32-bits MIPS CPU

需求文档

NonExist 组

2018 年 1 月 11 日

Contents

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。因此，本章将对 ucore 对硬件与指令系统的需求进行详细的分析。

本章分为以下几部分：

1. **Boot 过程**：操作系统的启动与初始化流程
2. **指令系统**：为运行 ucore，CPU 应支持的指令集合
3. **地址映射**：包括 ucore 的内存管理与外设调度，叙述其对 MMU 的需求
4. **异常处理**：描述 ucore 的异常处理机制，及硬件需实现的所有异常

操作系统逻辑复杂，每部分内容不尽相同，但在介绍流程上都遵循以下主体框架：

1. **相关文件**：列举 ucore 与这部分紧密相关的文件，便于读者查阅
2. **原理说明**：说明 ucore 在这部分的实现机制或流程
3. **需求分析**：根据以上分析，明确此部分对硬件的需求

3.1 Boot 阶段

Boot 阶段为操作系统的启动阶段。

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

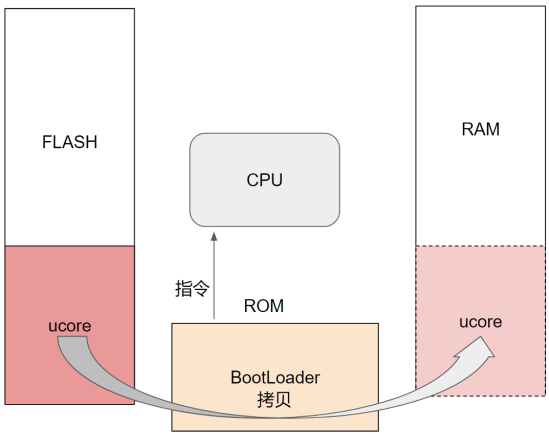

Figure 3.1: BootLoader 将 ucore 从 FLASH 拷贝至 RAM 中

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 指令系统

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 4 条，包括对 HILO 寄存器的读写指令
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。
该部分的需求为在硬件上实现这 45 条指令。详细分析见 “指令系统需求分析” 一章。

3.3 地址映射

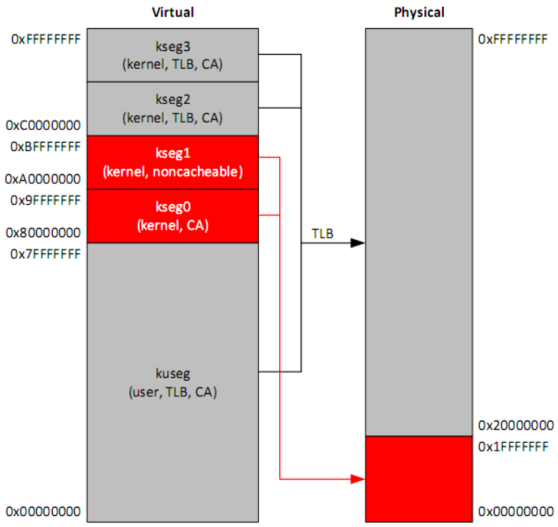

Figure 3.2: MIPS 标准地址映射

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 次访存：

1. **查询页表**：读取内存中的页表，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

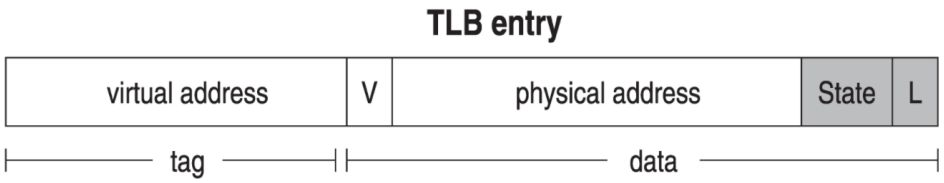

Figure 3.3: TLB 表项

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 480KB 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 - 0xFFFFFFF	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
kern/trap/vector.S	___exception__vector	异常处理向量，汇集各入口
kern/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
kern/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

指令系统是硬件（CPU）对 ucore 唯一透明可见的接口，也因而成为在 “运行 ucore” 之下的第二级需求。

在对 ucore 进行需求分析后可知，在指令系统这一层面，一共有 45 条需要实现的指令。

本章包括如下几个部分：

1. **概述**：对五级流水线架构及其运行方式作一简要概述
2. **算术逻辑指令**：共 22 条，包括加减乘、与或非、移位等指令
3. **分支跳转指令**：共 10 条，包括分支（B）与跳转（J）指令
4. **访存指令**：共 5 条，包括读取（L）与写入（S）指令
5. **移动指令**：共 4 条，包括对 HI/LO 寄存器的读写指令
6. **陷入指令**：共 1 条，包括 SYSCALL
7. **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
8. **总结**：以上指令对硬件需求的扼要总结

