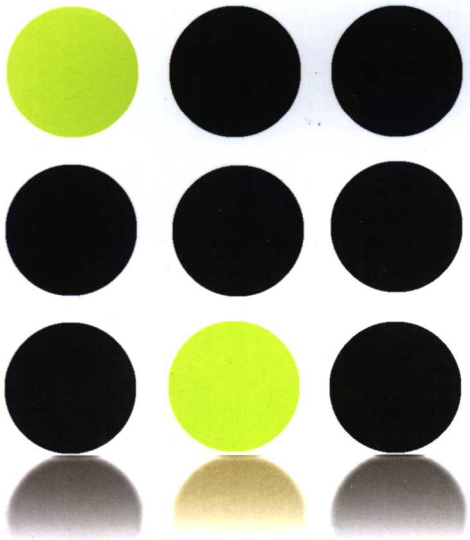
高等院校计算机专业课程综合实验系列规划教材



丛书主编　何钦铭　陈根才

楼学庆　平玲娣　著

王志英　主　审

计算机组成课程设计

ZHEJIANG UNIVERSITY PRESS
浙江大学出版社

浙江大学魏绍相计算机教材建设基金资助
高等院校计算机专业课程综合实验系列规划教材

计算机组成课程设计

楼学庆　平玲娣　著
王志英　主审

浙江大學出版社

内容提要

本书的特色是要求实验者用硬件描述语言(Verilog HDL)设计单元电路、功能部件和指令条数有限而功能较齐全的单时与多时钟CPU;采用先进的电子设计自动化工具(EDA)仿真模拟用Verilog语言设计的计算机功能单元,仿真模拟正确后,能把设计结果写入Spartan-3 Starter Kit Board开发板上的FPGA可编程芯片上,并能直接实验自己的设计(芯片)是否达到预期目标,其乐无穷。实验设计按照主干课程大纲由浅入深,内容丰富,每个实验目标明确。

本书含光盘,光盘中内容为Spartan-3软件系统。

图书在版编目(CIP)数据

计算机组成课程设计/楼学庆,平玲娣著. —杭州:浙江大学出版社,2007.11
(高等院校计算机专业课程综合实验系列规划教材)
ISBN 978-7-308-05402-7

Ⅰ.计… Ⅱ.①楼…②平… Ⅲ.计算机体系结构-高等学校-教材 Ⅳ.TP303

中国版本图书馆CIP数据核字(2007)第094982号

计算机组成课程设计

楼学庆 平玲娣 著

王志英 主审

丛书主编 何钦铭 陈根才
策 划 黄娟琴 希 言
责任编辑 邹小宁 冯 骏
封面设计 氧化光阴
出版发行 浙江大学出版社
(杭州天目山路148号 邮政编码310028)
(E-mail:jsjsyb@zju.edu.cn)
(网址:http://www.zjuxsj.com)
排 版 浙江大学出版社电脑排版中心
印 刷 德清县第二印刷厂
开 本 787mm×1092mm 1/16
印 张 16
字 数 233千
版 印 次 2007年11月第1版 2007年11月第1次印刷
印 数 0001—3000
书 号 ISBN 978-7-308-05402-7
定 价 28.00元(含光盘)

专家指导委员会

序　言

近10多年来，以计算机和通信技术为代表的信息技术迅猛发展，并已深入渗透到国民经济与社会发展的各个领域。信息技术成为国家产业结构调整和推动国民经济与社会快速发展的最重要的支撑技术。与此同时，深入掌握计算机专业知识、具有良好系统设计与分析能力的计算机高级专业人才在社会上深受欢迎。

