COMPILER DESIGN
Assignment 5

Experiment-15

AIM

Given a simple input program (subset of C-like language), generate Three-Address Code (quadruples or TAC) using lexical scan (LEX) and parser (YACC).

ALGORITHM

1. Start.
2. Define tokens for identifiers, numbers, and operators in LEX.
3. In YACC, write grammar rules for expressions, assignments, and control statements.
4. For each grammar rule, attach semantic actions to:
    a. Generate temporary variables (t1, t2, …).
    b. Emit TAC instructions like t1 = a + b.
5. Use helper functions:
    a. newTemp() -> creates new temporary variable.
    b. emit(op, arg1, arg2, result) -> stores TAC line.
6. On successful parsing, print or store all generated TAC lines.
7. Stop

SOURCE CODE

Ex15.l

```
%option noyywrap
%{
#include "Ex15.tab.h"

#include <stdlib.h>
#include <string.h>
%}

digit   [0-9]
id
[A-Za-z_][A-Za-z0-9_]* ws
        [ \t\r\n]+
```

Aditya
~23BCE1344

```
%%{ws}       {}
"if"            { return IF; }
"else"          { return ELSE; }
"while"         { return WHILE;
}
{digit}+        { yylval.intval = atoi(yytext); return NUMBER; }
{id}            { yylval.id = strdup(yytext); return ID; }
"=="            { return EQ; }
"!="            { return NE; }
"<="            { return LE; }
">="            { return GE; }
"<"             { return '<'; }
">"             { return '>'; }
"="             { return '='; }
"+"             { return '+'; }
"-"             { return '-'; }
"*"             { return '*'; }
"/"             { return '/'; }
"("             { return '('; }
")"             { return ')'; }
"{"             { return '{'; }
"}"             { return '}'; }
";"             { return ';'; }
.               { }
%%
```

Ex15.y

```
%{
#include    <stdio.h>
#include    <stdlib.h>
#include <string.h>

int yylex(void);
void yyerror(const char *s);
void emit(const char* format, ...);
int tempCount = 0;
int labelCount = 0;

char *newTemp() {
    char buf[16];
    sprintf(buf, "t%d", ++tempCount);
    return strdup(buf);
```

```
}
char *newLabel() {
    char buf[16];
    sprintf(buf, "L%d", ++labelCount);
    return strdup(buf);
}

#define LPSTACK_MAX 1024
static char* label_stack[LPSTACK_MAX];
static int label_top = 0;

static void pushL(char *label) {
    if (label_top < LPSTACK_MAX) {
        label_stack[label_top++] = label;
    } else {
        fprintf(stderr, "label stack overflow\n");
        exit(1);
    }
}

static char* popL(void) {
    if (label_top <= 0) {
        fprintf(stderr,"label stack underflow\n");
        exit(1);
    }
    return label_stack[--label_top];
}
%}

%code requires {
    typedef struct {
        char *place;
    } node;
}

%union { int
    intval;
    char *id;
    node *nptr;
