COMPILE CORE: from C to ASSEMBLY

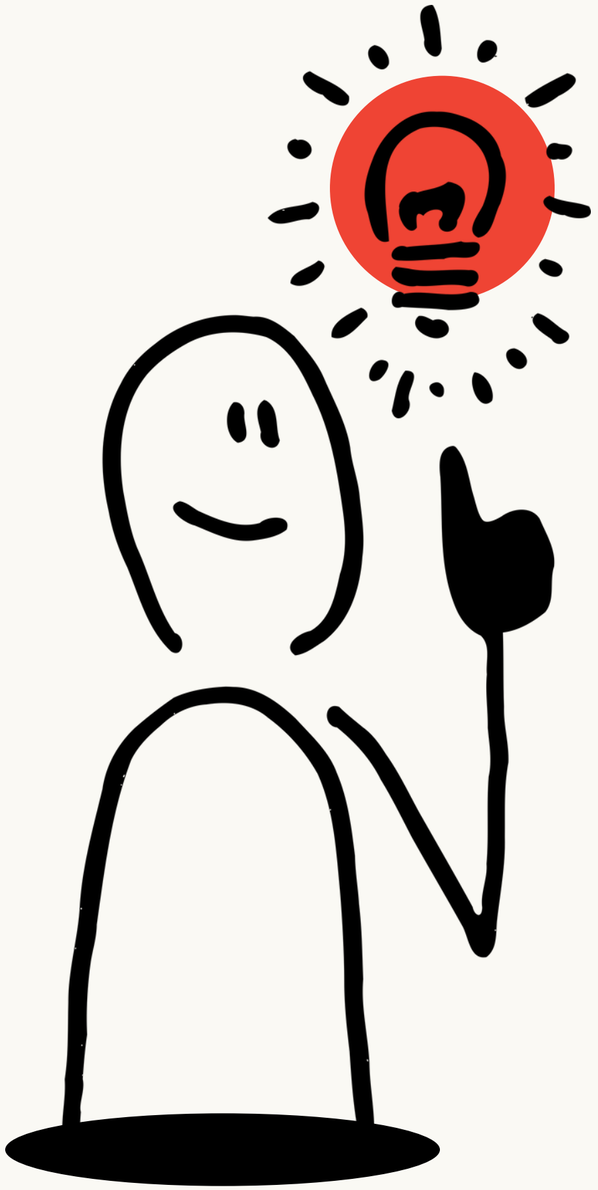
Presented by
Caine Xu

Sustainable Business

Understanding Green Jobs

TYNK UNLIMITED

Unlocking Opportunities for Environmental Stewardship and Economic Growth



CompileCore
旨在构建一个教学友好、架构清晰、全流程可视化的C语言编译系统。
将复杂的编译原理转化为可实践、可调试的学习工具，帮助学生深入理解从高级语言到机器指令的完整转换过程。

.

CORE TECHNICAL PROCESS

C语言源代码输入

词法分析（单词序列）

基于有限自动机原理，识别33个关键字、20+运算符、界符，生成符号表

语法分析（语法树）

采用LL(1)自上而下分析法，设计完整的C语言子集文法规则,精心设计Select集合，确保语法分析的准确性和无二义性

语义分析（四元式）

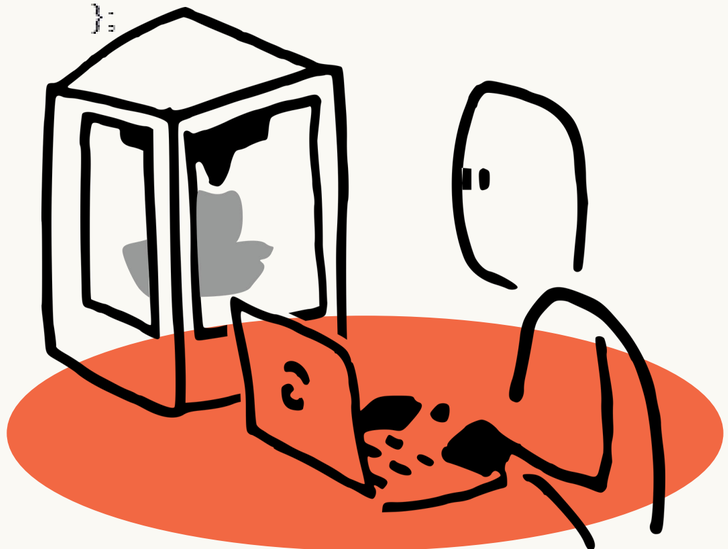
生成四元式中间代码，实现符号表管理和类型检查,采用临时变量复用策略，减少中间代码冗余

