



**FUNDAÇÃO EDSON QUEIROZ  
UNIVERSIDADE DE FORTALEZA – UNIFOR  
CENTRO DE CIÊNCIAS TECNOLÓGICAS – CCT  
CURSO ENGENHARIA DE COMPUTAÇÃO**

**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**

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

JUDAH HOLANDA CORREIA LIMA

**PROPOSTA DE ARQUITETURA DE ASSEMBLER  
ONLINE ÚNICO PARA DIVERSOS PROCESSADORES**

Trabalho de Conclusão de Curso – TCC, apresentado ao Curso de Engenharia de Computação, do Centro de Ciências Tecnológicas da Universidade de Fortaleza para obtenção do grau de bacharel.

Data da aprovação: Fortaleza, \_\_\_\_\_ de 2015.

BANCA EXAMINADORA



Assinatura:

Prof. Ms. Danilo Reis Vasconcelos – Universidade de Fortaleza  
Orientador



Assinatura:

Prof. Ms. Marcelo Ferreira de Sousa – Universidade de Fortaleza  
Membro

Assinatura:

Prof. Ms. José Wagner V. Alves – Universidade de Fortaleza  
Membro

Agradeço em primeiro lugar a Deus que iluminou o meu caminho durante esta caminhada. E depois a minha família que, com muito carinho e apoio, não mediram esforços para que eu chegassem até esta etapa de minha vida.

## **AGRADECIMENTOS**

A Deus por ter me dado saúde e força para superar as dificuldades.

Ao meu orientador Danilo Reis, pelo suporte no pouco tempo que lhe coube, pelas suas correções e incentivos.

Aos meus pais, pelo amor, incentivo e apoio incondicional.

E a todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

*“Tudo se submeterá ao exame da criança e nada se lhe enfiará na cabeça por simples autoridade e crédito. Que nenhum princípio, de Aristóteles, dos estóicos ou dos epicuristas, seja seu princípio. Apresentem-se-lhe todos em sua diversidade e que ele escolha se puder. E se não o puder fique na dúvida, pois só os loucos têm certeza absoluta em sua opinião”*

*Montaigne*

## LISTA DE FIGURAS

Figura 1: Exemplo de um diagrama de funcionamento de uma IDE .....	14
Figura 2: Exemplos de editores de texto: Microsoft Notepad e Microsoft WORD.....	15
Figura 3: Exemplo de editor de código, Visual Code .....	16
Figura 4: Exemplo de IDE, Microsoft Visual Studio 2015.....	17
Figura 5: Exemplo de funcionamento da IntelliSense do Visual Studio .....	18
Figura 6: Fluxo de execução de um código interpretado .....	18
Figura 7: Fluxo de execução de um programa tradutor de linguagens de programação .....	20
Figura 8: Árvore de responsabilidades.....	20
Figura 9: Fluxo de funcionamento .....	21
Figura 10: Exemplo de execução de um programa Java .....	23
Figura 11: Funcionamento do Java para cada plataforma .....	24
Figura 12: Fluxo de geração e execução de bytecode.....	24
Figura 13: Instância da JVM .....	25
Figura 14: Geração de um Java bytecode .....	26
Figura 15: demonstração do funcionamento da estrutura .NET.....	28
Figura 16: Fluxo de funcionamento de geração de código de máquina do sistema proposto.....	30
Figura 17: Fluxo de funcionamento de tradução do sistema proposto.....	32
Figura 18: Exemplo da Página Index, ao utilizar o módulo de body contendo a Home, em sua versão inicial .....	36

Figura 19: Demonstração da área da página que é definida com o Header.....	36
Figura 20: Apresentação da área da página que é definida com o Body.....	37
Figura 21: Imagem da área da página que é definida com o Footer.....	37
Figura 22: Demonstração da evolução da logomarca da Google .....	39
Figura 23: Diferença entre imagens vetorizadas e não vetorizadas, em tamanho original e ampliadas. No caso, imagens do tipo PNG e SVG.....	40
Figura 24: Exemplo de um arquivo de linguagem.....	41
Figura 25: Editor de código e suas possibilidades de linguagens para Edição.....	42
Figura 26: Demonstração do fluxo de login.....	43
Figura 27: Indicação do funcionamento de cada botão.....	44
Figura 28: Abrir Arquivo (Pesquisa) com pastas fechadas .....	44
Figura 29: Abrir Arquivo (Pesquisa) com uma pasta aberta.....	45
Figura 30: Exemplo de dicionário JSON, mostrando suas principais propriedades..	46
Figura 31: Imagem do resultado da tradução de um código Easembly para os Assemblies do 8051 e do z80 .....	48
Figura 32: Imagem do resultado da tradução de um código de Delay Easembly para os Assemblies do 8051 e do z80.....	49
Figura 33: Imagem do resultado da tradução de um código de Multiplicação Easembly para os Assemblies do 8051 e do z80 e sua versão feita manualmente para 8051 .....	50
Figura 34: Imagem do resultado da tradução de um código de Divisão Easembly para os Assemblies do 8051 e do z80 e sua versão feita manualmente para 8051 .	50
Figura 36: Imagem do resultado da tradução de um código de Divisão Easembly para os Assemblies do 8051 e do z80 e sua versão feita manualmente para 8051 .	60

## 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*, validado através de uma implementação simplificada. A ideia proposta foi utilizar um reduzido conjunto de instruções, comum a, inicialmente, dois processadores, mas suficientemente grande para realizar todas as operações básicas. Os resultados sugerem que, mesmo em estágio inicial, a arquitetura parece ser viável. A proposta foi implantada em PHP, hospedada em um servidor simples, e mesmo assim funcionou rapidamente para os testes realizados. Então, pode-se 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 .....	8
INTRODUÇÃO .....	11
1 DISCUSSÃO TEÓRICA .....	14
1.1 Editor .....	15
1.2 Interpretador .....	18
1.3 Tradutor .....	19
1.4 Compilador .....	20
1.5 Assembler (Montador/Ligador) .....	22
1.6 Linker (Link-Editor) .....	22
2 ARQUITETURAS RELACIONADAS .....	23
2.1 Java .....	23
2.1.1 Java Virtual Machine (JVM).....	25
2.1.2 Java Bytecode .....	25
2.2 .NET.....	27
2.2.1 Common Language Runtime (CLR) .....	28
2.2.2 Common Language Infrastructure (CLI) .....	29
2.2.3 Common Intermediate Language (CIL) .....	29
3 ARQUITETURA.....	30
3.1 Editor .....	31
3.2 Tradutor .....	32
3.3 Montador.....	34

4 ESTUDO DE CASO .....	35
4.1 <i>User Interface</i> .....	35
4.1.1 Single Page .....	35
4.1.1.1 Modularização da página .....	35
4.1.2 Cache/Storage manipulation .....	38
4.1.3 Ausência de Imagens .....	38
4.1.4 Sistema de tradução .....	40
4.1.5 Editor.....	41
4.1.6 Autenticação/Login .....	42
4.1.7 CRUD.....	44
4.2 Core .....	45
4.2.1 Tradutor .....	45
4.2.2 Compilador.....	47
5 ANÁLISE DOS RESULTADOS .....	48
CONCLUSÃO .....	52
REFERÊNCIAS .....	53
ANEXO .....	58

## INTRODUÇÃO

Atualmente, é possível perceber que *softwares*, programas que rodavam em ambiente *off-line*, estão se tornando serviços *web* [1,2]. Possibilitando seu uso em qualquer lugar que possua conexão com a *internet*, e facilitando 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 operacionais distintos, pois um ou mais programas podem não estar disponíveis para a plataforma utilizada no projeto. Outra importante questão é o tempo gasto para instalação desses softwares necessários para a elaboração e execução do projeto.

Depois desta etapa 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 estejam 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. A integração pode resultar em: modificação, encapsulamento e criação de trechos de códigos para integração. Isto implica em ler um código que, geralmente, não está seguindo o mesmo padrão de codificação e 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. Como a ferramenta está hospedada na nuvem, a colaboração entre programadores fica facilitada. Pois tanto a edição e como compartilhamento do código podem ser feitos a partir de qualquer dispositivo que possua um browser, entrada para texto e conexão com a internet.

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.

Outra questão relevante é que ao migrar softwares para diferentes plataformas é necessário aprender algo novo, e 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 o entendimento [7]. Como é o caso de uma linguagem nova, e no caso do *Assembly* este é um problema bem comum, principalmente com seus mnemônicos.

Como o *Assembly* utiliza mnemônicos, isso demanda o estudo e aprendizado de cada instrução, tornando a linguagem bem mais complexa que as 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 própria língua oficial, cria-se outro problema, visto que se impede que pessoas que não falem tal língua consigam utilizar o seu código escrito. 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 proposta abordar estas questões levantadas através de uma arquitetura que permita a edição e armazenamento, em um servidor na nuvem, de um Assembly comum a diversas plataformas. Para a validação foi feita uma implementação simplificada, onde foca apenas em duas plataformas: o microcontrolador da Intel 8051 e o microprocessador z80 da Zilog. O Assembly genérico proposto, que recebeu o nome *Easembly*, é um Assembly de nível um pouco mais alto que consiste no conjunto de instruções comum a ambas as plataformas, mas que mesmo que reduzido, ainda seja possível realizar todas as operações básicas. O projeto funcionará inicialmente de forma básica, pois a endianness (little-endian ou big-endian) será nesta fase ignorada. O montador do *Assembly* em si não ficará no escopo do projeto, sendo usado apenas com linha de comando no Linux do pacote SDCC.

O objetivo geral deste projeto é potencializar a reutilização de códigos com a criação de um *Assembly* unificado.

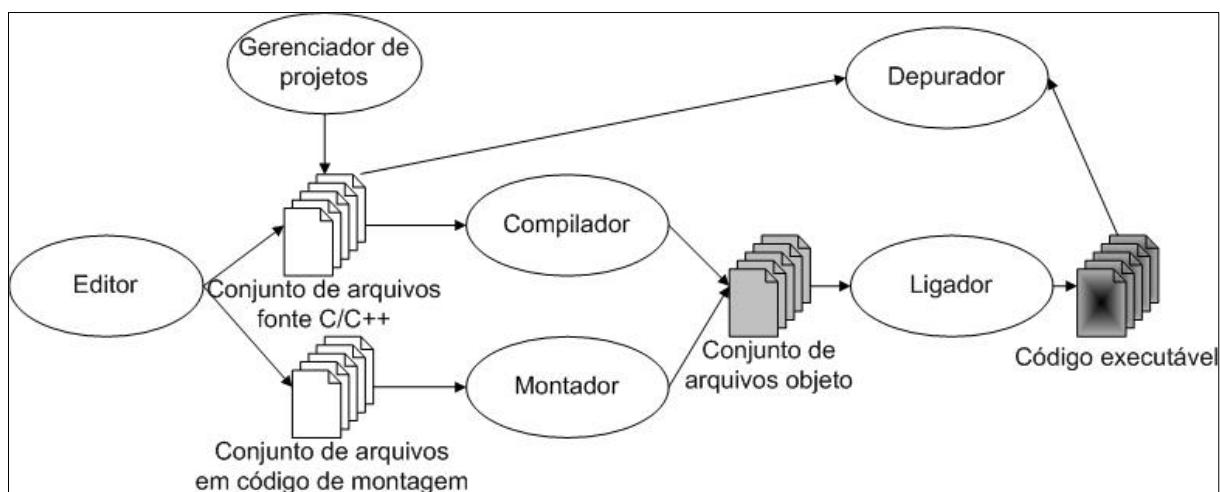
Como objetivos específicos destacam-se:

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

# 1 DISCUSSÃO TEÓRICA

A ideia deste conjunto reduzido de instruções é inspirada na arquitetura RISC dos processadores modernos. Para tornar este conjunto de instruções multiarquitetura é necessário reduzir o número de instruções ao máximo para que se tenha certeza de que ele vai estar presente em qualquer processador.

Mas para que o usuário possa codificar é necessário um ambiente de desenvolvimento, que no caso desta arquitetura é online. Um ambiente que integra todas as ferramentas necessárias para que o programador desenvolva, monte e depure erros de código, é chamado de IDE, ou Integrated Development Environment, embora nem sempre venha a ter todas estas funcionalidades. Um exemplo deste tipo de ferramenta pode ser visto na Figura 1. Na implementação realizada apenas uma parte deste tipo de programa foi codificada, deixando de lado o depurador de erros.



**Figura 1:** Exemplo de um diagrama de funcionamento de uma IDE

Fonte: <http://www.devmedia.com.br/processo-de-traducao-e-execucao-de-programas/26872>

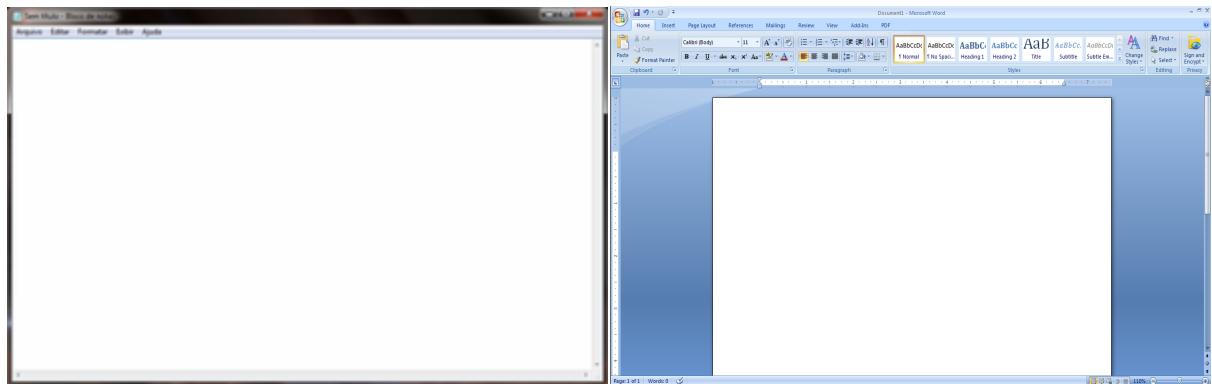
Na literatura científica é possível encontrar diversos trabalhos publicados, nos quais a ideia de uma IDE *Online* pode ser percebida, conforme mostrado na Figura 1. Embora não tenham sido encontrados trabalhos que exploram a ideia de *Assembler* unificado, foi possível encontrar trabalhos que utilizam uma linguagem independente de arquitetura, como Java [8,9,10,11,12,13]. Também foram encontrados *Assemblers* para processadores reconfiguráveis [14], embora não seja

o foco deste estudo, algumas instruções são mantidas ao reconfigurar processadores e isto foi de grande utilidade ao projetar o conjunto de instruções. E finalmente um gerador *Assembler* [15], que desempenha um papel fundamental na ideia de utilizar um tradutor de *Assemblies*.

