



UAT

VERDAD, BELLEZA, PROBIDAD

**IFI** Facultad de Ingeniería  
Arturo Narro Siller

# UNIVERSIDAD AUTÓNOMA DE TAMAULIPAS

Facultad de Ingeniería "Arturo Narro Siller"

**Ingeniería en Sistemas Computacionales**

**PROGRAMACIÓN DE SISTEMAS DE BASE  
II**

**Docente:** Muñoz Quintero Dante Adolfo

**Proyecto Final**

VERDAD, BELLEZA, PROBIDAD

**Alumnos:**

Aldama Trinidad Alfonso René  
Domínguez Reyes Pavel Noel

**Horario:** 13:00-14:00

**Semestre y Grupo:** 9° "J"

**Fecha de entrega:** 27 de noviembre del 2025

## Índice

Introducción .....	3
Instalación y configuración.....	3
Requisitos del Sistema .....	3
Estructura del Directorio.....	3
Guía de uso.....	4
Ejecucion Exitosa de ejemplos:.....	6
Documentación técnica Arquitectura del Compilador .....	13
1. Origen del Código y Herramientas Utilizadas .....	13
2. Descripción de Módulos por Fase.....	14
Fase 1: Análisis Léxico y Sintáctico (Frontend).....	14
Fase 2: Análisis Semántico .....	14
Fase 3: Generación de Código Intermedio (GCI) .....	15
Fase 4: Optimización Independiente de la Máquina .....	15
Fase 5: Generación de Código Objeto (Backend) .....	15
Código fuente .....	16

## Introducción

El presente proyecto tiene como objetivo desarrollar un compilador funcional para un subconjunto específico del lenguaje de programación Python. La finalidad principal del sistema es realizar el análisis léxico, sintáctico y semántico del código fuente ingresado para, posteriormente, traducir y generar su equivalente en Lenguaje Ensamblador para la arquitectura MIPS.

Este compilador no se limita a ser un intérprete, sino que realiza una traducción completa de código de alto nivel a bajo nivel, permitiendo visualizar el proceso de transformación de instrucciones lógicas a instrucciones de máquina.

Dado que se trata de un diseño académico enfocado en la comprensión de las fases de compilación, el lenguaje fuente soportado presenta características específicas que lo diferencian de la implementación estándar de Python:

- **Tipado y Datos:** El compilador está diseñado exclusivamente para el cálculo numérico. Soporta operaciones con números enteros.
- **Restricciones de Tipos:** No se incluye soporte para cadenas de caracteres (*strings*) ni números de punto flotante (*floats*).
- **Sintaxis y Estructura:** A diferencia de Python estándar, este subconjunto no utiliza la indentación como delimitador de bloques de código, simplificando el análisis sintáctico para enfocarse en la generación de código.
- **Arquitectura Objetivo:** El código resultante es compatible con procesadores o simuladores MIPS (como SPIM o MARS).

Para la fase de análisis (frontend del compilador), se ha utilizado la herramienta ANTLR (Another Tool for Language Recognition), la cual facilita la generación de analizadores léxicos y sintácticos a partir de una gramática formal definida.

## Instalación y configuración

### Requisitos del Sistema

Para la correcta ejecución del compilador, se requiere el siguiente software:

- **Python 3.x:** Lenguaje base del proyecto (confirmado por el entorno virtual `venv` visible).
- **Java Runtime Environment (JRE):** Necesario para ejecutar la herramienta ANTLR (`antlr-4.13.2-complete.jar`).
- **Librerías de Python:** Las dependencias se encuentran aisladas en el entorno virtual.

### Estructura del Directorio

El proyecto sigue una estructura modular organizada de la siguiente manera:

- **/src:** Contiene el código fuente del compilador.
  - main.py: Punto de entrada principal de la aplicación.
  - PythonSubset.g4: Gramática formal del lenguaje (definición de reglas para ANTLR).
  - VisitanteSemantico.py: Módulo encargado del análisis semántico y validación de tipos.
  - Tabla\_de\_Simbolos.py: Gestión de variables y ámbitos durante la compilación.
  - Codigointermedio.py: Generación de la representación intermedia.
  - Optimizador.py: Módulo de optimización de código.
  - GeneradorMIPS.py: Traducción final a lenguaje ensamblador.
- **/lib:** Contiene las herramientas externas (antlr-4.13.2-complete.jar).
- **/examples:** Casos de prueba para validación.
  - /valid: Ejemplos de código correcto.
  - /invalid: Ejemplos con errores intencionales para probar el manejo de excepciones.
  - /expected: Resultados esperados de la compilación.
- **/tests:** Scripts de pruebas automatizadas (run\_tests.py).

## Guía de uso

Se debe tener instalado:

- Python 3.8+: Para ejecutar el compilador
- Java Runtime (JRE): Necesario solo para regenerar la gramática con ANTLR
- Simulador Mars (Mars4\_5.jar): Para ejecutar el código ensamblador generado.

Se debe tener la estructura correcta del proyecto  
proyecto/

```

lib/ Contiene antlr-4.13.1-complete.jar

src/ Código fuente (Lexer, Parser, Generador, etc.)

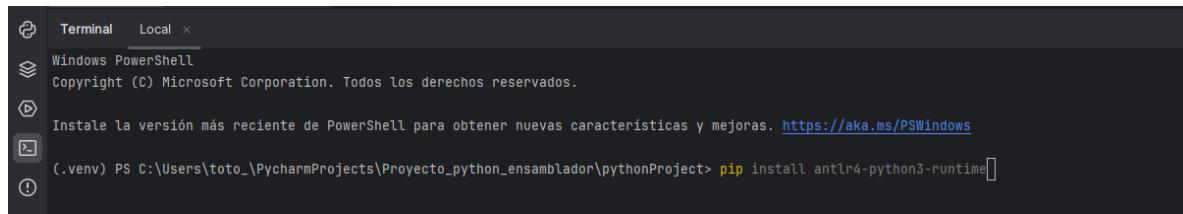
examples/ Ejemplos de prueba (.txt y .asm)

tests/ Scripts de pruebas automatizadas

Makefile Script de construcción
  
