32 位 MIPS 需求文档

清华大学计算机系
逆光组

修订日期：2014 年 12 月 29 日

修订人	日期	描述
章彦恺	2014-10-17	创建文档
沈光耀	2014-10-18	完成第一部分
章彦恺	2014-10-19	完成第二部分
周恩泽	2014-10-20	完成第三部分
章彦恺	2014-10-20	整合文档
章彦恺，沈光耀，周恩泽	2014-10-21	修订文档，定稿
沈光耀	2014-10-25	修改了 BIOS 部分
章彦恺，周恩泽	2014-10-27	2 次整合
沈光耀，周恩泽	2014-12-29	完工修订

目录

1 引言

1.1 编写目的

撰写这篇需求文档的目的在于以下几点：

1. 对该系统进行一个能够达成一致认可的描述，明确需要完成的功能
2. 明确开发资源与项目目标
3. 作为开发手册的一部分，降低开发中小组讨论的成本

由需求文档的目标不难指出，文档的预期读者不仅包括开发者，也包括使用该 CPU 示例资源的用户。

1.2 背景

本项目系统的名称叫 ShaDoW。采用该名字的原因是：开发者认为 CPU 作为系统的核心，在用户不知情的情况下支撑起系统的运作，可以说是“运行在阴影中”的部件。因此起名叫 ShaDoW，即阴影之意。

本项目任务为软件工程课程老师白晓颖老师和计算机组成原理课程老师刘卫东老师共同提出。

承担本项目的开发者为计 22 班章彦恺，周恩泽和沈光耀。此外还受到了刘卫东、白晓颖、李山山三位老师的指导和王钧奕、张乐两位助教的帮助。

1.3 定义

本段给出文档中对于一些术语的基本定义。详细解释见后文相关模块，如有问题暂时存疑即可。

术语	定义
组合逻辑元件	组合逻辑元件指与运行时钟无关的元件。如一个与门：它并不考虑一个时钟什么时候上升什么时候下降，而只是单纯地关心着根据输入信号的变化给出相应的输出信号
时序逻辑元件	时序逻辑元件是根据系统的时钟信号，在时钟的上升沿工作的元件。注意组合逻辑元件和时序逻辑元件往往没有明确的分界：一个寄存器往往仅在上升沿写入，但却随时按照组合逻辑方式将其内容进行输出
CPU	中央处理器，为本工程的最终目标
ALU	算术逻辑单元，为 CPU 中负责进行运算的部分，根据两个输入得到输出结果。为组合逻辑元件
MMU	内存管理单元，负责对虚拟地址进行映射，并总管访问各种存储器和输入输出接口（如串口）的功能

CP0	系统控制协处理器，是 CPU 和操作系统交互的窗口：CPU 通过 CP0 向操作系统传递运行信息和异常信息等，而操作系统则通过操作 CP0 设置硬件的运行状态
TLB	意为快表。本工程需对内存进行分页管理，即虚拟地址通过页表映射得到物理地址。快表即为页表的高速缓存，用于加快地址转换
RISC	Reduced Instruction Set Computer 的缩写，指精简 CPU 指令集，其设计思想是：指令集应当尽可能简单划一，以此简化 CPU 的设计
MIPS	Microprocessor without interlocked piped stages 的缩写，为一种典型 RISC 指令集
ROM	存储二进制程序的硬件，断电可保存信息
RAM	存储二进制程序的硬件，断电不保存信息，速度较快，常用于电脑主存储器
Flash	存储二进制程序的硬件，断电可保存信息
BIOS	一般指主板芯片中的最初启动程序，主要任务是检测硬件、主板设置及开始引导操作系统。本实验中任务简化为引导操作系统即可
Bootloader	由 BIOS 启动的一段程序，用于将操作系统 load 到内存中并跳转到开始地址执行
Controller	CPU 模块之一。根据指令和硬件反馈信息给出控制信号，控制各个部件和整个 CPU 的运行

1.4 MIPS 架构下 CPU 运行概述

本段给出 MIPS 架构下 CPU 运行的基本方式。这是设计 MIPS 架构 CPU 需要掌握的基础知识，也是以下一切需求分析的基础，望读者详加掌握。

MIPS 架构下的计算机和所有计算机一样，基本功能都是从内存中提取指令并执行，保存所提取的指令地址的寄存器称为 PC。围绕这一功能，定义了合适的体系结构实现方案。首先，硬件运行分为用户态和内核态（往往与操作系统等价）以进行权限管理。二者的不同之处有：内核态可使用的内存空间更大，可使用的指令条数和硬件模块更多，地址映射方法也和用户态不同。之所以如此设计，是为了方便操作系统完成 TLB 管理、异常处理等系统级操作，并将以上内容向用户隐藏。如此既可方便用户，也防止了用户的误操作造成危险。而在此过程中需要用到的，就是协处理器 CP0。如在异常处理过程中，硬件检测到异常后产生详细信息，将其与发生异常的指令地址一起保存在 CP0 寄存器中，并跳转到操作系统的通用异常处理入口处。随后该段代码通过检查 CP0 的保存数据识别到发生的异常，并分发到对其进行处理的代码段。完成处理后，CPU 跳转回被打断的指令继续执行，如此即完成了一次异常处理。内存管理方面采用虚拟地址机制，将虚拟地址映射为物理地址、对应到实际的存储模块。如此即可方便地设置权限，防止访问未经授权的资源而造成险情。同时使得各个进程间互不干扰，虚拟地址可以复用，大大简化了程序的设计。MIPS 中从虚拟地址到物理地址的映射有多种方法，而方法的选择则是依据虚拟地址的值本身（内核态所谓“地址映射方法和用户态不同”也是由使用不同的虚拟地址段实现的）。其中用户态下虚拟地址的映射较为复杂，需要通过查表进行对应，这就是所谓的

