

Universidade Federal de Santa Catarina - UFSC  
Centro Tecnológico - CTC  
Departamento de Informática e Estatística - INE  
Curso de Sistemas de Informação

# Uma Máquina Virtual para uso didático

**Autor:** Lucas Martins Silva

**Orientador:** Olinto José Varela Furtado

**Banca Examinadora:** José Eduardo De Lucca e Jerusa Marchi

**Palavras-chave:** máquinas virtuais, compiladores, geração de código, interpretadores, computação didática

## Sumário

Lista de Figuras.....	5
Lista de Abreviaturas.....	6
Resumo .....	7
<i>Abstract</i> .....	7
Introdução .....	8
1. Fundamentação Teórica.....	10
1.1. Tipos de Máquinas Virtuais.....	10
1.1.1. Máquinas Virtuais de sistema .....	10
1.1.2. Máquinas Virtuais de sistema operacional .....	11
1.1.3. Máquinas Virtuais de aplicação .....	13
1.2. Formas de Implementação .....	15
1.2.1. Emulação de um hardware real .....	15
1.2.1.1. Virtualização completa .....	16
1.2.1.2. Paravirtualização.....	17
1.2.1.3. Recompilação dinâmica .....	18
1.2.1.4. Hypervisor.....	19
1.2.2. Emulação de um hardware virtual .....	20
1.2.2.1. Java Virtual Machine (JVM) .....	20
1.2.2.2. Pascal-P.....	23
1.2.2.3. P-Code da Microsoft.....	23
1.2.2.4. Common Language Infrastructure (CLI).....	24
2. A Linguagem de Sistemas de Informação.....	25
2.1. Definição formal da linguagem.....	25
2.1.1. Especificação léxica.....	26
2.1.2. Especificação sintática.....	28
2.1.3. Especificação semântica .....	31
3. A Máquina Virtual da Linguagem de Sistemas de Informação.....	35
3.1. Arquitetura da MV-LSI.....	35
3.2. Tipos de dados.....	36
3.3. Especificação do formato do arquivo executável .....	37
3.4. Repertório de instruções .....	39
3.5. Geração de código intermediário .....	41
3.5.1. Início do programa .....	41
3.5.2. Declaração de variáveis.....	41
3.5.3. Declaração de constantes .....	43

3.5.4. Declaração de arrays.....	43
3.5.5. Declaração de métodos.....	43
3.5.6. Comando de atribuição - variável ou parâmetro.....	44
3.5.7. Comando de atribuição - arrays.....	46
3.5.8. Comando "se".....	46
3.5.9. Comando "enquanto".....	48
3.5.10. Comando de leitura e escrita de dados.....	49
3.5.11. Chamada de métodos.....	50
3.5.12. Passagem de parâmetros por referência.....	51
3.5.13. Comando de retorno de método.....	52
3.5.14. Expressões lógicas, matemáticas e de comparação.....	54
3.6. Otimizações de código.....	55
4. Implementação e uso da Máquina Virtual.....	57
4.1. O núcleo da Máquina Virtual.....	58
4.2. Interface gráfica.....	60
4.3. Pacote padrão de instruções e tipos.....	61
4.3.1. Tipos de dados.....	61
4.3.2. Instruções para alocação de espaço na pilha.....	62
4.3.3. Instruções para armazenamento.....	63
4.3.4. Instruções para carga de valores na pilha.....	64
4.3.5. Instruções para leitura e impressão.....	65
4.3.6. Instruções para operações lógicas.....	66
4.3.7. Instruções para operações matemáticas.....	67
4.3.8. Instruções para comparações.....	68
4.3.9. Instruções para conversões de tipos.....	69
4.3.10. Instruções para desvios de fluxo.....	71
4.3.11. Instruções para chamadas e retornos de métodos.....	71
4.3.12. Instruções para manipulação de cadeias de caracteres.....	73
4.3.13. Outras instruções.....	74
4.4. Criando pacotes de instruções e tipos (extensões).....	74
4.4.1. Criação de um novo tipo de dado.....	74
4.4.2. Estendendo a linguagem para adicionar suporte à arrays bidimensionais.....	80
4.5. Integrando a MV-LSI em outros projetos.....	83
4.6. Utilizando a MV-LSI para executar programas.....	87
5. Conclusão.....	89
Bibliografia.....	90

Apêndice A - Artigo no formato SBC .....	92
Apêndice B - Código fonte do núcleo da MV-LSI .....	96
Apêndice C - Código fonte da interface gráfica .....	120
Apêndice D - Código fonte do pacote padrão de extensões .....	169
Apêndice E - Código fonte do pacote de extensão - Matrículas .....	224
Apêndice F - Código fonte da extensão para Arrays Bidimensionais .....	234

## Lista de Figuras

Figura 1: esquema de máquinas virtuais de sistema .....	10
Figura 2: esquema de máquinas virtuais de sistema operacional.....	12
Figura 3: esquema de uma máquina virtual de aplicação (Java) .....	13
Figura 4: esquema de um emulador de hardware real.....	15
Figura 5: esquema de virtualização completa .....	16
Figura 6: esquema da paravirtualização com uso de hypercalls.....	17
Figura 7: esquema de monitor de máquinas virtuais do tipo II .....	19
Figura 8: arquitetura da máquina virtual Java .....	22
Figura 9: Arquitetura da MV-LSI.....	36
Figura 10: tela principal do aplicativo (InterfaceMVLSI.java).....	60
Figura 11: adicionando o núcleo da Máquina Virtual como biblioteca em outros projetos.....	84
Figura 12: tela inicial da MV-LSI.....	88
Figura 13: interface para gerenciamento dos pacotes de instruções.....	89
Figura 14: exemplo de máquina virtual de aplicação (Java Virtual Machine)...	93
Figura 15: Arquitetura da MV-LSI.....	94

## **Lista de Abreviaturas**

**BNF** - *Backus-Naur form* - forma de notação de gramáticas livres de contexto

**GALS** - Gerador de Analisadores Léxicos e Sintáticos

**ISA** - *Instruction set architecture* (conjunto de instruções de uma arquitetura)

**JIT Compiler** - *Just-in-time compiler* (compilador em tempo de execução)

**JRE** - *Java Runtime Environment* (Ambiente de Execução Java)

**JVM** - *Java Virtual Machine* (Máquina Virtual Java)

**LLVM** - inicialmente *Low Level Virtual Machine*

**LSI** - Linguagem de Sistemas de Informação

**LSIB** - *LSI-Binary* (formato de arquivo executável pela MV-LSI)

**LSIBC** - *LSI-Bytecode* (formato de arquivo em linguagem intermediária)

**MV** - Máquina Virtual

**MV-LSI** - Máquina Virtual da Linguagem de Sistemas de Informação

**SO** - Sistema Operacional

**VM** - *Virtual Machine* (máquina virtual)

**VMM** - *Virtual Machine Monitor* (monitor de máquinas virtuais)

## **Resumo**

Este trabalho aborda o desenvolvimento de uma máquina virtual (MV-LSI) adaptada às necessidades do curso de Sistemas de Informação da UFSC. O software auxiliará no ensino de disciplinas do curso, especialmente Compiladores, pois a máquina permite a execução de programas compilados e estudados na linguagem de programação própria da matéria (LSI).

Palavras-chave: máquinas virtuais, compiladores, geração de código, interpretadores, computação didática

## **Abstract**

*This paper approaches the development of a virtual machine (MV-LSI), adapted to the needs of the course of Information Systems at UFSC. The software will aid in teaching course subjects, especially Compilers, because the machine allows the execution of programs compiled and studied in the programming language of the course (LSI).*

*Key-words: virtual machines, compilers, code generation, interpreters, didactic computing*

## Introdução

No ensino de Compiladores no curso de Sistemas de Informação da UFSC, é utilizada uma linguagem de programação de alto nível, bastante simples e objetiva, que é levemente modificada a cada semestre. Com ela o professor tem a possibilidade de ensinar diversos aspectos da compilação de um programa, conceitos de compiladores, geração de código, entre outros. Entretanto, a máquina virtual existente é informal e muito limitada. A execução de programas da linguagem é feita manualmente — que é muito trabalhosa e suscetível a erros —, ou por meio de um interpretador simplificado, o que permite apenas a verificação de alguns aspectos. Além disso, o tempo disponibilizado para a matéria é bastante curto, não sobrando tempo para o desenvolvimento de um interpretador a cada semestre letivo.

Buscando resolver estes problemas e propiciar mais recursos que facilitem o aprendizado de tais conteúdos, neste trabalho será desenvolvida uma máquina virtual capaz de executar programas desta linguagem (chamada de *LSI - Linguagem de Sistemas de Informação*). Tal máquina deverá ser personalizável, suportando diferentes versões da linguagem fonte (LSI), bem como a experimentação de novos conceitos, por meio da inclusão de novas instruções e tipos à arquitetura. Utilizando a nomenclatura da área, o programa desenvolvido pode ser classificado como uma Máquina Virtual de Aplicação (Seção 1.1.3), que é implementada por meio da Emulação de um Hardware Virtual (Seção 1.2.2).

Existem atualmente diversas ferramentas de emulação de uma plataforma de hardware, como o Bochs, que emula um processador com o ISA (*instruction set architecture*) x86; o QEMU (Quick EMUlator), capaz de emular diversas arquiteturas de hardware; o Emu8086, que emula o processador Intel 8086, mostrando os valores dos registradores e instruções sendo executadas, etc.

Entretanto, a complexidade na geração de código para essas arquiteturas inviabilizaria seu uso, pois seria necessário todo um estudo da



arquitetura, o que não é possível devido ao tempo limitado da disciplina de compiladores. Por isso a necessidade de construção de uma máquina virtual simples que irá emular a execução de um programa LSI.

O trabalho ainda tem como objetivo abordar os principais conceitos de máquinas virtuais de diversos tipos, o processo de geração de código intermediário em um compilador para a MV-LSI, e otimizações simples de código. O programa gerado deve ser integrável a outros projetos (compiladores, por exemplo), além de permitir a criação de extensões (instruções e tipos extras).

Este documento é estruturado da seguinte forma: no capítulo 1 são estudados diversos conceitos acerca de virtualização. Dentro da seção 1.1 estão explicações sobre as formas de virtualização existentes atualmente, quais são as vantagens e desvantagens de cada uma, entre outros; A seção 1.2.2 é a que aborda os conceitos relacionados com o trabalho desenvolvido - emulação de hardwares virtuais. O capítulo 2 é dedicada à apresentação da Linguagem de Sistemas de Informação, uma linguagem de programação de alto nível utilizada na disciplina de Compiladores do curso na UFSC. Neste capítulo são detalhados os aspectos léxicos, sintáticos e semânticos da linguagem. O capítulo 3 trata da MV-LSI, sua arquitetura, linguagem intermediária, formato de arquivos, instruções e geração de código. No capítulo 4 são expostos os aspectos de implementação do projeto. Vale destacar as seções 4.4 e 4.5, que mostram, respectivamente, como criar extensões e como integrar este trabalho em outro projeto. O resultado do trabalho é detalhado no capítulo 5.

## 1. Fundamentação Teórica

Uma máquina virtual (*virtual machine* ou VM, em inglês) pode ser definida como um sistema operacional, chamado de "convidado", executando sobre outro sistema (sistema "host"), de forma "isolada".

Com base na definição acima, é possível imaginar diversas formas de se implementar esse conceito, que são descritas nos itens a seguir.

### 1.1. Tipos de Máquinas Virtuais

Uma forma de classificação das Máquinas Virtuais é dividi-las em três categorias: de sistema, de sistema operacional e de aplicação [MAZIERO, 2011].

#### 1.1.1. Máquinas Virtuais de sistema

As máquinas virtuais de sistema são aquelas onde a implementação ocorre entre o hardware real e o sistema operacional convidado. Na verdade, nesse tipo de VM, existe um gerenciador intermediário, chamado de Monitor de Máquinas Virtuais (VMM), ou Hypervisor. O VMM é responsável por fazer o controle de acesso à memória, dados e periféricos dos sistemas operacionais convidados, conforme a Figura 1.

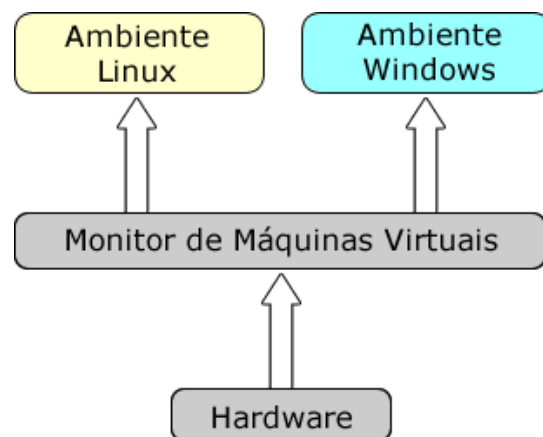


Figura 1: esquema de máquinas virtuais de sistema

Como vantagens desse modelo, pode-se citar:

- forte isolamento dos sistemas operacionais convidados, ou seja, é impossível um sistema convidado ter acesso direto à memória e arquivos pertencentes à outro sistema convidado.
- a possibilidade de executar, em uma mesma máquina real, sistemas operacionais distintos, inclusive de arquiteturas diferentes da máquina real.
- provisionamento e manutenção dos sistemas operacionais convidados pode ser facilitada bastante com o uso de VMM. Pode-se, por exemplo, mudar a qualquer momento o tamanho da memória e espaço em disco destinada a cada uma das máquinas virtuais convidadas.
- em razão do isolamento proporcionado pelo VMM, esse ambiente pode ser usado para testes de sistemas operacionais e outros programas não confiáveis, por exemplo. Assim, não há o risco de um vírus ou malware contaminar o sistema operacional hospedeiro ou as outras máquinas virtuais.

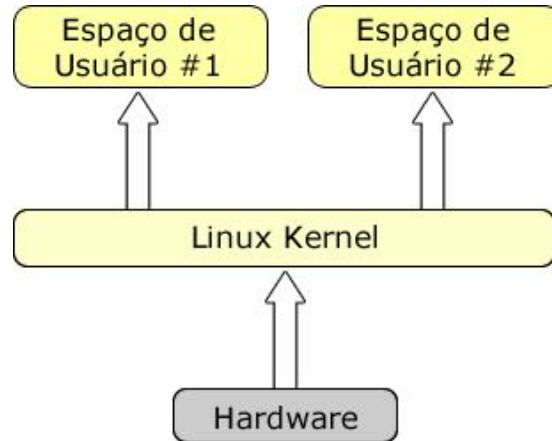
Entretanto, existem desvantagens nesse conceito, tais como:

- a execução de um sistema operacional dentro de uma máquina virtual sempre será mais lenta em comparação com o mesmo sistema rodando diretamente sobre o hardware.
- necessidade de utilização de um hardware poderoso no caso de existirem muitas máquinas virtuais executando ao mesmo tempo. Caso isso não seja possível, dependendo da carga de trabalho somada de todas as máquinas virtuais, algumas delas podem até deixar de responder em alguns momentos em decorrência da falta de recursos.

### **1.1.2. Máquinas Virtuais de sistema operacional**

As máquinas virtuais de sistema operacional são aquelas onde existe apenas um sistema operacional rodando sobre o hardware, entretanto, este SO tem a capacidade de disponibilizar diversos espaços de usuários separados. A cada um desses espaços, chamados de zonas ou domínios virtuais, são

alocados uma certa quantidade de memória, disco e tempo de uso do processador. Esse modelo também é chamado de "servidor virtual" e está representado na Figura 2.



**Figura 2: esquema de máquinas virtuais de sistema operacional**

O custo dessa forma de virtualização é bastante baixo, tendo em vista que existe apenas um Kernel para todas as máquinas virtuais. Assim, podem ser criadas milhares de VM's em apenas uma máquina real. O mesmo não poderia ser conseguido em outras arquiteturas de máquinas virtuais, como a virtualização completa, em que cada VM possui um kernel, gerando uma sobrecarga no sistema que impediria a execução de tantas VM's ao mesmo tempo.

O objetivo desse tipo de implementação não é ter a capacidade de executar programas de arquiteturas diferentes, e sim, de separar processos de "clientes" diferentes. Também, é uma opção quando existem diversas máquinas físicas sendo utilizadas para rodar aplicações servidoras, como servidores de e-mail, DNS, web, de jogos, entre outros, e a capacidade ociosa de processamento dessas máquinas é alta. Esses servidores podem ser todos alocados em apenas uma máquina real, utilizando esse conceito de servidor virtual, aumentando a eficiência no uso dos recursos de hardware.

Esse modelo é utilizado em empresas que alugam infra-estrutura de hardware para que outras empresas e pessoas possam executar aplicações.

Cada um desses clientes terá seu espaço de usuário definido, sem que um possa interferir no outro. Isso ajuda reduzir custos, pois em uma máquina física podem ser alocados diversos clientes, ao invés de ter que colocar uma máquina real para cada um.

### 1.1.3. Máquinas Virtuais de aplicação

As máquinas virtuais de aplicação, ou de processo, servem apenas para a execução de um software sobre um sistema operacional qualquer. Elas não são feitas para a execução de um sistema operacional completo.

Nessa arquitetura, um programa é escrito em uma linguagem de programação que será interpretada pela máquina virtual, que será responsável por ler o código binário (*bytecode*) do programa e executá-lo utilizando as instruções da máquina real.

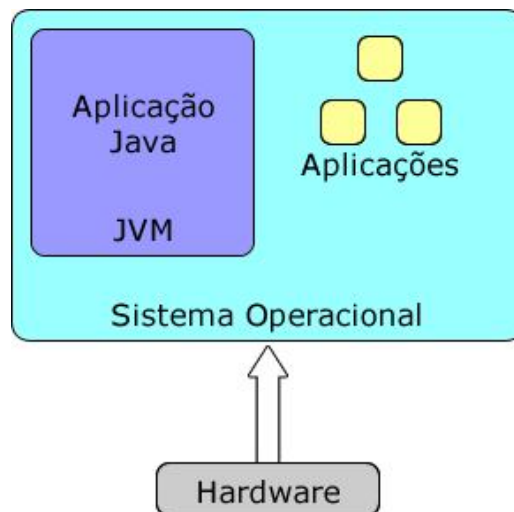


Figura 3: esquema de uma máquina virtual de aplicação (Java)

O exemplo mais conhecido desse tipo de máquina virtual é a Java Virtual Machine (JVM). Ao compilar um aplicativo em Java, o compilador irá gerar um *bytecode* contendo as instruções da máquina virtual Java. Posteriormente, esse *bytecode* será lido e interpretado pela JVM. A Figura 3 ilustra esse conceito.

Uma característica desse tipo de VM é que ela não cumpre a regra de isolamento, conforme descrito na seção 1.1. Aplicações desenvolvidas para a JVM, por exemplo, podem interagir com outros processos e manipular arquivos do sistema operacional hospedeiro.

Como vantagem desse método tem-se, entre outras:

- É multi-plataforma, ou seja, um programa escrito para a máquina virtual poderá ser executado em qualquer sistema operacional, basta que exista uma máquina virtual desenvolvida para esse SO. Assim, a dificuldade de se implementar um código para os mais diversos sistemas operacionais recai sobre o fabricante da máquina virtual. No caso da Oracle, que desenvolve o Java, são disponibilizadas JVM's (*Java Runtime Environment* — JRE, leia a seção 1.2.2.1) para Windows, Linux, Solaris, entre outros;
- Menos consumo de recursos do sistema, em comparação com os outros tipos de máquinas virtuais;
- As aplicações não precisam ser adaptadas ou programadas novamente em caso de mudança do sistema operacional da máquina real.
- Maior segurança, do ponto de vista do sistema operacional, pois uma aplicação executando em uma máquina virtual desse tipo tem pouca capacidade de causar danos ao sistema.
- Pode ajudar na economia de dinheiro em grandes organizações, onde o parque de máquinas é muito grande, ao padronizar as aplicações, manuais, treinamento, etc.;

Como desvantagem, a execução de programas é mais lenta, pois as instruções precisam ser interpretadas pela máquina virtual. Para amenizar este problema, algumas VM's se utilizam da chamada *Just-in-time compilation*, também chamada de recompilação dinâmica, em que partes do código da aplicação são compilados para a máquina real, durante a execução, aumentando a velocidade.

## 1.2. Formas de Implementação

Existem basicamente duas formas de se construir uma máquina virtual: por meio da virtualização de um hardware real — terá a capacidade de emular todas as instruções de um processador de verdade — ou um hardware virtual, cujas instruções não correspondem à um processador real.

### 1.2.1. Emulação de um hardware real

Nessa forma de virtualização, a ISA (*Instruction Set Architecture*) de um processador real (por exemplo, Intel x86) é virtualizada por meio de software. Isso pode ser feito de três maneiras: virtualização completa, em que todas as instruções são emuladas pela VMM, incluindo memória e periféricos; paravirtualização, que é semelhante à virtualização completa, mas algumas instruções particulares são modificadas e recompilação dinâmica (ou tradução dinâmica), em que blocos de instruções da VMM são diretamente convertidas em instruções da máquina real. [MAZIERO, 2011]

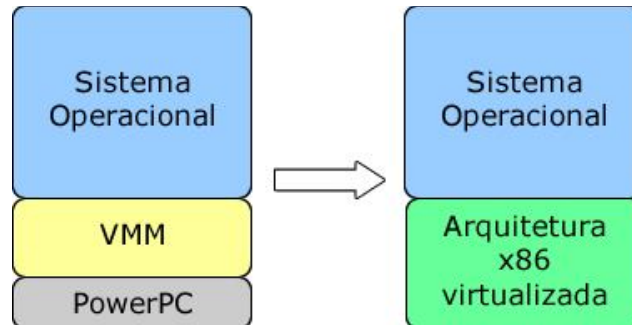


Figura 4: esquema de um emulador de hardware real

A Figura 4 mostra o conceito de emulação de um hardware real. O sistema operacional acredita estar executando em uma plataforma de hardware que não existe fisicamente no ambiente.

Nesta seção ainda serão abordados detalhes do conceito de *Hypervisor* (ou monitor de máquinas virtuais).

### 1.2.1.1. Virtualização completa

Esta técnica compreende a emulação de todas as instruções de um hardware específico, sem exceção, incluindo operações de entrada e saída de dados, interrupções, memória e periféricos.

Dessa forma, é possível que um sistema operacional desenhado para ser executado em uma plataforma de hardware possa funcionar em outra completamente diferente — inclusive, sem esse sistema operacional saber que está sendo virtualizado. Da mesma forma, o SO convidado não precisa de qualquer alteração em seu código para poder ser virtualizado. A Figura 5 ilustra o caso.

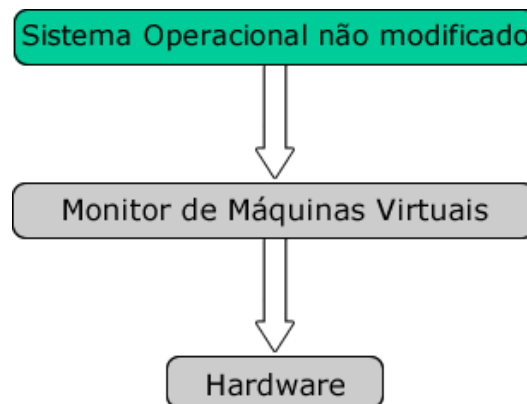


Figura 5: esquema de virtualização completa

Embora possa parecer simples, existem alguns detalhes que dificultam a construção de um monitor de máquina virtual que suporte a virtualização completa. Por exemplo, a plataforma x86, um dos mais populares conjuntos de arquiteturas de hardware, utilizada nos processadores Intel e AMD. Nela, existem quatro níveis (ou Rings) de execução, sendo o Ring 0 o mais privilegiado, e o Ring 3 o menos privilegiado. Os programas de usuário executam no Ring 3, enquanto que os comandos do sistema operacional são executados no Ring 0. Isso porque o sistema operacional precisa de acesso direto à memória e ao hardware. Só que quando esse SO é virtualizado, todas as suas instruções passariam a executar no Ring 3, o que impossibilitaria esse código de funcionar - a mesma instrução pode ter comportamento diferente



dependendo do Ring em que ela estiver executando. A solução foi a utilização de uma camada de software (a VMM), que executaria no nível 0, enquanto que os SO's convidados seriam executados em um nível de maior privilégio que as aplicações, porém com menos privilégios que a VMM. [VMWARE, 2007]

Como exemplo de VMM que usa essa técnica, podemos citar o VMWare Server e Workstation, que resolveu o problema de virtualizar a plataforma x86 por meio da utilização da recompilação dinâmica.

### 1.2.1.2. Paravirtualização

A paravirtualização é uma técnica muito semelhante à virtualização completa, exceto que algumas modificações são feitas na interface com o sistema operacional convidado. Dessa forma, o sistema operacional convidado precisará sofrer modificações em seu núcleo, para que possa enviar à VMM instruções específicas (chamadas de *hypercalls*) dessa plataforma de virtualização. Essas instruções podem servir para otimizar a execução de código do usuário e do SO convidado, de gerenciamento de memória e interrupções, controle do tempo, entre outras vantagens. Os programas de usuário do SO convidado não precisam ser alterados.

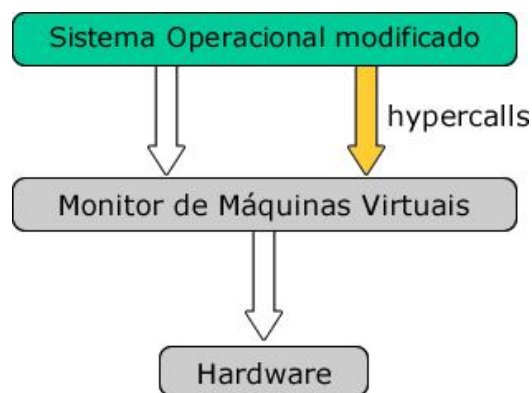


Figura 6: esquema da paravirtualização com uso de hypercalls

A Figura 6 acima mostra essa técnica, com as chamadas comuns de sistema sendo representadas pela seta branca que sai do retângulo verde, e as *hypercalls* representadas pela seta amarela.

Assim, o desempenho de um modo geral é melhor em ambientes paravirtualizados em relação aos que utilizam a virtualização completa.

Um dos pontos que pode ser considerado desvantagem, é que o sistema operacional convidado precisa ser modificado, para que este possa se comunicar com a VMM. Na virtualização completa isso não é necessário.

A construção de um monitor de máquina virtual baseado em paravirtualização é relativamente simples, segundo [VMWARE, 2007].

Um dos softwares de paravirtualização mais conhecidos é o Xen, que é open-source, enquanto que a VMWare também tem produtos que utilizam esse conceito.

### **1.2.1.3. Recompilação dinâmica**

Também chamada de Tradução Dinâmica, a recompilação dinâmica consiste em reorganizar e modificar blocos de instruções que são enviados pelo Sistema Operacional convidado para a VMM. Essa reorganização acontece em tempo de execução, de forma totalmente transparente para o usuário e para o SO convidado. [MAZIERO, 2011]

A otimização do desempenho na execução de um programa oferecida por essa técnica acontece porque o recompilador possui informações em tempo de execução, o que o compilador estático não tinha quando compilou esse programa. Assim, o recompilador pode guardar em cache trechos de códigos muito utilizados, aumentando ainda mais o desempenho.

A recompilação é feita pelos chamados JIT - Just In Time Compiler, e são encontrados em diversas plataformas de virtualização, como o Java e o .NET da Microsoft.

#### 1.2.1.4. Hypervisor

Os hypervisors, também chamados de Virtual Machine Monitors (VMM - monitor de máquinas virtuais, em português) são os programas que efetivamente proverão um ambiente onde ocorrerá a virtualização de hardware para sistemas operacionais convidados.

Segundo [POPEK e GOLDBERG, 1974], os monitores de máquinas virtuais podem ser de dois tipos:

- Tipo I: a VMM executa diretamente sobre o hardware. A Figura 1 (seção 1.1.1) mostra este tipo de hypervisor.
- Tipo II: a VMM é um programa que está dentro de um sistema operacional (host). A Figura 7 abaixo ilustra o Tipo II:

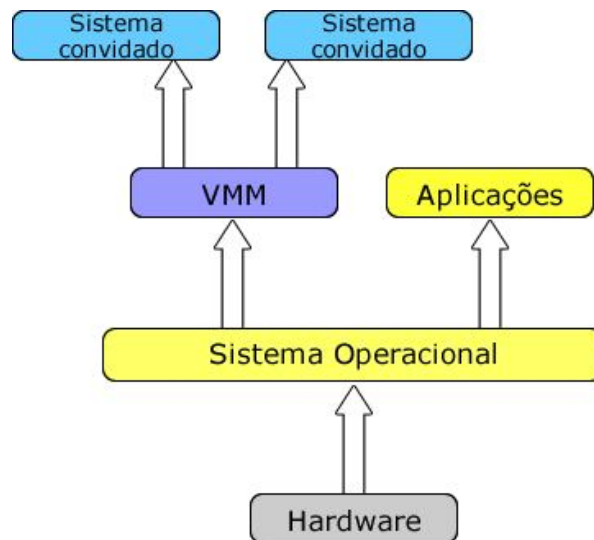


Figura 7: esquema de monitor de máquinas virtuais do tipo II

Os monitores de máquinas virtuais devem possuir três características para serem assim denominados:

1. Qualquer programa executando no ambiente virtualizado deve ter exatamente o mesmo comportamento que teria caso estivesse rodando diretamente sobre o hardware (exceto em relação à desempenho)
2. A segunda característica é a eficiência. Uma VMM deve executar uma porcentagem "*estatisticamente dominante*" de instruções diretamente no processador, sem intervenção via software da VMM. Assim, emuladores e interpretadores não poderiam ser classificados como máquinas virtuais.
3. A VMM deve ter controle sobre os recursos de hardware, como memória e periféricos, além do processador.

Sobre a segunda característica acima, é importante destacar que ela se refere à máquinas virtuais que virtualizam plataformas de hardware existentes fisicamente. Isso não se aplicaria, por exemplo, à JVM e a este trabalho, pois os processadores para o qual são gerados os bytecodes não existem fisicamente, logo todo o código deve ser emulado ou recompilado dinamicamente em tempo de execução.

## **1.2.2. Emulação de um hardware virtual**

Na emulação de um hardware virtual, é desenvolvida toda uma arquitetura de hardware, que não será construída fisicamente, e sim emulada via software. Dessa forma é possível desenvolver aplicações que podem ser independentes de plataforma de hardware, desde que exista um emulador disponível para essa plataforma.

### **1.2.2.1. Java Virtual Machine (JVM)**

A linguagem Java é o exemplo de sucesso das linguagens de programação que executam por meio de uma máquina virtual de aplicação, emulando um hardware virtual.

A Máquina Virtual Java é a aplicação responsável por interpretar os *byte-codes* presentes em um arquivo .jar (que por sua vez contém arquivos de

classes Java compiladas, em formato .class). A JVM emula um hardware que não existe fisicamente, com cerca de 256 instruções e baseado em uma pilha de execução — ao contrário dos processadores reais, que utilizam registradores. [LINDHOLM et al, 2012].

Existe também o Java Runtime Environment (JRE), que implementa a JVM para um hardware específico. Muitas vezes os termos JRE e JVM são confundidos. A JVM é apenas uma especificação de máquina virtual. Para que a execução de um programa Java ocorra, é preciso o JRE específico para cada sistema operacional. Todas estas JRE's implementam a mesma JVM. [MITCHELL, 2000].

A JVM não serve apenas para executar programas em Java. Na verdade, o que a JVM interpreta são as instruções contidas nos arquivos .class. Estas instruções podem ter sido resultado da compilação de um programa Java, mas nada impede que se crie uma outra linguagem totalmente diferente, que quando compilada, gere um código intermediário, seguindo as especificações do *bytecode* Java. Exemplos dessas linguagens são o Clojure <sup>1</sup>, MIDletPascal <sup>2</sup>, Groovy <sup>3</sup>, LLJVM<sup>4</sup> e Scala <sup>5</sup>.

O Java foi criado para ser uma linguagem considerada segura, com tipagem forte, validação de referências, controle rígido de acesso à métodos e atributos de classes, ausência de ponteiros, entre outras características. Muitos dos aspectos de segurança são verificados em tempo de execução. Por exemplo, as instruções de desvio sempre deverão apontar para outra instrução do mesmo método. Com isso, o usuário pode ter uma certa garantia de que a aplicação que está executando não causará danos aos seus arquivos ou ao sistema operacional. Além disso, o programador não precisa se preocupar com o gerenciamento de memória, uma vez que a JVM possui um *Garbage*

---

<sup>1</sup> <http://clojure.org/>

<sup>2</sup> <http://sourceforge.net/projects/midletpascal/>

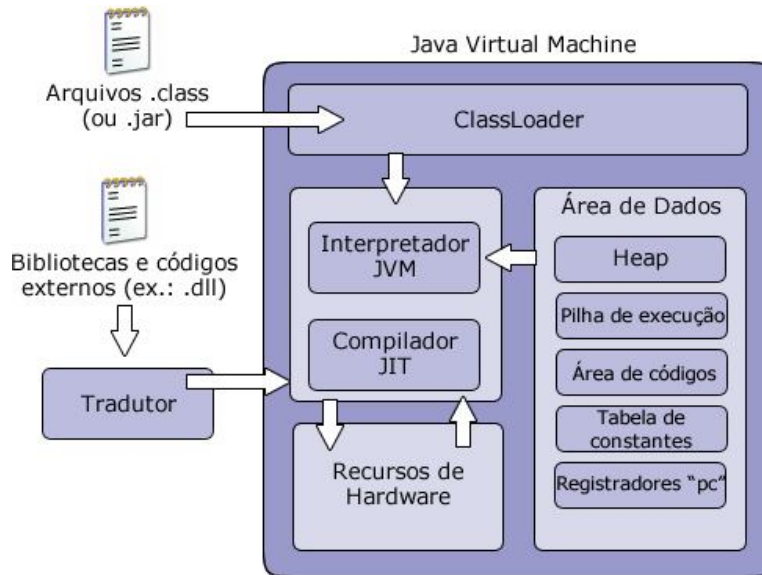
<sup>3</sup> <http://groovy.codehaus.org/>

<sup>4</sup> <http://da.vidr.cc/projects/lljvm/>

<sup>5</sup> <http://www.scala-lang.org/node/25>

*Collector* (Coletor de Lixo), que recupera espaços de memória que não são mais utilizados pela aplicação.

A arquitetura da JVM pode ser visualizada no esquema básico mostrado na Figura 8 abaixo:



**Figura 8: arquitetura da máquina virtual Java**

O compilador da linguagem Java gera arquivos .class (ou .jar, contendo vários .class), contendo informações de uma determinada classe Java, como tabela de constantes, atributos de classe, códigos dos métodos, etc. Esses arquivos são carregados pelo componente ClassLoader da JVM. Códigos externos (por exemplo, bibliotecas do sistema operacional ou API's) são carregados por um tradutor.

O módulo de execução conta um interpretador do bytecode Java e um Compilador JIT (Just-in-time compiler). Na área de dados, são separados 5 principais espaços:

- *Heap* (onde instâncias de classes e *arrays* são alocados);
- Pilhas (*stacks*) de execução (uma para cada *thread*),
- Área de Códigos (que armazena o código do programa carregado)

- Tabela de constantes (possui informações carregadas da tabela de constantes dos arquivos .class, como por exemplo a "tabela de símbolos")
- Registradores "pc": "pc" significa "*program counter*", ou contador de programa. Indica qual instrução será executada pelo módulo de execução. Há um pc para cada *thread* Java.

### 1.2.2.2. Pascal-P

O sistema Pascal-P foi desenvolvido em 1973 pela equipe do professor Niklaus Wirth, da Universidade de Zurique, Suíça. Ele foi criado especialmente para ser uma versão portátil do Pascal. A primeira versão, o Pascal-P1 implementava apenas uma parte da linguagem Pascal, em razão das máquinas da época terem pouca memória - uma programa interpretado consome mais memória do que um compilado para o hardware em que está ocorrendo a execução. Essa limitação da implementação de parte do Pascal original se manteve ao longo das versões P2, P3 e P4, inclusive no sistema UCSD, criado pela Universidade de San Diego. [MOORE, 2009a]

Em 2009, Scott Moore desenvolveu a versão P5 do Pascal-P, em que toda a linguagem Pascal (conforme a ISO 7185) foi implementada, pois o motivo que gerou a limitação das versões anteriores - a falta de memória - não existe mais. [MOORE, 2009b]

### 1.2.2.3. P-Code da Microsoft

O Visual Basic, da Microsoft, utiliza também o conceito de emulação de um hardware virtual. Os compiladores da linguagem geram um byte-code no formato P-Code, que significa "*packed code*", ou código compactado. Na verdade, o P-code da Microsoft era inspirado no Pascal-P, só que como o objetivo da empresa não era portabilidade, optou por tornar o significado de "p", que era "portável", para "*packed*" (compactado).

Mais tarde, com o aparecimento do Java e a popularização do nome "virtual machine", a Microsoft renomeou o seu sistema, chamando-o de Visual Basic Virtual Machine. A versão 6 (VB6) do Visual Basic conta com aproximadamente 1351 *op-codes* (instruções), enquanto que o Java possui 256 (sendo que vários são reservados para usos futuros). O motivo da grande quantidade de instruções é que muitas delas são para comunicação direta com o ambiente Windows e com outras aplicações da empresa. [CHAMBERLAIN, 2001]

Não foi encontrada uma especificação pública da VB6 VM, em razão do sistema ser protegido pela Microsoft. Muitas das aplicações do pacote Office utilizam o Visual Basic, então isto seria uma estratégia de proteção dos produtos da empresa.

#### **1.2.2.4. Common Language Infrastructure (CLI)**

O CLI - Common Language Infrastructure - é uma especificação aberta desenvolvida pela Microsoft, que foi padronizada pela ECMA-335<sup>6</sup> e ISO 23271. A linguagem intermediária, ambiente de execução e outras características do sistema são definidos nesses padrões.

A implementação da Microsoft dessa arquitetura se chama .NET Framework, que engloba o CLR - Common Language Runtime (ambiente de execução), o FCL - Framework Class Library (biblioteca de classes do framework) e BCL - Base Class Library (biblioteca padrão, parte do FCL).

Entre as linguagens de programação associadas ao .NET estão o C#, Visual Basic .NET, F#, J#, Visual C++, entre outras. Cada uma delas possui uma sintaxe diferente, com o objetivo de atingir o maior número de adeptos de seus produtos. Além disso, a empresa disponibiliza uma versão gratuita de sua IDE, o Visual Studio Express<sup>7</sup>.

---

<sup>6</sup> <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-335.pdf>  
<sup>7</sup> <http://www.microsoft.com/visualstudio/en-us/products/2010-editions/express>



A Microsoft desenvolveu o CLI e o .NET como uma forma de possibilitar o uso de diversas linguagens de programação para a criação de aplicações que rodassem no Windows, e assim aumentar a base de programadores e empresas que desenvolvem aplicações Windows, mantendo assim uma posição estratégica no mercado.

Como opção open-source, existe o Mono, desenvolvido pela empresa Ximian. Com ele é possível executar aplicações do .NET em outros sistemas operacionais, como Linux, BSD, Android, OS X, e no próprio Windows, como alternativa ao .NET Framework da Microsoft. O Mono também possui suas características próprias, como a implementação das instruções SIMD (*Single instruction, multiple data*), arrays de 64 bits, uso do LLVM, etc.

## **2. A Linguagem de Sistemas de Informação**

A LSI - Linguagem de Sistemas de Informação é uma linguagem de programação estruturada e imperativa, muito influenciada pelo Pascal. A cada semestre, o professor da matéria faz modificações nessa linguagem, inserindo ou removendo estruturas e modificando características léxicas, sintáticas e semânticas. Sendo assim, a máquina virtual deve ser capaz de aceitar essas alterações sem grandes transtornos.

A seguir é definida uma versão da LSI, para exemplificação neste trabalho. Esta definição pode ser utilizada no GALS, a fim de se obter o analisador léxico e sintático para a construção de compilador da linguagem. Estes passos não são abordados neste trabalho.

### **2.1. Definição formal da linguagem**

As definições a seguir utilizam métodos formais, estudados na disciplina de Compiladores.

### 2.1.1. Especificação léxica

Nesta seção é apresentada uma especificação léxica, com o objetivo de promover uma completa compreensão de exemplos de códigos utilizados neste trabalho.

Uma especificação léxica serve para determinar quais são os caracteres e símbolos aceitos em uma linguagem, e como eles devem estar dispostos no programa para formar tokens.

Para isso, são utilizadas Expressões Regulares, como mostradas abaixo:

```
LETRA : [a-zA-Z]
DIGITO : [0-9]
COM_LINHA : "/" "/" .*
COM_BLOCO : "/*" ["*"]* "+" ([^"*" "/" ][^"*"]* "*" )* "/"
EXPOENTE : [Ee]("-" "+")?{DIGITO}{DIGITO}+
```

As definições acima servem para determinar o que é uma letra (no caso, qualquer letra de A a Z, maiúsculas e minúsculas), dígito (algarismos de 0 a 9), comentários de linha (começam com //), comentários de bloco (começam com /\* e terminam com \*/ , e a parte exponencial dos números inteiros e reais (iniciar com "E" ou "e", seguidos opcionalmente de "+" ou "-", seguidos de dois ou mais dígitos).

Os Tokens correspondem aos símbolos básicos de uma linguagem de programação, os quais podem ser genéricos (identificadores e constantes, por exemplo), específicos (palavras reservadas) ou ainda sem valor sintático (comentários, por exemplo). O conjunto de tokens de uma linguagem de programação constitui a sua especificação léxica, e geralmente são definidos por meio de Expressões Regulares e Autômatos Finitos.

Abaixo, a lista de *tokens* da linguagem, expressados por meio de Expressões Regulares, em formato adequado para uso no GALS:

```
id : (({LETRA}|"@"({LETRA}|{DIGITO})+)((@"|_|"#")?({LETRA}|{DIGITO})+)* |
{LETRA}
```

```

num_int : {DIGITO}+{EXPOENTE}?
num_real : ({DIGITO}+"."{DIGITO}* | "."{DIGITO}+) {EXPOENTE}?

literal_cadeia : '([^\]|')*'

programa = id : "programa"
inteiro = id : "inteiro"
real = id : "real"
booleano = id : "booleano"
caracter = id : "caracter"
cadeia = id : "cadeia"
metodo = id : "metodo"
ref = id : "ref"
val = id : "val"
se = id : "se"
entao = id : "entao"
enquanto = id : "enquanto"
faca = id : "faca"
leia = id : "leia"
retorne = id : "retorne"
senao = id : "senao"
ou = id : "ou"
e = id : "e"
nao = id : "nao"
falso = id : "falso"
verdadeiro = id : "verdadeiro"
escreva = id : "escreva"
tamanho_cadeia = id : "tamanho_cadeia"
caracter_posicao = id : "caracter_posicao"
posicao_subcadeia = id : "posicao_subcadeia"

:! {COM_LINHA}
:! {COM_BLOCO}
: [\ \n\t\r]*

";"
"."
"["
"]"
","
"_"
"("
")"
":"
"{"
"}"
"<"
">"
"+"
"_"
"*"
"/"

":="
">="
"<="
"<>"
".."

```

Assim, da forma exposta acima, serão aceitos como identificadores de variáveis e métodos, as palavras que comecem por letra ou @ seguido de uma

letra ou dígito. Os caracteres "@", "#" e "\_" podem aparecer no identificador, desde que não sejam repetidos. "#" e "\_" não podem iniciar identificadores.

Números inteiros são formados por palavras que comecem com pelo menos um dígito, podendo ser seguidos de um expoente. Por exemplo: 589 , 590e01, 590e-01, 590e+99.

Números reais são como os inteiros, com a adição do ponto "." decimal. Então os números a seguir são números reais válidos: 10.5, 0.5, .5, 5. , 1.5e+10 .

As cadeias de caracteres literais devem começar e terminar com aspas simples. Correspondem às *strings* de outras linguagens de programação. O termo "literal" significa que o dado em questão aparece diretamente ("literalmente") no código fonte do programa.

É importante observar que todas as características citadas acima podem ser modificadas pelo usuário, podendo ou não interferir no meio como serão utilizadas na MV-LSI.

## **2.1.2. Especificação sintática**

A sintaxe da LSI é especificada formalmente por meio de uma Gramática Livre de Contexto. Uma GLC é composta por símbolos não-terminais, símbolos terminais, produções e símbolo inicial.

Os símbolos não-terminais de uma gramática em algum momento da compilação poderão ser substituídos por tokens. É com eles que a estrutura principal de uma gramática é construída. Também podem ser chamados de meta-variáveis. [FURTADO, 2002].

Abaixo está a lista de não-terminais utilizados na gramática da linguagem.

<programa>  
<bloco>

```

<dcl_var_const>
<dcl_metodos>
<com_composto>
<tipo>
<dimensao>
<lid>
<fator_const>
<constante>
<rep_lid>
<dcl_metodo>
<par_formais>
<tipo_metodo>
<mp_par>
<rep_par>
<comando>
<replistacomando>
<rcomid>
<expressao>
<senaoparte>
<rep_lexpr>
<expsimp>
<resto_expressao>
<oprel>
<termo>
<rep_expsimp>
<op_add>
<fator>
<rep_termo>
<op_mult>
<rvar>
<constante_explicita>

```

Símbolos terminais são os símbolos da linguagem propriamente dita, podendo ser utilizados na formação de sentenças de um programa. Produções são conjuntos de regras gramaticais que determinam como será o desdobramento de não-terminais em outros símbolos não-terminais e/ou terminais. Por exemplo:

```
<programa> ::= programa id ";" <bloco> "." ;
```

A produção acima diz que o não-terminal <programa> reconhece a sequência de tokens formada pela palavra "programa", um identificador, o ponto-e-virgula ";", o não-terminal <bloco> e o ponto final ".".

O símbolo inicial é o não-terminal que é utilizado pelo analisador sintático para começar a validação de um programa.

