A Máquina de Pilha (MP) – Parte I

por Antonio Atta
atta@uneb.br

A Máquina de Pilha – MP é uma das formas mais antigas, fáceis e acessíveis de se estruturar sistemas de execução (*run time systems*) em compiladores de linguagens de programação imperativas que seguem o modelo de C ou Pascal, considerando suas construções de alto nível (semântica de expressões e atribuições, mecanismos de chamadas e retornos de procedimentos, passagem de parâmetros, etc). Sistemas de execução que processam as ações ou comandos do programa sobre estruturas hipotéticas como a MP, também conhecidas como Máquinas Virtuais, são mais flexíveis e facilitam o processo de tradução dos compiladores se comparadas à tradução feita diretamente para a arquitetura (processador) alvo. Todo compilador, na sua fase de síntese, gera código para um sistema de execução específico e, posteriormente, esse código é traduzido de maneira muito mais simplificada e direta (quase que numa relação de um comando do sistema de execução para um ou poucos comandos) para o processador da arquitetura alvo. Essa abordagem permite a construção de compiladores para diferentes processadores, bastando para isso alterar o módulo do *back-end* do compilador que traduz comandos do sistema de execução (código intermediário) para cada processador específico – uma tarefa infinitamente mais simples que reescrever todo o compilador!

O código intermediário gerado para sistemas de execução, no entanto, não é dos mais eficientes, mas é flexível e claro o suficiente para permitir aplicações didáticas da tradução dos diversos tipos de comandos da classe de linguagens citadas. A principal estrutura do sistema de execução que apresentamos agora, a MP, consiste em utilizar uma pilha de dados que será responsável pelo armazenamento simultâneo da área de dados globais (variáveis globais) e dados que ocupam a pilha do sistema (variáveis locais e parâmetros de procedimentos, por exemplo) do programa traduzido. A seguir estão indicadas essa e as demais estruturas da MP e, em seguida, o seu conjunto de comandos (que são gerados no processo de tradução pelo compilador).

Em um outro texto veremos como projetar Esquemas de Tradução para os comandos de linguagens de alto nível usando a MP.

Estrutura de Dados da MP

A Figura 1 abaixo esquematiza as áreas e os componentes da MP. Ela é constituída por duas áreas de memória separadas, uma para a PILHA e outra para o CÓDIGO do programa. Considere que essas áreas são posições contíguas de memórias. O conteúdo da área de CÓDIGO (P no diagrama) são sempre os comandos da MP relativos ao programa compilado; o conteúdo da área de PILHA (M no diagrama) são as posições de dados das variáveis globais e locais declaradas no programa fonte e que serão manipuladas pelas instruções da MP presentes (carregadas) na área de CÓDIGO. O controle da posição de referência na área de CÓDIGO (posição da instrução a ser executada) é indicado pelo registrador I; e o topo da PILHA pelo registrador S. O dado no topo da PILHA é usado como valor que será desempilhado e processado por muitos dos comandos da MP na sua execução. A MP possui ainda dois registradores de base D[0] e D[1] que são usados para

apontar para os registros de ativação (ou janela de ativação)[1] dos procedimentos em execução. Um registro para cada escopo. Veremos sua utilização prática mais adiante juntamente com os comandos da MP que manipulam procedimentos.

