2021年北航计组P4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IFU, RF, IM, NPC, ALU, DM, EXT, 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	同步复位	当同步复位信号有效时，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 IFU

端口	输入/输出	位宽	描述
NPC	I	32	下一个PC
clk	I	1	时钟信号
reset	I	1	异步复位信号
PC	O	32	当前PC地址
Instr	O	32	当前PC对应指令

序号	功能名称	功能描述
1	同步复位	当复位信号有效时，将PC值设置为 0x00003000
2	取指令	根据当前PC值从IM中取出指令，并输出

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 DM

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。
DM_op	I	3	控制信号
clk	I	1	clock
reset	I	1	同步复位
DM_OUT	O	32	读取的数据

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]`
3	读字	`DMout <= mem[addr];`
4	读字节	`DMout <= signed_b(mem[addr][7+8*A[1:0] -:8]);`

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 1010
typedef long long ll;
using namespace std;
unsigned int grf[32];
int reg[] = {0, 1, 2, 3, 4, 5, 30, 31, 29};
int dm[1024];
#define R reg[rand() % 8]
#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)
{
    imm = rand()%65535;
    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()%50+1;
    while (jump[jaddr]||jaddr>=maxn)
        jaddr = k + rand()%50+1;
    printf("beq $%d,$%d,label%d\n", rs, rt, jaddr);
}
void j(int k)
{
    int jaddr = k + rand()%50+1;
    while (jump[jaddr]||jaddr>=maxn)
        jaddr = k + rand()%50+1;
    printf("j label%d\n", jaddr);
}
void jal(int k)
{
    int jaddr = k + rand()%50+1;
    while (jump[jaddr]||jaddr>=maxn)
        jaddr = k + rand()%50+1;
    printf("jal label%d\n", jaddr);
}
void jalr(int rd, int rs)
{
    printf("jalr $%d, $%d\n", rd, rs);
}
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+50)<<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() % 16;
        while ((i < 90 || last == 1) && instr >= 6 && instr <= 9)
        { //j+j
            instr = rand() % 16;
        }
        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 == 15)
        //  jalr(rd, rs);
        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
mips_tb >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)
            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)
                '''
                if l2=="" or l2 == None:
                    print("Wrong answer occur in line %d of code: "%(i)+"we got
"+l1+" when we expected Nothing")
                else:
                    print("Wrong answer occur in line %d of code: "%(i)+"we got
"+l1+" when we expected "+l2)
                '''
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)

```
subu $31,$31,$31
label0: lw $0,12($0)
label1: sb $4,1($0)
label2: sb $5,82($0)
label3: and $3,$3,$30
label4: lui $31,50940
label5: slt $30,$1,$5
label6: lb $1,71($0)
label7: addu $31,$0,$4
label8: lw $0,52($0)
label9: lw $2,56($0)
label10: lw $31,68($0)
label11: sw $0,24($0)
```

```
label12: lw $4,48($0)
label13: lw $31,16($0)
label14: lb $0,33($0)
label15: and $2,$5,$3
label16: slt $30,$1,$5
label17: lw $5,4($0)
label18: addu $30,$4,$5
label19: and $31,$30,$30
label20: lb $2,120($0)
label21: sll $1,$31,15
label22: nop
label23: ori $31,$2,28285
label24: sw $0,0($0)
label25: lui $4,21443
label26: subu $3,$4,$31
label27: sll $2,$0,8
label28: and $4,$4,$0
label29: sb $31,10($0)
label30: lb $31,79($0)
label31: sb $0,49($0)
