计算机组成原理实验报告—单周期CPU设计

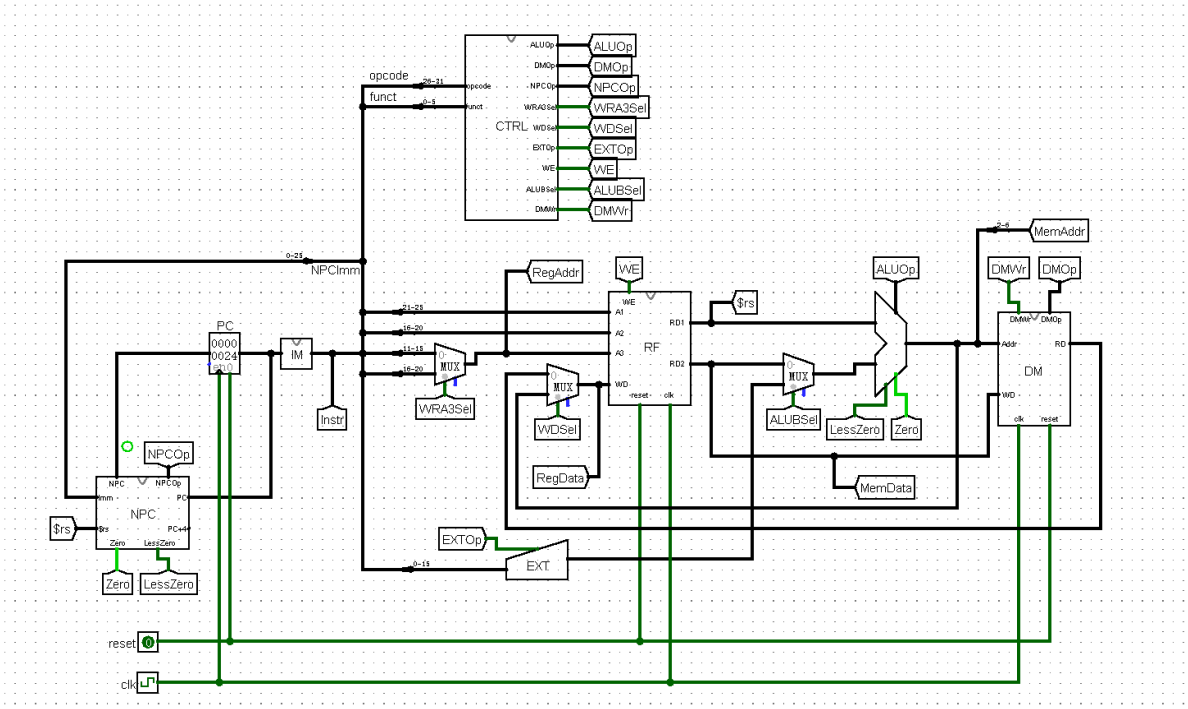
CPU设计方案综述

总体设计概述

要求实现的指令集包括**addu, subu, ori, lw, sw, beq, lui, nop，sll**

~~TODO：自行添加的指令将会包括**slt，jal，jr，bltz，lb，sb，and**~~

整体架构图如下：

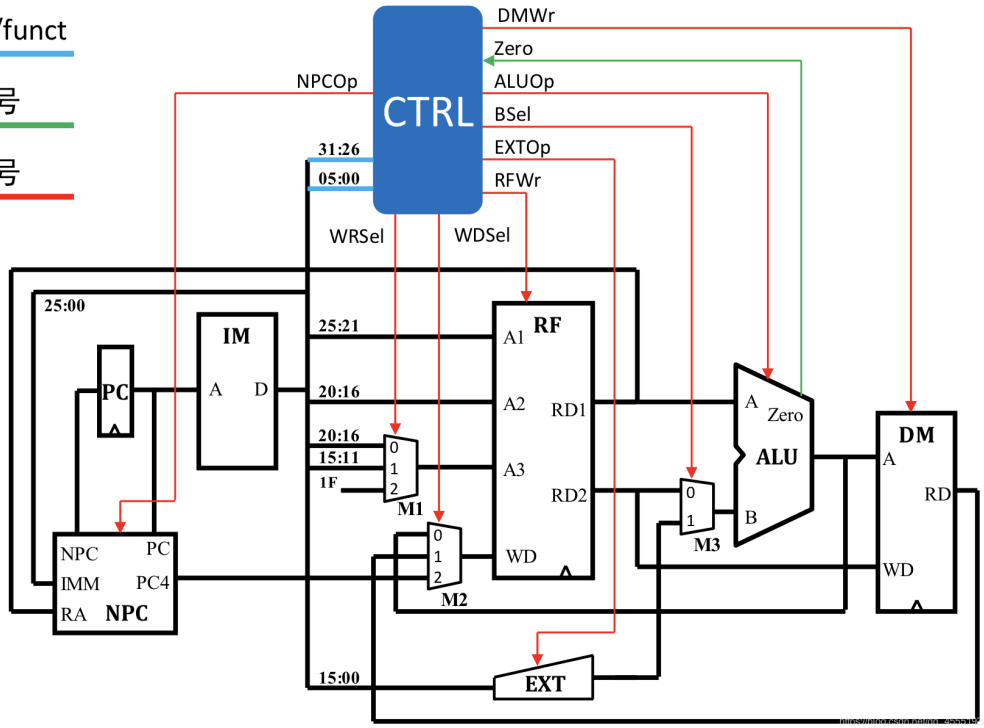


参考了课件中整体架构：

输入：opcode/funct

输入：状态信号

输出：控制信号



关键模块定义

GRF（寄存器堆）

端口说明

信号名称	方向	功能描述
A1[4:0]	输入	5位地址输入信号，将其储存的数据读出到RD1
A2[4:0]	输入	5位地址输入信号，将其储存的数据读出到RD2
A3[4:0]	输入	5位地址输入信号，将其作为写入数据的目标寄存器
