Архитектура BIOS Aleste

Дизайн для реального железа

h2w

28 декабря 2025 г.

Содержание

1. Архитектура BIOS Aleste: Дизайн для реального железа

Введение: Зачем нам нужен новый стандарт?

Когда пишешь код для Z80 на реальном железе вроде Aleste, каждая инструкция на счету. Существующие стандарты часто приходят с больших платформ и не учитывают специфику 8-битных систем. Мы создаем не абстракцию ради абстракции, а практичную систему, где:

- Прямой вызов драйвера стоит ровно 17 тактов
- Переключение банков памяти происходит только когда действительно нужно
- Обработчик прерываний аудио не тратит время на сохранение регистров, которые не использует

Этот документ — не просто спецификация. Это философия разработки для ограниченных ресурсов.

1.1. Философия: Скорость через предсказуемость

1. Фиксированные слоты вместо поиска

   Представьте, что вам нужно вывести пиксель на экран. В сложных системах вы бы:

   a) Искали драйвер видео в таблице

   b) Запрашивали функцию вывода

   c) Вызывали её через указатель

   В нашей системе вы просто пишете:

   ```
   call _video_draw_pixel ; Эквивалени call 0xFD03
   ```

   Почему это важно: 17 тактов против 50+. При частоте кадров 50 Гц у вас есть всего 20 мс на кадр. Каждая экономия тактов — это возможность добавить больше спрайтов, больше эффектов.

2. Реальные цифры:

   - Стандартный вызов через фиксированный слот: **17 тактов**
   - Вызов через таблицу указателей: ~50 тактов
   - Динамический поиск в runtime: 100+ тактов

1.2. Структура драйвера: Один файл — одно устройство

Мы отказались от разделения на `_api.asm` и `_impl.asm`. Почему?
Проблема: Когда драйвер раскидан по нескольким файлам:

1. Ты открываешь `video_api.asm`, видишь прототипы
2. Переходишь в `video_impl.asm`, ищешь реализацию
3. Забываешь, где лежат данные драйвера

Наше решение: Открой `drivers/video/crtc.asm` — и увидишь всё:

```
; В начале файла — таблица переходов
SECTION DRV_VTABLE
    ; луче устанавливать allign.
    allign 32
crtc_vtable:
    jp crtc_detect
    jp crtc_install
    ...

; Прямо здесь же — данные драйвера
SECTION DRV_DATA
crtc_mode: db 0
crtc_buffer: ds 256
```

```
; И сразу код
SECTION DRV_CODE
crtc_detect:
    ; проверка наличия CRTC
    ...
```

Результат: Локальность. Меньше прыжков по файлам, меньше ошибок.

1.3. Полиморфные драйверы в BIOS Aleste: Два паттерна использования VTable

В системе существуют два типа драйверов с разными схемами использования VTable:

1. Синглтон-драйверы (статичные, неделимые)

   Это драйверы для уникального железа, которое не имеет альтернатив:

   - **MMU** - только один тип маппера в системе
   - **CRTC** - конкретный видеоконтроллер
   - **PSG** - конкретный звуковой чип

   Структура:

   ```
   ; Единая таблица переходов
   PUBLIC _mmu_vtable, _mmu_detect, _mmu_install

   _mmu_vtable:
   _mmu_detect:    jp mmu_detect_impl
   _mmu_install:   jp mmu_install_impl
   ```

   Использование:

   - В заголовочном файле: прямые вызовы по именам меток
   - В коде: `call _mmu_install`
   - VTable используется только системой при загрузке

   ```
   bool mmu_detect(); // direct call _mmu_detect
   ```

2. Полиморфные драйверы (многорежимные, заменяемые)

   Это драйверы, имеющие несколько реализаций для одного интерфейса:

   - **Видео** - разные видеорежимы (текст, графика)
   - **Консоль** - разные кодировки или шрифты
   - **Файловая система** - FAT12, FAT16, TR-DOS

   a) Структура:

   ```
   ; Полиморфная таблица верхнего уровня
   PUBLIC _video_vtable, _video_draw_char, _video_set_mode

   _video_vtable:
   _video_draw_char:    jp 0          ; Заполняется при активации
   ```

```
_video_set_mode:     jp 0          ; Заполняется при активации

; Конкретная реализация например(, текстовый режим)
PUBLIC _text_vtable
_text_vtable:
    jp text_draw_char_impl
    jp text_set_mode_impl
```

b) Процесс активации:

