

Pontifícia Universidade Católica do Rio Grande do Sul
Fundamentos de Sistemas Computacionais
Engenharia de Software

Mateus Caçabuena, Carolina Ferreira e Lucca Paradedda

@mateus.cacabuena@edu.pucrs.br

@carolina.michel@edu.pucrs.br

@lucca.paradedda@edu.pucrs.br

Relatório do Trabalho 2

Porto Alegre

2022

Introdução

O presente relatório tem como objetivo descrever cada um dos 2 problemas apresentados, explicando as soluções desenvolvidas. Nesta explicação, será retratado a análise do programa, além de uma breve descrição do algoritmo e a verificação da funcionalidade dele.

Este trabalho foi feito utilizando a linguagem de programação denominada Assembly, para a arquitetura viking, pois utilizando linguagem de montagem a sua sintaxe dependendo para qual arquitetura você está escrevendo.

Ao final desse trabalho você irá saber como implementar um código em linguagem de montagem do computador Viking utilizando programação modular, a fim de resolver os seguintes problemas:

- **Problema 1:** Escreva um programa que conta o número de palavras armazenadas em uma *string* e apresenta o total no terminal.
- **Problema 2:** Considere uma sequência de n números inteiros. Para esta sequência, determine um segmento de soma máxima e o valor dessa soma.

A partir desse trabalho é possível concluir pelo menos uma forma de solucionar cada problema apresentado utilizando programação modular, para o desenvolvimento desse trabalho foi feito um estudo em cima do arquivo pdf denominado **viking_manual_pt.pdf** desenvolvido pelo professor Sérgio Johann Filho, da Universidade Pontifícia Católica do Rio grande do Sul (PUCRS). Este material foi desenvolvido no dia 31 de dezembro de 2018 e é o principal material de estudo sobre linguagem de montagem da disciplina fundamentos de sistemas computacionais, ele apresenta toda a base de estudo necessária para a conclusão desse trabalho.

Durante o desenvolvimento dessa atividade foram encontrados alguns desafios tais como, raciocínio lógico, compreensão da linguagem viking e execução com êxito dos problemas solicitados, que foram resolvidos através de leituras do manual disponibilizado e testes realizados diversas vezes para checagem do funcionamento correto do programa. No desenvolvimento dos problemas os três integrantes do grupo tiveram que exercer a sua capacidade intelectual para a realização completa e satisfeita de suas atividades.

Problema 1

“Escreva um programa que conta o número de palavras armazenadas em uma *string* e apresenta o total no terminal.”

As duas *strings* que foram usadas como exemplo são as seguintes frases:

- “Macacos me mordam”
- “O rato roeu a roupa do rei de Roma”

O programa implementado foi baseado no código dado como sugestão que, convertido no *assembly*, entregou o resultado esperado nas duas *strings* que foram usadas como exemplo.

Instruções:

- r1 = i
- r2 = words quantidade
- r3 = vetor de caracteres ou *string*
- r4 = vetsz
- r5 = armazenador do caractere localizado em r3
- r6 = variável multifuncional

Descrição:

Primeiramente, foram declarados 4 registradores que seriam usados em funções fixas durante todo o funcionamento do programa:

- ❖ **r1 = i:** Responsável por contar quantos caracteres foram lidos, ao chegar no mesmo número de caracteres do ‘vetsz’, o programa é encerrado.
- ❖ **r2 = words:** É o registrador que obtém o número de palavras que será retornado no terminal do simulador.
- ❖ **r3 = vetor:** Possui a função de armazenar o vetor de *string* que possui a mensagem contabilizada.
- ❖ **r4 = vetsz:** Neste registrador, é armazenado a quantia de slots de caracteres que possui no r3. Lembrando que, se é uma *string* de 20 caracteres, por exemplo, possui 19 slots pois é contado a partir do 0. Juntamente com o r1, são os responsáveis pela quantia de ciclos que possui no simulador.

