华中科技大学计算机科学与技术学院

C 语言课程设计报告

题目：C 语言课程设计报告（提高部分）

专　　业：计算机科学与技术

班　　级：计卓 1501

学　　号：U201514898

姓　　名：胡思勖

成　　绩：

指导教师：李开

完成日期：　2015 年　9　月　25　日

目 录

C 输入输出库函数的程序设计

一、系统需求分析

一、 总体要求:

1. 给定 getchar 和 putchar 函数，实现其它 C 输入输出库函数。如：gets，puts，printf，scanf 等。并且在原函数名前加 my 构成新函数名。如：mygets，myputs，myprintf，myscanf 等。
2. 创建 mylibrary.lib 库，将自己实现的库函数加入到该库中。
3. 对自行设计的每个库函数，编写实验程序，调用 mylibrary.lib 库自行设计的库函数，要求得到正确结果。

二、 输入数据

1. 将不同类型的数据用 myprintf 读入。
2. 将字符串类型的数据用 mygets 读入。

三、 输出数据

1. 用 gdb 调试观察 myprintf 和 myscanf 读入的变量。
2. 用 myprintf 和 myputs 观察读入的变量以测试函数功能的完整性。

四、 功能要求

1. myscanf 要求支持%c，%d，%s，%f，%Lf，并支持域宽和精度控制。
2. myprintf 要求支持%c，%d，%s，%f，%lf，并支持域宽控制和精度控制。

3. mygets 和 myputs 与 gets 和 puts 实现相同的功能即可。
4. 将编译好的函数打包为.a 库文件，并在另一个文件中进行测试。

五、 性能要求

1. 由于 scanf 和 printf 的健壮性并不是非常强，主要又程序员来控制程序的健壮性，因此本例实现的 myscanf 和 myprintf 只要求对所有能够支持的数据类型和精度类型保证处理结果的正确即可。mygets 和 myputs 要求读入和输出的数据正确。
2. .a 文件能够正确的被连接到测试文件上一起编译。

二、总体设计

一、 myprintf 和 myscanf 的设计

对于 myprintf 和 myscanf 来说，需要实现的就是根据当前状态和当前读入的字符来决定下一个状态的有限状态机。myprintf 和 myscanf 的区别仅在于少许状态的不同（如%Lf 和%lf），和输入输出的形式的不同：myprintf 的输入是字符串额而输出是 stdout，myscanf 的输入是 mystdin，输出将读入量存入各个变量中。由于变量的个数不确定，需要使用 stdarg.h 中的不定传参进行处理。

因此，使用 if 开关、switch…case 以及其嵌套即可对每一种情况进行判断，从而决定读入和输出的状态。具体的逻辑设计见详细设计部分。

二、 mygets 和 myputs 的设计

mygets 和 myputs 相对于 myscanf 和 myprintf 而言较为简单，mygets 从标准输入流中读入一个字符串（以空白字符作为结束），而 myputs 仅仅把一个字符串输出到屏幕上即可。

因此，mygets 函数只需判断函数的输入是够为空白字符，myputs 只需判断函数的输入是否到达字符串的结尾即可。

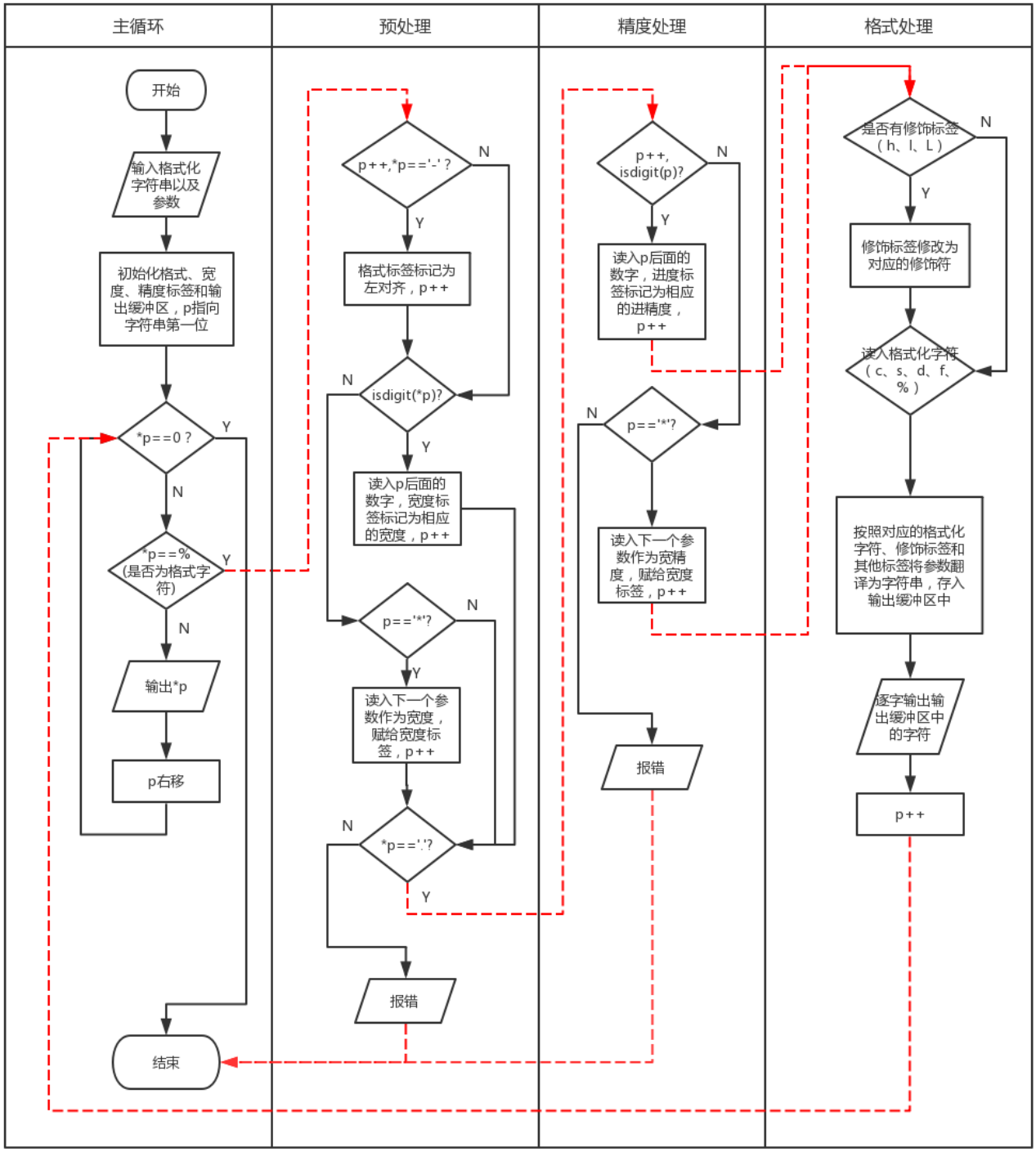
三、数据结构设计

一、 对于 myprintf 和 myscanf 而言，不采用特殊的数据结构，仅用一个字符数组来存放待输出的特殊变量，用一个指针来指向当前格式化字符串中的字符，以及用一系列的 int 变量来表示状态机的当前状态。

二、 对于 mygets 和 myputs 而言也不需要使用特殊数据结构，简单地从字符串读入数据或输入数据到字符串即可。

四、详细设计

一、 myprintf：输入格式化字符串和一系列的参数，通过分析格式化字符串中的格式字符%与对应的参数，将参数转化为字符串输出，其余部分作为普通字符串输出即可，但格式处理的部分较为复杂，具体流程图如下：



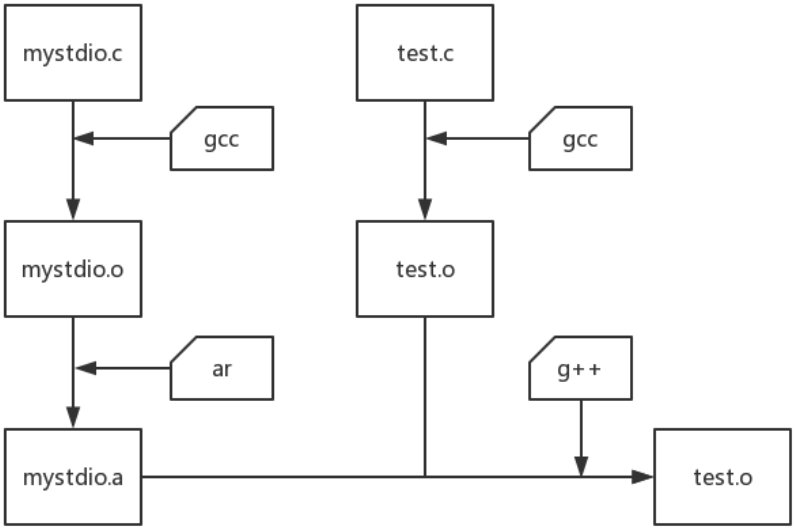
二、 myscanf：与 myprinf 处理的方式基本相同，不同的是讲格式处理中的“按照对应的格式化字符、修饰标签和其他标签将参数翻译为字符串，存入输出缓冲区中”改为“按照对应的格式化字符、修饰标签和其他标签将输入流中的字符（以空白字符分隔）翻译为对应的常量，存入参数所给的地址中”，然后将“逐字输出输出缓冲区的字符”删去即可。

三、 mygets：mygets 的具体实现就是不断地从标准输入流中读取数据，然后存入传参进来的字符串指针中，直到遇到空白字符（空格，换行，回车，回车换行等）。

四、 myputs：myputs 的具体实现就是简单地一个一个输出传入字符串的所有字符，直到遇到结束符’\0’为止。

五、系统实现

系统实现分为两个部分，第一部分：先将 mystdio.c 编译为 mystdio.o 用 ar 将 mystdio.o 打包为静态链接库 mystdio.a。第二部分：先将测试文件 test.c 编译为 test.o，然后再用 g++将 test.o 和 mystdio.a 链接为二进制文件 test，原理图如下(省去头文件，其中仅为 mystdio.c 中的函数原型)：



