浙江大学

《计算机系统原理》实验报告

作业名称：汇编器设计

小组成员：

产品经理：

指导老师：楼学庆

2020 年 6 月 30 日

1 引言

本程序可以进行汇编语言与 16 进制的机器码的汇编和反汇编处理。一共可以处理的指令共 66 条，可以处理 45 条基本指令和 21 条伪指令。所有可处理的指令如下表所示。

MIPS 指令	R-type	add,sub,slt,and,or,xor,nor,sll,sllv,srl,srlv,sra,srav,jr,jalr,syscall,addu,subu,
	I-type	lui,addi,sltu,sltiu,andi,ori,xori,lw,lwx,lh,lhx,lhu,lhux,sw,swx,sh,shx,beq, bne,bgezal,addiu,slti,
	J-type	j,jal
	C-type	mfc0,mtc0,erec
伪指令		push,pop,move,shi,shix,inc,dec,not,neg,abs,swap,beqz,bnez,beqi,bnei, blt,bgt,bge,ble,seq,sne

本程序利用 Qt 框架制作了图形界面，用户可以在文本框内输入、编辑指令或机器码，然后点击汇编或反汇编按钮来进行，汇编或反汇编。处理结果会在另一个文本框中显示。

2 原理

2.1 通用寄存器

MIPS 架构有 32 个通用寄存器，将 32 个寄存器按 0-31 编号，编号顺序如下表所示。

Name	Register Number	Usage
$zero	0	始终为 0
$at	1	为汇编保留，主要用于伪指令扩展
$v0-$v1	2-3	子程序返回
$a0-$a3	4-7	子程序调用参数
$t0-$t7	8-15	临时变量寄存器
$s0-$s7	16-23	变量寄存器
$t8-$t9	24-25	更多临时变量
$k0-$k1	26-27	操作系统的错误返回

$gp	28	全局指针
$sp	29	栈指针
$fp	30	帧指针
$ra	31	返回地址寄存器

2.2 指令类型

MIPS 指令为等长指令集，所有指令均为 32 位，其中操作符 6 位，其他位随不同的指令划分为不同的字段。

2.2.1 R 类型指令

寄存器类型指令，操作数均在寄存器中。操作符为 0。其指令功能通过最后 6 位的功能字段决定。

1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0
000000						rs					rt					rd					shamt					function						

2.2.2 I 类型指令

立即数类型指令。指令中有两个寄存器，并包含一个 16 位的立即数，用于不同的寻址方式，或立即数运算。

1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0
OpCode						rs					rt					Immediate																

2.2.3 J 类型指令

特殊类型指令，主要用于程序的无条件转移或者子程序调用。

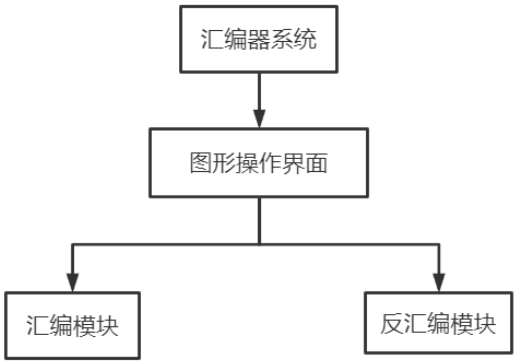
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0
00001x						Target																									

2.2.4 C 类型指令

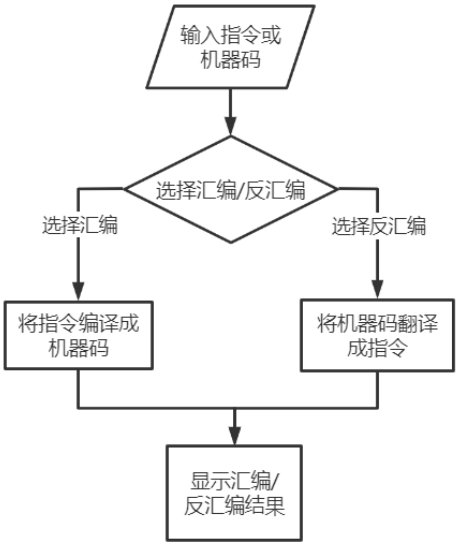
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0
0100xx						Format					ft					fs					fd					function						

