Вспоминаем АКОС

19 сентября, 2020

Про проект

- Всё таки будем писать с вами свою ОС под x86
- Что нужно будет, чтобы получить хорошую оценку?

Про проект

- Всё таки будем писать с вами свою ОС под x86
- Что нужно будет, чтобы получить хорошую оценку? It depends :)
- Минимум: загрузка через GRUB, изолированные userspace процессы, файловая система

Про проект

- Всё таки будем писать с вами свою ОС под x86
- Что нужно будет, чтобы получить хорошую оценку? It depends :)
- Минимум: загрузка через GRUB, изолированные userspace процессы, файловая система
- Максимум неограничен :)

Про userspace

- Подумайте над форматом: можно сделать собственный или взять ELF
- **Обязательное условие #1**: если свой формат, то должны быть инструменты, чтобы можно было программировать под вашу платформу
- Сисколлы необязательно должны быть POSIX-compliant, но подумайте о переносимости!
- **Обязательное условие #2**: если свои сисколлы, то нужна подробная документация о них
- **Обязательное условие #3**: процессы должны быть изолированы друг от друга
- **Обязательное условие #4**: пользователи должны уметь запускать C-программы (подумайте о вашей мини-libc)

Про помощь

- Я готов вам всегда помочь советами или совместным дебагом
- Если вы просите помочь подебагать, то у меня должна быть **РАБОТАЮЩАЯ** инструкция, как запустить ваш код
- Мой SLA в телеграме — сутки. Если я не ответил за это время, можно смело напоминать о себе!
- Ещё со мной можно говорить «за жизнь» :)

