Лабораторная работа 1

ОСНОВЫ РАБОТЫ В СРЕДЕ ATMEL STUDIO. АРХИТЕКТУРА И СИСТЕМА КОМАНД МИКРОКОНТРОЛЛЕРА ATMEGA32

Цель работы

Изучение основ программирования микроконтроллеров семейства Atmel ATmega в среде разработки Atmel Studio. Знакомство с системой команд микроконтроллера ATmega32.

Теоретические сведения

Возможности отладочной платы EasyAVR v7

Отладочные платы EasyAVR предоставляют возможность проектировать программно-аппаратные средства, основывающиеся на микроконтроллерах Atmel AVR. Семейство отладочных плат EasyAVR совершенствовалось в течение нескольких поколений и на сегодняшний день имеет множество пользователей среди студентов, любителей и профессионалов в области разработки микропроцессорных систем.

В состав отладочной платы EasyAVR v7 включены следующие компоненты и интерфейсы:

– программатор mikroProg;

– кварцевый резонатор;

– четыре порта ввода-вывода, при этом каждый вывод снабжён кнопкой, светодиодом и выводом к нескольким внешним разъёмам;

– интерфейс UART, выведенный к портам RS-232 и USB;

– микросхема EEPROM, подключённая к микроконтроллеру через интерфейс I_2C;

– четыре семисегментных индикатора;

– имитатор аналогово входного сигнала;

– динамик Piezo Buzzer;

– интерфейс для подключения 2x16 символьного ЖК-дисплея;

– интерфейс для подключения графического дисплея 128x64;

– интерфейс для подключения сенсорной панели;

– интерфейсы для подключения температурных датчиков;

– интерфейсы для подключения плат-расширений (GPS, GSM, карта памяти microSD, инфракрасный порт, интерфейс Wi-Fi и т.д.).

Отладочная плата EasyAVR v7 снабжена программатором mikroProg, позволяющим «перепрашивать» 65 различных AVR микроконтроллеров фирмы Atmel. Программатор подключается к персональному компьютеру через порт USB 2.0. Текущее состояние программатора показывается тремя светодиодными индикаторами. Светодиод «Link» загорается, когда установлена USB-связь с персонального компьютера. Светодиод «Active» светится, когда выполняется программа. Светодиод «Data» загорается при передаче данных между программатором и программным обеспечением AVRFlash на персональном компьютере.

Одним из главных достоинств EasyAVR v7 являются четыре группы 8-ми разрядных портов ввода/вывода. Все выводы портов оснащены кнопками и светодиодными индикаторами. Для каждого порта на плате размещены три 8-ми разрядных разъёма для подключения внешних устройств.

Для возможности взаимодействия с различными современными компьютерами плата оснащена разъёмом RS-232 и дополнительным разъёмом USB, подключёнными к интерфейсу UART. Микросхема FT232RL позволяет конвертировать UART сигналы в соответствии со стандартами USB. Для использования связи USB-UART требуется установка драйвера FTDI на компьютер.

Плата оснащена интерфейсом I_2C, который используется для подключения периферийных низкоскоростных устройств. Всего к одной шине могут быть подключены до 112 устройств. Каждое устройство должно иметь уникальный адрес. В состав платы входит микросхема EEPROM, подключённая к интерфейсу I_2C, имеющая 1024 байта доступной памяти при скорости передачи данных 400 кГц.

Плата оснащена четырьмя семисегментными индикаторами, которые могут быть использованы для отображения цифр от 0 до 9, разделителя в виде точки и некоторых букв.

Плата EasyAVR v7 содержит интерфейс в виде двух потенциометров для имитации аналоговых входных напряжений, которые могут быть использованы на любом из двенадцати поддерживающих аналоговые входные сигналы контактов.

Микроконтроллеры могут создавать звук при помощи широтно-импульсной модуляцией (PWM), подавая формируемый сигнал на

встроенный Piezo Buzzer. Резонансная частота Piezo Buzzer равна 3.8 кГц. Возможно создание звука в диапазоне от 2 до 4 кГц.

Микроконтроллер Atmel AVR ATmega32

По своим функциональным возможностям современные микроконтроллеры представляют собой полномасштабный компьютер (процессор, память программ и память данных, порты ввода-вывода, таймеры и периферийные устройства), выполненный на одном кристалле. Все необходимые для функционирования компоненты находятся внутри микросхемы, поэтому для работы микроконтроллера требуется только подача внешнего питания. В зависимости от рабочей частоты, модели и используемых периферийных устройств микроконтроллер потребляет от 5 до 200 мА.

Микроконтроллер реализовывает гарвардскую архитектуру, т.е. имеет энергонезависимую память программ, и память данных, которая сбрасывается при отключении питания. Энергонезависимая память может быть OTP (one-time programmable, записывается при производстве микроконтроллера), но в большинстве случаев используется EEPROM, что позволяет изменять исполняемую программу. Программа записывается в микроконтроллер с помощью специального устройства – программатора. Записываемый машинный код называется firmware, или «прошивкой».

Помимо процессора и памяти в микроконтроллере могут присутствовать следующие периферийные устройства: модуль USART, аналого-цифровой преобразователь, цифро-аналоговый преобразователь, таймер, аналоговый компаратор, интерфейсы SPI, CAN и другие. Микроконтроллеры отличаются по своим характеристикам: форм-фактору, количеству выводов, объёму памяти программ и данных, видам и количеству периферийных устройств, тактовой частоте и другим параметрам.

Подробную документацию по схемотехнике и организации каждого конкретного микроконтроллера можно получить из спецификации (datasheet) производителя.

Микроконтроллер ATmega32 является маломощным 8-разрядным КМОП микроконтроллером на основе усовершенствованной архитектуры RISC, выполняющий большинство инструкций за один такт.

Ядро ATmega32 сочетает в себе богатый набор команд и 32 регистра общего назначения. Все 32 регистра подключены непосредственно к

арифметико-логическому устройству (АЛУ), что позволяет инструкциям за один такт обращаться к двум независимым регистрам. Такая архитектура позволяет достигать пропускной способности до десяти раз быстрее по сравнению с обычными CISC микроконтроллерами. Блок-схема микроконтроллера приведена на рисунке 1.

ATmega32 предоставляет следующие возможности:

– 131 инструкция, большинство выполняемых за один такт

– пропускная способность до 16 MIPS при частоте до 16 МГц

– 32 КБ In-System Pragrammable (ISP) памяти программ с возможностью Read-While-Write;

– 1024 байт EEPROM;

– 2 КБ внутренней SRAM;

– 32 программируемых линии портов ввода/вывода общего назначения;

– 32 восьми разрядных регистра общего назначения;

– внешние и внутренние источники прерываний;

– интерфейс JTAG;

– два 8-разрядных таймеров-счётчиков с отдельными делителями частоты и режимами сравнения;

– один 16-разрядный таймер-счётчик с отдельным предварительным делителем, режимами сравнения и захвата;

– счётчик реального времени с отдельным генератором;

– программируемый Watchdog-таймер с встроенный генератором;

– четыре канала широтно-импульсной модуляции (ШИМ);

– программируемый последовательный интерфейс USART;

– последовательный интерфейс SPI (Master/Slave);

– встроенный аналоговый компаратор;

– 8-канальный 10-битовый АЦП с программируемым коэффициентом усиления;

– байт-ориентированный двухпроводной последовательный интерфейс;

