32位MIPS处理器实验需求文档

Cache小组

2015年1月9日

目录

1 引言

1.1 编写目的

计算机组成原理32位MIPS实验是在计原16位实验的基础上的扩展。在实验原理方面与多门课程相结合，涉及到操作系统、软件工程、编译原理等等多个方面，实验初期学习曲线较陡，在原理方面比较难以掌控。在编程实现方面涉及到大规模VHDL代码的书写，需要对数字逻辑设计有比较清晰的思路，工作量非常大。

因此，为控制整个开发过程，明确项目需求与目标，编写此需求文档。

文档预期读者为任务提出者：刘卫东老师、李山山老师、白晓颖老师。

1.2 背景

系统名称：32位MIPS处理器

任务提出者：计算机组成原理课程：刘卫东老师、李山山老师

软件工程课程：白晓颖老师

开发者：计23 李天润

计23 胡津铭

计23 孙皓

1.3 定义

以下是此次32位MIPS实验中需要用到的专业名词定义

表 1: 定义列表

名词	描述
MIPS	无内部互锁流水级的微处理器
CPU	中央处理器
CP0	协处理器0
MMU	内存管理单元
TLB	翻译后备缓冲
ALU	算术逻辑单元
RAM	存储程序的硬件，断电不保存信息
ROM	存储程序的硬件，断电可保存信息
Flash	存储程序的硬件，断电可保存信息，实验中用作硬盘
BIOS	主板上的启动程序，负责初始化硬件引导操作系统
BootLoader	一段特殊程序，将操作系统从Flash中加载到内存中，并且开始执行

1.4 参考资料

实验指导文档

OsLab实验参考文档

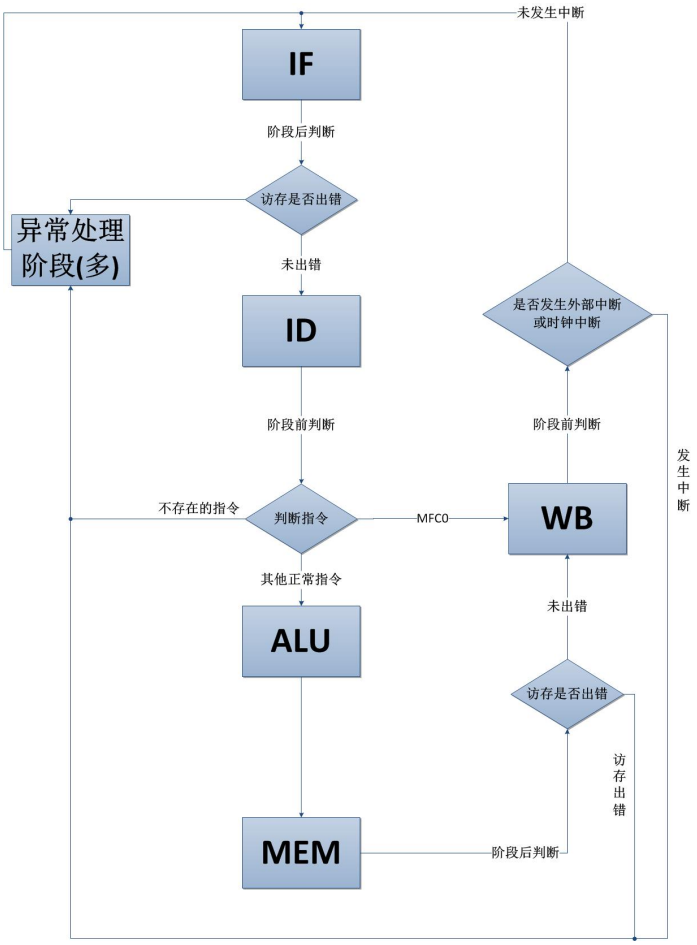
计算机组成原理综合实验报告 贾开
《计算机组成和设计 硬件/软件接口》
《See MIPS Run》

