Celestial Suite: uma Ferramenta para a Tradução e Execução de Código de Três Endereços

Guilherme Galante
Ciência da Computação
Universidade Estadual do Oeste do Paraná - Unioeste
Cascavel, Paraná, Brasil
guilherme.galante@unioeste.br

Daniel Carlos Chaves Boll
Ciência da Computação
Universidade Estadual do Oeste do Paraná - Unioeste
Cascavel, Paraná, Brasil
danielboll.academico@gmail.com

***Resumo*—O artigo descreve o desenvolvimento e a avaliação de uma ferramenta didática chamada Celestial Suite, destinada ao ensino de compiladores. A ferramenta realiza a conversão de programas descritos em código de três endereços para assembly MIPS, além de permitir a emulação e sua execução por meio de uma máquina virtual. A avaliação feita pelos alunos indicou que a ferramenta é eficaz como recurso didático, mas sugeriu melhorias, como a precisão das mensagens de erro, a implementação de dicas no editor e melhor documentação.**

***Index Terms*—Compiladores, Código de três endereços, MIPS.**

I. Introdução

Compiladores são ferramentas essenciais, pois constituem uma peça central na infraestrutura de construção de software. Praticamente todos os programas que executam em um computador foram processados por um compilador ou uma ferramenta similar [1]. Dada essa importância, a disciplina sobre a construção de compiladores é componente fundamental de um curso de ciência da computação, conforme pode-se encontrar em currículos propostos, por exemplo, pela Sociedade Brasileira da Computação (SBC)[1] e pela *Joint Task Force on Computer Science Curricula* - ACM/IEEE[2].

O conteúdo destas disciplinas incluem teoria de linguagens formais, teoria dos autômatos, análise léxica, análise sintática, análise semântica, geração de código intermediário, geração de código e otimização. O campo dos compiladores está ainda estreitamente ligado a outras disciplinas, incluindo arquitetura e organização de computadores, linguagens de programação, teoria de linguagens formais e autômatos, teoria da computação, engenharia de software e sistemas operacionais.

Nesse contexto se insere a disciplina de Compiladores do curso de Ciência da Computação da Unioeste, campus Cascavel-PR. Integrada ao quarto ano do curso, possui uma carga horária de 102 horas/aula e aborda os conceitos fundamentais mencionados anteriormente. As avaliações incluem provas teóricas e um projeto prático de construção de um compilador. Neste projeto, os alunos desenvolvem uma linguagem de programação, implementando os analisadores léxicos, sintáticos e semânticos, culminando na criação de um gerador de código intermediário, especificamente em código de três endereços (*Three Address Code* - TAC) [2], [3].

Esta representação intermediária, embora útil didaticamente, apresenta uma limitação prática fundamental: não existe uma maneira direta de convertê-la em código executável (*back-end*) utilizando ferramentas acessíveis. Assim, neste trabalho apresenta-se um conjunto de ferramentas web que permitem a tradução do código de três endereços para linguagem de montagem MIPS e posterior execução. Ao possibilitar a conversão de código de três endereços para a linguagem de montagem, a Celestial Suite oferece aos estudantes uma experiência mais integrada e completa para a disciplina de Compiladores em nosso curso, na qual o aluno inicia com a concepção da linguagem de programação e consegue visualizar todas as etapas até a execução do código.

O restante do trabalho está organizado da seguinte forma: A Seção II descreve alguns trabalhos relacionados. A Seção III apresenta os conceitos básicos necessários para a compreensão do estudo. A Seção IV justifica o desenvolvimento do trabalho. Na Seção V, a ferramenta proposta e seus componentes são detalhados. A Seção VI relata a avaliação da ferramenta pelos alunos da disciplina de Compiladores. Por fim, a Seção VII conclui o trabalho e sugere algumas possibilidades para pesquisas futuras.

II. Trabalhos relacionados

Devido ao grande número de algoritmos e às construções teóricas complexas, o ensino de compiladores representa um domínio desafiador tanto para os professores quanto para os alunos. Nesse contexto, os softwares educacionais desempenham um papel cada vez mais importante, auxiliando os alunos na aprendizagem ao transformar conceitos teóricos abstratos em objetos tangíveis com os quais eles podem interagir. A seguir, alguns softwares usados no ensino de compiladores são apresentados cronologicamente.

Mernik e Zumer [4] desenvolveram a ferramenta LISA, um sistema que gera automaticamente um compilador a partir de especificações formais de linguagens baseadas em gramáticas de atributos. Ao observar a execução do LISA, o acadêmico pode visualizar o processo de compilação e obter uma compreensão intuitiva da execução do compilador. Para cada fase

[1]https://www.sbc.org.br/documentos-da-sbc/summary/131-curriculos-de-referencia/760-curriculo-de-referencia-cc-ec-versao2005

[2]https://dl.acm.org/doi/pdf/10.1145/3664191

(léxica, sintática e semântica), uma animação é oferecida para aprimorar o modelo cognitivo do espectador.

