

Trabalho 2 de Fundamentos de Sistemas Computacionais - Computador Viking

Nomes: Guilherme Farias Stefani, Henrique Cardoso Zanette.

Turma: 127

Problema 1: Escreva um programa que percorre um vetor de 25 números e apresenta ao final de sua execução a soma dos mesmos.

```
main
    ldi r2,0
    ldi r3,0
    ldi r5,25
    ldi r6,end
    bnz r7,loop
    hcf

loop
    ldi r1,numbers
    add r1,r1,r2
    add r1,r1,r2
    ldw r1,r1
    add r3,r3,r1
    add r2,1
    sub r4,r2,r5
    bez r4,r6
    bnz r7,loop

end
    ldb r1,str
    stw r1,0xf000
    stw r3,0xf002
    stw r1,0xf000
    hcf

str    "\n"
N      25
numbers 25 24 23 22 21 20 19 18 17 1
```

Imagem 1 - Código do problema 1.

Para resolver este problema, primeiro inicializamos os registradores r2,r3 e r5 com os valores de 0, 0 e 25 respectivamente. O primeiro registrador servirá de contador, o segundo registrador armazenará os resultados das somas e o terceiro registrador armazenará o tamanho do vetor. Após isso, no registrador r6 salvamos o retorno da função e então partimos para o *loop*, através da instrução “bnz r7, loop”, que sempre terá retorno verdadeiro.

Dentro do *loop* os valores são carregados e armazenados, as duas instruções “add” são utilizadas já que estamos trabalhando com valores inteiros que ocupam 2 bytes, então precisamos avançar 2 bytes no vetor para chegar na próxima posição. Então armazenamos no registrador r3 o valor de r3 somado ao valor atribuído a r1, e incrementamos o valor de r2 em um, para manter o incremento gradativo.

Para manter o *loop* funcionando corretamente, checamos e armazenamos no registrador r4 se o valor de r2 já atingiu o valor de 25 através de uma subtração. Se o retorno for 0, o fluxo do programa será desviado para o endereço de memória de r6, caso não retorne 0, o *loop* será

continuado pela instrução “bnz r7,loop”. No fim do laço, o valor contido no registrador r3 é exibido em tela pela instrução “stw r3, 0xf002” e o programa é finalizado.

Resultado:

```
Assembling... done. Program size: 122 bytes (code + data).
325
Program halted at 0042.
```

Imagem 2 - Resultado do problema 1.

A soma de um vetor com os valores de 1 a 25 resulta em 325, como demonstrado corretamente no programa.

```
Registers:
r0 (at)  : f000
r1       : 000a
r2       : 0019
r3       : 0145
r4       : 0000
r5 (sr)  : 0019
r6 (lr)  : 002a
r7 (sp)  : dffe

PC       : 0042
```

Imagem 3 - Registradores do problema 1.

```

7% Memory dump
0000: 8a00 8b00 8d19 9e00 9e2a 9800 9812 d0e0 | .....*.....|
0010: 0003 9900 9948 5128 5128 4106 5364 5a01 | .....HQ(Q(A.SdZ.|
0020: 6454 c098 9800 9812 d0e0 9800 9844 0102 | dT.....D..|
0030: 98f0 9800 5022 98f0 9802 5062 98f0 9800 | ....P"....Pb....|
0040: 5022 0003 0a00 0019 0019 0018 0017 0016 | P".....|
0050: 0015 0014 0013 0012 0011 0010 000f 000e | .....|

```

Imagem 4 - Memory Dump do problema 1.

Problema 2: Implemente um programa que conta o número de palavras armazenadas em uma string e apresenta o total no terminal.

```

main
    ldi r3,0
    ldi r5,string
    ldi r6,end
    bnz r7,loop

end
    stw r3,0xf002
    hcf

loop
    ldb r4,r5
    bez r4,check-end
    sub r4,32
    bez r4,space

continue
    and r2,r4,r4
    add r5,1
    bnz r7,loop

space
    bez r2,continue
    add r3,1
    bnz r7,continue

check-end
    bez r2,r6
    add r3,1
    bnz r7,r6

string "Hello Word Test"

```

Imagem 5 - Código completo problema 2.

Para solucionar este problema, é necessário inicializar os registradores r3 com o valor 0, e o registrador r5 com o valor de nossa *string*. O registrador r3 armazenará a quantidade de palavras durante a execução do programa, enquanto o registrador r5 será responsável por acessar o conteúdo da *string*.

No registrador r6 será salvo o retorno da função, e então podemos partir para o *loop*, através da instrução “bnz r7, loop”, que será sempre verdadeiro. Dentro do *loop*, é carregado dentro do registrador r4, o caractere na posição atual da *string*, através da instrução “ldb r4,r5”, cujo endereço de memória está armazenado em r5. Caso o caractere armazenado em r4 seja nulo, o *loop* será encerrado e a função “check-end” para uma verificação final, que serve para contabilizar a última palavra encontrada na *string*, porém só devemos incrementar o contador de palavras se o último caractere não for um espaço.

