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 (ALFRED; SETHI; JEFFREY, 1986).

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 (NETO, 1987), 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

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.

2.1 Estrutura de controle de fluxo: IF

```
1          LD {R0} # carrega a primeira referencia
2          JZ {L0} # se 0, entao pula para L0 (else)
3          {C0}    # codigo if, C pode ser nulo ou mais coisas
4 {L0}     {N}     # N executa somente a expansao
```

2.2 Estrutura de controle de fluxo: IF-ELSE

```
1          LD {R0} # carrega a primeira referencia
2          JZ {L0} # se 0, entao pula para L0 (else)
3          {C0}    # codigo if, C pode ser nulo ou mais coisas
4          JP {L1} # codigo fim, pula para fim
5 {L0}     {C2}    # codigo else
6 {L1}     {N}     # N executa somente a expansao
```

2.3 Estrutura de controle de fluxo: WHILE

```
1        LD {R0}
2        JZ {L0}
3        {C0}
4 {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 {L0}
```

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  :               <   WORD_TO_GET
2  :               <   STORE_PTR
3  :               <   GET_WORD_LOCAL_VAR
```

```
4
5                          {C0}        # argumentos
6  {R0}                    JP /000     # label funcao
7                          {C1}        # salva valores na pilha
8                                      # LV =0
9                                      # MM WORD_TO_SAVE
10                                     # LV EXAMPLE_STACK_ARG
11                                     # MM ORIGIN_PTR
12                                     # SC SAVE_WORD_TO_LOCAL_VAR
13                         {C2}        # Corpo da funcao
14                                     # como carregar valor da pilha:
15                                     # ------------------------
16                                     # LV =0
17                                     # MM WORD_TO_GET
18                                     # LV EXAMPLE_STACK_ARG
19                                     # MM STORE_PTR
20                                     # SC GET_WORD_LOCAL_VAR
21                         {C3}        # pega valores da pilha
22                         LV {R0}
23                         MM POP_CALL_FCT
24                         SC POP_CALL
25                         RS {R0}
26 RETURN_EXAMPLE_STACK    LV EXAMPLE_STACK
27                         MM POP_CALL_FCT
28                         SC POP_CALL ;; trickery!
29
30                         SC PRINT_STACK_ADDRS    ;; deve imprimir 0
                              fff
31                         RS EXAMPLE_STACK
```

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

```
1  :    < PUSH_CALL_SIZELV
2  :    < PUSH_CALL
3       LV {R0}                # carrega o tamanho dos argumentos,
           temporarios e tudo mais
4       MM PUSH_CALL_SIZELV    #
5       {C0}                   # passa argumentos
6       SC PUSH_CALL           # primeiro dah push
```

```
7       SC {R1}                 # depois chama..
```

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 define cada uma das operações da pilha, como *push* e *pop*. Baseado nesse conceito de código alinhavado utilizado pela MVN, 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.

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.

5 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.

5.1 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 }
```

5.2 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  PUSH_CALL_SIZELV        <
8  PUSH_CALL_RET_ADDRS     <
9  PUSH_CALL_TMP_SZ        <
10 PUSH_CALL_PAR_SZ        <
11 PRINT_STACK_ADDRS       <
12 POP_CALL                <
13 READ_INT                <
14 STORE_PTR               <
15 GETS                    <
16 STORE_PTR_IO            <
17 WORD_TO_SAVE            <
18 WORD_TO_GET             <
19 GET_WORD_LOCAL_VAR      <
20 SAVE_WORD_TO_LOCAL_VAR <
21 ORIGIN_PTR <
22 POP_CALL_FCT <
23 PUSH_ARITH              <
24 POP_ARITH               <
25 SUM_ARITH               <
26 SUB_ARITH               <
27 DIV_ARITH               <
28 MUL_ARITH               <
29 AND_ARITH               <
30 OR_ARITH                <
31 NOT_ARITH               <
32 @ /0000
33 CZAR_INICIO_CODE        JP CZAR_INICIO
34 CONST_VAR_0             K =6   ; const int fat_10_rec = 10;
35 GLOBAL_VAR_0           K =0   ; decl int retorno;
36 ONE                     K /001
37
38 DBG_TMP                 K /000
39
40 FUNCTION_0_RETURN       K =0
41 FUNCTION_0_ARG_0        K =0   ; int n
```

