Escola Politécnica da Universidade de São Paulo

Linguagens e Compiladores

5ª parte do projeto – Tradução dos comandos

Henrique Sussumu Matsui Kano
Mi Che Li Lee

30 de novembro de 2015

Sumário

Capítulo 1

Tradução de comandos

1.1 Controle de Fluxo

1.1.1 If-then

Linguagem de Entrada

```
if (<expressao>) do
      <comandos>
endif
```

Linguagem de Saída

```
if_1  <expressao>
            JN          endif_1
      JP      do_if_1
do_if_1 <comandos>
endif_1     <comandos-2>
```

1.1.2 If-do-elsif-do-else

Linguagem de Entrada

```
if (<expressao-1>) do
      <comandos-1>
elsif (<expressao-2>) do
      <comandos-2>
else do
      <comandos-3>
endif
```

Linguagem de Saída

```
if_1          <expressao-1>
              JN       elsif_1_1
              JP       do_if_1
do_if_1       <comandos-1>
              JP       endif_1
elsif_1_1     <expressao-2>
              JN       else_1
              JP       do_elsif_1_1
do_elsif_1_1  <comandos-2>
              JP       endif_1
else_1              <comandos-3>
              JP       endif_1
endif_1       <comandos-4>
```

1.1.3 While

Linguagem de Entrada

```
while (<expressao>) do
      <comandos>
endwhile
```

Linguagem de saída

```
while_1       <expressao>
                    JN          endwhile_1
                    JP          do_while_1
do_while_1    <comandos>
              JP          while_1
endwhile_1    <comandos-2>
```

1.2 Comandos Imperativos

1.2.1 Atribuição de valor

Linguagem de Entrada

```
a = <expressao>;
```

Linguagem de saída

```
<expressao>
MM a;
```

1.2.2 Leitura (entrada)

Para fazer a entrada de dados são feitas várias leituras de words interpretadas de dois caracteres ASCII, mesmo para números.

Linguagem de Entrada

```
int a;
scan(a);
```

Linguagem de saída

```
READ_BUFFER_LIMIT K           /0
READ_COUNT        K           /0
READ_DATA         K           /0
READ_BASE         K           /0

READ              K           /0
                  MM          READ_BUFFER_LIMIT
                  LV          /0
                  MM          READ_COUNT
                  MM          READ_DATA


READ_DO           GD          /0000
                  MM          READ_DATA
                  -           ASCII_BR
                  JZ          READ_END
                  LD          READ_DATA
                  SC          BYTIFY
                  LD          BYTIFY_SECOND
                  -           ASCII_BELL
                  JZ          READ_FIRST
                  JP          READ_ALL

READ_FIRST        LD          READ_BUFFER_LIMIT
                  MM          WRITE_BASE
                  LD          BYTIFY_FIRST
                  MM          WRITE_DATA
                  LV          /2
                  *           READ_COUNT
                  SC          WRITE
                  LV          /1
                  +           READ_COUNT
```

```
                    MM          READ_COUNT
                    JP          READ_TEST_LIMIT

READ_ALL            LD          READ_BASE
                    MM          WRITE_BASE
                    LD          READ_DATA
                    MM          WRITE_DATA
                    LV          /2
                    *           READ_COUNT
                    SC          WRITE
                    LV          /1
                    +           READ_COUNT
                    MM          READ_COUNT
                    JP          READ_TEST_LIMIT

READ_TEST_LIMIT     LD          READ_COUNT
                    -           READ_BUFFER_LIMIT
                    JZ          READ_END
                    JP          READ_DO

READ_END            RS          READ
```

1.2.3 Impressão (saída)

A impressão tem o trabalho de formatar a informação e apresentá-lo na tela. Como na linguagem há strings de caractéres e numeros, foi concluído que será necessário pelo menos dois tipos de formatação: em ASCII e em uma base qualquer. No caso, serão utilizadas as formatações ASCII e em base 16 para os números com a adição do sinal de negativo quando o número for menor que zero.