Ваша домашняя страница на ближайший семестр

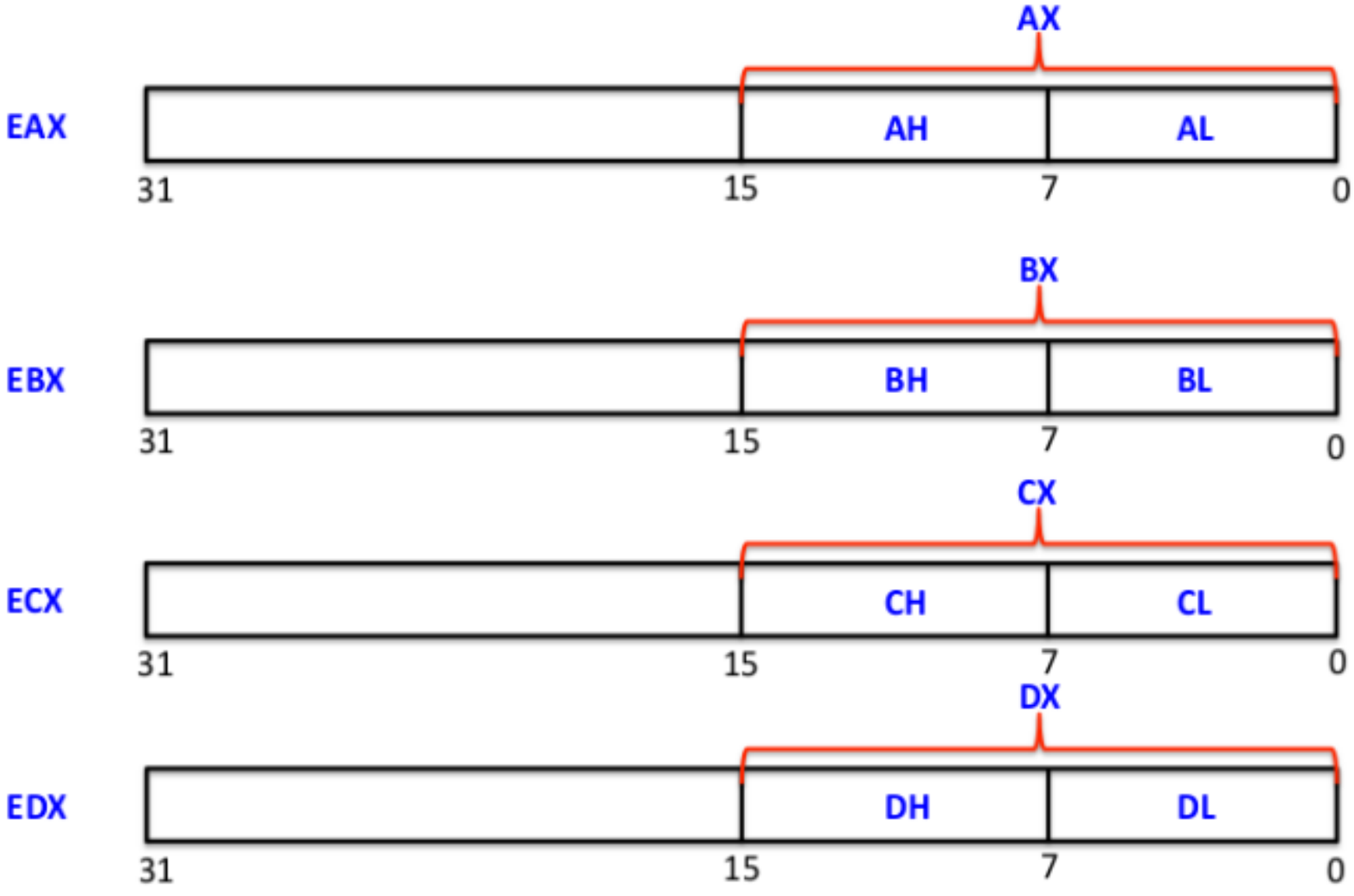
x86

- AMD64, x86, x86-64, IA-32, Intel32/Intel64
- Одна из самых распространённых архитектур процессора
- Используется в процессорах Intel и AMD
- CISC (Complex Instruction Set Computing)
- Регистровая модель (стек засчёт регистров)
- Страничная модель памяти*

General purpose registers

- 16 64-bit GPR: RAX, RBX, RCX, RDX, RSI, RDI, RBP, RSP, R{8-15}
- 8 32-bit GPR: EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP

General purpose registers



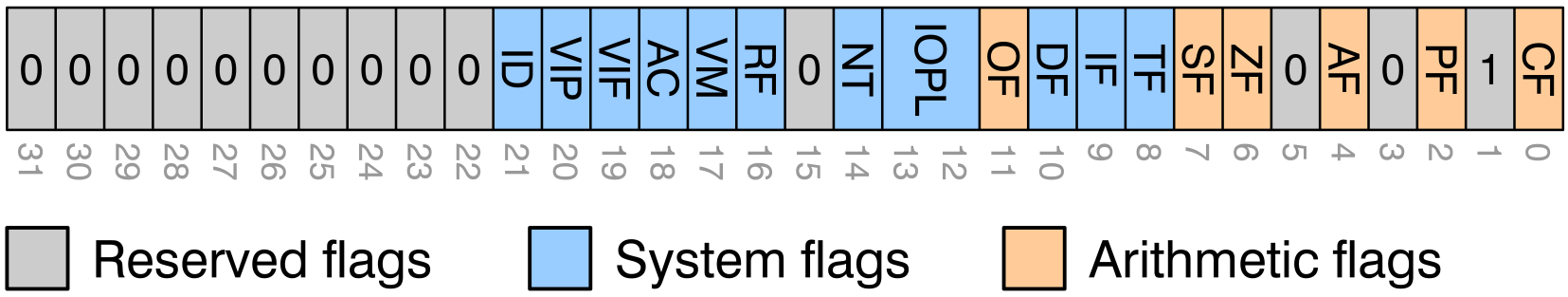
EFLAGS/RFLAGS

- Специальный регистр, каждый бит которого — какой-то флаг
- Нельзя записать значение напрямую, только косвенно через команды
- Два вида флагов: системные и арифметические
- Первые отвечают за текущие настройки поведение процессора
- Вторые предназначены для conditional jumps