页表管理机制。考虑到该查找操作非常频繁，而页表本身存放在内存中，对其进行访问代价很大，因此选择将页表的一部分放入 CPU 做成高速缓存，这就是快表 TLB。最后，访存操作还为 CPU 和外界的互动提供了统一接口：如从串口的输入输出就是通过读写某一特定地址来进行实现。硬件上总管从接收虚拟地址开始映射到读写数据完成这整个过程的模块就是 MMU。

1.5 参考资料

[1] 计算机硬件系统实验教程刘卫东，李山山，宋佳兴清华大学

[2] 计算机组成与设计 - 硬件/软件接口 David A. Patterson，John L. Hennessy

[3] 计算机体系结构：量化研究方法 John L. Hennessy David A. Patterson

[4] See MIPS run Linux Dominic Sweetman

2 功能需求

本工程的根本需求在于，书写一个能够运行 Linux 简化操作系统 ucore 的基于 MIPS 架构的 CPU。在引言中，我们已经简单叙述了 MIPS 架构下 CPU 的运行方法，而这也就决定了我们所设计的 CPU 需要满足以下细化要求：

1：能运行经简化的 MIPS 指令集，共 48 条指令。

2：能对某些异常和中断进行检测，进行现场保存，并在异常处理前后进行合理的跳转。

3：采用虚拟地址转换机制，包括简单映射和页式映射两种。

我们不妨认为，“运行 ucore”是本工程的 0 级需求，上述三条则是由 0 级需求所派生出的 1 级需求。而当 1 级需求落实到各个模块，就产生了实际设计中的 2 级需求。下文将首先论述 1 级需求，随后将其落实到 2 级需求。由于 2 级需求较为详细，需要更多篇幅，因此单列一章。注意 1 级需求第 3 条不单独论述，直接参见下一章的 MMU 部分。

2.1 简化 MIPS 指令集

本段给出 CPU 所需要支持的 47 条指令列表。注意，我们假设读者已经学习过 MIPS 汇编语言，并对 MIPS 指令格式有充分掌握。因此，我们将不再对 MIPS 指令系统中的一些基本术语进行详细解释。

详细的指令集列表见附录。

2.2 异常处理需求

MIPS 体系结构中的异常处理基本是按照以下流程进行：

1：硬件检测到异常发生。

2：硬件将异常发生时的信息保存到 CP0 的指定位置，并跳转到操作系统的通用异常处理入口。同时禁止一切异常的检测——这一时间段再发生异常将导致系统崩溃。

3：通用异常处理代码通过提取被保存的信息识别出发生的异常，据此分发到相应的异常处理代码。

4：所发生异常的处理代码运行完毕后，CPU 跳转到被异常打断的指令重新运行。同时重新使能异常检测。

因此对于硬件而言，所需要完成的工作为：

1. 能够识别操作系统所需要了解的异常。

2. 能够在异常发生时候将异常信息保存到指定位置。

3. 能够随时将异常处理代码地址或返回地址写入 PC，以完成异常处理的开始和返回。

4. 能够对异常的检测进行使能。

此处我们给出异常列表：

异常号	异常名	描述
0*	Interrupt	发生了系统中断，具体中断由中断位标识
1	TLB Modified	写未经授权的地址
2*	TLBL	在访存读取数据时发生了 TLB 缺失异常
3*	TLBS	在访存写数据时发生了 TLB 缺失异常
4	ADEL	读取一个非对齐地址
5	ADES	写一个非对齐地址
8*	Syscall	当前指令是一条系统调用指令
10	RI	不可识别的指令

上表中，加 * 的表示为运行操作系统所必须实现的，而其它的则未必：操作系统对它们的处理方式就是单纯地循环报错，因此我们可以假定它们不会发生——否则说明操作系统从本质上就是无法运行的。此处所说的“异常号”下文将称为 ExcCode，操作系统通过读取被硬件保存的它来判断发生了什么异常以及应当如何进行处理。当同时发生两个及以上异常时，系统将优先处理异常号大者。需要注意的是，Interrupt 异常与其他异常有所不同：一般异常均为 CPU 内部所发生，而 Interrupt 则标识着从外界输入的各种异常，并将其统称为“中断”。如：TLBL 发生在 CPU 内，为一般异常，而串口输入数据则来自不受控制的外界，属于中断异常。中断异常不仅有串口中断一种，还包括时钟中断、网口中断等等。此处给出我们需要实现的中断列表：

