S. B. JAIN INSTITUTE OF TECHNOLOGY, MANAGEMENT & RESEARCH, NAGPUR.

Practical No. 8

AIM: Construct a program to generate an assembly language instructions from the given three address code using simple code generation algorithm.

Name of Student: Shrutika Pradeep Bagdi

Roll No.: CS22130

Semester/Year: 6th Semester/3rd Year

Academic Session: 2024-25

Date of Performance:

Date of Submission:

AIM: Construct a program to generate an assembly language instruction from the given three address code using simple code generation algorithm.

OBJECTIVE / EXPECTED LEARNING OUTCOME:

The objectives and expected learning outcome of this practical are:

- To relate the prerequisite of course i.e., Computer Architecture and Organization with the Language Processor.
- To generate assembly code from given TAC (Three Address Code) using some algorithm.

HARDWARE AND SOFTWARE REQUIRMENTS:

Hardware Requirement:

- Processor: Dual Core
- RAM: 1GB
- Hard Disk Drive: > 80 GB

THEORY:

1)Machine model for the Code Generation

2) Simple code generation algorithm

3)Working of getreg() function in code generation algorithm

4)Limitations of simple code generation algorithm

ALGORITHM / PROCEDURE:

CODE:

csc15@linux-p2-1272il: ~/CS22130

```c
#include <stdio.h>
#include <string.h>

// Maximum number of 3-address code instructions and assembly instructions
#define MAX_INSTRUCTIONS 50
#define MAX_REGISTERS 4

// Data structure for 3-address code instructions
typedef struct {
    char result[10];   // Result variable
    char operand1[10]; // First operand
    char operand2[10]; // Second operand
    char op[2];        // Operator (+, -, *, /)
} ThreeAddressCode;

// Function to generate assembly code
void generateAssembly(ThreeAddressCode tac[], int numInstructions) {
    // Register names
    char* registers[MAX_REGISTERS] = {"R0", "R1", "R2", "R3"};
    int regIndex = 0; // Register index for the next free register

    // Output assembly code
    for (int i = 0; i < numInstructions; i++) {
        ThreeAddressCode ins = tac[i];

        // For the first instruction, move the first operand into a register
        if (regIndex < MAX_REGISTERS) {
            printf("MOV %s, %s\n", registers[regIndex], ins.operand1);
            regIndex++;
        }
```

```c
        // Generate the corresponding operation
        if (strcmp(ins.op, "+") == 0) {
            printf("ADD %s, %s\n", registers[regIndex - 1], ins.operand2);
        } else if (strcmp(ins.op, "-") == 0) {
            printf("SUB %s, %s\n", registers[regIndex - 1], ins.operand2);
        } else if (strcmp(ins.op, "*") == 0) {
            printf("MUL %s, %s\n", registers[regIndex - 1], ins.operand2);
        } else if (strcmp(ins.op, "/") == 0) {
            printf("DIV %s, %s\n", registers[regIndex - 1], ins.operand2);
        }

        // Move result to the destination register
        printf("MOV %s, %s\n", ins.result, registers[regIndex - 1]);
    }
}

int main() {
    int numInstructions;

    // Ask user for the number of instructions
    printf("Enter the number of 3-address code instructions: ");
    scanf("%d", &numInstructions);

    // Check if the number of instructions exceeds the limit
    if (numInstructions > MAX_INSTRUCTIONS) {
        printf("Number of instructions exceeds the maximum limit of %d.\n", MAX_INSTRUCTIONS);
        return 1;
    }

    // Array to store the 3-address code instructions
    ThreeAddressCode tac[MAX_INSTRUCTIONS];
```

```
    ThreeAddressCode tac[MAX_INSTRUCTIONS];

    // Take user input for each 3-address code instruction
    for (int i = 0; i < numInstructions; i++) {
        printf("\nEnter instruction %d (result operand1 operand2 operator):\n", i + 1);
        printf("Result: ");
        scanf("%s", tac[i].result);
        printf("Operand1: ");
        scanf("%s", tac[i].operand1);
        printf("Operand2: ");
        scanf("%s", tac[i].operand2);
        printf("Operator (+, -, *, /): ");
        scanf("%s", tac[i].op);
    }

    // Generate assembly code from 3-address code
    generateAssembly(tac, numInstructions);

    return 0;
}
```

OUTPUT:

```
csc15@linux-p2-1272il:~/CS22130$ vi Practical8.c
csc15@linux-p2-1272il:~/CS22130$ cc Practical8.c
csc15@linux-p2-1272il:~/CS22130$ ./a.out
Enter the number of 3-address code instructions: 4

Enter instruction 1 (result operand1 operand2 operator):
Result: t1
Operand1: a
Operand2: b
Operator (+, -, *, /): +

Enter instruction 2 (result operand1 operand2 operator):
Result: t2
Operand1: t1
Operand2: c
Operator (+, -, *, /): -

Enter instruction 3 (result operand1 operand2 operator):
Result: t3
Operand1: t2
Operand2: d
Operator (+, -, *, /): *

Enter instruction 4 (result operand1 operand2 operator):
Result: t4
Operand1: t3
Operand2: e
```

```
Operator (+, -, *, /): /
MOV R0, a
ADD R0, b
MOV t1, R0
MOV R1, t1
SUB R1, c
MOV t2, R1
MOV R2, t2
MUL R2, d
MOV t3, R2
MOV R3, t3
DIV R3, e
MOV t4, R3
csc15@linux-p2-1272il:~/CS22130$ vi Practical8.c
```

CONCLUSION:

DISCUSSION AND VIVA VOCE:

Q1: What is assembly code?

Q2: What are challenges involved in machine code/assembly code generation?

Q3: Can DAG be used in code generation phase? How?

Q4: What is labelling algorithm in code generation??

Q5: What are the different addressing modes in machine model?

REFERENCE:

- Book: Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman, "Compilers Principles, Techniques and Tools", Pearson Education, 2nd edition. 2010.
- Book: Compiler Design by O.G. Kakde, Laxmi Publications, 2006.
- Lab Manual of Compiler Design (Institute of Aeronautical Engineering, Dundigal, Hyderabad).