Já dentro dos *whiles*, foram implementados outros 2 registradores que foram responsáveis pelo funcionamento principal do programa, além do r7, que se responsabilizou pelos loops que um *while* precisa realizar:

- ❖ **r5 = caractere:** Registrador encarregado pelo acesso ao vetor r3. Nele, há o valor que endereça o caractere digitado no slot numerado pelo r1.
- ❖ **r6 = variável “Coringa”:** Este registrador foi o mais utilizado do programa. Foi registrado diversos valores nesta variável, majoritariamente usado para registrar os valores necessários para maior que e menor que, além de *if's* necessários para o funcionamento do *while*.

A seguir, o código de funcionamento do programa explicado detalhadamente:


```

1      main
2          ldi r1,0          ; regista r1 no sistema
3          ldi r2,0          ; regista r2 no sistema
4          ldi r3,vet1       ; endereça vet1 na variável r3
5          ldw r4,vetsz1     ; regista o valor de vetsz1 em r4
6
7      while
8          ldb r5,r3         ; regista o valor da string endereçada em r3 na variável r5
9
10         ldi r6,33         ; regista o valor de 33 em r6
11         slt r6,r5,r6      ; se r5 é menor que 33, r6 =1, caso contrário, r6=0
12         bnz r6,if        ; se r6 != 0, pula para o if
13                             ; se r6 = 0, segue o código normalmente
14         ldi r6,126        ; regista o valor de 126 em r6
15         slt r6,r6,r5      ; se 126 é menor ou igual a r5, r6=1, caso contrário, r6=0
16         bnz r6,if        ; se r6!=0, pula para o if
17
18     else
19         bnz r7,while2     ; caso não pule para o if nos dois casos acima, pula para o while2
20     if
21         slt r6,r4,r1      ; se o tamanho do vetor(r4) é menor ou igual ao i(r1), r6=1
22         bnz r6,end        ; se r6=1, pula para o fim do programa
23         add r3,1          ; endereça o próximo possível caractere vetor
24         add r1,1          ; adiciona +1 ao i, indicando a mudança de slot do vetor
25         bnz r7,while     ; reinicia o while
26
27     while2
28         ldb r5,r3         ; regista o valor da string endereçada em r3 na variável r5
29
30         ldi r6,32         ; regista o valor de 32 em r6
31         slt r6,r6,r5      ; se 32 é menor que r5, r6 =1, caso contrário, r6=0
32         bez r6,else2      ; se r6=0, pula para o else2
33
34         ldi r6,127        ; regista o valor de 127 em r6
35         slt r6,r5,r6      ; se r5 é menor que 127, r6=1, caso contrário, r6=0
36         bez r6,else2      ; se r6=0, pula para o else2
37
38         bnz r6,if2        ; se r6=1, pula para o if2
39
40     else2
41         add r2,1          ; adiciona 1 ao r2
42         bnz r7,while     ; reinicia o loop, para o primeiro while
43     if2
44         slt r6,r4,r1      ; se o tamanho do vetor(r4) é menor ou igual ao i(r1), r6=1
45         add r2,r2,r6      ; adiciona o valor de r6 para r2, podendo ser 1 ou 0
46         bnz r6,end        ; se r6=1, pula para o fim do programa
47
48         add r1,1          ; adiciona 1 para r1
49         add r3,1          ; adiciona 1 para r3
50         bnz r7,while2     ; reinicia o while2
51
52
53     end
54         stw r2,0xf002     ; retrata no terminal o valor de r2
55         hcf              ; encerra o programa
56
57     vet1    "Macacos me mordam"
58     vet2    "O rato roeu a roupa do rei de Roma"
59     vetsz1  16
60     vetsz2  33


```

Análise do Sistema:


Após implementar os 4 registradores iniciais, foi iniciado os ciclos dos 2 *whiles* que contabilizam as palavras:

 **While:** No primeiro loop *while* (linha 7 até 25), o programa é direcionado a descobrir se a numeração hexadecimal da letra endereçada no registrador é menor que 33 ou maior que 126, pois são números que não correspondem caracteres.

