Справка по Ассемблеру для AVR

Общая информация.

Компилятор транслирует исходные коды с языка ассемблера в объектный код. Полученный объектный код можно использовать в симуляторе ATMEL AVR Studio, либо в эмуляторе ATMEL AVR In-Circuit Emulator. Компилятор также генерирует код, который может быть непосредственно запрограммирован в микроконтроллеры AVR. Компилятор генерирует код, который не требует линковки.

Исходные коды.

Компилятор работает с исходными файлами, содержащими инструкции, метки и директивы. Инструкции и директивы, как правило, имеют один или несколько операндов.

Строка кода не должна быть длиннее 120 символов.

Любая строка может начинаться с метки, которая является набором символов заканчивающимся двоеточием. Метки используются для указания места, в которое передаётся управление при переходах, а также для задания имён переменных. Входная строка может иметь одну из четырёх форм:

```
[метка:] директива [операнды] [Комментарий]
[метка:] инструкция [операнды] [Комментарий]
Комментарий
Пустая строка
```

Комментарий имеет следующую форму:

```
; [Текст]
```

Позиции в квадратных скобках необязательны. Текст после точки с запятой (;) и до конца строки игнорируется компилятором. Метки, инструкции и директивы более детально описываются ниже.

Примеры:

```
label:  .EQU var1=100 ; Устанавливает var1 равным 100 (Это директива)
        .EQU var2=200 ; Устанавливает var2 равным 200
test:   rjmp test     ; Бесконечный цикл (Это инструкция)
                      ; Строка с одним только комментарием
                      ; Ещё одна строка с комментарием
```

Компилятор не требует чтобы метки, директивы, комментарии или инструкции находились в определённой колонке строки.

Инструкции процессоров AVR

Арифметические и логические инструкции

Мнемоника	Операнды	Описание	Операция	Флаги	Циклы
ADD	Rd,Rr	Суммирование без переноса	Rd = Rd + Rr	Z,C,N,V,H,S	1
ADC	Rd,Rr	Суммирование с переносом	Rd = Rd + Rr + C	Z,C,N,V,H,S	1
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 переносом	Rd = Rd - Rr - C	Z,C,N,V,H,S	1
SBCI	Rd,K8	Вычитание константы с переносом	Rd = Rd - K8 - C	Z,C,N,V,H,S	1
AND	Rd,Rr	Логическое И	Rd = Rd · Rr	Z,N,V,S	1
ANDI	Rd,K8	Логическое И с константой	Rd = Rd · K8	Z,N,V,S	1
OR	Rd,Rr	Логическое ИЛИ	Rd = Rd V Rr	Z,N,V,S	1
ORI	Rd,K8	Логическое ИЛИ с константой	Rd = Rd V K8	Z,N,V,S	1
EOR	Rd,Rr	Логическое исключающее ИЛИ	Rd = Rd EOR Rr	Z,N,V,S	1
COM	Rd	Побитная Инверсия	Rd = $FF - Rd	Z,C,N,V,S	1
NEG	Rd	Изменение знака (Доп. код)	Rd = $00 - Rd	Z,C,N,V,H,S	1
SBR	Rd,K8	Установить бит (биты) в регистре	Rd = Rd V K8	Z,C,N,V,S	1
CBR	Rd,K8	Сбросить бит (биты) в регистре	Rd = Rd · ($FF - K8)	Z,C,N,V,S	1
INC	Rd	Инкрементировать значение регистра	Rd = Rd + 1	Z,N,V,S	1
DEC	Rd	Декрементировать значение регистра	Rd = Rd -1	Z,N,V,S	1
TST	Rd	Проверка на ноль либо отрицательность	Rd = Rd · Rd	Z,C,N,V,S	1
CLR	Rd	Очистить регистр	Rd = 0	Z,C,N,V,S	1
SER	Rd	Установить регистр	Rd = $FF	None	1
ADIW	Rdl,K6	Сложить константу и слово	Rdh:Rdl = Rdh:Rdl + K6	Z,C,N,V,S	2
SBIW	Rdl,K6	Вычесть константу из слова	Rdh:Rdl = Rdh:Rdl - K 6	Z,C,N,V,S	2

Инструкции ветвления

Мнемоника	Операнды	Описание	Операция	Флаги	Циклы
RJMP	k	Относительный переход	PC = PC + k +1	None	2
IJMP	Нет	Косвенный переход на (Z)	PC = Z	None	2
EIJMP	Нет	Расширенный косвенный переход на (Z)	STACK = PC+1, PC(15:0) = Z, PC(21:16) = EIND	None	2
JMP	k	Переход	PC = k	None	3
RCALL	k	Относительный вызов подпрограммы	STACK = PC+1, PC = PC + k + 1	None	3/4*
ICALL	Нет	Косвенный вызов (Z)	STACK = PC+1, PC = Z	None	3/4*
EICALL	Нет	Расширенный косвенный вызов (Z)	STACK = PC+1, PC(15:0) = Z, PC(21:16) =EIND	None	4*
RET	Нет	Возврат из подпрограммы	PC = STACK	None	4/5*
RETI	Нет	Возврат из прерывания	PC = STACK	I	4/5*
CPSE	Rd,Rr	Сравнить, пропустить если равны	if (Rd ==Rr) PC = PC 2 or 3	None	1/2/3
CP	Rd,Rr	Сравнить	Rd -Rr	Z,C,N,V,H,S	1
CPC	Rd,Rr	Сравнить с переносом	Rd - Rr - C	Z,C,N,V,H,S	1
CPI	Rd,K8	Сравнить с константой	Rd - K	Z,C,N,V,H,S	1
SBRC	Rr,b	Пропустить если бит в регистре	if(Rr(b)==0) PC = PC + 2 or 3	None	1/2/3

		очищен			
SBRS	Rr,b	Пропустить если бит в регистре установлен	if(Rr(b)==1) PC = PC + 2 or 3	None	1/2/3
SBIC	P,b	Пропустить если бит в порту очищен	if(I/O(P,b)==0) PC = PC + 2 or 3	None	1/2/3
SBIS	P,b	Пропустить если бит в порту установлен	if(I/O(P,b)==1) PC = PC + 2 or 3	None	1/2/3
BRBC	s,k	Перейти если флаг в SREG очищен	if(SREG(s)==0) PC = PC + k + 1	None	1/2
BRBS	s,k	Перейти если флаг в SREG установлен	if(SREG(s)==1) PC = PC + k + 1	None	1/2
BREQ	k	Перейти если равно	if(Z==1) PC = PC + k + 1	None	1/2
BRNE	k	Перейти если не равно	if(Z==0) PC = PC + k + 1	None	1/2
BRCS	k	Перейти если перенос установлен	if(C==1) PC = PC + k + 1	None	1/2
BRCC	k	Перейти если перенос очищен	if(C==0) PC = PC + k + 1	None	1/2
BRSH	k	Перейти если равно или больше	if(C==0) PC = PC + k + 1	None	1/2
BRLO	k	Перейти если меньше	if(C==1) PC = PC + k + 1	None	1/2
BRMI	k	Перейти если минус	if(N==1) PC = PC + k + 1	None	1/2
BRPL	k	Перейти если плюс	if(N==0) PC = PC + k + 1	None	1/2
BRGE	k	Перейти если больше или равно (со знаком)	if(S==0) PC = PC + k + 1	None	1/2
BRLT	k	Перейти если меньше (со знаком)	if(S==1) PC = PC + k + 1	None	1/2
BRHS	k	Перейти если флаг внутреннего переноса установлен	if(H==1) PC = PC + k + 1	None	1/2
BRHC	k	Перейти если флаг внутреннего переноса очищен	if(H==0) PC = PC + k + 1	None	1/2
BRTS	k	Перейти если флаг T установлен	if(T==1) PC = PC + k + 1	None	1/2
BRTC	k	Перейти если флаг T очищен	if(T==0) PC = PC + k + 1	None	1/2
BRVS	k	Перейти если флаг переполнения установлен	if(V==1) PC = PC + k + 1	None	1/2
BRVC	k	Перейти если флаг переполнения очищен	if(V==0) PC = PC + k + 1	None	1/2
BRIE	k	Перейти если прерывания разрешены	if(I==1) PC = PC + k + 1	None	1/2
BRID	k	Перейти если прерывания запрещены	if(I==0) PC = PC + k + 1	None	1/2

* Для операций доступа к данным количество циклов указано при условии доступа к внутренней памяти данных, и не корректно при работе с внешним ОЗУ. Для инструкций ICALL, EICALL, RCALL, RET и RETI, необходимо добавить три цикла плюс по два цикла для каждого ожидания в контроллерах с PC меньшим 16 бит (128KB памяти программ). Для устройств с памятью программ свыше 128KB , добавьте пять циклов плюс по три цикла на каждое ожидание.

Инструкции передачи данных

Мнемоника	Операнды	Описание	Операция	Флаги	Циклы
MOV	Rd,Rr	Скопировать регистр	Rd = Rr	None	1
LDI	Rd,K8	Загрузить константу	Rd = K	None	1
LDS	Rd,k	Прямая загрузка	Rd = (k)	None	2*
LD	Rd,X	Косвенная загрузка	Rd = (X)	None	2*
LD	Rd,X+	Косвенная загрузка с пост-инкрементом	Rd = (X), X=X+1	None	2*
LD	Rd,-X	Косвенная загрузка с пре-декрементом	X=X-1, Rd = (X)	None	2*

LD	Rd,Y	Косвенная загрузка	Rd = (Y)	None	2*
LD	Rd,Y+	Косвенная загрузка с пост-инкрементом	Rd = (Y), Y=Y+1	None	2*
LD	Rd,-Y	Косвенная загрузка с пре-декрементом	Y=Y-1, Rd = (Y)	None	2*
LDD	Rd,Y+q	Косвенная загрузка с замещением	Rd = (Y+q)	None	2*
LD	Rd,Z	Косвенная загрузка	Rd = (Z)	None	2*
LD	Rd,Z+	Косвенная загрузка с пост-инкрементом	Rd = (Z), Z=Z+1	None	2*
LD	Rd,-Z	Косвенная загрузка с пре-декрементом	Z=Z-1, Rd = (Z)	None	2*
LDD	Rd,Z+q	Косвенная загрузка с замещением	Rd = (Z+q)	None	2*
STS	k,Rr	Прямое сохранение	(k) = Rr	None	2*
ST	X,Rr	Косвенное сохранение	(X) = Rr	None	2*
ST	X+,Rr	Косвенное сохранение с пост-инкрементом	(X) = Rr, X=X+1	None	2*
ST	-X,Rr	Косвенное сохранение с пре-декрементом	X=X-1, (X)=Rr	None	2*
ST	Y,Rr	Косвенное сохранение	(Y) = Rr	None	2*
ST	Y+,Rr	Косвенное сохранение с пост-инкрементом	(Y) = Rr, Y=Y+1	None	2
ST	-Y,Rr	Косвенное сохранение с пре-декрементом	Y=Y-1, (Y) = Rr	None	2
ST	Y+q,Rr	Косвенное сохранение с замещением	(Y+q) = Rr	None	2
ST	Z,Rr	Косвенное сохранение	(Z) = Rr	None	2
ST	Z+,Rr	Косвенное сохранение с пост-инкрементом	(Z) = Rr, Z=Z+1	None	2
ST	-Z,Rr	Косвенное сохранение с пре-декрементом	Z=Z-1, (Z) = Rr	None	2
ST	Z+q,Rr	Косвенное сохранение с замещением	(Z+q) = Rr	None	2
LPM	Нет	Загрузка из программной памяти	R0 = (Z)	None	3
LPM	Rd,Z	Загрузка из программной памяти	Rd = (Z)	None	3
LPM	Rd,Z+	Загрузка из программной памяти с пост-инкрементом	Rd = (Z), Z=Z+1	None	3
SPM	Нет	Сохранение в программной памяти	(Z) = R1:R0	None	-
IN	Rd,P	Чтение порта	Rd = P	None	1
OUT	P,Rr	Запись в порт	P = Rr	None	1
PUSH	Rr	Занесение регистра в стек	STACK = Rr	None	2
POP	Rd	Извлечение регистра из стека	Rd = STACK	None	2

* Для операций доступа к данным количество циклов указано при условии доступа к внутренней памяти данных, и не корректно при работе с внешним ОЗУ. Для инструкций LD, ST, LDD, STD, LDS, STS, PUSH и POP, необходимо добавить один цикл плюс по одному циклу для каждого ожидания.