计算机科学与技术是一门实践性很强的学科。良好的系统设计和分析能力的培养需要通过长期、系统的训练（包括理论和实践两方面）才能获得。高等学校的实践教学一般包括课程实验、综合性设计（课程设计）、课外科技活动、社会实践、毕业设计等，基本上可以分为三个层次：第一，是紧扣课堂教学内容，以掌握和巩固课程教学内容为主的课程实验和综合性设计；第二，是以社会体验和科学研究体验为主的社会实践和课外科技活动；第三，是以综合应用专业知识和全面检验专业知识应用能力的毕业设计。课程实践（含课程实验和课程设计）是大学教育中最重要也最基础的实践环节，直接影响后继课程的学习以及后继实践的质量。由于课程设计是以培养学生的系统设计与分析能力为目标，通过团队式合作、研究式分析、工程化设计完成较大型系统或软件的设计题目的，因此课程设计不仅有利于学生巩固、提高和融合所学的专业课程知识，更重要的是能够培养学生多方面的能力，如综合设计能力、动手能力、文献检索能力、团队合作能力、工程化能力、研究性学习能力、创新能力等。

浙江大学计算机学院在专业课程中实施课程设计（project）已有10多年的历史，积累了丰富的经验和资料。为全面总结专业课程设计建设的经验，推广建设成果，我们特别组织相关课程的骨干任课教师编写了这套综合实验系列教材。本系列教材的作者们不仅具有丰富的教学和科研经验，而且是浙江大学计算机学院和软件学院的教学核心力量。这支队伍目前已经获得了两门国家精品课程以及四门省部级精品课程，出版了几十部教材。

本套教材由《C程序设计基础课程设计》、《软件工程课程设计》、《数据结构课程设计》、《数值分析课程设计》、《编译原理课程设计》、《逻辑与计算机设计基础实验与课程设计》、《操作系统课程设计》、《数据库课程设计》、《Java程序设计课程设计》、《面向对象程序设计课程设计》、《计算机组成课程设计》、《计算机体系结构课程设计》和《计算机图形学课程设计》等十三门课程的综合实验教材所组成。该系列教材构思新颖、案例丰富，许多案例直接取材于作者多年教学、科研以及企业工程经验的积累，适用于作为计算机以及相关专业课程设计的实验教材；也适用于对计算机有浓厚兴趣的专业人士进一步提升计算机的系统设计

与分析能力。从实践的角度出发，大部分教材配备了随书光盘，以方便读者练习。

可以说，本套教材涵盖了计算机专业绝大部分必修课程和部分选修课程，是一套比较完整的专业课程设计系列教材，也是国内第一套由研究型大学计算机学院独立组织编写的专业课程设计系列教材。鉴于书中难免存在的谬误之处，敬请读者指正，以便不断完善。

主编 何钦铭、陈根才

2007 年 6 月于求是园

前　言

《计算机组成与设计》是计算机各专业本科生的一门必修的核心课程，是关于计算机系统结构方面的主干课程；也是学习后继课程，包括体系结构、操作系统、编译原理等课程的基础技术课程。

本书是《计算机组成与设计》课程的教学实验指导书，目的是指导学生，用所学到的计算机组成原理与计算机系统先进的设计方法，自己解决实验项目提出的问题，获得所见所得的感性认识，从而进一步提高理性认识。此外，实验者通过完成本书提出的课程设计，能系统地掌握当代设计功能单元、数字系统、CPU 复杂系统的先进方、设计调试和验证流程，熟练掌握 Verilog 语言、EDA 工具的使用方法、设计环境的配置建立过程和 FPGA 开发板的操作使用；能学到书本教学无法学到的新技术和新知识，并提练自己的新见解。

本书紧密结合课堂教学进程，着重实验项目设计，力求由浅入深地编排课程实验。实验主要是在逻辑与计算机所设计的简单功能组件基础上，基于 Verilog HDL，实现一些复杂组件、控制单元等的设计，并最终完成一个能执行有限指令的多时钟处理器，写入 FPGA，构成一个可以加电运行的芯片。

