UNIVERSIDADE DO VALE DO ITAJAÍ

CIÊNCIA DA COMPUTAÇÃO

ARQUITETURA E ORGANIZAÇÃO DE PROCESSADORES

PROFESSOR: THIAGO FELSKI PEREIRA

AUTORES:

GUSTAVO BARON LAURITZEN

MATHEUS BARON LAURITZEN

GABRIEL BÓSIO

AVALIAÇÃO 01

PROGRAMAÇÃO EM LINGUAGEM DE MONTAGEM

ITAJAÍ

20/04/2023

Programa 01:

- Código-fonte feito em C++:

```
#include <iostream>
using namespace std;
int main()
{
   int Vetor_A[8], Vetor_B[8];
   cout << "||Incersão de vetores||\n";
   int i = 0;
   cout << "Insira os valores do Vetor_A...\n";
   do{
      cout << "Vetor_A[" << i << "]: ";
      cin >> Vetor_A[i];
      if (Vetor_A[i] <= 0 || Vetor_A[i] > 8)
         cout << "Valor inválido, insira novamente...\n";
      else
         i++;
   } while (i < 8);
   i = 0;
   cout << "Insira os valores do Vetor_B...\n";
   do {
         cout << "'Vetor_B[" << i << "]: ";
         cin >> Vetor_B[i];
         if (Vetor_B[i] <= 0 || Vetor_B[i] > 8)
            cout << "Valor inválido, insira novamente...\n";
         else
            i++;
      } while (i < 8);
   cout << "\n\n||Troca de vetores||\n";
   i = 0;
   do {
      int temp;
      temp = Vetor_A[i];
      Vetor_A[i] = Vetor_B[i];
      Vetor_B[i] = temp;
      i++;
   } while (i < 8);
   cout << "||Vetores após a troca...||\n";
   i = 0;
   do {
      cout << "Vetor_A[" << i << "]: " << Vetor_A[i] << endl;
      i++;
   } while (i < 8);
   cout << "\n\n";
```

```
    i = 0;
    do {
        cout << "Vetor_B[" << i << "]: " << Vetor_B[i] << endl;
        i++;
    } while (i < 8);
    return 0;
}
```

Explicação:

O código permite ao usuário inserir valores em dois vetores, Vetor_A e Vetor_B, com 8 elementos cada. Os valores são validados para garantir que estejam dentro de um intervalo específico (maior que zero e menor ou igual a 8). Em seguida, os elementos dos vetores são trocados entre si usando uma variável temporária. Por fim, os vetores resultantes são impressos na tela, um elemento por vez, utilizando loops "do-while".

O programa começa solicitando ao usuário que insira os valores para Vetor_A. Os valores são lidos e validados usando um loop "do-while" que é executado até que o usuário insira 8 valores válidos. Os valores inválidos são rejeitados com uma mensagem de erro. Em seguida, o programa solicita ao usuário que insira os valores para Vetor_B, seguindo o mesmo processo de validação.

Depois de obter os valores para Vetor_A e Vetor_B, o programa usa outro loop "do-while" para trocar os elementos dos vetores entre si usando uma variável temporária. Cada elemento de Vetor_A é trocado com o elemento correspondente de Vetor_B. Por fim, o programa usa loops "do-while" separados para imprimir os vetores Vetor_A e Vetor_B na tela (com os elementos trocados), um elemento por vez, utilizando a função "cout".

Ele é encerrado com a função main() retornando 0, indicando que foi executado com sucesso. O código utiliza loops "do-while" para validação de entrada, troca de elementos entre vetores e impressão dos resultados na tela.

- ## Código-fonte feito em linguagem de montagem do RISC-V:

```
# Disciplina: Arquitetura e Organização de Computadores
# Atividade: Avaliação 01 - Programação em Linguagem de Montagem
# Programa 01
# Grupo: - Matheus Baron Lauritzen
#       - Gustavo Baron Lauritzen
#       - Gabriel Bosio

.data
Vetor_A: .word 0,0,0,0,0,0,0,0
Vetor_B: .word 0,0,0,0,0,0,0,0

msg1: .asciz "||Preenchimento dos vetores||\n\n"
msg2: .asciz "\n||Trocando valores do mesmo índice||\n\n"
msg3: .asciz "\n||Vetores após a troca||\n\n"

en_A: .asciz "\nInsira os dados do Vetor_A:"
req_A: .asciz "\nVetor_A["

en_B: .asciz "\nInsira os dados do Vetor_B:"
req_B:.asciz "\nVetor_B["
```

```
21. fim_req: .asciz "]: "
22.
23. er: .asciz "\nValor errado, digite novamente..."
24.
25. quebra: .asciz "\n"
26.
27. .text
28.
29. #parametros do laço de contagem
30. addi t0, zero, 0
31. addi s0, zero, 8
32.
33. #inicialização dos vetores
34. la s1, Vetor_A
35. la s2, Vetor_B
36.
37. #mensagem de início de preenchimento
38. addi a7, zero, 4
39. la a0, msg1
40. ecall
41.
42. #enunciado do Vetor_A
43. addi a7, zero, 4
44. la a0, en_A
45. ecall
46.
47. for1:
48.
49. #impressão do requerimento do Vetor_A
50. addi a7, zero, 4
51. la a0, req_A
52. ecall
53.
54. addi a7, zero, 1
55. add a0, zero, t0
56. ecall
57.
58. addi a7, zero, 4
59. la a0, fim_req
60. ecall
61.
62. #cin Vetor_A
63. addi a7, zero, 5
64. ecall
65. add t1, zero, a0
66.
67. #mostrará mensagem de erro caso t1 <= 0 ou t1 > 8
68. addi s5, zero, 0
69. ble t1, s5, erro1
70. bgt t1, s0, erro1
71.
```

```
#armazenamento no Vetor_A
slli t2, t0, 2
add s3, s1, t2
sw t1, 0(s3)

#cont++
addi t0, t0, 1

#sair caso chegue ao fim do laço ou repetir caso não
bge t0, s0, fim_1
jal zero, for1

erro1:

#mensagem de erro e repetição do último cin
addi a7, zero, 4
la a0, er
ecall
jal zero, for1

fim_1:

#enunciado do Vetor_B
addi a7, zero, 4
la a0, en_B
ecall

#zerando novamente o contador
addi t0, zero, 0

for2:

#impressão do requerimento do Vetor_B
addi a7, zero, 4
la a0, req_B
ecall

addi a7, zero, 1
add a0, zero, t0
ecall

addi a7, zero, 4
la a0, fim_req
ecall

#cin Vetor_B
addi a7, zero, 5
ecall
add t1, zero, a0

#mostrará mensagem de erro caso t1 <= 0 ou t1 > 8
ble t1, s5, erro2
bgt t1, s0, erro2
```

```
125.
126.#armazenamento no Vetor_B
127.slli t2, t0, 2
128.add s3, s2, t2
129.sw t1, 0(s3)
130.
131.#cont++
132.addi t0, t0, 1
133.
134.#sair caso chegue ao fim do laço ou repetir caso não
135.bge t0, s0, fim_2
136.jal zero, for2
137.
138.erro2:
139.
140.#mensagem de erro e repetição do último cin
141.addi a7, zero, 4
142.la a0, er
143.ecall
144.jal zero, for2
145.
146.fim_2:
147.
148.#mensagem de início de troca
149.addi a7, zero, 4
150.la a0, msg2
151.ecall
152.
153.#zerando contador
154.addi t0, zero, 0
155.
156.for_troca:
157.
158.#pegando o endereço do índice
159.slli t2, t0, 2
160.add s3, s1, t2
161.add s4, s2, t2
162.
163.#guardando o dado do Vetor_A num registrador temporário
164.lw a0, 0(s3)
165.add t3, zero, a0
166.
167.#passa o valor do Vetor_B para o Vetor_A
168.lw a0, 0(s4)
169.add t4, zero, a0
170.sw t4, 0(s3)
171.
172.#passa valor no registrador temp e passa para o Vetor_B
173.sw t3, 0(s4)
174.
175.addi t0, t0, 1
176.
177.#repete até que termine a contagem
```

```
bge t0, s0, troca_fim
jal zero, for_troca

troca_fim:

#mensagem de início de impressão
addi a7, zero, 4
la a0, msg3
ecall

addi t0, zero, 0

for_imp1:

#impressão da mensagem de amostragem
addi a7, zero, 4
la a0, req_A
ecall

addi a7, zero, 1
add a0, zero, t0
ecall

addi a7, zero, 4
la a0, fim_req
ecall

#encontrando o índice
slli t2, t0, 2
add s3, s1, t2
lw a0, 0(s3)

#imprimindo o valor
addi a7, zero, 1
ecall

#repetição do laço até o fim da contagem
addi t0, t0, 1
bge t0, s0, for_imp1_fim
jal zero, for_imp1

for_imp1_fim:

addi a7, zero, 4
la a0, quebra
ecall

addi t0, zero, 0

for_imp2:

#impressão da mensagem de amostragem
addi a7, zero, 4
```

