

**FUNDAÇÃO EDSON QUEIROZ**

**UNIVERSIDADE DE FORTALEZA – UNIFOR**

**CENTRO DE CIÊNCIAS TECNOLÓGICAS – CCT**

**CURSO ENGENHARIA DE COMPUTAÇÃO**

**Projeto de Pesquisa**

**TCC**

PROPOSTA DE ARQUITETURA DE ASSEMBLER ONLINE ÚNICO PARA DIVERSOS PROCESSADORES

Judah Holanda Correia Lima

Matr. 1010500-5

Maio - 2015

JUDAH HOLANDA CORREIA LIMA

PROPOSTA DE ARQUITETURA DE ASSEMBLER ONLINE ÚNICO PARA DIVERSOS PROCESSADORES

Projeto de pesquisa a ser apresentado e submetido à avaliação para elaboração de Trabalho de Conclusão de Curso – TCC do Curso de Engenharia de Computação do Centro de Ciências Tecnológicas da Universidade de Fortaleza.

Orientador: Danilo Reis Vasconcelos

Fortaleza – Ceará 2015

# **RESUMO**

Montadores *Assembly online* têm alta *performance,* e vêm com uma quantidade de ferramentas de auxílio comparáveis aos *softwares* instalados em disco rígido, para o mesmo propósito. Embora existam vários *Assemblers online*, nenhum deles utiliza o mesmo conjunto de instruções para todos os processadores aos quais ele dá suporte. O presente estudo explora o potencial de uma arquitetura de um montador *Assembly* multiplataforma. Sendo um estudo em progresso de uma proposta de arquitetura, bem como o produto em si, em sua versão inicial e simplificada. Para atingir tal objetivo, a ideia proposta foi utilizar um reduzido conjunto de instruções, comum a inicialmente dois processadores, mas suficientemente grande para serem feitas todas as operações básicas. Os testes demonstraram que, mesmo em estágio inicial, a arquitetura parece ser possível. Funcionando rapidamente mesmo sendo em um servidor simples rodando em PHP. Então, se pode constatar que este projeto tem uma arquitetura possivelmente viável, com um potencial de evolução e desempenho aceitável para os parâmetros do mercado.

**Palavras-chave**: Montadores *Assembly online*. *Softwares.* Estudo de uma arquitetura.

SUMÁRIO

RESUMO 2

1 INTRODUÇÃO 6

2 TEMA 8

2.1 Delimitação do Tema 9

3 OBJETO 10

3.1 Problema 10

3.2 Hipótese 10

4 OBJETIVOS 11

4.1 Geral 11

4.2 Específicos 11

5 JUSTIFICATIVA 12

6 DISCUSSÃO TEÓRICA 13

6.1 Editor 14

6.2 Interpretador 16

6.3 Tradutor 18

6.4 Compilador 19

6.5 Assembler (Montador) 21

6.6 Linker (Ligador/Link-Editor) 21

7 Arquitetura Relacionadas 22

7.1 Java 22

7.1.1 Java Virtual Machine (JVM) 25

7.1.2 Java Bytecode 25

7.2 .NET 27

7.2.2 Common Language Infrastructure (CLI) 29

7.2.3 Common Intermediate Language (CIL) 29

8 Arquitetura 29

8.1 Editor 30

8.2 Tradutor 30

8.3 Montador 31

9 Estudo de Caso 31

8.1 User Interface 31

8.1.1 Single Page 31

8.1.2 Cache/Storage manipulation 34

8.1.3 Ausência de Imagens 34

8.1.4 Sistema de tradução 37

8.1.5 Editor 37

8.1.6 Autenticação/Login 38

8.1.7 CRUD 40

8.2 Core 41

8.2.1 Tradutor 41

8.2.2 Compilador 47

10 Análise dos resultados 47

11 Conclusão 49

12 ANEXO 49

12.1 Processadores 49

12.1.1 8051 50

12.1.2 Z80 50

REFERÊNCIAS 51

# **1 INTRODUÇÃO**

Atualmente, é possível perceber que *softwares,* que eram programas que rodam em ambiente *off-line,* estão se tornando serviços *web* [1,2]. Isto possibilita o uso de diversas plataformas em qualquer lugar que possua conexão com a *internet* e facilita trabalhos colaborativos [3,4,5].

Para a criação de um projeto de *software*/*firmware* comercial normalmente é necessária a instalação de diversos programas, dentre eles IDE’s, programas de documentação, gerenciamento, controle de versão, entre outros. Cada um deles é direcionado para um sistema operacional específico e, algumas vezes, é necessário utilizar dois sistemas distintos, pois um ou mais programas não estão disponíveis para a plataforma utilizada no início do projeto, bem como o tempo gasto para instalação. Depois de toda esta maratona ainda existe a pesquisa de *frameworks* e códigos para serem reutilizados neste projeto, pois não existe um local em que todos os códigos do mundo estão armazenados e de fácil acesso. Finalmente, há a integração destes códigos com o projeto; isso implica em fazer tanto a integração com a IDE quanto com a plataforma utilizada. O que, algumas vezes, provoca a modificação do código o qual se deseja integrar, ou encapsulá-lo em outro que faça a compatibilidade entre os dois e isso implica em ler um código que, geralmente, não está seguindo o mesmo padrão de codificação e documentação, isto quando segue algum ou tem documentação.

Uma IDE *online* além de tornar desnecessária a instalação, permite o uso em qualquer lugar em que a *internet* esteja disponível, independente de plataforma ou sistema operacional. E esse é um dos motivos pelos quais os serviços *Web* têm alcançado tamanho sucesso e aumentado tanto ao longo dos anos, programas que antes precisavam ser instalados na máquina, agora, se tornam *online*, sendo necessário apenas um *browser*. Tornando-se desnecessário se preocupar até mesmo com a atualização de *software*.

A integração de todas essas ferramentas e códigos em um mesmo serviço torna a criação de projetos muito mais rápida e prática, além de aumentar significativamente o reuso de código. E como o código e o editor está na nuvem facilita a colaboração, pois a edição e compartilhamento do código podem ser feitos em qualquer lugar que possua internet, juntamente com um dispositivo que possua um browser e possibilidade de edição de texto.

Contudo, mesmo fazendo tudo isso, cada família de processador tem sua própria linguagem de máquina e, consequentemente, um *Assembly* próprio, o que dificulta muito a migração de uma plataforma para outra [6].

Este estudo propõe uma nova abordagem, criando um *Assembly* unificado que utilize apenas um conjunto restrito de instruções, facilitando assim a migração de plataformas.

Ainda assim existe outro problema a ser resolvido, pois ao se aprender algo novo, normalmente, o mais difícil de assimilar é a terminologia. Então, mesmo que a ideia fundamental seja simples, quando é descrita com palavras não familiares pode dificultar bastante um assunto [7]. Como é o caso de uma linguagem nova, e no caso do Assembly esse é um problema bem comum, principalmente com seus mnemônicos.

Como o *Assembly* utiliza mnemônicos, isso demanda um estudo e aprendizado de cada instrução, tornando a linguagem bem mais complexa e de alto nível, como C/C++ e Java, cujas funções são escritas de forma extensa, diferentemente de uma abreviação/sigla, como no caso dos mnemônicos. Por isso, usualmente se aprende primeiro linguagens de alto nível, pois elas estão mais próximas da linguagem humana [6,7]. Então se propõe que as instruções sejam escritas de forma extensa, facilitando a leitura do código e o aprendizado da linguagem.

Como em vários países existe o hábito de se programar na língua oficial deles, cria-se outro problema, impede-se que pessoas que não falem a língua não consigam utilizar o código escrito em tal língua. Então, para contornar este problema, o sistema terá como regra a codificação em inglês, devido a esta ser uma das línguas mais faladas no mundo, uma das mais fáceis de se aprender e por ter se tornado o padrão na maioria dos projetos.

Ainda assim é necessário delimitar um padrão para nomenclatura de variáveis e métodos, pois mesmo tendo uma linguagem padrão isso não garante que os códigos serão de fácil leitura. Então, um padrão de nomenclatura também deve ser adotado, bem como de chaveamento e padrões de projetos.

Este projeto tem como objetivo solucionar estes problemas ou pelo menos reduzi-los sendo então um editor de código online, bem como armazenando estes códigos. O tipo de código a ser editado será Assemblies específicos de cada processador e bem um Assembly genérico que possa ser traduzido para tais Assemblies. No caso os processadores escolhidos serão o microcontrolador 8051 e o microprocessador z80.

# 2 TEMA

Proposta de uma Arquitetura de *Assembler Online*.

## 2.1 Delimitação do Tema

Utilizar um conjunto de instruções ainda mais reduzido que o RISC convencional, o qual contenha todas as operações básicas que qualquer microprocessador possua, permitindo assim um conjunto de instruções multiplataforma.

Criar um conjunto de instruções fáceis de serem lidas.

Inicialmente o projeto funcionará de forma básica para apenas as arquiteturas dos processadores Intel 8051 e Zilog z80.Big ou little endian

O montador do Assembly em si não ficará no escopo do projeto sendo usado apenas com linha de comando no Linux do pacote SDCC.

# 3 OBJETO

## 3.1 Problema

É possível utilizar/criar uma arquitetura de Assembly Único para vários processadores de arquitetura diferente, gerando resultados em tempo viável?

## 3.2 Hipótese

É possível notar que muitos processadores têm instruções semelhantes e, algumas vezes, até mesmo, iguais. Se fosse possível encontrar um conjunto de instruções comum a todos eles, e estas funções, mesmo que reduzidas, pudessem executar todos os comandos básicos, seria possível criar um *Assembly* comum. Ainda que a linguagem de máquina resultante no final do processo só pudesse ser lida por determinado processador.

A velocidade de processamento em linguagem PHP por um Servidor em nuvem é suficientemente rápida para o Montador proposto.

A arquitetura reduzida RISC proposta é suficientemente rápida para boa parte das aplicações atuais, de tal forma que a perda de *performance,* devido à ausência de outras instruções, é justificada pela sua flexibilidade.

# 4 OBJETIVOS

## 4.1 Geral

Potencializar o reuso de códigos com a criação de um *Assembly* unificado.

## 4.2 Específicos

1. Criar conjunto de instruções;
2. Determinar padrão de nomenclatura a ser seguido pelos seus usuários;
3. Utilizar armazenamento no próprio sistema; e
4. Criar sistema básico (*Easembly*).

# 5 JUSTIFICATIVA

Mesmo já existindo linguagens multiplataforma em alto nível, não é possível utilizá-las em processadores com pouco poder de processamento, como vários microcontroladores de 8 bits.

Mnemônicos dificultam a leitura e entendimento dos códigos.

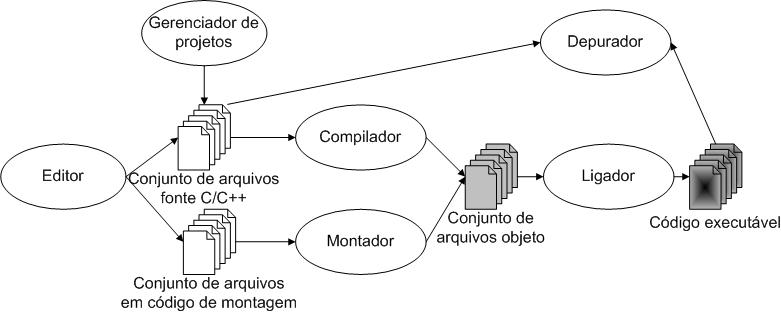
Conjuntos de instruções diferentes para cada família de processadores dificultam a migração de um processador para outro.

A utilização de uma plataforma *web* permite ao programador codificar, em qualquer lugar que possua *internet,* uma infinidade de dispositivos, sem a necessidade de instalar diversas ferramentas.

# 6 DISCUSSÃO TEÓRICA

A ideia deste conjunto reduzido de instruções resulta do sucesso da arquitetura RISC nos processadores modernos. Para tornar este conjunto de instruções multiarquitetura é necessário reduzir o número de instruções ao máximo para que se tenha certeza que este conjunto vai estar presente em qualquer processador.

