5ª parte do projeto - tradução dos comandos

PCS2056 - Linguagens e Compiladores

Prof. Ricardo Rocha

Data: 29/11/2016

Felipe de Paiva Miranda 7630486
Thiago Ryu Niwa Murakami 7626689

1. Tradução de estruturas de controle de fluxo

1.1. If-then

Linguagem de Entrada	**Linguagem de Saída**

if (condicao) {
  # comandos
}
	LD condicao
	JZ END_IF
	<comandos>
	END_IF

1.2. If-then-else

Linguagem de Entrada	**Linguagem de Saída**

if (condicao) {
  #comandos_if
} else {
  #comandos_else
}
	LD condicao
	JZ ELSE
	<comandos_if>
	JP END_IF
	ELSE <comandos_else>
	END_IF

1.3. While

Linguagem de Entrada	**Linguagem de Saída**

while (condicao) {
  #comandos
}
	LOOP LD condicao
	JZ END_WHILE
	<comandos>
	JP LOOP
	END_WHILE

2. Tradução de comandos imperativos

2.1. Atribuição de valor

Linguagem de Entrada	**Linguagem de Saída**
identificador = valor;	identificador K=0 LD valor MM identificador

2.2. Leitura (entrada)

Linguagem de Entrada	**Linguagem de Saída**
int numero = read();	numero K =0 SC read LD read_number MM numero

Subrotina

```
read_number   K =0 ; Variável de retorno
r_negative    K =0 ; Número digitado é negativo
zero          K =0
one           K =1
num_256       K =256
minus_sign    K =45
ascii_cr      K /D
ascii_lf      K /A
ascii_offset  K /30
r_temp        K =0
r_temp2       K =0

read          JP /0000
              LD zero ; Inicialização
              MM read_number
              MM r_negative
              MM r_temp
              MM r_temp2
              GD /0000 ; Leitura de número negativo
              MM r_temp ; Guarda caracteres lidos
              / num_256
              MM r_temp2
              - minus_sign
              JZ is_negative ; Verifica se número digitado é negativo
              JP char1
is_negative   LD one ; Carrega o i_negative com FFFF
              MM r_negative
              JP char2
r_loop        GD /0000 ; Loop de leitura
              MM r_temp ; Guarda caracteres lidos
              / num_256
              MM r_temp2
char1         - ascii_cr ; Primeiro caractere
              JZ r_end ; Verifica se é o fim
              LD r_temp2
              - ascii_lf
              JZ r_end ; Verifica se é o fim
              LD read_number ; Não é o último caracter
              * ten ; Aumenta uma dezena no resultado
              MM read_number
              LD r_temp2 ; Converte caracter lido em número
              - ascii_offset
```

```
                + read_number
                MM read_number ; Atualiza o resultado de retorno
char2           LD r_temp2 ; Segundo caracter
                * num_256
                MM r_temp2
                LD r_temp
                - r_temp2
                MM r_temp2
                - ascii_cr ; Verifica se é o fim
                JZ r_end
                LD r_temp2
                - ascii_lf
                JZ r_end ; Verifica se é o fim
                LD read_number
                * ten
                MM read_number
                LD r_temp2 ; Converte caracter lido em número
                - ascii_offset
                + read_number
                MM input_number ; Atualiza o resultado de retorno
                JP i_loop ; Proximo caracter
r_end           LD r_negative ; Transforma em negativo se negativo
                JZ r_return
                LD zero
                - read_number
                MM read_number
i_return        LD read_number
                RS input ;
```

2.3. Impressão (saída)

Linguagem de Entrada	**Linguagem de Saída**
write(numero);	LD numero MM write_number SC write

