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 **1352652**

姓　　名 周宇星

专　　业 计算机科学与技术

授课老师 陈永生

日　　期 2016 年 6 月 15 日

目录

实验目的

利用 VERILOG 硬件描述语言，借助开发板自行设计并实现一个具有 31 条指令的 MIPS 指令 CPU。

实验要求

设计并实现一个具有 31 条指令的 MIPS 指令 CPU，通过 31 条指令的测试，并在下板时 成功运行自己所编写的汇编测试指令。

实验步骤

了解 mips 指令集

实验要求的 MIPS 指令一共可以归纳为 3 类指令：运算指令，跳转指令，存取指令。我在除去控制器后，来分别讨论每种指令需要用到的部件。

运算指令又可以继续细分为两种

- ①寄存器与寄存器运算
- ②寄存器与立即数运算

可以看出，它们除了数据来源不同之外，剩下的操作都是相同的。给 Alu 发送控制信号，确定运算方式；给多路选择器发送选择信号，确定运算数据；给 Regfiles 写使能信号，读取 Alu 结果

所以运算指令的微操作为：①选择数据 ②计算数据 ③写回 Regfiles

跳转指令

跳转指令也可以继续细分为两种

- ①无条件跳转指令

- ②条件跳转指令

无条件跳转指令只与跳转地址相关。在 31 条指令中，无条件跳转一共只有 3 条 j, jal, jr。 其中 jal 属于比较特殊的跳转，跳转同时需要将 PC+8 的值存入$ra 中。

而条件跳转指令需要满足某种条件才可以跳转，相对于无条件跳转而已，我们需要先判 断条件是否满足。

j 与 jr 指令可以理解为立即数跳转与寄存器跳转。 所以 j 与 jr 本质上是一样的。将要跳转的地址送入 PC寄存器，PC寄存器使能，即可完 成跳转。

jal 比 j 多了先将 PC 寄存器的值+8 后存入$31，然后再将 PC寄存器的值更新。

对条件跳转指令(bne, beq)，我们可以先判断条件是否满足同时输送跳转地址，然后如果条件满足，我们就对 PC发使能信号，否则我们就不对 PC发使能信号。

存取指令

存取指令相对而已就简单多了，因为一共只有两条

- SW 存字操作

- LW 取字操作

SW 与 LW 基本一致，都需要先计算出地址，然后进行存取操作。

对于 SW 而言，需要对 RAM 发读使能信号

对于 LW 而已，需要对 Regfiles 发写使能信号

从上述分析，我们可以发现 CPU 必要的部件有 regfiles，alu，Ram，IR 寄存器和 PC 寄存器，加上需要一个控制器以及一个指令存储器。

部件	功能
Regfiles	存储中间数据
Alu	进行相关运算
Ram	存储数据
imemout	储存执行指令
PC	存储指令地址
控制器	分析指令，发出控制信号
Imem	存储指令数据

