# Архитектура BIOS Aleste

**Дизайн для реального железа**

h2w

29 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.

   Но не в коем случае не вызвать математикой или константой

   ```
   ;; очень плохо
   call 0xFD40
   ;; очень плохо
   call DRV_BASE + 3
   ```

2. Реальные цифры:

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

## 1.2. Важнейшее правило KISS

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

## 1.3. Принцип: 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
    public _crtc_vtable
_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.

### 1.4. Принцип: Один файл — одна ответственность

Три четких уровня:

```
Уровень
 1: Синтерфейс- (palette.h)
          ↓Уровень
 2: Полиморфная таблица + диспетчер (palette.asm + palette.inc)
          ↓Уровень
 3: Конкретные реализации (palette_native.asm, palette_cpc.asm, ...)
```

**Иначе говоря**

- palette.h - только интерфейсы
- palette.c - только интерфейсы реализация
- palette.inc - только константы и файл допускает множественное включение
- palette.asm - ТОЛЬКО полиморфная система
- palette_native.asm - конкретная реализация

**Диаграмма зависимостей**:

```
                palette.h/c (C интерфейс)
                      ↑
                palette.inc константы()
               ↗             ↖
palette.asm полиморфный()  palette_native.asm драйвер()
    ↓                            ↓системный
 слот              VTable драйвера
  0xFD40                    экспортируется()
```

### 1.5. Трёхслойная архитектура драйверов

**Слой 1: C-интерфейс (.h + .c)**

- .h файл: объявления функций для C программ
- .c файл: преобразование вызовов из стека в регистры
- Имена: без префиксов в C, с '_' для линкера

**Слой 2: VTable интерфейс (.asm)**

- Полиморфная VTable с методами '__*'
- Фиксированные адреса для прямых вызовов
- Только переходы на реализации

**Слой 3: ASM реализация (.asm)**

- Полные имена без лишеих префиксов или суффиксов
- Работают напрямую с регистрами
- Быстрые, без накладных расходов

### 1.6. Соглашения вызовов:

Для драйвера палитры:

1. C программа: palette_set_color(10, 0xFFF);
2. C-интерфейсный слой:
    - Извлекает 10 и 0xFFF из стека
    - Кладёт в регистры
    - Вызывает __palette_set_color
3. VTable: __palette_set_color: jp palette_set_color
4. ASM реализация: palette_set_color: ; A=10, BC=0xFFF уже здесь! ; ... быстрая реализация ...

### 1.7. Полиморфные драйверы: Два паттерна использования VTable

В системе существуют два типа драйверов с разными схемами использования VTable. В одном случе где драйвер-синглетон и предоставляет свою таблицу внешним модулям. В другом случае полиморфный драйвер который подменяет методы драйвера в зависимости от режима.

1. Синглтон-драйвер

   Это драйверы для уникального железа, которое не имеет альтернатив:

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

   **Структура**:

   К методу стоит домавть префикс __ что означает виртулаьный, или часть vtable

   ```
   ; 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) 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.

b) Структура:

```
; ПРАВИЛЬНЫЙ ПОДХОД: объявляем метки прямо в VTable
PUBLIC _video_draw_char, _video_set_mode, ...
_video_vtable:
_video_draw_char:    jp 0          ; Адрес = _video_vtable + 0
_video_set_mode:     jp 0          ; Адрес = _video_vtable + 3
    вычисляется( автоматически!)
; ... и так далее ...
SRCHe

 делайте лишних вычислений через defc:

#+BEGIN_SRC asm
; НЕПРАВИЛЬНЫЙ ПОДХОД нарушает( KISS):
defc _video_draw_char = _video_vtable + 0   ; Лишнее вычисление!
defc _video_set_mode = _video_vtable + 3    ; Ассемблер сделает это сам!
```

Для конкретной реализации драйвера например video_monochrome метки методов не обязательны. Так как они не могут использоваться напрямую.

#+BEGIN_SRC asm ; ПРАВИЛЬНЫЙ ПОДХОД: объявляем метки прямо в VTable PUBLIC _video_draw_char, _video_set_mode, ... _video_mono_vtable: _video_mono_draw_char: jp video_mono_draw_char ; Адрес = _video_vtable + 0 _video_mono_set_mode: jp video_mono_set_mode ; Адрес = _video_vtable + 3 (вычисляется автоматически!) ; ... и так далее ... SRC

**Правило для меток VTable**:

i. Объявляйте метки методов прямо в определении VTable

ii. Ассемблер автоматически вычислит правильные адреса

iii. Не используйте defc для вычисления смещений внутри одного модуля

iv. defc используйте ТОЛЬКО для:

- Экспорта констант в другие модули
- Определения абсолютных адресов (если нужно)
- Создания псевдонимов (алиасов)

c) Процесс активации:

3. Процесс активации драйвера

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.

a) Фиксированный размер VTable

Каждая VTable имеет ФИКСИРОВАННЫЙ размер:

- 5 обязательных методов × 3 байта = 15 байт
- + N специфичных методов × 3 байта

Пример для палитры (8 методов): defc PALETTE_VTABLE_SIZE = 24 ; 8 × 3

Это позволяет использовать простой ldir без вычислений.

b) Автоматическая инициализация: ДА или НЕТ?

Два подхода:

i. Активация БЕЗ инициализации:

