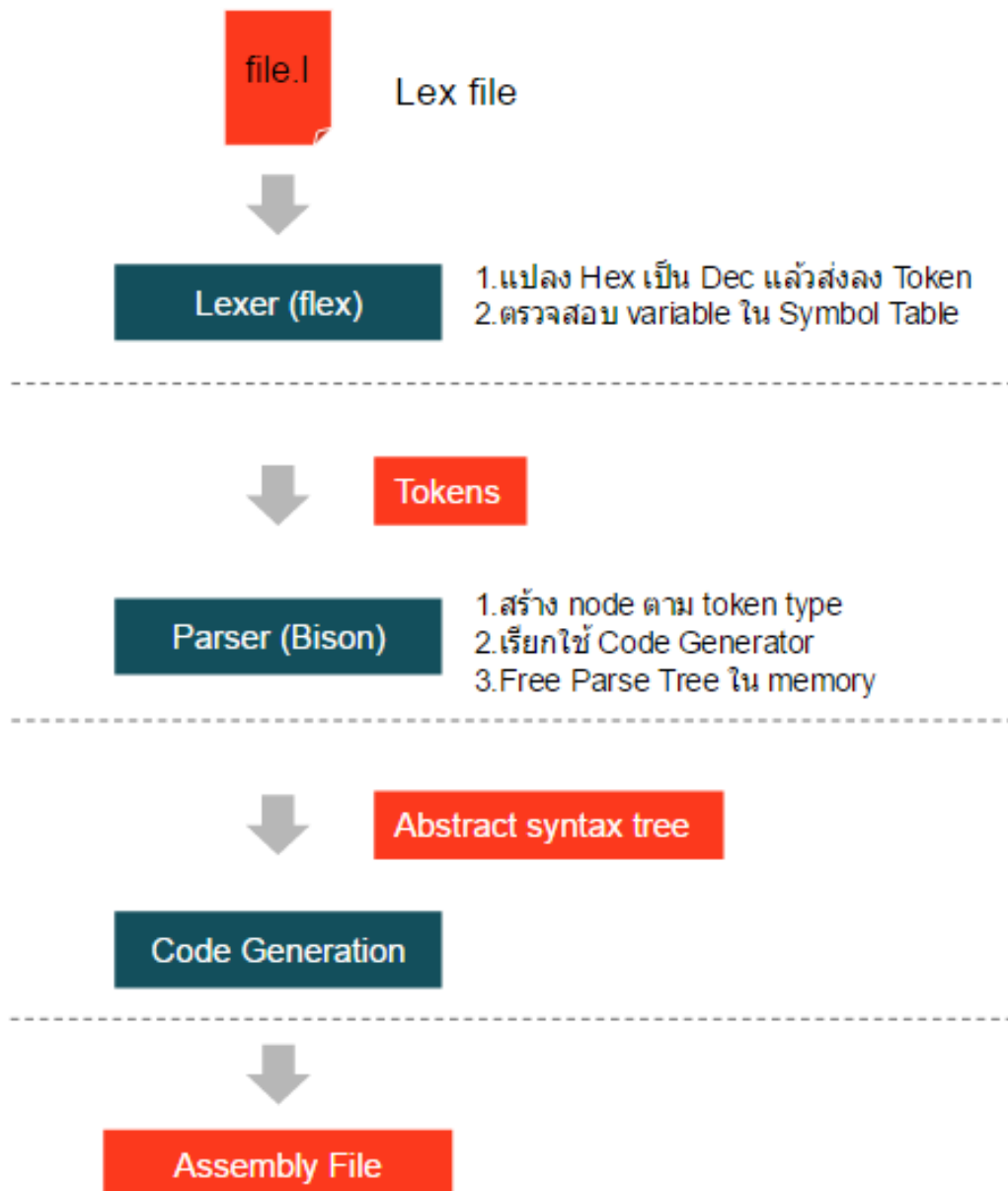
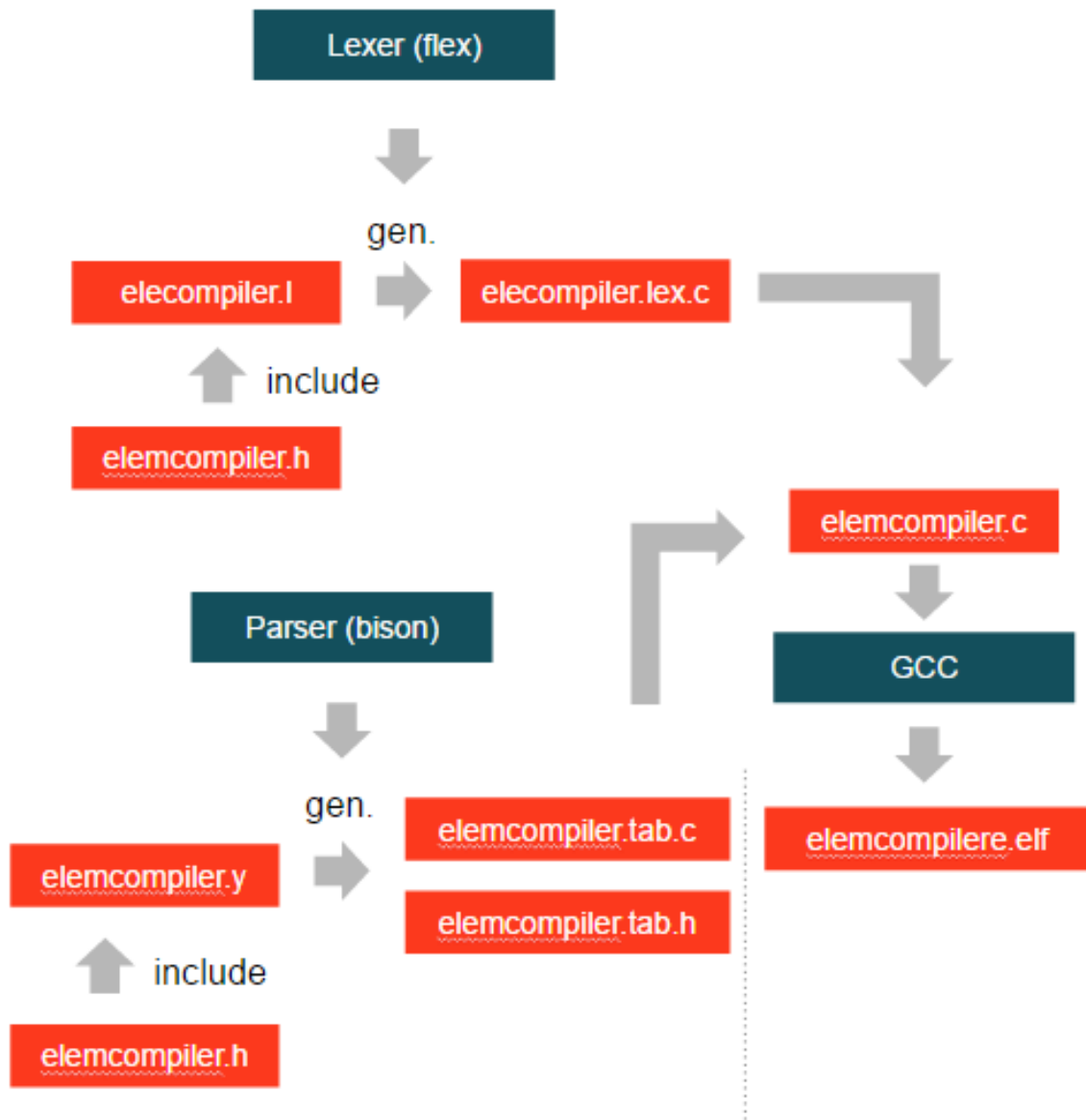


1.แนวคิดและวิธีการ



รูปที่ 1 : ภาพรวมการทำงานของโปรแกรม



รูปที่ 2 : ภาพรวมการทำงานของโปรแกรม

File elecompiler.l

ทำหน้าที่เป็น lexical analyzer หรือ lexer ในการแบ่ง string ใน file input เพื่อแบ่งข้อมูล input ออกเป็นส่วน โดยไม่คิดความหมาย เรียกว่า token เพื่อนำไปตรวจสอบความหมายต่อไป โดยรูปแบบของ token แต่ละประเภทจะโดนแยกด้วย regular expression (regex) โดยภายใน elecompiler.l จะมีการ include elecompiler.h และทำงานหลักๆคือ

- แปลง string โดนใช้ regex เป็น token
- มีการแปลง hex เป็น dec ก่อนเก็บลง token
- มีการ check symbol table ของ variable

โดยเมื่อนำ elecompiler.l ไปผ่าน flex แล้วจะได้ไฟล์ elecomiler.lex.c ที่เป็นไฟล์ code ที่ได้จากการ gen ของ flex

File elecompiler.y

ทำหน้าที่เป็น Syntax Analyzer หรือ parser เพื่อตรวจสอบไวยากรณ์หรือความหมายของ input เนื่องจาก flex ไม่ได้ตรวจสอบความหมายแต่เพียงตรวจสอบว่าคำถูกต้องหรือไม่ ภายใน parser จะทำการแยก syntax ตามที่ได้กำหนดไว้ โดยมีลำดับการทำงานคือ

