4ª Aula Prática de Compiladores: Geração de Código Objeto com CIL e ILAsm

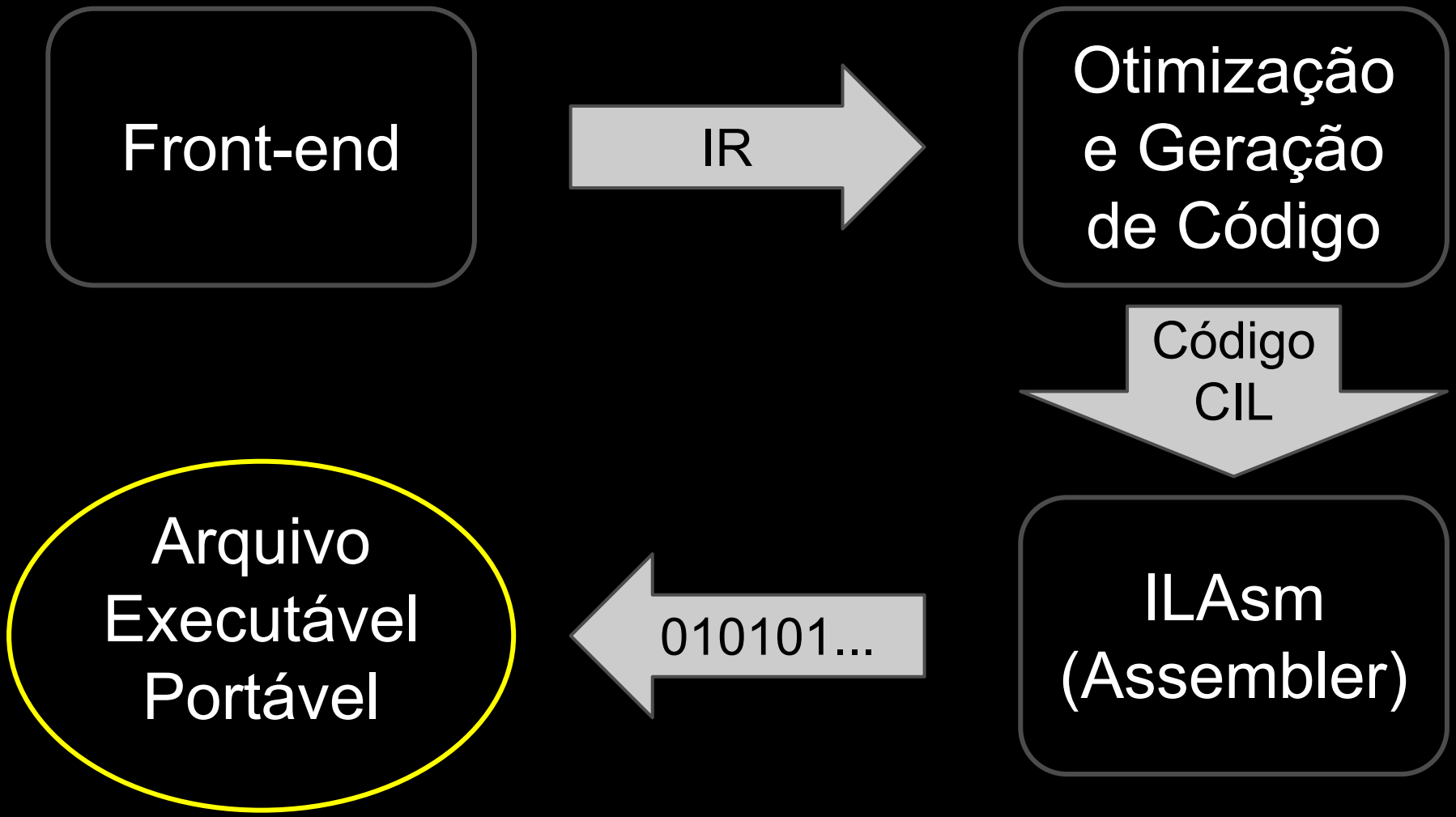
Roteiro

- CIL / ILAsm
- Referências de CIL
- Instruções Básicas
- Gerando Código CIL
- Execução
- Exercícios Práticos

