2021年北航计组P3实验报告

1 整体架构设计

1.1 CPU设计方案综述

本实验基于logisim实现了单周期cpu，支持指令集{addu, addiu, ori, and, subu, lw, sw, lb, sb, beq, blez, lui, , sll, slt, j, jr, jal, nop}，包含RF, IM, NPC, ALU, DM, EXT, MW, MR, CTR等模块

1.2 关键模块

1.2.1 RF

端口	输入/输出	位宽	描述
A1	I	5	指定 32 个寄存器中的一个，将其存储的数据读出到 RD1
A2	I	5	指定 32 个寄存器中的一个，将其存储的数据读出到 RD2
A3	I	5	指定 32 个寄存器中的一个，作为写入的目标寄存器
WD	I	32	写入寄存器的数据信号
WE	I	1	写使能信号，高电平有效
clk	I	1	时钟信号
reset	I	1	异步复位信号
RD1	O	32	输出 A1 指定的寄存器中的数据
RD2	O	32	输出 A2 指定的寄存器中的数据

序号	功能名称	功能描述
1	异步复位	当异步复位信号有效时，将所有寄存器的值设置为 0x00000000
2	读数据	读出 A1 和 A2 地址对应寄存器中存储的数据到 RD1 和 RD2
3	写数据	当 WE 有效且时钟上升沿到来时，将 WD 的数据写入A3 对应的寄存器中

1.2.2 IM

端口	输入/输出	位宽	描述
PC	I	32	设置下一个 PC 值

1.2.3 EXT

端口	输入/输出	位宽	描述
imm16	I	16	需要扩展的 16 位数据
EXT_op	I	2	EXT功能控制信号
EXT_OUT	O	32	将输入做扩展到 32 位的结果

序号	功能名称	功能描述
0	无符号扩展	将 imm16 输入的 16 位数据做无符号扩展
1	符号扩展	将 imm16 输入的 16 位数据无符号扩展
2	加载到高位	将imm16 输入的 16 位数据加载到 32 位输出的高位

1.2.4 NPC

端口	输入/输出	位宽	描述
imm	I	26	26 位立即数
NPC_op	I	3	NPC功能控制信号
cmp	I	1	rs寄存器与rt寄存器值的比较结果
PC	I	32	当前PC寄存器值
jr	I	32	寄存器rs的值，用于jr指令
PC4	O	32	输出PC + 4
NPC	O	32	下一个PC的值

序号	功能名称	功能描述
0	输出PC4	NPC = PC + 4
1	跳转b类型	对于branch类型的指令，输出下一个PC的值
2	跳转j类型	对于j，jal这样的指令，输出下一个PC的值
3	跳转寄存器类型	对于jr这样的指令，输出下一个PC的值

1.2.5 ALU

端口	输入/输出	位宽	描述
A	I	32	参与 ALU 计算的第一个值
B	I	32	参与 ALU 计算的第二个值
ALU_op	I	4	ALU 功能的选择信号，具体见功能定义
C	O	32	ALU 的计算结果
cmp	O	1	当 A 与 B 满足控制信号指定条件时为 1，否则为 0
shamt	I	5	对sll指令，确定移位的位数

功能定义

序号	功能名称	功能描述
0	加	C = A + B
1	减	C = A - B
2	逻辑左移	C = B << shamt
3	判断相等	cmp = (A==B)? 1:0
4	判断小于	cmp = (A<B)? 1:0
5	判断<=0	cmp = (A<=0)? 1:0
6	按位或	C = A | B
7	按位与	C = A & B

1.2.7 MW

端口	输入/输出	位宽	描述
A	I	32	写入数据的地址
WD	I	32	写入 DM 中的数据。
MW_op	I	3	控制信号
Mem	I	32	DM相应地址中的原数据
MW_OUT	O	32	写入DM_WD的数据

序号	功能名称	功能描述
1	写字	`mem[addr] <= WD`
2	写字节	`mem[addr][7+8*A[1:0] -:8] <= WD[7:0]`

1.2.8 MR

端口	输入/输出	位宽	描述
A	I	32	读取数据的地址
Mem	I	32	从 DM 中读出的数据。
MR_op	I	3	控制信号
MR_OUT	O	32	读出的数据

