汇编器实验报告

班级：智能 1602

学号：201608010722

姓名：孙传骐

实验目标

设计一个 RISC-V 基本整数指令汇编器，能够实现将汇编指令转化为二进制

实验要求

- 编写汇编器代码
- 汇编器的输入是进行模拟的汇编指令
- 汇编器的输出是经过汇编后的二进制指令文件

实验内容

汇编器简介

汇编器是将汇编语言翻译为机器语言的程序。一般而言，汇编生成的是目标代码，需要经链接器生成可执行代码才可以执行
汇编语言是一种以处理器指令系统为基础的低级语言，采用助记符表达指令操作码，采用标识符表示指令操作数。作为一门语言，对应于高级语言的编译器，需要一个汇编器将汇编语言原文件汇编成机器可执行的代码

RISC-V 指令集介绍

RV32I 指令集包含了六种基本指令格式，分别是：

R 类型指令：用于寄存器到寄存器操作

I 类型指令：用于短立即数和访存 load 操作

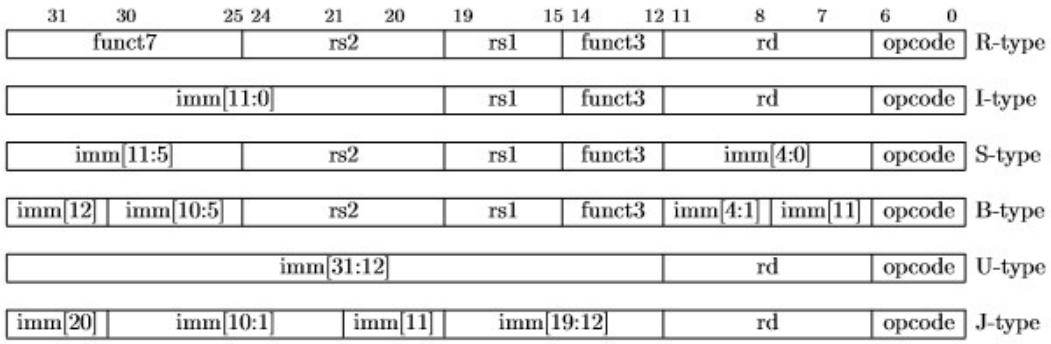
S 类型指令：用于访存 store 操作

B 类型指令：用于条件跳转操作

U 类型指令：用于长立即数

J 类型指令：用于无条件跳转

RISC-V 指令集编码格式



RISC-V 指令

Category Name	Fmt	RV32I Base	
Shifts			
Shift Left Logical	R	SLL	rd,rs1,rs2
Shift Left Log.Imm.	I	SLLI	rd,rs1,shamt
Shift Right Logical	R	SRL	rd,rs1,rs2
Shift Right Log.Imm.	I	SRLI	rd,rs1,shamt
Shift Right Arithmetic	R	SRA	rd,rs1,rs2
Shift Right Arith.Imm.	I	SRAI	rd,rs1,shamt
Arithmetic			
ADD	R	ADD	rd,rs1,rs2
ADD Immediate	I	ADDI	rd,rs1,imm
SUBtract	R	SUB	rd,rs1,rs2
Load Upper Imm	U	LUI	rd,imm
Add Upper Imm to PC	U	AUIPC	rd,imm
Logical			
XOR	R	XOR	rd,rs1,rs2
XOR Immediate	I	XORI	rd,rs1,imm
OR	R	OR	rd,rs1,rs2
OR Immediate	I	ORI	rd,rs1,imm
AND	R	AND	rd,rs1,rs2
AND Immediate	I	ANDI	rd,rs1,imm

Category Name	Fmt	RV32I Base	
Compare			
Set<	R	SLT	rd,rs1,rs2
Set<Immediate	I	SLTI	rd,rs1,rs2
Set<Unsigned	R	SLTU	rd,rs1,rs2
Set<Imm Unsigned	I	SLTIU	rd,rs1,imm
Branches			
Branch=	B	BEQ	rs1,rs2,imm
Branch≠	B	BNE	rs1,rs2,imm
Branch<	B	BLT	rs1,rs2,imm
Branch≥	B	BGE	rs1,rs2,imm
Branch<Unsigned	B	BLTU	rs1,rs2,imm
Branch≥Unsigned	B	BGEU	rs1,rs2,imm
Jump&Link			
J&L	J	JAL	rd,imm
Jump&Link Register	I	JALR	rd,rs1,imm
Synch			
Synch thread	I	FENCE	
Synch Instr&Data	I	FENCE.I	
Environment			
CALL	I	ECALL	
BREAK	I	EBREAK	
Control Status Register(CSR)			
Read/Write	I	CSRRW	rd,csr,rs1
Read&Set Bit	I	CSRRS	rd,csr,rs1
Read&Clear Bit	I	CSRRC	rd,csr,rs1
Read/Write Imm	I	CSRRWI	rd,csr,imm
Read&Set Bit Imm	I	CSRRSI	rd,csr,imm
Read&Clear Bit Imm	I	CSRRCI	rd,csr,imm
Loads			
Load Byte	I	LB	rd,rs1,imm
Load Halfword	I	LH	rd,rs1,imm
Load Byte Unsigned	I	LBU	rd,rs1,imm
Load Half Unsigned	I	LHU	rd,rs1,imm
Load Word	I	LW	rd,rs1,imm
Stores			
Store Byte	S	SB	rs1,rs2,imm
Store Halfword	S	SH	rs1,rs2,imm
Store Word	S	SW	rs1,rs2,imm

