Pontifícia Universidade Católica do Rio Grande do Sul

FELIPE FREITAS SILVA, LÁZARO MACIEL VASCONCELOS E LUIZA HELLER KROEFF PLÁ

RELATÓRIO: TRABALHO 2

Porto Alegre, Rio Grande do Sul

E-mails: Felipe Freitas Silva, f.freitas007@edu.pucrs.br | Lázaro Maciel Vasconcelos, lazaro.maciel@edu.pucrs.br | Luiza Heller Kroeff Plá, luiza.heller@edu.pucrs.br

2022

Sumário

1.Questão 1 3
1.1 Enunciado da Questão 1 3
1.2 Solução 4
1.3 Explicação Passo a passo do Código Acima: 8
1.4 Testes de Verificação 10
1.4.1Teste 1: frase: “Primeiro” 10
1.4.2Teste 2: frase: “Teste de 4 palavras” 12
1.4.3Teste 3: frase: “Frase de teste muito grande mesmo com mais de 10 palavras” 14
2.Questão 2 17
2.1 Enunciado da Questão 2 17
2.2 Solução em Código Assembly Viking 17
2.3 Explicação Passo a passo do Código Acima: 21
2.4 Testes de Verificação 23
2.4.1Teste 1: segmento 0 – 2 23
2.4.2Teste 2: 4 – 8 25
2.4.3Teste 3: 0 – 11 27
REFERÊNCIAS 29

1.Questão 1

1.1 Enunciado da Questão 1

Escreva um programa que conta o número de palavras armazenadas em uma String e apresenta o total no terminal. Como sugestão, utilize a função abaixo como referência para sua implementação, a ser chamada a partir do programa principal. Utilize duas Strings na demonstração do funcionamento.

```
int count_words(char *str, int size) {
    int i = 0, words = 0;
    while (1) {
        while ((str[i] < 33) || (str[i] > 126)) {
            if (i >= size)
                return words;
            i++;
        }
        while ((str[i] > 32) && (str[i] < 127)) {
            if (i >= size)
                return words + 1;
            i++;
        }
        words++;
    }
}
```

1.2 Solução

```
main
    ; r2 = totalPalavras(começa em 0)
    ldr r2,0

    ; r5 -> &frase
    ldi r5,frase

    ; salva r5 na pilha
    sub sp,2
    stw r5,sp

    ; r6 -> &count
    ldi r6,count

rep
    ; pega r5 da pilha
    ldw r5,sp
    add sp,2

    ; r4 = r5[i] (primeiro byte da frase)
    ldb r4,r5

    ; se não tiverem mais caracteres por ler, goTo(final)
    bez r4,print_final

    ; printa o byte em r4
    stw r4,0xf000

    ; r5 -> &mensagem + 1 (próximo byte)
    add r5,1
```

```
    ; salva r5 na pilha
    sub sp,2
    stw r5,sp

    ; r3 = 33 (primeiro caracter depois do " " na tabela ASCII)
    ldr r3,33

    ; r3 = r4 < r3 ? 1 : 0
    slt r3,r4,r3

    ; se r4 for um espaço, goTo(count)
    bnz r3,r6

    ; r3 = 126 (último caracter considerado como letra no exercício)
    ldr r3,126

    ; r3 = r4 < r3 ? 1 : 0
    slt r3,r4,r3

    ; se r4 não for uma letra, goTo(count)
    bez r3,r6

    ; repete o laço
    bnz r7,rep

count
    ; totalPalavras++
    add r2,1
```

```
    ; volta para o laço de repetição
    bnz r7,rep

print_final
    ; r4 -> &mensagem
    ldi r4,mensagem

print_rep
    ; r5 = r4[0] (primeiro byte da mensagem)
    ldb r5,r4

    ; print(r5)
    stw r5,0xf000

    ; r4 -> &mensagem + 1 (próximo byte)
    add r4,1

    ; se r5 não for mais caractere da frase, goTo(final)
    bez r5,final

    ; enquanto r5 for um caractere da frase, repete o laço
    bnz r5,print_rep

final
    ; totalPalavras++
    add r2,1

    ; print(totalPalavras)
    stw r2,0xf002
```

hcf

frase "Teste com 4 palavras"

mensagem "\nTotal de palavras:\t"

1.3 Explicação Passo a passo do Código Acima:

Código do método de rótulo: main:

1) Carrega-se o registrador r2 com o valor 0.
2) Carrega-se o endereço da variável mensagem no registrador r5.
3) Se vai para a próxima posição livre no registrador de armazenamento r7.
4) Carrega-se o valor do registrador r5, passo 2, na posição livre acessada.
5) O registrador r6 aponta para o método count.

Código do método de rótulo: rep:

6) Carrega-se o valor de r5 na posição livre do registrador de armazenamento r7.
7) Acessa-se a próxima posição livre do registrador de armazenamento.
8) Carrega-se o primeiro byte do registrador r5 no registrador r4.
9) Se o valor de r4 for igual a 0, ou seja, não for uma letra, ocorre um jump para o método de rótulo: print_final.
10) Printa-se o byte armazenado no registrador r4.
11) Adiciona-se 1 ao registrador r5, ou seja, aponta para a próxima letra ou espaço.
12) Acessa-se a próxima posição livre do registrador de armazenamento.
13) Armazena-se o valor de r5 em r7.
14) Guarda-se no registrador r3 o caractere de valor 33 da tabela ASCII.
15) Se o byte guardado em r4 for menor do que o byte guardado em r3, então o registrador r3 recebe o valor 1.
16) Se r3 for diferente de 0, em outras palavras, se o passo 14 for verdade, ocorre um jump para o endereço armazenado em r6 (método: count).
17) Guarda-se no registrador r3 o caractere de valor 126 da tabela ASCII.
18) Se o byte guardado em r4 for menor do que o byte guardado em r3, então o registrador r3 recebe o valor 1.
19) Se r3 for igual a 0, ou seja, o passo 17 for falso, ocorre um jump para o endereço armazenado em r6 (método: count).
20) Ocorre repetição do laço.

