实验六 54 条指令 CPU 设计

1.实验介绍

在本次实验中，我们将使用 Verilog HDL 语言实现 54 条 MIPS 指令的 CPU 的设计和仿真，设计的 CPU 可以是单周期的，也可以是多周期。

2.实验目标

- 深入了解 CPU 的原理。
- 画出实现 54 条指令的 CPU 的通路图。
- 学习使用 Verilog HDL 语言设计实现 54 条指令的 CPU。

3.实验原理

1）54 指令单周期 CPU
需要实现的 54 条 MIPS 指令，见表 8.6.1

表 8.6.1 54 条 MIPS 指令集

指令	指令说明	指令格式	OP 31-26	FUNCT 5-0	指令码 16 进制
addi	加立即数	addi rt, rs, immediate	001000		20000000
addiu	加立即数（无符号）	addiu rd, rs, immediate	001001		24000000
andi	立即数与	andi rt, rs, immediate	001100		30000000
ori	或立即数	ori rt, rs, immediate	001101		34000000
sltiu	小于立即数置 1（无符号）	sltiu rt, rs, immediate	001011		2C000000
lui	立即数加载高位	lui rt, immediate	001111		3C000000
xori	异或（立即数）	xori rt, rs, immediate	001110		38000000
slti	小于置 1（立即数）	slti rt, rs, immediate	001010		28000000
addu	加（无符号）	addu rd, rs, rt	000000	100001	00000021
and	与	and rd, rs, rt	000000	100100	00000024
beq	相等时分支	beq rs, rt, offset	000100		10000000
bne	不等时分支	bne rs, rt, offset	000101		14000000
j	跳转	j target	000010		08000000
jal	跳转并链接	jal target	000011		0C000000
jr	跳转至寄存器所指地址	jr rs	000000	001000	00000009
lw	取字	lw rt, offset(base)	100011		8C000000
xor	异或	xor rd, rs, rt	000000	100110	00000026
nor	或非	nor rd, rs, rt	000000	100111	00000027
or	或	or rd, rs, rt	000000	100101	00000025

sll	逻辑左移	sll rd, rt, sa	000000	000000	00000000
sllv	逻辑左移（位数可变）	sllv rd, rt, rs	000000	000100	00000004
sltu	小于置 1（无符号）	sltu rd, rs, rt	000000	101011	0000002B
sra	算数右移	sra rd, rt, sa	000000	000011	00000003
srl	逻辑右移	srl rd, rt, sa	000000	000010	00000002
subu	减（无符号）	sub rd, rs, rt	000000	100010	00000022
sw	存字	sw rt, offset(base)	101011		AC000000
add	加	add rd, rs, rt	000000	100000	00000020
sub	减	sub rd, rs, rt	000000	100010	00000022
slt	小于置 1	slt rd, rs, rt	000000	101010	0000002A
srlv	逻辑右移（位数可变）	srlv rd, rt, rs	000000	000110	00000006
srav	算数右移（位数可变）	srav rd, rt, rs	000000	000111	00000007
clz	前导零计数	clz rd, rs	011100	100000	70000020
divu	除（无符号）	divu rs, rt	000000	011011	0000001B
eret	异常返回	eret	010000	011000	42000018
jalr	跳转至寄存器所指地址，返回地址保存在	jalr rs	000000	001001	00000008
lb	取字节	lb rt, offset(base)	100000		80000000
lbu	取字节（无符号）	lbu rt, offset(base)	100100		90000000
lhu	取半字（无符号）	lhu rt, offset(base)	100101		94000000
sb	存字节	sb rt, offset(base)	101000		A0000000
sh	存半字	sh rt, offset(base)	101001		A4000000
lh	取半字	lh rt, offset(base)	100001		84000000
mfc0	读 CP0 寄存器	mfc0 rt, rd	010000	000000	40000000
mfhi	读 Hi 寄存器	mfhi rd	000000	010000	00000010
mflo	读 Lo 寄存器	mflo rd	000000	010010	00000012
mtc0	写 CP0 寄存器	mtc0 rt, rd	010000	000000	40800000
mthi	写 Hi 寄存器	mthi rd	000000	010001	00000011
mtlo	写 Lo 寄存器	mtlo rd	000000	010011	00000013
mul	乘	mul rd, rs, rt	011100	000010	70000002
multu	乘（无符号）	multu rs, rt	000000	011001	00000019
syscall	系统调用	syscall	000000	001100	0000000C
teq	相等异常	teq rs, rt	000000	110100	00000034
bgez	大于等于 0 时分支	bgez rs, offset	000001		04010000
break	断点	break	000000	001101	0000000D
div	除	div rs, rt	000000	011010	0000001A

