# 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 |

#### 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 |

#### 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 |

1. 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 |

1. 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 |

1. 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 |

1. 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 |

1. 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 |

1. 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 |

### 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] |  |  |  |  |  |  |  |  |

1. 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 |  |  |  |  |  |  |  |

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

MIPS运行结果：

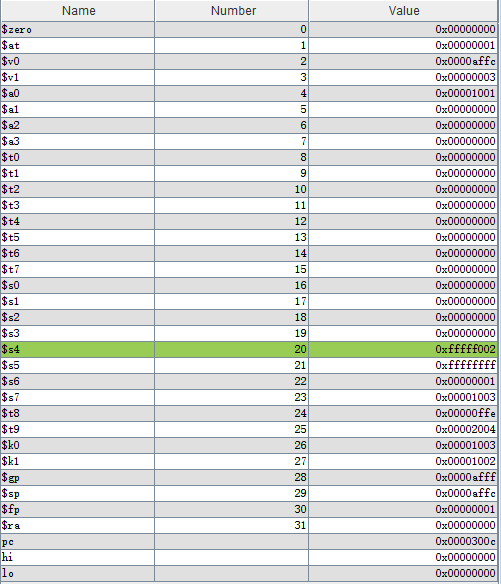


图 1 测试样例1的MIPS运行结果

#### ori、lui指令测试

|  |  |
| --- | --- |
| MIPS汇编指令 | 机器码 |
| ori $17, $0, 0  ori $18, $0, 256  ori $19, $18, 768  ori $17, $19, 0xffff  ori $28, $0, 1  ori $20, $28, 5  ori $31, $28, 6  ori $14, $20, 512  ori $28, $14, 0xaffc  ori $15, $30, 0xcbac  lui $6, 1  lui $7, 0xffff  lui $8, 0xaffc  lui $9, 256  lui $10, 0x1abc  lui $11, 0xbbca | 34110000  34120100  36530300  3671ffff  341c0001  37940005  379f0006  368e0200  35dcaffc  37cfcbac  3c060001  3c07ffff  3c08affc  3c090100  3c0a1abc  3c0bbbca |

MIPS运行结果：

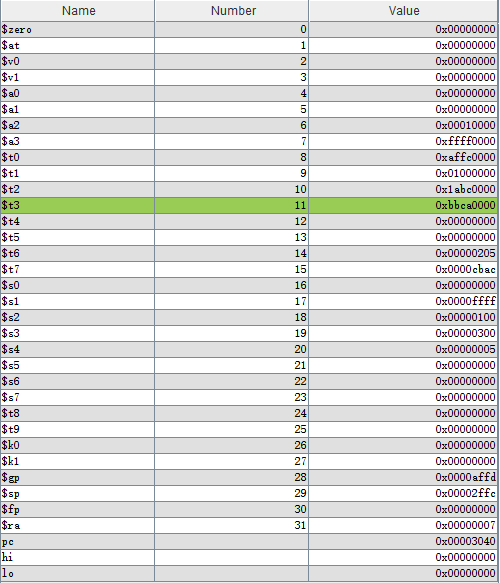


图 2 测试样例2的MIPS运行结果

#### lw、sw指令测试

|  |  |
| --- | --- |
| MIPS汇编指令 | 机器码 |
| ori $14, $0, 124  ori $15, $0, 4  ori $16, $0, 32  ori $1, $0, 0x1ffc  ori $2, $0, 256  ori $3, $0, 0xaffc  sw $1, 0($14)  sw $1, 4($16)  sw $2, -4($16)  sw $1, 64($0)  sw $3, 0($14)  sw $3, 4($16)  sw $3, -4($16)  lw $15, 4($16)  lw $16, -4($16)  lw $17, 0($14)  lw $18, 64($0)  lw $19, 0($0) | 340e007c  340f0004  34100020  34011ffc  34020100  3403affc  adc10000  ae010004  ae02fffc  ac010040  adc30000  ae030004  ae03fffc  8e0f0004  8e10fffc  8dd10000  8c120040  8c130000 |

