C语言程序MIPS编译

Homework9

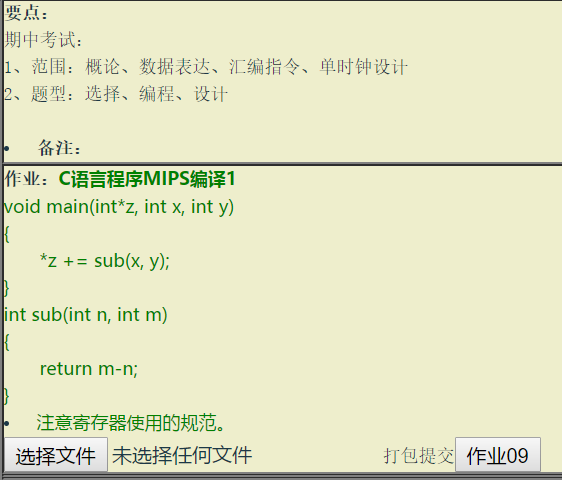
(MIPS in C)

姓名：蔡庆鹏

学号：3150102196

2017.4.29

1. 题目要求



要点：

wwwwwww

void main(int\*z, int x, int y)

{

　　\*z += sub(x, y);

}

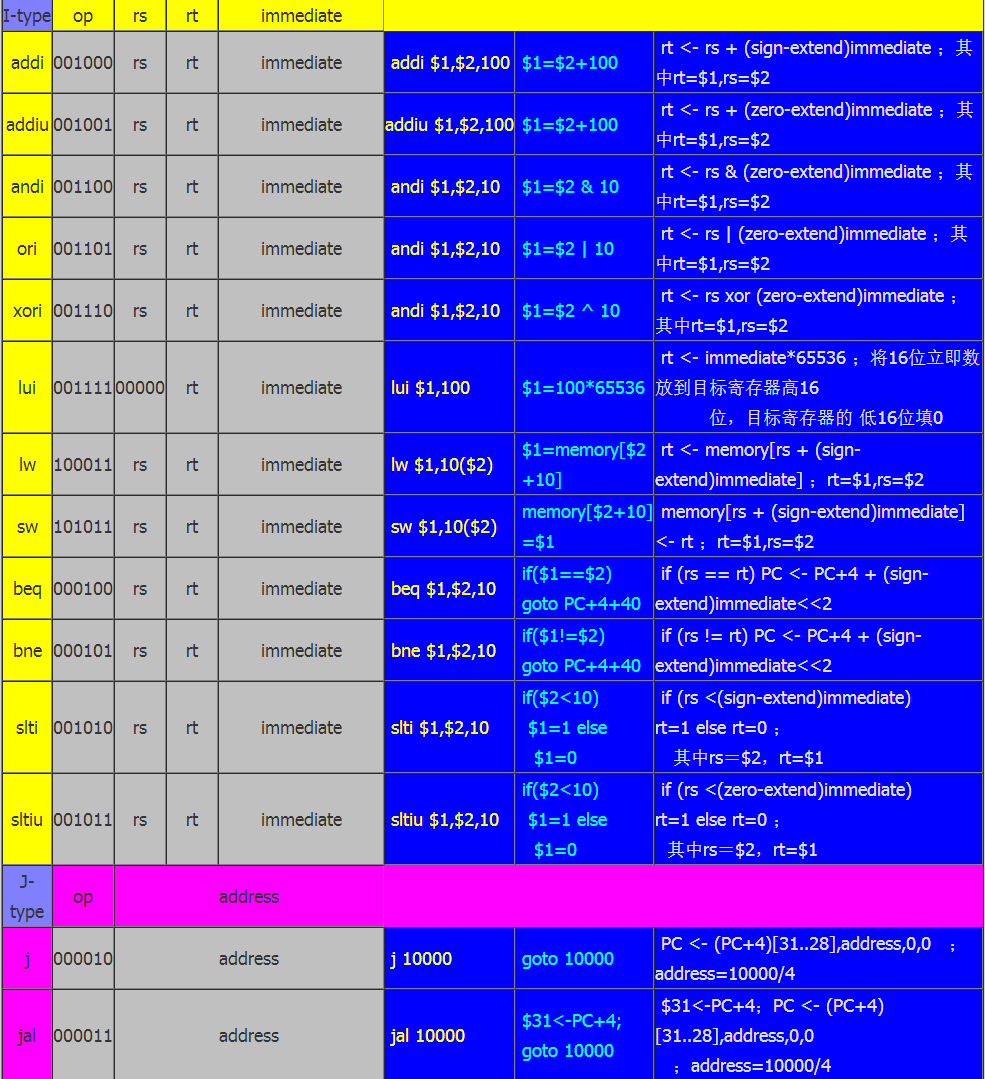
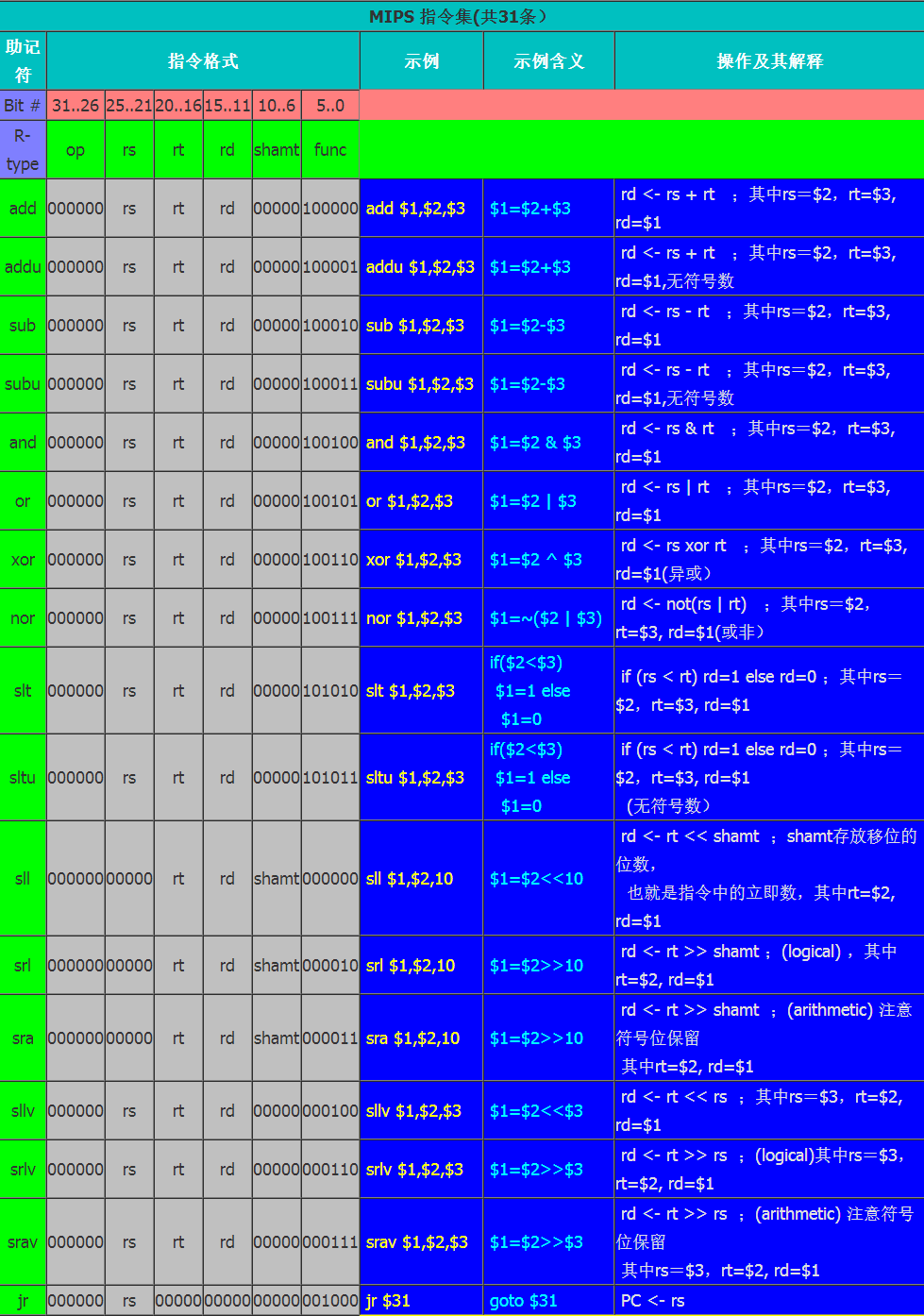
int sub(int n, int m)

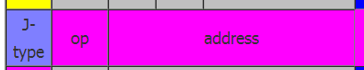
{

　　return m-n;

}

1. MIPS汇编



1. 算法思想
2. 机器码来说，机器码的分布主要是由3种格式。
   1. 
   2. 
   3. 