```
; Программист сам решает когда инициализировать
call __palette_activate_driver
; ... чтото- делаем ...
call __palette_install      ; Явный вызов когда нужно
```

i. Активация С инициализацией:

```
; Удобно для большинства случаев
call _palette_activate_and_init
```

Правило: Желательно исполльзовать ручную активацию и вот почему. Она позволяет сделать вызхвать методы драйвера до активации, например детектирование аппаратных возможноестей и т.д.

c) Размер VTable в коде vs вычисление

Размер таблицы лучше вычислять

```
; ХОРОШО
ld bc, _palette_vtable_end - _palette_vtable
```

d) Пример выбора драйвера

```
; Активировать конкретный драйвер палитры
; Вход: HL = адрес VTable конкретного драйвера
PUBLIC palette_activate_driver
palette_activate_driver:
    ld de, palette_vtable
    ld (current_driver), de
    ld bc, palette_vtable_end-palette_vtable
    ldir
    ret

; Получить текущую активную VTable
; Выход: HL = адрес текущей VTable
PUBLIC palette_get_current_vtable
palette_get_current_vtable:
    ld hl, (current_driver)
    ret

SECTION DRV_DATA
current_driver: dw 0x0000
```

**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 |

4. Метод по упочанию в пустой таблице

   Полиморфная таблица должна иметь значение по умолчанию _undefined_method

```
__vbuf_draw_pixe: JP _undefined_method
__vbuf_draw_line: JP _undefined_method
```

   Этот метод долже вывести сообщение об ошибке со стек трейсом. Впрочем стек трейс в связи со сложностью качественной реализации не так важен. Самое главное система должа прервать исполнение и сообщить максимум возможного.

## 1.8. Driver API: Минимум обязательного, максимум возможного

1. Обязательные методы

   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.

## 1.9. 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
```

## 1.10. Информация о драйвере (driver info)

1. Информация о драйвере

```
; 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>>.

## 1.11. Горячее обнаружение устройств

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.

## 1.12. Прерывания: Быстро или безопасно

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. 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. Регистрация обработчика

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

## 1.13. Зависимости между драйверами

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.

## 1.14. 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. 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.

## 1.15. Соглашения о регистрах

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
```

## 1.16. 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.

### 1.17. 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
   - Макросы для удобства использования (но не нарушая KISS)

   c) Заголовочный файл (.inc) не должее содержать:

   - equ или = константы (директивы ассемблера) так как они приводять к ошибкам редекларации

   **Важное различие**:

   - equ директива АССЕМБЛЕРА, заменяется на этапе ассемблирования
   - defc директива ЛИНКЕРА, создает уникальный символ в объектном файле
   - В .inc файлах используйте extern и макросы, но никогда equ, =, defc

2. 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
```

3. 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

4. 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.

5. Соглашения о вызовах: C vs Assembly

   a) Правило 1: Разделение интерфейсов

      - C программы используют .h файлы и вызывают C функции

- Ассемблерные программы используют .inc файлы и вызывают ассемблерные функции напрямую

b) Правило 2: Два слоя реализации

Для каждого драйвера должно быть:

i. C-слой (.c файл) - преобразует вызовы из стека в регистры

ii. ASM-слой (.asm файл) - работает только с регистрами

c) Правило 3: Чёткая документация

В .h файле должно быть ясно указано:

- Какие функции для C программ
- Какие inline-функции для ассемблера
- Где какие параметры ожидаются

d) Пример вызова:

```
// C код:
palette_set_color(10, 0xFFF);

// Преобразуется в:
// 1. Кладёт 10 и 0xFFF в стек
// 2. Вызывает Cфункцию- из palette.c
// 3. Та извлекает из стека в A и BC
// 4. Вызывает _palette_set_color ассемблер()

; Ассемблерный код:
ld a, 10
ld bc, 0xFFF
call _palette_set_color  ; Или через макрос
```

## 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
ym2149_vtable_end:

; ---------------------------------------------------------------------------
; 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
```

---

Тогда виртуальный драйвер

---

```
SECTION DRV_VTABLE
psg_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
psg_vtable_end:
```