}

%token <id> ID
%token <intval> NUMBER
```

```
%token IF ELSE WHILE
%token EQ NE LE GE

%type <nptr> expr term factor b_expr

%left '+' '-'
%left '*' '/'

%nonassoc IF_NO_ELSE
%nonassoc ELSE

%%

program:
    stmt_lis
    t
;

stmt_list:
  | stmt_list stmt
;

stmt:
    expr_stmt
  | compound_stmt
  | if_stmt
  | while_stmt
;

compound_stmt:
    '{' stmt_list '}'
;

expr_stmt:
    assignment ';'
  | ';'
;

assignment:
    ID '=' expr  {
        emit("%s = %s", $1, $3->place);
        free($1);
        free($3->place);
        free($3);
```

```
    }
;

expr:
    term              { $$ = $1; }
  | expr '+' term     {
        $$ = (node*)malloc(sizeof(node));
        $$->place = newTemp();
        emit("%s = %s + %s", $$->place, $1->place, $3->place);
        free($1->place); free($1);
        free($3->place); free($3);
    }
  | expr '-' term    {
        $$ = (node*)malloc(sizeof(node));
        $$->place = newTemp();
        emit("%s = %s - %s", $$->place, $1->place, $3->place);
        free($1->place); free($1);
        free($3->place); free($3);
    }
;

term:
    factor            { $$ = $1; }
  | term '*' factor {
        $$ = (node*)malloc(sizeof(node));
        $$->place = newTemp();
        emit("%s = %s * %s", $$->place, $1->place, $3->place);
        free($1->place); free($1);
        free($3->place); free($3);
    }
  | term '/' factor {
        $$ = (node*)malloc(sizeof(node));
        $$->place = newTemp();
        emit("%s = %s / %s", $$->place, $1->place, $3->place);
        free($1->place); free($1);
        free($3->place); free($3);
    }
;

factor:
    NUMBER {
        $$ = (node*)malloc(sizeof(node));
        char buf[32];
```

```
        sprintf(buf, "%d", $1);
        $$->place = strdup(buf);
    }
  | ID {
        $$ = (node*)malloc(sizeof(node));
        $$->place = strdup($1);
        free($1);
    }
  | '(' expr ')' { $$ = $2; }
;

if_stmt:
    IF '(' b_expr ')' {
        char *l_else = newLabel();
        char *l_end = newLabel();
        emit("if %s == 0 goto %s", $3->place, l_else);
        pushL(l_end);
        pushL(l_else);
        free($3->place); free($3);
    }
    stmt ELSE {
        char* l_else = popL();
        char* l_end = popL();
        emit("goto %s", l_end);
        emit("%s:", l_else);
        pushL(l_end);
    }
    stmt {
        char* l_end = popL();
        emit("%s:", l_end);
    }

  | /* if without else */
    IF '(' b_expr ')' {
        char *l_after = newLabel();
        emit("if %s == 0 goto %s", $3->place, l_after);
        pushL(l_after);
        free($3->place); free($3);
    }
    stmt  %prec  IF_NO_ELSE
        {    char*    l_after    =
        popL();       emit("%s:",
        l_after);
    }
```

