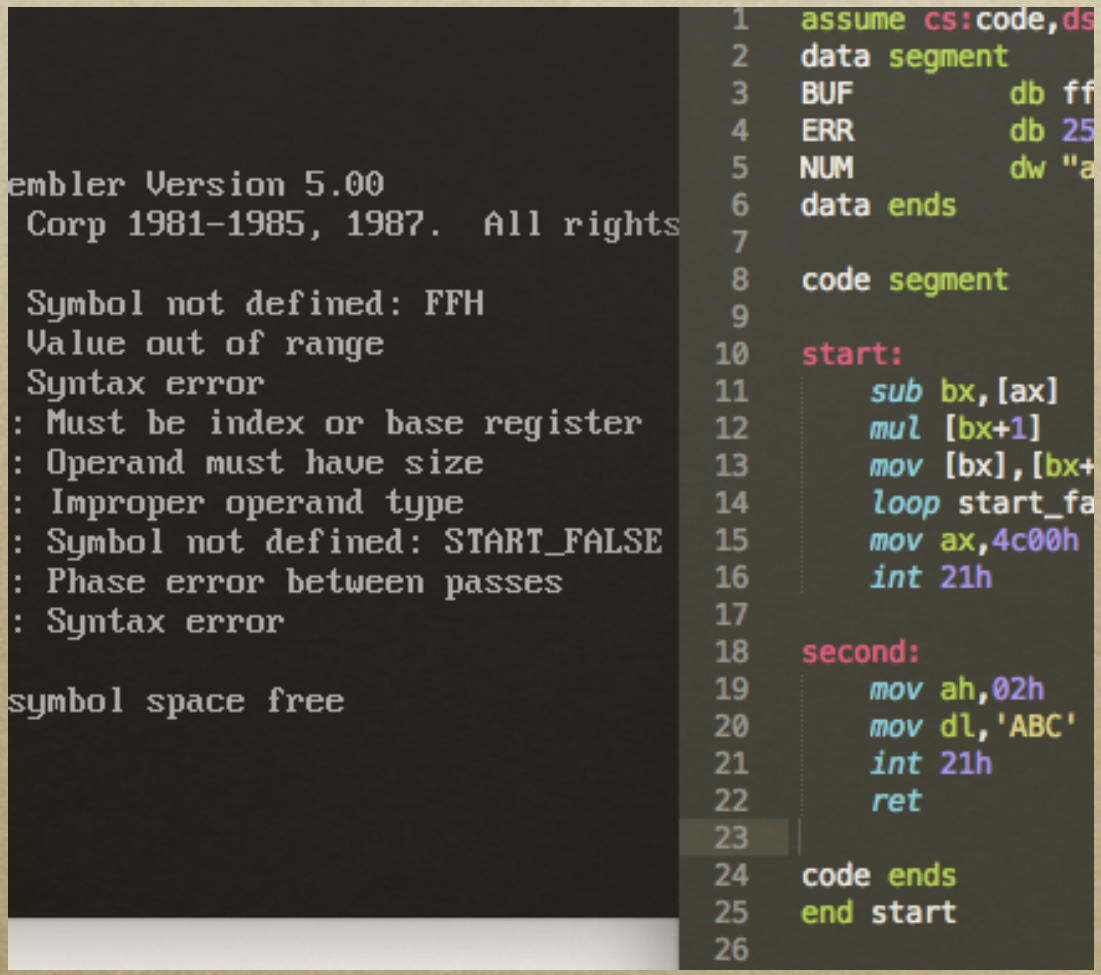
小型MASM编译器

李新星-PB13214040

一.为什么要做MASM编译器

- gcc和clang
- 微软的MASM编译器缺点：
  - 词法和语法错误报错信息不完善，只告诉你错，不告诉你为什么错
  - 无法检测运行时错误
- 配合DOS获得更多缺点：
  - 一旦陷入死循环就会死机
  - 有些版本只有大写、单色字母（太丑）