注意：常见的是这三种，但是对于第一种r-type和i-type类型，这两种分类内部还要继续分类。

1. 模拟器基于之前写的MIPS汇编和反汇编器，然后重新创建了一个模拟平台（命令行版）。
2. 首先是指令的真实操作，这个只要通过每条指令后面跟随的操作及其内容就可以很容易的去做相应操作，比如add操作，就是把两个寄存器的内容相加，通过开辟数组，模拟内存，就是一行语句的问题。
3. 然后是命令行的独特设计
   1. -->R-看寄存器

看寄存器只要把32个寄存器依次以 %s - %d 的形式输出即可

* 1. -->D-数据方式看内存

所谓数据方式看内存，也就是通过1010的机器码形式去看内存的真实数据，因为这是汇编模拟器，所以只需要把汇编语言转化成机器码即可显示内存数据。

* 1. -->U-指令方式看内存，

指令方式，只需要打印内存中储存的数据即可，但是我在代码中加入了新的设计，这个汇编模拟器是可以接受MIPS汇编代码，也可以接受0101这样的机器码，如果是输入的机器码，那也是可以接受指令方式转换的，通过反汇编。

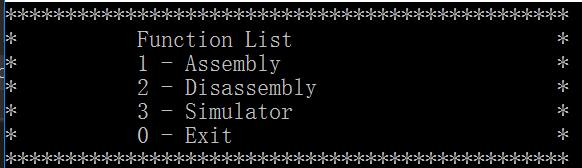
* 1. -->A-写汇编指令到内存，

写汇编指令到内存相对比较麻烦，因为在写汇编指令的时候需要考虑到lable的问题，因为要考虑到代码的设计元素，所以在读取的时候就需要对lable进行罗列和对应，因此相对会比较麻烦，但是这样可以减少后期的移动操作。增加算法效率。

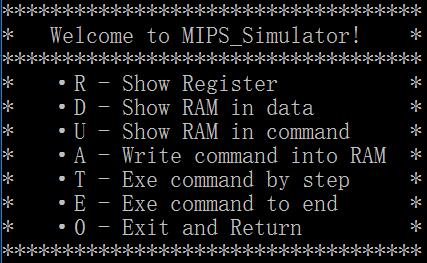
* 1. -->T-单步执行内存中的指令

MIPS汇编模拟平台在运行上面做了特殊设计，以命令行的模式模拟真实界面，仿造dos运行时的设计，我在平面上设计了代码区，寄存器区域，还有内存区域的显示设计，虽然样式很low但是已经是命令行版本的极限了。执行的时候，通过语句前面的小圆点来表示当前运行的语句和PC值。

