

Обзор x86-64

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)

Инструкции

- Мнемоники – представление инструкций для человека
- Оpcodes (opcodes) – машинное представление инструкций
- Размер опкода – от 1 до 15 байт
- Процессор последовательно* декодирует и исполняет инструкции

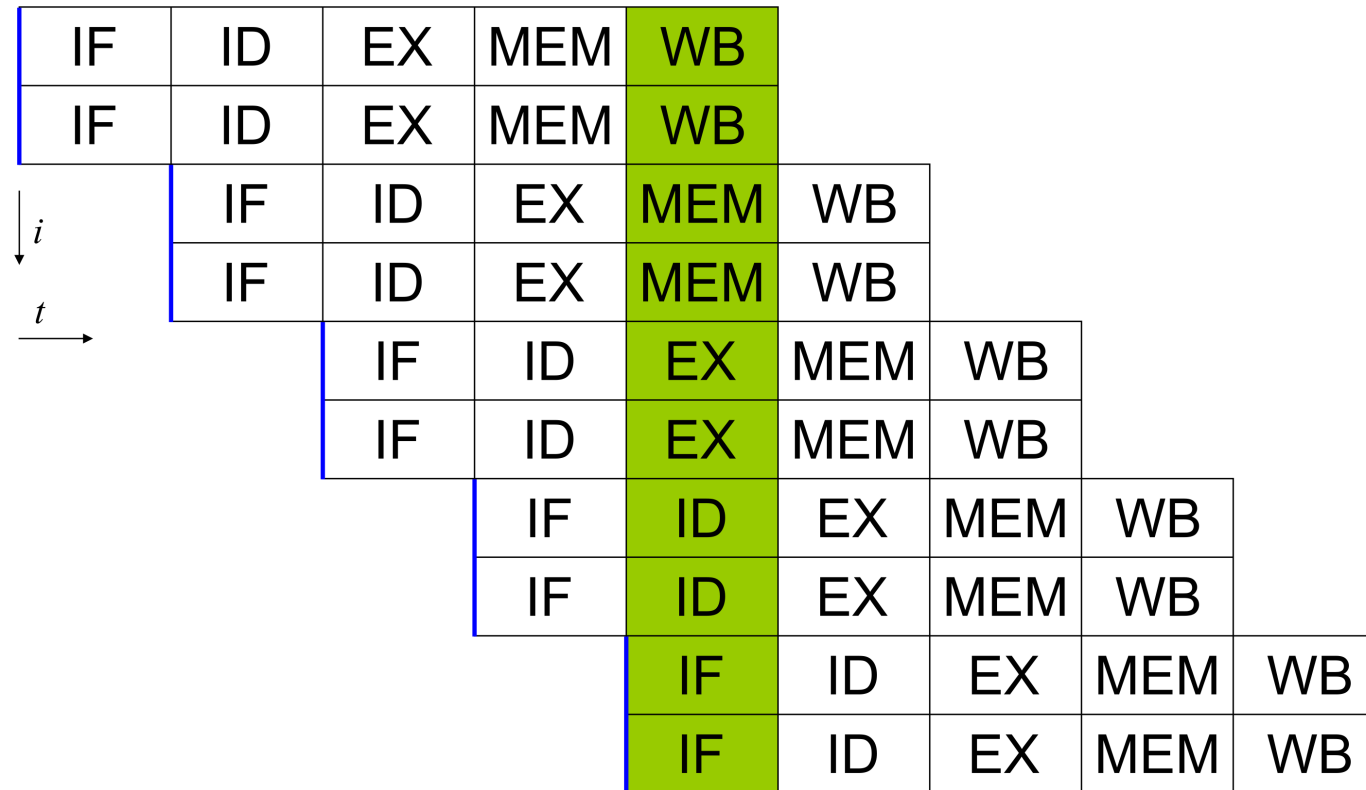
Инструкции

- Представляют из себя операции над памятью или регистрами
- Аргументы инструкций традиционно называют "операндами"
- Инструкции обычно имеют 1-3 операнда
- По крайней мере один из операндов должен быть регистром

CPU pipeline

- Каждая инструкция состоит из нескольких фаз: fetch, decode, execute, memory access, register writeback
- Фазы для нескольких инструкций могут выполняться параллельно
- Это называется *суперскалярностью*

CPU pipeline



General purpose registers

- Самая быстрая "память"
- Всего 16 штук: RAX, RBX, RCX, RDX, RSI, RDI, RBP, RSP, R{8-15}
- Каждый 64 бита, но делятся на более мелкие