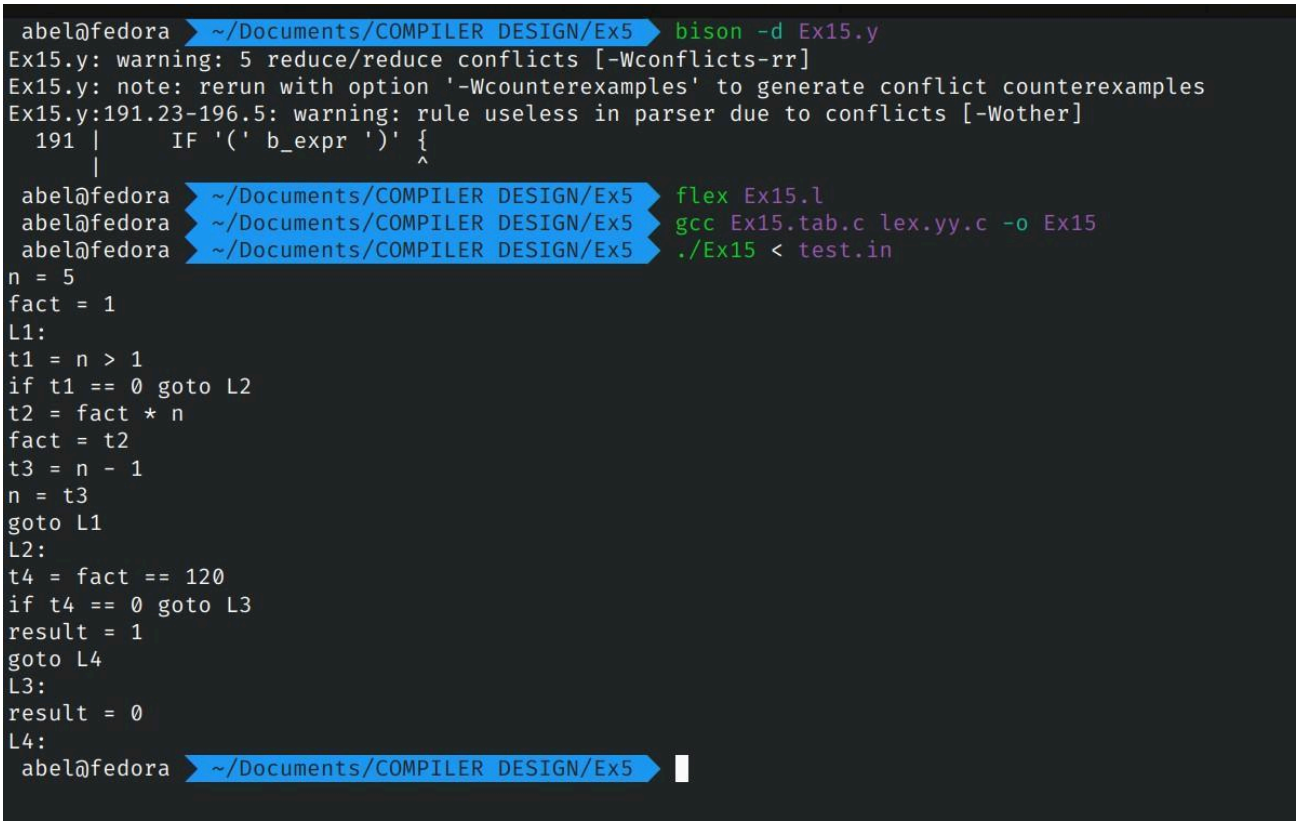
```
;

while_stmt:
    {
        char* l_start = newLabel();
        emit("%s:", l_start);
        pushL(l_start);
    }
    WHILE '(' b_expr ')' {
        char* l_end = newLabel();
        emit("if %s == 0 goto %s", $4->place, l_end);
        pushL(l_end);
        free($4->place); free($4);
    }
    stmt {
        char* l_end = popL();
        char* l_start = popL();
        emit("goto %s", l_start);
        emit("%s:", l_end);
    }
;

b_expr:
    expr EQ expr {
        $$ = (node*)malloc(sizeof(node));
        $$->place = newTemp();
        emit("%s = %s == %s", $$->place, $1->place, $3->place);
        free($1->place); free($1);
        free($3->place); free($3);
    }
  | expr NE expr {
        $$ = (node*)malloc(sizeof(node));
        $$->place = newTemp();
        emit("%s = %s != %s", $$->place, $1->place, $3->place);
        free($1->place); free($1);
        free($3->place); free($3);
    }
  | expr '<' expr {
        $$ = (node*)malloc(sizeof(node));
        $$->place = newTemp();
        emit("%s = %s < %s", $$->place, $1->place, $3->place);
        free($1->place); free($1);
        free($3->place); free($3);
```

```
    }
  | expr '>' expr {
        $$ = (node*)malloc(sizeof(node));
        $$->place = newTemp();
        emit("%s = %s > %s", $$->place, $1->place, $3->place);
        free($1->place); free($1);
        free($3->place); free($3);
    }
  | expr LE expr {
        $$ = (node*)malloc(sizeof(node));
        $$->place = newTemp();
        emit("%s = %s <= %s", $$->place, $1->place, $3->place);
        free($1->place); free($1);
        free($3->place); free($3);
    }
  | expr GE expr {
        $$ = (node*)malloc(sizeof(node));
        $$->place = newTemp();
        emit("%s = %s >= %s", $$->place, $1->place, $3->place);
        free($1->place); free($1);
        free($3->place); free($3);
    }
;

%%
void yyerror(const char *s) { fprintf(stderr,"Error: %s\n",s); }


#include <stdarg.h>


void emit(const char* format, ...) {
    va_list args;
    va_start(args, format);
    vprintf(format, args);
    va_end(args);
    printf("\n");
}

int main(void) {
    yyparse();
    return 0;
}
```