Linguagem de Entrada

```
print(a);
```

Linguagem de saída

```
PRINT_BASE              K     /0
PRINT_TYPE              K     /0

PRINT_COUNT             k     /0
PRINT_SIZE              k     /0
PRINT_DATA              k     /0
PRINT_FIRST             k     /0
PRINT_SECOND            k     /0
PRINT_THIRD             k     /0
PRINT_FOURTH            k     /0
```

```
PRINT_OFFSET            k     /0

PRINT                   K     /0

                        MM    PRINT_SIZE

                        LV    /0
                        MM    PRINT_COUNT
                        MM    PRINT_OFFSET


PRINT_BEGIN             LD    PRINT_BASE
                        +     PRINT_OFFSET
                        +     LOAD_PREFIX
                        MM    PRINT_LOAD_DATA
PRINT_LOAD_DATA         K     /0
                        MM    PRINT_DATA

                        LD    PRINT_TYPE

                        JZ    PRINT_AS_IS
                        JP    PRINT_AS_ASCII


PRINT_AS_ASCII          LV    /1
                        +     PRINT_COUNT
                        MM    PRINT_COUNT
                        -     PRINT_SIZE
                        JZ    PRINT_AS_ASCII_ODD

                        LV    /1
                        +     PRINT_COUNT
                        MM    PRINT_COUNT
                        JP    PRINT_AS_ASCII_EVEN

PRINT_AS_ASCII_ODD      LD    PRINT_DATA                ;
   SEPARA OS BYTES
                        /     SHIFT2
                        *     SHIFT2
                        PD    /0100
                        JP    PRINT_END

PRINT_AS_ASCII_EVEN     LD    PRINT_DATA
                        PD    /0100

                        LV    /2
                        +     PRINT_OFFSET
                        MM    PRINT_OFFSET
```

```
                        LD    PRINT_COUNT
                        -     PRINT_SIZE
                        JZ    PRINT_END
                        JP    PRINT_BEGIN


PRINT_AS_IS             LD    PRINT_DATA
                        JN    PRINT_NEG
                        JP    PRINT_POS
PRINT_NEG               LD    ASCII_MINUS
                        PD    /0100
                        LD    PRINT_DATA
                        *     FFFF
                        MM    PRINT_DATA

PRINT_POS               /     SHIFT3            ;Separa todos
    os numeros
                        *     SHIFT3
                        MM    PRINT_FIRST

                        LD    PRINT_DATA
                        -     PRINT_FIRST
                        /     SHIFT2
                        *     SHIFT2
                        MM    PRINT_SECOND

                        LD    PRINT_DATA
                        -     PRINT_FIRST
                        -     PRINT_SECOND
                        /     SHIFT1
                        *     SHIFT1
                        MM    PRINT_THIRD

                        LD    PRINT_DATA
                        -     PRINT_FIRST
                        -     PRINT_SECOND
                        -     PRINT_THIRD
                        MM    PRINT_FOURTH          ;
                           separado (0x1234 viraria: 0x1000,
                           0x0200, 0x0030 e 0x0004)

                        LD    PRINT_FIRST
                        /     SHIFT3
                        MM    PRINT_FIRST
                        -     A
                        JN    PRINT_SUM_NUMBER_1
                        JP    PRINT_SUM_LETTER_1
PRINT_SUM_NUMBER_1      LD    PRINT_FIRST
                        +     ASCII_0
```

```
                        MM    PRINT_FIRST
                        JP    PRINT_AS_IS_1
PRINT_SUM_LETTER_1      LD    PRINT_FIRST
                        +     HEX_LETTER_TO_ASCII
                        MM    PRINT_FIRST
PRINT_AS_IS_1           LD    PRINT_SECOND
                        /     SHIFT2
                        MM    PRINT_SECOND
                        -     A
                        JN    PRINT_SUM_NUMBER_2
                        JP    PRINT_SUM_LETTER_2
PRINT_SUM_NUMBER_2      LD    PRINT_SECOND
                        +     ASCII_0
                        MM    PRINT_SECOND
                        JP    PRINT_AS_IS_2
PRINT_SUM_LETTER_2      LD    PRINT_SECOND
                        +     HEX_LETTER_TO_ASCII
                        MM    PRINT_SECOND
PRINT_AS_IS_2           LD    PRINT_FIRST
                        *     SHIFT2
                        +     PRINT_SECOND
                        PD    /0100

                        LD    PRINT_THIRD
                        /     SHIFT1
                        MM    PRINT_THIRD
                        -     A
                        JN    PRINT_SUM_NUMBER_3
                        JP    PRINT_SUM_LETTER_3
PRINT_SUM_NUMBER_3      LD    PRINT_THIRD
                        +     ASCII_0
                        MM    PRINT_THIRD
                        JP    PRINT_AS_IS_3
PRINT_SUM_LETTER_3      LD    PRINT_THIRD
                        +     HEX_LETTER_TO_ASCII
                        MM    PRINT_THIRD
PRINT_AS_IS_3           LD    PRINT_FOURTH
                        -     A
                        JN    PRINT_SUM_NUMBER_4
                        JP    PRINT_SUM_LETTER_4
PRINT_SUM_NUMBER_4      LD    PRINT_FOURTH
                        +     ASCII_0
                        MM    PRINT_FOURTH
                        JP    PRINT_AS_IS_4
PRINT_SUM_LETTER_4      LD    PRINT_FOURTH
                        +     HEX_LETTER_TO_ASCII
                        MM    PRINT_FOURTH
PRINT_AS_IS_4           LD    PRINT_THIRD
                        *     SHIFT2
```

```
                        +       PRINT_FOURTH
                        PD      /0100

                        LV      /1
                        +       PRINT_COUNT
                        MM      PRINT_COUNT
                        -       PRINT_SIZE
                        JZ      PRINT_END

                        LV      /2
                        +       PRINT_OFFSET
                        MM      PRINT_OFFSET

                        JP      PRINT_BEGIN


PRINT_END               RS      PRINT
```