```
42 FUNCTION_0_TMP_0          K =0
43 FUNCTION_0_LOCAL_VAR_0    K =1   ;      decl int retorno = 1;
44 FUNCTION_0                JP /000 ; int fatorial_recursivo(int n)
       {
45                           ; SC PRINT_STACK_ADDRS
46                            ;;;; PREAMBULO
47                           LV =0
48                           MM WORD_TO_SAVE
49                           LV FUNCTION_0_ARG_0
50                           MM ORIGIN_PTR
51                           SC SAVE_WORD_TO_LOCAL_VAR ;; vai pra
                                 pilha
52                            ;;;; FIM PREAMBULO
53
54                           LV =1
55                           MM FUNCTION_0_LOCAL_VAR_0
56
57                           LV =1
58                           MM WORD_TO_SAVE
59                           LV FUNCTION_0_LOCAL_VAR_0
60                           MM ORIGIN_PTR
61                           SC SAVE_WORD_TO_LOCAL_VAR ;; vai pra
                                 pilha
62
63
64                           LD FUNCTION_0_ARG_0 ; n >= 1
65                           SC PUSH_ARITH
66                           LD ONE
67                           SC PUSH_ARITH
68                           SC SUB_ARITH
69                           SC POP_ARITH
70
71
72
73
74                           JN FUNCTION_0_LABEL_0 ; if (n >= 1) {
75
76                           LV =0
77                           MM WORD_TO_GET
```

```
78                          LV FUNCTION_0_ARG_0
79                          MM STORE_PTR
80                          SC GET_WORD_LOCAL_VAR
81
82                          LD FUNCTION_0_ARG_0 ; n - 1
83                          SC PUSH_ARITH
84                          LD ONE
85                          SC PUSH_ARITH
86                          SC SUB_ARITH
87                          SC POP_ARITH
88
89
90                          MM FUNCTION_0_ARG_0
91                          LV =2
92                          MM PUSH_CALL_SIZELV
93                          SC PUSH_CALL
94                          SC FUNCTION_0 ; call fatorial_recursivo
                               (n - 1);
95                          LD FUNCTION_0_RETURN
96                          MM FUNCTION_0_TMP_0
97
98
99                          LV =0
100                         MM WORD_TO_GET
101                         LV FUNCTION_0_ARG_0
102                         MM STORE_PTR
103                         SC GET_WORD_LOCAL_VAR
104
105                         LD FUNCTION_0_ARG_0 ; n * call
                               fatorial_recursivo (n - 1);
106
107                         SC PUSH_ARITH
108                         LD FUNCTION_0_TMP_0
109                         SC PUSH_ARITH
110                         SC MUL_ARITH
111
112                         SC POP_ARITH ; set retorno = *
113                         MM FUNCTION_0_LOCAL_VAR_0
114
```

```
115                             LV =1
116                             MM WORD_TO_SAVE
117                             LV FUNCTION_0_LOCAL_VAR_0
118                             MM ORIGIN_PTR
119                             SC SAVE_WORD_TO_LOCAL_VAR ;; vai pra
                                    pilha
120
121 FUNCTION_0_LABEL_0          * ONE
122                             LV =1
123                             MM WORD_TO_GET
124                             LV FUNCTION_0_LOCAL_VAR_0
125                             MM STORE_PTR
126                             SC GET_WORD_LOCAL_VAR
127
128                             LD FUNCTION_0_LOCAL_VAR_0
129                             MM FUNCTION_0_RETURN
130
131                             LV FUNCTION_0
132                             MM POP_CALL_FCT
133                             SC POP_CALL ;; trickery!
134                             RS FUNCTION_0    ;      return retorno;
135                             K /7777
136
137 CTR                         K =0
138 DBG                         * ONE
139                             LV /003C
140                             PD /100
141                             LD CTR
142                             MM TO_BE_PRINTED
143                             SC P_INT_ZERO
144                             LV /003E
145                             PD /100
146                             LV /000A
147                             PD /100
148                             LD CTR
149                             + ONE
150                             MM CTR
151                             RS DBG
152
```