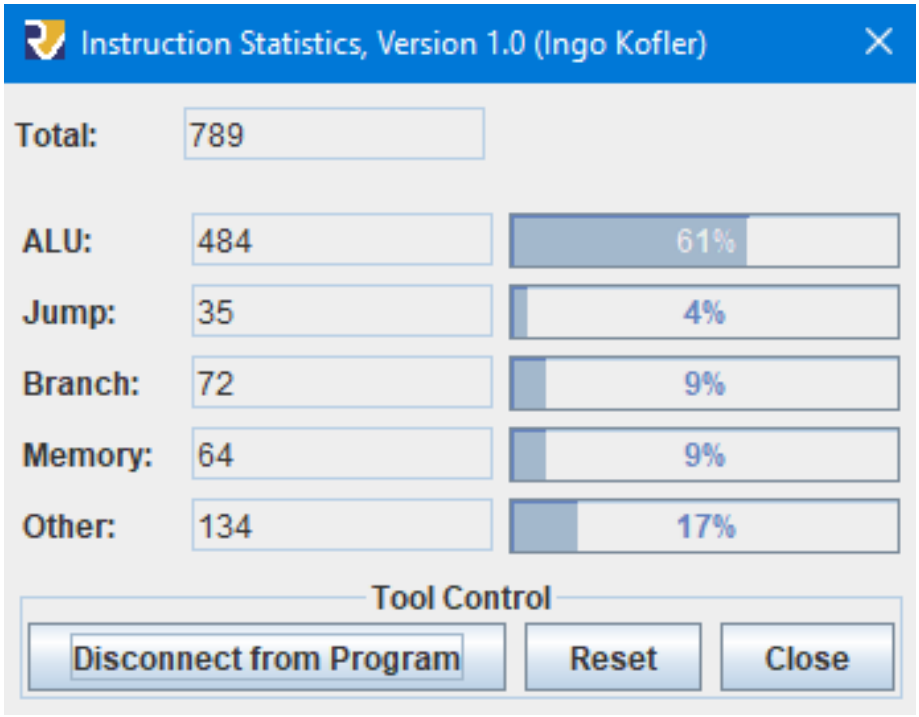
```
231.la a0, req_B
232.ecall
233.
234.addi a7, zero, 1
235.add a0, zero, t0
236.ecall
237.
238.addi a7, zero, 4
239.la a0, fim_req
240.ecall
241.
242.#encontrando o índice
243.slli t2, t0, 2
244.add s3, s2, t2
245.lw a0, 0(s3)
246.
247.#imprimindo o valor
248.addi a7, zero, 1
249.ecall
250.
251.#repetição do laço até o fim da contagem
252.addi t0, t0, 1
253.bge t0, s0, for_imp2_fim
254.jal zero, for_imp2
255.
256.for_imp2_fim:
```

Explicação:

Este código em assembly que realiza operações em dois vetores, Vetor_A e Vetor_B. O programa começa declarando esses vetores, cada um com 8 elementos, e também mensagens para serem impressas no console, como uma mensagem de início, uma mensagem para preencher Vetor_A, outra para preencher Vetor_B, entre outras. Em seguida, o programa inicializa dois registradores (s1 e s2) com os endereços dos vetores. Depois, o programa entra em um laço (for1) que é responsável por preencher o Vetor_A com os dados inseridos pelo usuário. Dentro desse laço, há uma sequência de instruções para imprimir no console um requerimento do elemento atual, ler a entrada do usuário, verificar se a entrada está dentro dos limites permitidos (1 a 8) e armazenar a entrada no Vetor_A. Em caso de erro, a mensagem de erro é impressa e o laço é repetido para o mesmo elemento.

Após o preenchimento de Vetor_A, o programa entra em um segundo laço (for2) que preenche o Vetor_B de maneira semelhante ao Vetor_A. Em seguida, o programa inicia outro laço (for_troca) que é responsável por trocar os valores dos elementos de mesmo índice em Vetor_A e Vetor_B. Para cada elemento, o programa armazena temporariamente o valor de Vetor_A em um registrador, copia o valor de Vetor_B para Vetor_A e, por fim, copia o valor armazenado anteriormente (de Vetor_A) para Vetor_B. Ao final, o programa imprime uma mensagem indicando que a troca foi realizada e termina. Em resumo, o programa realiza a troca de valores em dois vetores e é um exemplo simples do uso de laços, condicionais e manipulação de memória em um programa assembly.