Конкретная реализация может иметь или не иметь метки, так как чаще всего будет использованп полиморфная таблица. Для этого, когда драйверу устанавливается режим он копирует конкретную таблицу в полиморфную версию.

```
; Система копирует таблицу конкретного драйвера в слот
copy_driver_table:
    ; HL = source (_text_vtable)
    ; DE = destination (_video_vtable)
    ld bc, 15        ; 5 методов × 3 байта
    ldir
```

Использование:

c) Ключевые отличия:

- В заголовочном файле: всё те же прямые вызовы `call _video_draw_char`
- Runtime: вызов автоматически идет в текущую активную реализацию
- Переключение: копирование VTable без изменения вызывающего кода

Аспект	Синглтон-драйвер	Полиморфный драйвер
Количество реализаций	1	Много
VTable в памяти	Фиксированная, никогда не меняется	Меняется при переключении режима
Вызов из кода	Прямой переход по фиксированному адресу	Прямой переход по адресу, который меняет реализацию
Заголовочный файл	Содержит фиксированные адреса методов	Содержит адреса полиморфного слота
Примеры	MMU, таймер, PSG	Видео, консоль, FS

d) Философия:

i. **Единый интерфейс вызова** - код приложения всегда вызывает `_video_draw_char`, независимо от режима

ii. **Нулевые накладные расходы** - вызов всегда 17 тактов, даже при полиморфизме

iii. **Простота переключения** - сменить драйвер = скопировать 15 байт

iv. **Прозрачность** - отладчик видит прямой jump, а не косвенный вызов через регистр

Именно поэтому в документации говорится <<прямой вызов драйвера стоит ровно 17 тактов>> - потому что даже полиморфный вызов это прямой `jp` на конкретный адрес, просто этот адрес может меняться в runtime.

Полиморфизм достигается не через таблицы указателей в памяти (что давало бы +50 тактов), а через физическое копирование инструкций перехода. Это гениально просто для Z80.

1.4. Table API: Минимум обязательного, максимум возможного

1. Обязательные методы (первые 15 байт)

   Каждый драйвер обязан реализовать 5 методов:

Метод	Что делает	Когда вызывается
`detect`	Проверяет, присутствует ли устройство	При загрузке и (опционально) периодически
`install`	Инициализирует устройство	После успешного detect
`get_info`	Возвращает информацию о драйвере	По запросу системы или приложения
`command`	Универсальная точка расширения	Когда нужно специфичное действие
`uninstall`	Освобождает ресурсы	При выгрузке драйвера

2. Магия команды `command`

   Вместо того чтобы расширять таблицу прыжков для каждой новой функции:

```
; ПЛОХО: Таблица раздувается
jp video_set_mode
jp video_set_palette
jp video_set_sprite
jp video_scroll
... и ещё 20 функций

; ХОРОШО: Один метод на всё
jp video_command
```

   Как это работает:

```
; Установить видеорежим
ld a, CMD_VID_SET_MODE
ld hl, MODE_256x192
call 0xFD00  ; video_command

; Прокрутить экран
ld a, CMD_VID_SCROLL
ld bc, 10  ; на 10 пикселей
call 0xFD00
```

   Преимущество: Таблица прыжков всегда занимает 15 байт + специфичные методы. Не нужно резервировать слоты под функции, которые драйвер не поддерживает.

1.5. Типизация команд: Избегаем хаоса

1. Диапазоны команд

   Чтобы команды разных драйверов не конфликтовали:

```
0x00-0x0F: Системные все( драйверы)
  0x00 = RESET
  0x01 = STATUS
  0x02 = SUSPEND
  0x03 = RESUME

0x10-0x1F: Видео
  0x10 = SET_MODE
  0x11 = SET_PALETTE
  0x12 = SCROLL

0x20-0x2F: Аудио
  0x20 = PLAY
  0x21 = STOP
  0x22 = SET_VOLUME

0x30-0x3F: Хранилище
  0x30 = READ_SECTOR
  0x31 = WRITE_SECTOR
```

   Пример использования:

```
; Играть музыку
ld a, CMD_AUD_PLAY  ; 0x20
ld hl, song_data
call 0xFD00  ; audio_command

; Остановить
ld a, CMD_AUD_STOP  ; 0x21
call 0xFD00
```

