ЛАБОРАТОРНАЯ РАБОТА 2 Программная реализация арифметических преобразований однобайтных чисел

2.1 Индивидуальное задание

Разработать программную реализацию арифметического преобразования выражения, заданного *i*-ой строкой (*i* — порядковый номер по журналу посещаемости) нижеприведенной таблицы. Однобайтный результат вывести на светодиодную линейку, подключённую к порту B. Проверку работы программы осуществить в AVRstudio, Proteus и на отладочной плате EasyAVR 5A.

- a, b, c, d – переменные, расположенные в области SRAM (DSEG),
- g, f – константы, расположенные во flash-памяти программ (CSEG),
- численные коэффициенты формул задаются командой с непосредственной адресацией.

Все исходные числа, промежуточные и окончательные результаты должны находиться в пределах одного байта (0–255). Для этого предварительно продумать значения переменных (a, b, c, d) и констант (f, g).

i	задание	*i*	задание
1	$(a + b) / 8 + 2c*f - 2d*g$	14	$(f + g) / 8 + 2c*f - 2d*b$
2	$a*g + c^2 - 6f$	15	$fg^2 - (a + b - c) / 8$
3	$a^2 + 2ag + g^2 - ab$	16	$8ag + bc^2 - (g + f)/4$
4	$(a + b)^3 - 12fg + c/4$	17	$(f + g)^3 - 6bc + a/4$
5	$(a - b)^2 + 2fg^3$	18	$a^3 + 3ag^2 + 3b^2f + g^3$
6	$fg^3 - (a + b) / 8$	19	$4ag^2 - (a*f + b) / 8$
7	$(a - b)^3 + 2fg^2$	20	$(f + g + a)/8 + 2c*f - 2d*b$
8	$2fg^2 - (a*c + b) / 8$	21	$(f - g^2 + a^3)*b - 5c$
9	$(a + 2b - 3f) - g^3$	22	$2[(a + b)^2 - 12fg] + c/4$
10	$(a + f)^3 + bg^2$	23	$8a^2f^2 - (b + c + g)/2$
11	$(a + b) / 4 - 4fg^2$	24	$a^2 + 2af + g^3 - (ab)^2$
12	$(f - g)^2 - a*b$	25	$(a + b - c) / 4 + (4fg)^2$
13	$4c*f + a*b - g/4$	26	$(c + d + f)^2 + (a - b)^2 - g/16$

Примечание. Целочисленное деление на $\mathbf{2^n}$ реализуется командой логического сдвига вправо LSR (для деления на $\mathbf{2^n}$ команда выполняется **n** раз).

2.2 Контрольные вопросы

1. Каким образом в программу можно включить файл с определенными символическими именами регистров ввода-вывода?
2. Как регистру общего назначения МК AVR присвоить символическое имя?
3. Как присвоить символическим именам конкретные значения (задать символическую константу)?
4. Как разместить команду программы по конкретному адресу Flash-памяти программ?
5. Как осуществить подготовку решения задачи — разместить значения переменных в области памяти данных?
6. Как осуществить подготовку решения задачи — разместить значения констант во Flash-памяти программ?
7. Как загрузить в регистр R18 байтовую константу из программной памяти, имеющую символический адрес CONST?
8. Как перемножить 2 беззнаковых байтовых числа, находящихся в регистрах R16, R17. Сколько байтов может занимать результат умножения?
9. Как из содержимого регистра R18 вычесть содержимое регистра R19? В каком регистре будет находиться результат вычитания?
10. Как к содержимому регистра R16 прибавить содержимое регистра R17? В каком регистре будет находиться результат сложения?
11. Как поделить содержимое регистра R20 на 16?
12. Как поделить содержимое регистра R17 на 2^n?
13. Укажите в написанной вами программе все команды с непосредственной адресацией.
14. Укажите в написанной вами программе все команды с прямой адресацией.
15. Покажите в написанной вами программе все команды с косвеннорегистровой адресацией.
16. Покажите в написанной вами программе все команды с регистровой адресацией.
17. Как значение, находящееся в регистре R17 возвести в квадрат?
18. Как значение, находящееся в регистре R18 возвести в куб?
19. Для чего в программах на ассемблере используется директива .ORG?
20. Как заставить программу зависнуть в бесконечном цикле? Приведите фрагмент программы на ассемблере.
21. Как узнать машинный код команды, не имея справочника команд, а располагая установленным ПО: AVRstudio и (или) Proteus?
22. Для чего используются метки в программах на ассемблере? Как узнать численное значение метки (адрес памяти программ или данных)?