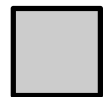
63	31	15	8	7	0	
%rax	%eax	%ax	%ah	%al		Return value
%rbx	%ebx	%bx	%bh	%bl		Callee saved
%rcx	%ecx	%cx	%ch	%cl		4th argument
%rdx	%edx	%dx	%dh	%dl		3rd argument
%rsi	%esi	%si		%sil		2nd argument
%rdi	%edi	%di		%dil		1st argument
%rbp	%ebp	%bp		%bpl		Callee saved
%rsp	%esp	%sp		%spl		Stack pointer
%r8	%r8d	%r8w		%r8b		5th argument
%r9	%r9d	%r9w		%r9b		6th argument
%r10	%r10d	%r10w		%r10b		Callee saved
%r11	%r11d	%r11w		%r11b		Used for linking
%r12	%r12d	%r12w		%r12b		Unused for C
%r13	%r13d	%r13w		%r13b		Callee saved
%r14	%r14d	%r14w		%r14b		Callee saved
%r15	%r15d	%r15w		%r15b		Callee saved

Регистр флагов EFLAGS

- Специальный регистр флагов, каждый бит = флаг
- Нельзя записать значение через mov, только косвенно через инструкции
- Два вида флагов: системные и арифметические
- Первые отвечают за текущее поведение процессора
- Вторые возникают после выполнения арифметических операций

EFLAGS

0	CF
1	1
2	PF
3	0
4	AF
5	0
6	ZF
7	SF
8	TF
9	IF
10	DF
11	OF
12	IOPL
13	
14	NT
15	0
16	RF
17	VM
18	AC
19	VIF
20	VIP
21	ID
22	0
23	0
24	0
25	0
26	0
27	0
28	0
29	0
30	0
31	0



Reserved flags



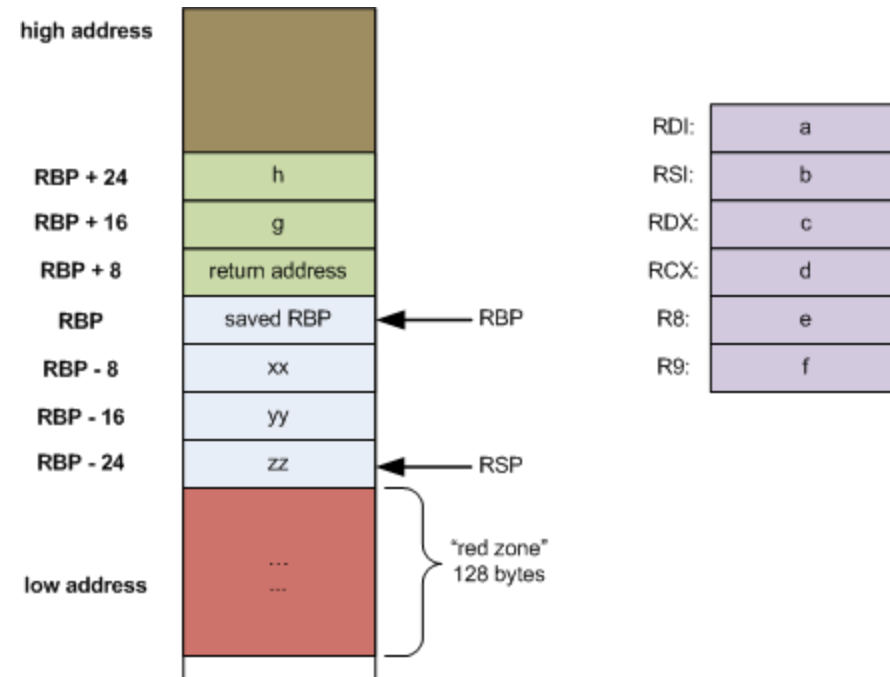
System flags



Arithmetic flags

Стек

- Используется для вызова функций
- Каждая функция имеет свой фрейм в стеке, где хранится часть её аргументов, адрес возврата и локальные переменные
- Как передаются аргументы определяет ABI – application binary interface
- System V AMD64 ABI



AT&T vs Intel syntax



Ltmp2:

```
.cfi_def_cfa_register %rbp
movslq %edi, %rax
imulq $1759218605, %rax,
movq %rsi, %rax
shrq $63, %rax
sarq $44, %rsi
addl %eax, %esi
leaq L_.str(%rip), %rdi
xorl %eax, %eax
callq _printf
xorl %eax, %eax
popq %rbp
retq
.cfi_endproc
```

