Verilog 流水线处理器

20373864　　谭立德

一、CPU 设计方案综述

（一）总体设计概述

本 CPU 为 verilog 实现的流水线 MIPS - CPU，支持的指令集包含 MIPS-C3={LB、LBU、LH、LHU、LW、SB、SH、SW、ADD、ADDU、 SUB、SUBU、 MULT、 MULTU、DIV、 DIVU、SLL、 SRL、 SRA、SLLV、SRLV、SRAV、AND、OR、XOR、NOR、ADDI、ADDIU、ANDI、ORI、 XORI、LUI、SLT、SLTI、SLTIU、SLTU、BEQ、BNE、BLEZ、BGTZ、 BLTZ、BGEZ、J、JAL、JALR、JR、MFHI、MFLO、MTHI、MTLO}。为了实现这些功能，CPU 主要包含了 IM、GRF、DM、ALU、PC、CU 等主要模块，这些模块按照自顶向下的顶层设计逐级展开。

（二）关键模块定义

1. GRF （通用寄存器组，也称为寄存器文件、寄存器堆）

GRF 端口定义：

表 0 GRF 端口表

信号名	方向	描述
Clk	I	时钟信号
Reset	I	复位信号，将 32 个寄存器中的值全部清零 1：复位 0：无效
We	I	写使能信号 1：可向 GRF 中写入数据 0：不可向 GEF 中写入数据
A1	I	5 位地址输入信号，指定 32 个寄存器中的一个，将其中存储的数据读出至 RD1
A2	I	5 位地址输入信号，指定 32 个寄存器中的一个，将其中存储的数据读出至 RD2
A3	I	5 位地址输入信号，指定 32 个寄存器中的一个作为写入的目标寄存器
WD3	I	32 位数据输入信号

RD1	O	输出指定的寄存器中的 32 位数据
RD2	O	输出指定的寄存器中的 32 位数据

GRF 模块功能定义：

表 1 GRF 功能表

序号	功能名称	描述
1	复位	Reset 信号有效时，所有寄存器储存的数值清零
2	读数据	读出 A1，A2 地址对应寄存器中所储存的数据到 RD1，RD2
3	写数据	当 WE 有效且时钟上升沿来临时，将 WD 写入 A3 所对应的寄存器中

2. DM （数据存储器）：

DM 端口定义：

表 2 DM 端口表

信号名	方向	描述
Clk	I	时钟信号
Reset	I	复位信号，将 32 个寄存器中的值全部清零 1：复位 0：无效
We	I	写使能信号 1：可向 DM 中写入数据 0：不可向 DM 中写入数据
A	I	5 位地址输入信号，指定中储存器上的地址，将其中存储的数据读出至 RD1
WD	I	32 位数据输入信号
RD	O	输出储存器指定地址上的 32 位数据

DM 模块功能定义：

表 3 DM 功能表

序号	功能名称	描述
1	复位	Reset 信号有效时，储存器储存的所有数值清零
2	读数据	读出 A 地址对应储存器中所储存的数据到 RD
3	写数据	当 WE 有效且时钟上升沿来临时，将 WD 写入 A3 所对应的寄存器中

3. ALU （算术逻辑运算单元）：

ALU 端口定义：

表 4 ALU 端口表

信号名	方向	描述
SrcA	I	32 位运算数输入信号
SrcB	I	32 位运算数输入信号
ALU Control	I	3 位逻辑运算选择信号，选择进行哪种逻辑运算
Zero	O	输出比较两运算数比较的 1 位输出
ALU Result	O	输出对两运算数进行指定逻辑运算后的 32 位结果

