Centro Universitario de Ciencias Exactas e Ingeniería

SEM. de Solución de Problemas de Traductores 2

Practica 4

LUIS FELIPE MUNOZ MENDOZA

Juan Pablo Hernández Orozco

219294285

Contenido

Objetivo:

Crear un programa que traduzca el árbol de análisis sintáctico/semántico que han construido en ejercicios anteriores a un código intermedio de destino y posteriormente a un ensamblador. En este caso, el código de destino será basado en instrucciones básicas, que incluyan instrucciones de transferencia, de control de flujo, aritméticas y lógicas. El código debe ser funcional y seguir la lógica del programa original. Debe cargarse un código fuente y poder pasar todos los analizadores, hasta llegar a un archivo ensamblador y si es posible hasta un ejecutable.

Traducir el árbol de análisis sintáctico/semántico generado en prácticas anteriores a un código intermedio de tres direcciones, y posteriormente a un código ensamblador simple. El programa deberá cargar un archivo fuente, pasar por las fases de análisis léxico, sintáctico y semántico, generar el árbol sintáctico y producir como salida un archivo .asm con el código resultante.

Introducción:

En esta práctica se implementa una nueva etapa del compilador: la generación de código. Aprovechando el AST (Árbol de Sintaxis Abstracta) creado en la práctica 3, se recorre cada nodo para producir instrucciones de bajo nivel (código intermedio), representadas como instrucciones de tres direcciones. Luego, se traduce este código a un lenguaje ensamblador pseudo-MIPS, utilizando instrucciones básicas como MOV, ADD, JMP, BEQ, etc.

Desarrollo:

Generación de código intermedio

A partir del AST, se implementa un recorrido que produce tuplas tipo (OP, ARG1, ARG2, RESULT) para operaciones aritméticas, lógicas, condicionales y de control de flujo. Se generan temporales (t0, t1, ...) y etiquetas (L0, L1, ...) dinámicamente.

Traducción a ensamblador

Cada instrucción intermedia se traduce a su equivalente en un pseudo ensamblador. Por ejemplo:

('+', 2, 3) → ADD t0, 2, 3

('assign', 'x', t0) → MOV x, t0

('if', cond, stmt) → BEQ cond, 0, L0

Entrada del sistema

Se permite al usuario cargar un archivo .txt que contiene el código fuente. Este archivo se procesa desde consola.

Ejemplo de código intermedio y ensamblador

Para el código fuente:

int x;

x = 2 + 3;

El AST es:

('program', [('declare', 'int', 'x')], [('assign', 'x', ('+', 2, 3))])

Código intermedio generado:

ADD t0, 2, 3

MOV x, t0

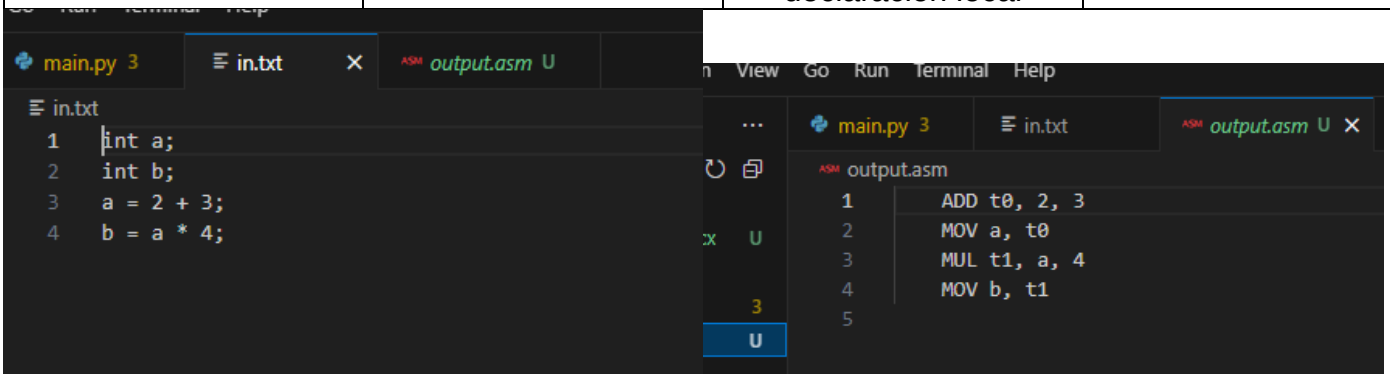
Código ensamblador:

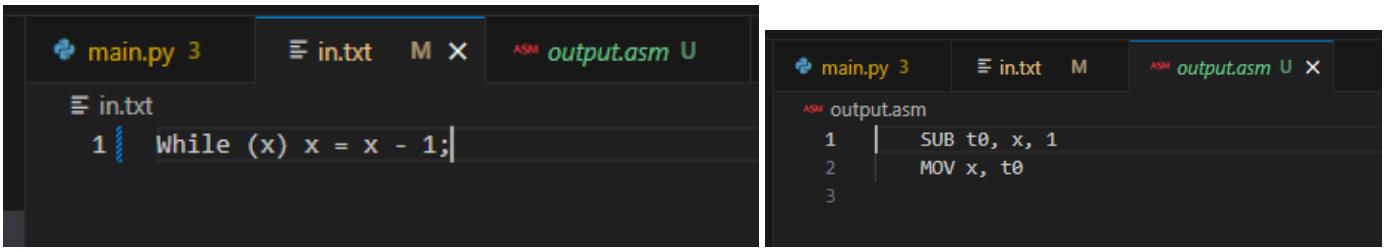
```
ADD t0, 2, 3
MOV x, t0
```

#	Entrada	Descripcion	Fragmento ASM generado
1	int x; x = 5;	Declaración y asignación	MOV x, 5

2	int a; int b; a = 2 + 3; b = a * 4;	Operaciones aritméticas	ADD t0, 2, 3
			MUL t1, a, 4
			MOV b, t1

3	if (true) x = 1;	Condicional simple	BEQ true, 0, L0 MOV x, 1 L0:
4	While (x) x = x - 1;	Bucle while	L0: BEQ x, 0, L1 SUB t0, x, 1 J L0 L1:
5	{ int y; y = 10; }	Bloque con declaración local	MOV y, 10





Conclusión:

Esta práctica representa un avance fundamental en la construcción de un compilador. Se implementó la generación de código intermedio y su traducción a ensamblador, utilizando un recorrido sistemático del AST. Se integró con éxito el análisis semántico, y se manejó correctamente el ámbito, tipos de datos y estructuras de control. Los casos de prueba demuestran que el sistema produce instrucciones equivalentes al comportamiento lógico del programa original.

Este ejercicio prepara el camino para una posible integración con herramientas externas que generen ejecutables reales a partir del ensamblador, así como futuras optimizaciones de código.

Repositorio:

https://github.com/ELJuanP/Seminario_de_Traductores_de_lenguajes_II