在不违反 MIPS 指令集的前提条件下对 CPU 进行改造

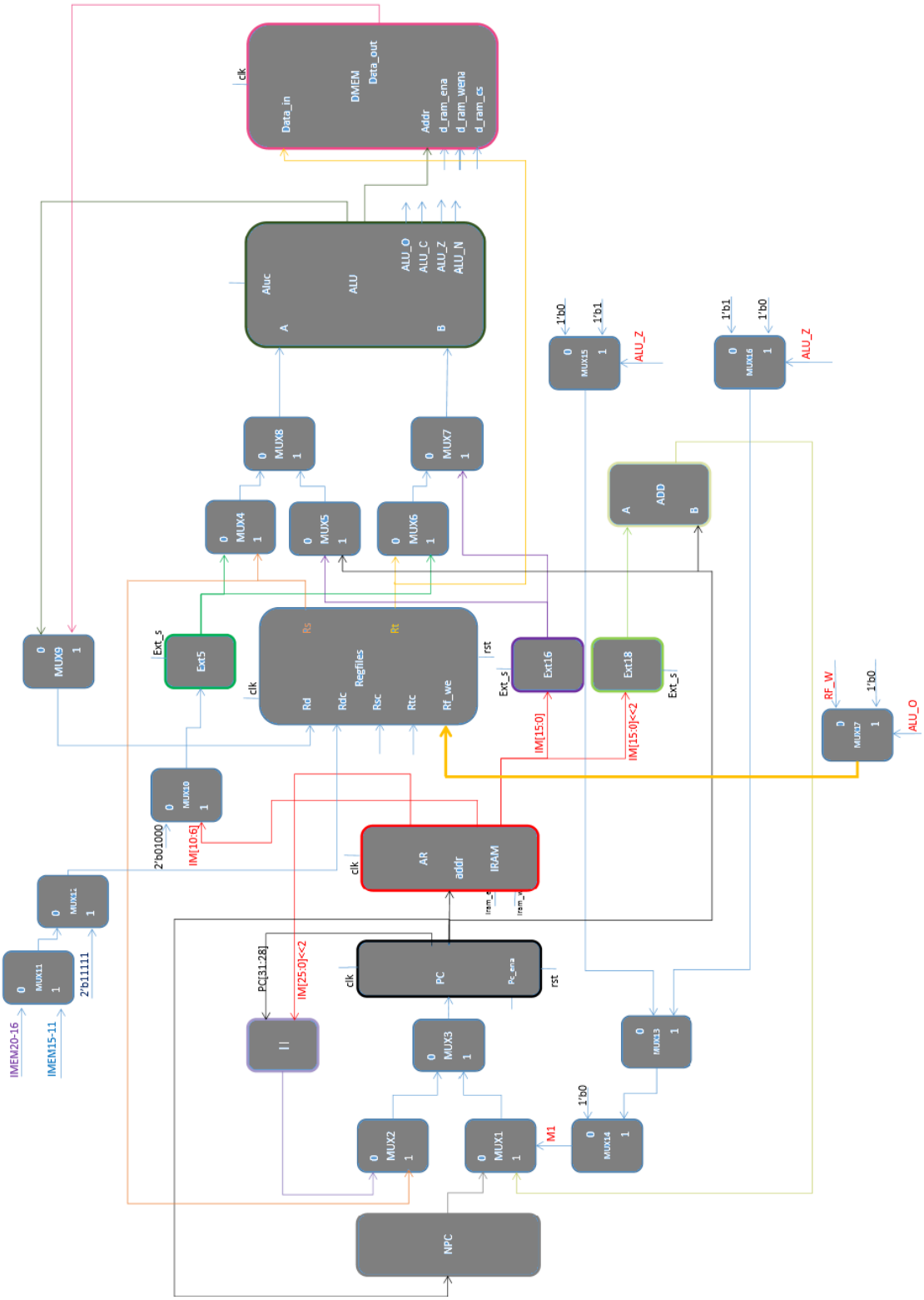
虽然，通过上述结构，我们可以构造出一个可以使用的 CPU，但是，由于上述数据全部为内部数据，我们无法展示 CPU 运行结果，所以在不违反 CPU 大的结构且不增加 CPU 指令 的前提条件下，我们对 CPU 进行功能补充。

1） 利用数码管展示数据。链接Regfiles中某一寄存器与数码管模块数据输入接口，即可实现寄存器数据输出。

2） 通过改造 Ram 模块使得 CPU 可以利用 lw 指令进行读取键盘数据。

利用指令集构建数据通路

由指令分析过程结合 Mips 文档，前期部件实验我们可以得出各个部件之间的数据关系，由此构造如下数据通路图：



构造控制命令

一条指令的实现可分成取指、计算地址、取数 及执行等几个步骤，对于单周期处理器来说，所有操作就在一个机器周期中实现。

1． 实现操作码译码器

通过 MIPS 指令集文档，我们可以用实现译码器

```
wire ADD=    imemout5 == 0 && imemout6 == 6'b100000;
wire ADDI=   imemout5 == 6'b001000 ;
wire ADDU=   imemout5 == 6'b000000   && imemout6 == 6'b100001 ;
wire ADDIU=  imemout5 == 6'b001001;
wire AND=    imemout5 == 6'b000000   && imemout6 == 6'b100100 ;
wire ANDI=   imemout5 == 6'b001100;
wire NOR=    imemout5 == 6'b000000   && imemout6 == 6'b100111 ;
wire OR=     imemout5 == 6'b000000   && imemout6 == 6'b100101 ;
wire ORI=    imemout5 == 6'b001101;
wire SLL=    imemout5 == 6'b000000   && imemout6 == 6'b000000 ;
wire SRL=    imemout5 == 6'b000000   && imemout6 == 6'b000010 ;
wire SRA=    imemout5 == 6'b000000   && imemout6 == 6'b000011 ;
wire SRLV=   imemout5 == 6'b000000   && imemout6 == 6'b000110 ;
wire SRAV=   imemout5 == 6'b000000   && imemout6 == 6'b000111 ;
wire SLLV=   imemout5 == 6'b000000   && imemout6 == 6'b000100 ;
wire SUB=    imemout5 == 6'b000000   && imemout6 == 6'b100010 ;
wire SUBU=   imemout5 == 6'b000000   && imemout6 == 6'b100011 ;
wire XOR=    imemout5 == 6'b000000   && imemout6 == 6'b100110 ;
wire XORI=   imemout5 == 6'b001110;
wire LUI=    imemout5 == 6'b001111;
wire SLT=    imemout5 == 6'b000000   && imemout6 == 6'b101010 ;
wire SLTI=   imemout5 == 6'b001010;
wire SLTU=   imemout5 == 6'b000000   && imemout6 == 6'b101011 ;
wire SLTIU=  imemout5 == 6'b001011;
wire BEQ=    imemout5 == 6'b000100;
wire BNE=    imemout5 == 6'b000101;
wire J=      imemout5 == 6'b000010;
wire JAL=    imemout5 == 6'b000011;
wire JR=     imemout5 == 6'b000000   && imemout6 == 6'b001000 ;
wire LW=     imemout5 == 6'b100011;
wire SW=     imemout5 == 6'b101011;
```

