

计算机组成课程实验报告—MIPS汇编软件实验

计算机组成课程实验报告—MIPS汇编软件实验

- 1. 基本信息
 - 1.1 小组成员信息
 - 1.2 实验信息一览
 - 编译器版本
 - 使用已编译文件运行步骤
 - 具体编译运行步骤
- 2. 题目要求
 - 2.1 总体目标
 - 2.2 具体要求
- 3. 实验原理
 - 3.1 头文件与命名空间
 - 3.2 C++面向对象特性的利用
 - MIPSAssembler 类的属性
 - MIPSAssembler 类的方法
 - MIPSAssembler 的完整定义
 - 3.3 宏的利用
 - 3.4 AssembleMIPS 函数的完整逻辑
 - 3.5 核心思路
 - 指令与寄存器
 - 文件名（输入输出）
 - 错误传递
 - 字符串流的使用
- 4. 实验过程与结果
 - 4.1 编写源码
 - 4.2 编写测试数据并测试
 - 正常输入
 - 各种错误情况
 - Linux下使用Windows的 CRLF 换行方式
 - 错误（不规范）的语法

1. 基本信息

1.1 小组成员信息

姓名	学号
徐震	3180105504
麦昌楷	3180101982
钟添芸	3180103009
江昊翰	3180101995

1.2 实验信息一览

编译器版本

注：

- 为了程序的完整性和跨平台性，我们在除了实验要求的平台外的其他平台做了相关的编译与运行测试。
- 为了让程序有更大的容错性，我们花了很大一部分精力来完善本程序的错误识别与错误处理系统。在完成基本要求：MIPS汇编器的基础上，我们的程序可以识别出用户输入中“可原谅”（例如某些位置的多余空格）以及“不可原谅”（语法错误，指令错误，寄存器错误等）的错误
 - 对于可原谅的错误，我们进行正常的输入输出。
 - 对于不可原谅的错误，我们停止程序运行，输出当前的汇编进度并进行错误报告，以便用户排查。

主要平台：

- Visual Studio 2019

经过测试的其他平台：

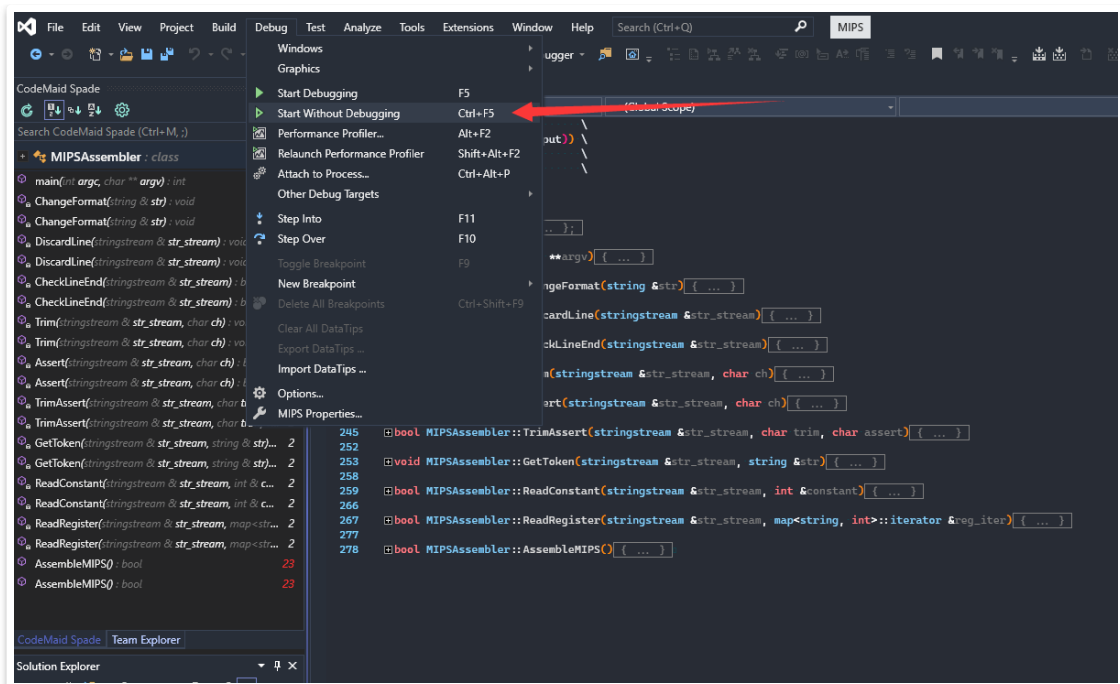
- Windows: GCC 8.0.0(mingw-w64), CLANG 9.0.0(LLVM)
- Linux: GCC 7.4.0(WSL Ubuntu),GCC 9.2.1(KALI DEBIAN)

使用已编译文件运行步骤

- 无命令行参数的情况下默认输入文件名为：`my1.asm`，输出为 `my1.txt`。
- 如果程序有两个命令行参数（两个以上则取前两个），那么以第一个为输入文件名，第二个为输出文件名。
- 若是在Windows平台运行，可以直接在工程文件夹的 `Windows` 目录下打开cmd或其他命令行工具（如Windows Terminal），然后直接运行 `.exe` 文件（各种编译器的结果都可）。
- 若是在Linux平台运行，可以直接在shell环境下进入工程文件夹下的 `Linux` 目录，然后直接运行 `.out` 文件。
- 注意Windows和Linux平台使用的换行方式不一样，因此两者的对应的文件夹下的 `my1.asm` 和 `my1.txt` 对应的换行方式也不同。

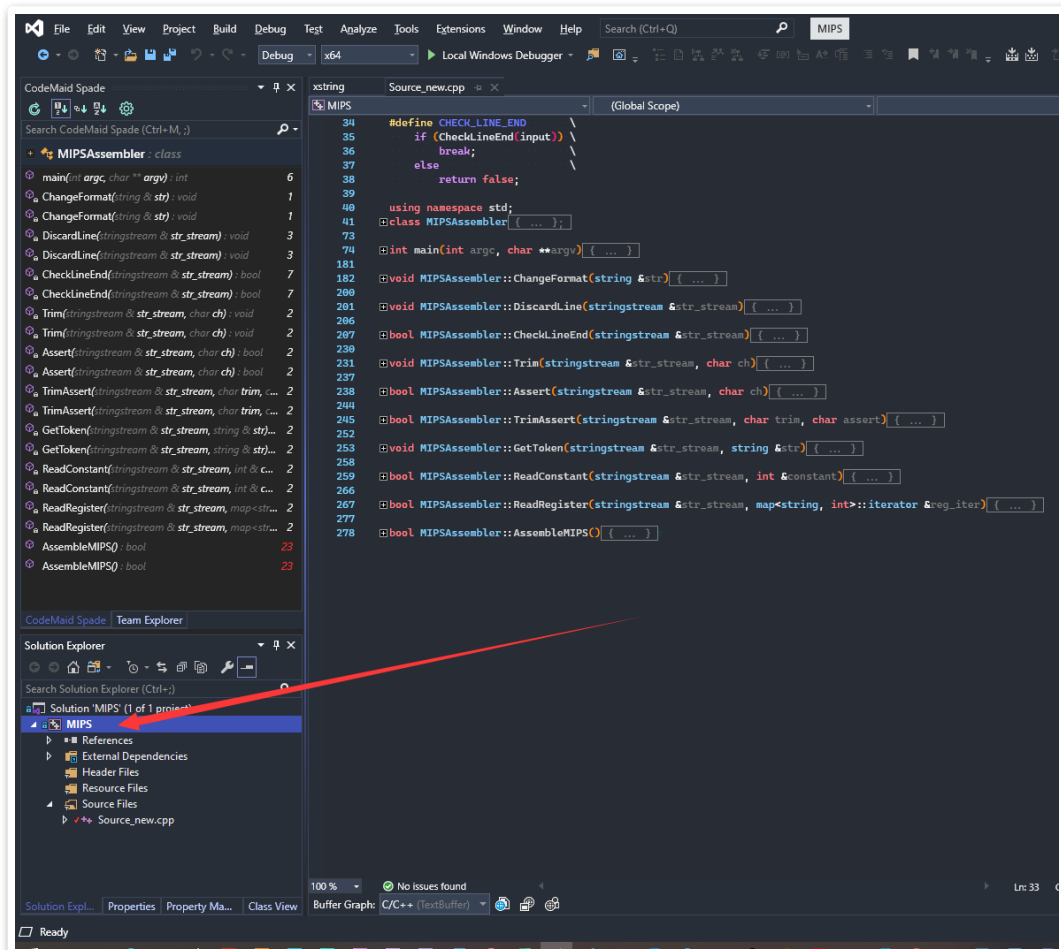
具体编译运行步骤

- 若使用Visual Studio 2019，可以在打开工程文件夹下的 `MIPS.sln` 文件后，直接按下快捷键 `Ctrl + F5` 编译并直接运行（不调试）。

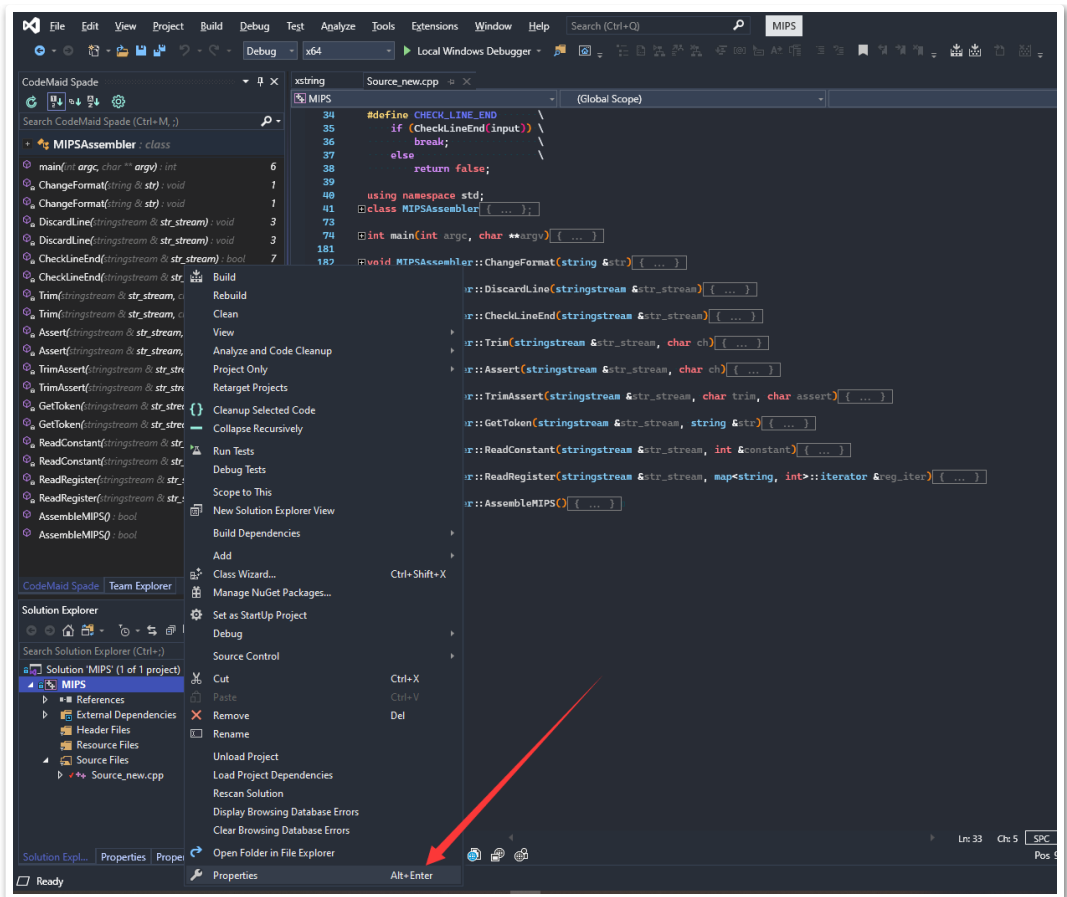


- 若使用较低版本的Visual Studio，可以新建一个空的工程，直接将工程文件夹下的 `Source.cpp`、`my1.asm`、`my1.txt` 复制到工程文件夹下，并在工程中添加 `Source.cpp` 文件，然后直接按下快捷键 `Ctrl` + `F5` 编译并直接运行（不调试）。
- 若要在Visual Studio环境下添加命令行参数，请采取以下步骤：