RD1[31:0]	输出	输出A1指定的寄存器中的32位数据
RD2[31:0]	输出	输出A2指定的寄存器中的32位数据
WD[31:0]	输入	32位数据输入信号
WE	输入	写使能信号；1：写入有效；0：写入无效
clk	输入	时钟信号
reset	输入	异步复位信号，将32个寄存器中的数据清零；1：复位；0：无效

控制信号说明

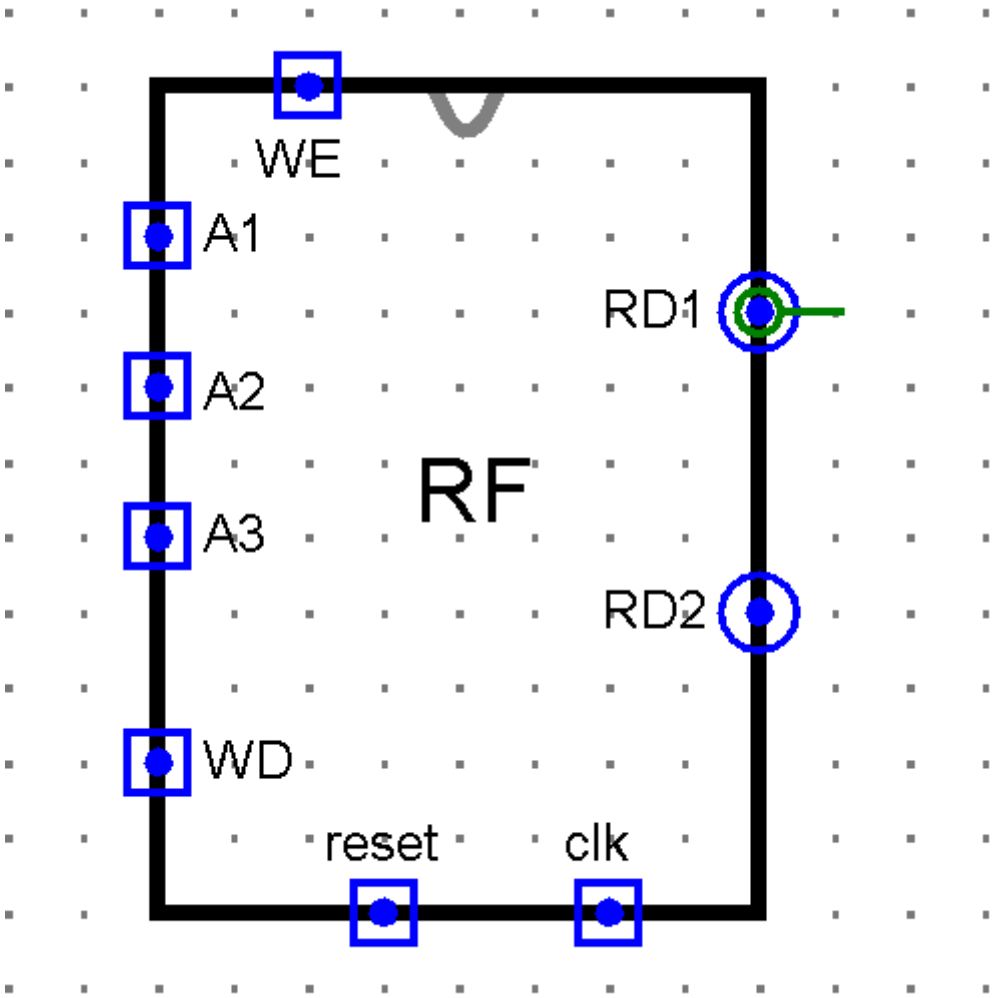
1. WRA3Sel

控制信号值	功能
`2'b00`	选择待写入寄存器地址来自`Instr[15:11]`
`2'b01`	选择待写入寄存器地址来自`Instr[20:16]`
`2'b10`	选择写入寄存器的地址为31(`$ra`)

2. WDSel

控制信号值	功能
`2'b00`	选择写入寄存器的数据来自DM
`2'b01`	选择写入寄存器的数据来自ALU运算结果
`2'b10`	选择写入寄存器的数据为PC+4