- Quadro de análise das instruções:



- A execução correta das entradas e saídas realizadas via console do simulador:

Entrada:

```
||Preenchimento dos vetores||


Insira os dados do Vetor_A:
Vetor_A[0]: 1

Vetor_A[1]: 2

Vetor_A[2]: 3

Vetor_A[3]: 4

Vetor_A[4]: 5

Vetor_A[5]: 6

Vetor_A[6]: 7

Vetor_A[7]: 8
```

```
Insira os dados do Vetor_B:
Vetor_B[0]: 8

Vetor_B[1]: 7

Vetor_B[2]: 6

Vetor_B[3]: 5

Vetor_B[4]: 4

Vetor_B[5]: 3

Vetor_B[6]: 2

Vetor_B[7]: 1
```

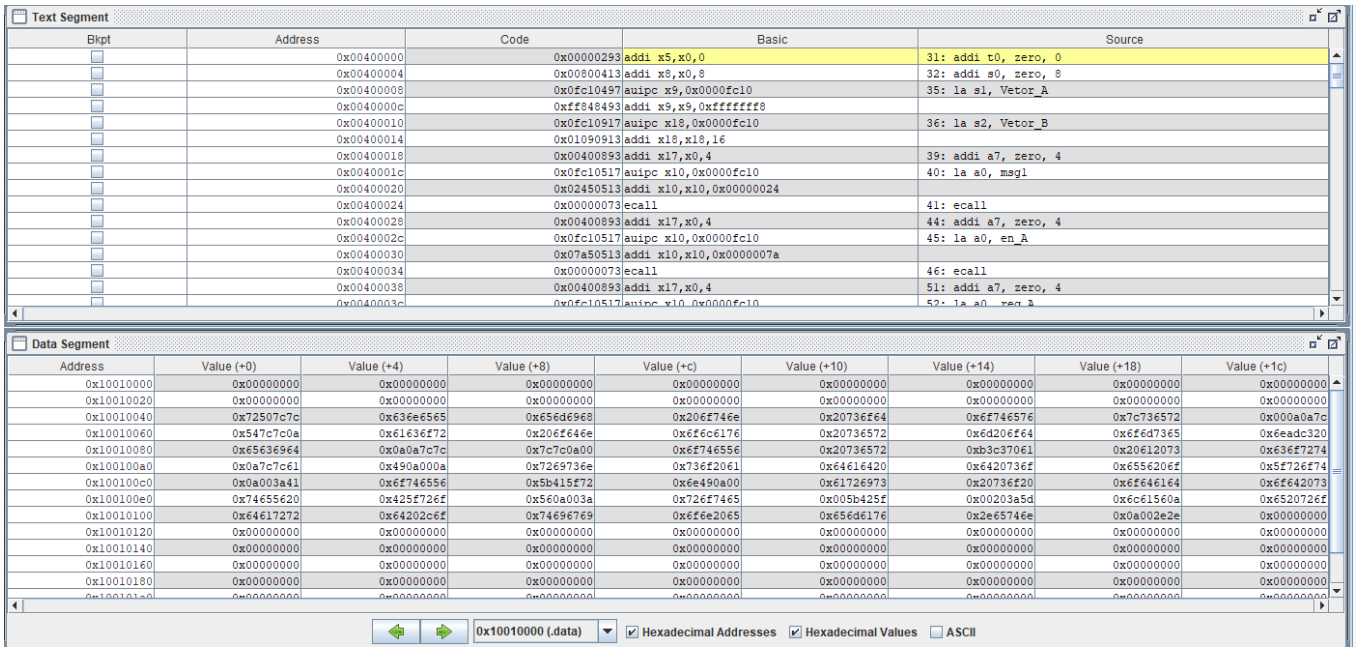
Saída:

```
||Trocando valores do mesmo índice||


||Vetores após a troca||


Vetor_A[0]: 1
Vetor_A[1]: 2
Vetor_A[2]: 3
Vetor_A[3]: 4
Vetor_A[4]: 5
Vetor_A[5]: 6
Vetor_A[6]: 7
Vetor_A[7]: 8
```

```
Vetor_B[0]: 1
Vetor_B[1]: 2
Vetor_B[2]: 3
Vetor_B[3]: 4
Vetor_B[4]: 5
Vetor_B[5]: 6
Vetor_B[6]: 7
Vetor_B[7]: 8
-- program is finished running (dropped off bottom) --
```

- Resultados da execução dos programas:

Antes da execução:



Text Segment

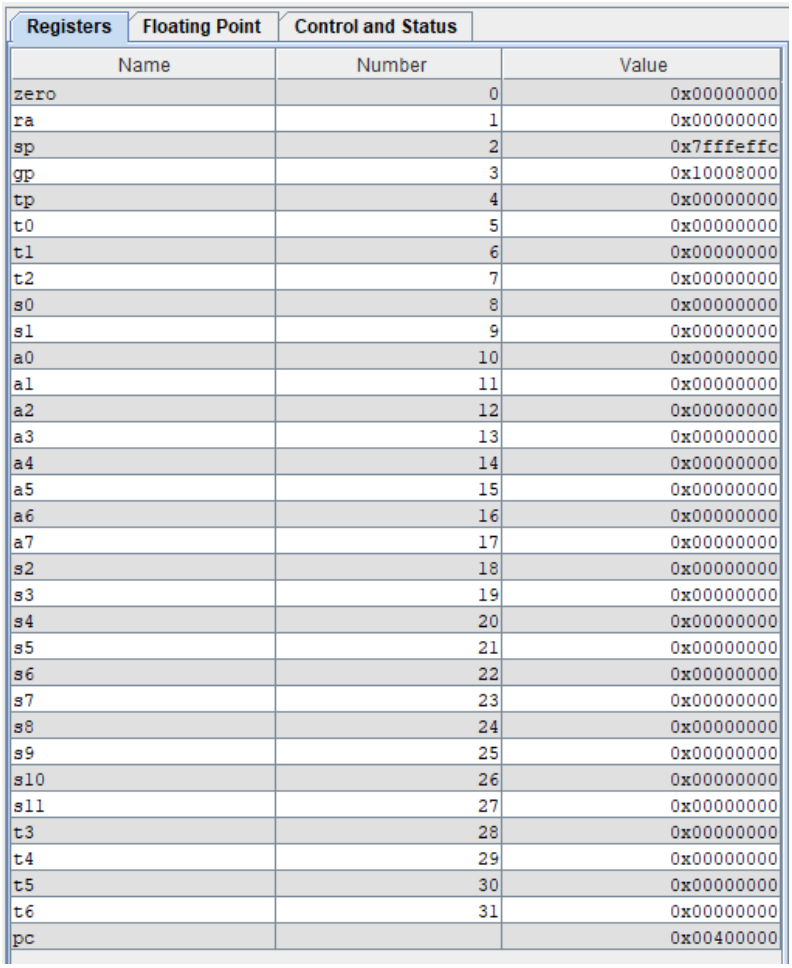
Bkpt	Address	Code	Basic	Source
	0x00400000	0x00000293	addi x5,x0,0	31: addi t0, zero, 0
	0x00400004	0x00800413	addi x8,x0,8	32: addi s0, zero, 8
	0x00400008	0x0fc10497	auipc x9,0x0000fc10	35: la s1, Vetor_A
	0x0040000c	0xff848493	addi x9,x9,0xfffffff8	
	0x00400010	0x0fc10917	auipc x18,0x0000fc10	36: la s2, Vetor_B
	0x00400014	0x01090913	addi x18,x18,16	
	0x00400018	0x00400893	addi x17,x0,4	39: addi a7, zero, 4
	0x0040001c	0x0fc10517	auipc x10,0x0000fc10	40: la a0, msg1
	0x00400020	0x02450513	addi x10,x10,0x00000024	
	0x00400024	0x00000073	ecall	41: ecall
	0x00400028	0x00400893	addi x17,x0,4	44: addi a7, zero, 4
	0x0040002c	0x0fc10517	auipc x10,0x0000fc10	45: la a0, en_A
	0x00400030	0x07a50513	addi x10,x10,0x0000007a	
	0x00400034	0x00000073	ecall	46: ecall
	0x00400038	0x00400893	addi x17,x0,4	51: addi a7, zero, 4
	0x0040003c	0x0fc10517	auipc x10,0x0000fc10	52: la a0, req_A

