实验八　单周期 CPU 中指令控制器的设计与仿真

一、实验目的

1、深入了解单周期 CPU 中指令控制器的结构和工作原理。
2、学习使用 Verilog HDL 设计实现单周期 CPU 中指令控制器。

二、实验设备

1、装有 vivado 的计算机　　1 台
2、EG01 开发板　　1 块

三、实验任务

1、设计和实现单周期 CPU 中指令控制器的结构并且进行功能仿真。

四、实验原理

1、一个单周期 CPU 的硬件结构如图 8.1 所示。其中红色框内部分是控制器。

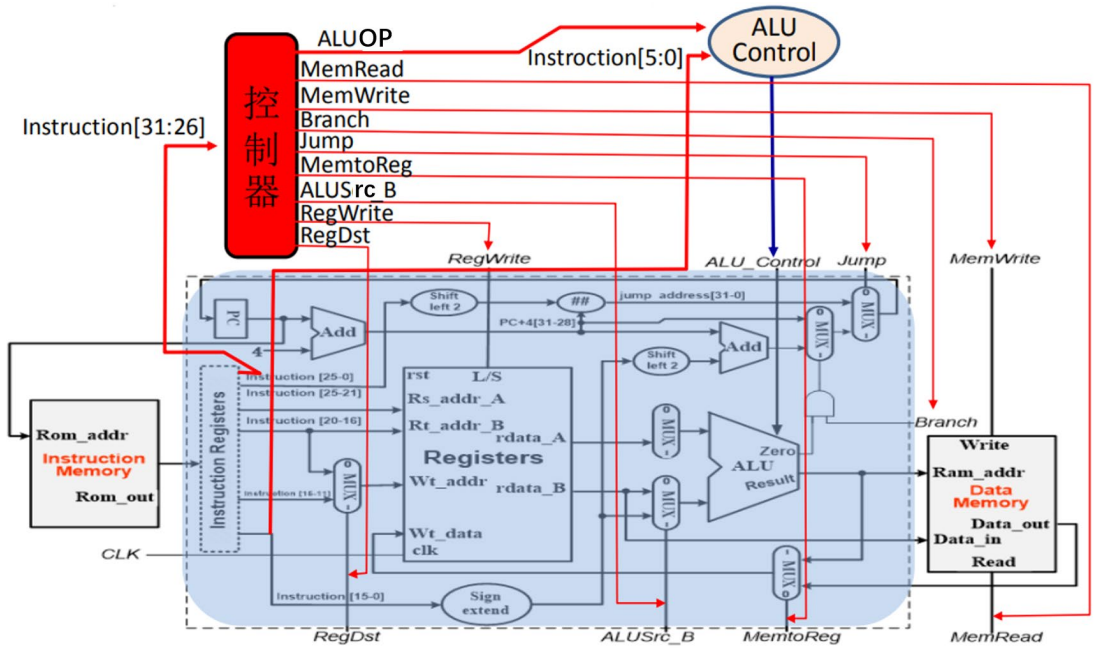

图 8.1　单周期 CPU 的结构

在图 8.1 中，控制器模块接收从指令存储器读出指令的机器码，根据指令机器码的高 6 位 instruction[31:26]进行译码后，生成控制信号 RegDst、RegWrite、ALUSrc_B、MemtoReg、Jump、Branch、MemWrite、MemRead、ALUOP。其中的控制信号 ALUOP 与指令机器码的低 6 位 instruction[5:0]联合起来，生成运算器运算类型相对应的控制信号 ALU_Control。

信号 MemRead：数据存储器读控制信号。
信号 MemWrite：数据存储器写控制信号。
信号 Branch：条件跳转控制信号，送给对应的二选一数据选择器。
信号 Jump：直接跳转控制信号，送给对应的二选一数据选择器。
信号 MemtoReg：将数据存储器读出的结果写入寄存器的控制信号。
信号 ALUSrc_B：操作数 B 的选择对应的控制信号。