本章除概述与总结外，每节遵从以下介绍流程：

1. **功能**：简要描述此类指令的功能
2. **硬件需求**：为实现此类指令，对硬件的结构和功能需求
3. **异常**：此类指令可能触发的异常，或对异常处理的影响

4.1 概述

4.1.1 五级流水线

本次项目计划实现基于*五级流水线*的指令系统。何为五级流水线？具体来说，体现在以下 2 个方面：

1. **每条指令被拆分为 5 个步骤**：共包括 5 个硬件单元，每个单元负责其中的一个环节。

名称	代号	功能
取指	IF	从指令存储器中读取指令
译码	ID	指令译码，同时读取寄存器
执行	EX	执行操作，或计算地址
访存	MEM	进行访存操作
回写	WB	将计算结果写入寄存器

2. **5 个硬件单元并行执行**：在任何一个时钟周期内，上述的 5 个硬件单元分别在处理第 $n+5, n+4, n+3, n+2, n+1$ 条指令的第 1, 2, 3, 4, 5 个阶段（如下图所示）：

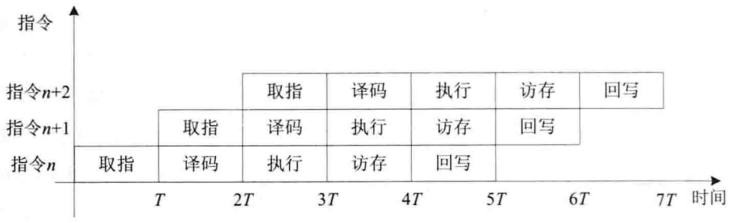

Figure 4.1: 流水线结构

硬件单元的并行执行同时有其优点和缺点：

- **优点**：并行结构加速指令的执行。设每条指令每个步骤的平均执行时间为 $\triangle t$，则对于一个多周期 CPU（无流水线结构）而言，n 条指令所需时间为 $5n\triangle t$；而对于流水线结构而言，n 条指令仅需 $(n+4)\triangle t$。可以看出，当指令数目很大时，流水线结构可带来约 4 倍左右的性能提升。
- **缺点**：并行结构同样带来了冒险问题。具体地，有 2 种情形由此产生：
  - **数据冒险**：例如后一条指令的 ID 阶段需读取前一条指令在 WB 阶段才写入的寄存器值，但这在时序上矛盾。
  - **控制冒险**：例如跳转指令在 ID 阶段才能确定目标地址，然而此时其下一条指令（延迟槽指令）已进入流水线的 IF 阶段。

4.1.2 需求总述

总体来看，指令系统对硬件的需求如下：

1. **流水线架构**：实现流水线框架，使得硬件在每个周期可以并行执行 5 条指令。此外，还需解决其中可能遇到的控制问题：
   - **运行控制**：能够对流水线进行暂停、插空周期或清空等控制干预。
   - **解决冒险**：解决数据冒险、控制冒险。
2. **指令实现**：在流水线框架上实现 ucore 所需的 45 条指令。

3. **外设集成**：能在流水线框架的某些硬件单元集成对外部设备的访问，以辅助实现某些指令（如load/store）。

4.2 算术逻辑指令

算术逻辑指令共 22 条，包括加减乘、与或非、移位等，总结如下：

4.2.1 功能

指令	运算	功能
ADDIU、ADDU	A + B	无符号数加法
SUBU	A - B	无符号数减法
MULT	A × B	乘法
SLT、SLTI、SLTIU、SLTU	A < B	符号数比较、无符号数比较
AND、ANDI	A and B	与
OR、ORI	A or B	或
NOR	A nor B	或非
XOR、XORI	A xor B	异或
SLL、SLLV	A sll B	逻辑左移
SRL、SRLV	A srl B	逻辑右移
SRA、SRAV	A sra B	算术右移
LUI	A = imm $\|\|$ 0^{16}	加载立即数至寄存器

4.2.2 硬件需求

这部分对硬件的需求主要体现在 ALU 上，它位于流水线的第 3 部分（EX）。具体地，ALU 需接受 2 个 32 位整数及相关控制信号作为输入，并以 1 个 32 位整数作为输出。控制信号决定了 ALU 执行的运算；此外，运算结果应在 WB 阶段被写入寄存器。