中断号	中断名	描述
2	NetInt	网口中断，标识网口有数据包到达（属于拓展要求，不在基础要求内）
4	SerialInt	串口中断，标识串口有数据到达
7	TInt	时钟中断，硬件按周期计时结束

由上述异常处理内容可以想见，当保存异常信息时，仅仅给出 ExcCode 是不够的，还需给出各个中断位的情况。这样才能在 ExcCode=0 时，判断出发生了哪一个或者哪些中断（对 ExcCode !=0 的情况，这一信息被忽略，不影响处理）。最后，为了能返回被打断的指令重新执行，该指令地址也应当作为异常信息的一部分进行保存。具体存储位置将在后文 CP0 模块叙述，异常的使能也是由 CP0 中特殊的域实现的，合在后文。

3 各模块需求分析

本章紧接上文，将从 1 级需求出发，分析得到 2 级需求，即各模块的功能需求。由于在完成 CPU 的过程中，需求与设计的关系并不是“前者指导后者”这么简单绝对，而是充满着互相影响、互相适应。比如“将 CPU 设计为多周期还是流水线”就决定着是否有书写数据转发单元的必要。因此在进行具体的需求分析前，本文首先将对我们的基本设计框架做一简单说明，随后再对各个模块详加分析。

3.1 CPU 设计概述

本次大实验中，我们完成的是一个多周期 CPU。多周期 CPU 的基本结构是: 在数据通路中设置好各个元件和相应的接口，然后由 Controller 根据指令和反馈信息操作一切元件。各元件所要做的，是实现其所必须完成的功能，并将控制接口提供给 Controller。至于组合应用这所有的元件，则是 Controller 的职责所在。如果将其他元件比作汽车检修中的扳手、千斤顶或监视器，那么 Controller 就是使用这些工具来进行修理的人，而不同的指令就好比不同的检修任务。检修往往要分不同阶段，体现在指令运行上就是分多周期完成，这也就决定了 Controller 的基本结构就是一个大型状态机。下文将给出各个元件所需要实现的功能。注意：如果你要设计的是一个流水线 CPU，则本章后面的内容对你就仅有少量参考价值，而且还有很多内容需要自行补充，如数据转发单元等。

3.2 ALU

ALU 负责硬件系统中双输入的算术计算功能。对于单输入的算术计算功能（例如左移固定两位，或者 PC+4 这样另一个输入固定）不通过 ALU 进行计算。此外，ALU 还负责进行两个数字的比较，并给出两个数字的大小关系。比较通过补码减法进行。所需完成的运算列表如下：

操作码	功能	描述	操作码	功能	描述
ADD	A+B	加法	NOR	A nor B	与非
SUB	A-B	减法	SLL	A sll B	逻辑左移 B 位
AND	A and B	与	SRL	A srl B	逻辑右移 B 位
OR	A or B	或	SRA	A sra B	算数右移 B 位
XOR	A xor B	与或	SLT	A slt B	A 与 B 比较

3.3 乘法器

由于指令集中有 MULT 这句指令，因而硬件系统中必须实现对乘法的支持。考虑到两个 32 位整数的乘法，结果是一个 64 位的整数。输出的格式与 ALU 并不兼容。这导致了乘法器不能和 ALU 共享结构，它必须是一个独立的元件。乘法器的输入数据和 ALU 一样，输出数据则是一个 64 位二进制串，表示两个数据的全乘积。

3.4 寄存器堆

Reg 寄存器堆元件，顾名思义是用来读写和存储寄存器数据的元件。将 FPGA 上的逻辑单元贡献出来作为硬件系统的数据存储单元来进行数据的缓存。寄存器堆是时序逻辑元件，只在硬件系统时钟上升沿才会对寄存器堆内的数据进行写操作。但另一方面，对于寄存器堆的读操作由一个 32 入口的选择器控制，是组合逻辑。

对于 32 位 MIPS 架构的硬件系统来说，需要在寄存器堆中实现 32 个 32 位的寄存器。它应该能同时读取出两个寄存器的值，并在时钟上升沿由 controller 进行使能，将写入数据写入被选择信号所选择的寄存器。

3.5 CP0

3.5.1 介绍

CP0 就是一个具有特殊定义的寄存器堆。它仅和操作系统交互，共同完成对 CPU 的控制、异常处理等功能。其中的很多内容是收集、汇总来自其他单元的信息，是操作系统了解 cpu 情况的窗口，如在异常发生时进行异常信息保存等；另有一些根据操作系统的需要设定，给往其他单元以控制其运行状态，如操作系统对 Status 寄存器中 ksu 域的设定决定了 CPU 是否处于 kernel 态。

为了完成该任务，需要提供一些特殊功能：

1. 需要提供两套写入入口。一套和普通寄存器堆一样：由选择信号选择要写的寄存器，由 controller 进行使能，将传递来的写入数据写入被选择的寄存器；另一套用于收集其他硬件的信息，由 controller 提供多个独立的写入使能，有多个明确定义的独立输入，直接写到寄存器相应的单个域而非写入整个寄存器。需要注意第二套入口优先级要高于第一套，即如果有特殊的写入需求时则通用入口写入被禁止。另外第一套入口一周期只会写一个完整寄存器，而第二套则可能写多个寄存器的多个分离的域。

