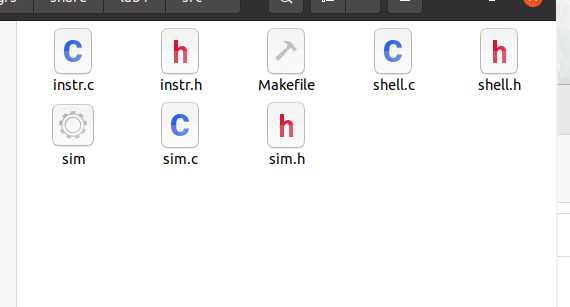
# Lab1仿真实验

## 第一部分：实验分析及执行



详情请见：[https://github.com/FufuWei/CA.git](https://github.com/FufuWei/CA)

### 1.addiu.s

.data

.text

.globl main

main:

addiu $v0, $zero, 10 #将寄存器v0的值设置为10

addiu $t0, $zero, 5#将寄存器t0的值设置为5

addiu $t1, $t0, 300#将寄存器t0的值和300相加，并将结果存入寄存器t1 addiu $t2, $zero, 500#将寄存器t2的值设置为500

addiu $t3, $t2, 34#将寄存器t2的值和34相加，并将结果存入寄存器t3

addiu $t3, $t3, 45#将寄存器t3的值和45相加，并将结果存入寄存器t3

Syscall

在src/中，有模拟器文件和用于编译它们的Makefile，

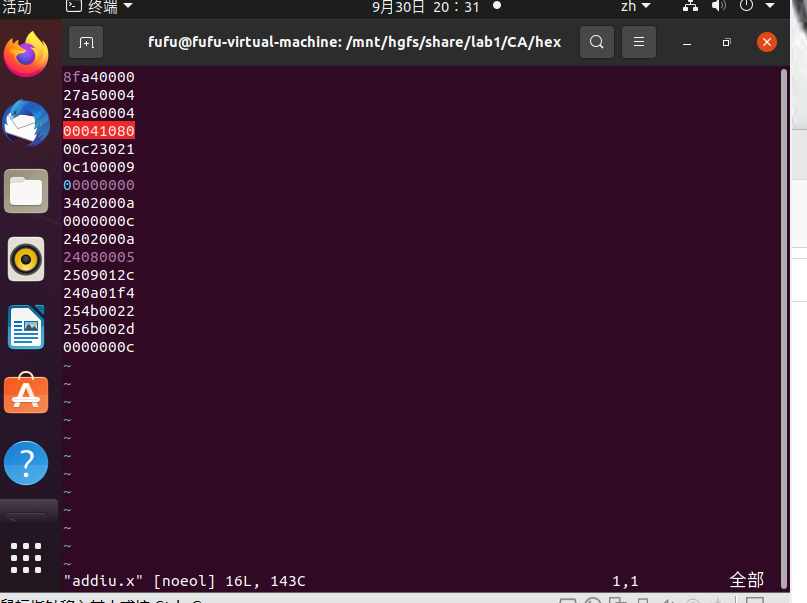
$ cd src/

$ make

inputs/包含用于测试模拟器的输入文件，使用Qtspim模拟器来将它们转换为十六进制代码。

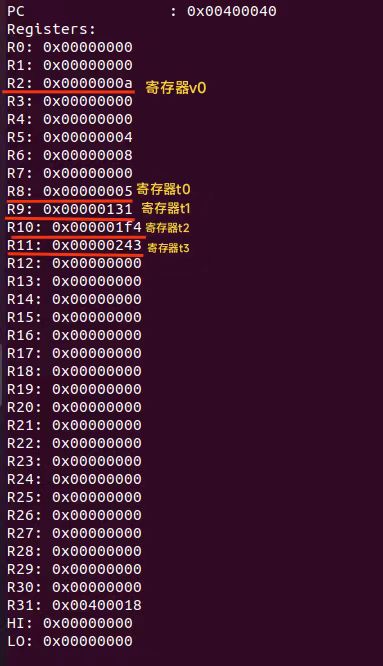
使用Qtspim模拟器来将它们转换为十六进制代码.x

src/sim inputs/hex/addiu.x



MIPS-SIM> go

MIPS-SIM> rdump



V0寄存器十进制为10，十六进制为a;