1.6. Информация о драйвере: Система должна знать, с кем имеет дело

1. Структура DRIVER_INFO

```
; Что возвращает get_info
driver_info:
  dw .name        ; "YM2149 Audio Driver"
  db 2, 1         ; Версия 2.1
  dw API_HAS_IRQ | API_STATIC_DEVICE  ; Флаги
  db IRQ_VECTOR   ; Номер вектора прерывания если( нужно)
  db 0            ; Резерв
.name:
  db "YM2149 v2.1", 0
```

2. Флаги возможностей:

```
API_HAoS_IRQ        equ 1 << 0  ; Использует прерывания
API_STATIC_DEVICE  equ 1 << 1  ; Не исчезнет встроенное( устройство)
API_USES_DMA       equ 1 << 2  ; Требует DMA
API_BANKED         equ 1 << 3  ; Работает с банковой памятью
API_POWER_SAVE     equ 1 << 4  ; Поддерживает энергосбережение
```

Зачем это нужно: Система видит флаг `API_STATIC_DEVICE` и понимает — этот видеоконтроллер встроенный, не нужно опрашивать его каждые 100 мс на предмет <<не исчез ли он>>.

1.7. Горячее обнаружение: Для тех, кто может исчезнуть

1. Два типа устройств
   a) **Статические** (встроенные): Видеоконтроллер, системный таймер
      - `detect` вызывается один раз при загрузке
      - Флаг `API_STATIC_DEVICE` установлен
   b) **Динамические** (сменные): Картриджи, внешние устройства
      - `detect` может вызываться периодически
      - Система отслеживает их наличие
2. Протокол обнаружения исчезновения

```
; Псевдокод ядра
check_hotplug:
    ; Для каждого динамического устройства
    call driver_detect
    jr z, .device_present

    ; Устройство исчезло!
    call driver_uninstall
    mark_slot_free

.device_present:
    ; Всё на месте
    ret
```

Важный момент: `uninstall` должен быть идемпотентным. Если вызвать его дважды — ничего страшного не случится.

1.8. Прерывания: Быстро или безопасно — выбор разработчика

1. Проблема стандартных обработчиков

   Типичный обработчик сохраняет ВСЕ регистры:

```
irq_handler:
    push af
    push bc
    push de
    ...
```

```
    ; 100+ тактов только на push/pop!
    pop hl
    pop de
    pop bc
    pop af
    ret
```

Для аудиодрайвера, который должен обновлять буфер каждые 20 мс, это неприемлемо.

2. Наше решение: Гибкость

   Драйвер регистрирует обработчик как есть. Если он написан на ассемблере и знает, какие регистры использует:

```
; Быстрый обработчик для YM2149
ym_irq_handler:
    push af
    push hl

    ; Обновляем только регистры YM
    ld hl, ym_buffer
    ld a, (hl)
    out (YM_PORT), a

    pop hl
    pop af
    ei
    ret
```

   Но предупреждение: Если обработчик написан на C или использует библиотечные функции — он ДОЛЖЕН сохранять все регистры. Система доверяет разработчику.

3. 7.3 Регистрация обработчика

```
; В install драйвера:
ld a, CMD_SYS_IRQ_ATTACH
ld hl, ym_irq_handler
ld bc, IRQ_VECTOR
call SYS_CALL  ; зарегистрировать
```

1.9. Зависимости: Явное лучше неявного

1. Декларация зависимостей

   В начале файла драйвера:

```
; drivers/console/text.asm
DEPENDS_ON:
    db "VIDEO", 0    ; Нужно для вывода
    db "KEYBOARD", 0 ; Нужно для ввода
    db 0             ; Конец списка
```

2. Что происходит при загрузке

a) Система читает `DEPENDS_ON` каждого драйвера

b) Проверяет, все ли зависимости доступны

c) Если нет — выводит понятную ошибку:

```
CONSOLE: Missing dependency: KEYBOARD
```

Преимущество: Не нужно в runtime проверять <<а подключена ли клавиатура?>>. Система знает это заранее.

1.10. Контекст памяти: Lazy switching

1. Проблема банковой памяти

   На Z80 с MMU переключение банков — дорого:

   ```
   ; Сохранить текущую конфигурацию
   ld a, (current_bank)
   push af

   ; Переключить на банк драйвера
   ld a, DRIVER_BANK
   out (MMU_PORT), a

   ; Выполнить операцию
   call driver_function

   ; Восстановить
   pop af
   out (MMU_PORT), a
   ```

   ~50 тактов на переключение туда-обратно!

