# Конспект: Микроконтроллеры AVR

## 1 Структура AVR

Микроконтроллеры AVR имеют гарвардскую архитектуру с раздельными шинами для команд и данных. Это обеспечивает параллельную загрузку инструкций и данных, увеличивая производительность.

Основные компоненты:

- 8-битное RISC-ядро с набором оптимизированных инструкций
- Flash-память программ (перепрограммируемая)
- Оперативная память (SRAM)
- Постоянная память (EEPROM)
- Таймеры/счётчики
- Аналого-цифровой преобразователь (АЦП)
- Последовательный интерфейс (USART, SPI, I2C)
- Прерывания и система управления питанием

## 2 Память

### 2.1 Организация памяти

Память микроконтроллера делится на три основные области:

- **Flash** — энергонезависимая память программ (до 256 КБ), хранит исполняемый код. Доступ осуществляется 16-битными словами.
- **SRAM** — оперативная память для переменных и стека. Обычно до 16 КБ. Ячейки 8-битные.
- **EEPROM** — энергонезависимая память для хранения пользовательских данных. Поддерживает около $10^5$ циклов записи.

## 2.2 Регистры

- 32 **регистра общего назначения** (R0–R31), используемые для большинства операций.
- **Регистр статуса** (SREG) содержит 8 флагов, отображающих состояние ALU (арифметико-логического устройства).
- **Указатель стека (SP)** — хранит адрес вершины стека, 16-битный.
- **I/O-регистры** — специальные регистры для управления периферией.

## 2.3 Флаги регистра SREG

- **I** — глобальное разрешение прерываний
- **T** — временный бит (Transfer Bit)
- **H** — половинный перенос (Half Carry)
- **S** — знак (Sign = N $\oplus$ V)
- **V** — переполнение (Overflow)
- **N** — отрицательный результат
- **Z** — результат равен нулю
- **C** — перенос (Carry)

# 3 Система команд

AVR использует компактный и эффективный набор инструкций. Большинство инструкций исполняется за 1 такт.

Типы инструкций:

- Арифметика и логика: `ADD, SUB, AND, OR, INC, DEC`
- Операции с битами: `SET, CLR, SBI, CBI`
- Переходы: `RJMP, IJMP, CALL, RET`
- Работа с памятью: `MOV, LDS, STS, LDI, PUSH, POP`
- Управление системой: `NOP, SLEEP, WDR, SEI, CLI`

# 4 Способы адресации

1. **Прямая** — указание регистров: `ADD R1, R2`
2. **Непосредственная** — значение закодировано в инструкции: `LDI R16, 0xFF`
3. **Косвенная** — через регистры X, Y, Z: `LD R0, X+`
4. **Относительная** — используется в переходах: `RJMP LABEL`
5. **Прямая адресация SRAM** — через абсолютный адрес: `LDS R16, 0x100`
6. **Адресация I/O** — доступ к портам и регистрам: `IN R16, PORTB`

# 5 Прерывания

## 5.1 Типы прерываний

- Внешние: `INT0`, `INT1`
- От таймеров/счётчиков (переполнение, сравнение)
- От USART (приём/передача данных)
- От АЦП (завершение преобразования)
- От SPI/I2C
- Аппаратный сброс и Watchdog

## 5.2 Обработка прерываний

1. Завершение текущей инструкции
2. Сохранение адреса возврата (PC) в стек
3. Переход к обработчику прерывания (ISR)
4. Выполнение обработчика
5. Команда `RETI` восстанавливает PC

Вектора прерываний расположены по фиксированным адресам в начале Flash-памяти.

# 6 Таймеры/Счётчики

AVR содержит до трёх таймеров:

- 8-битные и 16-битные таймеры (Timer0, Timer1, Timer2)
- Поддержка режима счёта, сравнения, PWM
- Возможность генерации прерываний по совпадению, переполнению
- Используются для временных задержек, ШИМ, и периодических событий

# 7 Аналого-цифровой преобразователь (АЦП)

- 10-битный АЦП (в большинстве AVR)
- Поддержка до 8 аналоговых входов (в зависимости от модели)
- Настраиваемое опорное напряжение (AVCC, AREF, внутреннее)
- Возможность автоматического запуска по триггеру
- Генерация прерывания по завершении преобразования

# 8 Архитектура шин

AVR использует модифицированную гарвардскую архитектуру, где:

- Команды и данные передаются по разным шинам — это позволяет загружать новую инструкцию, пока текущая выполняется.
- Доступ к Flash-памяти осуществляется отдельной шиной команд.
- SRAM и I/O устройства доступны через шину данных.

Это повышает производительность и позволяет большинству команд выполняться за 1 такт.

# 9 Стек

- Стек в AVR размещён в SRAM и управляется 16-битным указателем стека (SP).
- Используется для сохранения адреса возврата при вызовах подпрограмм и обработке прерываний.
- Команды `PUSH` и `POP` — явное управление стеком.

