Second Lab Assignment: TLB Cache Simulator

STUDENTS' IDENTIFICATION:

Number:	Name:
ist1110633	Filipe Oliveira
ist1110720	Francisco Andrade

1 Introdução

Este documento descreve a implementação de um simulador de Translation Lookaside Buffer (**TLB**) de dois níveis, concebido para acelerar o processo de tradução de endereços virtuais para físicos num sistema de memória paginada. O simulador, implementado no ficheiro tlb.c, modela uma hierarquia de caches composta por um TLB de Nível 1 (L1) e um TLB de Nível 2 (L2). Ambos os níveis são totalmente associativos, o que significa que qualquer tradução pode ser armazenada em qualquer posição da cache. A política de substituição de entradas é a **LRU** (Least Recently Used), que remove a entrada acedida há mais tempo. A política de escrita é **write-back**, onde as modificações (escritas) são mantidas na cache e apenas propagadas para o nível seguinte (L2 ou memória principal) quando a entrada é substituída ou invalidada.

2 Estruturas de Dados e Componentes Principais

2.1. Estrutura da Entrada da TLB (tlb_entry_t)

Cada entrada na TLB armazena a informação essencial para a tradução e gestão da cache. **valid** é um bit que, quando false, significa que a entrada está livre e pode ser usada para uma nova tradução sem necessidade de evicção. **dirty** é essencial para a política de write-back e é definido como true quando ocorre uma operação de escrita na página. **last_access** armazena o tempo do último acesso, obtido através da função get_time(). **virtual_page_number** e **physical_page_number** são o par que constitui o núcleo da tradução de endereços.

2.2. Funções Auxiliares Criadas

- **find_entry():** Procura uma entrada numa cache (L1 ou L2) com base no vpn. Retorna um ponteiro para a entrada se a encontrar, ou NULL caso contrário.
- **evict_entry():** Implementa a política de substituição LRU. Primeiro, procura uma entrada inválida (valid == false). Se não encontrar, percorre a cache para encontrar a entrada com o last_access mais antigo. Ao remover uma entrada dirty do L1, propaga o estado dirty para a entrada correspondente no L2.
- **insert_entry():** Combina a lógica de evicção e inserção. Chama evict_entry() para obter uma posição livre e preenche-a com os dados da nova tradução.

3 Fluxo de Tradução de Endereços (tlb_translate)

Cenário 1: Acerto no TLB L1 (L1 Hit)

O **vpn** é extraído do endereço virtual. A função **find_entry()** é chamada para procurar o **vpn** no **tlb_l1**. Se a entrada for encontrada, o contador **tlb_l1_hits** é incrementado. O **last_access** da entrada L1 e da sua correspondente no L2 é atualizado para o tempo atual, mantendo a consistência da política LRU em ambos os níveis. Se a operação for de escrita (**OP_WRITE**), o bit **dirty** é definido como true em ambas as entradas. O endereço físico é calculado e retornado imediatamente.

```
tlb_entry_t *e = find_entry(tlb_l1, TLB_L1_SIZE, vpn);
if (e)
{
  tlb_l1_hits++;

  e->last_access = get_time();
  if (OP_IS_WRITE(op))
    e->dirty = true;

  // Atualiza o L2 para consistência LRU e do bit dirty
  tlb_entry_t *l2_e = find_entry(tlb_l2, TLB_L2_SIZE, vpn);
  if (l2_e)
  {
    l2_e->last_access = get_time();
    if (OP_IS_WRITE(op))
      l2_e->dirty = true;
  }

  return (pa_dram_t)((e->physical_page_number << PAGE_SIZE_BITS) | offset);
}
```

Cenário 2: Falha no L1, Acerto no L2 (L1 Miss, L2 Hit)