Ltmp2:

```
.cfi_def_cfa_register rbp
movsxd rax, edi
imul rsi, rax, 175921860
mov rax, rsi
shr rax, 63
sar rsi, 44
add esi, eax
lea rdi, [rip + L_.str]
xor eax, eax
call _printf
xor eax, eax
```

Загрузка ОС

BIOS

- Basic Input-Output System
- При самом запуске процессор начинает исполнять код BIOS, который был
- Запускается POST = power-on self test
- Ищет на диске специальный 512ти байтовый сектор, содержащий MBR (master boot record) и загружает его по фиксированному адресу, передавая управление
- Драйвер диска/VGA довольно сложно уместить в один сектор, поэтому существуют BIOS interrupt calls – интерфейсы для чтения с диска/вывода на экран
- UEFI приходит на замену BIOS

Bootloader

- Загружает операционные системы с диска
- Поддерживает разные форматы ОС: например, ядро Linux – ELF файл
- Поддерживает разные файловые системы
- Примеры: GRUB, syslinux, LILO, rEFInd

Real, protected и long mode

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

Multiboot 2

- Стандарт взаимодействия загрузчика и ОС
- Определяет состояние регистров и внутренних структур процессора на момент передачи контроля в ОС
- Загрузчик также составляет специальные таблицы: memory map, framebuffer info, итд
- Всё можно найти в спецификации
- GRUB – пример Multiboot-совместимого загрузчика

Память в x86-64

Сегментная адресация

- Изначально x86 – 16-битная архитектура => адресуется 64 Kb памяти
адресуется одним регистром
- Сегментные регистры: CS, DS, SS, ES, FS, GS.
- $addr = segment * 16 + offset$
- 20 бит на адрес => адресуется 1 Mb
- `mov es:cx, 15h`

Global Descriptor Table

- Сегментный регистр используется как номер для индекса в специальной таблице – Global Descriptor Table (GDT)
- Каждый дескриптор описывает сегмент данных или кода
- GDB используется, чтобы понизить уровень привилегий кода пользовательских процессов

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

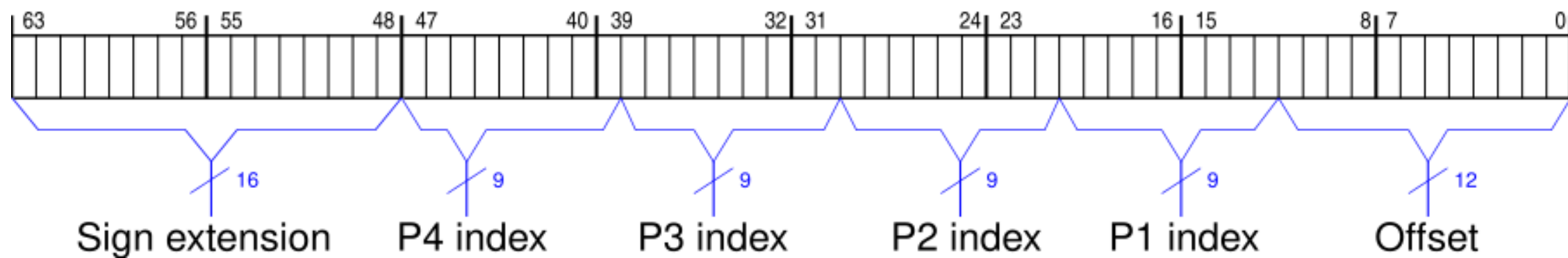
- Идея: давайте сделаем таблицы многоуровневыми — сначала поделим всё пространство на части, каждую из этих частей ещё на части итд
- Не храня лишние «дыры» мы будем экономить место
- x86-64 используется четырёхуровневые таблицы: P4, P3, P2, P1.
- Каждая таблица занимает ровно 4096 байт и содержит 512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