- 1.ทำการ parse จากล่างขึ้นบน (bottom-up)
- 2.สร้าง node ของแต่ละ node ขึ้นมาตาม type ของ node แต่ละ node
- 3.เมื่อถึง root node ก็จะมีการ return Abstract syntax tree ไปที่ generate assembly โดย
- 4.ทำการ free parse tree ใน memory ออก

File elecompiler.h

เป็นส่วนที่ใช้ในการกำหนดและสร้าง โครงสร้างของ node แต่ละ type โดยจะมี attribute ของ node หลักๆ คือ left_node , right_node ที่จะบอกตำแหน่งของ node ต่อไปซึ่งเป็นตัวกระทำหรือตัวดำเนินการทางด้านซ้ายด้านขวา และ value ซึ่งจะเก็บค่าของ token ของ node นั้นๆ

File elecompiler.c

เก็บ function หลักๆที่ใช้ในการสร้าง assembly code โดยจะมีการ tarverse ที่ใช้สร้าง code ภาษา assembly และทำการตรวจสอบว่า แต่ละ node มีตัวกระทำเป็นอะไรและทำการ เขียน assembly code ตามตัวกระทำนั้นๆ

File elecompiler.tab.c และ elecompiler.tab.h

เป็นไฟล์ที่ได้จากการ gen ของ bison เพื่อให้สามารถนำไปใช้งานได้โดยไม่ต้องมี bison และ elecompiler.y

File elecompiler.lex.c

เป็นไฟล์ที่ได้จากการ gen ของ flex เพื่อให้สามารถนำไปใช้งานได้โดยไม่ต้องมี flex และ elecompiler.l

File Makefile

เป็นไฟล์ที่ใช้ในการช่วยการ gen ไฟล์ flex , bison และ assembly โดยมีการทำงานดังนี้
ขั้นตอนการสร้าง ไฟล์

- 1. run คำสั่ง “bison -d elemcompiler.y” เพื่อ gen file elecompiler.tab.h และ elecompiler.tab.c
- 2. run คำสั่ง “flex -o elecompiler.lex.c elecompiler.l” เพื่อ gen file elecompiler.lex.c
- 3. run คำสั่ง “gcc -Wall elecompiler.c elecompiler.tab.c elecompiler.lex.c -o \$@ -lm”

2. ไวยากรณ์และตัวอย่างภาษา

1. การ Assign ค่าตัวแปร

Bison : VARIABLE_NAME = EXPRESSION ;

Example : a = 1;

2. เงื่อนไข (If statement)

Bison : IF (EXPRESSION) { EXPRESSION_LIST }

Example : if (a) {

a = 1+4;

b = 2;

}

3. การวนลูป (Loop statement)

Bison : loop (NUM_1 -> NUM_2 , NUM_3) { EXPRESSION_LIST }

Example : loop (0 -> 10 , 1) {

a = 2*3;

a = (2+2)*3;

}

4. การแสดงค่า Print

Bison-String : PRINT (STRING) ;

Example : print ("Hello Compil.....err") ;

Bison-Expression : PRINT (EXPRESSION) ;

Example : print (1+1);

3.คำอธิบาย Source code

Flex File: elecompiler.l

```


1  %option noyywrap nodefault yylineno
2
3  %{
4  #include "elemcompiler.h"
5  #include "elemcompiler.tab.h"
6
7  static int64_t hexToDec(char *str);
8  static int64_t strToDec(char *str);
9  %}
10
11 EXP ([Ee][-+]?[0-9]+)
12
13 %%
14
15 "->" ..... { return TO; }
16 "+" ..... { return PLUS; }
17 "-" ..... { return MINUS; }
18 "*" ..... { return MULT; }
19 "/" ..... { return DIV; }
20 "%" ..... { return MOD; }
21 "=" ..... { return EQ; }
22 "|" ..... { return OR; }
23 "," ..... { return COMMA; }
24 ";" ..... { return SEMI; }
25 "(" ..... { return LPAREN; }
26 ")" ..... { return RPAREN; }
27 "{" ..... { return LBRACE; }
28 "}" ..... { return RBRACE; }
29 "if" ..... { return IF; }
30 "loop" ..... { return LOOP; }
31 "print" ..... { return PRINT; }
32
33 "\"[^\"]+\"" ..... {
34 ..... yylval.str = strdup(yytext);
35 ..... return STRING;
36 }
37 0(x|X)[a-fA-F0-9]+ ..... {
38 ..... yylval.num = hexToDec(yytext);
39 ..... return NUMBER;
40 }
41 [0-9]+ ..... {
42 ..... yylval.num = strToDec(yytext);
43 ..... return NUMBER;
44 }
45 [a-zA-Z][a-zA-Z0-9]* ..... {
46 ..... yylval.sym = lookup(yytext);
47 ..... return VAR;
48 }
49
50 "//" ..... /* ignore whitespace */
51 [\t] ..... /* ignore line continuation */
52 \\n ..... /* ignore empty line */
53 ^\n ..... { return EOL; }
54 \n ..... { yyerror("Mystery character %c\n", *yytext); }
55 . ..... { yyerror("Mystery character %c\n", *yytext); }
56
57 %%

```

```

58
59 static int64_t c_power(uint8_t base, uint8_t index)
60 {
61     int64_t result = 1;
62     uint8_t count;
63     for (count = index; count; --count)
64         result *= base;
65     return result;
66 }
67
68 static int64_t hexToDec(char *str)
69 {
70     uint8_t len;
71     uint8_t count = 0;
72     int64_t result = 0;
73     char *startstr = (str + 2);
74     for (len = (strlen(startstr)); len; --len) {
75         char ch = (*(startstr + (len - 1)));
76         if (ch > 96)
77             result += (ch - 87) * c_power(16, count);
78         else if (ch > 64)
79             result += (ch - 55) * c_power(16, count);
80         else
81             result += (ch - 48) * c_power(16, count);
82         ++count;
83     }
84     return result;
85 }
86
87 static int64_t strToDec(char *str)
88 {
89     uint8_t len;
90     uint8_t count = 0;
91     int64_t result = 0;
92     for (len = (strlen(str)); len; --len) {
93         char ch = (*(str + (len - 1)));
94         result += (ch - 48) * c_power(10, count);
95         ++count;
96     }
97     return result;
98 }
99

```



อธิบายหลักการและส่วนหลักๆของไฟล์ elecompiler.l

- (1) ส่วนล่างสุดของ File elecompiler.l เป็นส่วนของ User function ที่ถูกสร้างขึ้นและประกาศใช้โดนเป็น function ที่ใช้ในการ แปลงเลข HEX เป็น DEC
- (2) ทำการแปลง regular expression ที่ได้จาก HEX เป็น DEC โดยใช้ User Function ที่ประกาศไว้แล้วเก็บค่าที่เป็น DEC ลง token

Bison File : elecompiler.y

```

1  %{
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <stdint.h>
5  #include "elemcompiler.h"
6  extern uint8_t symCount;
7  extern uint8_t litstrCount;
8  %}
9
10 %union{
11     struct ast *node;
12     struct symbol *sym; /* which symbol */
13     int64_t num;
14     char *str; /* literal string */
15 }
16
17 /* declare tokens */
18 %token <str> STRING
19 %token <num> NUMBER
20 %token <sym> VAR
21 %token EOL
22 %token IF LOOP TO PRINT
23
24 %right '='
25 %left '+' '-'
26 %left '*' '/' '%'
27 %nonassoc UMINUS
28 %type <node> term factor exp explist stmt
29
30 %start root
31 %%
32
33 root: /*nothing */
34     | root stmt EOL { //eval($2);
35                     gen_asm($2);
36                     treefree($2); }
37     | root error EOL { yyerrok; }
38     ;
39
40 stmt: exp ';'
41     | IF '(' exp ')' '{' explist '}' { $$ = newcond($3, $6); }
42     | LOOP '(' NUMBER TO NUMBER ' ' NUMBER ')' '{' explist '}'
43     | { $$ = newloop(newnum($3), newnum($5), newnum($7), $10); }
44     ;
45
46 exp: factor
47     | exp '+' factor { $$ = newast('+', $1, $3); }
48     | exp '-' factor { $$ = newast('-', $1, $3); }
49     | VAR '=' exp { $$ = newasgn($1, $3); }
50     | PRINT '(' STRING ')' { $$ = newprint('S', NULL, $3); ++litstrCount; }
51     | PRINT '(' exp ')' { $$ = newprint('A', $3, NULL); }
52     ;
53
54 factor: term
55     | factor '*' term { $$ = newast('*', $1, $3); }
56     | factor '/' term { $$ = newast('/', $1, $3); }
57     | factor '%' term { $$ = newast('%', $1, $3); }
58     ;
59
60 term: NUMBER { $$ = newnum($1); }
61     | VAR { $$ = newref($1); }
62     | '(' exp ')' { $$ = $2; }
63     | '-' term %prec UMINUS { $$ = newast('M', $2, NULL); }
64     ;
65
66 explist: /*nothing */ { $$ = NULL; }
67     | exp ';' explist { if ($3 == NULL) $$ = $1;
68                       else $$ = newast('E', $1, $3); }
69     ;

