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

Relatório de Compiladores
Quinta Etapa
Tradução dos Comandos
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

Este projeto tem como objetivo a construção de um compilador de um só passo, dirigido por sintaxe, com analisador e reconhecedor sintático baseado em autômato de pilha estruturado.

Em um primeiro momento, foi definida uma linguagem de programação e identificados os tipos de átomos. Para cada átomo foi escrito uma gramática linear representativa da sua lei de formação e um reconhecedor para o átomo. Desse modo, as gramáticas assim escritas foram unidas e convertidas em um autômato finito, o qual foi transformado em um transdutor e implementado como sub-rotina, dando origem ao analisador léxico propriamente dito. Também foi criada uma função principal para chamar o analisador léxico e possibilitar o seu teste.

Durante a segunda etapa, a sintaxe da linguagem, denonimada por nós de CZAR, foi definida formalmente a partir de uma definição informal e de exemplos de programas que criamos, misturando palavras-chave e conceitos de diferentes linguagens de programação. As três principais definições foram escritas na notação BNF[1], Wirth[2] e com diagramas de sintaxe.

Na terceira etapa, implementamos o módulo referente à parte sintática para a nossa linguagem. O analisador sintático construído obtém uma cadeia de *tokens* proveniente do analisador léxico, e verifica se a mesma pode ser gerada pela gramática da linguagem e, com isso, constrói a árvore sintática (**??**).

Para a quarta entrega, focamos no ambiente de execução. O compilador por nós criado terá como linguagem de saída um programa que será executado na máquina virtual conhecida como Máquina de von Neumann (MVN).

Para a entrega atual, buscamos completar a especificação do código gerado pelo compilador e das rotinas do ambiente de execução da nossa linguagem de alto nível, a CZAR.

Como material de consulta, além de sites sobre o assunto e das aulas ministradas, foi utilizado o livro indicado pelo professor no começo das aulas (**??**), para pesquisa de conceitos e possíveis implementações.

O documento apresenta a seguir o que foi solicitado na quinta etapa.

---

1 Ver http://en.wikipedia.org/wiki/Backus_Naur_Form
2 Ver http://en.wikipedia.org/wiki/Wirth_syntax_notation

2 Tradução de estruturas de controle de fluxo

Será apresentado nas próximas seções, as traduções das estruturas de controle de fluxo que constam na nossa linguagem e foram solicitadas para essa entrega, entre elas as estruturas if, if-else e while.

Cabe ressaltar que foram utilizadas simbologias nas traduções que serão substituídas pelo compilador no momento da geração de código. Uma dessas marcações é os dois pontos no começo de uma linha que significa que os comandos devem ser colocados no início do código gerado. Outra simbologia criada é da forma XN, onde X representa uma letra maiúscula qualquer e N é o índice da instância dentro do tipo de marcação X. As opções para X são as seguintes:

- `{C0}, {C1}, ...`: Conjunto de comandos
- `{R0}, {R1}, ...`: Referência
- `{L0}, {L1}, ...`: Label ou rótulo de uma instrução criados e exportados pelo código

Há também a marcação `{N}`, utilizada para denotar que a primeira instrução do código subsequente ao comando atual deve ser adicionada no lugar da marcação. Estamos considerando substituir sempre a marcação `{N}` por uma instrução simples que só sirva para simplificar, como por exemplo somar zero ao acumulador.

Conceitos da pilha aritmética são utilizados para o cálculo de expressões booleanas, no chapter 4 explicações mais detalhadas são apresentadas.

2.1 Estrutura de controle de fluxo: IF

```
1          {C0}    # calculo da expressao booleana
2          SC POP_ARITH
3          JZ {L0} # se 0, entao pula para L0 (else)
4          {C1}    # codigo if, C pode ser nulo ou mais coisas
5 {L0}     {N}     # N executa somente a expansao
```

2.2 Estrutura de controle de fluxo: IF-ELSE

```
1          {C0}    # calculo da expressao booleana
2          SC POP_ARITH
```

```
3           JZ {L0} # se 0, entao pula para L0 (else)
4           {C0}    # codigo if, C pode ser nulo ou mais coisas
5           JP {L1} # codigo fim, pula para fim
6 {L0}      {C2}    # codigo else
7 {L1}      {N}     # N executa somente a expansao
```

2.3 Estrutura de controle de fluxo: WHILE

```
1 {L0} +  ZER0    # STUB para facilitar criacao de codigo
2      {C0}       # calculo da expressao boleana
3      SC POP_ARITH
4      JZ {L0}
5      {C1}       # corpo do while
6 {L0} {N}
```

3 Tradução de comandos imperativos

Esse capítulo explica as traduções dos comandos imperativos que constam na nossa linguagem e foram solicitadas para essa entrega, entre os quais os comandos de atribuição de valor, leitura da entrada padrão, impressão na saída padrão e chamada de subrotinas, associado à definicão de novas subrotinas. As mesmas definições das marcações explicadas no Capítulo 2 são válidas para as traduções a seguir.

3.1 Atribuição de valor

```
1     LD {R0}
2     MM {R1}
```

3.2 Comando de leitura

```
1 :       < STORE_PTR_IO
2 :       < GETS
3         LV {R0}
4         MM STORE_PTR_IO
5         SC GETS
```

3.3 Comando de impressão

```
1 :   < STRING_PTR
2 :   < P_STRING
3     LV {R0}
4     MM STRING_PTR
5     SC P_STRING
```

3.4 Definição e chamada de subrotinas

No caso da definição de subrotinas, a tradução fica a seguinte:

```
1 :               <   PUSH_CALL
2 :               <   POP_CALL
3 :               <   LOAD_WORD_FROM_STACK
4 :               <   SAVE_WORD_ON_STACK
5 :               <   WORD_TO_BE_SAVED
6
```

```
7                       {C0}        # dados da funcao
8                       {C1}        # constante com o numero de dados da
9                                   #  funcao em nibbles
10 {L0}                 JP /000     # label funcao
11                      LV {L0}
12                      SC PUSH_CALL
13                      {C2}
14                      LV {L0}
15                      SC POP_CALL
16                      RS {L0}
```

Vale salientar que as funcoes que tratam a pilha de registro de ativação foram modificadas completamente para integração mais transparente na implementacao da função.

Já quando é identificada a chamada de uma subrotina já declarada, a seguinte tradução é utilizada:

```
1     {C0}                  # Copia dados para os argumentos
2                           # da funcao
3     SC {R0}
```

4 Cálculo de expressões aritméticas e booleanas

Além do que foi solicitado como obrigatório para essa entrega, pensamos ser importante definir a forma como fizemos a implementação do cálculo de expressões para a geração de código de saída.

