Gustavo P. Gouveia (6482819), Victor Lassance (6431325)

Relatório de Compiladores
Quarta Etapa
Especificação do Ambiente de Execução
Linguagem de programação CZAR

Texto apresentado à Escola Politécnica da Universidade de São Paulo como requisito para a aprovação na disciplina Linguagens e Compiladores no quinto módulo acadêmico do curso de graduação em Engenharia de Computação, junto ao Departamento de Engenharia de Computação e Sistemas Digitais (PCS).

Universidade de São Paulo

Escola Politécnica

Engenharia de Computação - Curso Cooperativo

Professor: Ricardo Luis de Azevedo da Rocha

São Paulo

2013

Sumário

1 Introdução

Até a terceira entrega do compilador, focamos nas duas primeiras etapas da compilação de um programa, a análise léxica e a análise sintática. Para essa entrega, focaremos no ambiente de execução. O compilador por nós criado terá como linguagem de saída um programa que será executado na máquina virtual chamada Máquina de von Neumann (MVN).

O Modelo de von Neumann procura oferecer uma alternativa prática, disponibilizando ações mais poderosas e ágeis em seu repertório de operações que o modelo de Turing. Isso viabiliza, codificações muito mais expressivas, compactas e eficientes. Para isso, a Máquina de von Neumann utiliza:

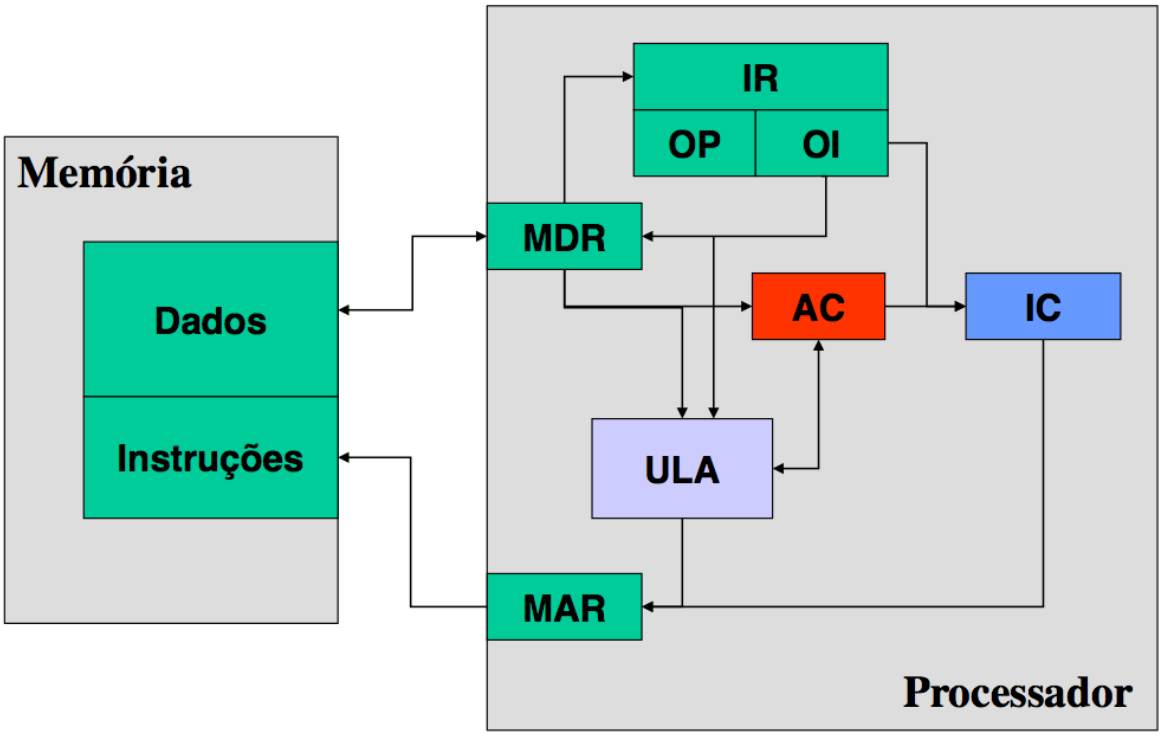
- Memória endereçável, usando acesso aleatório
- Programa armazenado na memória, para definir diretamente a função corrente da máquina (ao invés da Máquina de Estados Finitos)
- Dados representados na memória (ao invés da fita)
- Codificação numérica binária em lugar da unária
- Instruções variadas e expressivas para a realização de operações básicas muito frequentes (ao invés de sub-máquinas específicas)
- Maior flexibilidade para o usuário, permitindo operações de entrada e saída, comunicação física com o mundo real e controle dos modos de operação da máquina

Dessa forma, utilizaremos essa máquina para executar nosso compilador e realizar os testes necessários.

