

Operating Systems

Interrupts

Me

September 16, 2016

Внешние устройства

- ▶ Допустим у вас есть сетевая карта:
 - ▶ можно передать сетевой карте набор байт для отправки;
 - ▶ сетевая карта может получать данные, процессор должен скопировать эти данные в память, обработать и "показать" пользователю;
 - ▶ пакеты могут приходиться в произвольные моменты времени.
- ▶ Как узнать, что данные пришли на сетевую карту?
 - ▶ можно спросить устройство (проверить какой-нибудь бит в каком-нибудь регистре устройства);
 - ▶ такой вариант называют *polling*;
 - ▶ пока код исполняемый процессором опрашивает устройство, процессор не делает ничего полезного.

Прерывания

- ▶ Прерывания - сигнал процессору, который "прерывает" текущий исполняемый код процессора
 - ▶ сетевая карта посылает сигнал при получении данных;
 - ▶ вместо исполняемого кода вызывается специальный обработчик прерывания;
 - ▶ обработчик прерывания обслуживает устройство, и возвращает управление прерванному коду.
- ▶ Следствия:
 - ▶ не нужно тратить ресурсы процессора на бесполезный опрос устройств;
 - ▶ прерванный код может быть не готов к тому, что его прервут - задача обработчика позаботится об этом.

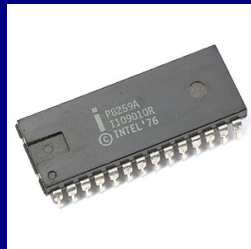
Контроллер прерываний

- ▶ А что если у нас много устройств требующих внимания процессора?
 - ▶ какое из устройств сгенерировало прерывание?
 - ▶ если сразу несколько устройств сгенерировали прерывания?
- ▶ Для разрешения этих проблем нужен посредник между устройствами и процессором
 - ▶ такого посредника называют контроллером прерываний;
 - ▶ контроллер прерываний выполняет арбитраж.

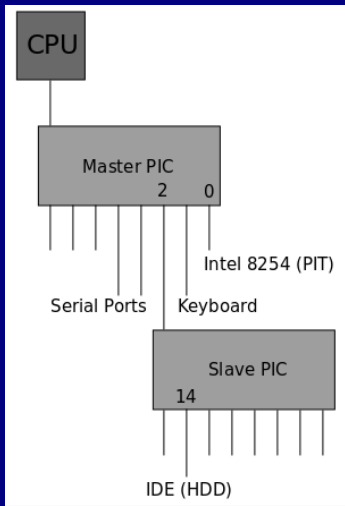
Intel 8259

Intel 8259 - *программируемый*
контроллер прерываний (далее просто
PIC)

- ▶ каскад из двух PIC использовался в IBM PC начиная с AT;
- ▶ сейчас он не используется, но его поведение эмулируется современными контроллерами прерываний.



Каскад intel 8259

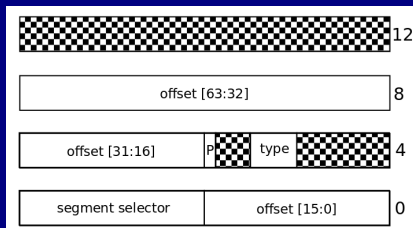


- ▶ 2 контроллера по 8 выходов - 1 выход = 15 внешних устройств;
- ▶ большинство из выходов в IBM PC заняты фиксированными устройствами;
- ▶ вы тесно познакомитесь с PIT (programmable interval timer).

Обработка прерываний

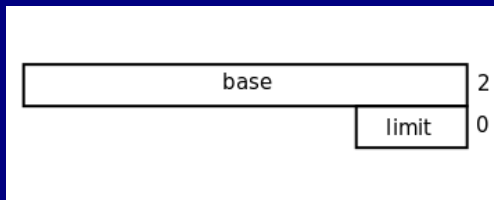
- ▶ Чтобы получать и обрабатывать прерывания необходимо:
 - ▶ запрограммировать контроллер прерываний (PIC);
 - ▶ указать процессору, где находятся обработчики прерываний;
- ▶ на x86 для указания на обработчики прерываний используется IDT
 - ▶ IDT (interrupt descriptor table) - таблица из максимум 256 дескрипторов, описывающих обработчики прерывания;
 - ▶ т. е. в x86 архитектуре можно задать не более 256 обработчиков прерываний;
 - ▶ из них первые 32 зарезервированы, т. е. остается 224 под наши нужды.

