计算机系统综合设计

设计报告

组长：唐兴盛

成员：吴哲凯

荆　宁

丁富来

汤思彦

东南大学计算机科学与工程学院

二00九年一月

设计名称	东大之芯嵌入式系统				
完成时间	2009-1-5	验收时间		成绩	

本组成员情况			
姓　名	学　号	承　担　的　任　务	个人成绩
唐兴盛	09005309	流水线 CPU、MiniC 编译器	
吴哲凯	09005328	流水线 CPU、汇编器	
荆　宁	09005323	集成开发环境 MiniSys IDE	
丁富来	09005324	汇编器	
汤思彦	09005336	外设	

注：本设计报告中各个部分如果页数不够，请大家自行扩页，原则是一定要把报告写详细，能说明本组设计的成果和特色，能够反应小组中每个人的工作。报告中应该叙述设计中的每个模块。设计报告将是评定每个人成绩的重要组成部分之一，因此要在报告中明确标明每个模块的设计者。

设计报告最后一页是验收表和教师综合评价，请大家打印报告的时候将此页一并打印装订。

本组设计的功能描述（含所有实现的模块的功能）

我们小组在本课程设计中一共完成了32位流水CPU及定时计数器,看门狗、PWM、UART、中断控制器这些外设，带全屏编辑功能的汇编器、Mini-C 编译器的 IDE 环境开发。并实现了系统的整体联合调试，以及下载调试。功能完全正常。

本组设计的主要特色

1.实现了 32 位 5 级流水结构的 CPU
2.实现了具有强大错误检测功能的汇编器
3.实现了 Mini-C 编译器
4.重写了 ALU 中的移位器
5.使用结构描述重新设计了寄存器组
6.完善地处理了流水中断及其嵌套的处理
7.整合了汇编器与编译器，设计出了一个集成开发环境 MiniSys IDE
8. MiniSys IDE 支持关键字高亮，并实现了与 Quartus 的无缝连接

本组设计的体系结构

I.32 位 5 级流水 CPU

系统的 CPU 采用了 MIPS 的经典 5 级流水结构。流水段分为 IF 取指、ID 译码、EXE 执行、MEM 内存读写、WB 寄存器回写。流水线设计关键是解决相关性。系统采用的是哈佛结构，所以不存在结构相关。主要解决的是数据相关和控制相关。解决数据相关采用了转发策略，将以后要写到寄存中的数据提前送到需要的地方；解决控制相关采取了阻塞策略。

II.汇编器

汇编器主要由四个部分组成：词法分析、语法分析、机器码翻译、错误检测。其中语法分析是核心。具体结构图 1 所示。

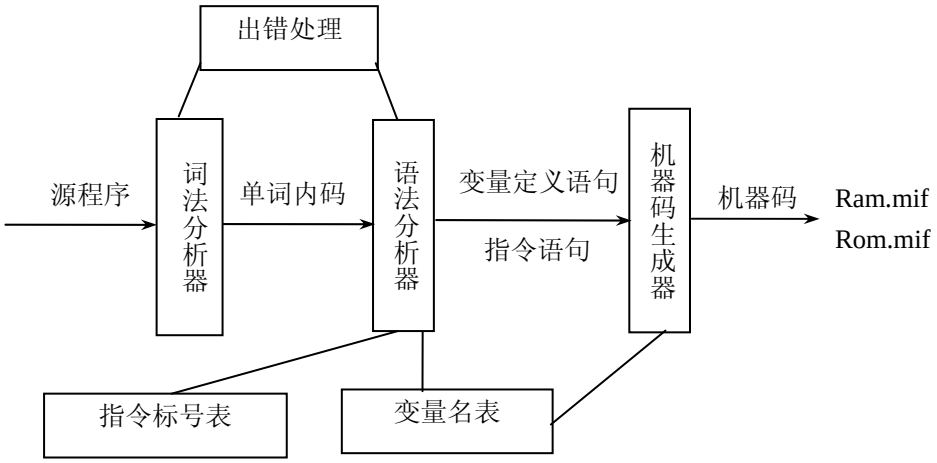

图 1 汇编器结构

词法分析器负责按照 MiniSys 汇编语言词法，组字符成词；语法分析器依据 MiniSys 汇编语言语法，组词成句；机器码生成器负责依据翻译规则（包括指令翻译和数据翻译），将一个完整的指令句子或变量定义语句转换为相应的机器指令。

词法分析器没有用自动机，而是采用了以下策略。我们用的是 LL（1）文法，所以每次读入一个单词前，基本已经知道了单词的类型。词法分析时，直接读入一个字符串，然后进行字符串匹配。比如通过语法分析，知道接下来是“ORG_CODE”。词法分析时只要把读入的字符串同“ORG_CODE”进行匹配，如果一样证明词法分析正确，否则就是出错。出错后，调用相应的错误处理程序，处理完后继续进行语法分析。

机器码的生成伴随着词法分析和语法分析。如果前一个单词识别正确，则进行翻译。否则跳过不进行翻译。

汇编器的关键是语法规则的定义，语法分析和词法分析都是严格按照语法规则进行的。

III.Mini-C 编译器

MiniC 编译器主要由以下几个部分组成,词法分析器，语法制导分析筐架（含中间代码生成）,寄存器分配模块，函数块翻译模块，代码综合输出模块。结构示意如图 2 所示。

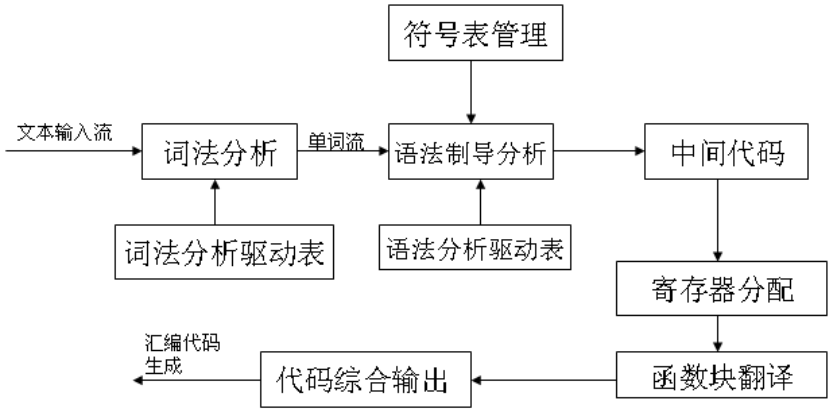

图 2 编译器结构

词法分析器在词法分析表的驱动下，从输入流中识别符号，为语法分析提供单词流。语法分析器采用了 LALR(1)分析法，并采用语法指导的方式进行中间代码的生成，符号表管理模块为语法分析过程提供单词登记，查询的功能，并参与变量的空间分配计算，为后续的代码生成,提供地址信息。寄存器分配部分负责对翻译过程中产生的临时变量进行寄存器关联，采用的策略是简单统计各个临时变量的使用频率，为使用最频繁的前 10 个临时变量分配寄存器，对超出部分的寄存器，为其分配内存空间。函数块的翻译负责将中间代码生成部分产生的各个函数的中间代码翻译为汇编代码，并加上相应的现场保护，现场恢复的代码，完成子程序的翻译。最后的代码综合输出主要作用是，将各个子程序进行总装，加上中断入口，程序初始化代码，产生完整的一个汇编代码，完成翻译。产生的汇编代码只要通过 Minisys 的汇编器翻译，就可以生成最终的机器代码，进入 CPU 运行。