- Assim, a linha 10 até 16 é utilizada para esse descobrimento, caso seja, é direcionado para o primeiro *if*, que utiliza o *r1* e *r4* para descobrir se já está ao final da frase ou não, além de endereçar o próximo número do *r3* juntamente a soma de 1 no *r1*.
- Caso não seja o caso de entrar no *if*, então assume-se que é um caractere e o *else* direciona este caractere para o *while2*

 **While2:** Neste segundo loop (linha 27 até 50), o programa é um pouco mais complexo. É preciso garantir de que o caractere está entre os números hexadecimais 32 e 127, para isso, foi implementado um *AND* feito com *SLT* e *BEZ* (linhas 31-32 e 35-36).

- Tendo garantido que é um caractere, o programa entra no segundo *if*, fazendo as mesmas execuções que o primeiro, com exceção da adição de +1 no registrador *r2* caso tenha chegado ao último loop, indicando que é uma palavra.

 **End:** Com o *r1* sendo maior ou igual a *r4*, o programa entende que a palavra já está completa, então pode encerrá-lo. Com o *STW*, é retratado no terminal quantas palavras há por meio do *r2* e com o *HCF* o programa é encerrado.

Resumindo o código de um modo simplificado, o programa recebe o valor endereçado e assim ele passa pelos *whiles*, caso não pertença a um deles, pertencerá ao outro. Assim permanece o ciclo destes 2 *whiles* até que o *r1* ultrapasse ou atinja o mesmo tamanho de *r4*.

A seguir, será mostrado a execução do programa com os dois exemplos requeridos no enunciado do problema.

Teste realizado com o *vet1*, resultado previsto de 3 palavras: "Macacos me mordam".

Program	Machine	Object code / disassembly:	Symbol table:	Registers:
main		0000 8900 ldr r1,0	0000 main	r0 (at) : f002
		0002 8a00 ldr r2,0	000e while2	r1 : 0011
		0004 9b00 ldc r3,0	0024 else	r2 : 0003
		0006 9b7c ldc r3,124	002a if	r3 : 008d
		0008 9800 ldc r0,0	003c while3	r4 : 0010
		000a 98b2 ldc r0,178	0058 else2	r5 (sr) : 0000
while2		000c 4402 ldw r4,r0,r0	0060 if2	r6 (lr) : 0001
		000e 050e ldb r5,r0,r3	0074 end	r7 (sp) : dffe
		0010 8e21 ldr r6,33	007c vet1	PC : 007a
		0012 36b8 slt r6,r5,r6	008e vet2	
		0014 9800 ldc r0,0	00b2 vetsz1	
		0016 982a ldc r0,42	00b4 vetsz2	
		0018 d0c0 bnz r0,r6,r0		
		001a 8e7e ldr r6,126		
		001c 36d4 slt r6,r6,r5		
		001e 9800 ldc r0,0		
else		0020 982a ldc r0,42		
		0022 d0c0 bnz r0,r6,r0		
if		0024 9800 ldc r0,0		
		0026 983c ldc r0,60		
		0028 d0e0 bnz r0,r7,r0		
		002a 3684 slt r6,r4,r1		
		002c 9800 ldc r0,0		
		002e 9874 ldc r0,116		

Control:

Cycle: 482

Reset

Stop

Run

Step

Assembling... done. Program size: 182 bytes (code + data).
 3
 Program halted at 007a.

Como foi retratado, o terminal retornou o número 3, que é a quantia de palavras contidas na primeira *string*. Agora, será testada uma frase mais extensa, com 9 palavras. Para isso, é necessário mudar a linha 4 e 5 do código: endereçar *vet2* no *r3* e *vetsz2* no *r4*, para o simulador entender que o teste agora é com outra *string*, de tamanho diferente.