其中 mystdio.c 到 mystdio.o 的编译命令为：

```
gcc -c mystdio.c -o mystdio.o
```

mystdio.o 到 mystdio.a 的打包命令为：

```
ar rc mystdio.a mystdio.o
```

test.c 到 test.o 的编译命令为：

```
gcc -Wall -std=c99 -c test.c -o test.o
```

g++的链接命令为：

```
g++ -o test test.o mystdio.a
```

各个程序的源码如下：

mystdio.c

```
#include "mylibrary.h"
#include <stdio.h>
#include <stdarg.h>
#include <ctype.h>
#include <float.h>
#include <math.h>

#define LEFT        0x01
#define SHORT       0x02
#define LONG        0x04
#define LONG_DOUBLE 0x08

/* my implementation of itoa */
void myitoa(int n, char s[]);
/* my implementation of dtoa */
void mydtoa(double n, char s[]);

int myprintf(const char *string, ...){
    const char *pointer =   string;
    /* record the current format-string format */
    int format;
    /* record the precision of the format-string */
    int width;
    /* record the precision of the character */
    int precision;
    /* the string buffer to store the int and float
argumentation */
    char buffer[50];
```

```
/* the pointer point to string argumentation */
char *str;
/* the list of argumentation */
va_list args;
/* count the argumentation number */
int counter =   0;
/* initialize the argumentation list */
va_start(args, string);
for(; *pointer!='\0'; pointer++){
    format      =   0;
    width       =   -1;
    precision   =   -1;
    /** if is not the format-string */
    if(*pointer !=  '%'){
        putchar(*pointer);
        continue;
    }
    /** if is the format-string **/
    pointer++,counter++;
    /* get the margin */
    if(*pointer=='-')
        format  |=  LEFT,   pointer++;

    /* get width */
    if(isdigit(*pointer)){
        width   =   0;
        for(;isdigit(*pointer);)
            width   =   width*10 + *(pointer++) - '0';
    }
    /* if width is in the arg list*/
    else if(*pointer=='*'){
```

```
            width    =    va_arg(args,int);
            if(width<0){
                width    =    -width;
                format   |=   LEFT;
            }
            pointer++;
        }
        /* get the precision */
        if(*pointer == '.'){
            pointer++;
            if(isdigit(*pointer)){
                precision    =    0;
                for(;isdigit(*pointer);)
                    precision    =    precision*10 +
*(pointer++) - '0';
            }
            /* if precision is in the arg list */
            else if(*pointer=='*'){
                precision    =    va_arg(args,int);
                if(precision<0)
                    precision    =    0;
                pointer++;
            }
        }
        /* get the modifier */
        switch(*pointer){
            case 'h':
                format   |=   SHORT;
                pointer++;
                break;
            case 'l':
```

```
                format  |=  LONG;
                pointer++;
                break;
            case 'L':
                format  |=  LONG_DOUBLE;
                pointer++;
                break;
        }

        /* get the format character and puts to the stdout
*/
        switch(*pointer){
            /* char */
            case 'c':
                /* if left aligned */
                if(format&LEFT){
                    int i=0;
                    putchar(va_arg(args,int));
                    for(;i<width-1;i++)
                        putchar(' ');
                }
                /* else if right aligned */
                else{
                    int i=0;
                    for(;i<width-1;i++)
                        putchar(' ');
                    putchar(va_arg(args,int));
                }

                break;
            /* int */
```

```
            case 'd':
                myitoa(va_arg(args,int), buffer);
                /* if left aligned */
                if(format&LEFT){
                    int i=0;
                    for(i=0; buffer[i]!='\0'; i++)
                        putchar(buffer[i]);
                    for(; i<width; i++);
                        putchar(' ');
                }
                /* right aligned */
                else{
                    int i=0;
                    /* in order not to use strlen */
                    for(; buffer[i]!='\0'; i++);
                    for(; i<width; i++)
                        putchar(' ');
                    for(i=0; buffer[i]!='\0'; i++)
                        putchar(buffer[i]);
                }
                break;
            /* string */
            case 's':
                str =   va_arg(args,char *);
                if(format&LEFT){
                    int i=0;
                    for(i=0; (precision<0 || i<precision)
&& str[i]!='\0'; i++)
                        putchar(str[i]);
                    for(; i<width; i++);
                        putchar(' ');
```

```
                }
                /* right aligned */
                else{
                    int i=0;
                    /* in order not to use strlen */
                    for(; str[i]!='\0'; i++);
                    for(; i<width; i++)
                        putchar(' ');
                    for(i=0; (precision<0 || i<precision)
&& str[i]!='\0'; i++)
                        putchar(str[i]);
                }
                break;
            /* float */
            case 'f':
                mydtoa(va_arg(args,double),buffer);
                if(precision<0)
                    precision   =   6;
                /* if left aligned */
                if(format&LEFT){
                    int i=0,j=0;
                    for(; buffer[i]!='\0' &&
buffer[i]!='.' ; i++)
                        putchar(buffer[i]);
                    putchar('.');
                    if(buffer[i]!='\0')
                        i++;
                    for(; j<precision;j++,i++){
                        /* if reach the end, print the
remaining '0' */
                        if(buffer[i]=='\0'){
```

```
                        for(;j<precision;j++,i++)
                            putchar('0');
                        break;
                    }
                    /* if didn't reach the end */
                    else
                        putchar(buffer[i]);
                }
                for(; i<width;i++)
                    putchar(' ');
            }
            /* if right aligned */
            else{
                int i=0, j=0;
                for(;buffer[i]!='\0' &&
buffer[i]!='.'; i++);
                i+=precision;
                for(;i<width;i++)
                    putchar(' ');
                for(i=0; buffer[i]!='\0' &&
buffer[i]!='.'; i++)
                    putchar(buffer[i]);
                putchar('.');
                if(buffer[i]!='\0')
                    i++;
                for(;j<precision;j++,i++){
                    if(buffer[i]=='\0'){
                        for(;j<precision;j++,i++)
                            putchar('0');
                        break;
                    }
```

```
                        /* if didn't reach the end */
                        else
                            putchar(buffer[i]);
                    }
                }
                break;
            case '%':
                putchar('%');
                break;
            default:
                putchar('%');
                break;
        }
    }
    return counter;
}


int myscanf(const char *fmt, ...){
    va_list args;
    va_start(args, fmt);
    char temp;
    int counter=0;
    for(;*fmt != '\0';){
        if (*(fmt) == '%'){
            counter++;
            switch(*++fmt){
                /* char */
                case 'c':{
                    char *pointer    =    va_arg(args,char
*);
```

```
                    *pointer    =   getchar();
                }
                break;
                /* integer */
                case 'd':{
                    int *pointer    =   va_arg(args,int
*);
                    *pointer        =   0;
                    for(; isspace(temp=getchar()););
                    /* judge if the number is nagative*/
                    int is_negative =   0;
                    if(temp=='-')
                        is_negative =   1;
                    else if(isdigit(temp))
                        *pointer    +=  temp-'0';
                    for(; isdigit(temp=getchar()); )
                        *pointer    =
(*pointer)*10+temp-'0';
                    if(is_negative)
                    *pointer        =   -(*pointer);
                    ungetc(temp, stdin);
                }
                break;
                /* string */
                case 's':{
                    char *pointer = va_arg(args, char *);
                    for(; isspace(temp=getchar()););
                    for(*pointer=temp, pointer++;
isspace(temp=getchar())==0;){
                        *pointer    =   temp;
                        pointer++;
```

```
                    }
                    *pointer    =   '\0';
                    ungetc(temp, stdin);
                }
                break;
                /* double */
                case 'l':{
                    if(*++fmt!='f')
                        break;
                    double *pointer =   va_arg(args,
double *);
                    *pointer = 0;
                    for(; isspace(temp=getchar()); );
                    /* is negative number */
                    int is_negative =   0;;
                    if (temp=='-')
                        is_negative = 1;
                    else if (isdigit(temp))
                        *pointer += temp - '0';
                    for(; isdigit(temp=getchar());)
                        *pointer    =
(*pointer)*10+temp-'0';
                    /* get digits after the dot */
                    if (temp=='.'){
                        int count = 10;
                        for(; isdigit(temp=getchar()); )
                            *pointer += (double)(temp-
'0')/count, count*=10;
                    }
                    if(is_negative)
                        *pointer = -(*pointer);
```

```
                    ungetc(temp, stdin);
                }
                break;
                /* float ,deal the same as double*/
                case 'f':{
                    float *pointer  =   va_arg(args, float
*);
                    *pointer = 0;
                    for(; isspace(temp=getchar()); );
                    /* is negative number */
                    int is_negative =   0;;
                    if (temp=='-')
                        is_negative = 1;
                    else if (isdigit(temp))
                        *pointer += temp - '0';
                    for(; isdigit(temp=getchar());)
                        *pointer    =
(*pointer)*10+temp-'0';
                    /* get digits after the dot */
                    if (temp=='.'){
                        int count = 10;
                        for(; isdigit(temp=getchar()); )
                            *pointer += (float)(temp-
'0')/count, count*=10;
                    }
                    if(is_negative)
                        *pointer = -(*pointer);
                    ungetc(temp, stdin);
                }
                break;
            }/* switch */
```

```
        }/* if */
        /* if not '%' */
        else{
            for(;isspace(temp=getchar()););
            ungetc(temp,stdin);
        }
        fmt++;
    }
    return counter;
}

char *mygets(char *dst){
    char *pointer   =   dst;
    char temp;

    for(;;){
        temp    =   getchar();
        /* if met EOF */
        if(temp==EOF){
            *pointer='\0';
            return NULL;
        }
        /* if met the space character */
        else if(temp=='\r' || temp=='\n' || temp==' '){
            if(temp=='\r'){
                putchar('\n');
                break;
            }
            else if(temp=='\n')
                break;
            else if(temp==' ')
```

```
                break;
            /* writer the character */
        }

        *pointer++ = temp;
    }
    *pointer = '\0';
    return dst;
}

int myputs(const char *string){
    for(; *string ; string++)
        putchar(*string);
    putchar('\n');
    return 0;
}

/* convert n to characters in s */
void myitoa(int n, char s[]){
    int i,j,k, sign;
    char c;

    /* if n is negative, convert it to positive */
    if ((sign = n) < 0)
        n   =   -n;
    i   =   0;

    /* generate digits in reverse order */
    do{
        s[i++]  =   n%10+'0';
    }while ((n/=10) > 0);
```

```
    /* if is negative, add the negative sign */
    if (sign < 0)
    s[i++]  =   '-';
    s[i]    =   '\0';

    /*in order not to use strlen */
    for(j=0;s[j]!='\0';j++);
    /* reverse s to the correct order */
    for (k=0, j--; k<j; k++, j--) {
        c       =   s[k];
        s[k]    =   s[j];
        s[j]    =   c;
    }
}


static double PRECISION = 0.00000000000001;

/* my dtoa (double to string), convert double and float to
string  */
void mydtoa(double n, char s[]){
    /* if is a special number */
    if(isnan(n))
        s[0]='n',s[1]='a',s[2]='n',s[3]='\0';
    else if(isinf(n))
        s[0]='i',s[1]='n',s[2]='f',s[3]='\0';
    else if(n==0.0)
        s[0]='0',s[1]='\0';
    else{
        int digit, m, m1;
        char *c =   s;
```

```
        int neg =   (n<0);
        if (neg)
            n   =   -n;
        /* calculate magnitude */
        m   =   log10(n);
        int useExp  =   (m>=14  ||  (neg && m>=9)   ||
m<=-9);
        if (neg)
            *(c++)  =   '-';
        /* if use the scientific notation */
        if (useExp) {
            if (m<0)
                m   -=  1.0;
            n   =   n/pow(10.0, m);
            m1  =   m;
            m   =   0;
        }
        if(m<1.0)
            m   =   0;
        /* convert numbers */
        while (n>PRECISION  ||  m>=0) {
            double weight   =   pow(10.0, m);
            if (weight>0 && !isinf(weight)) {
                digit   =   floor(n / weight);
                n       -=  (digit * weight);
                *(c++)  =   '0' + digit;
            }
            if (m==0 && n>0)
                *(c++)  =   '.';
            m--;
        }
```

```
        if(useExp){
            /* convert the exponent */
            int i, j;
            *(c++)  =   'e';
            if(m1>0)
                *(c++)  =   '+';
            else{
                *(c++)  =   '-';
                m1      =   -m1;
            }
            m   =   0;
            for(;m1>0;) {
                *(c++)  =   '0'+m1%10;
                m1      /=  10;
                m++;
            }
            c   -=  m;
            /* reverse the string */
            for (i=0,j=m-1; i<j; i++,j--) {
                c[i]    ^=  c[j];
                c[j]    ^=  c[i];
                c[i]    ^=  c[j];
            }
            c   +=  m;
        }
        *(c)    =   '\0';
    }
}
```

test.c：

```
#include "mylibrary.h"
#include "string.h"
#include <stdio.h>

int main(void){
    char    c;
    int     i;
    float   f;
    double  d;
    char    s[100];

    myscanf("%c %d %f %lf %s",&c, &i, &f, &d,
s);getchar();
    myprintf(   "\ngeneral format------------------------
------------------\n"
                "the char is   : %c\n"
                "the int is    : %d\n"
                "the float is  : %f\n"
                "the double is : %f\n"
                "the string is : %s\n",
                c,i,f,d,s);

    myprintf(   "\noutput with width 20-------------------
------------------\n"
                "the char is   : %20c\n"
                "the int is    : %20d\n"
                "the float is  : %20f\n"
                "the double is : %20f\n"
                "the string is : %20s\n",
```

```
                c,i,f,d,s);

    myprintf(   "\noutput with width 20 and aligned left--
------------------\n"
                "the char is   : %-20c\n"
                "the int is    : %-20d\n"
                "the float is  : %-20f\n"
                "the double is : %-20f\n"
                "the string is : %-20s\n",
                c,i,f,d,s);

    myprintf(   "\noutput with width 20 and precision 3---
------------------\n"
                "the char is   : %20.3c\n"
                "the int is    : %20.3d\n"
                "the float is  : %20.3f\n"
                "the double is : %20.3f\n"
                "the string is : %20.3s\n",
                c,i,f,d,s);

    myprintf(   "\noutput with width 20 aligned left and
precision 3--------\n"
                "the char is   : %-20.3c\n"
                "the int is    : %-20.3d\n"
                "the float is  : %-20.3f\n"
                "the double is : %-20.3f\n"
                "the string is : %-20.3s\n",
                c,i,f,d,s);
```

```
    myputs(     "\nmygets and myputs
test==================================\n");
    mygets(s);
    myputs(s);
    return 0;
}
```

六、运行测试与结果分析

运行结果充分说明了函数的可用性以及正确性，能够对不同的格式输出正确的结果，以下为运行截图：

```
a   2    123.4567       1.2345678    hello_world

general format-------------------------------------------
the char is   : a
the int is    : 2
the float is  : 123.456703
the double is : 1.234567
the string is : hello_world

output with width 20-------------------------------------
the char is   :                    a
the int is    :                    2
the float is  :           123.456703
the double is :             1.234567
the string is :          hello_world

output with width 20 and aligned left--------------------
the char is   : a
the int is    : 2
the float is  : 123.456703
the double is : 1.234567
the string is : hello_world

output with width 20 and precision 3----------------------
the char is   :                    a
the int is    :                    2
the float is  :              123.456
the double is :                1.234
the string is :          hel

output with width 20 aligned left and precision 3--------
the char is   : a
the int is    : 2
the float is  : 123.456
the double is : 1.234
the string is : hel

mygets and myputs test==================================

echo_this_sentense
echo_this_sentense

Process returned 0 (0x0)   execution time : 58.176 s
Press any key to continue.
```

七、总结

通过这个实验，我对标准输入输出库的底层实现有了更深入的了解，并用 getchar 与 putchar 实现了较为完整的输入输出出库，经过测试代码能实现的功能的运行结果与 stdio.h 中的 printf 和 scanf 一致，没有出现差错。

对比 linux 源码中的 vsprintf 代码，我了解到了自己代码功能的不完善，不能很好的实现 scanf 和 printf 的所有格式化输出功能，并且也没有调用更底层的 read 和 write 来编写 myprintf 和 myscanf 函数，此外还可以利用位运算来简化格式标签，这也是值得注意的地方。

