计算机组成原理 P3 实验报告

一、CPU 设计方案综述

（一）总体设计概述

本 CPU 为 Logisim 实现的单周期 MIPS - CPU，支持的指令集包含{addu、subu、ori、lw、sw、beq、lui、nop、jal、jr}。为了实现这些功能，CPU 主要包含了 PC、GRF、ALU、DM、Controller、NPC、IM、EXT、split 等模块。

（二）关键模块定义

1. PC

表 1 PC 模块定义

功能描述	主要部分为 32 位可复用寄存器，reset 有效时寄存器初始值为 0x00000000	
信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号
DI[31:0]	I	32 位输入下一条指令地址
DO[31:0]	O	32 位输出当前指令地址

2. NPC

表 2 NPC 模块定义

功能描述	次地址计算（包括跳转）	
信号名	方向	描述
PC[31:0]	I	32 位输入当前指令地址
Imm[31:0]	I	32 位输入用于跳转的立即数
NPCOp[1:0]	I	2 位输入用于选择 NPC
Zero	I	1 位输入辅助选择 NPC
RA	I	32 位输入来自 31 号寄存器的指令地址

NPC[31:0]	O	32 位输出次指令地址
PC4	O	32 位输出返回值

3. IM

表 3 IM 模块定义

功能描述	指令存储器，根据输入的地址输出指令	
信号名	方向	描述
ImAddr[31:0]	I	32 位输入 PC
ImData[31:0]	O	32 位输出指令信号

4. Split

表 4 Split 模块定义

功能描述	用于截取指令信号不同部分	
信号名	方向	描述
instr[31:0]	I	32 位输入指令信号
Opcode[5:0]	O	输出指令信号的 31-26 位
Func[5:0]	O	输出指令信号的 5-0 位
25-21[4:0]	O	输出指令信号的 25-21 位
20-16[4:0]	O	输出指令信号的 20-16 位
15-11[4:0]	O	输出指令信号的 15-11 位
Imm[25:0]	O	输出指令信号的 25-0 位

5. Controller

表 5 Controller 模块定义

功能描述	用于截取指令信号不同部分	
信号名	方向	描述
Func[5:0]	I	6 位输入
OpCode[5:0]	I	6 位输入
NPCOp[1:0]	O	2 位输出控制 NPC 选择

RegWrite	O	1 位输出控制 GRF 写使能
EXTOp[2:0]	O	3 位输出控制 EXT 功能选择
ALUOp[3:0]	O	4 位输出控制 ALU 功能选择
MemWrite	O	1 位输出控制 DM 写使能
RegA3Sel[1:0]	O	2 位输出用于 GRF 写入寄存器的选择
RegDataSel[1:0]	O	2 为输出用于 GRF 写入数据的选择
AluBSel	O	1 位输出控制 ALU 第二个数据的选择

6. EXT

表 6 EXT 模块定义

信号名	方向	功能
Imm[15:0]	I	16 位输入立即数
EXTOp[2:0]	I	3 位输入控制 EXT 功能选择
PC[31:28]	I	4 位输入用于计算 jal 跳转地址
Ext[31:0]	O	32 位输出扩展结果

7. ALU

表 7 ALU 模块定义

信号名	方向	功能
A[32:0]	I	第一个 32 位操作数
B[32:0]	I	第二个 32 位操作数
ALUOp[3:0]	I	4 位输入 ALU 功能选择
C[31:0]	O	32 位输出计算结果
Zero	O	1 位输出判断结果

8. GRF

表 8 GRF 模块定义

信号名	方向	功能
RegA1[4:0]	I	第一个读出寄存器编号
RegA2[4:0]	I	第二个读出寄存器编号

RegA3[4:0]	I	回写寄存器的编号
RegData[31:0]	I	回写寄存器的值
WE	I	写入使能
clk	I	时钟信号
reset	I	复位信号
RD1[31:0]	O	第一个寄存器编号读出的寄存器值
RD2[31:0]	O	第二个寄存器编号读出的值

9. DM

表 9 DM 模块定义

信号名	方向	功能
MemAddr[4:0]	I	写入的地址
MemData[31:0]	I	写入的数据
WE	I	写入使能
clk	I	时钟信号
reset	I	复位信号
RD[31:0]	O	读出数据