```

Se debe instalar el runtime de ANTLR

En la terminal del proyecto se ejecuta el siguiente comando: pip install antlr4-python3-runtime



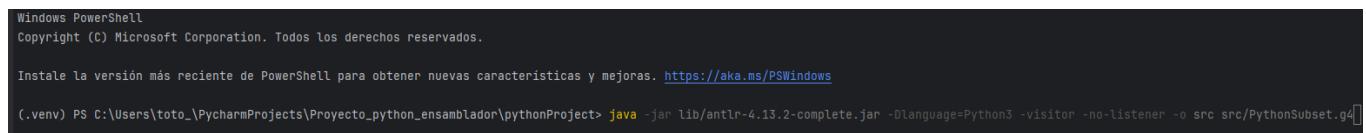
```
Terminal Local
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. https://aka.ms/PSWindows

(.venv) PS C:\Users\toto_\PycharmProjects\Proyecto_python_ensamblador\pythonProject> pip install antlr4-python3-runtime
```

Se debe traducir la gramática .g4 a código python, se ejecuta el siguiente comando en la terminal del proyecto:

```
java -jar lib/antlr-4.13.2-complete.jar -Dlanguage=Python3 -visitor -no-listener -o src
src/PythonSubset.g4
```



```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. https://aka.ms/PSWindows

(.venv) PS C:\Users\toto_\PycharmProjects\Proyecto_python_ensamblador\pythonProject> java -jar lib/antlr-4.13.2-complete.jar -Dlanguage=Python3 -visitor -no-listener -o src src/PythonSubset.g4
```

Despues de ejecutar el subset.g4 tendremos todo lo que necesitamos para usar nuestro traductor de código.

Iremos al archivo run\_tests.py y correremos el código aquí usara los casos validos e inválidos y mostrara en consola si fue exitoso y los casos de errores que preparamos mostrara el tipo de error.



```
== PRUEBAS VÁLIDAS (7) ==
Testing 01_prueba_lexico_sintactico.txt      ... DEBUG: Definida variable 'funcion_vacia' de tipo 'FUNCION' en alcance nivel 1
DEBUG: Definida variable 'x' de tipo 'ENTERO' en alcance nivel 1
PASS
Testing 02_prueba_alcance_semantico.txt      ... DEBUG: Definida variable 'a' de tipo 'ENTERO' en alcance nivel 1
DEBUG: Definida variable 'prueba' de tipo 'FUNCION' en alcance nivel 1
DEBUG: Definida variable 'x' de tipo 'PARAMETRO' en alcance nivel 2
DEBUG: Definida variable 'a' de tipo 'ENTERO' en alcance nivel 2
PASS
Testing 03_prueba_GeneracionCodigoIntermedio.txt ... DEBUG: Definida variable 'a' de tipo 'ENTERO' en alcance nivel 1
DEBUG: Definida variable 'b' de tipo 'ENTERO' en alcance nivel 1
DEBUG: Definida variable 'c' de tipo 'ENTERO' en alcance nivel 1
DEBUG: Definida variable 'res1' de tipo 'ENTERO' en alcance nivel 1
DEBUG: Definida variable 'res2' de tipo 'ENTERO' en alcance nivel 1
PASS
Testing 04_prueba_optimizacion.txt            ... DEBUG: Definida variable 'resultado' de tipo 'ENTERO' en alcance nivel 1
PASS
```

```

Testing 05_prueba_factorial.txt      ... DEBUG: Definida variable 'numero' de tipo 'ENTERO' en alcance nivel 1
DEBUG: Definida variable 'res' de tipo 'ENTERO' en alcance nivel 1
DEBUG: Definida variable 'res' de tipo 'ENTERO' en alcance nivel 1
DEBUG: Definida variable 'numero' de tipo 'ENTERO' en alcance nivel 1
    PASS
Testing 06_prueba_sumatoria.txt     ... DEBUG: Definida variable 'total' de tipo 'ENTERO' en alcance nivel 1
DEBUG: Definida variable 'contador' de tipo 'ENTERO' en alcance nivel 1
DEBUG: Definida variable 'limite' de tipo 'ENTERO' en alcance nivel 1
DEBUG: Definida variable 'total' de tipo 'ENTERO' en alcance nivel 1
DEBUG: Definida variable 'contador' de tipo 'ENTERO' en alcance nivel 1
    PASS
Testing 07_prueba_strings.txt      ... DEBUG: Definida variable 'x' de tipo 'ENTERO' en alcance nivel 1
    PASS

```

Como se puede observar todos los casos validos funcionan y muestra que pasaron las pruebas.

```

== PRUEBAS INVÁLIDAS (6) ==
Testing 01_error_lexico.txt          ... (Error detectado: Error Sintáctico en linea 4:6 - token recognition error at: '@')
Testing 02_error_sintactico.txt      ... (Error detectado: Error Sintáctico en linea 5:9 - missing ':' at '{')
Testing 03_error_Semantico.txt       ... DEBUG: Definida variable 'x' de tipo 'ENTERO' en alcance nivel 1
    (Error detectado: Error Semántico en linea 5: La variable 'z' no ha sido definida.)
Testing 04_error_semanticoDeAlcance.txt ... DEBUG: Definida variable 'calcular' de tipo 'FUNCION' en alcance nivel 1
DEBUG: Definida variable 'secreto' de tipo 'ENTERO' en alcance nivel 2
    (Error detectado: Error Semántico en linea 8: La variable 'secreto' no ha sido definida.)
Testing 05_error_semantico_parametros.txt ... DEBUG: Definida variable 'suma' de tipo 'FUNCION' en alcance nivel 1
DEBUG: Definida variable 'a' de tipo 'PARAMETRO' en alcance nivel 2
DEBUG: Definida variable 'b' de tipo 'PARAMETRO' en alcance nivel 2
    (Error detectado: Error Semántico en linea 9: 'suma' es una función, no una variable. ¿Olvidaste los paréntesis ()?)
Testing 06_error_optimizador.txt      ... DEBUG: Definida variable 'x' de tipo 'ENTERO' en alcance nivel 1
    (Error detectado: float division by zero)

```

En las pruebas de casos inválidos muestra el error detectado.

