МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПУТЕЙ СООБЩЕНИЯ (МИИТ)

Кафедра
«Управление и информатика в технических системах»

Л. Н. Логинова, А. И. Сафронов

ЯЗЫК АССЕМБЛЕРА
ДЛЯ МИКРОКОНТРОЛЛЕРОВ *ATMEGA8535*

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
К ЛАБОРАТОРНЫМ РАБОТАМ

по дисциплине

«Машинно-ориентированные языки»

МОСКВА – 2011

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПУТЕЙ СООБЩЕНИЯ (МИИТ)

Кафедра
«Управление и информатика в технических системах»

Л. Н. Логинова, А. И. Сафронов

ЯЗЫК АССЕМБЛЕРА ДЛЯ МИКРОКОНТРОЛЛЕРОВ *ATMEGA8535*

Рекомендовано редакционно-издательским советом
университета в качестве методических указаний
для студентов специальности
«Управление и информатика в технических системах»

МОСКВА – 2011

УДК 004

Л 69

Логинова Л. Н., Сафронов А. И. Язык Ассемблера для микроконтроллеров *ATmega8535*: Методические указания к лабораторной работе по дисциплине «Машинно-ориентированные языки». – М.: МИИТ, 2011. – 84 с.

В методических указаниях рассмотрена система команд микроконтроллеров *ATmega8535*. Рассмотрена *AVR Studio* – интегрированное отладочное средство для микроконтроллеров фирмы *Atmel* семейства *AVR*, включающее в себя компилятор с языка ассемблер.

Введение

Разнообразные устройства связи, радиоавтоматики или аудиовизуальной аппаратуры требуют присутствия в своём составе устройства управления (УУ) – контроллера. Контроллеры требуются практически во всех устройствах окружающей действительности.

Одним из самых распространённых в настоящее время является микроконтроллер фирмы «*Atmel*» из семейства *AVR* [1]. При том, что они появились на рынке в 1996 году, их популярность до сих пор невероятно высока. С каждым годом они захватывают всё новые и новые ниши на рынке микропроцессорной техники. Не последнюю роль в этом играет соотношение показателей цена / быстродействие / энергопотребление. *AVR* до сих пор является едва ли не лучшим на рынке 8-битных микроконтроллеров.

1. Описание и характерные особенности микроконтроллеров *ATmega8535*

Как и все микроконтроллеры *AVR* фирмы «*Atmel*», микроконтроллеры семейства *Mega*, в частности, *ATmega8535,* являются 8-разрядными микроконтроллерами, предназначенными для встраиваемых приложений. Они изготавливаются по малопотребляющей КМОП-технологии, которая в сочетании с усовершенствованной *RISC*-архитектурой[1] позволяет достичь наилучшего соотношения быстродействие / энергопотребление [2]. Контроллеры описываемого семейства являются наиболее развитыми представителями микроконтроллеров *AVR*.

К некоторым особенностям микроконтроллера *ATmega8535* относятся [2]:

– *FLASH*-память программ объемом 8 Кбайт с возможностью внутрисистемного перепрограммирования

[1] См приложение

и загрузки через последовательный канал *SPI* (число циклов стирания / записи не менее 1000);
– оперативная память (статическое ОЗУ – далее СОЗУ) объемом 512 байт;
– энергонезависимая память данных (*EEPROM*) объёмом 512 байт с возможностью внутрисистемного перепрограммирования и загрузки через последовательный канал *SPI* (число циклов стирания / записи не менее 100000);
– возможность защиты от чтения и модификации памяти программ и данных;
– возможность программирования непосредственно в системе через последовательные интерфейсы *SPI* и *JTAG*;
– возможность программного снижения частоты тактового генератора;
– 130 команд, большинство из которых выполняются за один машинный цикл;
– 17 внутренних + 3 внешних источников прерываний;
– наличие программного стека;
– наличие аппаратного умножителя;
– 32 8-битных регистра общего назначения (далее РОН);
– 32 программируемые линии ввода / вывода;
– диапазон напряжений питания от 4,5 В до 5,5 В;
– производительность до 8 *MIPS* при частоте 8 МГц;
– и т.д.

1.1. Устройства ввода / вывода *ATmega8535*

Микроконтроллеры семейства *Mega* имеют наиболее богатый набор периферийных устройств (ПУ). При этом в большинстве моделей имеются все ПУ, которые вообще встречаются в составе микроконтроллеров *AVR*. У микроконтроллера *ATmega8535* имеются в наличии [2]:

1. Многофункциональные, двунаправленные *GPIO* порты ввода-вывода с встроенными нагрузочными резисторами. Конфигурация портов ввода / вывода задаётся программным способом.
2. Два 8-разрядных таймера / счётчика (таймеры T0 и T2).
3. 16-разрядный таймер / счётчик (таймер T1).
4. 4 канала ШИМ-модулятора разрядностью 8 бит (один из режимов работы 8-разрядных таймеров / счётчиков T0 и T2).
5. Аналоговый компаратор.
6. Восьмиканальный 10-разрядный АЦП с дифференциальными входами:
   а) программируемый коэффициент усиления перед АЦП 1, 10 и 200;
   б) опорное напряжение 2,56 В.
7. Полнодуплексный универсальный асинхронный приемопередатчик *UART*.
8. Последовательный синхронный интерфейс *SPI*.
9. Последовательный двухпроводный интерфейс *TWI* (аналог интерфейса *I2C*).

1.2. Архитектура микроконтроллера *ATmega8535*

Микроконтроллер *ATmega8535* имеет Гарвардскую архитектуру (программа и данные находятся в разных адресных пространствах) и систему команд, близкую к идеологии *RISC*. Процессор имеет 32 8-битных регистров общего назначения (РОН) (*r0 – r31*), объединённых в регистровый файл. В отличие от «идеального» *RISC*, регистры не абсолютно ортогональны [2]:

– три «сдвоенных» 16-битных регистра-указателя ***X (r26:r27), Y (r28:r29) и Z (r30:r31)***;

– некоторые команды работают только с регистрами ***r16…r31***;

– результат умножения (в тех моделях, в которых есть модуль умножения) всегда помещается в ***r0:r1***.

Структура процессора представляется как «высокопроизводительная *RISC*-архитектура с пониженным энергопотреблением» Гарвардского типа. Одним из основных достоинств этого контроллера является быстрое выполнение команд – он выполняет команду за один такт. *AVR* имеет, вероятно, наиболее разносторонний по своим возможностям процессор из всех микроконтроллеров. Это означает, что при разработке приложений надо потратить немного больше времени на планирование размещения данных в памяти и регистрах, чем для других микроконтроллеров. Но благодаря своей разносторонности *AVR* очень прост в программировании как для разработчиков прикладных программ на языке ассемблера, так и для тех, кто пишет компиляторы языков высокого уровня.

Как и для любого процессора, его особенности являются следствием общих принципов их разработки. Организация набора регистров микропроцессоров *AVR* представлена в графическом виде на Рисунке 1. Такая организация обеспечивает высокую эффективность процессора при обработке данных.

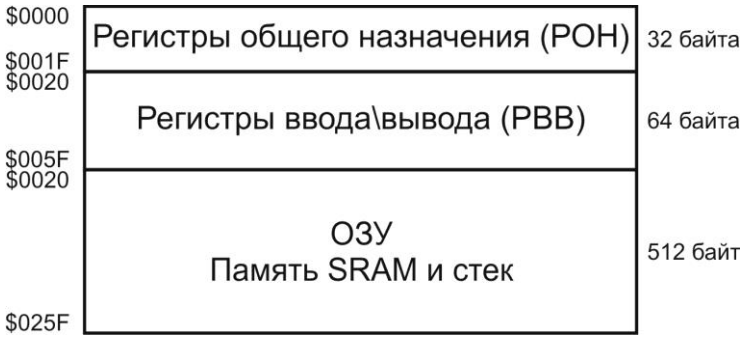


Рисунок 1 – Приоритеты регистров в архитектуре процессоров *AVR*

2. Система команд микроконтроллера

Система команд микроконтроллера *ATmega8535* весьма развита и насчитывает 130 различных инструкций. Большинство команд занимает только 1 ячейку памяти (16 бит) и выполняется за 1 такт [3].

Всё множество команд микроконтроллера можно разбить на несколько групп [4]:

– команды логических операций;
– команды арифметических операций и команды сдвига;
– команды операций с битами;
– команды пересылки данных;
– команды передачи управления;
– команды управления системой.

Управление периферийными устройствами осуществляется через адресное пространство данных. Для удобства существуют «сокращённые команды» *IN/OUT*.

В системе команд микроконтроллера *ATmega8535* используется обозначения, приведенные в Таблице 1.

Таблица 1 – Обозначения, используемые в системе команд

Обозначение	**Описание**
Регистры и операнды	
Rd	Регистр назначения (и источник) в регистровом файле
Rr	Регистр-источник в регистровом файле
R	Результат выполнения команды
K	Литерал или байт данных (8 бит)
k	Данные адреса константы для счётчика программ
b	Бит в регистровом файле или *I/O* регистр (3 бита)
s	Бит в регистре статуса (3 бита)
X, Y, Z	Регистр косвенной адресации (*X=R27:R26, Y=R29:R28, Z=R31:R30*)

P	Адрес *I/O* порта
q	Смещение при прямой адресации (6 бит)
Стек	
STACK:	Стек для адреса возврата и опущенных в стек регистров
SP:	Указатель стека
Регистр статуса (SREG)	
SREG:	Регистр статуса
C:	Флаг переноса
Z:	Флаг нулевого значения
N:	Флаг отрицательного значения
V:	Флаг-указатель переполнения дополнения до двух
S:	Флаг знака
H:	Флаг полупереноса
T:	Флаг пересылки, используемый командами *BLD* и *BST*
I:	Флаг разрешения / запрещения глобального прерывания
I/O регистры	
RAMPX, RAMPY, RAMPZ:	Регистры связанные с *X, Y* и *Z* регистрами, обеспечивающие косвенную адресацию всей области СОЗУ микроконтроллера с объёмом СОЗУ более 64 Кбайт

Полный список и описание всех арифметических, логических команд, команд пересылки данных и команды переходов приведены в Приложении 1. Подробно рассмотрим некоторые команды.

2.1. Команды пересылки данных

Команда *MOV*
Синтаксис: *MOV Rd, Rr*
0 < d < 31,
0 < r < 31.

Команда копирует содержимое регистра *Rr* в регистр *Rd*. Исходный регистр *Rr* остаётся неизменным, в регистр назначения *Rd* загружается копия содержимого регистра *Rr*.

Пример:
MOV r1, r2 ; скопировать содержимое *r2* в *r1*

Команда *LDI*
Синтаксис: *LDI Rd, k*
16 < *d* < 31,
0 < *k* < 255

Команда *LDI* загружает 8-разрядную константу в регистр *Rd*.

Пример:
LDI r30, $12 ; загрузить константу *$12* в *r30*

Команда *LD*
Синтаксис: *LD Rd, X*
LD Rd, X+
LDD Rd, -X
0 < *d* < 31

Команда *LD* загружает косвенно один байт из СОЗУ в регистр. Положение байта в СОЗУ указывается 16-разрядным регистром-указателем *X* в регистровом файле. Обращение к памяти ограничено текущей страницей объемом 64 Кбайта.

Регистр-указатель *X* может остаться неизменным после выполнения команды, но может быть инкрементирован *X+* или декрементирован *X-*.

