# Experimentos com Simulador de Caches

Yago Martins Escada

Orientadora: Nahri Balesdent Moreano

Faculdade de Computação (FACOM)
Universidade Federal de Mato Grosso do Sul (UFMS)

Este texto ilustra a utilização de um simulador de caches de uso educacional, o ECS (*Educational Cache Simulator*) [1] para a realização de experimentos com programas RISC-V e *traces* de endereços de memória, exemplificando como os estudantes podem estudar os conceitos de hierarquia de memórias através das simulações realizadas.

## 1. Experimento de Simulação de Programas RISC-V

Como o principal objetivo do ECS é o uso educacional, nesta seção é apresentado um experimento que utiliza a ferramenta durante a execução de programas RISC-V, visando proporcionar ao estudante uma compreensão mais aprofundada sobre os conceitos de organização e funcionamento da memória cache.

Os programas RISC-V utilizados nas simulações são listados abaixo e o código fonte deles encontra-se nos Apêndices I, II e III, respectivamente:

- soma de dois vetores;
- soma dos elementos de índice par de um vetor;
- ordenação bubblesort.

Cada programa apresenta um comportamento distinto no uso das caches. Dessa forma, o estudante pode observar diversas situações que ocorrem durante a execução dos programas em relação ao uso da cache. A Tabela 1 mostra quatro configurações de caches simuladas neste experimento.

| Caches de dados | Cache A | Cache B | Cache C | Cache D |
|---|---|---|---|---|
| Número total de blocos | 32 | 32 | 32 | 32 |
| Associatividade | 1 | 1 | 4 | 4 |
| Tamanho do bloco | 1 | 2 | 1 | 4 |
| Política de substituição | – | – | LRU | LRU |

**Tabela 1. Configurações utilizadas para os experimentos de simulações de programas RISC-V**

Ao simular os programas de soma de dois vetores e soma dos elementos de índice par de um vetor, utilizando as caches A e B, o estudante pode perceber que, no primeiro programa, a cache A apresenta uma taxa de falhas bastante alta, enquanto a cache B reduz essa taxa, proporcionando um desempenho superior. No entanto, no segundo programa, essa redução na taxa de falhas não é observada na cache B. O aluno é, então, incentivado a tirar conclusões sobre como o aumento do tamanho do bloco pode ter

influenciado as taxas de falhas, se os programas exibem maior ou menor localidade espacial, e como esses fatores impactam o desempenho das caches. As Tabelas 2 e 3 mostram os resultados de desempenho obtidos neste experimento.

| Caches | A | B |
|---|---|---|
| Access Count | 48 | 48 |
| Hit Count | 0 | 24 |
| Miss Count | 48 | 24 |
| Miss Rate | 100% | 50,00% |
| Replacement Count | 16 | 0 |
| Misses per Instruction | 0,26 | 0.13 |
| Average Memory Access Time | 101,00 ciclos | 51.00 ciclos |

**Tabela 2. Resultados de desempenho das caches A e B para o programa soma de dois vetores**

| Caches | A | B |
|---|---|---|
| Access Count | 8 | 8 |
| Hit Count | 0 | 0 |
| Miss Count | 8 | 8 |
| Miss Rate | 100% | 100% |
| Replacement Count | 0 | 0 |
| Misses per Instruction | 0,12 | 0,12 |
| Average Memory Access Time | 101,00 ciclos | 101.00 ciclos |

**Tabela 3. Resultados de desempenho das caches A e B para o programa soma dos elementos de índice par de um vetor**

Em seguida, ao simular o programa bubblesort com as caches A e D, o aluno observa que a primeira apresenta uma taxa de falhas mais alta do que a segunda, embora ambas possuam o mesmo número total de blocos, diferenciando-se apenas na associatividade. O aluno pode analisar o comportamento do programa, a localidade temporal que ele apresenta, e como o aumento da associatividade contribui para a redução da taxa de falhas. A Tabela 4 exibe os resultados de desempenho obtidos neste experimento. A Tabela 4 mostra os resultados de desempenho obtidos neste experimento.

| Caches | A | D |
|---|---|---|
| Access Count | 480 | 480 |
| Hit Count | 464 | 476 |
| Miss Count | 16 | 4 |
| Miss Rate | 3,33% | 0,83% |
| Replacement Count | 0 | 0 |
| Misses per Instruction | 0,01 | 0,00 |
| Average Memory Access Time | 4,33 ciclos | 1,83 ciclos |

