

Organização e Arquitetura de Computadores

Trabalho 1 - Memória do RISCv

João Pedro de Oliveira Silva

190057807

OBJETIVO

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

RESPOSTAS AS PERGUNTAS

A)

1 byte = conjunto de 8 bits

Word = 4 bytes = conjunto de 32 bits

Half word = 2 bytes = conjunto de 16 bits

B)

gcc (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0

C)

Tenho uma WSL de Ubuntu 20.04 no Windows 10.

D)

Visual Studio Code

Funções implementadas

`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 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 têm 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.

TESTES

```
jp@LAPTOP-NFERDPA3: ~/OAC
jp@LAPTOP-NFERDPA3:~/OAC$ gcc RiscV.c -o p
jp@LAPTOP-NFERDPA3:~/OAC$ ./p
a: mem[0] = 0x01020304
b: mem[1] = 0xFCFDFEFF
c: mem[2] = 0x00000000
d: mem[3] = 0x000000FF
e: mem[4] = 0x0000FFFF
f: mem[5] = 0xFFFFFFFF
g: mem[6] = 0x80000000

lb(4,0) = 0xFFFFFFFF
lb(4,1) = 0xFFFFFFFFE
lb(4,2) = 0xFFFFFFFFD
lb(4,3) = 0xFFFFFFFFC

lbu(4,0) = 0x000000FF
lbu(4,1) = 0x000000FE
lbu(4,2) = 0x000000FD
lbu(4,3) = 0x000000FC

lw(12,0) = 0x000000FF
lw(16,0) = 0x0000FFFF
lw(20,0) = 0xFFFFFFFF

Testes extras

Extra 1: Erro de Endereço 0x0000001D. Não é multiplo de 4
Extra 2: Leitura de byte negativo corretamente. 0xFFFFFFFF = -1
Extra 3: Leitura de byte sem sinal corretamente. 0x000000FF = 255
Extra 4: Error, Endereço fora da memória.
Extra 5: Erro de Endereço 0xFFFFFFFF. Não é multiplo de 4
```