Пример:

```
CLR r27          ; очистить старший байт X
LDI r26, $20     ; установить константу $20 в младший
                 ; байт X
LD r0, X+        ; загрузить в r0 содержимое SRAM по
                 ; адресу $20 (X постинкрементируется –
                 ; увеличивается на 1)
LD r1, X         ; загрузить в r1 содержимое SRAM по
                 ; адресу $21
LDI r26, $23     ; установить $23 в младший байт X
LD r2, X         ; загрузить в r2 содержимое SRAM по
                 ; адресу $23
LD r3, -X        ; загрузить в r3 содержимое SRAM по
                 ; адресу $22 (X преддекрементируется –
                 ; уменьшается на 1)
```

Команда *LDS*

Синтаксис: *LDS Rd, k*

$0 < d < 31$,

$0 < k < 65535$

Команда *LDS* выполняет загрузку одного байта из СОЗУ в регистр. Можно использовать 16-разрядный адрес. Обращение к памяти ограничено текущей страницей СОЗУ объёмом 64 Кбайта. Команда *LDS* использует для обращения к памяти выше 64 Кбайт регистр *RAMPZ.*

Пример:

```
LDS  r2, $FF00 ; загрузить r2 содержимым
               ; SRAM по адресу $FF00
```

Команда *STS*

Синтаксис: *STS k, Rr*

$0 < r < 31$,

$0 < k < 65535$

Команда *STS* осуществляет запись одного байта из регистра в СОЗУ. Можно использовать 16-разрядный адрес. Обращение к памяти ограничено текущей страницей СОЗУ объёмом 64 Кбайта. Команда *STS* использует для обращения к памяти выше 64 Кбайт регистр *RAMPZ.*

Пример:

```
LDS r2, $1203 ; загрузить в r2 содержимое
              ; SRAM по адресу $1203
ADD r2, r1    ; сложить r1 с r2
STS  $1203, r2 ; записать обратно
```

Команда *ST*

Синтаксис: *ST X, Rr*

ST X+, Rr

ST -X, Rr

$0 < d < 31$

Команда осуществляет косвенную запись одного байта из регистра в СОЗУ. Положение байта в СОЗУ указывается 16-разрядным регистром-указателем *X* в регистровом файле. Обращение к памяти ограничено текущей страницей объёмом 64 Кбайта. Для обращения к другой странице СОЗУ необходимо изменить регистр *RAMPX* в *I/O* области. Регистр-указатель *X* может остаться неизменным после выполнения команды, но может быть инкрементирован или декрементирован. Эта особенность удобна при использовании регистра-указателя *X* в качестве указателя стека.

Пример:

```
CLR   r27       ; очистить старший байт X
LDI   r26, $34  ; установить $34 в младший байт X
ST    X+, r0    ; сохранить в r0 содержимое SRAM
                ; по адресу $34 (X
                ; постинкрементируется)
ST    X, r1     ; сохранить в r1 содержимое
                ; SRAM по адресу $35
LDI   r26, $34  ; установить $34 в младший байт X
ST    r2, X     ; сохранить в r2 содержимое SRAM
                ; по адресу $34
SR    r3, -X    ; сохранить в r3 содержимое SRAM
                ; по адресу $33
                ; (X преддекрементируется)
```

Команда *PUSH*

Синтаксис: *PUSH Rr*

$0 < r < 31$

Команда заносит содержимое регистра *Rd* в стек.

Команда *POP*

Синтаксис: *POP Rr*

$0 < r < 31$

Команда загружает из стека байт в регистр *Rd*.

Пример:

```
PUSH r16    ; сохранить r16 в стеке
PUSH r17    ; сохранить r17 в стеке
POP  r17    ; восстановить r17
POP  r16    ; восстановить r16
```

2.2. Арифметические команды

Сложение без переноса *ADD* (*ADD – Add without Carry*)
Синтаксис: *ADD Rd, Rr*
$0 < d < 31$,
$0 < r < 31$.

Данная команда реализует сложение двух регистров без добавления содержимого флага переноса (*C*) и размещает результат в регистре назначения *Rd*. Устанавливает флаги *Z, C, N, V*.

Пример:

```
ADD r1, r2     ; сложить r2 с r1 (r1=r1+r2)
ADD r28, r28 ; сложить r28 с самим собой
```

Сложение с переносом (*ADC*)
Синтаксис: *ADC Rd, Rr*
$0 < d < 31$,
$0 < r < 31$

Данная команда осуществляет сложение двух регистров и содержимого флага переноса (*C*), размещает результат в регистре назначения *Rd*: *Rd <- Rd + Rr + C*. Устанавливает флаги *Z, C, N, V, H*.

Пример:

```
                ; сложить R1 : R0 с R3 : R2
ADC r2, r0    ; сложить младший байт
ADC r3, r1    ; сложить старший байт с переносом
```

Команда *ADIW (Add Immediate to Word*)
Синтаксис: *ADIW Rdl, k*
dl = {24, 26, 28, 30},
0 < k < 64

Данная команда осуществляет сложение непосредственного значения (0-63) с парой регистров и размещение результата в паре регистров: *Rdh:Rdl <- Rdh:Rdl + k*. Команда работает только с четырьмя верхними парами регистров, удобна для работы с регистрами-указателями. Устанавливает флаги *Z, C, N, V*.

Пример:

```
ADIW r24, $1   ; сложить $1 с r25:r24
ADIW r30, $63 ; сложить $63 с Z указателем (r31 : r30)
```

Команда *SUB*

Синтаксис: *SUB Rd, Rr*

$16 < d < 31$,

$0 < r < 31$

Команда осуществляет вычитание содержимого регистра-источника *Rr* из содержимого регистра-приёмника *Rd*, результат помещает в регистре назначения: *Rd = Rd – Rr*. Устанавливает флаги *H, V, N, Z, C*.

Пример:

```
SUB r13, r12 ; вычесть r12 из r13
```

Команда *SUBI*

Синтаксис: *SUB Rd, k*

$16 < d < 31$,

$0 < k < 255$

Команда *SUB* производит вычитание константы из содержимого регистра, размещает результат в регистре назначения. Устанавливает флаги *H, V, N, Z, C*.

Пример:

SUBI r22, $11 ; вычесть *$11* из *r22*

Команда *SBC*

Синтаксис: *SBC Rd, Rr*

$0 < d < 31$,

$0 < r < 31$

Команда *SBC* осуществляет вычитание содержимого регистра-источника и содержимого флага переноса (*C*) из регистра *Rd*, результат помещает в регистре назначения *Rd*. Устанавливает флаги *H, V, N, Z, C*.

Пример:

```
                    ; вычесть r1 : r0 из r3 : r2
SUB   r2, r0  ; вычесть младший байт
SBC   r3, r1  ; вычесть старший байт с переносом
```

Команда *SBCI*

Синтаксис: *SBCI Rd, k*

$0 < d < 31$,

$0 < k < 255$

Команда осуществляет вычитание константы и содержимого флага переноса (*C*) из содержимого регистра, размещает результат в регистре назначения *Rd*. Устанавливает флаги *H, V, N, Z, C* регистра флагов.

Пример:

```
                    ; вычесть $2567 из r14 : r15
SUBI  r14, $67  ; вычесть младший байт
SBCI  r15, $25  ; вычесть старший байт с переносом
```

Команда *SBIW*

Синтаксис: *SBIW Rdl, k*

dl = {24, 26, 28, 30},
$0 < k < 63$

Команда осуществляет вычитание непосредственного значения (0-63) из пары регистров и размещает результат в паре регистров. Команда работает с четырьмя верхними парами регистров, удобна для работы с регистрами указателями. Устанавливает флаги *H, V, N, Z, C* регистра флагов.

Пример:

```
SBIW  r24, $3        ; вычесть $3 из r25 : r24
SBIW  r28, $A2       ; вычесть $A2 из Y указателя
                     ; (r29 : r28)
```

Команда *DEC*

Синтаксис: *DEC Rd*

$0 < d < 31$

Команда осуществляет вычитание единицы («-1») из содержимого регистра *Rd* и размещает результат в регистре назначения *Rd*. Флаг переноса регистра статуса данной командой не активируется, что позволяет использовать команду *DEC* при реализации счётчика циклов для вычислений с повышенной точностью. Устанавливает флаги *V, N, Z* регистра флагов.

Пример:

```
        LDI r16, $09      ; загрузить константу в r16
metka1: ADD r3, r4        ; сложить r4 с r3
        DEC r16           ; уменьшить на 1 r16
        BRNE metka1       ; перейти если r16 <> 0
        NOP               ; продолжать (пустая операция)
```

Команда *INC*

Синтаксис: *INC Rd*

$0 < d < 31$

Команда увеличения на единицу содержимого регистра *Rd* и размещения результата в регистре назначения *Rd*. Устанавливает флаги *V, N, Z* регистра флагов.

Пример:

```
        CLR r23           ; очистить r23
metka2: INC r23           ; увеличить на 1 r23
        . . .
        CPI r23, $45      ; сравнить r23 с $45
        BRNE metka2       ; перейти, если не равно
        NOP               ; продолжать (пустая операция)
```

2.3. Команды умножения

Команда *MUL*

Синтаксис: *MUL Rd, Rr*

$0 < d < 31$,

$0 < r < 31$

Команда перемножает две 8-разрядные величины без знаков с получением 16-разрядного результата без знака. Множимое и множитель – два регистра: *Rr* и *Rd*, соответственно. Произведение размещается в регистрах *R1* (старший байт) и *R0* (младший байт). Следует учесть, что

если в качестве множимого и множителя выбрать *R0* или *R1*, то результат заместит прежние значения сразу после выполнения операции. Если в результате выполнения команды *MUL* установлен в единицу бит 15-ть результата, то флаг переноса (*C*) регистра флагов устанавливается в 1.

Пример:

```
MUL r3, r4      ; перемножить r3 и r4
MOV r3, r1      ; вернуть результат обратно в r3 : r4
MOV r4, r0      ; вернуть результат обратно в r3 : r4
```

Команда *FMUL*

Синтаксис: *FMUL Rd, Rr*

$16 \le d \le 23$,

$16 \le r \le 23$

Эта команда выполняет знаковое умножение 2-х 8-битных величин и сдвиг результата на 1 бит влево. 16-разрядный результат помещается в пару регистров: *R01* и *R00*, причём старший бит результата находится в регистре *R01*, а младший – в регистре *R00*.

Если в результате команды *FMUL* бит 15-ть результата до сдвига установлен, то флаг переноса (*C*) установлен в 1, иначе флаг *C=0*. Флаг *Z* установлен в том случае, если результат равен *$0000*, иначе флаг *Z* – очищен.

Пример:

```
                ; перемножить 2 величины 15d и 33d
LDI r16, 15
LDI r17, 33
FMUL r16, r17
                ; результат: R00 = DE h, R01 = 03 h
```

2.4. Команды сравнения

Команда *TST*
Синтаксис: *TSR Rd*
$0 < d < 31$

Команда осуществляет проверку на ноль или минус. При её выполнении производится логическое умножение («И») регистра источника самого на себя и выставляются соответствующие флаги, но сам результат логического умножения никуда не записывается. Устанавливает флаги *V, N, Z* регистра флагов.

Пример:

```
TST  r2        ; выставить флаги нулевого или
               ; отрицательного значения по
               ; содержимому регистра r2
```

Команда *CP*
Синтаксис: *CP Rd, Rr*
$0 < d < 31$,
$0 < r < 31$.

Данная команда выполняет сравнение содержимого двух регистров *Rd* и *Rr* путём вычитания *Rd-Rr*. Содержимое регистров не изменяется. После выполнения данной команды устанавливаются флаги: *H, Z, C, V, N, S*. Команда *CP* применяется вместе с командами перехода.

Пример:

```
CP r4, r19     ; сравнить r4 с r19
BRNE noteq     ; перейти, если r4 <> r19
```

Команда *CPC*

Синтаксис: *CPC Rd, Rr*

$0 < d < 31$,

$0 < r < 31$

Команда выполняет сравнение содержимого двух регистров *Rd* и *Rr* и учитывает также перенос предшествующей команды. Сравнение выполняется путём вычитания: *Rd-Rr-C*. Содержимое регистров не изменяется, а после выполнения устанавливаются флаги: *H, Z, C, V, N, S*. После этой команды можно выполнять любые условные переходы.

Пример:

```
                ; сравнить r3 : r2 с r1 : r0
    CP r2, r0   ; сравнить старший байт
    CPC r3, r1  ; сравнить младший байт
    BRNE noteq  ; перейти, если не равно
```

Команда *CPI*

Синтаксис: *CPI Rd, k*

$16 \le d \le 31$,

$0 < k < 255$

Команда выполняет сравнение содержимого регистра *Rd* с константой путём вычитания содержимого *Rd-k*. Содержимое регистра не изменяется, устанавливаются флаги: *H, Z, C, V, N, S*. После этой команды можно выполнять любые условные переходы.

Пример:

```
    CPI r12, 15   ; сравнить r12 с 15
    BRNE error    ; перейти, если r12 <> 15
```

2.5. Логические команды

Команда *AND*

Синтаксис: *AND Rd, Rr*

$0 < d < 31$,

$0 < r < 31$

Команда межрегистрового логического «И». Иначе: команда межрегистрового логического умножения, где логическое «да» наступает только в случае единогласного «да», а логическое «нет», соответственно, во всех остальных случаях (см. Таблицу 2).

Таблица 2 – Таблица истинности по логическому умножению

Вх. 1	Вх. 2	Вых.
0, «нет»	0, «нет»	0, «нет»
1, «да»	0, «нет»	0, «нет»
0, «нет»	1, «да»	0, «нет»
1, «да»	1, «да»	1, «да»

Пример:

```
AND  r2, r0     ; логически перемножить содержимое
                ; регистров r2 и r0
AND  r20, r30   ; логически перемножить содержимое
                ; регистров r20 и r30
```

Команда *ANDI*

Синтаксис: *ANDI Rd, k*

$0 < d < 31$,

$0 < k < 255$

Команда осуществляет операцию логического «И» (см. Таблицу 2) для содержимого регистра и непосредственного значения (константы).