## 1.1 Editor

Existem diversos tipos de editores, dentre eles se destacam dois: os que editam texto e código-fonte.

Editores de texto são aqueles que permitem a edição de arquivos de texto, em pelo menos um formato. Temos como exemplo de editor de texto o Word da Microsoft que conta com diversas ferramentas para formatação de texto e imagem, incluindo de efeitos até ferramentas para design de diagramas e gráfico, conforme mostrado na Figura 2. Outro exemplo é o Microsoft Notepad que inclui uma gama bem menor de funcionalidades, mas ainda permite a edição de texto, o qual também pode ser visualizado na Figura 2.



**Figura 2:** Exemplos de editores de texto: Microsoft Notepad e Microsoft WORD  
Fonte: Próprio Autor

Entretanto, o editor de código-fonte é um editor de texto com funcionalidades mais específicas. Este tipo de programa é desenhado especificamente para editar código-fonte de *Software/Firmware*. Editores como este podem ser uma aplicação, por si só, como pode ser visto na Figura 3. Como também podem fazer parte de outra, no caso, como de uma IDE, conforme mostrado na Figura 4, ou mesmo de um *Web Browser*. Este tipo de editor é ferramenta fundamental para que um programador possa fazer seu trabalho, que é escrever e editar código.

```

jquery.fileupload.css
1 @charset "UTF-8";
2 /*
3 * jQuery File Upload Plugin CSS 1.3.0
4 * https://github.com/blueimp/jQuery-File
5 *
6 * Copyright 2013, Sebastian Tschan
7 * https://blueimp.net
8 *
9 * Licensed under the MIT license:
10 * http://www.opensource.org/licenses/MIT
11 */
12
13 .fileinput-button {
14 position: relative;
15 overflow: hidden;
16 }
17 .fileinput-button input {
18 position: absolute;
19 top: 0;
20 right: 0;
21 margin: 0;
22 opacity: 0;
23 -ms-filter: 'alpha(opacity=0)';
24 font-size: 200px;
25 direction: ltr;
26 cursor: pointer;
27 }
28
29 /* Fixes for IE < 8 */
30 @media screen\9 {
31 .fileinput-button input {
32 filter: alpha(opacity=0);
33 font-size: 100%;
34 height: 100%;
35 }
36 }
37

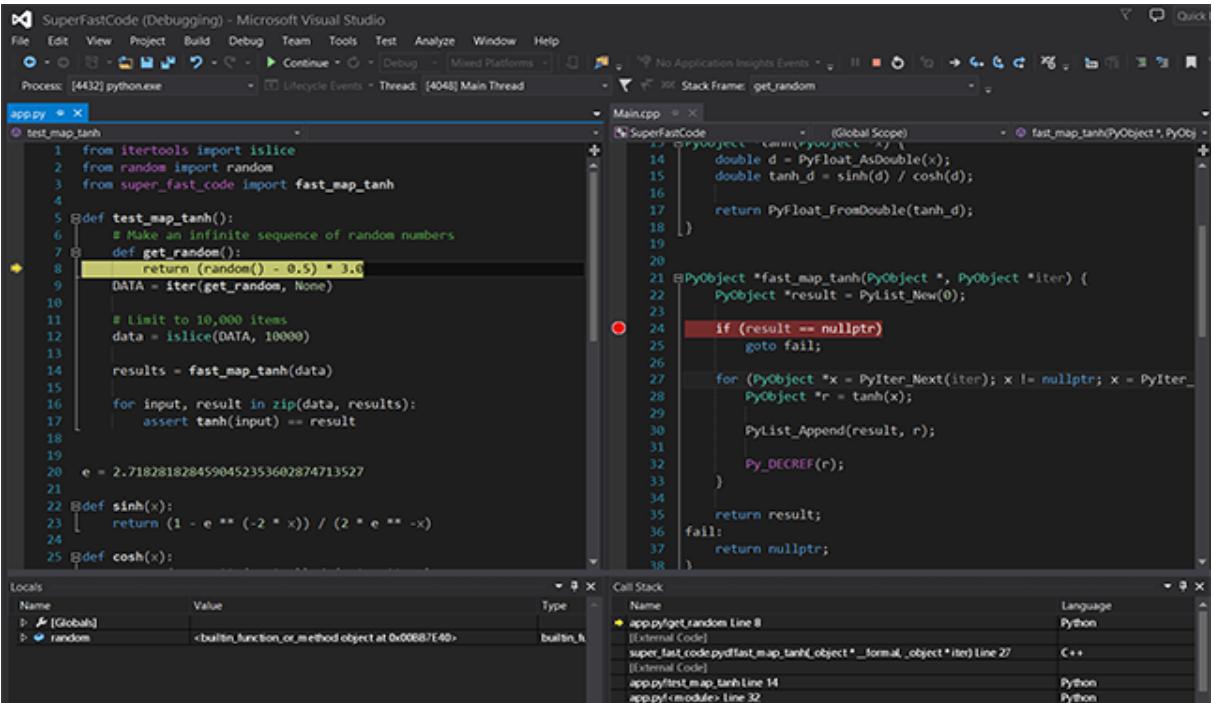
demo.css
1 @charset "UTF-8";
2 /*
3 * jQuery File Upload Demo CSS 1.1.0
4 * https://github.com/blueimp/jQuery-File
5 *
6 * Copyright 2013, Sebastian Tschan
7 * https://blueimp.net
8 *
9 * Licensed under the MIT license:
10 * http://www.opensource.org/licenses/MIT
11 */
12
13 body {
14 max-width: 750px;
15 margin: 0 auto;
16 padding: 1em;
17 font-family: "Lucida Grande", "Lucida S
18 color: #fff;
19 line-height: 1.4em;
20 background: #222;
21 color: #fff;
22 -webkit-text-size-adjust: 100%;
23 -ms-text-size-adjust: 100%;
24 }
25 a {
26 color: #orange;
27 text-decoration: none;
28 }
29 img {
30 border: 0;
31 vertical-align: middle;
32 }
33 h1 {
34 line-height: 1em;
35 }
36 blockquote {
37 padding: 0 0 15px;
38 margin: 0 0 20px;
39 border-left: 5px solid #eee;
40 }
41 table {
42 width: 100%;
43 margin: 10px 0;
44 }

jquery.fileupload-ui.css /css
1 @charset "UTF-8";
2 /*
3 * jQuery File Upload UI Plugin CSS 9.0.0
4 * https://github.com/blueimp/jQuery-File
5 *
6 * Copyright 2010, Sebastian Tschan
7 * https://blueimp.net
8 *
9 * Licensed under the MIT license:
10 * http://www.opensource.org/licenses/MIT
11 */
12
13 .fileupload-buttonbar .btn,
14 .fileupload-buttonbar .toggle {
15 margin-bottom: 5px;
16 }
17 .progress-animated .progress-bar,
18 .progress-animated .bar {
19 background: url("../img/progressbar.gif
20 filter: none;
21 }
22 .fileupload-process {
23 float: right;
24 display: none;
25 }
26 .fileupload-processing .fileupload-proces
27 .files .processing .preview {
28 display: block;
29 width: 32px;
30 height: 32px;
31 background: url("../img/loading.gif") c
32 background-size: contain;
33 }
34 .files audio,
35 .files video {
36 max-width: 300px;
37 }
38
39 @media (max-width: 767px) {
40 .fileupload-buttonbar .toggle,
41 .files .toggle,
42 .files .btn span {
43 display: none;
44 }

```

**Figura 3:** Exemplo de editor de código, Visual Code  
Fonte: <https://code.visualstudio.com/Docs/editor/codebasics>

Integrated Development Environment como já discutido é um ambiente que integra todas as funcionalidades básicas para a edição e debug de códigos fonte, como ferramentas de montagem e/ou compilação. Um exemplo de programa que reúne todas estas funcionalidades é o Microsoft Visual Studio, conforme mostrado na Figura 4. Embora existam IDE's que não possuem funcionalidades de montagem ou compilação, como em IDE's com foco em alguma linguagem interpretada, elas são normalmente definidas como ambientes que possuem todas as funcionalidades descritas anteriormente.



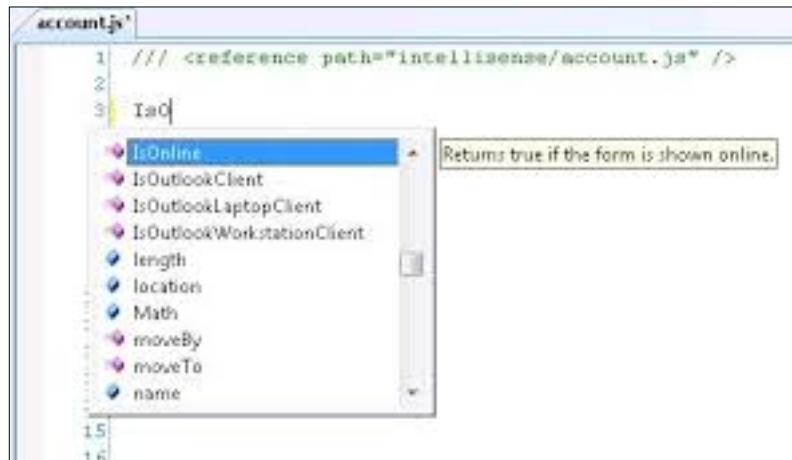
**Figura 4:** Exemplo de IDE, Microsoft Visual Studio 2015

Fonte: <https://www.visualstudio.com/en-us/products/visual-studio-community-vs.aspx>

Programas de grandes proporções podem chegar a possuir milhares de linhas de código, que seriam extremamente difíceis de serem produzidos sem editores de código. Tais editores provêm de uma série 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[16].

O termo *Intelligent code completion*, inicialmente, foi popularizado como "picklist", e até hoje em algumas implementações ainda o referenciam assim[17,18]. Uma funcionalidade que o ambiente completa o código que está sendo digitado pelo programador à medida que este digita, tendo como base o contexto em que está situado. Esta ferramenta está presente em vários ambientes de programação[19,20], com a função de acelerar o processo de codificação e reduzir erros de digitação e outros erros comuns, além de eliminar a necessidade de digitar certo termo por completo. A ferramenta irá sugerir a opção, o usuário a selecionará e o programa a completará. Normalmente, esta funcionalidade é implementada através de *pop-up's* com sugestões ao digitar. *Intelligent code completion* e ferramentas relacionadas, servem como documentação e desambiguação para nomes de variáveis, funções e

métodos, utilizando *reflection*. Um exemplo de implementação é o *Visual Studio's IntelliSense*[21], mostrado na Figura 5, abaixo:



**Figura 5:** Exemplo de funcionamento da IntelliSense do Visual Studio

Fonte: <http://crm.vdsnickt.eu/ms-crm-javascript-intellisense-generator/>

## 1.2 Interpretador

O interpretador é um programa de computador que executa instruções escritas em uma determinada linguagem de programação. Ou seja, um código feito em uma linguagem interpretada, necessita de uma aplicação ou ambiente para executar. Linguagens como *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 traduzi-lo em alguma eficiente representação intermediária, e depois executá-lo. Como por exemplo, o *Java* que, ao ser compilado, é representado em uma linguagem intermediária, o *bytecode*, que depois é executada por uma máquina virtual, conforme mostrado pela Figura 6.



**Figura 6:** Fluxo de execução de um código interpretado

Fonte: <https://panda.ime.usp.br/pensepy/static/pensepy/01-Introducao/introducao.html>

Para que este tipo de programa execute sua finalidade, alguns interpretadores 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 de possibilitar uma maior flexibilidade, pois Interpretadores podem ser implementados em diferentes tipos de arquiteturas de computadores, conservando a linguagem que 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, gerando um overhead cada vez que este é executado, e depois realizar 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"[22,23,24].

### 1.3 Tradutor

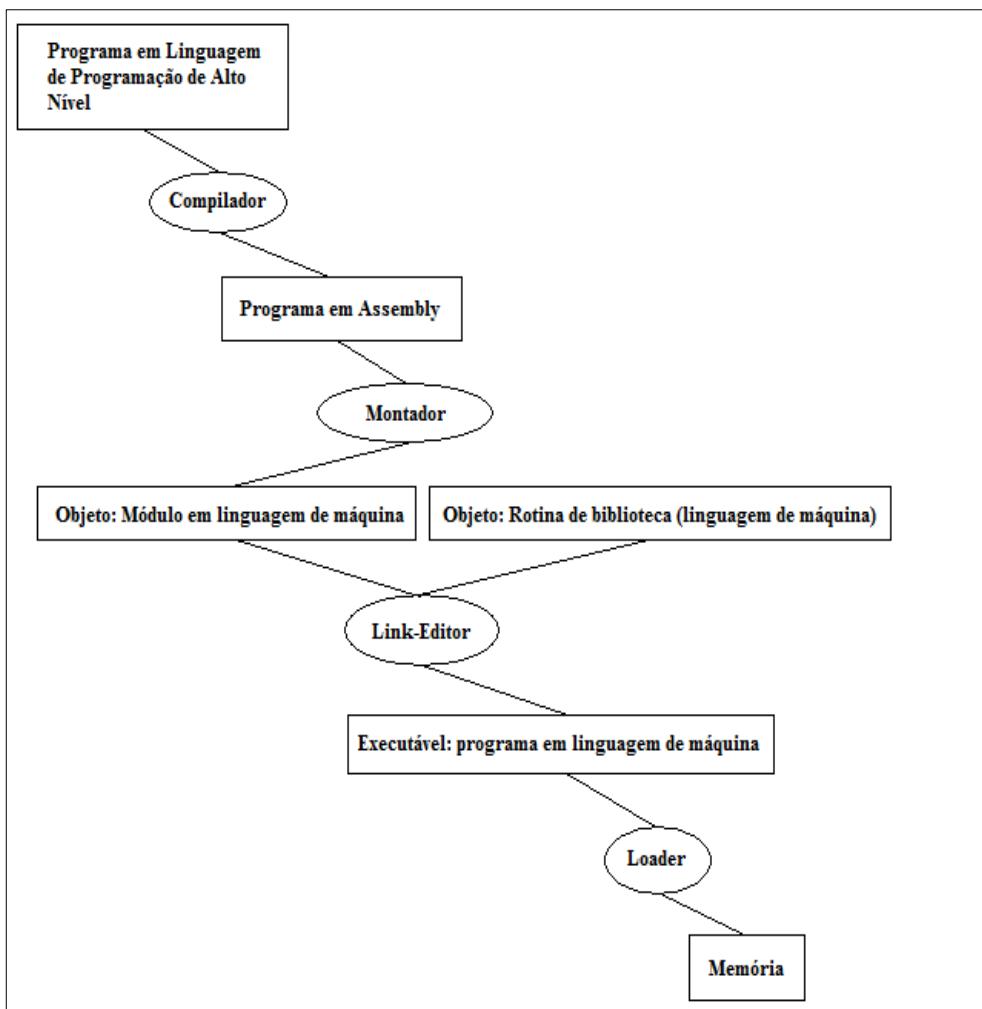
Um Tradutor de linguagem tem como objetivo, assim como um tradutor de línguas, traduzir uma linguagem *A* para uma linguagem *B*, *como exemplificado na Figura 7*. Um Interpretador pode ser considerado um tradutor, por exemplo, caso ele seja do tipo que utiliza uma linguagem intermediária; faz uma tradução da linguagem que interpreta para a intermediária. Um outro exemplo é o compilador, que da mesma maneira, traduz da linguagem a partir da qual compila para a linguagem Assembly e de forma semelhante o Montador realiza esta operação o decodifica para linguagem de máquina, ou seja, o código compilado.