1.2.4 Chamada de subrotina

A chamada de subrotina inclui os métodos do tratamento dos registros de ativação. Para cada registro de ativação, em ordem decrescente de endereços, temos: parâmetros da função chamada, endereço do último frame pointer (base pointer do registro de ativação anterior), endereço de retorno da função, resultado da função e variáveis locais e temporárias. O frame pointer sempre aponta para a posição do endereço de retorno da função no registro de ativação.

Linguagem de Entrada

```
begin do
      int param_1;
      int param_2;
      param_1 = 256;
      param_1 = 768;
      sub(param_1, param_2);
end
```

Linguagem de Saída

```
INIT        JP    MAIN
TWO         K     /0002
ZERO        K     /0000
LD_INSTR    LD    /0000       ; para fabricar instrucao de LD
MM_INSTR    MM    /0000       ; para fabricar instrucao de MM

PARAM_1     K     /0300       ; parametro 1 da funcao
PARAM_2     K     /0100       ; parametro 2 da funcao
```

```
FP          K     /0400       ; frame pointer
SP          K     /0400       ; stack pointer

; subrotina SUB ------------------------------------
F_PARAM_1   K     /0000
F_PARAM_2   K     /0000
F_RESULT    K     /0000
; corpo da subrotina
SUB         K     /0000
            LD    SUB    ; endereco de retorno
            MM    PUSH_P
            SC    PUSH_RA     ; push ADDR na pilha de ra

            LD    SP
            MM    FP          ; fp aponta para o ADDR na
               pilha de ra

            LV    =6
            MM    GET_P
            SC    GET_RA      ; carrega param 1 partir de
               endereco relativo no ra
            MM    F_PARAM_1

            LV    =4
            MM    GET_P
            SC    GET_RA      ; carrega param 2 a partir de
               endereco relativo no ra
            MM    F_PARAM_2

            LD    F_PARAM_1   ; carrega o primeiro param
            -     F_PARAM_2   ;
            MM    RTRN_RSLT   ; subtrai e guarda para return

            JP    RETURN

; fim SUB ------------------------------------------

; RETURN -------------------------------------
RTRN_RSLT   K     /0000
RTRN_ADDR   K     /0000
; corpo da subrotina
RETURN      LD    RTRN_RSLT
            MM    PUSH_P
            SC    PUSH_RA     ; push RESULT na pilha de ra

            LV    =0
            MM    GET_P
            SC    GET_RA
```

```
            MM    RTRN_ADDR   ; endereco de retorno
               recuperado da pilha

            LD    TWO
            MM    GET_P
            SC    GET_RA
            MM    SP          ; dropa a ra atual
            MM    FP          ; volta a fp antiga

            RS    RTRN_ADDR   ; retorna para o escopo
               anterior
; fim RETURN ----------------------------------------

; subrotina PUSH_RA ----------------------------------
PUSH_P      K     /0000
; corpo da subrotina
PUSH_RA     K     /0000
            LD    MM_INSTR    ; composicao do comando de
               carregar na pilha
            +     SP
            -     TWO
            MM    _NEW_INSTR1
            LD    PUSH_P
_NEW_INSTR1 K     /0000       ; carrega na pilha de fato
            LD    SP
            -     TWO
            MM    SP          ; sp, aponte para o vazio
            RS    PUSH_RA
; fim PUSH_RA ----------------------------------------

