Prova P1 **Disciplina: Organização de Computadores**

Professor: Ney Laert Vilar Calazans

Aluno: 18/abril/2012

Lista de associação de números e mnemônicos para os registradores do MIPS

Número (Decimal)	Nome
0	$zero
1	$at
2	$v0
3	$v1
4	$a0
5	$a1
6	$a2
7	$a3
8	$t0
9	$t1
10	$t2
11	$t3
12	$t4
13	$t5
14	$t6
15	$t7

Número (Decimal)	Nome
16	$s0
17	$s1
18	$s2
19	$s3
20	$s4
21	$s5
22	$s6
23	$s7
24	$t8
25	$t9
26	$k0
27	$k1
28	$gp
29	$sp
30	$fp
31	$ra

1. (4,0 pontos) Montagem/Desmontagem de código objeto. Abaixo se mostra parte de uma listagem gerada pelo ambiente MARS como resultado da montagem de um programa. Note que o trecho foi montado a partir do endereço de memória que não é o inicial do Mars. Calcule o endereço inicial. Pede-se: (a) Substituir as triplas interrogações pelo texto que deveria estar em seu lugar (existem 8 triplas ??? a substituir). Em alguns casos, isto implica gerar código objeto, enquanto em outros implica gerar código intermediário e/ou código fonte e/ou endereços. Caso uma instrução seja de salto, expresse o exato endereço para onde ela salta em hexadecimal, caso isto seja parte das interrogações.
   Dica 1: Prestem muita atenção ao tratamento de endereços e rótulos.
   Dica 2: Muito cuidado com a mistura de representações numéricas: hexa, binário, complemento de 2, etc.
   Obrigatório: Mostre os desenvolvimentos para obter os resultados nas folhas anexas, justificando este desenvolvimento.

```
     Endereço     Cód. Objeto Cód. Intermediário          Código Fonte
[1]      ???          ???          ???                75          li     $t4,0
[2]  0x00400088       ???      beq $10,$0,0x0006      76          beq    $t2,$zero,prim
[3]  0x0040008c   0xafdefffc   sw $30,0xfffc($30)     77          sw     $fp,-4($fp)
[4]  0x00400090   0x27de0008   addiu $30,$30,0x0008   79   cont:  addiu  $fp,$fp,8
[5]  0x00400094       ???          ???                80          sw     $t3,-8($fp)
[6]  0x00400098   0xafccffc    sw $12,0xfffc($30)     81          sw     $t4,-4($fp)
[7]  0x0040009c   0x2529ffff   addiu $9,$9,0xffff     82          addiu  $t1,$t1,-1
[8]  0x004000a0   0x08100019       ???                83              ???
```

2. (3,0 pontos) Verdadeiro ou Falso. Abaixo aparecem 10 afirmativas. Marque com V as afirmativas verdadeiras e com F as falsas. Se não souber a resposta correta, deixe em branco, pois cada resposta correta vale 0,3 pontos, mas cada resposta incorreta desconta 0,2 pontos do total positivo de pontos. Não é possível que a questão produza uma nota menor do que 0,0 pontos.
   a) ( ) Uma arquitetura baseada em um único acumulador possui necessariamente um único registrador na sua implementação em hardware.
   b) ( ) O modo de endereçamento base-deslocamento, conforme usado no MIPS, soma o conteúdo de dois registradores do banco de registradores do MIPS para obter um endereço de memória.
   c) ( ) Se um processador usa apenas instruções cujo código objeto ocupa exatamente 8 bits, o número máximo de instruções distintas que o processador pode ter no seu conjunto de instruções é 256.
   d) ( ) O campo de dado imediato nas instruções do MIPS como addi, ori e xori no MIPS permite especificar qualquer dado que caiba em um registrador do banco de registradores do MIPS.
   e) ( ) A instrução jal do MIPS não usa uma estrutura de pilha em memória.

f) ( ) O código objeto **0x110B3FF6** corresponde a uma instrução beq que quando saltar, o faz necessariamente para uma linha do programa anterior à linha onde há o beq.
g) ( ) A instrução **ori $t0,$t0,0xFFFF** escreve em $t0 um número natural maior ou igual ao valor que havia anteriormente neste registrador.
h) ( ) O código objeto **0x0c100014** corresponde a um salto para uma subrotina que inicia necessariamente em uma posição de memória abaixo da posição onde se encontra o salto.
i) ( ) A memória do MIPS é endereçada a byte, mas é possível escrever nela dados de 8, 16 ou 32 bits usando uma única instrução do processador.
j) ( ) Não é possível somar duas constantes de 64 bits usando o processador MIPS.

