



UNIVERSIDADE FEDERAL DE UBERLÂNDIA  
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO

Construção de Compiladores  
Av. João Naves de Ávila 2121, Campus Santa Mônica

---

# Construção de Compiladores JavaScript para CLR

**Nome:** Lara Carolina Luciana e Oliveira

**Matrícula:** 11511BCC038

**Email:** lara.carolinnaa@gmail.com

## SUMÁRIO

---

# Sumário

<b>1</b>	<b>Introdução</b>	<b>3</b>
<b>2</b>	<b>CLR</b>	<b>3</b>
<b>3</b>	<b>Linguagem Assembly (CIL)</b>	<b>4</b>
<b>4</b>	<b>Códigos em Assembly CIL ( comentados)</b>	<b>7</b>
4.1	Módulo mínimo que caracteriza um programa . . . . .	7
4.2	Hello World . . . . .	9
4.3	Execução baseada em pilha . . . . .	10
4.4	Condições e Iterações . . . . .	13
4.5	Fatorial . . . . .	14
<b>5</b>	<b>JavaScript</b>	<b>17</b>
<b>6</b>	<b>Preparação do Ambiente</b>	<b>18</b>
6.1	OCaml . . . . .	18
6.2	Mono . . . . .	18
6.3	Instalação . . . . .	18
6.4	Compilação . . . . .	18
6.4.1	Execução . . . . .	19
6.4.2	Geração do Assembly . . . . .	19
6.4.3	Geração do Executável . . . . .	19
<b>7</b>	<b>Códigos</b>	<b>19</b>
7.1	Nano 01 . . . . .	19
7.2	Nano 02 . . . . .	21
7.3	Nano 03 . . . . .	22
7.4	Nano 04 . . . . .	24
7.5	Nano 05 . . . . .	25
7.6	Nano 06 . . . . .	27
7.7	Nano 07 . . . . .	29
7.8	Nano 08 . . . . .	31
7.9	Nano 09 . . . . .	33
7.10	Nano 10 . . . . .	35

---

*UFU, Universidade Federal de Uberlândia, Minas Gerais, Brasil*

## SUMÁRIO

---

7.11 Nano 11 . . . . .	37
7.12 Nano 12 . . . . .	39
7.13 Micro 01 . . . . .	42
7.14 Micro 02 . . . . .	45
7.15 Micro 03 . . . . .	50
7.16 Micro 04 . . . . .	53
7.17 Micro 05 . . . . .	57
7.18 Micro 06 . . . . .	61
7.19 Micro 07 . . . . .	66
7.20 Micro 08 . . . . .	70
7.21 Micro 09 . . . . .	73
7.22 Micro 10 . . . . .	77
7.23 Micro 11 . . . . .	80
<b>8 Construção do Compilador</b>	<b>85</b>
8.1 Analisador Léxico . . . . .	85
8.2 Especificação Lexical para MiniJavaScript . . . . .	85
8.3 Geração do Código do Analisador Léxico . . . . .	90
8.4 Teste do Analisador Léxico . . . . .	91
<b>9 Bibliografia</b>	<b>93</b>

## 1 Introdução

Este relatório tem como propósito detalhar o processo de instalação e uso das ferramentas relacionadas à plataforma selecionada para o projeto e apresentar a linguagem Assembly associada a ela (CIL). As tecnologias aqui apresentadas irão possibilitar a utilização da CLR para criar um compilador para a linguagem Javascript, usando a linguagem Ocaml.

Para contextualização do assunto, inicialmente será apresentada uma introdução sobre a CLR, a linguagem Assembly utilizada e o Javascript. Em seguida, serão apresentadas as ferramentas instaladas. Por fim, serão mostrados pseudo códigos convertidos em Javascript e em C#, os quais serão compilados utilizando uma plataforma que segue o modelo de compilação CLR para a geração dos respectivos códigos em Assembly CIL.

## 2 CLR

A CLR (Common Language Runtime) é o componente da máquina virtual da plataforma .Net responsável por executar as aplicações desenvolvidas em .Net. Além da compilação, a CLR também oferece outros serviços como segurança e tratamento de exceções. A compilação é feita em tempo de execução.

O processo de compilação executado pela CLR pode ser descrito da seguinte maneira: ao passar pelo compilador, o código fonte dá origem ao código gerenciado, o qual é descrito em uma linguagem de programação de baixo nível (Assembly) chamada CIL (Common Intermediate Language). A CLR então, através da compilação just-in-time (compilação em tempo de execução), converte o código gerenciado para linguagem de máquina.

### 3 LINGUAGEM ASSEMBLY (CIL)

---

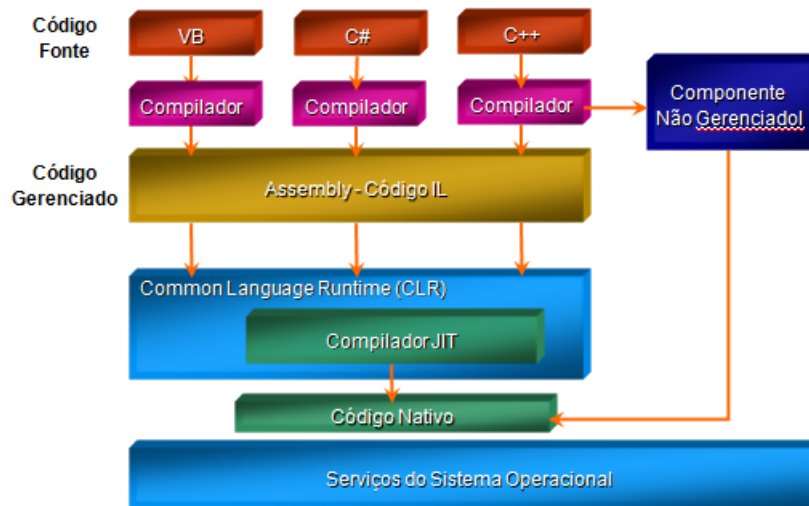


Figura 1: Modelo de Compilação CLR

A CLR suporta algumas linguagens como C#, C++, Visual Basic etc, porém não compila códigos escritos na linguagem selecionada, JavaScript. Desta forma, além dos códigos em Javascript, também serão apresentados no relatório códigos em C#, os quais de fato serão compilados pela CLR para a obtenção do código em linguagem Assembly correspondente.

## 3 Linguagem Assembly (CIL)

A CIL (Common Intermediate Language) é a linguagem Assembly associada à CLR. As instruções definidas pela CIL são convertidas para linguagem de máquina pelo compilador JIT.

A CIL é uma linguagem orientada a objetos, ou seja, envolve conceitos como criação de objetos, classes e chamada de métodos, e é baseada em pilha (dados são retirados de uma pilha e não de registradores).

A seguir são apresentadas as instruções do conjunto de instruções da CIL.

- add: retira os dois elementos do topo da pilha e coloca o resultado no topo.
- add.ovf: retira os dois elementos do topo da pilha e coloca o resultado no topo, verificando overflow.

### 3 LINGUAGEM ASSEMBLY (CIL)

---

- `and`: retira os dois elementos do topo da pilha, faz a operação `and bit` a bit e coloca o resultado no topo da pilha
- `arglist`: retorna o identificador da lista de argumentos para o método atual
- `beq`: pula para o endereço se os elementos topo da pilha são iguais
- `bge`: pula para o endereço se o topo da pilha é menor que o seu antecessor
- `bge.un.` : igual ao `bge`, mas a comparação é feita sem sinal.
- `bgt`: pular para o endereço se o segundo elemento da pilha é maior que o topo.
- `ble`: pula para o endereço se o topo é menor ou igual ao seu antecessor.
- `ble.un`: igual ao `ble`, mas a comparação é feita sem sinal.
- `blt` : pula pra o endereço se o o topo é menor que seu antecessor.
- `blt.un`: igual ao `blt`, mas a comparação é feita sem sinal.
- `bne`: pula para o endereço se o topo não for igual ou não ordenada.
- `bne.un`: igual ao `bne`, mas a comparação é feita sem sinal.
- `br`: salto incondicional
- `break`: instrução de breakpoint
- `brfalse`: pula para o endereço se falso, nulo, ou zero.
- `brtrue`: pula para o endereço se verdadeiro, diferente de zero, ou não nulo.
- `call`: chamada de método.
- `calli`: chamada de método indireto.
- `ceq`: compara se igual
- `cgt`: compara se maior que.

### 3 LINGUAGEM ASSEMBLY (CIL)

---

- cgt.un: igual ao cgt, mas a comparação é feita sem sinal.
- clt: compara se menor que
- clt.un igual ao clt, mas a comparação é feita sem sinal.
- conv : conversão de dados.
- conv.ovf: igual ao conv, mas com sinalização de overflow.
- conv.ovf.un: igual ao conv.of, mas a comparação é feita sem sinal.
- cpblc: copia dados da memória para a memória.
- div: divisão.
- div.un: igual ao div, mas a comparação é feita sem sinal.
- dup: duplica o valor da pilha.
- jmp: pula para um método.
- jmp: pular para um ponteiro de método.
- ldarg: carrega um argumento na pilha.
- ldarga: carrega um argumento a partir de um endereço.
- ldc: carrega uma constante numérica.
- Idelem: carrega um elemento do vetor no index para topo da pilha.
- Idelema: carrega o endereço do vetor na posicao index na pilha.
- Idstr: poe no topo da pilha a string.
- mul: multiplica os dois valores de cima da pilha e empilha o resultado.
- mul.ovf: igual ao mul, mas é feita sinalização de overflow.
- neg: altera o sinal do elemento do topo da pilha.
- Newarr: cria um array com o tipo definido onde seu tamanho está no topo da pilha.

## 4 CÓDIGOS EM ASSEMBLY CIL ( COMENTADOS)

---

- not: inverte os bits do elemento do topo da pilha.
- or: faz um ou lógico bit a bit dos elementos do topo, empilhando o resultado.
- pop: desempilha a pilha.
- rem: computa o resto da divisão do elemento abaixo do topo, pelo topo, e empilha o resultado.
- rem.un: igual ao rem, mas a comparação é feita sem sinal.
- shl: deslocamento de bits a esquerda.
- shr: deslocamento de bits a direita.
- starg: retira o elemento do topo da pilha e coloca em um argumento.
- sizeof: carrega o tamanho em bits do topo na pilha.
- sub: subtrai o segundo valor do primeiro e empilha o resultado.
- sub.ovf: igual ao sub, mas com a sinalização de overflow.
- xor: executa a operação XOR(bit a bit) entre dois elementos da pilha colocando seu resultado na mesma.

## 4 Códigos em Assembly CIL ( comentados)

### 4.1 Módulo mínimo que caracteriza um programa

Listing 1: Hello World em CIL

```
1  .assembly extern mscorlib
2  {
3  }
4
5  .assembly moduloMinimo
6  {
7  }
8
9  .module moduloMinimo.exe
10
```



## 4 CÓDIGOS EM ASSEMBLY CIL ( COMENTADOS)

---

```
11 .class private auto ansi beforefieldinit moduloMinimo.Program
    extends [mscorlib]System.Object
12 {
13     .method private hidebysig static void    Main(string[] args)
        cil managed
14     {
15         .entrypoint
16         .maxstack 8
17         ret
18     } // end of method Program::Main
19
20
21     .method public hidebysig specialname rtspecialname
        instance void    .ctor() cil managed
22     {
23
24         .maxstack 8
25         ldarg.0
26         call        instance void [mscorlib]System.Object::.ctor
            ()
27         ret
28     } // end of method Program::.ctor
29
30 }
```

Os nomes que se iniciam com o prefixo “.”, como .assembly, .method, .class, são chamados de Diretrizes CIL. A primeira diretriz no código, “.assembly extern mscorlib”, é usada para informar ao assembler que serão utilizados métodos e objetos de um assembly externo, no caso o “mscorlib”.

As próximas duas diretrizes “.assembly moduloMinimo” e “.module moduloMinimo.exe” informam ao programa que estamos criando um código assembly e nomeiam o módulo como moduloMinimo. Sem essas declarações, o assembler não consegue executar o código assembly.

A próxima linha “.class private auto ansi beforefieldinit moduloMinimo.Program extends [mscorlib]System.Object” declara a classe do programa e informa que ela “herda” da classe Object presente em mscorlib.

Após criar a classe é definido o método principal, cuja declaração informa que ele será gerenciado pelo assembly CIL: “.method private hidebysig static void Main(string[] args) cil managed.”. O atributo “hidebysig” significa que o membro na classe base com o mesmo nome e assinatura está oculto da classe derivada. É dentro deste método principal que o programa começa de fato. A diretriz .entrypoint é quem marca o início do programa.

Após o .entrypoint deve-se informar o número de itens que estão na pilha

## 4 CÓDIGOS EM ASSEMBLY CIL ( COMENTADOS)

---

no momento. Para isso, utiliza-se a diretriz `.maxstack`. Caso esse valor não seja informado, é utilizado um valor default igual a 8. Finalmente as instruções CIL podem ser declaradas. Como o objetivo deste programa é apenas apresentar o módulo mínimo de um código CIL, há apenas a instrução “ret” ou “return” que finaliza o método e o programa.

A diretriz `.ctor` representa o construtor em nível de instância. Para representar o construtor estático utiliza-se “`.cctor`” (construtor de classe). O `.ctor` está sempre qualificado com o atributo `specialname` e `rtspecialname`. O `specialname` é usado para indicar que este token pode ser tratado de forma diferente por diferentes ferramentas, por exemplo, em linguagem `c#`, o construtor não tem o tipo de retorno, mas no CIL ele tem o tipo de retorno `void`. A instrução `ldarg` é usada para carregar o argumento passado no método para a pilha. “`ldarg.0`” aqui significa carregar o primeiro argumento na pilha.

### 4.2 Hello World

O código a seguir refere-se a um Hello World escrito em CIL:

Listing 2: Hello World em CIL

```
1 .assembly extern mscorlib {}
2 .assembly Hello {}
3 .module Hello.exe
4
5 .class Hello.Program
6 extends [mscorlib]System.Object
7 {
8   .method static void Main(string[] args) cil managed
9   {
10    .entrypoint
11    .maxstack 8
12    ldstr "Hello World"
13    call void [mscorlib]System.Console:: WriteLine(string)
14    ret
15  }
16
17   .method public hidebysig specialname rtspecialname
18     instance void .ctor() cil managed
19   {
20     .maxstack 8
21     ldarg.0
22     call instance void [mscorlib]System.Object::.ctor()
23     ret
24  } // end of method Program::.ctor
```

## 4 CÓDIGOS EM ASSEMBLY CIL ( COMENTADOS)

---

25 }

O código acima possui todas as características estruturais apresentadas anteriormente. A diferença é que este código apresenta instruções declaradas após o `.entrypoint`. A instrução `ldstr` carrega a string “Hello World” na pilha e a instrução `call` chama o método `WriteLine` da classe `Console`. O método “pega” seus parâmetros da pilha. Por fim, o “ret” ou “return” finaliza o método e o programa.

### 4.3 Execução baseada em pilha

Antes do início de cada método, a pilha está vazia e durante a execução do método, as instruções CIL adicionam e/ou removem os itens da pilha, cujo resultado final é uma pilha vazia no final da execução desse método.

As instruções que copiam valores da memória para a pilha são chamados de `Load` e as instruções que copiam os valores da pilha para a memória são chamados de `Store`. Todas as instruções iniciadas com `ld` são usadas para carregar o item na pilha e as que começam com `st` são usadas para armazenar o item na memória.

Para entender melhor como funciona a pilha durante a execução de um método, considere o seguinte código escrito em C#:

Listing 3: Código em C#

```
1 static void add()
2 {
3     int value1 = 10;
4     int value2 = 20;
5     int value3 = value1 + value2;
6 }
```

O código correspondente em assembly CIL é mostrado a seguir. Os comentários adicionados a este código tem como finalidade mostrar a operação da pilha durante a execução.

Listing 4: Código em CIL

```
1 .method private static void 'add'() cil managed
2 {
3     maxstack 2
4     .locals init ([0] int32 value1,
5                  [1] int32 value2,
```

## 4 CÓDIGOS EM ASSEMBLY CIL ( COMENTADOS)

---

```
6          [2] int32 value3) // 3 variáveis locais do tipo
           int32 são declaradas
7
8 nop // sem operação ( sem push ou pop na pilha)
9
10 ldc.i4.s 10 // carrega o valor 10 de tipo int32 no topo
    da pilha. Itens na pilha = 1
11 stloc.0 // pega o item do topo da pilha (10) e armazena na
    primeira variável local. Itens na pilha = 0.
12 ldc.i4.s 20 // carrega o valor 20 de tipo int32 no topo
    da pilha. Itens na pilha = 1
13 stloc.1 0 // pega o item do topo da pilha (20) e armazena na
    primeira variável local. Itens na pilha = 0.
14 ldloc.0 // carrega o valor da primeira variável local na
    pilha. Itens na pilha = 1
15 ldloc.1 // carrega o valor da primeira variável local na
    pilha. Itens na pilha = 2
16 add // (pega os primeiros dois valores da pilha e envia o
    resultado da soma para a pilha. Itens na pilha = 2-2+1 = 1
17 stloc.2 // pega o valor do topo da pilha e armazena na
    terceira variável local. Itens na pilha = 0
18 ret // fim do método.
19 }
```

Observações:

- No começo do método do programa apresentado acima, temos a diretriz “.locals init”. Ela informa que serão declaradas variáveis locais a serem utilizadas no programa.
- Note que a declaração do método é feita de forma diferente do programa Hello World apresentado anteriormente: “.method private hidebysig static void 'add'() cil managed”. Isso porque este não é o método principal da classe. A declaração deste método indica que ele é do tipo “private” e “static”, não retorna valores (void) e que o seu nome é “add”. Como já foi dito, “cil managed” indica que o método será gerenciado pelo assembly CIL. Essa declaração é necessária para qualquer método escrito em código CIL.
- A declaração .entrypoint não está presente neste método pois essa diretriz faz parte apenas do método principal da classe.