Simulator and Assembler

一、 系统需求分析

一、 总体需求和计划

1. 编写一个汇编器和一个模拟器，汇编器能够将指定的汇编源码编译成二进制代码，而模拟器能够执行这种二进制代码运行得到正确的结果。
2. 汇编器和模拟器编写成一个程序，使用命令行参数来决定使用编译器还是模拟器，编译的文件名等等。

二、 输入数据

1. Assembler 的输入数据为一个汇编文件，其编写格式不定（即空格、制表符、命令或注释的位置是任意的，只要合法即可）。
2. Simulator 的输入文件是 Assembler 的输出二进制文件。

三、 输出数据

1. Assembler 的输出数据是一个 Simulator 能够执行的二进制文件，包含指令与初始化的数据，需保证其正确性。
2. Simulator 的输出是运行二进制文件的输出结果，依赖于源汇编文件的内容。

四、 功能要求

1. 对任意编写的合法的汇编文件要能够运行汇编器将其编译成正确的二进制文件。
2. 二进制文件要能够被正确的执行，其执行结果应与按照数学逻辑推理得到的程序执行结果一致。
3. 对于不合法的汇编文件，编译时要指出错误的类型，错误出现的位置，方便程序员修改。
4. 程序段，数据段的大小编译时需要决定，同时决定堆栈段和附加段的总大小。堆栈段和附加段在运行时地址不能重叠。
5. 对于运行时错误，模拟器要能给出错误原因和错误地址。

五、 性能要求

1. 编译器要能以线性时间复杂度完成对整个程序的编译，否则当程序过大时，时间会过长。
2. 编译器和模拟器的健壮性要高，本身不能出现崩溃，错误处理要全面。
3. 编译器和模拟器对于汇编文件的编译和模拟不能出现逻辑错误。

二、总体设计

一、 Assembler（汇编器）总体设计

1. 汇编器输入文件后，先进行三步预处理：

   1) 第一步，去除所有的注释和空行

   2) 第二步，为所有的变量声明（BYTE 和 WORD）分配空间并记录变量名与数据地址，为所有语句注明地址并记录所有标签（LABEL）的名称与所在语句的地址,在每一次记录时删除该声明或标签。

   3) 第三步，根据 2)的结果，将程序语句中的所有变量和跳转语句中的标签换为对应的地址。

2. 然后进行编译：将所有的程序语句合成为对应的二进制代码，将初始化的变量写入二进制文件。

3. 在预处理和编译的过程中采用哈希表存储数据，以减少编译所需时间。

4. 在上述所有过程中，遇到任何源文件不合法，目标文件不能打开，空间不足等情况都需要做错误处理，并以统一的一个结构返回，方便上层函数处理。

二、 Simulator（模拟器）总体设计

1. 相对于汇编器，模拟器的实现较为容易。读入二进制文件后，为初始化的数据分配空间，为程序分配内存空间。然后依照质量大小一次读入指令并执行即可。

2. 在上述过程中需要检查运行时错误，尤其是地址重叠、内存溢出等错误。

三、数据结构设计

一、 对于编译时错误处理的数据结构：

每个函数的返回值类型为 error_t，并使用了一个结构体定义了具体错误的位置，作为最后一个参数将其指针传递给一个函数，便于通过参数返回值确定错误的类型。其声明如下：

```
typedef char error_t;
#define no_error                        0x00
#define error_open_file                 0x01
/* grammar error*/
#define error_gramma_label_not_first    0x02
#define error_gramma_declare_incomplete 0x03
#define error_gramma_duplicate_symbol   0x04
#define error_token_not_get             0x05
#define error_gramma_declare_is_keyword 0x06
#define error_gramma_lack_symbol        0x07
#define error_gramma_duplicate_command  0x08
#define error_gramma_unidentified       0x09
#define error_gramma_too_less_argument  0x0A
#define error_gramma_too_much_argument  0x0B
#define error_gramma_declare_is_digit   0x0C
#define error_gramma_reg_illegal        0x0D

#define error_malloc                    0x0E
#define error_stack_overflow            0x0F

#define error_how_could_it_be           0x10

#define MAX_ERROR_STRLEN    200
typedef struct _Error{
```

```
    /* the error line or error message */
    char error_line[MAX_ERROR_STRLEN];
    /* point to the position when grammar error occurs */
    char error_pos;
}Error;
```

通用的函数原型为

```
error_t prototype(…, …, *Error);
```

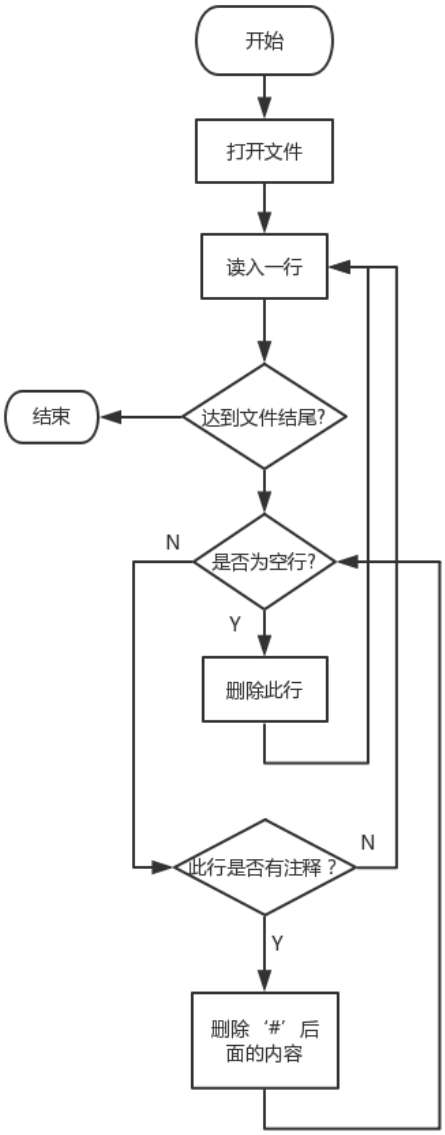
二、 变量地址和标签（LABEL:）地址临时存储的数据结构：

1. 采用哈希表存储。由于要频繁的查找命令所对应的二进制代码，标签和变量所对应的二进制地址以及标签是否重复，用链表或数组存储非常费时，每一次都需要遍历整个链表。因此，本程序采用哈希表存储，能够在 O(1)时间复杂度内找到所需的信息，使得能够在线性时间复杂度内完成编译。
2. 对于指令集的 32 条指令名，经测试，HASH_MASK 只需要达到 0x03FF，即只需要 1024 条指令的空间即可达到完全无碰撞，省去了哈希表上硬碰撞而产生接的链表，空间利用率达到 1/32，但总空间不大，而且是固定大小的。
3. 对于变量地址和标签地址，采用哈希表+链表存储结构体的方式，亦可以大大减少编译所需时间。

四、详细设计

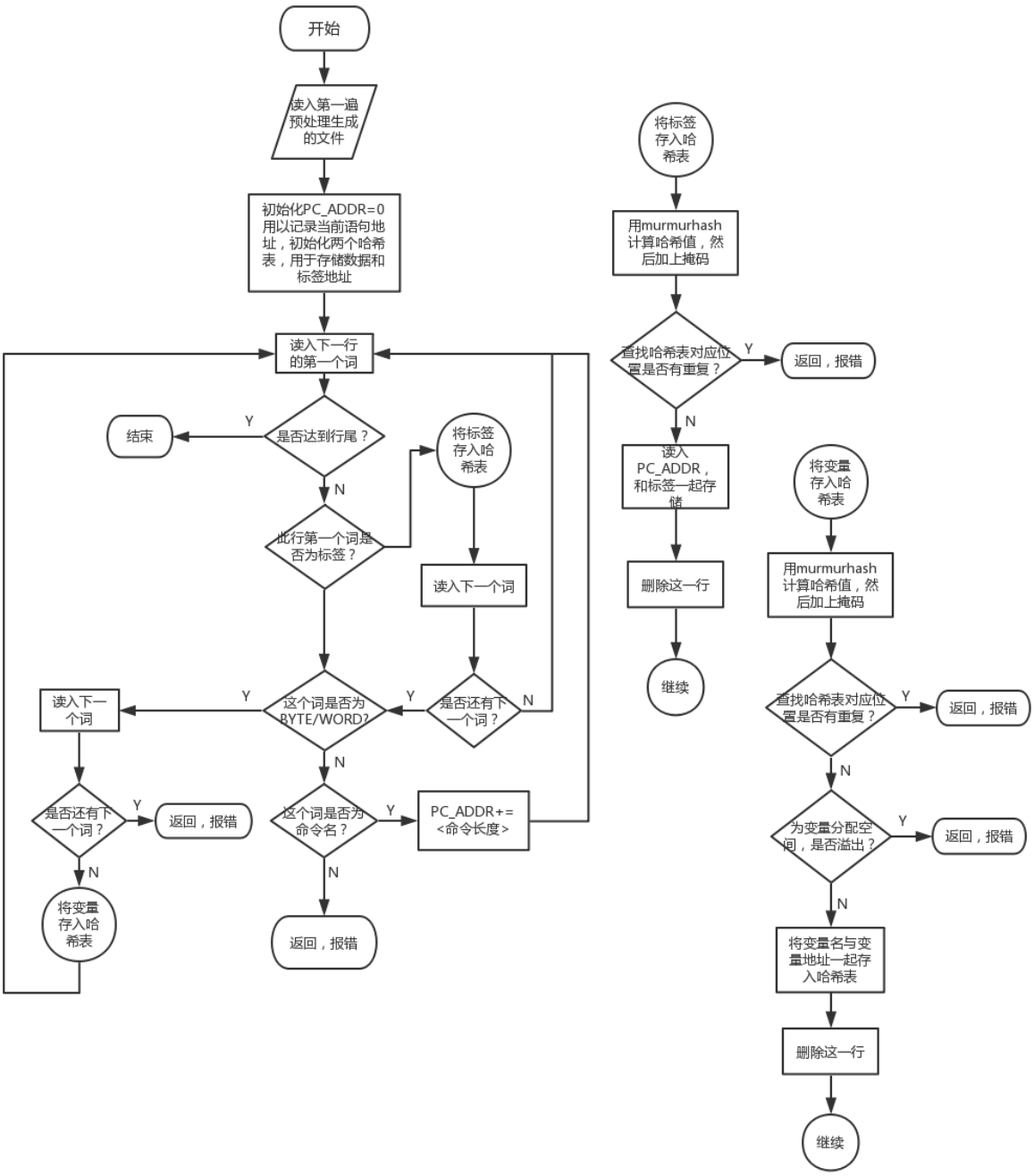
一、 第一遍预处理

删除所有的注释和空行，如果删除注释后是一条空行，则也删除此行。具体的流程图如下：



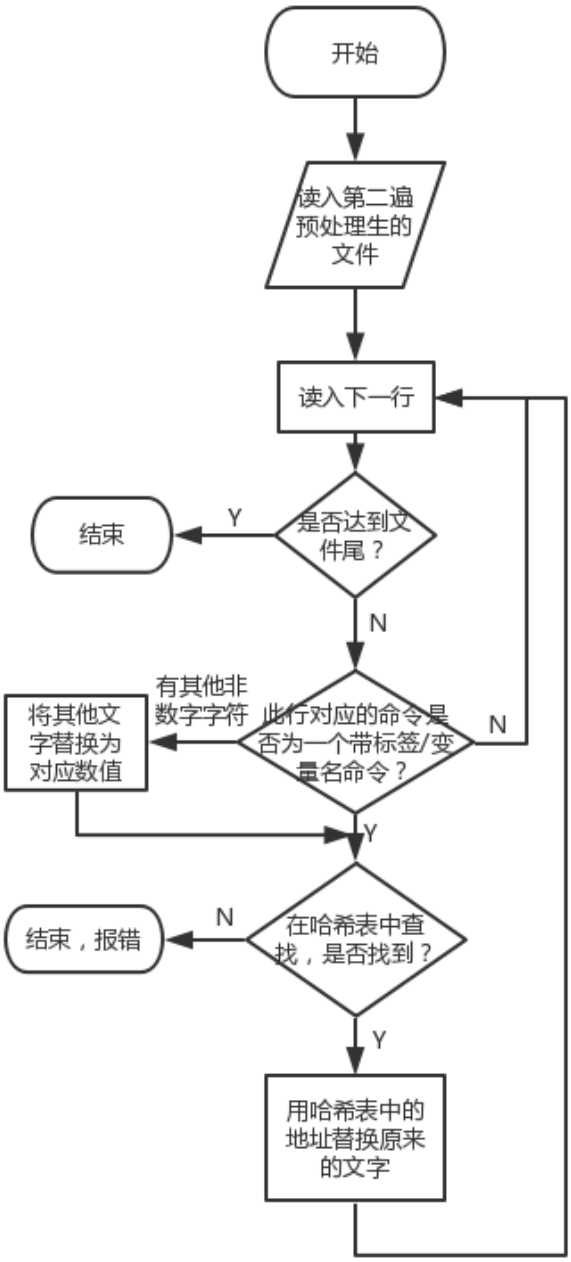
二、 第二遍预处理

扫描整个文件，为所有的变量声明分配空间，为所有的跳转标签找到地址，并将二者记录在哈希表中，同时查重。一下流程图所说的“词”指以空白字符分隔的连续的字符组成的字符串。