Como o professor Ricardo Rocha nos explicou, a MVN não tem uma implementação real de pilha, porém consegue simular a existência de uma pilha com o uso de indirecionamentos que definem cada uma das operações da pilha, como *push* e *pop*. Baseado nesse conceito de código alinhavado, definimos diversas funções auxiliares que realizam operações simples de forma independente. Essas funções nos permitiram realizar o cálculo de expressões de maneira mais clara e com menos erros.

Para explicar de forma mais detalhada o processo utilizado para calcular as expressões, vamos supor que lemos uma expressão `1 + 2 * 3`. A gramática que já implementamos nas etapas anteriores cria uma árvore que já considera a ordem de prioridade das operações, fazendo com que a multiplicação ocorra antes da soma. Para esse caso, o código de máquina deve primeiro empilhar o 1, em seguida o 2 e depois o 3. Ao notar que uma operação de multiplicação foi finalizada, ele retira da pilha dois operandos, no caso o 2 e o 3, realizando a multiplicação e retornando a pilha o resultado da operação, no caso 6. Em seguida, é efetuada a operação de soma com os dois operandos que estão na pilha, o 1 e o 6, adicionando novamente o resultado, 7, na pilha.

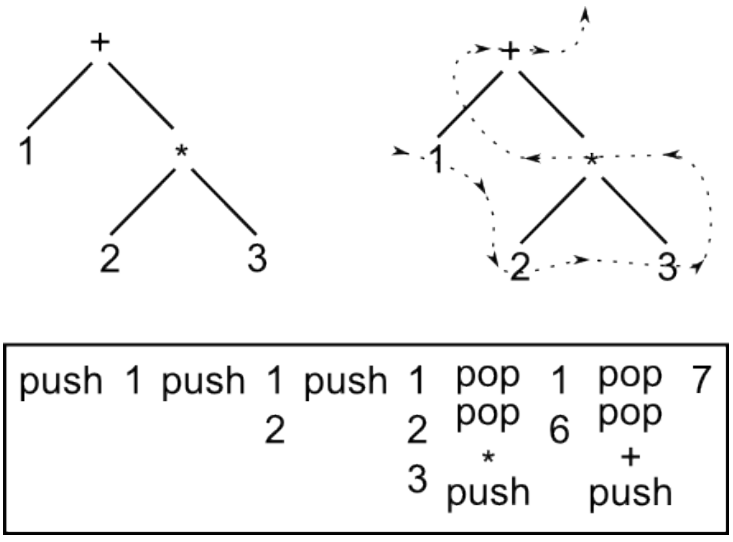

Figura 1 – Árvore da expressão e operações resultantes na pilha.

O mesmo tipo de lógica foi implementado também para operadores booleanos e

permite a geração de código de forma mais simples, visto que já desenvolvemos funções auxiliares para essas operações.

Apresentamos abaixo as operações principais da pilha aritmética. Todas as outras operações da pilha se encontram no arquivo `std.asm` no final do arquivo.

```
1  ;-------------------------PUSH_ARITH-----------------------------
2  PUSH_ARITH       JP /000
3                   MM TMP_1
4                   LD ARIT_PTR_STACK
5                   +  TWO
6                   MM ARIT_PTR_STACK
7                   +  MOVE_CONST
8                   MM OP_PUSH_ARITH
9                   LD TMP_1
10 OP_PUSH_ARITH    JP /000
11                  RS PUSH_ARITH
12 ;-------------------------POP_ARITH------------------------------
13 POP_ARITH        JP /000
14                  LD ARIT_PTR_STACK
15                  -  TWO
16                  MM ARIT_PTR_STACK
17                  +  TWO
18                  +  LOAD_CONST
19                  MM OP_POP_ARITH
20 OP_POP_ARITH     JP /000
21                  RS POP_ARITH
22 ;-------------------------SUM_ARITH------------------------------
23 SUM_ARITH        JP /000
24                  SC POP_ARITH
25                  MM TMP_2
26                  SC POP_ARITH
27                  +  TMP_2
28                  SC PUSH_ARITH
29                  RS SUM_ARITH
30 ;-------------------------MUL_ARITH------------------------------
31 MUL_ARITH        JP /000
32                  SC POP_ARITH
33                  MM TMP_2
34                  SC POP_ARITH
35                  *  TMP_2
36                  SC PUSH_ARITH
37                  RS MUL_ARITH
```

5 Arrays e Structs

Em *CZAR* **não** existem *Arrays* de tamanho dinâmico e sua criação está limitada à declaração. Sendo assim, suas dimensões internas são conhecidas pelo compilador a todo momento e seu cálculo de posição é facilitado e feito em tempo de compilação.

```
1  int i;
2
3  /*
4  * Array int[4][3][2]:
5  *
6  * [
7  *     [[0, 0], [0, 0], [0, 0]],
8  *     [[0, 0], [0, 0], [0, 0]],
9  *     [[0, 0], [0, 0], [0, 0]],
10 *     [[0, 0], [0, 0], [0, 0]]
11 * ]
12 *
13 * Preenchendo com:
14 * ----------------
15 * decl int i;
16 * decl int j;
17 * decl int k;
18 * decl int l;
19 * set i = 0;
20 * set j = 0;
21 * while (j < 4) {
22 *    set k = 0;
23 *    while (k < 3) {
24 *       set l = 0;
25 *       while (l < 2) {
26 *          set array_ex[j][k][l] = i;
27 *          set i = i + 1;
28 *          set l = l + 1;
29 *       }
30 *       set k = k + 1;
31 *    }
32 *    set j = j + 1;
33 * }
34 *
35 * Temos:
36 * ------
37 * [
38 *     [[0, 1], [2, 3], [4, 5]],
39 *     [[6, 7], [8, 9], [10, 11]],
```

```
*     [[12, 13], [14, 15], [16, 17]],
*     [[18, 19], [20, 21], [22, 23]]
* ]
* ou:
* ---
* [
*   0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
*   10, 11, 12, 13, 14, 15, 16, 17, 18,
*   19, 20, 21, 22, 23
* ]
*
*
*
*
*/
acc = acumulado[n_dimensoes - 1] = 1; // maybe long 1L
for (i = n_dimensoes-2; i >= 0; i--) {
   acc = acumulado[i+1] = dimensoes[i+1] * acc;
}
/*
   acumulado = [6, 2, 1];
*/
for (i = 0; i < n_dimensoes; i++) {
    acumulado[i] = acumulado[i] * size_cell; // celulas podem ter
                                             // tamanho variavel
}
for (i = 0; i < n_dimensoes; i++) {
    fprints(str, "␣LV␣=%d␣", acumulado[i]);
    cpy_to_lines_of_code(str);
    fprints(str, "␣*␣ARR_DIM_%d", acumulado[i]);
    cpy_to_lines_of_code(str);
    fprints(str, "␣+␣␣ADDRS_ACCUMULATOR");
    cpy_to_lines_of_code(str);
    fprints(str, "␣MM␣ADDRS_ACCUMULATOR");
    cpy_to_lines_of_code(str);
}
```