## 4 CÓDIGOS EM ASSEMBLY CIL ( COMENTADOS)

---

```
.assembly extern mscorlib
{
    .publickeytoken = (B7 7A 5C 56 19 34 E0 89 )
    .ver 4:0:0:0
}

.assembly HelloWorld
{
    ----lines omitted
}

.module HelloWorld.exe

.class private auto ansi beforefieldinit HelloWorld.Program
    extends [mscorlib]System.Object
{
    .method private hidebysig static void Main(string[] args) cil managed
    {
        .entrypoint
        // Code size      13 (0xd)
        .maxstack 8
        IL_0000: nop
        IL_0001: ldstr      "Hello World"
        IL_0006: call      void [mscorlib]System.Console::WriteLine(string)
        IL_000b: nop
        IL_000c: ret
    } // end of method Program::Main

    .method public hidebysig specialname rtspecialname
        instance void .ctor() cil managed
    {
        .maxstack 8
        IL_0000: ldarg.0
        IL_0001: call      instance void [mscorlib]System.Object::.ctor()
        IL_0006: ret
    } // end of method Program::.ctor
} // end of class HelloWorld.Program
```

Figura 2: Código Hello World em CIL

Os nomes que se iniciam com o prefixo ".", como .assembly, .method, .class, são chamados de Diretrizes CIL. Os nomes usados juntamente às diretrizes, como extends, implements, public, são chamados de Atributos CIL.

Antes de cada método a pilha está vazia e, durante a execução de cada método, as instruções da CIL adicionam e/ou removem os itens da pilha. No final do método, a pilha volta a ser vazia.

No começo da função, deve-se informar o número de itens que estão na pilha no momento. Para isso, utiliza-se a diretriz .maxstack. Caso esse valor não seja informado, é utilizado um valor default igual a 8.

## 4 CÓDIGOS EM ASSEMBLY CIL ( COMENTADOS)

### 4.4 Condições e Iterações

Para entender como funcionam condições e iterações em CIL será apresentado um código em C# e seu correspondente em CIL. Cada passo executado no código em assembly apresentado será explicado posteriormente.

Listing 5: Código em C#

```
1 static void IterationExample( )
2 {
3     int i = 0;
4     while (i < 5)
5     {
6         i++;
7     }
8 }
```

Listing 6: Código em CIL

```
1 .method private static void IterationExample() cil managed
2 {
3     .maxstack 2
4     .locals init ([0] int32 i,
5                  [1] bool CS$4$0000) // PASSO 1
6     IL_0000: nop //PASSO 2
7     IL_0001: ldc.i4.0 //PASSO 3
8     IL_0002: stloc.0 //PASSO 4
9     IL_0003: br.s IL_000b //PASSO 5
10    IL_0005: nop //PASSO 12 //PASSO 24 e assim
        por diante...
11    IL_0006: ldloc.0 //PASSO 13
12    IL_0007: ldc.i4.1 //PASSO 14
13    IL_0008: add //PASSO 15
14    IL_0009: stloc.0 //PASSO 16
15    IL_000a: nop //PASSO 17
16    IL_000b: ldloc.0 //PASSO 6 //PASSO 18
17    IL_000c: ldc.i4.5 //PASSO 7 //PASSO 19
18    IL_000d: clt //PASSO 8 //PASSO 20
19    IL_000f: stloc.1 //PASSO 9 //PASSO 21
20    IL_0010: ldloc.1 //PASSO 10 //PASSO 22
21    IL_0011: brtrue.s IL_0005 //PASSO 11 //PASSO 23
22    IL_0013: ret
23 }
```

PASSO 1: declara duas variáveis locais nos índices 0 e 1.

## 4 CÓDIGOS EM ASSEMBLY CIL ( COMENTADOS)

---

PASSO 2: sem operação na pilha.

PASSO 3: carrega o valor 0 no topo da pilha (itens na pilha = 1).

PASSO 4: pega o valor do topo da pilha e armazena na variável local 0 (itens na pilha = 0, valor = 0)

PASSO 5: ir para "IL000b".

PASSO 6: carrega a variável 0 na pilha (itens na pilha = 1).

PASSO 7: carrega o valor 5 na pilha (itens na pilha = 2).

PASSO 8: faz a comparação "menor que" com dois itens após removê-los da pilha. Carrega o valor 1 na pilha já que clt retorna 1 (verdadeiro) para a comparação (0j1) (itens na pilha = 1).

PASSO 9: pega o valor da pilha e armazena na variável local 1 (itens na pilha = 0).

PASSO 10: carrega o valor 1 na pilha (itens na pilha = 1, valor = 1).

PASSO 11: brtrue checa se o valor é maior que 0 e depois pula para IL0005. Neste caso, como o valor é 1 a execução vai para IL0005. Se o valor é 0, a execução para a linha que contém "ret" e sai do método.

PASSO 12: sem operação na pilha.

PASSO 13: carrega o valor da variável local 0 na pilha.

PASSO 14: carrega o valor 1 na pilha (itens na pilha = 2, valor = 1)

PASSO 15: pega dois itens na pilha e retorna sua soma para a pilha (itens na pilha = 1, valor = 1).

PASSO 16: armazena a soma na variável local 0 (itens na pilha = 0)

A execução continua até que a condição no PASSO 11 retorne 0.

Observação: nomes como IL0001 e IL0002 que aparecem no código são chamados rótulos CIL. Eles são opcionais, ou seja, podem ser incluídos ou não no código ou podem ser substituídos por qualquer texto. No código acima eles são uteis para direcionar a execução para outra linha do código.

### 4.5 Fatorial

Aqui será apresentado e explicado um código em assembly CIL um pouco mais complexo. Seja o código em C# que calcula o fatorial de um número:

Listing 7: Calcula o fatorial de um número(C#)

```
1 using System;
2 class calculaFatorial {
3     public static int fatorial (int n){
4         if ( n <= 0)
5             return 1;
```

## 4 CÓDIGOS EM ASSEMBLY CIL ( COMENTADOS)

---

```
6 else
7 return n * fatorial(n-1);
8
9 }
10
11 public static void Main(){
12 int numero, fat;
13
14 Console.Write("Digite um número: ");
15 numero = Convert.ToInt32(Console.ReadLine());
16 fat = fatorial(numero);
17 Console.Write("O fatorial de ");
18 Console.Write(numero);
19 Console.Write(" é ");
20 Console.WriteLine(fat);
21 }
22 }
```

O código correspondente em CIL é:

Listing 8: Calcula o fatorial de um número(CIL)

```
1 .assembly extern mscorlib
2 {
3
4 }
5 .assembly 'micro10'
6 {
7 }
8 .module calculaFatorial.exe
9
10
11 .class private auto ansi beforefieldinit micro10 extends [
    mscorlib]System.Object
12 {
13
14     // Metodo 1
15     .method public hidebysig specialname rtspecialname
        instance default void '.ctor' () cil managed
16     {
17         .maxstack 8
18     IL_0000: ldarg.0
19     IL_0001: call instance void object::.ctor'()
20     IL_0006: ret
21     } // end of method calculaFatorial::.ctor
22 }
```



#### 4 CÓDIGOS EM ASSEMBLY CIL ( COMENTADOS)

---

```
23 // Método 2
24 .method public static hidebysig default int32 fatorial (
    int32 n) cil managed
25 {
26 .maxstack 8
27 IL_0000: ldarg.0 // carrega argumento na pilha
28 IL_0001: ldc.i4.0 // carrega inteiro na pilha
29 IL_0002: bgt IL_0009 // pula para IL0009 se o segundo
    elemento da pilha é maior que o topo
30
31 IL_0007: ldc.i4.1 // carrega inteiro na pilha
32 IL_0008: ret // fim do método
33 IL_0009: ldarg.0 // carrega argumento na pilha
34 IL_000a: ldarg.0 // carrega argumento na pilha
35 IL_000b: ldc.i4.1 // carrega inteiro na pilha
36 IL_000c: sub // subtrai o segundo valor do primeiro e
    empilha o resultado. ( n-1)
37 IL_000d: call int32 class calculaFatorial::fatorial(int32)
    // invoca fatorial novamente (recursão)
38 IL_0012: mul // multiplica os dois valores de cima da
    pilha e empilha o resultado (n*fatorial(n-1))
39 IL_0013: ret // sai do metodo
40 } // end of method calculaFatorial::fatorial
41
42 // Metodo 3
43 .method public static hidebysig default void Main () cil
    managed
44 {
45 .entrypoint
46 .maxstack 1
47 .locals init (
48     int32 V_0,
49     int32 V_1) // declaração de variaveis
50 IL_0000: ldstr "Digite um numero:" // carrega string na
    pilha
51 IL_0005: call void class [mscorlib]System.Console::Write(
    string) // chama metodo para escrever a string
52 IL_000a: call string class [mscorlib]System.Console::
    ReadLine() // chama metodo para ler string
53 IL_000f: call int32 class [mscorlib]System.Convert::
    ToInt32(string) // converte a string digitada para
    inteiro
54 IL_0014: stloc.0 // pega o valor do topo da pilha e
    armazena na primeira variável local
55 IL_0015: ldloc.0 // carrega o valor da primeira variavel
```

```
        local na pilha
56  IL_0016:  call int32 class micro10::fatorial(int32) //
        invoca metodo fatorial
57  IL_001b:  stloc.1 // armazena o resultado na segunda
        variavel local
58  IL_001c:  ldstr "0 fatorial de " // carrega string na pilha
59  IL_0021:  call void class [mscorlib]System.Console::Write(
        string) // chama metodo para escrever a string
60  IL_0026:  ldloc.0 // carrega o valor da primeira variavel
        local na pilha
61  IL_0027:  call void class [mscorlib]System.Console::Write(
        int32) // chama metodo para escrever o inteiro (numero
        digitado pelo usuario)
62  IL_002c:  ldstr "é" // carrega string na pilha
63  IL_0031:  call void class [mscorlib]System.Console::Write(
        string) // chama metodo para escrever a string
64  IL_0036:  ldloc.1 // carrega o valor da segunda variavel
        local na pilha
65  IL_0037:  call void class [mscorlib]System.Console::
        WriteLine(int32) // chama metodo para escrever inteiro (
        resultado)
66  IL_003c:  ret
67  } // end of method micro10::Main
68
69  } // end of class calculaFatorial
```

O código CIL acima para calcular o fatorial de um número possui 3 métodos: método ".ctor", o método "fatorial" (calcula o fatorial) e o método Main (recebe o número do teclado, invoca o método "fatorial" e mostra o resultado). A explicação das instruções utilizadas está em forma de comentários no código.

## 5 JavaScript

JavaScript é uma linguagem de programação interpretada baseada em scripts. Os scripts desenvolvidos em JavaScript são executados no interior de programas e são amplamente integrados em páginas web. Tais scripts comumente são incluídos em páginas HTML e interagem com o Modelo de Objeto de Documentos (DOM) da página.

Os códigos em JavaScript apresentados neste relatório irão incluir apenas os scripts, sem a sua integração com outras linguagens.

# 6 Preparação do Ambiente

Esta seção tem como objetivo apresentar as ferramentas necessárias para este projeto e para a compilação segundo o modelo de compilação CLR dos programas propostos e obtenção dos respectivos códigos em linguagem Assembly CIL.

O sistema operacional utilizado é o Ubuntu 14.04.

## 6.1 OCaml

Ocaml é a linguagem de programação escolhida para a implementação do compilador. Para instalar a Ocaml no Ubuntu utiliza-se o seguinte comando:

```
> sudo apt-get install ocaml
```

## 6.2 Mono

O Mono é uma plataforma desenvolvida para criar ferramentas compatíveis com a plataforma .Net. Ele implementa um compilador CLR, ou seja, contém um motor de compilação just-in-time, assim como a plataforma .Net.

A .Net é uma plataforma exclusiva da Microsoft e portanto não será possível utilizá-la neste projeto devido à utilização do Ubuntu. Por isso, o Mono será utilizado.

## 6.3 Instalação

Para instalar o Mono no Ubuntu basta inserir o seguinte comando no terminal:

```
> sudo apt-get install mono-complete
```

## 6.4 Compilação

Para compilar na plataforma Mono um dado programa “olamundo.cs” escrito em linguagem C#, utiliza-se o seguinte comando:

```
> mcs olamundo.cs
```

## 7 CÓDIGOS

---

### 6.4.1 Execução

Para executar o programa:

```
> mono olamundo.exe
```

### 6.4.2 Geração do Assembly

Para obter o Assembly CIL associado ao executável deste programa, utiliza-se o comando:

```
> monodis olamundo.exe
```

### 6.4.3 Geração do Executável

Para obter o executável associado ao Assembly deste programa, utiliza-se o comando:

```
> ilasm olamundo.il
```

## 7 Códigos

Esta seção tem como finalidade apresentar os pseudo códigos propostos e convertê-los para linguagem Javascript e linguagem Assembly CIL. Para a compilação no Mono, os códigos em Javascript serão convertidos para C#.

### 7.1 Nano 01

Listing 9: Módulo mínimo que caracteriza um programa (JavaScript)

Listing 10: Módulo mínimo que caracteriza um programa (C#)

```
1 class nano1 {  
2     public static void Main(){  
3     }  
4 }
```

## 7 CÓDIGOS

Listing 11: Módulo mínimo que caracteriza um programa (CIL)

```
1 .assembly extern mscorlib
2 {
3   .ver 4:0:0:0
4   .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
5 }
6 .assembly 'nano01'
7 {
8   .custom instance void class [mscorlib]System.Runtime.
      CompilerServices.RuntimeCompatibilityAttribute::.ctor
      '() = (
9     01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78    // ....
      T..WrapNonEx
10    63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01         ) //
      ceptionThrows.
11
12   .hash algorithm 0x00008004
13   .ver 0:0:0:0
14 }
15 .module nano01.exe // GUID = {A5E14623-A7C9-4386-A2AB-
      A4439E4638AE}
16
17
18 .class private auto ansi beforefieldinit nano01
19   extends [mscorlib]System.Object
20   {
21
22     // method line 1
23     .method public hidebysig specialname rtspecialname
24       instance default void '.ctor' () cil managed
25     {
26       // Method begins at RVA 0x2050
27       // Code size 7 (0x7)
28       .maxstack 8
29       IL_0000: ldarg.0
30       IL_0001: call instance void object::.ctor'()
31       IL_0006: ret
32     } // end of method nano01::.ctor
33
34     // method line 2
35     .method public static hidebysig
36       default void Main () cil managed
37     {
38       // Method begins at RVA 0x2058
39       .entrypoint
```

## 7 CÓDIGOS

---

```
40 // Code size 1 (0x1)
41 .maxstack 8
42 IL_0000: ret
43 } // end of method nano01::Main
44
45 } // end of class nano01
```

### 7.2 Nano 02

Listing 12: Declaração de uma variável (JavaScript)

```
1 var n;
```

Listing 13: Declaração de uma variável (C#)

```
1 class nano02 {
2 public static void Main(){
3 int n;
4 }
5 }
```

Listing 14: Declaração de uma variável (CIL)

```
1 .assembly extern mscorlib
2 {
3   .ver 4:0:0:0
4   .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
5 }
6 .assembly 'nano02 '
7 {
8   .custom instance void class [mscorlib]System.Runtime.
      CompilerServices.RuntimeCompatibilityAttribute::.ctor
      '() = (
9     01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78 // ....
      T..WrapNonEx
10    63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01      ) //
      ceptionThrows.
11
12   .hash algorithm 0x00008004
13   .ver 0:0:0:0
14 }
15 .module nano02.exe // GUID = {9801BF40-553F-412A-8756-1
      EDCE3D5872B}
16
17
```

## 7 CÓDIGOS

---

```
18 .class private auto ansi beforefieldinit nano02
19     extends [mscorlib]System.Object
20 {
21
22     // method line 1
23     .method public hidebysig specialname rtspecialname
24         instance default void '.ctor' () cil managed
25     {
26         // Method begins at RVA 0x2050
27     // Code size 7 (0x7)
28     .maxstack 8
29     IL_0000: ldarg.0
30     IL_0001: call instance void object::.ctor'()
31     IL_0006: ret
32     } // end of method nano02::.ctor
33
34     // method line 2
35     .method public static hidebysig
36         default void Main () cil managed
37     {
38         // Method begins at RVA 0x2058
39     .entrypoint
40     // Code size 1 (0x1)
41     .maxstack 0
42     .locals init (
43         int32 V_0)
44     IL_0000: ret
45     } // end of method nano02::Main
46
47 } // end of class nano02
```

