计算机组成原理

课程设计报告

学　　号_____21071003_____

姓　　名_____高立扬_____

指导教师_____高明霞_____

提交日期_____2023.7_____

成绩评价表

报告内容	报告结构	报告最终成绩
□丰富正确 □基本正确 □有一些问题 □问题很大	□完全符合要求 □基本符合要求 □有比较多的缺陷 □完全不符合要求	
报告与 Project 功能一致性	报告图表	总体评价
□完全一致 □基本一致 □基本不一致	□符合规范 □基本符合规范 □有一些错误 □完全不正确	

教师签字:_____

目录

Project1 VerilogHDL 完成单周期处理器开发

一. 总体数据通路结构设计图

Project1 单周期处理器的总体数据通路结构设计图如图 1.所示，本处理器参考了 Project1 任务书所给的图，支持 MIPS-Lite1 指令集{MIPS-Lite，addi，addiu， slt，jal，jr}(MIPS-Lite 指令集：addu，subu，ori，lw，sw，beq，lui，j)。本处理器的取指模块没有采用大作业中的 IFU，而是效仿多周期处理器，将 IFU 拆解，设计了 PC 计算模块、PC 模块和 IM 模块，这么做是为了方便做 project2 和 3 的时候进行模块迭代，避免大幅度修改增加 BUG 出现几率，节省 DEBUG 时间开销。图中 M1-3 为 MUX 选择模块，原本可以分别内置在 GPR 和 ALU 内，但是为了方便调试，单独拆成了三个模块，直观，且体现了模块化编程的好处，减少了 GPR 和 ALU 代码复杂度，更注重于其功能特点。

本 project 自主编写；在线下检查时，添加的指令为“LH”

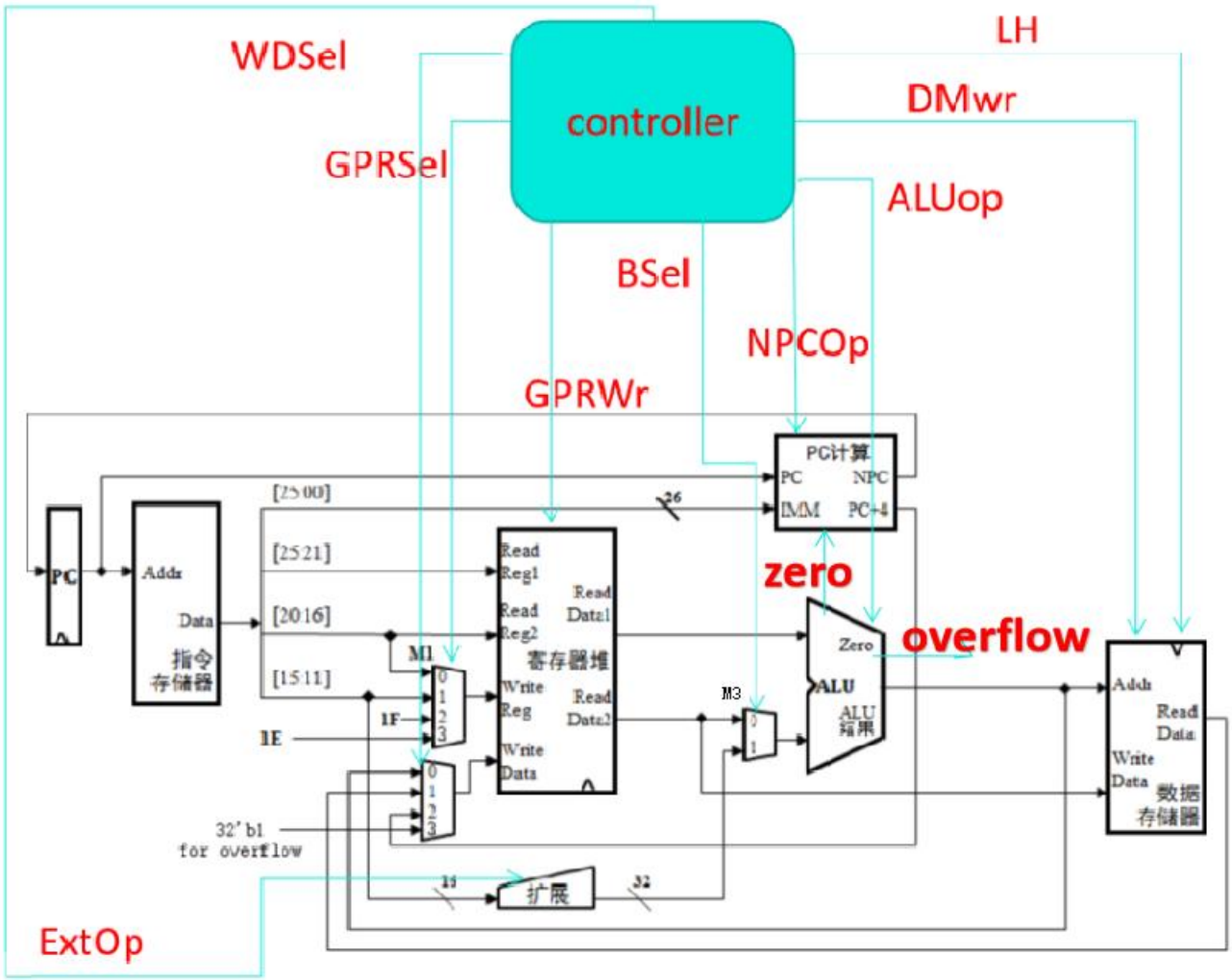

图 1.project1 总体数据通路图

二. 模块定义

2.1 PC 模块

2.1.1 基本描述

PC 模块是时序性模块，其功能为接收来自于 PC 计算模块输出的 NPC 信号，并在时钟信号 clk 上升沿时向外传输 NPC 输入的值

2.1.2 模块接口

信号名	方向	描述
clk	I	时钟信号
pcin[31:0]	I	来源于 PC 计算模块输入
pcout[31:0]	O	本模块输出值，和 PC 计算模块输入值的数值相等

2.1.3 功能定义

序号	功能名称	功能描述
1	取地址	当 clk 信号上升沿时，本模块将 PC 计算模块计算好并输入进来的 PC，输出出去（输出到 IM 模块）

2.2 IM 模块

2.2.1 基本描述

IM 模块是非时序的。其功能为读取事先写好的十六进制代码程序，存入内部的 1kb 指令寄存器，指令起始地址本应为 0000_3000h，但由于 1kb 的寄存器地址最大到 32 位地址的低 10 位，因此指令的起始地址在书写代码时可以等效为 0000_0000h。依据输入的 PC 地址信号，进行取指。小端序存储指令

2.2.2 模块接口

信号名	方向	描述

addr[9:0]	I	来源于 PC 模块的输出，也就是地址，由于 IM 模块存储指令的寄存器仅 1KB，因此只需要 PC 的低 10 位
dout[31:0]	O	本模块输出值，是地址所对应的 32 位指令

2.2.3 功能定义

序号	功能名称	功能描述
1	取指令	根据输入的地址的低 10 位，在本模块内部的指令寄存器里取得指令，并输出

2.3 PC 计算模块（以下简称 NPC 模块）

2.3.1 基本描述

NPC 模块是非时序性模块。本模块是推进 PC 地址更新的核心，它会根据当前指令的性质，判断并分析下一条指令的地址，传送给 PC 模块。对于 MIPS-Lite1 指令集，NPC 会根据 beq，j 和 jal 指令，jr 指令和此外的指令，进行四种地址分析策略，从而计算出下一条指令的地址。特别地，对于 jal 指令，NPC 模块还会输出 PC+4 以存入 31 号寄存器；对于 jr 指令，NPC 模块需要输入 GPR 输出的读取到 rs 寄存器的内容，便于跳转。

复位时，本模块锁定 PC 地址为 0000_3000h

2.3.2 模块接口

信号名	方向	描述
imm[25:0]	I	由 IM 输出的 32 位指令分线出低 26 位而来，用于计算 j 和 jal 指令跳转的地址，或计算 beq 跳转的地址
pcin[31:0]	I	来源于 PC 模块的输出，也就是接收目前 PC 地址，便于计算下一条地址
rd1[31:0]	I	来源于 GPR 模块的输出，其内容是 rs 寄存器内部存储的 32 位数据
npcop[1:0]	I	来源于 Controller 的输出，代表着本模块进行哪一种地址计算逻辑： 00：地址+4 推进 01：beq 指令跳转 10：j/jal 指令跳转

		11：jr 指令跳转
zero	I	来源于 ALU 的输出，用于判断 beq 指令条件，决定是否跳转，如果为 1 才会进行 beq 指令跳转，否则正常 pc+4 推进
rst	I	异步复位信号，为 1 时会将 PC 地址重置为 0000_3000h，直至 rst 信号重新变为 0
nextpc	O	计算好的 32 位地址，输出给 PC 模块
pc_4	O	输入的 pcin，在此基础上+4，专用于 jal 指令

2.3.3 功能定义

序号	功能名称	功能描述
1	异步复位	复位信号为 1 时，pcout 输出锁定为 0000_3000h
2	判断并计算	本模块会先计算出 j/jal 指令和 beq 指令跳转后的地址， beq 跳转：pc + 4 + imm<<2 j 跳转：{pc+4 的高四位, imm26, 00} 然后根据 npcop 进行如下操作： 00：输出 pc+4 01：如果 zero=1 输出 beq 跳转地址；否则 pc+4 10：j/jal 跳转地址 11：输出 rd1 信号内容
3	地址传递	计算好的新地址会传输给 PC 模块

2.4 GPR 模块

2.4.1 基本描述

本模块是时序性模块。但比较特殊的是，其数据输出并不受到时间信号的影响，仅存储数据时需要等待 clk 上升沿。本模块是 CPU 的寄存器堆，共有 32 个 32 位的寄存器，各司其职。其中 0 号寄存器恒为 0，无法被修改；31 号寄存器一般用来存储 jal 类指令所需要存储的地址，便于 jr 指令读取。

2.4.2 模块接口

信号名	方向	描述

clk	I	时钟信号
reset	I	复位信号
regwrite	I	来源于 Controller 模块，是寄存器的写入权限，1 代表可写
m1out[4:0]	I	来源于 M1 模块，是待写入数据的寄存器的编码
m2out[31:0]	I	来源于 M2 模块，是待写入 m1out 指定寄存器的数据
rs[4:0]	I	来源于 IM 模块的输出 dout[25:21]
rt[4:0]	I	来源于 IM 模块的输出 dout[20:16]
busa[31：0]	O	读出 rs 对应的寄存器的数值
busb[31:0]	O	读出 rt 对应的寄存器的数值

2.4.3 功能定义

序号	功能名称	功能描述
1	异步复位	复位信号为 1 时，32 个寄存器全置 0
2	写入数据	clk 为上升沿的时候，m2out 的内容存入 m1out 解码后对应的寄存器
3	读取数据	读取 IM 模块 dout 输出对应 dout[25:21]和[20:16]解码后对应的寄存器的数值并输出

2.5 EXT 模块

2.5.1 基本描述

本模块是非时序模块。本模块的功能是根据信号，对输入的 16 位数进行 32 位扩展并输出至目标模块 M3

2.5.2 模块接口

信号名	方向	描述
imm16[15	I	来源于 IM 模块的 dout 的低 16 位，代表着 I 型指令对

:0]		应的立即数
extop[1:0]	I	来源于 Controller 控制本模块扩展立即数的逻辑 00：无符号拓展 01：符号拓展 10：高位拓展（专用于 lui 指令）
out[31:0]	O	拓展完的数值进行输出

2.5.3 功能定义

序号	功能名称	功能描述
1	扩展并输出	对立即数进行相应的拓展并输出

2.6 DM 模块

2.6.1 基本描述

本模块是时序模块。本模块是专门存储数据的数据寄存器，容量为 1kb。当 clk 为上升沿的时候，允许写入数据。数据地址从 0000_0000h 开始。小端序存储数据

2.6.2 模块接口

信号名	方向	描述	
addr[9:0]	I	来源于 ALU 的输出，代表了本模块将要读写的地址	
din[31:0]	I	来源于 GPR 的 busb 输出，代表了本模块将要写入的值	
we	I	来源于 Controller 的输出，1 为 DM 可写	
clk	I	时钟信号	
lh	I	来源于 Controller 的输出，1 代表当前指令为 lh 指令	I
dout[31:0]	O	根据当前指令是否为 lh，决定本模块的输出 lh=0 根据 addr 输出 32 位数据 lh=1 根据 addr 输出 16 位数据符号扩展后的 32 位数据	

2.6.3 功能定义

序号	功能名称	功能描述
1	写入数据	写使能有效的时候，当 clk 为上升沿，根据地址进行数据

		写入
2	读取数据	根据地址和 lh 指令判断，进行数据读取并输出

2.7 ALU 模块

2.7.1 基本描述

本模块为非时序模块。本模块为运算模块，对输入的两个数，根据控制信号进行相应运算并进行输出。同时在特殊情况下会进行两数相等的判定，或两数相加溢出的判定，并输出对应的信号。

2.7.2 模块接口

信号名	方向	描述
A[31:0]	I	运算数 A
B[31:0]	I	来源于 M3 模块的输出，运算数 B
aluctr[2:0]	I	运算逻辑 000：加法 001：减法 010：按位或运算 011：(A 和 B 符号拓展) A < B ? 1 : 0，针对于 slt 指令 100：针对于 addi 指令的加法，会追加判断是否溢出
zero	O	针对于 beq 指令的信号，A=B 时输出 1，否则 0
overflow	O	针对于 addi 指令的信号，A+B 溢出时为 1，否则 0
out[31:0]	O	运算结果输出，为 A(运算逻辑)B

2.7.3 功能定义

序号	功能名称	功能描述
1	计算	根据 aluctr 进行相应逻辑的运算并输出结果
2	判断	针对于 beq、slt、addi 指令分别做出相应的判断，如上文所示

2.8 M1 模块

2.8.1 基本描述

本模块为非时序模块，用于 GPR 待写入寄存器编码的选择工作。

2.8.2 模块接口

信号名	方向	描述
gprsel[1:0]	I	来源于 Controller 模块的输出，用于选择 GPR 待写入寄存器的 5 位编码，控制着 m1out 信号的输出数值 00：rt 01：rd 10：1F 11：1E
rt[4:0]	I	来源于 IM 模块的输出，为 dout[20:16]
rd[4:0]	I	来源于 IM 模块的输出，为 dout[15:10]
m1out[4:0]	O	输出数值被 gprsel 控制

