



USAC
TRICENTENARIA
Universidad de San Carlos de Guatemala

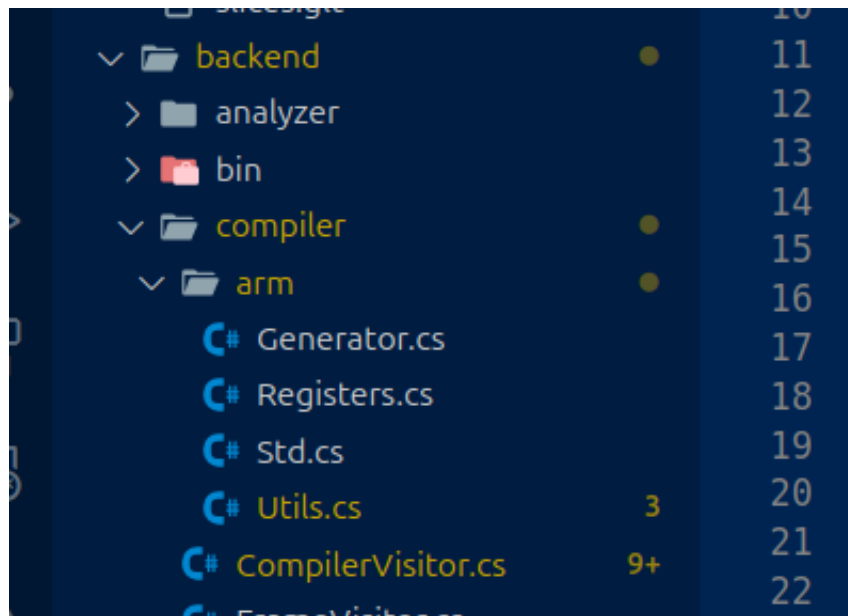
PROYECTO 2

MANUAL TÉCNICO

Organización de Lenguajes y Compiladores 2

Miguel Adrian Tubac Agustin
202101927

Descripción de la solución

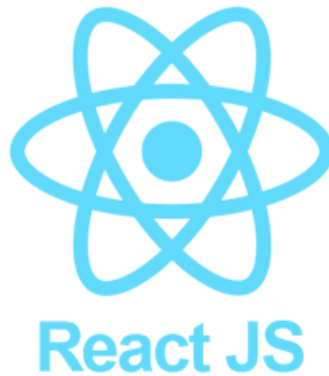


A screenshot of a file explorer window showing the directory structure of a project. The tree view is expanded to show the 'compiler' directory, which contains an 'arm' subdirectory. The 'arm' directory contains several C# files: 'Generator.cs', 'Registers.cs', 'Std.cs', 'Utils.cs', 'CompilerVisitor.cs', and 'FrameVisitor.cs'. The 'Utils.cs' file has a count of 3, and 'CompilerVisitor.cs' has a count of 9+.

✓ folder backend	●	10
> folder analyzer		11
> folder bin		12
✓ folder compiler	●	13
✓ folder arm	●	14
C# Generator.cs		15
C# Registers.cs		16
C# Std.cs		17
C# Utils.cs	3	18
C# CompilerVisitor.cs	9+	19
C# FrameVisitor.cs		20

El proyecto fundamentalmente realiza la generación de código ensamblador ARM64, lo que permitirá comprender el proceso de traducción de un lenguaje de alto nivel a instrucciones de bajo nivel en una arquitectura de conjunto de instrucciones reducido (RISC).

Lenguaje Utilizado:



Los requerimientos necesarios para la edición y ejecución del programa son la utilización del lenguaje de programación denominado C#, en el cual se crea el analizador. Por parte del cliente se desarrolla con el framework de React con JavaScript. Para la ejecución del código ensamblador creado se utiliza QEMU, con dicha herramienta se linkea y se genera el ejecutable.