2. Наше решение: Кэширование

   Каждый драйвер хранит свой <<отпечаток пальца>>:

   ```
   ; Memory Map драйвера
   crtc_mem_map:
       db 0xA3        ; Хэш конфигурации
       db 5, 6, 7, 8  ; Банки для окон
   ```

   Перед вызовом драйвера:

   ```
   require_memory:
       ; Сравнить хэш текущей конфигурации с хэшем драйвера
       ld a, (current_hash)
       cp (hl)  ; HL указывает на crtc_mem_map
       ret z    ; Уже правильная конфигурация!

       ; Переключить банки
       inc hl
       ld bc, MMU_PORT
       outi outi outi outi
   ```

```
    ; Обновить хэш
    ld (current_hash), a
    ret
```

Результат: Если два вызова идут к одному драйверу подряд — переключение произойдет только при первом вызове.

1.11. Соглашения о регистрах: Естественность для Z80

1. Передача параметров

```
; HL, DE, BC — в порядке приоритета
ld hl, buffer    ; Основной параметр
ld bc, 256       ; Дополнительный
call video_fill

; Иногда только A
ld a, COLOR_RED
call video_set_color
```

2. Возврат результата и ошибок

Гениальность в простоте:

```
; Успех: A=0, Carry сброшен
xor a  ; A=0, Carry=0
ret

; Ошибка: Акод=, Carry установлен
ld a, ERR_TIMEOUT
scf    ; Set Carry Flag
ret
```

Почему это удобно:

```
call disk_read
ret c           ; Выход при ошибке
; Продолжаем если OK

; Или цепочка вызовов:
call init_video
call c, .error
call init_audio
call c, .error
call init_input
call c, .error
```

Стандартные коды ошибок:

```
ERR_SUCCESS      equ 0
ERR_NOT_SUPPORTED equ 1
ERR_TIMEOUT      equ 4
ERR_NO_DEVICE    equ 11   ; Устройство отсутствует
```

1.12. Правила именования констант

#+END_SRC

1. Уровни иерархии (слева направо):

```
УСТРОЙСТВО
[]_ТИП[]_ИМЯ[]_КВАЛИФИКАТОР[]_СУФФИКС[]
```

   Пример: MMU_PORT_CTRL_SUPER_BIT

2. Устройство (Device) - всегда первый компонент:

```
MMU_      - Memory Management Unit
VIDEO_    - Видеоконтроллер
AUDIO_    - Звуковой чип
CRTC_     - Видеоконтроллер 6845
PALETTE_  - Палитра цветов
PSG_      - Программируемый генератор звука
FDC_      - Контроллер дисковода
DMA_      - DMA контроллер
RTC_      - Часы реального времени
SYS_      - Системные общие()
```

3. Тип (Type) - что это за сущность:

```
PORT_     - Порты вводавывода- (I/O address)
REG_      - Регистры в памяти (MMIO address)
CMD_      - Команды для( commandметода-)
FLAG_     - Флаги возможностей
ERR_      - Коды ошибок
BUF_      - Буферыобласти/ памяти
IRQ_      - Прерываниявекторы/
```

4. Имя (Name) - конкретный элемент:

```
CTRL      - Управлениеконтроль/
STATUS    - Статус
DATA      - Данные
ADDR      - Адрес
INDEX     - Индекс
CLOCK     - Тактовая частота
BANK      - Банк памяти
MODE      - Режим работы
CONFIG    - Конфигурация
```

5. Квалификатор (Qualifier) - уточнение:

```
SUPER     - Супервизорпривилегированный/
USER      - Пользовательский
NATIVE    - Нативный режим
LEGACY    - Совместимость с CPC
ENABLE    - Включение
DISABLE   - Выключение
```

```
RESET     - Сброс
INIT      - Инициализация
```

6. Суффикс (Suffix) - тип значения:

```
_BIT      - Одиночный бит (0x01, 0x02, 0x04...)
_MASK     - Битовая маска (0x0F, 0xF0...)
_SIZE     - Размер в байтах
_COUNT    - Количество элементов
_BASE     - Базовый адресзначение/
_OFFSET   - Смещение
_MIN      - Минимальное значение
_MAX      - Максимальное значение
_DEFAULT  - Значение по умолчанию
```