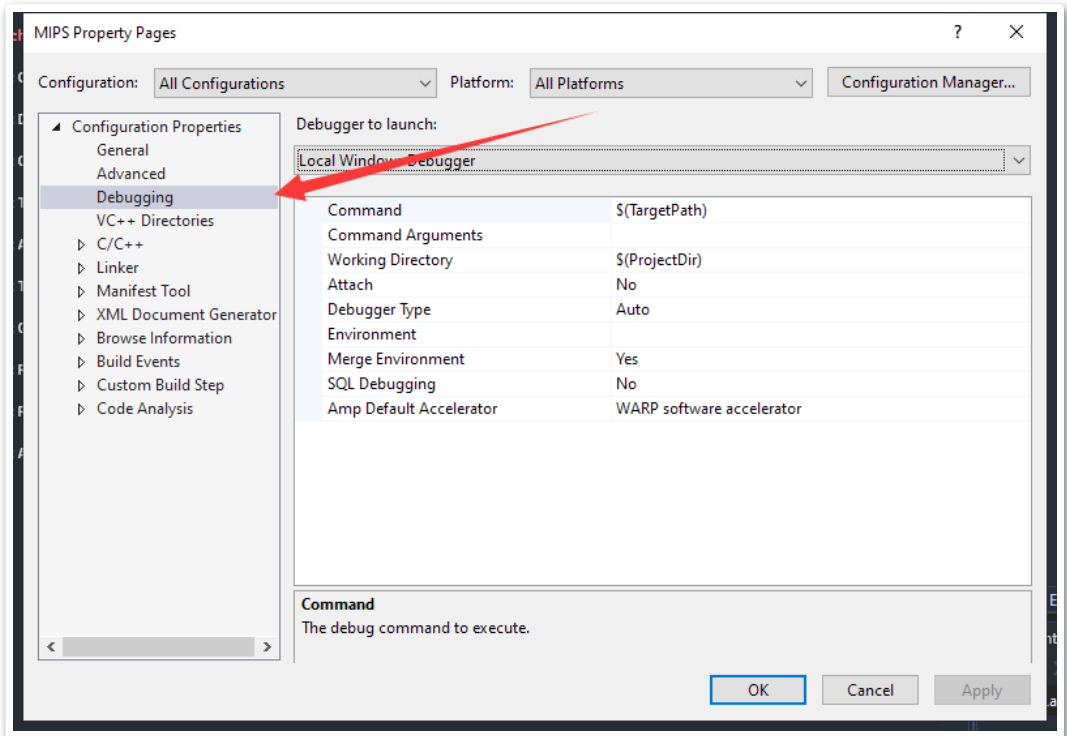
1. 在Visual Studio的工程位置点击右键



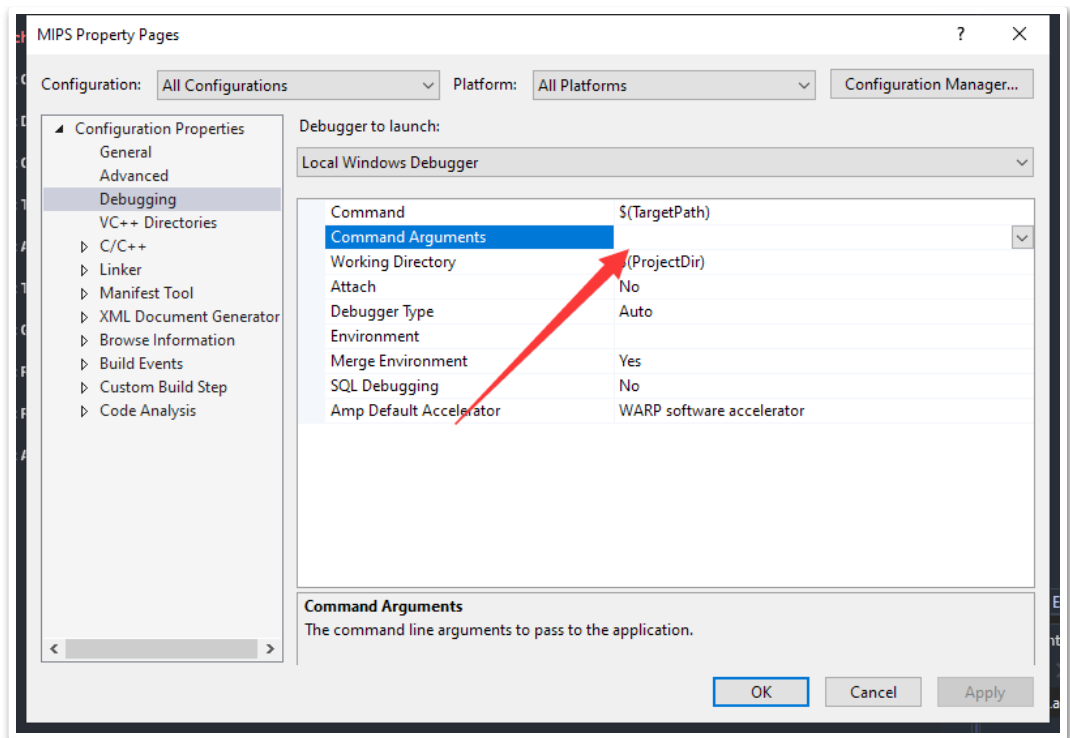
2. 在弹出的选项卡中选择属性一栏



3. 在弹出的选项中选择调试



4. 在右侧的命令行参数中输入需要的命令行参数



- 若使用其他命令行编译器如GCC，CLANG等，可以直接在源文件所在目录下输入形如如下指令的编译命令：

```
1 # gcc
2 gcc Source.cpp -o a.out
3 # clang
4 clang Source.cpp -o a.out
```

然后直接运行：

```
1 # 保证当前文件夹下有需要的my1.asm文件
2 # 且文件的换行方式符合系统要求：Linux LF，Windows CRLF
3 ./a.out
4 # 或者添加命令行参数
5 # 第一个参数为输入文件名
6 # 第二个参数为输出文件名
7 ./a.out my1.asm my1.txt
```

2. 题目要求

2.1 总体目标

实现MIPS源程序的汇编器（MIPS源程序变机器码，MIPS源程序在文件my1.asm中，生成的机器码文件可放在my1.txt，my1.asm和my1.txt都是文本文件，不是二进制文件，放在你的可执行文件所在目录，下同。my1.asm和my1.txt这两个固定文件名固定不变。）

2.2 具体要求

1. 编译器：

- visual studio 2010--2019版本的微软Visual Studio编译器，编写你的VC++程序。
- 最近5年发布的Ubuntu Linux的GCC
- 其他版本的编译器经老师和同学讨论后，决定并公告

实验报告第1章（或第1节）请说明你所用的编译器及版本，说明你的程序如何编译的各步骤说明。

- 要求实现第三版教材图2-20的所有指令（见下面图1）、下面图2的 `addi` 指令、下面图3的 `lb`（load byte）和 `sb`（store byte）指令，你的汇编源程序可以无实际意义，但每条指令需要出现3次（3者的操作数差别要尽可能大）。对所有指令（共20条），除了系统专用的3个寄存器（`$at,$k1,$k2`）外的29个寄存器，都必须出现2次或以上。

MIPS machine language								
Name	Format	Example						Comments
<code>add</code>	R	0	18	19	17	0	32	<code>add \$s1,\$s2,\$s3</code>
<code>sub</code>	R	0	18	19	17	0	34	<code>sub \$s1,\$s2,\$s3</code>
<code>lw</code>	I	35	18	17	100			<code>lw \$s1,100(\$s2)</code>
<code>sw</code>	I	43	18	17	100			<code>sw \$s1,100(\$s2)</code>
<code>and</code>	R	0	18	19	17	0	36	<code>and \$s1,\$s2,\$s3</code>
<code>or</code>	R	0	18	19	17	0	37	<code>or \$s1,\$s2,\$s3</code>
<code>nor</code>	R	0	18	19	17	0	39	<code>nor \$s1,\$s2,\$s3</code>
<code>andi</code>	I	12	18	17	100			<code>andi \$s1,\$s2,100</code>
<code>ori</code>	I	13	18	17	100			<code>ori \$s1,\$s2,100</code>
<code>sll</code>	R	0	0	18	17	10	0	<code>sll \$s1,\$s2,10</code>
<code>srl</code>	R	0	0	18	17	10	2	<code>srl \$s1,\$s2,10</code>
<code>beq</code>	I	4	17	18	25			<code>beq \$s1,\$s2,100</code>
<code>bne</code>	I	5	17	18	25			<code>bne \$s1,\$s2,100</code>
<code>slt</code>	R	0	18	19	17	0	42	<code>slt \$s1,\$s2,\$s3</code>
<code>j</code>	J	2	2500					<code>j 10000 (see Section 2.9)</code>
Field size		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions 32 bits
R-format	R	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	I	op	rs	rt	address			Data transfer, branch format

FIGURE 2.13 MIPS machine language revealed through Section 2.6. Highlighted portions show MIPS structures introduced in Section 2.6. The J-format, used for jump instructions, is explained in Section 2.9. Section 2.9 also explains the proper values in address fields of branch instructions.

Name	Register number	Usage	Preserved on call?
<code>\$zero</code>	0	The constant value 0	n.a.
<code>\$v0-\$v1</code>	2–3	Values for results and expression evaluation	no
<code>\$a0-\$a3</code>	4–7	Arguments	no
<code>\$t0-\$t7</code>	8–15	Temporaries	no
<code>\$s0-\$s7</code>	16–23	Saved	yes
<code>\$t8-\$t9</code>	24–25	More temporaries	no
<code>\$gp</code>	28	Global pointer	yes
<code>\$sp</code>	29	Stack pointer	yes
<code>\$fp</code>	30	Frame pointer	yes
<code>\$ra</code>	31	Return address	yes

FIGURE 2.14 MIPS register conventions. Register 1, called `$at`, is reserved for the assembler (see Section 2.12), and registers 26–27, called `$k0-$k1`, are reserved for the operating system. This information is also found in Column 2 of the MIPS Reference Data Card at the front of this book.

Load byte

`lb rt, address`

0x20	rs	rt	Offset
6	5	5	16

Load unsigned byte

`lbu rt, address`

0x24	rs	rt	Offset
6	5	5	16

Load the byte at *address* into register *rt*. The byte is sign-extended by `lb`, but not by `lbu`.

Store byte

sb rt, address

0x28	rs	rt	Offset
6	5	5	16

Store the low byte from register *rt* at *address*.