2. 输出同样需要两套出口。一套和普通寄存器堆一样：由选择信号选择要读取的寄存器，按组合逻辑方式直接给到输出数据线（这就代表与时钟无关，选择信号变化或内容变化都会立刻影响到输出）；另一套给往其他单元以控制其运行状态，即将某些寄存器中特殊定义的域以组合逻辑方式直接给到输出数据线，从此发往其他元件。

3. 需要内置一个小加法器，完成 32 位 +1 运算，供 Count 寄存器每周期自增使用。

CP0 有 32 个寄存器，此处只给出运行 ucore 所必需的部分，列表如下。

寄存器号	寄存器名	详细介绍
0	Index	硬件读写 TLB 的索引。由于 TLB 共有 16 项，奇偶成一对，分为 8 对进行管理，因此只截取低 3 位使用。即输出信号 Index=(2:0)。
2	EntryLo0	(25:6) 为要写入 TLB 的偶数物理页的页号。
3	EctryLo1	(25:6) 为要写入 TLB 的奇数物理页的页号。
8	BadVAddr	存储发生异常时的虚拟地址，供操作系统处理异常使用。
9	Count	与 Compare 组成片内计时器，当二者相等则发出时钟中断信号。一般每周期自增 1，但当时钟中断时则保持原值，不再自增。
10	EntryHi	(31:13) 为要写入 TLB 的虚拟页号，用于索引查找物理页号。发生异常时存入要寻找的虚拟页号通知操作系统。
11	Compare	与 Count 组成片内计时器，当二者相等则发出时钟中断信号。
12	Status	IM=(15:8) 为 8 个中断的使能位，每位控制一个中断，不过本工程实际只使用其中三个。EXL=(1) 异常的总体使能位，由硬件软件共同控制，=1 则所有异常被忽略。IE=(0) 异常的总体使能位，只由软件控制，=0 则所有异常被忽略。

13	Cause	IPMod=(15:8) 为发生异常时经 Status 的 IM 使能后各个中断位的信号，由硬件产生并保存于此，供操作系统判断发生了什么或者哪些中断以进行处理。ExcCode=(6:2) 为异常编号，由硬件产生并保存于此，供操作系统判断发生了什么或者哪些异常以进行处理。注意中断是异常的其中一种。
14	EPC	保存异常发生时的 PC，供操作系统完成异常处理后从被打断的指令开始重新执行。
15	Ebase	保存异常处理程序的入口地址，供硬件在发现异常后跳转到操作系统的异常处理程序。
其他	其他	行为同普通寄存器。

表 2: CP0 各个特殊域表

3.6 ERR

本单元为异常识别单元，需要设计为组合逻辑元件，如此才能在异常发生当周期产生异常信息并通知控制器进行异常处理。各模块给出的各种异常信号、从外部接收的各种中断信号、Status 寄存器所存储的中断掩码等在此汇总，得出是否发生了异常、ExcCode 和中断位表 3 个输出。这里关键的是使能关系，如发生时钟中断异常的条件为：

1. TInt=1；
2. CP0.Status.IM(7)=1；
3. CP0.Status.Exl=0；
4. CP0.Status.IE=1；

注意，此处给出的是最基础的判断条件，由设计不同可以增删新条件。

3.7 BIOS

一般而言，BIOS 的基本任务是进行开机硬件检测、设置主板参数并引导系统开始 boot。本实验中简化为开始 boot 即可，BIOS 即为启动 ucore 所需的 bootloader 程序。为了防止 Flash 读写不稳定，将直接把 BIOS 部分写在片上 ROM。具体任务有：

1. 将 Flash 中的操作系统加载到内存
2. 跳转到操作系统的初始化代码

3.8 电脑端

在本工程中要求实现开发板从 PC 远程抓取程序的功能。因此在 PC 端需要实现一套完整的通信协议，来确保开发板和 PC 机能够正常地进行通讯，并方便调试。

PC 端的驱动程序将采用 C 语言编写。在 Windows 系统中，对串口的操作与对文件的操作十分相似，可以使用 fopen 函数打开端口，并使用 fgets 和 fputs 方便地进行读写操作。

3.9 MMU 存储控制单元

MMU 的全称是 Memory Management Unit，它管理从虚拟地址转换到物理地址再到访问相应元件、读取或写入数据的全过程，并在需要时给出所检测到的异常信号。因此，它必须完成最基本的访问 Flash、访问 Ram 和片内 Rom 功能，且同时集成了 TLB 模块。由于串口访问、网口访问等都是以读写特定虚拟地址的形式实现，它还一并集成了这两部分的控制。虽然这里有着复杂的时序和状态转移，但对 CPU 而言，所看到的只是简单地通过给定一个虚地址，一个读还是写的信号，以及一个 32 位的数据数据，并等待 MMU 的完成信号，就能够完成对于一个元件的控制。以下将首先给出内存管理中的虚拟地址映射机制，随后细化介绍各个集成模块的功能需求。