INPUT

```
(text.in) n = 5;
fact = 1; while
(n > 1) {
  fact = fact * n;
  n = n - 1;
}

if (fact == 120) {
    result = 1;
} else {
    result = 0;
}
```

OUTPUT



Aditya
~23BCE1344

Experiment-16

AIM

Given TAC as input, perform local/global passes to apply optimizations and produce improved TAC.

ALGORITHM

1. Start.
2. Read input TAC code.
3. Apply Constant Folding:
    a. If both operands are constants -> evaluate at compile time.
    b. Replace instruction with computed value.
4. Apply Strength Reduction:
    a. Replace costly operations:
        i. x = y * 2 -> x = y << 1
        ii. x = y / 2 -> x = y >> 1
5. Apply Algebraic Transformation:
    a. Simplify expressions:
        i. x = y + 0 -> x = y
        ii. x = y * 1 -> x = y
        iii. x = y * 0 -> x = 0
6. Remove redundant or dead code if any.
7. Output optimized TAC code.
8. Stop.

SOURCE CODE

Ex16.cpp

```
#include <bits/stdc++.h>
using namespace std;

string trim(const string &s) {
    size_t a = s.find_first_not_of(" \t\r\n");
    if (a==string::npos) return "";
    size_t b = s.find_last_not_of(" \t\r\n");
    return s.substr(a, b-a+1);
```

```
}

bool isInteger(const string &s) { if
    (s.empty()) return false; size_t
    i = 0;
    if (s[0]=='-' || s[0]=='+') i=1;
    for (; i<s.size(); ++i) if (!isdigit((unsigned char)s[i])) return false; return
    true;
}
long long toInt(const string &s) { return stoll(s); }
struct Instr {
    string raw;
    string lhs, op1, op, op2;
    bool isBinary=false;
    bool isAssign=false;
    bool other=false;
};

Instr parseLine(const string &ln) {
    Instr ins; ins.raw = ln;
    string s = trim(ln);
    if (s.empty()) { ins.other=true; return ins; }
    if (s.back()==':') { ins.other=true; return ins; }
    vector<string> keywords = {"if ", "goto ", "call ", "param ", "return ", "print ", "read "}; for
    (auto &k: keywords) if (s.rfind(k,0)==0) { ins.other=true; return ins; }
    auto eq = s.find('=');
    if (eq==string::npos) { ins.other=true; return ins; }
    ins.lhs = trim(s.substr(0, eq));
    string rhs = trim(s.substr(eq+1)); vector<string>
    tokens;
    {
        string t; istringstream iss(rhs);
        while (iss >> t) tokens.push_back(t);
    }
    if (tokens.size() == 3) {
        ins.isBinary = true;
        ins.op1 = tokens[0];
        ins.op = tokens[1];
        ins.op2 = tokens[2];
    } else if (tokens.size() == 1) {
        ins.isAssign = true;
```

```
        ins.op1 = tokens[0];
    } else {
        ins.other = true;
    }
    return ins;
}

bool isPowerOfTwo(long long n, int &k) { if
    (n<=0) return false;
    if ((n & (n-1)) != 0) return false; k
    = _builtin_ctzll(n);
    return true;
}

int main(){
    ios::sync_with_stdio(false)
    ; cin.tie(nullptr);
    vector<string> lines; string
    line;
    while (getline(cin, line)) {
        lines.push_back(line);
    }
    vector<Instr> ins;
    for (auto &ln: lines) ins.push_back(parseLine(ln));

    unordered_map<string, optional<long long>> cval;
    auto getConst = [&](const string &s)->optional<long long>{ if
        (isInteger(s)) return toInt(s);
        auto it = cval.find(s);
        if (it!=cval.end()) return it->second;
        return nullopt;
    };

    bool changed = true;
    int passes = 0;
    while (changed && passes < 10) {
        changed = false;
        ++passes;
        for (auto &I: ins) {
            if (I.isBinary) {
                auto c1 = getConst(I.op1);
                auto c2 = getConst(I.op2);
                if (c1 && c2) {
```

```
    long long a = *c1, b = *c2, res=0; bool
    foldable=true;
    if (I.op == "+") res = a+b; else if
    (I.op == "-") res = a-b; else if
    (I.op == "*") res = a*b;
    else if (I.op == "/") { if (b==0) foldable=false; else res = a/b; } else if
    (I.op == "%") { if (b==0) foldable=false; else res = a%b; } else if
    (I.op == "<<") res = a<<b;
    else if (I.op == ">>") res = a>>b; else
    foldable=false;
    if (foldable) {
        string newraw = I.lhs + " = " + to_string(res);
        if (I.raw != newraw) { I.raw = newraw; changed=true; }