ALU 模块功能定义：

表 5 ALU 功能表

序号	功能名称	描述
1	计算	根据控制信号进行对应的逻辑计算并输出
2	比较	判断两个输入是否相等

4. IM （指令存储器）：

IM 端口定义：

表 6 IM 端口表

信号名	方向	描述
PC	I	5 位输入地址信号
Instr	O	输出地址所储存 32 位指令

IM 模块功能定义：

表 7 IM 功能表

序号	功能名称	描述
1	读指令	根据输入输出对应 32 位指令

5. Control Unit （指令译码器）：

Control Unit 端口定义：

表 8 Control Unit 端口表

信号名	方向	描述
Opcode[5:0]	I	指令操作码
Funct[5:0]	I	指令功能码

Jump	0	跳转信号
ToHigh16	0	高位置位信号
ExtOp	0	位扩展方式
MemtoReg	0	读内存信号
MemWrite	0	内存写使能信号
Branch	0	分支信号
ALUCtrl[2:0]	0	ALU 控制信号
ALUSrc	0	ALU 操作数 2 的来源 0：寄存器 1：立即数
RegDst	0	寄存器写地址选择 0：Instr[20:16] 1：Instr[15:11]
RegWrite	0	寄存器写使能信号
DMop[1:0]	0	存储、读取方式控制信号

（三）重要机制实现方法

1. J 类型指令

根据输入判断和 ALU 模块协同工作算出跳转地址后跳转。

2. R 类型指令

根据输入判断和 ALU 模块协同工作算出结果后存储回寄存器堆中以实现指令 R 类型指令。

3. I 类型指令

根据输入判断和 ALU 模块和 DM 模块协同工作支持 I 类型指令。

二、测试方案

（同 P5）

test_input.py > mipsDir

```python
# Coding in UTF-8
import os
import re
import shutil

################################################################################################
f = open("result.txt", "w")


def fileCmp(std_path, ise_path, std, ise, filename):  # file a,b
    stdText = std.read()
    iseText = ise.read()

    isSame = True

    stdLogs = re.findall("@[^\\n]*\\n?", stdText)
    iseLogs = re.findall("@[^\\n]*\\n?", iseText)

    for i in range(len(stdLogs)):
        if (stdLogs[i] != iseLogs[i]):
            isSame = False
            f.write(filename + ":\n")
            f.write("\tWrong: " + "At Line " + str(i) + " : " + "we want " +
                    stdLogs[i] + " " + " but we get " + iseLogs[i] + "\n")
            break
    if (isSame is True):
        print("\tAccepted")
        f.write("Accepted: " + filename + "\n")
        flag = 1
    else:
        print("\tWrongAnswer")
        flag = 0

    stdLog.close()
    iseLog.close()

    if (flag == 1):
        os.remove(std_path)
        os.remove(ise_path)

    return flag


################################################################################
# mipsDir = input(
#     "需要编译的mips程序绝对地址(e.g. D:/mips.asm): \n")  # Mips File For Mars to Compile

mipsDirs = []
for filename in os.listdir():
    if re.match(r"[\w]+\.asm", filename):
        mipsDirs.append(filename)
hexCodeDir = "code.txt"  # Hex Code For ISE

for mipsDir in mipsDirs:
    #  Dump Hex Code And Get Std Log
    # spMarsJarDir = input("修改版Mars绝对地址(e.g. D:/Mars.jar)\n")
    spMarsJarDir = "Mars_test.jar"  # 修改版Mars地址
    stdLogDir = mipsDir[:-4] + "_std_ans.txt"  # 标准输出
    os.system("java -jar " + spMarsJarDir + " " + mipsDir +
```

```
    os.system("java -jar " + spMarsJarDir + " " + mipsDir +
              " 100000 db nc mc CompactDataAtZero a dump .text HexText " + hexCodeDir)
    os.system("java -jar " + spMarsJarDir + " " + mipsDir +
              " 100000 db nc mc CompactDataAtZero >" + stdLogDir)

    # Prepare ISE exe
    testDir = input("工程文件夹地址(e.g. D:/test): \n")
    # testDir = "E:/ISE/Project_4"
    # tcl文件和prj文件需要放在工程同目录下
    tclFile = open(testDir + "/test.tcl", "w")
    # tcl文件声明了工程运行的参数
    tclFile.write("run 100us;\nexit")
    # prj文件声明了工程所含各模块的位置
    prjFile = open(testDir + "/test.prj", "w")
    for root, dirs, files in os.walk(testDir):
        for fileName in files:
            if re.match(r"[\w]+\.v", fileName):
                prjFile.write("Verilog work " + root + "/" + fileName + "\n")

    tclFile.close()
    prjFile.close()
    # Run ISE simulation
    iseCompileLogDir = "ise_log.txt"
    userLogDir = mipsDir[:-4] + "_ise_ans.txt"  # 我的输出

    ise_path = input("ISE安装文件夹(e.g. D:/Xilinx/14.7/ISE_DS/ISE):\n")
    os.environ['XILINX'] = ise_path
    # os.environ['XILINX'] = "D:/Xilinx/14.7/ISE_DS/ISE"  # ISE安装文件夹