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x72507c7c	0x636e6565	0x656d6968	0x206f746e	0x20736f64	0x6f746576	0x7c736572	0x000a0a7c
0x10010060	0x547c7c0a	0x61636f72	0x206f646e	0x6f6c6176	0x20736572	0x6d206f64	0x6f6d7365	0x6eadc320
0x10010080	0x65636964	0x0a0a7c7c	0x7c7c0a00	0x6f746556	0x20736572	0xb3c37061	0x20612073	0x636f7274
0x100100a0	0x0a7c7c61	0x490a000a	0x7269736e	0x736f2061	0x64616420	0x6420736f	0x6556206f	0x5f726f74
0x100100c0	0x0a003a41	0x6f746556	0x5b415f72	0x6e490a00	0x61726973	0x20736f20	0x6f646164	0x6f642073
0x100100e0	0x74655620	0x425f726f	0x560a003a	0x726f7465	0x005b425f	0x00203a5d	0x6c61560a	0x6520726f
0x10010100	0x64617272	0x64202c6f	0x74696769	0x6f6e2065	0x656d6176	0x2e65746e	0x0a002e2e	0x00000000
0x10010120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

0x10010000 (.data) ☑ Hexadecimal Addresses ☑ Hexadecimal Values ☐ ASCII

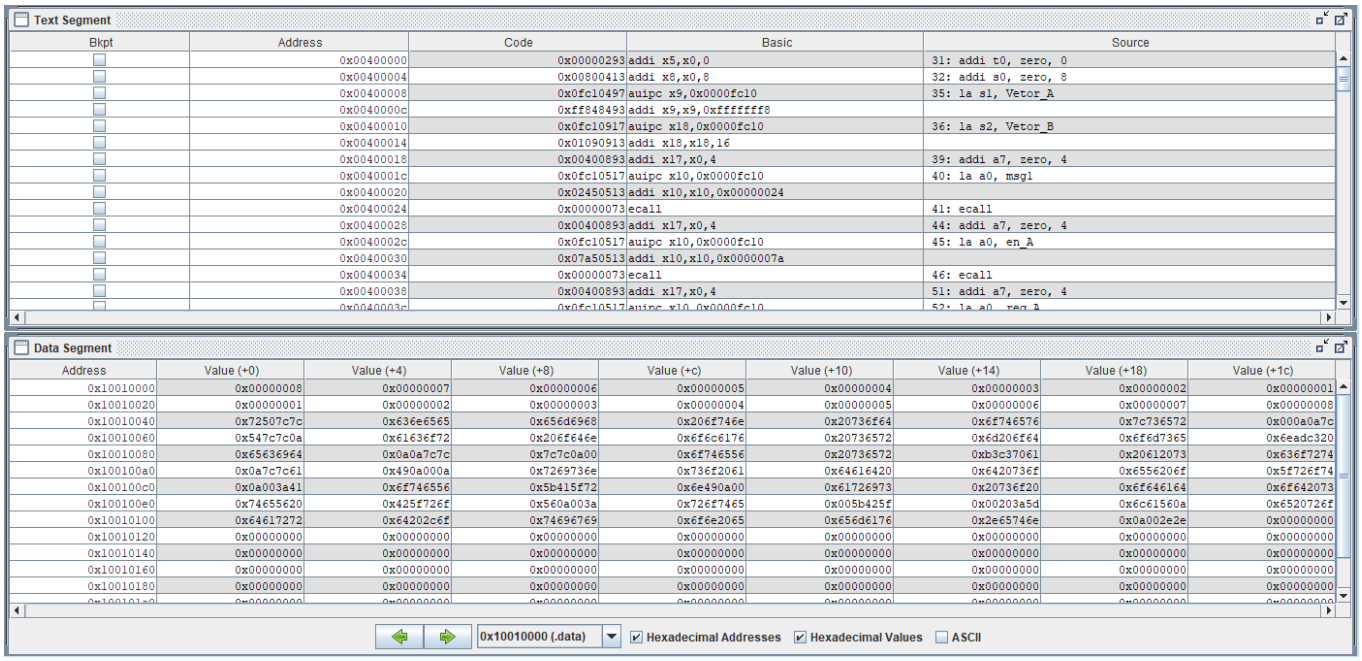
OBS: Se ficar ruim a interpretação das capturas de tela, por favor aproxime o zoom



Registers | Floating Point | Control and Status

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7fffeffc
gp	3	0x10008000
tp	4	0x00000000
t0	5	0x00000000
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x00000000
a0	10	0x00000000
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0x00000000
s3	19	0x00000000
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000
s11	27	0x00000000
t3	28	0x00000000
t4	29	0x00000000
t5	30	0x00000000
t6	31	0x00000000
pc		0x00400000

Depois da execução:



Text Segment

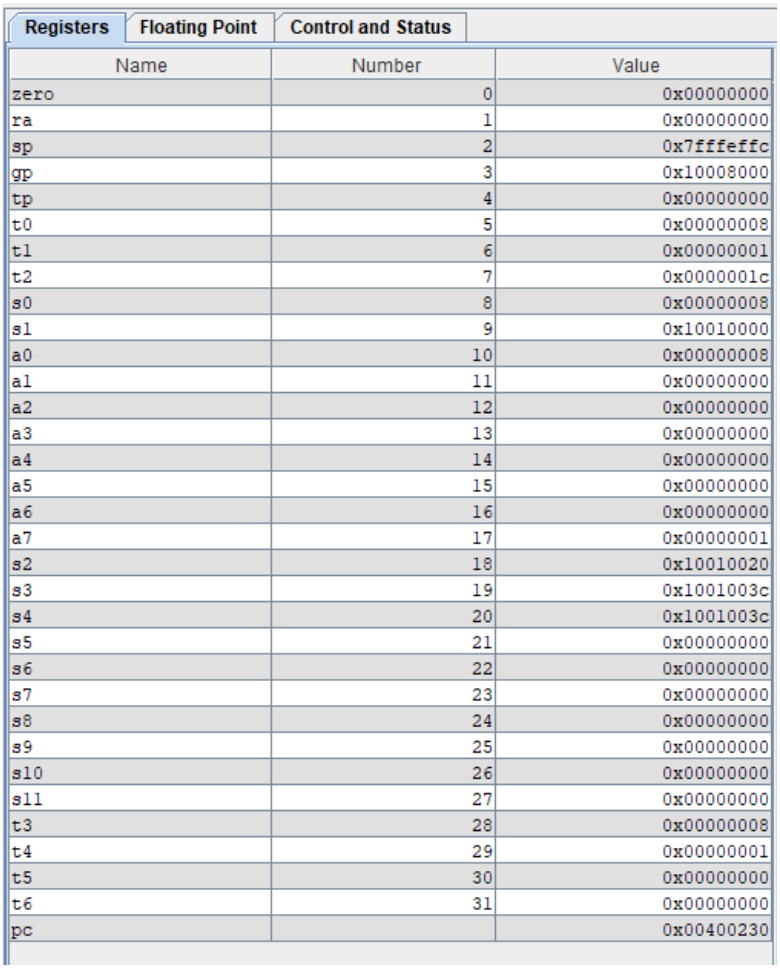
Bkpt	Address	Code	Basic	Source
	0x00400000	0x00000293	addi x5,x0,0	31: addi t0, zero, 0
	0x00400004	0x00800413	addi x8,x0,8	32: addi s0, zero, 8
	0x00400008	0x0fc10497	auipc x9,0x0000fc10	35: la s1, Vetor_A
	0x0040000c	0xff848493	addi x9,x9,0xffffff8	
	0x00400010	0x0fc10917	auipc x18,0x0000fc10	36: la s2, Vetor_B
	0x00400014	0x01090913	addi x18,x18,16	
	0x00400018	0x00400893	addi x17,x0,4	39: addi a7, zero, 4
	0x0040001c	0x0fc10517	auipc x10,0x0000fc10	40: la a0, msg1
	0x00400020	0x02450513	addi x10,x10,0x00000024	
	0x00400024	0x00000073	ecall	41: ecall
	0x00400028	0x00400893	addi x17,x0,4	44: addi a7, zero, 4
	0x0040002c	0x0fc10517	auipc x10,0x0000fc10	45: la a0, en_A
	0x00400030	0x07a50513	addi x10,x10,0x0000007a	
	0x00400034	0x00000073	ecall	46: ecall
	0x00400038	0x00400893	addi x17,x0,4	51: addi a7, zero, 4
	0x0040003c	0x0fc10517	auipc x10,0x0000fc10	52: la a0, reg_A

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000008	0x00000007	0x00000006	0x00000005	0x00000004	0x00000003	0x00000002	0x00000001
0x10010020	0x00000001	0x00000002	0x00000003	0x00000004	0x00000005	0x00000006	0x00000007	0x00000008
0x10010040	0x72507c7c	0x636e6565	0x656d6968	0x206f746e	0x20736f64	0x6f746576	0x7c736572	0x000a0a7c
0x10010060	0x547c7c0a	0x61636f72	0x206f646e	0x6f6c6176	0x20736572	0x6d206f64	0x6f6d7365	0x6eadc320
0x10010080	0x65636964	0x0a0a7c7c	0x7c7c0a00	0x6f746556	0x20736572	0xb3c37061	0x20612073	0x636f7274
0x100100a0	0x0a7c7c61	0x490a000a	0x7269736e	0x736f2061	0x64616420	0x6420736f	0x6556206f	0x5f726f74
0x100100c0	0x0a003a41	0x6f746556	0x5b415f72	0x6e490a00	0x61726973	0x20736f20	0x6f646164	0x6f642073
0x100100e0	0x74655620	0x425f726f	0x560a003a	0x726f7465	0x005b425f	0x00203a5d	0x6c61560a	0x6520726f
0x10010100	0x64617272	0x64202c6f	0x74696769	0x6f6e2065	0x656d6176	0x2e65746e	0x0a002e2e	0x00000000
0x10010120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

0x10010000 (.data) ☑ Hexadecimal Addresses ☑ Hexadecimal Values ☐ ASCII

OBS: Se ficar ruim a interpretação das capturas de tela, por favor aproxime o zoom



Registers | Floating Point | Control and Status

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7fffeffc
gp	3	0x10008000
tp	4	0x00000000
t0	5	0x00000008
t1	6	0x00000001
t2	7	0x0000001c
s0	8	0x00000008
s1	9	0x10010000
a0	10	0x00000008
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000001
s2	18	0x10010020
s3	19	0x1001003c
s4	20	0x1001003c
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000
s11	27	0x00000000
t3	28	0x00000008
t4	29	0x00000001
t5	30	0x00000000
t6	31	0x00000000
pc		0x00400230

Programa 02:

- Código-fonte feito em C++:

```
#include <iostream>
using namespace std;
int main() {
    unsigned int presenca[16] = {0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF,
                                 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF,
                                 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF,
                                 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF};
    int aula, aluno, registro;
    while (true) {
      // Leitura dos dados
      do {
        cout << "Entre com o número da aula (de 0 a 15): ";
        cin >> aula;
      } while (aula < 0 || aula > 15);
      do {
        cout << "Entre com o número do aluno (de 0 a 31): ";
        cin >> aluno;
      } while (aluno < 0 || aluno > 31);
      do {
        cout << "Entre com o tipo do registro (presença = 1; ausência = 0): ";
        cin >> registro;
      } while (registro < 0 || registro > 1);
      // Construção da máscara
      unsigned int mask = 1 << aluno;
      mask = ~(mask);
      // Registro da presença ou ausência
      if (registro == 1) {
        presenca[aula] |= (1 << aluno);
      }
      else if (registro == 0) {
        presenca[aula] &= mask;
      }
      // Impressão da máscara
      cout << "Máscara de bits para aula " << aula << ":" << endl;
      for (int i = 31; i >= 0; i--) {
        unsigned int bit = (presenca[aula] >> i) & 1;
        cout << bit;
      }
      cout << endl << endl;
    }
    return 0;
}
```

Explicação:

O código é um programa em C++ que permite registrar a presença ou ausência de alunos em determinadas aulas usando máscaras de bits. A linguagem C++ oferece suporte para trabalhar com

máscaras de bits, que são valores numéricos compostos de uma série de bits que representam opções, permissões ou estados. As operações de máscara de bits envolvem o uso de operadores bitwise, como &, |, ^ e ~, que realizam operações lógicas em nível de bit. Esses operadores permitem manipular bits individuais de um número e combinar múltiplos bits em um único valor.