### Ejecucion Exitosa de ejemplos:

Para las pruebas del código traducido a ensamblador se ocupó Mars 4.5 que se encuentra en la carpeta lib que es un simulador que ejecuta código Ensamblador MIPS y se ejecutaron las salidas validas.

En el archivo de salidas esperadas muestra que debe aparecer en la consola en cada prueba al ejecutarse el código ensamblador.

≡ Salidas Esperadas.txt ×

1	Prueba 1
2	100
3	
4	prueba 2
5	10
6	
7	Prueba 3
8	110
9	150
10	
11	Prueba 4
12	60
13	
14	Prueba 5
15	120
16	
17	Prueba 6
18	10
19	55
20	
21	Prueba 7
22	<u>Inicio del programa</u>
23	El <u>numero</u> es mayor a 5
24	10
25	<u>Fin del programa</u>

## Prueba 1: Ensamblamos el código y todo funciona correctamente y nos da la salida esperada

The screenshot shows the Mars Debugger interface during a successful assembly run. The assembly code is displayed in the Text Segment window, showing instructions like addiu, sw, lw, beq, and syscall. The Registers window shows various CPU registers (r0 to r31, s0 to s19) initialized to zero. The Data Segment window shows memory starting at address 0x10010000. The Mars Messages window indicates the assembly was completed successfully.

## Prueba 2: Ensamblamos el código y funciona correctamente y ejecutamos

The screenshot shows the Mars Debugger interface after a successful assembly and execution. The assembly code is identical to the first test. The Registers window shows the state of the CPU after execution. The Data Segment window shows memory starting at address 0x10010000. The Mars Messages window shows the program has finished running.

Mars Messages Run I/O

```

10
-- program is finished running --

```

**Clear**

Prueba 3: Ensamblamos el código y funciona correctamente y ejecutamos

Edit Execute

Text Segment

Blpt	Address	Code	Basic	Source
0x00400000	0x2408000a	addiu \$8,\$0,0x0000000a	15: li \$t0, 10	
0x00400004	0x3c011001lui	\$1,0x00000001	16: sw \$t0, a	
0x00400008	0x3c011001lw	\$1,0x00000001	17: li \$t0, 20	
0x0040000c	0x3c0110014addiu	\$1,0x000000014	18: li \$t0, 20	
0x00400010	0x3c011001lui	\$1,0x00000001	19: sw \$t0, b	
0x00400014	0xac300008lw	\$5,0x00000008(\$1)		
0x00400018	0x24080008	addiu \$8,\$9,0x00000005	21: li \$t0, 5	
0x0040001c	0x3c011001lui	\$1,0x00000001001	22: sw \$t0, c	
0x00400020	0xac280001cm	\$8,0x00000001(\$1)		
0x00400024	0x3c011001lui	\$1,0x00000001001	24: lw \$t0, b	
0x00400028	0x3c280008lw	\$8,0x00000008(\$1)		
0x0040002c	0x3c011001lui	\$1,0x00000001001	25: lw \$t1, c	
0x00400030	0x8c29001clui	\$9,0x00000001(\$1)		
0x00400034	0x71095002mul	\$10,\$8,\$9	26: mul \$t2, \$t0, \$t1	
0x00400038	0x3c011001lui	\$1,0x00000001001	27: sw \$t2, t1	
0x0040003c	0xac2a0020sw	\$10,0x000000020(\$1)		
0x00400040	0x3c011001lui	\$1,0x00000001001	29: lw \$t0, a	

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100101a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100101c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Mars Messages Run I/O

```

Assemble: assembling C:\Users\toto_\PycharmProjects\Proyecto_python_ensamblador\pythonProject\examples\expected\03_prueba_GeneracionCodigoIntermedio.asm
Assemble: operation completed successfully.

```

Mars Messages Run I/O

```

110
150
-- program is finished running --

```

**Clear**

Prueba 4: Ensamblamos el código y funciona correctamente y ejecutamos

**Edit Execute**

**Text Segment**

Bkpt	Address	Code	Basic	Source
	0x00400000	0x2408001e	addiu \$8, \$0, 0x00000001	9: li \$t0, 30
	0x00400004	0x3c011001	lui \$1, 0x00001001	10: sw \$t0, T1
	0x00400008	0xac280004	sw \$8, 0x00000004(\$1)	
	0x0040000c	0x2408003c	addiu \$8, \$0, 0x0000003c	12: li \$t0, 60
	0x00400010	0x3c011001	lui \$1, 0x00001001	13: sw \$t0, T2
	0x00400014	0xac280004	sw \$8, 0x00000008(\$1)	
	0x00400018	0x2408003c	addiu \$8, \$0, 0x0000003c	15: li \$t0, 60
	0x0040001c	0x3c011001	lui \$1, 0x00001001	16: sw \$t0, resultado
	0x00400020	0xac280004	sw \$8, 0x00000000(\$1)	
	0x00400024	0x24020001	addiu \$2, \$0, 0x00000001	18: li \$r0, 1
	0x00400028	0x3c011001	lui \$1, 0x00001001	19: lw \$a0, resultado
	0x0040002c	0x8c2400001a	\$4, 0x00000000(\$1)	
	0x00400030	0x0000000c	syscall	20: syscall
	0x00400034	0x24040004	addiu \$4, \$0, 0x0000000a	21: li \$a0, 10
	0x00400038	0x24020008	addiu \$2, \$0, 0x0000000b	22: li \$v0, 11
	0x0040003c	0x0000000c	syscall	23: syscall
	0x00400040	0x24020004	addiu \$2, \$0, 0x0000000a	26: li \$v0, 10