Примеры правильного именования: Порты ввода-вывода:

```
MMU_PORT_CTRL       equ 0xF0    ; Порт управления MMU
MMU_PORT_CLOCK      equ 0xF3    ; Порт управления частотой
MMU_PORT_BANK0      equ 0xFC    ; Порт банка окна 0
MMU_PORT_BANK1      equ 0xFD    ; Порт банка окна 1

VIDEO_PORT_PALETTE  equ 0x??    ; Порт палитры
AUDIO_PORT_PSG      equ 0x??    ; Порт PSG
```

Значения для портов (биты/маски):

```
; Для MMU_PORT_CTRL
MMU_CTRL_SUPER_BIT  equ 0x01    ; Бит супервизора
MMU_CTRL_NATIVE_BIT equ 0x02    ; Бит нативного режима
MMU_CTRL_TRAP_BIT   equ 0x04    ; Бит захвата прерываний
MMU_CTRL_USERLOCK_BIT equ 0x10  ; Бит блокировки User MMIO

MMU_CTRL_MODE_MASK  equ 0x03    ; Маска режимов биты( 0-1)
MMU_CTRL_DEFAULT    equ MMU_CTRL_SUPER_BIT | MMU_CTRL_NATIVE_BIT

; Для MMU_PORT_CLOCK
MMU_CLOCK_DIV2      equ 0x02    ; Делитель 2
MMU_CLOCK_DIV4      equ 0x04    ; Делитель 4
MMU_CLOCK_CPC_BIT   equ 0x10    ; Бит режима CPC

MMU_CLOCK_DIV_MASK  equ 0x0F    ; Маска делителя биты( 0-3)
```

7. MMIO регистры (через MMIO_WINDOW):

```
; При MMIO_PAGE = 0x02
PALETTE_REG_INDEX   equ 0x00    ; Регистр индекса цвета
PALETTE_REG_DATA_LO equ 0x01    ; Регистр данных мл(. байт)
PALETTE_REG_DATA_HI equ 0x02    ; Регистр данных ст(. байт)
PALETTE_REG_CTRL    equ 0x05    ; Регистр управления

CRTC_REG_ADDR       equ 0x10    ; Регистр адреса CRTC
CRTC_REG_DATA       equ 0x11    ; Регистр данных CRTC
```

Команды для драйверов:

```
; Базовые команды все( драйверы)
DRIVER_CMD_RESET    equ 0x00
DRIVER_CMD_STATUS   equ 0x01

; Специфичные команды
MMU_CMD_GET_TOTAL   equ 0x60    ; В диапазоне MMU
MMU_CMD_GET_FREE    equ 0x61

VIDEO_CMD_SET_MODE  equ 0x10    ; В диапазоне VIDEO
VIDEO_CMD_SET_PALETTE equ 0x11

AUDIO_CMD_PLAY      equ 0x20    ; В диапазоне AUDIO
AUDIO_CMD_STOP      equ 0x21
```

Флаги возможностей драйверов:

```
DRIVER_FLAG_IRQ_BIT     equ 1 << 0  ; Использует прерывания
DRIVER_FLAG_STATIC_BIT  equ 1 << 1  ; Статическое устройство
DRIVER_FLAG_DMA_BIT     equ 1 << 2  ; Использует DMA
DRIVER_FLAG_BANKED_BIT  equ 1 << 3  ; Работает с банками

; Пример использования
MMU_INFO_FLAGS      equ DRIVER_FLAG_STATIC_BIT | DRIVER_FLAG_BANKED_BIT
```

Области памяти:

```
MEMORY_WINDOW0_BASE equ 0x0000  ; Базовый адрес окна 0
MEMORY_WINDOW0_SIZE equ 0x4000  ; Размер окна 0 K(16)

MMIO_LO_BASE        equ 0xFF0000 ; Базовый адрес MMIO_LO
MMIO_LO_SIZE        equ 0x4000   ; Размер MMIO_LO K(16)

ROM_BIOS_BASE       equ 0xFF0000 ; Базовый адрес BIOS
ROM_BIOS_SIZE       equ 0x8000   ; Размер BIOS K(32)
```