A seguir apresenta-se uma gramática da linguagem LSI, em notação BNF. A gramática é construída utilizando os símbolos não-terminais, listados acima, e os tokens, definidos na seção 2.1.1.

```

<programa> ::= programa id ";" <bloco> "." ;
<bloco> ::= <dcl_var_const> <dcl_metodos> <com_composto> ;
<dcl_var_const> ::= <tipo> <dimensao> <lid> <fator_const> ";" <dcl_var_const> | î ;
<tipo> ::= inteiro | real | booleano | caracter | cadeia ;
<dimensao> ::= "[" <constante> "]" | î ;
<lid> ::= id <rep_lid> ;
<rep_lid> ::= "," <lid> | î ;
<fator_const> ::= "=" <constante> | î ;
<dcl_metodos> ::= <dcl_metodo> ";" <dcl_metodos> | î ;
<dcl_metodo> ::= metodo id <par_formais> <tipo_metodo> ";" <bloco>;
<par_formais> ::= "(" <mp_par> <lid> ":" <tipo> <rep_par> ")" | î ;
<rep_par> ::= ";" <mp_par> <lid> ":" <tipo> <rep_par> | î ;
<tipo_metodo> ::= ":" <tipo> | î ;
<mp_par> ::= ref | val;
<com_composto> ::= "{"<comando> <replistacomando> "}";
<replistacomando> ::= ";" <comando> <replistacomando> | î ;
<comando> ::= id <rcomid>
| <com_composto>
| se <expressao> entao <comando> <senaoparte>
| enquanto <expressao> faca <comando>
| leia "(" <lid> ")"
| escreva "(" <expressao> <rep_lexpr> ")"
| tamanho_cadeia "(" <expressao> ")"
| caracter_posicao "(" <expressao> "," <expressao> ")"
| posicao_subcadeia "(" <expressao> "," <expressao> ")"
| retorne <expressao>
| î;
<senaoparte> ::= senao <comando> | î;
<rcomid> ::= "!=" <expressao> | "[" <expressao> "]" "!=" <expressao> | "("
<expressao> <rep_lexpr> ")" | î;
<rep_lexpr> ::= "," <expressao> <rep_lexpr> | î;
<expressao> ::= <expsimp> <resto_expressao>;
<resto_expressao> ::= <oprel> <expsimp> | î;
<oprel> ::= "=" | "<" | ">" | ">=" | "<=" | "<>";
<expsimp> ::= <termo> <rep_expsimp>;

```

```

<rep_expsimp> ::= <op_add> <termo> <rep_expsimp> | î;
<op_add> ::= "+" | "-" | ou;
<termo> ::= <fator> <rep_termo>;
<rep_termo> ::= <op_mult> <fator> <rep_termo> | î;
<op_mult> ::= "*" | "/" | e ;
<fator> ::= nao <fator> | "-" <fator> | "(" <expressao> ")" | id <rvar> |
<constante_explicita> ;
<rvar> ::= "(" <expressao> <rep_lexpr>)" | "[" <expressao> "]" | î ;
<constante> ::= id | <constante_explicita>;
<constante_explicita> ::= num_int | num_real | falso | verdadeiro | literal_cadeia;

```

Na notação acima, o caracter "î" é utilizado como Épsilon (sentença vazia).

### 2.1.3. Especificação semântica

A semântica da linguagem LSI é determinada de forma semi-formal, por meio de linguagem natural e Ações Semânticas, em razão das dificuldades encontradas nas especificações formais existentes. [FURTADO, 2010].

Ações semânticas são uma forma bastante interessante de implementar a semântica de uma linguagem. No GALS, essas ações são implementadas adicionando o caracter # seguido de um número no meio da gramática BNF. Por exemplo:

```

<tipo> ::= inteiro #105 | real #106 | booleano #107 | caracter #108 | cadeia #109;

```

No compilador, essas ações irão gerar a chamada de um método do analisador semântico, passando como parâmetro o último token lido do programa e o número da ação. Assim, o analisador semântico será capaz de determinar se este token é válido ou não na posição em que se encontra. No exemplo, as ações 105 a 109 dizem ao analisador semântico qual é o tipo da variável sendo utilizada no momento.

Abaixo segue uma lista de regras semânticas que devem ser garantidas no compilador da LSI. Novamente, reiteramos que é apenas um exemplo, e que outras linguagens podem ter regras semânticas totalmente diferentes.

## **Declaração e uso de identificadores:**

1. O identificador do programa não poderá ter nenhum outro uso.
2. Todos os identificadores usados no programa devem ser previamente declarados.
3. Identificadores não podem ser declarados mais de uma vez no mesmo nível (independentemente de sua categoria).
4. Os identificadores só terão validade no escopo (nível) onde foram declarados e nos escopos (níveis) internos a ele.
5. Identificadores declarados com valor inicial, devem ser considerados como constantes.
6. Arrays e cadeias não podem ter valor inicial.
7. A constante usada na inicialização de um identificador deve ser do mesmo tipo do identificador;
8. Identificadores devem ser usados de acordo com a categoria na qual foram declarados, ex.:
  1. id de variável só pode ser usado em contextos que aceitem variáveis,
  2. id de método sem tipo só pode ser usado no comando de chamada de procedimento
  3. id de constante não pode ser usado no lado esquerdo de uma atribuição nem em comandos de leitura
  4. id de método com tipo só pode ser usado em expressões.
9. Variáveis do tipo Array só podem ser usados de forma indexada e o índice deve ser uma expressão inteira;
10. O tamanho de um Array (número de elementos) deve ser estabelecido por uma constante de tipo inteiro.
11. Arrays não podem ser do tipo cadeia de caracteres.
12. Somente variáveis do tipo Array podem ser indexadas.
13. Identificadores de parâmetros podem ser usados em todos os contextos onde for válido o uso de variáveis.
14. Parâmetros devem ser de tipo pré-definido.



15. Constantes literais de tamanho 1 devem ser consideradas como sendo do tipo caracter.

**Comandos "SE" e "ENQUANTO":**

16. A expressão de controle deve ser booleana ou inteira (neste caso, se = 0 considera-se falso, senão verdadeiro).

**Comandos "LEIA" e "ESCREVA":**

17. Os identificadores a serem lidos devem ser das categorias variável ou parâmetro e devem ser de tipo pré-definido (exceto booleano).
18. As expressões a serem impressas deverão ser de tipo pré-definido ou cadeia (literal).

**Chamada de Método:**

19. Deve haver correspondência em número e tipo entre parâmetros atuais e formais.
20. Parâmetros formais por referência devem ter um id (da categoria variável ou parâmetro) como parâmetro atual correspondente.
21. Id de métodos sem tipo (procedimentos) só podem ser usados como comandos.
22. Métodos sem tipo (procedimentos) não devem possuir comando de retorno.
23. Id de métodos com tipo (funções) só podem ser usados em expressões.
24. Todo método com tipo (função) deve possuir pelo menos um comando de retorno (cláusula retorne).
25. O tipo da expressão usada na cláusula de retorno deve ser pré-definido e compatível com o tipo do método.

**Comando de atribuição:**

26. O id do lado esquerdo deve ser uma variável (simples ou array indexado) ou um parâmetro.
27. O tipo da expressão deve ser igual ao tipo do id usado no lado esquerdo da atribuição. Exceções: REAL aceita INTEIRO e CADEIA aceita CHARACTER.

**Operadores relacionais:**

28. Os operandos envolvidos devem ser do mesmo tipo (exceto real / inteiro e cadeia / caracter).

**Operadores aritméticos:**

29. Podem ser aplicados a operandos inteiros e reais (qualquer combinação).

30. Expressões numéricas mistas (envolvendo inteiros e reais) resultarão em tipo real.

31. O uso do operador “/” entre inteiros resultará em tipo real.

32. O operador unário “-” não podem ser usado consecutivamente.

**Operadores lógicos:**

33. Os operandos envolvidos devem ser booleanos.

34. O operador “não” não pode ser usado consecutivamente.

**Operandos constantes:**

35. Literais (cadeias de caracteres) de tamanho 1 devem ser considerados constantes do tipo caracter.

36. Falso é menor que Verdadeiro.

**Operandos variáveis:**

37. Devem ser de tipo pré-definido, cadeia ou elemento de um Array.

38. Identificadores da categoria “parâmetro” devem ser vistos como variáveis.

39. Variáveis do tipo Array devem ser referenciadas de forma indexada e a expressão usada como índice deve ser do tipo inteiro. Para verificação de tipo do operando, vale o tipo dos elementos (sempre um tipo pré-definido).

**Operandos que são identificadores de métodos:**

40. O tipo do retorno do método deve ser compatível no contexto em que o método está sendo chamado.

### 3. A Máquina Virtual da Linguagem de Sistemas de Informação

Nesta seção será apresentada a Máquina Virtual implementada e suas características.

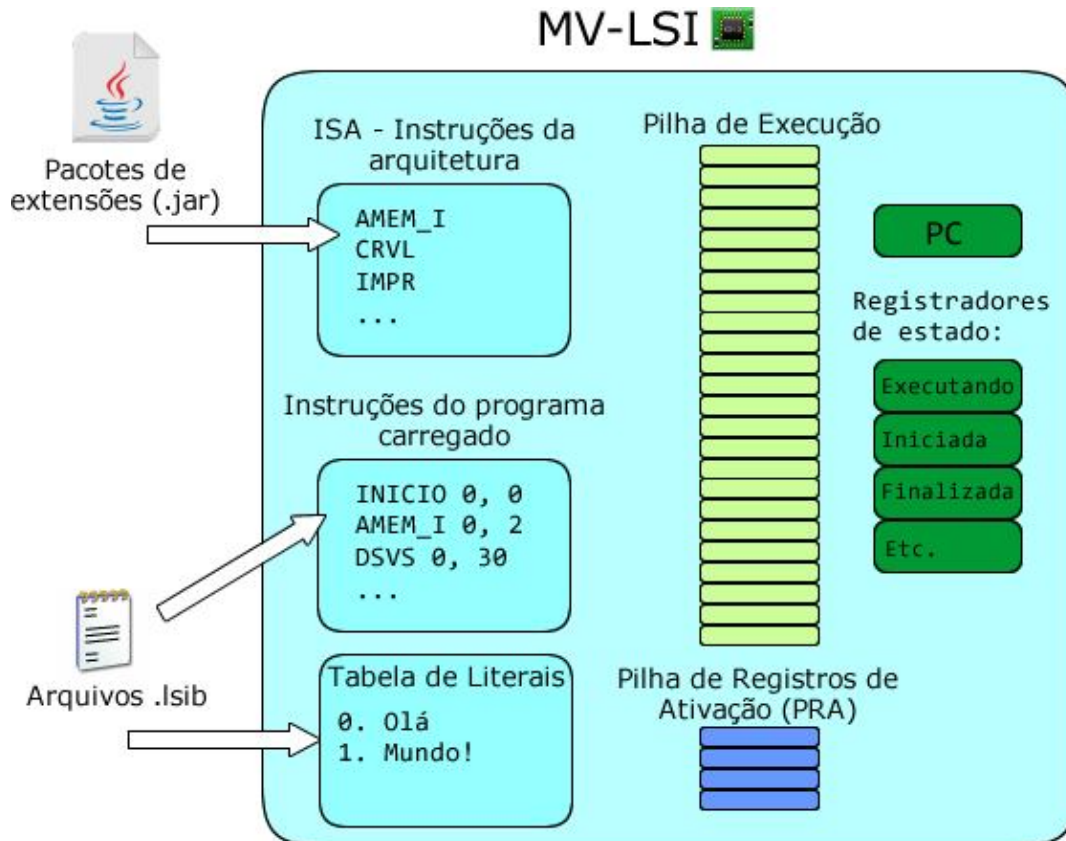
A MV-LSI é uma máquina virtual de aplicação (conforme detalhado na seção 1.1.3), pois nada mais é do que uma aplicação que executará como um processo comum em um sistema operacional já existente, e é implementada por meio da emulação de um hardware virtual (seção 1.2.2).

#### 3.1. Arquitetura da MV-LSI

A arquitetura da MV-LSI é relativamente simples e de fácil compreensão. Basicamente, é composta pelos seguintes itens:

- Tabela de instruções do programa
- Tabela de instruções disponíveis (corresponde ao ISA da MV)
- Tabela de literais: uma área da memória para armazenamento de literais. Literais são tipos de dados mais complexos, como cadeias de caracteres, que podem requerer muito espaço em memória para serem armazenados.
- Pilha de execução: o principal componente da MV. É onde tudo acontece, desde armazenamento de variáveis até chamada e retorno de métodos.
- PC: *program counter*, indica qual a próxima instrução a ser executada.
- PRA: pilha de registros de ativação, aponta o endereço base dos métodos em execução;
- Variáveis de controle: indicam se a MV está iniciada, finalizada, travada, parada ou no meio da execução de uma instrução.

A figura 9, a seguir, ilustra a arquitetura da MV-LSI:



**Figura 9: Arquitetura da MV-LSI**

Toda a arquitetura está implementada no chamado Núcleo da Máquina Virtual, que é abordado na seção 4.1.

Como diferenças em relação à arquitetura da JVM, podemos listar as seguintes:

- A MV-LSI não permite o carregamento de código externo, logo não é necessário um tradutor de códigos nativos.
- Não há um compilador JIT, todo o código é interpretado.
- A MV-LSI não suporta threads e concorrência.
- A JVM suporta o paradigma de orientação a objetos, já MV-LSI, não.
- No momento não é possível criar bibliotecas de métodos ou mesmo separar um programa em vários arquivos.

### 3.2. Tipos de dados

Por "tipo de dado", neste trabalho, deve ser utilizado o conceito de espaço ocupado na pilha de execução. Um "dado" deve ocupar apenas um espaço da pilha de execução. Como consequência dessa interpretação, os *arrays*, da forma como foram implementados no pacote padrão, não correspondem a um tipo de dado. Isto porque cada espaço do *array* possui um espaço correspondente na pilha de execução. As instruções que lidam com *arrays* devem conhecer a forma como estes são guardados na pilha.

Da forma como a MV-LSI foi desenvolvida, existe a possibilidade de criação de tipos de dados por parte do usuário. No pacote padrão estão definidos 6 tipos de dados:

- Booleano
- Caracter
- Cadeia de caracteres (*string*)
- Endereço
- Inteiro
- Real

Todos os tipos, com exceção do tipo "Endereço", também são definidos na gramática da linguagem de exemplo deste trabalho. O tipo Endereço é utilizado internamente, durante a passagem e parâmetros por referência na chamada de métodos.

Os tipos citados acima são abordados com mais detalhes na seção 4.3.1.

### 3.3. Especificação do formato do arquivo executável

O compilador da máquina virtual LSI deverá gerar um arquivo executável em binário, seguindo a especificação abaixo, na ordem em que é apresentada:

Offset	Tamanho em bytes	Descrição
0	4	Número mágico do arquivo executável da MV-LSI. 0x4C534942 (Significa

		LSI-Binary)
4	1	Versão do formato do arquivo
5	tam_const [variável]	Tabela de constantes
5 + tam_const	tam_met [variável]	Tabela de métodos

Tabela de constantes literais:

Offset	Tamanho em bytes	Descrição
0	4	Número de registros (java int)
4	4	Tamanho da primeira constante (java int)
8	1	Tipo da primeira constante
9	tam_reg1 [variável]	Valor da primeira constante
9 + tam_reg1	4	Tamanho da segunda constante
9 + tam_reg1 + 4	1	Tipo da segunda constante
9 + tam_reg1 + 4 + 1	tam_reg2 [variável]	Valor da segunda constante
....		

Obs: os tamanhos de constantes e o número de registros são armazenados no formato "integer" do Java (inteiro de 32 bits com sinal - complemento de dois).

Tabela de Instruções:

Offset	Tamanho em bytes	Descrição
0	4	Número de instruções (java int)
4	10	Instrução 1
14	10	Instrução 2
24	10	Instrução 3
34	10	Instrução x
....		

Formato de uma instrução:

Instrução	Parâmetro #1	Parâmetro #2
2 bytes	4 bytes	4 bytes

Os dois primeiros bytes de uma instrução devem ser no formato java *short* (inteiro de 16 bits com sinal na forma de complemento de dois). Os 8 bytes restantes podem ser usados de duas formas: como dois parâmetros do tipo java int, com 4 bytes cada um, ou como um único parâmetro de 64 bits (8 bytes), correspondendo a um valor no tipo java *double* (ponto flutuante de dupla precisão, padrão IEEE 754). Caso um dos parâmetros ou nenhum parâmetro seja utilizado, esse espaço deverá ser preenchido com zeros.

### 3.4. Repertório de instruções

As instruções básicas da MV-LSI estão listadas abaixo. Todas elas estão detalhadas na seção 4.3. Esse repertório foi implementado como um pacote separado, como se fosse uma extensão. Só que ele é fundamental para a execução de qualquer programa. A forma de converter sentenças da linguagem em sequências de instruções é abordada na seção 3.5.

Mnemônico	Descrição
AMEM_B	Aloca espaços para booleanos
AMEM_C	Aloca espaços para caracteres
AMEM_I	Aloca espaços para inteiros
AMEM_R	Aloca espaços para reais
AMEMCAD	Aloca espaços para cadeias *
DMEM	Remove espaços da pilha
ARMZ	Armazenamento direto
ARMZIND	Armazenamento indireto
AVET	Armazenamento direto em array
CRVL	Copia um item da pilha para o topo
CRVLIND	Copia um item indireto da pilha para ao topo

CREN	Carrega um endereço na pilha
CRCT_B	Carrega um booleano constante
CRCT_C	Carrega um caracter constante
CRCT_I	Carrega um inteiro constante
CRCT_R	Carrega um real constante
CRCAD	Carrega uma cadeia no topo da pilha
CVET	Carrega um valor de um array
LEIA_C	Lê um caracter
LEIA_I	Lê um número inteiro
LEIA_R	Lê um número real
LEIACAD	Lê uma cadeia de caracteres ( <i>string</i> )
IMPR	Imprime o que estiver no topo da pilha
IMPRIT	Imprime um literal
CONJ	Conjunção lógica
DISJ	Disjunção lógica
NEGA	Negação
SOMA_I	Soma números inteiros
SOMA_R	Soma números reais
SUB_I	Subtrai números inteiros
SUB_R	Subtrai números reais
MULT_I	Multiplica inteiros
MULT_R	Multiplica reais
DIV	Divisão de números reais
MUN_I	Menos unário para inteiros
MUN_R	Menos unário para reais
CMIG	Compara se valores são iguais
CMDF	Compara se valores são diferentes
CMMA	Compara se valor é maior
CMME	Compara se valor é menor
CMEI	Compara se valor é menor ou igual
CMAI	Compara se valor é maior ou igual
CMIGCAD	Compara se duas cadeias são iguais
CMDFCAD	Compara se duas cadeias são diferentes
I2R	Inteiro para real
I2C	Inteiro para caracter
I2B	Inteiro para booleano
I2CAD	Inteiro para cadeia de caracteres
R2I	Real para inteiro
R2C	Real para caracter
R2B	Real para booleano
R2CAD	Real para cadeia de caracteres
B2I	Booleano para inteiro
B2R	Booleano para real
B2CAD	Booleano para cadeia de caracteres
C2I	Caracter para inteiro
C2CAD	Caracter para cadeia de caracteres
CAD2I	Cadeia de caracteres para inteiro
CAD2C	Cadeia de caracteres para caracter



CAD2R	Cadeia de caracteres para real
DSVS	Desvia sempre
DSVF	Desvio condicional
CALL	Chamada de método
RETU	Retorno de método
TAMCAD	Calcula o tamanho da cadeia
CONCAT	Concatenação de duas cadeias de caracteres
POSCAD	Informa a posição de uma subcadeia
SUBCAD	Retorna uma subcadeia
CHARAT	Retorna um caracter em uma posição
INICIO	Indica o inicio do programa
FIM	Indica o fim do programa
NADA	Não faz nada

### 3.5. Geração de código intermediário

A geração de código intermediário para a MV-LSI pode ser feita juntamente com a Análise Semântica no compilador, por meio do mesmo mecanismo de Ações explicado na seção 2.1.3. Ao validar o aspecto semântico de um trecho de código, o compilador imediatamente gera o código correspondente a esse trecho.

Nesta seção especificaremos quais instruções devem ser geradas para cada construção da gramática da linguagem.

#### 3.5.1. Início do programa

Todo código deve começar com a instrução abaixo:

```
INICIO 0 0
```

#### 3.5.2. Declaração de variáveis

Para cada variável declarada, deve-se alocar um espaço na pilha de execução. A instrução utilizada depende do tipo da variável. O segundo parâmetro das instruções denota a quantidade de espaços a serem alocados. Então, em uma declaração como "inteiro a,b,c;", podemos utilizar apenas uma instrução, só que utilizando o segundo parâmetro com o valor 3.

Os códigos abaixo alocam um espaço na pilha para cada tipo de dado:

- Booleano: AMEM\_B 0 1
- Caracter: AMEM\_C 0 1
- Inteiro: AMEM\_I 0 1
- Real: AMEM\_R 0 1
- Cadeia: AMEMCAD 0 1

Após fazer a alocação, é preciso que o compilador armazene a posição da pilha correspondente ao espaço que foi alocado (nível e deslocamento), para que, quando essas variáveis forem utilizadas, seja possível saber onde elas estão armazenadas. Por exemplo, a primeira variável do nível zero (programa), estará no nível 0, deslocamento 0. A segunda, nível 0, deslocamento 1.

Suponha o trecho de código abaixo:

```
programa exemplo_declaracoes;
    inteiro A, B;
    inteiro [5] C;
    inteiro D;

    metodo exemplo;
        inteiro E;
    { }
{ }.
```

Os identificadores A, B, C, D e E possuirão os seguintes níveis e deslocamentos:

- A: nível 0, deslocamento 0
- B: nível 0, deslocamento 1
- C: nível 0, deslocamento 2
- D: nível 0, deslocamento 8 \*
- E: nível 1, deslocamento 0

\* conforme a seção 3.5.4 abaixo, arrays consomem n+1 espaços na pilha de execução.

### 3.5.3. Declaração de constantes

Em relação às constantes, não faz sentido existir código intermediário. Constantes devem ser automaticamente substituídas pelo próprio compilador quando forem utilizadas.

### 3.5.4. Declaração de arrays

Um array é uma estrutura que armazena diversos elementos do mesmo tipo na pilha de execução, e é referenciado de forma indexada.

Na pilha de execução da MV-LSI, os arrays ocuparão  $n+1$  espaços, onde  $n$  é o número de elementos do array. O espaço na pilha para o qual o identificador do array aponta deve conter o número de elementos do array. Essa informação é necessária para que as instruções relacionadas à arrays possam impedir tentativas de acessar uma posição inválida, por exemplo.

Ao encontrar uma declaração de array, deve-se utilizar CRCT\_I para guardar o número de posições, e alguma das instruções AMEM\_X, dependendo do tipo dos elementos do array. Por exemplo, em um array inteiros com 8 posições, o código gerado seria:

```
CRCT_I 0 8  
AMEM_I 0 8
```

### 3.5.5. Declaração de métodos

Ao encontrar uma declaração de método, o compilador deve gerar uma instrução de desvio incondicional (DSVS), pois um método só pode ser executado por meio da instrução CALL.

```
DSVS 0 X
```

Onde X corresponde a posição da instrução imediatamente após o término do método. A posição é contada a partir de 0 (sendo assim a instrução INICIO estará na posição 0).

Em seguida, o compilador deve armazenar em um local a posição de início do método, que será usada quando esse método for chamado. Depois, caso o método possua parâmetros formais, o compilador deverá guardar, para cada identificador de parâmetro, o nível e deslocamento deles, além do tipo e forma de passagem (valor ou referência). O deslocamento dos parâmetros será sempre negativo, calculado da seguinte forma:

$$-(N + 2 - i)$$

Onde 'N' é o número de parâmetros desse método e 'i' é a ordem do parâmetro, iniciada em 0.

Considere o código abaixo para exemplificarmos:

```
programa declaracao_metodo;
    metodo exemplo_metodo (val par1 : inteiro; val par2 :
        inteiro);
    { }
{ }.
```

Os níveis e deslocamentos dos parâmetros par1 e par2 seriam:

- par1: nível 1, deslocamento -4
- par2: nível 1, deslocamento -3

O motivo de usarmos deslocamentos negativos é simples: os parâmetros são armazenados na pilha antes da chamada do método com a instrução CALL. Para mais detalhes, veja a seção 4.3.11. Os valores armazenados nas posições -2 e -1 de cada nível são utilizados pelas instruções CALL e RETU para controlar os contextos de chamadas de métodos.

Se o método possuir um tipo de retorno, esse tipo deve ser armazenado também, para ser utilizado quando o método for chamado.

### **3.5.6. Comando de atribuição - variável ou parâmetro**

O comando de atribuição utilizado na linguagem LSI é o ':='. Logo, o identificador que está antes desse comando é que receberá o valor que estiver após o comando. Esse identificador pode ser uma variável qualquer, um identificador de parâmetro ou uma posição de um array (abordado na seção 3.5.7).

A instrução utilizada para armazenar um valor em uma variável (posição na pilha de execução) é ARMZ. A seção 4.3.3 detalha o funcionamento da instrução.

Assim, após a avaliação total da expressão (que deve terminar com apenas um valor no topo da pilha) que estiver no lado direito do comando ':=', esta instrução deverá ser chamada:

```
ARMZ N X
```

Onde N é o nível do identificador, e X o deslocamento. Esta instrução é capaz de funcionar com qualquer tipo de dado, mas ela irá verificar se o tipo do resultado da expressão do lado direito é igual ao tipo do identificador utilizado no lado esquerdo.

Exemplo:

```
programa atribuicao_exemplo;
    inteiro var1;
    {
        var1 := 10;
    }.

```

O código gerado para o programa será:

```
INICIO 0 0
AMEM_I 0 1 // alocação de espaço para var1
CRCT_I 0 10 // carrega a constante 10
ARMZ 0 0 // armazena na posição PRA[0]+0, que corresponde a var1
FIM 0 0

```

### 3.5.7. Comando de atribuição - arrays

O comando de atribuição em posições de arrays utiliza a instrução AVET, ao invés de ARMZ. Veja a seção 4.3.3 para conhecer o funcionamento da instrução.

O índice utilizado deve corresponder a uma expressão que resultará em um valor inteiro no topo da pilha. No exemplo abaixo essa expressão é a constante 5, mas poderia ser uma expressão matemática complexa, que poderia inclusive incluir chamadas para métodos).

Exemplo:

```
meuArray[5] := 10;
```

Considere que meuArray é um array de inteiros com 6 ou mais posições. O código gerado para esse comando será:

```
CRCT_I 0 5  
CRCT_I 0 10  
AVET N X
```

A primeira instrução indica o índice do array; a segunda instrução é o valor que será armazenado, e a terceira, armazena o valor no array localizado no nível N e deslocamento X.

### 3.5.8. Comando "se"

O comando "se" utiliza um valor booleano para determinar o fluxo de execução. Assim, a expressão utilizada na avaliação deve resultar em um item do tipo booleano no topo da pilha. A instrução DSVF (seção 4.3.10) é a que desviará o fluxo (para o fim do "se" ou para a cláusula "senao", se existir).

Exemplo:

```
se (2+2 = 2*2) entao {  
    escreva("2+2 é igual a 2*2");  
}
```

A expressão "2+2=2\*2" resulta em um valor booleano, por isso pode ser usado na cláusula "se".

O código gerado desse exemplo será:

```
10. ... (instruções dos comandos antes do "se")
11. CRCT_I 0 2 }
12. CRCT_I 0 2 } <expressao> do comando "se"
13. SOMA_I 0 0 }
14. CRCT_I 0 2 }
15. CRCT_I 0 2 }
16. MULT_I 0 0 }
17. CMIG 0 0 }
18. DSVF 0 20
19. IMPRLIT 0 X // ver seção 4.3.5
20. ... (instruções dos comandos após o "se")
```

No caso do comando "se" possuir um "senao", deve ser gerada uma instrução DSVS ao final do bloco do "se", apontando para instrução imediatamente após o final do bloco "senao".

Exemplo:

```
se(5 > 4+1) entao {
    escreva("5 é maior que 4+1!");
} senao {
    escreva("5 não é maior que 4+1");
}
```

O código gerado para o trecho acima é:

```
10. ... (instruções dos comandos anteriores ao "se")
11. CRCT_I 0 5 }
12. CRCT_I 0 4 } <expressao> do comando "se"
13. CRCT_I 0 1 }
14. SOMA_I 0 0 }
15. CMMA 0 0 }
16. DSVF 0 19
17. IMPRLIT 0 X
```

```
18. DSVS 0 20
19. IMPRLIT 0 Y
20. ... (instruções dos comandos posteriores ao "se")
```

Se a expressão do "se" for falsa, a instrução DSVF desviará o fluxo para a instrução 19 (correspondente à primeira instrução do "senao"). Se a expressão fosse verdadeira, DSVF não mudaria o fluxo, a instrução 17 seria executada, e em seguida, DSVS desviaria o fluxo para a primeira instrução após o "senao", que no caso é a instrução 20.

### 3.5.9. Comando "enquanto"

O comando enquanto é a única estrutura de *loopings* da linguagem. Equivale ao "while" das linguagens tradicionais. Seu funcionamento é semelhante ao comando "se", com a diferença que ao final do bloco de comandos, o fluxo é deslocado novamente para a expressão de avaliação.

Exemplo:

```
programa exemplo_enquanto;
  inteiro x;
  {
    x := 10;
    enquanto(x >= 0) faca {
      x := x - 1;
    }
  }.

```

O programa acima gera o seguinte *bytecode*:

```
0. inicio 0 0
1. amem_i 0 1
2. crct_i 0 10
3. armz 0 0
4. crvl 0 0
5. crct_i 0 0 } <expressao> do comando "enquanto"
6. cmai 0 0
7. dsvf 0 13
8. crvl 0 0
9. crct_i 0 1 } instruções do comando x := x - 1;
10. sub_i 0 0
11. armz 0 0
12. dsvs 0 4
13. fim 0 0
```



### 3.5.10. Comando de leitura e escrita de dados

O comando "leia" da linguagem permite a leitura de dados (com entrada pelo teclado, por exemplo), e pode receber indefinidos parâmetros. Todos os parâmetros devem ser identificadores de variáveis, nas quais serão armazenados os valores lidos, de forma sequencial.

Da mesma forma, o comando "escreva" também pode receber indeterminados parâmetros, só que sem a restrição de serem identificadores de variáveis. Um parâmetro do comando escreva por ser uma expressão matemática, por exemplo. Quando o comando escreva recebe um literal como parâmetro, a instrução utilizada é IMPRLIT, onde o segundo parâmetro corresponde ao índice do literal na tabela de literais (esse índice é iniciado em 0, e é criado a partir da leitura sequencial do programa). Já a instrução IMPR é a utilizada nos demais casos (ela consegue trabalhar com todos os tipos de dados). Veja a seção 4.3.5 para mais detalhes sobre essas instruções.

Por exemplo:

```
programa exemplo_leia;
    inteiro idade;
    cadeia nome;
    real altura;
    caracter sexo;

    {
        escreva("Digite a sua idade: ");
        leia(idade);
        escreva("Digite seu nome: ");
        leia(nome);
        escreva("Digite sua altura, em metros: ");
        leia(altura);
        escreva("Informe o sexo (m/f): ");
        leia(sexo);
    }.

```

O código acima gera o seguinte bytecode:

Literais:

0. 'Digite a sua idade: '
1. 'Digite seu nome: '
2. 'Digite sua altura, em metros: '
3. 'Informe o sexo (m/f): '

Instruções:

```
0. inicio 0 0
1. amem_i 0 1
2. amemcad 0 1
3. amem_r 0 1
4. amem_c 0 1
5. imprlit 0 0
6. leia_i 0 0
7. armz 0 0
8. imprlit 0 1
9. leiacad 0 0
10. armz 0 1
11. imprlit 0 2
12. leia_r 0 0
13. armz 0 2
14. imprlit 0 3
15. leia_c 0 0
16. armz 0 3
17. fim 0 0
```

} declaração das variáveis

### 3.5.11. Chamada de métodos

A chamada de métodos é feita por meio da instrução CALL (detalhes de seu uso em 4.3.11). A instrução é chamada utilizando informações guardadas durante a declaração do método (seção 3.5.5): nível e instrução de início.

Exemplo:

```
programa exemplo_chamada_metodo;
  metodo exemplo_chamada;
  {
    escreva("Ola Mundo!");
  }
  {
    exemplo_chamada;
  }.
}
```

O bytecode gerado será:

```
0. inicio 0 0
1. dsvs 0 4
2. imprlit 0 0
3. retu 1 0
4. call 1 2
5. fim 0 0
```

Quando o método possui retorno, uma instrução AMEM\_X deverá ser chamada antes da instrução CALL (e antes da carga de parâmetros também, se houver), que será o espaço usado para guardar o valor de retorno. A seção 3.5.13 possui um exemplo de um método com retorno, que também ilustra a chamada de métodos com parâmetros.

### 3.5.12. Passagem de parâmetros por referência

Ao passar um parâmetro por referência, estaremos passando para o método apenas o endereço de um identificador, e não o seu valor. A instrução utilizada para isto é CREN, ao invés de CRVL. Dentro do corpo do método, é necessário utilizar a instrução CRVLIND e ARMZIND, para carregar e armazenar, respectivamente, valores de um parâmetro que foi passado por referência.

Exemplo:

```
programa passagem_referencia;
  inteiro x;
  metodo eleva_ao_quadrado(ref numero : inteiro);
  {
    numero := numero * numero;
  }
  {
    escreva("Digite x: ");
    leia(x);
    eleva_ao_quadrado(x);
    escreva("X^2: ", x);
  }.
}
```

Literais:

```
0. 'Digite x: '
1. 'X^2: '
```

Instruções:

```
0. inicio 0 0
1. amem_i 0 1 // aloca espaço para x
2. dsvs 0 8 // desvia para o corpo do programa
3. crvlind 1 -3 // carrega indiretamente o valor de 'numero'
4. crvlind 1 -3 // carrega indiretamente o valor de 'numero'
5. mult_i 0 0 // multiplica
6. armzind 1 -3 // armazena indiretamente o valor de 'numero'
7. retu 1 1 // retorna do metodo
8. imprlit 0 0 // imprime 'Digite x: '
9. leia_i 0 0 // lê número inteiro
10. armz 0 0 // armazena em x
11. cren 0 0 // carrega endereço de x na pilha
12. call 1 3 // chama método eleva_ao_quadrado
13. imprlit 0 1 // imprime 'X^2: '
14. crvl 0 0 // carrega x na pilha
15. impr 0 0 // imprime x
16. fim 0 0
```

### 3.5.13. Comando de retorno de método

O encerramento da execução de métodos deve ser feita com a instrução RETU, mesmo em métodos que não possuam retorno! Isto é necessário para a MV reconstruir o contexto anterior ao da chamada do método.

No caso do método possuir várias cláusulas "retorne", deverão ser usadas várias instruções RETU também, uma para cada caso. Para mais detalhes sobre essa instrução, leia a seção 4.3.11.

O valor ou expressão utilizado na cláusula de retorno deve ser armazenado (instrução ARMZ) na pilha de execução, na posição calculada por:

- (N + 3)

Onde N é o número de parâmetros do método. Assim, em um método sem parâmetros e no nível léxico 1, a instrução de armazenamento do retorno seria

```
ARMZ 1 -3
```

Exemplo:

```
programa exemplo_retorno;
    inteiro x;
    metodo eleva_ao_quadrado(val numero : inteiro) : inteiro;
        {
            retorne numero * numero;
        }
    {
        escreva("Digite x: ");
        leia(x);
        x := eleva_ao_quadrado(x);
        escreva("X^2: ", x);
    }.
}
```

O *bytecode* gerado para esse programa será:

Literais:

```
0. 'Digite x: '
1. 'X^2: '
```

Instruções:

```
0. inicio 0 0
1. amem_i 0 1 // aloca espaço para x
2. dsvs 0 8 // desvia para o corpo do programa
3. crvl 1 -3 // carrega o parâmetro 'numero'
4. crvl 1 -3 // carrega o parâmetro 'numero' novamente
5. mult_i 0 0 // multiplica os dois valores
6. armz 1 -4 // armazena na posição de retorno
7. retu 1 1 // retorna do método
8. imprlit 0 0 // imprime 'Digite x: '
9. leia_i 0 0 // lê um inteiro
10. armz 0 0 // armazena o inteiro em x
11. amem_i 0 1 // aloca espaço para retorno do método
12. crvl 0 0 // carrega x como parâmetro do método
13. call 1 3 // chama o método eleva_ao_quadrado
14. armz 0 0 // armazena o resultado do método em x
15. imprlit 0 1 // imprime 'X^2: '
16. crvl 0 0 // carrega x
17. impr 0 0 // imprime x
18. fim 0 0
```

No caso do método possuir variáveis locais, elas deverão ser desalocadas com a instrução DMEM, antes da instrução RETU. Exemplo:

```

programa exemplo_retorno2;
  metodo desaloca_local : inteiro;
    inteiro var_local;
    {
      var_local1 := 10;
      retorne var_local;
    }
  {
    escreva(desaloca_local);
  }.

```

*Bytecode:*

```

0. inicio 0 0
1. dsvs 0 9 // desvia para o corpo do programa
2. amem_i 0 1 // aloca espaço para var_local
3. crct_i 0 10 // carrega constante 10
4. armz 1 0 // armazena em var_local
5. crvl 1 0 // carrega var_local
6. armz 1 -3 // armazena no espaço de retorno do metodo
7. dmem 0 1 // desaloca espaço de var_local
8. retu 1 0 // retorna do método
9. amem_i 0 1 // aloca espaço para retorno do método
10. call 1 2 // chama o método desaloca_local
11. impr 0 0 // imprime o resultado da chamada do método
12. fim 0 0

```

### 3.5.14. Expressões lógicas, matemáticas e de comparação

Todos os operadores de comparação (maior, menor, igual, diferente, maior igual, menor igual), matemáticos (soma, subtração, divisão e multiplicação - exceto "menos unário") e lógicos (AND e OR, exceto o operador de negação) utilizam dois valores. Então, para gerar código para essas expressões, é necessário carregar no topo da pilha os dois valores necessários para as instruções (detalhes em 4.3.6, 4.3.7 e 4.3.8).

Exemplo 1:

```
x := 10 + (5 * 8);
```

Gera a sequência de bytecodes:

```

crct_i 0 10 //carrega o primeiro valor para a instrução soma_i
crct_i 0 5 //carrega o primeiro valor para a instrução mult_i
crct_i 0 8 //carrega o segundo valor para a instrução mult_i
mult_i 0 0 //multiplicação

```

```
soma_i 0 0 //utiliza o resultado de mult_i como segundo valor
armz 0 X
```

Exemplo 2:

```
bool := (2 * 6) >= (3 + 9) || falso;
```

Gera a sequência de instruções:

```
crct_i 0 2 }
crct_i 0 6 } 2 * 6
mult_i 0 0 }
crct_i 0 3 }
crct_i 0 9 } 3 + 9
soma_i 0 0 }
cmai 0 0 // aplica o operador ">=" ao resultado de 2*6 e 3+9
crct_b 0 0 // carrega o booleano falso na pilha
disj 0 0 // disjunção = ||
armz X Y // armazena em 'bool'
```

### 3.6. Otimizações de código

O código gerado por um compilador pode ter muitos trechos ineficientes, com blocos de instruções iguais repetidos, códigos inacessíveis, operações desnecessárias, etc.

Abaixo listamos algumas otimizações simples de código que podem ser feitas no bytecode.

#### Cálculo de expressões constantes

```
PI := 3.14;
raio := 25;
area := PI * (raio * raio);
```

O código acima, sem otimização, gera o bytecode:

```
...
CRCT_R 3.14
ARMZ pi
CRCT_I 0 25
ARMZ raio
CRVL pi
CRVL raio
CRVL raio
```

```
MULT_I 0 0
MULT_I 0 0
ARMZ area
...
```

Como "PI" e "raio" são conhecidos em tempo de compilação, o código pode ser simplificado para:

```
area := 1962.5;
```

Bytecode:

```
CRCT_R 1962.5
ARMZ area
```

### **Remoção de códigos inacessíveis (código morto)**

Códigos inacessíveis podem ser gerados de diversas formas, como condições SE ou ENQUANTO que sempre serão falsas, ou mesmo métodos inteiros que nunca são chamados. Todos esses trechos podem ser removidos sem problema algum.

### **Variáveis com apenas uma atribuição constante**

Quando variáveis recebem apenas uma atribuição, e esse valor é conhecido em tempo de compilação, pode-se converter a variável em constante e removê-la completamente do código final gerado.

### **Métodos que são chamados apenas uma vez**

O código de um método que é chamado apenas uma vez pode ser deslocado para o local onde é chamado, evitando a criação de novos "frames" na pilha de instruções, ocupando menos espaço na memória.

### **Movendo expressões constantes para fora de loopings**

Blocos de códigos dentro de loopings podem afetar negativamente a performance de um programa. Se esses trechos são constantes (realizam



sempre as mesmas operações, com os mesmos parâmetros e valores), podem ser colocados fora do loop e executados apenas uma vez.

### **Remoção de recursão de cauda**

Quando a última operação de um método antes de retornar for a chamada de outro (ou do mesmo método), é possível evitar a criação de um novo contexto na pilha de execução ("frame"), diminuindo o consumo de memória. Isso pode ser feito fazendo o método que está sendo chamado no final, retornar diretamente para quem chamou o primeiro método.

## **4. Implementação e uso da Máquina Virtual**

Todo o projeto foi desenvolvido utilizando a tecnologia Java, por meio da IDE NetBeans. Visando dar flexibilidade ao projeto, o mesmo foi dividido basicamente em três partes: núcleo da máquina virtual, interface gráfica e pacote de instruções. Cada parte corresponde a um pacote java (*package*).

Essa separação foi feita por diversos motivos:

- Com o núcleo da MV separado, ela poderá ser posteriormente integrada a outro projeto, como por exemplo, o compilador desenvolvido por um aluno, que passará a também executar o código compilado.
- A Interface gráfica desenvolvida aqui mostra diversos aspectos da máquina virtual em tempo de execução, como as instruções carregadas, pilha de execução, tabela de literais, registradores, entre outros. Outro desenvolvedor poderia adicionar mais características a serem exibidas, ou eliminar todas, deixando à mostra para o usuário apenas o resultado da execução do programa.
- As instruções foram separadas em um pacote para dar a liberdade do usuário definir quais serão as instruções que ele irá utilizar. Assim, o aluno pode criar seu próprio pacote de instruções, que complementam ou mesmo substituem as instruções do pacote padrão.

## 4.1. O núcleo da Máquina Virtual

O núcleo da máquina-virtual, responsável por efetivamente executar instruções, além de possuir as estruturas da arquitetura (seção 3.1), como a pilha de execução e registradores de controle. Encontra-se no pacote `MaquinaVirtual.jar` (apêndice B).

A classe principal do núcleo é a `MaquinaVirtual.java`. É uma classe que se estende à classe `Thread`, do próprio Java. Optou-se por essa alternativa pois assim a MV ficará independente da interface gráfica, sendo executada em sua própria *Thread*.

As estruturas da arquitetura são implementadas no núcleo. O registrador PC, por exemplo, é um atributo privado da classe `MaquinaVirtual.java`. A tabela de literais, ISA e a tabela de instruções do programa são implementadas utilizando `ArrayLists`.

A pilha de execução é implementada na classe `PilhaExecucao.java`. Ela se estende à classe `ObservableArrayList.java` (deste mesmo pacote - essa classe implementa métodos que permitem adicionar observadores a um `ArrayList`). Possui o registrador topo, que guarda o índice do último elemento válido na pilha de execução, além de implementar a PRA - pilha de registros de ativação - e os métodos necessários para utilizá-la.

Outras classes importantes também estão neste pacote:

- `ItemPilha.java`: é uma classe abstrata. Os Tipos de dados são classes que devem obrigatoriamente ser extensões desta classe.
- `Literal.java`: é uma classe abstrata. Tipos de dados literais (como cadeia de caracteres), devem ter uma classe que se estenda à classe `Literal` também.
- `MaquinaVirtualInterface.java`: uma interface (declara métodos que outras classes podem implementar) que declara os métodos de leitura, escrita e de controle que uma Interface Gráfica deve possuir.

- `Instrucao.java`: classe abstrata. Todas as instruções dos pacotes de extensões devem ser estendidas desta classe. Ela possui os métodos para acesso dos parâmetros da instrução, além de atributos para guardar o mnemônico e opcode.

Abaixo a lista de classes correspondentes à exceções, que podem ser geradas pela própria MV ou pelas instruções:

- `FormatoIncorretoException.java`: gerada pela MV quando não consegue abrir um arquivo LSIB.
- `IndiceInvalidoException.java`: pode ser usado genericamente quando algum "índice" não for válido. É usado, por exemplo, na instrução AVET quando o programa do usuário tenta armazenar um item em uma posição inválida de um Array.
- `InstrucaoNaoReconhecidaException.java`: gerada pela classe `MaquinaVirtual` quando o programa do usuário contém uma instrução não pertencente ao ISA da MV.
- `TipoLiteralNaoReconhecidoException.java`: gerado quando uma constante literal de um arquivo LSIB não é reconhecida pela máquina virtual.
- `InterrompidoException.java`: Esta exceção deve ser iniciada pelo controlador da MV (Interface Gráfica) quando a Thread da MV for interrompida com o método `interrupt()` - esse método fará a thread voltar a execução caso já não esteja. Deve ser usada para forçar uma parada da Máquina Virtual. Isso porque chamar o método `parar()` da MV não funcionará no caso desta estiver esperando entrada de dados do usuário (método `read()` do controlador da MV), por exemplo.
- `TipoIncorretoException.java`: usada por instruções quando o tipo do dado no topo da pilha não for o esperado.

## 4.2. Interface gráfica

O pacote `InterfaceMVLSI.jar` (apêndice C) possui todas as classes Java relacionadas com a exibição e controle da Máquina Virtual LSI. Com ela é possível carregar e executar programas, inserir e remover pacotes de extensões, entre outras ações (o manual de uso da interface está na seção 4.6).

A classe `InterfaceMVLSI.java` é a principal, pois corresponde ao frame principal do aplicativo e contém o método `main`. Também, é a classe que instancia a thread da Máquina Virtual e implementa os métodos da interface `MaquinaVirtualController.java` (do pacote `MaquinaVirtual.jar`), ou seja, esta classe é a responsável por enviar e receber dados e comandos da MV. Esta classe também é a responsável por fazer a carga das instruções ativadas na ISA da máquina virtual. Por isso ela precisa utilizar alguma forma de persistência (utilizou-se o `Java Preferences`<sup>8</sup>, que, no caso do Windows, utiliza o registro do sistema para armazenar quais pacotes estão carregados, quais instruções estão ativadas e quais foram os últimos programas abertos). Na figura 10 é mostrado o estado inicial da interface.