Foi possível encontrar trabalhos publicados, nos quais a ideia de uma IDE *Online* se repete, embora não com o mesmo intuito de um *Assembler* unificado, mas, muitas vezes, utilizando uma linguagem independente de arquitetura, como Java [8,9,10,11,12,13]. Foram encontrados *Assemblers* para processadores reconfiguráveis [14], embora não seja o foco deste estudo, algumas instruções são mantidas ao reconfigurar o processador e isto será de grande utilidade para projetar o conjunto de instruções, assim como um gerador Assembler [15].



(Exemplo de um diagrama de funcionamento de uma IDE)

O armazenamento de códigos nada mais é do que banco de dados de *Strings* com metadados para gerenciamento destes.

## 6.1 Editor

Um Editor de Código Fonte (imagem1) é um editor de texto (imagem2) desenhado especificamente para editar código fonte de Software/Firmware. O qual pode ser uma aplicação por si só como também pode fazer parte de outra, no caso, uma IDE (Integrated Development Environment) (imagem3) ou um Web Browser. Este tipo de editor é a ferramenta mais fundamental para que um programador possa fazer seu trabalho, o qual é o de escrever e editar código.

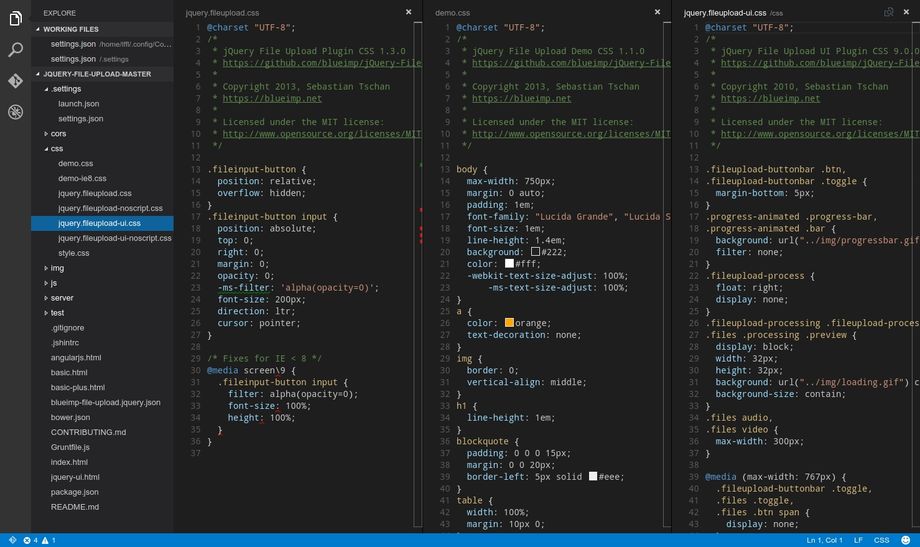


Imagem 1 (exemplo de editor de código, Visual Code)

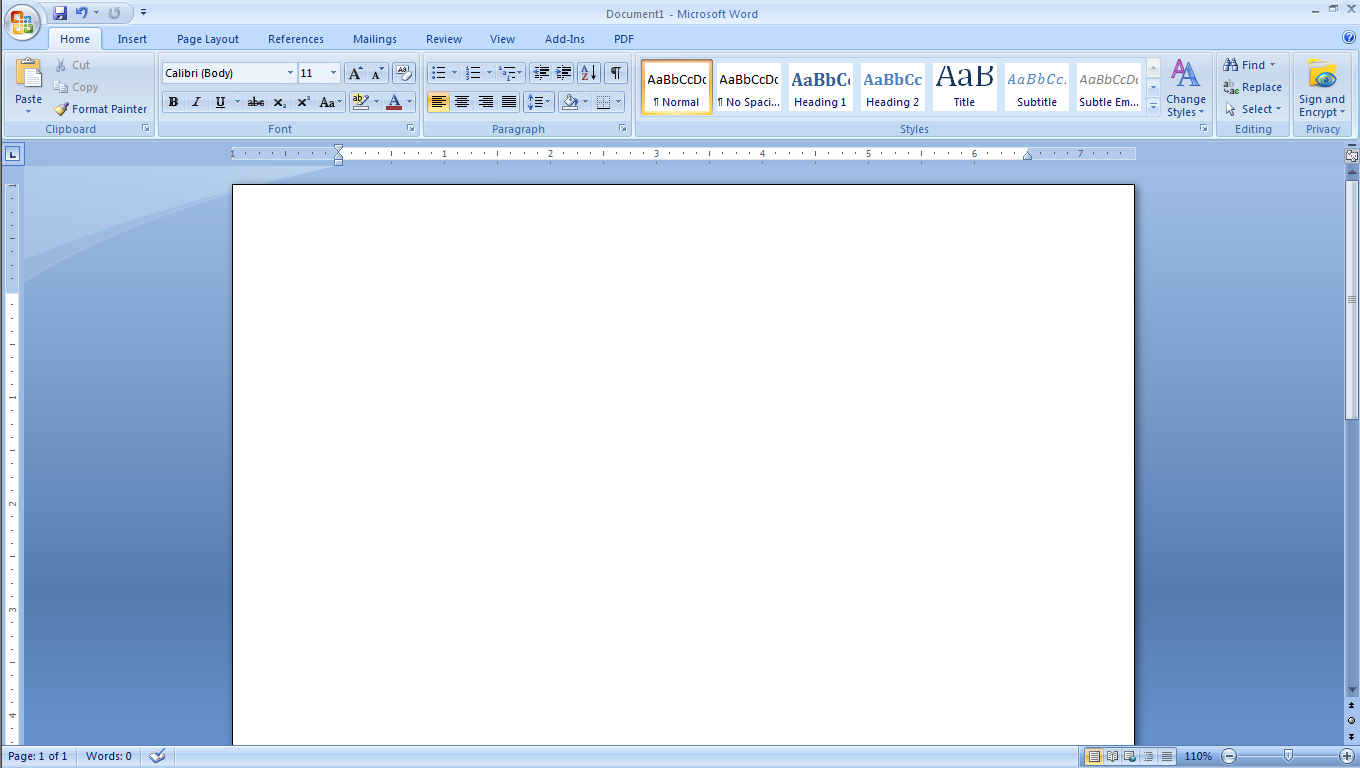
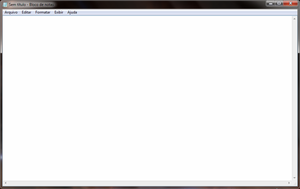


Imagem 2 (exemplo de editor de texto, Microsoft Notepad e exemplo de editor de texto, Microsoft WORD)

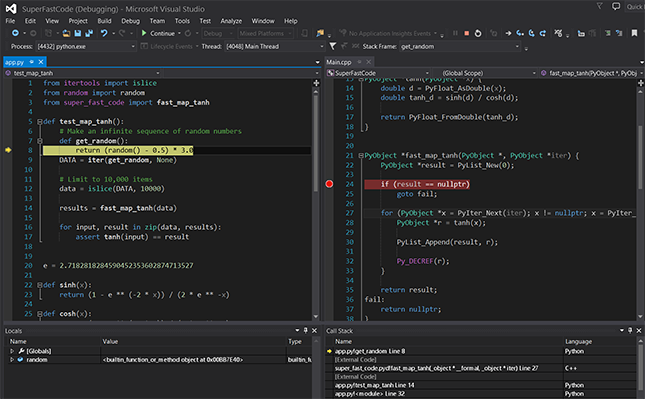
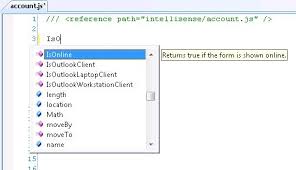


Imagem 3 (exemplo de IDE, Microsoft Visual Studio 2015)

Programas de grandes proporções, possam chegar possuir milhares de linhas de código seriam extremamente difíceis de serem produzidos Editores de código. Tais editores provém uma serie de referências entre todos os elementos do programa para que o programador possa ver e rever como um determinado trecho de código está sendo usado por outro. Além de uma série de ferramentas para acelerar a codificação, como Intelligent Code Completion e geradores de trechos de código.

[ <http://www.pcmag.com/encyclopedia/term/51762/source-code-editor>]

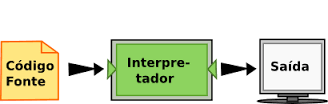
O termo Intelligent code completion que inicialmente popularizado como "picklist" e que até hoje em algumas implementações ainda é referenciado como tal [ <https://developer.salesforce.com/blogs/developer-relations/2008/12/using-the-metadata-api-to-retrieve-picklist-values.html>]. Intelligent code completion [http://code-recommenders.blogspot.com.br/2010/05/its-all-about-intelligent-code.html][http://insenvim.sourceforge.net/] é uma funcionalidade onde a ferramenta completa o código que esta sendo digitado pelo programador, à medida que digita, tendo como base o contexto em que está situado. Esta funcionalidade está presente em vários ambientes de programação [<http://wiki.codeblocks.org/index.php?title=FAQ>] [http://doc.qt.io/qtcreator/creator-completing-code.html], o qual acelera o processo de codificação por reduzir erros de digitação e outros erros comuns, além de eliminar a necessidade de digitar certo termo completamente, afinal o ambiente vai sugerir a opção e o usuário a seleciona e o programa o completa. Normalmente esta funcionalidade é implementada através de popus com sugestões ao digitar (imagem). Intelligent code completion e ferramentas relacionadas servem como documentação e desambiguação para nomes de variáveis, funções e métodos usando reflection. Um exemplo de implementação é o Visual Studio's IntelliSense [https://msdn.microsoft.com/en-us/library/hcw1s69b.aspx].



(exemplo de funcionamento da IntelliSense do Visual Studio)

## 6.2 Interpretador

O interpretador é um programa de computador que executa instruções escritas em uma determinada linguagem de programação. As linguagens PHP, Basic, Prolog, Python, JavaScript e Java, por exemplo, são comumente interpretadas. Um interpretador normalmente usa uma das seguintes estratégias para a execução do programa: executar o código fonte diretamente, como o JavaScript, ou traduzir o código fonte em alguma eficiente representação intermediária e depois executar este código, como por exemplo, o Java que, ao ser compilado, é representado em uma linguagem intermediária, o bytecode, o qual depois é executado por uma máquina virtual.



(fluxo de execução de um código interpretado)

Para tal finalidade, alguns de tradutores transformam uma linguagem fonte em uma linguagem simplificada, chamada de código intermediário, a qual pode ser “executada” diretamente por um programa chamado interpretador. Pode-se imaginar o código intermediário como uma linguagem de máquina de um computador virtual projetado para executar o código fonte, o que de fato ocorre no caso do Java que utiliza a Java Virtual Machine (JVM) para realizar esta tarefa.

Interpretadores são, em geral, menores que compiladores e facilitam a implementação de construções complexas em linguagens de programação, além é claro de possibilitar uma maior flexibilidade, pois Interpretadores podem ser implementados em diferentes tipos de arquiteturas de computadores, conservando a linguagem a qual interpreta. Entretanto, o tempo de execução de um programa interpretado é geralmente maior que o tempo de execução deste mesmo programa compilado, pois o interpretador deve analisar cada instrução do programa a cada vez que é executado e depois executar a ação desejada, enquanto que o código compilado apenas executa a ação dentro de um contexto fixo, anteriormente determinado pela compilação. Este tempo no processo de análise é conhecido como "overhead interpretativa".

[http://producao.virtual.ufpb.br/books/camyle/introducao-a-computacao-livro/livro/livro.chunked/ch05s04.html]

[http://www.inf.ufsc.br/~barreto/cca/arquitet/arq4.htm]

[http://www.inf.pucrs.br/manssour/LinguagemC/PoligC-Cap01.pdf]

[http://www.inf.pucrs.br/~gustavo/disciplinas/pli/material/paradigmas-aula08.pdf]

## 6.3 Tradutor

Um Tradutor de linguagem têm como objetivo, assim como um tradutor de línguas, traduzir de uma linguagem A para uma linguagem B. Um Interpretador pode ser considerado um tradutor, por exemplo, caso ele seja do tipo em que utiliza uma linguagem intermediária, ele precisa fazer uma tradução da linguagem a qual interpreta para a intermediária. Da mesma forma um compilador que traduz da linguagem a partir de qual compila para linguagem de máquina, ou seja o código compilado.