Инструкции работы с битами

Мнемоника	Операнды	Описание	Операция	Флаги	Циклы
LSL	Rd	Логический сдвиг влево	Rd(n+1)=Rd(n), Rd(0)=0, C=Rd(7)	Z,C,N,V,H,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	Циклический сдвиг влево через C	Rd(0)=C, Rd(n+1)=Rd(n), C=Rd(7)	Z,C,N,V,H,S	1
ROR	Rd	Циклический сдвиг вправо через C	Rd(7)=C, Rd(n)=Rd(n+1), C=Rd(0)	Z,C,N,V,S	1
ASR	Rd	Арифметический сдвиг вправо	Rd(n)=Rd(n+1), n=0,...,6	Z,C,N,V,S	1
SWAP	Rd	Перестановка тетрад	Rd(3..0) = Rd(7..4), Rd(7..4) = Rd(3..0)	None	1
BSET	s	Установка флага	SREG(s) = 1	SREG(s)	1
BCLR	s	Очистка флага	SREG(s) = 0	SREG(s)	1

SBI	P,b	Установить бит в порту	I/O(P,b) = 1	None	2
CBI	P,b	Очистить бит в порту	I/O(P,b) = 0	None	2
BST	Rr,b	Сохранить бит из регистра в T	T = Rr(b)	T	1
BLD	Rd,b	Загрузить бит из T в регистр	Rd(b) = T	None	1
SEC	Нет	Установить флаг переноса	C =1	C	1
CLC	Нет	Очистить флаг переноса	C = 0	C	1
SEN	Нет	Установить флаг отрицательного числа	N = 1	N	1
CLN	Нет	Очистить флаг отрицательного числа	N = 0	N	1
SEZ	Нет	Установить флаг нуля	Z = 1	Z	1
CLZ	Нет	Очистить флаг нуля	Z = 0	Z	1
SEI	Нет	Установить флаг прерываний	I = 1	I	1
CLI	Нет	Очистить флаг прерываний	I = 0	I	1
SES	Нет	Установить флаг числа со знаком	S = 1	S	1
CLN	Нет	Очистить флаг числа со знаком	S = 0	S	1
SEV	Нет	Установить флаг переполнения	V = 1	V	1
CLV	Нет	Очистить флаг переполнения	V = 0	V	1
SET	Нет	Установить флаг T	T = 1	T	1
CLT	Нет	Очистить флаг T	T = 0	T	1
SEH	Нет	Установить флаг внутреннего переноса	H = 1	H	1
CLH	Нет	Очистить флаг внутреннего переноса	H = 0	H	1
NOP	Нет	Нет операции	Нет	None	1
SLEEP	Нет	Спать (уменьшить энергопотребление)	Смотрите описание инструкции	None	1
WDR	Нет	Сброс сторожевого таймера	Смотрите описание инструкции	None	1

Ассемблер не различает регистр символов. 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)

1. Команда ADC - Сложить с переносом

Сложение двух регистров и содержимого флага переноса (C), размещение результата в регистре назначения Rd.

(i) Rd <- Rd + Rr + C

Синтаксис	Операнды:	Счетчик программ:
(i) ADC Rd,Rr	0 < d < 31, 0 < r < 31	PC < PC + 1

```
; СложитьR1 : R0 с R3 : R2
add r2, r0 ; Сложить младший байт
adc r3, r1 ; Сложить старший байт с переносом
```

2. Команда ADD - Add without Carry - Сложить без переноса

Сложение двух регистров без добавления содержимого флага переноса (C), размещение результата в регистре назначения Rd.

(i) Rd <- Rd + Rr

Синтаксис	Операнды:	Счетчик программ:
(i) ADD Rd,Rr	0 < d < 31, 0 < r < 31	PC < PC + 1

```
add r1,r2   ; Сложить r2 с r1 (r1=r1+r2)
adc r28,r28 ; Сложить r28 с самим собой
          (r28=r28+r28)
```

3. Команда ADIW - Add Immediate to Word- Сложить непосредственное значение со словом

Сложение непосредственного значения (0-63) с парой регистров и размещение результата в паре регистров. Команда работает с четырьмя верхними парами регистров, удобна для работы с регистрами указателями.

(i) Rdh:Rdl <- Rdh:Rdl + K

Синтаксис	Операнды:	Счетчик программ:
(i) ADIW Rdl,K	dl ∈ {24,26,28,30}, 0 < K < 63	PC < PC + 1

```
      adiw r24, 1  ; Сложить 1 с r25:r24
adiw r30, 63 ; Сложить 63 с Z указателем (r31 : r30)
```

4. Команда AND - Выполнить логическое AND

Выполнение логического AND между содержимым регистров Rd и Rr и помещение результата в регистр назначения Rd.

(i) Rd <- Rd*Rr

Синтаксис	Операнды:	Счетчик программ:
(i) AND Rd,Rr	0 < d < 31,0 < r < 31	PC <- PC + 1

```
   and r2, r3  ; Поразрядное and r2 и r3, результат поместить в r2
ldi r16, 1  ; Установить маску 0000 0001 в r16
and r2, r16 ; Выделить бит 0 в r2
```

5. Команда ANDI - Выполнить логическое AND с непосредственным значением

Выполнение логического AND между содержимым регистра Rd и константой и помещение результата в регистр назначения Rd.

(i) Rd <- Rd*K

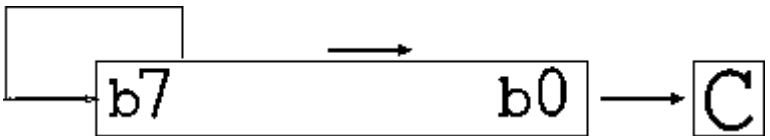
Синтаксис	Операнды:	Счетчик программ:
(i) ANDI Rd,K	16 < d <31, 0 < K <255	PC <- PC + 1

```
andi r17, $0F ; Очистить старший ниббл r17
andi r18, $10 ; Выделить бит 4 в r18
```

andi r19, $AA ; Очистить нечетные биты r19

6. Команда ASR - Арифметически сдвинуть вправо

Выполнение сдвига всех битов Rd на одно место вправо. Состояние бита 7 не изменяется. Бит 0 загружается во флаг переноса (C) регистра состояния (SREG). Эта команда эффективно делит значение дополнения до двух на два, без изменения знака. Флаг переноса может быть использован для округления результата.

(i) 

	Синтаксис	Операнды:	Счетчик программ:
(i)	ASR Rd	0 < d <31	PC <- PC + 1

```
ldi r16, $10 ; Загрузить десятичное значение 16 в r16
asr r16      ; r16=r16 / 2
ldi r17, $FC ; Загрузить -4 в r17
asr r17      ; r17=r17 / 2
```

7. Команда BCLR - Очистить бит в регистре статуса (SREG)

Очистка одного флага в регистре статуса

(i) SREG(s) <- 0

	Синтаксис	Операнды:	Счетчик программ:
(i)	BCLR s	0 < S < 7	PC <- PC + 1

```
bclr 0 ; Очистить флаг переноса
bclr 7 ; Запретить прерывания
```

8. Команда BLD - Загрузить содержимое T флага регистра статуса (SREG) в бит регистра

Копирование содержимого T флага регистра статуса в бит b регистра Rd

(i) Rd(b) <- T

	Синтаксис	Операнды:	Счетчик программ:
(i)	BLD Rd,b	0 < d < 31, 0 < b < 7	PC <- PC + 1

```
; Скопировать бит
bst r1, 0 ; Сохранить бит 2 регистра r1 во флаге T
bld r0, 4 ; Загрузить T в бит 4 регистра r0
```

9. Команда BRBC - Перейти если бит в регистре статуса очищен

Условный относительный переход. Тестируется один из битов регистра статуса и, если бит очищен, выполняется переход относительно состояния счетчика программ. Данная команда выполняет переход в любом направлении относительно состояния счетчика программ (PC-64 < назначение < PC+63). Параметр k является смещением относительно состояния счетчика программ и представлен в форме дополнения до двух.

(i) If SREG(s) = 0 then PC <- PC + k +1, else PC <- PC +1

	Синтаксис	Операнды:	Счетчик программ:
(i)	BRBC s, k	0 < s <7, -64 < k <+63	PC <- PC + k + 1 PC <- PC + 1, если условия не соблюдены

```
cpi r20, 5   ;Сравнить r20 со значением 5
brbc 1,noteq ;Перейти если флаг нуля очищен
.....
noteq:   nop        ;Перейти по назначению(пустая операция)
```

10. Команда BRBS - Перейти если бит в регистре статуса установлен

Условный относительный переход. Тестируется один из битов регистра статуса и, если бит установлен, выполняется переход относительно состояния счетчика программ. Данная команда выполняет переход в любом направлении относительно состояния счетчика программ (PC-64 < назначение < PC+63). Параметр k является смещением относительно счетчика программ и представлен в форме дополнения до двух.

(i) If SREG(s) = 1 then PC <- PC + k +1, else PC <- PC +1

	Синтаксис	Операнды:	Счетчик программ:
(i)	BRBS s, k	0 < s <7, -64 < k <+63	PC <- PC + k + 1 PC <- PC + 1, если условия не соблюдены

```
bst r0, 3    ;Загрузить T битом 3 регистра r0
brbs 6,bitset ;Перейти если бит T установлен
.....
bitset:  nop        ;Перейти по назначению(пустая операция)
```

11. Команда BRCC - Перейти если флаг переноса очищен

Условный относительный переход. Тестируется бит флага переноса (C) регистра статуса и, если бит очищен, выполняется переход относительно состояния счетчика программ. Данная команда выполняет переход в любом направлении относительно состояния счетчика программ (PC-64 < назначение < PC+63). Параметр k является смещением относительно состояния счетчика программ и представлен в форме дополнения до двух. (Команда эквивалентна BRBC 0,k).

(i) If C= 0 then PC <- PC + k +1, else PC <- PC +1

	Синтаксис	Операнды:	Счетчик программ:
(i)	BRCC k	-64 < k <+63	PC <- PC + k + 1 PC <- PC + 1, если условия не соблюдены

```
add r22, r23 ; Сложить r23 с r22
brcc nocarry ; Перейти если перенос очищен
.....
nocarry:  nop        ; Перейти по назначению (пустая операция)
```

12. Команда BRCS - Перейти если флаг переноса установлен

Условный относительный переход. Тестируется бит флага переноса (C) регистра статуса и, если бит установлен, выполняется переход относительно состояния счетчика программ. Данная команда выполняет переход

в любом направлении относительно состояния счетчика программ (PC-64 < назначение < PC+63). Параметр k является смещением относительно состояния счетчика программ и представлен в форме дополнения до двух. (Команда эквивалентна BRBS 0,k).

(i) If C= 1 then PC <- PC + k +1, else PC <- PC +1

	Синтаксис	Операнды:	Счетчик программ:
(i)	BRCS k	-64 < k <+63	PC <- PC + k + 1 PC <- PC + 1, если условия не соблюдены

```
cpi r26, $56 ; Сравнить r26 с $56
brcs carry   ; Перейти если перенос установлен
.....
carry:  nop        ; Перейти по назначению (пустая операция)
```

13. Команда BREQ - Перейти если равно

Условный относительный переход. Тестируется бит флага нулевого значения (Z) регистра статуса и, если бит установлен, выполняется переход относительно состояния счетчика программ. Если команда выполняется непосредственно после выполнения любой из команд CP, CPI, SUB или SUBI переход произойдет если, и только если, двоичное число, со знаком или без знака, представленное в Rd, эквивалентно двоичному числу, со знаком или без знака, представленному в Rr. Данная команда выполняет переход в любом направлении относительно состояния счетчика программ (PC-64 < назначение < PC+63). Параметр k является смещением относительно состояния счетчика программ и представлен в форме дополнения до двух. (Команда эквивалентна BRBS 1,k).

(i) If Rd = Rr (Z = 1) then PC <- PC + k +1, else PC <- PC +1

	Синтаксис	Операнды:	Счетчик программ:
(i)	BREQ k	-64 < k <+63	PC <- PC + k + 1 PC <- PC + 1, если условия не соблюдены

```
cp r1, r0  ; Сравнить регистры r1 и r0
breq equal ; Перейти если содержимое регистров совпадает
.....
equal:  nop        ; Перейти по назначению (пустая операция)
```

14. Команда BRGE - Перейти если больше или равно (с учетом знака)

Условный относительный переход. Тестируется бит флага знака (S) регистра статуса и, если бит очищен, выполняется переход относительно состояния счетчика программ. Если команда выполняется непосредственно после выполнения любой из команд CP, CPI, SUB или SUBI переход произойдет если, и только если, двоичное число, со знаком представленное в Rd, больше или эквивалентно двоичному числу со знаком, представленному в Rr. Данная команда выполняет переход в любом направлении относительно состояния счетчика программ (PC-64 < назначение < PC+63). Параметр k является смещением относительно состояния счетчика программ и представлен в форме дополнения до двух. (Команда эквивалентна BRBC 4,k).

(i) If Rd > Rr (NEV = 0) then PC <- PC + k +1, else PC <- PC +1

	Синтаксис	Операнды:	Счетчик программ:
(i)	BRGE k	-64 < k <+63	PC <- PC + k + 1 PC <- PC + 1, если условия не соблюдены

```
cp r11, r12  ; Сравнить регистры r11 и r12
brge greateq ; Перейти если r11 >= r12 (со знаком)
.....
```

greateq: nop ; Перейти по назначению (пустая операция)

15. Команда BRHC - Перейти если флаг полупереноса очищен

Условный относительный переход. Тестируется бит флага полупереноса (H) регистра статуса и, если бит очищен, выполняется переход относительно состояния счетчика программ. Данная команда выполняет переход в любом направлении относительно состояния счетчика программ (PC-64 < назначение < PC+63). Параметр k является смещением относительно состояния счетчика программ и представлен в форме дополнения до двух. (Команда эквивалентна BRBC 5,k).

(i) If H = 0 then PC <- PC + k +1, else PC <- PC +1

	Синтаксис	Операнды:	Счетчик программ:
(i)	BRHC k	-64 < k <+63	PC <- PC + k + 1 PC <- PC + 1, если условия не соблюдены

```
brhc hclear ; Перейти если флаг полупереноса очищен
.....
hclear:  nop        ; Перейти по назначению (пустая операция)
```

16. Команда BRHS - Перейти если флаг полупереноса установлен

Условный относительный переход. Тестируется бит флага полупереноса (H) регистра статуса и, если бит установлен, выполняется переход относительно состояния счетчика программ. Данная команда выполняет переход в любом направлении относительно состояния счетчика программ (PC-64 < назначение < PC+63). Параметр k является смещением относительно состояния счетчика программ и представлен в форме дополнения до двух. (Команда эквивалентна BRBS 5,k).