MIPS运行结果：

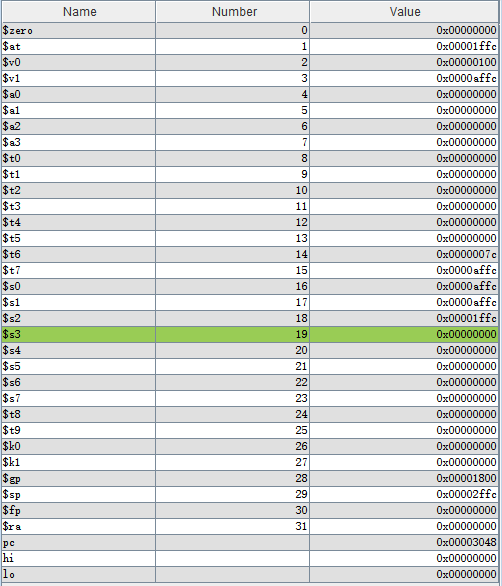


图 3 测试样例3的MIPS运行结果

1. beq、nop指令测试

|  |  |
| --- | --- |
| MIPS汇编指令 | 机器码 |
| ori $25, $0, 256  ori $27, $0, 14  addu $28, $25, $27  addu $29, $28, $28  label1:  addu $30, $28, $25  subu $31, $29, $27  ori $25, $25, 1  ori $27, $27, 2  nop  beq $25, $27, label1  beq $30, $31, label2  addu $14, $27, $27  label2:  ori $4, $0, 5  nop  addu $5, $4, $4 | 34190100  341b000e  033be021  039ce821  0399f021  03bbf823  37390001  377b0002  00000000  133bfffa  13df0001  037b7021  34040005  00000000  00842821 |

MIPS运行结果：

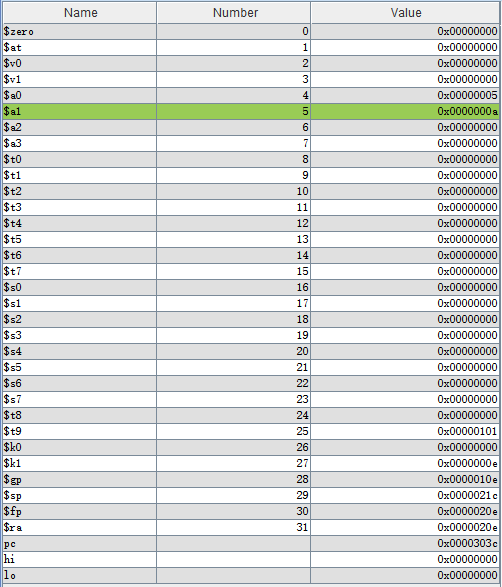


图 4 测试样例4的MIPS运行结果

1. 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  ori $2, $0, 512  addu $3, $1, $2  jal table1  lui $5, 0xaffc  jal table2  addu $6, $1, $0  nop    table1:  addu $5, $3, $3  subu $27, $3, $5  jr $ra  table2:  ori $7, $5, 101  addu $8, $7, $7  jr $ra | 34010100  34020200  00221821  0c000008  3c05affc  0c00000b  00203021  00000000  00632821  0065d823  03e00008  34a70065  00e74021  03e00008 |