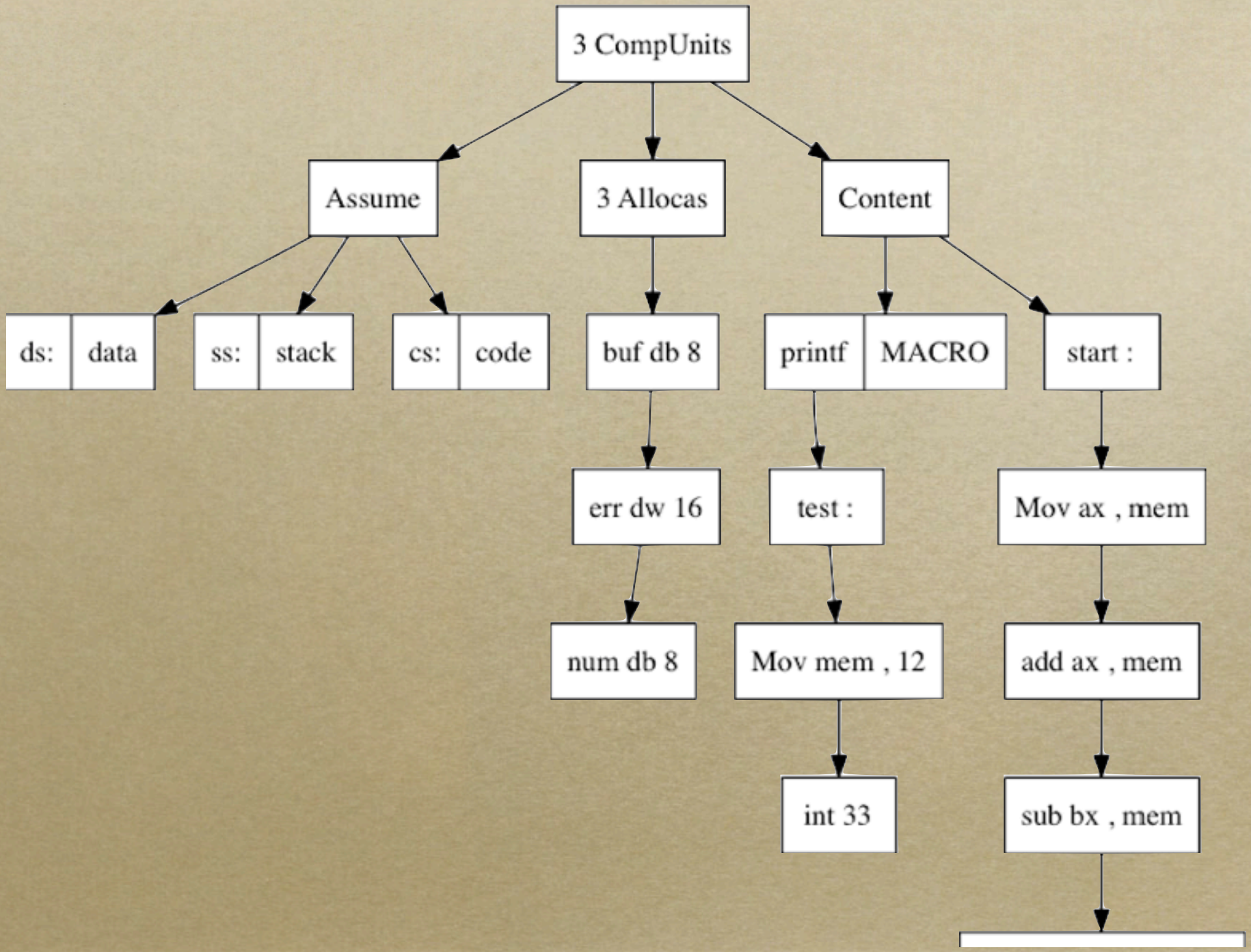
二.我在我的编译器上做的工作

- 自己构建语法结构，实现词法分析、语法分析

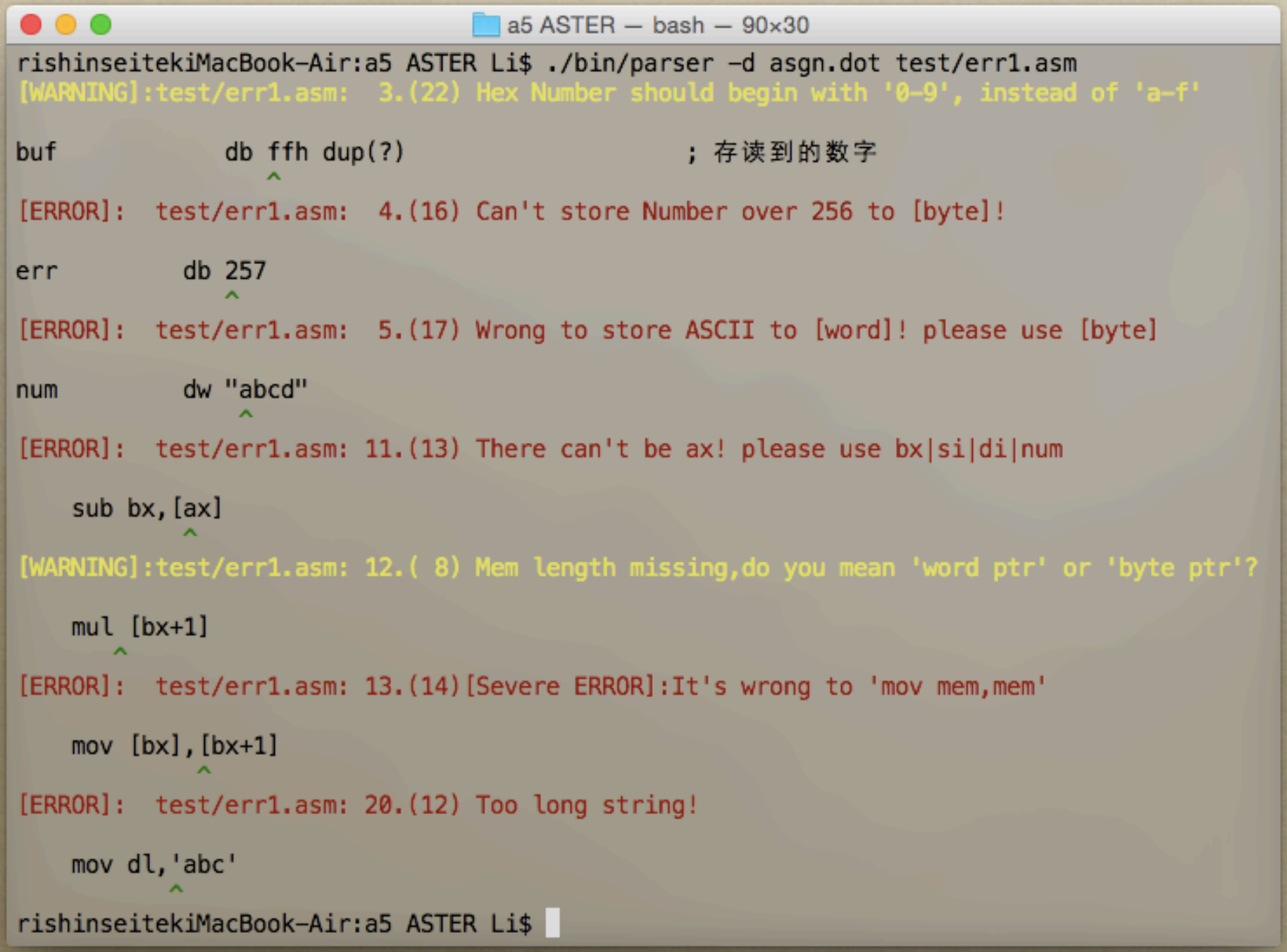

```
assume cs:code,ds:data
data segment
BUF         db 0ffh dup(?)                  ; 存读到
ERR         dw 256
NUM         db "abcd"
data ends

code segment
printf  MACRO x
test:
        mov x,12
        int 21h
        ENDM

start:
    mov ax,offset BUF          ; mov指令 + 偏移量寻址
    add ax,ERR                 ; add指令 + 直接寻址
    sub bx,[bx]                ; sub指令 + 寄存器寻址
    mul byte ptr [bx+1]        ; mul指令 + 寄存器加偏移
    div word ptr [bx+di+1]     ; div指令 + 更灵活的寻址
```

```
    inc NUM                    ; inc指令
    dec cx                     ; dec指令
    loop start                 ; loop指令
    cmp cx,ax                  ; cmp指令
    jge  start                 ; 比较指令
    call second                ; call指令
    push ax                    ; push指令
    pop  ax                    ; pop指令
    mov ax,4c00h
    int 21h                    ; 调用中断驱动，结束程序

second:                        ; 定义函数
    mov ah,02h
    mov dl,'A'
    int 21h
    ret

code ends
end start
```