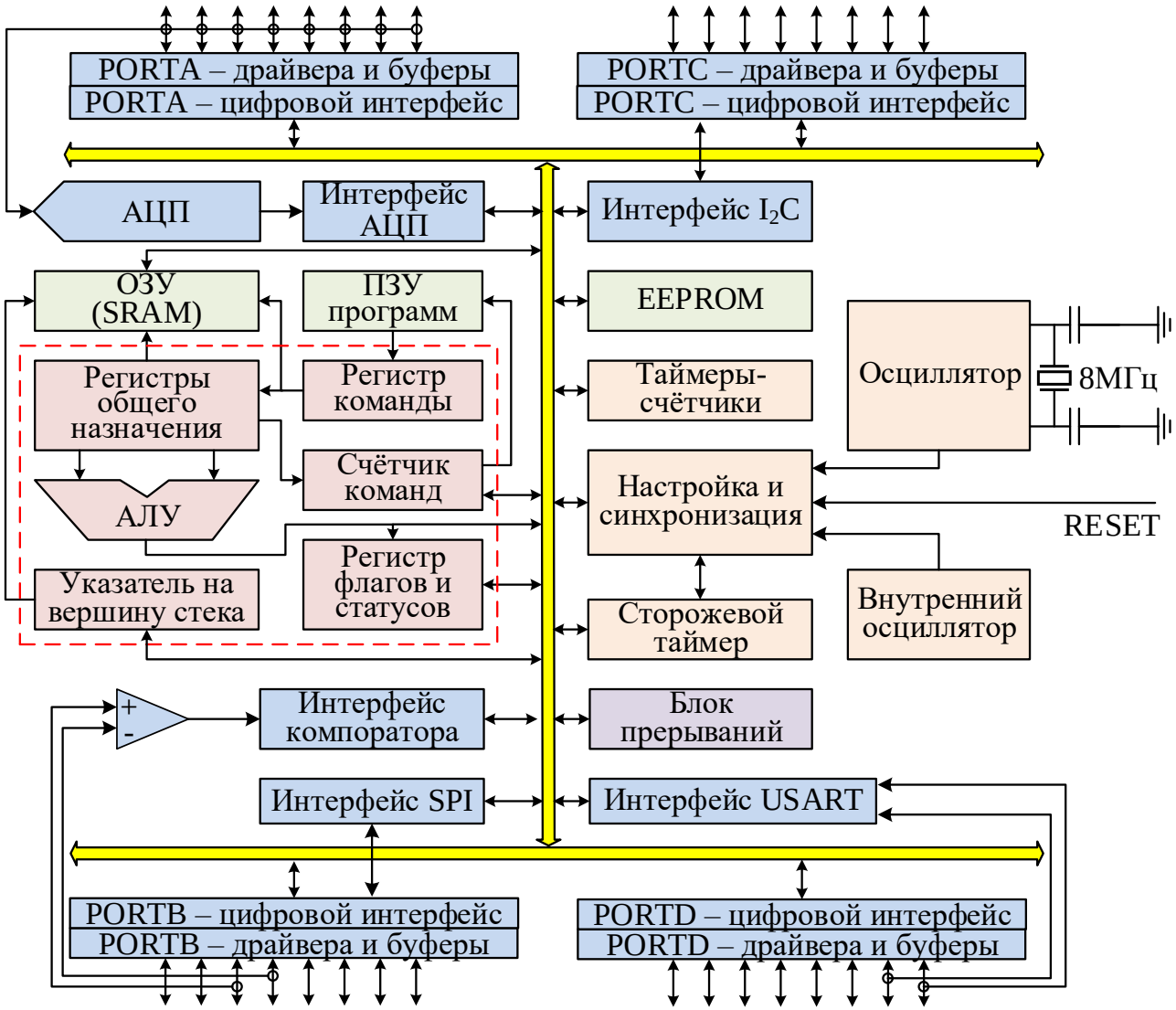
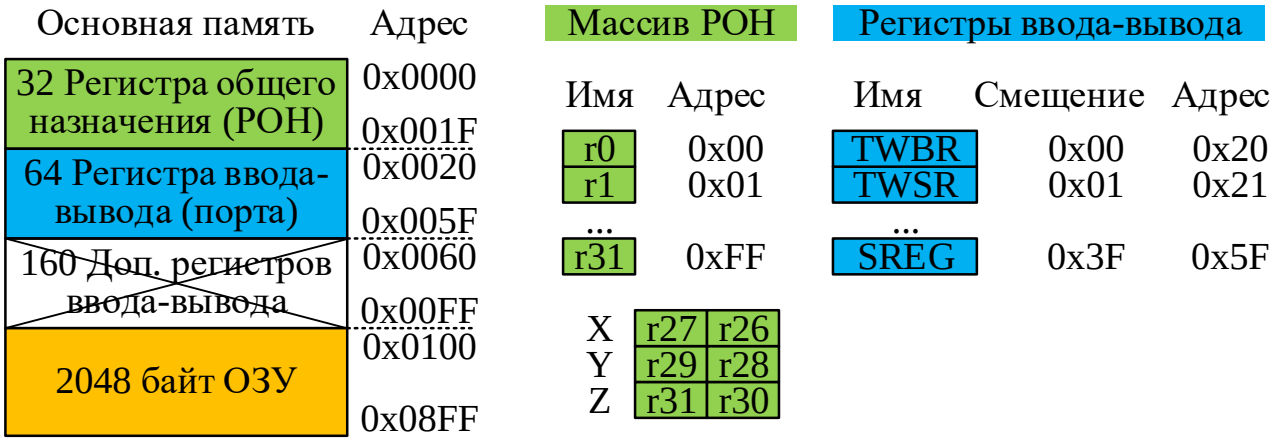


Рис. 1. блок-схема микроконтроллера Atmel AVR ATmega32

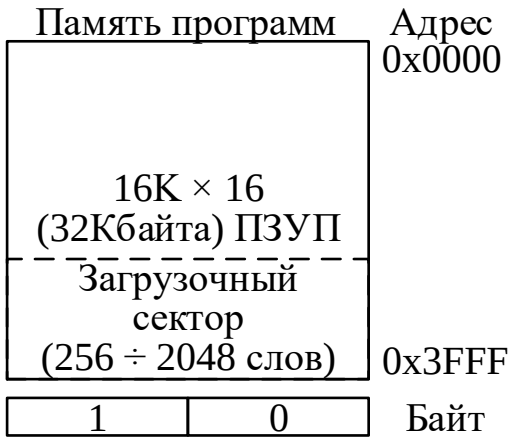
Встроенная ISP память позволяет перепрограммировать микроконтроллер через последовательный SPI интерфейс с помощью обычного программатора энергонезависимой памяти или с помощью встроенной программы загрузчика из ядра AVR. Программа загрузки может использовать любой интерфейс для загрузки приложения в память программ.

Адресное пространство МК

В МК используются два отдельных адресных пространства – одно для оперативной памяти (регистры общего назначения, регистры ввода-вывода, ОЗУ), второе для постоянной памяти программ (ПЗУП).



Оперативная память (данные)



Постоянная память программ

Регистр флагов и состояний SREG

Номер разряда	7	6	5	4	3	2	1	0
Имя флага	I	T	H	S	V	N	Z	C

Назначение разрядов регистра SREG:

Номер разряда	Обозначение	Имя флага	Описание
0	C	Carry flag	Флаг переноса. Устанавливается, если во время выполнения операции был перенос из старшего разряда результата
1	Z	Zero flag	Флаг нулевого результата. Устанавливается, если результат операции равен 0
2	N	Negative flag	Флаг отрицательного результата. Устанавливается, если MSB (Most Significant Bit - старший бит) результата равен 1 (правильно показывает знак результата, если не было переполнения разрядной сетки знакового числа)
3	V	Two's complement overflow	Флаг переполнения дополнения до двух. Устанавливается, если во время выполнения операции было переполнение разрядной сетки знакового результата

4	S	Sign flag	Бит знака, S = N XOR V. Бит S всегда равен исключающему ИЛИ между флагами N (отрицательный результат) и V (переполнение дополнения до двух). Правильно показывает знак результата и при переполнении разрядной сетки знакового числа
5	H	Half Carry flag	Флаг половинного переноса. устанавливается, если во время выполнения операции был перенос из 3-го разряда результата
6	T	Transfer bit	Хранение копируемого бита. Команды копирования битов BLD (Bit LoaD) и BST (Bit STore) используют этот бит как источник и приёмник обрабатываемого бита. Бит из регистра регистрового файла может быть скопирован в T командой BST, бит T может быть скопирован в бит регистрового файла командой BLD
7	I	Global interrupt flag	Общее разрешение прерываний. Для разрешения прерываний этот бит должен быть установлен в единицу. Управление отдельными прерываниями производится регистром маски прерываний - GIMSK/TIMSK. Если флаг сброшен (0), независимо от состояния GIMSK/TIMSK, прерывания не разрешены. Бит I очищается аппаратно после входа в прерывание и восстанавливается командой RETI, для разрешения обработки следующих прерываний

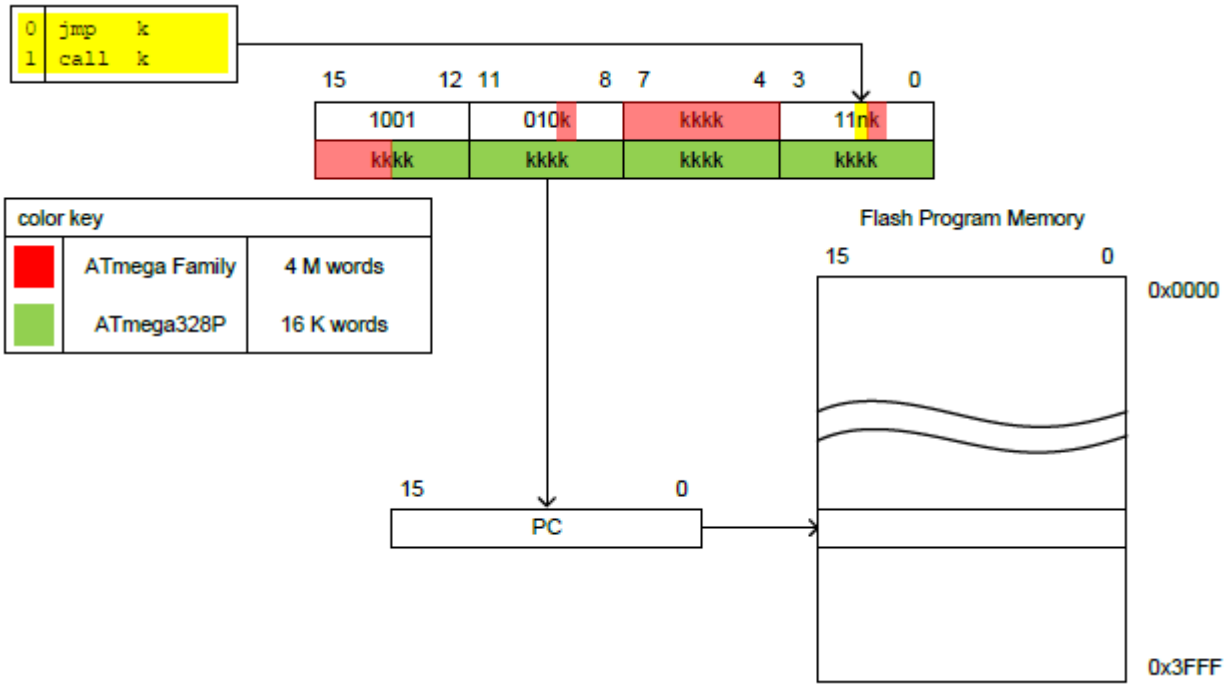
Форматы команд МК AVR ATmega32

Условное обозначение:
PC – счётчик команд
r и d – разряд адреса (номера) регистров общего назначение
A – разряд номера (адреса) регистров ввода-вывода (портов)
k – разряд адреса (при прямой адресации)
K – разряд константы (при непосредственной адресации)
q – разряд константы (при адресации со смещением)
b – разряд в регистре общего назначения или регистре ввода-вывода
s – разряд в регистре флагов и состояния SREG