(Fluxo de execução de um programa tradutor de linguagens de programação)

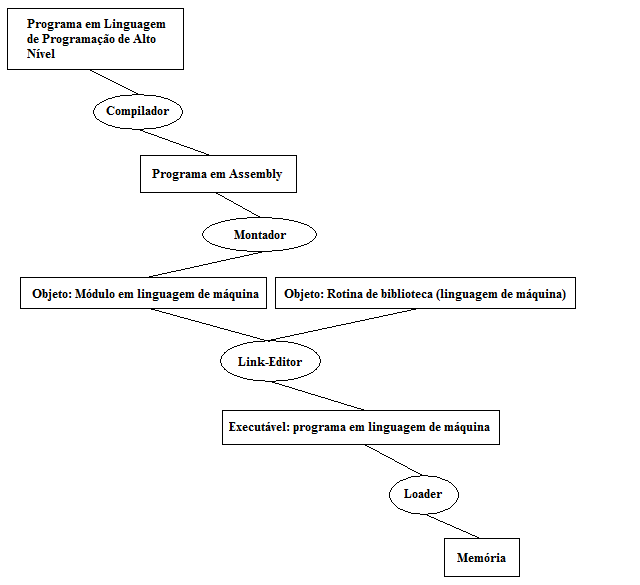
[http://www.inf.ufsc.br/~barreto/cca/arquitet/arq4.htm]

[http://producao.virtual.ufpb.br/books/camyle/introducao-a-computacao-livro/livro/livro.chunked/ch05s03.html]

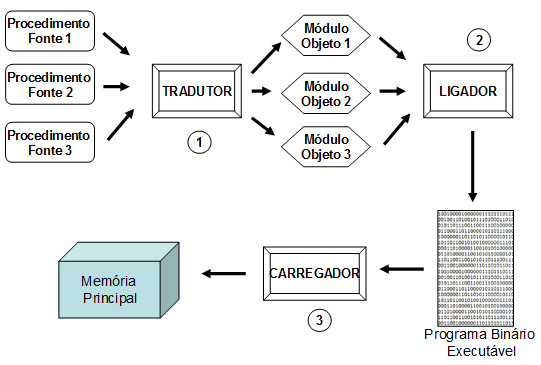
[http://www.inf.pucrs.br/~gustavo/disciplinas/pli/material/paradigmas-aula08.pdf]

## 6.4 Compilador

Um Compilador é aquele que traduz um programa de uma linguagem de programação em um programa assembly que é um conjunto de instruções que serão mapeadas para seu respectivo conjunto de instruções da linguagem de máquina, ou seja, é uma forma simbólica de como a máquina entende (imagem 1 /imagem 2). Linguagens de alto nível, como C, utilizam muito menos código do que o assembly, isso aumenta significativamente a produtividade do programador. Isso ocorre porque o assembly possui um número reduzido de instruções que precisam ser combinadas para realizar o que uma única instrução em código alto nível realiza. Mas isso acarreta em um problema de performance pois muitas vezes o otimizador deixa passar uma solução em assembly mais performática que um programador experiente facilmente iria encontrar.



Arvore de responsabilidades



Fluxo de funcionamento

(Tradutor=Compilador, neste caso) (Ligador=Montador) (Carregador=Loader)

[http://www.devmedia.com.br/processo-de-traducao-e-execucao-de-programas/26872]

[http://producao.virtual.ufpb.br/books/camyle/introducao-a-computacao-livro/livro/livro.chunked/ch05s04.html]

[http://www.inf.ufsc.br/~barreto/cca/arquitet/arq4.htm]

[http://www.inf.pucrs.br/manssour/LinguagemC/PoligC-Cap01.pdf]

[http://www.inf.pucrs.br/~gustavo/disciplinas/pli/material/paradigmas-aula08.pdf]

## 6.5 Assembler (Montador)

O processo de traduzir um programa em linguagem de montagem, ou assembly, para programa em linguagem de máquina é chamado de processo de montagem. Este processo é muito simples, uma vez que existe um mapeamento de um para um de cada comando em linguagem de montagem para seu correspondente em linguagem de máquina. Diferentemente da compilação, onde um comando em linguagem de alto nível pode ser traduzido em vários comandos em linguagem de máquina.

[http://producao.virtual.ufpb.br/books/camyle/introducao-a-computacao-livro/livro/livro.chunked/ch05s04.html]

[http://www.inf.ufsc.br/~barreto/cca/arquitet/arq4.htm]

[http://www.inf.pucrs.br/manssour/LinguagemC/PoligC-Cap01.pdf]

[http://www.inf.pucrs.br/~gustavo/disciplinas/pli/material/paradigmas-aula08.pdf]

## 6.6 Linker (Ligador/Link-Editor)

A função do linker, ou ligador, é coletar procedimentos, já traduzidos em linguagem de máquina separadamente, e ligá-los uns aos outros para que eles possam executar como uma unidade chamada programa binário executável.

Caso o compilador ou o montador lesse um conjunto de procedimentos fonte e produzisse diretamente um programa em linguagem de máquina pronto para ser executado, bastaria que um único comando fonte fosse alterado para que todos os procedimentos fonte tivessem que ser novamente traduzidos. Usando esta técnica, onde os módulos binários estão separados, o único procedimento a ser novamente traduzido seria aquele modificado. Havendo a necessidade de realizar apenas a etapa de ligação dos objetos separados novamente, sendo esta tarefa mais rápida que a de tradução.

[http://producao.virtual.ufpb.br/books/camyle/introducao-a-computacao-livro/livro/livro.chunked/ch05s04.html]

[http://www.inf.ufsc.br/~barreto/cca/arquitet/arq4.htm]

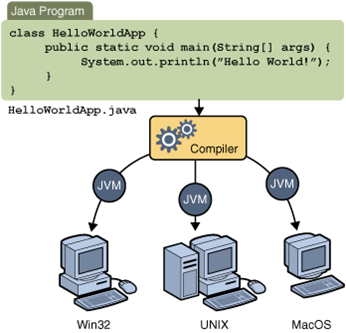
[http://www.inf.pucrs.br/manssour/LinguagemC/PoligC-Cap01.pdf]

[http://www.inf.pucrs.br/~gustavo/disciplinas/pli/material/paradigmas-aula08.pdf]

# 7 Arquitetura Relacionadas

## 7.1 Java

Java é uma linguagem de programação de computadores de propósito geral que é concorrente, orientada a objeto,[Gosling, James; Joy, Bill; Steele, Guy; Bracha, Gilad; Buckley, Alex (2014). The Java® Language Specification (PDF) (Java SE 8 ed.).] e desenhada para ter o mínimo possível de dependências de implementação. Tem como objetivo ser uma linguagem onde uma vez codificada possa ser executada em qualquer plataforma [http://www.computerweekly.com/feature/Write-once-run-anywhere], no caso uma vez que o programa em Java foi compilado ele pode executar em todas as plataformas que usam Java, sem a necessidade de recompilar ou ter de reescrever o código [http://www.oracle.com/technetwork/java/intro-141325.html]. As aplicações Java são geralmente compiladas em um bytecode, o qual pode rodar em qual Java virtual machine (JVM) independente da arquitetura do computador. E no ano 2015, Java continua sendo umas das linguagens de programação mais populares em uso,[ <http://www.wired.com/2013/01/java-no-longer-a-favorite/>] [http://redmonk.com/sogrady/2015/01/14/language-rankings-1-15/] [http://www.langpop.com/] [http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html].

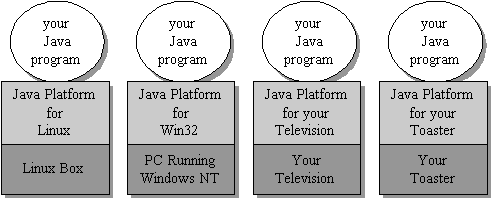


(exemplo de execução de um programa java)

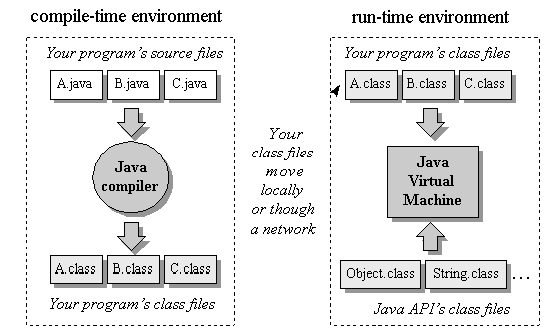
Esta linguagem foi originalmente desenvolvida por James Gosling da Sun Microsystems (a qual foi comprada pela Oracle Corporation) e disponibilizada em 1995 como o componente principal da Sun Microsystems' Java platform. Ela possui muitas semelhanças de sintaxe com C e C++, mas não possui tantas facilidades e ferramentas low-level quanto.

[http://www.oracle.com/technetwork/java/javase/overview/javahistory-index-198355.html]

[Object-oriented Programming with Java: Essentials and Applications, Autor Buyya, Editora Tata McGraw-Hill Education, 2009]



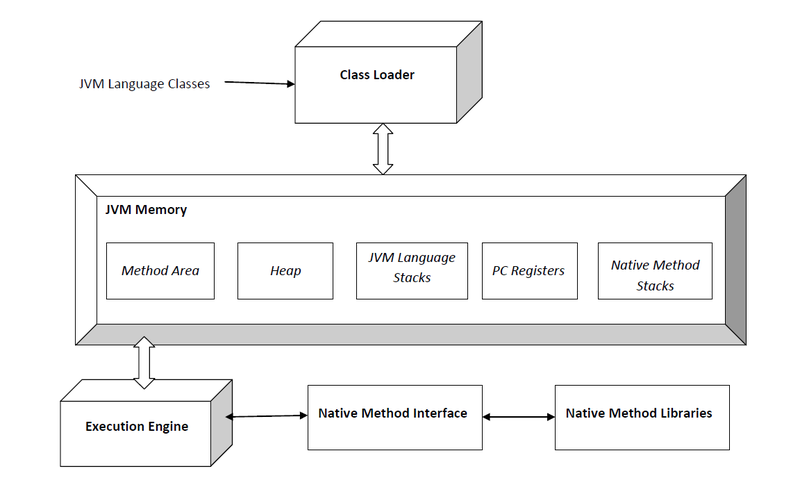
(Funcionamento do Java para cada plataforma)

Ao escrever um programa em Java, significa que o programador expressa o que deseja que seja feito no programa em um código fonte Java, este código será então compilado para arquivos Java class , e posteriormente estes arquivos serão executados na Java virtual machine. [http://www.artima.com/insidejvm/ed2/introarch2.html]

(Fluxo de geração e execução de bytecode)

### 7.1.1 Java Virtual Machine (JVM)

A Java virtual machine (JVM) é uma máquina virtual que permite que um computador execute um programa Java. A JVM pode ser descrita em três níveis, os quais são: especificação, implementação e instância. A especificação é o documento que descreve formalmente o que é requerido para a implementação da JVM. Ao existir apenas uma única especificação, assegura-se que toas as implementações são interoperáveis. A implementação da JVM é uma aplicação de computador que atente a todos os requisitos da especificação da JVM. E finalmente uma instância da JVM, nada mais é que uma implementação rodando num processo que executa um programa de computador compilada em Java bytecode.



