5.6 GENERATING ASSEMBLY CODE FROM THREE ADDRESS CODE

Introduction

Three Address Code (TAC) is an intermediate code used in compiler design. It represents a program using simple instructions with at most three addresses (two operands and one result).

After generating TAC, the compiler converts it into assembly language. This process is called Code Generation.

The main purpose is to generate efficient assembly instructions that can be executed by the processor.

5.6.1 Three Address Code

Consider the expression:

a = b + c * d

The corresponding Three Address Code is:

t1 = c * d

t2 = b + t1

a = t2

Here, t1 and t2 are temporary variables used to store intermediate results.

5.6.2 Assembly Code Generation

The compiler converts each TAC instruction into equivalent assembly instructions.

Example:

MOV R1, c

MUL R1, d

MOV R2, b

ADD R2, R1

MOV a, R2

Here:

- MOV – Copies a value.
- ADD – Performs addition.
- MUL – Performs multiplication.

- R1, R2 – Processor registers.

Basic Flow:

Source Program

↓

Intermediate Code

↓

Three Address Code

↓

Instruction Selection

↓

Register Allocation

↓

Assembly Code

↓

Machine Code

5.6.3 Steps in Generating Assembly Code

1. Identify the Operations

The compiler first identifies the operations in TAC, such as:

- Addition
- Subtraction
- Multiplication
- Division
- Assignment
- Comparison
- Conditional jump

2. Select Suitable Instructions

The compiler selects suitable assembly instructions for each TAC instruction.

Example:

TAC:

t1 = a + b

Assembly:

MOV R1, a

ADD R1, b

MOV t1, R1

3. Register Allocation

Registers are small storage locations inside the processor. The compiler decides which variables and temporary values should be stored in registers.

Example:

MOV R1, a

ADD R1, b

Here, R1 stores the intermediate result.

4. Generate Assembly Instructions

After selecting instructions and assigning registers, the compiler generates the final assembly code.

Example:

TAC:

t1 = b + c

t2 = t1 * d

a = t2

Assembly Code:

MOV R1, b

ADD R1, c

MOV R2, R1

MUL R2, d

MOV a, R2

Steps:

TAC: t1 = b + c

↓

MOV R1, b

ADD R1, c

↓

TAC: t2 = t1 * d

↓

MOV R2, R1

MUL R2, d

↓

TAC: a = t2

↓

MOV a, R2

5.6.4 Generating Assembly for Conditional Statements

Three Address Code for a conditional statement:

if a < b goto L1

goto L2

L1: c = a + b

L2:

Corresponding Assembly Code:

MOV R1, a

CMP R1, b

JL L1

JMP L2

L1:

MOV R2, a

ADD R2, b

MOV c, R2

L2:

Here:

- CMP – Compares two values.
- JL – Jump if Less.
- JMP – Performs an unconditional jump.
- L1 and L2 – Labels used for jumping.

5.6.5 Important Considerations

While generating assembly code, the compiler considers the following factors:

1. Instruction Selection: Choosing suitable machine instructions.
2. Register Allocation: Assigning variables to available registers.
3. Addressing Modes: Selecting efficient ways to access operands.
4. Instruction Ordering: Arranging instructions in the correct order.
5. Memory Usage: Managing variables and temporary values.
6. Code Optimization: Removing unnecessary instructions.
7. Target Machine Architecture: Generating code according to the target processor.