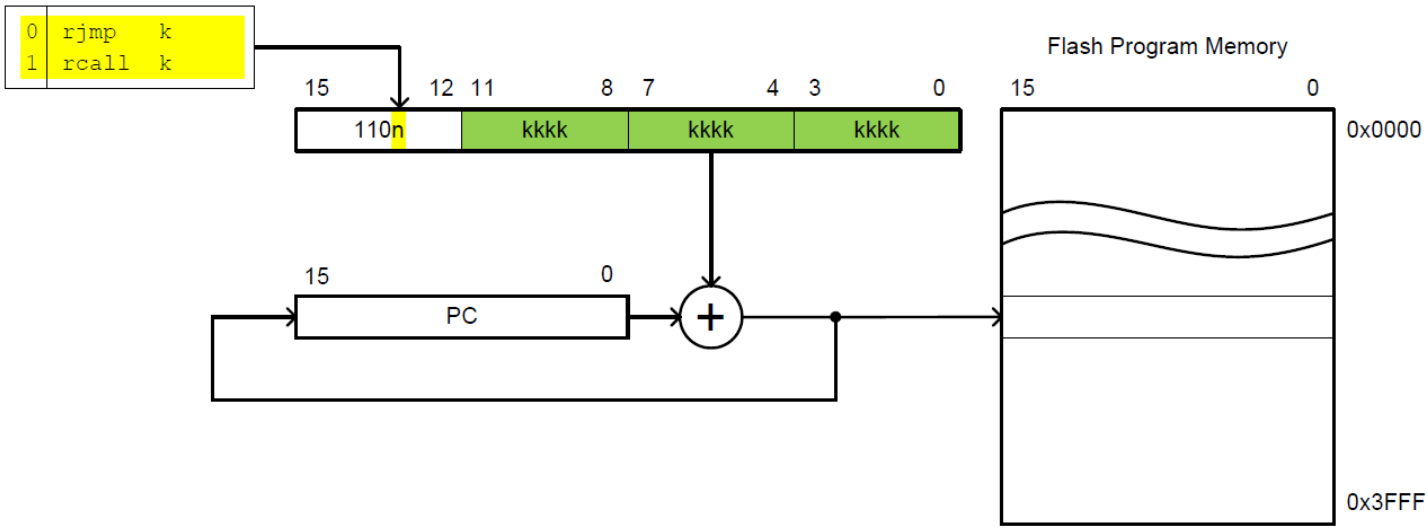
Диапазоны допустимых значений (по умолчанию):
$r \in [0;31]$
$d \in [0;31]$
$A \in [0;63]$
$s \in [0;7]$
$b \in [0;7]$

Команды передачи управления

Безусловные с прямой адресацией

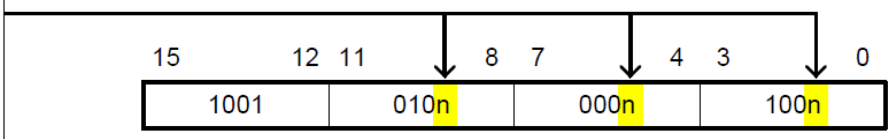


Безусловные с относительной адресацией

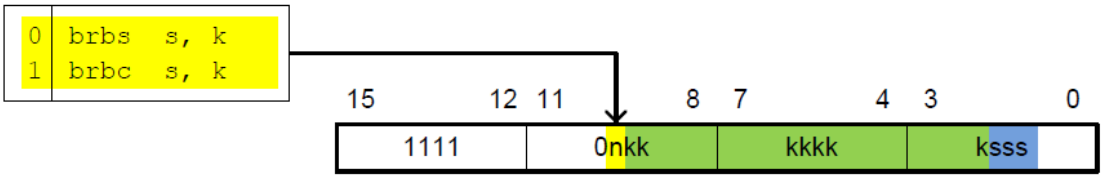


Безусловные с косвенно-регистровой адресацией

nnn	opcode	notes
000		
001	ijmp	see illustration
010		
011	eijmp	n/a to 328P
100	ret	PC ← M[SP + 1]
101	icall	see illustration
110	reti	PC ← M[SP + 1]
111	eicall	n/a to 328P



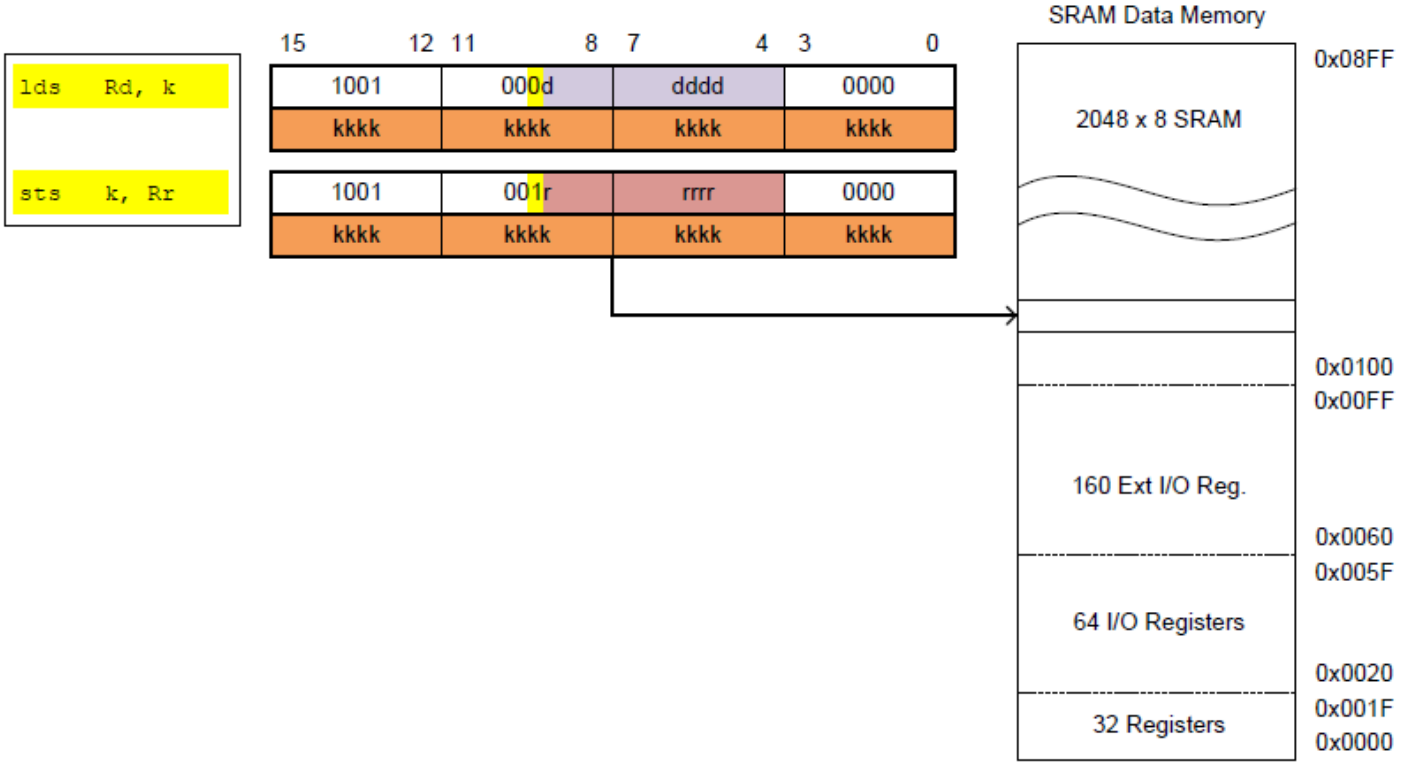
Условные (с относительной адресацией)



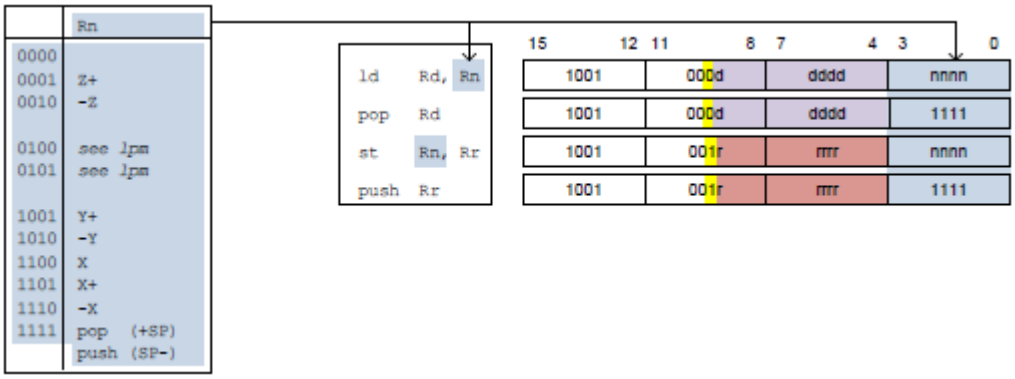
alias			
SREG	sss	brbs s, k	brbc s, k
I	111	brie k	brid k
T	110	brts k	brtc k
H	101	brhs k	brhc k
S	100	brlt k	brge k
V	011	brvs k	brvc k
N	010	brmi k	brpl k
Z	001	breq k	brne k
C	000	brcs k brlo k	brcc k brsh k