3. 要求对实验结果截图，并说明。提交包含源程序的工程文件文件夹（删除 `obj` 文件及其他大于 500KB 的编译中间文件，但留下 `exe` 文件（对 Windows 系统的 Visual Studio）或可执行文件（对 Visual Studio 之外的系统）、PDF 或 WORD 格式的实验报告、`my1.asm`、`my1.txt` 文件，这些文件和目录压缩成一个压缩包文件。
4. `my1.txt` 内容是文本，不是二进制形式的不可见字符（乱码）。`My1.txt` 的典型格式如下：

```
1      1000:10001100 10001001 00000000 00000000      100011 00100 01001 00000 00000
      000000
2      1004:00000000 00000000 01000000 00100000      000000 00000 00000 01000 00000
      100000
3      .....
```

上面的标号 1000 是十进制（16 进制也可以），接下去，左边 32 位是“8 位-8 位-8 位-8 位”的机器码，右边 32 位是“6 位-5 位-5 位-5 位-5 位-6 位”格式的机器码。

5. 你的程序需要能够由助教或老师试运行，会抽查，请提供可执行文件，典型数据文件，实验报告最后一节请提供程序如何运行的使用说明（最好有截图）。

3. 实验原理

3.1 头文件与命名空间

```
1  #include <bitset>
2  #include <fstream>
3  #include <iomanip>
4  #include <iostream>
5  #include <map>
6  #include <set>
7  #include <sstream>
8  #include <string>
9  #include <vector>
10
11  using namespace std;
```

3.2 C++ 面向对象特性的利用

MIPSAssembler 类的属性

我们在实验的实现过程中将汇编器抽象为一个对象。他有如下的属性：

- `struct inst`：一个用于方便的表示一条指令的小结构，包含
 - `format`：指令的结构代码（R、I 或 J）
 - `op`：指令的操作码

- `funct` : 指令的功能码

```
1 // This small structure is for easily declaring different part of an
  instruction
2 struct inst {
3     char format; // Format of an instruction(R, I or J)
4     int op;      // Opcode of an instruction
5     int funct;   // Funct code of an instruction
6 };
```

- `stringstream input, output` : 用于储存输入输出数据的字符串流
 - `input` : 输入数据流
 - `output` : 输出数据流

```
1 stringstream input, output; // string stream for storing input and output
```

- `map<string, inst> inst_map` : 指令集，指令的字符串表示与指令的特征相对应

```
1 map<string, inst> inst_map; // inst_map: Instruction Map, maps a string
  representation of an instruction to its corresponding format and
  opcode/function
```

- `map<string, int> reg_map` : 寄存器集，寄存器的字符串表示与寄存器数字化表示相对应

```
1 map<string, int> reg_map; // reg_map: Register Map, maps a string
  representation of a register to its corresponding register number
```

- `int init_line_number = 0x1000` : 输出的起始行号，每编码一次指令自增4，以Byte为单位

```
1 set<char> reserved_char; // Reserved characters that should be considered
  an end for token recognition
```

- `set<string> reserved_char` : 保留的字符，本 `set` 中的所有元素会被认为是一个token的结尾（token不包括该元素）
 - 值得注意的是，我们将“#”也作为保留字符，因为我们在 `asm` 文件中实现了注释（注释以“#”开头）

```
1 set<char> reserved_char; // Reserved characters that should be considered
  an end for token recognition
```

- `bool Linux_warning = false` : 与下面 `DiscardLine` 中提到的问题相对应

```
1 bool Linux_warning = false;
```

- `int source_line_number = 1` : 用以统计源文件的行号，便于出错后信息的调试

MIPSAssembler 类的方法

汇编器这一对象包含的方法有：

- `void ChangeFormat(string &str)` : 将一组未拓展的数据拓展成实验要求的格式，例如
 - 从 `10001100100010010000000000000000` 到 `10001100 10001001 00000000 00000000`
`100011 00100 01001 00000 00000 000000`

```
1 void MIPSAssembler::ChangeFormat(string &str)
2 {
3     str.append(str); // Duplicate string
```



```

4      /**
5       * Add space correspondingly
6       * Note that string::insert will insert things at the front of the position
       specified
7       * that is, inserting at string::end() is legal, while inserting at
       string::begin() is not
8       */
9       str.insert(8,      1, ' ');
10      str.insert(16 + 1, 1, ' ');
11      str.insert(24 + 2, 1, ' ');
12      str.insert(32 + 3, 6, ' ');
13      str.insert(38 + 9, 1, ' ');
14      str.insert(43 + 10, 1, ' ');
15      str.insert(48 + 11, 1, ' ');
16      str.insert(53 + 12, 1, ' ');
17      str.insert(58 + 13, 1, ' ');
18  }

```

- `void DiscardLine(stringstream &str_stream)` : 对字符串流进行操作，删除本行（行末可能是回车或 EOF）。

```

1  void MIPSAssembler::DiscardLine(stringstream &str_stream)
2  {
3      source_line_number++;
4      while ((str_stream.get() != '\n') && (!str_stream.eof()))
5          ;
6  }

```

- `bool CheckLineEnd(stringstream &str_stream)` : 检查当前的字符串流的行结尾是否符合要求。本方法在Linux与Windows平台的表现不同：
 - Windows下默认的回车换行模式是 `CRLF` : Carriage Return Line Feed，在文件中储存为两个字符 `\r` (0x0A) 与 `\n` (0x0D)，但在各类涉及到字符串交互的操作中，Windows会将 `\n` 解释为 `\r`，`\n` 两个字符。
 - Linux下的回车换行模式是 `LF` : Line Feed，在文本中储存为 `\n` (0x0D) 一个字符，且Linux下的 `\n` 就会被解释为 `\n`，因此在Linux下读取Windows中保存的文件时，如果换行方式为 `CRLF` 就会出现换行符的读取问题。
 - 在我们的程序中，如果试图在Linux中用Linux下编译的可执行文件读取以 `CRLF` 方式换行的文件，我们会给出警告，但是汇编工作会继续进行。
 - 如果某个文件是混合模式换行的（既有 `CRLF` 也有 `LF`），在Linux下会得到警告，而Windows会忽略这个问题。
 - 同样的，Windows会将 `LF` 和 `CRLF` 模式的文件等同对待。
 - 注意，在 `DiscardLine` 中其实也会遇到类似的问题，但由于Windows和Linux对 `\n` 的解读方式不同，程序中这样的写法完全能胜任我们删除本行的要求。

```

1  bool MIPSAssembler::CheckLineEnd(stringstream &str_stream)
2  {
3      Trim(str_stream, ' ');
4      if (str_stream.peek() == '#' || str_stream.peek() == '\n') {
5          DiscardLine(input);
6          return true;
7      } else if (str_stream.peek() == '\r') {
8          if (!Linux_warning) {
9              Linux_warning = true;
10             cerr << "WARNING: Processing CRLF file on Linux system." << endl;
11         }

```

```

12     str_stream.get();
13     if (str_stream.peek() != '\n') {
14         cerr << "FATAL: Carriage Return encountered, however no Line Feed
is found, file corrupted." << endl;
15         return false;
16     }
17     DiscardLine(input);
18     return true;
19 } else if (str_stream.peek() != EOF) {
20     cerr << "FATAL: Unexpected line end." << endl;
21     return false;
22 } else
23     return true;
24 }

```

- `void Trim(stringstream &str_stream, char ch)` : 本方法会将参数中的字符串流接下来的所有 `ch` 清除掉，直到遇到不同于 `ch` 的字符。

```

1 void MIPSAssembler::Trim(stringstream &str_stream, char ch)
2 {
3     while (str_stream.get() == ch) // Will consume an extra character, so we
put it back
4         ;
5     str_stream.unget();
6 }

```

- `bool Assert(stringstream &str_stream, char ch)` : 本方法会检测当前字符串的下一个字符是否为我们认为的，返回 `true` 如果是，如果不是返回 `false`。

```

1 bool MIPSAssembler::Assert(stringstream &str_stream, char ch)
2 {
3     if (str_stream.get() == ch) return true; // Get next character and validate
it
4     cerr << "FATAL: Unable to read the character: " << ch << endl;
5     return false;
6 }

```

- `bool TrimAssert(stringstream &str_stream, char trim, char assert)` : 本方法用以处理正则表达式 `\s*,\s*` 会匹配的内容，即空格，逗号，空格，通常见于寄存器间的分隔。也可以用于其他情况的处理，例如 `lw`，`sw` 等指令的左右括号。

```

1 bool MIPSAssembler::TrimAssert(stringstream &str_stream, char trim, char
assert)
2 {
3     Trim(input, trim);
4     if (!Assert(str_stream, assert)) return false;
5     Trim(input, trim);
6     return true;
7 }

```

- `void GetToken(stringstream &str_stream, string &str)` : 获取一个单字，注意，这个方法不会对空格等进行处理，所以若是没有删除多余的空格，`str` 不会获得任何赋值而会被清空，因此本方法一般与 `Trim` 或者 `TrimAssert` 连用。

```

1 void MIPSAssembler::GetToken(stringstream &str_stream, string &str)
2 {
3     str.clear();
4     // Clear string buffer
5     while (!reserved_char.count(str_stream.peek()))
6         str.push_back(str_stream.get()); // Get char if not reserved
7 }

```

- `bool ReadRegister(stringstream &str_stream, map<string, int>::iterator ®_iter)` : `GetToken` 并尝试将本寄存器映射到事先定义好的集合里，若失败则会报错。注意，同样不会清空空格，因此一般与 `Trim` 或者 `TrimAssert` 连用。

```

1 bool MIPSAssembler::ReadRegister(stringstream &str_stream, map<string,
2 int>::iterator &reg_iter)
3 {
4     string str;
5     GetToken(input, str);
6
7     // Return false if we cannot map the string to a register
8     if ((reg_iter = reg_map.find(str)) != reg_map.end()) return true;
9     cerr << "FATAL: Unable to map register: " << str << " to int value." <<
10    endl;
11    return false;
12 }

```

- `bool ReadConstant(stringstream &str_stream, int &constant)` : 由于内部实现中用了流操作，因此会去除空格等，但在实际使用该方法的时候，我们还是将其与 `Trim` 或者 `TrimAssert` 连用。

```

1 bool MIPSAssembler::ReadConstant(stringstream &str_stream, int &constant)
2 {
3
4     if ((str_stream >> constant)) return true; // Try interpreting the string
5     into an integer
6     cerr << "FATAL: Unable to read the constant." << endl;
7     return false;
8 }

```

- `bool AssembleMIPS()` : 核心方法。对输入和输出进行汇编。是本汇编器最主要的函数，通过调用上面列举的各种方法以实现我们需要的逻辑判断，字符匹配，错误处理等。也是本汇编器唯一对外暴露的方法。
 - 关于 `AssembleMIPS` 的具体实现会在下面宏定义的章节详细阐述。

MIPSAssembler 的完整定义

```

1 class MIPSAssembler
2 {
3     // This small structure is for easily declaring different part of an
4     instruction
5     struct inst {
6         char format; // Format of an instruction(R, I or J)
7         int op;      // Opcode of an instruction
8         int funct;   // Funct code of an instruction
9     };
10    void ChangeFormat(string &str);
11    // Convert tightly spaced string to the asked form

```

```

10     void DiscardLine(stringstream &str_stream);
    // Discard current line until \n or EOF
11     bool CheckLineEnd(stringstream &str_stream);
    // When the recognition of one full instruction is done, check current line for
    extra char
12     void Trim(stringstream &str_stream, char ch);
    // Discard a certain character, usually space
13     bool Assert(stringstream &str_stream, char ch);
    // return false if the next character is not what we expected
14     bool TrimAssert(stringstream &str_stream, char trim, char assert);
    // Trim a certain char, assert if the next is not expected, then trim the char
    again. Usually for "\s*,\s*"
15     void GetToken(stringstream &str_stream, string &str);
    // Get a token until reserved char is encountered. Usually a register
16     bool ReadRegister(stringstream &str_stream, map<string, int>::iterator
    &reg_iter); // Get a token and try to map it to an actual register. Returns false
    if map is unsuccessful
17     bool ReadConstant(stringstream &str_stream, int &constant);
    // Read in a constant. Using stream
18     bool Linux_warning = false;
19
20     public:
21     /**
22     * Using map can give us a clear interface for the instructions in case we want
    to add or delete or modify them
23     * For instructions of format 'J' and 'I', we use 0 to fulfill the empty block
    that will not be used
24     * We've considered using enum, but found it unnecessary
25     */
26     bool AssembleMIPS();           // Core method and the only method exposed.
    Returns false if the assembling is unsuccessful
27     stringstream input, output;   // string stream for storing input and output
28     map<string, inst> inst_map;     // inst_map: Instruction Map, maps a string
    representation of an instruction to its corresponding format and opcode/function
29     map<string, int> reg_map;       // reg_map: Register Map, maps a string
    representation of a register to its corresponding register number
30     set<char> reserved_char;       // Reserved characters that should be considered
    an end for token recognition
31     int init_line_number = 0x1000; // The counter to add at the beginning of a line
32     int source_line_number = 1;    // Counter for source code file line number
33 };

```

3.3 宏的利用

在我们的 `MIPSAssembler` 对象中，最主要的方法是 `AssembleMIPS`，然而为了能够正确处理各种不同的错误，本函数的逻辑变得十分复杂。在Visual Studio的Code Maid插件中本方法的复杂度已经到了20。