---

## 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)

# 4. Дополнения

## 4.1. ports_aleste.asm

```
; ============================================================================
; MEMORY_ALESTE@INC - Карта памяти Aleste LX с режимами Supervisor/User
; Соответствует спецификации MMU Architecture v2@0
; ============================================================================

; ============================================================================
; БАЗОВЫЕ КОНСТАНТЫ ПАМЯТИ
; ============================================================================

; Адресное пространство (24 бита)
MEMORY_SIZE        equ 0x01000000  ; 16 МБ общее адресное пространство
PHYSICAL_RAM_SIZE  equ 0x00100000  ; 1 МБ физической оперативной памяти

; Базовые адреса
RAM_BASE           equ 0x00000000  ; Начало оперативной памяти
MMIO_BASE          equ 0x00FF0000  ; Начало MMIO пространства последние( К64)

; ============================================================================
; ОКОННАЯ БАНКОВАЯ ПАМЯТЬ (WINDOWED BANK MEMORY)
; ============================================================================

; Общее оконное пространство до( MMIO)
WB_MEMORY_BEGIN    equ 0x000000    ; Начало оконной памяти
WB_MEMORY_END      equ 0xFEFFFF    ; Конец оконной памяти перед( MMIO)
WB_MEMORY_SIZE     equ 0xFF0000    ; Размер: МБ16 - К64

; Логические окна по( К16 каждое)
WB_WINDOW_SIZE     equ 0x00004000  ; 16 КБ на окно

; Окно 0: 0000-3FFF
WB_WINDOW0_BEGIN   equ 0x000000
WB_WINDOW0_END     equ 0x003FFF
WB_WINDOW0_DEFAULT equ 0           ; Банк по умолчанию

; Окно 1: 4000-7FFF
WB_WINDOW1_BEGIN   equ 0x004000
WB_WINDOW1_END     equ 0x007FFF
WB_WINDOW1_DEFAULT equ 1

; Окно 2: 8000-BFFF
WB_WINDOW2_BEGIN   equ 0x008000
WB_WINDOW2_END     equ 0x00BFFF
WB_WINDOW2_DEFAULT equ 2

; Окно 3: C000-FFFF
WB_WINDOW3_BEGIN   equ 0x00C000
WB_WINDOW3_END     equ 0x00FFFF
WB_WINDOW3_DEFAULT equ 3
```

```
; Размер банка K(64)
BANK_SIZE          equ 0x00010000  ; 64 КБ

; ============================================================================
; MMIO ПРОСТРАНСТВО (Memory Mapped I/O)
; ============================================================================

MMIO_BEGIN         equ 0xFF0000    ; Начало MMIO адрес( МБ16 - K64)
MMIO_END           equ 0xFFFFFF    ; Конец MMIO адрес( МБ16 - 1)
MMIO_SIZE          equ 0x00010000  ; 64 КБ

; MMIO_LO новые( устройства Aleste)
MMIO_LO_BEGIN      equ 0xFF0000    ; Начало MMIO_LO
MMIO_LO_END        equ 0xFF3FFF    ; Конец MMIO_LO
MMIO_LO_SIZE       equ 0x00004000  ; 16 КБ

; MMIO_HI эмуляция( Legacy устройств)
MMIO_HI_BEGIN      equ 0xFF4000    ; Начало MMIO_HI
MMIO_HI_END        equ 0xFFFFFF    ; Конец MMIO_HI
MMIO_HI_SIZE       equ 0x0000C000  ; 48 КБ

; ============================================================================
; УСТРОЙСТВА В MMIO_LO новые( устройства Aleste)
; ============================================================================

; Страница 0 MMIO_LO (FF0000-FF007F)
MMIO_PAGE0_BEGIN   equ 0xFF0000

; PIC Controller Прерывания()
MMIO_PIC_BASE      equ 0xFF0000
MMIO_PIC_SIZE      equ 32

; IPC Mailbox Системный( почтовый ящик)
MMIO_IPC_BASE      equ 0xFF0020
MMIO_IPC_SIZE      equ 32

; System Timer
MMIO_TIMER_BASE    equ 0xFF0040
MMIO_TIMER_SIZE    equ 32

; RTC Controller
MMIO_RTC_BASE      equ 0xFF0060
MMIO_RTC_SIZE      equ 32

; Расширенный доступ к MMU
MMIO_MMU_EXT_BEGIN equ 0xFF0080
MMIO_MMU_EXT_END   equ 0xFF00BF
MMIO_MMU_EXT_SIZE  equ 64          ; 16 регистров × 4 байта

; Legacy Emulation страница( 2: FF0100-FF017F)
MMIO_LEGACY_BEGIN  equ 0xFF0100
MMIO_LEGACY_END    equ 0xFF017F
MMIO_LEGACY_SIZE   equ 128
```

```
; Палитра цветов в( Legacy области)
MMIO_PALETTE_BASE  equ 0xFF0100    ; Первые 32 байта Legacy области
MMIO_PALETTE_SIZE  equ 32

; 6845 CRTC в( Legacy области)
MMIO_6845_BASE     equ 0xFF0120    ; Следующие 32 байта
MMIO_6845_SIZE     equ 32

; ================================================================================
; MMIO_HI эмуляция( Legacy устройств CPC)
; ================================================================================

; Gate Array (7FXXh порты)
MMIO_GA_BASE       equ 0xFF7F00
MMIO_GA_SIZE       equ 256         ; Полное отображение портов 00-FF

; CRTC 6845 (BCXXh/BDXXh порты)
MMIO_CRTC_INDEX    equ 0xFFBC00    ; BCXXh
MMIO_CRTC_DATA     equ 0xFFBD00    ; BDXXh