T0寄存器十进制为5，十六进制为5;

V1寄存器十进制为305，十六进制为131;

V2寄存器十进制为500，十六进制为1f4;

V3寄存器十进制为579，十六进制为243;

2.arithtest.s**(基本算术指令测试)**src/sim inputs/hex/arithtest.x

# Basic arithmetic instructions

# This is a hodgepodge of arithmetic instructions to test

# your basic functionality.

# No overflow exceptions should occur

.data

.text

.globl main

main:

addiu $2, $zero, 1024 # 将寄存器$2设置为1024

addu $3, $2, $2 # 将寄存器$3设置为$2的两倍

or $4, $3, $2 # 将寄存器$4设置为$3和$2的按位或结果

add $5, $zero, 1234 # 将寄存器$5设置为1234

sll $6, $5, 16 # 将寄存器$6设置为$5左移16位的结果

addiu $7, $6, 9999 # 将寄存器$7设置为$6加9999的结果

subu $8, $7, $2 # 将寄存器$8设置为$7减去$2的结果

xor $9, $4, $3 # 将寄存器$9设置为$4和$3的按位异或结果

xori $10, $2, 255 # 将寄存器$10设置为$2和255的按位异或结果

srl $11, $6, 5 # 将寄存器$11设置为$6右移5位的结果

sra $12, $6, 4 # 将寄存器$12设置为$6算术右移4位的结果

and $13, $11, $5 # 将寄存器$13设置为$11和$5的按位与结果

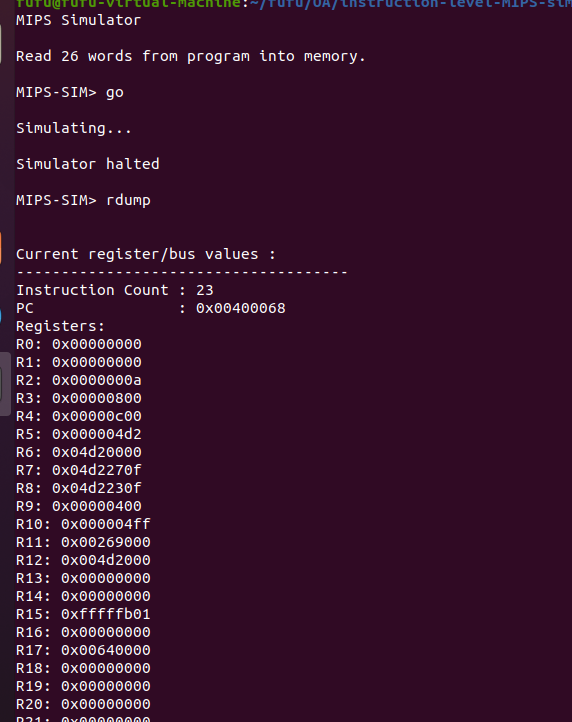
andi $14, $4, 100 # 将寄存器$14设置为$4和100的按位与结果

sub $15, $zero, $10 # 将寄存器$15设置为0减去$10的结果

lui $17, 100 # 将寄存器$17的高16位设置为100

addiu $v0, $zero, 0xa # 将寄存器$v0设置为10

syscall # 执行系统调用



3.Brtest0.s(分支跳转指令测试) src/sim inputs/hex/brtest0.x

# Basic branch test

.data

.text

.globl main

main:

addiu $v0, $zero, 0xa # 将寄存器$v0的值设置为10

l\_0:

addiu $5, $zero, 1 # 将寄存器$5的值设置为1

j l\_1 # 无条件跳转到标签l\_1处

addiu $10, $10, 0xf00 # 将寄存器$10的值加上0xf00

ori $0, $0, 0 # 将寄存器$0的值与0相或，结果仍为0

ori $0, $0, 0 # 将寄存器$0的值与0相或，结果仍为0

addiu $5, $zero, 100 # 将寄存器$5的值设置为100

syscall # 系统调用

l\_1:

bne $zero, $zero, l\_3 # 如果$zero不等于$zero，则跳转到标签l\_3处（永远不会发生）

ori $0, $0, 0 # 将寄存器$0的值与0相或，结果仍为0

ori $0, $0, 0 # 将寄存器$0的值与0相或，结果仍为0

addiu $6, $zero, 0x1337 # 将寄存器$6的值设置为0x1337

l\_2:

beq $zero, $zero, l\_4 # 如果$zero等于$zero，则跳转到标签l\_4处（永远会发生）

ori $0, $0, 0 # 将寄存器$0的值与0相或，结果仍为0

ori $0, $0, 0 # 将寄存器$0的值与0相或，结果仍为0

# 不应该到达这里

addiu $7, $zero, 0x347 # 将寄存器$7的值设置为0x347

syscall # 系统调用

l\_3:

# 不应该到达这里

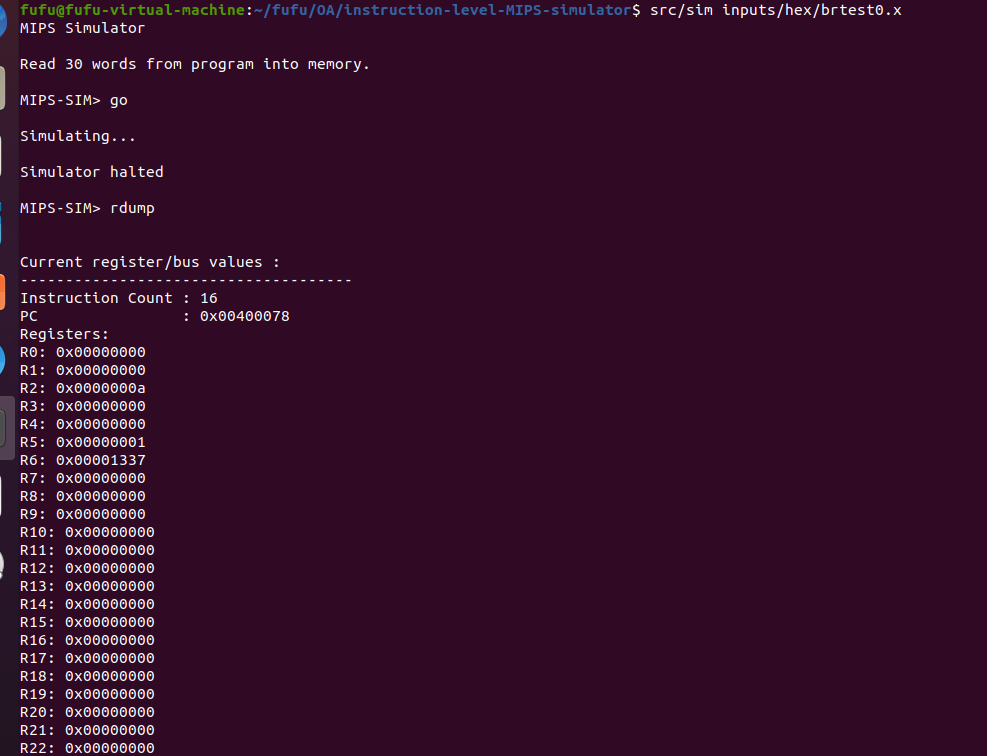
addiu $8, $zero, 0x347 # 将寄存器$8的值设置为0x347

syscall # 系统调用

l\_4:

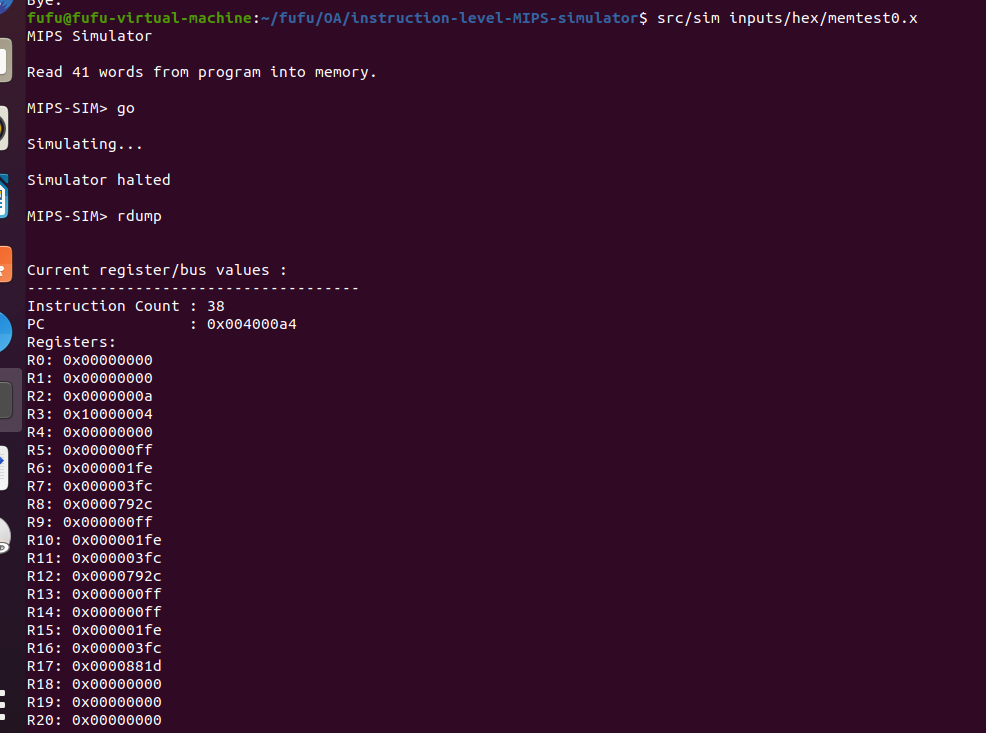
addiu $2, $zero, 10 # 将寄存器$2的值设置为10

syscall # 系统调用



同理我们分析src/sim inputs/hex/brtest1.x, src/sim inputs/hex/brtest2.x。

### 4.Load/Store测试（src/sim inputs/hex/memtest0.x）



同理分析src/sim inputs/hex/memtest1.x。

## 第二部分：思路及代码

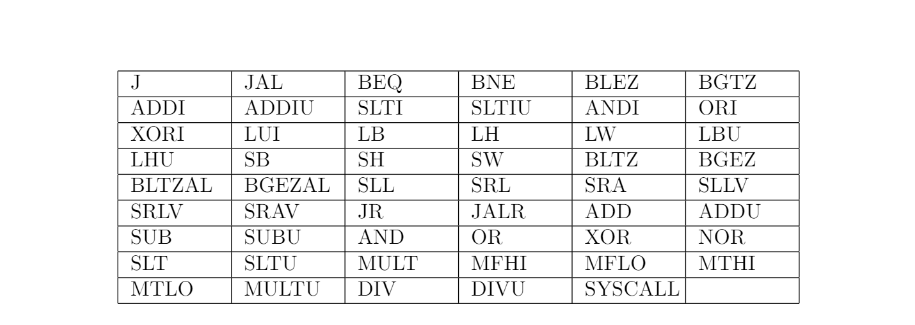
1. **要求：**

在sim.c文件中实现process\_instruction()函数。process\_instruction()函数应能够模拟以下MIPS指令的指令级执行：

* add：将两个寄存器的值相加并将结果存入目标寄存器。
* sub：将两个寄存器的值相减并将结果存入目标寄存器。
* lw：从内存中加载一个字（4个字节）到寄存器中。
* sw：将寄存器的值存入内存中。
* beq：如果两个寄存器的值相等，则跳转到指定的地址。
* j：无条件跳转到指定的地址。

在process\_instruction()函数中实现这些指令的模拟执行，编写代码来模拟这些指令的操作，并根据指令的操作码和操作数执行相应的操作。

如对于add指令，从指令中提取操作数和目标寄存器的编号，然后将两个操作数相加，并将结果存入目标寄存器；lw指令，从指令中提取内存地址和目标寄存器的编号，然后从内存中加载相应的数据，并将其存入目标寄存器。