目标代码生成（x86汇编）

将四元式转换为x86汇编指令，支持条件分支、循环、算术运算,实现MOV、ADD、SUB、MUL、CMP、JMP等核心x86指令的映射

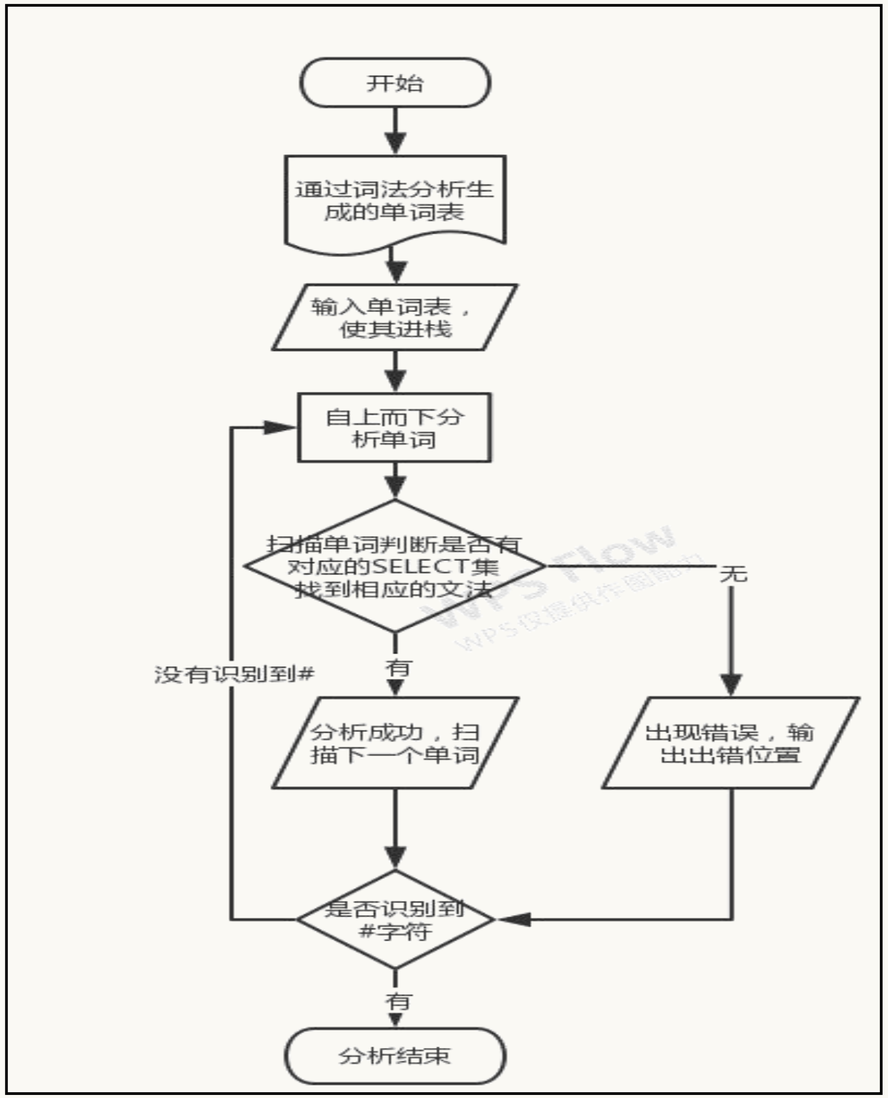
Crucial Code

```
struct mpl
{
    char str[20] = { 0 };//取到的字符
    int num = 0;//字符的标号
    int row = 0;//程序中所在行数
    int numw = 0;//维度
    char len[20] = { 0 };//一个结束符
};
```



语法分析

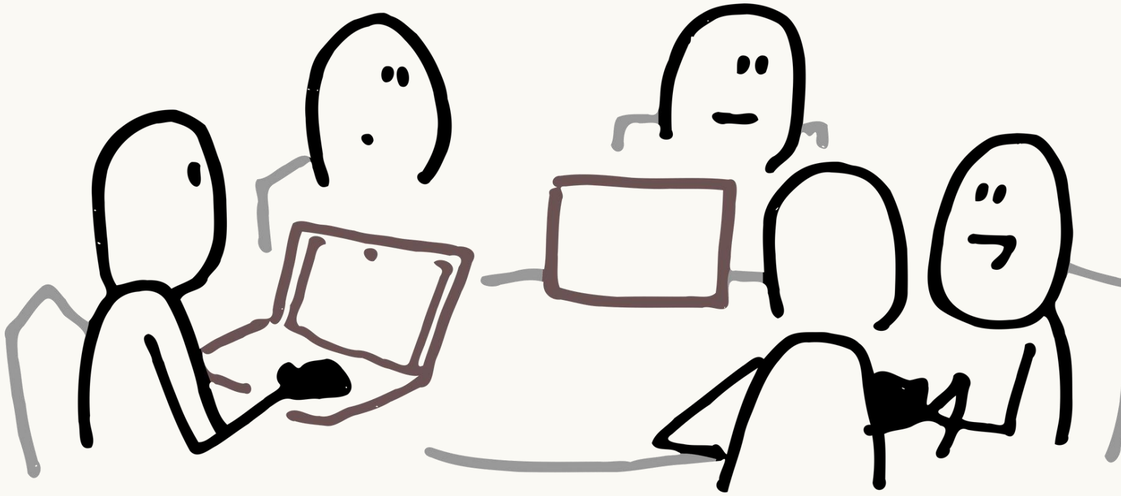
```
struct mpl array[max];              //定义结构体数组
int i = 0;//与 array 数组关联

struct symtable
{
    char symname[20] = { 0 };//标识符
    int type = -1;//int26:1 int:2 char:3 float:4 double:5
    int value = -1; //-1 表示没有赋值。1 表示有赋值
};
struct symtable sym[max];//符号表数组
int symt = 0;//与 sym 数组(符号表)关联
```