元件示意图



PC（程序计数器）

端口说明

信号名称	方向	功能描述
clk	输入	时钟信号
reset	输入	异步复位信号
PCIn	输入	输入NPC
PCOut	输出	输出当前指令地址

NPC（次地址计算单元）

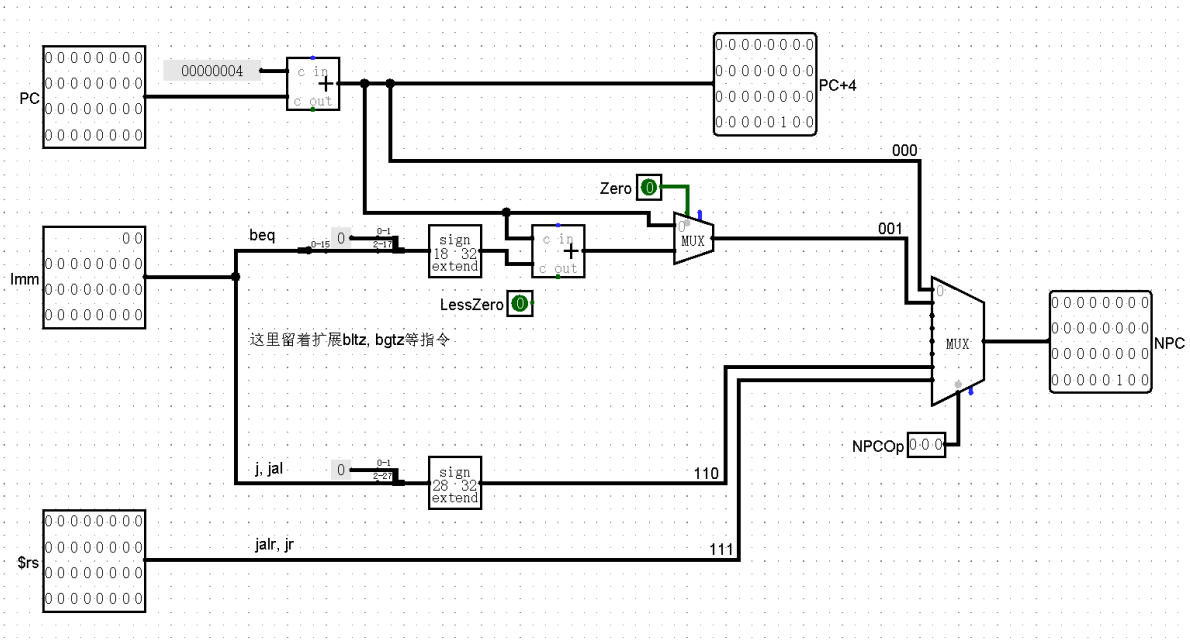
端口说明

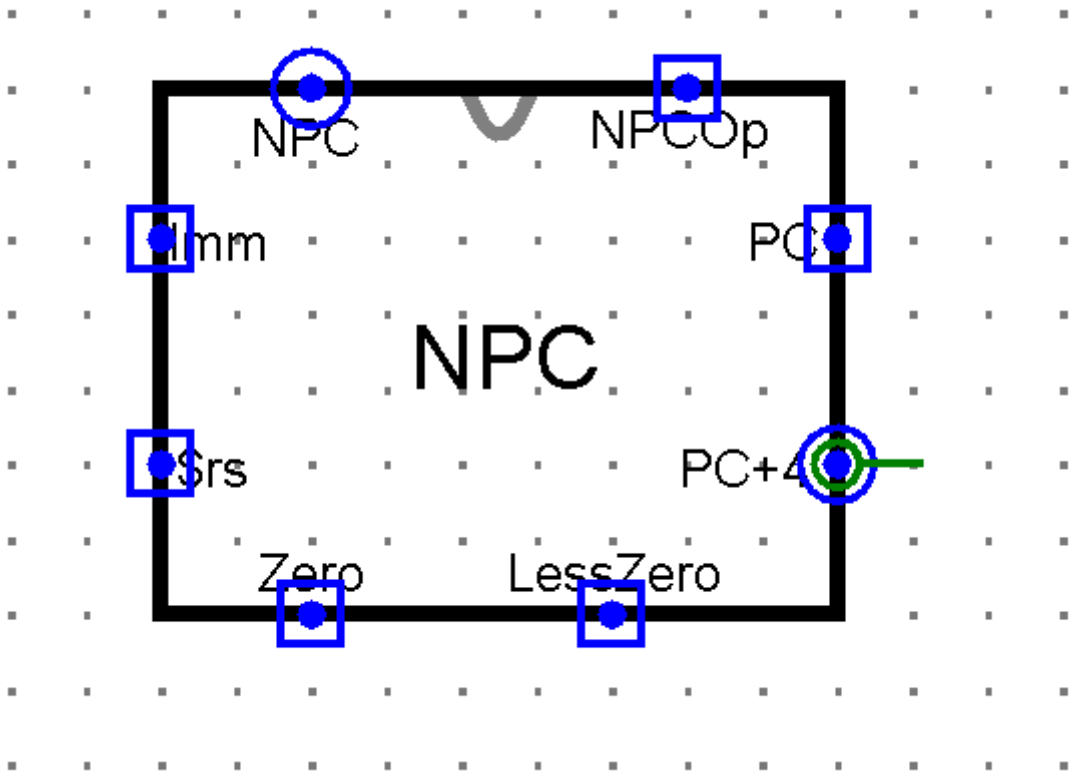
信号名称	方向	功能描述
PC[31:0]	输入	32位输入当前地址
NPC[31:0]	输出	32位输出次地址
NPCOp[1:0]	输入	控制信号
RA[31:0]	输入	`$ra`寄存器保存的32位地址
Zero	输入	`$rs`与`$rt`是否相等的标志；1：相等；0：不等
LessZero	输入	`$rs`是否小于`$rt`的标志；1：小于；0：大于或等于
PC+4[31:0]	输出	输出PC+4的值

