Compilador para C-

Diseño de compiladores - Doc. Víctor Manuel de la Cueva

Entrega Final

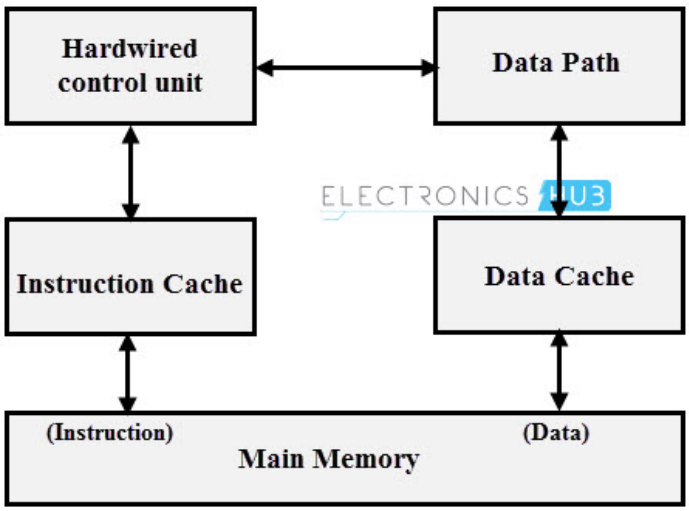
José Manuel Beauregard Méndez

A01021716

Introducción

Para la parte final de este proyecto se tuvo como reto traducir el código de C- a ensamblador con la finalidad de que la computadora pueda entender dicho código.

En clase se revisó la implementación con un procesador MIPS (Microprocessor without Interlocked Pipeline Stages) el cual es una versión reducida de un RISC.



Utilizando dicho procesador es posible nos permite completar el último paso de un compilador, generación de código. Se utilizó dicho procesador debido a que en una investigación que se realizó de otras opciones de procesadores como TM, y en dicha indagación encontré una fuente que comentaba lo común que es la incorporación de un MIPS en compiladores escolares gracias a su flexibilidad y facilidad de escribir código ensamblador.

Manual de usuarios

Utilizando dicho procesador es posible nos permite completar el último paso de un compilador, generación Un compilador está compuesto principalmente por tres partes, un analizador léxico, semántico, por último, un que recorra todo el programa identificando lo que esta pasando en cada línea. En este proyecto se abarco dichas partes, donde cada “módulo” se encarga de llevar a cabo su trabajo en el todo el proceso de compilación.

A continuación se mostrará el flujo por el cual los módulos interactúan entre sí:

Input => LEXER -> PARSER (AST) -> SEMANTICA -> Code-Generation => Output

Cabe destacar que este compilador fue producido para fines didácticos, si este compilador se quisiera llevar a un nivel profesional, antes se debería implementar mejores practicas para optimizar muchas de las tareas que se realizan. Para poder correr el programa es necesario tener la librería de ply en python3.x. Si no se tiene dicho módulo correr la siguiente línea en una terminal con python3.x:

pip3 install ply

Dicha librería es utilizada en las primeras dos partes del compilador, es decir, lexer y parser. Se escogió esta librería por su facilidad y flexibilidad al momento de escribir un compilador con todas sus partes. Este seria otro cambio importante para hacer en caso de que quisiera producir de manera profesional.

Considerando que el usuario ya tiene instalando python3.x y la librería de ply, entonces será posible correr el código de la siguiente forma:

```
from globalTypes import *
from parser import *
from semantica import *
from cgen import *

f = open('./sample.c-', 'r')
programa = f.read()     # lee todo el archivo a compilar
progLong = len(programa)   # longitud original del programa
programa = programa + '$'   # agregar un caracter $ que represente EOF
posicion = 0       # posición del caracter actual del string
# función para pasar los valores iniciales de las variables globales
globales(programa, posicion, progLong)
AST = parser(False)
semantica(AST, True)
codeGen(AST, 'file.s')
```

Este es un ejemplo de un script que fusione las cuatro partes del código. En caso de querer imprimir los resultados de cada paso es importante pasar True en el caso de parser(...) y semántica(AST, …). Supongamos que le pongamos como nombre “test.py” al script anterior, entonces así se corre en la terminal dicho código:

python3 test.py

Si en la terminal no hay ningún mensaje del compilador, entonces ya podremos encontrar un archivo llamado “file.s” donde podemos encontrar el código ensamblador generado. A continuación se mostrarán resultados posibles que el compilador pudiese arrojar:

Lexer error:

```
✘ 0jmbea0@Peps-Macbook-Pro ~/Documents/compilers/codeGeneration master ●
➜ python3 test.py
Syntax Error @ line 7:10
  k = low % 1;
```

Semantic error:

```
0jmbea0@Peps-Macbook-Pro ~/Documents/compilers/codeGeneration master ●
➜ python3 test.py
Error: Variable not declared "sort"
```

AST

```
0jmbea0@Peps-Macbook-Pro ~/Documents/compilers/codeGeneration master ●
➜ python3 test.py
|  start program
| - declaration list
| -- declaration list
| --- declaration list
| ---- declaration list
| ----- declaration
| ------ fun declaration
| ------- type specifier
| ------- int
| ------ minloc
| ------ (
| ------- params
| -------- param list
| --------- param list
| ---------- param list
| ---------- param
| ----------- type specifier
| ----------- int
| ---------- a
| ---------- [
| --------- ,
```

Symbol tables

```
{'scope': 0, 'minloc': 'int,fun,int,a,int,low,int,high', 'sort': 'void,fun,int,a,int,low,int,high', 'main': 'void,fun,void'}
{'scope': 1, 'a': 'int', 'low': 'int', 'high': 'int', 'i': 'int', 'x': 'int', 'k': 'int'}
{'scope': 2, 'a': 'int', 'low': 'int', 'high': 'int', 'i': 'int', 'k': 'int', 't': 'int'}
{'scope': 3, 'i': 'int'}
```

En las siguientes secciones se mostrarán diferentes entregables necesarios para la elaboración de un compilador.