本组设计中各个部件的设计与特色概述

I.流水处理器部分

- **总体设计**

系统的 CPU 采用了 MIPS 的经典 5 级流水结构。流水段分为 IF 取指、ID 译码、EXE 执行、MEM 内存读写、WB 寄存器回写。流水线设计关键是解决相关性。系统采用的是哈佛结构，所以不存在结构相关。主要解决的是数据相关和控制相关。解决数据相关采用了转发策略，将以后要写到寄存中的数据提前送到需要的地方；解决控制相关采取了阻塞策略。

- **功能介绍**

- 与 MIPS 指令集兼容的 32 位流水 CPU
- 两级外部中断控制器

- **主要模块实现技术**

- 中断控制器

流水线的中断具有一定的复杂性，本设计针对不同情况采取了相应的策略实现了精确中断。中断的检测在时钟下降沿完成。

（1）非转移指令的中断处理。此类情形的中断时最简单那的，只需要在中断返回地址寄存器中写入下一条指令的地址，同时输出新 PC 为中断入口地址，在下一时钟清除 IF/ID 流水寄存器。但此类情况有个复杂点的问题，下一周期时,IF/ID 被清除，此时高优先级的中断将当前的低优先级中断中断了，当保存的中断返回地址为低优先级的中断的入口地址。所以，设计时，要向 ID/EXE 流水寄存器传递当前是否有低优先级的中断发生。

（2）转移指令的中断处理。对于转移发生前，只需要保存转移的目的地址，同时设置新的 PC 值，清除 IF/ID 寄存器。对于下一个时钟周期，由于 IF/ID 被清除，此时发生中断嵌套，则类似于 a 中的处理方法，利用 ID/EXE 保存的上个时钟的信息，保存低优先级中断的入口地址。

（3）被阻塞的指令。中断发生时，中断控制器会检测是否有阻塞信号，并自动延迟 1 个时钟周期执行中断过程。

- 寄存器模块

寄存器组位宽为 32 位，采用 D 触发器组建。由地址译码器、32 位寄存器组、输出多路开关、转发器多路开关组成。寄存器文件提供寄存器的读写以及前向数据的转发功能，是处理器的核心部件之一。为了解决结构相关性，规定寄存器组的写入发生在时钟下降沿。寄存器组 RTL 图如图 3 所示。

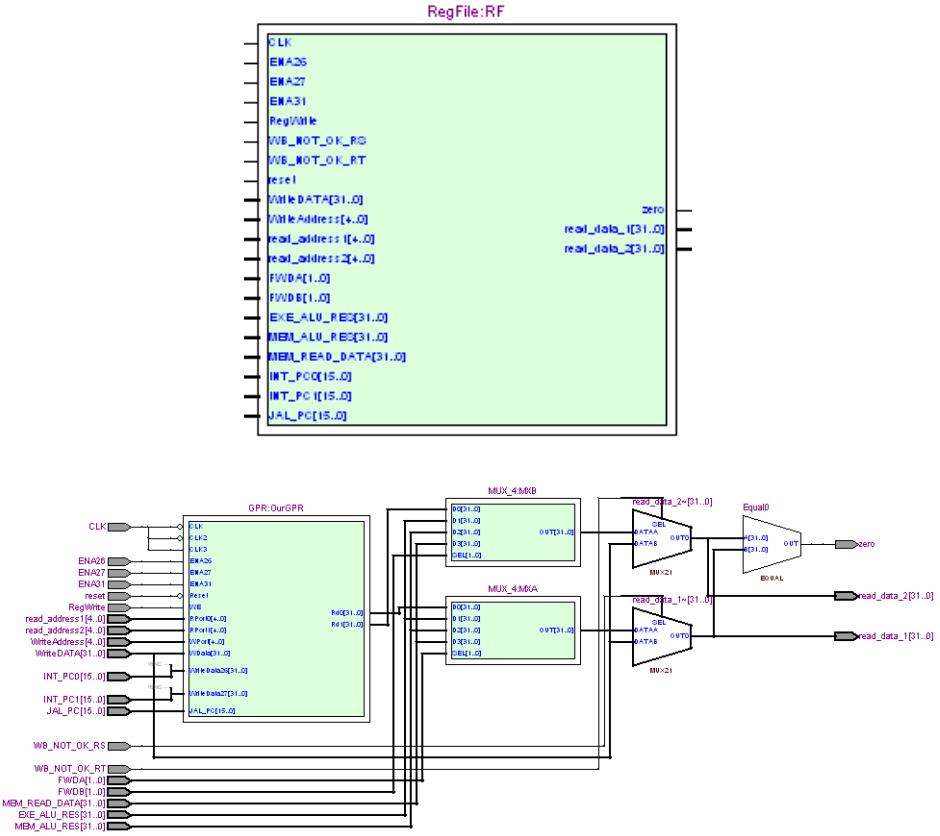

图 3 寄存器组的 RTL 图

- 译码单元

采用了组合电路译码方式，产生相关控制信号、扩展 16 位立即数到 32 位、分析数据相关性、指导数据流的转发。为了消除对 ALU 造成的结构相关性，PC 和 MEM/IO 地址的计算，都在译码单元中完成。这样可以提高转移指令的效率。译码单元是整个 CPU 的核心部分，CPU 的正常运行完全靠译码单元的控制和协调。

- 执行单元

执行单元的设计至关重要，其性能的高低直接决定 CPU 的延迟和主频。为了尽可能避免触发器造成的延迟，采用了纯组合逻辑设计。运算部件采用高效结构，比如 32 位组内并行组间也并行的加法器、桶形移位器（见图 4）等。以此尽可能的减少延迟。

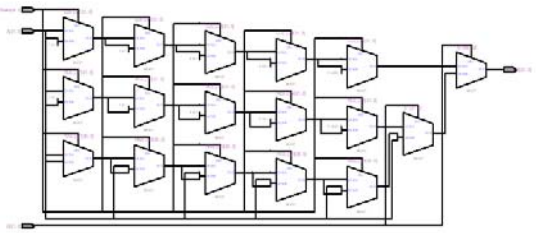

图 4 桶形移位器

- MEM/IO 单元

内存使用了 Altera 公司提供的宏模块实现，使用的是 lpm_ram 宏模块。在时钟的下降沿完成 MEM 的写入和读出。

I/O 与 MEM 采用统一编址，由 MEMIO 模块负责 MEM/IO 地址译码，产生 MEM 和 IO 的选通信号。同时，该模块产生写入数据，送往 MEM 和 IO 部件。为了整合 MEM 和 IO 的读出数据，IOREAD 模块根据内存和 IO 的读写信号以及地址信号，对 MEM 和 IO 的读出数据进行选择，从而实现读出数据的过滤。IOREAD 的 RTL 结构如图 5 所示。