3. (3,0 pontos) Dado o programa abaixo, escrito em linguagem de montagem do MIPS, responda às questões que seguem sobre o mesmo.

```
[1]            .data
[2]    cad:    .word   0x41 0x6C 0x6F 0x20 0x4D 0x61 0x6D 0x61 0x65 0x0
[3]    caso:   .word   0x0
[4]
[5]            .text
[6]            .globl  main
[7]    main:
[8]            la      $t0,cad
[9]            lw      $t1,0($t0)
[10]           la      $t3,caso
[11]           lw      $t4,0($t3)
[12]   loop:   blez    $t1,end
[13]           sltiu   $t2,$t1,65
[14]           bne     $t2,$zero,nxt_c
[15]           sltiu   $t2,$t1,91
[16]           beq     $t2,$zero,nxt_c
[17]           addiu   $t4,$t4,1
[18]   nxt_c: addiu    $t0,$t0,4
[19]           lw      $t1,0($t0)
[20]           j       loop
[21]
[22]   end:    sw      $t4,0($t3)
[23]
[24]           li      $v0,10
[25]           syscall
```

Pede-se:

- (1,0 ponto) Diga o que faz o programa acima (do ponto de vista semântico), detalhando seu funcionamento, e diga o conteúdo final de posições de memória alteradas, se for o caso. Comente semanticamente o texto do programa.
- (1,0 ponto) O programa contém alguma subrotina/função? O programa escreve algo na memória externa ao processador? Qual ou quais registradores possuem resultado relevante ao fim da execução?
- (1,0 ponto) Calcule qual o tamanho do programa, em bytes. Qual o tamanho da área de dados, em bytes?

Prova P1 **Disciplina: Organização de Computadores**

Gabarito **Professor: Ney Laert Vilar Calazans**

18/abril/2012

Lista de associação de números e mnemônicos para os registradores do MIPS

Número (Decimal)	Nome
0	$zero
1	$at
2	$v0
3	$v1
4	$a0
5	$a1
6	$a2
7	$a3
8	$t0
9	$t1
10	$t2
11	$t3
12	$t4
13	$t5
14	$t6
15	$t7

Número (Decimal)	Nome
16	$s0
17	$s1
18	$s2
19	$s3
20	$s4
21	$s5
22	$s6
23	$s7
24	$t8
25	$t9
26	$k0
27	$k1
28	$gp
29	$sp
30	$fp
31	$ra

1. (4,0 pontos) Montagem/Desmontagem de código objeto. Abaixo se mostra parte de uma listagem gerada pelo ambiente MARS como resultado da montagem de um programa. Note que o trecho foi montado a partir do endereço de memória que não é o inicial do Mars. Calcule o endereço inicial. Pede-se: (a) Substituir as triplas interrogações pelo texto que deveria estar em seu lugar (existem 8 triplas ??? a substituir). Em alguns casos, isto implica gerar código objeto, enquanto em outros implica gerar código intermediário e/ou código fonte e/ou endereços. Caso uma instrução seja de salto, expresse o exato endereço para onde ela salta em hexadecimal, caso isto seja parte das interrogações.
   Dica 1: Prestem muita atenção ao tratamento de endereços e rótulos.
   Dica 2: Muito cuidado com a mistura de representações numéricas: hexa, binário, complemento de 2, etc.
   Obrigatório: Mostre os desenvolvimentos para obter os resultados nas folhas anexas, justificando este desenvolvimento.

```
     Endereço     Cód. Objeto Cód. Intermediário          Código Fonte
[1]      ???          ???          ???              75          li     $t4,0
[2]  0x00400088       ???      beq $10,$0,0x0006    76          beq    $t2,$zero,prim
[3]  0x0040008c  0xafdefffc   sw $30,0xfffc($30)    77          sw     $fp,-4($fp)
[4]  0x00400090  0x27de0008   addiu $30,$30,0x0008  79   cont:  addiu  $fp,$fp,8
[5]  0x00400094       ???          ???              80          sw     $t3,-8($fp)
[6]  0x00400098  0xafccfffc   sw $12,0xfffc($30)    81          sw     $t4,-4($fp)
[7]  0x0040009c  0x2529ffff   addiu $9,$9,0xffff    82          addiu  $t1,$t1,-1
[8]  0x004000a0  0x08100019       ???              83              ???
```