1. 结果截图
   1. 功能界面截图



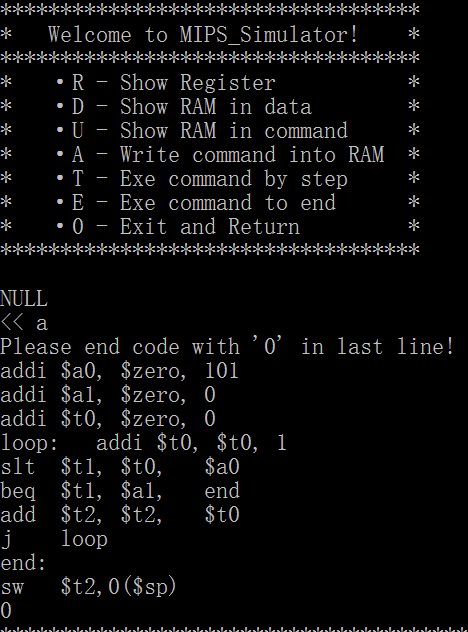
* 1. 模拟器使用说明



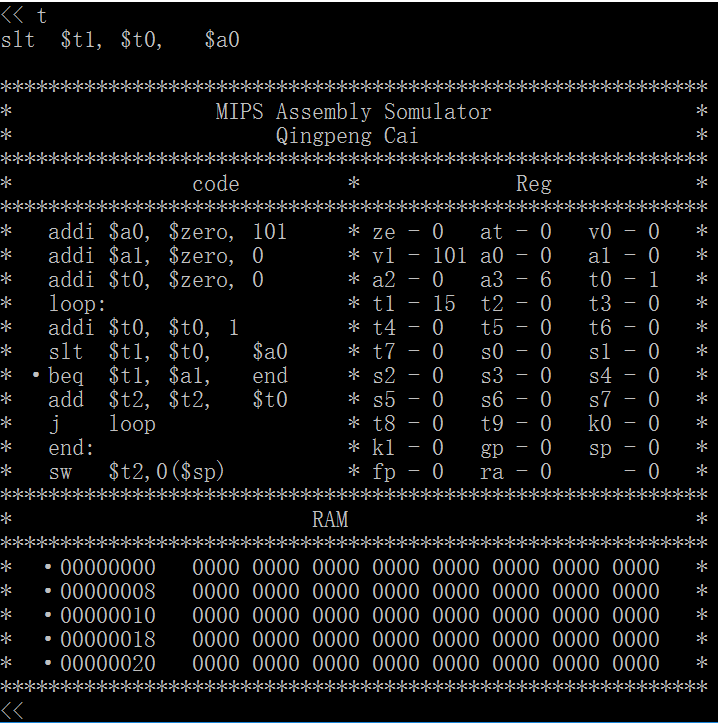
* 1. 模拟器界面设计



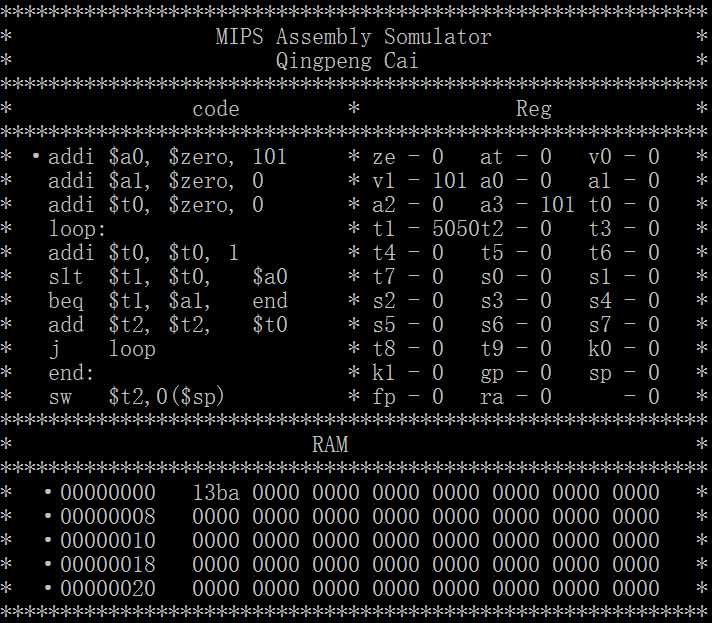
* 1. 运算模拟过程
     1. 代码输入a



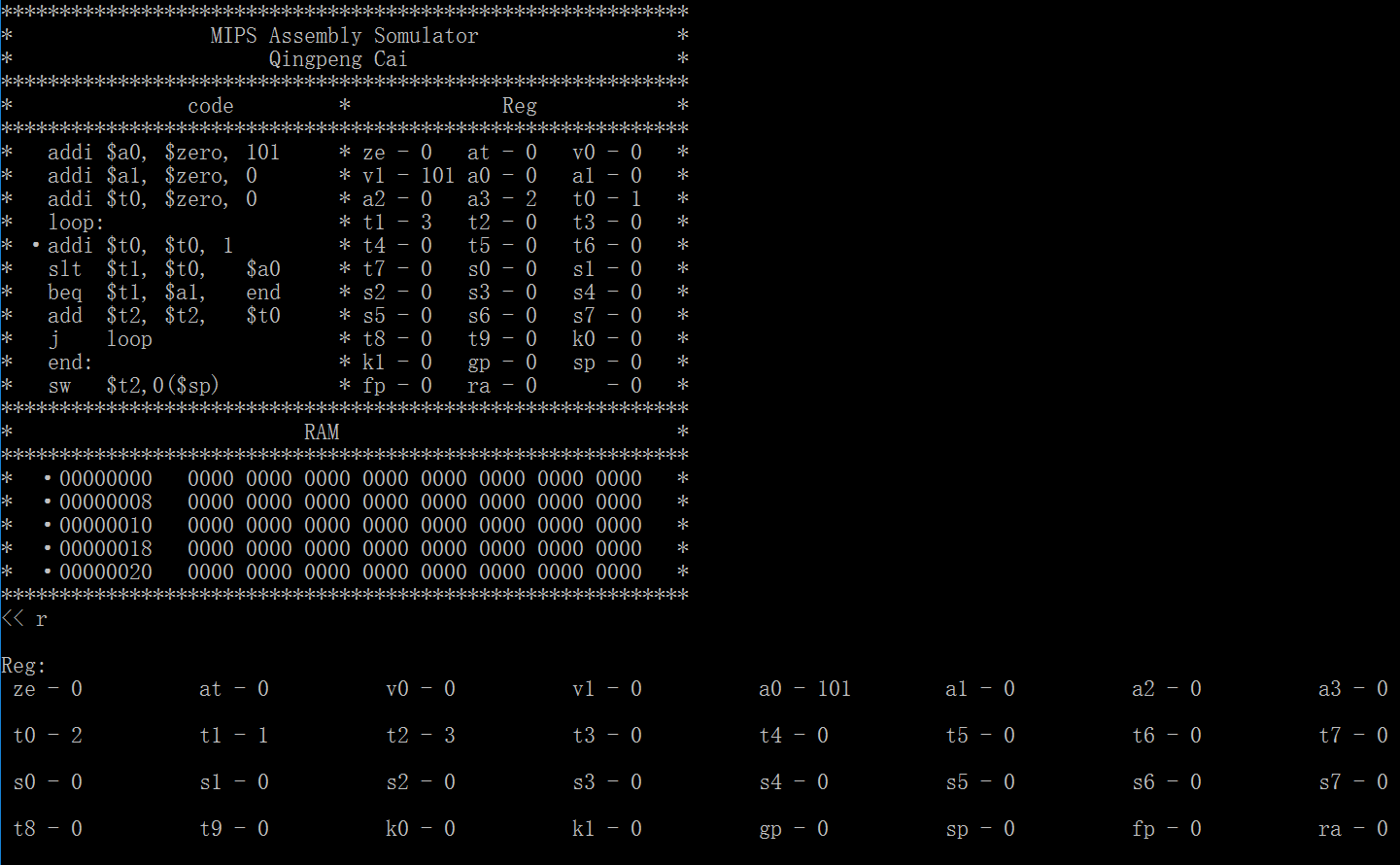
* + 1. 代码单步运行t



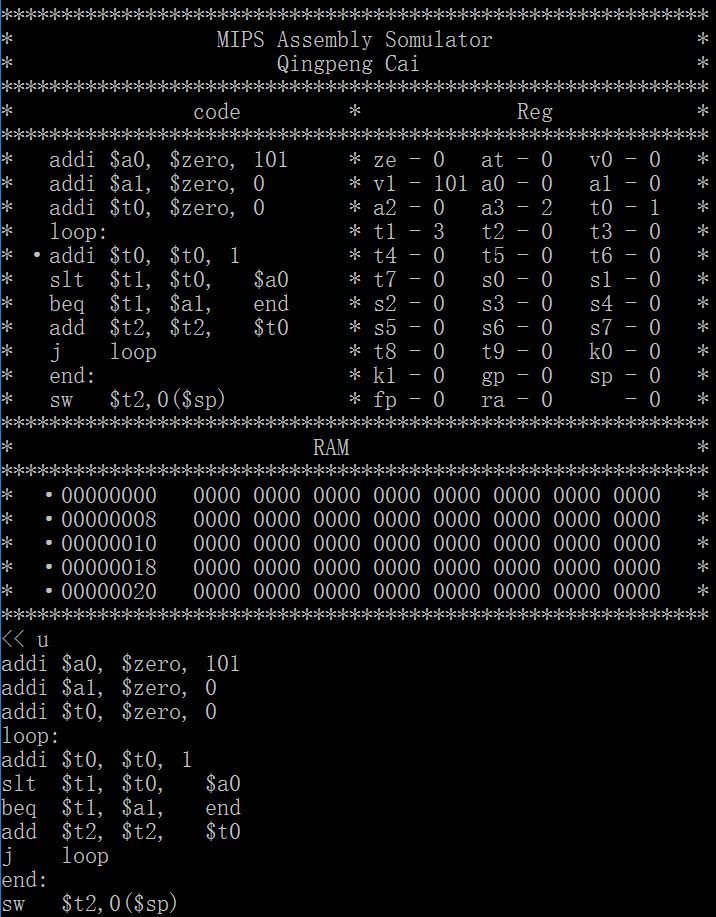
* + 1. 代码执行结果



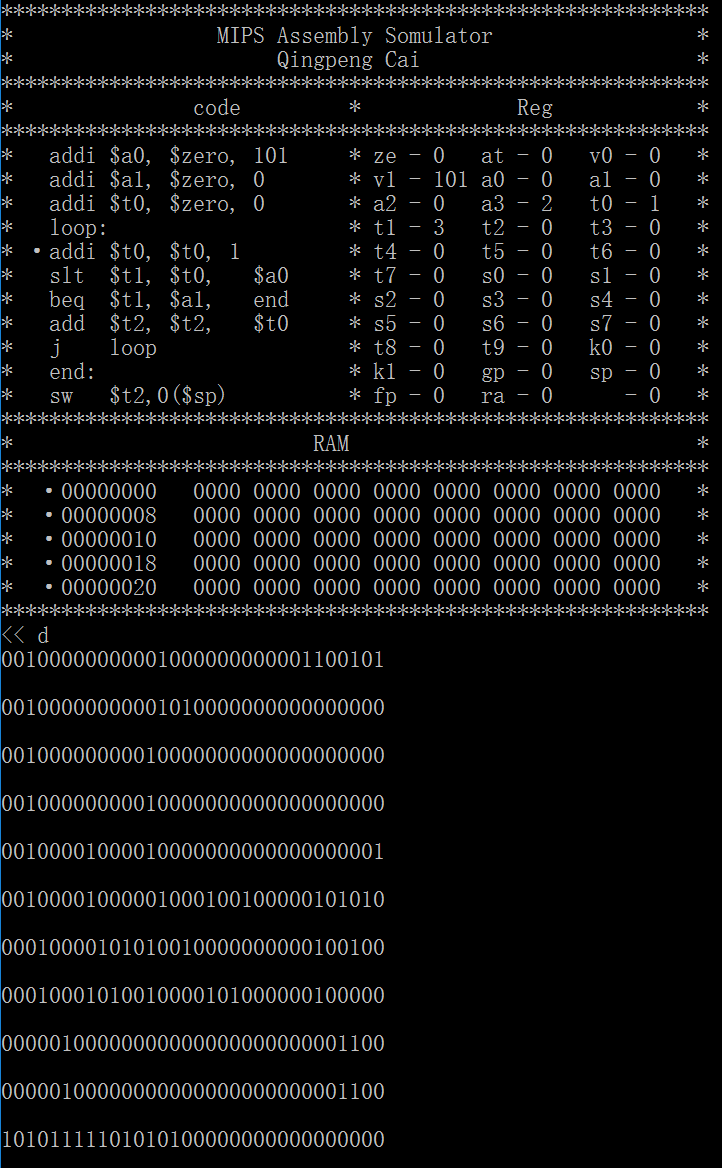
* + 1. 寄存器显示



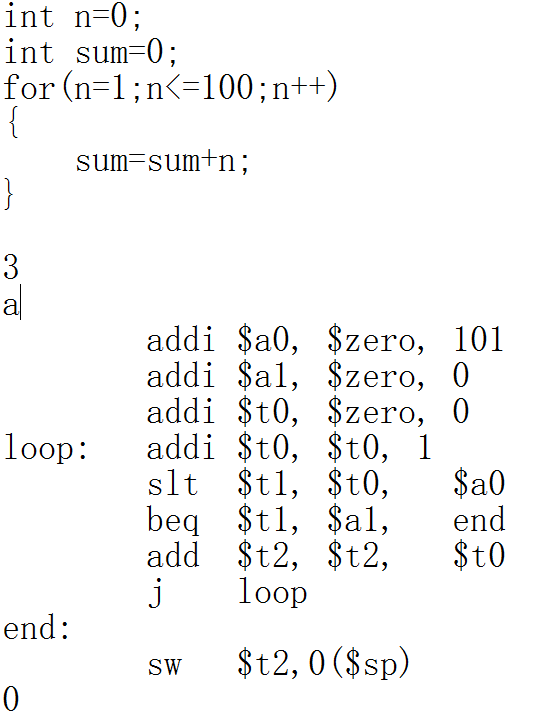
* + 1. 代码机器码显示



* + 1. 代码汇编指令显示



* + 1. 源代码含义



计算1-100的求和汇编代码

1. 程序优缺点

优点：

1. 三种程序的结合，MIPS汇编，MIPS反汇编，MIPS运行模拟器
2. 除了单步设计还加入了完整运行的功能
3. 特制图形化界面，显示代码，寄存器和内存
4. 特别设计使用·来表示当前PC，也就是当前所处的命令行
5. 报错信息
6. Lable 的合理处理

缺点：

1. 同图形化界面相比还有较大差距。
2. 改进方案

待定…

1. 周记

这周的作业相当有技术含量，不过基于模拟器基于之前写的MIPS汇编和反汇编器，然后重新创建了一个模拟平台（命令行版）。首先是指令的真实操作，这个只要通过每条指令后面跟随的操作及其内容就可以很容易的去做相应操作，比如add操作，就是把两个寄存器的内容相加，通过开辟数组，模拟内存，就是一行语句的问题。然后是命令行的独特设计。

指令方式，只需要打印内存中储存的数据即可，但是我在代码中加入了新的设计，这个汇编模拟器是可以接受MIPS汇编代码，也可以接受0101这样的机器码，如果是输入的机器码，那也是可以接受指令方式转换的，通过反汇编。

