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

Mateus Caçabuena, Carolina Ferreira e Lucca Paradeda
@mateus.cacabuena@edu.pucrs.br
@carolina.michel@edu.pucrs.br
@lucca.paradeda@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 Pontifica 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 tiverem 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 *r*7, que se responsabilizou pelos loops que um *while* precisa realizar:

- **❖ r5 = caractere:** Registrador encarregado pelo acesso ao vetor *r*3. Nele, há o valor que endereça o caractere digitado no slot numerado pelo *r*1.
- ❖ 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
                                       ; registra r1 no sistema
                                       ; registra r2 no sistema
3
                   ldi r2.0
4
                   ldi r3,vet1
                                       ; endereça vet1 na variável r3
                   ldw r4,vetsz1
5
                                       ; registra o valor de vetsz1 em r4
6
7
         while
8
                   ldb r5,r3
                                       ; registra o valor da string endereçada em r3 na variável r5
9
                   ldi r6,33
10
                                       ; registra o valor de 33 em r6
                   slt r6,r5,r6
                                       ; se r5 é menor que 33, r6 =1, caso contrário, r6=0
11
                   bnz r6,if
12
                                       ; se r6 != 0, pula para o if
13
                                       ; se r6 = 0, segue o código normalmente
14
                   ldi r6,126
                                       ; registra 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
                   bnz r6,if
16
                                       ; 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
         if
20
21
                   slt r6,r4,r1
                                       ; se o tamanho do vetor(r4) é menor ou igual ao i(r1), r6=1
                   bnz r6,end
22
                                       ; se r6=1, pula para o fim do programa
                   add r3,1
23
                                       ; endereça o próximo possível caractere vetor
                   add r1,1
                                       ; adiciona +1 ao i, indicando a mudança de slot do vetor
24
25
                   bnz r7, while
                                       ; reinicia o while
26
27
         while2
28
                   ldb r5,r3
                                       ; registra o valor da string endereçada em r3 na variável r5
29
30
                   ldi r6,32
                                       ; registra 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
                   ldi r6,127
                                       ; registra o valor de 127 em r6
34
                                       ; se r5 é menor que 127, r6=1, caso contrário, r6=0
35
                   slt r6,r5,r6
                   bez r6,else2
                                       ; se r6=0, pula para o else2
36
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
                                       ; adiciona o valor de r6 para r2, podendo ser 1 ou 0
45
                   add r2,r2,r6
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
                    "Macacos me mordam"
57
         vet1
58
         vet2
                   "O rato roeu a roupa do rei de Roma"
59
         vetsz1
                   16
60
         vetsz2
                   33
```

Análise do Sistema:

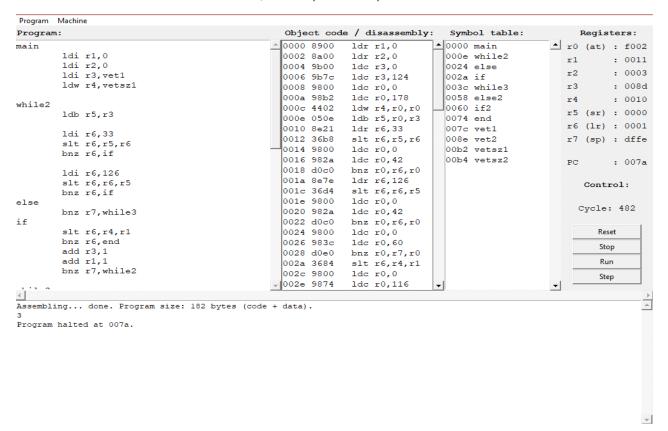
Após implementar os 4 registradores iniciais, foi iniciado os ciclos dos 2 *whiles* que contabilizam as palavras:

- ➡ 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
- ₩hile2: 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*.

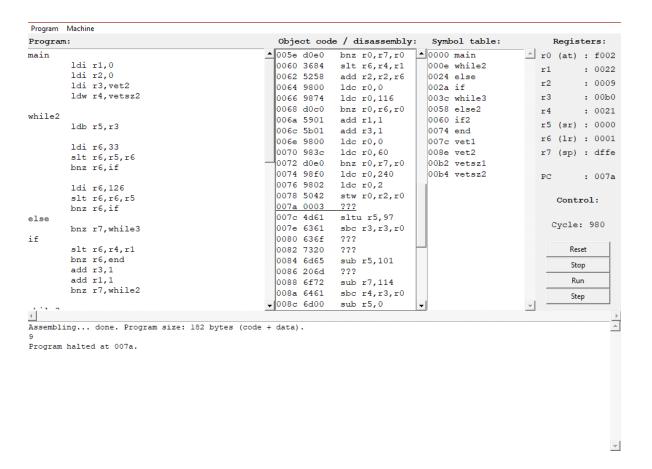
A seguir, será mostrado a execução do programa com os dois exemplos requeridos no enunciado do problema.

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



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".



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:

- 52-2-731410-39-641
- 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:

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

Análise do Sistema:

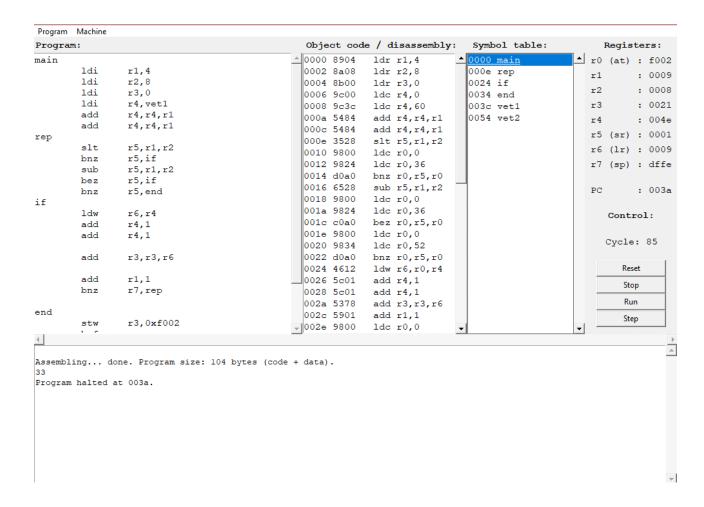
Após implementar os 4 registradores iniciais, foi iniciado os ciclos do *rep* que somam os números necessários:

- **▼ 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.

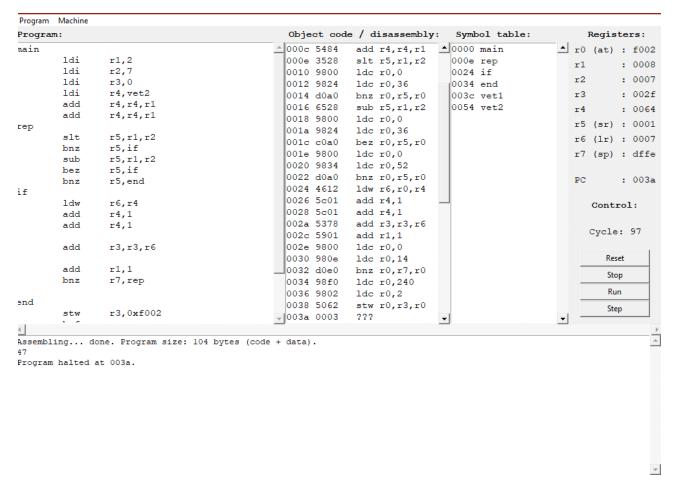
A seguir, será mostrado a execução do programa com os dois exemplos requeridos no enunciado do problema.

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



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



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