2.3 Методические рекомендации по программной реализации арифметического преобразования однобайтных чисел

Целью данной работы является ознакомление с основными принципами программирования на языке Ассемблер, с методами адресации операндов, с системой команд языка и основными директивами.

Следует отметить, что программа на ассемблере, реализующая задание, будет состоять из двух частей (рис. 2.1):

1. Подготовка условия задачи — размещение констант во flash-памяти и размещение переменных с заданными значениями в памяти данных. Для размещения констант в памяти программ используются директивы для инициализированных данных: .DB, .DW. Для размещения переменных в памяти данных используются директивы резервирования места для неинициализированных данных: .BYTE N, .WORD N (где N — количество переменных) с последующими командами пересылки по адресам переменных в SRAM необходимых исходных значений.

2. Собственно решение задачи с выполнением арифметических операций, сохранением промежуточных результатов, нахождением окончательного результата и вывода его в порт.

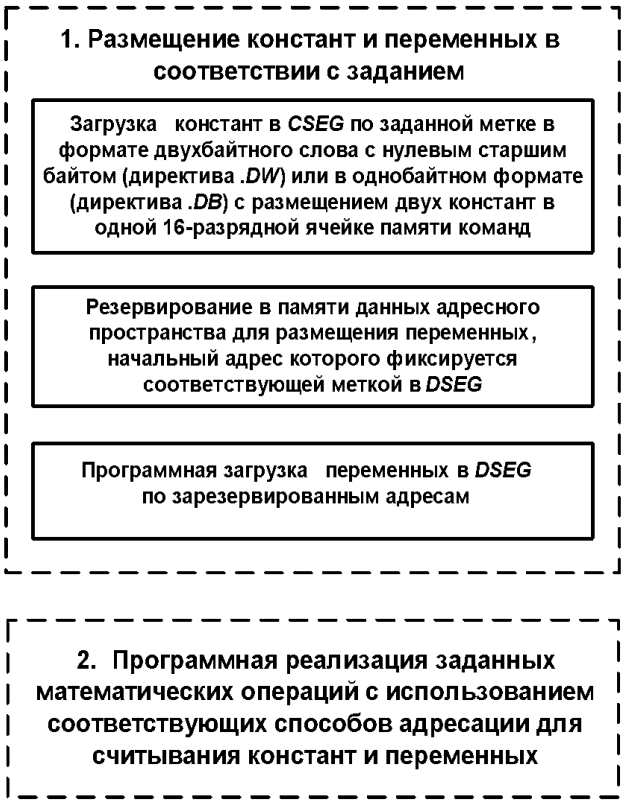

Рисунок 2.1 — Обобщенный алгоритм реализации операций над константами и переменными, расположенными в памяти микроконтроллера

Следует обратить внимание на способы адресации переменных и констант. Рекомендуется для удобства работы при написании программы иметь под рукой распечатку кодов команд (инструкций) и директив (см. стр.) или справочник Справка по ассемблеру AVR на русском в PDF-формате.

