同济大学
计算机科学与技术系

计算机组成原理实验报告

学　　号 **1551445**

姓　　名 童佳燕

专　　业 计算机科学与技术

授课老师 郝泳涛

日　　期 **2018 年 6 月 4 日**

一、 实验内容：54 条指令 CPU 设计

在本次实验中，我们要求要求画出 54 条指令的 CPU 的通路图，并使用 Verilog HDL 语言实现 54 条 MIPS 指令的 CPU 的设计和仿真。

需要实现的 54 条 MIPS 指令

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

二、 实验步骤

本实验完成的是 54 条 MIPS 指令单周期 CPU，实验步骤如下：

1. 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
2. 确定每条指令在执行过程中所用到的部件以及数据的流向
3. 使用表格列出指令所用部件，在表格中填入每个部件的数据输入来源

   表格如下图

表 3.1.指令 ADD-JR 相关部件数据来源表

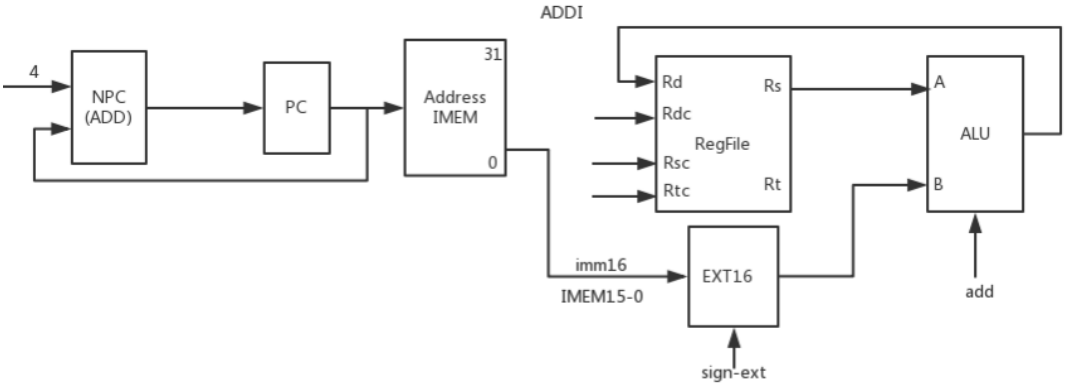
	PC	NPC	IM	RegFile			ALU		EXT5	EXT16	EXT18	DMEM		ADD		||	
指令				Rd		R31	A	B				Addr	Data	A	B	A	B
ADD	NPC	PC	PC	ALU			Rs	Rt									
ADDU	NPC	PC	PC	ALU			Rs	Rt									
SUB	NPC	PC	PC	ALU			Rs	Rt									
SUBU	NPC	PC	PC	ALU			Rs	Rt									
AND	NPC	PC	PC	ALU			Rs	Rt									
OR	NPC	PC	PC	ALU			Rs	Rt									
XOR	NPC	PC	PC	ALU			Rs	Rt									
NOR	NPC	PC	PC	ALU			Rs	Rt									
SLT	NPC	PC	PC	ALU			Rs	Rt									
SLTU	NPC	PC	PC	ALU			Rs	Rt									
SRAV	NPC	PC	PC	ALU			Rs	Rt									
SLLV	NPC	PC	PC	ALU			Rs	Rt									
SRLV	NPC	PC	PC	ALU			Rs	Rt									
SLL	NPC	PC	PC	ALU			EXT5	Rt	sa								
SRL	NPC	PC	PC	ALU			EXT5	Rt	sa								
SRA	NPC	PC	PC	ALU			EXT5	Rt	sa								
ADDI	NPC	PC	PC	ALU			Rs	EXT16		imm16							
ADDIU	NPC	PC	PC	ALU			Rs	EXT16		imm16							
ANDI	NPC	PC	PC	ALU			Rs	EXT16		imm16							
ORI	NPC	PC	PC	ALU			Rs	EXT16		imm16							
XORI	NPC	PC	PC	ALU			Rs	EXT16		imm16							
SLTI	NPC	PC	PC	ALU			Rs	EXT16		imm16							
SLTIU	NPC	PC	PC	ALU			Rs	EXT16		imm16							
LUI	NPC	PC	PC	ALU				EXT16		imm16							
LW	NPC	PC	PC	DMEM(Data)			Rs	EXT16		offset		ALU					
SW	NPC	PC	PC				Rs	EXT16		offset		ALU	Rt				
BEQ	ADD	PC	PC				Rs	Rt			offset			EXT18	NPC		
BNE	ADD	PC	PC				Rs	Rt			offset			EXT18	NPC		
J	||	PC	PC													PC[31:28]	offset||00
JAL	||	PC	PC			PC									PC[31:28]	offset||00		
JR	Rs	PC	PC														

表 3.2.指令 ADD-JR 相关部件数据来源表

	PC	NPC	IM	RegFile	ALU		EXT5	EXT16	EXT18	DMEM		ADD		DivMult		HI	LO	CPRegfile
指令				Rd	A	B				Addr	Data	A	B	A	B			
BGEZ	ADD	PC	PC		Rs	0			offset			EXT18	NPC					
BREAK																		
ERET																		
SYSCALL																		
TEQ					Rs	Rt												
DIV	NPC	PC	PC											Rs	Rt	r	q	
DIVU	NPC	PC	PC											Rs	Rt	r	q	
MULT	NPC	PC	PC											Rs	Rt	r	q	
MULTU	NPC	PC	PC											Rs	Rt	r	q	
JALR	Rs	PC	PC	PC														
LB	NPC	PC	PC	DMEM(Data)	EXT5	EXT16	base	offset		ALU								
LBU	NPC	PC	PC	DMEM(Data)	EXT5	EXT16	base	offset		ALU								
LH	NPC	PC	PC	DMEM(Data)	EXT5	EXT16	base	offset		ALU								
LHU	NPC	PC	PC	DMEM(Data)	EXT5	EXT16	base	offset		ALU								
SB	NPC	PC	PC		EXT5	EXT16	base	offset		ALU	Rt							
SH	NPC	PC	PC		EXT5	EXT16	base	offset		ALU	Rt							
MFC0	NPC	PC	PC	CPRegfile														IM[15:11]
MFLO	NPC	PC	PC	LO														
MTC0	NPC	PC	PC															Regfile[rt
MTHI																		
MTLO																		
MFHI																		
CLZ	NPC	PC	PC	CountZero														

