Урок 4: Демонстрація переривань

Катерина Ковальчук

Грудень 2024р

Анотація

У цьому документі описується процес та структура переривань та їхньої обробки у x86_64.

1. Вступ

Переривання (interrupt) — сигнал, що повідомляє процесор або мікроконтролер про настання якої-небудь події, яка потребує невідкладної уваги щодо її обробки.

Початково переривання в Intel процесорах бути реалізовані у версії процесора 8080. Він мав спеціальний сигнал переривання під назвою INT, який, зазвичай, підключався до першого програмованого контролера переривань Intel (PICU), 8214, і допоміжного пристрою вводу-виводу, 8212.

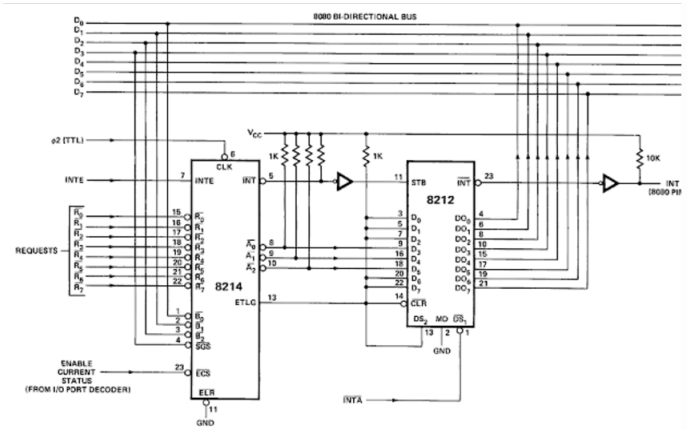

Рис. 1: Схема 8080

8214 буде підключено до периферійних пристроїв вводу-виводу. Кількість переривань може бути збільшена шляхом «каскадування» кількох 8214 PICU, максимум до 40 переривань. Порядок зведення операцій виглядає так:

1. Периферійний пристрій просить 8214 перервати 8080.

2. 8214 стверджує INT до 8080.
3. 8080 відповідає через 8212 з підтвердженням переривання — INTA#.
4. ЦП виконує відповідну процедуру обслуговування переривань.
5. ЦП повертається до нормального виконання програми.

Переривання можна ввімкнути та вимкнути за допомогою сигналу на 8080 під назвою INTE , "Interrupt Enable".

2. Компіляція

Процес компіляції повністю поторює процес із попередніх уроків із потрібними змінами у Makefile.

ПІД ЧАС ПОБУДОВИ ДОКЕР-ЗОБРАЖЕННЯ МОЖЕ ВИНИКНУТИ ТАКА ПОМИЛКА:

```
docker build buildenv -t myos-buildenv
[+] Building 0.4s (3/3) FINISHED
    docker:default
 => [internal] load build definition from Dockerfile
     0.0s
 => => transferring dockerfile: 294B
     0.0s
 => [internal] load .dockerignore
     0.0s
 => => transferring context: 2B
     0.0s
 => ERROR [internal] load metadata for
     docker.io/randomdude/gcc-cross-x86_64-elf:latest
     0.4s
------
 > [internal] load metadata for
     docker.io/randomdude/gcc-cross-x86_64-elf:latest:
------
Dockerfile:1
--------------------
   1 | >>> FROM randomdude/gcc-cross-x86_64-elf
   2 |
   3 |     RUN apt-get update
--------------------
ERROR: failed to solve: randomdude/gcc-cross-x86_64-elf: error
    getting credentials - err: exit status 1, out: ``
```

Це може бути пов'язане із правами доступу до цього образу.
Цю проблему можна виправити такою командою:

```
docker pull randomdude/gcc-cross-x86_64-elf
```

Та повторити спробу побудови Докер-Зображення.

3. Структура директорії

Структура директорії заишається такою ж, як у попередніх двох уроках за винятком того, що додастся interrupt_table.hpp та у всіх інший .c файлах ми перейдемо на .cpp.