Пример 1. Составить программу вычисления функции: $15f*g + a/8 - 4b$

```
/* Составить программу вычисления следующей функции:      15f*g + a/8 – 4b
Константы «f» «g» хранятся в памяти программ, переменные «a» и «b» — в памяти
данных (SRAM) */
/*директива включения в программу файла  с именами регистров ввода/вывода */
        .include "m16def.inc"
//директивы присвоения регистрам символических имен
        .def    Temp=R16            ;присвоение регистру R16 имени Temp
        .def    res=R11             ;регистру R11 результата вычисления
/*директивы присвоения численных значений константам и переменным*/
        .equ    a=16
        .equ    b=17
        .equ    f=3
        .equ    g=4
//директива присвоения  исходного адреса
        .org    00
        rjmp    start               /*переход к нач. адр. разработанной программы */
```

```
        .org    0x2A                    /*директива присвоения начального адреса
                                        для точки входа разработанной программы*/
start:                                  ;метка начального адреса (точки входа)
/*формирование исходного массива переменных в SRAM с использованием
прямой адресации*/
        ldi     temp,0xFF
        out     ddrc, temp              ;настройка порта C на вывод данных
        ldi     temp,a                  ;загрузка переменной «a» в память SRAM
        sts     var,temp
        ldi     temp,b          ;загрузка переменной «b» в память SRAM
        sts     var+1,temp
//реализация вычислений
        ldi     ZL,low(const*2)         /*считывание конст. «f» из памяти программ CSEG*/
        ldi     ZH,high(const*2)
        lpm     R0,Z
        ldi     temp,15                 ;загрузка множителя 15 в регистр temp
        mul     R0,temp                 ;умножение 15*f (результат размещается в R0)
        mov     res,R0                  ;перенос произведения 15*f в рег. результата
        ldi     ZL,low(2*(const+1)) ;считывание конст. «g» из памяти программ (CSEG)
        ldi     ZH,high(2*(const+1))
        lpm     R0,Z
        mul     R0,res                  ;реализация первого слагаемого 15f*g
        mov     res,R0
        lds     temp,var                ;считывание «a» из SRAM
        lsr     temp                    ;деление «a» на 8
        lsr     temp
        lsr     temp
/*сложение частного от деления с предыдущим результатом*/
        add     res,temp
        lds     R0,var+1                ;считывание «b» из SRAM
        ldi     temp,4          ;нахождение произведения 4*b
        mul     temp,R0
        sub     res,R0                  /*вычитание 4b из суммы двух первых слагаемых*/
        out     PORTC,res               ;вывод результата в порт C
/*организация бесконечного цикла для завершения программы*/
L1:     rjmp    L1
//размещение констант в памяти программ
        .CSEG                           ;директива обращения к памяти программ
        .org    $50     ;директива  начального адреса констант (должен быть не меньше
                        ;размера предшествующего программного кода). При отсутствии
                        ;директивы, константы разместятся сразу за программой —
                        ;это удобнее и предотвратит ошибки перекрытия (overlap)
const:                  ;метка начала размещения констант
        .DW     f,g                     /*запись в CSEG двух констант в формате
                                        двухбайтного слова (старший байт нулевой)*/
/*резервирование ячеек памяти SRAM для размещения переменных «a» и «b»*/
        .DSEG                           ;директива обращения к памяти данных
        .org    $75                     ;директива  начального адреса переменных
```

```
var:                    /*метка начального адреса  DSEG переменных*/
        .BYTE  2        /*резервирование в DSEG 2-х однобайтных ячеек*/
```