Se a tradução não está no L1, o próximo passo é procurar no L2. O contador **tlb_l1_misses** é incrementado. **find_entry()** é usado para procurar o **vpn** no **tlb_l2**. Se encontrado, o contador **tlb_l2_hits** é incrementado. A entrada é promovida para o L1: **evict_entry()** é chamada para o **tlb_l1** para encontrar uma vítima. Se a vítima do L1 estiver **dirty**, o seu estado é propagado para a entrada correspondente no L2. A entrada encontrada no L2 é copiada para a posição da vítima no L1. O **last_access** de ambas as entradas (a nova no L1 e a original no L2) é atualizado. O bit **dirty** da entrada L2 é atualizado se a operação for de escrita. A nova entrada L1 herda o estado **dirty** do L2 ou torna-se **dirty** se a operação atual for uma escrita. O endereço físico é calculado e retornado.

```
e = find_entry(tlb_l2, TLB_L2_SIZE, vpn);
if (e)
{
  tlb_l2_hits++;
  e->last_access = get_time();

  if (OP_IS_WRITE(op))
    e->dirty = true;

  pa_dram_t pa_from_l2 = e->physical_page_number << PAGE_SIZE_BITS;

  /* Promove a entrada para L1 (Manual Insertion) */
  tlb_entry_t *l1_e = evict_entry(tlb_l1, TLB_L1_SIZE);
  l1_e->valid = true;
  l1_e->dirty = e->dirty || OP_IS_WRITE(op);
  l1_e->virtual_page_number = vpn;
  l1_e->physical_page_number = pa_from_l2 >> PAGE_SIZE_BITS;
  l1_e->last_access = get_time();

  log_dbg("Promoted VPN 0x%" PRIx64 " -> PFN 0x%" PRIx64 " into L1%s",
          vpn, l1_e->physical_page_number, l1_e->dirty ? " (dirty)" : "");


  return (pa_dram_t)((e->physical_page_number << PAGE_SIZE_BITS) | offset);
}
```

Cenário 3: Falha em L1 e L2 (L1 Miss, L2 Miss)

Quando a tradução não existe em nenhuma das caches, é necessário consultar a Tabela de Páginas. Os contadores **tlb_l1_misses** e **tlb_l2_misses** são incrementados. A função **page_table_translate()** é chamada. Após obter a tradução (o endereço físico **pa**), a nova entrada (**vpn --> pfn**) é inserida em ambas as caches: **insert_entry()** é chamada para o **tlb_l2**. Esta função internamente trata da evicção de uma entrada LRU do L2, se necessário. **insert_entry()** é chamada para o **tlb_l1**, garantindo que a tradução mais recente esteja disponível na cache mais rápida. O endereço físico final é retornado.

```
pa_dram_t pa = page_table_translate(virtual_address, op);
pa_dram_t page_aligned_pa = pa & ~PAGE_OFFSET_MASK;

// Insere a nova tradução no L2 e L1
insert_entry(tlb_l2, TLB_L2_SIZE, vpn, page_aligned_pa, op);
insert_entry(tlb_l1, TLB_L1_SIZE, vpn, page_aligned_pa, op);

log_dbg("Address returned from Page Table: 0x%" PRIx64 "\n", pa);
return pa;
```

4 Invalidação de Entradas (tlb_invalidate)

A função **tlb_invalidate** é chamada pelo sistema operativo quando uma página é removida da memória física (por exemplo, para ser movida para o disco). A sua função é remover a tradução de ambas as caches para evitar o uso de um mapeamento inválido. Percorre o **tlb_l1** em busca da entrada com o **virtual_page_number** especificado. Se a encontra, define o seu bit **valid** como false e incrementa o contador **tlb_l1_invalidations**. Repete o processo para o **tlb_l2**, invalidando a entrada correspondente e incrementando **tlb_l2_invalidations**.

```
void tlb_invalidate(va_t virtual_page_number)
{
  increment_time(TLB_L1_LATENCY_NS);
  for (int i = 0; i < TLB_L1_SIZE; i++)
  {
    if (tlb_l1[i].valid && tlb_l1[i].virtual_page_number == virtual_page_number)
    {
      if (tlb_l1[i].dirty)
      {
        tlb_l1[i].dirty = false;
      }
      tlb_l1[i].valid = false;
      tlb_l1_invalidations++;
    }
  }

  increment_time(TLB_L2_LATENCY_NS);
  for (int i = 0; i < TLB_L2_SIZE; i++)
  {
    if (tlb_l2[i].valid && tlb_l2[i].virtual_page_number == virtual_page_number)
    {
      if (tlb_l2[i].dirty)
      {
        tlb_l2[i].dirty = false;
      }
      tlb_l2[i].valid = false;
      tlb_l2_invalidations++;
    }
  }
}
```