ESQUEMA DE TRADUÇÃO nº1

```
<expressão>::=      <termo>  <expressão_>
<expressão_>::=     +  <termo>  #1  <expressão_>
                  |  -  <termo>  #1  <expressão_>
                  |  ε
-----------------------------------------------------
<termo>::=          <elemento>  <termo_>
<termo_>::=         *  <elemento>  #1  <termo_>
                  |  /  <elemento>  #2  <termo_>
                  |  ε
-----------------------------------------------------
<elemento>::=       constante.inteira     #3
                  |  constante.real        #4
                  |  (  <expressão>  )
```

registros semânticos:

`pilha` // pilha de tipos

ações semânticas:

ação #1:

```
tipo1:= pilha.desempilha
tipo2:= pilha.desempilha
se (tipo1=float64) ou (tipo2=float64)
então pilha.empilha (float64)
senão pilha.empilha (int64)
```

ação #2:

```
pilha.desempilha
pilha.desempilha
pilha.empilha (float64)
```

ação #3:

```
pilha.empilha (int64)
```

ação #4:

```
pilha.empilha (float64)
```

ESQUEMA DE TRADUÇÃO nº2

```
<expressão>::=      <termo>  <expressão_>
<expressão_>::=     +  <termo>  #1  <expressão_>
                  |  -  <termo>  #2  <expressão_>
                  |  ε
-----------------------------------------------------
<termo>::=          <elemento>  <termo_>
<termo_>::=         *  <elemento>  #3  <termo_>
                  |  /  <elemento>  #4  <termo_>
                  |  ε
-----------------------------------------------------
<elemento>::=       constante.inteira     #5
                  |  constante.real        #6
                  |  (  <expressão>  )
```

registros semânticos:

`código` // código gerado

ações semânticas:

ação #1:

```
código.adiciona (add)
```

ação #2:

```
código.adiciona (sub)
```

ação #3:

```
código.adiciona (mul)
```

ação #4:

```
código.adiciona (div)
```

ação #5:

```
código.adiciona (ldc.i8 token.getLexeme)
```

ação #6:

```
código.adiciona (ldc.r8 token.getLexeme)
```

ESQUEMA DE TRADUÇÃO nº3

DESCRIÇÃO TEXTUAL:

- determinar o tipo de uma **<expressão>**, conforme descrito no esquema de tradução nº1, sendo que os operandos do operador de divisão devem ser do mesmo tipo (**int64**, **int64** ou **float64**, **float64**) e o resultado do tipo correspondente (**int64** ou **float64**, respectivamente);
- gerar código para uma **<expressão>**, conforme descrito no esquema de tradução nº2;

<expressão>::=	<termo> <expressão_>	
<expressão_>::=	**+** <termo> **#1** <expressão_>	
	| **-** <termo> **#2** <expressão_>	
	| ε	
<termo>::=	<elemento> <termo_>	
<termo_>::=	* <elemento> **#3** <termo_>	
	| / <elemento> **#4** <termo_>	
	| ε	
<elemento>::=	constante.inteira	**#5**
	| constante.real	**#6**
	| **(** <expressão> **)**	

registros semânticos:

código // código Gerado
pilha // pilha de tipos

ações semânticas:

ação #1:
```
tipo1:= pilha.desempilha
tipo2:= pilha.desempilha
se (tipo1=float64) ou (tipo2=float64)
então pilha.empilha (float64)
senão pilha.empilha (int64)
código.adiciona (add)
```

ação #2:
```
tipo1:= pilha.desempilha
tipo2:= pilha.desempilha
se (tipo1=float64) ou (tipo2=float64)
então pilha.empilha (float64)
senão pilha.empilha (int64)
código.adiciona (sub)
```

ação #3:
```
tipo1:= pilha.desempilha
tipo2:= pilha.desempilha
se (tipo1=float64) ou (tipo2=float64)
então pilha.empilha (float64)
senão pilha.empilha (int64)
código.adiciona (mul)
```

ação #4:
```
tipo1:= pilha.desempilha
tipo2:= pilha.desempilha
se (tipo1=tipo2)
então pilha.empilha (tipo1)
senão erro semântico, parar
código.adiciona (div)
```

ação #5:
```
pilha.empilha (int64)
código.adiciona (ldc.i8 token.getLexeme)
código.adiciona (conv.r8)
```

ação #6:
```
pilha.empilha (float64)
código.adiciona (ldc.r8 token.getLexeme)
```

ESQUEMA DE TRADUÇÃO nº4 – com operadores unários

DESCRIÇÃO TEXTUAL:

- determinar o tipo de uma **<expressão>** com operadores unários, sendo que o operando deve ser do **`int64`** ou **`float64`** e o resultado do tipo correspondente (**`int64`** ou **`float64`**, respectivamente);
- gerar código para uma **<expressão>** com operadores unários;