Código do método de rótulo: count:

21) Adiciona-se 1 ao registrador r2, ou seja, achou-se mais uma palavra.
22) Repetição do laço (de rótulo: rep).

Código do método de rótulo: print_final

23) O registrador r4 aponta para o endereço da variável mensagem.

Código do método de rótulo: print_rep

24) R5 vai receber o primeiro byte da mensagem, que está no registrador r4.
25) Printa o valor do registrador r5 (o próximo caractere da mensagem).
26) R4 aponta para o próximo byte da mensagem.
27) Se o registrador r5 não encontrar mais nenhum byte da mensagem, ocorre um jump para o método de rótulo: final.

28) Se o registrador r5 encontrar um byte de caractere na mensagem, ocorre repetição do laço.

Código do método de rótulo: final:

29) Conta mais uma palavra.
30) Printa o total de palavras.
31) Halt and catch fire.

1.4 Testes de Verificação

1.4.1Teste 1: frase: “Primeiro”

Tabela de símbolos:

```
Symbol table:
0000 main
000e rep
0038 count
0040 print_final
0044 print_rep
005a final
0064 frase
006e mensagem
```

Código de máquina:

```
Object code / disassembly:
0000 8a00   ldr r2,0
0002 9d00   ldc r5,0
0004 9d64   ldc r5,100
0006 6f02   sub r7,2
0008 50be   stw r0,r5,r7
000a 9e00   ldc r6,0
000c 9e38   ldc r6,56
000e 451e   ldw r5,r0,r7
0010 5f02   add r7,2
0012 0416   ldb r4,r0,r5
0014 9800   ldc r0,0
0016 9840   ldc r0,64
0018 c080   bez r0,r4,r0
001a 98f0   ldc r0,240
001c 9800   ldc r0,0
001e 5082   stw r0,r4,r0
0020 5d01   add r5,1
0022 6f02   sub r7,2
0024 50be   stw r0,r5,r7
0026 8b21   ldr r3,33
0028 338c   slt r3,r4,r3
002a d078   bnz r0,r3,r6
002c 8b7e   ldr r3,126
002e 338c   slt r3,r4,r3
0030 c078   bez r0,r3,r6
0032 9800   ldc r0,0
0034 980e   ldc r0,14
0036 d0e0   bnz r0,r7,r0
0038 5a01   add r2,1
003a 9800   ldc r0,0
003c 980e   ldc r0,14
003e d0e0   bnz r0,r7,r0
0040 9c00   ldc r4,0
0042 9c6e   ldc r4,110
0044 0512   ldb r5,r0,r4
0046 98f0   ldc r0,240
0048 9800   ldc r0,0
004a 50a2   stw r0,r5,r0
004c 5c01   add r4,1
004e 9800   ldc r0,0
0050 985a   ldc r0,90
0052 c0a0   bez r0,r5,r0
0054 9800   ldc r0,0
0056 9844   ldc r0,68
0058 d0a0   bnz r0,r5,r0
005a 5a01   add r2,1
005c 98f0   ldc r0,240
005e 9802   ldc r0,2
0060 5042   stw r0,r2,r0
0062 0003   ???
0064 5072   stw r0,r3,r4
0066 696d   sub r1,109
0068 6569   sbc r5,r3,r2
006a 726f   ???
006c 0000   and r0,r0,r0
006e 0a54   and r2,84
0070 6f74   sub r7,116
0072 616c   sub r1,r3,r3
0074 2064   xor r0,r3,r1
0076 6520   sub r5,r1,r0
0078 7061   ???
007a 6c61   sub r4,97
007c 7672   ???
007e 6173   ???
0080 3a09   slt r2,9
0082 0000   and r0,r0,r0
```

Dump de memória:

```
0000: 8a00 9d00 9d64 6f02 50be 9e00 9e38 451e |.....do.P....8E.|
0010: 5f02 0416 9800 9840 c080 98f0 9800 5082 |_......@......P.|
0020: 5d01 6f02 50be 8b21 338c d078 8b7e 338c |].o.P..!3..x.~3.|
0030: c078 9800 980e d0e0 5a01 9800 980e d0e0 |.x......Z.......|
0040: 9c00 9c6e 0512 98f0 9800 50a2 5c01 9800 |...n......P.\...|
0050: 985a c0a0 9800 9844 d0a0 5a01 98f0 9802 |.Z.....D..Z.....|
0060: 5042 0003 5072 696d 6569 726f 0000 0a54 |PB..Primeiro...T|
0070: 6f74 616c 2064 6520 7061 6c61 7672 6173 |otal de palavras|
0080: 3a09 0000 0000 0000 0000 0000 0000 0000 |:...............|
0090: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
00a0: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
00b0: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
00c0: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
00d0: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
00e0: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
00f0: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
0100: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
0110: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
0120: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
0130: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
0140: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
0150: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
```

Saída no terminal:

```
Assembling... done. Program size: 132 bytes (code + data).
Primeiro
Total de palavras: 1
Program halted at 0062.
```

1.4.2Teste 2: frase: “Teste de 4 palavras”

Tabela de símbolos:

```
Symbol table:
0000 main
000e rep
0038 count
0040 print_final
0044 print_rep
005a final
0064 frase
0078 mensagem
```