Если очень хочется перезаписать EFLAGS/RFLAGS

- `pushf` кладёт EFLAGS на стек
- `popf` восстанавливает EFLAGS со стека

EFLAGS/RFLAGS



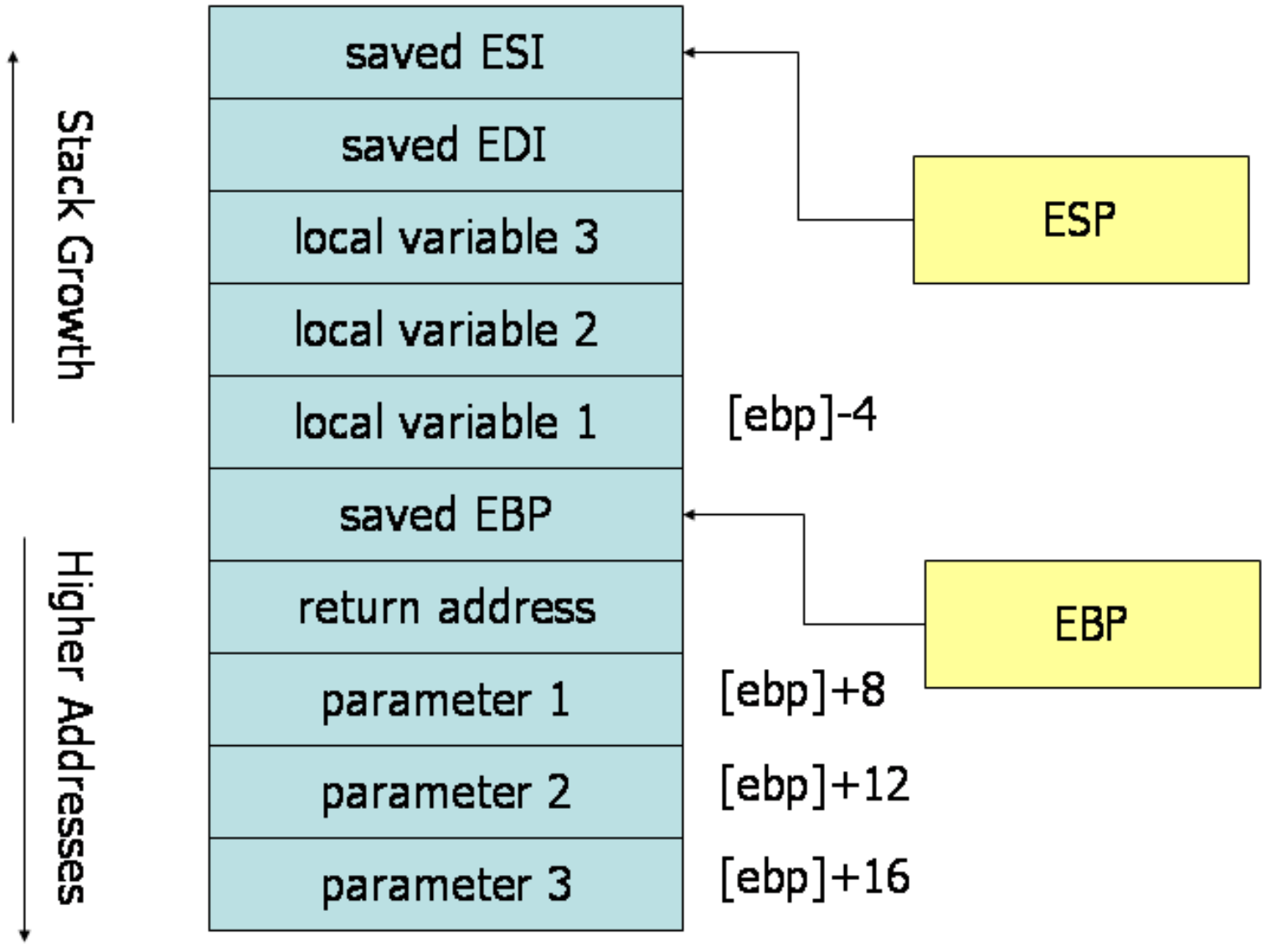
x86 memory segmentation

- Изначально x86 — 16-битная архитектура => только 64 Kb памяти адресуется
- Сегментные регистры: CS, DS, SS, ES, FS, GS.
- $addr = segment * 16 + offset$
- `mov es:cx, 15h`
- FS и GS до сих пор используются :)