Clases

CompilerVisitor: en esta clase se definen las propiedades de los métodos visiti que generan el código ensamblador.

```
1 referencia | You, hace 2 horas | 1 author (You)
public class CompilerVisitor : LanguageBaseVisitor<Object?> //
{
    //Se genera una instancia de la clase de generador de arm6
    99+ referencias
    public ArmGenerator c = new ArmGenerator();

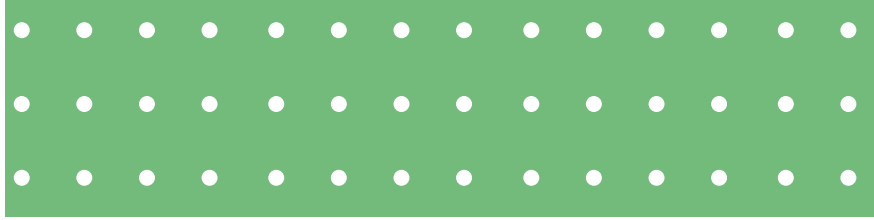
    //Esto me servira para guardar el nombre de las etiquetas
    //Ya que estas seran usadas para las sentecias de transfer
    8 referencias
    private String? continueLabel = null;
    11 referencias
    private String? breakLabel = null;
```

```
3 referencias | You, hace 3 días | 1 author (You)
public class FunctionMetadata
{
    3 referencias
    public int FrameSize;
    2 referencias
    public StackObject.StackObjectType ReturnType
}
```

FunctionMetadata: en esta clase se definen las propiedades de las funciones que se realizaran en la compilacion.

StackObject: esta es la clase principal para almacenar la estructura de los datos en la pila virtual del mismo programa de C#.

```
99+ referencias | You, hace 3 días | 1 author (You)
public class StackObject
{
    99+ referencias | 61 referencias | 65 referencias | 14 re
    public enum StackObjectType {Int,
    72 referencias
    public StackObjectType Type {get; s
    14 referencias
    public int Length {get; set;}
    //Este es el numeor del entorno
    14 referencias
    public int Depth {get; set;}
```



```
2 referencias | You, la semana pasada | 1 author (You)
public class ArmGenerator
{
    //Esto es para el codigo en general
    83 referencias
    public List<string> instrucciones =
    //Esto es para las funciones
    2 referencias
    public List<string> funcInstrucciones
    32 referencias
    private readonly StandardLibrary st
```

ArmGenerator: en esta clase se definen los métodos que me generaran el código ensamblador, por lo tanto, cuenta con una lista que almacena el texto.

Register: en esta clase se definen los registros que cuenta la estructura de ARM, es decir, los registros de X0 a X31.

```
public static class Register
{
    // General purpose registers
    99+ referencias
    public static string X0 => "x0";
    69 referencias
    public static string X1 => "x1";
    21 referencias
    public static string X2 => "x2";
    2 referencias
    public static string X3 => "x3";
```

```
2 referencias | You, la semana pasada | 1 author (You)
public class StandardLibrary
{
    2 referencias
    private readonly HashSet<string> U
    31 referencias
    public void Use(string function)
    {
        UsedFunctions.Add(function);
    }
}
```

StandardLibrary: en esta clase se definen las funciones auxiliares para mi código. Entre las principales se encuentra la de imprimir numeros.

Utils: en esta clase se definen las funciones auxiliares que me servirán para la generación del código, como por ejemplo obtener el registró adecuado.

```
2 referencias | 100, hace 4 días | 1 author (100)
public static class Utils
{
    1 referencia
    public static List<byte> StringTo1B
    {
        var resultado = new List<byte>(
        int elementIndex = 0;

        while (elementIndex < str.Length
        {
```

```
3 referencias | 100, anteayer | 1 author (100)
public class FrameVisitor : LanguageBas
{
    4 referencias
    public List<FrameElement> Frame;
    5 referencias
    public int LocalOffset;
    3 referencias
    public int BaseOffset;

    1 referencia
```

FrameVisitor: en esta clase se definen las propiedades que ayudaran a saber la cantidad de parámetros que se utilizaran en las funciones.

FrameElement: en esta clase se la estructura de los elementos que se guardaran de las funciones. Ya que nos brinda el desplazamiento de las variables.

```
public class FrameElement
{
    2 referencias
    public string Name {get; set;}
    2 referencias
    public int Offset {get; set;}

    2 referencias
    public FrameElement(string name, int
    {
        Name = name;
```