        I.isBinary=false; I.isAssign=true; I.op1 = to_string(res); I.op=""; I.op2="";
        cval[I.lhs] = res;
        continue;
    }
}
if (I.op == "+" && ((c2 && *c2==0) || (c1 && *c1==0))) { string
    keep = c2 && *c2==0 ? I.op1 : I.op2;
    string newraw = I.lhs + " = " + keep;
    if (I.raw != newraw) { I.raw = newraw; changed=true; }
    I.isBinary=false; I.isAssign=true; I.op1 = keep; I.op=""; I.op2=""; if
    (isInteger(keep)) cval[I.lhs]=toInt(keep); else cval.erase(I.lhs);
    continue;
}
if (I.op == "-" && (c2 && *c2==0)) { string
    newraw = I.lhs + " = " + I.op1;
    if (I.raw != newraw) { I.raw = newraw; changed=true; }
    I.isBinary=false; I.isAssign=true; I.op1 = I.op1; I.op=""; I.op2=""; if
    (isInteger(I.op1)) cval[I.lhs]=toInt(I.op1); else cval.erase(I.lhs);
    continue;
}
if (I.op == "*") {
    if ((c2 && *c2==1) || (c1 && *c1==1)) {
        string keep = (c2 && *c2==1) ? I.op1 : I.op2;
        string newraw = I.lhs + " = " + keep;
        if (I.raw != newraw) { I.raw = newraw; changed=true; }
        I.isBinary=false; I.isAssign=true; I.op1 = keep; I.op=""; I.op2=""; if
        (isInteger(keep)) cval[I.lhs]=toInt(keep); else cval.erase(I.lhs);
        continue;
    }
    if ((c2 && *c2==0) || (c1 && *c1==0)) {
```

```
        string newraw = I.lhs + " = 0";
        if (I.raw != newraw) { I.raw = newraw; changed=true; } I.isBinary=false;
        I.isAssign=true; I.op1 = "0"; I.op=""; I.op2=""; cval[I.lhs]=0;
        continue;
    }
    if (c2 && *c2 > 0) { int
        k;
        if (isPowerOfTwo(*c2, k)) {
            string newraw = I.lhs + " = " + I.op1 + " << " + to_string(k); if
            (I.raw != newraw) { I.raw = newraw; changed=true; }
            I.op = "<<"; I.op2 = to_string(k);
            continue;
        }
    }
    if (c1 && *c1 > 0) { int
        k;
        if (isPowerOfTwo(*c1, k)) {
            string newraw = I.lhs + " = " + I.op2 + " << " + to_string(k); if
            (I.raw != newraw) { I.raw = newraw; changed=true; }
            I.op1 = I.op2; I.op = "<<"; I.op2 = to_string(k);
            continue;
        }
    }
}
if (I.op == "-" && I.op1 == I.op2) { string
    newraw = I.lhs + " = 0";
    if (I.raw != newraw) { I.raw = newraw; changed=true; }
    I.isBinary=false; I.isAssign=true; I.op1 = "0"; I.op=""; I.op2="";
    cval[I.lhs]=0;
    continue;
}

string newrhs;
bool replaced = false;
{
    string left = I.op1, right = I.op2; if
    (!isInteger(left)) {
        auto c = getConst(left);
        if (c) { left = to_string(*c); replaced=true; }
    }
    if (!isInteger(right)) {
        auto c = getConst(right);
```

```
                    if (c) { right = to_string(*c); replaced=true; }
                }
                if (replaced) {
                    newrhs = left + " " + I.op + " " + right; string
                    newraw = I.lhs + " = " + newrhs;
                    if (I.raw != newraw) { I.raw=newraw; changed=true; }
                    I.op1 = left; I.op2 = right;
                }
            }
            cval.erase(I.lhs);
        } else if (I.isAssign) {
            if (isInteger(I.op1)) {
                long long v = toInt(I.op1);
                if (!cval.count(I.lhs) || cval[I.lhs] != optional<long long>(v)) {
                    cval[I.lhs] = v; changed=true;
                }
            } else {
                auto c = cval.find(I.op1);
                if (c!=cval.end() && c->second.has_value()) {
                    cval[I.lhs] = c->second;
                    string newraw = I.lhs + " = " + to_string(*c->second); if
                    (I.raw != newraw) { I.raw=newraw; changed=true; }
                } else {
                    if (cval.count(I.lhs)) { cval.erase(I.lhs); changed=true; }
                }
            }
        } else {
        }
    }
}

for (auto &I: ins) {
    if (I.isBinary) {
        I.raw = I.lhs + " = " + I.op1 + " " + I.op + " " + I.op2;
    } else if (I.isAssign) {
        I.raw = I.lhs + " = " + I.op1;
    }
    cout << I.raw << "\n";
}
return 0;
}
```