三、 第三次预处理

将命令中所有的跳转标签和变量替换为对应的地址，寄存器等字符也替换为对应的值。若发现未存储的变量名或标签则报错。



四、 编译

将所有的命令及其参数合成为二进制代码，如果有参数类型错误，则报错，由于预处理已经比较完善，用简单的位运算即可实现，故不再各处流程图。

五、 模拟

读入二进制文件，初始化内存和各个寄存器，每次读取一条命令，用位运算（&MASK）分别取出命令值，参数值，然后根据命令值调用不同的函数即可，期间若有运行时错误则报错。

五、系统实现

将所有源文件编译为可执行文件即可。编译命令:

```
export CPATH=$(INC_DIRS)
gcc -Wall -g -std=c99 -o$(TARGET) $(SRC)
```

其中 TARGET 为编译目标，SRC 为所有源文件，INC_DIRS 为头文件目录，以':'分隔。下面给出程序源码。

main.h

```
#ifndef MAIN_H_INCLUDED
#define MAIN_H_INCLUDED

void show_help(void);

#endif // MAIN_H_INCLUDED
```

main.c

```
#include "main.h"
#include "assembler.h"
#include "simulator.h"
#include "hash.h"
#include "common.h"
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

#define MASK 0x03FF

error_t p_error(error_t error_type);
```

```
int main(int argc, char *argv[]){
    /* do the pretreatment */
    if(argc==1){
        show_help();
        return 0;
    }
    if(argv[1][0]!='-' || strlen(argv[1])!=2){
        show_help();
        return 0;
    }

    char a[1000];
    char b[1000];

    switch(argv[1][1]){
        /* Assembler (compiler) */
        case 'c':
            /* read the src and dst file name*/
            if(argc!=3 && argc!=4){
                printf("USAGE : asm-sim -c <src>
[dst]\n");
                break;
            }
            strcpy(a,argv[2]);
            /* if no dst name */
            if(argc==3){
                strcpy(b,a);
                strcat(b,".s");
            }
            /* if has dst name */
            else
```

```
                strcpy(b,argv[3]);

            assembler_init();

            Error error;
            /* do the compile job and output the error */
            if(p_error(assembler(a, b, &error)) !=
no_error)
                printf("%-
*s\n",error.error_pos,"^~~~here\n");

            break;
        /* Simulator */
        case 's':
            if(argc!=3){
                printf("USAGE : asm-sim -s <file> \n");
                break;
            }
            strcpy(b,argv[2]);

            simulator_init();
            if(p_error(simulator(b, &error))!=no_error)
                printf("%-
*s\n",error.error_pos,"^~~~here\n");
            break;
        /* help */
        case 'h':
            show_help();
            break;
    }
```

```
    return  0;
}


void show_help(void){
    printf(
    "\n USAGE: asm-sim <COMMAND> [arg1] [arg2] \n"
    "\n COMMAND:\n"
    "\n -h : show this help doc\n"
    "\n -c : compile the assembler file to binary file \n"
    "      arg1 : the source asm file\n"
    "      arg2 : the dst file\n"
    "      if arg2 is not specified, the assembler will
create arg1.s\n"

    "\n -s  :  loade a binary file and simulate it\n"

    "      arg1 : the file to excute\n"

    "\nemail  :  husixu1@hotmail.com\n"

    "github  :  github.com/husixu1\n\n");


}

/* output the errors */
inline error_t p_error(error_t error_type){
    switch(error_type){
        /* common error */
        case no_error                        :
            printf("no error occurred.\n");
            return error_type;
```

```
        case error_open_file                :
            printf("cannot open file.\n");
            break;
        /* grammar error*/
        case error_gramma_label_not_first   :
            printf("label must be the first of a
command:\n");
            break;
        case error_gramma_declare_incomplete:
            printf("declaration incomplete:\n");
            break;
        case error_gramma_duplicate_symbol  :
            printf("duplicate symbol:\n");
            break;
        case error_token_not_get            :
            printf("incomplete command:\n");
            break;
        case error_gramma_declare_is_keyword:
            printf("declaration can not be a keyword:\n");
            break;
        case error_gramma_lack_symbol       :
            printf("incomplete command:\n");
            break;
        case error_gramma_duplicate_command :
            printf("there should be no command after a
declaration:\n");
            break;
        case error_gramma_unidentified      :
            printf("unidentified command:\n");
            break;
        case error_gramma_too_less_argument :
```

```
            printf("too less argument:\n");
            break;
        case error_gramma_too_much_argument :

            printf("too many argument：\n");

            break;
        case error_gramma_declare_is_digit  :
            printf("a declaration cannot start with a
digit:\n");
            break;
        case error_gramma_reg_illegal       :
            printf("illegal register:\n");
            break;
        /* memory error */
        case error_malloc                   :
            printf("no more spaces for mallocing!\n");
            break;
        case error_stack_overflow           :
            printf("stack overflow.\n");
            break;
        case error_how_could_it_be          :
            printf("unknown error, please contact the
author.\n");
            break;
    }
    return error_type;
}
```

common.h

```
#ifndef COMMON_H_INCLUDED
#define COMMON_H_INCLUDED

#include <stdio.h>

typedef char error_t;
#define no_error                        0x00
#define error_open_file                 0x01
/* grammar error*/
#define error_gramma_label_not_first    0x02
#define error_gramma_declare_incomplete 0x03
#define error_gramma_duplicate_symbol   0x04
#define error_token_not_get             0x05
#define error_gramma_declare_is_keyword 0x06
#define error_gramma_lack_symbol        0x07
#define error_gramma_duplicate_command  0x08
#define error_gramma_unidentified       0x09
#define error_gramma_too_less_argument  0x0A
#define error_gramma_too_much_argument  0x0B
#define error_gramma_declare_is_digit   0x0C
#define error_gramma_reg_illegal        0x0D

#define error_malloc                    0x0E
#define error_stack_overflow            0x0F

#define error_how_could_it_be           0x10
typedef unsigned char   uint8_t;
typedef unsigned short  uint16_t;
typedef unsigned int    uint32_t;
typedef short           int16_t;
```

```
#define MAX_ERROR_STRLEN    200
typedef struct _Error{
    /* the error line or error message */
    char error_line[MAX_ERROR_STRLEN];
    /* point to the position when grammar error occurs */
    char error_pos;
}Error;

/* the length of data stack*/
#define MAX_DATA_SPACE  (1<<23)

#endif // COMMON_H_INCLUDED
```

assembler.h

```
#ifndef ASSEMBLER_H_INCLUDED
#define ASSEMBLER_H_INCLUDED

#include "common.h"
/* the mask of hash result */
/* enlarge it if hash result collied, when you're trying
to change the operator set */
#define HASH_MASK   0x03FF
/* the max length of the reading buffer (also the max
lenth of a line )*/
#define MAX_BUF_LEN     500

/*==================================================
===================*/
/* the number of operator */
```

```
#define OP_NUM  32
/* the max length of operator*/
#define OP_MXLEN 8

/* stores the command format
*/
/* 'o' for operator, 'r' for register, 'a' for address,
'i' for immediate   */
/* 'p' for padding, 't'for port, the number follows stands
for bits          */
typedef char format_t;
#define o5_p27          0x01
#define o5_p3_a24       0x02
#define o5_r3_p24       0x03
#define o5_r3_a24       0x04
#define o5_r3_p8_i16    0x05
#define o5_r3_p16_t8    0x06
#define o5_r3_r4_r4_p16 0x07
#define o5_r3_r4_p20    0x08

/* the operator */
typedef struct _Op{
    char        name[OP_MXLEN];     /* the name of the
operator */
    uint32_t    value;              /* the value of the
operator (use with '|') */
    format_t    format;             /* the format of the
operator*/
}Op;
/* the hash table that stores the operator value(key is
the name) */
```

```
extern Op op_hash[HASH_MASK];

/*==========================================================
====================*/

/* initialize the operator hash table */
error_t assembler_init(void);
/* the main assembler function */
error_t assembler(const char *src, const char *dst, Error
*error);


/* get next token on buffer from *temp position */
/* buffer must end with \n\0 and *temp must be on the
buffer */
error_t __get_nth_token(const char *buffer, char
*buffer_sub, char n);

#endif // ASSEMBLER_H_INCLUDED
```

assembler.c

```
#include "assembler.h"
#include "assembler_pretreatment.h"
#include "assembler_replacement.h"
#include "hash.h"
#include "common.h"
#include <string.h>
#include <stdlib.h>
```

```
/* the operator set */
const Op op_set[OP_NUM] =   {
    {"HLT",     (0<<27),    o5_p27          },  {"JMP",
(1<<27),    o5_p3_a24       },
    {"CJMP",    (2<<27),    o5_p3_a24       },  {"OJMP",
(3<<27),    o5_p3_a24       },
    {"CALL",    (4<<27),    o5_p3_a24       },  {"RET",
(5<<27),    o5_p27          },
    {"PUSH",    (6<<27),    o5_r3_p24       },  {"POP",
(7<<27),    o5_r3_p24       },
    {"LOADB",   (8<<27),    o5_r3_a24       },  {"LOADW",
(9<<27),    o5_r3_a24       },
    {"STOREB",  (10<<27),   o5_r3_a24       },  {"STOREW",
(11<<27),   o5_r3_a24       },
    {"LOADI",   (12<<27),   o5_r3_p8_i16    },  {"NOP",
(13<<27),   o5_p27          },
    {"IN",      (14<<27),   o5_r3_p16_t8    },  {"OUT",
(15<<27),   o5_r3_p16_t8    },
    {"ADD",     (16<<27),   o5_r3_r4_r4_p16 },  {"ADDI",
(17<<27),   o5_r3_p8_i16    },
    {"SUB",     (18<<27),   o5_r3_r4_r4_p16 },  {"SUBI",
(19<<27),   o5_r3_p8_i16    },
    {"MUL",     (20<<27),   o5_r3_r4_r4_p16 },  {"DIV",
(21<<27),   o5_r3_r4_r4_p16 },
    {"AND",     (22<<27),   o5_r3_r4_r4_p16 },  {"OR",
(23<<27),   o5_r3_r4_r4_p16 },
    {"NOR",     (24<<27),   o5_r3_r4_r4_p16 },  {"NOTB",
(25<<27),   o5_r3_r4_p20    },
    {"SAL",     (26<<27),   o5_r3_r4_r4_p16 },  {"SAR",
(27<<27),   o5_r3_r4_r4_p16 },
    {"EQU",     (28<<27),   o5_r3_r4_p20    },  {"LT",
```

```
(29<<27),   o5_r3_r4_p20    },
    {"LTE",     (30<<27),   o5_r3_r4_p20    },  {"NOTC",
(31<<27),   o5_p27          }
};

/* the hash table that stores the operator value and
formate (key is the name) */
/* HASH_MASK is big enough to ensure there's no collide in
the hash table */
Op op_hash[HASH_MASK];

error_t assembler_init(void){
    memset(op_hash,0,sizeof(Op)*HASH_MASK);
    /* initialize the op_hash table */
    for(int i=0; i<OP_NUM; i++)
        op_hash[hash_32(op_set[i].name, HASH_SEED) &
HASH_MASK] =   op_set[i];
    return no_error;
}

/* the main assembler function */
error_t assembler(const char *src, const char *dst, Error
*error){
    error_t error_type;
    char *temp1 =   (char
*)malloc(sizeof(char)*(strlen(dst)+3));
    char *temp2 =   (char
*)malloc(sizeof(char)*(strlen(dst)+6));
    strcpy(temp1,dst);
    strcpy(temp2,dst);
    /* do the pretreatment stage 0 */
```

```
    if((error_type=assembler_pretreatment_stage0(src,
strcat(temp1,"_s0") ))!=no_error)
        return error_type;
    /* do the pretreatment stage 1 */
    if((error_type=assembler_pretreatment_stage1(temp1,
strcat(temp2,"_s1"), error))!=no_error)
        return error_type;
    /* do the replacement */
    if((error_type=assembler_replacement(temp2, dst,
error))!=no_error)
        return error_type;
    return no_error;
}


/* get the Nth(start from 0) token on buffer */
/* buffer must end with \n\0 */
error_t __get_nth_token(const char *buffer, char
*buffer_sub, char n){
    /* initialize the format string */
    char *fmt    =    (char *)malloc(sizeof(char)*4*(n+1));
    fmt[0]       =    '\0';
    for(int i=0; i<n; i++)
        strcat(fmt, "%*s ");
    strcat(fmt,"%s");
    /* get the nth token */
    if(sscanf(buffer,fmt,buffer_sub)!=1)
        return  error_token_not_get;
    return no_error;
}
```