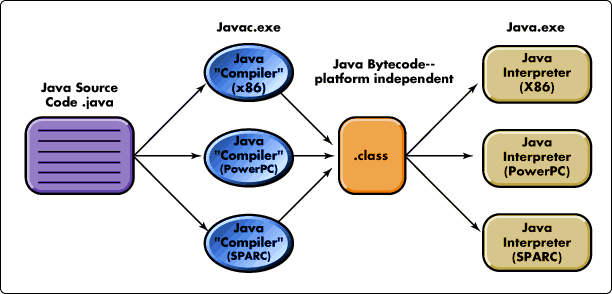
(Diagrama mostrando as partes essenciais da JVM)

[http://www.artima.com/insidejvm/ed2/index.html]

[https://docs.oracle.com/javase/specs/jvms/se7/html/index.html]

### 7.1.2 Java Bytecode

O Java Bytecode é a linguagem de máquina da Java Virtual Machine. Quando a JVM carrega um arquivo .class, ele recebe um stream de bytecode para cada método no .class. Os streams de bytecode são armazenados na área de método da JVM. Os bytecodes do método são executados ao invocar o médodo, o qual acontece durante a execução do programa. Eles podem ser executados por intepretação, compilação just-in-time, ou por qualquer outra maneira desenvolvida pelo designer de uma JVM em particular.



(geração de um Java bytecode)

Um stream de bytecode de um método é a sequência de instruções para a Java Virtual Machine. Cada instrução consiste de um byte de opcode seguido ou não por um ou mais operandos. O opcode indica o comando a ser feito, tal como é feito no assembly. Se mais informação é requerida antes que a JVM possa tomar alguma ação, aquela informação é codificada em um ou mais operandos que, como já discutido, vêm imediatamente após o opcode.

Cada opcode tem o tamanho de um byte, portanto o número de diferentes códigos de operação está limitado a 256. Os 256 possíveis valores para códigos de operação não são todos utilizados. Na verdade, alguns dos códigos foram inclusive reservados para nunca serem implementados. Destes, no ano 2015, 198 estão em uso (~77%), 55 estão reservados para uso futuro e 3 instruções (~1%) estão destinadas a nunca serem implementadas.

[http://www.artima.com/insidejvm/ed2/index.html]

[https://docs.oracle.com/javase/specs/jvms/se7/html/index.html]

[https://docs.oracle.com/javase/specs/jvms/se7/html/jvms-6.html]

[http://www.javaworld.com/article/2077233/core-java/bytecode-basics.html]

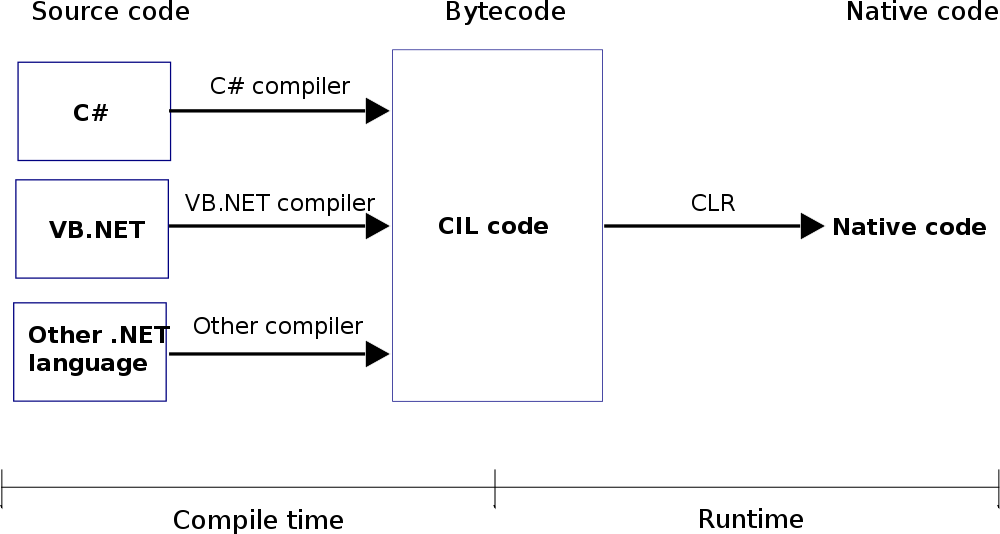
[http://www.ibm.com/developerworks/library/it-haggar\_bytecode/]

## 7.2 .NET

O .NET Framework é projetado para cumprir os seguintes objetivos:

* Proporcionar um ambiente de programação orientada a objeto consistente tanto quando código objeto é armazenado e executado localmente, ou executado localmente, mas distribuído, ou até executados remotamente.
* Proporcionar um ambiente de execução de código que minimiza conflitos de implantação de software e de controle de versão.
* Proporcionar um ambiente de execução de código que promove a execução segura de código, incluindo o código criado por um terceiro desconhecido ou semi-confiável.
* Proporcionar um ambiente de execução de código que elimina os problemas de desempenho ambientes roteirizados ou interpretados.
* Tornar a experiência do desenvolvedor consistente em variados tipos de aplicações, tais como aplicativos baseados no Windows e aplicativos baseados na Web.
* Para construir toda a comunicação em padrões da indústria para garantir que o código baseado no .NET Framework pode se integrar com qualquer outro código. System\_CAPS\_noteNote .

O .NET Framework é composto pelo Common Language Runtime e biblioteca de classes do .NET Framework. O Common Language Runtime é a base do .NET Framework. O tempo de execução pode ser imaginado como um agente que gere código em tempo de execução, fornecimento de serviços essenciais, tais como gerenciamento de memória, gerenciamento de segmentos e comunicação remota, ao mesmo tempo que faz-se cumprir estrita segurança de tipo e outras formas de precisão código que promovam a segurança e robustez. Na verdade, o conceito de gerenciamento de código é um princípio fundamental do tempo de execução. Código que tem como alvo o tempo de execução é conhecido como código gerenciado, enquanto o código que não tem como alvo o tempo de execução é conhecido como código não gerenciado. A biblioteca de classes é uma coleção abrangente, orientada a objetos de tipos reutilizáveis ​​que pode ser utilizada para desenvolver aplicações que vão desde de linha de comando tradicional até interface gráfica do usuário (GUI) para aplicações de pedidos com base nos mais recentes inovações fornecidos pelo ASP.NET, tais como Web Forms e XML Web Services.

[https://msdn.microsoft.com/en-us/library/zw4w595w.aspx] 

(imagem demonstrando o funcionamento da estrutura .NET)

### 7.2.1 Common Language Runtime (CLR)

O .NET Framework fornece um ambiente de tempo de execução chamado Common Language Runtime, que executa o código e fornece serviços que tornam o processo de desenvolvimento mais fácil.

Compiladores e ferramentas expõem a funcionalidade do Common Language Runtime e permitem a escrita de um código que se beneficia desta gestão ambiente de execução. O código gerado para Runtime beneficia-se de funcionalidades, tais como a integração entre linguagens, tratamento de exceções entre linguagens, segurança aprimorada, controle de versões e suporte de implantação, um modelo simplificado para a interação de componentes, e depuração e serviços de perfil.

[https://msdn.microsoft.com/en-us/library/8bs2ecf4.aspx]

### 7.2.2 Common Language Infrastructure (CLI)

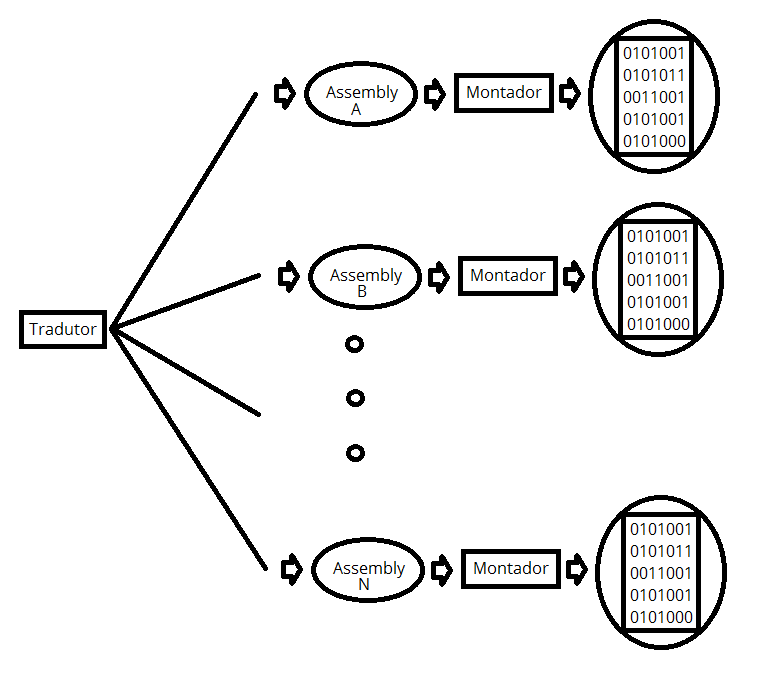
O Common Language Infrastructure (CLI) é uma especificação aberta desenvolvida pela Microsoft e padronizado pela ISO e ECMA que descreve o código executável e um ambiente de tempo de execução que permitem que várias linguagens de alto nível serem usadas em diferentes plataformas de computador sem ser reescrito para arquiteturas específicas. O .NET Framework, open source Mono e o Portable.NET são implementações da CLI.

### 7.2.3 Common Intermediate Language (CIL)

Common Intermediate Language (CIL, pronunciado ou "sil" ou "kil") (anteriormente chamado Microsoft Intermediate Language ou MSIL) é a linguagem de programação do nível mais baixo ainda legível definido pela especificação Common Language Infrastructure (CLI) e é usada pelo .NET Framework e Mono. Linguagens que têm como alvo um ambiente de tempo de execução compatível com CLI compilam para CIL, que é montado em um código de objeto que tem um formato de estilo bytecode. CIL é uma linguagem de montagem orientada a objeto, e é inteiramente baseado em pilha. Sua bytecode é traduzida em código nativo ou - mais comumente - executado por uma máquina virtual.

# 8 Arquitetura

O sistema consiste em um tradutor de assembly, a partir de um genérico, onde pelo qual serão feitas as traduções para o assembly específico de cada arquitetura (imagem). De forma semelhante ao Java Bytecode, mas com a diferença que ao invés de executar numa máquina virtual/ter o assembly interpretado, ele será traduzido para o assembly nativo da máquina onde será executado.



(Fluxo de funciona mento de geração de código de máquina do sistema proposto)

O sistema consiste em 3 níveis: Editor de Assembly Intermediário, Tradutor do Assembly Intermediário para Nativo e Montador Nativo.

## 8.1 Editor

O editor consistirá de um editor de código, onde a linguagem a qual se direciona é o Assembly Intermediário, que no caso, o Easembly. Este editor deve possuir Intelligent code completion para facilitar a implementação do código e fazer uma interpretação básica da linguagem para checagem de erros de sintaxe.

## 8.2 Tradutor

Este tradutor tem como função traduzir o Assembly Intermediário para o Assembly Nativo onde a partir de um arquivo de configurações conseguirá fazer a tradução para uma ou mais linhas da linguagem a qual se deseja fazer a tradução.

## 8.3 Montador

Ao final o código transcrito para o Assembly Nativo deve ser montado para linguagem de máquina relativa ao Assembly transcrito.