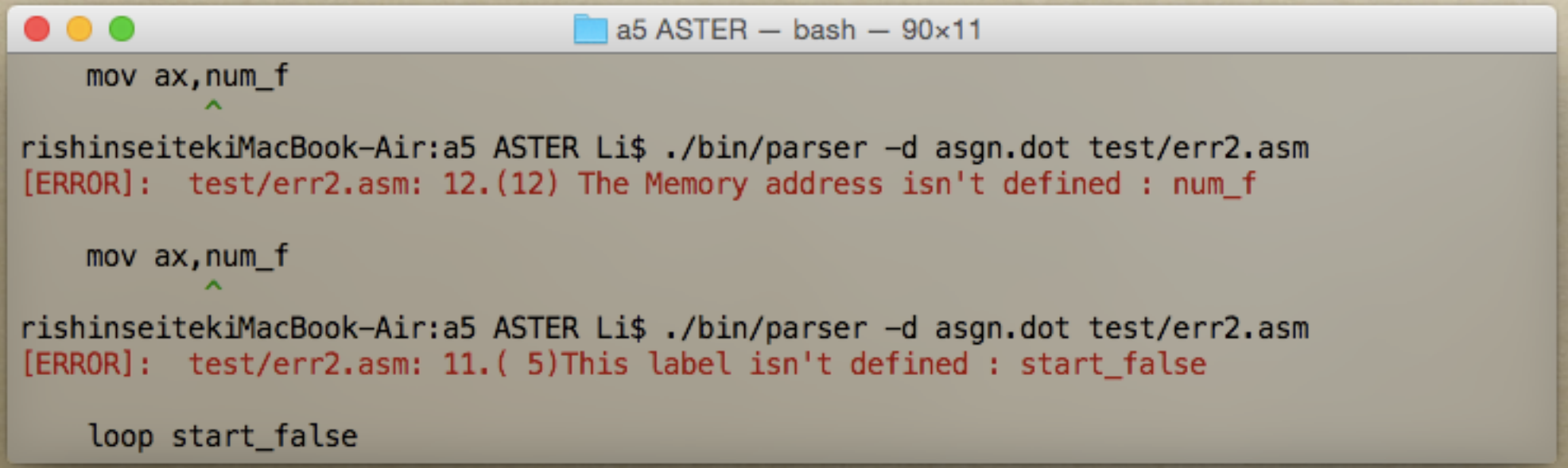
- 构建抽象语法树（AST），画出简单示意图



- 首先对一些词法和语法上的错误进行检错(同时输出对应行源码)

