Лабораторная работа №3	ФИО: Кинзябулатов Эдуард Шамилевич
Название работы: Кэш	Группа: M3137

Цель работы: моделирование системы “процессор-кэш-память”

Ссылка на репозиторий: https://github.com/skkv-itmo/itmo-comp-arch-2023-cache-Tortik3000

Инструментарий и требования к работе: Python (3.11.6)

Результат работы:

LRU: hit perc. 99.125% time: 3782274

pLRU: hit perc. 99.4046% time: 3707110

RR: hit perc. 98.9627% time: 3825684

Кэш

MEM_SIZE	2^16 байт
ADDR_LEN	16 бит
Конфигурация кэша	look-through write-back
Политика вытеснения кэша	LRU, bit-pLRU, Round-robin
CACHE_WAY	4
CACHE_TAG_LEN	6 бит
CACHE_IDX_LEN	5 бит
CACHE_OFFSET_LEN	5 бит
CACHE_SIZE	2^12 байт
CACHE_LINE_SIZE	32 байт
CACHE_LINE_COUNT	128
CACHE_SETS_COUNT	32

MEM_SIZE=2^ADDR_LEN

CACHE_INDX_LEN = log(CACHE_LINE_SIZE)

CACHE_OFFSET_LEN = log(CACHE_LINE_SIZE)

CACHE_SIZE = CACHE_LINE_COUNT * CACHE_LINE_SIZE

CACHE_LINE_COUNT = CACHE_WAY * CACHE_SETS_COUNT

CACHE_TAG_LEN = ADDR_LEN - CACHE_IDX_LEN – CACHE_OFFSET_LEN

Размерность шин

Шина	Обозначение	Размерность
A1, A2	ADDR1_BUS_SIZE ADDR2_BUS_SIZE	16 бит 11 бит
D1, D2	DATA1_BUS_SIZE DATA2_BUS_SIZE	16 бит 32 ,bn
C1, C2	CTR1_BUS_SIZE CTR2_BUS_SIZE	3 бита 2 бита

ADDR1_BUS_SIZE = ADDR_LEN

ADDR2_BUS_SIZE = CACHE_TAG_LEN + CACHE_IDX_LEN

8 команд(read8, read16, read32, write8, write16, write32, response, non) CTR1_BUS_SIZE = 3 бита

4 команды(read, write, response, non) CTR2_BUS_SIZE = 2 бита

Кэш – look-through. В этой системе процессор и память не связаны, когда процессор подает запрос в кэш, если кэш-попадание, то кэш отправляет ответ, иначе отправляет запрос в память.

Write-back – измененные данные записываются сначала в кэш, а при вытеснении из кэша записываются в память.

Кэш в коде представлен массивом блоков линий.

Блоки содержат CACHE_WAY линий, линии состоят из адреса и служебной информации(flag)

Для LRU и pLRU flag состоит из 3-х ячеек:

Если flag[0]=0 эта кэш линия свободна, иначе занята

flag[1] отвечает за возраст

если flag[2](dirty)=0 то линия содержит такие же данные, что mem

Для RR flag не содержит ячейку для возраста

```
cache = []
flag = 2

for i in range(CACHE_LINE_COUNT // CACHE_WAY):
    block = []
    for j in range(CACHE_WAY):
        block.append([0] * (ADDR_LEN + flag))
    cache.append(block)
```

Когда от процессора поступает команда, тогда из адресса извлекается тэг и индекс блока:

```
2 usages
def readLRU(adr, bytes):
    global hit, miss, requests, tact
    requests += 1
    tact += 1  # addr, command
    tag = adr[:CACHE_TAG_LEN]
    index = adr[CACHE_TAG_LEN:CACHE_IDX_LEN + CACHE_TAG_LEN]
    index = list(map(str, index))
    indexBlock = int("".join(index), 2)
```