语义分析

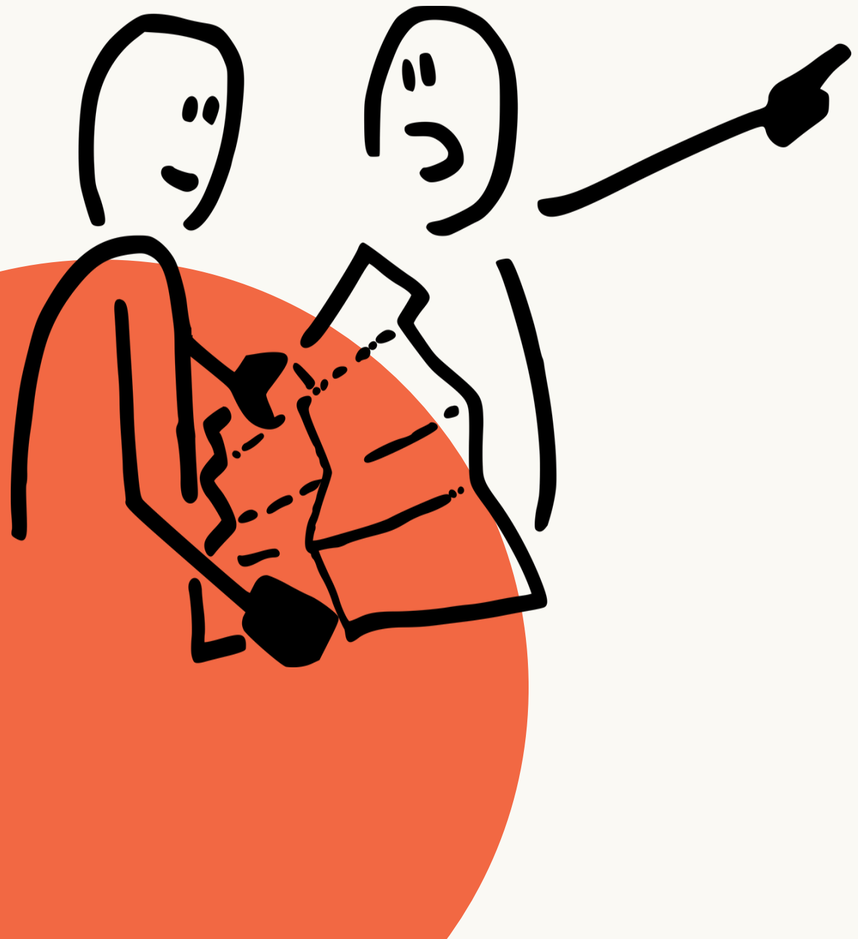
```
char result[max][10] = { "t1","t2","t3","t4","t5","t6","t7","t8","t9","t10" };//临时变量数组
int r = 0;//与 result 关联

int serial = 1;//四元式序号

char op[5], v1[10], v2[10], res[10];//四元式各项临时变量

struct Quaternary plus[10];//for 循环第三表达式,遇到}时才输出
int plust = 0;
int plustt = 0;
int plusflag = 0;//表示 for 循环第三表达式是++、--模式

int trueout[10] = { 0 };//回填时候的真出口
int truet = 0;
char trueouttemp[5];
int fakeout[10] = { 0 };//回填时候的假出口
int faket = 0;
char fakeouttemp[5];
```