（三）数据通路的综合

1. 所有指令的指令级别数据通路

表 10 所有指令的指令级别数据通路

部件	PC	NPC			IM	RF				EXT	ALU		DM	
输入信号	DI	PC	Imm	RS	PC	RegA1	RegA2	RegA3	RegData	Imm	A	B	MemData	MemAddr
addu	NPC.Npc	PC.DO			PC.DO	IM.D[25:21]	IM.D[20:16]	IM.D[15:11]	ALU.C		RF.RD1	RF.RD2		
subu	NPC.Npc	PC.DO			PC.DO	IM.D[25:21]	IM.D[20:16]	IM.D[15:11]	ALU.C		RF.RD1	RF.RD2		
ori	NPC.Npc	PC.DO			PC.DO	IM.D[25:21]		IM.D[20:16]	ALU.C	IM.D[25:0]	RF.RD1	EXT.Ext		
lw	NPC.Npc	PC.DO			PC.DO	IM.D[25:21]		IM.D[20:16]	DM.RD	IM.D[25:0]	RF.RD1	EXT.Ext		ALU.C
sw	NPC.Npc	PC.DO			PC.DO	IM.D[25:21]	IM.D[20:16]			IM.D[25:0]	RF.RD1	EXT.Ext	RF.RD2	ALU.C
beq	NPC.Npc	PC.DO	EXT.Ext		PC.DO	IM.D[25:21]	IM.D[20:16]				RF.RD1	RF.RD2		
lui	NPC.Npc	PC.DO			PC.DO			IM.D[20:16]	EXT.Ext	IM.D[25:0]				
jal	NPC.Npc	PC.DO	EXT.Ext		PC.DO			0x1f	NPC.PC4	IM.D[25:0]				
jr	NPC.Npc	PC.DO		RF.RD1	PC.DO	IM.D[25:21]								

2. 控制信号真值表

表 11 控制信号真值表

指令	NPCOp	RegWrite	EXTOp	ALUOp	MemWrite	RegA3Sel	RegDataSel	AluBSel
addu	00	1		000		00	00	0
subu	00	1		001		00	00	0
ori	00	1	000	010		01	00	1
lw	00	1	001	000		01	01	1
sw	00		001	000	1			1
beq	01		010	011				0
lui	00	1	011			01	10	
jal	10	1	100			10	11	
jr	11							

3. 控制信号状态描述

表 12 控制信号状态描述

NPCOp		EXTOp		ALUOp		RegA3Sel		RegDataSel		AluBSel	
控制信号	状态描述	控制信号	状态描述	控制信号	状态描述	控制信号	状态描述	控制信号	状态描述	控制信号	状态描述
00	PC+4	000	0 扩展	000	加法运算	00	存入寄存器为 IM.D[15:11]	00	存入寄存器数据为 ALU.C	0	ALU 的第二个操作数为 RF.RD2
01	PC+4+Imm 用于 beq 跳转	001	符号扩展	001	减法运算	01	存入寄存器为 IM.D[20:16]	01	存入寄存器数据为 DM.RD	1	ALU 的第二个操作数为 EXT.Ext
10	jal 次地址	010	左移 2 位符号扩展	010	按位或	10	存入 31 号寄存器	10	存入寄存器数据为 EXT.Ext		
11	来自 ra 的次地址	011	加载至高 16 位	011	判断是否相等			11	存入寄存器数据位 PC4		
		100	jal 的次地址计算								

（四）重要机制实现方法

1. 跳转

在 EXT 模块里附加了用于跳转的地址计算（本应将此功能放入 NPC 模块中，无奈扩展指令时没有考虑到此问题）。

二、测试方案

（一）典型测试样例

1. addu、subu 指令测试

MIPS 汇编指令	机器码
ori $1, $0, 1	34010001
ori $2, $0, 0xaffc	3402affc
ori $3, $0, 3	34030003
tab1:	34041001
ori $4, $0, 0x1001	0020f021
addu $30, $1, $0	0002e821
tab2:	0062e021
addu $29, $0, $2	0081d821
addu $28, $3, $2	0361d021
addu $27, $4, $1	037bc821
addu $26, $27, $1	0083c023
addu $25, $27, $27	0324b823
subu $24, $4, $3	0020b023
subu $23, $25, $4	0001a823
subu $22, $1, $0	0064a023
subu $21, $0, $1	13bbfff5
subu $20, $3, $4	1357fff2
beq $29, $27, tab2	
beq $26, $23, tab1	