# 9 Estudo de Caso

A arquitetura do sistema consiste em um servidor utilizando PHP, tanto para realizar o processamento do core do sistema como para organizar e estruturar as páginas com o HTML5, e javaScript no lado do cliente, tanto para comunicar com o servidor, como para realizar a parte mais complexa do sistema o Editor de Código, incluindo também o roteamento das “páginas” (este foi colocado entre parêntesis, pois a aplicação é single page) e tratamento de cache para acelerar o processamento/carregamento da página. E requererá que seus usuários sigam o padrão de nomenclatura descrito pelo projeto “Pattern” o Object Oriented Programming Languages Pattern.[ https://github.com/Judahh/Pattern/blob/master/General/Object%20Oriented%20Programming%20Languages%20Pattern.mediawiki]

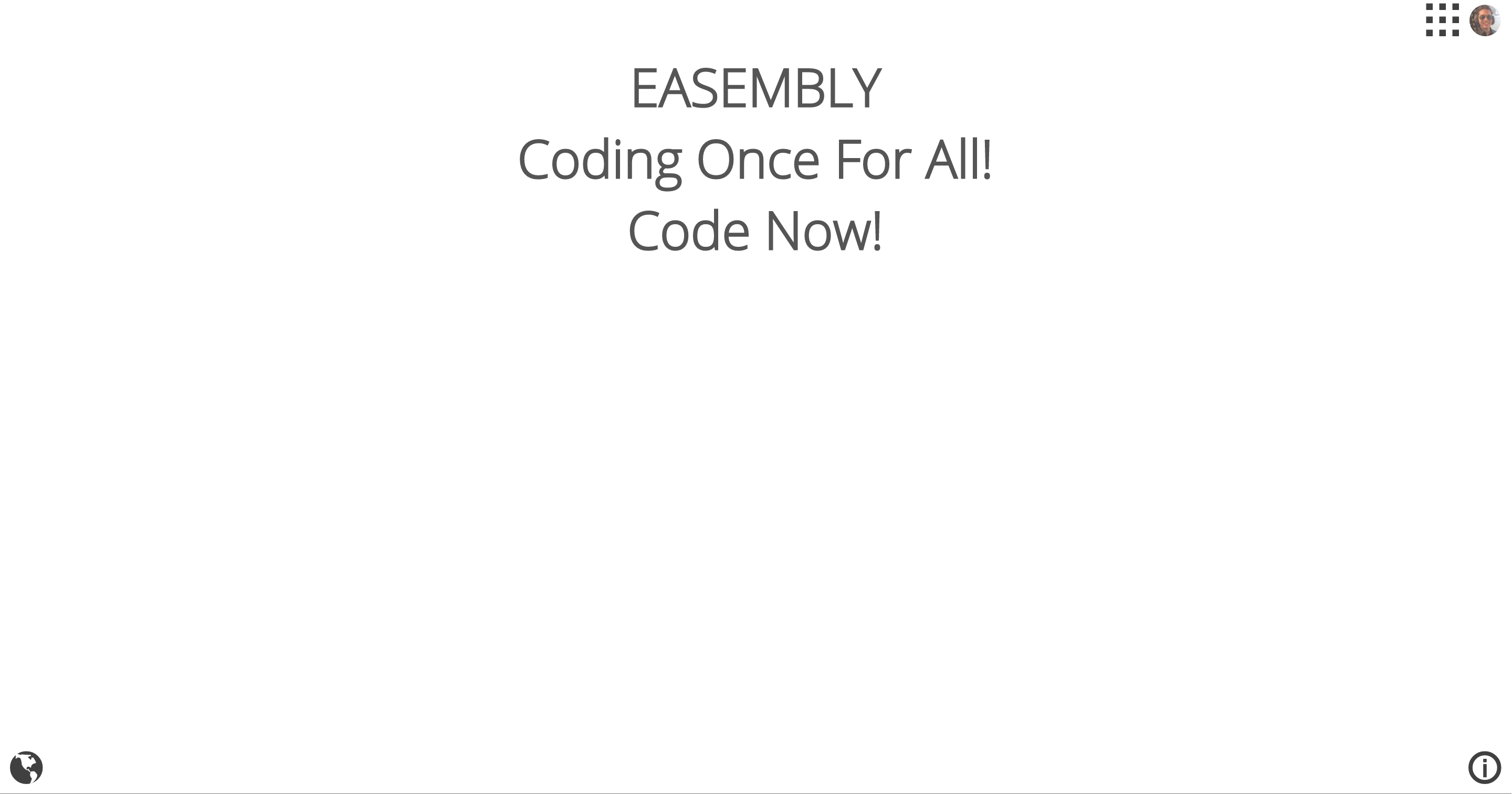
## 8.1 User Interface

### 8.1.1 Single Page

O design HTML/CSS foi um desafio nesse sistema, pois a experiência necessária em design HTML/CSS para fazer um sistema como este, não existia ou era pouca, e a medida que se vai avançando o design vai ficando maior e mais difícil de fazer a manutenção, o que para alguém que não tem experiência dificulta bastante, mas uma medida que facilitou bastante esta implementação foi a modularização da pagina feita a partir dos includes do PHP, o qual pode-se fundir ao HTML.

#### 8.1.1.1 Modularização da página

A única pagina acessada pelo usuário é a index (imagem). Esta pagina foi divida em módulos inicias (que nada mais são que outros arquivos PHP/HTML), os quais são: Head, Header, Body e Footer. Onde cada um carregará outros módulos a partir de configurações que são enviadas para o servidor a partir do JavaScript do cliente e/ou a partir de configurações padrões, Cache, Storage, IP address, localização e língua do navegador (como no caso do sistema de tradução).



(exemplo da Pagina Index usando o modulo de body contendo a Home em sua versão inicial)

O Head fica responsável por armazenar e informar para o navegador informações como título, autor, keywords, descrição, ícone e afins. É responsável também pelos links de CSS utilizados pelas páginas, incluindo a seleção do CSS específico para cada tipo e tamanho de dispositivo.

O Header fica responsável pelo menu estático no topo da página (imagem), o qual tem como atribuições a navegação entre as sub-páginas carregadas no Body e login/logoff.

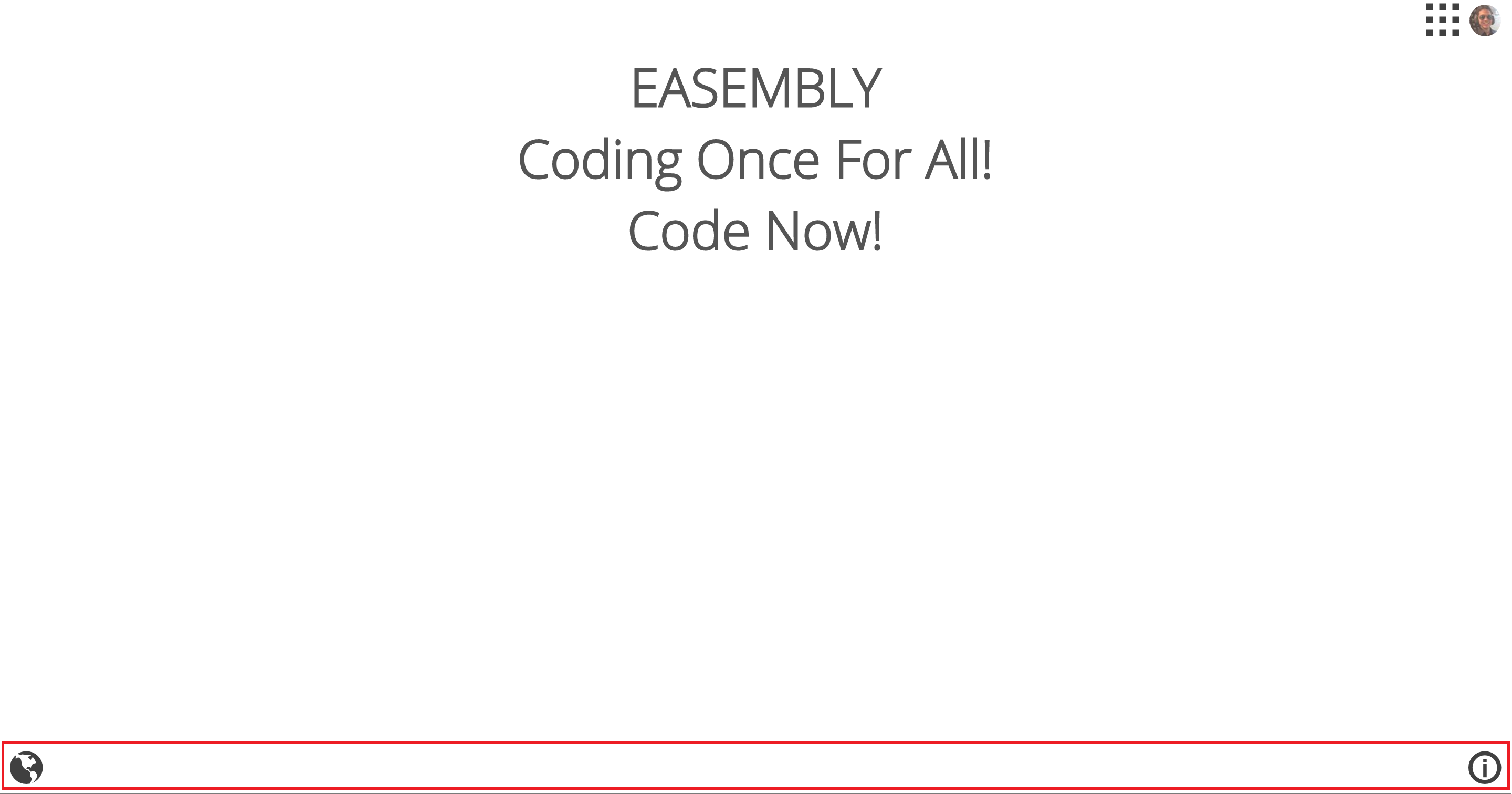
(imagem mostrando a área da pagina que é definida com o Header)

O Body é o local onde são carregaras as páginas onde o usuário vai navegar pelo site além de ter a possibilidade de carregar uma janela por cima de si mesmo.



(imagem mostrando a área da pagina que é definida com o Body)

E finalmente temos o Footer que tem um menu estático no fim dá página o possui dois botões que abrem suas respectivas janelas, um para tradução e outro para informações de contado e do website.



(imagem mostrando a área da pagina que é definida com o Footer)

### 8.1.2 Cache/Storage manipulation

Para acelerar o carregamento da pagina foi utilizado bastante Cache e Web Storage. Para que não exista requisições desnecessárias ao servidor. Ao carregar um modulo PHP/HTML, que no caso ao chegar no cliente chega como apenas um trecho HTML, ao chegar ele é salvo no Cache e no Storage. E antes de fazeruma requisição ao servidor ele checa se já existe no Storage o Cache e se ele existir ele simplesmente a carrega no seu trecho específico. Impedindo que exista requisições desnecessárias ao servidor e carregando as páginas mais rapidamente.

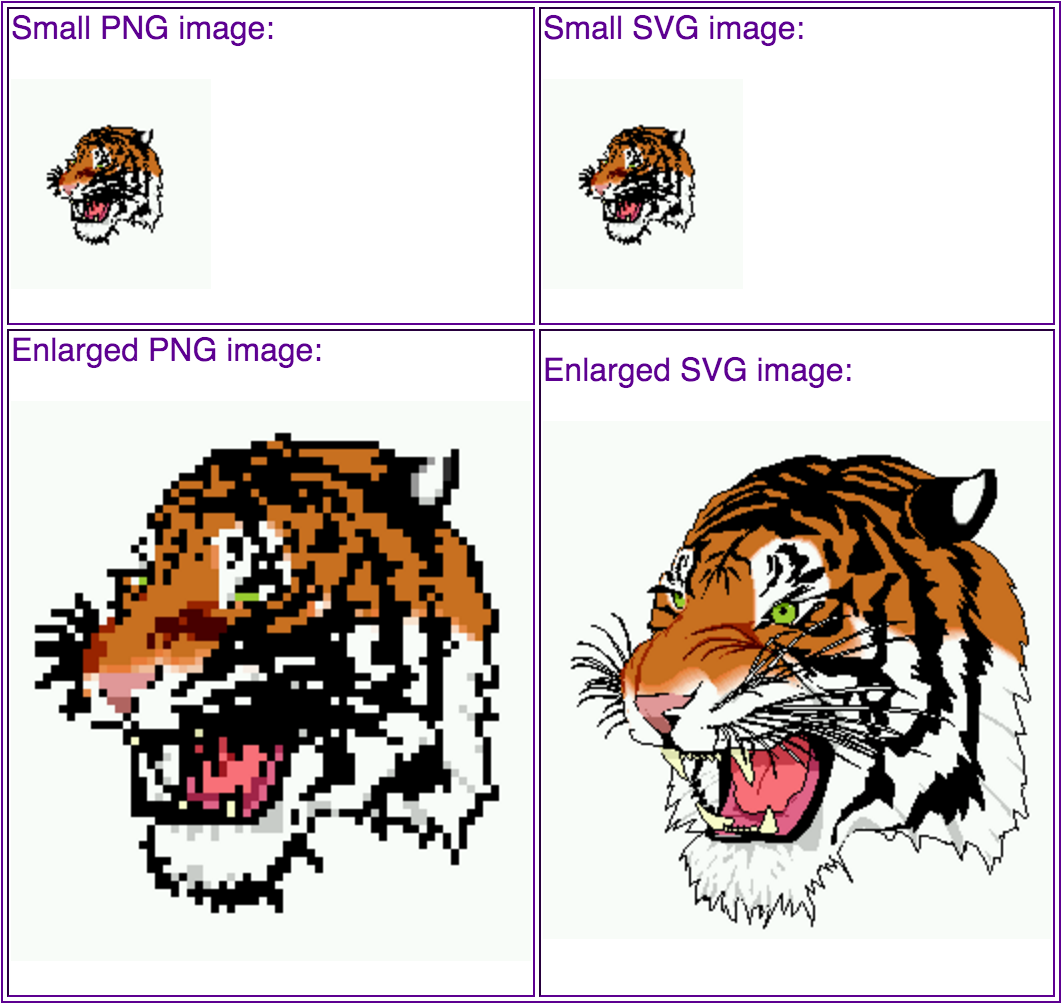
### 8.1.3 Ausência de Imagens

O uso de imagens sempre foi um gargalo no trafego de dados pela rede WEB. E elas aparentam ser imprescindíveis, à primeira vista, pois é uma das coisas digitalizadas que mais se aproxima do mundo real, o que as torna tão importantes para que se fazer uma interface intuitiva, afinal os botões, por exemplo, nada mais são do que uma reprodução de um elemento físico real muito conhecido pela humanidade, o botão.