O cálculo de *structs* é resolvido em tempo de compilação. Uma vez que o tamanho de cada parte da estrutura é conhecida em tempo de compilação, é possível se fazer toda a aritmética de acesso via programação em *C*.

```
    int deslocamento_para_celula(struct_struct* vi_struct, int
        cell_to_access) {
        int sum_up_to_ptr = 0;
        for (i = 0; i < cell_to_access; i++) {
            sum_up_to_ptr += vi_struct->sizes[i];
        }
```

```
6           return sum_up_to_ptr;
7       }
```

6 Exemplo de programa traduzido

A fim de demonstrar tudo o que foi pensado como a maneira de traduzir os comandos de alto nível da nossa linguagem CZAR, nós traduzimos um programa simples de fatorial que permite visualizar e testar a nossa tradução.

Para isso, apresentamos o exemplo de programa escrito em três diferentes linguagens: (i) na nossa linguagem de alto nível CZAR; (ii) tradução para linguagem de máquina, utilizando as bibliotecas complementares *std* e *stdio*; (iii) tradução para linguagem de saída MVN.

Adicionamos as bibliotecas *std* e *stdio* como apêndices (ver Apêndice A e B) desse documento para consulta sobre o que já foi efetivamente desenvolvido.

6.1 Exemplo de programa fatorial na linguagem de alto nível

```
1  const int fat_10_rec = 6;
2  decl int retorno;
3
4  meth
5
6  int fatorial_recursivo(int n) {
7      decl int retorno = 1;
8      if (n >= 1) {
9          set retorno = n * call fatorial_recursivo (n - 1);
10     }
11     return retorno;
12 }
13
14 main () {
15     set retorno = call fatorial_recursivo(fat_10_rec);
16     call io_print_int(retorno);
17 }
```

6.2 Tradução do programa fatorial para linguagem de máquina

```
1  P_STRING                <
2  STRING_PTR              <
3  P_INT_ZERO              <
4  TO_BE_PRINTED           <
5  P_LINE                  <
6  PUSH_CALL               <
```

```
7  PRINT_STACK_ADDRS       <
8  POP_CALL                <
9  READ_INT                <
10 STORE_PTR               <
11 GETS                    <
12 STORE_PTR_IO            <
13 WORD_TO_BE_SAVED        <
14 SAVE_WORD_ON_STACK      <
15 LOAD_WORD_FROM_STACK    <
16 ORIGIN_PTR              <
17 POP_CALL_FCT            <
18 PUSH_ARITH              <
19 POP_ARITH               <
20 SUM_ARITH               <
21 SUB_ARITH               <
22 DIV_ARITH               <
23 MUL_ARITH               <
24 AND_ARITH               <
25 OR_ARITH                <
26 NOT_ARITH               <
27 GEQ_OPER_ARITH          <
28 LEQ_OPER_ARITH          <
29 DBG                     <
30 @ /0000
31 CZAR_INICIO_CODE        JP CZAR_INICIO
32 CZAR_STUB               K =1
33 ;;=========================================================
34
35 CONST_VAR_0             K =6  ; const int fat_10_rec = 6;
36 GLOBAL_VAR_0            K =0  ; decl int retorno;
37 _CONST_NUM_1            K =0001
38
39 FUNCTION_0_RETURN       K =0  ; int
40 FUNCTION_0_ARG_0        K =0  ; int n
41 FUNCTION_0_TMP_0        K =0  ; function return
42 FUNCTION_0_LOCAL_VAR_0  K =0  ; decl int retorno = 1;
43                         K =4  ; int fatorial_recursivo(int n) {
44 FUNCTION_0              JP /000 ;
45                         LV FUNCTION_0
46                         SC PUSH_CALL
47
48                         LV =1                    ; retorno = 1;
49                         MM WORD_TO_BE_SAVED
50                         LV =3                    ; FUNCTION_0_LOCAL_VAR_0
51                         SC SAVE_WORD_ON_STACK
52                         ;; (n >= 1)
53                         LV =2                    ; FUNCTION_0_ARG_0
```

```
54                          SC LOAD_WORD_FROM_STACK
55                          SC PUSH_ARITH
56                          LD _CONST_NUM_1
57                          SC PUSH_ARITH
58                          SC GEQ_OPER_ARITH
59                          SC POP_ARITH
60                          JZ FUNCTION_0_LABEL_0
61
62                          LV =2
63                          SC LOAD_WORD_FROM_STACK
64                          SC PUSH_ARITH
65                          LD _CONST_NUM_1
66                          SC PUSH_ARITH
67                          SC SUB_ARITH
68                          SC POP_ARITH
69
70                          MM FUNCTION_0_ARG_0
71                          SC FUNCTION_0           ; call
                               fatorial_recursivo (n - 1);
72                          LD FUNCTION_0_RETURN
73                          MM WORD_TO_BE_SAVED
74                          LV =1
75                          SC SAVE_WORD_ON_STACK
76
77                          LV =2                   ; n * call
                               fatorial_recursivo (n - 1);
78                          SC LOAD_WORD_FROM_STACK
79                          SC PUSH_ARITH
80                          LV =1
81                          SC LOAD_WORD_FROM_STACK
82                          SC PUSH_ARITH
83                          SC MUL_ARITH
84
85                          SC POP_ARITH ; set retorno = *
86                          MM WORD_TO_BE_SAVED
87                          LV =3
88                          SC SAVE_WORD_ON_STACK
89  FUNCTION_0_LABEL_0      *  CZAR_STUB
90                          LV =3
91                          SC LOAD_WORD_FROM_STACK
92                          SC PUSH_ARITH
93                          SC POP_ARITH
94                          MM FUNCTION_0_RETURN
95                          LV FUNCTION_0
96                          SC POP_CALL     ;; trickery!
97                          RS FUNCTION_0   ;     return retorno;
98  ;; INIT PROG ==========================================
```

```
99  CZAR_INICIO             * CZAR_STUB ;; stub instruction
100                         LD CONST_VAR_0
101                         MM FUNCTION_0_ARG_0
102                         SC FUNCTION_0       ;; depois chama..
103                         LD FUNCTION_0_RETURN
104                         MM GLOBAL_VAR_0
105                         LD GLOBAL_VAR_0
106                         MM TO_BE_PRINTED
107                         SC P_INT_ZERO
108 FIM                     HM /00
109 # CZAR_INICIO_CODE
```