Código de máquina:

```
Object code / disassembly:
0000 8a00   ldr r2,0
0002 9d00   ldc r5,0
0004 9d64   ldc r5,100
0006 6f02   sub r7,2
0008 50be   stw r0,r5,r7
000a 9e00   ldc r6,0
000c 9e38   ldc r6,56
000e 451e   ldw r5,r0,r7
0010 5f02   add r7,2
0012 0416   ldb r4,r0,r5
0014 9800   ldc r0,0
0016 9840   ldc r0,64
0018 c080   bez r0,r4,r0
001a 98f0   ldc r0,240
001c 9800   ldc r0,0
001e 5082   stw r0,r4,r0
0020 5d01   add r5,1
0022 6f02   sub r7,2
0024 50be   stw r0,r5,r7
0026 8b21   ldr r3,33
0028 338c   slt r3,r4,r3
002a d078   bnz r0,r3,r6
002c 8b7e   ldr r3,126
002e 338c   slt r3,r4,r3
0030 c078   bez r0,r3,r6
0032 9800   ldc r0,0
0034 980e   ldc r0,14
0036 d0e0   bnz r0,r7,r0
0038 5a01   add r2,1
003a 9800   ldc r0,0
003c 980e   ldc r0,14
003e d0e0   bnz r0,r7,r0
0040 9c00   ldc r4,0
0042 9c78   ldc r4,120
0044 0512   ldb r5,r0,r4
0046 98f0   ldc r0,240
0048 9800   ldc r0,0
004a 50a2   stw r0,r5,r0
004c 5c01   add r4,1
004e 9800   ldc r0,0
0050 985a   ldc r0,90
0052 c0a0   bez r0,r5,r0
0054 9800   ldc r0,0
0056 9844   ldc r0,68
0058 d0a0   bnz r0,r5,r0
005a 5a01   add r2,1
005c 98f0   ldc r0,240
005e 9802   ldc r0,2
0060 5042   stw r0,r2,r0
0062 0003   ???
0064 5465   adc r4,r3,r1
0066 7374   ???
0068 6520   sub r5,r1,r0
006a 6465   sbc r4,r3,r1
006c 2034   xor r0,r1,r5
006e 2070   xor r0,r3,r4
0070 616c   sub r1,r3,r3
0072 6176   ???
0074 7261   ???
0076 7300   ???
0078 0a54   and r2,84
007a 6f74   sub r7,116
007c 616c   sub r1,r3,r3
007e 2064   xor r0,r3,r1
0080 6520   sub r5,r1,r0
0082 7061   ???
0084 6c61   sub r4,97
0086 7672   ???
0088 6173   ???
008a 3a09   slt r2,9
008c 0000   and r0,r0,r0
```

Dump da memória:

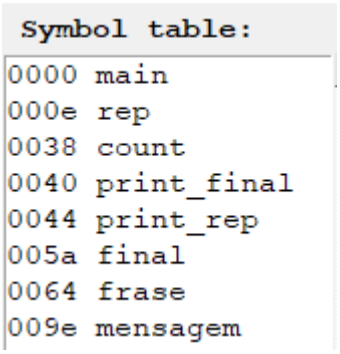
```
0000: 8a00 9d00 9d64 6f02 50be 9e00 9e38 451e |.....do.P....8E.|
0010: 5f02 0416 9800 9840 c080 98f0 9800 5082 |_......@......P.|
0020: 5d01 6f02 50be 8b21 338c d078 8b7e 338c |].o.P..!3..x.~3.|
0030: c078 9800 980e d0e0 5a01 9800 980e d0e0 |.x......Z.......|
0040: 9c00 9c78 0512 98f0 9800 50a2 5c01 9800 |...x......P.\...|
0050: 985a c0a0 9800 9844 d0a0 5a01 98f0 9802 |.Z.....D..Z.....|
0060: 5042 0003 5465 7374 6520 6465 2034 2070 |PB..Teste de 4 p|
0070: 616c 6176 7261 7300 0a54 6f74 616c 2064 |alavras..Total d|
0080: 6520 7061 6c61 7672 6173 3a09 0000 0000 |e palavras:.....|
0090: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
00a0: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
00b0: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
00c0: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
00d0: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
00e0: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
00f0: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
0100: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
0110: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
0120: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
0130: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
0140: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
0150: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
```

Saída no terminal:

```
Assembling... done. Program size: 142 bytes (code + data).
Teste de 4 palavras
Total de palavras: 4
Program halted at 0062.
```

1.4.3Teste 3: frase: “Frase de teste muito grande mesmo com mais de 10 palavras”

Tabela de símbolos:



Código de máquina:

```
Object code / disassembly:
0000 8a00    ldr r2,0
0002 9d00    ldc r5,0
0004 9d64    ldc r5,100
0006 6f02    sub r7,2
0008 50be    stw r0,r5,r7
000a 9e00    ldc r6,0
000c 9e38    ldc r6,56
000e 451e    ldw r5,r0,r7
0010 5f02    add r7,2
0012 0416    ldb r4,r0,r5
0014 9800    ldc r0,0
0016 9840    ldc r0,64
0018 c080    bez r0,r4,r0
001a 98f0    ldc r0,240
001c 9800    ldc r0,0
001e 5082    stw r0,r4,r0
0020 5d01    add r5,1
0022 6f02    sub r7,2
0024 50be    stw r0,r5,r7
0026 8b21    ldr r3,33
0028 338c    slt r3,r4,r3
002a d078    bnz r0,r3,r6
002c 8b7e    ldr r3,126
002e 338c    slt r3,r4,r3
0030 c078    bez r0,r3,r6
0032 9800    ldc r0,0
0034 980e    ldc r0,14
0036 d0e0    bnz r0,r7,r0
0038 5a01    add r2,1
003a 9800    ldc r0,0
003c 980e    ldc r0,14
003e d0e0    bnz r0,r7,r0
0040 9c00    ldc r4,0
0042 9c9e    ldc r4,158
0044 0512    ldb r5,r0,r4
0046 98f0    ldc r0,240
0048 9800    ldc r0,0
004a 50a2    stw r0,r5,r0
004c 5c01    add r4,1
004e 9800    ldc r0,0
0050 985a    ldc r0,90
0052 c0a0    bez r0,r5,r0
0054 9800    ldc r0,0
0056 9844    ldc r0,68
0058 d0a0    bnz r0,r5,r0
005a 5a01    add r2,1
005c 98f0    ldc r0,240
005e 9802    ldc r0,2
0060 5042    stw r0,r2,r0
0062 0003    ???
0064 4672    ldw r6,r3,r4
0066 6173    ???
0068 6520    sub r5,r1,r0
006a 6465    sbc r4,r3,r1
006c 2074    xor r0,r3,r5
006e 6573    ???
0070 7465    ???
0072 206d    ???
0074 7569    ???
0076 746f    ???
0078 2067    ???
007a 7261    ???
007c 6e64    sub r6,100
007e 6520    sub r5,r1,r0
0080 6d65    sub r5,101
0082 736d    ???
0084 6f20    sub r7,32
0086 636f    ???
0088 6d20    sub r5,32
008a 6d61    sub r5,97
008c 6973    sub r1,115
008e 2064    xor r0,r3,r1
0090 6520    sub r5,r1,r0
0092 3130    slt r1,r1,r4
0094 2070    xor r0,r3,r4
0096 616c    sub r1,r3,r3
0098 6176    ???
009a 7261    ???
009c 7300    ???
009e 0a54    and r2,84
00a0 6f74    sub r7,116
00a2 616c    sub r1,r3,r3
00a4 2064    xor r0,r3,r1
00a6 6520    sub r5,r1,r0
00a8 7061    ???
00aa 6c61    sub r4,97
00ac 7672    ???
00ae 6173    ???
00b0 3a09    slt r2,9
00b2 0000    and r0,r0,r0
```

Dump da memória:

```
0000: 8a00 9d00 9d64 6f02 50be 9e00 9e38 451e |.....do.P....8E.|
0010: 5f02 0416 9800 9840 c080 98f0 9800 5082 |_......@......P.|
0020: 5d01 6f02 50be 8b21 338c d078 8b7e 338c |].o.P..!3..x.~3.|
0030: c078 9800 980e d0e0 5a01 9800 980e d0e0 |.x......Z.......|
0040: 9c00 9c9e 0512 98f0 9800 50a2 5c01 9800 |..........P.\...|
0050: 985a c0a0 9800 9844 d0a0 5a01 98f0 9802 |.Z.....D..Z.....|
0060: 5042 0003 4672 6173 6520 6465 2074 6573 |PB..Frase de tes|
0070: 7465 206d 7569 746f 2067 7261 6e64 6520 |te muito grande |
0080: 6d65 736d 6f20 636f 6d20 6d61 6973 2064 |mesmo com mais d|
0090: 6520 3130 2070 616c 6176 7261 7300 0a54 |e 10 palavras..T|
00a0: 6f74 616c 2064 6520 7061 6c61 7672 6173 |otal de palavras|
00b0: 3a09 0000 0000 0000 0000 0000 0000 0000 |:...............|
00c0: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
00d0: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
00e0: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
00f0: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
0100: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
0110: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
0120: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
0130: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
0140: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
0150: 0000 0000 0000 0000 0000 0000 0000 0000 |................|
```

Saída no terminal:

```
Assembling... done. Program size: 180 bytes (code + data).
Frase de teste muito grande mesmo com mais de 10 palavras
Total de palavras: 11
Program halted at 0062.
```

2.Questão 2

2.1 Enunciado da Questão 2

Considere uma sequência de n números inteiros. Para esta sequência, determine um segmento de soma máxima e o valor dessa soma. Por exemplo, para a sequência de valores 5, 2, -2, -7, 3, 14, 10, -3, 9, -6, 4, 1, o segmento está entre o índices 4 e 8 e a soma dos valores desse segmento ´e 33. Utilize dois vetores de números na demonstração do funcionamento da sua solução.

2.2 Solução em Código Assembly Viking

main

```
; r1 = soma(começa em 0)


; r2 = sequencia
ldi r2,sequencia


; r3 → segmento
ldi r3,segmento


; r4 = segmento[0]
ldw r4,r3


; r5 = segmento[1]
add r3,2
ldw r5,r3


; salva r1,r2
sub sp,2
stw r1,sp
sub sp,2
stw r2,sp


; r1 = r2 = 0
ldi r1,0
ldi r2,0
```

```
        ; r1 = r5 - r4
        sub r1,r5,r4

        ; r6 = r1 (quantas vezes repito o laco)
        xor r6,r1,r2

        ; recupera r1,r2
        ldw r2,sp
        add sp,2
        ldw r1,sp
        add sp,2

        ; r2 -> sequence[i-1]
        sub r2,2

        ; r5 = -1
        ldi r5,-1

move_pointer
        ; r2 -> sequence[i+1]
        add r2,2

        ; r3 = 0
        ldi r3,0

        ; r4--;
        sub r4,1

        ; if (r4 >= 0) {
        slt r3,r5,r4

        ; repeat }
```

```
        bnz r3,move_pointer

        ; r2 -> sequence[i-1]
        sub r2,2

laco_soma
        ; r2 -> sequence[i+1]
        add r2,2

        ; r3 = 0
        ldi r3,0

        ; r3 = sequencia[i]
        ldw r3,r2

        ; r1 += r3
        add r1,r3,r1

        ; r6--
        sub r6,1

        ; if (r6 >= 0) {
        slt r4,r5,r6

        ; repeat }
        bnz r4,laco_soma

atr_msg
   ; r5 -> mensagem[0]
   ldi r5,mensagem

print_str
```

```
    ; r4 = mensagem[i]

    ldb r4,r5

    ; print(r4)
    stw r4,0xf000

    ; r5 -> mensagem[i+1] (goToNextChar)
    add r5,1

    ; repeat while there are words
    bnz r4,print_str

print_number
        stw r1,0xf002
        hcf

sequencia 5 2 -2 -7 3 14 10 -3 9 -6 4 1
segmento 4 8
mensagem "Soma: "
```