2.8.3 功能定义

序号	功能名称	功能描述
1	多路选择	选择 GPR 待写入寄存器编码

2.9 M2 模块

2.9.1 基本描述

本模块为非时序模块，用于 GPR 待写入数据的选择工作。

2.9.2 模块接口

信号名	方向	描述
wdsel[1:0]	I	来源于 Controller 模块的输出，用于选择 GPR 待写入寄数据，控制着 m2out 输出信号的值 00：aluo 01：dmout 10：pc_4 11：0000_0001h（专用于 addi 指令 overflow 的情况）
aluo[31:0]	I	来源于 IM 模块的输出，为 dout[20:16]
dmout[31	I	来源于 IM 模块的输出，为 dout[15:10]

:0]		
pc_4[31:0]	I	来源于 NPC 模块的输出，为 PC+4
m2out[31:0]	O	输出数值被 wdsel 控制

2.9.3 功能定义

序号	功能名称	功能描述
1	多路选择	选择 GPR 待写入数据

2.10 M3 模块

2.10.1 基本描述

本模块为非时序模块，用于 ALU 运算数 B 的选择工作

2.10.2 模块接口

信号名	方向	描述
bsel	I	来源于 Controller 模块的输出，用于 ALU 运算数 B 的选择工作，控制着 m3out 输出信号的值 0：bo 1：imm32
bo[31:0]	I	来源于 GPR 模块的输出，为 busb
imm32[31:0]	I	来源于 EXT 模块的输出，为 EXT 模块的 out
m3out[31:0]	O	输出数值被 bsel 控制

2.10.3 功能定义

序号	功能名称	功能描述
1	多路选择	ALU 运算数 B 的选择

2.11 Controller 模块

2.11.1 基本描述

本模块为非时序模块。本模块是整个 CPU 中的重中之重，犹如人之大脑一般重要。其功能为输入 opcode 和 funct，从而对指令进行译码，输出控制信号，控制几乎 CPU 中每一个部件，没有了本模块，CPU 中的各个模块只能是一盘散沙，可见其重要性所在

本模块中有一个没有用到的信号，名为 j，由于 controller 内部信号书写问题，删除此信号怕出现 bug，后续没有对 j 信号进行删除，予以保留（不影响模块正确执行所有规定指令），在此特殊说明

2.11.2 模块接口

信号名	方向	描述
opcode[5:0]	I	来源于 IM 模块的输出，dout 的高六位
funct[5:0]	I	来源于 IM 模块的输出，dout 的低六位
overflow	I	来源于 ALU 模块的输出，是对 addi 指令运算结果是否溢出的判断信号，1 代表溢出
j	O	废弃信号，怕删除出 bug，予以保留
aluop[2:0]	O	输出到 ALU 模块，控制运算逻辑
gprsel[1:0]	O	输出到 M1 模块，选择待写入数据的寄存器的五位编码
gprwr	O	输出到 GPR 模块，控制 GPR 的写入权限
extop[1:0]	O	输出到 EXT 模块，控制其拓展逻辑
wdsel[1:0]	O	输出到 M2 模块，选择写入 GPR 目标寄存器的数值
npcop[1:0]	O	输出到 NPC 模块，选择 PC 地址运算逻辑
dmwr	O	输出到 DM 模块，控制 DM 的写入权限
bsel	O	输出到 M3 模块，选择输入 ALU 作为运算数 B 的数值
lh	O	输出到 DM 模块，选择 dmout 输出的数值

2.11.3 功能定义

序号	功能名称	功能描述
1	译码产生控制信号	根据 opcode 和 funct 进行译码，获得上文所示的各种重要控制信号

三. MIPS-Lite1 指令集+LH 指令 机器指令描述

表 1.MIPS-LITE1 指令+LH 指令

序号	助记符	opcode	funct	指令功能	
1	addu	000000	100001	两个来自寄存器的数无符号加法	GPR[rd] ← GPR[rs] + GPR[rt]
2	subu	000000	100011	两个来自寄存器的数无符号减法	GPR[rd] ← GPR[rs] - GPR[rt]
3	ori	001101	-	两个来自寄存器的数逻辑或运算	GPR[rt] ← GPR[rs] or immediate
4	lw	100011	-	读取 DM 中一个字到指定寄存器中	GPR[rt] ← memory[GPR[base] + offset]
5	sw	101011	-	指定寄存器中的字内容写到 DM 中	memory[GPR[base] + offset] ← GPR[rt]
6	beq	000100	-	两数相等则跳转	if GPR[rs] = GPR[rt] then branch
7	lui	001111	-	16 位立即数高位拓展	GPR[rt] ← immediate || 016
8	j	000010	-	无条件跳转	PC ← PCGPRLEN-1..28 || instr_index || 02

9	addi	001000	-	加立即数（符号拓展，有溢出检验）	temp ← (GPR[rs]31||GPR[rs]31..0) + sign_extend(immediate)
					if temp32 ≠ temp31 then
					SignalException(IntegerOverflow)
					else
					GPR[rt] ← temp
					endif

10	addiu	001001	-	加立即数（符号拓展，无溢出检验）	GPR[rt] ← GPR[rs] + immediate
11	slt	000000	101010	小于则置 1	GPR[rd] ← (GPR[rs] < GPR[rt])

12	jal	000011	-	跳转并链接（PC+4 存到$31）	GPR[31]← PC + 8 PC ← PCGPRLEN-1..28 || instr_index || 02
13	jr	000000	001000	跳转到寄存器存储的地址	PC ← GPR[rs]
14	lh	100001	-	读取 DM 中半个字（符号拓展存到 GPR 寄存器）	GPR[rt] ← memory[GPR[base] + offset]

四．测试程序

4.1 MIPS-LITE1 指令集测试程序

```
# 变量初始化工作
ori $16, $0, 1          # 【$16 = 1】
ori $17, $0, 3          # 【$17 = 3】
ori $8, $0, 1           # 【$8 = 1】
ori $12, $0,0xabab      # 【$12 = abab】
lui $13, 10             # 【$13 = 10_0000】

# 函数start开始
start:addu $4, $0,$16   # 【$4 = (unsigned)$0 + $16】
addu $5, $0,$8          # 【$5 = (unsigned)$0 + $8】
jal newadd              # 【函数调用 $31 = 3020;jmp to 30c8】
addu $16, $0, $2        # 【函数返回 (unsigned)$16 = $0 + $2】
subu $17,$17,$8         # 【(unsigned) $17 = $17 - $ 8】1
beq $16, $17, start     # 【beq $16 == $17 ? jmp to 3014 : jmp to 302c】
# 函数start结束

# 函数start2前准备工作
ori $8, $0,4            # 【$8 = bddf】
addiu $24,$0,0x7fffffff # 【$24 = INTmax】
addiu $9,$24,3          # 【$9 = 1000_0002】
addiu $10,$24,5         # 【$10 = 1000_0004】
addu $0,$0,$0           # 【$0 *= 2】
addi $22,$24,6          # 【$22溢出,$30 = 1;$22 = 0】 ################
#ori $0, $0, 0
```

```
26 # 函数start2开始
27 start2:sw $9, 0($8)      # 【gpr[$8] = $9】以8号寄存器中的内容为基地址，加上偏移量0后的数据存储器地址的存储单元，存9号寄存器内容
28 lw $14, 0($8)           # 【$14 = gpr[$8]】以8号寄存器中的内容为基地址，加上偏移量0后 的数据存储器地址的存储单元的内容，存入14号寄存器
29 sw $10,4($8)            # 【gpr[$8 + 4] = $10】以8号寄存器的内容为基地址，加上偏移量4后的数据存储器地址的存储单元，存10号寄存器内容
30 lw $15,4($8)            # 【$15 = gpr[$8 + 4]】以8号寄存器的内容为基地址，加上偏移量4后的数据存储器地址的存储单元的内容，存入15号寄存器
31 sw $4, -4($8)           # 【gpr[$8 - 4] = $4】以8号寄存器的内容为基地址，加上偏移量-4后的数据存储器地址的存储单元，存4号寄存器内容
32 lw $18, -4($8)          # 【$18 = gpr[$8 - 4]】以8号寄存器的内容为基地址，加上偏移量-4后的数据存储器地址的存储单元的内容，存入18号寄存器
33 addu $4,$0,$8           # 【$4 = $0 + $8】8号寄存器内容 与 0号寄存器内容 进行无符号加法，结果存入4号寄存器
34 addu $5,$0,$9           # 【$5 = $0 + $9】9号寄存器内容 与 0号寄存器内容 进行无符号加法，结果存入5号寄存器
35 jal newadd              # 【函数调用】
36 slt $25,$10,$8          # 如果10号寄存器内容 小于 8号寄存器内容， 25号寄存器内容置1， 否则置0
37 beq $25, $0,end2        # 【程序终止判断】比较25号寄存器 与 0号寄存器内容（0）， 相等则跳转end2处，否则顺序执行
38 slt $20,$12,$4          # 如果12号寄存器内容 小于 4号寄存器内容， 20号寄存器内容置1， 否则置0
39 beq $20, $0, end1       # 【本函数结束判断】比较20号寄存器 与 0号寄存器内容（0）， 相等则跳转end1处，否则顺序执行
40 lui $12, 65535          # 将十进制65535(1_0000)加载到32位的高16位，低16位补0， 扩展结果存入12号寄存器(1000_000)
41 end1:ori $0, $0,1       # 0号寄存器内容 与 1 相或， 结果存0号寄存器
42 # 函数start2结束
43
44 # 函数start3前准备工作
45 lui $19, 0xefef          # 十六进制数efef加载至32位中的高16位，低位补0， 结果存入19号寄存器
46 addiu $3,$0,0xababcdcd  # 0号寄存器内容 无符号加立即数 ababcdcd（十六进制）， 结果存入3号寄存器中
```

```
48 # 函数start3开始
49 start3:addiu $4, $3, 2  # 3号寄存器内容 无符号加立即数2， 结果存入4号寄存器中
50 addi $23, $3, 5         # 3号寄存器内容 加 5， 有符号数的加法， 结果存23号寄存器，若结果溢出，30号寄存器存1，否则为0
51 jal newadd              # 将下一条指令地址存入31号寄存器， 跳转到newadd处
52 addu $8, $0, $2         # 0号寄存器内容 与 2号寄存器内容 进行无符号加法， 结果存8号寄存器
53 addu $4, $0, $8         # 0号寄存器内容 与 8号寄存器内容 进行无符号加法， 结果存4号寄存器
54 addu $5, $0, $9         # 0号寄存器内容 与 9号寄存器内容 进行无符号加法， 结果存5号寄存器
55 jal newadd              # 将下一条指令地址存入31号寄存器， 跳转到newadd处
56 addu $9, $0, $2         # 将2号寄存器内容与0号寄存器内容进行无符号加法，结果存入9号寄存器
57 addu $9, $8, $0         # 将0号寄存器内容与8号寄存器内容进行无符号加法，结果存入9号寄存器
58 lui $10, 0x69           # 将十六进制数69加载到32位数的高16位，低位补0，结果存入10号寄存器
59 beq $8, $9, start4      # 比较8号寄存器 和 9号寄存器的内容， 相等跳转start4处，否则顺序执行
60 beq $0, $0, start3      # 比较0号寄存器 和 0号寄存器的内容， 相等跳转start3处，否则顺序执行
61 start4:
62 j end          # 【程序结束】
63
64 # 【函数newadd】
65 newadd:addu $2, $4, $5  # 【函数进入 $2 = $4 + $5】4号寄存器内容 无符号加 5号寄存器内容， 结果存入2号寄存器中
66 addi $0,$12,0x1234      # 【重定义0 $0 = $12 + 1234】十六进制数1234 和12号寄存器的内容 有符号加法， 结果存0寄存器（0寄存器理论上无法被写入）
67 jr $31                  # 【函数返回 jmp $31】跳转到 31号寄存器存储的内容 所指示的地址处
68 # 【函数末尾】
69
70 end2:addi $26,$0,0x5678 # 十六进制数5678 和 0号寄存器内容 有符号加法， 结果存26号寄存器
71 end:                    # end标志处
```

图 2-4.针对于 MIPS-LITE1 指令集的测试程序

4.2 新增 LH 指令测试程序

```
63 start4:
64 lh $22, 8($0)
65 j end          # 【程序结束】
```

图 5.新增指令测试程序

说明：在 4.1 所示程序（对应提交文件中的 p1-test2.asm）的第 64 行新增 lh $22,8($0)指令，其余指令不变。对于读取半个字的两种情况（高 16 位或低 16 位），设计了两套测试程序，第二套程序即“lh $22,6($0)”，offset 为 8 对应读取 DM 第三个字的低半字，offset 为 6 对应读取 DM 第二个字的高半字。两个测试程序分别对应提交文件中的 p1-test-lh.asm 和 p1-test-lh2.asm。后续结果测试时，只需关注 22 号寄存器存储的数值是否正确即可

五．测试结果

5.1 MARS 中测试结果

Name	Number	Value
$zero	0	0x00000000
$at	1	0xababcdcd
$v0	2	0xababcdd3
$v1	3	0xababcdcd
$a0	4	0x2babcdd1
$a1	5	0x80000002
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x2babcdd1
$t1	9	0x2babcdd1
$t2	10	0x00690000
$t3	11	0x00000000
$t4	12	0x0000abab
$t5	13	0x000a0000
$t6	14	0x80000002
$t7	15	0x80000004
$s0	16	0x00000003
$s1	17	0x00000001
$s2	18	0x00000002
$s3	19	0xefef0000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0xababcdd2
$t8	24	0x7fffffff
$t9	25	0x00000001
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x00001800
$sp	29	0x00002ffc
$fp	30	0x00000000
$ra	31	0x000030b4
pc		0x000030dc
hi		0x00000000
lo		0x00000000

图 6.MARS 中 GPR 最终结果

Value (+0)	Value (+4)	Value (+8)
0x00000002	0x80000002	0x80000004

图 7.MARS 中 DM 最终结果

特殊说明：addi 指令运算结果若溢出，在 MARS 中会报错导致程序无法跑完，因此临时替换 addi 语句为“ori $0, $0, 0”可以在不影响所有 GPR 和 DM 数据的情况下跑通程序，图 6.的结果按 MIPS-LITE1 的要求，$30 最终存储的值应当为 0000_0001，这将在下文着重验证