LEXER

Diseño de compiladores - Doc. Víctor Manuel de la Cueva

Primera Entrega

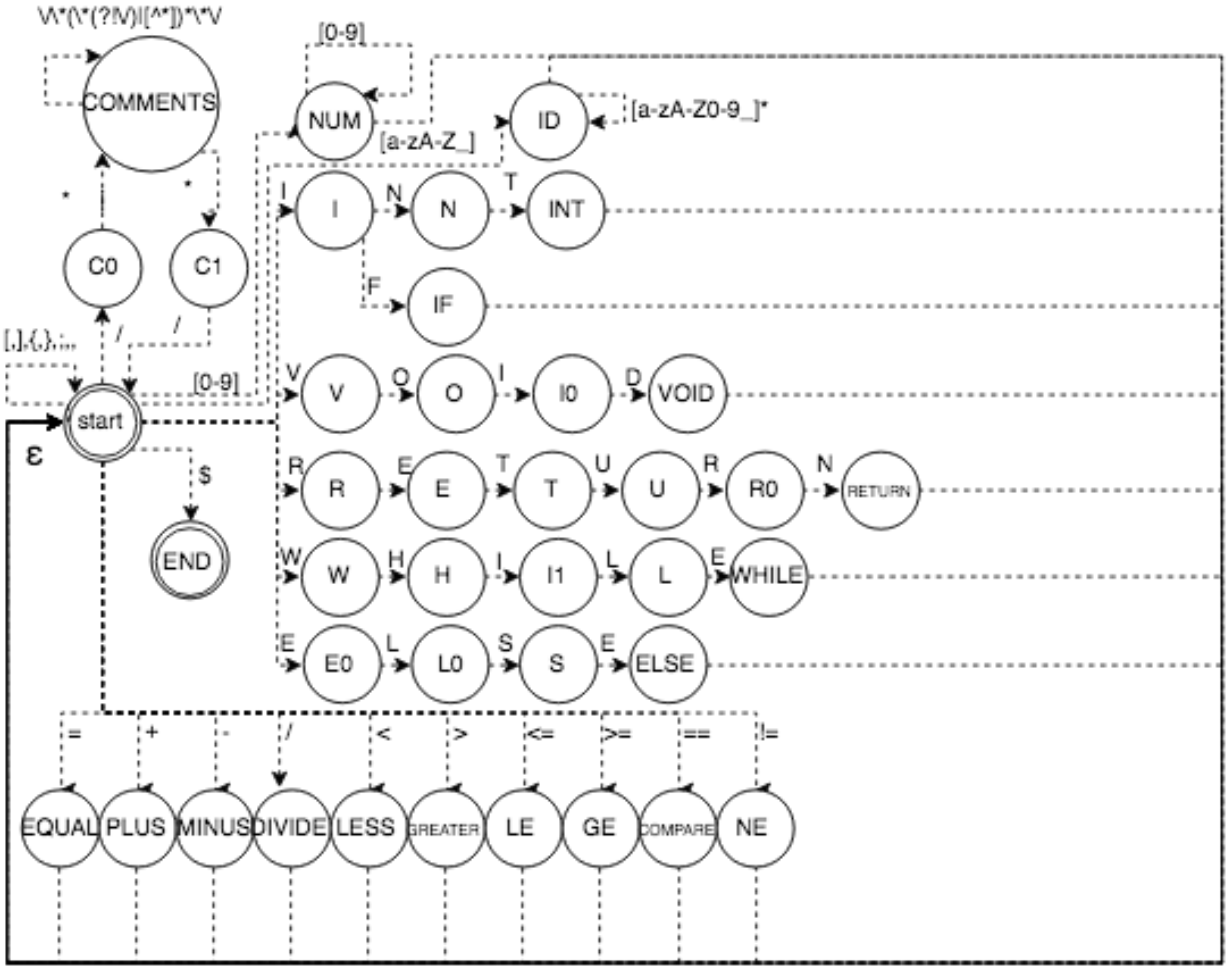
José Manuel Beauregard Méndez

A01021716

Expresiones regulares

Expresión Regular	Token
\d+	NUM
[a-zA-Z_][a-zA-Z0-9_]*	ID
\/*(*(?!\/)|[^*])**\/	COMMENT
“=“	EQUAL
\+	PLUS
-	MINUS
*	TIMES
/	DIVIDE
<	LESS
>	GREATER
;	SEMICOLOM
,	COMMA
\(	LPAREN
\)	RPAREN
\[	LBRACKET
\]	RBRACKET
\{	LBLOCK
}	RBLOCK
\$	ENDFILE

DFA


\/*(*(?!\/)|[^*])**\/
COMMENTS
*
*
C0
C1
/
/
[,],{,},;,,,
[0-9]
start
ε
$
END
[0-9]
NUM
[a-zA-Z_]
ID
[a-zA-Z0-9_]*
I
N
T
INT
F
IF
V
O
I
I0
D
VOID
R
E
T
U
R
R0
N
RETURN
W
H
I
I1
L
E
WHILE
E
L
S
E
E0
L0
S
ELSE
=
+
-
/
<
>
<=
>=
==
!=
EQUAL
PLUS
MINUS
DIVIDE
LESS
GREATER
LE
GE
COMPARE
NE

