礼，宋毅芳．VHDL 语言在 FPGA 中的应用．微计算机信息，2006, 22(12-2):149~151

[33] MIPS32 Architecture For Programmers Volume II: The MIPS32 Instruction Set. MIPS Inc., 2005

[34] Dominic Sweetman. MIPS 处理器设计透视．赵俊良，张福新等译．北京:北京航天航空大学出版社, 2005

[35] 贾琳，樊晓桠．32 位 RISC 微处理器流水线设计．计算机工程与应用, 2005(14):115~117

[36] 杨洸，齐家月. 32 位 RISC 微处理器流水线设计．微电子学, 2001, 31(1):58~61

[37] 胡敏杰，邬齐荣．微控制器的流水线设计及时序优化．四川大学学报(自然科学版), 2007, 44(3):603~607

[38] 蔡启先，李日初．DLX 处理器整数流水线性能的研究．计算机应用, 2005(25):374~376,373

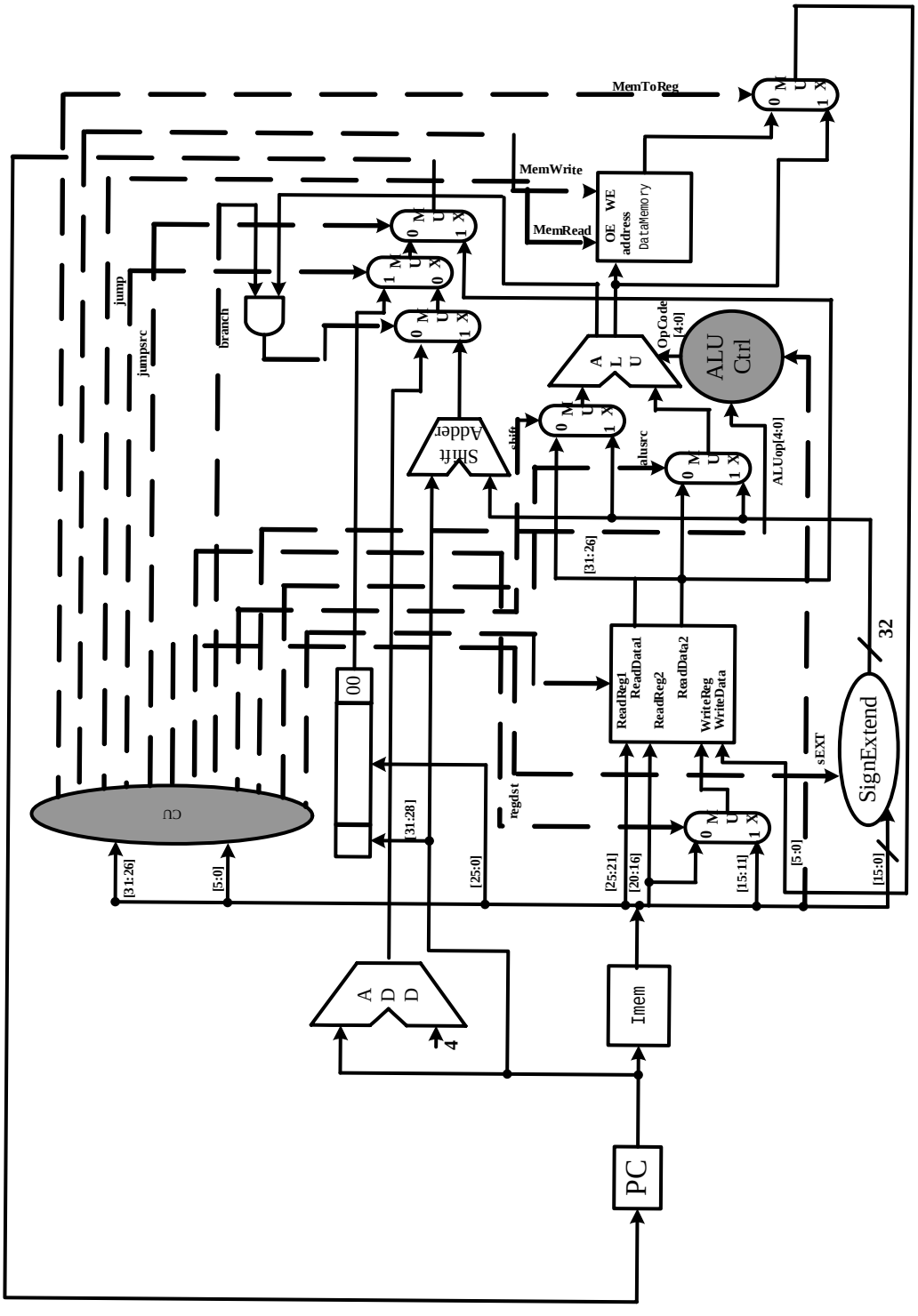
[39] Block Memory Generator v2.6. Xilinx Inc., 2007