写汇编指令到内存相对比较麻烦，因为在写汇编指令的时候需要考虑到lable的问题，因为要考虑到代码的设计元素，所以在读取的时候就需要对lable进行罗列和对应，因此相对会比较麻烦，但是这样可以减少后期的移动操作。增加算法效率。

MIPS汇编模拟平台在运行上面做了特殊设计，以命令行的模式模拟真实界面，仿造dos运行时的设计，我在平面上设计了代码区，寄存器区域，还有内存区域的显示设计，虽然样式很low但是已经是命令行版本的极限了。执行的时候，通过语句前面的小圆点来表示当前运行的语句和PC值。

1. 算法代码

#include<stdio.h>

#include<string.h>

typedef unsigned int uint;

/\*

const int zero = 0; //Always has the value 0.Any writes to this registerare ignored.

int at; //Reserve for assembler

int v[2]; //values for results and expression evaluation

int a[4]; //arguments

int t[10]; //temporaries

int s[2]; //saved

int k[2]; //Reserved for use by the operating system kernel and for exception return.

int gp; //global pointer

int sp; //stack pointer

int fp; //frame pointer

int ra; //return address

\*/

enum COMMAND

{

ADD = 32, ADDU = 33, SUB = 34, SUBU = 35,

AND = 36, OR = 37, XOR = 38, NOR = 39,

SLT = 42, SLTU = 43, SLL = 0, SRL = 2, SRA = 3,

SLLV = 4, SRLV = 6, SRAV = 7, JR = 8,

ADDI = 8, ADDIU = 9, ANDI = 12, ORI = 13, XORI = 14,

LUI = 15, LW = 35, SW = 43, BEQ = 4, BNE = 5, SLTI = 10, SLTIU = 11,

J = 2, JAL = 3,

MOVE = 501, LI = 502, LA = 503, NOT = 504, NEG = 505, PUSH = 506, POP = 507,

BLT = 508, BGE = 509, BLE = 510, BGT = 511, ABS = 512, SWAP = 513, SNE = 514, SEQ = 515

};

#define $zero 0

#define $at 1

#define $v 2

#define $a 4

#define $t 8

#define $s 16

#define $m 24

#define $k 26

#define $gp 28

#define $sp 29

#define $fp 30

#define $ra 31

#define DIGIT 32

#define CommandSize 50

#define CodeLine 50

#define ERROR 0x7FFFFFFF

char Memory[32][3] = {

"ze", "at", "v0", "v1", "a0", "a1", "a2", "a3",

"t0", "t1", "t2", "t3", "t4", "t5", "t6", "t7",

"s0", "s1", "s2", "s3", "s4", "s5", "s6", "s7",

"t8", "t9", "k0", "k1", "gp", "sp", "fp", "ra"

};

int Reg[32];

int Stack[100];

uint RAM[40];

uint Command;

uint rs, rt, rd;

int imme;

uint Op;

uint func;

uint gp, sp, fp, ra;

char input[CommandSize] = { 0 };

char code[CodeLine][CommandSize];

char TableName[CodeLine][10];

uint TableAddr[CodeLine];

uint TableNum;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

除第一个寄存器始终为 0。最后一个寄存器，在调子程序时保存返回地址。其它寄存器从硬件上均

完全一样。区别仅在于编程的约定。

$t：为主调保存寄存器，由主程序负责寄存器的数据安全。即在子程序中可以随便使用，如果寄

存器内容的改变可能影响程序运行，由主程序负责将寄存器内容存入堆栈，在子程序返回时出栈恢复。

$s：为被调保存寄存器，由子程序负责寄存器的数据安全。即在子程序中如果需要改变寄存器内

容，则必须在改变之前将寄存器内容存入堆栈保存，在返回主程序之前恢复寄存器内容。

$a：为子程序调用参数寄存器，按顺序将调用参数存放在寄存器中，在子程序中使用。$a 也用于

系统功能调用的参数。

$v：子程序返回参数寄存器。

$sp：为堆栈指针。

$at：只作为汇编器进行程序汇编时，为伪指令扩展之用，程序员在自己编程序时不可使用

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int instruction();

//Disassembly

int ReadDIgit(char\*, int a, int b);

int ReadNum(char\*, int a, int b);

int DCheckOp(char \*);

void ReadThreeReg(char \*);

void ReadTwoRegShamt(char \*);

void ReadOneReg(char \*);

void ReadTwoRegImme(char \*);

void ReadOneRegImme(char \*);

void ReadAddr(char \*);

//assembly

uint GetNum(uint num, uint start, uint end);

uint WriteNum(uint &num, uint start, uint end, uint data);

uint ACheckOp(char\* command);

uint CheckReg(char\* command);

uint Imme(char\*command, uint up, uint \*p);

uint DecNum(char\*command, uint up, uint \*p);

uint HexNum(char\*command, uint up, uint \*p);

uint BinNum(char\*command, uint up, uint \*p);

void TypeImmeAddr(char\* command, uint Op);

void TypeThreeReg(char\* command);

void TypeTwoRegImme(char\* command);

void TypeTwoReg(char\* command);

void TypeOneReg(char\* command);

void TypeJumpAddr(char\* command);

void WriteImmeAddr(char\* command, uint Op);

void WriteThreeReg(char\* command, uint Op);

void WriteTwoRegImme(char\* command, uint Op);

void WriteTwoReg(char\* command, uint Op);

void WriteOneReg(char\* command, uint reg, uint Op);

void WriteJumpAddr(char\* command, uint Op);

void Output(uint Command);

void Error(uint num);

//simulator

uint Simulator();

void GraphicalInterface(uint code\_line, uint PC);

void SimClear();

uint Handle(char \*command,uint PC);

uint SCheckOp(char \*command);

void ShowReg();

uint WriteRAM(uint code\_line);

uint ExeCommandByStep(uint code\_line,uint PC);

uint ExeCommandToEnd(uint code\_line, uint PC);

void ReadRAMinComm(uint code\_line);

void ReadRAMinData(uint code\_line);

int main()

{

int i = 0;

int choice = 0;

while (1) {

i = 0;

rs = 0; rt = 0; rd = 0;

imme = 0;

Op = 0;

func = 0;

Command = 0;

memset(input, 0, sizeof(input));

memset(Reg, 0, sizeof(Reg));

choice = instruction();

switch (choice)

{

case 0: return 0;

case 1:

while (1)

{

printf("Please input your MIPS code:\n");

gets\_s(input);

if (input[0] == '0')

break;

if (input[0] == '1')

{

printf("Please input your MIPS code:\n");

gets\_s(input);

}

while (input[i] == ' ')i++;

ACheckOp(input + i);

Output(Command);

printf("\n");

printf("Do you want to go on?<1/0>\n");

}

break;

case 2:

printf("Please input your Machine Code:\n");

while (1)

{

gets\_s(input);

if (input[0] == 'N')

break;

if (input[0] == 'Y')

{

printf("Please input your Machine Code:\n");

gets\_s(input);

}

while (input[i] == ' ')i++;

DCheckOp(input + i);

printf("\n");

printf("Do you want to go on?<Y/N>\n");

}

break;

case 3:

Simulator();

break;

default:

break;

}

}

return 0;

}

int instruction()

{

int i = 10;

while (i)

{

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\* Function List \*\n");

printf("\* 1 - Assembly \*\n");

printf("\* 2 - Disassembly \*\n");

printf("\* 3 - Simulator \*\n");

printf("\* 0 - Exit \*\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

scanf("%d", &i);

if (i == 1 || i == 2 || i == 3)

{

while (getchar() != '\n');

return i;

}

Error(5);

while (getchar() != '\n');

}

return 0;

}

uint Simulator()

{

int i = 0;

uint code\_line = 0;

uint PC = 0;

char order = '\n';

TableNum = 0;

memset(Reg, 0, sizeof(Reg));

memset(code, 0, sizeof(code));

memset(TableName, 0, sizeof(TableName));

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\* Welcome to MIPS\_Simulator! \*\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\* ·R - Show Register \*\n");

printf("\* ·D - Show RAM in data \*\n");

printf("\* ·U - Show RAM in command \*\n");

printf("\* ·A - Write command into RAM \*\n");

printf("\* ·T - Exe command by step \*\n");

printf("\* ·E - Exe command to end \*\n");

printf("\* ·0 - Exit and Return \*\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");

while (1)

{

if (code\_line == 0)

printf("NULL\n");

else GraphicalInterface(code\_line,PC);

printf("<< ");

scanf("%c", &order);

while (order == '\n')

{

printf("<< ");

scanf("%c", &order);

}

while (getchar() != '\n');

switch (order)

{

case 'R':case 'r':ShowReg(); break;

case 'D':case 'd':ReadRAMinData(code\_line); break;

case 'U':case 'u':ReadRAMinComm(code\_line); break;

case 'A':case 'a':code\_line = WriteRAM(code\_line); break;

case 'T':case 't':PC = ExeCommandByStep(code\_line, PC); break;

case 'E':case 'e':ExeCommandToEnd(code\_line, PC); break;

case '0': return 0;

case '\n':break;

default:

printf("You input a wrong command!\n");

printf("Please input again!\n");

break;