; Верхний ROM (DFXXh)
MMIO_UPPER_ROM     equ 0xFFDF00

; PPI 8255 (F4XXh-F7XXh)
MMIO_PPI_PORTA     equ 0xFFF400    ; F4XXh
MMIO_PPI_PORTB     equ 0xFFF500    ; F5XXh
MMIO_PPI_PORTC     equ 0xFFF600    ; F6XXh
MMIO_PPI_CTRL      equ 0xFFF700    ; F7XXh

; ================================================================================
; СИСТЕМНЫЕ РЕГИСТРЫ MMU порты( вводавывода-)
; ================================================================================

; Регистры Native Mode порты( F0-FF)
PORT_GLOBAL_CTRL   equ 0xF0        ; Главный регистр управления
PORT_MMIO_PAGE     equ 0xF1        ; Выбор страницы MMIO
PORT_SYSCALL       equ 0xF2        ; Системный вызов
PORT_CLOCK_CTRL    equ 0xF3        ; Управление частотой CPU
PORT_SUPER_SLOT    equ 0xF9        ; Активный слот для Supervisor
PORT_USER_SLOT     equ 0xFB        ; Активный слот для User
PORT_BANK_0        equ 0xFC        ; Банк для окна 0
PORT_BANK_1        equ 0xFD        ; Банк для окна 1
PORT_BANK_2        equ 0xFE        ; Банк для окна 2
PORT_BANK_3        equ 0xFF        ; Банк для окна 3

; Регистры Legacy Mode
PORT_LEGACY_RMR    equ 0x7F00      ; RMR регистр CPC
PORT_LEGACY_MMR    equ 0x7F00      ; MMR регистр CPC
PORT_LEGACY_UPPER_ROM equ 0xDF00   ; Выбор верхнего ROM
PORT_LEGACY_SYSCALL equ 0xF200     ; Системный вызов в Legacy

; ================================================================================
```

```
; БИТЫ РЕГИСТРА GLOBAL_CTRL
; ===========================================================================

; Позиции битов в регистре GLOBAL_CTRL порт( F0h)
GLOBAL_SUPERVISOR  equ 0           ; Бит 0: supervisor_mode
GLOBAL_NATIVE      equ 1           ; Бит 1: native_mode
GLOBAL_TRAP_ENABLE equ 2           ; Бит 2: supervisor_hook включение( trap)
GLOBAL_RESERVED1   equ 3           ; Бит 3: зарезервировано
GLOBAL_MMIO_UNLOCK equ 4           ; Бит 4: mmio_user_unlock
GLOBAL_RESERVED2   equ 5           ; Бит 5: зарезервировано
GLOBAL_RESERVED3   equ 6           ; Бит 6: зарезервировано
GLOBAL_RESERVED4   equ 7           ; Бит 7: зарезервировано

; Значения по умолчанию после сброса
GLOBAL_DEFAULT     equ (1 << GLOBAL_SUPERVISOR) | (1 << GLOBAL_NATIVE)
                                   ; supervisor=1, native=1, mmio_userlock=0

; ===========================================================================
; СЛОТЫ ПАМЯТИ для( SUPER_SLOT/USER_SLOT)
; ===========================================================================

; Доступные слоты (0-3)
SLOT_0             equ 0           ; Слот 0
SLOT_1             equ 1           ; Слот 1
SLOT_2             equ 2           ; Слот 2
SLOT_3             equ 3           ; Слот 3

; ===========================================================================
; МАКРОСЫ ДЛЯ РАБОТЫ С MMU
; ===========================================================================

; Получить текущий активный слот в( зависимости от режима)
; Выход: A = текущий слот (0-3)
MACRO GET_CURRENT_SLOT
    in a, (PORT_GLOBAL_CTRL)
    bit GLOBAL_SUPERVISOR, a
    jr z, @user_mode
    ; Supervisor mode
    in a, (PORT_SUPER_SLOT)
    jr @end
@user_mode:
    ; User mode
    in a, (PORT_USER_SLOT)
@end:
ENDM

; Переключить банк в текущем активном слоте
; Вход: window (0-3), bank (0-255)
MACRO SWITCH_BANK window, bank
    ld a, bank
    if window = 0
        out (PORT_BANK_0), a
    else if window = 1
```

```
        out (PORT_BANK_1), a
    else if window = 2
        out (PORT_BANK_2), a
    else if window = 3
        out (PORT_BANK_3), a
    endif
ENDM

; Быстрое переключение банка кода окно( 1)
MACRO SWITCH_CODE_BANK bank
    ld a, bank
    out (PORT_BANK_1), a
ENDM

; Быстрое переключение банка данных окно( 2)
MACRO SWITCH_DATA_BANK bank
    ld a, bank
    out (PORT_BANK_2), a
ENDM

; Системный вызов
; Вход: A = код функции
MACRO SYSCALL code
    ld a, code
    out (PORT_SYSCALL), a
ENDM

; Legacy системный вызов
MACRO SYSCALL_LEGACY code
    ld a, code
    out (PORT_LEGACY_SYSCALL), a
ENDM

; Вход в Supervisor Mode через trap
; Установить обработчики прерываний и включить trap
MACRO ENABLE_TRAP
    ld a, GLOBAL_DEFAULT | (1 << GLOBAL_TRAP_ENABLE)
    out (PORT_GLOBAL_CTRL), a
ENDM

; Выход из Supervisor Mode
MACRO EXIT_SUPERVISOR native_mode
    if native_mode = 1
        ld a, (1 << GLOBAL_NATIVE)  ; supervisor=0, native=1
    else