序号	功能名称	功能描述
1	读字	`MRout <= mem[addr]`
2	读字节	`MRout <= signed_ext(mem[addr][7+8*A[1:0] -:8])`

1.2.9 CTR

端口	输入/输出	位宽	描述
Instr	I	32	当前指令
NPC_op	O	3	NPC模块控制信号
RF_wr	O	1	RF写能使信号
RF_A3_sel	O	2	RF_A3 MUX选择信号
RF_WD_sel	O	2	RF_WD MUX选择信号
EXT_op	O	2	EXT模块控制信号
ALU_op	O	4	ALU模块控制信号
ALU_B_sel	O	1	ALU_B MUX选择信号
DM_wr	O	1	DM模块写能使信号
MW_op	O	3	MW模块控制信号
MR_op	O	3	MR模块控制信号

Control Signals Table

opcode	**000000**				**100011**	**101011**	**000100**	**000110**	**001101**	**001001**	**000010**	**000011**
func	100001	001000	000000	101010	x							
	addu	jr	sll	slt	lw	sw	beq	blez	ori	addiu	j	jal
NPC_op	**000**	**011**	**000**	**000**	**000**	**000**	**001**	**001**	**000**	**000**	**010**	**010**
EXT_op	**x**	**x**	**x**	**x**	**1**	**1**	**x**	**x**	**0**	**1**	**x**	**x**
RF_wr	**1**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**1**	**0**	**1**
RF_A3_sel	**00**	**x**	**00**	**00**	**01**	**x**	**x**	**x**	**01**	**01**	**x**	**10**
RF_WD_sel	**00**	**x**	**00**	**00**	**01**	**x**	**x**	**x**	**00**	**00**	**x**	**10**
ALU_B_sel	**0**	**x**	**x**	**0**	**1**	**1**	**0**	**x**	**1**	**1**	**x**	**x**
DM_wr	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**
ALU_op	**0000**	**x**	**0010**	**0100**	**0000**	**0000**	**0011**	**0101**	**0110**	**0000**	**x**	**x**

1.3 DataPath Table

CPU	PC	NPC	IM	RF	EXT	ALU	MW	DM	MR												
	NPC	Imm	PC	CMP	RD	PC	A1	A2	A3	WD	In	A	B	Shamt	A	WD	Mem	A	WD	A	Mem	
addu	NPC.NPC	NPC.NPC		PC.PC			PC.PC	IM.Istr[25:21]	IM.Istr[20:16]	IM.Istr[15:11]	ALU.C		RF.RD1	RF.RD2									
lw	NPC.NPC		PC.PC			PC.PC	IM.Istr[25:21]		IM.Istr[20:16]	MR.Out	IM.Istr[15:0]	RF.RD1	EXT.Out[31:0]					ALU.C			DM.RD	
lb	NPC.NPC		PC.PC			PC.PC	IM.Istr[25:21]		IM.Istr[20:16]	MR.Out	IM.Istr[15:0]	RF.RD1	EXT.Out[31:0]					ALU.C		ALU.C	DM.RD	
sw	NPC.NPC		PC.PC			PC.PC	IM.Istr[25:21]	IM.Istr[20:16]			IM.Istr[15:0]	RF.RD1	EXT.Out[31:0]			RF.RD2		ALU.C	MW.Out			
sb	NPC.NPC		PC.PC			PC.PC	IM.Istr[25:21]	IM.Istr[20:16]			IM.Istr[15:0]	RF.RD1	EXT.Out[31:0]		ALU.C	RF.RD2	DM.RD	ALU.C	MW.Out			
addiu	NPC.NPC		PC.PC			PC.PC	IM.Istr[25:21]		IM.Istr[20:16]	ALU.C	IM.Istr[15:0]	RF.RD1	EXT.Out[31:0]									
ori	NPC.NPC		PC.PC			PC.PC	IM.Istr[25:21]		IM.Istr[20:16]	ALU.C	IM.Istr[15:0]	RF.RD1	EXT.Out[31:0]									
beq	NPC.NPC	IM.Istr[15:0]	PC.PC	ALU.cmp		PC.PC	IM.Istr[25:21]	IM.Istr[20:16]				RF.RD1	RF.RD2									
blez	NPC.NPC	IM.Istr[15:0]	PC.PC	ALU.cmp		PC.PC	IM.Istr[25:21]					RF.RD1										
jr	NPC.NPC		PC.PC		RF.RD1	PC.PC	IM.Istr[25:21]															
jal	NPC.NPC	IM.Istr[25:0]	PC.PC			PC.PC			5'd31	NPC.PC4												
j	NPC.NPC	IM.Istr[25:0]	PC.PC			PC.PC																
sll	NPC.NPC		PC.PC			PC.PC		IM.Istr[20:16]	IM.Istr[15:11]	ALU.C		RF.RD2		IM.Istr[10:6]								
slt	NPC.NPC		PC.PC			PC.PC	IM.Istr[25:21]	IM.Istr[20:16]	IM.Istr[15:11]	ALU.C		RF.RD1	RF.RD2									
syn		NPC.NPC	IM.Istr[25:0]	PC.PC	ALU.cmp	RF.RD1	PC.PC	IM.Istr[25:21]	IM.Istr[20:16]	IM.Istr[15:11] IM.Istr[20:16] 5'd31	ALU.C MR.Out NPC.PC4	IM.Istr[15:0]	RF.RD1 RF.RD2	RF.RD2 EXT.Out[31:0]	IM.Istr[10:6]	ALU.C	RF.RD2	DM.RD	ALU.C	RF.RD2 MW.Out	ALU.C	DM.RD