O programa utiliza um array de 16 elementos chamado presenca para representar as presenças ou ausências dos alunos em cada uma das aulas. Cada elemento do array é um valor numérico de 32 bits, que pode ser interpretado como uma sequência de 32 bits, em que cada bit representa a presença ou ausência de um aluno. Cada bit é identificado por um número de 0 a 31, que representa a posição do bit no valor numérico.

O programa inicia com um loop while (true) que permite que o usuário execute o programa continuamente, permitindo que o usuário realize quantos registros desejar. Dentro do loop, o programa solicita que o usuário entre com o número da aula, o número do aluno e o tipo do registro (presença ou ausência). O programa valida os valores de entrada e armazena as informações em variáveis locais. Em seguida, o programa constrói uma máscara de bits a partir do número do aluno e aplica a máscara ao valor do array presenca.

As operações de máscara de bits são utilizadas para modificar os valores do array presenca. O programa constrói uma máscara de bits com o valor 1 deslocado para a posição do bit correspondente ao número do aluno. Em seguida, o programa aplica a máscara ao valor do array presenca. Se o registro for de presença, o programa utiliza o operador bitwise | para combinar a máscara com o valor do array. Se o registro for de ausência, o programa utiliza o operador bitwise & para aplicar a máscara invertida ao valor do array. Finalmente, o programa imprime a máscara de bits atualizada para a aula correspondente.

- Código-fonte feito em linguagem de montagem do RISC-V:

```
# Disciplina: Arquitetura e Organização de Computadores
# Atividade: Avaliação 01 – Programação em Linguagem de Montagem
# Programa 02
# Grupo: - Matheus Baron Lauritzen
#        - Gustavo Baron Lauritzen
#        - Gabriel Bosio

.data
mensagemAula: .asciz "Entre com o número da aula (de 0 a 15):"
mensagemAluno: .asciz "Entre com o número do aluno (de 0 a 31):"
mensagemRegistro: .asciz "Entre com o tipo do registro (presença = 1; ausência = 0):"

# Define o vetor com 16 elementos inicializados com todos os bits em 1
vetor: .word 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF,
       0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF,
       0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF,
              0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF

.text
la s1, vetor #endereço base do vetor com 16 elementos 0xFFFFFFFF

xori t6, zero, -1 # registrador com tudo 1

loop1:
    # Exibe mensagem solicitando o número da aula
```

```
    addi a7, zero, 4
    la a0, mensagemAula
    ecall

    # Lê o número da aula
    addi a7, zero, 5
    ecall
    add s2, zero, a0

    # Exibe mensagem solicitando o número do aluno
    addi a7, zero, 4
    la a0, mensagemAluno
    ecall

    # Lê o número do aluno
    addi a7, zero, 5
    ecall
    add s3, zero, a0

    # Exibe mensagem solicitando o tipo de registro
    addi a7, zero, 4
    la a0, mensagemRegistro
    ecall
    # Lê o tipo de registro
    addi a7, zero, 5
    ecall
    add s4, zero, a0

#Percorre o array ate chegar na posicao(aula) que o user mandou
slli t1, s2, 2
add s7, s1, t1
#pega a mascara original e coloca no t2
lw t2, 0(s7)

    #Desloca o valor t3=1 em s3=aluno posicoes
    addi t3, zero, 1
    sll t4, t3, s3

    DesligaBit:
        bne s4, zero, LigaBit
        #invertendo todos os bits da máscara e aplicando no t5
        xor t5, t4, t6
        #desliga bit
        and t2, t2, t5
        jal zero, fimOperacaoBit
    LigaBit:
        #liga bit
        or t2, t2, t4
```

```
74.         jal zero, fimOperacaoBit
75.
76.     fimOperacaoBit:
77.     #Armazena a mascara original modificada na posicao do vetor
78.     sw t2, 0(s7)
79.
80.     # Volta para o início do loop
81.     jal zero, loop1
```

Explicação:

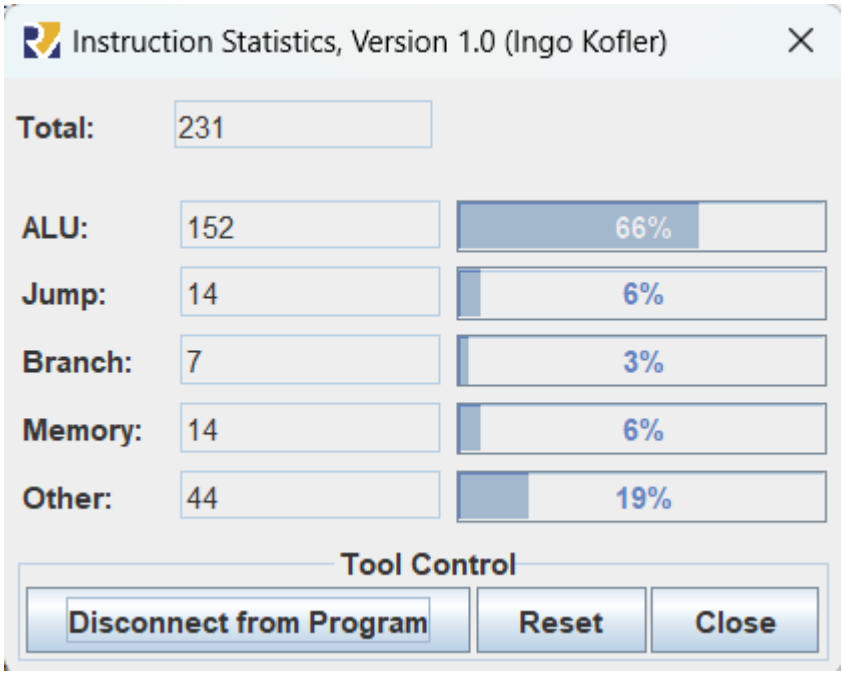
O código recebe como entrada o número de uma aula (0 a 15), o número de um aluno (0 a 31) e o tipo de registro (presença = 1; ausência = 0) e modifica um vetor de 16 elementos inicializados com todos os bits em 1. Cada elemento do vetor representa uma aula e cada bit representa a presença ou ausência de um aluno naquela aula. No início do código é definido três mensagens a serem exibidas na tela: mensagemAula, mensagemAluno e mensagemRegistro. Em seguida, é definido o vetor de 16 elementos com valor inicial de 0xFFFFFFFF.

No início do loop principal (loop1), o endereço base do vetor é carregado no registrador s1 e o valor -1 (tudo 1) é carregado no registrador t6. As mensagens são exibidas na tela e os valores de entrada são lidos e armazenados nos registradores s2 (número da aula), s3 (número do aluno) e s4 (tipo de registro). Em seguida, o código calcula o endereço da posição do vetor correspondente à aula informada pelo usuário e carrega a máscara original (todos os bits 1) para o registrador t2. A variável t3 é definida como 1 e, em seguida, é deslocada para a esquerda (sll) s3 posições, criando uma máscara com um único bit 1 na posição correspondente ao aluno informado pelo usuário. O código então entra em uma condição que verifica o tipo de registro (s4) e executa uma das duas operações a seguir:

- LigaBit: Se s4 for diferente de zero (indicando que o registro é de presença), a operação or é usada para ligar o bit correspondente ao aluno na máscara original t2.
- DesligaBit: Se s4 for igual a zero (indicando que o registro é de ausência), a operação xor é usada para inverter todos os bits da máscara t4 (com um único bit 1 na posição correspondente ao aluno) e a operação and é usada para desligar o bit correspondente na máscara original t2.