$28 和$29 分别为 global pointer 和 stack pointer，在 MARS 运行时会有数值，而对于 MIPS-LITE1 简单指令集，不会涉及到这两个寄存器的写入访问，因此在 modelsim 波形仿真时，这两个寄存器的最终结果应当为 0

5.2 modelsim 验证结果

5.2.1 波形图（重点检验溢出）

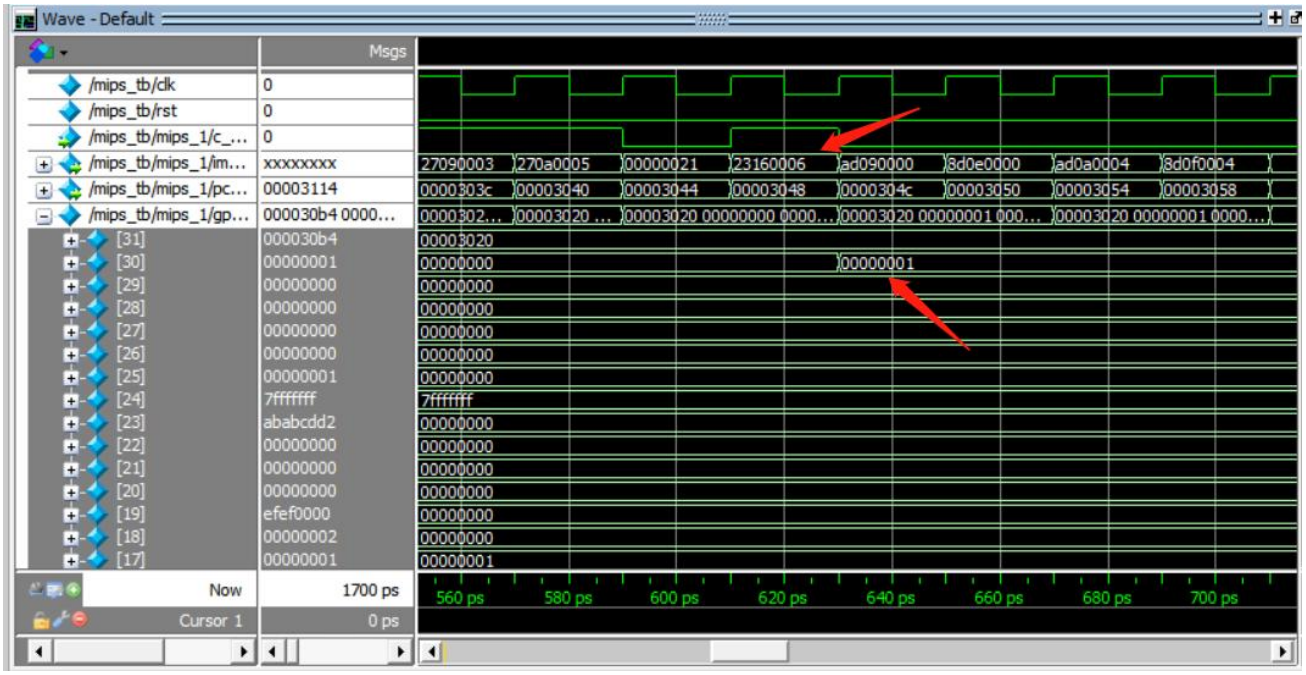

图 8.overflow 的判断和$30 存储 0000_0001h 数据

如图 8.所示，23160006 正是指令“addi $22,$24,6”，由图可知，执行本指令的时候 24 号寄存器存储的是 32 位二进制数正数最大值，+6 肯定会发生溢出，而后续的波形表示，不仅判断出了指令的溢出，而且还将 0000_0001h 存入了 30 号寄存器

5.2.2 DM 和 GPR 最终结果

```
0000001f 000030b4 00000001 00000000 00000000 00000000 00000000 00000001 7fffffff ababcdd2 00000000 00000000 00000000
00000013 efef0000 00000002 00000001 00000003 80000004 80000002 000a0000 0000abab 00000000 00690000 2babcdd1 2babcdd1
00000007 00000000 00000000 80000002 2babcdd1 ababcdcd ababcdd3 ababcdcd 00000000
```

图 9.modelsim 中 GPR 最终结果

```
00000057 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000033 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000f 00 00 00 00 80 00 00 04 80 00 00 02 00 00 00 02
```

图 10.modelsim 中 DM 最终结果

结果正确，说明 CPU 设计合理，已经能正确执行 MIPS-LITE1 指定的所有指令

5.3 新增指令测试

5.3.1 MARS 中测试结果

$s6	22	0x00000004
$s6	22	0xffff8000

图 11-12.MARS 中 22 号寄存器存储结果，分别对应 offset=8 和 6

5.3.2 modelsim 中测试结果

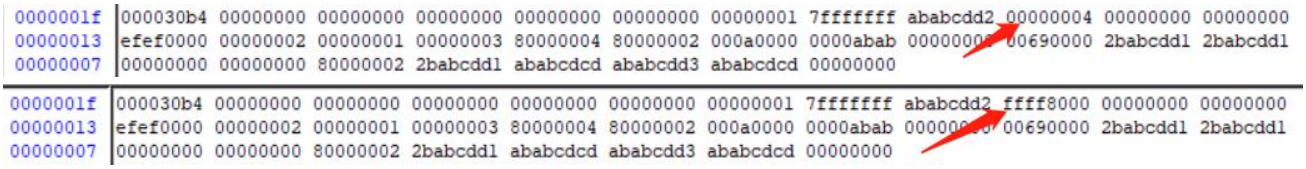

图 13-14.modelsim 中 22 号寄存器的存储结果

结果正确，说明针对新指令的设计合理，能正确执行

六．Project1 完成后的心得体会

这是自数字逻辑实验以来，又一次使用 Verilog 语言。因为有大作业的基础，project1 的编写并不难，再加上有学习通中的教程，我很快就写完了所有模块的代码，学会了用 testbench，但是调试耗费了我很多的时间。经过数小时的调试，我将常见错误归纳为：Controller 控制信号判断和编写有误、npc 指令传递错误、顶层连线错误（包括信号连错和位宽错误）、testbench 中信号书写有误。在调试的时候，我掌握了很多诀窍和方法，使得我能快速定位到错误所在，在后续两个 project 中节省了不少的时间。

起初，我本来设计好了 IFU 模块，但是想到后续两个 project 是多周期，又看到 project1 的例图中的模块也和多周期的模块极度相似，我就尝试将 IFU 拆开进行编写，并自主设计了 M1-3 模块，体验到了模块化对顶层连线、代码理解和调试带来的好处。进行设计的时候，我还手画了顶层设计图，进一步加深了对单周期数据通路的理解。

在检查作业时，通过老师的提示，我发现了针对 addi 指令进行$30 置 1 时，只需要将信号传入 GPR，在存储的时候进行 if else 判断即可完成任务，并不需要单独在 M1 和 M2 模块占用一个多路选择信号，还要在 controller 里判断多个控制信号的逻辑，十分占用资源。不过我在编写 GPR 的时候，因为配合 M1 和 M2 的模块化设计，导致了不好按照上述思路单独修改 GPR，况且通过溢出信号输入 GPR 的方法来决定存储加法结果还是溢出后的 1，本质上也是一个二路选择器，所以我对原来的代码予以保留，没有进行修改，但是我已经理解了上述的思路，并且在 project2 检查的时候正确复述了出来。

本次添加指令环节，我的任务是添加 LH 指令，一开始因为紧张，我一时不知道应该从哪个模块开始入手修改，后来我开始对 Controller 进行分析，并画图辅助理解，快速完成了模块修改。接下来就要进行测试程序编写了，编写完成后导出指令并进行验证，我用时 23 分钟及时完成了任务，并在课后及时归纳了添加指令的分析步骤，便于后续 project2 检查节省时间。

Project2 VerilogHDL 完成多周期处理器开发

一．总体数据通路结构设计图

project2 的总体数据通路结构设计如图 15.。本处理器参考了 Project2 任务书所给的图，支持 MIPS-Lite2 指令集{MIPS-Lite1，lb，sb}。本处理器的模块完美继承了 project1 中的模块，使

得在进行多周期模块修改时，不仅方便且节省时间，而且尽最大可能减少了代码修改行数，进而避免了修改代码导致不必要的 BUG 出现，节省了 DEBUG 时间开销。在继承 p1 中的模块后，任务只剩下了修改个别模块（只需添加几行代码），新增图中的时序性寄存器模块，调整顶层连接，最后调试即可。状态机参考的同样是 PPT 当中的 10 个状态设计，将在下文 Controller 模块着重介绍。

本 project 自主编写；在线下检查时，添加的指令为“JALR”

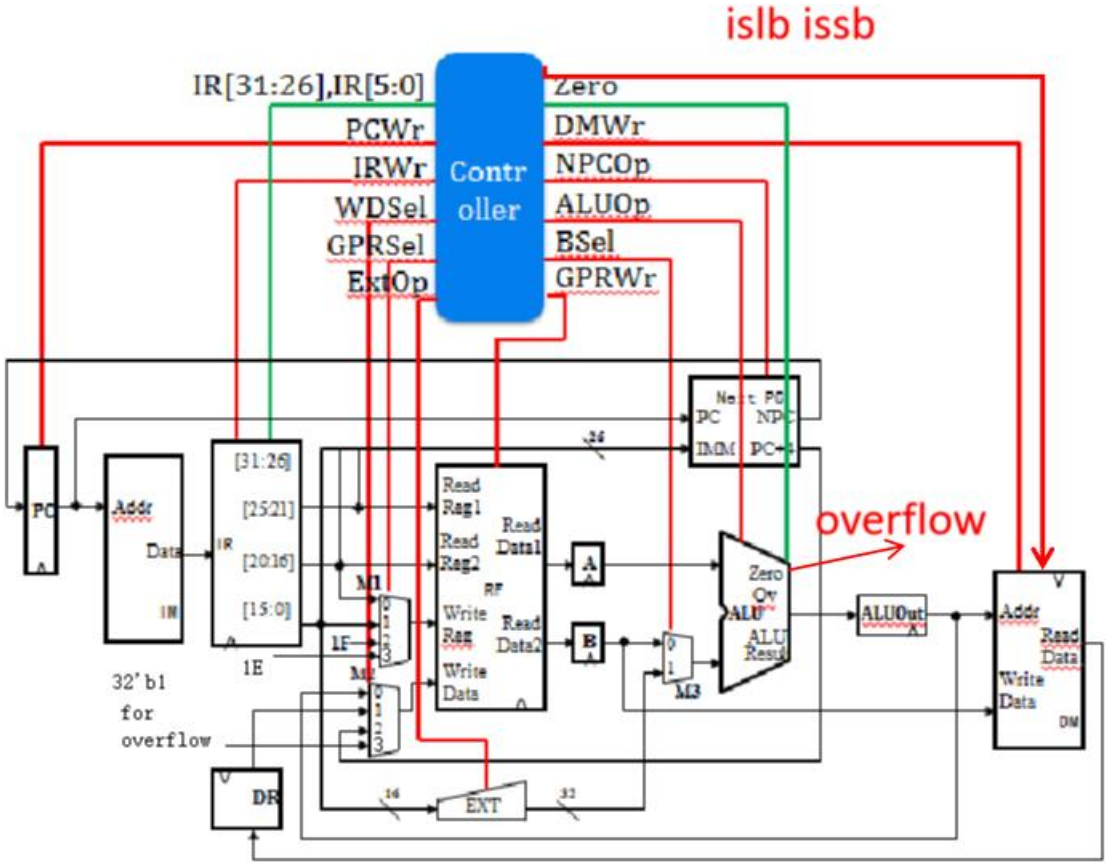

图 15.project2 总体数据通路结构设计图

二. 模块定义

2.1 PC 模块

2.1.1 基本描述

PC 模块是时序性模块，其功能为接收来自于 PC 计算模块输出的 NPC 信号，并在时钟信号 clk 上升沿时向外传输 NPC 输入的值

2.1.2 模块接口

信号名	方向	描述

clk	I	时钟信号
pcin[31:0]	I	来源于 PC 计算模块输入
pcwr	I	PC 模块写入权限，1 为可写
pcout[31:0]	O	本模块输出值，和 PC 计算模块输入值的数值相等

2.1.3 功能定义

序号	功能名称	功能描述
1	取地址	当 clk 信号上升沿时，如果 PC 可写，本模块将 NPC 模块计算好并输入进来的 PC，输出出去（输出到 IM 模块）

2.2 IM 模块

2.2.1 基本描述

IM 模块是非时序的。其功能为读取事先写好的十六进制代码程序，存入内部的 1kb 指令寄存器，指令起始地址本应为 0000_3000h，但由于 1kb 的寄存器地址最大到 32 位地址的低 10 位，因此指令的起始地址在书写代码时可以等效为 0000_0000h。依据输入的 PC 地址信号，进行取指。小端序存储指令

2.2.2 模块接口

信号名	方向	描述
addr[9:0]	I	来源于 PC 模块的输出，也就是地址，由于 IM 模块存储指令的寄存器仅 1KB，因此只需要 PC 的低 10 位
dout[31:0]	O	本模块输出值，是地址所对应的 32 位指令

2.2.3 功能定义

序号	功能名称	功能描述
1	取指令	根据输入的地址的低 10 位，在本模块内部的指令寄存器里取得指令，并输出

2.3 NPC 模块

2.3.1 基本描述

NPC 模块是非时序性模块。本模块是推进 PC 地址更新的核心，它会根据当前指令的性质，判断并分析下一条指令的地址，传送给 PC 模块。对于 MIPS-Lite2 指令集，NPC 会根据 beq，j 和 jal 指令，jr 和 jalr 指令和此外的指令，进行四种地址分析策略，从而计算出下一条指令的地址。特别地，对于 jal 指令，NPC 模块还会输出 PC+4 以存入 31 号寄存器；对于 jr 和 jalr 指令，NPC 模块需要输入 GPR 输出的读取到 rs 寄存器的内容，便于跳转。

