

Ex-no: 8
07-11-2024

M .Rohith
3122 21 5001 085

SSN COLLEGE OF ENGINEERING, KALAVAKKAM
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UCS 2702 - Compiler Lab

Programming Assignment-8 Implementation of code generation

Consider the given sequence of three address code as input and generate the output in the form of assembly code written using the instruction set of 8086.

Your code generator supports assignment statement, arithmetic operators, relational operators, iterative constructs, and conditional constructs.

The source program, generated TAC and the assembly code is shown below.

Source Program:

```
x=0;
for(i=1;i<=10;i++)
x=x+1
```

Three address code:

```
x=0
i=1
L3: if i<=10 goto L1
goto L2
L1: t1=x+i
x=t1
t2=i+1
i=t2
goto L3
L2:
```

Assembly code generated would be

```
MOV R0,#0
MOV R1,#1
MOV R2,#10
L3: CMP R1, R2
JLE L1
JMP L2
L1: ADD R0, R1
MOV R3, #1
ADD R1, R3
JMP L3
L2:
```

Program code:

Codegeneration.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

// Define a structure to map variables and constants to registers
typedef struct {
    char var[10];
    char reg[3];
} RegisterMap;

int reg_counter = 0;
RegisterMap reg_map[10]; // Array to store variable and constant-register mappings

// Function to get or allocate a register for a variable or constant
const char* get_register(const char* identifier, FILE *output_file) {
    // Check if the identifier (variable or constant) is already mapped
    for (int i = 0; i < reg_counter; i++) {
        if (strcmp(reg_map[i].var, identifier) == 0) {
            return reg_map[i].reg;
        }
    }

    // Allocate a new register if not found
    sprintf(reg_map[reg_counter].var, "%s", identifier);
    sprintf(reg_map[reg_counter].reg, "R%d", reg_counter);

    // If it's a constant, initialize it in the assembly code
    if (isdigit(identifier[0])) {
        fprintf(output_file, "MOV %s, #%s\n", reg_map[reg_counter].reg, identifier);
    }

    reg_counter++;
    return reg_map[reg_counter - 1].reg;
}

void translate_to_assembly(const char *line, FILE *output_file) {
    char clean_line[100];
    strcpy(clean_line, line);
    clean_line[strcspn(clean_line, "\r\n")] = 0;

    // Check if the line is a label (e.g., "L1:")
    if (strchr(clean_line, ':') && !strstr(clean_line, "=") && !strstr(clean_line, "goto")) {
        fprintf(output_file, "%s\n", clean_line);
        return;
    }

    // Handle "if" conditions, e.g., "if i <= 10 goto L1"
```

```

if (strstr(clean_line, "if")) {
    char var1[10], op[3], var2[10], label[10];
    sscanf(clean_line, "if %s %2s %s goto %s", var1, op, var2, label);

    // Get or allocate registers for variables and constants
    const char *reg1 = get_register(var1, output_file);
    const char *reg2 = isdigit(var2[0]) ? get_register(var2, output_file) : get_register(var2,
output_file);

    // Output the CMP instruction and appropriate jump based on the condition
    fprintf(output_file, "CMP %s, %s\n", reg1, reg2);
    if (strcmp(op, "<=") == 0) {
        fprintf(output_file, "JLE %s\n", label);
    } else if (strcmp(op, "<") == 0) {
        fprintf(output_file, "JL %s\n", label);
    } else if (strcmp(op, ">=") == 0) {
        fprintf(output_file, "JGE %s\n", label);
    } else if (strcmp(op, ">") == 0) {
        fprintf(output_file, "JG %s\n", label);
    } else if (strcmp(op, "==") == 0) {
        fprintf(output_file, "JE %s\n", label);
    } else if (strcmp(op, "!=") == 0) {
        fprintf(output_file, "JNE %s\n", label);
    }
    return;
}

// Handle assignment operations
if (strstr(clean_line, "=")) {
    char var[10], expr[50];
    sscanf(clean_line, "%[^]=%s", var, expr);

    // Get or allocate a register for the variable
    const char *reg_var = get_register(var, output_file);

    if (strstr(expr, "+")) {
        char lhs[10], rhs[10];
        sscanf(expr, "%[^+]+%s", lhs, rhs);

        // Get or allocate registers for operands
        const char *reg_lhs = get_register(lhs, output_file);
        const char *reg_rhs = get_register(rhs, output_file);
        fprintf(output_file, "MOV %s, %s\n", reg_var, reg_lhs);
        fprintf(output_file, "ADD %s, %s\n", reg_var, reg_rhs);
    } else if (isdigit(expr[0]) || expr[0] == '#') {
        // If it's an immediate value, store it in the register directly
        fprintf(output_file, "MOV %s, %s\n", reg_var, expr);
    } else {
        // Otherwise, assume it's a variable and move it to the target register
        fprintf(output_file, "MOV %s, %s\n", reg_var, get_register(expr, output_file));
    }
    return;
}

```