ПРИЛОЖЕНИЕ А Сводная таблица команд микроконтроллеров семейства ATmega

Обозначение	Операнды	Описание	Операция	Флаги	Такты
Команды пересылки данных					
MOV	**Rd, Rr**	Копия регистра	Rd ← Rr	—	1
MOVW	**Rd, Rr**	Копия регистр. слова	Rd+1:Rd ← Rr+1:Rr, r, d even	—	1
LDI	**Rd, K8**	Загрузка константы	Rd ← K8 $16 \le d \le 31$	—	1
LD	**Rd, X**	Косвенная загрузка	Rd ← [X]	—	2*
LD	**Rd, X+**	Косвенная загрузка с постинкрементом	Rd ← [X], X←X+1	—	2*
LD	**Rd, –X**	Косвенная загрузка с преддекрементом	X←X–1, Rd ← [X]	—	2*
LD	**Rd, Y**	Косвенная загрузка	Rd ← [Y]	—	2*
LD	**Rd, Y+**	Косв. загр. с постинкрем.	Rd ← [Y], Y←Y+1	—	2*
LD	**Rd, –Y**	Косв. загр. с преддекрем.	Y←Y–1, Rd ← [Y]	—	2*
LD	**Rd, Z**	Косвенная загрузка	Rd ← [Z]	—	2*
LD	**Rd, Z+**	Косвенная загрузка с постинкрементом	Rd ← [Z], Z←Z+1	—	2*
LD	**Rd, –Z**	Косвенная загрузка с преддекрементом	Z←Z–1, Rd ← [Z]	—	2*
LDD	**Rd, Y+q**	Косвенная загрузка со смещением (относ. адр.)	Rd ← [Y+q]	—	2*
LDD	**Rd, Z+q**	Косвенная загрузка со смещением (относ. адр.)	Rd ← [Z+q]	—	2*
LDS	**Rd, k**	Прямое чтение ЯП	Rd ← (k)	—	2*
ST	**X, Rr**	Косвенная запись	[X] ← Rr	—	2*
ST	**X+, Rr**	Косвенная запись с постинкрементом	[X] ← Rr, X←X+1	—	2*
ST	**–X, Rr**	Косвенная запись с преддекрементом	X←X–1, [X] ← Rr	—	2*
ST	**Y, Rr**	Косвенная запись	[Y] ← Rr	—	2*
ST	**Y+, Rr**	Косвенная запись с постинкрементом	[Y] ← Rr, Y←Y+1	—	2
ST	**–Y, Rr**	Косвенная запись с преддекрементом	Y←Y–1, [Y] ← Rr	—	2
ST	**Z, Rr**	Косвенная запись	[Z] ← Rr	—	2

Команды пересылки данных (окончание)					
ST	**Z+, Rr**	Косвенная запись с постинкрементом	[Z] ← Rr, Z←Z+1	—	2
ST	**–Z, Rr**	Косвенная запись с преддекрементом	Z←Z–1, [Z] ← Rr	—	2
STD	**Y+q, Rr**	Косв. запись со смещ-ем	[Y+q] ← Rr	—	2
STD	**Z+q, Rr**	Косв. запись со смещ-ем	[Z+q] ← Rr	—	2
STS	**k, Rr**	Прямая запись ЯП	(k) ← Rr	—	2*
LPM		Загрузка байта памяти программ	R0 ← [Z]	—	3
LPM	**Rd, Z**	Загрузка байта памяти программ	Rd ← [Z]	—	3
LPM	**Rd, Z+**	Загрузка байта памяти программ с постинкрементом	Rd ← [Z], Z ← Z+1	—	3
SPM		Запись в память программ	[Z] ← R1:R0	—	—
PUSH	**Rr**	Запись регистра в стек	Stack←Rr, SP=SP–1	—	2
POP	**Rd**	Загрузка регистра из стека	SP=SP+1, Rd←Stack	—	2
OUT	**P, Rr**	Запись данных из регистра в порт I/O	P ← Rr	—	1
IN	**Rd, P**	Загрузка данных из порта I/O в регистр	Rd ← P	—	1
Арифметические операции					
ADD	**Rd, Rr**	Сложение без переноса	Rd ← Rd + Rr	Z,C,N,V,H,S	1
ADC	**Rd, Rr**	Сложение с переносом	Rd ← Rd + Rr + C	Z,C,N,V,H,S	1
ADIW	**Rdh:Rdl,K6**	Сложение константы со словом	Rdh:Rdl ← Rdh:Rdl + K6 24≤d≤30, d — четный	Z,C,N,V,S	2
SUB	**Rd, Rr**	Вычитание без переноса	Rd ← Rd – Rr	Z,C,N,V,H,S	1
SUBI	**Rd, K8**	Вычитание константы	Rd ← Rd – K8	Z,C,N,V,H,S	1
SBC	**Rd, Rr**	Вычитание с заемом	Rd ← Rd – Rr – C	Z,C,N,V,H,S	1
SBCI	**Rd, K8**	Вычитание константы из РОН с заемом	Rd ← Rd – K8 – C 16≤d≤31	Z,C,N,V,H,S	1
SBIW	**Rdl, K6**	Вычитание константы из слова	Rdh:Rdl ← Rdh:Rdl – K6 24≤d≤30, d — четный	Z,C,N,V,S	2
CP	**Rd, Rr**	Сравнение	Rd – Rr	Z,C,N,V,H,S	1
CPC	**Rd, Rr**	Сравн-е с учетом перен.	Rd – Rr – C	Z,C,N,V,H,S	1
CPI	**Rd, K8**	Сравнение с константой	Rd – K8	Z,C,N,V,H,S	1
INC	**Rd**	Инкремент	Rd ← Rd + 1	Z, N,V,S	1