3.9.1 虚拟地址映射

区别于物理地址，虚拟地址可以进行灵活的划分。操作系统可以通过访问某一段特定的虚拟地址来访问硬件上的某个特定的元件，在硬件上由 MMU 实现。

虚拟地址	描述
$BFD003F8_h$	串口数据读写
$[BFD00000_h, BFDFFFFF_h] - BFD003F8_h$	串口状态描述
$[BE000000_h, BEFFFFFF_h]$	映射到 Flash 地址空间
$[BFC00000_h, BFCFFFFF_h]$	片上 ROM 地址空间
$BA000000_h$	DM9000AEP 网口的地址线
$BA000004_h$	DM9000AEP 的数据线
其他	遵照 TLB 映射结果访问 Ram

表 3: 虚拟地址划分表

其中串口状态描述为一个前 30 位均为 0，次低位表示串口是否有数据可读，最低位表示串口是否可写。

有时候虚拟地址的地址空间会大于硬件的地址空间，例如对于 Flash 来说，虚拟地址空间从 0xBE000000 到 0xBEFFFFFF，一共有 24 位，但 Flash 的有效地址线只有 22 位长。对于虚地址中多余的两位，在操作 Flash 的时候直接忽略，即进行了地址的取模操作。Rom、Ram 的多余虚地址空间也类似。

3.9.2 TLB

如上 1 级需求所述，在用户态下，所有的虚拟地址都需要通过查表得到对应的物理地址，即页表管理。一般而言，页表被保存在内存中，因此访问代价十分高昂。同时对每一条指令的读取都需要经过虚拟地址映射，导致映射操作的执行频率极高。在这种条件下，通过访问内存来实现地址转换的效率变得不可接受。因此常将页表的一部分提取到 CPU 中并缓存起来，以此加速转换，这就是快表 TLB。由于数据局部性，查找 TLB 几乎总能找到目标。在少数找不到目标的情况下，系统产生 TLBMiss 异常，由操作系统根据 CP0 中保存的信息将相应页表项提取到 TLB 中并重新进行访存，此时便可顺利进行地址映射。

3.9.3 DRAM 访问

在我们的 CPU 中，DRAM 所起的作用相当于实际电脑系统中的内存，要求能够对其进行顺畅稳定的读写。功能的详细实现需参见《计算机硬件系统实验教程》。

3.9.4 Flash 访问

在我们的 CPU 中，Flash 所起的作用相当于实际电脑系统中的硬盘。在最基本的要求下，我们需要将 Flash 中的数据读到内存中，以此完成加载操作系统的任务。至于写 Flash，对于我们的任务并非完全必要。功能的详细实现参见《计算机硬件系统实验教程》。

3.9.5 串口

在我们的 CPU 中，串口负责与计算机的通信，操作系统的运行信息都是通过串口显示在电脑屏幕上。串口访问的详细实现参见《计算机硬件系统实验教程》。

3.9.6 网口

本段为拓展需求，即通过开发板上搭载的 DM9000AEP 网口芯片与 PC 机进行网络通信。当收到数据包时，通过中断信号方式通知 CPU 触发异常，由操作系统对数据包进行处理。需要注意，完成本需求还需对操作系统进行改写：在初始化时添加网口中断使能、添加网口芯片初始化代码、手动实现网络通信协议和网口驱动等。具体实现可以参照 http://blog.csdn.net/anqi55/article/details/6687829。

4 数据通路

4.1 功能需求

精简 MIPS 指令集的各条指令基本可以分解为取指令 IF、指令译码/取寄存器 ID、运算 EX、访问内存 MEM、写回 WB 五种操作。有些指令仅需其中三段（如 beq），少数需要五段（如 lw），个别需要超过五段（sb 两次访存）。本实验计划以多周期 cpu 实现，有以下功能需求：

1. 将数据通路按照取指令、指令译码/取寄存器、运算、访问内存、写回五种操作划分为五段来安排硬件。
2. 在各操作段间必要处使用寄存器保存操作结果供下一周期使用。
3. 建立以状态机实现的控制单元，根据指令需要转移到不同状态，在周期间控制指令运行。
4. 由控制单元合理控制数据转发和选择。如将 ALU 的输出转发到存储器写入线、对 ALU 的操作数进行选择等。
5. 由控制单元合理控制存储器的读写使能，防止读写冲突。并设置高低位分别访问控制。
6. 使用指令寄存器对指令进行解析。由于解析阶段并不知道指令类型，因此指令中全部可能用到的字段都进行解析，统一处理。
7. 在数据通路适当位置将上述 PC、异常处理、中断控制、TLB 单元加入。
8. 添加其他小部件，如对立即数的符号扩展单元。