Sheshat [5] é uma ferramenta baseada na web para ensino de conceitos relacionados à análise léxica, incluindo linguagens regulares, expressões regulares e autômatos finitos. Com foco na análise sintática, Sangal et al. [6] descrevem a ferramenta PAVT. O PAVT pode ser usado para visualizar como conjuntos *First* e *Follow*, conjuntos de itens, tabela de análise, árvore de análise e derivação mais à esquerda ou mais à direita, dependendo do algoritmo sendo analisado.

Stamenković e Jovanović [7] apresentam o ComVis, um sistema Web bastante completo para a visualização e simulação de conceitos e algoritmos básicos de compiladores. Nele é possível visualizar as fases de análise léxica, sintática e semântica, bem como a geração de representações intermediárias e código em linguagem de montagem.

Muñoz et al. [8] descreve a implementação de uma aplicação de avaliação automática voltada à disciplina de compiladores, construída como um plug-in de um sistema de avaliação (SIETTE) que é capaz de gerar automaticamente uma gramática livre de contexto (GLC) aleatória, determinar se ela é adequada para análise LL(1) e/ou SLR(1), construir as tabelas de análise passo a passo e avaliar as respostas dos alunos.

Outras ferramentas são analisadas por Stamenković et al. [9]. É possível notar que as diversas ferramentas apresentadas possuem características e funcionalidades distintas, mas complementares entre si, podendo ser utilizadas para cobrir o conteúdo de uma disciplina de compiladores quase que por completo. No entanto, a ferramenta proposta neste trabalho se diferencia por sua singularidade na literatura, uma vez que não existem ferramentas que, a partir da representação intermediária de três endereços, gerem código para MIPS e permitam sua execução.

III. Conceitos Básicos

Nesta seção aborda-se os conceitos básicos necessários para a compreensão do trabalho.

A. Fases de um Compilador

O processo de compilação é muito complexo, sendo estruturado em duas grandes fases conhecidas como análise e síntese. A fase de análise, também chamada de *front-end*, divide o programa fonte em partes e impõe uma estrutura gramatical sobre elas. Uma das principais responsabilidades dessa fase é garantir que a sintaxe e a semântica do programa fonte estejam corretas. A fase de síntese, conhecida como *back-end*, constrói o programa objeto a partir de uma representação intermediária criada na fase de análise.

Se o compilador for cuidadosamente projetado, é possível produzir tradutores para diferentes linguagens fonte e arquiteturas alvo, combinando diferentes estruturas de *front-end* com um único *back-end* ou um único *front-end* com diferentes *back-ends* (Figura 1).

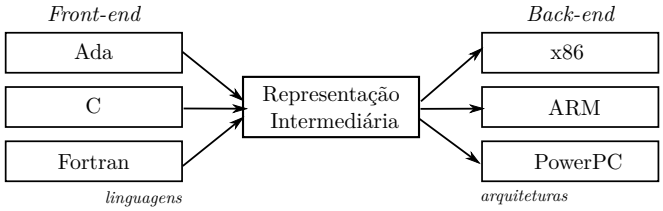

Figura 1. Combinações de diversos Front-ends e Back-ends.

O processo de compilação começa com o analisador léxico, que varre todo o programa fonte e transforma o texto em um fluxo de tokens. Em seguida, a análise sintática lê o fluxo de tokens e valida a estrutura do programa, criando uma árvore sintática. A terceira fase é a análise semântica, responsável por garantir as regras semânticas, o contexto e o inter-relacionamento das partes do programa. Todas essas fases compõem a tarefa de análise.

A próxima fase é a geração de código intermediário, que cria uma abstração do código. Logo após, vem a fase de geração do código objeto, cujo objetivo é gerar o código de baixo nível baseado na arquitetura da máquina alvo. Pode haver ainda uma fase de otimização do código antes das fases de geração de código intermediário e objeto. Essas fases fazem parte da tarefa de síntese. A Figura 2 sintetiza as fases envolvidas na compilação.

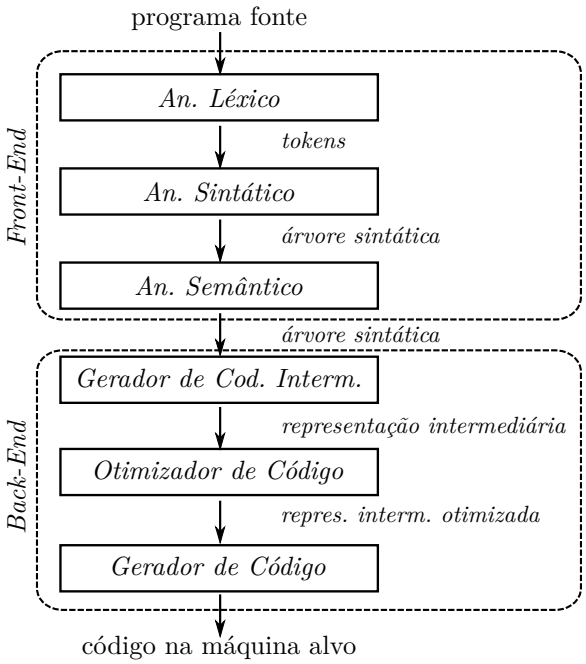

Figura 2. Fases de um compilador.