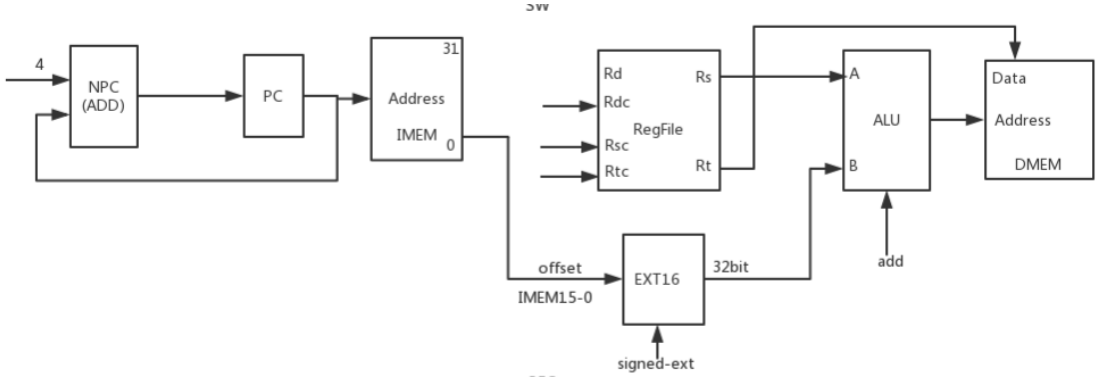
4. 根据表格所涉及部件和部件的数据输入来源，绘制每个指令的数据流图，并在绘制图的过程中记录所需控制信号。由于 54 条指令设计大量指令数据流图，报告中仅呈现部分代表性指令，详细指令数据流图见附件 1.1,1.2,1.3.

图 4.1 ADDI 指令数据流图



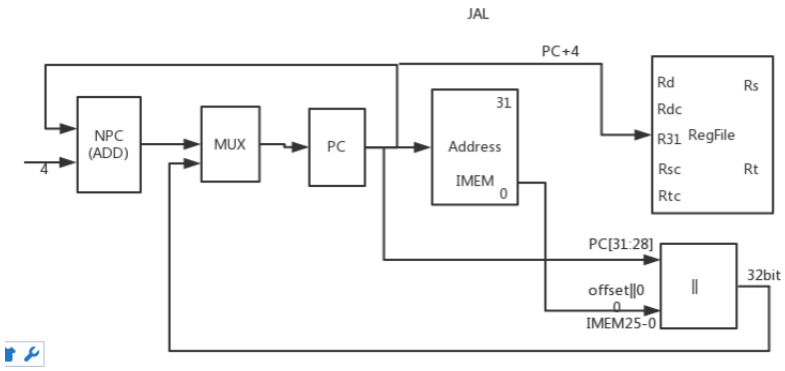
说明：NPC 部件执行 PC+4 操作，准备下一条指令地址；PC 寄存器在时钟上升沿到来时作为 addr 读取 IMEM 中的 32 位指令；根据指令中对应位的数据，在时钟下降沿到来时取出 Rs，与 16 位 imm 扩展后的数据一起送入 ALU 部件执行加法；最后将结果送到 Regfile 的 rd 中。

图 4.2 SW 指令数据流图



说明：NPC 部件执行 PC+4 操作，准备下一条指令地址；PC 寄存器在时钟上升沿到来时作为 addr 读取 IMEM 中的 32 位指令；根据指令中对应位的数据，在时钟下降沿到来时取出 Rs，与 16 位 imm 扩展后的数据一起送入 ALU 部件执行加法，作为数据存储器写入数据的地址；DMEM 部件根据 ALU 的结果是写入数据，完成 SW 的全部操作。

图 4.3 JAL 指令数据流图



说明：JAL 指令在 J 的基础上将返回地址保存在 Regfile 的 31 号寄存器中。下一条指令，在之前的指令的基础上，会有两个数据来源：PC+4 会让 PC[31:28]||offset||00。NPC 部件执行 PC+4 操作，准备 PC+4 地址，||部件执行数据连接操作，准备 PC[31:28]||offset||00 地址。同时 PC+4 作为返回地址送到 Regfile 的 31 号寄存器进行存储

表 4.1 54 条单周期 CPU 控制信号表

	ADDU	ADD	SUB	SUBU	AND	OR	XOR	NOR
PC_CLK	1	1	1	1	1	1	1	1
IM_ENA	1	1	1	1	1	1	1	1
Rsc4-0	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21
Rtc4-0	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16
Rdc4-0	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11
CPR_Rsc4-0								
CPR_Rtc4-0								
CPR_Rdc4-0								
RF_W	1	1	1	1	1	1	1	1
CPR_W	0	0	0	0	0	0	0	0
M0	0	0	0	0	0	0	0	0
M1	0	0	0	0	0	0	0	0
M2	0	0	0	0	0	0	0	0
M3	1	1	1	1	1	1	1	1
M4	0	0	0	0	0	0	0	0
M5	0	0	0	0	0	0	0	0
M7	1	1	1	1	1	1	1	1
M10	x	x	x	x	x	x	x	x
M11	x	x	x	x	x	x	x	x
PC_SOURCE[1]	0	0	0	0	0	0	0	0
PC_SOURCE[0]	0	0	0	0	0	0	0	0
ALUC3	0	0	0	0	0	0	0	0
ALUC2	0	0	0	0	1	1	1	1
ALUC1	0	1	1	0	0	0	1	1
ALUC0	0	0	1	1	0	1	0	1
DM_W	0	0	0	0	0	0	0	0
SEXT	x	x	x	x	x	x	x	x
IS_JAL	0	0	0	0	0	0	0	0
DIVMUL_sel[1]	x	x	x	x	x	x	x	x
DIVMUL_sel[0]	x	x	x	x	x	x	x	x
BH_sel[2]	x	x	x	x	x	x	x	x
BH_sel[1]	x	x	x	x	x	x	x	x
BH_sel[0]	x	x	x	x	x	x	x	x
DM_W_BH[1]	x	x	x	x	x	x	x	x
DM_W_BH[0]	x	x	x	x	x	x	x	x
LO_W	0	0	0	0	0	0	0	0
HI_W	0	0	0	0	0	0	0	0

表 4.2 54 条单周期 CPU 控制信号表

	SLT	SLTU	SRAV	SLLV	SRLV	SLL	SRL	SRA
PC_CLK	1	1	1	1	1	1	1	1
IM_ENA	1	1	1	1	1	1	1	1
Rsc4-0	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21			
Rtc4-0	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16
Rdc4-0	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11
CPR_Rsc4-0								
CPR_Rtc4-0								
CPR_Rdc4-0								
RF_W	1	1	1	1	1	1	1	1
CPR_W	0	0	0	0	0	0	0	0
M0	0	0	0	0	0	0	0	0
M1	0	0	0	0	0	0	0	0
M2	0	0	0	0	0	0	0	0
M3	1	1	1	1	1	0	0	0
M4	0	0	0	0	0	0	0	0
M5	0	0	0	0	0	0	0	0
M7	1	1	1	1	1	1	1	1
M10	x	x	x	x	x	x	x	x
M11	x	x	x	x	x	x	x	x
C_SOURCE[1	0	0	0	0	0	0	0	0
C_SOURCE[0	0	0	0	0	0	0	0	0
ALUC3	1	1	1	1	1	1	1	1
ALUC2	0	0	1	1	1	1	1	1
ALUC1	1	1	0	1	0	1	0	0
ALUC0	1	0	0	x	1	x	1	0
DM_W	0	0	0	0	0	0	0	0
SEXT	x	x	x	x	x	0	0	0
IS_JAL	0	0	0	0	0	0	0	0
VMUL_sel[	x	x	x	x	x	x	x	x
VMUL_sel[0	x	x	x	x	x	x	x	x
BH_sel[2]	x	x	x	x	x	x	x	x
BH_sel[1]	x	x	x	x	x	x	x	x
BH_sel[0]	x	x	x	x	x	x	x	x
DM_W_BH[1]	x	x	x	x	x	x	x	x
DM_W_BH[0]	x	x	x	x	x	x	x	x
LO_W	0	0	0	0	0	0	0	0
HI_W	0	0	0	0	0	0	0	0