assembler_pretreatment.h

```
#ifndef ASSEMBLER_PRETREATMENT_H_INCLUDED
#define ASSEMBLER_PRETREATMENT_H_INCLUDED

#include "common.h"
#include <stdio.h>


/* struct that stores the label/variable and its address
*/
typedef struct _Addr{
    char            is_set;     /* if the symbol is a
declaration, mark if it's a set (2 for label)*/
    char            symbol[MAX_BUF_LEN];
    uint32_t        addr;
    struct _Addr    *next;
}Addr;

error_t assembler_pretreatment_stage0(const char *src,
const char *dst);
error_t assembler_pretreatment_stage1(const char *src,
const char *dst, Error *error);

/* the temporary data stack */
/* the structure of data stack: |space|val|space|val|...|
*/
extern int16_t data_stack[MAX_DATA_SPACE];
/* store the current reading address (metric: 2Byte)
(for :LABEL) */
extern uint32_t pc_addr;
```

```
#endif // ASSEMBLER_PRETREATMENT_H_INCLUDED
```

assembler_pretreatment.c

```
#include "assembler.h"
#include "assembler_pretreatment.h"
#include "hash.h"
#include "common.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <math.h>

/* the hash table of label address */
Addr *addr_table[HASH_MASK];
/* the temporary data stack */
int16_t data_stack[MAX_DATA_SPACE];
/* store the current reading address (metric: 2Byte)
(for :LABEL) */
uint32_t pc_addr    =   0;

/* function that stores the label or declaration in hash
table*/
static error_t __hash_store(const char *symbol, const
uint32_t addr);

#define __return_error(error_code)  \
{strcpy(error->error_line,buffer);\
error->error_pos    =   (char)(strstr(buffer,buffer_sub)-
```

```
buffer);\
return  error_code;}

/* delete all the comments and empty lines int the source
file */
error_t assembler_pretreatment_stage0(const char *src,
const char *dst){
    /* opening necessary files */
    FILE *file_src  =   fopen(src, "r");
    FILE *file_dst  =   fopen(dst, "w");
    if(!file_src || !file_dst)
        return error_open_file;
    /* for analyze the line */
    char    buffer[MAX_BUF_LEN];
    char    buffer_sub[MAX_BUF_LEN];
    char    *temp;
    for(;fgets(buffer, MAX_BUF_LEN, file_src);){
        if((temp=strchr(buffer, '#'))!=NULL){
            *temp   =   '\n';
            *++temp =   '\0';
        }
        if(__get_nth_token(buffer, buffer_sub,
0)==error_token_not_get);
        else
            fputs(buffer, file_dst);
    }
    fclose(file_src);
    fclose(file_dst);
    return  no_error;
}
/* substitute the BYTE, WORD declaration and :LABEL */
```

```
/* returning temp_dst(the temp file of pre-treatment */
error_t assembler_pretreatment_stage1(const char *src,
const char *dst, Error *error){
    /* initialize the address table */
    for(int i=0; i<HASH_MASK; i++){
        addr_table[i]       =   malloc(sizeof(Addr));
        addr_table[i]->next =   NULL;
    }
    /* initialize the temp data stack */
    memset(data_stack,0,sizeof(int16_t)*MAX_DATA_SPACE);

    /* open necessary files */
    FILE *file_src  =   fopen(src, "r");
    FILE *file_dst  =   fopen(dst, "w");
    if(!file_src || !file_dst)
        return error_open_file;

    /* store the reading line */
    char buffer[MAX_BUF_LEN];
    /* analyze the reading line */
    char buffer_sub[MAX_BUF_LEN];

    /* store the current data distribution address
(metric: 2Byte) (relative addr, start from DS) */
    uint32_t data_addr  =   0;
    /* mark if this line has a label */
    char flag_label     =   0;

    /*** stage one, find the labels and declarations ***/
    for(; fgets(buffer, MAX_BUF_LEN, file_src)!=NULL;
pc_addr+=2){
```

```
        flag_label  =   0;
        /* if is the empty line */
        if(__get_nth_token(buffer, buffer_sub,
0)==error_token_not_get){
            pc_addr-=2;
            continue;   /* next line */
        }
        /* switch(the first token) */
        /* if this line *has* a label */
        if(buffer_sub[strlen(buffer_sub)-1]==':'){
            /* store the label into hash table */
            buffer_sub[strlen(buffer_sub)-1]    =   '\0';
            /* if find duplicate symbol */
            if(__hash_store(buffer_sub,
pc_addr)==error_gramma_duplicate_symbol)

__return_error(error_gramma_duplicate_symbol);
            /* get the next token */
            if(__get_nth_token(buffer, buffer_sub,
1)==error_token_not_get){
                pc_addr-=2;
                continue;
            }
            flag_label  =   1;
        }
        /* if this line is a byte or world declaration */
        if(strcmp(buffer_sub,"BYTE")==0 ||
strstr(buffer,"WORD")){
            /* if there noting next */
            if(__get_nth_token(buffer, buffer_sub,
flag_label+1)==error_token_not_get)
```

```
                __return_error(error_gramma_lack_symbol);

            /* distribute data space */
            int space    =    0;
            char *temp   =    strstr(buffer, buffer_sub);
            /* make sure the symbol is legal */
            if(isdigit(*temp))

__return_error(error_gramma_declare_is_digit);
            int i=0;
            for(; isalnum(*temp) || *temp=='_'; temp++,
i++)
                buffer_sub[i]    =    *temp;
            buffer_sub[i]='\0';
            /* if the symbol is a keyword */
            if(strcmp(op_hash[hash_32(buffer_sub,
HASH_SEED) & HASH_MASK].name,buffer_sub)==0)

__return_error(error_gramma_declare_is_keyword);
            /* store the name and continue */
            if(__hash_store(buffer_sub,
data_addr)==error_gramma_duplicate_symbol)

__return_error(error_gramma_duplicate_symbol);

            /* if it is not a set */
            if(__get_nth_token(temp, buffer_sub,
0)==error_token_not_get){
                /* mark that it's not a set */
                data_stack[data_addr]    =    1;
                space = 1;
```

```
                goto end;
            }
            temp    =   strstr(buffer, buffer_sub);
            /* otherwise */
            if(*temp=='['){
                temp++;
                for(; isspace(*temp); temp++);
                for(; isdigit(*temp); temp++)
                    space   =   space*10 + (*temp) - '0';
                for(; isspace(*temp); temp++);
                if(*temp != ']')

__return_error(error_gramma_declare_incomplete);
                temp++;
                /* mark that it's a set */
                data_stack[data_addr]   =   space;
            }

            /* if not set the value */
            if(__get_nth_token(temp, buffer_sub,
0)==error_token_not_get){
                goto end;
            }

            temp    =   strstr(buffer, buffer_sub);
            if(*temp=='='){
                temp++;
                for(; isspace(*temp); temp++);
                /* if it is a set */
                if(*temp=='{'){
                    int counter =   0;
```

```
                    uint32_t data;
                    char fmt[MAX_BUF_LEN]   =   "%d";
                    for(;sscanf(temp,fmt,&data)==1  &&
counter<=space ;counter++){
                        data_stack[data_addr+counter+1] =
data;
                        fmt[0]  =   '\0';
                        for(int i=0; i<counter; i++)
                            strcat(fmt,"%*d");
                        strcat(fmt,"%d");
                    }
                    /* make the pointer be out of the set
*/
                    for(; *temp!='}'; temp++);
                    temp++;
                }
                /* if it is a string */
                else if(*temp==''"'){
                    int counter =   0;
                    temp++;
                    for(; *temp!=''"'  && counter <= space;
temp++,counter++)
                        data_stack[data_addr+counter+1] =
*temp;
                    temp++;
                }
                /* if it's a number */
                else if(isdigit(*temp)){
                    space   =   1;
                    sscanf(temp, "%hd",
&data_stack[data_addr+1]);
```

```
                    for(; isdigit(*temp); temp++);
                }
                else

__return_error(error_gramma_too_less_argument);
            }
            else

__return_error(error_gramma_too_much_argument);

            end:
            /* distribute space(relative addr ) */
            data_addr+=(space+1);
            if(data_addr>=MAX_DATA_SPACE)
                return  error_stack_overflow;

            /* if there's still token after the
declaration and it's not comment */
            if( __get_nth_token(temp, buffer_sub,
0)==no_error){

__return_error(error_gramma_duplicate_command);
            }

            pc_addr -=  2;
            continue;   /* next line */
        }

        /* if is a keyword (test if the number of token is
right ) */
        if(strcmp(op_hash[hash_32(buffer_sub, HASH_SEED) &
```

```
HASH_MASK].name,buffer_sub)==0){
            int num=0, counter=0;
            switch(op_hash[hash_32(buffer_sub, HASH_SEED)
& HASH_MASK].format){
                case o5_p27:
                    num =   0;
                    break;
                case o5_p3_a24:
                case o5_r3_p24:
                    num =   1;
                    break;
                case o5_r3_a24:
                case o5_r3_p16_t8:
                case o5_r3_p8_i16:
                case o5_r3_r4_p20:
                    num =   2;
                    break;
                case o5_r3_r4_r4_p16:
                    num =   3;
                    break;
            }
            /* make sure the number of argument is correct
*/
            for(counter=0; counter<num; counter++)
                if(__get_nth_token(buffer, buffer_sub,
flag_label+counter+1)==error_token_not_get)

__return_error(error_gramma_too_less_argument);
            if(__get_nth_token(buffer,
buffer_sub,flag_label+counter+1)==no_error)
```

```
__return_error(error_gramma_too_much_argument);
            continue;
        }
        /* it is noting */
        else
            __return_error(error_gramma_unidentified);
    }
    /*** stage two, replace all the symbols with address,
write into temp_dst ***/
    rewind(file_src);
    for(;fgets(buffer, MAX_BUF_LEN, file_src);){
        flag_label  =   0;
        /* ignore empty line */
        if(__get_nth_token(buffer, buffer_sub,
0)==error_token_not_get)
            continue;
        /* has label */
        if(buffer_sub[strlen(buffer_sub)-1]==':'){
            flag_label  =   1;
            /* get next token */
            if(__get_nth_token(buffer, buffer_sub,
1)==error_token_not_get)
                continue;
        }
        /* ignore declaration */
        if(strcmp(buffer_sub,"BYTE")==0 ||
strcmp(buffer_sub,"WORD")==0)
            continue;
        /* replace label and declarations */
        if(strcmp(op_hash[hash_32(buffer_sub,HASH_SEED) &
HASH_MASK].name, buffer_sub)==0){
```

```
            Addr    *temp;
            int i=0;
            /* print the operator into temp_dst */
            i   =   flag_label?1:0;
            __get_nth_token(buffer, buffer_sub, i);
            fprintf(file_dst,"\t%s",buffer_sub);
            /* print the next argument */
            for(i=0; __get_nth_token(buffer, buffer_sub,
i+flag_label+1)==no_error; i++){

temp=addr_table[hash_32(buffer_sub,HASH_SEED) &
HASH_MASK];
                /* try to find the symbol in hash
(whatever it is) */
                for(temp=temp->next; temp!=NULL;
temp=temp->next){
                    if(strcmp(temp->symbol,buffer_sub)==0)
                        break;
                }
                if(temp==NULL)
                    fprintf(file_dst, "\t%s", buffer_sub);
                else
                    fprintf(file_dst, "\t%u", temp->addr);
            }
            fprintf(file_dst, "\n");
        }
        else
            return error_how_could_it_be;
    }
    fclose(file_src);
    fclose(file_dst);
```

```
    Addr *temp, *temp_shadow;
    for(int i=0; i<HASH_MASK; i++){
        temp        =   addr_table[i];
        temp_shadow =   temp;
        for(;temp!=NULL;){
            temp_shadow =   temp;
            temp        =   temp->next;
            free(temp_shadow);
        }
    }
    return no_error;
}

static error_t __hash_store(const char *symbol, const
uint32_t addr){
    Addr *addr_temp =   addr_table[hash_32(symbol,
HASH_SEED) & HASH_MASK];
    for(; addr_temp->next!=NULL; addr_temp=addr_temp-
>next)
        /* find the duplicate */
        if(!strcmp(addr_temp->next->symbol,symbol))
            return error_gramma_duplicate_symbol;
    /* store into the table */
    addr_temp->next     =   malloc(sizeof(Addr));
    addr_temp           =   addr_temp->next;
    strcpy(addr_temp->symbol,symbol);
    addr_temp->addr     =   addr;
    addr_temp->next     =   NULL;
    return no_error;
}
```

assembler_replacement.h

```
#ifndef ASSEMBLER_REPLACEMENT_H_INCLUDED
#define ASSEMBLER_REPLACEMENT_H_INCLUDED

#include "common.h"

/* translate all the tokens to machine code*/
/* assuming the file already passed stage 0&1&2 */
/* file structure:
|data_len(uint32_t)|data[data_len]|commands|EOF| */
error_t assembler_replacement(const char *src, const char
*dst, Error *error);


#endif // ASSEMBLER_REPLACEMENT_H_INCLUDED
```