6.3 Tradução do programa fatorial para linguagem de saída MVN

```
1  4000 0000 ; "P_STRING<"
2  4001 0000 ; "STRING_PTR<"
3  4002 0000 ; "P_INT_ZERO<"
4  4003 0000 ; "TO_BE_PRINTED<"
5  4004 0000 ; "P_LINE<"
6  4005 0000 ; "PUSH_CALL<"
7  4006 0000 ; "PRINT_STACK_ADDRS<"
8  4007 0000 ; "POP_CALL<"
9  4008 0000 ; "READ_INT<"
10 4009 0000 ; "STORE_PTR<"
11 4010 0000 ; "GETS<"
12 4011 0000 ; "STORE_PTR_IO<"
13 4012 0000 ; "WORD_TO_BE_SAVED<"
14 4013 0000 ; "SAVE_WORD_ON_STACK<"
15 4014 0000 ; "LOAD_WORD_FROM_STACK<"
16 4015 0000 ; "ORIGIN_PTR<"
17 4016 0000 ; "POP_CALL_FCT<"
18 4017 0000 ; "PUSH_ARITH<"
19 4018 0000 ; "POP_ARITH<"
20 4019 0000 ; "SUM_ARITH<"
21 4020 0000 ; "SUB_ARITH<"
22 4021 0000 ; "DIV_ARITH<"
23 4022 0000 ; "MUL_ARITH<"
24 4023 0000 ; "AND_ARITH<"
25 4024 0000 ; "OR_ARITH<"
26 4025 0000 ; "NOT_ARITH<"
27 4026 0000 ; "GEQ_OPER_ARITH<"
28 4027 0000 ; "LEQ_OPER_ARITH<"
29 4028 0000 ; "DBG<"
30 0000 0074
31 0002 0001
32 0004 0006
```

```
0006 0000
0008 0001
000a 0000
000c 0000
000e 0000
0010 0000
0012 0004
0014 0000
0016 3014
5018 a005
001a 3001
501c 9012
001e 3003
5020 a013
0022 3002
5024 a014
5026 a017
0028 8008
502a a017
502c a026
502e a018
0030 1062
0032 3002
5034 a014
5036 a017
0038 8008
503a a017
503c a020
503e a018
0040 900c
0042 a014
0044 800a
5046 9012
0048 3001
504a a013
004c 3002
504e a014
5050 a017
0052 3001
5054 a014
5056 a017
5058 a022
505a a018
505c 9012
005e 3003
5060 a013
0062 6002
```

```
0064 3003
5066 a014
5068 a017
506a a018
006c 900a
006e 3014
5070 a007
0072 b014
0074 6002
0076 8004
0078 900c
007a 3666
007c a014
007e 800a
0080 9006
0082 8006
5084 9003
5086 a002
0088 c000
```

Apêndices

APÊNDICE A – Biblioteca auxiliar *std*

```
; Biblioteca de ambiente
; ======================
; exports
DBG                     >
PUSH_CALL               >
POP_CALL                >
SAVE_WORD_ON_STACK      >
LOAD_WORD_FROM_STACK    >
WORD_TO_SAVE            >
ORIGIN_PTR              >
GET_WORD_LOCAL_VAR      >
WORD_TO_GET             >
STORE_PTR               >
PRINT_STACK_ADDRS       >
POP_CALL_FCT            >
WORD_TO_BE_SAVED        >
;; ARITH:
PUSH_ARITH              >
POP_ARITH               >
SUM_ARITH               >
SUB_ARITH               >
DIV_ARITH               >
MUL_ARITH               >
AND_ARITH               >
OR_ARITH                >
NOT_ARITH               >
EQ_OPER_ARITH           >
NEQ_OPER_ARITH          >
LT_OPER_ARITH           >
GEQ_OPER_ARITH          >
GT_OPER_ARITH           >
LEQ_OPER_ARITH          >
;; imports
READ_INT                <
PONEASCII               <
TO_BE_PRINTED           <
P_STRING                <
STRING_PTR              <
P_INT_ZERO              <
P_LINE                  <
```

```
STORE_PTR_IO            <
; relocavel
& /0000
START_STD_LIB_PADDING K /0000
; alocando para K
SHIFT_BYTE          K /0100
LOAD_CONST          K /8000   ; instruction to load
LOADV_CONST         K /3000   ; instruction to load
MOVE_CONST          K /9000   ; instruction to move to memory
ZERO                K /0000
ONE                 K /0001
TWO                 K /0002
;; *Registers
TMP_1               K /0000
TMP_2               K /0000
TMP_3               K /0000
TMP_4               K /0000
;; VARIAVEIS GLOBAIS
;; comeco da pilha = FFF
;; tamanho da pilha = 2FF
;;    | ptr to old_stack_head  | \___ STACK_PTR
;;    |      savedregist       |
;;    |         ...            |
;;    |      local var         |
;;    |         ...            |
;;    |      temporaries       |
;;    |      parameters        |
;;    |         ...            |
;;    |     ref parameters     |  ____ OLD STACK_PTR
;;    |      returnaddrs       | /     (STACK_PTR points here)
STACK_PTR           K /0FFE

;; FIM VARIAVEIS GLOBAIS

;; **** P_LINE ****
;; imprime uma linha
;;

COUNT_IS_STR      K /436f
                  K /756e
                  K /7465
                  K /7220
                  K /6973
                  K /3a00
                  K /0000

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
;; MISCELANEA ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
HIGH_V              JP /000
LOW_V               JP /000
;; *** HIGH_LOW HIGH_V ***
HIGH_LOW            JP /000
                    LD HIGH_V
                    MM LOW_V
                    LD LOW_V
                    *  SHIFT_BYTE
                    /  SHIFT_BYTE
                    MM LOW_V
                    LD HIGH_V
                    /  SHIFT_BYTE
                    MM HIGH_V

                    LD LOW_V
                    /  SHIFT_BYTE
                    *  SHIFT_BYTE
                    -  LOW_V
                    MM LOW_V
                    LD ZERO
                    -  LOW_V
                    MM LOW_V

                    LD HIGH_V
                    /  SHIFT_BYTE
                    *  SHIFT_BYTE
                    -  HIGH_V
                    MM HIGH_V
                    LD ZERO
                    -  HIGH_V
                    MM HIGH_V
                    RS HIGH_LOW

;;    Empilha um espaco de memoria
;; para a pilha.

;;  PUSH_CALL
;;  CALLEE_FCT
;;    POP_CALL
FUNCTION_BEING_CALLED           K  /000
SIZE_OF_CONTEXT                 K  /000
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; PUSH_CALL ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Use:                                this must be here -|        ;;
```

```
;;                              [CONTEXT HERE]          |        ;;
;;                              K  =<size of context> <-|        ;;
;; FUNCTION_EX                  JP /000                          ;;
;;                              LV FUNCTION_EX                   ;;
;;                              SC PUSH_CALL                     ;;
;;                              ...                              ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
PUSH_CALL                       JP /000
                                MM FUNCTION_BEING_CALLED
                                -  TWO
                                +  LOAD_CONST
                                MM LOAD_SIZE_OF_CONTEXT
LOAD_SIZE_OF_CONTEXT            JP /000
                                MM SIZE_OF_CONTEXT
                                LD FUNCTION_BEING_CALLED  ;; Will copy the
                                                          ;; return address
                                +  LOAD_CONST     ;; TODO VERIFY THIS THING RIGHT HERE
                                MM LOAD_PUSH_CALL ;; WE MAY NEED TO SUM UP 2 IN ORDER TO JUMP
                                                  ;; OVER SOME INSTR
                                LD STACK_PTR
                                -  TWO
                                +  MOVE_CONST
                                MM MOVE_PUSH_CALL
LOAD_PUSH_CALL                  JP /000
MOVE_PUSH_CALL                  JP /000
                                LD STACK_PTR
                                -  TWO
                                -  TWO
                                -  SIZE_OF_CONTEXT
                                -  SIZE_OF_CONTEXT
                                MM TMP_1
                                LD TMP_1
                                +  MOVE_CONST
                                MM MRKR_PC_SAVE_HEAD
                                LD STACK_PTR
MRKR_PC_SAVE_HEAD               JP /000
                                LD TMP_1
                                MM STACK_PTR
                                LD SIZE_OF_CONTEXT
                                *  TWO
                                MM SIZE_OF_CONTEXT