label32: sb $5,123($0)
label33: lw $5,60($0)
label34: ori $5,$0,16741
label35: addu $4,$31,$2
label36: sll $4,$31,10
label37: sw $31,96($0)
label38: nop
label39: sb $1,24($0)
label40: lw $31,104($0)
label41: sw $1,32($0)
label42: addu $30,$0,$31
label43: subu $2,$3,$4
label44: lw $2,20($0)
label45: sw $31,48($0)
label46: sb $3,3($0)
label47: lui $3,33341
label48: nop
label49: lui $1,36188
label50: and $0,$31,$5
label51: sb $30,87($0)
label52: addu $0,$30,$31
label53: subu $1,$5,$4
label54: slt $0,$30,$30
label55: sll $5,$1,9
label56: and $0,$3,$0
label57: lw $5,116($0)
label58: and $1,$2,$30
label59: nop
label60: lui $2,54044
label61: slt $1,$31,$1
label62: lui $30,31679
label63: slt $5,$0,$3
label64: slt $2,$5,$2
label65: slt $4,$4,$30
label66: addu $3,$0,$2
label67: sw $30,72($0)
label68: lw $3,92($0)
label69: sb $30,42($0)
```

```
label70: subu $31,$5,$31
label71: lui $1,33153
label72: addu $30,$5,$3
label73: sw $2,76($0)
label74: lui $3,19816
label75: addu $5,$30,$3
label76: sw $30,84($0)
label77: ori $5,$2,16786
label78: addu $3,$0,$1
label79: slt $1,$30,$31
label80: lb $31,40($0)
label81: slt $2,$0,$2
label82: sll $3,$4,8
label83: slt $2,$5,$5
label84: addu $0,$2,$4
label85: addu $4,$5,$4
label86: lw $0,8($0)
label87: nop
label88: lui $3,9217
label89: ori $31,$4,48523
label90: jal label116
label91: sw $2,32($0)
label92: and $4,$4,$2
label93: addu $5,$3,$4
label94: ori $0,$2,10897
label95: sw $31,36($0)
label96: beq $30,$30,label98
label97: addu $5,$2,$2
label98: sll $1,$30,17
label99: and $0,$1,$2
label100: subu $30,$2,$31
label101: nop
label102: sw $31,48($0)
label103: beq $5,$31,label109
label104: ori $2,$2,43182
label105: sw $3,76($0)
label106: lb $5,25($0)
label107: subu $2,$31,$5
label108: slt $5,$30,$31
label109: and $5,$5,$2
label110: sw $31,100($0)
label111: addu $30,$3,$31
label112: sb $3,84($0)
label113: ori $1,$2,14981
label114: slt $3,$1,$4
label115: beq $2,$5,label164
label116: ori $0,$3,10078
label117: sb $1,93($0)
label118: subu $3,$3,$4
label119: nop
label120: jal label151
label121: slt $2,$5,$3
label122: subu $1,$31,$31
label123: jal label170
label124: sw $1,28($0)
label125: ori $4,$31,22347
label126: subu $2,$30,$4
label127: subu $1,$30,$31
```

```
label128: beq $3,$0,label148
label129: ori $31,$31,42068
label130: slt $2,$30,$4
label131: and $30,$1,$5
label132: subu $4,$4,$4
label133: sw $5,104($0)
label134: subu $1,$2,$0
label135: sb $5,71($0)
label136: lb $4,29($0)
label137: lui $30,48512
label138: subu $2,$2,$0
label139: j label163
label140: lb $5,1($0)
label141: jal label191
label142: sb $1,95($0)
label143: jal label158
label144: lw $31,60($0)
label145: ori $4,$2,6859
label146: nop
label147: nop
label148: lui $5,57412
label149: and $30,$31,$30
label150: jal label153
label151: sll $2,$1,0
label152: nop
label153: ori $1,$31,33784
label154: slt $3,$1,$5
label155: nop
label156: lui $4,10869
label157: lui $3,46482
label158: lb $5,7($0)
label159: lui $2,22265
label160: beq $2,$4,label192
label161: slt $0,$4,$5
label162: addu $5,$5,$2
label163: ori $1,$5,45108
label164: sll $5,$5,21
label165: lb $2,68($0)
label166: beq $2,$0,label185
label167: lb $1,59($0)
label168: j label185
label169: lw $5,108($0)
label170: lb $3,30($0)
label171: nop
label172: addu $30,$3,$0