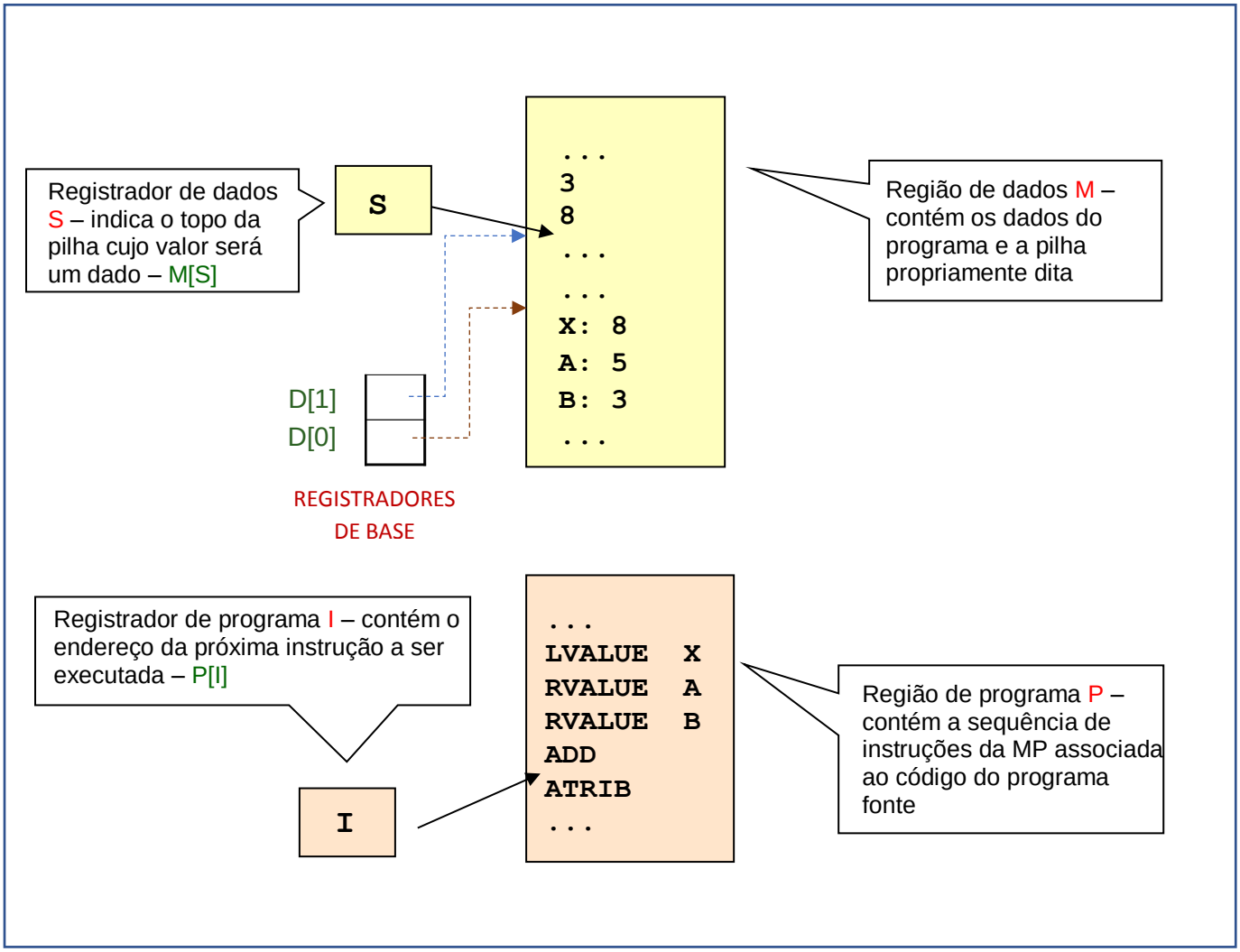

Figura 1 - Estrutura e esquema da MP

Algoritmo de execução da MP

A sequencia de ações a seguir corresponde ao algoritmo que é executado pela MP do momento em que a execução do código se inicia até o momento que uma instrução de parada de execução é encontrada:

1. ***Carregar o programa na região P;***
2. ***Inicializar os registradores I e S;***
3. ***Para cada instrução da MP:***
    a. ***Executar a instrução;***

[1] Recordando, ***registros de ativação*** são áreas contíguas de memória na PILHA do sistema de execução, usadas para guardar informações imprescindíveis para a execução dos procedimentos, a exemplo de valores e endereços dos parâmetros, endereço de retorno do próprio procedimento, alocação de variáveis locais, etc.

b. ***Incrementar o registrador I (no caso de instrução de desvio, atualizar I para apontar para a instrução para onde a execução deve ser desviada);***

Comados básicos da MP

A seguir encontram-se os comandos básicos da MP que permitem construir código para a manipulação de dados, operações aritméticas e de controle de fluxo.