9. 在 ALU 中添加各种计算结果标志位，供分支指令判断使用。

4.2 通路设计

基本数据通路图即参照《计算机组成与设计硬件/软件接口》P223，做一定扩展。
基本控制单元状态机即参照同一本书 P224，并按照相同原理对更多指令添加扩展。

5 性能需求

针对多时钟周期的开发目标，我们在此提出 12MHz 的 CPU 目标运行频率。

6 运行环境需求

6.1 设备

对于 Xilinx Spartan6 xc6slx100 可用的十万个逻辑单元，预期使用五至八万个逻辑单元完成预定的工作。

主要的硬件设备信息为：

FPGA Xilinx Spartan6 xc6slx100，具有 100k 个逻辑单元和 4824kb 的 RAM。

CPLD XC95144，具有 144 个逻辑单元，用于控制串口和 PS2 接口。

RAM 61lv102416，32 位 RAM

Flash 28F640，8MB 的 8 位 Flash

网口 集成带有通用处理器接口的以太网控制器 DM9000A。

USB 集成 ISP1362 芯片。在本工程中暂时不打算使用

6.2 控制

本次的工程将会充分利用起开发板的可用资源，采用键盘作为输入设备，PVGA 显示作为输出，从终端通过串口和网口实现 PC 和开发板的双向通信和数据交流。

A 指令集

表中二进制（高位）表示 32 位的指令中高 16 位的内容，即指令 31 位置到 16 位置的内容。二进制（低位）内容类似。表中 rs，rt，rd 指示 32 个寄存器的编号，immediate 为立即数。0 和 1 表示确定的数字。X 表示 0、1 均可。