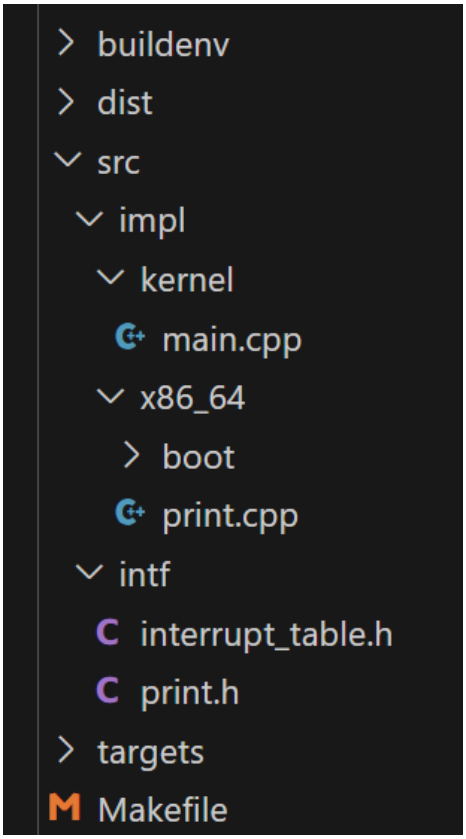

Рис. 2: Структура директорії

4. Код та теорія

Для того, щоб продемонструвати переривання, нам потрібно налаштувати GDT та IDT.

GDT (Global Descriptor Table) - глобальна таблиця дескрипторів, службова структура даних в архітектурі x86, що визначає глобальні (загальні для всіх завдань) сегменти .

```
section .read_only_data
gdt64:
    dq 0 ; zero entry
.code_segment: equ $ - gdt64
    dq (1 << 43) | (1 << 44) | (1 << 47) | (1 << 53) ; code
        segment
.pointer:
    dw $ - gdt64 - 1 ; length
    dq gdt64 ; address
```

Вона має починатись з 0 входу (dq 0), після ми повинні увімкнути потрібні сегменти таблиці дескрипторів:

- **Прапорець P (сегмент присутній) (47 біт):** Показує, чи присутній сегмент у пам'яті (встановлений) або відсутній (скинутий). Якщо цей прапорець чистий, процесор генерує виключення відсутності сегмента (#NP), коли селектор сегмента, що вказує на дескриптор сегмента, завантажується у сегментний регістр.
- **Прапорець S (тип дескриптора) (44 біт):** Визначає, чи є дескриптор сегмента системним сегментом (S очищено), чи сегментом коду або даних (S встановлено).
- **Сегмент виконуваного коду (43 біт):** Прапорець D вказує на довжину за замовчуванням для ефективних адрес та операндів:
  - Якщо прапорець встановлено - використовуються 32-розрядні адреси та 32-розрядні або 8-розрядні операнди.
  - Якщо прапорець не встановлено - використовуються 16-розрядні адреси та 16-розрядні або 8-розрядні операнди.

  Префікс 66H вибирає розмір операнда, відмінний від значення за замовчуванням, а префікс 67H — розмір адреси.
- **Прапорець L (64-бітовий сегмент коду) (53 біт):** У режимі IA-32e біт 21 другого подвійного слова дескриптора визначає, чи сегмент коду містить 64-бітовий код:
  - Значення 1 — інструкції виконуються у 64-бітному режимі.
  - Значення 0 — інструкції виконуються у режимі сумісності.

Якщо встановлено L-біт, D-біт має бути очищений. Біт 21 не використовується поза режимом IA-32e. Уникати завантаження CS із дескриптора з встановленим L-бітом поза режимом IA-32e.

Нарешті треба налаштувати вказівник на цю структуру та завантажити її у gdtr - регістр глобальної таблиці дескрипторів.

Після налаштування Глобальної Таблиці Дескрипторів нам потрібна Таблиця Переривань. Таблиця переривань, або якщо правильніше, Таблиця Описів Переривань (Interrupt Descriptor Table, IDT) використовується в архітектурі x86 і слугує для визначення правильної відповіді на переривання і винятки.

Тобто коли стається переривання код звертається до IDT, щоб дізнатись де в пам'яті знаходиться обробник конкретного переривання та звернутись до нього, щоб той правильно зреагував на переривання.

Структуру нашої Таблиці Описів Переривань ми описуємо у interrupt_table.hpp:

```
struct IDTEntry {
    uint16_t offset_low;      // Lower 16 bits of the handler address
    uint16_t selector;        // Code segment selector in GDT
    uint8_t ist;              // Interrupt Stack Table offset (set to 0 for simplicity)
    uint8_t type_attr;        // Type and attributes (e.g., 0x8E for interrupt gate)
    uint16_t offset_mid;      // Middle 16 bits of the handler address
    uint32_t offset_high;     // High 32 bits of the handler address
    uint32_t zero;            // Reserved (set to 0)

    IDTEntry(void (*handler)(), uint16_t sel, uint8_t attr) {
        uint64_t handler_address =
            reinterpret_cast<uint64_t>(handler);
        offset_low = handler_address & 0xFFFF;
        selector = sel;
        ist = 0;
        type_attr = attr;
        offset_mid = (handler_address >> 16) & 0xFFFF;
        offset_high = (handler_address >> 32) & 0xFFFFFFFF;
        zero = 0;
    }
};
```

У структурі IDTEntry є кілька важливих полів, кожне з яких виконує конкретну роль:

- offset_low, offset_mid, offset_high - це частини адреси обробника переривання. Вони разом формують повну 64-розрядну адресу обробника.
- selector - сегмент коду з GDT (Global Descriptor Table), який використовується для доступу до коду обробника переривання.