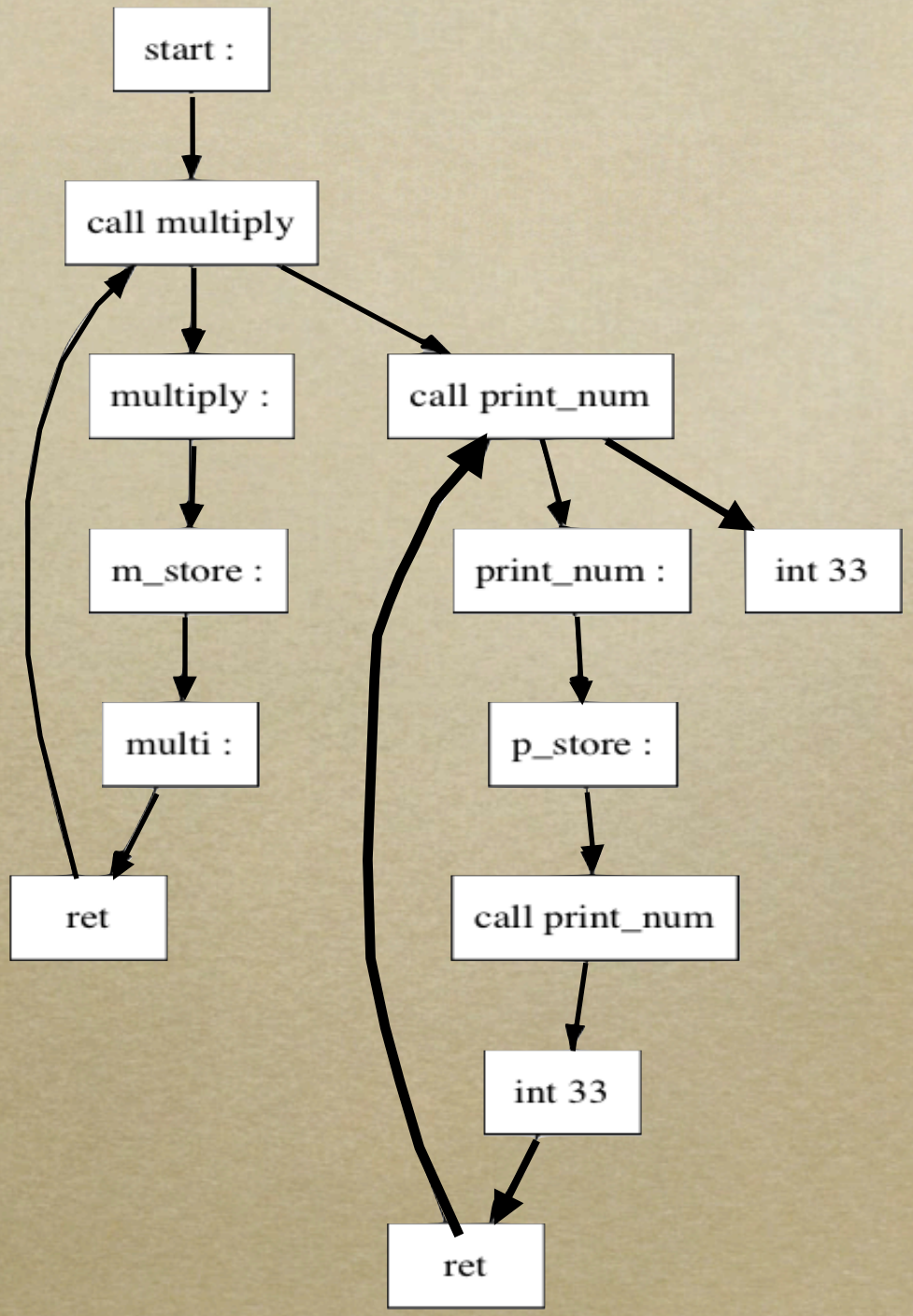


- 其次，建立符号表，检查引用Label和Mem时的错误



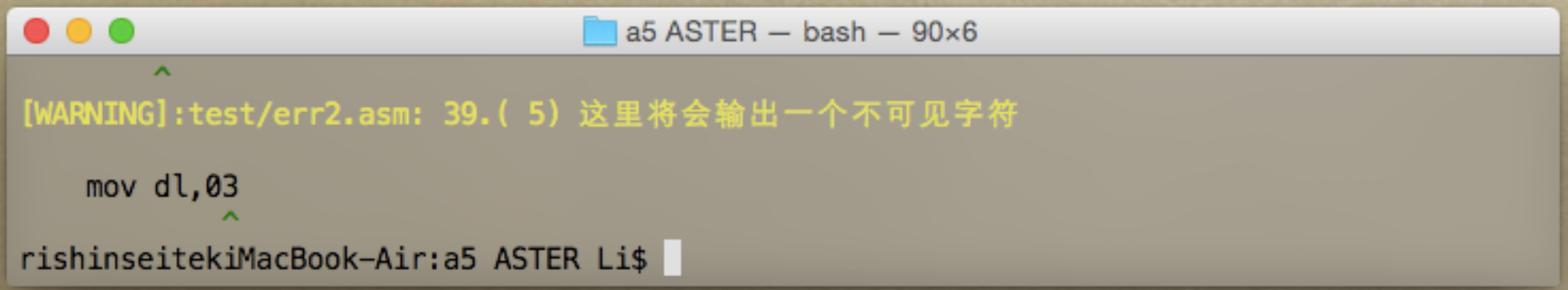
- 接下来要在对语法树的第2、3遍扫描上做文章了

- 1.可以画出函数调用图（其实更像是label的流程图）



- 2.如果程序企图输出一个不可见字符，
  那么这个错误会被检测出来

```
36  second:
37      mov ah,02h
38      mov dl,03
39      int 21h
40      mov ah,02h
41      mov dl,'A'
```



- 3.可以检测出多种死循环代码（重要！）

```
14  loop1:
15      mov cx,2
16      loop loop1
17  loop2:
18      push 3
19      pop ax
20      cmp ax,3
21      je loop2
22  loop3:
23      add ax,30
24      cmp ax,20
25      jge loop3
26  loop4:
27      mul ah
28      div al
29      cmp cx,10
30      jle loop4
```

```
a5 ASTER — bash — 76×20
[WARNING]:test/err2.asm: 16.(14) There is a died circle! P

    loop loop1
         ^
[WARNING]:test/err2.asm: 21.(12) There is a died circle! P

    je loop2
       ^
[WARNING]:test/err2.asm: 25.(13) There is a died circle! P

    jge loop3
        ^
[WARNING]:test/err2.asm: 28.(10) 除数不能为0 !

    div al
        ^
[WARNING]:test/err2.asm: 30.(13) There is a died circle! Please fix it!

    jle loop4
        ^
```

- 3.可以检测出多种死循环代码（重要！）

```
14  loop1:
15      mov cx,2
16      loop loop1
17  loop2:
18      push 3
19      pop ax
20      cmp ax,3
21      je loop2
```

```
ax,30
ax,20
loop3

ah
al
cx,10
loop4
```

a5 ASTER — bash — 76×20

```
[WARNING]:test/err2.asm: 16.(14) There is a died circle! Please fix it!

    loop loop1
         ^
[WARNING]:test/err2.asm: 21.(12) There is a died circle! Please fix it!

    je loop2
       ^