表 4.3 54 条单周期 CPU 控制信号表

	ADDI	ADDIU	ANDI	ORI	XORI	LUI	SLTI	SLTIU	LW	SW
PC_CLK	1	1	1	1	1	1	1	1	1	1
IM_ENA	1	1	1	1	1	1	1	1	1	1
Rsc4-0	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21		IM25-21	IM25-21	IM25-21	IM25-21
Rtc4-0										IM20-16
Rdc4-0	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	
CPR_Rsc4-0										
CPR_Rtc4-0										
CPR_Rdc4-0										
RF_W	1	1	1	1	1	1	1	1	1	0
CPR_W	0	0	0	0	0	0	0	0	0	0
M0	0	0	0	0	0	0	0	0	0	x
M1	0	0	0	0	0	0	0	0	0	x
M2	0	0	0	0	0	0	0	0	1	x
M3	1	1	1	1	1	x	1	1	1	1
M4	1	1	1	1	1	1	1	1	1	1
M5	0	0	0	0	0	0	0	0	0	0
M7	0	0	0	0	0	0	0	0	0	x
M10	x	x	x	x	x	x	x	x	x	x
M11	x	x	x	x	x	x	x	x	x	x
C_SOURCE[1	0	0	0	0	0	0	0	0	0	0
C_SOURCE[0	0	0	0	0	0	0	0	0	0	0
ALUC3	0	0	0	0	0	1	1	1	0	0
ALUC2	0	0	1	1	1	0	0	0	0	0
ALUC1	1	0	0	0	1	0	1	1	1	1
ALUC0	0	0	0	1	0	x	1	0	0	0
DM_W	0	0	0	0	0	0	0	0	0	1
SEXT	1	1	0	0	0	1	1	1	1	1
IS_JAL	0	0	0	0	0	0	0	0	0	0
VMUL_sel[	x	x	x	x	x	x	x	x	x	x
VMUL_sel[0	x	x	x	x	x	x	x	x	x	x
BH_sel[2]	x	x	x	x	x	x	x	x	0	x
BH_sel[1]	x	x	x	x	x	x	x	x	x	x
BH_sel[0]	x	x	x	x	x	x	x	x	x	x
DM_W_BH[1]	x	x	x	x	x	x	x	x	x	0
DM_W_BH[0]	x	x	x	x	x	x	x	x	x	x
LO_W	0	0	0	0	0	0	0	0	0	0
HI_W	0	0	0	0	0	0	0	0	0	0

表 4.4 54 条单周期 CPU 控制信号表

	BEQ	BNE	J	JAL	JR	BGEZ	DIV	DIVU	MULT	MULTU
PC_CLK	1	1	1	1	1	1	1	1	1	1
IM_ENA	1	1	1	1	1	1	1	1	1	1
Rsc4-0	IM25-21	IM25-21			IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21
Rtc4-0	IM20-16	IM20-16					IM20-16	IM20-16	IM20-16	IM20-16
Rdc4-0										
PR_Rsc4-0										
PR_Rtc4-0										
PR_Rdc4-0										
RF_W	0	0	0	1	0	0	0	0	1	1
CPR_W	0	0	0	0	0	0	0	0	0	0
M0	x	x	x	0	x	x	x	x	1	1
M1	x	x	x	1	x	x	x	x	1	1
M2	x	x	x	0	x	x	x	x	1	1
M3	1	1	x	x	1	1	x	x	x	x
M4	0	0	x	x	0	0	x	x	x	x
M5	0	0	x	x	0	1	x	x	x	x
M7	x	x	x	x	x	x	x	x	1	1
M10	x	x	x	x	x	x	0	0	0	0
M11	x	x	x	x	x	x	1	1	1	1
_SOURCE[1	0	0	1	1	1	0	0	0	0	0
_SOURCE[0	1	1	0	0	1	1	0	0	0	0
ALUC3	0	0	x	x	x	0	x	x	x	x
ALUC2	0	0	x	x	x	0	x	x	x	x
ALUC1	1	1	x	x	x	1	x	x	x	x
ALUC0	1	1	x	x	x	1	x	x	x	x
DM_W	0	0	0	0	0	0	0	0	0	0
SEXT	1	1	0	0	0	x	x	x	x	x
IS_JAL	0	0	0	1	0	0	0	0	0	0
VMUL_sel[1	x	x	x	x	x	x	0	0	1	1
VMUL_sel[0	x	x	x	x	x	x	0	1	0	1
BH_sel[2]	x	x	x	x	x	x	x	x	x	x
BH_sel[1]	x	x	x	x	x	x	x	x	x	x
BH_sel[0]	x	x	x	x	x	x	x	x	x	x
DM_W_BH[1]	x	x	x	x	x	x	x	x	x	x
DM_W_BH[0]	x	x	x	x	x	x	x	x	x	x
LO_W	0	0	0	0	0	0	1	1	1	1
HI_W	0	0	0	0	0	0	1	1	1	1