2 测试方案

通过手造数据，现成数据，自动生成数据相结合的方式生成数据，并利用python脚本与mars对拍实现自动化测试

2.1 测试程序生成器

```
#include <cstdio>
#include <algorithm>
#include <queue>
#include <map>
#include <cstring>
#include <cmath>
#include <cstdlib>
#include <set>
#include <unordered_map>
#include <vector>
#include <ctime>
#define maxn 31
typedef long long ll;
using namespace std;
unsigned int grf[32];
int reg[] = {0, 1, 2, 3, 31};
int dm[1024];
#define R reg[rand() % 5]
#define I (rand() + rand())
#define B (rand() % 30)
void addu(int rs, int rt, int rd)
{
    printf("addu $%d,$%d,$%d\n", rd, rt, rs);
    if (rd)
        grf[rd] = grf[rs] + grf[rt];
}
void _and(int rs, int rt, int rd)
{
    printf("and $%d,$%d,$%d\n", rd, rt, rs);
    if (rd)
        grf[rd] = grf[rs] & grf[rt];
}
void subu(int rs, int rt, int rd)
{
    printf("subu $%d,$%d,$%d\n", rd, rt, rs);
    if (rd)
```

```
        grf[rd] = grf[rs] - grf[rt];
}
void sll(int rs, int rt, int rd)
{
    int s = rand()%31;
    printf("sll $%d,$%d,%d\n", rd, rt, s);
    if (rd)
        grf[rd] = grf[rt] << s;
}
void slt(int rs, int rt, int rd)
{
    printf("slt $%d,$%d,$%d\n", rd, rs, rt);
    if (rd)
        grf[rd] = (grf[rs] < grf[rt]);
}
void ori(int rs, int rt, int imm)
{
    printf("ori $%d,$%d,%d\n", rt, rs, imm);
    if (rt)
        grf[rt] = grf[rs] | imm;
}
void lui(int rs, int rt, int imm)
{
    printf("lui $%d,%d\n", rs, imm);
    if (rs)
        grf[rs] = 1u * imm << 16;
}
void addiu(int rs, int rt, int imm)
{
    printf("addiu $%d,$%d,%d\n", rs, rt, imm);
    if (rs)
        grf[rs] = grf[rt] + imm;
}
void lw(int rs, int rt)
{
    int imm = rand() % 31 * 4;
    printf("lw $%d,%d($0)\n", rt, imm);
    grf[rt] = dm[imm / 4];
}
void sw(int rs, int rt)
{
    int imm = rand() % 31 * 4;
    printf("sw $%d,%d($0)\n", rt, imm);
    dm[imm / 4] = grf[rt];
}
void lb(int rs, int rt)
{
    int imm = rand() % 127;
    printf("lb $%d,%d($0)\n", rt, imm);
    int byte = imm%4;
    int mask = 0;
    for(int i=8*byte;i<=7+8*byte;i++) mask |= (1<<i);
    grf[rt] = dm[imm / 4] & mask;
}
void sb(int rs, int rt)
{
    int imm = rand() % 127;
    printf("sb $%d,%d($0)\n", rt, imm);
```

```
}

int jump[1010];
void beq(int rs, int rt, int k)
{
    int jaddr = k + rand()%5+1;
    while (jump[jaddr]||jaddr>=maxn)
        jaddr = k + rand()%5+1;
