Instituto de Ciências Matemáticas e de Computação – ICMC/USP
Departamento de Sistemas de Computação – SSC
SCE-607 Organização de Computadores Digitais I – Engenharia de Computação
2º Trabalho - Simulador de hierarquia de memória multinível do MIPS
Disponibilizado em 24/05/2013. Data para entrega do trabalho será definida em sala de aula.

O objetivo deste Segundo Trabalho prático é implementar na Linguagem C um simulador de acesso à hierarquia de memória multinível visto nas nossas aulas teóricas. Este trabalho deverá implementar dois níveis de memória cache (L1 e L2) e a memória principal.

Importante: o simulador deverá estar baseado no conteúdo ministrado em sala de aula.

Para o acesso à hierarquia de memória (caches L1 e L2 + memória principal), utilize, obrigatoriamente, esta função:

int memory_access(int addr, int *data, int type);

onde:

- ***addr*** é o endereço (a *B*yte) da palavra de 32 bits a ser obtida na memória;
- ***data*** é um ponteiro para o dado a ser escrito na memória (quando a operação for de escrita). Quando a operação for de leitura, ***data*** terá a palavra lida da memória;
- ***type*** é o tipo de acesso (0 para leitura e 1 para escrita);
- ***memory_access()*** retorna 1 (um) caso tenha sucesso no acesso e, -1 caso tenha ocorrido um erro no acesso.

Lembre-se que é apenas através da função ***memory_access()*** que se fará todo o acesso à cache e à memória principal.

A memória principal deverá ter 256 palavras no total. Ela é representada por um vetor de bytes (char) com RAM_SIZE posições (uma macro contendo 1024, no caso). Lembrando que cada palavra possui 4bytes (char), totalizando assim, uma memória principal de 1024 bytes.

Para o acesso à cache, considere os seguintes parâmetros:

- **Cache L1: cache associativa por conjunto com: 4 conjuntos, 2 posições (ou 2 blocos) por conjunto, 2 palavras por bloco e 32 bits em cada palavra. A política de substituição é a FIFO e a técnica de atualização *Write-Throught***. A cache L1 deve ser representada como um vetor de conjuntos com CACHE_L1_SIZE posições (uma macro contendo 4, no caso).

- **Cache L2: cache associativa por conjunto com: 8 conjuntos, 2 posições (ou 2 blocos) por conjunto, 2 palavras por bloco e 32 bits em cada palavra. A política de substituição é a FIFO e a técnica de atualização *Write-Back***. A cache L2 deve ser representada como um vetor de conjuntos com CACHE_L2_SIZE posições (uma macro contendo 8, no caso).

Ao inserir um novo bloco em um conjunto com blocos não preenchidos, escolha sempre primeiro os blocos de menor ordem (número). Em outras palavras, escolha os blocos livres de um conjunto da esquerda para a direita, conforme explicado em sala de aula.

Utilize a ordenação de bytes ***big-endian***.

Considere, obrigatoriamente, o código abaixo com a função *main* (no arquivo *main.c*) e as estruturas (no arquivo *memory_manager.h*) que definem a memória principal (*memory*) e as caches (*cache_L1* e *cache_L2*). A função ***memory_access()*** está implementada no arquivo denominado "*memory_manager.c*". A função ***memory_access()*** não deve imprimir nenhuma informação, a função ***main()*** ficará responsável por essa tarefa. **Tanto a função *main*() como as estruturas disponibilizadas NÃO devem ser alteradas em nenhuma hipótese.**

O código escrito na linguagem C deve compilar com sucesso no compilador ***gcc*** **(de 32bits)** e o executável deverá funcionar no sistema operacional Linux de 32bits. Juntamente com os códigos está sendo disponibilizado um arquivo *makefile* para facilitar o processo de compilação (utilize o comando: *make*), o executável gerado se chamará "*main"*.

Este trabalho deverá ser feito em grupo, o qual já foi determinado no início do semestre letivo. Envie via Moodle STOA apenas um arquivo por grupo contendo o código fonte em C ("*memory_manager.c*"). Este arquivo deve ter o nome: TY-GXX-nnnn (onde Y indica a Turma (1 ou 2), XX indica o nr do grupo e nnnn indica o nome de um dos integrantes do grupo). Forneça, obrigatoriamente, como comentário no corpo do arquivo submetido o número da turma, o número do grupo e o nome de todos os integrantes do grupo que efetivamente participaram do desenvolvimento.

Quaisquer dúvidas/sugestões/erros na especificação ou no código fonte passado aos grupos deverão ser repassados ao professor e aos estagiários PAEs para que os mesmos possam ser sanados.