表 4.5 54 条单周期 CPU 控制信号表

	JALR	LB	LBU	LH	LHU	SB	SH	CLZ	MFC0	MFLO	MTC0	MTHI	MTLO	MFHI
PC_CLK	1	1	1	1	1	1	1	1	1	1	1	1	1	1
IM_ENA	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Rsc4-0	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21				IM25-21	IM25-21	
Rtc4-0						IM20-16	IM20-16				IM20-16			
Rdc4-0	IM15-11	IM20-16	IM20-16	IM20-16	IM20-16			IM15-11	IM20-16	IM15-11				IM15-11
PR_Rsc4-0														
PR_Rtc4-0									IM15-11					
PR_Rdc4-0											IM15-11			
RF_W	1	1	1	1	1	0	0	1	1	1	0	0	0	1
CPR_W	0	0	0	0	0	0	0	0	0	0	1	0	0	0
M0	0	0	0	0	0	x	x	0	1	1	x	x	x	1
M1	1	0	0	0	0	x	x	1	1	0	x	x	x	0
M2	0	1	1	1	1	x	x	1	0	0	x	x	x	1
M3	x	1	1	1	1	1	1	x	x	x	x	x	x	x
M4	x	1	1	1	1	1	1	x	x	x	x	x	x	x
M5	x	0	0	0	0	0	0	x	x	x	x	x	x	x
M7	1	0	0	0	0	x	x	1	0	1	x	x	x	1
M10	x	x	x	x	x	x	x	x	x	x	x	1	x	x
M11	x	x	x	x	x	x	x	x	x	x	x	x	0	x
_SOURCE[1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
_SOURCE[0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
ALUC3	x	0	0	0	0	0	0	x	x	x	x	x	x	x
ALUC2	x	0	0	0	0	0	0	x	x	x	x	x	x	x
ALUC1	x	1	1	1	1	1	1	x	x	x	x	x	x	x
ALUC0	x	0	0	0	0	0	0	x	x	x	x	x	x	x
DM_W	0	0	0	0	0	1	1	0	0	0	0	0	0	0
SEXT	x	1	1	1	1	1	1	x	x	x	x	x	x	x
IS_JAL	0	0	0	0	0	0	0	0	0	0	0	0	0	0
VMUL_sel[1	x	x	x	x	x	x	x	x	x	x	x	x	x	x
VMUL_sel[0	x	x	x	x	x	x	x	x	x	x	x	x	x	x
BH_sel[2]	x	1	1	1	1	x	x	x	x	x	x	x	x	x
BH_sel[1]	x	0	0	1	1	x	x	x	x	x	x	x	x	x
BH_sel[0]	x	0	1	1	1	x	x	x	x	x	x	x	x	x
DM_W_BH[1]	x	x	x	x	x	1	1	x	x	x	x	x	x	x
DM_W_BH[0]	x	x	x	x	x	0	1	x	x	x	x	x	x	x
LO_W	0	0	0	0	0	0	0	0	0	0	0	0	1	0
HI_W	0	0	0	0	0	0	0	0	0	0	0	1	0	0

表 4.6 ALU 部件选择信号说明

	ALU3	ALU2	ALU1	ALU0
ADDU	0	0	0	0
ADD	0	0	1	0
SUBU	0	0	0	1
SUB	0	0	1	1
AND	0	1	0	0
OR	0	1	0	1
XOR	0	1	1	0
NOR	0	1	1	1
LUI	1	0	0	x
SLT	1	0	1	1
SLTU	1	0	1	0
SRA	1	1	0	0
SLL	1	1	1	x
SRL	1	1	0	1

表 4.7 rf_wdata 数据来源选择信号说明

	M0	M1	M2
ALU	0	0	0
DMEM	0	0	1
NPC	0	1	0
COUTZERO	0	1	1
LO	1	0	0
HI	1	0	1
CPO	1	1	0
DIVMUL	1	1	1

表 4.8 乘除法器功能选择信号说明

	DIMU_SEL[1]	DIMU_SEL[0]
DIV	0	0
DIVU	0	1
MUL	1	0
MULU	1	1

表 4.9 LH/LB/LHU/LBU 指令选择信号说明

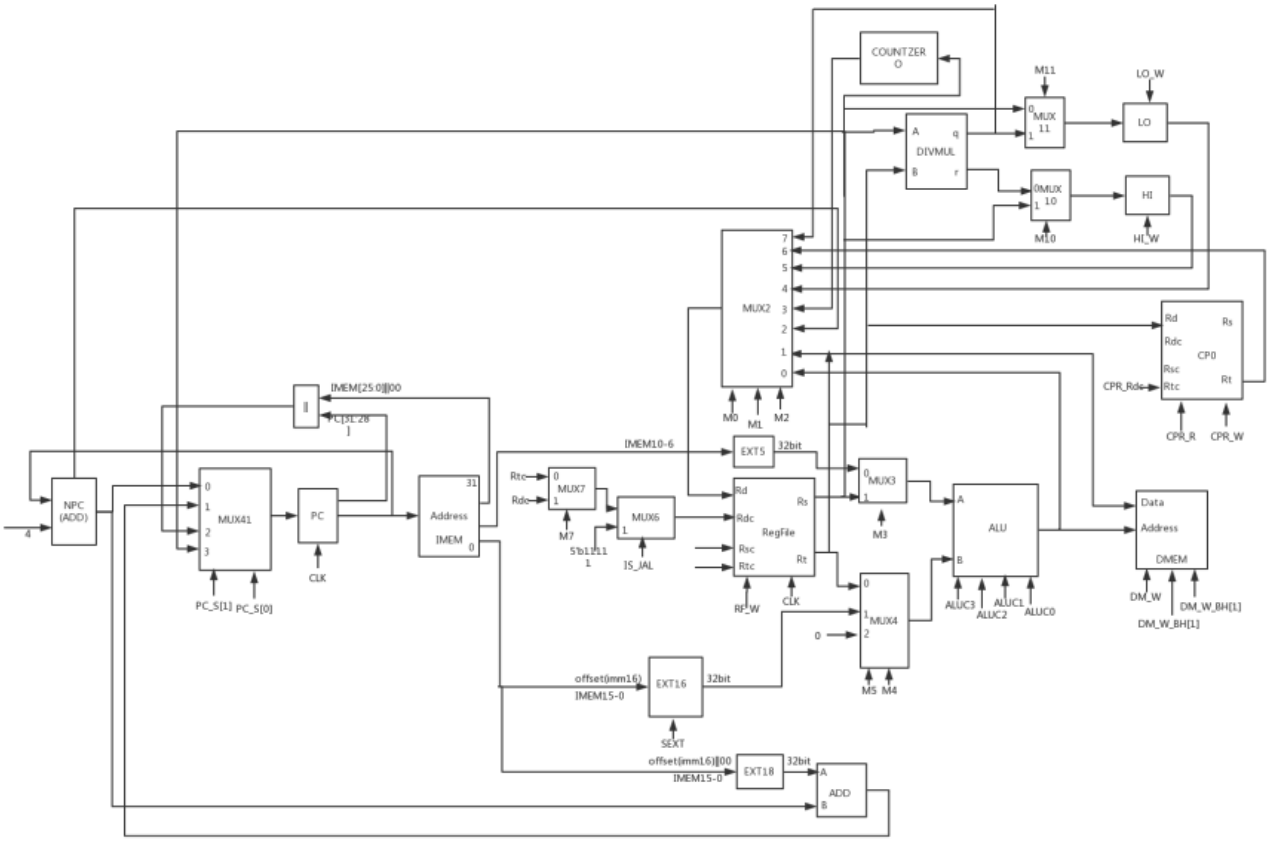
	BH_sel[2]	BH_sel[1]	BH_sel[0]
OTHER	x	X	X
LB	1	0	0
LBU	1	0	1
LH	1	1	0
LHU	1	1	1
LW	0	x	x

阶段性总结：结合上一次 31 条 MIPS 指令单周期 CPU 的实验过程，这一步骤从整体过程中是至关重要，也是耗时最多的一步。完成这一步，要求我们明确每一条指令的功能以及工作流程。在这个基础上将指令的工作分配给相应的部件，同时设计需要的控制信号来正确的调控各个部件的工作。这一步完成之后，我们可以明确所有部件之间数