### 7.3 Nano 03

Listing 15: Atribuição de um inteiro a uma variável (JavaScript)

```
1 var n;
2 n=1;
```

Listing 16: Atribuição de um inteiro a uma variável (C#

```
1 class nano03 {
2     public static void Main(){
3         int n;
4         n = 1;
5     }
```

## 7 CÓDIGOS

---

6 }

Listing 17: Atribuição de um inteiro a uma variável (CIL)

```
1 .assembly extern mscorlib
2 {
3   .ver 4:0:0:0
4   .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
5 }
6 .assembly 'nano03'
7 {
8   .custom instance void class [mscorlib]System.Runtime.
      CompilerServices.RuntimeCompatibilityAttribute::.ctor
      '() = (
9     01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78    // ....
      T..WrapNonEx
10    63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01          ) //
      ceptionThrows.
11
12   .hash algorithm 0x00008004
13   .ver 0:0:0:0
14 }
15 .module nano03.exe // GUID = {7D1CB53A-C8D0-4E7C-8710-6
      F8E3B9028CF}
16
17
18 .class private auto ansi beforefieldinit nano03
19   extends [mscorlib]System.Object
20 {
21
22   // method line 1
23   .method public hidebysig specialname rtspecialname
24     instance default void '.ctor' () cil managed
25   {
26     // Method begins at RVA 0x2050
27     // Code size 7 (0x7)
28     .maxstack 8
29     IL_0000: ldarg.0
30     IL_0001: call instance void object::.ctor'()
31     IL_0006: ret
32   } // end of method nano03::.ctor
33
34   // method line 2
35   .method public static hidebysig
36     default void Main () cil managed
```



## 7 CÓDIGOS

---

```
37     {
38         // Method begins at RVA 0x2058
39     .entrypoint
40     // Code size 3 (0x3)
41     .maxstack 1
42     .locals init (
43         int32 V_0)
44     IL_0000: ldc.i4.1
45     IL_0001: stloc.0
46     IL_0002: ret
47     } // end of method nano03::Main
48
49 } // end of class nano03
```

### 7.4 Nano 04

Listing 18: Atribuição de uma soma de inteiros a uma variável (JavaScript)

```
1 var n;
2 n=1 +2;
```

Listing 19: Atribuição de uma soma de inteiros a uma variável (C#)

```
1 class nano04 {
2     public static void Main(){
3         int n;
4         n = 1 + 2;
5     }
6 }
```

Listing 20: Atribuição de uma soma de inteiros a uma variável (CIL)

```
1 .assembly extern mscorlib
2 {
3     .ver 4:0:0:0
4     .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
5 }
6 .assembly 'nano04'
7 {
8     .custom instance void class [mscorlib]System.Runtime.
        CompilerServices.RuntimeCompatibilityAttribute::.ctor
        '() = (
9         01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78    // ....
        T..WrapNonEx
```

## 7 CÓDIGOS

---

```
10      63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01      ) //
      ceptionThrows.
11
12      .hash algorithm 0x00008004
13      .ver 0:0:0:0
14  }
15  .module nano04.exe // GUID = {F2B9D2CB-2730-4B2A-9ED4-432
      ADD5A882E}
16
17
18  .class private auto ansi beforefieldinit nano04
19      extends [mscorlib]System.Object
20  {
21
22      // method line 1
23      .method public hidebysig specialname rtspecialname
24          instance default void '.ctor' () cil managed
25      {
26          // Method begins at RVA 0x2050
27      // Code size 7 (0x7)
28      .maxstack 8
29      IL_0000: ldarg.0
30      IL_0001: call instance void object::.ctor'()
31      IL_0006: ret
32      } // end of method nano04::.ctor
33
34      // method line 2
35      .method public static hidebysig
36          default void Main () cil managed
37      {
38          // Method begins at RVA 0x2058
39      .entrypoint
40      // Code size 3 (0x3)
41      .maxstack 1
42      .locals init (
43          int32 V_0)
44      IL_0000: ldc.i4.3
45      IL_0001: stloc.0
46      IL_0002: ret
47      } // end of method nano04::Main
48
49  } // end of class nano04
```

### 7.5 Nano 05

## 7 CÓDIGOS

---

Listing 21: Inclusão do comando de impressão (JavaScript)

```
1 var n;  
2 n = 2;  
3 document.write(n);
```

Listing 22: Inclusão do comando de impressão (C#)

```
1 using System;  
2 class nano05 {  
3     public static void Main(){  
4         int n;  
5         n = 2;  
6         Console.Write(n);  
7     }  
8 }
```

Listing 23: Inclusão do comando de impressão (CIL)

```
1 .assembly 'nano05 '  
2 {  
3     .custom instance void class [mscorlib]System.Runtime.  
        CompilerServices.RuntimeCompatibilityAttribute::.ctor  
        '() = (  
4         01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78    // ....  
        T..WrapNonEx  
5         63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01        ) //  
        ceptionThrows.  
6  
7     .hash algorithm 0x00008004  
8     .ver 0:0:0:0  
9 }  
10 .module nano05.exe // GUID = {A4654B95-D993-490C-BB82-2705  
    BA3ACD45}  
11  
12  
13 .class private auto ansi beforefieldinit nano05  
14     extends [mscorlib]System.Object  
15 {  
16  
17     // method line 1  
18     .method public hidebysig specialname rtspecialname  
19         instance default void '.ctor' () cil managed  
20     {  
21         // Method begins at RVA 0x2050  
22         // Code size 7 (0x7)
```

## 7 CÓDIGOS

---

```
23  .maxstack 8
24  IL_0000:  ldarg.0
25  IL_0001:  call instance void object::.ctor'()
26  IL_0006:  ret
27  } // end of method nano05::.ctor
28
29  // method line 2
30  .method public static hidebysig
31      default void Main () cil managed
32  {
33      // Method begins at RVA 0x2058
34  .entrypoint
35  // Code size 9 (0x9)
36  .maxstack 1
37  .locals init (
38      int32 V_0)
39  IL_0000:  ldc.i4.2
40  IL_0001:  stloc.0
41  IL_0002:  ldloc.0
42  IL_0003:  call void class [mscorlib]System.Console::Write(
43      int32)
44  IL_0008:  ret
45  } // end of method nano05::Main
46  } // end of class nano05
```

### 7.6 Nano 06

Listing 24: Atribuição de uma subtração de inteiros a uma variável (JavaScript)

```
1 var n;
2 n=1-2;
3 document.write(n);
```

Listing 25: Atribuição de uma subtração de inteiros a uma variável (C#)

```
1 using System;
2 class nano05 {
3     public static void Main(){
4         int n;
5         n = 1 - 2;
6         Console.Write(n);
7     }
8 }
```

## 7 CÓDIGOS

Listing 26: Atribuição de uma subtração de inteiros a uma variável (CIL)

```
1 .assembly extern mscorlib
2 {
3     .ver 4:0:0:0
4     .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
5 }
6 .assembly 'nano06'
7 {
8     .custom instance void class [mscorlib]System.Runtime.
        CompilerServices.RuntimeCompatibilityAttribute::.ctor
        '() = (
9         01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78 // ....
        T..WrapNonEx
10        63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01      ) //
        ceptionThrows.
11
12    .hash algorithm 0x00008004
13    .ver 0:0:0:0
14 }
15 .module nano06.exe // GUID = {5201859C-0488-442D-B835-
    AB64377EC240}
16
17
18 .class private auto ansi beforefieldinit nano06
19     extends [mscorlib]System.Object
20 {
21
22     // method line 1
23     .method public hidebysig specialname rtspecialname
24         instance default void '.ctor' () cil managed
25     {
26         // Method begins at RVA 0x2050
27         // Code size 7 (0x7)
28         .maxstack 8
29         IL_0000: ldarg.0
30         IL_0001: call instance void object::.ctor'()
31         IL_0006: ret
32     } // end of method nano06::.ctor
33
34     // method line 2
35     .method public static hidebysig
36         default void Main () cil managed
37     {
38         // Method begins at RVA 0x2058
39     .entrypoint
```

## 7 CÓDIGOS

---

```
40 // Code size 9 (0x9)
41 .maxstack 1
42 .locals init (
43     int32 V_0)
44 IL_0000: ldc.i4.m1
45 IL_0001: stloc.0
46 IL_0002: ldloc.0
47 IL_0003: call void class [mscorlib]System.Console::Write(
48     int32)
49 IL_0008: ret
50 } // end of method nano06::Main
51 } // end of class nano06
```

### 7.7 Nano 07

Listing 27: Inclusão do comando condicional (JavaScript)

```
1 var n;
2 n=1;
3 if(n==1)
4 document.write(n);
```

Listing 28: Inclusão de comando condicional (C#)

```
1 using System;
2 class nano07 {
3     public static void Main(){
4         int n;
5         n = 1;
6         if ( n == 1)
7             Console.Write(n);
8     }
9 }
10 }
```

Listing 29: Inclusão de comando condicional (CIL)

```
1 .assembly 'nano07 '
2 {
3     .custom instance void class [mscorlib]System.Runtime.
4         CompilerServices.RuntimeCompatibilityAttribute::.ctor
5         '() = (
6             01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78 // ....
7             T..WrapNonEx
```

## 7 CÓDIGOS

---

```
5      63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01      ) //
      ceptionThrows.
6
7      .hash algorithm 0x00008004
8      .ver 0:0:0:0
9  }
10 .module nano07.exe // GUID = {CCD3B1BD-FA30-4D9A-B653-
      E815EAE203E2}
11
12
13 .class private auto ansi beforefieldinit nano07
14     extends [mscorlib]System.Object
15 {
16
17     // method line 1
18     .method public hidebysig specialname rtspecialname
19         instance default void '.ctor' () cil managed
20     {
21         // Method begins at RVA 0x2050
22         // Code size 7 (0x7)
23         .maxstack 8
24         IL_0000: ldarg.0
25         IL_0001: call instance void object::.ctor'()
26         IL_0006: ret
27     } // end of method nano07::.ctor
28
29     // method line 2
30     .method public static hidebysig
31         default void Main () cil managed
32     {
33         // Method begins at RVA 0x2058
34         .entrypoint
35         // Code size 16 (0x10)
36         .maxstack 2
37         .locals init (
38             int32 V_0)
39         IL_0000: ldc.i4.1
40         IL_0001: stloc.0
41         IL_0002: ldloc.0
42         IL_0003: ldc.i4.1
43         IL_0004: bne.un IL_000f
44
45         IL_0009: ldloc.0
46         IL_000a: call void class [mscorlib]System.Console::Write(
            int32)
```

## 7 CÓDIGOS

---

```
47 IL_000f: ret
48     } // end of method nano07::Main
49
50 } // end of class nano07
```

### 7.8 Nano 08

Listing 30: Inclusão do comando condicional com parte senão(JavaScript)

```
1 var n;
2 n=1;
3 if(n=1){
4 document.write(n);
5 }
6 else{
7 document.write(0);
8 }
```

Listing 31: Inclusão do comando condicional com parte senão(C#)

```
1 using System;
2 class nano08 {
3 public static void Main(){
4 int n;
5 n = 1;
6 if ( n == 1)
7 Console.Write(n);
8 else
9 Console.Write(0);
10
11 }
12 }
```

Listing 32: Inclusão do comando condicional com parte senão(CIL)

```
1 .assembly extern mscorlib
2 {
3     .ver 4:0:0:0
4     .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
5 }
6 .assembly 'nano08'
7 {
8     .custom instance void class [mscorlib]System.Runtime.
        CompilerServices.RuntimeCompatibilityAttribute::.ctor
        '() = (
```



## 7 CÓDIGOS

---

```
9      01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78    // ....
      T..WrapNonEx
10     63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01        ) //
      ceptionThrows.
11
12     .hash algorithm 0x00008004
13     .ver 0:0:0:0
14 }
15 .module nano08.exe // GUID = {00005516-A6A3-43D8-842A-1
      C4C9041633F}
16
17
18     .class private auto ansi beforefieldinit nano08
19     extends [mscorlib]System.Object
20 {
21
22     // method line 1
23     .method public hidebysig specialname rtspecialname
24         instance default void '.ctor' () cil managed
25     {
26         // Method begins at RVA 0x2050
27     // Code size 7 (0x7)
28     .maxstack 8
29     IL_0000: ldarg.0
30     IL_0001: call instance void object::.ctor'()
31     IL_0006: ret
32     } // end of method nano08::.ctor
33
34     // method line 2
35     .method public static hidebysig
36         default void Main () cil managed
37     {
38         // Method begins at RVA 0x2058
39     .entrypoint
40     // Code size 27 (0x1b)
41     .maxstack 2
42     .locals init (
43         int32 V_0)
44     IL_0000: ldc.i4.1
45     IL_0001: stloc.0
46     IL_0002: ldloc.0
47     IL_0003: ldc.i4.1
48     IL_0004: bne.un IL_0014
49
50     IL_0009: ldloc.0
```

## 7 CÓDIGOS

---

```
51 IL_000a: call void class [mscorlib]System.Console::Write(  
    int32)  
52 IL_000f: br IL_001a  
53  
54 IL_0014: ldc.i4.0  
55 IL_0015: call void class [mscorlib]System.Console::Write(  
    int32)  
56 IL_001a: ret  
57 } // end of method nano08::Main  
58  
59 } // end of class nano08
```

### 7.9 Nano 09

Listing 33: Atribuição de duas operações aritméticas sobre inteiros a uma variável(JavaScript)

```
1 var n;  
2 n=1+1/2;  
3 if(n=1){  
4 document.write(n);  
5 }  
6 else{  
7 document.write(0);  
8 }
```

Listing 34: Atribuição de duas operações aritméticas sobre inteiros a uma variável(C#)

```
1 using System;  
2 class nano09{  
3 public static void Main(){  
4 int n;  
5 n = 1 + 1 / 2;  
6 if ( n == 1)  
7 Console.Write(n);  
8 else  
9 Console.Write(0);  
10  
11 }  
12 }
```

Listing 35: Atribuição de duas operações aritméticas sobre inteiros a uma variável(CIL)

## 7 CÓDIGOS

---

```
1 .assembly extern mscorlib
2 {
3   .ver 4:0:0:0
4   .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
5 }
6 .assembly 'nano09'
7 {
8   .custom instance void class [mscorlib]System.Runtime.
      CompilerServices.RuntimeCompatibilityAttribute::.ctor
      '() = (
9     01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78    // ....
      T..WrapNonEx
10    63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01        ) //
      ceptionThrows.
11
12   .hash algorithm 0x00008004
13   .ver 0:0:0:0
14 }
15 .module nano09.exe // GUID = {2A0BCBD4-349E-49BB-9AA6-
      FA7742A27A80}
16
17
18 .class private auto ansi beforefieldinit nano09
19   extends [mscorlib]System.Object
20 {
21
22   // method line 1
23   .method public hidebysig specialname rtspecialname
24     instance default void '.ctor' () cil managed
25   {
26     // Method begins at RVA 0x2050
27   // Code size 7 (0x7)
28   .maxstack 8
29   IL_0000: ldarg.0
30   IL_0001: call instance void object::.ctor'()
31   IL_0006: ret
32   } // end of method nano09::.ctor
33
34   // method line 2
35   .method public static hidebysig
36     default void Main () cil managed
37   {
38     // Method begins at RVA 0x2058
39   .entrypoint
40   // Code size 27 (0x1b)
```

## 7 CÓDIGOS

---

```
41  .maxstack 2
42  .locals init (
43      int32 V_0)
44  IL_0000: ldc.i4.1
45  IL_0001: stloc.0
46  IL_0002: ldloc.0
47  IL_0003: ldc.i4.1
48  IL_0004: bne.un IL_0014
49
50  IL_0009: ldloc.0
51  IL_000a: call void class [mscorlib]System.Console::Write(
52      int32)
53  IL_000f: br IL_001a
54
55  IL_0014: ldc.i4.0
56  IL_0015: call void class [mscorlib]System.Console::Write(
57      int32)
58  IL_001a: ret
59  } // end of method nano09::Main
60
61  } // end of class nano09
```

### 7.10 Nano 10

Listing 36: Atribuição de variáveis inteiras(JavaScript)

```
1 var n,m;
2 n=1;
3 m=2;
4 if(n==m){
5     document.write(n);
6 }
7 else{
8     document.write(0);
9 }
```

Listing 37: Atribuição de variáveis inteiras(C#)

```
1 using System;
2 class nano10{
3     public static void Main(){
4         int n, m;
5         n = 1;
6         m = 2;
7     }
```

## 7 CÓDIGOS

---

```
8 if ( n == m)
9 Console.Write(n);
10 else
11 Console.Write(0);
12
13 }
14 }
```

