O ensino de *Assembly* na construção de um pensamento crítico em programação de computadores

Pedro Henrique Magalhães Botelho[1], **Roberto Cabral Rabêlo Filho**[1]

[1]Universidade Federal do Ceará (UFC) - Campus de Quixadá
Av. José de Freitas Queiroz, 5003 -– Cedro —– Quixadá – CE – Brazil 63902-580

pedrobotelho15@alu.ufc.br, rbcabral@ufc.br

Abstract. *This article deals with the impact that Assembly teaching has on the construction of critical thinking in programming languages. The article's main objective is to deal with the importance of knowing the computer in which one is programmed and propose a teaching model for Assembly that addresses the issues about the student's immersion in the low-level abstraction knowledge of his computer.*

Resumo. *Este artigo trata do impacto que o ensino de Assembly tem sobre a construção de um pensamento crítico na utilização de linguagens de programação. O objetivo central do artigo é tratar da importância de se entender o funcionamento do computador em que se programa, e propor um modelo de ensino de Assembly que aborde os assuntos pertinentes à imersão do estudante no conhecimento de baixo nível de abstração de seu computador.*

1. Introdução

A utilização de computadores já faz parte do nosso cotidiano a muito tempo. Ouvir músicas, assistir filmes, comunicar-nos com amigos e familiares, estudar, etc. Tudo isso foi facilitado com o advento dos computadores e da *internet*. Isso se deve à todas as pesquisas, experimentos e estudos realizados na área da eletrônica. A labuta de estudar e desenvolver aplicações para os computadores, então, tem um valor mais do que nobre, buscando sempre usufruir ao máximo dos recursos da máquina, e, com diferentes propósitos trouxeram muitas inovações, atingindo assim os mais diversos públicos.

Sendo o computador uma máquina eletrônica é essencial, para operá-la, conhecer seu funcionamento a nível digital, sua organização e sua arquitetura. Conhecendo todo o funcionamento intrínseco do computador o programador poderá aproveitar melhor os recursos do *hardware*. Os processadores trabalham com um conjunto de instruções, em formato binário, o qual é legível para a máquina mas não pelos seres humanos. Então, em 1949, foi criada a **linguagem *Assembly***, legível para os seres humanos, para operar o processador. Saber operar o *Assembly* leva o programador a um entendimento profundo da arquitetura do processador, tendo capacidade para escrever programas mais otimizados, mesmo sem precisar programar nele diretamente. Tem-se, então, a necessidade de conhecer a fundo a máquina para poder desenvolver aplicações fidedignas.

Este artigo está dividido em 5 seções, e, nas próximas seções, será dissertado sobre a importância de aprender a linguagem *Assembly*, bem como um modelo de ensino baseado em pesquisas, utilizando o *Assembly* como uma ferramenta de aprendizagem de arquitetura de computadores. Também serão mostrados trabalhos que se relacionam com este, trazendo à tona outras utilidades de se aprender *Assembly*.

2. Trabalhos Relacionados

Como dito antes, os processadores operam por meio de instruções, que podem ser mapeadas de forma legível em uma linguagem de programação, o ***Assembly***.

Em [de Andrade 2014] é mostrado como a linguagem *Assembly* foi a primeira forma de se programar um processador de uma maneira mais dinâmica do que apenas números binários ou vias físicas, como eram as linguagens de programação da primeira geração, sendo então a linguagem *Assembly* considerada a primeira linguagem da segunda geração, além de falar de sua utilidade em sistemas embarcados.

Em [Vidal 2017] é dissertado sobre as principais aplicações do *Assembly* na programação e na engenharia. Uma das principais aplicações, segundo o autor, é no entendimento da operação da CPU, no desenvolvimento de sistemas embarcados e sistemas operacionais de tempo real, áreas de enfoque do engenheiro de computação.

No primeiro capítulo de [Jorgensen 2020] é debatido sobre as vantagens de se conhecer a linguagem *Assembly*, como entender melhor a arquitetura do processador, assim como chamada de funções, interrupções e operações de I/O, entender melhor as ferramentas rotineiramente utilizadas, como o compilador e melhorar as técnicas de programação.

Outros trabalhos como [Silva 2021] e [Faquer 2017] tratam da importância da programação em *Assembly* em áreas mais específicas: engenharia reversa (além de exploração de binários, otimização de código e programação do *MS-DOS*) e segurança da informação (como análise de *malware*), respectivamente.

3. Fundamentação Teórica

O trabalho apresentado é uma dissertação a respeito de como o aprendizado de *Assembly* pode proporcionar melhorias na carreira de um programador.

Todo programador deve entender como funciona a plataforma para onde seu programa é destinado, podendo, então, tirar proveito de seus recursos. Dessa forma, o programador pode construir programas mais otimizados, utilizando melhor os recursos disponíveis. O estudo do funcionamento básico de um computador se divide em duas partes:

Organização de Computadores: se refere aos elementos que compõem um computador e suas interconexões, como por exemplo: memória principal, memória *cache*, memória externa, dispositivos de entrada e saída e unidade central de processamento.