据流的传输以及所有控制信号的工作内容，以及来源，在此基础上可以完成控制器部件的代码设计。

5. 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

图 5.1 数据通路图



Ps:由于在载入图片过程中进行了一定程度的压缩，因此图片清晰度受损，清晰的 PDF 版本请见附录 5.0

三、测试过程及结果

在完成 CPU 主体部分之后，添加一个 tb.v 文件，在 tb 模块中实例化 CPU，并使用测试指令初始化指令寄存器，同时对 regfile 中的数据进行输出，用于查看 CPU 工作的正确性。

Tb.v 代码如下:

```
module  sccomp_df_tb();
      reg clk;
      reg rst;
      always
            #1 clk<=!clk;

      integer  file_output;
      integer  counter=0;
```

```
reg[31:0]p;

sccomp_dataflow inst_scc(clk, rst);

initial begin
    //修改路径
    file_output=$fopen("F:\\result_my.txt");
    clk=0;
    rst=1;
    p=1;
    #2
    rst=0;
end


always@(posedge clk)begin
    p<=inst_scc.ip_in;
end
always @(posedge clk)begin

  //   if(p!=inst_scc.ip_in)begin
    if(1==1)begin
    counter=counter+1;
    $fdisplay(file_output,"pc: %h",inst_scc.ip_in);
    $fdisplay(file_output,"instr: %h",inst_scc.inst);


    $fdisplay(file_output,"regfile0: %h",inst_scc.sccpu.cpu_ref.array_reg[0]);
    $fdisplay(file_output,"regfile1: %h",inst_scc.sccpu.cpu_ref.array_reg[1]);
    $fdisplay(file_output,"regfile2: %h",inst_scc.sccpu.cpu_ref.array_reg[2]);
    $fdisplay(file_output,"regfile3: %h",inst_scc.sccpu.cpu_ref.array_reg[3]);
    $fdisplay(file_output,"regfile4: %h",inst_scc.sccpu.cpu_ref.array_reg[4]);
    $fdisplay(file_output,"regfile5: %h",inst_scc.sccpu.cpu_ref.array_reg[5]);
    $fdisplay(file_output,"regfile6: %h",inst_scc.sccpu.cpu_ref.array_reg[6]);
    $fdisplay(file_output,"regfile7: %h",inst_scc.sccpu.cpu_ref.array_reg[7]);
    $fdisplay(file_output,"regfile8: %h",inst_scc.sccpu.cpu_ref.array_reg[8]);
    $fdisplay(file_output,"regfile9: %h",inst_scc.sccpu.cpu_ref.array_reg[9]);
    $fdisplay(file_output,"regfile10: %h",inst_scc.sccpu.cpu_ref.array_reg[10]);
    $fdisplay(file_output,"regfile11: %h",inst_scc.sccpu.cpu_ref.array_reg[11]);
    $fdisplay(file_output,"regfile12: %h",inst_scc.sccpu.cpu_ref.array_reg[12]);
    $fdisplay(file_output,"regfile13: %h",inst_scc.sccpu.cpu_ref.array_reg[13]);
    $fdisplay(file_output,"regfile14: %h",inst_scc.sccpu.cpu_ref.array_reg[14]);
    $fdisplay(file_output,"regfile15: %h",inst_scc.sccpu.cpu_ref.array_reg[15]);
    $fdisplay(file_output,"regfile16: %h",inst_scc.sccpu.cpu_ref.array_reg[16]);
```

```
            $fdisplay(file_output,"regfile17: %h",inst_scc.sccpu.cpu_ref.array_reg[17]);
            $fdisplay(file_output,"regfile18: %h",inst_scc.sccpu.cpu_ref.array_reg[18]);
            $fdisplay(file_output,"regfile19: %h",inst_scc.sccpu.cpu_ref.array_reg[19]);
            $fdisplay(file_output,"regfile20: %h",inst_scc.sccpu.cpu_ref.array_reg[20]);
            $fdisplay(file_output,"regfile21: %h",inst_scc.sccpu.cpu_ref.array_reg[21]);
            $fdisplay(file_output,"regfile22: %h",inst_scc.sccpu.cpu_ref.array_reg[22]);
            $fdisplay(file_output,"regfile23: %h",inst_scc.sccpu.cpu_ref.array_reg[23]);
            $fdisplay(file_output,"regfile24: %h",inst_scc.sccpu.cpu_ref.array_reg[24]);
            $fdisplay(file_output,"regfile25: %h",inst_scc.sccpu.cpu_ref.array_reg[25]);
            $fdisplay(file_output,"regfile26: %h",inst_scc.sccpu.cpu_ref.array_reg[26]);
            $fdisplay(file_output,"regfile27: %h",inst_scc.sccpu.cpu_ref.array_reg[27]);
            $fdisplay(file_output,"regfile28: %h",inst_scc.sccpu.cpu_ref.array_reg[28]);
            $fdisplay(file_output,"regfile29: %h",inst_scc.sccpu.cpu_ref.array_reg[29]);
            $fdisplay(file_output,"regfile30: %h",inst_scc.sccpu.cpu_ref.array_reg[30]);
            $fdisplay(file_output,"regfile31: %h",inst_scc.sccpu.cpu_ref.array_reg[31]);
        end
            end
endmodule
```

经过 modelsim 仿真之后会生成 result.txt 文件，使用 TkDiff 软件将其与测试文件对应的正确结果进行比对，以验证 CPU 工作的正确性。

测试过程中经验总结最重要的一个方法：**打印相关接口数据，顺藤摸瓜**

【以 regfile 无法写入数据为例】

解决思路：

- 列出 regfile 相关的变量：RF_W, wdata, wdata
- 将这些变量加入 cpu 的接口列表中，同时注释掉 cpu 模块代码主体中对这些变量的声明
- 修改对应的 tb 文件中对 cpu 的实例化
- simulation，观察 modelsim 中这些变量的值，是否符合预想

以上这个思路是在 debug 我的 cpu 的时候最常用的方法，可以帮助**检查到出错的端口**。在此基础上，**顺藤摸瓜，一步一步排查出错的量，直到找到问题的根源**。

四、部分代码

Dataflow.v

```
module sccomp_dataflow(
    input clk_in,
```

```
    input reset//高电平有效，复位
    //output [31:0] inst,
    //output [31:0] pc,
    //output [31:0] addr
    );
    wire [31:0] inst;
    wire [31:0] pc;
    wire [31:0] addr;
    wire [31:0] dram_data_out;
    wire DM_W;
    wire [31:0] wdata;
    wire [31:0] ip_in;
    assign ip_in=pc-32'h00400000;
    wire [1:0] DM_W_BH;
    wire [2:0] BH_sel;
    cpu sccpu(clk_in,reset,inst,dram_data_out,pc,DM_W_BH,BH_sel,DM_W,addr,wdata);
    iram imem_inst(ip_in[12:2],inst);//右移 2 位，取低 11 位
    dram dm(clk_in,DM_W,DM_W_BH,BH_sel,addr-32'h10010000,wdata,dram_data_out);
endmodule
```