<expressão>::=	<termo> <expressão_>	
<expressão_>::=	**+** <termo> **#1** <expressão_>	
	| **-** <termo> **#2** <expressão_>	
	| ε	
<termo>::=	<elemento> <termo_>	
<termo_>::=	***** <elemento> **#3** <termo_>	
	| **/** <elemento> **#4** <termo_>	
	| ε	
<elemento>::=	constante.inteira	**#5**
	| constante.real	**#6**
	| **(** <expressão> **)**	
	| **+** <elemento>	**#7**
	| **-** <elemento>	**#8**

ação #7:

```
tipo:= pilha.desempilha
se (tipo=float64) ou (tipo=int64)
então pilha.empilha (tipo)
senão erro semântico, parar
```

ação #8:

```
tipo:= pilha.desempilha
se (tipo=float64) ou (tipo=int64)
então pilha.empilha (tipo)
senão erro semântico, parar
código.adiciona (ldc.i8 -1)
se (tipo=int64)
então código.adiciona (conv.r8)
código.adiciona (mul)
```

ESQUEMA DE TRADUÇÃO nº5 – com operadores relacionais

DESCRIÇÃO TEXTUAL:

- determinar o tipo de uma expressão **<relacional>**, sendo que os operandos devem ser do mesmo tipo e o resultado do tipo **bool**;
- gerar código para uma expressão **<relacional>**;

<relacional>::=	<expressão> <operador> **#9** <expressão> **#10**	
|	<expressão>	
<operador>::=	**< | > | =**	
<expressão>::=	<termo> <expressão_>	
<expressão_>::=	**+** <termo> **#1** <expressão_>	
|	**-** <termo> **#2** <expressão_>	
|	ε	
<termo>::=	<elemento> <termo_>	
<termo_>::=	***** <elemento> **#3** <termo_>	
|	**/** <elemento> **#4** <termo_>	
|	ε	
<elemento>::=	constante.inteira	**#5**
|	constante.real	**#6**
|	**(** <expressão> **)**	
|	**+** <elemento>	**#7**
|	**-** <elemento>	**#8**

registros semânticos:

`operador` // operador relacional

ações semânticas:

ação #9:

```
operador:= token.getlexeme
```

ação #10:

```
tipo1:= pilha.desempilha
tipo2:= pilha.desempilha
se (tipo1=tipo2)
então pilha.empilha (bool)
senão erro semântico, parar
caso operador
  >  código.adiciona (cgt)
  <  código.adiciona (clt)
  =  código.adiciona (ceq)
```

ESQUEMA DE TRADUÇÃO nº6 – com constantes lógicas e `not`

DESCRIÇÃO TEXTUAL:

- o tipo de uma constante lógica (**true** ou **false**) é **`bool`**;
- o tipo de uma expressão **<lógica>** envolvendo o operador not é **`bool`**, sendo que o operando deve ser do tipo **`bool`**;
- gerar código para uma expressão **<lógica>**;

<lógica>::=	**true**	**#11**
|	**false**	**#12**
|	**not (<lógica>)**	**#13**
|	<relacional>	
<relacional>::=	<expressão> <operador> **#9** <expressão> **#10**	
|	<expressão>	
<operador>::=	**< | > | =**	
<expressão>::=	<termo> <expressão_>	
<expressão_>::=	**+** <termo> **#1** <expressão_>	
|	**-** <termo> **#2** <expressão_>	
|	ε	
<termo>::=	<elemento> <termo_>	
<termo_>::=	***** <elemento> **#3** <termo_>	
|	**/** <elemento> **#4** <termo_>	
|	ε	
<elemento>::=	constante.inteira	**#5**
|	constante.real	**#6**
|	**(** <expressão> **)**	
|	**+** <elemento>	**#7**
|	**-** <elemento>	**#8**

ação #11

```
pilha.empilha (bool)
código.adiciona (ldc.i4.1)
```

ação #12

```
pilha.empilha (bool)
código.adiciona (ldc.i4.0)
```

ação #13:

```
tipo:= pilha.desempilha
se (tipo=bool)
então pilha.empilha (bool)
senão erro semântico, parar
código.adiciona (ldc.i4.1)
código.adiciona (xor)
```

ESQUEMA DE TRADUÇÃO nº7 – com comando de saída

<saída>	::= **write** **(** <lista expressão> **)**	
<lista expressão>	::= <lógica> **#14**	
	| <lógica> **#14** **,** <lista expressão>	
<lógica>	::= **true**	**#11**
	| **false**	**#12**
	| **not (** <lógica> **)**	**#13**
	| <relacional>	
<relacional>	::= <expressão> <operador> **#9** <expressão> **#10**	
	| <expressão>	
<operador>	::= **<** | **>** | **=**	
<expressão>	::= <termo> <expressão_>	
<expressão_>	::= **+** <termo> **#1** <expressão_>	
	| **-** <termo> **#2** <expressão_>	
	| ε	
<termo>	::= <elemento> <termo_>	
<termo_>	::= ***** <elemento> **#3** <termo_>	
	| **/** <elemento> **#4** <termo_>	
	| ε	
<elemento>	::= constante.inteira	**#5**
	| constante.real	**#6**
	| **(** <expressão> **)**	
	| **+** <elemento>	**#7**
	| **-** <elemento>	**#8**

ação #14:

```
tipo:= pilha.desempilha
se (tipo=int64)
então código.adiciona (conv.i8)
código.adiciona ("call void [mscorlib]System.Console::Write(" + tipo + ")")
```

ação #15:

```
código.adiciona(
  .assembly extern mscorlib {}
  .assembly _codigo_objeto{}
  .module   _codigo_objeto.exe

  .class public _UNICA{
  .method static public void _principal() {
    .entrypoint
)
```

ação #16:

```
código.adiciona(
      ret
      }
  }
)
```