```

```
178 ITER_AND_COPY_CONTEXT           LD SIZE_OF_CONTEXT
179                                 JZ NO_MORE_CONTEXT_COPIES
180                                 LD FUNCTION_BEING_CALLED
181                                 -  TWO
182                                 -  SIZE_OF_CONTEXT
183                                 +  LOAD_CONST
184                                 MM LOAD_VAR_PUSH_CALL
185                                 LD STACK_PTR
186                                 ;+  TWO
187                                 +  SIZE_OF_CONTEXT
188                                 +  MOVE_CONST
189                                 MM MOVE_VAR_PUSH_CALL
190 LOAD_VAR_PUSH_CALL              JP /000
191 MOVE_VAR_PUSH_CALL              JP /000
192                                 LD SIZE_OF_CONTEXT
193                                 -  TWO
194                                 MM SIZE_OF_CONTEXT
195
196                                 JP ITER_AND_COPY_CONTEXT
197 NO_MORE_CONTEXT_COPIES          * ONE
198                                 RS PUSH_CALL
199
200 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
201 ;; POP_CALL ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
202 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
203 ;; Use:                                                                      ;;
204 ;;                                  ...                                    ;;
205 ;;                                  LV FCT_EXAMPLE                         ;;
206 ;;                                  SC POP_CALL                            ;;
207 ;;                                  RS FCT_EXAMPLE                         ;;
208 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
209 POP_CALL_FCT                       K /0000
210 POP_CALL                           JP /000 ; retorno
211                                     MM POP_CALL_FCT
212                                     LD STACK_PTR
213                                     +  LOAD_CONST
214                                     MM MRKR_PC_LOAD_HEAD
215 MRKR_PC_LOAD_HEAD                  JP /000
216                                     MM STACK_PTR
217                                     LD STACK_PTR
218                                     -  TWO
219                                     +  LOAD_CONST
220                                     MM LOAD_RETURN_ADDRS_2
221                                     LD POP_CALL_FCT
222                                     +  MOVE_CONST
223                                     MM MOVE_RETURN_ADDRS_2
224 LOAD_RETURN_ADDRS_2                JP /000
```

```
MOVE_RETURN_ADDRS_2             JP /000  ;; engana a funcao para ela
   pensar que ela
                                         ;; tem que retornar para esse
                                            valor
                                RS POP_CALL

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; SAVE_WORD_ON_STACK ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Use:                                                                      ;;
;;                                 ...                                       ;;
;;                                 LD VERY_IMPORTANT_VAR                     ;;
;;                                 MM WORD_TO_BE_SAVED                       ;;
;;                                 LV =3                                     ;;
;;                                 SC SAVE_WORD_ON_STACK                     ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

WORD_NUMBER_TO_SAVE             K /000
WORD_TO_BE_SAVED                K /000
SAVE_WORD_ON_STACK              K /000
                                MM WORD_NUMBER_TO_SAVE
                                LD STACK_PTR
                                + TWO           ;; first word
                                + WORD_NUMBER_TO_SAVE
                                + WORD_NUMBER_TO_SAVE
                                + MOVE_CONST    ;;
                                MM MOVE_WORD_LOCAL_VAR_2
                                LD WORD_TO_BE_SAVED
MOVE_WORD_LOCAL_VAR_2           JP /000 ;; 9TOPTR
                                RS SAVE_WORD_ON_STACK

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; LOAD_WORD_FROM_STACK ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Use:                                                                      ;;
;;                                 ...                                       ;;
;;                                 LV =3                                     ;;
;;                                 SC LOAD_WORD_FROM_STACK                   ;;
;;                                 ...                                       ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
WORD_TO_GET                     K /000
LOAD_WORD_FROM_STACK            JP /000
                                MM WORD_TO_GET
                                LD STACK_PTR
                                + TWO           ;; first word
                                + WORD_TO_GET
                                + WORD_TO_GET   ;; WORD_TO_GET * 2
```

```
                            + LOAD_CONST    ;;
                            MM LOAD_WORD_LOCAL_VAR
LOAD_WORD_LOCAL_VAR         JP /000 ;; 8FROMPTR
                            RS LOAD_WORD_FROM_STACK


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; PRINT_STACK_ADDRS ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Use:                                                           ;;
;;                              ...                               ;;
;;                              SC PRINT_STACK_ADDRS              ;;
;;                              ...                               ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

STACK_WAS_STR K  /5374
              K  /6163
              K  /6b20
              K  /0077
              K  /6173
              K  /3a20
              K  /0000

PRINT_STACK_ADDRS JP /000
                  LV STACK_WAS_STR    ; carrega o ponteiro da string
                  MM STRING_PTR ; copia o ponteiro para o parametro
                  SC P_STRING   ; roda p_string
                  LD STACK_PTR
                  MM TO_BE_PRINTED
                  SC P_INT_ZERO
                  SC P_LINE
                  RS PRINT_STACK_ADDRS

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; PILHA ARITIMETICA ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

ARIT_STACK_ZERO K /0000   ;; 0
                K /0000   ;; 1
                K /0000   ;; 2
                K /0000   ;; 3
                K /0000   ;; 4
                K /0000   ;; 5
                K /0000   ;; 6
                K /0000   ;; 7
                K /0000   ;; 8
                K /0000   ;; 9
```

```
317                 K /0000    ;; 10
318                 K /0000    ;; 11
319                 K /0000    ;; 12
320                 K /0000    ;; 13
321                 K /0000    ;; 14
322                 K /0000    ;; 15
323                 K /0000    ;; 16
324                 K /0000    ;; 17
325                 K /0000    ;; 18
326                 K /0000    ;; 19
327                 K /0000    ;; 20
328                 K /0000    ;; 31
329                 K /0000    ;; 32
330                 K /0000    ;; 33
331                 K /0000    ;; 34
332                 K /0000    ;; 35
333                 K /0000    ;; 36
334                 K /0000    ;; 37
335                 K /0000    ;; 38
336                 K /0000    ;; 39
337                 K /0000    ;; 30
338 ARIT_PTR_STACK  K ARIT_STACK_ZERO
339