Cpu.v

```
    module cpu(
        input clk_in,
        input reset,//高电平有效
        input [31:0] instruction,
        input [31:0] dram_data_out,
        output [31:0] pc,
        output [1:0] DM_W_BH,
        output [2:0] BH_sel,
        output DM_W,
        output [31:0] addr,//数据存储器待存取的地址
        output [31:0] wdata
        );
        wire RF_W;//寄存器堆的写信号，下降沿+高电平有效
        //wire CPR_R
        wire CPR_W;//特殊寄存器的写信号，下降沿+高电平有效
        wire M0;
        wire M1;
        wire M2;//以上三个主要负责写入寄存器堆的数据的选择
        wire M3;
        wire M4;
        wire M5;//以上三个主要负责传入 ALU 部件 A,B 接口的数据选择
        wire M7;//选择 Rdc
        wire M10;//HI 寄存器的数据来源选择
```

```
        wire M11;//LO 寄存器的数据来源选择
        wire [1:0] PC_SOURCE;//PC 数据来源的选择
        wire [3:0] ALUC;
        //wire DM_R,
        //wire DM_W;//数据寄存器的写信号，高电平有效
        wire SEXT;//传给扩展部件，指示符号扩展（高电平）还是 0 扩展（低电平）
        wire IS_JAL;
        //wire DIV_MUL_RST;//高电平有效。不执行乘除运算时，对 DIVMUL 部件的寄存器置零;
        //wire DIV_MUL_START;//START=!RST & ~busy
        wire [1:0] DIVMUL_sel;//乘除法器部件中，有/无符号乘/除法的选择
        //wire [2:0] BH_sel;//LB,LH,LBU,LHU，LW 的选择信号
        //wire [1:0] DM_W_BH;//SW,SH,SB 的处理，传入 DMEM
        wire LO_W;//写入 LO 寄存器的写信号，高电平有效，时钟下降沿写入
        wire HI_W;//写入 HI 寄存器的写信号，高电平有效，时钟下降沿写入
        wire MFC0;
        wire MTC0;

        //接口，信号定义////////////////////////////////
        //拆解指令 instruction
        wire [5:0] op;
        wire [5:0] func;
        wire [4:0] rsc;//Rsc
        wire [4:0] rtc;//Rtc
        wire [4:0] rdc;
        wire [4:0] r31=5'b11111;
        wire [31:0] rs;
        wire [31:0] rt;
        wire [4:0] shamt;//5 位的 offset
        wire [15:0] immediate;//16 位立即数
        wire [17:0]immediate_tmp;//immediate||00
        wire [25:0] j_addr;//J,Jal 指令中的 address

        //PC 四路选择器数据来源
        wire [31:0]next_pc;
        wire [31:0]pc4;//PC+4
        wire [31:0]pc4offset;//pc+4+offset||00
        wire [31:0]pc4offset_tmp;
        wire [31:0]pc4addr00;//(pc+4)[31:28]||address||00
        wire [31:0]pc4addr00_tmp;
        wire [31:0]pcrs;

        wire [31:0] rf_wdata;//写入 dataram 的数据
```

```
        //ALU 数据输入及运算标志
        wire [31:0] alu_a;
        wire [31:0] alu_b;
        wire [31:0] alu_r;
        wire zero;      //判断运算结果是否为 0
        wire carry;      //判断是否进位
        wire negative;//判断结果是否为负
        wire overflow;//判断结果是否溢出

        wire [31:0] zero_null;//替死鬼
        wire [31:0] zero_num; //前导 0 的个数
        wire [31:0] LO_out;//LO 寄存器读取的数据
        wire [31:0] LO_in;
        wire [31:0] HI_in;
        wire [31:0] HI_out;//HI 寄存器读取的数据
        wire [31:0] cpr_out;//从 CP0 特殊寄存器堆读取的数据

        wire EXCEPTION;
        wire ERET;
        wire [4:0] CAUSE;
        wire [31:0] status;
        wire [31:0] exc_addr;

        wire [31:0] q;
        wire [31:0] r;

       // wire [31:0] dram_data_out;
        wire [31:0] res; //从乘除法器出来，结果 result 的低 32 位写入 regfile
        //wire [31:0] addr;
        ////////////////////////////////////////////////



        //实例化控制器
        control_unit_54
cu54(op,rsc,rs,rt,func,zero,negative,RF_W,CPR_W,M0,M1,M2,M3,M4,M5,M7,M10,M11,P
C_SOURCE,ALUC,DM_W,SEXT,IS_JAL,DIVMUL_sel,BH_sel,DM_W_BH,LO_W,HI_W,
MFC0,MTC0,EXCEPTION,ERET,CAUSE);

        //由于写入地址有 rtc 和 rdc 两种，所以要先经过 mux7
        wire [4:0] rdc_tmp0;
        wire [4:0] rdc_tmp1;
        assign op[5:0] = instruction[31:26];
        assign func[5:0] = instruction[5:0];
```

```
        assign rsc = instruction[25:21];
        assign rtc = instruction[20:16];
        assign rdc_tmp1 = instruction[15:11];
        assign shamt = instruction[10:6];
        assign immediate = instruction[15:0];
        assign j_addr = instruction[25:0];
        assign immediate_tmp ={immediate,{2'b00}} ;//在计算 pc+4+offset||00 时需要用



        //时钟上升沿控制 PC 的更新，将 next_pc 传给 PC，其中 next_pc 已经经过数据来源选择器
        add2 npc(pc,32'h0000_0004,pc4);
        ext18 extoffset(immediate_tmp,SEXT,pc4offset_tmp);
        add2 pc4offset_inst(pc4,pc4offset_tmp,pc4offset);
        assign pc4addr00_tmp = {{pc4[31:28]},j_addr,{2'b00}};//j     jal;
        //assign pc4addr00 = pc4addr00_tmp-32'h0040_0000;
        assign pc4addr00 = pc4addr00_tmp;
        //assign pcrs = rs-32'h0040_0000;
        assign pcrs = rs;
        mux32x41 mux41(pc4,pc4offset,pc4addr00,pcrs,PC_SOURCE,next_pc);
        pcreg pcreg_inst(clk_in,reset,next_pc,pc);

        //实例化指令寄存器
        //wire im_ena=1'b1;
        //instmem instmen_inst(im_ena,pc[9:0],instruction);
        //iram iram_inst(pc,in)
        //选择 Rdc
        mux5x21 mux7(rtc,rdc_tmp1,M7,rdc_tmp0);
        mux5x21 mux8(rdc_tmp0,r31,IS_JAL,rdc);

        //选择 rf_wdata

        wire [31:0] pc4_jal;
        //assign pc4_jal = pc4+32'h0040_0000;
        assign pc4_jal=pc4;
        //从这七个数据来源中，根据选择信号 M0-M2 进行选择，得到 rf_wdata
        mux32x81
mux2(alu_r,dram_data_out,pc4_jal,zero_num,LO_out,HI_out,cpr_out,res,M0,M1,M2,rf_wdata);
        //实例化 regfiles
        regfile cpu_ref(clk_in,reset,RF_W,rsc,rtc,rdc,rf_wdata,rs,rt);
```

```
        //实例化 ALU 部件


        wire [31:0] tmp_a;
        wire [31:0] tmp_b;
        ext5 extshamt(shamt,SEXT,tmp_a);
        mux32x21 mux3(tmp_a,rs,M3,alu_a);
        ext16 extimm(immediate,SEXT,tmp_b);
        wire [31:0] zero_b_in;
        assign zero_b_in = 32'h0;
        assign zero_null = 32'h0;
        mux32x41 mux4(rt,tmp_b,zero_b_in,zero_null,{M5,M4},alu_b);
        alu alu_inst(alu_a,alu_b,ALUC,alu_r,zero,carry,negative,overflow);


        //实例化 countZero 部件
        countZero cz(rs,zero_num);

        //实例化乘除法器
        divmul divm(rs,rt,DIVMUL_sel,q,r,res);
        //实例化 LO 部件
        mux32x21 mux11(rs,q,M11,LO_in);
        HILOreg LO(LO_in,LO_W,clk_in,LO_out);
        //实例化 HI 部件
        mux32x21 mux10(r,rs,M10,HI_in);
        HILOreg HI(HI_in,HI_W,clk_in,HI_out);

        //实例化 CP0，即特殊寄存器堆
        CP0
cp0(clk_in,reset,CPR_W,MFC0,MTC0,pc,rdc_tmp1,rt,EXCEPTION,ERET,CAUSE,cpr_out,
status,exc_addr);

        //实例化 dmem,之后拿出 CPU
        //wire [31:0] addr;
        //assign addr=alu_r-32'h10010000;
        assign addr=alu_r;
        assign wdata=rt;
        //dram dm(clk_in,reset,DM_W,DM_W_BH,BH_sel,addr,wdata,dram_data_out);
        //dram dm(clk_in,DM_W,DM_W_BH,BH_sel,alu_r[9:0],rt,dram_data_out);
        //dram dm(clk_in,DM_W,alu_r[9:0],rt,dram_data_out);
endmodule
```