IDT



- ▶ *offset* - адрес обработчика;
- ▶ *segment selector* (биты [31:16] в 1-ом слове) - селектор (регистр CS);
- ▶ *P* (Present, бит 15 во 2-ом слове) - должен быть равен 1;
- ▶ *type* (биты [11:8] во 2-ом слова) - домашнее задание разобраться в разнице между Interrupt Gate и Trap Gate;
- ▶ все остальное должно быть равно 0.

IDTR



- ▶ Информация о местоположении IDT хранится в специальном регистре (IDTR);
- ▶ загрузить значение в регистр можно с помощью специальной инструкции `lidt`;
 - ▶ параметром инструкции является специальный "дескриптор";
 - ▶ *base* - 64-битный адрес IDT в памяти;
 - ▶ *limit* - 16-битный размер IDT в байтах минус единица.

Программирование PIC

- ▶ Как выбирается какую запись IDT использовать при прерывании?
 - ▶ вам нужно записать в контроллер на какие записи IDT отображаются входы контроллера;
 - ▶ в случае PIC, вам нужно указать на какую запись IDT отображается самый первый вход каждого контроллера, все остальные идут по порядку.
- ▶ Кроме отображения контроллеру прерываний также нужно указать:
 - ▶ конфигурацию каскада (как Master и Slave соединены);
 - ▶ тип прерывания (edge/level);
 - ▶ и много другого не интересного.

Взаимодействие с PIC

- ▶ Для общения с PIC используются два 8-ми битных регистра: регистр команд и регистр данных;
 - ▶ в регистр команд записывается действие, которое нужно сделать;
 - ▶ затем в регистр данных записываются данные (сколько и какие именно зависит от команды).
- ▶ Для доступа к регистрам PIC-ов в x86 используется пространство ввода/вывода и специальные инструкции работы с ним:
 - ▶ инструкции называются *in* и *out*, а аргументы порт ввода/вывода и данные;
 - ▶ регистр команд Master PIC соответствует порту *0x20*, а регистр данных порту *0x21*;
 - ▶ Slave PIC использует порты *0xA0* и *0xA1*.

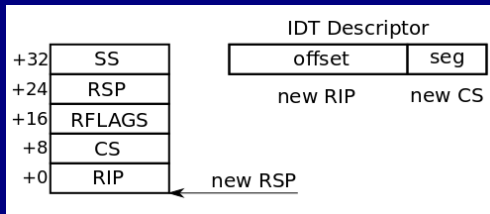
Отображение входов PIC

- ▶ Чтобы настроить отображение необходимо записать в командный регистр значение `0x11` - команда инициализации контроллера;
- ▶ команда инициализации ожидает три байта данных:
 - ▶ номер записи IDT, соответствующей самой первому входу контроллера;
 - ▶ параметры каскада:
 - ▶ для Master PIC - битовую маску входов, к которым подключены Slave PIC-и (в нашем случае это просто 4 - 2-ой бит равен 1, остальные 0);
 - ▶ для Slave PIC - номер входа Master PIC, к которому он подключен (в нашем случае это 2);
 - ▶ прочие неинтересные параметры (в нашем случае нужно записать число 1).

Маскировка прерываний

- ▶ Иногда полезно запретить доставку определенных прерываний (замаскировать)
 - ▶ до того как драйвер/ОС настроило устройство, прерывания от него лучше отключить;
 - ▶ т. е. пре инициализации PIC все прерывания лучше замаскировать.
- ▶ Для маскировки прерываний в случае PIC используется регистр данных:
 - ▶ запишите в регистр данных битовую маску прерываний (замаскированным соответствуют 1-цы).
- ▶ Прерывания можно замаскировать на процессоре
 - ▶ на x86 это делается инструкцией *cli*, обратная к ней инструкция *sti*.

Вызов обработчика прерываний



- ▶ При прерывании процессор сохраняет минимум информации:
 - ▶ *CS* и *RIP* указывают какой код и с каким уровнем привелегий мы прервали;
 - ▶ *RFLAGS* - процессор может менять значение флагового регистра при вызове обработчика и сохраняет старое;
 - ▶ *SS* и *RSP* указывают на стек прерванного кода.