assembler_replacement.c

```
#include "assembler.h"
#include "assembler_pretreatment.h"
#include "assembler_replacement.h"
#include "common.h"
#include "assembler.h"
#include "hash.h"
#include "common.h"
#include <string.h>
#include <stdlib.h>
#include <stdio.h>

/* get the register name according to its name */
```

```
static error_t __get_reg_value(const char *name, uint32_t
*value);

#define __return_error(error_code)  \
{strcpy(error->error_line,buffer);\
error->error_pos    =   (char)(strstr(buffer,buffer_sub)-
buffer);\
return  error_code;}

/* translate all the tokens to machine code*/
/* assuming the file already passed stage 0&1&2 */
error_t assembler_replacement(const char *src, const char
*dst, Error *error){
    /* open necessary files */
    FILE *file_src  =   fopen(src, "r");
    FILE *file_dst  =   fopen(dst, "wb");
    if(!file_src || !file_dst)
        return error_open_file;
    /* the string used for analyze */
    char buffer[MAX_BUF_LEN];
    char buffer_sub[MAX_BUF_LEN];
    /* store the formatted machine code */
    uint32_t    code;
    Op          op_temp;
    uint32_t    reg_value;

    pc_addr++;
    /* write the size of the data */
    fwrite(&pc_addr, sizeof(uint32_t), 1, file_dst);
    /* dump the predefined data into file */
    for(uint32_t i=0; i<pc_addr; i++)
```

```
        fwrite(&data_stack[i], sizeof(int16_t), 1,
file_dst);
    /* translate and write the commands */
    for(; fgets(buffer, MAX_BUF_LEN, file_src);){
        __get_nth_token(buffer, buffer_sub, 0);
        op_temp = op_hash[hash_32(buffer_sub,HASH_SEED) &
HASH_MASK];
        code    =   op_temp.value;
        /* different code format */
        switch(op_temp.format){
            case o5_p27:
                break;
            case o5_p3_a24:
                __get_nth_token(buffer, buffer_sub, 1);
                code    |=  atoi(buffer_sub);
                break;
            case o5_r3_a24:
                __get_nth_token(buffer, buffer_sub, 1);
                if(__get_reg_value(buffer_sub,
&reg_value)==error_gramma_reg_illegal)

__return_error(error_gramma_reg_illegal);
                code    |=  (reg_value<<24);

                __get_nth_token(buffer, buffer_sub, 2);
                code    |=  (uint32_t)atoi(buffer_sub);
                break;
            case o5_r3_p16_t8:

                __get_nth_token(buffer, buffer_sub, 1);
                if(__get_reg_value(buffer_sub,
```

```
&reg_value)==error_gramma_reg_illegal)

__return_error(error_gramma_reg_illegal);
                code    |=  (reg_value<<24);

                __get_nth_token(buffer, buffer_sub, 2);
                code    |=  (uint32_t)(atoi(buffer_sub)) &
0xFF;
                break;
            case o5_r3_p24:
                __get_nth_token(buffer, buffer_sub, 1);
                if(__get_reg_value(buffer_sub,
&reg_value)==error_gramma_reg_illegal)

__return_error(error_gramma_reg_illegal);
                code    |=  (reg_value<<24);
                break;
            case o5_r3_p8_i16:
                __get_nth_token(buffer, buffer_sub, 1);
                if(__get_reg_value(buffer_sub,
&reg_value)==error_gramma_reg_illegal)

__return_error(error_gramma_reg_illegal);
                code    |=  (reg_value<<24);

                __get_nth_token(buffer, buffer_sub, 2);
                code    |=  (uint32_t)(atoi(buffer_sub)) &
0xFFFF;
                break;
            case o5_r3_r4_p20:
                __get_nth_token(buffer, buffer_sub, 1);
```

```
                if(__get_reg_value(buffer_sub,
&reg_value)==error_gramma_reg_illegal)

__return_error(error_gramma_reg_illegal);
                code     |=   (reg_value<<24);

                __get_nth_token(buffer, buffer_sub, 2);
                if(__get_reg_value(buffer_sub,
&reg_value)==error_gramma_reg_illegal)

__return_error(error_gramma_reg_illegal);
                code     |=   (reg_value<<20);
                break;
            case o5_r3_r4_r4_p16:
                __get_nth_token(buffer, buffer_sub, 1);
                if(__get_reg_value(buffer_sub,
&reg_value)==error_gramma_reg_illegal)

__return_error(error_gramma_reg_illegal);
                code     |=   (reg_value<<24);

                __get_nth_token(buffer, buffer_sub, 2);
                if(__get_reg_value(buffer_sub,
&reg_value)==error_gramma_reg_illegal)

__return_error(error_gramma_reg_illegal);
                code     |=   (reg_value<<20);

                __get_nth_token(buffer, buffer_sub, 3);
                if(__get_reg_value(buffer_sub,
&reg_value)==error_gramma_reg_illegal)
```

```
__return_error(error_gramma_reg_illegal);
                code    |=  (reg_value<<16);
                break;
        }
        fwrite(&code, sizeof(uint32_t), 1, file_dst);
    }
    fclose(file_src);
    fclose(file_dst);
    return no_error;
}

static error_t __get_reg_value(const char *name, uint32_t
*value){
    if(strlen(name)!=1)
        return error_gramma_reg_illegal;
    else if('A' <=name[0] && name[0] <= 'G'){
        *value  =   (uint32_t)(name[0]-'A'+1);
        return no_error;
    }
    else if(name[0] == 'Z'){
        *value  =   0;
        return no_error;
    }
    else
        return error_gramma_reg_illegal;
}
```

hash.h

```
#ifndef MURMURHASH_H_INCLUDED
#define MURMURHASH_H_INCLUDED

#include "common.h"

/* do not change it if not necessary */
#define HASH_SEED   0
/* use the murmurhash3 algorithm */
/* you can change the hash algorithm without worrying
damaging the program */
uint32_t hash_32(const char *key, uint32_t seed);

#endif // MURMURHASH_H_INCLUDED
```

hash.c

```
#include "common.h"
#include <string.h>
#define ROT32(x, y) ((x << y) | (x >> (32 - y)))

uint32_t hash_32(const char *key, uint32_t seed){
    uint32_t len    =   strlen(key);
    static const uint32_t c1 = 0xcc9e2d51;
    static const uint32_t c2 = 0x1b873593;
    static const uint32_t r1 = 15;
    static const uint32_t r2 = 13;
    static const uint32_t m = 5;
    static const uint32_t n = 0xe6546b64;
```

```
    uint32_t hash = seed;

    const int nblocks = len / 4;
    const uint32_t *blocks = (const uint32_t *) key;
    int i;
    uint32_t k;
    for (i = 0; i < nblocks; i++) {
        k = blocks[i];
        k *= c1;
        k = ROT32(k, r1);
        k *= c2;

        hash ^= k;
        hash = ROT32(hash, r2) * m + n;
    }

    const uint8_t *tail = (const uint8_t *) (key + nblocks
* 4);
    uint32_t k1 = 0;

    switch (len & 3) {
    case 3:
        k1 ^= tail[2] << 16;
    case 2:
        k1 ^= tail[1] << 8;
    case 1:
        k1 ^= tail[0];

        k1 *= c1;
        k1 = ROT32(k1, r1);
        k1 *= c2;
```

```
        hash ^= k1;
    }

    hash ^= len;
    hash ^= (hash >> 16);
    hash *= 0x85ebca6b;
    hash ^= (hash >> 13);
    hash *= 0xc2b2ae35;
    hash ^= (hash >> 16);

    return hash;
}
```

simulator.h

```
#ifndef SIMULATOR_H_INCLUDED
#define SIMULATOR_H_INCLUDED

#include "common.h"
#include "simulator.h"
/* memory */
extern int16_t *mem;
/* Z A B C D E F G */
extern int16_t reg[8];
/* stack register */
extern uint32_t CS, DS, ES, SS;
/* program register */
extern uint32_t PC, IR;
/* program status word */
extern uint16_t PSW;
```

```
/* control on overflow flag*/
#define SETO     (PSW|=0x0001)
#define UNSETO   (PSW&=0xFFFE)
#define GETO     (PSW&0x0001)
/* control on compare flag*/
#define SETC     (PSW|=0x0002)
#define UNSETC   (PSW&=0xFFFD)
#define GETC     ((PSW&0x02)>>1)
/* the the nth reg in the command */
#define REG0     (reg[((IR & 0x07000000)>>24)])
#define REG1     (reg[((IR & 0x00F00000)>>20)])
#define REG2     (reg[((IR & 0x000F0000)>>16)])
#define ADDR     ((IR & 0x00FFFFFF))
#define IMMD     ((int16_t)(IR & 0x0000FFFF))
#define IMMD_MAX     ((int16_t)(0x7FFF))

/* judge if a number is a negative number */
#define ISNEG(reg)   (((reg)&0x80000000)>>31)

error_t simulator_init(void);

error_t simulator(const char *path, Error *error);
#endif // SIMULATOR_H_INCLUDED
```

simulator.c

```
#include "simulator.h"
#include "simulator_op_bitcalculate.h"
#include "simulator_op_control.h"
#include "simulator_op_data.h"
```

```
#include "simulator_op_logic.h"
#include "simulator_op_math.h"
#include "simulator_op_others.h"
#include "common.h"
#include <stdio.h>
#include <stdlib.h>

//============
#include "assembler.h"
extern Op op_set[32];
//============
/* memory */
int16_t *mem;
/* Z A B C D E F G */
int16_t reg[8];
/* stack register */
uint32_t CS, DS, ES, SS;
/* program register */
uint32_t PC, IR;
/* program status word */
uint16_t PSW;

/* distribute spaces and initialize the registers */
error_t simulator_init(void){
    mem =   (int16_t *)malloc(sizeof(int16_t)*(1<<24));
    if(!mem)
        return error_malloc;
    PC  =   0;
    IR  =   0;
    PSW =   0;
    return no_error;
```

```
}
/* free the spaces of the simulator*/
error_t simulator_deinit(void){
    free(mem);
    return no_error;
}
error_t simulator(const char *path, Error *error){
    error   =   error;
    FILE    *file   =   fopen(path, "rb");

    /* read the size of the data*/
    fread(&ES, sizeof(uint32_t), 1, file);
    /* set the pointer to the program part */
    fseek(file,
sizeof(int16_t)*ES+sizeof(uint32_t),SEEK_SET);
    /* read the program into CS stack and set DS */
    for(CS=0, DS=0; fread(&IR,sizeof(uint32_t),1,file);
DS+=2){
        mem[DS]     =   ((IR & 0xFFFF0000)>>16);
        mem[DS+1]   =   (IR & 0xFFFF);
    }
    fseek(file,sizeof(uint32_t),SEEK_SET);
    /* load the predefined data */
    for(uint32_t i=0; i<ES; i++)
        fread(&mem[DS+i], sizeof(int16_t), 1, file);
    ES  +=  (DS+1);
    SS  =   (1<<24)-1;
    fclose(file);

    /* run the simulator */
    for(;;PC+=2){
```

```
IR  =   ((mem[PC]<<16) | mem[PC+1]);
switch((IR & 0xF8000000)>>27){
    case 0:     HLT();      goto halt;
    case 1:     JMP();      break;
    case 2:     CJMP();     break;
    case 3:     OJMP();     break;
    case 4:     CALL();     break;
    case 5:     RET();      break;

    case 6:     PUSH();     break;
    case 7:     POP();      break;

    case 8:     LOADB();    break;
    case 9:     LOADW();    break;
    case 10:    STOREB();   break;
    case 11:    STOREW();   break;
    case 12:    LOADI();    break;
    case 13:    NOP();      break;

    case 14:    IN();       break;
    case 15:    OUT();      break;

    case 16:    ADD();      break;
    case 17:    ADDI();     break;
    case 18:    SUB();      break;
    case 19:    SUBI();     break;
    case 20:    MUL();      break;
    case 21:    DIV();      break;

    case 22:    AND();      break;
    case 23:    OR();       break;
```

```
            case 24:    NOR();      break;
            case 25:    NOTB();     break;
            case 26:    SAL();      break;
            case 27:    SAR();      break;

            case 28:    EQU();      break;
            case 29:    LT();       break;
            case 30:    LTE();      break;
            case 31:    NOTC();     break;
        }
    }
    halt:
    return no_error;
}
```

simulator_op_bitcalculate.h

```
#ifndef SIMULATOR_OP_BITCALCULATE_H_INCLUDED
#define SIMULATOR_OP_BITCALCULATE_H_INCLUDED

void AND(void);
void OR(void);
void NOR(void);
void NOTB(void);
void SAL(void);
void SAR(void);

#endif // SIMULATOR_OP_BITCALCULATE_H_INCLUDED
```

simulator_op_bitcalculate.c

```
#include"simulator.h"
#include"simulator_op_bitcalculate.h"

void AND(void){
    REG0    =   REG1 & REG2;
}
void OR(void){
    REG0    =   REG1 | REG2;
}
void NOR(void){
    REG0    =   REG1 ^ REG2;
}
void NOTB(void){
    REG0    =   ~REG1;
}
void SAL(void){
    REG0    =   REG1<<REG2;
}
void SAR(void){
    REG0    =   REG1>>REG2;
}
```

simulator_op_control.h

```
#ifndef SIMULATOR_OP_CONTROL_H_INCLUDED
#define SIMULATOR_OP_CONTROL_H_INCLUDED

#include "common.h"
```

```
void HLT(void);
void JMP(void);
void CJMP(void);
void OJMP(void);
void CALL(void);
void RET(void);

#endif // SIMULATOR_OP_CONTROL_H_INCLUDED
```