Solução da Questão 1 (4,0 pontos). Cada ??? vale 0,5 pontos

```
[1]      ???          ???          ???              75          li     $t4,0
```

(1) Obviamente, a questão consiste em demonstrar conhecimento de como se monta códigos intermediário e objeto de uma instrução. Note-se que se trata de uma pseudo-instrução (li – load immediate) que escreve a constante 0 no registrador $t4. É possível realizar isto com exatamente uma instrução do MIPS. De fato há várias possibilidades. Uma delas é transformar **li $t4,0** em **addiu $t4,$zero,0**. Usarei esta possibilidade e gerarei código para a instrução em questão. Primeiro, o código intermediário é a instrução escolhida, com os nomes mnemônicos dos registradores transformados em nomes numéricos, ou seja **addiu $12,$0,0x0000**. Para produzir o código objeto, inicia-se consultando o Apêndice A, para obter o formato de instrução, o que fornece:

```
addiu rt, rs, immed
  9   rs rt immed
Número de bits/campo: 6   5  5   16
```

A partir daí, e sabendo-se que o campo imed do formato corresponde à constante, já se pode obter diretamente os 32 bits do código objeto, seguindo a ordem dada no formato. O

que se obtém é 001001 (0x9 em 6 bits), 01100 ($t4=$12, logo 12 em binário 5 bits, Rs), 00000 ($zero=$0, logo 0 em binário 5 bits, Rt) e 0000 0000 0000 0000 (0 em binário de 16 bits). Juntando os 4 campos, obtém-se 0010 0100 0000 1100 0000 0000 0000 0000, ou, em hexadecimal: **0x240C0000**, que é o código objeto da instrução. Como se trata de apenas uma instrução, ela ocupa 4 bytes. Assim, o último conjunto de ??? deve ser substituído pelo endereço que se obtém subtraindo 4 do endereço da linha seguinte, ou seja 0x00400088-8=**0x00400084**.

Resposta final:

```
[1] 0x00400084 0x240C0000  addiu $12,$0,0x0000     75          li     $t4,0

[2] 0x00400088      ???     beq $10,$0,0x0006      76          beq    $t2,$zero,prim
```

(2) Neste item, o objetivo é gerar o código objeto da instrução beq, usando o código fonte e o código intermediário, ambos dados. Usando o Apêndice A do livro-texto encontra-se o formato desta instrução, que é:

beq	**rs,**	**rt,**	**label**	
0x4	**rs**	**rt**	**offset**	
Número de bits/campo:	**6**	**5**	**5**	**16**

Agora basta gerar o código em binário, usando o formato e os dados constantes no código fonte e no código intermediário. O resultado é 000100 (0x4 em binário, 6 bits), 01010 ($10=Rs em 5 bits, tirado do código intermediário), 00000 ($0=Rt em 5 bits, tirado do código intermediário) e 0000000000000110 (0x0006, o offset em binário 16 bits). Juntando os campos em um único código de 32 bits na ordem do formato, reagrupados de 4 em 4, tem-se 0001 0001 0100 0000 0000 0000 0000 0110. Em hexa, isto dá **0x11400006**, que é o código objeto desejado.

Resposta final:

```
[2] 0x00400088  0x11400006 beq $10,$0,0x0006     76          beq    $t2,$zero,prim

[5] 0x00400094      ???         ???              80          sw     $t3,-8($fp)
```

(3) Neste item, novamente o objetivo é a montagem de códigos objeto e intermediário. O ponto de partida é de novo encontrar o formato da instrução, que é:

sw	**rt,**	**address**		
0x2b	**rs**	**rt**	**offset**	
Número de bits/campo:	**6**	**5**	**5**	**16**

Esta instrução usa modo de endereçamento base-deslocamento para calcular o endereço (address) os campos **rs** e **offset** do formato são usados respectivamente com o registrador base e o deslocamento. Assim, o código intermediário fica **sw $11, 0xFFF8($30)**, pois 30 é o índice (endereço) do registrador $fp no banco de registradores (veja tabela no início da prova), e 0xFFF8 é -8 (expresso em complemento de 2, 16 bits e codificado em hexadecimal). Com o formato e estes valores se pode gerar o código objeto, que fica 101011 (0x2B em binário 6 bits), 11110 (rs, 30 em 5 bits), 01011 (rt, 11 em 5 bits) e 1111111111111000 (0xFFF8 em binário 16 bits). Juntando os campos na ordem definida no formato e separando os bits de 4 em 4, tem-se: 1010 1111 1100 1011 1111 1111 1111 1000, que é o código objeto da instrução. Em hexa, tem-se que o código objeto fica **0xAFCBFFF8**.