B. Código de Três Endereços

Na fase de geração de código intermediário, há diversas possibilidades de representação. Dentre elas há o código de três endereços (*Three Address Code* - TAC), comumente utilizado em disciplinas de compiladores por sua simplicidade e fácil aplicação didática. Como visto, essa representação serve como uma ponte entre a análise semântica e a geração de

código, facilitando a tradução de programas de alto nível para código de máquina ou código de um nível mais baixo [3].

Código de três endereços é uma sequência de instruções na forma A = B OP C, onde A, B e C são nomes de variáveis, constantes ou variáveis temporárias geradas pelo compilador. OP é qualquer operação aritmética ou lógica aplicada aos operandos B e C. O nome reflete que há no máximo três variáveis, onde duas são operandos e uma é para o resultado. Em uma instrução de três endereços, apenas um operador é permitido. Caso uma expressão seja composta por múltiplos operadores, ela deve ser dividida em uma sequência de subexpressões [10]. Por exemplo, a expressão $a * -(b + c)$ é convertida no código:

```
t1 = b + c
ts = uminus t1
t3 = a * t2
```

Um outro exemplo, de conversão é dado a seguir, onde o trecho de código de um laço for em C a seguir

```
for(i = 1; i<=10; i++)
{
   a[i] = x * 5;
}
```

é convertido em:

```
i = 1                 //inicialização de i
L1:                   //início do laço
if i > 10 goto L2     //condição do laço
t1 = x * 5            //bloco do laço
a[i] = t1
i = i+1
goto L1
L2:                   //fim do laço
```

Mais detalhes sobra a TAC pode ser encontrada na literatura, por exemplo, nos livros de Aho et al. [2], Torczon et al [3] e Sunita [10]. Neste trabalho, utiliza-se uma versão estendida do código de três endereços, ETAC, descrita na Seção V-A.

C. MIPS

A arquitetura MIPS (*Microprocessor without Interlocked Pipeline Stages*) é uma arquitetura desenvolvida pela *MIPS Computer Systems*. A principal característica da arquitetura MIPS é o uso do modelo load/store para manipulação de dados, onde todas as operações são realizadas em registradores do processador, e o acesso à memória principal é feito exclusivamente por instruções de carga e armazenamento. Pela sua elegância e simplicidade, processadores MIPS são bastante usados em cursos de arquiteturas de computadores de muitas universidades. Ele é considerado um processador bastante didático.

A linguagem de montagem MIPS é conhecida por sua simplicidade e eficiência. Essas características são evidenciadas por sua estrutura de instrução e design. A MIPS adota um modelo de conjunto de instruções reduzido (*Reduced Instruction Set Computer, RISC*), que se concentra em um número menor de instruções frequentemente usadas. Essas instruções são de tamanho fixo e seguem um formato consistente, facilitando a compreensão e implementação.

Mais detalhes da arquitetura MIPS pode ser encontrados na literatura técnica [11].

IV. Justificativa da plataforma: A disciplina de Compiladores

A disciplina de Compiladores do curso de Ciência da Computação da Unioeste é ofertada na quarta série, com carga horária de 102 horas/aula distribuída ao longo de dois semestres[3]. A ementa contempla os seguintes itens: (i) Conceitos básicos sobre compiladores e interpretadores. (ii) Análise Léxica. (iii) Análise Sintática. (iv) Análise Semântica. (v) Geração e Otimização de Código. (vi) Projeto e implementação de um Compilador.

Os itens (i) a (v) são apresentados de modo teórico e são avaliados por meio de provas escritas. Já no item (vi), o objetivo é desenvolver o *front-end* do compilador, a partir de uma linguagem de programação especificada pelos próprios alunos. Na primeira fase do projeto, especifica-se a linguagem de programação e implementa-se o analisador léxico. Na segunda, o aluno desenvolve um analisador sintático, do zero, baseado na abordagem *Top-Down* ou *Bottom-Up*. Na última etapa, implementa-se o analisador semântico e o gerador de código intermediário. Salienta-se que as implementações são realizadas sem o uso de geradores automáticos como Flex e Bison, de modo que o discente compreenda como as técnicas envolvidas funcionam na prática.