Assim, consultados o valor de r2, se ele for 0, sabemos que o caractere era anterior era um espaço, e, portanto, não incrementamos o contador de palavras. Se o valor do registrador não for 0, então incrementamos em uma unidade o valor de r3. Após a validação final, o programa é desviado para o endereço de retorno de função armazenado em r6.

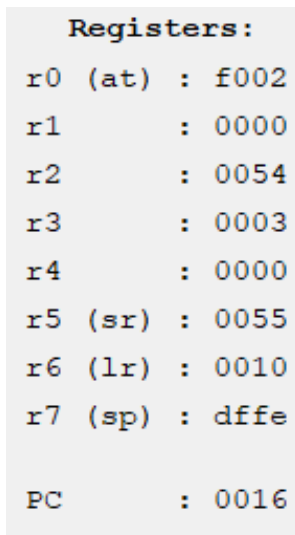
Se o valor do caractere armazenado em r4 não seja nulo, subtraímos 32 do valor contido em r4. Já que o número em base decimal 32 representa o caractere de espaço na tabela ASCII. Caso a subtração resulte em 0, r4 armazenará 0, e isso significa que um caractere de espaço na *string* foi encontrado. Podendo assim, desviar o fluxo do programa para a função “space”, através da instrução “bez r4, space”.

Dentro de “space”, é realizada a verificação se o valor armazenado em r2 é diferente de 0, pois r2 serve para armazenar o valor antigo de r4. Quando o valor de r4 é 0, significa que um espaço foi encontrado, porém, não é de nosso interesse que r4 e r2 estejam armazenando este valor, considerando que isso implicaria dois espaços consecutivos na frase. Se este for o caso, não deve-se contabilizar em r3 mais uma palavra, por isso o programa é desviado para a função “continue”. Entretanto, se r4 for 0 e r2 diferente de 0, incrementamos o registrador r3, que é responsável por manter um rastreamento da quantidade de palavras presentes na *string*. Assim, o incremento mencionado é feito com a instrução “add r3,1”, e somente após isso o fluxo parte para a função “continue”.

Na função “continue”, primeiro armazena-se o valor atual de r4 em 42, fazendo uma cópia através da operação “and r2,r4,r4”, incrementa-se em 1 posição o registrador r5, para acessar a próxima posição da *string* no próximo loop, e voltamos ao início com a instrução “bnz r7, loop”. No final do *loop* o valor armazenado em r3 é exibido em tela pela função “stw r3, 0xf002” e o programa é finalizado.

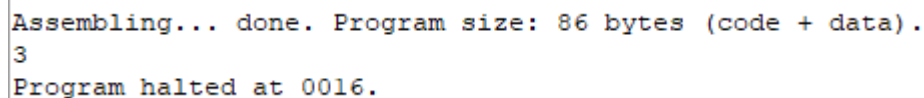
Resultados:

Com a *string* “Hello World Test”, o valor esperado de palavras é 3. E ao final da execução do programa, o registrador r3 estava armazenando o valor previsto.



Registers:	
r0 (at)	: f002
r1	: 0000
r2	: 0054
r3	: 0003
r4	: 0000
r5 (sr)	: 0055
r6 (lr)	: 0010
r7 (sp)	: dffe
PC	: 0016

Imagem 6 - Registradores do problema 2.



```
Assembling... done. Program size: 86 bytes (code + data).
3
Program halted at 0016.
```

Imagem 7 - Resultado da execução do programa.

```
7% Memory dump
0000: 8b00 9d00 9d46 9e00 9e10 9800 9818 d0e0 |.....F.....|
0010: 98f0 9802 5062 0003 0416 9800 9840 c080 |....Pb.....@..|
0020: 6c20 9800 9832 c080 0290 5d01 9800 9818 |l ...2.....]|
0030: d0e0 9800 9828 c040 5b01 9800 9828 d0e0 |.....(.@[.....(|
0040: c058 5b01 d0f8 4865 6c6c 6f20 576f 7264 |.X[...Hello Word|
0050: 2054 6573 7400 0000 0000 0000 0000 0000 | Test.....|
0060: 0000 0000 0000 0000 0000 0000 0000 0000 |                |
```

Imagem 8 - *Dump* de memória do programa 2.

REFERÊNCIAS:

Viking CPU - Manual de referência v0.5, 31 de dez de 2018. Acesso em: 11 de junho de 2021.

Code using RISC-V a program that gives the largest value in a sequence, 2 de fev. de 2020.

Disponível em <<https://stackoverflow.com/questions/60022937/code-using-risc-v-a-program-that-gives-the-largest-value-in-a-sequence>>. Acesso em: 13 de junho de 2021.

Assembly - Arithmetic Instructions, .

Disponível em

<https://www.tutorialspoint.com/assembly_programming/assembly_arithmetic_instructions.htm>.

Acesso em : 13 de junho de 2021.

Finding maximum or minimum of 2 numbers without using jmps, 26 de nov de 2003. Disponível em <<http://www.asmcommunity.net/forums/topic/?id=16187>> Acesso: 14 de junho de 2021.