(i) If H = 1 then PC <- PC + k +1, else PC <- PC +1

	Синтаксис	Операнды:	Счетчик программ:
(i)	BRHS k	-64 < k <+63	PC <- PC + k + 1 PC <- PC + 1, если условия не соблюдены

```
brhs hset ; Перейти если флаг полупереноса установлен
.....
hset:    nop      ; Перейти по назначению (пустая операция)
```

17. Команда BRID - Перейти если глобальное прерывание запрещено

Условный относительный переход. Тестируется бит флага глобального прерывания (I) регистра статуса и, если бит сброшен, выполняется переход относительно состояния счетчика программ. Данная команда выполняет переход в любом направлении относительно состояния счетчика программ (PC-64 << назначение < PC+63). Параметр k является смещением относительно состояния счетчика программ и представлен в форме дополнения до двух. (Команда эквивалентна BRBC 7,k).

(i) If I = 0 then PC <- PC + k +1, else PC <- PC +1

	Синтаксис	Операнды:	Счетчик программ:
(i)	BRID k	-64 < k <+63	PC <- PC + k + 1 PC <- PC + 1, если условия не соблюдены

```
brid intdis ; Перейти если глобальное прерывание запрещено
.....
intdis:  nop        ; Перейти по назначению (пустая операция)
```

18. Команда BRIE - Перейти если глобальное прерывание разрешено

Условный относительный переход. Тестируется бит флага глобального прерывания (I) регистра статуса и, если бит установлен, выполняется переход относительно состояния счетчика программ. Данная команда выполняет переход в любом направлении относительно состояния счетчика программ (PC64 < назначение < PC+63). Параметр k является смещением относительно состояния счетчика программ и представлен в форме дополнения до двух. (Команда эквивалентна BRBS 7,k).

(i) If I = 1 then PC <- PC + k +1, else PC <- PC +1

	Синтаксис	Операнды:	Счетчик программ:
(i)	BRIE k	-64 < k <+63	PC <- PC + k + 1 PC <- PC + 1, если условия не соблюдены

```
brie inten ;Перейти если глобальное прерывание разрешено
.....
inten:   nop       ;Перейти по назначению (пустая операция)
```

19. Команда BRLO - Перейти если меньше (без знака)

Условный относительный переход. Тестируется бит флага переноса (C) регистра статуса и, если бит установлен, выполняется переход относительно состояния счетчика программ. Если команда выполняется непосредственно после выполнения любой из команд CP, CPI, SUB или SUBI переход произойдет если, и только если, двоичное число без знака, представленное в Rd, меньше двоичного числа без знака, представленного в Rr Данная команда выполняет переход в любом направлении относительно состояния счетчика программ (PC-64 < назначение < PC+63). Параметр k является смещением относительно состояния счетчика программ и представлен в форме дополнения до двух. (Команда эквивалентна BRBS 0,k).

(i) If Rd < Rr (C = 1) then PC <- PC + k +1, else PC <- PC +1

	Синтаксис	Операнды:	Счетчик программ:
(i)	BRLO k	-64 < k <+63	PC <- PC + k + 1 PC <- PC + 1, если условия не соблюдены

```
eor r19, r19 ; Очистить r19
loop:   inc r19     ; Увеличить на 1 r19
.....
cpi r19, $10 ; Сравнить r19 с $10
brlo loop    ; Перейти если r19 < $10 (без знака)
nop          ; Выйти из петли (пустая операция)
```

20. Команда BRLT - Перейти если меньше чем (со знаком)

Условный относительный переход. Тестируется бит флага знака (S) регистра статуса и, если бит установлен, выполняется переход относительно состояния счетчика программ. Если команда выполняется непосредственно после выполнения любой из команд CP, CPI, SUB или SUBI переход произойдет если, и только если, двоичное число со знаком, представленное в Rd, меньше двоичного числа со знаком, представленного в Rr. Данная команда выполняет переход в любом направлении относительно состояния счетчика программ (PC-64 < назначение < PC+63). Параметр k является смещением относительно состояния счетчика программ и представлен в форме дополнения до двух. (Команда эквивалентна BRBS 4,k).

(i) If Rd < Rr (NEV = 1) then PC <- PC + k +1, else PC <- PC +1

	Синтаксис	Операнды:	Счетчик программ:
(i)	BRLT k	-64 < k <+63	PC <- PC + k + 1 PC <- PC + 1, если условия не соблюдены

```
cp r16, r1 ; Сравнить r16 с r1
brlt less  ; Перейти если r16 < r1 (со знаком)
.....
less nop   ; Перейти по назначению (пустая операция)
```

21. Команда BRMI - Перейти если минус

Условный относительный переход. Тестируется бит флага отрицательного значения (N) регистра статуса и, если бит установлен, выполняется переход относительно состояния счетчика программ. Данная команда выполняет переход в любом направлении относительно состояния счетчика программ (PC64 < назначение < PC+63). Параметр k является смещением относительно состояния счетчика программ и представлен в форме дополнения до двух. (Команда эквивалентна BRBS r, k).

(i) If N = 1 then then PC <- PC + k +1, else PC <- PC +1

	Синтаксис	Операнды:	Счетчик программ:
(i)	BRMI k	-64 < k <+63	PC <- PC + k + 1 PC <- PC + 1, если условия не соблюдены

```
subi r18, 4   ; Вычесть 4 из r18
brmi negative ; Перейти если результат отрицательный
.....
negative: nop         ; Перейти по назначению (пустая операция)
```

22. Команда BRNE - Перейти если не равно

Условный относительный переход. Тестируется бит флага нулевого значения (Z) регистра статуса и, если бит очищен, выполняется переход относительно состояния счетчика программ. Если команда выполняется непосредственно после выполнения любой из команд CP, CPI, SUB или SUBI переход произойдет если, и только если, двоичное число со знаком или без знака, представленное в Rd, не равно двоичному числу со знаком или без знака, представленному в Rr Данная команда выполняет переход в любом направлении относительно состояния счетчика программ (PC-64 < назначение < PC+63). Параметр k является смещением относительно состояния счетчика программ и представлен в форме дополнения до двух. (Команда эквивалентна BRBC 1,k).

(i) If Rd ≠ Rr (Z = 0) then then PC <- PC + k +1, else PC <- PC +1

	Синтаксис	Операнды:	Счетчик программ:
(i)	BRNE k	-64 < k <+63	PC <- PC + k + 1 PC <- PC + 1, если условия не соблюдены

```
eor r27, r27 ; Очистить r27
loop:   inc r27     ; Увеличить на 1 r27
```

```
.....
cpi r27, 5   ; Сравнить r27 с 5
brne loop    ; Перейти если r27 <> 5
nop          ; Выйти из петли (пустая операция)
```

23. Команда BRPL - Перейти если плюс

Условный относительный переход. Тестируется бит флага отрицательного значения (N) регистра статуса и, если бит очищен, выполняется переход относительно состояния счетчика программ. Данная команда выполняет переход в любом направлении относительно состояния счетчика программ (PC-64 < назначение < PC+63). Параметр k является смещением относительно состояния счетчика программ и представлен в форме дополнения до двух. (Команда эквивалентна BRBC 2,k).

(i) If N = 0 then then PC <- PC + k +1, else PC <- PC +1

	Синтаксис	Операнды:	Счетчик программ:
(i)	BRPL k	-64 < k <+63	PC <- PC + k + 1 PC <- PC + 1, если условия не соблюдены

```
subi r26, $50 ; Вычесть $50 из r26
brpl positive ; Перейти если r26 положителен
.....
positive: nop          ; Перейти по назначению (пустая операция)
```

24. Команда BRSH - Перейти если равно или больше (без знака)

Условный относительный переход. Тестируется бит флага перехода (C) регистра статуса и, если бит очищен, выполняется переход относительно состояния счетчика программ. Если команда выполняется непосредственно после выполнения любой из команд CP, CPI, SUB или SUBI переход произойдет если, и только если, двоичное число без знака, представленное в Rd, больше или равно двоичному числу без знака, представленному в Rr Данная команда выполняет переход в любом направлении относительно состояния счетчика программ (PC-64 < назначение < PC+63). Параметр k является смещением относительно состояния счетчика программ и представлен в форме дополнения до двух. (Команда эквивалентна BRBC 0,k).

(i) If Rd > Rr (C = 0) then then PC <- PC + k +1, else PC <- PC +1

	Синтаксис	Операнды:	Счетчик программ:
(i)	BRSH k	-64 < k <+63	PC <- PC + k + 1 PC <- PC + 1, если условия не соблюдены

```
subi r19, 4 ; Вычесть 4 из r19
brsh highsm ; Перейти если r2 >= 4 (без знака)
.....
highsm:   nop        ; Перейти по назначению (пустая операция)
```

25. Команда BRTC - Перейти если флаг T очищен

Условный относительный переход. Тестируется бит флага пересылки (T) регистра статуса и, если бит очищен, выполняется переход относительно состояния счетчика программ. Данная команда выполняет переход в любом направлении относительно состояния счетчика программ (PC-64 < назначение < PC+63). Параметр k является смещением относительно состояния счетчика программ и представлен в форме дополнения до двух. (Команда эквивалентна BRBC 6,k).

(i) If T = 0 then then PC <- PC + k +1, else PC <- PC +1

	Синтаксис	Операнды:	Счетчик программ:
(i)	BRTC k	-64 < k <+63	PC <- PC + k + 1 PC <- PC + 1, если условия не соблюдены

```
bst r3, 5   ; Сохранить бит 5 регистра r3 во флаге T
brtc tclear ; Перейти если этот бит очищен
.....
tclear:  nop        ; Перейти по назначению (пустая операция)
```

26. Команда BRTS - Перейти если флаг T установлен

Условный относительный переход. Тестируется бит флага пересылки (T) регистра статуса и, если бит установлен, выполняется переход относительно состояния счетчика программ. Данная команда выполняет переход в любом направлении относительно состояния счетчика программ (PC-64 < назначение < PC+63). Параметр k является смещением относительно состояния счетчика программ и представлен в форме дополнения до двух. (Команда эквивалентна BRBC 6,k).

(i) If T = 1 then then PC <- PC + k +1, else PC <- PC +1

	Синтаксис	Операнды:	Счетчик программ:
(i)	BRTS k	-64 < k <+63	PC <- PC + k + 1 PC <- PC + 1, если условия не соблюдены

```
bst r3, 5 ; Сохранить бит 5 регистра r3 во флаге T
brts tset ; Перейти если этот бит установлен
.....
tset:   nop      ; Перейти по назначению (пустая операция)
```

27. Команда BRVC - Перейти если переполнение очищено

Условный относительный переход. Тестируется бит флага переполнения (V) регистра статуса и, если бит очищен, выполняется переход относительно состояния счетчика программ. Данная команда выполняет переход в любом направлении относительно состояния счетчика программ (PC-64 < назначение < PC+63). Параметр k является смещением относительно состояния счетчика программ и представлен в форме дополнения до двух. (Команда эквивалентна BRBC 3,k).

(i) If V = 0 then then PC <- PC + k +1, else PC <- PC +1

	Синтаксис	Операнды:	Счетчик программ:
(i)	BRVC k	-64 < k <+63	PC <- PC + k + 1 PC <- PC + 1, если условия не соблюдены

```
add r3, r4  ; Сложить r4 с r3
brvc noover ; Перейти если нет переполнения
.....
noover:  nop        ; Перейти по назначению (пустая операция)
```

28. Команда BRVS - Перейти если переполнение установлено

Условный относительный переход. Тестируется бит флага переполнения (V) регистра статуса и, если бит установлен, выполняется переход относительно состояния счетчика программ. Данная команда выполняет переход в любом направлении относительно состояния счетчика программ (PC-64 < назначение < PC+63). Параметр k является смещением относительно состояния счетчика программ и представлен в форме дополнения до двух. (Команда эквивалентна BRBC 3,k).

(i) If V = 1 then then PC <- PC + k +1, else PC <- PC +1

	Синтаксис	Операнды:	Счетчик программ:
(i)	BRVS k	-64 < k <+63	PC <- PC + k + 1 PC <- PC + 1, если условия не соблюдены

```
add r3, r4  ; Сложит r4 с r3
brvs overfl ; Перейти если есть переполнение
.....
overfl:   nop        ; Перейти по назначению (пустая операция)
```

29. Команда BSET - Установить бит в регистре статуса (SREG)

Установка одного флага в регистре статуса.

(i) SREG(s)<-- 1

	Синтаксис	Операнды:	Счетчик программ:
(i)	BSET s	0 < s < 7	PC <- PC + 1

```
bset 6 ; Установить флаг T
bset 7 ; Разрешить прерывание
```

30. Команда BST - Переписать бит из регистра во флаг T регистра статуса (SREG)

Перезапись бита b из регистра Rd в флаг T регистра статуса (SREG)

(i) T <-- Rd(b)

	Синтаксис	Операнды:	Счетчик программ:
(i)	BST Rd,b	0 < d < 31, 0 < b < 7	PC <- PC + 1

```
; Копировать бит
bst r1, 2 ; Сохранить бит 2 регистра r1 во флаге T
bld r0, 4 ; Загрузить T в бит 4регистра r0
```

31. Команда CBI - Очистить бит в регистре I/O

Очистка определенного бита в регистре ввода/вывода. Команда работает с младшими 32 регистрами ввода/вывода - адреса с 0 по 31.

(i) I/O(P,b)<-- 0

	Синтаксис	Операнды:	Счетчик программ:
(i)	CBI P,b	0 < P < 31, 0 < b < 7	PC <-- PC + 1

```
cbi $12. 7 ; Очистить бит 7 в Порте D
```

32. Команда CBR - очистить биты в регистре

Очистка определенных битов регистра Rd. Выполняется логическое AND между содержимым регистра Rd и комплементом постоянной K

(i) Rd <-- Rd * ($FF - K)

	Синтаксис	Операнды:	Счетчик программ:
(i)	CBR Rd	16 < d < 31, 0 < K <255	PC <- PC + 1

```
cbr r16, $F0 ; Очистить старший ниббл регистра r16
cbr r18, 1   ; Очистить бит в r18
```

33. Команда CLC - очистить флаг переноса в регистре статуса (SREG)

Очистка флага переноса (C) в регистре статуса (SREG)

(i) C <-- 0

	Синтаксис	Операнды:	Счетчик программ:
(i)	CLC	None	PC <- PC + 1

```
add r0, r0 ; Сложить r0 с самим собой
clc        ; Очистить флаг переноса
```

34. Команда CLH - очистить флаг полупереноса в регистре статуса (SREG)

Очистка флага полупереноса (H) в регистре статуса (SREG).