A arquitetura de Von Neumann é composta por um processador e uma memória principal. Na memória principal armazenam-se as instruções do código-fonte e os dados, sendo a divisão mostrada na figura 1 apenas ilustrativa. Além da Unidade Lógica Aritmética (ULA), responsável pelo processamento de operações lógicas e aritméticas, o processador possui um conjunto de elementos físicos de armazenamento de informações e é comum dividir esses componentes nos seguintes módulos resgistradores:

1. MAR - Registrador de endereço de memória

Figura 1 – Arquitetura MVN



Indica qual é a origem ou o destino, na memória principal, dos dados contidos no registrador de dados de memória.

2. MDR - Registrador de dados da memória

   Serve como ponte para os dados que trafegam entre a memória e os outros elementos da máquina.

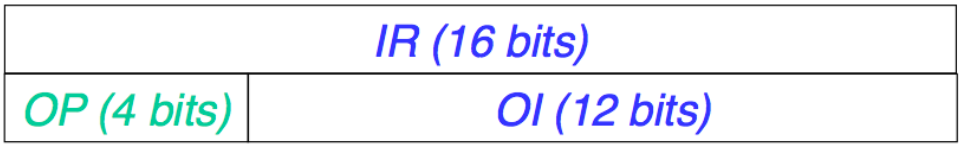
3. IC - Registrador de endereço de instrução

   Indica a cada instante qual será a próxima instrução a ser executada pelo processador.

4. IR - Registrador de instrução

   Contém a instrução atual a ser executada. é subdividido em dois outros registradores, como na figura 2.

Figura 2 – Estrutura do registro de instrução (IR)



a) OP - Registrador de código de operação

Parte do registrador de instrução que identifica a instrução que está sendo executada.

b) OI - Registrador de operando de instrução

Complementa a instrução indicando o dado ou o endereço sobre o qual ela deve agir.

5. AC - Acumulador

Funciona como a área de trabalho para execução de operações lógicas ou aritméticas. Acumula o resultado de tais operações.

A máquina executa um programa em diversos passos, listadas abaixo:

1. Determinação da Próxima Instrução a Executar

2. Fase de Obtenção da Instrução

Obter na memória, no endereço contido no registrador de Endereço da Próxima Instrução, o código da instrução desejada.

3. Fase de Decodificação da Instrução

Decompor a instrução em duas partes: o código da instrução e o seu operando, depositando essas partes nos registradores de instrução e de operando, respectivamente. Selecionar, com base no conteúdo do registrador de instrução, um procedimento de execução dentre os disponíveis no repertório do simulador (passo 4).

4. Fase de Execução da Instrução

Executar o procedimento selecionado no passo 3, usando como operando o conteúdo do registrador de operando, preenchido anteriormente.

Caso a instrução executada não seja de desvio, incrementar o registrador de endereço da próxima instrução a executar. Caso contrário, o procedimento de execução já terá atualizado convenientemente tal informação.

a) Execução da instrução (decodificada no passo 3)

De acordo com o código da instrução a executar (contido no registrador de instrução), executar os procedimentos de simulação correspondentes (detalhados adiante).

b) Acerto do registrador de Endereço da Próxima Instrução para apontar a próxima instrução a ser simulada:

Incrementar o registrador de Endereço da Próxima Instrução.

2 Instruções da Linguagem de Saída

As instruções da MVN podem ser resumidas pela tabela da figura 3.

Figura 3 – Lista de instruções da MVN

Código (hexa)	Instrução	Operando
0	Desvio incondicional	endereço do desvio
1	Desvio se acumulador é zero	endereço do desvio
2	Desvio se acumulador é negativo	endereço do desvio
3	Deposita uma constante no acumulador	constante relativa de 12 bits
4	Soma	endereço da parcela
5	Subtração	endereço do subtraendo
6	Multiplicação	endereço do multiplicador
7	Divisão	endereço do divisor
8	Memória para acumulador	endereço-origem do dado
9	Acumulador para memória	endereço-destino do dado
A	Desvio para subprograma (função)	endereço do subprograma
B	Retorno de subprograma (função)	endereço do resultado
C	Parada	endereço do desvio
D	Entrada	dispositivo de e/s
E	Saída	dispositivo de e/s
F	Chamada de supervisor	constante (**)

(**) por ora, este operando (tipo da chamada) é irrelevante, e esta instrução nada faz.