Обработчик прерывания

- ▶ Обработчик прерывания должен сохранить состояние прерванного кода
 - ▶ как минимум нужно сохранить регистры общего назначения;
 - ▶ для x86: RAX, RBX, RCX, RDX, RBP, RDI, RSI, R9 - R15.
- ▶ Обработчик прерывания "должен" вернуть управление прерванному коду;
 - ▶ в x86 для этого, *обычно*, используют инструкцию *iretq*, которая восстанавливает со стека сохраненные регистры CS, SS, RFLAGS, RIP и RSP;
 - ▶ т. е. перед исполнением *iretq* стек нужно вернуть в "исходное" состояние.

End Of Interrupt

- ▶ Обработчик прерывания должен уведомить контроллер прерывания о завершении обработки (послать EOI);
 - ▶ контроллер прерываний должен знать когда выдать следующий сигнал.
- ▶ Послать EOI PIC-у можно записав в регистр команд значение специального вида:
 - ▶ $0x60 + irq$, где irq - номер входа PIC (от 0 до 7), от которого было получено прерывание;
 - ▶ обратите внимание, т. к. PIC-и объединены в каскад, то если прерывание пришло на Slave PIC, то EOI нужно посылать сразу и Master и Slave PIC-ам.

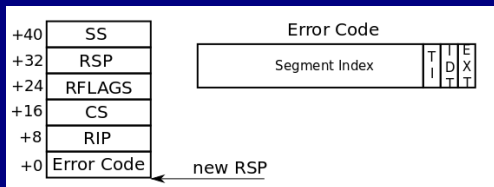
Исключения

- ▶ Во время исполнения команд процессор может обнаружить ошибку:
 - ▶ попытка исполнить некорректную инструкцию;
 - ▶ недостаточный уровень привелегий, для выполнения действия;
 - ▶ деление на 0;
 - ▶ многое другое...
- ▶ Такие ситуации называются исключительными и требуют обработки
 - ▶ x86 сообщает об исключениях в виде прерывания (одного из тех 32 зарезервированных);
 - ▶ эти прерывания не связаны с контроллерами прерываний и внешними устройствами, а являются внутренними для процессора.

Критические и некритические ошибки

- ▶ В x86 исключения делятся на три категории:
 - ▶ не критические ошибки (faults) - предполагается, что такие ошибки можно исправить и возобновить работу с инструкции приведшей к ошибке (т. е. RIP на стеке содержит адрес "плохой" инструкции);
 - ▶ ловушки (traps) - позволяют, условно, отслеживать выполнение некоторых инструкций, после обработки исключение управление передается следующей инструкции;
 - ▶ критические ошибки (aborts) - с такой ошибкой мало что можно сделать, остается только сообщить о ней и "упасть".

Особенности обработки исключений в x86



- ▶ Для некоторых исключений стек содержит код:
 - ▶ №8 Double Fault Exception - *Error Code* всегда равен 0;
 - ▶ №10 Invalid TSS Exception;
 - ▶ №11 Segment Not Present;
 - ▶ №12 Stack Fault Exception;
 - ▶ №13 General Protection Exception;
 - ▶ №14 Page-Fault Exception - *Error Code* имеет свой формат;
 - ▶ №17 Alignment Check Exception - *Error Code* 0 или 1.

Программные исключения

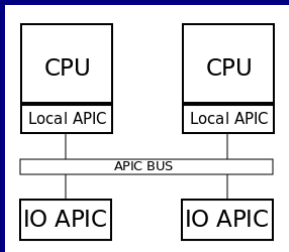
- ▶ Кроме аппаратных прерываний и исключений, есть еще и программные прерывания:
 - ▶ такие прерывания генерируются специальной инструкцией (для x86 это *int*, т. е. по сути это *trap*);
 - ▶ под такие прерывания можно отвести любые незанятые дескрипторы IDT;
 - ▶ так же как и исключения, программные прерывания не связаны с контроллером прерываний;
- ▶ Зачем генерировать исключения программно?
 - ▶ для вызова привелигированного кода из непривелигированного;