    printf("beq $%d,$%d,label%d\n", rs, rt, jaddr);
}
void j(int k)
{
    int jaddr = k + rand()%5+1;
    while (jump[jaddr]||jaddr>=maxn)
        jaddr = k + rand()%5+1;
    printf("j label%d\n", jaddr);
}
void jal(int k)
{
    int jaddr = k + rand()%5+1;
    while (jump[jaddr]||jaddr>=maxn)
        jaddr = k + rand()%5+1;
    printf("jal label%d\n", jaddr);
}
int jr(int rs, int rt, int k)
{
    int i;
    vector<int> can;
    can.clear();
    for (i = 0; i < 10; i++)
        if (reg[i] > (0x3000+(k<<2)) && reg[i] < 0x3000+((k+7)<<2) && reg[i] <
0x3000+(maxn<<2))
            can.push_back(reg[i]);
    if (can.size() == 0)
    {
        beq(rs, rt, k);
        return 0;
    }
    rs = can[rand() % can.size()];
    printf("jr $%d\n", rs);
    return 1;
}
void nop()
{
    printf("nop\n");
}
int main()
{
    int i;
    srand(time(NULL));
    freopen("test.asm", "w", stdout);
    printf("subu $31,$31,$31\n"); //���$sp
    int last = -1;
    for (i = 0; i < maxn; i++)
    {
        printf("label%d: ", i);
        int instr = rand() % 15;
        while ((i < 9 || last == 1) && instr >= 6 && instr <= 9)
```

```
        { //j+j
            instr = rand() % 15;
        }
        int rs = R, rt = R, rd = R, imm = I;
        if (instr == 0)
            addu(rs, rt, rd);
        else if (instr == 1)
            subu(rs, rt, rd);
        else if (instr == 2)
            ori(rs, rt, imm);
        else if (instr == 3)
            lui(rs, 0, imm);
        else if (instr == 4)
            lw(rs, rt);
        else if (instr == 5)
            sw(rs, rt);
        else if (instr == 6)
            beq(rs, rt, i);
        else if (instr == 7)
            j(i);
        else if (instr == 8)
            jal(i);
        else if (instr == 14)
            _and(rs, rt, rd);
        else if (instr == 10)
            sll(rs, rt, rd);
        else if (instr == 11)
            slt(rs, rt, rd);
        else if (instr == 12)
            lb(rs, rt);
        else if (instr == 13)
            sb(rs, rt);
        else if (instr == 9)
        {
            int yes = jr(rs, rt, i);
            if (!yes)
                instr = 6; //beq
        }
        else
            nop();
        jump[i] = last = (instr >= 6 && instr <= 9);
    }
    //printf("label:\n beq $0,$0,label");
    return 0;
}
```