信号 RegWrite：对寄存器进行写操作的控制信号。

信号 RegDst：从二选一选择器中选中对应的值，写入对应的寄存器。具体参看表 8.1。

控制器输出信号赋值为 0 和赋值为 1 代表的含义如下表 8.1 所示：

表 8.1 控制器输出信号功能定义表

信号	源数目	功能定义	赋值 0 时动作	赋值 1 时动作
ALUSrc_B	2	ALU 端口 B 输入选择	选择寄存器 B 数据	选择 32 位立即数(符号扩展后)
RegDst	2	寄存器写地址选择	选择指令 rt 域	选择指令 rs 域
MemtoReg	2	寄存器写入数据选择	选择存储器数据	选择 ALU 输出
Branch	2	Beq 指令目标地址选择	选择 PC+4 地址	选择转移地址(Zero=1)
Jump	2	J 指令目标地址选择	选择 J 目标地址	由 Branch 决定输出
RegWrite	-	寄存器写控制	禁止寄存器写	使能寄存器写
MemWrite	-	存储器写控制	禁止存储器写	使能存储器写
MemRead	-	存储器读控制	禁止存储器读	使能存储器读
ALU_Control	000-111	3 位 ALU 操作控制		

在 MIP 指令集中，指令格式分为三种，分别为 R 类型、I 类型、J 类型指令。如下图 8.2。

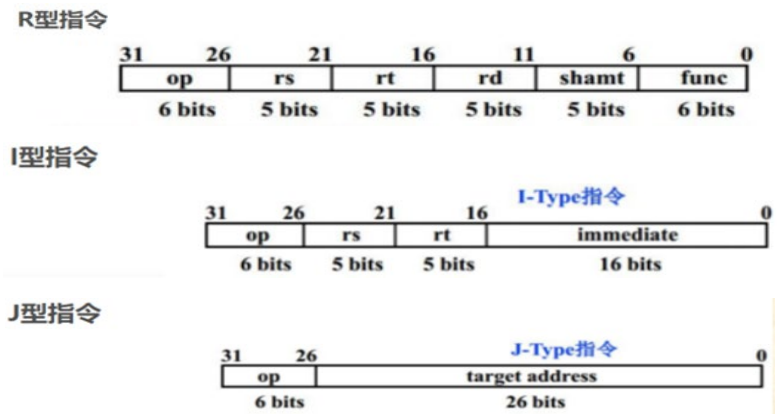

图 8.2 MIPS 指令集中三种指令格式

图 8.2 中，op 部分字段代表操作码。func 部分字段代表功能码。

表 8.2 列出了 MIPS 指令集中常见的 20 条指令的机器码的格式。

表 8.2　MIPS 指令集中 20 条指令的机器码

指令	[31:26]	[25:21]	[20:16]	[15:11]	[10: 6]	[5:0]	功能
R 型指令							
Add	000000	rs	rt	rd	000000	100000	寄存器加
Sub	000000	rs	rt	rd	000000	100010	寄存器减
And	000000	rs	rt	rd	000000	100100	寄存器与
Or	000000	rs	rt	rd	000000	100101	寄存器或
Xor	000000	rs	rt	rd	000000	100110	寄存器异或
Sll	000000	00000	rt	rd	sa	000000	左移
Srl	000000	00000	rt	rd	sa	000010	逻辑右移
Sra	000000	00000	rt	rd	sa	000011	算术右移
Jr	000000	rs	rt	rd	000000	001000	寄存器跳
I 型指令							
Addi	001000	rs	rt	immediate			立即数加
Andi	001100	rs	rt	immediate			立即数与
Ori	001101	rs	rt	immediate			立即数或
Xori	001110	rs	rt	immediate			立即数异或
Lw	100011	rs	rt	offset			取数据
Sw	101011	rs	rt	offset			存数据
Beq	000100	rs	rt	offset			相等转移
Bne	000101	rs	rt	offset			不等转移
Lui	001111	00000	rt	immediate			设置高位
J 型指令							
J	000010	address					跳转
Jal	000011	address					调用

控制器输入的操作码与输出信号对应关系，如下表 8.3 所示。请可以提前填写完成表 8.3 中有确定值的空白方格。有些方格的值是不确定的，填写 X 即可。

表 8.3 控制器输入的操作码与输出信号对应表

OP	对应的指令	Reg Dst	ALU Src	Mem toReg	Reg Write	Mem Read	Mem Write	Branch	Jump	ALU op1	ALU op0
R-格式 000000	ADD	1	0	0	1	0	0	0	0	1	0
R-格式 000000	SUB	1	0	0	1	0	0	0	0	1	0
R-格式 000000	AND										
R-格式 000000	OR										

R-格式 000000	SLL										
I-格式 LW	LOAD										
I-格式 SW	STORE										
I-格式 beq	BEQ										
J-格式 指令	Jump										