A seguir, especificaremos o que é realizado pela máquina ao executar cada tipo de operação.

- Registrador de instrução = 0 (desvio incondicional)

  Modifica o conteúdo do registrador de Endereço da Próxima Instrução (IC) armazenando nele o conteúdo do registrador de operando (OI)

  IC := OI

- Registrador de instrução = 1 (desvio se acumulador é zero)

  Se o conteúdo do acumulador (AC) for zero, então modifica o conteúdo do registrador de Endereço da Próxima Instrução (IC), armazenando nele o conteúdo do registrador de operando (OI)

  Se AC = 0 então IC := OI

Se não IC := IC + 1

- Registrador de instrução = 2 (desvio se negativo)

  Se o conteúdo do acumulador (AC) for negativo, isto é, se o bit mais significativo for 1, então modifica o conteúdo do registrador de Endereço da Próxima Instrução (IC) armazenando nele o conteúdo do registrador de operando (OI)

  Se AC < 0 então IC := OI

  Se não IC := IC + 1

- Registrador de instrução = 3 (constante para acumulador)

  Armazena no acumulador (AC) o número relativo de 12 bits contido no registrador de operando (OI), estendendo seu bit mais significativo (bit de sinal) para completar os 16 bits do acumulador

  AC := OI

  IC := IC +1

- Registrador de instrução = 4 (soma)

  Soma ao conteúdo do acumulador (AC) o conteúdo da posição de memória indicada pelo registrador de operando MEM[OI]. Guarda o resultado no acumulador

  AC := AC + MEM[OI]

  IC := IC + 1

- Registrador de instrução = 5 (subtração)

  Subtrai do conteúdo do acumulador (AC) o conteúdo da posição de memória indicada pelo registrador de operando MEM[OI]. Guarda o resultado no acumulador

  AC := AC - MEM[OI]

  IC := IC + 1

- Registrador de instrução = 6 (multiplicação)

  Multiplica o conteúdo do acumulador (AC) pelo conteúdo da posição de memória indicada pelo registrador de operando MEM[OI]. Guarda o resultado no acumulador

  AC := AC * MEM[OI]

  IC := IC + 1

- Registrador de instrução = 7 (divisão inteira)

  Dividir o conteúdo do acumulador (AC) pelo conteúdo da posição de memória indicada pelo registrador de operando MEM[OI]. Guarda a parte inteira do resultado no acumulador

AC := int (AC / MEM[OI])

IC := IC + 1

- Registrador de instrução = 8 (memória para acumulador)

  Armazena no acumulador (AC) o conteúdo da posição de memória endereçada pelom registrador de operando (OI)

  AC := MEM[OI]

  IC := IC + 1

- Registrador de instrução = 9 (acumulador para memória)

  Guarda o conteúdo do acumulador (AC) na posição de memória endereçada pelo registrador de operando (OI)

  MEM[OI] := AC

  IC := IC + 1

- Registrador de instrução = A (desvio para subprograma)

  Armazena o conteúdo do registrador de Endereço da Próxima Instrução (IC), incrementado de uma unidade, no registrador de endereço de retorno (RA). Armazena no registrador de Endereço da Próxima Instrução (IC) o conteúdo do registrador de operando (OI).

  RA := IC + 1

  IC := OI

- Registrador de instrução = B (retorno de subprograma)

  Armazena no registrador de Endereço da Próxima Instrução (IC) o conteúdo do registrador de endereço de retorno (RA), e no acumulador (AC) o conteúdo da posição de memória apontada pelo registrador de operando (OI)

  AC := MEM[OI]

  IC := RA

- Registrador de instrução = C (stop)

  Modifica o conteúdo do registrador de Endereço da Próxima Instrução (IC) armazenando nele o conteúdo do registrador de operando (OI) e para o processamento

  IC := OI

- Registrador de instrução = D (input)

  Aciona o dispositivo padrão de entrada e aguardar que o usuário forneça o próximo dado a ser lido. Transfere o dado para o acumulador

Aguarda

AC := dado de entrada

IC := IC + 1

- Registrador de instrução = E (output)

  Transfere o conteúdo do acumulador (AC) para o dispositivo padrão de saída. Aciona o dispositivo padrão de saída e aguardar que este termine de executar a operação de saída

  dado de saída := AC

  aguarda

  IC := IC + 1

- Registrador de instrução = F (supervisor call)

  Não implementado: por enquanto esta instrução não faz nada.