3 系统架构

3.1 系统层次图



3.2 程序流程图



4 详细设计

4.1 汇编模块

4.1.1 模块概述

该模块接收用户输入的指令集合，将指令一句句地翻译成 16 进制的机器码，然后将翻译信息返回给图形界面显示。

4.1.2 类设计

```
class Assembler{
public:
    Assembler();
    // 编译指令
    QString Compile(QString& stringNeedCompile);
    QStringList CompleteMachineCode(QString thisLine);
    QStringList CompilePseudo(QStringList& stringList, QString thisLine);
    int Case_contained(QString& Op);

    // 处理字符串
    QStringList Split_enter(QString& stringNeedCompile);
    QString Join_enter(QStringList& rawAssembledStringList);
    QString GetSubstring(QString input, int beginposition);
    // 将数字转化成补码
    QString convertTocomplement(int offset);
    // 在机器码的指定位置填入数字
    void change_s(const QString& rs, QString& to_machinecode);
    void change_t(const QString& rt, QString& to_machinecode);
    void change_d(const QString& rd, QString& to_machinecode);
    void change_a(const QString& sa, QString& to_machinecode);
    void change_i(const QString& imm, QString& to_machinecode);
    void change_labelori(QString& thisLine, const QString& label, QString& to_machinecode);
    void change_target(const QString& target, QString& to_machinecode);

private:
    QString AssembledString;
    QMap<QString, QString> KeyCode;
    QMap<QString, QString> Register;
    QMap<QString, QString> LabelAddr;
    QMap<QString, int> InstructionAddr;

    // 分别存储不同类型的指令
    QStringList STD_Instruction;
    QStringList TDA_Instruction;
    QStringList S___Instruction;
    QStringList STI_Instruction;
    QStringList STII_Instruction;
    QStringList BSTI_Instruction;
    QStringList J_Instruction;
    QStringList S_D_Instruction;
    QStringList TI_Instruction;
    QStringList S_I_Instruction;
```

```
43.     QStringList TD_Instruction;
44.     QStringList SYS_Instruction;
45.     QStringList Pseudo_Instruction;
46.     // 编译的机器码结果
47.     QStringList resultList;
48. };
```

4.1.3 设计方法

在 Assembler 类的构造函数中，KeyCode 和 Register 中存储了指令与操作码格式、寄存器名字与编号的一一对应关系，如图（仅展示部分）：

```
Assembler::Assembler()
{
    KeyCode = QMap<QString, QString>({
        {"add",     "000000ssssstttttddddd00000100000"},
        {"addu",    "000000ssssstttttddddd00000100001"},
        {"and",     "000000ssssstttttddddd00000100100"},
        {"nor",     "000000ssssstttttddddd00000100111"},
        {"or",      "000000ssssstttttddddd00000100101"},
        {"sub",     "000000ssssstttttddddd00000100010"},
        {"subu",    "000000ssssstttttddddd00000100011"},
        {"xor",     "000000ssssstttttddddd00000100110"},
        {"slt",     "000000ssssstttttddddd00000101010"},
        {"sltu",    "000000ssssstttttddddd00000101011"},
        {"sllv",    "000000ssssstttttddddd00000000100"},
        {"srlv",    "000000ssssstttttddddd00000000110"},
        {"srav",    "000000ssssstttttddddd00000000111"},
```

```
Register = QMap<QString,QString>({
    {"$zero","00000"},
    {"$at","00001"},
    {"$v0","00010"},
    {"$v1","00011"},
    {"$a0","00100"},
    {"$a1","00101"},
    {"$a2","00110"},
    {"$a3","00111"},
    {"$t0","01000"},
    {"$t1","01001"},
    {"$t2","01010"},
    {"$t3","01011"},
    {"$t4","01100"},
    {"$t5","01101"},
    {"$t6","01110"},
    {"$t7","01111"},
```

将操作码格式相似的指令存入同一个列表中，如图（仅展示部分）：

```
STD_Instruction << "add" << "addu" << "and" << "nor" << "or" << "sub" << "subu" << "xor" << "
TDA_Instruction << "sll" << "srl" << "sra";
S___Instruction << "jr";
STI_Instruction << "addi" << "addiu" << "andi" << "ori" << "xori" << "slti" << "sltiu";
STII_Instruction << "sw" << "lw" << "lwx" << "lh" << "lhx" << "lhu" << "lhux" << "swx" <<"sh"
J_Instruction << "j" << "jal";
BSTI_Instruction << "beq" << "bne";
```