Resposta Final:

```
[5] 0x00400094  0xAFCBFFF8  sw $11, 0xFFF8($30)   80          sw     $t3,-8($fp)

[8] 0x004000a0  0x08100019      ???               83              ???
```

(4) Neste item, o objetivo é a desmontagem de código objeto para produzir os códigos intermediário e fonte. O ponto de partida é gerar a descrição do código objeto em binário (0000 1000 0001 0000 0000 0000 0001 1001) e observar os 6 bits mais significativos do código objeto para descobrir de que instrução se trata. O resultado é 000010, ou, em hexadecimal 0x2. Usando a Tabela A.19 do Apêndice A do livro-texto (2ª. Edição em inglês), descobre-se que se trata da instrução **j**. Mais adiante no mesmo material se encontra o formato desta instrução, que é:

j	**target**	
0x2	**target**	
Número de bits/campo:	**6**	**26**

Esta instrução usa modo de endereçamento pseudo-absoluto, onde os 26 bits inferiores do código objeto (campo **target**) devem ser acrescidos de dois 0s à direita e de uma cópia dos 4 bits mais significativos do PC no momento da execução à esquerda dos mesmos 26 bits.

O valor resultante é o endereço para onde saltar. Lembra-se que o valor do PC no momento da execução é o endereço da instrução seguinte, que neste caso é 0x004000A4. Assim, ter-se-á como alvo final do salto (**target**) o endereço: 0000 (quatro bits superiores de 0x004000A4) & 00 0001 0000 0000 0000 0001 1001 (26 bits inferiores do código objeto da instrução **j**) & 00, ou 0000 0000 0100 0000 0000 0000 0110 0100, ou, em hexadecimal, 0x00400064. Este endereço é usado em ambos código fonte e intermediário, uma vez que o rótulo se encontra fora do trecho fornecido na prova. A solução para o código intermediário e código fonte são assim idênticas **j 0x00400064**.

Resposta Final:

```
[8]  0x004000a0  0x08100019  j 0x00400064     83      j 0x00400064
```

Fim da Solução da Questão 1 (4,0 pontos)

2. (3,0 pontos) Verdadeiro ou Falso. Abaixo aparecem 10 afirmativas. Marque com V as afirmativas verdadeiras e com F as falsas. Se não souber a resposta correta, deixe em branco, pois cada resposta correta vale 0,3 pontos, mas cada resposta incorreta desconta 0,2 pontos do total positivo de pontos. Não é possível que a questão produza uma nota menor do que 0,0 pontos.

Solução da Questão 2 (3,0 pontos)

a) (F) Uma arquitetura baseada em um único acumulador possui necessariamente um único registrador na sua implementação em hardware.
Falsa, pois como visto em aula, pelo menos alguns registradores de controle são necessários ao implementar uma arquitetura de processador em uma organização de hardware, tais como o Contador de Programa (PC) e o Registrador de Instruções.

b) (F) O modo de endereçamento base-deslocamento, conforme usado no MIPS, soma o conteúdo de dois registradores do banco de registradores do MIPS para obter um endereço de memória.
Falsa, este modo de endereçamento na realidade soma o conteúdo de um registrador base com uma constante que ocupa um campo de 16 bits no código objeto da instrução, representando um valor em complemento de 2, resultando em um endereço de memória de 32 bits.

c) (V) Se um processador usa apenas instruções cujo código objeto ocupa exatamente 8 bits, o número máximo de instruções distintas que o processador pode ter no seu conjunto de instruções é 256.
Verdadeira, pois com 8 bits pode-se ter 2^8=256 códigos distintos. Se cada um destes representar uma instrução distinta do processador chega-se ao máximo de instruções. Na prática, o número seria menor, caso as instruções possuam campos para especificar operandos diversos, como registradores a usar, constantes, etc.