# 10 Ввод/вывод (I/O)

Работа с портами осуществляется через три регистра:

- `DDRx` — направление (1 — выход, 0 — вход)
- `PORTx` — установка значения (для выхода) или подтягивание (для входа)
- `PINx` — чтение значения на входе

**Пример:** установка PORTB0 как выхода и подача лог. 1

```
sbi DDRB, 0   ; PORTB0 как выход
sbi PORTB, 0  ; установить лог. 1 на PORTB0
```

# 11 Тактирование

- Варианты тактовых источников: внутренний RC-генератор, внешний кварц, внешний источник.
- Частота может достигать до 20 МГц (в зависимости от модели).
- Делители частоты (prescalers) используются в таймерах и ADC.

# 12 Управление питанием

AVR поддерживает несколько режимов энергосбережения:

- `Idle` — остановка ЦП, но работают прерывания и периферия
- `Power-down` — почти всё отключено, минимальное потребление
- `ADC Noise Reduction` — минимизация помех при работе АЦП

Переход осуществляется через команду `SLEEP`, а выход — по прерыванию.

# 13 Арифметико-логическое устройство (ALU)

ALU выполняет основные арифметические и логические операции:

- Сложение, вычитание (с флагами переноса и заёма)
- Побитовые операции (AND, OR, XOR, NOT)
- Сдвиги (LSL, LSR, ASR, ROR)
- Управление флагами регистра SREG

ALU работает с регистрами общего назначения (R0–R31), в большинстве операций участвуют только они.

# 14 Пример простой программы на ассемблере

**Программа мигания светодиодом на порту B0:**

```
    ldi r16, 0x01        ; установить бит 0
    out DDRB, r16        ; порт B0 как выход
loop:
    out PORTB, r16       ; включить светодиод
    rcall delay
    out PORTB, r1        ; выключить
    rcall delay
    rjmp loop

delay:                  ; простая задержка
    ldi r18, 100
wait1:
    ldi r19, 255
wait2:
    dec r19
    brne wait2
    dec r18
    brne wait1
    ret
```

# 15 Типы микроконтроллеров AVR

Линейка AVR включает несколько семейств:

- **tinyAVR** — простые, маломощные (например, ATtiny85)
- **megaAVR** — более функциональные, с большим числом портов (например, ATmega328P)
- **XMEGA** — более мощные, с улучшенным управлением питанием, DMA, повышенной частотой

# 16 Системы сброса (reset)

AVR поддерживает несколько источников сброса:

- **Аппаратный (external)** — по входу `RESET`
- **Watchdog reset** — при переполнении таймера сторожевого таймера
- **Power-on reset** — при включении питания
- **Brown-out reset** — при просадке питания ниже допустимого уровня

Состояние сброса можно отследить через регистр `MCUSR`.

# 17 Программаторы и прошивка

AVR поддерживает несколько способов прошивки:

- **ISP (In-System Programming)** — наиболее распространённый способ (через SPI)
- **PDI, TPI** — у новейших моделей
- **JTAG** — поддерживается для отладки и прошивки старших моделей

Типовые программаторы:

- USBasp (самодельный/китайский)
- AVRISP mkII
- Arduino как программатор (через ArduinoISP)

# 18 Сравнение с другими архитектурами

AVR по сравнению с другими МК:

- Простая архитектура, легко изучать
- Интуитивно понятные регистры и I/O
- По сравнению с ARM — медленнее, меньше памяти, но проще
- По сравнению с PIC — чище архитектура и более дружелюбная к C/ASM разработке

AVR идеально подходит для учебных целей, прототипирования и простых проектов.

# 19 Циклы машинных тактов

- Большинство инструкций AVR выполняются за 1 такт.
- Некоторые, например `CALL`, `RET`, занимают 3 и более тактов.
- Тактовая частота контролируется системным кварцем или внутренним RC-генератором.
- От частоты зависит скорость выполнения всех операций и таймеров.

# 20 Фьюзы (Fuse-биты)

Фьюзы — специальные биты настройки, определяющие работу МК:

- Выбор источника тактирования (внутренний RC, внешний кварц и т.п.)
- Включение/отключение сброса
- Активация режима загрузчика (bootloader)
- Защита от чтения/записи Flash

Фьюзы не хранятся в обычной памяти, и их можно изменить только через программатор.

# 21 Bootloader

Некоторые AVR поддерживают область загрузчика:

- Разделённая часть Flash-памяти.
- Позволяет перепрошивать микроконтроллер без внешнего программатора.
- Часто используется в Arduino-подобных платах.

# 22 Работа с прерываниями в C

**Пример на языке C с использованием** `avr-gcc`**:**

```
#include <avr/io.h>
#include <avr/interrupt.h>

ISR(TIMER0_OVF_vect) {
    PORTB ^= (1 << PB0); // Инвертировать PORTB0 при переполнении таймера
}

int main(void) {
    DDRB |= (1 << PB0);         // PORTB0 как выход
    TCCR0 |= (1 << CS02);       // Предделитель 256
    TIMSK |= (1 << TOIE0);      // Разрешить прерывание переполнения
    sei();                      // Глобальное разрешение прерываний

    while (1) {
        // Главный цикл пустой, всё делает прерывание
    }
}
```

# 23 Советы по разработке

- Используйте симулятор (AVR Studio/Proteus) для отладки логики.
- Минимизируйте количество операций внутри ISR.
- Проверяйте настройку фьюзов при прошивке.
- Старайтесь избегать конфликтов между ISR и main при доступе к общим данным — используйте `volatile`.