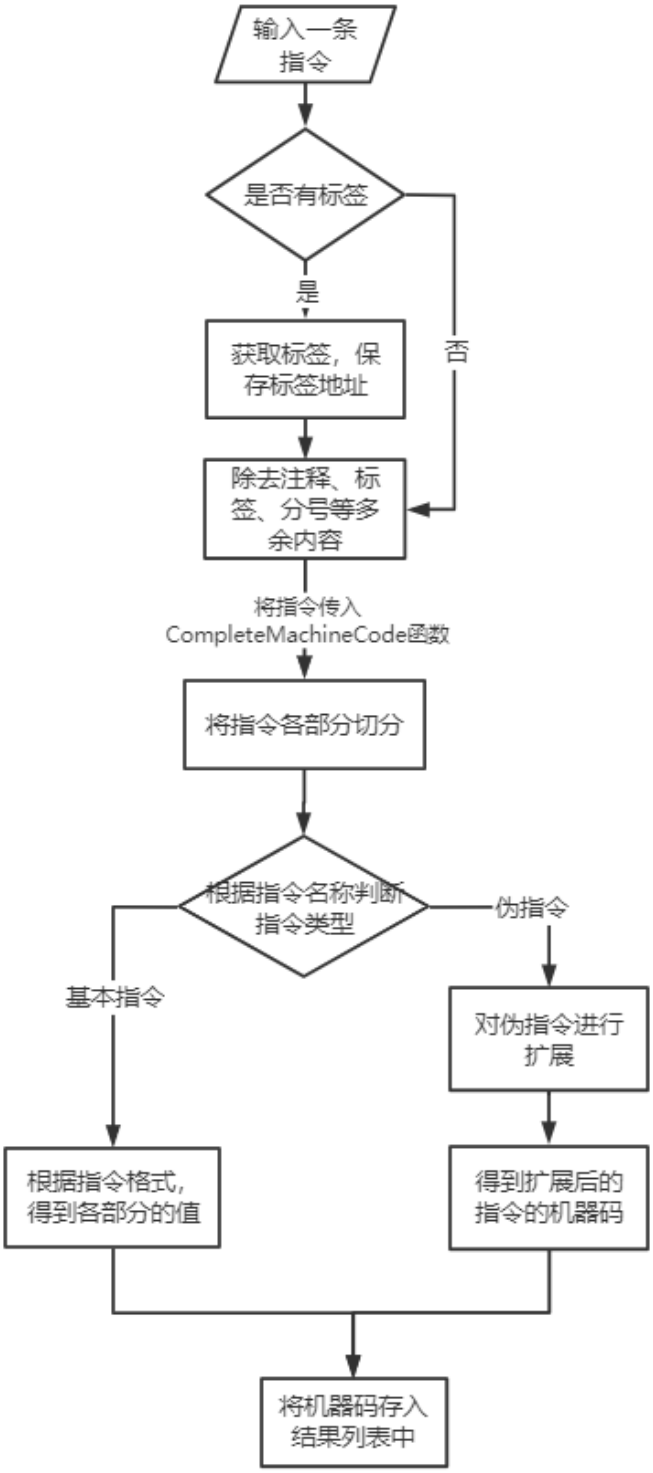
程序运行时调用 Compile 函数，将传入的字符串以'\n'作为分割点进行分割，分成一条条指令依次处理。每次处理一条指令，先检查是否有标签或注释，若有标签，则将该标签和指令地址的对应关系存入 LabelAddr 中。接着将字符串中多余的内容除去，将指令交给 CompleteMachineCode 函数处理。

CompleteMachineCode 函数根据括号、逗号、空格等符号将指令分割，读出每个部分的值。然后根据指令名称，获取指令类型和格式，根据格式得到机器码各部分的值，然后返回结果。如果是一个伪指令，则再调用 CompilePseudo 函数对伪指令进行扩展，再得到机器码，然后返回。

Compile 函数得到 CompleteMachineCode 函数返回的编译的机器码后，将该机器码

添加到结果列表 resultList 中，再继续处理下一条输入的指令，直到指令都处理完毕。同时，每往 resultList 中添加一条指令，指令地址都要增加 4。