Команды передачи данных

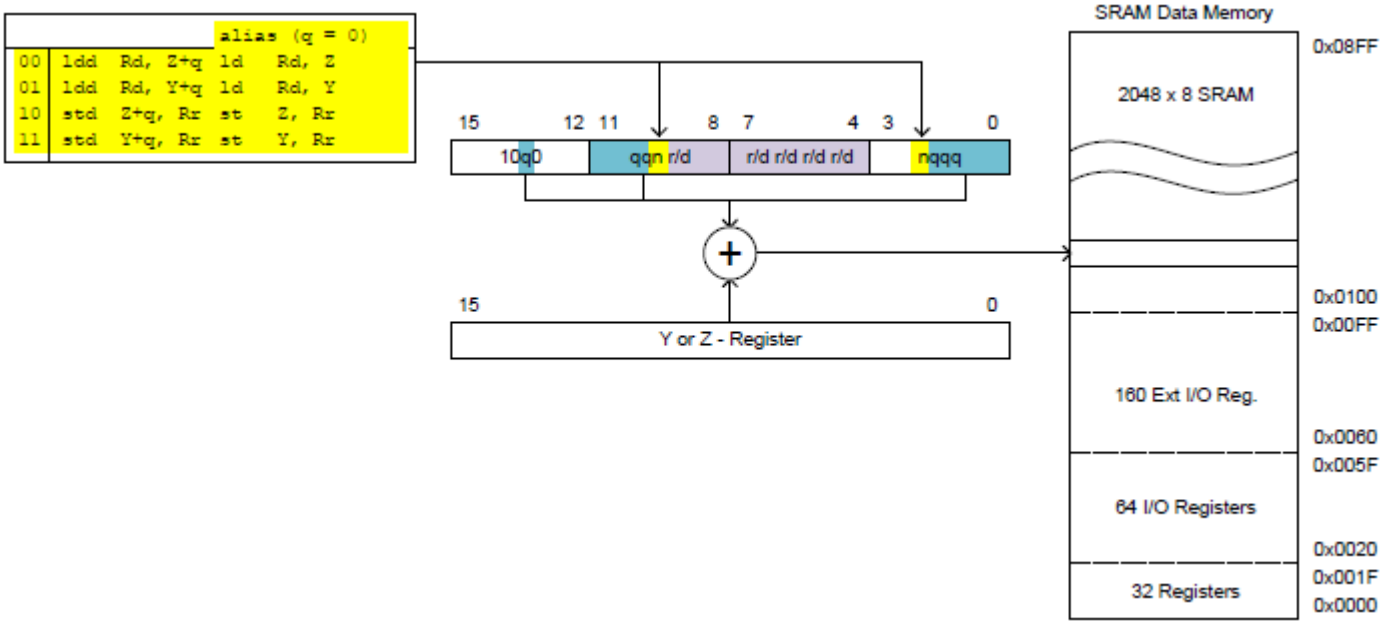
С прямой адресацией



С косвенно-регистровой адресацией

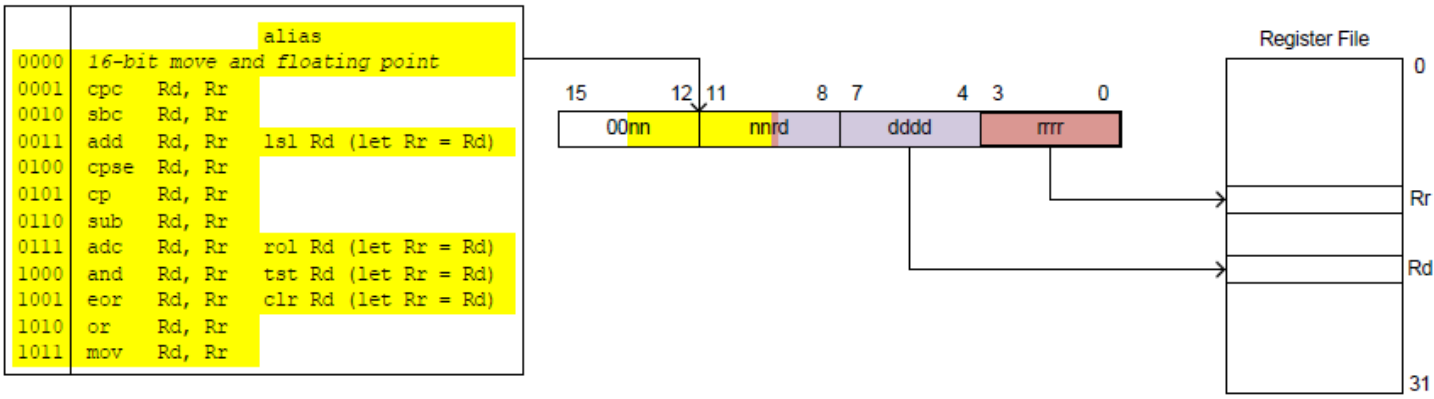


С косвенно-регистровой адресацией и смещением

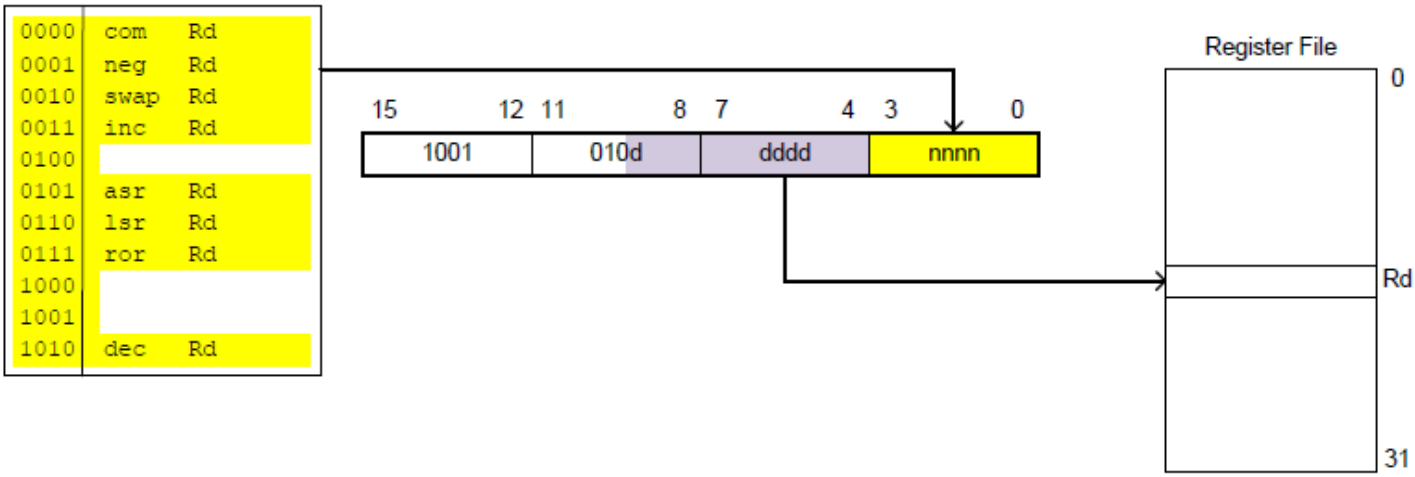


Арифметические и логические команды

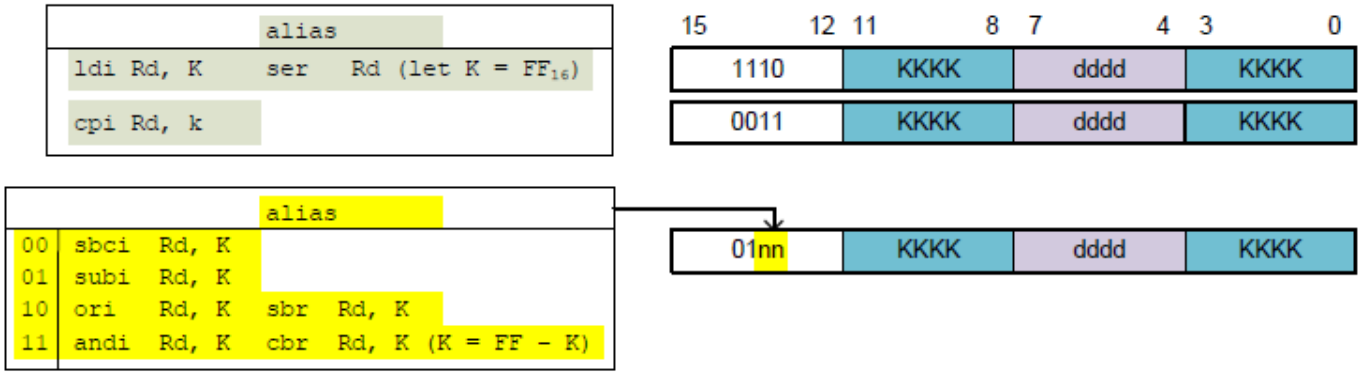
Бинарные арифметические и логические команды



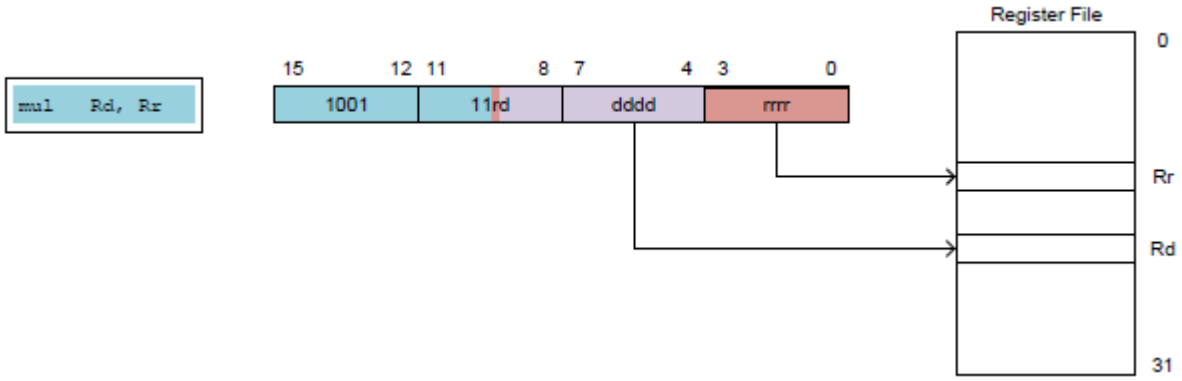
Унарные команды (безоперандные)