Mas elas têm um problema que as transforma um gargalo, o tamanho. Inicialmente eram apenas uma matriz de cores, agora existem diversos formatos, que incluem inclusive compressão, que as vezes resultam até em perda de qualidade. Mesmo com toda essas inovações para que elas deixem de ser um gargalo ainda não são suficientes para muitos casos. A pouco tempo o Google fez uma transformação do seu logo, onde uma de suas preocupações foi o tamanho, que saltou de entorno de 14 kbytes para 305 bytes, o que resulta em um carregamento mais rápido, o que é muito importante num site como este que é tão usado [https://design.google.com/articles/evolving-the-google-identity/] (imagem). Mesmo assim a imagem ainda tem um problema difícil de ser contornado, que é permanecer com uma resolução agradável a todos os dispositivos, onde cada um tem uma resolução diferente e um tamanho físico diferente. E não adianta apenas dar um Zoom na imagem, pois esta imagem vai perder sua qualidade. Então como resolver um problema de tamanho sem ter uma equipe de design especializada nisso? Como resolver o problema de imagens diferentes para diferentes dispositivos?

(imagem mostrando a evolução da logomarca da Google)

Existe então uma maneira que não é muito utilizada pelos sistemas hoje em tia que são imagens vetorizadas que não utilizam muito espaço em disco e não perdem qualidade ao serem redimensionadas, como SVG [http://www.w3.org/TR/SVG-access/] (imagem), mas é difícil encontrar um editor que tenha esse tipo de suporte, e ainda podem ser que existam browsers e sistemas que as reproduza. Então foi seguido um caminho menos ortodoxo, o qual seria o de utilizar fontes, fontes de texto. Processadores de fontes são altamente performáticos, fontes ocupam um espaço mínimo em disco e possuem um grande conjunto de símbolos, onde cada um desses símbolos podem substituir uma imagem, sem contar que não perdem qualidade ao serem redimensionados e possuem um grande suporte.



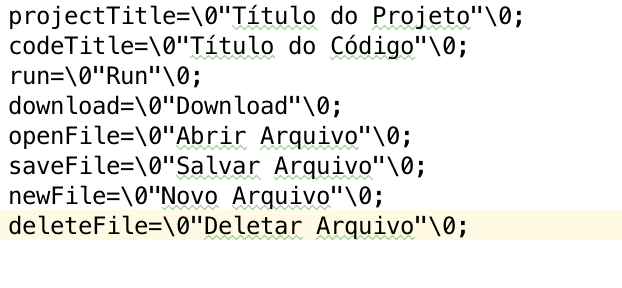
(imagem mostrando a diferença entre imagens vetorizadas e não-vetorizadas, em seu tamanho original e ao ampliar, no caso imagens do tipo PNG e SVG)

Então a estratégia seguida para contornar esse gargalo foi utilizar o mínimo de imagens possível, e onde não era possível removê-las, foram substituídas por símbolos de uma fonte.

### 8.1.4 Sistema de tradução

No design HTML não existe texto puro, sempre é uma variável carregada a partir de uma função de uma classe responsável pela linguagem. Tal classe checa a linguagem, seja por configuração, default, localização, IP address, língua do browser ou mesmo seleção do usuário, e o procura no sistema de arquivo se existe o arquivo para aquele módulo e para aquela linguagem, se existe ele abre o arquivo de linguagem e procura pela variável requisitada e por fim retorna seu valor, caso contrário ele retorna para a linguagem default que é inglês.

O arquivo de linguagem foi feito para que seja o mais simples possível, para que um tradutor leigo possa fazer um novo arquivo para uma nova linguagem sem ter que entender nada de tags complexas de um tipo de arquivo específico como XML ou JSON (imagem).

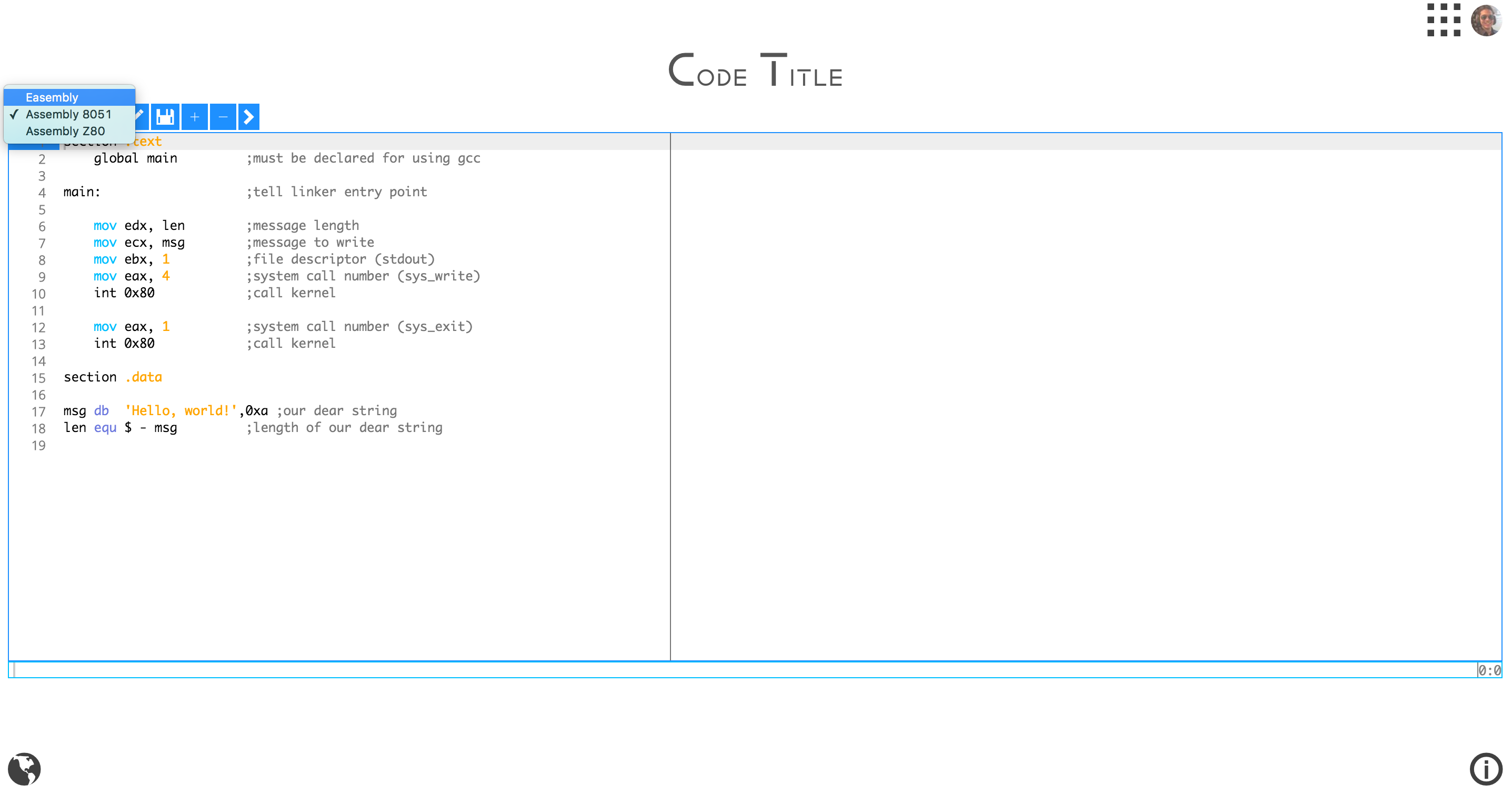


(Exemplo de um arquivo de linguagem)

### 8.1.5 Editor

Pra fazer um editor com Intelligent Code Completion e um Interpretador básico para erros de sintaxe, foi utilizado um embedded editor chamado Ace, infelizmente devido a algumas tecnologias usadas no projeto e por usar linguagens não suportadas por este editor, tiveram que ser feitas algumas modificações em seu código fonte, como inclusões e edições de arquivos JavaScript em pastas como Model, Documents, Requires, e outras mais. Além é claro de um CSS específico para se encaixar com o Design do site.

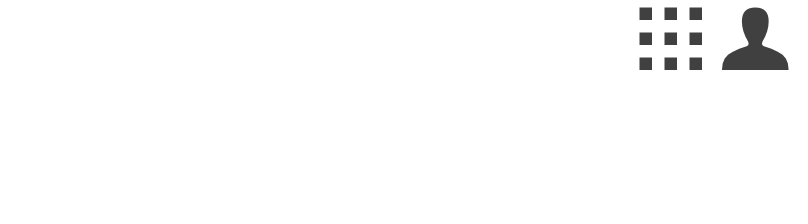
Neste caso ele pode ser usado para Codificar para Easembly, Assembly do Z80 e Assembly do 8051.



(imagem mostranto o Editor de código e suas possibilidades de linguagens para Edição)

### 8.1.6 Autenticação/Login

O Login e a Autenticação do usuário é feita pela API do Facebook, eliminando assim a preocupação com o registro, armazenamento e segurança dos dados do usuário.