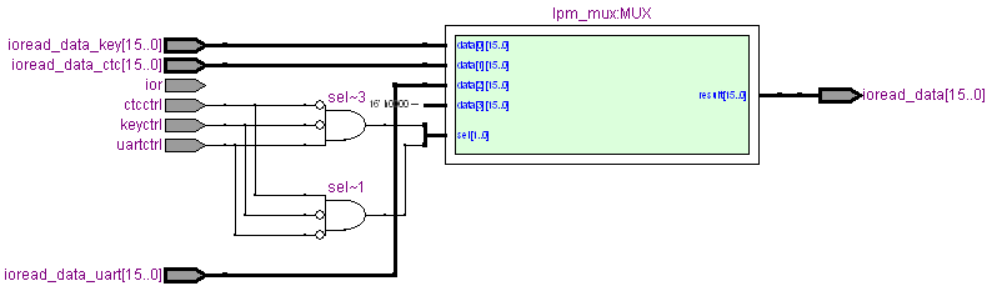

图 5　IOREAD 的 RTL 结构

接口部件

➢ 总体设计

在计算机系统中，外设不会直接挂在系统总线或 CPU 上，这是因为外设在信号种类、时序等方面往往会有差别，所以需要提供相应的接口电路来完成与系统总线之间的各种转换。各种接口电路中，需要设计供 CPU 直接存取访问的寄存器或特定电路，称之为 I/O 端口。

I/O 地址空间的设计有两种方案，一种是独立编址、另一种是统一编址。考虑到指令系统中没有专门的 I/O 指令，只能使用 LW 和 SW 两条指令进行 RAM 访问和 I/O 访问，所以采用了 I/O 统一编址方式。系统中 RAM 与 I/O 地址空间分配如图 6 所示。

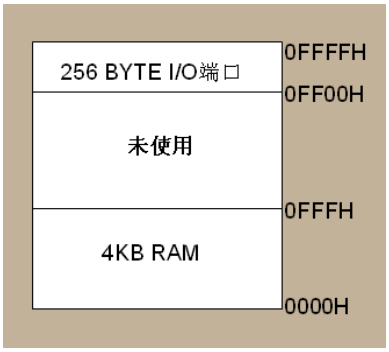

图 6 RAM 与 I/O 地址空间分配

➢ 各模块用途简介

- 7 段 LED 是系统中最常用的输出设备
- 4×4 键盘是系统中最常用的输入设备
- 16 位定时器可以用来输出时钟中断
- PWM 控制器用来控制模拟电路
- 32 位并口用来模拟彩灯效果
- 看门狗控制器用来使失控的程序恢复正常
- 32 位内部定时器提供内部定时

➢ **各模块具体功能**

- 4 位 7 段 LED 控制器

数码管有两种，共阳极数码管和共阴极数码管。系统针对共阴极数码管进行设计。即数据位为“1”时，数码管亮，为“0”时灭。

系统中有 4 个 7 段数码管，可用来表示 4 位 10 进制或 4 位 16 进制数。每半个字节为 1 组，从 16 位数据线的高位到低位共分成 4 组，分别表示 LED3、LED2、LED1 和 LED0。

模块中含有一个 16 位寄存器，保存显示到数码管上的数值。端口地址为 0FF00H。

- 4×4 键盘控制器

为了减少连线个数，节省布线面积，系统采用了矩阵键盘形式。键盘控制器的功能为自动扫描 4×4 键盘，当有键按下时扫描键值，将键值记录到键值寄存器中。

键盘的扫描算法是：让所有行线全为了 0，读出所有列线状态，如果有列线为 0，则说明有键按下；然后从第 0 行开始扫描每一行，令该行所对应的行线为 0，其余行线为 1；然后读列线，如果有一列为 0，则该行该列交叉处的键被按下；如果读出所有列都为 1，则行号加 1，并将下一行线清 0，其他行线置 1，顺序扫描下一行。扫描到按键后，将行线和列线进行编码，输出键值。MiniSys 成功的用电路实现了这一键盘扫描算法。

模块中含有一个 4 位寄存器，用来保存当前键值。端口地址为 0FF10H。

- 16 位定时/计数器

系统带两个定时/计数器，采用了加 1 计数器。支持重复定时/计数。定时方式中，计数到设定值输出一个高电平。默认情况下， SCT2 的定时输出接 0 号中断（时钟中断）。计数方式中，计数到设定值时设置状态寄存器的相应位。

模块中含有四个 16 位的寄存器，用来保存当前状态值或当前计数值。端口地址为 0FF20H、0FF22H、0FF24H、0FF26H。

- 16 位 PWM 控制器

控制器内部有一个 16 位计数器和一个 16 位对比值，计数器周而复始的加 1 计数，计数到最大值时跳变为 0 再计算。当计数器的值低于对比值时输出低电平，否则输出高电平。不难发现，对比值的不同决定了输出脉冲的占空比。当然计数器的最大值可以由程序设置。

模块中含有三个 16 位的寄存器，分别用来保存计数器的最大值、计数器的对比值、计数器的使能信息。端口地址分别为 0FF30H、0FF32H、0FF34H。

- 看门狗控制器

模块中含有一个 16 位定时器，系统复位后计数值为 0FFFFH，之后计数值减 1，当减到 0 的时候，向 CPU 发送 4 个时期的 RESET 信号，同时计数器复位到 0FFFFH 并继续计数。可

以通过程序向看门狗端口发写信号（俗称“喂狗”）来复位看门狗，使计数器重新开始计数。模块的端口地址为 0FF50H。

- 32 位并口控制器

  本模块很简单，就是含一个 32 位的寄存器。可以读写这个寄存器，端口地址为 0FF70H。

➢ 设计难点及解决方案

- 流水线 CPU 设计的问题和解决策略

（1）数据相关性

MiniSys 的设计采用了数据定向转发技术来解决数据的相关性。ID 段含两个四选一的数据选择通路，以此实现数据的定向转发。数据选择通路的四个输入是 EXE 段的 ALU 运算结果、MEM 段保存的 ALU 运算结果、MEM 段读出的数据、寄存器的回写数据。根据 ID 段产生的控制信号，选择其中的一条通路作为 EXE 的运算数据输入。除了一种特殊的情况，其他的数据相关都可以通过定向转发解决。对于 EXE 段指令为 lw，而 ID 段的指令需要 lw 读出的内存值。此时，内存的数据只能在下一个时钟周期才能被读出来，因此 ID 段和 IF 段需要阻塞一个时钟周期，同时在下一个周期开始时清除 ID 段流向 EXE 的数据和控制信号，以等待内存数据的读出。

（2）控制相关性

对于 jr、j 指令，一定会发生转移，因此在 ID 段就可以计算出新 PC 值。对于 jal,在 ID 段就写入$ra 寄存器。同时，应当在下一时钟周期清除 IF/ID 段的流水寄存器。

对于 bne,beq 指令,通过在 ID 段判断寄存器的内容，确定转移是否发生，控制新 PC 的产生。由于可能发生和 EXE 段的 lw 相关，加上转移发生要清除一段流水寄存器，其最大可能造成两段的空隙，这是 bne,beq 和 j,jal 的不同。