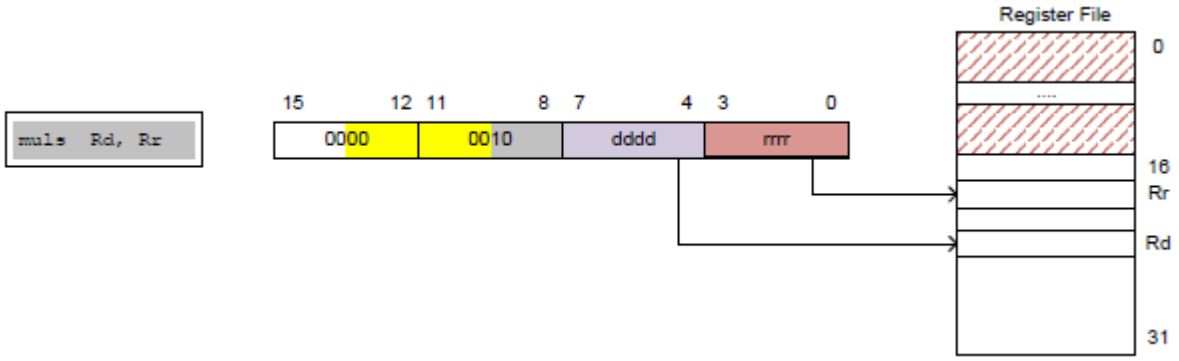
Унарные команды (с операндами)