INSTRUÇÕES DE MANIPULAÇÃO DE DADOS

PUSH v	- Empilha a constante v no topo da pilha
POP	- Desempilha o valor no topo da pilha
COPY	- Duplica o valor do topo da pilha
STOR x	- Desempilha e armazena o conteúdo do topo da pilha na área de dados x
LOAD y	- Empilha o valor da área de dados y

INSTRUÇÕES ARITMÉTICAS COM VALORES INTEIROS

ADD - Desempilha topo e sub-topo, soma e empilha resultado
SUB - Desempilha topo e sub-topo, subtrai (sub-topo – topo) e empilha resultado
MUL - Desempilha topo e sub-topo, multiplica e empilha resultado
DIV - Desempilha topo e sub-topo, divide (sub-topo / topo) e empilha resultado

INSTRUÇÕES ARITMÉTICAS COM VALORES REAIS

ADDF - Desempilha topo e sub-topo, soma e empilha o resultado
SUBF - Desempilha topo e sub-topo, subtrai (sub-topo – topo) e empilha o resultado
MULF - Desempilha topo e sub-topo, multiplica e empilha o resultado
DIVF - Desempilha topo e sub-topo, divide (sub-topo / topo) e empilha o resultado

INSTRUÇÕES DE CONTROLE DE FLUXO

LABEL L	- Rotula a próxima instrução com rótulo L
GOTO L	- Desvia para a instrução rotulada com rótulo L
GOFALSE L	- Desempilha e desvia o fluxo de execução para a instrução rotulada L se o valor for **igual** a zero
GOTRUE L	- Desempilha e desvia o fluxo de execução para a instrução rotulada L se o valor for **maior** que zero

INSTRUÇÃO DE FIM DE PROGRAMA

HALT - Pára a execução do programa

INSTRUÇÕES DE ENTRADA/SAÍDA (E/S)

GET - Lê um valor inteiro da entrada padrão e coloca no topo da pilha
GETF - Lê um valor real da entrada padrão e coloca no topo da pilha
PUT - Desempilha valor inteiro e escreve na saída padrão
PUTF - Desempilha valor real e escreve na saída padrão

Apenas com o conjunto básicos de comandos da MP acima é possível traduzir qualquer sequência de comandos contendo operações de atribuição, comandos de seleção ou comandos de laço. Apenas as chamadas de procedimento e alocação de memória para variáveis não têm suporte a partir desses comandos da MP. Para comprovar essa afirmação e ajudar a compreender melhor o funcionamento da MP e seu potencial para uso como sistema de execução, vejamos exemplos de traduções possíveis a partir de diversos tipos de comandos que encontramos em uma linguagem como C – como ainda não foi mostrado a forma com que a alocação de memória para variáveis é feita na MP, por simplificação, usaremos o próprio nome das variáveis nos comandos da MP que fazem acesso à memória para buscar ou armazenar o conteúdo em uma variável; no futuro esses nomes serão substituídos por endereços relativos na região M da PILHA.

COMANDO DE ATRIBUIÇÃO COM EXPRESSÃO

O comando em linguagem C:

```
delta = b*b - 4*a*c;
```

Gera a seguinte possível sequência de comandos da MP quando traduzido por um programa compilador. A coluna semântica da execução do comando na MP indica o que ocorre internamente na MP durante a execução do respectivo comando:

COMANDOS DA MP	SEMÂNTICA DA EXECUÇÃO DO COMANDO NA MP
LOAD b	S=S+1; M[S]=b
LOAD b	S=S+1; M[S]=b
MUL	M[S-1]=M[S-1]*M[S]; S=S-1
PUSH 4	S=S+1; M[S]=4
LOAD a	S=S+1; M[S]=a
MUL	M[S-1=M[S-1]*M[S]; S=S-1
LOAD c	S=S+1; M[S]=c
MUL	M[S-1]=M[S-1]*M[S]; S=S-1
STOR delta	delta=M[S]; S=S-1;