Nas últimas ofertas da disciplina, optou-se por utilizar como representação intermediária código de três endereços, considerando a sua simplicidade e facilidade de compreensão e geração. Representações como LLVM são interessantes por já contemplarem um *back-end* completo e funcional, mas a linguagem LLVM-IR é bastante complexa e sua geração não é trivial, o que acabava desencorajando os alunos a desenvolverem a última etapa do trabalho. No entanto, a opção pelo TAC apresenta uma limitação: não há um *back-end* implementado que use essa representação como entrada, e assim, os alunos não conseguiam gerar a execução de seus códigos. Dessa limitação surgiu a ideia de implementar uma plataforma, a Celestial Suite, que contempla um tradutor de código de três endereços para a linguagem de montagem MIPS e uma máquina virtual para a execução esse código.

V. Celestial Suite

A Celestial Suite inspira-se na arte da navegação celestial, sendo composta por componentes nomeados em alusão a instrumentos de navegação ou elementos astronômicos:

- *Compass*: tradutor do código de três endereços para linguagem de montagem MIPS;
- *Astrolabe*: analisadores para o código MIPS;
- *Sextant*: máquina virtual para execução do código MIPS;

[3]Na Unioeste o regime é anual, havendo disciplinas que perduram por um ou dois semestres.

- *Horizon*: interface web da ferramenta;
- *Orion's Belt*: biblioteca de interligação entre as ferramentas.

A ferramenta está disponível publicamente para uso[4]. Os códigos-fonte estão disponíveis no GitHub[5].

A. Extended Three-Address Code - ETAC

O Celestial Suite utiliza uma versão estendida da tradicional representação intermediária de código de três endereços, denominado ETAC (*Extended Three-Address Code*). O ETAC diferencia-se da TAC tradicional principalmente pela inclusão de tipagem forte, operando sobre variáveis com tipos de dados definidos explicitamente.

1) Tipos: O ETAC suporta uma variedade de tipos de dados, como apresentado na Tabela I. Esta diversidade permite simular um ambiente de programação mais próximo de linguagens reais, enfatizando a importância da consistência de tipos para evitar erros comuns em linguagens de baixo nível.

Tabela I
TIPOS DE DADOS SUPORTADOS PELO ETAC

Tipo	Descrição
`i8`	Inteiro de 8 bits
`i16`	Inteiro de 16 bits
`i32`	Inteiro de 32 bits
`f32`	Número de ponto flutuante de 32 bits
`f64`	Número de ponto flutuante de 64 bits
`bool`	Tipo booleano
`ptr`	Ponteiro, contendo um endereço de memória

2) Atribuições e Expressões: As operações de atribuição e expressões no ETAC seguem uma abordagem rigorosa de tipagem, como ilustrado a seguir.

```
t1: i32 = 100
t2: f64 = 3.14
t3: i32 = t1 + t2  # Erro: t2 nao é um i32
t4: f64 = t2 * t2
t6: f64 = (f64) t1 # Conv. de t1 para f64
```

Esta estrutura ajuda os alunos a compreenderem as implicações das conversões de tipo e as operações entre diferentes tipos.

Estas extensões implementadas no ETAC foram projetadas para cobrir aspectos importantes da programação de computadores encontrados em ambientes de desenvolvimento reais. Ao mesmo tempo, mantém-se alinhada com os objetivos pedagógicos da disciplina, garantindo que os conceitos vistos na teoria sejam aplicáveis. Exemplos de uso da ETAC estão disponíveis na documentação do Celestial Hub.

B. Compass e Astrolabe

O tradutor Compass é um componente central no pipeline do projeto. É o componente que processa o código de três endereços estendido (ETAC) e realiza a tokenização e a construção da Árvore de Sintaxe Abstrata (AST) através de análises sintática e semântica.

A linguagem Rust foi escolhida para a implementação do compilador, dada a sua eficiência e robustez em desenvolvimento de compiladores. Para a análise léxica, foi utilizado o Logos[6], que oferece uma interface intuitiva para a definição de tokens com expressões regulares. Para a análise sintática e semântica utilizou-se o LALRPOP[7]. Este gerador de analisador sintático, análogo a ferramentas como YACC e ANTLR, permite definir a gramática do parser com base na sintaxe do Rust. A integração entre o Lexer do Logos e o LALRPOP é realizada adaptando o Lexer para funcionar como um iterador que o LALRPOP pode processar.

O processo de tradução realizada pelo Compass inicia-se com a análise léxica para identificar e reportar erros relacionados a tokens malformados ou desconhecidos. Na sequência, executa a análise sintática, que constrói uma Árvore de Sintaxe Abstrata (AST) do código de entrada, seguida por uma análise semântica que assegura a corretude dos tipos e declarações de variáveis. Após a análise bem-sucedida, o Compass procede para a geração de código, onde a AST é traduzida para um subconjunto de instruções específico de assembly MIPS.