值得一提的是，本 CPU 为多周期，按照 10 个状态设计状态机，因此第 n 周期读取到的指令将会在第 n+1 个周期进行译码和计算等工作，也就是延后一周期，但是 NPC 对 PC 的推进并没有延后一周期。这导致了在进行 jal 和 jalr 指令时，如果 PC_4 信号还按照 PC+4 来计算的话，$31 存入的地址实际上为 PC+8，因为指令的执行实际上比 PC 的推进滞后。解决这种错误的办法是 pc_4 传递 pc 而不是 pc+4

复位时，本模块锁定 PC 地址为 0000_3000h

2.3.2 模块接口

信号名	方向	描述
imm[25:0]	I	由 IM 输出的 32 位指令分线出低 26 位而来，用于计算 j 和 jal 指令跳转的地址，或计算 beq 跳转的地址
pcin[31:0]	I	来源于 PC 模块的输出，也就是接收目前 PC 地址，便于计算下一条地址
rd1[31:0]	I	来源于 GPR 模块的输出，其内容是 rs 寄存器内部存储的 32 位数据
npcop[1:0]	I	来源于 Controller 的输出，代表着本模块进行哪一种地址计算逻辑： 00：传输 PC 01：beq 指令跳转 10：j/jal 指令跳转 11：jr 和 jalr 指令跳转
zero	I	来源于 ALU 的输出，用于判断 beq 指令条件，决定是否跳转，如果为 1 才会进行 beq 指令跳转，否则正常 pc+4 推进
rst	I	异步复位信号，为 1 时会将 PC 地址重置为 0000_3000h，直至 rst 信号重新变为 0
nextpc	O	计算好的 32 位地址，输出给 PC 模块
pc_4	O	输入的 pcin，在此基础上+4，专用于 jal 指令

2.3.3 功能定义

序号	功能名称	功能描述
1	异步复位	复位信号为 1 时，pcout 输出锁定为 0000_3000h
2	判断并计算	本模块会先计算出 j/jal 指令和 beq 指令跳转后的地址， beq 跳转：pc + 4 + imm<<2 j 跳转：{pc+4 的高四位, imm26, 00} 然后根据 npcop 进行如下操作： 00：输出 pc 01：如果 zero=1 输出 beq 跳转地址；否则 pc+4 10：j/jal 跳转地址 11：输出 rd1 信号内容
3	地址传递	计算好的新地址会传输给 PC 模块

2.4 GPR 模块

2.4.1 基本描述

本模块是时序性模块。但比较特殊的是，其数据输出并不受到时间信号的影响，仅存储数据时需要等待 clk 上升沿。本模块是 CPU 的寄存器堆，共有 32 个 32 位的寄存器，各司其职。其中 0 号寄存器恒为 0，无法被修改；31 号寄存器一般用来存储 jal 类指令所需要存储的地址，便于 jr 指令读取。

2.4.2 模块接口

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号
regwrite	I	来源于 Controller 模块，是寄存器的写入权限，1 代表可写
m1out[4:0]	I	来源于 M1 模块，是待写入数据的寄存器的编码
m2out[31:0]	I	来源于 M2 模块，是待写入 m1out 指定寄存器的数据
rs[4:0]	I	来源于 IR 模块的输出 irout[25:21]
rt[4:0]	I	来源于 IR 模块的输出 irout[20:16]

busa[31：0]	O	读出 rs 对应的寄存器的数值
busb[31:0]	O	读出 rt 对应的寄存器的数值

2.4.3 功能定义

序号	功能名称	功能描述
1	异步复位	复位信号为 1 时，32 个寄存器全置 0
2	写入数据	clk 为上升沿的时候，m2out 的内容存入 m1out 解码后对应的寄存器
3	读取数据	读取 IR 模块 irout 输出对应 irout[25:21]和[20:16]解码后对应的寄存器的数值并输出

2.5 EXT 模块

2.5.1 基本描述

本模块是非时序模块。本模块的功能是根据信号，对输入的 16 位数进行 32 位扩展并输出至目标模块 M3

2.5.2 模块接口

信号名	方向	描述
imm16[15:0]	I	来源于 IR 模块的 irout 的低 16 位，代表着 I 型指令对应的立即数
extop[1:0]	I	来源于 Controller 控制本模块扩展立即数的逻辑 00：无符号拓展 01：符号拓展 10：高位拓展（专用于 lui 指令）
out[31:0]	O	拓展完的数值进行输出

2.5.3 功能定义

序号	功能名称	功能描述
1	扩展并输出	对立即数进行相应的拓展并输出

2.6 DM 模块

2.6.1 基本描述

本模块是时序模块。本模块是专门存储数据的数据寄存器，容量为 1kb。当 clk 为上升沿的时候，允许写入数据。数据地址从 0000_0000h 开始。小端序存储数据

根据 islb 和 issb 信号会进行字节读写，由于 project3 没有再新增针对于 CPU 内部的指令，而是新增了三个和外设交互的指令，因此在 sb 存储环节没有单独设计新的多路选择模块，而是选择集成在了模块代码内部

2.6.2 模块接口

信号名	方向	描述
addr[9:0]	I	来源于 ALU 的输出，代表了本模块将要读写的地址
din[31:0]	I	来源于 GPR 的 busb 输出，代表了本模块将要写入的值
we	I	来源于 Controller 的输出，1 为 DM 可写
clk	I	时钟信号
islb	I	来源于 Controller 的输出，1 代表当前指令为 lb 指令
issb	I	来源于 Controller 的输出，1 代表当前指令为 sb 指令
dout[31:0]	O	输出数据

2.6.3 功能定义

序号	功能名称	功能描述
1	写入数据	写使能有效的时候，当 clk 为上升沿，根据 sb 指令判断，再根据地址进行数据写入
2	读取数据	根据地址和 lb 指令判断，进行数据读取并输出

2.7 ALU 模块

2.7.1 基本描述

本模块为非时序模块。本模块为运算模块，对输入的两个数，根据控制信号进行相应运算并进行输出。同时在特殊情况下会进行两数相等的判定，或两数相加溢出的判定，并输出对应的信号。

2.7.2 模块接口

信号名	方向	描述
A[31:0]	I	运算数 A
B[31:0]	I	来源于 M3 模块的输出，运算数 B
aluctr[2:0]	I	运算逻辑 000：加法 001：减法 010：按位或运算 011：(A 和 B 符号拓展) A < B ? 1 : 0，针对于 slt 指令 100：针对于 addi 指令的加法，会追加判断是否溢出
zero	O	针对于 beq 指令的信号，A=B 时输出 1，否则 0
overflow	O	针对于 addi 指令的信号，A+B 溢出时为 1，否则 0
out[31:0]	O	运算结果输出，为 A(运算逻辑)B

2.7.3 功能定义

序号	功能名称	功能描述
1	计算	根据 aluctr 进行相应逻辑的运算并输出结果
2	判断	针对于 beq、slt、addi 指令分别做出相应的判断，如上文所示

2.8 M1 模块

2.8.1 基本描述

本模块为非时序模块，用于 GPR 待写入寄存器编码的选择工作。

2.8.2 模块接口

信号名	方向	描述
gprsel[1:0]	I	来源于 Controller 模块的输出，用于选择 GPR 待写入寄存器的 5 位编码，控制着 m1out 信号的输出数值 00：rt 01：isjalr==1 ？ $31 ： rd 10：1F 11：1E
isjalr	I	来源于 Controller 的输出，1 代表执行的是本条指令
rt[4:0]	I	来源于 IM 模块的输出，为 dout[20:16]
rd[4:0]	I	来源于 IM 模块的输出，为 dout[15:10]
m1out[4:0]	O	输出数值被 gprsel 控制

2.8.3 功能定义

序号	功能名称	功能描述
1	多路选择	选择 GPR 待写入寄存器编码

2.9 M2 模块

2.9.1 基本描述

本模块为非时序模块，用于 GPR 待写入数据的选择工作。

2.9.2 模块接口

信号名	方向	描述
wdsel[1:0]	I	来源于 Controller 模块的输出，用于选择 GPR 待写入寄数据，控制着 m2out 输出信号的值 00：aluo 01：dmout 10：pc_4 11：0000_0001h（专用于 addi 指令 overflow 的情况）
aluo[31:0]	I	来源于 IM 模块的输出，为 dout[20:16]

dmout[31:0]	I	来源于 IM 模块的输出，为 dout[15:10]
pc_4[31:0]	I	来源于 NPC 模块的输出，为 PC+4
m2out[31:0]	O	输出数值被 wdsel 控制

2.9.3 功能定义

序号	功能名称	功能描述
1	多路选择	选择 GPR 待写入数据

2.10 M3 模块

2.10.1 基本描述

本模块为非时序模块，用于 ALU 运算数 B 的选择工作

2.10.2 模块接口

信号名	方向	描述
bsel	I	来源于 Controller 模块的输出，用于 ALU 运算数 B 的选择工作，控制着 m3out 输出信号的值 0：bo 1：imm32
bo[31:0]	I	来源于 GPR 模块的输出，为 busb
imm32[31:0]	I	来源于 EXT 模块的输出，为 EXT 模块的 out
m3out[31:0]	O	输出数值被 bsel 控制

2.10.3 功能定义

序号	功能名称	功能描述

1	多路选择	ALU 运算数 B 的选择

2.11 Controller 模块

2.11.1 基本描述

本模块为非时序模块。本模块是整个 CPU 中的重中之重，犹如人之大脑一般重要。其功能为输入 opcode 和 funct，从而对指令进行译码，输出控制信号，控制几乎 CPU 中每一个部件，没有了本模块，CPU 中的各个模块只能是一盘散沙，可见其重要性所在

本模块针对于多周期 CPU 新增了不少控制信号，并且删除了 project1 中未删除的 j 信号；同时加入了本 project 的重中之重：状态机。本 CPU 采用 10 个状态的状态机如图 16.

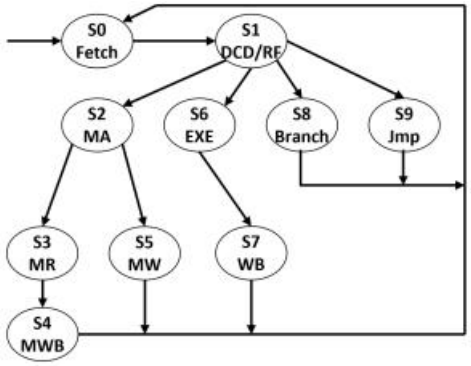

图 16.状态图

S0 为取指，S1 为译码，是 MIPS-LITE2 指令集中每一条指令必经之路。

S2 为针对于 DM 读写相关指令的入口，也就是 lw,lb,sw,sb；其中 lw 和 lb 会进入 S3 状态先对 DM 进行读取，之后进入 S4 状态进行 GPR 回写；sw 和 sb 会进入 S5 状态访存 DM

S6 为针对于 ALU 运算相关指令的入口，也就是 addu,subu,ori,addi,addiu,lui,slt，运算后会进入 S7 状态进行 GPR 回写

S8 为针对于分支相关指令的入口，也就是 beq 指令

S9 为针对于 j 型指令的入口，也就是 j,jal,jr,jalr 指令，本状态跳转和回写同时进行

由图可知，S4,S5,S7,S8,S9 执行完毕后，重回 S0 状态取指，完成一个周期

rst 复位会让状态机的状态锁定到 S0

2.11.2 模块接口

信号名	方向	描述
opcode[5:0]	I	来源于 IM 模块的输出，dout 的高六位
funct[5:0]	I	来源于 IM 模块的输出，dout 的低六位
overflow	I	来源于 ALU 模块的输出，是对 addi 指令运算结果是否溢出的判断信号，1 代表溢出
clk	I	时钟信号

rst	I	复位信号
zero	I	来源于 ALU 模块的输出，代表了两数是否相等
aluop[2:0]	O	输出到 ALU 模块，控制运算逻辑
gprsel[1:0]	O	输出到 M1 模块，选择待写入数据的寄存器的五位编码
gprwr	O	输出到 GPR 模块，控制 GPR 的写入权限
extop[1:0]	O	输出到 EXT 模块，控制其拓展逻辑
wdsel[1:0]	O	输出到 M2 模块，选择写入 GPR 目标寄存器的数值
npcop[1:0]	O	输出到 NPC 模块，选择 PC 地址运算逻辑
dmwr	O	输出到 DM 模块，控制 DM 的写入权限
bsel	O	输出到 M3 模块，选择输入 ALU 作为运算数 B 的数值
pcwr	O	输出到 PC 模块，决定 PC 是否写入来自 NPC 的输入
irwr	O	输出到 IR 模块，决定 IR 是否写入来自 IM 的输入
islb	O	输出到 DM 模块，决定 DM 的输出
issb	O	输出到 DM 模块，决定 DM 的写入
isjalr	O	输出到 M1 模块，参与 M1 的多路选择

2.11.3 功能定义

序号	功能名称	功能描述
1	译码产生控制信号	根据 opcode 和 funct 以及 zero 和 overflow 进行译码，获得上文所示的各种重要控制信号
2	状态机推进	推进状态机，完成多周期
3	复位	重置状态机

2.12 ALUOUT 模块

2.12.1 基本描述

本模块为时序性模块。本模块接收 ALU 的输出结果，当时钟上升沿时，输出相应结果

2.12.2 模块接口

信号名	方向	描述

clk	I	时钟信号
aluoutin[31:0]	I	来源于 ALU 模块的输出，是两数相运算的结果
aluoutout[31:0]	O	输出 aluoutin

2.12.3 功能定义

序号	功能名称	功能描述
1	传递数据	clk 为上升沿的时候，传递 ALU 的输出

2.13 AR 模块 & BR 模块

2.13.1 基本描述

AR 和 BR 模块均为时序性模块，类似于 ALUOUT 模块，也是传递信息用的，传递的是 GPR 读取到的 busa 和 busb。由于二者性质完全相同，仅仅模块名和信号名为了区分才不同，因此下面只介绍 AR，BR 同理