```


อธิบายหลักการและส่วนหลักๆของไฟล์ elecompiler.y

- (1,2) ส่วนล่างสุดของ Abstract syntax tree ที่โดนสร้างขึ้นมา
- (3) เมื่อสร้างมาจนถึง root ของ Abstract syntax tree ก็จะทำการเรียก function gen_ast() เพื่อทำการ generate assembly code ต่อไป
- (4) หลังจากทำการ generate assembly code ทำการ free memory ที่เก็บ Abstract syntax tree

Ast prototype File : elecompiler.h

```

1  #include <stdint.h>
2
3  #define NHASH 9997
4
5  /* symbol table */
6  struct symbol { ..... /* a variable name */
7      .... char *name;
8      .... int64_t value;
9      .... uint8_t offset;
10 };
11
12 /* node types
13  * + - * / %
14  * M unary minus
15  * E expression list
16  * C IF statement
17  * L LOOP statement
18  * V symbol ref
19  * = assignment
20  * K constant
21  * S literal string
22  * A Expression
23 */
24
25 /* node in the abstraction syntax tree */
26 /* all have common initial nodetype */
27 struct ast {
28     .... int nodetype;
29     .... struct ast *l;
30     .... struct ast *r;
31 };
32
33 struct cond {
34     .... int nodetype; ..... /* type C */
35     .... struct ast *cond; ..... /* condition */
36     .... struct ast *tl; ..... /* then branch or do list */
37 };
38
39 struct loop {
40     .... int nodetype; ..... /* type L */
41     .... struct ast *from; ..... /* start number */
42     .... struct ast *to; ..... /* end number */
43     .... struct ast *inc; ..... /* incremental number */
44     .... struct ast *tl; ..... /* then branch or do list */
45 };
46
47 struct litstr {
48     .... char *str;
49     .... uint8_t label;
50 };

```

```

51
52 struct print {
53     ... int nodetype; ... /* type S or A */
54     ... union arg { ... /* argument of the print function */
55         ... struct litstr *ls; ... /* string argument */
56         ... struct ast *exp; ... /* expression argument */
57     ... } arg;
58 };
59
60 struct numval {
61     ... int nodetype; ... /* type K */
62     ... int64_t number;
63 };
64
65 struct symref {
66     ... int nodetype; ... /* type V */
67     ... struct symbol *s;
68 };
69
70 struct symasgn {
71     ... int nodetype; ... /* type = */
72     ... struct symbol *s;
73     ... struct ast *v; ... /* value */
74 };
75
76 /* simple symtab of fixed size */
77 struct symbol symtab[NHASH];
78
79 struct symbol *lookup(char*);
80
81 /* build an AST */
82 struct ast *newast(int nodetype, struct ast *l, struct ast *r);
83 struct ast *newprint(int nodetype, struct ast *exp, char *str);
84 struct ast *newref(struct symbol *s);
85 struct ast *newasgn(struct symbol *s, struct ast *v);
86 struct ast *newnum(int64_t d);
87 struct ast *newcond(struct ast *cond, struct ast *tl);
88 struct ast *newloop(struct ast *from, struct ast *to, struct ast *inc, struct ast *tl);
89
90 /* evaluate an AST */
91 int64_t eval(struct ast *);
92 void gen_asm(struct ast *);
93
94 /* delete and free an AST */
95 void treefree(struct ast *);
96
97 /* interface to the lexer */
98 extern int yylineno; ... /* from lexer */
99 void yyerror(char *s, ...);
100
101

```

อธิบายหลักการและส่วนหลักๆของไฟล์ elecompiler.h

- (1) Struct ของ symbol table
- (2) Struct ของ Abstract syntax tree
- (3,4,5,6,7,8,9) Struct ของ type ของ node แต่ละชนิด ที่จะต้องมี Struct ของ Abstract syntax tree เพื่อที่จะให้สามารถต่อ tree ไปแต่ละ Node ได้ ซึ่ง Attribute ของแต่ละ Node จะแตกต่างกันไปตาม แต่ละชนิดข้อมูลที่ได้จาก การทำ Parser

Main Compiler File : elecompiler.c

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdarg.h>
4  #include <string.h>
5  #include <math.h>
6
7  #include "elemcompiler.h"
8  #include "codegen.h"
9
10 static uint8_t offsetCount = 0;
11 static uint8_t branchCount = 0;
12 static uint8_t litstrCount = 2;
13 static uint8_t symCount = 0;
14
15 static FILE *fp;
16 static char *text;
17 static char *data;
18
19
20
21 /* symbol table */
22 /* hash a symbol */
23 static unsigned symhash(char *sym)
24 {
25     unsigned int hash = 0;
26     unsigned c;
27
28     while ((c = *(sym++)))
29         hash = hash * 9 ^ c;
30
31     return hash;
32 }
33
34 struct symbol *lookup(char *sym)
35 {
36     struct symbol *sp = &symtab[symhash(sym)%NHASH];
37     int scount = NHASH; /* how many have we looked at */
38
39     while (--scount >= 0) {
40         if (sp->name && !strcmp(sp->name, sym))
41             return sp;
42
43         if (!sp->name) { /* new entry */
44             sp->name = strdup(sym);
45             sp->value = 0;
46             sp->offset = 0;
47
48             ++symCount;
49
50             return sp;
51         }
52
53         if (++sp >= symtab+NHASH)
54             sp = symtab; /* try the next entry */
55     }
56
57     yyerror("symbol table overflow\n");
58     abort(); /* tried them all, table is full */
59 }
60

```

```

61 struct ast* newast (int nodetype, struct ast* l, struct ast* r)
62 {
63     struct ast* a = (struct ast*) malloc(sizeof(struct ast));
64
65     if (!a) {
66         yyerror("out of space");
67         exit(0);
68     }
69
70     a->nodetype = nodetype;
71     a->l = l;
72     a->r = r;
73
74     return a;
75 }
76
77 struct ast* newnum (int64_t num)
78 {
79     struct numval* a = (struct numval*) malloc(sizeof(struct numval));
80
81     if (!a) {
82         yyerror("out of space");
83         exit(0);
84     }
85
86     a->nodetype = 'K';
87     a->number = num;
88
89     return (struct ast*) a;
90 }
91
92 struct ast* newprint (int nodetype, struct ast* exp, char* str)
93 {
94     struct print* a = (struct print*) malloc(sizeof(struct print));
95     struct litstr* ls = (struct litstr*) malloc(sizeof(struct litstr));
96
97     if (!a) {
98         yyerror("out of space");
99         exit(0);
100     }
101
102     a->nodetype = nodetype;
103
104     if (nodetype == 'A')
105         a->arg.exp = exp;
106     else {
107         char buf[21];
108
109         sprintf(buf, "LC%u", litstrCount);
110         data = buildDataSection(data, defineString(buf, str));
111
112         ls->str = str;
113         ls->label = litstrCount++;
114         a->arg.ls = ls;
115     }
116
117     return (struct ast*) a;
118 }
119

```

```

120 struct ast* newref (struct symbol *s)
121 {
122     struct symref *a = (struct symref*) malloc(sizeof(struct symref));
123
124     if (!a) {
125         yyerror("out of space");
126         exit(0);
127     }
128
129     a->nodetype = 'V';
130     a->s = s;
131
132     return (struct ast*) a;
133 }
134
135 struct ast* newasgn (struct symbol *s, struct ast *v)
136 {
137     struct symasgn *a = (struct symasgn*) malloc(sizeof(struct symasgn));
138
139     if (!a) {
140         yyerror("out of space");
141         exit(0);
142     }
143
144     a->nodetype = '=';
145     a->s = s;
146     a->v = v;
147
148     return (struct ast*) a;
149 }
150
151 struct ast* newcond (struct ast *cond, struct ast *tl)
152 {
153     struct cond *a = (struct cond*) malloc(sizeof(struct cond));
154
155     if (!a) {
156         yyerror("out of space");
157         exit(0);
158     }
159
160     a->nodetype = 'C';
161     a->cond = cond;
162     a->tl = tl;
163
164     return (struct ast*) a;
165 }
166
167 struct ast* newloop (struct ast *from, struct ast *to, struct ast *inc, struct ast *tl)
168 {
169     struct loop *a = (struct loop*) malloc(sizeof(struct loop));
170
171     if (!a) {
172         yyerror("out of space");
173         exit(0);
174     }
175
176     a->nodetype = 'L';
177     a->from = from;
178     a->to = to;
179     a->inc = inc;
180     a->tl = tl;
181
182     return (struct ast*) a;
183 }

```

Diagram illustrating the flow of control between function calls:

- Red line 5 connects the return statement of `newref` to its caller.
- Red line 6 connects the return statement of `newasgn` to its caller.
- Red line 7 connects the return statement of `newcond` to its caller.
- Red line 8 connects the return statement of `newloop` to its caller.