DEC	**Rd**	Декремент	Rd ← Rd – 1	Z, N,V,S	1
Арифметические операции (окончание)					
MUL	**Rd, Rr**	Умножение беззнаковых	R1:R0←Rd×Rr	Z, C	2
MULS	**Rd, Rr**	Умножение со знаком	R1:R0←Rd×Rr 16≤d≤31	Z, C	2
MULSU	**Rd, Rr**	Умножение беззнакового на число со знаком	R1:R0 ← Rd × Rr 16≤d≤23	Z, C	2
FMUL	**Rd, Rr**	Умножение дробных беззнаковых чисел	R1:R0←(Rd×Rr)<<1 16≤d≤23	Z, C	2
FMULS	**Rd, Rr**	Умножение дробных чисел со знаком	R1:R0←(Rd×Rr)<<1 16≤d≤23	Z, C	2
FMULSU	**Rd, Rr**	Умножение дробных беззнакового и знакового	R1:R0←(Rd×Rr)<<1 16≤d≤23	Z, C	2
Логические операции					
AND	**Rd, Rr**	Логическое И	Rd ← Rd·Rr	Z, N,V,S	1
ANDI	**Rd, K8**	Логическое И с константой	Rd ← Rd·K8 16≤d≤31	Z, N,V,S	1
OR	**Rd, Rr**	Логическое ИЛИ	Rd ← Rd ∨ Rr	Z, N,V,S	1
ORI	**Rd, K8**	Логическое ИЛИ с конст.	Rd ← Rd ∨ K8 16≤d≤31	Z, N,V,S	1
EOR	**Rd, Rr**	Исключающее ИЛИ	Rd ← Rd ⊕ Rr	Z, N,V,S	1
COM	**Rd**	Дополнение до единицы	Rd ← 0xFF – Rd	Z,C,N,V,S	1
NEG	**Rd**	Дополнение до двух	Rd ← 0x00 – Rd	Z,C,N,V,H,S	1
TST	**Rd**	Проверка на ноль и минус	Rd ← Rd·Rd	Z,N,V,S	1
SBR	**Rd, K8**	Установка бит в регистре	Rd ← Rd ∨ K8 16≤d≤31	Z,N,V,S	1
CBR	**Rd, K8**	Очистка бит в регистре	Rd←Rd·(0xFF–K8) 16≤d≤31	Z,N,V,S	1
SER	**Rd**	Установка бит регистра	Rd ← 0xFF 16≤d≤31	—	1
CLR	**Rd**	Очистка регистра	Rd ← Rd ⊕ Rd	Z,N,V,S	1
Команды операций над битами					
LSL	**Rd**	Логический сдвиг влево	Rd(n+1) ← Rd(n), Rd(0) ← 0 C←Rd(7)	Z,C,N,V,S	1
LSR	**Rd**	Логический сдвиг вправо	Rd(n) ← Rd(n+1), Rd(7) ← 0 C←Rd(0)	Z,C,N,V,S	1
ROL	**Rd**	Сдвиг влево через перенос	Rd(0)←C, Rd(n+1) ← Rd(n), C ← Rd(7)	Z,C,N,V,H, S	1
ROR	**Rd**	Сдвиг вправо через перенос	Rd(7) ← C, Rd(n) ← Rd(n+1), C ← Rd(0)	S,V,N,Z,C	1