```

```
    os.system(ise_path + "/bin/nt64/fuse -nodebug -prj " +
              testDir + "/test.prj" + " -o " + "testmips.exe mips_test>" +
              iseCompileLogDir)

    os.system("testmips.exe -nolog -tclbatch " + testDir + "/test.tcl" + ">" +
              userLogDir)
    # Mars Log Complete
    stdLog = open(stdLogDir, "r")  # 标准答案
    iseLog = open(userLogDir, "r")  # 你的答案

    fileCmp(stdLogDir, userLogDir, stdLog, iseLog, mipsDir)

    # os.remove("test/code.txt")
    os.remove("fuse.log")
    os.remove("fuse.xmsgs")
    os.remove("fuseRelaunch.cmd")
    os.remove("isim.wdb")
    os.remove("testmips.exe")
    os.remove("ise_log.txt")
    shutil.rmtree("isim")

f.close()

```

三、思考题

1.为什么需要有单独的乘除法部件而不是整合进 ALU？为何需要有独立的 HI、LO 寄存器？

解：

乘除指令运算非常慢，所以我们的 CPU 要模拟这种时延（两种乘法延时 5 个周期，两种除法延时 10 个周期）。为了不让乘除法拖慢速度，我们遇到相关指令时让他们进入一个特殊的乘除部件进行运算，并且让结果存储在 HI，LO 中，只在特殊指令进行调用，其他指令继续执行。如果乘除法正在计算，而我们遇到了新的要用到乘除部件的指令，那么就 stall。

2.参照你对延迟槽的理解，试解释 “乘除槽”。

解：

当乘除法进行或即将开始时，下一条乘除有关指令会被阻塞在 IF/ID 流水线寄存器，即相当于处于“乘除槽”，而其他指令正常流水执行。

3.举例说明并分析何时按字节访问内存相对于按字访问内存性能上更有优势。（Hint： 考虑 C 语言中字符串的情况）

解：

当访问类型只占一个字节时，比如 char 时。

4.在本实验中你遇到了哪些不同指令类型组合产生的冲突？你又是如何解决的？相应的测试样例是什么样的？（如果你是手动构造的样例，请说明构造策略，说明你的测试程序如何保证覆盖了所有需要测试的情况；如果你是完全随机生成的测试样例，请思考完全随机的测试程序有何不足之处；如果你在生成测试样例时采用了特殊的策略，比如构造连续数据冒险序列，请你描述一下你使用的策略如何结合了随机性达到强测的效果。（此思考题请同学们结合自己测试 CPU 使用的具体手段，按照自己的实际情况进行回答））