**Tabela 4. Resultados de desempenho das caches A e D para o programa bubblesort**

É também importante que o aluno observe, na ilustração da organização da cache disponível na aba Cache Organization do ECS, como os circuitos auxiliares de acesso à cache variam entre as quatro configurações de cache, com ou sem a utilização de multiplexadores. O aluno deve compreender a função de cada multiplexador: selecionar a palavra do bloco acessada pelo processador (como nas caches B e D) ou escolher a via onde ocorreu o acerto (como nas caches C e D). As Figuras 1, 2 e 3 mostram a organização das caches B, C e D, respectivamente.

As conclusões obtidas neste experimento auxiliam o estudante a compreender melhor a organização das memórias cache com diferentes configurações, como essas configurações influenciam a interpretação dos bits do endereço de memória e como os campos extraídos desse endereço são utilizados no acesso à cache. Além disso, permitem que o aluno identifique situações em que a cache aproveita a localidade temporal e/ou espacial do programa para reduzir a taxa de falhas, melhorando assim o desempenho do programa.

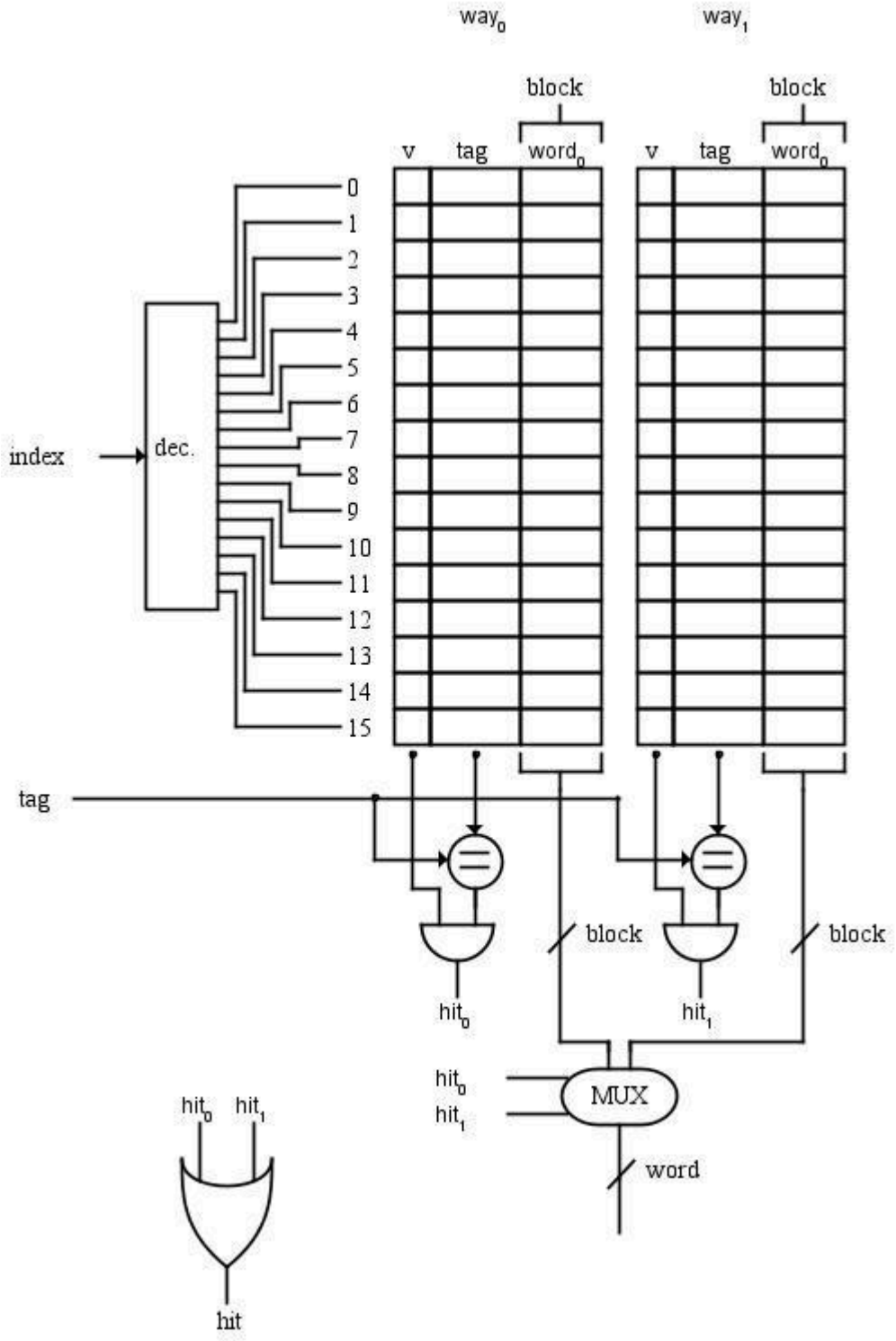


**Figura 1. Organização da cache B**

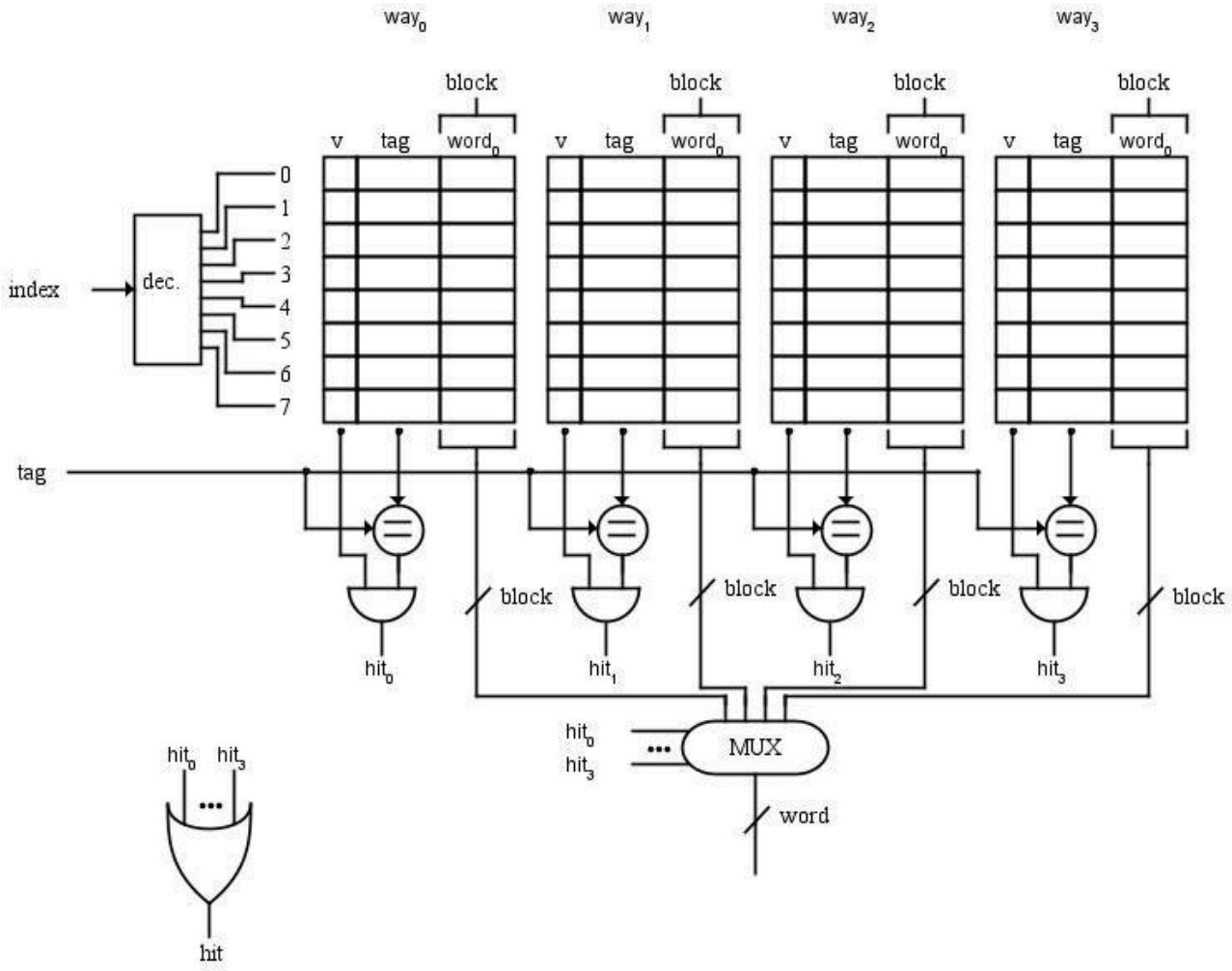


**Figura 2. Organização da cache C**

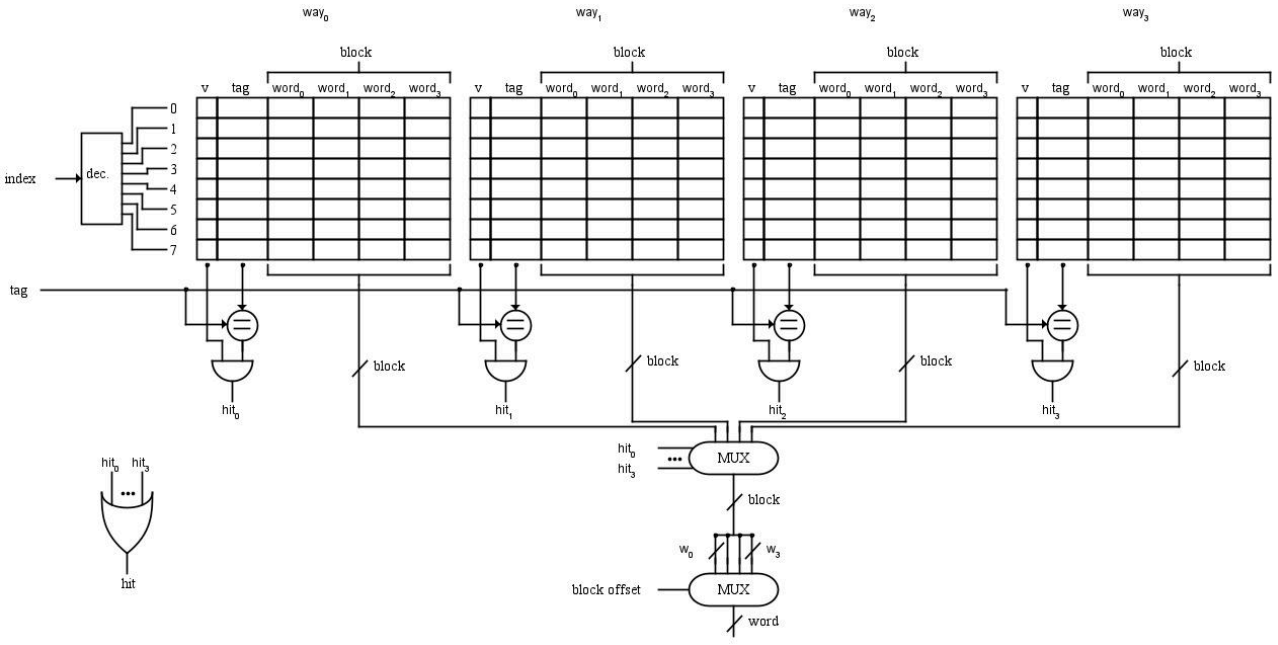


**Figura 3. Organização da cache D**

## 2. Experimento de Simulação de *Trace* de Endereços

Outra forma de utilizar o ECS é por meio de um arquivo de *trace* de endereços de memória, que contém a sequência de endereços acessados durante a execução de um programa. Dessa maneira, não é necessário simular o programa completo, sendo suficiente simular os acessos à cache com base nesses endereços. Ao usar o ECS como um simulador *memory trace-driven*, é possível simular caches com *traces* gerados por outras ferramentas, além de permitir a realização de experimentos com caches que envolvem um número muito grande de acessos.

Neste experimento, utiliza-se um *trace* de memória, empregado em [2], que contém 55.986.879 acessos (a instruções e dados), referentes à execução do programa Rijndael encoder do *benchmark* MiBench. O aluno pode realizar simulações considerando três cenários diferentes para a hierarquia de memórias:

a. Com uma única cache unificada, com instruções e dados;
b. Com duas caches, uma apenas com instruções e outra apenas com dados (caches *split*);
c. Sem memórias cache (ou seja, todo acesso é feito diretamente à memória principal).