340 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
341 ;; ARITHMETIC CALLS ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
342 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
343
344 ;-------------------------PUSH_ARITH-----------------------------
345 PUSH_ARITH      JP /000
346                 MM TMP_1
347                 LD ARIT_PTR_STACK
348                 +  TWO
349                 MM ARIT_PTR_STACK
350                 +  MOVE_CONST
351                 MM OP_PUSH_ARITH
352                 LD TMP_1
353 OP_PUSH_ARITH   JP /000
354                 RS PUSH_ARITH
355 ;-------------------------POP_ARITH------------------------------
356 POP_ARITH       JP /000
357                 LD ARIT_PTR_STACK
358                 -  TWO
359                 MM ARIT_PTR_STACK
360                 +  TWO
361                 +  LOAD_CONST
362                 MM OP_POP_ARITH
363 OP_POP_ARITH    JP /000
```

```
364                 RS POP_ARITH
365 ;--------------------------FLIP_ARITH------------------------------
366 FLIP_ARITH      JP /000
367                 MM TMP_4
368                 SC POP_ARITH
369                 MM TMP_2
370                 SC POP_ARITH
371                 MM TMP_3
372                 LD TMP_2
373                 SC PUSH_ARITH
374                 LD TMP_3
375                 SC PUSH_ARITH
376                 LD TMP_4
377                 RS FLIP_ARITH
378 ;--------------------------SUM_ARITH-------------------------------
379 SUM_ARITH       JP /000
380                 SC POP_ARITH
381                 MM TMP_2
382                 SC POP_ARITH
383                 +  TMP_2
384                 SC PUSH_ARITH
385                 RS SUM_ARITH
386 ;--------------------------SUB_ARITH-------------------------------
387 SUB_ARITH       JP /000
388                 SC POP_ARITH
389                 MM TMP_2
390                 SC POP_ARITH
391                 -  TMP_2
392
393                 SC PUSH_ARITH
394                 RS SUB_ARITH
395
396 ;--------------------------DIV_ARITH-------------------------------
397 DIV_ARITH       JP /000
398                 SC POP_ARITH
399                 MM TMP_2
400                 SC POP_ARITH
401                 /  TMP_2
402                 SC PUSH_ARITH
403                 RS DIV_ARITH
404
405 ;--------------------------MUL_ARITH-------------------------------
406 MUL_ARITH       JP /000
407                 SC POP_ARITH
408                 MM TMP_2
409                 SC POP_ARITH
410                 *  TMP_2
```

```
411                 SC PUSH_ARITH
412                 RS MUL_ARITH
413
414 ;-------------------------AND_ARITH-------------------------------
415 AND_ARITH        JP /000
416                 SC POP_ARITH
417                 MM TMP_2
418                 SC POP_ARITH
419                 JZ PUSH_ZERO_AND_ARITH
420                 LD TMP_2
421                 JZ PUSH_ZERO_AND_ARITH
422                 LV /001
423                 SC PUSH_ARITH
424                 RS AND_ARITH
425 PUSH_ZERO_AND_ARITH  LV /000
426                      SC PUSH_ARITH
427                      RS AND_ARITH
428
429 ;-------------------------OR_ARITH--------------------------------
430 OR_ARITH             JP /000
431                      SC POP_ARITH
432                      MM TMP_2
433                      SC POP_ARITH
434                      JZ ONE_ZERO_OR_ARITH
435                      LV /001
436                      SC PUSH_ARITH
437                      RS OR_ARITH
438 ONE_ZERO_OR_ARITH   LD TMP_2
439                      JZ PUSH_ZERO_OR_ARITH
440                      LV /001
441                      SC PUSH_ARITH
442                      RS OR_ARITH
443 PUSH_ZERO_OR_ARITH  LV /000
444                      SC PUSH_ARITH
445                      RS OR_ARITH
446
447 ;-------------------------EQ_OPER_ARITH---------------------------
448 EQ_OPER_ARITH        JP /000
449                      SC SUB_ARITH
450                      SC POP_ARITH
451                      JZ EQ_OPER_ARITH_EQUAL
452                      LD ZERO
453                      SC PUSH_ARITH
454                      RS EQ_OPER_ARITH
455 EQ_OPER_ARITH_EQUAL LD ONE
456                      SC PUSH_ARITH
457                      RS EQ_OPER_ARITH
```

```
458
459 ;-------------------------NEQ_OPER_ARITH-------------------------
460 NEQ_OPER_ARITH       JP /000
461                      SC EQ_OPER_ARITH
462                      SC NOT_ARITH
463                      RS NEQ_OPER_ARITH
464
465 ;-------------------------LT_OPER_ARITH-------------------------
466 LT_OPER_ARITH         JP /000            ;; a1 < a2
467                      SC SUB_ARITH
468                      SC POP_ARITH
469                      JN LT_OPER_ARITH_RET_ONE
470                      LD ZERO
471                      SC PUSH_ARITH
472                      RS LT_OPER_ARITH
473 LT_OPER_ARITH_RET_ONE LD ONE
474                      SC PUSH_ARITH
475                      RS LT_OPER_ARITH
476
477 ;-------------------------GEQ_OPER_ARITH-------------------------
478 GEQ_OPER_ARITH        JP /000     ;; a1 >= a2
479                      SC LT_OPER_ARITH
480                      SC NOT_ARITH
481                      RS GEQ_OPER_ARITH
482
483 ;-------------------------GT_OPER_ARITH-------------------------
484 GT_OPER_ARITH         JP /000    ;; a1 > a2
485                      SC FLIP_ARITH
486                      SC LT_OPER_ARITH
487                      RS GT_OPER_ARITH
488
489
490 ;-------------------------LEQ_OPER_ARITH-------------------------
491 LEQ_OPER_ARITH        JP /000     ;; a1 <= a2
492                      SC GT_OPER_ARITH
493                      SC NOT_ARITH
494                      RS LEQ_OPER_ARITH
495
496 ;-------------------------NOT_ARITH-------------------------------
497 NOT_ARITH           JP /000
498
499                     SC POP_ARITH
500                     JZ PUSH_ONE_NOT_ARITH
501                     LV =0
502                     SC PUSH_ARITH
503                     RS NOT_ARITH
504 PUSH_ONE_NOT_ARITH  LV =1
```

```
505                     SC PUSH_ARITH
506                     RS NOT_ARITH
507
508 ;-------------------------DBG_ARITH-------------------------------
509 DBG_ARITH           JP /000
510                     MM TMP_3
511                     SC POP_ARITH
512                     MM TMP_4
513                     MM TO_BE_PRINTED
514                     SC P_INT_ZERO
515                     SC P_LINE
516                     LD TMP_4
517                     SC PUSH_ARITH
518                     LD ARIT_PTR_STACK
519                     MM TO_BE_PRINTED
520                     SC P_INT_ZERO
521                     SC P_LINE
522                     SC P_LINE
523                     LD TMP_3
524                     RS DBG_ARITH
525
526 # START_STD_LIB_PADDING
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541 CTR                     K =0
542 _KEEP_ALIVE_DBG         K =0
543 DBG                     MM _KEEP_ALIVE_DBG
544                         LV /003C
545                         PD /100
546                         LD CTR
547                         MM TO_BE_PRINTED
548                         SC P_INT_ZERO
549                         LV /003E
550                         PD /100
551                         LV /000A
```

```
552                             PD /100
553                             LV =1
554                             +  CTR
555                             MM CTR
556                             SC PRINT_STACK_ADDRS
557                             LD _KEEP_ALIVE_DBG
558                             RS DBG
```

