

## всякое про КЭШ и конвейер

1. память стали делать таким образом, чтобы за одно обращение к ней она выдавала не по одному байту, а сразу по несколько байт с последовательными адресами. Для этого оперативную память разбивают на отдельные микросхемы, которые могут работать параллельно. Этот приём называют **расслоением памяти**
2. теневые регистры - регистры что использует процессор, к ним нет доступа
3. буфер команд - буфер в который загружаются команды целом блоком, что позволяет анализировать контроллеру написанное
4. множественность АЛУ - просто как факт АЛУ может быть несколько, тогда выполнение арифметических операций возможно одновременно
5. конвейерность устройств - определения
  - (a) latency - задержка выполнения машинной команды, это число машинных тактов, которые требуется для полного выполнения команды (включая такты ожидания).
  - (b) throughput - пропускная способность, количество независимых команд выполняемых одновременно
  - (c) время разгона конвейера - время которое требуется чтобы 1 команда прошла весь конвейер
6. причины прерывания конвейера (возникновение пузырей - ступор конвейера)
  - (a) данные на вход оказались неполными
  - (b) зависимость от дополнительных входов
7. Можно, например, выделить следующие основные шаги выполнения команды:
  - (a) Выбор команды
  - (b) Декодирование команды
  - (c) вычисление адресов операндов
  - (d) выбор операндов
  - (e) выполнение
  - (f) запись
8. КЭШ память - память внутри процессора причём она есть у каждого ядра есть общие для нескольких ядер. Отсюда возникает проблема согласованности Кэша для нескольких процессорных функций
9. L кэши разделяются по:
  - (a) Размер
  - (b) Назначение (L1|2|3)
  - (c) по организации отображения - прямое отображение LRU или ассоциативное отображение Full associative cache
10. КЭШ линия или Кэш строка - данные в кэше относящиеся к одному блоку данных
11. Промах - процессор не нашёл в кэше - промах
12. Попадание - процессор нашёл в кэше что хотел
13. Сквозная запись в кэше предполагает, что запись производится непосредственно в основную память (и дублируется в кэш), то есть запись не кэшируется. Преимущества такого подхода — быстрое получение, безопасность и полная согласованность данных. Недостатки — высокая нагрузка на кэш.
14. Отложенная запись в кэше предполагает, что запись данных производится в кэш, а запись в основную память производится позже (при вытеснении или по истечении времени). Преимущества — высокая скорость записи данных. Недостатки — риск потери данных в случае аварийного завершения работы приложения до записи данных в основное хранилище.

15. Прямое отображение - делим оперативную память на страницы их делим на блоки данных. в Кэше для каждой страницы есть место для 1 блока данных и его состояния, какой именно узнаётся благодаря префиксу. в случае попадания (мы ищем кэш линию соответственно страницы и видим, что номер блока данных совпадает) - заберём байт из КЭШа, при промахе (номер блока не совпал) мы обращаемся к памяти и записываем номер нового блока и данные в кэш линию.
16. Ассоциативное отображение - в отличии от LRU в кэш линии - адрес, состояния - invalid modified stored , счётчик (обращений), данные и его. при попадании счётчик увеличивается, при промахе заменяется кэш линия с наименьшим значением счётчика
17. Уровни кэшей - L1, L2, L3 - L1 у каждого ядра, L2 - у нескольких ядер одновременно, L3 - у всех один обращение происходит по иерархии
18. предсказатель ветвлений - При выполнении на конвейере команды условного перехода, делается предположение о том, выполнит ли эта команда переход или продолжится последовательное выполнение программы. В соответствие с этим предположением на конвейер выбираются команды из наиболее вероятной
19. Что такое спекулятивное выполнение? - При выполнении на конвейере команды условного перехода, на конвейер поступают (и выполняются) команды обеих ветвей (программы). Когда же становится известно, следует или нет выполнять переход, учитывается результат выполнения команд нужной ветви; результат выполнения команд второй ветви отбрасывается
20. Механизм реализации :
  - (a) на абум
  - (b) согласно таблице переходов в которой у каждого ветвления есть 2 поля: число вызовов и число успешных переходов по их отношению выбирается наиболее вероятное спекулятивное выполнение

## Трансляция команд

Сальников добавил:

Зная что такое контроллер управления дешифрует что то

код в ассемблере:

add byte[edx], 45

преобразуется контроллером В

mov  $тень_i$ , 45; константы подгружаем

load  $тень_j$ ,  $edx$ ;  $тень_j \leftarrow [edx]$

add  $тень_k$ ,  $тень_j$ ,  $тень_i$

store  $edx$ ,  $тень_k$

load - внутренняя инструкция загрузки, store -внутренняя инструкция выгрузки в память

вся арифметика - 3 операторная

в процессе счёта  $тень_i$  могут переименовываться в  $eax$ ,  $edx$ ,  $ebx$

другой пример: mov  $eax$ , X  $\rightarrow$  mov  $тень_i$ , X; load  $eax$ ,  $тень_i$ ; X - метка

## про прерывания

читать Баулу

Стандартный пролог и эпилог:

к сожалению не записал (надеюсь на святика)

вообще там чуть чуть написано в конспекте моём (последняя тема)

## инструкции с 34 и 35 вопроса

1. префикс lock - гарантирует эксклюзивный доступ к памяти на время выполнения инструкции. (никто кроме этой инструкции не сможет одновременно обратиться к памяти)

2. `clwb [mem]` - Cache Line Write Back записывает в память всю кэш-линию (обычно 64 байта), содержащую указанный адрес,
3. `clflush [mem]` - Cache Line Flush принудительно сбрасывающая содержимое кэш-линии в основную память и инвалидирующая (удаляющая) её из всех уровней кэша. Это ключевой инструмент для управления когерентностью кэша (согласованности кэша между ядрами) и работы с памятью, обходящей кэширование. аналогично делает это с линией по адресу
4. `sfence (store fence)` - Гарантирует, что все операции записи (`store`) в память, инициированные до барьера, завершатся до операций записи, начатых после барьера.
5. `lfence (load fence)` - Гарантирует, что все операции чтения (`load`) из памяти, инициированные до барьера, завершатся до операций чтения, начатых после барьера.
6. `mfence (memory fence)` - Комбинированный барьер: гарантирует порядок для и чтения, и записи.  
(`fence` - барьер)
7. `movntxx` - инструкции записи в память минуя КЭШ
  - (a) `MOVNTI` - целочисленное
  - (b) `MOVNTPS` - запись упакованных `float` (4\*32) выравнивание 16
  - (c) `MOVNTPD` - запись упакованных `float` (2\*64) выравнивание 16
  - (d) `MOVNTDQ` - запись целого числа 128 бит 16 байт выравнивание
8. `prefetch0 [mem]` - загрузит в L0 L1 L2 кэши
9. `prefetch1 [mem]` - загрузит в L1 L2 кэши
10. `prefetch2 [mem]` - загрузит в L3 кэш
11. `int n` -  $n \in [0, 255]$  обеспечивает программную генерацию прерывания и передачу управления в соответствующий обработчик.
12. `iret` - Возврат из прерывания

регистр флагов - IF (Interrupt Flags) IF = 0 - на прерывание не реагирует

IF = 1 - на прерывание реагирует

`cli sti` - ставит 0 и 1 соответственно

`Idtr (interrupt description table register)` указатель на `Idt (interrupt description table)` - таблицу поля IDT:

- 1 используется или нет
- 2 адрес обработчика прерываний (виртуальный)
- 3 тип обработчика прерываний
- 4' номер сегмента кода (где лежит обработчик прерываний) в GDT
- 4'' Номер TSS (task save segment) сегмента
- 5 DPL - desired privilege level
- 6 размер адреса

при возникновении прерывания оно может выполняться:

1. на старом стеке
2. на новом стеке

В зависимости от уровня привилегий процессор сам может сбрасывать IF или нет

на старом стеке: программа выполнялась дошла до N инструкции произошло прерывание

1. старый CS, `eFlags` и EIP кладем на стек в случае ошибки - информация о ней.

2. сменили сегменты (CS) на тот что в таблице
3. сменили EIP на тот что в таблице
4. CLI - аппаратно

на новом стеке - всё то же самое но сохраняется SS и ESP сохраним: SS ESP CS eflags EIP \*сообщение об ошибке\*  
также используется `iret` новый стек возьмём из TSS  
в прерывании необходимо сохранить все регистры в случае использования TSS не надо.

как строить табличку: в строках команды и их нахождение на такт (номер строки)