控制信号说明

控制信号值	功能
`3'b000`	`NPC=PC+4`
`3'b001`	执行`beq`指令
`3'b110`	执行`j`，`jal`指令
`3'b111`	执行`jalr`，`jr`指令

元件示意图





ALU（算术逻辑单元）

端口说明

信号名称	方向	功能描述
A[31:0]	输入	32位输入运算数A
B[31:0]	输入	32位输入运算数B
C[31:0]	输出	32位输出运算结果
ALUOp[3:0]	输入	控制信号
Zero	输出	若$A-B=0$则置为1，否则置为0
LessZero	输出	若$A<B$则置为1，否则置为0

控制信号说明

1. ALUOp

控制信号值	功能
`3'b000`	执行加法运算
`3'b001`	执行减法运算
`3'b010`	执行逻辑或运算
`3'b011`	执行`sll`运算
`3'b100`	执行`lui`指令

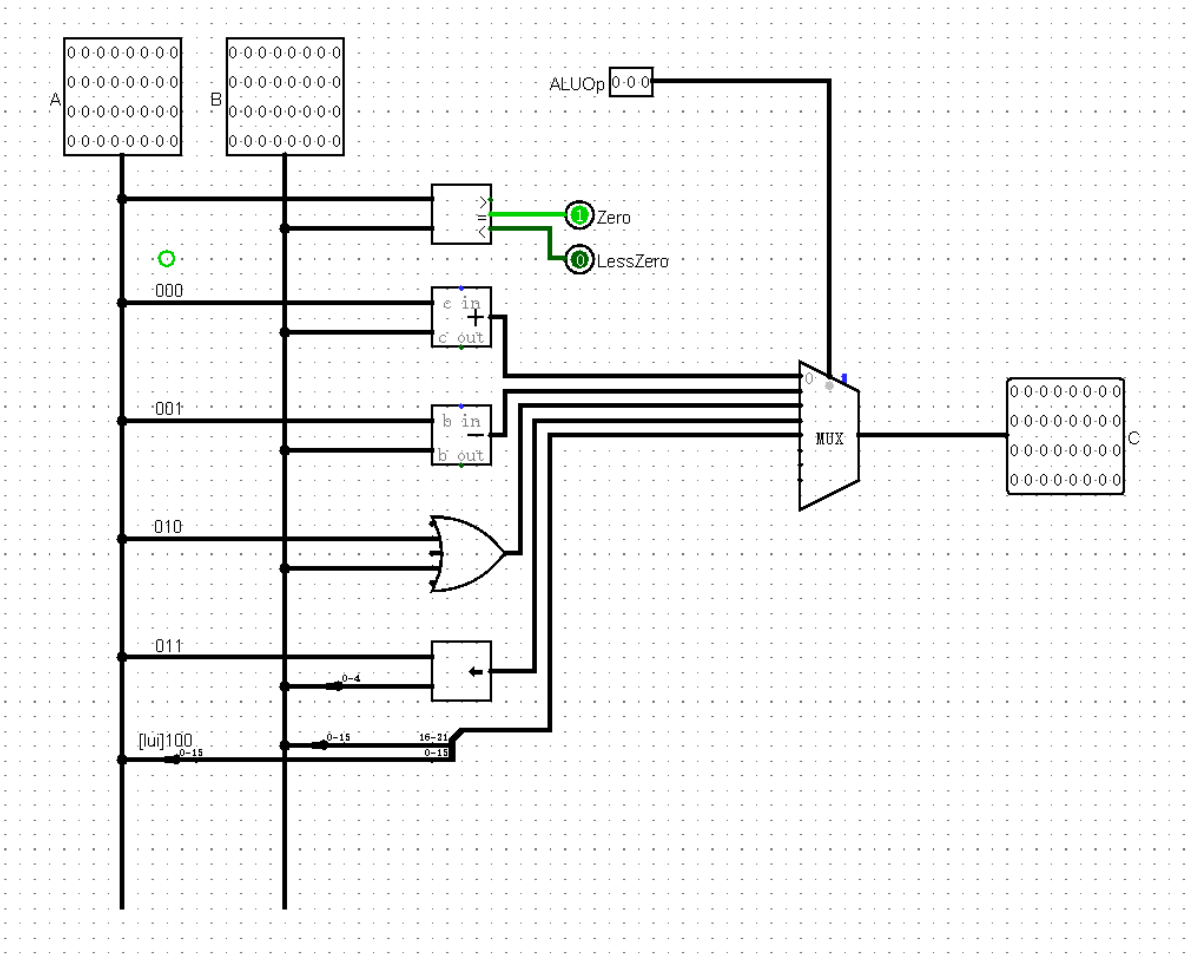
2. ALUBSel

控制信号值	功能
`1'b0`	选择寄存器中的值进行运算
`1'b1`	选择立即数进行运算

3. ALUASel