2 功能需求

2.1 CPU

CPU整体工作流程图

图 1: CPU各阶段转换图



2.1.1 ALU

功能需求:

1. 完成数据和地址的算术、逻辑和移位运算，输入两个数据根据ALUOp得出输出结果。

2. 根据指令系统的需求，完成指令系统中全部的算术指令（包括乘法相关指令）。

实现方式：

1. ALU有两个输入数据，通过ALUOp生成不同结果。ALUOp目前整理出17种运算，详细定义见数据通路表。
2. 只输出最终运算结果。不需要标志位。

2.1.2 乘法器

功能需求：

1. 完成乘法功能，结果保存在LO和HI寄存器中，可以通过MT和MF类指令访问和修改计算结果。

实现方式：

1. 乘法直接使用VHDL语言中提供的乘运算符实现。
2. 直接与主ALU相结合，在ALUOp中加入乘法的操作码，实现乘法的计算。
3. 乘法指令需要的计算时间较长，但是只要时钟频率低于50MHz，都可以在一个时钟周期内完成运算。因此，在预定的时钟频率下，乘法运算对时钟周期数的需求与其他运算相同，暂不考虑为乘法指令设计多个时钟周期的问题。
4. MFLO，MFHI，MTLO，MTHI，MULT指令查看与修改。 在ALUOp中也加入支持MTHI、MTLO的操作码，实现对LO和HI的修改。 同时LO和HI始终处于可读的状态，连接至写回阶段，MFLO与MFHI通过对写回阶段RegValue的控制来实现。

2.1.3 CP0

功能需求：
辅助操作系统对硬件进行管理。

1. 内存管理：辅助操作系统，完成虚拟地址和物理地址的转换、不同进程之间的内存切换、用户态与内核态内存的分离
2. 异常处理：检测指令执行过程中可能出现的异常、对异常进行分类、记录异常出现的指令地址或错误的内存地址
3. 外部中断：负责检测外部中断，辅助实现CPU与外设之间的交互

实现方式：

1. CP0记录CPU当前状态，通过MFC0、MTC0指令等进行读写操作，与通用寄存器建立联系

表 2: 需要实现的CP0寄存器

寄存器编号	名称	功能
0	Index	用于TLBWI指令访问TLB入口的索引序号
2	EntryLo0	作为TLBWI及其他TLB指令接口,管理偶数页入口
3	EntryLo1	作为TLBWI及其他TLB指令接口,管理奇数页入口
8	BadVAddr	捕捉最近一次地址错误或TLB异常(重填、失效、修改)时的虚拟地址
9	Count	每隔一个时钟周期增加1,用作计时器,并可使能控制
10	EntryHi	TLB异常时,系统将虚拟地址部分写入EntryHi寄存器中用于TLB匹配
11	Compare	当Count值与Compare相等时,SI_TimerInt 引脚是变高电平直到有数值写入Compare，用于定时中断
12	Status	表示协理器的操作模式、中断使能及诊断状态
13	Cause	记录最近依次异常的原因,控制软件中断请求以及中断协理派分的向量
14	EPC	存储异常处理之后程序恢复执行的地址
15	EBase	识别多协理器系统中不同的协理器异常向量的基地址

2. 在异常状态发生时保存异常的返回地址，异常类型等信息

3. 在内存映射中对TLB表项进行支持，为TLB充填等功能提供硬件支持

4. CP0寄存器的写方式有两种：

   (a) 正常执行状态下，通过MTC0指令，将CP0寄存器编号，CP0写入值，CP0控制信号传递给CP0寄存器，进行写入操作。

   (b) 异常状态下，通过异常处理模块到CP0的连接，直接将异常原因、异常指令地址、访存错误地址等信息，写入cause、EPC、BadVAddr寄存器中

5. CP0寄存器的读方式有两种：

   (a) 正常执行状态下，通过MFC0指令，将CP0寄存器编号传递给CP0寄存器，之后读取的值通过写回阶段写到通用寄存器中

   (b) 针对TLBWI等指令对CP0寄存器的读需求（需要同时读取5个CP0寄存器），将CP0寄存器与TLB相关的直接连接到MMU单元，通过TLBWI的使能信号控制写入TLB