Listing 38: Atribuição de variáveis inteiras(CIL)

```
1 .assembly 'nano10 '
2 {
3   .custom instance void class [mscorlib]System.Runtime.
        CompilerServices.RuntimeCompatibilityAttribute::.ctor
        '() = (
4     01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78    // ....
        T..WrapNonEx
5     63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01        ) //
        ceptionThrows.
6
7   .hash algorithm 0x00008004
8   .ver 0:0:0:0
9 }
10 .module nano10.exe // GUID = {E8BF1B6B-B890-43AC-A3FF-65
        AF216D4F02}
11
12
13 .class private auto ansi beforefieldinit nano10
14     extends [mscorlib]System.Object
15 {
16
17     // method line 1
18     .method public hidebysig specialname rtspecialname
19         instance default void '.ctor' () cil managed
20     {
21         // Method begins at RVA 0x2050
22         // Code size 7 (0x7)
23         .maxstack 8
24         IL_0000: ldarg.0
25         IL_0001: call instance void object::.ctor'()
26         IL_0006: ret
27     } // end of method nano10::.ctor
28
29     // method line 2
30     .method public static hidebysig
```

## 7 CÓDIGOS

---

```
31         default void Main () cil managed
32     {
33         // Method begins at RVA 0x2058
34         .entrypoint
35         // Code size 29 (0x1d)
36         .maxstack 2
37         .locals init (
38             int32 V_0,
39             int32 V_1)
40     IL_0000: ldc.i4.1
41     IL_0001: stloc.0
42     IL_0002: ldc.i4.2
43     IL_0003: stloc.1
44     IL_0004: ldloc.0
45     IL_0005: ldloc.1
46     IL_0006: bne.un IL_0016
47
48     IL_000b: ldloc.0
49     IL_000c: call void class [mscorlib]System.Console::Write(
50         int32)
51     IL_0011: br IL_001c
52
53     IL_0016: ldc.i4.0
54     IL_0017: call void class [mscorlib]System.Console::Write(
55         int32)
56     IL_001c: ret
57     } // end of method nano10::Main
58 } // end of class nano10
```

### 7.11 Nano 11

Listing 39: Introdução do comando de repetição enquanto(JavaScript)

```
1 var n, m, x;
2 n=1;
3 m=2;
4 x=5;
5 while(x > n){
6     n = n + m;
7     document.write(n);
8 }
```

Listing 40: Introdução do comando de repetição enquanto(C#)

## 7 CÓDIGOS

---

```
1 using System;
2 class nano11 {
3     public static void Main(){
4         int n, m, x;
5         n = 1;
6         m = 2;
7         x = 5;
8
9         while ( x > n){
10            n = n + m;
11            Console.Write(n);
12        }
13    }
14 }
```

Listing 41: Introdução do comando de repetição enquanto(CIL)

```
1 .assembly 'nano11'
2 {
3     .custom instance void class [mscorlib]System.Runtime.
        CompilerServices.RuntimeCompatibilityAttribute::.ctor
        '() = (
4         01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78    // ....
        T..WrapNonEx
5         63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01        ) //
        ceptionThrows.
6
7     .hash algorithm 0x00008004
8     .ver 0:0:0:0
9 }
10 .module nano11.exe // GUID = {AF3ECC5A-57D3-401A-9BCE-0
    A8CCFA6FCBB}
11
12
13 .class private auto ansi beforefieldinit nano11
14     extends [mscorlib]System.Object
15 {
16
17     // method line 1
18     .method public hidebysig specialname rtspecialname
19         instance default void '.ctor' () cil managed
20     {
21         // Method begins at RVA 0x2050
22         // Code size 7 (0x7)
23     .maxstack 8
```

## 7 CÓDIGOS

---

```
24 IL_0000: ldarg.0
25 IL_0001: call instance void object::.ctor'()
26 IL_0006: ret
27 } // end of method nano11::.ctor
28
29 // method line 2
30 .method public static hidebysig
31     default void Main () cil managed
32 {
33     // Method begins at RVA 0x2058
34 .entrypoint
35 // Code size 29 (0x1d)
36 .maxstack 2
37 .locals init (
38     int32 V_0,
39     int32 V_1,
40     int32 V_2)
41 IL_0000: ldc.i4.1
42 IL_0001: stloc.0
43 IL_0002: ldc.i4.2
44 IL_0003: stloc.1
45 IL_0004: ldc.i4.5
46 IL_0005: stloc.2
47 IL_0006: br IL_0015
48
49 IL_000b: ldloc.0
50 IL_000c: ldloc.1
51 IL_000d: add
52 IL_000e: stloc.0
53 IL_000f: ldloc.0
54 IL_0010: call void class [mscorlib]System.Console::Write(
55     int32)
56 IL_0015: ldloc.2
57 IL_0016: ldloc.0
58 IL_0017: bgt IL_000b
59
60 IL_001c: ret
61 } // end of method nano11::Main
62 } // end of class nano11
```

### 7.12 Nano 12

Listing 42: Comando condicional aninhando em um comando de repetição(JavaScript)



## 7 CÓDIGOS

---

```
1 var n, m, x;
2 n = 1;
3 m = 2;
4 x = 5;
5 while(x>n){
6   if(n == m){
7     document.write(n);
8   }
9   else{
10    document.write(0);
11  }
12  x = x -1;
13 }
```

Listing 43: Comando condicional aninhando em um comando de repetição(C#)

```
1 using System;
2 class nano12 {
3   public static void Main(){
4     int n, m, x;
5     n = 1;
6     m = 2;
7     x = 5;
8
9     while ( x > n){
10      if ( n == m)
11        Console.Write(n);
12
13      else
14        Console.Write(0);
15      x = x - 1;
16    }
17  }
18 }
```

Listing 44: Comando condicional aninhando em um comando de repetição(CIL)

```
1 .assembly extern mscorlib
2 {
3   .ver 4:0:0:0
4   .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
5 }
6 .assembly 'nano12'
```

## 7 CÓDIGOS

---

```
7 {
8   .custom instance void class [mscorlib]System.Runtime.
      CompilerServices.RuntimeCompatibilityAttribute::.ctor
      '() = (
9     01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78    // ....
      T..WrapNonEx
10    63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01        ) //
      ceptionThrows.
11
12   .hash algorithm 0x00008004
13   .ver 0:0:0:0
14 }
15 .module nano12.exe // GUID = {15C6B382-B3A6-4401-8043-
      CB4CD765B5EF}
16
17
18 .class private auto ansi beforefieldinit nano12
19   extends [mscorlib]System.Object
20 {
21
22   // method line 1
23   .method public hidebysig specialname rtspecialname
24     instance default void '.ctor' () cil managed
25   {
26     // Method begins at RVA 0x2050
27   // Code size 7 (0x7)
28   .maxstack 8
29   IL_0000: ldarg.0
30   IL_0001: call instance void object::.ctor'()
31   IL_0006: ret
32   } // end of method nano12::.ctor
33
34   // method line 2
35   .method public static hidebysig
36     default void Main () cil managed
37   {
38     // Method begins at RVA 0x2058
39   .entrypoint
40   // Code size 47 (0x2f)
41   .maxstack 2
42   .locals init (
43     int32 V_0,
44     int32 V_1,
45     int32 V_2)
46   IL_0000: ldc.i4.1
```

## 7 CÓDIGOS

---

```
47 IL_0001: stloc.0
48 IL_0002: ldc.i4.2
49 IL_0003: stloc.1
50 IL_0004: ldc.i4.5
51 IL_0005: stloc.2
52 IL_0006: br IL_0027
53
54 IL_000b: ldloc.0
55 IL_000c: ldloc.1
56 IL_000d: bne.un IL_001d
57
58 IL_0012: ldloc.0
59 IL_0013: call void class [mscorlib]System.Console::Write(
    int32)
60 IL_0018: br IL_0023
61
62 IL_001d: ldc.i4.0
63 IL_001e: call void class [mscorlib]System.Console::Write(
    int32)
64 IL_0023: ldloc.2
65 IL_0024: ldc.i4.1
66 IL_0025: sub
67 IL_0026: stloc.2
68 IL_0027: ldloc.2
69 IL_0028: ldloc.0
70 IL_0029: bgt IL_000b
71
72 IL_002e: ret
73 } // end of method nano12::Main
74
75 } // end of class nano12
```

### 7.13 Micro 01

Listing 45: Converte graus Celsius para Fahrenheit(JavaScript)

```
1 var cel, far;
2 document.write("Tabela de conversao: Celsuis -> Fahrenheit");
3 cel = prompt("Digite a temperatura em Celsius: ");
4 far = (9*cel+160)/5
5 document.write("A nova temperatura é: " +far);
```

Listing 46: Converte graus Celsius para Fahrenheit(C#)

```
1 using System;
```

## 7 CÓDIGOS

---

```
2 class micro01 {
3 public static void Main(){
4 double cel, far;
5 Console.WriteLine("Tabela de conversão: Celsius -> Fahrenheit
    ");
6 Console.Write("Digite a temperatura em Celsius: ");
7 cel = Convert.ToDouble(Console.ReadLine());
8 far = (9*cel+160)/5;
9 Console.WriteLine("A nova temperatura é " + far + " F");
10 }
11 }
```

Listing 47: Converte graus Celsius para Fahrenheit(CIL)

```
1 .assembly extern mscorlib
2 {
3   .ver 4:0:0:0
4   .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
5 }
6 .assembly 'micro01'
7 {
8   .custom instance void class [mscorlib]System.Runtime.
        CompilerServices.RuntimeCompatibilityAttribute::.ctor
        '() = (
9       01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78    // ....
        T..WrapNonEx
10      63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01          ) //
        ceptionThrows.
11
12   .hash algorithm 0x00008004
13   .ver 0:0:0:0
14 }
15 .module micro01.exe // GUID = {BB705713-99F6-4277-9269-743
        CACC6EC45}
16
17
18 .class private auto ansi beforefieldinit micro01
19     extends [mscorlib]System.Object
20 {
21
22     // method line 1
23     .method public hidebysig specialname rtspecialname
24         instance default void '.ctor' () cil managed
25     {
26         // Method begins at RVA 0x2050
```

## 7 CÓDIGOS

---

```
27 // Code size 7 (0x7)
28 .maxstack 8
29 IL_0000: ldarg.0
30 IL_0001: call instance void object::.ctor'()
31 IL_0006: ret
32 } // end of method micro01::.ctor
33
34 // method line 2
35 .method public static hidebysig
36     default void Main () cil managed
37 {
38     // Method begins at RVA 0x2058
39 .entrypoint
40 // Code size 90 (0x5a)
41 .maxstack 3
42 .locals init (
43     float64 V_0,
44     float64 V_1)
45 IL_0000: ldstr bytearray (
46 54 00 61 00 62 00 65 00 6c 00 61 00 20 00 64 00 // T.a.b.e
47     .l.a. .d.
48 65 00 20 00 63 00 6f 00 6e 00 76 00 65 00 72 00 // e. .c.o
49     .n.v.e.r.
50 73 00 e3 00 6f 00 3a 00 20 00 43 00 65 00 6c 00 // s...o
51     ...C.e.l.
52 73 00 69 00 75 00 73 00 20 00 2d 00 3e 00 20 00 // s.i.u.s
53     . .-.>. .
54 46 00 61 00 68 00 72 00 65 00 6e 00 68 00 65 00 // F.a.h.r
55     .e.n.h.e.
56 69 00 74 00 01 ) // i.t..
57
58 IL_0005: call void class [mscorlib]System.Console::
59     WriteLine(string)
60 IL_000a: ldstr "Digite a temperatura em Celsius: "
61 IL_000f: call void class [mscorlib]System.Console::Write(
62     string)
63 IL_0014: call string class [mscorlib]System.Console::
64     ReadLine()
65 IL_0019: call float64 class [mscorlib]System.Convert::
66     ToDouble(string)
67 IL_001e: stloc.0
68 IL_001f: ldc.r8 9.
69 IL_0028: ldloc.0
70 IL_0029: mul
71 IL_002a: ldc.r8 160.
```

## 7 CÓDIGOS

---

```
63 IL_0033: add
64 IL_0034: ldc.r8 5.
65 IL_003d: div
66 IL_003e: stloc.1
67 IL_003f: ldstr bytearray (
68 41 00 20 00 6e 00 6f 00 76 00 61 00 20 00 74 00 // A. .n.o
    .v.a. .t.
69 65 00 6d 00 70 00 65 00 72 00 61 00 74 00 75 00 // e.m.p.e
    .r.a.t.u.
70 72 00 61 00 20 00 e9 00 20 00 01 ) // r.a.
    ... ..
71
72 IL_0044: ldloc.1
73 IL_0045: box [mscorlib]System.Double
74 IL_004a: ldstr " F"
75 IL_004f: call string string::Concat(object, object, object
    )
76 IL_0054: call void class [mscorlib]System.Console::
    WriteLine(string)
77 IL_0059: ret
78 } // end of method micro01::Main
79
80 } // end of class micro01
```

### 7.14 Micro 02

Listing 48: Lê dois inteiros e decide qual é maior (JavaScript)

```
1 var num1, num2;
2 num1 = prompt("Digite o primeiro numero: ");
3 num2 = prompt("Digite o segundo numero: ");
4 if(num1 > num2){
5 document.write("O primeiro numero " +num1+ "é maior que o
    segundo" +num2);
6 }
7 else{
8 document.write("O segundo numero " +num2+ "é maior que o
    primeiro" +num1);
9 }
```

Listing 49: Ler dois inteiros e decide qual é maior (C#)

```
1 using System;
2 class micro02 {
3 public static void Main(){
```

## 7 CÓDIGOS

---

```
4 int num1, num2;
5 Console.Write("Digite o primeiro número: ");
6 num1 = Convert.ToInt32(Console.ReadLine());
7 Console.Write("Digite o segundo número: ");
8 num2 = Convert.ToInt32(Console.ReadLine());
9 if ( num1 > num2)
10 Console.Write("O primeiro número " + num1 + " é maior que o
    segundo " + num2);
11 else
12
13 Console.Write("O segundo número " + num2 + " é maior que o
    primeiro" + num2);
14 }
15 }
```

Listing 50: Ler dois inteiros e decide qual é maior (CIL)

```
1 .assembly extern mscorlib
2 {
3     .ver 4:0:0:0
4     .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
5 }
6 .assembly 'micro02'
7 {
8     .custom instance void class [mscorlib]System.Runtime.
        CompilerServices.RuntimeCompatibilityAttribute::.ctor
        '() = (
9         01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78    // ....
            T..WrapNonEx
10         63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01        ) //
            ceptionThrows.
11
12     .hash algorithm 0x00008004
13     .ver 0:0:0:0
14 }
15 .module micro02.exe // GUID = {E62D9F80-4E16-4D3D-9E3A
    -5761710E4171}
16
17
18 .class private auto ansi beforefieldinit micro02
19     extends [mscorlib]System.Object
20 {
21
22     // method line 1
23     .method public hidebysig specialname rtspecialname
```

## 7 CÓDIGOS

---

```
24         instance default void '.ctor' () cil managed
25     {
26         // Method begins at RVA 0x2050
27         // Code size 7 (0x7)
28         .maxstack 8
29         IL_0000: ldarg.0
30         IL_0001: call instance void object::.ctor'()
31         IL_0006: ret
32     } // end of method micro02::.ctor
33
34     // method line 2
35     .method public static hidebysig
36         default void Main () cil managed
37     {
38         // Method begins at RVA 0x2058
39         .entrypoint
40         // Code size 155 (0x9b)
41         .maxstack 4
42         .locals init (
43             int32 V_0,
44             int32 V_1)
45         IL_0000: ldstr bytearray (
46         44 00 69 00 67 00 69 00 74 00 65 00 20 00 6f 00 // D.i.g.i
47             .t.e. .o.
48         20 00 70 00 72 00 69 00 6d 00 65 00 69 00 72 00 // .p.r.i
49             .m.e.i.r.
50         6f 00 20 00 6e 00 fa 00 6d 00 65 00 72 00 6f 00 // o. .n
51             ...m.e.r.o.
52         3a 00 20 00 01 ) // :. ..
53
54         IL_0005: call void class [mscorlib]System.Console::Write(
55             string)
56         IL_000a: call string class [mscorlib]System.Console::
57             ReadLine()
58         IL_000f: call int32 class [mscorlib]System.Convert::
59             ToInt32(string)
60         IL_0014: stloc.0
61         IL_0015: ldstr bytearray (
62         44 00 69 00 67 00 69 00 74 00 65 00 20 00 6f 00 // D.i.g.i
63             .t.e. .o.
64         20 00 73 00 65 00 67 00 75 00 6e 00 64 00 6f 00 // .s.e.g
65             .u.n.d.o.
66         20 00 6e 00 fa 00 6d 00 65 00 72 00 6f 00 3a 00 // .n...m
67             .e.r.o...
68         20 00 01 ) // ..
```