控制信号值	功能
`1'b0`	选择寄存器中的值进行运算
`1'b1`	选择 `sll` 指令中移位的位数

元件示意图



EXT（位扩展）

将16位二进制数进行零扩展或符号扩展到32位

控制信号说明

控制信号值	功能
`1'b0`	零扩展
`1'b1`	符号扩展

IM（指令储存器）

将PC中储存的对应地址的指令取出

DM（数据储存器）

端口说明

信号名称	方向	功能描述
Addr[31:0]	输入	待操作的内存地址
WD[31:0]	输入	待写入内存的值
clk	输入	时钟信号
reset	输入	异步复位信号
DMWr	输入	写使能信号；1：写入有效；0：写入无效
DMOp	输入	控制信号
RD[31:0]	输出	输入地址指向的内存中储存的值

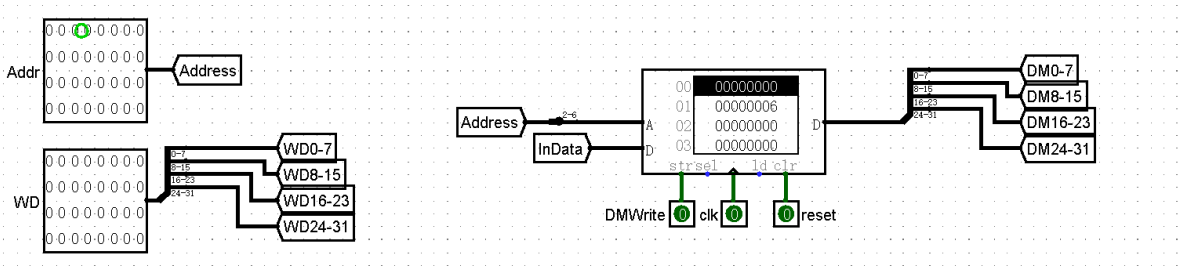
控制信号说明

控制信号值	功能
`2'b00`	对应 `lw` 和 `sw` 指令，写入或读取整个字
`2'b01`	（保留）对应 `lh` 和 `sh` 指令，写入或读取半字
`2'b10`	（保留）对应 `lb` 和 `sb` 指令，写入或读取整个字