Пример:

```
ANDI  r30, $50      ; логически умножить
                    ; содержимое регистра r30 на 80
ANDI  r23, $B5      ; логически умножить
                    ; содержимое регистра r23 на
                    ; 181
```

Команда *OR*

Синтаксис: *OR Rd, Rr*

$0 < d < 31$,

$0 < r < 31$

Команда межрегистрового логического «ИЛИ». Иначе: команда логического сложения, где логическое «нет» наступает только в случае единогласного «нет», в остальных случаях – логическое «да» (см. Таблицу 3).

Таблица 3 – Таблица истинности по логическом сложению

Вх. 1	Вх. 2	Вых.
0, «нет»	0, «нет»	0, «нет»
1, «да»	0, «нет»	1, «да»
0, «нет»	1, «да»	1, «да»
1, «да»	1, «да»	1, «да»

Пример:

```
OR  r4, r8      ; логически сложить содержимое
                ; регистра r4 с содержимым регистра r8
OR  r3, r5      ; логически сложить содержимое
                ; регистра r3 с содержимым регистра r5
```

Команда *ORI*

Синтаксис: *ORI Rd, k*

$0 < d < 31$,

$0 < k < 255$

Команда логического «ИЛИ» (см. Таблицу 3) для содержимого регистра и непосредственного значения (константы).

Пример:

```
ORI  r28, $A0        ; логически сложить содержимое
                     ; регистра r28 с константой,
                     ; равной 160
ORI  r4, $04         ; логически сложить содержимое
                     ; регистра r4 с константой,
                     ; равной 4
```

Команда *EOR*

Синтаксис: *EOR Rd, Rr*

$0 < d < 31$,

$0 < r < 31$

Команда осуществляет операцию исключающего «ИЛИ» над содержимым двух регистров (см. Таблицу 4).

Таблица 4 – Таблица истинности по исключающему ИЛИ

Вх. 1	Вх. 2	Вых.
0, «нет»	0, «нет»	0, «нет»
1, «да»	0, «нет»	1, «да»
0, «нет»	1, «да»	1, «да»
1, «да»	1, «да»	0, «нет»

Пример:

```
EOR  r6, r9     ; выполнить логическую операцию
                ; исключающего ИЛИ над содержимым
                ; регистра r6 и содержимым регистра r9
EOR  r17, r21   ; выполнить логическую операцию
                ; исключающего ИЛИ над содержимым
                ; регистра r17 и содержимым регистра
                ; r21
```

2.6. Команды сдвигов и операций с битами

Команда *CLR (Clear Register)*
Синтаксис: *CLR Rd*
$16 < d < 31$

Команда установки значения логического «нет» (нуля) во все биты регистра приёмника. В регистр приёмник заносится шестнадцатеричная константа *00*. Устанавливает флаги *Z*, *N*, *V* регистра флагов.

Пример:

```
CLR  r16        ; обнулить / очистить содержимое
                ; регистра r16
```

Команда *SER (Set Register)*
Синтаксис: *SER Rd*
$16 < d < 31$

Команда установки значения логического «да» (единицы) во все биты регистра приёмника. Иначе: в регистр приёмник заносится шестнадцатеричная константа *FF*.

Пример:

```
SER  r16        ; установить содержимое регистра r16
                ; равным FF (11111111)
```

Команда *COM*
Синтаксис: *COM Rd*
$0 < d < 31$

Команда получения в регистре приёмнике единичного дополнения, хранящегося в регистре приёмнике, путём вычитания шестнадцатеричного значения регистра приёмника из *FF*: *Rd* ← *SFF* – *Rd*, где *S* – флаг знака. Устанавливает флаги *C, Z, N, V* регистра флагов.

Пример:

```
COM  r16      ; преобразовать содержимое регистра
              ; r16 в единичное дополнение
```

Команда *NEG*
Синтаксис: *NEG Rd*
$0 < d < 31$

Команда изменения знака значения регистра приёмника путём вычитания шестнадцатеричного значения регистра приёмника из нуля. Иначе: получение в регистре приёмнике двоичного дополнения, хранимого в нём ранее значения: *Rd* ← *S00* – *Rd*, где *S* – флаг знака. Устанавливает флаги *C, Z, N, V, H* регистра флагов.

Пример:

```
NEG  r24      ; преобразовать и установить
              ; содержимое регистра r24
              ; отрицательным значением
```

Команда *SBR*
Синтаксис: *SBR Rd, k*
$16 < d < 31$,
$0 < k < 255$

Команда установки поименованных битов в регистре приёмнике (остальные биты команда не трогает). Обозначаются биты шестнадцатеричной константой. Так *06h* (*0000 0110d*) соответствует установке 1-го и 2-го младших битов (нумерация битов начинается с нуля), а *41h* (*0100 0001d*) установке 0-го младшего и 2-го старшего битов. Команда *SBR* формально записывается так: $Rd \leftarrow Rd \vee k$. Устанавливает флаги *Z, N, V* регистра флагов.

Пример:

```
SBR  r19, $06  ; установить в регистре r19 2-й и 3-й
               ; биты
SBR  r17, $41  ; установить в регистре r17 1-й и 7-й
               ; биты
```

Команда *CBR*

Синтаксис: *CBR Rd, k*

$16 < d < 31$,

$0 < k < 255$

Команда очистки поименованных битов в регистре (на остальные биты команда не влияет). Обозначаются биты для сброса шестнадцатеричной константой. Так *03h* (*0000 0011d*) соответствует сбросу 0-го и 1-го бита. Формально записывается так $Rd \leftarrow Rd \bullet (SFF - K)$, где *S* – флаг знака. Устанавливает флаги *Z, N, V* регистра флагов.

Пример:

```
CBR  r18, $03 ; очистить в регистре r18  0-й и 1-й
              ; биты
```

Команда *SEC (Set Carry flag)*
Синтаксис: *SEC*

Команда установки флага переноса (флаг переноса принимает значение логического «да»): *C = 1*.

Команда *CLC (Clear Carry flag)*
Синтаксис: *CLC*

Команда очистки флага переноса (флаг переноса принимает значение логического «нет»): *C = 0*.

Команда *SEN (Set Negative flag)*
Синтаксис: *SEN*

Команда установки флага отрицательного значения (флагу отрицательного значения присваивается логическое «да»): *N = 1*.

Команда *CLN (Clear Negative flag)*
Синтаксис: *CLN*

Команда очистки флага отрицательного значения (флагу отрицательного значения присваивается логическое «нет»): *N = 0*.

Команда *SEZ (Set Zero flag)*
Синтаксис: *SEZ*

Команда установки флага нулевого значения (флагу нулевого значения присваивается логическое «да»): *Z = 1*.

Команда *CLZ (Clear Zero flag)*
Синтаксис: *CLZ*

Команда очистки флага нулевого значения (флагу нулевого значения присваивается логическое «нет»): *Z = 0*.

Команды *SEI, CLI (Clear Interrupt flag), SES (Set Sign flag), CLS (Clear Sign flag), SEV (Set oVerflow flag), CLV (Clear oVerflow flag), SET (Set Temporary flag), CLT (Clear Temporary flag), SEH (Set Half-carry flag), CLH (Clear Half-carry flag)* аналогичны вышерассмотренным командам, устанавливают флаги (=1) или сбрасывают (=0) (см. Таблицу 9 в Приложении 1).

Команда *BSET (Bit Set)*
Синтаксис: *BSET n*
$0 < n < 7$

Команда установки того или иного флага через соответствующий бит регистра *SREG* (статусный регистр).

Пример:

```
BSET 2    ; установить второй бит регистра SREG
          ; (флаг отрицательного значения N)
```

Команда *BCLR (Bit Clear)*
Синтаксис: *BCLR n*
$0 < n < 7$

Команда очистки того или иного флага через соответствующий бит регистра *SREG* (статусный регистр).

Пример:

```
BCLR 4          ; установить четвертый бит регистра
                ; SREG (флаг знака S)
```

Команда *BST*

Синтаксис: *BST Rd, n*

$0 < d < 31$,
$0 < n < 7$

Команда установки в пользовательский флаг значения, равного указанному биту значения, хранящегося в регистре приёмнике.

Пример:

```
BST r3, 3       ; записать во флаг T то же значение, что
                ; и в 3-м бите (0000x000) регистра r3
```

Команда *BLD*

Синтаксис: *BLD Rd, n*

$0 < d < 31$,
$0 < n < 7$

Команда установки флага в указанный бит. Иначе: перенос значения из регистра-приёмника в пользовательский флаг.

Пример:

```
BLD r5, 6       ; записать в шестой бит регистра r5
                ; (0x000000) то же значение, что
                ; хранится во флаге T
```

Команда *SBI*

Синтаксис: *SBI IO, n*

$0 < n < 7$

Команда установки указанного бита в регистре *I/O* (ввода/вывода).

Пример:

SBI pinc, 4

Команда *CBI*

Синтаксис: *CBI IO, n*

$0 < n < 7$

Команда очистки указанного бита в регистре *I/O* (ввода/вывода).

Пример:

CBI pina, 7

Команда *LSL (Logical Shift Left)*

Синтаксис: *LSL Rd*

$0 < d < 31$

Команда осуществляет логический сдвиг влево содержимого казанного регистра *Rd*. Старший бит записывается во флаг переноса, последовательность сдвигается влево и в младший бит записывается ноль (см. Рис. 2). Устанавливает флаги *Z, C, N, V, H* регистра флагов.

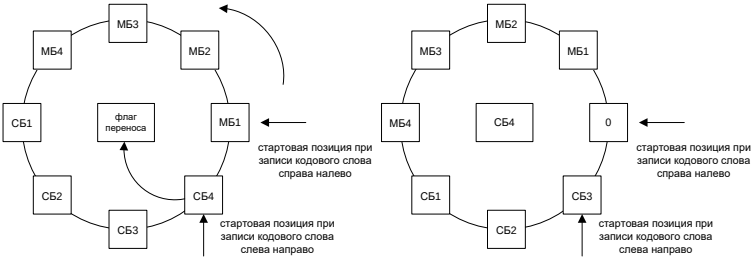


Рисунок 2 – Логический сдвиг влево (*LSL*)

Команда *LSR (Logical Shift Right)*
Синтаксис: *LSR Rd*
$0 < d < 31$

Команда осуществляет логический сдвиг вправо содержимого регистра *Rd*. Младший бит записывается во флаг переноса, последовательность сдвигается вправо и в старший бит записывается ноль (см. Рис. 3). Устанавливает флаги *Z, C, N, V, H* регистра флагов.

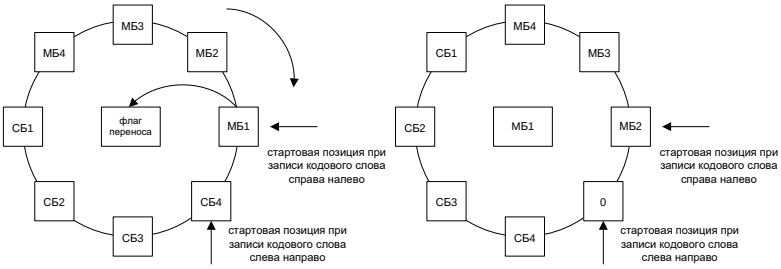


Рисунок 3 – Логический сдвиг вправо (*LSR*)

Команда *ROL (Rotate Left)*
Синтаксис: *ROL Rd*
$0 < d < 31$

Команда осуществляет циклический сдвиг влево содержимого регистра *Rd*. Старший бит записывается во флаг переноса, последовательность сдвигается влево, и во младший бит заносится значение, записанное во флаге переноса (см. Рис. 4). Устанавливает флаги *Z, C, N, V, H* регистра флагов.