Специальные случаи:

a) Для очень длинных имён можно опустить некоторые уровни:

```
; Вместо: MMU_PORT_MAPPER_SLOT0_WINDOW0
MMU_PORT_SLOT0_WIN0 equ 0xE0    ; Достаточно ясно

; Вместо: VIDEO_CRTC_REG_HORIZONTAL_TOTAL
CRTC_REG_HTOTAL     equ 0x00    ; CRTC уже подразумевает VIDEO
```

b) Legacy/совместимые порты (CPC):

```
CPC_PORT_RMR        equ 0x7F00  ; CPC RMR регистр
CPC_PORT_UPPER_ROM  equ 0xDF00  ; CPC выбор верхнего ROM
CPC_PORT_SYSCALL    equ 0xF200  ; Legacy системный вызов
```

8. Системные/общие константы (без префикса устройства):

```
; Ошибки уже( есть ERR_ префикс в errors.inc)
ERR_SUCCESS         equ 0x00
ERR_NOT_SUPPORTED   equ 0x01

; Системные вызовы
SYS_CALL_BASE       equ 0xF2    ; Базовый порт syscall
SYS_CALL_MAX        equ 255     ; Максимальный номер вызова
```

9. Правила в одном предложении:

   Начинай с устройства (MMU_), затем тип (PORT_/REG_/CMD_), затем имя, уточняй квалификатором, заканчивай суффиксом типа значения.

1.13. Правило канона Aleste BIOS для констант и импорта:

1. Правило: Разделение определений и объявлений

   a) ВСЕ определения констант через defc делаются ТОЛЬКО ВНУТРИ самого драйвера (.asm файл)

   b) Заголовочный файл (.inc) содержит ТОЛЬКО:

   - extern объявления для публичных символов драйвера
   - equ или = константы (директивы ассемблера), которые можно безопасно дублировать
     - Чисто числовые значения (0xFD00, 255 и т.д.)
     - Простые вычисления (PORT_BASE + 3)
     - Без создания новых символов для линкера
   - Макросы для удобства использования (но не нарушая KISS)

2. Важное различие:

   - equ/=' = директива АССЕМБЛЕРА, заменяется на этапе ассемблирования
   - defc = директива ЛИНКЕРА, создает уникальный символ в объектном файле

   В .inc файлах используйте ТОЛЬКО equ/=, НИКОГДА defc

3. Правило: Структура заголовочного файла (.inc)

```
; example.inc
module example

; 1. ТОЛЬКО extern объявления
extern _driver_init, _driver_process

; 2. ТОЛЬКО КОНСТАНТЫ ДЛЯ АССЕМБЛЕРА не( defc!)
;    Используйте 'equ' или '=' для чисто числовых значений
;    которые не требуют уникальных символов в линковке
DRIVER_SLOT        equ 0xFD00
CMD_INIT           equ 0x00
CMD_PROCESS        equ 0x01
```

```
; 3. ТОЛЬКО макросы
macro DRIVER_CALL addr
    call addr
endm
```

4. Правило: Структура файла драйвера (.asm)

   defc используется ТОЛЬКО в .asm файлах для:

   - Создания символов линкера (адреса функций, переменных)
   - Когда значение должно быть уникальным в итоговом бинарнике
   - Когда символ будет использоваться через extern в других модулях

   #+BEGIN_SRC asm ; example.asm module example_driver

   ; Включаем ТОЛЬКО equ-константы include <<palette_const.inc>> ; если нужны общие числовые константы

   ; 1. Включаем константы (если нужны) ; 2. Определяем ВСЕ константы defc defc _driver_const_a = 0xFD00 defc _driver_const_b = 0xFD03

   ; 3. Экспортируем публичные символы public _driver_init, _driver_process public _driver_vtable, DRIVER_VTABLE_ADDR

   ; 4. Определяем данные и код section driver_data _driver_vtable: _driver_init: jp driver_init _driver_process: jp driver_process

   ; 5. Внешние зависимости (если есть) extern _some_external_func

   ; 6. Объявления PUBLIC public _foo _foo: ret public _bar _bar: ret #+END_#+END_SRC

5. Краткая формулировка для канона:

   В .inc файлах — только extern, equ и макросы. Все defc определения — строго в .asm файлах драйверов. Equ для числовых констант, defc для символов линкера.

   Такой подход исключает переопределения и делает архитектуру предсказуемой и надежной.