### 2.Shell命令

1、go：模拟程序执行，直到遇到一个SYSCALL指令，并且该指令中的$v0寄存器（寄存器2）的值等于0x0A，此时模拟器将停止运行。

2、run <n>：模拟器执行n条指令。

3、mdump <low> <high>：将内存中从低地址low到高地址high的内容输出到屏幕和dumpsim文件。

4、rdump：将当前指令计数器、R0-R31寄存器的内容以及PC寄存器的值输出到屏幕和dumpsim文件。

5、input reg\_num reg\_val：将通用寄存器reg\_num的值设置为reg\_val。

6、high value：将HI寄存器的值设置为value。

7、low value：将LO寄存器的值设置为value。

8、?：打印出所有shell命令的列表。

9、quit：退出shell。

### 3.Sim.c

①首先，它会从内存中读取一个32位的指令，并将其存储在变量instruction中。然后，它会从instruction中提取出操作码（op）、源寄存器（rs）、目标寄存器（rt）、目的寄存器（rd）、移位量（shamt）、函数码（funct）、立即数（imm）和目标地址（target）。

接下来，根据不同的操作码，程序会执行不同的操作。如果指令为0，即nop指令，那么将执行nop函数。如果操作码为0，表示为R-Type指令，将执行rtype函数，参数为rs、rt、rd、shamt和funct。如果操作码为1，表示为特殊分支指令，将执行itype\_sbranches函数，参数为rs、rt和imm。如果操作码为2或3，表示为J-Type指令，将执行jtype函数，参数为op和target。否则，将执行itype函数，参数为op、rs、rt和imm。

最后，程序会将NEXT\_STATE.PC增加4，以便执行下一条指令。

void process\_instruction() {

uint32\_t instruction = 0x0;

instruction = mem\_read\_32(CURRENT\_STATE.PC);

uint8\_t op = 0;

uint8\_t rs = 0, rt = 0, rd = 0, shamt = 0, funct = 0;

uint16\_t imm = 0;

uint32\_t target = 0;

op = ((instruction >> 26) & 0x3F);

rs = ((instruction >> 21) & 0x1F);

rt = ((instruction >> 16) & 0x1F);

rd = ((instruction >> 11) & 0x1F);

shamt = ((instruction >> 6) & 0x1F);

funct = (instruction & 0x3F);

imm = instruction;

target = instruction & 0x03FFFFFF;

if(instruction == 0)

nop();

else if(op == 0) // R-Type Instructions.

rtype(rs, rt, rd, shamt, funct);

else if(op == 1) // Special Branches.

itype\_sbranches(rs, rt, imm);

else if(op == 2 || op == 3)

jtype(op, target);

else

itype(op, rs, rt, imm);

NEXT\_STATE.PC += 4;

}

②函数名为rtype，参数类型为uint8\_t，参数名分别为rs、rt、rd、shamt和funct。

该函数根据funct的值调用不同的子函数来执行不同的操作。具体如下：

如果funct小于等于7，则调用shifts函数，参数为rt、rd、shamt和funct。

如果funct的值在24到27之间，或者在32到35之间，则调用arithmetic函数，参数为rs、rt、rd和funct。

如果funct的值在36到39之间，则调用logical函数，参数为rs、rt、rd和funct。

如果funct的值等于42或43，则调用conditional函数，参数为rs、rt、rd和funct。

如果funct的值等于8或9，则调用rtype\_jump函数，参数为rs、rd和funct。

如果funct的值在16到19之间，则调用rtype\_lo\_hi函数，参数为rs、rd和funct。

如果funct的值等于12，则调用syscall函数。

如果以上条件都不满足，则什么也不做。

void rtype(uint8\_t rs, uint8\_t rt, uint8\_t rd, uint8\_t shamt, uint8\_t funct) {

if(funct <= 7)

shifts(rt, rd, shamt, funct);

else if( (funct >= 24 && funct <= 27) || (funct >= 32 && funct <= 35) )

arithmetic(rs, rt, rd, funct);

else if(funct >= 36 && funct <= 39)

logical(rs, rt, rd, funct);

else if(funct == 42 || funct == 43)

conditional(rs, rt, rd, funct);

else if(funct == 8 || funct == 9)

rtype\_jump(rs, rd, funct);

else if(funct >= 16 && funct <= 19)

rtype\_lo\_hi(rs, rd, funct);

else if(funct == 12)

syscall();

else

;