label173: sb $2,88($0)
label174: beq $30,$0,label206
label175: lui $30,42464
label176: slt $30,$2,$30
label177: beq $3,$4,label224
label178: nop
label179: slt $4,$3,$1
label180: nop
label181: sll $31,$1,7
label182: sll $4,$30,18
label183: lui $4,41133
label184: subu $2,$30,$4
label185: j label188
```

```
label186: lui $31,55006
label187: jal label218
label188: sb $3,53($0)
label189: jal label235
label190: and $5,$31,$5
label191: slt $0,$3,$3
label192: sb $30,119($0)
label193: sw $4,4($0)
label194: beq $0,$2,label224
label195: slt $1,$4,$3
label196: sll $5,$0,21
label197: lb $4,81($0)
label198: sll $30,$3,0
label199: and $0,$0,$31
label200: sb $0,52($0)
label201: beq $2,$30,label231
label202: sw $3,80($0)
label203: subu $3,$1,$31
label204: sw $31,76($0)
label205: sw $1,64($0)
label206: addu $31,$2,$5
label207: addu $1,$31,$5
label208: lui $3,6798
label209: sb $4,38($0)
label210: subu $31,$2,$5
label211: addu $5,$4,$30
label212: slt $3,$4,$0
label213: addu $30,$3,$1
label214: nop
label215: ori $4,$3,31888
label216: sb $0,38($0)
label217: beq $1,$3,label263
label218: subu $4,$30,$3
label219: and $1,$5,$1
label220: lb $30,104($0)
label221: addu $5,$2,$31
label222: sw $4,92($0)
label223: slt $30,$2,$30
label224: ori $31,$5,59111
label225: slt $31,$1,$30
label226: and $0,$4,$3
label227: lw $3,52($0)
label228: j label241
label229: lui $3,49052
label230: beq $5,$5,label233
label231: lui $1,35736
label232: sw $2,120($0)
label233: jal label258
label234: ori $31,$31,7088
label235: j label257
label236: lb $31,45($0)
label237: beq $2,$5,label273
label238: addu $3,$3,$2
label239: lw $1,0($0)
label240: lui $2,44168
label241: beq $31,$0,label251
label242: subu $3,$0,$0
label243: ori $2,$0,44962
```

```
label244: nop
label245: lb $5,0($0)
label246: addu $4,$30,$30
label247: sb $1,61($0)
label248: and $2,$30,$0
label249: jal label282
label250: ori $3,$1,33898
label251: jal label283
label252: lui $3,45278
label253: addu $30,$0,$31
label254: ori $30,$5,7157
label255: ori $2,$4,32300
label256: beq $3,$0,label270
label257: lw $0,16($0)
label258: sw $31,112($0)
label259: lb $3,82($0)
label260: j label286
label261: sw $3,56($0)
label262: beq $1,$5,label297
label263: sw $3,60($0)
label264: sll $30,$1,23
label265: and $0,$1,$2
label266: jal label282
label267: ori $30,$2,58136
label268: beq $5,$5,label300
label269: lw $0,40($0)
label270: lui $2,37604
label271: addu $31,$4,$3
label272: and $3,$2,$5
label273: addu $5,$0,$2
label274: sll $2,$31,14
label275: nop
label276: addu $0,$4,$3
label277: lb $3,65($0)
label278: lui $30,17196
label279: slt $4,$3,$5
label280: addu $0,$0,$5
label281: slt $2,$2,$2
label282: ori $3,$2,39430
label283: addu $0,$0,$3
label284: sll $3,$5,28
label285: sll $2,$4,2
label286: subu $3,$5,$5
label287: ori $3,$5,31211
label288: lui $1,18630
label289: ori $31,$30,36322
label290: addu $5,$31,$3
label291: slt $1,$5,$31
label292: nop
label293: jal label320
label294: and $3,$0,$2
label295: addu $2,$4,$1
label296: addu $5,$4,$31
label297: addu $5,$1,$30
label298: sb $0,46($0)
label299: sll $2,$3,5
label300: and $30,$4,$1
label301: addu $2,$5,$3
```

```
label302: sb $31,66($0)
label303: slt $31,$1,$2
label304: sll $3,$3,27
label305: lw $3,120($0)
label306: sw $31,44($0)