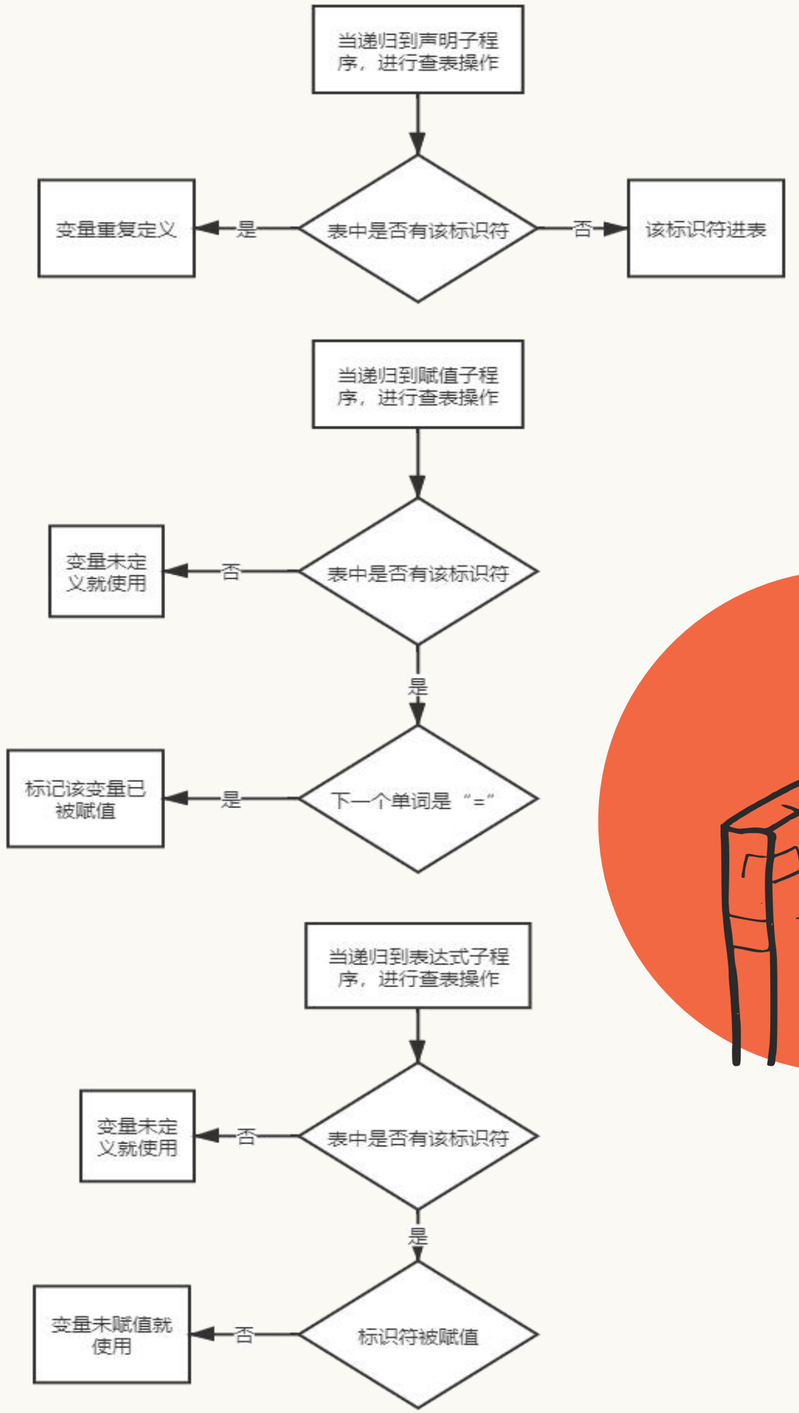
```
int elseout = 0;//取条件跳转的真出口：if 成立时的出口
char elseouttemp[5] = { 0 };

char rech[max];//存储修改后的程序
int p = 0;//与 rech 关联

char type[20] = { 0 };//保存赋值语句时标识符的类型
```