解：

不同类型组合产生的冲突：

241,158,192

s_s/s_v/mf -> cal_r

转发：

0	1	2	3	4	5		ID/EX		EX/MEM				MEM/WB				
输入0	AO_M	M4	PC8_E	PC8_M	PC8_W		Tnew		Tnew				Tnew				
							jal	jalr	cal_r	cal_i	jal	jalr	cal_r	cal_i	load	jal	jalr
流水级	源寄存器	涉及指令		MUX	控制信号	输入0	0/31	0/rd	0/rd	O/rt	0/31	0/rd	0/rd	O/rt	O/rt	0/31	0/rd
D	rs	c_r,c_i,ld,st,b,jr,jalr,s_s		MF_RS_D	F_RS_Dsel	RF.RD1	PC8_E	PC8_E	AO	AO	PC8_M	PC8_M	M4	M4	M4	PC8_W	PC8_W
D	rt	c_r,st,b,s_s		MF_RT_D	F_RT_Dsel	RF.RD2	PC8_E	PC8_E	AO	AO	PC8_M	PC8_M	M4	M4	M4	PC8_W	PC8_W
E	rs	c_r,c_i,ld,st,s_s		MF_ALUA_E	F_ALUA_Esel	V1@E			AO	AO	PC8_M	PC8_M	M4	M4	M4	PC8_W	PC8_W
ALU_E	rt	c_r,st,s_s		MF_ALUB_E	F_ALUB_Esel	V2@E			AO	AO	PC8_M	PC8_M	M4	M4	M4	PC8_W	PC8_W
M	rt	st		MF_WD_M	F_WD_Msel	V2@M							M4	M4	M4	PC8_W	PC8_W
DM_M																	

暂停：

IF/ID当前指令			ID/EX			EX/MEM
指令类型	源寄存器	Tuse	Tnew			Tnew
			cal_r	cal_i	load	load
			1/rd	1/rt	2/rt	1/rt
beq	rs/rt	0	暂停	暂停	暂停	暂停
cal_r	rs/rt	1			暂停	
cal_i	rs	1			暂停	
load	rs	1			暂停	
store	rs	1			暂停	
store	rt	2				
jr	rs	0	暂停	暂停	暂停	暂停
jalr	rs	0	暂停	暂停	暂停	暂停

解决方法：构造转发和暂停机制，在遇到冲突冒险时进行对应转发和冒险操作。

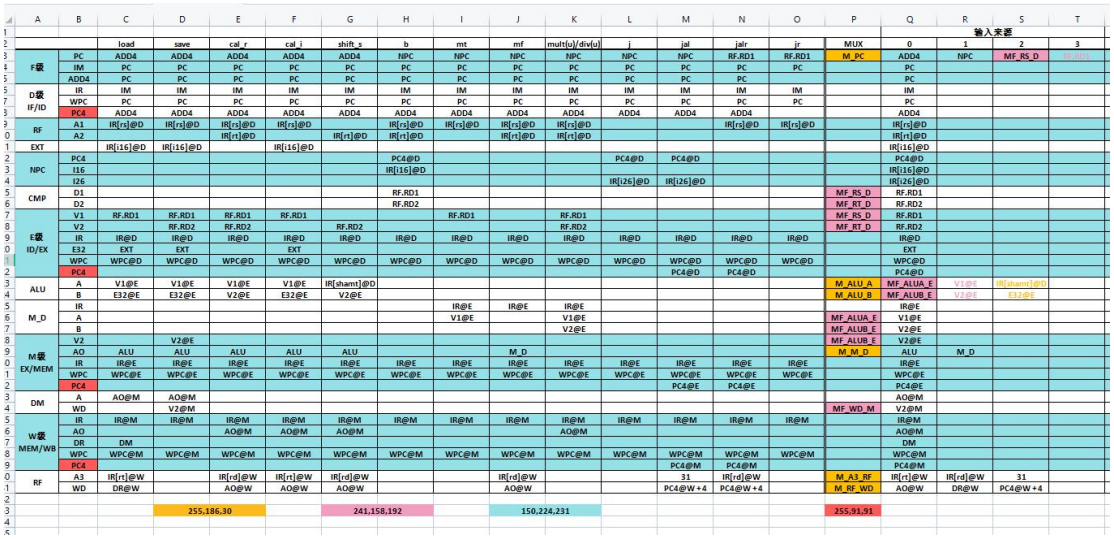
5.为了对抗复杂性你采取了哪些抽象和规范手段？这些手段在译码和处理数据冲突的时候有什么样的特点与帮助？