**Data Segment**

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100101a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100101c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

0x10010000 (data)  Hexadecimal Addresses  Hexadecimal Values  ASCII

Mars Messages Run I/O

```
Assemble: assembling C:\Users\toto_\PycharmProjects\Proyecto_python_ensamblador\pythonProject\examples\expected\04_prueba_optimizacion.asm
Assemble: operation completed successfully.
```

**Clear**

**Mars Messages Run I/O**

60

-- program is finished running --

**Clear**

Prueba 5: Ensamblamos el código y funciona correctamente y ejecutamos

**Text Segment**

Bkpt	Address	Code	Basic	Source
	0x00400000	0x24080005	addiu \$8,\$0,rx00000005	11: li \$t0, 5
	0x00400004	0x3c011001	lui \$1,0x00001001	12: sw \$t0, numero
	0x00400008	0xac280000	sw \$8,0x00000000(\$1)	
	0x0040000c	0x24080001	addiu \$8,\$0,0x00000001	14: li \$t0, 1
	0x00400010	0x3c011001	lui \$1,0x00001001	15: sw \$t0, res
	0x00400014	0xac280000	sw \$8,0x0000000c(\$1)	
	0x00400018	0x3c011001	lui \$1,0x00001001	19: lw \$t0, numero
	0x0040001c	0x8c280000	lw \$8,0x00000000(\$1)	
	0x00400020	0x24090001	addiu \$9,\$0,0x00000001	20: li \$t1, 1
	0x00400024	0x0128502a	slt \$10,\$9,\$8	21: sgt \$t2,\$t0,\$t1
	0x00400028	0x3c011001	lui \$1,0x00001001	22: sw \$t2, Tl
	0x0040002c	0xac2a0010	lw \$10,0x00000010(\$1)	
	0x00400030	0x3c011001	lui \$1,0x00001001	24: lw \$t0, Tl
	0x00400034	0x8c280010	lw \$8,0x00000010(\$1)	
	0x00400038	0x11000016	beq \$8,\$0,0x00000016	25: beq \$t0, L2
	0x0040003c	0x3c011001	lui \$1,0x00001001	27: lw \$t0, res
	0x00400040	0x8c280000	lw \$8,0x0000000c(\$1)	

**Data Segment**

Address	Value (+0)	Value (+4)	Value (+8)	Value (+C)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100101a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100101c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

◀ ▶ 0x10010000 (.data)  Hexadecimal Addresses  Hexadecimal Values  ASCII

**Mars Messages** **Run I/O**

```
Assemble: assembling C:\Users\toto_\PycharmProjects\Proyecto_python_ensamblador\pythonProject\examples\expected\05_prueba_factorial.asm
Assemble: operation completed successfully.
```

**Mars Messages** **Run I/O**

```
120
-- program is finished running --
```

**Clear**

Prueba 6: Ensamblamos el código y funciona correctamente y ejecutamos

Screenshot of the Mars Assembly Editor showing the assembly code for the sumatoria.asm program:

```

Text Segment
Bkpt Address Code Basic Source
0x00400000 0x24080000 addiu $8,$0,0x00000000 13: li $t0, 0
0x00400004 0x3c011001lui $1,0x000001001 14: sw $t0, total
0x00400008 0xac280000lw $8,0x00000000($1)
0x0040000c 0x24080001 addiu $8,$0,0x00000001 16: li $t0, 1
0x00400010 0x3c011001lui $1,0x000001001 17: sw $t0, contador
0x00400014 0xac280000lw $8,0x00000000($1)
0x00400018 0x24080002 addiu $8,$0,0x00000000 19: li $t0, 11
0x0040001c 0x3c011001lui $1,0x000001001 20: sw $t0, limite
0x00400020 0xac280010lw $8,0x00000010($1)
0x00400024 0x3c011001lui $1,0x000001001 24: lw $t0, contador
0x00400028 0x8c280000lw $8,0x00000000($1)
0x0040002c 0x3c011001lui $1,0x000001001 25: lw $t1, limite
0x00400030 0x8c280010lw $9,0x00000010($1)
0x00400034 0x0109502aslt $10,$8,$9 26: slt $t2, $t0, $t1
0x00400038 0x3c011001lui $1,0x000001001 27: sw $t2, $t1
0x0040003c 0xac2a0018lw $10,0x00000018($1)
0x00402040 0x3c011001lui $1,0x000001001 29: lw $t0, $t1

```

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000000							
0x10010020	0x00000000							
0x10010040	0x00000000							
0x10010060	0x00000000							
0x10010080	0x00000000							
0x100100a0	0x00000000							
0x100100c0	0x00000000							
0x100100e0	0x00000000							
0x10010100	0x00000000							
0x10010120	0x00000000							
0x10010140	0x00000000							
0x10010160	0x00000000							
0x10010180	0x00000000							
0x100101a0	0x00000000							
Reinitializ.								

Run I/O

Mars Messages

Assemble: assembling C:\Users\toto\_\PycharmProjects\Proyecto\_python\_ensamblador\pythonProject\examples\expected\06\_prueba\_sumatoria.asm  
Assemble: operation completed successfully.

Clear

Screenshot of the Mars I/O window showing the execution results:

```

Mars Messages Run I/O
10
55
-- program is finished running --

```

Clear

Prueba 7: Ensamblamos el código y funciona correctamente y ejecutamos

The screenshot shows the Mars Assembler interface with the following sections:

- Text Segment:** Displays assembly code with columns for Bkpt, Address, Code, Basic, and Source.
- Data Segment:** Displays memory dump with columns for Address, Value (+0), Value (+4), Value (+8), Value (+c), Value (+10), Value (+14), Value (+18), and Value (+1c).
- Mars Messages:** Shows the assembly process log:
 

```

Assemble: assembling C:\Users\toto_\PycharmProjects\Proyecto_python_ensamblador\pythonProject\examples\expected\07_prueba_strings.asm
Assemble: operation completed successfully.