纵使我们已经利用了C++中的面向对象特性优化了部分复杂之处（本程序的第一个版本（未优化）`AssembleMIPS`的复杂度达到了50，是坚决不可接受的），这一核心函数仍旧有变得整洁的可能性，于是我们使用了C++语言中的宏来让代码变得整洁。

在 `AssembleMIPS` 中我们定义了如下的宏：

```

1  #define PROCESS_REGISTER \
2      if (!ReadRegister(input, reg_iter)) return false; \
3      output << bitset<5>((*reg_iter).second);

```

```

4
5  #define PROCESS_FIRST_REGISTER \
6      Trim(input, ' ');          \
7      PROCESS_REGISTER
8
9  #define PROCESS_OTHER_REGISTER(ch) \
10     if (!TrimAssert(input, ' ', (ch))) return false; \
11     PROCESS_REGISTER
12
13  #define PROCESS_CONSTANT(width) \
14     if (!ReadConstant(input, constant)) return false; \
15     output << bitset<(width)>(constant);
16
17  #define PROCESS_FIRST_CONSTANT(width) \
18     Trim(input, ' ');          \
19     PROCESS_CONSTANT(width)
20
21  #define PROCESS_OTHER_CONSTANT(ch, width) \
22     if (!TrimAssert(input, ' ', (ch))) return false; \
23     PROCESS_CONSTANT(width)
24

```

使用宏的原因如下：

- 宏就是文字内容的展开，没有额外的堆栈或参数传递问题，没有开销，并且可以对当前范围的变量进行操作。
- 最重要的是，宏可以将函数中包含返回指令的语句简化，这也是本程序经常需要用到的（检测到错误则返回 `false`）。
- 用宏相互嵌套可以让函数的逻辑更清晰。剔除复杂的部分，提高程序的抽象程度。

3.4 AssembleMIPS 函数的完整逻辑

程序的详细运行逻辑可以从注释/具体代码看到。

```

1  bool MIPSAssembler::AssembleMIPS()
2  {
3      #define PROCESS_REGISTER ...
4      #define PROCESS_FIRST_REGISTER ...
5      #define PROCESS_OTHER_REGISTER(ch) ...
6      #define PROCESS_CONSTANT(width) ...
7      #define PROCESS_FIRST_CONSTANT(width) ...
8      #define PROCESS_OTHER_CONSTANT(ch, width) ...
9      #define CHECK_LINE_END ...
10
11      int constant;                // A constant
12      string str;                  // General purpose string
13      map<string, inst>::iterator inst_iter; // Reserve the result of std::map::find
14      map<string, int>::iterator reg_iter;    // Reserve the result of std::map::find
15      while (1) {
16          // Skip comment and empty line and discard leading space
17          Trim(input, ' ');
18          if (input.peek() == '#' || input.peek() == '\n') {
19              DiscardLine(input);
20              continue;
21          } else if (input.peek() == '\r') {
22              if (!Linux_warning) {
23                  Linux_warning = true;

```

```

24         cerr << "WARNING: Processing CRLF file on Linux system." << endl;
25     }
26     input.get();
27     if (input.peek() != '\n') {
28         cerr << "FATAL: Carriage Return encountered, however no Line Feed
is found, file corrupted." << endl;
29         return false;
30     }
31     DiscardLine(input);
32     continue;
33 }
34
35     GetToken(input, str);                                // Try getting the
first token(instruction)
36     if (str.empty() && input.peek() == EOF) return true; // If the EOF is
encountered when trying to get a token, we should stop
37
38     output << setbase(16) << init_line_number << ':'; // Output line number
39     init_line_number += 4;                             // Increment the line
number
40
41     if ((inst_iter = inst_map.find(str)) == inst_map.end()) {
42         cerr << "FATAL: Unable to map instruction: \"" << str << "\" in
instruction map." << endl;
43         return false;
44     }                                                    // Return false if we cannot
map the string to an instruction
45     output << bitset<6>((*inst_iter).second.op); // Output opcode if
instruction is found
46
47     switch ((*inst_iter).second.format) {
48     case 'R':
49         if ((*inst_iter).first == "jr") {
50
51             PROCESS_FIRST_REGISTER
52             for (int i = 0; i < 3; i++) output << bitset<5>(0); // Zero fill
53
54         } else if (((*inst_iter).first == "sll") || ((*inst_iter).first ==
"srl")) {
55
56             // Similar implementation as the I format(only these two
instructions are of I format)
57             output << bitset<5>(0); // First five bit of sll and srl is filled
with zero
58
59             PROCESS_FIRST_REGISTER
60             PROCESS_OTHER_REGISTER(',', ')
61             PROCESS_OTHER_CONSTANT(',', ', 5)
62
63         } else {
64
65             // Process the first register
66             PROCESS_FIRST_REGISTER
67
68             // Process two more register
69             PROCESS_OTHER_REGISTER(',', ')
70             PROCESS_OTHER_REGISTER(',', ')
71
72             output << bitset<5>(0); // Output shamt

```

```

72         }
73
74         // Output funct code of an instruction
75         output << bitset<6>((*inst_iter).second.funct);
76         break;
77
78     case 'I':
79
80         if (set<string>({"lw", "sw", "lb", "sb"}).count((*inst_iter).first)) {
81
82             PROCESS_FIRST_REGISTER
83             PROCESS_OTHER_CONSTANT(',', 0)
84             PROCESS_OTHER_REGISTER('(')
85             output << bitset<16>(constant);
86             if (!TrimAssert(input, ' ', ')')) return false;
87
88         } else {
89
90             PROCESS_FIRST_REGISTER
91             PROCESS_OTHER_REGISTER(',')
92             PROCESS_OTHER_CONSTANT(',', 16)
93         }
94         break;
95
96     case 'J':
97         Trim(input, ' ');
98         // Read in the constant(immediate value)
99         if (!ReadConstant(input, constant)) return false;
100        output << bitset<26>(constant);
101        break;
102    }
103    if (!CheckLineEnd(input)) return false;
104    // Splitting the machine code we've generated
105    output << endl; // Append a newline for getline to work
106    output.seekg(-32 - 1, ios::end); // Change stream input pointer
107    getline(output, str); // Store whole line
108    ChangeFormat(str); // Reformat
109    output.seekp(-32 - 1, ios::end); // Change stream output pointer
110    output << str; // Append the formatted string
111    output << endl; // previous newline character is consumed
112    by getline
113    }
114    return true; // Assembly success
115 }

```

3.5 核心思路

指令与寄存器

为让我们的 `MIPSAssembler` 正常工作，我们在主程序中应首先将其实例化。然后为需要的常数进行赋值：

```

1 // Initializing constants
2 A.init_line_number = 0x1000;
3 A.reserved_char = {' ', '#', '(', ')', ' ', EOF, '\n', '\r'};
4 A.inst_map = {
5     {"add", {'R', 0, 32}},

```

```

6      {"sub", {'R', 0, 34}},
7      {"lw", {'I', 35, 0}},
8      {"sw", {'I', 43, 0}},
9      {"and", {'R', 0, 36}},
10     {"or", {'R', 0, 37}},
11     {"nor", {'R', 0, 39}},
12     {"andi", {'I', 12, 0}},
13     {"ori", {'I', 13, 0}},
14     {"sll", {'R', 0, 0}},
15     {"srl", {'R', 0, 2}},
16     {"beq", {'I', 4, 0}},
17     {"bne", {'I', 5, 0}},
18     {"slt", {'R', 0, 42}},
19     {"j", {'J', 2, 0}},
20     {"jr", {'R', 0, 8}},
21     {"jal", {'J', 3, 0}},
22     {"addi", {'I', 8, 0}},
23     {"lb", {'I', 32, 0}},
24     {"sb", {'I', 40, 0}},
25 };
26 A.reg_map = {
27     {"$zero", 0},
28     {"$at", 1},
29     {"$v0", 2},
30     {"$v1", 3},
31     {"$a0", 4},
32     {"$a1", 5},
33     {"$a2", 6},
34     {"$a3", 7},
35     {"$t0", 8},
36     {"$t1", 9},
37     {"$t2", 10},
38     {"$t3", 11},
39     {"$t4", 12},
40     {"$t5", 13},
41     {"$t6", 14},
42     {"$t7", 15},
43     {"$s0", 16},
44     {"$s1", 17},
45     {"$s2", 18},
46     {"$s3", 19},
47     {"$s4", 20},
48     {"$s5", 21},
49     {"$s6", 22},
50     {"$s7", 23},
51     {"$t8", 24},
52     {"$t9", 25},
53     {"$k0", 26},
54     {"$k1", 27},
55     {"$gp", 28},
56     {"$sp", 29},
57     {"$fp", 30},
58     {"$ra", 31},
59 };

```

注意，在 `inst_map` 中我们对I类和J的指令的 `funct` 域也进行了赋值，虽然他们在实际的汇编过程中不会被使用。

文件名（输入输出）

我们使用了命令行参数，当然，如果你选择不使用参数或者只使用一个，没有被赋值的文件名会采用默认值。

赋值的顺序是第一个参数对应输入文件名，第二个参数对应输出文件名。

他们的默认值分别为：

1. 输入：my1.asm
2. 输出：my1.txt

```
1  int main(int argc, char **argv)
2  {
3      string input_file_name = "my1.asm";
4      string output_file_name = "my1.txt";
5      if (argc ≥ 2) input_file_name = argv[1];
6      if (argc ≥ 3) output_file_name = argv[2];
7      MIPSAssembler A; // Instantiate the MIPS assembler
8      ifstream mylasm_file(input_file_name); // open input file
9      if (!mylasm_file.is_open()) {
10         cerr << "FATAL: Unable to open the file specified";
11         return 2;
12     }
13     A.input << mylasm_file.rdbuf(); // load filestream content
14     mylasm_file.close(); // Close input file
15
16     // Open output file
17     ofstream myltxt_file(output_file_name);
18     if (!myltxt_file.is_open()) {
19         cerr << "FATAL: Unable to open the file specified.";
20         return 3;
21     }
22     ...
23     ...
24 }
```

错误传递

在本程序中，错误是通过返回值（布尔值）传递的。例如：

- `main` 函数发现 `AssembleMIPS` 返回 `false` 就知道汇编失败了，会进行DUMP等一系列操作：

```
1      cerr << "FATAL: Illegal instruction or format. Check your syntax." <<
endl; // Error prompt
2      A.output.seekg(0, ios::beg);
// Change output stream pointer position
3      cout << "DUMP: Current line of input stringstream is:" << endl;
// Dump current input string stream
4      A.input.clear();
5      while (A.input.peek() ≠ '\n' && A.input.tellg()) A.input.unget();
6      A.input.get();
7      string buffer;
8      getline(A.input, buffer);
9      cout << buffer << endl;
// Dump current input line
10     cout << "DUMP: Current line number is: " << A.source_line_number <<
endl; // Dump current line number
```

```

11         cout << "DUMP: Current buffer of output stringstream is:" << endl;
           // Dump current output string stream
12         cout << A.output.rdbuf() << endl;
           // Dump current output string stream
13
14         output_file << A.output.rdbuf(); // Dump the stringstream to file
15         output_file.close();           // Close output file
16         return 1;

```

- 如果返回 `true` 则进行字符串流转换，关闭打开的文件等操作：

```

1  A.output.seekg(0, ios::beg);           // Change output stream pointer
    position
2  output_file << A.output.rdbuf();       // Save output string stream to file
    stream
3  output_file.close();                  // Close output file
4  cout << "Assembly successful." << endl; // Successful prompt
5  cout << "Input file is: \"" << input_file_name << "\" << endl;
6  cout << "Output file is: \"" << output_file_name << "\" << endl;
7  return 0;

```

- 同样的，在 `AssembleMIPS` 函数中，各种不同的调用之间也通过 `bool` 类型的返回值进行交流。

字符串流的使用

在本程序中，我们使用了字符串流这一种储存在内存中的对象。虽然它的各种操作和文件流类似，但是使用字符串流给了我们快速操作的可能。（读取内存中的内容远比读取硬盘上的文件快得多）