解：

主要采用了指令分类的方法，P6 相对于 P5 所新增的指令对应的特点都可以归纳为几类，因此对于每条指令而言，只需译码后将其加入对应的分类，构造数据通路，分析转发、暂停条件，并添加一个乘除模块即可。

附 1：

数据通路



																输入来源			
		load	save	cal_r	cal_i	shift_s	b	mt	mf	mult(u)/div(u)	j	jal	jalr	jr	MUX	0	1	2	3
F级	PC	ADD4	ADD4	ADD4	ADD4	ADD4	NPC	NPC	NPC	NPC	NPC	NPC	RF.RD1	RF.RD1	M_PC	ADD4	NPC	MF_RS_D	MF_RT_D
	IM	PC	PC	PC	PC	PC	PC	PC	PC	PC	PC	PC	PC	PC		PC			
	ADD4	PC	PC	PC	PC	PC	PC	PC	PC	PC	PC	PC	PC			PC			
D级 IF/ID	IR	IM	IM	IM	IM	IM	IM	IM	IM	IM	IM	IM	IM	IM		IM			
	WPC	PC	PC	PC	PC	PC	PC	PC	PC	PC	PC	PC	PC	PC		PC			
	PC4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4			ADD4			
RF	A1	IR[rs]@D	IR[rs]@D	IR[rs]@D	IR[rs]@D		IR[rs]@D	IR[rs]@D	IR[rs]@D	IR[rs]@D			IR[rs]@D	IR[rs]@D		IR[rs]@D			
	A2			IR[rt]@D		IR[rt]@D	IR[rt]@D		IR[rt]@D	IR[rt]@D						IR[rt]@D			
EXT		IR[i16]@D	IR[i16]@D		IR[i16]@D											IR[i16]@D			
NPC	PC4						PC4@D				PC4@D	PC4@D				PC4@D			
	I16						IR[i16]@D									IR[i16]@D			
	I26										IR[i26]@D	IR[i26]@D				IR[i26]@D			
CMP	D1						RF.RD1								MF_RS_D	RF.RD1			
	D2						RF.RD2								MF_RT_D	RF.RD2			
E级 ID/EX	V1	RF.RD1	RF.RD1	RF.RD1	RF.RD1			RF.RD1		RF.RD1					MF_RS_D	RF.RD1			
	V2		RF.RD2	RF.RD2		RF.RD2				RF.RD2					MF_RT_D	RF.RD2			
	IR	IR@D	IR@D	IR@D	IR@D	IR@D	IR@D	IR@D	IR@D	IR@D	IR@D	IR@D	IR@D	IR@D		IR@D			
	E32	EXT	EXT		EXT											EXT			
	WPC	WPC@D	WPC@D	WPC@D	WPC@D	WPC@D	WPC@D	WPC@D	WPC@D	WPC@D	WPC@D	WPC@D	WPC@D	WPC@D		WPC@D			
	PC4											PC4@D	PC4@D			PC4@D			
ALU	A	V1@E	V1@E	V1@E	V1@E	IR[shamt]@D									M_ALU_A	MF_ALUA_E	V1@E	IR[shamt]@D	
	B	E32@E	E32@E	V2@E	E32@E	V2@E									M_ALU_B	MF_ALUB_E	V2@E	E32@E	
M_D	IR							IR@E	IR@E	IR@E						IR@E			
	A							V1@E		V1@E					MF_ALUA_E	V1@E			
	B									V2@E					MF_ALUB_E	V2@E			
M级 EX/MEM	V2		V2@E												MF_ALUB_E	V2@E			
	AO	ALU	ALU	ALU	ALU	ALU			M_D						M_M_D	ALU	M_D		
	IR	IR@E	IR@E	IR@E	IR@E	IR@E	IR@E	IR@E	IR@E	IR@E	IR@E	IR@E	IR@E	IR@E		IR@E			
	WPC	WPC@E	WPC@E	WPC@E	WPC@E	WPC@E	WPC@E	WPC@E	WPC@E	WPC@E	WPC@E	WPC@E	WPC@E	WPC@E		WPC@E			
	PC4											PC4@E	PC4@E			PC4@E			
DM	A	AO@M	AO@M													AO@M			
	WD		V2@M												MF_WD_M	V2@M			
W级 MEM/WB	IR	IR@M	IR@M	IR@M	IR@M	IR@M	IR@M	IR@M	IR@M	IR@M	IR@M	IR@M	IR@M	IR@M		IR@M			
	AO			AO@M	AO@M	AO@M				AO@M						AO@M			
	DR	DM														DM			
	WPC	WPC@M	WPC@M	WPC@M	WPC@M	WPC@M	WPC@M	WPC@M	WPC@M	WPC@M	WPC@M	WPC@M	WPC@M	WPC@M		WPC@M			
	PC4											PC4@M	PC4@M			PC4@M			
RF	A3	IR[rt]@W		IR[rd]@W	IR[rt]@W	IR[rd]@W			IR[rd]@W			31	IR[rd]@W		M_A3_RF	IR[rt]@W	IR[rd]@W	31	
	WD	DR@W		AO@W	AO@W	AO@W			AO@W			PC4@W+4	PC4@W+4		M_RF_WD	AO@W	DR@W	PC4@W+4	
			255,186,30			241,158,192			150,224,231						255,91,91				