2.13.2 模块接口

信号名	方向	描述
clk	I	时钟信号
arin[31:0]	I	来源于 GPR 模块的输出，也就是 busa
arout[31:0]	O	输出 arin

2.13.3 功能定义

序号	功能名称	功能描述
1	传递数据	clk 为上升沿的时候，传递 GPR 的输出

2.14 DR 模块

2.14.1 基本描述

DR 模块为时序性模块，用于传递 DM 的输出

2.14.2 模块接口

信号名	方向	描述
clk	I	时钟信号
drin[31:0]	I	来源于 DM 模块的输出，也就是 dmout
drout[31:0]	O	输出 dmout

2.14.3 功能定义

序号	功能名称	功能描述
1	传递数据	clk 为上升沿的时候，传递 DM 的输出

2.15 IR 模块

2.15.1 基本描述

IR 模块为时序性模块，用于传递 IM 的输出

2.15.2 模块接口

信号名	方向	描述
clk	I	时钟信号
irin[31:0]	I	来源于 IM 模块的输出，也就是 dout
irwr	I	写入权限
irout[31:0]	O	输出 dout

2.15.3 功能定义

序号	功能名称	功能描述
1	传递数据	clk 为上升沿的时候，且 irwr 为 1 时，传递 DM 的输出

三．MIPS-Lite2 指令集+JALR 指令 机器指令描述

表 2.MIPS-LITE2 指令+JALR 指令

序号	助记符	opcode	funct	指令功能	
1	addu	000000	100001	两个来自寄存器的数无符号加法	GPR[rd] ← GPR[rs] + GPR[rt]
2	subu	000000	100011	两个来自寄存器的数无符号减法	GPR[rd] ← GPR[rs] - GPR[rt]
3	ori	001101	-	两个来自寄存器的数逻辑或运算	GPR[rt] ← GPR[rs] or immediate
4	lw	100011	-	读取 DM 中一个字到指定寄存器中	GPR[rt] ← memory[GPR[base] + offset]
5	sw	101011	-	指定寄存器中的字内容写到 DM 中	memory[GPR[base] + offset] ← GPR[rt]
6	beq	000100	-	两数相等则跳转	if GPR[rs] = GPR[rt] then branch
7	lui	001111	-	16 位立即数高位拓展	GPR[rt] ← immediate || 016
8	j	000010	-	无条件跳转	PC ← PCGPRLEN-1..28 || instr_index || 02

9	addi	001000	-	加立即数（符号拓展，有溢出检验）	temp ← (GPR[rs]31||GPR[rs]31..0) + sign_extend(immediate)
					if temp32 ≠ temp31 then
					SignalException(IntegerOverflow)
					else
					GPR[rt] ← temp
					endif

10	addiu	001001	-	加立即数（符号拓展，无溢出检验）	GPR[rt] ← GPR[rs] + immediate
11	slt	000000	101010	小于则置 1	GPR[rd] ← (GPR[rs] < GPR[rt])
12	jal	000011	-	跳转并链接（PC+4 存到$31）	GPR[31]← PC + 8 PC ← PCGPRLEN-1..28 || instr_index || 02
13	jr	000000	001000	跳转到寄存器存储的地址	PC ← GPR[rs]
14	lb	100000	-	读取 DM 中的字节	GPR[rt] ← memory[GPR[base] + offset]
15	sb	101000	-	向 DM 中存储字节	memory[GPR[base] + offset] ← GPR[rt]
16	jalr	000000	001001	跳转到 rs 中存储的地址，并将 PC+4 存储到 rd 中，如果没有指定 rd，则	GPR[rd] ← return_addr, PC ← GPR[rs]

				默认为 31 号寄存器	

四．测试程序

4.1 MIPS-LITE2 指令集测试程序

本测试程序和 MIPS-LITE1 的测试程序相比，仅仅在 19-25 行有改变（当然也不含 lh 指令），其余代码不变，因此下面只展示改变的代码。由于 lb 和 sb 和 lw，sw 相通，因此没有做注释

```
19  start2:sw $9, -4($8)
20  sw $1, 0($8)
21  lb $14, 3($8)
22  sb $12,7($8)
23  lw $15,4($8)
24  sb $4, -3($8)
25  lb $18, -1($8)
```

图 17.相对于 MIPS-LITE1 中测试程序改变的全部代码

4.2 新增 JALR 指令测试程序

```
ori $22, 0x30d0
jalr $22
```

```
ori $22, 0x30d0
jalr $23, $22
```

图 18-19.新增指令

本此测试同样分了两种情况，来测试 jalr 指令存储 pc+4 的默认情况和指定存储到哪个寄存器的情况，在后续的验证中也需要特别注意相应寄存器的存储

五．测试结果

5.1 MARS 中测试结果

Name	Number	Value
$zero	0	0x00000000
$at	1	0xababcdcd
$v0	2	0xababcdd3
$v1	3	0xababcdcd
$a0	4	0x2babcdd1
$a1	5	0x80000002
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x2babcdd1
$t1	9	0x2babcdd1
$t2	10	0x00690000
$t3	11	0x00000000
$t4	12	0x0000abab
$t5	13	0x000a0000
$t6	14	0x0000007f
$t7	15	0xab000000
$s0	16	0x00000003
$s1	17	0x00000001
$s2	18	0xffffff80
$s3	19	0xefef0000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0xababcdd2
$t8	24	0x7fffffff
$t9	25	0x00000001
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x00001800
$sp	29	0x00002ffc
$fp	30	0x00000000
$ra	31	0x000030b8
pc		0x000030e0
hi		0x00000000
lo		0x00000000

图 19.MARS 中 GPR 最终结果

Value (+0)	Value (+4)	Value (+8)
0x80000202	0x7fffffff	0xab000000

图 20.MARS 中 DM 最终结果

特殊说明：addi 指令运算结果若溢出，在 MARS 中会报错导致程序无法跑完，因此临时替换 addi 语句为“ori $0, $0, 0”可以在不影响所有 GPR 和 DM 数据的情况下跑通程序，图 6.的结果按 MIPS-LITE1 的要求，$30 最终存储的值应当为 0000_0001，这将在下文着重验证

$28 和$29 分别为 global pointer 和 stack pointer，在 MARS 运行时会有数值，而对于 MIPS-LITE1 简单指令集，不会涉及到这两个寄存器的写入访问，因此在 modelsim 波形仿真时，这两个寄存器的最终结果应当为 0

5.2 modelsim 验证结果

5.2.1 波形图（重点检验溢出）

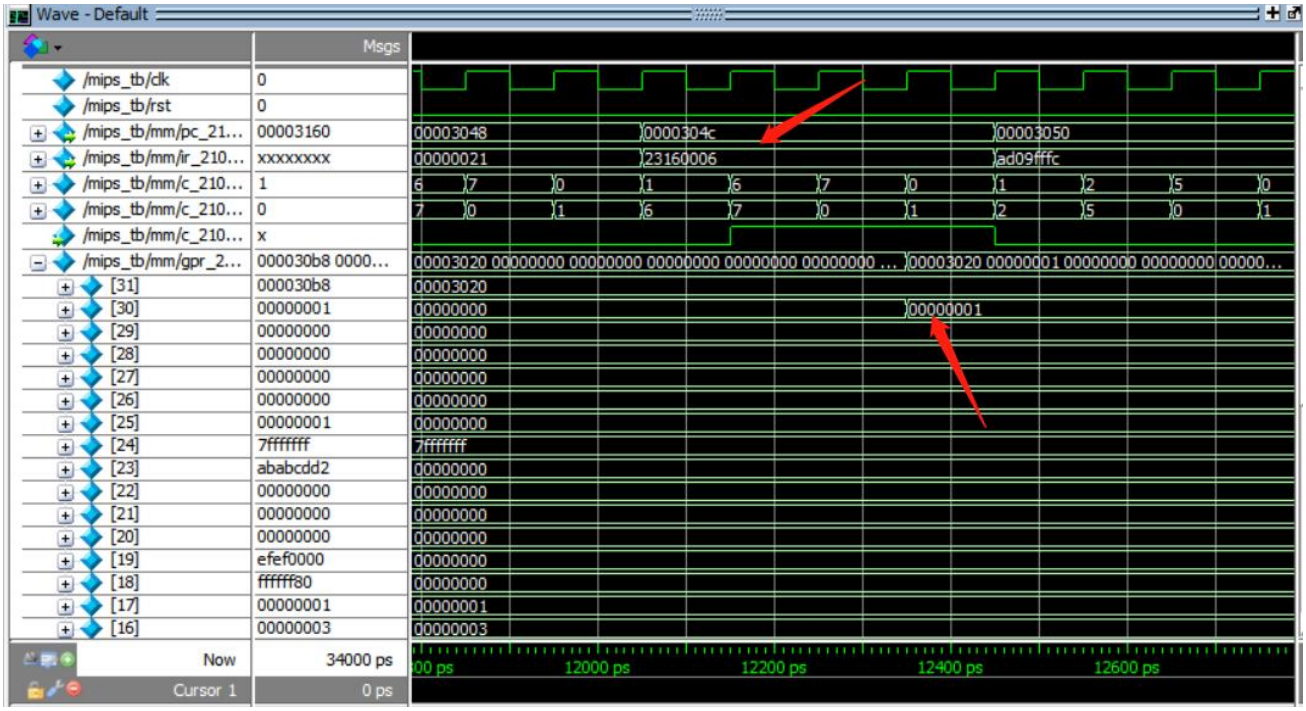

图 21.overflow 的判断和$30 存储 0000_0001h 数据

如图 21.所示，23160006 正是指令“addi $22,$24,6”，由图可知，执行本指令的时候 24 号寄存器存储的是 32 位二进制数正数最大值，+6 肯定会发生溢出，而后续的波形表示，不仅判断出了指令的溢出，而且还将 0000_0001h 存入了 30 号寄存器

5.2.2 DM 和 GPR 最终结果

```
0000001f 000030b8 00000001 00000000 00000000 00000000 00000000 00000001 7fffffff ababcdd2 00000000 00000000 00000000
00000013 efef0000 ffffff80 00000001 00000003 ab000000 0000007f 000a0000 0000abab 00000000 00690000 2babcdd1 2babcdd1
00000007 00000000 00000000 80000002 2babcdd1 ababcdcd ababcdd3 ababcdcd 00000000
```

图 22.modelsim 中 GPR 最终结果

```
00000057 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000033 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000f 00 00 00 00 ab 00 00 00 7f ff ff ff 80 00 02 02
```

图 23.modelsim 中 DM 最终结果

结果正确，说明 CPU 设计合理，已经能正确执行 MIPS-LITE1 指定的所有指令

5.3 新增指令测试

5.3.1 MARS 中测试结果

$s6	22	0x000030d0
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x00001800
$sp	29	0x00002ffc
$fp	30	0x00000000
$ra	31	0x00003008

$s6	22	0x000030d0
$s7	23	0x00003008
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x00001800
$sp	29	0x00002ffc
$fp	30	0x00000000
$ra	31	0x00000000

图 24-25.MARS 中 22 号寄存器存储结果，分别对应不指定 rd 和指定 rd

5.3.2 modelsim 中测试结果

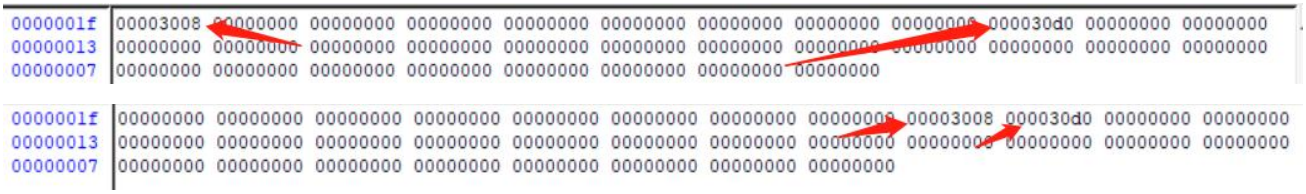

图 26-27.modelsim 中 22 号和 23 号寄存器的存储结果，其中 23 号寄存器为指定的寄存器结果正确，说明针对新指令的设计合理，能正确执行

六．Project2 完成后的心得体会

由于 project1 编写时，本着可以迭代到 project2 的理念，进行了模块化编程，所以在进行 project2 编写的时候费时极少，基本耗时都花在了状态机的理解和书写上。多周期对于单周期来说，重中之重就在于 controller 模块的修改，不仅要添加很多信号，在信号翻译时还需要注意信号是否是时序性，还要兼顾状态机的书写，考虑进各种转移情况。但是程序写好后，调试却用了很久很久。

由于对状态机的理解尚停留在书本的范畴，本次实操完毕在验证时，我对着延迟执行的指令疑惑了很久，再加上 jal 指令 PC+4 变为 PC+8 的问题，我一直以为是我代码设计和书写出现了问题，于是我开始画图辅助理解，尝试修改 testbench，都没有任何效果。本来萌生过修改 pc_4 信号从 pc+4 变为 pc，但是当时的我认为这是面向结果编程，并不会解决实质性问题，所以没有修改。直到第二天和同学交流才发现，原来我的想法是正确的，只不过是对状态机的理解还不够透彻，没有真正清除每一次 clk 推进，哪些模块都干了

些什么事情，经过快速修改后，果然解决了问题。这也同样说明了理论需要与实践结合，不能将自己的水平停留在书本上，一定要亲自动手，收获更宝贵的财富。

在检查时，我的任务是添加 jalr 指令。吸取了第一次检查的教训，我有条不紊地进行修改和调试，20 分钟不到就完成了任务，然而没有考虑到 jalr 还有不指定 rd 的默认情况，随后我又进行了修改，差点超时。这警示我不能浮躁，尽管准备充分，也应当处处小心谨慎。

Project3 VerilogHDL 完成 MIPS 微系统开发(支持设备与中断)

一. 总体数据通路结构设计图

project3 的总体数据通路结构设计如图 28.。本处理器参考了计算机组成原理 PPT5-5 所给的图，支持 MIPS-Lite3 指令集{MIPS-Lite2，ERET、MFC0、MTC0}。本处理器的模块继承了 project2 中的模块，因此只需要针对三条指令和 CP0 模块与 CPU 外设的模块编写，和原来模块的修改即可。