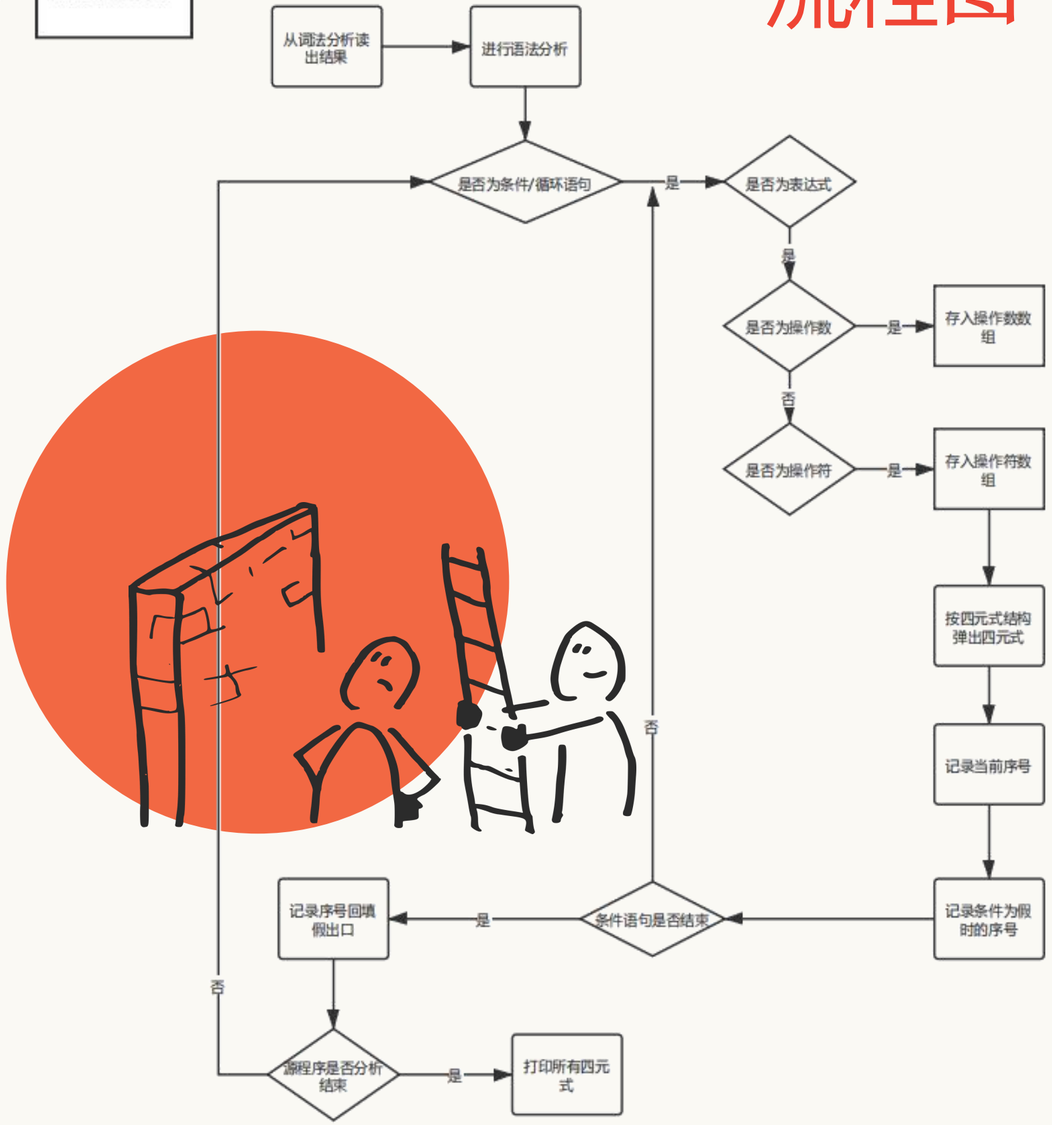
Flowchart

语义分析流程图

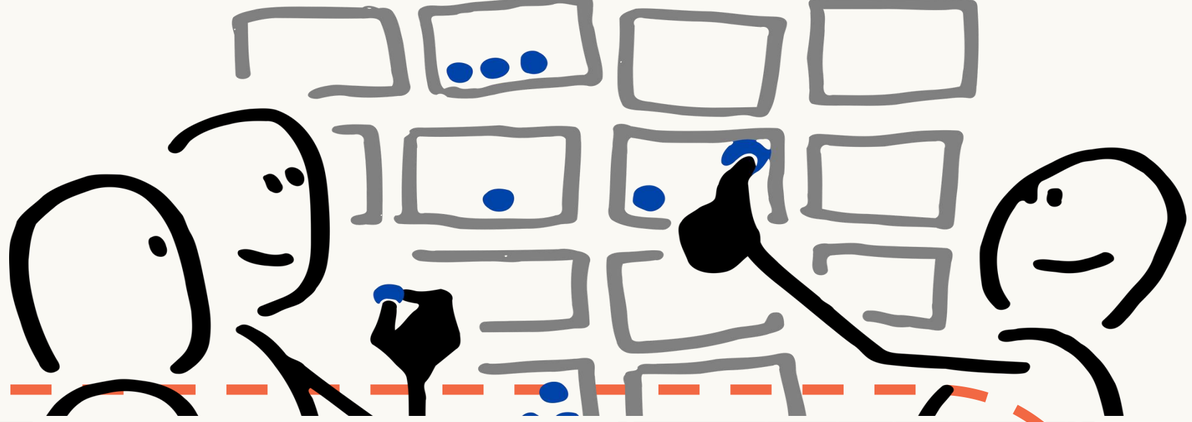




生成四元式流程图



Final program



C语言代码

```
重构的程序为:
void main()
{
 int b;
 int c;
 int a;
 a=0;
 c=1;
 if(c==1)
 {
  b=1;
 }
 else
 {
  b=10;
 }
 while(c<5)
 {
  c=c+1;
 }
 for(a<c;a++)
 {
  c=c-1;
 }
#
```

词法分析

```
词法分析程序结果为:
[    void , 32 , 1 , 0 ,    $]
[    main , 33 , 1 , 0 ,    $]
[       ( ,  3 , 1 , 0 ,    $]
[       ) ,  4 , 1 , 0 ,    $]
[       { ,  1 , 2 , 0 ,    $]
[     int , 32 , 3 , 0 ,    $]
[       b , 33 , 3 , 0 ,    $]
[       ; ,  5 , 3 , 0 ,    $]
[     int , 32 , 4 , 0 ,    $]
[       c , 33 , 4 , 0 ,    $]
[       ; ,  5 , 4 , 0 ,    $]
[     int , 32 , 5 , 0 ,    $]
[       a , 33 , 5 , 0 ,    $]
[       ; ,  5 , 5 , 0 ,    $]
[       a , 33 , 6 , 0 ,    $]
[       = , 22 , 6 , 0 ,    $]
[       0 , 34 , 6 , 0 ,    $]
[       ; ,  5 , 6 , 0 ,    $]
[       c , 33 , 7 , 0 ,    $]
[       = , 22 , 7 , 0 ,    $]
[       1 , 34 , 7 , 0 ,    $]
[       ; ,  5 , 7 , 0 ,    $]
[      if , 32 , 8 , 0 ,    $]
[       ( ,  3 , 8 , 0 ,    $]
[       c , 33 , 8 , 0 ,    $]
[      == , 23 , 8 , 0 ,    $]
[       1 , 34 , 8 , 0 ,    $]
[       ) ,  4 , 8 , 0 ,    $]
[       { ,  1 , 9 , 0 ,    $]
[       b , 33 ,10 , 0 ,    $]
[       = , 22 ,10 , 0 ,    $]