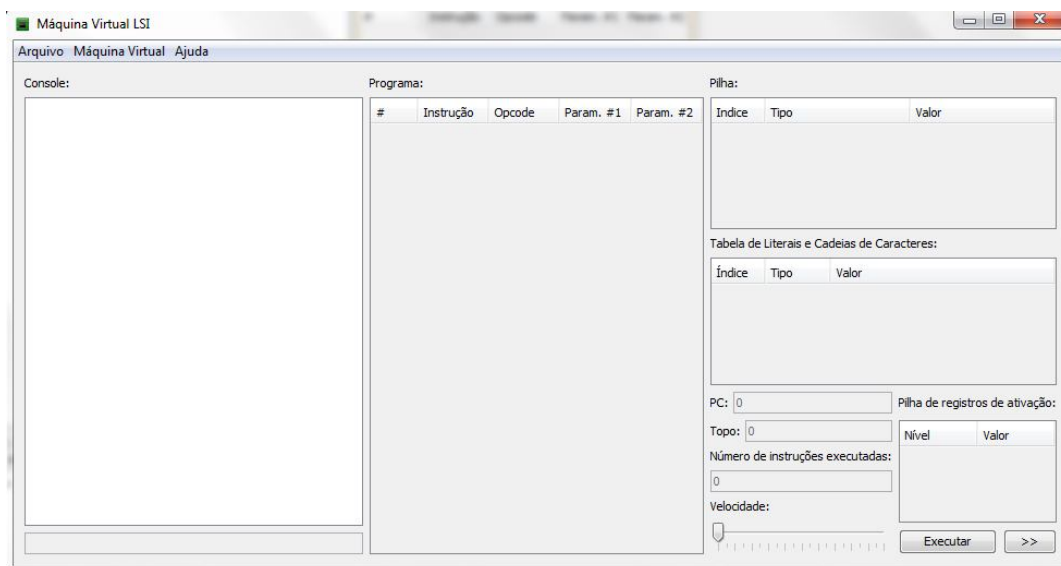


Figura 10: tela principal do aplicativo (`InterfaceMVLSI.java`)

<sup>8</sup> <http://docs.oracle.com/javase/7/docs/api/java/util/prefs/Preferences.html>

A classe `GerenciadorExtensoes.java`, como o nome diz, é a responsável por inserir ou remover pacotes e instruções da ISA da máquina virtual. Com ela é possível também ver quais instruções e tipos existem dentro de um determinado `.jar` de uma extensão.

Existem no pacote ainda outras classes do tipo "observadoras", que são associadas à estruturas de dados da MáquinaVirtual, reagindo à qualquer alteração. Por exemplo, a classe `PilhaExecucaoInterfaceObserver.java` é associada ao `ArrayList` da MáquinaVirtual que implementa o conceito de pilha de execução. Quando alguma instrução insere, altera ou remove algum item da pilha, esta classe é avisada dessas mudanças para que elas sejam refletidas na interface gráfica. É o conceito de "observador" e "observado". As outras classes (`LiteraisInterfaceObserver.java`, `RegistradoresBaseInterfaceObserver.java` e `TabelaInstrucoesInterfaceObserver.java`) são usadas, respectivamente para observar a tabela de literais, pilha de registradores base e a tabela de instruções do programa carregado.

### 4.3. Pacote padrão de instruções e tipos

O pacote padrão extensões inclui 72 instruções e 6 tipos de dados. Com essas instruções e tipos é possível executar todas as características da LSI mostradas no capítulo 2. O código fonte do pacote está no apêndice D.

#### 4.3.1. Tipos de dados

**Booleano** (arquivo `ItemPilhaBooleano.java`)

Este tipo dado tem dois valores possíveis: verdadeiro ou falso.

**Cadeia de caracteres** (arquivo `ItemPilhaCadeia.java`)

O tipo de dado "cadeia de caracteres" possui como valor um número inteiro, que corresponde ao índice na tabela de literais da máquina virtual em que a cadeia de caracteres (string) se encontra. Assim, todas as instruções que forem lidar com strings devem acessar a tabela de literais da MV para obter a cadeia de caracteres desejada.

### **Caracter** (arquivo ItemPilhaCaracter.java)

Corresponde a um "char" da linguagem Java.

### **Endereço** (arquivo ItemPilhaEndereco.java)

O tipo de dado Endereço representa uma referência a um espaço na pilha de execução da máquina virtual. Essa referência é construída a partir de dois valores: nível e deslocamento. O nível corresponde ao nível léxico de execução (valor base), e o deslocamento é somado ao valor base. Importante notar que o deslocamento pode ser zero ou mesmo negativo. Deslocamentos negativos são muito usados quando se quer acessar os parâmetros atuais durante a chamada de um método, ou na hora de armazenar o valor de retorno de um método.

### **Inteiro** (arquivo ItemPilhaInteiro.java)

O tipo inteiro armazena um inteiro de 32 bits com sinal na forma de complemento de 2 (o *int* do Java).

### **Real** (arquivo ItemPilhaReal.java)

Representa um valor com dupla precisão, de 64 bits, seguindo o padrão IEEE-754. Corresponde na prática ao tipo *double* do Java.

## **4.3.2. Instruções para alocação de espaço na pilha**

As instruções abaixo servem para alocar espaços na pilha de execução. Elas recebem apenas o segundo parâmetro de 4 bytes (java int), determinando o número de espaços a serem alocados. Cada instrução aloca espaço para um tipo de dado específico.

<b>Mnemônico</b>	<b>Opcode</b>	<b>Descrição</b>
AMEM_B	0x0013	Aloca espaços para booleanos
AMEM_C	0x0012	Aloca espaços para caracteres
AMEM_I	0x0010	Aloca espaços para inteiros
AMEM_R	0x0011	Aloca espaços para reais
AMEMCAD	0x0015	Aloca espaços para cadeias *
DMEM	0x0017	Remove espaços da pilha

\* Apenas aloca espaço na pilha, e não na tabela de literais.

A instrução DMEM recebe apenas o segundo parâmetro, indicando quantos espaços devem ser apagados do topo da pilha.

### 4.3.3. Instruções para armazenamento

As instruções de armazenamento servem para mover o item do topo da pilha para outra posição. Existem três instruções que fazem isso: ARMZ, ARMZIND e AVET.

Mnemônico	Opcode	Descrição
ARMZ	0x0022	Armazenamento direto
ARMZIND	0x0032	Armazenamento indireto
AVET	0x0410	Armazenamento direto em array

A instrução ARMZ moverá o item que estiver no topo da pilha para posição calculada pela expressão:  $PRA[nível] + deslocamento$ . O nível é obtido do primeiro parâmetro da instrução, e o deslocamento estará no segundo parâmetro.

A instrução ARMZIND é utilizada quando a variável utilizada é uma referência para outra. Isto pode acontecer nos métodos em que um parâmetro é definido com passagem por referência. O funcionamento dela é igual à ARMZ, exceto que, se a posição de destino calculada pela expressão  $PRA[nível]+deslocamento$  conter um item do tipo Endereço, este endereço será seguido iterativamente até o momento que o item apontado não for do tipo Endereço. Cuidado: a instrução pode travar caso exista um ciclo fechado de referências nesses endereços.

A instrução AVET recebe como parâmetros o nível léxico e o deslocamento, nessa ordem, que apontam onde o Array se encontra na pilha. O elemento apontado deverá ser do tipo inteiro, que indica o tamanho do Array, conforme a seção 3.5.4. O índice do array em que o elemento será armazenado deverá estar no subtopo da pilha (deve ser do tipo Inteiro).

Nas três instruções, o tipo de dado do item de destino deve ser igual ao tipo do dado no topo da pilha. Caso contrário, haverá uma exceção `TipoIncorretoException`.

#### 4.3.4. Instruções para carga de valores na pilha

As instruções abaixo servem para copiar ou adicionar valores ao topo da pilha de execução.

Mnemônico	Opcode	Descrição
CRVL	0x0020	Copia um item da pilha para o topo
CRVLIND	0x0031	Copia um item indireto da pilha para ao topo
CREN	0x0030	Carrega na pilha um endereço
CRCT_B	0x0028	Carrega um booleano constante
CRCT_C	0x0027	Carrega um caracter constante
CRCT_I	0x0025	Carrega um inteiro constante
CRCT_R	0x0026	Carrega um real constante
CRCAD	0x0605	Carrega uma cadeia no topo da pilha
CVET	0x0400	Carrega um valor de um array

A instrução CRVL recebe como parâmetros o nível e o deslocamento de um item na pilha. Este item será copiado para o topo da pilha.

A instrução CRVLIND funciona da mesma forma, exceto que se o item apontado por `PRA[nível]+deslocamento` for do tipo Endereço, esta instrução irá seguir o endereço até encontrar um item que não seja endereço. Da mesma forma que na instrução ARMZIND, a instrução pode travar caso haja um ciclo fechado de apontamentos nesses endereços.

A instrução CREN recebe dois parâmetros também, nível e deslocamento. Um item do tipo endereço com esse nível e deslocamento será criado e colocado no topo da pilha.

As instruções CRCT\_B, CRCT\_C e CRCT\_I recebem apenas o segundo parâmetro. Em CRCT\_B, será colocado no topo da pilha um item do tipo booleano. Esse valor será falso se o segundo parâmetro da instrução for igual a zero, e verdadeiro caso seja qualquer outro valor. Em CRCT\_C, será adicionado ao topo da pilha um item do tipo Caracter, com valor



correspondente ao valor do segundo parâmetro da instrução. E CRCT\_I criará no topo da pilha um item do tipo Inteiro com valor igual ao segundo parâmetro.

A instrução CRCT\_R recebe o parâmetro único de 8 bytes, correspondente a um Real (double do Java). Será criado um item do tipo Real com esse valor e copiado no topo da pilha.

A instrução CRCAD recebe o segundo parâmetro, que será usado como índice para buscar na tabela de literais uma cadeia de caracteres (string). Essa cadeia de caracteres será então colocada no topo da pilha.

A instrução CVET recebe dois parâmetros indicando o nível e deslocamento de um Array. No topo da pilha deverá estar um item do tipo Inteiro que representará o índice do Array. O item da pilha apontado pela expressão PRA[nível] + deslocamento + 1 + índice será copiado para o topo da pilha. O número "1" é somado na expressão porque o item na pilha para qual o Array aponta contém o tamanho do Array, e não o primeiro elemento. O item da pilha que representava o índice é apagado.

#### 4.3.5. Instruções para leitura e impressão

As instruções de leitura permitem obter dados digitados pelo usuário. São quatro deste tipo, conforme a tabela abaixo.

Mnemônico	Opcode	Descrição
LEIA_C	0x0502	Lê um caracter
LEIA_I	0x0500	Lê um número inteiro <sup>9</sup>
LEIA_R	0x0501	Lê um número real
LEIACAD	0x0503	Lê uma cadeia de caracteres ( <i>string</i> )

A instrução LEIA\_C irá solicitar ao controlador da MV que obtenha do usuário um texto digitado (*string*). O primeiro caracter do texto retornado é que será considerado e colocado no topo da pilha.

<sup>9</sup> Os inteiros e reais lidos não necessariamente serão no mesmo formato definido na seção 2.1.1. Os formatos de números definidos naquele item são para uso dentro do código de um programa, e não para uso externo (leitura e gravação de dados, por exemplo).

Da mesma forma, as instruções LEIA\_I e LEIA\_R irão tentar converter o texto retornado em inteiro e real, respectivamente. Caso o texto digitado não possa ser convertido, a exceção `NumberFormatException` será gerada.

A instrução LEIACAD pegará o texto digitado e inseri-lo na tabela de literais da máquina virtual. Um item do tipo `Cadeia` será criado contendo o índice no qual o texto foi guardado na tabela de literais, e colocado no topo da pilha.

Quanto à impressão (ou saída de dados para o controlador/interface), são duas instruções:

Mnemônico	Opcode	Descrição
IMPR	0x0550	Imprime o que estiver no topo da pilha
IMPRLIT	0x0554	Imprime um literal

A instrução IMPR pegará o item que estiver no topo da pilha e enviará para a saída do controlador/interface. Se o item for do tipo `Cadeia`, a instrução automaticamente busca a cadeia de caracteres (string) na tabela de literais e a retorna.

A instrução IMPRLIT recebe como segundo parâmetro um número inteiro (java int) que representa o índice da cadeia de caracteres na tabela de literais que deverá ser impressa. Esta instrução existe para simplificar o processo de geração de código. Ela pode ser substituída pela sequência de instruções CRCAD e IMPR. CRCAD colocaria uma cadeia no topo da pilha, e IMPR imprimiria.

#### 4.3.6. Instruções para operações lógicas

Estas instruções trabalham com o tipo de dado booleano.

Mnemônico	Opcode	Descrição
CONJ	0x0050	Conjunção lógica
DISJ	0x0051	Disjunção lógica
NEGA	0x0052	Negação

As instruções não recebem nenhum parâmetro. CONJ e DISJ trabalharão com os dois valores booleanos A e B do topo da pilha de execução. No caso de CONJ, o booleano resultante da operação de conjunção lógica com o A e B ("A && B"), é colocado no topo da pilha. Em DISJ, a operação é de disjunção ("A || B"). Os dois booleanos utilizados na operação são removidos da pilha. Se um dos dois valores iniciais usados na operação não forem do tipo booleano, a exceção `TipoIncorretoException` será gerada.

A instrução NEGA pega o booleano do topo da pilha e inverte seu valor. Se for falso passa a ser verdadeiro, e se for verdadeiro, passa a ser falso. Se o tipo do item no topo da pilha não for booleano, também é gerada a exceção `TipoIncorretoException`.

#### 4.3.7. Instruções para operações matemáticas

O pacote padrão de instruções contém instruções para as quatro operações básicas: somar, subtrair, multiplicar e dividir, além do operador menos unário. No quadro abaixo estão detalhes dessas instruções.

Mnemônico	Opcode	Descrição
SOMA_I	0x0100	Soma números inteiros
SOMA_R	0x0101	Soma números reais
SUB_I	0x0102	Subtrai números inteiros
SUB_R	0x0103	Subtrai números reais
MULT_I	0x0104	Multiplica inteiros
MULT_R	0x0105	Multiplica reais
DIV	0x0107	Divisão de números reais
MUN_I	0x0108	Menos unário para inteiros
MUN_R	0x0109	Menos unário para reais

As instruções SOMA\_I e SOMA\_R irão utilizar os dois itens no topo da pilha - a primeira instrução trabalha com tipo inteiro, e a segunda, com o tipo real. O resultado da operação é colocado no topo da pilha, e os valores utilizados são removidos. A mesma ideia é aplicada para SUB\_I e SUB\_R, MULT\_I e MULT\_R. Nas instruções de subtração, o item do topo da pilha será subtraído do item do sub-topo.

A instrução DIV dividirá o item do sub-topo pelo item presente no topo da pilha. O tipo usado é apenas Real, pois uma divisão de inteiros nem sempre resultará em inteiro.

As instruções MUN\_I e MUN\_R multiplicarão o número inteiro ou real que estiver no topo da pilha por -1.

Embora a linguagem de exemplo permita operações matemáticas com tipos diferentes, as instruções desenvolvidas operam apenas com um tipo. Para resolver isto, deve-se utilizar instruções de conversão de tipo, descritas na seção 4.3.9.

A exceção TipoIncorretoException pode ser gerada quando os tipos dos dados utilizados na pilha não forem do tipo esperado pelas instruções.

#### 4.3.8. Instruções para comparações

Para os operadores igual ("="), diferente ("<>"), maior (">"), menor ("<"), menor ou igual ("<=") e maior igual (">=") foram criadas algumas instruções, como mostrado abaixo:

Mnemônico	Opcode	Descrição
CMIG	0x0200	Compara se valores são iguais (inteiros)
CMDF	0x0210	Compara se valores são diferentes (inteiros)
CMMA	0x0220	Compara se valor é maior (inteiros)
CMME	0x0230	Compara se valor é menor (inteiros)
CMEI	0x0240	Compara se valor é menor ou igual (inteiros)
CMAI	0x0250	Compara se valor é maior ou igual (inteiros)

Todas as instruções de comparação irão utilizar os dois valores no topo da pilha. Caso esses valores não sejam do mesmo tipo, a exceção TipoIncorretoException é gerada. Após a execução, esses dois valores são retirados da pilha, e em seguida, um item do tipo booleano contendo o resultado é colocado no topo da pilha.

Quando os valores no topo da pilha forem do tipo `Character`, a comparação é feita levando em conta o valor numérico do item. No Java, o tipo `"char"` é um valor de 16 bits que corresponde a um número entre 0 e 65.535.

Para o tipo `Cadeia`, foram desenvolvidas as instruções `CMIGCAD` e `CMDFCAD`, para comparar se duas cadeias de caracteres são iguais ou diferentes, respectivamente. Para que elas funcionem, os dois itens do topo da pilha devem ser do tipo `Cadeia`. Importante observar que a comparação é feita com base no valor guardado na tabela de literais.

Mnemônico	Opcode	Descrição
CMIGCAD	0x0607	Compara se duas cadeias são iguais
CMDFCAD	0x0608	Compara se duas cadeias são diferentes

Utilizar `CMIG` ou `CMDF` com valores do tipo `Cadeia` pode gerar resultados inesperados, pois essas instruções utilizarão o índice da `Cadeia` na tabela de literais, e não o literal propriamente dito. Veja a seção 4.3.1 para mais detalhes sobre o tipo `Cadeia de Caracteres`.

#### 4.3.9. Instruções para conversões de tipos

Em alguns casos pode ser necessário converter dados de um tipo para outro. Por exemplo, quando somamos uma variável do tipo `Inteiro` com outra do tipo `Real`. Não foi definida uma instrução que trabalhe com os dois tipos simultaneamente. Então, antes de se somar (usando a instrução `SOMA_R`), é preciso converter o valor inteiro em real. Essas instruções são usadas também em conversões explícitas de tipos. A tabela abaixo apresenta as instruções de conversão:

Mnemônico	Opcode	Descrição
I2R	0x0300	Inteiro para real
I2C	0x0301	Inteiro para caracter
I2B	0x0302	Inteiro para booleano
I2CAD	0x0303	Inteiro para cadeia de caracteres
R2I	0x0310	Real para inteiro
R2C	0x0311	Real para caracter
R2B	0x0312	Real para booleano
R2CAD	0x0313	Real para cadeia de caracteres
B2I	0x0320	Booleano para inteiro
B2R	0x0321	Booleano para real

B2CAD	0x0322	Booleano para cadeia de caracteres
C2I	0x0330	Caracter para inteiro
C2CAD	0x0331	Caracter para cadeia de caracteres
CAD2I	0x0340	Cadeia de caracteres para inteiro
CAD2C	0x0341	Cadeia de caracteres para caracter
CAD2R	0x0342	Cadeia de caracteres para real

As instruções I2B e R2B irão converter um número para booleano. Se o valor numérico do item no topo da pilha (inteiro ou real, respectivamente) for igual a 0, então o valor booleano será falso, caso contrário, será verdadeiro.

As instruções I2C e R2C farão a conversão para um Caracter. No caso de I2C é simples, o caracter resultante terá o valor correspondente ao inteiro na tabela ASCII. Por exemplo, o inteiro 65 resultará no caracter 'A'. Na instrução R2C, o processo é o mesmo, exceto que a parte decimal do número é eliminada (truncada). Se o objetivo for transformar, por exemplo, o Inteiro 5 no caracter '5', deve-se utilizar a instrução I2CAD seguida de CAD2C. Em C2I, o conceito aplicado é o mesmo de I2C, só que inverso.

A instrução I2R transformará o valor no topo da pilha em real. A instrução oposta, R2I, transformará um real em inteiro. A parte decimal do número real é eliminada (truncada). Assim, o real 3.9 seria convertido no inteiro 3. Caso haja necessidade de arredondamento, deve-se criar uma instrução customizada.

As instruções I2CAD, R2CAD, B2CAD e C2CAD converterão inteiros, reais, booleanos e caracteres em valores do tipo Cadeia.

CAD2I, CAD2R e CAD2C irão converter cadeias de caracteres em inteiros, reais e caracteres, respectivamente. Caso a cadeia de caracteres não possa ser convertida em inteiro ou real, a exceção `FormatoIncorretoException` é gerada. CAD2C converterá apenas o primeiro caracter da cadeia de caracteres. Por exemplo, a cadeia "exemplo" resultaria em 'e'.

### 4.3.10. Instruções para desvios de fluxo

As duas instruções para desvios de fluxo são apresentadas abaixo:

Mnemônico	Opcod	Descrição
DSVS	0x0060	Desvia sempre
DSVF	0x0061	Desvio condicional

A instrução DSVS recebe como segundo parâmetro um inteiro indicando para qual instrução o fluxo será sempre desviado. Esse valor deve estar entre 0 e o número de instruções do programa menos 1.

A instrução DSVF irá verificar se o booleano no topo da pilha é falso. Se for, o fluxo é desviado para instrução apontada no segundo parâmetro, caso contrário, o fluxo continua normalmente para a próxima instrução. Se o valor no topo da pilha não for booleano, a exceção `TipoIncorretoException` é gerada. No final, o valor booleano utilizado é removido da pilha.

### 4.3.11. Instruções para chamadas e retornos de métodos

São apenas duas instruções usadas na definição de um método: CALL e RETU.

Mnemônico	Opcod	Descrição
CALL	0x0070	Chamada de método
RETU	0x0071	Retorno de método

A instrução CALL recebe dois parâmetros: o primeiro indica o nível léxico estático em que o método será chamado, e o segundo indica a posição da primeira instrução do método.

A posição da próxima instrução depois de CALL é guardada no topo da pilha, assim, quando o método retornar, a execução continuará a partir deste ponto. O valor presente na Pilha de Registros de Ativação correspondente ao nível léxico utilizado também é guardado (isto permite várias chamadas de métodos do mesmo nível). Após esses dados serem guardados na pilha, o fluxo é desviado para a instrução indicada no segundo parâmetro (início do método).

O nível léxico é conhecido em tempo de compilação. Veja o exemplo abaixo:

```
programa exemplo;
metodo met1;
metodo met2;
{
    // corpo de met2
}
{
    // corpo de met1
}
{
    // corpo do programa exemplo
}.
```

O método met1 está declarado no nível léxico 1 - verde (o corpo do programa corresponde ao nível 0 - vermelho), e o método met2 está no nível 2 - azul. Observe que dentro um método no nível X, só poderão ser chamados métodos de níveis mais altos. Por exemplo, o método met2 pode ser chamado no corpo do método met1, mas met1 não pode ser chamado em met2, pois met1 está em um nível mais baixo que met2.

A instrução RETU é chamada no retorno dos métodos (ou quando o método acaba, naqueles em que não for definido um retorno). Recebe dois parâmetros: o nível do qual se está saindo (o mesmo usado na instrução CALL), e o segundo parâmetro deve ser o número de parâmetros formais do método.

Este método irá restaurar os valores da PRA antes da chamada do método em questão, além de retornar o fluxo para a instrução seguinte à instrução CALL. Também é responsável por remover da pilha os parâmetros atuais alocados para o método.

O valor de retorno de um método é armazenado com a instrução ARMZ, em um espaço na pilha previamente alocado. Isto está descrito na seção 3.5.13.



### 4.3.12. Instruções para manipulação de cadeias de caracteres

Algumas instruções básicas para manipulação de cadeias de caracteres foram incluídas no pacote básico. São elas:

Mnemônico	Opcode	Descrição
TAMCAD	0x0601	Calcula o tamanho da cadeia
CONCAT	0x0602	Concatenação de duas cadeias de caracteres
POSCAD	0x0603	Informa a posição de uma subcadeia
SUBCAD	0x0604	Retorna uma subcadeia
CHARAT	0x0606	Retorna um caracter em uma posição

A instrução TAMCAD calcula o tamanho da cadeia de caracteres presente no topo da pilha. O valor resultante é do tipo Inteiro.

CONCAT trabalha com duas cadeias de caracteres presentes no topo da pilha. Elas serão concatenadas (cadeia do subtopo + cadeia do topo), e o resultado colocado na pilha.

POSCAD também utiliza duas cadeias de caracteres do topo da pilha. O resultado é um valor do tipo Inteiro indicando a primeira ocorrência da cadeia localizada no topo na cadeia do subtopo. Caso não exista ocorrência, o valor -1 será retornado. As duas cadeias utilizadas são removidas do topo da pilha.

SUBCAD utiliza três valores presentes no topo da pilha:

- Topo - 2 (sub-sub-topo): cadeia de caracteres
- Topo - 1 (sub-topo): início da sub-cadeia
- Topo: fim da sub-cadeia

A instrução colocará no topo da pilha a subcadeia (*substring*) presente entre as posições topo-1 e topo da cadeia localizada em topo-2.

A instrução CHARAT utiliza uma cadeia de caracteres (subtopo) e um índice inteiro (topo). O resultado é um item do tipo caracter com valor igual ao caracter da posição indicada pelo índice (começando de 0). Caso o índice usado seja maior que o tamanho da cadeia, a exceção `IndiceInvalidoException` é gerada.

### 4.3.13. Outras instruções

Abaixo algumas outras instruções:

Mnemônico	Opcode	Descrição
INICIO	0x494E	Indica o início do programa
FIM	0x4649	Indica o fim do programa
NADA	0x4E44	Não faz nada

A instrução INICIO deve ser a primeira instrução de todo programa, bem como a instrução FIM deverá ser a última (a MV só para de executar ao chegar nesta instrução). NADA não faz nada e pula para a próxima instrução do programa. É geralmente usada "demarcar" o início e fim de blocos "se", "enquanto" e métodos, por exemplo, quando alunos compilam programas manualmente.

### 4.4. Criando pacotes de instruções e tipos (extensões)

Antes de criar novas instruções e tipos de dados, é necessário saber o que será feito com eles. Isso é feito por meio da alteração da linguagem desde o início, nas definições regulares, tokens e da gramática. Nos itens abaixo mostraremos as etapas para a criação de um tipo de dado novo, utilizando o GALS, e também todas as alterações necessárias da linguagem para que a mesma suporte arrays bidimensionais.

#### 4.4.1. Criação de um novo tipo de dado

O exemplo que utilizaremos envolve a criação do tipo de dado MATRICULA, que poderá ser usado em um sistema de gerenciamento de cadastros de funcionários ou alunos, por exemplo.

Neste exemplo, a matrícula tem o formato *A00000BB* (uma letra maiúscula, cinco dígitos e duas letras maiúsculas). Nas definições regulares da especificação léxica da LSI (seção 2.1.1), não foi definido o que seria uma letra maiúscula (apenas "letra", o que inclui as minúsculas). Então, precisamos adicionar a definição regular:

LETRAM : [A-Z]

O próximo passo é adicionar o token de acordo com o formato especificado no parágrafo acima:

```
literal_matricula : {LETRAM}{DIGITO}{DIGITO}{DIGITO}{DIGITO}{DIGITO}{LETRAM}{LETRAM}
```

Observação: o trecho acima deve ser colocado antes da definição do token "id", caso contrário, as matrículas seriam reconhecidos como identificadores.

Também é necessário reservar a palavra "matricula", para poder ser usada em declaração de variáveis (colocar depois da definição do token "id"):

```
matricula = id : "matricula"
```

Agora, é necessário fazer algumas modificações na gramática da linguagem.

Editar a definição do não-terminal <tipo>, para:

```
<tipo> ::= inteiro | real | booleano | caracter | cadeia |  
matricula;
```

Editar a definição do não-terminal <constante\_explicita>, para:

```
<constante_explicita> ::= num_int | num_real | falso |  
verdadeiro | literal_cadeia | literal_matricula;
```

Como esta é uma pequena alteração da linguagem, não restou necessário a alteração de mais estruturas da gramática, nem a criação de mais não-terminais.

Agora, o próximo passo é fazer a validação semântica da utilização deste tipo. Para isso, devemos adicionar regras semânticas que devem ser garantidas pelo compilador (isto não é abordado neste trabalho):

1. No comando de atribuição, variáveis do tipo CADEIA devem aceitar valores do tipo MATRICULA

2. Em operações de comparação entre matrículas, a comparação deve ocorrer primeiro pela letra inicial, depois pelas duas letras finais, e finalmente, pelos cinco dígitos do meio. Por exemplo, A00002AA é menor que A00002AB.
3. Valores do tipo MATRICULA não participam de operações matemáticas (somar, dividir, multiplicar, subtrair, etc.).

As alterações na linguagem acabaram. Para que a MV possa trabalhar com matrículas, precisaremos também criar um novo tipo de dado da MV (não devemos confundir tipo de dado da linguagem com tipo de dado da máquina virtual. Em nosso trabalho, o único tipo de dado da MV que não possui um correspondente na linguagem é o tipo Endereço). Chegou a parte de programação.

Um pacote de extensão da MV-LSI nada mais é do que um pacote Java (.jar) com algumas classes dentro dele, conforme as regras abaixo:

1. O nome do pacote deve começar com UFSC.MaquinaVirtual.Extensoes. Neste exemplo, nomearemos o pacote de UFSC.MaquinaVirtual.Extensoes.TipoMatricula.
2. Instruções serão todas as classes desse pacote que se estenderem à classe Instrucao do pacote do núcleo da MV (UFSC.MaquinaVirtual).
3. Tipos serão todas as classes do pacote que se estenderem à classe ItemPilha do pacote do núcleo da MV.
4. Tipos literais (que utilizam a tabela de literais, como cadeia e matricula) devem incluir ainda uma classe que seja extensão da classe Literal do pacote do núcleo da MV.

No apêndice E estão os códigos fontes da classe ItemPilhaMatricula, que deverá estar no pacote UFSC.MaquinaVirtual.Extensoes.TipoMatricula. O núcleo da MV (MaquinaVirtual.jar) e o pacote padrão (PacotePadrao.jar) devem ser adicionados à lista de bibliotecas, para que suas classes e métodos possam ser usados no pacote de extensão.

O próximo passo é identificar em quais trechos da gramática vai ser preciso alterar a geração de código. Neste exemplo, identificamos as seguintes necessidades:

1. Alocação de variáveis: criação de uma instrução para alocação de espaços do tipo MATRICULA na pilha de execução
2. Leitura: criação de instrução para leitura de matrículas
3. Impressão: não é necessário, pois a instrução IMPR do pacote padrão é capaz de imprimir literais.
4. Carga de constante: instrução para carga de matricula constante na pilha de execução
5. Conversão: instrução para conversão de valores do tipo MATRICULA para o tipo CADEIA, para quando uma variável de matricula possa ser usada como cadeia de caracteres (em atribuições, concatenações, entre outras operações relacionadas à cadeias de caracteres)
6. Comparação: instruções que digam se uma matricula é diferente, maior, maior igual, igual, menor igual ou menor que outra.

Na tabela abaixo estão listadas as novas instruções para manipulação de matrículas. O mnemônico e opcode podem ser escolhidos livremente, desde que não conflitem com instruções de outros pacotes.

Mnemônico	Opcode	Descrição
AMEMMATR	0x0701	Aloca espaço para matrículas
LEIAMATR	0x0702	Lê uma matrícula
CRMATR	0x0703	Carrega uma matrícula no topo da pilha
MATR2CAD	0x0704	Converte matrícula em cadeia de caracteres
CMIGMATR	0x0705	Compara se duas matrículas são iguais
CMDFMATR	0x0706	Compara se duas matrículas são diferentes
CMMAMATR	0x0707	Comparação ">" entre duas matrículas
CMAIMATR	0x0708	Comparação ">=" entre duas matrículas
CMEMATR	0x0709	Comparação "<" entre duas matrículas
CMEIMATR	0x0710	Comparação "<=" entre duas matrículas

Nos arquivos executáveis (LSIB), as constantes literais do tipo matrícula deverão ocupar 8 bytes, sendo um byte para cada caracter da matrícula.

Consulte a seção 3.3 para detalhes sobre o formato da tabela de literais num arquivo executável.

O código fonte das instruções listadas acima e das classes do tipo MATRICULA está no apêndice E.

Em relação à geração de código para esse novo tipo, será bastante semelhante ao código gerado para o tipo cadeia. No exemplo abaixo é possível encontrar o uso de diversas instruções do pacote:

Código LSI:

```
programa exemplo_matriculas;
  matricula matr1;
  matricula matr2 = L00023MS;
  cadeia msg;
  {
    escreva("Digite uma matrícula: ");
    leia(matr1);
    msg := "Matricula digitada (matr1): "+matr1+"\n";
    escreva(msg);
    escreva("matr2 = ");
    escreva(matr2);
    escreva("\n");

    se(matr1 = matr2) entao {
      escreva("matr1 = matr2 (" +matr2+")\n");
    }

    se(matr1 > matr2) entao {
      escreva("matr1 > matr2 (" +matr2+")\n");
    }

    se(matr1 < matr2) entao {
      escreva("matr1 < matr2 (" +matr2+")\n");
    }
  }.

```

*Bytecode* gerado:

Literais:

```
0. L00023MS (do tipo matricula)
1. Digite uma matrícula:
2. Matricula digitada (matr1):
3. \n

```

```
4. matr2 =
5. matr1 = matr2 (
6. )\n
7. matr1 > matr2 (
8. matr1 < matr2 (
```

Instruções:

```
0. inicio 0 0
1. amemmatr 0 1 // aloca espaço para matricula
2. amemcad 0 1 // aloca espaço para uma cadeia
3. imprlit 0 1 // imprime " Digite uma matrícula: "
4. leiamatr 0 0 // lê uma matricula
5. armz 0 0 // armazena em matr1
6. crcad 0 2 // carrega "Matricula digitada (matr1):" na pilha
7. crvl 0 0 // carrega matr1 na pilha
8. matr2cad 0 0 // transforma de matricula para cadeia
9. concat 0 0 // concatena as duas cadeias no topo da pilha
10. crcad 0 3 // carrega "\n" no topo da pilha
11. concat 0 0 // concatena as duas cadeias no topo da pilha
12. armz 0 1 // armazena a cadeia em msg
13. crvl 0 1 // carrega msg no topo da pilha
14. impr 0 0 // imprime msg
15. imprlit 0 4 // imprime "matr2 = "
16. imprlit 0 0 // imprime "L00023MS"
17. imprlit 0 3 // imprime "\n"
18. crvl 0 0 // carrega matr1 na pilha
19. crmatr 0 0 // carrega "L00023MS" na pilha
20. cmigmatr 0 0 //compara as duas matriculas no topo da pilha
21. dsvf 0 29 // desvia se não forem iguais
22. crcad 0 5 // carrega "matr1 = matr2 (" na pilha
23. crmatr 0 0 // carrega "L00023MS" na pilha
24. matr2cad 0 0// converte matricula em cadeia
25. concat 0 0 // concatena
26. crcad 0 6 // carrega ")\n" na pilha
27. concat 0 0 // concatena
28. impr 0 0 // imprime
29. crvl 0 0 // carrega matr1 na pilha
30. crmatr 0 0 // carrega "L00023MS" na pilha
31. cmmamatr 0 0// compara se matr1 é maior
32. dsvf 0 40 // desvia se falso
33. crcad 0 7 // carrega "matr1 > matr2 (" na pilha
34. crmatr 0 0 // carrega "L00023MS" na pilha
35. matr2cad 0 0// converte para cadeia
36. concat 0 0 // concatena
37. crcad 0 6 // carrega ")\n" na pilha
38. concat 0 0 // concatena
39. impr 0 0 // imprime
40. crvl 0 0 // carrega matr1 na pilha
```

```

41. crmatr 0 0 // carrega "L00023MS" na pilha
42. cmmematr 0 0// compara se matr1 é menor
43. dsvf 0 51 // desvia se falso
44. crcad 0 8 // carrega "matr1 < matr2 (" na pilha
45. crmatr 0 0 // carrega "L00023MS" na pilha
46. matr2cad 0 0// converte para cadeia
47. concat 0 0 // concatena
48. crcad 0 6 // carrega ")\n" na pilha
49. concat 0 0 // concatena
50. impr 0 0 // imprime
51. fim 0 0

```

#### 4.4.2. Estendendo a linguagem para adicionar suporte à arrays bidimensionais

Para adicionar suporte a arrays bidimensionais, não é necessária a criação de um novo tipo de dado, tendo em vista que apenas modificações na linguagem e a criação de algumas instruções será suficiente para atingir esse objetivo.

O primeiro passo é modificar a gramática da linguagem. A produção do não-terminal <dimensao> passa a ser definida assim:

```
<dimensao> ::= "[" <constante> "]" <dimensao2> | î ;
```

O não terminal <dimensao2> deve ser criado:

```
<dimensao2> ::= "[" <constante> "]" | î ;
```

A produção do não-terminal <rcomid> deve ser modificada também:

```

<rcomid> ::= ":" <expressao>
          | "[" <expressao> "]" ":" <expressao>
          | "["<expressao>"]""["<expressao>"]" ":" <expressao>
          | "(" <expressao> <rep_lexpr> ")"
          | î ;

```

E o não-terminal <rvar> também é alterado para:

```

<rvar> ::= "(" <expressao> <rep_lexpr> ")" | "[" <expressao>
 "]" | "["<expressao>"]""["<expressao>"]" | î ;

```

O próximo passo é aplicar verificações semânticas no compilador para:



1. Arrays bidimensionais não podem possuir valor inicial
2. Variáveis do tipo array bidimensional só podem ser usadas de forma indexada (com dois índices, como "array[5][9]"), e a expressão entre colchetes deve ser inteira
3. O tamanho de um array bidimensional deve ser definido com duas expressões do tipo inteiro (como em inteiro[10][5] array\_bid).
4. Variáveis do tipo Array bidimensional devem ser referenciadas de forma indexada. Em operações lógicas, matemáticas ou de comparação, deve ser verificado o tipo dos elementos.

A parte mais importante é definir como o array bidimensional será armazenado na pilha de execução. A maneira mais simples é guardar o tamanho as duas dimensões no início do array (como feito no array unidimensional), e os elementos virão depois. Por exemplo:

```
array[3][4] inteiro;
```

Representa um array de  $3 * 4 = 12$  elementos inteiros

Na pilha ele é armazenado como:

```
[3][4][0][0][0][0][0][0][0][0][0][0]
^ base do array
```

Os elementos de cada dimensão estão marcados de cores diferentes. O exemplo abaixo mostra como calcular a posição de um elemento no array acima:

```
inteiro elem = array[1][2];
```

elem deve receber o valor apontado pela expressão abaixo:

```
array + 2 + (indice1 * tamanho_segunda_dimensao) + indice2
```

Trocando para os valores do exemplo:

```
array + 2 + (1 * 4) + 2
```

As instruções necessárias para utilização de arrays bidimensionais são apenas duas:

Mnemônico	Opcode	Descrição
AVET2	0x0801	Armazena em array bidimensional
CVET2	0x0802	Carrega elemento de array bidimensional

Na geração de código, existirão três casos: declaração, armazenamento e acesso a arrays bidimensionais.

### Declaração:

Como o número de dimensões é armazenado no início do array, devem ser chamadas duas instruções CRCT\_I, e logo em seguida, AMEM\_X para alocar os espaços necessários.

Por exemplo:

```
inteiro[4][8] meuArray;
```

Gera o bytecode:

```
CRCT_I 0 4  
CRCT_I 0 8  
AMEM_I 0 32 // 4*8 = 32
```

### Armazenamento em arrays bidimensionais:

A instrução AVET2 utiliza três elementos do topo da pilha: índice da primeira dimensão (topo-2), índice da segunda dimensão (topo-1), elemento a ser armazenado (topo). Os parâmetros da instrução indicam o nível e o deslocamento do array. Por exemplo:

```
programa exemplo_arr2_arm;  
    inteiro[3][4] arr;  
    {  
        arr[1][2] := 99;  
    }.
```

Gera o bytecode:

```
inicio 0 0  
crct_i 0 3 // primeira dimensão  
crct_i 0 4 // segunda dimensão
```

```

amem_i 0 12      // aloca espaço para os elementos
crct_i 0 1       // carrega o índice da primeira dimensão
crct_i 0 2       // carrega o índice da segunda dimensão
crct_i 0 99      // carrega o valor a ser armazenado
avet2 0 0        // armazena no array arr.
fim 0 0

```

Os códigos fontes das instruções AVET2 e CVET2 estão no apêndice F.

### **Acesso a arrays bidimensionais:**

O acesso ao array bidimensional é provido pela instrução CVET2. Ela utiliza dois elementos do topo da pilha, como índice da primeira dimensão (topo-1) e índice da segunda dimensão (topo-2). Os parâmetros da instrução apontam o nível e o deslocamento do array. Exemplo:

```

programa exemplo_arr2_arm;
  inteiro[3][4] arr;
  {
    arr[1][2] := 99;
    escreva(arr[1][2]);
  }.

```

Gera o bytecode:

```

inicio 0 0
crct_i 0 3
crct_i 0 4
amem_i 0 12
crct_i 0 1
crct_i 0 2
crct_i 0 99
avet2 0 0
crct_i 0 1 // índice da primeira dimensão
crct_i 0 2 // índice da segunda dimensão
cvet2 0 0 // carrega o valor (99) presente no array
impr 0 0 // imprime
fim 0 0

```

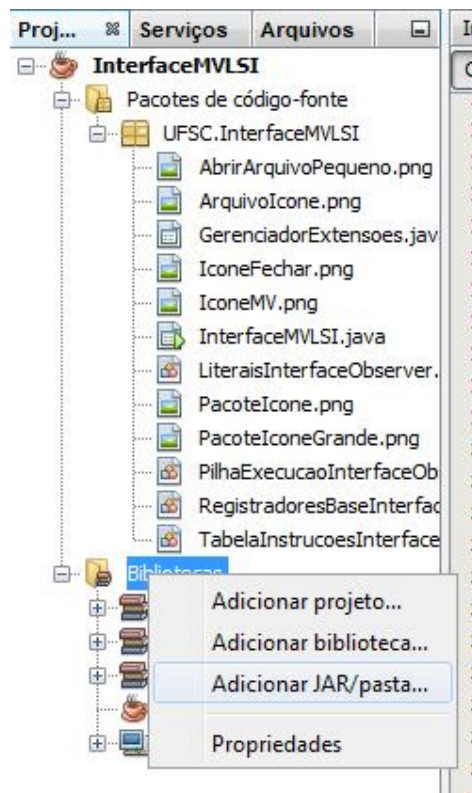
## **4.5. Integrando a MV-LSI em outros projetos**

Para permitir que este projeto pudesse ser aproveitado em outras aplicações, dividimos o trabalho em três partes, como descrito no capítulo 4: interface, núcleo e pacotes de extensões.

Integrar a MV-LSI em outro projeto significa reutilizar o pacote referente ao núcleo. Dessa forma, é possível, por exemplo, construir um compilador para uma linguagem qualquer e já poder executar programas (como em uma IDE), dentro da mesma aplicação.

Nesta seção serão abordadas todas as etapas necessárias para fazer isto.

O primeiro passo é adicionar o pacote `MaquinaVirtual.jar` na lista de bibliotecas do seu projeto. Na IDE NetBeans, isto é feito clicando com o botão direito do mouse no item "Bibliotecas" na árvore de arquivos do seu projeto, conforme a figura abaixo:



**Figura 11: adicionando o núcleo da Máquina Virtual como biblioteca em outros projetos**

A partir de agora, é possível utilizar todas as classes do pacote `MaquinaVirtual.jar`, que iniciam com o nome `UFSC.MaquinaVirtual`.

A classe mais importante desse pacote é a `MaquinaVirtual`. Ela é uma extensão da classe `Thread`, do próprio Java. Assim, para criar um objeto `MaquinaVirtual`, utiliza-se o código:

```
MaquinaVirtual mv = new MaquinaVirtual(objetocontrolador);
```

A variável `objetocontrolador` é uma instância de algum objeto que implementa a interface `MaquinaVirtualInterface`. Este objeto que será chamado em eventos da MV, como quando o programa do usuário requisitar entrada de dados pelo teclado; quando a MV iniciar, reiniciar, parar, travar, etc.

Em seguida, é necessário iniciar a `Thread`:

```
mv.start();
```

Isto apenas faz a `Thread` da MV iniciar, e não a execução do programa do usuário (até porque nenhum programa foi carregado ainda).

Agora, é preciso carregar as instruções que irão compor a ISA da nossa máquina virtual. Para isso é utilizado o método `addInstructionToISA`:

```
mv.addInstructionToISA(instrucao);
```

Este método recebe como parâmetro uma instância de alguma instrução de um pacote de extensões - lembrando que todas as instruções devem ser extensões da classe `Instrucao`, do pacote `MaquinaVirtual`. O método retornará `true` se a instrução for inserida com sucesso, e `false` caso contrário (se já existir outra instrução com o mesmo opcode na ISA). Há ainda os métodos `removeInstructionFromISA`, para remover instruções já inseridas e `clearInstructionSet`, que remove todas as instruções.

O mesmo precisa ser feito quanto aos tipos literais (que necessitam da tabela de literais). O método usado é o `addTipoLiteral`. Para remover, usa-se `removeTipoLiteral`, ou ainda `clearTiposLiterais`.

Na interface gráfica desenvolvida neste trabalho, os pacotes de extensões são carregados dinamicamente em tempo de execução, pelo método `loadClassesFromJar` da classe `InterfaceMVLSI`. Mas isto não é necessário fazer caso não haja necessidade dessa característica - o(s) pacote(s) de extensões poderiam ser adicionados como bibliotecas do seu projeto durante o desenvolvimento.

O próximo passo é carregar um programa na máquina virtual. Para isso utiliza-se o método `carregarPrograma(arquivo)`, em que *arquivo* é uma instância da classe `java File`. Esse método pode gerar algumas exceções, como:

- `FileNotFoundException`: o arquivo usado como parâmetro não existe;
- `FormatoIncorretoException`: o arquivo não está no formato especificado na seção 3.3;
- `InstrucaoNaoReconhecidaException`: o arquivo utiliza uma instrução não constante no ISA da máquina virtual;
- `TipoLiteralNaoReconhecidoException`: o arquivo utiliza um tipo literal inexistente nos pacotes de extensão.
- `IOException`: erros diversos na abertura e leitura do arquivo;
- `IllegalStateException`: há uma instrução em execução, e por isso outro programa não pode ser aberto.

Se tudo ocorrer bem na abertura do arquivo, nenhuma exceção será gerada, e os métodos de controle da MV poderão ser finalmente usados. Esses métodos são:

- `iniciar()`: inicia a execução das instruções, de forma sequencial;
- `parar()`: para a execução do programa atual;

- `reiniciar()`: retoma a execução (que pode ter sido parada por `parar()`);
- `resetar()`: deixa a MV pronta para uma nova execução do programa atual;
- `executarPasso()`: executará a próxima instrução do programa imediatamente. Se quando esse método for chamado outra instrução ainda estiver executando, nada ocorre.
- `finalizar()`: indica que o programa atual chegou ao fim da execução. Deve ser chamado apenas pela instrução FIM ou outra equivalente!

Após a execução, pode-se utilizar o método `fecharArquivo()` para descarregar o programa atual da MV.