O aluno é incentivado a analisar e comparar o desempenho obtido em cada um desses cenários. Utilizando as métricas de desempenho fornecidas pelo ECS, o aluno pode calcular a taxa de falhas combinada das caches *split* no cenário (b) e compará-la com a taxa de falhas da cache unificada no cenário (a). Além disso, pode calcular o tempo médio de acesso à memória (*Average Memory Access Time* – AMAT) para os cenários (b) e (c), usando os tempos de acesso definidos e a taxa de falhas combinada (para o cenário (b). Dessa forma, o aluno poderá comparar esses valores com o AMAT calculado pelo ECS para o cenário (a) e avaliar o impacto do uso e da organização das caches (unificada ou *split*) no desempenho da hierarquia de memórias. A Tabela 5 apresenta os resultados de desempenho obtidos neste experimento.

| Caches | Dados | Instruções | Unificada |
|---|---|---|---|
| Miss Rate | 12,37% | 71,12% | 71,18% |
| Average Memory Access Time | 13,37 | 72,12 | 72,18 |

**Tabela 5. Resultados de desempenho das caches A e C para o programa bubblesort**

## 3. Conclusão

A ferramenta ECS é um recurso pedagógico valioso para o ensino de conceitos relacionados à hierarquia de memórias e memórias cache nas disciplinas de Arquitetura e Organização de Computadores. O simulador pode ser utilizado pelos alunos tanto em aulas práticas de laboratório quanto no estudo fora de sala de aula, por meio da realização de diversos experimentos.

## 4. Bibliografia


1. Escada, Y. Martins, Simulador de Caches para o Processador RISC-V, Trabalho de Conclusão de Curso, Faculdade de Computação, 2024.
2. Lima, D. P. e Moreano, N. ECS e EMCS: Simuladores de Caches para o Apoio Pedagógico no Ensino de Arquitetura de Computadores, Anais do XXIX Workshop sobre Educação em Computação, 2021.


### Apêndice I: Programa RISC-V de Soma de Dois Vetores

```
#------------------------------------------------------------------------
# PROGRAMA PRINCIPAL
# Algoritmo:
#       soma = 0
#       for (i = 0 ; i < n ; i++)
#           c[i] = a[i] + b[i]
#       Encerra execução do programa
# Uso dos registradores:
#       s0: n
#       s1: endereço de a[i] na memória
#       s2: endereço de b[i] na memória
#       s3: endereço de c[i] na memória
#       s4: i
#       t0: auxiliar para comparação
#       t1: a[i]
#       t2: b[i]
#       t3: c[i]
#       a7: parâmetro para chamada ao sistema

                                                # Inicialização
main:               addi    s0, zero, 16        # n = 16
                    la      s1, veta                # s1 = endereço do a[0] na memória
                    la      s2, vetb                # s2 = endereço do b[0] na memória
                    la      s3, vetc                # s3 = endereço do c[0] na memória
                    add s4, zero, zero          # i = 0
for:                slt t0, s4, s0              # i < n ?
                    beq t0, zero, fora_for  # Se i >= n, desvia para fora do for
                    lw      t1, 0 (s1)              # t1 = valor do a[i] lido da memória
                    lw      t2, 0 (s2)              # t2 = valor do b[i] lido da memória
                    add t3, t1, t2              # t3 = a[i] + b[i]
                    sw      t3, 0 (s3)              # c[i] = $t3
                    addi    s4, s4, 1               # i++
                    addi    s1, s1, 4               # s1 = endereço do próximo elemento do vetor a
                    addi    s2, s2, 4               # s2 = endereço do próximo elemento do vetor b
                    addi    s3, s3, 4               # s3 = endereço do próximo elemento do vetor c
                    j       for                     # Desvia para início do for
fora_for:       addi    a7, zero, 10            # Chamada ao sistema para encerrar programa
                    ecall
#------------------------------------------------------------------------
                    .data                           # Área de dados
#------------------------------------------------------------------------
                                                # Variáveis e estruturas de dados do programa
veta:               .word 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
vetb:               .word 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
vetc:               .word 0 0 0 0 0 0 0 0 0 0  0  0  0  0  0  0
#------------------------------------------------------------------------
```