Control_unit.v

```
    module control_unit_54(
```

```
        input [5:0] op,
        input [4:0] rsc,
        input [31:0] rs,
        input [31:0] rt,
        input [5:0] func,
        //input busy,//由乘除法器传出的信号
        input z,//ALU 的 zero 属性
        input n,//ALU 的 negetive 属性
        //output RF_R,/
        output RF_W,//寄存器堆的写信号，下降沿+高电平有效
        //output CPR_R
        output CPR_W,//特殊寄存器的写信号，下降沿+高电平有效
        output M0,
        output M1,
        output M2,//以上三个主要负责写入寄存器堆的数据的选择
        output M3,
        output M4,
        output M5,//以上三个主要负责传入 ALU 部件 A,B 接口的数据选择
        output M7,//选择 Rdc
        output M10,//HI 寄存器的数据来源选择
        output M11,//LO 寄存器的数据来源选择
        output [1:0] PC_SOURCE,//PC 数据来源的选择
        output [3:0] ALUC,
        //output DM_R,
        output DM_W,//数据寄存器的写信号，高电平有效
        output SEXT,//传给扩展部件，指示符号扩展（高电平）还是 0 扩展（低电平）
        output IS_JAL,
        //output DIV_MUL_RST,//高电平有效。不执行乘除运算时，对 DIVMUL 部件
的寄存器置零；
        //output DIV_MUL_START,//START=!RST & ~busy
        output [1:0] DIVMUL_sel,//乘除法器部件中，有/无符号乘/除法的选择
        output [2:0] BH_sel,//LB,LH,LBU,LHU，LW 的选择信号
        output [1:0] DM_W_BH,//SW,SH,SB 的处理，传入 DMEM
        output LO_W,//写入 LO 寄存器的写信号，高电平有效，时钟下降沿写入
        output HI_W, //写入 HI 寄存器的写信号，高电平有效，时钟下降沿写入
        output MFC0,
        output MTC0,
        output EXCEPTION,
        output ERET,
        output [4:0] CAUSE
        );
        wire ADD,ADDU,SUB,SUBU,AND,OR,XOR,NOR,SLT,SLTU;
        wire SLL,SRL,SRA,SLLV,SRLV,SRAV,JR,ADDI,ADDIU,ANDI,ORI,XORI;
        wire LW,SW,BEQ,BNE,SLTI,SLTIU,LUI,J,JAL,CLZ,DIVU;
```

```
        wire JALR,LB,LBU,LHU,SB,SH,LH,MFHI,MFLO;
        wire MTHI,MTLO,MUL,MULTU,SYSCALL,TEQ,BGEZ,BREAK,DIV;

        instruction_decoder inst_decode(op,rsc,func,ADD,ADDU,SUB,SUBU,AND,
OR,XOR,

NOR,SLT,SLTU,SLL,SRL,SRA,SLLV,SRLV,SRAV,JR,ADDI,ADDIU,ANDI,ORI,XORI,

LW,SW,BEQ,BNE,SLTI,SLTIU,LUI,J,JAL,CLZ,DIVU,ERET,JALR,LB,LBU,LHU,
                SB,SH,LH,MFC0,MFHI,MFLO,MTC0,MTHI,MTLO,//字母 o
                MUL,MULTU,SYSCALL,TEQ, BGEZ,BREAK,DIV);

        assign EXCEPTION = BREAK | TEQ &(rs==rt) | SYSCALL;
        assign CAUSE[4]=1'b0;
        assign CAUSE[3]=BREAK||SYSCALL||(TEQ && rt==rs);
        assign CAUSE[2]=TEQ;
        assign CAUSE[1]=1'b0;
        assign CAUSE[0]=(TEQ && rt==rs)||BREAK;
        assign HI_W = DIV | DIVU | MUL | MULTU | MTHI;
        assign LO_W = DIV | DIVU | MUL | MULTU | MTLO;
        assign DM_W_BH[0] = SH;
        assign DM_W_BH[1] = SH | SB;
        assign BH_sel[0] = LBU | LHU;
        assign BH_sel[1] = LH | LHU;
        assign BH_sel[2] = LB | LBU | LH | LHU;
        assign DIVMUL_sel[0] = MULTU | DIVU;
        assign DIVMUL_sel[1] = MULTU | MUL;
        //assign DIV_MUL_RST = ~MUL & ~MULTU & ~DIV & ~DIVU;
        //assign DIV_MUL_START = ~DIV_MUL_RST & ~busy;
        assign IS_JAL = JAL ;
        assign SEXT = ADDI | ADDIU | LUI | SLTI | LW | SW | BEQ | BNE | LB | LBU |
LHU | LH | SB | SH;
        assign DM_W = SW | SB | SH;
        assign ALUC[0] = SUB | SUBU | OR | NOR | SLT | SRLV | SRL | ORI | SLTI |
BEQ | BNE | BGEZ;
        assign ALUC[1] = ADD | SUB | XOR | NOR | SLT | SLTU | SLLV | SLL | ADDI |
XORI | SLTI | SLTIU | LW | SW | BEQ | BNE | BGEZ | LB | LH | LBU | LHU | SB | SH ;
        assign ALUC[2] = AND | OR | XOR | NOR | SRAV | SLLV | SRLV | SLL | SRA |
SRL | ANDI | ORI | XORI;
        assign ALUC[3] = SRAV | SLLV | SRLV | SLL | SRA | SRL | SLT | SLTU | LUI |
SLTI | SLTIU;
        assign PC_SOURCE[0] = BEQ & z | BNE & ~z | JR | BGEZ & ~n | JALR;
        assign PC_SOURCE[1] = J | JAL | JR | JALR;
        assign M11 = DIV | DIVU | MUL | MULTU;
```

```
        assign M10 = MTHI;
        assign M7 = ADDU | ADD | SUB | SUBU | AND | OR | XOR | NOR | SLT | SLTU |
SRAV | SLLV | SRLV | SLL | SRA | SRL | JALR | CLZ | MFLO | MFHI | MUL | MULTU;
        assign M5 = BGEZ;
        assign M4 = ADDI | ADDIU | ANDI | ADDI | ADDIU | ORI | XORI | LUI | SLTI |
SLTIU | LW | SW | LB | LH | LBU | LHU | SB | SH ;
        assign M3 = ADDU | ADD | SUB | SUBU | AND | OR | XOR | NOR | SLT | SLTU |
SRAV | SLLV | SRLV | ADDI | ADDIU | ANDI | ORI | XORI | SLTI | SLTIU | SW | LW |
BEQ | BNE | JR | BGEZ | LH | LHU | LB | LBU | SB | SH ;
        assign M2 = LW | LB | LBU | LH | LHU | CLZ | MFHI | MUL | MULTU;
        assign M1 = JAL | JALR | CLZ | MFC0 | MUL | MULTU;
        assign M0 = MFC0 | MFLO | MFHI | MUL | MULTU;
        assign CPR_W = MTC0 | EXCEPTION; ////////////////////////////////////
        assign RF_W = ADDU | ADD | SUB | SUBU | AND | OR | XOR | NOR | SLT |
SLTU | SRAV | SLLV | SRLV | SLL | SRL | SRA |ADDI | ADDIU | ANDI | ORI | XORI | LUI
| SLTI | SLTIU | LW | JAL | JALR | LB | LBU | LH | LHU | CLZ | MFC0 | MFLO | MFHI |
MUL | MULTU ;
endmodule
```