4.1.4 流程图



4.2 反汇编模块

4.2.1 模块概述

该模块接收用户输入的机器码集合，将机器码一句句地翻译成 MIPS 指令，然后将翻译信息返回给图形界面显示。

4.2.2 类设计

```
class Disassembler{
public:
    Disassembler();
    // 翻译机器码
    QString Discompile(QString& stringNeedCompile);
    QString InstructionToMipsCode(QString currentLine);
    int Cases_contained(QString& Op);
    // 处理字符串
    QStringList Addn(QString& stringNeedCompile);
    QString Join_e(QStringList& rawAssembledStringList);
    QString GetSubString(QString input, int beginposition, int endposition);

    // 将二进制数转成十进制
    QString convertToFabs(QString Bin);
    QString convertBinToD(QString);
    QString convertBinToDFour(QString BinString);
private:
    QString DisassembledString;

    QMap<QString, QString> KeyOp;
    QMap<QString, QString> Reg;

    // 分别存储不同类型的指令
    QStringList STD_Instruction;
    QStringList TDA_Instruction;
    QStringList S___Instruction;
    QStringList STI_Instruction;
    QStringList STII_Instruction;
    QStringList BSTI_Instruction;
    QStringList J_Instruction;
    QStringList S_D_Instruction;
    QStringList TI_Instruction;
    QStringList S_I_Instruction;
    QStringList TD_Instruction;
    QStringList SYS_Instruction;
    // 翻译的指令结果
```

```
36.     QStringList MipsresultList;
37. };
```

4.2.3 设计方法

在 Disassembler 类的构造函数中，对于有自己的 OpCode 的指令名称，KeyOp 会存储指令码和指令名称的一一对应的关系；如果是没有 OpCode 的指令名称，则存储机器码前六位 OpCode 和末六位 function 与指令名称的一一对应关系，Reg 中存储了编号与寄存器名称的一一对一关系，如图（仅展示部分）：

```
Disassembler::Disassembler()
{
    KeyOp = QMap<QString, QString>({
        {"000000100000",      "add"},
        {"000000100001",     "addu"},
        {"000000100100",      "and"},
        {"000000100111",      "nor"},
        {"000000100101",       "or"},
        {"000000100010",      "sub"},

        {"000000001000",       "jr"},

        {"001000",           "addi"},
        {"001001",          "addiu"},
        {"001100",           "andi"},
        {"001101",            "ori"},
        {"001110",           "xori"},
        {"001010",           "slti"},
        {"001001",          "sltiu"},
```

```
Reg = QMap<QString,QString>({
    {"00000","$zero"},
    {"00001","$at"},
    {"00010","$v0"},
    {"00011","$v1"},
    {"00100","$a0"},
    {"00101","$a1"},
    {"00110","$a2"},
    {"00111","$a3"},
    {"01000","$t0"},
    {"01001","$t1"},
    {"01010","$t2"},
```