CIL / ILAsm

- CIL: Common Intermediate Language
  - Também chamada de MSIL (Microsoft Intermediate Language)
  - Linguagem de baixo nível, similar ao bytecode Java
  - Definida pela especificação CLI (Common Language Infrastructure)
  - Usada por: .NET, Mono
- ILAsm: Ferramenta (Assembler) para geração de executáveis a partir de código CIL

CIL / ILAsm


Front-end
IR
Otimização e Geração de Código
Código CIL
ILAsm (Assembler)
010101...
Arquivo Executável Portável

Referências de CIL

- Artigo do codeguru sobre MSIL/CIL
- Lista de Instruções
- Especificação da CIL (Partition III)

Instruções Básicas

- `add, sub, mul, div` - Operações aritméticas
- `clt, cgt, ceq` - Comparação entre valores (<, > e ==)
- `call method` - Chamada a um método

Instruções Básicas

- `ldc.i4 x` - Carrega a constante inteira *x* na pilha
- `ldstr x` - Carrega a string *x* na pilha
- `ldloc x` - Carrega a variável local *x* na pilha
- `stloc.s x` - Armazena o valor do topo da pilha na variável local *x*

Instruções Básicas

- `br label` - Desvia a execução para *label*
- `brfalse label` - Desvia para *label* se no topo da pilha for 0
- `brtrue label` - Desvia para *label* se no topo da pilha for 1
- `ret` - Retorna o valor do topo da pilha

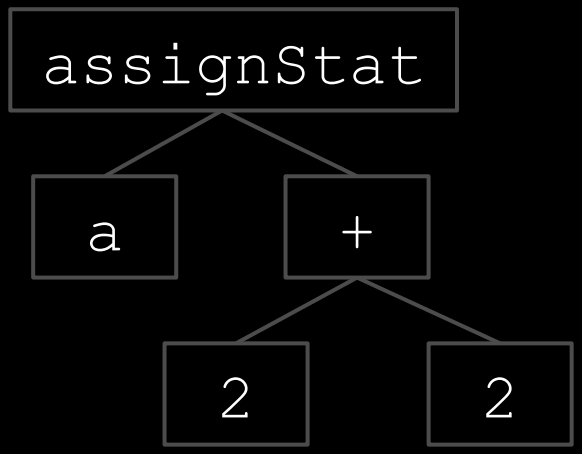
Gerando Código CIL

- Método mais simples: visite a árvore e gere o código para cada nó
- Ex.: `a = 2 + 2;`

Gerando Código CIL

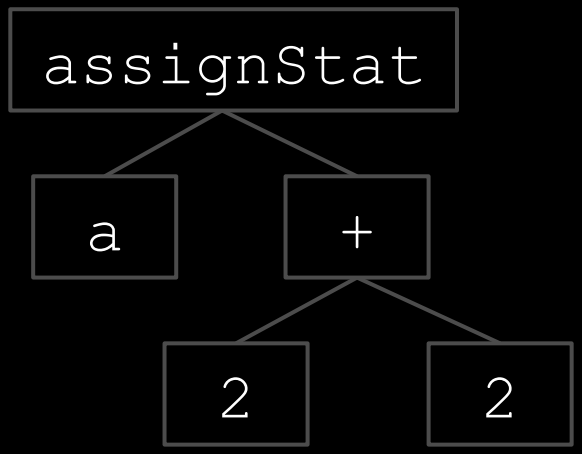
- Método mais simples: visite a árvore e gere o código para cada nó

