Relatorio Simulador MIPS em C

Jader Martins Camboim de Sá

29 março, 2018

1 Objetivo

Este trabalho consiste na simulação das instruções de acesso à memória do MIPS em linguagem C explorando diferentes tipos de dados.

1.1 Descrição do Problema

O simulador aqui descrito, busca implementar funções de `STORE` para armazenamento na memoria e de `LOAD` para carregamento dos dados, para diferentes tamanhos de dados (`byte`, `half`, `word`).

1.2 Funções Implementadas

Foram implementadas as seguintes funções através de máscaras e operações *bitwise*:

1. **sb(address, kte, dado)**

   Insere um `dado` de 1 byte na região de memoria `address` na posição `kte`.

2. **sh(address, kte, dado)**

   Insere um `dado` de 2 bytes (meia-palavra) na região de memoria `address` na posição `kte`.

3. **sw(address, kte, dado)**

   Insere um `dado` de 4 bytes (palavra) na região de memoria `address` na posição `kte`.

4. **lb(address, kte)**

   Carrega um dado de 1 byte da região de memoria `address` na posição `kte`.

5. **lbu(address, kte)**

   Carrega um dado de 1 byte **sem sinal** da região de memoria `address` na posição `kte`.

6. **lh(address, kte)**

   Carrega um dado de 2 bytes (meia-palavra) da região de memoria `address` na posição `kte`.

7. **lhu(address, kte)**

   Carrega um dado de 2 bytes (meia-palavra) **sem sinal** da região de memoria `address` na posição `kte`.

8. **lw(address, kte)**

   Carrega um dado de 4 bytes (palavra) da região de memoria `address` na posição `kte`.

9. **dump_mem(add, size)**

   Imprime conteúdo da memória da posição `add` até a posição `size`.

1.3 Testes e Resultados

Para realização dos testes utilizei a biblioteca `CATCH` para testes unitários, avaliando cada uma das funções em cases. Os testes foram os seguintes: **test.cpp**

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
a.out is a Catch v1.9.4 host application.
Run with -? for options

--------------------------------------------------------------------------------
Byte memory test
--------------------------------------------------------------------------------
test.cpp:5
................................................................................

test.cpp:8:
PASSED:
  REQUIRE( mem[0] == 0x01020304 )
with expansion:
  16909060 (0x1020304)
  ==
  16909060 (0x1020304)

test.cpp:9:
PASSED:
  REQUIRE( mem[1] == 0xfcfdfeff )
```

```
with expansion:
  -50462977 == 4244504319 (0xfcfdfeff)

-------------------------------------------------------------------------------
Half-Word memory test
-------------------------------------------------------------------------------
test.cpp:12
.....................................................................

test.cpp:14:
PASSED:
  REQUIRE( mem[2] == 0x008cfff0 )
with expansion:
  9240560 (0x8cfff0) == 9240560 (0x8cfff0)

-------------------------------------------------------------------------------
Word memory test
-------------------------------------------------------------------------------
test.cpp:17
.....................................................................

test.cpp:22:
PASSED:
  REQUIRE( mem[3] == 0x000000FF )
with expansion:
  255 == 255

test.cpp:23:
PASSED:
  REQUIRE( mem[4] == 0x0000FFFF )
with expansion:
  65535 (0xffff) == 65535 (0xffff)

test.cpp:24:
PASSED:
  REQUIRE( mem[5] == 0xFFFFFFFF )
with expansion:
  -1 == 4294967295 (0xffffffff)

test.cpp:25:
PASSED:
  REQUIRE( mem[6] == 0x80000000 )
with expansion:
  -2147483648 == 2147483648 (0x80000000)

-------------------------------------------------------------------------------
```

```
Load byte sig-unsig test
-------------------------------------------------------------------------------
test.cpp:28
...............................................................................