[       1 , 34 ,10 , 0 ,    $]
[       ; ,  5 ,10 , 0 ,    $]
[       } ,  2 ,11 , 0 ,    $]
[    else , 32 ,12 , 0 ,    $]
[       { ,  1 ,13 , 0 ,    $]
[       b , 33 ,14 , 0 ,    $]
[       = , 22 ,14 , 0 ,    $]
[      10 , 34 ,14 , 0 ,    $]
[       ; ,  5 ,14 , 0 ,    $]
[       } ,  2 ,15 , 0 ,    $]
[   while , 32 ,16 , 0 ,    $]
[       ( ,  3 ,16 , 0 ,    $]
[       c , 33 ,16 , 0 ,    $]
[       < , 28 ,16 , 0 ,    $]
[       5 , 34 ,16 , 0 ,    $]
[       ) ,  4 ,16 , 0 ,    $]
[       { ,  1 ,17 , 0 ,    $]
[       c , 33 ,18 , 0 ,    $]
[       = , 22 ,18 , 0 ,    $]
[       c , 33 ,18 , 0 ,    $]
[       + , 10 ,18 , 0 ,    $]
[       1 , 34 ,18 , 0 ,    $]
[       ; ,  5 ,18 , 0 ,    $]
[       } ,  2 ,19 , 0 ,    $]
[     for , 32 ,20 , 0 ,    $]
[       ( ,  3 ,20 , 0 ,    $]
[       a , 33 ,20 , 0 ,    $]
[       < , 28 ,20 , 0 ,    $]
[       c , 33 ,20 , 0 ,    $]
[       ; ,  5 ,20 , 0 ,    $]
[       a , 33 ,20 , 0 ,    $]
[      ++ , 11 ,20 , 0 ,    $]
[       ) ,  4 ,20 , 0 ,    $]
[       { ,  1 ,21 , 0 ,    $]
[       c , 33 ,22 , 0 ,    $]
[       = , 22 ,22 , 0 ,    $]
[       c , 33 ,22 , 0 ,    $]
[       - , 13 ,22 , 0 ,    $]
[       1 , 34 ,22 , 0 ,    $]
[       ; ,  5 ,22 , 0 ,    $]
[       } ,  2 ,23 , 0 ,    $]
程序分析结束!
```