1.14. Важнейшее правило KISS

Лучше не объявлять и реализовывать функцию, переменную, константу или макрос чем реализовать не понятно зачем и для чего KISS

2. Пример из реальной жизни: Аудиодрайвер для YM2149

```
; ===========================================================================
; Драйвер звукового чипа YM2149
; ===========================================================================

SECTION DRV_VTABLE
ym2149_vtable:
    jp ym_detect      ; +0
    jp ym_install     ; +3
    jp ym_get_info    ; +6
```

```
    jp ym_command     ; +9
    jp ym_uninstall   ; +12
    jp ym_play_note   ; +15 специфичный( метод)

; ---------------------------------------------------------------------
; Зависимости: нет, самостоятельное устройство
; ---------------------------------------------------------------------
DEPENDS_ON:
    db 0

; ---------------------------------------------------------------------
; Данные
; ---------------------------------------------------------------------
SECTION DRV_DATA

ym_mem_map:
    db 0xC1        ; Хэш конфигурации
    db 2, 3, 4, 5  ; Банки памяти

ym_info:
    dw .name
    db 1, 2        ; Версия 1.2
    dw API_HAS_IRQ | API_STATIC_DEVICE
    db 0x38        ; Вектор прерывания
.name:
    db "YM2149 PSG v1.2", 0

; ---------------------------------------------------------------------
; Код драйвера
; ---------------------------------------------------------------------
SECTION DRV_CODE

ym_detect:
    ; Проверяем наличие YM2149
    call require_memory

    ; Пробуем записатьсчитать/ тестовое значение
    ld a, 0xFF
    out (YM_REG), a
    in a, (YM_REG)
    cp 0xFF
    jr nz, .not_found

    ; Устройство найдено
    ld a, 1
    ret

.not_found:
    xor a
    ret

ym_command:
    ; Обработка команд IOCTL
```

```
    cp CMD_RESET
    jr z, .reset
    cp CMD_AUD_PLAY
    jr z, .play_cmd
    cp CMD_AUD_STOP
    jr z, .stop_cmd

    ; Неизвестная команда
    ld a, ERR_NOT_SUPPORTED
    scf
    ret

.reset:
    ; Сброс звукового чипа
    xor a
    ld bc, 13
    ld hl, ym_registers
.reset_loop:
    out (c), a
    inc c
    djnz .reset_loop
    xor a
    ret

.play_cmd:
    ; HL = данные ноты
    ld a, (hl)
    out (YM_FREQ_A), a
    inc hl
    ld a, (hl)
    out (YM_FREQ_A_HI), a
    xor a
    ret

; Быстрый обработчик прерываний
; Сохраняет только то, что использует
ym_irq_handler:
    push af
    push hl

    ; Обновляем частоты
    ld hl, (ym_freq_ptr)
    ld a, (hl)
    out (YM_FREQ_A), a

    ; Следующий байт
    inc hl
    ld (ym_freq_ptr), hl

    pop hl
    pop af
    ei
    ret
```

3. Структура проекта: Как это всё собирается

```
# Правильная структура для проекта
XiAlesteBIOS/├──
 include/                    ← для C программ│
   ├── aleste.h               ← главный для C│
   ├── bios/                  ← подсистемы BIOS (C)│
   │   ├── drivers.h          ← DriverInfo, флаги (C)│
   │   ├── errors.h           ← ERR_SUCCESS (C)│
   │   └── system.h           ← системные вызовы (C)│
   └── drivers/               ← публичные API драйверов (C)│
       ├── mmu.h│
       └── video.h│├──

 inc/                        ← для ассемблера те( же константы)│
   ├── aleste.inc             ← главный для ASM│
   ├── bios/                  ← подсистемы BIOS (ASM)│
   │   ├── drivers.inc        ← DRV_FLAG_* (ASM эквивалент)│
   │   ├── errors.inc         ← ERR_SUCCESS equ (ASM)│
   │   └── system.inc         ← системные вызовы (ASM)│
   ├── drivers/               ← интерфейсы драйверов (ASM)│
   │   ├── mmu.inc            ← метки методов│
   │   └── video.inc│
   └── hardware/│
       ├── ports.inc│
       └── memory.inc│└──

 drivers/                    ← РЕАЛИЗАЦИЯ драйверов
    ├── mmu/
    │   ├── mmu.asm            ← include "../../inc/bios/drivers.inc"
    │   └── mmu_private.inc    ← приватные константы
    └── video/
        ├── video.asm          ← include "../../inc/bios/drivers.inc"
        └── video_private.inc
```

Ключевое правило:

- include/ - что видят ВНЕШНИЕ программы (твой main.c и другие приложения)
- drivers/ - внутренняя реализация (собирается в библиотеку BIOS)