32-bits MIPS CPU

需求文档

NonExist 组

2018 年 1 月 7 日

Contents

1 引言 **3**

1.1 背景 . . . 3
1.2 编写目的 . . . 3
1.3 项目概览 . . . 3
1.4 定义 . . . 3
1.5 参考资料 . . . 3

2 开发环境 **4**

2.1 总览 . . . 4
2.2 EDA 工具 . . . 4
2.3 MIPS 运行环境 . . . 4
2.3.1 编译 . . . 4
2.3.2 仿真 . . . 4

3 ucore 需求分析 **5**

3.1 Boot 阶段 . . . 5
3.1.1 相关文件 . . . 5
3.1.2 Boot 过程 . . . 5
3.1.3 需求分析 . . . 6
3.2 指令分析 . . . 6
3.2.1 相关文件 . . . 6
3.2.2 需求分析 . . . 6
3.3 地址映射 . . . 7
3.3.1 相关文件 . . . 7
3.3.2 内存管理 . . . 7
3.3.3 外设调度 . . . 8
3.3.4 需求分析 . . . 9
3.4 异常处理 . . . 9
3.4.1 相关文件 . . . 9
3.4.2 异常处理流程 . . . 9
3.4.3 需求分析 . . . 10

4 指令系统需求分析 **11**
4.1 指令系统 . . . 11
4.2 算术逻辑指令 . . . 11
4.2.1 功能 . . . 11
4.2.2 硬件需求 . . . 11
4.2.3 异常 . . . 11
4.3 分支跳转指令 . . . 11
4.3.1 功能 . . . 11
4.3.2 硬件需求 . . . 11
4.3.3 异常 . . . 11
4.4 访存指令 . . . 11
4.4.1 功能 . . . 12
4.4.2 硬件需求 . . . 12
4.4.3 异常 . . . 12
4.5 陷入指令 . . . 12
4.5.1 功能 . . . 12
4.5.2 硬件需求 . . . 12
4.5.3 异常 . . . 12
4.6 特权指令 . . . 12
4.6.1 功能 . . . 12
4.6.2 硬件需求 . . . 12
4.6.3 异常 . . . 12
4.6.4 空指令 . . . 12

5 CPU 需求分析 **13**
5.1 指令流水 . . . 13
5.2 寄存器 . . . 13
5.3 CP0 . . . 13
5.4 Control . . . 13
5.5 MMU . . . 13

6 外设需求分析 **14**
6.1 ROM . . . 14
6.2 RAM . . . 14
6.3 Flash . . . 14
6.4 串口 . . . 14
6.5 VGA . . . 14

Chapter 1

引言

1.1 背景

本项目的顶级需求为在 Thinpad 开发板上运行 ucore 操作系统。其衍生需求为设计一个基于 MIPS 32 架构的 CPU，以实现 ucore 所需的 46 条指令、精确异常与外设调度。

本项目是计算机组成原理和软件工程课程的联合实验，项目需求方为计算机组成原理与软件工程课程。

本项目的承担方为 NonExist 小组，包括计 55 班张钰晖、计 55 班杨一滨、计 54 班周正平 3 位成员。

1.2 编写目的

本需求文档的编写目的如下：

1. **需求分析：** 明确软件（ucore）对硬件（CPU）的具体需求，及需完成的功能
2. **系统概述：** 对系统总体框架进行清晰完整的描述，确定迭代开发计划
3. **学习目标：** 明确开发所需技术，订立学习目标