Final program

生成四元式

```
------四元式生成成功------
1 (=, 0, _, a)
2 (=, 1, _, c)
3 (FJ, 6, c, _)
4 (=, 1, _, b)
5 (RJ, 7, _, _)
6 (=, 10, _, b)
7 (<, c, 5, t1)
8 (FJ, 12, t1, _)
9 (+, c, 1, t2)
10 (=, t2, _, c)
11 (RJ, 7, _, _)
12 (<, a, c, t3)
13 (FJ, 19, t3, _)
14 (-, c, 1, t4)
15 (=, t4, _, c)
16 (+, a, 1, t5)
17 (=, t5, _, a)
18 (RJ, 12, _, _)
```

目标代码

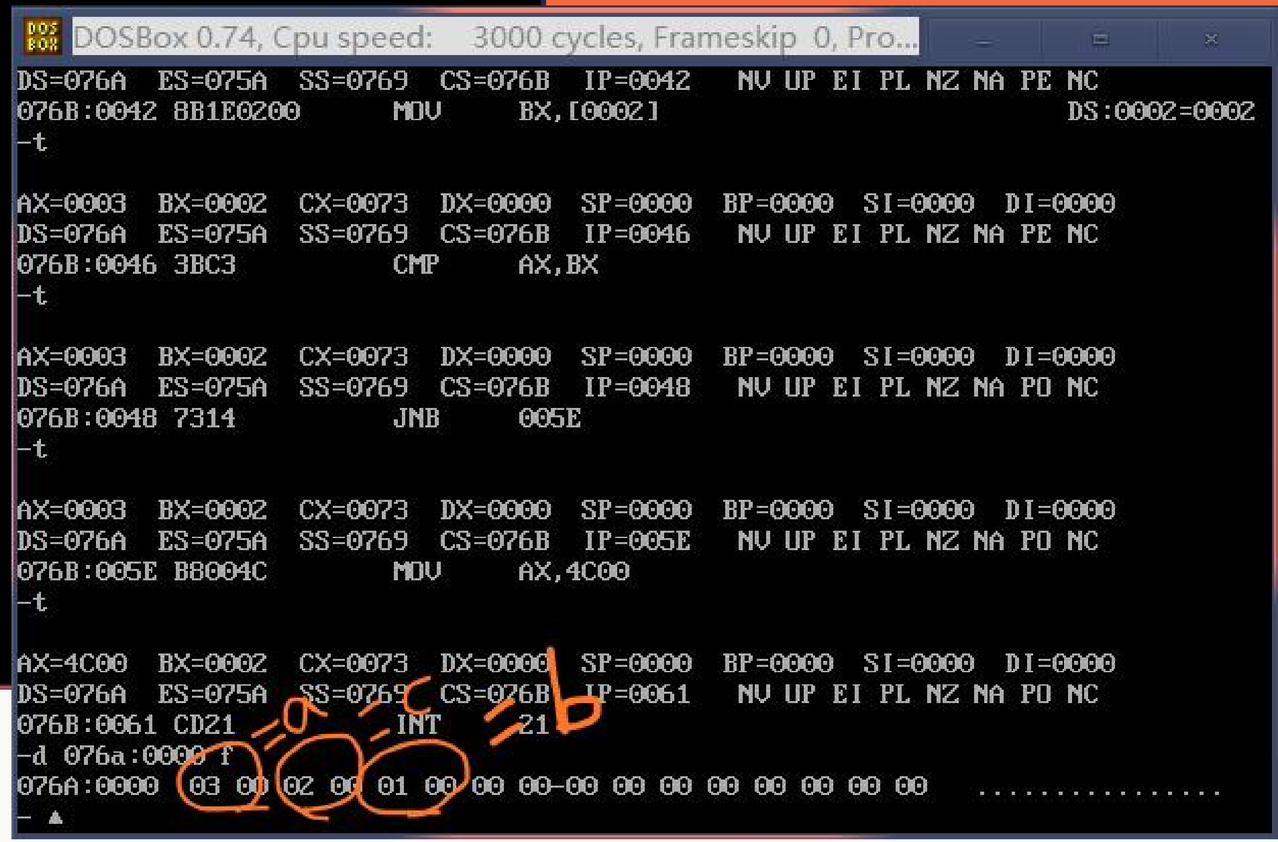
```
-----------assemble-----------
ASSUME CS : CODE , DS : DATA
DATA SEGMENT
        a       DW  0
        c       DW  0
        b       DW  0
DATA ENDS
CODE SEGMENT

START:
        MOV  AX , DATA
        MOV  DS , AX
        MOV  AX , 0
        MOV  a , AX
        MOV  AX , 1
        MOV  c , AX
        MOV  AX , c
        MOV  BX , 1
        CMP  AX , BX
        JNE  s0
        MOV  AX , 1
        MOV  b , AX
        JMP  s1
s0:
        MOV  AX , 10
        MOV  b , AX
```

```
s1:
        MOV  AX , c
        MOV  BX , 5
        CMP  AX , BX
        JNB  s2
        MOV  AX , c
        ADD  AX , 1
        MOV  c , AX
        JMP  s1
s2:
        MOV  AX , a
        MOV  BX , c
        CMP  AX , BX
        JNB  s4
        MOV  AX , c
        SUB  AX , 1
        MOV  c , AX
        MOV  AX , a
        ADD  AX , 1
        MOV  a , AX
        JMP  s2
s4:


NEXT:   MOV AX, 4c00H
        INT 21H

CODE ENDS
 END START
```