**Figura 7:** Fluxo de execução de um programa tradutor de linguagens de programação  
Fonte: Próprio Autor

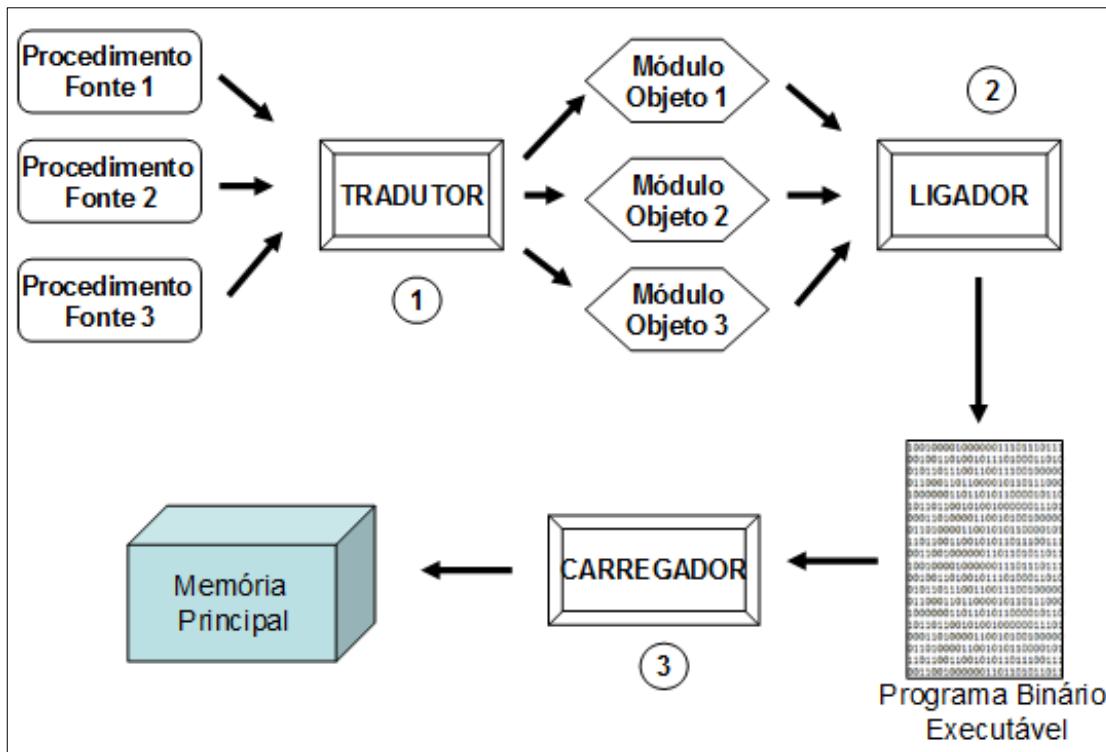
## 1.4 Compilador

Um Compilador é aquele que traduz um programa de uma linguagem de programação em **Assembly**, que é um conjunto de instruções que serão mapeadas para seu respectivo conjunto de comandos da linguagem de máquina, ou seja, é uma forma simbólica de como a máquina entende, conforme se observa nas Figuras 8 e 9, a seguir.



**Figura 8:** Árvore de responsabilidades  
Fonte: <http://www.inf.pucrs.br/manssour/LinguagemC/PoligC-Cap01.pdf>  
<http://www.inf.pucrs.br/~gustavo/disciplinas/pli/material/paradigmas-aula08.pdf>

Na Figura 8 acima é possível observar as responsabilidades de cada ferramenta e seus respectivos papéis na execução de um código feito em uma linguagem de programação de alto nível, indo desde o código no formato de texto até o seu carregamento, já transcrito em código de máquina, em memória para execução. Já na Figura 9 abaixo existe um fluxograma mais reduzido do papel do compilador na execução do programa, no caso ele foi identificado como tradutor.



**Figura 9:** Fluxo de funcionamento

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

Linguagens de alto nível, como C, utilizam bem menos código do que o Assembly, aumentando significativamente a produtividade do programador. Isso ocorre porque ele possui um número reduzido de instruções que precisam ser combinadas para realizar o que uma única instrução em código alto nível faz. Mas isso acarreta um problema de performance, pois, muitas vezes, o otimizador deixa passar uma solução em Assembly mais performática que um programador experiente facilmente iria detectar.

## 1.5 Assembler (Montador/Ligador)

O processo de tradução de um programa em linguagem de montagem, ou *Assembly*, para programa em linguagem de máquina é chamado de processo de montagem. Este procedimento 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, na qual um comando em linguagem de alto nível pode ser traduzido em vários comandos em linguagem de máquina[22,23,24].

## 1.6 Linker (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.

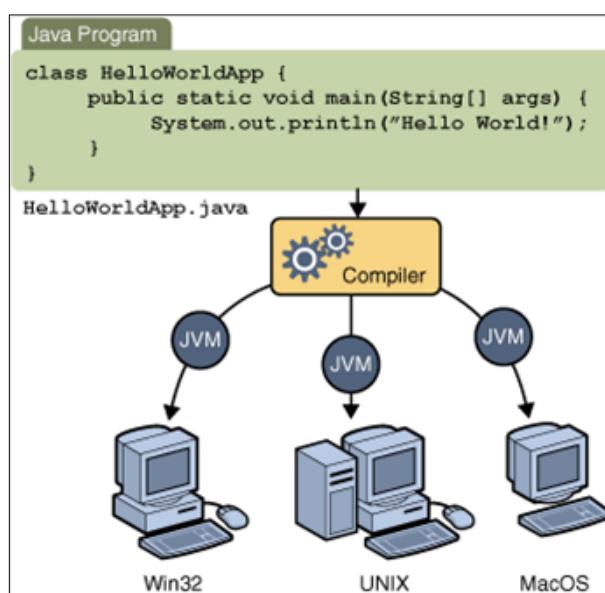
Caso o compilador ou o montador lesse um conjunto de procedimentos-fonte e produzisse diretamente um programa em linguagem de máquina, pronto para ser executado, bastaria que um único comando-fonte fosse alterado para que todos os procedimentos-fonte precisassem ser novamente traduzidos. Usando esta técnica, em que 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[22,23,24].

## 2 ARQUITETURAS RELACIONADAS

### 2.1 Java

Java é uma linguagem de programação de computadores de propósito geral, a qual é concorrente, orientada a objeto[25] e desenhada para ter o mínimo possível de dependências de implementação. Tem como objetivo ser uma linguagem que, uma vez codificada, possa ser executada em qualquer plataforma[26]. No caso, uma vez que o programa em *Java* foi compilado, ele pode ser executado em todas as plataformas que usam *Java*, sem a necessidade de recompilar ou de reescrever o código.[27]

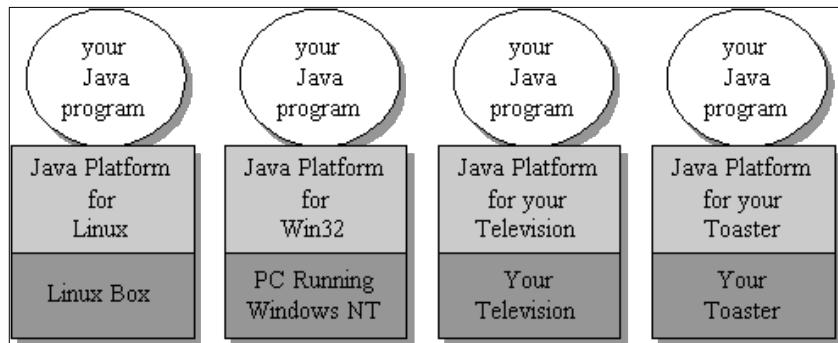
As aplicações *Java* são, geralmente, compiladas em um *bytecode*, que pode rodar em qualquer *Java Virtual Machine* (JVM), independente da arquitetura do computador, um exemplo de seu funcionamento pode ser visto na Figura 10. Apesar desse nível a mais em sua execução, *Java* continua sendo umas das linguagens de programação mais populares em uso.[28,29]



**Figura 10:** Exemplo de execução de um programa Java  
Fonte: <http://da2i.univ-lille1.fr/doc/tutorial-java/getStarted/intro/definition.html>

Esta linguagem foi, originalmente, desenvolvida por James Gosling, da Sun Microsystems (a qual foi comprada pela Oracle Corporation), e disponibilizada, em 1995, como componente principal da Sun Microsystems' *Java Platform*. A *Java Platform* é exclusiva de cada plataforma, mas para que o mesmo programa funcione

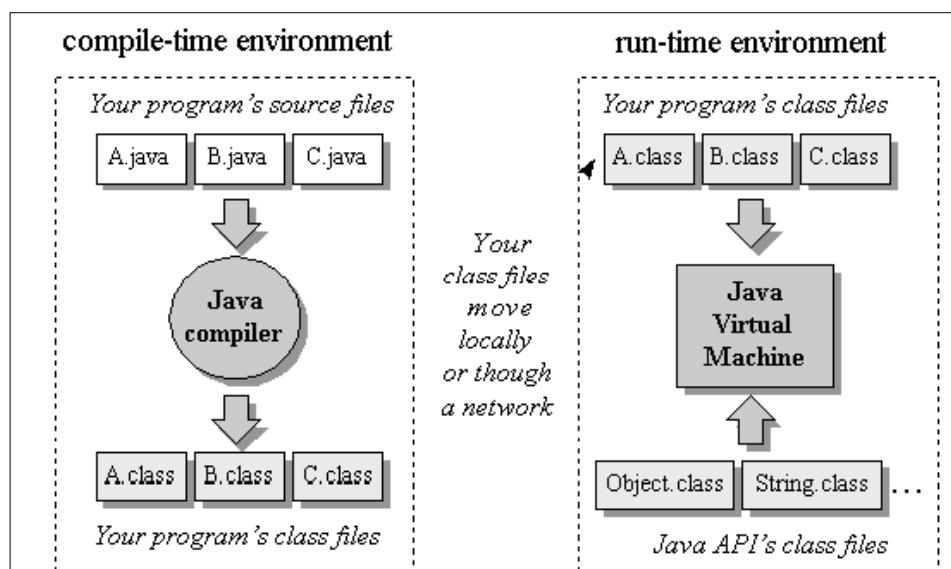
em todas as plataformas, suas implementações devem seguir a mesma especificação e isto pode ser visualizado na Figura 11. Ela possui muitas semelhanças de sintaxe com C e C++, no entanto não possui tantas facilidades e ferramentas *low-level*.[30,31]



**Figura 11:** Funcionamento do Java para cada plataforma

Fonte: <http://www.artima.com/insidejvm/ed2/introarch2.html>

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*, como mostrado na Figura 12.

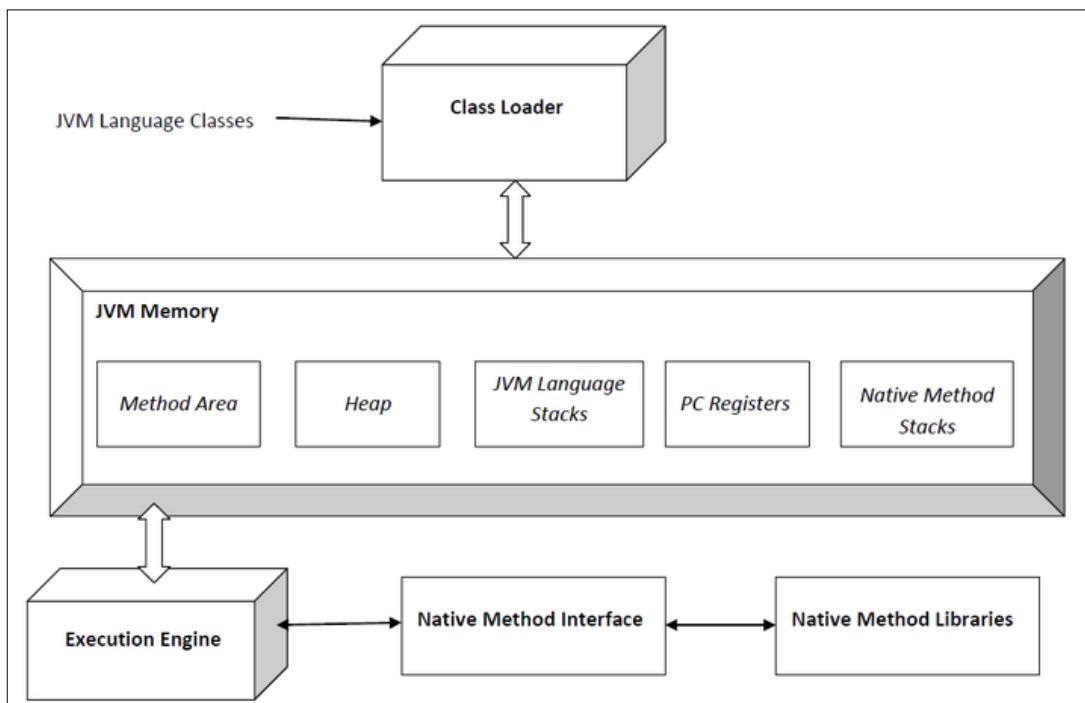


**Figura 12:** Fluxo de geração e execução de bytecode

Fonte: <http://www.artima.com/insidejvm/ed2/introarch2.html>

### 2.1.1 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: 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 todas as implementações são interoperáveis. A implementação da JVM é uma aplicação de computador que atenda a todos os requisitos da especificação da JVM. E, finalmente, uma instância da JVM é uma implementação rodando num processo que executa um programa de computador compilado em *Java bytecode*, como pode ser visualizado na Figura 13.