Stack

- Реализуется через память
- Стек растёт обратно адресам (самый первый элемент в стеке имеет самый большой адрес)
- RSP указывает на последний занятый байт
- `push` = `sub rsp, size`; `mov [rsp], op`
- `pop` = `mov op, [rsp]`; `add rsp, size`

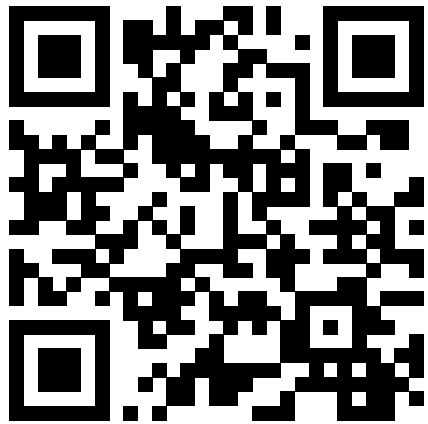
Stack frame


Stack Growth
Higher Addresses
saved ESI
saved EDI
local variable 3
local variable 2
local variable 1
saved EBP
return address
parameter 1
parameter 2
parameter 3
ESP
EBP
[ebp]-4
[ebp]+8
[ebp]+12
[ebp]+16

Calling conventions

- Соглашения о том, как реализуются вызовы функций
- Определяют где лежат аргументы и кто их сохраняет, выравнивание стека итд
- Под разными платформами и ОС — разные
- System V ABI (application binary interface)
- Можете придумать свои, но тогда придётся переписать компиляторы :)

Про инструкции



Intel manual

Как загружается компьютер?

- POST = Power-On Self Test
- BIOS ищет девайсы (партиции диска), помеченные boot-флагом и ищет там специальную MBR-запись в нулевом секторе диска
- 512-байтный сектор загружается по специальному адресу в память
- Процессор прыгает на этот адрес и начинается выполнение вашей ОС
- Интересный факт: так как питание процессора может быть не отключено (нажали на reset), то значения регистров могут быть от старого запуска

Загрузчики

- Иногда бывает dual boot: Windows + Linux
- Не хочется, чтобы ОС реализовывала всю эту загрузку сама
- Загрузчик (например, GRUB) ставится на диск, а ОС загружается уже из файла на диске
- Загрузчик умеет работать с некоторыми файловыми системами и форматами ядер (ELF + бинарный формат)

Real, protected и long mode

- Это различные режимы x86 процессоров, каждый обратно совместим с предыдущим
- После загрузки MBR вы находитесь в real mode — самый древний 16-битный режим
- Затем ОС делает подготовительные шаги и переходит в protected mode
- Аналогично с long mode
- Сделано это для того, чтобы на современных процессорах можно было запускать старые ОС (обратная совместимость)

Real mode

- Отсутствует виртуальная память => нет прав на память, она вся RWX
- Сегментная адресация => максимум 1 Мб памяти адресуем
- Можно использовать специальные BIOS functions для доступа к периферии

Protected mode

- Использует 32ух битные адреса
- Виртуальная память и страничная адресация
- Поддерживает механизм привилегий (т.н. кольца, rings)
- Поддержка кооперативной многозадачности из коробки (TSS = task state segment)

Виртуальная память и страничная адресация

- Вся физическая память разделена на *фреймы* — куски размером 4096 байт
- Вся виртуальная память аналогично разделена на *страницы*
- Трансляцией виртуальной памяти в физическую занимается *memory management unit* (MMU)

Page tables

- Специальные структуры, которые хранят отображение виртуальной памяти в физическую
- Всего существует 2^{52} страниц памяти
- Если каждая страница описывается 8 байтами, то такая структура занимает 2^{60} байт в памяти
- Нужен более экономный способ хранить это отображение

Multi-level page tables

- Идея: давайте сделаем таблицы многоуровневыми — сначала поделим всё пространство на части, каждую из этих частей ещё на части итд
- Не храня лишние «дыры» мы будем экономить место