(i) H <-- 0

	Синтаксис	Операнды:	Счетчик программ:
(i)	CLH	None	PC <- PC + 1

```
clh ; Очистить флаг полупереноса
```

35. Команда CLI - очистить флаг глобального прерывания в регистре статуса (SREG)

Очистка флага глобального прерывания (I) в регистре статуса (SREG).

(i) I <-- 0

	Синтаксис	Операнды:	Счетчик программ:
(i)	CLI	None	PC <- PC + 1

```
cli        ; Запретить прерывания
in r11, $16 ; Считать Порт B
sei        ; Разрешить прерывания
```

36. Команда CLN - очистить флаг отрицательного значения в регистре статуса (SREG)

Очистка флага отрицательного значения (N) в регистре статуса (SREG).

(i) N <-- 0

	Синтаксис	Операнды:	Счетчик программ:
(i)	CLN	None	PC <- PC + 1

```
add  r2, r3 ; Сложить r3 с r2
cln         ; Очистить флаг отрицательного значения
```

37. Команда CLR - очистить регистр

Очистка регистра. Команда выполняет Exclusive OR содержимого регистра с самим собой. Это приводит к очистке всех битов регистра.

(i) Rd <-- Rd ⊕Rd

	Синтаксис	Операнды:	Счетчик программ:
(i)	CLR Rd	0 < d < 31	PC <- PC + 1

```
clr r18      ; Очистить r18
loop:   inc r18      ; Увеличить на 1 r18
. . .
cpi r18, $50 ; Сравнить r18 с $50
brne loop
```

38. Команда CLS - очистить флаг знака

Очистка флага знака (S) в регистре статуса (SREG).

(i) S <-- 0

	Синтаксис	Операнды:	Счетчик программ:
(i)	CLS	None	PC <- PC + 1

```
add r2, r3 ; Сложить r3 с r2
cls        ; Очистить флаг знака
```

39. Команда CLT - очистить T флаг

Очистка флага пересылки (T) в регистре статуса (SREG).

(i) T <-- 0

	Синтаксис	Операнды:	Счетчик программ:
(i)	CLT	None	PC <- PC + 1

```
clt ; Очистить T флаг
```

40. Команда CLV - очистить флаг переполнения

Очистка флага переполнения (V) в регистре статуса (SREG).

(i) V <-- 0

	Синтаксис	Операнды:	Счетчик программ:
(i)	CLV	None	PC <- PC + 1

```
add   r2, r3 ; Сложить r3 с r2
clv          ; Очистить флаг переполнения
```

41. Команда CLZ - очистить флаг нулевого значения

Очистка флага нулевого значения (Z) в регистре статуса (SREG).

(i) Z <-- 0

	Синтаксис	Операнды:	Счетчик программ:
(i)	CLZ	None	PC <- PC + 1

```
add   r2, r3 ; Сложить r3 с r2
clz          ; Очистить флаг нулевого значения
```

42. Команда COM - Выполнить дополнение до единицы

Команда выполняет дополнение до единицы (реализует обратный код) содержимого регистра Rd.

(i) Rd <-- $FF * Rd

	Синтаксис	Операнды:	Счетчик программ:
(i)	COM Rd	0 < d < 31	PC <- PC + 1

```
com r4    ; Выполнить дополнение до единицы r4
breq zero ; Перейти если ноль
. . .
zero:   nop       ; Перейти по назначению (пустая операция)
```

43. Команда CP - сравнить

Команда выполняет сравнение содержимого двух регистров Rd и Rr. Содержимое регистров не изменяется. После этой команды можно выполнять любые условные переходы.

(i) Rd = Rr

	Синтаксис	Операнды:	Счетчик программ:
(i)	Cp Rd, Rr	0 < d < 31,0 < r < 31	PC <- PC + 1

```
cp r4, r19 ; Сравнить r4 с r19
brne noteq ; Перейти если r4 <> r19
. . .
noteq:   nop       ; Перейти по назначению (пустая операция)
```

44. Команда CPC - сравнить с учетом переноса

Команда выполняет сравнение содержимого двух регистров Rd и Rr и учитывает также предшествовавший перенос. Содержимое регистров не изменяется. После этой команды можно выполнять любые условные переходы.

(i) Rd = Rr = C

	Синтаксис	Операнды:	Счетчик программ:
(i)	CPC Rd, Rr	0 < d < 31,0 < r < 31	PC <- PC + 1

```
; Сравнить r3 : r2 с r1 : r0
cp r2, r0  ; Сравнить старший байт
cpc r3, r1 ; Сравнить младший байт
brne noteq ; Перейти если не равно
. . .
noteq:   nop       ; Перейти по назначению (пустая операция)
```

45. Команда CPI - сравнить с константой

Команда выполняет сравнение содержимого регистра Rd с константой. Содержимое регистра не изменяется. После этой команды можно выполнять любые условные переходы.

(i) Rd = K

	Синтаксис	Операнды:	Счетчик программ:
(i)	CPI Rd, K	0 < d < 31,0 < K <255	PC <- PC + 1

```
cpi r19, 3 ; Сравнить r19 с 3
brne error ; Перейти если r4 <> 3
. . .
```

```
error:   nop      ; Перейти по назначению (пустая операция)
```

46. Команда CPSE - сравнить и пропустить если равно

Команда выполняет сравнение содержимого регистров Rd и Rr и пропускает следующую команду если Rd = Rr.

(i) If Rd = Rr then PC <-- PC + 2 (or 3), else PC <-- PC +1

	Синтаксис	Операнды:	Счетчик программ:
(i)	CPSE Rd,Rr	0 < d <31, 0 < r<31	PC<-- PC + 1, если условия не соблюдены, то пропуска нет PC<-- PC + 2, пропуск одного слова команды

```
inc r4      ; Увеличить на 1 r4
cpse r4, r0  ; Сравнить r4 с r0
neg r4      ; Выполнить если r4 <> r0
nop         ; Продолжать (пустая операция)
```

47. Команда DEC - декрементировать

Вычитание единицы - 1 - из содержимого регистра Rd и размещение результата в регистре назначения Rd. Флаг переноса регистра статуса данной командой не активируется, что позволяет использовать команду DEC использовать при реализации счетчика циклов для вычислений с повышенной точностью. При обработке чисел без знаков за командой могут выполняться переходы BREQ и BRNE. При обработке значений в форме дополнения до двух допустимы все учитывающие знак переходы.

(i) Rd<-- Rd - 1

	Синтаксис	Операнды:	Счетчик программ:
(i)	DEC Rd	0 < d < 31	PC <- PC + 1

```
ldi r17, $10 ; Загрузить константу в r17
loop:    add r1, r2   ; Сложить r2 с r1
dec r17      ; Уменьшить на 1 r17
brne loop    ; Перейти если r17 <> 0
nop          ; Продолжать (пустая операция)
```

48. Команда EOR - выполнить исключающее OR

Выполнение логического исключающего OR между содержимым регистра Rd и регистром Rr и помещение результата в регистр назначения Rd.

(i) Rd<-- Rd⊕Rr

	Синтаксис	Операнды:	Счетчик программ:
(i)	EOR Rd,Rr	0 < d < 31,0 < r < 31	PC <- PC + 1

```
eor r4, r4  ; Очистить r4
eor r0, r22 ; Поразрядно выполнить исключающее or между r0 и r22
```

49. Команда ICALL - вызвать подпрограмму косвенно

Косвенный вызов подпрограммы указанной регистром-указателем Z (16 разрядов) в регистровом файле. Регистр-указатель Z (16-разрядного формата) позволяет вызвать подпрограмму из текущей секции пространства памяти программ объемом 64К слов (128 Кбайт).

(i) PC(15-0)<-- Z(15-0) Приборы с 16-разрядным счетчиком программ, максимальный объем памяти программ 128К.

	Синтаксис	Операнды:	Счетчик программ:	Стек
(i)	ICALL	None	См. Операция	STACK<-- PC + 1 SP<-- SP-2, (2 байта, 16 битов)

```
mov r30, r0 ; Установить смещение в таблицу вызовов
icall       ; Вызвать подпрограмму указанную r31 : r30
```

50. Команда IJMP - перейти косвенно

Выполняется косвенный переход по адресу указанному регистром-указателем Z (16 разрядов) в регистровом файле. Регистр-указатель Z (16-разрядного формата) позволяет вызвать подпрограмму из текущей секции пространства памяти программ объемом 64К слов (128 Кбайт).

(i) PC<-- Z(15-0) Приборы с 16-разрядным счетчиком программ, максимальный объем памяти программ 128К.

```
mov    r30, r0 ; Установить смещение в таблицу переходов
ijmp           ; Перейти к подпрограмме указанной r31 : r30
```

51. Команда IN - загрузить данные из порта I/O в регистр

Команда загружает данные из пространства входа/выхода (порты, таймеры, регистры конфигурации и т.п.) в регистр Rd регистрового файла.

(i) Pd<-- P

	Синтаксис	Операнды:	Счетчик программ:
(i)	IN Rd,P	0 < d < 31, 0 < P < 63	PC<-- PC + 1

```
in r25, $16 ; Считать Порт B
cpi r25, r4 ; Сравнить считанное значение с константой
breq exit   ; Перейти если r25=4
. . .
exit:  nop        ; Перейти по назначению (пустая операция)
```

Команда INC - инкрементировать
Добавление единицы - 1 - к содержимому регистра Rd и размещение результата в регистре назначения Rd.Флаг переноса регистра статуса данной командой не активируется, что позволяет использовать команду DEC использовать при реализации счетчика циклов для вычислений с повышенной точностью. При обработке чисел без знаков за командой могут выполняться переходы BREQ и BRNE. При обработке значений в форме дополнения до двух допустимы все учитывающие знак переходы.

(i) Rd<-- Rd + 1

	Синтаксис	Операнды:	Счетчик программ:
(i)	INC Rd	0 < d < 31	PC <- PC + 1

```
clr r22     ; Очистить r22
loop:  inc r22     ; Увеличить на 1 r22
. . .
cpi r22, $4F ; Сравнить r22 с $4F
brne loop    ; Перейти если не равно
nop          ; Продолжать (пустая операция)
```

52. Команда LD - загрузить косвенно из СОЗУ в регистр с использованием индекса X

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. 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. Использование регистра-указателя X обеспечивает удобную возможность обращения к матрицам, таблицам, указателю стека.

Использование X-указателя:

	Операция:	Комментарий:	
(i)	Rd <-- (X)		X: Неизменен
(ii)	Rd <-- (X)	X <-- X + 1	X: Инкрементирован впоследствии
(iii)	X <-- X - 1	Rd <-- (X)	X: Предварительно декрементирован

	Синтаксис	Операнды:	Счетчик программ:
(i)	LD Rd,X	0 < d < 31	PC<-- + 1
(ii)	LD Rd,X+	0 < d < 31	PC<-- + 1
(iii)	LDD Rd,-X	0 < d < 31	PC<-- + 1

```
clr   r27     ;Очистить старший байт X
ldi   r26, $20 ;Установить $20 в младший байт X
ld    r0, X+  ;Загрузить в r0 содержимое SRAM по адресу $20 (X постинкрементируется)
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)
```

53. Команда LD (LDD) - загрузить косвенно из СОЗУ в регистр с использованием индекса Y

Загружает косвенно, со смещением или без смещения, один байт из СОЗУ в регистр. Положение байта в СОЗУ указывается 16-разрядным регистром-указателем Y в регистровом файле. Обращение к памяти ограничено текущей страницей объемом 64 Кбайта. Для обращения к другой странице СОЗУ необходимо изменить регистр RAMPY в I/O области. Регистр-указатель Y может остаться неизменным после выполнения команды, но может быть инкрементирован или декрементирован. Использование регистра-указателя Y обеспечивает удобную возможность обращения к матрицам, таблицам, указателю стека.

Использование Y-указателя:

	Операция:	Комментарий:	
(i)	Rd <-- (Y)		Y: Неизменен
(ii)	Rd <-- (Y)	Y <-- Y + 1	Y: Инкрементирован впоследствии
(iii)	Y <-- Y + 1	Rd <-- (Y)	Y: Предварительно декрементирован
(iv)	Rd <-- (Y + q)		Y: Неизменен, q: смещение

	Синтаксис	Операнды:	Счетчик программ:
(i)	LD Rd,Y	0 < d < 31	PC<-- + 1
(ii)	LD Rd,Y+	0 < d < 31	PC<-- + 1
(iii)	LD Rd,-Y	0 < d < 31	PC<-- + 1
(iv)	LDD Rd, Y + q	0 < d < 31 0 < q < 63	PC<-- + 1

```
clr  r29      ;Очистить старший байт Y
ldi  r28, $20  ;Установить $20 в младший байт Y
ld   r0, Y+    ;Загрузить в r0 содерж. SRAM по адресу $20 (Y постинкрементируется)
ld   r1, Y     ;Загрузить в r1 содержимое SRAM по адресу $21
ldi  r28, $23  ;Установить $23 в младший байт Y
ld   r2, Y     ;Загрузить в r2 содержимое SRAM по адресу $23
ld   r3, -Y    ;Загрузить в r3 содерж. SRAM по адресу $22 (Y преддекрементируется)
ldd  r4, Y+2   ;Загрузить в r4 содержимое SRAM по адресу $24
```

54. Команда LD (LDD) - загрузить косвенно из СОЗУ в регистр с использованием индекса Z

Загружает косвенно, со смещением или без смещения, один байт из СОЗУ в регистр. Положение байта в СОЗУ указывается 16-разрядным регистром-указателем Z в регистровом файле. Обращение к памяти ограничено текущей страницей объемом 64 Кбайта. Для обращения к другой странице СОЗУ необходимо изменить регистр RAMPZ в I/O области. Регистр-указатель Z может остаться неизменным после выполнения команды, но может быть инкрементирован или декрементирован. Эта особенность очень удобна при использовании регистра-указателя Z в качестве указателя стека, однако, поскольку регистр-указатель Z может быть использован для косвенного вызова подпрограмм, косвенных переходов и табличных преобразований, более удобно использовать в качестве указателя стека регистры-указатели X и Y. Об использовании указателя Z для просмотра таблиц в памяти программ см. команду LPM.