## 7 CÓDIGOS

---

```
60
61 IL_001a:  call void class [mscorlib]System.Console::Write(
        string)
62 IL_001f:  call string class [mscorlib]System.Console::
        ReadLine()
63 IL_0024:  call int32 class [mscorlib]System.Convert::
        ToInt32(string)
64 IL_0029:  stloc.1
65 IL_002a:  ldloc.0
66 IL_002b:  ldloc.1
67 IL_002c:  ble IL_0068
68
69 IL_0031:  ldc.i4.4
70 IL_0032:  newarr [mscorlib]System.Object
71 IL_0037:  dup
72 IL_0038:  ldc.i4.0
73 IL_0039:  ldstr bytearray (
74 4f 00 20 00 70 00 72 00 69 00 6d 00 65 00 69 00 // 0. .p.r
        .i.m.e.i.
75 72 00 6f 00 20 00 6e 00 fa 00 6d 00 65 00 72 00 // r.o. .n
        ...m.e.r.
76 6f 00 20 00 01 ) // o. ..
77
78 IL_003e:  stelem.ref
79 IL_003f:  dup
80 IL_0040:  ldc.i4.1
81 IL_0041:  ldloc.0
82 IL_0042:  box [mscorlib]System.Int32
83 IL_0047:  stelem.ref
84 IL_0048:  dup
85 IL_0049:  ldc.i4.2
86 IL_004a:  ldstr bytearray (
87 20 00 e9 00 20 00 6d 00 61 00 69 00 6f 00 72 00 // ... .m
        .a.i.o.r.
88 20 00 71 00 75 00 65 00 20 00 6f 00 20 00 73 00 // .q.u.e
        . .o. .s.
89 65 00 67 00 75 00 6e 00 64 00 6f 00 20 00 01 ) // e.g.u.n
        .d.o. ..
90
91 IL_004f:  stelem.ref
92 IL_0050:  dup
93 IL_0051:  ldc.i4.3
94 IL_0052:  ldloc.1
95 IL_0053:  box [mscorlib]System.Int32
96 IL_0058:  stelem.ref
```

## 7 CÓDIGOS

---

```
97  IL_0059:  call string string::Concat(object[])
98  IL_005e:  call void class [mscorlib]System.Console::Write(
          string)
99  IL_0063:  br IL_009a
100
101  IL_0068:  ldc.i4.4
102  IL_0069:  newarr [mscorlib]System.Object
103  IL_006e:  dup
104  IL_006f:  ldc.i4.0
105  IL_0070:  ldstr bytearray (
106  4f 00 20 00 73 00 65 00 67 00 75 00 6e 00 64 00 // 0. .s.e
          .g.u.n.d.
107  6f 00 20 00 6e 00 fa 00 6d 00 65 00 72 00 6f 00 // o. .n
          ...m.e.r.o.
108  20 00 01 ) // ..
109
110  IL_0075:  stelem.ref
111  IL_0076:  dup
112  IL_0077:  ldc.i4.1
113  IL_0078:  ldloc.1
114  IL_0079:  box [mscorlib]System.Int32
115  IL_007e:  stelem.ref
116  IL_007f:  dup
117  IL_0080:  ldc.i4.2
118  IL_0081:  ldstr bytearray (
119  20 00 e9 00 20 00 6d 00 61 00 69 00 6f 00 72 00 // ... .m
          .a.i.o.r.
120  20 00 71 00 75 00 65 00 20 00 6f 00 20 00 70 00 // .q.u.e
          . .o. .p.
121  72 00 69 00 6d 00 65 00 69 00 72 00 6f 00 01 ) // r.i.m.e
          .i.r.o..
122
123  IL_0086:  stelem.ref
124  IL_0087:  dup
125  IL_0088:  ldc.i4.3
126  IL_0089:  ldloc.1
127  IL_008a:  box [mscorlib]System.Int32
128  IL_008f:  stelem.ref
129  IL_0090:  call string string::Concat(object[])
130  IL_0095:  call void class [mscorlib]System.Console::Write(
          string)
131  IL_009a:  ret
132  } // end of method micro02::Main
133
134  } // end of class micro02
```

## 7 CÓDIGOS

---

### 7.15 Micro 03

Listing 51: Lê um número e verifica se ele está entre 100 e 200(JavaScript)

```
1 var numero;  
2 numero = prompt("Digite um numero: ");  
3 if(numero >= 100){  
4 if(numero <= 200){  
5 document.write("O numero está no intervalo entre 100 e 200");  
6 }  
7 else{  
8 document.write("O numero nao está no intervalo entre 100 e  
9     200");  
10 }  
11 }  
12 document.write("O numero nao está no intervalo entre 100 e  
13     200");  
14 }
```

Listing 52: Lê um número e verifica se ele está entre 100 e 200(C#)

```
1 using System;  
2 class micro03 {  
3     public static void Main(){  
4         int numero;  
5         Console.Write("Digite um número: ");  
6         numero = Convert.ToInt32(Console.ReadLine());  
7         if ( numero >= 100){  
8             if ( numero <= 200)  
9                 Console.WriteLine("O número  esta no intervalo entre 100 e  
10                    200");  
11         }  
12         else  
13             Console.WriteLine("O número não está no intervalo entre 100 e  
14                    200");  
15     }  
16 }
```

Listing 53: Lê um número e verifica se ele está entre 100 e 200(CIL)

```
1 .assembly extern mscorlib
```

## 7 CÓDIGOS

---

```
2 {
3   .ver 4:0:0:0
4   .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
5 }
6 .assembly 'micro03'
7 {
8   .custom instance void class [mscorlib]System.Runtime.
      CompilerServices.RuntimeCompatibilityAttribute::.ctor
      '() = (
9     01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78    // ....
      T..WrapNonEx
10    63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01        ) //
      ceptionThrows.
11
12   .hash algorithm 0x00008004
13   .ver 0:0:0:0
14 }
15 .module micro03.exe // GUID = {E42F4BE8-EE27-49F9-BE32-1
      C00BE87B41B}
16
17
18 .class private auto ansi beforefieldinit micro03
19   extends [mscorlib]System.Object
20 {
21
22   // method line 1
23   .method public hidebysig specialname rtspecialname
24     instance default void '.ctor' () cil managed
25   {
26     // Method begins at RVA 0x2050
27     // Code size 7 (0x7)
28     .maxstack 8
29     IL_0000: ldarg.0
30     IL_0001: call instance void object::.ctor'()
31     IL_0006: ret
32   } // end of method micro03::.ctor
33
34   // method line 2
35   .method public static hidebysig
36     default void Main () cil managed
37   {
38     // Method begins at RVA 0x2058
39     .entrypoint
40     // Code size 81 (0x51)
41     .maxstack 2
```

## 7 CÓDIGOS

---

```
42  .locals init (
43      int32 V_0)
44  IL_0000:  ldstr bytearray (
45      44 00 69 00 67 00 69 00 74 00 65 00 20 00 75 00 // D.i.g.i
         .t.e. .u.
46      6d 00 20 00 6e 00 fa 00 6d 00 65 00 72 00 6f 00 // m. .n
         ...m.e.r.o.
47      3a 00 20 00 01 ) // :. .
48
49  IL_0005:  call void class [mscorlib]System.Console::Write(
         string)
50  IL_000a:  call string class [mscorlib]System.Console::
         ReadLine()
51  IL_000f:  call int32 class [mscorlib]System.Convert::
         ToInt32(string)
52  IL_0014:  stloc.0
53  IL_0015:  ldloc.0
54  IL_0016:  ldc.i4.s 0x64
55  IL_0018:  blt IL_0046
56
57  IL_001d:  ldloc.0
58  IL_001e:  ldc.i4 200
59  IL_0023:  bgt IL_0037
60
61  IL_0028:  ldstr bytearray (
62      4f 00 20 00 6e 00 fa 00 6d 00 65 00 72 00 6f 00 // 0. .n
         ...m.e.r.o.
63      20 00 20 00 65 00 73 00 74 00 61 00 20 00 6e 00 // . .e.s
         .t.a. .n.
64      6f 00 20 00 69 00 6e 00 74 00 65 00 72 00 76 00 // o. .i.n
         .t.e.r.v.
65      61 00 6c 00 6f 00 20 00 65 00 6e 00 74 00 72 00 // a.l.o.
         .e.n.t.r.
66      65 00 20 00 31 00 30 00 30 00 20 00 65 00 20 00 // e.
         .1.0.0. .e. .
67      32 00 30 00 30 00 01 ) // 2.0.0..
68
69  IL_002d:  call void class [mscorlib]System.Console::
         WriteLine(string)
70  IL_0032:  br IL_0041
71
72  IL_0037:  ldstr bytearray (
73      4f 00 20 00 6e 00 fa 00 6d 00 65 00 72 00 6f 00 // 0. .n
         ...m.e.r.o.
74      20 00 6e 00 e3 00 6f 00 20 00 65 00 73 00 74 00 // .n...o
```

## 7 CÓDIGOS

---

```

    . .e.s.t.
75  e1 00 20 00 6e 00 6f 00 20 00 69 00 6e 00 74 00 // .. .n.o
    . .i.n.t.
76  65 00 72 00 76 00 61 00 6c 00 6f 00 20 00 65 00 // e.r.v.a
    .l.o. .e.
77  6e 00 74 00 72 00 65 00 20 00 31 00 30 00 30 00 // n.t.r.e
    . .1.0.0.
78  20 00 65 00 20 00 32 00 30 00 30 00 01 )          // .e.
    .2.0.0..
79
80  IL_003c:  call void class [mscorlib]System.Console::
           WriteLine(string)
81  IL_0041:  br IL_0050
82
83  IL_0046:  ldstr bytearray (
84  4f 00 20 00 6e 00 fa 00 6d 00 65 00 72 00 6f 00 // 0. .n
    ...m.e.r.o.
85  20 00 6e 00 e3 00 6f 00 20 00 65 00 73 00 74 00 // .n...o
    . .e.s.t.
86  e1 00 20 00 6e 00 6f 00 20 00 69 00 6e 00 74 00 // .. .n.o
    . .i.n.t.
87  65 00 72 00 76 00 61 00 6c 00 6f 00 20 00 65 00 // e.r.v.a
    .l.o. .e.
88  6e 00 74 00 72 00 65 00 20 00 31 00 30 00 30 00 // n.t.r.e
    . .1.0.0.
89  20 00 65 00 20 00 32 00 30 00 30 00 01 )          // .e.
    .2.0.0..
90
91  IL_004b:  call void class [mscorlib]System.Console::
           WriteLine(string)
92  IL_0050:  ret
93  } // end of method micro03::Main
94
95  } // end of class micro03
```

### 7.16 Micro 04

Listing 54: Lê números e informa quais estão entre 10 e 150(JavaScript)

```
1 var x, num, intervalo;
2 intervalo = 0;
3 for(x=1;x<=5;x++){
4 num = prompt("Digite um numero");
5 if(num >= 10){
6 if(num <= 150){
```

## 7 CÓDIGOS

---

```
7 intervalo = intervalo +1;
8 }
9 }
10 }
11 document.write("Ao total foram digitados " +intervalo+ "
    numeros no intervalo entre 10 e 50");
```

Listing 55: Lê números e informa quais estão entre 10 e 150(C#)

```
1 using System;
2 class micro04 {
3     public static void Main(){
4         int x, num, intervalo;
5
6         intervalo = 0;
7
8         for ( x = 1; x <= 5; x++){
9             Console.Write("Digite um número: ");
10            num = Convert.ToInt32(Console.ReadLine());
11            if ( num >= 10 )
12
13            if ( num <= 150 )
14                intervalo = intervalo + 1;
15
16        }
17        Console.WriteLine("Ao total, foram digitados " + intervalo +
18            " números no intervalo entre 10 e 150");
19    }
20 }
```

Listing 56: Lê números e informa quais estão entre 10 e 150(CIL)

```
1 .assembly extern mscorlib
2 {
3     .ver 4:0:0:0
4     .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
5 }
6 .assembly 'micro04'
7 {
8     .custom instance void class [mscorlib]System.Runtime.
        CompilerServices.RuntimeCompatibilityAttribute::.ctor
        '() = (
9         01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78 // ....
        T..WrapNonEx
```

## 7 CÓDIGOS

---

```
10      63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01      ) //
      ceptionThrows.
11
12      .hash algorithm 0x00008004
13      .ver 0:0:0:0
14  }
15  .module micro04.exe // GUID = {6FD00F32-6B00-44B9-BB2C-278
      DA96AAABA}
16
17
18  .class private auto ansi beforefieldinit micro04
19      extends [mscorlib]System.Object
20  {
21
22      // method line 1
23      .method public hidebysig specialname rtspecialname
24          instance default void '.ctor' () cil managed
25      {
26          // Method begins at RVA 0x2050
27      // Code size 7 (0x7)
28      .maxstack 8
29      IL_0000: ldarg.0
30      IL_0001: call instance void object::.ctor'()
31      IL_0006: ret
32      } // end of method micro04::.ctor
33
34      // method line 2
35      .method public static hidebysig
36          default void Main () cil managed
37      {
38          // Method begins at RVA 0x2058
39      .entrypoint
40      // Code size 91 (0x5b)
41      .maxstack 3
42      .locals init (
43          int32 V_0,
44          int32 V_1,
45          int32 V_2)
46      IL_0000: ldc.i4.0
47      IL_0001: stloc.2
48      IL_0002: ldc.i4.1
49      IL_0003: stloc.0
50      IL_0004: br IL_0039
51
52      IL_0009: ldstr bytearray (
```



## 7 CÓDIGOS

---

```
53  44 00 69 00 67 00 69 00 74 00 65 00 20 00 75 00 // D.i.g.i
    .t.e. .u.
54  6d 00 20 00 6e 00 fa 00 6d 00 65 00 72 00 6f 00 // m. .n
    ...m.e.r.o.
55  3a 00 20 00 01 ) // :. .
56
57  IL_000e:  call void class [mscorlib]System.Console::Write(
    string)
58  IL_0013:  call string class [mscorlib]System.Console::
    ReadLine()
59  IL_0018:  call int32 class [mscorlib]System.Convert::
    ToInt32(string)
60  IL_001d:  stloc.1
61  IL_001e:  ldloc.1
62  IL_001f:  ldc.i4.s 0x0a
63  IL_0021:  blt IL_0035
64
65  IL_0026:  ldloc.1
66  IL_0027:  ldc.i4 150
67  IL_002c:  bgt IL_0035
68
69  IL_0031:  ldloc.2
70  IL_0032:  ldc.i4.1
71  IL_0033:  add
72  IL_0034:  stloc.2
73  IL_0035:  ldloc.0
74  IL_0036:  ldc.i4.1
75  IL_0037:  add
76  IL_0038:  stloc.0
77  IL_0039:  ldloc.0
78  IL_003a:  ldc.i4.5
79  IL_003b:  ble IL_0009
80
81  IL_0040:  ldstr "Ao total, foram digitados "
82  IL_0045:  ldloc.2
83  IL_0046:  box [mscorlib]System.Int32
84  IL_004b:  ldstr bytearray (
85  20 00 6e 00 fa 00 6d 00 65 00 72 00 6f 00 73 00 // .n...m
    .e.r.o.s.
86  20 00 6e 00 6f 00 20 00 69 00 6e 00 74 00 65 00 // .n.o.
    .i.n.t.e.
87  72 00 76 00 61 00 6c 00 6f 00 20 00 65 00 6e 00 // r.v.a.l
    .o. .e.n.
88  74 00 72 00 65 00 20 00 31 00 30 00 20 00 65 00 // t.r.e.
    .1.0. .e.
```

## 7 CÓDIGOS

---

```
89  20 00 31 00 35 00 30 00 01 )           //
    .1.5.0..
90
91  IL_0050:  call string string::Concat(object, object, object
    )
92  IL_0055:  call void class [mscorlib]System.Console::
    WriteLine(string)
93  IL_005a:  ret
94  } // end of method micro04::Main
95
96  } // end of class micro04
```

### 7.17 Micro 05

Listing 57: Lê strings e caracteres(JavaScript)

```
1 var nome, sexo, x, h, m;
2 h = 0;
3 m = 0;
4 for(x=1;x<=5;x++){
5 nome = prompt("Digite o nome: ");
6 sexo = prompt("H  Homem ou M  Mulher: ");
7 switch(sexo){
8 case 'H':
9 h = h +1;
10 break;
11 case 'M':
12 m = m +1;
13 break;
14 default:
15 document.write("Sexo só pode ser H ou M!");
16 }
17 }
18 document.write("Foram inseridos " +h+ "homens");
19 document.write("Foram inseridos " +m+ "mulheres");
```

Listing 58: Lê strings e caracteres(C#)

```
1 using System;
2 class micro05 {
3 public static void Main(){
4 string nome;
5 char sexo;
6 int x, h, m;
7
```

## 7 CÓDIGOS

---

```
8 h = 0;
9 m = 0;
10
11 for ( x = 1; x <= 5; x++){
12 Console.Write("Digite o nome: ");
13 nome = Console.ReadLine();
14 Console.Write("H - Homem ou M - Mulher: ");
15 sexo = Convert.ToChar(Console.Read());
16 switch (sexo){
17 case 'H':
18     h = h + 1;
19     break;
20 case 'M':
21     m = m + 1;
22     break;
23 default:
24 Console.WriteLine("Sexo só pode ser H ou M!");
25 break;
26
27 }
28 }
29 Console.WriteLine("Foram inseridos " + h + " Homens");
30 Console.WriteLine("Foram inseridos " + m + " Mulheres");
31 }
32 }
```