Teste realizado com o *vet2*, resultado previsto de 9 palavras: “O rato roeu a roupa do rei de Roma”

Program Machine

Program:

main

ldi r1,0
ldi r2,0
ldi r3,vet2
ldw r4,vetsz2
while2
ldb r5,r3
ldi r6,33
slt r6,r5,r6
bnz r6,if
ldi r6,126
slt r6,r6,r5
bnz r6,if
else
bnz r7,while3
if
slt r6,r4,r1
bnz r6,end
add r3,1
add r1,1
bnz r7,while2

Object code / disassembly:

005e d0e0 bnz r0,r7,r0
0060 3684 slt r6,r4,r1
0062 5258 add r2,r2,r6
0064 9800 ldc r0,0
0066 9874 ldc r0,116
0068 d0c0 bnz r0,r6,r0
006a 5901 add r1,1
006c 5b01 add r3,1
006e 9800 ldc r0,0
0070 983c ldc r0,60
0072 d0e0 bnz r0,r7,r0
0074 98f0 ldc r0,240
0076 9802 ldc r0,2
0078 5042 stw r0,r2,r0
007a 0003 ???
007c 4d61 sltu r5,97
007e 6361 sbc r3,r3,r0
0080 636f ???
0082 7320 ???
0084 6d65 sub r5,101
0086 206d ???
0088 6f72 sub r7,114
008a 6461 sbc r4,r3,r0
008c 6d00 sub r5,0

Symbol table:

0000 main
000e while2
0024 else
002a if
003c while3
0058 else2
0060 if2
0074 end
007c vet1
008e vet2
00b2 vetsz1
00b4 vetsz2

Registers:

r0 (at) : f002
r1 : 0022
r2 : 0009
r3 : 00b0
r4 : 0021
r5 (sr) : 0000
r6 (lr) : 0001
r7 (sp) : dffe
PC : 007a

Control:

Cycle: 980

Reset

Stop

Run

Step

Assembling... done. Program size: 182 bytes (code + data).

9

Program halted at 007a.

Nota-se, com os resultados corretos tanto no primeiro teste, quanto no segundo, que o programa funciona com êxito no simulador.

Problema 2

“Considere uma sequência de n números inteiros. Para esta sequência, determine um segmento de soma máxima e o valor dessa soma.”

Os dois vetores que foram usados como exemplo possuem a seguinte numeração:

- 5 2 -2 -7 3 14 10 -3 9 -6 4 1
- 2 34 12 -17 22 38 -15 7 10 41

O programa implementado foi feito sem baseamento em qualquer outro código, foi digitalizado com o raciocínio lógico e determinado conhecimento do simulador adquirido com a leitura do manual disponibilizado.

Instruções:

- $r1$ = índice inicial
- $r2$ = índice final
- $r3$ = soma dos segmentos
- $r4$ = vetor de números
- $r5$ = variável multifuncional
- $r6$ = armazenador do número localizado em $r4$

Descrição:

Primeiramente, foram declarados 4 registradores que seriam usados em funções fixas durante todo o funcionamento do programa:

- ❖ **$r1$ = índice inicial:** É o registrador que possui o slot que o vetor começará a contar para ser adicionado na soma de segmentos.
- ❖ **$r2$ = índice final:** É o registrador que possui o último slot que o vetor terminará a contar para ser adicionado na soma de segmentos.
- ❖ **$r3$ = soma:** Este é o registrador que será impresso no terminal, pois é o responsável por somar o valor de cada segmento.
- ❖ **$r4$ = vetor:** É o registrador que endereça o vetor, a partir dele será obtido os valores para serem somados.

Já dentro do *rep* e do *if* que há nele, foram implementados outros 2 registradores que foram responsáveis pelo funcionamento principal do programa, além do $r7$, que se responsabilizou pela repetição destas linhas.

- ❖ **$r5$ = variável “Coringa”:** Este registrador obteve diversos valores nele, serviu para executarmos o maior que e menor que, que limitava o acesso para o *if*.
- ❖ **$r6$ = vetor[i]:** Registrador encarregado por carregar o número que era endereçado pelo vetor. É com ele que soma no $r3$ para imprimir no terminal.