d) (F) O campo de dado imediato nas instruções do MIPS como addi, ori e xori no MIPS permite especificar qualquer dado que caiba em um registrador do banco de registradores do MIPS.
Falsa, pois o campo de dado imediato nestas e em outras instruções similares possui apenas 16 bits. Assim, apenas valores que podem ser representados em 16 bits podem ser usados em tais instruções. Caso o programador especifique um dado que precisa de mais bits para ser representado, um montador pode gerar código correto (com mais de uma instrução e a linha então corresponde a uma pseudo-instrução e não a uma instrução.

e) (V) A instrução jal do MIPS não usa uma estrutura de pilha em memória.
Verdadeira, ela usa um registrador do banco ($ra=$31) para guardar o endereço de retorno da subrotina. Qualquer manipulação de estruturas de pilha é responsabilidade do programador ou do compilador.

f) (F) O código objeto 0x110B3FF6 corresponde a uma instrução beq que quando saltar, o faz necessariamente para alguma linha do programa anterior à linha onde há o beq.
Falsa. OK, os seis primeiros bits da instrução, 000100 correspondem ao valor decimal (ou hexa) 4, o código de operação que denota a instrução beq (ver formato do beq no Apêndice A). Como o campo de Offset (últimos 16 bits do código objeto, 0x3FF6) correspondem a um número positivo em complemento de 2, trata-se de um salto para linha posterior do programa. Ora, o número 0x3FF6 em 16 bits tem o bit mais significativo em 0, sendo portanto um valor positivo. A única situação em que o salto não seria para endereço maior que o da instrução beq ocorre se a soma do valor do PC com

0x00003FF6 der estouro de campo, o que é ignorado aqui, por ser raro e de pouca aplicabilidade na prática.

g) (V) A instrução ori $t0,$t0,0xFFFF escreve em $t0 um número natural maior ou igual ao valor que havia anteriormente neste registrador.
Verdadeira, pois ori faz extensão de 0 sobre o valor antes de operar com o conteúdo de $t0, o que produz 0x0000FFFF. Ora, ao realizar um OU lógico deste valor com o conteúdo de $t0 (interpretando o conteúdo de $t0 como um número natural, ou seja, estritamente positivo) obtém-se como efeito a conversão de todos os bits 0 da metade inferior de $t0 para bits 1 e a manutenção dos valores dos16 bits mais significativos. Isto só pode gerar um resultado que interpretado como número natural é maior ou no pior caso igual ao valor original contido em $t0.

h) (F) O código objeto 0x0c100014 corresponde a um salto para uma subrotina que inicia necessariamente em uma posição de memória abaixo da posição onde se encontra o salto.
Falsa. Efetivamente, trata-se do código objeto de uma instrução jal, pois os seis bits mais significativos do código (000011) correspondem ao valor decimal (ou hexa) 3, que denota a instrução jal (ver formato do jal no Apêndice A). Contudo, a instrução jal usa o modo de endereçamento pseudo-absoluto, onde o valor do operando da instrução (os 26 bits menos significativos do código objeto) não depende da relação entre a posição para onde se quer saltar e a posição onde se encontra o salto. Assim o salto em questão pode ser para posição acima, abaixo ou mesmo para a posição onde se encontra o salto, sendo impossível determinar, apenas a partir do código objeto, qual destas situações ocorre.

i) (V) A memória do MIPS é endereçada a byte, mas é possível escrever nela dados de 8, 16 ou 32 bits usando uma única instrução do processador.
Verdadeira. De fato, como salientado em múltiplas ocasiões ao longo deste curso, o MIPS assume que cada endereço de memória contém exatamente um byte de informação. A escrita na memória pode ser realizada usando as instruções sw (escreve 32 bits na memória), sh (escreve 16 bits na memória) ou sb (escreve 8 bits na memória).

j) (F) Não é possível somar duas constantes de 64 bits usando o processador MIPS.
Falsa. Embora não exista uma instrução no MIPS que possa somar dois valores de 64 bits, é obviamente possível escrever um programa que executa no processador MIPS para somar duas constantes de 64 bits, realizando as somas em partes de 32 bits (ou menos) a partir da direita dos operandos e propagando valores de vai-um para as partes mais à esquerda dos operandos. Aliás, o desenvolvimento de tal programa foi alvo de exercício em laboratório.