COMANDO DE SELEÇÃO SIMPLES

O comando em linguagem C:

```
if (<EXPR_COND>) <CMD>
```

Gera a seguinte possível sequência de comandos da MP quando traduzido por um programa compilador. Como no item anterior, a coluna semântica da execução do comando na MP indica o que ocorre internamente na MP durante a execução do respectivo comando. Para garantir que os comandos da MP relativos ao bloco de comandos internos (representado por <CMD>) só sejam executados se a expressão condicional (<EXPR_COND>) for verdadeira, fica definido que a execução dos comandos da MP gerados para <EXPR_COND> deve "deixar" no topo da PILHA

da MP um valor igual a 0 (zero), se a expressão condicional for falsa, e 1 (um) se ela for verdadeira – ver geração de código da MP para expressões condicionais mais adiante neste texto:

COMANDOS DA MP	SEMÂNTICA DA EXECUÇÃO DO COMANDO NA MP
Comandos da MP de <EXPR_COND>	*Deixam no topo da PILHA o valor 0 ou 1*
GOFALSE Lx	S=S-1; Se M[S+1] igual a 0 vá para Lx (I=instrução rotulada com Lx)
Comandos da MP de <CMD>	*Sequência de comandos da MP relativos à execução de <CMD>*
LABEL Lx	*Não executa nada e passa para a próxima instrução*

COMANDO DE SELEÇÃO COMPLETA

O comando em linguagem C:

```
if (<EXPR_COND>) <CMD1> else <CMD2>
```

Gera a seguinte possível sequência de comandos da MP quando traduzido por um programa compilador. Como no item anterior, para garantir que os comandos da MP relativos ao bloco de comandos internos da seleção principal (representado por <CMD$_1$>) só sejam executados se a expressão condicional (<EXPR_COND>) for verdadeira, fica definido que a execução dos comandos da MP gerados para <EXPR_COND> deve "deixar" no topo da PILHA da MP um valor igual a 0 (zero), se a expressão condicional for falsa, e 1 (um) se ela for verdadeira:

COMANDOS DA MP	SEMÂNTICA DA EXECUÇÃO DO COMANDO NA MP
Comandos da MP de <EXPR_COND>	*Comandos da MP gerados pela compilação de <EXPR_COND>. Deixam no topo da PILHA o valor 0 ou 1*
GOFALSE Lx	S=S-1; Se M[S+1] igual a 0 vá para Lx (I=instrução rotulada com Lx)
Comandos da MP de <CMD$_1$>	*Sequência de comandos da MP relativos à compilação de <CMD$_1$>*
GOTO Ly	*Salta o bloco de comandos <CMD$_2$>* (I=instrução rotulada com Ly)
LABEL Lx	*Não executa nada e passa para a próxima instrução*
Comandos da MP de <CMD$_2$>	*Sequência de comandos da MP relativos à compilação de <CMD$_2$>*
LABEL Ly	*Não executa nada e passa para a próxima instrução*

COMANDO DE LAÇO "FAÇA ENQUANTO"

O comando em linguagem C:

```
while (<EXPR_COND>) <CMD>
```

Gera a seguinte possível sequência de comandos da MP quando traduzido por um programa compilador. Como definido nos comandos de seleção anteriores, para garantir que os comandos da MP relativos ao bloco de comandos internos do laço (representado por <CMD>) só sejam executados enquanto a expressão condicional (<EXPR_COND>) for verdadeira, fica definido que a execução dos comandos da MP gerados para <EXPR_COND> deve "deixar" no topo da PILHA da MP um valor igual a 0 (zero), se a expressão condicional for falsa, e 1 (um) se ela for verdadeira:

COMANDOS DA MP	SEMÂNTICA DA EXECUÇÃO DO COMANDO NA MP
LABEL Lx	*Não executa nada e passa para a próxima instrução*
Comandos da MP de <EXPR_COND>	*Comandos da MP gerados pela compilação de <EXPR_COND>. Deixam no topo da PILHA o valor 0 ou 1*
GOFALSE Ly	S=S-1; Se M[S+1] igual a 0 vá para Ly (I=instrução rotulada com Ly)
Comandos da MP de <CMD>	*Sequência de comandos da MP relativos à compilação de <CMD>*
GOTO Lx	*Desvia para o teste da condição* (I=instrução rotulada com Lx)
LABEL Ly	*Não executa nada e passa para a próxima instrução*

COMANDO DE LAÇO "FAÇA PARA"

O comando em linguagem C:

```
for (<EXPR_INIC> ; <EXPR_COND> ; <EXPR_INCR>) <CMD>
```

Gera a seguinte possível sequência de comandos da MP quando traduzido por um programa compilador. Como definido nos comandos de seleção e laço anteriores, para garantir que os comandos da MP relativos ao bloco de comandos internos do laço (representado por <CMD>) só sejam executados enquanto a expressão condicional (<EXPR_COND>) for verdadeira, fica definido que a execução dos comandos da MP gerados para <EXPR_COND> deve "deixar" no topo da PILHA da MP um valor igual a 0 (zero), se a expressão condicional for falsa, e 1 (um) se ela for verdadeira. Para esse tipo de comando temos ainda uma situação especial: o compilador passará por <EXPR_INCR> antes de passar por <CMD>; portanto, o código da MP de <EXPR_INCR> será gerado antes do código de <CMD> mas semanticamente o bloco de comandos relativos a <EXPR_INCR> só deve ser executado após a execução do bloco de comandos de <CMD>. Um jogo com os comandos de desvio incondicional (GOTO) e rótulos (LABEL) da MP resolve a situação e os códigos da MP relativos a <EXPR_INIC>, <EXPR_COND>, <EXPR_INCR> e <CMD> são gerados

na ordem em que eles ocorrem no processo de compilação, mas a execução dos mesmos ocorre em ordem distinta.

COMANDOS DA MP	SEMÂNTICA DA EXECUÇÃO DO COMANDO NA MP
Comandos da MP de <EXPR_INIC>	*Comandos da MP gerados pela compilação de <EXPR_INIC>.*
LABEL Lw	*Não executa nada e passa para a próxima instrução*
Comandos da MP de <EXPR_COND>	*Comandos da MP gerados pela compilação de <EXPR_COND>. Deixam no topo da PILHA o valor 0 ou 1*
GOFALSE Lx	S=S-1; Se M[S+1] igual a 0 vá para Lx (I=instrução rotulada com Lx)
GOTO Ly	S=S-1; Se M[S+1] igual a 0 vá para Ly (I=instrução rotulada com Ly)
LABEL Lz	
Comandos da MP de <EXPR_INCR>	*Sequência de comandos da MP relativos à compilação de <CMD>*
GOTO Lw	*Desvia para o teste da condição* (I=instrução rotulada com Lx)
LABEL Ly	*Não executa nada e passa para a próxima instrução*
Comandos da MP de <CMD>	*Sequência de comandos da MP relativos à compilação de <CMD>*
GOTO Lz	*Desvia para o bloco de comandos de <EXPR_INCR>*
LABEL Lx	*Não executa nada e passa para a próxima instrução*

Código da MP gerado para expressões condicionais simples

Como visto, as expressões condicionais, aquelas que aplicam um operador relacional (do tipo <, >, <=, >=, == ou != na linguagem C), ocorrem em comando de seleção e de laço e podem aparecer do lado direito em operações de atribuição também. Em todos esses casos um valor lógico (booleano) deve resultar da execução do código gerado por esse tipo de expressão. Na MP esse valor corresponde a 0 (zero), se a expressão for falsa, ou a 1 (um), se for verdadeira, e deve ser armazenado no topo da PILHA para uso pelas instruções seguintes que testam a condição em comandos como os de seleção ou laço condicional vistos.