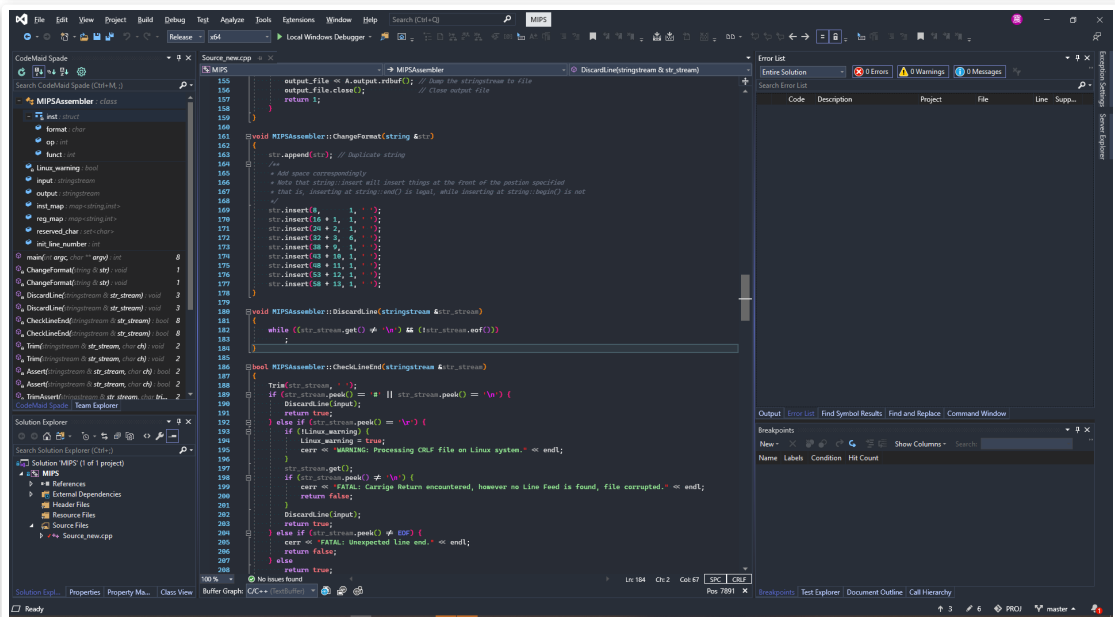
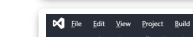
并且结合着面向对象的思想，我们完全可以在读取文件非常大的时候将输入在 `main` 函数中分割，并多次调用 `AssembleMIPS`，这样就实现了大文件的处理。（在现在的版本中暂时没有进行这样的操作，但这种设计方式给了我们这样操作的可能性）

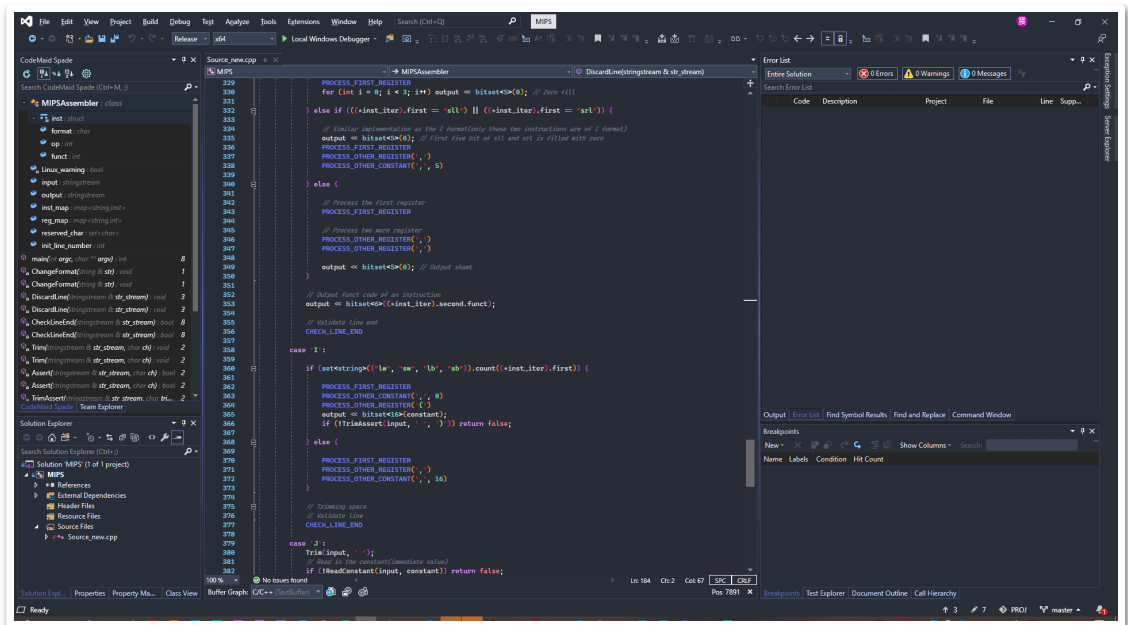
4. 实验过程与结果

4.1 编写源码

在代码编辑器中编写上述源码。

- 编写好代码框架





4.2 编写测试数据并测试

正常输入

以普通格式测试所有涉及到的指令和寄存器以及常数输入。

- 注意在此测试数据中，我们同时测试了空行和注释功能。

```

1  # All implemented instructions and registers test
2  add $v0, $v1, $zero
3  sub $a0, $a1, $a2
4  and $a3, $t0, $t1
5  or $t2, $t3, $t4
6  nor $t5, $t6, $t7
7  slt $s0, $s1, $s2
8
9  add $s3, $s4, $s5
10 sub $s6, $s7, $t8
11 and $t9, $gp, $sp
12 or $fp, $ra, $zero
13 nor $v0, $v1, $zero
14 slt $a0, $a1, $a2
15
16 add $a3, $t0, $t1
17 sub $t2, $t3, $t4
18 and $t5, $t6, $t7
19 or $s0, $s1, $s2
20 nor $s0, $s1, $s2
21 slt $s6, $s7, $t8
22
23 add $t9, $gp, $sp
24 sub $fp, $ra, $zero
25 and $v0, $v1, $zero
26 or $a0, $a1, $a2
27 nor $a3, $t0, $t1
28 slt $t2, $t3, $t4
29
30 andi $t9, $gp, 1
31 ori $s7, $t8, 1

```

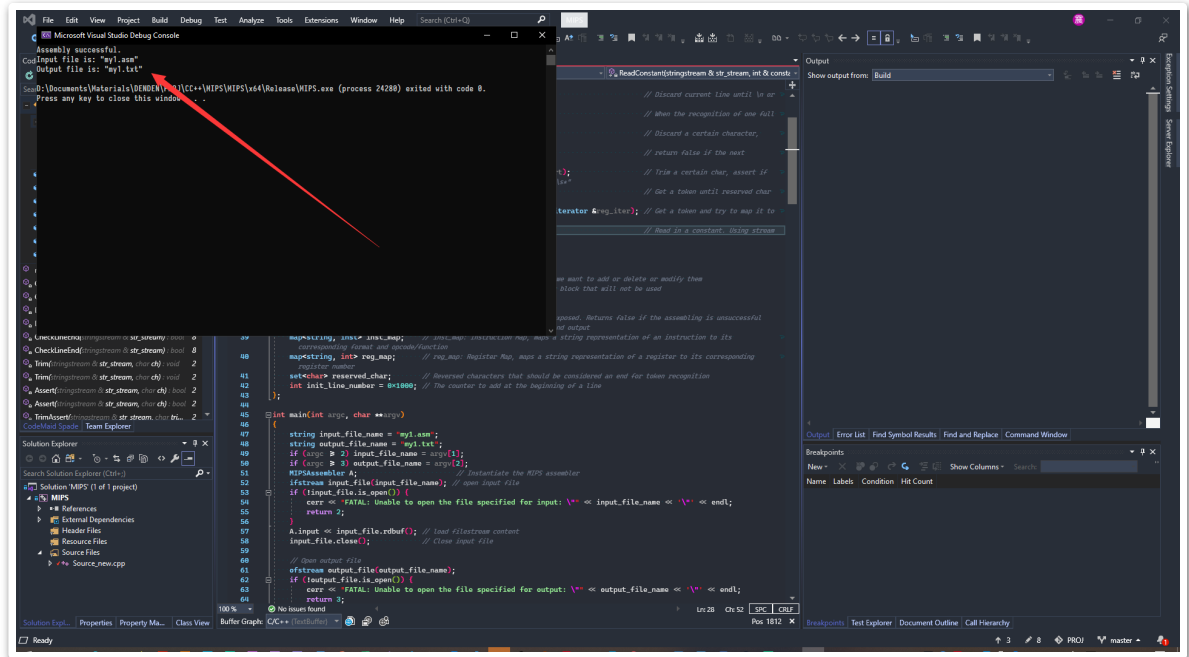
```

32  sll  $s5, $s6, 1
33  srl  $s3, $s4, 1
34  beq  $s1, $s2, 1
35  bne  $t7, $s0, 1
36  addi $t5, $t6, 1
37
38  andi $t3, $t4, 10
39  ori  $t1, $t2, 10
40  sll  $a3, $t0, 10
41  srl  $a1, $a2, 10
42  beq  $a0, $zero, 10
43  bne  $v0, $v1, 10
44  addi $ra, $zero, 10
45
46  andi $sp, $fp, -100
47  ori  $t9, $gp, -100
48  sll  $s7, $t8, -100
49  srl  $s5, $s6, -100
50  beq  $s3, $s4, -100
51  bne  $s1, $s2, -100
52  addi $t7, $s0, -100
53
54  andi $t5, $t6, 1000
55  ori  $t3, $t4, 1000
56  sll  $t1, $t2, 1000
57  srl  $a3, $t0, 1000
58  beq  $a1, $a2, 1000
59  bne  $a0, $zero, 1000
60  addi $v0, $v1, 1000
61
62
63  sw   $v0, 10($v1)
64  lw   $a0, 10($zero)
65  lb   $a1, 10($a2)
66  sb   $a3, 10($t0)
67
68  sw   $t1, 100($t2)
69  lw   $t3, 100($t4)
70  lb   $t5, 100($t6)
71  sb   $t7, 100($s0)
72
73  sw   $s1, 1000($s2)
74  lw   $s3, 1000($s4)
75  lb   $s5, 1000($s6)
76  sb   $s7, 1000($t8)
77
78  sw   $t9, 1($gp)
79  lw   $sp, 1($fp)
80  lb   $ra, 1($zero)
81  sb   $v0, 1($v1)
82
83  jr   $v0
84  jr   $s0
85  jr   $t0
86  jr   $zero
87
88  j    1
89  jal  2

```

```
90
91     j      -3
92     jal    -4
93
94     j      500
95     jal    600
96
97     j      1000
98     jal    -1000
```

运行截图：



输入文件内容截图：

PROJ > CC++ > MIPS > MIPS > my1.asm

You, a few seconds ago | 1 author (You)

All implemented instructions and registers test

```
1  add $v0, $v1, $zero
2  sub $a0, $a1, $a2
3  and $a3, $t0, $t1
4  or  $t2, $t3, $t4
5  nor $t5, $t6, $t7
6  slt $s0, $s1, $s2
7
8
9  add $s3, $s4, $s5
10 sub $s6, $s7, $t8
11 and $t9, $gp, $sp
12 or  $fp, $ra, $zero
13 nor $v0, $v1, $zero
14 slt $a0, $a1, $a2
15
16 add $a3, $t0, $t1
17 sub $t2, $t3, $t4
18 and $t5, $t6, $t7
19 or  $s0, $s1, $s2
20 nor $s0, $s1, $s2
21 slt $s6, $s7, $t8
22
23 add $t9, $gp, $sp
24 sub $fp, $ra, $zero
25 and $v0, $v1, $zero
26 or  $a0, $a1, $a2
27 nor $a3, $t0, $t1
28 slt $t2, $t3, $t4
29
30 andi $t9, $gp, 1
31 ori  $s7, $t8, 1
32 sll  $s5, $s6, 1
33 srl  $s3, $s4, 1
34 beq  $s1, $s2, 1
35 bne  $t7, $s0, 1
36 addi $t5, $t6, 1
37
38 andi $t3, $t4, 10
39 ori  $t1, $t2, 10
40 sll  $a3, $t0, 10
41 srl  $a1, $a2, 10
42 beq  $a0, $zero, 10
43 bne  $v0, $v1, 10
44 addi $ra, $zero, 10
45
46 andi $sp, $fp, -100
47 ori  $t9, $gp, -100
48 sll  $s7, $t8, -100
49 srl  $s5, $s6, -100
50 beq  $s3, $s4, -100
51 bne  $s1, $s2, -100
52 addi $t7, $s0, -100
53
54 andi $t5, $t6, 1000
55 ori  $t3, $t4, 1000
56 sll  $t1, $t2, 1000
57 srl  $a3, $t0, 1000
58 beq  $a1, $a2, 1000
```