A seguir, o código de funcionamento do programa explicado detalhadamente:



```


1      main
2          ldi r1,4          ; regista o valor 4 no r1
3          ldi r2,8          ; regista o valor 8 no r2
4          ldi r3,0          ; regista o valor r3 como 0
5          ldi r4,vet1       ; regista o endereçamento do vetor no r4
6
7          add r4,r4,r1      ; método de ponteiro, adiciona valor de r1 ao r4
8          add r4,r4,r1      ; método de ponteiro, adiciona valor de r1 ao r4
9
10         rep
11             slt r5,r1,r2    ; se r1 for menor que r2, r5=1, caso contrário, r6=0
12             bnz r5,if      ; se r5!=0 pula para o if
13
14             sub r5,r1,r2    ; r5 = r1 - r2
15             bez r5,if      ; se r5 = 0, pula para o if
16             bnz r5,end     ; se r5 != 0, pula para o end
17
18         if
19             ldw r6,r4       ; regista o valor endereçado de r4 em r6
20
21             add r4,1        ; método de ponteiro, adiciona +1 ao r4
22             add r4,1        ; método de ponteiro, adiciona +1 ao r4
23
24             add r3,r3,r6    ; índice de soma, soma r6 ao r3
25
26             add r1,1        ; soma +1 ao r1
27             bnz r7,rep     ; reinicia o rep
28
29         end
30             stw r3,0xf002   ; imprime no terminal o valor numérico de r3
31             hcf            ; encerra o programa
32
33
34         vet1    5 2 -2 -7 3 14 10 -3 9 -6 4 1
35         vet2    2 34 12 -17 22 38 -15 7 10 41

```

Análise do Sistema:

Após implementar os 4 registradores iniciais, foi iniciado os ciclos do *rep* que somam os números necessários:

 **rep:** A primeira coisa verificada é se o segmento inicial é menor que o segmento final, caso não seja, é também checado se possuem o mesmo tamanho. O programa acaba caso nenhum destes 2 casos seja verdadeiro, retratando o valor atual de *r3*. Ademais, caso algum destes casos seja verdadeiro, o código pula para o *if*, onde acontece as principais ações do programa.

 **If:** Ao entrar na operação *if*, é compreendido que o número que está endereçado será somado no *r3*, portanto, é armazenado este valor no registrador *r6*. Consequentemente, o endereço do vetor já vai para o próximo número, pois não há mais nada a fazer com este endereço, já que o valor já foi armazenado em *r6*. Finalmente, o código soma o valor de *r6* no *r3* e depois de adicionar +1 ao segmento inicial, demonstrando que o ciclo do segmento foi completo, reinicia-se o *rep*.

Em resumo, o programa recebe o valor endereçado do vetor e soma ele a um registrador até que o segmento inicial seja maior que o segmento final.

A seguir, será mostrado a execução do programa com os dois exemplos requeridos no enunciado do problema.

Teste realizado com o *vet1*, com os índices de segmento iguais ao do exemplo dado no enunciado: 4 a 8
 Números deste segmento: 3 14 10 -3 9
 Resultado previsto: 33

Program	Machine	Object code / disassembly:	Symbol table:	Registers:
main		0000 8904 ldr r1,4	0000 main	r0 (at) : f002
ldi r1,4		0002 8a08 ldr r2,8	000e rep	r1 : 0009
ldi r2,8		0004 8b00 ldr r3,0	0024 if	r2 : 0008
ldi r3,0		0006 9c00 ldc r4,0	0034 end	r3 : 0021
ldi r4,vet1		0008 9c3c ldc r4,60	003c vet1	r4 : 004e
add r4,r4,r1		000a 5484 add r4,r4,r1	0054 vet2	r5 (sr) : 0001
add r4,r4,r1		000c 5484 add r4,r4,r1		r6 (lr) : 0009
rep		000e 3528 slt r5,r1,r2		r7 (sp) : dffe
slt r5,r1,r2		0010 9800 ldc r0,0		PC : 003a
bnz r5,if		0012 9824 ldc r0,36		Control:
sub r5,r1,r2		0014 d0a0 bnz r0,r5,r0		Cycle: 85
bez r5,if		0016 6528 sub r5,r1,r2		Reset
bnz r5,end		0018 9800 ldc r0,0		Stop
if		001a 9824 ldc r0,36		Run
ldw r6,r4		001c c0a0 bez r0,r5,r0		Step
add r4,1		001e 9800 ldc r0,0		
add r4,1		0020 9834 ldc r0,52		
add r3,r3,r6		0022 d0a0 bnz r0,r5,r0		
add r1,1		0024 4612 ldw r6,r0,r4		
bnz r7,rep		0026 5c01 add r4,1		
end		0028 5c01 add r4,1		
stw r3,0xf002		002a 5378 add r3,r3,r6		
		002c 5901 add r1,1		
		002e 9800 ldc r0,0		

Assembling... done. Program size: 104 bytes (code + data).
 33
 Program halted at 003a.