Многопроцессорные системы и прерывания

- ▶ Многопроцессорные системы поднимают ряд новых вопросов:
 - ▶ какой из процессоров должен обрабатывать прерывание?
 - ▶ все ли процессоры эквивалентны для обработки прерываний?
 - ▶ как идентифицировать нужный процессор?
- ▶ На замену старым PIC-ам в многопроцессорных системах пришли APIC-и
 - ▶ APIC - Advanced Programmable Interrupt Controller;
 - ▶ например, на ваших домашних компьютерах и ноутбуках используются, почти наверняка, используются именно они;

APIC

- ▶ IO APIC - контроллер, к которому подключаются устройства;
 - ▶ каждый вход настраивается независимо;
 - ▶ сообщение можно отправить любому Local APIC-у или группе Local APIC-ов;
- ▶ Local APIC - свой у каждого CPU;
 - ▶ каждый Local APIC имеет свой идентификатор;
 - ▶ Local APIC-и могут обмениваться сообщениями друг с другом;



Message Signaled Interrupts

- ▶ С обычными прерываниями каждое устройство занимает как минимум один вход контроллера прерываний
 - ▶ несколько устройств могут использовать одну линию, но это может создавать трудности;
 - ▶ входы контроллера прерываний нужно "развести" на схеме.
- ▶ Зачем нам IO APIC и его провода?
 - ▶ пусть устройства напрямую отправляют сообщения к Local APIC;
 - ▶ реализовать устройство поддерживающее MSI тяжелее, но нет необходимости "разделять" прерывания;
 - ▶ MSI обладают меньшими задержками чем обычные прерывания.

Intel 8254

- ▶ Intel 8254 (aka PIT) - интервальный таймер, т. е. устройство генерирующее сигналы с заданной частотой;
 - ▶ обычно PIT подключен к 0 входу Master PIC;
 - ▶ у PIT есть 3 выхода, из которых к Master PIC подключен только 0-ой.
- ▶ Частота работы PIT 1193180 Гц
 - ▶ задав "коэффициент деления" можно получить меньшую частоту генерации сигналов;
 - ▶ коэффициент деления k , значит, что на выход попадает только каждый k -ый сигнал.
- ▶ PIT имеет внутренний счетчик
 - ▶ мы можем задать начальное значение для счетчика, и он будет уменьшаться на каждый сигнал пока не дойдет до 0;
 - ▶ что произойдет дальше зависит от режима работы.

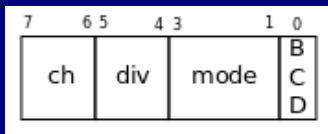
Режимы работы PIT

- ▶ One Shot (mode 0)
 - ▶ задаем начальное значение счетчика, по достижении 0 генерируется прерывание, на этом работа останавливается.
- ▶ Rate Generator (mode 2)
 - ▶ по достижении 0 генерируется прерывание, счетчик перезагружается и отсчет повторяется заново.
- ▶ Есть и другие режимы работы (но нам они не особо интересны):
 - ▶ Square Wave Generator (mode 3);
 - ▶ Software Triggered Strobe (mode 4);
 - ▶ и другие...

Программирование PIT

- ▶ Программирование PIT осуществляется через два порта ввода/вывода:
 - ▶ Control Port $0x43$ - "команда" и выбор канала (в нашем случае канал всегда 0);
 - ▶ Data Port $0x40$ - данные команды (у каждого канала свой Data Port $0x40$ используется для 0-ого).

Формат команды PIT



- ▶ BCD - формат представления чисел (вам точно не нужна 1 в этом бите);
- ▶ mode - режим работы;
- ▶ div - какие байты делителя вы хотите задать (делитель 16-битное число):
 - ▶ 1 - хотим записать только младший байт;
 - ▶ 2 - хотим записать только старший байт;
 - ▶ 3 - хотим записать оба байта;
- ▶ ch - канал, нам нужен 0.

Задание коэффициента деления

- ▶ Если в команде поле `div` не 0, то PIT будет ожидать записи коэффициента деления в Data Port;
 - ▶ если в поле `div` установлен только один бит, то нужно просто записать соответствующий байт в Data Port;
 - ▶ если в поле `div` установлены оба бита, то сначала нужно записать младший байт, а затем старший байт.

Q&A