}

③定义了一个名为itype的函数，该函数接受四个参数：op、rs、rt和imm，其中op和rt的类型是uint8\_t，imm的类型是int16\_t。

函数根据op的值来执行不同的操作：

如果op的值在4到7之间，则调用itype\_branches函数，传递参数op、rs、rt和imm。

如果op的值在8到15之间，则调用itype\_arithmetic函数，传递参数op、rs、rt和imm。

如果op的值在32到43之间，则调用mem\_load\_store函数，传递参数op、rs、rt和imm。

如果以上条件都不满足，则什么也不做。

void itype(uint8\_t op, uint8\_t rs, uint8\_t rt, int16\_t imm) {

if(op >= 4 && op <= 7)

itype\_branches(op, rs, rt, imm);

else if(op >= 8 && op <= 15)

itype\_arithmetic(op, rs, rt, imm);

else if(op >= 32 && op <= 43)

mem\_load\_store(op, rs, rt, imm);

else

;

}

④定义了一个名为jtype的函数，该函数接受两个参数：op和target，其中op的类型是uint8\_t，target的类型是uint32\_t。

函数根据op的值来执行不同的操作：

如果op的值等于J，则调用j函数，传递参数target，并立即返回。

如果op的值等于JAL，则调用jal函数，传递参数target，并立即返回。

void jtype(uint8\_t op, uint32\_t target) {

switch(op) {

case J:

j(target);

return;

case JAL:

jal(target);

return;

}

}

⑤定义了一个名为shifts的函数，该函数接受四个参数：rt、rd、shamt和funct，其中rt、rd、shamt和funct的类型都是uint8\_t。

函数根据funct的值来执行不同的操作：

如果funct的值等于SLL，则调用sll函数，传递参数rt、rd和shamt，并立即返回。

如果funct的值等于SRL，则调用srl函数，传递参数rt、rd和shamt，并立即返回。

如果funct的值等于SRA，则调用sra函数，传递参数rt、rd和shamt，并立即返回。

如果funct的值等于SLLV，则调用sllv函数，传递参数rt、rd和shamt，并立即返回。

如果funct的值等于SRLV，则调用srlv函数，传递参数rt、rd和shamt，并立即返回。

如果funct的值等于SRAV，则调用srav函数，传递参数rt、rd和shamt，并立即返回。

void shifts(uint8\_t rt, uint8\_t rd, uint8\_t shamt, uint8\_t funct) {

switch(funct) {

case SLL:

sll(rt, rd, shamt);

return;

case SRL:

srl(rt, rd, shamt);

return;

case SRA:

sra(rt, rd, shamt);

return;

case SLLV:

sllv(rt, rd, shamt);

return;

case SRLV:

srlv(rt, rd, shamt);

return;

case SRAV:

srav(rt, rd, shamt);

return;

}

}

⑥定义了一个名为arithmetic的函数，该函数接受四个参数：rs、rt、rd和funct，其中rs、rt、rd和funct的类型都是uint8\_t。

函数根据funct的值来执行不同的操作：

如果funct的值等于ADD，则调用add函数，传递参数rs、rt和rd，并立即返回。

如果funct的值等于ADDU，则调用addu函数，传递参数rs、rt和rd，并立即返回。

如果funct的值等于SUB，则调用sub函数，传递参数rs、rt和rd，并立即返回。

如果funct的值等于SUBU，则调用subu函数，传递参数rs、rt和rd，并立即返回。

如果funct的值等于MULT，则调用mult函数，传递参数rs和rt，并立即返回。

如果funct的值等于MULTU，则调用multu函数，传递参数rs和rt，并立即返回。

如果funct的值等于DIV，则调用div函数，传递参数rs和rt，并立即返回。

如果funct的值等于DIVU，则调用divu函数，传递参数rs和rt，并立即返回。

void arithmetic(uint8\_t rs, uint8\_t rt, uint8\_t rd, uint8\_t funct) {

switch(funct) {

case ADD:

add(rs, rt, rd);

return;

case ADDU:

addu(rs, rt, rd);

return;

case SUB:

sub(rs, rt, rd);

return;

case SUBU:

subu(rs, rt, rd);

return;

case MULT:

mult(rs, rt);

return;

case MULTU:

multu(rs, rt);

return;

case DIV:

div(rs, rt);

return;

case DIVU:

divu(rs, rt);

return;