Para acompanhar o estado da Máquina Virtual, pode-se utilizar os métodos abaixo, que retornam valores booleanos:

- `executando()`: informa se a MV está iniciada (se o método `iniciar()` ou `reiniciar()` foi chamado)
- `parou()`: informa se a MV está parada (esse é o estado inicial e quando o método `parar()` é chamado)
- `travou()`: quando alguma instrução gera uma exceção, este método passa a retornar `true` e a MV para.
- `iniciou()`: retorna `true` após o método `iniciar()` ser chamado, ou se a execução começou por meio da chamada de `executarPasso()`;
- `finalizou()`: informa se a execução do programa chegou ao fim, sem travar;

Para mais detalhes do funcionamento da classe `MaquinaVirtual`, veja o apêndice B.

#### **4.6. Utilizando a MV-LSI para executar programas**

Neste item apresentamos a interface gráfica da MV-LSI e como utilizá-la.

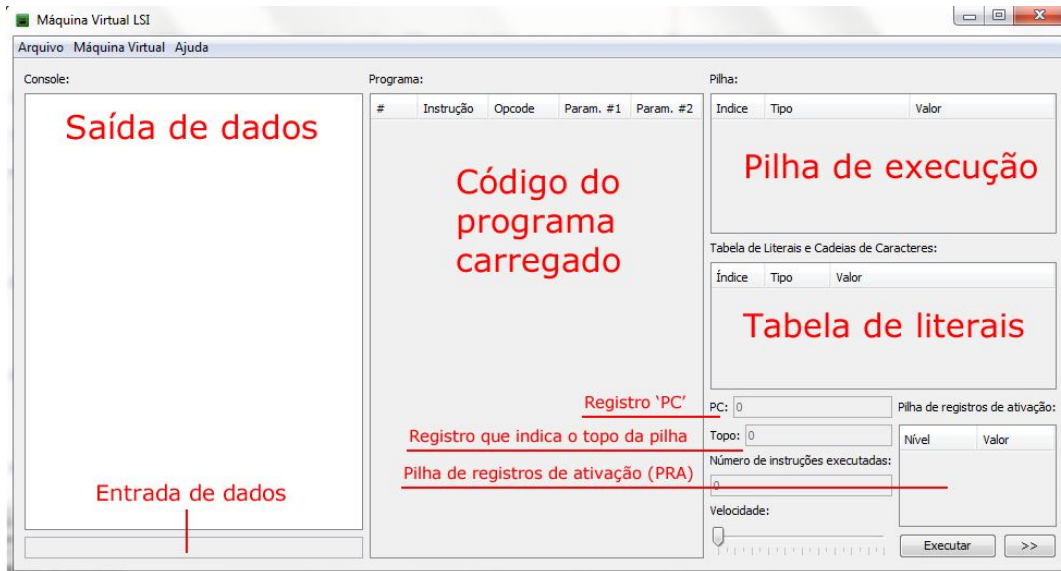


Figura 12: tela inicial da MV-LSI

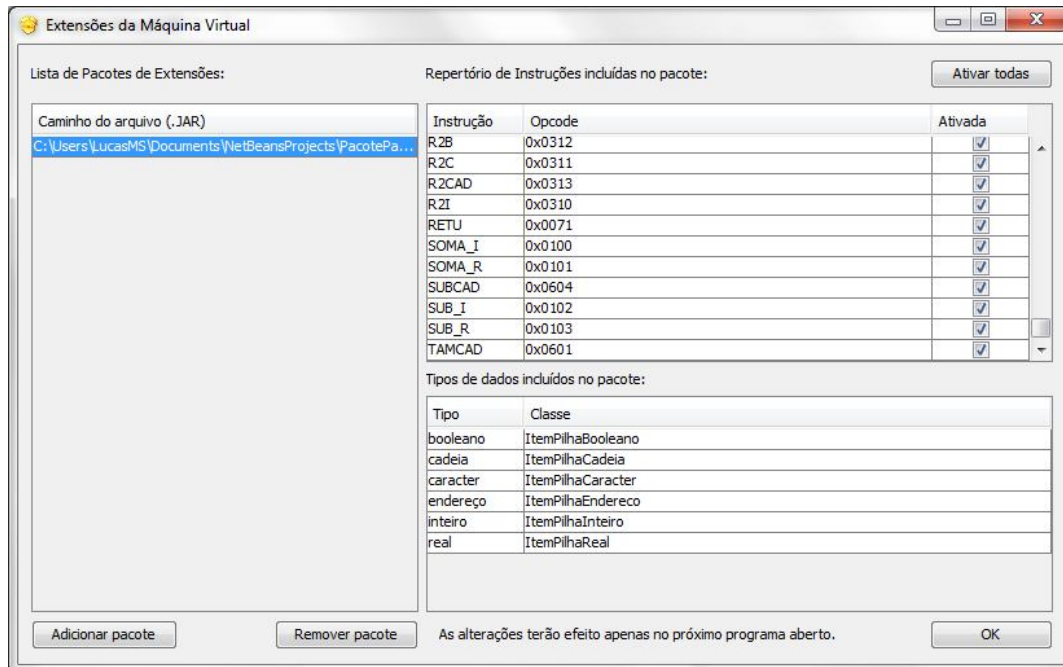
A Figura 12 mostra as partes principais da interface da MV-LSI. Nela é possível ver o programa carregado, a saída de dados gerada, inserção e remoção de valores da pilha de execução, registros e qual a instrução está sendo executada no momento.

Ao carregar um programa pelo menu "Arquivo -> Abrir ...", as instruções do mesmo serão mostradas na área central, com seus respectivos mnemônicos, opcodes e parâmetros. A tabela de literais também será preenchida (caso o programa utilize literais). A pilha de execução inicialmente estará vazia (o componente utilizado para representar a pilha, um `JTable`, não permite que as linhas sejam adicionadas de baixo para cima (como em uma "pilha" de verdade), por isso o primeiro item da pilha estará acima do segundo item, e assim sucessivamente. O "topo" da pilha, na verdade, será a última linha dessa tabela.

Para executar o programa há duas maneiras: clicando no botão "Executar", ou no botão ">>", que executa apenas uma instrução. O botão "Executar" irá executar as instruções do programa automaticamente, com um intervalo de tempo determinado pelo *slider* "Velocidade". O botão executar transforma-se em "Pausar" ou "Reiniciar" quando apropriado.



Na Figura 13, abaixo, a tela apresentada ao clicar no menu "Máquina Virtual -> Pacotes de Extensões".



**Figura 13: interface para gerenciamento dos pacotes de instruções.**

Aqui é possível adicionar ou remover pacotes de extensões. Em cada pacote é possível determinar quais instruções estão ativadas (irão compor o ISA da máquina virtual). O programa não permitirá que duas instruções (mesmo que de pacotes diferentes) com o mesmo Opcode sejam ativadas simultaneamente, caso contrário a MV não saberia qual instrução utilizar.

## 5. Conclusão

Este trabalho gerou uma importante ferramenta de aprendizado na área de compiladores, especialmente por estar completamente adaptada à metodologia adotada nesta disciplina na UFSC. Por meio da máquina virtual será possível dar um passo adiante na disciplina, permitindo a execução de forma simples e interativa de códigos gerados pelos alunos. Mais ainda, permite a programação de extensões, em que novas estruturas e paradigmas poderão ser experimentados no mesmo ambiente.

Como sugestão para trabalhos futuros, abaixo algumas ideias:

- Armazenar a tabela de símbolos do programa no arquivo executável, a fim de permitir a exibição de mensagens de erro baseadas no código fonte (e não no intermediário);
- Divisão dos programas em vários arquivos, permitindo a criação de bibliotecas de métodos, por exemplo;
- Sistema de exceções;
- Modificação na linguagem e na arquitetura para permitir o paradigma de orientação a objetos.
- Uma versão da MV-LSI baseada em registradores.

## Bibliografia

[LAUREANO e MAZIERO, 2008] Marcos Aurelio Pchek Laureano e Carlos Alberto Maziero. *Virtualização: Conceitos e Aplicações em Segurança*. Disponível em <<http://dainf.ct.utfpr.edu.br/~maziero/lib/exe/fetch.php/research:2008-sbseg-mc.pdf>>.

[MAZIERO, 2011] Carlos Alberto Maziero. *Sistemas Operacionais - IX - Máquinas Virtuais*. Disponível em <<http://dainf.ct.utfpr.edu.br/~maziero/lib/exe/fetch.php/so:so-cap09.pdf>>

[FURTADO, 2002] Olinto José Varela Furtado. *Linguagens Formais e Compiladores*. Disponível em <<http://www.inf.ufsc.br/~olinto/apostila-lfc.doc>>

[FURTADO, 2010] Olinto José Varela Furtado. *Capítulo VI - Análise Semântica*. Disponível em: <<http://www.inf.ufsc.br/~olinto/ine5622-cap6-102-p1.doc>>

[GESSER, 2003] Carlos Eduardo Gesser. *GALS - Gerador de Analisadores Léxicos e Sintáticos*. Disponível em <[http://projetos.inf.ufsc.br/arquivos\\_projetos/projeto\\_353/Gals.pdf](http://projetos.inf.ufsc.br/arquivos_projetos/projeto_353/Gals.pdf)>.

[VMWARE, 2007] *Understanding Full Virtualization, Paravirtualization and Hardware Assist*. Disponível em <[http://www.vmware.com/files/pdf/VMware\\_paravirtualization.pdf](http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf)>.

[POPEK e GOLDBERG, 1974] Gerald J. Popek e Robert P. Goldberg. "Formal Requirements for Virtualizable Third Generation Architectures".

[MOORE, 2009a] Scott Moore. *Pascal-P: The Portable Pascal compiler*. Disponível em: <<http://www.moorecad.com/standardpascal/PascalP.html>>.

[MOORE, 2009b] Scott Moore. *The P5 Compiler*. Disponível em: <<http://www.moorecad.com/standardpascal/p5.html>>.

[CHAMBERLAIN, 2001] John Chamberlain. *Microsoft's P-code Implementation*. Disponível em: <<http://www.programmersheaven.com/2/P-Code-Implementation>>.

[LINDHOLM et al., 2012] Tim Lindholm, Frank Yellin, Gilad Bracha e Alex Buckley. *The Java™ Virtual Machine Specification. Java SE 7 Edition*. Disponível em: <<http://docs.oracle.com/javase/specs/jvms/se7/jvms7.pdf>>.

[MITCHELL, 2000] John Mitchell. *What are the differences between "JVM" and "JRE" ?*. Disponível em: <<http://www.jguru.com/faq/view.jsp?EID=46212>>.

## Apêndice A - Artigo no formato SBC

### Uma Máquina Virtual para uso didático

Lucas Martins Silva<sup>1</sup>

<sup>1</sup>Departamento de Informática e Estatística (INE) – Universidade Federal de Santa Catarina (UFSC)  
Florianópolis - SC - Brasil  
lucasms@inf.ufsc.br

**Abstract.** *This paper approaches the development of a virtual machine (MV-LSI), adapted to the needs of the course of Information Systems at UFSC. The software will aid in teaching course subjects, especially Compilers, because the machine allows the execution of programs compiled and studied in the programming language of the course (LSI).*

*Key-words: virtual machines, compilers, code generation, interpreters, didactic computing*

**Resumo.** *Este trabalho aborda o desenvolvimento de uma máquina virtual (MV-LSI) adaptada às necessidades do curso de Sistemas de Informação da UFSC. O software auxiliará no ensino de disciplinas do curso, especialmente Compiladores, pois a máquina permite a execução de programas compilados e estudados na linguagem de programação própria da matéria (LSI).*

*Palavras-chave: máquinas virtuais, compiladores, geração de código, interpretadores, computação didática*

## 1. Introdução

No ensino de Compiladores no curso de Sistemas de Informação da UFSC, é utilizada uma linguagem de programação de alto nível, bastante simples e objetiva, que é levemente modificada a cada semestre. Com ela o professor tem a possibilidade de ensinar diversos aspectos da compilação de um programa, conceitos de compiladores, geração de código, entre outros. Entretanto, a máquina virtual existente é informal e muito limitada. A execução de programas da linguagem é feita manualmente — que é muito trabalhosa e suscetível a erros —, ou por meio de um interpretador simplificado, o que permite apenas a verificação de alguns aspectos. Além disso, o tempo disponibilizado para a matéria é bastante curto, não sobrando tempo para o desenvolvimento de um interpretador a cada semestre letivo.

Buscando resolver estes problemas e propiciar mais recursos que facilitem o aprendizado de tais conteúdos, neste trabalho será desenvolvida uma máquina virtual capaz de executar programas desta linguagem (chamada de *LSI - Linguagem de Sistemas de Informação*). Tal máquina deverá ser personalizável, suportando diferentes versões da linguagem fonte (LSI), bem como a experimentação de novos conceitos, por meio da inclusão de novas instruções e tipos à arquitetura.

O trabalho ainda tem como objetivo abordar os principais conceitos de máquinas virtuais de diversos tipos, o processo de geração de código intermediário em um

compilador para a MV-LSI, e otimizações simples de código. O programa gerado deve ser integrável a outros projetos (compiladores, por exemplo), além de permitir a criação de extensões (instruções e tipos extras).

## 2. Fundamentação Teórica

A máquina desenvolvida é classificada como Máquina Virtual de Aplicação. As máquinas virtuais de aplicação, ou de processo, servem apenas para a execução de um software sobre um sistema operacional qualquer. Elas não são feitas para a execução de um sistema operacional completo.

Nessa arquitetura, um programa é escrito em uma linguagem de programação que será interpretada pela máquina virtual, que será responsável por ler o código binário (*bytecode*) do programa e executá-lo utilizando as instruções da máquina real. A JVM (*Java Virtual Machine*) é um exemplo bastante conhecido, conforme a figura 1:

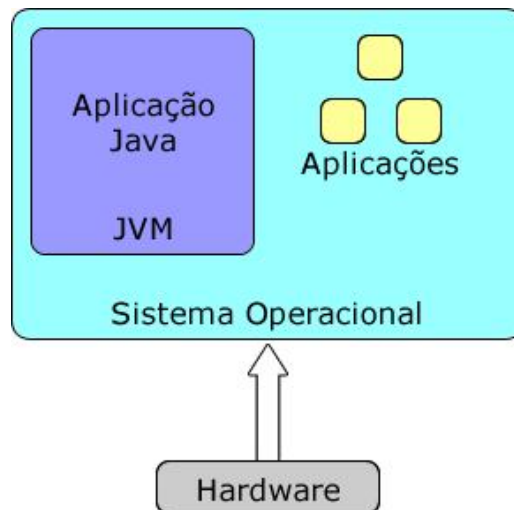


Figura 14: exemplo de máquina virtual de aplicação (Java Virtual Machine)

A implementação é feita utilizando a técnica de emulação de um hardware virtual, em que é desenvolvida toda uma arquitetura de hardware, que não será construída fisicamente, e sim emulada via software. Dessa forma é possível desenvolver aplicações que podem ser independentes de plataforma de hardware, desde que exista um emulador disponível para essa plataforma.

## 3. A Linguagem de Sistemas de Informação

A LSI - Linguagem de Sistemas de Informação é uma linguagem de programação estruturada e imperativa, muito influenciada pelo Pascal. A cada semestre, o professor da matéria faz modificações nessa linguagem, inserindo ou removendo estruturas e modificando características léxicas, sintáticas e semânticas. Sendo assim, a máquina virtual deve ser capaz de aceitar essas alterações sem grandes transtornos.

A especificação da linguagem é feita em três etapas: especificação léxica, sintática e semântica. As duas primeiras são totalmente formais, utilizando Expressões Regulares e Gramática Livre de Contexto, respectivamente. A especificação semântica

semi-formal, já que é feita por meio de "ações semânticas" durante a interpretação de um programa e por regras escritas em linguagem natural.

#### 4. A Máquina Virtual de Sistemas de Informação

A arquitetura da MV-LSI é relativamente simples e de fácil compreensão. Basicamente, é composta pelos seguintes itens:

- Tabela de instruções do programa
- Tabela de instruções disponíveis (ISA - *Instruction Set Architecture*)
- Tabela de literais: uma área da memória para armazenamento de literais. Literais são tipos de dados mais complexos, como cadeias de caracteres, que podem requerer muito espaço em memória para serem armazenados.
- Pilha de execução: o principal componente da MV. É onde tudo acontece, desde armazenamento de variáveis até chamada e retorno de métodos.
- PC: *program counter*, indica qual a próxima instrução a ser executada.
- PRA: pilha de registros de ativação, aponta o endereço base dos métodos em execução;
- Variáveis de controle: indicam se a MV está iniciada, finalizada, travada, parada ou no meio da execução de uma instrução.

A figura abaixo ilustra a arquitetura desenvolvida:

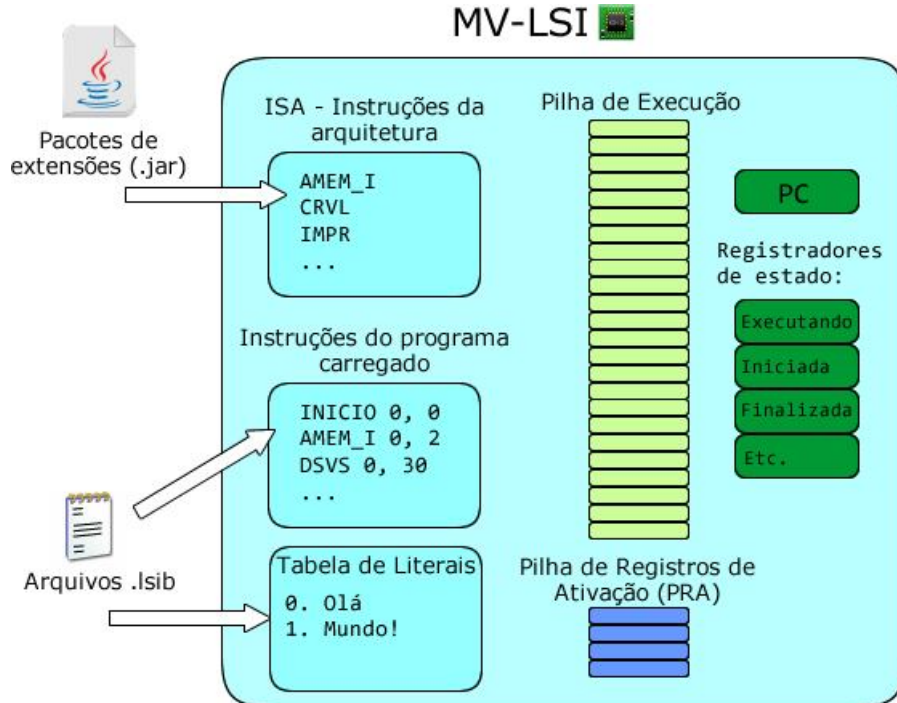


Figura 15: Arquitetura da MV-LSI

## 5. Conclusão

Este trabalho gerou uma importante ferramenta de aprendizado na área de compiladores, especialmente por estar completamente adaptada à metodologia adotada nesta disciplina na UFSC. Por meio da máquina virtual será possível dar um passo adiante na disciplina, permitindo a execução de forma simples e interativa de códigos gerados pelos alunos. Mais ainda, permite a programação de extensões, em que novas estruturas e paradigmas poderão ser experimentados no mesmo ambiente.

Como sugestão para trabalhos futuros, abaixo algumas ideias:

- Armazenar a tabela de símbolos do programa no arquivo executável, a fim de permitir a exibição de mensagens de erro baseadas no código fonte (e não no intermediário);
- Divisão dos programas em vários arquivos, permitindo a criação de bibliotecas de métodos, por exemplo;
- Sistema de exceções;
- Modificação na linguagem e na arquitetura para permitir o paradigma de orientação a objetos.
- Uma versão da MV-LSI baseada em registradores.

## Referências bibliográficas

[LAUREANO e MAZIERO, 2008] Marcos Aurelio Pchek Laureano e Carlos Alberto Maziero. *Virtualização: Conceitos e Aplicações em Segurança*. Disponível em <<http://dainf.ct.utfpr.edu.br/~maziero/lib/exe/fetch.php/research:2008-sbseg-mc.pdf>>.

[MAZIERO, 2011] Carlos Alberto Maziero. *Sistemas Operacionais - IX - Máquinas Virtuais*. Disponível em <<http://dainf.ct.utfpr.edu.br/~maziero/lib/exe/fetch.php/so:so-cap09.pdf>>

[FURTADO, 2002] Olinto José Varela Furtado. *Linguagens Formais e Compiladores*. Disponível em <<http://www.inf.ufsc.br/~olinto/apostila-lfc.doc>>

[FURTADO, 2010] Olinto José Varela Furtado. *Capítulo VI - Análise Semântica*. Disponível em: <<http://www.inf.ufsc.br/~olinto/ine5622-cap6-102-p1.doc>>

[GESSER, 2003] Carlos Eduardo Gesser. *GALS - Gerador de Analisadores Léxicos e Sintáticos*. Disponível em <[http://projetos.inf.ufsc.br/arquivos\\_projetos/projeto\\_353/Gals.pdf](http://projetos.inf.ufsc.br/arquivos_projetos/projeto_353/Gals.pdf)>.

[POPEK e GOLDBERG, 1974] Gerald J. Popek e Robert P. Goldberg. "Formal Requirements for Virtualizable Third Generation Architectures".

## Apêndice B - Código fonte do núcleo da MV-LSI

### ArrayListObserver.java

```
package UFSC.MaquinaVirtual;

import java.util.Collection;

/**
 * Interface padrão para observadores de objetos ArrayList<E>.
 * @author LucasMS
 */
public interface ArrayListObserver<E> {
    public void onAdd( E element );
    public void onAdd( int index, E element );
    public void onClear();
    public void onRemove( int index );
    public void onRemove( Object obj );
    public void onRemoveAll( Collection<?> c );
    public void onSet( int index, E element );
}
```

### FormatoIncorretoException.java

```
package UFSC.MaquinaVirtual;

/**
 * Esta excessão ocorre quando a MáquinaVirtual não consegue abrir um arquivo
 com programa em LSI-Binário.
 * @author LucasMS
 */
public class FormatoIncorretoException extends Exception {

    /**
     * Construtor padrão, sem mensagem.
     */
    public FormatoIncorretoException() {
    }

    /**
     * Construtor da Exception com mensagem de erro.
     * @param msg Mensagem de Erro.
     */
    public FormatoIncorretoException(String msg) {
        super(msg);
    }
}
```

### IndiceInvalidoException.java

```
package UFSC.MaquinaVirtual;

/**
 * Excessão utilizada quando o programa tenta acessar uma posição inválida em
 um array.
 * @author LucasMS
 */
public class IndiceInvalidoException extends Exception {
```



```

    /*
     * Método construtor
     * @param message Mensagem de erro.
     */
    public IndiceInvalidoException(String message) {
        super(message);
    }
}

```

## Instrucao.java

```

package UFSC.MaquinaVirtual;

import java.io.File;

/**
 *
 * @author LucasMS
 */
public abstract class Instrucao {
    protected short opcode;
    protected String mnemonico;

    protected double param64;
    protected int param1;
    protected int param2;
    protected boolean umparam = false;

    public abstract void executar(MaquinaVirtual maquina) throws
    TipoIncorretoException, IndiceInvalidoException, FormatoIncorretoException;

    public short getOpcode() { return this.opcode; }
    public double getParam64() { return this.param64; }
    public int getParam1() { return this.param1; }
    public int getParam2() { return this.param2; }
    public boolean umParam() { return this.umparam; }

    public void setParam1(int a) { this.param1 = a; }
    public void setParam2(int b) { this.param2 = b; }
    public void setParam64(double c) { this.param64 = c; }

    public String getMnemonico() { return this.mnemonico.toUpperCase(); }

    protected boolean ativada = false;
    protected File jar;

    public final void setActive(boolean a) { this.ativada = a; }
    public final boolean isActive() { return this.ativada; }
    public final File getJar() { return this.jar; }
    public final void setJar(File jarfile) { this.jar = jarfile; }

    @Override
    public final String toString() { return this.mnemonico.toUpperCase(); }

    public String getOpcodeString() {
        return Instrucao.opcodeToString(this.opcode);
    }
}

```

```

    public static String opcodeToString(short opcode) {
        String hex_opcode = Integer.toHexString(opcode).toUpperCase();
        String hex_opcode0s = "";
        if(hex_opcode.length() < 4) {
            for(int z = 0; z < 4-hex_opcode.length(); z++) {
                hex_opcode0s += "0";
            }
            hex_opcode = hex_opcode0s + hex_opcode;
        }
        return "0x"+hex_opcode;
    }
}

```

### InstrucaoNaoReconhecidaException.java

```

package UFSC.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class InstrucaoNaoReconhecidaException extends Exception {

    public InstrucaoNaoReconhecidaException() {
    }

    public InstrucaoNaoReconhecidaException(String msg) {
        super(msg);
    }
}

```

### InterrompidoException.java

```

package UFSC.MaquinaVirtual;

/**
 * Esta exceção deve ser iniciada pelo controlador da MV quando a Thread da
 * MV for interrompida com o método interrupt().
 * Deve ser usada para forçar uma parada da Máquina Virtual. Isso porquê
 * chamar o método parar() da MV não funcionará no caso desta estiver esperando
 * entrada de dados do usuário (método read() do controlador da MV), por
 * exemplo.
 * @author LucasMS
 */
public class InterrompidoException extends RuntimeException {
    public InterrompidoException() {
        super();
    }
}

```

### ItemPilha.java

```

package UFSC.MaquinaVirtual;

import java.io.File;

/**
 *
 * @author LucasMS

```

```

*/
public abstract class ItemPilha {

    protected Object valor;
    protected File jar;
    protected boolean tipo_literal = false;

    public ItemPilha() {}

    public ItemPilha(Object valor) {
        this.valor = valor;
    }

    public ItemPilha(int valor) {
        this.valor = new Integer(valor);
    }

    public ItemPilha(double valor) {
        this.valor = new Double(valor);
    }

    public ItemPilha(char valor) {
        this.valor = new Character(valor);
    }

    public ItemPilha(boolean valor) {
        this.valor = valor;
    }

    public Object getValor() {
        return valor;
    }

    public abstract String getTipo();
    public abstract int comparar(ItemPilha i);

    @Override
    public String toString() {
        return valor.toString();
    }

    public final File getJar() { return this.jar; }
    public final void setJar(File jarfile) { this.jar = jarfile; }
    public final void setTipoLiteral(boolean b) { this.tipo_literal = b; }
    public final boolean isTipoLiteral() { return this.tipo_literal; }
}

```

## Literal.java

```

package UFSC.MaquinaVirtual;

import java.io.File;

/**
 * Representa um objeto armazenado na tabela Heap da máquina virtual.
 * Esta classe deve ser estendida para suportar diferentes tipos de literais
 * (strings, por exemplo)
 * @author LucasMS
 */

```

```

public abstract class Literal {

    protected byte tipo;
    protected int tamanho;
    protected byte[] valor;
    protected boolean from_source_code; // se este objeto corresponde a um
valor digitado diretamente no código fonte do programa.
// se falso, indica que é um literal
construído em tempo de execução

    public Literal() {}

    public Literal(byte type, int size, byte[] value, boolean lit) {
        this(type, size, value);
        this.from_source_code = lit;
    }

    public Literal(byte type, int size, byte[] value) {
        this.tipo = type;
        this.tamanho = size;
        this.valor = value;
        this.from_source_code = false;
    }

    public byte getTipo() { return this.tipo; }
    public int getTamanho() { return this.tamanho; }
    public final byte[] getValor() { return this.valor.clone(); }

    public void setTamanho(int t) { this.tamanho = t; }
    public void setValor(byte[] v) { this.valor = v; }

    public abstract String getTipoString();
    @Override
    public abstract String toString();
    public String toEscapedString() {
        String str = this.toString();
        str = str.replaceAll("\\t", "\\t");
        str = str.replaceAll("\\n", "\\n");
        str = str.replaceAll("\\r", "\\r");
        return str;
    };

    public boolean fromSourceCode() { return this.from_source_code; }
    public void setFromSourceCode(boolean b) { this.from_source_code = b; }

    protected File jar;
    public final File getJar() { return this.jar; }
    public final void setJar(File jarfile) { this.jar = jarfile; }
}

```

## MaquinaVirtual.java

```

package UFSC.MaquinaVirtual;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.RandomAccessFile;
import java.math.BigInteger;

```

```

import java.util.ArrayList;
import java.util.Iterator;
import javax.xml.bind.DatatypeConverter;

/**
 * Classe que implementa todas as funcionalidades da máquina virtual LSI.
 * @author LucasMS
 */
public class MaquinaVirtual extends Thread {

    /**
     * Program Counter, indica qual a instrução será executada
     */
    public int PC;

    /**
     * A "pilha" de execução. Na verdade é um ArrayList com alguns métodos
    adicionais
     */
    public PilhaExecucao<ItemPilha> pilha;

    /**
     * Tabela de strings, literais, etc.
     */
    public ObservableArrayList<Literal> literais;

    /**
     * Contém o conjunto de instruções do programa
     */
    public ObservableArrayList<Instrucao> instrucoes;

    /**
     * Sequência de 4 bytes que deve estar presente no início de um arquivo
    executável da MV-LSI.
     * É uma forma de identificar facilmente os arquivos que podem ser
    executados ou não.
     */
    public static int numero_magico = 0x4C534942;

    // Contém o conjunto de instruções da máquina virtual
    private ArrayList<Instrucao> instruction_set;

    private ArrayList<Literal> tiposliterais;

    // Objeto que está no controle desta máquina virtual, responsável por
    enviar comandos para iniciar execução, parar, etc.
    private MaquinaVirtualInterface controller;

    private int velocidade = 100;
    private final int max_vel = 0;
    private final int min_vel = 500;

    private final Object wait = new Object(); // usado para deixar a thread
    aguardando em alguns casos, como a entrada de dados pelo usuário

    // número de instruções executadas na execução atual
    private int num_execucoes = 0;

    // arquivo de um executável carregado na máquina virtual

```

```

private File arquivo;

// Não utilizado ainda. Servirá para dar à máquina virtual a
possibilidade de executar arquivos de diferentes versões do formato de
arquivo LSIB.
private byte arquivo_versao;

// utilizado para dar a possibilidade de executar uma instrução a
qualquer momento (método executarPasso)
private boolean passo = false;

/**
 * Variáveis usadas no controle de estado da Máquina Virtual
 */
private boolean travou = false; // indica a ocorrência de alguma falha na
execução do programa
private boolean iniciou = false; // indica que a MV iniciou a execução do
programa atual
private boolean finalizou = false; // indica se a execução do programa
carregado chegou ao fim
private boolean instrucao_executando = false; // indica se a máquina
virtual está no meio da execução de uma instrução

/**
 * Indica se a MV está parada ou executando. Não confundir com a execução
da Thread.
 * AVISO: chamar o método executarPasso() não irá alterar o valor deste
estado. Este método executará uma instrução na hora em que for chamado,
 * não importando se a MV está parada ou já executando.
 */
private boolean parou = true;

/**
 * Único construtor da MáquinaVirtual. Recebe como parâmetro um objeto da
classe {@link MaquinaVirtualInterface}.
 * @param mvc objeto da classe {@link MaquinaVirtualInterface} que está
no controle desta máquina virtual
 */
public MaquinaVirtual(MaquinaVirtualInterface mvc) {
    if(mvc == null) {
        throw new IllegalArgumentException("O parâmetro mvc (tipo
MaquinaVirtualController) não pode ser null.");
    }

    this.setName("MaquinaVirtual");

    this.controller = mvc;
    this.pilha = new PilhaExecucao();
    this.instrucoes = new ObservableArrayList();
    this.instrucao_set = new ArrayList();
    this.literais = new ObservableArrayList();
    this.tiposliterais = new ArrayList();
}

@Override
public void run() {
    while(true) {
        synchronized(this.wait) {
            while(this.parou && !passo) {

```

```

        try {
            this.wait.wait();
        } catch (InterruptedException e) {}
    }
}

this.executarProxima();

if(!passo) {
    if(this.velocidade > 0) {
        try {
            MaquinaVirtual.sleep(this.velocidade);
        } catch (InterruptedException e) {}
    }
}

passo = false;
}
}

/**
 * Método usado para executar uma instrução imediatamente.
 * Este método pode ser chamado enquanto a máquina estiver parada (não
 * tiver sido iniciada com {@link iniciar()} , ou quando tiver sido parada com
 * {@link parar()} ).
 * <b>AVISO:</b> este método não executará nenhuma instrução enquanto a
 * execução instrução anterior não terminar.
 */
public void executarPasso() {
    if(this.travou() || this.instrucao_executando) { return; }

    if(!this.iniciou() || this.finalizou()) {
        this.preparar();
        this.iniciou = true;
    }

    synchronized(this.wait) {

        if(this.executando()) {
            // quando ocorrer uma chamada de executarPasso() e a VM
            // estiver executando (parou=false),
            // a thread poderá estar "dormindo", fazendo com que o passo
            // não seja executado imediatamente.
            this.interrupt();
        }

        this.passo = true;
        this.wait.notifyAll();

        if(this.finalizou()) {
            this.resetar();
        }
    }
}

// executa apenas 1 instrução
private void executarProxima() {

```

```

// não é possível executar duas ou mais instruções ao mesmo tempo
// isso vale para instruções que esperam dados do usuário, por
exemplo
if(this.instrucao_executando) { return; }

this.instrucao_executando = true;

Instrucao inst = null;

try {
    inst = this.instrucoes.get(PC);
} catch(IndexOutOfBoundsException e) {
    this.parar();
    this.controller.erro("O programa tentou desviar para uma
instrução inexistente.");
    return;
}

try {
    // informa à interface de usuário que a instrução PC será
executada
    this.controller.executando(PC);
    inst.executar(this);
} catch(InterrompidoException e) {
    this.parar();
} catch(TipoIncorretoException e) {
    this.travou = true;
    this.parar();
    this.controller.erro(e.getMessage());
} catch(IndiceInvalidoException e) {
    this.travou = true;
    this.parar();
    this.controller.erro(e.getMessage());
} catch(FormatoIncorretoException e) {
    this.travou = true;
    this.parar();
    this.controller.erro(e.getMessage());
} catch(NullPointerException e) {
    this.travou = true;
    this.parar();
    this.controller.erro(e.getMessage());
} catch(RuntimeException e) {
    this.travou = true;
    this.parar();
    this.controller.erro(e.getMessage());
} finally {
    this.num_execucoes++;
    this.instrucao_executando = false;
}
}

/**
 * Informa se a máquina virtual está executando. Não confundir com o
estado da Thread da máquina virtual.
 * @return true se a máquina estiver executando, false caso contrário.
Chamadas para o método {@link executarPasso()} enquanto a máquina virtual
estiver parada não faz com que este valor se transforme em verdadeiro.
 */

```



```

    public boolean executando() { return !this.parou; }

    /**
     * Informa se a máquina virtual está parada. Não confundir com o estado
da Thread da máquina virtual.
     * @return true se a máquina estiver executando, false caso contrário.
     */
    public boolean parou() { return this.parou; }

    public boolean travou() { return this.travou; }

    public boolean iniciou() { return this.iniciou; }

    /**
     *
     * @return
     */
    public boolean finalizou() { return this.finalizou; }
    /**
     *
     * @param ui
     */
    public void setController(MaquinaVirtualInterface ui) { this.controller =
ui; }
    /**
     *
     * @return
     */
    public MaquinaVirtualInterface getController() { return this.controller;
}

    /**
     *
     * @param arquivo
     * @throws FileNotFoundException
     * @throws FormatoIncorretoException
     * @throws InstrucaoNaoReconhecidaException
     */
    public void carregarPrograma(File arquivo) throws FileNotFoundException,
FormatoIncorretoException,
InstrucaoNaoReconhecidaException,
TipoLiteralNaoReconhecidoException,
IOException {

        if(this.instrucao_executando) {
            throw new IllegalStateException("Não é possível carregar outro
programa durante a execução de uma instrução.");
        }

        this.fecharArquivo(false);

        this.arquivo = arquivo;

        RandomAccessFile file = new RandomAccessFile(arquivo, "r");

        try {

```

```

        // verifica se o formato do arquivo corresponde a um executável
da máquina virtual LSI.
        int magic = file.readInt();
        if(magic == numero_magico) {

            this.arquivo_versao = file.readByte();

            int num_literais = file.readInt();

            this.instrucoes.clear();
            this.literais.clear();

            short numconst = 0;

            while(numconst < num_literais) {
                int tamconst = file.readInt();
                byte tipoconst = file.readByte();
                byte[] valorconst = new byte[tamconst];

                for(int i = 0; i < tamconst; i++) {
                    byte bytelido = file.readByte();
                    valorconst[i] = bytelido;
                }

                try {
                    Literal constante =
this.getNewTipoLiteralInstance(tipoconst);
                    constante.setTamanho(tamconst);
                    constante.setValor(valorconst);
                    constante.setFromSourceCode(true);
                    this.literais.add(constante);
                } catch(TipoLiteralNaoReconhecidoException e) {
                    this.fecharArquivo(false);
                    throw e;
                }

                numconst++;
            }

            int numinst = file.readInt();

            for(int x = 0; x < numinst; x++) {
                short instrucao = file.readShort();

                double param64 = file.readDouble();
                file.seek(file.getFilePointer() - 8);
                int param1 = file.readInt();
                int param2 = file.readInt();

                try {
                    Instrucao inst =
this.getNewInstructionInstance(instrucao);
                    inst.setParam1(param1);
                    inst.setParam2(param2);
                    inst.setParam64(param64);
                    this.instrucoes.add(inst);
                } catch(InstrucaoNaoReconhecidaException e) {
                    this.fecharArquivo(false);
                    throw e;
                }
            }
        }
    }
}

```

```

        }
    }

    this.controller.abriu_programa();

    } else {
        this.fecharArquivo(false);
        throw new FormatoIncorretoException("O arquivo não é um
executável da MV-LSI!");
    }
} catch(IOException e) {
    file.close();
    this.fecharArquivo(false);
    throw e;
}
}

public void fecharArquivo() {
    this.fecharArquivo(true);
}

/**
 *
 */
public void fecharArquivo(boolean tellcontroller) {
    this.interrupt();

    this.pilha.clear();
    this.PC = 0;
    this.finalizou = false;
    this.parou = true;
    this.iniciou = false;
    this.num_execucoes = 0;

    this.instrucao_executando = false;
    this.instrucoes.clear();
    this.literais.clear();
    this.arquivo = null;
    this.arquivo_versao = 0;

    if(tellcontroller) {
        this.controller.fechou_programa();
    }
}

/*
 * Métodos de controle da máquina virtual: iniciar, parar, reiniciar,
finalizar
 */

// para iniciar a execução do programa
/**
 *
 */
public void iniciar() {
    this.preparar();
}

```

```

        this.iniciou = true;
        this.parou = false;

        synchronized(this.wait) {
            this.wait.notifyAll();
        }

        this.controller.iniciou();
    }

    /**
     * Deixa a MV pronta para iniciar uma nova execução (seja pela invocação
do método iniciar() ou executarPasso())
     */
    private void preparar() {
        // remove objetos da heap que não sejam literais.
        Iterator i = this.literais.iterator();
        while(i.hasNext()) {
            Literal o = (Literal)i.next();
            if(!o.fromSourceCode()) i.remove();
        }

        this.pilha.clear();
        this.pilha.fillBase();
        this.PC = 0;
        this.finalizou = false;
        this.parou = true;
        this.num_execucoes = 0;
        this.travou = false;
    }

    /**
     * Para parar a execução do programa. A MV irá parar após o término da
execução da instrução atual.
     */
    public void parar() {
        this.parou = true;
        this.controller.parou();
    }

    // para reiniciar a execução do programa previamente parado com parar()
    /**
     *
     */
    public void reiniciar() {
        this.parou = false;

        synchronized(this.wait) {
            this.wait.notifyAll();
        }

        this.controller.reiniciou();
    }

    /**
     * Retorna para as configurações iniciais, deixando a máquina pronta para
uma nova execução.
     */
    public void resetar() {

```

```

        this.pilha.clear();
        this.PC = 0;
        this.finalizou = false;
        this.parou = true;
        this.num_execucoes = 0;
        this.travou = true;
        this.iniciou = false;

        this.controller.resetou();
    }

    /**
     * Esse método não deve ser chamado pela interface do usuário, apenas
     * pela instrução FIM (ou equivalente)
     */
    public void finalizar() {
        this.parou = true;
        this.finalizou = true;
        this.controller.finalizou();
    }

    /**
     * Adiciona um observador à pilha da máquina virtual.
     * @param ob Um objeto da classe {@link ArrayListObserver}.
     */
    public void addPilhaObserver(ArrayListObserver ob) {
        this.pilha.registerObserver(ob); }

    /**
     * Adiciona um observador à tabela de literais da máquina virtual.
     * @param ob Um objeto da classe {@link ArrayListObserver}.
     */
    public void addLiteraisObserver(ArrayListObserver ob) {
        this.literais.registerObserver(ob); }

    /**
     * Adiciona um observador ao conjunto de instruções do programa carregado
     * na máquina virtual.
     * @param ob Um objeto da classe {@link ArrayListObserver}.
     */
    public void addInstrucoesObserver(ArrayListObserver ob) {
        this.instrucoes.registerObserver(ob); }

    /**
     * Adiciona um observador na tabela de registradores-base da máquina
     * virtual.
     * @param ob Um objeto da classe {@link ArrayListObserver}.
     */
    public void addRegistradoresBaseObserver(ArrayListObserver ob) {
        this.pilha.addBaseObserver(ob); }

    /**
     * Método utilizado para ajustar a velocidade de execução da máquina.
     * @param velocidade Um valor inteiro entre 0 e 100, sendo 100 o mais
     * rápido.
     */
    public void ajustarVelocidade(int velocidade) {
        // velocidade = 0 a 100, sendo 100 o mais rapido
        int v2 = 100 - velocidade;

```

```

        int vel = max_vel+(((min_vel-max_vel)/100)*v2);
        this.velocidade = vel;
    }

    /**
     * Retorna a quantidade de instruções do programa carregado.
     * @return Número de instruções do programa carregado na máquina.
     */
    public int getNumeroInstrucoes() { return this.instrucoes.size(); }

    /**
     * Retorna o número de instruções executadas até o momento. Este campo é
     zerado toda vez que o programa termina de executar.
     * @return Número de instruções executadas pela máquina virtual.
     */
    public int getNumeroInstrucoesExecutadas() { return this.num_execucoes; }

    /**
     * Retorna a instrução <code>index</code> do conjunto de instruções do
     programa carregado.
     * @param index Índice da instrução, começando em 0.
     * @return O objeto da classe {@link Instrucao} com índice index.
     */
    public Instrucao getInstrucao(int index) { return
this.instrucoes.get(index); }

    /**
     * Método utilizado pela
     * @param opcode
     * @return
     * @throws InstrucaoNaoReconhecidaException
     */
    private Instrucao getNewInstructionInstance(short opcode) throws
InstrucaoNaoReconhecidaException {

        Iterator it = this.instruction_set.iterator();

        try {
            while(it.hasNext()) {
                Instrucao inst = (Instrucao)it.next();
                if(inst.isActive() && inst.getOpcode() == opcode) {
                    return (Instrucao)inst.getClass().newInstance();
                }
            }
        } catch( InstantiationException e) {
            javax.swing.JOptionPane.showMessageDialog(null, "Ocorreu um erro
ao obter instância de instrução. \n"+e.getMessage(), "Erro",
javax.swing.JOptionPane.ERROR_MESSAGE);
            System.exit(1);
        } catch( IllegalAccessException e) {
            javax.swing.JOptionPane.showMessageDialog(null, "Ocorreu um erro
ao obter instância de instrução. \n"+e.getMessage(), "Erro",
javax.swing.JOptionPane.ERROR_MESSAGE);
            System.exit(1);
        }

        throw new InstrucaoNaoReconhecidaException("Não há nenhuma instrução
ativa com o Opcode "+Instrucao.opcodeToString(opcode));
    }
}

```

```

private Literal getNewTipoLiteralInstance(byte tipoconst) throws
TipoLiteralNaoReconhecidoException {

    Iterator it = this.tiposliterais.iterator();

    try {
        while(it.hasNext()) {
            Literal literal = (Literal)it.next();
            if(literal.getTipo() == tipoconst) {
                return (Literal)literal.getClass().newInstance();
            }
        }
    } catch( InstantiationException e) {
        javax.swing.JOptionPane.showMessageDialog(null, "Ocorreu um erro
ao obter instância de tipo literal. \n"+e.getMessage(), "Erro",
javax.swing.JOptionPane.ERROR_MESSAGE);
        System.exit(1);
    } catch( IllegalAccessException e) {
        javax.swing.JOptionPane.showMessageDialog(null, "Ocorreu um erro
ao obter instância de tipo literal. \n"+e.getMessage(), "Erro",
javax.swing.JOptionPane.ERROR_MESSAGE);
        System.exit(1);
    }

    throw new TipoLiteralNaoReconhecidoException("O tipo literal
"+Byte.toString(tipoconst)+" não existe nos pacotes de extensões
carregados");
}

/**
 *
 */
public void clearInstructionSet() { this.instruction_set.clear(); }

public void clearTiposLiterais() { this.tiposliterais.clear(); }

/**
 * Adiciona uma instrução ao conjunto de instruções (ISA) disponíveis
nesta máquina virtual.
 * @param inst
 * @return true caso a instrução tenha sido incluída com sucesso, false
caso já exista uma instrução com o mesmo op-code no ISA desta MV.
 */
public boolean addInstructionToISA(Instrucao inst) {
    for(Instrucao i : this.instruction_set) {
        // evita que duas instruções do ISA possuam o mesmo op-code
        if(inst.getOpcode() == i.getOpcode()) {
            return false;
        }
    }

    this.instruction_set.add(inst);
    return true;
}

/**
 * Remove uma instrução do conjunto de instruções disponíveis nesta
máquina virtual. O op-code da instrução é o parâmetro utilizado para