附 2：

冲突策略矩阵

	Tuse	
指令	rs	rt
addu	1	1
subu	1	1
andi	1	
ori	1	
lw	1	
sw	1	2
beq	0	0
jr	0	
j	-	
jal	-	
lui	-	

		Tnew		
指令	功能部件	E	M	W
addu	ALU	1	0	0
subu	ALU	1	0	0
andi	ALU	1	0	0
ori	ALU	1	0	0
lw	DM	2	1	0
sw	-			
beq	-			
jal	PC	0	0	0
lui	EXT	0	0	0
j	-			
jr	-			

rs策略矩阵

S：暂停

F：转发

Tuse\Tnew	E			M			W		
	ALU	DM	PC	ALU	DM	PC	ALU	DM	PC
	1	2	0	0	1	0	0	0	0
0	S	S	F	F	S	F	F	F	F
1	F	S	F	F	F	F	F	F	F

rt策略矩阵

S：暂停

F：转发

Tuse\Tnew	E			M			W		
	ALU	DM	PC	ALU	DM	PC	ALU	DM	PC
	1	2	0	0	1	0	0	0	0
0	S	S	F	F	S	F	F	F	F
1	F	S	F	F	F	F	F	F	F
2	F	F	F	F	F	F	F	F	F

241,158,192

s_s/s_v/mf -> cal_r

转发：

	0	1	2	3	4	5		ID/EX		EX/MEM				MEM/WB				
	输入0	AO_M	M4	PC8_E	PC8_M	PC8_W		Tnew		Tnew				Tnew				
								jal	jalr	cal_r	cal_i	jal	jalr	cal_r	cal_i	load	jal	jalr
	流水级	源寄存器	涉及指令		MUX	控制信号	输入0	0/31	0/rd	0/rd	O/rt	0/31	0/rd	0/rd	O/rt	O/rt	0/31	0/rd
	D	rs	c_r,c_i,ld,st,b,jr,jalr,s_s		MF_RS_D	F_RS_Dsel	RF.RD1	PC8_E	PC8_E	AO	AO	PC8_M	PC8_M	M4	M4	M4	PC8_W	PC8_W
	D	rt	c_r,st,b,s_s		MF_RT_D	F_RT_Dsel	RF.RD2	PC8_E	PC8_E	AO	AO	PC8_M	PC8_M	M4	M4	M4	PC8_W	PC8_W
	E	rs	c_r,c_i,ld,st,s_s		MF_ALUA_E	F_ALUA_Esel	V1@E			AO	AO	PC8_M	PC8_M	M4	M4	M4	PC8_W	PC8_W
	ALU_E	rt	c_r,st,s_s		MF_ALUB_E	F_ALUB_Esel	V2@E			AO	AO	PC8_M	PC8_M	M4	M4	M4	PC8_W	PC8_W
	M	rt	st		MF_WD_M	F_WD_Msel	V2@M							M4	M4	M4	PC8_W	PC8_W
	DM_M																	