2.2 自动评测程序

```
import os
import re
import random


machine=[]
```

```
hex_to_bi={"0":"0000","1":"0001","2":"0010","3":"0011",
           "4":"0100","5":"0101","6":"0110","7":"0111",
           "8":"1000","9":"1001","a":"1010","b":"1011",
           "c":"1100","d":"1101","e":"1110","f":"1111"}
reg={"0":"$0",  "1":"$at", "2":"$v0", "3":"$v1",
     "4":"$a0", "5":"$a1", "6":"$a2", "7":"$a3",
     "8":"$t0", "9":"$t1", "10":"$t2", "11":"$t3",
     "12":"$t4", "13":"$t5", "14":"$t6", "15":"$t7",
     "16":"$s0", "17":"$s1", "18":"$s2", "19":"$s3",
     "20":"$s4", "21":"$s5", "22":"$s6", "23":"$s7",
     "24":"$t8", "25":"$t9", "26":"$k0", "27":"$k1",
     "28":"$gp", "29":"$sp", "30":"$fp", "31":"$ra"}
def bi_to_hex(a):
    bcode = ""
    for char in a:
        if not(char==" "):
            bcode += char
    return hex(int(str(int(bcode,2))))


def dasm(hexcode):
    out=["" for i in range(200)]
    labelcount=1
    label={}
    mipscount=0
    bicode=""
    for char in hexcode:
        if not(char==" "):
            bicode += char

    op=bicode[0:6]
    func=bicode[26:32]
    rs=reg[str(int(bicode[6:11],2))]
    rt=reg[str(int(bicode[11:16],2))]
    rd=reg[str(int(bicode[16:21],2))]
    shamt=bicode[21:26]
    imm=bi_to_hex(bicode[16:32])
    mips=""

    if op=='000000':
        itype="R"
    elif op=='000010' or op=='000011':
        itype="J"
    else:
        itype="I"

    if itype=="J":
        if op=='000010':
            mips="j "
        elif op=='000011':
            mips="jal "
        mips += imm

    elif itype=="R":
        if bicode=='00000000000000000000000000000000':
            mips="nop"
        elif func=='100000':
            mips="add "+rd+", "+rs+", "+rt
```

```
        elif func=='100001':
            mips="addu "+rd+", "+rs+", "+rt
        elif func=='100100':
            mips="and "+rd+", "+rs+", "+rt
        elif func=='001101':
            mips="break"
        elif func=='011010':
            mips="div "+rs+", "+rt
        elif func=='011011':
            mips="divu "+rs+", "+rt
        elif func=='001001':
            mips="jalr "+rd+", "+rs
        elif func=='001000':
            mips="jr "+rs
        elif func=='010000':
            mips="mfhi "+rd
        elif func=='010010':
            mips="mflo "+rd
        elif func=='010001':
            mips="mthi "+rd
        elif func=='010011':
            mips="mtlo "+rd
        elif func=='011000':
            mips="mult "+rs+", "+rt
        elif func=='011001':
            mips="multu "+rs+", "+rt
        elif func=='100111':
            mips="nor "+rd+", "+rs+", "+rt
        elif func=='100101':
            mips="or "+rd+", "+rs+", "+rt
        elif func=='000000':
            mips="sll "+rd+", "+rt+", "+shamt
        elif func=='000100':
            mips="sllv "+rd+", "+rt+", "+rs
        elif func=='101010':
            mips="slt "+rd+", "+rs+", "+rt
        elif func=='101011':
            mips="sltu "+rd+", "+rs+", "+rt
        elif func=='000011':
            mips="sra "+rd+", "+rt+", "+shamt
        elif func=='000111':
            mips="srav "+rd+", "+rt+", "+rs
        elif func=='000010':
            mips="srl "+rd+", "+rt+", "+shamt
        elif func=='000110':
            mips="srlv "+rd+", "+rt+", "+rs
        elif func=='100010':
            mips="sub "+rd+", "+rs+", "+rt
        elif func=='100011':
            mips="subu "+rd+", "+rs+", "+rt
        elif func=='001100':
            mips="syscall"
        elif func=='100110':
            mips="xor "+rd+", "+rs+", "+rt

    elif itype=="I":
        if op=='001000':
            mips="addi "+rt+", "+rs+", "+imm
```

```
        elif op=='001001':
            mips="addiu "+rt+", "+rs+", "+imm
        elif op=='001100':
            mips="andi "+rt+", "+rs+", "+imm
        elif op=='000100':
            mips="beq "+rs+", "+rt+", "+imm