特别地，乘法运算的结果是一个 64 位整数，因此需要 HI/LO 寄存器分别用于存储乘积的高、低 32 位，而不能存入通用寄存器。

算术逻辑运算指令对硬件的需求总结如下：

1. **流水线**：能对算术运算指令进行解码识别。能读写通用寄存器。能计算 2 个 32 位整数作指定运算的结果，并产生一个 32 位整数作为输出。

2. **HI/LO**：当运算类型为乘法时，可以保存 64 位的乘法运算结果。

4.2.3 异常

无。

4.3 分支跳转指令

分支跳转指令共 10 条，包括分支（Branch）与跳转（Jump）2 类指令，总结如下：

4.3.1 功能

指令	功能
BEQ、BNE、BGEZ、BGTZ、BLEZ、BLTZ	分支指令，用于有条件跳转。 条件包括 $==, !=, \geq, >, \leq, <$。
J、JR JAL、JALR	跳转指令，用于无条件跳转； 同上，但会使用寄存器（常为 R[31]）预先保存此前的 PC 值

4.3.2 硬件需求

这部分对硬件的需求主要集中于流水线的设计。此外，由于延迟槽指令的存在，异常处理需增加一些特殊的判断逻辑，详细分析见 “异常” 部分。

跳转指令对硬件的需求主要集中于流水线，总结如下：

1. **流水线**：能对分支跳转指令进行解码识别。能读写通用寄存器。此外能改变指令的执行顺序并判断状态：
   - **PC**：可以在每个周期取指时，根据 “是否需要跳转” 这一控制信号，对 PC 的下一次取指做出判断：如需跳转，则将 PC 赋值为跳转目标地址；否则 PC 照常自增 4。
   - **延迟槽**：流水线中各个硬件单元可以确定当前正在执行的指令是否在延迟槽中。更具体的分析见 “异常部分”。

4.3.3 异常

分支跳转指令本身不会产生异常。但由于分支跳转指令引入了*延迟槽指令*，这会对异常处理带来额外的判断逻辑。

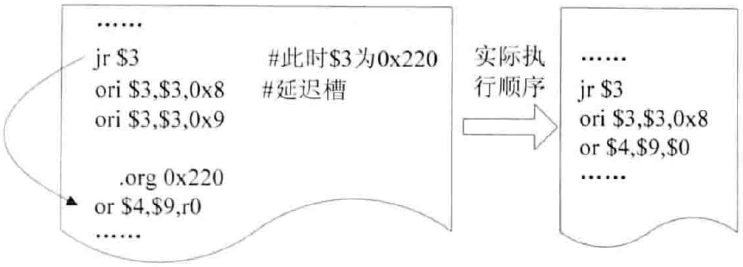

Figure 4.2: 延迟槽指令

延迟槽指令　我们规定分支指令后面的指令位置为*分支延迟槽*，其中的指令称为*延迟指令*。由于在计算分支目标地址时，延迟指令已经进入流水线，因而无论跳转发生与否，它都必然被执行。

异常处理相关 具体地，在异常处理时，需将异常发生时的指令地址（PC 值）存入 CP0 的 EPC 寄存器中，以便异常处理结束后返回该地址继续执行。然而如果触发异常的指令为延迟槽指令，则需要将 EPC 寄存器的值置为 PC-4，而非 PC 的当前值。这是因为在延迟槽指令之前可能已经发生了跳转，因而其后需继续执行的指令序列应位于跳转的目标地址，而非延迟槽指令之后。

因此，流水线中的各个硬件单元需能够确定当前指令是否位于延迟槽，以便在发生异常时可以向 EPC 寄存器中保存正确的地址。

4.4 访存指令

访存指令共 5 条，包括读取（Load）与写入（Store）2 类指令，总结如下：

4.4.1 功能

指令	功能
LB、LBU	读内存中的一个字节（8 位）
LW	读内存中的一个字（32 位）
SB	写内存中第一个字节（8 位）
SW	写内存中的一个字（32 位）

4.4.2 硬件需求

这一部分对硬件的需求包括 3 大部分。首先，流水线需要对访存指令进行译码，如果是读，则需要读取相应的寄存器；否则为写，需要将结果写入寄存器中；其次，需要在流水线框架之上集成其他外设，包括在 ucore 部分已经有所介绍的 ROM、RAM、FLASH、串口、VGA 等；最后，为了在指令向 CPU 提供的虚拟地址与外设的物理地址之间实现衔接，硬件需要通过 MMU 与 TLB 进行地址映射。