```

identificar a instrução a ser removida.

```
* @param inst
* @return true caso alguma instrução tenha sido removida, false se
nenhuma instrução foi removida.
*/
public boolean removeInstructionFromISA(Instrucao inst) {
    Iterator it = this.instruction_set.iterator();
    while(it.hasNext()) {
        Instrucao i = (Instrucao)it.next();
        if(i.getOpcode() == inst.getOpcode()) {
            it.remove();
            return true;
        }
    }
    return false;
}

public boolean addTipoLiteral(Literal tipolit) {
    for(Literal i : this.tiposliterais) {
        if(i.getTipo() == tipolit.getTipo()) {
            return false;
        }
    }

    this.tiposliterais.add(tipolit);
    return true;
}

public boolean removeTipoLiteral(Literal tipolit) {
    Iterator it = this.tiposliterais.iterator();
    while(it.hasNext()) {
        Literal i = (Literal)it.next();
        if(i.getTipo() == tipolit.getTipo()) {
            it.remove();
            return true;
        }
    }
    return false;
}

public boolean removeInstruction(int instIndex) {
    try {
        this.instrucoes.remove(instIndex);
        return true;
    } catch(IndexOutOfBoundsException e) {
        return false;
    }
}

public boolean insertInstruction(int index, short opcode, int param1, int
param2, double param64) {
    if(index >= 0 && index < (this.instrucoes.size()+1)) {
        try {
            Instrucao inst = this.getNewInstructionInstance(opcode);
            inst.setParam1(param1);
            inst.setParam2(param2);
            inst.setParam64(param64);
            this.instrucoes.add(index, inst);
        }
    }
}
```



```

        return true;
    } catch(InstrucaoNaoReconhecidaException e) {
        return false;
    }
}
return false;
}

public boolean editInstruction(int instIndex, int what, String value) {
    if(instIndex >= 0 && instIndex < this.instrucoes.size()) {
        switch(what) {
            case 0: // mnemonico

                break;
            case 1: // opcode
                if(value.toLowerCase().startsWith("0x")) { value =
value.substring(2); } else { return false; }
                short new_opcode = Short.parseShort(value, 16);

                /*
                boolean found = false;
                for(Instrucao inst : this.instruction_set) {
                    if(inst.opcode == new_opcode) {
                        found = true; break;
                    }
                }
                if(!found) { return false; }
                */

                try {
                    Instrucao inst =
this.getNewInstructionInstance(new_opcode);

                    inst.setParam1(this.instrucoes.get(instIndex).param1);
                    inst.setParam2(this.instrucoes.get(instIndex).param2);
                    inst.setParam64(this.instrucoes.get(instIndex).param64);

                    this.instrucoes.set(instIndex, inst);
                } catch(InstrucaoNaoReconhecidaException e) {
                    return false;
                }
                break;
            case 2: // parametro 1
                try {
                    this.instrucoes.get(instIndex).param1 =
Integer.parseInt(value);
                } catch(NumberFormatException e) {
                    return false;
                }
                break;
            case 3: // parametro 2
                try {
                    this.instrucoes.get(instIndex).param2 =
Integer.parseInt(value);
                } catch(NumberFormatException e) {
                    return false;
                }
            }
        }
    }
}

```

```

        break;
    }
    return true;
} else {
    return false;
}
}
}
}

```

## MaquinaVirtualInterface.java

```

package UFSC.MaquinaVirtual;

/**
 * Interface utilizada para representar o objeto que possui o controle sobre
 a Máquina Virtual.
 * @author LucasMS
 */
public interface MaquinaVirtualInterface {

    /**
     * Método para ler e retornar o input do usuário.
     * @return O String que o usuário digitou.
     */
    public String read();

    /**
     * Método chamado quando a Máquina Virtual precisar imprimir algo na
 saída.
     * @param texto
     */
    public void write(String texto);

    /**
     * Método chamado quando a MV inicia a execução de um programa.
     */
    public void iniciou();

    /**
     * Chamado quando a MV para a execução do programa (por exemplo, ao
 chamar o método {@link MaquinaVirtual#parar()}).
     */
    public void parou();

    /**
     * Chamado quando a MV reinicia a execução do programa anteriormente
 parado.
     */
    public void reiniciou();

    /**
     * Chamado quando a MV reseta para o estado inicial.
     */
    public void resetou();

    /**
     * Chamado quando a MV termina a execução do programa (por exemplo,
 quando a instrução FIM ou outra equivalente chamar o método {@link
 MaquinaVirtual#finalizar()}).
     */

```

```

    */
    public void finalizou();

    /**
     * Método chamado imediatamente antes de iniciar a execução da instrução
PC.
     * @param PC Número da instrução no programa fonte, iniciando em zero.
     */
    public void executando(int PC);

    /**
     * Método utilizado para informar o controlador que a MáquinaVirtual
encontrou um erro na execução e não pode continuar.
     * @param mensagem A mensagem de erro.
     */
    public void erro(String mensagem);

    public void fechou_programa();

    public void abriu_programa();
}

```

## ObservableArrayList.java

```

package UFSC.MaquinaVirtual;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

/**
 *
 * @author LucasMS
 */
public class ObservableArrayList<E> extends ArrayList<E> {
    private List<ArrayListObserver<E>> observers = null;

    @Override
    public E get(int index) {
        return super.get(index);
    }

    @Override
    public boolean add(E element) {
        boolean result = super.add(element);

        if(result) {
            Iterator<ArrayListObserver<E>> it =
this.getObservers().iterator();
            while(it.hasNext()) {
                it.next().onAdd(element);
            }
        }

        return result;
    }

    @Override
    public void add(int index, E element) {

```

```

        super.add(index, element);

        Iterator<ArrayListObserver<E>> it = this.getObservers().iterator();
        while(it.hasNext()) {
            it.next().onAdd(index, element);
        }
    }

    @Override
    public void clear() {
        super.clear();

        Iterator<ArrayListObserver<E>> it = this.getObservers().iterator();
        while(it.hasNext()) {
            it.next().onClear();
        }
    }

    @Override
    public E remove(int index) {
        E toRet = super.remove(index);

        Iterator<ArrayListObserver<E>> it = this.getObservers().iterator();
        while(it.hasNext()) {
            it.next().onRemove(index);
        }

        return toRet;
    }

    @Override
    public boolean remove(Object obj) {
        boolean result = super.remove(obj);
        if (result) {
            Iterator<ArrayListObserver<E>> it =
this.getObservers().iterator();
            while(it.hasNext()) {
                it.next().onRemove(obj);
            }
        }
        return result;
    }

    @Override
    public E set( int index, E element ) {
        E toRet = super.set( index, element );

        Iterator<ArrayListObserver<E>> it = this.getObservers().iterator();
        while(it.hasNext()) {
            it.next().onSet(index, element);
        }

        return toRet;
    }

    public List<ArrayListObserver<E>> getObservers() {
        if ( observers == null ) {
            observers = new ArrayList<ArrayListObserver<E>>();
        }
    }

```

```

        return observers;
    }

    public void registerObserver( ArrayListObserver<E> observer ) {
        this.getObservers().add(observer);
    }

    public void unregisterObserver( ArrayListObserver<E> observer ) {
        this.getObservers().remove(observer);
    }
}

```

## PilhaExecucao.java

```

package UFSC.MaquinaVirtual;

/**
 * Classe que representa a "pilha" de execução. Na verdade não é uma pilha
 * (LIFO) propriamente dita,
 * pois deve ser possível acessar ou alterar qualquer dado na estrutura.
 * Permite o monitoramento por meio de observers.
 * @author LucasMS
 */
public class PilhaExecucao<E> extends ObservableArrayList<E> {

    /**
     * Variável utilizada para indicar o topo da pilha.
     */
    public int topo; // indica o topo da pilha de execução

    /**
     * Representa a PRA (Pilha de Registros de Ativação), utilizada quando a
     * Máquina Virtual muda de nível léxico (por exemplo, em uma chamada de método).
     * O acesso direto ao atributo base foi bloqueado em razão da necessidade
     * da garantia que os valores inseridos nesta estrutura
     * sejam apenas objetos do tipo Integer. Caso um elemento de tipo
     * primitivo inteiro fosse adicionado, ocorreriam problemas quanto a
     * determinação do
     * índice de um valor na estrutura.
     *
     * Tome como exemplo o código abaixo:
     *
     * for(int x = 0; x < 10; x++) {
     *     base.add(0);
     * }
     *
     * Seriam adicionados dez valores primitivos de valor 0. Desta forma, é
     * impossível determinar em qual posição cada valor está usando o método
     * indexOf.
     * Com o bloqueio ao acesso direto à estrutura, foram criados alguns
     * métodos para garantir que todo valor inserido seja do tipo Integer.
     * Assim, a estrutura pode ter diversos elementos Integer contendo o
     * valor 0, que todos serão considerados diferentes e identificáveis pelo método
     * indexOf.
     *
     * Independente da forma como fosse implementado, a máquina virtual
     * funcionaria corretamente. O problema seria apenas na exibição correta da
     * estrutura
     * da PRA (índice e valor correspondente) em uma interface gráfica

```

utilizando o esquema de observadores-observado.

```
    */
    private ObservableArrayList base; // indica o início de um nível na pilha
de execução

    public PilhaExecucao() {
        base = new ObservableArrayList() {
            /*
             * O método padrão indexOf utiliza o método equals dos elementos
para encontrar o índice.
             * Neste ArrayList precisamos que a comparação seja feita com
base nas referências dos elementos.
             */
            @Override
            public int indexOf(Object o) {
                for(int x = 0; x < this.size(); x++) {
                    if((Object)this.get(x) == o) {
                        return x;
                    }
                }
                return -1;
            }
        };
    }

    /*
     * Método utilizado para apagar todos os dados da pilha, inclusive topo e
a PRA (Pilha de Registros de Ativação).
     * A PRA é iniciada com o valor 0 na posição 0.
     */
    @Override
    public void clear() {
        super.clear();
        this.topo = -1;
        this.base.clear();
    }

    /*****
     Métodos para acessar e alterar valores da PRA.
    *****/

    /**
     * Adiciona um valor na pilha de registros de ativação.
     * @param valor O índice da pilha que corresponde à base de novo nível
léxico.
     * @return true
     */
    public boolean addBase(int valor) {
        Integer val = new Integer(valor);
        return this.base.add(val);
    }

    public Integer setBase(int indice, int valor) {
        return (Integer)this.base.set(indice, new Integer(valor));
    }

    public Integer getBase(int indice) {
        return (Integer)this.base.get(indice);
    }
}
```

```

    public void addBaseObserver(ArrayListObserver<E> observer) {
        this.base.registerObserver(observer);
    }

    public void removeBaseObserver(ArrayListObserver<E> observer) {
        this.base.unregisterObserver(observer);
    }

    public int baseIndexOf(Object item) {
        return this.base.indexOf(item);
    }

    public void fillBase() {
        for(int x = 0; x < 50; x++) {
            this.addBase(0);
        }
    }
}

```

#### TipoIncorretoException.java

```

package UFSC.MaquinaVirtual;

import javax.swing.JOptionPane;

/**
 *
 * @author LucasMS
 */
public class TipoIncorretoException extends Exception {

    public TipoIncorretoException(String message) {
        super(message);
    }
}

```

#### TipoLiteralNaoReconhecidoException.java

```

package UFSC.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class TipoLiteralNaoReconhecidoException extends Exception {

    public TipoLiteralNaoReconhecidoException() {
    }

    public TipoLiteralNaoReconhecidoException(String msg) {
        super(msg);
    }
}

```

## Apêndice C - Código fonte da interface gráfica

GerenciadorExtensoes.java

```
package UFSC.InterfaceMVLISI;

import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.Instrucao;
import java.awt.HeadlessException;
import java.io.*;
import java.net.MalformedURLException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.prefs.*;
import javax.swing.JFileChooser;
import javax.swing.JOptionPane;
import javax.swing.ListSelectionModel;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
import javax.swing.event.TableModelEvent;
import javax.swing.event.TableModelListener;
import javax.swing.table.DefaultTableModel;

/**
 *
 * @author LucasMS
 */
public class GerenciadorExtensoes extends javax.swing.JFrame {

    /**
     * Creates new form GerenciadorExtensoes
     */
    public GerenciadorExtensoes() {
        initComponents();

        this.tabelaInstrucoes.getColumnModel().getColumn(0).setMaxWidth(100);
        this.tabelaInstrucoes.getColumnModel().getColumn(2).setMaxWidth(100);

        this.tabelaPacotes.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);

        this.tabelaInstrucoes.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);

        this.tabelaTipos.getColumnModel().getColumn(0).setMaxWidth(100);

        this.tabelaPacotes.getSelectionModel().addListSelectionListener(new
        ListSelectionListener() {
            @Override
            public void valueChanged(ListSelectionEvent e) {
                atualizaTabelaInstrucoes();
                atualizaTabelaTipos();
            }
        });

        this.tabelaInstrucoes.getModel().addTableModelListener(new
        TableModelListener() {
            @Override
            public void tableChanged(TableModelEvent e) {
                int col = e.getColumn();
```



```

        int row = e.getLastRow();

        if(col < 0 || row < 0 || col != 2) { return; }

        Instrucao inst =
(Instrucao)tabelaInstrucoes.getModel().getValueAt(row, 0);
        boolean ativada =
(Boolean)tabelaInstrucoes.getModel().getValueAt(row, 2);

        try {
            ArrayList<Instrucao> instrucoes =
InterfaceMVLIS.getInstance().getIsa();

            if(ativada) {
                int inst_iguais_ativadas = 0;

                // verifica por outras instruções com o mesmo opcode
                for(int x = 0; x < instrucoes.size(); x++) {
                    Instrucao cli = instrucoes.get(x);
                    // essa comparação é possível pois cli e inst
apontam para A MESMA INSTÂNCIA!
                    if(cli != inst) {
                        // podem ocorrer de duas instruções
diferentes com o mesmo opcode
                        if(cli.getOpcode() == inst.getOpcode()) {
                            // se a outra instrucao estiver ativada,
e estiver tentando ativar uma instrucao com o mesmo opcode
                            if(cli.isActive()) {
                                inst_iguais_ativadas++;
                                int resp =
JOptionPane.showConfirmDialog(null, "Existe uma instrução
("+cli.getMnemonic()+") ativada com o mesmo opcode
("+cli.getOpcodeString()+") no pacote
abaixo:\n\n"+cli.getJar().getCanonicalPath()+"\n\n Não podem existir duas
instruções ativadas com o mesmo opcode ao mesmo tempo. Deseja desativar a
outra instrução?", "Confirmação", JOptionPane.YES_NO_OPTION,
JOptionPane.QUESTION_MESSAGE);

                                if(JOptionPane.YES_OPTION == resp) {
                                    // se sim, precisamos desativar a
outra instrução
                                    // e também verificar se a outra
instrução está no mesmo pacote. Se estiver,
                                    // também é necessário desmarcar
o checkbox correspondente a essa instrução
                                    cli.setActive(false);
                                    inst.setActive(true);

                                    if(cli.getJar().getCanonicalPath().equals(inst.getJar().getCanonicalPath()))
                                    {
                                        int t_rows =
tabelaInstrucoes.getModel().getRowCount();
                                        for(int y = 0; y < t_rows;
y++) {
                                            Instrucao t_cli =
(Instrucao)tabelaInstrucoes.getModel().getValueAt(y, 0);
                                            if(t_cli == cli) {
                                                tabelaInstrucoes.getModel().setValueAt(false, y, 2);
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }

```

```

        }
    }
} else if(JOptionPane.NO_OPTION ==
resp) {
    inst.setActive(false);

tabelaInstrucoes.getModel().setValueAt(false, row, 2);
    }
    }
    }
    }
    }
    if(inst_iguais_ativadas == 0) {
        inst.setActive(true);
    }
    } else {
        inst.setActive(false);
    }
    }

    salvarInstrucoesAtivas();

    } catch(IOException ex) {
    } catch(HeadlessException ex) {
    }
    }
});

//
this.atualizaTabelaPacotes();
}

/**
 * This method is called from within the constructor to initialize the
form.
 * WARNING: Do NOT modify this code. The content of this method is always
 * regenerated by the Form Editor.
 */
SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {

    addJarButton = new javax.swing.JButton();
    jScrollPane1 = new javax.swing.JScrollPane();
    tabelaInstrucoes = new javax.swing.JTable();
    okButton = new javax.swing.JButton();
    jLabel1 = new javax.swing.JLabel();
    jScrollPane2 = new javax.swing.JScrollPane();
    tabelaPacotes = new javax.swing.JTable();
    jLabel2 = new javax.swing.JLabel();
    removerPacoteButton = new javax.swing.JButton();
    jLabel3 = new javax.swing.JLabel();
    ativarTodasBtn = new javax.swing.JButton();
    jLabel4 = new javax.swing.JLabel();
    jScrollPane3 = new javax.swing.JScrollPane();
    tabelaTipos = new javax.swing.JTable();

```

```

setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
    setTitle("Extensões da Máquina Virtual");
    setIconImage(new
javax.swing.ImageIcon(getClass().getResource("/Icones/PacoteIcôneGrande.png")
).getImage());
    setMinimumSize(new java.awt.Dimension(680, 478));

    addJarButton.setText("Adicionar pacote");
    addJarButton.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            addJarButtonActionPerformed(evt);
        }
    });

    tabelaInstrucoes.setModel(new javax.swing.table.DefaultTableModel(
        new Object [][] {

        },
        new String [] {
            "Instrução", "Opcode", "Ativada"
        }
    ) {
        Class[] types = new Class [] {
            Instrucao.class, java.lang.String.class,
java.lang.Boolean.class
        };

        public Class getColumnClass(int columnIndex) {
            return types [columnIndex];
        }

        public boolean isCellEditable(int row, int col) {
            if(col == 2) {
                return true;
            }
            return false;
        }
    });
    jScrollPane1.setViewportViewView(tabelaInstrucoes);

    okButton.setText("OK");
    okButton.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            okButtonActionPerformed(evt);
        }
    });

    jLabel1.setText("Repertório de Instruções incluídas no pacote:");

    tabelaPacotes.setModel(new javax.swing.table.DefaultTableModel(
        new Object [][] {

        },
        new String [] {
            "Caminho do arquivo (.JAR)"
        }
    ) {
        Class[] types = new Class [] {

```

```

        java.lang.String.class
    };
    boolean[] canEdit = new boolean [] {
        false
    };

    public Class getColumnClass(int columnIndex) {
        return types [columnIndex];
    }

    public boolean isCellEditable(int rowIndex, int columnIndex) {
        return canEdit [columnIndex];
    }
});
tabelaPacotes.addKeyListener(new java.awt.event.KeyAdapter() {
    public void keyPressed(java.awt.event.KeyEvent evt) {
        tabelaPacotesKeyPressed(evt);
    }
});
jScrollPane2.setViewportView(tabelaPacotes);

jLabel2.setText("Lista de Pacotes de Extensões:");

removePacoteButton.setLabel("Remover pacote");
removePacoteButton.addActionListener(new
java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        removePacoteButtonActionPerformed(evt);
    }
});

jLabel3.setText("As alterações terão efeito apenas no próximo
programa aberto.");

ativarTodasBtn.setText("Ativar todas");
ativarTodasBtn.addActionListener(new java.awt.event.ActionListener()
{
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        ativarTodasBtnActionPerformed(evt);
    }
});

jLabel4.setText("Tipos de dados incluídos no pacote:");

tabelaTipos.setModel(new javax.swing.table.DefaultTableModel(
    new Object [][] {

    },
    new String [] {
        "Tipo", "Classe"
    }
) {
    Class[] types = new Class [] {
        java.lang.String.class, java.lang.String.class
    };
    boolean[] canEdit = new boolean [] {
        false, false
    };
};

```

```

    public Class getColumnClass(int columnIndex) {
        return types [columnIndex];
    }

    public boolean isCellEditable(int rowIndex, int columnIndex) {
        return canEdit [columnIndex];
    }
});
jScrollPane3.setViewportView(tabelaTipos);

javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
getContentPane().setLayout(layout);
layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGroup(layout.createSequentialGroup()
        .add(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
            .addGroup(layout.createSequentialGroup()
                .add(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                    .addComponent(addJarButton)
                    .addGroup(layout.createSequentialGroup()
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addComponent(removerPacoteButton)
                        .addGroup(layout.createSequentialGroup()
                            .add(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING,
layout.createSequentialGroup()
                                .add(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                                    .addComponent(jLabel2)
                                    .addGap(0, 0, Short.MAX_VALUE))
                                .addComponent(jScrollPane2,
javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.PREFERRED_SIZE, 0, Short.MAX_VALUE))
                            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                            .add(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                                .addGroup(layout.createSequentialGroup()
                                    .add(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                                        .addComponent(jLabel1,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                                        .addComponent(ativarTodasBtn,
javax.swing.GroupLayout.PREFERRED_SIZE, 95,
javax.swing.GroupLayout.PREFERRED_SIZE))
                                    .addGroup(layout.createSequentialGroup()
                                        .add(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                                            .addGap(10, 10, 10)
                                            .addComponent(jLabel13)

                            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                            .addComponent(okButton,
javax.swing.GroupLayout.PREFERRED_SIZE, 95,
javax.swing.GroupLayout.PREFERRED_SIZE))
                            .addComponent(jScrollPane3,

```

```

javax.swing.GroupLayout.PREFERRED_SIZE, 0, Short.MAX_VALUE)
    .addComponent(jScrollPane1,
javax.swing.GroupLayout.PREFERRED_SIZE, 0, Short.MAX_VALUE)
    .addComponent(jLabel14,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
        .addContainerGap()
    );
    layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGroup(layout.createSequentialGroup()
        .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
    .addComponent(jLabel11)
    .addComponent(jLabel12)
    .addComponent(ativarTodasBtn,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addComponent(jScrollPane2,
javax.swing.GroupLayout.DEFAULT_SIZE, 393, Short.MAX_VALUE)
    .addGroup(layout.createSequentialGroup()
        .addComponent(jScrollPane1,
javax.swing.GroupLayout.PREFERRED_SIZE, 0, Short.MAX_VALUE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
    .addComponent(jLabel14)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
    .addComponent(jScrollPane3,
javax.swing.GroupLayout.PREFERRED_SIZE, 167,
javax.swing.GroupLayout.PREFERRED_SIZE)))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
    .addComponent(okButton)
    .addComponent(addJarButton)
    .addComponent(removerPacoteButton)
    .addComponent(jLabel13))
        .addContainerGap()
    );

    pack();
} // </editor-fold>

private void atualizaTabelaPacotes() {
    ArrayList<File> jars = InterfaceMVLIS.getInstance().getJars();
    DefaultTableModel model =
(DefaultTableModel)this.tabelaPacotes.getModel();
    model.setRowCount(0);

```

```

        for(int x = 0; x < jars.size(); x++) {
            try {
                model.insertRow(x, new
Object[] {jars.get(x).getCanonicalPath()});
            } catch(Exception e) {
                JOptionPane.showMessageDialog(this, e.getMessage(),
"Exception", JOptionPane.ERROR_MESSAGE);
            }
        }
    }

    private void atualizaTabelaTipos() {

        int selrow = this.tabelaPacotes.getSelectedRow();

        ArrayList<ItemPilha> tipos = InterfaceMVLISI.getInstance().getTipos();
        DefaultTableModel model =
(DefaultTableModel)this.tabelaTipos.getModel();
        model.setRowCount(0);

        if(selrow == -1) {
            return;
        }

        String filename = this.tabelaPacotes.getModel().getValueAt(selrow,
0).toString();

        int y = 0;
        for(int x = 0; x < tipos.size(); x++) {

            try {
                String tipojar = tipos.get(x).getJar().getCanonicalPath();
                if(!tipojar.equals(filename)) {
                    continue;
                }

                String classcanonicalname =
tipos.get(x).getClass().getCanonicalName();
                String classname =
classcanonicalname.substring(classcanonicalname.lastIndexOf(".") + 1);
                model.insertRow(y++, new Object[] {tipos.get(x).getTipo(),
classname});
            } catch(IOException e) {
                JOptionPane.showMessageDialog(this, e.getMessage(),
"IIOException", JOptionPane.ERROR_MESSAGE);
                continue;
            }
        }
    }

    private void atualizaTabelaInstrucoes() {

        int selrow = this.tabelaPacotes.getSelectedRow();

        ArrayList<Instrucao> instrucoes =
InterfaceMVLISI.getInstance().getIsa();
        DefaultTableModel model =
(DefaultTableModel)this.tabelaInstrucoes.getModel();

```

```

        model.setRowCount(0);

        if(selrow == -1) {
            return;
        }

        String filename = this.tabelaPacotes.getModel().getValueAt(selrow,
0).toString();

        int y = 0;
        for(int x = 0; x < instrucoes.size(); x++) {

            try {

                if(!instrucoes.get(x).getJar().getCanonicalPath().equals(filename)) {
                    continue;
                }
            } catch(IOException e) {
                JOptionPane.showMessageDialog(this, e.getMessage(),
"IOException", JOptionPane.ERROR_MESSAGE);
                continue;
            }

            model.insertRow(y++, new Object[]{instrucoes.get(x),
instrucoes.get(x).getOpcodeString(), instrucoes.get(x).isActive()});
        }

        // remove o pacote selecionado
        private void removerPacote() {
            // identifica o pacote selecionado
            // a linha deve conter o caminho completo do pacote
            int selrow = this.tabelaPacotes.getSelectedRow();
            if(selrow > -1) {
                String pacote = this.tabelaPacotes.getModel().getValueAt(selrow,
0).toString();

                // dialogo com o usuário
                if(JOptionPane.YES_OPTION == JOptionPane.showConfirmDialog(this,
"Tem certeza que deseja remover o pacote abaixo? \n\n"+pacote, "Confirmação",
JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE)) {

                    // obtém lista de jars
                    ArrayList<File> jars =
InterfaceMVLIS.getInstance().getJars();

                    // obtém lista de instruções e tipos
                    ArrayList<Instrucao> isa =
InterfaceMVLIS.getInstance().getIsa();
                    ArrayList<ItemPilha> tipos =
InterfaceMVLIS.getInstance().getTipos();

                    Iterator it = jars.iterator();

                    ArrayList<String> jarsfiles = new ArrayList();

                    // iteração na lista de jars
                    while(it.hasNext()) {
                        File jar = (File)it.next();

```



```

        try {
            if(jar.getCanonicalPath().equals(pacote)) {
                Iterator it_isa = isa.iterator();
                // remove todas as instruções da ISA que
                // a comparação é feita usando o caminho completo
                while(it_isa.hasNext()) {
                    Instrucao inst = (Instrucao)it_isa.next();

                    if(inst.getJar().getCanonicalPath().equals(jar.getCanonicalPath())) {
                        it_isa.remove();
                    }
                }

                Iterator it_tipos = tipos.iterator();
                while(it_tipos.hasNext()) {
                    ItemPilha tipo = (ItemPilha)it_tipos.next();

                    if(tipo.getJar().getCanonicalPath().equals(jar.getCanonicalPath())) {
                        it_tipos.remove();
                    }
                }

                it.remove();
            } else {
                jarsfiles.add(jar.getCanonicalPath());
            }
        } catch(IOException e) {
            JOptionPane.showMessageDialog(this, e.getMessage(),
                "IOException", JOptionPane.ERROR_MESSAGE);
        }

        String jarsfilesok = "";
        Object[] strArray = jarsfiles.toArray();
        for(int i=0;i<strArray.length;i++) {
            jarsfilesok += (i == strArray.length - 1) ? strArray[i] :
strArray[i] + "::::";
        }
        InterfaceMVLISI.getInstance().getPreferences().put("jars",
jarsfilesok);

        InterfaceMVLISI.getInstance().getPreferences().remove("jar_"+pacote.hashCode()
);

        try {
            InterfaceMVLISI.getInstance().getPreferences().flush();
        } catch(BackingStoreException e) {
            JOptionPane.showMessageDialog(this, e.getMessage(),
                "BackingStoreException", JOptionPane.ERROR_MESSAGE);
        }

        // remove a linha da tabela

        ((DefaultTableModel)this.tabelaPacotes.getModel()).removeRow(selrow);

        this.salvarInstrucoesAtivas();
    }

```

```

        } else {
            JOptionPane.showMessageDialog(this, "Nenhum pacote selecionado!",
"Aviso", JOptionPane.WARNING_MESSAGE);
        }
    }

    private void addJarButtonActionPerformed(java.awt.event.ActionEvent evt)
    {
        JFileChooser fc = new JFileChooser();
        int retorno = fc.showOpenDialog(this);

        if(retorno == JFileChooser.APPROVE_OPTION) {
            try {
                File jarfilef = fc.getSelectedFile();
                if(jarfilef.exists()) {
                    if(jarfilef.canRead()) {

if(!InterfaceMVLISI.getInstance().isJarLoaded(jarfilef)) {
InterfaceMVLISI.getInstance().getJars().add(jarfilef);
InterfaceMVLISI.getInstance().loadClassesFromJar(jarfilef);

                String jarsfiles =
InterfaceMVLISI.getInstance().getPreferences().get("jars", "");

                    if(!jarsfiles.equals("")) {
                        jarsfiles += "::-" +
jarfilef.getCanonicalPath();
                    } else {
                        jarsfiles = jarfilef.getCanonicalPath();
                    }

InterfaceMVLISI.getInstance().getPreferences().put("jars", jarsfiles);

                try {

InterfaceMVLISI.getInstance().getPreferences().flush();
                } catch(BackingStoreException e) {
                    JOptionPane.showMessageDialog(this,
e.getMessage(), "BackingStoreException", JOptionPane.ERROR_MESSAGE);
                }

                this.atualizaTabelaPacotes();
                this.atualizaTabelaInstrucoes();
                this.salvarInstrucoesAtivas();
            } else {
                JOptionPane.showMessageDialog(this, "O pacote
"+jarfilef.getCanonicalPath()+" já foi incluído!\n", "Erro",
JOptionPane.ERROR_MESSAGE);
            }
            } else {
                JOptionPane.showMessageDialog(this, "Não é possível
ler o arquivo: "+jarfilef.getCanonicalPath()+"\n", "Erro",
JOptionPane.ERROR_MESSAGE);
            }
        } else {
            JOptionPane.showMessageDialog(this, "Não foi possível

```

```

encontrar "+jarfilef.getCanonicalPath()+"\n", "Erro",
JOptionPane.ERROR_MESSAGE);
    }

    } catch (MalformedURLException e) {
        JOptionPane.showMessageDialog(this, e.getMessage(),
"MalformedURLException", JOptionPane.ERROR_MESSAGE);
    } catch (IOException e) {
        JOptionPane.showMessageDialog(this, e.getMessage(),
"IOException", JOptionPane.ERROR_MESSAGE);
    }
}

public void salvarInstrucoesAtivas() {
    for(int x = 0; x < InterfaceMVLISi.getInstance().getJars().size();
x++) {

        String key = "";
        try {
            key =
"jar_"+InterfaceMVLISi.getInstance().getJars().get(x).getCanonicalPath().hashCode();
        } catch (IOException e) {
            JOptionPane.showMessageDialog(this, e.getMessage(),
"IOException", JOptionPane.ERROR_MESSAGE);
            return;
        }

        String value = "";
        for(int y = 0; y < InterfaceMVLISi.getInstance().getIsa().size();
y++) {
            if(InterfaceMVLISi.getInstance().getJars().get(x) ==
InterfaceMVLISi.getInstance().getIsa().get(y).getJar()) {
                value +=
InterfaceMVLISi.getInstance().getIsa().get(y).getClass().getSimpleName();
            }
            if(InterfaceMVLISi.getInstance().getIsa().get(y).isActive()) {
                value += "=1|";
            } else {
                value += "=0|";
            }
        }
        InterfaceMVLISi.getInstance().getPreferences().put(key, value);
    }

    try {
        InterfaceMVLISi.getInstance().getPreferences().flush();
    } catch (BackingStoreException e) {
        JOptionPane.showMessageDialog(this, e.getMessage(),
"BackingStoreException", JOptionPane.ERROR_MESSAGE);
        return;
    }

    InterfaceMVLISi.getInstance().carregarExtensoesNaMV();
}
}

```

```

        private void okButtonActionPerformed(java.awt.event.ActionEvent evt) {
            salvarInstrucoesAtivas();
            this.dispose();
        }

        private void tabelaPacotesKeyPressed(java.awt.event.KeyEvent evt) {
            if(evt.getKeyCode() == java.awt.event.KeyEvent.VK_DELETE) {
                this.removerPacote();
            }
        }

        private void
removerPacoteButtonActionPerformed(java.awt.event.ActionEvent evt) {
            this.removerPacote();
        }

        private void ativarTodasBtnActionPerformed(java.awt.event.ActionEvent
evt) {
            for(int x = 0; x < this.tabelaInstrucoes.getModel().getRowCount();
x++) {
                if((Boolean)this.tabelaInstrucoes.getValueAt(x, 2) == false) {
                    this.tabelaInstrucoes.getModel().setValueAt(true, x, 2);
                }
            }
        }

        // Variables declaration - do not modify
        private javax.swing.JButton addJarButton;
        private javax.swing.JButton ativarTodasBtn;
        private javax.swing.JLabel jLabel1;
        private javax.swing.JLabel jLabel2;
        private javax.swing.JLabel jLabel3;
        private javax.swing.JLabel jLabel4;
        private javax.swing.JScrollPane jScrollPane1;
        private javax.swing.JScrollPane jScrollPane2;
        private javax.swing.JScrollPane jScrollPane3;
        private javax.swing.JButton okButton;
        private javax.swing.JButton removerPacoteButton;
        private javax.swing.JTable tabelaInstrucoes;
        private javax.swing.JTable tabelaPacotes;
        private javax.swing.JTable tabelaTipos;
        // End of variables declaration
    }

```

## InserirInstrucao.java

```

package UFSC.InterfaceMVLSI;

import UFSC.MaquinaVirtual.Instrucao;
import java.util.ArrayList;
import javax.swing.JOptionPane;
import javax.swing.event.*;
import java.nio.ByteBuffer;

/**
 *
 * @author LucasMS
 */
public class InserirInstrucao extends javax.swing.JFrame {

```

```

private int insert_index;
/**
 * Creates new form InserirInstrucao
 */
public InserirInstrucao(int insert_index) {
    initComponents();

    this.insert_index = insert_index;

    ArrayList<Instrucao> isa = InterfaceMVLIS.getInstance().getIsa();

    this.instComboBox.addItem("");
    for(Instrucao inst : isa) {
        if(inst.isActive())
            this.instComboBox.addItem(inst);
    }

    DocumentListener iid1 = new DocumentListener() {
        public void changedUpdate(DocumentEvent e) {
            if(par1TextField.getText().length() > 0 ||
par2TextField.getText().length() > 0) {
                par64Textfield.setEnabled(false);
                par64Textfield.setText("");
            } else {
                par64Textfield.setEnabled(true);
                par64Textfield.setText("");
            }
        }
        public void removeUpdate(DocumentEvent e) {
            this.changedUpdate(e);
        }
        public void insertUpdate(DocumentEvent e) {
            this.changedUpdate(e);
        }
    };

    this.par1TextField.getDocument().addDocumentListener(iid1);
    this.par2TextField.getDocument().addDocumentListener(iid1);
    this.par64Textfield.getDocument().addDocumentListener(new
DocumentListener() {
        public void changedUpdate(DocumentEvent e) {
            if(par64Textfield.getText().length() > 0) {
                par1TextField.setEnabled(false);
                par2TextField.setEnabled(false);
            } else {
                par1TextField.setEnabled(true);
                par2TextField.setEnabled(true);
            }
            par1TextField.setText("");
            par2TextField.setText("");
        }
        public void removeUpdate(DocumentEvent e) {
            this.changedUpdate(e);
        }
        public void insertUpdate(DocumentEvent e) {
            this.changedUpdate(e);
        }
    });
}

```

```

    }

    /**
     * This method is called from within the constructor to initialize the
    form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        instComboBox = new javax.swing.JComboBox();
        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        par1TextField = new javax.swing.JTextField();
        jLabel3 = new javax.swing.JLabel();
        par2TextField = new javax.swing.JTextField();
        okBtn = new javax.swing.JButton();
        cancelBtn = new javax.swing.JButton();
        jLabel4 = new javax.swing.JLabel();
        opcodeTextfield = new javax.swing.JTextField();
        jLabel5 = new javax.swing.JLabel();
        par64Textfield = new javax.swing.JTextField();

        setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
        setTitle("Inserir");
        setAlwaysOnTop(true);
        setResizable(false);
        setType(java.awt.Window.Type.POPUP);

        instComboBox.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                instComboBoxActionPerformed(evt);
            }
        });

        jLabel1.setText("Instrução:");

        jLabel2.setText("Parâmetro #1");

        par1TextField.setEnabled(false);

        jLabel3.setText("Parâmetro #2");

        par2TextField.setEnabled(false);

        okBtn.setText("OK");
        okBtn.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                okBtnActionPerformed(evt);
            }
        });

        cancelBtn.setText("Cancelar");
        cancelBtn.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                cancelBtnActionPerformed(evt);
            }
        });
    }

```

```

    });

    jLabel14.setText("Opcode");
    opcodeTextfield.setEnabled(false);

    jLabel15.setText("OU Parâmetro de 64 bits (tipo Double)");
    par64Textfield.setEnabled(false);

    javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
getContentPane().setLayout(layout);
layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
        .addContainerGap()

        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
            .addComponent(opcodeTextfield)
            .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
layout.createSequentialGroup()
                .addComponent(cancelBtn,
javax.swing.GroupLayout.PREFERRED_SIZE, 87,
javax.swing.GroupLayout.PREFERRED_SIZE)

                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                    .addComponent(okBtn,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
                .addComponent(par64Textfield,
javax.swing.GroupLayout.Alignment.LEADING)
                .addComponent(instComboBox,
javax.swing.GroupLayout.Alignment.LEADING, 0,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addComponent(par1Textfield,
javax.swing.GroupLayout.Alignment.LEADING)
                .addComponent(jLabel15,
javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                .addComponent(par2Textfield,
javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
layout.createSequentialGroup()

                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(jLabel11)
                    .addComponent(jLabel12)
                    .addComponent(jLabel13)
                    .addComponent(jLabel14))
                .addGap(0, 0, Short.MAX_VALUE)))
            .addContainerGap())
        );

```

```

        layout.setVerticalGroup(
layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
        .addContainerGap()
        .addComponent(jLabel11)

        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(instComboBox,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(jLabel14)

        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(opcodeTextField,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(jLabel12)

        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(par1TextField,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(jLabel13)

        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(par2TextField,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(jLabel15)

        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(par64TextField,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)

        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
        .addComponent(cancelBtn)
        .addComponent(okBtn))
        .addContainerGap()
    );

    pack();
} // </editor-fold>

private void cancelBtnActionPerformed(java.awt.event.ActionEvent evt) {

```



```

        this.dispose();
    }

    private void okBtnActionPerformed(java.awt.event.ActionEvent evt) {
        if(this.instComboBox.getSelectedIndex() > 0) {
            Instrucao inst = (Instrucao)this.instComboBox.getSelectedItem();

            String par1 = this.par1TextField.getText();
            String par2 = this.par2TextField.getText();
            String par64 = this.par64Textfield.getText();

            int par_1 = 0;
            int par_2 = 0;
            double par_64 = 0.0;

            if(!par64.equals("")) {
                try {
                    par_64 =
Double.parseDouble(this.par64Textfield.getText());
                    ByteBuffer buf =
ByteBuffer.allocate(8).putDouble(par_64);
                    buf.rewind();
                    par_1 = buf.getInt();
                    par_2 = buf.getInt();
                } catch(NumberFormatException e) {
                    JOptionPane.showMessageDialog(this, "O número digitado
como parâmetro de 64 bits não é válido!", "Erro!",
JOptionPane.ERROR_MESSAGE);
                    return;
                }
            } else {
                try {
                    par_1 = Integer.parseInt(this.par1TextField.getText());
                    par_2 = Integer.parseInt(this.par2TextField.getText());

                    ByteBuffer par_64_buffer =
ByteBuffer.allocate(8).putInt(par_1).putInt(par_2);
                    par_64_buffer.rewind();
                    par_64 = par_64_buffer.getDouble();
                } catch(NumberFormatException e) {
                    JOptionPane.showMessageDialog(this, "Os números digitados
no 1º ou 2º parâmetros não são válidos!", "Erro!",
JOptionPane.ERROR_MESSAGE);
                    return;
                }
            }

            InterfaceMVLSE.getInstance().getMaquinaVirtual().insertInstruction(this.inser
t_index, inst.getOpcode(), par_1, par_2, par_64);
            this.dispose();
        } else {
            JOptionPane.showMessageDialog(this, "Selecione uma instrução no
menu!", "Erro!", JOptionPane.ERROR_MESSAGE);
        }
    }

    private void instComboBoxActionPerformed(java.awt.event.ActionEvent evt)
{

```

```

        if(this.instComboBox.getSelectedIndex() > 0) {
            Instrucao inst = (Instrucao)this.instComboBox.getSelectedItem();
            this.opcodeTextfield.setText(inst.getOpcodeString());
            this.par1Textfield.setEnabled(true);
            this.par2Textfield.setEnabled(true);
            this.par64Textfield.setEnabled(true);
        } else {
            this.opcodeTextfield.setText("");
        }
        this.par1Textfield.setText("");
        this.par2Textfield.setText("");
    }

    // Variables declaration - do not modify
    private javax.swing.JButton cancelBtn;
    private javax.swing.JComboBox instComboBox;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JButton okBtn;
    private javax.swing.JTextField opcodeTextfield;
    private javax.swing.JTextField par1Textfield;
    private javax.swing.JTextField par2Textfield;
    private javax.swing.JTextField par64Textfield;
    // End of variables declaration
}

```

## InterfaceMVLSI.java

```

package UFSC.InterfaceMVLSI;

import UFSC.MaquinaVirtual.*;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.net.URL;
import java.net.URLClassLoader;
import java.util.ArrayList;
import java.util.Enumeration;
import java.util.LinkedList;
import java.util.jar.JarEntry;
import java.util.jar.JarFile;
import java.util.prefs.BackingStoreException;
import java.util.prefs.Preferences;
import javax.swing.*;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
import javax.swing.table.AbstractTableModel;
import javax.swing.table.DefaultTableModel;

/**
 *
 * @author LucasMS
 */
public class InterfaceMVLSI extends javax.swing.JFrame implements

```

```

MaquinaVirtualInterface {

    private static InterfaceMVLSI instance;

    protected static final String window_title = "Máquina Virtual LSI";

    private final MaquinaVirtual mv;

    private File arquivo = null;

    private String entrada = ""; // entrada de dados pelo usuário

    // lista com pacotes .jar contendo instruções da máquina virtual
    private ArrayList<File> jars = new ArrayList();

    private GerenciadorExtensoes inst_manager_wnd = null;
    private InserirInstrucao insert_inst_wnd = null;

    // lista com todas as classes de instruções
    // MESMO instruções repetidas
    private ArrayList<Instrucao> isa = new ArrayList();

    // lista com todos os tipos de dados dos pacotes
    // pode conter tipos com nomes repetidos
    private ArrayList<ItemPilha> tipos = new ArrayList();

    // lista com os tipos literais de dados
    private ArrayList<Literal> tiposliterais = new ArrayList();

    private final Object read_lock = new Object();

    private Preferences prefs =
Preferences.userNodeForPackage(InterfaceMVLSI.class);
    private ArquivosRecentesList arquivos_recentes;

    private MVLSI_ClassLoader classloader_extensoes = new
MVLSI_ClassLoader();

    /**
     * Creates new form InterfaceMVLSI
     */
    public InterfaceMVLSI() {

        InterfaceMVLSI.instance = this;

        initComponents();
        this.arquivos_recentes = new ArquivosRecentesList();

        // desativa o reordenamento de colunas nas tabelas
        this.tabelaInstrucoes.getTableHeader().setReorderingAllowed(false);
        this.tabelaPilha.getTableHeader().setReorderingAllowed(false);
        this.tabelaLiterais.getTableHeader().setReorderingAllowed(false);

        // fixa a largura das colunas de indices
        this.tabelaInstrucoes.getColumnModel().getColumn(0).setMaxWidth(50);
        this.tabelaPilha.getColumnModel().getColumn(0).setMaxWidth(50);
        this.tabelaLiterais.getColumnModel().getColumn(0).setMaxWidth(50);
        this.tabelaLiterais.getColumnModel().getColumn(1).setMaxWidth(60);
    }
}

```

```

// nenhum programa carregado ainda
this.entradaComando.setEnabled(false);

this.tabelaInstrucoes.putClientProperty("terminateEditOnFocusLost",
Boolean.TRUE);
this.tabelaInstrucoes.editingStopped(new
javax.swing.event.ChangeEvent(this.tabelaInstrucoes));

// carrega pacotes de instrucoes
try {
    String jarsfiles = this.prefs.get("jars", "");

    if(!jarsfiles.equals("")) {
        String[] jarfilesa = jarsfiles.split(":::");

        String reading_errors = "";

        LinkedList<String> jarfilesok = new LinkedList();

        for(String jarfile : jarfilesa) {
            if(jarfile.length() == 0) continue;

            File jarfilef = new File(jarfile);

            if(jarfilef.exists()) {
                if(jarfilef.canRead()) {
                    if(!this.isJarLoaded(jarfilef)) {
                        this.jars.add(jarfilef);
                        jarfilesok.add(jarfile);
                    } else {
                        reading_errors += "- O pacote "+jarfile+" já
foi incluído\n";
                    }
                } else {
                    reading_errors += "- Não é possível ler o
arquivo: "+jarfile+"\n";
                }
            } else {
                reading_errors += "- Não foi possível encontrar:
"+jarfile+"\n";
            }
        }

        if(!reading_errors.equals("")) {
            JOptionPane.showMessageDialog(this, "Erros ocorreram ao
abrir os pacotes de instruções:\n\n"+reading_errors, "Erro!",
JOptionPane.ERROR_MESSAGE);
            String jarfilesok2 = "";
            Object[] strArray = jarfilesok.toArray();
            for(int i=0;i<strArray.length;i++) {
                jarfilesok2 += (i == strArray.length - 1) ?
strArray[i] : strArray[i] + "::::";
            }
            this.prefs.put("jars", jarfilesok2);

            try {
                this.prefs.flush();
            } catch(BackingStoreException e) {

```

```

        JOptionPane.showMessageDialog(this, e.getMessage(),
"BackingStoreException", JOptionPane.ERROR_MESSAGE);
    }
}

if(this.jars.isEmpty()) {
    JOptionPane.showMessageDialog(this, "Nenhum pacote de
extensões carregado!\nVocê precisa adicionar um pacote utilizando o menu
Máquina Virtual -> Pacotes de Extensões.", "Aviso!",
JOptionPane.WARNING_MESSAGE);
} else for(File jar : this.jars) {
    this.loadClassesFromJar(jar);
}

} catch (IOException ex) {
    JOptionPane.showMessageDialog(this, ex.getMessage(),
"IOException", JOptionPane.ERROR_MESSAGE);
}

// auto-scroll da tabela de instruções durante a execução, quando
esta possuir barra de rolagem

this.tabelaInstrucoes.getSelectionModel().addListSelectionListener(new
ListSelectionListener() {
    Override
    public void valueChanged(ListSelectionEvent event) {

tabelaInstrucoes.scrollRectToVisible(tabelaInstrucoes.getCellRect(tabelaInstr
ucoes.getSelectedRow(), 0, true));
        InterfaceMVLSI.this.repaint();
    }
});

// cria a thread da máquina virtual
this.mv = new MaquinaVirtual(this);
this.mv.start();

// configura o seletor da velocidade para o minimo de 0 a 100.
default: 75
this.ajusteVelocidade.setValue( this.prefs.getInt("velocidade", 75));

this.mv.addPilhaObserver(new PilhaExecucaoInterfaceObserver());
this.mv.addLiteraisObserver(new LiteraisInterfaceObserver());
this.mv.addInstrucoesObserver(new
TabelaInstrucoesInterfaceObserver());
this.mv.addRegistradoresBaseObserver(new
RegistradoresBaseInterfaceObserver());
//this.mv.resetar();

this.carregarExtensoesNaMV();

// thread para atualizar os valores dos registradores na tela
new Thread(new Runnable() {
    Override
    public void run() {
        while(true) {
            try {
                if(mv != null) {