INPUT

(Ex16input.tac)

```
t1 = 2 + 3
t2 = t1 * 4
a = b + 0
c = 0 * d x
= y * 8
z = x - x
p = 10
q = p + 5
r = q * 1
```

OUTPUT

Aditya
~23BCE1344

Experiment-17

AIM

Given Three-Address Code as input, generate equivalent 8086 assembly (MASM/TASM style) respecting calling convention, registers, memory layout.

ALGORITHM

1. Start.
2. Read TAC instructions as input.
3. Build symbol table and assign memory locations to variables.
4. For each TAC instruction:
    a. Translate using 8086 templates:
        i. t1 = a + b -> MOV AX, a -> ADD AX, b -> MOV t1, AX
        ii. if t1 < t2 goto L1 -> MOV AX, t1 -> CMP AX, t2 -> JL L1
        iii. goto L2 -> JMP L2
    b. Emit corresponding assembly lines.
5. Add labels and data segment declarations.
6. Write prologue (.data, .code, main:) and epilogue (HLT or RET).
7. Output final 8086 assembly program.
8. Stop.

SOURCE CODE

Ex17.cpp

```
#include <iostream>
#include <string>
#include <vector>
#include <sstream>
#include <algorithm>
#include <set>

bool isNumber(const std::string& s) { if
    (s.empty()) return false;
    if (s[0] == '-' && s.length() > 1) {
        return std::all_of(s.begin() + 1, s.end(), ::isdigit);
    }
```

```
    return std::all_of(s.begin(), s.end(), ::isdigit);
}


std::string getJumpInstruction(const std::string& op) { if
    (op == "==") return "JE";
    if (op == "!=") return "JNE";
    if (op == "<") return "JL";
    if (op == "<=") return "JLE"; if
    (op == ">") return "JG";
    if (op == ">=") return "JGE";
    return "";
}

void generateAssembly(const std::vector<std::string>& tac) {
    std::set<std::string> variables;

    for (const auto& line : tac) {
        std::stringstream ss(line);
        std::string token;
        std::vector<std::string> tokens;
        while(ss >> token) {
            tokens.push_back(token);
        }
        if (tokens.empty()) continue;


        if (tokens[0] == "if") {


            if (!isNumber(tokens[1])) variables.insert(tokens[1]);
            if (tokens.size() > 3 && !isNumber(tokens[3])) variables.insert(tokens[3]);
        } else if (tokens.size() > 1 && tokens[1] == "=") { // e.g., t1 = a + b if
            (!isNumber(tokens[0])) variables.insert(tokens[0]);
            if (!isNumber(tokens[2])) variables.insert(tokens[2]);
            if (tokens.size() > 4 && !isNumber(tokens[4])) variables.insert(tokens[4]);
        }
    }

    std::cout << "; Generated 8086 Assembly Code" << std::endl;
    std::cout << "MODEL SMALL" << std::endl;
    std::cout << "DATA" << std::endl; for
    (const auto& var : variables) {
        std::cout << "       " << var << " DW 0" << std::endl;
```