（3）结构相关

由于计算 PC 值有单独的加法单元，同时采用哈佛结构（分离的指令和数据存储器），因此 MiniSys 的设计不存在结构相关性。

- 关于数据转发策略的细节

MiniSys 的设计中有四种数据需要向前转发：EXE 段的 ALU 运算结果、MEM 段保存的 ALU 运算结果、MEM 段读出的数据、寄存器回写数据。

按照转发最新数据的原则，四种数据的优先级为 EXE 段的 ALU 结果 >MEM 段保存的 ALU 结果或 MEM 段读出的数据 > 寄存器回写数据。

当前指令需要读取数据时，根据控制信号判断是否需要转发。如果需要则从下面四个输入中选一个作为转发数据：EXE 段的 ALU 运算结果、MEM 段保存的 ALU 运算结果、MEM 段读出的数据、寄存器回写数据。如果不需要则直接读寄存器文件。

转发策略完全由组合逻辑实现，并采用了优先级结构以保证较短的电路延迟。

- PC 流水的设计细节

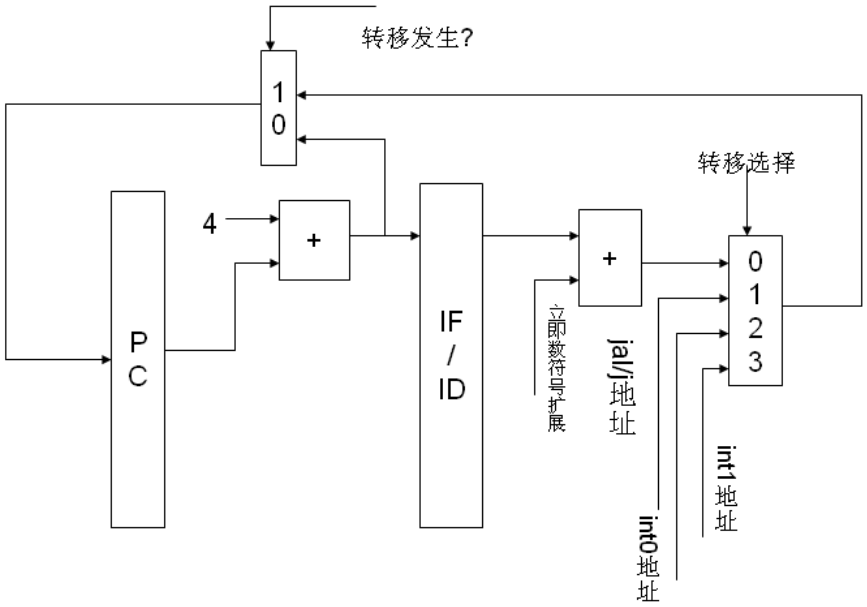

图 7 PC 的流水结构

图 7 就是 PC(指令计数器)流水的结构,从结构图我们看出，PC 流水实际上就是对新 PC 的选择过程，如果没有转移发生时，PC 顺序地产生，只有当转移发生时，PC 才会终止顺序产生，而装入新的 PC 值。多路开关的控制信号由译码单元和中断控制器联合产生。上述的 PC 流水通路在 PCGEN 模块中实现。图 8 是 PCGEN 的内部 RTL 结构。

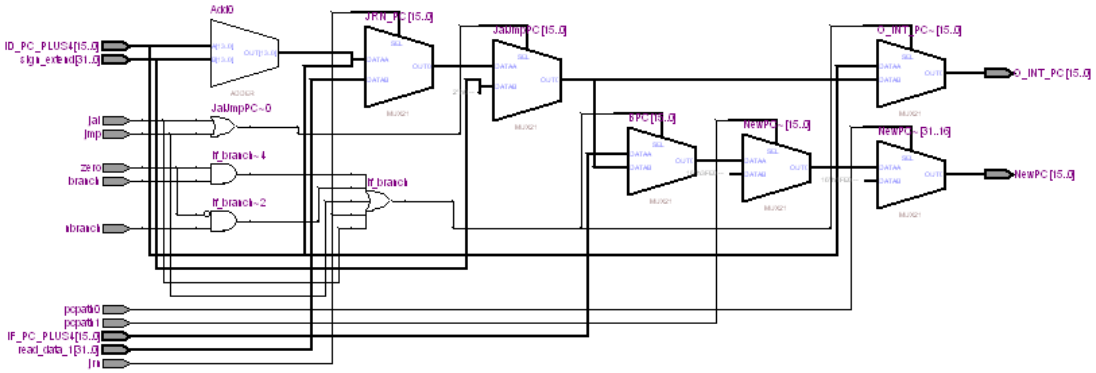

图 8　PCGEN 的内部 RTL 结构

II.汇编器

➢ 总体设计

汇编器主要由四个部分组成：词法分析、语法分析、机器码翻译、错误检测。其中语法分析是核心。具体结构如图 9 所示。

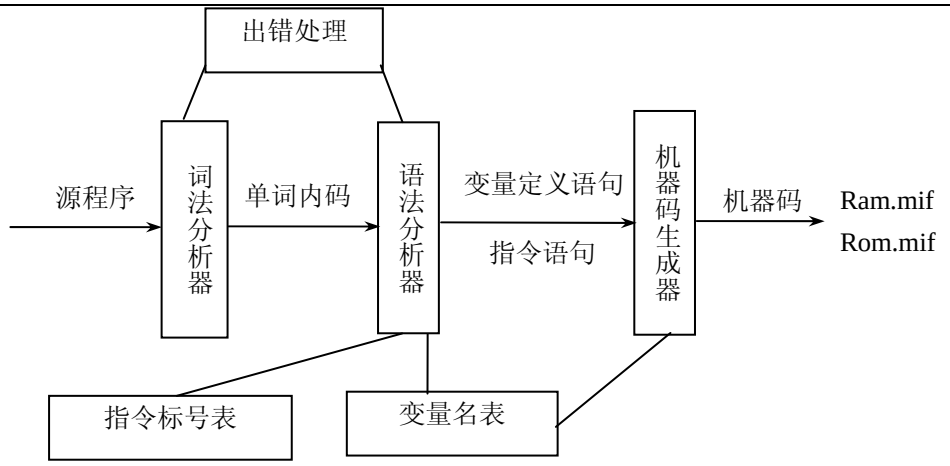

图 9 汇编器结构

词法分析器负责按照 MiniSys 汇编语言词法，组字符成词；语法分析器依据 MiniSys 汇编语言语法，组词成句；机器码生成器负责依据翻译规则（包括指令翻译和数据翻译），将一个完整的指令句子或变量定义语句转换为相应的机器指令。

词法分析器没有用自动机，而是采用了以下策略。我们用的是 LL（1）文法，所以每次读入一个单词前，基本已经知道了单词的类型。词法分析时，直接读入一个字符串，然后进行字符串匹配。比如通过语法分析，知道接下来是“ORG_CODE”。词法分析时只要把读入的字符串同“ORG_CODE”进行匹配，如果一样证明词法分析正确，否则就是出错。出错后，调用相应的错误处理程序，处理完后继续进行语法分析。

