Отчет

Лабораторная работа №3. Кэш

Автор: Халили Алина Ниязовна, M3136

Язык: Python (3.8)

Вариант: 3

Репозиторий https://github.com/skkv-itmo2/itmo-comp-arch-2023-cache-AlinaK12345

4)Результаты программы:

LRU: hit perc. 99.9335% time: 2897205.0

pLRU: hit perc. 99.9335% time: 2897205.0

5)Расчет параметров:

MEM_SIZE	1048576	2**ADDR_LEN
ADDR_LEN	20 бит	
Конфигурация кэша	look-through write-back	
Политика вытеснения кэша	LRU и bit-pLRU	
CACHE_WAY	4	
CACHE_TAG_LEN	9 бит	
CACHE_IDX_LEN	4	ADDR_LEN - CACHE_TAG_LEN - CACHE_OFFSET_LEN
CACHE_OFFSET_LEN	7	Log2(CACHE_LINE_SIZE)
CACHE_SIZE	8192	CACHE_LINE_SIZE * CACHE_WAY * CACHE_SETS_COUNT
CACHE_LINE_SIZE	128 байт	
CACHE_LINE_COUNT	64	CACHE_SETS_COUNT * CACHE_WAY
CACHE_SETS_COUNT	16	2**CACHE_IDX_LEN

Описание работы кода:

Реализовано LRU и pLRU.

В модели: храню в кэше только теги.

-LRU: для каждой линии хранится ее время от 0 – 3 (так как ассоциативность 4).
3 – у самой недавней по обращению, 0 – у самой давней. (изначально все 0)
`time_used_LRU`

-pLRU: хранится 1 бит актуальности, меняется на 1, если недавно обращались к линии. Когда все = 1, все (из 4) меняются на 0, кроме последнего обращения.
`time_used_bitLRU`

`Write-back, look-throw.` Для реализации храню число для каждой линии: 0 – значение хранится в оперативке | 1 – значения нет в оперативке. | -1 – еще ни разу не использовалась.

```
cache_lable = []
cache_lable_bit = []
```

Функции почти одинаково реализованы для LRU и pLRU.

address = [tag, ind, offset], выделяем tag, ind

-Read(адрес):

-Если есть совпадение в блоке ассоциативности, считаю такты, попадание, и возвращаю значение +пересчитываю время актуальности

-Промах:

-ищу минимальное по времени обращение

-если lable = -1 или 0, ищем в оперативке и записываем в кэш, lable = 0

-lable = 1, тогда сохраняем значение в оперативку, ищем там нужное и сохраняем на то же место в кэше, lable = 0

+пересчитываю время актуальности

-Write(адрес):

-Попадание:

- записываем тег в кэш, lable = 1 +пересчитываю время актуальности

-Промах:

- ищем мин по времени, если lable = 0 | -1, измением в кэше, lable =1

-lable = 1, записываем старое значение в память, меняем значение в кэше, lable = 1.

+пересчитываю время актуальности

Такты считаются:

-запрос: если без передачи данных, то +1 на передачу запроса

`Если с данными, то(cache <-> MEM) +CACHE_LINE_SIZE*8/DATA2_BUS_LEN)`
`(CPU <-> cache) – считаем в задаче, размер данных / DATA1_BUS_LEN`

```
ADDR1_BUS_LEN = ADDR_LEN
ADDR2_BUS_LEN = ADDR_LEN
DATA1_BUS_LEN = 16
DATA2_BUS_LEN = 16
CTR1_BUS_LEN = 3 # 8 различных команд
CTR2_BUS_LEN = 2 #cache <-> memory , 4 команды
```

Моделирование задачи:

```
clock_common = 0

M = 64
N = 60
K = 32
clock_common += 3

PA = 0x40000
PB = M*K + PA
PC = PB + K*N*2

clock_common+=1 #pa
clock_common+=1 # pc

clock_common += 1 #y
for y in range(M):
    clock_common += 1 # y++

    clock_common += 1 #x
    for x in range(N):
        clock_common += 1 # x++

        clock_common += 1 #pb
        clock_common += 1 #s

        clock_common += 1 #k
        for k in range(K):
            clock_common += 1 # k++

            clock_common += 1 #делаем запрос в кэш(адрес передаем)
            read_bitLRU(y*K + k + PA) # pa[k]
            read_LRU(y*K + k + PA) # pa[k]
            count_all += 1
            clock_common += 1 #8 // DATA1_BUS_LEN округлям к 1 -принимаем
данные


            clock_common += 1 #делаем запрос в кэш(адрес передаем)
            read_LRU(k*N*2 + x*2 + PB) # pb[k]
            read_bitLRU(k*N*2 + x*2 + PB) # pb[k]
            count_all += 1
            clock_common += 1 #16 // DATA1_BUS_LEN

            clock_common += 5 # *
            clock_common += 1 # +

        clock_common += 2 # 32/DATA1_BUS_LEN
        write_LRU((y*N + x)*4 + PC) # pc
        write_bitLRU(4*(y*N + x) + PC) # pc
        count_all += 1
        clock_common += 1 #обратно передавать CPU

    clock_common += 1 # pa+=k
    clock_common += 1 #pc+=n

clock_common += 1 #выход из ф-ии

clock_bit += clock_common
clock += clock_common
```