PARSER

Diseño de compiladores - Doc. Víctor Manuel de la Cueva

Segunda Entrega

José Manuel Beauregard Méndez

A01021716

Gramática

Rule 0 S' -> start

Rule 1 start -> declaration_list

Rule 2 declaration_list -> declaration_list declaration

Rule 3 declaration_list -> declaration

Rule 4 declaration -> var_declaration

Rule 5 declaration -> fun_declaration

Rule 6 declaration -> ENDFILE

Rule 7 var_declaration -> type_specifier ID SEMICOLON

Rule 8 var_declaration -> type_specifier ID LBRACKET NUMBER RBRACKET SEMICOLON

Rule 9 type_specifier -> INT

Rule 10 type_specifier -> VOID

Rule 11 fun_declaration -> type_specifier ID LPAREN params RPAREN compound_stmt

Rule 12 params -> param_list

Rule 13 params -> VOID

Rule 14 param_list -> param_list COMMA param

Rule 15 param_list -> param

Rule 16 param -> type_specifier ID

Rule 17 param -> type_specifier LBRACKET RBRACKET

Rule 18 compound_stmt -> LBLOCK local_declarations statement_list RBLOCK

Rule 19 local_declarations -> local_declarations var_declaration

Rule 20 local_declarations -> <empty>

Rule 21 statement_list -> statement_list statement

Rule 22 statement_list -> <empty>

Rule 23 statement -> expression_stmt

Rule 24 statement -> compound_stmt

Rule 25 statement -> selection_stmt

Rule 26 statement -> iteration_stmt

Rule 27 statement -> return_stmt

Rule 28 expression_stmt -> expression SEMICOLON

Rule 29 expression_stmt -> SEMICOLON

Rule 30 selection_stmt -> IF LPAREN expression RPAREN statement

Rule 31 selection_stmt -> IF LPAREN expression RPAREN statement ELSE statement

Rule 32 iteration_stmt -> WHILE LPAREN expression RPAREN statement

Rule 33 return_stmt -> RETURN SEMICOLON

Rule 34 return_stmt -> RETURN expression SEMICOLON

Rule 35 expression -> var EQUAL expression

Rule 36 expression -> simple_expression

Rule 37 var -> ID

Rule 38 var -> ID LBRACKET expression RBRACKET

Rule 39 simple_expression -> additive_expression relop additive_expression

Rule 40 simple_expression -> additive_expression

Rule 41 relop -> LE

Rule 42 relop -> LT

Rule 43 relop -> GREATER

Rule 44 relop -> LESS

Rule 45 relop -> COMPARE

Rule 46 relop -> NE

Rule 47 additive_expression -> additive_expression addop term

Rule 48 additive_expression -> term

Rule 49 addop -> PLUS

Rule 50 addop -> MINUS

Rule 51 term -> term mulop factor

Rule 52 term -> factor

Rule 53 mulop -> TIMES

Rule 54 mulop -> DIVIDE

Rule 55 factor -> LPAREN expression RPAREN

Rule 56 factor -> ID

Rule 57 factor -> call

Rule 58 factor -> NUMBER

Rule 59 call -> ID LPAREN args RPAREN

Rule 60 args -> arg_list

Rule 61 args -> <empty>

Rule 62 arg_list -> arg_list COMMA expression

Rule 63 arg_list -> expression

SEMANTICA

Diseño de compiladores - Doc. Víctor Manuel de la Cueva

Tercera Entrega

José Manuel Beauregard Méndez

A01021716

Symbol Tables

Symbols	Order
Program	0
Declaration List	x
Declaration var	2
Declaration	1
Function Declaration	2
Void	2
Compound Statement	0
Local Declarations	0
Statement List	0
Expression Statement	0
Expression	0
Var	0
Selection Statement	0
Iteration Statement	0
Return Statement	0
Simple or Additive Statement	0
Simple/Additive Statement	0
Arg List	0

Inference Rules

S' -> program
program -> declaration_list
declaration_list -> declaration_list declaration
declaration_list -> declaration
declaration -> var_declaration
declaration -> fun_declaration
var_declaration -> type_specifier ID SEMICOLON
var_declaration -> type_specifier ID LBRACK NUM RBRACK SEMICOLON type_specifier -> INT
type_specifier -> VOID
fun_declaration -> type_specifier ID LPAREN params RPAREN compound_stmt params ->
param_list
params -> VOID
param_list -> param_list COMA param
param_list -> param
param -> type_specifier ID
param -> type_specifier LBRACK RBRACK
compound_stmt -> LCURLY local_declarations statement_list RCURLY local_declarations ->
local_declarations var_declaration
local_declarations -> <empty>
statement_list -> statement_list statement
statement_list -> <empty>
statement -> expression_stmt
statement -> compound_stmt
statement -> selection_stmt
statement -> iteration_stmt
statement -> return_stmt
expression_stmt -> expression SEMICOLON
expression_stmt -> SEMICOLON
selection_stmt -> IF LPAREN expression RPAREN statement
selection_stmt -> IF LPAREN expression RPAREN statement ELSE statement iteration_stmt ->
WHILE LPAREN expression RPAREN statement return_stmt -> RETURN SEMICOLON
return_stmt -> RETURN expression SEMICOLON

expression -> var EQUALS expression
expression -> simple_expression
var -> ID
var -> ID LBRACK expression RBRACK
simple_expression -> additive_expression relop additive_expression simple_expression ->
additive_expression
relop -> LTHANEQ
relop -> LTHAN
relop -> GTHAN
relop -> GTHANEQ
relop -> EQUALTO
relop -> NOTEQUALTO
additive_expression -> additive_expression addop term
additive_expression -> term
addop -> PLUS
addop -> MINUS
term -> term mulop factor
term -> factor
mulop -> TIMES
mulop -> DIVIDE
factor -> LPAREN expression RPAREN
factor -> ID
factor -> NUM