Assignment 6 – Implementation of code generation

Date: **05/05/2023**

Aim:

To generate the output in the form of assembly code written using the instruction set of 8086.

Code:

//ex6.l

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "y.tab.h"
int yylex(void);
int yyerror(char* s);
extern YYSTYPE yylval;
int line = 0;
%}

identifier [a-zA-Z_][a-zA-Z0-9_]*

doubConst (\+|\-)?[0-9][0-9]*\.[0-9]+
intConst (\+|\-)?[0-9][0-9]*
charConst \'.\'
stringConst \"(.)*\"

op ("+"|"-"|"*"|"/"|"%")
relop "<"|">"|"<="|">="|"=="|"!="
boolop ("!"|"&&"|"||")

%%
[ \t]          {}
\n             {}

"void"         {printf(" %25s | %-25s \n", yytext, "type void"); return VOID;}
"char"           {printf(" %25s | %-25s \n", yytext, "type char"); return CHAR;}
"int"          {printf(" %25s | %-25s \n", yytext, "type int"); return INT;}
"float"          {printf(" %25s | %-25s \n", yytext, "type float"); return FLOAT;}
"double"           {printf(" %25s | %-25s \n", yytext, "type double"); return DOUBLE;}
```

```
"String"        {printf(" %25s | %-25s \n", yytext, "type string"); return STR;}

"{"             {printf(" %25s | %-25s \n", yytext, "left curly"); return LEFTCURLY;}
"}"             {printf(" %25s | %-25s \n", yytext, "right curly"); return RIGHTCURLY;}
"["             {printf(" %25s | %-25s \n", yytext, "left square"); return LEFTSQUARE;}
"]"             {printf(" %25s | %-25s \n", yytext, "right square"); return RIGHTSQUARE;}
"("             {printf(" %25s | %-25s \n", yytext, "left parantheses"); return LEFT;}
")"             {printf(" %25s | %-25s \n", yytext, "right parantheses"); return RIGHT;}

"="             {printf(" %25s | %-25s \n", yytext, "equal to op"); return ASSIGN_OP;}

"+"             {printf(" %25s | %-25s \n", yytext, "plus op"); return PLUS_OP;}
"-"             {printf(" %25s | %-25s \n", yytext, "minus op"); return MINUS_OP;}
"*"             {printf(" %25s | %-25s \n", yytext, "mul op"); return MUL_OP;}
"/"             {printf(" %25s | %-25s \n", yytext, "div op"); return DIV_OP;}
"%"             {printf(" %25s | %-25s \n", yytext, "mod op"); return MOD_OP;}

"!"             {printf(" %25s | %-25s \n", yytext, "not op"); return NOT_OP;}
"&&"            {printf(" %25s | %-25s \n", yytext, "and op"); return AND_OP;}
"||"            {printf(" %25s | %-25s \n", yytext, "or op"); return OR_OP;}

{relop}         {printf(" %25s | %-25s \n", yytext, "rel op"); return REL_OP;}

{intConst}|{doubConst}|{charConst}      {printf(" %25s | %-25s \n", yytext, "numeric constant");
yylval.num=atoi(strdup(yytext)); return CONSTANT;}
{stringConst}   {printf(" %25s | %-25s \n", yytext, "string constant"); yylval.string=strdup(yytext);
return STRING;}

{identifier}    {printf(" %25s | %-25s \n", yytext, "identifier"); yylval.string=strdup(yytext); return
ID;}

","             {printf(" %25s | %-25s \n", yytext, "comma"); return COMMA;}
";"             {printf(" %25s | %-25s \n", yytext, "semi-colon"); return EOS;}
.

%%



//ex6.y

%{
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "y.tab.h"
int yylex(void);
int yyerror();
extern FILE* yyin;
extern int line;
int error = 0;
int regCount = 1;
char reg[10];
%}

%union {
    char* string;
    int num;
    };

%type <string> expr boolop
%type <num> numexpr

%token <string> VOID INT FLOAT DOUBLE CHAR STR
%token <string> LEFTCURLY RIGHTCURLY LEFTSQUARE RIGHTSQUARE LEFT RIGHT
COMMA EOS
%token <string> ASSIGN_OP PLUS_OP MINUS_OP MUL_OP DIV_OP MOD_OP REL_OP
NOT_OP AND_OP OR_OP
%token <string> ID STRING
%token <num> CONSTANT