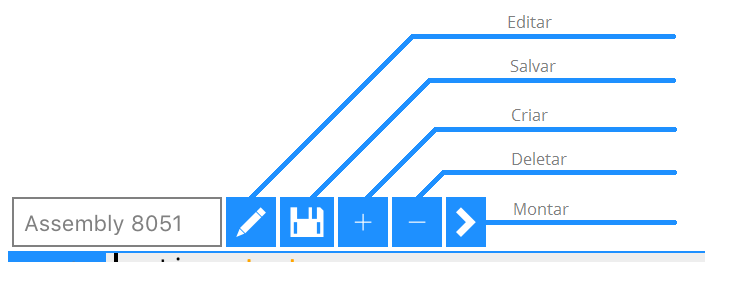




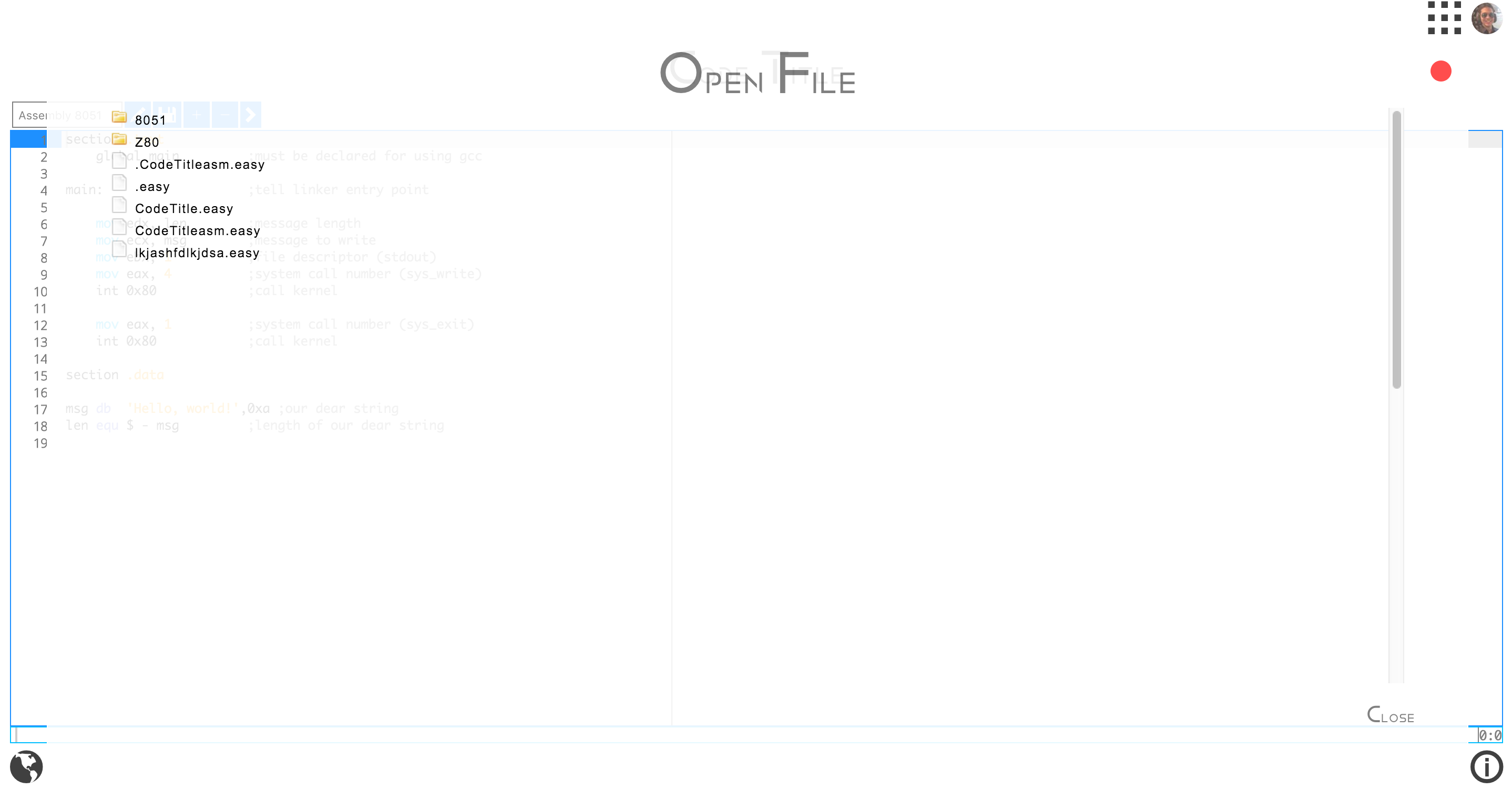
[Demonstração do fluxo de login]

### 8.1.7 CRUD

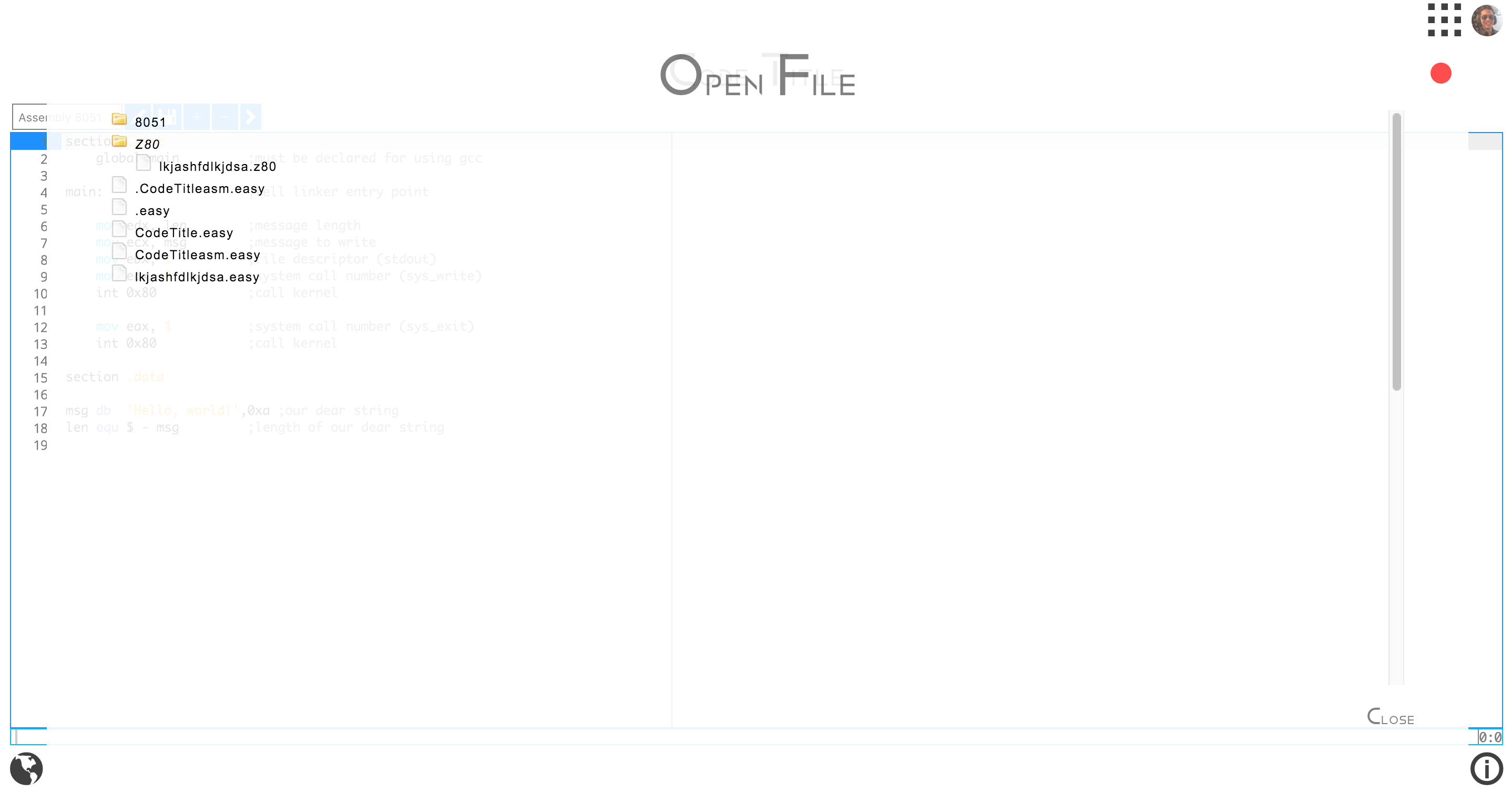
A UI permite que o usuário crie, pesquise, remova e edite (imagem) arquivos Assembly de 8051, Z80 e o Easembly (imagem 2).



(indicação do funcionamento de cada botão)



(Abrir Arquivo (Pesquisa), com pastas fechadas)



(Abrir Arquivo (Pesquisa), com uma pasta aberta)

## 8.2 Core

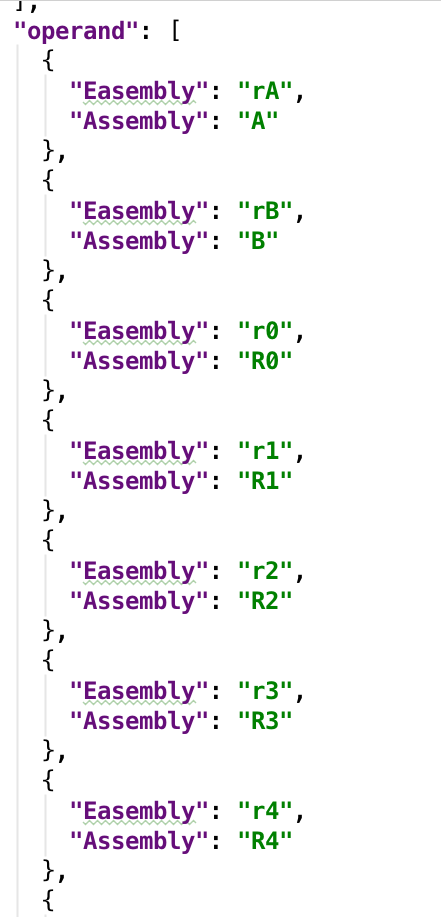
### 8.2.1 Tradutor

O Tradutor de Easembly para outro assembly funciona a partir do momento que o usuário clica no botão de salvar ou compilar. Feito isso é enviado o código feito pelo usuário para o servidor, este código é armazenado e então lido. A fase seguinte é a tradução.

Para cada assembly que é requisitado para se fazer uma tradução, é necessário um dicionário que diga para o tradutor como fazer esta tradução em questão. Este dicionário nada mais é que um arquivo JSON, este arquivo contém a instrução ou instruções que respectiva(s) no assembly destino para uma instrução do Easembly, assim como o mapeamento de registradores e identificadores de base numérica (prefixos e/ou sufixos que determinam a base de um valor numérico).









[exemplo de dicionário JSON, mostrando suas principais propriedades]

Com o dicionário então é possível fazer a tradução e então armazenar o arquivo assembly resultante da tradução.

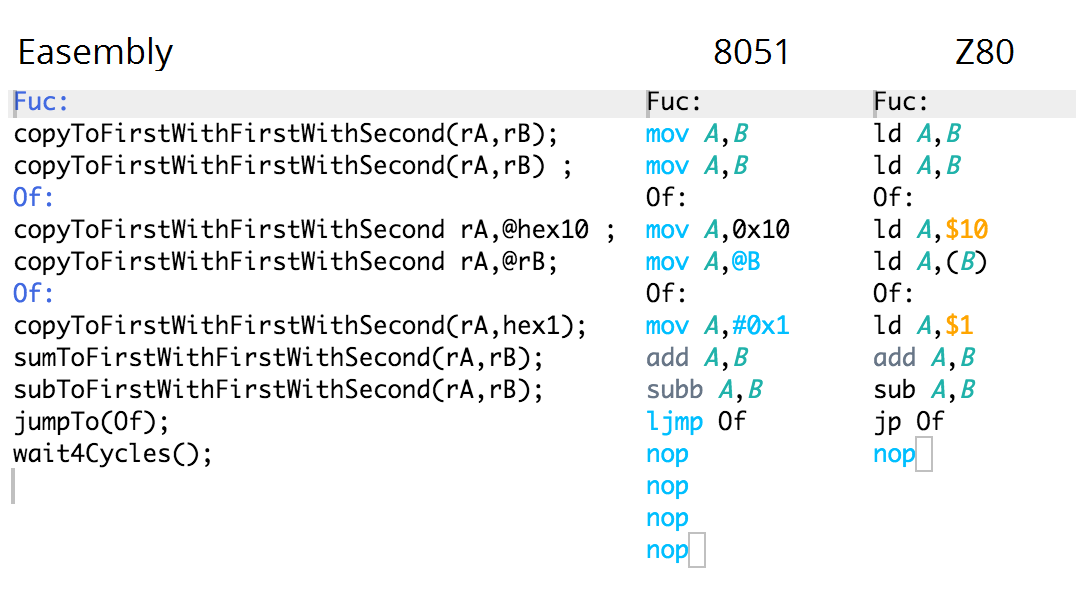
### 8.2.2 Compilador

O compilador utilizado é o SDCC, que pode ser utilizado via linha de comando, a qual é executada após a tradução ser feita, se o usuário tiver clicado no botão para compilar.

Este compilador foi escolhido pois pode fazer a compilação dos dois assemblies selecionados para o projeto, das arquiteturas dos processadores Z80 e 8051.

# 10 Análise dos resultados

A análise a seguir será superficial, pois o estudo só tem como objetivo mostrar a potencial possibilidade de funcionar para todos os assemblies, também devido ao prazo, não foi possível fazer estudos mais aprofundados, nem testes de performance, com estatísticas.