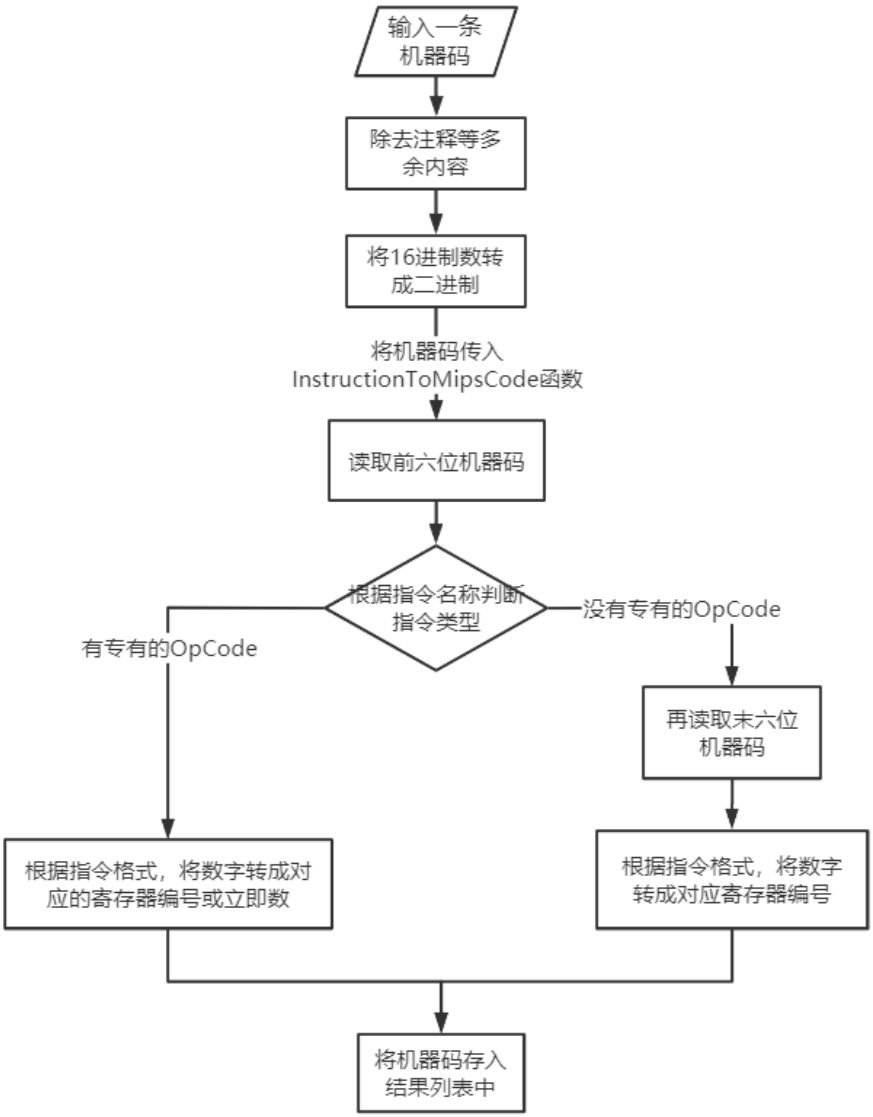
将操作码格式相似的指令存入同一个列表中，如图（仅展示部分）：

```
STD_Instruction << "add" << "addu" << "and" << "nor" << "or" << "sub" << "subu" << "xor" << "
TDA_Instruction << "sll" << "srl" << "sra";
S___Instruction << "jr";
STI_Instruction << "addi" << "addiu" << "andi" << "ori" << "xori" << "slti" << "sltiu";
STII_Instruction << "sw" << "lw" << "lwx" << "lh" << "lhx" << "lhu" << "lhux" << "swx" <<"sh"
J_Instruction << "j" << "jal";
BSTI_Instruction << "beq" << "bne";
```

反汇编时，调用 Disassembler 函数，对于读入的每一条机器码，首先除去注释，然后将 16 进制的机器码转成二进制表示，将二进制的机器码传给函数进行反汇编。

InstructionToMipsCode 函数先根据机器码的前六位，判断是否是有专有的 OpCode 的指令，如果是，则根据 OpCode 的类型，将机器码的各部分数字转成对应的寄存器名称或是立即数；如果不是，则再读取机器码的末六尾，再判断机器码的格式，然后将机器码各部分转成寄存器名称。