暂停：

	IF/ID当前指令			ID/EX			EX/MEM
	指令类型	源寄存器	Tuse	Tnew			Tnew
				cal_r	cal_i	load	load
				1/rd	1/rt	2/rt	1/rt
	beq	rs/rt	0	暂停	暂停	暂停	暂停
	cal_r	rs/rt	1			暂停	
	cal_i	rs	1			暂停	
	load	rs	1			暂停	
	store	rs	1			暂停	
	store	rt	2				
	jr	rs	0	暂停	暂停	暂停	暂停
	jalr	rs	0	暂停	暂停	暂停	暂停

附 3：

控制器

指令	Opcode	RegWrite	DMop_R	DMop_S	ALU_Control	CMP_type	NPC_sel	EXTop	start	PC_sel	ALU_Asel	ALU_Bsel	M_D_sel	A3_RFsel	RF_WDsel
R	000000	1/...	xxx	000	...	xxx	xxx	xx	0	00/...	0	0	0	01/...	00/...
load	.....	1	...	000	0010	xxx	xxx	01	0	00	0	1	0	00	01
save	.....	0	xxx	...	0010	xxx	xxx	01	0	00	0	1	0	xx	xx
cal_i	.....	1	xxx	000	...	xxx	xxx	...	0	00	0	1	0	00	00
sll_s	.....	1	xxx	000	...	xxx	xxx	xx	0	00	1	0	0	01	00
b	.....	0	xxx	000	xxxx	...	010	xx	0	01	x	x	x	xx	xx
addi	001000	1	xxx	000	0010	xxx	xxx	01	0	00	0	1	0	00	00
j	000010	0	xxx	000	xxxx	xxx	001	xx	0	01	x	x	x	xx	xx
jal	000011	1	xxx	000	xxxx	xxx	001	xx	0	01	x	x	x	10	10
R_jr					xxxx				0	10	x	x			
R_jalr					xxxx				0	10	x	x		01	10
R_mt		0			xxxx				0		x	x	x		
R_mf		1			xxxx				0		x	x	1	01	00
R_mult(u)/R_div		0			xxxx				1		x	x	x		

ALU译码器真值表：R类

Funct	ALUControl
100000(add)	010（加）
100010(sub)	110（减）
100100(and)	000（与）
100101(or)	001（或）
101010(slt)	111（小于置位）
100001(addu)	010
100011(subu)	110
001001(jr)	

0000	0001	0010	0011	0100	0101	0110	0111
与	或	加	异或	或非	小于置1	减	符号）小于置1
1000	1001	1010	1011				
lui	逻辑左移	逻辑右移	算数右移				

附 4：

多路选择器控制信号

				优先级		
				低	中	高
功能MUX	控制信号	转发MUX	控制信号	0	1	2
M_PC	PC_sel	MF_RS_D	F_RS_Dsel	RF.RD1	AO@M	M_RF_WD
M_ALU_A	ALU_Asel	MF_RT_D	F_RT_Dsel	RF.RD2	AO@M	M_RF_WD
M_ALU_B	ALU_Bsel	MF_ALUA_E	F_ALUA_Esel	V1@E	AO@M	M_RF_WD
M_M_D	M_Dsel	MF_ALUB_E	F_ALUB_Esel	V2@E	AO@M	M_RF_WD
M_A3_RF	A3_RF_sel	MF_WD_M	F_WD_Msel	V2@M	M_RF_WD	
M_RF_WD	RF_WDsel					