}

}

return 0;

}

void GraphicalInterface(uint code\_line, uint PC)

{

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\* MIPS Assembly Somulator \*\n");

printf("\* Qingpeng Cai \*\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\* code \* Reg \*\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

int Reg\_num = 0;

int Ram\_num = 0;

for (int j = 0; j < code\_line; j++)

{

if (j == PC)

{

printf("\* ·");

printf("%-25s\* ", code[j]);

if (Reg\_num < 32) {

printf("%-2s - %-4d", Memory[Reg\_num++], Reg[Reg\_num]);

if (Reg\_num == 32){

printf("%-2s - %-4d\*\n", Memory[Reg\_num++], Reg[Reg\_num]);

}

else{

printf("%-2s - %-4d", Memory[Reg\_num++], Reg[Reg\_num]);

printf("%-2s - %-4d\*\n", Memory[Reg\_num++], Reg[Reg\_num]);

}

}

else

printf(" \*\n");

}

else

{

printf("\* ");

printf("%-25s\* ", code[j]);

if (Reg\_num < 32) {

printf("%-2s - %-4d", Memory[Reg\_num++], Reg[Reg\_num]);

if (Reg\_num == 32) {

printf("%-2s - %-4d\*\n", Memory[Reg\_num++], Reg[Reg\_num]);

}

else {

printf("%-2s - %-4d", Memory[Reg\_num++], Reg[Reg\_num]);

printf("%-2s - %-4d\*\n", Memory[Reg\_num++], Reg[Reg\_num]);

}

}

else

printf(" \*\n");

}

}

while (Reg\_num < 32)

{

printf("\* \* ");

printf("%-2s - %-4d", Memory[Reg\_num++], Reg[Reg\_num]);

if (Reg\_num < 32) {

printf("%-2s - %-4d", Memory[Reg\_num++], Reg[Reg\_num]);

if (Reg\_num == 32) {

printf("%-2s - %-4d\*\n", Memory[Reg\_num++], Reg[Reg\_num]);

}

else {

printf("%-2s - %-4d", Memory[Reg\_num++], Reg[Reg\_num]);

printf("%-2s - %-4d\*\n", Memory[Reg\_num++], Reg[Reg\_num]);

}

}

}

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\* RAM \*\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

for (int i = 0; i < 5; i++)

{

printf("\* ·%08x ", i \* 8);

for (int j = 0; j < 8; j++) {

printf("%04x ", RAM[i \* 8 + j]);

}

printf(" \*\n");

}

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

}

uint Handle(char \*command,uint PC)

{

uint Op;

int i = 0;

while (command[i] != '\0') {

if (command[i] == ':')

{

PC++;

return PC;

}

i++;

}

Op = SCheckOp(command);

if (Op == ADD)

{

Reg[rd] = Reg[rs] + Reg[rt];

}

else if (Op == ADDI)

{

Reg[rt] = Reg[rs] + imme;

}

else if (Op == SUB)

{

Reg[rd] = Reg[rs] - Reg[rt];

}

else if (Op == SLT)

{

if (Reg[rs] < Reg[rt])

Reg[rd] = 1;

else Reg[rd] = 0;

}

else if (Op == LW)

{

Reg[rt] = RAM[rs + imme];

}

else if (Op == SW)

{

RAM[Reg[rs] + imme] = Reg[rt];

}

else if (Op == BEQ)

{

if(Reg[rs]==Reg[rt])

PC = (imme >> 2) - 1;

}

else if (Op == J)

{

PC = imme/4 - 1;

}

if(Op!=0)

PC++;

return PC;

}

uint SCheckOp(char \*command)

{

char check[5];

int i = 0;

uint record = 0;

strncpy(check, command, sizeof(char) \* 5);

while (check[i] != ' ')i++;

if (i < 5)

check[i] = '\0';

if (0 == strcmp(check, "add") || 0 == strcmp(check, "ADD"))

return ADD;

if (0 == strcmp(check, "addi") || 0 == strcmp(check, "ADDI"))

return ADDI;

if (0 == strcmp(check, "sub") || 0 == strcmp(check, "SUB"))

return SUB;

if (0 == strcmp(check, "slt") || 0 == strcmp(check, "SLT"))

return SLT;

if (0 == strcmp(check, "lw") || 0 == strcmp(check, "LW"))

return LW;

if (0 == strcmp(check, "sw") || 0 == strcmp(check, "SW"))

return SW;

if (0 == strcmp(check, "beq") || 0 == strcmp(check, "BEQ"))

return BEQ;

if (0 == strcmp(check, "j") || 0 == strcmp(check, "J"))

return J;

else return 0;

}

void SimClear()

{

rs = 0; rt = 0; rd = 0;

imme = 0;

Op = 0;

func = 0;

Command = 0;

memset(input, 0, sizeof(input));

}

void ShowReg()

{

printf("\nReg:\n");

for (int i = 0; i < 32; i++)

{

printf(" %s - %d \t", Memory[i], Reg[i]);

if (i==7||i==15||i==23||i==31)

printf("\n");

}

}

uint WriteRAM(uint code\_line)

{

printf("Please end code with '0' in last line!\n");

for (int j = code\_line; j < CodeLine; j++)

{

gets\_s(code[j]);

if (code[j][0] == '0')

{

return j;

}

for (int m = 0; m < CommandSize; m++)

{

if (code[j][m] == ':') {

int n = 0;

int size = 0;

while (code[j][n] == ' ') //去掉空格

n++;

while (code[j][n] != ':') {

TableName[TableNum][size++] = code[j][n++];

}

TableName[TableNum][size] = '\0';

code[j][n++] = ':';

if (size == 0) {

printf("Error: you have TableName mistake in %d line\n", j);

return ERROR;

}

else {

TableAddr[TableNum++] = j;

j++;

for (int m = n; m < CommandSize; m++) {

if (code[j - 1][m] == '\0')

{

j--;

break;

}

if (code[j - 1][m] != ' ')

{

int k = 0;

while (code[j - 1][m] != '\0') {

code[j][k++] = code[j - 1][m++];

}

code[j][k] = '\0';

code[j - 1][n] = '\0';

break;

}

}

}

break;

}

}

}return 1;

}

uint ExeCommandByStep(uint code\_line,uint PC)

{

int i = 0;

//while (code[PC][0] != '0')

if(PC<=code\_line)

{

SimClear();

//ShowReg();

memcpy(input, code[PC], sizeof(char) \* 50);

printf("%s\n\n", code[PC]);

while (input[i] == ' ')i++;

ACheckOp(input + i); //把汇编先转化成command命令行

PC = Handle(input + i, PC);

}

return PC;

}

uint ExeCommandToEnd(uint code\_line, uint PC)

{

int i = 0;

while (code[PC][0] != '0')

{

SimClear();

//ShowReg();

memcpy(input, code[PC], sizeof(char) \* 50);

printf("%s\n\n", code[PC]);

while (input[i] == ' ')i++;

ACheckOp(input + i); //把汇编先转化成command命令行

PC = Handle(input + i, PC);

}

return PC;

}

void ReadRAMinComm(uint code\_line)

{

int i = 0;

while (code[0][i] == ' ')i++;

if (code[0][i] != '0' && code[0][i] != '1')

{

for (int j = 0; j < code\_line; j++)

{

printf("%s\n", code[j]);

}

}

else

{

for (int j = 0; j < code\_line; j++)

{

while (code[j][i] == ' ')i++;

DCheckOp(code[j] + i);

printf("\n");

}

}

}

void ReadRAMinData(uint code\_line)

{

int i = 0;

while (code[0][i] == ' ')i++;

if (code[0][i] == '0' || code[0][i] == '1')

{

for (int j = 0; j < code\_line; j++)

{

printf("%s\n", code[j]);

}

}

else

{

for (int j = 0; j < code\_line; j++)

{

while (code[j][i] == ' ')i++;

ACheckOp(code[j] + i);

Output(Command);

printf("\n");

}

}

}

//disassembly

int ReadDIgit(char\* command, int a, int b)

{

int i = 0;

uint num = 0;

for (i = a; i <= b; i++)

{

num <<= 1;

num += command[i] - '0' + 0;

}

return num;

}

int DCheckOp(char\* command)

{

Op = ReadDIgit(command, 0, 5);

switch (Op)

{

case 0:func = ReadDIgit(command, 26, 31);

switch (func)

{

case ADD: case ADDU: case SUB: case SUBU: case AND: case OR:

case XOR: case NOR: case SLT: case SLTU: case SLLV: case SRLV: case SRAV:

ReadThreeReg(command);

break;

case SLL: case SRL:case SRA:

ReadTwoRegShamt(command);

break;

case JR: ReadOneReg(command); break;

default:

break;

}

case ADDI: case ADDIU: case ANDI: case ORI: case XORI:

case LW: case SW: case BEQ: case BNE: case SLTI: case SLTIU:

ReadTwoRegImme(command);

break;

case LUI:

ReadOneRegImme(command);

break;

case J:case JAL:

ReadAddr(command);

break;

default:

break;

}

return 0;

}

int ReadNum(char\*command, int a, int b)

{

int i = 0;

int num = 0;

for (i = a + 1; i <= b; i++)

{

num <<= 1;

num += command[i] - '0' + 0;

}

if (command[a] == '1')

return ~num + 1;

else if (command[a] == '0')

return num;

}

void ReadThreeReg(char \*command)

{

rs = ReadDIgit(command, 6, 10);

rt = ReadDIgit(command, 11, 15);

rd = ReadDIgit(command, 16, 20);

switch (func)

{

case ADD: printf("ADD $%s,$%s,$%s", Memory[rd], Memory[rs], Memory[rt]); break;

case ADDU:printf("ADDU $%s,$%s,$%s", Memory[rd], Memory[rs], Memory[rt]); break;

case SUB: printf("SUB $%s,$%s,$%s", Memory[rd], Memory[rs], Memory[rt]); break;

case SUBU:printf("SUBU $%s,$%s,$%s", Memory[rd], Memory[rs], Memory[rt]); break;

case AND: printf("AND $%s,$%s,$%s", Memory[rd], Memory[rs], Memory[rt]); break;

case OR: printf("OR $%s,$%s,$%s", Memory[rd], Memory[rs], Memory[rt]); break;

case XOR: printf("XOR $%s,$%s,$%s", Memory[rd], Memory[rs], Memory[rt]); break;

case NOR: printf("NOR $%s,$%s,$%s", Memory[rd], Memory[rs], Memory[rt]); break;

case SLT: printf("SLT $%s,$%s,$%s", Memory[rd], Memory[rs], Memory[rt]); break;

case SLTU:printf("SLTU $%s,$%s,$%s", Memory[rd], Memory[rs], Memory[rt]); break;

case SLLV:printf("SLLV $%s,$%s,$%s", Memory[rd], Memory[rs], Memory[rt]); break;

case SRLV:printf("SRLV $%s,$%s,$%s", Memory[rd], Memory[rs], Memory[rt]); break;

case SRAV:printf("SRAV $%s,$%s,$%s", Memory[rd], Memory[rs], Memory[rt]); break;

default:

break;

}

}

void ReadTwoRegShamt(char \*command)

{

rt = ReadDIgit(command, 11, 15);

rd = ReadDIgit(command, 16, 20);

imme = ReadDIgit(command, 21, 25);

switch (func)

{

case SLL: printf("SLL $%s,$%s,%d", Memory[rd], Memory[rt], imme); break;

case SRL: printf("SRL $%s,$%s,%d", Memory[rd], Memory[rt], imme); break;

case SRA: printf("SRA $%s,$%s,%d", Memory[rd], Memory[rt], imme); break;

default:

break;

}

}

void ReadOneReg(char \*command)

{

rs = ReadDIgit(command, 6, 10);

switch (func)

{

case JR: printf("JR $%s", Memory[rs]); break;

default:

break;

}

}

void ReadTwoRegImme(char \*command)

{

rs = ReadDIgit(command, 6, 10);

rt = ReadDIgit(command, 11, 15);

imme = ReadNum(command, 16, 31);

switch (Op)

{

case ADDI: printf("ADDI $%s,$%s,%d", Memory[rt], Memory[rs], imme); break;

case ADDIU: printf("ADDIU $%s,$%s,%d", Memory[rt], Memory[rs], imme); break;

case ANDI: printf("ANDI $%s,$%s,%d", Memory[rt], Memory[rs], imme); break;

case ORI: printf("ORI $%s,$%s,%d", Memory[rt], Memory[rs], imme); break;

case XORI: printf("XORI $%s,$%s,%d", Memory[rt], Memory[rs], imme); break;

case LW: printf("LW $%s,%d($%s)", Memory[rt], imme, Memory[rs]); break;

case SW: printf("SW $%s,%d($%s)", Memory[rt], imme, Memory[rs]); break;

case BEQ: printf("BEQ $%s,$%s,%d", Memory[rt], Memory[rs], imme); break;

case BNE: printf("BNE $%s,$%s,%d", Memory[rt], Memory[rs], imme); break;

case SLTI: printf("SLTI $%s,$%s,%d", Memory[rt], Memory[rs], imme); break;

case SLTIU: printf("SLTIU $%s,$%s,%d", Memory[rt], Memory[rs], imme); break;

default:

break;

}

}

void ReadOneRegImme(char \*command)

{

rt = ReadDIgit(command, 11, 15);

imme = ReadNum(command, 16, 31);

switch (Op)

{

case LUI:printf("LUI $%s,%d", Memory[rt], imme); break;

default:

break;

}

}

void ReadAddr(char \*command)

{

imme = ReadNum(command, 6, 31);

switch (Op)

{

case J: printf("J %d", imme); break;

case JAL:printf("JAL %d", imme); break;

default:

break;

}

}

//assembly

uint GetNum(uint num, uint start, uint end)

{

if (start >= 0 && end < 32 && start < end)

{

uint check = 0x80000000;

uint copy = end;

while (copy > start)

{

check >>= 1;

check += 0x80000000;

copy--;

}

check >>= start;

check &= num;

check >>= (31 - end);

return check;

}

else

{

Error(1);

}

return 0;

}

uint WriteNum(uint &num, uint start, uint end, uint data)

{

if (start >= 0 && end < 32 && start < end)

{

uint check = 0x80000000;

uint copy = end;

while (copy > start)

{

check >>= 1;

check += 0x80000000;

copy--;

}

check >>= start;

check = ~check;

num &= check;

data <<= (31 - end);

num |= data;

return num;

}

else

{

Error(2);

}

return 0;

}

uint ACheckOp(char\* command)

{

char check[5];

int i = 0;

uint record = 0;

strncpy(check, command, sizeof(char) \* 5);

while (check[i] != ' ')i++;

if (i < 5)

check[i] = '\0';

if (0 == strcmp(check, "lui") || 0 == strcmp(check, "LUI"))

{

TypeImmeAddr(command + i, LUI);

WriteImmeAddr(command + i, LUI);

return 1;

}

if (0 == strcmp(check, "lw") || 0 == strcmp(check, "LW"))

{

TypeImmeAddr(command + i, LW);

WriteImmeAddr(command + i, LW);

return 1;

}

if (0 == strcmp(check, "sw") || 0 == strcmp(check, "SW"))

{

TypeImmeAddr(command + i, SW);

WriteImmeAddr(command + i, SW);

return 1;

}

if (0 == strcmp(check, "add") || 0 == strcmp(check, "ADD"))

{

TypeThreeReg(command + i);

WriteThreeReg(command + i, ADD);

return 1;

}

if (0 == strcmp(check, "addu") || 0 == strcmp(check, "ADDU"))

{

TypeThreeReg(command + i);

WriteThreeReg(command + i, ADDU);

return 1;

}

if (0 == strcmp(check, "addi") || 0 == strcmp(check, "ADDI"))

{

TypeImmeAddr(command + i, ADDI);

WriteImmeAddr(command + i, ADDI);

return 1;

}

if (0 == strcmp(check, "addiu") || 0 == strcmp(check, "ADDIU"))

{

TypeImmeAddr(command + i, ADDIU);

WriteImmeAddr(command + i, ADDIU);

return 1;

}

if (0 == strcmp(check, "sub") || 0 == strcmp(check, "SUB"))

{

TypeThreeReg(command + i);

WriteThreeReg(command + i, SUB);

return 1;

}

if (0 == strcmp(check, "subu") || 0 == strcmp(check, "SUBU"))

{

TypeThreeReg(command + i);

WriteThreeReg(command + i, SUBU);

return 1;

}

if (0 == strcmp(check, "slt") || 0 == strcmp(check, "SLT"))

{

TypeThreeReg(command + i);

WriteThreeReg(command + i, SLT);

return 1;

}

if (0 == strcmp(check, "sltu") || 0 == strcmp(check, "SLTU"))

{

TypeThreeReg(command + i);

WriteThreeReg(command + i, SLTU);

return 1;

}

if (0 == strcmp(check, "slti") || 0 == strcmp(check, "SLTI"))

{

TypeImmeAddr(command + i, SLTI);

WriteImmeAddr(command + i, SLTI);

return 1;

}

if (0 == strcmp(check, "sltiu") || 0 == strcmp(check, "SLTIU"))

{

TypeImmeAddr(command + i, SLTIU);

WriteImmeAddr(command + i, SLTIU);

return 1;

}

if (0 == strcmp(check, "and") || 0 == strcmp(check, "AND"))

{

TypeThreeReg(command + i);

WriteThreeReg(command + i, AND - 300);

return 1;

}

if (0 == strcmp(check, "andi") || 0 == strcmp(check, "ANDI"))

{

TypeImmeAddr(command + i, ANDI);

WriteImmeAddr(command + i, ANDI);

return 1;