Vejamos então como gerar código compilado para expressões simples mantendo essa característica em todas as situações de operadores relacionais. Mostraremos exemplos de códigos para alguns operadores relacionais e deixaremos o restante como exercício. Todos os exemplos partem de uma operação de subtração dos operandos da expressão e avaliação do resultado no topo da PILHA para em seguida descobrir se deve deixar 0 (falso) ou 1(verdadeiro) no topo da pilha. A solução é igualmente válida se do lado direito e/ou do lado esquerdo da expressão condicional existir uma expressão aritmética, como em "A+B == C–D", já que nos dois casos o resultado de cada expressão aritmética ficaria no topo da PILHA e sendo passível de comparação.

Deve ficar registrado que os exemplos abaixo são possibilidades válidas, mas não únicas, de geração de código da MP por um compilador para expressões condicionais. Procurou-se priorizar a clareza do esquema de tradução relacionado à expressão condicional que se está compilando – não o seu desempenho na execução.

CÓDIGO DA MP PARA EXPRESSÕES CONDICIONAIS DO TIPO "A == B"

COMANDOS DA MP	SEMÂNTICA DA EXECUÇÃO DO COMANDO NA MP
LOAD A	S=S+1; M[S]=A
LOAD B	S=S+1; M[S]=B
SUB	M[S-1]=M[S-1] - M[S]; S=S-1
GOFALSE Lx	S=S-1; Se M[S+1] igual a 0 vá para Lx (I=instrução rotulada com Lx)
PUSH 0	*Se o desvio anterior foi feito, os valores de A e B são iguais; senão então a expressão condicional é falsa* S=S+1; M[S]=0
GOTO Ly	*Desvia para o fim da sequência de comandos da expressão condicional*
LABEL Lx	*Não executa nada e passa para a próxima instrução*
PUSH 1	*Valores de A e B iguais* S=S+1; M[S]=1
LABEL Ly	*Não executa nada e passa para a próxima instrução*

Note como ao final da sequência de comandos acima o valor 0 ou o valor 1 fica no topo da PILHA a depender do resultado da comparação da expressão condicional. Fica como exercício o esquema de tradução para "A != B".

CÓDIGO DA MP PARA EXPRESSÕES CONDICIONAIS DO TIPO "A > B"

COMANDOS DA MP	SEMÂNTICA DA EXECUÇÃO DO COMANDO NA MP
LOAD A	S=S+1; M[S]=A
LOAD B	S=S+1; M[S]=B
SUB	M[S-1]=M[S-1] - M[S]; S=S-1
GOTRUE Lx	S=S-1; Se M[S+1] maior que 0 vá para Lx (I=instrução rotulada com Lx)
PUSH 0	*Se o desvio anterior foi feito, o valor de A > B; senão então a expressão condicional é falsa (e A <= B)* S=S+1; M[S]=0
GOTO Ly	*Desvia para o fim da sequência de comandos da expressão condicional*
LABEL Lx	*Não executa nada e passa para a próxima instrução*
PUSH 1	*Valor de A > B* S=S+1; M[S]=1
LABEL Ly	*Não executa nada e passa para a próxima instrução*

Fica como exercício o esquema de tradução para “A <= B”.

CÓDIGO DA MP PARA EXPRESSÕES CONDICIONAIS DO TIPO “A < B”

COMANDOS DA MP	SEMÂNTICA DA EXECUÇÃO DO COMANDO NA MP
LOAD A	S=S+1; M[S]=A
LOAD B	S=S+1; M[S]=B
SUB	M[S-1]=M[S-1] - M[S]; S=S-1
COPY	*Duplica o valor no topo da pilha pois será necessário um duplo teste* S=S+1; M[S]=M[S-1];
GOFALSE Lx	*Se A for igual a B o resultado é falso*
GOTRUE Ly	*Se não, se A > B o resultado também é falso*
PUSH 1	*A não é igual e nem é maior que B, o resultado é verdadeiro* S=S+1; M[S]=1
GOTO Lz	*Desvia para o fim da sequência de comandos da expressão condicional*
LABEL Lx	*Não executa nada e passa para a próxima instrução*
POP	*Retira da pilha o valor duplicado que não será mais necessário*
LABEL Ly	*Não executa nada e passa para a próxima instrução*
PUSH 0	*A >= B, o resultado é falso* S=S+1; M[S]=0
LABEL Lz	*Não executa nada e passa para a próxima instrução*

Fica como exercício o esquema de tradução para “A >= B”.