        elif op=='000001' and bicode[11:16]=='00001':
            mips="bgez "+rs+", "+imm
        elif op=='000111':
            mips="bgtz "+rs+", "+imm
        elif op=='000110':
            mips="blez "+rs+", "+imm
        elif op=='000001' and bicode[11:16]=='00000':
            mips="bltz "+rs+", "+imm
        elif op=='000101':
            mips="bne "+rs+", "+rt+", "+imm
        elif op=='010000' and func=='011000':
            mips="eret"
        elif op=='100000':
            mips="lb "+rt+", "+imm+"("+rs+")"
        elif op=='100100':
            mips="lbu "+rt+", "+imm+"("+rs+")"
        elif op=='100001':
            mips="lh "+rt+", "+imm+"("+rs+")"
        elif op=='100101':
            mips="lhu "+rt+", "+imm+"("+rs+")"
        elif op=='001111':
            mips="lui "+rt+", "+imm
        elif op=='100011':
            mips="lw "+rt+", "+imm+"("+rs+")"
        elif op=='010000' and bicode[6:11]=='00000':
            mips="mfc0 "+rt+", "+rd
        elif op=='010000' and bicode[6:11]=='00100':
            mips="mtc0 "+rt+", "+rd
        elif op=='001101':
            mips="ori "+rt+", "+rs+", "+imm
        elif op=='101000':
            mips="sb "+rt+", "+imm+"("+rs+")"
        elif op=='101001':
            mips="sh "+rt+", "+imm+"("+rs+")"
        elif op=='001010':
            mips="slti "+rt+", "+rs+", "+imm
        elif op=='001011':
            mips="sltiu "+rt+", "+rs+", "+imm
        elif op=='101011':
            mips="sw "+rt+", "+imm+"("+rs+")"
        elif op=='001110':
            mips="xori "+rt+", "+rs+", "+imm
    out[mipscount] += mips
    mipscount += 1
    return out[0]


asmfilename="test.asm"
xlinx="D:\\PROGRAM\\14.7\\ISE_DS\\ISE"
time="10us"
os.environ['XILINX']=xlinx
path=os.path.dirname(os.path.realpath(__file__))
```

```
os.chdir(path)
filelist=os.walk(path)
with open("mips.prj","w") as prj:
    for folder in filelist:
        for file in folder[2]:
            if(len(file.split("."))>1 and file.split(".")[1]=="v"):
                prj.write("verilog work \""+folder[0]+"\\"+file+"\"\n")
with open("mips.tcl","w") as tcl:
    tcl.write("run "+time+";\nexit;\n")

print("start running")
    #"java -jar Mars.jar test.asm nc mc CompactTextAtZero a dump .text HexText
"+rom_name
    # problem: can not exit mars
os.system("java -jar Mars.jar  a nc mc CompactDataAtZero dump .text HexText
data0.txt 1000000 "+asmfilename)
os.system("java -jar Mars.jar  nc mc CompactDataAtZero dump .text HexText
data0.txt >out_std.txt 1000000 "+asmfilename)
print("std done")
os.system(xlinx+"\\bin\\nt64\\fuse "+"--nodebug  --prj mips.prj -o mips.exe
mipsAutoTest >log.txt")
os.system("mips.exe -nolog -tclbatch mips.tcl >out_source.txt")
print("source done")

process=0
with open("out_source.txt","r") as my:
    lines=my.readlines()
    if(len(lines)==0):
        print("fail to simulate")
        os._exit(1)
    if(lines[0][0]=='I'):
        process=1
n=0
while(1):
    if(lines[n][0]=="@"):
        break
    else:
        n=n+1
if(process):
    with open("out_source.txt","w") as my:
        my.writelines(lines[n:])
i=0
biao=0
instr = open("Instr.txt","r")
with open("out_source.txt","r") as source:
    with open("out_std.txt","r") as std:
        while(1):
            i+=1
            l1=source.readline().strip()
            l2=std.readline().strip()
            Instr = instr.readline().strip()
            asm = dasm(Instr)
            if((l1== "" or l1==None) and (l2=="" or l2==None)):
                break
            elif l1==l2:
                print("AC at line:%d "%(i)+"    source::"+l1+"  asm::"+asm)
            elif l1!=l2 and not "$ 0"in l2 and not "$ 0" in  l1:
                biao=1
```

```
                print("WA at line:%d "%(i)+"    source::"+l1+"  std::"+l2+"
asm::"+asm)

if biao==0:
    print("Accept on the point ")
else:
    os._exit(1)
```