2.3 Explicação Passo a passo do Código Acima:

1) R1 recebe uma soma que começa em 0, enquanto r2 recebe uma sequência numérica, indicada na variável ao final do código.
2) Carrega-se a sequência para o registrador r2.
3) Carrega-se o segmento (entre os índices 4 e 8) no registrador r3.
4) O registrador r4 recebe o valor do segmento de índice 0 (de valor 4).
5) O registrador r3 passa a apontar para o próximo índice do segmento.
6) O registrador r5 recebe o valor do segmento de índice 1 (de valor 8).
7) A próxima posição livre na pilha é acessada.
8) Guarda-se o valor de r1 na posição livre do registrador de armazenamento (r7).
9) Acessa-se a próxima posição livre na pilha.
10) Guarda-se o valor do registrador r2 na posição livre acessada.
11) Carrega-se o valor zero no registrador r1.
12) Carrega-se o valor zero no registrador r2.
13) Realiza-se uma subtração dos valores armazenados em r5 e r4, armazenando o resultado no registrador r1 (8-4=4).
14) O valor de r1 é passado para r6, e esse valor indica a quantidade de vezes que o laço será realizado.
15) Carrega-se o valor da última posição do registrador de armazenamento (índice 1 = 8) no registrador r2.
16) Acessa-se a próxima posição do registrador de armazenamento.
17) Carrega-se o valor da posição acessada na última instrução no registrador r1.
18) Acessa-se o próximo valor livre do registrador r7. (Instruções realizadas para recuperar os registradores r1 e r2.)
19) O registrador r2 aponta para o índice i-1 da sequência.
20) Carrega-se o valor -1 no registrador r5.

Código do método de rótulo: move_pointer:

21) O registrador r2 aponta para o índice i+1 da sequência. (ou seja, volta para o 0).
22) Carrega-se o valor 0 no registrador r3.
23) O valor r4 irá sofrer a subtração: r4 = r4 – 1 (O valor da subtração será mantido em r4).
24) Se o valor do registrador r5 for menor que o valor do registrador r4, o registrador r3 recebe 1.
25) Se o registrador r3 não receber 0, ou seja, se r5 for realmente menor que r4, ocorre um jump para o método move_pointer, logo repetindo o laço.
26) O registrador r2 aponta para o índice i-1 da sequencia.

Código do método de rótulo: laço_soma:

27) O registrador r2 aponta para o índice i+1 da sequência.
28) Carrega-se o valor 0 no registrador r3.
29) R3 recebe o valor da posição de índice que r2 está apontando.
30) R1 recebe o valor de r3 + r1.

31) R6 irá sofrer uma subtração: r6 -1, e esse resultado será armazenado no registrador r6.
32) Se r5 for menor que r6, r4 recebe 1, caso contrário r4 recebe 0. (em outras palavras, se r6 for maior ou igual a 0).
33) Se r4 não for 0 ocorre um jump para o método laço_soma e repete-se o laço.

Código do método de rótulo: atr_msg:

34) R5 aponta para o endereço de mensagem (S).

Código do método de rótulo: print_str:

35) Carrega-se o valor do registrador r5 no registrador r4, em outras palavras r4 recebe o caractere.
36) Printa-se o valor armazenado em r4.
37) R5 aponta para o índice i+1 da variável mensagem (o).
38) Enquanto palavras existirem ocorre um jump para o método print_str, repetindo o laço, ou seja, imprime o próximo caractere.

Código do método de rótulo: print_number:

39) Printa-se o valor armazenado no registrador r1 (o resultado da soma).
40) Halt and catch fire.

2.4 Testes de Verificação

2.4.1Teste 1: segmento 0 – 2

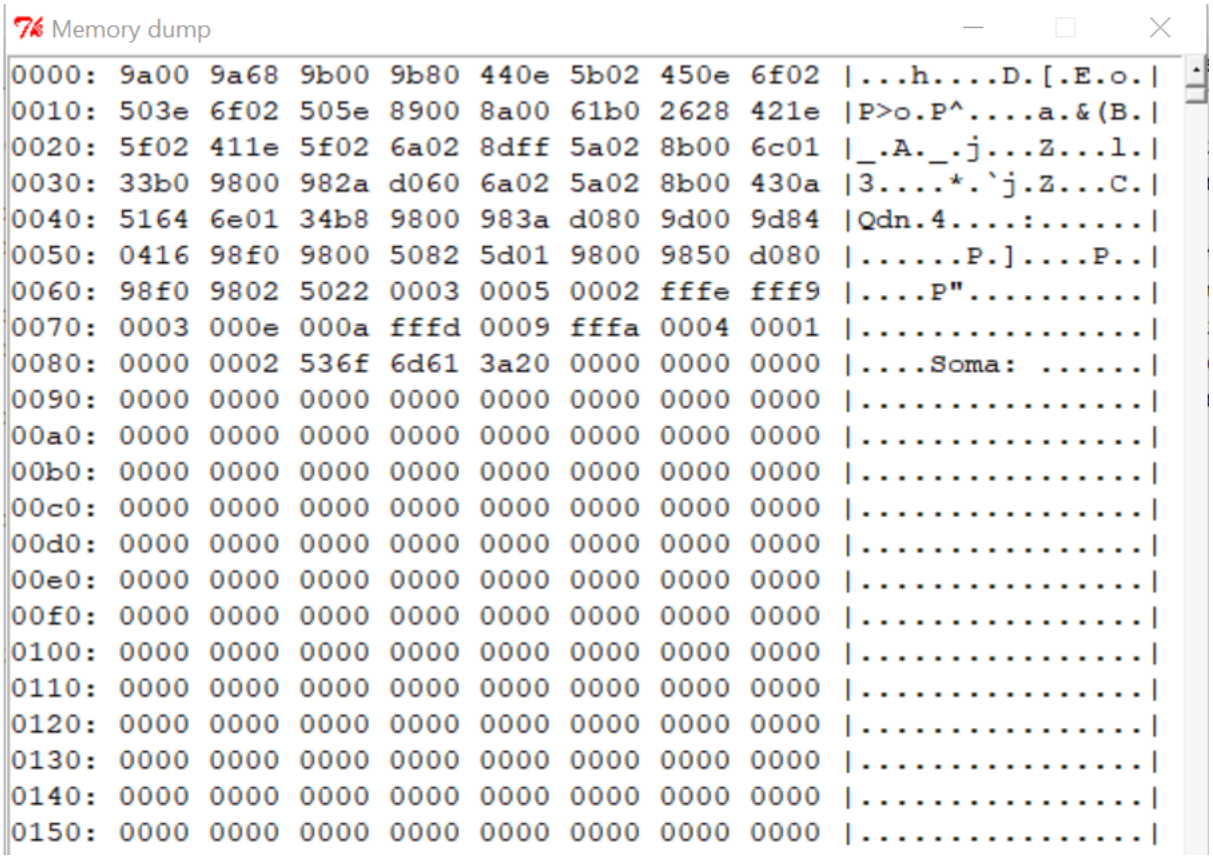
Tabela de Símbolos:

```
Symbol table:
0000 main
002a move_pointer
003a laco_soma
004c atr_msg
0050 print_str
0060 print_number
0068 sequencia
0080 segmento
0084 mensagem
```