label307: j label349
label308: nop
label309: lui $4,46891
label310: beq $2,$30,label355
label311: addu $2,$0,$3
label312: subu $30,$2,$1
label313: lb $3,85($0)
label314: lw $4,52($0)
label315: jal label361
label316: nop
label317: jal label338
label318: lw $30,72($0)
label319: sll $5,$30,15
label320: ori $5,$2,35032
label321: j label344
label322: slt $30,$31,$2
label323: ori $5,$31,47740
label324: beq $4,$1,label337
label325: addu $4,$31,$1
label326: sb $5,28($0)
label327: jal label353
label328: nop
label329: beq $1,$1,label334
label330: sll $31,$2,11
label331: jal label363
label332: nop
label333: jal label351
label334: sll $30,$4,11
label335: lw $5,108($0)
label336: and $30,$0,$1
label337: sll $31,$1,2
label338: addu $3,$5,$1
label339: sb $2,32($0)
label340: sll $31,$0,11
label341: sll $31,$0,0
label342: lui $0,29255
label343: sw $5,48($0)
label344: sw $5,24($0)
label345: slt $4,$4,$4
label346: sb $2,111($0)
label347: beq $0,$1,label364
label348: sb $1,61($0)
label349: sw $4,4($0)
label350: subu $0,$30,$4
label351: sw $5,112($0)
label352: lw $2,108($0)
label353: addu $3,$31,$2
label354: lb $30,66($0)
label355: j label377
label356: lw $3,84($0)
label357: subu $30,$30,$5
label358: and $30,$30,$31
label359: sb $1,40($0)
```

```
label360: slt $3,$31,$5
label361: nop
label362: and $5,$5,$2
label363: lw $30,48($0)
label364: lui $30,30454
label365: j label405
label366: sb $5,37($0)
label367: lui $2,32833
label368: jal label370
label369: lui $4,34700
label370: sw $31,48($0)
label371: and $31,$0,$4
label372: sb $2,68($0)
label373: nop
label374: lui $31,23199
label375: sb $3,11($0)
label376: addu $0,$5,$2
label377: and $1,$0,$30
label378: sw $1,64($0)
label379: beq $3,$1,label403
label380: lui $3,19353
label381: beq $1,$2,label390
label382: lui $31,17992
label383: beq $31,$31,label413
label384: addu $31,$31,$4
label385: beq $0,$5,label406
label386: nop
label387: beq $1,$2,label426
label388: lb $2,19($0)
label389: lb $1,14($0)
label390: beq $5,$2,label415
label391: addu $0,$4,$5
label392: beq $5,$4,label427
label393: and $4,$3,$31
label394: lb $30,59($0)
label395: lb $31,76($0)
label396: lw $31,4($0)
label397: slt $5,$4,$2
label398: jal label441
label399: addu $1,$2,$4
label400: sw $31,96($0)
label401: subu $2,$5,$4
label402: sll $1,$30,3
label403: lw $31,64($0)
label404: nop
label405: and $4,$30,$1
label406: jal label440
label407: sll $5,$5,30
label408: j label432
label409: subu $2,$0,$2
label410: slt $5,$3,$5
label411: sb $0,110($0)
label412: nop
label413: sll $31,$2,9
label414: beq $1,$0,label427
label415: addu $2,$0,$30
label416: sw $1,44($0)
label417: sll $1,$2,25
```

```
label418: nop
label419: lw $5,92($0)
label420: addu $4,$2,$2
label421: nop
label422: sw $31,116($0)
label423: addu $4,$30,$30
label424: slt $30,$1,$5
label425: sll $0,$1,4
label426: beq $31,$0,label447
label427: sll $30,$5,9
label428: lw $2,68($0)
label429: lb $5,118($0)
label430: sll $0,$2,30
label431: lui $5,55194
label432: addu $30,$5,$4
label433: slt $4,$30,$2
label434: beq $3,$3,label471
label435: subu $4,$30,$30
label436: beq $4,$30,label473
label437: and $1,$31,$4
label438: beq $5,$2,label453
label439: and $30,$4,$31
label440: and $5,$4,$0
label441: beq $5,$1,label444
label442: sw $30,52($0)
label443: nop
```

