Trabalho 2 de Organização e Arquitetura de Computadores

Nomes: Arthur Sudbrack Ibarra, Gustavo Giongo Lottermann, Kevin Ribas
Professor: Sérgio Johann Filho
Turma: 127

Problema 1: Implemente um programa que conta o número de palavras armazenadas em uma string e apresenta o total no terminal. O separador de palavras pode ser um número qualquer de espaços, e o seu programa deve considerar isso.

```
main
    ldi r3,0
    ldi r5,string

    ldi r6,end

    bnz r7,repeat
end
    stw r3,0xf002
    hcf
repeat
    ldb r4,r5

    bez r4,end-of-word

    sub r4,32

    bez r4,check-spaces
continue
    and r2,r4,r4

    add r5,1

    bnz r7,repeat
check-spaces
    bez r2,continue
    add r3,1
    bnz r7,continue
end-of-word
    bez r2,r6
    add r3,1
    bnz r7,r6

string "Melancia Hipopotamo    Couve"
```

Solução: Para a resolução do problema, devemos realizar os seguinte passos:

- Inicializar os registradores r3 e r5 com o valor 0 e com o endereço de nossa string, respectivamente. O registrador r3 armazenará a quantidade de palavras encontradas de forma gradual ao decorrer do programa; O registrador r5 será responsável por acessar o conteúdo da string.

- Salvar no registrador r6 onde será dado o retorno da função.

- Ir para o laço de repetição “repeat” através da instrução “bnz r7, repeat”, a qual sempre terá um retorno verdadeiro.

- É carregado em r4, através da instrução “ldb r4,r5”, o caractere na posição atual da string, cujo endereço de memória está armazenado em r5.

- Caso o caractere armazenado em r4 seja nulo, o laço de repetição se dará fim e iremos para o endereço “end-of-word” para realizar uma verificação final. Essa verificação final é aplicada para contabilizar a última palavra encontrada na string, no entanto, nesse caso, somente devemos incrementar o contador de palavras se o último ou últimos caracteres não forem um espaço, visto que o programa contabiliza as palavras por espaços encontrados. Ou seja, "um abacate” possui 2 palavras, da mesma forma que “um abacate “ também deve possuir 2 palavras, por isso devemos fazer essa validação. Para isso, consultamos o valor de r2, se este for 0, devido à lógica que está sendo utilizada no programa, sabemos que o caractere anterior era um espaço, e, portanto, não incrementamos o contador de palavras. Se o valor do registrador r2 não for 0, então incrementamos em 1 unidade o valor de r3. Após a validação final, o fluxo do programa é desviado para o endereço de retorno de função, armazenado em r6.

- Caso o caractere armazenado em r4 NÃO seja nulo, então subtraímos 32 do valor contido em r4. Isso porque, o número em base decimal 32 representa o caractere de espaço na tabela ASCII.

- Caso a subtração resulte em 0, r4 estará armazenando o valor 0, e isso significa que encontramos um caractere de espaço na string sendo percorrida. Dessa forma, o fluxo do programa será desviado para “check-spaces” através da instrução “bez r4,check-spaces”.

- Dentro de “check-spaces”, estamos verificando se o valor armazenado em r2 é diferente de 0. Isso é feito pois r2 serve para armazenar o valor antigo de r4. Sabemos que quando r4 é 0, então é porque havíamos encontrado um caractere de espaço em nossa string, no entanto, não é de nosso interesse que tanto r4 quanto r2 estejam armazenando o valor 0, considerando que isso implica que existe mais de um espaço consecutivo na frase. Se este for o caso, não devemos contabilizar em r3 mais uma palavra, e, por este motivo, o programa é desviado para “continue”. Todavia, se r4 for 0 e r2 for diferente de 0, então nesse caso devemos incrementar o

nosso registrador r3, responsável por manter um rastreamento da quantidade de palavras presente na string. Assim, o incremento mencionado é feito com a instrução “add r3,1”, e somente após isso, desviamos o fluxo do programa para “continue”.

- Dentro de “continue”, primeiramente armazenamos o valor atual de r4 em r2, fazendo-se uma cópia através da operação “and r2,r4,r4”

- Incrementamos em 1 posição o registrador r5 para que possamos acessar a próxima posição da string na próxima repetição do laço.

- Voltamos ao início do laço de repetição com a instrução “bnz r7,repeat”.

- Ao final do laço de repetição e da função, o valor armazenado em r3 (quantidade de palavras) é exibido na tela através da instrução “stw r3,0xf002” e o programa é finalizado.