```

```

        pcTextField.setText((mv.PC)+"");
        topoTextField.setText(mv.pilha.topo+"");
numExecsTextfield.setText(mv.getNumeroInstrucoesExecutadas()+"");
        } else {
            pcTextField.setText("");
        }
        Thread.sleep(100);
    } catch (InterruptedException e) {
        break;
    }
    }
    }.start();
}

public static InterfaceMVLSI getInstance() {
    return InterfaceMVLSI.instance;
}

/**
 * This method is called from within the constructor to initialize the
form.
 * WARNING: Do NOT modify this code. The content of this method is always
 * regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {

    popupInstrucoes = new javax.swing.JPopupMenu();
    inserirInstrucaoMenuItem = new javax.swing.JMenuItem();
    inserirInstrucaoAcimaMenuItem = new javax.swing.JMenuItem();
    inserirInstrucaoAbaixoMenuItem = new javax.swing.JMenuItem();
    removerInstrucaoMenuItem = new javax.swing.JMenuItem();
    jScrollPane1 = new javax.swing.JScrollPane();
    console = new javax.swing.JTextArea();
    jLabel1 = new javax.swing.JLabel();
    jLabel2 = new javax.swing.JLabel();
    jScrollPane2 = new javax.swing.JScrollPane();
    tabelaInstrucoes = new javax.swing.JTable();
    jLabel3 = new javax.swing.JLabel();
    jScrollPane3 = new javax.swing.JScrollPane();
    tabelaPilha = new javax.swing.JTable();
    entradaComando = new javax.swing.JTextField();
    execButton = new javax.swing.JButton();
    ajusteVelocidade = new javax.swing.JSlider();
    jLabel4 = new javax.swing.JLabel();
    jLabel5 = new javax.swing.JLabel();
    pcTextField = new javax.swing.JTextField();
    jScrollPane4 = new javax.swing.JScrollPane();
    registradoresBase = new javax.swing.JTable();
    topoTextField = new javax.swing.JTextField();
    jLabel6 = new javax.swing.JLabel();
    jLabel7 = new javax.swing.JLabel();
    numExecsTextfield = new javax.swing.JTextField();
    jLabel8 = new javax.swing.JLabel();
    stepButton = new javax.swing.JButton();
    jScrollPane5 = new javax.swing.JScrollPane();

```

```

tabelaLiterais = new javax.swing.JTable();
jLabel19 = new javax.swing.JLabel();
jMenuBar1 = new javax.swing.JMenuBar();
jMenu1 = new javax.swing.JMenu();
abrirMenuItem = new javax.swing.JMenuItem();
fecharMenuItem = new javax.swing.JMenuItem();
jSeparator1 = new javax.swing.JPopupMenu.Separator();
abertosRecentementeMenu = new javax.swing.JMenu();
nenhumMenuItem = new javax.swing.JMenuItem();
jSeparator3 = new javax.swing.JPopupMenu.Separator();
sairMenuItem = new javax.swing.JMenuItem();
jMenu3 = new javax.swing.JMenu();
instrucoesMenuItem = new javax.swing.JMenuItem();
jMenu2 = new javax.swing.JMenu();
sobreMenuItem = new javax.swing.JMenuItem();

inserirInstrucaoMenuItem.setText("Inserir instrução ...");

inserirInstrucaoAcimaMenuItem.setText("Acima");
inserirInstrucaoAcimaMenuItem.addActionListener(new
java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        inserirInstrucaoAcimaMenuItemActionPerformed(evt);
    }
});
inserirInstrucaoMenuItem.add(inserirInstrucaoAcimaMenuItem);

inserirInstrucaoAbaixoMenuItem.setText("Abaixo");
inserirInstrucaoAbaixoMenuItem.addActionListener(new
java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        inserirInstrucaoAbaixoMenuItemActionPerformed(evt);
    }
});
inserirInstrucaoMenuItem.add(inserirInstrucaoAbaixoMenuItem);

popupInstrucoes.add(inserirInstrucaoMenuItem);

removerInstrucaoMenuItem.setText("Remover instrução");
removerInstrucaoMenuItem.addActionListener(new
java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        removerInstrucaoMenuItemActionPerformed(evt);
    }
});
popupInstrucoes.add(removerInstrucaoMenuItem);

setDefaultCloseOperation(javax.swing.WindowConstants.DO_NOTHING_ON_CLOSE);
setTitle("Máquina Virtual LSI");
setIconImage(new
javax.swing.ImageIcon(getClass().getResource("/Icones/IconeMV.png")).getImage
());
addWindowListener(new java.awt.event.WindowAdapter() {
    public void windowClosing(java.awt.event.WindowEvent evt) {
        formWindowClosing(evt);
    }
});

```

```

console.setEditable(false);
console.setColumns(20);
console.setRows(5);
jScrollPane1.setViewportView(console);

jLabel1.setText("Console:");

jLabel2.setText("Programa:");

tabelaInstrucoes.setModel(new javax.swing.table.DefaultTableModel(
    new Object [][] {

    },
    new String [] {
        "#", "Instrução", "Opcode", "Param. #1", "Param. #2"
    }
) {
    Class[] types = new Class [] {
        java.lang.Integer.class, java.lang.String.class,
java.lang.String.class, java.lang.String.class, java.lang.String.class
    };
    boolean[] canEdit = new boolean [] {
        false, false, true, true, true
    };

    public Class getColumnClass(int columnIndex) {
        return types [columnIndex];
    }

    public boolean isCellEditable(int rowIndex, int columnIndex) {
        return canEdit [columnIndex];
    }

    public void setValueAt(Object value, int row, int col) {
        if(InterfaceMVLSI.this.mv.editInstruction(row, col-1,
value.toString())) {
            super.setValueAt(value, row, col);
        }
    }
});

tabelaInstrucoes.setSelectionMode(javax.swing.ListSelectionModel.SINGLE_SELECTION);
tabelaInstrucoes.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseReleased(java.awt.event.MouseEvent evt) {
        tabelaInstrucoesMouseReleased(evt);
    }
});
tabelaInstrucoes.addKeyListener(new java.awt.event.KeyAdapter() {
    public void keyPressed(java.awt.event.KeyEvent evt) {
        tabelaInstrucoesKeyPressed(evt);
    }
});
jScrollPane2.setViewportView(tabelaInstrucoes);

jLabel3.setText("Pilha:");

tabelaPilha.setModel(new DefaultTableModel(
    new Object [][] {

```



```

    },
    new String [] {
        "Indice", "Tipo", "Valor"
    }
) {
    Class[] types = new Class [] {
        java.lang.Integer.class, String.class, java.lang.String.class
    };
    boolean[] canEdit = new boolean [] {
        false, false, false
    };

    public Class getColumnClass(int columnIndex) {
        return types [columnIndex];
    }

    public boolean isCellEditable(int rowIndex, int columnIndex) {
        return canEdit [columnIndex];
    }

    public Object getValueAt(int row, int column) {
        Object o = null;
        try {
            o = super.getValueAt(row, column);
        } catch(ArrayIndexOutOfBoundsException ex) {
        }
        return o;
    }
}

});

tabelaPilha.setSelectionMode(javax.swing.ListSelectionModel.SINGLE_SELECTION)
;

jScrollPane3.setViewportViewView(tabelaPilha);

entradaComando.addKeyListener(new java.awt.event.KeyAdapter() {
    public void keyPressed(java.awt.event.KeyEvent evt) {
        entradaComandoKeyPressed(evt);
    }
});

execButton.setText("Executar");
execButton.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        execButtonActionPerformed(evt);
    }
});

ajusteVelocidade.setMajorTickSpacing(10);
ajusteVelocidade.setMinorTickSpacing(5);
ajusteVelocidade.setPaintTicks(true);
ajusteVelocidade.setSnapToTicks(true);
ajusteVelocidade.setValueIsAdjusting(true);
ajusteVelocidade.addChangeListener(new
javax.swing.event.ChangeListener() {
    public void stateChanged(javax.swing.event.ChangeEvent evt) {
        ajusteVelocidadeStateChanged(evt);
    }
});

```

```

    }
});

jLabel4.setText("Velocidade:");
jLabel5.setText("PC:");
pcTextField.setEnabled(false);

registradoresBase.setModel(new DefaultTableModel(
    new Object [][] {

    },
    new String [] {
        "Nível", "Valor"
    }
) {
    Class[] types = new Class [] {
        java.lang.Integer.class, java.lang.Integer.class
    };
    boolean[] canEdit = new boolean [] {
        false, false
    };

    public Class getColumnClass(int columnIndex) {
        return types [columnIndex];
    }

    public boolean isCellEditable(int rowIndex, int columnIndex) {
        return canEdit [columnIndex];
    }

    public Object getValueAt(int row, int column) {
        Object o = null;
        try {
            o = super.getValueAt(row, column);
        } catch(IndexOutOfBoundsException ex) {
        }
        return o;
    }
});

registradoresBase.setSelectionMode(javax.swing.ListSelectionModel.SINGLE_SELECTION);
jScrollPane4.setViewportViewView(registradoresBase);

topoTextField.setEnabled(false);

jLabel6.setText("Topo:");
jLabel7.setText("Número de instruções executadas:");
numExecsTextfield.setEnabled(false);
jLabel8.setText("Pilha de registros de ativação:");
stepButton.setText(">>");

```

```

        stepButton.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                stepButtonActionPerformed(evt);
            }
        });

        tabelaLiterais.setModel(new LiteraisTableModel());

tabelaLiterais.setSelectionMode(javax.swing.ListSelectionModel.SINGLE_SELECTION);
        jScrollPane5.setViewportViewView(tabelaLiterais);

        jLabel19.setText("Tabela de Literais:");

        jMenuItem1.setText("Arquivo");

        abrirMenuItem.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/Icones/AbrirArquivoPequeno.png
"))); // NOI18N
        abrirMenuItem.setLabel("Abrir ...");
        abrirMenuItem.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                abrirMenuItemActionPerformed(evt);
            }
        });
        jMenuItem1.add(abrirMenuItem);

        fecharMenuItem.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/Icones/IconeFechar.png"))); //
NOI18N
        fecharMenuItem.setText("Fechar arquivo");
        fecharMenuItem.setEnabled(false);
        fecharMenuItem.addActionListener(new java.awt.event.ActionListener()
{
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                fecharMenuItemActionPerformed(evt);
            }
        });
        jMenuItem1.add(fecharMenuItem);
        jMenuItem1.add(jSeparator1);

        abertosRecentementeMenu.setText("Aberto recentemente");

        nenhumMenuItem.setText("nenhum");
        nenhumMenuItem.setEnabled(false);
        abertosRecentementeMenu.add(nenhumMenuItem);

        jMenuItem1.add(abertosRecentementeMenu);
        jMenuItem1.add(jSeparator3);

        sairMenuItem.setText("Sair");
        sairMenuItem.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                sairMenuItemActionPerformed(evt);
            }
        });
        jMenuItem1.add(sairMenuItem);

        jMenuItemBar1.add(jMenuItem1);

```

```

jMenu3.setText("Máquina Virtual");

instrucoesMenuItem.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/Icones/PacoteIcône.png"))); //
NOI18N
instrucoesMenuItem.setText("Pacotes de Extensões");
instrucoesMenuItem.addActionListener(new
java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        instrucoesMenuItemActionPerformed(evt);
    }
});
jMenu3.add(instrucoesMenuItem);

jMenuBar1.add(jMenu3);

jMenu2.setText("Ajuda");

sobreMenuItem.setText("Sobre");
sobreMenuItem.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        sobreMenuItemActionPerformed(evt);
    }
});
jMenu2.add(sobreMenuItem);

jMenuBar1.add(jMenu2);

setJMenuBar(jMenuBar1);

javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
getContentPane().setLayout(layout);
layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGroup(layout.createSequentialGroup()
        .add(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .add(layout.createSequentialGroup()
                .add(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .add(jLabel1)
                    .add(jScrollPane1)
                    .add(entradaComando))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .add(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .add(jLabel2)
                    .add(jScrollPane2,
javax.swing.GroupLayout.PREFERRED_SIZE, 307,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

```

```

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
G)
        .addGroup(layout.createSequentialGroup())

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
G)
        .addComponent(jScrollPane3,
javax.swing.GroupLayout.PREFERRED_SIZE, 0, Short.MAX_VALUE)
        .addGroup(layout.createSequentialGroup()
        .addComponent(jLabel13)
        .addGap(0, 0, Short.MAX_VALUE))
        .addComponent(jScrollPane5,
javax.swing.GroupLayout.PREFERRED_SIZE, 0, Short.MAX_VALUE))
        .addGap(10, 10, 10))
        .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING, false)
        .addComponent(ajusteVelocidade,
javax.swing.GroupLayout.PREFERRED_SIZE, 0, Short.MAX_VALUE)

.addGroup(javax.swing.GroupLayout.Alignment.LEADING,
layout.createSequentialGroup()
        .addComponent(jLabel16)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(topoTextField))
        .addGroup(layout.createSequentialGroup()
        .addComponent(jLabel15)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(pcTextField))
        .addComponent(jLabel17,
javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
        .addComponent(numExecsTextfield,
javax.swing.GroupLayout.Alignment.LEADING)
        .addComponent(jLabel14,
javax.swing.GroupLayout.Alignment.LEADING))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
G)

.addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
        .addComponent(execButton,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(stepButton))
        .addComponent(jScrollPane4,
javax.swing.GroupLayout.PREFERRED_SIZE, 0, Short.MAX_VALUE)
        .addGroup(layout.createSequentialGroup()
        .addComponent(jLabel18)
        .addGap(0, 31, Short.MAX_VALUE)))

```

```

        .addContainerGap()
        .addGroup(layout.createSequentialGroup()
            .addComponent(jLabel19)

.addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))))
    );
    layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
    .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
G)
    .addGroup(layout.createSequentialGroup()
        .addComponent(jLabel11)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
    .addComponent(jScrollPane1)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
    .addComponent(entradaComando,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
    .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
NE)
    .addComponent(jLabel12)
    .addComponent(jLabel13))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
G)
    .addComponent(jScrollPane2,
javax.swing.GroupLayout.DEFAULT_SIZE, 530, Short.MAX_VALUE)
    .addGroup(layout.createSequentialGroup()
        .addComponent(jScrollPane3,
javax.swing.GroupLayout.PREFERRED_SIZE, 0, Short.MAX_VALUE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
    .addComponent(jLabel19)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
    .addComponent(jScrollPane5,
javax.swing.GroupLayout.PREFERRED_SIZE, 117,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
NE)
    .addComponent(jLabel15)
    .addComponent(pcTextField,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
    .addComponent(jLabel18,

```

```

javax.swing.GroupLayout.PREFERRED_SIZE, 14,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)

.addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
.addComponent(jScrollPane4,
javax.swing.GroupLayout.PREFERRED_SIZE, 94,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
.addComponent(execButton)
.addComponent(stepButton)))

.addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
.addComponent(jLabel6)
.addComponent(topoTextField,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
.addComponent(jLabel17)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
.addComponent(numExecsTextField,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
.addComponent(jLabel14)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
.addComponent(ajusteVelocidade,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))))))
.addComponent()
);

pack();
} // </editor-fold>

public final JTable getTabelaPilha() {return tabelaPilha;}
public final JTable getTabelaLiterais() { return tabelaLiterais;}
public final JTable getTabelaInstrucoes() {return tabelaInstrucoes;}
public final JTable getTabelaRegistadoresBase() { return
registadoresBase;}
public final MaquinaVirtual getMaquinaVirtual() {return this.mv; }

```

```

public final String getEntrada() {return this.entrada;}
public final void setEntrada(String s) {this.entrada = s;}
public final JTextField getEntradaComando() {return entradaComando;}
public final JTextArea getConsole() {return console;}
public final JButton getExecButton() {return execButton;}
public final ArrayList<File> getJars() {return this.jars;}
public final ArrayList<Instrucao> getIsa() { return this.isa; }
public final ArrayList<ItemPilha> getTipos() { return this.tipos; }
public final ArrayList<Literal> getTiposLiterais() { return
this.tiposliterais; }
public final Preferences getPreferences() { return this.prefs; }

public final void carregarExtensoesNaMV() {
    this.mv.clearInstructionSet();
    this.mv.clearTiposLiterais();

    for(Instrucao inst : this.isa) {
        if(inst.isActive()) {
            this.mv.addInstructionToISA(inst);
        }
    }

    for(Literal tipolit : this.tiposliterais) {
        this.mv.addTipoLiteral(tipolit);
    }
}

public final boolean isJarLoaded(File jar) throws IOException {
    for(File f : this.jars) {
        if(f.getCanonicalPath().equals(jar.getCanonicalPath())) {
            return true;
        }
    }
    return false;
}

public final void loadClassesFromJar(File jar) throws IOException {
    JarFile jf = new JarFile(jar);

    try {
        URL url = jar.toURL();

        this.classloader_extensoes.addUrl(url);

        String[] inst_ativadas =
this.prefs.get("jar_"+jar.getCanonicalPath().hashCode(), "").split("\\|");

        // loop em todas as entradas do jar
        Enumeration allEntries = jf.entries();
        while (allEntries.hasMoreElements()) {
            JarEntry entry = (JarEntry) allEntries.nextElement();
            String name = entry.getName();

            // as classes devem estar em um pacote que comece com
ufsc.maquinavirtual.extensoes

if(name.toLowerCase().startsWith("ufsc/maquinavirtual/extensoes") &&
name.endsWith(".class")) {
            name = name.replaceAll("/", ".").replaceAll(".class", "");

```



```

        try {
            // carrega a classe
            Class cls =
this.classloader_extensoes.loadClass(name);

            Class superclass = cls.getSuperclass();

            // verifica se a classe é uma extensão da classe
Instrucao
            if(superclass == Instrucao.class) {
                boolean ativada = false;
                for(String inst_ativada : inst_ativadas) {
                    if(inst_ativada.startsWith(cls.getSimpleName()+"=")) {
                        if(inst_ativada.endsWith("1")) {
                            ativada = true;
                        }
                        break;
                    }
                }
                Instrucao inst = (Instrucao)cls.newInstance();
                inst.setJar(jar);
                inst.setActive(ativada);
                this.isa.add(inst);
            } else if(superclass == ItemPilha.class) {
                // se não é instrução, então pode ser um tipo de
                ItemPilha tipo = (ItemPilha)cls.newInstance();
                tipo.setJar(jar);
                this.tipos.add(tipo);
            } else if(superclass == Literal.class) {
                // caso seja algum tipo de from_source_code
                Literal tipoliteral = (Literal)cls.newInstance();
                tipoliteral.setJar(jar);
                this.tiposliterais.add(tipoliteral);
            }
        } catch(ClassNotFoundException e) {
            JOptionPane.showMessageDialog(this, e.getMessage(),
"ClassNotFoundException", JOptionPane.ERROR_MESSAGE);
        } catch(InstantiationException e) {
            JOptionPane.showMessageDialog(this, e.getMessage(),
e.getClass().getSimpleName(), JOptionPane.ERROR_MESSAGE);
        } catch(IllegalAccessException e) {
            JOptionPane.showMessageDialog(this, e.getMessage(),
e.getClass().getSimpleName(), JOptionPane.ERROR_MESSAGE);
        }
    }
} catch(IOException e) {
    jf.close();
    throw e;
}
}

private void sobreMenuItemActionPerformed(java.awt.event.ActionEvent evt)
{
    // TODO add your handling code here:
}

```

```

    private void abrirMenuItemActionPerformed(java.awt.event.ActionEvent evt)
    {
        JFileChooser fc = new JFileChooser() {
            Override
            protected JDialog createDialog( java.awt.Component parent )
throws java.awt.HeadlessException {
                JDialog dialog = super.createDialog(parent);
                dialog.setIconImage(new
javax.swing.ImageIcon(getClass().getResource("/Icones/AbrirArquivoPequeno.png
")).getImage());
                return dialog;
            }
        };

        File lastOpened = this.arquivos_recentes.lista.peekFirst();
        if(lastOpened != null) {
            fc.setCurrentDirectory(lastOpened);
        }

        int retorno = fc.showOpenDialog(this);

        if(retorno == JFileChooser.APPROVE_OPTION) {
            this.abrirArquivo(fc.getSelectedFile());
        }
    }

    private void execButtonActionPerformed(java.awt.event.ActionEvent evt) {

        if(this.arquivo == null) {
            JOptionPane.showMessageDialog(this, "Nenhum programa carregado!",
"Erro", JOptionPane.ERROR_MESSAGE);
            return;
        }

        // se a MV estiver esperando para reiniciar
        if(this.mv.parou()) {
            if(this.mv.finalizou() ||
                this.mv.travou() ||
                !this.mv.iniciou()) {

                // se a MV finalizou, ou se travou, ou se não tiver iniciado
ainda
                this.mv.iniciar();
            } else {
                this.mv.reiniciar();
            }
        } else {
            this.mv.parar();
        }
    }

    private void entradaComandoKeyPressed(java.awt.event.KeyEvent evt) {
        if(evt.getKeyCode() == java.awt.event.KeyEvent.VK_ENTER) {
            synchronized(this.read_lock) {
                this.console.append(this.entradaComando.getText()+"\n");
                this.entrada = this.entradaComando.getText();
                this.entradaComando.setText("");
            }
        }
    }

```

```

        this.read_lock.notifyAll();
    }
}

private void ajusteVelocidadeStateChanged(javax.swing.event.ChangeEvent
evt) {
    javax.swing.JSlider slider = (javax.swing.JSlider) evt.getSource();
    if(this.mv != null) {
        this.mv.ajustarVelocidade(slider.getValue());
    }
    this.prefs.putInt("velocidade", slider.getValue());
}

private void instrucoesMenuItemActionPerformed(java.awt.event.ActionEvent
evt) {
    if(this.inst_manager_wnd != null) {
        this.inst_manager_wnd.dispose();
    }
    this.inst_manager_wnd = new GerenciadorExtensoes();
    this.inst_manager_wnd.setLocationRelativeTo(this.getRootPane());
    this.inst_manager_wnd.setVisible(true);
}

private void fecharMenuItemActionPerformed(java.awt.event.ActionEvent
evt) {
    if(mv.executando()) {
        if(JOptionPane.NO_OPTION == JOptionPane.showConfirmDialog(this,
"Deseja interromper a execução do programa atual?", "Fechar arquivo",
JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE)) {
            return;
        }
    }

    mv.fecharArquivo();
}

private void stepButtonActionPerformed(java.awt.event.ActionEvent evt) {
    if(this.arquivo == null) {
        JOptionPane.showMessageDialog(this, "Nenhum programa carregado!",
"Erro", JOptionPane.ERROR_MESSAGE);
        return;
    }

    mv.executarPasso();
}

private void sairMenuItemActionPerformed(java.awt.event.ActionEvent evt)
{
    this.fecharInterfaceMVLSI();
}

private void formWindowClosing(java.awt.event.WindowEvent evt) {
    this.fecharInterfaceMVLSI();
}

private int rightClickRow = 0;

private void tabelaInstrucoesMouseReleased(java.awt.event.MouseEvent evt)

```

```

{

    if(!evt.isPopupTrigger()) { return; }

    JTable table = (JTable)evt.getSource();
    int row = table.rowAtPoint(evt.getPoint());
    int col = table.columnAtPoint(evt.getPoint());

    table.setRowSelectionInterval(row, row);

    this.rightClickRow = row;

    this.popupInstrucoes.show(evt.getComponent(), evt.getX(),
evt.getY());
}

private void
removerInstrucaoMenuItemActionPerformed(java.awt.event.ActionEvent evt) {
    int instIndex = this.tabelaInstrucoes.getSelectedRow();
    this.mv.removeInstruction(instIndex);
}

private void tabelaInstrucoesKeyPressed(java.awt.event.KeyEvent evt) {
    if(evt.getKeyCode() == java.awt.event.KeyEvent.VK_DELETE) {
        int instIndex = this.tabelaInstrucoes.getSelectedRow();
        this.mv.removeInstruction(instIndex);
    }
}

private void
inserirInstrucaoAcimaMenuItemActionPerformed(java.awt.event.ActionEvent evt)
{
    int instIndex = this.tabelaInstrucoes.getSelectedRow();

    if(this.insert_inst_wnd != null) {
        this.insert_inst_wnd.dispose();
    }
    this.insert_inst_wnd = new InserirInstrucao(instIndex);
    this.insert_inst_wnd.setLocationRelativeTo(this.getRootPane());
    this.insert_inst_wnd.setVisible(true);
}

private void
inserirInstrucaoAbaixoMenuItemActionPerformed(java.awt.event.ActionEvent evt)
{
    int instIndex = this.tabelaInstrucoes.getSelectedRow();

    if(this.insert_inst_wnd != null) {
        this.insert_inst_wnd.dispose();
    }
    this.insert_inst_wnd = new InserirInstrucao(instIndex+1);
    this.insert_inst_wnd.setLocationRelativeTo(this.getRootPane());
    this.insert_inst_wnd.setVisible(true);
}

private void fecharInterfaceMVLSI() {
    if(mv.executando()) {
        if(JOptionPane.NO_OPTION == JOptionPane.showConfirmDialog(this,

```

```

"Deseja interromper a execução do programa atual?", "Sair",
JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE)) {
    return;
}
}

mv.fecharArquivo();

System.exit(0);
}

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
    try {

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    } catch(ClassNotFoundException e) {
    } catch(InstantiationException e) {
    } catch(UnsupportedLookAndFeelException e) {
    } catch(IllegalAccessException e) {
    }

    /*
     * Create and display the form
     */
    java.awt.EventQueue.invokeLater(new Runnable() {
        Override
        public void run() {
            new InterfaceMVLIS().setVisible(true);
        }
    });
}

// Variables declaration - do not modify
private javax.swing.JMenu abertosRecentementeMenu;
private javax.swing.JMenuItem abrirMenuItem;
private javax.swing.JSlider ajusteVelocidade;
private javax.swing.JTextArea console;
private javax.swing.JTextField entradaComando;
private javax.swing.JButton execButton;
private javax.swing.JMenuItem fecharMenuItem;
private javax.swing.JMenuItem inserirInstrucaoAbaixoMenuItem;
private javax.swing.JMenuItem inserirInstrucaoAcimaMenuItem;
private javax.swing.JMenu inserirInstrucaoMenuItem;
private javax.swing.JMenuItem instrucoesMenuItem;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JLabel jLabel4;
private javax.swing.JLabel jLabel5;
private javax.swing.JLabel jLabel6;
private javax.swing.JLabel jLabel7;
private javax.swing.JLabel jLabel8;
private javax.swing.JLabel jLabel9;
private javax.swing.JMenu jMenu1;
private javax.swing.JMenu jMenu2;
private javax.swing.JMenu jMenu3;

```

```

private javax.swing.JMenuBar jMenuBar1;
private javax.swing.JScrollPane jScrollPane1;
private javax.swing.JScrollPane jScrollPane2;
private javax.swing.JScrollPane jScrollPane3;
private javax.swing.JScrollPane jScrollPane4;
private javax.swing.JScrollPane jScrollPane5;
private javax.swing.JPopupMenu.Separator jSeparator1;
private javax.swing.JPopupMenu.Separator jSeparator3;
private javax.swing.JMenuItem nenhumMenuItem;
private javax.swing.JTextField numExecsTextfield;
private javax.swing.JTextField pcTextfield;
private javax.swing.JPopupMenu popupInstrucoes;
private javax.swing.JTable registradoresBase;
private javax.swing.JMenuItem removerInstrucaoMenuItem;
private javax.swing.JMenuItem sairMenuItem;
private javax.swing.JMenuItem sobreMenuItem;
private javax.swing.JButton stepButton;
private javax.swing.JTable tabelaInstrucoes;
private javax.swing.JTable tabelaLiterais;
private javax.swing.JTable tabelaPilha;
private javax.swing.JTextField topoTextfield;
// End of variables declaration

private void abrirArquivo(File arq) {
    this.arquivo = arq;

    try {
        this.mv.carregarPrograma(this.arquivo);
    } catch (FileNotFoundException e) {
        JOptionPane.showMessageDialog(this, "Arquivo não encontrado!",
"Erro", JOptionPane.ERROR_MESSAGE);
    } catch (IOException e) {
        JOptionPane.showMessageDialog(this, e.getMessage(), "Erro",
JOptionPane.ERROR_MESSAGE);
    } catch (IllegalStateException e) {
        JOptionPane.showMessageDialog(this, e.getMessage(), "Erro",
JOptionPane.ERROR_MESSAGE);
    } catch (InstrucaoNaoReconhecidaException e) {
        JOptionPane.showMessageDialog(this, e.getMessage(), "Erro",
JOptionPane.ERROR_MESSAGE);
    } catch (TipoLiteralNaoReconhecidoException e) {
        JOptionPane.showMessageDialog(this, e.getMessage(), "Erro",
JOptionPane.ERROR_MESSAGE);
    } catch (FormatoIncorretoException e) {
        JOptionPane.showMessageDialog(this, e.getMessage(), "Erro",
JOptionPane.ERROR_MESSAGE);
    }
}

private class ArquivosRecentesList {

    static final int MAXIMO_ELEMENTOS = 10;
    private LinkedList<File> lista = new LinkedList();

    public ArquivosRecentesList() {
        String recente =
InterfaceMVL SI.this.prefs.get("arquivos_recentes", "");
        if(!recente.equals("")) {

```

```

String[] arquivos = recente.split(":::");
int x = 0;
for(String arquivo : arquivos) {
    File arq = new File(arquivo);
    if(this.lista.contains(arq)) { continue; }

    if(x++ > MAXIMO_ELEMENTOS) break;

    if(arq.exists()) {
        this.lista.add(arq);
    }
}
this.flush();
}

public void add(File elem) {
    if(lista.contains(elem)) {
        lista.remove(elem);
    }

    lista.addFirst(elem);
    if(lista.size() >= MAXIMO_ELEMENTOS) { lista.pollLast(); }
    this.flush();
}

private void flush() {
    String recente = "";

    InterfaceMVLSI.this.abertosRecentementeMenu.removeAll();

    for(File elem : this.lista) {
        if(elem == null) break;
        try {
            recente += elem.getCanonicalPath() + "::: ";

            JMenuItem item = new ArquivoJMenuItem(elem);

            item.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/Icones/ArquivoIcane.png")));
            item.setText(elem.getName());
            item.addActionListener(new
java.awt.event.ActionListener() {
                Override
                public void
actionPerformed(java.awt.event.ActionEvent e) {
                    ArquivoJMenuItem item =
(ArquivoJMenuItem)e.getSource();
                    InterfaceMVLSI.this.abrirArquivo(item.arquivo);
                }
            });
            InterfaceMVLSI.this.abertosRecentementeMenu.add(item);

        } catch(IOException e) {}
    }

    if(this.lista.size() == 0) {

InterfaceMVLSI.this.abertosRecentementeMenu.add(InterfaceMVLSI.this.nenhumMen

```

```

uItem);
    }

    InterfaceMVLSI.this.prefs.put("arquivos_recentes", recente);
    try {
        InterfaceMVLSI.this.prefs.flush();
    } catch(BackingStoreException e) {}
}

class ArquivoJMenuItem extends JMenuItem {
    private File arquivo;
    public ArquivoJMenuItem(File arq) {
        super();
        this.arquivo = arq;
    }
}

protected class LiteraisTableModel extends AbstractTableModel {
    ArrayList<Literal> objs = new ArrayList();

    Class[] types = new Class [] {Integer.class, String.class,
Literal.class};

    Override
    public Class getColumnClass(int columnIndex) { return types
[columnIndex]; }
    Override
    public boolean isCellEditable(int rowIndex, int columnIndex) { return
false; }

    Override
    public int getColumnCount() { return 3; }

    Override
    public String getColumnName(int column) {
        switch(column) {
            case 0: return "Índice";
            case 1: return "Tipo";
            case 2: return "Valor";
        }
        return "";
    }

    Override
    public int getRowCount() { return objs.size(); }

    Override
    public Object getValueAt(int row, int column) {
        Literal o = objs.get(row);

        switch(column) {
            case 0:
                return objs.indexOf(o);

            case 1:
                return o.getTipoString();

            case 2:

```



```

        return o.toEscapedString();
    }
    return null;
}

public void setValue(int row, Literal o) {
    this.objs.set(row, o);
    this.fireTableRowsUpdated(row, row);
}

public void removeRow(Literal o) {
    this.objs.remove(o);
    this.fireTableDataChanged();
}

public void removeRow(int row) {
    this.objs.remove(row);
    this.fireTableRowsDeleted(row, row);
}

public void clear() {
    this.objs.clear();
    this.fireTableDataChanged();
}

public void addRow(Literal o) {
    this.objs.add(o);
    this.fireTableDataChanged();
}

public void addRow(int index, Literal o) {
    this.objs.add(index, o);
    this.fireTableDataChanged();
}
}

class MVLSI_ClassLoader extends URLClassLoader {
    public MVLSI_ClassLoader() {
        super(new URL[]{});
    }
    public void addUr1(URL u) {
        super.addURL(u);
    }
}

/*****
    Implementação dos métodos da interface MaquinaVirtualInterface
*****/

/*
 * Método usado pela MV para escrever na tela de saída
 */
@Override
public void write(String s) {
    this.console.append(s);
}

/*
 * Método usado pela MV para receber o input do usuário

```

```

*/
@Override
public String read() {
    String input;

    this.setEntrada("");
    this.entradaComando.setEnabled(true);
    this.entradaComando.requestFocus();

    synchronized(this.read_lock) {
        while(this.entrada.equals("")) {
            try {
                this.read_lock.wait();
            } catch (InterruptedException e) {
                this.entradaComando.setEnabled(false);
                throw new InterruptedException();
            }
        }
    }

    input = this.entrada;
    this.entradaComando.setEnabled(false);
    return input;
}

@Override
public void iniciou() {
    this.console.setText("");
    this.execButton.setText("Pausar");
}

@Override
public void parou() {
    this.execButton.setText("Continuar");
}

@Override
public void reiniciou() {
    this.execButton.setText("Pausar");
}

@Override
public void resetou() {
    this.execButton.setText("Executar");
    this.console.setText("");
}

@Override
public void finalizou() {
    this.execButton.setText("Executar");
}

@Override
public void executando(int PC) {
    this.tabelaInstrucoes.setColumnSelectionInterval(0, 1);
    this.tabelaInstrucoes.setRowSelectionInterval(PC, PC);
}

@Override

```

```

    public void erro(String mensagem) {
        JOptionPane.showMessageDialog(this, mensagem+"\n\nNão é possível
continuar a execução.", "Erro de Execução", JOptionPane.ERROR_MESSAGE);
        this.execButton.setText("Reiniciar");
    }

    @Override
    public void fechou_programa() {
        this.arquivo = null;
        this.fecharMenuItem.setEnabled(false);
        this.setTitle(InterfaceMVLSI.window_title);
        this.execButton.setText("Executar");
        this.console.setText("");
    }

    @Override
    public void abriu_programa() {
        this.entradaComando.setEnabled(false);
        this.arquivos_recentes.add(this.arquivo);
        this.fecharMenuItem.setEnabled(true);
        this.setTitle(InterfaceMVLSI.window_title+" -
"+this.arquivo.getName());
        this.execButton.setText("Executar");
        this.console.setText("");
    }
}

```

#### LiteraisInterfaceObserver.java

```

package UFSC.InterfaceMVLSI;

import UFSC.InterfaceMVLSI.InterfaceMVLSI.LiteraisTableModel;
import UFSC.MaquinaVirtual.ArrayListObserver;
import UFSC.MaquinaVirtual.Literal;
import java.util.Collection;

/**
 * Classe utilizada pela interface gráfica para monitorar a tabela de
 literais da Máquina Virtual.
 * @author LucasMS
 */
public class LiteraisInterfaceObserver<E> implements ArrayListObserver<E> {
    Override
    public void onAdd(E element) {
        Literal elem = (Literal)element;

        ((LiteraisTableModel)InterfaceMVLSI.getInstance().getTabelaLiterais()).getMode
l().addRow(elem);
    }

    public void onAdd(int index, E element) {
        Literal elem = (Literal)element;

        ((LiteraisTableModel)InterfaceMVLSI.getInstance().getTabelaLiterais()).getMode
l().addRow(index, elem);
    }

    Override
    public void onSet(int index, E element) {

```

```

        Literal elem = (Literal)element;

        ((LiteraisTableModel)InterfaceMVLSI.getInstance().getTabelaLiterais().getModel()).setValue(index, elem);
    }

    Override
    public void onRemove( int index ) {

        ((LiteraisTableModel)InterfaceMVLSI.getInstance().getTabelaLiterais().getModel()).removeRow(index);
    }

    Override
    public void onClear() {

        ((LiteraisTableModel)InterfaceMVLSI.getInstance().getTabelaLiterais().getModel()).clear();
    }

    Override
    public void onRemove( Object obj ) {

        ((LiteraisTableModel)InterfaceMVLSI.getInstance().getTabelaLiterais().getModel()).removeRow((Literal)obj);
    };

    Override
    public void onRemoveAll( Collection<?> c ) {
        throw new UnsupportedOperationException();
    };
}

```

### PilhaExecucaoInterfaceObserver.java

```

package UFSC.InterfaceMVLSI;

import UFSC.MaquinaVirtual.ArrayListObserver;
import UFSC.MaquinaVirtual.ItemPilha;
import java.util.Collection;
import javax.swing.table.DefaultTableModel;

/**
 * Classe utilizada pela interface gráfica para monitorar a pilha de execução
 * e atualizar a tabela (JTable) que representa a pilha
 * @author LucasMS
 */
public class PilhaExecucaoInterfaceObserver<E> implements
ArrayListObserver<E> {
    Override
    public void onAdd(E element) {
        ItemPilha elem = (ItemPilha)element;
        Object[] el = new
Object[]{InterfaceMVLSI.getInstance().getMaquinaVirtual().pilha.size()-1,
elem.getTipo(), elem};

        ((DefaultTableModel)InterfaceMVLSI.getInstance().getTabelaPilha().getModel())
.addRow(el);
    }
}

```

```

        public void onAdd(int index, E element) {
            ItemPilha elem = (ItemPilha)element;
            Object[] e1 = new
Object[]{InterfaceMVLSI.getInstance().getMaquinaVirtual().pilha.size()-1,
elem.getTipo(), elem};

            ((DefaultTableModel)InterfaceMVLSI.getInstance().getTabelaPilha().getModel())
            .insertRow(index, e1);
        }

        Override
        public void onSet(int index, E element) {
            ItemPilha elem = (ItemPilha)element;

            ((DefaultTableModel)InterfaceMVLSI.getInstance().getTabelaPilha().getModel())
            .setValueAt(elem.getTipo(), index, 1);

            ((DefaultTableModel)InterfaceMVLSI.getInstance().getTabelaPilha().getModel())
            .setValueAt(elem, index, 2);
        }

        Override
        public void onRemove( int index ) {

            ((DefaultTableModel)InterfaceMVLSI.getInstance().getTabelaPilha().getModel())
            .removeRow(index);
        }
        Override
        public void onClear() {

            ((DefaultTableModel)InterfaceMVLSI.getInstance().getTabelaPilha().getModel())
            .setRowCount(0);
        }

        Override
        public void onRemove( Object obj ) { throw new
UnsupportedOperationException(); };
        Override
        public void onRemoveAll( Collection<?> c ) { throw new
UnsupportedOperationException(); };
    }

```

### RegistradoresBaseInterfaceObserver.java

```

package UFSC.InterfaceMVLSI;

import UFSC.MaquinaVirtual.ArrayListObserver;
import java.util.Collection;
import javax.swing.table.DefaultTableModel;

/**
 * Classe utilizada pela interface gráfica para monitorar a pilha de execução
 * e atualizar a tabela (JTable) que representa a pilha
 * @author LucasMS
 */
public class RegistradoresBaseInterfaceObserver<E> implements
ArrayListObserver<E> {
    Override

```

```

        public void onAdd(E element) {
            Object[] e1 = new
Object[] {InterfaceMVLSI.getInstance().getMaquinaVirtual().pilha.baseIndexOf(e
element), element};

            ((DefaultTableModel)InterfaceMVLSI.getInstance().getTabelaRegistadoresBase()
.getModel()).addRow(e1);
        }

        public void onAdd(int index, E element) {
            Object[] e1 = new
Object[] {InterfaceMVLSI.getInstance().getMaquinaVirtual().pilha.baseIndexOf(e
element), element};

            ((DefaultTableModel)InterfaceMVLSI.getInstance().getTabelaRegistadoresBase()
.getModel()).insertRow(index, e1);
        }

        Override
        public void onSet(int index, E element) {

            ((DefaultTableModel)InterfaceMVLSI.getInstance().getTabelaRegistadoresBase()
.getModel()).setValueAt(element, index, 1);
        }

        Override
        public void onRemove( int index ) {

            ((DefaultTableModel)InterfaceMVLSI.getInstance().getTabelaRegistadoresBase()
.getModel()).removeRow(index);
        }
        Override
        public void onClear() {

            ((DefaultTableModel)InterfaceMVLSI.getInstance().getTabelaRegistadoresBase()
.getModel()).setRowCount(0);
        }

        Override
        public void onRemove( Object obj ) { throw new
UnsupportedOperationException(); };
        Override
        public void onRemoveAll( Collection<?> c ) { throw new
UnsupportedOperationException(); };
    }

```

### TabelaInstrucoesInterfaceObserver.java

```

package UFSC.InterfaceMVLSI;

import UFSC.MaquinaVirtual.ArrayListObserver;
import UFSC.MaquinaVirtual.Instrucao;
import java.util.Collection;
import javax.swing.table.DefaultTableModel;

/**
 * Classe utilizada pela interface gráfica para monitorar a pilha de execução
 * e atualizar a tabela (JTable) que representa o programa (lista de instruções
 * @author LucasMS

```

```

*/
public class TabelaInstrucoesInterfaceObserver<E> implements
ArrayListObserver<E> {
    Override
    public void onAdd(E inst) {
        Instrucao i = (Instrucao)inst;

        short opcode = i.getOpcode();

        Object[] e1;

        if(i.umParam()) {
            e1 = new
Object[]{InterfaceMVLISI.getInstance().getMaquinaVirtual().instrucoes.indexOf(
inst), i.getMnemonico().toUpperCase(), i.getOpcodeString(), "",
i.getParam64()};
        } else {
            e1 = new
Object[]{InterfaceMVLISI.getInstance().getMaquinaVirtual().instrucoes.indexOf(
inst), i.getMnemonico().toUpperCase(), i.getOpcodeString(), i.getParam1(),
i.getParam2()};
        }

        ((DefaultTableModel)InterfaceMVLISI.getInstance().getTabelaInstrucoes().getMod
el()).addRow(e1);
    }

    Override
    public void onAdd(int index, E inst) {
        Instrucao i = (Instrucao)inst;

        short opcode = i.getOpcode();

        Object[] e1;

        if(i.umParam()) {
            e1 = new
Object[]{InterfaceMVLISI.getInstance().getMaquinaVirtual().instrucoes.indexOf(
inst), i.getMnemonico().toUpperCase(), i.getOpcodeString(), "",
i.getParam64()};
        } else {
            e1 = new
Object[]{InterfaceMVLISI.getInstance().getMaquinaVirtual().instrucoes.indexOf(
inst), i.getMnemonico().toUpperCase(), i.getOpcodeString(), i.getParam1(),
i.getParam2()};
        }

        ((DefaultTableModel)InterfaceMVLISI.getInstance().getTabelaInstrucoes().getMod
el()).insertRow(index, e1);

        for(int x = index; x
<InterfaceMVLISI.getInstance().getMaquinaVirtual().instrucoes.size(); x++) {

            ((DefaultTableModel)InterfaceMVLISI.getInstance().getTabelaInstrucoes().getMod
el()).setValueAt(x, x, 0);
        }
    }

    Override

```

```

    public void onSet(int index, E element) {

((DefaultTableModel)InterfaceMVLSI.getInstance().getTabelaInstrucoes().getMod
el()).setValueAt(element, index, 1);
    }

    Override
    public void onRemove( int index ) {

((DefaultTableModel)InterfaceMVLSI.getInstance().getTabelaInstrucoes().getMod
el()).removeRow(index);
        for(int i = index; i
<InterfaceMVLSI.getInstance().getMaquinaVirtual().instrucoes.size(); i++) {

((DefaultTableModel)InterfaceMVLSI.getInstance().getTabelaInstrucoes().getMod
el()).setValueAt(i, i, 0);
        }
    }
    Override
    public void onClear() {

((DefaultTableModel)InterfaceMVLSI.getInstance().getTabelaInstrucoes().getMod
el()).setRowCount(0);
    }

    Override
    public void onRemove( Object obj ) { throw new
UnsupportedOperationException(); }
    Override
    public void onRemoveAll( Collection<?> c ) { throw new
UnsupportedOperationException(); }

}

```



## Apêndice D - Código fonte do pacote padrão de extensões

### AMEMCAD.java

```
package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class AMEMCAD extends Instrucao {

    public AMEMCAD() {
        super.opcode = 0x0015;
        super.mnemonico = "amemcad";
    }

    Override
    public void executar(MaquinaVirtual mv) {
        for(int x = 0; x < this.param2; x++) {
            mv.pilha.add(new ItemPilhaCadeia(0));
            mv.pilha.topo++;
        }
        mv.PC++;
    }
}
```

### AMEM\_B.java

```
package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class AMEM_B extends Instrucao {
    public AMEM_B() {
        super.opcode = 0x0013;
        super.mnemonico = "amem_b";
    }

    Override
    public void executar(MaquinaVirtual mv) {
        for(int x = 0; x < this.param2; x++) {
            mv.pilha.add(new ItemPilhaBooleano(false));
            mv.pilha.topo++;
        }
        mv.PC++;
    }
}
```

### AMEM\_C.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class AMEM_C extends Instrucao {
    public AMEM_C() {
        super.opcode = 0x0012;
        super.mnemonico = "amem_c";
    }

    Override
    public void executar(MaquinaVirtual mv) {
        for(int x = 0; x < this.param2; x++) {
            mv.pilha.add(new ItemPilhaCaracter(' '));
            mv.pilha.topo++;
        }
        mv.PC++;
    }
}

```

#### AMEM\_I.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class AMEM_I extends Instrucao {

    public AMEM_I() {
        super.opcode = 0x0010;
        super.mnemonico = "amem_i";
    }

    Override
    public void executar(MaquinaVirtual mv) {
        for(int x = 0; x < this.param2; x++) {
            mv.pilha.add(new ItemPilhaInteiro(0));
            mv.pilha.topo++;
        }
        mv.PC++;
    }
}

```

#### AMEM\_R.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

```

```

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class AMEM_R extends Instrucao {

    public AMEM_R() {
        super.opcode = 0x0011;
        super.mnemonico = "amem_r";
    }

    Override
    public void executar(MaquinaVirtual mv) {
        for(int x = 0; x < this.param2; x++) {
            mv.pilha.add(new ItemPilhaReal(0.0));
            mv.pilha.topo++;
        }
        mv.PC++;
    }
}

```

## ARMZ.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.MaquinaVirtual;
import UFSC.MaquinaVirtual.TipoIncorretoException;

/**
 *
 * @author LucasMS
 */
public class ARMZ extends Instrucao {

    public ARMZ() {
        super.opcode = 0x0021;
        super.mnemonico = "armz";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        ItemPilha item_fonte = mv.pilha.get(mv.pilha.topo);
        ItemPilha item_destino = mv.pilha.get(mv.pilha.getBase(this.param1) +
this.param2);

        if(!(item_fonte.getClass() == item_destino.getClass())) {
            throw new TipoIncorretoException("Instrução ARMZ. Tipo do espaço
de destino é diferente do novo valor atribuído. Encontrado:
"+item_fonte.getTipo()+". Esperado: "+item_destino.getTipo());
        }

        mv.pilha.set(mv.pilha.getBase(this.param1) + this.param2,
item_fonte);
    }
}

```

```

        mv.pilha.remove(mv.pilha.topo);
        mv.pilha.topo--;
        mv.PC++;
    }
}

```

## ARMZIND.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class ARMZIND extends Instrucao {

    public ARMZIND() {
        super.opcode = 0x0032;
        super.mnemonico = "armzind";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        ItemPilha item_fonte = mv.pilha.get(mv.pilha.topo);

        int indice = mv.pilha.getBase(this.param1) + this.param2;

        ItemPilha item_destino;
        ItemPilhaEndereco item2;

        while(true) {
            item_destino = mv.pilha.get(indice);
            if(item_destino instanceof ItemPilhaEndereco) {
                item2 = (ItemPilhaEndereco)item_destino;
                indice = mv.pilha.getBase(item2.getNivel()) +
item2.getDeslocamento();
            } else {
                break;
            }
        }

        if(!(item_fonte.getClass() == item_destino.getClass())) {
            throw new TipoIncorretoException("Instrução ARMZIND. Tipo do
espaço de destino é diferente do novo valor atribuído. Encontrado:
"+item_fonte.getTipo()+". Esperado: "+item_destino.getTipo());
        }

        mv.pilha.set(indice, item_fonte);

        mv.pilha.remove(mv.pilha.topo);
        mv.pilha.topo--;
        mv.PC++;
    }
}

```

## AVET.java

```
package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.IndiceInvalidoException;
import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.MaquinaVirtual;
import UFSC.MaquinaVirtual.TipoIncorretoException;

/**
 *
 * @author LucasMS
 */
public class AVET extends Instrucao {

    public AVET() {
        super.opcode = 0x0410;
        super.mnemonico = "AVET";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException,
    IndiceInvalidoException {
        // param1: nivel
        // param2: deslocamento do vetor (primeiro espaço contém o tamanho do
        vetor!)