## Apêndice II: Programa RISC-V de Soma dos Elementos de Índice Par de um Vetor

```
#------------------------------------------------------------------------
# PROGRAMA PRINCIPAL
# Algoritmo:
#       soma = 0
#       for (i = 0 ; i < n ; i=i+2)
#           soma = soma + vetor[i]
#       Encerra execucao do programa
# Uso dos registradores:
#       s0: n
#       s1: endereço inicial de vetor[i] na memoria
#       s2: soma
#       s3: i
#       t0: auxiliar para comparacao
#       t1: vetor[i]
#       a7: parametro para chamada ao sistema

                                                    # Inicializacao
main:               addi    s0, zero, 16            # n = 16
                    la      s1, vetor               # s1 = endereco de vetor[0] na memoria
                    add s2, zero, zero          # soma = 0
                    add s3, zero, zero          # i = 0
for:                slt t0, s3, s0              # i < n ?
                    beq t0, zero, fora_for      # Se i >= n, desvia para fora do for
                    lw      t1, 0 (s1)              # t1 = valor do vetor[i] lido da memoria
                    add s2, s2, t1              # soma = soma + vetor[i]
                    addi    s3, s3, 2               # i = i + 2
                    addi    s1, s1, 8               # s1 = endereco do proximo elemento de índice par de vetor
                    j       for                     # Desvia para início do for
fora_for:       addi    a7, zero, 10            # Chamada ao sistema para encerrar programa
                    ecall
#------------------------------------------------------------------------
                    .data                               # Area de dados
#------------------------------------------------------------------------
                                                    # Variaveis e estruturas de dados do programa
vetor:          .word 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
#------------------------------------------------------------------------
```

## Apêndice III: Programa RISC-V de Ordenação BubbleSort

```
#------------------------------------------------------------------------
# PROGRAMA PRINCIPAL
# Algoritmo:
#       trocou = true
#       limite = n-1
#       while (limite > 0) AND (trocou)
#           trocou = false
#           for (i = 0 ; i < limite ; i++)
#               if vetor[i] > vetor[i+1]
#                   // Troca elementos vetor[i] e vetor[i+1]
#                   aux = vetor[i]
#                   vetor[i] = vetor[i+1]
#                   vetor[i+1] = aux
#                   trocou = true
#           limite--
#       Encerra execução do programa
# Uso dos registradores:
#       s0: n
#       s1: endereço inicial do vetor na memoria
#       s2: trocou
#       s3: limite
#       s4: i
#       t0: auxiliar para comparacao
#       t1: endereco de vetor[i] na memoria
#       t2: vetor[i]
#       t3: vetor[i+1]
#       a7: parametro para chamada ao sistema

                                                # Inicializacao
main:
                addi    s0, zero, 16            # n = 16
                la      s1, vetor               # s1 = endereco inicial do vetor na memoria
                addi    s2, zero, 1             # trocou = 1
                addi    s3, s0, -1              # limite = n-1
while:          slt t0, zero, s3            # limite > 0 ?
                beq t0, zero, fora_while        # Se limite <= 0, desvia para fora do while
                beq s2, zero, fora_while        # Se trocou == false, desvia para fora do while
                add s2, zero, zero              # trocou = 0
                add s4, zero, zero              # i = 0
for:            slt t0, s4, s3                  # i < limite ?
                beq t0, zero, fora_for      # Se i >= limite, desvia para fora do for
                add t1, s4, s4                  # t1 = i * 4
                add t1, t1, t1
                add t1, s1, t1                  # t1 = endereco de vetor[i] na memoria
                lw      t2, 0 (t1)              # t2 = valor de vetor[i] lido da memoria
                lw      t3, 4 (t1)              # t3 = valor de vetor[i+1] lido da memoria
                slt t0, t3, t2              # vetor[i] < vetor[i+1] ?
                beq t0, zero, fora_if           # Se vetor[i] <= vetor[i+1], desvia para fora do if
                                                # Troca vetor[i] e vetor[i+1]
                sw      t3, 0 (t1)              # vetor[i] = t3
                sw      t2, 4 (t1)              # vetor[i+1] = t2
                addi    s2, zero, 1             # trocou = 1
fora_if:        addi    s4, s4, 1               # i++
                j       for                     # Desvia para inicio do for
fora_for:       addi    s3, s3, -1              # limite--
                j       while                   # Desvia para inicio do while
fora_while:     addi    a7, zero, 10            # Chamada ao sistema para encerrar programa
                ecall
#------------------------------------------------------------------------
                .data                           # Area de dados
#------------------------------------------------------------------------
                                                # Variaveis e estruturas de dados do programa
vetor:          .word 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 # Vetor a ser ordenado
#------------------------------------------------------------------------
```