MIPS 运行结果：

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000001
$v0	2	0x0000affc
$v1	3	0x00000003
$a0	4	0x00001001
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0xfffff002
$s5	21	0xffffffff
$s6	22	0x00000001
$s7	23	0x00001003
$t8	24	0x00000ffe
$t9	25	0x00002004
$k0	26	0x00001003
$k1	27	0x00001002
$gp	28	0x0000afff
$sp	29	0x0000affc
$fp	30	0x00000001
$ra	31	0x00000000
pc		0x0000300c
hi		0x00000000
lo		0x00000000

图 1 测试样例 1 的 MIPS 运行结果

2. ori、lui 指令测试

MIPS 汇编指令	机器码
ori $17, $0, 0	34110000
ori $18, $0, 256	34120100
ori $19, $18, 768	36530300
ori $17, $19, 0xffff	3671ffff
ori $28, $0, 1	341c0001
ori $20, $28, 5	37940005
ori $31, $28, 6	379f0006
ori $14, $20, 512	368e0200
ori $28, $14, 0xaffc	35dcaffc
ori $15, $30, 0xcbac	37cfcbac
lui $6, 1	3c060001
lui $7, 0xffff	3c07ffff
lui $8, 0xaffc	3c08affc
lui $9, 256	3c090100

lui $10, 0x1abc	3c0a1abc
lui $11, 0xbbca	3c0bbbca

MIPS 运行结果：

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00010000
$a3	7	0xffff0000
$t0	8	0xaffc0000
$t1	9	0x01000000
$t2	10	0x1abc0000
$t3	11	0xbbca0000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000205
$t7	15	0x0000cbac
$s0	16	0x00000000
$s1	17	0x0000ffff
$s2	18	0x00000100
$s3	19	0x00000300
$s4	20	0x00000005
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x0000affd
$sp	29	0x00002ffc
$fp	30	0x00000000
$ra	31	0x00000007
pc		0x00003040
hi		0x00000000
lo		0x00000000

图 2 测试样例 2 的 MIPS 运行结果

3. lw、sw 指令测试

MIPS 汇编指令	机器码
ori $14, $0, 124	340e007c
ori $15, $0, 4	340f0004
ori $16, $0, 32	34100020
ori $1, $0, 0x1ffc	34011ffc
ori $2, $0, 256	34020100
ori $3, $0, 0xaffc	3403affc
sw $1, 0($14)	adc10000
sw $1, 4($16)	ae010004
sw $2, -4($16)	ae02fffc
sw $1, 64($0)	ac010040
sw $3, 0($14)	adc30000

sw $3, 4($16)	ae030004
sw $3, -4($16)	ae03fffc
lw $15, 4($16)	8e0f0004
lw $16, -4($16)	8e10fffc
lw $17, 0($14)	8dd10000
lw $18, 64($0)	8c120040
lw $19, 0($0)	8c130000

MIPS 运行结果：

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00001ffc
$v0	2	0x00000100
$v1	3	0x0000affc
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x0000007c
$t7	15	0x0000affc
$s0	16	0x0000affc
$s1	17	0x0000affc
$s2	18	0x00001ffc
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$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	0x00000000
pc		0x00003048
hi		0x00000000
lo		0x00000000

图 3 测试样例 3 的 MIPS 运行结果

4. beq、nop 指令测试

MIPS 汇编指令	机器码
ori $25, $0, 256	34190100
ori $27, $0, 14	341b000e
addu $28, $25, $27	033be021
addu $29, $28, $28	039ce821
label1:	0399f021
addu $30, $28, $25	03bbf823

subu $31, $29, $27	37390001
ori $25, $25, 1	377b0002
ori $27, $27, 2	00000000
nop	133bfffa
beq $25, $27, label1	13df0001
beq $30, $31, label2	037b7021
addu $14, $27, $27	34040005
label2:	00000000
ori $4, $0, 5	00842821
nop	
addu $5, $4, $4	