2. 编排指令操作时间表

依据各条机器指令的操作流程及控制方式， 将每个微操作安排到周期节拍中。

	ADD	ADDU	SUB	SUBU	AND	OR	XOR	NOR
PC_CLK	1	1	1	1	1	1	1	1
IM_R	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

M4	1	1	1	1	1	1	1	1
M5								
M8	0	0	0	0	0	0	0	0
M6	0	0	0	0	0	0	0	0
M7	0	0	0	0	0	0	0	0
ALUC[3]	0	0	0	0	0	0	0	0
ALUC[2]	0	0	0	0	1	1	1	1
ALUC[1]	1	0	1	0	0	0	1	1
ALUC[0]	0	0	1	1	0	1	0	1
M9	0	0	0	0	0	0	0	0
RDC4-0	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11
RF_W	1	1	1	1	1	1	1	1
RF_CLK	1	1	1	1	1	1	1	1
M1	0	0	0	0	0	0	0	0
M2								
M3	1	1	1	1	1	1	1	1
M10								
EXT5_S								
EXT16_S								
EXT18_S								
DMEM_CLK								
DMEM_R								
DMEM_W								
M11	1	1	1	1	1	1	1	1
M12	0	0	0	0	0	0	0	0
M13								
M14								
	SLT	SLTU	SLL	SRL	SRA	SLLV	SRLV	SRAV
PC_CLK	1	1	1	1	1	1	1	1
IM_R	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
M4	1	1	0	0	0	1	1	1
M5								
M8	0	0	0	0	0	0	0	0
M6	0	0	0	0	0	0	0	0
M7	0	0	0	0	0	0	0	0
ALUC[3]	1	1	1	1	1	1	1	1
ALUC[2]	0	0	1	1	1	1	1	1
ALUC[1]	1	1	1	0	0	1	0	0
ALUC[0]	1	0		1	0		1	0
M9	0	0	0	0	0	0	0	0
RDC4-0	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11

RF_W	1	1	1	1	1	1	1	1
RF_CLK	1	1	1	1	1	1	1	1
M1	0	0	0	0	0	0	0	0
M2								
M3	1	1	1	1	1	1	1	1
M10			1	1	1			
EXT5_S			0	0	0			
EXT16_S								
EXT18_S								
DMEM_CLK								
DMEM_R								
DMEM_W								
M11	1	1	1	1	1	1	1	1
M12	0	0	0	0	0	0	0	0
M13								
M14								
	JR	ADDI	ADDIU	ANDI	ORI	XORI	LW	SW
PC_CLK	1	1	1	1	1	1	1	1
IM_R	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
M4		1	1	1	1	1	1	1
M5								
M8		0	0	0	0	0	0	0
M6								
M7		1	1	1	1	1	1	1
ALUC[3]		0	0	0	0	0	0	0
ALUC[2]		0	0	1	1	1	0	0
ALUC[1]		1	0	0	0	1	1	1
ALUC[0]		0	0	0	1	0	0	0
M9		0	0	0	0	0	1	
RDC4-0		IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	
RF_W	0	1	1	1	1	1	1	0
RF_CLK		1	1	1	1	1	1	0
M1		0	0	0	0	0	0	0
M2	1							
M3	0	1	1	1	1	1	1	1
M10								
EXT5_S								
EXT16_S		1	1	0	0	0	1	1
EXT18_S								
DMEM_CLK							1	1
DMEM_R							1	0