Resultados Obtidos:

Com a string sendo "Melancia Hipopotamo Couve", o valor esperado da quantidade de palavras é 3. Ao final da execução do programa, o registrador r3 estava armazenando o valor previsto. Isso pôde ser constatado no teste abaixo:

```
   Registers:
r0 (at) : f002
r1      : 0000
r2      : 0045
r3      : 0003
r4      : 0000
r5 (sr) : 0062
r6 (lr) : 0010
r7 (sp) : dffe

PC      : 0016

   Control:

  Cycle: 405
```

```
Assembling... done. Program size: 100 bytes (code + data).
3
Program halted at 0016.
```

Já com a string sendo "PimentaMalagueta Sanduiche ", o valor esperado da quantidade de palavras é 2. Ao final da execução do programa, o registrador r3 estava armazenando o valor previsto. Podemos evidenciar isso no teste abaixo:

```
   Registers:
r0 (at) : f002
r1      : 0000
r2      : 0000
r3      : 0002
r4      : 0000
r5 (sr) : 0061
r6 (lr) : 0010
r7 (sp) : dffe

PC      : 0016

   Control:

  Cycle: 381
```

```
Assembling... done. Program size: 98 bytes (code + data).
2
Program halted at 0016.
```

Problema 2: Escreva um programa que percorre um vetor de 15 números e apresenta ao final de sua execução a soma dos mesmos.

```
main
    ldi r2,0
    ldi r3,0
    ldi r5,15

    ldi r6,end

    bnz r7,repeat

end
    stw r3,0xf002
    hcf
repeat
    ldi r1,array
    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,repeat

array 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

Solução: Para a resolução do problema, devemos realizar os seguinte passos:

- Inicializar os registradores r2, r3 e r5 com os valores 0, 0 e 15, respectivamente. O registrador r2 irá servir de contador até 15 para que possamos percorrer exatamente a quantidade de posições do array; O registrador r3 servirá para armazenar os resultados contínuos das somas dos valores contidos no array; O registrador r5 armazena o tamanho do array.

- Salvar no registrador r6 onde será dado o retorno da função.

- Ir para o laço de repetição “repeat” através da instrução “bnz r7, repeat”, a qual sempre terá um retorno verdadeiro.

- Dentro do laço de repetição, as primeiras 4 instruções são responsáveis por carregar o array e armazenar os valores contidos nele, um por um, no registrador r1. Note que as duas instruções “add” são apresentadas ali, pois, como estamos trabalhando com valores inteiros,

cada um ocupa 2 bytes, e, portanto, é preciso avançar 2 bytes no array para acessar uma nova posição.

- Após isso, armazenamos em r3 o valor de r3 somado com o valor atribuído à r1, de forma a ir gradativamente incrementando este registrador r3.

- Incrementamos o valor de r2 em 1 unidade.

- Checamos e armazenamos no registrador r4 se o valor armazenado em r2 já atingiu 15 através de uma operação de subtração.

- Caso a subtração acima retorne 0, sabemos que é porque o registrador r2 já atingiu 15, uma vez que 15 - 15 = 0. Nesse caso, a operação “bez r4,r6” reconhecerá que dentro de r4 temos o valor 0 e desviará o fluxo do programa para onde deve ser retornada a função, através do endereço de memória que havia sido armazenado em r6.

- Caso a subtração não retorne 0, o processo de repetição dará continuidade através da instrução “bnz r7,repeat”, a qual, incondicionalmente, desviará o fluxo do programa para o laço de repetição novamente.

- Ao fim do laço de repetição, o valor contido no registrador r3 é exibido na tela através da instrução “stw r3,0xf002” e o programa é finalizado.

Resultados Obtidos:

Com o Array valendo 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15, o valor esperado da soma dos valores seria 120. Ao final da execução do programa, o registrador r3 estava armazenando o valor 0078 em hexadecimal, o qual, para a base decimal, se torna 120. Isso pôde ser constatado no teste abaixo:

```
  Registers:
r0 (at) : f002
r1      : 000f
r2      : 000f
r3      : 0078
r4      : 0000
r5 (sr) : 000f
r6 (lr) : 0010
r7 (sp) : dffe

PC      : 0016

   Control:

  Cycle: 188
```

```
Assembling... done. Program size: 78 bytes (code + data).
120
Program halted at 0016.
```

Já, com o Array valendo 10 14 19 20 17 2 68 71 31 1 47 24 100 300 5, o valor esperado da soma dos valores seria 729. Ao final da execução do programa, o registrador r3 estava armazenando o valor 02d9 em hexadecimal, o qual, para a base decimal, se torna 729. Isso pôde ser constatado no teste abaixo:

```
  Registers:
r0 (at) : f002
r1      : 0005
r2      : 000f
r3      : 02d9
r4      : 0000
r5 (sr) : 000f
r6 (lr) : 0010
r7 (sp) : dffe

PC      : 0016

   Control:

  Cycle: 188
```

```
Assembling... done. Program size: 78 bytes (code + data).
729
Program halted at 0016.
```

Problema 3: Faça um programa que conta o número de elementos pares e ímpares de um vetor. Dica: utilize a operação and reg,1 para descobrir se um número é par ou ímpar. Se o número no registrador reg for par o resultado será 0, senão será 1.