Assim, como no primeiro código, para realizar o segundo teste é necessário mudar a linha 5 do código, colocando *vet2* invés de *vet1*. Este método de implementação permite o programa a realizar o que é pedido com qualquer tipo de vetor, desde que seja registrado, sendo necessário apenas a alteração do nome no registrador.

Teste realizado com o vet2, com os índices de segmento: 2 a 7
 Números deste segmento: 12 -17 22 38 -15 7
 Resultado previsto: 47

Program	Machine	Object code / disassembly:	Symbol table:	Registers:
Program:				
nain		000c 5484 add r4,r4,r1	0000 main	r0 (at) : f002
	ldi r1,2	000e 3528 slt r5,r1,r2	000e rep	r1 : 0008
	ldi r2,7	0010 9800 ldc r0,0	0024 if	r2 : 0007
	ldi r3,0	0012 9824 ldc r0,36	0034 end	r3 : 002f
	ldi r4,vet2	0014 d0a0 bnz r0,r5,r0	003c vet1	r4 : 0064
	add r4,r4,r1	0016 6528 sub r5,r1,r2	0054 vet2	r5 (sr) : 0001
	add r4,r4,r1	0018 9800 ldc r0,0		r6 (lr) : 0007
rep		001a 9824 ldc r0,36		r7 (sp) : dffe
	slt r5,r1,r2	001c c0a0 bez r0,r5,r0		PC : 003a
	bnz r5,if	001e 9800 ldc r0,0		
	sub r5,r1,r2	0020 9834 ldc r0,52		
	bez r5,if	0022 d0a0 bnz r0,r5,r0		
	bnz r5,end	0024 4612 ldw r6,r0,r4		
if		0026 5c01 add r4,1		
	ldw r6,r4	0028 5c01 add r4,1		
	add r4,1	002a 5378 add r3,r3,r6		
	add r4,1	002c 5901 add r1,1		
		002e 9800 ldc r0,0		
	add r3,r3,r6	0030 980e ldc r0,14		
		0032 d0e0 bnz r0,r7,r0		
	add r1,1	0034 98f0 ldc r0,240		
	bnz r7,rep	0036 9802 ldc r0,2		
end		0038 5062 stw r0,r3,r0		
	stw r3,0xf002	003a 0003 ???		

Control:

Cycle: 97

Reset

Stop

Run

Step

Assembling... done. Program size: 104 bytes (code + data).

47

Program halted at 003a.

Nota-se, com os resultados corretos tanto no primeiro teste, quanto no segundo, que o programa funciona com êxito no simulador.

Conclusão

Após a realização deste trabalho avaliativo, conclui-se que o raciocínio feito para realizar ambos os problemas contribuíram muito para o aprendizado do simulador e do assembly. Como dito anteriormente, foi estudado intensamente o manual viking e a partir dele foram compreendidos como fazer um *while*, *if* e outras operações que são muito fáceis de realizar no java, mas que no assembly é necessário certo raciocínio lógico.

Outro ponto necessário a se destacar, foi o uso limitado de variáveis. Tendo em vista que são apenas 6 registradores possíveis de modificar valores (abstendo-se do *r7*), foi compreendido que não é necessário criar uma grande variedade de novas variáveis para realizar um programa usual.

Para finalizar, é imprescindível destacar que no assembly foi descoberto diversos detalhes com a realização desta avaliação, como o endereçamento de um vetor, método ponteiro, armazenar um valor endereçado do vetor, entre outros tópicos que, por mais simples que pareça o código, é necessário um bom tempo de raciocínio para implementá-lo corretamente.

Bibliografia

Filho, Sergio. Viking CPU - Manual de referência v0.5, 2018