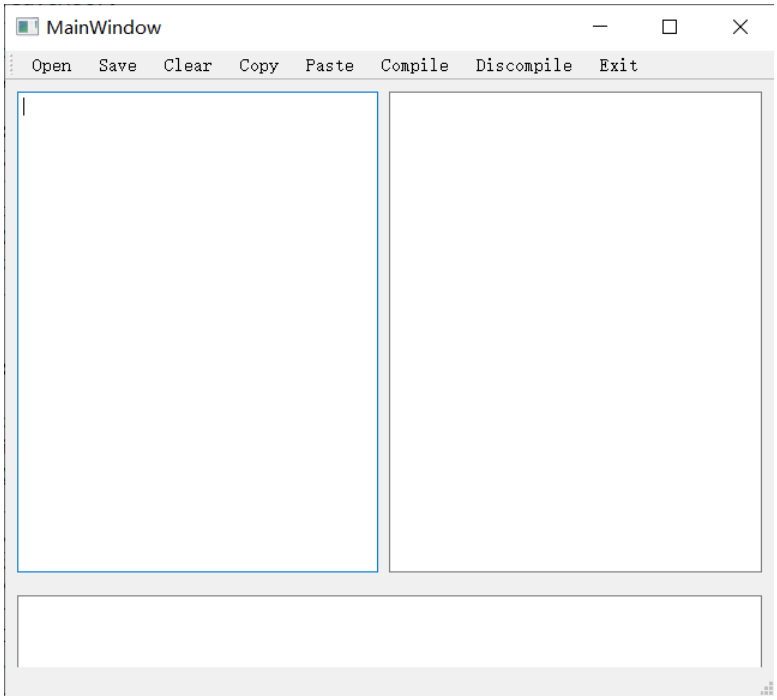
4.2.4 流程图



5 使用手册

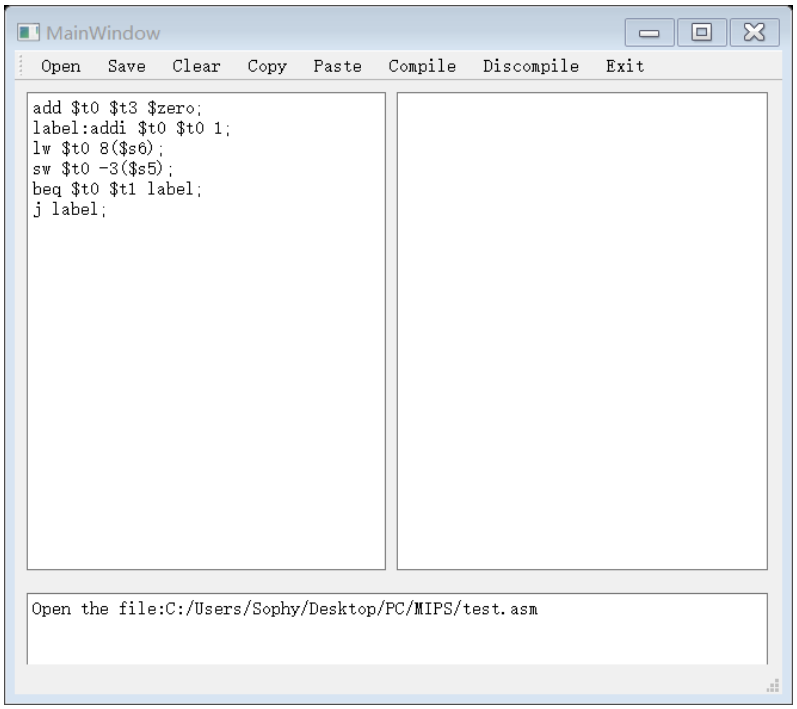
5.1 程序界面

最上方的菜单栏，分别可以执行打开文件、保存输入的指令、清空输入框、复制、粘贴、汇编、反汇编、退出程序的功能。

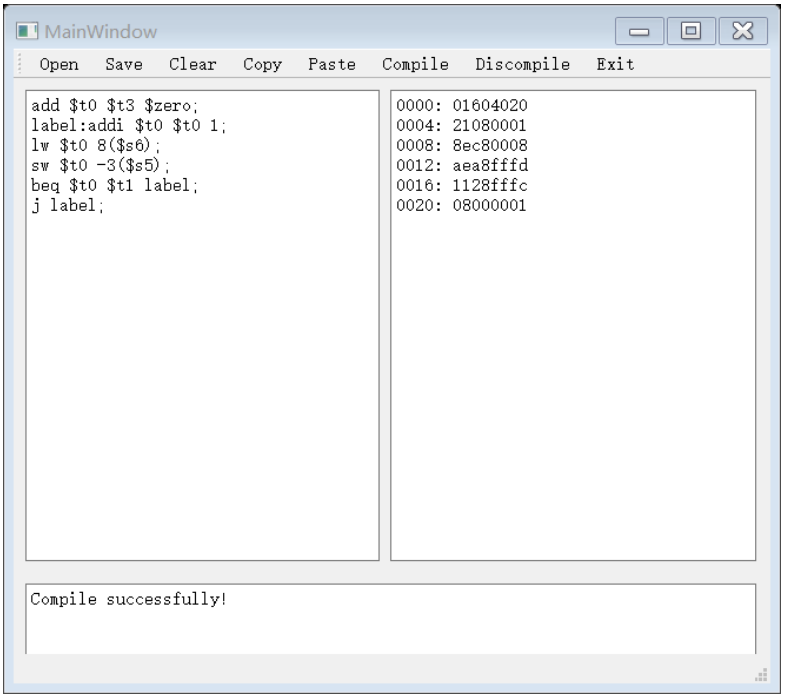


5.2 汇编

在左侧的文本框内输入 MIPS 指令，或是点击“Open”，打开写有 MIPS 指令的文件。下方的文本框会显示操作的结果，如图所示：

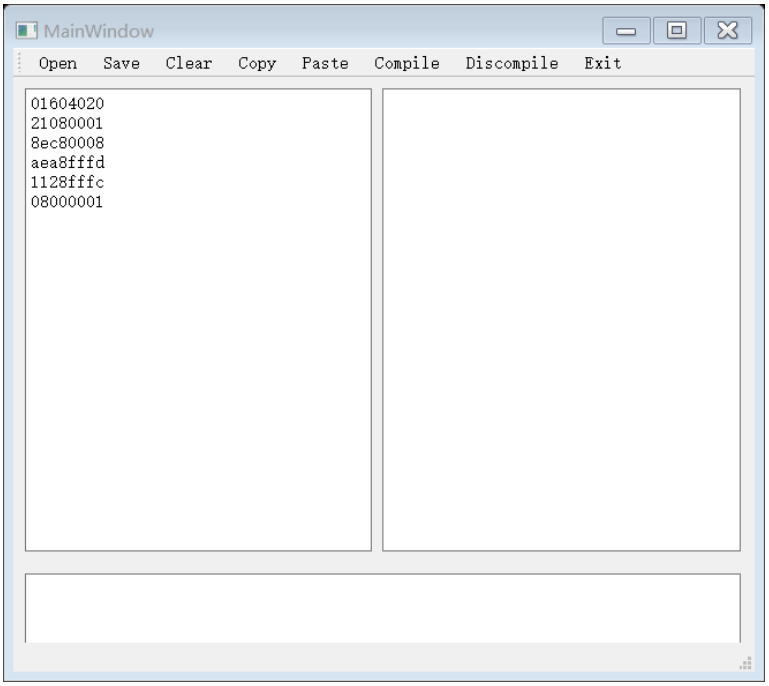


然后点击“Compile”选项，汇编结果将在右侧文本框显示。如图：