```
153
154 ;; INIT PROG ==========================================================
155 CZAR_INICIO          * ONE ;; stub instruction
156                      ; SC PRINT_STACK_ADDRS
157                       LD CONST_VAR_0
158                      MM FUNCTION_0_ARG_0
159                       LV =2
160                      MM PUSH_CALL_SIZELV
161
162                       SC PUSH_CALL          ;; primeiro dah
                             push
163                       SC FUNCTION_0         ;; depois chama..
164                       LD FUNCTION_0_RETURN
165                      MM GLOBAL_VAR_0
166
167                       LD GLOBAL_VAR_0
168                      MM TO_BE_PRINTED
169                       SC P_INT_ZERO
170
171 FIM                  HM /00
172 # CZAR_INICIO_CODE
```

5.3 Tradução do programa fatorial para linguagem de saída MVN

```
1  P_STRING              <
2  STRING_PTR            <
3  P_INT_ZERO            <
4  TO_BE_PRINTED         <
5  P_LINE                <
6  PUSH_CALL             <
7  PUSH_CALL_SIZELV      <
8  PUSH_CALL_RET_ADDRS   <
9  PUSH_CALL_TMP_SZ      <
10 PUSH_CALL_PAR_SZ      <
11 PRINT_STACK_ADDRS     <
12 POP_CALL              <
13 READ_INT              <
14 STORE_PTR             <
15 GETS                  <
```

```
16 STORE_PTR_IO            <
17 WORD_TO_SAVE            <
18 WORD_TO_GET             <
19 GET_WORD_LOCAL_VAR      <
20 SAVE_WORD_TO_LOCAL_VAR <
21 ORIGIN_PTR <
22 POP_CALL_FCT <
23 PUSH_ARITH              <
24 POP_ARITH               <
25 SUM_ARITH               <
26 SUB_ARITH               <
27 DIV_ARITH               <
28 MUL_ARITH               <
29 AND_ARITH               <
30 OR_ARITH                <
31 NOT_ARITH               <
32 @ /0000
33 CZAR_INICIO_CODE          JP CZAR_INICIO
34 CONST_VAR_0               K =6   ; const int fat_10_rec = 10;
35 GLOBAL_VAR_0              K =0   ; decl int retorno;
36 ONE                       K /001
37
38 DBG_TMP                   K /000
39
40 FUNCTION_0_RETURN         K =0
41 FUNCTION_0_ARG_0          K =0   ; int n
42 FUNCTION_0_TMP_0          K =0
43 FUNCTION_0_LOCAL_VAR_0    K =1   ;      decl int retorno = 1;
44 FUNCTION_0                JP /000 ; int fatorial_recursivo(int n)
       {
45                          ; SC PRINT_STACK_ADDRS
46                           ;;;; PREAMBULO
47                          LV =0
48                          MM WORD_TO_SAVE
49                          LV FUNCTION_0_ARG_0
50                          MM ORIGIN_PTR
51                          SC SAVE_WORD_TO_LOCAL_VAR ;; vai pra
                                pilha
52                           ;;;; FIM PREAMBULO
```

```
53
54                          LV =1
55                          MM FUNCTION_0_LOCAL_VAR_0
56
57                          LV =1
58                          MM WORD_TO_SAVE
59                          LV FUNCTION_0_LOCAL_VAR_0
60                          MM ORIGIN_PTR
61                          SC SAVE_WORD_TO_LOCAL_VAR ;; vai pra
                                pilha
62
63
64                          LD FUNCTION_0_ARG_0 ; n >= 1
65                          SC PUSH_ARITH
66                          LD ONE
67                          SC PUSH_ARITH
68                          SC SUB_ARITH
69                          SC POP_ARITH
70
71
72
73
74                          JN FUNCTION_0_LABEL_0 ; if (n >= 1) {
75
76                          LV =0
77                          MM WORD_TO_GET
78                          LV FUNCTION_0_ARG_0
79                          MM STORE_PTR
80                          SC GET_WORD_LOCAL_VAR
81
82                          LD FUNCTION_0_ARG_0 ; n - 1
83                          SC PUSH_ARITH
84                          LD ONE
85                          SC PUSH_ARITH
86                          SC SUB_ARITH
87                          SC POP_ARITH
88
89
90                          MM FUNCTION_0_ARG_0
```