}

if (0 == strcmp(check, "or") || 0 == strcmp(check, "OR"))

{

TypeThreeReg(command + i);

WriteThreeReg(command + i, OR);

return 1;

}

if (0 == strcmp(check, "ori") || 0 == strcmp(check, "ORI"))

{

TypeImmeAddr(command + i, ORI);

WriteImmeAddr(command + i, ORI);

return 1;

}

if (0 == strcmp(check, "xor") || 0 == strcmp(check, "XOR"))

{

TypeThreeReg(command + i);

WriteThreeReg(command + i, XOR);

return 1;

}

if (0 == strcmp(check, "xori") || 0 == strcmp(check, "XORI"))

{

TypeImmeAddr(command + i, XORI);

WriteImmeAddr(command + i, XORI);

return 1;

}

if (0 == strcmp(check, "nor") || 0 == strcmp(check, "NOR"))

{

TypeThreeReg(command + i);

WriteThreeReg(command + i, NOR);

return 1;

}

if (0 == strcmp(check, "sll") || 0 == strcmp(check, "SLL"))

{

TypeTwoRegImme(command + i);

WriteTwoRegImme(command + i, SLL);

return 1;

}

if (0 == strcmp(check, "sllv") || 0 == strcmp(check, "SLLV"))

{

TypeThreeReg(command + i);

WriteThreeReg(command + i, SLLV);

return 1;

}

if (0 == strcmp(check, "srl") || 0 == strcmp(check, "SRL"))

{

TypeTwoRegImme(command + i);

WriteTwoRegImme(command + i, SRL);

return 1;

}

if (0 == strcmp(check, "srlv") || 0 == strcmp(check, "SRLV"))

{

TypeThreeReg(command + i);

WriteThreeReg(command + i, SRLV);

return 1;

}

if (0 == strcmp(check, "sra") || 0 == strcmp(check, "SRA"))

{

TypeTwoRegImme(command + i);

WriteTwoRegImme(command + i, SRA);

return 1;

}

if (0 == strcmp(check, "srav") || 0 == strcmp(check, "SRAV"))

{

TypeThreeReg(command + i);

WriteThreeReg(command + i, SRAV);

return 1;

}

if (0 == strcmp(check, "beq") || 0 == strcmp(check, "BEQ"))

{

TypeImmeAddr(command + i, BEQ);

WriteImmeAddr(command + i, BEQ);

return 1;

}

if (0 == strcmp(check, "bne") || 0 == strcmp(check, "BNE"))

{

TypeImmeAddr(command + i, BNE);

WriteImmeAddr(command + i, BNE);

return 1;

}

if (0 == strcmp(check, "j") || 0 == strcmp(check, "J"))

{

TypeJumpAddr(command + i);

WriteJumpAddr(command + i, J);

return 1;

}

if (0 == strcmp(check, "jar") || 0 == strcmp(check, "JAR"))

{

TypeJumpAddr(command + i);

WriteJumpAddr(command + i, JAL);

return 1;

}

if (0 == strcmp(check, "jr") || 0 == strcmp(check, "JR"))

{

TypeOneReg(command + i);

WriteOneReg(command + i, 1, JR);

return 1;

}

if (0 == strcmp(check, "move") || 0 == strcmp(check, "MOVE"))

{

while (command[i] != '$')i++;//$rd

rd = CheckReg(command + i);

i = i + 3;//,

while (command[i] != ',')i++;

i++;

while (command[i] != '$')i++;//$rs

rs = CheckReg(command + i);

i = i + 3;//,

WriteThreeReg(command + i, ADD);

return 1;

}

if (0 == strcmp(check, "li") || 0 == strcmp(check, "LI"))

{

while (command[i] != '$')i++;//$rd

rd = CheckReg(command + i);

i = i + 3;//,

while (command[i] == ' ')i++;//num

imme = Imme(command + i, 16, &record);

i += record;

if (imme < 0x00010000)

WriteImmeAddr(command + i, ADDI);

else

{

WriteImmeAddr(command + i, LUI);

WriteImmeAddr(command + i, ORI);

}

return 1;

}

// if (0 == strcmp(check, "la") || 0 == strcmp(check, "LA"))

if (0 == strcmp(check, "not") || 0 == strcmp(check, "NOT"))

{

while (command[i] != '$')i++;//$rd

rd = CheckReg(command + i);

i = i + 3;//,

while (command[i] != ',')i++;

i++;

while (command[i] != '$')i++;//$rs

rs = CheckReg(command + i);

i = i + 3;//,

WriteThreeReg(command + i, NOR);

return 1;

}

if (0 == strcmp(check, "neg") || 0 == strcmp(check, "NEG"))

{

rs = 0;

while (command[i] != '$')i++;//$rd

rd = CheckReg(command + i);

i = i + 3;//,

while (command[i] != ',')i++;

i++;

while (command[i] != '$')i++;//$rs

rt = CheckReg(command + i);

i = i + 3;//,

WriteThreeReg(command + i, SUB);

return 1;

}

// if (0 == strcmp(check, "push") || 0 == strcmp(check, "PUSH"))

// if (0 == strcmp(check, "pop") || 0 == strcmp(check, "POP"))

// if (0 == strcmp(check, "blt") || 0 == strcmp(check, "BLT"))

// if (0 == strcmp(check, "bgt") || 0 == strcmp(check, "BGT"))

// if (0 == strcmp(check, "ble") || 0 == strcmp(check, "BLE"))

// if (0 == strcmp(check, "bge") || 0 == strcmp(check, "BGE"))

if (0 == strcmp(check, "abs") || 0 == strcmp(check, "ABS"))

{

uint record\_rt;

while (command[i] != '$')i++;//$rt

rt = CheckReg(command + i);

i = i + 3;//,

while (command[i] != ',')i++;

i++;

record\_rt = rt;

rt = 31;

rd = $at;

while (command[i] != '$')i++;//$rs

rs = CheckReg(command + i);

i = i + 3;//,

WriteTwoRegImme(command + i, SRA);

Output(Command);

rd = record\_rt;

rt = $at;

WriteThreeReg(command + i, XOR);

Output(Command);

rs = rd;

rt = $at;

WriteThreeReg(command + i, SUB);

return 1;

}

if (0 == strcmp(check, "swap") || 0 == strcmp(check, "SWAP"))

{

uint r1, r2;

while (command[i] != '$')i++;//$rt

r1 = CheckReg(command + i);

i = i + 3;//,

while (command[i] != ',')i++;

while (command[i] != '$')i++;//$rt

r2 = CheckReg(command + i);

i = i + 3;//,

while (command[i] != ',')i++;

rd = r1; rs = r1; rt = r2;

WriteThreeReg(command + i, XOR);

Output(Command);

rd = r2; rs = r1; rt = r2;

WriteThreeReg(command + i, XOR);

Output(Command);

rd = r1; rs = r1; rt = r2;

WriteThreeReg(command + i, XOR);

return 1;

}

if (0 == strcmp(check, "sne") || 0 == strcmp(check, "SNE"))

{

uint r1, r2, r3;

while (command[i] != '$')i++;//$rt

r1 = CheckReg(command + i);

i = i + 3;//,

while (command[i] != ',')i++;

while (command[i] != '$')i++;//$rt

r2 = CheckReg(command + i);

i = i + 3;//,

while (command[i] != ',')i++;

while (command[i] != '$')i++;//$rt

r3 = CheckReg(command + i);

i = i + 3;//,

rd = $at; rs = r2, rt = r3;

WriteThreeReg(command + i, SUB);

Output(Command);

rd = r1; rs = $zero; rt = $at;

WriteThreeReg(command + i, SLTU);

return 1;

}

if (0 == strcmp(check, "seq") || 0 == strcmp(check, "SEQ"))

{

uint r1, r2, r3;

while (command[i] != '$')i++;//$rt

r1 = CheckReg(command + i);

i = i + 3;//,

while (command[i] != ',')i++;

while (command[i] != '$')i++;//$rt

r2 = CheckReg(command + i);

i = i + 3;//,

while (command[i] != ',')i++;

while (command[i] != '$')i++;//$rt

r3 = CheckReg(command + i);

i = i + 3;//,

rd = $at; rs = r2, rt = r3;

WriteThreeReg(command + i, SUB);

rd = r1; rs = $zero; rt = 1;

WriteThreeReg(command + i, SLTU);

}

else return 0;

return 1;

}

uint CheckReg(char\* command)

{

char check[5];

int i = 0;

strncpy(check, command + 1, sizeof(char) \* 5);

while ((check[i] >= 'a'&&check[i] <= 'z') || (check[i] >= '0'&&check[i] <= '9'))i++;

check[i] = '\0';

for (uint j = 0; j < 32; j++)

{

if (0 == strcmp(check, Memory[j]))

return j;

}

return 0;

}

uint Imme(char\*command, uint up, uint \*p)

{

int i = 0;

int n = 0;

while (command[n] != '\0'&&command[n]!=' '){

n++;

}

command[n] = '\0';

for (n = 0; n < TableNum; n++)