Использование Z-указателя:

	Операция:	Комментарий:	
(i)	Rd <-- (Y)		Y: Неизменен
(ii)	Rd <-- (Y)	Y <-- Y + 1	Y: Инкрементирован впоследствии
(iii)	Y <-- Y + 1	Rd <-- (Y)	Y: Предварительно декрементирован
(iv)	Rd <-- (Y + q)		Y: Неизменен, q: смещение

	Синтаксис	Операнды:	Счетчик программ:
(i)	LD Rd,Y	0 < d < 31	PC<-- + 1
(ii)	LD Rd,Y+	0 < d < 31	PC<-- + 1
(iii)	LD Rd,-Y	0 < d < 31	PC<-- + 1
(iv)	LDD Rd, Y + q	0 < d < 31 0 < q < 63	PC<-- + 1

```
clr r31      ; Очистить старший байт Z
ldi r30, $F0  ; Установить $F0 в младший байт Z
lpm          ; Загрузить константу из программы
; Память отмечена в Z
```

55. Команда LDS - загрузить непосредственно из СОЗУ

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

(i) Rd <-- (k)

	Синтаксис	Операнды:	Счетчик программ:
(i)	LDS Rd,k	0 < d < 31,0 < k < 65535	PC<-- + 2

```
lds  r2, $FF00 ; Загрузить r2 содержимым SRAM по адресу $FF00
add  r2, r1    ; Сложить r1 с r2
sts  $FF00, r2 ; Записать обратно
```

56. Команда LPM - загрузить байт памяти программ

Загружает один байт, адресованный регистром Z, в регистр 0 (R0). Команда обеспечивает эффективную загрузку констант или выборку постоянных данных. Память программ организована из 16-разрядных слов и младший значащий разряд (LSB) 16-разрядного указателя Z выбирает или младший (0) или старший (1) байт. Команда может адресовать первые 64 Кбайта (32 Кслов) памяти программ.

	Операция:	Комментарий:
(i)	R0<-- (Z)	Z указывает на память программ

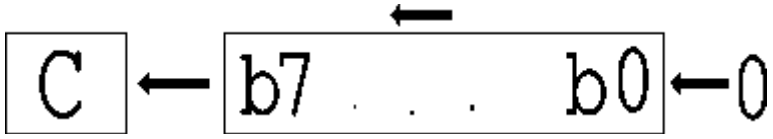
	Синтаксис	Операнды:	Счетчик программ:
(i)	LPM	None	PC<-- + 1

```
clr r31      ; Очистить старший байт Z
ldi r30, $F0 ; Установить младший байт Z
lpm          ; Загрузить константу из памяти программ
отмеченную Z (r31 : r30)
```

57. Команда LSL - логически сдвинуть влево

Выполнение сдвига всех битов Rd на одно место влево. Бит 0 стирается. Бит 7 загружается во флаг переноса (C) регистра состояния (SREG). Эта команда эффективно умножает на два значение величины без знака.

(i)



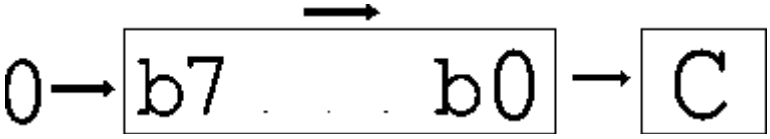
Синтаксис	Операнды:	Счетчик программ:
(i) LSL Rd	0 < d < 31	PC <-- PC + 1

```
add r0, r4 ; Сложить r4 с r0
lsl r0     ; Умножить r0 на 2
```

58. Команда LSR - логически сдвинуть вправо

Сдвиг всех битов Rd на одно место вправо. Бит 7очищается. Бит 0 загружается во флаг переноса (C) регистра состояния (SREG). Эта команда эффективно делит на два величину без знака на два. Флаг переноса может быть использован для округления результата.

(i)



Синтаксис	Операнды:	Счетчик программ:
(i) LSR Rd	0 < d < 31	PC <- PC + 1

```
add r0, r4 ; Сложить r4 с r0
lsr r0     ; Разделить r0 на 2
```

59. Команда MOV - копировать регистр

Команда создает копию одного регистра в другом регистре. Исходный регистр Rr остается неизменным, в регистр назначения Rd загружается копия содержимого регистра Rr.

(i) Rd <-- Rr

Синтаксис	Операнды:	Счетчик программ:
(i) MOV Rd,Rr	0 < d < 31, 0 < r <31	PC<-- + 1

```
mov   r16, r0     ; Копировать r0 в r16
call  check       ; Вызвать подпрограмму
. . .
check cpi r16, $11 ; Сравнить r16 с $11
. . .
ret               ; Вернуться из подпрограммы
```

60. Команда NEG - выполнить дополнение до двух

Заменяет содержимое регистра Rd его дополнением до двух. Значение $80 остается неизменным.

(i) Rd <-- $00 - Rd

Синтаксис	Операнды:	Счетчик программ:
(i) NEG Rd	0 < d < 31	PC <- PC + 1

```
sub  r11, r0  ; Вычесть r0 из r11
brpl positive ; Перейти если результат положительный
neg  r11      ; Выполнить дополнение до двух r11
positive:  nop          ; Перейти по назначению (пустая операция)
```

61. Команда NOP - выполнить холостую команду

Команда выполняется за один цикл без выполнения операции.

(i) No

Синтаксис	Операнды:	Счетчик программ:
(i) NOP	None	PC<-- + 1

```
clr  r16      ; Очистить r16
ser  r17      ; Установить r17
out  $18, r16 ; Записать ноль в Порт B
nop           ; Ожидать (пустая операция)
out  $18, r17 ; Записать 1 в Порт B
```

62. Команда OR - выполнить логическое OR

Команда выполняет логическое OR содержимого регистров Rd и Rr и размещает результат в регистре назначения Rd.

(i) Rd <-- Rd v Rr

Синтаксис	Операнды:	Счетчик программ:
(i) OR Rd,Rr	0 < d < 31,0 < r <31	PC <- PC + 1

```
or r15, r16  ; Выполнить поразрядное or между регистрами
bst r15, 6   ; Сохранить бит 6 регистра 15 во флаге T
brst ok      ; Перейти если флаг T установлен
. . .
ok:  nop          ; Перейти по назначению (пустая операция)
```

63. Команда ORI - выполнить логическое OR с непосредственным значением

Команда выполняет логическое OR между содержимым регистра Rd и константой и размещает результат в регистре назначения Rd.

(i) Rd <-- Rd v K

Синтаксис	Операнды:	Счетчик программ:
(i) ORI Rd,K	16 < d < 31,0 < K <255	PC <- PC + 1

```
ori  r16, $F0 ; Установить старший ниббл r16
ori  r17, 1   ; Установить бит 0 регистра r17
```

64. Команда OUT - записать данные из регистра в порт I/O

Команда сохраняет данные регистра Rr в регистровом файле пространства I/O (порты, таймеры, регистры конфигурации и т.п.).

(i) P <-- Rr

Синтаксис	Операнды:	Счетчик программ:
(i) OUT P,Rr	0 < r <31, 0 < P <63	PC<-- + 1

```
clr r16      ; Очистить r16
ser r17      ; Установить r17
out $18, r16 ; Записать нули в Порт B
nop          ; Ожидать (пустая операция)
out $18, r17 ; Записать единицы в Порт B
```

65. Команда POP - записать регистр из стека

Команда загружает регистр Rd байтом содержимого стека.

(i) Rd <-- STACK

Синтаксис	Операнды:	Счетчик программ:
(i) POP Rd	0 < d <31	PC<-- + 1 SP<-- SP + 1

```
rcall routine ; Вызвать подпрограмму
```

```
. . .
routine: push r14     ; Сохранить r14 в стеке
push r13     ; Сохранить r13 в стеке
. . .
pop  r13     ; Восстановить r13
pop  r14     ; Восстановить r14
ret          ; Вернуться из подпрограммы
```

66. Команда PUSH - поместить регистр в стек

Команда помещает содержимое регистра Rd в стек.

(i) STACK <-- Rr

Синтаксис	Операнды:	Счетчик программ:
(i) PUSH Rr	0 < d <31	PC<-- + 1 SP<-- SP - 1

```
rcall routine ; Вызвать подпрограмму
. . .
routine: push r14     ; Сохранить r14 в стеке
push r13     ; Сохранить r13 в стеке
. . .
pop  r13     ; Восстановить r13
pop  r14     ; Восстановить r14
ret          ; Вернуться из подпрограммы
```

67. Команда RCALL - вызвать подпрограмму относительно

Команда вызывает подпрограмму в пределах +2 Кслов (4 Кбайт). Адрес возврата (после выполнения команды RCALL) сохраняется в стеке (См. также команду CALL).

(i) PC <-- PC + k + 1 Приборы с 16-разрядным счетчиком команд, максимум 128 Кбайт памяти программ

Синтаксис	Операнды:	Счетчик программ:	Стек
(i) RCALL k	-2K < k <2K	PC <-- PC + k + 1	STACK <-- PC + 1 SP <-- SP-2 (2 байта, 16 бит)

```
rcall routine ; Вызвать подпрограмму
. . .
routine: push  r14     ; Сохранить r14 в стеке
. . .
pop   r14     ; Восстановить r14
ret           ; Вернуться из подпрограммы
```

68. Команда RET - вернуться из подпрограммы

Команда возвращает из подпрограммы. Адрес возврата загружается из стека.

(i) PC(15-0) <-- STACK Приборы с 16-разрядным счетчиком команд, максимум 128 Кбайт памяти программ

Синтаксис	Операнды:	Счетчик программ:	Стек
(i) RET	None	См. операцию	SP <-- SP+2 (2 байта, 16 бит)

```
rcall routine ; Вызвать подпрограмму
. . .
routine:  push r14     ; Сохранить r14 в стеке
. . .
pop  r14     ; Восстановить r14
ret          ; Вернуться из подпрограммы
```

69. Команда RETI - вернуться из прерывания

Команда возвращает из прерывания. Адрес возврата выгружается из стека и устанавливается флаг глобального прерывания.

(i) PC(15-0) <-- STACK Приборы с 16-разрядным счетчиком команд, максимум 128 Кбайт памяти программ

	Синтаксис	Операнды:	Счетчик программ:	Стек
(i)	RETI	None	См. операцию	SP <-- SP+2 (2 байта, 16 бит)

```
. . .
extint:  push r0  ; Сохранить r0 в стеке
. . .
pop  r0  ; Восстановить r0
reti     ; Вернуться и разрешить прерывания
```

70. Команда RJMP - перейти относительно

Команда выполняет относительный переход по адресу в пределах +2 Кслов (4 Кбайт) текущего состояния счетчика команд. В ассемблере вместо относительных операндов используются метки. Для AVR микроконтроллеров с памятью программ не превышающей 4 Кслов (8 Кбайт) данная команда может адресовать всю память программ.

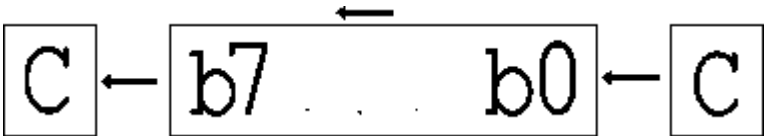
(i) PC <-- PC + k + 1

	Синтаксис	Операнды:	Счетчик программ:	Стек
(i)	RJMP k	-2K < k < 2K	PC <-- PC + k + 1	Стек не меняется

```
cpi   r16, $42 ; Сравнить r16 с $42
brne  error    ; Перейти если r16 <> $42
rjmp  ok       ; Безусловный переход
error:   add   r16, r17 ; Сложить r17 с r16
inc   r16      ; Увеличить на 1 r16
ok:      nop           ; Назначение для rjmp (пустая операция)
```

71. Команда ROL - сдвинуть влево через перенос

Сдвиг всех битов Rd на одно место влево. Флаг переноса (C) регистра состояния (SREG) смещается на место бита 0 регистра Rd. Бит 7 смещается во флаг переноса (C).

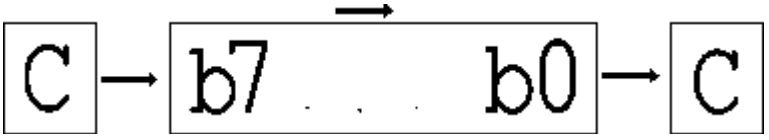


	Синтаксис	Операнды:	Счетчик программ:
(i)	ROL Rd	0 < d < 31	PC <- PC + 1

```
rol   r15    ; Сдвигать влево
brcs  oneenc ; Перейти если установлен перенос
. . .
oneenc:  nop         ; Перейти по назначению (пустая операция)
```

72. Команда ROR - сдвинуть вправо через перенос

Сдвиг всех битов Rd на одно место вправо. Флаг переноса (C) регистра состояния (SREG) смещается на место бита 7 регистра Rd. Бит 0 смещается во флаг переноса (C).