DMEM_W							0	1
M11		0	0	0	0	0	0	
M12		0	0	0	0	0	0	
M13								
M14								
	BEQ	BNE	SLTI	SLTIU	LUI	J	JAL	
PC_CLK	1	1	1	1	1	1	1	
IM_R	1	1	1	1	1	1	1	
RSC4-0	IM25-21	IM25-21	IM25-21	IM25-21				
RTC4-0	IM20-16	IM20-16						
M4	1	1	1	1				
M5							1	
M8	0	0	0	0			1	
M6	0	0					1	
M7	0	0	1	1	1		0	
ALUC[3]	0	0	1	1	1		0	
ALUC[2]	0	0	0	0	0		0	
ALUC[1]	1	1	1	1	0		1	
ALUC[0]	1	1	1	0			0	
M9			0	0	0		0	
RDC4-0			IM20-16	IM20-16	IM20-16		2'b11111	
RF_W	0	0	1	1	1	0	1	
RF_CLK	0	0	1	1	1	0	1	
M1	Alu_Z	!ALU_Z	0	0	0			
M2						0	0	
M3	1	1	1	1	1	0	0	
M10							0	
EXT5_S								
EXT16_S			1	1	1			
EXT18_S	1	1						
DMEM_CLK	0	0				0	0	
DMEM_R	0	0				0	0	
DMEM_W	0	0				0	0	
M11			0	0	0			
M12			0	0	0		1	
M13	0	1						
M14	1	1						

3. 进行微操作综合

按照所有机器指令的操作时间表，把相同的微操作综合起来：

```
assign RSC4_0 = {imemout0[25:21]};
assign RTC4_0 = {imemout0[20:16]};
assign m[4] = ADD | ADDU | SUB | SUBU | AND | OR | XOR | NOR | SLT | SLTU |
SLLV | SRLV | SRAV | ADDI | ADDIU | ANDI | ORI | XORI | LW | SW | BEQ | BNE | SLTI |
```

```
SLTIU;
    assign m[5] = JAL;
    assign m[6] = JAL;
    assign m[7] = ADDI | ADDIU | ANDI | ORI | XORI | LW | SW | SLTI | SLTIU | LUI;
    assign m[8] = JAL;
    assign aluc[3] = SLT | SLTU | SLL | SRL | SRA | SLLV | SRLV | SRAV | SLTI | SLTIU |
LUI;
    assign aluc[2] = AND | OR | XOR | NOR | SLL | SRL | SRA | SLLV | SRLV | SRAV |
ANDI | ORI | XORI;
    assign aluc[1] = ADD | SUB | XOR | NOR | SLT | SLTU | SLL | SLLV | ADDI | XORI |
LW | SW | BEQ | BNE | SLTI | SLTIU | JAL;
    assign aluc[0] = SUB | SUBU | OR | NOR | SLT | SRL | SRLV | ORI | BEQ | BNE | SLTI;
    assign m[9] = LW;
    assign IM1511 = {imemout0[15:11]};
    assign IM2016 = {imemout0[20:16]};
    assign RF_W = ADD | ADDU | SUB | SUBU | AND | OR | XOR | NOR | SLT | SLTU |
SLL | SRL | SRA | SLLV | SRLV | SRAV | ADDI | ADDIU | ANDI | ORI | XORI | LW | SLTI |
SLTIU | LUI | JAL;
    assign m[2] = JR;
    assign m[3] = ADD | ADDU | SUB | SUBU | AND | OR | XOR | NOR | SLT | SLTU | SLL
| SRL | SRA | SLLV | SRLV | SRAV | ADDI | ADDIU | ANDI | ORI | XORI | LW | SW | BEQ |
BNE | SLTI | SLTIU | LUI;
    assign m[10] = SLL | SRL | SRA;
    assign ext5_s = 0;
    assign ext16_s = ADDI | ADDIU | LW | SW | SLTI | SLTIU | LUI;
    assign ext18_s = BEQ | BNE;
    assign dmem_cs = LW | SW;
    assign dmem_r = LW;
    assign dmem_w = SW;
    assign m[11] = ADD | ADDU | SUB | SUBU | AND | OR | XOR | NOR | SLT | SLTU |
SLL | SRL | SRA | SLLV | SRLV | SRAV;
    assign m[12] = JAL;
    assign m[13] = BNE;
    assign m[14] = BEQ | BNE;
```