```

Como se puede observar todas las salidas esperadas fueron correctas cuando se ejecuto en el código assembler.

## Documentación técnica Arquitectura del Compilador

### 1. Origen del Código y Herramientas Utilizadas

El código fuente de este proyecto se divide en dos categorías según su procedencia:

- Módulos Generados Automáticamente:** Son archivos creados por la herramienta ANTLR4 a partir de la gramática definida en el archivo PythonSubset.g4. Estos módulos (Lexer, Parser y Visitor) implementan la teoría de autómatas y gramáticas libres de contexto, proporcionando la base estructural para el análisis del lenguaje.
- Módulos de Implementación Lógica:** Son los archivos desarrollados manualmente (VisitanteSemantico, CodigoIntermedio, Optimizador, GeneradorMIPS, etc.). Estos

contienen la lógica específica del compilador para realizar validaciones, gestión de memoria, optimización y traducción a lenguaje ensamblador.

## 2. Descripción de Módulos por Fase

### Fase 1: Análisis Léxico y Sintáctico (Frontend)

Esta fase convierte el código fuente en una estructura jerárquica comprensible para la máquina.

- **PythonSubsetLexer.py (Generado por ANTLR):**
  - **Función:** Es el analizador léxico. Su tarea es leer el flujo de caracteres de entrada y agruparlos en Tokens (identificadores, palabras reservadas como if o while, números, operadores, etc.). Filtra elementos irrelevantes como espacios en blanco y comentarios.
- **PythonSubsetParser.py (Generado por ANTLR):**
  - **Función:** Es el analizador sintáctico. Toma los tokens del lexer y verifica que sigan el orden gramatical definido. Construye el Árbol de Análisis Sintáctico (Parse Tree), una estructura de datos que representa la jerarquía gramatical del programa (ej. una sentencia if contiene una expresión condicional y un bloque de código).
- **PythonSubsetVisitor.py (Generado por ANTLR):**
  - **Función:** Provee la interfaz base del patrón de diseño Visitor. Define los métodos vacíos para recorrer el árbol sintáctico. Las clases lógicas del proyecto heredan de este archivo para implementar sus propias acciones al visitar cada nodo del árbol.

### Fase 2: Análisis Semántico

Esta fase valida que el programa tenga sentido lógico, más allá de estar bien escrito gramaticalmente.

- **Tabla\_de\_Simbolos.py:**
  - **Función:** Estructura de datos que simula la memoria durante la compilación. Utiliza una pila de diccionarios para gestionar los alcances (scopes). Permite registrar variables y funciones, y verificar su existencia respetando la visibilidad local y global.
- **VisitanteSemantico.py:**
  - **Función:** Recorre el árbol sintáctico para aplicar las reglas de tipo y contexto. Verifica que las variables usadas hayan sido declaradas previamente, gestiona la creación de nuevos entornos al entrar en funciones y asegura que no se realicen operaciones inválidas entre tipos incompatibles.

### Fase 3: Generación de Código Intermedio (GCI)

Esta fase "aplana" la estructura jerárquica del árbol para facilitar la traducción y optimización.

- **Codigointermedio.py:**

- **Función:** Transforma el árbol sintáctico en una lista lineal de instrucciones conocida como Código de Tres Direcciones (Cuádruplos). Descompone expresiones complejas (ej.  $x = a + b * c$ ) en pasos simples utilizando variables temporales ( $T1, T2$ ) y traduce las estructuras de control (if, while) en etiquetas ( $L1$ ) y saltos (GOTO, IF\_FALSE).

### Fase 4: Optimización Independiente de la Máquina

Esta fase mejora la eficiencia del código antes de llegar a la arquitectura destino.

- **Optimizador.py:**

- **Función:** Analiza la lista de cuádruplos para encontrar ineficiencias. Implementa técnicas como la Propagación de Constantes y el Plegado de Constantes (Constant Folding), pre-calculando operaciones aritméticas cuyos valores son conocidos en tiempo de compilación (ej. convertir  $x = 2 + 3$  directamente en  $x = 5$ ), reduciendo así el número de instrucciones finales.

### Fase 5: Generación de Código Objeto (Backend)

Esta fase realiza la síntesis final hacia el lenguaje de la máquina destino.

- **GeneradorMIPS.py:**

- **Función:** Traduce los cuádruplos optimizados a instrucciones del lenguaje ensamblador MIPS. Gestiona la asignación de memoria estática (sección .data), la traducción de operaciones aritméticas a instrucciones de registro (add, lw, sw), la implementación de saltos (bne, j) y el uso de llamadas al sistema (syscall) para entrada y salida de datos.

## Orquestación

- **main.py:**

- **Función:** Es el punto de entrada del compilador. Integra todas las fases anteriores en un pipeline secuencial: lee el código fuente, invoca al lexer/parser, ejecuta las validaciones semánticas, genera el código intermedio, lo optimiza y finalmente escribe el archivo de salida .asm.