Multi-level page tables

- Идея: давайте сделаем таблицы многоуровневыми — сначала поделим всё пространство на части, каждую из этих частей ещё на части итд
- Не храня лишние «дыры» мы будем экономить место
- Под x86-64 используются четырёхуровневые таблицы: P4, P3, P2, P1.
- Каждая таблица занимает ровно 4096 байт и содержит 512 записей (*PTE = page table entry*) по 8 байт
- Каждая запись ссылается на индекс в следующей таблице, последняя таблица ссылается на адрес фрейма

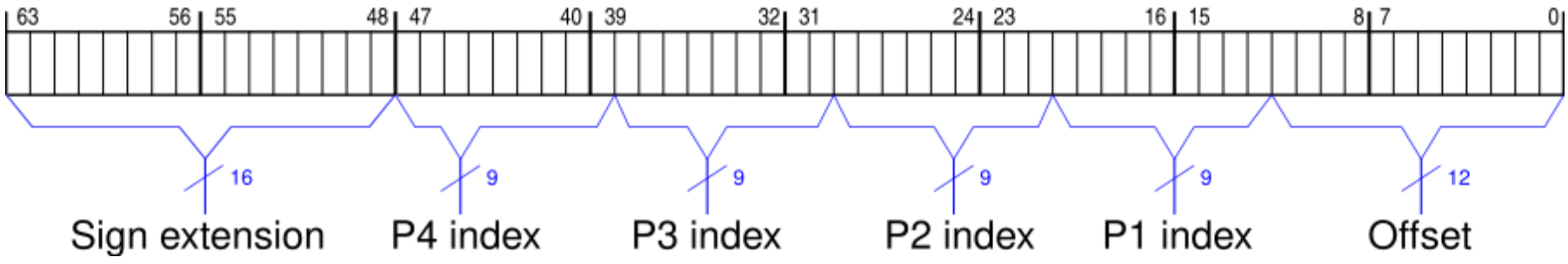
Что хранится в PTE?

- Индексация следующей таблицы или фрейма не занимает все 8 байт PTE
- Кроме неё в PTE есть ещё специальные *флаги страниц*
- Например, 1-ый бит отвечает за то, будет ли страница доступна на запись
- 63ий – за то, будет ли процессор исполнять код на этой странице
- Также в некоторые биты процессор сам пишет флаги, например, dirty-бит устанавливается всегда, когда происходит запись в страницу
- Флаги имеют иерархическую видимость: если в P2 writeable-бит равен 0, а в P4 – 1, то страница будет доступна на запись