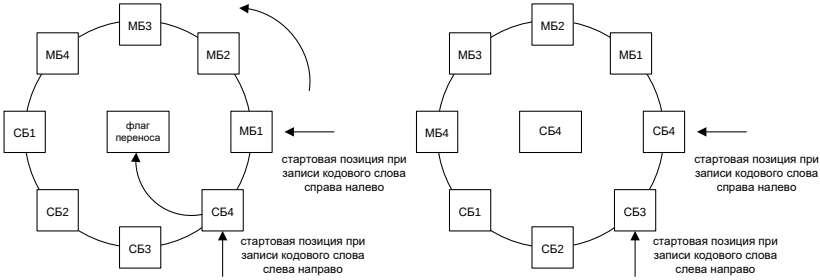


Рисунок 4 – Циклический сдвиг влево (*ROL*)

Команда *ROR (Rotate Right)*
Синтаксис: *ROL Rd*

$0 < d < 31$

Команда осуществляет циклический сдвиг вправо содержимого регистра *Rd*. Младший бит записывается во флаг переноса, последовательность сдвигается вправо, и в старший бит заносится значение, записанное во флаге переноса (см. Рис. 5). Устанавливает флаги *Z, C, N, V* регистра флагов.

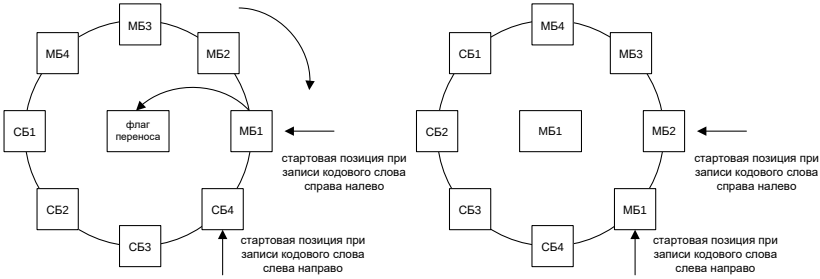


Рисунок 5 – Циклический сдвиг вправо (*ROR*)

Команда *ASR (Arithmetical Shift Right)*
Синтаксис: *ASR Rd*
$0 < d < 31$

Команда выполняет арифметический сдвиг вправо содержимого регистра *Rd* (см. Рис. 6). Устанавливает флаги *Z, C, N, V* регистра флагов.

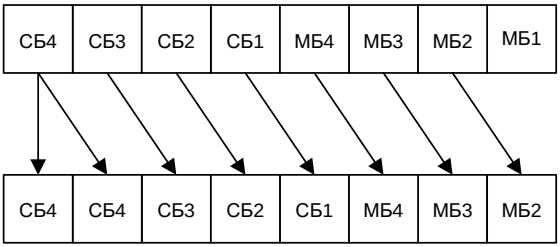


Рисунок 6 – Арифметический сдвиг вправо (*ASR*)

Команда *SWAP*
Синтаксис: *SWAP Rd*
$0 < d < 31$

Команда обмена нибблов (старших и младших разрядов) значения, хранящегося в регистре приёмнике (см. Рис. 7).

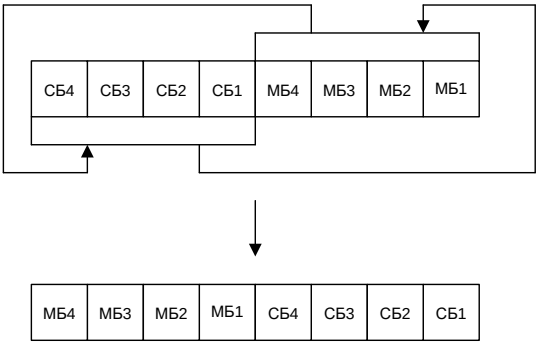


Рисунок 7 – Обмен значений в разрядах (*SWAP*)

2.7. Команды безусловного перехода

Команда *RJMP*
Синтаксис: *RJMP n*
-2048 < *n* < 2048
(-2*k* < *n* < 2*k*)

Команда относительного перехода через *n+1* команду. Переход можно осуществить не более чем на 2048 команд в обе стороны.

Пример:

```
RJMP Init       ; перейти к адресу с меткой Init,
                ; находящемуся в пределах двух
                ; килобайт от текущей позиции.
```

Команда *IJMP*
Синтаксис: *IJMP*

Команда косвенного перехода к адресу, записанному в регистровой паре *Z*. Под регистровой парой *Z* тут нужно понимать 30-й и 31-й регистры.

Команда *JMP*
Синтаксис: *JMP addr*
0 < *addr* < 4 000 000
(0 < *addr* < 4*M*)

Команда прямого перехода к указанному адресу.

Пример:

```
JMP Init        ; перейти к адресу с меткой Init
```

2.8. Команды обращения к процедурам

Команда *RCALL*
Синтаксис: *RCALL n*
-2048 < *n* < 2048
(-2*k* < *n* < 2*k*)

Команда относительного вызова подпрограммы, расположенной через *n+1* команду от текущей позиции. Переход можно осуществить не более чем на 2048 команд в обе стороны.

Пример:

```
RCALL Init      ; вызвать процедуру Init, находящуюся
                ; по адресу, расположенному в пределах
                ; двух ;килобайт от текущей позиции.
```

Команда *ICALL*
Синтаксис: *ICALL*

Команда косвенного вызова процедуры через регистровую пару *Z*. Под регистровой парой *Z* понимается 30-й и 31-й регистры.

Команда *CALL*
Синтаксис: *CALL addr*
0 < *addr* < 4 000 000
(0 < *addr* < 4*M*)

Команда непосредственного вызова процедуры по её стартовому адресу. Позиция вызова заносится в стек. **Важно**:

1. Процедура обязательно должна быть завершена командой *RET*.

2. Стек имеет конечный объём, потому в случае рекуррентных процедур нужно тщательно планировать вычислительный процесс, избегая переполнения стека.

Пример:

```
CALL my_proc        ; вызов процедуры, находящейся
                    ; по адресу с меткой «my_proc».
```

Команда *RET*
Синтаксис: *RET*

Команда возврата из процедуры к позиции, записанной в стек при вызове.

Команда *RETI*
Синтаксис: *RETI*

Команда возврата из глобального прерывания к позиции, записанной в стек при возникновении прерывания. Устанавливает или сбрасывает флаг *I* регистра флагов.

2.9. Команды условного перехода

Команда *CPSE*
Синтаксис: *CPSE Rd, Rr*
$0 < d < 31$,
$0 < r < 31$

Команда межрегистрового сравнения и перехода через последующую команду, в случае равенства содержимого регистра источнике и регистра приёмнике.

Пример:

```
CPSE r28, r30        ; перешагнуть следующую
                     ; команду, если содержимое
                     ; r28 = r30
```

Команда *SBRC*

Синтаксис: *SBRC Rd, n*

$0 < d < 31$,

$0 < n < 7$

Команда перехода через последующую команду, в случае, когда указанный бит в регистре приёмнике сброшен (равен нулю).

Пример:

```
SBRC r0, 2      ; перешагнуть следующую команду в
                ; случае, когда содержимое r0 вида
                ; xxxxx0xx
```

Команда *SBRS*

Синтаксис: *SBRS Rd, n*

$0 < d < 31$,

$0 < n < 7$

Команда перехода через последующую команду, в случае, когда указанный бит в регистре приёмнике установлен (равен единице).

Пример:

```
SBRS r30, 5     ; перешагнуть следующую команду в
                ; случае, когда в регистре r30
                ; установлен 5-й бит.
```

Команда *SBIC*

Синтаксис: *SBIC Rd, n*

$0 < d < 31$,

$0 < n < 7$

Команда перехода через последующую команду, в случае, когда указанный бит в регистре *I/O* сброшен (равен нулю).

Пример:

```
SBIC pina, 5    ; перешагнуть следующую команду в
                ; случае, когда порт A содержит
                ; последовательность вида xx0xxxxx
```

Команда *SBIS*

Синтаксис: *SBIS Rd, n*

$0 < d < 31$,

$0 < n < 7$

Команда перехода через последующую команду, в случае, когда указанный бит в регистре *I/O* установлен (равен единице).

Пример:

```
SBIS pinb, 3    ; перешагнуть следующую команду в
                ; случае, когда порт B содержит
                ; последовательность вида xxxx1xxx
```

Команда *BRBS*

Синтаксис: *BRBS s, k*

$0 < s < 7$,

$-64 < k < 63$

Команда относительного перехода через *k+1* команду, в случае, если указанный флаг статусного регистра установлен

(равен единице). Переходить можно не более чем на 63 команды вперёд или на 64 команды назад, относительно текущей позиции курсора.

Пример:

```
BRBS 3, Exit   ; переход на метку Exit, находящуюся в
               ; пределах 64 команд в случае, когда
               ; установлен флаг переполнения
```

Команда *BRBC*

Синтаксис: *BRBC s, k*

$0 < s < 7$,

$-64 < k < 63$

Команда относительного перехода через *k+1* команду, в случае, если указанный флаг статусного регистра сброшен (равен нулю). Переходить можно не более чем на 63 команды вперёд или на 64 команды назад, относительно текущей позиции курсора.

Пример:

```
BRBC 1, Exit   ; переход на метку Exit, находящуюся в
               ; пределах 64 команд в случае, когда
               ; сброшен флаг нулевого значения
```

Команда *BREQ*

Синтаксис: *BREQ n*

$-64 < n < 63$

Команда относительного перехода через *n+1* команду, если в результате выполнения предыдущей команды флаг нулевого значения установлен в единицу. Переходить можно не более чем на 63 команды вперёд или на 64 команды назад, относительно текущей позиции курсора.

Команда *BRNE*
Синтаксис: *BRNE n*
-64 < *n* < 63

Команда относительного перехода через *n+1* команду, в случае, если после выполнения предыдущей команды флаг нулевого значения сброшен, т.е. равен нулю. Переходить можно не более чем на 63 команды вперёд или на 64 команды назад, относительно текущей позиции курсора.

Команда *BRCS*
Синтаксис: *BRCS n*
-64 < *n* < 63

Команда относительного перехода через *n+1* команду, в случае, если установлен (равен единице) флаг переноса. Переходить можно не более чем на 63 команды вперёд или на 64 команды назад, относительно текущей позиции курсора.

Команда *BRCC*
Синтаксис: *BRCC n*
-64 < *n* < 63

Команда относительного перехода через *n+1* команду, в случае, если сброшен (равен нулю) флаг переноса. Переходить можно не более чем на 63 команды вперёд или на 64 команды назад, относительно текущей позиции курсора.

Команда *BRSH*
Синтаксис: *BRSH n*
-64 < *n* < 63

Команда относительного перехода через *n+1* команду, в случае, если после выполнения предыдущей команды сброшен флаг переноса (равный нулю). Переходить можно не

более чем на 63 команды вперёд или на 64 команды назад, относительно текущей позиции курсора.

Команда *BRLO*
Синтаксис: *BRLO n*
-64 < *n* < 63

Команда относительного перехода через *n+1* команду, в случае, если после выполнения предыдущей команды установлен (равен единице) флаг переноса. Переходить можно не более чем на 63 команды вперёд или на 64 команды назад, относительно текущей позиции курсора.

Команда *BRMI*
Синтаксис: *BRMI n*
-64 < *n* < 63

Команда относительного перехода через *n+1* команду, в случае, если установлен (равен единице) флаг отрицательного значения. Переходить можно не более чем на 63 команды вперёд или на 64 команды назад, относительно текущей позиции курсора.

Команда *BRPL*
Синтаксис: *BRPL n*
-64 < *n* < 63

Команда относительного перехода через *n+1* команду, в случае, если сброшен (равен нулю) флаг переноса. Переходить можно не более чем на 63 команды вперёд или на 64 команды назад, относительно текущей позиции курсора.

Команда *BRGE*
Синтаксис: *BRGE n*
-64 < *n* < 63

Команда относительного перехода через *n+1* команду, в случае, если после выполнения предыдущей команды результат больше или равен нулю. Определяющую роль играет равенство нулю суммы по модулю 2 флагов отрицательного значения и переполнения. Переходить можно не более чем на 63 команды вперёд или на 64 команды назад, относительно текущей позиции курсора.

Команда *BRLT*
Синтаксис: *BRLT n*
-64 < *n* < 63

Команда относительного перехода через *n+1* команду, в случае, если после выполнения предыдущей команды установлен флаг знака (*S*) регистра статуса. Переходить можно не более чем на 63 команды вперёд или на 64 команды назад, относительно текущей позиции курсора.

Команда *BRHS*
Синтаксис: *BRHS n*
-64 < *n* < 63

Команда относительного перехода через *n+1* команду, в случае, если установлен (равен единице) флаг половинного переноса. Переходить можно не более чем на 63 команды вперёд или на 64 команды назад, относительно текущей позиции курсора.

Команда *BRHC*
Синтаксис: *BRHC n*
-64 < *n* < 63

Команда относительного перехода через *n+1* команду, в случае, если сброшен (равен нулю) флаг половинного переноса. Переходить можно не более чем на 63 команды вперёд или на 64 команды назад, относительно текущей позиции курсора.