```

184
185 /* free a tree of ASTs */
186 void treefree (struct ast *a)
187 {
188     switch (a->nodetype) {
189         /* two subtrees */
190         case '+':
191         case '-':
192         case '*':
193         case '/':
194         case '%':
195         case '1':
196         case '2':
197         case '3':
198         case '4':
199         case '5':
200         case '6':
201         case 'E':
202             treefree(a->r);
203
204         /* one subtrees */
205         case 'M':
206             treefree(a->l);
207
208         /* no subtree */
209         case 'K':
210         case 'V':
211             break;
212
213         /* print expression */
214         case 'A':
215             treefree(((struct print*)a)->arg.exp);
216             break;
217
218         /* print string */
219         case 'S':
220             free(((struct print*)a)->arg.ls->str);
221             free(((struct print*)a)->arg.ls);
222             break;
223
224         /* assignment */
225         case '=':
226             treefree(((struct symasn*)a)->v);
227             break;
228
229         /* condition */
230         case 'C':
231             treefree(((struct cond*)a)->cond);
232
233             if (((struct cond*)a)->t1)
234                 treefree(((struct cond*)a)->t1);
235
236             break;
237
238         /* loop */
239         case 'L':
240             treefree(((struct loop*)a)->from);
241             treefree(((struct loop*)a)->to);
242             treefree(((struct loop*)a)->inc);
243
244             if (((struct loop*)a)->t1)
245                 treefree(((struct loop*)a)->t1);
246
247             break;
248
249         default:
250             printf("internal error: free bad node %c\n", a->nodetype);
251     }
252
253     free(a); /* always free the node itself */
254 }

```



```

256 void gen_asm (struct ast *a)
257 {
258     struct symbol *s;
259     char buf[21];
260
261     if (!a) {
262         yyerror("internal error, null eval");
263
264         return;
265     }
266
267     switch (a->nodetype) {
268         /* constant */
269         case 'K':
270             //printf("\n# constant\n");
271
272             text = buildTextSection(text, sub("$8", "%rsp"));
273             sprintf(buf, "$%ld", ((struct numval *)a)->number);
274             text = buildTextSection(text, movl(buf, "%eax"));
275             text = buildTextSection(text, movq("%rax", "(%rsp)"));
276
277             break;
278
279         /* symbol reference */
280         case 'V':
281             //printf("\n# reference\n");
282
283             s = lookup(((struct symasn *)a)->s->name);
284
285             if (s->offset == 0) {
286                 s->offset = (++offsetCount) * 8;
287                 sprintf(buf, "-%u(%rbp)", s->offset);
288                 text = buildTextSection(text, movq("$0", buf));
289             }
290
291             sprintf(buf, "-%u(%rbp)", s->offset);
292             text = buildTextSection(text, movq(buf, "%rax"));
293             text = buildTextSection(text, sub("$8", "%rsp"));
294             text = buildTextSection(text, movq("%rax", "(%rsp)"));
295
296             break;
297
298         /* print expression */
299         case 'A':
300             gen_asm(((struct print *)a)->arg.exp);
301
302             //printf("\n# print A\n");
303
304             text = buildTextSection(text, movq("(%rsp)", "%rax"));
305             text = buildTextSection(text, add("$8", "%rsp"));
306             text = buildTextSection(text, movq("%rax", "%rsi"));
307             text = buildTextSection(text, movl("$.LC0", "%edi"));
308             text = buildTextSection(text, movl("$0", "%eax"));
309             text = buildTextSection(text, sysCallPrint());
310
311             break;
312

```

```

312 .....
313 ..... /* print literal string */
314 ..... case 'S':
315 ..... //printf("\n# print S\n");
316 .....
317 ..... sprintf(buf, "%.LC%u", ((struct print*)a)->arg.ls->label);
318 ..... text = buildTextSection(text, movl(buf, "%esi"));
319 ..... text = buildTextSection(text, movl("%.LC1", "%edi"));
320 ..... text = buildTextSection(text, movl("$0", "%eax"));
321 ..... text = buildTextSection(text, syscallPrint());
322 ..... break;
323 .....
324 ..... /* assignment */
325 ..... case '=':
326 ..... s = lookup(((struct symasgn*)a)->s->name);
327 .....
328 ..... if (s->offset == 0) {
329 .....     s->offset = (++offsetCount) * 8;
330 ..... }
331 .....
332 ..... gen_asm(((struct symasgn*)a)->v);
333 .....
334 ..... //printf("\n# assignment\n");
335 .....
336 ..... text = buildTextSection(text, movq("(%rsp)", "%rax"));
337 ..... text = buildTextSection(text, add("$8", "%rsp"));
338 ..... sprintf(buf, "-%u(%%rbp)", s->offset);
339 ..... text = buildTextSection(text, movq("%rax", buf));
340 ..... text = buildTextSection(text, sub("$8", "%rsp"));
341 ..... text = buildTextSection(text, movq("%rax",("(%rsp)"));
342 .....
343 ..... break;
344 .....
345 ..... /* expressions */
346 ..... case '+':
347 .....     gen_asm(a->r);
348 .....     gen_asm(a->l);
349 .....
350 ..... //printf("\n# addition\n");
351 .....
352 ..... text = buildTextSection(text, movq("(%rsp)", "%rdx"));
353 ..... text = buildTextSection(text, add("$8", "%rsp"));
354 ..... text = buildTextSection(text, movq("(%rsp)", "%rax"));
355 ..... text = buildTextSection(text, add("$8", "%rsp"));
356 ..... text = buildTextSection(text, add("%rdx", "%rax"));
357 ..... text = buildTextSection(text, sub("$8", "%rsp"));
358 ..... text = buildTextSection(text, movq("%rax",("(%rsp)"));
359 .....
360 ..... break;
361 ..... case '-':
362 .....     gen_asm(a->r);
363 .....     gen_asm(a->l);
364 .....
365 ..... //printf("\n# subtraction\n");
366 .....
367 ..... text = buildTextSection(text, movq("(%rsp)", "%rax"));
368 ..... text = buildTextSection(text, add("$8", "%rsp"));
369 ..... text = buildTextSection(text, sub("(%rsp)", "%rax"));
370 ..... text = buildTextSection(text, movq("%rax",("(%rsp)"));

```

```

371
372 ..... break;
373 ..... case '*':
374 .....     gen_asm(a->r);
375 .....     gen_asm(a->l);
376
377 .....     //printf("\n# multiplication\n");
378
379 .....     text = buildTextSection(text, movq("(%rsp)", "%rax"));
380 .....     text = buildTextSection(text, add("$8", "%rsp"));
381 .....     text = buildTextSection(text, imul("(%rsp)", "%rax"));
382 .....     text = buildTextSection(text, movq("%rax", "%rsp"));
383
384 .....     break;
385 ..... case '/':
386 .....     gen_asm(a->r);
387 .....     gen_asm(a->l);
388
389 .....     //printf("\n# division\n");
390
391 .....     text = buildTextSection(text, movq("(%rsp)", "%rax"));
392 .....     text = buildTextSection(text, add("$8", "%rsp"));
393 .....     text = buildTextSection(text, cqto());
394 .....     text = buildTextSection(text, idiv("(%rsp)"));
395 .....     text = buildTextSection(text, movq("%rax", "%rsp"));
396
397 .....     break;
398 ..... case '%':
399 .....     gen_asm(a->r);
400 .....     gen_asm(a->l);
401
402 .....     //printf("\n# modulation\n");
403
404 .....     text = buildTextSection(text, movq("(%rsp)", "%rax"));
405 .....     text = buildTextSection(text, add("$8", "%rsp"));
406 .....     text = buildTextSection(text, cqto());
407 .....     text = buildTextSection(text, idiv("(%rsp)"));
408 .....     text = buildTextSection(text, movq("%rdx", "%rsp"));
409
410 .....     break;
411 ..... case 'M':
412 .....     gen_asm(a->l);
413
414 .....     //printf("\n# Negation\n");
415
416 .....     text = buildTextSection(text, movq("(%rsp)", "%rax"));
417 .....     text = buildTextSection(text, neg("%rax"));
418 .....     text = buildTextSection(text, movq("%rax", "%rsp"));
419
420 .....     break;
421
422 .....     /* control flow */
423 .....     /* null expressions allowed in the grammar, so check for them */
424
425 .....     /* condition */
426 .....     case 'C':
427 .....         gen_asm(((struct cond*)a)->cond);
428
429 .....         //printf("\n# Condition\n");
430

```

```

430
431 ..... text = buildTextSection(text, compare("$0", "(%rsp)"));
432 ..... sprintf(buf, "L%u", branchCount);
433 ..... text = buildTextSection(text, buildJump("je", buf));
434
435 ..... if (((struct cond*)a)->t1)
436 ..... gen_asm(((struct cond*)a)->t1);
437
438 ..... sprintf(buf, "L%u", branchCount++);
439 ..... text = buildTextSection(text, buildLabel(buf));
440
441 ..... break;
442
443 ..... /* loop */
444 ..... case 'L':
445 ..... if (((struct loop*)a)->t1) {
446 ..... //printf("\n# Loop\n");
447
448 ..... sprintf(buf, "$%ld", eval(((struct loop*)a)->from));
449 ..... text = buildTextSection(text, movl(buf, "%eax"));
450 ..... text = buildTextSection(text, movq("%rax", "from(%rip)"));
451 ..... sprintf(buf, "L%u", branchCount);
452 ..... text = buildTextSection(text, buildJump("jmp", buf));
453 ..... sprintf(buf, "L%u", ++branchCount);
454 ..... text = buildTextSection(text, buildLabel(buf));
455
456 ..... gen_asm(((struct loop*)a)->t1);
457
458 ..... text = buildTextSection(text, movq("from(%rip)", "%rdx"));
459 ..... sprintf(buf, "$%ld", eval(((struct loop*)a)->inc));
460 ..... text = buildTextSection(text, movl(buf, "%eax"));
461 ..... text = buildTextSection(text, add("%rdx", "%rax"));
462 ..... text = buildTextSection(text, movq("%rax", "from(%rip)"));
463 ..... sprintf(buf, "L%u", (branchCount - 1));
464 ..... text = buildTextSection(text, buildLabel(buf));
465 ..... sprintf(buf, "$%ld", eval(((struct loop*)a)->to));
466 ..... text = buildTextSection(text, movl(buf, "%eax"));
467 ..... text = buildTextSection(text, compare("%rax", "from(%rip)"));
468 ..... sprintf(buf, "L%u", branchCount);
469 ..... text = buildTextSection(text, buildJump("jle", buf));
470
471 ..... ++branchCount;
472 ..... }
473
474 ..... break;
475
476 ..... /* list of statements */
477 ..... case 'E':
478 ..... gen_asm(a->l);
479 ..... gen_asm(a->r);
480 ..... break;
481
482 ..... default:
483 ..... printf("internal error: bad node %c\n", a->nodetype);
484 ..... }
485 }
486

```