        xor a                       ; supervisor=0, native=0
    endif
    out (PORT_GLOBAL_CTRL), a
ENDM

; ============================================================================
; MMIO РАБОТА доступ( через окно)
; ============================================================================
```

```
; Выбрать страницу MMIO
; Вход: A = номер страницы (0-127)
MACRO SET_MMIO_PAGE page
    ld a, page
    out (PORT_MMIO_PAGE), a
ENDM

; Запись в регистр MMIO
; Вход: page = страница, offset = смещение (0-127), value = значение
MACRO WRITE_MMIO page, offset, value
    ld a, page
    out (PORT_MMIO_PAGE), a
    ld a, value
    out (offset), a
ENDM

; Чтение из регистра MMIO
; Вход: page = страница, offset = смещение (0-127)
; Выход: A = значение
MACRO READ_MMIO page, offset
    ld a, page
    out (PORT_MMIO_PAGE), a
    in a, (offset)
ENDM

; ============================================================================
; РАБОТА С РАСШИРЕННЫМ MMU (MMU_EXT область)
; ============================================================================

; Сохранить состояние маппера
; Вход: HL = указатель на буфер (64 байта)
MACRO SAVE_MMU_STATE buffer
    ld hl, buffer
    ld bc, 0040h           ; 64 байта
    ld a, 0                ; Страница 0
    out (PORT_MMIO_PAGE), a
    ld a, 80h              ; Начало MMU_EXT области
    ld d, 16               ; 16 регистров
@save_loop:
    ; Здесь должен быть код чтения через MMIO_WINDOW
    ; зависит( от конкретной реализации доступа)
    inc a
    dec d
    jr nz, @save_loop
ENDM

; ============================================================================
; КОНСТАНТЫ ДЛЯ СИСТЕМНЫХ ВЫЗОВОВ
; ============================================================================

; Коды системных вызовов
SYS_MMIO_ACCESS_REQUEST equ 0xFE  ; Запрос доступа к MMIO
```

```
; ============================================================================
; СИСТЕМНЫЕ ОБЛАСТИ ПАМЯТИ
; ============================================================================

; Таблица векторов прерываний в( начале памяти)
IVT_BEGIN          equ 0x000000
IVT_END            equ 0x0000FF
IVT_SIZE          equ 0x00000100  ; 256 байт

; Системный стек в( Supervisor слоте)
SYS_STACK_BEGIN    equ 0x00FF00    ; В конце окна 3
SYS_STACK_END      equ 0x00FFFF
SYS_STACK_SIZE     equ 0x00000100  ; 256 байт

; Область системных переменных
SYS_VARS_BEGIN     equ 0x000100
SYS_VARS_END       equ 0x0003FF
SYS_VARS_SIZE      equ 0x00000300  ; 768 байт

; ============================================================================
; РАЗМЕЩЕНИЕ ПРОГРАММ
; ============================================================================

; Базовый адрес загрузки программ окно( 1)
PROGRAM_LOAD_BASE  equ 0x004000
PROGRAM_LOAD_END   equ 0x007FFF
PROGRAM_LOAD_SIZE  equ 0x00004000  ; K16

; Область данных программы окно( 2)
PROGRAM_DATA_BASE  equ 0x008000
PROGRAM_DATA_END   equ 0x00BFFF
PROGRAM_DATA_SIZE  equ 0x00004000  ; K16

; Стек программы окно( 3)
PROGRAM_STACK_BASE equ 0x00C000
PROGRAM_STACK_TOP  equ 0x00FFFF    ; Растёт вниз
PROGRAM_STACK_SIZE equ 0x00004000  ; K16

; ============================================================================
; РЕЖИМЫ ДОСТУПА
; ============================================================================

; Режимы пользователя сочетания( supervisor/native)
MODE_SUPERVISOR_NATIVE equ (1 << GLOBAL_SUPERVISOR) | (1 << GLOBAL_NATIVE)
MODE_USER_NATIVE      equ (1 << GLOBAL_NATIVE)
MODE_USER_LEGACY      equ 0

; Проверка режима доступа
; Вход: mode = режим для проверки
; Выход: Z=1 если текущий режим совпадает
MACRO CHECK_MODE mode
    in a, (PORT_GLOBAL_CTRL)
```

```
    and (1 << GLOBAL_SUPERVISOR) | (1 << GLOBAL_NATIVE)
    cp mode
ENDM

; ============================================================================
; ФЛАГИ БЕЗОПАСНОСТИ
; ============================================================================

; Проверка блокировки MMIO в User Mode
; Выход: Z=0 если доступ заблокирован и( мы в User Mode)
MACRO CHECK_MMIO_LOCK
    in a, (PORT_GLOBAL_CTRL)
    bit GLOBAL_SUPERVISOR, a
    jr nz, @supervisor    ; В Supervisor - всегда разрешено
    bit GLOBAL_MMIO_UNLOCK, a
    jr z, @locked         ; В User и mmio_userlock=0
@supervisor:
    xor a                 ; Z=1 разрешено()
    ret
@locked:
    or 1                  ; Z=0 запрещено()
ENDM
```