PROJ > CC++ > MIPS > MIPS > [10] my1.asm

```
44 addi $ra, $zero, 10
45
46 andi $sp, $fp, -100
47 ori $t9, $gp, -100
48 sll $s7, $t8, -100
49 srl $s5, $s6, -100
50 beq $s3, $s4, -100
51 bne $s1, $s2, -100
52 addi $t7, $s0, -100
53
54 andi $t5, $t6, 1000
55 ori $t3, $t4, 1000
56 sll $t1, $t2, 1000
57 srl $a3, $t0, 1000
58 beq $a1, $a2, 1000
59 bne $a0, $zero, 1000
60 addi $v0, $v1, 1000
61
62
63 sw $v0, 10($v1)
64 lw $a0, 10($zero)
65 lb $a1, 10($a2)
66 sb $a3, 10($t0)
67
68 sw $t1, 100($t2)
69 lw $t3, 100($t4)
70 lb $t5, 100($t6)
71 sb $t7, 100($s0)
72
73 sw $s1, 1000($s2)
74 lw $s3, 1000($s4)
75 lb $s5, 1000($s6)
76 sb $s7, 1000($t8)
77
78 sw $t9, 1($gp)
79 lw $sp, 1($fp)
80 lb $ra, 1($zero)
81 sb $v0, 1($v1)
82
83 jr $v0
84 jr $s0
85 jr $t0
86 jr $zero
87
88 j 1
89 jal 2
90
91 j -3
92 jal -4
93
94 j 500
95 jal 600
96
97 j 1000
98 jal -1000
```


汇编结果截图：

The screenshot displays a web-based MIPS assembler interface. The top navigation bar includes tabs for 'Source_new.cpp', 'my1.txt', 'myvue.css', and 'my1.asm'. The main editor area is titled 'PROJ > CC++ > MIPS > MIPS > my1.txt' and shows the assembly of a C++ program. The assembly output is displayed in a table with columns for address, instruction, and comment. The instructions are in MIPS assembly code, and the comments are in Chinese. The assembly process is shown for a MIPS architecture, with the output being a MIPS assembly file named 'my1.asm'.

```
1 1000:00000000 01000011 00000000 00100000 000000 00010 00011 00000 00000 100000
2 1004:00000000 10000101 00110000 00100010 000000 00100 00101 00110 00000 100010 You, a fe
3 1008:00000000 11101000 01001000 00100100 000000 00111 01000 01001 00000 100100
4 100c:00000001 01001011 01100000 00100101 000000 01010 01011 01100 00000 100101
5 1010:00000001 10101110 01111000 00100111 000000 01101 01110 01111 00000 100111
6 1014:00000010 00010001 10010000 00101010 000000 10000 10001 10010 00000 101010
7 1018:00000010 01110100 10101000 00100000 000000 10011 10100 10101 00000 100000
8 101c:00000010 11010111 11000000 00100010 000000 10110 10111 11000 00000 100010
9 1020:00000011 00111100 11101000 00100100 000000 11001 11100 11101 00000 100100
10 1024:00000011 11011111 00000000 00100101 000000 11110 11111 00000 00000 100101
11 1028:00000000 01000011 00000000 00100111 000000 00010 00011 00000 00000 100111
12 102c:00000000 10000101 00110000 00101010 000000 00100 00101 00110 00000 101010
13 1030:00000000 11101000 01001000 00100000 000000 00111 01000 01001 00000 100000
14 1034:00000001 01001011 01100000 00100010 000000 01010 01011 01100 00000 100010
15 1038:00000001 10101110 01111000 00100100 000000 01101 01110 01111 00000 100100
16 103c:00000010 00010001 10010000 00100101 000000 10000 10001 10010 00000 100101
17 1040:00000010 00010001 10010000 00100111 000000 10000 10001 10010 00000 100111
18 1044:00000010 11010111 11000000 00101010 000000 10110 10111 11000 00000 101010
19 1048:00000011 00111100 11101000 00100000 000000 11001 11100 11101 00000 100000
20 104c:00000011 11011111 00000000 00100010 000000 11110 11111 00000 00000 100010
21 1050:00000000 01000011 00000000 00100100 000000 00010 00011 00000 00000 100100
22 1054:00000000 10000101 00110000 00100101 000000 00100 00101 00110 00000 100101
23 1058:00000000 11101000 01001000 00100111 000000 00111 01000 01001 00000 100111
24 105c:00000001 01001011 01100000 00101010 000000 01010 01011 01100 00000 101010
25 1060:00110011 00111100 00000000 00000001 001100 11001 11100 00000 00000 000001
26 1064:00110110 11111000 00000000 00000001 001101 10111 11000 00000 00000 000001
27 1068:00000000 00010101 10110000 01000000 000000 00000 00000 10101 10110 00001 000000
28 106c:00000000 00010011 10100000 01000010 000000 00000 00000 10011 10100 00001 000010
29 1070:00010010 00110010 00000000 00000001 000100 10001 10010 00000 00000 000001
30 1074:00010101 11110000 00000000 00000001 000101 01111 10000 00000 00000 000001
31 1078:00100001 10101110 00000000 00000001 001000 01101 01110 00000 00000 000001
32 107c:00110001 01101100 00000000 00001010 001100 01011 01100 00000 00000 001010
33 1080:00110101 00101010 00000000 00001010 001101 01001 01010 00000 00000 001010
34 1084:00000000 00000111 01000010 10000000 000000 00000 00000 00111 01000 01010 000000
35 1088:00000000 00000101 00110010 10000010 000000 00000 00000 00101 00110 01010 000010
36 108c:00010000 10000000 00000000 00001010 000100 00100 00000 00000 00000 001010
37 1090:00010100 01000011 00000000 00001010 000101 00010 00011 00000 00000 001010
38 1094:00100011 11100000 00000000 00001010 001000 11111 00000 00000 00000 001010
39 1098:00110011 10111110 11111111 10011100 001100 11101 11110 11111 11110 011100
40 109c:00110111 00111100 11111111 10011100 001101 11001 11100 11111 11110 011100
41 10a0:00000000 00010111 11000111 00000000 000000 00000 00000 10111 11000 11100 000000
42 10a4:00000000 00010101 10110111 00000010 000000 00000 00000 10101 10110 11100 000010
43 10a8:00010010 01110100 11111111 10011100 000100 10011 10100 11111 11110 011100
44 10ac:00010110 00110010 11111111 10011100 000101 10001 10010 11111 11110 011100
45 10b0:00100001 11110000 11111111 10011100 001000 01111 10000 11111 11110 011100
46 10b4:00110001 10101110 00000011 11101000 001100 01101 01110 00000 01111 101000
47 10b8:00110101 01101100 00000011 11101000 001101 01011 01100 00000 01111 101000
48 10bc:00000000 00001001 01010010 00000000 000000 00000 00000 01001 01010 01000 000000
49 10c0:00000000 00000111 01000010 00000010 000000 00000 00000 00111 01000 01000 000010
50 10c4:00010000 10100110 00000011 11101000 000100 00101 00110 00000 01111 101000
51 10c8:00010100 10000000 00000011 11101000 000101 00100 00000 00000 01111 101000
52 10cc:00100000 01000011 00000011 11101000 001000 00010 00011 00000 01111 101000
53 10d0:10101100 01000011 00000000 00001010 101011 00010 00011 00000 00000 001010
54 10d4:10001100 10000000 00000000 00001010 100011 00100 00000 00000 00000 001010
55 10d8:10000000 10100110 00000000 00001010 100000 00101 00110 00000 00000 001010
56 10dc:10100000 11101000 00000000 00001010 101000 00111 01000 00000 00000 001010
57 10e0:10101101 00101010 00000000 01100100 101011 01001 01010 00000 00001 100100
58 10e4:10001101 01101100 00000000 01100100 100011 01011 01100 00000 00001 100100
```

connect Server not selected log | ✓ my1.txt

The screenshot shows a code editor with the following tabs: Source_new.cpp, my1.txt, myvue.css, and my1.asm. The active tab is my1.asm, displaying MIPS assembly code. The code is organized into columns: address, instruction in hexadecimal, instruction in binary, and a comment column. The code spans from address 1078 to 113c. The editor interface includes a top toolbar with icons for file operations, a left sidebar with a file tree, and a bottom status bar showing 'Server not selected' and 'log | my1.txt'.