Arquitetura de Computadores: se refere aos elementos que tem impacto direto sobre a execução de programas, como por exemplo: conjunto de instruções, *pipeline*, linguagem de montagem, técnicas de entrada e saída e tamanho dos barramentos.

Sendo os computadores máquinas digitais devem ser programados com valores binários, representando as instruções da CPU, dados e endereços de memória, onde cada arquitetura tem seu próprio conjunto de instruções. Nesse contexto surge a linguagem *Assembly* como uma forma de dinamizar a programação de computadores, com instruções representadas por mnemônicos e endereços representados por etiquetas, tornando o código legível. Esse código é transformado em código de máquina binário por um *software* chamado de ***Assembler***. Ao estudar *Assembly* também se aprende a arquitetura do computador, melhorarando as habilidades de programação, já que se deve programar de maneira pouco convencional, tendo que pensar mais. Existem coisas que só

podem ser feitas em *Assembly*, como inspeção em arquivos binários, engenharia reversa e interrupções de *software*, sendo possível ter acesso total aos recursos do *hardware*.

4. Procedimentos Metodológicos

Como dito anteriormente, o objetivo deste artigo é explicitar como o conhecimento em linguagem de montagem impulsiona as habilidades do programador, além de estabelecer um modelo de ensino.

O fato de saber programar em *Assembly* ajudar na programação em alto nível, mesmo sem escrever uma linha de código em *Assembly*, levanta algumas dúvidas. Por isso, esse artigo foi separado em 3 métricas, como mostrado a seguir:

4.1. A linguagem Assembly e a Arquitetura de Computadores

O estudo de uma arquitetura específica de processadores, ou seja, uma linha de processadores com características semelhantes, depende diretamente do aprendizado de sua linguagem de montagem, já que dita as capacidades do processador, bem como suas especificidades mais íntimas, como mencionado antes.

Esse artigo trata de como é possível, a partir do aprendizado de *Assembly*, conhecer melhor a arquitetura de seu processador e, a partir disso, desenvolver um pensamento crítico na programação de computadores, podendo programar de maneira mais otimizada, mesmo sem utilizar códigos em *Assembly*, mas sim as habilidades e conhecimentos desenvolvidos no aprendizado.

4.2. A Engenharia de Computação

O engenheiro de computação é o profissional capacitado a desenvolver o projeto de sistemas embarcados. Isso inclui desenvolver seu *firmware*, desenvolver um sistema elétrico permitindo sua interação com o mundo externo, e, é claro, conhecer seu *hardware* para que haja uma perfeita integração entre o *software* e o *hardware*.

Entende-se logo que o engenheiro deve conhecer o *hardware* do sistema embarcado a fundo para que possa desenvolver um programa com maestria. Tendo em vista que a maioria dos sistemas embarcados operam em torno de um microcontrolador, um pequeno computador em um circuito integrado, pode-se, então, utilizar a linguagem *Assembly* para entender o funcionamento do microcontrolador, já que ela dita quais são as operações que o microcontrolador consegue realizar.

4.3. Modelo de Ensino

Sendo um programador ou um engenheiro, entender a construção do computador em que se trabalha é essencial para o desenvolvimento do projeto. Um engenheiro, por exemplo, tem que saber escolher um *hardware* mais barato o possível, mas que atenda às necessidades, e, ao mesmo tempo, saber extrair 100% do processamento do mesmo. Então, tem-se a necessidade de estudar a arquitetura do computador alvo. É necessário entender toda a estrutura do sistema, seja para o projeto de *hardware* ou para a programação de um *firmware*.

Sendo o *Assembly* uma ferramenta que descreve as capacidades de um processador, programar em *Assembly* é então a melhor forma de se aprender como é o funcionamento de uma família de processadores. Devemos então, modelar um plano de ensino de

Assembly, de forma a abordar os conteúdos mais importantes no que diz respeito a arquitetura de um processador, levando o programador ou engenheiro a um grau de compreensão elevado, podendo executar projetos com maestria.

5. Resultados

Foi realizado uma pesquisa com os alunos do terceiro semestre de Engenharia de Computação na Universidade Federal do Ceará, ao final da disciplina de "Arquitetura e Organização de Computadores I", onde é tratado da arquitetura x86-32, com perguntas sobre a utilidade de *Assembly* na vida do programador e engenheiro.

Nessa pesquisa 75% dos alunos veem necessidade de utilizar *Assembly* em alguma aplicação, além de acharem viável um *firmware* embarcado escrito em C, em conjunto com *Assembly*. Os conteúdos votados como mais relevantes para o assunto foram: pilha de execução, interrupções, procedimentos, funcionamento do compilador e biblioteca externa escrita em *Assembly*. Ainda, os alunos relataram terem obtido uma visão mais ampla, podendo otimizar o código para consumir menos ciclos de *clock* e menos memória. Além disso, relataram conseguir especificar melhor sistemas embarcados, buscando o melhor desempenho baseado em sua arquitetura.