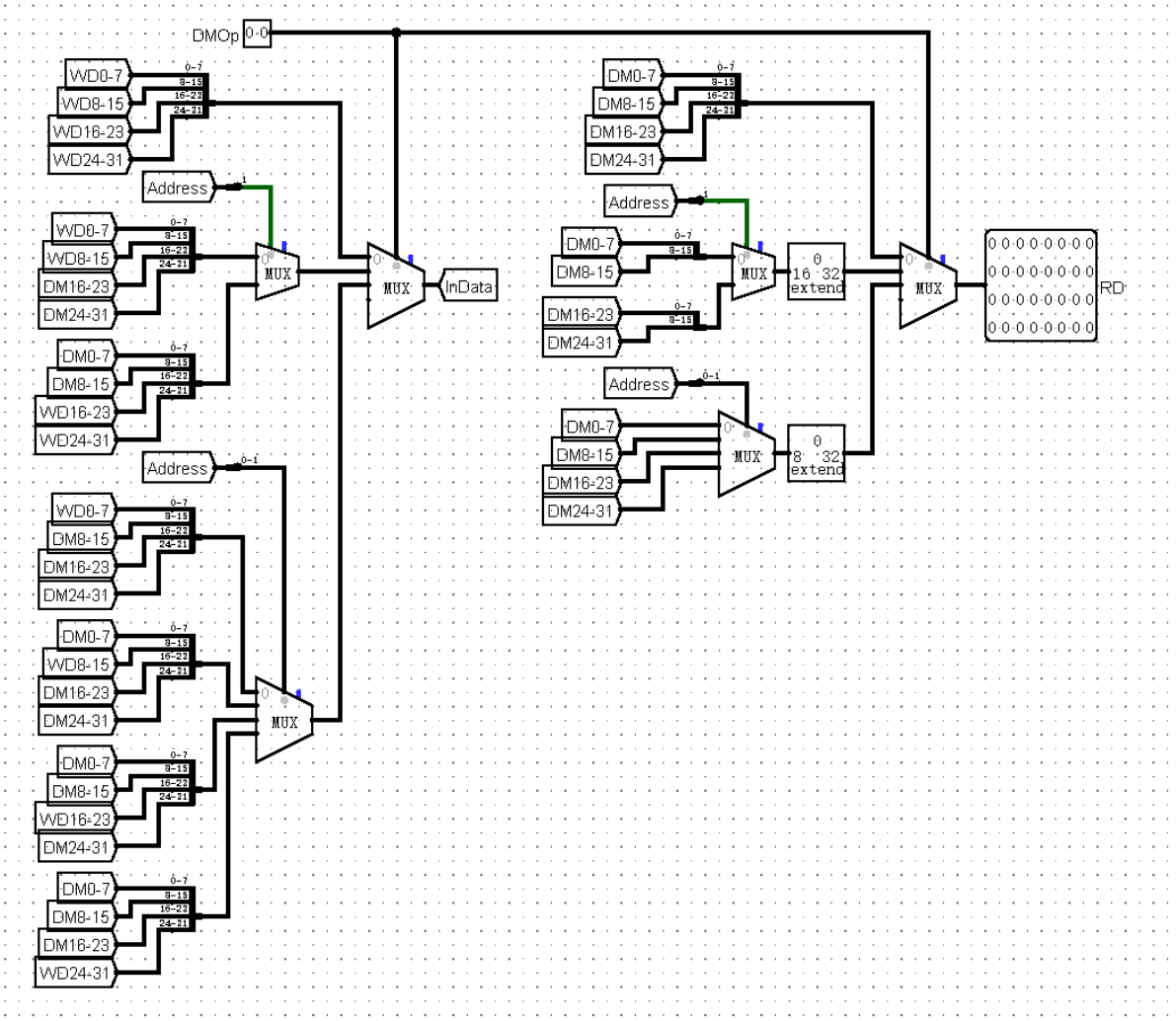
也许对于 `lbu` 指令还需要添加额外的控制信号

元件示意图

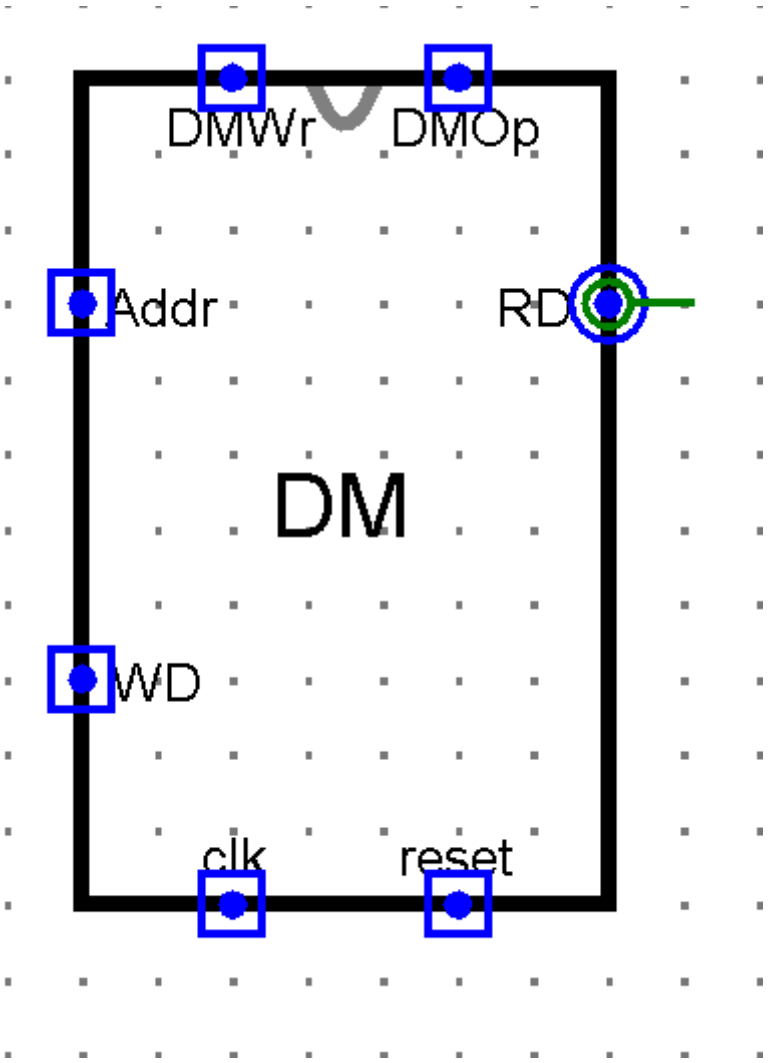
主体部分：



对于 `lh`，`sh`，`lb`，`sb` 的特殊处理部分


DMOp
WD0-7
WD8-15
WD16-23
WD24-31
Address
MUX
InData
DM0-7
DM8-15
DM16-23
DM24-31
0
16 32
extend
8 32
extend
RD

整体:



数据通路分析

指令	opcode	funct	NPCOp	WRA3Sel	WDSel	EXTOp	WE	ALUASel	ALUBSel	ALUOp	DMWr	DMOp
addu	000000	100001	000	0	1	X	1		0	000	0	X
subu	000000	100011	000	0	1	X	1		0	001	0	X
ori	001101		000	1	1	0	1		1	010	0	X
lw	100011		000	1	0	1	1		1	000	0	00
sw	101011		000	1	0	1	0		1	000	1	00
beq	000100		001	X	X	X	0		0	001	0	X
lui	001111		000	1	1	X	1		1	100	0	X

调试过程记录

- 课下测试很弱，需要注意的点就是刚开始看清楚`beq`指令的RTL语言，`beq`中的立即数不是绝对地址，而是偏移量`offset`，真正跳转的地址为`addr = PC + 4 + offset`
- 在自动化测试时，如何让Logisim停止模拟？可以添加一个输出端口`halt`，当这个端口恒置为1时，Logisim会自动检测到，停止仿真模拟

调试方法与辅助工具

综合讨论区中的代码，修改不合适之处，综合形成了自动化测试工具

利用`C++`生成MIPS汇编代码（来源：评论区，原代码不能直接使用，做了一些修改）

```
// Filename: code_generate.cpp
// Update on 2021 Nov. 8
```

```cpp
// Fix some bugs and change the code for the test in Computer Organization course in Fall, 2021
#include<fstream>
#include<vector>
#include<algorithm>
#include<cstdlib>
#include<ctime>
using namespace std;
ofstream cout("mips_code.txt");
struct op{      // type=0:nop, type=1:addu, type=2:subu, type=3:lui, type=4:ori, type=5:lw, type=6:sw, type=7:beq
    int type,r1,r2,r3;
    op(){}
    op(int a1,int a2,int a3,int a4){
        type=a1,r1=a2,r2=a3,r3=a4;
    }
};
int size_im,size_dm,size_op,small_reg;
op a[10010];
vector<int> branch[10010];
int is_small[40],small[40],val[40],cnt;
int r(int a,int b,int except1){  // Randomly generate a register except the $1($at), because $1 may cause some unexpected errors
                                 // Function: Generate a random number in range (a, b), except number 1 when except1=1
    int t0=rand()*2+rand()%2;
    int res=t0%(b-a+1)+a;
    return ((res==1)&&except1==1)?0:res;
}
int t[10010];
void get_small(){               // Randomly select some registers for process 'lw' and 'sw' in the test
    int i;
    for(i=1;i<=26;i++) t[i]=i;
    for(i=1;i<=26;i++) swap(t[i],t[rand()%30+1]);
    for(i=1;i<=small_reg;i++) small[i]=t[i]+1,is_small[t[i]+1]=1;
}
int nosmall(){                  // Randomly generate a register which is not in the array small[]
    int u=r(0,27,1);
    while(is_small[u]==1){u=r(0,27,1);}
    return u;
}
void print(int x){
    if(a[x].type==0) cout<<"nop"<<endl;
    if(a[x].type==1) cout<<"addu $"<<a[x].r1<<",$"<<a[x].r2<<",$"<<a[x].r3<<endl;
    if(a[x].type==2) cout<<"subu $"<<a[x].r1<<",$"<<a[x].r2<<",$"<<a[x].r3<<endl;
    if(a[x].type==3) cout<<"lui $"<<a[x].r1<<","<<a[x].r3<<endl;
    if(a[x].type==4) cout<<"ori $"<<a[x].r1<<",$"<<a[x].r2<<","<<a[x].r3<<endl;
    if(a[x].type==5) cout<<"lw $"<<a[x].r1<<","<<a[x].r3<<"($"<<a[x].r2<<")"<<endl;
    if(a[x].type==6) cout<<"sw $"<<a[x].r1<<","<<a[x].r3<<"($"<<a[x].r2<<")"<<endl;
    if(a[x].type==7) cout<<"beq $"<<a[x].r1<<",$"<<a[x].r2<<",branch"<<a[x].r3<<endl;
}
```

```
int main(){
    int i,j;
    srand(time(NULL));
    size_im=32,size_dm=32,size_op=8,small_reg=8;        // The size of IM and DM
is 32 in the test

    get_small();
    for(i=1;i<=small_reg;i++)                           // Firstly, use 'ori' to
assign primary data for the registers
        a[i]=(op(4,small[i],0,val[small[i]]=r(0,size_dm-1,0)));
    for(i=small_reg+1;i<=size_im;i++){
        int op0=r(0,size_op-1,0),r1,r2,r3;
        if(op0==0) a[i]=(op(0,0,0,0));
        if(op0==1||op0==2){                             // 'addu' and 'subu'
            a[i]=op(op0,nosmall(),r(0,27,1),r(0,27,1));
        }
        if(op0==3||op0==4){                             // 'lui' and 'ori'
            a[i]=(op(op0,nosmall(),r(0,27,1),r(0,65535,0)));
        }
        if(op0==5){                                     // 'lw'
            r1=r(0,size_dm-1,0);
            r2=r(1,small_reg,0);
            a[i]=(op(op0,nosmall(),small[r2],(r1-val[small[r2]])*4));
        }
        if(op0==6){                                     // 'sw'
            r1=r(0,size_dm-1,0);
            r2=r(1,small_reg,0);
            a[i]=(op(op0,r(0,27,1),small[r2],(r1-val[small[r2]])*4));
        }
        if(op0==7){                                     // 'beq' branch above
may cause bugs, just generate branch below
            r3=r(i,size_im,0);
            a[i]=(op(op0,r(0,27,1),r(0,27,1),++cnt));
            branch[r3].push_back(cnt);                  // Randomly put the
branch tags
        }
    }
    for(i=1;i<=size_im;i++) {
        print(i);
        for(j=0;j<branch[i].size();j++)
            cout << "branch" << branch[i][j] << ":" << endl;
    }
    return 0;
}
```