MIPS运行结果：

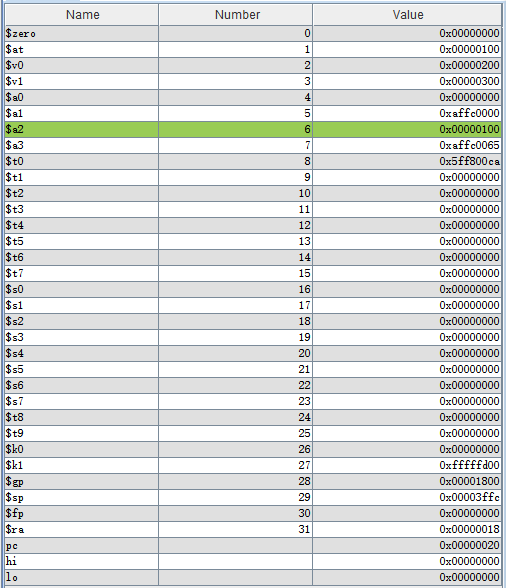


图 5 测试样例5的MIPS运行结果

## 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。

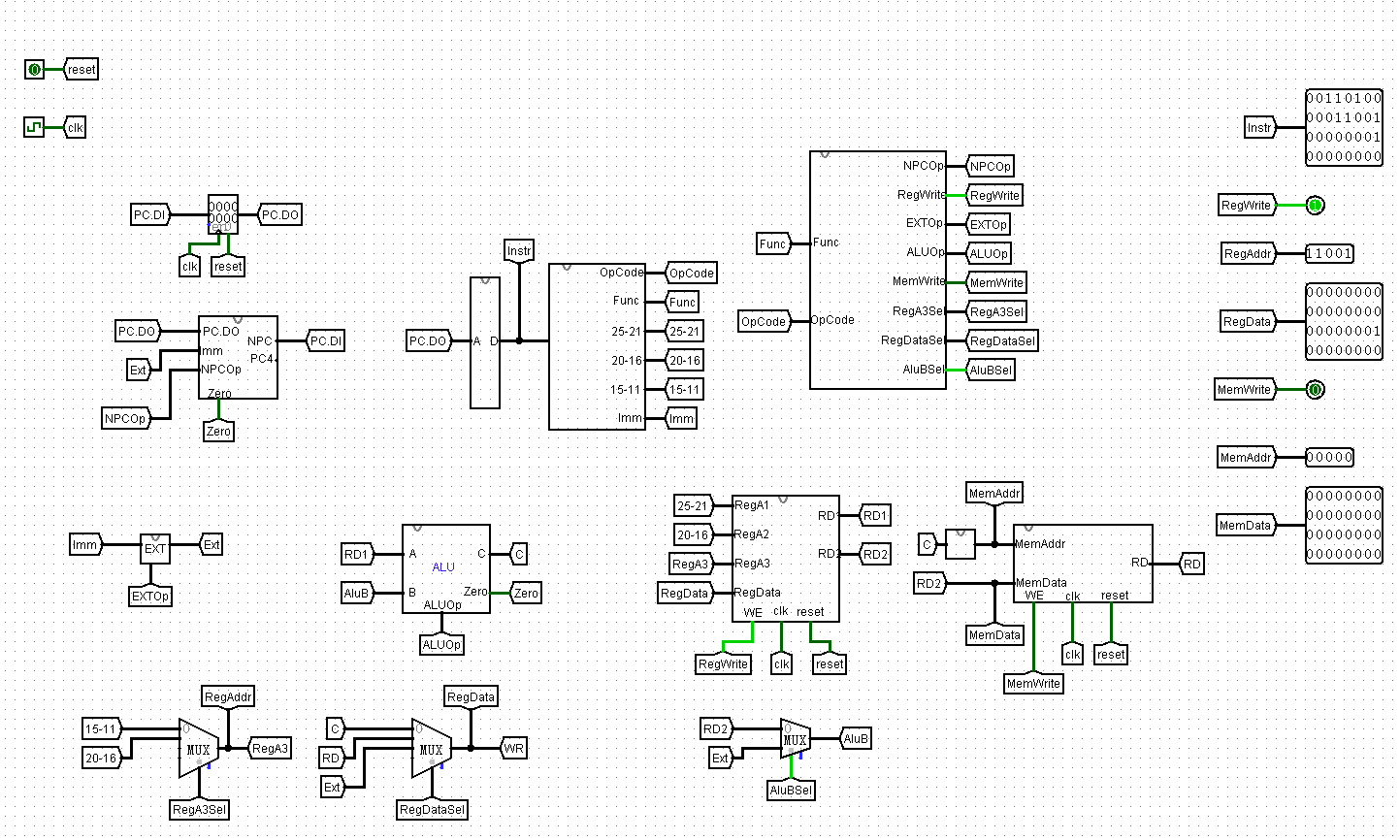


图 6 第一版顶层模块