访存指令对硬件的主要需求可以总结如下：

1. **流水线**：能对访存指令进行解码识别。能读写通用寄存器。能通过 MMU 以统一的接口访问外设。能执行 TLB 异常处理流程。
2. **外设**：能在流水线的框架之上集成外设，实现其读写操作。
3. **MMU**：能将指令提供的虚拟地址映射到对应的外设，实现地址转换。
4. **CP0**：能提供寄存器记录可能的异常信息。

4.4.3 异常

在 ucore 需求分析部分得出的异常列表中，访存指令可能触发其中的 TLB MISS 异常、地址未对齐异常。其中，TLB MISS 异常为硬件需要实现的异常。

4.5 移动指令

移动指令共 4 条，主要包括对 HI/LO 寄存器的读写指令。总结如下：

4.5.1 功能

指令	功能
MFHI、MFLO	读 HI/LO 寄存器
MTHI、MTLO	写 HI/LO 寄存器

4.5.2 硬件需求

这一部分对硬件的需求集中于对 HI/LO 寄存器的访问。具体地，流水线需要在识别出这些指令的类型后，实现与 HI/LO 寄存器的读写交互。HI/LO 寄存器则需要实现相应的访问接口以支持读写。

总结如下：

1. **流水线**：能对移动指令进行解码识别。能读写通用寄存器。能访问 HI/LO 寄存器的读写接口。
2. **HI/LO**：能提供接口支持读写操作。

4.5.3 异常

无。

4.6 陷入指令

陷入指令共包括 1 条指令 SYSCALL。

4.6.1 功能

指令	功能
SYSCALL	系统调用指令，操统执行此指令时，陷入内核态并调用相应异常处理代码

4.6.2 硬件需求

这一部分对硬件的需求主要集中于异常处理：此条指令会触发 SYSCALL 异常，因而硬件需要首先在流水线阶段识别该指令，然后依通用异常处理流程将异常原因等写入 CP0 相关寄存器中。

SYSCALL 指令对硬件的主要需求可以总结如下：

1. **流水线**：能对 SYSCALL 指令进行解码识别。能执行通用异常处理流程。
2. **CP0**：能提供寄存器记录异常信息。

4.6.3 异常

触发 SYSCALL 异常。

4.7 特权指令

特权指令共 5 条，包括对 CP0 的访问、TLB 的维护、异常的返回等功能，总结如下：

4.7.1 功能

指令	功能
MFC0、MTC0	读写协处理器 CP0
TLBWI、TLBWR	写 TLB 表项
ERET	异常返回：取消异常标志并返回受害指令地址

4.7.2 硬件需求

这一部分对硬件的需求较而杂。首先，这类指令之所以称为特权指令，是因为它们只能在内核态被调用，其功能多与异常处理、TLB 维护（多用于 TLB 异常处理）等相关。其次，无论是 TLB 的维护，还是异常的处理，都需要流水线、MMU、CP0 等多个硬件单元的协同配合。

这里仅简要总结这部分指令对硬件的需求：

1. **流水线**：能对各特权指令进行解码识别。能读写通用寄存器。能执行通用异常、TLB 异常处理流程。
2. **MMU**：能写入 TLB 表项。
3. **CP0**：能提供寄存器记录可能的异常信息。

4.7.3 异常

无。

4.8 总结

总体来说，指令系统对硬件的需求可以统一总结如下：

硬件单元	需求
流水线	**取指（IF）**：能每周期取得下一条指令的地址。 **译码（ID）**：能对不同的指令解码识别。能读写通用寄存器。 **执行（EX）**：能实现 ALU 提供的运算功能。能从 CP0、HI/LO 寄存器读出数据。 **访存（MEM）**：能通过 MMU 提供的接口对外部设备进行读写访问。 **回写（WB）**：能将数据写入通用寄存器或 CP0、HI/LO 寄存器。 各个部分能协同实现通用异常、TLB 异常处理流程。
寄存器堆	能提供 32 个通用寄存器用于实现指令中操作数的传递。
HI/LO 寄存器	能保存乘法运算的结果（64 位整数）。能提供接口支持读写操作。
MMU	能实现 ucore 所需的地址映射功能。能实现 TLB 的表项写入操作。
外设	能以黑盒方式使用外设，将其集成到 MMU 之下，提供读写接口。
CP0	能提供不同寄存器以记录异常信息的不同字段。能提供接口支持读写操作。

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