54 条指令 CPU 数据通路设计的方法和 31 指令的一样：

- 阅读每条指令，对每条指令所需执行的功能与过程都有充分的了解
- 确定每条指令在执行过程中所用到的部件
- 使用表格列出指令所用部件，并在表格中填入每个部件的数据输入来源
- 根据表格所涉及部件和部件的数据输入来源，画出整个数据通路

54 条指令在 31 条的基础上添加了乘除法运算、对 Lo/Hi 寄存器的读写、内存半字和字节的存取操作、CP0 的异常处理指令和 CP0 寄存器的读写，以及一些跳转指令。

乘、除法器和 CP0 模块在前面的部分已经有所介绍，在 CPU 中仅需处理相应指令的控制信号和输入输出引脚，剩余的主要是添加对内存块的读写控制。在 CPU 通路中需要加入乘、除法器模块和 CP0 模块。

指令的测试和 31 条指令 CPU 的测试方法类似，对 CP0 模块的测试需要自行编写测试用例，要对 CP0 寄存器的读写功能、异常发生时的跳转功能和异常返回等环节进行测试，主要验证几个关键寄存器值的正确写入和控制信号的判断处理。实验中异常的入口地址为 0x4，可在入口处添加跳转指令，跳入统一的异常处理程序，再判断异常号然后进入相应的处理入口。

2）54 指令多周期 CPU

多周期 CPU 的中心思想是把一条指令的执行分成若干个小周期，根据每条指令的复杂程度，使用不同数量的小周期去执行，许多个小周期加在一起相当于单周期 CPU 中的一个周期。

在我们实现的 54 条指令中，最复杂的指令之一是 lw rt，offset(rs)，它需要 5 个周期，其整个执行的过程为：

a） 根据 PC 取指令，并把 PC 加 4；
b） 对指令译码并读出 rs 寄存器的内容；
c） rs 寄存器的内容与指令中的偏移量 offset 相加，计算得到存储器地址；
d） 使用计算好的地址访问存储器，从中读出一个 32 位的数据；
e） 把该数据写入寄存器堆中的 rt 寄存器。

而最简单的指令是 j address，两个周期就行了：

a） 根据 PC 取指令，并把 PC 加 4；
b） 指令中的 address 左移两位与 PC 的高 4 位拼接起来，写入 PC。

ALU 计算类型的指令需要 4 个周期：

a） 根据 PC 取指令，并把 PC 加 4；
b） 读出 rs 和 rt 两个寄存器的内容；
c） 由 ALU 完成对两个寄存器数据（或一个寄存器数据和一个立即数）的计算；
d） 把计算结果写入寄存器堆中的 rd（或 rt）寄存器。

转移类型的指令，例如 bne，需要 3 个周期：

a） 根据 PC 取指令，并把 PC 加 4；
b） 读出 rs 和 rt 两个寄存器的数据并锁存，同时 ALU 计算转移地址并锁存；
c） 由 ALU 比较两个寄存器数据，并决定是否把转移地址写入 PC。

3）8 条指令多周期 CPU 设计

下面以 8 条指令多周期 CPU 的设计过程为例具体说明。

- 数据通路设计

数据通路的设计过程与单周期的数据通路设计过程一样，要注意的是，CPU 中的某些资源可以复用，比如 ALU，既可以完成算术和逻辑运算，还可以用于 PC 的增值运算。我们可以使用一个指令和数据公用的存储器，而不需分为两个独立的指令和数据存储器。此外，我们增加了一个指令寄存器 IR 和一个暂存器 Z，数据通路入图 8.6.3 所示。