- ist - зміщення для визначення стеку, який використовувати під час обробки переривання. За замовчуванням значення дорівнює 0.
- type_attr - тип переривання, який визначає, чи є це перериванням типу *Gate* (пода0x8Ельші переривання блокуються під час виклику обробника для конкретного переривання) або *Trap* (обробка переривань під час якої дозволяються подальші переривання під час виклику обробника для конкретного переривання).
- zero - зарезервовані біти, які за замовчуванням встановлені в 0.

Після опису одного входження в Таблицю Описів Переривань треба описати її розмір та базову адресу:

```
struct IDTDescriptor {
    uint16_t limit;       // Size of the IDT - 1
    uint64_t base;        // Base address of the IDT

    IDTDescriptor(IDTEntry const* baseAddress, uint16_t size) {
        limit = size - 1;
        base = reinterpret_cast<uint64_t>(baseAddress);
    }
} __attribute__((packed));
```

Далі ми визначаємо функцію-обробник переривання. В цьому випадку ми ніяк конкретно не обробляємо переривання, а просто повертаємось до виконання коду за допомогою команди iretq.
iretq - Interrupt Return - Повернення Перривання - Повертає керування програмою від обробника винятків або переривань до програми або процедури, яка була перервана виключенням, зовнішнім перериванням або програмним перериванням.

```
void isr_handler() {
    asm volatile (
        "iretq"
    );
}
```

Після того, як ми описали наші структури нам потрібно туди записати обробник наших переривань в нульовий запис Таблиці Описів Переривань обробник, який буде викликатись для переривання із номером 0 (isr_handler).

- 0x08: це місце де в нашій GDT (яку ми налаштували раніше), де знаходиться код нашого обробника. Воно позначає сегмент коду в GDT.
- 0x8E: тип Interrupt Gate.

Після формування першого запису Таблиці Описів Переривань, нам потрібно його в цю ж таблицю записати:

```
static const IDTEntry idt[1] = {
    IDTEntry(isr_handler, 0x08, 0x8E)
};

static const IDTDescriptor idtDescriptor(idt, sizeof(idt));
```

Також нам потрібна функція для завантаження нашої IDT. Завантаження IDT виконує команда lidt - load IDT - Завантажує значення у вихідному операнді у глобальний регістр таблиці дескрипторів (GDTR) або регістр таблиці дескрипторів переривань (IDTR). "r" означає, що передана йому змінна - регістр, в цьому випадку - із адресою нашої IDT.

```
static void load_idt() {
    asm volatile (
        "lidt (%0)"
        :
        : "r" (&idtDescriptor)
    );
}
```

В нашому основному main.cpp ми викликаємо завантаження IDT, тоді намгаємось вивести повідомлення на екран і викликаємо переривання. Оскільки переривання обробляється тим, що ми просто повертаємось до виконання коду (isr_handler), то ми повертаємось до виводу повідомлення на екран і знову викликаємо переривання, і так по колу.

```
#include "print.h"
#include "interrupt_table.h"

extern "C" {

void kernel_main() {
    static const char *msg = "Welcome to our 64-bit kernel!";
    load_idt();
    print_clear();
    print_set_color(PRINT_COLOR_YELLOW, PRINT_COLOR_BLACK);
    print_str(msg);

    asm volatile (
        "int $0x0\n\t"
        "hlt\n\t"
        "hlt\n\t"
    );
}

}
```

Демонстрації результату цього уроку не буде, тому що .pdf файли не дозволяють завантажувати відео, проте якщо ваша система після запуску коду не може завантажитись - ви все зробили правильно.

Література

[1] Felix Cloutier. *LGDT: LIDT*. Available at: https://www.felixcloutier.com/x86/lgdt:lidt.

[2] Basic Input/Output. *Pardon the Interruption: A Brief History of Interrupts*. Available at: https://www.basicinputoutput.com/2024/05/pardon-interruption-brief-history-of.html.

[3] Wikipedia. *Переривання*. Available at: https://uk.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D1%80%D0%B8%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F.

[4] Felix Cloutier. *IRET: IRETD: IRETQ*. Available at: https://www.felixcloutier.com/x86/iret:iretd:iretq.