本书包括十二个实验和 A、B 两部分附录，具体内容如下。

第 1 章：将 MIPS 汇编转换成机器码，并模拟执行。目的是了解汇编语言的转换过程，实验中产生的机器代码在以后的设计的 CPU 中，写入存储器模拟执行。

第 2 章：熟悉 Xilinx EDA 软件，在《逻辑与计算机设计》实验的基础上，对软件的使用作准备。

第 3 章与第 4 章：用 EDA 软件先设计单个组件，再从组件设计开始，继续熟悉掌握 Xilinx 软件的各种功能，所设计的组件作为主体设计的模块。

第 5 章：将模块组成能够执行单个指令功能的 CPU（单个功能）便于通过数码显示等检验运行状态。

第 6 章与第 7 章：设计单时钟 CPU，通过控制器将多条指令集合组成单时钟 CPU，并了解 Xilinx 软件中对存储器的使用，写入简单程序模拟执行。

第 8 章：用有限状态机控制方式设计多时钟数据通路。

第 9 章与第 10 章：利用微程序方式的控制单元设计多时钟数据通道。

第 11 章与第 12 章：这两个实验为开放型拓展实验，要求根据用户对 CPU 的要求设计。设计是在多时钟 CPU 的基础上，增加指令，实现更大指令集的有限指令 CPU 设计，并编写简单的操作系统，完成程序的加载及一些简单的系统功能。如果能够实现的指令集足够完整，可以下载一些微核系统程序作为操作系统。

附录 A：本书的实验是以 MIPS 汇编语言作为基础的，附录 A 对 MIPS 汇编语言作了简

单介绍。

附录 B:介绍计算机中各种基本数据的表示,并希望通过程序演示,加深对各种数据表示的理解,加强从计算机的角度思考问题的能力。

CPU 设计是理论与实践的结合,在每次实验之前,一定要先做好逻辑设计工作,包括各种模块的功能与需要的逻辑组件。在设计有限指令集时,可以根据自己所设计的系统需要,考虑增加 MIPS 以外的一些指令。

本书主要由楼学庆编写,平玲娣编写了部分章节,并审阅了全书。

浙江大学计算机学院的部分教师参与了本书的资料收集等工作,在此一并表示感谢。

由于编写仓促,不妥之处在所难免,希望读者提出宝贵意见。

作 者

2007 年 10 月

目　录

第1章 MIPS 汇编软件模拟实验

一、实验目的

1. 熟悉汇编语言，掌握 MIPS 汇编语言程序的汇编、反汇编方法。

2. 用 C、C ++ 或 Java 语言编写程序，实现 MIPS 语言到机器指令的汇编，与由机器指令到汇编语言的反汇编。

3. 用 C、C ++ 或 Java 语言编写 MIPS 软件模拟机。

4. 设计有限指令 CPU 的系统软件及应用程序加载运行模式。

二、实验设备

1. 常用微机一台。

2. C、C ++ 或 Java 编译环境。

3. Pcspim 或 xspim 软件一套。（选）

三、实验任务

MIPS 是精简指令集计算机（RISC）体系结构中典型的汇编语言。它采用等长指令结构，每条指令长 32 位、占 4 个字节，其中高 6 位为操作码。根据不同的用途，这些指令可分成三大类。

• R 类型（其结构如图 1.1 所示）：操作数主要在寄存器中。MIPS 汇编语言共用 32 个通用寄存器，每个均 32 位（bit），需 5bit 寻址。

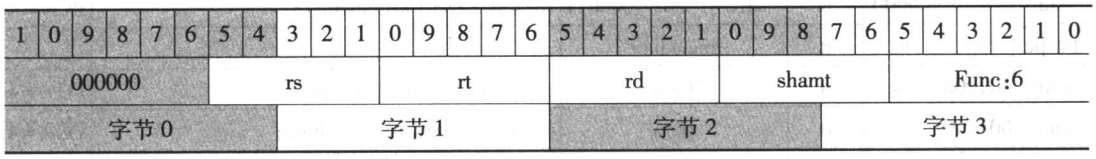


图 1.1　R 型指令结构

• I 类型（其结构如图 1.2 所示）：其中的一个操作数为直接编码在指令中的立即数。由于 MIPS 指令为等长的 32 位，指令中的立即数为遵循补码规则的 16 位有符号数。当立即数与寄存器中的内容做运算时，需按补码规则扩展到 32 位。为扩大寻址范围，当用立即数作指令寻址时，立即数表示的为指令数而非字节数。用立即数寻址时，需将立即数左移 2 位（立即数乘 4）。