2.3 测试数据及结果

手动构造数据

```
subu $31,$31,$31
label0: addiu $1,$0, 1
label1: addiu $2,$0, 2
label2: addiu $3,$0, 3
label3: addiu $4,$0, 4
label4: addiu $5,$0, 5
label6: lw $2,0($0)
label7: lw $4,4($0)
label8: lw $5,8($0)
label10: lw $1,124($0)
label11: lui $31, 32768
label12: sw $10,8($0)
label13: sw $10,40($0)
label14: sw $10,12($0)
label15: subu $11,$3,$2
label16: addu $7, $6, $t0
label17: beq $6, $7, label20
label18: nop
label19: lui $0,32063
label20: lui $0,36507
label21: ori $20,$3,45891
label22: jal label24
label23: lui $2,4496
label24: sw $31,64($0)
label25: sb $6,65($2)
label26: lw $22,64($0)
label27: slt $23, $2, $3
label28: addu $28, $0, 69
label29: lb $24,-2($28)
label30: jr $ra
```

```
@00003000: $31 <= 00000000
@00003004: $ 1 <= 00000001
@00003008: $ 2 <= 00000002
@0000300c: $ 3 <= 00000003
@00003010: $ 4 <= 00000004
@00003014: $ 5 <= 00000005
@00003018: $ 6 <= 00000006
@0000301c: $ 4 <= 00000000
@00003020: $ 6 <= 00000000
@00003024: $ 1 <= 00000000
@00003028: $31 <= 80000000
@0000302c: *00000008 <= 00000000
@00003030: *00000028 <= 00000000
```

```
@00003034: *0000000c <= 00000000
@00003038: $11 <= 00000001
@0000303c: $ 7 <= 00000000
@00003050: $20 <= 0000b343
@00003054: $31 <= 00003058
@0000305c: *00000040 <= 00003058
@00003060: *00000043 <= 00000000
@00003064: $22 <= 00003058
@00003068: $23 <= 00000001
@0000306c: $ 1 <= 00000000
@00003070: $ 1 <= 00000045
@00003074: $28 <= 00000045
@00003078: $24 <= 00000000
@00003058: $ 2 <= 11900000
@0000305c: *00000040 <= 00003058
```