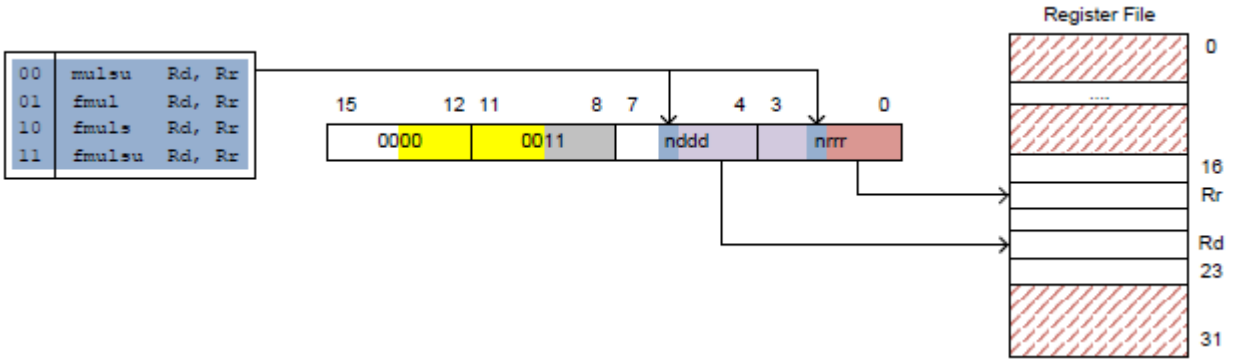
Беззнаковое умножение



Умножение чисел со знаком

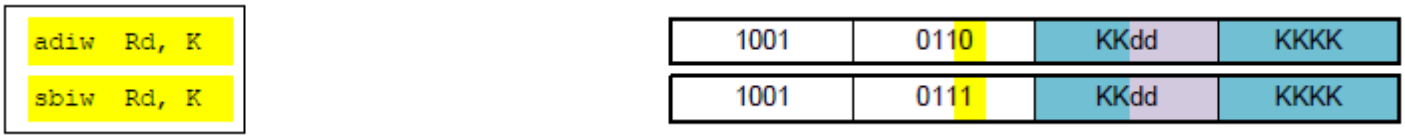


Умножение смешанных (беззнаковое с числом со знаком) чисел и дробное умножение

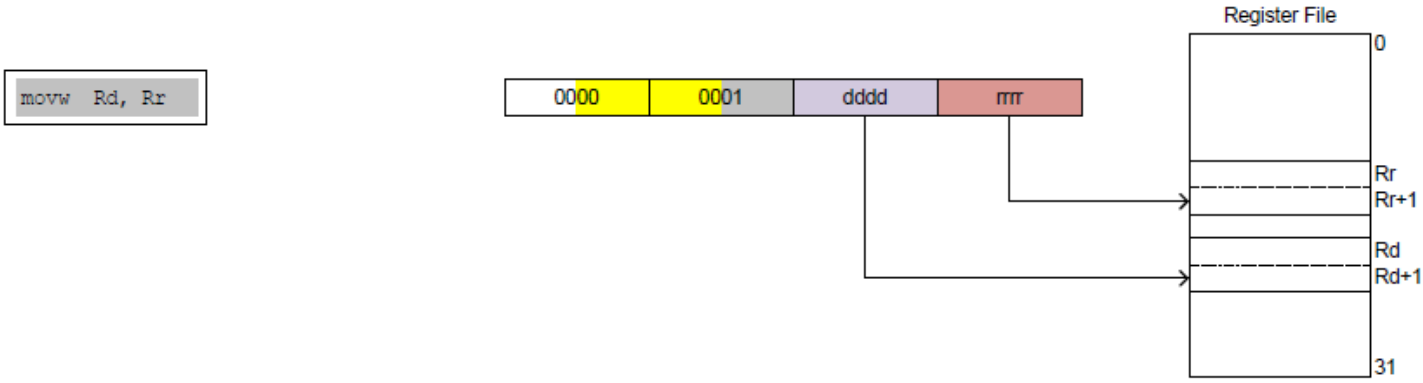


Команда для обработки 16-разрядных слов

Арифметические

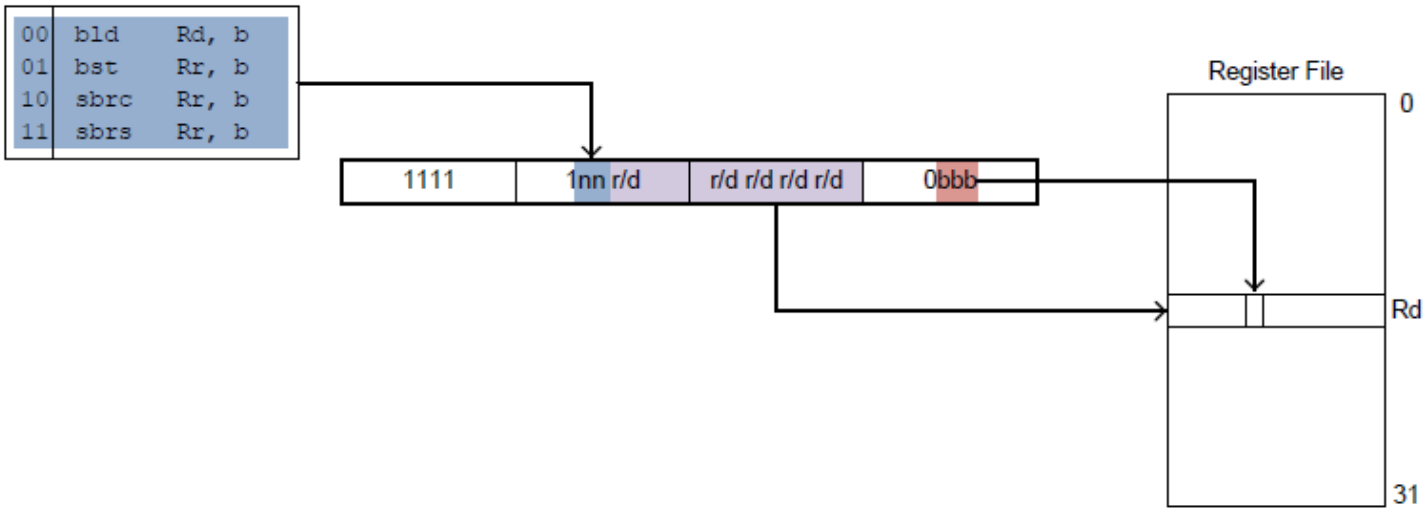


Пересылка

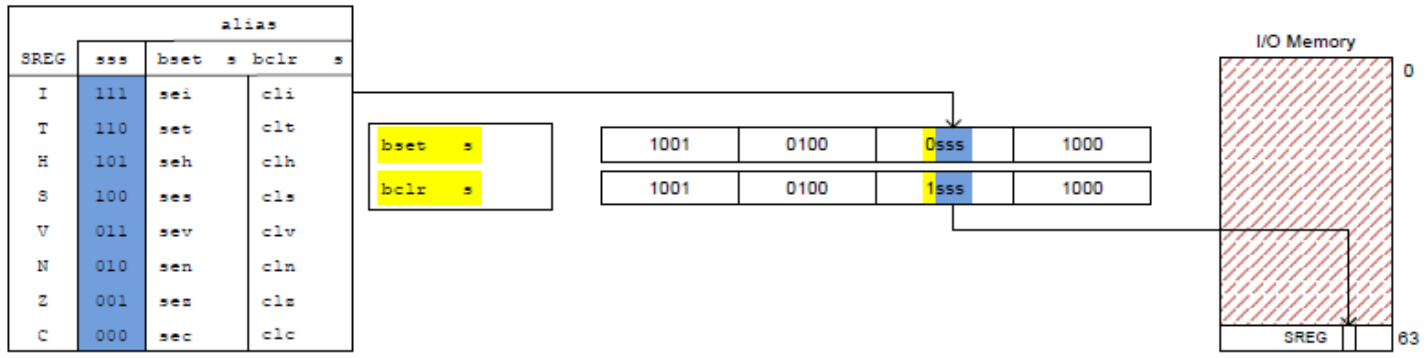


Битовые команды

Команды установки/очистки/управления по отдельным разрядам регистров

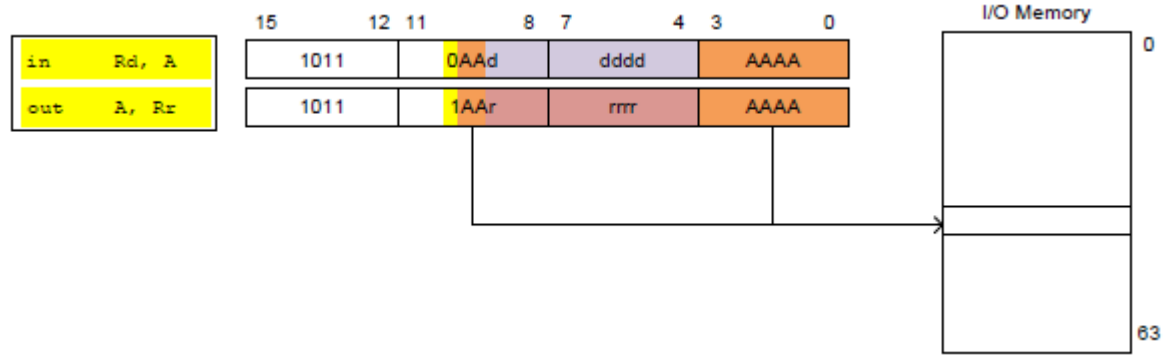


Команды установки флагов в регистре SREG

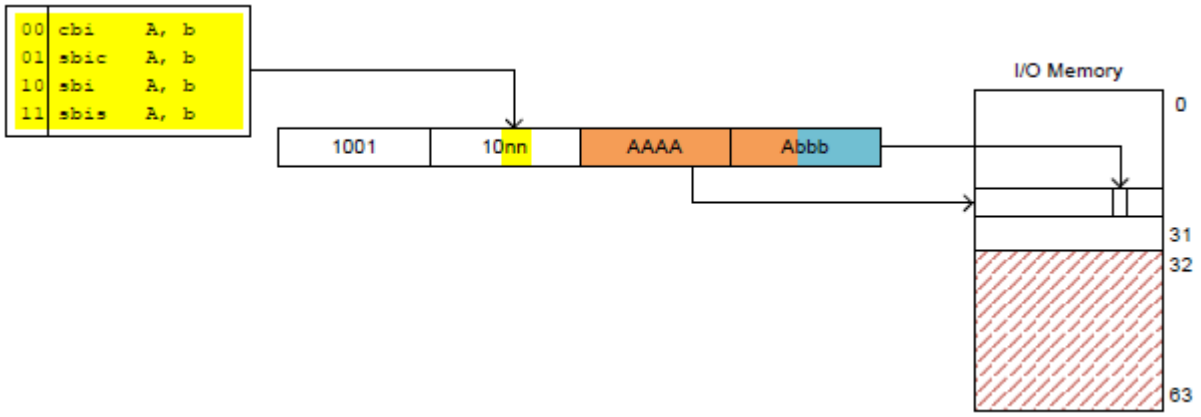


Команды конфигурирования портов ввода-вывода

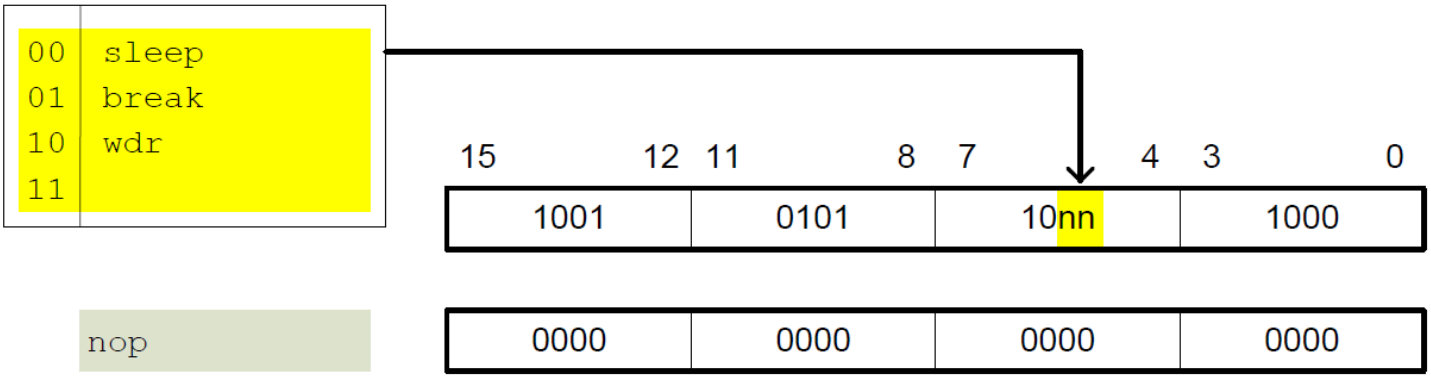
Команды установки/считывания всех разрядов портов ввода-вывода



Команды установки/считывания/переходов по отдельным разрядам портов ввода-вывода



Управляющие команды



Средства упрощения программирования на языке ассемблера

1. Переименование регистров:
.def TMP = R16
.def A0 = R20
.def A1 = R21

2. Сохранение констант:
.equ FREQ = 8000000

3. Макросы:
.macro outi ; Макрос для записи константы в порт, например, outi DDRD, 0xFC
ldi TMP, @1
out @0, TMP
.endm

Применение пользовательского разряда T в регистре SREG

Пользовательский разряд T регистра SREG позволяет программисту существенно упростить участки кода, оперирующие одноразрядной логической переменной. В МК Atmega32 существует всего 6 команд, предназначенные для работы с флагом T:

SET		Установка флага T	T ← 1

CLT		Очистка флага T	T ← 0
BST	Rr, b	Сохранить бит регистра Rr во флаг T регистра SREG	T ← Rr(b)
BLD	Rd, b	Считать флаг T регистра SREG в бит регистра Rd	Rd(b) ← T
BRTS	k, k∈[-64;63]	Переход при установленном флаге передачи (T=1)	if (T = 1) then PC ← PC + k + 1
BRTC	k, k∈[-64;63]	Переход при снятом флаге передачи (T=0)	if (T = 0) then PC ← PC + k + 1

Примеры использования:

1. Программа имеет два режима работы, в зависимости от которого изменяется логика работы основного цикла и прерываний

```
    _main:
CLT; установка первого режима
…
    _loop:
BRTS _loop_mode2; нужна 1 команда для определения режима
_loop_mode1: ; начало обработки первого режима программы
…
RJMP _loop
    _loop_mode2: ; начало обработки второго режима программы
…
RJMP loop

    _int0:
SET; переключение на второй режим работы
RETI

    _int1:
BLD TMP, 0
COM TMP ; переключение между режимами работы
BST TMP, 0
RETI

    _timer0_cmp:
BRTS _ timer0_mode2; нужна 1 команда для определения режима
_ timer0_mode1: ; начало обработки первого режима программы
…
RJMP _timer0_exit
    _ timer0_mode2: ; начало обработки второго режима программы
…
; RJMP _timer0_exit
    _timer0_exit:
RETI
```

2. Организация циклического сдвига

```
BST A0, 0 ; сохранение младшего бита во флаге T
LSR A0 ; логический сдвиг вправо
BLD A0, 7 ; заполнение 7 бита значением из флага T
```

Обработка многоразрядных данных в МК Atmega32

1. В МК Atmega32 существует ряд команд, предназначенных для обработки регистровых пар. В основном эти команды используются для

работы с десятиразрядными адресами в ОЗУ и имеют существенные ограничения:

ADIW	Rdl, K	dl∈{24,26,28,30}, K∈[0;63]	Сложение константы с регистровой парой	Rdh:Rdl ← Rdh:Rdl + K
SBIW	Rdl,K	dl∈{24,26,28,30}, K∈[0;63]	Вычитание константы из регистровой пары	Rdh:Rdl ← Rdh:Rdl - K
MOVW	Rd, Rr	d∈{0,2,…,30}, r∈{0,2,…,30}	Копирование регистровой пары	Rd+1:Rd ← Rr+1:Rr

2. Существуют ряд команд, учитывающих состояние флага переноса, что позволяет использовать их для обработки фрагментов многоразрядных чисел:

ADC	Rd, Rr	Сложение с учётом флага переноса	Rd ← Rd + Rr + C
SBC	Rd, Rr	Вычитание с учётом флага переноса	Rd ← Rd - Rr - C
SBCI	Rd, K, d∈[16;31]	Вычитание константы с учётом флага переноса	Rd ← Rd - K - C
ROL	Rd	Циклический сдвиг влево (через флаг переноса)	Rd(0)←C, Rd(n+1)← Rd(n), C←Rd(7)
ROR	Rd	Циклический сдвиг вправо (через флаг переноса)	Rd(7)←C, Rd(n)← Rd(n+1), C←Rd(0)
CPC	Rd,Rr	Сравнение с учётом флага переноса	Rd - Rr - C

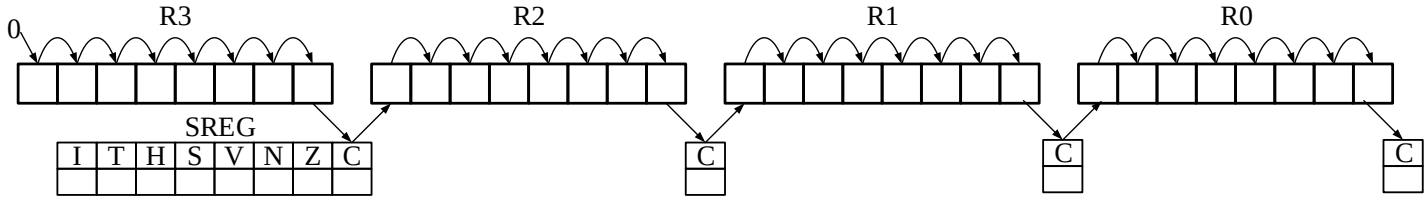
Примеры обработки многоразрядных чисел:

1. Сравнение двух 32-разрядных чисел:

```
CPI R0, R3
CPC R1, R4 ; сравнение с учётом результата предыдущего сравнения
CPC R2, R5
```