Listing 59: Lê strings e caracteres(CIL)

```
1 .assembly extern mscorlib
2 {
3     .ver 4:0:0:0
4     .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
5 }
6 .assembly 'micro05'
7 {
8     .custom instance void class [mscorlib]System.Runtime.
        CompilerServices.RuntimeCompatibilityAttribute::.ctor
        '() = (
9         01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78    // ....
            T..WrapNonEx
10         63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01        ) //
            ceptionThrows.
11
12     .hash algorithm 0x00008004
13     .ver 0:0:0:0
```

## 7 CÓDIGOS

---

```
14 }
15 .module micro05.exe // GUID = {72D3380D-485B-4ADB-8D92-492
    D7DB0832A}
16
17
18 .class private auto ansi beforefieldinit micro05
19     extends [mscorlib]System.Object
20 {
21
22     // method line 1
23     .method public hidebysig specialname rtspecialname
24         instance default void '.ctor' () cil managed
25     {
26         // Method begins at RVA 0x2050
27     // Code size 7 (0x7)
28     .maxstack 8
29     IL_0000: ldarg.0
30     IL_0001: call instance void object::.ctor'()
31     IL_0006: ret
32     } // end of method micro05::.ctor
33
34     // method line 2
35     .method public static hidebysig
36         default void Main () cil managed
37     {
38         // Method begins at RVA 0x2058
39     .entrypoint
40     // Code size 170 (0xaa)
41     .maxstack 3
42     .locals init (
43         string V_0,
44         char V_1,
45         int32 V_2,
46         int32 V_3,
47         int32 V_4)
48     IL_0000: ldc.i4.0
49     IL_0001: stloc.3
50     IL_0002: ldc.i4.0
51     IL_0003: stloc.s 4
52     IL_0005: ldc.i4.1
53     IL_0006: stloc.2
54     IL_0007: br IL_006d
55
56     IL_000c: ldstr "Digite o nome: "
57     IL_0011: call void class [mscorlib]System.Console::Write(
```

## 7 CÓDIGOS

---

```
        string)
58  IL_0016:  call string class [mscorlib]System.Console::
        ReadLine()
59  IL_001b:  stloc.0
60  IL_001c:  ldstr "H - Homem ou M - Mulher: "
61  IL_0021:  call void class [mscorlib]System.Console::Write(
        string)
62  IL_0026:  call int32 class [mscorlib]System.Console::Read()
63  IL_002b:  call char class [mscorlib]System.Convert::ToChar(
        int32)
64  IL_0030:  stloc.1
65  IL_0031:  ldloc.1
66  IL_0032:  ldc.i4.s 0x48
67  IL_0034:  beq IL_0046
68
69  IL_0039:  ldloc.1
70  IL_003a:  ldc.i4.s 0x4d
71  IL_003c:  beq IL_004f
72
73  IL_0041:  br IL_005a
74
75  IL_0046:  ldloc.3
76  IL_0047:  ldc.i4.1
77  IL_0048:  add
78  IL_0049:  stloc.3
79  IL_004a:  br IL_0069
80
81  IL_004f:  ldloc.s 4
82  IL_0051:  ldc.i4.1
83  IL_0052:  add
84  IL_0053:  stloc.s 4
85  IL_0055:  br IL_0069
86
87  IL_005a:  ldstr bytearray (
88  53 00 65 00 78 00 6f 00 20 00 73 00 f3 00 20 00 // S.e.x.o
        . .s... .
89  70 00 6f 00 64 00 65 00 20 00 73 00 65 00 72 00 // p.o.d.e
        . .s.e.r.
90  20 00 48 00 20 00 6f 00 75 00 20 00 4d 00 21 00 // .H. .o
        .u. .M.!.
91  01 ) // .
92
93  IL_005f:  call void class [mscorlib]System.Console::
        WriteLine(string)
94  IL_0064:  br IL_0069
```

## 7 CÓDIGOS

---

```
95
96 IL_0069:  ldloc.2
97 IL_006a:  ldc.i4.1
98 IL_006b:  add
99 IL_006c:  stloc.2
100 IL_006d:  ldloc.2
101 IL_006e:  ldc.i4.5
102 IL_006f:  ble IL_000c
103
104 IL_0074:  ldstr "Foram inseridos "
105 IL_0079:  ldloc.3
106 IL_007a:  box [mscorlib]System.Int32
107 IL_007f:  ldstr " Homens"
108 IL_0084:  call string string::Concat(object, object, object
    )
109 IL_0089:  call void class [mscorlib]System.Console::
    WriteLine(string)
110 IL_008e:  ldstr "Foram inseridos "
111 IL_0093:  ldloc.s 4
112 IL_0095:  box [mscorlib]System.Int32
113 IL_009a:  ldstr " Mulheres"
114 IL_009f:  call string string::Concat(object, object, object
    )
115 IL_00a4:  call void class [mscorlib]System.Console::
    WriteLine(string)
116 IL_00a9:  ret
117     } // end of method micro05::Main
118
119 } // end of class micro05
```

### 7.18 Micro 06

Listing 60: Escreve um número lido por extenso(JavaScript)

```
1 var numero;
2 numero = prompt("Digite um numero de 1 a 5: ");
3 switch(numero){
4 case 1:
5 document.write("Um");
6 break;
7 case 2:
8 document.write("Dois");
9 break;
10 case 3:
11 document.write("Três");
```

## 7 CÓDIGOS

---

```
12 break;
13 case 4:
14 document.write("Quatro");
15 break;
16 case 5:
17 document.write("Cinco");
18 break;
19 default:
20 document.write("Numero invalido!");
21 break;
22 }
```

Listing 61: Escreve um número lido por extenso(C#)

```
1 using System;
2 class micro06 {
3     public static void Main(){
4         int numero;
5         Console.Write("Digite um número de 1 a 5: ");
6         numero = Convert.ToInt32(Console.ReadLine());
7         switch(numero){
8             case 1:
9                 Console.WriteLine("Um");
10                break;
11             case 2:
12                 Console.WriteLine("Dois");
13                break;
14             case 3:
15                 Console.WriteLine("Três");
16                break;
17             case 4:
18                 Console.WriteLine("Quatro");
19                break;
20             case 5:
21                 Console.WriteLine("Cinco");
22                break;
23             default:
24                 Console.WriteLine("Número Inválido!!!");
25                break;
26         }
27     }
28 }
```

Listing 62: Escreve um número lido por extenso(C#)

## 7 CÓDIGOS

---

```
1 using System;
2 class micro06 {
3     public static void Main(){
4         int numero;
5         Console.Write("Digite um número de 1 a 5: ");
6         numero = Convert.ToInt32(Console.ReadLine());
7         switch(numero){
8             case 1:
9                 Console.WriteLine("Um");
10                break;
11             case 2:
12                 Console.WriteLine("Dois");
13                break;
14             case 3:
15                 Console.WriteLine("Três");
16                break;
17             case 4:
18                 Console.WriteLine("Quatro");
19                break;
20             case 5:
21                 Console.WriteLine("Cinco");
22                break;
23             default:
24                 Console.WriteLine("Número Inválido!!!");
25                break;
26         }
27     }
28 }
```

Listing 63: Escreve um número lido por extenso(CIL)

```
1 .assembly extern mscorlib
2 {
3     .ver 4:0:0:0
4     .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
5 }
6 .assembly 'micro06'
7 {
8     .custom instance void class [mscorlib]System.Runtime.
          CompilerServices.RuntimeCompatibilityAttribute::.ctor
          '() = (
9         01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78    // ....
          T..WrapNonEx
10        63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01        ) //
          ceptionThrows.
```



## 7 CÓDIGOS

---

```
11
12 .hash algorithm 0x00008004
13 .ver 0:0:0:0
14 }
15 .module micro06.exe // GUID = {C6D580A0-F5D7-4265-A1A7-4
    A6EC7748578}
16
17
18 .class private auto ansi beforefieldinit micro06
19     extends [mscorlib]System.Object
20 {
21
22     // method line 1
23     .method public hidebysig specialname rtspecialname
24         instance default void '.ctor' () cil managed
25     {
26         // Method begins at RVA 0x2050
27     // Code size 7 (0x7)
28     .maxstack 8
29     IL_0000: ldarg.0
30     IL_0001: call instance void object::.ctor'()
31     IL_0006: ret
32     } // end of method micro06::.ctor
33
34     // method line 2
35     .method public static hidebysig
36         default void Main () cil managed
37     {
38         // Method begins at RVA 0x2058
39     .entrypoint
40     // Code size 145 (0x91)
41     .maxstack 2
42     .locals init (
43         int32 V_0)
44     IL_0000: ldstr bytearray (
45     44 00 69 00 67 00 69 00 74 00 65 00 20 00 75 00 // D.i.g.i
46         .t.e. .u.
47     6d 00 20 00 6e 00 fa 00 6d 00 65 00 72 00 6f 00 // m. .n
48         ...m.e.r.o.
49     20 00 64 00 65 00 20 00 31 00 20 00 61 00 20 00 // .d.e.
50         .1. .a. .
51     35 00 3a 00 20 00 01 ) // 5... ..
52
53     IL_0005: call void class [mscorlib]System.Console::Write(
54         string)
```

## 7 CÓDIGOS

---

```
51  IL_000a:  call string class [mscorlib]System.Console::
           ReadLine()
52  IL_000f:  call int32 class [mscorlib]System.Convert::
           ToInt32(string)
53  IL_0014:  stloc.0
54  IL_0015:  ldloc.0
55  IL_0016:  ldc.i4.1
56  IL_0017:  sub
57  IL_0018:  switch (
58      IL_0036,
59      IL_0045,
60      IL_0054,
61      IL_0063,
62      IL_0072)
63  IL_0031:  br IL_0081
64
65  IL_0036:  ldstr "Um"
66  IL_003b:  call void class [mscorlib]System.Console::
           WriteLine(string)
67  IL_0040:  br IL_0090
68
69  IL_0045:  ldstr "Dois"
70  IL_004a:  call void class [mscorlib]System.Console::
           WriteLine(string)
71  IL_004f:  br IL_0090
72
73  IL_0054:  ldstr bytearray (
74  54 00 72 00 ea 00 73 00 01 )           // T.r...s
           ..
75
76  IL_0059:  call void class [mscorlib]System.Console::
           WriteLine(string)
77  IL_005e:  br IL_0090
78
79  IL_0063:  ldstr "Quatro"
80  IL_0068:  call void class [mscorlib]System.Console::
           WriteLine(string)
81  IL_006d:  br IL_0090
82
83  IL_0072:  ldstr "Cinco"
84  IL_0077:  call void class [mscorlib]System.Console::
           WriteLine(string)
85  IL_007c:  br IL_0090
86
87  IL_0081:  ldstr bytearray (
```

## 7 CÓDIGOS

---

```
88  4e 00 fa 00 6d 00 65 00 72 00 6f 00 20 00 49 00  // N...m.e
      .r.o. .I.
89  6e 00 76 00 e1 00 6c 00 69 00 64 00 6f 00 21 00  // n.v...l
      .i.d.o.!.
90  21 00 21 00 01 )                                // !...
91
92  IL_0086:  call void class [mscorlib]System.Console::
      WriteLine(string)
93  IL_008b:  br IL_0090
94
95  IL_0090:  ret
96      } // end of method micro06::Main
97
98  } // end of class micro06
```

### 7.19 Micro 07

Listing 64: Decide se os números são positivos zeros ou negativos(JavaScript)

```
1 var programa, numero, opc;
2 programa = 1;
3 while(programa){
4  numero = prompt("Digite um numero: ");
5  if(numero > 0)
6  document.write("Positivo");
7  else{
8  if(numero == 0)
9  document.write("0 numero é igual a 0");
10 else{
11 if(numero < 0)
12 document.write("Negativo");
13 }
14 }
15 opc = prompt("Deseja finalizar? (S/N) ");
16 if(opc == 'S')
17 programa = 0;
18 }
```

Listing 65: Decide se os números são positivos zeros ou negativos(C#)

```
1 using System;
2 class micro07 {
3  public static void Main(){
4
5  int programa, numero;
```

## 7 CÓDIGOS

---

```
6 char opc;
7 programa = 1;
8 while ( programa == 1 ){
9 Console.WriteLine("Digite um número: ");
10 numero = Convert.ToInt32(Console.ReadLine());
11 if ( numero > 0 )
12 Console.WriteLine("Positivo");
13 else {
14 if ( numero == 0 )
15 Console.WriteLine("0 número é igual a 0");
16 if ( numero < 0 )
17 Console.WriteLine("Negativo");
18
19 }
20 Console.Write("Desja finalizar? (S/N) ");
21 opc = Convert.ToChar(Console.ReadLine());
22 if ( opc == 'S' )
23 programa = 0;
24
25 }
26 }
27 }
```

Listing 66: Decide se os números são positivos zeros ou negativos(CIL)

```
1 .assembly extern mscorlib
2 {
3   .ver 4:0:0:0
4   .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
5 }
6 .assembly 'micro07'
7 {
8   .custom instance void class [mscorlib]System.Runtime.
      CompilerServices.RuntimeCompatibilityAttribute::.ctor
      '() = (
9     01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78 // ....
      T..WrapNonEx
10    63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01      ) //
      ceptionThrows.
11
12   .hash algorithm 0x00008004
13   .ver 0:0:0:0
14 }
15 .module micro07.exe // GUID = {09933DC8-14E9-4C2F-9CF0-
      B1DBBC62502F}
```

## 7 CÓDIGOS

---

```
16
17
18 .class private auto ansi beforefieldinit micro07
19     extends [mscorlib]System.Object
20 {
21
22     // method line 1
23     .method public hidebysig specialname rtspecialname
24         instance default void '.ctor' () cil managed
25     {
26         // Method begins at RVA 0x2050
27     // Code size 7 (0x7)
28     .maxstack 8
29     IL_0000: ldarg.0
30     IL_0001: call instance void object::.ctor()
31     IL_0006: ret
32     } // end of method micro07::.ctor
33
34     // method line 2
35     .method public static hidebysig
36         default void Main () cil managed
37     {
38         // Method begins at RVA 0x2058
39     .entrypoint
40     // Code size 122 (0x7a)
41     .maxstack 2
42     .locals init (
43         int32 V_0,
44         int32 V_1,
45         char V_2)
46     IL_0000: ldc.i4.1
47     IL_0001: stloc.0
48     IL_0002: br IL_0072
49
50     IL_0007: ldstr bytearray (
51     44 00 69 00 67 00 69 00 74 00 65 00 20 00 75 00 // D.i.g.i
52     .t.e. .u.
53     6d 00 20 00 6e 00 fa 00 6d 00 65 00 72 00 6f 00 // m. .n
54     ...m.e.r.o.
55     3a 00 20 00 01 ) // :. ..
56
57     IL_000c: call void class [mscorlib]System.Console::
58         WriteLine(string)
59     IL_0011: call string class [mscorlib]System.Console::
60         ReadLine()
```

## 7 CÓDIGOS

---

```
57 IL_0016: call int32 class [mscorlib]System.Convert::
    ToInt32(string)
58 IL_001b: stloc.1
59 IL_001c: ldloc.1
60 IL_001d: ldc.i4.0
61 IL_001e: ble IL_0032
62
63 IL_0023: ldstr "Positivo"
64 IL_0028: call void class [mscorlib]System.Console::
    WriteLine(string)
65 IL_002d: br IL_0053
66
67 IL_0032: ldloc.1
68 IL_0033: brtrue IL_0042
69
70 IL_0038: ldstr bytearray (
71 4f 00 20 00 6e 00 fa 00 6d 00 65 00 72 00 6f 00 // 0. .n
    ...m.e.r.o.
72 20 00 e9 00 20 00 69 00 67 00 75 00 61 00 6c 00 // ... .i
    .g.u.a.l.
73 20 00 61 00 20 00 30 00 01 ) // .a.
    .0..
74
75 IL_003d: call void class [mscorlib]System.Console::
    WriteLine(string)
76 IL_0042: ldloc.1
77 IL_0043: ldc.i4.0
78 IL_0044: bge IL_0053
79
80 IL_0049: ldstr "Negativo"
81 IL_004e: call void class [mscorlib]System.Console::
    WriteLine(string)
82 IL_0053: ldstr "Desja finalizar? (S/N) "
83 IL_0058: call void class [mscorlib]System.Console::Write(
    string)
84 IL_005d: call string class [mscorlib]System.Console::
    ReadLine()
85 IL_0062: call char class [mscorlib]System.Convert::ToChar(
    string)
86 IL_0067: stloc.2
87 IL_0068: ldloc.2
88 IL_0069: ldc.i4.s 0x53
89 IL_006b: bne.un IL_0072
90
91 IL_0070: ldc.i4.0
```

## 7 CÓDIGOS

---

```
92 IL_0071: stloc.0
93 IL_0072: ldloc.0
94 IL_0073: ldc.i4.1
95 IL_0074: beq IL_0007
96
97 IL_0079: ret
98     } // end of method micro07::Main
99
100 } // end of class micro07
```