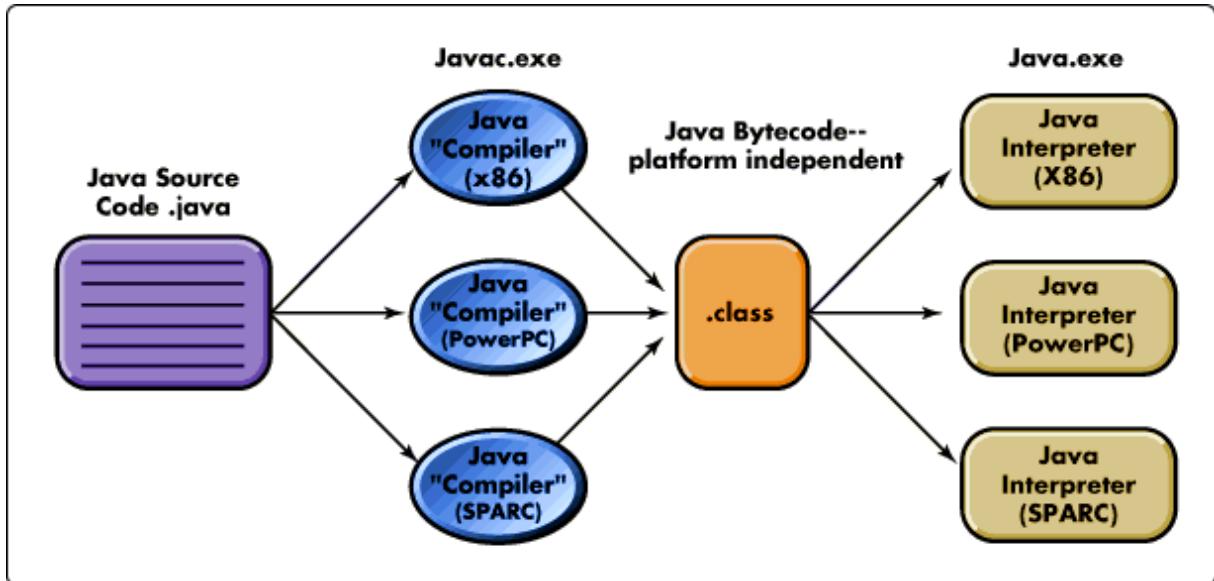
**Figura 13:** Instância da JVM

Fonte: <http://www.artima.com/insidejvm/ed2/index.html>  
<https://docs.oracle.com/javase/specs/jvms/se7/html/index.html>

### 2.1.2 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* da função são executados, ao invocá-la, o que acontece durante

a execução do programa. Eles podem ser executados por interpretação, compilação *just-in-time*, ou por qualquer outra forma desenvolvida pelo *designer* de uma JVM em particular, como mostrado na Figura 14.



**Figura 14:** Geração de um Java bytecode  
Fonte: <http://howwhywhat.in/what-is-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 ocorre no *Assembly*. Se mais uma informação for requerida antes que a JVM possa realizar alguma ação, tal informação será codificada em um ou mais operando que, como já discutido, vem 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 utilizados em sua totalidade. 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.[32,33,34,35]

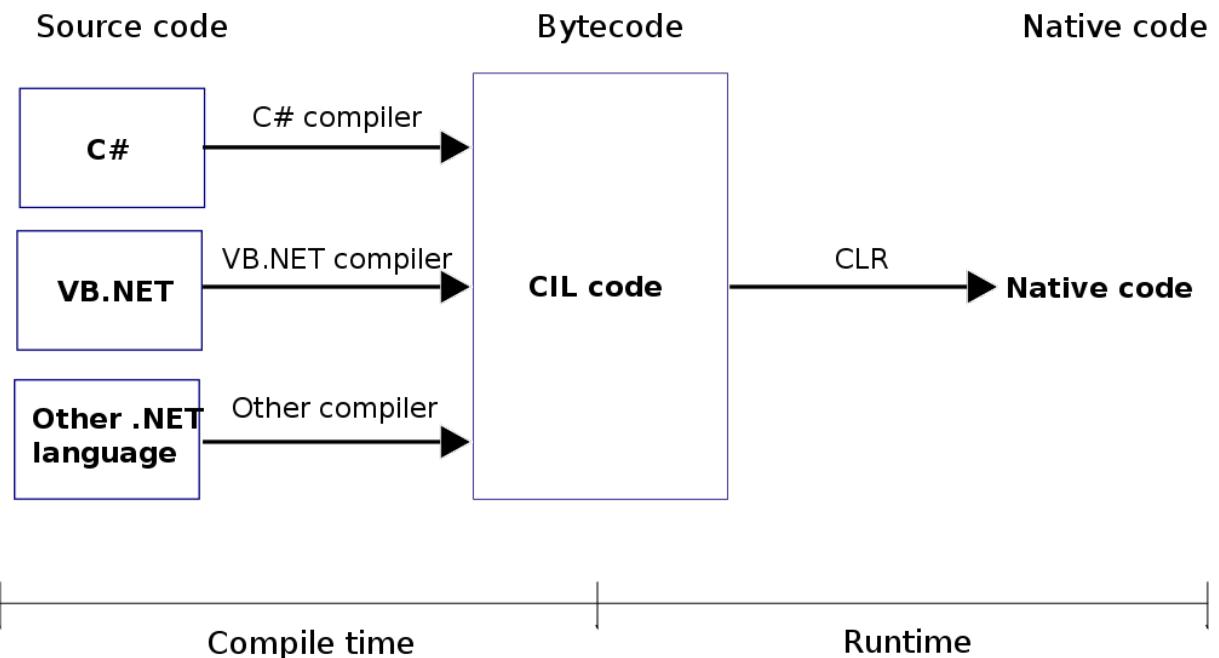
## 2.2 .NET

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

- Proporcionar um ambiente de programação orientada a objeto consistente para o código objeto, que é, armazenado e executado localmente, ou executado localmente, mas distribuído, ou até executado 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 de Scripted Environments, como também ambientes interpretados (Interpreted Environments).
- Tornar a experiência do desenvolvedor consistente em variados tipos de aplicações, tais como aplicativos baseados no Windows e aplicativos baseados na Web.
- Construir toda a comunicação nos padrões da indústria para garantir que o código baseado no .NET Framework possa se integrar com qualquer outro código.

O .NET Framework é composto pelo Common Language Runtime e pela 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 gera código em tempo de execução, fornecimento de serviços essenciais, tais como comunicação remota, gerenciamento de memória, de segmentos e, ao mesmo tempo faz cumprir estrita segurança de tipo e outras formas de precisão de 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 [36]. Este design pode ser visualizado na Figura 15 abaixo.



**Figura 15:** demonstração do funcionamento da estrutura .NET

Fonte: <http://www.bluesharktutorials.com/2013/11/understanding-common-language-runtime.html>

### 2.2.1 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 que expõem a funcionalidade do Common Language Runtime permitem a escrita de um código que se beneficia desta gestão de 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[37].

### *2.2.2 Common Language Infrastructure (CLI)*

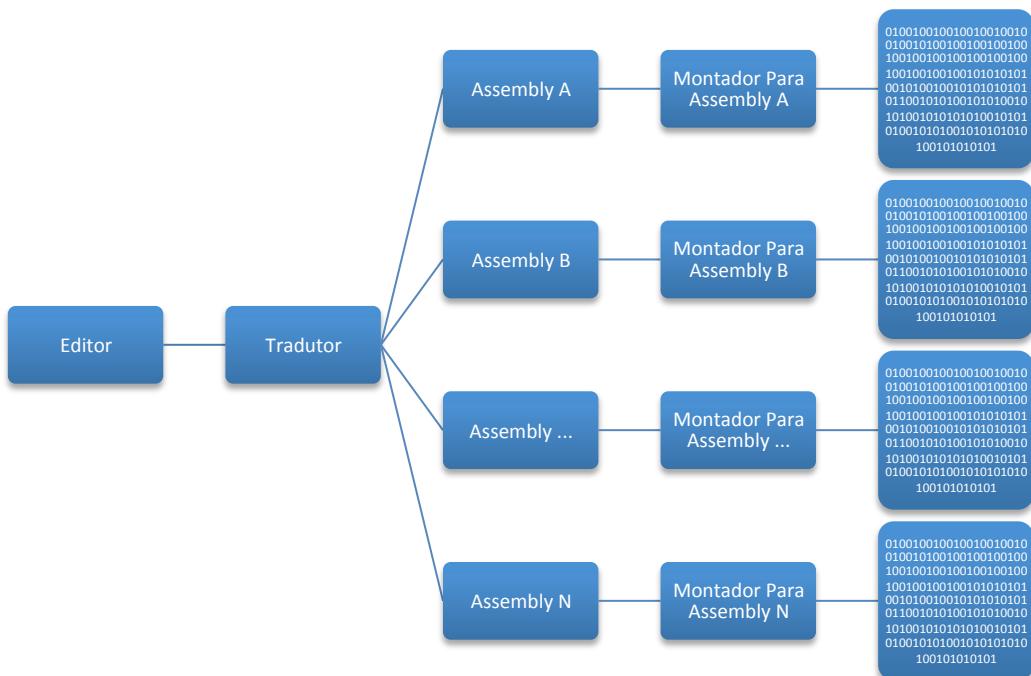
O Common Language Infrastructure (CLI) é uma especificação aberta desenvolvida pela Microsoft e padronizado pela ISO e ECMA que descreve o código executável e um ambiente de tempo de execução que permitem que várias linguagens de alto nível serem usadas em diferentes plataformas de computador sem ser reescrito para arquiteturas específicas. O .NET Framework, open source Mono e o Portable.NET são implementações da CLI [36,37].

### *2.2.3 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 [36,37].

### 3 ARQUITETURA

O sistema consiste em um tradutor de *Assembly genérico*, por meio do qual serão feitas as traduções para o *Assembly* específico de cada arquitetura, como mostrado na Figura 16. De forma semelhante ao *Java Bytecode*, porém, ao invés de executar numa máquina virtual ou interpretar o *Assembly*, ele será traduzido para o *Assembly* nativo da máquina em que será executado.



**Figura 16:** Fluxo de funcionamento de geração de código de máquina do sistema proposto

Fonte: Próprio autor

O sistema consiste em 3 níveis: editor de *Assembly* intermediário, tradutor do *Assembly* intermediário para nativo e montador nativo.

### 3.1 Editor

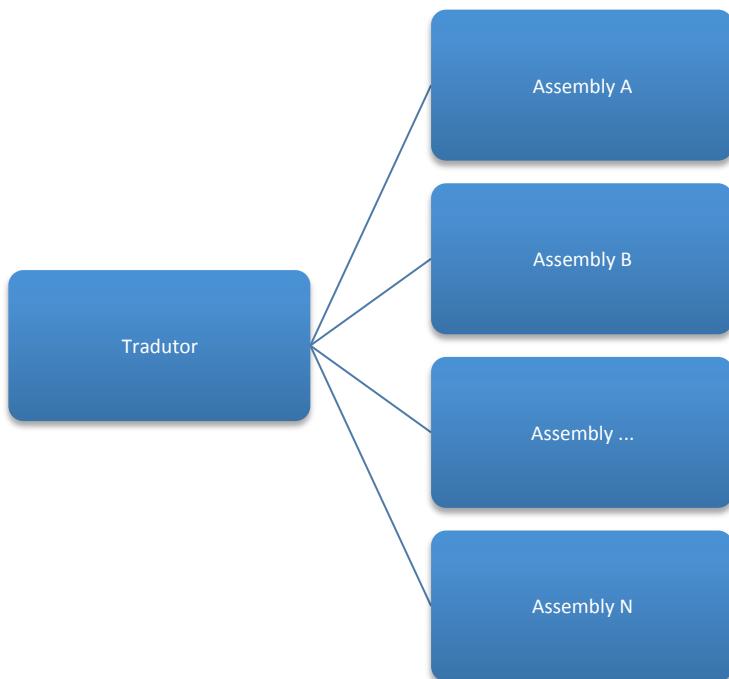
Trata-se de um editor de código, em que a linguagem para a qual se direciona é um *Assembly* Intermediário, neste caso, o *Easembly*. Este editor deve possuir *Intelligent Code Completion* para facilitar a implementação do código e fazer uma interpretação básica da linguagem para checagem de erros de sintaxe.

Para que o Editor possa fazer o Intelligent Code Completion ele necessitará da sintaxe da linguagem para qual se deseja fazer o uso da ferramenta. Então é necessário que o projeto tenha um arquivo de sintaxe, o qual pode ser implementado como um dicionário dentro do código ou num arquivo de configurações a parte.

Para editar o código é necessário que o sistema permita a inclusão, exclusão e armazenamento do código.

### 3.2 Tradutor

Este tradutor tem como função traduzir o *Assembly* Intermediário para o *Assembly* Nativo, no qual a partir de um arquivo de configurações conseguirá fazer a tradução para uma ou mais linhas da linguagem para a qual se deseja traduzir. Como mostrado na Figura 17 abaixo.



**Figura 17:** Fluxo de funcionamento de tradução do sistema proposto  
Fonte: Próprio autor

Para que a tradução seja realizada, o sistema requer um arquivo dicionário para cada *Assembly* que se deseja traduzir. A fim de que não exista inconsistências nas implementações desta arquitetura, foi especificado que o arquivo dicionário será um arquivo no formato JSON com a seguinte estrutura:

- O objeto JSON consistirá de um vetor de comandos, especificação de registradores, identificação de endereço, identificação de valores comuns, identificação de base numérica e outro de seleção de bit.
- Para cada comando é especificado o seu nome, um vetor de suas variações e se ele é complexo, ou seja, se este comando será traduzido em mais de uma instrução para o *Assembly* alvo.

- Cada variação do comando tem como propriedades: o numero de operandos e um vetor com as instruções em que serão traduzidas para o Assembly alvo.
- Cada Instrução tem como elementos de sua composição o mnemônico alvo, o vetor de operandos e o index de sua complexidade, ou seja a identificação do conjunto complexo de instruções a qual vai ser traduzido.
- Para cada operando é especificado o link para qual registrador Easembly este comando aponta, um booleano especificando se usa um registrador, um booleano especificando se este operando é um endereço e outro especificando o tipo de registrador
- Cada valor de especificação de registradores possui o seu respectivo nome em Easembly e um para o Assembly alvo. Caso ele possa ser visto de forma diferente para um mnemônico é possível colocar uma variação no mnemônico alvo.
- Para cada identificação de endereço, têm-se duas propriedades, uma para caso seja um operando registrador e outra para valor. Para cada uma destas existe um prefixo e um sufixo.
- Da mesma forma que a identificação de endereço possui seu sufixo e prefixo para registradores e valores, eles existem para cada identificação de valores comuns.
- De forma semelhante cada identificação de base numérica tem seu sufixo e prefixo para cada base numérica.
- E finalmente existe a forma de seleção de bit, onde se especifica por um booleano se o bit será localizada pelo inicio do valor ou ao final e uma string com o separador para identificação do bit.