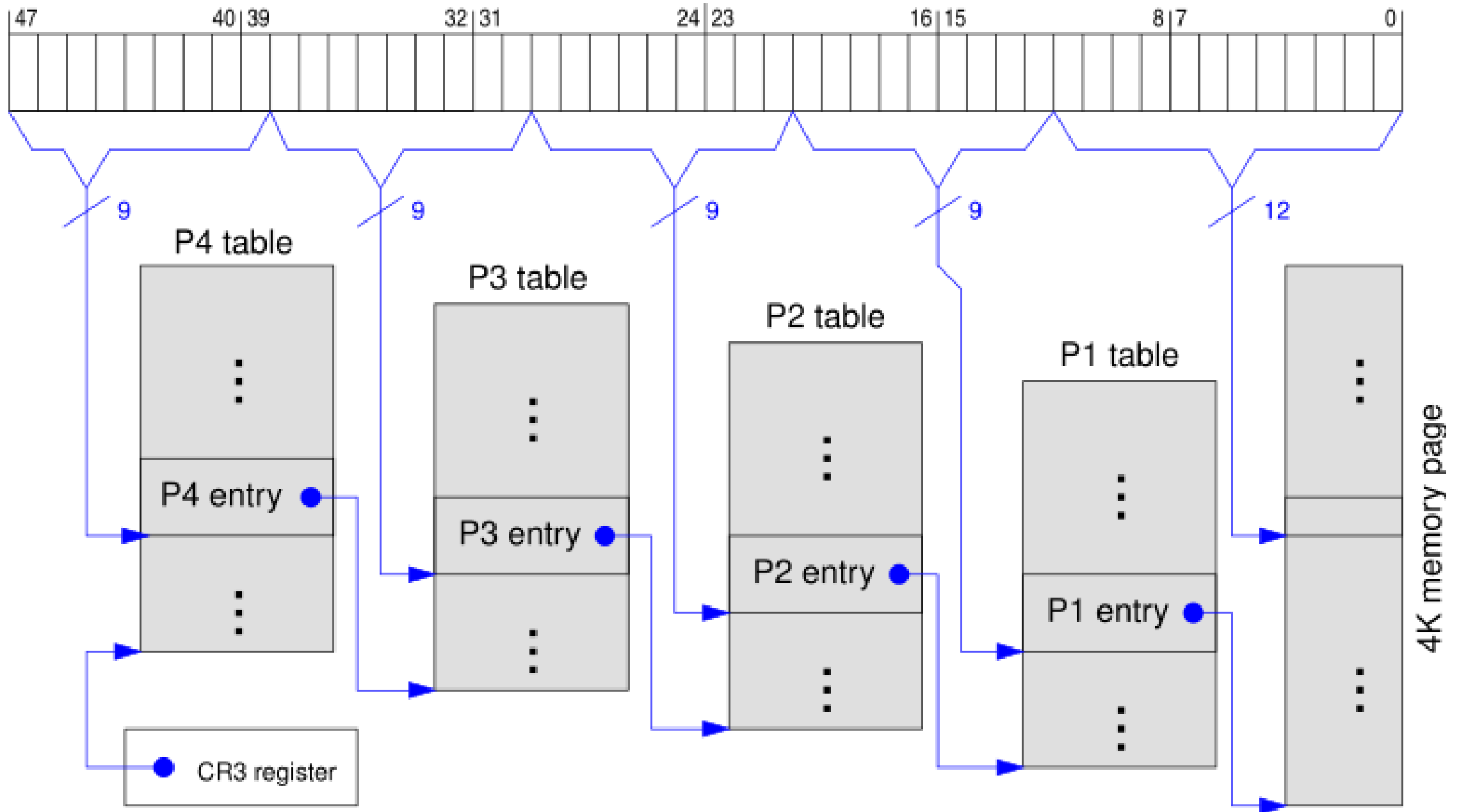
Что хранится в PTE?

- Фреймы выровнены по границе 4096 байт => их адреса на конце всегда имеют 12 нулевых бит
- Внутри PTE эти 12 бит хранят специальные *флаги страниц*
- Некоторые флаги: write, present, user/supervisor
- Флаги, которые устанавливает процессор при доступе к странице: dirty, accessed
- 63ий (MSB) – за то, будет ли процессор исполнять код на этой странице (NX flag)
- 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 (256 Tb)
- Старшие биты (с 48 по 63) должны быть sign extended копией 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 ссылается непосредственно на фрейм
- Биты с 0 по 11 адресуют смещение внутри фрейма





ОС и таблицы страниц

- Операционная система хранит таблицы страниц для каждого процесса
- Таблица страниц переключается каждый раз при context switch
- Физический адрес текущей Р4 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