控制器接口信号如下图 8.3 所示。

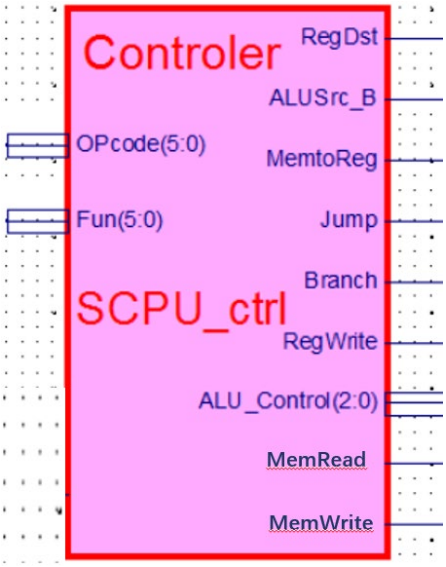

图 8.3 控制器的接口引脚示意图

CPU 部件之控制器 SCPU_ctrl 接口代码如下:

```
module SCPU_ctrl(
input[5:0] OPcode, //OPcode
input[5:0] func, //function
output reg RegDst,
output reg ALUSrc_B,
output reg MemtoReg,
output reg Jump,
output reg Branch,
output reg RegWrite,
output reg[2:0] ALU_Control,
output reg MemRead,
output reg MemWrite  );
……
 endmodule
```

控制器对应的控制电路结构图类似下图 8.4

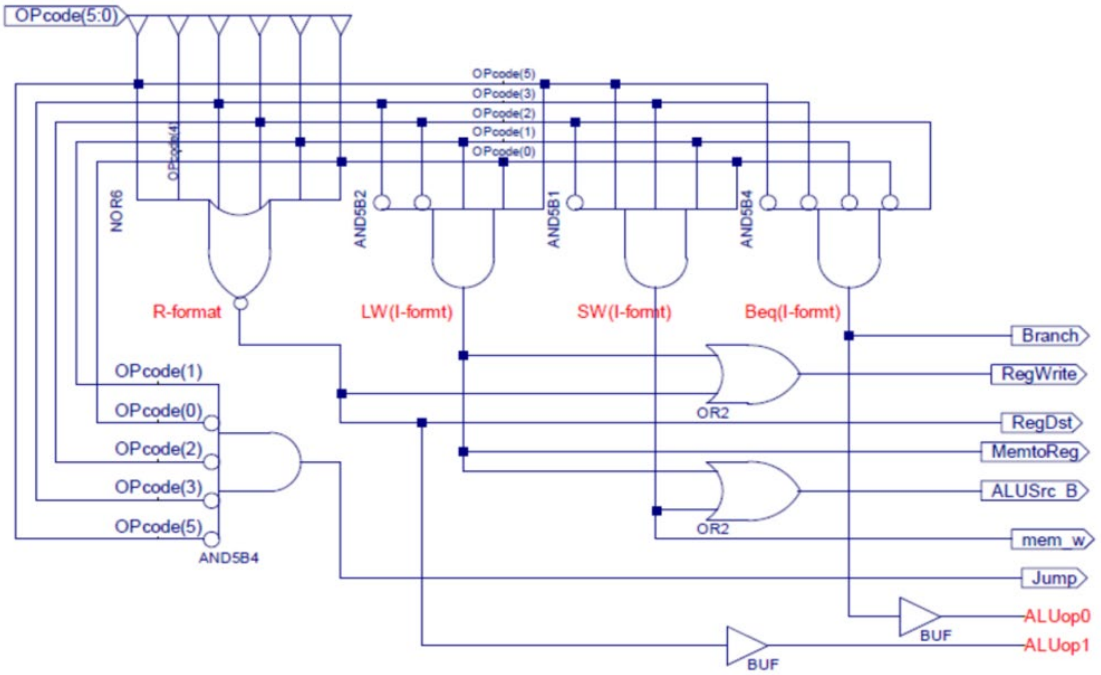

图 8.4 控制器控制电路结构示意图

图 8.1 中的控制信号 ALUOP 与指令机器码的低 6 位 instruction[5:0]联合起来，生成运算器运算类型相对应的控制信号 ALU_Control。控制器输入的操作码与运算器控制信号 ALU_Control 对应关系如表 8.4 所示。