```
main
    ldi r2,0
    ldw r3,max
    ldw r5,max

    ldi r6,end

    bnz r7,repeat

end
    stw r3,0xf002
    ldi r4,0x0a
    stw r4,0xf000
    sub r4,r5,r3
    stw r4,0xf002
    hcf
repeat
    ldi r1,array
    add r1,r1,r2
    add r1,r1,r2
    ldw r1,r1

    and r1,1

    sub r3,r3,r1

    add r2,1

    sub r4,r2,r5

    bez r4,r6

    bnz r7,repeat

array 1 2 3 4 5 6 7 8 9 10
max 10
```

Solução: Para a resolução do problema, devemos realizar os seguinte passos:

- Inicializar os registradores r2, r3 e r5 com os valores 0, max e max, respectivamente. O registrador r2 irá servir de contador até o valor definido na variável max para que possamos percorrer exatamente a quantidade de posições do array; O registrador r3 servirá para gradativamente armazenar a quantidade de valores pares contidos no array; O registrador r5 armazena fixamente o valor de max, ou seja, o tamanho do array.

- Salvar no registrador r6 onde será dado o retorno da função.

- Ir para o laço de repetição “repeat” através da instrução “bnz r7, repeat”, a qual sempre terá um retorno verdadeiro.

- Dentro do laço de repetição, as primeiras 4 instruções são responsáveis por carregar o array e armazenar os valores contidos nele, um por um, no registrador r1. Note que as duas instruções “add” são apresentadas ali, pois, como estamos trabalhando com valores inteiros, cada um ocupa 2 bytes, e, portanto, é preciso avançar 2 bytes no array para acessar uma nova posição.

- Armazenamos novamente dentro de r1 o resultado da operação “and r1,1”. Essa operação é responsável por indicar se o valor contido dentro do registrador r1 é par ou ímpar, sendo que se ele for par, a operação retornará 0, caso o contrário, 1.

- Armazenamos em r3 a subtração do valor já contido em r3 com o valor obtido para r1. Isto porque, a cada número ímpar, estaremos diminuindo 1 unidade de nosso total de números, de modo que, ao final da execução do programa, r3 armazenará a diferença entre a quantidade total de elementos do array e a quantidade de números ímpares deste array, ou seja, os números pares.

- Incrementamos o valor de r2 em 1 unidade.

- Checamos e armazenamos no registrador r4 se o valor armazenado em r2 já atingiu max através de uma operação de subtração.

- Caso a subtração acima retorne 0, sabemos que é porque o registrador r2 já atingiu max, uma vez que max - max = 0. Nesse caso, a operação “bez r4,r6” reconhecerá que dentro de r4 temos o valor 0 e desviará o fluxo do programa para onde deve ser retornada a função, através do endereço de memória que havia sido armazenado em r6.

- Caso a subtração não retorne 0, o processo de repetição dará continuidade através da instrução “bnz r7,repeat”, a qual, incondicionalmente, desviará o fluxo do programa para o laço de repetição novamente.

- Ao fim do laço de repetição, o valor contido no registrador r3 (quantidade de números pares) é exibido na tela através da instrução “stw r3,0xf002”. Além disso, a quantidade de números ímpares será armazenada em r4 através da diferença entre o total de elementos do array (r5) e

a quantidade de números pares (r3), e essa quantidade também será exibida na tela com a instrução “r4,0xf002”. Note que as instruções “ldi r4,0x0a” e “stw r4,0xf000” são utilizadas apenas para adicionar uma quebra de linha entre a exibição da quantidade de números pares e ímpares.

Resultados Obtidos:

Com um Array de tamanho 10 (max = 10) e com os valores valendo 1 2 3 4 5 6 7 8 9 10 é esperado que número de elementos pares seja 5 e de ímpares seja 5 também. Ao final da execução do programa, chegamos ao resultado esperado, pois o registrador r3 está armazenando o valor 5, o qual representa o número de elementos pares, já o registrador r4 está armazenando o valor 5 e representa o número de elementos ímpares. Isso pôde ser constatado no teste abaixo:

```
  Registers:
r0 (at) : f002
r1      : 0000
r2      : 000a
r3      : 0005
r4      : 0005
r5 (sr) : 000a
r6 (lr) : 0018
r7 (sp) : dffe

PC      : 0030

  Control:

 Cycle: 151
```

```
Assembling... done. Program size: 98 bytes (code + data).
5
5
Program halted at 0030.
```

Com um Array de tamanho 5 (max = 5) e com os valores valendo 28 19 47 21 2 é esperado que número de elementos pares seja 2 e de ímpares seja 3. Ao final da execução do programa chegamos ao resultado esperado, pois o registrador r3 está armazenando o valor 2, o qual representa o número de elementos pares, já o r4 está armazenando o valor 3 e representa o número de elementos ímpares. Isso pôde ser constatado no teste abaixo:

```
  Registers:
r0 (at) : f002
r1      : 0000
r2      : 0005
r3      : 0002
r4      : 0003
r5 (sr) : 0005
r6 (lr) : 0018
r7 (sp) : dffe

PC      : 0030
```

```
Assembling... done. Program size: 88 bytes (code + data).
2
3
Program halted at 0030.
```