### 7.20 Micro 08

Listing 67: Decide se um número é maior ou menor que 10(JavaScript)

```
1 var numero;
2 numero = 1;
3 while(numero != 0){
4 numero = prompt("Digite um numero: ");
5 if(numero > 10)
6 document.write("O numero " +numero+ " é maior que 10");
7 else
8 document.write("O numero " +numero+ " é menor que 10");
9 }
```

Listing 68: Decide se um número é maior ou menor que 10(C#)

```
1 using System;
2 class micro08 {
3 public static void Main(){
4 int numero;
5 numero = 1;
6 while ( numero != 0 ) {
7 Console.WriteLine("Digite um número: ");
8 numero = Convert.ToInt32(Console.ReadLine());
9 if ( numero > 10 )
10 Console.WriteLine("O número " + numero + " é maior que 10");
11
12 else
13 Console.WriteLine("O número " + numero + " é menor que 10");
14 }
15 }
16 }
```

Listing 69: Decide se um número é maior ou menor que 10(CIL)

## 7 CÓDIGOS

---

```
1 .assembly extern mscorlib
2 {
3   .ver 4:0:0:0
4   .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
5 }
6 .assembly 'micro08'
7 {
8   .custom instance void class [mscorlib]System.Runtime.
      CompilerServices.RuntimeCompatibilityAttribute::.ctor
      '() = (
9     01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78    // ....
      T..WrapNonEx
10    63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01        ) //
      ceptionThrows.
11
12   .hash algorithm 0x00008004
13   .ver 0:0:0:0
14 }
15 .module micro08.exe // GUID = {24F28363-2C63-4D9B-BB72-8
      CF2C7E16F06}
16
17
18 .class private auto ansi beforefieldinit micro08
19   extends [mscorlib]System.Object
20 {
21
22   // method line 1
23   .method public hidebysig specialname rtspecialname
24     instance default void '.ctor' () cil managed
25   {
26     // Method begins at RVA 0x2050
27     // Code size 7 (0x7)
28     .maxstack 8
29     IL_0000: ldarg.0
30     IL_0001: call instance void object::.ctor'()
31     IL_0006: ret
32   } // end of method micro08::.ctor
33
34   // method line 2
35   .method public static hidebysig
36     default void Main () cil managed
37   {
38     // Method begins at RVA 0x2058
39     .entrypoint
40     // Code size 100 (0x64)
```



## 7 CÓDIGOS

---

```
41 .maxstack 3
42 .locals init (
43     int32 V_0)
44 IL_0000: ldc.i4.1
45 IL_0001: stloc.0
46 IL_0002: br IL_005d
47
48 IL_0007: ldstr bytearray (
49     44 00 69 00 67 00 69 00 74 00 65 00 20 00 75 00 // D.i.g.i
50     .t.e. .u.
51     6d 00 20 00 6e 00 fa 00 6d 00 65 00 72 00 6f 00 // m. .n
52     ...m.e.r.o.
53     3a 00 20 00 01 ) // :. .
54
55 IL_000c: call void class [mscorlib]System.Console::Write(
56     string)
57 IL_0011: call string class [mscorlib]System.Console::
58     ReadLine()
59 IL_0016: call int32 class [mscorlib]System.Convert::
60     ToInt32(string)
61 IL_001b: stloc.0
62 IL_001c: ldloc.0
63 IL_001d: ldc.i4.s 0x0a
64 IL_001f: ble IL_0043
65
66 IL_0024: ldstr bytearray (
67     4f 00 20 00 6e 00 fa 00 6d 00 65 00 72 00 6f 00 // 0. .n
68     ...m.e.r.o.
69     20 00 01 ) // ..
70
71 IL_0029: ldloc.0
72 IL_002a: box [mscorlib]System.Int32
73 IL_002f: ldstr bytearray (
74     20 00 e9 00 20 00 6d 00 61 00 69 00 6f 00 72 00 // ... .m
75     .a.i.o.r.
76     20 00 71 00 75 00 65 00 20 00 31 00 30 00 01 ) // .q.u.e
77     . .1.0..
78
79 IL_0034: call string string::Concat(object, object, object
80     )
81 IL_0039: call void class [mscorlib]System.Console::
82     WriteLine(string)
83 IL_003e: br IL_005d
84
85 IL_0043: ldstr bytearray (
```

## 7 CÓDIGOS

---

```
76  4f 00 20 00 6e 00 fa 00 6d 00 65 00 72 00 6f 00  // 0. .n
    ...m.e.r.o.
77  20 00 01 )                                     // ..
78
79  IL_0048:  ldloc.0
80  IL_0049:  box [mscorlib]System.Int32
81  IL_004e:  ldstr bytearray (
82  20 00 e9 00 20 00 6d 00 65 00 6e 00 6f 00 72 00  // ... .m
    .e.n.o.r.
83  20 00 71 00 75 00 65 00 20 00 31 00 30 00 01 )  // .q.u.e
    . .1.0..
84
85  IL_0053:  call string string::Concat(object, object, object
    )
86  IL_0058:  call void class [mscorlib]System.Console::
    WriteLine(string)
87  IL_005d:  ldloc.0
88  IL_005e:  brtrue IL_0007
89
90  IL_0063:  ret
91  } // end of method micro08::Main
92
93  } // end of class micro08
```

### 7.21 Micro 09

#### Listing 70: Cálculo de preços(JavaScript)

```
1 var preco, venda, novo_preco;
2 novo_preco = 0;
3 preco = prompt("Digite o preco: ");
4 venda = prompt("Digite a venda: ");
5 if(venda < 500 | preco < 30)
6 novo_preco = preco + 10/100 * preco;
7 eles if((venda >= 500 & venda < 1200) | (preco >= 30 & preco
    < 80))
8 novo_preco = preco + 15/10 * preco;
9 else if(venda >= 1200 | preco >= 80)
10 novo_preco = preco - 20/100 * preco;
11 document.write("O novo preco é " +novo_preco);
```

#### Listing 71: Cálculo de preços(C#)

```
1 using System;
2 class micro09 {
```

## 7 CÓDIGOS

---

```
3 public static void Main(){
4 double preco, venda, novo_preco;
5 novo_preco = 0;
6 Console.Write("Digite o preco: ");
7 preco = Convert.ToDouble(Console.ReadLine());
8 Console.Write("Digite a venda: ");
9 venda = Convert.ToDouble(Console.ReadLine());
10 if ( (venda < 500) || (preco < 30) )
11 novo_preco = preco + 10/100 * preco;
12 else if ( (venda >= 500 && venda < 1200) || (preco >= 30 &&
    preco < 80) )
13 novo_preco = preco + 15/100 * preco;
14 else if ( venda >= 1200 || preco >= 80 )
15 novo_preco = preco - 20/100 * preco;
16 Console.Write("O novo preco e " + novo_preco);
17 }
18 }
```

### Listing 72: Cálculo de preços(CIL)

```
1 .assembly extern mscorlib
2 {
3     .ver 4:0:0:0
4     .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
5 }
6 .assembly 'micro09'
7 {
8     .custom instance void class [mscorlib]System.Runtime.
        CompilerServices.RuntimeCompatibilityAttribute::.ctor
        '() = (
9         01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78 // ....
        T..WrapNonEx
10        63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01 //
        ceptionThrows.
11
12    .hash algorithm 0x00008004
13    .ver 0:0:0:0
14 }
15 .module micro09.exe // GUID = {18F4C6D1-F37A-470C-B18B-
    EFA7F29E4AAB}
16
17
18 .class private auto ansi beforefieldinit micro09
19     extends [mscorlib]System.Object
20 {
```

## 7 CÓDIGOS

---

```
21
22     // method line 1
23     .method public hidebysig specialname rtspecialname
24         instance default void '.ctor' () cil managed
25     {
26         // Method begins at RVA 0x2050
27     // Code size 7 (0x7)
28     .maxstack 8
29     IL_0000: ldarg.0
30     IL_0001: call instance void object::.ctor'()
31     IL_0006: ret
32     } // end of method micro09::.ctor
33
34     // method line 2
35     .method public static hidebysig
36         default void Main () cil managed
37     {
38         // Method begins at RVA 0x2058
39     .entrypoint
40     // Code size 246 (0xf6)
41     .maxstack 3
42     .locals init (
43         float64 V_0,
44         float64 V_1,
45         float64 V_2)
46     IL_0000: ldc.r8 0.
47     IL_0009: stloc.2
48     IL_000a: ldstr "Digite o preco: "
49     IL_000f: call void class [mscorlib]System.Console::Write(
50         string)
51     IL_0014: call string class [mscorlib]System.Console::
52         ReadLine()
53     IL_0019: call float64 class [mscorlib]System.Convert::
54         ToDouble(string)
55     IL_001e: stloc.0
56     IL_001f: ldstr "Digite a venda: "
57     IL_0024: call void class [mscorlib]System.Console::Write(
58         string)
59     IL_0029: call string class [mscorlib]System.Console::
60         ReadLine()
61     IL_002e: call float64 class [mscorlib]System.Convert::
62         ToDouble(string)
63     IL_0033: stloc.1
64     IL_0034: ldloc.1
65     IL_0035: ldc.r8 500.
```

## 7 CÓDIGOS

---

```
60  IL_003e:  blt  IL_0052
61
62  IL_0043:  ldloc.0
63  IL_0044:  ldc.r8 30.
64  IL_004d:  bge.un IL_0065
65
66  IL_0052:  ldloc.0
67  IL_0053:  ldc.r8 0.
68  IL_005c:  ldloc.0
69  IL_005d:  mul
70  IL_005e:  add
71  IL_005f:  stloc.2
72  IL_0060:  br  IL_00e0
73
74  IL_0065:  ldloc.1
75  IL_0066:  ldc.r8 500.
76  IL_006f:  blt.un IL_0083
77
78  IL_0074:  ldloc.1
79  IL_0075:  ldc.r8 1200.
80  IL_007e:  blt  IL_00a1
81
82  IL_0083:  ldloc.0
83  IL_0084:  ldc.r8 30.
84  IL_008d:  blt.un IL_00b4
85
86  IL_0092:  ldloc.0
87  IL_0093:  ldc.r8 80.
88  IL_009c:  bge.un IL_00b4
89
90  IL_00a1:  ldloc.0
91  IL_00a2:  ldc.r8 0.
92  IL_00ab:  ldloc.0
93  IL_00ac:  mul
94  IL_00ad:  add
95  IL_00ae:  stloc.2
96  IL_00af:  br  IL_00e0
97
98  IL_00b4:  ldloc.1
99  IL_00b5:  ldc.r8 1200.
100 IL_00be:  bge IL_00d2
101
102 IL_00c3:  ldloc.0
103 IL_00c4:  ldc.r8 80.
104 IL_00cd:  blt.un IL_00e0
```

## 7 CÓDIGOS

---

```
105
106 IL_00d2:  ldloc.0
107 IL_00d3:  ldc.r8 0.
108 IL_00dc:  ldloc.0
109 IL_00dd:  mul
110 IL_00de:  sub
111 IL_00df:  stloc.2
112 IL_00e0:  ldstr "0 novo preco e "
113 IL_00e5:  ldloc.2
114 IL_00e6:  box [mscorlib]System.Double
115 IL_00eb:  call string string::Concat(object, object)
116 IL_00f0:  call void class [mscorlib]System.Console::Write(
    string)
117 IL_00f5:  ret
118     } // end of method micro09::Main
119
120 } // end of class micro09
```

### 7.22 Micro 10

Listing 73: Calcula o fatorial de um número(JavaScript)

```
1 function fatorial(n){
2   if (n <= 0)
3     return 1;
4   else
5     return n * fatorial(n-1);
6 }
7
8 var numero, fat;
9 numero = prompt("Digite um numero: ");
10 fat = fatorial(numero);
11 document.write("0 fatorial de " +numero+ " é " +fat);
```

Listing 74: Calcula o fatorial de um número(C#)

```
1 using System;
2 class micro10 {
3   public static int fatorial (int n){
4     if ( n <= 0)
5       return 1;
6     else
7       return n * fatorial(n-1);
8   }
9 }
```

## 7 CÓDIGOS

---

```
10
11 public static void Main(){
12     int numero, fat;
13
14     Console.Write("Digite um número: ");
15     numero = Convert.ToInt32(Console.ReadLine());
16     fat = fatorial(numero);
17     Console.Write("O fatorial de ");
18     Console.Write(numero);
19     Console.Write(" é ");
20     Console.WriteLine(fat);
21 }
22 }
```

Listing 75: Calcula o fatorial de um número(CIL)

```
1 .assembly extern mscorlib
2 {
3     .ver 4:0:0:0
4     .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
5 }
6 .assembly 'micro10'
7 {
8     .custom instance void class [mscorlib]System.Runtime.
        CompilerServices.RuntimeCompatibilityAttribute::.ctor
        '() = (
9         01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78    // ....
        T..WrapNonEx
10        63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01          ) //
        ceptionThrows.
11
12     .hash algorithm 0x00008004
13     .ver 0:0:0:0
14 }
15 .module micro10.exe // GUID = {BBA21109-28D2-406B-8D4B-
        B6D586DD5E21}
16
17
18 .class private auto ansi beforefieldinit micro10
19     extends [mscorlib]System.Object
20 {
21
22     // method line 1
23     .method public hidebysig specialname rtspecialname
24         instance default void '.ctor' () cil managed
```

## 7 CÓDIGOS

---

```
25     {
26         // Method begins at RVA 0x2050
27     // Code size 7 (0x7)
28     .maxstack 8
29     IL_0000:  ldarg.0
30     IL_0001:  call instance void object::.ctor'()
31     IL_0006:  ret
32     } // end of method micro10::.ctor
33
34     // method line 2
35     .method public static hidebysig
36         default int32 factorial (int32 n)  cil managed
37     {
38         // Method begins at RVA 0x2058
39     // Code size 20 (0x14)
40     .maxstack 8
41     IL_0000:  ldarg.0
42     IL_0001:  ldc.i4.0
43     IL_0002:  bgt IL_0009
44
45     IL_0007:  ldc.i4.1
46     IL_0008:  ret
47     IL_0009:  ldarg.0
48     IL_000a:  ldarg.0
49     IL_000b:  ldc.i4.1
50     IL_000c:  sub
51     IL_000d:  call int32 class micro10::factorial(int32)
52     IL_0012:  mul
53     IL_0013:  ret
54     } // end of method micro10::factorial
55
56     // method line 3
57     .method public static hidebysig
58         default void Main ()  cil managed
59     {
60         // Method begins at RVA 0x2070
61     .entrypoint
62     // Code size 61 (0x3d)
63     .maxstack 1
64     .locals init (
65         int32 V_0,
66         int32 V_1)
67     IL_0000:  ldstr bytearray (
68     44 00 69 00 67 00 69 00 74 00 65 00 20 00 75 00  // D.i.g.i
        .t.e. .u.
```



## 7 CÓDIGOS

---

```
69  6d 00 20 00 6e 00 fa 00 6d 00 65 00 72 00 6f 00  // m. .n
    ...m.e.r.o.
70  3a 00 20 00 01 )                                // :. ..
71
72  IL_0005:  call void class [mscorlib]System.Console::Write(
    string)
73  IL_000a:  call string class [mscorlib]System.Console::
    ReadLine()
74  IL_000f:  call int32 class [mscorlib]System.Convert::
    ToInt32(string)
75  IL_0014:  stloc.0
76  IL_0015:  ldloc.0
77  IL_0016:  call int32 class micro10::fatorial(int32)
78  IL_001b:  stloc.1
79  IL_001c:  ldstr "0 fatorial de "
80  IL_0021:  call void class [mscorlib]System.Console::Write(
    string)
81  IL_0026:  ldloc.0
82  IL_0027:  call void class [mscorlib]System.Console::Write(
    int32)
83  IL_002c:  ldstr bytearray (
84  20 00 e9 00 20 00 01 )                            // ... ..
85
86  IL_0031:  call void class [mscorlib]System.Console::Write(
    string)
87  IL_0036:  ldloc.1
88  IL_0037:  call void class [mscorlib]System.Console::
    WriteLine(int32)
89  IL_003c:  ret
90  } // end of method micro10::Main
91
92  } // end of class micro10
```

### 7.23 Micro 11

Listing 76: Decide se um número é positivo zero ou negativo com auxílio de uma função(JavaScript)

```
1 function verifica(n){
2 var res;
3 if( n > 0)
4 res = 1;
5 else if(n < 0)
6 res = -1;
7 eles
```

## 7 CÓDIGOS

---

```
8 res = 0;
9 return res;
10 }
11
12 var numero, x;
13 numero = prompt("Digite um numero: ");
14 x = verifica(numero);
15 if(x == 1)
16 document.write("Numero positivo");
17 else if(x == 0)
18 document.write("Zero");
19 else
20 document.write("Numero negativo");
```

