UNIVERSIDADE DE BRASÍLIA
INSTITUTO DE CIÊNCIAS EXATAS
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

116394 ORGANIZAÇÃO E ARQUITETURA DE COMPUTADORES

Trabalho II: Memória do RISCV

OBJETIVO

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

DESCRIÇÃO

Tipos de dados

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

uint32_t, uint16_t, uint8_t : inteiro sem sinal de 32, 16 e 8 bits, respectivamente.

int32_t, int16_t, int8_t : inteiro com sinal de 32, 16 e 8 bits, respectivamente.

Exemplo de uso:

```
#include <stdio.h>
#include <stdint.h>

int main(int argc, char *argv[])
{
    uint32_t u32;
    uint16_t u16;
    uint8_t u8;

    printf("sizeof(u32) = %d\n", sizeof(u32) );
    printf("sizeof(u16) = %d\n", sizeof(u16) );
    printf("sizeof(u8) = %d\n", sizeof(u8) );

    return 0;
}
```

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.

Desenvolver a função `void dump_mem(uint32_t addr, uint32_t wsize)` que imprime o conteúdo da memória no formato hexa, *wsize* palavras iniciando no endereço *addr*.

Ex:

```
mem[0] = 01020304
```

Dump_mem utiliza endereços de *byte*, e imprime palavra por palavra. Os endereços devem ser múltiplos de 4.

Acesso à Memória

Desenvolver as funções:

```
int32_t  lw(uint32_t address, int32_t kte);
      => lê um inteiro alinhado - endereços múltiplos de 4
int32_t  lh(uint32_t address, int32_t kte);
      => lê meia palavra, 16 bits - retorna inteiro com sinal
int32_t  lhu(uint32_t address, int32_t kte);
      => lê meia palavra, 16 bits formato inteiro sem sinal
int32_t  lb(uint32_t address, int32_t kte);
      => lê um byte - retorna inteiro com sinal
int32_t  lbu(uint32_t address, int32_t kte);
      => lê um byte - 8 bits formato inteiro sem sinal
void  sw(uint32_t address, int32_t kte, int32_t dado);
      => escreve um inteiro alinhado na memória - endereços múltiplos
de 4
void  sh(uint32_t address, int32_t kte, int16_t dado);
      => escreve meia palavra, 16 bits - endereços múltiplos de 2
void  sb(uint32_t address, int32_t kte, int8_t dado);
      => escreve um byte na memória
```

Atenção: **endereço do dado** = (**address + kte)** !

Os endereços são todos de *byte*. A operação de leitura de *byte* retorna um inteiro com o *byte* lido na posição menos significativa. A escrita de um *byte* deve colocá-lo na posição correta dentro da palavra de memória. Para endereçar palavras e meias-palavras é necessário ajustar o endereço de acordo.

Verificação do Simulador

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.sh(8, 0, 0xFFF0); sh(8, 2, 0x8C);
```

```
d.sw(12, 0, 0xFF);
e.sw(16, 0, 0xFFFF);
f.sw(20, 0, 0xFFFFFFFF);
g.sw(24, 0, 0x80000000);
```

2. Imprimir o conteúdo da memória:

   a. dump_mem(0, 7):

```
mem[0] = 01020304
mem[1] = fcfdfeff
mem[2] = 008cfff0
mem[3] = 000000FF
mem[4] = 0000FFFF
mem[5] = FFFFFFFF
mem[6] = 80000000
```

3. Ler os dados da seguinte maneira:

   Obs: *lb()* deve ser impresso apenas com dois dígitos em hexa, e *lh()* apenas com 4.

   a. lb(0,0), lb(0,1), lb(0,2) lb(0,3) : imprimir em hexa e decimal

   b. lb(4,0), lb(4,1), lb(4,2) lb(4,3) : imprimir em hexa e decimal

   c. lbu(4,0), lbu(4,1), lbu(4,2) lbu(4,3) : imprimir em decimal

   d. lh(8,0), lh(8,2) : imprimir em hexa e decimal

   e. lhu(8,0), lhu(8,2) : imprimir em decimal

   f. lw(12,0), lw(16, 0), lw(20,0) : imprimir em hexa e decimal

Entrega

Entregar:
- relatório da implementação:
  - descrição do problema
  - descrição sucinta das funções implementadas
  - testes e resultados
- o código fonte do simulador, com a indicação da plataforma utilizada:
  - qual compilador empregado
  - sistema operacional
  - IDE (Eclipse, XCode, etc)