Когда процессор передает команду чтения, сначала идет проверка на наличие линии с нужным тэгом в кэше, если кэш попадание, то передаем процессору нужные байты:

```
for indexLine in range(CACHE_WAY):
    line = cache[indexBlock][indexLine]
    if (line[0] == 1 and tag == line[flag:CACHE_TAG_LEN + flag]):
        tact += 7 + bytes  # hit + response + bytes
        hit += 1

        timeLRU(indexBlock, indexLine)
        return
```

Иначе в зависимости от политики вытеснения выбираем линию для замещения

и если флаг dirty = 1, то сначала передаем это линию на запись в память, потом из памяти в кэш записывается нужная кэш-линия:

```
# read from mem
tact += 4  # miss
if (cache[indexBlock][CACHE_WAY - 1][2] == 1):
    tact += 1  # command
    tact += 100  # write in mem
    tact += CACHE_LINE_SIZE // 4  # transfer line
    tact += 1  # response

tact += 1  # command
tact += 100  # write in cache
tact += CACHE_LINE_SIZE // 4
tact += 1  # response

tact += bytes + 1  # transfer byte + response

cache[indexBlock][CACHE_WAY - 1] = [1, 0, 0] + adr
timeLRU(indexBlock, CACHE_WAY - 1)
```

При записи, сначала идет проверка на наличие линии с нужным тэгом, если такая найдена

то в нее записываются измененные байты и значение флага dirty = 1:

```
for indexLine in range(CACHE_WAY):
    line = cache[indexBlock][indexLine]
    if (line[0] == 1 and tag == line[flag:CACHE_TAG_LEN + flag]):
        # replace line
        tact += 7  # hit + response
        hit += 1
        cache[indexBlock][indexLine] = [1, 0, 1] + adr
        timeLRU(indexBlock, indexLine)
        return
```

Иначе в зависимости от политики вытеснения выбираем линию для замещения и если флаг dirty=1, то сначала передаем это линию на запись в память, потом из памяти в кэш записывается нужная кэш-линия, потом в нее записываются переданные байты и флаг dirty=1 :

```
    tact += 4  # miss

    if (cache[indexBlock][CACHE_WAY - 1][2] == 1):
        tact += 1  # command
        tact += 100  # write in mem
        tact += CACHE_LINE_SIZE // 4  # transfer line
        tact += 1  # response

    tact += 1  # command
    tact += 100  # write in cache
    tact += CACHE_LINE_SIZE // 4  # transfer line
    tact += 1  # response

    tact += 1  # response

    cache[indexBlock][CACHE_WAY - 1] = [1, 0, 1] + adr
    timeLRU(indexBlock, CACHE_WAY - 1)
```

Реализация LRU:

После обращения к кэш-линии ее возраст становится равным нулю а остальные возраста в блоке увеличиваются на 1. При необходимости замещения линии в кэше заменяется линия с наибольшим возрастом.

```
def timeLRU(indexBlock, indexLine):
    block = cache[indexBlock]
    for line in block:
        line[1] += 1
    block[indexLine][1] = 0
    block.sort(key=lambda x: x[1])
    for i in range(CACHE_WAY):
        block[i][1] = i
```

Реализация pLRU:

После обращения к кэш-линии ее бит возраста становится равным единице и если количество

кэш-линий в блоке с битом возраста равным единице равно CACHE_WAY, то все остальные биты возраста переходят в 0. При необходимости замещения линии в кэше заменяется первая линия с возрастом 0.

```
def timePLRU(indexBlock, indexLine):
    block = cache[indexBlock]
    count = 0
    for line in block:
        if (line[1] == 1):
            count += 1
    if count == CACHE_WAY:
        for line in block:
            line[1] = 0
    block[indexLine][1] = 1
```

Реализация RR:

При записи линии в кэш блок , линия записывается в первую ячейку блока, потом все ячейки циклически сдвигаются:

```
def timeRound_Robin(indexBlock):
    block = cache[indexBlock]
    tent = block[0]
    block[0] = block[1]
    block[1] = block[2]
    block[2] = block[3]
    block[3] = tent
```