CP0具体实现方式参照下面两个小节。
此次实验中需要实现的CP0寄存器见表2

2.1.4 MMU与TLB

功能需求：

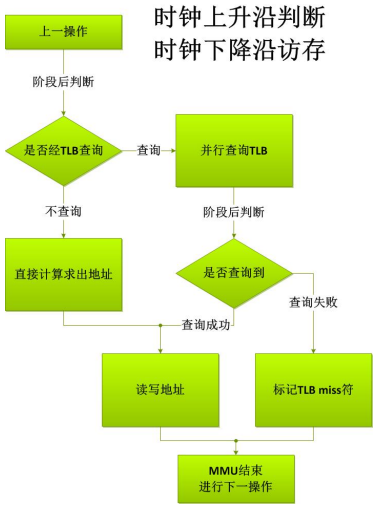
1. 实现虚拟地址（线性地址）到物理地址的转换。只对部分地址（[0x20000000~0x80000000]和[0xC0000000~0xFFFFFFFF]）进行地址映射，其余部分直接映射。
2. 实现内核态与用户态内存的区分。
3. 实现相对应的异常处理：TLBS、TLBL、TLB Modified。

实现方式：

图 2: TLB电路结构

	VPN2	PFN1	V1	D1	PFN2	V2	D2
范围	62-44	43-24	23	22	21-2	1	0

图 3: 内存访问流程图



1. (a) TLB共16项，每项可以将连续两个虚页映射到两个不同的物理页，相当于可以处理32个页
   (b) 虚拟地址高19位作为虚页号VPN进行选择TLB表项，第20位通过奇偶判断，选择两个物理页之一
   (c) 出于简化的考虑，不维护进程号ASID和全局标记G，默认所有G均为1，只维护每个物理地址的D和V标记
2. 硬件支持：
   (a) 虚拟地址高19位在TLB表项中并行查询EntryHi部分。

(b) 如果查找到，根据第20位选择EntryLo，直接与低12位结合得到真实的物理地址。

(c) 如果未查找到，触发TLBMiss异常：设置Cause寄存器中的ExcCode为TLB异常，EPC寄存器为当前指令的地址，BadvAddr寄存器为错误的地址，之后触发一个异常，PC跳转到EBase的异常处理基地址，之后由操作系统接管。

(d) 对于地址不对齐异常，在内存映射阶段对虚拟地址offset部分最低两位进行检测如果不为00，则触发地址不对齐异常，交由操作系统处理

3. 操作系统：

(a) 进入异常处理向量0x80000180。

(b) 跳转到处理函数部分（trap/trapentry.S/__alltraps）。

(c) 先保存异常现场，之后进入trap函数（trap/trap.c）。

(d) 根据是否发生嵌套异常调整trapframe，之后进入trap_dispatch函数（trap/trap.c）。

(e) 根据cause寄存器的值进行分类，调用tlb_miss_handler函数。

(f) 得到异常地址对应的物理页号，之后进行tlb_write（mm/tlb.c）。

(g) 操作系统维护index，写EntryLo0、EntryLo1、EntryHi、 PageMask、Index五个寄存器，之后用tlbwi写到TLB的第Index项。此处需要硬件提供MTC0、MFC0、TLBWI指令的支持。

(h) 异常返回，重新执行取地址命令。

2.1.5 异常中断处理

功能需求：

1. 实现对异常和外部中断的处理。

2. 实现的异常有内存访问异常、地址不对齐异常、系统调用、未定义的指令异常、未定义的寄存器异常。

3. 中断：串口中断、时钟中断。

实现方式：

1. 硬件支持

(a) 异常处理，在数据通路上添加异常处理部分，在多周期的任意一个周期检测到异常后，在Cause设置异常代码、在EPC寄存器设置异常处理的返回地址，将Status寄存器的EXL为设置为1，表示进入内核态处理异常之后根据EBase跳转到异常处理向量基地址，之后操作系统接管。