Aditya
~23BCE1344

```
}
std::cout << "CODE" << std::endl; std::cout
<< "MAIN PROC" << std::endl;
std::cout << "       MOV AX, @DATA" << std::endl;
std::cout << "       MOV DS, AX" << std::endl;
std::cout << std::endl;

for (const auto& line : tac) {
    std::cout << "; TAC: " << line << std::endl;
    std::stringstream ss(line);
    std::string
    first_word; ss >>
    first_word;

    if (first_word.back() == ':') {
        std::cout << first_word << std::endl;
        continue;
    }

    if (first_word == "goto") {
        std::string label;
        ss >> label;
        std::cout << "       JMP " << label << std::endl;
        std::cout << std::endl;
        continue;
    }

    if (first_word == "if") {
        std::string src1, op, src2, go, label;
        ss >> src1 >> op >> src2 >> go >> label;

        std::cout << "       MOV AX, " << src1 << std::endl;
        if(isNumber(src2)) {
            std::cout << "       CMP AX, " << src2 << std::endl;
        } else {
            std::cout << "       CMP AX, " << src2 << std::endl;
        }

        std::string jmp_inst = getJumpInstruction(op);
        std::cout << "       " << jmp_inst << " " << label << std::endl;
        std::cout << std::endl;
        continue;
    }
```

```
        std::stringstream line_ss(line);
        std::string dest, eq_op, src1, op, src2;
        line_ss >> dest >> eq_op >> src1;

        if (line_ss >> op >> src2) { if
            (isNumber(src1)) {
                std::cout << "       MOV AX, " << src1 << std::endl;
            } else {
                std::cout << "       MOV AX, " << src1 << " ; Move value of " << src1 << " to AX" <<
std::endl;
            }

            if (op == "+") {
                if (isNumber(src2)) std::cout << "       ADD AX, " << src2 << std::endl;
                else std::cout << "       ADD AX, " << src2 << " ; Add value of " << src2 << " to AX"
<< std::endl;
            } else if (op == "-") {
                if (isNumber(src2)) std::cout << "       SUB AX, " << src2 << std::endl;
                else std::cout << "       SUB AX, " << src2 << " ; Subtract value of " << src2 << " from
AX" << std::endl;
            } else if (op == "*") {
                if (isNumber(src2)) {
                    std::cout << "       MOV BX, " << src2 << std::endl;
                    std::cout << "       MUL BX      ; Multiply AX by BX" << std::endl;
                } else {
                    std::cout << "       MUL " << src2 << "          ; Multiply AX by value of " << src2 <<
std::endl;
                }
            } else if (op == "/") {
                std::cout << "       MOV DX, 0 ; Clear DX for division" << std::endl; if
                (isNumber(src2)) {
                    std::cout << "       MOV BX, " << src2 << std::endl;
                    std::cout << "       DIV BX      ; Divide DX:AX by BX" << std::endl;
                } else {
                    std::cout << "       DIV " << src2 << "          ; Divide DX:AX by value of " << src2 <<
std::endl;
                }
            }
            std::cout << "       MOV " << dest << ", AX" << " ; Store result in " << dest <<
std::endl;
        } else {
            if (isNumber(src1)) {
```

```
            std::cout << "       MOV AX, " << src1 << std::endl;
        } else {
            std::cout << "       MOV AX, " << src1 << " ; Move value of " << src1 << " to AX"
<< std::endl;
        }
        std::cout << "       MOV " << dest << ", AX" << " ; Store result in " << dest <<
std::endl;
    }
    std::cout << std::endl;
  }

  std::cout << "       MOV AH, 4CH" << std::endl;
  std::cout << "       INT 21H" << std::endl;
  std::cout << "MAIN ENDP" << std::endl;
  std::cout << "END MAIN" << std::endl;
}

int main() {
  std::vector<std::string> three_address_code;
  std::string line;

  std::cout << "Enter Three-Address Code (one instruction per line)." << std::endl; std::cout <<
  "Press Ctrl+D (Linux/macOS) or Ctrl+Z then Enter (Windows) to generate
code." << std::endl;
  std::cout << "  ----------------------------------------------------------" << std::endl;

  while (std::getline(std::cin, line)) {
    if (!line.empty()) {
      three_address_code.push_back(line);
    }
  }

  std::cout << "\n--- Generating 8086 Assembly Code ---\n" << std::endl;

  generateAssembly(three_address_code);

  return 0;
}
```