本 project 编写时参考了外部资料，半自主编写

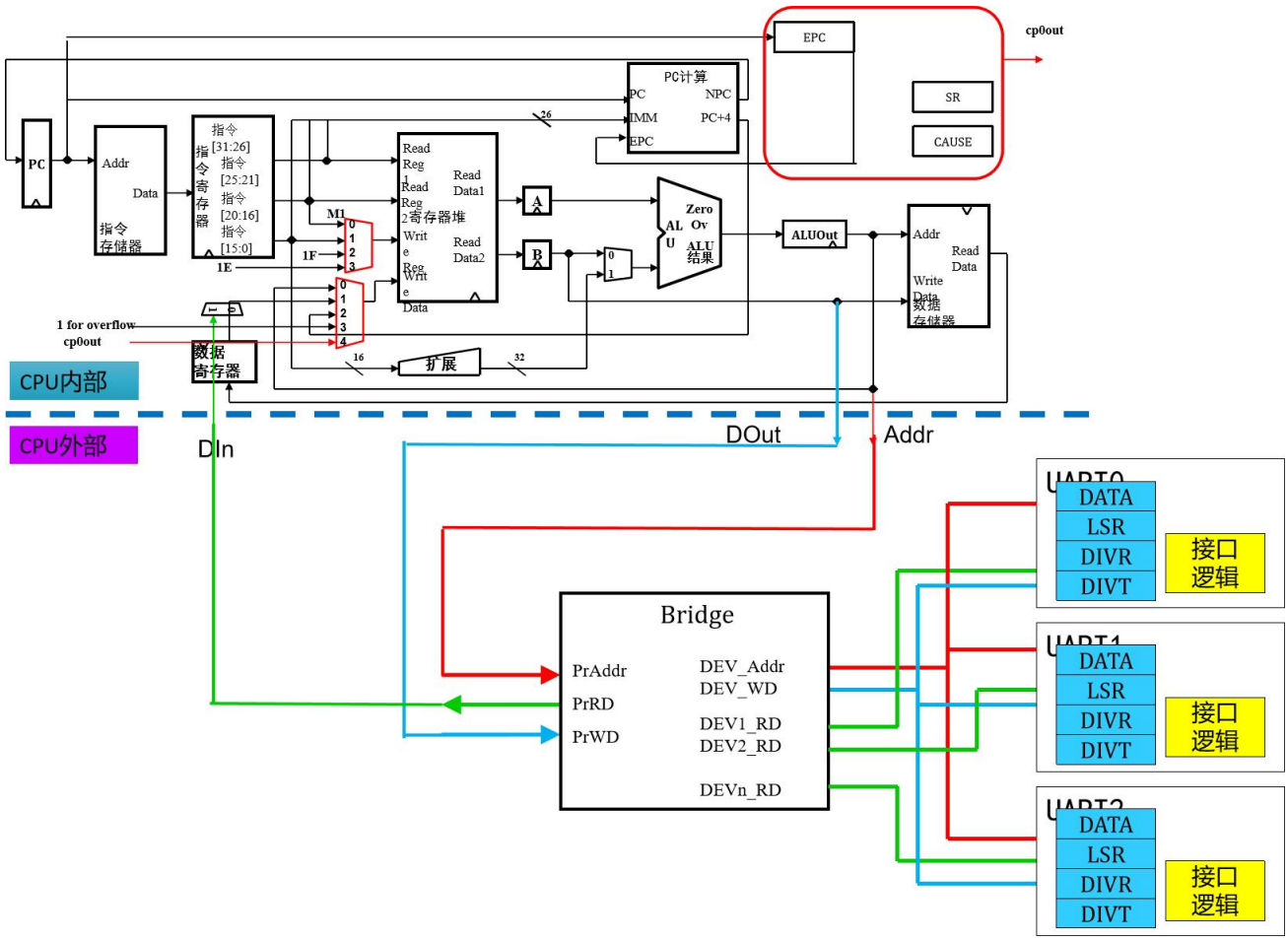

图 28.project3 总体数据通路结构设计图

二. 模块定义

2.1 PC 模块

2.1.1 基本描述

PC 模块是时序性模块，其功能为接收来自于 PC 计算模块输出的 NPC 信号，并在时钟信号 clk 上升沿时向外传输 NPC 输入的值

2.1.2 模块接口

信号名	方向	描述
clk	I	时钟信号
pcin[31:0]	I	来源于 PC 计算模块输入
pcwr	I	PC 模块写入权限，1 为可写
pcout[31:0]	O	本模块输出值，和 PC 计算模块输入值的数值相等

2.1.3 功能定义

序号	功能名称	功能描述
1	取地址	当 clk 信号上升沿时，如果 PC 可写，本模块将 NPC 模块计算好并输入进来的 PC，输出出去（输出到 IM 模块）

2.2 IM 模块

2.2.1 基本描述

IM 模块是非时序的。其功能为读取事先写好的十六进制代码程序，存入内部的 8kb 指令寄存器，依据输入的 PC 地址信号，进行取指。小端序存储指令。将中断处理运行的指令存储在 0000_4180h

2.2.2 模块接口

信号名	方向	描述
addr[12:0]	I	来源于 PC 模块的输出，也就是地址，由于 IM 模块存储

		指令的寄存器仅 8KB，因此只需要 PC 的低 13 位
dout[31:0]	O	本模块输出值，是地址所对应的 32 位指令

2.2.3 功能定义

序号	功能名称	功能描述
1	取指令	根据输入的地址，在本模块内部的指令寄存器里取得指令，并输出

2.3 NPC 模块

2.3.1 基本描述

NPC 模块是非时序性模块。本模块是推进 PC 地址更新的核心，它会根据当前指令的性质，判断并分析下一条指令的地址，传送给 PC 模块。对于 MIPS-Lite2 指令集，NPC 会根据 beq，j 和 jal 指令，jr 和 jalr 指令和此外的指令，进行四种地址分析策略，从而计算出下一条指令的地址。特别地，对于 jal 指令，NPC 模块还会输出 PC+4 以存入 31 号寄存器；对于 jr 和 jalr 指令，NPC 模块需要输入 GPR 输出的读取到 rs 寄存器的内容，便于跳转。

而对于 MIPS-LITE3 指令集，NPC 模块新加入了 EPC 模块输入的接口，并根据 IntReq 和 ERET 指令情况决定是否更新地址为 EPC 存储的内容。

复位时，本模块锁定 PC 地址为 0000_3000h

2.3.2 模块接口

信号名	方向	描述
imm[25:0]	I	由 IM 输出的 32 位指令分线出低 26 位而来，用于计算 j 和 jal 指令跳转的地址，或计算 beq 跳转的地址
pcin[31:0]	I	来源于 PC 模块的输出，也就是接收目前 PC 地址，便于计算下一条地址
rd1[31:0]	I	来源于 GPR 模块的输出，其内容是 rs 寄存器内部存储的 32 位数据
npcop[1:0]	I	来源于 Controller 的输出，代表着本模块进行哪一种地址计算逻辑： 00：传输 PC 01：beq 指令跳转 10：j/jal 指令跳转

		11：jr 和 jalr 指令跳转
zero	I	来源于 ALU 的输出，用于判断 beq 指令条件，决定是否跳转，如果为 1 才会进行 beq 指令跳转，否则正常 pc+4 推进
rst	I	异步复位信号，为 1 时会将 PC 地址重置为 0000_3000h，直至 rst 信号重新变为 0
epc[31:0]	I	来源于 CP0 中 EPC 寄存器的输出，存储中断地址
ERET	I	来源于 Controller 模块的输出，判断当前执行指令是否为 ERET
IntReq	I	来源于 Controller 模块的输出，代表着当前有中断请求且为 S10 状态时，PC 传入异常处理地址 0000_4180h
nextpc	O	计算好的 32 位地址，输出给 PC 模块
pc_4	O	输入的 pcin，在此基础上+4，专用于 jal 指令

2.3.3 功能定义

序号	功能名称	功能描述
1	异步复位	复位信号为 1 时，pcout 输出锁定为 0000_3000h
2	判断并计算	本模块会先计算出 j/jal 指令和 beq 指令跳转后的地址， beq 跳转：pc + 4 + imm<<2 j 跳转：{pc+4 的高四位, imm26, 00} 然后根据 npcop 进行如下操作： 00：输出 pc 01：如果 zero=1 输出 beq 跳转地址；否则 pc+4 10：j/jal 跳转地址 11：输出 rd1 信号内容
3	地址传递	计算好的新地址会传输给 PC 模块
4	支持中断和返回	通过 ERET 信号和 IntReq 信号，决定 PC 是否存入 EPC 存的值，或 PC 是否跳转到 0000_4180h

2.4 GPR 模块

2.4.1 基本描述

本模块是时序性模块。但比较特殊的是，其数据输出并不受到时间信号的影响，仅存储数

据时需要等待 clk 上升沿。本模块是 CPU 的寄存器堆，共有 32 个 32 位的寄存器，各司其职。其中 0 号寄存器恒为 0，无法被修改；31 号寄存器一般用来存储 jal 类指令所需要存储的地址，便于 jr 指令读取。

2.4.2 模块接口

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号
regwrite	I	来源于 Controller 模块，是寄存器的写入权限，1 代表可写
m1out[4:0]	I	来源于 M1 模块，是待写入数据的寄存器的编码
m2out[31:0]	I	来源于 M2 模块，是待写入 m1out 指定寄存器的数据
rs[4:0]	I	来源于 IR 模块的输出 irout[25:21]
rt[4:0]	I	来源于 IR 模块的输出 irout[20:16]
busa[31：0]	O	读出 rs 对应的寄存器的数值
busb[31:0]	O	读出 rt 对应的寄存器的数值

2.4.3 功能定义

序号	功能名称	功能描述
1	异步复位	复位信号为 1 时，32 个寄存器全置 0
2	写入数据	clk 为上升沿的时候，m2out 的内容存入 m1out 解码后对应的寄存器
3	读取数据	读取 IR 模块 irout 输出对应 irout[25:21]和[20:16]解码后对应的寄存器的数值并输出

2.5 EXT 模块

2.5.1 基本描述

本模块是非时序模块。本模块的功能是根据信号，对输入的 16 位数进行 32 位扩展并输出至目标模块 M3

2.5.2 模块接口

信号名	方向	描述
imm16[15:0]	I	来源于 IR 模块的 irout 的低 16 位，代表着 I 型指令对应的立即数
extop[1:0]	I	来源于 Controller 控制本模块扩展立即数的逻辑 00：无符号拓展 01：符号拓展 10：高位拓展（专用于 lui 指令）
out[31:0]	O	拓展完的数值进行输出

2.5.3 功能定义

序号	功能名称	功能描述
1	扩展并输出	对立即数进行相应的拓展并输出

2.6 DM 模块

2.6.1 基本描述

本模块是时序模块。本模块是专门存储数据的数据寄存器，容量为 12kb。当 clk 为上升沿的时候，允许写入数据。数据地址从 0000_0000h 开始。小端序存储数据

根据 islb 和 issb 信号会进行字节读写

2.6.2 模块接口

信号名	方向	描述
addr[13:0]	I	来源于 ALU 的输出，代表了本模块将要读写的地址
din[31:0]	I	来源于 GPR 的 busb 输出，代表了本模块将要写入的值

we	I	来源于 Controller 的输出，1 为 DM 可写
clk	I	时钟信号
islb	I	来源于 Controller 的输出，1 代表当前指令为 lb 指令
issb	I	来源于 Controller 的输出，1 代表当前指令为 sb 指令
dout[31:0]	O	输出数据

2.6.3 功能定义

序号	功能名称	功能描述
1	写入数据	写使能有效的时候，当 clk 为上升沿，根据 sb 指令判断，再根据地址进行数据写入
2	读取数据	根据地址和 lb 指令判断，进行数据读取并输出

2.7 ALU 模块

2.7.1 基本描述

本模块为非时序模块。本模块为运算模块，对输入的两个数，根据控制信号进行相应运算并进行输出。同时在特殊情况下会进行两数相等的判定，或两数相加溢出的判定，并输出对应的信号。

2.7.2 模块接口

信号名	方向	描述
A[31:0]	I	运算数 A
B[31:0]	I	来源于 M3 模块的输出，运算数 B
aluctr[2:0]	I	运算逻辑 000：加法 001：减法 010：按位或运算 011：(A 和 B 符号拓展) A < B ? 1 : 0，针对于 slt 指令 100：针对于 addi 指令的加法，会追加判断是否溢出
zero	O	针对于 beq 指令的信号，A=B 时输出 1，否则 0
overflow	O	针对于 addi 指令的信号，A+B 溢出时为 1，否则 0
out[31:0]	O	运算结果输出，为 A(运算逻辑)B

2.7.3 功能定义

序号	功能名称	功能描述
1	计算	根据 aluctr 进行相应逻辑的运算并输出结果
2	判断	针对于 beq、slt、addi 指令分别做出相应的判断，如上文所示

2.8 M1 模块

2.8.1 基本描述

本模块为非时序模块，用于 GPR 待写入寄存器编码的选择工作。

2.8.2 模块接口

信号名	方向	描述
gprsel[1:0]	I	来源于 Controller 模块的输出，用于选择 GPR 待写入寄存器的 5 位编码，控制着 m1out 信号的输出数值 00：rt 01：isjalr==1 ？ $31 ： rd 10：1F 11：1E
isjalr	I	来源于 Controller 的输出，1 代表执行的是本条指令
rt[4:0]	I	来源于 IM 模块的输出，为 dout[20:16]
rd[4:0]	I	来源于 IM 模块的输出，为 dout[15:10]
m1out[4:0]	O	输出数值被 gprsel 控制

2.8.3 功能定义

序号	功能名称	功能描述
1	多路选择	选择 GPR 待写入寄存器编码

2.9 M2 模块

2.9.1 基本描述

本模块为非时序模块，用于 GPR 待写入数据的选择工作。

2.9.2 模块接口

信号名	方向	描述
wdsel[2:0]	I	来源于 Controller 模块的输出，用于选择 GPR 待写入寄数据，控制着 m2out 输出信号的值 000：aluo 001：dmout 010：pc_4 011：0000_0001h（专用于 addi 指令 overflow 的情况） 100：cp0out
aluo[31:0]	I	来源于 IM 模块的输出，为 dout[20:16]
dmout[31:0]	I	来源于 IM 模块的输出，为 dout[15:10]
pc_4[31:0]	I	来源于 NPC 模块的输出，为 PC+4
cp0out[31:0]	I	来源于 CP0 模块的输出，即 DOut 信号
m2out[31:0]	O	输出数值被 wdsel 控制

2.9.3 功能定义

序号	功能名称	功能描述
1	多路选择	选择 GPR 待写入数据

2.10 M3 模块

2.10.1 基本描述

本模块为非时序模块，用于 ALU 运算数 B 的选择工作

2.10.2 模块接口

信号名	方向	描述
bsel	I	来源于 Controller 模块的输出，用于 ALU 运算数 B 的选择工作，控制着 m3out 输出信号的值 0：bo 1：imm32
bo[31:0]	I	来源于 GPR 模块的输出，为 busb
imm32[31:0]	I	来源于 EXT 模块的输出，为 EXT 模块的 out
m3out[31:0]	O	输出数值被 bsel 控制