## Código fuente

### PythonSubset.g4

```
grammar PythonSubset;
program
    : (functionDef | statement)* EOF
    ;
// Definición de función: def nombre(arg1, arg2) { código }
functionDef
    : 'def' ID '(' parameterList? ')' block
    ;
// Lista de parámetros: a, b, c
parameterList
    : ID (',' ID)*
    ;
statement
    : assignmentStmt
    | printStmt
    | ifStmt
    | whileStmt
    | returnStmt // Nueva sentencia para funciones
    | functionCall // Llamada a función como sentencia
    | block
    ;
assignmentStmt
    : ID '=' expression
    ;
printStmt
    : 'print' '(' expression ')'
    ;
ifStmt
    : 'if' expression ':' statement ('else' ':' statement)?
    ;
whileStmt
    : 'while' expression ':' statement
    ;
returnStmt
    : 'return' expression?
    ;
block
    : '{' statement* '}'
    ;
```

```

expression
: expression ('*' | '/') expression    # MulDiv
| expression ('+' | '-') expression    # AddSub
| expression ('>' | '<' | '==' ) expression # Relational
| ID '(' expressionList? ')'          # FunctionCallExpr
| STRING                         # StringLiteral
| INT                            # Number
| ID                             # Id
| '(' expression ')'            # Parens
;
// Lista de argumentos para llamar función: (1, x, 3)
expressionList
: expression (',' expression)*
;
// Llamada a función aislada (para usarla en statement)
functionCall
: ID '(' expressionList? ')'
;
// LEXER (Tokens)
DEF   : 'def';
RETURN : 'return';
IF    : 'if';
ELSE  : 'else';
WHILE : 'while';
PRINT : 'print';
STRING : """.*?"";
ID    : [a-zA-Z_][a-zA-Z0-9_]* ;
INT   : [0-9]+ ;
COMMENT : '#' ~[\r\n]* -> skip ;
WS    : [\t\r\n]+ -> skip ;
// Operadores y símbolos
PLUS  : '+';
MINUS : '-';
MUL   : '*';
DIV   : '/';
ASSIGN : '=';
LPAREN : '(';
RPAREN : ')';
LBRACE : '{';
RBRACE : '}';
COLON : ':';
COMMA : ',';
GT    : '>';

```

```
LT    : '<';
EQ    : '==';
```

## VisitanteSemántico.py

```
from PythonSubsetVisitor import PythonSubsetVisitor
from Tabla_de_Simbolos import TablaSimbolos

class VisitanteSemantico(PythonSubsetVisitor):
    def __init__(self):
        self.tabla_simbolos = TablaSimbolos()

    # --- Visitante del Programa Principal ---
    def visitProgram(self, ctx):
        self.visitChildren(ctx)

    # --- Visitante de Definición de Funciones ---
    def visitFunctionDef(self, ctx):
        nombre_funcion = ctx.ID().getText()
        self.tabla_simbolos.definir(nombre_funcion, 'FUNCION')

        self.tabla_simbolos.crear_alcance()

        if ctx.parameterList():
            for param in ctx.parameterList().ID():
                nombre_parametro = param.getText()
                self.tabla_simbolos.definir(nombre_parametro, 'PARAMETRO')

        self.visit(ctx.block())
        self.tabla_simbolos.salir_alcance()

    # --- Visitante de Asignaciones ---
    def visitAssignmentStmt(self, ctx):
        nombre_variable = ctx.ID().getText()

        # Primero visitamos la expresión para validar que las variables usadas existan
        self.visit(ctx.expression())

        self.tabla_simbolos.definir(nombre_variable, 'ENTERO')

    # --- Visitante de Identificadores en Expresiones ---
    def visitId(self, ctx):
        nombre_variable = ctx.getText()

        info_tipo = self.tabla_simbolos.buscar(nombre_variable)
```

```

# 1. Verificamos existencia
if info_tipo is None:
    linea = ctx.start.line
    raise Exception(f"Error Semántico en línea {linea}: La variable '{nombre_variable}' no ha sido definida.")

# 2. Verificamos que NO sea una función usada sin paréntesis
if info_tipo == 'FUNCION':
    linea = ctx.start.line
    raise Exception(
        f"Error Semántico en línea {linea}: '{nombre_variable}' es una función, no una variable. ¿Olvidaste los paréntesis ()?")
return self.visitChildren(ctx)

```

## Codigointermedio.py

```
from PythonSubsetVisitor import PythonSubsetVisitor
```

```

class Cuadruplo:
    def __init__(self, op, arg1, arg2, res):
        self.op = op
        self.arg1 = arg1
        self.arg2 = arg2
        self.res = res

    def __repr__(self):
        return f"({self.op}, {self.arg1}, {self.arg2}, {self.res})"

class VisitanteIntermedio(PythonSubsetVisitor):
    def __init__(self):
        self.cuadruvos = []
        self.contador_temporales = 0
        self.contador_etiquetas = 0

    def nuevo_temporal(self):
        self.contador_temporales += 1
        return f"T{self.contador_temporales}"

    def nueva_etiqueta(self):
        """Genera etiquetas para saltos (GOTO): L1, L2..."""
        self.contador_etiquetas += 1

```

```

    return f'L{self.contador_etiquetas}'

# --- MANEJO DE VALORES ---

def visitNumber(self, ctx):
    # Si vemos un número, simplemente retornamos su texto (ej: "5")
    return ctx.getText()

def visitId(self, ctx):
    # Si vemos una variable, retornamos su nombre (ej: "x")
    return ctx.getText()

def visitStringLiteral(self, ctx):
    # Devuelve el texto con comillas
    return ctx.getText()

# --- OPERACIONES ARITMÉTICAS ---

def visitMulDiv(self, ctx):
    # 1. Visitar hijos recursivamente para obtener sus direcciones
    izq = self.visit(ctx.expression(0)) # Lado izquierdo
    der = self.visit(ctx.expression(1)) # Lado derecho

    # 2. Crear temporal para guardar resultado
    temporal = self.nuevo_temporal()

    # 3. Obtener operador (* o /)
    op = ctx.getChild(1).getText()

    # 4. Generar Cuádruplo
    self.cuadraplos.append(Cuadruplo(op, izq, der, temporal))

    # 5. Retornar el nombre del temporal para que lo use el padre
    return temporal

def visitAddSub(self, ctx):
    izq = self.visit(ctx.expression(0))
    der = self.visit(ctx.expression(1))
    temporal = self.nuevo_temporal()
    op = ctx.getChild(1).getText()
    self.cuadraplos.append(Cuadruplo(op, izq, der, temporal))
    return temporal

# --- ASIGNACIÓN ---