INPUT

(Ex17input.tac)

```
i = 0
sum = 0
L_START:
if i >= 5 goto L_END
t1 = sum + i
sum = t1
t2 = i + 1 i
= t2
goto L_START
L_END:
if sum != 10 goto L_ELSE
result = 1
goto L_END_IF
L_ELSE:
result = 0
L_END_IF
:
```

OUTPUT

```
 abel@fedora  ~/Documents/COMPILER DESIGN/Ex5  g++ Ex17.cpp -o Ex17
 abel@fedora  ~/Documents/COMPILER DESIGN/Ex5  ./Ex17 < Ex17input.tac
Enter Three-Address Code (one instruction per line).
Press Ctrl+D (Linux/macOS) or Ctrl+Z then Enter (Windows) to generate code.
----------------------------------------------------------------

--- Generating 8086 Assembly Code ---

; Generated 8086 Assembly Code
MODEL SMALL
DATA
    i DW 0
    result DW 0
    sum DW 0
    t1 DW 0
    t2 DW 0
CODE
MAIN PROC
    MOV AX, @DATA
    MOV DS, AX

; TAC: i = 0
    MOV AX, 0
    MOV i, AX ; Store result in i

; TAC: sum = 0
    MOV AX, 0
    MOV sum, AX ; Store result in sum

; TAC: L_START:
L_START:
; TAC: if i >= 5 goto L_END
    MOV AX, i
    CMP AX, 5
    JGE L_END

; TAC: t1 = sum + i
    MOV AX, sum ; Move value of sum to AX
    ADD AX, i ; Add value of i to AX
```

```
; TAC: t1 = sum + i
    MOV AX, sum ; Move value of sum to AX
    ADD AX, i ; Add value of i to AX
    MOV t1, AX ; Store result in t1

; TAC: sum = t1
    MOV AX, t1 ; Move value of t1 to AX
    MOV sum, AX ; Store result in sum

; TAC: t2 = i + 1
    MOV AX, i ; Move value of i to AX
    ADD AX, 1
    MOV t2, AX ; Store result in t2

; TAC: i = t2
    MOV AX, t2 ; Move value of t2 to AX
    MOV i, AX ; Store result in i

; TAC: goto L_START
    JMP L_START

; TAC: L_END:
L_END:
; TAC: if sum != 10 goto L_ELSE
    MOV AX, sum
    CMP AX, 10
    JNE L_ELSE

; TAC: result = 1
    MOV AX, 1
    MOV result, AX ; Store result in result

; TAC: goto L_END_IF
    JMP L_END_IF

; TAC: L_ELSE:
L_ELSE:
; TAC: result = 0
    MOV AX, 0
```

```
; TAC: L_ELSE:
L_ELSE:
; TAC: result = 0
    MOV AX, 0
    MOV result, AX ; Store result in result

; TAC: L_END_IF:
L_END_IF:
    MOV AH, 4CH
    INT 21H
MAIN ENDP
END MAIN
 abel@fedora  ~/Documents/COMPILER DESIGN/Ex5
```

Aditya
~23BCE1344