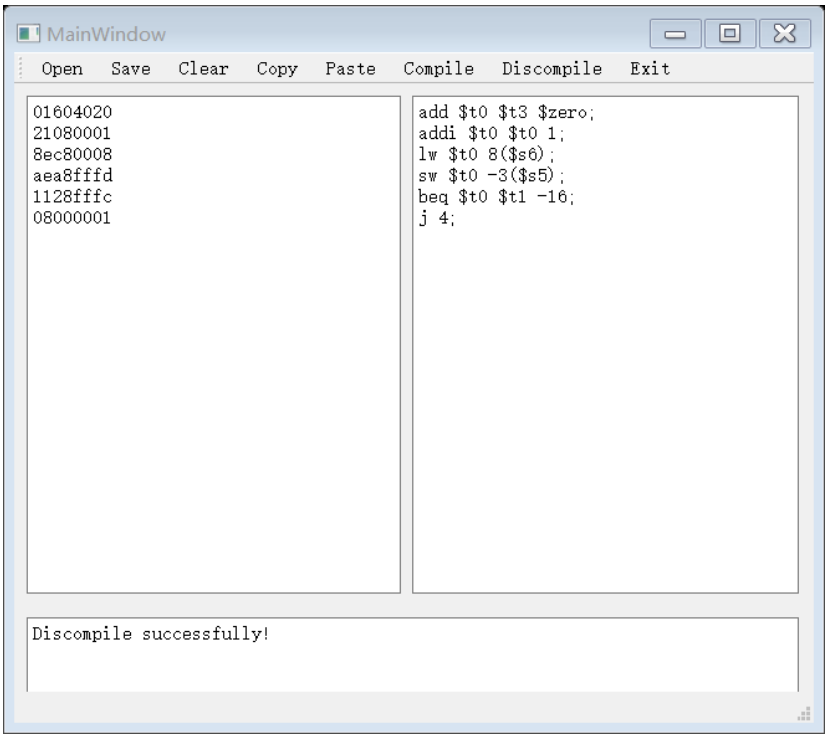


5.3 反汇编

在左侧的文本框内输入机器码，或是点击“Open”，打开写有机器码的文件，如图所示：



然后点击“Discompile”选项，反汇编结果将在右侧文本框显示。如图：


MainWindow
Open
Save
Clear
Copy
Paste
Compile
Discompile
Exit
01604020
21080001
8ec80008
aea8fffd
1128fffc
08000001
add $t0 $t3 $zero;
addi $t0 $t0 1;
lw $t0 8($s6);
sw $t0 -3($s5);
beq $t0 $t1 -16;
j 4;
Discompile successfully!