O código MIPS gerado é então enviado para o Astrolabe, que após as análises léxica e sintática, gera uma AST que é usada posteriormente pela máquina virtual Sextant. Assim como no Compass, o Astrolabe emprega o Logos e LALRPOP para a sua construção.

C. Sextant

O Sextant funciona como uma máquina virtual (MV) que interpreta e executa a partir da AST MIPS. Esta MV permite uma configuração personalizada, permitindo aos usuários ajustar a quantidade de memória e o tamanho da pilha. Essa flexibilidade ajuda a adaptar a MV a uma variedade de cenários de uso, variando de ambientes educacionais a aplicações com diferentes exigências.

Uma vez inicializada, a Sextant procede com a avaliação da AST MIPS, alocando memória conforme necessário e processando as instruções MIPS. Importante destacar que o conjunto de instruções implementado na Sextant é uma versão restrita do assembly MIPS completo. A máquina virtual mantém os 32 registradores originais da arquitetura MIPS, porém não suporta todas as instruções MIPS, especialmente aquelas relacionadas a operações com números em ponto flutuante. A representação de números negativos é feita utilizando o método de complemento de dois.

O sistema de E/S da Sextant é modular, permitindo fácil integração com diferentes interfaces de usuário e mecanismos de entrada e saída. Em ambientes web, por exemplo, a MV pode interagir com elementos do *Document Object Model* (DOM) ou produzir alertas, demonstrando assim sua adaptabilidade e versatilidade.

[4]https://horizon.unioeste.br/
[5]https://github.com/celestial-hub
[6]https://crates.io/crates/logos-codegen
[7]http://lalrpop.github.io/lalrpop/

D. Horizon e Orion's Belt

O Horizon é a interface do usuário (*User Interface*, UI) do projeto, desenvolvida como uma plataforma web para maximizar acessibilidade e disponibilidade. A utilização de uma interface baseada na web elimina a necessidade de instalação de software adicional, facilitando o acesso em diversas plataformas. A plataforma, construída com Next.JS, incorpora módulos-chave, como a aba de transpilação, onde os usuários inserem código de três endereços e recebem o código MIPS correspondente, e a aba da máquina virtual, que permite a carga e visualização do estado da máquina durante a execução.

A tela de conversão de código de três endereços para assembly MIPS pode ser visualizada na Figura 3. Esta tela contém duas caixas de texto: à esquerda, o usuário pode inserir ou selecionar códigos ETAC pré-definidos através de uma caixa de seleção acima do editor. Após a inserção do código, o usuário pode converter o código ETAC para MIPS clicando no botão *Transpile*. O código MIPS resultante é exibido no editor à direita.

A segunda tela exibe a máquina virtual (MV), onde o código MIPS pode ser carregado utilizando o botão *Load to VM*, localizado no canto superior direito do editor de código MIPS. A interface da MV mostra os estados visíveis, incluindo registradores, memória e a instrução atual, além de saída de resultados e controles da VM conforme ilustrado na Figura 4.

A documentação da ferramenta está disponível em https://horizon.unioeste.br/docs, a qual aborda a especificação do ETAC e o uso das ferramentas desenvolvidas. A documentação inclui tutoriais, exemplos práticos e uma descrição completa das funcionalidades de cada componente, visando auxiliar os usuários na compreensão e utilização eficaz das ferramentas.

O Orion's Belt integra os componentes desenvolvidos em Rust com a interface de usuário em JavaScript. Esta integração é realizada através do Napi RS[8], que facilita a comunicação entre módulos escritos em diferentes linguagens, conectando-se à API nativa do NodeJS. Tal abordagem é estratégica para permitir a interação fluida entre as distintas tecnologias empregadas no projeto.

Esta configuração do Horizon e do Orion's Belt garante uma interface do usuário funcional e acessível, promovendo uma experiência de usuário fluida e eficiente, enquanto oferece uma ponte robusta entre Rust e JavaScript.

E. Funcionamento da Ferramenta

Basicamente, o uso da ferramenta é composto por dois passos: (1) conversão do ETAC para assembly MIPS e (2) emulação da execução do código MIPS.

A conversão do ETAC para MIPS é apresentado no diagrama de sequência apresentado na Figura 5. O processo inicia com o usuário inserindo o código de três endereços no Horizon e submetendo-o para tradução. Na sequência, o Horizon solicita ao Compass, via Orion's Belt, a tradução do código. O Compass executa o parsing e a análise do código de três endereços, convertendo-o em código MIPS. Este código MIPS é então retornado ao Horizon e exibido para o usuário.