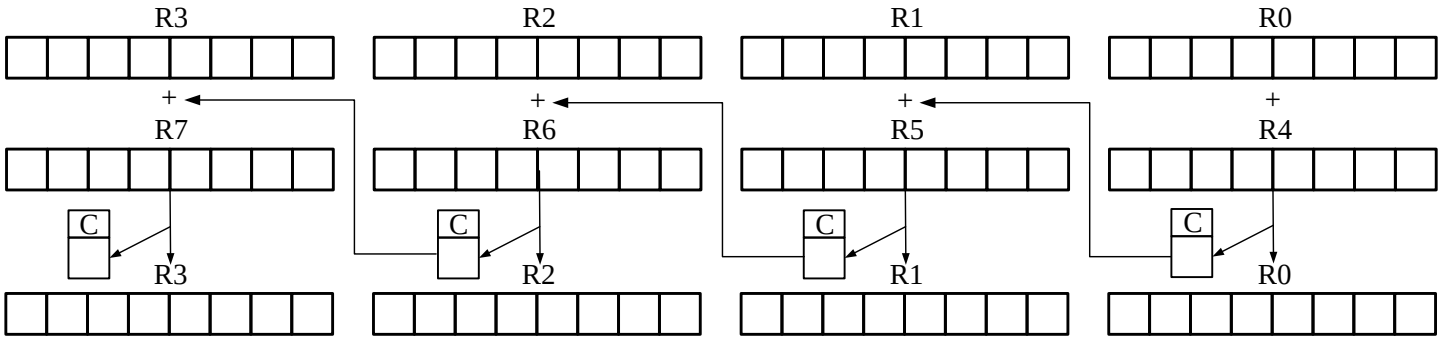
2. Арифметический сдвиг вправо 32-разрядного числа:

```
LSR R3
ROR R2 ; сдвиг с учётом младшего разряда старшего байта
ROR R1
ROR R0
```

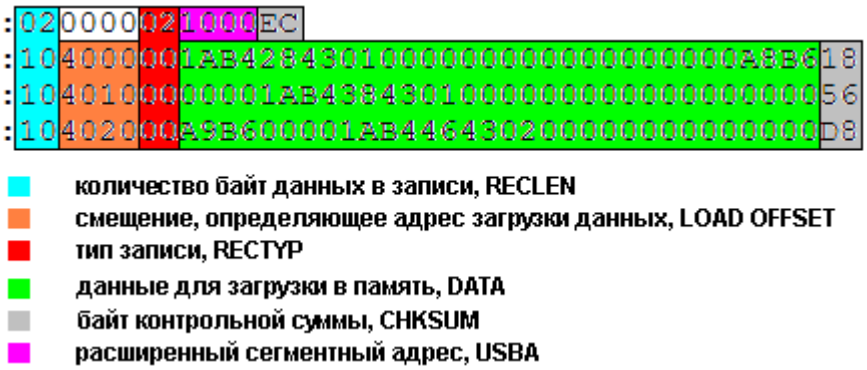


3. Сложение двух 32-разрядных чисел:

```
ADD R0, R4
ADC R1, R5 ; сложение с учётом переполнения при сложении младших байтов
ADC R2, R6
ADC R3, R7
```



Формат «.hex»-файла



Шестнадцатеричное представление двоичного файла закодировано с помощью цифробуквенных символов ASCII. Шестнадцатеричный объектный файл разбит на блоки записей (строки), каждая из которых содержит тип записи, длину, адрес загрузки в память и дополнительную контрольную сумму. Существует три основных типа записей:

- **Data Record**, запись для данных (8-, 16- или 32-bit форматы)
- **End of File Record**, запись для сигнала о конце файла (8-, 16- или 32-bit форматы)
- **Extended Segment Address Record**, запись для адреса расширенного сегмента (16- или 32-bit форматы)

Общий формат записи (General Record Format)

RECORD MARK ':'	RECLEN	LOAD OFFSET	RECTYP	INFO или DATA	CHKSUM
1 байт	1 байт	2 байта	1 байт	n байт	1 байт

Каждая запись в HEX-файле (строка) начинается с поля маркера начала записи **RECORD MARK**, содержащего ASCII-код 03AH, символ двоеточия ':'.

Следующее поле каждой записи **RECLEN**, которое задаёт количество байт полезной информации, которая содержится в записи (эти байты идут за полем RECTYP). Помните, что один байт данных представлен двумя символами ASCII. Максимальное значение для поля RECLEN является шестнадцатеричное 'FF', т. е. полезных данных в записи может быть от 0 до 255 байт.

Следующее поле в каждой записи **LOAD OFFSET**, которое указывает 16-битный начальный адрес загрузки байт данных, так что это поле используется только для записей данных (Data Record). В других типах

записей, где это поле не используется, оно должно быть закодировано четырьмя ASCII-символами нуля ('0000' или 030303030H).

Следующее поле в каждой записи **RECTYP**, которое обозначает тип этой записи. Поле RECTYP используется для интерпретации информации в записи. Вот как кодируются типы записей:

- '**00**' Data Record (запись, содержащая данные)
- '**01**' End of File Record (запись, сигнализирующая о конце файла)
- '**02**' Extended Segment Address Record (запись адреса расширенного сегмента)

Data Record (8-, 16- или 32-битный формат)

RECORD MARK ':'	RECLEN	LOAD OFFSET	RECTYPE '00'	DATA	CHKSUM
1 байт	1 байт	2 байта	1 байт	n байт	1 байт

Запись Data Record предоставляет набор шестнадцатеричных цифр, в которых находится ASCII-код байтов данных, которые содержатся в порции данных образа памяти. В отдельных полях записи имеется следующее содержимое:

- RECORD MARK – Это поле содержит байт 03AH, символ двоеточия, закодированный шестнадцатеричным символом ASCII (':').
- RECLEN – Это поле содержит две шестнадцатеричные цифры ASCII, которые задают количество байт данных, находящихся в записи. Максимальное значение равно 'FF' или 04646H (что соответствует десятичному значению 255).
- LOAD OFFSET – Это поле содержит четыре шестнадцатеричные цифры ASCII, представляющие смещение от LBA (см. раздел Extended Linear Address Record) или SBA (см. раздел Extended Segment Address Record), что задает адрес, куда будет помещен первый байт записи.
- RECTYP – Это поле содержит байты 03030H, кодирующие шестнадцатеричными символами ASCII '00', что задает тип записи Data Record.
- DATA – Это поле содержит пары шестнадцатеричных цифр ASCII, где каждая пара кодирует один байт данных.
- CHKSUM – Это поле содержит контрольную сумму полей RECLEN, LOAD OFFSET, RECTYP и DATA.

Пример расчёта времени выполнения фрагмента кода

```
delay:
        LDI     R17, 198;  y = 198
        LDI     R16, 102;  x = 102
delay_sub:
        DEC     R16
        BRNE    delay_sub      } Внутренний цикл
        DEC     R17
        BRNE    delay_sub      } Внешний цикл
        NOP
```

Команда	Число тактов
LDI	1 - загрузка
DEC	1 - декремент
BRNE	1 - сравнение 1/0 - переход
NOP	1 - простой

Флаг-бит Z устанавливается в 1 при нулевом результате операции
Команда условного перехода BRNE выполняет переход, если Z = 0
Число тактов до первого выхода из внутреннего цикла:

$N_{вн.1} = (1_{DEC} + 2_{BRNE}) \times (x - 1) + 1_{DEC} + 1_{BRNE} = 3 \times x - 1$

Число тактов до последующих выходов из внутреннего цикла:

$N_{вн.x} = (1_{DEC} + 2_{BRNE}) \times (2^8 - 1) + 1_{DEC} + 1_{BRNE} = 3 \times 2^8 - 1$

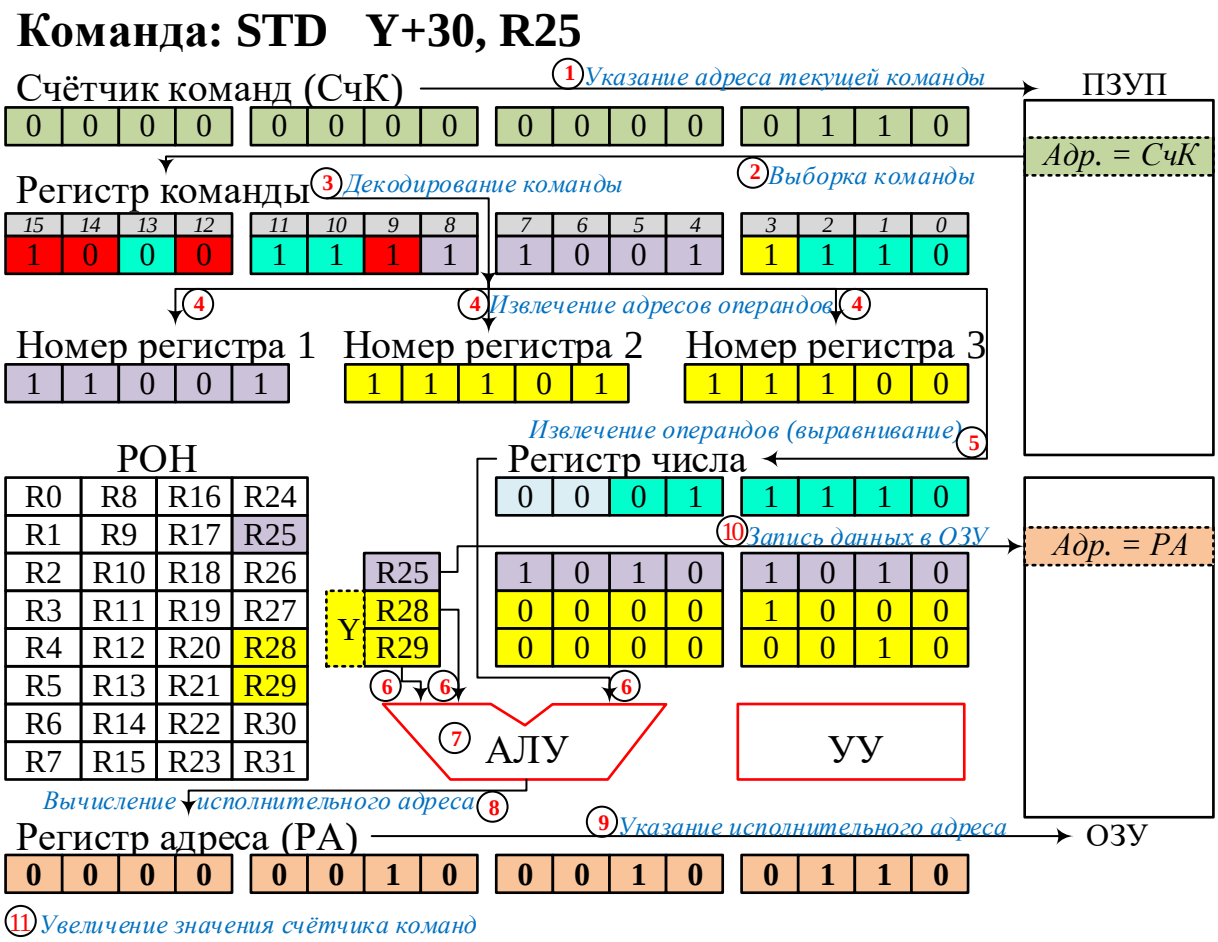
Общее число тактов:

$N_{общ} = 1_{LDI} \times 2 + (N_{вн.1} + 1_{DEC} + 2_{BRNE}) + (N_{вн.x} + 1_{DEC} + 2_{BRNE}) \times (y - 2) + (N_{вн.x} + 1_{DEC} + 1_{BRNE}) + 1_{NOP} =$

$= 3 \times x + 770 \times (y - 1) + 4 = 152\,000$

Время выполнения: $t = N / T = 152\,000 / 8\,000\,000 = 0{,}019\ с = 19\ мс$

Пример порядка выполнения команды STD в МК ATmega32



Этапы выполнения команд:

1. Передача памяти программ (ПЗУП) адреса текущей команды
2. Извлечение команды из ПЗУП и запись в регистр команды
3. Декодирование команды – определение типа операции и формата
4. Извлечение адресов операндов (номеров регистров общего назначения (РОН))
5. Извлечение операндов из команды (непосредственная адресация) и выравнивание данных (в данном случае расширение знакового операнда с 6 до 8 разрядов)
6. Передача операндов в сумматор
7. Выполнение операции суммирования
8. Передача результата операции суммирования в регистр адреса
9. Передача исполнительного адреса в ОЗУ
10. Запись данных из R25 в ОЗУ
11. Увеличение значения счётчика команд

Для других команд могут встречаться этапы считывания или записи флагов регистра SREG, чтение или запись из портов ввода-вывода и другие, неуказанные в примере выше.

Порядок выполнения работы

1. В базовом примере (см. ниже) заменить фрагмент кода, выделенный зелёным, на фрагмент из своего варианта.

2. Создать новый проект в Atmel Studio на языке ассемблера, разместить исходный код из п.1 в файле проекта. Сформировать «.lss»-файл и «.hex»-файл. Записать «.hex»-файл в контроллер с помощью программы AVRFlash.

3. Изучить архитектуру и систему команд микроконтроллера ATmega32. Разобраться в алгоритме работы текущей программы. Убедиться в правильности работы программы.

4. Определить зависимость количества тактов, за которое выполняется заменённый блок кода, от констант x и y.

5. Вычислить значения констант x и y при которых блок кода «delay» будет выполняться ровно 0,01 секунды (при тактовой частоте 8 МГц). При невозможности обеспечения точной величины задержки необходимо дополнить блок кода соответствующим количеством команд NOP (на месте закомментированной команды NOP в базовом примере).

6. Изучить сформированный «.lss»-файл, выписать адреса всех меток программы, перечислить используемые форматы команд в части состава и размерности операндов.

7. Изучить структуру сформированного «.hex»-файла, определить количество записей в файле и количество машинных слов программы.

8. Взять команду ассемблера в соответствии с вариантом и описать порядок её выполнения внутри центрального процессора: определить этапы командного цикла, задействованные узлы МК, состав пересылаемых данных и управляющих сигналов.

Базовый пример

```
.def TMP = R20

.org $000
   JMP reset ; Указатель на начало программы

; Функция паузы
delay: ; задержка 120 мс
   LDI  R31, 5
   LDI  R30, 223
   LDI  R29, 188
delay_sub:
   DEC  R29
   BRNE delay_sub
   DEC  R30
```

```
    BRNE delay_sub
    DEC  R31
    BRNE delay_sub
    NOP
    NOP
    RET

; Начальная настройка
reset:
; настройка исходных значений
    LDI  TMP, 0x01;
    MOV  R0, TMP
    CLR  TMP;
    MOV  R1, TMP
    MOV  R2, TMP
    MOV  R3, TMP
; настройка портов ввода-вывода
    SER  TMP ; 0xFF
    OUT  DDRA, TMP ; Вывод
    OUT  DDRB, TMP ; Вывод
    OUT  DDRC, TMP ; Вывод
    OUT  DDRD, TMP ; Вывод
; Установка вершины стека в конец ОЗУ
    LDI  TMP, HIGH(RAMEND) ; Старшие разряды адреса
    OUT  SPH, TMP
    LDI  TMP, LOW(RAMEND) ; Младшие разряды адреса
    OUT  SPL, TMP

; Основной цикл
loop:
; Циклический сдвиг 32-разрядного числа R0-R3
    BST R0, 0 ; сохранение младшего бита во флаге T
    LSR R3 ; логический сдвиг вправо
    ROR R2 ; циклический сдвиг вправо
    ROR R1 ; циклический сдвиг вправо
    ROR R0 ; циклический сдвиг вправо
    BLD R3, 7 ; заполнение 7 бита значением из флага T
; Вывод 32-разрядного числа R0-R3 на порты PORTA-PORTD
    OUT PORTA, R0
    OUT PORTB, R1
    OUT PORTC, R2
    OUT PORTD, R3
; Пауза
    CALL delay;
; Возврат в начало основного цикла
    RJMP loop ;
```

Содержание отчёта

1. Схема установки (задействованные узлы отладочной платы).
2. Блок-схема алгоритма работы программы.
3. Комментированный листинг программы на языке ассемблера.

4. Алгоритм выполнения задействованных в программе команд ассемблера.

5. Ответы на контрольные вопросы.

Контрольные вопросы

1. Укажите, в чём проявляются признаки RISC-архитектуры в микроконтроллере ATmega32. В чём преимущества и недостатки приведённых особенностей?

2. От чего зависит время выполнения команд SBIS и BRLT? Приведите примеры кода (до 3-10 команд каждый), приводящие к различной продолжительности выполнения указанных команд

3. Укажите команды и их аргументы в виде имён регистров общего назначения, портов ввода-вывода и констант в десятичной или шестнадцатеричной системе счисления для следующих машинных слов: 1001 0110 0111 1111, 1111 0011 1110 1101, 1010 1010 0011 1010 и 1001 1010 0011 0110?

4. Приведите пример выполнения циклического сдвига вправо 8-разрядного числа (с переходом младшего разряда в старший) без использования флага T. Приведите пример из трёх машинных команд, обеспечивающих сложение 24-разрядного числа с 24-разрядной константой.

5. Обоснуйте, чем вызваны ограничения допустимых значений номеров регистров и диапазонов констант в некоторых командах микроконтроллера ATmega32?

Варианты заданий – заменяемые блоки кода

Номер варианта	Фрагмент кода (п. 1)	Команда (п. 8)
1а	delay:	LD R30, X+
1б	LDI R30, 200; y	OUT DDRD, R5
1в	LDI R29, 153; x	CBR R20, 100
	delay_sub:	
	DEC R29	
	BRNE delay_sub	
	NOP	
	DEC R30	
	BRNE delay_sub	
	RET	

2а	delay:	SBRS R6, 3
2б	LDI R30, 200; y	IN PINA, R4
2в	LDI R29, 153; x delay_sub: DEC R29 NOP BRNE delay_sub DEC R30 NOP BRNE delay_sub RET	ADIW R30, 10
3а	delay:	SUB R21, R1
3б	LDI R30, 200; y	SBR R30, 76
3в	LDI R29, 153; x delay_sub: INC R29 BRNE delay_sub NOP DEC R30 BRNE delay_sub RET	PUSH R0
4а	delay:	SBIW R30, 63
4б	LDI R30, 200; y	CPSE R0, R1
4в	LDI R29, 153; x delay_sub: INC R29 BRNE delay_sub NOP INC R30 BRNE delay_sub RET	LD R30, Y
5а	delay:	SBRC R17, 3
5б	LDI R30, 200; y	MULSU R19, R20
5в	LDI R29, 153; x delay_sub: DEC R29 NOP BRNE delay_sub DEC R30 BRNE delay_sub RET	SBI DDRC, 3
6а	delay:	SBIS PORTA, 0

6б	LDI R30, 10; y	MULS R21, R20
6в	LDI R29, 153; x	LD R5, Z+
	delay_sub:	
	INC R29	
	NOP	
	BRNE delay_sub	
	NOP	
	INC R30	
	BRNE delay_sub	
	RET	
7а	delay:	ADD R12, R8
7б	LDI R30, 200; y	SBI PORTD, 7
7в	LDI R29, 153; x	ADC R15, R16
	delay_sub:	
	INC R29	
	NOP	
	BRNE delay_sub	
	NOP	
	DEC R30	
	BRNE delay_sub	
	RET	
8а	delay:	SBIC DDRC, 1
8б	LDI R30, 200; y	BST R3, 7
8в	LDI R29, 153; x	BRLT -20
	delay_sub:	
	INC R29	
	BRNE delay_sub	
	DEC R30	
	NOP	
	BRNE delay_sub	
	RET	
9а	delay:	CLR R7
9б	LDI R30, 200; y	ANDI R18,100
9в	LDI R29, 153; x	CPSE R6, R7
	delay_sub:	
	NOP	
	DEC R29	
	NOP	
	BRNE delay_sub	
	INC R30	
	BRNE delay_sub	
	RET	

10а	delay:	CBI ADMUX, 5
10б	LDI R30, 200; y	MUL R0, R2
10в	LDI R29, 153; x	ROL R7
	delay_sub:	
	INC R29	
	NOP	
	BRNE delay_sub	
	DEC R30	
	NOP	
	NOP	
	BRNE delay_sub	
	RET	