Subrotina

```
write_number    K =0      ; Número a ser impresso
minus_one       K /FFFF ; valor -1
one             K =1      ; valor 1
ten             K =10     ; valor 10
minus_sign      K =45     ; Sinal de menos em ASCII
ascii_offset    K =48     ; Offset para o código de um número na tabela ASCII
w_temp1         K =0      ; Guarda o valor da última dezena
w_temp2         K =10      ; Indicador da dezena

write           JP /0000
                LD write_number
```

```
                JN w_negative       ; Número negativo
                JP w_start          ; Número posítivo
w_negative      LD minus_sign       ; imprime "-"
                PD /0100
                LD minus_one
                - write_number
                + one               ; inverte o número
                MM write_number
w_start         MM w_temp1
w_loop          LD write_number
                / w_temp2
                JZ w_print          ; É o número mais a esquerda?
                MM w_temp1          ; Guarda a última casa decimal visitada
                LD w_temp2
                * ten
                MM w_temp2          ; Próxima casa decimal
                JP w_loop
w_print         LD w_temp1          ; Número a ser impresso
                + ascii_offset
                PD /0100
                LD w_temp2
                / ten
                MM w_temp2
                - one
                JZ w_end            ; Verifica se é o último número
                LD w_temp1
                * w_temp2
                MM w_temp1
                LD write_number
                - w_temp1           ; Atualiza o número para impressão
                MM write_number
                MM w_temp1
                LD ten
                MM w_temp2          ; Próxima dezena
                JP w_loop           ; Próximo caractere
w_end           RS output
```

2.4. Chamada de subrotina

Linguagem de Entrada	**Linguagem de Saída**
funcao(a, b, c, ...)	LD func_size ; Carrega o tamanho do R.A. MM call_stack_size LD return_adr ; Carrega o endreço de retorno do R.A. MM call_stack_adr SC create_call_stack ; Cria R.A. LD 1 ; Carrega parâmetro da função

	MM arg_pos SC store_cs_pos SC função ; Executa a função

Subrotina do ambiente de execução

```
two                 K =2
STOP                K /0FF0
load_instruction    LD /0000   ; Instrução para o acesso indireto
store_instruction   MM /0000 ; Instrução para store indireto
arg_pos             K =0       ; Posição do argumento da função

load_cs_pos         JP /0000   ; Ponto de entrada da subrotina
                    LD STOP  ; Carrega topo da pilha do R.A.
                    - two ; Diminui um endereço na pilha do R.A.
                    - arg_pos ; Accumulador com o endereço correto
                    + load_instruction ; Cria nova instrução
                    MM instruct ; Armazena como proxima instrução
instruct            K /0 ; Reservado para guardar a instrução recém-montada
                    RS load_cs_pos
store_cs_pos        JP /0000
                    LD STOP ; Carrega topo da pilha do R.A.
                    - two ; Diminui um endereço na pilha do R.A.
                    - arg_pos ; Accumulador com o endereço correto
                    + store_instruction ; Here's the magic: Cria instrução nova!
                    MM instruct2 ; Armazena como proxima instrução
instruct2           K /0 ; Reservado para guardar a instrução recém-montada
                    RS store_cs_pos
call_stack_size     K =0
call_stack_adr      K =0400
create_call_stack   JP /0000
                    LD STOP
                    + two
                    MM STOP
                    LD zero
                    MM arg_pos
                    LD call_stack_adr
                    SC store_cs_pos
                    LD STOP
                    + call_stack_size
                    MM STOP
                    RS create_call_stack
```

3. Exemplo de programa traduzido

Na tabela abaixo, segue um programa exemplo que calcula o fatorial de um número digitado.

Cálculo de Fatorial

```
program {
        int fat;
        int num = read();

        if(num < 0){
                fat = 0;
        }else{
                fat = 1;
                while(num > 0){
                        fat = fat * num;
                        num = num -1;
                }
        }
        write(fat);
}
```

Na tabela abaixo, encontra-se o código gerado (programa traduzido) pelo compilador.

Cálculo de Fatorial (Programa traduzido)

```
one         K=1

            @ /0200
temp        K =0
fat         K =0 ; int fat
num         K =0 ; num = read()
            SC write
            LD read_number
            MM num
            LD num ; (num < 0)
            - zero
            JN TRUE1
FALSE1      LV =0
            JP END1
TRUE1       LV =1
END1        MM temp
            LD temp ; if () {} else{}
            JZ ELSE1
            LV =0 ; fat = 0
            MM fat
            JP END_IF1
ELSE1       LV =1 ; fat = 1
            MM fat
LOOP1       LV =0 ; while(num > 0) {}
            - num
            JN TRUE2
```

```
FALSE2      LV =0
            JP END2
TRUE2       LV =1
END2        MM temp
            LD temp
            JZ END_WHILE1
            LD fat ; fat = fat*num
            * num
            MM fat
            LD num ; num= num - 1
            - one
            MM num
            JP LOOP1
END_WHILE1 JP END_IF1
END_IF1     LD fat
            MM write_number
            SC write
            HM /00
```

Referências

1. Neto J. J. Introdução à Compilação. 1987.