其目标读者包括 NonExist 小组的开发者、成品 CPU 的使用者、项目的评估者。

1.3 项目概览

首先给出本项目的需求分析图：

<TODO>

1.4 定义

<TODO>

1.5 参考资料

<TODO>

Chapter 2

开发环境

2.1 总览

1. EDA 工具

硬件端平台 Thinpad 开发板软件端平台 Windows 工具开发工具：Vivado 串口工具：串口调试精灵调试工具：二进制显示软件

2. MIPS 运行环境

平台 Ubuntu 工具编译工具：mips-sde-elf 仿真工具：qemu-thumips

2.2 EDA 工具

<TODO> 型号

2.3 MIPS 运行环境

操统的编译和仿真

2.3.1 编译

2.3.2 仿真

Chapter 3

ucore 需求分析

正如引言部分所述，“运行 ucore 操作系统” 为本项目的顶级需求。这一部分将主要介绍

3.1 Boot 阶段

3.1.1 相关文件

文件	简介
boot/bootasm.S	BootLoader 的 MIPS 汇编代码，需编译后存入 ROM
init/init.c	操作系统入口

3.1.2 Boot 过程

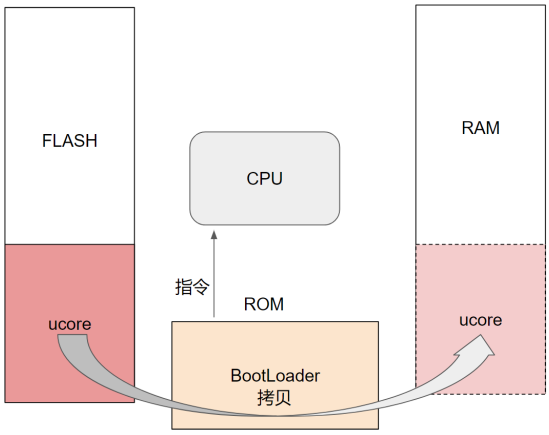

Figure 3.1: BootLoader 将 ucore 从 FLASH 拷贝至 RAM 中

由于 FLASH 断电不消失的特性，ucore 需要存放在 FLASH 中；而系统加载完毕后，ucore 应该在内存（RAM）中。因此，需要一个 Boot 程序，负责在硬件启动时将 ucore 从 FLASH 导入至 RAM。

CPU 初始化时，置 PC 为 Boot 程序首地址，从 ROM 开始处（VA 0xBFC00000）运行 Boot 程序。每次从 FLASH 读出数据再写入 RAM，直至拷贝完成。

BootLoader 拷贝完成后，将 PC 跳转至 VA 0x80000000，进入操作系统入口。此时 ucore 进行初始化并输出调试信息。

3.1.3 需求分析

由上述分析可知，Boot 过程的需求包括以下几点：

1. **ROM**：使用 FPGA 实现 ROM，并设置硬件初始化时的 PC 值为 ROM 的虚拟地址（见“地址映射”一节）
2. **BootLoader**：将 BootLoader 编译后写入 ROM
3. **ucore**：将 ucore 编译后写入 FLASH

3.2 指令分析

3.2.1 相关文件

ucore 的所有文件

3.2.2 需求分析

ucore 编译完成后，便会被转化成一条条 MIPS 指令。通过进行 qemu 仿真可知，ucore 运行共需要 45 条指令，包括如下几部分：

1. **算术逻辑指令**：共 22 条，包括加减乘、与或非、移位等指令
2. **分支跳转指令**：共 10 条，包括分支（B）与跳转（J）指令
3. **访存指令**：共 5 条，包括读取（L）与写入（S）指令
4. **移动指令**：共 2 条，包括向 HILO 寄存器的移动
5. **陷入指令**：共 1 条，包括 SYSCALL
6. **特权指令**：共 5 条，包括对 CP0 的访问、异常返回及 TLB 异常时使用的指令

通过这 45 条指令，ucore 实现了作为操作系统的调度功能。

3.3 地址映射

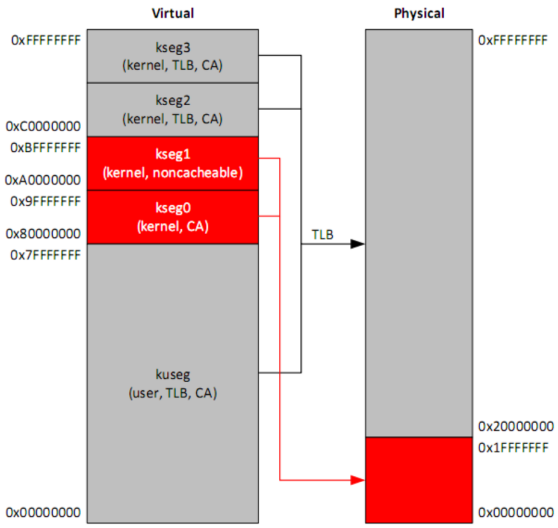

Figure 3.2: MIPS 标准地址映射

地址映射在硬件上由 MMU（Memory Manage Unit）完成，其最重要的意义体现在 2 方面：

1. **内存管理（面向软件）**：使进程在寻址时可以超过物理内存大小，且多个进程的地址空间互不干扰
2. **外设调度（面向硬件）**：面向 CPU 统一各个外设的访问接口，使之均可使用访存指令进行读写

这一部分将深入解析 ucore 的地址映射标准，从而明确对硬件 MMU（Memory Manage Unit）的需求。

3.3.1 相关文件

首先列出 ucore 定义地址映射的关键文件：

文件	关键变量	值	简介
boot/bootasm.S	FLASH_START	0xBE000000	FLASH 的起始虚拟地址
	FLASH_SIZE	0x01000000	FLASH 的地址大小
kern/mm/memlayout.h	KMEMSIZE	1M	内存总大小
kern/include/mips_vm.h	MIPS_KUSEG	0x00000000	kuseg 段起始地址
	MIPS_KSEG0	0x80000000	kseg0 段起始地址
	MIPS_KSEG1	0xa0000000	kseg1 段起始地址
	MIPS_KSEG2	0xc0000000	kseg2 段起始地址
include/thumips.h	COM1	0xBFD003F8	串口虚拟地址