实现外部设备

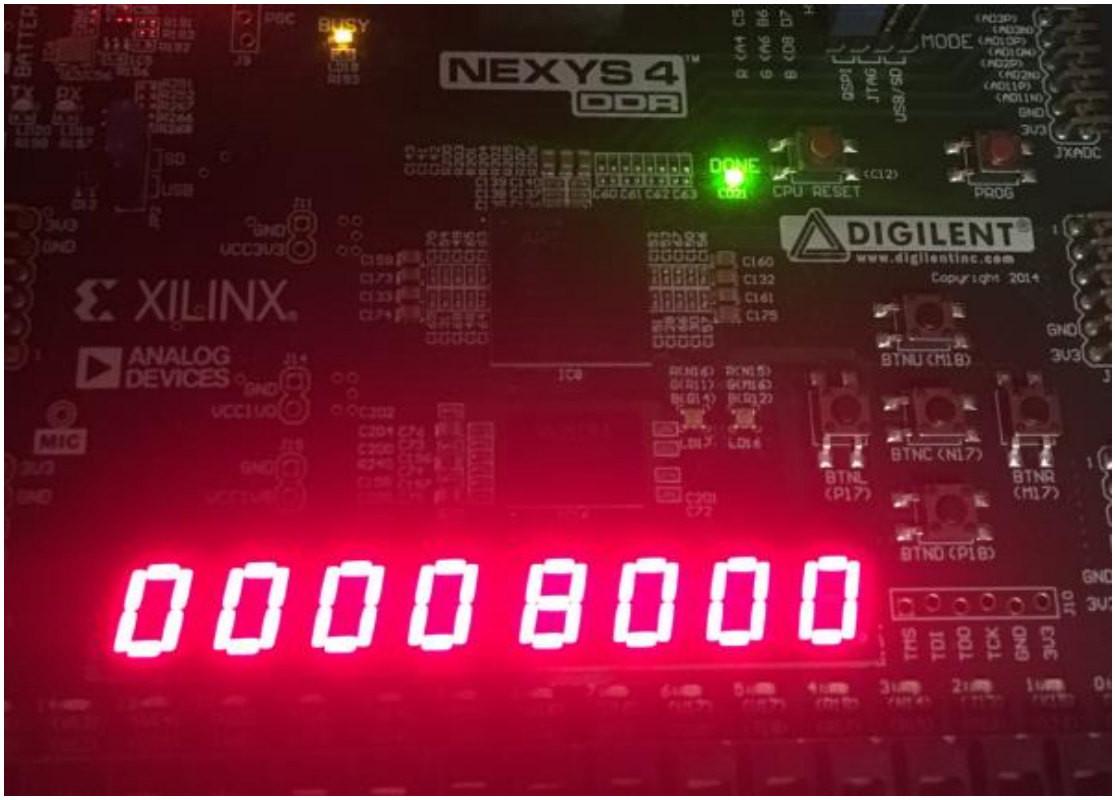
1. 数码管
   数码管由两个端口控制，digitselect 控制显示的单元,segment 控制一个单元上的 8 段电平。通过合适频率的刷新（1k）左右，加上编码器，可以实现同时显示 8 个 16 进制数。
2. 键盘
   键盘并不记录按键按下的状态，而仅仅是在按键按下和弹开的时候发送对应的扫描码。所以我们可以记录某个键按下/松开的状态。当收到扫描码时，更新按键的状态。

CPU 测试

1. 前仿真
   利用 MIPS246 提供的标准输入输出以及文本对比软件进行比对。
2. 后期测试
   通过下板后利用数码管显示寄存器的值，以此判断是否执行正确。
3. 估计性能
   通过仿真测试，以 lw 指令为例，记录从上升沿到 RD 稳定的时间，本 CPU 约为 21ns，故此估计其频率约 23.8Mhz
   在下板测试中，选用 12.5Mhz（1/8*100Mhz）的频率进行测试，所有测试程序均正确运行

CPU 演示程序制作

利用数码管和键盘，可以实现经典小游戏打地鼠，8 个数码显示位，在游戏进行时会随机显示 0 或 8 分别代表无/有地鼠出现，在规定时间内按下键盘对应按键，即可成功打到地鼠，得分加 1，否则进入失败状态(左 4 个位置显示 FFFF)，此时右 4 位显示得分。