```
91                              LV =2
92                              MM PUSH_CALL_SIZELV
93                              SC PUSH_CALL
94                              SC FUNCTION_0 ; call fatorial_recursivo
                                    (n - 1);
95                              LD FUNCTION_0_RETURN
96                              MM FUNCTION_0_TMP_0
97
98
99                              LV =0
100                             MM WORD_TO_GET
101                             LV FUNCTION_0_ARG_0
102                             MM STORE_PTR
103                             SC GET_WORD_LOCAL_VAR
104
105                             LD FUNCTION_0_ARG_0 ; n * call
                                    fatorial_recursivo (n - 1);
106
107                             SC PUSH_ARITH
108                             LD FUNCTION_0_TMP_0
109                             SC PUSH_ARITH
110                             SC MUL_ARITH
111
112                             SC POP_ARITH ; set retorno = *
113                             MM FUNCTION_0_LOCAL_VAR_0
114
115                             LV =1
116                             MM WORD_TO_SAVE
117                             LV FUNCTION_0_LOCAL_VAR_0
118                             MM ORIGIN_PTR
119                             SC SAVE_WORD_TO_LOCAL_VAR ;; vai pra
                                    pilha
120
121 FUNCTION_0_LABEL_0         * ONE
122                             LV =1
123                             MM WORD_TO_GET
124                             LV FUNCTION_0_LOCAL_VAR_0
125                             MM STORE_PTR
126                             SC GET_WORD_LOCAL_VAR
```

```
127
128                             LD FUNCTION_0_LOCAL_VAR_0
129                             MM FUNCTION_0_RETURN
130
131                             LV FUNCTION_0
132                             MM POP_CALL_FCT
133                             SC POP_CALL ;; trickery!
134                             RS FUNCTION_0    ;      return retorno;
135                             K   /7777
136
137 CTR                         K =0
138 DBG                         * ONE
139                             LV /003C
140                             PD /100
141                             LD CTR
142                             MM TO_BE_PRINTED
143                             SC P_INT_ZERO
144                             LV /003E
145                             PD /100
146                             LV /000A
147                             PD /100
148                             LD CTR
149                             +  ONE
150                             MM CTR
151                             RS DBG
152
153
154 ;; INIT PROG ==========================================================
155 CZAR_INICIO                 * ONE ;; stub instruction
156                            ; SC PRINT_STACK_ADDRS
157                             LD CONST_VAR_0
158                             MM FUNCTION_0_ARG_0
159                             LV =2
160                             MM PUSH_CALL_SIZELV
161
162                             SC PUSH_CALL             ;; primeiro dah
                                push
163                             SC FUNCTION_0         ;; depois chama..
164                             LD FUNCTION_0_RETURN
```

```
165                                 MM GLOBAL_VAR_0
166
167                                 LD  GLOBAL_VAR_0
168                                 MM TO_BE_PRINTED
169                                 SC  P_INT_ZERO
170
171 FIM                             HM  /00
172 # CZAR_INICIO_CODE
```

Referências

ALFRED, V.; SETHI, R.; JEFFREY, D. *Compilers: principles, techniques and tools.* [S.l.]: Addison-Wesley, 1986.

NETO, J. J. *Introdução à Compilação.* [S.l.]: LTC, 1987. (ENGENHARIA DE COMPUTAÇÃO).

Apêndices

APÊNDICE A – Biblioteca auxiliar *std*

```
1  ; Biblioteca de ambiente
2  ; ========================
3  ; exports
4  PUSH_CALL_SIZELV          >
5  PUSH_CALL_RET_ADDRS       >
6  PUSH_CALL_TMP_SZ          >
7  PUSH_CALL_PAR_SZ          >
8  PUSH_CALL                 >
9  POP_CALL                  >
10 SAVE_WORD_TO_LOCAL_VAR    >
11 WORD_TO_SAVE              >
12 ORIGIN_PTR                >
13 GET_WORD_LOCAL_VAR        >
14 WORD_TO_GET               >
15 STORE_PTR                 >
16 PRINT_STACK_ADDRS         >
17 POP_CALL_FCT              >
18 EXAMPLE_STACK             >
19 EXAMPLE_STACK_ARG         >
20 ;; ARITH:
21 PUSH_ARITH                >
22 POP_ARITH                 >
23 SUM_ARITH                 >
24 SUB_ARITH                 >
25 DIV_ARITH                 >
26 MUL_ARITH                 >
27 AND_ARITH                 >
28 OR_ARITH                  >
29 NOT_ARITH                 >
30 ;; imports
31 READ_INT                  <
32 PONEASCII                 <
33 TO_BE_PRINTED             <
34 P_STRING                  <
35 STRING_PTR                <
```