- 执行结果与mars一致

3 思考题

1. 根据你的理解，在下面给出的DM的输入示例中，地址信号addr位数为什么是 [11:2]而不是[9:0]？这个addr信号又是从哪里来的？
    - lw，sw 的16位立即数以Byte为单位，而我们设计的 DM 模块是以 word 为单位的。我们通过 ALU 运算 出来的 MemAddress 是以Byte为单位的，所以要除以 4，也就是右移两位，才是真正的 MemAddress。因此取 ALU 输出的 32 位先右移两位，再取最低的10位，即取[11:2]才是所需要的真正 的 MemAddress。所以这个 address 信号来自于ALU 的输出 32 位中的[11:2]这10位。
2. 思考Verilog语言设计控制器的译码方式，给出代码示例，并尝试对比各方式的优 劣。
    - **本设计采取的方法** 先用宏定义指令的opcode，func，以及各个模块的控制信号，接着，使用如下所示译码-分类-产生控制信号的方法解决

```
`timescale 1ns / 1ps
`include "const.v"
`default_nettype wire

module CTR(
    input [31:0] Instr,
    output [3:0] ALU_op,
    output [2:0] NPC_op,
    output [2:0] EXT_op,
    output [2:0] DM_op,
    output [2:0] ALU_B_sel,
```

```
    output [2:0] RF_WD_sel,
    output [2:0] RF_A3_sel,
    output DM_wr,
    output RF_wr,
    output cali,
    output calr,
    output load,
    output store,
    output branch
    );

    wire [5:0] op = Instr[31:26];
    wire [5:0] func = Instr[5:0];

    wire addi   = (op==`op_addi);
    wire addiu  = (op==`op_addiu);
    wire beq    = (op==`op_beq);
    wire blez   = (op==`op_blez);
    wire j  = (op==`op_j);
    wire jal    = (op==`op_jal);
    wire lb = (op==`op_lb);
    wire lbu    = (op==`op_lbu);
    wire lh = (op==`op_lh);
    wire lhu    = (op==`op_lhu);
    wire lui    = (op==`op_lui);
    wire lw = (op==`op_lw);
    wire ori    = (op==`op_ori);
    wire sw = (op==`op_sw);
    wire sh = (op==`op_sh);
    wire sb = (op==`op_sb);

    wire add        = (op==`op_sp&&func==`func_add);
    wire addu   = (op==`op_sp&&func==`func_addu);
    wire _and   = (op==`op_sp&&func==`func_and);
    wire div        = (op==`op_sp&&func==`func_div);
    wire divu   = (op==`op_sp&&func==`func_divu);
    wire jalr   = (op==`op_sp&&func==`func_jalr);
    wire jr     = (op==`op_sp&&func==`func_jr);
    wire mfhi   = (op==`op_sp&&func==`func_mfhi);
    wire mflo   = (op==`op_sp&&func==`func_mflo);
    wire mult   = (op==`op_sp&&func==`func_mult);
    wire multu  = (op==`op_sp&&func==`func_multu);
    wire _nor   = (op==`op_sp&&func==`func_nor);
    wire _or        = (op==`op_sp&&func==`func_or);
    wire sll        = (op==`op_sp&&func==`func_sll);
    wire sllv   = (op==`op_sp&&func==`func_sllv);
    wire slt        = (op==`op_sp&&func==`func_slt);
    wire sltu   = (op==`op_sp&&func==`func_sltu);
    wire sra        = (op==`op_sp&&func==`func_sra);
    wire srav   = (op==`op_sp&&func==`func_srav);
    wire srl        = (op==`op_sp&&func==`func_srl);
    wire srlv   = (op==`op_sp&&func==`func_srlv);
    wire sub        = (op==`op_sp&&func==`func_sub);
    wire subu   = (op==`op_sp&&func==`func_subu);
    wire _xor   = (op==`op_sp&&func==`func_xor);

    assign load = lw|lb|lbu|lh|lhu|lui;
    assign store = sw|sb|sh;
```

```
    assign branch = beq|blez;
    assign cali = addiu|ori;
    assign calr = addu|_and|subu|slt|sll;

    assign ALU_op = (sub|subu)? `ALU_sub:
                    (sll)? `ALU_sll:
                    (beq)? `ALU_eq:
                    (slt)? `ALU_lt:
                    (blez)? `ALU_lez:
                    (ori)? `ALU_or:
                    (_and)? `ALU_and:
                    (srl)? `ALU_srl: `ALU_add;

    assign NPC_op = (branch)? `NPC_branch:
                    (j|jal)? `NPC_j:
                    (jr)? `NPC_jr: `NPC_default;

    assign EXT_op = (lw|sw|lb|sb|addiu)? `EXT_signed:
                    (lui)? `EXT_lui: `EXT_unsigned;

    assign DM_op = (lw|sw)? `DM_w:
                   (lh|sh)? `DM_h:
                   (lhu)? `DM_hu:
                   (lb|sb)? `DM_b:
                   (lbu)? `DM_bu: `DM_err;

    assign RF_A3_sel = (jal)? `RF_A3_31:
                       (lb|lw|lui|addiu|ori)? `RF_A3_rt:
`RF_A3_rd;
    assign RF_WD_sel = (jal)? `RF_WD_PC4:
                       (lb|lw)? `RF_WD_DM:
                       (lui)? `RF_WD_EXT: `RF_WD_ALU;
    assign ALU_B_sel = (lw|sw|lb|sb|addiu|ori)? `ALU_EXT: `ALU_RD2;

    assign DM_wr = store;
    assign RF_wr = (load|calr|cali|jal)? 1:0;
endmodule
```