O formato de arquivo JSON foi escolhido devido a sua simplicidade em relação a quantidade de tags e de informação necessária ao se comparar com o XML, bem como familiaridade.

### 3.3 Montador

Após a transcrição do código para o *Assembly* nativo, este deve ser montado para a linguagem de máquina relacionada ao *Assembly* transcreto. Para que isso aconteça de forma consistente é importante que o compilador seja o mesmo para todas as suas implementações. O compilador proposto para o sistema faz parte de um pacote Linux já existente no mercado: o SDCC, que possui vários montadores *Assemblies*, os quais serão utilizados para fazer a transcrição do código de *Assembly* gerado para o código de máquina da plataforma alvo.

O SDCC ou Small Device C Compiler é um compilador open source projetado para microprocessadores 8 bits, incluindo o microprocessador z80 e o microcontrolador 8051 utilizados nesse projeto. Ele pode ser utilizado por linha de comando, bem como seus montadores, possibilitando a sua utilização a partir de outro código, como o implementado neste trabalho. Além de mostrar erros de montagem, que são importantes para a validação do trabalho. No entanto só utilizados os montadores do z80 e do 8051 para esse projeto.

## 4 ESTUDO DE CASO

Este estudo de caso tem como objetivo fazer uma validação simples da arquitetura proposta, para isto foi implementado uma versão desta arquitetura e foram realizados alguns testes.

A arquitetura do sistema consiste em um servidor que utiliza PHP, tanto para realizar o processamento do *core* do sistema quanto para organizar e estruturar as páginas com o HTML5. No lado do cliente foi utilizado JavaScript, tanto para se comunicar com o servidor, quanto para realizar a parte mais complexa do sistema, o Editor de Código, incluindo também o roteamento dos módulos, que serão explicados com maior profundidade no decorrer deste capítulo, e tratamento de *cache* para acelerar o processamento/carregamento da página. E finalmente requererá que seus usuários sigam o padrão de nomenclatura “*Object Oriented Programming Languages Pattern*” descrito pelo projeto Open Source: “Pattern”[38].

### 4.1 *User Interface*

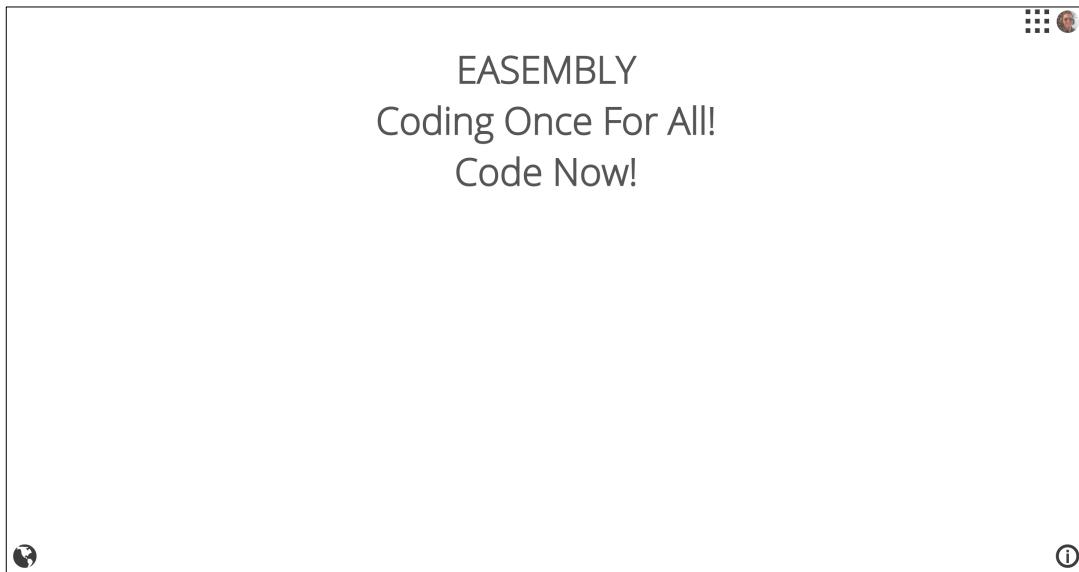
#### 4.1.1 *Single Page*

O *design* HTML/CSS foi um dos maiores empecilhos para este sistema, pois a experiência necessária em *design* HTML/CSS, para fazer um sistema como este, era mínimo ou inexistente. E à medida que se avança no *design*, ele se torna maior e mais complexo, o que para alguém sem experiência dificulta imensamente o trabalho. Uma medida que facilitou esta implementação foi a modularização da página feita a partir dos includes do PHP, o qual pode se fundir ao HTML.

##### 4.1.1.1 Modularização da página

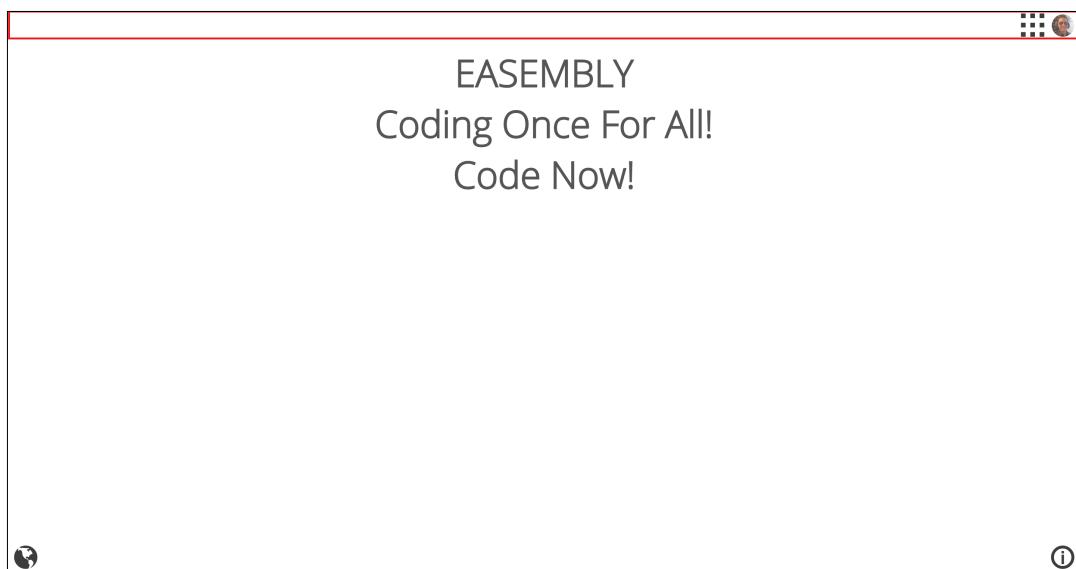
A única página acessada pelo usuário é a *index*, mostrada pela Figura 18. Esta página foi divida em módulos, que são arquivos no formato PHP/HTML, iniciais: *Head*, *Header*, *Body* e *Footer*, os quais, individualmente, carregarão outros módulos a partir de configurações enviadas para o servidor, a partir do *JavaScript* do cliente

e/ou configurações padrão, *Cache*, *Storage*, *IP address*, localização e língua do navegador (como no caso do sistema de tradução).



**Figura 18:** Exemplo da Página Index, ao utilizar o módulo de body contendo a Home, em sua versão inicial  
Fonte: Próprio autor

O *Head* é 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* é responsável pelo menu estático no topo da página como mostrado na Figura 19, cujas atribuições são a navegação entre as subpáginas carregadas no *Body*, e o *login/logout*.



**Figura 19:** Demonstração da área da página que é definida com o Header  
Fonte: Próprio autor

O *Body* é o local onde são carregadas as páginas em que o usuário navegará pelo *site*, além de ter a possibilidade de carregar uma janela, por cima dele mesmo, o qual pode ser visualizado na Figura 20.



**Figura 20:** Apresentação da área da página que é definida com o Body  
Fonte: Próprio autor

E, finalmente, há o Footer, com um menu estático no fim dā página, no qual se encontram dois botões que abrem suas respectivas janelas, um para tradução e outro para informações de contato e do *website*. *Este módulo pode ser visto na Figura 21.*



**Figura 21:** Imagem da área da página que é definida com o Footer  
Fonte: Próprio autor

#### 4.1.2 Cache/Storage manipulation

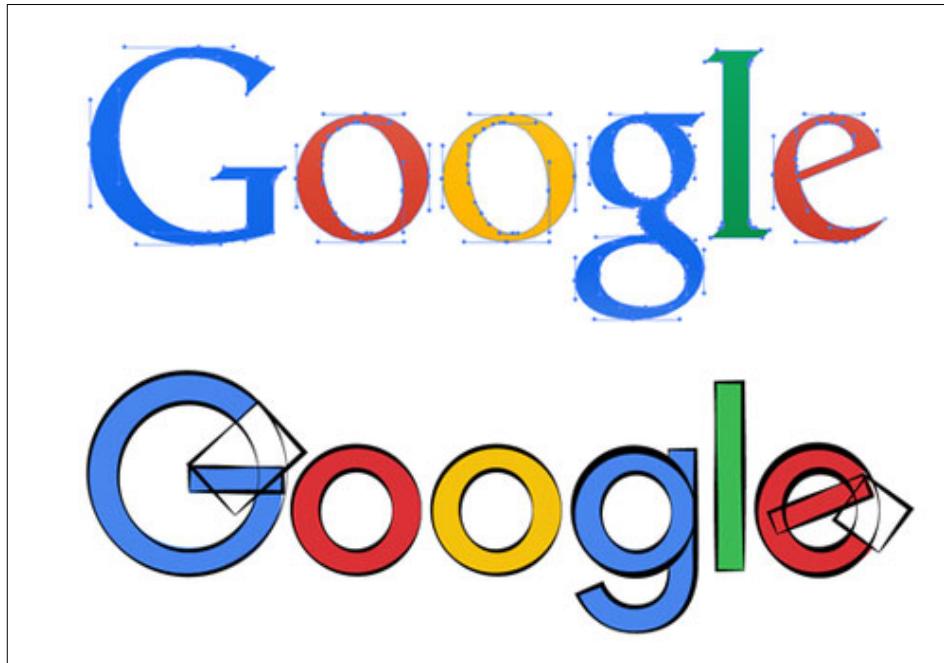
Para acelerar o carregamento da página foram utilizados *Cache* e *Web Storage*, a fim de que não existam requisições desnecessárias ao servidor. Ao carregar um módulo PHP/HTML, que para o cliente chega apenas como um trecho HTML, ele é salvo no *Cache* e no *Storage*. Antes de fazer uma requisição ao servidor ele checa se a informação já existe no *Cache* ou no *Storage* e, se ela existir, simplesmente a carrega no seu trecho específico, impedindo a existência de requisições desnecessárias ao servidor, e assim carrega as páginas mais rapidamente.

#### 4.1.3 Ausência de Imagens

O uso de imagens sempre foi um empecilho no tráfego de dados pela rede *WEB*, e elas apareciam ser imprescindíveis, à primeira vista, pois são os símbolos digitalizados que mais se aproximam do mundo real. Isso as torna importantes para a realização de uma *interface* intuitiva, afinal, os botões, por exemplo, são uma reprodução de um elemento físico real muito conhecido pela humanidade, o botão.

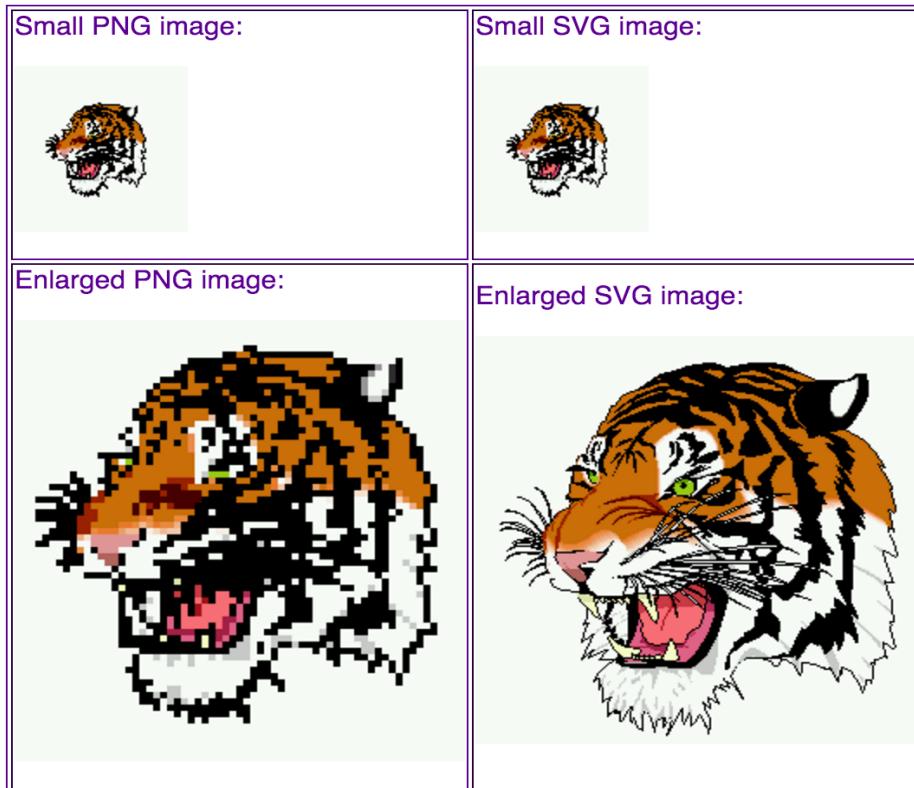
Mas as imagens apresentam uma característica que pode transformá-las em um estorvo, o tamanho. Inicialmente, eram apenas uma matriz de cores, agora existem diversos formatos, que incluem, inclusive, compressão, que, às vezes, resultam até em perda de qualidade. Mesmo com todas as inovações para que elas deixem de ser um obstáculo, ainda não são suficientes, em muitos casos. Há pouco tempo o *Google* realizou uma transformação do seu *logo*, sendo o tamanho uma de suas preocupações, que saltou, em torno, de 14 *kbytes* para 305 *bytes*, resultando em um carregamento mais rápido, o que é muito importante num *site* como ele, muito utilizado[39], como mostrado na Figura 22.