[40] 王江，刘佩林．基于 MIPS 内核的 SoC 软硬件协同仿真．计算机工程, 2006(16): 247~249

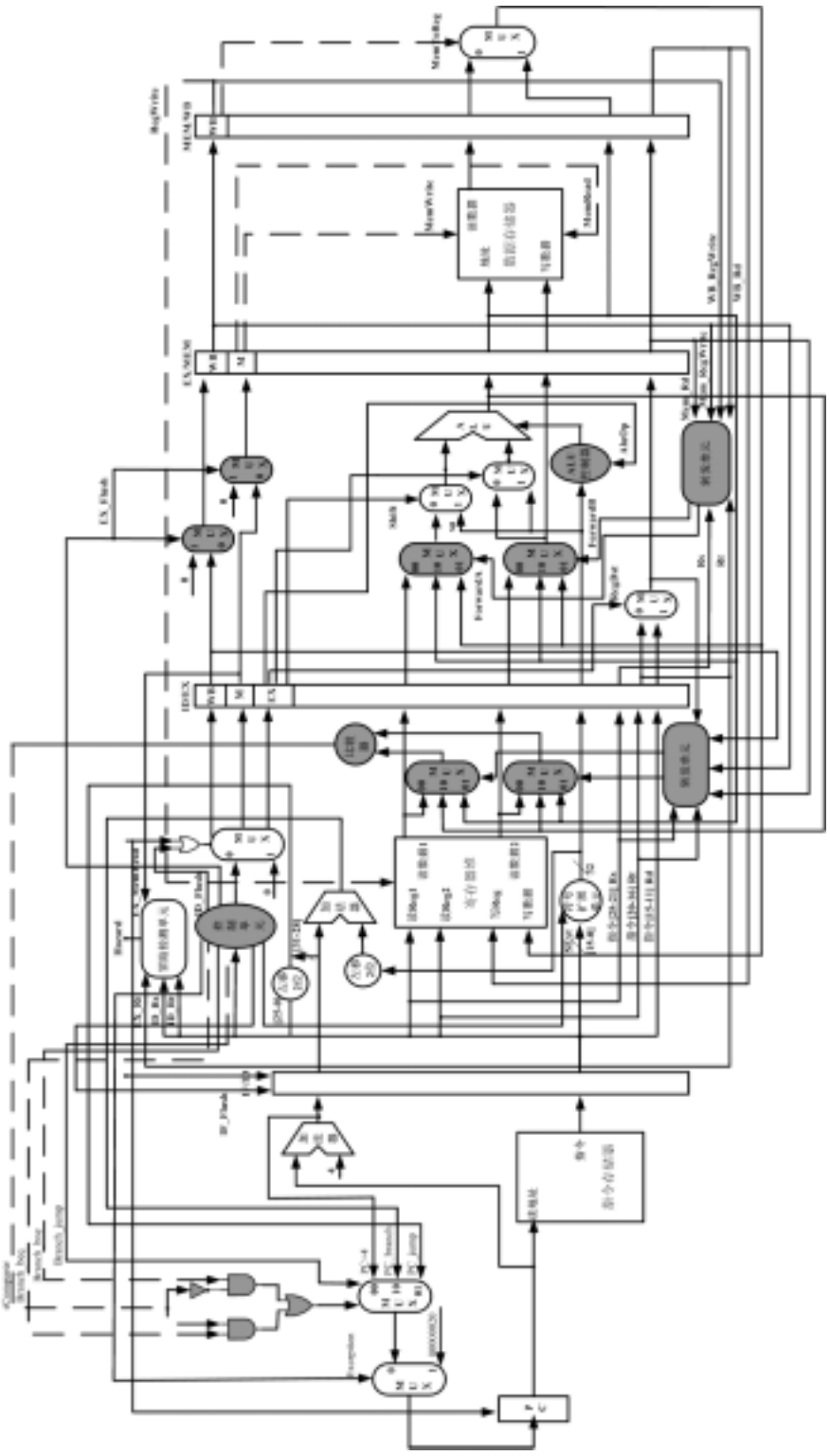
[41] 夏新恩，洪远泉．基于单片机和 EDA 技术的逻辑分析仪设计．计算机工程, 2005, 31(16):207~210,221