附汇编代码：

```
sll $0,$0,0
lui $21,0x005f
ori $19,$21,0xffff
lui $21,0x003f
ori $20,$21,0xffff
addiu $10,$0,0x1000
addiu $25,$0,0x0001

addiu $1,$0,0x0042
addiu $2,$0,0x003b
addiu $3,$0,0x0033
addiu $4,$0,0x0034
addiu $5,$0,0x002b
addiu $6,$0,0x0023
addiu $7,$0,0x001b
addiu $8,$0,0x001c

lui $21,0xffff
addu $27,$0,$0

xia:
    addiu $23,$25,0x0033
    andi $25,$23,0x0007
```

```
        addiu $9,$0,0x0000
        beq $25,$9,NOW1
        addiu $9,$0,0x0001
        beq $25,$9,NOW2
        addiu $9,$0,0x0002
        beq $25,$9,NOW3
        addiu $9,$0,0x0003
        beq $25,$9,NOW4
        addiu $9,$0,0x0004
        beq $25,$9,NOW5
        addiu $9,$0,0x0005
        beq $25,$9,NOW6
        addiu $9,$0,0x0006
        beq $25,$9,NOW7
        addiu $9,$0,0x0007
        beq $25,$9,NOW8
j xia

SUCC:
        addu $17,$0,$0
        addi $27,$27,1
        rests:
                add $18,$0,$0
        _times:
                nop
                beq $18,$19,xia
                addi $18,$18,1
        j _times

FAIL:
        or $29,$21,$27
        addu $17,$0,$29
        restf:
                add $18,$0,$0
        _timef:
                nop
                beq $18,$19,xia
                addi $18,$18,1
        j _timef


NOW1:
        addiu $17,$0,0x0008
        rest1:
```

```
            add $18,$0,$0
        _time1:
            lw $16,0($10)
            beq $16,$1,SUCC
            beq $18,$20,FAIL
            addi $18,$18,1
        j _time1

NOW2:
        addiu $17,$0,0x0080
        rest2:
            add $18,$0,$0
        _time2:
            lw $16,0($10)
            beq $16,$2,SUCC
            beq $18,$20,FAIL
            addi $18,$18,1
        j _time2

NOW3:
        addiu $17,$0,0x0800
        rest3:
            add $18,$0,$0
        _time3:
            lw $16,0($10)
            beq $16,$3,SUCC
            beq $18,$20,FAIL
            addi $18,$18,1
        j _time3

NOW4:
        ori $17,$0,0x8000
        rest4:
            add $18,$0,$0
        _time4:
            lw $16,0($10)
            beq $16,$4,SUCC
            beq $18,$20,FAIL
            addi $18,$18,1
        j _time4

NOW5:
        lui $17,0x0008
        rest5:
```

```
            add $18,$0,$0
        _time5:
            lw $16,0($10)
            beq $16,$5,SUCC
            beq $18,$20,FAIL
            addi $18,$18,1
        j _time5

NOW6:
        lui $17,0x0080
        rest6:
            add $18,$0,$0
        _time6:
            lw $16,0($10)
            beq $16,$6,SUCC
            beq $18,$20,FAIL
            addi $18,$18,1
        j _time6

NOW7:
        lui $17,0x0800
        rest7:
            add $18,$0,$0
        _time7:
            lw $16,0($10)
            beq $16,$7,SUCC
            beq $18,$20,FAIL
            addi $18,$18,1
        j _time7

NOW8:
        lui $17,0x8000
        rest8:
            add $18,$0,$0
        _time8:
            lw $16,0($10)
            beq $16,$8,SUCC
            beq $18,$20,FAIL
            addi $18,$18,1
        j _time8
```

总结及反思

本次实验我最大的教训是，一完成初期制作就测试指令合集，结果出现错误，无法查明错误原因。经过反思，决定先使用指令测试，确定指令测试结果正确后再进行汇编代码的测试。

其次，感受到了时序非常重要，在本次实验 CPU 设计中，在我看来只要不出现时序错误的问题，按照指令集完成一个 CPU 是比较简单的。但是如果时序错误，则需要花费巨大的精力从各方面着手查错并修改，十分锻炼综合能力。

最后，在实现 CPU 代码的过程中，学会使用一些版本控制方法，这样在修改代码之后发现错误时可以有效的退回前面的版本，避免了出现修改失误后再花大量的时间进行查错。

完成实验非常有利于我的成长。一个人在实验过程中踩遍无数问题，这些问题有他人留下的，有自己留下的，有软件下的。但是这一切都是非常值得的。在解决问题的过程中逐渐积累这方面的经验，可以为我以后从事相关行业打下坚实基础。