}

}

⑦定义了一个名为logical的函数，该函数接受四个参数：rs、rt、rd和funct，其中rs、rt、rd和funct的类型都是uint8\_t。

函数根据funct的值来执行不同的操作：

如果funct的值等于AND，则调用and函数，传递参数rs、rt和rd，并立即返回。

如果funct的值等于OR，则调用or函数，传递参数rs、rt和rd，并立即返回。

如果funct的值等于XOR，则调用xor函数，传递参数rs、rt和rd，并立即返回。

如果funct的值等于NOR，则调用nor函数，传递参数rs、rt和rd，并立即返回。

void logical(uint8\_t rs, uint8\_t rt, uint8\_t rd, uint8\_t funct) {

switch(funct) {

case AND:

and(rs, rt, rd);

return;

case OR:

or(rs, rt, rd);

return;

case XOR:

xor(rs, rt, rd);

return;

case NOR:

nor(rs, rt, rd);

return;

}

}

⑧定义了一个名为conditional的函数，该函数接受四个参数：rs、rt、rd和funct，其中rs、rt、rd和funct的类型都是uint8\_t。

函数根据funct的值来执行不同的操作：

如果funct的值等于SLT，则调用slt函数，传递参数rs、rt和rd，并立即返回。

如果funct的值等于SLTU，则调用sltu函数，传递参数rs、rt和rd，并立即返回。

void conditional(uint8\_t rs, uint8\_t rt, uint8\_t rd, uint8\_t funct) {

switch(funct) {

case SLT:

slt(rs, rt, rd);

return;

case SLTU:

sltu(rs, rt, rd);

return;

}

}

⑨定义了一个名为rtype\_jump的函数，该函数接受三个参数：rs、rd和funct，其中rs和rd的类型都是uint8\_t，而funct的类型也是uint8\_t。

函数根据funct的值来执行不同的操作：

如果funct的值等于JR，则调用jr函数，传递参数rs，并立即返回。

如果funct的值等于JALR，则调用jalr函数，传递参数rs和rd，并立即返回。

void rtype\_jump(uint8\_t rs, uint8\_t rd, uint8\_t funct) {

switch(funct) {

case JR:

jr(rs);

return;

case JALR:

jalr(rs, rd);

return;

}

}

⑩定义了一个名为rtype\_lo\_hi的函数，该函数接受三个参数：rs、rd和funct，其中rs和rd的类型都是uint8\_t，而funct的类型也是uint8\_t。

函数根据funct的值来执行不同的操作：

如果funct的值等于MFHI，则调用mfhi函数，传递参数rd，并立即返回。

如果funct的值等于MFLO，则调用mflo函数，传递参数rd，并立即返回。

如果funct的值等于MTHI，则调用mthi函数，传递参数rs，并立即返回。

如果funct的值等于MTLO，则调用mtlo函数，传递参数rs，并立即返回。

void rtype\_lo\_hi(uint8\_t rs, uint8\_t rd, uint8\_t funct) {

switch(funct) {

case MFHI:

mfhi(rd);

return;

case MFLO:

mflo(rd);

return;

case MTHI:

mthi(rs);

return;

case MTLO:

mtlo(rs);

return;

}

}

11.定义了一个名为itype\_sbranches的函数，该函数接受三个参数：rs、rt和imm，其中rs和rt的类型都是uint8\_t，而imm的类型是int16\_t。

函数根据rt的值来执行不同的操作：

如果rt的值等于BLTZ，则调用bltz函数，传递参数rs和imm，并立即返回。

如果rt的值等于BGEZ，则调用bgez函数，传递参数rs和imm，并立即返回。

如果rt的值等于BLTZAL，则调用bltzal函数，传递参数rs和imm，并立即返回。

如果rt的值等于BGEZAL，则调用bgezal函数，传递参数rs和imm，并立即返回。

void itype\_sbranches(uint8\_t rs, uint8\_t rt, int16\_t imm) {

switch(rt) {

case BLTZ:

bltz(rs, imm);

return;

case BGEZ:

bgez(rs, imm);

return;

case BLTZAL:

bltzal(rs, imm);

return;

case BGEZAL:

bgezal(rs, imm);

return;