[42] 薛小刚，葛毅敏. Xilinx ise 9.x 设计指南．北京:人民邮电出版社, 2007

附录 1 （单周期数据通路图）



附录 2 （流水线数据通路图）



基于MIPS指令集的RISC微处理器数据通路的设计与实现

作者：　　刘宁
学位授予单位：　　华中科技大学

相似文献(7条)

1.期刊论文 江艳.廉殿斌.李勇.JIANG Yan.LIAN Dian-bin.LI Yong 64位RISC微处理器的结构设计 -微电子学与计算机2005,22(4)

文章介绍了一种64位RISC微处理器的结构设计.采用MIPS指令集,详细分析该处理器的各主要功能单元,五级流水线控制,并对该设计中潜在流水线冒险问题提供完整解决方案,最后通过在线仿真调试及配置FPGA验证了设计的正确性.

2.学位论文 邹志斌 基于MIPS指令集的RISC微处理器控制模块的设计与实现 2008

随着集成电路设计和工艺技术的发展，嵌入式系统因为具有高性能、低功耗、便携式的优点，已经在移动通信、机顶盒、智能卡等信息终端中得到了广泛的应用。而作为嵌入式系统核心的微处理器，其性能直接影响着整个系统的性能，目前精简指令集（Reduced Instruction Set Computer，RISC）架构作为微处理器设计策略的一种类型已越来越多地应用于微处理器的体系设计中。

微处理器设计首先要确定指令系统。采用与MIPS指令兼容的设计思想，根据微处理器要实现的功能选择MIPS核心指令中的34条作为指令系统。在32位单周期微处理器设计中，按照这些指令运行的数据通路，设计各种控制信号，采用组合逻辑实现控制单元。在32位多周期微处理器设计中，由于指令运行需要的时钟周期不一样，存在多个状态，使用有限状态机来描述控制单元。

在5级流水线的32位RISC微处理器设计中，指令执行过程被分为取指令、指令译码、指令执行、存储器访问和数据回写5个阶段。由于采用流水线技术，就出现了数据冒险和分支冒险的问题。对于数据冒险问题，通过在流水线中设计数据转发单元和冒险检测单元来解决。由分支或跳转语句引发的分支冒险问题，可以采用缩短分支延迟的方法，在指令译码阶段增加比较器和数据转发单元并修改相应的地址选择逻辑来解决。

基于FPGA的实验验证，首先根据对模块的设计，采用硬件描述语言描述实现，然后对每一个模块和整个系统进行功能仿真，最后将完整的RISC微处理器核综合并下载到FPGA开发板上进行验证。

3.学位论文 刘元锋 RISC架构微处理器扩展对称密码处理指令的研究 2006

增强嵌入式微处理器处理密码运算性能主要有三种途径：一是通过扩展指令集ISA(InstructionSetArchitecture)来加速密码运算模块如S盒模块、置换模块等。二是通过添加密码协处理器来加速密码算法执行效率。三是开发新处理器体系架构的并行性和专用功能运算模块，特别是对公钥密码体制。如IntelItaniumIA-64架构通过加速RSA算法关键运算模块来加速算法执行速度。

论文针对如何提高嵌入式RISC架构微处理器处理对称密码算法的效率展开研究，采用通过扩展指令集ISA来加速密码运算模块的设计思想。经过对对称密码算法运算特性、MIPS微处理器体系结构、关键运算模块的硬件实现理论等进行了深入研究，探讨了基于MIPS指令集架构提高嵌入式微处理器处理对称密码效率的有效思想。本文的主要贡献：

1、对现有密码处理平台进行了深入分析，研究如何通过扩展专用指令集来提高RISC微处理器处理对称密码算法的效率。

2、深入研究了RISC微处理器体系结构，设计了一款基于MIPS指令集格式的微处理器，通过该处理器的设计为所研究扩展对称密码处理指令提供验证性平台。

3、分析对称密码算法的运算特点、以及MIPS架构微处理器处理对称密码算法特点，研究了影响处理对称密码算法效率的几个密码运算单元。

4、基于FPGA方式研究了移位运算、S盒实现、模(2n+1)加、模(2n+1)乘、比特置换等运算的硬件实现原理，实现了扩展对称密码处理指令。

研究结果表明扩展指令后的微处理器执行多种对称密码算法的性能比通用微处理器性能提高1.7～10.8倍，比并行向量微处理器性能提高了0.8～2.03倍。结果说明本研究既能保证对称密码算法应用的灵活性，又能达到较高的性能。

4.期刊论文 李杰.贺占庄 基于MIPS架构的RISC微处理器RM7000A -单片机与嵌入式系统应用2004,""(2)

概要介绍基于MIPS指令集的RM7000A微处理器的大容量片内缓存、超标量流水线、指令双发射、大量寄存器组等主要特性,并对其两种应用方案进行探讨.