（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。

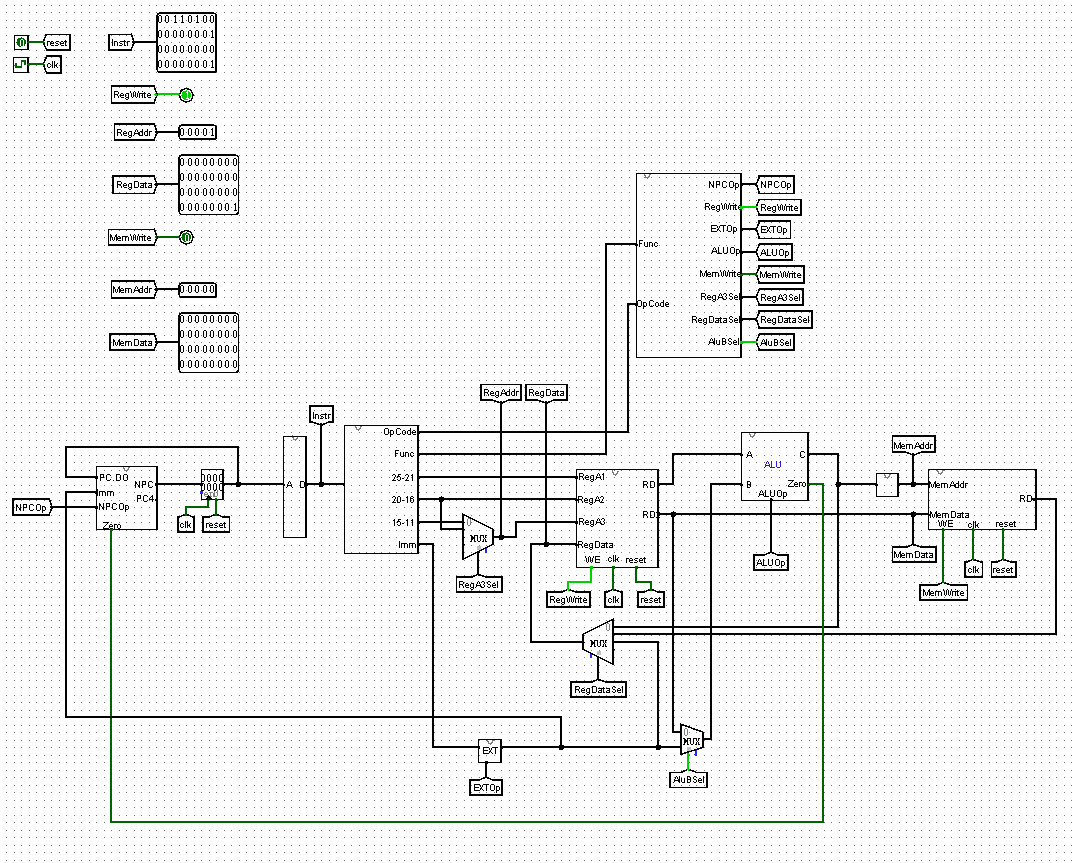


图 7 第二版顶层模块

（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）。

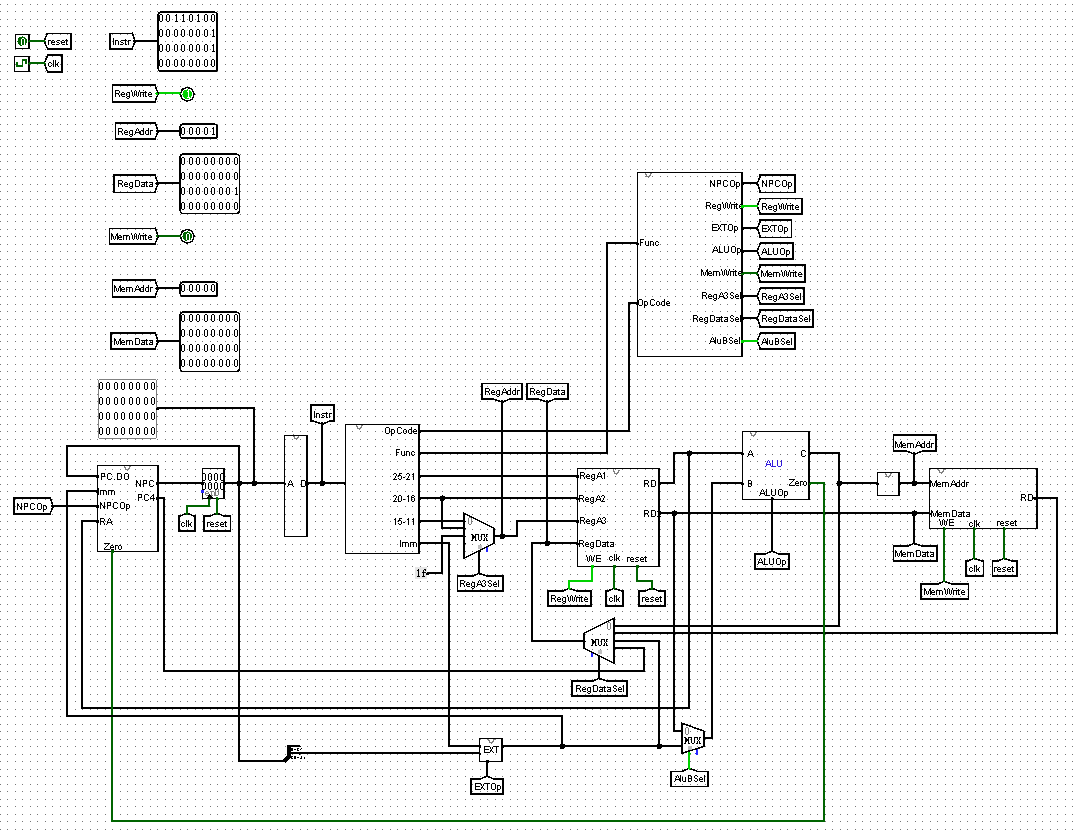


图 8 第三版顶层模块