Прерывания процессора

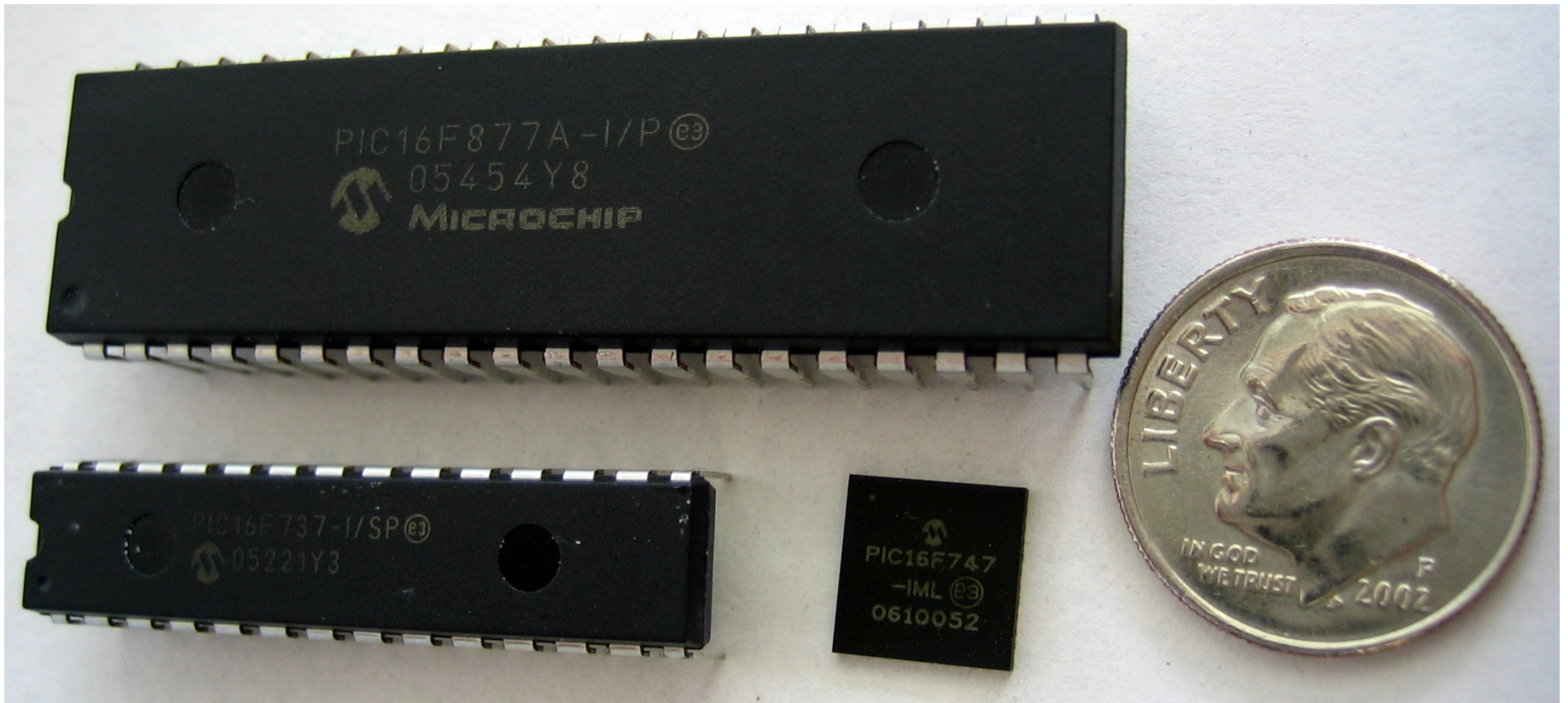
- Могут выполняться *после* любой инструкции процессора
- Специальные участки кода с наибольшим приоритетом
- Прерывание в прерывании, ух!
- Exceptions – вызываются ошибками памяти (обращение к несуществующей странице, запись в readonly страницу)
- Interrupt request (IRQ) или hardware interrupts – событиями от периферийных устройств (сетевая карта, клавиатура)
- Software interrupts могут вызываться из кода, используются в основном для системных вызовов
- Можно включить/отключить через флаг IF в EFLAGS
- Одно из самых важных для ОС прерываний — timer interrupt

Прерывания процессора

- Когда возникает прерывание, процессор сохраняет своё минимальное состояние на *текущем* стеке: EFLAGS, CS, EIP
- Далее берётся адрес из специальной таблицы IDT (её ещё иногда называют вектором прерываний) и передаётся туда управление
- Однако, x86-64 позволяет подменять стек до этого
- Сохранение и восстановление остальных регистров (контекста) – дело операционной системы

Контроллеры прерываний: PIC и APIC

- Peripheral Interface Controller (PIC) – контроллер управления прерываниями
- Advanced Programmable Interrupt Controller (APIC) – более современная версия
- Обычно PIC – отдельный контроллер, APIC находится прямо в процессоре
- Позволяют отключать (mask) или включать (unmask) прерывания
- Также позволяют менять номер прерывания
- Level triggered: если пин прерывания == 1, то прерывание всегда срабатывает
- Edge triggered: прерывание срабатывает только когда пин поменялся с 0 на 1



Thank you!