表 8.4 控制器输入的操作码与运算器控制信号 ALU_Control 对应表

Instruction opcode	ALUOP	Instruction operation	func field	Desired ALU action	ALU_Control
LW	00	Load word	xxxxxx	Load word	010
SW	00	Store word	xxxxxx	Store word	010
Beq	01	branch equal	xxxxxx	branch equal	110
R-type	10	add	10 0000	add	010
R-type	10	subtract	10 0010	subtract	110
R-type	10	AND	10 0100	AND	000
R-type	10	OR	10 0101	OR	001
R-type	10	Set on less than	10 1010	Set on less than	111
R-type	10	NOR	10 0111	NOR	100

控制器输入的操作码与运算器控制信号 ALU_Control 对应逻辑电路的结构如图 8.5 所示。

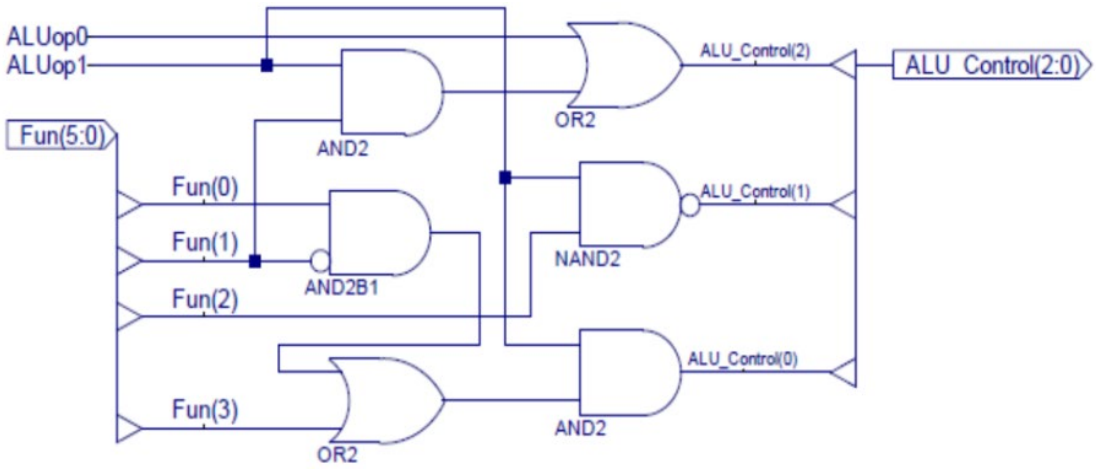

图 8.5 控制器输入的操作码与运算器控制信号 ALU_Control 对应逻辑电路的结构

五、实验步骤

（一）设计单周期 CPU 的指令控制器（实验室完成和进行仿真验收）

1、新建 Vivado 工程　　参照实验五的步骤创建新工程。

参考上面的图 8.1、图 8.4 和图 8.5，设计单周期 CPU 的**指令控制器电路**，接口定义类似如下:

```
module SCPU_ctrl(
input[5:0] OPcode,     //OPcode
input[5:0] func,       //function
output reg RegDst,
output reg ALUSrc_B,
output reg MemtoReg,
output reg Jump,
output reg Branch,
output reg RegWrite,
output reg[2:0] ALU_Control,
output reg MemRead,
output reg MemWrite  );
```

……请在此处补充对应的 verilog 代码，要求在实验前完成。进实验室时会检查。验收时会围绕核心代码进行提问。

```
 endmodule
```

请补充上面控制器模块中没有完成的 verilog 代码。

2、在 Vivado 中进行仿真测试单周期 CPU 的控制器模块。

建立了顶层模块后,如果需要对顶层模块进行测试的话,就需要给顶层模块添加相应的输入,观察顶层模块的输出所具有的特点。例如：将加、减、与、或、跳转等多条指令的机器码,每隔 100ns 依次送入。观察控制器输出信号对应的波形是否正确。

完成下面控制器激励输入模块:

module testcontrol(……)

请在此处补充对应的 verilog 代码，要求在实验前完成。进实验室时会检查。验收时会围绕核心代码进行提问。

endmodule

进行仿真，验证仿真后的波形图是否正确。

在下面粘贴你综合后生成的电路的原理图。

在下面粘贴你的指令执行对应的仿真波形截图，并分析指令执行结果的正确性。

(三)、结合 EGO1 开发板，在 Vivado 中生成 bit 文件和下载。将多条指令的机器码存放到 instruction.txt 文件中，通过 readmem（）函数可以加载 instruction.txt 文件中指令的机器码，在开关的控制下依次将指令送入单周期 CPU 的控制电路中，将控制电路的输出送到 EGO1 开发板上 LED0、LED1、LED2、LED3…和 LED15 这些发光二极管上去。要求开关 sw0、sw1……sw7 中某个开关值为 1 就执行对应的指令。例如 sw0、sw1……sw7 中 8 个开关值为 10000000 时，执行第一条指令。sw0、sw1……sw7 中 8 个开关值为 01000000 时，执行第二条指令。sw0、sw1……sw7 中 8 个开关值为 00100000 时，执行第三条指令……依此类推，观察 16 个发光二极管的亮灭情况，记入下表 8.5。可以用 16 个开关对应于 16 条指令的执行。也可以自己定义 EGO1 开发板上开关与指令之间的对应关系。