========== CÓDIGO COM AS ESTRUTURAS - *[memory_manager.h]* ==========

```
/*
 * ATENCAO: NAO ALTERAR ESTE ARQUIVO
 * Seu trabalho deve ser desenvolvido no arquivo memory_manager.c
 * na funcao memory_access(int addr, int *data, int type).
 *
 * Para compilar o projeto execute o comando
 *     make
 * Para executar digite o comando:
 *    ./main
 *
 * */

#ifndef MEMORY_MANAGER_H_INCLUDED
#define MEMORY_MANAGER_H_INCLUDED

#define RAM_SIZE 1024     // A RAM possui 256 words dividida em 1024 bytes (4 bytes por
word)
#define SETS_L1 4         // Quantidade de conjuntos para cache L1
#define SETS_L2 8         // Quantidade de conjuntos para cache L2
#define BLOCKS_L1 2       // Quantidade de blocos para cache L1
#define BLOCKS_L2 2       // Quantidade de blocos para cache L2
#define WORDS_L1 2        // Quantidade de palavras da cache L1
#define WORDS_L2 4        // Quantidade de palavras da cache L2


typedef struct {
  int valid;              // bit de validade
  int subst;              // bit de controle para politica de substituicao
  int tag;                // tag do bloco
  unsigned int words[WORDS_L1];    // palavras do bloco
} block_L1;

typedef struct {
  int valid;              // bit de validade
  int modified;           // bit de modificado
  int subst;              // bit de controle para politica de substituicao
  int tag;                // tag do bloco
  unsigned int words[WORDS_L2];    // palavras do bloco
} block_L2;

typedef struct {
  block_L1 blocks[BLOCKS_L1];
} set_L1;

typedef struct {
  block_L2 blocks[BLOCKS_L2];
} set_L2;

unsigned char memory[RAM_SIZE];    // char representa 1 byte de 8 bits (desconsidera bit de
sinal)
set_L1 cache_L1[SETS_L1];
set_L2 cache_L2[SETS_L2];

int memory_access(int addr, int *data, int type);

#endif
```

========== CÓDIGO COM A FUNÇÃO *MEMORY_ACCESS()*-[memory_manager.c] ==========

```c
/*
 * ATENCAO: NAO ALTERAR OS ARQUIVO "main.c" e "memory_manager.h"
 * Seu trabalho deve ser desenvolvido neste arquivo "memory_manager.c"
 * na funcao memory_access(int addr, int *data, int type).
 *
 * Para compilar o projeto execute o comando
 *     make
 * Para executar digite o comando:
 *    ./main
 *
 * */

#include <stdlib.h>
#include "memory_manager.h"

int memory_access(int addr, int *data, int type) {
  /* seu trabalho comeca aqui :)  */
}
```

