

UNIVERSIDADE DE BRASÍLIA  
INSTITUTO DE CIÊNCIAS EXATAS  
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO  
**116394 ORGANIZAÇÃO E ARQUITETURA DE COMPUTADORES**

Trabalho I: Memória do RISCv

## **OBJETIVO**

Este trabalho consiste na simulação de instruções de acesso à memória do RISCv RV32I em linguagem C.

## **DESCRIÇÃO**

### ***Tipos de dados***

Utilizar os tipos de dados definidos em <stdint.h>:

**uint8\_t** : inteiro sem sinal de 8 bits.

**int32\_t, int16\_t, int8\_t** : inteiro com sinal de 32, 16 e 8 bits, respectivamente.

### ***Memória***

A memória é simulada como um arranjo de inteiros de 32 bits.

```
#define MEM_SIZE 4096  
int32_t mem[MEM_SIZE];
```

Ou seja, a memória é um arranjo de 4KWords, ou 16KBytes.

### ***Funções de acesso à Memória***

Desenvolver as funções:

```
int32_t lw(uint32_t address, int32_t kte);
```

Lê um inteiro alinhado - endereços múltiplos de 4.

A função calcula o endereço de memória somando os parâmetros:

**Endereço palavra = address + kte**

A função deve checar se o endereço é um múltiplo de 4 (%4 == 0).

Se não for, deve escrever uma mensagem de erro e retornar zero.

Se o endereço estiver correto, a função deve:

- Dividi-lo por 4 para obter o índice do vetor memória
- Retornar o valor lido da memória

```
int32_t lb(uint32_t address, int32_t kte);
```

Lê um byte do vetor memória e retorna-o, estendendo o sinal para 32 bits.

Lembrando que as palavras da memória tem 4 bytes cada, para acessar um byte dentro da palavra pode-se:

- Ler a palavra que contém o byte e, por operações de mascaramento, extrair byte endereçado, ou
- Criar um ponteiro para byte e fazer um *type cast* (coerção de tipo) do endereço do vetor memória (*int \**) para byte (*char \**).

```
int32_t lbu(uint32_t address, int32_t kte);
```

Lê um byte do vetor memória e retorna-o como um número positivo, ou seja, todos os bits superiores devem ser zerados.

```
void sw(uint32_t address, int32_t kte, int32_t dado);
```

Escreve um inteiro alinhado na memória - endereços múltiplos de 4. O cálculo do endereço é realizado da mesma forma que na operação *lw()*.

```
void sb(uint32_t address, int32_t kte, int8_t dado);
```

Escreve um byte na memória. Caso utilize operações de mascaramento, a palavra que contém o byte deve ser lida da memória, o byte deve ser posicionado corretamente através de deslocamentos e a escrita ocorre utilizando máscaras. Alternativamente pode-se utilizar a coerção para (*char \**) e escrever diretamente na posição usando o endereço calculado como índice.

## Verificação

*A seguir são sugeridos procedimentos de teste das funções.*

O aluno é encorajado a realizar um procedimento de teste mais completo. **Um ponto da avaliação deste trabalho será dado pela inclusão de outros testes.**

1. Iniciar a memória: executar a seguinte sequência de operações de escrita.

- a. `sb(0, 0, 0x04); sb(0, 1, 0x03); sb(0, 2, 0x02); sb(0, 3, 0x01);`
- b. `sb(4, 0, 0xFF); sb(4, 1, 0xFE); sb(4, 2, 0xFD); sb(4, 3, 0xFC);`
- c. `sw(12, 0, 0xFF);`
- d. `sw(16, 0, 0xFFFF);`
- e. `sw(20, 0, 0xFFFFFFFF);`
- f. `sw(24, 0, 0x80000000);`

2. Imprimir o conteúdo da memória em formato hexa. O resultado deve ser:

- a. `mem[0] = 01020304`
- b. `mem[1] = fcfdfeff`
- c. `mem[2] = 00000000`
- d. `mem[3] = 000000FF`
- e. `mem[4] = 0000FFFF`

```
f. mem[5] = FFFFFFFF
g. mem[6] = 80000000
```

3. Ler os dados e imprimir em hexadecimal:

- a. lb(4,0), lb(4,1), lb(4,2) lb(4,3)
- b. lbu(4,0), lbu(4,1), lbu(4,2) lbu(4,3)
- c. lw(12,0), lw(16, 0), lw(20,0)

### **Entrega**

- o código fonte do simulador, com a indicação da plataforma utilizada:
  - qual compilador empregado
  - sistema operacional
  - IDE (Eclipse, XCode, etc)