图 4: 异常中断处理流程

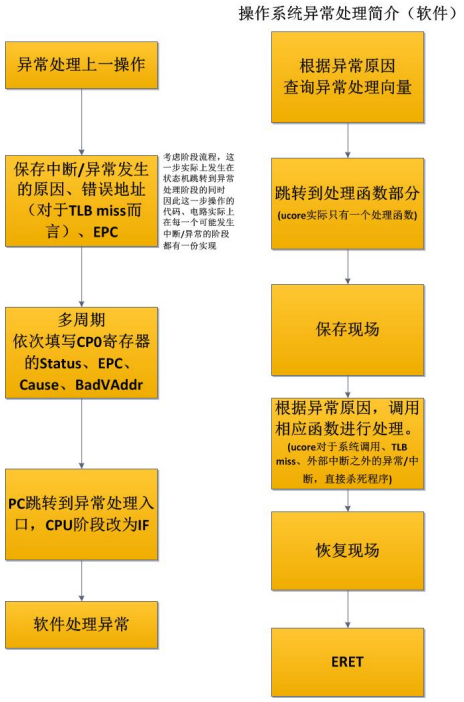

操作系统异常处理简介（软件）
异常处理上一操作
保存中断/异常发生的原因、错误地址（对于TLB miss而言）、EPC
考虑阶段流程，这一步实际上发生在状态机跳转到异常处理阶段的同时因此这一步操作的代码、电路实际上在每一个可能发生中断/异常的阶段都有一份实现
多周期
依次填写CP0寄存器的Status、EPC、Cause、BadVAddr
PC跳转到异常处理入口，CPU阶段改为IF
软件处理异常
根据异常原因
查询异常处理向量
跳转到处理函数部分
(ucore实际只有一个处理函数)
保存现场
根据异常原因，调用相应函数进行处理。
(ucore对于系统调用、TLB miss、外部中断之外的异常/中断，直接杀死程序)
恢复现场
ERET

表 3: 需要实现的异常

异常号	异常名	描述
0	Interrupt	外部中断,异步发生,由硬件引起
1	TLB Modified	内存修改异常,发生在Memory阶段
2	TLBL	读未在TLB中映射的内存地址触发的异常
3	TLBS	写未在TLB中映射的内存地址触发的异常
4	ADEL	读访问一个非对齐地址触发的异常
5	ADES	写访问一个非对齐地址触发的异常
8	SYSCALL	系统调用
10	RI	执行未定义指令异常

(b) 外部中断处理，设置外部中断检查信号，外部设备触发时，异步地将检查信号置1；每个指令IF阶段检查该信号，若为1则进入外部中断处理，否则正常执行指令。

(c) 时钟中断处理。当寄存器Count和Compare寄存器中的数值相等时，触发一个时钟异常，交由操作系统处理。

2. 操作系统

(a) 进入异常处理向量0x80000180。

(b) 跳转到处理函数部分（trap/trapentry.S/__alltraps）。

(c) 先保存异常现场，之后进入trap函数（trap/trap.c）。

(d) 根据是否发生嵌套异常调整trapframe，之后进入trap_dispatch函数（trap/trap.c）。

(e) 根据cause寄存器的值进行分类，调用cons_getc、syscall等等函数进行处理，如果是地址不对齐异常，或者未定义的异常，直接退出。如果是未定义的指令异常，可能是除法指令所导致的，通过ri_handler函数尝试进行模拟除法，如果仍然不能得到正确结果则退出程序。