(imagem mostrando o resultado da tradução de um código Easembly para os Assemblies do 8051 e do z80)

A imagem acima mostra um código Easembly e o resultado de sua tradução para os assemblies dos processadores 8051 e Z80, respectivamente. É possivel notar que o Easembly vai ficar sempre limitado pelo processador mais “simples”. No exemplo da imagem é possível notar que como o Z80 tem uma instrução de delay de 4 ciclos de clock e o 8051 de apenas um ciclo, o sistema só vai suportar delay múltiplos de 4, perdendo, então, todas as possibilidades da instrução mais flexível, no caso um código feito em 8051 poderia ter um delay de 5 ciclos, enquanto um feito em Z80 ou gerados a partir o Easembly atual não poderiam.

Como o Easembly tem um conjunto de instruções menor que os conjunto das plataformas que gera ele perde no quesito performance, pois existe instruções em ambas arquiteturas que podem fazer o que necessita de até 5 do Easembly. O Easembly até poderia ter um conjunto maior que as arquiteturas destino atuais e incluir todas as instruções específicas, e assim acabar com o problema de desempenho. Caso isso fosse feito, outro problema apareceria, instruções específicas não existem nas outras plataformas, isso significa que aquela instrução, nas demais arquiteturas terá de ser compostas por uma sequencia de instruções presentes na arquitetura em questão que possa ter o mesmo resultado. Esta sequencia poderá implicar no uso de registradores. Para usá-los temos algumas possibilidades, caso não exista nenhum registrador a mais que o Easembly o sistema teria que de alguma fora saber se existe algum que não tenha sido usado ou usar a pilha caso não esteja cheia, caso tenha algum registrador a mais, ou sejam definidos um ou mais registradores exclusivos, é possível utilizá-los.

Outro problema encontrado foi o numero de registradores, e este numero tem que ser reduzido para no máximo a quantidade de registradores da arquitetura que tem menos. Isso implica em que o sistema restringirá o programador, pelo número do processador com menos registradores.

Enfim o Easembly sempre ficará restringido de alguma forma pelos processadores mais “simples”.

# 11 Conclusão

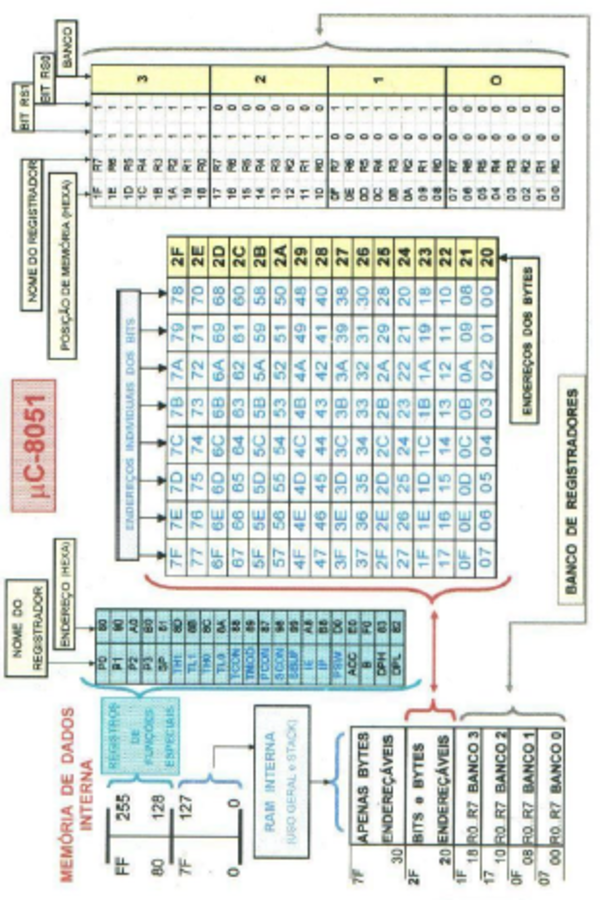
A partir da análise dos resultados obtidos foi possível perceber que possivelmente a arquitetura é possível para unir os Assemblies de diversos processadores, mas sempre ficará restringido de alguma forma pelos processadores mais básicos.

Então o projeto mostrou-se provavelmente possível, o que o torna de alguma forma útil, mas provavelmente terá a performance de seus programas caindo a medida que mais processadores forem se integrando a plataforma. Para solucionar este problema sugere-se um trabalho futuro. O trabalho futuro tem como objetivo ter uma arquitetura mais flexível para que as plataformas superiores não sejam tão mal aproveitadas devido a inferioridade das inferiores.

O trabalho futuro proposto aqui terá uma ideia semelhante ao atual Easembly, mas este o programador selecionará para quais plataformas deseja programar, e o sistema criará um set de instruções para aquele conjunto de processadores, de modo que aproveite melhor as instruções, já que o conjunto de instruções semelhantes será mais provavelmente maior com um conjunto reduzido de processadores selecionados pelo usuário, inclusive o número de registradores provavelmente seria similar, pois o programador selecionaria processadores que fossem compatíveis com o projeto que ele desejasse fazer, indicando uma provável semelhança.

# 12 ANEXO

## 12.1 Processadores

Foram selecionados dois processadores para os quais o sistema se preocupará que são o 8051 e o Z80.

### 12.1.1 8051

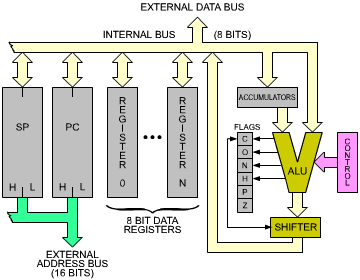
#### 12.1.1.1 Arquitetura

#### 12.1.1.2 Assembly

Todos os membros da família 8051 executam o mesmo conjunto de instruções. Este conjunto é composto por instruções otimizadas para aplicações de controle, facilitando as operações de dados através de vários modos de endereçamento. Capacita ainda a operação de variáveis de um bit, permitindo operação em sistemas que demandam processamento booleano. Vamos descrever rapidamente o modo de operação de várias das instruções contidas neste conjunto. Maiores informações podem ser encontradas nos manuais dos fabricantes, os quais descrevem detalhadamente o conjunto de instruções.

### 12.1.2 Z80

#### 12.1.2.1 Arquitetura



#### 12.1.2.2 Assembly

O 8080 (base do Z80) usa instruções que podem variar seu tamanho de 1 até 3 bytes. No entanto, o Z80 é equipado com instruções indexadas adicionais, as quais podem requerer um byte a mais. No caso do Z80, os opcodes são, em geral, um byte, exceto para instruções especiais as quais requerem dois bytes de opcode.

Algumas instruções requerem que um byte de dados siga o opcode. Neste caso, esta instrução será uma de 2 bytes, onde o segundo byte são dados (exceto para indexação, o qual adiciona um byte extra). Pra o resto das instruções, é possível que seja requerido a especificação de um endereço.

Um endereço requer 16 bits (2 bytes). Sabendo disto uma instrução que use endereço pode variar seu tamanho de 3 à 4 bytes.

Para cada byte de instrução, a unidade de controle terá de realizar um fetch de memoria, o qual requer 4 ciclos de clock. Então isso significa que quanto menor a instrução, mais rápida será a execução (assim como a maioria dos processadores).

# REFERÊNCIAS

[1] CANFORA, G. ***Migrating interactive legacy systems to Web services***. European Conference. Bari. 22-24 *March* 2006. Disponível em: <http://ieeexplore. ieee.org/xpl/login.jsp?tp=&arnumber=1602355&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs\_all.jsp%3Farnumber%3D1602355 >. Acesso em: 26 maio 2015.

[2] KIM, S.M.; ROSU, M.C. ***A Survey of Public Web Services***. *E-Commerce and Web Technologies*.*Springer/Berlin- Heidelberg*, 2004. Disponível em: <http://link.springer.com/book/10.1007%2Fb100074>. Acesso em: 26 maio 2015.

[3] CHU, K.W.S.; KENNEDY, D.M. *Using online collaborative tools for groups to co-construct knowledge*. ***Online Information Review***, v. 35, n. 4, p. 581-597, 2011.

[4] MILLER, M. ***Cloud Computing****: Web-Based Applications That Change the Way You Work and Collaborate Online. Que Publishing*, 2008.

[5] BOURNE, J.; HARRIS, D;; MAYADAS, F. *Online Engineering Education: Learning Anywhere, Anytime*. ***Journal of Engineering Education***. p. 130-146, 2005.

[6] HYDE, R. *The Art of Assembly Language*. No Starch Press, 2003.

[7] SMITH, J.R. ***Programming the PIC Microcontroller with MBasic***. EUA: *Newnes*, 2005.

[8] KUAN-CHENG, L. ***An On-line Instruction/Learning Environment for Supporting Individualized Learning in Java Programming***. *Taiwan*: *National Chung Hsing University*, 2007.

[9] MORE, A. ***Web Based Programming Assistance Tool for Novices****. International Conference: Chennai, Tamil Nadu, 14-16 July 2011*. Disponível em: <http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6004399&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs\_all.jsp%3Farnumber%3D6004399>. Acesso em: 26 maio 2015.

[10] DATTA, A. ***Online compiler as a cloud service***. *International Conference: Ramanathapuram, 8-10 May 2014*. Disponível em: <<http://ieeexplore.ieee.org/xpl/> login.jsp?tp=&arnumber=7019416&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs\_all.jsp%3Farnumber%3D7019416>. Acesso em: 26 maio 2015.

[11] MOHAMMED, S. ***WIDE an interactive Web integrated development environment to practice C programming in distance education***. *International Conference:* Porto*, Oct. 31 2013 - Nov. 1 2013*. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6701964>. Acesso em: 26 maio 2015.

[12] Minzhe, G. ***Back to Results Design of Online Runtime and Testing Environment for Instant Java Programming Assessment***. *International Conference:* \_\_\_\_\_\_Local\_\_\_\_\_\_*, \_\_\_\_\_\_*Data*\_\_\_\_\_\_*. Disponível em: <\_\_\_\_\_\_\_\_\_\_\_\_\_Endereço eletônico\_\_\_\_\_\_\_\_\_\_\_\_>. Acesso em: 26 maio 2015.

[13] Ansari, A.N. ***Online C/C++ compiler using cloud computing***. *International Conference: Hangzhou,* 26-28 July 2011. Disponível em: <http://ieeexplore.ieee.org/ xpl/login.jsp?tp=&arnumber=6002124&url=http%3A%2F%2Fieeexplore.ieee.org%2Fiel5%2F5981419%2F6001647%2F06002124.pdf%3Farnumber%3D6002124>. Acesso em: 26 maio 2015.

[14] SALIN MD, S.I. ***Back to Results One-pass Assembler Design for a Low-end Reconfigurable RISC processor***. *International Conference:* \_\_\_\_Local\_\_\_\_*, \_\_\_\_*2014*\_\_\_*. Disponível em: <\_\_\_\_\_\_\_\_\_\_\_\_\_Endereço eletônico\_\_\_\_\_\_\_\_\_\_\_\_>. Acesso em: 26 maio 2015.

[15] ARBONE, C. **M*odel-Driven Inline Assembler Generator for Retargetable Compilers***. *International Conference:* Bucharest*,* 29-31 May 2013. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6569246>. Acesso em: 26 maio 2015.

[16] Hatfield, Bo. A GENERAL-PURPOSE CUSTOM-DESIGN EDASSEMBLER IN C. : IEEE, 2003.

[17] R. Tavernier, Karel. Macro-Based Cross Assemblers. : IEEE, 1980.

[18] Nakano, Koji. Processor, Assembler, and Compiler Design Education using an FPGA. : IEEE, 2008.

[19] Mathiske, Bernd. An assembler and disassembler framework for JavaTM programmers. : ScienceDirect, 2008.

[20] J. Davis, Ian. From Whence It Came: Detecting Source Code Clones by Analyzing Assembler. 2010.

[21] Md Salin, S.I. Two-pass Assembler Design for a Reconfigurable RISC Processor.: IEEE, 2013.