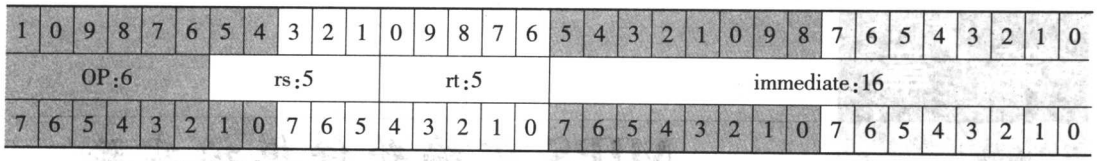


图 1.2 I 型指令结构

• J 类型(其结构如图 1.3 所示):除操作码以外的 16 位为一 26 位长的立即数。同 I 类型一样,用立即数寻找地址时,如果立即数为指令地址,需将立即数左移 2 位(立即数乘 4)。

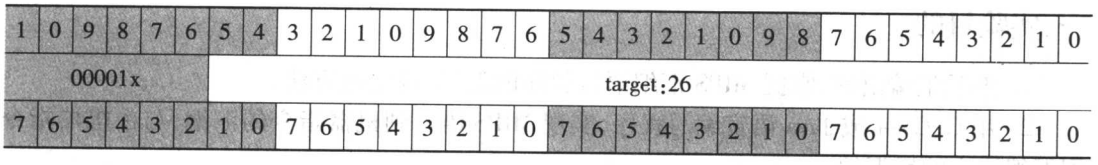


图 1.3 J 型指令结构

表 1-1 是一些常见 MIPS 汇编指令及其机器码格式与操作功能,其他指令可参考相关教材,网上也有许多这方面资料。

表 1-1 常见 MIPS 汇编指令

MIPS 指令								
Bit #	31..26	25..21	20..16	15..11	10..6	5..0	操作	
R-type	op	rs	rt	rd	Sa	func		
add	000000	rs	rt	rd	00000	100000	rd = rs + rt;	PC +=4
sub		rs	rt	rd	00000	100010	rd = rs − rt;	PC +=4
and		rs	rt	rd	00000	100100	rd = rs & rt;	PC +=4
or		rs	rt	rd	00000	100101	rd = rs | rt;	PC +=4
slt		rs	rt	rd	00000	101010	if(rs < rt) rd = 1; else rd = 0;	PC +=4
sll		00000	rt	rd	Sa	000000	rd = rt << sa;	PC +=4
srl		00000	rt	rd	Sa	000010	rd = rt >> sa (logical);	PC +=4
sra		00000	rt	rd	Sa	000011	Rd = rt >> sa (arithmetic);	PC +=4
I-type	op	rs	rt	immediate				
addi	001000	rs	rt	Imm			rt = rs + (sign_extend) imm;	PC +=4
andi	001100	rs	rt	Imm			rt = rs & (zero_extend) imm;	PC +=4
ori	001101	rs	rt	Imm			rt = rs | (zero_extend) imm;	PC +=4
lw	100011	rs	rt	Imm			rt = memory[rs + (sign_extend) imm];	PC +=4
sw	101011	rs	rt	Imm			memory[rs + (sign_extend) imm] <-- rt;	PC +=4
beq	000100	rs	rt	Imm			if (rs ==rt) PC +=4 +(sign_extend)imm <<2; else	PC +=4
bne	000101	rs	rt	Imm			if (rs! =rt) PC +=4 +(sign_extend)imm <<2; else	PC +=4
J-type	op	Address						
j	000010	Address					PC = (PC +4)[31..28], address <<2	

MIPS 有 32 个寄存器，均为 32 位，见表 1-2。其中 0 号寄存器$zero 中的值恒为 0；31 号寄存器$ra 存放着正常函数调用指令(jal)的返回地址。

与 INTEL 汇编 call 指令将子程序调用地址直接存放在堆栈中不同，MIPS 在第一级调用子程序时，将地址存放在寄存器$ra 中。如果子程序需再调子用 jal 程序，需先将$ra 存入堆栈。这样做可以提高第一次调用的速度。另外，MIPS 也可以用 call-by-register 的 jalr 指令使用任何寄存器来存放其返回地址。