%%
program:
statement program {
    return 0;
    }
| '\n'
|
;

statement:
declnstatement EOS
| assignment EOS
;
```

```
declnstatement:
type var
;

type:
INT
| FLOAT
| DOUBLE
| CHAR
| STR
| type LEFTSQUARE RIGHTSQUARE
;

var:
var COMMA var
| ID
| assignment
;

assignment:
ID ASSIGN_OP expr {
    printf("MOV %s, %s\n", $1, $3);
}
| ID ASSIGN_OP numexpr {
    printf("MOV %s, %d\n", $1, $3);
}
;

expr:
ID {
    sprintf(reg, "reg%d", regCount++);
    printf("MOV %s, %s\n", reg, $1);
    $$ = strdup(reg);
}
| expr PLUS_OP expr {
    if (strcmp($1, "0") && strcmp($3, "0")) {
        printf("ADD %s, %s\n", $1, $3);
        $$ = strdup($1);
    }
    else {
        $$ = strdup($1);
    }
}
```

```
| expr MINUS_OP expr {
    if (strcmp($1, "0") && strcmp($3, "0")) {
        printf("SUB %s, %s\n", $1, $3);
        $$ = strdup($1);
    }
    else {
        $$ = strdup($1);
    }
}
| expr MUL_OP expr {
    if (strcmp($1, "1") && strcmp($3, "1")) {
        printf("MUL %s, %s\n", $1, $3);
        $$ = strdup($1);
    }
    else {
        $$ = strdup($1);
    }
}
| expr DIV_OP expr {
    if (strcmp($1, "1") && strcmp($3, "1")) {
        printf("DIV %s, %s\n", $1, $3);
        $$ = strdup($1);
    }
    else {
        $$ = strdup($1);
    }
}
| expr REL_OP expr
| expr boolop expr
| NOT_OP expr
| numexpr {
    char number[10];
    sprintf(number, "%d", $1);
    $$ = strdup(number);
}
;

numexpr:
CONSTANT {
    $$ = $1;
}
| numexpr PLUS_OP numexpr {
    $$ = $1 + $3;
```

```
}
| numexpr MINUS_OP numexpr {
   $$ = $1 - $3;
}
| numexpr MUL_OP numexpr {
   $$ = $1 * $3;
}
| numexpr DIV_OP numexpr {
   $$ = $1 / $3;
}
;

boolop:
AND_OP {
   $$ = strdup($1);
}
| OR_OP {
   $$ = strdup($1);
}
;

%%

int yywrap(){
   return 1;
}

int yyerror() {
   fprintf(stderr, "\n\nSyntax is NOT valid! Error at line %d\n", line);
   error = 1;
   return 0;
}

int main(int argc, char *argv[])
{
   printf("(till) Code Generator: \n");
   if (argc != 2) {
      printf("Please enter the sample file as the second argument!\n");
      exit(0);
   }

   yyin = fopen(argv[1], "r");
   if (!yyin) {
```

```
        printf("File not found!\n");
        exit(0);
    }

    yyparse();
    if(!error){
        printf("\n\nValid syntax!\n");
    }

    return 0;
}
```

Output:

```
kri@Krithika-PC-Win11:/mnt/e/ssn/sem 6/compiler design/lab/assignments/ex6$ ./a.out sample.txt
(till) Code Generator:
                        a | identifier
                        = | equal to op
                        b | identifier
MOV reg1, b
                        + | plus op
                        c | identifier
MOV reg2, c
                        * | mul op
                        4 | numeric constant
                        + | plus op
                        5 | numeric constant
                        ; | semi-colon
MUL reg2, 9
ADD reg1, reg2
MOV a, reg1


Valid syntax!
kri@Krithika-PC-Win11:/mnt/e/ssn/sem 6/compiler design/lab/assignments/ex6$ cat sample.txt
a = b + c * 4 + 5;
kri@Krithika-PC-Win11:/mnt/e/ssn/sem 6/compiler design/lab/assignments/ex6$
```

Learning outcomes:

- The internal working of a compiler was analysed and understood.
- The concept of tokens and parsing for tokens in Java was understood and implemented.
- A syntax analyser was implemented for a Java program using the lex and yacc tools.
- Intermediate code was generated for the given sample code using the lex and yacc tools.
- Intermediate code was optimised for the given sample code.
- Target was generated for the given sample code.