MIPS 运行结果：

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000005
$a1	5	0x0000000a
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000101
$k0	26	0x00000000
$k1	27	0x0000000e
$gp	28	0x0000010e
$sp	29	0x0000021c
$fp	30	0x0000020e
$ra	31	0x0000020e
pc		0x0000303c
hi		0x00000000
lo		0x00000000

图 4 测试样例 4 的 MIPS 运行结果

5. jal、jr 指令测试

本测试样例需设置 text 起始地址为 0。

MIPS 汇编指令	机器码
ori $1, $0, 256	34010100

ori $2, $0, 512	34020200
addu $3, $1, $2	00221821
jal table1	0c000008
lui $5, 0xaffc	3c05affc
jal table2	0c00000b
addu $6, $1, $0	00203021
nop	00000000
	00632821
table1:	0065d823
addu $5, $3, $3	03e00008
subu $27, $3, $5	34a70065
jr $ra	00e74021
table2:	03e00008
ori $7, $5, 101	
addu $8, $7, $7	
jr $ra	

MIPS 运行结果：

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000100
$v0	2	0x00000200
$v1	3	0x00000300
$a0	4	0x00000000
$a1	5	0xaffc0000
$a2	6	0x00000100
$a3	7	0xaffc0065
$t0	8	0x5ff800ca
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0xfffffd00
$gp	28	0x00001800
$sp	29	0x00003ffc
$fp	30	0x00000000
$ra	31	0x00000018
pc		0x00000020
hi		0x00000000
lo		0x00000000

图 5 测试样例 5 的 MIPS 运行结果

三、思考题

（一）题目：现在我们的模块中 IM 使用 ROM， DM 使用 RAM， GRF 使用 Register，这种做法合理吗？ 请给出分析，若有改进意见也请一并给出。

答：我认为这样做是合理的。在本单周期 CPU 中，所有指令是提前存储的且之后不再需要修改，因此单端口的只读存储器 ROM 即可满足这一需求；对于数据存储器，一方面需要写入数据（如 sw），另一方面也需要将其中存储的数据读出（如 lw），这便需要一个支持读取和写入的存储器，而同时数据存储器的读取的写入并不会有同时进行的情况，因此可以使用读取和写入分离的 RAM 存储器，并不需要使用 Register；对于 GRF，其需要同时完成内容的读取的下一内容的存入，因此只能使用可同时读取存入的 Register。

（二）题目：事实上，实现 nop 空指令，我们并不需要将它加入控制信号真值表，为什么？请给出你的理由。

答：我认为在真值表中为 1 的位才会与相应控制信号相连并实现控制作用，而 nop 空指令各位均为 0，因此不论是否加入真值表，都不会对控制信号的设置产生影响，故并不需要将其加入。

（三）题目：上文提到，MARS 不能导出 PC 与 DM 起始地址均为 0 的机器码。实际上，可以通过为 DM 增添片选信号，来避免手工修改的麻烦，请查阅相关资料进行了解，并阐释为了解决这个问题，你最终采用的方法。

答：在我的电路中，有一子电路“32to5”用于将 32 位信号的 2-6 位截取为 5 位地址信号，为解决题目所述问题，我将该子电路替换为了一个新的子电路“NewDMAddr”，在 Mars 中设置 Text at Address 0，此时有.data base address 为 0x00002000，在子电路中首先将 32 位信号使用减法器减,0x00002000，得到的新地址即为起始地址为 0，由此便可解决本问题。（PC 是不是涉及到 jal 指令的时候才会用到呀，现阶段是不是不需要考虑这个问题呢？）（所以这和片选信号有什么关系不太懂...）

