1 INTRODUÇÃO

Migração para Web

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].

Desenvolvimento de Softwares Atualmente

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 que 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.

Vantagens de uma IDE Online

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*.

Integração de Todos os Serviços

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.

Arquitetura Diferentes

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].

Proposta

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 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.

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.

Desenhar um banco de dados para que se possa ter um banco de códigos com controle de versão básico, para que o programador não tenha que ficar se preocupando com programas externos para este fim.

Criar um buscador de códigos e indexador, para que se torne fácil pesquisar um código e utilizá-lo de forma rápida e simples, a fim de facilitar o reuso de códigos.

# 3 OBJETO

## 3.1 Problema

É possível utilizar/criar um *Assembly* Único para vários processadores de arquitetura diferente?

O processamento do montador é viável pra a linguagem PHP?

O Sistema tem uma abordagem eficiente?

## 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 controle de versão e 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 um poder de processamento mais modesto.

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 inúmeros 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].

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

# 7 METODOLOGIA

A metodologia utilizada para a criação desta arquitetura será realizada a partir da pesquisa e estudo do estado da arte de projetos, sistemas e artigos que se assemelham ao tema e, assim, desenhar a arquitetura baseada neste estudo. Ao final, fazer um protótipo. A validação da arquitetura será realizada a partir de um estudo de caso sobre o protótipo.