Mesmo assim, a imagem ainda tem um problema difícil de ser contornado, que é permanecer com uma resolução agradável a todos os dispositivos, uma vez que cada um tem resolução e tamanho físico diferentes, não adiantando apenas dar um *Zoom* na imagem, pois esta perderá sua qualidade. Então, como resolver um problema de tamanho sem ter uma equipe de *design* especializada? Como resolver o problema de imagens diferentes, para diferentes dispositivos?



**Figura 22:** Demonstração da evolução da logomarca da Google  
Fonte: <https://design.google.com/articles/evolving-the-google-identity/>

Existem, então, formas pouco utilizadas pelos sistemas da atualidade, as imagens vetorizadas, que não utilizam muito espaço em disco e não perdem qualidade ao serem redimensionadas, como SVG [40], visualizado na Figura 23. No entanto é difícil encontrar um editor que tenha esse tipo de suporte, e que possa ser reproduzido por *browsers* e sistemas. Então, foi seguido um caminho menos ortodoxo, utilizando-se fontes de texto. Os processadores de fontes são altamente performáticos, fontes ocupam um espaço mínimo em disco e possuem um grande conjunto de símbolos. Cada um destes símbolos pode substituir uma imagem, sem contar que não perdem qualidade ao serem redimensionados, e possuem um grande suporte.



**Figura 23:** Diferença entre imagens vetorizadas e não vetorizadas, em tamanho original e ampliadas. No caso, imagens do tipo PNG e SVG  
Fonte: <http://www.w3.org/TR/SVG-access/>

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.

Enfim este recurso teve como objetivo diminuir o tráfego de bytes para servidor, permitindo que ele possa focar seus recursos computacionais em tarefas mais importantes como a tradução do Easembly e na compilação. Além de tornar o site mais leve, rápido de ser carregado e de utilizar seus recursos.

#### 4.1.4 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, idioma do browser ou mesmo seleção do usuário, e busca no sistema de arquivo se existe o arquivo para aquele módulo e para aquela linguagem. Caso exista, ele abre o arquivo de linguagem e procura a variável requisitada e, por fim, retorna seu valor. Caso contrário, ele retorna para o idioma *default*, que é inglês.

O arquivo de idioma foi criado para ser o mais simples possível, para que um tradutor leigo possa fazer um novo arquivo para uma nova linguagem, sem precisar entender de *tags* complexas de um tipo de arquivo específico como XML ou JSON, um exemplo deste aquivo pode ser visualizado na Figura 24.

```
projectTitle=\0" Título do Projeto"\0;
codeTitle=\0" Título do Código"\0;
run=\0"Run"\0;
download=\0"Download"\0;
openFile=\0" Abrir Arquivo"\0;
saveFile=\0" Salvar Arquivo"\0;
newFile=\0" Novo Arquivo"\0;
deleteFile=\0" Deletar Arquivo"\0;
```

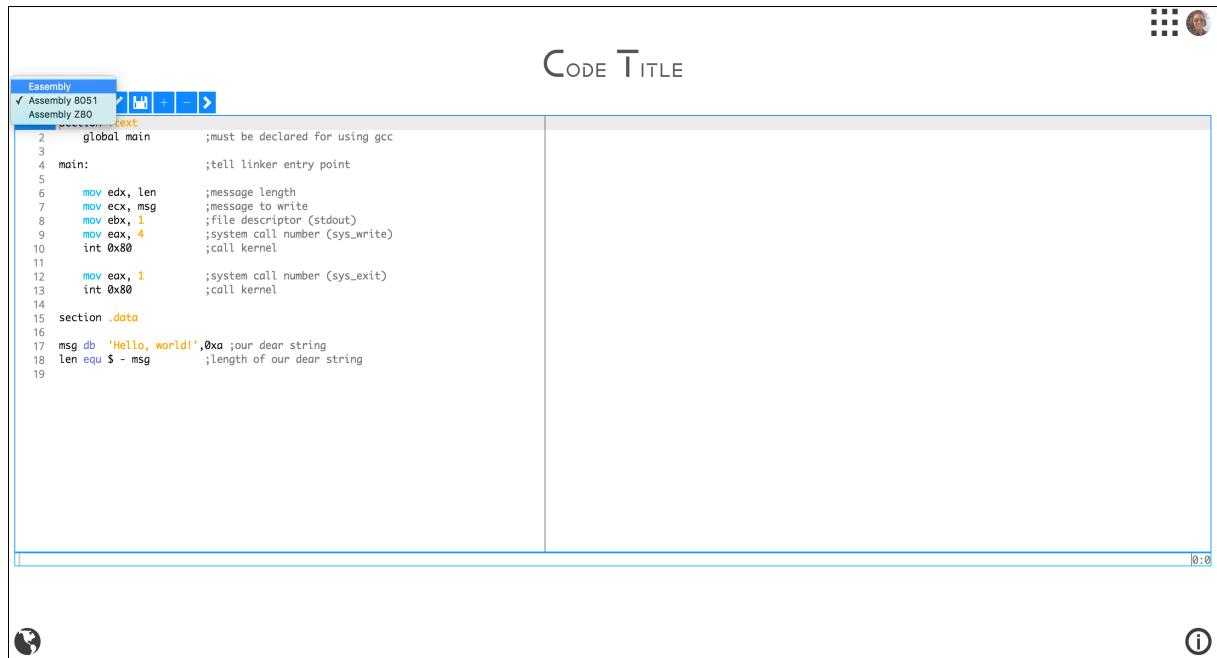
**Figura 24:** Exemplo de um arquivo de linguagem

Fonte: Próprio autor

#### 4.1.5 Editor

O Editor do sistema implementado possui diversas ferramentas para facilitar a codificação, como pesquisa, refatoração de nome de variáveis, Intelligent Code Completion e várias outras ferramentas. Para 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 utilizar linguagens não suportadas por este editor, foram necessárias 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 ao *Design do site*.

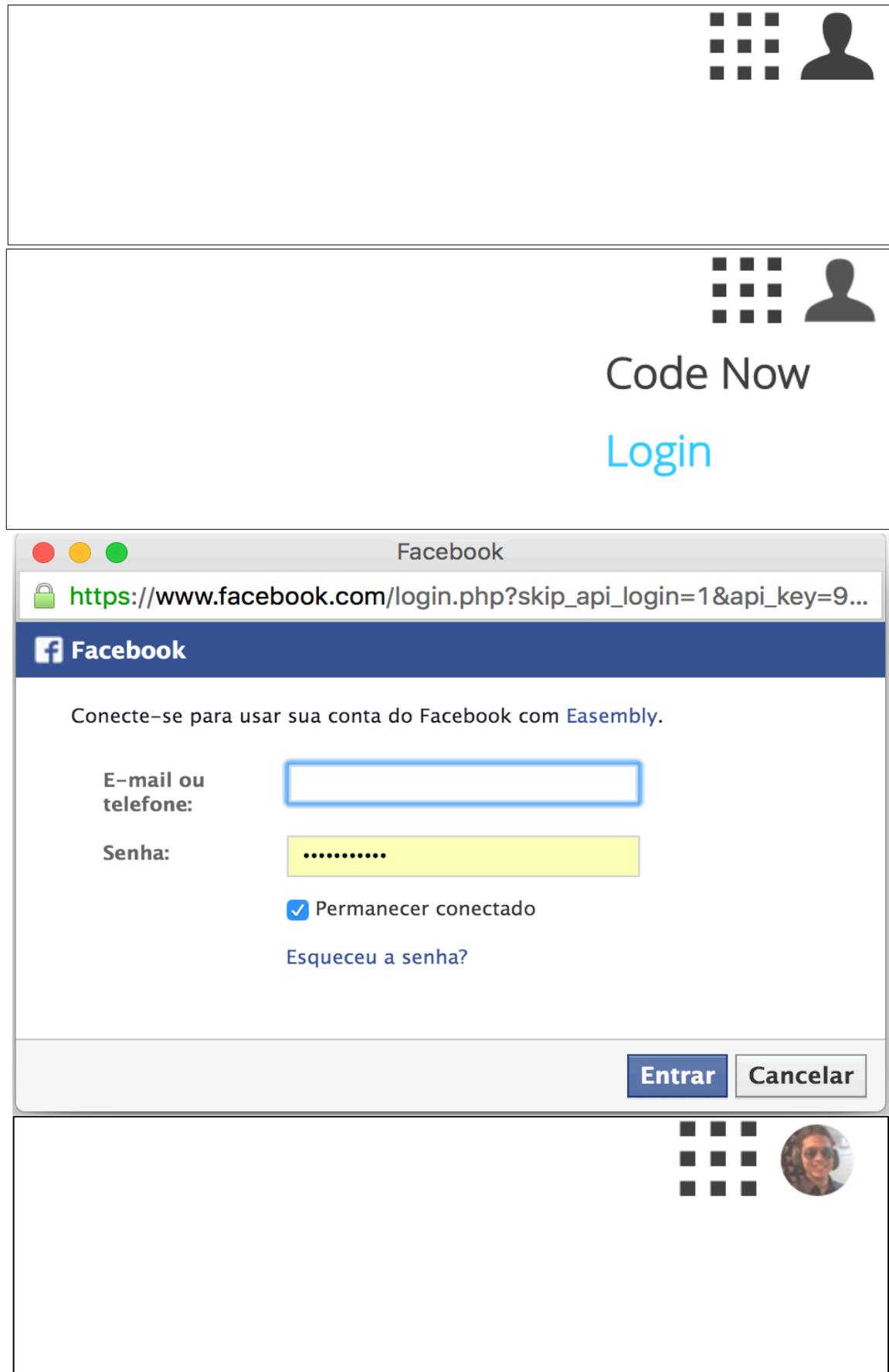
Neste caso, ele pode ser usado para Codificar para *Easembly*, *Assembly* do Z80 e *Assembly* do 8051 como mostrado na Figura 25.



**Figura 25:** Editor de código e suas possibilidades de linguagens para Edição  
Fonte: Próprio autor

#### 4.1.6 Autenticação/Login

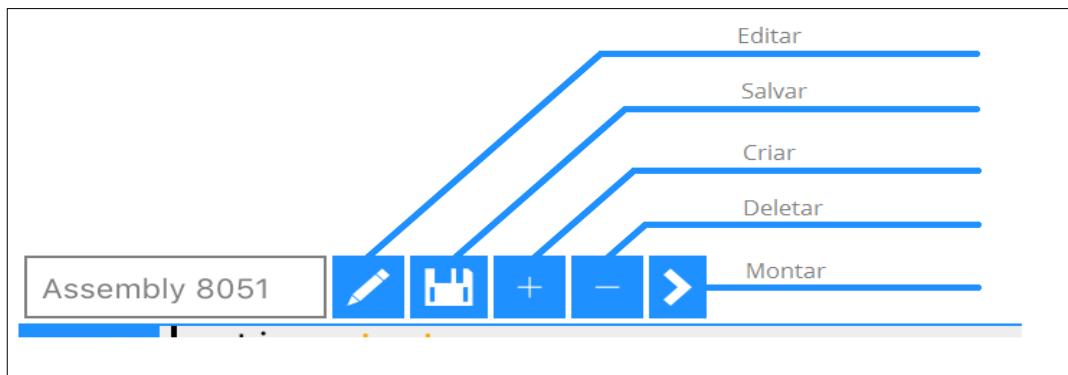
O *Login* e a Autenticação do usuário são feitas pela API do *Facebook*, eliminando assim a preocupação com o registro, armazenamento e segurança dos dados do usuário, o seu fluxo de login básico pode ser visualizado na Figura 26.



**Figura 26:** Demonstração do fluxo de login  
Fonte: Próprio autor

#### 4.1.7 CRUD

A UI permite que o usuário crie, pesquise, remova e edite, como a Figura 27 mostra, arquivos *Assembly* de 8051, Z80 e o *Easembly*, que pode ser visualizado nas Figuras 27, 28 e 29).



**Figura 27:** Indicação do funcionamento de cada botão

Fonte: Próprio autor

A captura de tela mostra a interface de usuário do OpenFile. No topo, há uma barra com ícones para fechar, minimizar e maximizar. O título da janela é "OPENFILE". Na parte superior esquerda, uma barra de menu mostra "Assembly 8051" e "Z80". O conteúdo principal é uma área de texto com o seguinte código Assembly:

```

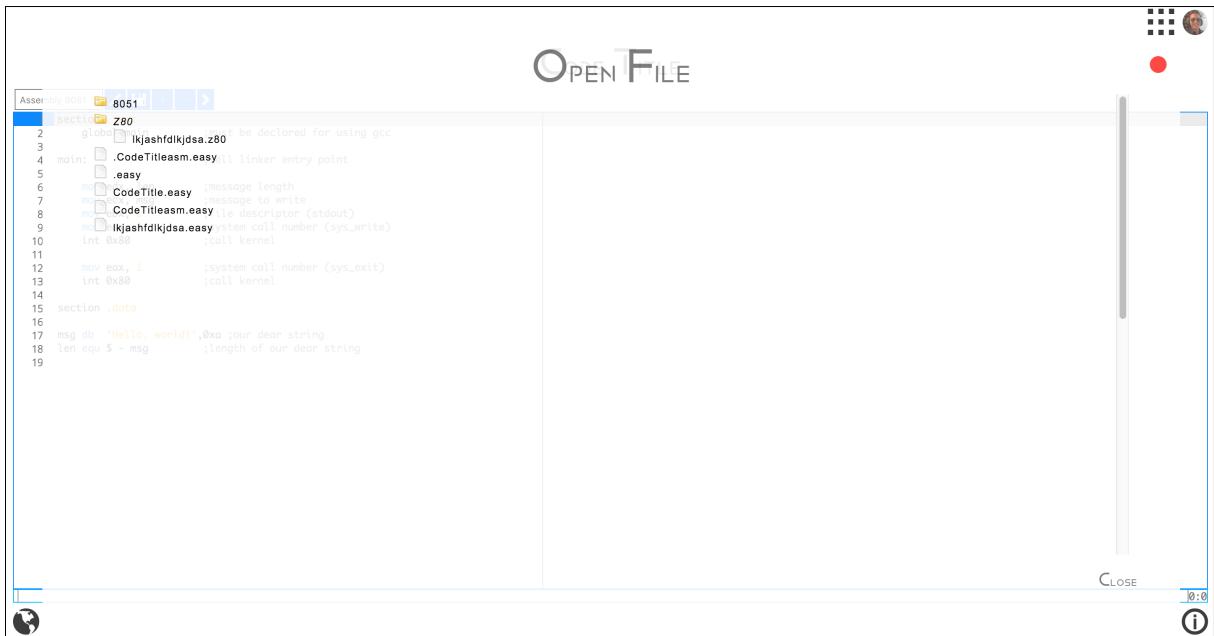
1    section .text
2    global _start
3    _start:
4        mov r0, #1
5        ldi r1, #0x00
6        ldi r2, #0x00
7        ldi r3, #0x00
8        ldi r4, #0x00
9        ldi r5, #0x00
10       ldi r6, #0x00
11       ldi r7, #0x00
12       ldi r8, #0x00
13       ldi r9, #0x00
14       ldi r10, #0x00
15       ldi r11, #0x00
16       ldi r12, #0x00
17       ldi r13, #0x00
18       ldi r14, #0x00
19

```

No lado direito da janela, há uma barra vertical de rolagem. No rodapé, há botões para "CLOSE" e "INFO".