汇编器执行过程

> 简而言之就是编写--〉编译--〉连接--〉执行

1. 创建一个源文件。

2. 汇编器读取源文件后生成目标文件。

3. 链接器读取并检查目标文件，以便发现该程序是否包含了任何对链接库中过程的调用。链接器从链接库中复制任何被请求的过程，将它们与目标文件组合，以生成可执行文件。

4. 操作系统加载程序将可执行文件.exe 读入内存，并使 CPU 分支到该程序的起始地址，然后程序开始执行。

汇编器设计框架

整个汇编器的框架设计如下：

```
<标号>: add x1, x2, x3
<标号>: 10101010101101001

// 汇编程序文件 file.asm
// 一行一个汇编语句

// 初始化地址计数器 addr_counter = 0;
 while(file.asm 没有到文件尾) {
  // 读入一行
   while(读入的是纯标号且不是文件尾) { 继续读一行 }

  // 拆开行，得到标号（有可能没有），操作码或者伪指令助记符，操作数

   if(有标号) { //记下标号和当前地址计数器的值，保存到符号表；
                // 查看未决汇编语句是否需要这个标号，并解决
              }

   if(操作码助记符) {
         生成操作码编码;
         操作数 -> 寄存器编号或者立即数
         if(操作数是标号) { 查找符号表，如果查到，计算得到偏移量；
                        如果没查到，记下当前汇编语句和地址
```

```
        }

        生成指令的二进制表示 }
    else(伪指令助记符) { 根据伪指令含义执行相应转换 }

}
```

测试

测试平台

模块	配置
CPU	Core i7-6700
操作系统	Windows10
运行环境	C++

###测试结果

输入： 输入的结果存储在 input.txt 里

```
ADD r3,r1,r2
SUB r3,r1,r2
XOR r3,r1,r2
OR r3,r1,r2
AND r3,r1,r2
SLL r3,r1,r2
SRL r3,r1,r2
SRA r3,r1,r2
SLT r3,r1,r2
SLTU r3,r1,r2
LB r2,r1,10
LH r2,r1,10
LW r2,r1,10
```

```
LBU r2,r1,10
LHU r2,r1,10
ADDI r2,r1,10
SLTI r2,r1,10
SLTIU r2,r1,10
XORI r2,r1,10
ORI r2,r1,10
ADDI r2,r1,10
SLLI r2,r1,10
SRLI r2,r1,10
SRAI r2,r1,10
SB r1,r2,36
SH r1,r2,36
SW r1,r2,36
LUI r1,200
AUIPC r1,200
BEQ r1,r2,200
BNE r1,r2,200
BLT r1,r2,200
BGE r1,r2,200
BLTU r1,r2,200
BGEU r1,r2,200
JAL r1,100
JALR r2,r1,10
```

输出： 输出的结果存储在 output.txt 里

```
00000000001000001000000110110011
01000000001000001000000110110011
00000000001000001100000110110011
00000000001000001110000110110011
00000000001000001111000110110011
00000000001000001001000110110011
00000000001000001101000110110011
01000000001000001101000110110011
00000000001000001010000110110011
00000000001000001011000110110011
00000000101000001000000100000011
00000000101000001001000100000011
00000000101000001010000100000011
00000000101000001100000100000011
00000000101000001101000100000011
00000000101000001000000100010011
00000000101000001010000100010011
00000000101000001011000100010011
```

```
00000000101000001100000100010011
00000000101000001110000100010011
00000000101000001000000100010011
00000000101000001001000100010011
00000000101000001101000100010011
01000000101000001101000100010011
00000010001000001000001000100011
00000010001000001001001000100011
00000010001000001010001000100011
00000000000001100100000010110111
00000000000001100100000010010111
00011000001000001000100001100011
00011000001000001001100001100011
00011000001000001100100001100011
00011000001000001101100001100011
00011000001000001110100001100011
00011000001000001111100001100011
00001100100000000000000011101111
00000110010000001000000101100111
```

结果分析

从结果中可知，汇编代码编译输出了二进制结果，达到了实验的目的。