  IC := IC + 1

Escrever um programa usando diretamente codificação binária não é uma tarefa simples, e tampouco agradável. Naturalmente, se um programa é muito grande ou se lida com diversas estruturas complexas (listas, etc.), a sua codificação se torna ainda mais difícil e complexa.

Por conta disso, torna-se imprescindível construir alguma abstração que facilite a programação e a verificação dos programas. A primeira idéia, mais natural, é utilizar o modelo de máquina existente e, a partir dele, definir nomes (mnemônicos) para cada instrução da máquina. Posteriormente, verifica-se que somente isso não basta, pois é necessário lidar com os endereços dentro de um programa (rótulos, operandos, sub-rotinas), com a reserva de espaço para tabelas, com valores constantes. Enfim, é necessário definir uma linguagem simbólica.

Para a construção de um montador, cujo esquema geral está representado na figura 4 pressupõe-se que sejam tratadas as seguintes questões:

- definição das instruções: determinar os mnemônicos que as representam simbolicamente;
- definição das pseudo-instruções: determinar os mnemônicos que as representam, bem como sua função para o montador.

As instruções para a MVN são apresentadas na figura 5.

Figura 4 – Esquema geral de um montador

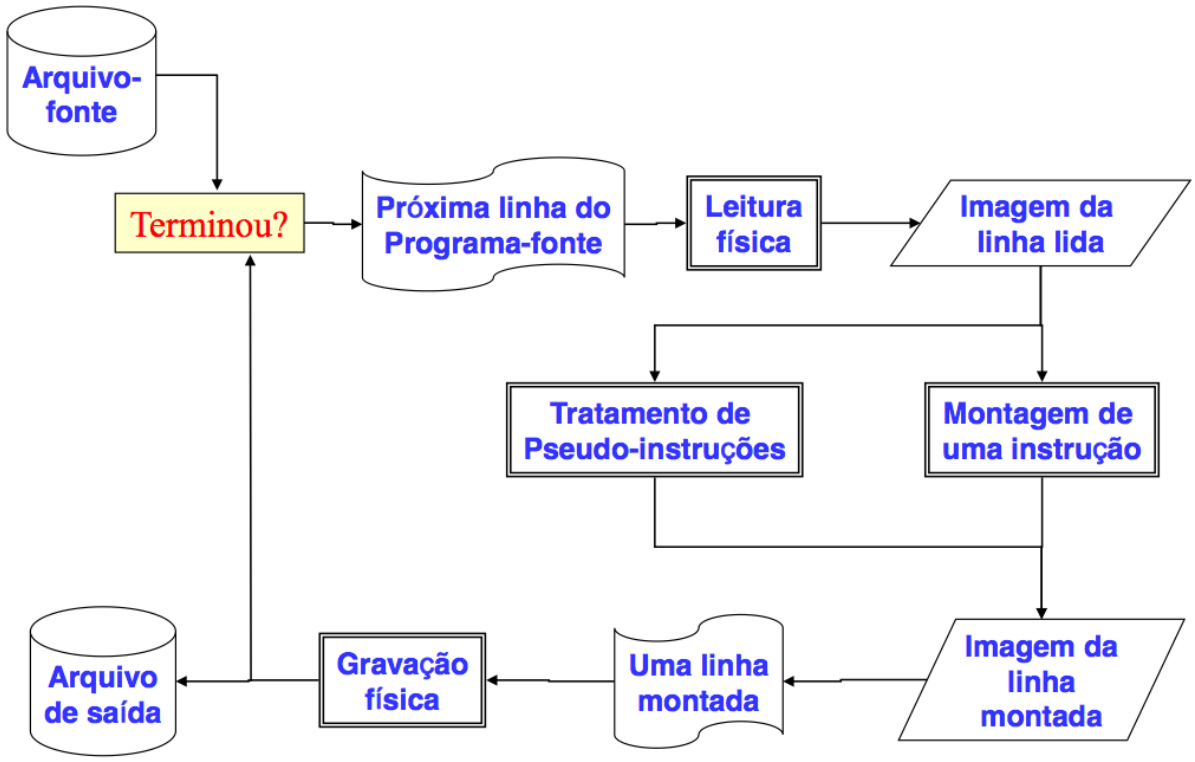


Figura 5 – Tabela de mnemônicos para a MVN (de 2 caracteres)