simulator_op_control.c

```
#include "simulator.h"
#include "simulator_op_control.h"

/* the implementation of control command */
inline void HLT(void){
    return;
}
inline void JMP(void){
    PC  =   (IR&0x00FFFFFF)-2;
}
inline void CJMP(void){
    PC  =   GETC?(IR&0x00FFFFFF)-2:PC;
}
inline void OJMP(void){
    PC  =   GETO?(IR&0x00FFFFFF)-2:PC;
}
inline void CALL(void){
    for(int i=0; i<8; ES++, i++)
        mem[ES] =   reg[i];
```

```
    mem[ES] =   PSW;    ES++;
    mem[ES] =   (PC&0xFFFF0000)>>16;    ES++;
    mem[ES] =   (PC&0x0000FFFF);        ES++;
    PC  =   (IR&0x00FFFFFF)-2;
}
inline void RET(void){
    ES--;   PC  =   mem[ES];
    ES--;   PC  |=  (mem[ES]<<16);
    ES--;   PSW =   mem[ES];
    for(int i=0; i<8; i++)
        ES--, reg[7-i]  =   mem[ES];
}
```

simulator_op_data.h

```
#ifndef SIMULATOR_OP_DATA_H_INCLUDED
#define SIMULATOR_OP_DATA_H_INCLUDED

void LOADB(void);
void LOADW(void);
void STOREB(void);
void STOREW(void);
void LOADI(void);
void NOP(void);

#endif // SIMULATOR_OP_DATA_H_INCLUDED
```

simulator_op_data.c

```
#include "simulator.h"
#include "simulator_op_data.h"

/* reg[7] is the 'G' register ,replace the relative addr
to absolute addr */
void LOADB(void){
    /* if it's not a set or the length of the set is 1 */
    if(mem[ADDR+DS]==1)
        REG0    =   mem[ADDR+DS+1] & 0xFF;
    /* otherwise if it's a set */
    REG0    =   mem[ADDR+reg[7]+DS+1] & 0xFF;
}
void LOADW(void){
    /* if it's not a set or the length of the set is 1 */
    if(mem[ADDR+DS]==1)
        REG0    =   mem[ADDR+DS+1];
    /* otherwise if it's a set */
    REG0    =   mem[ADDR+reg[7]+DS+1];
}
void STOREB(void){
    if(mem[ADDR+DS]==1)
        mem[ADDR+DS+1]  =   REG0 & 0xFF;
    mem[ADDR+reg[7]+DS+1]   =   REG0 & 0xFF;
}
void STOREW(void){
    if(mem[ADDR+DS]==1)
        mem[ADDR+DS+1]  =   REG0;
    mem[ADDR+reg[7]+DS+1]   =   REG0;
}
void LOADI(void){
```

```
    REG0     =    IMMD&0xFF;
}
void NOP(void){
    return;
}
```

simulator_op_logic.h

```
#ifndef SIMULATOR_OP_LOGIC_H_INCLUDED
#define SIMULATOR_OP_LOGIC_H_INCLUDED


void EQU(void);
void LT(void);
void LTE(void);
void NOTC(void);

#endif // SIMULATOR_OP_LOGIC_H_INCLUDED
```

simulator_op_logic.c

```
#include "simulator.h"
#include "simulator_op_logic.h"

void EQU(void){
    (REG0 == REG1)?SETC:UNSETC;
}
void LT(void){
    (REG0 < REG1)?SETC:UNSETC;
```

```
}
void LTE(void){
    (REG0 <= REG1)?SETC:UNSETC;
}
void NOTC(void){
    GETC?UNSETC:SETC;
}
```

simulator_op_math.h

```
#ifndef SIMULATOR_OP_MATH_H_INCLUDED
#define SIMULATOR_OP_MATH_H_INCLUDED


#include "common.h"

void ADD(void);
void ADDI(void);
void SUB(void);
void SUBI(void);
void MUL(void);
void DIV(void);


#endif // SIMULATOR_OP_MATH_H_INCLUDED
```

simulator_op_math.c

```
#include "simulator.h"
#include "simulator_op_math.h"

inline void ADD(void){
    UNSETO;
    REG0    =   REG1+REG2;
    /* if the sign of two number is the same and overflows
*/
    if( !(ISNEG(REG1)^ISNEG(REG2)) &&
(ISNEG(REG0)^ISNEG(REG1)) )
        SETO;
}
inline void ADDI(void){
    UNSETO;
    uint16_t temp   =   REG0;
    REG0    +=  IMMD;
    if(ISNEG(REG0)^ISNEG(temp))
        SETO;
}
inline void SUB(void){
    UNSETO;
    REG0    =   REG1-REG2;
    /* if the sign of the two number is different and
overflows */
    if((ISNEG(REG1)^ISNEG(REG2) &&
(ISNEG(REG0)^ISNEG(REG1))) )
        SETO;
}
inline void SUBI(void){
    UNSETO;
```

```
    uint16_t temp    =    REG0;
    REG0     -=   IMMD;
    if(ISNEG(REG0)^ISNEG((-temp)))
        SETO;
}
inline void MUL(void){
    UNSETO;
    REG0     =    REG1*REG2;
    if(REG1!=0 && REG0/REG1 != REG2)
        SETO;
}
inline void DIV(void){
    UNSETO;
    if(REG2==0){
        SETO;
        return;
    }
    REG0     =    REG1/REG2;
}
```

simulator_op_others.h

```
#ifndef SIMULATOR_OP_OTHERS_H_INCLUDED
#define SIMULATOR_OP_OTHERS_H_INCLUDED

void PUSH(void);
void POP(void);
void IN(void);
void OUT(void);
```

```
#endif // SIMULATOR_OP_OTHERS_H_INCLUDED
```

simulator_op_others.c

```
#include "simulator.h"
#include "simulator_op_others.h"
#include <stdio.h>

void PUSH(void){
    mem[SS--]    =    REG0;
}
void POP(void){
    REG0     =    mem[++SS];
}
void IN(void){
    REG0     =    getchar();
}
void OUT(void){
    putchar(REG0);
}
```

用于测试的汇编程序如下：

queen.txt

```
            # 八皇后问题

            WORD      cnt = 0                    # 解计数器
            BYTE      sltn[8] = {0,0,0,0,0,0,0,0}    # 存放解的
数组，元素值依次为各行上皇后的位置
            BYTE      cell[64]                   # 元素值表示对应
```

单元位置受皇后攻击状况

```
            # 将数组cell的元素值初始化为 0
            LOADI     A         64              # 设置数组下标的
上界为 64
            LOADI     G         0               # 数组下标初始化
为 0
init:       STOREB    Z         cell            # 将寄存器Z的值存
入cell[G]
            ADDI      G         1               # 下标增加 1
            LT        G         A               # 关系运算 G <
64
            CJMP      init                      # 比较结果为真则
转至标号init，否则往下执行

            LOADI     B         0               # 将行号 0 存入寄
存器B中
            PUSH      B                         # 将B值压入堆栈
            CALL      dfs                       # 调用子程序dfs

            HLT                                 # 终止程序运行

            # 深度优先搜索算法，采用递归实现
dfs:        POP       B                         # 从堆栈中取出行
号值存入寄存器B
            LOADI     C         8               # 将行号的上界值
8 存入寄存器C，行号取值范围 0-7
            LT        B         C               # 关系运算 B <
C，比较行号是否越界
            CJMP      next                      # 没有越界则转至
标号next，否则往下执行
            CALL      prnt                      # 行号越界表明得
到了一个解，调用子程序prnt，输出解
            RET                                 # 控制返回子程序
被调用处

next:       LOADI     A         0               # 将 0 存入寄存器
A，从B行的第 0 列开始测试
```

```
n1:         MUL     D       B       C       # 计算 D = B * C，计算第B行元素的起始下标
            ADD     G       A       D       # 计算 G = A + D，得到第B行第A列的元素下标值
            LOADB   D       cell            # 将cell[G]取出，存入寄存器D
            EQU     D       Z               # 关系运算 D == 0(Z)，为真表明第B行第A列没有受到皇后攻击
            NOTC                            # 将比较标志位的值反转
            CJMP    n2                      # 此时比较标志为真表示D不等于 0，转至标号n2，否则可在此处放置皇后
            PUSH    A                       # 将寄存器A的值压入堆栈
            PUSH    B                       # 将寄存器B的值压入堆栈
            CALL    tag                     # 调用子程序tag在B行A列放置皇，并在皇后所能攻击到的位置上做标记
            ADD     D       B       Z       # 将寄存器B的值存入寄存器D  D = B + Z
            ADDI    D       1               # 将寄存器D中的值增加 1  D = D + 1，得到下一行行号
            PUSH    D                       # 将下一行行号D压入堆栈
            CALL    dfs                     # 递归调用，在下一行合适的位置上放置皇后
            PUSH    A                       # 将列号A压入堆栈
            PUSH    B                       # 将行号B压入堆栈
            CALL    tag                     # 再次调用tag，抹去标记

n2:         ADDI    A       1               # 列号增加 1
            LT      A       C               # 关系运算 列号 < 8
            CJMP    n1                      # 为真则转至标号n1，继续测试新一列

            RET                             # 否则返回子程序
```

被调用处

```
            # 输出解
prnt:       LOADI    G        0                 # 下标置为 0
            LOADW    C        cnt               # 将解计数器的值
加载到寄存器C中
            ADDI     C        1                 # 计数器的值增加
1
            STOREW   C        cnt               # 存入计数器
            LOADI    E        1                 # 将 1 存入寄存器
E，用作按位与运算的屏蔽码 0x1
            AND      D        C        E        # 按位与运算，将C
的最低位取出，存入寄存器D中
            PUSH     D                          # D中值为 1，表明
解的个数为奇数，为 0 则是偶数，先将D入栈
            LOADI    D        10                # D = 10，十进制
进位值
loop1:      DIV      E        C        D        # E = C / D，E
中为C除以 10 的商，整数除，切掉C的个位数字
            MUL      F        D        E        # F = D * E
            SUB      F        C        F        # F = C - F，得
到C除以 10 的余数
            PUSH     F                          # 将余数F入栈
            ADDI     G        1                 # G用来为余数压栈
次数进行计数
            ADD      C        E        Z        # 将商E存入C
            LT       Z        C                 # 关系运算 Z < C
            CJMP     loop1                      # 为真表示商大于
0，转至loop1，继续求余数
loop2:      POP      C                          # 余数出栈，存入C
            ADDI     C        48                # C = C + 48，
将数字转为数字字符
            OUT      C        15                # 输出C中的数字字
符
            SUBI     G        1                 # 余数计数器减 1
            LT       Z        G                 # 关系运算符 Z <
G
            CJMP     loop2                      # 为真表明入栈的
```

```
余数还未取完，转loop2 继续取余数并转换为数字字符输出
            LOADI    C        58          # 将字符 ':' 存
入寄存器C
            OUT      C        15          # 输出字符 ':'
            LOADI    C        10          # 将换行符 '\n'
存入寄存器C
            OUT      C        15          # 输出换行符

            # 依次输出八行上皇后的位置
            LOADI    A        8           # 将数组下标上限
8 存入寄存器A
            LOADI    G        0           # 将下标置为 0
loop3:      LOADB    C        sltn        # C = sltn[G]，
每个元素的值表示元素下标对应的行上皇后放置的位置
            PUSH     C                    # C入栈，C值表示
皇后在第G行上的位置，即列号
            CALL     aline                # 调用子程序
aline 输出第G行的摆子情况
            ADDI     G        1           # G = G + 1，下
标增加 1
            LT       G        A           # 关系运算 G <
A，判断下标是否越界
            CJMP     loop3                # 为真，则转到标
号loop3 输出下一行，否则继续执行
            LOADI    C        10          # 将换行符 '\n'
存入C
            OUT      C        15          # 输出换行符

            POP      D                    # 解个数的最低位
出栈，存入D
            LT       Z        D           # 关系运算 0(Z)
< D
            CJMP     loop4                # 为真，表明解的
个数是奇数，转至loop4，否则执行下一行
            IN       D        0           # 等待从键盘输入
一个字符。这样实现了每输出两个解程序停顿一下的效果

loop4:      RET                           # 返回
```

```
            # 输出棋盘一行的摆子情况
aline:      POP     B                       # 皇后所在的列号出栈到B中
            LOADI   A       8               # A = 8
            LOADI   D       0               # D = 0，列号初始化为 0
            LOADI   E       32              # E = 32，空格字符
a1:         LOADI   C       49              # C = 49，数字字符 1
            EQU     B       D               # 关系运算 B == D
            CJMP    a2                      # 为真，表明D列上放置皇后，转至a2，否则执行下行
            SUBI    C       1               # C = C - 1，得到数字字符 0
a2:         OUT     C       15              # 将C中的数字字符输出
            OUT     E       15              # 输出一个空格字符
            ADDI    D       1               # D = D + 1，列号增加 1
            LT      D       A               # 关系运算 D < A
            CJMP    a1                      # 为真，表明此行还没有全部输出，转至a1 继续输出
            LOADI   C       10              # 将换行符 \n 存入C
            OUT     C       15              # 输出一个换行符
            RET                             # 返回

            # 在第B行A列上放置皇后，在后面各行皇后能够攻击到的单元位置上做标记
tag:        POP     B                       # 行号出栈到B
            POP     A                       # 列号出栈到A
            ADD     G       B       Z       # G = B( + Z)，即将行号存入G，作为数组下标
            STOREB  A       sltn            # 将列号存入解数
```