Código da MP gerado para expressões condicionais compostas

Expressões condicionais podem ainda ser compostas por conjuntos de expressões condicionais simples ligadas por operadores relacionais como em “((A > B || C <D) && (!(X==Y)))” na linguagem C. Nesses casos, a solução na geração de código da MP durante a compilação permanece similar às expressões condicionais simples ficando um 0 (zero) ou 1(um) no topo da PILHA caso a expressão condicional composta resulte em falsa ou verdadeira, respectivamente. O que muda é o tratamento que é dado às expressões condicionais simples a depender do operado lógico de ligação que é usado. Veja abaixo como podemos tratar cada caso:

1. **Se o operador lógico for o de “negação” (!)**: avalia-se, como antes, a expressão condicional associada ao operador de negação e, ao final, inverte-se o valor no topo da PILHA;
2. **Se o operador lógico for o “ou” (||)**: em uma sequência de expressões condicionais simples ligadas por operadores lógicos “ou” basta que uma das expressões condicionais simples seja verdadeira para que todo o conjunto seja verdadeiro, dispensando a avaliação de todas as expressões condicionais simples do conjunto; se a avaliação tiver

que prosseguir até a última expressão condicional simples da sequência (todas as anteriores resultaram em falso), o resultado lógico dessa última expressão será o resultado do conjunto;

3. **Se o operador lógico for o "e" (&&)**: em uma sequência de expressões condicionais simples ligadas por operadores lógicos "e" basta que uma das expressões condicionais simples seja falsa para que todo o conjunto resulte em falso, dispensando a avaliação de todas as expressões condicionais simples do conjunto; se a avaliação tiver que prosseguir até a última expressão condicional simples da sequência(todas as anteriores resultaram em verdadeiro), o resultado lógico dessa última expressão será o resultado do conjunto;

CÓDIGO DA MP PARA EXPRESSÕES CONDICIONAIS LÓGICAS DO TIPO "! <EXPR_COND>"

COMANDOS DA MP	SEMÂNTICA DA EXECUÇÃO DO COMANDO NA MP
Comandos da MP de <EXPR_COND>	*Comandos da MP gerados pela compilação de <EXPR_COND>. Deixam no topo da pilha o valor 0 ou 1*
GOFALSE Lx	*Desvia se topo da pilha contém 0 (zero)* S=S-1; Se M[S+1] igual a 0 vá para Lx (I=instrução rotulada com Lx)
PUSH 0	*Se não desviou no comando anterior é porque o topo da pilha continha 1(um) - inverte* S=S+1; M[S]=0
GOTO Ly	*Desvia para o final da sequência de comandos* (I=instrução rotulada com Ly)
LABEL Lx	*Não executa nada e passa para a próxima instrução*
PUSH 1	*O topo da pilha continha 0(zero) - inverte* S=S+1; M[S]=1
LABEL Ly	*Não executa nada e passa para a próxima instrução*

CÓDIGO DA MP PARA EXPRESSÕES CONDICIONAIS LÓGICAS DO TIPO "<EXPR_COND$_1$> || "<EXPR_COND$_2$>" || ... || "<EXPR_COND$_n$> "

COMANDOS DA MP	SEMÂNTICA DA EXECUÇÃO DO COMANDO NA MP
Comandos da MP de <EXPR_COND$_1$>	*Comandos da MP gerados pela compilação de <EXPR_COND$_1$>. Deixam no topo da pilha o valor 0 ou 1.*
COPY	*Duplica valor no topo da pilha* S=S+1; M[S]=M[S-1];
GOTRUE Lx	*Desvia se topo da pilha contém 1 (um)* S=S-1; Se M[S+1] igual a 1 vá para Lx (I=instrução rotulada com Lx)
POP	*Desempilha valor duplicado pois não será mais necessário* S=S-1