然后利用`Python`程序去运行Mars导出机器码，并将机器码导入Logisim电路ROM中，同时读取并格式化输出数据，进行比较

```
# Filename: auto_judger.py
import os
import re

# 由于测试程序需要直接修改电路源代码文件，请务必提前进行备份

def logProcess(log,dest):           # 感谢评论区提供的格式化输出代码
    logText=log.readlines()
```

```
    for line in logText:
        linestr=re.sub(r"\s+","",line)
        dest.write("@"+"{:>08x}".format(int(linestr[0:32],2))+":\t")
        if(linestr[32]=='1'):
            dest.write("$"+"{:2d}".format(int(linestr[33:38],2))+" <= "+"
{:>08x}".format(int(linestr[38:70],2))+"\t")
        if(linestr[70]=='1'):
            dest.write("#"+"{:>08x}".format(int(linestr[71:76],2))+" <= "+"
{:>08x}".format(int(linestr[76:108],2))+"\t")
        dest.write("\n")

if __name__ == '__main__':

    # -------------------- Prepare the testbench --------------------------
    WorkDir = "G:/MyWorkspace/Computer_Organization/P3/Auto_Judge"
    os.chdir(WorkDir)
    os.system(".\\code_generate.exe")
    os.system("java -jar .\\mars.jar mips_code.txt nc mc CompactTextAtZero a
dump .text HexText mips_hex_code.txt")

    # -------------------- Prepare the circuit --------------------------
    testbench = open("mips_hex_code.txt", "r").read()
    src = open("src.circ", "r", encoding='utf8').read()
    src_rep = re.sub(r'addr/data: 5 32([\s\S]*)</a>', "addr/data: 5 32\n"+
testbench +"</a>", src)
    open("src.circ", "w").write(src_rep)
    std = open("std.circ", "r", encoding='gbk').read()
    std_rep = re.sub(r'addr/data: 5 32([\s\S]*)</a>', "addr/data: 5 32\n"+
testbench +"</a>", std)
    open("std.circ", "w").write(std_rep)

    # ---------------------- Running --------------------------
    os.system("java -jar .\\logisim.jar .\\src.circ -tty table speed  >
src_running_res.txt")
    os.system("java -jar .\\logisim.jar .\\std.circ -tty table speed  >
std_running_res.txt")
    # os.system("fc src_running_res.txt std_running_res.txt")
    login = open('src_running_res.txt', 'r')
    logdest = open('src_info.txt', 'w')
    logProcess(login, logdest)
    login = open('std_running_res.txt', 'r')
    logdest = open('std_info.txt', 'w')
    logProcess(login, logdest)
```

最后用Windows批处理脚本重复运行，比较输出

```
echo off
:loop
python auto_judger.py
fc src_info.txt std_info.txt
goto loop
```

思考题

思考题1 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。

解答 部分合理；IM只需被读取，ROM只有读取功能；DM既要进行读取，又要进行写入，但是一个周期只会进行读取和写入之一，RAM的单一地址和各一个的读写端口满足了这种要求。用寄存器也能实现DM，但是DM需要较大的空间，使用寄存器太"浪费"；GRF需要读写，且其与ALU直接连接，需要高速地读写，故使用寄存器堆搭建合理。但是事实上，现代计算机都是指令和数据放在同一个存储器下的，因此存在必合理，IM和DM使用同一个RAM也有合理之处。

思考题2 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？

解答 `sll $0, $0, 0` 对应的指令码是 `0x0000_0000`，也被认为是 `nop`(空操作指令)。 该指令有时被用于空循环，有时被编译器用于与体系结构相关的编译优化。

思考题3.1 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。

解答 如果寄存器中存储的DM的地址被映射在 `0x3000_0000` 到 `0x3fff_ffff` 间，而我们的DM起始地址是0，那么，我们可以将输入地址用一个减法器直接减去0x3000_0000，再作为DM的地址输入。

思考题3.2 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。

解答

形式验证的优点如下：

- 所有可能的情况进行验证，覆盖率达到了100%。
- 形式验证的验证时间短，可以很快发现和改正电路设计中的错误，可以缩短设计周期。

形式验证的缺点如下：

- 形式验证只能检验电路设计的正确性，却无法检验其它方面如电路能耗等的优劣。