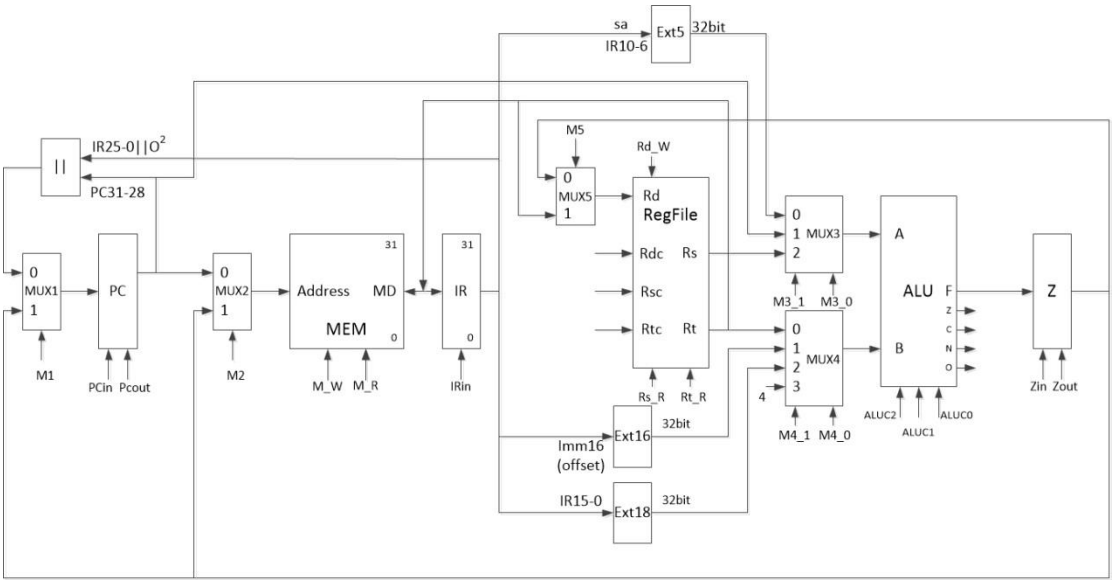

图 8.6.3 8 条指令的多周期 CPU 数据通路

- 用有限状态机实现多周期 CPU 的控制部件

我们可以用时序电路来实现多周期CPU的控制部件，主要工作是确定状态转移图。状态转移图不是唯一的，只要能实现各条指令所经过的周期即可。

- 多周期 CPU 的控制部件的状态转移图

图 8.6.4 给出的是一种状态转移图。以 8 条基础指令为例，这里使用了最少的状态数，从图中可以看出，跳转指令 j 用两个周期，条件转移指令 beq 用三个周期，lw 指令用五个周期，其余指令 addu、subu、ori、sll 均用四个周期。图中的五个状态分别有五个名字，而且我们为每个状态分别指定了一个唯一的 3 位二进制数，不允许两个不同的状态有相同的二进制数。