对于其他寄存器，硬件没有强制性的指定寄存器使用规则，但在具体 MIPS 汇编中，均需按一定的约定使用。虽然这些约定与硬件无关，但如果想使用其他用户的代码、编译器和操作系统，就一定遵循这些约定。

寄存器约定用法引入了一系列的寄存器约定名。在使用寄存器的时候，要尽量用这些约定名或助记符，而不是直接引用寄存器编号。

MIPS 里没有状态码。CPU 状态寄存器或内部都不包含任何用户程序计算的结果状态信息。

表 1-2　MIPS 寄存器

寄存器名	寄存器号	个数	Usage(使用约定)
$zero	0	1	永远返回值为 0
$at	1	1	保留给汇编器，用做汇编器的暂时变量(如处理伪指令)
$v0 - $v1	2 - 3	2	子函数调用返回结果
$a0 - $a3	4 - 7	4	子函数调用的参数
$t0 - $t7	8 - 15	8	暂时变量，子函数使用时不需要保存与恢复
$s0 - $s7	16 - 23	8	寄存器变量。子函数若使用必须保存，并在返回前恢复原值，以保证在主函数中这些寄存器的值没有改变
$t8 - $t9	24 - 25	2	暂时变量，子函数使用时不需要保存与恢复
$k0 - $k1	26 - 27	2	保留给 OS，通常被中断或异常处理程序使用，系统参数
$gp	28	1	全局指针。一些运行系统维护这个指针来更方便的存取“static”和“extern”变量
$sp	29	1	堆栈指针
$s8/$fp	30	1	第 9 个寄存器变量。子函数可以用来做帧指针
$ra	31	1	子函数的返回地

本实验是先将一段 C 语言程序用手工编译(如果想自动编译，请参考《编译原理》)成 MIPS 汇编语言；然后编写 C 或 Java 语言程序，将 MIPS 汇编语言程序翻译为机器指令；或将机器指令反汇编为 MIPS 汇编程序，并用软件模拟 MIPS 机器执行程序。如下例所示。

C 语言程序	MIPS 汇编

int sum(int *p, int n)
{
int sum = p[0];
for(int i = 1; i < n; i ++)sum += p[i];
return sum;
}
	LW $T1, 0($A0)
	ADD $T0, $ZERO,$ZERO
	R0: ADDi $T0, $ZERO, 1
	SLT $T2, $T0, $A1
	BEQ $T2, $ZERO, RR
	ADD $T2, $T0, $A0
	LW $T2, 0($T2)
	ADD $T1, $T1, $T2
	J R0
	RR:ADD $V0, $T1, $ZERO
	JR $RA

对于转移指令,需假设程序加载地址。假设以上程序在存储器的 0 地址开始,则其实对应的机器码如下。

汇编程序			机器指令	
地址	MIPS 汇编		分字段	十六进制
00000000:	LW	$T1 , $A0 (0)	35,2,9,0	8C890000
00000004:	ADD	$T0 , $ZERO, $ZERO	0,0,0,8,32	00004020
00000008:	ADDI	$T0 , $ZERO, 1	R0: 8,0,8,1	20080001
0000000C:	SLT	$T2 , $T0 , $A1	0,8,3,10,842	0105502A
00000010:	BEQ	$T2 , $ZERO, 00000024	4,10,0,R1	11400004
00000014:	ADD	$T2 , $T0 , $A0	0,10,8,2,32	01045020
00000018:	LW	$T2 , $T2 (0)	35,10,10,0	8D4A0000
0000001C:	ADD	$T1 , $T1 , $T2	0,9,10,9,32	012A4820
00000020:	J	00000008	2,R0	08000002
00000024:	ADD	$V0 , $T1 , $ZERO	R1: 0,9,0,4,32	01201020
00000028:	JR	$RA	0,32,0,0,8	03E00008

MIPS 汇编:

做汇编时,为了简化程序,可以对汇编程序的书写格式进行限制,比如只能用小写,不能有多余空格等等。如果程序功能强,可以放宽格式限制,加强语法检查,增加错误信息提示等,以使汇编程序书写更加灵活;也可考虑增加对伪指令的处理。