自动生成数据(节选)

```
test point 0 begin:::
generating...
test.asm done
start running
source done
std done
AC at line:1            ams::subu $ra, $ra, $ra
AC at line:2            ams::subu $at, $ra, $v0
AC at line:3            ams::ori $0, $at, 0xd14a
AC at line:4            ams::addu $v0, $v1, $v0
AC at line:5            ams::nop
AC at line:6            ams::lw $0, 0xc($0)
AC at line:7            ams::lw $ra, 0x6c($0)
AC at line:8            ams::nop
AC at line:9            ams::sw $at, 0x58($0)
AC at line:10           ams::ori $0, $v1, 0x7d65
AC at line:11           ams::ori $at, $ra, 0xc746
AC at line:13           ams::lw $ra, 0x44($0)
AC at line:15           ams::subu $at, $ra, $0
AC at line:16           ams::sw $ra, 0x8($0)
AC at line:19           ams::lui $v1, 0x4c9a
AC at line:20           ams::subu $ra, $ra, $ra
AC at line:21           ams::subu $at, $ra, $v0
AC at line:22           ams::ori $0, $at, 0xd14a
AC at line:23           ams::addu $v0, $v1, $v0
AC at line:24           ams::nop
AC at line:25           ams::lw $0, 0xc($0)
AC at line:26           ams::lw $ra, 0x6c($0)
AC at line:27           ams::nop
AC at line:28           ams::sw $at, 0x58($0)
AC at line:29           ams::ori $0, $v1, 0x7d65
AC at line:30           ams::ori $at, $ra, 0xc746
AC at line:32           ams::lw $ra, 0x44($0)
Traceback (most recent call last):
  File "C:\Users\86135\Desktop\test_demo\test_p3\test_old\run.py", line 248, in
<module>
    asm = dasm(l1[0:39])
  File "C:\Users\86135\Desktop\test_demo\test_p3\test_old\run.py", line 38, in
dasm
```

```
    rs=reg[str(int(bicode[6:11],2))]
ValueError: invalid literal for int() with base 2: ''
test point 1 begin:::
generating...
test.asm done
start running
source done
std done
AC at line:1            ams::subu $ra, $ra, $ra
AC at line:2            ams::lui $v0, 0xbed7
AC at line:3            ams::addu $v1, $0, $v0
AC at line:4            ams::addu $0, $v0, $0
AC at line:5            ams::subu $v1, $v1, $v1
AC at line:6            ams::lw $ra, 0x60($0)
AC at line:7            ams::addu $v0, $ra, $v0
AC at line:8            ams::lw $at, 0x68($0)
AC at line:9            ams::ori $v1, $v1, 0x55dc
AC at line:10           ams::ori $0, $v0, 0x436f
AC at line:11           ams::lui $ra, 0xab5c
AC at line:13           ams::ori $v0, $v1, 0x9bab
AC at line:14           ams::sw $0, 0x14($0)
AC at line:15           ams::nop
AC at line:16           ams::sw $0, 0x70($0)
AC at line:19           ams::sw $ra, 0x5c($0)
AC at line:22           ams::lui $v1, 0xda6b
AC at line:24           ams::subu $0, $v1, $at
AC at line:25           ams::sw $0, 0x40($0)
AC at line:26           ams::subu $0, $v0, $at
AC at line:27           ams::subu $v1, $ra, $ra
AC at line:28           ams::subu $ra, $ra, $ra
AC at line:29           ams::lui $v0, 0xbed7
AC at line:30           ams::addu $v1, $0, $v0
AC at line:31           ams::addu $0, $v0, $0
AC at line:32           ams::subu $v1, $v1, $v1
```

3 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。
    - ROM是只读存储器，适合存储固定不变的数据，而指令数据在单周期cpu运行过程有限个周期中不 变的，因此选用ROM是合理的。RAM是读写存储器，DM要求存储器可以读和写，且速度要求不高，因 此选取速度不高但是功能齐全的RAM比较合理。GRF需要经常对数据进行读写且速度要求比较高，因此 选取Register比较合理。
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？请给出你的理由。
    - nop指令数据是0x00000000，除了PC=PC+4之外，没有进行任何其他操作，因此没有对电路中的 逻辑真值运算产生任何影响，存在与否对电路无影响。
3. 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。
    - 假设DM存储大小是256MB，且数据地址从0x10000000-1fffffff，则将要存储的地址的最高四位和 0x1进行比较，得到一个片选信号，相同则将这个数据存储到DM中，否则将数据存储到其他相应的位 置。

4. 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。

    - 所谓形式验证，是指从数学上完备地证明或验证电路的实现方案是否确实实现了电路设计所描述的 功能。形式验证方法分为等价性验证、模型检验和定理证明等。
      对组合逻辑来说，不存在状态寄存器，其输出值仅仅依赖于当前的输入值。这时只要对每个输入值组合 证明其输出的数据组合相同即可。
      对时序逻辑而言，可以把它看成一个有限状态机。电路功能的等价可以用有限状态机的等价来判断。假 定有两个状态机A和B，对他们的验证可以转化为以下的方法，当A和B有相同的接口，而且从相同的初始 状态出发，两者对有效输入值序列产生相同的输出值序列，则可以说A和B等价。
    - 形式验证的优点是：(1)形式验证是对指定描述的所有可能的情况进行验证，覆盖率达到了100%。(2)形 式验证技术是借用数学上的方法将待验证电路和功能描述或参考设计直接进行比较，不需要开发测试激 励。(3)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。
    - 缺点是：验证方法复杂抽象，难以准确把握。而且形式验证是数学逻辑分析，而不是电路分析，不能有 效的验证电路的性能，如电路的时延和功耗等。