- 其他方式：采用always语句建模，always块内代码冗长容易出错，不符合高内聚低耦合的设计思想；不使用宏定义，同样容易出错。

3. 在相应的部件中，reset的优先级比其他控制信号（不包括clk信号）都要高，且 相应的设计都是同步复位。清零信号reset所驱动的部件具有什么共同特点?
   - 三个需要重置的部件：PC,DM,GRF
     - PC 复位到 0x00003000 这一初值，即程序重新开始运行
     - DM 为内存空间，程序运行后可能向这里写入数据，不同程序执行时对内存空间写的值和位置可能 是不一样的，复位时需要清零重置
     - GRF 是寄存器文件，程序运行后可能向这里写入数据，不同程序执行时对寄存器文件写的值和寄对 应寄存器编号可能是不一样的，复位应该清零重置
     - 共同特点：不重置清零很可能会影响下一次程序的运行。
4. C语言是一种弱类型程序设计语言。C语言中不对计算结果溢出进行处理，这意味 着C语言要求程序员必须很清楚计算结果是否会导致溢出。因此，如果仅仅支持C语 言，MIPS指令的所有计算指令均可以忽略溢出。 请说明为什么在忽略溢出的前提 下，addi与addiu是等价的，add与addu是等价的。提示：阅读《MIPS32® Architecture For Programmers Volume II: The MIPS32® Instruction Set》 中相关指令的Operation部分 。

- add 指令把两个操作数的最高位当做第 33 位，实现的 33 位加法，但实际上低32位的结果只跟 GPR[rs]，GPR[rt]有关，即两者之和，如果有进位 1，则 temp32=1+GPR[rs]31+ GPR[rt]31+ in[31]（in 表示进位）,没有则 temp32=GPR[rs]31+ GPR[rt]31+in[31]，temp31 是只跟GPR[rs]，GPR[rt]有关的， 计算出 temp31,temp32 后可以用来判断是否溢出。
- 但如果忽略溢出，add 指令保留的 GPR[rd] ← temp 31..0 也就是 addu 所保留的 GPR[rd] ← GPR[rs] + GPR[rt]，即 rs,rt 两个寄存器的和，跟溢出无关，所以在忽略溢出的前提下 add 与 addu 是等价的。
- 同理，addi和addiu，低 32 位的加法只跟 GPR[rs] ，sign_extend(immediate)有关，所以如果忽略 溢出，addi 指令保留的 GPR[rd] ← temp 31..0 也就是 addiu 所保留的 GPR[rd]← GPR[rs] + sign_extend(immediate)，即 rs 寄存器和 sign_extend(immediate)的和，跟溢出无关，所以在忽略溢 出的前提下 addi 与 addiu 是等价的。

5. 根据自己的设计说明单周期处理器的优缺点。
   - 优点：设计简单，结构简单，都由统一时钟控制
   - 缺点:
     1. 所有指令都在一个周期内完成，然而不同类型的指令可能具有不同的指令周期，这就导致了单周期 处理器速度慢，吞吐量低；同步时钟的设计，时钟周期是一个常数，需要设置成足够长从而能够满 足所需时间最长的指令，而大部分指令执行时时间是较短的，比如R 型指令不需要访问存储器，比 lw 指令所需时间短，类似这样的问题带来了时间效率上的消耗。
     2. CPU设计中有三个加法器，一个用于 ALU，两个用于 PC 的逻辑（PC+4 和 beq 指令的跳转），而 加法器是占用芯片面积较多，真正搭建时可能成本过高。
     3. 采用独立的指令存储器 IM 和数据存储器 DM，在实际的CPU电路搭建过程中不太容易实现。