No segundo caso, o usuário solicita a execução do código MIPS na máquina virtual (MV) integrada. Aqui, o Astrolabe executa uma análise sintática e semântica do código MIPS e retorna uma Árvore de Sintaxe Abstrata (AST) para o Sextant. O Sextant, por sua vez, prepara as instruções de código de máquina correspondentes e envia o estado da MV e as instruções de volta ao Horizon, que as exibe para o usuário, permitindo que visualize as instruções executadas e o estado atual da MV.

VI. Avaliação dos Alunos

Ao final da disciplina de compiladores do ano letivo de 2023, foi solicitado aos alunos que avaliassem a ferramenta desenvolvida. O objetivo principal foi avaliar a eficácia da ferramenta Celestial como um recurso didático e identificar possíveis melhorias. Antes da pesquisa, foi feita uma apresentação detalhada do contexto do projeto, a relevância do que havia sido aprendido na disciplina até aquele momento, e uma introdução ao funcionamento da ferramenta, incluindo demonstrações de seu uso através de exemplos práticos. Após a explanação foi dado um período de 30 minutos para que os alunos pudessem explorar a ferramenta.

Na sequência, um formulário foi submetido aos alunos, o qual 9 alunos responderam. As respostas obtidas foram usadas entender a percepção dos alunos sobre a especificação do ETAC, a funcionalidade de visualização de execução do código, a eficácia geral da ferramenta e sugestões de melhoria.

Todos os alunos concordaram que a especificação do ETAC estava alinhada com o conteúdo visto em aula e que a visualização de execução do código era satisfatória. Quanto à avaliação do Celestial como uma ferramenta didática, obteve-se uma média geral de 4,5 em 5. Isso indica uma recepção positiva da ferramenta em termos de sua integração educacional e usabilidade, refletindo seu potencial como um recurso valioso no ensino de compiladores.

Quanto às melhorias para a ferramenta, destacam-se:

- Mensagens de Erro: A dificuldade na interpretação das mensagens de erro foi destacada por cinco alunos. Melhorar a precisão na indicação de erros no código foi uma sugestão comum, visando facilitar a identificação e correção de problemas;
- Funcionalidades Adicionais: Foi sugerida a implementação de dicas no editor e de um recurso de autocomplete para funções, para acelerar o aprendizado e a usabilidade da ferramenta;
- Documentação: A necessidade de clareza e consistência na documentação também foi enfatizada, com recomendações para que ela detalhe mais profundamente as funcionalidades e especificações da linguagem.

[8]https://napi.rs/pt-BR

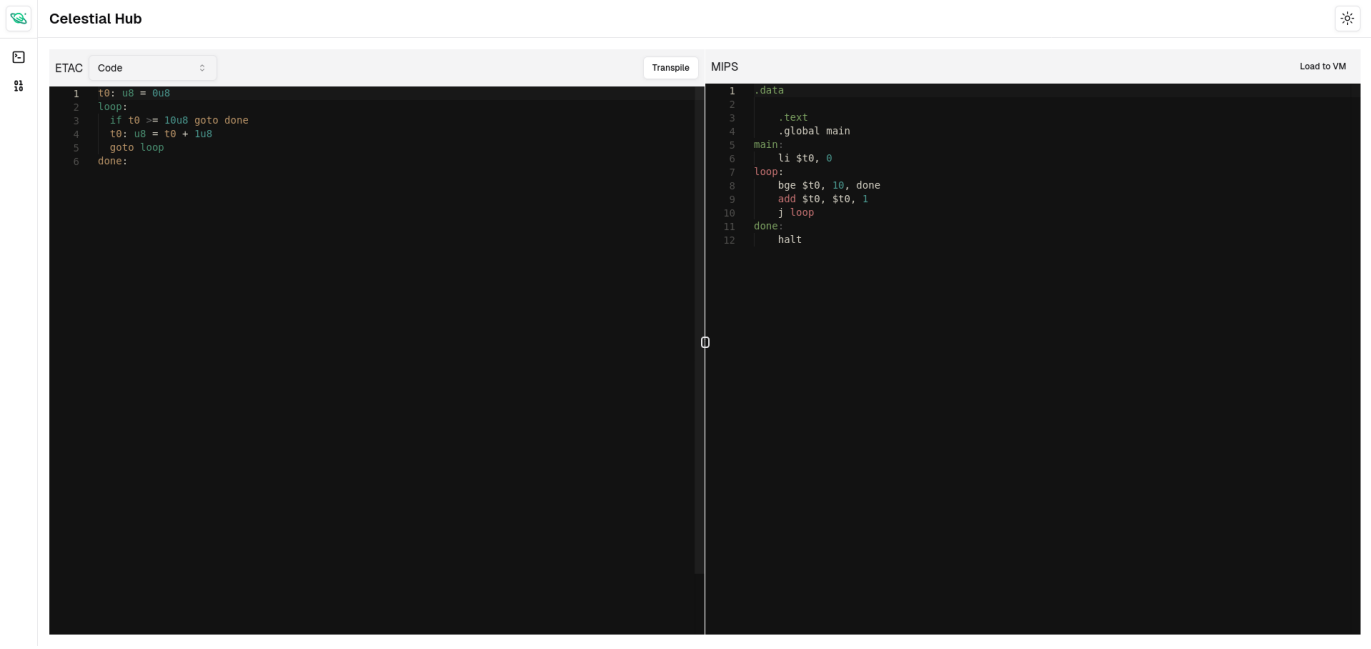

Figura 3. Horizon: Conversão de ETAC para assembly MIPS.

