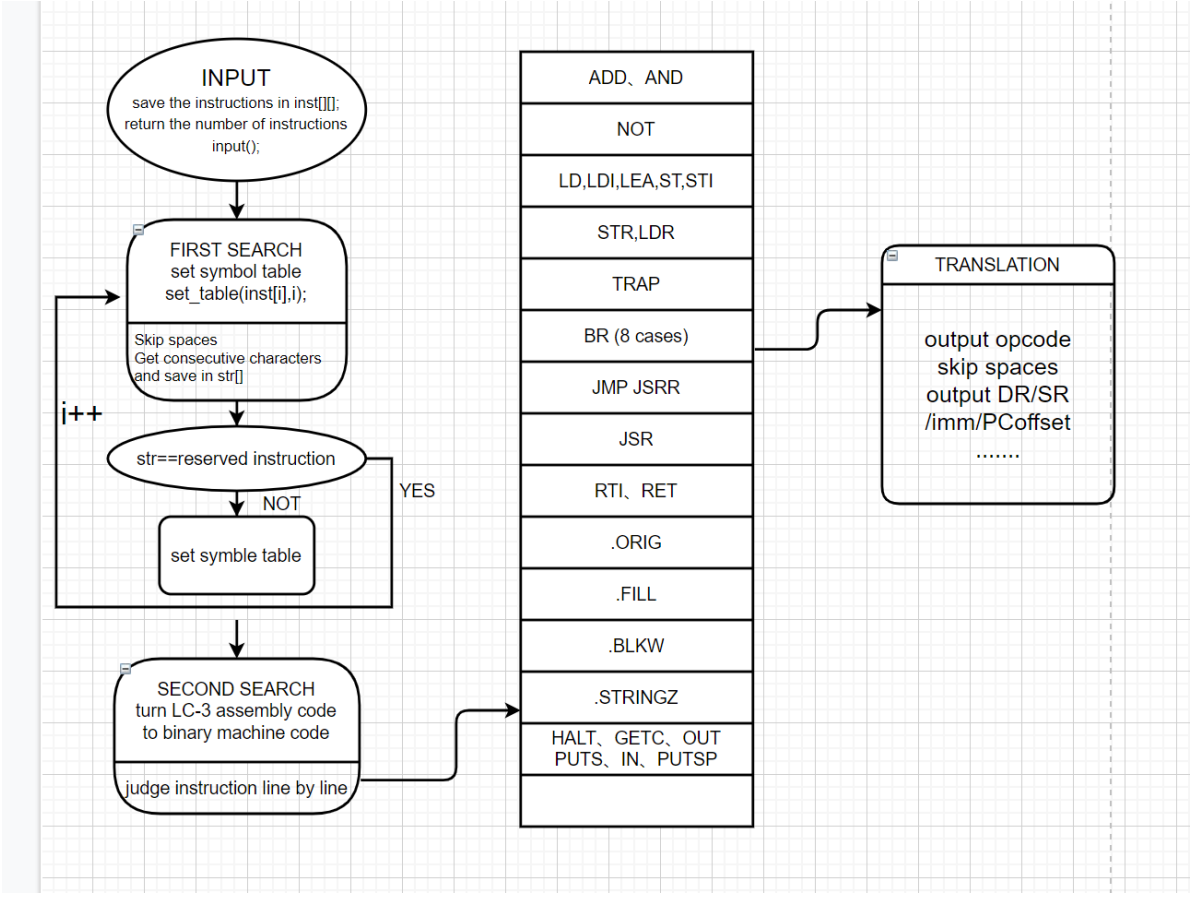
1.Algorithm



2.Essential parts of my codes

```
//输入指令，删除空行，保存在inst[][]中，返回行数
int input()
{
    int i=0;
    int j=0;
    char ch;
    while(gets(inst[i])!=NULL){
        ch=*inst[i];
        j=0;
        while(ch==32){      //跳过空格
        j++;
        ch=*(inst[i]+j);
        }
        if(ch==0) continue; //跳过空行，或者是一行读取结束
        i++;
    }
    return i;
}


int judge(char*ptr,int i){    //判断读入指令类型
    char ch;
    char str[20];           //保存opcode或者label
    int j=0;
    int k=0;
    int p=0;
```

```
    int flag;               //若flag为0，str内为保留命令
    ch=*ptr;
    while(ch==32) {         //跳过空格
        j++;
        ch=*(ptr+j);
    }
    while(ch!=32&&ch!=0){   //获取最前面的连续字符，并存储在str[]中
        str[k]=ch;
        k++;
        j++;
        ch=*(ptr+j);
    }
    str[k]='\0';
    while(k){               //判断是否为35种保留命令之一，若是，返回该保留字对应的序号
        for(p=0;p<35;p++){
            flag=strcmp(str,saved_sy[p]);
            if(flag==0) return p;
        }
        k=0;                //出现标签时，获取标签后真正的保留命令
            while(ch==32) {
                j++;
                ch=*(ptr+j);
            }
            while(ch!=32&&ch!=0){
                str[k]=ch;
                k++;
                j++;
                ch=*(ptr+j);
            }
            str[k]='\0';
                for(p=0;p<35;p++){      //判断是哪一种保留命令，并返回对应序号
                    flag=strcmp(str,saved_sy[p]);
                    if(flag==0) return p;
                }
        return -1;      //若不包含任何保留命令，返回-1
    }
}


void set_table(char*ptr,int i)      //建立symbol_table
{
    char ch;
    int j=0;
    int k=0;
    int p=0;
    int q=0;
    int flag;
    int match=0;
    char str[20];
    ch=*ptr;

    inst_add[i]=inst_add[i-1]+count+1;  //更新当前命令的存储地址
    count=0;                            //count清零，重新计算当前命令占据的内存
    //计算count
    while(ch==32) {         //跳过空格
        j++;
        ch=*(ptr+j);    }
    while(ch!=32&&ch!=0){   //获取最前面的连续字符
```

```
        str[k]=ch;
        k++;
        j++;
        ch=*(ptr+j);
    }
    str[k]='\0';
    q=j;
        for(p=0;p<35;p++){  //判断指令类型，与35种保留字进行比较，并判断有无label
            flag=strcmp(str,saved_sy[p]);
            if(flag==0){    //没有label
            if(p==26){      //当前指令为.BLKW时，count为#后的数
            j=0;
            while(inst[i][j]!='#')j++;
            j++;
            while(inst[i][j]==32) j++;
            while(inst[i][j]!=32&&inst[i][j]!=0){
                    count=count*10+inst[i][j]-'0';
                    j++;
            }
            count--;        //每次指令地址会增加一，count-- 抵消自增的1
            }
            else if(p==27){ //当前指令为.SRTINGZ ，count为“”内字符个数加1
                j=0;
                while(inst[i][j]!=34) j++;