**Figura 28:** Abrir Arquivo (Pesquisa) com pastas fechadas

Fonte: Próprio autor



**Figura 29:** Abrir Arquivo (Pesquisa) com uma pasta aberta  
Fonte: Próprio autor

## 4.2 Core

### 4.2.1 Tradutor

O Tradutor de *Easembly* para outro *Assembly* funciona a partir do momento em 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 em arquivo texto no formato *.easy* e, então, lido. A fase seguinte é a tradução.

Para cada *Assembly* requisitado para a realização de uma tradução, é necessário um dicionário para dizer ao tradutor como fazer tal procedimento. Este dicionário utiliza um arquivo JSON como mostrado na Figura 30, que contém a instrução, ou instruções, respectiva no *Assembly* destino para um comando 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).

<pre>{   "command": [     {       "name": "copyToFirstWithFirstWithSecond",       "complexType": false,       "variation": [         {           "numberofOperands": 2,           "command": [             {               "mnemonic": "mov",               "complexType": 0,               "operand": [                 {                   "link": "rA",                   "isLink": true                 },                 {                   "link": "rB",                   "isLink": true                 }               ]             },             {               "numberofOperands": 2,               "command": [                 {                   "mnemonic": "mov",                   "complexType": 0,                   "operand": [                     {                       "link": "rA",                       "isLink": true                     },                     {                       "link": "rB",                       "isLink": true,                       "isAddress": true                     }                   ]                 }               ]             }           ]         }       ]     }   ] }</pre>	<pre>,   {     "link": "rB",     "isLink": true   } ] }, {   "numberofOperands": 2,   "command": [     {       "mnemonic": "mov",       "complexType": 0,       "operand": [         {           "link": "rA",           "isLink": true         },         {           "link": "rB",           "isLink": true,           "isAddress": true         }       ]     }   ] }, {   "pointer": {     "register": {       "before": "@"     },     "other": {     }   },   "regular": {     "register": {     },     "other": {       "before": "#"     }   },   "number": {     "binary": {       "before": "0b"     },     "decimal": {       "before": "..."     },     "hexadecimal": {       "before": "0x"     }   } }</pre>
<pre>,   "operand": [     {       "Easembly": "rA",       "Assembly": "A"     },     {       "Easembly": "rB",       "Assembly": "B"     },     {       "Easembly": "r0",       "Assembly": "R0"     },     {       "Easembly": "r1",       "Assembly": "R1"     },     {       "Easembly": "r2",       "Assembly": "R2"     },     {       "Easembly": "r3",       "Assembly": "R3"     },     {       "Easembly": "r4",       "Assembly": "R4"     }   ] }</pre>	

**Figura 30:** Exemplo de dicionário JSON, mostrando suas principais propriedades  
Fonte: Próprio autor

Com o dicionário é possível fazer a tradução e, então, armazenar o arquivo *Assembly* resultante da tradução.

#### *4.2.2 Compilador*

O sistema optou por utilizar o compilador SDCC de código aberto, o qual foi descrito anteriormente na arquitetura, que pode ser utilizado via linha de comando, a qual é executada após a tradução ser realizada, se o usuário tiver clicado no botão para compilar.

Este compilador foi escolhido, pois pode fazer a montagem dos dois *Assemblies* selecionados para o projeto das arquiteturas dos processadores Z80 e 8051, além de diversos outras arquitetura que o Easembly possa no futuro traduzir.

## 5 ANÁLISE DOS RESULTADOS

A análise a seguir será feita de forma simplificada, pois o estudo apenas 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.

O primeiro programa teste tem como objetivo testar o uso da instrução de movimentação de dados, envolvendo todas suas variações básicas, além de testar todas as possibilidades de usar um comando Easembly e testar um comando complexo, ou seja um comando que se traduz em mais de uma instrução no Assembly alvo. Este programa e o resultado de suas traduções podem ser visualizados na Figura 31. E foi possível notar que o resultado da tradução foi coerente e dentro do esperado. Ao final estes programas foram montados pelo SDCC, sem acusar nenhum erro de compilação, indicando que o programa transscrito é montável.

Easembly	8051	Z80
<b>Fuc:</b>	<b>Fuc:</b>	<b>Fuc:</b>
copyToFirstWithFirstWithSecond(rA,rB);	mov A,B	ld A,B
copyToFirstWithFirstWithSecond(rA,rB) ;	mov A,B	ld A,B
<b>Of:</b>	Of:	Of:
copyToFirstWithFirstWithSecond rA,@hex10 ;	mov A,0x10	ld A,\$10
copyToFirstWithFirstWithSecond rA,@rB;	mov A,@B	ld A,(B)
<b>Of:</b>	Of:	Of:
copyToFirstWithFirstWithSecond(rA,hex1);	mov A,#0x1	ld A,\$1
sumToFirstWithFirstWithSecond(rA,rB);	add A,B	add A,B
subToFirstWithFirstWithSecond(rA,rB);	subb A,B	sub A,B
jumpTo(0f);	ljmp Of	jp Of
wait4Cycles();	nop	nop
	nop	
	nop	
	nop	

**Figura 31:** Imagem do resultado da tradução de um código Easembly para os Assemblies do 8051 e do z80  
Fonte: Próprio autor

A imagem acima mostra um código *Easembly* e o resultado de sua tradução para os *Assemblies* dos processadores 8051 e Z80, respectivamente. É possível notar que o *Easembly* ficará 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 suportará *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 gerado a partir o *Easembly* atual não poderia.

O segundo programa tem como objetivo gerar um programa de *delay* na ordem de 40 ciclos de *clock*, verificar a variação de ciclos gerado entre os códigos gerados além de testar loop. Os ciclos de cada instrução podem ser vistos na Figura 32, abaixo. Foi possível notar que estes programas transcritos permanecem coerentes depois da tradução.

Easembly	Z80	8051
copyToFirstWithFirstWithSecond(r0,dec10);	ld D,#10 2 cycles	mov R0,#10 1 cycle
copyToFirstWithFirstWithSecond(rB,r0);	ld B,D 2 cycles	mov B,R0 1 cycle
KeepWaiting:	KeepWaiting:	KeepWaiting:
wait4Cycles();	nop 4 cycles	nop 1 cycle
decrementRBIfRBIsNotEqualToZeroJumpTo(KeepWaiting);	djnz KeepWaiting 2-3 cycles (B->b==0, 3->B!=0)	nop 1 cycle nop 1 cycle nop 1 cycle nop 1 cycle djnz B, KeepWaiting 2 cycles

**Figura 32:** Imagem do resultado da tradução de um código de Delay Easembly para os Assemblies do 8051 e do z80  
Fonte: Próprio autor

O terceiro programa é um programa para testar operações lógico-aritméticas, no caso foi feito um programa para multiplicar dois valores. E em seguida foram feitos programas manualmente para verificar a perda de performance do código gerado. O código feito em Z80 manualmente não houve muita diferença em relação ao código gerado, então foi omitido, mas o código feito para 8051 teve uma diferença considerável. O código manual, como permite utilizar todas as instruções, utiliza apenas 6 ciclos enquanto o gerado pelo sistema é da ordem de n, n sendo o número de vezes que o número será somado repetidas vezes. A razão pela qual o programa foi feito desta maneira, ocorreu porque o sistema ignora a endianness dos processadores. O resultado deste teste pode ser visualizado na Figura 33. Mesmo a complexidade deste programa sendo mais complexa o resultado de sua tradução também demonstrou-se correta.

Easembly	8051	8051(manual)	z80
<pre> Start: copyToFirstWithFirstWithSecond(rA,dec52); copyToFirstWithFirstWithSecond(rB,dec10); copyToFirstWithFirstWithSecond(@hex10,rB); compareRAWithFirstIfNotEqualJumpToSecond(@hex10,NotEqual); Equal: Greater: subToFirstWithFirstWithSecond(rA,rB); sumToFirstWithFirstWithSecond(r0,dec1); copyToFirstWithFirstWithSecond(rB,dec10); jumpTo(Start); NotEqual: jumpIfCarryIsSetTo(Less); jumpTo(Greater); </pre>	<pre> Start: mov A,#52      1 cycle mov B,#10      1 cycle mov 0x10,B      1 cycle div AB ; 4 cycles cjne A, 0x10,NotEqual 2 cycles  Equal: Greater: subb A,B      1 cycle add R0,#1      1 cycle mov B,#10      1 cycle ljmp Start      2 cycles  NotEqual: jc Less      2 cycles ljmp Greater  2 cycles </pre>	<pre> Start: mov A,#52 ; 1 cycle mov B,#10 ; 1 cycle div AB ; 4 cycles cjne A, 0x10,NotEqual 2 cycles  Equal: Greater: sub A,B      1 cycle add D,#1      1 cycle ld B,#10      2 cycles jp Start      3 cycles  NotEqual: jc C, Less    3 cycles jp Greater    3 cycles </pre>	<pre> Start: ld A,#52      2 cycles ld B,#10      2 cycles ld 0x10,B      2 cycles cp 0x10      2 cycles jp NZ, 0x10 3 cycles  Equal: Greater: sub A,B      1 cycle add D,#1      1 cycle ld B,#10      2 cycles jp Start      3 cycles  NotEqual: jc C, Less    3 cycles jp Greater    3 cycles </pre>

**Figura 33:** Imagem do resultado da tradução de um código de Multiplicação Easembly para os Assemblies do 8051 e do z80 e sua versão feita manualmente para 8051  
Fonte: Próprio autor

O quarto programa foi um programa de divisão de um valor por outro, seguindo estratégia semelhante ao programa de multiplicação. E novamente tem-se uma diferença considerável entre o programa manual e o gerado pelo Easembly. O resultado da tradução pode ser visualizado na figura 34 abaixo. Este programa de forma semelhante aos outros acima, também obteve sucesso ao traduzir para as plataformas alvo de forma correta.

Easembly	8051	8051(manual)	z80
<pre> Start: copyToFirstWithFirstWithSecond(rA,dec52); copyToFirstWithFirstWithSecond(rB,dec10); copyToFirstWithFirstWithSecond(@hex10,rB); compareRAWithFirstIfNotEqualJumpToSecond(@hex10,NotEqual); Equal: Greater: subToFirstWithFirstWithSecond(rA,rB); sumToFirstWithFirstWithSecond(r0,dec1); copyToFirstWithFirstWithSecond(rB,dec10); jumpTo(Start); NotEqual: jumpIfCarryIsSetTo(Less); jumpTo(Greater); </pre>	<pre> Start: mov A,#52      1 cycle mov B,#10      1 cycle mov 0x10,B      1 cycle div AB ; 4 cycles cjne A, 0x10,NotEqual 2 cycles  Equal: Greater: subb A,B      1 cycle add R0,#1      1 cycle mov B,#10      1 cycle ljmp Start      2 cycles  NotEqual: jc Less      2 cycles ljmp Greater  2 cycles </pre>	<pre> Start: mov A,#52 ; 1 cycle mov B,#10 ; 1 cycle div AB ; 4 cycles cjne A, 0x10,NotEqual 2 cycles  Equal: Greater: sub A,B      1 cycle add D,#1      1 cycle ld B,#10      2 cycles jp Start      3 cycles  NotEqual: jc C, Less    3 cycles jp Greater    3 cycles </pre>	<pre> Start: ld A,#52      2 cycles ld B,#10      2 cycles ld 0x10,B      2 cycles cp 0x10      2 cycles jp NZ, 0x10 3 cycles  Equal: Greater: sub A,B      1 cycle add D,#1      1 cycle ld B,#10      2 cycles jp Start      3 cycles  NotEqual: jc C, Less    3 cycles jp Greater    3 cycles </pre>

**Figura 34:** Imagem do resultado da tradução de um código de Divisão Easembly para os Assemblies do 8051 e do z80 e sua versão feita manualmente para 8051  
Fonte: Próprio autor

Como o *Easembly* tem um conjunto de instruções menor que o conjunto das plataformas que gera, ele perde no quesito performance, pois existem instruções em ambas as arquiteturas que podem fazer em apenas um comando, o que o *Easembly* faz utilizando várias instruções. O *Easembly* poderia ter um conjunto maior que as arquiteturas destino atuais, incluir todas as instruções específicas e, assim, sanar o

problema de desempenho. Caso isso fosse feito, outro problema apareceria. Como instruções específicas podem não existir em uma ou mais das plataformas alvo, isso implicaria em uma sequência de comandos para executar a operação desejada. Isto significa que uma instrução exclusiva de uma plataforma, nas demais arquiteturas, deverá ser composta por uma sequência de instruções, para ter o mesmo resultado. Esta sequência de comandos poderá implicar no uso de registradores. Para implementar esta ideia, existem algumas possibilidades, na hipótese de não existir nenhum registrador a mais que o *Easembly*, o sistema teria que, de alguma forma, saber se existe algum registrador que não tenha sido usado ou usar a pilha, caso não esteja cheia. Na possibilidade de ter algum registrador a mais, ou que sejam definidos um ou mais registradores exclusivos, é possível utilizá-los para contornar este problema.

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

Enfim, o *Easembly* sempre ficará restrinido, de alguma forma, pelos processadores mais “simples”. Mas é importante ressaltar que o sistema obteve sucesso em traduzir todos os testes feitos com a lógica correta, além de suas transcrições serem livres de erros de montagem, pelo menos para os montadores do pacote SDCC.

## CONCLUSÃO

A partir da análise dos resultados obtidos foi possível perceber que a arquitetura permite a união dos Assemblies de diversos processadores. No entanto ficará restrinido, de alguma forma, pelos processadores mais básicos.

Então, a arquitetura proposta mostrou-se viável, o que o torna de alguma forma útil, mas, provavelmente, haverá uma queda na performance de seus programas, à medida que mais processadores forem se integrando à plataforma. Para solucionar este problema, sugere-se um trabalho futuro, cujo objetivo é ter uma arquitetura mais flexível para que as plataformas superiores não sejam tão mal aproveitadas devido ao rebaixamento das inferiores.