3.3.2 内存管理

内存管理需求主要面向运行在操统之上的应用软件。首先，应用程序的大小不应受到物理内存的限制，例如开发板上的 RAM 总大小仅有 8M，但对于 32 位机器而言，可用虚拟地址空间可达 $2^{32} = 4G$；其次，

各应用程序的地址空间应各自独立。

页表 操统因而需维护页表（Page Table），以实现从虚拟地址到物理地址的转换。具体而言，每当程序访问内存时，需进行 2 次访问：

1. **查询页表**：读取页表，查找对应的物理地址
2. **获取数据**：访问该物理地址，获取数据

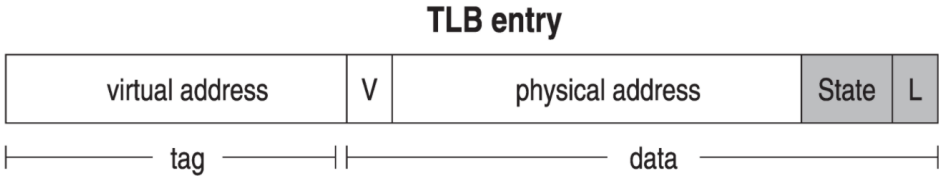

Figure 3.3: TLB 表项

TLB 由于内存访问具有局部性，如能将页表最近被访问的一部分以 CPU 内部的逻辑单元存储，则访存效率可获得极大提高。TLB（Translation Lookaside Buffer）就是这样一种高速缓存。具体而言，在加入 TLB 之后，程序访存流程如下：

1. **查询 TLB**：根据虚拟地址的高 20 位（32 减去页位数 12），查找 TLB 中的表项
2. **TLB HIT**：如 TLB 中有此表项，则直接其对应的物理地址
3. **TLB MISS**：如 TLB 中无此表项，则再进一步访问内存查询页表，获取物理地址之后，将其作为一个新表项写入 TLB

3.3.3 外设调度

在 CPU 核心之外，运转着 ROM、RAM（相当于内存）、FLASH（相当于硬盘）、串口（相当于标准输出）、VGA（相当于显示屏）等多个外部设备。它们的用途、接口、访问时序各不相同。

然而，对于操作系统与 CPU 而言，它们的访问接口是统一的：均使用 L/S 型指令实现收/发数据。这个过程中需要进行地址映射。

RAM 除操统文件中的定义、“内存管理” 小节的说明外，考虑到开发板上提供了 2 块大小为 4M 的 RAM（Base RAM 与 Ext RAM），可将 KMEMSIZE 改为 8M，从而提供更大的内存空间。

FLASH、串口 见操统文件中的定义。

ROM 在 Boot 阶段，CPU 复位时 PC 位于 VA 0xBFC00000 处，因而该虚拟地址应映射到 ROM。此外，考虑 BootLoader 指令较少，为 ROM 分配 1KB 的地址空间即可。

VGA 此外，项目计划实现拓展功能 VGA，其屏幕大小为 800x600，故而需要 480000 B 的地址空间用于显存。姑且在 kseg1 段中分配一段开始于 VA 0xBA000000、大小为 512KB 的地址空间留作显存。

3.3.4 需求分析

综上所述，MMU 需要将 VA 的 kuseg、kseg2 段通过 TLB 映射到内存（RAM），在 kseg0 段通过抹去最高位直接得到 RAM 中的 PA，在 kseg1 段将部分地址映射到除 RAM 外的各个外设。

以下给出最终地址映射方案：

段（权限）	虚拟地址	映射目标（物理地址）	备注
kuseg（用户态）	0x00000000 - 0x7FFFFFFF	RAM（通过查询 TLB 动态确定）	用户程序
kseg0（内核态）	0x80000000 - 0x807FFFFF	RAM（0x00000000 - 0x007FFFFF）	开机时存放操作系统
kseg1（内核态）	0xBE000000 - 0xBEFFFFFF	FLASH	关机时存放操作系统
	0xBFC00000 - 0xBFC00FFF	ROM	存放 BootLoader
	0xBFD003F8 - 0xBFD003FC	串口	串口数据/串口状态
	0xBA000000 - 0xBA080000	VGA	显存，用于显示图像
kseg2（内核态）	0xC0000000 - 0xFFFFFFFF	RAM（通过查询 TLB 动态确定）	

3.4 异常处理

在 MIPS32 架构中，有一些事件会打断程序的正常执行流程。一些由**外部事件**触发，如串口有数据待读入，称为中断；另一些则由 CPU **内部指令**引起，如算术溢出、系统调用等。这些事件统称为异常。