（四）题目：除了编写程序进行测试外，还有一种验证 CPU 设计正确性的办法——形式验证。 形式验证的含义是根据某个或某些形式规范或属性，使用数学的方法证明其

正确性或非正确性。请搜索“形式验证（Formal Verification)”了解相关内容后，简要阐述相比于测试，形式验证的优劣之处。

答：实现测试需要完成创建模型、开发测试平台、创建激励、执行测试等众多环节，消耗大量时间精力且很容易出现边缘情形无法覆盖的情况，而相比之下形式验证直接从数学逻辑层面面向高层需求，其验证是完备的，可以对所有可能情况进行验证而非仅仅对某一个设定的子集进行检验，提高了效率也增加了准确性。但另一方面，形式验证与测试相比是在一个更抽象的层次上完成的，因此无法像实际测试一样反映模块的实际运行状态，对于功耗、延迟等非理想问题很难发现与解决，这也是形式验证的一个不足之处。

四、版本迭代记录

（一）第一版

顶层模块几乎全部使用 tunnle，外观较为整洁，但 debug 过程困难较大，出现了 tunnle 找不全的状况。甚至第一版 DM 的 reset 没有连接至子电路端口，系统弱测竟然通过了。

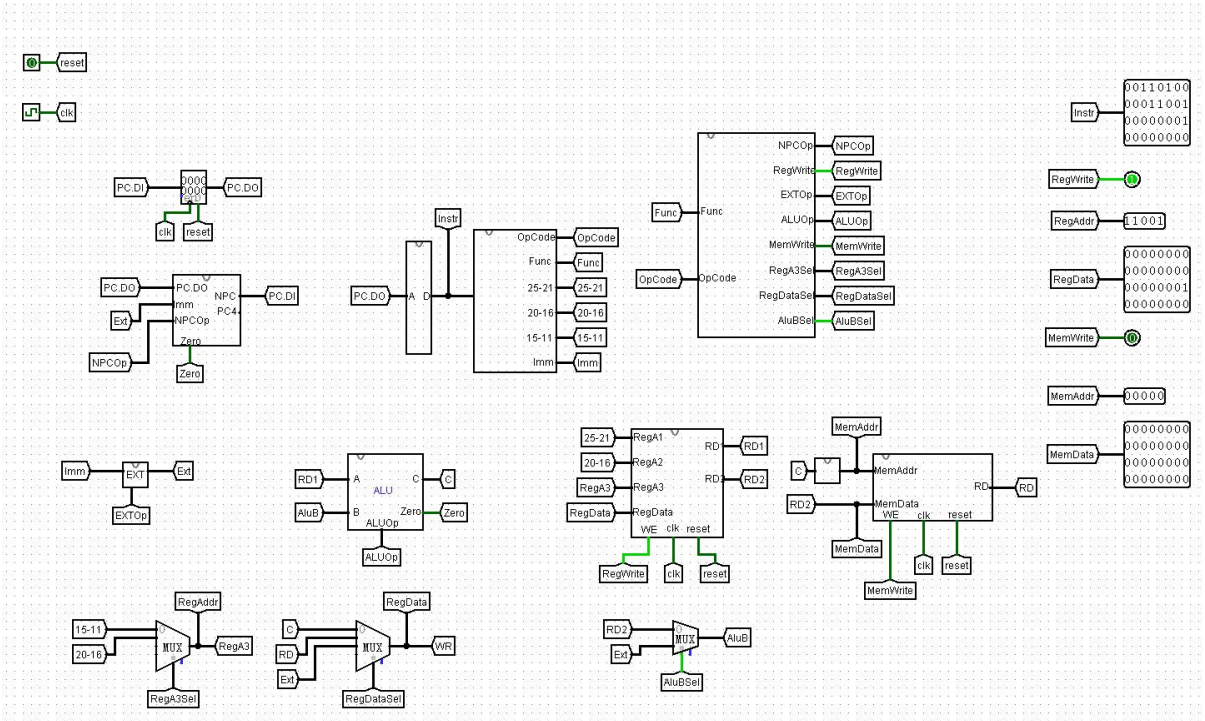

图 6 第一版顶层模块

（二）第二版

将顶层模块改用布线排布，各端口连接更为直观，但由于本人布线水平确实不高，顶层模块看起来仍较为杂乱。

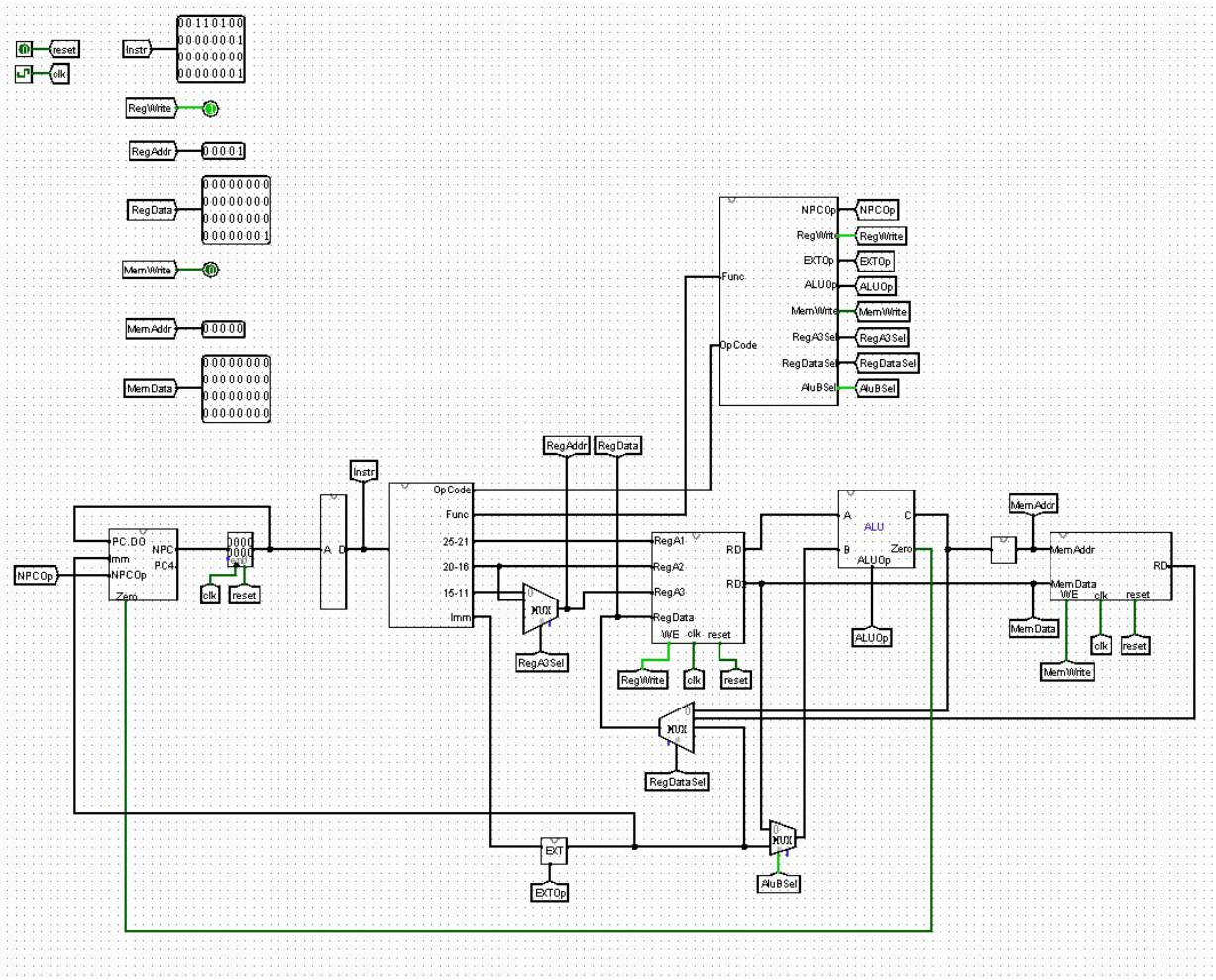

图 7 第二版顶层模块

（三）第三版

在原有七个指令集中增添了 jal、jr 指令，将 NPCOp 选择信号扩展了一位，顶层模块看起来有点乱，发现问题出在了次指令地址计算上，本应将其放入 NPC 模块，但在搭建过冲中误以为 PC 的扩展也要放入 EXT 模块，因此 NPC 的 Imm 端口只能与 EXT 的 Ext 端口相接，造成了混乱（时间来不及修改了）。