(f) 使用ERET指令从异常返回，将EXL位重新设置为0，表示进入用户态重新执行取地址命令

2.1.6 其他功能部件

2.1.6.1 PC

功能需求：

1. 实现PC的多种变换方式：正常执行、分支、跳转。

2. 实现异常处理时对PC的处理，跳转到异常处理向量部分。

实现方式

1. 在写回模块对PC进行操作，将立即数或寄存器的值与当前PC进行运算实现PC的多种变化方式。决定是否跳转的比较操作也在这一模块完成。
2. 针对异常状态、从异常返回两种情况的PC变化，从CP0寄存器中引出EPC与EBase两个地址到PC计算单元。 通过当前的状态进行判断PC的取值。
3. 下一条指令的PC在本条指令的写回阶段得到确定， 保证在下一条指令的取指令时钟上升沿开始之前，正确送到MMU进行取指令。

2.1.6.2 寄存器堆

功能需求：

1. 实现通用寄存器，以及在数据通路中的读写控制。

实现方式

1. 在CPU中实现寄存器堆并实现读写控制。
2. 在解码指令阶段，根据指令的rs、rt、rd位置读取通用寄存器。 与写回模块相配合，完成寄存器堆的写入

2.1.6.3 串口

功能需求：

1. 实现与PC机通信，通过计算机键盘输入数据，向计算机输出数据。

实现方式

1. 使用开源的ASync transmitter and receiver，通过在每个信号周期内多次采样，并滤波得到稳定的信号。CPLD仅负责TX/RX端口数据的转发，对数据的编码与解码在FPGA中进行。这样设计提高了数据传送的准确率，并且串口数据不需要占用内存数据线，无需处理冲突，简化了设计。数据通信协议在开发过程中设计，有待进一步说明。

2.1.7 扩展部分

在开发过程中测试是必不可少的环节，因此，此次实验的扩展部分计划实现一个类似gdb的工具，并给出一份较为详细的测试方案。

调试工具在电脑端提供与gdb类似的终端，支持gdb中的print、break等基本命令。在硬件上，通过串口进行指令的输入与数据的输出，尽可能减少对原有CPU设计的修改。

在每个模块正常工作之后，将各个模块进行组合，形成整体的CPU设计。通过操作系统的lab循序渐进地对CPU进行测试，此时可以使用开发的类gdb工具，设置断点、输出信号进行调试。

2.1.8 指令集与数据通路

指令集为标准MIPS系统子集

实现多周期CPU，针对指令集设计数据通路。仿照《软件硬件接口》书中的多周期CPU进行设计，在书中7条指令的原型上进行扩展，以支持标准MIPS系统的子集。

指令执行过程主要分为取指、解码、执行、访存、写回五个周期，某些指令可能需要其他特殊的周期支持，采用多周期实现比较灵活。

数据通路包括状态机与控制线设计，每条指令有不同的状态机变化，而且对指令进行解码，能够得到相应的控制线。在多周期的不同周期利用控制线对数据通路进行控制，使指令正常执行。

数据通路图在设计文档中有详细体现。

2.2 BIOS

准备阶段：

1. 操作系统和文件系统烧写到Flash中
2. bootloader编译出的二进制文件生成ROM
3. CPU代码烧写到FPGA中。

启动阶段：

1. 从ROM中的bootasm.S启动
2. 将Flash中的全部操作系统拷贝到RAM中，然后控制权交给操作系统。

操作系统启动阶段：

1. bootasm.S工作结束后，进入操作系统entry.S文件中进行初始化。
2. entry.S：栈寄存器初始化，之后跳转到init.c中的kern_init。
3. kern_init：完成异常中断初始化，TLB初始化（全部TLB表项清零），物理内存与虚拟内存的转换页表的初始化（创建初始页表）
4. 最后完成操作系统的进程和文件管理系统，至此ucore系统开始正常工作，

3 性能需求

实现多周期CPU，性能上可能与流水线存在一定差距，尽量通过良好的内部设计弥补，减小各个模块之间的延时。

参照贾开学长12.5MHz的时钟主频，将我们的主频目标暂定为6.25MHz，主频可能根据具体实现过程进行调整。

4 运行环境需求

4.1 设备

表 4: 设备与外部接口

环境	描述
FPGA	Xilinx Spartan6 xc6slx100
RAM	32-bit字长, 4块, 共8MB
Flash	16-bit 字长, 共8MB
CPLD	与FPGA相连, 用于I/O
串口	2个
USB 串口	1个
ps/2 接口	1个
以太网接口	1个
VGA 接口	1个