	Синтаксис	Операнды:	Счетчик программ:
(i)	ROR Rd	0 < d < 31	PC <- PC + 1

```
ror  r15     ; Сдвигать вправо
brcc zeroenc ; Перейти если перенос очищен
```

. . .
zeroenc: nop ; Перейти по назначению (пустая операция)

73. Команда SBC - вычесть с переносом

Вычитание содержимого регистра-источника и содержимого флага переноса (C) из регистра Rd, размещение результата в регистре назначения Rd.

(i) Rd <-- Rd - Rr - C

	Синтаксис	Операнды:	Счетчик программ:
(i)	SBC Rd,Rr	0 < d < 31, 0 < r < 31	PC <- PC + 1

; Вычесть r1 : r0 из r3 : r2
sub r2, r0 ; Вычесть младший байт
sbc r3, r1 ; Вычесть старший байт с переносом

74. Команда SBCI - вычесть непосредственное значение с переносом

Вычитание константы и содержимого флага переноса (C) из содержимого регистра, размещение результата в регистре назначения Rd.

(i) Rd <-- Rd - K - C

	Синтаксис	Операнды:	Счетчик программ:
(i)	SBCI Rd,K	0 < d < 31, 0 < K < 255	PC <- PC + 1

; Вычесть $4F23 из r17 : r16
subi r16, r23 ; Вычесть младший байт
sbci r17, $4F ; Вычесть старший байт с переносом

75. Команда SBI - установить бит в регистр I/O

Команда устанавливает заданный бит в регистр I/O. Команда работает с младшими 32 регистрами I/O (адреса с 0 по 31)

Операция (i) I/O(P,b) <-- 1

	Синтаксис	Операнды:	Счетчик программ:
(i)	SBI P,b	0 < P <31, 0 < b < 7	PC <-- PC + k + 1

out $1E, r0 ; Записать адрес EEPROM
sbi $1C, 0 ; Установить бит чтения в EECR
in r1, $1D ; Считать данные EEPROM

76. Команда SBIC - пропустить если бит в регистре I/O очищен

Команда проверяет состояние бита в регистре I/O и, если этот бит очищен, пропускает следующую команду. Данная команда работает с младшими 32 регистрами I/O (адреса с 0 по 31).

(i) If I/O(P,b) = 0 then PC <-- PC + 2 (or 3) else PC <-- PC + 1

	Синтаксис	Операнды:	Счетчик программ:
(i)	SBIC P,b	0 < P <31, 0 < b < 7	PC <-- PC + 1, если условия не соблюдены, нет пропуска PC <-- PC + 2, если следующая команда длиной в 1слово

e2wait: sbic $1C, 1 ; Пропустить следующую команду если EEWE очищен
rjmp e2wait ; Запись EEPROM не завершена
nop ; Продолжать (пустая операция)

77. Команда SBIS - пропустить если бит в регистре I/O установлен

Команда проверяет состояние бита в регистре I/O и, если этот бит установлен, пропускает следующую команду. Данная команда работает с младшими 32 регистрами I/O (адреса с 0 по 31).

(i) If I/O(P,b) = 1 then PC <-- PC + 2 (or 3) else PC <-- PC + 1

	Синтаксис	Операнды:	Счетчик программ:
(i)	SBIS P,b	0 < P <31, 0 < b < 7	PC <-- PC + 1, если условия не соблюдены, нет пропуска PC <-- PC + 2, если следующая команда длиной в 1слово

```
waitset:  sbis$10, 0   ; Пропустить следующую команду если установлен бит 0 в Порте D
rjmp waitset ; Бит не установлен
nop         ; Продолжать (пустая операция)
```

78. Команда SBIW - вычесть непосредственное значение из слова

Вычитание непосредственного значения (0-63) из пары регистров и размещение результата в паре регистров. Команда работает с четырьмя верхними парами регистров, удобна для работы с регистрами указателями.

(i) Rdh:Rdl <-- Rdh:Rdl - K

	Синтаксис	Операнды:	Счетчик программ:
(i)	SBIW Rdl,K	dl∈{24,26,28,30}, 0 < K <63	PC <- PC + 1

```
sbiw  r24, 1  ; Вычесть 1 из r25:r24
sbiw  r28, 63 ; Вычесть 63 из Y указателя (r29 : r28)
```

79. Команда SBR - установить биты в регистре

Команда выполняет установку определенных битов в регистре Rd. Команда выполняет логическое ORI между содержимым регистра Rd и маской-константой K и размещает результат в регистре назначения Rd.

(i) Rd <-- Rd v K

	Синтаксис	Операнды:	Счетчик программ:
(i)	SBR Rd,K	16 < d < 31, 0 < K < 255	PC <- PC + 1

```
sbr r16, 3F0 ; Установить биты 0 и 1 в r16
sbr r17, $F0 ; Установить старшие 4 бита в r17
```

80. Команда SBRC - пропустить если бит в регистре очищен

Команда проверяет состояние бита в регистре и, если этот бит очищен, пропускает следующую команду.

(i) If Rr (b) = 0 then PC <-- PC + 2 (or 3) else PC <-- PC + 1

	Синтаксис	Операнды:	Счетчик программ:
(i)	SBRC Rr,b	0 < r < 31, 0 < b < 7	PC <-- PC + 1, если условия не соблюдены, нет пропуска PC <-- PC + 2, если следующая команда длиной в 1слово

```
sub r0, r1 ; Вычесть r1 из r0
sbrc r0, 7 ; Пропустить если бит 7 в r0 очищен
sub r0, r1 ; Выполняется только если бит 7 в r0 не очищен
nop        ; Продолжать (пустая операция)
```

81. Команда SEC - установить флаг переноса

Команда устанавливает флаг переноса (C) в регистре статуса (SREG)

(i) C <-- 1

Синтаксис	Операнды:	Счетчик программ:

(i)	SEC	None	PC <-- PC + 1

```
sec         ; Установить флаг переноса
adc r0, r1   ; r0 = r0 + r1 + 1
```

82. Команда SEH - установить флаг полупереноса

Команда устанавливает флаг полупереноса (H) в регистре статуса (SREG).

(i) H <-- 1

	Синтаксис	Операнды:	Счетчик программ:
(i)	SEH	None	PC <-- PC + 1

```
seh ; Установить флаг полупереноса
```

83. Команда SEI - установить флаг глобального прерывания

Команда устанавливает флаг глобального прерывания (I) в регистре статуса (SREG).

(i) I <-- 1

	Синтаксис	Операнды:	Счетчик программ:
(i)	SEI	None	PC <-- PC + 1

```
cli         ; Запретить прерывания
in r13, $16 ; Считать Порт B
sei         ; Разрешить прерывания
```

84. Команда SEN - установить флаг отрицательного значения

Команда устанавливает флаг отрицательного значения (N) в регистре статуса (SREG).

(i) N <-- 1

	Синтаксис	Операнды:	Счетчик программ:
(i)	SEN	None	PC <-- PC + 1

```
add r2, r19 ; Сложить r19 с r2
sen         ; Установить флаг отрицательного значения
```

85. Команда SER - установить все биты регистра

Значение $FF заносится непосредственно в регистр назначения Rd.

(i) Rd <-- $FF

	Синтаксис	Операнды:	Счетчик программ:
(i)	SER Rd	16 < d < 31	PC <-- PC + 1

```
clr  r16     ; Очистить r16
ser  r17     ; Установить r17
out  #18, r16 ; Записать нули в Порт B
nop          ; Задержка (пустая операция)
out  #18, r17 ; Записать единицы в Порт B
```

86. Команда SES - установить флаг знака

Команда устанавливает флаг учета знака (S) в регистре статуса (SREG).

(i) S <-- 1

	Синтаксис	Операнды:	Счетчик программ:
(i)	SES	None	PC <-- PC + 1

```
add r2, r19 ; Сложить r19 с r2
ses         ; Установить флаг отрицательного значения
```

87. Команда SET - установить флаг T

Команда устанавливает флаг пересылки (T) в регистре статуса (SREG).

(i) T <-- 1

	Синтаксис	Операнды:	Счетчик программ:
(i)	SET	None	PC <-- PC + 1

```
set ; Установить T флаг
```

88. Команда SEV - установить флаг переполнения

Команда устанавливает флаг переполнения (V) в регистре статуса (SREG).

(i) V <-- 1

	Синтаксис	Операнды:	Счетчик программ:
(i)	SEV	None	PC <-- PC + 1

```
add r2, r19 ; Сложить r19 с r2
sev         ; Установить флаг переполнения
```

89. Команда SEZ - установить флаг нулевого значения

Команда устанавливает флаг нулевого значения (Z) в регистре статуса (SREG).

(i) Z <-- 1

	Синтаксис	Операнды:	Счетчик программ:
(i)	SEZ	None	PC <-- PC + 1

```
add r2, r19 ; Сложить r19 с r2
sez         ; Установить флаг нулевого значения
```

90. Команда SLEEP - установить режим SLEEP

Команда устанавливает схему в SLEEP режим, определяемый регистром управления ЦПУ. Когда прерывание выводит ЦПУ из SLEEP режима команда, следующая за командой SLEEP, будет выполнена прежде, чем отработает обработчик прерывания.

	Синтаксис	Операнды:	Счетчик программ:
(i)	SLEEP	None	PC <-- PC + 1

```
mov r0, r11 ; Копировать r11 в r0
sleep       ; Перевести MCU в режим sleep
```

91. Команда ST - записать косвенно из регистра в СОЗУ с использованием индекса X

Записывается косвенно один байт из регистра в СОЗУ. Положение байта в СОЗУ указывается 16-разрядным регистром-указателем X в регистровом файле. Обращение к памяти ограничено текущей страницей объемом 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 очень удобна при использовании регистра-указателя X в качестве указателя стека.

Использование X-указателя:

	Операция:		Комментарий:
(i)	(X) <-- Rr		X: Неизменен
(ii)	(X) <-- Rr	X <-- X + 1	X: Инкрементирован впоследствии

(iii)	X <-- X - 1	(X) <-- Rr	X: Предварительно декрементирован

	Синтаксис	Операнды:	Счетчик программ:
(i)	ST X,Rr	0 < d < 31	PC <-- PC + 1
(ii)	ST X+,Rr	0 < d < 31	PC <-- PC + 1
(iii)	ST -X,Rr	0 < d < 31	PC <-- PC + 1

```
clr   r27      ; Очистить старший байт X
ldi   r26, $20 ; Установить $20 в младший байт X
st    X+,r0    ; Сохранить в r0 содержимое SRAM по адресу $20 (X постинкрементируется)
st    X, r1    ; Сохранить в r1 содержимое SRAM по адресу $21
ldi   r26, $23 ; Установить $23 в младший байт X
st    r2, X    ; Сохранить в r2 содержимое SRAM по адресу $23
st    r3, -X   ; Сохранить в r3 содержимое SRAM по адресу $22 (X преддекрементируется)
```

92. Команда ST (STD) - записать косвенно из регистра в СОЗУ с использованием индекса Y

Записывается косвенно, со смещением или без смещения, один байт из регистра в СОЗУ. Положение байта в СОЗУ указывается 16-разрядным регистром-указателем Y в регистровом файле. Обращение к памяти ограничено текущей страницей объемом 64 Кбайта. Для обращения к другой странице СОЗУ необходимо изменить регистр RAMPY в I/O области.Регистр-указатель Y может остаться неизменным после выполнения команды, но может быть инкрементирован или декрементирован. Эта особенность очень удобна при использовании регистра-указателя Y в качестве указателя стека.

Использование Y-указателя:

	Операция:		Комментарий:
(i)	(Y) <-- Rr		Y: Неизменен
(ii)	(Y) <-- Rr	Y <-- Y + 1	Y: Инкрементирован впоследствии
(iii)	Y <-- Y - 1	(Y) <-- Rr	Y: Предварительно декрементирован
(iv)	(Y + q) <-- Rr		Y: Неизменен, q: смещение

	Синтаксис	Операнды:	Счетчик программ:
(i)	ST Y,Rr	0 < d < 31	PC <-- PC + 1
(ii)	ST Y+,Rr	0 < d < 31	PC <-- PC + 1
(iii)	ST -Y,Rr	0 < d < 31	PC <-- PC + 1
(iv)	STD Y+q,Rr	0 < d < 31, 0 < q < 63	PC <-- PC + 1

```
clr   r29      ;Очистить старший байт Y
ldi   r28, $20 ;Установить $20 в младший байт Y
st    Y+, r0   ;Сохранить в r0 содерж. SRAM по адресу $20 (Y постинкрементируется)
st    Y, r1    ;Сохранить в r1 содержимое SRAM по адресу $21
ldi   r28, $23 ;Установить $23 в младший байт Y
st    Y, r2    ;Сохранить в r2 содержимое SRAM по адресу $23
st    -Y, r3   ;Сохранить в r3 содерж. SRAM по адресу $22 (Y преддекрементируется)
std   Y+2, r4  ;Сохранить в r4 содержимое SRAM по адресу $24
```

93. Команда ST (STD) - записать косвенно из регистра в СОЗУ с использованием индекса Z

Записывается косвенно, со смещением или без смещения, один байт из регистра в СОЗУ. Положение байта в СОЗУ указывается 16-разрядным регистром-указателем Z в регистровом файле. Обращение к памяти ограничено текущей страницей объемом 64 Кбайта. Для обращения к другой странице СОЗУ необходимо изменить регистр RAMPZ в I/O области. Регистр-указатель Z может остаться неизменным после выполнения команды, но может быть инкрементирован или декрементирован. Эта особенность очень удобна при использовании регистра-указателя Z в качестве указателя стека, однако, поскольку регистр-указатель Z может быть использован для косвенного вызова подпрограмм, косвенных переходов и табличных преобразований, более удобно использовать в качестве указателя стека регистры-указатели X и Y.