机器码的生成伴随着词法分析和语法分析。如果前一个单词识别正确，则进行翻译。否则跳过不进行翻译。

汇编器的关键是语法规则的定义，语法分析和词法分析都是严格按照语法规则进行的。

➢ 功能介绍

- 支持 MIPS 指令集的一个子集,含 31 条指令
- 采用软件模拟，扩展了 6 条指令：push、pop、jle、jl、jg、jge
- 数据类型支持 2、10、16 进制，以后缀区分
- 具有全局查错功能，即一次汇编尽可能多的找出错误
- 数据段定义支持多变量、多行定义、地址重定位
- 允许多行注释

➢ 模块实现

- 词法分析模块

该模块含一个扫描器的总控程序，总控程序根据需要调用不同的扫描子程序。比如语法分析过程中，需要词法分析程序识别一个标识符。它将标识符的内码传递给扫描器的总控程序 Scanner（T_IDNAME），然后再调用 ScanIdname()识别一个标识符。如果成功返回 0，失败则返回－1。一些简单的识别，比如识“DATA”或”CODE”等，直接在总控程序中实现，并没有单独定义函数。下面是词法分析程序关键函数声明：

```
int Scanner(int TypeOfToken);          //扫描器总控程序
```

```
int ScanCom();                          //扫描指令助记符
int ScanIdname();                       //扫描标识符
int Scan16Radix(int maxlen);            //扫描 16 进制数
int Scan10Radix(int maxlen);            //扫描 10 进制数
int Scan2Radix(int maxlen);             //扫描 2 进制数
int ScanReg(int ToeknID);               //扫描寄存器
int ScanEndl();                         //扫描注释
```

- 语法分析模块

语法分析采用了 LL（1）文法。下面是语法分析程序关键函数声明：

```
int Parser();                           //语法分析器
int Deduction();                        //语法推导器
void Succeed();                         //汇编成功
void Fail();                            //汇编失败
```

语法分析的具体算法如图 10 所示。

```
int assembler::Parser()     //词法分析总控程序
{
    虑除文件开头的注释
    Scanner(N_PRO);    //扫描第一个单词，如果出错汇编结束
    While(1)           //语法分析器核心代码
    {
        if(下推栈栈顶为非终结符号)
            进行推导
        else if(下推栈栈顶为终结符号)
            进行匹配,并进行翻译
        else if(已到文件末尾)
            判断汇编是否成功，即查看错误个数是否为0
            如果为0，输出数据文件和指令文件
        else
            出错，汇编结束
    }
}
```

图 10 语法分析算法

- 机器码生成模块

该模块主要是将汇编程序中的数据和指令翻译程机器码，其中数据放在 Ram.mif 文件，指令放在 Rom.mif 文件。翻译采取的策略是每识别一个正确的单词，调用翻译程序进行翻译。比如翻译这条指令：add $1,$2,$3 。词法程序正确识别指令助记符号“add”，接着调用翻译器，在指令链表中添加一条新指令，填充功能号和操作码。词法程序识别“$1”、“$2”、“$3”后，翻译器往先前添加的指令中填充 Rd、Rs、Rt 字段。这样就完成了一条指令的翻译。下面是机器码生成模块的关键函数：

```
int  Translater();                  //翻译器核心程序
int  TranslateImmi();               //翻译立即数
```

```
    int  TranslateAddr();           //翻译地址
    int  TranslateID();             //翻译变量
    //下面都是具体指令翻译函数
    int  TranslateRcom();
    int  TranslateSrcom();
    int  TranslateSllRcom();
    int  TranslateIcom();
    int  TranslateSicom();
    int  TranslateLwicom();
    int  TranslateJcom();
    int  TranslateBcom();
    int  TranslatePcom();
    int  TranslateJbcom();
    int  TranslateNop();
```

- 错误处理模块

可处理常见的词法、语法、语义错误。词法和语法错误是在词法分析中发现的，而语义错误是在翻译过程中发现的。遇到错误后，调用错误总控程序 Error(int TypeOfError)，错误总控程序根据错误类型调用不同的处理函数。处理函数打印错误信息，同时调整下推栈和输入流，这样语法分析程序就可以继续进行下去。下面是错误处理模块的关键函数：

```
    int Error(int TypeOfError);       //错误处理总控程序
    int ErrorLexical();               //词法语法错误处理
    int ErrorSemantic();              //语义错误处理
    int ReadErrorToken();             //读取错误单词
    int ReadErrorSentence();          //读取错误句子
```

- **语法规则**

1. <N_PRO>→<N_DATA><N_CODE>
2. <N_DATA>→<N_DATA SEG> <N_DATA>′<N_DATA ENDS>
3. <N_DATA>→ε
4. <N_DATA SEG>→{T_DATA}{T_SEG}{T_ENDL}
5. <N_DATA>′→<N_ORG_DATA><N_VARS><N_DATA>′
6. <N_DATA>′→<N_VARS><N_DATA>′
7. <N_DATA>′→ε
8. <N_ORG_DATA>→{T_ORG_DATA}{T_ADDR}{T_ENDL}
9. <N_VARS>→<N_VAR><N_VARS>
10. <N_VARS>→ε
11. <N_VAR>→{T_IDNAME}{T_DW}{T_NUM}<N_VAR>′{T_ENDL}<N_VAR>″
12. <N_VAR>′→{T_COMMA}{T_NUM}<N_VAR>′
13. <N_VAR>′→ε
14. <N_VAR>″ →{T_DW}{T_NUM}<N_VAR>′{T_ENDL}<N_VAR>″
15. <N_VAR>″ →ε
16. <N_DATA ENDS>→{T_DATA}{T_ENDS}{T_ENDL}

17. <N_CODE>→<N_CODE SEG><N_CODE>′<N_CODE ENDS>
18. <N_CODE SEG>→{T_CODE}{T_SEG}{T_ENDL}
19. <N_CODE>′→<N_ORG_CODE><N_SEG>
20. <N_CODE>′→<N_SEG>
21. <N_ORG_CODE>→{T_ORG_CODE}{T_ADDR}{T_ENDL}
22. <N_SEG>→<N_START SEGID><N_ORDER><N_ORDERS><N_ ENDSEGID>
23. <N_START SEGID>→{T_IDNAME}{T_COLON}
24. <N_ORDER>→<N_COM>{T_ENDL}
25. <N_ORDER>→<N_ORG_CODE>
26. <N_COM>→{T_RCOM}{T_RD}{T_COMMA}{T_RS}{T_COMMA}{T_RT}
27. <N_COM>→{T_SRCOM}{T_RS}
28. <N_COM>→{T_SLLRCOM}{T_RD}{T_COMMA}{T_RT}{T_COMMA}{T_SHAMT}
29. <N_COM>→{T_ICOM}{T_RT}{T_COMMA}{T_RS}{T_COMMA}{T_IMMEDIATE}
30. <N_COM>→{T_SICOM}{T_RT}{T_COMMA}{T_IMMEDIATE}
31. <N_COM>→{T_LWICOM}{T_RT}{T_COMMA}<N_ IMMEDIATE >{T_BRS}
32. <N_COM>→{T_JCOM}<N_ADDR>
33.<N_COM>→{T_BCOM}{T_RT}{T_COMMA}{T_RS}{T_COMMA}<N_ IMMEDIATE
34.<N_COM>→{T_PCOM}{T_RS}
35. <N_COM>→{T_JBCOM}{T_RT}{T_COMMA}{T_RS}{T_COMMA}<N_ IMMEDIATE >
36. <N_COM>→{T_NOP}
37. <N_ IMMEDIATE >→{T_IDNAME}
38. <N_ IMMEDIATE >→{T_ IMMEDIATE }
39. <N_ADDR>→{T_IDNAME}
40. <N_ADDR>→{T_ADDR}
41. <N_ORDERS>→<N_ORDER><N_ORDERS>
42. <N_ORDERS>→<N_SUBSEGID><N_ORDER><N_ORDERS>
43. <N_ORDERS>→ε
44. <N_SUBSEGID>→{T_IDNAME}{T_COLON}
45. <N_ENDSEGID>→{T_END}{T_IDNAME}{T_ENDL}
46. <N_CODE ENDS>→{T_CODE}{T_ENDS}{T_ENDL}