4.2 控制

控制部分采用串口作为输入设备，如果有需要可以启用实验板上的第二个串口。

5 附录

5.1 指令系统

指令系统参考了MIPS Instructions Reference。其中Encoding部分标记为-的表示可以为任意01组合，但实际编程实现中应当做0处理。

Syntax:	addiu $d, $s, $t
Description:	Adds a register and a sign-extended immediate value and stores the result in a register
Operation:	$t=$s + imm;
Encoding:	0010 00ss ssst tttt iiii iiii iiii iiii

Syntax:	addu $d, $s, $t
Description:	Adds two registers and stores the result in a register
Operation:	$d = $s + $t;
Encoding:	0000 00ss ssst tttt dddd d000 0010 0001

Syntax:	slt $d, $s, $t
Description:	If $s is less than $t, $d is set to one. It gets zero otherwise.
Operation:	if $s < $t $d = 1 else $d = 0;
Encoding:	0000 00ss ssst tttt dddd d000 0010 1010

Syntax:	slti $t, $s, imm
Description:	If $s is less than immediate, $t is set to one. It gets zero otherwise.
Operation:	if $s < imm $t = 1 else $t = 0
Encoding:	0010 10ss ssst tttt iiii iiii iiii iiii

Syntax:	sltiu $t, $s, imm
Description:	If $s is less than the unsigned immediate, $t is set to one. It gets zero otherwise.
Operation:	if $s < imm $t = 1 else $t = 0;
Encoding:	0010 11ss ssst tttt iiii iiii iiii iiii

Syntax:	sltu $d, $s, $t
Description:	If $s is less than $t, $d is set to one. It gets zero otherwise.
Operation:	if $s < $t $d = 1 else $d = 0;
Encoding:	0000 00ss ssst tttt dddd d000 0010 1011

Syntax:	subu $d, $s, $t
Description:	Subtracts two registers and stores the result in a register
Operation:	$d = $s - $t;
Encoding:	0000 00ss ssst tttt dddd d000 0010 0011

Syntax:	mult $s, $t
Description:	Multiplies $s by $t and stores the result in $HI\|$LO.
Operation:	$HI\|$LO = $s * $t;
Encoding:	0000 00ss ssst tttt 0000 0000 0001 1000

Syntax:	mflo $d
Description:	The contents of register LO are moved to the specified register.
Operation:	$d = $LO;
Encoding:	0000 0000 0000 0000 dddd d000 0001 0010

Syntax:	mfhi $d
Description:	The contents of register HI are moved to the specified register.
Operation:	$d = $HI;
Encoding:	0000 0000 0000 0000 dddd d000 0001 0000

Syntax:	mtlo $s
Description:	The contents of register LO are replaced by the specified register.
Operation:	$LO = $s;
Encoding:	0000 00ss sss0 0000 0000 0000 0001 0011

Syntax:	mthi $s
Description:	The contents of register HI are replaced by the specified register.
Operation:	$HI = $s;
Encoding:	0000 00ss sss0 0000 0000 0000 0001 0001

Syntax:	and $d, $s, $t
Description:	Bitwise ands two registers and stores the result in a register.
Operation:	$d = $s & $t;
Encoding:	0000 00ss ssst tttt dddd d000 0010 0100

Syntax:	andi $t, $s, imm
Description:	Bitwise ands a register and an immediate value and stores the result in a register.
Operation:	$t = $s & imm;
Encoding:	0011 00ss ssst tttt iiii iiii iiii iiii