Listing 77: Decide se um número é positivo zero ou negativo com auxílio de uma função(C#)

```
1 using System;
2 class micro11 {
3     public static int verifica(int n){
4         int res;
5         if ( n > 0 )
6             res = 1;
7         else if ( n < 0)
8             res = -1;
9         else
10            res = 0;
11         return res;
12     }
13     public static void Main(){
14         int numero, x;
15
16         Console.Write("Digite um número: ");
17         numero = Convert.ToInt32(Console.ReadLine());
18         x = verifica(numero);
19         if ( x == 1 )
20             Console.WriteLine("Numero positivo");
21         else if ( x == 0)
22             Console.WriteLine("Zero");
23         else Console.WriteLine("Numero negativo");
24     }
25 }
```

## 7 CÓDIGOS

---

Listing 78: Decide se um número é positivo zero ou negativo com auxílio de uma função(CIL)

```
1 .assembly extern mscorlib
2 {
3     .ver 4:0:0:0
4     .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
5 }
6 .assembly 'micro11'
7 {
8     .custom instance void class [mscorlib]System.Runtime.
        CompilerServices.RuntimeCompatibilityAttribute::.ctor
        '() = (
9         01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78    // ....
        T..WrapNonEx
10        63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01         ) //
        ceptionThrows.
11
12    .hash algorithm 0x00008004
13    .ver 0:0:0:0
14 }
15 .module micro11.exe // GUID = {5A9256D0-682E-43BF-AB66-
    CE37CDFF3D0A}
16
17
18 .class private auto ansi beforefieldinit micro11
19     extends [mscorlib]System.Object
20 {
21
22     // method line 1
23     .method public hidebysig specialname rtspecialname
24         instance default void '.ctor' () cil managed
25     {
26         // Method begins at RVA 0x2050
27         // Code size 7 (0x7)
28         .maxstack 8
29         IL_0000: ldarg.0
30         IL_0001: call instance void object::.ctor'()
31         IL_0006: ret
32     } // end of method micro11::.ctor
33
34     // method line 2
35     .method public static hidebysig
36         default int32 verifica (int32 n) cil managed
37     {
38         // Method begins at RVA 0x2058
```

## 7 CÓDIGOS

---

```
39 // Code size 32 (0x20)
40 .maxstack 2
41 .locals init (
42     int32 V_0)
43 IL_0000: ldarg.0
44 IL_0001: ldc.i4.0
45 IL_0002: ble IL_000e
46
47 IL_0007: ldc.i4.1
48 IL_0008: stloc.0
49 IL_0009: br IL_001e
50
51 IL_000e: ldarg.0
52 IL_000f: ldc.i4.0
53 IL_0010: bge IL_001c
54
55 IL_0015: ldc.i4.m1
56 IL_0016: stloc.0
57 IL_0017: br IL_001e
58
59 IL_001c: ldc.i4.0
60 IL_001d: stloc.0
61 IL_001e: ldloc.0
62 IL_001f: ret
63 } // end of method micro11::verifica
64
65 // method line 3
66 .method public static hidebysig
67     default void Main () cil managed
68 {
69     // Method begins at RVA 0x2084
70 .entrypoint
71 // Code size 82 (0x52)
72 .maxstack 2
73 .locals init (
74     int32 V_0,
75     int32 V_1)
76 IL_0000: ldstr bytearray (
77 44 00 69 00 67 00 69 00 74 00 65 00 20 00 75 00 // D.i.g.i
78     .t.e. .u.
79 6d 00 20 00 6e 00 fa 00 6d 00 65 00 72 00 6f 00 // m. .n
80     ...m.e.r.o.
81 3a 00 20 00 01 ) // :. .
82 IL_0005: call void class [mscorlib]System.Console::Write(
```

## 7 CÓDIGOS

---

```
        string)
82  IL_000a:  call string class [mscorlib]System.Console::
        ReadLine()
83  IL_000f:  call int32 class [mscorlib]System.Convert::
        ToInt32(string)
84  IL_0014:  stloc.0
85  IL_0015:  ldloc.0
86  IL_0016:  call int32 class micro11::verifica(int32)
87  IL_001b:  stloc.1
88  IL_001c:  ldloc.1
89  IL_001d:  ldc.i4.1
90  IL_001e:  bne.un IL_0032
91
92  IL_0023:  ldstr "Numero positivo"
93  IL_0028:  call void class [mscorlib]System.Console::
        WriteLine(string)
94  IL_002d:  br IL_0051
95
96  IL_0032:  ldloc.1
97  IL_0033:  brtrue IL_0047
98
99  IL_0038:  ldstr "Zero"
100 IL_003d:  call void class [mscorlib]System.Console::
        WriteLine(string)
101 IL_0042:  br IL_0051
102
103 IL_0047:  ldstr "Numero negativo"
104 IL_004c:  call void class [mscorlib]System.Console::
        WriteLine(string)
105 IL_0051:  ret
106     } // end of method micro11::Main
107
108 } // end of class micro11
```

## 8 Construção do Compilador

Esta seção apresenta as etapas de implementação necessárias para a construção do compilador.

### 8.1 Analisador Léxico

A análise léxica é a primeira etapa do processo de compilação. Nesta etapa, o código fonte do programa a ser compilado é varrido caractere por caractere e cada palavra reservada, constante, identificador ou outras palavras que pertencem a linguagem de programação, são traduzidas para uma sequência de símbolos chamados "tokens".

Além de extrair e classificar tokens, a análise léxica também elimina espaços em branco e comentários e identifica erros léxicos, simplificando a próxima etapa do processo de compilação, a análise sintática.

Neste trabalho será usado o gerador de analisadores lexicais da linguagem Ocaml. o Ocamllex. Este gerador produz o código do analisador léxico a partir de uma especificação lexical. Tal especificação deve conter a definição dos tokens da linguagem (representados por expressões regulares) e a ação tomada para cada token.

### 8.2 Especificação Lexical para MiniJavaScript

O código em Ocaml a seguir apresenta uma especificação lexical simplificada para a linguagem Javascript. Este código deve ser salvo como um arquivo no formato ".mll".

Listing 79: Especificação Lexical

```
1 {  
2   open Lexing  
3   open Printf  
4  
5   let incr_num_linha lexbuf =  
6     let pos = lexbuf.lex_curr_p in  
7     lexbuf.lex_curr_p <- { pos with  
8       pos_lnum = pos.pos_lnum + 1;  
9       pos_bol = pos.pos_cnum;  
10    }  
11  
12   let msg_erro lexbuf c =
```

## 8 CONSTRUÇÃO DO COMPILADOR

---

```
13     let pos = lexbuf.lex_curr_p in
14     let lin = pos.pos_lnum
15     and col = pos.pos_cnum - pos.pos_bol - 1 in
16     sprintf "%d-%d: caracter desconhecido %c" lin col c
17
18     let erro lin col msg =
19         let mensagem = sprintf "%d-%d: %s" lin col msg in
20             failwith mensagem
21
22
23 type tokens = APAR
24             | FPAR
25             | ACHAVE
26             | FCHAVE
27             | ACOLCH
28             | FCOLCH
29             | MAIS
30             | MENOS
31             | MULT
32             | DIV
33             | MOD
34             | POT
35             | MAIOR
36             | MENOR
37             | ATRIB
38             | IGUAL
39             | MAISATRIB
40             | MENOSATRIB
41             | DIVATRIB
42             | MULTATRIB
43             | MENORIGUAL
44             | MAIORIGUAL
45             | INCR
46             | DECR
47             | AND
48             | OR
49             | NOT
50             | VIRG
51             | PONTOVIRG
52             | DIF
53             | FOR
54             | IF
55             | ELSE
56             | WHILE
57             | SWITCH
```

## 8 CONSTRUÇÃO DO COMPILADOR

---

```
58         | CASE
59         | BREAK
60         | DEFAULT
61         | CONSOLELOG
62         | FUNCTION
63         | RETURN
64         | LITINT of int
65         | LITFLOAT of float
66         | LITSTRING of string
67         | VAR
68         | LET
69         | ID of string
70         | NEW
71         | OBJ
72         | PONTO
73         | DOISPTO
74         | CONT
75         | DO
76         | CLASS
77         | CONST
78         | IN
79         | OF
80         | NULL
81         | TRUE
82         | FALSE
83         | PROMPT
84         | EOF
85     }
86
87 let digito = ['0' - '9']
88 let inteiro = digito+
89 let real = (digito+ '.' digito+ | '.' digito+)
90 let letra = ['a' - 'z' 'A' - 'Z']
91 let identificador = letra ( letra | digito | '_' ) *
92
93 let brancos = [' ' '\t'] +
94 let novalinha = '\r' | '\n' | "\r\n"
95
96 let comentario = "//" [^ '\r' '\n' ] *
97
98 rule token = parse
99     brancos          { token lexbuf }
100 | novalinha         { incr_num_linha lexbuf; token lexbuf }
101 | comentario        { token lexbuf }
102 | "/" *             { comentario_bloco 0 lexbuf }
```



## 8 CONSTRUÇÃO DO COMPILADOR

---

```
103 | '('          { APAR }
104 | ')'          { FPAR }
105 | '{'          { ACHAVE }
106 | '}'          { FCHAVE }
107 | '['          { ACOLCH }
108 | ']'          { FCOLCH }
109 | '+'          { MAIS }
110 | '-'          { MENOS }
111 | '*'          { MULT }
112 | '/'          { DIV }
113 | '%'          { MOD }
114 | "**"          { POT }
115 | '='          { ATRIB }
116 | "+="         { MAISATRIB }
117 | "-="         { MENOSATRIB }
118 | "/="         { DIVATRIB }
119 | "*="         { MULTATRIB }
120 | '>'          { MAIOR }
121 | '<'          { MENOR }
122 | "<="         { MENORIGUAL }
123 | ">="         { MAIORIGUAL }
124 | "=="         { IGUAL }
125 | "++"         { INCR }
126 | "--"         { DECR }
127 | "&&"         { AND }
128 | "||"         { OR }
129 | '!'          { NOT }
130 | "!="         { DIF }
131 | ','          { VIRG }
132 | ';'          { PONTOVIRG }
133 | '.'          { PONTO }
134 | ':'          { DOISPTO }
135 | "for"        { FOR }
136 | "if"         { IF }
137 | "else"       { ELSE }
138 | "switch"     { SWITCH }
139 | "case"       { CASE }
140 | "break"      { BREAK }
141 | "default"    { DEFAULT }
142 | "console.log" { CONSOLELOG }
143 | "function"   { FUNCTION }
144 | "return"     { RETURN }
145 | "var"        { VAR }
146 | "while"      { WHILE }
147 | "new"        { NEW }
```

## 8 CONSTRUÇÃO DO COMPILADOR

---

```
148 | "let"                { LET }
149 | "Object"            { OBJ }
150 | "continue"          { CONT }
151 | "do"                 { DO }
152 | "in"                 { IN }
153 | "of"                 { OF }
154 | "class"              { CLASS }
155 | "const"              { CONST }
156 | "null"               { NULL }
157 | "true"               { TRUE }
158 | "false"              { FALSE }
159 | "prompt"             { PROMPT }
160 | '"'                  { let pos = lexbuf.lex_curr_p in
161                        let lin = pos.pos_lnum
162                        and col = pos.pos_cnum - pos.pos_bol - 1
                                in
163                        let buffer = Buffer.create 1 in
164                        let str = leia_string lin col buffer
                                lexbuf in
165                        LITSTRING str }
166 | inteiro as num      { let numero = int_of_string num in
167                        LITINT numero }
168 | real as num          { let numero = float_of_string num in
169                        LITFLOAT numero }
170 | identificador as id { ID id }
171 | _ as c               { failwith (msg_erro lexbuf c) }
172 | eof                  { EOF }
173
174 and leia_string lin col buffer = parse
175 '"' { Buffer.contents buffer}
176 | "\\t" { Buffer.add_char buffer '\t';
177 leia_string lin col buffer lexbuf }
178 | "\\n" { Buffer.add_char buffer '\n';
179 leia_string lin col buffer lexbuf }
180 | '\\ ' '"' { Buffer.add_char buffer '"';
181 leia_string lin col buffer lexbuf }
182 | '\\ ' '\\' { Buffer.add_char buffer '\\';
183 leia_string lin col buffer lexbuf }
184 | _ as c { Buffer.add_char buffer c;
185 leia_string lin col buffer lexbuf }
186 | eof { erro lin col "A string não foi fechada"}
187
188 and comentario_bloco n = parse
189 "*/" { if n=0 then token lexbuf
190        else comentario_bloco (n-1) lexbuf }
```

## 8 CONSTRUÇÃO DO COMPILADOR

---

```
191 | "/"*      { failwith "Comentarios aninhados nao permitidos"
      }
192 | novalinha { incr_num_linha lexbuf;
193              comentario_bloco n lexbuf }
194 | _         { comentario_bloco n lexbuf }
195 | eof       { failwith "Coment ário não fechado" }
```

Listing 80: Carregadorl

```
1 {
2  \#load "lexico.cmo";;
3
4  let rec tokens lexbuf =
5    let tok = Lexico.token lexbuf in
6    match tok with
7    | Lexico.EOF -> [Lexico.EOF]
8    | _ -> tok :: tokens lexbuf
9  ;;
10
11 let lexico str =
12   let lexbuf = Lexing.from_string str in
13   tokens lexbuf
14 ;;
15
16 let lex arq =
17   let ic = open_in arq in
18   let lexbuf = Lexing.from_channel ic in
19   let toks = tokens lexbuf in
20   let _ = close_in ic in
21   toks
```

### 8.3 Geração do Código do Analisador Léxico

A geração do código do analisador léxico usando o Ocamllex é feita utilizando os seguintes comandos:

```
> ocamllex lexicoJS.mll
> ocamlc -c lexicoJS.ml
```

Onde "lexicoJS.mll" é o nome do arquivo que contém a especificação lexical. Após utilizar estes comandos será gerado um arquivo "lexicoJS.ml", o qual de fato apresenta o código do analisador léxico.

### 8.4 Teste do Analisador Léxico

Para testar o funcionamento do analisador léxico foi criado um arquivo "teste.js" contendo um código em JavaScript. O analisador léxico varrerá este arquivo, caractere por caractere, e produzirá uma sequência de tokens. Para isso, utiliza-se os comandos:

```
> rlwrap ocaml
>#use "carregador.ml";
> lex "teste.js";;
```

```
1 {
2 function factorial(num)
3 {
4     // coment
5     if (num < 0) {
6         return -1;
7     }
8     else if (num == 0) {
9         return 1;
10    }
11    else {
12        return (num * factorial(num - 1));
13    }
14 }
15
16 var result = factorial(8);
17 document.write(result);
```

Para o código em JavaScript acima foi gerada a seguinte sequência de tokens:

```
- : LexicoJS.tokens list =
[LexicoJS.FUNCTION; LexicoJS.ID "factorial"; LexicoJS.APAR;
 LexicoJS.ID "num"; LexicoJS.FPAR; LexicoJS.ACHAVE; LexicoJS.
  IF;
 LexicoJS.APAR; LexicoJS.ID "num"; LexicoJS.MENOR; LexicoJS.
  LITINT 0;
 LexicoJS.FPAR; LexicoJS.ACHAVE; LexicoJS.RETURN; LexicoJS.
  MENOS;
 LexicoJS.LITINT 1; LexicoJS.PONTOVIRG; LexicoJS.FCHAVE;
  LexicoJS.ELSE;
 LexicoJS.IF; LexicoJS.APAR; LexicoJS.ID "num"; LexicoJS.
  IGUAL;
 LexicoJS.LITINT 0; LexicoJS.FPAR; LexicoJS.ACHAVE; LexicoJS.
  RETURN;
```

## 8 CONSTRUÇÃO DO COMPILADOR

---

```
LexicoJS.LITINT 1; LexicoJS.PONTOVIRG; LexicoJS.FCHAVE;  
    LexicoJS.ELSE;  
LexicoJS.ACHAVE; LexicoJS.RETURN; LexicoJS.APAR; LexicoJS.ID  
    "num";  
LexicoJS.MULT; LexicoJS.ID "factorial"; LexicoJS.APAR;  
    LexicoJS.ID "num";  
LexicoJS.MENOS; LexicoJS.LITINT 1; LexicoJS.FPAR; LexicoJS.  
    FPAR;  
LexicoJS.PONTOVIRG; LexicoJS.FCHAVE; LexicoJS.FCHAVE;  
    LexicoJS.VAR;  
LexicoJS.ID "result"; LexicoJS.ATRIB; LexicoJS.ID "factorial  
    ";  
LexicoJS.APAR; LexicoJS.LITINT 8; LexicoJS.FPAR; LexicoJS.  
    PONTOVIRG;  
LexicoJS.CONSOLELOG; LexicoJS.APAR; LexicoJS.ID "result";  
    LexicoJS.FPAR;  
LexicoJS.PONTOVIRG; LexicoJS.EOF]
```

## 9 Bibliografia

1. *Guia do .NET - CLR (Common Language Runtime)*
2. *.Net Framework - Common Language Runtime)*
3. *Understanding Common Intermediate Language (CIL)*
4. *Documentação do Mono*
5. *Lista de instruções CIL*
6. *Trabalho de Construção de Compiladores - Portugol para CLR (Eduardo Costa de Paiva)*