## 4.2. memory_aleste.asm

```
; ============================================================================
; MEMORY_ALESTE@INC - Карта памяти Aleste LX с режимами Supervisor/User
; Соответствует спецификации MMU Architecture v2@0
; ============================================================================

; ============================================================================
; БАЗОВЫЕ КОНСТАНТЫ ПАМЯТИ
; ============================================================================

; Адресное пространство (24 бита)
MEMORY_SIZE        equ 0x01000000  ; 16 МБ общее адресное пространство
PHYSICAL_RAM_SIZE  equ 0x00100000  ; 1 МБ физической оперативной памяти

; Базовые адреса
RAM_BASE           equ 0x00000000  ; Начало оперативной памяти
MMIO_BASE          equ 0x00FF0000  ; Начало MMIO пространства последние( К64)

; ============================================================================
; ОКОННАЯ БАНКОВАЯ ПАМЯТЬ (WINDOWED BANK MEMORY)
; ============================================================================

; Общее оконное пространство до( MMIO)
WB_MEMORY_BEGIN    equ 0x000000    ; Начало оконной памяти
WB_MEMORY_END      equ 0xFEFFFF    ; Конец оконной памяти перед( MMIO)
WB_MEMORY_SIZE     equ 0xFF0000    ; Размер: МБ16 - К64

; Логические окна по( К16 каждое)
```

```
WB_WINDOW_SIZE     equ 0x00004000  ; 16 КБ на окно

; Окно 0: 0000-3FFF
WB_WINDOW0_BEGIN   equ 0x000000
WB_WINDOW0_END     equ 0x003FFF
WB_WINDOW0_DEFAULT equ 0           ; Банк по умолчанию

; Окно 1: 4000-7FFF
WB_WINDOW1_BEGIN   equ 0x004000
WB_WINDOW1_END     equ 0x007FFF
WB_WINDOW1_DEFAULT equ 1

; Окно 2: 8000-BFFF
WB_WINDOW2_BEGIN   equ 0x008000
WB_WINDOW2_END     equ 0x00BFFF
WB_WINDOW2_DEFAULT equ 2

; Окно 3: C000-FFFF
WB_WINDOW3_BEGIN   equ 0x00C000
WB_WINDOW3_END     equ 0x00FFFF
WB_WINDOW3_DEFAULT equ 3

; Размер банка K(64)
BANK_SIZE          equ 0x00010000  ; 64 КБ

; ============================================================================
; MMIO ПРОСТРАНСТВО (Memory Mapped I/O)
; ============================================================================

MMIO_BEGIN         equ 0xFF0000    ; Начало MMIO адрес( МБ16 - К64)
MMIO_END           equ 0xFFFFFF    ; Конец MMIO адрес( МБ16 - 1)
MMIO_SIZE          equ 0x00010000  ; 64 КБ

; MMIO_LO новые( устройства Aleste)
MMIO_LO_BEGIN      equ 0xFF0000    ; Начало MMIO_LO
MMIO_LO_END        equ 0xFF3FFF    ; Конец MMIO_LO
MMIO_LO_SIZE       equ 0x00004000  ; 16 КБ

; MMIO_HI эмуляция( Legacy устройств)
MMIO_HI_BEGIN      equ 0xFF4000    ; Начало MMIO_HI
MMIO_HI_END        equ 0xFFFFFF    ; Конец MMIO_HI
MMIO_HI_SIZE       equ 0x0000C000  ; 48 КБ

; ============================================================================
; УСТРОЙСТВА В MMIO_LO новые( устройства Aleste)
; ============================================================================

; Страница 0 MMIO_LO (FF0000-FF007F)
MMIO_PAGE0_BEGIN   equ 0xFF0000

; PIC Controller Прерывания()
MMIO_PIC_BASE      equ 0xFF0000
MMIO_PIC_SIZE      equ 32
```

```
; IPC Mailbox Системный( почтовый ящик)
MMIO_IPC_BASE      equ 0xFF0020
MMIO_IPC_SIZE      equ 32

; System Timer
MMIO_TIMER_BASE    equ 0xFF0040
MMIO_TIMER_SIZE    equ 32

; RTC Controller
MMIO_RTC_BASE      equ 0xFF0060
MMIO_RTC_SIZE      equ 32

; Расширенный доступ к MMU
MMIO_MMU_EXT_BEGIN equ 0xFF0080
MMIO_MMU_EXT_END   equ 0xFF00BF
MMIO_MMU_EXT_SIZE  equ 64          ; 16 регистров × 4 байта

; Legacy Emulation страница( 2: FF0100-FF017F)
MMIO_LEGACY_BEGIN  equ 0xFF0100
MMIO_LEGACY_END    equ 0xFF017F
MMIO_LEGACY_SIZE   equ 128

; Палитра цветов в( Legacy области)
MMIO_PALETTE_BASE  equ 0xFF0100    ; Первые 32 байта Legacy области
MMIO_PALETTE_SIZE  equ 32

; 6845 CRTC в( Legacy области)
MMIO_6845_BASE     equ 0xFF0120    ; Следующие 32 байта
MMIO_6845_SIZE     equ 32

; ================================================================================
; MMIO_HI эмуляция( Legacy устройств CPC)
; ================================================================================

; Gate Array (7FXXh порты)
MMIO_GA_BASE       equ 0xFF7F00
MMIO_GA_SIZE       equ 256         ; Полное отображение портов 00-FF

; CRTC 6845 (BCXXh/BDXXh порты)
MMIO_CRTC_INDEX    equ 0xFFBC00    ; BCXXh
MMIO_CRTC_DATA     equ 0xFFBD00    ; BDXXh

; Верхний ROM (DFXXh)
MMIO_UPPER_ROM     equ 0xFFDF00