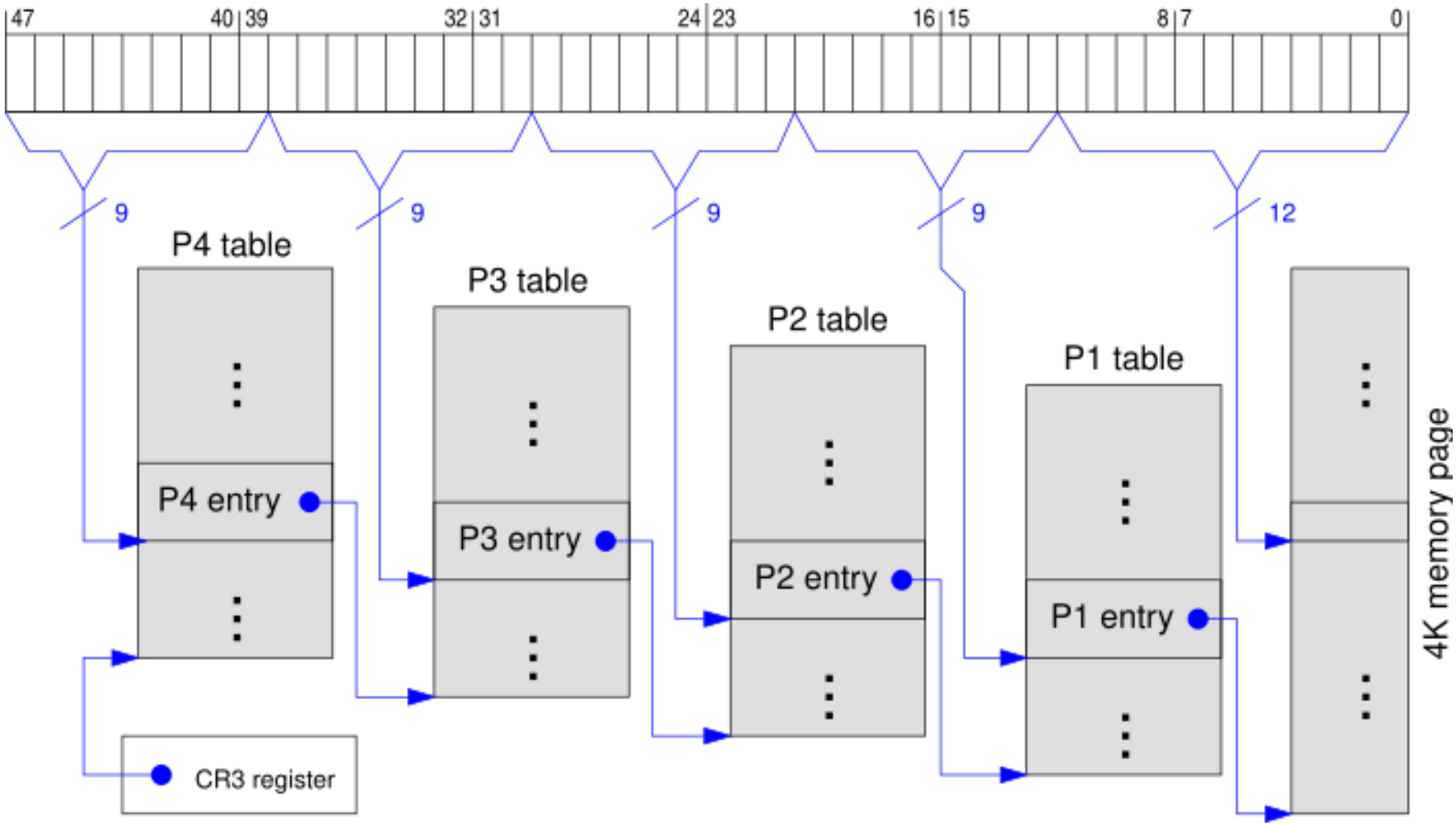
Устройство виртуального адреса

- На текущий момент x86-64 позволяет адресовать 48 бит физической памяти
- Старшие биты (с 48 по 63) должны быть sign extended копии 47ого бита
- Следующие биты (с 38 по 47) адресуют PTE в P4
- Биты с 29 по 37 адресуют PTE в P3
- Биты с 21 по 28 адресуют PTE в P2
- Биты с 12 по 20 адресуют PTE в P1, которая ссылает непосредственно на фрейм
- Биты с 0 по 11 адресуют смещение внутри фрейма

Устройство виртуального адреса



Устройство виртуального адреса



ОС и таблицы страниц

- Операционная система хранит таблицы страниц для каждого процесса
- Таблица страниц переключается каждый раз при context switch
- Физический адрес текущей P4 хранит специальный регистр CR3 (такие регистры называются *MSR = model specific registers*)
- В реальности каждое обращение к памяти не вызывает прыжки по таблицам, оно кэшируется в *TLB = translation lookaside buffer*
- При context switch TLB полностью сбрасывается

Выделение памяти: on-demand paging

- Современные ОС не выделяют всю запрошенную память сразу
- Вместо этого используется on-demand paging
- Если страницы нет в текущем memory mapping'е, то процессор сгенерирует специальное исключение, называемое *page fault*'ом
- Идея состоит в том, чтобы детектировать с помощью page fault'ов реальные обращения к памяти и только тогда её выделять

Прерывания процессора

- Могут выполниться после любой инструкции процессора
- Специальные участки кода с наибольшим приоритетом
- Прерывание в прерывании, ух!
- Вызываются ошибками памяти (обращение к несуществующей странице, запись в readonly страницу)
- Или событиями от периферийных устройств (сетевая карта, клавиатура)
- Одно из самых важных для ОС прерываний — timer interrupt

Спасибо!