Código de Máquina:

```
Object code / disassembly:
0000 9a00   ldc r2,0
0002 9a68   ldc r2,104
0004 9b00   ldc r3,0
0006 9b80   ldc r3,128
0008 440e   ldw r4,r0,r3
000a 5b02   add r3,2
000c 450e   ldw r5,r0,r3
000e 6f02   sub r7,2
0010 503e   stw r0,r1,r7
0012 6f02   sub r7,2
0014 505e   stw r0,r2,r7
0016 8900   ldr r1,0
0018 8a00   ldr r2,0
001a 61b0   sub r1,r5,r4
001c 2628   xor r6,r1,r2
001e 421e   ldw r2,r0,r7
0020 5f02   add r7,2
0022 411e   ldw r1,r0,r7
0024 5f02   add r7,2
0026 6a02   sub r2,2
0028 8dff   ldr r5,255
002a 5a02   add r2,2
002c 8b00   ldr r3,0
002e 6c01   sub r4,1
0030 33b0   slt r3,r5,r4
0032 9800   ldc r0,0
0034 982a   ldc r0,42
0036 d060   bnz r0,r3,r0
0038 6a02   sub r2,2
003a 5a02   add r2,2
003c 8b00   ldr r3,0
003e 430a   ldw r3,r0,r2
0040 5164   add r1,r3,r1
0042 6e01   sub r6,1
0044 34b8   slt r4,r5,r6
0046 9800   ldc r0,0
0048 983a   ldc r0,58
004a d080   bnz r0,r4,r0
004c 9d00   ldc r5,0
004e 9d84   ldc r5,132
0050 0416   ldb r4,r0,r5
0052 98f0   ldc r0,240
0054 9800   ldc r0,0
0056 5082   stw r0,r4,r0
0058 5d01   add r5,1
005a 9800   ldc r0,0
005c 9850   ldc r0,80
005e d080   bnz r0,r4,r0
0060 98f0   ldc r0,240
0062 9802   ldc r0,2
0064 5022   stw r0,r1,r0
0066 0003   ???
0068 0005   ???
006a 0002   ldb r0,r0,r0
006c fffe   ???
006e fff9   ???
0070 0003   ???
0072 000e   ldb r0,r0,r3
0074 000a   ldb r0,r0,r2
0076 fffd   ???
0078 0009   ???
007a fffa   ???
007c 0004   and r0,r0,r1
007e 0001   ???
0080 0000   and r0,r0,r0
0082 0002   ldb r0,r0,r0
0084 536f   ???
0086 6d61   sub r5,97
0088 3a20   slt r2,32
008a 0000   and r0,r0,r0
```

Dump da Memória:



Saída no Terminal:

```
Assembling... done. Program size: 140 bytes (code + data).
Soma: 5
Program halted at 0066.
```