; PPI 8255 (F4XXh-F7XXh)
MMIO_PPI_PORTA     equ 0xFFF400    ; F4XXh
MMIO_PPI_PORTB     equ 0xFFF500    ; F5XXh
MMIO_PPI_PORTC     equ 0xFFF600    ; F6XXh
MMIO_PPI_CTRL      equ 0xFFF700    ; F7XXh

; ================================================================================
```

```
; СИСТЕМНЫЕ РЕГИСТРЫ MMU порты( вводавывода-)
; ============================================================================

; Регистры Native Mode порты( F0-FF)
PORT_GLOBAL_CTRL   equ 0xF0        ; Главный регистр управления
PORT_MMIO_PAGE     equ 0xF1        ; Выбор страницы MMIO
PORT_SYSCALL       equ 0xF2        ; Системный вызов
PORT_CLOCK_CTRL    equ 0xF3        ; Управление частотой CPU
PORT_SUPER_SLOT    equ 0xF9        ; Активный слот для Supervisor
PORT_USER_SLOT     equ 0xFB        ; Активный слот для User
PORT_BANK_0        equ 0xFC        ; Банк для окна 0
PORT_BANK_1        equ 0xFD        ; Банк для окна 1
PORT_BANK_2        equ 0xFE        ; Банк для окна 2
PORT_BANK_3        equ 0xFF        ; Банк для окна 3

; Регистры Legacy Mode
PORT_LEGACY_RMR    equ 0x7F00      ; RMR регистр CPC
PORT_LEGACY_MMR    equ 0x7F00      ; MMR регистр CPC
PORT_LEGACY_UPPER_ROM equ 0xDF00   ; Выбор верхнего ROM
PORT_LEGACY_SYSCALL equ 0xF200     ; Системный вызов в Legacy

; ============================================================================
; БИТЫ РЕГИСТРА GLOBAL_CTRL
; ============================================================================

; Позиции битов в регистре GLOBAL_CTRL порт( F0h)
GLOBAL_SUPERVISOR  equ 0           ; Бит 0: supervisor_mode
GLOBAL_NATIVE      equ 1           ; Бит 1: native_mode
GLOBAL_TRAP_ENABLE equ 2           ; Бит 2: supervisor_hook включение( trap)
GLOBAL_RESERVED1   equ 3           ; Бит 3: зарезервировано
GLOBAL_MMIO_UNLOCK equ 4           ; Бит 4: mmio_user_unlock
GLOBAL_RESERVED2   equ 5           ; Бит 5: зарезервировано
GLOBAL_RESERVED3   equ 6           ; Бит 6: зарезервировано
GLOBAL_RESERVED4   equ 7           ; Бит 7: зарезервировано

; Значения по умолчанию после сброса
GLOBAL_DEFAULT     equ (1 << GLOBAL_SUPERVISOR) | (1 << GLOBAL_NATIVE)
                                   ; supervisor=1, native=1, mmio_userlock=0

; ============================================================================
; СЛОТЫ ПАМЯТИ для( SUPER_SLOT/USER_SLOT)
; ============================================================================

; Доступные слоты (0-3)
SLOT_0             equ 0           ; Слот 0
SLOT_1             equ 1           ; Слот 1
SLOT_2             equ 2           ; Слот 2
SLOT_3             equ 3           ; Слот 3

; ============================================================================
; МАКРОСЫ ДЛЯ РАБОТЫ С MMU
; ============================================================================
```

```
; Получить текущий активный слот в( зависимости от режима)
; Выход: A = текущий слот (0-3)
MACRO GET_CURRENT_SLOT
    in a, (PORT_GLOBAL_CTRL)
    bit GLOBAL_SUPERVISOR, a
    jr z, @user_mode
    ; Supervisor mode
    in a, (PORT_SUPER_SLOT)
    jr @end
@user_mode:
    ; User mode
    in a, (PORT_USER_SLOT)
@end:
ENDM

; Переключить банк в текущем активном слоте
; Вход: window (0-3), bank (0-255)
MACRO SWITCH_BANK window, bank
    ld a, bank
    if window = 0
        out (PORT_BANK_0), a
    else if window = 1
        out (PORT_BANK_1), a
    else if window = 2
        out (PORT_BANK_2), a
    else if window = 3
        out (PORT_BANK_3), a
    endif
ENDM

; Быстрое переключение банка кода окно( 1)
MACRO SWITCH_CODE_BANK bank
    ld a, bank
    out (PORT_BANK_1), a
ENDM

; Быстрое переключение банка данных окно( 2)
MACRO SWITCH_DATA_BANK bank
    ld a, bank
    out (PORT_BANK_2), a
ENDM

; Системный вызов
; Вход: A = код функции
MACRO SYSCALL code
    ld a, code
    out (PORT_SYSCALL), a
ENDM

; Legacy системный вызов
MACRO SYSCALL_LEGACY code
    ld a, code
    out (PORT_LEGACY_SYSCALL), a
```

```
ENDM

; Вход в Supervisor Mode через trap
; Установить обработчики прерываний и включить trap
MACRO ENABLE_TRAP
    ld a, GLOBAL_DEFAULT | (1 << GLOBAL_TRAP_ENABLE)
    out (PORT_GLOBAL_CTRL), a
ENDM

; Выход из Supervisor Mode
MACRO EXIT_SUPERVISOR native_mode
    if native_mode = 1
        ld a, (1 << GLOBAL_NATIVE)  ; supervisor=0, native=1
    else