Использование Z-указателя:

Операция:	Комментарий:

(i)	(Z) <-- Rr		Z: Неизменен
(ii)	(Z) <-- Rr	Z <-- Y + 1	Z: Инкрементирован впоследствии
(iii)	Z <-- Z - 1	(Z) <-- Rr	Z: Предварительно декрементирован
(iv)	(Z + q) <-- Rr		Z: Неизменен, q: смещение

	Синтаксис	Операнды:	Счетчик программ:
(i)	ST Z,Rr	0 < d < 31	PC <-- PC + 1
(ii)	ST Z+,Rr	0 < d < 31	PC <-- PC + 1
(iii)	ST -Z,Rr	0 < d < 31	PC <-- PC + 1
(iv)	STD Z+q,Rr	0 < d < 31, 0 < q < 63	PC <-- PC + 1

```
clr   r31      ; Очистить старший байт Z
ldi   r30, $20 ; Установить $20 в младший байт Z
st    Z+,r0    ;Сохранить в r0 содерж. SRAM по адр. $20 (Z постинкрементируется)
st    Z, r1    ; Сохранить в r1 содержимое SRAM по адресу $21
ldi   r30, $23 ; Установить $23 в младший байт Z
st    Z, r2    ; Сохранить в r2 содержимое SRAM по адресу $23
st    -Z, r3   ; Сохранить в r3 содерж.SRAM по адр. $22 (Z преддекрементируется)
std   Z+2, r4  ; Сохранить в r4 содержимое SRAM по адресу $24
```

94. Команда STS - загрузить непосредственно в СОЗУ

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

(i) (k) <-- Rr

	Синтаксис	Операнды:	Счетчик программ:
(i)	STS k,Rr	0 < r < 31, 0 < k < 65535	PC <-- PC + 2

```
lds r2, $FF00 ; Загрузить в r2 содержимое SRAM по адресу $FF00
add r2, r1    ; Сложить r1 с r2
sts $FF00, r2 ; Записать обратно
```

95. Команда SUB - вычесть без переноса

Вычитание содержимого регистра-источника Rr из содержимого регистра Rd, размещение результата в регистре назначения Rd.

(i) Rd <-- Rd - Rr

	Синтаксис	Операнды:	Счетчик программ:
(i)	SUB Rd,Rr	16 < d < 31, 0 < r < 31	PC <- PC + 1

```
sub r13, r12 ; Вычесть r12 из r13
brne noteq   ; Перейти если r12 <> r13
noteq:  nop          ; Перейти по назначению (пустая операция)
```

96. Команда SUBI - вычесть непосредственное значение

Вычитание константы из содержимого регистра, размещение результата в регистре назначения Rd.

(i) Rd <-- Rd - K

	Синтаксис	Операнды:	Счетчик программ:
(i)	SUB Rd,K	16 < d < 31, 0 < K < 255	PC <- PC + 1

```
subi r22, $11 ; Вычесть $11 из r22
brne noteq    ; Перейти если r22 <> $11
. . .
noteq:  nop          ; Перейти по назначению (пустая операция)
```

97. Команда SWAP - поменять нибблы местами

Команда меняет местами старший и младший нибблы (полубайты) регистра.

(i) R(7-4) <-- Rd(3-0), R(3-0) <-- Rd(7-4)

	Синтаксис	Операнды:	Счетчик программ:
(i)	SWAP Rd	0 < d < 31	PC <-- PC + k + 1

```
inc  r1  ; Увеличить на 1 r1
swap r1  ; Поменять местами нибблы r1
inc  r1  ; Увеличить на 1 старший ниббл r1
swap r1  ; Снова поменять местами нибблы r1
```

98. Команда TST - проверить на ноль или минус

Регистр проверяется на нулевое или отрицательное состояние. Выполняется логическое AND содержимого регистра с самим собой. Содержимое регистра остается неизменным.

(i) Rd <-- Rd * Rd

	Синтаксис	Операнды:	Счетчик программ:
(i)	TST Rd	0 < d < 31	PC <- PC + 1

```
tst  r0   ; Проверить r0
breq zero ; Перейти если r0 = 0
. . .
zero:   nop       ; Перейти по назначению (пустая операция)
```

99. Команда WDR - сбросить сторожевой таймер

Команда сбрасывает сторожевой таймер (Watchdog Timer). Команда может быть выполнена внутри заданного прескалером сторожевого таймера промежутка времени (см. аппаратные характеристики сторожевого таймера).

(i) Перезапускается WD (сторожевой таймер)

	Синтаксис	Операнды:	Счетчик программ:
(i)	WDR	None	PC <-- PC + 1

```
wdr ; Сбросить сторожевой таймер
```

Директивы ассемблера

Компилятор поддерживает ряд директив. Директивы не транслируются непосредственно в код. Вместо этого они используются для указания положения в программной памяти, определения макросов, инициализации памяти и т.д. Список директив приведён в следующей таблице.

Директива	**Описание**
BYTE	Зарезервировать байты в ОЗУ
CSEG	Программный сегмент
DB	Определить байты во флэш или EEPROM

DEF	Назначить регистру символическое имя
DEVICE	Определить устройство для которого компилируется программа
DSEG	Сегмент данных
DW	Определить слова во флэш или EEPROM
ENDM, ENDMACRO	Конец макроса
EQU	Установить постоянное выражение
ESEG	Сегмент EEPROM
EXIT	Выйти из файла
INCLUDE	Вложить другой файл
LIST	Включить генерацию листинга
LISTMAC	Включить разворачивание макросов в листинге
MACRO	Начало макроса
NOLIST	Выключить генерацию листинга
ORG	Установить положение в сегменте
SET	Установить переменный символический эквивалент выражения

Все директивы предваряются точкой.

BYTE - Зарезервировать байты в ОЗУ

Директива BYTE резервирует байты в ОЗУ. Если Вы хотите иметь возможность ссылаться на выделенную область памяти, то директива BYTE должна быть предварена меткой. Директива принимает один обязательный параметр, который указывает количество выделяемых байт. Эта директива может использоваться только в сегменте данных(смотреть директивы CSEG и DSEG). Выделенные байты не инициализируются.

Синтаксис:
МЕТКА: .BYTE выражение

Пример:
```
.DSEG
var1:   .BYTE 1          ; резервирует 1 байт для var1
table:  .BYTE tab_size   ; резервирует tab_size байт

.CSEG
      ldi r30,low(var1)  ; Загружает младший байт регистра Z
      ldi r31,high(var1) ; Загружает старший байт регистра Z
      ld r1,Z            ; Загружает VAR1 в регистр 1
```

CSEG - Программный сегмент

Директива CSEG определяет начало программного сегмента. Исходный файл может состоять из нескольких программных сегментов, которые объединяются в один программный сегмент при компиляции. Программный сегмент является сегментом по умолчанию. Программные сегменты имеют свои собственные счётчики положения которые считают не побайтно, а по словно. Директива ORG может быть использована для размещения кода и констант в необходимом месте сегмента. Директива CSEG не имеет параметров.

Синтаксис:
.CSEG

Пример:
```
.DSEG                ; Начало сегмента данных
vartab: .BYTE 4      ; Резервирует 4 байта в ОЗУ

.CSEG                ; Начало кодового сегмента
const:  .DW 2        ; Разместить константу 0x0002 в памяти программ
      mov r1,r0      ; Выполнить действия
```

DB - Определить байты во флэш или EEPROM

Директива DB резервирует необходимое количество байт в памяти программ или в EEPROM. Если Вы хотите иметь возможность ссылаться на выделенную область памяти, то директива DB должна быть предварена меткой. Директива DB должна иметь хотя бы один параметр. Данная директива может быть размещена только в сегменте программ (CSEG) или в сегменте EEPROM (ESEG).

Параметры передаваемые директиве - это последовательность выражений разделённых запятыми. Каждое выражение должно быть или числом в диапазоне (-128..255), или в результате вычисления должно давать результат в этом же диапазоне, в противном случае число усекается до байта, причём БЕЗ выдачи предупреждений.

Если директива получает более одного параметра и текущим является программный сегмент, то параметры упаковываются в слова (первый параметр - младший байт), и если число параметров нечётно, то последнее выражение будет усечено до байта и записано как слово со старшим байтом равным нулю, даже если далее идет ещё одна директива DB.

Синтаксис:
МЕТКА: .DB список_выражений

Пример:
```
.CSEG
consts: .DB 0, 255, 0b01010101, -128, 0xaa

.ESEG
const2: .DB 1,2,3
```

DEF - Назначить регистру символическое имя

Директива DEF позволяет ссылаться на регистр через некоторое символическое имя. Назначенное имя может использоваться во всей нижеследующей части программы для обращений к данному регистру. Регистр может иметь несколько различных имен. Символическое имя может быть переназначено позднее в программе.

Синтаксис:
.DEF Символическое_имя = Регистр

Пример:
```
.DEF temp=R16
.DEF ior=R0

.CSEG
 ldi temp,0xf0  ; Загрузить 0xf0 в регистр temp (R16)
 in ior,0x3f  ; Прочитать SREG в регистр ior (R0)
 eor temp,ior  ; Регистры temp и ior складываются по исключающему или
```

DEVICE - Определить устройство для которого компилируется программа

Директива DEVICE позволяет указать для какого устройства компилируется программа. При использовании данной директивы компилятор выдаст предупреждение, если будет найдена инструкция, которую не поддерживает данный микроконтроллер. Также будет выдано предупреждение, если программный сегмент, либо сегмент EEPROM превысят размер допускаемый устройством. Если же директива не используется то все инструкции считаются допустимыми, и отсутствуют ограничения на размер сегментов.

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

Пример:
```
.DEVICE AT90S1200  ; Используется AT90S1200

.CSEG
      push r30   ; Эта инструкция вызовет предупреждение
            ; поскольку AT90S1200 её не имеет
```

DSEG - Сегмент данных

Директива DSEG определяет начало сегмента данных. Исходный файл может состоять из нескольких сегментов данных, которые объединяются в один сегмент при компиляции. Сегмент данных обычно состоит только из директив BYTE и меток. Сегменты данных имеют свои собственные побайтные счётчики положения. Директива ORG может быть использована для размещения переменных в необходимом месте ОЗУ. Директива не имеет параметров.

Синтаксис:
```
.DSEG
```

Пример:
```
.DSEG                  ; Начало сегмента данных
var1:  .BYTE 1             ; зарезервировать 1 байт для var1
table:  .BYTE tab_size       ; зарезервировать tab_size байт.

.CSEG
      ldi r30,low(var1)    ; Загрузить младший байт регистра Z
      ldi r31,high(var1)   ; Загрузить старший байт регистра Z
      ld r1,Z            ; Загрузить var1 в регистр r1
```

DW - Определить слова во флэш или EEPROM

Директива DW резервирует необходимое количество слов в памяти программ или в EEPROM. Если Вы хотите иметь возможность ссылаться на выделенную область памяти, то

директива DW должна быть предварена меткой. Директива DW должна иметь хотя бы один параметр. Данная директива может быть размещена только в сегменте программ (CSEG) или в сегменте EEPROM (ESEG). Параметры передаваемые директиве - это последовательность выражений разделённых запятыми. Каждое выражение должно быть или числом в диапазоне (-32768..65535), или в результате вычисления должно давать результат в этом же диапазоне, в противном случае число усекается до слова, причем БЕЗ выдачи предупреждений.

Синтаксис:
METKA: .DW expressionlist

Пример:
```
.CSEG
varlist:  .DW 0, 0xffff, 0b1001110001010101, -32768, 65535

.ESEG
eevarlst: .DW 0,0xffff,10
```

ENDMACRO - Конец макроса

Директива определяет конец макроопределения, и не принимает никаких параметров. Для информации по определению макросов смотрите директиву MACRO.

Синтаксис:
.ENDMACRO

Пример:
```
.MACRO SUBI16              ; Начало определения макроса
      subi r16,low(@0)    ; Вычесть младший байт первого параметра
      sbci r17,high(@0)   ; Вычесть старший байт первого параметра
.ENDMACRO
```

EQU - Установить постоянное выражение

Директива EQU присваивает метке значение. Эта метка может позднее использоваться в выражениях. Метка которой присвоено значение данной директивой не может быть переназначена и её значение не может быть изменено.

Синтаксис:
.EQU метка = выражение

Пример:
```
.EQU io_offset = 0x23
.EQU porta     = io_offset + 2

.CSEG               ; Начало сегмента данных
      clr r2        ; Очистить регистр r2
      out porta,r2  ; Записать в порт A
```

ESEG - Сегмент EEPROM

Директива ESEG определяет начало сегмента EEPROM. Исходный файл может состоять из нескольких сегментов EEPROM, которые объединяются в один сегмент при компиляции. Сегмент EEPROM обычно состоит только из директив DB, DW и меток. Сегменты EEPROM имеют свои собственные побайтные счётчики положения. Директива ORG может быть

использована для размещения переменных в необходимом месте EEPROM. Директива не имеет параметров.

Синтаксис:
.ESEG
Пример:
```
.DSEG                ; Начало сегмента данных
var1:   .BYTE 1      ; зарезервировать 1 байт для var1
table:  .BYTE tab_size   ; зарезервировать tab_size байт.
.ESEG
eevar1: .DW 0xffff       ; проинициализировать 1 слово в EEPROM
```

EXIT - Выйти из файла

Встретив директиву EXIT компилятор прекращает компиляцию данного файла. Если директива использована во вложенном файле (см. директиву INCLUDE), то компиляция продолжается со строки следующей после директивы INCLUDE. Если же файл не является вложенным, то компиляция прекращается.