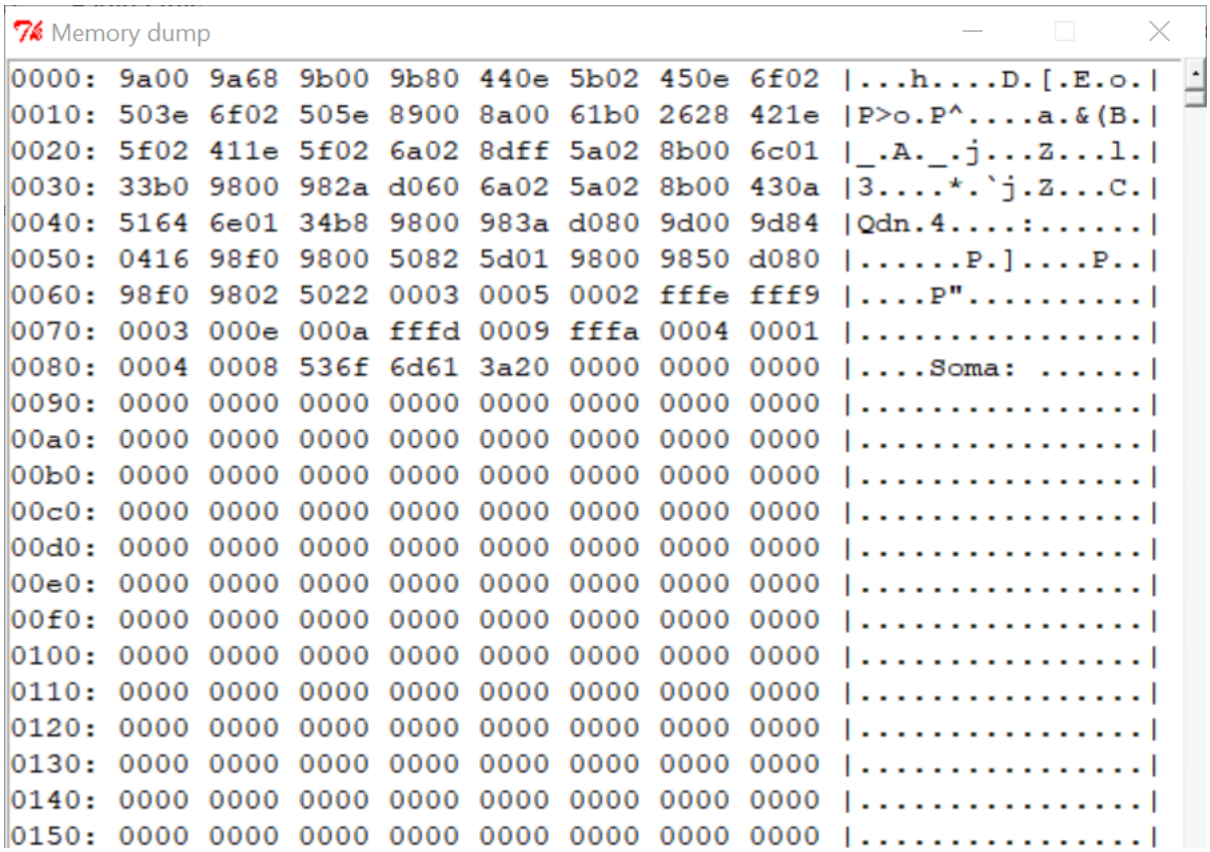
2.4.2Teste 2: 4 – 8
Tabela de Símbolos:

```
Symbol table:
0000 main
002a move_pointer
003a laco_soma
004c atr_msg
0050 print_str
0060 print_number
0068 sequencia
0080 segmento
0084 mensagem
```

Código de Máquina:

```
Object code / disassembly:
0000 9a00    ldc r2,0
0002 9a68    ldc r2,104
0004 9b00    ldc r3,0
0006 9b80    ldc r3,128
0008 440e    ldw r4,r0,r3
000a 5b02    add r3,2
000c 450e    ldw r5,r0,r3
000e 6f02    sub r7,2
0010 503e    stw r0,r1,r7
0012 6f02    sub r7,2
0014 505e    stw r0,r2,r7
0016 8900    ldr r1,0
0018 8a00    ldr r2,0
001a 61b0    sub r1,r5,r4
001c 2628    xor r6,r1,r2
001e 421e    ldw r2,r0,r7
0020 5f02    add r7,2
0022 411e    ldw r1,r0,r7
0024 5f02    add r7,2
0026 6a02    sub r2,2
0028 8dff    ldr r5,255
002a 5a02    add r2,2
002c 8b00    ldr r3,0
002e 6c01    sub r4,1
0030 33b0    slt r3,r5,r4
0032 9800    ldc r0,0
0034 982a    ldc r0,42
0036 d060    bnz r0,r3,r0
0038 6a02    sub r2,2
003a 5a02    add r2,2
003c 8b00    ldr r3,0
003e 430a    ldw r3,r0,r2
0040 5164    add r1,r3,r1
0042 6e01    sub r6,1
0044 34b8    slt r4,r5,r6
0046 9800    ldc r0,0
0048 983a    ldc r0,58
004a d080    bnz r0,r4,r0
004c 9d00    ldc r5,0
004e 9d84    ldc r5,132
0050 0416    ldb r4,r0,r5
0052 98f0    ldc r0,240
0054 9800    ldc r0,0
0056 5082    stw r0,r4,r0
0058 5d01    add r5,1
005a 9800    ldc r0,0
005c 9850    ldc r0,80
005e d080    bnz r0,r4,r0
0060 98f0    ldc r0,240
0062 9802    ldc r0,2
0064 5022    stw r0,r1,r0
0066 0003    ???
0068 0005    ???
006a 0002    ldb r0,r0,r0
006c fffe    ???
006e fff9    ???
0070 0003    ???
0072 000e    ldb r0,r0,r3
0074 000a    ldb r0,r0,r2
0076 fffd    ???
0078 0009    ???
007a fffa    ???
007c 0004    and r0,r0,r1
007e 0001    ???
0080 0004    and r0,r0,r1
0082 0008    and r0,r0,r2
0084 536f    ???
0086 6d61    sub r5,97
0088 3a20    slt r2,32
008a 0000    and r0,r0,r0
```

Dump da Memória:



Saída no Terminal:

```
Assembling... done. Program size: 140 bytes (code + data).
Soma: 33
Program halted at 0066.
```