                j++;
                while(inst[i][j]==32) j++;
                while(inst[i][j]!=34){
                    count++;
                    j++;
                }
            }
            match++;        //若match>0，当前指令中没有出现label
            break;
            }
        }

        if(match==0){       //若match=0,当前指令中出现LABEL
            memcpy(sym_t[table_line].label,str,sizeof(char)*20); //建立
symbol_table
            sym_t[table_line].addr=inst_add[i];
            table_line++;
            //再次读取连续字符，判断保留命令类型，用于计算count
            k=0;
            while(ch==32){
                q++;
                ch=*(ptr+q);
            }
            while(ch!=32&&ch!=0){
                str[k]=ch;
                k++;
                q++;
                ch=*(ptr+q);
            }
            str[k]='\0';

            for(p=0;p<35;p++){  //与35种保留命令进行比较
            flag=strcmp(str,saved_sy[p]);
            if(flag==0){
```

```
            if(p==26){          //当前指令为.BLKW时，count为#后的数
            j=0;
            while(inst[i][j]!='#')j++;
            j++;
            while(inst[i][j]==32) j++;
            while(inst[i][j]!=32&&inst[i][j]!=0){
                    count=count*10+inst[i][j]-'0';
                    j++;
                }
            count--;
            }
            else if(p==27){     //当前指令为.SRTINGZ ，count为“”内字符个数加1
                j=0;
                while(inst[i][j]!=34) j++;
                j++;
                while(inst[i][j]==32) j++;
                while(inst[i][j]!=34){
                    count++;
                    j++;
                }
                }
            }
        }
        }
}


void out_binary(int a,int length){       //整数a转换成二进制以length长度输出
    int len1=0;                          //a为正数时，返回buffer的长度
    int i;
    char buffer[16];                     //保存a的16-bit two's complement integer
    myitoa (a,buffer);
    if(a>=0){                           //若a为正数，限制长度输出，前面补0
        len1=strlen(buffer);
        for(i=0;i<length-len1;i++){
            printf("0");
        }
        printf ("%s",buffer);
    }
    if(a<0){                            //若a为负数，输出后length位
        for(i=length;i>0;i--)
        printf("%c",buffer[16-i]);
    }
}

char *myitoa(int num,char*str)    //把整数a转换成字符串，并返回指向转换后的字符串的指针
{
    char index[]="01";
    unsigned unum;               //中间变量
    int i=0,j,k;
    unum=(unsigned short)num;     //把num强制类型转换成16位无符号整数型
    do {
        str[i++]=index[unum%2];  //十进制转换成二进制，且用字符表示存储
        unum/=2;
    }while(unum);
    str[i]='\0';
    char temp;
    for(j=0;j<=(i-1)/2.0;j++)     //调整数组元素顺序为输出顺序
```

```
    {
        temp=str[j];
        str[j]=str[i-j-1];
        str[i-j-1]=temp;
    }
    return str;
}
```