        ItemPilha item_fonte = mv.pilha.get(mv.pilha.topo);

        // verifica se o item representando o tamanho do array é inteiro
        int arr_size_pos = mv.pilha.getBase(this.param1) + this.param2;
        ItemPilha arr_size = mv.pilha.get(arr_size_pos);
        if(!(arr_size instanceof ItemPilhaInteiro)) {
            throw new TipoIncorretoException("Instrução AVET. Tipo incorreto
na pilha na posição "+arr_size_pos+", que deveria indicar o tamanho do array.
Encontrado: "+arr_size.getTipo()+". Esperado: inteiro");
        }

        // verifica se o índice está dentro dos valores permitidos
        ItemPilha item_indice = mv.pilha.get(mv.pilha.topo-1);
        int val_indice = (Integer)item_indice.getValor();
        if(val_indice < 0 || val_indice > ((Integer)arr_size.getValor()-1) )
        {
            throw new IndiceInvalidoException("Instrução AVET. Índice
inválido: "+val_indice+" (não está entre 0 e "+((Integer)arr_size.getValor()-
1)+")");
        }

        // verifica se o item de destino é compatível com o item que será
armazenado
        ItemPilha item_destino = mv.pilha.get(mv.pilha.getBase(this.param1) +
this.param2 + 1 + val_indice);
        if(!(item_fonte.getClass() == item_destino.getClass())) {
            throw new TipoIncorretoException("Instrução AVET. Tipo do array
de destino é diferente do novo valor atribuído. Encontrado:
"+item_destino.getTipo()+". Esperado: "+item_fonte.getTipo());
        }
    }
}
```

```

        mv.pilha.set(mv.pilha.getBase(this.param1) + this.param2 + 1 +
val_indice, item_fonte);

        mv.pilha.remove(mv.pilha.topo);
        mv.pilha.topo--;

        mv.pilha.remove(mv.pilha.topo);
        mv.pilha.topo--;

        mv.PC++;
    }
}

```

## B2CAD.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class B2CAD extends Instrucao {

    public B2CAD() {
        super.opcode = 0x0322;
        super.mnemonico = "b2cad";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        // verifica os tipos de dados no topo da pilha antes de operar
        ItemPilha item1 = mv.pilha.get(mv.pilha.topo);

        if(!(item1 instanceof ItemPilhaBooleano)) {
            throw new TipoIncorretoException("Instrução B2CAD. Tipo incorreto
no topo da pilha. Encontrado: "+item1.getTipo()+". Esperado: booleano");
        }

        // true = verdadeiro, false = falso
        boolean value = ((ItemPilhaBooleano)item1).getValor();

        LiteralCadeia novacadeia = new LiteralCadeia( value ? "verdadeiro" :
"false" );
        int indice = -1;
        synchronized(mv.literais) {
            mv.literais.add(novacadeia);
            indice = mv.literais.size()-1;
        }

        mv.pilha.set(mv.pilha.topo, new ItemPilhaCadeia( indice ));

        mv.PC++;
    }
}

```

```
}  
}
```

## B2I.java

```
package UFSC.MaquinaVirtual.Extensoes.Padrao;  
  
import UFSC.MaquinaVirtual.Instrucao;  
import UFSC.MaquinaVirtual.ItemPilha;  
import UFSC.MaquinaVirtual.TipoIncorretoException;  
import UFSC.MaquinaVirtual.MaquinaVirtual;  
  
/**  
 *  
 * @author LucasMS  
 */  
public class B2I extends Instrucao {  
  
    public B2I() {  
        super.opcode = 0x0320;  
        super.mnemonico = "b2i";  
    }  
  
    Override  
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {  
        // verifica os tipos de dados no topo da pilha antes de operar  
        ItemPilha item1 = mv.pilha.get(mv.pilha.topo);  
  
        if(!(item1 instanceof ItemPilhaBooleano)) {  
            throw new TipoIncorretoException("Instrução B2I. Tipo incorreto  
no topo da pilha. Encontrado: "+item1.getTipo()+". Esperado: booleano");  
        }  
  
        // true = 1, false = 0  
        boolean value = ((ItemPilhaBooleano)item1).getValor();  
  
        mv.pilha.set(mv.pilha.topo, new ItemPilhaInteiro( value ? 1 : 0 ));  
  
        mv.PC++;  
    }  
}
```

## B2R.java

```
package UFSC.MaquinaVirtual.Extensoes.Padrao;  
  
import UFSC.MaquinaVirtual.Instrucao;  
import UFSC.MaquinaVirtual.ItemPilha;  
import UFSC.MaquinaVirtual.TipoIncorretoException;  
import UFSC.MaquinaVirtual.MaquinaVirtual;  
  
/**  
 *  
 * @author LucasMS  
 */  
public class B2R extends Instrucao {  
  
    public B2R() {  
        super.opcode = 0x0321;
```

```

        super.mnemonico = "b2r";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        // verifica os tipos de dados no topo da pilha antes de operar
        ItemPilha item1 = mv.pilha.get(mv.pilha.topo);

        if(!(item1 instanceof ItemPilhaBooleano)) {
            throw new TipoIncorretoException("Instrução B2R. Tipo incorreto
no topo da pilha. Encontrado: "+item1.getTipo()+". Esperado: booleano");
        }

        // true = 1.0, false = 0.0
        boolean value = ((ItemPilhaBooleano)item1).getValor();

        mv.pilha.set(mv.pilha.topo, new ItemPilhaReal( value ? 1.0 : 0.0 ));

        mv.PC++;
    }
}

```

## C2CAD.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class C2CAD extends Instrucao {

    public C2CAD() {
        super.opcode = 0x0331;
        super.mnemonico = "c2cad";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        // verifica os tipos de dados no topo da pilha antes de operar
        ItemPilha item1 = mv.pilha.get(mv.pilha.topo);

        if(!(item1 instanceof ItemPilhaCaracter)) {
            throw new TipoIncorretoException("Instrução C2CAD. Tipo incorreto
no topo da pilha. Encontrado: "+item1.getTipo()+". Esperado: caracter");
        }

        char valor = ((ItemPilhaCaracter)item1).getValor();

        LiteralCadeia novacadeia = new LiteralCadeia( String.valueOf(valor));
        int indice = -1;
        synchronized(mv.literais) {
            mv.literais.add(novacadeia);
            indice = mv.literais.size()-1;
        }
    }
}

```



```

    }

    mv.pilha.set(mv.pilha.topo, new ItemPilhaCadeia(indice));

    mv.PC++;
}
}

```

## C2I.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class C2I extends Instrucao {

    public C2I() {
        super.opcode = 0x0330;
        super.mnemonico = "c2i";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        // verifica os tipos de dados no topo da pilha antes de operar
        ItemPilha item1 = mv.pilha.get(mv.pilha.topo);

        if(!(item1 instanceof ItemPilhaCaracter)) {
            throw new TipoIncorretoException("Instrução C2I. Tipo incorreto
no topo da pilha. Encontrado: "+item1.getTipo()+". Esperado: caracter");
        }

        char valor = ((ItemPilhaCaracter)item1).getValor();
        mv.pilha.set(mv.pilha.topo, new ItemPilhaInteiro(valor));

        mv.PC++;
    }
}

```

## CAD2C.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class CAD2C extends Instrucao {

```

```

public CAD2C() {
    super.opcode = 0x0341;
    super.mnemonic = "cad2c";
}

Override
public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
    // verifica os tipos de dados no topo da pilha antes de operar
    ItemPilha item1 = mv.pilha.get(mv.pilha.topo);

    if(!(item1 instanceof ItemPilhaCadeia)) {
        throw new TipoIncorretoException("Instrução CAD2C. Tipo incorreto
no topo da pilha. Encontrado: "+item1.getTipo()+". Esperado: cadeia");
    }

    // pega apenas o primeiro caracter da cadeia.
    LiteralCadeia str =
(LiteralCadeia)mv.literais.get(((ItemPilhaCadeia)item1).getValor());
    char value = str.toString().charAt(0);
    mv.pilha.set(mv.pilha.topo, new ItemPilhaCaracter(value));

    mv.PC++;
}
}

```

## CAD2I.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;
import UFSC.MaquinaVirtual.FormatoIncorretoException;

/**
 *
 * @author LucasMS
 */
public class CAD2I extends Instrucao {

    public CAD2I() {
        super.opcode = 0x0340;
        super.mnemonic = "cad2i";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException,
FormatoIncorretoException {
        // verifica os tipos de dados no topo da pilha antes de operar
        ItemPilha item1 = mv.pilha.get(mv.pilha.topo);

        if(!(item1 instanceof ItemPilhaCadeia)) {
            throw new TipoIncorretoException("Instrução CAD2I. Tipo incorreto
no topo da pilha. Encontrado: "+item1.getTipo()+". Esperado: cadeia");
        }

        LiteralCadeia str =

```

```

(LiteralCadeia)mv.literais.get(((ItemPilhaCadeia)item1).getValor());

    int value = 0;

    try {
        value = Integer.parseInt(str.toString());
    } catch(NumberFormatException e) {
        throw new FormatoIncorretoException("Não foi possível converter
Cadeia de caracteres em Inteiro");
    }
    mv.pilha.set(mv.pilha.topo, new ItemPilhaInteiro(value));

    mv.PC++;
}
}

```

## CAD2R.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;
import UFSC.MaquinaVirtual.FormatoIncorretoException;

/**
 *
 * @author LucasMS
 */
public class CAD2R extends Instrucao {

    public CAD2R() {
        super.opcode = 0x0342;
        super.mnemonico = "cad2r";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException,
FormatoIncorretoException {
        // verifica os tipos de dados no topo da pilha antes de operar
        ItemPilha item1 = mv.pilha.get(mv.pilha.topo);

        if(!(item1 instanceof ItemPilhaCadeia)) {
            throw new TipoIncorretoException("Instrução CAD2R. Tipo incorreto
no topo da pilha. Encontrado: "+item1.getTipo()+". Esperado: cadeia");
        }

        LiteralCadeia str =
(LiteralCadeia)mv.literais.get(((ItemPilhaCadeia)item1).getValor());

        double value = 0.0;

        try {
            value = Double.parseDouble(str.toString());
        } catch(NumberFormatException e) {
            throw new FormatoIncorretoException("Não foi possível converter
Cadeia de caracteres em Real");
        }
    }
}

```

```

        mv.pilha.set(mv.pilha.topo, new ItemPilhaReal(value));

        mv.PC++;
    }
}

```

## CALL.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class CALL extends Instrucao {

    public CALL() {
        super.opcode = 0x0070;
        super.mnemonico = "call";
    }

    Override
    public void executar(MaquinaVirtual mv) {
        mv.pilha.topo++;
        mv.pilha.add(new ItemPilhaInteiro(mv.PC+1));

        mv.PC = this.param2;

        mv.pilha.topo++;
        mv.pilha.add(new ItemPilhaInteiro(mv.pilha.getBase(this.param1)));
        mv.pilha.setBase(this.param1, mv.pilha.topo+1);
    }
}

```

## CHARAT.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;
import UFSC.MaquinaVirtual.IndiceInvalidoException;

/**
 *
 * @author LucasMS
 */
public class CHARAT extends Instrucao {

    public CHARAT() {
        super.opcode = 0x0606;
        super.mnemonico = "charat";
    }

    Override

```

```

    public void executar(MaquinaVirtual mv) throws TipoIncorretoException,
IndiceInvalidoException {
        ItemPilha cadeia = mv.pilha.get(mv.pilha.topo-1);
        ItemPilha index = mv.pilha.get(mv.pilha.topo);

        if(!(cadeia instanceof ItemPilhaCadeia) || !(index instanceof
ItemPilhaInteiro)) {
            throw new TipoIncorretoException("Instrução CHARAT. Tipos
incorretos no topo da pilha. Encontrado: "+cadeia.getTipo()+",
"+index.getTipo()+". Esperado: cadeia, inteiro");
        }

        LiteralCadeia obj =
(LiteralCadeia)mv.literais.get(((ItemPilhaCadeia)cadeia).getValor());

        char carac = ' ';

        try {
            carac = obj.toString().charAt(
((ItemPilhaInteiro)index).getValor() );
        } catch(IndexOutOfBoundsException e) {
            throw new IndiceInvalidoException("Instrução CHARAT. Posição
inválida");
        }

        ItemPilhaCaracter ch = new ItemPilhaCaracter(carac);

        mv.pilha.set(mv.pilha.topo-1, ch);
        mv.pilha.remove(mv.pilha.topo);
        mv.pilha.topo--;
        mv.PC++;
    }
}

```

## CMAI.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.MaquinaVirtual;
import UFSC.MaquinaVirtual.TipoIncorretoException;

/**
 *
 * @author LucasMS
 */
public class CMAI extends Instrucao {

    public CMAI() {
        super.opcode = 0x0250;
        super.mnemonico = "cmai";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        // verifica os tipos de dados no topo da pilha antes de operar

        ItemPilha item1 = mv.pilha.get(mv.pilha.topo-1);
    }
}

```

```

        ItemPilha item2 = mv.pilha.get(mv.pilha.topo);

        if(item1.getClass() != item2.getClass()) {
            throw new TipoIncorretoException("Instrução CMAI. Tipos
incorretos no topo da pilha. Encontrado: "+item1.getTipo()+",
"+item2.getTipo()+". Esperado: dois valores do mesmo tipo");
        }

        if(item1.comparar(item2) >= 0) {
            mv.pilha.set(mv.pilha.topo-1, new ItemPilhaBooleano(true));
        } else {
            mv.pilha.set(mv.pilha.topo-1, new ItemPilhaBooleano(false));
        }

        mv.pilha.remove(mv.pilha.topo);
        mv.pilha.topo--;
        mv.PC++;
    }
}

```

## CMDF.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class CMDF extends Instrucao {

    public CMDF() {
        super.opcode = 0x0210;
        super.mnemonico = "cmdf";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        // verifica os tipos de dados no topo da pilha antes de operar
        ItemPilha item1 = mv.pilha.get(mv.pilha.topo-1);
        ItemPilha item2 = mv.pilha.get(mv.pilha.topo);
        if(item1.getClass() != item2.getClass()) {
            throw new TipoIncorretoException("Instrução CMDF. Tipos
incorretos no topo da pilha. Encontrado: "+item1.getTipo()+",
"+item2.getTipo()+". Esperado: dois valores do mesmo tipo");
        }

        if(item1.comparar(item2) != 0) {
            mv.pilha.set(mv.pilha.topo-1, new ItemPilhaBooleano(true));
        } else {
            mv.pilha.set(mv.pilha.topo-1, new ItemPilhaBooleano(false));
        }

        mv.pilha.remove(mv.pilha.topo);
        mv.pilha.topo--;
    }
}

```

```

        mv.PC++;
    }
}

```

## CMDFCAD.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class CMDFCAD extends Instrucao {

    public CMDFCAD() {
        super.opcode = 0x0608;
        super.mnemonico = "cmdfcad";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        // verifica os tipos de dados no topo da pilha antes de operar
        ItemPilha item1 = mv.pilha.get(mv.pilha.topo-1);
        ItemPilha item2 = mv.pilha.get(mv.pilha.topo);
        if(!(item1 instanceof ItemPilhaCadeia) || !(item2 instanceof
ItemPilhaCadeia)) {
            throw new TipoIncorretoException("Instrução CMDFCAD. Tipos
incorretos no topo da pilha. Encontrado: "+item1.getTipo()+",
"+item2.getTipo()+". Esperado: cadeia, cadeia");
        }

        String s1 = mv.literais.get( ((ItemPilhaCadeia)item1).getValor()
).toString();
        String s2 = mv.literais.get( ((ItemPilhaCadeia)item2).getValor()
).toString();

        if(!s1.equals(s2)) {
            mv.pilha.set(mv.pilha.topo-1, new ItemPilhaBooleano(true));
        } else {
            mv.pilha.set(mv.pilha.topo-1, new ItemPilhaBooleano(false));
        }

        mv.pilha.remove(mv.pilha.topo);
        mv.pilha.topo--;
        mv.PC++;
    }
}

```

## CMEI.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;

```

```

import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class CMEI extends Instrucao {

    public CMEI() {
        super.opcode = 0x0240;
        super.mnemonico = "cmei";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        // verifica os tipos de dados no topo da pilha antes de operar
        ItemPilha item1 = mv.pilha.get(mv.pilha.topo-1);
        ItemPilha item2 = mv.pilha.get(mv.pilha.topo);
        if(item1.getClass() != item2.getClass()) {
            throw new TipoIncorretoException("Instrução CMEI. Tipos
            incorretos no topo da pilha. Encontrado: "+item1.getTipo()+",
            "+item2.getTipo()+". Esperado: dois valores do mesmo tipo");
        }

        if(item1.comparar(item2) <= 0) {
            mv.pilha.set(mv.pilha.topo-1, new ItemPilhaBooleano(true));
        } else {
            mv.pilha.set(mv.pilha.topo-1, new ItemPilhaBooleano(false));
        }

        mv.pilha.remove(mv.pilha.topo);
        mv.pilha.topo--;
        mv.PC++;
    }
}

```

## CMIG.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class CMIG extends Instrucao {

    public CMIG() {
        super.opcode = 0x0200;
        super.mnemonico = "cmig";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {

```



```

        // verifica os tipos de dados no topo da pilha antes de operar
        ItemPilha item1 = mv.pilha.get(mv.pilha.topo-1);
        ItemPilha item2 = mv.pilha.get(mv.pilha.topo);
        if(item1.getClass() != item2.getClass()) {
            throw new TipoIncorretoException("Instrução CMIG. Tipos
incorretos no topo da pilha. Encontrado: "+item1.getTipo()+",
"+item2.getTipo()+". Esperado: dois valores do mesmo tipo");
        }

        if(item1.comparar(item2) == 0) {
            mv.pilha.set(mv.pilha.topo-1, new ItemPilhaBooleano(true));
        } else {
            mv.pilha.set(mv.pilha.topo-1, new ItemPilhaBooleano(false));
        }

        mv.pilha.remove(mv.pilha.topo);
        mv.pilha.topo--;
        mv.PC++;
    }
}

```

## CMIGCAD.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class CMIGCAD extends Instrucao {

    public CMIGCAD() {
        super.opcode = 0x0607;
        super.mnemonic = "cmigcad";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        // verifica os tipos de dados no topo da pilha antes de operar
        ItemPilha item1 = mv.pilha.get(mv.pilha.topo-1);
        ItemPilha item2 = mv.pilha.get(mv.pilha.topo);
        if(!(item1 instanceof ItemPilhaCadeia) || !(item2 instanceof
ItemPilhaCadeia)) {
            throw new TipoIncorretoException("Instrução CMIGCAD. Tipos
incorretos no topo da pilha. Encontrado: "+item1.getTipo()+",
"+item2.getTipo()+". Esperado: cadeia, cadeia");
        }

        String s1 = mv.literais.get( ((ItemPilhaCadeia)item1).getValor()
).toString();
        String s2 = mv.literais.get( ((ItemPilhaCadeia)item2).getValor()
).toString();

        if(s1.equals(s2)) {

```

```

        mv.pilha.set(mv.pilha.topo-1, new ItemPilhaBooleano(true));
    } else {
        mv.pilha.set(mv.pilha.topo-1, new ItemPilhaBooleano(false));
    }

    mv.pilha.remove(mv.pilha.topo);
    mv.pilha.topo--;
    mv.PC++;
}
}
}

```

## CMMA.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class CMMA extends Instrucao {

    public CMMA() {
        super.opcode = 0x0220;
        super.mnemonico = "cmma";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        // verifica os tipos de dados no topo da pilha antes de operar
        ItemPilha item1 = mv.pilha.get(mv.pilha.topo-1);
        ItemPilha item2 = mv.pilha.get(mv.pilha.topo);
        if(item1.getClass() != item2.getClass()) {
            throw new TipoIncorretoException("Instrução CMMA. Tipos
            incorretos no topo da pilha. Encontrado: "+item1.getTipo()+",
            "+item2.getTipo()+". Esperado: dois valores do mesmo tipo");
        }

        if(item1.comparar(item2) > 0) {
            mv.pilha.set(mv.pilha.topo-1, new ItemPilhaBooleano(true));
        } else {
            mv.pilha.set(mv.pilha.topo-1, new ItemPilhaBooleano(false));
        }

        mv.pilha.remove(mv.pilha.topo);
        mv.pilha.topo--;
        mv.PC++;
    }
}

```

## CMME.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.MaquinaVirtual;

```

```

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.ItemPilha;

/**
 *
 * @author LucasMS
 */
public class CMME extends Instrucao {

    public CMME() {
        super.opcode = 0x0230;
        super.mnemonico = "cmme";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        // verifica os tipos de dados no topo da pilha antes de operar
        ItemPilha item1 = mv.pilha.get(mv.pilha.topo-1);
        ItemPilha item2 = mv.pilha.get(mv.pilha.topo);
        if(item1.getClass() != item2.getClass()) {
            throw new TipoIncorretoException("Instrução CMME. Tipos
            incorretos no topo da pilha. Encontrado: "+item1.getTipo()+",
            "+item2.getTipo()+". Esperado: dois valores do mesmo tipo");
        }

        if(item1.comparar(item2) < 0) {
            mv.pilha.set(mv.pilha.topo-1, new ItemPilhaBooleano(true));
        } else {
            mv.pilha.set(mv.pilha.topo-1, new ItemPilhaBooleano(false));
        }

        mv.pilha.remove(mv.pilha.topo);
        mv.pilha.topo--;
        mv.PC++;
    }
}

```

## CONCAT.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class CONCAT extends Instrucao {

    public CONCAT() {
        super.opcode = 0x0602;
        super.mnemonico = "concat";
    }

    Override

```

```

        public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
            ItemPilha item = mv.pilha.get(mv.pilha.topo-1);
            ItemPilha item2 = mv.pilha.get(mv.pilha.topo);

            if(!(item instanceof ItemPilhaCadeia) || !(item2 instanceof
ItemPilhaCadeia)) {
                throw new TipoIncorretoException("Instrução CONCAT. Tipos
incorretos no topo da pilha. Encontrado: "+item.getTipo()+",
"+item2.getTipo()+". Esperado: cadeia, cadeia");
            }

            LiteralCadeia o1 =
(LiteralCadeia)mv.literais.get(((ItemPilhaCadeia)item).getValor());
            LiteralCadeia o2 =
(LiteralCadeia)mv.literais.get(((ItemPilhaCadeia)item2).getValor());

            LiteralCadeia novacadeia = new LiteralCadeia(o1.toString() +
o2.toString());
            int indice = -1;
            synchronized(mv.literais) {
                mv.literais.add(novacadeia);
                indice = mv.literais.size()-1;
            }

            ItemPilhaCadeia cadeia = new ItemPilhaCadeia( indice );

            mv.pilha.set(mv.pilha.topo-1, cadeia);
            mv.pilha.remove(mv.pilha.topo);
            mv.pilha.topo--;
            mv.PC++;
        }
    }
}

```

## CONJ.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class CONJ extends Instrucao {

    public CONJ() {
        super.opcode = 0x0050;
        super.mnemonico = "conj";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        // verifica os tipos de dados no topo da pilha antes de operar
        ItemPilha item1 = mv.pilha.get(mv.pilha.topo-1);
        ItemPilha item2 = mv.pilha.get(mv.pilha.topo);
        if(!(item1 instanceof ItemPilhaBooleano) || !(item2 instanceof

```

```

ItemPilhaBooleano)) {
    throw new TipoIncorretoException("Instrução CONJ. Tipos
    incorretos no topo da pilha. Encontrado: "+item1.getTipo()+",
    "+item2.getTipo()+". Esperado: booleano, booleano");
}

    if(((ItemPilhaBooleano)item1).getValor() == true &&
    ((ItemPilhaBooleano)item2).getValor() == true) {
        mv.pilha.set(mv.pilha.topo-1, new ItemPilhaBooleano(true));
    } else {
        mv.pilha.set(mv.pilha.topo-1, new ItemPilhaBooleano(false));
    }

    mv.pilha.remove(mv.pilha.topo);
    mv.pilha.topo--;
    mv.PC++;
}
}
}

```

### CRCAD.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class CRCAD extends Instrucao {

    public CRCAD() {
        super.opcode = 0x0605;
        super.mnemonico = "crcad";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {

        ItemPilhaCadeia cadeia = new ItemPilhaCadeia( this.param2 );

        mv.pilha.add(cadeia);
        mv.pilha.topo++;
        mv.PC++;
    }
}

```

### CRCT\_B.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *

```

```

* @author LucasMS
*/
public class CRCT_B extends Instrucao {

    public CRCT_B() {
        super.opcode = 0x0028;
        super.mnemonico = "crct_b";
    }

    Override
    public void executar(MaquinaVirtual mv) {
        boolean val;
        if(this.param2 > 0) {
            val = true;
        } else {
            val = false;
        }

        mv.pilha.add(new ItemPilhaBooleano(val));
        mv.pilha.topo++;
        mv.PC++;
    }
}

```

#### CRCT\_C.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class CRCT_C extends Instrucao {

    public CRCT_C() {
        super.opcode = 0x0027;
        super.mnemonico = "crct_c";
    }

    Override
    public void executar(MaquinaVirtual mv) {
        mv.pilha.add(new ItemPilhaCaracter((char)this.param2));
        mv.pilha.topo++;
        mv.PC++;
    }
}

```

#### CRCT\_I.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *

```

```

* @author LucasMS
*/
public class CRCT_I extends Instrucao {

    public CRCT_I() {
        super.opcode = 0x0025;
        super.mnemonico = "crct_i";
    }

    Override
    public void executar(MaquinaVirtual mv) {
        mv.pilha.add(new ItemPilhaInteiro(this.param2));
        mv.pilha.topo++;
        mv.PC++;
    }
}

```

### CRCT\_R.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class CRCT_R extends Instrucao {

    public CRCT_R() {
        super.opcode = 0x0026;
        super.mnemonico = "crct_r";
        super.umparam = true;
    }

    Override
    public void executar(MaquinaVirtual mv) {
        mv.pilha.add(new ItemPilhaReal(this.param64));
        mv.pilha.topo++;
        mv.PC++;
    }
}

```

### CREN.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class CREN extends Instrucao {

    public CREN() {
        super.opcode = 0x0030;
    }
}

```

```

        super.mnemonic = "cren";
    }

    Override
    public void executar(MaquinaVirtual mv) {
        ItemPilhaEndereco item = new ItemPilhaEndereco(this.param1,
this.param2);

        mv.pilha.add(item);
        mv.pilha.topo++;
        mv.PC++;
    }
}

```

## CRVL.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class CRVL extends Instrucao {

    public CRVL() {
        super.opcode = 0x0020;
        super.mnemonic = "crv1";
    }

    Override
    public void executar(MaquinaVirtual mv) {
        ItemPilha x = mv.pilha.get(mv.pilha.getBase(this.param1) +
this.param2);
        mv.pilha.add(x);
        mv.pilha.topo++;
        mv.PC++;
    }
}

```

## CRVLIND.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class CRVLIND extends Instrucao {

    public CRVLIND() {
        super.opcode = 0x0031;
    }
}

```



```

        super.mnemonico = "crvlind";
    }

    Override
    public void executar(MaquinaVirtual mv) {

        int indice = mv.pilha.getBase(this.param1) + this.param2;

        ItemPilha item;
        ItemPilhaEndereco item2;

        while(true) {
            item = mv.pilha.get(indice);
            if(item instanceof ItemPilhaEndereco) {
                item2 = (ItemPilhaEndereco)item;
                indice = mv.pilha.getBase(item2.getNivel()) +
item2.getDeslocamento();
            } else {
                break;
            }
        }

        mv.pilha.add(item);
        mv.pilha.topo++;
        mv.PC++;
    }
}

```

## CVET.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;
import UFSC.MaquinaVirtual.IndiceInvalidoException;

/**
 *
 * @author LucasMS
 */
public class CVET extends Instrucao {

    public CVET() {
        super.opcode = 0x0400;
        super.mnemonico = "CVET";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException,
IndiceInvalidoException {
        // verifica se o índice no topo da pilha é inteiro
        ItemPilha indice = mv.pilha.get(mv.pilha.topo);
        if(!(indice instanceof ItemPilhaInteiro)) {
            throw new TipoIncorretoException("Instrução CVET. Tipo incorreto
no topo da pilha. Encontrado: "+indice.getTipo()+". Esperado: inteiro");
        }
    }
}

```

```

        // tamanho do array está na posicao PRA[1]+deslocamento . Primeiro
        valor estará em PRA[1]+deslocamento+1
        int arr_size_pos = mv.pilha.getBase(this.param1) + param2;
        ItemPilha arr_size = mv.pilha.get(arr_size_pos);
        if(!(arr_size instanceof ItemPilhaInteiro)) {
            throw new TipoIncorretoException("Instrução CVET. Tipo incorreto
na pilha na posição "+arr_size_pos+", que deveria indicar o tamanho do array.
Encontrado: "+arr_size.getTipo()+". Esperado: inteiro");
        }

        // verifica se o índice está dentro dos valores permitidos
        int val_indice = (Integer)indice.getValor();
        if(val_indice < 0 || val_indice > ((Integer)arr_size.getValor()-1) )
    {
        throw new IndiceInvalidoException("Instrução CVET. Índice
inválido: "+val_indice+" (não está entre 0 e "+((Integer)arr_size.getValor()-
1)+")");
    }

        ItemPilha valor = mv.pilha.get(mv.pilha.getBase(this.param1) +
this.param2 + val_indice + 1);

        mv.pilha.set(mv.pilha.topo, valor);

        mv.PC++;
    }
}

```

## DISJ.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class DISJ extends Instrucao {

    public DISJ() {
        super.opcode = 0x0051;
        super.mnemonico = "disj";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        // verifica os tipos de dados no topo da pilha antes de operar
        ItemPilha item1 = mv.pilha.get(mv.pilha.topo-1);
        ItemPilha item2 = mv.pilha.get(mv.pilha.topo);
        if(!(item1 instanceof ItemPilhaBooleano) || !(item2 instanceof
ItemPilhaBooleano)) {
            throw new TipoIncorretoException("Instrução DISJ. Tipos
incorretos no topo da pilha. Encontrado: "+item1.getTipo()+",
"+item2.getTipo()+". Esperado: booleano, booleano");
        }
    }
}

```

```

        if(((ItemPilhaBooleano)item1).getValor() == true ||
((ItemPilhaBooleano)item2).getValor() == true) {
            mv.pilha.set(mv.pilha.topo-1, new ItemPilhaBooleano(true));
        } else {
            mv.pilha.set(mv.pilha.topo-1, new ItemPilhaBooleano(false));
        }

        mv.pilha.remove(mv.pilha.topo);
        mv.pilha.topo--;
        mv.PC++;
    }
}

```

## DIV.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class DIV extends Instrucao {

    public DIV() {
        super.opcode = 0x0107;
        super.mnemonico = "div";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {

        ItemPilha item1 = mv.pilha.get(mv.pilha.topo-1);
        ItemPilha item2 = mv.pilha.get(mv.pilha.topo);

        if(!(item1 instanceof ItemPilhaReal) || !(item2 instanceof
ItemPilhaReal)) {
            throw new TipoIncorretoException("Instrução DIV. Tipos incorretos
no topo da pilha. Encontrado: "+item1.getTipo()+", "+item2.getTipo()+".
Esperado: real, real");
        }

        // faz a operação de divisão
        Double resultado = ((ItemPilhaReal)item1).getValor() /
((ItemPilhaReal)item2).getValor();
        ItemPilhaReal item_resultado = new ItemPilhaReal(resultado);

        // verifica se a posição de destino do resultado é do mesmo tipo
        if(!(mv.pilha.get(mv.pilha.topo-1) instanceof ItemPilhaReal)) {
            throw new TipoIncorretoException("Instrução DIV. Tipo incorreto
para armazenar valor. Encontrado: "+mv.pilha.get(mv.pilha.topo-
1).getTipo()+". Esperado: real");
        }
    }
}

```

```

        mv.pilha.set(mv.pilha.topo-1, item_resultado);
        mv.pilha.remove(mv.pilha.topo);
        mv.pilha.topo--;
        mv.PC++;
    }
}

```

## DMEM.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class DMEM extends Instrucao {

    public DMEM() {
        super.opcode = 0x0017;
        super.mnemonico = "dmem";
    }

    Override
    public void executar(MaquinaVirtual mv) {
        for(int x = 0; x < this.param2; x++) {
            mv.pilha.remove(mv.pilha.topo);
            mv.pilha.topo--;
        }
        mv.PC++;
    }
}

```

## DSVF.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class DSVF extends Instrucao {

    public DSVF() {
        super.opcode = 0x0061;
        super.mnemonico = "dsvf";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        // verifica os tipos de dados no topo da pilha antes de operar
        ItemPilha item = mv.pilha.get(mv.pilha.topo);
    }
}

```

```

        if(!(item instanceof ItemPilhaBooleano)) {
            throw new TipoIncorretoException("Instrução DSVF. Tipo incorreto
no topo da pilha. Encontrado: "+item.getTipo()+". Esperado: booleano");
        }

        if((Boolean)item.getValor() == false) {
            mv.PC = this.param2;
        } else {
            mv.PC++;
        }

        mv.pilha.remove(mv.pilha.topo);
        mv.pilha.topo--;
    }
}

```

### DSVS.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class DSVS extends Instrucao {

    public DSVS() {
        super.opcode = 0x0060;
        super.mnemonico = "dvs";
    }

    Override
    public void executar(MaquinaVirtual mv) {
        mv.PC = this.param2;
    }
}

```

### FIM.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class FIM extends Instrucao {

    public FIM() {
        super.opcode = 0x4649;
        super.mnemonico = "fim";
    }

    Override

```

```

        public void executar(MaquinaVirtual mv) {
            mv.finalizar();
        }
    }
}

```

## I2B.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class I2B extends Instrucao {

    public I2B() {
        super.opcode = 0x0302;
        super.mnemonico = "i2b";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        // verifica os tipos de dados no topo da pilha antes de operar
        ItemPilha item1 = mv.pilha.get(mv.pilha.topo);

        if(!(item1 instanceof ItemPilhaInteiro)) {
            throw new TipoIncorretoException("Instrução I2B. Tipo incorreto
no topo da pilha. Encontrado: "+item1.getTipo()+". Esperado: inteiro");
        }

        // qualquer valor diferente de zero será true. 0 = falso
        int value = ((ItemPilhaInteiro)item1).getValor();
        mv.pilha.set(mv.pilha.topo, new ItemPilhaBooleano( value != 0 ));

        mv.PC++;
    }
}

```

## I2C.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class I2C extends Instrucao {

    public I2C() {

```

```

        super.opcode = 0x0301;
        super.mnemonico = "i2c";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        // verifica os tipos de dados no topo da pilha antes de operar
        ItemPilha item1 = mv.pilha.get(mv.pilha.topo);

        if(!(item1 instanceof ItemPilhaInteiro)) {
            throw new TipoIncorretoException("Instrução I2C. Tipo incorreto
no topo da pilha. Encontrado: "+item1.getTipo()+". Esperado: inteiro");
        }

        int valor = ((ItemPilhaInteiro)item1).getValor();
        char c = (char)valor;
        mv.pilha.set(mv.pilha.topo, new ItemPilhaCaracter(c));

        mv.PC++;
    }
}

```

## I2CAD.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class I2CAD extends Instrucao {

    public I2CAD() {
        super.opcode = 0x0303;
        super.mnemonico = "i2cad";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        // verifica os tipos de dados no topo da pilha antes de operar
        ItemPilha item1 = mv.pilha.get(mv.pilha.topo);

        if(!(item1 instanceof ItemPilhaInteiro)) {
            throw new TipoIncorretoException("Instrução I2CAD. Tipo incorreto
no topo da pilha. Encontrado: "+item1.getTipo()+". Esperado: inteiro");
        }

        String value = ((ItemPilhaInteiro)item1).getValor().toString();

        LiteralCadeia novacadeia = new LiteralCadeia( value );
        int indice = -1;
        synchronized(mv.literais) {
            mv.literais.add(novacadeia);
            indice = mv.literais.size()-1;
        }
    }
}

```

```

    }
    mv.pilha.set(mv.pilha.topo, new ItemPilhaCadeia(indice));
    mv.PC++;
}
}

```

## I2R.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class I2R extends Instrucao {

    public I2R() {
        super.opcode = 0x0300;
        super.mnemonico = "i2r";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        // verifica os tipos de dados no topo da pilha antes de operar
        ItemPilha item1 = mv.pilha.get(mv.pilha.topo);

        if(!(item1 instanceof ItemPilhaInteiro)) {
            throw new TipoIncorretoException("Instrução I2R. Tipo incorreto
no topo da pilha. Encontrado: "+item1.getTipo()+". Esperado: inteiro");
        }

        mv.pilha.set(mv.pilha.topo, new ItemPilhaReal(
((ItemPilhaInteiro)item1).getValor().doubleValue() ));

        mv.PC++;
    }
}

```

## IMPR.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class IMPR extends Instrucao {

```



```

public IMPR() {
    super.opcode = 0x550;
    super.mnemonico = "impr";
}

Override
public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
    ItemPilha item = mv.pilha.get(mv.pilha.topo);

    if(item.isTipoliteral()) {
        mv.getController().write( mv.literais.get(
((Integer)item.getValor()) ).toString() );
    } else {
        mv.getController().write( item.getValor().toString() );
    }

    mv.pilha.remove(mv.pilha.topo);
    mv.pilha.topo--;
    mv.PC++;
}
}

```

#### IMPRLIT.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class IMPRLIT extends Instrucao {

    public IMPRLIT() {
        super.opcode = 0x0554;
        super.mnemonico = "imprlit";
    }

    Override
    public void executar(MaquinaVirtual mv) {
        mv.getController().write( mv.literais.get(this.param2).toString() );
        mv.PC++;
    }
}

```

#### INICIO.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */

```

```

public class INICIO extends Instrucao {

    public INICIO() {
        super.opcode = 0x494E;
        super.mnemonic = "inicio";
    }

    Override
    public void executar(MaquinaVirtual mv) {
        mv.PC++;
    }
}

```

### ItemPilhaBooleano.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.ItemPilha;

/**
 *
 * @author LucasMS
 */
public class ItemPilhaBooleano extends ItemPilha {

    public ItemPilhaBooleano() {}

    public ItemPilhaBooleano(boolean valor) {
        super(valor);
    }

    Override
    public Boolean getValor() {
        return (Boolean)super.valor;
    }

    Override
    public String getTipo() {
        return "booleano";
    }

    Override
    public String toString() {
        if((Boolean)super.valor) {
            return "verdadeiro";
        }
        return "falso";
    }

    Override
    public int comparar(ItemPilha item) {
        if(this.getValor() == (Boolean)item.getValor()) {
            return 0;
        }
        return -1;
    }
}

```

### ItemPilhaCadeia.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.ItemPilha;

/**
 *
 * @author LucasMS
 */
public class ItemPilhaCadeia extends ItemPilha {

    public ItemPilhaCadeia() {
        super.tipo_literal = true;
    }

    public ItemPilhaCadeia(int indice) {
        super(indice);
        super.tipo_literal = true;
    }

    Override
    public Integer getValor() {
        return (Integer)super.valor;
    }

    Override
    public String getTipo() {
        return "cadeia";
    }

    Override
    public String toString() {
        return super.valor+"";
    }

    Override
    public int comparar(ItemPilha item) {
        throw new UnsupportedOperationException("Método não implementado!");
    }
}

```

#### ItemPilhaCaracter.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.ItemPilha;

/**
 *
 * @author LucasMS
 */
public class ItemPilhaCaracter extends ItemPilha {

    public ItemPilhaCaracter() {}

    public ItemPilhaCaracter(char valor) {
        super(new Character(valor));
    }
}

```

```

Override
public Character getValor() {
    return (Character)super.valor;
}

Override
public String getTipo() {
    return "caracter";
}

// compara este item com outro item inteiro
// retorna 0 se igual, 1 se este item for maior, e -1 se este item for
menor
public int comparar(ItemPilha item) {
    int res = 0;

    if(this.getValor() == item.getValor()) {
        res = 0;
    } else if(this.getValor() > (Character)item.getValor()) {
        res = 1;
    } else if(this.getValor() < (Character)item.getValor()){
        res = -1;
    }
    return res;
}
}

```

#### ItemPilhaEndereco.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.ItemPilha;

/**
 *
 * @author LucasMS
 */
public class ItemPilhaEndereco extends ItemPilha {

    private int nivel;
    private int deslocamento;

    public ItemPilhaEndereco() {}

    public ItemPilhaEndereco(int n, int x) {
        super("(" + n + ", " + x + ")");

        this.nivel = n;
        this.deslocamento = x;
    }

    Override
    public String getValor() {
        return (String)super.valor;
    }

    Override
    public String getTipo() {
        return "endereco";
    }
}

```

```

    }

    public int getNivel() { return this.nivel; }
    public int getDeslocamento() { return this.deslocamento; }

    Override
    public int comparar(ItemPilha item) {
        return 0;
    };
}

```

### ItemPilhaInteiro.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.ItemPilha;

/**
 *
 * @author LucasMS
 */
public class ItemPilhaInteiro extends ItemPilha {

    public ItemPilhaInteiro() {}

    public ItemPilhaInteiro(int valor) {
        super(new Integer(valor));
    }

    Override
    public Integer getValor() {
        return (Integer)super.valor;
    }

    Override
    public String getTipo() {
        return "inteiro";
    }

    // compara este item com outro item inteiro
    // retorna 0 se igual, 1 se este item for maior, e -1 se este item for
menor
    public int comparar(ItemPilha item) {
        int res = 0;

        if(this.getValor() == item.getValor()) {
            res = 0;
        } else if(this.getValor() > (Integer)item.getValor()) {
            res = 1;
        } else if(this.getValor() < (Integer)item.getValor()){
            res = -1;
        }
        return res;
    }
}

```

### ItemPilhaReal.java

```

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.ItemPilha;

/**
 *
 * @author LucasMS
 */
public class ItemPilhaReal extends ItemPilha {

    public ItemPilhaReal() {}

    public ItemPilhaReal(double valor) {
        super(new Double(valor));
    }

    Override
    public Double getValor() {
        return (Double)super.valor;
    }

    Override
    public String getTipo() {
        return "real";
    }

    // compara este item com outro item real
    // retorna 0 se igual, 1 se este item for maior, e -1 se este item for
menor
    public int comparar(ItemPilha item) {
        int res = 0;

        if(this.getValor() == item.getValor()) {
            res = 0;
        } else if(this.getValor() > (Double)item.getValor()) {
            res = 1;
        } else if(this.getValor() < (Double)item.getValor()){
            res = -1;
        }
        return res;
    }
}

```

## LEIACAD.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class LEIACAD extends Instrucao {

```

```

public LEIACAD() {
    super.mnemonico = "leiacad";
    super.opcode = 0x0503;
}

Override
public void executar(MaquinaVirtual mv) {
    String input = mv.getController().read();
    int indice = -1;

    synchronized(mv.literais) {
        LiteralCadeia literal = new LiteralCadeia(input);
        mv.literais.add(literal);
        indice = mv.literais.size() - 1;
    }

    mv.pilha.add(new ItemPilhaCadeia(indice));
    mv.pilha.topo++;
    mv.PC++;
}
}

```

#### LEIA\_C.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class LEIA_C extends Instrucao {

    public LEIA_C() {
        super.mnemonico = "leia_c";
        super.opcode = 0x0502;
    }

    Override
    public void executar(MaquinaVirtual mv) {
        String input = mv.getController().read();

        try {
            char caracter = input.charAt(0);
            mv.pilha.add(new ItemPilhaCaracter(caracter));
            mv.pilha.topo++;
            mv.PC++;
        } catch (IndexOutOfBoundsException e) {
            mv.getController().write("Erro: caracter inválido.\n");
        }
    }
}