ASR	**Rd**	Арифметический сдвиг вправо	Rd(n) ← Rd(n+1), n=0...6 C ← Rd(0)	S,V,N,Z,C	1
SWAP	**Rd**	Перестановка тетрад	Rd(3...0) ← Rd(7…4), Rd(7…4) ← Rd(3…0)	—	1
Команды операций над битами (окончание)					
SBI	**P, b**	Установка бита в рег. I/O	I/O (P,b)←1 $0 \le IOR \le 31$	—	2
CBI	**P, b**	Очистка бита в рег. I/O	I/O (P, b)←0 $0 \le IOR \le 31$	—	2
BSET	**s**	Установка флага	SREG(s) ← 1	SREG(s)	1
BCLR	**s**	Очистка флага	SREG(s) ← 0	SREG(s)	1
BLD	**Rr, b**	Загрузка флага T в бит регистра	Rd(b) ← T	—	1
BST	**Rd, b**	Перенос бита из регистра в флаг T	T ← Rd(b)	T	1
CLC		Очистка флага переноса	C ← 0	C	1
SEC		Установка флага переноса	C ← 1	C	1
CLZ		Очистить флаг нуля	Z ← 0	Z	1
SEZ		Установка флага нулевого значения	Z ← 1	Z	1
CLN		Очистка флага отр. знач.	N ← 0	N	1
SEN		Установка флага отр. зн.	N ← 1	N	1
CLV		Очистка флага перен.	V ← 0	V	1
SEV		Установка флага перен.	V ← 1	V	1
CLS		Очистка флага знака	S ← 0	S	1
SES		Установка флага знака	S ← 1	S	1
CLH		Очистка флага полупереноса	H ← 0	H	1
SEH		Установка флага полупереноса	H ← 1	H	1
CLT		Очистка флага T	T ← 0	T	1
SET		Установка флага T	T ← 1	T	1
CLI		Общее запрещение прерываний	I ← 0	I	1
SEI		Общее разрешение прерываний	I ← 1	I	1
Команды передачи управления (сравнения и ветвления)					
CPSE	**Rd, Rr**	Сравнение, если равно, пропустить след. команду	if (Rd=Rr) PC ← PC+2 or 3	—	1/2/3