Comandos da MP de <EXPR_COND$_2$>	*Comandos da MP gerados pela compilação de <EXPR_COND$_2$>. Deixam no topo da pilha o valor 0 ou 1.*
COPY	*Duplica valor no topo da pilha* S=S+1; M[S]=M[S-1];
GOTRUE Lx	*Desvia se topo da pilha contém 1 (um)* S=S-1; Se M[S+1] igual a 1 vá para Lx (I=instrução rotulada com Lx)
POP	*Desempilha valor duplicado pois não será mais necessário* S=S-1
...	
Comandos da MP de <EXPR_COND$_n$>	*Se chegou até aqui, o resultado da avaliação dessa expressão condicional é o que ficará no topo da pilha como resultado da avaliação do conjunto de expressões condicionais ligadas pelo "ou lógico"*
LABEL Lx	*Não executa nada e passa para a próxima instrução*

Fica como exercício o esquema de tradução para expressões condicionais compostas do tipo "<EXPR_COND$_1$> && "<EXPR_COND$_2$>" && ... && "<EXPR_COND$_n$> " que usam o operador lógico "e" como conectivo.

Juntando tudo

Vejamos agora como usar os conhecimentos dos esquemas de tradução acima, na geração de código intermediário compilado de um trecho de programa que contém vários dos comando de linguagem de alto nível trabalhados.

Considere como referência o seguinte trecho de código escrito em linguagem C:

```
...
x = 55;    y = 5;   aux = x;
if  (y==0)   z = -1;
else  {
   z = 0;
   while (aux >= y)   {
        aux = aux - y;
        z = z + 1;
   }
}
quoc = z;
resto = aux;
...
```

Segue o código compilado correspondente gerado para a MP usando os esquemas de tradução apresentados neste texto. Os realces nos comandos identificam a tradução dos comandos do código fonte com realce similar:

INICIO DO CÓDIGO DA MP	CONTINUAÇÃO ①	CONTINUAÇÃO ②
. . .	STOR z	LABEL L8
PUSH 55	GOTO L4	GOFALSE L9
STOR x	LABEL L3	LOAD aux
PUSH 5	PUSH 0	LOAD y
STOR y	STOR z	SUB
LOAD x	LABEL L5	STOR aux
STOR aux	LOAD aux	LOAD z
LOAD y	LOAD y	PUSH 1
PUSH 0	SUB	ADD
SUB	COPY	STOR z
GOFALSE L1	GOFALSE L6	GOTO L5
PUSH 0	GOTRUE L7	LABEL L9
GOTO L2	PUSH 0	LABEL L4
LABEL L1	GOTO L8	LOAD z
PUSH 1	LABEL L6	STOR quoc
LABEL L2	POP	LOAD aux
GOFALSE L3	LABEL L7	STOR resto
PUSH -1	PUSH 1	. . .

Considerações Finais

Este texto serve de material de estudo introdutório para a compreensão de sistemas de execução baseados em Maquinas de Pilha. Ele apresenta também como esquemas de tradução de comando básicos de linguagens de programação imperativas, como a linguagem C, podem ser concebidos visando a geração de código para a MP por um processo de tradução dirigida pela sintaxe. Com base no material apresentado é possível, portanto, desenvolver esquemas de tradução e incorporá-los a analisadores sintáticos descendente recursivos - ASDR. Para isso, o projeto do esquema de tradução a partir da gramática da linguagem fonte deve prever os pontos de inserção das emissões de código da MP durante o processo de montagem da árvore sintática (etapa conhecida como geração de código intermediário).

O texto corresponde à primeira parte da apresentação da MP, já que ele não apresenta como a MP suporta a alocação de espaço de memória para as variáveis (neste texto usamos o próprio nome da variável nos comandos da MP quando estes comandos deveriam fazer referência a endereços de memória) e também o suporte às chamadas e retornos de procedimentos, incluindo a passagem de parâmetros. Estes serão temas da Parte II deste material.