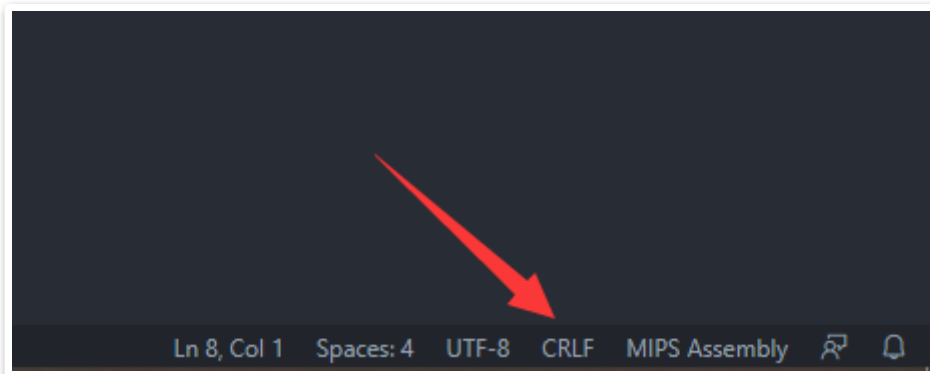
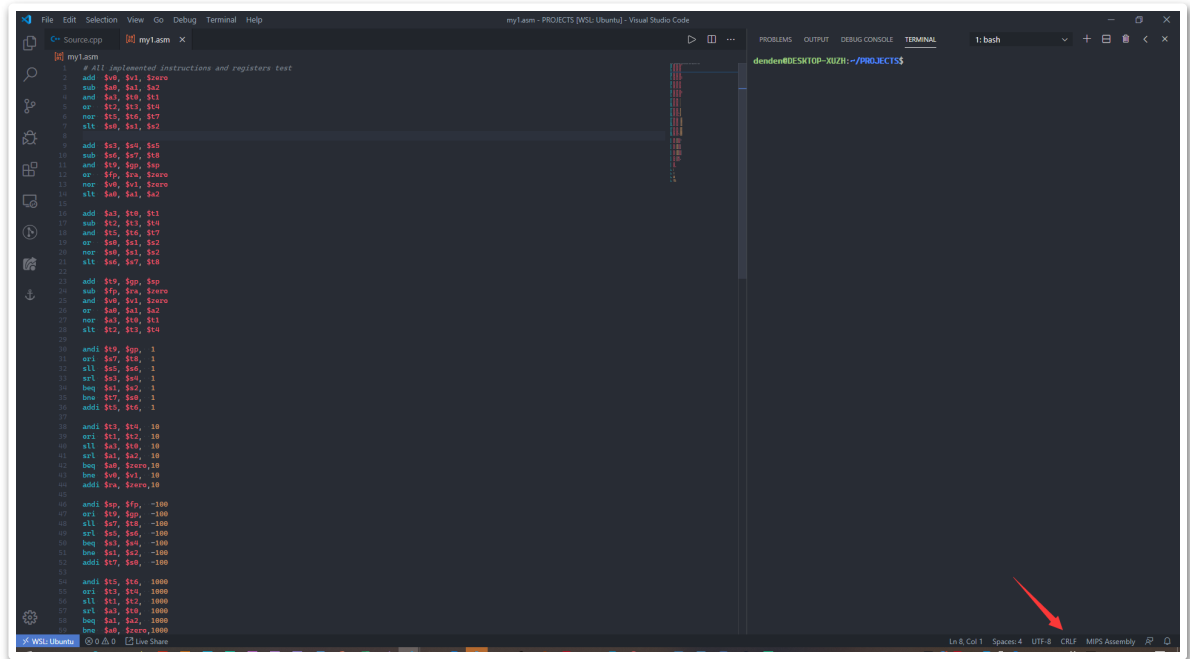
```
PROJ > CC++ > MIPS > MIPS > my1.txt
31 1078:00100001 10101110 00000000 00000001 001000 01101 01110 00000 00000 000001
32 107c:00110001 01101100 00000000 00001010 001100 01011 01100 00000 00000 001010
33 1080:00110101 00101010 00000000 00001010 001101 01001 01010 00000 00000 001010
34 1084:00000000 00000111 01000010 10000000 000000 00000 00111 01000 01010 000000
35 1088:00000000 00000101 00110010 10000010 000000 00000 00101 00110 01010 000010
36 108c:00010000 10000000 00000000 00001010 000100 00100 00000 00000 00000 001010
37 1090:00010100 01000011 00000000 00001010 000101 00010 00011 00000 00000 001010
38 1094:00100011 11100000 00000000 00001010 001000 11111 00000 00000 00000 001010
39 1098:00110011 10111110 11111111 10011100 001100 11101 11110 11111 11110 011100
40 109c:00110111 00111100 11111111 10011100 001101 11001 11100 11111 11110 011100
41 10a0:00000000 00010111 11000111 00000000 000000 00000 10111 11000 11100 000000
42 10a4:00000000 00010101 10110111 00000010 000000 00000 10101 10110 11100 000010
43 10a8:00010010 01110100 11111111 10011100 000100 10011 10100 11111 11110 011100
44 10ac:00010110 00110100 11111111 10011100 000101 10001 10010 11111 11110 011100
45 10b0:00100001 11110000 11111111 10011100 001000 01111 10000 11111 11110 011100
46 10b4:00110001 10101110 00000011 11101000 001100 01101 01110 00000 01111 101000
47 10b8:00110101 01101100 00000011 11101000 001101 01011 01100 00000 01111 101000
48 10bc:00000000 00001001 01010010 00000000 000000 00000 01001 01010 01000 000000
49 10c0:00000000 00000111 01000010 00000010 000000 00000 00111 01000 01000 000010
50 10c4:00010000 10100110 00000011 11101000 000100 00101 00110 00000 01111 101000
51 10c8:00010100 10000000 00000011 11101000 000101 00100 00000 00000 01111 101000
52 10cc:00100000 01000011 00000011 11101000 001000 00010 00011 00000 01111 101000
53 10d0:10101100 01000011 00000000 00001010 101011 00010 00011 00000 00000 001010
54 10d4:10001100 10000000 00000000 00001010 100011 00100 00000 00000 00000 001010
55 10d8:10000000 10100110 00000000 00001010 100000 00101 00110 00000 00000 001010
56 10dc:10100000 11101000 00000000 00001010 101000 00111 01000 00000 00000 001010
57 10e0:10101101 00101010 00000000 01100100 101011 01001 01010 00000 00001 100100
58 10e4:10001101 01101100 00000000 01100100 100011 01011 01100 00000 00001 100100
59 10e8:10000001 10101110 00000000 01100100 100000 01101 01110 00000 00001 100100
60 10ec:10100001 11110000 00000000 01100100 101000 01111 10000 00000 00001 100100
61 10f0:10101110 00110010 00000011 11101000 101011 10001 10010 00000 01111 101000
62 10f4:10001110 01110100 00000011 11101000 100011 10011 10100 00000 01111 101000
63 10f8:10000010 10110110 00000011 11101000 100000 10101 10110 00000 01111 101000
64 10fc:10100010 11111000 00000011 11101000 101000 10111 11000 00000 01111 101000
65 1100:10101111 00111100 00000000 00000001 101011 11001 11100 00000 00000 000001
66 1104:10001111 10111110 00000000 00000001 100011 11101 11110 00000 00000 000001
67 1108:10000011 11100000 00000000 00000001 100000 11111 00000 00000 00000 000001
68 110c:10100000 01000011 00000000 00000001 101000 00010 00011 00000 00000 000001
69 1110:00000000 01000000 00000000 00001000 000000 00010 00000 00000 00000 001000
70 1114:00000010 00000000 00000000 00001000 000000 10000 00000 00000 00000 001000
71 1118:00000001 00000000 00000000 00001000 000000 01000 00000 00000 00000 001000
72 111c:00000000 00000000 00000000 00001000 000000 00000 00000 00000 00000 001000
73 1120:00001000 00000000 00000000 00000001 000010 00000 00000 00000 00000 000001
74 1124:00001100 00000000 00000000 00000010 000011 00000 00000 00000 00000 000010
75 1128:00001011 11111111 11111111 11111101 000010 11111 11111 11111 11111 111101
76 112c:00001111 11111111 11111111 11111100 000011 11111 11111 11111 11111 111100
77 1130:00001000 00000000 00000000 00000001 000010 00000 00000 00000 00111 110100
78 1134:00001100 00000000 00000010 01011000 000011 00000 00000 00000 01001 011000
79 1138:00001000 00000000 00000011 11101000 000010 00000 00000 00000 01111 101000
80 113c:00001111 11111111 11111100 00011000 000011 11111 11111 11111 10000 011000
81
```

各种错误情况

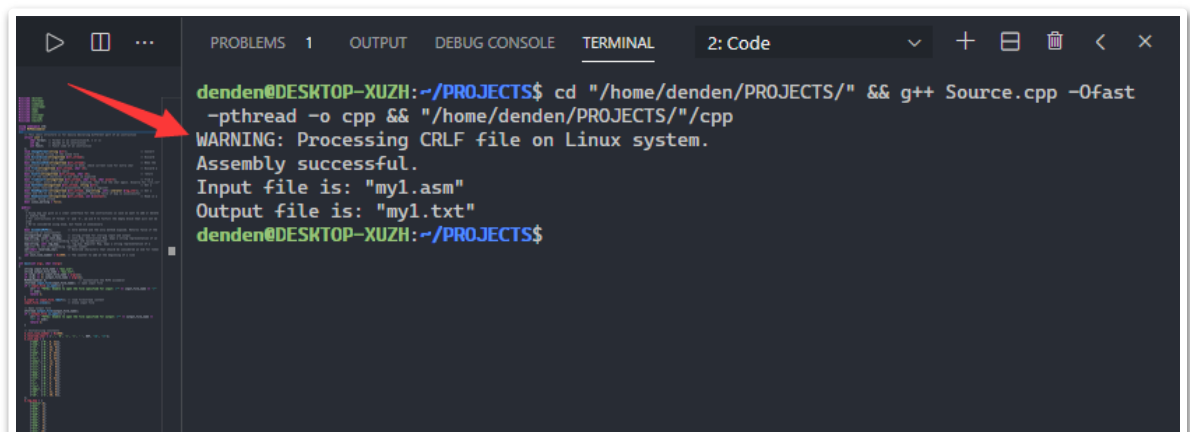
Linux下使用Windows的 [CRLF](#) 换行方式

等级：WARNING

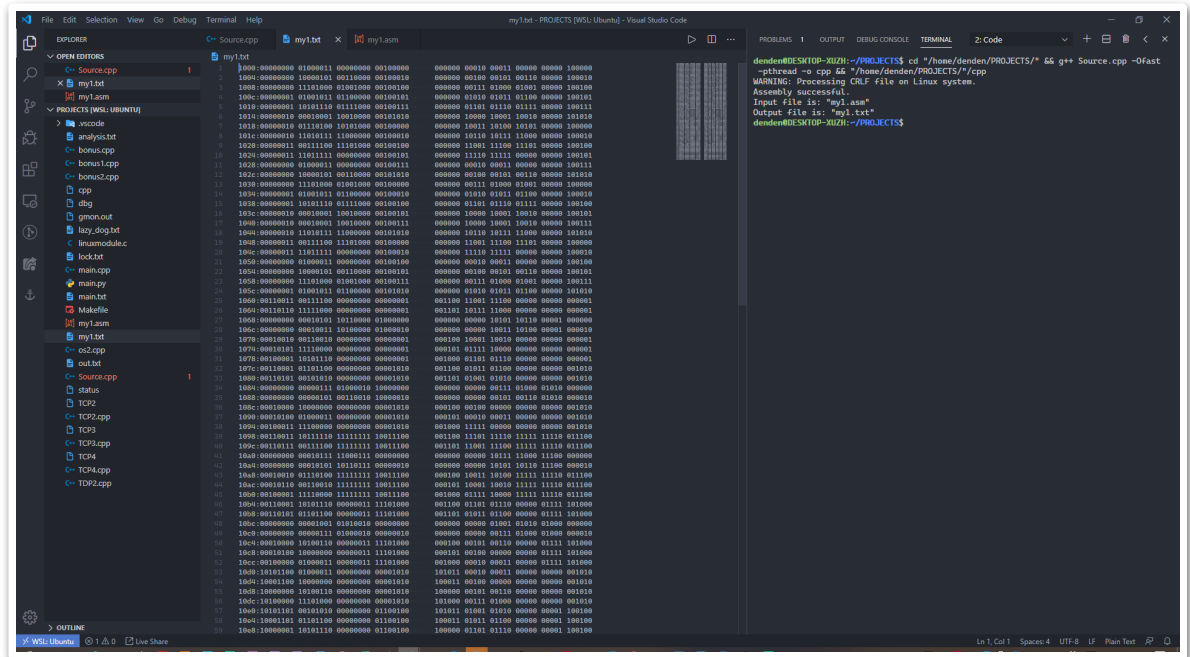
输入文件内容



控制台输出



进行汇编后的文件内容



错误 (不规范) 的语法

各类警告错误的列举

```

1      # Implemented comment character '#' for marker in the asm file
2      # When '#' is encountered, the entire line will be ignored
3      # Tester for irregular input, like trailing space or space every where
4      add $s0,$s1,$s2
5      add $s0, $s1, $s2
6      add $s0, $s1, $s2
7      add $s0,          $s1, $s2
8      add $s0, $s1, $s2
9      add $s0, $s1, $s2
10     lw $t0, 100(          $t2          )
11     lw $t0,    100( $t2 )
12     lw $t0, 100( $t2 )
13     lb $t0, 100( $t2 )
14     sw $t0, 100( $t2 )
15     lb $t0, 100( $t2 )
16     sb $t0, 100( $t2 )
17     jr $ra
18     jr $ra
19     jr $ra
20     j 10000
21     jal 100
22     addi $v0, $v1, 10000 # This is some inline comment, please ignore me
23     sll $t1, $t2, 10
24     srl $t1, $t2, 10

```

输入文件的截图

```
PROJ > CC++ > MIPS > MIPS > my1.asm
You, a few seconds ago | 1 author (You)
1  # Implemented comment character '#' for marker in the asm file
2  # When '#' is encountered, the entire line will be ignored
3  # Tester for irregular input, like trailing space or space every where
4  add $s0, $s1, $s2
5  add $s0, $s1, $s2
6  add $s0, $s1, $s2
7  add $s0, $s1, $s2
8  add $s0, $s1, $s2
9  add $s0, $s1, $s2
10 lw $t0, 100($t2)
11 lw $t0, 100($t2)
12 lw $t0, 100($t2)
13 lb $t0, 100($t2)
14 sw $t0, 100($t2)
15 lb $t0, 100($t2)
16 sb $t0, 100($t2)
17 jr $ra
18 jr $ra
19 jr $ra
20 j 10000
21 jal 100
22 addi $v0, $v1, 10000 # This is some inline comment, please ignore me
23 sll $t1, $t2, 10
24 srl $t1, $t2, 10
25
```