```

#### LEIA\_I.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class LEIA_I extends Instrucao {

    public LEIA_I() {
        super.mnemonico = "leia_i";
        super.opcode = 0x0500;
    }

    Override
    public void executar(MaquinaVirtual mv) {
        String input = mv.getController().read();

        try {
            int inteiro = Integer.parseInt(input);
            mv.pilha.add(new ItemPilhaInteiro(inteiro));
            mv.pilha.topo++;
            mv.PC++;
        } catch(NumberFormatException e) {
            mv.getController().write("Erro: apenas números inteiros são
aceitos como entrada.\n");
        }
    }
}

```

#### LEIA\_R.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class LEIA_R extends Instrucao {

    public LEIA_R() {
        super.mnemonico = "leia_r";
        super.opcode = 0x0501;
    }

    Override
    public void executar(MaquinaVirtual mv) {
        String input = mv.getController().read();

        try {
            double numero = Double.parseDouble(input);
            mv.pilha.add(new ItemPilhaReal(numero));
            mv.pilha.topo++;
            mv.PC++;
        }
    }
}

```



```

        } catch(NumberFormatException e) {
            mv.getController().write("Erro: apenas números reais são aceitos
como entrada.\n");
        }
    }
}

```

### LiteralCadeia.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Literal;
import java.nio.charset.Charset;

/**
 * Representa uma cadeia de caracteres (string)
 * @author LucasMS
 */
public class LiteralCadeia extends Literal {

    // necessário para instânciação dinâmica
    public LiteralCadeia() {
        super.tipo = 0x01;
    }

    public LiteralCadeia(int size, byte[] value, boolean literal) {
        super((byte)0x01, size, value, literal);
    }

    /**
     public LiteralCadeia(int size, byte[] value) {
         super((byte)0x01, size, value);
     }
    */

    public LiteralCadeia(String value) {
        super((byte)0x01, value.length(), value.getBytes());
    }

    Override
    public String toString() {
        return new String(this.getValor(), Charset.forName("UTF-8"));
    }

    Override
    public String getTipoString() {
        return "cadeia";
    }
}

```

### MULT\_I.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;

```

```

/**
 *
 * @author LucasMS
 */
public class MULT_I extends Instrucao {

    public MULT_I() {
        super.opcode = 0x0104;
        super.mnemonico = "mult_i";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        // nesta instrução, é necessário verificar:
        // - os tipos de dados das duas instruções do topo
        // - o tipo de dado do item de destino do resultado (nesta instrução,
o subtopo)

        ItemPilha item1 = mv.pilha.get(mv.pilha.topo-1);
        ItemPilha item2 = mv.pilha.get(mv.pilha.topo);

        // verifica os tipos de dados no topo da pilha antes de operar
        if(!(item1 instanceof ItemPilhaInteiro) || !(item2 instanceof
ItemPilhaInteiro)) {
            throw new TipoIncorretoException("Instrução MULT_I. Tipos
incorretos no topo da pilha. Encontrado: "+item1.getTipo()+",
"+item2.getTipo()+". Esperado: inteiro, inteiro");
        }

        // faz a operação de multiplicação
        Integer resultado = ((ItemPilhaInteiro)item1).getValor() *
((ItemPilhaInteiro)item2).getValor();
        ItemPilhaInteiro item_resultado = new ItemPilhaInteiro(resultado);

        // verifica se a posição de destino do resultado é do mesmo tipo
        if(!(mv.pilha.get(mv.pilha.topo-1) instanceof ItemPilhaInteiro)) {
            throw new TipoIncorretoException("Instrução MULT_I. Tipo
incorreto para armazenar valor. Encontrado: "+mv.pilha.get(mv.pilha.topo-
1).getTipo()+". Esperado: inteiro");
        }

        mv.pilha.set(mv.pilha.topo-1, item_resultado);
        mv.pilha.remove(mv.pilha.topo);
        mv.pilha.topo--;
        mv.PC++;
    }
}

```

## MULT\_R.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**

```

```

*
* @author LucasMS
*/
public class MULT_R extends Instrucao {

    public MULT_R() {
        super.opcode = 0x0105;
        super.mnemonico = "mult_r";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        // nesta instrução, é necessário verificar:
        // - os tipos de dados das duas instruções do topo
        // - o tipo de dado do item de destino do resultado (nesta instrução,
o subtopo)

        ItemPilha item1 = mv.pilha.get(mv.pilha.topo-1);
        ItemPilha item2 = mv.pilha.get(mv.pilha.topo);

        // verifica os tipos de dados no topo da pilha antes de operar
        if(!(item1 instanceof ItemPilhaReal) || !(item2 instanceof
ItemPilhaReal)) {
            throw new TipoIncorretoException("Instrução MULT_R. Tipos
incorretos no topo da pilha. Encontrado: "+item1.getTipo()+",
"+item2.getTipo()+". Esperado: real, real");
        }

        // faz a operação de multiplicação
        Double resultado = ((ItemPilhaReal)item1).getValor() *
((ItemPilhaReal)item2).getValor();
        ItemPilhaReal item_resultado = new ItemPilhaReal(resultado);

        // verifica se a posição de destino do resultado é do mesmo tipo
        if(!(mv.pilha.get(mv.pilha.topo-1) instanceof ItemPilhaReal)) {
            throw new TipoIncorretoException("Instrução MULT_R. Tipo
incorreto para armazenar valor. Encontrado: "+mv.pilha.get(mv.pilha.topo-
1).getTipo()+". Esperado: real");
        }

        mv.pilha.set(mv.pilha.topo-1, item_resultado);
        mv.pilha.remove(mv.pilha.topo);
        mv.pilha.topo--;
        mv.PC++;
    }
}

```

## MUN\_I.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS

```

```

*/
public class MUN_I extends Instrucao {

    public MUN_I() {
        super.opcode = 0x0108;
        super.mnemonico = "mun_i";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        // verifica os tipos de dados no topo da pilha antes de operar
        ItemPilha item1 = mv.pilha.get(mv.pilha.topo);

        if(!(item1 instanceof ItemPilhaInteiro)) {
            throw new TipoIncorretoException("Instrução MUN_I. Tipo incorreto
no topo da pilha. Encontrado: "+item1.getTipo()+". Esperado: inteiro");
        }

        mv.pilha.set(mv.pilha.topo, new ItemPilhaInteiro(
((ItemPilhaInteiro)item1).getValor() * -1));

        mv.PC++;
    }
}

```

#### MUN\_R.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class MUN_R extends Instrucao {

    public MUN_R() {
        super.opcode = 0x0109;
        super.mnemonico = "mun_r";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        // verifica os tipos de dados no topo da pilha antes de operar
        ItemPilha item1 = mv.pilha.get(mv.pilha.topo);

        if(!(item1 instanceof ItemPilhaReal)) {
            throw new TipoIncorretoException("Instrução MUN_R. Tipo incorreto
no topo da pilha. Encontrado: "+item1.getTipo()+". Esperado: real");
        }

        mv.pilha.set(mv.pilha.topo, new ItemPilhaReal(
((ItemPilhaReal)item1).getValor() * -1));

        mv.PC++;
    }
}

```

```
}  
}
```

## NADA.java

```
package UFSC.MaquinaVirtual.Extensoes.Padrao;  
  
import UFSC.MaquinaVirtual.Instrucao;  
import UFSC.MaquinaVirtual.MaquinaVirtual;  
  
/**  
 *  
 * @author LucasMS  
 */  
public class NADA extends Instrucao {  
    public NADA() {  
        super.opcode = 0x4E44;  
        super.mnemonico = "nada";  
    }  
  
    Override  
    public void executar(MaquinaVirtual mv) {  
        mv.PC++;  
    }  
}
```

## NEGA.java

```
package UFSC.MaquinaVirtual.Extensoes.Padrao;  
  
import UFSC.MaquinaVirtual.Instrucao;  
import UFSC.MaquinaVirtual.ItemPilha;  
import UFSC.MaquinaVirtual.TipoIncorretoException;  
import UFSC.MaquinaVirtual.MaquinaVirtual;  
  
/**  
 *  
 * @author LucasMS  
 */  
public class NEGA extends Instrucao {  
  
    public NEGA() {  
        super.opcode = 0x0052;  
        super.mnemonico = "nega";  
    }  
  
    Override  
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {  
        // verifica os tipos de dados no topo da pilha antes de operar  
        ItemPilha item1 = mv.pilha.get(mv.pilha.topo);  
  
        if(!(item1 instanceof ItemPilhaBooleano)) {  
            throw new TipoIncorretoException("Instrução NEGA. Tipos  
incorretos no topo da pilha. Encontrado: "+item1.getTipo()+". Esperado:  
booleano");  
        }  
  
        if(((ItemPilhaBooleano)item1).getValor() == true) {  
            mv.pilha.set(mv.pilha.topo, new ItemPilhaBooleano(false));  
        }  
    }  
}
```

```

    } else {
        mv.pilha.set(mv.pilha.topo, new ItemPilhaBooleano(true));
    }

    mv.PC++;
}
}
}

```

## POSCAD.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class POSCAD extends Instrucao {

    public POSCAD() {
        super.opcode = 0x0603;
        super.mnemonico = "poscad";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        ItemPilha cadeia = mv.pilha.get(mv.pilha.topo-1);
        ItemPilha subcadeia = mv.pilha.get(mv.pilha.topo);

        if(!(cadeia instanceof ItemPilhaCadeia) || !(subcadeia instanceof
ItemPilhaCadeia)) {
            throw new TipoIncorretoException("Instrução POSCAD. Tipo
incorreto no topo da pilha. Encontrado: "+cadeia.getTipo()+",
"+subcadeia.getTipo()+". Esperado: cadeia, cadeia");
        }

        String s1 = ((LiteralCadeia)mv.literais.get(
((ItemPilhaCadeia)cadeia).getValor() )).toString();
        String s2 = ((LiteralCadeia)mv.literais.get(
((ItemPilhaCadeia)subcadeia).getValor() )).toString();

        ItemPilhaInteiro posicao = new ItemPilhaInteiro(s1.indexOf(s2));

        mv.pilha.set(mv.pilha.topo-1, posicao);
        mv.pilha.remove(mv.pilha.topo);
        mv.pilha.topo--;
        mv.PC++;
    }
}

```

## R2B.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;

```

```

import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class R2B extends Instrucao {

    public R2B() {
        super.opcode = 0x0312;
        super.mnemonico = "r2b";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        // verifica os tipos de dados no topo da pilha antes de operar
        ItemPilha item1 = mv.pilha.get(mv.pilha.topo);

        if(!(item1 instanceof ItemPilhaReal)) {
            throw new TipoIncorretoException("Instrução R2B. Tipo incorreto
no topo da pilha. Encontrado: "+item1.getTipo()+". Esperado: real");
        }

        // qualquer valor diferente de zero será true. 0 = falso
        double value = ((ItemPilhaReal)item1).getValor();
        mv.pilha.set(mv.pilha.topo, new ItemPilhaBooleano( value != 0 ));

        mv.PC++;
    }
}

```

## R2C.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class R2C extends Instrucao {

    public R2C() {
        super.opcode = 0x0311;
        super.mnemonico = "r2c";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        // verifica os tipos de dados no topo da pilha antes de operar
        ItemPilha item1 = mv.pilha.get(mv.pilha.topo);

        if(!(item1 instanceof ItemPilhaReal)) {

```

```

        throw new TipoIncorretoException("Instrução R2C. Tipo incorreto
no topo da pilha. Encontrado: "+item1.getTipo()+". Esperado: real");
    }

    double valor = ((ItemPilhaReal)item1).getValor();
    // trunca o valor double para char (elimina a parte decimal)
    char c = (char)valor;
    mv.pilha.set(mv.pilha.topo, new ItemPilhaCaracter(c));

    mv.PC++;
}
}

```

## R2CAD.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class R2CAD extends Instrucao {

    public R2CAD() {
        super.opcode = 0x0313;
        super.mnemonico = "r2cad";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        // verifica os tipos de dados no topo da pilha antes de operar
        ItemPilha item1 = mv.pilha.get(mv.pilha.topo);

        if(!(item1 instanceof ItemPilhaReal)) {
            throw new TipoIncorretoException("Instrução R2CAD. Tipo incorreto
no topo da pilha. Encontrado: "+item1.getTipo()+". Esperado: real");
        }

        String value = ((ItemPilhaReal)item1).getValor().toString();

        LiteralCadeia novacadeia = new LiteralCadeia( value );
        int indice;

        synchronized(mv.literais) {
            mv.literais.add(novacadeia);
            indice = mv.literais.size()-1;
        }

        mv.pilha.set(mv.pilha.topo, new ItemPilhaCadeia(indice));

        mv.PC++;
    }
}

```



## R2I.java

```
package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class R2I extends Instrucao {

    public R2I() {
        super.opcode = 0x0310;
        super.mnemonico = "r2i";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        // verifica os tipos de dados no topo da pilha antes de operar
        ItemPilha item1 = mv.pilha.get(mv.pilha.topo);

        if(!(item1 instanceof ItemPilhaReal)) {
            throw new TipoIncorretoException("Instrução R2I. Tipo incorreto
no topo da pilha. Encontrado: "+item1.getTipo()+". Esperado: real");
        }

        // 3.999 -> 3 (truncamento)
        mv.pilha.set(mv.pilha.topo, new ItemPilhaInteiro(
((ItemPilhaReal)item1).getValor().intValue() ));

        mv.PC++;
    }
}
```

## RETU.java

```
package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class RETU extends Instrucao {

    public RETU() {
        super.opcode = 0x0071;
        super.mnemonico = "retu";
    }

    Override
    public void executar(MaquinaVirtual mv) {
```

```

        mv.pilha.setBase(this.param1,
(Integer)mv.pilha.get(mv.pilha.topo).getValor());

        mv.pilha.remove(mv.pilha.topo);
        mv.pilha.topo--;

        mv.PC = ((Integer)mv.pilha.get(mv.pilha.topo).getValor());

        mv.pilha.remove(mv.pilha.topo);
        mv.pilha.topo--;

        for(int x = 0; x < this.param2; x++) {
            mv.pilha.remove(mv.pilha.topo);
            mv.pilha.topo--;
        }
    }
}

```

### SOMA\_I.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class SOMA_I extends Instrucao {

    public SOMA_I() {
        super.opcode = 0x0100;
        super.mnemonico = "soma_i";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        // nesta instrução, é necessário verificar:
        // - os tipos de dados das duas instruções do topo
        // - o tipo de dado do item de destino do resultado (nesta instrução,
o subtopo)

        ItemPilha item1 = mv.pilha.get(mv.pilha.topo-1);
        ItemPilha item2 = mv.pilha.get(mv.pilha.topo);

        // verifica os tipos de dados no topo da pilha antes de operar
        if(!(item1 instanceof ItemPilhaInteiro) || !(item2 instanceof
ItemPilhaInteiro)) {
            throw new TipoIncorretoException("Instrução SOMA_I. Tipos
incorretos no topo da pilha. Encontrado: "+item1.getTipo()+",
"+item2.getTipo()+". Esperado: inteiro, inteiro");
        }

        // faz a operação de soma
        Integer resultado = ((ItemPilhaInteiro)item1).getValor() +
((ItemPilhaInteiro)item2).getValor();
    }
}

```

```

        ItemPilhaInteiro item_resultado = new ItemPilhaInteiro(resultado);

        // verifica se a posição de destino do resultado é do mesmo tipo
        if(!(mv.pilha.get(mv.pilha.topo-1) instanceof ItemPilhaInteiro)) {
            throw new TipoIncorretoException("Instrução SOMA_I. Tipo
            incorreto para armazenar valor. Encontrado: "+mv.pilha.get(mv.pilha.topo-
            1).getTipo()+". Esperado: inteiro");
        }

        mv.pilha.set(mv.pilha.topo-1, item_resultado);
        mv.pilha.remove(mv.pilha.topo);
        mv.pilha.topo--;
        mv.PC++;
    }
}

```

## SOMA\_R.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class SOMA_R extends Instrucao {

    public SOMA_R() {
        super.opcode = 0x0101;
        super.mnemonico = "soma_r";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        // nesta instrução, é necessário verificar:
        // - os tipos de dados das duas instruções do topo
        // - o tipo de dado do item de destino do resultado (nesta instrução,
        o subtopo)

        ItemPilha item1 = mv.pilha.get(mv.pilha.topo-1);
        ItemPilha item2 = mv.pilha.get(mv.pilha.topo);

        // verifica os tipos de dados no topo da pilha antes de operar
        if(!(item1 instanceof ItemPilhaReal) || !(item2 instanceof
        ItemPilhaReal)) {
            throw new TipoIncorretoException("Instrução SOMA_R. Tipos
            incorretos no topo da pilha. Encontrado: "+item1.getTipo()+",
            "+item2.getTipo()+". Esperado: real, real");
        }

        // faz a operação de soma
        Double resultado = ((ItemPilhaReal)item1).getValor() +
        ((ItemPilhaReal)item2).getValor();
        ItemPilhaReal item_resultado = new ItemPilhaReal(resultado);
    }
}

```

```

        // verifica se a posição de destino do resultado é do mesmo tipo
        if(!(mv.pilha.get(mv.pilha.topo-1) instanceof ItemPilhaReal)) {
            throw new TipoIncorretoException("Instrução SOMA_R. Tipo
            incorreto para armazenar valor. Encontrado: "+mv.pilha.get(mv.pilha.topo-
            1).getTipo()+". Esperado: real");
        }

        mv.pilha.set(mv.pilha.topo-1, item_resultado);
        mv.pilha.remove(mv.pilha.topo);
        mv.pilha.topo--;
        mv.PC++;
    }
}

```

## SUBCAD.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class SUBCAD extends Instrucao {

    public SUBCAD() {
        super.opcode = 0x0604;
        super.mnemonico = "subcad";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        ItemPilha cadeia = mv.pilha.get(mv.pilha.topo-2);
        ItemPilha inicio = mv.pilha.get(mv.pilha.topo-1);
        ItemPilha fim = mv.pilha.get(mv.pilha.topo);

        if(!(cadeia instanceof ItemPilhaCadeia) || !(inicio instanceof
        ItemPilhaInteiro) || !(fim instanceof ItemPilhaInteiro)) {
            throw new TipoIncorretoException("Instrução SUBCAD. Tipo
            incorreto no topo da pilha. Encontrado: "+cadeia.getTipo()+",
            "+inicio.getTipo()+", "+fim.getTipo()+". Esperado: cadeia, inteiro,
            inteiro");
        }

        String str = ((LiteralCadeia)mv.literais.get(
        ((ItemPilhaCadeia)cadeia).getValor() )).toString();
        int start = ((ItemPilhaInteiro)inicio).getValor();
        int end = ((ItemPilhaInteiro)fim).getValor();

        String substr = str.substring(start, end);

        LiteralCadeia novacadeia = new LiteralCadeia(substr);
        int indice = -1;
        synchronized(mv.literais) {
            mv.literais.add(novacadeia);
        }
    }
}

```

```

        indice = mv.literais.size()-1;
    }

    ItemPilhaCadeia subcadeia = new ItemPilhaCadeia( indice );

    mv.pilha.set(mv.pilha.topo-2, subcadeia);
    mv.pilha.remove(mv.pilha.topo);
    mv.pilha.topo--;
    mv.pilha.remove(mv.pilha.topo);
    mv.pilha.topo--;
    mv.PC++;
}
}
}

```

## SUB\_I.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class SUB_I extends Instrucao {

    public SUB_I() {
        super.opcode = 0x0102;
        super.mnemonico = "sub_i";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        // nesta instrução, é necessário verificar:
        // - os tipos de dados das duas instruções do topo
        // - o tipo de dado do item de destino do resultado (nesta instrução,
o subtopo)

        ItemPilha item1 = mv.pilha.get(mv.pilha.topo-1);
        ItemPilha item2 = mv.pilha.get(mv.pilha.topo);

        // verifica os tipos de dados no topo da pilha antes de operar
        if(!(item1 instanceof ItemPilhaInteiro) || !(item2 instanceof
ItemPilhaInteiro)) {
            throw new TipoIncorretoException("Instrução SUB_I. Tipos
incorretos no topo da pilha. Encontrado: "+item1.getTipo()+",
"+item2.getTipo()+". Esperado: inteiro, inteiro");
        }

        // faz a operação de subtração
        Integer resultado = ((ItemPilhaInteiro)item1).getValor() -
((ItemPilhaInteiro)item2).getValor();
        ItemPilhaInteiro item_resultado = new ItemPilhaInteiro(resultado);

        // verifica se a posição de destino do resultado é do mesmo tipo
        if(!(mv.pilha.get(mv.pilha.topo-1) instanceof ItemPilhaInteiro)) {

```

```

        throw new TipoIncorretoException("Instrução SUB_I. Tipo incorreto
para armazenar valor. Encontrado: "+mv.pilha.get(mv.pilha.topo-
1).getTipo()+". Esperado: inteiro");
    }

    mv.pilha.set(mv.pilha.topo-1, item_resultado);
    mv.pilha.remove(mv.pilha.topo);
    mv.pilha.topo--;
    mv.PC++;
}
}
}

```

## SUB\_R.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class SUB_R extends Instrucao {

    public SUB_R() {
        super.opcode = 0x0103;
        super.mnemonico = "sub_r";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        // nesta instrução, é necessário verificar:
        // - os tipos de dados das duas instruções do topo
        // - o tipo de dado do item de destino do resultado (nesta instrução,
o subtopo)

        ItemPilha item1 = mv.pilha.get(mv.pilha.topo-1);
        ItemPilha item2 = mv.pilha.get(mv.pilha.topo);

        // verifica os tipos de dados no topo da pilha antes de operar
        if(!(item1 instanceof ItemPilhaReal) || !(item2 instanceof
ItemPilhaReal)) {
            throw new TipoIncorretoException("Instrução SUB_R. Tipos
incorretos no topo da pilha. Encontrado: "+item1.getTipo()+",
"+item2.getTipo()+". Esperado: real, real");
        }

        // faz a operação de subtração
        Double resultado = ((ItemPilhaReal)item1).getValor() -
((ItemPilhaReal)item2).getValor();
        ItemPilhaReal item_resultado = new ItemPilhaReal(resultado);

        // verifica se a posição de destino do resultado é do mesmo tipo
        if(!(mv.pilha.get(mv.pilha.topo-1) instanceof ItemPilhaReal)) {
            throw new TipoIncorretoException("Instrução SUB_R. Tipo incorreto
para armazenar valor. Encontrado: "+mv.pilha.get(mv.pilha.topo-

```

```

1).getTipo()+". Esperado: real");
    }

    mv.pilha.set(mv.pilha.topo-1, item_resultado);
    mv.pilha.remove(mv.pilha.topo);
    mv.pilha.topo--;
    mv.PC++;
}
}

```

## TAMCAD.java

```

package UFSC.MaquinaVirtual.Extensoes.Padrao;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.TipoIncorretoException;
import UFSC.MaquinaVirtual.MaquinaVirtual;

/**
 *
 * @author LucasMS
 */
public class TAMCAD extends Instrucao {

    public TAMCAD() {
        super.opcode = 0x0601;
        super.mnemonico = "tamcad";
    }

    Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        ItemPilha item = mv.pilha.get(mv.pilha.topo);

        if(!(item instanceof ItemPilhaCadeia)) {
            throw new TipoIncorretoException("Instrução TAMCAD. Tipo
            incorreto no topo da pilha. Encontrado: "+item.getTipo()+". Esperado:
            cadeia");
        }

        LiteralCadeia obj = (LiteralCadeia)mv.literais.get(
        ((ItemPilhaCadeia)item).getValor() );
        ItemPilhaInteiro tamanho = new
        ItemPilhaInteiro(obj.toString().length());

        mv.pilha.set(mv.pilha.topo, tamanho);
        mv.PC++;
    }
}

```

## Apêndice E - Código fonte do pacote de extensão - Matrículas

### AMEMMATR.java

```
package UFSC.MaquinaVirtual.Extensoes.TipoMatricula;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.MaquinaVirtual;

public class AMEMMATR extends Instrucao {

    public AMEMMATR() {
        super.opcode = 0x0701;
        super.mnemonico = "amemmatr";
    }

    @Override
    public void executar(MaquinaVirtual mv) {
        for(int x = 0; x < this.param2; x++) {
            mv.pilha.add(new ItemPilhaMatricula(0));
            mv.pilha.topo++;
        }
        mv.PC++;
    }
}
```

### CMAIMATR.java

```
package UFSC.MaquinaVirtual.Extensoes.TipoMatricula;

import UFSC.MaquinaVirtual.Extensoes.Padrao.ItemPilhaBooleano;
import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.MaquinaVirtual;
import UFSC.MaquinaVirtual.TipoIncorretoException;

public class CMAIMATR extends Instrucao {

    public CMAIMATR() {
        super.opcode = 0x0708;
        super.mnemonico = "cmaimatr";
    }

    @Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        // verifica os tipos de dados no topo da pilha antes de operar
        ItemPilha item1 = mv.pilha.get(mv.pilha.topo-1);
        ItemPilha item2 = mv.pilha.get(mv.pilha.topo);

        if(!(item1 instanceof ItemPilhaMatricula) || !(item2 instanceof
ItemPilhaMatricula)) {
            throw new TipoIncorretoException("Instrução CMAIMATR. Tipos
incorretos no topo da pilha. Encontrado: "+item1.getTipo()+",
"+item2.getTipo()+". Esperado: matricula, matricula");
        }

        String matr1 = mv.literais.get(
((ItemPilhaMatricula)item1).getValor() ).toString();
```



```

        String matr2 = mv.literais.get(
((ItemPilhaMatricula)item2).getValor() ).toString();

        // A00000AA

        char fl_matr1 = matr1.charAt(0);
        char fl_matr2 = matr2.charAt(0);

        String l2l_matr1 = matr1.substring(6);
        String l2l_matr2 = matr2.substring(6);

        int matr1_no = Integer.parseInt(matr1.substring(1, 6));
        int matr2_no = Integer.parseInt(matr2.substring(1, 6));

        boolean ret = false;

        if(fl_matr1 >= fl_matr2 &&
            l2l_matr1.compareTo(l2l_matr2) >= 0 &&
            matr1_no >= matr2_no) {
            ret = true;
        }

        mv.pilha.set(mv.pilha.topo-1, new ItemPilhaBooleano(ret));

        mv.pilha.remove(mv.pilha.topo);
        mv.pilha.topo--;
        mv.PC++;
    }
}

```

## CMDFMATR.java

```

package UFSC.MaquinaVirtual.Extensoes.TipoMatricula;

import UFSC.MaquinaVirtual.*;
import UFSC.MaquinaVirtual.Extensoes.Padrao.*;

public class CMDFMATR extends Instrucao {

    public CMDFMATR() {
        super.opcode = 0x0706;
        super.mnemonico = "cmdfmatr";
    }

    @Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        // verifica os tipos de dados no topo da pilha antes de operar
        ItemPilha item1 = mv.pilha.get(mv.pilha.topo-1);
        ItemPilha item2 = mv.pilha.get(mv.pilha.topo);

        if(!(item1 instanceof ItemPilhaMatricula) || !(item2 instanceof
ItemPilhaMatricula)) {
            throw new TipoIncorretoException("Instrução CMDFMATR. Tipos
incorretos no topo da pilha. Encontrado: "+item1.getTipo()+",
"+item2.getTipo()+". Esperado: matricula, matricula");
        }

        String s1 = mv.literais.get( ((ItemPilhaMatricula)item1).getValor()
).toString();

```

```

        String s2 = mv.literais.get( ((ItemPilhaMatricula)item2).getValor()
).toString();

        if(!s1.equals(s2)) {
            mv.pilha.set(mv.pilha.topo-1, new ItemPilhaBooleano(true));
        } else {
            mv.pilha.set(mv.pilha.topo-1, new ItemPilhaBooleano(false));
        }

        mv.pilha.remove(mv.pilha.topo);
        mv.pilha.topo--;
        mv.PC++;
    }
}

```

## CMEIMATR.java

```

package UFSC.MaquinaVirtual.Extensoes.TipoMatricula;

import UFSC.MaquinaVirtual.*;
import UFSC.MaquinaVirtual.Extensoes.Padrao.*;

public class CMEIMATR extends Instrucao {

    public CMEIMATR() {
        super.opcode = 0x0710;
        super.mnemonico = "cmeimatr";
    }

    @Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        // verifica os tipos de dados no topo da pilha antes de operar
        ItemPilha item1 = mv.pilha.get(mv.pilha.topo-1);
        ItemPilha item2 = mv.pilha.get(mv.pilha.topo);

        if(!(item1 instanceof ItemPilhaMatricula) || !(item2 instanceof
ItemPilhaMatricula)) {
            throw new TipoIncorretoException("Instrução CMEIMATR. Tipos
incorretos no topo da pilha. Encontrado: "+item1.getTipo()+",
"+item2.getTipo()+". Esperado: matricula, matricula");
        }

        String matr1 = mv.literais.get(
((ItemPilhaMatricula)item1).getValor() ).toString();
        String matr2 = mv.literais.get(
((ItemPilhaMatricula)item2).getValor() ).toString();

        // A00000AA

        char fl_matr1 = matr1.charAt(0);
        char fl_matr2 = matr2.charAt(0);

        String l2l_matr1 = matr1.substring(6);
        String l2l_matr2 = matr2.substring(6);

        int matr1_no = Integer.parseInt(matr1.substring(1, 6));
        int matr2_no = Integer.parseInt(matr2.substring(1, 6));

        boolean ret = false;
    }
}

```

```

        if(f1_matr1 <= f1_matr2 &&
           121_matr1.compareTo(121_matr2) <= 0 &&
           matr1_no <= matr2_no) {
            ret = true;
        }

        mv.pilha.set(mv.pilha.topo-1, new ItemPilhaBooleano(ret));

        mv.pilha.remove(mv.pilha.topo);
        mv.pilha.topo--;
        mv.PC++;
    }
}

```

## CMIGMATR.java

```

package UFSC.MaquinaVirtual.Extensoes.TipoMatricula;

import UFSC.MaquinaVirtual.*;
import UFSC.MaquinaVirtual.Extensoes.Padrao.*;

public class CMIGMATR extends Instrucao {

    public CMIGMATR() {
        super.opcode = 0x0705;
        super.mnemonico = "cmigmatr";
    }

    @Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        // verifica os tipos de dados no topo da pilha antes de operar
        ItemPilha item1 = mv.pilha.get(mv.pilha.topo-1);
        ItemPilha item2 = mv.pilha.get(mv.pilha.topo);

        if(!(item1 instanceof ItemPilhaMatricula) || !(item2 instanceof
ItemPilhaMatricula)) {
            throw new TipoIncorretoException("Instrução CMIGMATR. Tipos
incorretos no topo da pilha. Encontrado: "+item1.getTipo()+",
"+item2.getTipo()+". Esperado: matricula, matricula");
        }

        String s1 = mv.literais.get( ((ItemPilhaMatricula)item1).getValor()
).toString();
        String s2 = mv.literais.get( ((ItemPilhaMatricula)item2).getValor()
).toString();

        if(s1.equals(s2)) {
            mv.pilha.set(mv.pilha.topo-1, new ItemPilhaBooleano(true));
        } else {
            mv.pilha.set(mv.pilha.topo-1, new ItemPilhaBooleano(false));
        }

        mv.pilha.remove(mv.pilha.topo);
        mv.pilha.topo--;
        mv.PC++;
    }
}

```

## CMMAMATR.java

```
package UFSC.MaquinaVirtual.Extensoes.TipoMatricula;

import UFSC.MaquinaVirtual.*;
import UFSC.MaquinaVirtual.Extensoes.Padrao.*;

public class CMMAMATR extends Instrucao {

    public CMMAMATR() {
        super.opcode = 0x0707;
        super.mnemonico = "cmmamatr";
    }

    @Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        // verifica os tipos de dados no topo da pilha antes de operar
        ItemPilha item1 = mv.pilha.get(mv.pilha.topo-1);
        ItemPilha item2 = mv.pilha.get(mv.pilha.topo);

        if(!(item1 instanceof ItemPilhaMatricula) || !(item2 instanceof
ItemPilhaMatricula)) {
            throw new TipoIncorretoException("Instrução CMMAMATR. Tipos
incorretos no topo da pilha. Encontrado: "+item1.getTipo()+",
"+item2.getTipo()+". Esperado: matricula, matricula");
        }

        String matr1 = mv.literais.get(
((ItemPilhaMatricula)item1).getValor() ).toString();
        String matr2 = mv.literais.get(
((ItemPilhaMatricula)item2).getValor() ).toString();

        // A00000AA

        char f1_matr1 = matr1.charAt(0);
        char f1_matr2 = matr2.charAt(0);

        String l2l_matr1 = matr1.substring(6);
        String l2l_matr2 = matr2.substring(6);

        int matr1_no = Integer.parseInt(matr1.substring(1, 6));
        int matr2_no = Integer.parseInt(matr2.substring(1, 6));

        boolean ret = false;

        if(f1_matr1 >= f1_matr2 &&
l2l_matr1.compareTo(l2l_matr2) >= 0 &&
matr1_no > matr2_no) {
            ret = true;
        }

        mv.pilha.set(mv.pilha.topo-1, new ItemPilhaBooleano(ret));

        mv.pilha.remove(mv.pilha.topo);
        mv.pilha.topo--;
        mv.PC++;
    }
}
```

## CMMEMATR.java

```
package UFSC.MaquinaVirtual.Extensoes.TipoMatricula;

import UFSC.MaquinaVirtual.*;
import UFSC.MaquinaVirtual.Extensoes.Padrao.*;

public class CMMEMATR extends Instrucao {

    public CMMEMATR() {
        super.opcode = 0x0709;
        super.mnemonico = "cmmematr";
    }

    @Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        // verifica os tipos de dados no topo da pilha antes de operar
        ItemPilha item1 = mv.pilha.get(mv.pilha.topo-1);
        ItemPilha item2 = mv.pilha.get(mv.pilha.topo);

        if(!(item1 instanceof ItemPilhaMatricula) || !(item2 instanceof
ItemPilhaMatricula)) {
            throw new TipoIncorretoException("Instrução CMMEMATR. Tipos
incorretos no topo da pilha. Encontrado: "+item1.getTipo()+",
"+item2.getTipo()+". Esperado: matricula, matricula");
        }

        String matr1 = mv.literais.get(
((ItemPilhaMatricula)item1).getValor() ).toString();
        String matr2 = mv.literais.get(
((ItemPilhaMatricula)item2).getValor() ).toString();

        // A00000AA

        char f1_matr1 = matr1.charAt(0);
        char f1_matr2 = matr2.charAt(0);

        String l2l_matr1 = matr1.substring(6);
        String l2l_matr2 = matr2.substring(6);

        int matr1_no = Integer.parseInt(matr1.substring(1, 6));
        int matr2_no = Integer.parseInt(matr2.substring(1, 6));

        boolean ret = false;

        if(f1_matr1 <= f1_matr2 &&
l2l_matr1.compareTo(l2l_matr2) <= 0 &&
matr1_no < matr2_no) {
            ret = true;
        }

        mv.pilha.set(mv.pilha.topo-1, new ItemPilhaBooleano(ret));

        mv.pilha.remove(mv.pilha.topo);
        mv.pilha.topo--;
        mv.PC++;
    }
}
```

## CRMATR.java

```
package UFSC.MaquinaVirtual.Extensoes.TipoMatricula;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.MaquinaVirtual;
import UFSC.MaquinaVirtual.TipoIncorretoException;

public class CRMATR extends Instrucao {

    public CRMATR() {
        super.opcode = 0x0703;
        super.mnemonico = "crmatr";
    }

    @Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        ItemPilhaMatricula matr = new ItemPilhaMatricula( this.param2 );

        mv.pilha.add(matr);
        mv.pilha.topo++;
        mv.PC++;
    }
}
```

## LEIAMATR.java

```
package UFSC.MaquinaVirtual.Extensoes.TipoMatricula;

import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.MaquinaVirtual;

public class LEIAMATR extends Instrucao {

    public LEIAMATR() {
        super.mnemonico = "leiamatr";
        super.opcode = 0x0702;
    }

    @Override
    public void executar(MaquinaVirtual mv) {

        String input = "";

        while(true) {
            input = mv.getController().read();
            if(!input.matches("[A-Z]{1}[0-9]{5}[A-Z]{2}$")) {
                mv.getController().write("Erro: a matrícula digitada não está
no formato correto (A0000AA).\n");
            } else { break; }
        }

        LiteralMatricula literal = new LiteralMatricula(input);
        mv.literais.add(literal);
        int indice = mv.literais.size() - 1;

        mv.pilha.add(new ItemPilhaMatricula(indice));
        mv.pilha.topo++;
    }
}
```

```

        mv.PC++;
    }
}

```

### MATR2CAD.java

```

package UFSC.MaquinaVirtual.Extensoes.TipoMatricula;

import UFSC.MaquinaVirtual.Extensoes.Padrao.ItemPilhaCadeia;
import UFSC.MaquinaVirtual.Extensoes.Padrao.LiteralCadeia;
import UFSC.MaquinaVirtual.Instrucao;
import UFSC.MaquinaVirtual.ItemPilha;
import UFSC.MaquinaVirtual.MaquinaVirtual;
import UFSC.MaquinaVirtual.TipoIncorretoException;

public class MATR2CAD extends Instrucao {

    public MATR2CAD() {
        super.opcode = 0x0704;
        super.mnemonico = "matr2cad";
    }

    @Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException {
        // verifica os tipos de dados no topo da pilha antes de operar
        ItemPilha item1 = mv.pilha.get(mv.pilha.topo);

        if(!(item1 instanceof ItemPilhaMatricula)) {
            throw new TipoIncorretoException("Instrução MATR2CAD. Tipo
incorreto no topo da pilha. Encontrado: "+item1.getTipo()+". Esperado:
matricula");
        }

        String value =
mv.literais.get(((ItemPilhaMatricula)item1).getValor()).toString();

        LiteralCadeia novacadeia = new LiteralCadeia( value );

        mv.literais.add(novacadeia);
        int indice = mv.literais.size()-1;

        mv.pilha.set(mv.pilha.topo, new ItemPilhaCadeia(indice));

        mv.PC++;
    }
}

```

### ItemPilhaMatricula.java

```

package UFSC.MaquinaVirtual.Extensoes.TipoMatricula;

import UFSC.MaquinaVirtual.ItemPilha;

public class ItemPilhaMatricula extends ItemPilha {

    // o construtor sem parâmetros DEVE estar implementado, mesmo
    // que não possua nenhum comando pois é necessário para
    // a instanciação dinâmica dessa classe
    public ItemPilhaMatricula() {

```

```

        super.tipo_literal = true;
    }

    public ItemPilhaMatricula(int indice) {
        super(indice);
        super.tipo_literal = true;
    }

    @Override
    public Integer getValor() {
        return (Integer)super.valor;
    }

    @Override
    public String getTipo() {
        return "matricula";
    }

    @Override
    public int comparar(ItemPilha item) {
        throw new UnsupportedOperationException("Método não implementado!");
    }
}

```

### LiteralMatricula.java

```

package UFSC.MaquinaVirtual.Extensoes.TipoMatricula;

import UFSC.MaquinaVirtual.Literal;
import java.nio.charset.Charset;

/**
 * Representa uma matrícula na tabela de literais
 * @author LucasMS
 */
public class LiteralMatricula extends Literal {

    // necessário para instanciação dinâmica
    public LiteralMatricula() {
        // configurar para qualquer valor, que identifica
        // unicamente cada tipo literal. Esse é o valor que deve ser usado
para
        // identificar o tipo na tabela de constantes literais em arquivos
        // executáveis da MV-LSI.

        super.tipo = 0x22;
    }

    public LiteralMatricula(int size, byte[] value, boolean literal) {
        super((byte)0x22, size, value, literal);
    }

    public LiteralMatricula(String value) {
        super((byte)0x22, value.length(), value.getBytes());
    }

    @Override
    public String toString() {
        return new String(this.getValor(), Charset.forName("UTF-8"));
    }
}

```



```
    }  
    @Override  
    public String getTipoString() {  
        return "matricula";  
    }  
}
```

## Apêndice F - Código fonte da extensão para Arrays Bidimensionais

### AVET2.java

```
package UFSC.MaquinaVirtual.Extensoes.ArrayBidimensional;
import UFSC.MaquinaVirtual.Extensoes.Padrao.ItemPilhaInteiro;
import UFSC.MaquinaVirtual.*;

public class AVET2 extends Instrucao {

    public AVET2() {
        super.opcode = 0x0801;
        super.mnemonico = "AVET2";
    }

    @Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException,
IndiceInvalidoException {

        ItemPilha item_fonte = mv.pilha.get(mv.pilha.topo);

        // verifica se os tamanhos da primeira e segunda dimensões são
válidos
        int arr_pos = mv.pilha.getBase(this.param1) + this.param2;
        ItemPilha arr_size1 = mv.pilha.get(arr_pos);
        ItemPilha arr_size2 = mv.pilha.get(arr_pos+1);
        if(!(arr_size1 instanceof ItemPilhaInteiro) || !(arr_size2 instanceof
ItemPilhaInteiro)) {
            throw new TipoIncorretoException("Instrução AVET2. Tipos
incorretos na pilha na posição "+arr_pos+", que deveriam indicar o tamanho do
array. Encontrado: "+arr_size1.getTipo()+", "+arr_size2.getTipo()+".
Esperado: inteiro, inteiro");
        }

        ItemPilha item_indice1 = mv.pilha.get(mv.pilha.topo-2);
        ItemPilha item_indice2 = mv.pilha.get(mv.pilha.topo-1);
        if(!(item_indice1 instanceof ItemPilhaInteiro) || !(item_indice2
instanceof ItemPilhaInteiro)) {
            throw new TipoIncorretoException("Instrução AVET2. Tipos
incorretos na pilha em topo-2 e topo-1. Encontrado:
"+item_indice1.getTipo()+", "+item_indice2.getTipo()+". Esperado: inteiro,
inteiro");
        }

        // verifica se o índice está dentro dos valores permitidos
        int val_indice1 = (Integer)item_indice1.getValor();
        int val_indice2 = (Integer)item_indice2.getValor();
        if(val_indice1 < 0 || val_indice1 > ((Integer)arr_size1.getValor()-1)
||
        val_indice2 < 0 || val_indice2 > ((Integer)arr_size2.getValor()-
1)) {
            throw new IndiceInvalidoException("Instrução AVET2. Índice
inválido: ["+val_indice1+"]["+val_indice2+"]");
        }

        // verifica se o item de destino é compatível com o item que será
armazenado
```

```

        int tam2adimensao = ((ItemPilhaInteiro)arr_size2).getValor();
        ItemPilha item_destino = mv.pilha.get(mv.pilha.getBase(this.param1) +
this.param2 + 2 + (val_indice1 * tam2adimensao) + val_indice2);
        if(!(item_fonte.getClass() == item_destino.getClass())) {
            throw new TipoIncorretoException("Instrução AVET2. Tipo do
elemento de destino é diferente do novo valor atribuído. Encontrado:
"+item_destino.getTipo()+". Esperado: "+item_fonte.getTipo());
        }

        mv.pilha.set(mv.pilha.getBase(this.param1) + this.param2 + 2 +
(val_indice1 * tam2adimensao) + val_indice2, item_fonte);

        mv.pilha.remove(mv.pilha.topo);
        mv.pilha.topo--;
        mv.pilha.remove(mv.pilha.topo);
        mv.pilha.topo--;
        mv.pilha.remove(mv.pilha.topo);
        mv.pilha.topo--;

        mv.PC++;
    }
}

```

## CVET2.java

```

package UFSC.MaquinaVirtual.Extensoes.ArrayBidimensional;
import UFSC.MaquinaVirtual.Extensoes.Padrao.ItemPilhaInteiro;
import UFSC.MaquinaVirtual.*;

public class CVET2 extends Instrucao {

    public CVET2() {
        super.opcode = 0x0802;
        super.mnemonico = "CVET2";
    }

    @Override
    public void executar(MaquinaVirtual mv) throws TipoIncorretoException,
IndiceInvalidoException {
        // verifica se os índices no topo da pilha são inteiro
        ItemPilha indice1 = mv.pilha.get(mv.pilha.topo-1);
        ItemPilha indice2 = mv.pilha.get(mv.pilha.topo);
        if(!(indice1 instanceof ItemPilhaInteiro) || !(indice2 instanceof
ItemPilhaInteiro)) {
            throw new TipoIncorretoException("Instrução CVET2. Tipo incorreto
no topo da pilha. Encontrado: "+indice1.getTipo()+", "+indice2.getTipo()+".
Esperado: inteiro, inteiro");
        }

        int arr_pos = mv.pilha.getBase(this.param1) + param2;
        ItemPilha arr_size1 = mv.pilha.get(arr_pos);
        ItemPilha arr_size2 = mv.pilha.get(arr_pos+1);
        if(!(arr_size1 instanceof ItemPilhaInteiro) || !(arr_size2 instanceof
ItemPilhaInteiro)) {
            throw new TipoIncorretoException("Instrução CVET2. Tipos
incorretos na pilha na posição "+arr_pos+", que deveriam indicar o tamanho do
array. Encontrado: "+arr_size1.getTipo()+", "+arr_size2.getTipo()+".
Esperado: inteiro, inteiro");
        }
    }
}

```

```

// verifica se os índices estão dentro dos valores permitidos
int val_indice1 = (Integer)indice1.getValor();
int val_indice2 = (Integer)indice2.getValor();
if(val_indice1 < 0 || val_indice1 > ((Integer)arr_size1.getValor()-1)
||
    val_indice2 < 0 || val_indice2 > ((Integer)arr_size2.getValor()-
1)) {
    throw new IndiceInvalidoException("Instrução CVET2. Índice
inválido: ["+val_indice1+"]["+val_indice2+"]");
}

int tam2adimensao = ((ItemPilhaInteiro)arr_size2).getValor();
ItemPilha valor = mv.pilha.get(mv.pilha.getBase(this.param1) +
this.param2 + 2 + (val_indice1 * tam2adimensao) + val_indice2);

mv.pilha.set(mv.pilha.topo-1, valor);

mv.pilha.remove(mv.pilha.topo);
mv.pilha.topo--;

mv.PC++;
}
}

```