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	0	1	0	0	1	rs					rt				
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	immediate															
MIPS 语言	ADDIU rt rs immediate															
指令功能	R[t] ⇐ R[s] + Sign-extend(immediate)															
功能说明	对立即数进行符号扩展后与寄存器 rs 的值求和，结果保存到寄存器 rt 中															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	0	0	0	0	0	rs					rt				
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	rd					0	0	0	0	0	1	0	0	0	0	1
MIPS 语言	ADDU rd rs rt															
指令功能	R[d] ⇐ R[s] + R[t]															
功能说明	将寄存器 rs 与寄存器 rt 的值求和，结果保存到寄存器 rd 中															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	0	0	0	0	0	rs					rt				
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	rd					0	0	0	0	0	1	0	1	0	1	0
MIPS 语言	SLT rd rs rt															
指令功能	if(R[s] <R[d]) then R[d] = 1, else R[d] = 0															
功能说明	比较寄存器 rs 与寄存器 rt 的值并根据结果对寄存器 rd 赋值															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	0	1	0	1	0	rs					rt				
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	immediate															
MIPS 语言	SLTI rt rs immediate															
指令功能	if(R[s] <Sign-extend(immediate)) R[t] = 1, else R[t] = 0															
功能说明	比较寄存器 rs 与立即数进行符号扩展后的值并根据结果对寄存器 rt 赋值															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	0	1	0	1	1	rs					rt				
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	immediate															
MIPS 语言	SLTIU rt rs immediate															
指令功能	if(R[s] <Zero-extend(immediate)) R[t] = 1, else R[t] = 0															
功能说明	比较寄存器 rs 与立即数进行零扩展后的值并根据结果对寄存器 rt 赋值															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	0	0	0	0	0	rs					rt				
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	rd					0	0	0	0	0	1	0	1	0	1	1
MIPS 语言	SLTU rd rs rt															
指令功能	if(R[s] <R[t]) R[d] = 1, else R[d] = 0															
功能说明	比较寄存器 rs 与寄存器 rt 的值并根据结果对寄存器 rd 赋值															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	0	0	0	0	0	rs					rt				
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	rd					0	0	0	0	0	1	0	0	0	1	1
MIPS 语言	SUBU rd rs rt															
指令功能	R[d] ⇐ R[s] - R[t]															
功能说明	用寄存器 rs 的值减寄存器 rt 的值，结果保存到寄存器 rd 中															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	0	0	0	0	0	rs					rt				
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
MIPS 语言	MULT rs rt															
指令功能	HI/LO ⇐ R[s] * R[t]															
功能说明	将寄存器 rs 与寄存器 rt 的值相乘，保存到寄存器 HI/LO 中															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	rd					0	0	0	0	0	0	1	0	0	1	0
MIPS 语言	MFLO rd															
指令功能	R[d] ⇐ LO															
功能说明	将 LO 寄存器的值保存到寄存器 rd 中															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	rd					0	0	0	0	0	0	1	0	0	0	0
MIPS 语言	MFHI rd															
指令功能	R[d] ⇐ HI															
功能说明	将 HI 寄存器的值保存到 rd 寄存器中															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	0	0	0	0	0	rs					0	0	0	0	0
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1
MIPS 语言	MTLO rs															
指令功能	LO ⇐ R[s]															
功能说明	将寄存器 rs 的值保存到 LO 寄存器中															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	0	0	0	0	0	rs					0	0	0	0	0
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
MIPS 语言	MTHI rs															
指令功能	R[d] ⇐ HI															
功能说明	将寄存器 rs 的值保存到 HI 寄存器中															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	0	0	1	0	0	rs					rt				
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	immediate															
MIPS 语言	BEQ rs rt immediate															
指令功能	if(R[s] = R[t]) PC ⇐ PC + Sign-extend(immediate)															
功能说明	如果寄存器 rs 与 rt 的值相等，则跳转目的地址执行															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	0	0	0	0	1	rs					0	0	0	0	1
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	immediate															
MIPS 语言	BGEZ rs immediate															
指令功能	if(R[s] ≥ 0) PC ⇐ PC + Sign-extend(immediate)															
功能说明	如果寄存器 rs 的值大于等于 0，则跳转目的地址执行															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	0	0	1	1	1	rs					0	0	0	0	0
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	immediate															
MIPS 语言	BGTZ rs immediate															
指令功能	if(R[s] >0) PC ⇐ PC + Sign-extend(immediate)															
功能说明	如果寄存器 rs 的值大于 0，则跳转目的地址执行															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	0	0	1	1	0	rs					0	0	0	0	0
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	immediate															
MIPS 语言	BLEZ rs immediate															
指令功能	if(R[s] ≤ 0) PC ⇐ PC + Sign-extend(immediate)															
功能说明	如果寄存器 rs 的值小于等于 0，则跳转目的地址执行															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	0	0	0	0	1	rs					0	0	0	0	0
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	immediate															
MIPS 语言	BLTZ rs immediate															
指令功能	if(R[s] <0) PC ⇐ PC + Sign-extend(immediate)															
功能说明	如果寄存器 rs 的值小于 0，则跳转目的地址执行															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	0	0	1	0	1	rs					rt				
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	immediate															
MIPS 语言	BNE rs rt immediate															
指令功能	if(R[s] != R[t]) PC ⇐ PC + Sign-extend(immediate)															
功能说明	如果寄存器 rs 与 rt 的值不相等，则跳转目的地址执行															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	0	0	0	1	0	immediate(26bit)									
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	immediate(26bit)															
MIPS 语言	J immediate															
指令功能	PC ⇐ PC + Sign-extend(immediate)															
功能说明	无条件跳转目的地址执行															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	0	0	0	1	1	immediate(26bit)									
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	immediate(26bit)															
MIPS 语言	JAL immediate															
指令功能	PC ⇐ PC + Sign-extend(immediate)，RA ⇐ RPC															
功能说明	无条件跳转目的地址执行，将延迟槽后一条指令存入 RA															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	0	0	0	0	0	rs					0	0	0	0	0
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	rd					0	0	0	0	0	0	0	1	0	0	1
MIPS 语言	JALR immediate															
指令功能	PC ⇐ R[s]，R[d] ⇐ RPC															
功能说明	无条件跳转目的地址 rs 执行，将延迟槽后一条指令存入 RA															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	0	0	0	0	0	rs					0	0	0	0	0
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
MIPS 语言	JR rs															
指令功能	PC ⇐ R[s]															
功能说明	无条件跳转目的地址 rs 执行															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	1	0	0	0	1	1	rs					rt				
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	immediate															
MIPS 语言	LW rt rs immediate															
指令功能	R[t] ⇐ MEM[R[s]+Sign-extend(immediate)]															
功能说明	将寄存器 rs 的值与立即数 immediate 符号扩展后相加所得存至 rt 中															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	1	0	1	0	1	1	rs					rt				
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	immediate															
MIPS 语言	SW rt rs immediate															
指令功能	MEM[R[s]+Sign-extend(immediate)] ⇐ R[t]															