```

487 int64_t eval (struct ast *a)
488 {
489     int64_t v = 0;
490
491     if (!a) {
492         yyerror("internal error, null eval");
493     }
494     return 0;
495 }
496
497 switch (a->nodetype) {
498     /* constant */
499     case 'K':
500         v = ((struct numval *)a)->number;
501         break;
502
503     /* symbol reference */
504     case 'V':
505         v = ((struct symref *)a)->s->value;
506         break;
507
508     /* print expression */
509     case 'A':
510         v = eval(((struct print *)a)->arg.exp);
511         printf("%ld\n", v);
512         break;
513
514     /* print literal string */
515     case 'S':
516         printf("%s\n", (((struct print *)a)->arg.ls->str));
517         break;
518
519     /* assignment */
520     case '=':
521         v = ((struct symasn *)a)->s->value = eval(((struct symasn *)a)->v);
522         break;
523
524     /* expressions */
525     case '+':
526         v = eval(a->l) + eval(a->r);
527         break;
528     case '-':
529         v = eval(a->l) - eval(a->r);
530         break;
531     case '*':
532         v = eval(a->l) * eval(a->r);
533         break;
534     case '/':
535         v = eval(a->l) / eval(a->r);
536         break;
537     case '%':
538         v = eval(a->l) % eval(a->r);
539         break;

```

```

540 ..... case 'M':
541 .....     v = -eval(a->l);
542 .....     break;
543 .....
544 .....     /* control flow */
545 .....     /* null expressions allowed in the grammar, so check for them */
546 .....
547 .....     /* condition */
548 .....     case 'C':
549 .....         if (eval(((struct cond*)a)->cond) != 0) {
550 .....             if (((struct cond*)a)->t1)
551 .....                 v = eval(((struct cond*)a)->t1);
552 .....             }
553 .....             break;
554 .....
555 .....     /* loop */
556 .....     case 'L':
557 .....         if (((struct loop*)a)->t1) {
558 .....             int64_t count = eval(((struct loop*)a)->from);
559 .....             int64_t to = eval(((struct loop*)a)->to);
560 .....             int64_t inc = eval(((struct loop*)a)->inc);
561 .....
562 .....             while (count <= to) {
563 .....                 v = eval(((struct loop*)a)->t1);
564 .....                 count += inc;
565 .....             }
566 .....         }
567 .....
568 .....     break;
569 .....
570 .....     /* list of statements */
571 .....     case 'E':
572 .....         eval(a->l);
573 .....         v = eval(a->r);
574 .....         break;
575 .....
576 .....     default:
577 .....         printf("internal error: bad node %c\n", a->nodetype);
578 .....     }
579 .....
580 ..... return v;
581 }
582

```

```

583 void yyerror(char *s, ...)
584 {
585     va_list ap;
586     va_start(ap, s);
587
588     fprintf(stderr, "%d: error: ", yylineno);
589     vfprintf(stderr, s, ap);
590     fprintf(stderr, "\n");
591 }
592
593 int main(int argc, char **argv)
594 {
595     extern FILE *yyin;
596     char *asmFile;
597     char *ptr;
598     char buf[21];
599
600     if (argc > 1) {
601         if (!(yyin = fopen(argv[1], "r"))) {
602             perror(argv[1]);
603             return 1;
604         }
605     }
606     else {
607         printf("\nPlease specify a source file.\n\n");
608         return 1;
609     }
610
611     text = (char *) malloc(sizeof(char));
612     data = (char *) malloc(sizeof(char));
613     *(text) = '\0';
614     *(data) = '\0';
615
616     asmFile = strdup(argv[1]);
617     ptr = strchr(asmFile, '.');
618
619     *(ptr+1) = 's';
620     *(ptr+2) = '\0';
621
622     fp = createFile(asmFile);
623
624     printf("Start Parsing ... \n");
625     printf("... \n");
626     printf("... \n");
627
628     if (yyparse()) {
629         printf("\nParsing Error\n");
630         closeFile(fp);
631
632         return -1;
633     }
634
635     sprintf(buf, "$%u", (symCount+1)*8);
636     text = buildTextSection(sub(buf, "%rsp"), text);
637
638     putBssSection(fp, "");
639     putDataSection(fp, data);
640     putTextSection(fp, text);
641     closeFile(fp);
642
643     printf("Finished Parsing\n");
644
645     return 0;
646 }
647

```

อธิบายหลักการและส่วนหลักๆของไฟล์ elecompiler.c

- (1) struct symbol* lookup (char *sym)
ใช้ในการ lookup symbol table ที่ implement แบบ hash table ถ้าไม่เจอจะทำการ add symbol ใหม่เข้าไป
- (2) struct ast* newast (int nodetype, struct ast *l, struct ast *r)
สร้าง node ใน AST ตัวใหม่ขึ้นมา โดยมี child สองตัวคือ left (l) และ right (r)
- (3) struct ast* newnum (int64_t num)
สร้าง node ใน AST ตัวใหม่ขึ้นมา ใช้เก็บค่าคงที่ (literal)
- (4) struct ast* newprint (int nodetype, struct ast *exp, char *str)
สร้าง node ใน AST ตัวใหม่ขึ้นมา ใช้ในการเก็บ print() statement โดย node ชนิดนี้จะมีอยู่สองแบบ คือ 1. ใช้สำหรับ print literal string และ 2. ใช้สำหรับ print ค่าตัวแปร
- (5) struct ast* newref (struct symbol *s)
สร้าง node ใน AST ตัวใหม่ขึ้นมา ใช้ในการอ้างถึงตัวแปร (symbol)
- (6) struct ast* newasgn (struct symbol *s, struct ast *v)
สร้าง node ใน AST ตัวใหม่ขึ้นมา ใช้เก็บ assignment statement symbol (s) และ expression (v)
- (7) struct ast* newcond (struct ast *cond, struct ast *tl)
สร้าง node ใน AST ตัวใหม่ขึ้นมา ใช้เก็บ if-then statement โดยมี child เป็น condition และ then (tl)
- (8) struct ast* newloop (struct ast *from, struct ast *to, struct ast *inc, struct ast *tl)
สร้าง node ใน AST ตัวใหม่ขึ้นมา ใช้เก็บ loop statement โดยมี child สามตัว คือ 1. เลขตั้งต้น (from)
2. เลขสุดท้าย (to) และ 3. จำนวนที่จะใช้บวกเข้าไปกับเลขตั้งต้น (increment)
- (9) void treefree (struct ast *a)
ใช้ในการ free AST ที่ได้สร้างขึ้นมาในขั้นตอน parsing
- (10) void gen_asm (struct ast *a)
ใช้ในการ walk through AST และสร้าง file assembly
- (11) int64_t eval (struct ast *a)
ใช้ evaluate tree ในที่นี้หมายถึง AST
- (12) int main(int argc, char **argv)
Function หลักที่ใช้ในการทำงานของ program

4. ผลการรัน

4.1 ผลการรันกับตัวอย่างที่ผิด

1. `1 + "test"`

Error เนื่องจากไม่สามารถ + ค่า int กับ string ได้

2. `loop (1->10 , a){ I = 1; }`

Error เนื่องจากตำแหน่งของ a ใน gramma เป็น Constant ไม่ใช่ Variable

3. `"test" = 3`

Error เนื่องจาก ไม่สามารถ assign ค่าให้ variable string ได้

4. `If ("x" == "x")`

Error เนื่องจาก ไม่สามารถ check condition ระหว่าง string ได้

5. `Print if(x =8)`

Error เนื่องจาก ไม่สามารถ print statement ได้

4.2 ผลการรันกับตัวอย่างที่ถูก

1. $a = (1+2)*2$;

Assembly Code