```
36 P_INT_ZERO                <
37 P_LINE                    <
38 STORE_PTR_IO              <
39 ; relocavel
40 & /0000
41 START_STD_LIB_PADDING K /0000
42 ; alocando para K
43 SHIFT_BYTE           K  /0100
44 LOAD_CONST           K /8000    ; instruction to load
45 LOADV_CONST          K /3000    ; instruction to load
46 MOVE_CONST           K /9000    ; instruction to move to memory
47 ZERO                 K /0000
48 TWO                  K /0002
49 ONE                  K /0001
50 TEN                  K =10
51 TMP_1                K /0000
52 TMP_2                K /0000
53 TMP_3                K /0000
54 TMP_4                K /0000
55 ;; VARIAVEIS GLOBAIS
56 ;; comeco da pilha = FFF
57 ;; tamanho da pilha = 2FF
58 ;;    | ptr to old_stack_head  | \___ STACK_PTR
59 ;;    |       savedregist      |
60 ;;    |          ...           |
61 ;;    |       local var        |
62 ;;    |          ...           |
63 ;;    |       temporaries      |
64 ;;    |       parameters       |
65 ;;    |          ...           |
66 ;;    |      ref parameters    |  _____ OLD STACK_PTR
67 ;;    |       returnaddrs      | /      (STACK_PTR points here)
68 STACK_PTR            K /0FFF
69 ;;STACK_FRAME         K /0FFF
70
71 ;; FIM VARIAVEIS GLOBAIS
72
73 ;; **** P_LINE ****
74 ;; imprime uma linha
```

```
75  ;;
76
77  COUNT_IS        K /436f
78                  K /756e
79                  K /7465
80                  K /7220
81                  K /6973
82                  K /3a00
83                  K /0000
84
85  EXAMPLE_STACK_ARG   K  /0000
86  EXAMPLE_STACK       JP /000
87                      SC PRINT_STACK_ADDRS   ;; deve imprimir 0fff
88                      ;;; SALVAR ARGUMENTOS na pilha
89                      LV =0
90                      MM WORD_TO_SAVE
91                      LV EXAMPLE_STACK_ARG
92                      MM ORIGIN_PTR
93                      SC SAVE_WORD_TO_LOCAL_VAR
94                      ;;;; CORPO DA FUNCAO
95                      ;;; CARREGANDO UM VALOR DA PILHA
96                      LV =0
97                      MM WORD_TO_GET
98                      LV EXAMPLE_STACK_ARG
99                      MM STORE_PTR
100                     SC GET_WORD_LOCAL_VAR
101                     ;;; IMPRIME
102                     LV COUNT_IS
103                     MM STRING_PTR
104                     SC P_STRING  ;; inline fct, no need to stack
105                     LD EXAMPLE_STACK_ARG
106                     MM TO_BE_PRINTED
107                     SC P_INT_ZERO
108                     SC P_LINE
109
110                     LD EXAMPLE_STACK_ARG
111                     JZ RETURN_EXAMPLE_STACK
112
113                     LD EXAMPLE_STACK_ARG
```