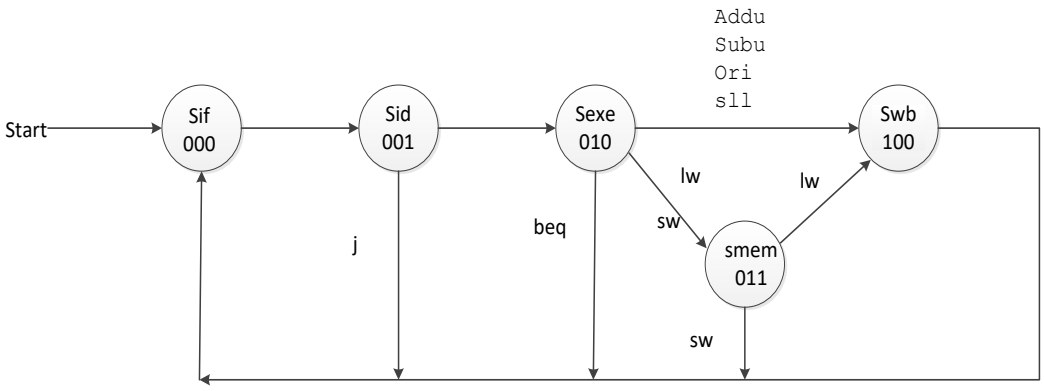

图 8.6.4 8 条指令多周期 CPU 控制部件的状态转移图

与单周期一样，控制单元需要根据指令的 opcode 字段和 funct 字段计算控制信号，在多周期 CPU 中，要在不同的周期中产生相应的控制信号。图 8.6.5 为 8 条指令多周期 CPU 的控制信号状态转移图，控制信号包括选择信号和使能信号，为了使状态转移图可读，只列出了相关的控制信号，只有当选择信号的值重要时才列出，使能信号仅在有效时列出，否则为 0。

在取 IF 周期，需要取指令并将 PC+4；在 ID 周期，若为 j 指令，则将跳转地址写入 PC，若为其它指令，读出 rs 和 rt 两个寄存器的数据；在 EXE 周期，j 指令已经执行完毕，若为 beq 指令，由 ALU 比较两个寄存器数据，并决定是否把转移地址写入 PC，若为 addu、subu、ori、sll 指令，由 ALU 完成对两个寄存器数据（或一个寄存器数据和一个立即数）的计算，若为 sw 或 lw 指令，将 rs 寄存器的内容与偏移量 offset 相加，计算得到存储器地址；在 MEM 周期，只有 lw 和 sw 指令进行操作，使用计算好的地址访问存储器，从中读出或写入数据；最后，在 WB 周期把 ALU 的计算结果或存储器取来的数据写入寄存器堆。

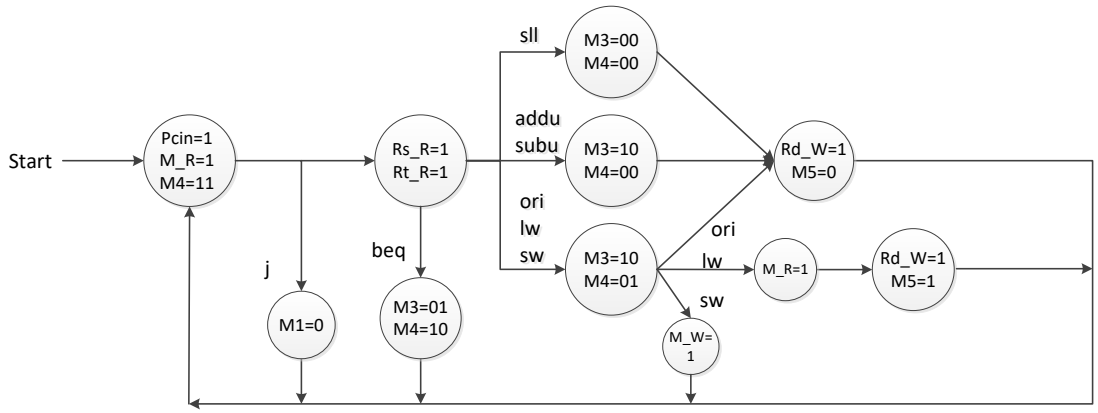

图 8.6.5 8 条指令多周期 CPU 控制信号状态转移图

- 多周期 CPU 的控制部件的总体结构

图 8.6.6 所示的是多周期 CPU 控制部件的电路结构图。在这个时序电路中，D 触发器保存当前状态，其余两个模块是组合电路，分别产生表示下一状态的 3 位二进制数和控制信号。