- **错误检测**

集成开发环境具有强大的错误检测功能，支持全局查错。主要是利用词法规则，当遇到词法和语义错误时，假定识别成功，然后对下推栈和输入流进行相应调整。当遇到以下几种情况时，汇编结束。（1）第一单词不是 DATA 或 CODE、（2）程序未完成、（3）指令地址有冲突、（4）数据段定义有问题。图 11 是四种情况的示例程序。

```
DATE   SEG
  Buf DW 00001234H
DATA    ENDS
CODE SEG
Start:
    lw   $1, 0000h($0)
END Start
CODE ENDS
错误类型：第一单词不是DATA或CODE
错误提示：line 1 : syntax error : program doesn't begin with 'data' or 'code'
```

```
DATA   SEG
    Buf DW 00001234H
DATA   ENDS
CODE   SEG
Start:
    lw   $1, 0000h($0)

错误类型：程序未完成
错误提示：line 6 : syntax error : program has not been completed
```

```
DATA   SEG
  Buf DW 00001234H
DATA    ENDS
CODE SEG
Start:
    lw   $1, 0000h($0)
ORG_CODE 0000H
    or $2,$0,$1
END Start
CODE ENDS
错误类型：指令地址有冲突
错误提示：line 8 : 'or' : semantic error : address conflict
```

```
DATA   SEG
ORG_CODE 0004H
  Buf DW 00001234H
DATA    ENDS
CODE SEG
Start:
    lw   $1, 0000h($0)
END Start
CODE ENDS
错误类型：数据段定义有问题
错误提示：line 2 : syntax error : wrong data seg
```

图 11 汇编中止的四种情况

III. 编译器

- **总体设计**

MiniC 编译器主要由以下几个部分组成,词法分析器，语法制导分析筐架（含中间代码生成）,寄存器分配模块，函数块翻译模块，代码综合输出模块。结构示意如图 12 所示。

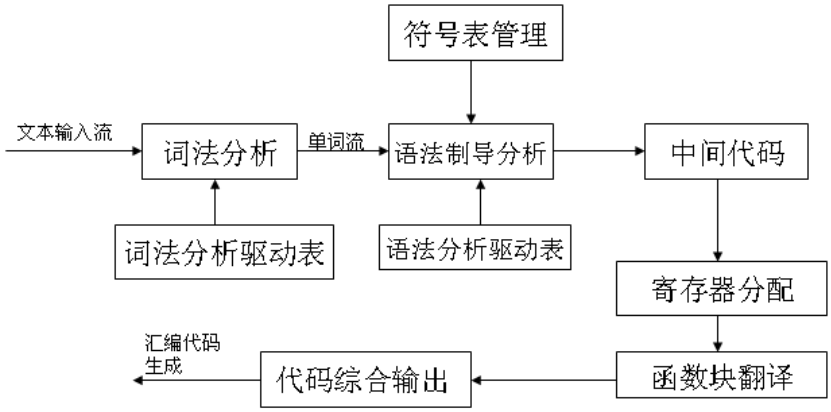

图 12 编译器结构

词法分析器在词法分析表的驱动下，从输入流中识别符号，为语法分析提供单词流。语法分析器采用了 LALR(1)分析法，并采用语法指导的方式进行中间代码的生成，符号表管理模块为语法分析过程提供单词登记，查询的功能，并参与变量的空间分配计算，为后续的代码生成,提供地址信息。寄存器分配部分负责对翻译过程中产生的临时变量进行寄存器关联，采用的策略是简单统计各个临时变量的使用频率，为使用最频繁的前 10 个临时变量分配寄存器，对超出部分的寄存器，为其分配内存空间。函数块的翻译负责将中间代码生成部分产生的各个函数的中间代码翻译为汇编代码，并加上相应的现场保护，现场恢复的代码，完成子程序的翻译。最后的代码综合输出主要作用是，将各个子程序进行总装，加上中断入口，程序初始化代码，产生完整的一个汇编代码，完成翻译。产生的汇编代码只要通过 Minisys 的汇编器翻译，就可以生成最终的机器代码，进入 CPU 运行。

- **词法规则(Lex描述)**

```
Digit            ([0-9])|([1-9][0-9]*)
letter           [a-zA-Z]
id               {letter}({letter}|{digit})*
num              {digit}[1-9]*
hex              0(x|X)({letter}|[0-9])*
%%
"void"           {return VOID;}
"continue"       {return CONTINUE;}
"if"             {return IF; }
"while"          {return WHILE;}
"else"           {return ELSE;}
"break"          {return BREAK;}
"int"            {return INT;}
"return"         {return RETURN;}
"\|\|"           {return OR;}
"&&"             {return AND;}
{id}             {yyval.IDENT_v.ID_NAME=yytext;return IDENT;}
{hex}            {yyval.int_literal_v.int_val=ConvertHexToint(yytext);return
                 HEXNUM;}
```

```
{num}            {yyval.int_literal_v.int_val=atoi(yytext);return DECNUM;}
"\<="            {return LE;}
"\>="            {return GE;}
"\=\="           {return EQ;}
"\!\="           {return NE;}
"\>"             {return '>';}
"\<"             {return '<';}
"\,"             {return ','; }
"\;"             {return ';'; }
"\{"             {return '{'; }
"\}"             {return '}'; }
"\%"             {return '%'; }
"\*"             {return  '*';}
"\+"             {return '+'; }
"\-"             {return '-'; }
"/"              {return '/'; }
"\="             {return '='; }
"\("             {return '('; }
"\)"             {return ')'; }
"~"              {return '~';}
"&"              {return '&';}
"^"              {return '^';}
"\["             {return '[';}
"\]"             {return ']';}
"\<\<"           {return LSHIFT;}
"\>\>"           {return RSHIFT;}
"\|"             {return '|';}
\t|\             { }
\n|\r\n          {Lineno++;yylineno++;}
"$"              {return '$';}
%%
```