```
114                         - ONE
115                        MM EXAMPLE_STACK_ARG
116
117                         LV =1
118                        MM PUSH_CALL_SIZELV
119                         LV =0
120                        MM PUSH_CALL_RET_ADDRS
121                         LV =0
122                        MM PUSH_CALL_TMP_SZ
123                         LV =0
124                        MM PUSH_CALL_PAR_SZ
125                         SC PUSH_CALL
126
127                         SC EXAMPLE_STACK  ;; chamada recursiva
128                         ;;;; FIM DO CORPO DA FUNCAO
129 RETURN_EXAMPLE_STACK LV EXAMPLE_STACK
130                        MM POP_CALL_FCT
131                         SC POP_CALL ;; trickery!
132
133                         SC PRINT_STACK_ADDRS   ;; deve imprimir 0fff
134                         RS EXAMPLE_STACK
135
136
137 ; MISCELANEA:
138 ;; *** HIGH_LOW HIGH_V ***
139 HIGH_V                  JP /000
140 LOW_V                   JP /000
141 HIGH_LOW                JP /000
142                         LD HIGH_V
143                        MM LOW_V
144                         LD LOW_V
145                         * SHIFT_BYTE
146                         / SHIFT_BYTE
147                        MM LOW_V
148                         LD HIGH_V
149                         / SHIFT_BYTE
150                        MM HIGH_V
151
152                         LD LOW_V
```

```
153                         /   SHIFT_BYTE
154                         *   SHIFT_BYTE
155                         -   LOW_V
156                         MM  LOW_V
157                         LD  ZERO
158                         -   LOW_V
159                         MM  LOW_V
160
161                         LD  HIGH_V
162                         /   SHIFT_BYTE
163                         *   SHIFT_BYTE
164                         -   HIGH_V
165                         MM  HIGH_V
166                         LD  ZERO
167                         -   HIGH_V
168                         MM  HIGH_V
169
170                         RS  HIGH_LOW
171
172 ;; ****  PUSH_CALL PUSH_CALL_SIZELV
173 ;;       PUSH_CALL_RET_ADDRS PUSH_CALL_TMP_SZ
174 ;;       PUSH_CALL_PAR_SZ ****
175 ;;     Empilha um espaco de memoria
176 ;; para a pilha.
177
178 PUSH_CALL_SIZELV        K /000
179 PUSH_CALL_RET_ADDRS     K /000
180 PUSH_CALL_TMP_SZ        K /000
181 PUSH_CALL_PAR_SZ        K /000
182
183 ;;  PUSH_CALL
184 ;;  CALLEE_FCT
185 ;;     POP_CALL
186
187 PUSH_CALL               JP  /000
188                         LD  PUSH_CALL  ;; get return addrs
189                         +   TWO ;; return address of the callee
190                         +   LOADV_CONST
191                         MM  LOAD_RETURN_ADDRS
```

```
192                         LD STACK_PTR
193                         - TWO          ;; new return addrs
194                         + MOVE_CONST
195                         MM MOVE_RETURN_ADDRS
196 LOAD_RETURN_ADDRS       JP /000
197 MOVE_RETURN_ADDRS       JP /000    ;; return addrs salvo
198                         LD STACK_PTR
199                         - TWO
200                         - TWO
201                         - PUSH_CALL_SIZELV
202                         - PUSH_CALL_RET_ADDRS
203                         - PUSH_CALL_TMP_SZ
204                         - PUSH_CALL_PAR_SZ
205                         - TWO  ;; return addrs
206                         MM TMP_1
207                         LD TMP_1
208                         + MOVE_CONST
209                         MM MRKR_PC_SAVE_HEAD
210                         LD STACK_PTR
211 MRKR_PC_SAVE_HEAD       JP /000
212                         LD TMP_1
213                         MM STACK_PTR
214                         RS PUSH_CALL
215 ;; **** POP_CALL ****
216                         K /6666
217 POP_CALL_FCT            K /0000
218 POP_CALL                JP /000 ; retorno
219 ;;                        SC PRINT_STACK_ADDRS ;; TODO remove
220 POP_CALL_INIT           LD STACK_PTR
221                         + LOAD_CONST
222                         MM MRKR_PC_LOAD_HEAD
223 MRKR_PC_LOAD_HEAD       JP /000
224                         MM STACK_PTR
225                         LD STACK_PTR
226                         - TWO
227                         + LOAD_CONST
228                         MM LOAD_RETURN_ADDRS_2
229                         LD POP_CALL_FCT
230                         + MOVE_CONST
```