É possível ver, então, que os alunos perceberam mudança na sua forma de programar, e, após um semestre programando em *Assembly*, puderam entender melhor o funcionamento da arquitetura, podendo escrever códigos em alto nível mais otimizados. Isso se deve ao fato de que, ao programar em *Assembly*, pensa-se mais, para criar um programa otimizado, e depois, ao programar em C, é possível escrever códigos mais limpos e rápidos, mesmo que no programa nenhuma linha de *Assembly* tenha sido escrita.

Para chegar a um conhecimento razoável em *Assembly* deve-se conhecer primeiro as bases da arquitetura onde se trabalha, para depois entender a linguagem. Após estudar as bases o aluno pode ir para conteúdos mais avançados. Abaixo segue um modelo de um curso de *Assembly* para a arquitetura x86:

1. **Introdução**: Deve-se definir o que é a linguagem *Assembly*, explicitando o *background* histórico da linguagem, o que são os utilitários, como o *assembler* e o compilador funcionam e quais as vantagens de programar em *Assembly*.
2. **Noções de Organização de Computadores**: O aluno precisa ter base de organização de computadores para poder entender uma arquitetura, sabendo o que são memórias, processadores, barramentos, dispositivos de entrada e saída, além de entender como funciona um sistema digital.
3. **A Arquitetura x86**: O aluno deve entender as bases da arquitetura em questão, como o conjunto de instruções, *pipeline*, registradores, organização de memória.
4. **Movimentação de Dados**: O aluno tem que entender primeiramente como funcionam as instruções que operam a memória, como funciona o endereçamento e segmentação de memória, para que possa avançar no estudo de instruções.
5. **Aritmética e *Flags***: Depois, o aluno deve entender como a CPU realiza operações aritméticas, uma das bases para a programação em *Assembly*.
6. **Controle de Fluxo**: Deve-se entender como são implementadas estrutura condicionais em *Assembly*, por meio de instruções de comparação e saltos condicionais.
7. **Procedimentos**: Após as bases, o aluno deve aprender como implementar funções em *Assembly*, entendendo como funciona a pilha de execução e o escopo de variáveis, assim como criar bibliotecas externas.

8. **Interface de Alto nível**: Deve-se também como interfacear o código *Assembly* com um código de alto nível, como C, utilizando o melhor dos dois mundos.
9. **Interrupções**: O aluno deve ter capacidade de utilizar interrupções de *software*, compreendendo as chamadas de sistema de um sistema operacional **Linux**, além de compreender interrupções de *hardware* e exceções. Deve entender também como funciona a interface de entrada e saída, especialmente em embarcados.
10. **Conjunto de Instruções Estendido**: O aluno deve ter bases em instruções que operam valores em ponto flutuante, como a FPU x87, e instruções vetoriais, como SSE e AVX, entendendo as dificuldades envolvendo esses quesitos.
11. **Programação *Bare Metal***: Como forma de aplicar os conhecimentos o aluno deverá desenvolver um *firmware*, onde, sem interferência do Sistema Operacional, deverá utilizar interrupções de *software* para interação com os dispositivos externos. Dessa forma o aluno irá conseguir ter um controle de *hardware* melhor, podendo compreender melhor como a arquitetura funciona e a importância do SO.

Ao final o aluno terá um entendimento melhor sobre a arquitetura, podendo escrever códigos em alto nível mais otimizados, estando mais atento a detalhes de baixo nível, obtendo assim, uma visão crítica sobre sua arquitetura e sobre como programar para ela de maneira melhor. O *Assembly* tem aplicações além do aprendizado, então dependendo da necessidade será necessário ou não realizar uma comunicação direta com o processador.

6. Conclusão

Após todo o trabalho, ficou claro que o estudo ressaltou as vantagens em saber utilizar *Assembly*, seja para desenvolver sistemas, seja como uma ferramenta de aprendizado. Vimos que, diferente da crença popular, a linguagem *Assembly* não está morta! Como uma linguagem que descreve operações do processador estaria morta? O *toolchain*, no processo de transformar um código C em um binário executável, passa por um passo intermediário, a compilação, onde o código é transformado em *Assembly*, por exemplo. Usamos então o *Assembly* como uma forma de entender como o computador se comporta. A área de estudo das arquiteturas de computadores cresce a cada dia. Cada vez mais é necessário conhecer nossas máquinas, que a cada dia evoluem mais.

De maneira geral, podemos dizer que a linguagem *Assembly* deveria ser aprendida por todo programador, afinal, conhecer a plataforma onde seu programa vai rodar é fundamental. O *Assembly* tem aplicações além do aprendizado, e é essencial para programadores em linguagem C, por exemplo, para poder verificar arquivos executáveis por possíveis erros e conseguir utilizar melhor o *hardware*, por meio de bibliotecas externas. Como o estrategista Sun Tzu disse em "Arte da Guerra", "Conhecer o campo de batalha é antecipar o movimento do inimigo. Conhecer o inimigo é antecipar a vitória".

Referências

de Andrade, E. (2014). História da computação: Um pouco de assembly.

Faquer, C. (2017). Importância da linguagem de programação de baixo nível.

Jorgensen, E. (2020). *x86-64 Assembly Language Programming with Ubuntu*.

Silva, L. F. (2021). *Aprendendo Assembly*.

Vidal, V. (2017). Vale a pena aprender o bom e velho assembly?