功能说明	将 rt 中的值存至寄存器 rs 的值与立即数 immediate 符号扩展后相加所得地址中															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	1	0	0	0	0	0	rs					rt				
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	immediate															
MIPS 语言	LB rt rs immediate															
指令功能	R[t] ⇐ Sign-extend(MEM_BYTE[R[s]+Sign-extend(immediate)])															
功能说明	将寄存器 rs 的值与立即数符号扩展后相加所得中的首字节符号扩展后存至 rt 中															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	1	0	0	1	0	0	rs					rt				
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	immediate															
MIPS 语言	LBU rt rs immediate															
指令功能	R[t] ⇐ Zero-extend(MEM_BYTE[R[s]+Sign-extend(immediate)])															
功能说明	将寄存器 rs 的值与立即数符号扩展后相加所得中的首字节零扩展后存至 rt 中															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	1	0	1	0	0	0	rs					rt				
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	immediate															
MIPS 语言	SB rt rs immediate															
指令功能	MEM_BYTE[R[s]+Sign-extend(immediate)] ⇐ LOW_BYTE[R[t]]															
功能说明	将 rt 的最低字节存至寄存器 rs 的值与立即数符号扩展后相加所得地址中															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	0	0	0	0	0	rs					rt				
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	rd					0	0	0	0	0	1	0	0	1	0	0
MIPS 语言	AND rd rs rt															
指令功能	R[d] ⇐ R[s] & R[t]															
功能说明	将寄存器 rs 与寄存器 rt 的值相与后的结果保存至寄存器 rd 中															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	0	1	1	0	0	rs					rt				
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	immediate															
MIPS 语言	ANDI rt rs immediate															
指令功能	R[t] ⇐ R[s] & Zero-extend(R[t])															
功能说明	将寄存器 rs 的值与立即数零扩展后相与的结果保存至寄存器 rt 中															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	0	1	1	1	1	0	0	0	0	0	rt				
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	immediate															
MIPS 语言	LUI rt immediate															
指令功能	R[t] ⇐ immediate * 65536															
功能说明	将 16 位立即数放在寄存器 rt 的高 16 位中															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	0	0	0	0	0	rs					rt				
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	rd					0	0	0	0	0	1	0	0	1	1	1
MIPS 语言	NOR rd rs rt															
指令功能	R[d] ⇐ not (R[s] or R[t])															
功能说明	将寄存器 rs 与寄存器 rt 的值或非后的结果保存至寄存器 rd 中															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	0	0	0	0	0	rs					rt				
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	rd					0	0	0	0	0	1	0	0	1	0	1
MIPS 语言	OR rd rs rt															
指令功能	R[d] ⇐ R[s] or R[t]															
功能说明	将寄存器 rs 与寄存器 rt 的值相或后的结果保存至寄存器 rd 中															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	0	1	1	0	1	rs					rt				
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	immediate															
MIPS 语言	ORI rt rs immediate															
指令功能	R[t] ⇐ R[s] or Zero-extend(immediate)															
功能说明	将寄存器 rs 与立即数零扩展相或后的结果保存至寄存器 rt 中															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	0	0	0	0	0	rs					rt				
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	rd					0	0	0	0	0	1	0	0	1	1	0
MIPS 语言	XOR rd rs rt															
指令功能	R[d] ⇐ R[s] xor R[t]															
功能说明	将寄存器 rs 与寄存器 rt 的值异或后的结果保存至寄存器 rd 中															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	0	1	1	1	0	rs					rt				
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	immediate															
MIPS 语言	XORI rt rs immediate															
指令功能	R[t] ⇐ R[s] xor Zero-extend(immediate)															
功能说明	将寄存器 rs 与立即数零扩展相异或后的结果保存至寄存器 rt 中															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	0	0	0	0	0	0	0	0	0	0	rt				
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	rd					immediate					0	0	0	0	0	0
MIPS 语言	SLL rd rt immediate															
指令功能	R[d] ⇐ R[t] << immediate															
功能说明	将寄存器 rt 中的值逻辑左移 immdiate 位后的结果保存至寄存器 rt 中															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	0	0	0	0	0	rd					rt				
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	rd					0	0	0	0	0	0	0	0	1	0	0
MIPS 语言	SLLV rd rt rs															
指令功能	R[d] ⇐ R[t] << R[s]															
功能说明	将寄存器 rt 中的值逻辑左移寄存器 rs 中的值位后的结果保存至寄存器 rt 中															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	0	0	0	0	0	0	0	0	0	0	rt				
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	rd					immediate					0	0	0	0	1	1
MIPS 语言	SRA rd rt immediate															
指令功能	R[d] ⇐ R[t] >> immediate(arithmetic)															
功能说明	将寄存器 rt 中的值算术右移 immdiate 位后的结果保存至寄存器 rt 中															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	0	0	0	0	0	rd					rt				
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	rd					0	0	0	0	0	0	0	0	1	1	1
MIPS 语言	SRAV rd rt rs															
指令功能	R[d] ⇐ R[t] >> R[s](arithmetic)															
功能说明	将寄存器 rt 中的值算术右移寄存器 rs 中的值位后的结果保存至寄存器 rt 中															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	0	0	0	0	0	0	0	0	0	0	rt				
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	rd					immediate					0	0	0	0	1	0
MIPS 语言	SRL rd rt immediate															
指令功能	R[d] ⇐ R[t] >> immediate(logical)															
功能说明	将寄存器 rt 中的值逻辑右移 immdiate 位后的结果保存至寄存器 rt 中															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	0	0	0	0	0	rd					rt				
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	rd					0	0	0	0	0	0	0	0	1	1	0
MIPS 语言	SRLV rd rt rs															
指令功能	R[d] ⇐ R[t] >> R[s](logical)															
功能说明	将寄存器 rt 中的值逻辑右移寄存器 rs 中的值位后的结果保存至寄存器 rt 中															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
MIPS 语言	SYSCALL															
指令功能	中断号 ⇐ SYSCALL															
功能说明	执行后触发中断															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	1	0	1	1	1	1	rs					rt				
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	immidiate															
MIPS 语言	CACHE															
指令功能	无															
功能说明	不做 CACHE，视为 NOP															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
MIPS 语言	ERET															
指令功能	PC ⇐ EPC															
功能说明	返回 EPC 寄存器的地址执行，并设置 Status 寄存器的 EXL 位为 0															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	1	0	0	0	0	0	0	0	0	0	rt				
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	rd					0	0	0	0	0	0	0	0	0	0	0
MIPS 语言	MFC0 rd rt															
指令功能	R[t] ⇐ CP0[R[d]]															
功能说明	将 CP0 中的 rd 寄存器的值保存到 rt 寄存器中															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	1	0	0	0	0	0	0	1	0	0	rt				
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	rd					0	0	0	0	0	0	0	0	0	0	X
MIPS 语言	*MTC0 rd rt															
指令功能	CP0[R[d]] ⇐ R[t]															
功能说明	将寄存器 rt 的值保存到 CP0 中的 rd 寄存器中															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
MIPS 语言	TLBWI															
指令功能																
功能说明	写索引 TLB 项															

二进制（高位）	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	1	0	0	1	0	1	rs					rt				
二进制（低位）	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	immediate															
MIPS 语言	LHU rt rs immediate															
指令功能	R[t] ⇐ Zero-extend(MEM_HALFWORD[R[s]+Sign-extend(immediate)])															
功能说明	将 rs 中的值与立即数符号扩展后所得地址的低两字节取出零扩展后存至 rt															