APÊNDICE B – Biblioteca auxiliar *stdio*

```
PONEASCII               >
TO_BE_PRINTED           >
P_STRING                >
STRING_PTR              >
P_INT_ZERO              >
P_LINE                  >
STORE_PTR_IO            >
READ_INT                >
GETS                    >



& /0000
START_STD_LIB_PADDING K /0000

SHIFT_BYTE          K /0100
TO_BE_PRINTED       K /0000
TO_BE_PRINTED_TMP   K /0000
LOAD_CONST          K /8000   ; instruction to load
LOADV_CONST         K /3000   ; instruction to load
MOVE_CONST          K /9000   ; instruction to move to memory
ZERO                K /0000
TWO                 K /0002
ONE                 K /0001
TEN                 K =10
STRING_PTR          K /0000   ; ponteiro para o comeco da string
TMP_1               K /0000
TMP_2               K /0000
TMP_3               K /0000
TMP_4               K /0000


; MISCELANEA:
;; *** HIGH_LOW HIGH_V ***
HIGH_V              JP /000
LOW_V               JP /000
HIGH_LOW            JP /000
                    LD HIGH_V
                    MM LOW_V
                    LD LOW_V
```

```
41                      *  SHIFT_BYTE
42                      /  SHIFT_BYTE
43                      MM LOW_V
44                      LD HIGH_V
45                      /  SHIFT_BYTE
46                      MM HIGH_V
47
48                      LD LOW_V
49                      /  SHIFT_BYTE
50                      *  SHIFT_BYTE
51                      -  LOW_V
52                      MM LOW_V
53                      LD ZERO
54                      -  LOW_V
55                      MM LOW_V
56
57                      LD HIGH_V
58                      /  SHIFT_BYTE
59                      *  SHIFT_BYTE
60                      -  HIGH_V
61                      MM HIGH_V
62                      LD ZERO
63                      -  HIGH_V
64                      MM HIGH_V
65
66                      RS HIGH_LOW
67 ;;*** P_LINE***
68 ;; imprime uma linha nova
69                      K  /6665
70 P_LINE               JP /000           ; alocando para o endereo de
       retorno
71                      LV /00A  ; carregar a variavel
72                      PD /100           ; imprimir na tela
73                      RS P_LINE         ; retornar a rotina
74
75 ; inicio da rotina
76 PONEASCII            JP /000           ; alocando para o endereo de
       retorno
77 PONEASCII_INIT       LD TO_BE_PRINTED  ; carregar a variavel
78                      PD /100           ; imprimir na tela
79                      RS PONEASCII      ; retornar a rotina
80
81 ;; ****  P_STRING &STRING_PTR ****
82 ;;   Imprime a string apontada por STRING_PTR ate
83 ;; o caractere /000
84
85 P_STRING             JP /000           ; endereco de retorno
```

```
86  PSTRINGINIT         LD STRING_PTR
87                      MM TO_BE_PRINTED_TMP
88  LOAD_TO_BE_PRINTED  LD TO_BE_PRINTED_TMP
89                      +  LOAD_CONST
90                      MM LABELLOAD
91  LABELLOAD           K  /0000
92                      JZ P_STRING_END  ; se zero vamos para o final!
93                      PD /100
94                      LD TO_BE_PRINTED_TMP
95                      +  TWO
96                      MM TO_BE_PRINTED_TMP
97                      JP LOAD_TO_BE_PRINTED
98  P_STRING_END        RS P_STRING
99
100 ;; *** READ_INT STORE_PTR_IO ***
101 ;; doesnt care about buffers, should have a trailing char at the end of
        the
102 ;; stream otherwise it will just discard it..
103 STORE_PTR_IO        JP /000
104 ZERO_M_ONE          K  /002F
105 NINE_P_ONE          K  /0039
106
107 LOW                 K  /0000
108 HIGH                K  /0000
109 GO_IF_NUMBER        K  /0000
110 TO_BE_TRIMMED       K  /0000
111 TBT_TMP             K  /0000
112
113 TRIM_INT            JP /000
114                     LD TO_BE_TRIMMED
115                     /  SHIFT_BYTE
116                     *  SHIFT_BYTE
117                     MM TBT_TMP
118                     LD TO_BE_TRIMMED
119                     -  TBT_TMP
120                     MM TO_BE_TRIMMED
121                     RS TRIM_INT
122
123 READ_INT_WORD       JP /000
124                     GD /000
125                     MM TMP_3
126                     LD TMP_3
127                     /  SHIFT_BYTE
128                     MM TO_BE_TRIMMED
129                     SC TRIM_INT
130                     LD TO_BE_TRIMMED
131                     MM HIGH
```

```
132
133                     LD TMP_3
134                     MM TO_BE_TRIMMED
135                     SC TRIM_INT
136                     LD TO_BE_TRIMMED
137                     MM LOW
138                     RS READ_INT_WORD
139
140 READ_INT             JP /000
141                     LV =0
142                     MM TMP_4
143 READ_INT_LOOP        SC READ_INT_WORD
144                     LD HIGH
145                     MM TMP_3
146                     LV CONT1
147                     MM GO_IF_NUMBER
148                     JP IF_NUMBER_CONTINUE
149 CONT1                LD LOW
150                     MM TMP_3
151                     LV READ_INT_LOOP
152                     MM GO_IF_NUMBER
153                     JP IF_NUMBER_CONTINUE