}

}

12.定义了一个名为itype\_arithmetic的函数，该函数接受四个参数：op、rs、rt和imm，其中op、rs和rt的类型都是uint8\_t，而imm的类型是int16\_t。

函数根据op的值来执行不同的操作：

如果op的值等于ADDI，则调用addi函数，传递参数rs、rt和imm，并立即返回。

如果op的值等于ADDIU，则调用addiu函数，传递参数rs、rt和imm，并立即返回。

如果op的值等于SLTI，则调用slti函数，传递参数rs、rt和imm，并立即返回。

如果op的值等于SLTIU，则调用sltiu函数，传递参数rs、rt和imm，并立即返回。

如果op的值等于ANDI，则调用andi函数，传递参数rs、rt和imm，并立即返回。

如果op的值等于ORI，则调用ori函数，传递参数rs、rt和imm，并立即返回。

如果op的值等于XORI，则调用xori函数，传递参数rs、rt和imm，并立即返回。

如果op的值等于LUI，则调用lui函数，传递参数rt和imm，并立即返回。

void itype\_arithmetic(uint8\_t op, uint8\_t rs, uint8\_t rt, int16\_t imm) {

switch(op) {

case ADDI:

addi(rs, rt, imm);

return;

case ADDIU:

addiu(rs, rt, imm);

return;

case SLTI:

slti(rs, rt, imm);

return;

case SLTIU:

sltiu(rs, rt, imm);

return;

case ANDI:

andi(rs, rt, imm);

return;

case ORI:

ori(rs, rt, imm);

return;

case XORI:

xori(rs, rt, imm);

return;

case LUI:

lui(rt, imm);

return;

}

}

13.定义了一个名为itype\_branches的函数，该函数接受四个参数：op、rs、rt和imm，其中op、rs和rt的类型都是uint8\_t，而imm的类型是int16\_t。

函数根据op的值来执行不同的操作：

如果op的值等于BEQ，则调用beq函数，传递参数rs、rt和imm，并立即返回。

如果op的值等于BNE，则调用bne函数，传递参数rs、rt和imm，并立即返回。

如果op的值等于BLEZ，则调用blez函数，传递参数rs和imm，并立即返回。

如果op的值等于BGTZ，则调用bgtz函数，传递参数rs和imm，并立即返回。

void itype\_branches(uint8\_t op, uint8\_t rs, uint8\_t rt, int16\_t imm) {

switch(op) {

case BEQ:

beq(rs, rt, imm);

return;

case BNE:

bne(rs, rt, imm);

return;

case BLEZ:

blez(rs, imm);

return;

case BGTZ:

bgtz(rs, imm);

return;

}

}

14.定义了一个名为mem\_load\_store的函数，该函数接受四个参数：op、base、rt和imm，其中op、base和rt的类型都是uint8\_t，而imm的类型是int16\_t。

函数根据op的值来执行不同的操作：

如果op的值等于LB，则调用lb函数，传递参数base、rt和imm，并立即返回。

如果op的值等于LH，则调用lh函数，传递参数base、rt和imm，并立即返回。

如果op的值等于LW，则调用lw函数，传递参数base、rt和imm，并立即返回。

如果op的值等于LBU，则调用lbu函数，传递参数base、rt和imm，并立即返回。

如果op的值等于LHU，则调用lhu函数，传递参数base、rt和imm，并立即返回。

如果op的值等于SB，则调用sb函数，传递参数base、rt和imm，并立即返回。

如果op的值等于SH，则调用sh函数，传递参数base、rt和imm，并立即返回。

如果op的值等于SW，则调用sw函数，传递参数base、rt和imm，并立即返回。

void mem\_load\_store(uint8\_t op, uint8\_t base, uint8\_t rt, int16\_t imm) {

switch(op) {

case LB:

lb(base, rt, imm);

return;

case LH:

lh(base, rt, imm);

return;

case LW:

lw(base, rt, imm);

return;

case LBU:

lbu(base, rt, imm);

return;

case LHU:

lhu(base, rt, imm);

return;

case SB:

sb(base, rt, imm);

return;

case SH:

sh(base, rt, imm);

return;

case SW:

sw(base, rt, imm);

return;

}

}