        xor a                       ; supervisor=0, native=0
    endif
    out (PORT_GLOBAL_CTRL), a
ENDM

; ============================================================================
; MMIO РАБОТА доступ( через окно)
; ============================================================================

; Выбрать страницу MMIO
; Вход: A = номер страницы (0-127)
MACRO SET_MMIO_PAGE page
    ld a, page
    out (PORT_MMIO_PAGE), a
ENDM

; Запись в регистр MMIO
; Вход: page = страница, offset = смещение (0-127), value = значение
MACRO WRITE_MMIO page, offset, value
    ld a, page
    out (PORT_MMIO_PAGE), a
    ld a, value
    out (offset), a
ENDM

; Чтение из регистра MMIO
; Вход: page = страница, offset = смещение (0-127)
; Выход: A = значение
MACRO READ_MMIO page, offset
    ld a, page
    out (PORT_MMIO_PAGE), a
    in a, (offset)
ENDM

; ============================================================================
; РАБОТА С РАСШИРЕННЫМ MMU (MMU_EXT область)
; ============================================================================

; Сохранить состояние маппера
```

```
; Вход: HL = указатель на буфер (64 байта)
MACRO SAVE_MMU_STATE buffer
    ld hl, buffer
    ld bc, 0040h           ; 64 байта
    ld a, 0                ; Страница 0
    out (PORT_MMIO_PAGE), a
    ld a, 80h              ; Начало MMU_EXT области
    ld d, 16               ; 16 регистров
@save_loop:
    ; Здесь должен быть код чтения через MMIO_WINDOW
    ; зависит( от конкретной реализации доступа)
    inc a
    dec d
    jr nz, @save_loop
ENDM

; ============================================================================
; КОНСТАНТЫ ДЛЯ СИСТЕМНЫХ ВЫЗОВОВ
; ============================================================================

; Коды системных вызовов
SYS_MMIO_ACCESS_REQUEST equ 0xFE  ; Запрос доступа к MMIO

; ============================================================================
; СИСТЕМНЫЕ ОБЛАСТИ ПАМЯТИ
; ============================================================================

; Таблица векторов прерываний в( начале памяти)
IVT_BEGIN          equ 0x000000
IVT_END            equ 0x0000FF
IVT_SIZE          equ 0x00000100  ; 256 байт

; Системный стек в( Supervisor слоте)
SYS_STACK_BEGIN    equ 0x00FF00    ; В конце окна 3
SYS_STACK_END      equ 0x00FFFF
SYS_STACK_SIZE     equ 0x00000100  ; 256 байт

; Область системных переменных
SYS_VARS_BEGIN     equ 0x000100
SYS_VARS_END       equ 0x0003FF
SYS_VARS_SIZE      equ 0x00000300  ; 768 байт

; ============================================================================
; РАЗМЕЩЕНИЕ ПРОГРАММ
; ============================================================================

; Базовый адрес загрузки программ окно( 1)
PROGRAM_LOAD_BASE  equ 0x004000
PROGRAM_LOAD_END   equ 0x007FFF
PROGRAM_LOAD_SIZE  equ 0x00004000  ; K16

; Область данных программы окно( 2)
PROGRAM_DATA_BASE  equ 0x008000
```

```
PROGRAM_DATA_END   equ 0x00BFFF
PROGRAM_DATA_SIZE  equ 0x00004000  ; K16

; Стек программы окно( 3)
PROGRAM_STACK_BASE equ 0x00C000
PROGRAM_STACK_TOP  equ 0x00FFFF    ; Растёт вниз
PROGRAM_STACK_SIZE equ 0x00004000  ; K16

; ============================================================================
; РЕЖИМЫ ДОСТУПА
; ============================================================================

; Режимы пользователя сочетания( supervisor/native)
MODE_SUPERVISOR_NATIVE equ (1 << GLOBAL_SUPERVISOR) | (1 << GLOBAL_NATIVE)
MODE_USER_NATIVE      equ (1 << GLOBAL_NATIVE)
MODE_USER_LEGACY      equ 0

; Проверка режима доступа
; Вход: mode = режим для проверки
; Выход: Z=1 если текущий режим совпадает
MACRO CHECK_MODE mode
    in a, (PORT_GLOBAL_CTRL)
    and (1 << GLOBAL_SUPERVISOR) | (1 << GLOBAL_NATIVE)
    cp mode
ENDM

; ============================================================================
; ФЛАГИ БЕЗОПАСНОСТИ
; ============================================================================

; Проверка блокировки MMIO в User Mode
; Выход: Z=0 если доступ заблокирован и( мы в User Mode)
MACRO CHECK_MMIO_LOCK
    in a, (PORT_GLOBAL_CTRL)
    bit GLOBAL_SUPERVISOR, a
    jr nz, @supervisor    ; В Supervisor - всегда разрешено
    bit GLOBAL_MMIO_UNLOCK, a
    jr z, @locked         ; В User и mmio_userlock=0
@supervisor:
    xor a                 ; Z=1 разрешено()
    ret
@locked:
    or 1                  ; Z=0 запрещено()
ENDM
```