2.10.3 功能定义

序号	功能名称	功能描述
1	多路选择	ALU 运算数 B 的选择

2.11 Controller 模块

2.11.1 基本描述

本模块为非时序模块。本模块是整个 CPU 中的重中之重，犹如人之大脑一般重要。其功能为输入 opcode 和 funct，从而对指令进行译码，输出控制信号，控制几乎 CPU 中每一个部件，没有了本模块，CPU 中的各个模块只能是一盘散沙，可见其重要性所在。在 project2 的基础上，针对 MIPS-LITE3 的要求，添加了中断状态 S10

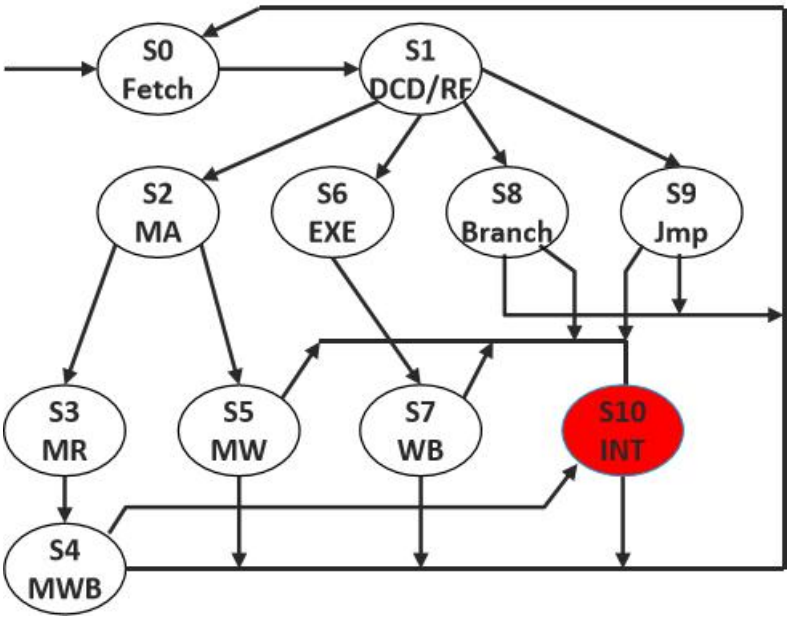

图 29.状态图

S0 为取指，S1 为译码，是 MIPS-LITE2 指令集中每一条指令必经之路。

S2 为针对于 DM 读写相关指令的入口，也就是 lw,lb,sw,sb；其中 lw 和 lb 会进入 S3 状态先对 DM 进行读取，之后进入 S4 状态进行 GPR 回写；sw 和 sb 会进入 S5 状态访存 DM

S6 为针对于 ALU 运算相关指令的入口，也就是 addu,subu,ori,addi,addiu,lui,slt，运算后会进入 S7 状态进行 GPR 回写

S8 为针对于分支相关指令的入口，也就是 beq 指令

S9 为针对于 j 型指令的入口，也就是 j,jal,jr,jalr 指令，本状态跳转和回写同时进行

由图可知，S4,S5,S7,S8,S9 执行完毕后，重回 S0 状态取指，完成一个周期

S4,S5,S7,S8,S9 若收到中断请求，均会转移到 S10 状态，进行中断处理

rst 复位会让状态机的状态锁定到 S0

2.11.2 模块接口

信号名	方向	描述
opcode[5:0]	I	来源于 IM 模块的输出，dout 的高六位
funct[5:0]	I	来源于 IM 模块的输出，dout 的低六位
overflow	I	来源于 ALU 模块的输出，是对 addi 指令运算结果是否溢出的判断信号，1 代表溢出
clk	I	时钟信号
rst	I	复位信号
zero	I	来源于 ALU 模块的输出，代表了两数是否相等
MF	I	来源于 IR 模块的输出，irout[25:21]
IntReq	I	来源于 CP0 模块的输出，1 代表中断请求

aluop[2:0]	O	输出到 ALU 模块，控制运算逻辑
gprsel[1:0]	O	输出到 M1 模块，选择待写入数据的寄存器的五位编码
gprwr	O	输出到 GPR 模块，控制 GPR 的写入权限
extop[1:0]	O	输出到 EXT 模块，控制其拓展逻辑
wdsel[2:0]	O	输出到 M2 模块，选择写入 GPR 目标寄存器的数值
npcop[1:0]	O	输出到 NPC 模块，选择 PC 地址运算逻辑
dmwr	O	输出到 DM 模块，控制 DM 的写入权限
bsel	O	输出到 M3 模块，选择输入 ALU 作为运算数 B 的数值
pcwr	O	输出到 PC 模块，决定 PC 是否写入来自 NPC 的输入
irwr	O	输出到 IR 模块，决定 IR 是否写入来自 IM 的输入
islb	O	输出到 DM 模块，决定 DM 的输出
issb	O	输出到 DM 模块，决定 DM 的写入
isjalr	O	输出到 M1 模块，参与 M1 的多路选择
cp0_wen	O	输出到 CP0 模块，决定了 CP0 内部寄存器的写入权限
bridge_wen	O	输出到 BRIDGE 模块，决定了 CPU 内外部访存的开关
EXLSet	O	输出到 CP0 模块，参与对 SR 寄存器置位
EXLClr	O	输出到 CP0 模块，参与对 SR 寄存器的重置
IntPc	O	输出到 NPC 模块，决定 NPC 是否读取 EPC

2.11.3 功能定义

序号	功能名称	功能描述
1	译码产生控制信号	根据 opcode 和 funct、zero 和 overflow 以及 IntReq 和 MF 进行译码，获得上文所示的各种重要控制信号
2	状态机推进	推进状态机，完成多周期
3	复位	重置状态机

2.12 ALUOUT 模块

2.12.1 基本描述

本模块为时序性模块。本模块接收 ALU 的输出结果，当时钟上升沿时，输出相应结果

2.12.2 模块接口

信号名	方向	描述
clk	I	时钟信号
aluoutin[31:0]	I	来源于 ALU 模块的输出，是两数相运算的结果
aluoutout[31:0]	O	输出 aluoutin

2.12.3 功能定义

序号	功能名称	功能描述
1	传递数据	clk 为上升沿的时候，传递 ALU 的输出

2.13 AR 模块 & BR 模块

2.13.1 基本描述

AR 和 BR 模块均为时序性模块，类似于 ALUOUT 模块，也是传递信息用的，传递的是 GPR 读取到的 busa 和 busb。由于二者性质完全相同，仅仅模块名和信号名为了区分才不同，因此下面只介绍 AR，BR 同理

2.13.2 模块接口

信号名	方向	描述
clk	I	时钟信号
arin[31:0]	I	来源于 GPR 模块的输出，也就是 busa
arout[31:0]	O	输出 arin

2.13.3 功能定义

序号	功能名称	功能描述
1	传递数据	clk 为上升沿的时候，传递 GPR 的输出

2.14 DR 模块

2.14.1 基本描述

DR 模块为时序性模块，用于传递 DM 的输出

2.14.2 模块接口

信号名	方向	描述
clk	I	时钟信号
drin[31:0]	I	来源于 DM 模块的输出，也就是 dmout
drout[31:0]	O	输出 dmout

2.14.3 功能定义

序号	功能名称	功能描述
1	传递数据	clk 为上升沿的时候，传递 DM 的输出

2.15 IR 模块

2.15.1 基本描述

IR 模块为时序性模块，用于传递 IM 的输出

2.15.2 模块接口

信号名	方向	描述
clk	I	时钟信号
irin[31:0]	I	来源于 IM 模块的输出，也就是 dout
irwr	I	写入权限
irout[31:0	O	输出 dout

]		

2.15.3 功能定义

序号	功能名称	功能描述
1	传递数据	clk 为上升沿的时候，且 irwr 为 1 时，传递 DM 的输出

2.16 M4 模块

2.16.1 基本描述

M4 模块为非时序模块，用于选择 M2 模块 001 接口的输入

2.16.2 模块接口

信号名	方向	描述
drout[31:0]	I	来源于 DR 模块的输出
prrd[31:0]	I	来源于 BRIDGE 模块的输出
addr[31:0]	I	来源于 ALUOUT 模块的输出，用于判断当前读取数据来源于 CPU 外部还是内部 本着节省引脚资源，只用判断 addr 的[15:8]==8'h7f 即可 0：drout 1：prrd
m4out[31:0]	O	模块输出

2.16.3 功能定义

序号	功能名称	功能描述
1	多路选择	选择 M2 模块 001 接口的输入

2.17 TIMER 模块

2.17.1 基本描述

本模块为时序性模块。本模块是外部设备，内部包含了三个 32 位小设备 CTRL、PRESET 和 COUNT，地址是 0000_7F00 ~ 0000_7F0B。具有计时功能，并且当程序进入死循环时可以发送中断请求，能进行计时设置和置位。

当计数器值为 0 时，根据计时器不同的模式执行不同的操作。当为模式 0 时，则一直保持 0 不变，直到 PRESET 再次被外部写入后，PRESET 值再次被加载至 COUNT，COUNT 重新启动倒计数，如果此时 CTRL[3]为 1，那么将会发送中算请求 IRQ；当为模式 1 时，则加载 PRESET 的值到 COUNT，开始新一轮的倒计时。

值得一提的是，TIMER 作为外部设备，其 clk 频率和 CPU 内部频率并不相同，因此 TIMER 和 CPU 交互的时候还需要设置分频器，但是我在阅读设计要求时没有注意到这个细节，在进行 TIMER 设计时虽然有过疑惑为什么 clk 信号要单独写成 CLK_I，但是没重视，没有进行分频器的编写。

2.17.2 模块接口

信号名	方向	描述
CLK_I	I	TIMER 设备自己的时钟信号，为 1 秒
RST_I[3:2]	I	TIMER 设备专属复位信号
WE_I	I	TIMER 设备的写入权限
ADD_I[3:2]	I	TIMER 内部三设备选址
DAT_I[31:0]	I	待写入 TIMER 的数据
DAT_O[31:0]	O	TIMER 读出的数据
IRQ	O	中断请求

2.17.3 功能定义

序号	功能名称	功能描述
1	复位	重置内部三个设备的值
2	计数	倒计时计数
3	产生中断	发出中断请求

2.18 OUTPUTDEV 模块

2.18.1 基本描述

本模块为时序模块。本模块是外部设备中的输出设备。包含了两个设备，也就是两个32 位的寄存器，规定其地址为 0000_7F10h——0000_7F18h。基于地址特点，我们仅需要地址位的[3:2]即可进行硬件选择

2.18.2 模块接口

信号名	方向	描述
clk	I	时钟信号
en	I	写使能
din[31;0]	I	待写入的数据
addr[3:2]	I	硬件地址
dout[31:0]	O	读取数据

2.18.3 功能定义

序号	功能名称	功能描述
1	存储数据	存储数据
2	读取数据	读取数据

2.19 BRIDGE 模块

2.19.1 基本描述

本模块为非时序模块。本模块是链接 CPU 内外的桥梁，正如其名。它具有一套地址，一套写数据，和 n 套读数据（n=设备个数）。有了本模块，CPU 内外部得以获得联系，是中断和异常的必须模块。

2.19.2 模块接口

信号名	方向	描述
praddr[31	I	来源于 ALUOUT 的输出，是外部设备读取地址

:0]		
prwd[31:0]	I	来源于 BR 的输出，是待写入外部设备的数据
dev0_rd[31:0]	I	来源于从外部设备 1 读取到的数据
dev1_rd[31:0]	I	来源于从外部设备 2 读取到的数据
dev2_rd[31:0]	I	来源于从外部设备 3 读取到的数据
wen	I	来源于 Controller 的输出，代表了 CPU 内外部链接是否开启，即外部设备写入使能
irq	I	来源于 TIMER 的输出，代表了中断请求
prrd[31:0]	O	读取到的硬件信息内容
dev_wd[31:0]	O	待写入数据输出
dev_addr[31:0]	O	写入地址输出
dev0_we	O	外部设备 1 的写使能
dev2_we	O	外部设备 3 的写使能
hwint[5:0]	O	外部中断信号

2.19.3 功能定义

序号	功能名称	功能描述
1	输出地址	直接输出 prraddr[X:2],X 由 N 个设备中地址空间需求最大者决定
2	地址匹配	设备地址译码
3	CPU 读数据	所有设备的数据输出汇聚至 CPU 的数据输入 MUX 的控制由 PrAddr 中某些位译码决定
4	CPU 写数据	CPU 写数据：连接至所有设备的输入 直通输出，不需要再转换

2.20 CP0 模块

2.20.1 基本描述

本模块为时序模块。本模块是 CPU 中的协处理器 0，内部含有 32 个 32 位的寄存器，本 project 只关注其中四个寄存器，它们的编号和名字依次是 12：SR　13：CAUSE　14：EPC 15：PRID。

SR 寄存器只读，允许某硬件的中断请求，控制着全局使能，并且可以对已经中断的设备进行标记防止再次进入中断；CAUSE 寄存器记录硬件中断是否有效；EPC 寄存器保存 PC，并进行指令读取；PRID 可以存入开发商编写的只读的、独特的、个性的 32 位标识码。

正如其名，CP0 协助着 CPU，主要掌管异常和中断操作

2.20.2 模块接口

信号名	方向	描述
PC[31:0]	I	来源于 PC 模块的输出，保存中断 PC 至 EPC
DIn[31:0]	I	来源于 BR 寄存器，为 CP0 待写入数据
HWInt[5:0]	I	来源于 BRIDGE 模块输出，是外部中断信号
Sel[4:0]	I	来源于 IR 的输出，是 irout[15:11]也就是 rd
Wen	I	来源于 Controller 的输出，是 CP0 的写使能
EXLSet	I	来源于 Controller 的输出，参与 SR 寄存器置位
EXLClr	I	来源于 Controller 的输出，参与 SR 寄存器复位
clk	I	时钟信号
rst	I	复位信号
IntReq	O	中断请求信号
epc[31:0]	O	epc 内容输出
DOut[31:0]	O	CP0 的寄存器[Sel]输出