```
231                         MM MOVE_RETURN_ADDRS_2
232 LOAD_RETURN_ADDRS_2 JP /000
233 MOVE_RETURN_ADDRS_2 JP /000   ;; engana a funcao para ela pensar
        que ela
234                                ;; tem que retornar para esse valor
235                         RS POP_CALL
236
237 ;; **** SAVE_WORD_TO_LOCAL_VAR WORD_TO_SAVE ORIGIN_PTR ****
238
239
240 WORD_TO_SAVE              K /000
241 ORIGIN_PTR                K /000
242
243 SAVE_WORD_TO_LOCAL_VAR      JP /000
244                             LD STACK_PTR
245                             + TWO           ;; first word
246                             + WORD_TO_SAVE
247                             + WORD_TO_SAVE   ;; WORD_TO_GET * 2
248                             + MOVE_CONST    ;;
249                             MM MOVE_WORD_LOCAL_VAR_2
250                             LD ORIGIN_PTR
251                             + LOAD_CONST
252                             MM LOAD_WORD_LOCAL_VAR_2
253 LOAD_WORD_LOCAL_VAR_2       JP /000 ;; 8FROMPTR
254 MOVE_WORD_LOCAL_VAR_2       JP /000 ;; 9TOPTR
255                             RS SAVE_WORD_TO_LOCAL_VAR
256
257 ;; **** GET_WORD_LOCAL_VAR WORD_TO_GET STORE_PTR ****
258
259
260 WORD_TO_GET           K /000
261 STORE_PTR             K /000
262
263 GET_WORD_LOCAL_VAR        JP /000
264                           LD STACK_PTR
265                           + TWO           ;; first word
266                           + WORD_TO_GET
267                           + WORD_TO_GET   ;; WORD_TO_GET * 2
268                           + LOAD_CONST    ;;
```

```
269                                 MM LOAD_WORD_LOCAL_VAR
270                                 LD STORE_PTR
271                                 + MOVE_CONST
272                                 MM MOVE_WORD_LOCAL_VAR
273 LOAD_WORD_LOCAL_VAR             JP /000 ;; 8FROMPTR
274 MOVE_WORD_LOCAL_VAR             JP /000 ;; 9TOPTR
275                                 RS GET_WORD_LOCAL_VAR
276
277 STACK_WAS_STR K   /5374
278                K   /6163
279                K   /6b20
280                K   /0077
281                K   /6173
282                K   /3a20
283                K   /0000
284
285 PRINT_STACK_ADDRS JP /000
286                    LV STACK_WAS_STR   ; carrega o ponteiro da
                          string
287                    MM STRING_PTR ; copia o ponteiro para o
                          parametro
288                    SC P_STRING    ; roda p_string
289                    LD STACK_PTR
290                    MM TO_BE_PRINTED
291                    SC P_INT_ZERO
292                    SC P_LINE
293                    RS PRINT_STACK_ADDRS
294
295
296 ;; pilha aritim tica:
297 ARIT_STACK_ZERO K /0000    ;; 0
298                 K /0000    ;; 1
299                 K /0000    ;; 2
300                 K /0000    ;; 3
301                 K /0000    ;; 4
302                 K /0000    ;; 5
303                 K /0000    ;; 6
304                 K /0000    ;; 7
305                 K /0000    ;; 8
```

```
306                     K /0000     ;; 9
307                     K /0000     ;; 10
308                     K /0000     ;; 11
309                     K /0000     ;; 12
310                     K /0000     ;; 13
311                     K /0000     ;; 14
312                     K /0000     ;; 15
313                     K /0000     ;; 16
314                     K /0000     ;; 17
315                     K /0000     ;; 18
316                     K /0000     ;; 19
317                     K /0000     ;; 20
318
319 ARIT_PTR_STACK     K ARIT_STACK_ZERO
320
321 PUSH_ARITH         JP /000
322                     MM TMP_1
323                     LD ARIT_PTR_STACK
324                     + ONE
325                     + ONE
326                     MM ARIT_PTR_STACK
327                     + MOVE_CONST
328                     MM OP_PUSH_ARITH
329                     LD TMP_1
330 OP_PUSH_ARITH      JP /000
331                     RS PUSH_ARITH
332
333
334 POP_ARITH          JP /000
335                     LD ARIT_PTR_STACK
336                     - ONE
337                     - ONE
338                     MM ARIT_PTR_STACK
339                     + ONE
340                     + ONE
341                     + LOAD_CONST
342                     MM OP_POP_ARITH
343 OP_POP_ARITH       JP /000
344                     RS POP_ARITH
```