154 NOT_NUMBER           LD STORE_PTR_IO
155                     +  MOVE_CONST
156                     MM MOVE_READ_INT
157                     LD TMP_4
158 MOVE_READ_INT        JP /000
159                     RS READ_INT
160
161 IF_NUMBER_CONTINUE  LD TMP_3
162                     -  ZERO_M_ONE
163                     JN NOT_NUMBER
164                     LD NINE_P_ONE
165                     -  TMP_3
166                     JN NOT_NUMBER
167
168                     LD TMP_4
169                     *  TEN
170                     MM TMP_4
171
172
173                     LD TMP_3
174                     -  ZERO_M_ONE
175                     -  ONE
176                     +  TMP_4
177                     MM TMP_4
178
```

```
179                     LD GO_IF_NUMBER
180                     MM END_READ_INT
181 END_READ_INT        JP /000
182
183 ;; *** GETS STORE_PTR_IO ***
184 ;; Existe um problema de buffer aqui... nao vamos
185 ;; trata-lo
186 LAST_CONTROL_CHAR_P_ONE    K /0021
187 ARRAY_POS_BYTE  JP /000
188 GETS            JP /000
189                 LD STORE_PTR_IO
190                 MM ARRAY_POS_BYTE
191 GETS_LOOP       GD /000
192                 MM HIGH_V
193                 SC HIGH_LOW
194                 LD HIGH_V
195                 -  LAST_CONTROL_CHAR_P_ONE
196                 JN RETURN_GETS
197                 LD ARRAY_POS_BYTE
198                 +  MOVE_CONST
199                 MM MOVE_HIGH_V
200                 LD HIGH_V
201 MOVE_HIGH_V     JP /000
202
203                 LD ARRAY_POS_BYTE
204                 +  TWO
205                 MM ARRAY_POS_BYTE
206
207                 LD LOW_V
208                 -  LAST_CONTROL_CHAR_P_ONE
209                 JN RETURN_GETS
210                 LD ARRAY_POS_BYTE
211                 +  MOVE_CONST
212                 MM MOVE_LOW_V
213                 LD LOW_V
214 MOVE_LOW_V      JP /000
215
216                 LD ARRAY_POS_BYTE
217                 +  TWO
218                 MM ARRAY_POS_BYTE
219
220                 JP GETS_LOOP
221
222 RETURN_GETS     LD ARRAY_POS_BYTE
223                 +  MOVE_CONST
224                 MM MOVE_ZERO
225                 LV =000
```

```
226 MOVE_ZERO       JP /000
227
228                 LD ARRAY_POS_BYTE
229                 +  TWO
230                 MM ARRAY_POS_BYTE
231                 RS GETS
232
233
234 ;; *** P_INT_ZERO TO_BE_PRINTED ***
235 ;;  Imprime um inteiro (com zeros a esquerda)
236 ;; ex:
237 ;;  INT_2 K  =345
238 ;;        LD INT_2
239 ;;        MM TO_BE_PRINTED
240 ;;        SC P_INT_ZERO
241 ;; imprime 00345
242 ;;
243 ;;
244 ;; Esta funcao esta com o loop inline
245 ;; sendo simples e robusta
246
247 P_INT_ZERO          JP /000
248 P_INT_INIT          JP P_INT_REAL_INIT
249 ZERO_BASE           K /30
250 ;; bases para a conversao:
251 INT_POT_1           K =10000
252 INT_POT_2           K =1000
253 INT_POT_3           K =100
254 INT_POT_4           K =10
255 INT_POT_5           K =1
256 P_INT_REAL_INIT     LD TO_BE_PRINTED       ;; PRIMEIRO CHAR
257                     MM TMP_1
258                     /  INT_POT_1
259                     +  ZERO_BASE
260                     PD /100                    ;; imprime
261                     LD TMP_1
262                     /  INT_POT_1
263                     *  INT_POT_1
264                     MM TMP_2
265                     LD TMP_1
266                     -  TMP_2
267                     MM TMP_1
268                     /  INT_POT_2               ;; segundo char
269                     +  ZERO_BASE
270                     PD /100                    ;; imprime
271                     LD TMP_1
272                     /  INT_POT_2
```

```
273                     *  INT_POT_2
274                     MM TMP_2
275                     LD TMP_1
276                     -  TMP_2
277                     MM TMP_1
278                     /  INT_POT_3                ;; terceiro char
279                     +  ZERO_BASE
280                     PD /100                     ;; imprime
281                     LD TMP_1
282                     /  INT_POT_3
283                     *  INT_POT_3
284                     MM TMP_2
285                     LD TMP_1
286                     -  TMP_2
287                     MM TMP_1
288                     /  INT_POT_4                ;; quarto char
289                     +  ZERO_BASE
290                     PD /100                     ;; imprime
291                     LD TMP_1
292                     /  INT_POT_4
293                     *  INT_POT_4
294                     MM TMP_2
295                     LD TMP_1
296                     -  TMP_2
297                     MM TMP_1
298                     /  INT_POT_5                ;; quinto char
299                     +  ZERO_BASE
300                     PD /100                     ;; imprime
301                     LD TMP_1
302                     /  INT_POT_5
303                     *  INT_POT_5
304                     MM TMP_2
305                     LD TMP_1
306                     -  TMP_2
307                     MM TMP_1
308                     RS P_INT_ZERO
309
310 # START_STD_LIB_PADDING
```