```

7 from:
8     .zero    8
9     .section  .rodata
10  .LC0:
11     .string  "%ld"
12  .LC1:
13     .string  "%s"
14     .text
15     .globl  main
16     .type   main, @function
17 main:
18  .LFB0:
19     .cfi_startproc
20     pushq   %rbp
21     .cfi_def_cfa_offset 16
22     .cfi_offset 6, -16
23     movq    %rsp, %rbp
24     .cfi_def_cfa_register 6
25     subq    $16, %rsp
26     subq    $8, %rsp
27     movl    $2, %eax
28     movq    %rax, (%rsp)
29     subq    $8, %rsp
30     movl    $2, %eax
31     movq    %rax, (%rsp)
32     subq    $8, %rsp
33     movl    $1, %eax
34     movq    %rax, (%rsp)
35     movq    (%rsp), %rdx
36     addq    $8, %rsp
37     movq    (%rsp), %rax
38     addq    $8, %rsp
39     addq    %rdx, %rax
40     subq    $8, %rsp
41     movq    %rax, (%rsp)
42     movq    (%rsp), %rax
43     addq    $8, %rsp
44     imulq   (%rsp), %rax
45     movq    %rax, (%rsp)
46     movq    (%rsp), %rax
47     addq    $8, %rsp
48     movq    %rax, -8(%rbp)
49     subq    $8, %rsp
50     movq    %rax, (%rsp)
51     movl    $0, %eax
52     leave
53     .cfi_def_cfa 7, 8
54     ret
55     .cfi_endproc
56  .LFE0:
57     .size   main, .-main
58     .ident  "GCC: (GNU) 6.1.1 20160501"
59     .section .note.GNU-stack,"",@progbits

```

2. $a = 1 \cdot 2 + 3$;

$a = a \cdot a$;

Assembly Code

```

1      .file    "test.c"
2      .globl   from
3      .bss
4      .align   8
5      .type    from, @object
6      .size    from, 8
7 from:
8      .zero    8
9      .section  .rodata
10     .LC0:
11     .string   "%ld"
12     .LC1:
13     .string   "%s"
14     .text
15     .globl   main
16     .type    main, @function
17 main:
18     .LFB0:
19     .cfi_startproc
20     pushq    %rbp
21     .cfi_def_cfa_offset 16
22     .cfi_offset 6, -16
23     movq     %rsp, %rbp
24     .cfi_def_cfa_register 6
25     subq     $16, %rsp
26     subq     $8, %rsp
27     movl     $3, %eax
28     movq     %rax, (%rsp)
29     subq     $8, %rsp
30     movl     $2, %eax
31     movq     %rax, (%rsp)
32     subq     $8, %rsp
33     movl     $1, %eax
34     movq     %rax, (%rsp)
35     movq     (%rsp), %rax
36     addq     $8, %rsp
37     imulq    (%rsp), %rax
38     movq     %rax, (%rsp)
39     movq     (%rsp), %rdx
40     addq     $8, %rsp
41     movq     (%rsp), %rax
42     addq     $8, %rsp
43     addq     %rdx, %rax
44     subq     $8, %rsp
45     movq     %rax, (%rsp)
46     movq     (%rsp), %rax
47     addq     $8, %rsp
48     movq     %rax, -8(%rbp)
49     subq     $8, %rsp

```

```

50     movq    %rax, (%rsp)
51     movq    -8(%rbp), %rax
52     subq    $8, %rsp
53     movq    %rax, (%rsp)
54     movq    -8(%rbp), %rax
55     subq    $8, %rsp
56     movq    %rax, (%rsp)
57     movq    (%rsp), %rax
58     addq    $8, %rsp
59     imulq   (%rsp), %rax
60     movq    %rax, (%rsp)
61     movq    (%rsp), %rax
62     addq    $8, %rsp
63     movq    %rax, -8(%rbp)
64     subq    $8, %rsp
65     movq    %rax, (%rsp)
66     movl    $0, %eax
67     leave
68     .cfi_def_cfa 7, 8
69     ret
70     .cfi_endproc
71 .LFE0:
72     .size    main, .-main
73     .ident   "GCC: (GNU) 6.1.1 20160501"
74     .section .note.GNU-stack,"",@progbits

```

```
3. a = (1*2*3)%4;
```

```
    print (a*3);
```

Assembly Code

```

1      .file      "test.c"
2      .globl    from
3      .bss
4      .align    8
5      .type     from, @object
6      .size     from, 8
7 from:
8      .zero     8
9      .section   .rodata
10     .LC0:
11     .string    "%ld"
12     .LC1:
13     .string    "%s"
14     .text
15     .globl    main
16     .type     main, @function
17 main:
18     .LFB0:
19     .cfi_startproc
20     pushq     %rbp
21     .cfi_def_cfa_offset 16
22     .cfi_offset 6, -16
23     movq     %rsp, %rbp
24     .cfi_def_cfa_register 6
25     subq     $16, %rsp
26     subq     $8, %rsp
27     movl     $4, %eax
28     movq     %rax, (%rsp)
29     subq     $8, %rsp
30     movl     $3, %eax
31     movq     %rax, (%rsp)
32     subq     $8, %rsp
33     movl     $2, %eax
34     movq     %rax, (%rsp)
35     subq     $8, %rsp
36     movl     $1, %eax
37     movq     %rax, (%rsp)
38     movq     (%rsp), %rax
39     addq     $8, %rsp
40     imulq    (%rsp), %rax
41     movq     %rax, (%rsp)
42     movq     (%rsp), %rax
43     addq     $8, %rsp
44     imulq    (%rsp), %rax
45     movq     %rax, (%rsp)
46     movq     (%rsp), %rax
47     addq     $8, %rsp
48     cqto
49     idivq    (%rsp)

```

```

50     movq    %rdx, (%rsp)
51     movq    (%rsp), %rax
52     addq    $8, %rsp
53     movq    %rax, -8(%rbp)
54     subq    $8, %rsp
55     movq    %rax, (%rsp)
56     subq    $8, %rsp
57     movl    $3, %eax
58     movq    %rax, (%rsp)
59     movq    -8(%rbp), %rax
60     subq    $8, %rsp
61     movq    %rax, (%rsp)
62     movq    (%rsp), %rax
63     addq    $8, %rsp
64     imulq   (%rsp), %rax
65     movq    %rax, (%rsp)
66     movq    (%rsp), %rax
67     addq    $8, %rsp
68     movq    %rax, %rsi
69     movl    $.LC0, %edi
70     movl    $0, %eax
71     call    printf
72     movl    $0, %eax
73     leave
74     .cfi_def_cfa 7, 8
75     ret
76     .cfi_endproc
77 .LFE0:
78     .size    main, .-main
79     .ident   "GCC: (GNU) 6.1.1 20160501"
80     .section .note.GNU-stack,"",@progbits

```

Output

```

kohpai@Archie sample $ ./t3
6kohpai@Archie sample $ ./t3
6kohpai@Archie sample $ ./t3
6kohpai@Archie sample $ ./t3
6kohpai@Archie sample $ █

```

```

4. a = -4;

    if(-a){

        print (a);\

    }

    print ("a >v 0");

```

Assembly Code

```

1      .file      "test.c"
2      .globl     from
3      .bss
4      .align     8
5      .type      from, @object
6      .size      from, 8
7 from:
8      .zero      8
9      .section    .rodata
10     .LC0:
11     .string     "%ld"
12     .LC1:
13     .string     "%s"
14     .LC2:
15     .string     "a >v 0"
16     .text
17     .globl     main
18     .type      main, @function
19 main:
20     .LFB0:
21     .cfi_startproc
22     pushq      %rbp
23     .cfi_def_cfa_offset 16
24     .cfi_offset 6, -16
25     movq       %rsp, %rbp
26     .cfi_def_cfa_register 6
27     subq       $16, %rsp
28     subq       $8, %rsp
29     movl       $4, %eax
30     movq       %rax, (%rsp)
31     movq       (%rsp), %rax
32     negq       %rax
33     movq       %rax, (%rsp)
34     movq       (%rsp), %rax
35     addq       $8, %rsp
36     movq       %rax, -8(%rbp)
37     subq       $8, %rsp
38     movq       %rax, (%rsp)
39     movq       -8(%rbp), %rax
40     subq       $8, %rsp
41     movq       %rax, (%rsp)
42     movq       (%rsp), %rax
43     negq       %rax

```

```

44     movq    %rax, (%rsp)
45     cmpq    $0, (%rsp)
46     je      .L0
47     movq    -8(%rbp), %rax
48     subq    $8, %rsp
49     movq    %rax, (%rsp)
50     movq    (%rsp), %rax
51     addq    $8, %rsp
52     movq    %rax, %rsi
53     movl    $.LC0, %edi
54     movl    $0, %eax
55     call    printf
56 .L0:
57     movl    $.LC2, %esi
58     movl    $.LC1, %edi
59     movl    $0, %eax
60     call    printf
61     movl    $0, %eax
62     leave
63     .cfi_def_cfa 7, 8
64     ret
65     .cfi_endproc
66 .LFE0:
67     .size   main, .-main
68     .ident  "GCC: (GNU) 6.1.1 20160501"
69     .section .note.GNU-stack,"",@progbits

```

Output

```

-4a >v 0kohpai@Archie sample $ ./t4
-4a >v 0kohpai@Archie sample $ ./t4
-4a >v 0kohpai@Archie sample $ ./t4
-4a >v 0kohpai@Archie sample $ █

```