SBIC	**P, b**	Пропуск следующей команды, если бит в регистре I/O очищен	if (P(b)=0) PC ← PC+2 or 3 $0 \le IOR \le 31$	—	1/2/3
SBIS	**P, b**	Пропуск следующей команды, если бит в регистре I/O установлен	if (P(b)=1) PC ← PC+2 or 3 $0 \le IOR \le 31$	—	1/2/3
Команды передачи управления (продолжение)					
SBRC	**Rr, b**	Пропуск следующей команды, если бит в регистре очищен	if (Rr(b)=0) PC ← PC+2 or 3	—	1/2/3
SBRS	**Rr, b**	Пропуск следующей команды, если бит в регистре установлен	if (Rr(b)=1) PC ← PC+2 or 3	—	1/2/3
BRBC	**s, k**	Переход, если бит S в регистре статуса очищен	if (SREG(s)=0) then PC ← PC+k+1	—	1/2
BRBS	**s, k**	Переход, если бит S в рег. статуса установлен	if (SREG(s)=1) then PC ← PC+k+1	—	1/2
BRCC	**k**	Переход, если флаг переноса очищен	if (C=0) then PC ← PC+k+1	—	1/2
BRCS	**k**	Переход, если флаг переноса установлен	if (C=1) then PC ← PC+k+1	—	1/2
BRSH	**k**	Переход, если равно или больше (без знака)	if (C=0) then PC ← PC+k+1	—	1/2
BRLO	**k**	Переход, если меньше (без знака)	if (C=1) then PC ← PC+k+1	—	1/2
BRNE	**k**	Переход, если не равно	if (Z=0) then PC ← PC+k+1	—	1/2
BREQ	**k**	Переход, если равно	if (Z=1) then PC ← PC+k+1	—	1/2
BRPL	**k**	Переход, если плюс	if (N=0) then PC ← PC+k+1	—	1/2
BRMI	**k**	Переход, если минус	if (N=1) then PC ← PC+k+1	—	1/2
BRVC	**k**	Переход, если флаг переполнения очищен	if (V=0) then PC ← PC+k+1	—	1/2
BRVS	**k**	Переход, если флаг переполнения установл.	if (V=1) then PC ← PC+k+1	—	1/2
BRGE	**k**	Переход, если больше или равно (с учетом знака)	if (S=0) then PC ← PC+k+1	—	1/2

BRLT	**k**	Переход, если меньше чем (со знаком)	if (S=1) then PC ← PC+k+1	—	1/2
BRHC	**k**	Переход, если флаг полупереноса очищен	if (H=0) then PC ← PC+k+1	—	1/2
BRHS	**k**	Переход, если флаг полупереноса установлен	if (H=1) then PC ← PC+k+1	—	1/2
Команды передачи управления (окончание)					
BRTC	**k**	Переход, если флаг T очищен	if (T=0) then PC ← PC+k+1	—	1/2
BRTS	**k**	Переход, если флаг T установлен	if (T=1) then PC ← PC+k+1	—	1/2
BRID	**k**	Переход, если глобал. прерывание запрещено	if (I=0) then PC ← PC+k+1	—	1/2
BRIE	**k**	Переход, если глобальное прерывание разрешено	if (I=1) then PC ← PC+k+1	—	1/2
RJMP	**k**	Относительный переход	PC ← PC+k+1	—	2
IJMP		Косвенный переход	PC ← Z	—	2
JMP	**k**	Абсолютный переход	PC ← k	—	3
RCALL	**k**	Относительный вызов подпрограммы	(PC)+1→ Stack PC ← PC+k+1	—	3/4*
ICALL		Косвенный вызов подпрограммы	PC ← Z (PC)+1→ Stack	—	3/4*
CALL	**k**	Абсолютный вызов подпрограммы	(PC)+2 → Stack PC ← k	—	4/5*
RET		Возврат из подпрограммы	PC ← Stack	—	4/5*
RETI		Возврат из прерывания	PC ← Stack I = 1	I	4/5*
Команды управления системой					
SLEEP		Переход в режим SLEEP		—	3
WDR		Сброс сторожевого таймера		—	1
NOP		Холостая команда		—	1

Операнды могут быть таких видов:

Rd — Результирующий (и исходный) регистр в регистровом файле

Rr — Исходный регистр в регистровом файле

b — Константа (3 бита), может быть константное выражение

s — Константа (3 бита), может быть константное выражение

P — Константа (5-6 бит), может быть константное выражение

K6 — Константа (6 бит), может быть константное выражение

K8 — Константа (8 бит), может быть константное выражение
k — Константа (размер зависит от инструкции), может быть константное выражение
q — Константа (6 бит), может быть константное выражение
Rdl — R24, R26, R28, R30. Для инструкций ADIW и SBIW.
X, Y, Z — Регистры косвенной адресации (X=R27:R26, Y=R29:R28, Z=R31:R30).