5.学位论文 马钢 精简指令集微处理器流水线架构模型的设计 2005

随着半导体和通信技术的迅猛发展，人们对嵌入式专用集成电路(ASIC)的要求日益提高。不仅要求芯片有很强的数据处理能力，而且还要满足实时响应、功耗小、故障率低等苛刻条件。目前中国的半导体制造业的蓬勃发展给予中国芯片设计公司采用世界上最先进嵌入式系统技术——片上系统(SOC)一个机会。在任何片上系统模型中，微处理器都是其核心部分，本论文就是提出了一种可以商用化通用微处理器数据通道设计模型。

通过对比和参照国内外先进的微处理器设计理念和商业方案，决定设计采取精简指令集计算机(RISC)微处理器的设计，因为它满足嵌入式系统对微处理器的基本要求。在指令集方面选取了兼容MIPS指令集的方法，这样不仅微处理器的结构更容易被理解，也减轻了自主编写操作系统的压力。论文设计中采用了五级流水线结构，分别为取指令、指令译码、指令执行、存储器访问和寄存器写回。论文中还设计了部分控制电路来解决由于采用指令并行结构而带来的流水线冲突问题——结构灾害、数据灾害和控制灾害。通过在芯片性能、成本和功耗间折衷考虑，提供了在工程上允许的解决方案。最后，采用半导体设计公司常用的设计文档书写格式，运用硬件描述语言VerilogHDL对整个流水线的数据通道进行详细的行为级建模。

6.期刊论文 李杰.贺占庄.白军元 基于MIPS IV指令集的RISC微处理器-RM7000A及其应用的实现 -微电子学与计算机2003,20(z1)

本文概要地介绍了基于MIPS指令集的RM7000A微处理器大容量片内缓存、超标量流水线、指令双发射、大量寄存器组等主要特性,并对其两种应用方案进行了探讨.

7.学位论文 忻凌 针对嵌入式系统的Java虚拟机的硬件实现 2004

随着集成电路技术的发展,嵌入式系统正在被越来越广泛地使用.从网络、系统控制等高性能设备到个人数字助手(PDA)和手机,都可以见到它的身影.而随着互联网应用市场的高速发展,人们对Java的使用投入了极大的兴趣.仅管Java语言最初因网络发展的需要而产生,但其指令短小、安全、平台无关的特点使得它开始在嵌入式系统中被大量使用,特别是对于那些便携式设备,Java的那些特点似乎生来就是为它们而准备的.嵌入式系统中的关键部分是微处理器,它基本决定了整个嵌入式系统的性能.我们工作的目标是设计一个针对嵌入式系统的微处理器,它执行Java语言有较好性能,并且兼容一种RISC指令集(选用ARM或者MIPS指令集).在20世纪80年代提出的RISC结构的处理器相当适合嵌入式系统的使用.由高级RISC机器公司提供的ARMR处理器和由MIPS技术公司提供的MIPSR处理器是现在世界上最著名和使用最广泛的嵌入式通用微处理器.文中我们将讨论基于嵌入式系统RISC微处理器的Java平台,它的几种实现方式的特点和异同,并提出能以低成本实现基于RISC设计的硬件直接执行Java的字节码.该文中,我们需要设计两个32位嵌入式RISC处理器分别兼容ARM7和MIPS2000.它们是一种新的32位微处理器结构和有一种附加加速JCVM的硬件.在这两个处理器中,不仅能执行本地指令集而且支持Java卡虚拟机指令集.这些处理器的特点是有两个程序执行状态:Java状态和本地状态.我们在两个处理器中分别采用了不同的硬件加速技术来提高Java的执行效率.这些技术包括堆栈寄存器堆、快速本地变量、指令折叠、堆栈指示等.它们能在两个状态间无缝切换,这使得用户能方便地使用Java应用程序和原始的RISC程序而不需要增加额外的处理器.而提高Java执行能力仅仅需要在原初的RISC芯片中增加很小的面积.最后,我们使用TSMC的0.18umCMOS工艺库,完成这些芯片的设计,并使用Xilinx的FPGA板作为我们的硬件验证平台,它能为其它类似的嵌入式微处理器和Java硬件IP的设计提供一个系统环境.