Dmem.v

```
    module dram(
        input clk_in,//下降沿触发写操作，当 wena 有效时，写
        input wena,//高电平有效
        input [1:0] DM_W_BH,//写入一个字节，或 2 个字节的选择信号
        input [2:0] BH_sel,//读取数据时，选择读出 B,H，BU,HU 或者 W
        input [31:0] addr,
        input [31:0] data_in,
        output [31:0] data_out
        );
        reg [31:0]dataram[0:1023];
        reg [31:0] data_out_tmp;
        reg [7:0] tmp_8;
        reg [15:0] tmp_16;
        assign data_out=data_out_tmp;
        always @(*)
        begin
            if (BH_sel[2]==0)//LW
            begin
                data_out_tmp=dataram[addr];
            end

            else if (BH_sel[1]==0)
            begin
                tmp_8=dataram[addr][7:0];
```

```
                    if (BH_sel[0]==1'b0)//LB
                    begin
                        data_out_tmp={{24{tmp_8[7]}},tmp_8};
                    end
                    else //LBU
                    begin
                        data_out_tmp={24'b0,tmp_8};
                    end
                end

                else
                begin
                    tmp_16=dataram[addr][15:0];
                    if(BH_sel[0]==1'b0)//LH
                    begin
                        data_out_tmp={{16{tmp_16[15]}},tmp_16};
                    end
                    else
                    begin
                        data_out_tmp={16'b0,tmp_16};
                    end
                end
            end

            always @(negedge clk_in)
            begin
                if (wena==1'b1)
                begin
                    // write
                    if (DM_W_BH[1]==1'b0)
                        dataram[addr]=data_in;
                    else if(DM_W_BH[0]==1'b0)//SB,存 data_in 的低 8 位，到 dataram[addr]
                        dataram[addr][7:0]=data_in[7:0];
                    else//SH,存 data_in 的低 16 位,到 dataram[addr]
                        dataram[addr][15:0]=data_in[15:0];
                end
            end
endmodule
```

Regfile.v

```
    module regfile(
        input clk_in,//下降沿写入数据
        input rst,//清零信号，高电平有效，所有寄存器清零，异步
        input we,//写信号，高电平有效
```

```
input [4:0] raddr1,//读数据的地址 1，Rs
input [4:0] raddr2,//读数据的地址 2，Rt
input [4:0] waddr,//写数据的地址，Rd
input [31:0] wdata,//写入的数据
output [31:0] rdata1,//读取的数据 1
output [31:0] rdata2//读取的数据 2
);
reg [31:0] array_reg [0:32];
assign rdata1 = array_reg[raddr1];
assign rdata2 = array_reg[raddr2];
always @(negedge clk_in or posedge rst)
begin
    if (rst==1'b1)
    begin
        // reset
        array_reg[0]=32'h0;
        array_reg[1]=32'h0;
        array_reg[2]=32'h0;
        array_reg[3]=32'h0;
        array_reg[4]=32'h0;
        array_reg[5]=32'h0;
        array_reg[6]=32'h0;
        array_reg[7]=32'h0;
        array_reg[8]=32'h0;
        array_reg[9]=32'h0;
        array_reg[10]=32'h0;
        array_reg[11]=32'h0;
        array_reg[12]=32'h0;
        array_reg[13]=32'h0;
        array_reg[14]=32'h0;
        array_reg[15]=32'h0;
        array_reg[16]=32'h0;
        array_reg[17]=32'h0;
        array_reg[18]=32'h0;
        array_reg[19]=32'h0;
        array_reg[20]=32'h0;
        array_reg[21]=32'h0;
        array_reg[22]=32'h0;
        array_reg[23]=32'h0;
        array_reg[24]=32'h0;
        array_reg[25]=32'h0;
        array_reg[26]=32'h0;
        array_reg[27]=32'h0;
        array_reg[28]=32'h0;
```

```
                array_reg[29]=32'h0;
                array_reg[30]=32'h0;
                array_reg[31]=32'h0;
                array_reg[32]=32'h0;
            end
            else if(we==1'b1) //写信号有效
            begin
                if(waddr==5'b00000)
                    array_reg[waddr]=32'h0;
                else
                    array_reg[waddr]=wdata;
            end
        end
endmodule
```

五、实验总结

在实验过程中，我对 MIPS 体系结构下的单周期 CPU 的工作流程有了深刻的认识，了解 CPU 内部各个部件之间的关系以及工作时数据的流动。同时在实验过程中大大提高了对 verilog 语言的掌握程度，对 vivado ，modelsim 等软件的使用，更加重要的是，在知识架构中填补了非常重要的底层硬件相关的空白，为将来的学习铺设重要的基础。