```

    }

    // Handle "goto" statements
    if (strstr(clean_line, "goto")) {
        char label[10];
        sscanf(clean_line, "goto %s", label);
        fprintf(output_file, "JMP %s\n", label);
    }
}

int main() {
    FILE *input_file = fopen("input.txt", "r");
    FILE *output_file = fopen("assembly_code.txt", "w");

    if (!input_file || !output_file) {
        perror("Error opening file");
        return 1;
    }

    char line[100];
    while (fgets(line, sizeof(line), input_file)) {
        translate_to_assembly(line, output_file);
    }

    fclose(input_file);
    fclose(output_file);
    printf("Translation completed! Assembly code saved in assembly_code.asm.\n");

    return 0;
}

```

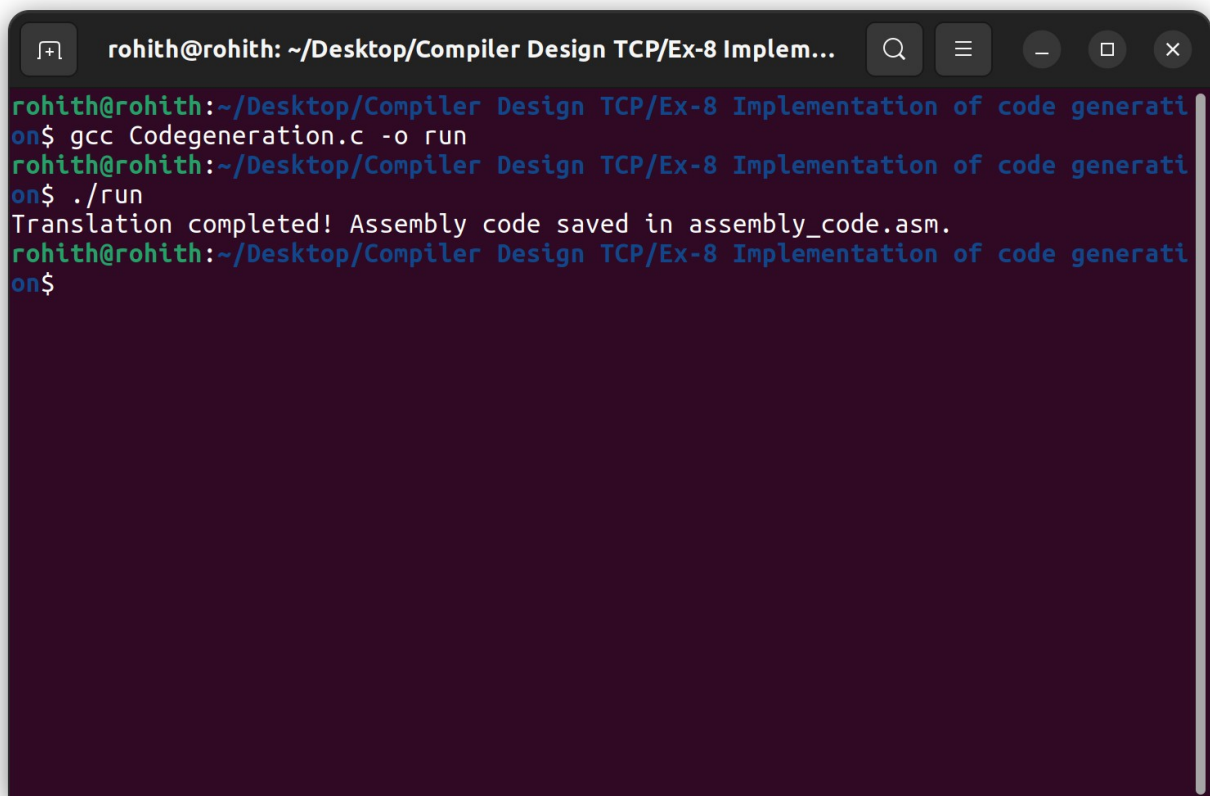
input.txt

```

x=0
i=1
L3:
if i <= 10 goto L1
goto L2
L1:
t1=x+i
x=t1
t2=i+1
i=t2
goto L3
L2:

```

Output



```
rohith@rohith: ~/Desktop/Compiler Design TCP/Ex-8 Implementation of code generati
on$ gcc Codegeneration.c -o run
rohith@rohith:~/Desktop/Compiler Design TCP/Ex-8 Implementation of code generati
on$ ./run
Translation completed! Assembly code saved in assembly_code.asm.
rohith@rohith:~/Desktop/Compiler Design TCP/Ex-8 Implementation of code generati
on$
```

assembly_code.txt

```
MOV R0, #0
MOV R1, #1
L3:
MOV R2, #10
CMP R1, R2
JLE L1
JMP L2
L1:
MOV R3, R0
ADD R3, R1
MOV R0, R3
MOV R5, #1
MOV R4, R1
ADD R4, R5
MOV R1, R4
JMP L3
L2:
```