组元素sltn[G]，表示第B行A列放置皇后

```
            LOADI    C        8                 # C = 8
            LOADI    D        0                 # D = 0
t1:         ADDI     D        1                 # D = D + 1
            ADD      E        B        D        # E = B + D，B
行后的第D行行号存入E
            LT       E        C                 # 关系运算 E <
C，行号小于 8
            CJMP     t2                         # 为真则转到标号
t2
            RET                                 # 否则返回

            # 在E行A列单元上用行号B做标记
t2:         PUSH     A                          # A入栈
            PUSH     B                          # B入栈
            PUSH     E                          # 需做标记的行E入
栈
            CALL     mark                       # 调用子程序
mark，在第E行上做标记

            # 在E行A-D列单元上用行号B做标记
            SUB      F        A        D        # F = A - D
            LT       F        Z                 # 关系运算 F < 0
            CJMP     t3                         # 列号小于 0，转
至t3
            PUSH     F
            PUSH     B
            PUSH     E
            CALL     mark

            # 在E行A+D列单元上用行号B做标记
t3:         ADD      F        A        D        # F = A + D
            LTE      C        F                 # 关系运算 8 <=
F
            CJMP     t1
            PUSH     F
            PUSH     B
            PUSH     E
```

```
        CALL    mark
        JMP     t1

        # 在C行A列单元上用B做标记
mark:   POP     C
        POP     B
        POP     A
        LOADI   D       8               # D = 8
        MUL     E       C       D       # E = C * D
        ADD     G       A       E       # G = A + E,G
为C行A列单元数组元素下标
        LOADB   E       cell            # E = cell[G]
        LOADI   F       1               # F = 1
        SAL     B       F       B       # B = F << B
        NOR     E       E       B       # E = E ^ B
        STOREB  E       cell            # 将E存入cell[G]
        RET                             # 返回
```

convert.txt

```
        LOADI C     10
        LOADI D     0
input:  IN    B     0         #输入
        EQU   C     B
        SUBI  B     48
        CJMP  main
        MUL   D     D     C
        ADD   D     D     B
        JMP input

main:   ADD   A     D     Z
        LOADI B     16        #进制
        LOADI F     0         #计数
        LOADI G     10        #输出

loop:   DIV   C     A     B   #C除法结果
```

```
        MUL   D    C    B    #D中间
        SUB   E    A    D    #E余数

        ADDI  F    1
        PUSH  E
        ADD   A    C    Z

        EQU   C    Z
        CJMP  print
        JMP   loop

print:  POP   A
        LT    A    G
        CJMP  num
        JMP   chr
cont:   SUBI  F    1
        LTE   F    Z
        CJMP  end
        JMP   print

num:    ADDI  A    48
        OUT   A    15
        JMP   cont

chr:    ADDI  A    55
        OUT   A    15
        JMP   cont

end:    LOADI   A   10
    OUT A   15
    HLT
```

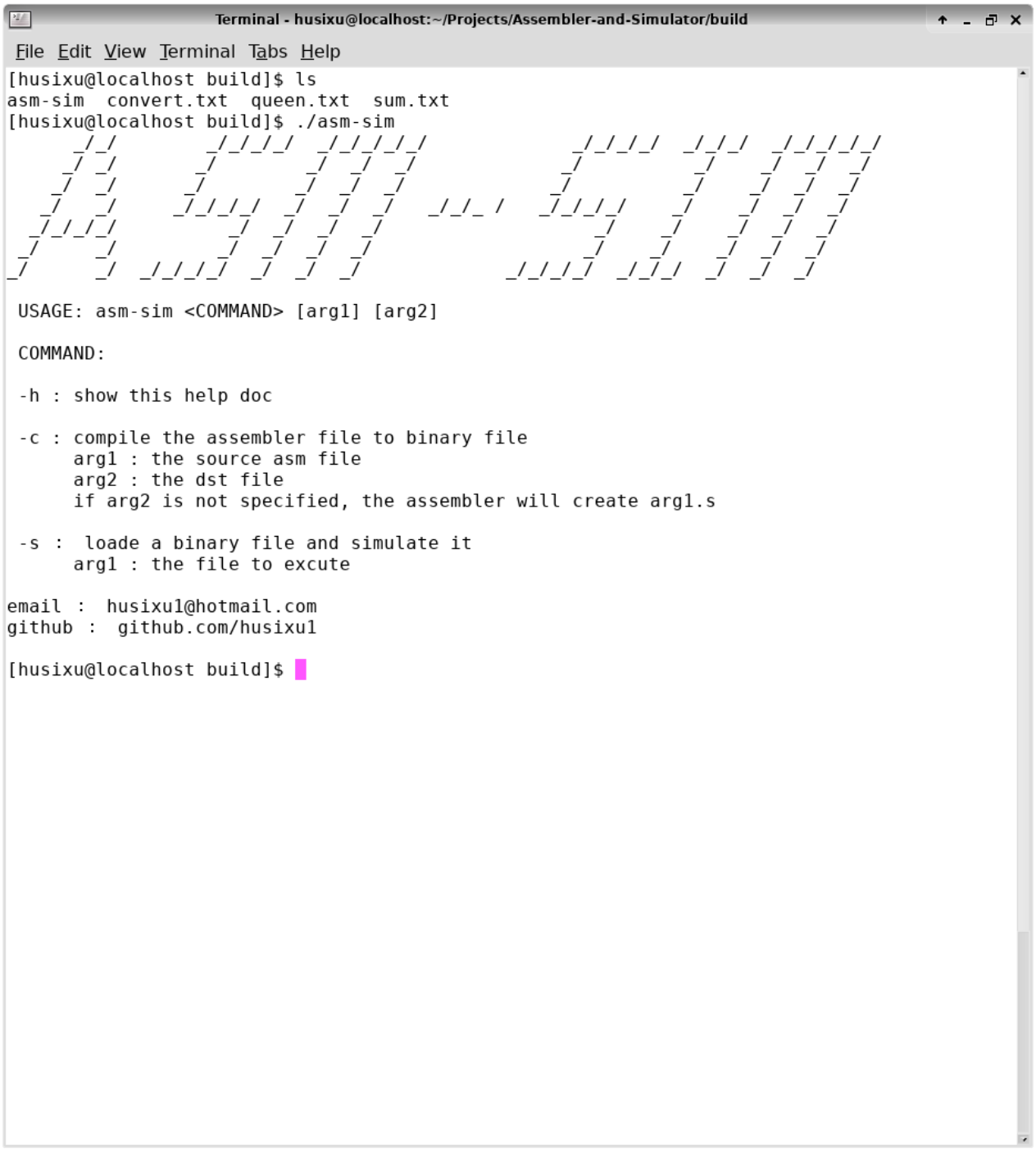
sum.txt

```
        BYTE    num = 100

        LOADI   A   0       #记录和
        LOADB   B   num     #记录n
        LOADI   C   1       #记录当前数
        LOADI   D   1       #自增步长
        LOADI   E   10
        LOADI   F   0

loop:   LT      B   C
        CJMP    print
        ADD     A   A   C
        ADDI    C   1
        JMP     loop

print:  DIV     C   A   E
        MUL     C   C   E
        SUB     D   A   C
        ADDI    D   48
        PUSH    D
        ADDI    F   1
        DIV     A   A   E
        EQU     A   Z
        CJMP    print2
        JMP     print
print2: POP     A
        SUBI    F   1
        OUT     A   15
        EQU     F   Z
        CJMP    end
        JMP     print2
end:    LOADI   A   10
    OUT A   15
    HLT
```

六、运行测试与结果分析

主要测试程序对汇编程序的编译、运行情况。

一、 直接运行程序，弹出帮助文档。



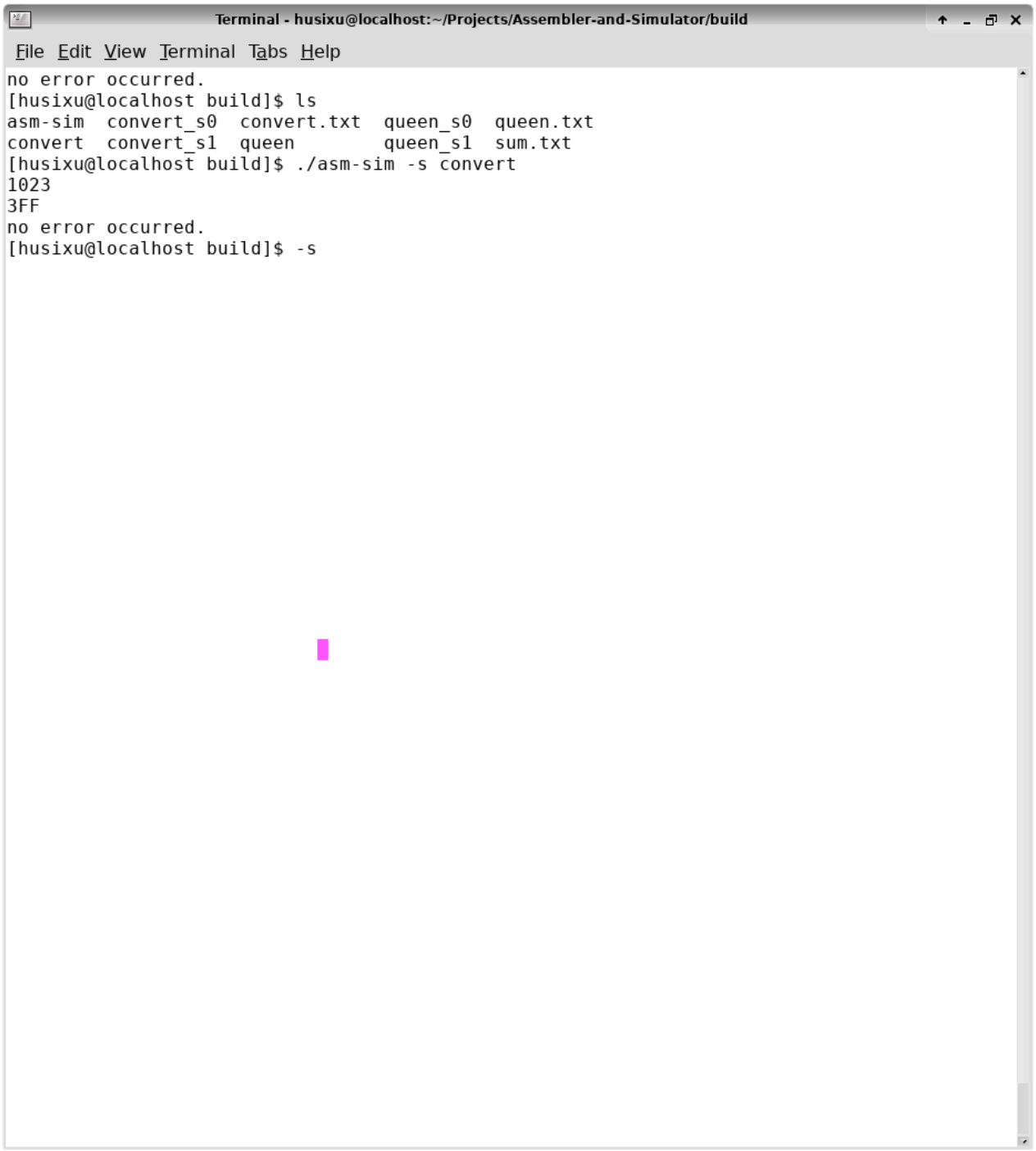
二、 -c 参数运行编译器，-s 参数运行模拟器，便以模拟 queen.txt(由于输出结果过长，不予完全显示)。

Terminal - husixu@localhost:~/Projects/Assembler-and-Simulator/build

File Edit View Terminal Tabs Help

```
[husixu@localhost build]$ ./asm-sim -c queen.txt queen
no error occurred.
[husixu@localhost build]$ ls
asm-sim  convert.txt  queen  queen_s0  queen_s1  queen.txt  sum.txt
[husixu@localhost build]$ ./asm-sim -s queen
1:
1 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0
0 0 0 0 0 0 1 0
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0

2:
1 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 1
0 0 1 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 1 0 0 0 0
0 1 0 0 0 0 0 0
0 0 0 0 1 0 0 0


3:
1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 0 0 0 1 0 0 0
0 0 1 0 0 0 0 0

4:
1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0
```

三、 编译运行 convert.txt



四、 编译运行 sum.txt，不输入输出文件名则会自动在输入文件结尾添加.s 作为输出文件名。

七、总结

通过这次实验，我对编译原理和汇编程序的运行原理有了总体上的认识和了解，了解并实现了预处理的基本方法。最终得到的编译器也能够正常编译和模拟给定的指令集编写的任意合法指令，并且实现的时间复杂度非常理想。

不足的地方是对于汇编程序声明的变量内存空间的分配没有达到最高效率，其实可以用单独的一张表存储变量的类型（数组/单变量），以达到最大空间利用率；二进制文件的分段读取也是可以考虑的内容之一，如果文件过大，采用分段读取的方式会省空间。

八、参考文献

1、 《C 语言程序与设计》，曹计昌 卢平 李开，电子工业出版社

2、 《汇编语言》，王爽，清华大学出版社

九、指导教师评语