表 8.5 指令执行时控制器输出值

指令	对应的指令	Reg Dst	ALU Src	Mem toReg	Reg Write	Mem Read	Mem Write	Branch	Jump	ALU op1	ALU op0
第 1 条	ADD										
第 2 条	SUB										
第 3 条	OR										
第 4 条	AND										
第 5 条	NOT										
第 6 条	LOAD										
第 7 条	STORE										
第 8 条	Branch										
第 9 条	Jump										

对于 instruction.txt 文件中存放的指令的机器码，可以参考和采用下面汇编代码对应的机器码。也可以自己设置对应的机器码。

注意下面的汇编指令与后面注释里机器码的对应关系，从而方便相互转换。下面竖线后面显示的是每条指令对应的执行结果。$1：代表编号为 1 的寄存器。#19：代表立即数 19。

```
addiu $1 ,$0,#1              // 32'h24010001; // 00H: addiu $1 ,$0,#1     | $1 = 0000_0001H
sll   $2 ,$1,#4              // 32'h00011100; // 04H: sll   $2 ,$1,#4     | $2 = 0000_0010H
addu  $3 ,$2,$1              // 32'h00411821; // 08H: addu  $3 ,$2,$1     | $3 = 0000_0011H
srl   $4 ,$2,#2              // 32'h00022082; // 0CH: srl   $4 ,$2,#2     | $4 = 0000_0004H
subu  $5 ,$3,$4              // 32'h00642823; // 10H: subu  $5 ,$3,$4     | $5 = 0000_000DH
sw  $5 ,#19($1)   //32'hAC250013; // 14H: sw $5 ,#19($1) | Mem[0000_0014H] = 0000_000DH
nor   $6 ,$5,$2              // 32'h00A23027; // 18H: nor   $6 ,$5,$2     | $6 = FFFF_FFE2H
or    $7 ,$6,$3              //32'h00C33825; // 1CH: or    $7 ,$6,$3     | $7 = FFFF_FFF3H
xor   $8 ,$7,$6              // 32'h00E64026; // 20H: xor   $8 ,$7,$6     | $8 = 0000_0011H
sw $8 ,#28($0)   //32'hAC08001C; // 24H: sw $8 ,#28($0) | Mem[0000_001CH] = 0000_0011H
slt   $9 ,$6,$7              //32'h00C7482A; // 28H: slt   $9 ,$6,$7     | $9 = 0000_0001H
beq   $9 ,$1,#2              //32'h11210002; // 2CH: beq   $9 ,$1,#2     | 跳转到指令 34H
addiu $1 ,$0,#4              //32'h24010004; // 30H: addiu $1 ,$0,#4     | 不执行
lw    $10,#19($1)            //32'h8C2A0013; // 34H: lw    $10,#19($1) | $10 = 0000_000DH
bne   $10,$5,#3              //32'h15450003; // 38H: bne   $10,$5,#3     | 不跳转
and   $11,$2,$1              //32'h00415824; // 3CH: and   $11,$2,$1     | $11 = 0000_0000H
sw  $11,#28($0)   //32'hAC0B001C;//40H: sw $11,#28($0) | Men[0000_001CH] = 0000_0000H
sw $4 ,#16($0)    //32'hAC040010; // 44H: sw $4 ,#16($0) | Mem[0000_0010H] = 0000_0004H
lui   $12,#12                // 32'h3C0C000C; // 48H: lui   $12,#12        | [R12] = 000C_0000H
j     00H                    //32'h08000000; // 4CH: j     00H           | 跳转指令 00H
```

（四）、结合图 8.1 单周期 CPU 的结构，补充完整整个 CPU 的硬件结构，可以描述 PC 模块、执行模块、数据存储器模块、指令存储器模块、寄存器组模块、时钟模块以及顶层模块 top。然后对整个单周期 CPU 硬件结构进行激励输入和仿真。（不验收）