[WARNING]:test/err2.asm: 25.(13) There is a died circle! Please fix it!

    jge loop3
        ^
[WARNING]:test/err2.asm: 28.(10) 除数不能为0 !

    div al
        ^
[WARNING]:test/err2.asm: 30.(13) There is a died circle! Please fix it!

    jle loop4
        ^
```

三.总结

- 能分析主流指令，生成语法树
- 能对微软编译器不友好显示的信息，更加详细地报错，提示你为什么错，正确的应该怎么写
- 建立符号表，能检出符号表错误
- 针对运行时检查：
  - 能检出尝试输出不可见字符的错误
  - 能检出死循环错误
  - 能输出函数调用图
- 缺憾：不能完成代码生成

四. BUG & 新的理解

```
Block      : var_tok ':'          {
                                    $$ = new BlockNode($1,NULL);
                                    $$->setLoc((Loc*)&(@$));
                                    all_node.push_back((Node*)$$);
                                    global->SetLabelNode($1,((BlockNode*)$$));
                                  }
           BlockItem              {
                                    line("Line:%-4d",@$.first_line);
                                    debug ("Block ::= %s : BlockItem\n",$1->c_str());
                                    if(error_flag == 0)
                                    {
                                      BlockNode * now = global->GetLabelNode($1);
                                      now ->block_root = (BlockItemNode *)$4;
                                      now ->setLoc((Loc*)&(@$));
                                      $$ = now;
                                    }
```

四. BUG & 新的理解

var_tok MACRO

var_tok :

var_tok db

var_tok segment

var_tok ends

LR(2) or LR(1)?

Thanks