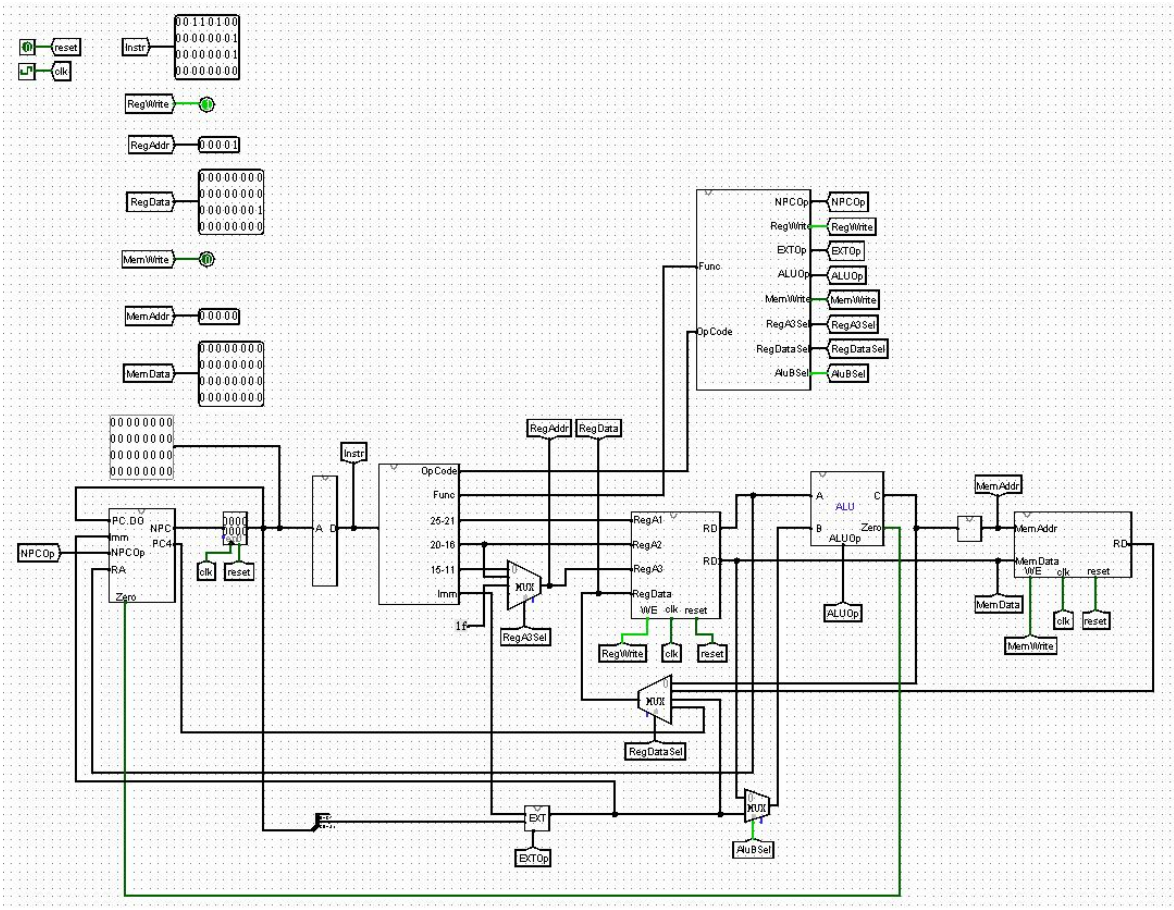
reset
clk
Instr
RegWrite
RegAddr
RegData
MemWrite
MemAddr
MemData
NPCOp
RegWrite
EXTOp
ALUOp
MemWrite
RegA3Sel
RegDataSel
AluBSel
Func
OpCode
PC_DO
NPC
PC4
Imm
RA
Zero
A
D
OpCode
Func
25-21
20-16
15-11
Imm
16
RegA1
RegA2
RegA3
RegData
WE
RD
RD2
ALU
A
B
C
Zero
MUX
RegA3Sel
RegDataSel
AluBSel
EXT
EXTOp

图 8 第三版顶层模块