Questions on Computer Architecture for students

Caches/Memory - I

Prerequisite questions

1. Почему в современных процессорах используется кэш для основной памяти?
2. Скажите, каким свойствам желательно, чтобы удовлетворяли запросы (access patterns) в память, чтобы кэш использовался наиболее эффективно?
3. Скажите, Вы знакомы с тем как устроены и как организуются кэши (cache organizations) в современных процессорах? К примеру, можете сказать чем отличаются direct-mapped cache (кэш прямого отображения) и set-associative кэши?
4. (Если проблемы с предыдущим вопросом) Можете рассказать в принципе, что Вы знаете об устройстве и работе кэшей в микропроцессорах?

Main questions

1. Допустим, мы работаем с set-associative кэшом. (*На какие группы бит разбивается адрес обращения в памяти?*) Как адрес обращения в память обычно разбивается на группы бит, соответствующие полям set, tag и offset? (обязательно уточнить про то, где старшие биты, а где младшие биты в адресе)
2. *Follow-up question: А почему именно так располагают поля в адресе? Что будет происходить, если к примеру поменять set и tag местами?*

   Answer: Адрес делится на битовые группы, соответствующие tag, set и offset. |tag|set|offset| от старшего бита адреса к младшему. Если B - размер кэш-блока в байтах, то размер поля offset будет [log2(B)]. Разделение происходит именно таким образом, чтобы кэш-линии с адресами A, A + B, A + 2 * B и т.д. попадали в разные set-ы. В случае, если поменять set и tag местами на один set будут отображаться линейные блоки размером B * 2^(tag_size), что будет приводить к потере чувствительности к spatial locality, conflict misses, потере эффективности.

3. (Hard, need to show params on the screen) Предположим, что мы рассматриваем direct-mapped cache с политикой замещения LRU в микропроцессоре с 32-битной адресацией. Будем считать, что это L1 кэш данных. Размер кэш-линии в данном кэше составляет 64 байта. При этом, при разработке процессора что-то пошло не так, и поля set и tag поменялись местами в адресе обращения (т.е. |set|tag|offset| от старшего бита к младшему). Программист Вася написал следующий отрывок программы

```
(a) for (int i = 0; I < 100000; ++i)
        for (int j = 0; j < 1000; j += STEP)
            std::cout << a[j];
```

```
(b) auto a = new Bstruct[1000];
    auto end = a + 1000;
    for (int i = 0; i < 1000000; ++i)
        for (auto ptr = a; ptr != end; ptr++)
            std::cout << *(std::uint64_t*)ptr;
```

   (a) Можете привести пример какого-либо значения STEP,
   (b) Можете привести пример размера структуры Bstruct,
   при котором производительность процессора с данным кэшом с измененным порядком set и tag будет для этой программы лучше, чем со стандартным (неизмененным)?

   Help-1:

   Давайте начнем с того, что определим принципиальный момент, почему производительность программы с измененным порядком может быть лучше

   Help-2:

   Давайте попробуем написать общую формулу сначала для данной ситуации, не привязываясь к конкретным числам и посмотрим, каких параметров нам не хватает

   Help-3:

   Сколько бит в адресе будет приходиться на offset, а сколько на set + tag?

   Answer:

   - В данном случае главный access pattern LRU-friendly [A, A + K, A + 2 * K, ..., A + 1000 * K].
   - Кэш - direct-mapped, это значит, что кэш-линии, отображенные на один и тот же set будут гарантированно перезаписывать друг друга. Будем далее считать, что B - размер кэш-линии

в байтах, #sets - число set-ов.

- В случае нормального расположения битовых групп в один и тот же set будут попадать кэш линии с адресами A, A + #sets * B, A + 2 * #sets * B
- В случае измененного расположения битовых групп в разные set-ы будут попадать кэш-линии с адресами A, A + 2^(tag_size) * B, A + 2 * 2^(tag_size) * B.
- В таком случае должно быть 2^(tag_size) = #sets. Так как адрес составляет 32 бита и 6 битов отводится на offset, то 26 битов приходится на tag + set. А значит #sets = 2^(tag_size) = 2^13.
- Следовательно, STEP = sizeof(Bstruct) = 2^13 * 2^6 = 2^19 = 524,288 (байт) = (512 КБ)

4. (Вопрос, если на 1 вопрос был дан неправильный (и быстрый) ответ, но при этом OFFSET на правильном месте; может быть вместо 3)

   Дан кэш direct-mapped с размером кэш линии в 32 байта. Что будет выведено на экран? Что будет происходить с кэшом для данной программы (предполагается, что в ответе на 1 вопрос set и tag были поставлены наоборот)

```
struct A {
        std::uint32_t my_array[3];
        std::uint8_t value;
};

int j = 0;
for (int i = 0; i < 1000000; ++i)
    for (j = 0; j < 30; j += sizeof(A))
        std::cout << a[j];

std::printf("%d\n", j);
```

   Answer: 30 * sizeof(A) + уточнение про padding. Если ответ 600, то спросить всегда ли так. Будет перетирание внутри одного set, в первом вопросе должно быть |tag|set|offset|

Branch prediction - I

Prerequisite questions

1. Можете сказать, что такое branch prediction (предсказание переходов) и почему его используют в современных процессорах?
2. Скажите, а если у нас в системе команд процессора нет инструкций условного перехода (conditional branches), нужен ли нам branch predictor?
3. Можете рассказать принцип работы bimodal branch predictor (2-bit saturating counter)?
4. Вы знаете, что такое local branch predictor и global branch predictor, и в чем между ними отличие?

Main questions

1. Пусть рассматривается локальный (two-level adaptive predictor) branch predictor. Про него известно, что для каждого PC бранча мы храним полную локальную историю, по которой с помощью бимодального предиктора мы делаем предсказание. Считаем, что в начальном состоянии бимодальный предиктор находится в состоянии Strongly Not-taken. Размера бранч предиктора хватает, чтобы хранить информацию о всех бранчах, присутствующих в рассматриваемых программах.

```
for (int I = 0; I < 100; ++I)
        if (I % 2 == 0 || I % 7 == 0)
                std::cout << "Even\n";
```