{

if (0 == strcmp(command, TableName[n]))

return TableAddr[n] \* 4;

}

while (command[i] >= '0' && command[i] <= '9')i++;

if ((command[i] == 'B' || command[i] == 'b') && command[i - 1] == '0'&&i == 1)

return BinNum(command + i + 1, up, p);

else if ((command[i] == 'X' || command[i] == 'x') && command[i - 1] == '0'&&i == 1)

return HexNum(command + i + 1, up, p);

else return DecNum(command, up, p);

}

uint DecNum(char\*command, uint up, uint \*p)

{

int i = 0, j = 0;

int flag = 0;

uint result = 0;

if (command[i] != '-'&&(command[i]<'0'||command[i]>'9'))

{

Error(3);

return ERROR;

}

else

{

if (command[i] == '-') //负数

{

flag = 1;

i++;

}

while (command[i] >= '0' && command[i] <= '9')

{

result \*= 10;

result += command[i] - '0' + 0;

i++;

}

if (result < (int)(0x00000001 << (up - 1)))

{

if (flag)

{

result = ~result;

result += 1;

result &= 0x0000ffff;

}

if (command[i] == '+')

result += Imme(command + i + 1, up, p);

else if (command[i] == '-')

result -= Imme(command + i + 1, up, p);

else if (command[i] == '\*')

result \*= Imme(command + i + 1, up, p);

else if (command[i] == '/')

result /= Imme(command + i + 1, up, p);

\*p += i;

if (result < (int)(0x00000001 << (up - 1)))

return result;

else

{

Error(4);

return 0;

}

}

else

{

Error(4);

return 0;

}

}

return 0;

}

uint HexNum(char\*command, uint up, uint \*p)

{

int i = 0;

uint result = 0;

while ((command[i] >= '0' && command[i] <= '9')

|| (command[i] >= 'a' && command[i] <= 'f')

|| (command[i] >= 'A' && command[i] <= 'F'))

{

result \*= 16;

if (command[i] >= '0' && command[i] <= '9')

result += command[i] - '0' + 0;

else if (command[i] >= 'a' && command[i] <= 'f')

result += command[i] - 'a' + 10;

else result += command[i] - 'A' + 10;

i++;

}

if (command[i] == '+')

result += Imme(command + i + 1, up, p);

if (command[i] == '-')

result -= Imme(command + i + 1, up, p);

if (command[i] == '\*')

result \*= Imme(command + i + 1, up, p);

if (command[i] == '/')

result /= Imme(command + i + 1, up, p);

if (result < (int)(0x00000001 << (up - 1)))

{

\*p += i + 2;

return result;

}

else

{

Error(4);

return 0;

}

}

uint BinNum(char\*command, uint up, uint \*p)

{

int i = 0;

uint result = 0;

while (command[i] == '0' || command[i] == '1')

{

result \*= 2;

result += command[i] - '0' + 0;

i++;

}

if (command[i] == '+')

result += Imme(command + i + 1, up, p);

if (command[i] == '-')

result -= Imme(command + i + 1, up, p);

if (command[i] == '\*')

result \*= Imme(command + i + 1, up, p);

if (command[i] == '/')

result /= Imme(command + i + 1, up, p);

if (result < (int)(0x00000001 << (up - 1)))

{

\*p += i + 2;

return result;

}

else

{

Error(4);

return 0;

}

}

void Output(uint Command)

{

uint check = 0x80000000;

while (check)

{

if (Command&check)

printf("1");

else printf("0");

check >>= 1;

}

printf("\n");

}

void TypeImmeAddr(char\* command, uint Op)

{

int i = 0;

uint record = 0;

if (Op == LW || Op == SW)

{

while (command[i] != '$')i++;//$rt

rt = CheckReg(command + i); //rt:?-(7~15)

i = i + 3;//,

while (command[i] != ',')i++;

i++;

while (command[i] == ' ')i++;//num

imme = Imme(command + i, 16, &record);

i += record;

while (command[i] != '$')i++;//rs

rs = CheckReg(command + i); //rs:0-(6~10)

i += 3;

}

else if (Op == LUI)

{

while (command[i] != '$')i++;//$rt

rt = CheckReg(command + i); //rt:?-(7~15)

i = i + 3;//,

while (command[i] != ',')i++;

i++;

while (command[i] == ' ')i++;//num

imme = Imme(command + i, 16, &record);

i += record;

}

else

{

while (command[i] != '$')i++;//$rt

rt = CheckReg(command + i); //rt:?-(7~15)

i = i + 3;//,

while (command[i] != ',')i++;

i++;

while (command[i] != '$')i++;//rs

rs = CheckReg(command + i); //rs:0-(6~10)

while (command[i] != ',')i++;

i++;

while (command[i] == ' ')i++;//0x1234

imme = Imme(command + i, 16, &record);

i += record;

}

}

void TypeThreeReg(char\* command)

{

int i = 0;

while (command[i] != '$')i++;//$rd

rd = CheckReg(command + i); //rt:?-(7~15)

i = i + 3;//,

while (command[i] != ',')i++;

i++;

while (command[i] != '$')i++;//$rs

rs = CheckReg(command + i); //rt:?-(7~15)

i = i + 3;//,

while (command[i] != ',')i++;

i++;

while (command[i] != '$')i++;//$rt

rt = CheckReg(command + i); //rt:?-(7~15)

}

void TypeTwoRegImme(char\* command)

{

int i = 0;

uint record = 0;

while (command[i] != '$')i++;//$rt

rt = CheckReg(command + i);

i = i + 3;//,

while (command[i] != ',')i++;

i++;

while (command[i] != '$')i++;//$rs

rs = CheckReg(command + i);

i = i + 3;//,

while (command[i] != ',')i++;

i++;

while (command[i] == ' ')i++;//0x1234

imme = Imme(command + i, 5, &record);

i += record;

}

void TypeOneReg(char\* command)

{

int i = 0;

while (command[i] != '$')i++;//$rs

rs = CheckReg(command + i);

i = i + 3;//,

}

void TypeTwoReg(char\* command)

{

int i = 0;

while (command[i] != '$')i++;//$rs

rs = CheckReg(command + i);

i = i + 3;//,

while (command[i] != ',')i++;

i++;

while (command[i] != '$')i++;//$rt

rt = CheckReg(command + i);

i = i + 3;//,

}

void TypeJumpAddr(char\* command)

{

int i = 0;

uint record = 0;

uint num = 0;

while (command[i] == ' ')i++;//0x1234

imme = Imme(command + i, 16, &record);

i += record;

}

void WriteImmeAddr(char\* command, uint Op)

{

WriteNum(Command, 0, 5, Op); //15-(0~5) 操作符号

if (Op == LW || Op == SW)

{

WriteNum(Command, 11, 15, rt); //rt:?-(7~15)

WriteNum(Command, 16, 31, imme); //imme:?-(16~31)

WriteNum(Command, 6, 10, rs); //rs:0-(6~10)

}

else if (Op == LUI)

{

WriteNum(Command, 11, 15, rt); //rt:?-(7~15)

WriteNum(Command, 16, 31, imme); //imme:?-(16~31)

}

else

{

WriteNum(Command, 11, 15, rt); //rt:?-(7~15)

WriteNum(Command, 6, 10, rs); //rs:0-(6~10)

WriteNum(Command, 16, 31, imme); //imme:?-(16~31)

}

}

void WriteThreeReg(char\* command, uint Op)

{

WriteNum(Command, 16, 20, rd); //rt:?-(7~15)

WriteNum(Command, 6, 10, rs); //rt:?-(7~15)

WriteNum(Command, 11, 15, rt); //rt:?-(7~15)

WriteNum(Command, 26, 31, Op);

}

void WriteTwoRegImme(char\* command, uint Op)

{

WriteNum(Command, 26, 31, Op); //?-(0~5) 操作符号

WriteNum(Command, 6, 10, rs);

WriteNum(Command, 11, 15, rt);

WriteNum(Command, 21, 25, imme); //imme:?-(16~31)

}

void WriteOneReg(char\* command, uint reg, uint Op)

{

if (reg == 1)

WriteNum(Command, 6, 10, rs);

if (reg == 2)

WriteNum(Command, 11, 15, rt);

if (reg == 3)

WriteNum(Command, 16, 20, rd);

WriteNum(Command, 26, 31, Op);

}

void WriteTwoReg(char\* command, uint Op)

{

WriteNum(Command, 26, 31, Op); //?-(0~5) 操作符号

WriteNum(Command, 6, 10, rs);

WriteNum(Command, 11, 15, rt);

}

void WriteJumpAddr(char\* command, uint Op)

{

WriteNum(Command, 6, 31, imme);

WriteNum(Command, 0, 6, Op);

}

void Error(uint num)

{

switch (num)

{

case 1:printf("Error num to get num\n"); break;

case 2:printf("Error num to write num\n"); break;

case 3:printf("Error wrong dec num for immediate num\n"); break;

case 4:printf("Error num overflow \n"); break;

case 5:printf("You input a wrong number, please input again! \n"); break;

default:

break;

}

}