

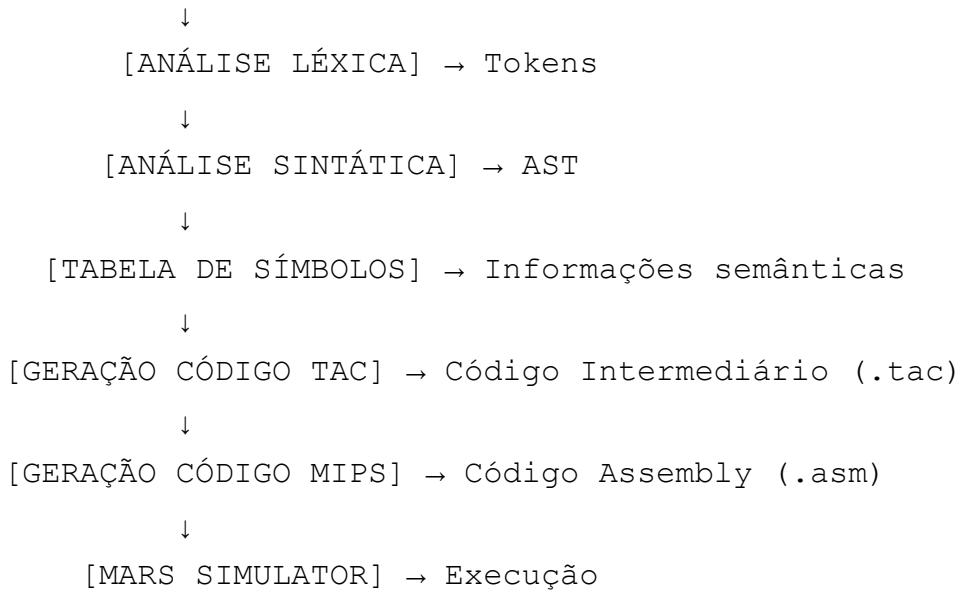
SEGUNDA ENTREGA - TRABALHO DE COMPILADORES

DESCRIÇÃO DO PROJETO

Esta segunda entrega, sendo uma continuação da primeira parte, acrescenta a compilação, adicionando três componentes fundamentais: **tabela de símbolos**, **geração de código intermediário (Three-Address Code - TAC)** e **geração de código MIPS**. O sistema agora é capaz de transformar código fonte Ada em código executável para arquitetura MIPS.

ARQUITETURA DO COMPILADOR

CÓDIGO FONTE ADA (.ada)



COMPONENTES ADICIONADOS

1. TABELA DE SÍMBOLOS (*SymbolTable.hs*)

Sistema completo de gerenciamento de símbolos com suporte a escopos aninhados:

- **Estruturas de Dados:**

- *SymbolCategory* : Classifica símbolos como Variável, Função ou Tipo

- `DataType` : Suporte a tipos primitivos (inteiro, float, booleano, caractere) e compostos (arrays, registros)
- `SymbolInfo` : Armazena categoria, tipo, constante e valor de cada símbolo
- `ScopeStack` : Pilha de tabelas de símbolos para escopos aninhados

- **Funcionalidades:**

- Inserção e busca de símbolos com resolução de escopo
- Declaração de variáveis, funções e tipos
- Verificação de tipos simples
- Suporte a constantes com valores associados

2. CÓDIGO DE TRÊS ENDEREÇOS (`TAC.hs` e `TACGenerator.hs`)

Representação intermediária simplificada para facilitar a geração de código final:

- **Instruções Suportadas:**

- `x = y op z` → Operações aritméticas e lógicas
- `ifz x goto L` → Desvio condicional se zero
- `ifnz x goto L` → Desvio condicional se não-zero
- `goto L` → Desvio incondicional
- `param x` → Passagem de parâmetros
- `call f, n` → Chamada de função com n parâmetros
- `return x` → Retorno de função
- `label L`: → Definição de rótulo

- **Expressões TAC:**

- `TACVar` : Variáveis do programa
- `TACTemp` : Variáveis temporárias geradas
- `TACInt` : Literais inteiros
- `TACStr` : Literais string
- `TACBool` : Literais booleanos

- **Tradução de Construções Ada:**

- Atribuições(`:=`)
- Estruturas condicionais(`if-then-else`)
- Laços(`while`)
- Chamadas de função/procedimento
- Operadores aritméticos e lógicos

3. GERADOR DE CÓDIGO MIPS (`MIPSGenerator.hs`)

Conversão do código TAC para assembly MIPS 32-bit:

- **Mapeamento de Registradores:**

- Variáveis do programa → Registradores salvos (`$s0-$s7`)
- Variáveis temporárias → Registradores temporários (`$t0-$t9`)
- Argumentos → `$a0-$a3`
- Retorno → `$v0-$v1`

- **Seções Geradas:**

- `.data` : Strings constantes, buffers de entrada
- `.text` : Código executável com rótulo `main`

- **Tradução de Instruções Especiais:**

- `Put_Line` → Chamada de sistema para impressão (`syscall 4`)
- `Get_Line` → Chamada de sistema para leitura (`syscall 8`)
- Operações aritméticas → Instruções MIPS correspondentes (add, sub, mul, div)
- Desvios condicionais → beq, bne, beqz, bnez

SAÍDAS GERADAS

O compilador produz dois arquivos para cada entrada:

1. `<nome>.tac` - Código de Três Endereços (intermediário)

- Formato legível para depuração
- Pode ser usado para otimizações futuras

2. `<nome>.asm` - Código Assembly MIPS

- Pronto para execução no MARS
- Inclui todas as diretivas necessárias

DEPURAÇÃO

Para auxiliar na depuração, o compilador exibe passo a passo do código:

1. Tokens reconhecidos pelo lexer
2. Árvore sintática abstrata (AST) gerada
3. Código TAC intermediário

4. Código MIPS final