Fim da Solução da Questão 2 (3,0 pontos)

3. (3,0 pontos) Dado o programa abaixo, escrito em linguagem de montagem do MIPS, responda às questões que seguem sobre o mesmo.

```
[1]                 .data
[2]      cad:     .word   0x41 0x6C 0x6F 0x20 0x4D 0x61 0x6D 0x61 0x65 0x0    # Loads string "Alo Mamae" in memory
[3]      caso:    .word   0x0               # Variable to contain the value of upper/lower case letters in string
[4]
[5]               .text
[6]               .globl  main
[7]      main:
[8]               la      $t0,cad           # 8 register $t0 contains the address of the string
[9]               lw      $t1,0($t0)        # 4 get first character
[10]              la      $t3,caso          # 8 get address of uppercase counter
[11]              lw      $t4,0($t3)        # 4 get value of counter. Initially, it should be zero
[12]     loop:    blez    $t1,end           # 4 if char is 0, end of string
[13]              sltiu   $t2,$t1,65        # 4 if char is smaller than letter "A" than it is not uppercase
[14]              bne     $t2,$zero,nxt_c   # 4 if smaller than "A" get next char
[15]              sltiu   $t2,$t1,91        # 4 if smaller than "Z"+1, than it is an uppercase letter
[16]              beq     $t2,$zero,nxt_c   # 4 if bigger than "Z", get next char
[17]              addiu   $t4,$t4,1         # 4 if here, Bingo!, found one uppercase char. Increment case counter
[18]     nxt_c: addiu     $t0,$t0,4         # 4 increment char pointer
[19]              lw      $t1,0($t0)        # 4 get next char
[20]              j       loop              # 4 go back to test for end of string reached
[21]
[22]     end:     sw      $t4,0($t3)        # 4 just store counter of uppercase letters in caso
[23]
[24]              li      $v0,10            # 4 prepare to exit
[25]              syscall                   # 4 and exit
```

Pede-se:

- (1,0 ponto) Diga o que faz o programa acima (do ponto de vista semântico), detalhando seu funcionamento, e diga o conteúdo final de posições de memória alteradas, se for o caso. Comente semanticamente o texto do programa.

- (1,0 ponto) O programa contém alguma subrotina/função? O programa escreve algo na memória externa ao processador? Qual ou quais registradores possuem resultado relevante ao fim da execução?
- (1,0 ponto) Calcule qual o tamanho do programa, em bytes. Qual o tamanho da área de dados, em bytes?

Solução da Questão 3 (3,0 pontos)

- Este programa processa uma cadeia de caracteres descompactada (ou seja, cada caracter da cadeia ocupa o espaço de uma palavra na memória, 4 bytes, onde o byte menos significativo é o código ASCIIE do caracter e os três bytes mais significativos são sempre 0x00) armazenada em memória a partir do rótulo **cad**, contando quantos caracteres da cadeia correspondem a letras maiúsculas. O valor final calculado é armazenado na variável **caso**. Comentários no texto do programa acima.
- Não existe subrotina no programa, pois não existem instruções **jal x/jr $ra** no código fonte. Sim, o programa escreve um valor em memória, ao final de sua execução. Trata-se do valor do contador de caracteres maiúsculos, computado no registrador **$t4** do banco e escrito ao final na posição de memória **caso**. No final da execução, apenas $t4 possui realmente um valor relevante. $t1 possui o caracter final da cadeia (o código ASCCIIE do caractere de controle NULL), $t0 aponta para o último caractere da cadeia na memória e $t3 possui o endereço da posição de memória onde guardar o resultado da contagem.
- As linhas em branco ou com diretivas (texto começando com .xxx) ou apenas com rótulos (texto concluindo com xxx:) não geram nenhum código objeto. Sobram 18 linhas. As linhas [2] e [3] definem a área de dados, e correspondem a 11 palavras de memória, ou seja, **44** bytes, que é o tamanho da área de dados. Existem 3 linhas com pseudo-instruções, sendo duas com **la** e uma com **li**. As com **la** geram duas instruções (**lui/ori**), enquanto **li** gera apenas uma instrução (**addiu**). Assim, ocupam 20 bytes (4 bytes por instrução, como sempre). Restam 13 linhas com uma instrução do MIPS em cada uma delas, perfazendo 4*13=52 bytes. Logo á área de instruções ocupa 52+20=**72** bytes. O programa como um todo ocupa um total de 72+44=**116** bytes.

Fim da Solução da Questão 3 (3,0 pontos)