test.cpp:31:
PASSED:
  REQUIRE( lb(0, 3) == 0x01 )
with expansion:
  1 == 1

test.cpp:32:
PASSED:
  REQUIRE( lbu(4, 2) == 0xfd )
with expansion:
  253 == 253

-------------------------------------------------------------------------------
Load half-word sig-unsig test
-------------------------------------------------------------------------------
test.cpp:35
...............................................................................

test.cpp:37:
PASSED:
  REQUIRE( lb(0, 3) == 0x01 )
with expansion:
  1 == 1

test.cpp:38:
PASSED:
  REQUIRE( lbu(4, 2) == 0xfd )
with expansion:
  253 == 253

-------------------------------------------------------------------------------
Load word test
-------------------------------------------------------------------------------
test.cpp:41
...............................................................................

test.cpp:46:
PASSED:
  REQUIRE( lw(16, 0) == 0xFFFF )
with expansion:
  65535 (0xffff) == 65535 (0xffff)
```

```
test.cpp:47:
PASSED:
  REQUIRE( lw(24, 0) == 0x80000000 )
with expansion:
  -2147483648 == 2147483648 (0x80000000)

===============================================================================
All tests passed (13 assertions in 6 test cases)


~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
a.out is a Catch v1.9.4 host application.
Run with -? for options

-------------------------------------------------------------------------------
Byte memory test
-------------------------------------------------------------------------------
test.cpp:5
.....................................................................

test.cpp:8:
PASSED:
  REQUIRE( mem[0] == 0x01020304 )
with expansion:
  16909060 (0x1020304)
  ==
  16909060 (0x1020304)

test.cpp:9:
PASSED:
  REQUIRE( (int32_t)mem[1] == 0xfcfdfeff )
with expansion:
  -50462977 == 4244504319 (0xfcfdfeff)

-------------------------------------------------------------------------------
Half-Word memory test
-------------------------------------------------------------------------------
test.cpp:12
.....................................................................

test.cpp:14:
PASSED:
  REQUIRE( (int32_t)mem[2] == 0x008cfff0 )
with expansion:
  9240560 (0x8cfff0) == 9240560 (0x8cfff0)
```

```
-------------------------------------------------------------------------------
Word memory test
-------------------------------------------------------------------------------
test.cpp:17
...............................................................................

test.cpp:22:
PASSED:
  REQUIRE( (int32_t)mem[3] == (int32_t)0x000000FF )
with expansion:
  255 == 255

test.cpp:23:
PASSED:
  REQUIRE( (int32_t)mem[4] == (int32_t)0x0000FFFF )
with expansion:
  65535 (0xffff) == 65535 (0xffff)

test.cpp:24:
PASSED:
  REQUIRE( (int32_t)mem[5] == (int32_t)0xFFFFFFFF )
with expansion:
  -1 == -1

test.cpp:25:
PASSED:
  REQUIRE( (int32_t)mem[6] == (int32_t)0x80000000 )
with expansion:
  -2147483648 == -2147483648

-------------------------------------------------------------------------------
Load byte sig-unsig test
-------------------------------------------------------------------------------
test.cpp:28
...............................................................................

test.cpp:31:
PASSED:
  REQUIRE( (int32_t)lb(0, 3) == 0x01 )
with expansion:
  1 == 1

test.cpp:32:
PASSED:
  REQUIRE( (int32_t)lbu(4, 2) == 0xfd )
```

```
with expansion:
  253 == 253

-------------------------------------------------------------------------------
Load half-word sig-unsig test
-------------------------------------------------------------------------------
test.cpp:35
...............................................................................

test.cpp:37:
PASSED:
  REQUIRE( lb(0, 3) == 0x01 )
with expansion:
  1 == 1

test.cpp:38:
PASSED:
  REQUIRE( lbu(4, 2) == 0xfd )
with expansion:
  253 == 253

-------------------------------------------------------------------------------
Load word test
-------------------------------------------------------------------------------
test.cpp:41
...............................................................................