2.20.3 功能定义

序号	功能名称	功能描述
1	复位	四个寄存器复位
2	记录 PC	记录中断产生的地址
3	写数据	向 CP0 的寄存器组写入数据
4	产生中断	根据中断请求和中断有效，产生中断

5	记录异常	HWInt 不断被 CAUSE 寄存器锁存

三．MIPS-Lite3 指令集 机器指令描述

表 3.MIPS-LITE3 指令

序号	助记符	opcode	funct	指令功能	
1	addu	000000	100001	两个来自寄存器的数无符号加法	GPR[rd] ← GPR[rs] + GPR[rt]
2	subu	000000	100011	两个来自寄存器的数无符号减法	GPR[rd] ← GPR[rs] - GPR[rt]
3	ori	001101	-	两个来自寄存器的数逻辑或运算	GPR[rt] ← GPR[rs] or immediate
4	lw	100011	-	读取 DM 中一个字到指定寄存器中	GPR[rt] ← memory[GPR[base] + offset]
5	sw	101011	-	指定寄存器中的字内容写到 DM 中	memory[GPR[base] + offset] ← GPR[rt]
6	beq	000100	-	两数相等则跳转	if GPR[rs] = GPR[rt] then branch
7	lui	001111	-	16 位立即数高位拓展	GPR[rt] ← immediate || 016
8	j	000010	-	无条件跳转	PC ← PCGPRLEN-1..28 || instr_index || 02

9	addi	001000	-	加立即数（符号拓展，有溢出检验）	temp ← (GPR[rs]31||GPR[rs]31..0) + sign_extend(immediate)
					if temp32 ≠ temp31 then
					SignalException(IntegerOverflow)
					else
					GPR[rt] ← temp
					endif

10	addiu	001001	-	加立即数（符号拓展，无溢出检验）	GPR[rt] ← GPR[rs] + immediate
11	slt	000000	101010	小于则置 1	GPR[rd] ← (GPR[rs] < GPR[rt])
12	jal	000011	-	跳转并链接（PC+4 存到$31）	GPR[31]← PC + 8 PC ← PCGPRLEN-1..28 || instr_index || 02
13	jr	000000	001000	跳转到寄存器存储的地址	PC ← GPR[rs]

14	lb	100000	-	读取 DM 中的字节	GPR[rt] ← memory[GPR[base] + offset]
15	sb	101000	-	向 DM 中存储字节	memory[GPR[base] + offset] ← GPR[rt]
16	jalr	000000	001001	跳转到 rs 中存储的地址，并将 PC+4 存储到 rd 中，如果没有指定 rd，则默认为 31 号寄存器	GPR[rd] ← return_addr, PC ← GPR[rs]
17	eret	010000	011000	中断返回	PC ← EPC
18	mtc0	010000	irout[25:21]=00100	CP0 寄存器写入	CPR[0, rd] ← GPR[rt]
19	mfc0	010000	irout[25:21]=00000	读取 CP0 寄存器	GPR[rt] ← CPR[0,rd]

四．测试程序

4.1 MIPS-LITE3 指令集测试程序

```
1  ori $1,$0,0x7f00        # ctrl寄存器地址
2  ori $2,$0,0x7f04        # preset寄存器地址
3  ori $3,$0,0x7F08        # count寄存器地址
4  ori $4,$0,0x7F0C        # input设备地址
5  ori $5,$0,0x7F10        # OPD preData输出设备初值寄存器地址
6  ori $6,$0,0x7F14        # OPD curData输出设备当前值寄存器地址
7  lw $7,0($4)             # 获取输入设备当前的输入值
8  sw $7,0($5)             # preData <- input
9  sw $7,0($6)             # curData <- input
10 ori $12,$0,0x0401       # sr <- 100_0000_0001
11 mtc0 $12,$12            # cp0[SR] <- $12
12 mfc0 $21,$15            # $21 <- cp0[PRID]
13 ori $13,$0,0x000a       # 10 倒计时
14 ori $9, $0, 0x0009      # ctrl控制位 1001
15 sw $13,0($2)            # preset <- 10
16 sw $13,0($3)            # count <- 10
17 sw $9,0($1)             # ctrl <- 1001
18 loop:j loop             # 死循环
```

图 30.主程序

```
1  lw $15,0($4)     # $15 <- input
2  lw $16,0($5)     # $16 <- preData
3  beq $16,$15,eq   # 比较二者的值，相等跳eq处
4  sw $15,0($5)     # 如果不相等，preData <- $5
5  sw $15,0($6)     # 如果不相等，curData <- $6
6  j jmp
7  eq:lw $16,0($6)  # $16 <- curData
8  addiu $16,$16,1  # $16 += 1
9  sw $16,0($6)     # curData <- $16
10 jmp:ori $13,$0,0x000a   # 倒计时重置
11 sw $13,0($2)     # 完成设置
12 eret             #中断返回
```

图 31.中断子程序

①主程序通过读取 32 位输入设备内容并显示在 32 位输出设备上。

②主程序将定时器初始化为模式 0，并加载正确的计数初值至预置计数初值寄存器以产生 1s 的计数周期。

③主程序启动定时器计数后进入死循环。

④中断子程序不断读取新的输入设备内容，一旦发现与之前的 32 位输入值不同，则更新 32 位输出设备显示为当前新值；否则将输出设备显示内容加 1。然后重置初值寄存器从而再次启动定时器计数，实现
新一轮秒计数。

由于 MARS 在对外设和中断程序执行调试很繁琐，因此下面只展示 modelsim 验证波形

五．测试结果

5.1 波形图（重点观察倒计时和中断返回）

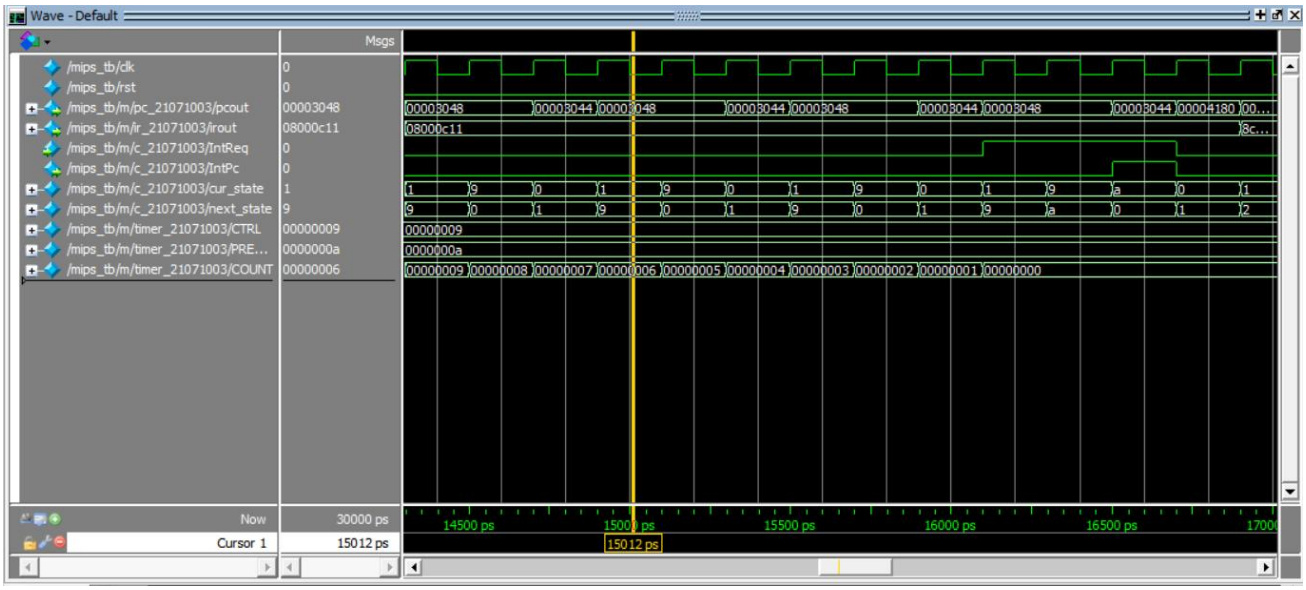

图 32.倒计时

由图 32.可见，在死循环反复跳转时，倒计时模块进行倒计时功能，当倒计时到 0 时，中断请求发起，进入预先在 0000_4180h 放入的中断指令，也就是图 31.的指令开始执行。

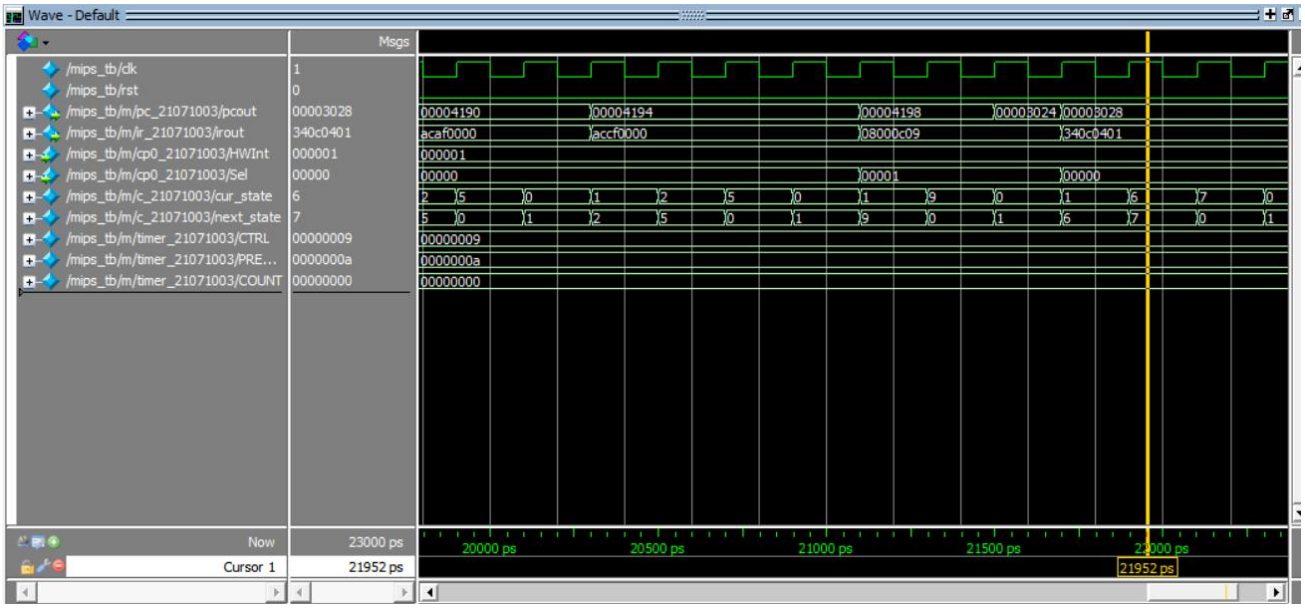

图 33.中断返回

由图 33.可见，中断返回，回到了中断产生的地址，由于中断产生的地址对应指令为死循环，因此后续的波形将循环进行中断和中断后指令处理，从而对输出设备的数值进行循环更新。

5.2 寄存器结果

5.2.1 GPR

```
0000001f 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 21071003 00000000 00000000 00000000 00000000
00000010 00000042 00000032 00000000 0000000a 00000401 00000000 00000000 00000009 00000000 00000042 00007f14 00007f10 00007f0c 00007f08 00007f04
00000001 00007f00 00000000
```

图 34.GPR 结果

由图 34.可见，mfc0 指令正确执行，读取到了 PRID

5.2.2 CP0 寄存器

```
0000001f 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000010 00000000 21071003 00000000 00000400 00000401 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000001 00000000 00000000
```

图 35.CP0 寄存器结果 1

由图 35.，图为中断跳转前的寄存器，可以看出 SR 寄存器此时已经放置好初值，CAUSE 寄存器已经锁存好 HWInt，而 EPC 因为未跳转，还未存入地址，PRID 固定为设置好的 21071003

```
0000001f 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000010 00000000 21071003 00003044 00000400 00000403 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000001 00000000 00000000
```

图 36.CP0 寄存器结果 2

由图 36.，图为中断跳转后的寄存器，可以看出 SR 寄存器中 exl 已经置 1，CAUSE 寄存器仍在锁存，而 EPC 已经存入了中断产生的地址，便于后续的返回操作

经过如上验证，证明 CPU 设计合理，已经基本完成了规定的任务

六．Project3 完成后的心得体会

project3 给我的感觉就是非常难，在周二调试完 p2 后，我马不停蹄赶到了图书馆去重新学习中断相关知识，然而废了很久时间，仅仅是能自主画出数据通路图，大致理解了 CPU 内外部沟通的逻辑，但是当上手写代码的时候，发现各个控制信号什么时候该是什么还是不理解。

由于检查时间紧张，我选择了半自主编写代码，和同学进行了探讨和交流，互相借鉴思路，并且借助了一些外部资料，来辅助自己理解各信号间的关系，最后做了很多代码注释，理解了 p3 的全部知识内容。在检查时我也坦白了 project3 并非全自主完成，对此拿不到 A 档满分我心服口服，更何况我没有注意到 COUNTER 周期要求设置为 1s 的细节，没有设计对应的分频器。

我认为本 project 非自主完整，大部分原因是我课堂上没有认真学习中断知识，认为考试不考就不需要花费时间去复习巩固，然而等临近检查才开始学习为时已晚，本次扣分已经给了我经验教训。同时我和同学认为 p3 的任务指导书有些不具体，不足够我们理解 project3 的全部内容，已经在检查当日汇报。我们还发现任务书中提到的“《异常中断及协处理器》”资料找不到。

从入学开始，我就深知计算机组成原理对于计科专业的重要性，它是考研必考且占比很大的科目，它是计科人必备的知识，它是理解硬件最底层最基础的原理，通过一个学期的学习，我见识到了它的难度和强度，也通过课设亲自体会感受到了 CPU 设计的神奇之处。本学期，我萌生了以后向硬件、芯片研究的想法，在结束了计组和课设的学习后，我将会承载着实践中获得的知识和经验，在暑假进一步学习中科院“一生一芯”计划，不断努力！

最后，请允许我在课设报告的结尾，感谢老师一学期以来的教学，感谢学习过程中老师、同学们给予的帮助和鼓励！为我的计组和课设画上了完美的句号！