Лабораторная работа №3	22.Б05	2023
Кэш	Назаров Матвей Антонович	

Инструментарий и требования к работе: работа выполняется на Python (3.11.5).

Ссылка на репозиторий: https://github.com/skkv-mkn/mkn-comp-arch-2023-cache-lomalovo

Расчёт констант:
Дано: MEM_SIZE = 512 Кбайт
CACHE_TAG_LEN = 10 бит
CACHE_LINE_SIZE = 32 байт
CACHE_LINE_COUNT = 64
Теперь найдем:
ADDR_LEN = 19 бит(Размер памяти это 2^19 байт, а значит адрес занимает 19 бит)
CACHE_SIZE = 32 * 64 байт (Произведение количества линий на длину одной)
CACHE_OFFSET_LEN = 5 бит (логарифм от CACHE_LINE_SIZE)
CACHE_IDX_LEN = 4 бит (19 – 10 – 5 Из общей длины вычитаем tag и offset)
CACHE_SETS_COUNT = 16 (Это просто 2 в степени CACHE_IDX_LEN)
CACHE_WAY = 4 (Это CACHE_LINE_COUNT / CACHE_SETS_COUNT)

Размерность шины адреса A1 мы уже посчитали: 19 бит
Размерность шины адреса A2 = CACHE_TAG_LEN + CACHE_IDX_LEN = 14 бит
Размерность шины команд: шина C1 должна кодировать 7 команд, а значит ее размерность – 3 бита C2 должна кодировать 3 команды, а значит размерность 2 бита

Теперь перейдем к описанию работы программы:

Кэш реализован в class CpuCacheMem, при создании представителя я ввожу константы из условия, а также в качестве переменной передаю режим работы кэша: “LRU” или “pLRU”.

Кроме того ввожу массив self.cache tags, который отображает данные в кэше, а также cache_times, в котором для каждых данных будет записано их время, которое я буду использовать в дальнейшем для удаления. Также

есть self.modified_tags, который отображает была ли модифицирована линия внутри кэша или она совпадает с линией в памяти (Это будет нужно в дальнейшем для того, чтобы не переписывать лишний раз данные из кэша в память, если это не нужно) Еще ввожу переменные для подсчета запросов, промахов и тактов соответственно: self.requests, self.misses, self.tact

Работа с кэшем реализована через функцию класса

def request(self, address, size, command)

В нее мы передаем адрес начала данных, их размер, команду “R” или “W”. Сначала мы считаем необходимое количество тактов, необходимое для передачи данных и кэш линии: **DTact** = max(1, size // self.SPEED) **CacheTac**t = max(1, self.CACHE_LINE_SIZE * 8 // self.SPEED)

Далее я вычисляю tag и index из address:

tag = address >> (self.CACHE_OFFSET_LEN+self.CACHE_IDX_LEN)
index = (address >> self.CACHE_OFFSET_LEN)%
(2**self.CACHE_IDX_LEN)

Далее проверяю есть ли tag в кэше или нет:

1)Кэш хит, тогда сразу посчитаем такты. У нас два варианта: 1) Команда “R”, тогда мы сначала отправляем команду в кэш за 1 такт, 6 тактов кэш отвечает и Dtact времени отправляет обратно. 2) Команда “W”, мы отправляем данные в кэш за Dtact, далее 6 тактов кэш отвечает, а далее кэш отправляет ответ в процессор еще за 1 такт. В обоих случаях мы увеличиваем такты на 7+Dtact.

Теперь мы обновляем время (нужное для LRU и pLRU), для этого я использую функцию def upgrade_time(self, index, pos). Она в зависимости от режима: 1)LRU тогда она для всех остальных tag в нашем set увеличивает время на 1, а в нашем ставит 0. 2)pLRU ставит время 1, а потом проверяет не оказалось ли так, что все стали единичками, если так, то все, кроме нашей позиции становятся 0.

В конце обработки случая Кэш попадания мы, если была команда W, помечаем наш tag как исправленный

2)Кэш промах. Посчитаем такты: 1) Пусть команда “R”. Отправляем запрос в кэш 1 такт, происходит кэш-промах 4 такта, далее кэш смотрит, если у него есть свободное место по данному индексу, то 1 такт отправляет запрос в память, иначе он отдельно записывает кэш линию в память. Память отвечает 100 тактов, отправляет кэш линию в кэш еще за CacheTact(16)

тактов. Далее данные передаются в процессор за DTact. 2)Теперь команда “W”. Сначала мы за DTact отправляем данные в кэш, происходит кэш-промах 4 такта, далее кэш смотрит, если у него есть свободное место по данному индексу, то 1 такт отправляет запрос в память, иначе он отдельно записывает кэш линию в память. Память отвечает 100 тактов, отправляет кэш линию в кэш еще за CacheTact(16) тактов. 1 такт мы отвечаем процессору.

Теперь реализация кэш промаха. Сначала, в зависимости от режима, мы находим место, в которое мы будем писать. Меняем значение в найденном месте на tag, не забываем upgrade_time и изменить modified_tags. Также как раз тут я обрабатываю, нужно ли нам перезаписывать данные из кэша в память, если по нашему месту modified_tags это 1, то я, как объяснял раньше мы за CacheTact(16) + 100 + 1 такт отправляем кэш линию в память

Теперь рассмотрим симуляцию, она реализована в

def simulation(mode)

Сначала я ввожу параметры, отображающие такты необходимые для одной или той операции. Потом я создаю наши массивы a, b, c, в которые я вместо данных пишу адреса соответственных клеток массива. Давайте считать такты. Сначала инициализируем 2 указателя – 2 такта, потом на каждый for мы инициализируем переменную – 1 такт (вне for) потом мы 1 такт инкрементируем переменную for, 1 такт итерируемся. Внутри второго for мы инициализируем еще один указатель и сумму – 2 такта. Внутри третьего for мы делаем 2 сложения и умножения – 7 тактов. Делаем кэш запросы, а в конце первого for увеличиваем 2 указателя – 2 такта, завершаем функцию – 1 такт.

В результате работы и такого подсчета тактов я получил:

LRU: hit perc. 96.6571% time: 4144144

pLRU: hit perc. 96.6406% time: 4149093