考虑到标号的使用,汇编需作两次扫描。第一次预汇编时,由于不知道标号的具体数值,所有标号均填 0,同时建立标号地址表。第二次再查表将标号换上实际值。

如指令:BEQ $T2, $ZERO, RR

由于标号 RR 出现在本指令的后面,所以第一遍不能确定 BEQ 指令中立即数的值,可以先用 0 代替。第一次汇编为:0001 0001 0100 0000 *0000 0000 0000 0000* 。等汇编到指令 RR: ADD $V0, $T1, $ZERO 时,得知 RR 所在地址为 0x000024,将 RR 及当前指令地址(0x000024)存入标号表中。在第二次扫描时,对所有的标号用表中的值代替原先的 0。由

于当前指令地址为 10,BEQ 采用相对于 PC 转移的寻址方式,所以指令中的立即数为:

(0x0024 - (0x0010 + 4)) >> 2 = 0x0010/4 = 4

汇编指令最后的机器码为:

0001 0001 0100 0000 0000 0000 0000 0100 →11 40 00 04

在程序中增加符号变量的使用,对指令中的直接地址用符号形式表示,汇编时再转化为相应的立即数。

编写汇编程序的一般步骤如下。

(1)将指令的每个词按常用分隔符分开。

```
    typedef unsigned long    dword;
typedef struct {
    char    nam[10];
    int        lin;    //行号,指令序
} TABLE;            //标号表

int        num, lin;
char    wrd[10][12];    //每条指令的词表
TABLE    lab[100];

int strcut(char buf[])
{
    int    i = 0, j, m = 0;

    while(buf[i]){
        while(buf[i] == ' ' || buf[i] == 9)i ++;
        j = 0;
        while(buf[i]! = ' ' && buf[i]! = ','
           && buf[i]! = '(' && buf[i]! = ')'
           && buf[i]! = ':' && buf[i]! = ';'
           && buf[i]! = 10   && buf[i]! = 13
           && buf[i]! = 9    && buf[i]! = 0)
            wrd[m][j ++] = (buf[i] >= 'A' && buf[i] <= 'Z')
                        ? (buf[i ++] |0x20) : buf[i ++];
        if(j == 0)break;

        wrd[m][j] = 0;
        if(buf[i] == ':'){
            strcpy(lab[num].nam, wrd[m]);
            lab[num].lin = lin;
            num ++;
        } else m ++;
        i ++;
    }
```

```
    return m;
}
```

(2)将寄存器名转换为0~31的二进制寄存器号。

```
long regN(char s[])
{
    long reg;
    switch(s[1]){
        case 'z':   reg=0;                  break;
        case 'v':   reg=s[2]-'0'+2;         break;
        case 'k':   reg=s[2]-'0'+62;        break;
        case 'a':
            if(s[2]=='t')reg=1;
            else reg=s[2]-'0'+4;
            break;
        case 's':
            if(s[2]=='p') reg=29;
            else reg=s[2]-'0'+16;
            break;
        case 't':
            if((s[2]-'0')>=8) reg=s[2]-'0'+24;
            else reg=s[2]-'0'+8;
            break;
        case 'g':   reg=28;             break;
        case 'f':   reg=30;             break;
        case 'r':   reg=31;             break;
        default:    reg=-1;             break;
    }
    return (reg&31);
}
```

(3)检查操作码(除标号的第一个词),确定指令类型。并转化成32位的二进制机器指令码。

```
if(strcmp(wrd[0], "add")==0){
    ir=(regN(wrd[2])<<21)|(regN(wrd[3])<<16)
                         |(regN(wrd[1])<<11)|32;
} else if(strcmp(wrd[0], "sub")==0){
    ir=(regN(wrd[2])<<21)|(regN(wrd[3])<<16)
                         |(regN(wrd[1])<<11)|34;
} else if(strcmp(wrd[0], "slt")==0){
    ir=(regN(wrd[2])<<21)|(regN(wrd[3])<<16)
                         |(regN(wrd[1])<<11)|42;
} else if(strcmp(wrd[0], "beq")==0){
...
```