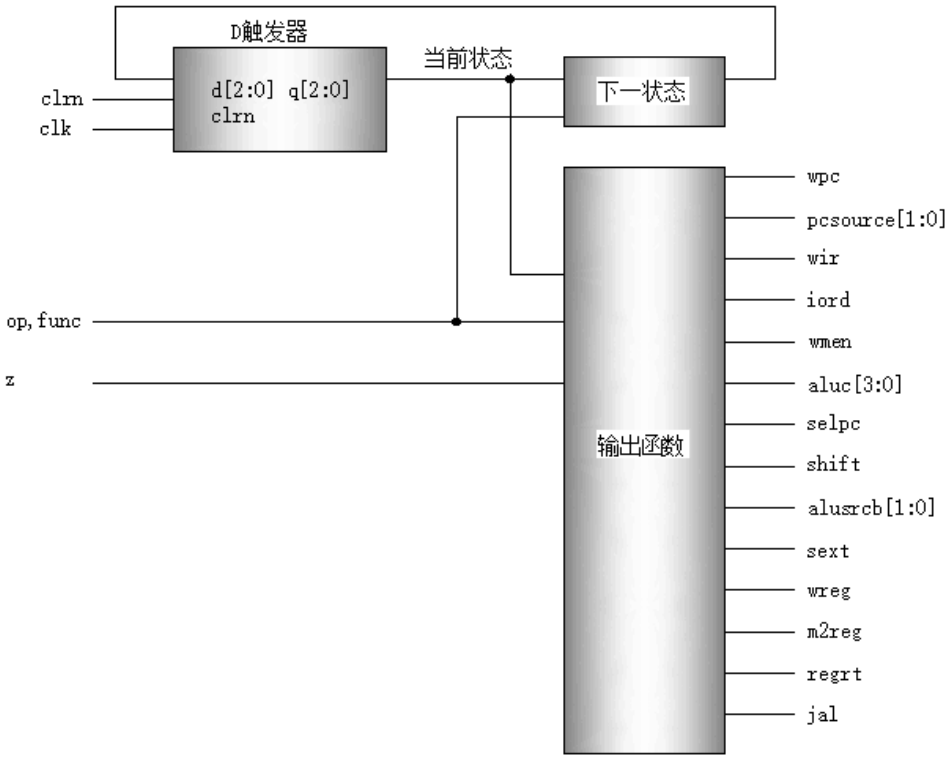

图 8.6.6 多周期 CPU 控制部件的电路结构图

图 8.6.6 中的“下一状态”模块是组合电路，用于产生下一状态的信息，我们用的 d[2:0]是当前状态。根据图 8.6.4 所示的控制部件的状态转移图，我们有如表 8.6.2 所示的真值表。我们的目的是求出的 d[2:0]每一位的逻辑表达式。

由真值表，我们得到如下的逻辑表达式，其中的 sif、sid、sexe、smem 和 swb 表示状态，可以认为是中间变量。

```
sif = q[2]' q[1]' q[0]';
sid = q[2]' q[1]' q[0];
sexe = q[2]' q[1] q[0]';
smem = q[2]' q[1] q[0];
swb = q[2] q[1]' q[0]';
d[0] = sif+sexe(i_lw+i_sw);
d[1] = sid(i_j)+sexe(i_lw+i_sw);
d[2] = sexe(i_beq+i_lw+i_sw)+smem i_lw;
```

表 8.6.2 下一状态函数的真值表

当前状态		输入	下一状态	
状态	q[2:0]	op[5:0]	状态	d[2:0]
sif	000	x	sid	001

sid	001	i_j	sif	000
		others	sexe	010
sexe	010	i_beq	sif	000
		i_lw	smem	011
		i_sw	smem	011
		others	swb	100
smem	011	i_lw	swb	100
		i_sw	sif	000
swb	100	x	sif	000

4.实验步骤

1. 新建 Vivado 工程，可在 31 指令 CPU 基础上编写各个模块。
2. 用 ModelSim 前仿真逐条测试所有指令。
3. 用 ModelSim 进行后仿真测试。
4. 配置 XDC 文件，综合下板，并观察实验现象。
5. 按照要求书写实验报告。