; subrotina GET_RA -----------------------------------
GET_P       K     /0000       ; endereco relativo na ra
; corpo da subrotina
GET_RA      K     /0000
            LD    LD_INSTR
            +     FP          ;
            +     GET_P       ;
            MM    _NEW_INSTR2 ;
_NEW_INSTR2 K     /0000       ; carrega valor da ra no
   acumulador
            RS    GET_RA
; fim GET_RA -----------------------------------------

; corpo do main --------------------------------------
MAIN        LD    PARAM_1
            MM    PUSH_P
            SC    PUSH_RA     ; push PARAM_1 na pilha de ra

            LD    PARAM_2
```

```
            MM    PUSH_P
            SC    PUSH_RA     ; push PARAM_2 na pilha de ra

            LD    FP
            MM    PUSH_P
            SC    PUSH_RA     ; push old FP na pilha de ra

            SC    SUB         ; chama SUB
;fim main ------------------------------------------------
```

1.3 Estruturas de Dados

1.3.1 Struct

A struct, sendo uma variável que possui outras variáveis em seu interior, será representado na mvn por espaços de memória concatenados, sendo que em seu cabeçalho haverá uma informação de quantos campos ele possui.

Linguagem de Entrada

```
struct tipoEsquisito do
    int structInt;
    bool structBool;
    string structString;
endstruct
```

Linguagem de saída

```
var_tipoEsquisito                          K     /3
var_tipoEsquisito.structInt   $     =2
var_tipoEsquisito.structBool  $     =2
var_tipoEsquisito.structBool  $     =100
```

1.3.2 Array

Será criado um bloco de memória com o número necessário de bytes mais dois, sendo que a primeira word irá conter quantas "casas"a array possui.

Linguagem de Entrada

```
int intArray[10];
```

Linguagem de saída

```
intArray          $           =22
```

1.4 Exemplo de programa traduzido

Linguagem de Entrada

```
begin do
      int fat;
      int a;
      scan(a);
      read(a);

      if(a < 0) do
            fat = 0;
      else do
            fat = 1;
            while(num > 1) do
                  fat = fat * a;
                  a = a - 1;
            endwhile
      endif

      print(a);
end
```

Linguagem de saída

```
INIT              JP          MAIN

A                 K           /0000
FAT               K           /0000
MAIN              SC          READ
                  SC          PRINT
if_1              LD          A
                  JN          else_1
                  JP          endif_1
do_if_1           LV          =0
                  MM          FAT
                  JP          endif_1
else_1            LV          =1
                  MM          FAT
while_1           LD          A
                  JN          endwhile_1
                  JP          do_while_1
do_while_1        LD          FAT
                  *           A
                  MM          FAT
                  LD          A
                  -           ONE
```

```
                    MM          A
                    JP          while_1
endwhile_1          JP          endif_1
endif_1             LD          FAT
                    SC          PRINT
END                 HM          INIT

# INIT
```

Referências Bibliográficas

[1] R. Ricardo S. Jaime A. Reginaldo, B. Anarosa. Introdução máquina de von neumann.

[2] João José Neto. *Introdução à Compilação*. Escola Politécnica da USP, 1 edition, 1986.