Команда *BRTS*
Синтаксис: *BRTS n*
-64 < *n* < 63

Команда относительного перехода через *n+1* команду, в случае, если установлен (равен единице) пользовательский флаг. Переходить можно не более чем на 63 команды вперёд или на 64 команды назад, относительно текущей позиции курсора.

Команда *BRTC*
Синтаксис: *BRTC n*
-64 < *n* < 63

Команда относительного перехода через *n+1* команду, в случае, если сброшен (равен нулю) пользовательский флаг. Переходить можно не более чем на 63 команды вперёд или на 64 команды назад, относительно текущей позиции курсора.

Команда *BRVS*
Синтаксис: *BRVS n*
-64 < *n* < 63

Команда относительного перехода через *n+1* команду, в случае, если установлен (равен единице) флаг переполнения.

Переходить можно не более чем на 63 команды вперёд или на 64 команды назад, относительно текущей позиции курсора.

Команда *BRVC*
Синтаксис: *BRVC n*
-64 < *n* < 63

Команда относительного перехода через *n+1* команду, в случае, если сброшен (равен нулю) флаг переполнения. Переходить можно не более чем на 63 команды вперёд или на 64 команды назад, относительно текущей позиции курсора.

Команда *BRIE*
Синтаксис: *BRIE n*
-64 < *n* < 63

Команда относительного перехода через *n+1* команду, в случае, если установлен (равен единице) флаг глобального прерывания, то есть прерывание разрешено. Переходить можно не более чем на 63 команды вперёд или на 64 команды назад, относительно текущей позиции курсора.

Команда *BRID*
Синтаксис: *BRID n*
-64 < *n* < 63

Команда относительного перехода через *n+1* команду, в случае, если сброшен (равен нулю) флаг глобального прерывания, то есть прерывание запрещено. Переходить можно не более чем на 63 команды вперёд или на 64 команды назад, относительно текущей позиции курсора.

3. Создание программ на языке Ассемблер

Исходный файл, написанный на языке Ассемблер, содержит мнемоники, директивы и метки.

Перед каждой строкой программы ставится метка, которая является алфавитно-цифровой строкой, заканчивающейся двоеточием. Метки используются как указания для безусловного перехода и команд условного перехода.

Строкой программы может быть директива, команда, комментарий. Допускается пустая строка в программе.

Комментарий имеет следующую форму:

; [Текст]

Любой текст после символа «;» игнорируется ассемблером и имеет значение только для пользователя.

Операнды можно задавать в различных форматах:

– десятичный (по умолчанию): *10, 255*

– шестнадцатеричный (два способа): *0×0a, $0a*

– двоичный: *0b00001010, 0b11111111*

– восьмеричный (впереди ноль): *010, 077*

Директивы

Директивами называют управляющие указания для программы Ассемблер. Они не транслируются в код, как это делает команда. Они инициализируют участки памяти, определяют константы в памяти, устанавливают счётчик команд на определенный адрес и т.д.

Все директивы Ассемблера приведены в Таблице 5.

Таблица 5 – Директивы Ассемблера

Директива	Описание
BYTE	Зарезервировать байт под переменную
CSEG	Сегмент кодов
DB	Задать постоянным(и) байт(ы) в памяти
DEF	Задать символическое имя регистру
DEVICE	Задать для какого типа микроконтроллера компилировать
DSEG	Сегмент данных
DW	Задать постоянное(ые) слово(а) в памяти
EQU	Установить символ равный выражению
ESEG	Сегмент *EEPROM*
EXIT	Выход из файла
INCLUDE	Включить исходный код из другого файла
LIST	Включить генерацию *.lst*-файла
NOLIST	Выключить генерацию *.lst*-файла
ORG	Начальный адрес программы
SET	Установить символ равный выражению

Синтаксис директив: *.[директива]*

Внимание: перед директивой должна стоять точка. Иначе ассемблер воспринимает это как метку.

Рассмотрим директивы ассемблера:

1. **Директива *CSEG (Code segment)*** указывает на начало сегмента кода. Ассемблируемый файл может иметь несколько кодовых сегментов, которые будут объединены в один при ассемблировании.

Синтаксис: *.CSEG*

Пример:

```
        .DSEG           ; начало сегмента данных
vartab: .BYTE 4         ; резервируется 4 байта в СОЗУ
        .CSEG           ; начало сегмента кода
const:  .DW 2           ; записать 0x0002  в программной
                        ; памяти
        MOV r1, r0      ; перенести из одного регистра в другой
```

2. **Директива *DSEG (Data Segment)*** указывает на начало сегмента данных. Ассемблируемый файл может содержать несколько сегментов данных, которые потом будут собраны в один при ассемблировании. Обычно сегмент данных состоит лишь из директив *BYTE* и меток.

Синтаксис: *.DSEG*

Пример:

```
        .DSEG           ; начало сегмента данных
var1:   .BYTE 1         ; резервировать 1 байт под
                        ; переменную
table:  .BYTE tab_size  ; резервировать tab_size байт
        .CSEG
        LDI r30, low(var1)
        LDI r31, high(var1)
        LD r1, Z
```

3. **Директива *ESEG (EEPROM Segment)*** указывает на начало сегмента *EEPROM* памяти. Ассемблируемый файл может содержать несколько *EEPROM* сегментов, которые будут собраны в один сегмент при ассемблировании. Обычно сегмент *EEPROM* состоит из *DB* и *DW* директив (и меток). Сегмент *EEPROM* памяти имеет свой собственный счётчик.

Синтаксис: *.ESEG*

Пример:

```
        .DSEG           ; начало сегмента данных
var1:   .BYTE 1         ; резервировать 1 байт под переменную
table: .BYTE tab_size   ; зарезервировать tab_size байт
        .ESEG
eevar1: .DW 0xffff      ; записать 1 слово в EEPROM
```

4. **Директива *ORG*** присваивает значения локальным счётчикам. Используется только совместно с директивами *.CSEG*, *.DSEG*, *.ESEG*.

Синтаксис: *.ORG addr*

Пример:

```
        .DSEG           ; начало сегмента данных
        .ORG 0x37       ; установить адрес СОЗУ на 37h
variable: .BYTE 1       ; зарезервировать байт СОЗУ по
                        ; адресу 37h
        .CSEG
        .ORG 0x10       ; установить счётчик команд на
                        ; адрес 10h
        MOV r0, r1      ; перенести из одного регистра в другой
```

5. **Директива *DB*** резервирует ресурсы в программной памяти или в *EEPROM*. Директиве должна предшествовать метка. *DB* задает список выражений и должна содержать по крайней мере одно выражение. Размещать директиву следует в сегменте кодов или в *EEPROM* сегменте.

Список выражений представляет собой последовательность выражений, разделённых запятыми.

Каждое выражение должно быть величиной между -128 и 255.

Если директива *DB* указывается в сегменте кодов, а список выражений содержит более двух величин, то выражения будут записаны так, что 2 байта разместятся в каждом слове *Flash*-памяти.

Синтаксис: *LABEL: .DB* список выражений

Пример:

```
        .CSEG
consts: .DB 0, 255, 0b01010101, -128, 0xaa
        .ESEG
const2: .DB 1,2,3
```

6. **Директива *DW*** резервирует ресурсы в программной памяти или в *EEPROM*. Директиве должна предшествовать метка. Директива *DW* задает список выражений и должна содержать по крайней мере одно выражение. Размещать директиву следует в сегменте кодов или в *EEPROM* сегменте.

Список выражений представляет собой последовательность выражений, разделенных запятыми. Каждое выражение должно быть величиной между -32768 и 65535.

Синтаксис: *LABEL: .DW* список выражений

Пример:

```
        .CSEG
varlist:  .DW 0, 0xffff, 0b1001110001010101, -32768, 65535
        .ESEG
eevarlst: .DW 0, 0xffff, 10
```

7. **Директива *DEF*** позволяет присвоить символическое имя регистру. Регистр может иметь несколько символических имён.

Синтаксис: *.DEF* Имя=Регистр

Пример:

```
        .DEF temp=R16
        .DEF ior=R0
        .CSEG
        LDI temp, 0xf0          ; загрузить 0xf0 в регистр temp
        IN ior, 0x3f            ; прочитать SREG в регистр ior
        EOR temp,
```

8. **Директива *EQU*** присваивает значение метке, которая может быть использована в других выражениях. Значение этой метки нельзя изменить или переопределить.

Синтаксис: *.EQU* метка=выражение

Пример:

```
        .EQU io_offset = 0x23
        .EQU porta = io_offset + 2
        .CSEG                   ; начало сегмента кодов
        CLR r2                  ; очистить регистр r2
        OUT porta, r2           ; записать в порт A
```

9. **Директива *INCLUDE*** говорит Ассемблеру начать читать из другого файла. Ассемблер будет включать файл, указанные после директивы, до конца или до директивы *EXIT*. Включаемый файл может сам включать директивы *INCLUDE*.

Синтаксис: *.INCLUDE* имя файла

Пример:

```
.EQU sreg = 0x3f       ; регистр статуса
.EQU sphigh = 0x3e     ; старший байт указателя стека
.EQU splow  = 0x3d     ; младший байт указателя стека
.INCLUDE iodefs.asm ; включить файл «iodefs.asm»
IN r0, sreg            ; прочитать регистр статуса
```

10. **Директива *EXIT*** позволяет Ассемблеру остановить ассемблирование текущего файла. Обычно Ассемблер работает до конца файла. Если он встретит директиву *EXIT*, то продолжит ассемблировать со строки, следующей за директивой *INCLUDE*.

Синтаксис: *.EXIT*

Пример:

```
.EXIT                  ; выйти из этого файла
```

11. **Директива *DEVICE*** предназначена для того, чтобы сообщить Ассемблеру для какого типа устройства пишется программа. Если Ассемблер встретит команду, которая не поддерживается указанным типом микроконтроллера, то будет выдано сообщение об ошибке. Так же сообщение появится в случае, если размер программы превысит объём имеющейся на этом устройстве памяти.

Синтаксис: *.DEVICE AT90S1200 |AT90S2313 | AT90S2323 | AT90S2333 | AT90S2343 | AT90S4414 | AT90S4433 | AT90S4434 | AT90S8515 | AT90S8534 | AT90S8535 | ATtiny11 | ATtiny12 | ATtiny22 | ATmega603 | ATmega103| Atmega8535*

Пример:

```
.DEVICE ATmega8535    ; использовать ATmega8535
.CSEG
.ORG 0000
JMP label1            ; при ассемблировании
                      ; появится сообщение,
                      ; что ATmega8535 не
                      ; поддерживает команду JMP
```

4. Порядок выполнения лабораторных работ

Средой для изучения системы команд микроконтроллера *Atmega8535* была выбрана *AVR Studio* – интегрированное отладочное средство для микроконтроллеров фирмы *Atmel* семейства *AVR*, включающее в себя компилятор с языка Ассемблер. *AVR Studio* позволяет пользователю полностью контролировать выполнение программ с использованием симулятора, который поддерживает все типы микроконтроллеров *AVR*.

Для создания программы:

1. Запустите *AVR Studio*, выбрав пункт меню *Пуск\Atmel Avr Tools\AVR Studio*. При запуске появляется окно, предлагающее ввести новый проект или открыть существующий (см. Рисунок 8).

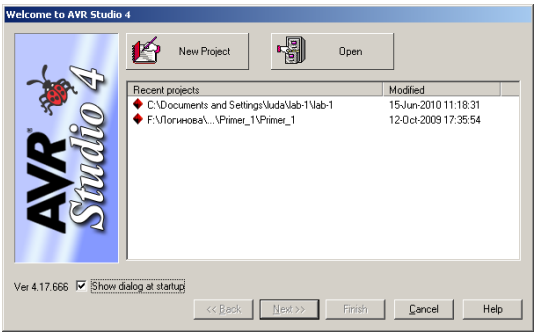
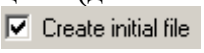


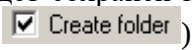
Рисунок 8 – Окно создания / открытия проекта

2. Нажмите на кнопку *New Project*. В появившемся окне (см. Рисунок 9) необходимо указать:

– тип проекта *Atmel AVR Assembler*,
– название программы в строке *«Project Name»*, например, *«Lab_№1»*,

– название файла инициализации (для этого необходимо установить галочку 

и ввести имя файла инициализации в строке *Initial file*),

– создать директорию, где будет сохранён проект (для этого установить галочку 

).

– путь для расположения проекта, для чего необходимо нажать на кнопку 

в строке *Location* и в появившемся окне выбрать необходимую директорию.