Syntax:	lui $t, imm
Description:	The immediate value is shifted left 16 bits and stored in the register. The lower 16 bits are zeroes.
Operation:	$t = (imm << 16);
Encoding:	0011 11-- ---t tttt iiii iiii iiii iiii

Syntax:	nor $d, $s, $t
Description:	Bitwise logical ors two registers and stores the logical not result in a register.
Operation:	$d = $\sim$ ($s | $t);
Encoding:	0000 00ss ssst tttt dddd d000 0010 0111

Syntax:	or $d, $s, $t
Description:	Bitwise logical ors two registers and stores the result in a register.
Operation:	$d = $s | $t;
Encoding:	0000 00ss ssst tttt dddd d000 0010 0101

Syntax:	ori $t, $s, imm
Description:	Bitwise ors a register and an immediate value and stores the result in a register.
Operation:	$t = $s | imm;
Encoding:	0011 01ss ssst tttt iiii iiii iiii iiii

Syntax:	xor $d, $s, $t
Description:	Exclusive ors two registers and stores the result in a register.
Operation:	$d = $s $\oplus$ $t;
Encoding:	0000 00ss ssst tttt dddd d--- --10 0110

Syntax:	xori $t, $s, imm
Description:	Bitwise exclusive ors a register and an immediate value and stores the result in a register.
Operation:	$t = $s $\oplus$ imm;
Encoding:	0011 10ss ssst tttt iiii iiii iiii iiii

Syntax:	sll $d, $t, h
Description:	Shifts a register value left by the shift amount listed in the instruction and places the result in a third register. Zeroes are shifted in.
Operation:	$d = $t << h
Encoding:	0000 00ss ssst tttt dddd dhhh hh00 0000

Syntax:	sllv $d, $t, $s
Description:	Shifts a register value left by the value in a second register and places the result in a third register. Zeroes are shifted in.
Operation:	$d = $t << $s;
Encoding:	0000 00ss ssst tttt dddd d--- --00 0100

Syntax:	sra $d, $t, h
Description:	Shifts a register value right by the shift amount (shamt) and places the value in the destination register. The sign bit is shifted in.
Operation:	$d = $t >> h;
Encoding:	0000 00-- ---t tttt dddd dhhh hh00 0011

Syntax:	srav $d, $t, $s
Description:	Shifts a register value right by the value in a second register and places the value in the destination register. The sign bit is shifted in.
Operation:	$d = $t >> $s;
Encoding:	0000 00ss ssst tttt dddd d000 0000 0111

Syntax:	srl $d, $t, h
Description:	Shifts a register value right by the shift amount (shamt) and places the value in the destination register. Zeroes are shifted in.
Operation:	$d = $t >> h;
Encoding:	0000 00-- ---t tttt dddd dhhh hh00 0010

Syntax:	srlv $d, $t, $s
Description:	Shifts a register value right by the amount specified in $s and places the value in the destination register. Zeroes are shifted in.
Operation:	$d = $t >> $s;
Encoding:	0000 00ss ssst tttt dddd d000 0000 0110

Syntax:	beq $s, $t, offset
Description:	Branches if the two registers are equal.
Operation:	if $s == $t advance_pc (offset << 2); else advance_pc (4);
Encoding:	0001 00ss ssst tttt iiii iiii iiii iiii

Syntax:	bgez $s, offset
Description:	Branches if the register is greater than or equal to zero.
Operation:	if $s >= 0 advance_pc (offset << 2)); else advance_pc (4);
Encoding:	0000 01ss sss0 0001 iiii iiii iiii iiii

Syntax:	bgtz $s, offset
Description:	Branches if the register is greater than zero.
Operation:	if $s > 0 advance_pc (offset << 2)); else advance_pc (4);
Encoding:	0001 11ss sss0 0000 iiii iiii iiii iiii

Syntax:	blez $s, offset
Description:	Branches if the register is less than or equal to zero.
Operation:	if $s <= 0 advance_pc (offset << 2)); else advance_pc (4);
Encoding:	0001 10ss sss0 0000 iiii iiii iiii iiii