```

```

def visitAssignmentStmt(self, ctx):
    nombre_variable = ctx.ID().getText()

    # Visitamos la expresión de la derecha para obtener el valor/temporal final
    valor = self.visit(ctx.expression())

    # Generamos cuádruplo de asignación
    self.cuadruplos.append(Cuadruplo('=', valor, None, nombre_variable))

# --- PRINT ---
def visitPrintStmt(self, ctx):
    valor = self.visit(ctx.expression())
    self.cuadruplos.append(Cuadruplo('PRINT', valor, None, None))

# --- PARÉNTESIS ---
def visitParens(self, ctx):
    # Simplemente visitamos la expresión de adentro y devolvemos su resultado.
    return self.visit(ctx.expression())

# --- COMPARACIONES (>, <, ==) ---
def visitRelational(self, ctx):
    izq = self.visit(ctx.expression(0))
    der = self.visit(ctx.expression(1))
    temporal = self.nuevo_temporal()
    op = ctx.getChild(1).getText() # >, <, ==

    self.cuadruplos.append(Cuadruplo(op, izq, der, temporal))
    return temporal

# --- IF STATEMENT ---
def visitIfStmt(self, ctx):
    # if expresion : statement (else : statement)?

    # 1. Evaluar la condición
    condicion = self.visit(ctx.expression())

    # 2. Generar etiqueta para SALIR (o ir al else)
    etiqueta_falso = self.nueva_etiqueta() # L1

    # 3. Generar salto
    self.cuadruplos.append(Cuadruplo('IF_FALSE', condicion, None, etiqueta_falso))

    # 4. Generar el código del bloque verdadero

```

```

self.visit(ctx.statement(0))

# Revisar si hay else
if len(ctx.statement()) > 1:
    etiqueta_fin = self.nueva_etiqueta()

    # Al terminar el bloque True, saltamos al final para no ejecutar el Else
    self.cuadraplos.append(Cuadruplo('GOTO', None, None, etiqueta_fin))

    self.cuadraplos.append(Cuadruplo('LABEL', None, None, etiqueta_falso))

    self.visit(ctx.statement(1))

    # Ponemos la etiqueta del final
    self.cuadraplos.append(Cuadruplo('LABEL', None, None, etiqueta_fin))
else:
    # Si no hay else, simplemente ponemos la etiqueta de salida aquí
    self.cuadraplos.append(Cuadruplo('LABEL', None, None, etiqueta_falso))

# --- WHILE STATEMENT ---
def visitWhileStmt(self, ctx):
    etiqueta_inicio = self.nueva_etiqueta()
    etiqueta_fin = self.nueva_etiqueta()

    # 1. Poner etiqueta de inicio
    self.cuadraplos.append(Cuadruplo('LABEL', None, None, etiqueta_inicio))

    condicion = self.visit(ctx.expression())

    # 3. Si es falso, salimos del ciclo
    self.cuadraplos.append(Cuadruplo('IF_FALSE', condicion, None, etiqueta_fin))

    # 4. Ejecutar el cuerpo del ciclo
    self.visit(ctx.statement())

    # 5. Volver al inicio automáticamente
    self.cuadraplos.append(Cuadruplo('GOTO', None, None, etiqueta_inicio))

    # 6. Etiqueta de salida
    self.cuadraplos.append(Cuadruplo('LABEL', None, None, etiqueta_fin))

```

## Optimizador.py

```

class Optimizador:
    def __init__(self):

```

```

self.constantes = {}

def es_numero(self, s):
    try:
        float(s)
        return True
    except ValueError:
        return False

def es_temporal(self, s):
    return s and s.startswith('T') and s[1:].isdigit()

def optimizar(self, lista_cuadraplos):
    cuadraplos_optimizados = []

    for cuad in lista_cuadraplos:
        # 1. PROPAGACIÓN DE CONSTANTES
        if cuad.arg1 in self.constantes:
            cuad.arg1 = self.constantes[cuad.arg1]

        if cuad.arg2 in self.constantes:
            cuad.arg2 = self.constantes[cuad.arg2]

        # 2. PLEGADO DE CONSTANTES
        if cuad.op in ['+', '-', '*', '/'] and self.es_numero(cuad.arg1) and
           self.es_numero(cuad.arg2):

            resultado = 0
            val1 = float(cuad.arg1)
            val2 = float(cuad.arg2)

            if cuad.op == '+':
                resultado = val1 + val2
            elif cuad.op == '-':
                resultado = val1 - val2
            elif cuad.op == '*':
                resultado = val1 * val2
            elif cuad.op == '/':
                resultado = val1 / val2

            if resultado.is_integer():
                resultado = int(resultado)

            cuad.op = '='

```

```

        cuad.arg1 = str(resultado)
        cuad.arg2 = None

        if self.es_temporal(cuad.res):
            self.constantes[cuad.res] = str(resultado)

        elif cuad.op == '=' and self.es_numero(cuad.arg1):
            if self.es_temporal(cuad.res):
                self.constantes[cuad.res] = cuad.arg1

        cuadruplos_optimizados.append(cuad)

    return cuadruplos_optimizados

```

## GeneradorMIPS.py

```

class GeneradorMIPS:
    def __init__(self):
        self.variables = set()
        self.cadenas = {}
        self.contador_cadenas = 0

    def es_numero(self, s):
        try:
            float(s)
            return True
        except ValueError:
            return False

    def es_cadena(self, s):
        return s and s.startswith("") and s.endswith("")

    def recolectar_variables(self, lista_cuadruplos):

        for cuad in lista_cuadruplos:
            # --- 1. RECOLECCIÓN DE VARIABLES NUMÉRICAS ---
            for arg in [cuad.arg1, cuad.arg2, cuad.res]:
                # Debe existir, NO ser número, NO ser string
                if arg and not self.es_numero(arg) and not self.es_cadena(arg):
                    if cuad.op in ['LABEL', 'GOTO', 'IF_FALSE'] and arg == cuad.res:
                        continue # Es una etiqueta, ignorar

                self.variables.add(arg)

            # --- 2. RECOLECCIÓN DE CADENAS (STRINGS) ---