========== CÓDIGO COM A FUNÇÃO *MAIN() - [main.c]* ==========

```c
/*
 * ATENCAO: NAO ALTERAR ESTE ARQUIVO
 * Seu trabalho deve ser desenvolvido no arquivo memory_manager.c
 * na funcao memory_access(int addr, int *data, int type).
 *
 * Para compilar o projeto execute o comando
 *     make
 * Para executar digite o comando:
 *    ./main
 *
 * */

#include <stdio.h>
#include <stdlib.h>
#include "memory_manager.h"

void fill_memory() {
  int byte_addr;
  for (byte_addr = 0; byte_addr < RAM_SIZE; byte_addr++) {
    memory[byte_addr] = 255;
  }
}

void fill_cache_l1() {
  int set;
  int block;
  int word;
  for (set = 0; set < SETS_L1; set++) {
    for (block = 0; block < BLOCKS_L1; block++) {
      cache_L1[set].blocks[block].valid = 0;
      cache_L1[set].blocks[block].subst = 0;
      cache_L1[set].blocks[block].tag = 0;
      for (word = 0; word < WORDS_L1; word++) {
        cache_L1[set].blocks[block].words[word] = 0;
      }
    }
  }
}

void fill_cache_l2() {
  int set;
  int block;
  int word;
  for (set = 0; set < SETS_L2; set++) {
    for (block = 0; block < BLOCKS_L2; block++) {
      cache_L2[set].blocks[block].valid = 0;
      cache_L2[set].blocks[block].subst = 0;
      cache_L2[set].blocks[block].tag = 0;
      for (word = 0; word < WORDS_L2; word++) {
        cache_L2[set].blocks[block].words[word] = 0;
      }
    }
  }
}

/*
 * Assumindo que estamos utilizando big endian
 * o byte mais significativo eh o byte 0 e o
 * menos significativo eh o byte 3.
 */
unsigned int byte_to_int(unsigned char bytes[]) {
  return (unsigned int)((unsigned int)bytes[0] << 24) + ((unsigned int)bytes[1] << 16) +
    ((unsigned int)bytes[2] << 8) + ((unsigned int)bytes[3]);
}
```

```
void print_memory() {
  int byte_addr;
  ;
  int word_addr;
  int block;
  int unsigned content;

  printf("------------ MEMORIA -------------\n");
  printf("Block\tWord\tByte\tContent\n");
  printf("---------------------------------\n");

  for (byte_addr = 0; byte_addr < RAM_SIZE; byte_addr = byte_addr + 4) {
    unsigned char word_byte[4] = { memory[byte_addr], memory[byte_addr + 1],
      memory[byte_addr + 2], memory[byte_addr + 3]};

    word_addr = byte_addr/4;
    content = byte_to_int(word_byte);

    if (word_addr % WORDS_L2 == WORDS_L2/2) {
      block = word_addr / WORDS_L2;
      printf("%d\t%d\t%d\t%10u\n", block, word_addr, byte_addr, content);
    } else {
      printf("\t%d\t%d\t%10u\n", word_addr, byte_addr, content);
      if (word_addr % WORDS_L2 == WORDS_L2 - 1) {
        printf("---------------------------------\n");
      }
    }
  }
}

void print_cache_l1() {
  int set;
  int block;
  int word;
  printf("%53s\n","CACHE L1");
  printf("|-------------------- BLOCK 0 --------------------|");
  printf("|-------------------- BLOCK 1 --------------------|\n");
  printf("| Set | v | subst | tag |     w0     |     w1     |");
  printf("| Set | v | subst | tag |     w0     |     w1     |\n");
  for (set = 0; set < SETS_L1; set++) {
    for (block = 0; block < BLOCKS_L1; block++) {
      int valid = cache_L1[set].blocks[block].valid;
      int subst = cache_L1[set].blocks[block].subst;
      int tag = cache_L1[set].blocks[block].tag;
      printf("|  %d  | %d |   %d   | %03d | ", set, valid, subst, tag);
      for (word = 0; word < WORDS_L1; word++) {
        int w = cache_L1[set].blocks[block].words[word];
        printf("%10u |", w);
      }
    }
    printf("\n");
  }
  printf("|-------------------------------------------------|");
  printf("|-------------------------------------------------|\n");
}

void print_cache_l2() {
  int set;
  int block;
  int word;
  printf("%65s\n","CACHE L2");
  printf("|------------------------ BLOCK 0 ------------------------|");
  printf("|------------------------ BLOCK 1 ------------------------|\n");
  printf("| Set | v | m | subst | tag |   w0   |   w1  |   w2  |   w3  |");
```

```
  printf("| Set | v | m | subst | tag |   w0   |   w1  |   w2  |   w3  |\n");
  for (set = 0; set < SETS_L2; set++) {
    for (block = 0; block < BLOCKS_L2; block++) {
      int valid = cache_L2[set].blocks[block].valid;
      int modified = cache_L2[set].blocks[block].modified;
      int subst = cache_L2[set].blocks[block].subst;
      int tag = cache_L2[set].blocks[block].tag;
      printf("|  %d  | %d | %d |   %d   | %03d | ", set, valid, modified, subst, tag);
      for (word = 0; word < WORDS_L2; word++) {
        int w = cache_L2[set].blocks[block].words[word];
        printf("%6u |", w);
      }
    }
    printf("\n");
  }
  printf("|-------------------------------------------------------------|");
  printf("|-------------------------------------------------------------|\n");
}

int main(int argc, char *argv[]) {
  fill_memory();
  fill_cache_l1();
  fill_cache_l2();

  int exit = 0;
  int addr;
  int data;
  int type;
  int sucess;
  while (!exit) {
    printf("\n\nEndereco de acesso (byte) (-1 encerra o programa): ");
    scanf("%d", &addr);
    printf("\n");

    if (addr == -1) {
      exit = 1;
      continue;
    }

    if (addr < 0 || addr > 255) {
      printf("Endereco invalido (0-255).\n");
    }

    printf("Informe o tipo do acesso ( 0 - Leitura, 1 - Escrita): ");
    scanf("%d", &type);
    printf("\n");

    if (type != 0 && type != 1) {
      printf("Tipo invalido.");
      continue;
    }

    if (type == 1) {
      printf("Informe o dado a ser escrito: ");
      scanf("%d", &data);
      printf("\n");
    }

    sucess = memory_access(addr, &data, type);

    if (sucess == 1) {
      if (type == 1) printf("O dado %d foi inserido na posicao %d com sucesso.\n", data,
addr);
      else printf("O dado do endereco %d eh %d\n", addr, data);
    } else printf("Nao foi possivel realizar o acesso a memoria.\n");
```

```
  }

  print_memory();
  print_cache_l1();
  print_cache_l2();
}
```