No final do loop, a máscara modificada é armazenada na posição correspondente do vetor. O código então volta ao início do loop principal, onde a próxima entrada do usuário é lida e o vetor é atualizado novamente.

- Quadro de análise das instruções:



- A execução correta das entradas e saídas realizadas via console do simulador:

Entradas:

```
Entre com o número da aula (de 0 a 15):4
Entre com o número do aluno (de 0 a 31):2
Entre com o tipo do registro (presença = 1; ausência = 0):0
Entre com o número da aula (de 0 a 15):4
Entre com o número do aluno (de 0 a 31):2
Entre com o tipo do registro (presença = 1; ausência = 0):0
```

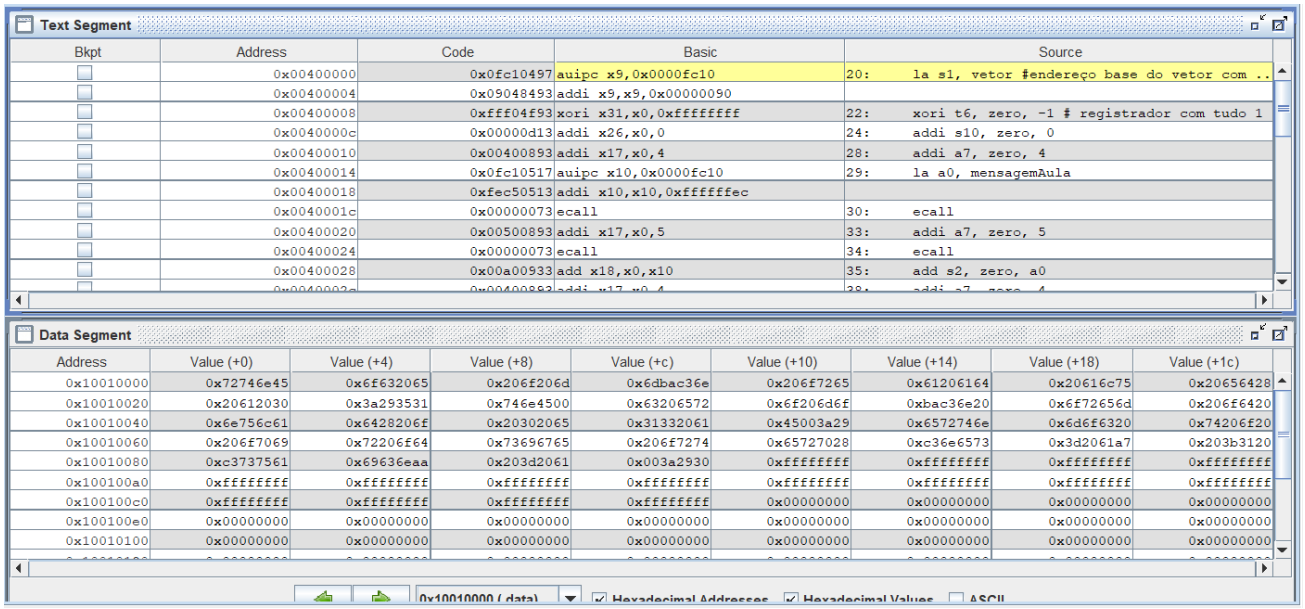
```
Entre com o número da aula (de 0 a 15):10
Entre com o número do aluno (de 0 a 31):23
Entre com o tipo do registro (presença = 1; ausência = 0):1
```

```
Entre com o número da aula (de 0 a 15):2
Entre com o número do aluno (de 0 a 31):15
Entre com o tipo do registro (presença = 1; ausência = 0):0
Entre com o número da aula (de 0 a 15):2
Entre com o número do aluno (de 0 a 31):15
Entre com o tipo do registro (presença = 1; ausência = 0):1
```

```
Entre com o número da aula (de 0 a 15):14
Entre com o número do aluno (de 0 a 31):30
Entre com o tipo do registro (presença = 1; ausência = 0):1
Entre com o número da aula (de 0 a 15):14
Entre com o número do aluno (de 0 a 31):30
Entre com o tipo do registro (presença = 1; ausência = 0):0
Entre com o número da aula (de 0 a 15):
```

- Resultados da execução dos programas:

Antes da execução:



Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x0fc10497	auipc x9,0x0000fc10	20: la s1, vetor #endereço base do vetor com ..
	0x00400004	0x09048493	addi x9,x9,0x00000090	
	0x00400008	0xfff04f93	xori x31,x0,0xffffffff	22: xori t6, zero, -1 # registrador com tudo 1
	0x0040000c	0x00000d13	addi x26,x0,0	24: addi s10, zero, 0
	0x00400010	0x00400893	addi x17,x0,4	28: addi a7, zero, 4
	0x00400014	0x0fc10517	auipc x10,0x0000fc10	29: la a0, mensagemAula
	0x00400018	0xfec50513	addi x10,x10,0xfffffec	
	0x0040001c	0x00000073	ecall	30: ecall
	0x00400020	0x00500893	addi x17,x0,5	33: addi a7, zero, 5
	0x00400024	0x00000073	ecall	34: ecall
	0x00400028	0x00a00933	add x18,x0,x10	35: add s2, zero, a0

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x72746e45	0x6f632065	0x206f206d	0x6dbac36e	0x206f7265	0x61206164	0x20616c75	0x20656428
0x10010020	0x20612030	0x3a293531	0x746e4500	0x63206572	0x6f206d6f	0xbac36e20	0x6f72656d	0x206f6420
0x10010040	0x6e756c61	0x6428206f	0x20302065	0x31332061	0x45003a29	0x6572746e	0x6d6f6320	0x74206f20
0x10010060	0x206f7069	0x72206f64	0x73696765	0x206f7274	0x65727028	0xc36e6573	0x3d2061a7	0x203b3120
0x10010080	0xc3737561	0x69636eaa	0x203d2061	0x003a2930	0xffffffff	0xffffffff	0xffffffff	0xffffffff
0x100100a0	0xffffffff	0xffffffff	0xffffffff	0xffffffff	0xffffffff	0xffffffff	0xffffffff	0xffffffff
0x100100c0	0xffffffff	0xffffffff	0xffffffff	0xffffffff	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