Синтаксис:
.EXIT
Пример:
```
.EXIT  ; Выйти из данного файла
```

INCLUDE - Вложить другой файл

Встретив директиву INCLUDE компилятор открывает указанный в ней файл, компилирует его пока файл не закончится или не встретится директива EXIT, после этого продолжает компиляцию начального файла со строки следующей за директивой INCLUDE. Вложенный файл может также содержать директивы INCLUDE.

Синтаксис:
.INCLUDE "имя_файла"
Пример:
```
; файл iodefs.asm:
.EQU sreg   = 0x3f     ; Регистр статуса
.EQU sphigh = 0x3e     ; Старший байт указателя стека
.EQU splow  = 0x3d     ; Младший байт указателя стека
; файл incdemo.asm
.INCLUDE iodefs.asm    ; Вложить определения портов
      in r0,sreg       ; Прочитать регистр статуса
```

LIST - Включить генерацию листинга

Директива LIST указывает компилятору на необходимость создания листинга. Листинг представляет из себя комбинацию ассемблерного кода, адресов и кодов операций. По умолчанию генерация листинга включена, однако данная директива используется совместно с директивой NOLIST для получения листингов отдельных частей исходных файлов.

Синтаксис:
.LIST
Пример:
```
.NOLIST              ; Отключить генерацию листинга
.INCLUDE "macro.inc"   ; Вложенные файлы не будут
.INCLUDE "const.def"   ; отображены в листинге
.LIST                ; Включить генерацию листинга
```

LISTMAC - Включить разворачивание макросов в листинге

После директивы LISTMAC компилятор будет показывать в листинге содержимое макроса. По умолчанию в листинге показывается только вызов макроса и передаваемые параметры.

Синтаксис:
.LISTMAC
Пример:
```
.MACRO MACX        ; Определение макроса
      add  r0,@0  ; Тело макроса
      eor  r1,@1
.ENDMACRO          ; Конец макроопределения
.LISTMAC           ; Включить разворачивание макросов
      MACX r2,r1  ; Вызов макроса (в листинге будет показано тело макроса)
```

MACRO - Начало макроса

С директивы MACRO начинается определение макроса. В качестве параметра директиве передаётся имя макроса. При встрече имени макроса позднее в тексте программы, компилятор заменяет это имя на тело макроса. Макрос может иметь до 10 параметров, к которым в его теле обращаются через @0-@9. При вызове параметры перечисляются через запятые. Определение макроса заканчивается директивой ENDMACRO. По умолчанию в листинг включается только вызов макроса, для разворачивания макроса необходимо использовать директиву LISTMAC. Макрос в листинге показывается знаком +.

Синтаксис:
.MACRO макроимя
Пример:
```
.MACRO SUBI16                ; Начало макроопределения
      subi @1,low(@0)        ; Вычесть младший байт параметра 0 из параметра 1
      sbci @2,high(@0)       ; Вычесть старший байт параметра 0 из параметра 2
.ENDMACRO                    ; Конец макроопределения
.CSEG                        ; Начало программного сегмента
      SUBI16 0x1234,r16,r17   ; Вычесть 0x1234 из r17:r16
```

NOLIST - Выключить генерацию листинга

Директива NOLIST указывает компилятору на необходимость прекращения генерации листинга. Листинг представляет из себя комбинацию ассемблерного кода, адресов и кодов операций. По умолчанию генерация листинга включена, однако может быть отключена данной директивой. Кроме того данная директива может быть использована совместно с директивой LIST для получения листингов отдельных частей исходных файлов

Синтаксис:
.NOLIST
Пример:
```
.NOLIST              ; Отключить генерацию листинга
.INCLUDE "macro.inc"   ; Вложенные файлы не будут
.INCLUDE "const.def"   ; отображены в листинге
.LIST                ; Включить генерацию листинга
```

ORG - Установить положение в сегменте

Директива ORG устанавливает счётчик положения равным заданной величине, которая передаётся как параметр. Для сегмента данных она устанавливает счётчик положения в SRAM

(ОЗУ), для сегмента программ это программный счётчик, а для сегмента EEPROM это положение в EEPROM. Если директиве предшествует метка (в той же строке) то метка размещается по адресу указанному в параметре директивы. Перед началом компиляции программный счётчик и счётчик EEPROM равны нулю, а счётчик ОЗУ равен 32 (поскольку адреса 0-31 заняты регистрами). Обратите внимание что для ОЗУ и EEPROM используются побайтные счётчики а для программного сегмента - пословный.

Синтаксис:
.ORG выражение
Пример:
```
.DSEG              ; Начало сегмента данных
.ORG 0x37          ; Установить адрес SRAM равным 0x37
variable: .BYTE 1  ; Зарезервировать байт по адресу 0x37H
.CSEG
.ORG 0x10          ; Установить программный счётчик равным 0x10
      mov r0,r1  ; Данная команда будет размещена по адресу 0x10
```

SET - Установить переменный символический эквивалент выражения

Директива SET присваивает имени некоторое значение. Это имя позднее может быть использовано в выражениях. Причем в отличии от директивы EQU значение имени может быть изменено другой директивой SET.

Синтаксис:
.SET имя = выражение
Пример:
```
.SET io_offset = 0x23
.SET porta     = io_offset + 2
.CSEG              ; Начало кодового сегмента
      clr r2       ; Очистить регистр 2
      out porta,r2  ; Записать в порт A
```

Выражения

Компилятор позволяет использовать в программе выражения которые могут состоять операндов, операторов и функций. Все выражения являются 32-битными.

Операнды

Могут быть использованы следующие операнды:

- Метки определённые пользователем (дают значение своего положения).
- Переменные определённые директивой SET
- Константы определённые директивой EQU
- Числа заданные в формате:
  - Десятичном (принят по умолчанию): 10, 255
  - Шестнадцатеричном (два варианта записи): 0x0a, $0a, 0xff, $ff
  - Двоичном: 0b00001010, 0b11111111
  - Восьмеричном (начинаются с нуля): 010, 077
- PC - текущее значение программного счётчика (Programm Counter)

Операторы

Компилятор поддерживает ряд операторов которые перечислены в таблице (чем выше положение в таблице, тем выше приоритет оператора). Выражения могут заключаться в круглые скобки, такие выражения вычисляются перед выражениями за скобками.

Приоритет	Символ	Описание
14	!	Логическое отрицание
14	~	Побитное отрицание
14	-	Минус
13	*	Умножение
13	/	Деление
12	+	Суммирование
12	-	Вычитание
11	<<	Сдвиг влево
11	>>	Сдвиг вправо
10	<	Меньше чем
10	<=	Меньше или равно
10	>	Больше чем
10	>=	Больше или равно
9	==	Равно
9	!=	Не равно
8	&	Побитное И
7	^	Побитное исключающее ИЛИ
6	|	Побитное ИЛИ
5	&&	Логическое И
4	||	Логическое ИЛИ

Логическое отрицание

Символ: !
Описание: Возвращает 1 если выражение равно 0, и наоборот
Приоритет: 14
Пример: ldi r16, !0xf0 ; В r16 загрузить 0x00

Побитное отрицание

Символ: ~
Описание: Возвращает выражение в котором все биты проинвертированы
Приоритет: 14
Пример: ldi r16, ~0xf0 ; В r16 загрузить 0x0f

Минус

Символ: -
Описание: Возвращает арифметическое отрицание выражения

Приоритет: 14
Пример: ldi r16,-2 ; Загрузить -2(0xfe) в r16

Умножение

Символ: *
Описание: Возвращает результат умножения двух выражений
Приоритет: 13
Пример: ldi r30, label*2

Деление

Символ: /
Описание: Возвращает целую часть результата деления левого выражения на правое
Приоритет: 13
Пример: ldi r30, label/2

Суммирование

Символ: +
Описание: Возвращает сумму двух выражений
Приоритет: 12
Пример: ldi r30, c1+c2

Вычитание

Символ: -
Описание: Возвращает результат вычитания правого выражения из левого
Приоритет: 12
Пример: ldi r17, c1-c2

Сдвиг влево

Символ: <<
Описание: Возвращает левое выражение сдвинутое влево на число бит указанное справа
Приоритет: 11
Пример: ldi r17, 1<<bitmask ; В r17 загрузить 1 сдвинутую влево bitmask раз

Сдвиг вправо

Символ: >>
Описание: Возвращает левое выражение сдвинутое вправо на число бит указанное справа
Приоритет: 11
Пример: ldi r17, c1>>c2 ; В r17 загрузить c1 сдвинутое вправо c2 раз

Меньше чем

Символ: <
Описание: Возвращает 1 если левое выражение меньше чем правое (учитывается знак), и 0 в противном случае
Приоритет: 10
Пример: ori r18, bitmask*(c1<c2)+1

Меньше или равно

Символ: <=
Описание: Возвращает 1 если левое выражение меньше или равно чем правое (учитывается знак), и 0 в противном случае

Приоритет: 10
Пример: ori r18, bitmask*(c1<=c2)+1

Больше чем

Символ: >
Описание: Возвращает 1 если левое выражение больше чем правое (учитывается знак), и 0 в противном случае
Приоритет: 10
Пример: ori r18, bitmask*(c1>c2)+1

Больше или равно

Символ: >=
Описание: Возвращает 1 если левое выражение больше или равно чем правое (учитывается знак), и 0 в противном случае
Приоритет: 10
Пример: ori r18, bitmask*(c1>=c2)+1

Равно

Символ: ==
Описание: Возвращает 1 если левое выражение равно правому (учитывается знак), и 0 в противном случае
Приоритет: 9
Пример: andi r19, bitmask*(c1==c2)+1

Не равно

Символ: !=
Описание: Возвращает 1 если левое выражение не равно правому (учитывается знак), и 0 в противном случае
Приоритет: 9
Пример: .SET flag = (c1!=c2) ;Установить flag равным 1 или 0

Побитное И

Символ: &
Описание: Возвращает результат побитового И выражений
Приоритет: 8
Пример: ldi r18, High(c1&c2)

Побитное исключающее ИЛИ

Символ: ^
Описание: Возвращает результат побитового исключающего ИЛИ выражений
Приоритет: 7
Пример: ldi r18, Low(c1^c2)

Побитное ИЛИ

Символ: |
Описание: Возвращает результат побитового ИЛИ выражений
Приоритет: 6
Пример: ldi r18, Low(c1|c2)

Логическое И

Символ: &&
Описание: Возвращает 1 если оба выражения не равны нулю, и 0 в противном случае

Приоритет: 5
Пример: ldi r18, Low(c1&&c2)

Логическое ИЛИ

Символ: ||
Описание: Возвращает 1 если хотя бы одно выражение не равно нулю, и 0 в противном случае
Приоритет: 4
Пример: ldi r18, Low(c1||c2)

Функции

Определены следующие функции:

- LOW(выражение) возвращает младший байт выражения
- HIGH(выражение) возвращает второй байт выражения
- BYTE2(выражение) то же что и функция HIGH
- BYTE3(выражение) возвращает третий байт выражения
- BYTE4(выражение) возвращает четвёртый байт выражения
- LWRD(выражение) возвращает биты 0-15 выражения
- HWRD(выражение) возвращает биты 16-31 выражения
- PAGE(выражение) возвращает биты 16-21 выражения
- EXP2(выражение) возвращает 2 в степени (выражение)
- LOG2(выражение) возвращает целую часть log2(выражение)

Использование программы

Открытие файлов

В WAVRASM могут быть открыты как новые так и существующие файлы. Количество открытых файлов ограничено размером памяти, однако объём одного файла не может превышать 28 килобайт (в связи с ограничением MS-Windows). Компиляция файлов большего размера возможна, но они не могут быть редактируемы встроенным редактором. Каждый файл открывается в отдельном окне.

Сообщения об ошибках

После компиляции программы появляется окно сообщений. Все обнаруженные компилятором ошибки будут перечислены в этом окне. При выборе строки с сообщением о ошибке, строка исходного файла, в которой найдена ошибка, становится красной. Если же ошибка находится во вложенном файле, то этого подсвечивания не произойдёт.

Если по строке в окне сообщений клацнуть дважды, то окно файла с указанной ошибкой становится активным, и курсор помещается в начале строки содержащей ошибку. Если же файл с ошибкой не открыт (например это вложенный файл) то он будет автоматически открыт.

Учтите, что если Вы внесли изменения в исходные тексты (добавили или удалили строки), то информация о номерах строк в окне сообщений не является корректной.

Опции

Некоторые установки программы могут быть изменены через пункт меню Options. Если выбрать этот пункт то появится вот такое диалоговое окно:

В поле ввода озаглавленном "List-file extension" вводится расширение используемое для файла листинга, а в поле "Output-file extension" находится расширение для файлов с результатом компиляции программы. В прямоугольнике "Output file format" можно выбрать формат выходного файла (как правило используется интеловский). Однако это не влияет на объектный файл (используемый AVR Studio), который всегда имеет один и тот же формат, и расширение OBJ. Если в исходном файле присутствует сегмент EEPROM то будет также создан файл с расширением EEP. Установки заданные в данном окне запоминаются на постоянно, и при следующем запуске программы, их нет необходимости переустанавливать.

Опция "Wrap relative jumps" даёт возможность "заворачивать" адреса. Эта опция может быть использована только на чипах с объёмом программной памяти 4К слов (8К байт), при этом становится возможным делать относительные переходы (rjmp) и вызовы подпрограмм (rcall) по всей памяти.

Опция "Save before assemble" указывает программе на необходимость автоматического сохранения активного окна (и только его) перед компиляцией.

Если Вы хотите чтобы при закрытии программы, закрывались все открытые окна, то поставьте галочку в поле "Close all windows before exit".