- Ex.: `a = 2 + 2;`



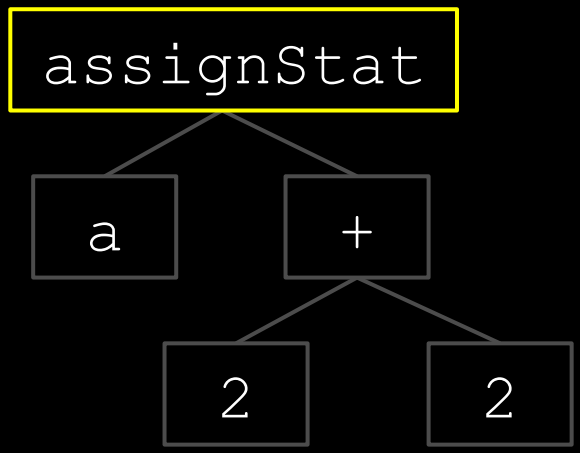
Gerando Código CIL

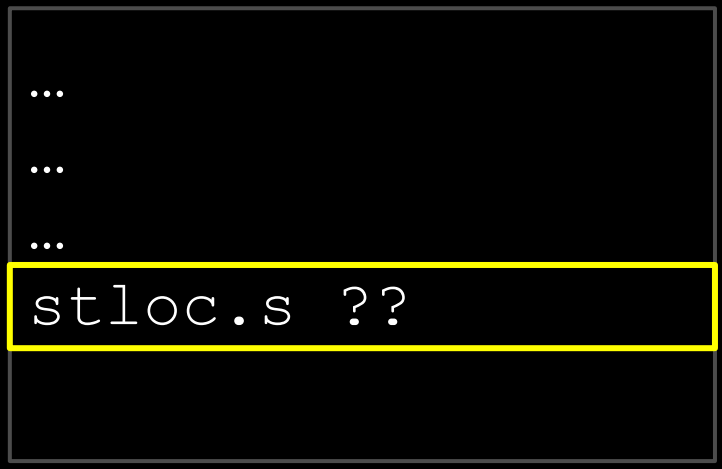
- Método mais simples: visite a árvore e gere o código para cada nó
- Ex.: `a = 2 + 2;`



Gerando Código CIL

- Método mais simples: visite a árvore e gere o código para cada nó
- Ex.: `a = 2 + 2;`

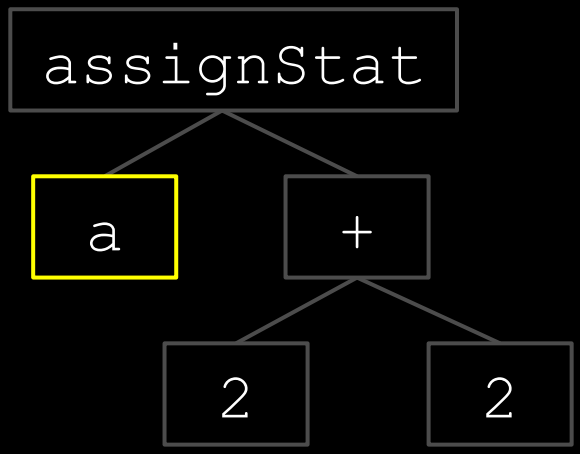




Gerando Código CIL

- Método mais simples: visite a árvore e gere o código para cada nó

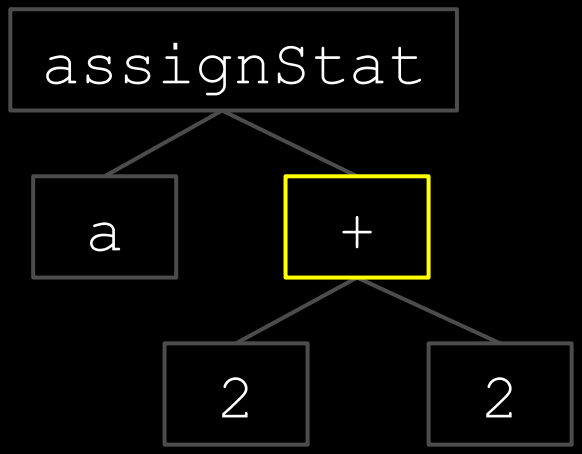
- Ex.: `a = 2 + 2;`



Gerando Código CIL

- Método mais simples: visite a árvore e gere o código para cada nó

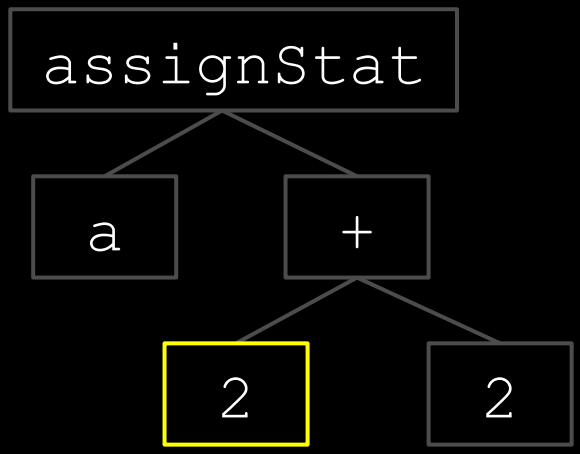
- Ex.: `a = 2 + 2;`




```
...
...
add
stloc.s a
```