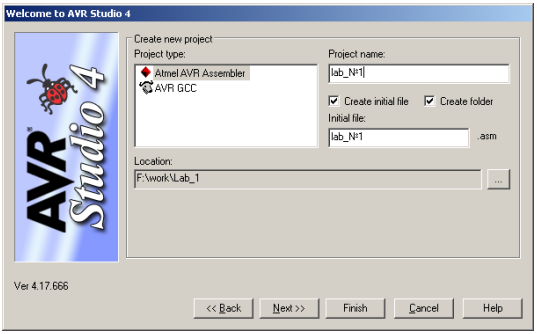


Рисунок 9 – Создание имени проекта

3. Нажмите на кнопку 

и в появившемся окне укажите:

– отладочную платформу – *Debug platform: AVR Simulator*,

– устройство – *Device: ATmega8535*.

4. Нажмите кнопку 

, после чего будет создан проект и файл инициализации.

После этого появится окно отладки программы (Рисунок 10), содержащее три рабочих области:

– окно «*I/O View*» для просмотра регистров ввода/вывода;
– окно набора программы на Ассемблере;
– окно сообщений *«Message»*, комментирующих выполняемые действия.

Рассмотрим более подробно окно регистров ввода-вывода.

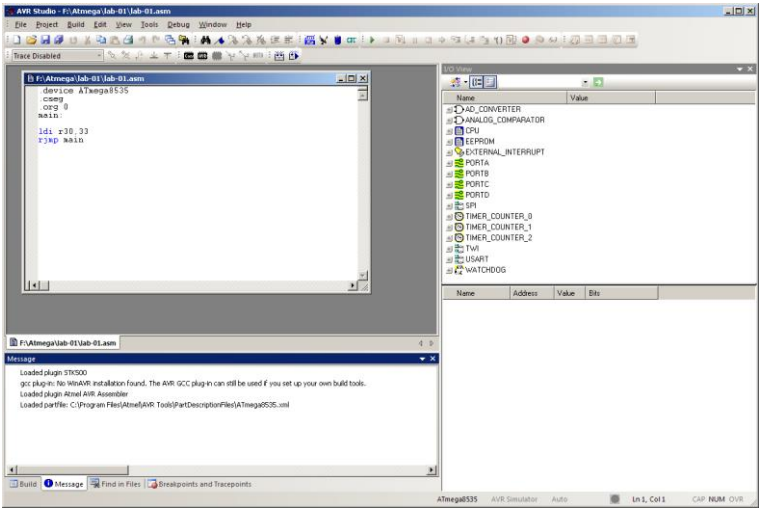


Рисунок 10 – Окно отладки программы

Окно регистров ввода / вывода

Окно регистров содержит дерево всех устройств. Напротив каждого устройства стоит знак «+». Для того, чтобы увидеть все его регистры, необходимо раскрыть устройство, нажимая на ⊞ напротив устройства. Так, например, регистр ввода / вывода порта А содержит три регистра: регистр данных *PORTA*, регистр направления *DDRA* и выводы порта *PINA* (Рисунок 11). Справа от обозначения порта выведено его текущее состояние в виде

шестнадцатеричного числа и битовое изображение. Мышкой можно задавать значения битов "0" или "1". Этим эмулируется воздействие внешних сигналов.

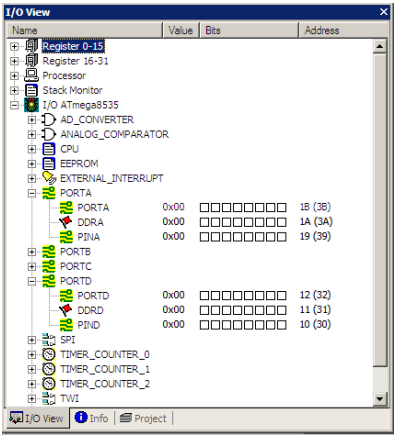


Рисунок 11 – Окно регистров ввода / вывода

Окно просмотра содержимого памяти

Для просмотра содержимого оперативной памяти необходимо в меню «*View*» выбрать пункт «*Memory*» (Рисунок 12).

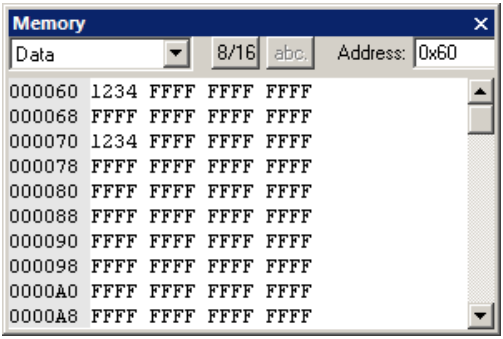


Рисунок 12 – Окно просмотра содержимого памяти

Просматривать можно оперативную память (512 байт), энергонезависимую память данных *EEPROM* (512 байт), регистровую память, память регистров ввода-вывода (*I/O*). Для этого необходимо выбрать соответствующее значение (Рисунок 13).

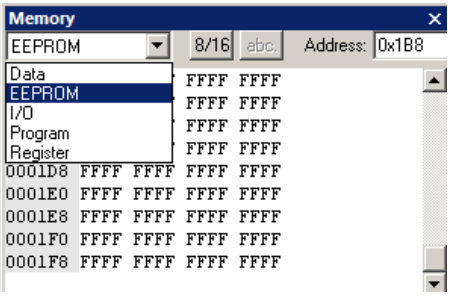


Рисунок 13 – Выбор памяти для просмотра

Можно переходить на требуемый адрес либо используя боковой *Scrollbar*, либо задавая в поле *Address* непосредственное значение адреса в 16-й системе счисления.

Запуск эмулятора

Программа *"AVR Studio"* позволяет запустить программу в реальном времени и в пошаговом режиме. В меню отладки *«Debug»* находятся все варианты запуска программы, например:

– *Run*, запуск программы в реальном времени, результат будет виден только после остановки программы;

– *Break*, останов программы, после просмотра исполнение программы можно продолжить;

– *Reset*, сброс программы установка счётчика команд на начало программы;

 – *Step Over*, пошаговое исполнение, при этом программа останавливается после каждой команды, стрелка указывает на текущую команду;

 – *AutoStep*, запуск программы на непрерывное исполнение с возможностью просмотра текущих шагов исполнения.

В пошаговом режиме можно наблюдать за состоянием регистров после исполнения каждой команды, проверяя правильность операций. Запустив программу кнопкой  *"AutoStep"*, получим её непрерывное исполнение и индикацию регистров во времени. Жёлтая стрелка в окне дизассемблера показывает текущую исполняемую команду.

Приостановив исполнение программы кнопкой  *"Break"*, можно изменить значения регистров ввода / вывода, то есть задать разные значения входных сигналов. И далее снова запустив программу кнопкой *"AutoStep"*, просмотреть реакцию микроконтроллера на эти воздействия (Рисунок 14).

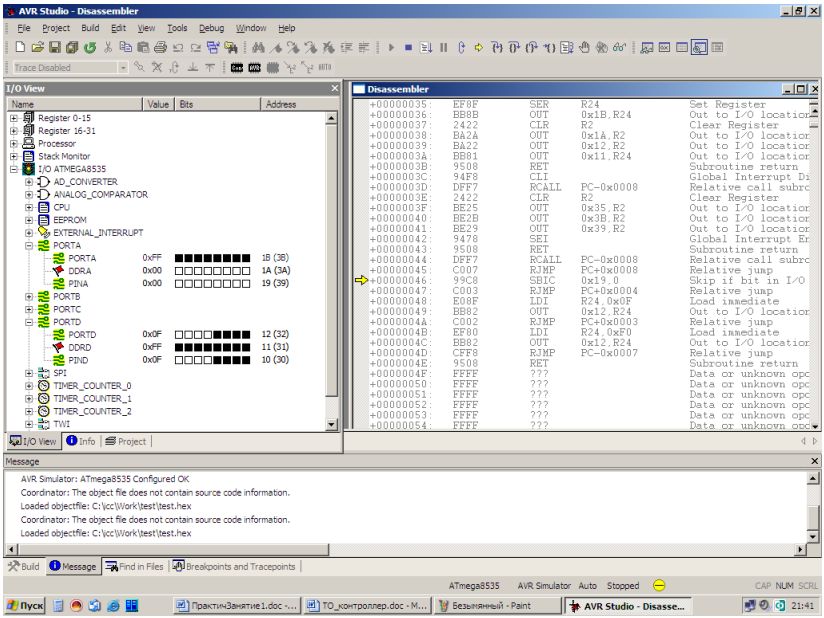


Рисунок 14 – Запуск эмулятора на исполнение

После того, как программа запущена на пошаговое выполнение, для перехода к следующему шагу необходимо нажать клавишу *F11*.

Программа, написанная на Ассемблере, имеет определенную структуру.

Для *ATmega8535* структура программы имеет следующий вид:

```
                                ; название программы,
                                необходимые пояснения
.INCLUDE «m8535def.inc»         ; файл описания ATmega8535
. INCLUDE                       ; включение дополнительных
«имя_файла1.расширение»         файлов

.EQU имя1 = xxx                 ; объявление глобальных констант
.EQU имя2 = nnnn
```

.DEF имя1= регистр	; объявление глобальных
.DEF имя2= регистр	регистровых переменных
.DSEG	; указание сегмента данных
.ORD xxxx	; адрес первого зарезервированного байта
label1: .BYTE 1	; резервировать 1 байт под переменную *label1*
label2: .BYTE m	; резервировать *m* байт под переменную *label2*
.ESEG	; сегмент *EEPROM*
.ORG xxxx	; адрес первого зарезервированного байта
.DB выражение1, выражение2,…	; записать список байтов в *EEPROM*
.DW выражение1, выражение2,…	; записать список слов в *EEPROM*.
.CSEG	; сегмент кодов
.ORG $0000	; адрес начала программы в программной памяти
	; указание векторов прерываний (если они используются)
RJMP reset	; прерывание по сбросу
.ORG $0002	
RJMP INT0	; обработчик прерывания *INT0*
.org $0004	
RJMP INT1	; обработчик прерывания *INT1*
.ORG adrINTx	; адрес следующего обработчика прерываний
RJMP INTx	; обработчик прерывания *x*
…….	; далее располагаются обработчики остальных прерываний
main:	; начало основной программы
<команды>	: тело основной программы

… …

```
subr1: <команда>          ; переход к подпрограмме 1

……    …….
                ……

RET                       ; возврат из подпрограммы

INT0: <команда>           ; программы обработчиков
                                прерываний

……    …….        ……

RETI                      ; возврат из программы
                          обработчиков прерываний
                          ; конец программы не обозначается
```

Лабораторная работа №1
«Регистры, данные и команды пересылки данных»

Цель работы

Изучение регистров общего назначения (РОН) и команд пересылки данных.

Постановка задачи

1. Занести число из столбца «Число 1» Таблицы 6 и строки, соответствующей заданному варианту, в регистр *r24*.
2. Занести число из столбца «Число 2» Таблицы 6 и строки, соответствующей заданному варианту, в регистр *r25*.
3. Обменять числа, хранящиеся в регистрах *r24* и *r25*, между собой 3-мя разными способами.
4. Занести число из столбца «Число 3» Таблицы 6 и строки, соответствующей заданному варианту, в регистровую пару *X(r26:r27)*.
5. Занести число из столбца «Число 4» Таблицы 6 и строки, соответствующей заданному варианту, в регистровую пару *Y(r28:29*).
6. Обменять числа, хранящиеся в регистровых парах *X* и *Y*, между собой 3-мя разными способами.
7. Занести содержимое регистровой пары *X* в оперативную память, при этом младшую часть регистровой памяти занести по адресу *$0070*, старшую – по адресу *$0071*.
8. Занести содержимое регистровой пары *Y* в оперативную память, при этом младшую часть регистровой памяти занести по адресу *$0080*, старшую – по адресу *$0081*.
9. Обменять между собой содержимое четырёх ячеек оперативной памяти: содержимое ячейки памяти с адресом *$0070* поменять с содержимым ячейки *$0080*, а содержимое ячейки памяти с адресом *$0071* поменять с содержимым ячейки памяти с адресом *$0081*.