```

5. a = -100000;

    if(a){

        print ("a < 0");\

    }

    if(-a){

    }

    print ("a > 0");

```

Assembly Code

```

1      .file      "test.c"
2      .globl    from
3      .bss
4      .align    8
5      .type     from, @object
6      .size     from, 8
7  from:
8      .zero     8
9      .section   .rodata
10     .LC0:
11     .string    "%ld"
12     .LC1:
13     .string    "%s"
14     .LC2:
15     .string    "a < 0"
16     .LC3:
17     .string    "a > 0"
18     .text
19     .globl    main
20     .type     main, @function
21  main:
22     .LFB0:
23     .cfi_startproc
24     pushq     %rbp
25     .cfi_def_cfa_offset 16
26     .cfi_offset 6, -16
27     movq     %rsp, %rbp
28     .cfi_def_cfa_register 6
29     subq     $16, %rsp
30     subq     $8, %rsp
31     movl     $100000, %eax
32     movq     %rax, (%rsp)
33     movq     (%rsp), %rax
34     negq     %rax
35     movq     %rax, (%rsp)
36     movq     (%rsp), %rax
37     addq     $8, %rsp
38     movq     %rax, -8(%rbp)

```

```

39     subq    $8, %rsp
40     movq    %rax, (%rsp)
41     movq    -8(%rbp), %rax
42     subq    $8, %rsp
43     movq    %rax, (%rsp)
44     cmpq    $0, (%rsp)
45     je      .L0
46     movl    $.LC2, %esi
47     movl    $.LC1, %edi
48     movl    $0, %eax
49     call    printf
50 .L0:
51     movq    -8(%rbp), %rax
52     subq    $8, %rsp
53     movq    %rax, (%rsp)
54     movq    (%rsp), %rax
55     negq    %rax
56     movq    %rax, (%rsp)
57     cmpq    $0, (%rsp)
58     je      .L1
59 .L1:
60     movl    $.LC3, %esi
61     movl    $.LC1, %edi
62     movl    $0, %eax
63     call    printf
64     movl    $0, %eax
65     leave
66     .cfi_def_cfa 7, 8
67     ret
68     .cfi_endproc
69 .LFE0:
70     .size   main, .-main
71     .ident   "GCC: (GNU) 6.1.1 20160501"
72     .section .note.GNU-stack,"",@progbits

```

Output

```

kohpai@Archie sample $ ./t5
a < 0a > 0kohpai@Archie sample $ ./t5
a < 0a > 0kohpai@Archie sample $ ./t5
a < 0a > 0kohpai@Archie sample $ 

```

```

6. i = 1;
   loop (0->10,1){
       i = i+1;\
       print (i*3);\
   }

```

Assembly Code

```

1  .file "test.c"
2  .globl from
3  .bss
4  .align 8
5  .type from, @object
6  .size from, 8
7  from:
8  .zero 8
9  .section .rodata
10 .LC0:
11 .string "%ld"
12 .LC1:
13 .string "%s"
14 .text
15 .globl main
16 .type main, @function
17 main:
18 .LFB0:
19 .cfi_startproc
20 pushq %rbp
21 .cfi_def_cfa_offset 16
22 .cfi_offset 6, -16
23 movq %rsp, %rbp
24 .cfi_def_cfa_register 6
25 subq $16, %rsp
26 subq $8, %rsp
27 movl $1, %eax
28 movq %rax, (%rsp)
29 movq (%rsp), %rax
30 addq $8, %rsp
31 movq %rax, -8(%rbp)
32 subq $8, %rsp
33 movq %rax, (%rsp)
34 movl $0, %eax
35 movq %rax, from(%rip)
36 jmp .L0
37 .L1:
38 subq $8, %rsp
39 movl $1, %eax
40 movq %rax, (%rsp)
41 movq -8(%rbp), %rax
42 subq $8, %rsp
43 movq %rax, (%rsp)
44 movq (%rsp), %rdx

```

```

45     addq    $8, %rsp
46     movq    (%rsp), %rax
47     addq    $8, %rsp
48     addq    %rdx, %rax
49     subq    $8, %rsp
50     movq    %rax, (%rsp)
51     movq    (%rsp), %rax
52     addq    $8, %rsp
53     movq    %rax, -8(%rbp)
54     subq    $8, %rsp
55     movq    %rax, (%rsp)
56     subq    $8, %rsp
57     movl    $3, %eax
58     movq    %rax, (%rsp)
59     movq    -8(%rbp), %rax
60     subq    $8, %rsp
61     movq    %rax, (%rsp)
62     movq    (%rsp), %rax
63     addq    $8, %rsp
64     imulq    (%rsp), %rax
65     movq    %rax, (%rsp)
66     movq    (%rsp), %rax
67     addq    $8, %rsp
68     movq    %rax, %rsi
69     movl    $.LC0, %edi
70     movl    $0, %eax
71     call    printf
72     movq    from(%rip), %rdx
73     movl    $1, %eax
74     addq    %rdx, %rax
75     movq    %rax, from(%rip)
76 .L0:
77     movl    $10, %eax
78     cmpq    %rax, from(%rip)
79     jle .L1
80     movl    $0, %eax
81     leave
82     .cfi_def_cfa 7, 8
83     ret
84     .cfi_endproc
85 .LFE0:
86     .size   main, .-main
87     .ident  "GCC: (GNU) 6.1.1 20160501"
88     .section .note.GNU-stack,"",@progbits

```

Output

```

kohpai@Archie sample $ ./t6 -
69121518212427303336kohpai@Archie sample $ ./t6
69121518212427303336kohpai@Archie sample $ ./t6
69121518212427303336kohpai@Archie sample $ ./t6
69121518212427303336kohpai@Archie sample $ ./t6
69121518212427303336kohpai@Archie sample $ █

```

```

7. i = 1;

j = 0;

loop (0->10,1){

    i = i+1;\

    j=i*i;\

    print (i*3);\

}

if(i-100){

    print ("a > 100");\

}

```

Assembly Code

```

1      .file      "test.c"
2      .globl    from
3      .bss
4      .align    8
5      .type     from, @object
6      .size     from, 8
7 from:
8      .zero     8
9      .section   .rodata
10 .LC0:
11     .string    "%ld"
12 .LC1:
13     .string    "%s"
14 .LC2:
15     .string    "a > 100"
16     .text
17     .globl    main
18     .type     main, @function
19 main:
20 .LFB0:
21     .cfi_startproc
22     pushq     %rbp
23     .cfi_def_cfa_offset 16
24     .cfi_offset 6, -16
25     movq     %rsp, %rbp
26     .cfi_def_cfa_register 6
27     subq     $24, %rsp
28     subq     $8, %rsp
29     movl     $1, %eax
30     movq     %rax, (%rsp)
31     movq     (%rsp), %rax
32     addq     $8, %rsp

```

```

33     movq    %rax, -8(%rbp)
34     subq    $8, %rsp
35     movq    %rax, (%rsp)
36     subq    $8, %rsp
37     movl    $0, %eax
38     movq    %rax, (%rsp)
39     movq    (%rsp), %rax
40     addq    $8, %rsp
41     movq    %rax, -16(%rbp)
42     subq    $8, %rsp
43     movq    %rax, (%rsp)
44     movl    $0, %eax
45     movq    %rax, from(%rip)
46     jmp     .L0
47 .L1:
48     subq    $8, %rsp
49     movl    $1, %eax
50     movq    %rax, (%rsp)
51     movq    -8(%rbp), %rax
52     subq    $8, %rsp
53     movq    %rax, (%rsp)
54     movq    (%rsp), %rdx
55     addq    $8, %rsp
56     movq    (%rsp), %rax
57     addq    $8, %rsp
58     addq    %rdx, %rax
59     subq    $8, %rsp
60     movq    %rax, (%rsp)
61     movq    (%rsp), %rax
62     addq    $8, %rsp
63     movq    %rax, -8(%rbp)
64     subq    $8, %rsp
65     movq    %rax, (%rsp)
66     movq    -8(%rbp), %rax
67     subq    $8, %rsp
68     movq    %rax, (%rsp)
69     movq    -8(%rbp), %rax
70     subq    $8, %rsp
71     movq    %rax, (%rsp)
72     movq    (%rsp), %rax
73     addq    $8, %rsp
74     imulq   (%rsp), %rax
75     movq    %rax, (%rsp)
76     movq    (%rsp), %rax
77     addq    $8, %rsp
78     movq    %rax, -16(%rbp)
79     subq    $8, %rsp
80     movq    %rax, (%rsp)
81     subq    $8, %rsp
82     movl    $3, %eax
83     movq    %rax, (%rsp)
84     movq    -8(%rbp), %rax
85     subq    $8, %rsp
86     movq    %rax, (%rsp)
87     movq    (%rsp), %rax
88     addq    $8, %rsp
89     imulq   (%rsp), %rax

```