六、实验验收要求和思考

1、可以使用实验室的电脑进行验收。也可以自己带电脑到实验室，在自己的电脑上完成实验，然后在自己的电脑上进行验收。验收时，要求能简单介绍一下系统能实现的主要功能，核心代码的含义和实验中引脚的分配等等。老师会简单提问，如果回答问题较差，会酌情扣分。所以要求对代码进行提前预习和准备。

2、实验八的验收规则为：只验收到实验步骤 2，验收成绩 80 分。验收到步骤 3，验收成绩 100 分。如果当次实验课没有验收完，可以在其他班的实验课时进行补验收。

3、下载情况下，输出信号还可以送到 EGO1 开发板上的哪里进行显示？如果输出信号送到七段显示器上进行显示，verilog 代码应该怎么写？

七、附录参考（MIPS 指令集）

指令编号	MIPS	执行效果	含义	十六进制	二进制
0	lui $t0,255		#立即数255加载至寄存器t0高16位	3c0800ff	00111100000010000000000011111111
1	lui $t1,1		#立即数2加载至寄存器t1高16位	3c090001	00111100000010010000000000000001
2	addu $t2,$t0,$t1		#把寄存器t0与t1的值相加，结果存在t2中	01095021	00000001000010010101000000100001
3	subu $t2,$t2,$t1		#把寄存器t2与t1的值相减，结果存在t2中	01495023	00000001010010010101000000100011
4	ori $t3,$t0,21845		#把寄存器t0中的值与21845按位或，结果存在寄存器t3中	350b5555	00110101000010110101010101010101
5	lui $t4,1		#立即数1加载至寄存器t4高16位	3c0c0001	00111100000011000000000000000001
6	lui $t5,1		#立即数1加载至寄存器t5高16位	3c0d0001	00111100000011010000000000000001
7	ori $t5,$t5,1		#把寄存器t5中的值与1按位或，结果存在寄存器t5中	35ad0001	00110101101011010000000000000001
8	subu $t4,$t5,$t4	#t4 = 1	#把寄存器t5与t4的值相减，结果存在t4中	01ac6023	00000001101011000110000000100011
9	addu $t0,$t7,$t4	#t0 = 1	#把寄存器t7与t4的值相加，结果存在t0中	01ec4021	00000001111011000100000000100001
10	addu $t0,$t0,$t0	#t0 = 2	#把寄存器t0与t0的值相加，结果存在t0中(t0=t0*2)	01084021	00000001000010000100000000100001
11	addu $t0,$t0,$t0	#t0 = 4 设置循环执行4次	#(t0=t0*2)	01084021	00000001000010000100000000100001
12	addu $t1,$t7,$t7	#t1 = 0 设置循环从第0次开始	#把寄存器t7与t7的值相加，结果存在t1中	01ef4821	00000001111011110100100000100001
13	addu $t5,$t4,$t4	#t5 = 2 希望t5 = 5	#把寄存器t4与t4的值相加，结果存在t5中	018c6821	00000001100011000110100000100001
14	addu $t5,$t5,$t5	#t5 = 4 希望t5 = 5	#把寄存器t5与t5的值相加，结果存在t5中	01ad6821	00000001101011010110100000100001
15	label:addu $t5,$t5,$t4	#t5 = 5 循环开始	#把寄存器t4与t5的值相加，结果存在t5中	01ac6821	00000001101011000110100000100001
16	addu $t6,$t5,$t4	#t6 = 6 循环部分	#把寄存器t4与t5的值相加，结果存在t6中	01ac7021	00000001101011000111000000100001
17	addu $t7,$t6,$t4	#t7 = 7 循环部分	#把寄存器t4与t6的值相加，结果存在t7中	01cc7821	00000001110011000111100000100001
18	addu $t1,$t1,$t4	#循环执行次数++	#把寄存器t4与t1的值相加，结果存在t1中	012c4821	00000001001011000100100000100001
19	beq $t0,$t1,lbbreak	如果循环执行次数 = 4，则break	如果t0的值等于t1的值，则跳转到lbbreak标签	11090001	00010001000010010000000000000001
20	j label	循环结尾	无条件跳转至label标签	0810000f	00001000000010000000000000001111
21	lbbreak:lui $t0,0	break出来之后执行的语句	#立即数0拓展到高16后加载至寄存器t0	3c080000	00111100000010000000000000000000