Таблица 6 – Данные для выполнения Лабораторной работы 1

Номер варианта	**Число 1**	**Число 2**	**Число 3**	**Число 4**
1	38	45	*4F5A*	*8765*
2	56	48	*6BCF*	*7654*
3	54	42	*0123*	*6543*
4	48	54	*1234*	*5432*
5	89	52	*2345*	*4321*
6	58	59	*3456*	*3210*
7	55	12	*4567*	*210F*
8	98	25	*5678*	*10FE*
9	48	34	*6789*	*0FED*
10	54	48	*789A*	*2468*
11	75	56	*89AB*	*468A*
12	25	67	*9ABC*	*68AC*
13	26	62	*ABCD*	*8ACE*
14	53	78	*BCDE*	*ACE8*
15	24	83	*CDEF*	*CE86*
16	15	94	*DEF0*	*E864*
17	81	71	*EF01*	*8642*
18	11	54	*F012*	*1357*
19	15	82	*FEDC*	*3579*
20	45	8	*EDCB*	*579B*
21	52	3	*DCBA*	*79BD*
22	57	54	*CBA9*	*9BDE*
23	64	58	*BA98*	*BDE1*
24	54	64	*A987*	*DE13*
25	68	67	*9876*	*E135*

Содержание отчёта

Отчёт по лабораторной работе должен содержать:

1. Титульный лист, включающий номер и название лабораторной работы.

2. Цель лабораторной работы.
3. Задание на лабораторную работу.
4. Текст программы на языке ассемблера с поясняющими комментариями.
5. Снимки экрана, поясняющие работу программы.
6. Снимки экрана, отображающие результаты работы программы.
7. Вывод.

Лабораторная работа №2
«Арифметические команды»

Цель работы

Изучение арифметических команд и получение навыка работы с ними.

Постановка задачи

Занести числа в память:

a = N+4;
*b = (N+2)*3;*
c = N;
d = N-1;
*e = N*4+N;*

Для вариантов 1 – 10:
$f = N^2$;
Для вариантов после 10:
$f = N \cdot 2$.

N – номер варианта по списку группы или выданный преподавателем.

Произвести операции над числами *a, b, c, d, e, f* по вариантам, заданным в Таблице 7.

Таблица 7 – Данные для выполнения Лабораторной работы 2

Номер вар.	**Выражение**	**Номер вар.**	**Выражение**
1	$a \cdot b + c \cdot d$	15	$(b - a + c - 1) \cdot (d - 10)$
2	$a + d \cdot b - c$	16	$f \cdot (e - b - a)$
3	$a - c \cdot d$	17	$(a - c) \cdot (b - d) - 1)$

4	$(-1)\cdot c + a\cdot d$	18	$(c-d)\cdot(e-b)+a$
5	$(a-c)\cdot d$	19	$d+(-1)\cdot a + c\cdot 4$
6	$c\cdot d+(-1)\cdot a$	20	$(f+d)+(c+a)\cdot(-1)$
7	$(b+c)(a-d)$	21	$(e+(-1)\cdot a)\cdot(c-d)$
8	$(a+(-1)\cdot d)\cdot c$	22	$(e-f)\cdot(c-d)-1$
9	$(c+d)\cdot a-1$	23	$(b+(-1)\cdot f)\cdot(c-d)$
10	$(a+d+(-1)\cdot c)+b$	24	$(e+d\cdot 3)+(-1)\cdot f$
11	$(c\cdot d-1)+f$	25	$(a-d)\cdot(b-f-c)$
12	$f+c\cdot a$	26	$e+(-1)\cdot b+a$
13	$(f-15-b)+(-1)\cdot c$	27	$(a-(-1)\cdot d)+f$
14	$(e-c+a)\cdot 5$	28	$(d+c\cdot 2)-(f-a)$

Содержание отчёта

Отчёт по лабораторной работе должен содержать:

1. Титульный лист, включающий номер и название лабораторной работы.
2. Цель лабораторной работы.
3. Задание на лабораторную работу.
4. Текст программы на языке ассемблера с поясняющими комментариями.
5. Снимки экрана, поясняющие работу программы.
6. Снимки экрана, отображающие результаты работы программы.
7. Вывод.

Лабораторная работа №3
«Логические команды и команды манипулирования битами»

Цель работы

Изучение логических команд и команд манипулирования битами и получение навыка работы с ними.

Постановка задачи

Занести числа в память:

a = N+100;
*b = N*13;*
c = N+60;
d = N+125.

N – номер варианта по списку группы или выданный преподавателем.

Произвести следующие операции над числами *a, b, c, d*:

1. Посчитать количество ненулевых бит в числе.
2. Посчитать количество нулевых бит в числе.
3. Посчитать количество чётных единиц в байте.
4. Посчитать количество нечётных единиц в байте.
5. Посчитать количество чётных нулей в байте.
6. Подсчитать количество нечётных нулей в байте.

Содержание отчёта

Отчёт по лабораторной работе должен содержать:

1. Титульный лист, включающий номер и название лабораторной работы.
2. Цель лабораторной работы.
3. Задание на лабораторную работу.
4. Текст программы на языке ассемблера с поясняющими комментариями.
5. Снимки экрана, поясняющие работу программы.
6. Снимки экрана, отображающие результаты работы программы.
7. Вывод.

Содержание

Список литературы

1. *Component market of Russia* – Рынок электронных компонентов в цифрах [Электронный ресурс] : Микроконтроллеры: статистика запросов на *eFind.ru* – *Component market of Russia*. *URL*: *https://commarketru.com/mikrokontrollery-statistika-zaprosov-na-efind-ru/* (дата обращения: 27.02.2011).
2. Хусаинов, Р. З. Программирование микроконтроллеров *ATmega8535*: методические указания к выполнению лабораторных работ / Р. З. Хусаинов, В. Б. Садов. – Челябинск. – 2009. – 123 с.
3. Самый информированный сервер микроэлектроника, описания – *rs232, rs232*, микросхемы, *hd44780, atmel*, ацп, цап, *irda*, микроконтроллер [Электронный ресурс] : Система команд микроконтроллера *avr* Система команд 8-разрядных *RISC* микроконтроллеров семейства *AVR*. *URL: http://www.gaw.ru/html.cgi/txt/doc/micros/avr/asm/start.htm* (дата обращения: 12.05.2011).
4. Балакина, Е. П. Интегрирование программных модулей ассемблера в среде *DELPHI*. Методические указания к лабораторным работам по дисциплине «Машинно-ориентированные языки» / Е. П. Балакина, Л. Н. Воробьёва, В. А. Гречишников. – М.: МИИТ. – 2009. – 56 с.

Приложение 1

Таблица 8 – Арифметические и логические команды

Мнемо-ника	**Операнды**	**Описание**	**Операция**	**Флаги**	**Кол-во циклов**
ADD	*Rd, Rr* $0 \le d \le 31$, $0 \le r \le 31$	Сложить без переноса	*Rd* ← *Rd + Rr*	*Z, C, N, V, H*	1
ADC	*Rd, Rr* $0 \le d \le 31$, $0 \le r \le 31$	Сложить с переносом	*Rd* ← *Rd + Rr + C*	*Z, C, N, V, H*	1
ADIW	*Rd, k* *dE* {24, 26, 28, 30} $0 \le k \le 63$	Сложить непосредственное значение со словом	*Rdh:Rdl* ← *Rdh:Rdl + k*	*Z, C, N, V*	2
SUB	*Rd, Rr* $0 \le d \le 31$, $0 \le r \le 31$	Вычесть без заёма	*Rd* ← *Rd – Rr*	*Z, C, N, V, H*	1
SUBI	*Rd, k* $16 \le d \le 31$ $0 \le k \le 255$	Вычесть непосредственное значение	*Rd* ← *Rd – K*	*Z, C, N, V, H*	1
SBC	*Rd, Rr* $0 \le d \le 31$, $0 \le r \le 31$	Вычесть с заёмом	*Rd* ← *Rd – Rr – C*	*Z, C, N, V, H*	1
SBCI	*Rd, k* $16 \le d \le 32$, $0 \le k \le 255$	Вычесть непосредственное значение с заёмом	*Rd* ← *Rd – K – C*	*Z, C, N, V, H*	1
SBIW	*Rd, k* *dE* {24, 26, 28, 30} $0 \le k \le 63$	Вычесть непосредственное значение из слова	*Rdh:Rdl* ← *Rdh:Rdl – k*	*Z, C, N, V*	2

AND	*Rd, Rr* $0 \le d \le 31$, $0 \le r \le 31$	Выполнить логическое умножение	*Rd* ← *Rd • Rr*	*Z, N, V*	1
ANDI	*Rd, k* $16 < d < 31$ $0 < k \le 255$	Выполнить логическое умножение	*Rd* ← *Rd • K*	*Z, N, V*	1
OR	*Rd, Rr* $0 \le d \ge 31$, $0 \le r \le 31$	Выполнить логическое сложение	*Rd* ← *Rd* ∨ *Rr*	*Z, N, V*	1
ORI	*Rd, k* $16 \le d \le 31$, $0 \le k \le 255$	Выполнить логическое сложение с непосредственным значением	*Rd* ← *Rd* ∨ *k*	*Z, N, V*	1
EOR	Rd, Rr $0 \le d \le 31$, $0 \le r \le 31$	Выполнить исключающее ИЛИ	Rd ← Rd ⩛ Rr	*Z, N, V*	1
COM	*Rd* $0 \le d \le 31$	Выполнить дополнение до единицы	*Rd* ← *SFF-Rd*	*Z, C, N, V*	1
NEG	*Rd* $0 \le d \le 31$	Выполнить дополнение до двух	*Rd* ← *S00 - Rd*	*Z, C, N, V, H*	1
SBR	*Rd, k* $16 \le d \le 31$, $0 \le k \le 255$	Установить биты в регистре	*Rd* ← *Rd* ∨ *k*	*Z, N, V*	1
CBR	*Rd, k* $16 \le d \le 31$ $0 \le k \le 255$	Очистить биты в регистре	*Rd* ← *Rd • (SFF-k)*	*Z, N, V*	1
INC	*Rd* $0 \le d \le 31$	Инкрементировать	*Rd* ← *Rd + 1*	*Z, N, V*	1
DEC	*Rd* $0 \le d \le 31$	Декрементировать	*Rd* ← *Rd - 1*	*Z, N, V*	1

TST	*Rd* $0 \le r \le 31$	Проверить на ноль или минус	*Rd← Rd. Rd*	*Z, N, V*	1
CLR	*Rd* $0 \le d \le 31$	Очистить регистр	*Rd ← Rd ⊕ Rd*	*Z, N, V*	1
SER	*Rd* $16 \le d \le 31$	Установить все биты регистра	*Rd ← SFF*	нет	1
MUL	*Rd, Rr* $0 \le d \le 31$, $0 \le r \le 31$	Беззнаковое умножение целых чисел	*R1:R0 ← Rd * Rr*	*Z, C*	2
MULS	*Rd, Rr* $16 \le d \le 31$, $16 \le r \le 31$	Умножение целых чисел с учётом знака	*R1:R0 ← Rd * Rr*	*Z, C*	2
MULSU	*Rd, Rr* $16 \le d \le 23$, $16 \le r \le 23$	Целочисленное умножение числа со знаком на число без знака	*R1:R0 ← Rd * Rr*	*Z, C*	2
FMUL	*Rd, Rr* $16 \le d \le 23$, $16 \le r \le 23$	Беззнаковое умножение дробных чисел	*R1:R0 ← (Rd * Rr) << 1*	*Z, C*	2
FMULS	*Rd, Rr* $16 \le d \le 23$, $16 \le r \le 23$	Умножение дробных чисел с учётом знака	*R1:R0 ← (Rd * Rr) << 1*	*Z, C*	2

FMULSU	*Rd, Rr* $16 \le d \le 23$, $16 \le r \le 23$	Умножение дробного числа со знаком на дробное число без знака	*R1:R0 ← (Rd * Rr) << 1*	*Z, C*	2
CP	*Rd, Rr* $0 \le d \le 31$, $0 \le r \le 31$	Сравнить	*Rd - Rr*	*Z, C, N, V, H*	1
CPC	*Rd, Rr* $0 \le d \le 31$, $0 \le r \le 31$	Сравнить с учётом переноса	*Rd - Rr - C*	*Z, C, N, V, H*	1
CPI	*Rd, k* $16 \le d \le 31$, $0 \le k \le 255$	Сравнить с константой	*Rd - k*	*Z, C, N, V, H*	1

Таблица 9 – Команды сдвигов и операций с битами

Мнемо-ника	**Операнды**	**Описание**	**Операция**	**Флаги**	**Кол-во циклов**
LSL	*Rd* $0 \le d \le 31$	Логически сдвинуть влево	*Rd(n+1) ← Rd(n),* *Rd(0) ← 0,* *C← Rd(7)*	*Z, C, N, V, H*	1
LSR	*Rd* $0 \le d \le 31$	Логически сдвинуть вправо	*Rd(n) ← Rd(n+1),* *Rd(7) ← 0,* *C← Rd(0)*	*Z, C, N, V*	1
ROL	*Rd* $0 \le d \le 31$	Сдвинуть влево через перенос	*Rd(0) ← C,* *Rd(n+1) ← Rd(n),* *C ← Rd(7)*	*Z, C, N, V, H*	1
ROR	*Rd* $0 \le d \le 31$	Сдвинуть вправо через перенос	*Rd(7) ← C,* *Rd(n) ← Rd(n+1),* *C ← Rd(0)*	*Z, C, N, V*	1