Operação **0** **Jump** Mnemônico **JP**	Operação **1** **Jump if Zero** Mnemônico **JZ**	Operação **2** **Jump if Negative** Mnemônico **JN**	Operação **3** **Load Value** Mnemônico **LV**
Operação **4** **Add** Mnemônico **+**	Operação **5** **Subtract** Mnemônico **–**	Operação **6** **Multiply** Mnemônico *****	Operação **7** **Divide** Mnemônico **/**
Operação **8** **Load** Mnemônico **LD**	Operação **9** Move to **Memory** Mnemônico **MM**	Operação **A** **Subroutine Call** Mnemônico **SC**	Operação **B** **Return from Sub.** Mnemônico **RS**
Operação **C** **Halt Machine** Mnemônico **HM**	Operação **D** **Get Data** Mnemônico **GD**	Operação **E** **Put Data** Mnemônico **PD**	Operação **F** **Operating System** Mnemônico **OS**

3 Pseudoinstruções da Linguagem de Saída

Programas absolutos são executáveis estritamente nas posições de memória em que foram criados, tornando difícil a manutenção e o trabalho em equipe. A utilização de programas relocáveis permitem sua execução em qualquer posição de memória, tornando possível utilizar partes de código projetadas externamente (uso de bibliotecas, por exemplo).

Para que se possa exprimir um programa relocável e com possibilidade de construção em módulos, separadamente desenvolvidos, é necessário que:

- Haja a possibilidade de representar e identificar endereços absolutos e endereços relativos;
- Um programa possa ser montado sem que os seus endereços simbólicos estejam todos resolvidos;
- Seja possível identificar, em um módulo, símbolos que possam ser referenciados simbolicamente em outros módulos.

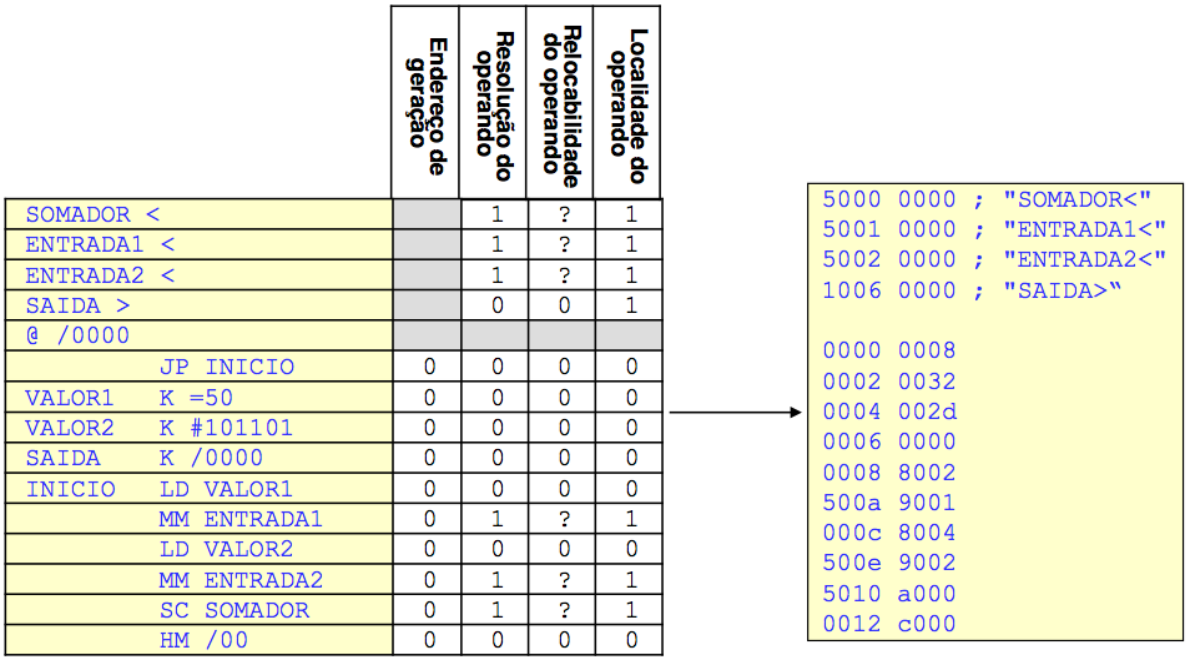
Sendo assim, a linguagem simbólica não possui somente os mnemônicos das instruções da MVN, mas também comandos chamados de pseudo-instruções da linguagem de montagem. Na linguagem de montagem, as pseudo-instruções também são representadas por mnemônicos, listados abaixo:

- @ : Origem Absoluta. Recebe um operando numérico, define o endereço da instrução seguinte;
- K : Constante, o operando numérico tem o valor da constante (em hexadecimal). Define uma área preenchida por uma CONSTANTE de 2 bytes;
- $: Reserva de área de dados, o operando numérico define o tamanho da área a ser reservada. Define um BLOCO DE MEMóRIA com número especificado de words;
- # : Final físico do texto fonte;
- & : Origem relocável;
- > : Endereço simbólico de entrada (entry point). Define um endereço simbólico local como entry-point do programa;

- < : Endereço simbólico externo (external). Define um endereço simbólico que referencia um entry-point externo.

Na figura 6, temos um exemplo de um somador escrito em linguagem de montagem, visto na aula de Fundamentos de Eng. de Computação, e sua respectiva tradução pelos módulos Montador, *Linker* e Relocador, módulos extras porém integrados no nosso caso:

Figura 6 – Exemplo de um somador



	Endereço de geração	Resolução do operando	Relocabilidade do operando	Localidade do operando
SOMADOR <		1	?	1
ENTRADA1 <		1	?	1
ENTRADA2 <		1	?	1
SAIDA >		0	0	1
@ /0000				
JP INICIO	0	0	0	0
VALOR1 K =50	0	0	0	0
VALOR2 K #101101	0	0	0	0
SAIDA K /0000	0	0	0	0
INICIO LD VALOR1	0	0	0	0
MM ENTRADA1	0	1	?	1
LD VALOR2	0	0	0	0
MM ENTRADA2	0	1	?	1
SC SOMADOR	0	1	?	1
HM /00	0	0	0	0

```
5000 0000 ; "SOMADOR<"
5001 0000 ; "ENTRADA1<"
5002 0000 ; "ENTRADA2<"
1006 0000 ; "SAIDA>"

0000 0008
0002 0032
0004 002d
0006 0000
0008 8002
500a 9001
000c 8004
500e 9002
5010 a000
0012 c000
```

4 Características Gerais

4.1 Organização da memória

O ambiente de execução da MVN fornece aos programadores um total de 4Kb de memória para ser usado tanto para o código quanto para as variáveis do programa. Desses 4Kb, a parte inicial da memória é reservada para guardar as instruções que serão executadas pelo programa. A parte final da memória deve ser usada especialmente para o uso do registro de ativação.

4.2 Registro de ativação

O registro de ativação nesse ambiente de execução será feito sob a forma de uma pilha, onde a cada chamada de função todos os dados do referentes a função, bem como o endereço de retorno, devem ser empilhados para serem usados. Os dados a serem empilhados no registro de ativação são:

- Endereço de retorno - Endereço do próximo endereço da pilha - Parâmetros da função - Variáveis locais das função

O endereço de retorno fica localizado no primeiro endereço do bloco empilhado no registro de ativação. O segundo endereço é referente ao endereço do primeiro endereço do próximo bloco do registro de ativação. Esse endereço é usado para mudar o valor ponteiro do registro de ativação, para que a função que chamou a outra possa voltar a enxergar suas variáveis. Do terceiro endereço em diante estão localizados os parâmetros da função. Após o final dos parâmetros, estão localizadas as variáveis locais necessárias para guardar executar as operações durante a execução da função.

Os endereços das variáveis locais e dos parâmetros podem ser calculados usando o ponteiro do registro de ativação, somando dois mais os tamanhos das variáveis existentes anteriormente.

A figura abaixo ilustra a organização da pilha de ativação.

registros-ativacao.png

O uso do registro de ativação permite entre outras coisas a chamada recursiva de funções, uma vez isso não é possível de forma nativa no ambiente da MVN. Com registro de ativação, realizar uma recursão significa empilhar um novo bloco à pilha e relançar a

execução da função.

5 Biblioteca Desenvolvida em Assembly

TODO Biblioteca Assembly