test.cpp:46:
PASSED:
  REQUIRE( lw(16, 0) == 0xFFFF )
with expansion:
  65535 (0xffff) == 65535 (0xffff)

test.cpp:47:
PASSED:
  REQUIRE( lw(24, 0) == (int32_t)0x80000000 )
with expansion:
  -2147483648 == -2147483648

===============================================================================
All tests passed (13 assertions in 6 test cases)
```

2 Implementação e Especificações

Está sessão apresenta os códigos utilizados para o simulador, especificações de software, compilação e desenvolvimento.

2.1 Código Fonte

funcs.h

```
#ifndef FUNCS_H
#define FUNCS_H

#include <stdio.h>
#include <math.h>
#include <stdint.h>
#define MEM_SIZE 4096

int32_t mem[MEM_SIZE];

void dump_mem(uint32_t add, uint32_t size) {
    for (int i = add/4; i < size/4; ++i) {
        printf("mem[%d]\t= %08x\n", i, mem[i]);
    }
}

int32_t lw(uint32_t address, int16_t kte) {
    // => lê um inteiro alinhado - endereços múltiplos de 4
    int32_t dado = mem[address/4];
    return dado;
}

int32_t lh(uint32_t address, int16_t kte) {
    // => lê meia palavra, 16 bits - retorna inteiro com sinal
    int32_t dado = (mem[address/4] >> (kte*8)) & 0xFFFF;
    return dado;
}

uint32_t lhu(uint32_t address, int16_t kte) {
    // => lê meia palavra, 16 bits formato inteiro sem sinal
    uint32_t dado = (mem[address/4] >> (kte*8)) & 0xFFFF;
    return dado;
}

int32_t lb(uint32_t address, int16_t kte) {
    // => lê um byte - retorna inteiro com sinal
```

```
    int32_t dado = (mem[address/4] >> (kte*8)) & 0xFF;
    return dado;
}

uint32_t lbu(uint32_t address, int16_t kte) {
    // => lê um byte - 8 bits formato inteiro sem sinal
    uint32_t dado = (mem[address/4] >> (kte*8)) & 0xFF;
    return dado;
}

void sw(uint32_t address, int16_t kte, int32_t dado) {
    // => escreve um inteiro alinhado na memória - endereços múltiplos de 4
    mem[address/4] = dado;
}

void sh(uint32_t address, int16_t kte, int16_t dado) {
    // => escreve meia palavra, 16 bits - endereços múltiplos de 2
    uint32_t shifter = kte * 8;
    uint32_t mask = ~(0xffff << shifter);
    mem[address/4] = (mem[address/4] & mask) | (dado << shifter);
}

void sb(uint32_t address, int16_t kte, int8_t dado) {
    // => escreve um byte na memória
    uint32_t shifter = kte * 8;
    uint32_t mask = ~(0xff << shifter);
    mem[address/4] = (mem[address/4] & mask) | (dado << shifter);
}
#endif
```

main.c

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

int main(int argc, char *argv[])
{
    sb(0, 0, 0x04); sb(0, 1, 0x03); sb(0, 2, 0x02); sb(0, 3, 0x01);
    sb(4, 0, 0xFF); sb(4, 1, 0xFE); sb(4, 2, 0xFD); sb(4, 3, 0xFC);
    sh(8, 0, 0xFFF0); sh(8, 2, 0x8C);
    sw(12, 0, 0xFF);
    sw(16, 0, 0xFFFF);
    sw(20, 0, 0xFFFFFFFF);
    sw(24, 0, 0x80000000);
    dump_mem(0, 28);
    return 0;
```

```
}
```

2.2 Especificações de Desenvolvimento

Para a escrita do simulador utilizei o `VIM` com plugins auxiliares e o `GNU/Make`, para compilação utilizei o `GCC` e para os testes a biblioteca `CATCH`, o sistema operacional é o `Ubuntu 16.04 LTS`.