以下对异常处理流程进行归纳，并总结 ucore 涉及的所有异常，从而明确对硬件的需求。

3.4.1 相关文件

以下列出 ucore 异常处理的部分关键文件：

文件	关键函数	简介
trap/vector.S	___exception__vector	异常处理向量，汇集各入口
trap/exception.S	ramExcHandle_tlbmiss	TLB 异常处理入口
	ramExcHandle_general	通用异常处理入口
	common_exception	保存现场、调用操统异常处理代码、恢复现场
trap/trap.c	mips_trap	操统异常处理代码
	trap_dispatch	被 mips_trap 调用，分类处理各种异常

3.4.2 异常处理流程

硬件端（CPU）

1. **异常响应**：硬件检测异常，并将异常原因、类型等存入 CP0 相关寄存器中
2. **异常处理**：将 PC 跳转到 CP0 Ebase 寄存器所指示的操统异常处理入口地址，并禁用异常检测
3. **异常返回**：执行 ERET 指令，跳转回被异常打断的指令，重新使能异常检测

软件端（ucore）

1. **异常响应**：保存现场（通用寄存器等）至内存中
2. **异常处理**：从异常处理入口处开始，根据异常类型，执行异常处理代码
3. **异常返回**：从内存中恢复现场，并使用 ERET 指令返回

3.4.3 需求分析

根据上述分析，只需针对 ucore 能够处理的那些异常，在硬件上加以实现，即可满足异常处理需求。

阅读 trap.c : trap_dispatch 函数可知 ucore 处理的所有异常类型。它们通过 CP0 Cause 寄存器的 ExcCode（异常号）字段加以区分：

中断 中断由 CP0 Cause 寄存器的 IP 标志位（中断号）进一步区分，包括如下 2 种：

异常名（异常号）	中断名（中断号）	硬件触发条件	ucore 处理流程
Interrupt（0）	时钟中断（7）	CP0 Compare 与 Count 寄存器的值相等	进程调度后重启时钟
	串口中断（4）	串行接口处有数据待写入	读串口并写入 stdin

TLB MISS 在访存 load 或 store 时 TLB 中无匹配表项时产生，包括如下 2 种异常：

异常名（异常号）	硬件触发条件	ucore 处理流程
TLBL（2）	访存 load 时 TLB 中无匹配表项	首先根据 CP0 BadVAddr 寄存器确定缺失地址，然后通过设置 CP0 相关寄存器构造新的 TLB 表项，最后使用 tlbwr 指令随机选择一个位置，重填该项
TLBS（3）	访存 store 时 TLB 中无匹配表项	同上

系统调用 系统调用由 Syscall 序号进一步区分，其类型在操作系统中定义，硬件无需关心其具体实现：

异常名（异常号）	硬件触发条件	ucore 处理流程
Syscall（8）	执行指令 SYSCALL	根据 Syscall 序号，调用 syscall/syscall.c 中相应的处理代码

其他异常 此类异常无需在硬件上实现。因为它们在 ucore 中的实现均为简单报错，如发生在用户态，则用户进程退出；如发生在内核态，则系统 panic。总而言之，即使硬件实现了这些异常，亦无法使 ucore 在这些情况下正常运行。为完整起见，亦在此列出：

异常名（异常号）	硬件触发条件	ucore 处理流程
ADEL（4）	访存 load 时地址未对齐	简单报错
ADES（5）	访存 store 时地址未对齐	简单报错
RI（10）	无效指令	简单报错
CPU（11）	协处理器不可用	简单报错
OV（12）	算术溢出	简单报错

Chapter 4

指令系统需求分析

4.1 指令系统

1. MIPS32 架构 2. 46 条指令，包括。。

4.2 算术逻辑指令

< 总表 >

4.2.1 功能

4.2.2 硬件需求

4.2.3 异常

4.3 分支跳转指令

< 总表 >

4.3.1 功能

4.3.2 硬件需求

4.3.3 异常

4.4 访存指令

< 总表 >

4.4.1 功能

4.4.2 硬件需求

4.4.3 异常

4.5 陷入指令

< 总表 >

4.5.1 功能

4.5.2 硬件需求

4.5.3 异常

4.6 特权指令

< 总表 >

4.6.1 功能

4.6.2 硬件需求

4.6.3 异常

4.6.4 空指令

< 总表 >

Chapter 5

CPU 需求分析

5.1 指令流水

5.2 寄存器

5.3 CP0

5.4 Control

5.5 MMU

Chapter 6

外设需求分析

6.1 ROM

6.2 RAM

6.3 Flash

6.4 串口

6.5 VGA