输出文件的截图

```
PROJ > CC++ > MIPS > MIPS > my1.txt
You, 21 hours ago | 1 author (You)
1  1000:00000010 00010001 10010000 00100000 000000 10000 10001 10010 00000 100000
2  1004:00000010 00010001 10010000 00100000 000000 10000 10001 10010 00000 100000
3  1008:00000010 00010001 10010000 00100000 000000 10000 10001 10010 00000 100000
4  100c:00000010 00010001 10010000 00100000 000000 10000 10001 10010 00000 100000
5  1010:00000010 00010001 10010000 00100000 000000 10000 10001 10010 00000 100000
6  1014:00000010 00010001 10010000 00100000 000000 10000 10001 10010 00000 100000
7  1018:10001101 00001010 00000000 01100100 100011 01000 01010 00000 00001 100100
8  101c:10001101 00001010 00000000 01100100 100011 01000 01010 00000 00001 100100
9  1020:10001101 00001010 00000000 01100100 100011 01000 01010 00000 00001 100100
10 1024:10000001 00001010 00000000 01100100 100000 01000 01010 00000 00001 100100
11 1028:10101101 00001010 00000000 01100100 101011 01000 01010 00000 00001 100100
12 102c:10000001 00001010 00000000 01100100 100000 01000 01010 00000 00001 100100
13 1030:10100001 00001010 00000000 01100100 101000 01000 01010 00000 00001 100100
14 1034:00000011 11100000 00000000 00001000 000000 11111 00000 00000 00000 001000
15 1038:00000011 11100000 00000000 00001000 000000 11111 00000 00000 00000 001000
16 103c:00000011 11100000 00000000 00001000 000000 11111 00000 00000 00000 001000
17 1040:00001000 00000000 00100111 00010000 000010 00000 00000 00100 11100 010000
18 1044:00001100 00000000 00000000 01100100 000011 00000 00000 00000 00001 100100
19 1048:00100000 01000011 00100111 00010000 001000 00010 00011 00100 11100 010000
20 104c:00000000 00001001 01010010 10000000 000000 00000 01001 01010 01010 000000
21 1050:00000000 00001001 01010010 10000000 000000 00000 01001 01010 01010 000010
22
```

各类致命错误语法的列举

- 1 # Extra comma
- 2 # add \$s0, \$s1, \$s2,
- 3 # add \$s0, \$s1,, \$s2
- 4
- 5 # Unrecognized instruction
- 6 # abc \$s0, \$s1, \$s2
- 7
- 8 # Unrecognized register

```

9      # add $abc,$s1, $s2
10
11     # Wrong syntax(Unable to match with already defined syntax)
12     # add  $v0, $v1, $zero,
13     # add  $v0, $v1
14     # add  $v0, $v1, 100
15     # add  100, $v0, $v1
16     #
17     # andi $t9, $gp, 100,
18     # andi $t9, $gp,
19     # andi $t9, $gp, $s0
20     # andi $t9, 100
21     # andi 100, $t1
22     #
23     # sw  $v0, 10($v1
24     # sw  $v0, 10$v1)
25     # sw  $v0, 10($v1),
26     # sw  $v0, 10($v1), 10
27     #
28     # j    1,
29     # j    $v1
30     # j    ,
31     # j    1,1
32     #
33     # jr   $v0,
34     # jr   0
35     # jr   $v0, 10
36     # jr   $v0, $v1

```

- 额外的逗号


```
Source_new.cpp my1.txt myvue.css [10] my1.asm X
PROJ > CC++ > MIPS > MIPS > [10] my1.asm
You, a few seconds ago | 1 author (You)
1 # Implemented comment for marker in the asm file
2 # The following is some error case we've implemented
3 # You can uncomment some of them to test the result
4
5 # Extra comma
6 add $s0, $s1, $s2, You, a few seconds ago • Uncommitted cha
7 # add $s0, $s1,, $s2
8
9 # Unrecognized instruction
10 # abc $s0, $s1, $s2
11
12 # Unrecognized register
13 # add $abc,$s1, $s2
14
15 # Wrong syntax
16 # add $v0, $v1, $zero,
17 # add $v0, $v1
18 # add $v0, $v1, 100
19 # add 100, $v0, $v1
20 #
21 # andi $t9, $gp, 100,
22 # andi $t9, $gp,
23 # andi $t9, $gp, $s0
24 # andi $t9, 100
25 # andi 100, $t1
26 #
27 # sw $v0, 10($v1
28 # sw $v0, 10$v1)
29 # sw $v0, 10($v1),
30 # sw $v0, 10($v1), 10
31 #
32 # j $s1,
33 # j $s1, $v1
34 # j $s1,
35 # j $s1, 1, 1
36 #
37 # jr $v0,
38 # jr $s0
39 # jr $v0, 10
40 # jr $v0, $v1
41
```

```
D:\Documents\Materials\DENDEN\PROJ\CC++\MIPS\MIPS>MIPS
FATAL: Unexpected line end.
FATAL: Illegal instruction or format. Check your syntax.
DUMP: Current line of input stringstream is:
add $s0, $s1, $s2,
DUMP: Current line number is: 6
DUMP: Current buffer of output stringstream is:
1000:00000010000100011001000000100000
D:\Documents\Materials\DENDEN\PROJ\CC++\MIPS\MIPS>
```

```

You, a few seconds ago | 1 author (You)
1  # Implemented comment for marker in the asm file
2  # The following is some error case we've implemented
3  # You can uncomment some of them to test the result
4
5  # Extra comma
6  # add $s0, $s1, $s2,
7  add $s0, $s1,, $s2
8

```

```

D:\Documents\Materials\DENDEN\PROJ\CC++\MIPS\MIPS>MIPS
FATAL: Unable to map register:  to int value.
FATAL: Illegal instruction or format. Check your syntax.
DUMP: Current line of input stringstream is:
add $s0, $s1,, $s2
DUMP: Current line number is: 7
DUMP: Current buffer of output stringstream is:
1000:0000001000010001

D:\Documents\Materials\DENDEN\PROJ\CC++\MIPS\MIPS>

```

- 无法识别的指令

```

You, a few seconds ago | 1 author (You)
1  # Implemented comment for marker in the asm file
2  # The following is some error case we've implemented
3  # You can uncomment some of them to test the result
4
5  # Extra comma
6  # add $s0, $s1, $s2,
7  # add $s0, $s1,, $s2
8
9  # Unrecognized instruction
10 abc $s0, $s1, $s2
11
12 # Unrecognized register
13 # add $abc, $s1, $s2

```

```

D:\Documents\Materials\DENDEN\PROJ\CC++\MIPS\MIPS>MIPS
FATAL: Unable to map instruction: "abc" in instruction map.
FATAL: Illegal instruction or format. Check your syntax.
DUMP: Current line of input stringstream is:
abc $s0, $s1, $s2
DUMP: Current line number is: 10
DUMP: Current buffer of output stringstream is:
1000:

D:\Documents\Materials\DENDEN\PROJ\CC++\MIPS\MIPS>

```

- 无法识别的寄存器


```

1  # Implemented comment for marker in the asm file
2  # The following is some error case we've implemented
3  # You can uncomment some of them to test the result
4
5  # Extra comma
6  # add $s0, $s1, $s2,
7  # add $s0, $s1,, $s2
8
9  # Unrecognized instruction
10 # abc $s0, $s1, $s2
11
12 # Unrecognized register
13 add $abc,$s1, $s2

```

```

D:\Documents\Materials\DENDEN\PROJ\CC++\MIPS\MIPS>MIPS
FATAL: Unable to map register: $abc to int value.
FATAL: Illegal instruction or format. Check your syntax.
DUMP: Current line of input stringstream is:
add $abc,$s1, $s2
DUMP: Current line number is: 13
DUMP: Current buffer of output stringstream is:
1000:0000000
D:\Documents\Materials\DENDEN\PROJ\CC++\MIPS\MIPS>

```

- 错误的指令格式 (部分)

```

12 # Unrecognized register
13 # add $abc,$s1, $s2
14
15 # Wrong syntax
16 # add $v0, $v1, $zero,
17 # add $v0, $v1
18 add $v0, $v1, 100
19 # add 100, $v0, $v1
20 #
21 # andi $t9, $gp, 100,
22 # andi $t9, $gp,

```

```

D:\Documents\Materials\DENDEN\PROJ\CC++\MIPS\MIPS>MIPS
FATAL: Unable to map register: 100 to int value.
FATAL: Illegal instruction or format. Check your syntax.
DUMP: Current line of input stringstream is:
# add 100, $v0, $v1
DUMP: Current line number is: 18
DUMP: Current buffer of output stringstream is:
1000:0000000001000011
D:\Documents\Materials\DENDEN\PROJ\CC++\MIPS\MIPS>

```

```

# Wrong syntax
# add $v0, $v1, $zero,
# add $v0, $v1
# add $v0, $v1, 100
# add 100, $v0, $v1
#
# andi $t9, $gp, 100,
# andi $t9, $gp,
andi $t9, $gp, $s0
# andi $t9, 100
# andi 100, $t1
#
# sw $v0, 10($v1
# sw $v0, 10$v1)
# sw $v0, 10($v1),
# sw $v0, 10($v1), 10
#

```

```

D:\Documents\Materials\DENDEN\PROJ\CC++\MIPS\MIPS>MIPS
FATAL: Unable to read the constant.
FATAL: Illegal instruction or format. Check your syntax.
DUMP: Current line of input stringstream is:
andi $t9, $gp, $s0
DUMP: Current line number is: 23
DUMP: Current buffer of output stringstream is:
1000:0011001100111100

```

```

D:\Documents\Materials\DENDEN\PROJ\CC++\MIPS\MIPS>

```

```

24 # andi $t9, 100
25 # andi 100, $t1
26 #
27 sw $v0, 10($v1
28 # sw $v0, 10$v1)
29 # sw $v0, 10($v1),
30 # sw $v0, 10($v1), 10
31 #
32 # j 1,
33 # j $v1

```

```

D:\Documents\Materials\DENDEN\PROJ\CC++\MIPS\MIPS>MIPS
FATAL: Unable to read the constant.
FATAL: Illegal instruction or format. Check your syntax.
DUMP: Current line of input stringstream is:
andi $t9, $gp, $s0
DUMP: Current line number is: 23
DUMP: Current buffer of output stringstream is:
1000:0011001100111100

```

```

D:\Documents\Materials\DENDEN\PROJ\CC++\MIPS\MIPS>

```

```

25 # andi 100, $t1
26 #
27 # sw $v0, 10($v1
28 sw $v0, 10$v1) You, a
29 # sw $v0, 10($v1),
30 # sw $v0, 10($v1), 10
31 #
32 # j 1,

```

```
D:\Documents\Materials\DENDEN\PROJ\CC++\MIPS\MIPS>MIPS
FATAL: Unable to read the character: (
FATAL: Illegal instruction or format. Check your syntax.
DUMP: Current line of input stringstream is:
sw $v0, 10$v1)
DUMP: Current line number is: 28
DUMP: Current buffer of output stringstream is:
1000:10101100010
```

```
D:\Documents\Materials\DENDEN\PROJ\CC++\MIPS\MIPS>
```

```
31 #
32 # j--,1,
33 | j--, $v1, You,
34 # j--,
35 # j--,1,1
36 #
37 # jr--, $v0,
```

```
D:\Documents\Materials\DENDEN\PROJ\CC++\MIPS\MIPS>MIPS
FATAL: Unable to read the constant.
FATAL: Illegal instruction or format. Check your syntax.
DUMP: Current line of input stringstream is:
j $v1
DUMP: Current line number is: 33
DUMP: Current buffer of output stringstream is:
1000:000010
```

```
D:\Documents\Materials\DENDEN\PROJ\CC++\MIPS\MIPS>
```

```
34 # j--,
35 # j--,1,1
36 #
37 # jr--, $v0,
38 # jr--, 0
39 | jr--, $v0, 10, You, a fe
40 # jr--, $v0, - $v1
```

```
D:\Documents\Materials\DENDEN\PROJ\CC++\MIPS\MIPS>MIPS
FATAL: Unexpected line end.
FATAL: Illegal instruction or format. Check your syntax.
DUMP: Current line of input stringstream is:
jr $v0, 10
DUMP: Current line number is: 39
DUMP: Current buffer of output stringstream is:
1000:000000000010000000000000000000001000
```

```
D:\Documents\Materials\DENDEN\PROJ\CC++\MIPS\MIPS>
```

5. 附录：实现框图