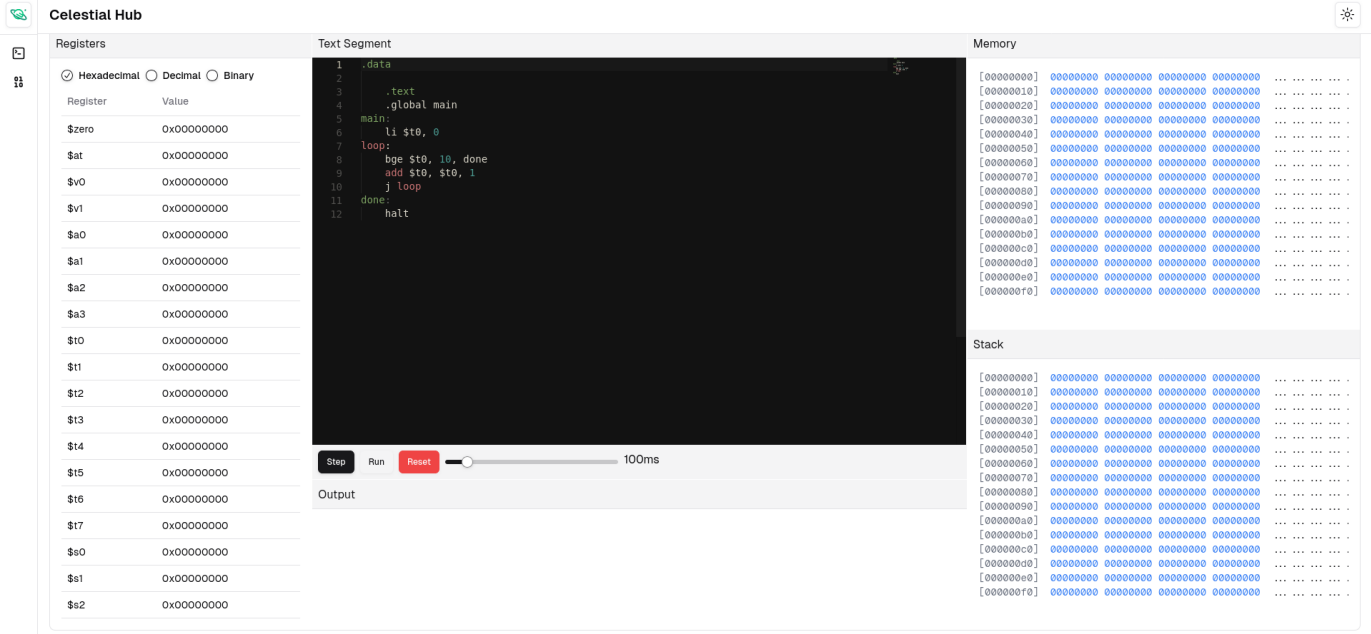

Figura 4. Horizon: Execução do código MIPS.