DOSBox Test

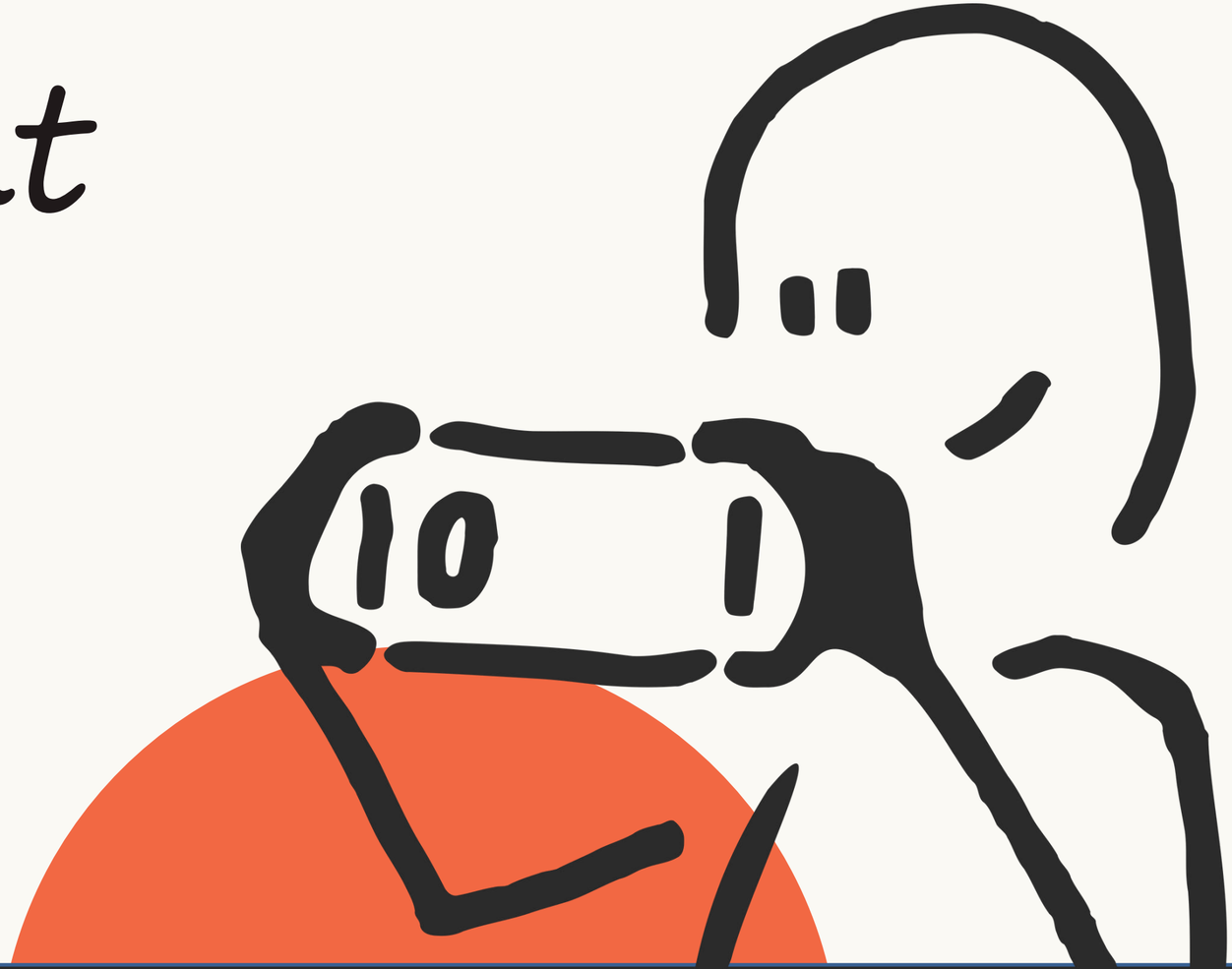

DOSBox 0.74, Cpu speed: 3000 cycles, Fram
C:\>link assemble
Microsoft (R) Overlay Linker Version 3.60
Copyright (C) Microsoft Corp 1983-1987. All
Run File [ASSEMBLE.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:
LINK : warning L4021: no stack segment
C:\>debug assemble.exe
-u
076B:0000 B86A07 MOV AX,076A
076B:0003 8ED8 MOV DS,AX
076B:0005 B80000 MOV AX,0000
076B:0008 A30000 MOV [0000],AX
076B:000B B80100 MOV AX,0001
076B:000E A30200 MOV [0002],AX
076B:0011 A10200 MOV AX,[0002]
076B:0014 BB0100 MOV BX,0001
076B:0017 3BC3 CMP AX,BX
076B:0019 7509 JNZ 0024
076B:001B B80100 MOV AX,0001
076B:001E A30400 MOV [0004],AX

DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Pro...
076B:0024 B80A00 MOV AX,000A
076B:0027 A30400 MOV [0004],AX
076B:002A A10200 MOV AX,[0002]
076B:002D BB0500 MOV BX,0005
076B:0030 3BC3 CMP AX,BX
076B:0032 730B JNB 003F
076B:0034 A10200 MOV AX,[0002]
076B:0037 050100 ADD AX,0001
076B:003A A30200 MOV [0002],AX
076B:003D EBEB JMP 002A
076B:003F A10000 MOV AX,[0000]
-u
076B:0042 8B1E0200 MOV BX,[0002]
076B:0046 3BC3 CMP AX,BX
076B:0048 7314 JNB 005E
076B:004A A10200 MOV AX,[0002]
076B:004D 2D0100 SUB AX,0001
076B:0050 A30200 MOV [0002],AX
076B:0053 A10000 MOV AX,[0000]
076B:0056 050100 ADD AX,0001
076B:0059 A30000 MOV [0000],AX
076B:005C EBE1 JMP 003F
076B:005E B8004C MOV AX,4C00
076B:0061 CD21 INT 21
-a


DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Pro...
DS=076A ES=075A SS=0769 CS=076B IP=0042 NV UP EI PL NZ NA PE NC
076B:0042 8B1E0200 MOV BX,[0002] DS:0002=0002
-t
AX=0003 BX=0002 CX=0073 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=076A ES=075A SS=0769 CS=076B IP=0046 NV UP EI PL NZ NA PE NC
076B:0046 3BC3 CMP AX,BX
-t
AX=0003 BX=0002 CX=0073 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=076A ES=075A SS=0769 CS=076B IP=0048 NV UP EI PL NZ NA PO NC
076B:0048 7314 JNB 005E
-t
AX=0003 BX=0002 CX=0073 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=076A ES=075A SS=0769 CS=076B IP=005E NV UP EI PL NZ NA PO NC
076B:005E B8004C MOV AX,4C00
-t
AX=4C00 BX=0002 CX=0073 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=076A ES=075A SS=0769 CS=076B IP=0061 NV UP EI PL NZ NA PO NC
076B:0061 CD21 INT 21
-d 076a:0000 f
076A:0000 03 00 02 00 01 00 00 00-00 00 00 00 00 00 00 00 ....

Improvement

- 对计算机系统本质的深刻理解： 通过亲手实现编译器的四个核心阶段，彻底理解了词法分析、语法分析、语义分析和代码生成的原理，构建了坚实的系统级知识底座。
- 复杂系统设计与构建能力： 编译器是一个极其复杂的系统，本项目极大地锻炼了我的软件架构能力，学会了如何设计清晰的数据结构和模块接口来管理复杂性。
- 严谨的工程素养： 编译器的任何微小错误都会导致完全失败，这培养了我极致的代码严谨性和调试能力，学会了如何系统性地设计和执行测试用例。



CompileCore是一次“解剖地基”的旅程。
它让我深入到计算机科学最核心的领域，去理解我们编写的每一行代码最终是如何被机器理解和执行的。从设计LL(1)文法时的小心求证，到生成第一条正确的MOV指令时的激动万分，这个过程极大地锻炼了我的系统思维和逻辑严密性。
这让我对计算机系统的整体理解产生了质的飞跃。