反汇编：

汇编是机器指令的助记形式，每条机器指令都对应一个汇编指令，因此相对比较容易实现，不必考虑伪指令。反汇编是将机器指令转换成相应的 MIPS 汇编。

MIPS 采用等长指令系统，指令的最高 6 位固定为操作码，所以首先应根据高 6 位决定指令类型，再对其余 26 位划分字段，决定操作数的内容。

编写反汇编程序的一般步骤如下。

(1)二进制寄存器号与寄存器名的转换子程序。

```
int    regNam(char str[], int r)
{
    if(r >=8 && r <=15)sprintf(str, "$t%d", r -8);
    else if(r >=16 && r <=23)sprintf(str, "$s%d", r -16);
    else switch(r&31){
        case 0:strcpy(str, "$zero");    break;
        case 1:strcpy(str, "$at");      break;
        case 2:strcpy(str, "$v0");      break;
        case 3:strcpy(str, "$v1");      break;
        case 4:strcpy(str, "$a0");      break;
        case 5:strcpy(str, "$a1");      break;
        case 6:strcpy(str, "$a2");      break;
        case 7:strcpy(str, "$a3");      break;
        case 24:strcpy(str, "$t8");     break;
        case 25:strcpy(str, "$t9");     break;
        case 26:strcpy(str, "$k0");     break;
        case 27:strcpy(str, "$k1");     break;
        case 28:strcpy(str, "$gp");     break;
        case 29:strcpy(str, "$sp");     break;
        case 30:strcpy(str, "$fp");     break;
        case 31:strcpy(str, "$ra");     break;
        default:    return 0;           break;
    }
    return 1;
}
```

为方便随后的转换，先按各种可能的指令类型确定每个操作数值：操作码、源 1 寄存器中、源 2 寄存器、目标寄存器、立即数和长立即数(J 指令)。

```
ir =((dword)mem[pc +3] <<24) |((dword)mem[pc +2] <<16)
  |((dword)mem[pc +1] <<8 ) |(dword)mem[pc];     //指令
pc +=4;

op = ir >>26;          //操作码
rs =(ir >>21)&31;          regNam(rstr, rs);
rt =(ir >>16)&31;          regNam(tstr, rt);
rd =(ir >>11)&31;          regNam(dstr, rd);
```

```
shmt = (ir >> 6)&31;        func = ir&63;
data = (ir&0xffff);
addr = (pc&0xf0000000) |((ir&0x03ffffff) <<2);
```

(2)取高6位操作码,决定指令类型。

将操作数转换成相应的符号形式。对于转移地址,反汇编程序不可能知道原始的标号,可按地址形式给出,也可用设计一种统一的标号规则确定。

```
switch(op){
case 0:                              //R-format
    switch(func){
    case 32:                         //add
        sprintf(mstr, "add \t% s, % s, % s", dstr, rstr, tstr);
        break;
    case 34:                         //sub
        sprintf(mstr, "sub \t% s, % s, % s", dstr, rstr, tstr);
        break;
    case 42:                         //slt
        sprintf(mstr, "slt \t% s, % s, % s", dstr, rstr, tstr);
        break;
    }    break;
case 4:                              //beq
    sprintf(mstr, "beq \t% s, % s, % d", rstr, tstr, pc + data * 4);
    break;
...
```

MIPS 模拟机:

以面向对象的方式,建立 mipscpu 类。用程序变量表示寄存器,用字节数组(对应字节编址)表示内存作为类的属性,而以指令等作为类的方法。通过加载执行编译好的 MIPS 指令,改变寄存器与内存的状态。类似 Windows 中 DEBUG 的形式,边执行边显示寄存器、内存的值。

```
#define MAXMEM    1024
class mispcpu {
    long    reg[32];
    byte    mem[MAXMEM];
    dword   ir;
    long    pc;
...
```

除了以上在 CPU 中真实的寄存器外,还可以再设置一些变量,对应数据通道中的内部寄存器与引线等中间值。

```
short    op, rs, rt, rd, shmt, func, data;
long     addr;
char     rstr[10], tstr[10], dstr[10];//为显示用
```

译码方法:

```
int    mispcpu::decode()
```