```

90     movq    %rax, (%rsp)
91     movq    (%rsp), %rax
92     addq    $8, %rsp
93     movq    %rax, %rsi
94     movl    $.LC0, %edi
95     movl    $0, %eax
96     call    printf
97     movq    from(%rip), %rdx
98     movl    $1, %eax
99     addq    %rdx, %rax
100    movq    %rax, from(%rip)
101    .L0:
102        movl    $10, %eax
103        cmpq    %rax, from(%rip)
104        jle    .L1
105        subq    $8, %rsp
106        movl    $100, %eax
107        movq    %rax, (%rsp)
108        movq    -8(%rbp), %rax
109        subq    $8, %rsp
110        movq    %rax, (%rsp)
111        movq    (%rsp), %rax
112        addq    $8, %rsp
113        subq    (%rsp), %rax
114        movq    %rax, (%rsp)
115        cmpq    $0, (%rsp)
116        je     .L2
117        movl    $.LC2, %esi
118        movl    $.LC1, %edi
119        movl    $0, %eax
120        call    printf
121    .L2:
122        movl    $0, %eax
123        leave
124        .cfi_def_cfa 7, 8
125        ret
126        .cfi_endproc
127    .LFE0:
128        .size    main, .-main
129        .ident    "GCC: (GNU) 6.1.1 20160501"
130        .section    .note.GNU-stack,"",@progbits

```

Output

```

kohpai@Archie sample $ ./t7
69121518212427303336a > 100kohpai@Archie sample $ ./t7
69121518212427303336a > 100kohpai@Archie sample $ ./t7
69121518212427303336a > 100kohpai@Archie sample $ ./t7
69121518212427303336a > 100kohpai@Archie sample $ █

```

```

8.i = 1;

loop (0->10,1){\

    i = i+1;\

    print (i*3);\

    print("\n");\

}

```

Assembly Code

```

1      .file      "test.c"
2      .globl    from
3      .bss
4      .align    8
5      .type     from, @object
6      .size     from, 8
7 from:
8      .zero     8
9      .section   .rodata
10 .LC0:
11     .string    "%ld"
12 .LC1:
13     .string    "%s"
14 .LC2:
15     .string    "\n"
16     .text
17     .globl    main
18     .type     main, @function
19 main:
20 .LFB0:
21     .cfi_startproc
22     pushq     %rbp
23     .cfi_def_cfa_offset 16
24     .cfi_offset 6, -16
25     movq     %rsp, %rbp
26     .cfi_def_cfa_register 6
27     subq     $16, %rsp
28     subq     $8, %rsp
29     movl     $1, %eax
30     movq     %rax, (%rsp)
31     movq     (%rsp), %rax
32     addq     $8, %rsp
33     movq     %rax, -8(%rbp)
34     subq     $8, %rsp
35     movq     %rax, (%rsp)
36     movl     $0, %eax
37     movq     %rax, from(%rip)
38     jmp     .L0
39 .L1:

```



```

40     subq    $8, %rsp
41     movl    $1, %eax
42     movq    %rax, (%rsp)
43     movq    -8(%rbp), %rax
44     subq    $8, %rsp
45     movq    %rax, (%rsp)
46     movq    (%rsp), %rdx
47     addq    $8, %rsp
48     movq    (%rsp), %rax
49     addq    $8, %rsp
50     addq    %rdx, %rax
51     subq    $8, %rsp
52     movq    %rax, (%rsp)
53     movq    (%rsp), %rax
54     addq    $8, %rsp
55     movq    %rax, -8(%rbp)
56     subq    $8, %rsp
57     movq    %rax, (%rsp)
58     movl    $.LC2, %esi
59     movl    $.LC1, %edi
60     movl    $0, %eax
61     call    printf
62     movq    from(%rip), %rdx
63     movl    $1, %eax
64     addq    %rdx, %rax
65     movq    %rax, from(%rip)
66 .L0:
67     movl    $10, %eax
68     cmpq    %rax, from(%rip)
69     jle .L1
70     movl    $0, %eax
71     leave
72     .cfi_def_cfa 7, 8
73     ret
74     .cfi_endproc
75 .LFE0:
76     .size   main, .-main
77     .ident  "GCC: (GNU) 6.1.1 20160501"
78     .section .note.GNU-stack,"",@progbits

```

Output

```

kohpai@Archie sample $ ./t8
6
9
12
15
18
21
24
27
30
33
36
kohpai@Archie sample $ █

```

```

9.i = 1;

j = i+1;

print ("i+j : ");

print(i+j);

print("\n");

```

Assembly Code

```

1      .file      "test.c"
2      .globl    from
3      .bss
4      .align    8
5      .type     from, @object
6      .size     from, 8
7 from:
8      .zero     8
9      .section   .rodata
10 .LC0:
11     .string    "%ld"
12 .LC1:
13     .string    "%s"
14 .LC2:
15     .string    "i+j : "
16 .LC3:
17     .string    "\n"
18     .text
19     .globl    main
20     .type     main, @function
21 main:
22 .LFB0:
23     .cfi_startproc
24     pushq     %rbp
25     .cfi_def_cfa_offset 16
26     .cfi_offset 6, -16
27     movq     %rsp, %rbp
28     .cfi_def_cfa_register 6
29     subq     $24, %rsp
30     subq     $8, %rsp
31     movl     $1, %eax
32     movq     %rax, (%rsp)
33     movq     (%rsp), %rax
34     addq     $8, %rsp
35     movq     %rax, -8(%rbp)
36     subq     $8, %rsp
37     movq     %rax, (%rsp)
38     subq     $8, %rsp
39     movl     $1, %eax
40     movq     %rax, (%rsp)
41     movq     -8(%rbp), %rax
42     subq     $8, %rsp
43     movq     %rax, (%rsp)

```

```

44     movq    (%rsp), %rdx
45     addq    $8, %rsp
46     movq    (%rsp), %rax
47     addq    $8, %rsp
48     addq    %rdx, %rax
49     subq    $8, %rsp
50     movq    %rax, (%rsp)
51     movq    (%rsp), %rax
52     addq    $8, %rsp
53     movq    %rax, -16(%rbp)
54     subq    $8, %rsp
55     movq    %rax, (%rsp)
56     movl    $.LC2, %esi
57     movl    $.LC1, %edi
58     movl    $0, %eax
59     call    printf
60     movq    -16(%rbp), %rax
61     subq    $8, %rsp
62     movq    %rax, (%rsp)
63     movq    -8(%rbp), %rax
64     subq    $8, %rsp
65     movq    %rax, (%rsp)
66     movq    (%rsp), %rdx
67     addq    $8, %rsp
68     movq    (%rsp), %rax
69     addq    $8, %rsp
70     addq    %rdx, %rax
71     subq    $8, %rsp
72     movq    %rax, (%rsp)
73     movq    (%rsp), %rax
74     addq    $8, %rsp
75     movq    %rax, %rsi
76     movl    $.LC0, %edi
77     movl    $0, %eax
78     call    printf
79     movl    $.LC3, %esi
80     movl    $.LC1, %edi
81     movl    $0, %eax
82     call    printf
83     movl    $0, %eax
84     leave
85     .cfi_def_cfa 7, 8
86     ret
87     .cfi_endproc
88 .LFE0:
89     .size    main, .-main
90     .ident   "GCC: (GNU) 6.1.1 20160501"
91     .section .note.GNU-stack,"",@progbits

```

Output

```

kohpai@Archie sample $ ./t9
i+j : 3
kohpai@Archie sample $

```

```

10. i = 1;

    print (i);

    i = 10;

    print (i);

```

Assembly Code

```

1      .file      "test.c"
2      .globl    from
3      .bss
4      .align    8
5      .type     from, @object
6      .size     from, 8
7 from:
8      .zero     8
9      .section   .rodata
10 .LC0:
11     .string    "%ld"
12 .LC1:
13     .string    "%s"
14     .text
15     .globl    main
16     .type     main, @function
17 main:
18 .LFB0:
19     .cfi_startproc
20     pushq     %rbp
21     .cfi_def_cfa_offset 16
22     .cfi_offset 6, -16
23     movq     %rsp, %rbp
24     .cfi_def_cfa_register 6
25     subq     $16, %rsp
26     subq     $8, %rsp
27     movl     $1, %eax
28     movq     %rax, (%rsp)
29     movq     (%rsp), %rax
30     addq     $8, %rsp
31     movq     %rax, -8(%rbp)
32     subq     $8, %rsp
33     movq     %rax, (%rsp)
34     movq     -8(%rbp), %rax
35     subq     $8, %rsp
36     movq     %rax, (%rsp)
37     movq     (%rsp), %rax
38     addq     $8, %rsp
39     movq     %rax, %rsi
40     movl     $.LC0, %edi
41     movl     $0, %eax
42     call     printf
43     subq     $8, %rsp
44     movl     $10, %eax
45     movq     %rax, (%rsp)

```

```

46     movq    (%rsp), %rax
47     addq    $8, %rsp
48     movq    %rax, -8(%rbp)
49     subq    $8, %rsp
50     movq    %rax, (%rsp)
51     movq    -8(%rbp), %rax
52     subq    $8, %rsp
53     movq    %rax, (%rsp)
54     movq    (%rsp), %rax
55     addq    $8, %rsp
56     movq    %rax, %rsi
57     movl    $.LC0, %edi
58     movl    $0, %eax
59     call    printf
60     movl    $0, %eax
61     leave
62     .cfi_def_cfa 7, 8
63     ret
64     .cfi_endproc
65 .LFE0:
66     .size    main, .-main
67     .ident   "GCC: (GNU) 6.1.1 20160501"
68     .section .note.GNU-stack,"",@progbits

```

Output

```

110kohpai@Archie sample $ ./t10
110kohpai@Archie sample $ ./t10
110kohpai@Archie sample $ ./t10
110kohpai@Archie sample $ █

```