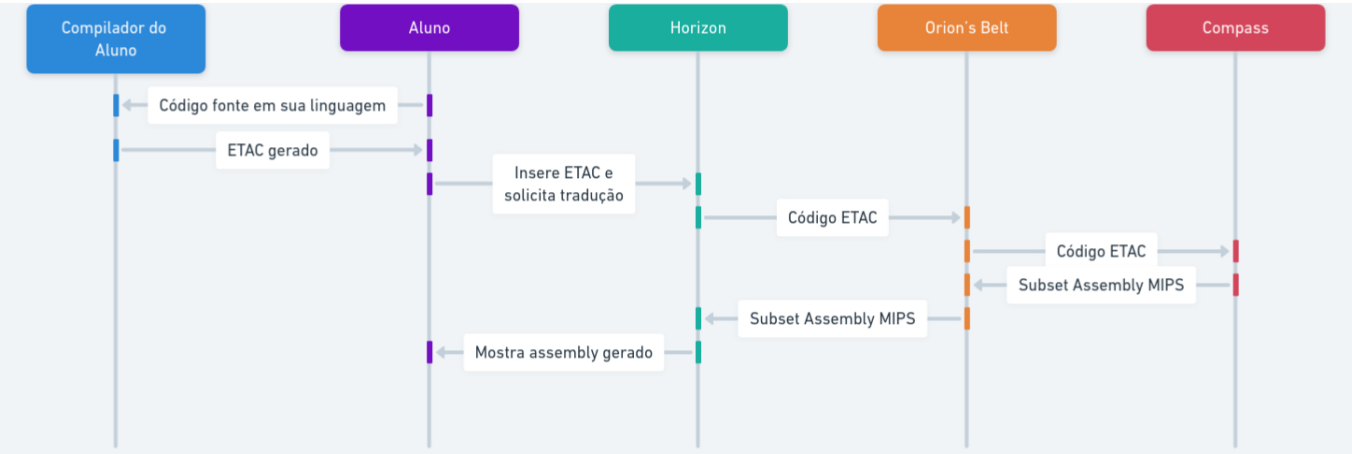

Figura 5. Diagrama do processo de tradução.

VII. Considerações Finais

As ferramentas desenvolvidas neste trabalho atendem à demanda da disciplina de Compiladores da Unioeste, permitindo a execução de programas em código de três endereços gerados pelos alunos. Dessa forma, após o desenvolvimento completo do *front-end* e do gerador de representação intermediária (etapas iniciais da avaliação), o aluno pode verificar como um código para uma arquitetura alvo é gerado e executado, permitindo visualizar todo o processo de tradução.

Embora o conjunto de ferramentas tenha sido bem avaliado pelos alunos, foram identificados pontos de melhoria que serão implementados nas próximas etapas do trabalho.

Destacam-se também algumas funcionalidades que podem ser agregadas à Celestial Suite para melhorar a experiência de uso:

1) Expansão de Arquiteturas Suportadas: ampliar a Celestial Suite para permitir a tradução do conjunto de instruções completo do MIPS e de outras arquiteturas diferentes;
2) Integração com outras ferramentas e ambientes de aprendizado: integrar a Celestial Suite com outras ferramentas relacionadas a outras etapas de compilação e plataformas de aprendizado de modo a proporcionar uma experiência de aprendizado mais abrangente;
3) Pesquisa sobre Métodos de Ensino: Investigar e testar diferentes métodos pedagógicos usando a Celestial Suite para determinar quais abordagens são mais eficazes no ensino de compiladores.

VIII. Agradecimentos

Agradecimentos especiais aos alunos que contribuíram com os comentários e pareceres sobre a ferramenta.

Referências

[1] J. Chen, J. Patra, M. Pradel, Y. Xiong, H. Zhang, D. Hao, and L. Zhang, "A survey of compiler testing," vol. 53, no. 1, feb 2020. [Online]. Available: https://doi.org/10.1145/3363562

[2] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Addison Wesley, August 2006.

[3] L. Torczon and K. Cooper, *Engineering A Compiler*, 2nd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007.

[4] M. Mernik and V. Zumer, "An educational tool for teaching compiler construction," *IEEE Transactions on Education*, vol. 46, no. 1, p. 61–68, Feb. 2003. [Online]. Available: http://dx.doi.org/10.1109/TE.2002.808277

[5] A. Arnaiz-González, J.-F. Díez-Pastor, I. Ramos-Pérez, and C. García-Osorio, "Seshat - a web-based educational resource for teaching the most common algorithms of lexical analysis," *Computer Applications in Engineering Education*, vol. 26, no. 6, p. 2255–2265, Jul. 2018. [Online]. Available: http://dx.doi.org/10.1002/cae.22036

[6] S. Sangal, S. Kataria, T. Tyagi, N. Gupta, Y. Kirtani, S. Agrawal, and P. Chakraborty, "Pavt: a tool to visualize and teach parsing algorithms," *Education and Information Technologies*, vol. 23, no. 6, p. 2737–2764, May 2018. [Online]. Available: http://dx.doi.org/10.1007/s10639-018-9739-x

[7] S. Stamenković and N. Jovanović, "A web-based educational system for teaching compilers," *IEEE Transactions on Learning Technologies*, vol. 17, pp. 143–156, 2024.

[8] R. C. Muñoz, B. B. Blanco, J. d. Campo-Ávila, and J. L. T. Rodriguez, "Teaching compilers: Automatic question generation and intelligent assessment of grammars' parsing," *IEEE Transactions on Learning Technologies*, vol. 17, pp. 1734–1744, 2024.

[9] S. Stamenković, N. Jovanović, and P. Chakraborty, "Evaluation of simulation systems suitable for teaching compiler construction courses," *Computer Applications in Engineering Education*, vol. 28, no. 3, p. 606–625, Mar. 2020. [Online]. Available: http://dx.doi.org/10.1002/cae.22231

[10] K. Sunitha, *Compiler Construction*, ser. Always learning. Pearson Education India, 2013.

[11] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Sixth Edition: A Quantitative Approach*, 6th ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2017.