Syntax:	bltz $s, offset
Description:	Branches if the register is less than zero.
Operation:	if $s < 0 advance_pc (offset << 2)); else advance_pc (4);
Encoding:	0000 01ss sss0 0000 iiii iiii iiii iiii

Syntax:	bne $s, $t, offset
Description:	Branches if the two registers are not equal.
Operation:	if $s != $t advance_pc (offset << 2)); else advance_pc (4);
Encoding:	0001 01ss ssst tttt iiii iiii iiii iiii

Syntax:	j target
Description:	Jumps to the calculated address.
Operation:	PC = (PC & 0xf0000000) | (target << 2)
Encoding:	0000 10ii iiii iiii iiii iiii iiii iiii

Syntax:	jal target
Description:	Jumps to the calculated address and stores the return address in $RA
Operation:	$RA = RPC; PC = (PC & 0xf0000000) | (target << 2);
Encoding:	0000 11ii iiii iiii iiii iiii iiii iiii

Syntax:	jalr $s, $d
Description:	Jumps to the address of the first register, and stores the return address in the second register
Operation:	$d = RPC; PC = $s
Encoding:	0000 11ii iiii iiii iiii iiii iiii iiii

Syntax:	jr $s
Description:	Jump to the address contained in register $s.
Operation:	PC = $s;
Encoding:	0000 00ss sss0 0000 0000 0000 0000 1000

Syntax:	lw $t, offset($s)
Description:	A word is loaded into a register from the specified address.
Operation:	$t = MEM[$s + offset];
Encoding:	1000 11ss ssst tttt iiii iiii iiii iiii

Syntax:	sw $t, offset($s)
Description:	The contents of $t is stored at the specified address.
Operation:	MEM[$s + offset] = $t;
Encoding:	1010 11ss ssst tttt iiii iiii iiii iiii

Syntax:	lb $t, offset($s)
Description:	A byte is loaded into a register from the specified address.
Operation:	$t = MEMByte[$s + offset];
Encoding:	1000 00ss ssst tttt iiii iiii iiii iiii

Syntax:	lbu $t, offset($s)
Description:	A byte is loaded into a register from the specified address with zero extend.
Operation:	$t = zero-extend(MEMByte[$s + offset]);
Encoding:	1001 00ss ssst tttt iiii iiii iiii iiii

Syntax:	sb $t, offset($s)
Description:	The least significant byte of $t is stored at the specified address.
Operation:	MEM[$s + offset] = (0xff & $t);
Encoding:	1010 00ss ssst tttt iiii iiii iiii iiii

Syntax:	syscall
Description:	Generates a software interrupt.
Operation:	nothing
Encoding:	0000 00-- ---- ---- ---- ---- --00 1100

Syntax:	cache
Description:	Nothing, alias to NOP
Operation:	nothing
Encoding:	1011 11ss ssst tttt iiii iiii iiii iiii

Syntax:	eret
Description:	Return from trap
Operation:	PC = EPC
Encoding:	0100 0010 0000 0000 0000 0000 0001 1000

Syntax:	mfc0 $t, $d
Description:	Move the value of a CP0 register to a general register.
Operation:	$t = CP0[$d]
Encoding:	0100 0000 000t tttt dddd d000 0000 0000

Syntax:	mtc0 $d, $t
Description:	Move the value of a general register to a CP0 register.
Operation:	CP0[$d] = $t
Encoding:	0100 0000 100t tttt dddd d000 0000 0000

Syntax:	tlbwi
Description:	Write a TLB entry using Index, EntryHi, EntryLo0, EntryLo1 from CP0 register.
Operation:	Write a TLB entry.
Encoding:	0100 0010 0000 0000 0000 0000 0000 0010

Syntax:	lhu $t, $s, imm
Description:	Load half word from memory to general register.
Operation:	$t = zero-extend(MEMHalfWord[$s + offset]);
Encoding:	1001 01ss ssst tttt iiii iiii iiii iiii