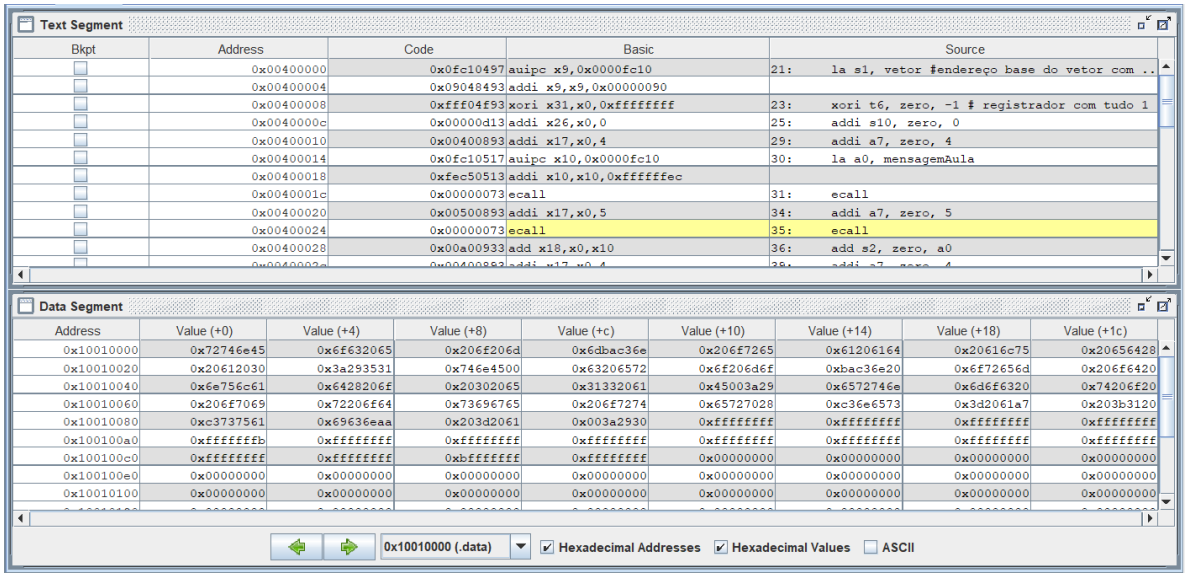
0x10010000 (.data) Hexadecimal Addresses Hexadecimal Values ASCII

OBS: Se ficar ruim a interpretação das capturas de tela, por favor aproxime o zoom

Registers | Floating Point | Control and Status

Name	Number	Value
zero	0	0x0000000000000000
ra	1	0x0000000000000000
sp	2	0x000000007fffeffc
gp	3	0x0000000010008000
tp	4	0x0000000000000000
t0	5	0x0000000000000000
t1	6	0x0000000000000000
t2	7	0x0000000000000000
s0	8	0x0000000000000000
s1	9	0x0000000000000000
a0	10	0x0000000000000000
a1	11	0x0000000000000000
a2	12	0x0000000000000000
a3	13	0x0000000000000000
a4	14	0x0000000000000000
a5	15	0x0000000000000000
a6	16	0x0000000000000000
a7	17	0x0000000000000000
s2	18	0x0000000000000000
s3	19	0x0000000000000000
s4	20	0x0000000000000000
s5	21	0x0000000000000000
s6	22	0x0000000000000000
s7	23	0x0000000000000000
s8	24	0x0000000000000000
s9	25	0x0000000000000000
s10	26	0x0000000000000000
s11	27	0x0000000000000000
t3	28	0x0000000000000000
t4	29	0x0000000000000000
t5	30	0x0000000000000000
t6	31	0x0000000000000000
pc		0x0000000000400000

Depois da execução:



Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x0fc10497	auipc x9,0x0000fc10	21: la s1, vetor #endereço base do vetor com ..
	0x00400004	0x09048493	addi x9,x9,0x00000090	
	0x00400008	0xfff04f93	xori x31,x0,0xffffffff	23: xori t6, zero, -1 # registrador com tudo 1
	0x0040000c	0x00000d13	addi x26,x0,0	25: addi s10, zero, 0
	0x00400010	0x00400893	addi x17,x0,4	29: addi a7, zero, 4
	0x00400014	0x0fc10517	auipc x10,0x0000fc10	30: la a0, mensagemAula
	0x00400018	0xfec50513	addi x10,x10,0xffffffec	
	0x0040001c	0x00000073	ecall	31: ecall
	0x00400020	0x00500893	addi x17,x0,5	34: addi a7, zero, 5
	0x00400024	0x00000073	ecall	35: ecall
	0x00400028	0x00a00933	add x18,x0,x10	36: add s2, zero, a0

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x72746e45	0x6f632065	0x206f206d	0x6dbac36e	0x206f7265	0x61206164	0x20616c75	0x20656428
0x10010020	0x20612030	0x3a293531	0x746e4500	0x63206572	0x6f206d6f	0xbac36e20	0x6f72656d	0x206f6420
0x10010040	0x6e756c61	0x6428206f	0x20302065	0x31332061	0x45003a29	0x6572746e	0x6d6f6320	0x74206f20
0x10010060	0x206f7069	0x72206f64	0x73696765	0x206f7274	0x65727028	0xc36e6573	0x3d2061a7	0x203b3120
0x10010080	0xc3737561	0x69636eaa	0x203d2061	0x003a2930	0xffffffff	0xffffffff	0xffffffff	0xffffffff
0x100100a0	0xfffffffb	0xffffffff	0xffffffff	0xffffffff	0xffffffff	0xffffffff	0xffffffff	0xffffffff
0x100100c0	0xffffffff	0xffffffff	0xbfffffff	0xffffffff	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

0x10010000 (.data) ☑ Hexadecimal Addresses ☑ Hexadecimal Values ☐ ASCII

OBS: Se ficar ruim a interpretação das capturas de tela, por favor aproxime o zoom

Registers | Floating Point | Control and Status

Name	Number	Value
zero	0	0x0000000000000000
ra	1	0x0000000000000000
sp	2	0x000000007fffeffc
gp	3	0x0000000010008000
tp	4	0x0000000000000000
t0	5	0x0000000000000000
t1	6	0x0000000000000038
t2	7	0xffffffffbfffffff
s0	8	0x0000000000000000
s1	9	0x0000000010010090
a0	10	0x0000000010010000
a1	11	0x0000000000000000
a2	12	0x0000000000000000
a3	13	0x0000000000000000
a4	14	0x0000000000000000
a5	15	0x0000000000000000
a6	16	0x0000000000000000
a7	17	0x0000000000000005
s2	18	0x000000000000000e
s3	19	0x000000000000001e
s4	20	0x0000000000000000
s5	21	0x0000000000000000
s6	22	0x0000000000000000
s7	23	0x00000000100100c8
s8	24	0x0000000000000000
s9	25	0x0000000000000000
s10	26	0x0000000000000000
s11	27	0x0000000000000000
t3	28	0x0000000000000001
t4	29	0x0000000040000000
t5	30	0xffffffffbfffffff
t6	31	0xffffffffffffffff
pc		0x0000000000400028