```
345
346 ;;; BIN OPER:
347
348 SUM_ARITH       JP /000
349                 SC POP_ARITH
350                 MM TMP_2
351                 SC POP_ARITH
352                 + TMP_2
353                 SC PUSH_ARITH
354                 RS SUM_ARITH
355
356 SUB_ARITH       JP /000
357                 SC POP_ARITH
358                 MM TMP_2
359                 SC POP_ARITH
360                 - TMP_2
361                 SC PUSH_ARITH
362                 RS SUB_ARITH
363
364 DIV_ARITH       JP /000
365                 SC POP_ARITH
366                 MM TMP_2
367                 SC POP_ARITH
368                 / TMP_2
369                 SC PUSH_ARITH
370                 RS DIV_ARITH
371
372 MUL_ARITH       JP /000
373                 SC POP_ARITH
374                 MM TMP_2
375                 SC POP_ARITH
376                 * TMP_2
377                 SC PUSH_ARITH
378                 RS MUL_ARITH
379
380 AND_ARITH       JP /000
381                 SC POP_ARITH
382                 MM TMP_2
383                 SC POP_ARITH
```

```
384                     JZ PUSH_ZERO_AND_ARITH
385                     LD TMP_2
386                     JZ PUSH_ZERO_AND_ARITH
387                     LV /001
388                     SC PUSH_ARITH
389                     RS AND_ARITH
390 PUSH_ZERO_AND_ARITH  LV /000
391                       SC PUSH_ARITH
392                       RS AND_ARITH
393
394 OR_ARITH            JP /000
395                     SC POP_ARITH
396                     MM TMP_2
397                     SC POP_ARITH
398                     JZ ONE_ZERO_OR_ARITH
399                     LV /001
400                     SC PUSH_ARITH
401                     RS OR_ARITH
402 ONE_ZERO_OR_ARITH LD TMP_2
403                       JZ PUSH_ZERO_OR_ARITH
404                       LV /001
405                       SC PUSH_ARITH
406                       RS OR_ARITH
407 PUSH_ZERO_OR_ARITH LV /000
408                        SC PUSH_ARITH
409                        RS OR_ARITH
410
411 ;; UNARY OPER:
412
413 NOT_ARITH             JP /000
414                       SC POP_ARITH
415                       JZ PUSH_ONE_NOT_ARITH
416                       LV /000
417                       SC PUSH_ARITH
418                       RS NOT_ARITH
419 PUSH_ONE_NOT_ARITH  LV /001
420                       SC PUSH_ARITH
421                       RS NOT_ARITH
422 # START_STD_LIB_PADDING
```

APÊNDICE B – Biblioteca auxiliar *stdio*

```
1  PONEASCII                >
2  TO_BE_PRINTED            >
3  P_STRING                 >
4  STRING_PTR               >
5  P_INT_ZERO               >
6  P_LINE                   >
7  STORE_PTR_IO             >
8  READ_INT                 >
9  GETS                     >
10
11
12
13 & /0000
14 START_STD_LIB_PADDING K /0000
15
16 SHIFT_BYTE          K /0100
17 TO_BE_PRINTED       K /0000
18 TO_BE_PRINTED_TMP   K /0000
19 LOAD_CONST          K /8000   ; instruction to load
20 LOADV_CONST         K /3000   ; instruction to load
21 MOVE_CONST          K /9000   ; instruction to move to memory
22 ZERO                K /0000
23 TWO                 K /0002
24 ONE                 K /0001
25 TEN                 K =10
26 STRING_PTR          K /0000   ; ponteiro para o comeco da string
27 TMP_1               K /0000
28 TMP_2               K /0000
29 TMP_3               K /0000
30 TMP_4               K /0000
31
32
33 ; MISCELANEA:
34 ;; *** HIGH_LOW HIGH_V ***
35 HIGH_V              JP /000
```

```
36 LOW_V          JP /000
37 HIGH_LOW       JP /000
38                LD HIGH_V
39                MM LOW_V
40                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
       endere o 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
       endere o 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            ; endere o 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    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
```