```

```

if self.es_cadena(cuad.arg1) and cuad.arg1 not in self.cadenas:
    self.contador_cadenas += 1
    etiqueta = f"str_{self.contador_cadenas}"
    self.cadenas[cuad.arg1] = etiqueta

def generar(self, lista_cuadruplos, nombre_archivo="resultado.asm"):
    self.recolectar_variables(lista_cuadruplos)

    with open(nombre_archivo, 'w') as f:
        # --- SECCIÓN DE DATOS (.data) ---
        f.write(".data\n")

        # 1. Declarar Variables Numéricas (.word)
        for var in self.variables:
            f.write(f"    {var}: .word 0\n")

        # 2. Declarar Cadenas (.asciiz)
        for contenido, etiqueta in self.cadenas.items():
            f.write(f"    {etiqueta}: .asciiz {contenido}\n")

        # --- SECCIÓN DE CÓDIGO (.text) ---
        f.write("\n.text\n")
        f.write("main:\n")

        for cuad in lista_cuadruplos:
            f.write(f"    # {cuad}\n")
            op = cuad.op

            # --- OPERACIONES ARITMÉTICAS ---
            if op in ['+', '-', '*', '/', '<', '>', '==']:
                # Cargar Operando 1
                if self.es_numero(cuad.arg1):
                    f.write(f"    li $t0, {cuad.arg1}\n")
                else:
                    f.write(f"    lw $t0, {cuad.arg1}\n")

                # Cargar Operando 2
                if self.es_numero(cuad.arg2):
                    f.write(f"    li $t1, {cuad.arg2}\n")
                else:
                    f.write(f"    lw $t1, {cuad.arg2}\n")

                # Operar
                if op == '+':

```

```

        f.write("    add $t2, $t0, $t1\n")
    elif op == '-':
        f.write("    sub $t2, $t0, $t1\n")
    elif op == '*':
        f.write("    mul $t2, $t0, $t1\n")
    elif op == '/':
        f.write("    div $t0, $t1\n")
        f.write("    mflo $t2\n")
    elif op == '<':
        f.write("    slt $t2, $t0, $t1\n")
    elif op == '>':
        f.write("    sgt $t2, $t0, $t1\n")
    elif op == '==':
        f.write("    seq $t2, $t0, $t1\n")

# Guardar resultado
f.write(f"    sw $t2, {cuad.res}\n")

# --- ASIGNACIÓN (=) ---
elif op == '=':
    if self.es_numero(cuad.arg1):
        f.write(f"    li $t0, {cuad.arg1}\n")
    else:
        f.write(f"    lw $t0, {cuad.arg1}\n")
    f.write(f"    sw $t0, {cuad.res}\n")

# --- PRINT (Manejo híbrido String/Int) ---
elif op == 'PRINT':
    # CASO A: Imprimir Texto
    if self.es_cadena(cuad.arg1):
        f.write("    li $v0, 4\n") # Syscall 4: Print String
        etiqueta = self.cadenas[cuad.arg1] # Recuperar etiqueta (str_1)
        f.write(f"    la $a0, {etiqueta}\n") # Cargar dirección

    # CASO B: Imprimir Número
    else:
        f.write("    li $v0, 1\n")
        if self.es_numero(cuad.arg1):
            f.write(f"    li $a0, {cuad.arg1}\n")
        else:
            f.write(f"    lw $a0, {cuad.arg1}\n")

f.write("    syscall\n")

```

```

        f.write("    li $a0, 10\n")
        f.write("    li $v0, 11\n")
        f.write("    syscall\n")

# --- SALTOS Y ETIQUETAS ---
elif op == 'LABEL':
    f.write(f"{cuad.res}:\n")

elif op == 'GOTO':
    f.write(f"    j {cuad.res}\n")

elif op == 'IF_FALSE':
    if self.es_numero(cuad.arg1):
        f.write(f"    li $t0, {cuad.arg1}\n")
    else:
        f.write(f"    lw $t0, {cuad.arg1}\n")
    f.write(f"    beqz $t0, {cuad.res}\n")

f.write("\n    # Fin del programa\n")
f.write("    li $v0, 10\n")
f.write("    syscall\n")

```

## Tabla\_de\_Simbolos.py

```

class TablaSimbolos:
    def __init__(self):
        self.pila_alcances = [{}]

    def crear_alcance(self):
        """Crea un nuevo nivel de alcance"""
        self.pila_alcances.append({})

    def salir_alcance(self):
        """Destruye el alcance actual"""
        self.pila_alcances.pop()

    def definir(self, nombre, tipo_info):
        """Guarda una variable en el alcance actual"""
        scope = self.pila_alcances[-1]
        scope[nombre] = tipo_info
        print(f"DEBUG: Definida variable '{nombre}' de tipo '{tipo_info}' en alcance nivel {len(self.pila_alcances)}")

    def buscar(self, nombre):
        """Busca una variable. Empieza en el alcance actual y va subiendo hasta el global."""

```

```

for scope in reversed(self.pila_alcances):
    if nombre in scope:
        return scope[nombre]
return None # No encontrada

```

## Tabla\_de\_Simbolos.py

```

class TablaSimbolos:
    def __init__(self):
        self.pila_alcances = [{}]

    def crear_alcance(self):
        """Crea un nuevo nivel de alcance"""
        self.pila_alcances.append({})

    def salir_alcance(self):
        """Destruye el alcance actual"""
        self.pila_alcances.pop()

    def definir(self, nombre, tipo_info):
        """Guarda una variable en el alcance actual"""
        scope = self.pila_alcances[-1]
        scope[nombre] = tipo_info
        print(f"DEBUG: Definida variable '{nombre}' de tipo '{tipo_info}' en alcance nivel {len(self.pila_alcances)}")

    def buscar(self, nombre):
        """Busca una variable. Empieza en el alcance actual y va subiendo hasta el global."""
        for scope in reversed(self.pila_alcances):
            if nombre in scope:
                return scope[nombre]
        return None # No encontrada

```