Gerando Código CIL

- Método mais simples: visite a árvore e gere o código para cada nó
- Ex.: `a = 2 + 2;`

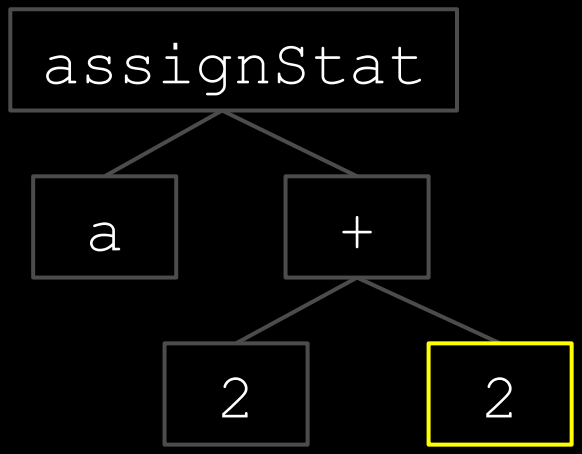



```
ldc.i4 2
…
add
stloc.s a
```

Gerando Código CIL

- Método mais simples: visite a árvore e gere o código para cada nó

- Ex.: `a = 2 + 2;`




```
ldc.i4 2
ldc.i4 2
add
stloc.s a
```

Gerando Código CIL (Exemplo HelloWorld)

```
.assembly HelloWorld {} // Nome do seu arquivo
.assembly extern mscorlib {} // "import" mscorlib

.method static void Main( ){ // Declaração do main
  .entrypoint // Ponto de entrada do programa
  .maxstack 2 // Tamanho máximo da pilha
  ldstr "Hello World!" // Carrega a string na pilha
  call void [mscorlib]System.Console::WriteLine(string)
  ret // Retorno
}
```

Gerando Código CIL (Exemplo DoSomething)

```
.assembly DoSomething {} // Nome do seu arquivo
.assembly extern mscorlib {} // "import" mscorlib

.method static void Main( ){ // Declaração do main
  .entrypoint // Ponto de entrada do programa
  .maxstack 2 // Tamanho máximo da pilha
  .locals init(int32 a) // Definição das variáveis locais
  call int32 temp( ) // Chamada ao método temp
  stloc.s a // Carrega o valor na variável local "a"
  ldloc.s a // Carrega o valor da variável local na pilha
  call void [mscorlib]System.Console::WriteLine(int32)
  call string[mscorlib]System.Console::ReadLine( )
  pop // Tira o primeiro elemento da pilha
  ret // Retorno
}
```

Gerando Código CIL (Exemplo DoSomething)

```
// Definição de um método, int32 é o tipo de retorno
.method public static int32 temp( ) {
  ldc.i4 10
  ret
}
```

Execução

- Confiram as instruções do arquivo README.txt na pasta compilers-cin\aulas-praticas\ap4\ do repositório da disciplina

Exercícios Práticos

1. Modifique o visitor CymbolGenerationVisitor para dar suporte às funcionalidades necessárias para gerar código CIL para o programa contido no arquivo codegen.cym. A implementação do visitor deverá ler o código fonte de um arquivo de entrada do mesmo diretório com o nome “input.cym” e um arquivo de saída, também no mesmo diretório, com o nome “output.il”.

Exercícios Práticos

Observações:

- A função main será usada como ponto de entrada
- A função println imprime um inteiro no console
- O tamanho máximo da pilha deve ser o suficiente para gerar o código CIL do arquivo codegen.cym
- Pode ser necessário renomear o nome de algumas funções para evitar conflitos com as instruções da CIL
- Desconsidere a utilização de variáveis globais
- Considere que todos os arquivos de entrada não apresentarão erros de compilação ou execução

Exercícios Práticos

- O exercício prático deve ser realizado individualmente ou em dupla e enviado por e-mail com o assunto “EXERCÍCIOS PRÁTICOS 04” para monitoria-if688-l@cin.ufpe.br até as 23:59 da quinta-feira (14.12.2017)
- A resolução do exercício prático deve estar em um arquivo comprimido com o nome “Q1.zip” e deve conter os arquivos de código fonte CodeGenerationApplicationVisitor.java e CymbolGenerationVisitor.java