O trabalho futuro aqui proposto aproveitará a ideia do Easembly, no entanto o programador selecionará para quais plataformas deseja programar, e o sistema criará um set de instruções para aquele conjunto de processadores. Dessa forma as plataformas serão melhor aproveitadas, já que o conjunto de instruções semelhantes será, provavelmente maior, em virtude do conjunto de processadores selecionados pelo usuário ser reduzido. Inclusive, o número de registradores seria, presumivelmente similar, pois o programador selecionaria processadores compatíveis com o projeto que ele desejasse realizar, indicando uma provável semelhança.

## 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](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](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](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: Washington DC, Abr. 12 2010. Disponível em: <<http://dl.acm.org/citation.cfm?id=1848870>>. 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: Washington DC, Jun. 20 2014. Disponível em: <<http://www.jatit.org/volumes/Vol64No2/20Vol64No2.pdf>>. Acesso em: 26 maio 2015.

[15] ARBONE, C. ***Model-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] Source Code Editor. PCMag. Disponível em:< <http://www.pcmag.com/encyclopedia/term/51762/source-code-editor>>. Acesso em: 26 maio 2015.

[17] Using the Metadata API to Retrive Picklist Values. Selesforce Developers. Disponível em:<<https://developer.salesforce.com/blogs/developer-relations/2008/12/using-the-metadata-api-to-retrieve-picklist-values.html>>. Acesso em: 26 maio 2015.

[18] It's All About Intelligent Code Completion. Selesforce Developers. Disponível em:< <http://code-recommenders.blogspot.com.br/2010/05/its-all-about-intelligent-code.html> >. Acesso em: 26 maio 2015.

[19] Code::Blocks Wiki FAQ. Code::Blocks Wiki. Disponível em:< <http://wiki.codeblocks.org/index.php?title=FAQ> >. Acesso em: 26 maio 2015.

[20] Qt Creator Completing Code. Qt Documentation. Disponível em:< <http://doc.qt.io/qtcreator/creator-completing-code.html> >. Acesso em: 26 maio 2015.

[21] Using Intellisense. Microsoft Documentation. Disponível em:< <https://msdn.microsoft.com/en-us/library/hcw1s69b.aspx> >. Acesso em: 26 maio 2015.

- [22] FARIAS, G; MEDEIROS, E. S. *Introdução a Computação*. Disponível em:< <http://producao.virtual.ufpb.br/books/camyle/introducao-a-computacao-livro/livro/livro.pdf> >. Acesso em: 26 maio 2015. 88p.
- [23] BARRETO, J.M. S. *Introdução a Computação*. Disponível em:< <http://www.inf.ufsc.br/~barreto/cca/arquitet/arq4.htm> >. Acesso em: 26 maio 2015.
- [24] MANSOUR, I.H. S. *Linguagem de Programação C*. Disponível em:< <http://www.inf.pucrs.br/manssour/LinguagemC/> >. Acesso em: 26 maio 2015.
- [25] GOSLING, James; JOY, Bill; STEELE, Guy; BRACHA, Gilad; Buckley, Alex. The Java® Language Specification. Java SE. 8 ed. Redwood City – California, 2014.
- [26] Write Once Run Anywhere. Computer Weekly. Disponível em:< <http://www.computerweekly.com/feature/Write-once-run-anywhere> >. Acesso em: 26 maio 2015.
- [27] The Java Language Environment. Oracle. Disponível em:< <http://www.oracle.com/technetwork/java/intro-141325.html> >. Acesso em: 26 maio 2015.
- [28] Java No Longer A Favorite. Wired. Disponível em:< <http://www.wired.com/2013/01/java-no-longer-a-favorite/> >. Acesso em: 26 maio 2015.
- [29] The RedMonk Programming Language Rankings. RedMonk. Disponível em:< <http://redmonk.com/sogrady/2015/01/14/language-rankings-1-15/> >. Acesso em: 26 maio 2015.
- [30] The History Of Java Technology. Oracle. Disponível em:< <http://www.oracle.com/technetwork/java/javase/overview/javahistory-index-198355.html> >. Acesso em: 26 maio 2015.
- [31] BUYYA, Rajkumar; SELVI, S. Thamarai; CHU, Xingchen. Object-oriented Programming with Java: essentials and Applications. New Delhi: Tata McGraw-Hill Education, 2009.
- [32] VENNERS, B. Inside The Java Virtual Machine. Artima Developer. . Disponível em:< <http://www.artima.com/insidejvm/ed2/index.html> >. Acesso em: 26 maio 2015.

- [33] LINDHOLM, T; YELLIN, F; BRACHA, G; Buckley A. *The Java Virtual Machine Specification*. Java SE. 7 ed. Redwood City – California, 2013.
- [34] *Bytecode Basics: A First Look At The Bytecodes Of The Java Virtual Machine*. Java World. Disponível em:< <http://www.javaworld.com/article/2077233/core-java bytecode-basics.html> >. Acesso em: 26 maio 2015.
- [35] *Java Bytecode: Understanding Bytecode Makes You A Better Programmer*. IBM developerWorks. Disponível em:< [http://www.ibm.com/developerworks/library/it-haggar\\_bytecode/](http://www.ibm.com/developerworks/library/it-haggar_bytecode/) >. Acesso em: 26 maio 2015.
- [36] Overview Of The .Net Framework. MSDN Microsoft. Disponível em:< <https://msdn.microsoft.com/en-us/library/zw4w595w.aspx> >. Acesso em: 26 maio 2015.
- [37] Common Language Runtime (CLR). MSDN Microsoft. Disponível em:< <https://msdn.microsoft.com/en-us/library/8bs2ecf4.aspx> >. Acesso em: 26 maio 2015.
- [38] Object Oriented Programming Languages Pattern. Disponível em:< <https://github.com/Judahh/Pattern/blob/master/General/Object%20Oriented%20Programming%20Languages%20Pattern.mediawiki> >. Acesso em: 26 maio 2015.
- [39] COOK, A; JARVIS, J; LEE, J. Evolving The Google Identity. Google Design. Disponível em:< <https://design.google.com/articles/evolving-the-google-identity/> >. Acesso em: 26 maio 2015.
- [40] MCCATHIENEVILE, C; KOIVUNEN, M. Accessibility Features of SVG. W3C. Disponível em:< <http://www.w3.org/TR/SVG-access/> >. Acesso em: 26 maio 2015.
- [41] HATFIELD, B.O; ZHANG, M.; ZHEN-LAN, J. ***A General-Purpose Custom-Design Edassembler In C***. *Frontiers in Education*: IEEE, v. 2, p. F3C-1-2 5-8 Nov. 2003.
- [42] TAVERNIER, K.R.; NOTREDAME, P.H. ***Macro-Based Cross Assemblers. Software Engineering, IEEE Transactions on***: IEEE, v. 6, p. 334 – 340, July 1980.
- [43] NAKANO, K; ITO, Y. *Processor, Assembler, and Compiler Design Education using an FPGA. Parallel and Distributed Systems*: IEEE. p. 723-728. *International Conference on Melbourne*, VIC, 8-10 Dec. 2008.
- [44] MATHISKE, B. SIMON, D.; UNGAR, D. *An assembler and disassembler framework for JavaTM programmers*. ***Science of Computer Programming***, v. 70, p. 127–148, 2008.
- [45] DAVIS, I.J.; GODFREY, M.W. *From Whence It Came: Detecting Source Code Clones by Analyzing Assembler*. *Working Conference on Beverly, MA. Reverse Engineering*: IEEE, p. 242–246, 13-16 Oct. 2010.

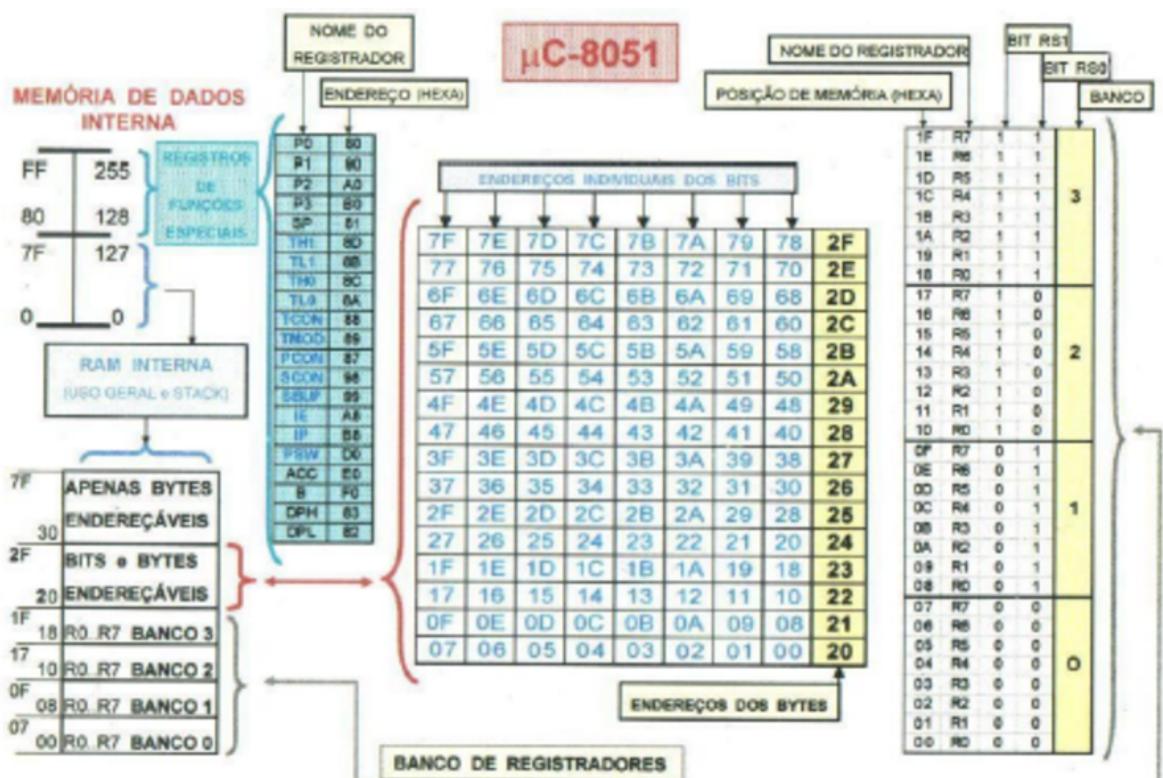
- [46] SALIN, S.I.; SULAIMAN, H.A.; JAMALUDDIN, R.; SALAHUDDIN, L.; ZAINUDIN, M.N.S.; SALIM, A.J. *Two-pass Assembler Design for a Reconfigurable RISC Processor. Conference on Kuching. Open Systems: IEEE*, p. 77-82, 2-4 Dec. 2013.

## ANEXO

### Processadores

Foram selecionados dois processadores com os quais o sistema se preocupará, são eles: o 8051 e o Z80.

No 8051, tem-se a seguinte configuração:



**Figura 35:** Imagem do resultado da tradução de um código de Divisão Easembly para os Assemblies do 8051 e do z80 e sua versão feita manualmente para 8051

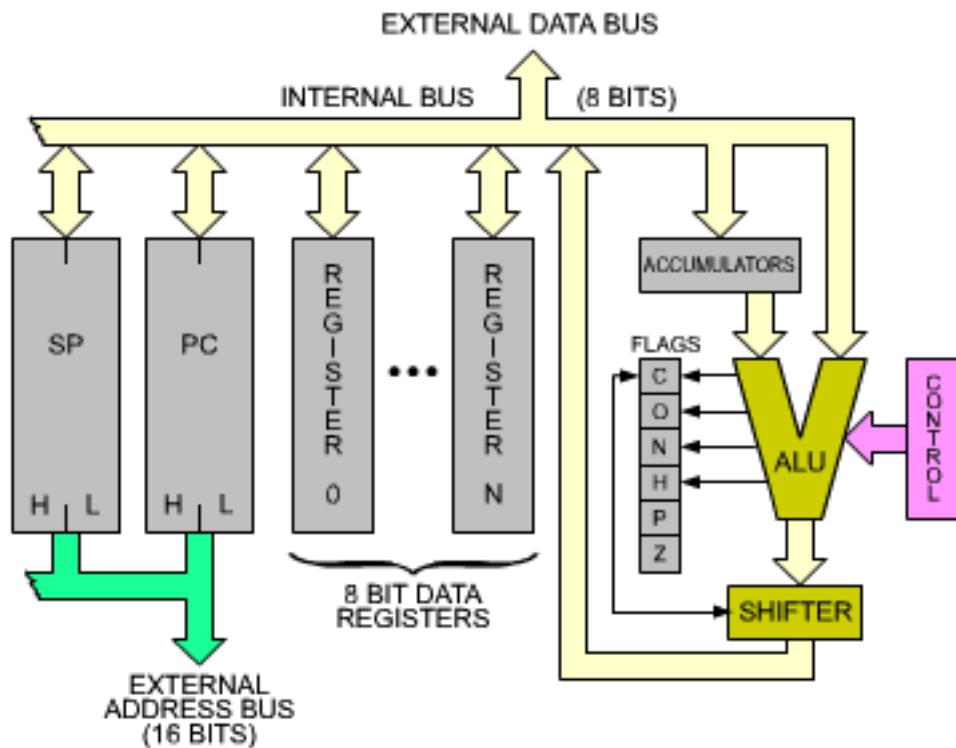
## Arquitetura

### Assembly

Todos os membros da família 8051 executam o mesmo conjunto de instruções, composto por orientações otimizadas para aplicações de controle, facilitando as operações de dados, por meio de vários modos de endereçamento. Capacita, ainda, a operação de variáveis de um *bit*, permitindo operação em sistemas que demandam processamento booleano. Adiante, descreve-se brevemente o modo de operação de várias instruções contidas neste conjunto. Maiores informações podem ser encontradas nos manuais dos fabricantes, os quais descrevem detalhadamente o conjunto de instruções.

Já no Z80 tem-se o seguinte:

## Arquitetura



**Figura 36:** Imagem do resultado da tradução de um código de Divisão Easembly para os Assemblies do 8051 e do z80 e sua versão feita manualmente para 8051

## 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, que 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, em que o segundo byte consta de dados (exceto para indexação, o qual adiciona um byte extra). Para o restante das instruções, é possível que seja requerida a especificação de um endereço.

Um endereço requer 16 *bits* (2 *bytes*). Sabendo disto, uma instrução que use endereço pode variar, em tamanho, de 3 a 4 *bytes*.

Para cada *byte* de instrução, a unidade de controle deverá realizar um *fetch* de memória, 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).