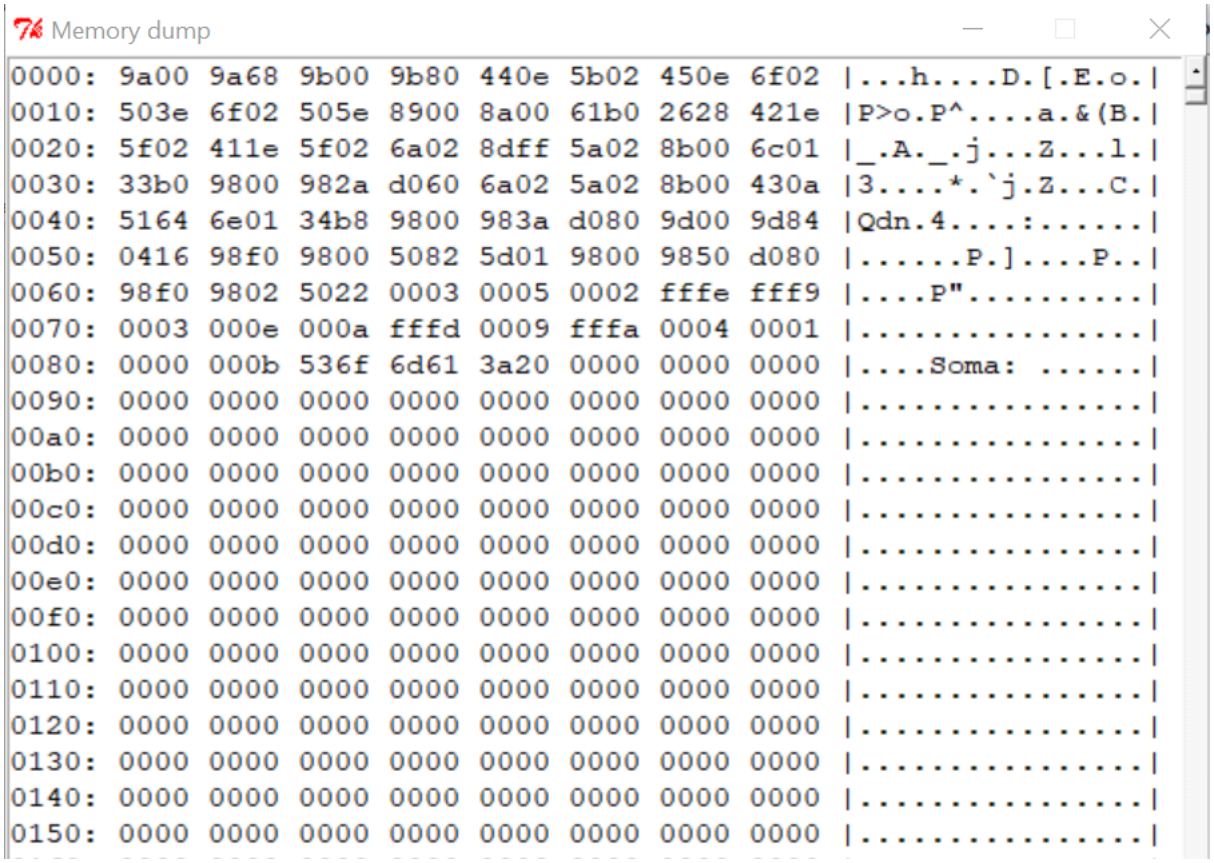
2.4.3Teste 3: 0 – 11
Tabela de Símbolos:

```
Symbol table:
0000 main
002a move_pointer
003a laco_soma
004c atr_msg
0050 print_str
0060 print_number
0068 sequencia
0080 segmento
0084 mensagem
```

Código de Máquina:

```
Object code / disassembly:
0000 9a00   ldc r2,0
0002 9a68   ldc r2,104
0004 9b00   ldc r3,0
0006 9b80   ldc r3,128
0008 440e   ldw r4,r0,r3
000a 5b02   add r3,2
000c 450e   ldw r5,r0,r3
000e 6f02   sub r7,2
0010 503e   stw r0,r1,r7
0012 6f02   sub r7,2
0014 505e   stw r0,r2,r7
0016 8900   ldr r1,0
0018 8a00   ldr r2,0
001a 61b0   sub r1,r5,r4
001c 2628   xor r6,r1,r2
001e 421e   ldw r2,r0,r7
0020 5f02   add r7,2
0022 411e   ldw r1,r0,r7
0024 5f02   add r7,2
0026 6a02   sub r2,2
0028 8dff   ldr r5,255
002a 5a02   add r2,2
002c 8b00   ldr r3,0
002e 6c01   sub r4,1
0030 33b0   slt r3,r5,r4
0032 9800   ldc r0,0
0034 982a   ldc r0,42
0036 d060   bnz r0,r3,r0
0038 6a02   sub r2,2
003a 5a02   add r2,2
003c 8b00   ldr r3,0
003e 430a   ldw r3,r0,r2
0040 5164   add r1,r3,r1
0042 6e01   sub r6,1
0044 34b8   slt r4,r5,r6
0046 9800   ldc r0,0
0048 983a   ldc r0,58
004a d080   bnz r0,r4,r0
004c 9d00   ldc r5,0
004e 9d84   ldc r5,132
0050 0416   ldb r4,r0,r5
0052 98f0   ldc r0,240
0054 9800   ldc r0,0
0056 5082   stw r0,r4,r0
0058 5d01   add r5,1
005a 9800   ldc r0,0
005c 9850   ldc r0,80
005e d080   bnz r0,r4,r0
0060 98f0   ldc r0,240
0062 9802   ldc r0,2
0064 5022   stw r0,r1,r0
0066 0003   ???
0068 0005   ???
006a 0002   ldb r0,r0,r0
006c fffe   ???
006e fff9   ???
0070 0003   ???
0072 000e   ldb r0,r0,r3
0074 000a   ldb r0,r0,r2
0076 fffd   ???
0078 0009   ???
007a fffa   ???
007c 0004   and r0,r0,r1
007e 0001   ???
0080 0000   and r0,r0,r0
0082 000b   ???
0084 536f   ???
0086 6d61   sub r5,97
0088 3a20   slt r2,32
008a 0000   and r0,r0,r0
```

Dump da Memória:



Saída no Terminal:

```
Assembling... done. Program size: 140 bytes (code + data).
Soma: 30
Program halted at 0066.
```

REFERÊNCIAS

FILHO, Sérgio Johann. Viking CPU - **Manual de referência v0.5**, dez. 2018.