- **语法规则(Yacc描述)**

```
%token IDENT VOID INT WHILE IF ELSE RETURN EQ NE LE GE AND OR DECNUM
CONTINUE BREAK HEXNUM LSHIFT RSHIFT
%left OR
%left AND
%left EQ NE LE GE '<' '>'   /*关系运算*/
%left '+' '-'
%left '|'
%left '&' '^'
%left '*' '/' '%'   /*算术运算*/
%right LSHIFT RSHIFT
```

```
%right '!'
%right '~'
%nonassoc UMINUS
%nonassoc MPR
%start program
%%
program   : decl_list   ; /*程序由变量描述或函数描述组成（decl）*/
decl_list   : decl_list decl | decl ;
decl   : var_decl | fun_decl ;
var_decl   : type_spec IDENT ';' | type_spec IDENT '[' int_literal ']' ';' ; /*变量包括简单变量和一维数组变量*/
type_spec   : VOID | INT ; /*函数返回值类型或变量类型包括整型或 VOID*/
fun_decl    : type_spec FUNCTION_IDENT '(' params ')' compound_stmt
        | type_spec FUNCTION_IDENT '(' params ')' ';' ; //要考虑设置全局函数信息为假，和函数表为申明
FUNCTION_IDENT : IDENT {cout<<"function ident"<<endl;} ; /*建立全局函数名变量*/
params    :   param_list | VOID ; /*函数参数个数可为 0 或多个*/
param_list   : param_list ',' param | param ;
param      :   type_spec IDENT | type_spec IDENT '[' int_literal ']' ;
stmt_list    : stmt_list stmt | ;
stmt      :   expr_stmt | block_stmt | if_stmt | while_stmt |return_stmt | continue_stmt | break_stmt   ;
expr_stmt   : IDENT '=' expr ';' | IDENT '[' expr ']' '=' expr ';' | '$' expr '=' expr ';' | IDENT '(' args ')' ';' ; /*赋值语句*/
while_stmt : WHILE_IDENT '(' expr ')' stmt ; /*WHILE 语句*/
WHILE_IDENT : WHILE {cout<<"while ident"<<endl;} ;/*建立入口出口信息全局变量*/
block_stmt : '{' stmt_list '}'   ; /*语句块*/
compound_stmt : '{' local_decls stmt_list '}'   ; /*函数内部描述，包括局部变量和语句描述*/
local_decls   : local_decls local_decl | ; /*函数内部变量描述*/
local_decl    : type_spec IDENT ';' | type_spec IDENT '[' int_literal ']' ';' ;
if_stmt   :   IF '(' expr ')' stmt %prec UMINUS | IF '(' expr ')'   stmt ELSE stmt %prec MPR ;
return_stmt   : RETURN ';' | RETURN expr ';' ;
expr   :   expr OR expr   /*逻辑或表达式，运算符为'||'*/
  | expr EQ expr | expr NE expr /*关系表达式*/
  | expr LE expr | expr '<' expr | expr GE expr | expr '>' expr /*关系表达式*/
  | expr AND expr /*逻辑与表达式，运算符为’&&’**/
  | expr '+' expr | expr '-' expr /*算术表达式*/
  | expr '*' expr | expr '/' expr | expr '%' expr   /*算术表达式*/
  | '!' expr %prec UMINUS | '-' expr %prec UMINUS | '+' expr %prec UMINUS | '$' expr %prec UMINUS   /*$ expr 为取端口地址为 expr 值的端口值*/
  | '(' expr ')'
  | IDENT | IDENT '[' expr ']' | IDENT '(' args ')'   /* IDENT ( args )为函数调用*/
  | int_literal   /*数值常量*/
  | expr '&' expr /*按位与*/
```

```
   | expr '^' expr /*按位异或*/
   | '~' expr    /*按位取反*/
   | expr LSHIFT expr /*逻辑左移*/
   | expr RSHIFT expr /*逻辑右移*/
   | expr '|' expr /*按位或*/
    ;
int_literal   :   DECNUM
                  |   HEXNUM ; /*数值常量是十进制整数*/
arg_list     :   arg_list ',' expr | expr ;
args     :     arg_list | ;
continue_stmt : CONTINUE ';' ;
break_stmt : BREAK ';' ;
%%
```

注:本程序的词法和语法分析器由我们自己设计的 SeuLex+SeuYacc 工具生成，详细细节请参考 SeuLex 和 SeuYacc 的设计文档。

- **运行时的内存布局**

Mini C 编译器产生的程序，运行时的活动记录基本类似于 ANSI C 的结构，考虑到 Mini C 对 C 语言进行了裁剪，对活动记录进行了适应性修改，活动记录采用了向上生长的方式，具体结构如图 9 所示。

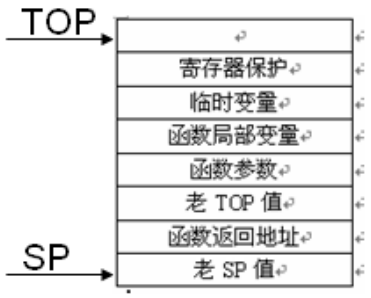

图 13 活动记录结构

SP 是指向当前活动记录底部的指针，TOP 是指向当前活动记录顶部的第一个空单元的指针。

当函数调用发生时，产生一个新的活动记录，当一个函数返回时，它的活动记录也被释放，从而实现了一个栈式动态分配。

- 活动记录的创建。当一个函数调用发生时，向 8(TOP),0(TOP)保存老 SP 和老 TOP,保护现场,初始化新的 SP 为 SP=TOP,新 TOP 初始化为老 TOP+新活动记录的大小。
- 活动记录的销毁和释放。当一个函数返回时,先完成现场的恢复，恢复 TOP=8(SP),恢复 SP=0(SP)。

图 10 是一般程序的内存布局示意图。

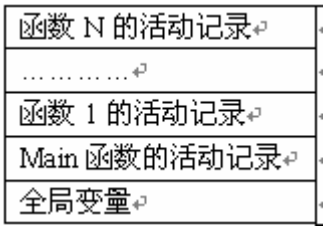

图 13 一般程序的内存布局

- **中间代码到汇编代码翻译的技巧**

中间代码在完成临时变量的寄存器分配后，一共有三种变量形式，常数，t 类寄存器(10 个),内存数，为了简化翻译过程,以四个 s 寄存器为基础，加入 MOV 中间指令，负责在三种变量类型之间进行转换,从而省略了其他中间指令翻译的代码量,实现了模块的最大化重用。MOV 中间指令的思想很简单,借鉴了 X86 指令集的 MOV,同时考虑到 MIPS 指令的规整性,以及 Mini C 的特点，将数据转换传送过程交给 MOV 处理，解决了代码重复问题。

- **错误检测**

提供一定的语法错误检测能力和丰富的语义错误检测功能。对于非法的符号，词法分析器有一定的能力跳过，从而使分析过程能够继续下去。语义检测功能中，提供了很好的类型匹配和参数匹配检测，像 ANSI C 一样，不提供对数组下标的检查。下面是一个简单的例子:

```
int fib(int n);
void main(void)
{ fib(1,2); }
```