ASR	*Rd* $0 \le d \le 31$	Арифметически сдвинуть вправо	*Rd(n)* ← *Rd(n+1), n=0...6,* *Rd(0)* ← *C*	*Z, C, N, V*	1
SWAP	*Rd* $0 \le d \le 31$	Поменять нибблы местами	*Rd(3...0)* <–> *Rd(7...4)*	Нет	1
BSET	*s* $0 \le s \le 7$	Установить флаг	*SREG(s)* ← *1*	*SREG(s)*	1
BCLR	*s* $0 \le s \le 7$	Очистить флаг	*SREG(s)* ← *0*	*SREG(s)*	1
SBI	*P, b* $0 \le P \le 31$, $0 \le b \le 7$	Установить бит в регистр *I/O*	*I/O(P,b)* ← *1*	Нет	2
CBI	*P, b* $0 \le P \le 31$, $0 \le b \le 7$	Очистить бит в регистре *I/O*	*I/O(P,b)* ← *0*	Нет	2
BST	*Rd, b* $0 \le d \le 31$, $0 \le b \le 7$	Переписать бит из регистра во флаг *T*	*T* ← *Rd(b)*	*T*	1
BLD	*Rd, b* $0 \le d \le 31$, $0 \le b \le 7$	Загрузить *T* флаг в бит регистра	*Rd(b)* ← *T*	Нет	1
SEC		Установить флаг переноса	*C* ← *1*	*C*	1
CLC		Очистить флаг переноса	*C* ← *0*	*C*	1
SEN		Установить флаг отрицательного значения	*N* ← *1*	*N*	1

CLN		Очистить флаг отрицательного значения	$N \leftarrow 0$	*N*	1
SEZ		Установить флаг нулевого значения	$Z \leftarrow 1$	*Z*	1
CLZ		Очистить флаг нулевого значения	$Z \leftarrow 0$	*Z*	1
SEI		Установить флаг глобального прерывания	$I \leftarrow 1$	*I*	1
CLI		Очистить флаг глобального прерывания	$I \leftarrow 0$	*I*	1
SES		Установить флаг знака	$S \leftarrow 1$	*S*	1
CLS		Очистить флаг знака	$S \leftarrow 0$	*S*	1
SEV		Установить флаг переполнения	$V \leftarrow 1$	*V*	1
CLV		Очистить флаг переполнения	$V \leftarrow 0$	*V*	1
SET		Установить флаг *T*	$T \leftarrow 1$	*T*	1
CLT		Очистить флаг *T*	$T \leftarrow 0$	*T*	1

SEH		Установить флаг полу переноса	$H \leftarrow 1$	*H*	1
CLH		Очистить флаг полу переноса	$H \leftarrow 0$	*H*	1
NOP		Выполнить холостую команду		Нет	1
SLEEP		Установить режим SLEEP		Нет	1
WDR		Сбросить сторожевой таймер		Нет	1

Таблица 10 – Команды пересылки данных

Мнемо-ника	**Операнды**	**Описание**	**Операция**	**Флаги**	**Кол-во циклов**
ELPM		Расширенная загрузка из памяти программ в регистр *RO*	*R0* ← *(Z+RAMPZ)*	Нет	3
MOV	*Rd, Rr* $0 \le d \le 31$, $0 \le r \le 31$	Копировать регистр	$Rd \leftarrow Rr$	Нет	1
LDI	*Rd, k* $16 \le d \le 31$, $0 \le k \le 255$	Загрузить непосредственное значение	$Rd \leftarrow k$	Нет	1
LDS	*Rd, k* $0 \le d \le 31$ $0 \le k \le 65535$	Загрузить из ОЗУ	$Rd \leftarrow k$	Нет	3
LD	*Rd, X* $0 \le d \le 31$	Загрузить косвенно	$Rd \leftarrow X$	Нет	2

LD	*Rd, X+* $0 \le d \le 31$	Загрузить косвенно с постинкрементом	*Rd* ← *X*, *X* ← *X+1*	Нет	2
LD	*Rd, -X* $0 \le d \le 31$	Загрузить косвенно с преддекрементом	*X* ← *X-1*, *Rd* ← *X*	Нет	2
LDD	*Rd, X+q* $0 \le d \le 31$, $0 \le q \le 63$	Загрузить косвенно со смещением	*Rd* ← *X+q*	Нет	2
STS	*k, Rr* $0 \le d \le 31$, $0 \le k \le 65535$	Загрузить непосредственно в ОЗУ	*k* ← *Rr*	Нет	3
ST	*X, Rr* $0 \le r \le 31$	Записать косвенно	*X* ← *Rr*	Нет	2
ST	*X+, Rr* $0 \le r \le 31$	Записать косвенно с постинкрементом	*X* ← *Rr*, *X* ← *X + 1*	Нет	2
ST	*-X, Rr* $0 \le r \le 31$	Записать косвенно с преддекрементом	*X* ← *X-1*, *X* ← *Rr*	Нет	2
STD	*X+q, Rr* $0 \le r \le 31$, $0 \le q \le 63$	Записать косвенно со смещением	*X+q* ← *Rr*	Нет	2
LPM		Загрузить байт из памяти программы	*R0* ← *Z*	Нет	3
IN	*Rd, P* $0 \le d \le 31$, $0 \le P \le 63$	Загрузить данные из порта *I/O* в регистр	*Rd* ← *P*	Нет	1
OUT	*P, Rr* $0 \le r \le 31$,	Записать данные из	*P* ← *Rr*	Нет	1

	$0 \le P \le 63$	регистра в порт *I/O*			
PUSH	*Rr* $0 \le r \le 31$	Сохранить регистр в стеке	*STACK* ← *Rr*	Нет	2
POP	*Rr* $0 \le r \le 31$	Загрузить в регистр из стека	*Rr* ← *STACK*	Нет	2

Таблица 11 – Команды переходов

Мнемо-ника	**Операнды**	**Описание**	**Операция**	**Флаги**	**Кол-во циклов**
RJMP	*k* $-2k < k < 2k$	Перейти относительно	*PC* ← *PC + k + 1*	Нет	2
LJMP		Перейти косвенно	*PC* ← *Z*	Нет	2
JMP	*k* $0 < k < 4M$	Перейти	*PC* ← *k*	Нет	3
RCALL	*k* $-2k \le k \le 2k$	Вызвать подпрограмму относительно	*PC* ← *PC + k + 1*	Нет	3
ICALL		Вызвать подпрограмму косвенно	*PC* ← *Z*	Нет	3
CALL	*k* $0 \le k \le 64k$	Выполнить длинный вызов подпрограммы	*PC* ← *k*	Нет	4
RET		Вернуться из подпрограммы	*PC* ← *STACK*	Нет	4

RETI		Вернуться из прерывания	*PC ← STACK*	*I*	4
CPSE	*Rd, Rr* $0 \le d \le 31$, $0 \le r \le 31$	Сравнить и пропустить, если равно	*если Rd=Rr, то PC ← PC + 2 (или 3)*	Нет	½/3
SBRC	*Rr, b* $0 \le r \le 31$, $0 \le b \le 7$	Пропустить, если бит в регистре очищен	*если Rr(b)=0, то PC ← PC + 2 (или 3)*	Нет	½/3
SBRS	*Rr, b* $0 \le r \le 31$, $0 \le b \le 7$	Пропустить, если бит в регистре установлен	*если Rr(b)=1, то PC ← PC + 2 (или 3)*	Нет	½/3
SBIC	*P, b* $0 \le P \le 31$, $0 \le b \le 7$	Пропустить, если бит в регистре *I/O* очищен	*если I/O P(b)=0, то PC ← PC + 2 (или 3)*	Нет	½/3
SBIS	*P, b* $0 \le r \le 31$, $0 \le b \le 7$	Пропустить, если бит в регистре *I/O* установлен	*если I/O P(b)=1, то PC ← PC + 2 (или 3)*	Нет	½/3
BRBS	*s, k* $0 \le s \le 7$, $-64 \le k \le 63$	Перейти, если бит в регистре статуса установлен	*если SREG(s)=1, то PC ← PC + k + 1*	Нет	½

BRBC	*s, k* $0 \le s \le 7$, $-64 \le k \le 63$	Перейти, если бит в регистре статуса очищен	*если SREG(s)=0, то* *PC ←* *PC + k + 1*	Нет	½
BREQ	*k* $-64 \le k \le 63$	Перейти, если равно	*если Rd=Rr, Z=1, то* *PC ←* *PC + k + 1*	Нет	½
BRNE	*k* $-64 \le k \le 63$	Перейти, если не равно	*если Rd≠Rr, Z=0, то* *PC ←* *PC + k + 1*	Нет	½
BRCS	*k* $-64 \le k \le 63$	Перейти, если флаг переноса установлен	*если C = 1, то* *PC ←* *PC + k + 1*	Нет	1/2
BRCC	*k* $-64 \le k \le 63$	Перейти, если флаг переноса очищен	*если C = 0, то* *PC ←* *PC + k + 1*	Нет	1/2
BRSH	*k* $-64 \le k \le 63$	Перейти, если равно или больше (без знака)	*если* *Rd < Rr,* *C = 0, то* *PC ←* *PC+ k + 1*	Нет	1/2
BRLO	*k* $-64 \le k \le 63$	Перейти, если меньше (без знака)	*если* *Rd < Rr,* *C = 1, то* *PC ←* *PC + k + 1*	Нет	1/2
BRMI	*k* $-64 \le k \le 63$	Перейти, если минус	*если N = 1, то* *PC ← P* *C + k + 1*	Нет	1/2

BRPL	*k* -64 ≤ *k* ≤ 63	Перейти, если плюс	*если N = 0, то* *PC* ← *PC + k + 1*	Нет	1/2
BRGE	*k* -64 ≤ *k* ≤ 63	Перейти, если больше или равно (с учётом знака)	*если* *Rd > Rr, N⊕V = 0, то PC* ← *PC + k + 1*	Нет	1/2
BRLT	*k* -64 ≤ *k* ≤ 63	Перейти, если меньше чем (со знаком)	*если* *Rd < Rr, N⊕V = 1), то PC* ← *PC + k + 1*	Нет	1/2
BRHS	*k* -64 ≤ *k* ≤ 63	Перейти, если флаг полуперено са установлен	*если H = 1, то* *PC* ← *PC + k + 1*	Нет	1/2
BRHC	*k* -64 ≤ *k* ≤ 63	Перейти, если флаг полуперено са очищен	*если H = 0, то* *PC* ← *PC + k + 1*	Нет	1/2
BRTS	*k* -64 ≤ *k* ≤ 63	Перейти, если флаг *T* установлен	*если T = 1, то* *PC* ← *PC + k + 1*	Нет	1/2
BRTC	*k* -64 ≤ *k* ≤ 63	Перейти, если флаг *T* очищен	*если T = 0, то* *PC* ← *PC + k + 1*	Нет	1/2
BRVS	*k* -64 ≤ *k* ≤ 63	Перейти, если флаг переполнен ия установлен	*если V = 1, то* *PC* ← *PC + k + 1*	Нет	1/2

BRVC	*k* -64 ≤ *k* ≤ 63	Перейти, если флаг переполнения очищен	*если V = 0, то* *PC* ← *PC + k + 1*	Нет	1/2
BRIE	*k* -64 ≤ *k* ≤ 63	Перейти, если глобальное прерывание разрешено	*если I = 1, то* *PC* ← *PC + k + 1*	Нет	1/2
BRID	*k* -64 ≤ *k* ≤ 63	Перейти, если глобальное прерывание запрещено	*если I = 0, то* *PC* ← *PC + k + 1*	Нет	1/2

УЧЕБНО-МЕТОДИЧЕСКОЕ ИЗДАНИЕ

Логинова Людмила Николаевна
Сафронов Антон Игоревич

Язык Ассемблера для микроконтроллеров *ATmega8535*

Методические указания к лабораторным работам

Подписано к печати Формат 60×84/16 Тираж **100** экз. **06.12.2011**

Усл. печ. л. **5,25** Заказ № **75** Изд. № **285-11**

127994, Москва, ул. Образцова, 9 стр.9.
Типография МИИТа.