编译器会检测出 fib 函数的参数数目不匹配，并给出提示。具体的细节，参阅使用说明。

IV.编辑器

- **总体设计**

编辑器采用了 MFC 典型的文档视图结构。其中文档负责数据的读取和存储，视图负责数据的显示以及和用户的交互。编辑器主要通过四个过程实现：新建和打开、文本编辑、汇编和编译、调试。编辑器的结构图如图 14 所示。

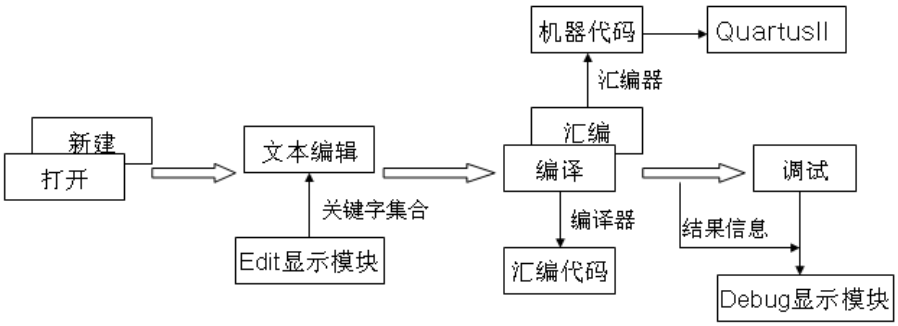

图 14　编辑器结构图

➢ **功能介绍**

- 支持一般编辑器的功能
- 支持 Mini C 和汇编两种类型的工程文件
- 支持 Mini C 和汇编两种类型的关键字高亮显示
- 具有简单的调试和错误显示功能
- 与 QuartusII 软件无缝集成

➢ **主要模块实现**

- 新建和打开

用户新建工程后，会在用户指定的路径下建立工程文件夹，同时根据用户选择的工程类型新建文档，接着调用显示视图模块进行必要的初始化。

- 文本编辑

用户输入程序后，MFC 的键盘消息就会触发，并调用显示视图模块，将关键字高亮显示出来，以此循环，用户每输入一个字符，显示模块就会检查缓存中的字符串是否构成关键字，以决定最终的显示效果。

- 汇编和编译

用户输入结束后，根据工程类型，可以调用相应的“汇编”或“编译”的按钮命令。按钮命令调用后，首先保存文档，然后将该文档送入后续汇编器或编译器进行汇编或编译处理。

- 调试

汇编和编译后的结果信息，通过一个临时文件保存，并由编辑器的 Debug 显示模块显示。对于 Mini C 程序，编译无误后会在工程文件夹下建立汇编文件（一个以原文件为文件名，以.asm 为后缀的汇编文件）；对于汇编程序，汇编无误后则会在工程文件夹下建立机器代码文件(Rom、Ram 和指令统计文件)，并同时在 MiniSysSoC 目录下建立同样的 Rom.mif、Ram.mif 文件，以便和 MiniSys 的硬件一起使用 QuartusII 进行联合编译。若用户主机上安装有 QuartusII，则相应的“Quartus”按钮就会亮起，用户可以以此调用 QuartusII 进行联合编译，并查看机器代码相应的波形。

本组设计的 MiniSys 汇编程序使用手册

详细信息请参阅.chm 帮助文档.

本组设计中的 Verilog HDL 关键程序清单及 GDT 图（如果有）
请参阅工程包.

本组设计主要测试结果与性能分析（.vwf、.rpt 中的资源使用情况）

I.资源使用

Fitter Status	Successful - Mon Jan 05 13:13:36 2009
Quartus II Version	7.2 Build 151 09/26/2007 SJ Full Version
Revision Name	MIPS
Top-level Entity Name	minisys
Family	Cyclone
Device	EP1C6Q240C8
Timing Models	Final
Total logic elements	4,018 / 5,980 (67 %)
Total pins	50 / 185 (27 %)
Total virtual pins	0
Total memory bits	65,536 / 92,160 (71 %)
Total PLLs	0 / 2 (0 %)

II.性能

	Value
From	RegFile:RF|GPR:OurGPR|Dffe32:reg9|De4
To	RegFile:RF|GPR:OurGPR|Dffe32:reg26|De14
Clock period	35.538 ns
Frequency	28.14 MHz

III.功耗

PowerPlay Power Analyzer Status	Successful - Mon Jan 05 13:14:58 2009
Quartus II Version	7.2 Build 151 09/26/2007 SJ Full Version
Revision Name	MIPS
Top-level Entity Name	minisys
Family	Cyclone
Device	EP1C6Q240C8
Power Models	Final
Total Thermal Power Dissipation	60.11 mW
Core Dynamic Thermal Power Dissipation	0.00 mW
Core Static Thermal Power Dissipation	60.07 mW
I/O Thermal Power Dissipation	0.04 mW
Power Estimation Confidence	Low: user provided insufficient toggle rate data

课程设计总结（包括设计的总结和还需改进的内容）

本课程设计，我们组在全体成员的集体努力下，圆满完成了任务。本小组本着求是进取，努力创新的目标，以最高的标准要求自己，完成了流水线 CPU、汇编器、编译器、集成开发环境、外设的开发。并完成了完善的下载验证和综合测试，使得我们小组的系统具备的极强的稳定性和实用性。总的来说，我们小组出色完成了任务。

下面谈谈需要改进的地方:

1.CPU 部分，由于设计时间仓促，CPU 本身有较大的延迟，使得主频只有 28MHz,能进行的改进就是优化线路和逻辑结构，提高主频。同时加上指令预取，分支预测，动态执行来提高效能.

2.编译器部分，输出代码由于未进行优化，导致代码体积偏大。改进的内容就是加入代码优化，同时引入新的寄存器分配算法，减少临时变量，提高性能。

3.汇编器部分，错误检测功能有待进一步加强。目前汇编器遇到四种情况就结束汇编，可以做一些优化使得汇编器自动纠正这些错误。

4.编辑器部分，不支持多文档，不能在每行开头显示行号。

5.外设部分，主要是按照书上的范例设计的，影响了系统的整体性能。

验 收 报 告

主频	MHz	逻辑单元数	(%)	功 耗	mW
CPU 类型		☐单周期 ☐多周期 ☐流水线 ☐超标量			
CPU 设计	☐通过 ☐未通过		数码管控制	☐通过 ☐未通过	
定时/计数器	☐通过 ☐未通过		键盘控制器	☐通过 ☐未通过	
PWM 控制	☐通过 ☐未通过		UART 控制	☐通过 ☐未通过	
看门狗控制	☐通过 ☐未通过		合 成	☐通过 ☐未通过	
汇编器设计	☐通过 ☐未通过		BIOS	☐有 ☐无 ☐未通过	
中断	☐有 ☐无 ☐未通过		验收答辩	☐通过 ☐未通过	
加分项目					
验收结论	☐优秀 ☐良好 ☐中等 ☐及格 ☐不及格				

教师综合评价:

教师签名: _____