Справка по Ассемблеру для AVR®

Добро пожаловать в ATMEL® AVR® Assembler

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.
Компилятор работает под Microsoft Windows 3.11, Microsoft Windows95 и Microsoft Windows NT. Кроме этого есть консольная версия для MS-DOS.
Набор инструкций семейства микроконтроллеров AVR описан в данном документе кратко, для более полной информации по инструкциям обращайтесь к полному описанию инструкций и документации по конкретному микроконтроллеру.

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

Ниже приведен набор команд процессоров AVR, более детальное описание их можно найти в AVR Data Book.

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
MUL	Rd,Rr	Умножение чисел без знака	R1:R0 = Rd * Rr	Z,C	2
MULS	Rd,Rr	Умножение чисел со знаком	R1:R0 = Rd * Rr	Z,C	2
MULSU	Rd,Rr	Умножение числа со знаком с числом без знака	R1:R0 = Rd * Rr	Z,C	2
FMUL	Rd,Rr	Умножение дробных чисел без знака	R1:R0 = (Rd * Rr) << 1	Z,C	2
FMULS	Rd,Rr	Умножение дробных чисел со знаком	R1:R0 = (Rd *Rr) << 1	Z,C	2
FMULSU	Rd,Rr	Умножение дробного числа со знаком с числом без знака	R1:R0 = (Rd * Rr) << 1	Z,C	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*

CALL	k	Вызов подпрограммы	STACK = PC+2, PC = k	None	4/5*
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 очищен	if(Rr(b)==0) PC = PC + 2 or 3	None	1/2/3
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 CALL, 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
MOVW	Rd,Rr	Скопировать пару регистров	Rd+1:Rd = Rr+1:Rr, r,d even	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
ELPM	Нет	Расширенная загрузка из программной памяти	R0 = (RAMPZ:Z)	None	3
ELPM	Rd,Z	Расширенная загрузка из программной памяти	Rd = (RAMPZ:Z)	None	3
ELPM	Rd,Z+	Расширенная загрузка из программной памяти с пост-инкрементом	Rd = (RAMPZ:Z), Z = Z+1	None	3
SPM	Нет	Сохранение в программной памяти	(Z) = R1:R0	None	-
ESPM	Нет	Расширенное сохранение в программной памяти	(RAMPZ: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)

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:
```
МЕТКА: .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 теломакроса)
```

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
HIGHBYTE3(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)

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 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".

Перевод выполнил Руслан Шимкевич, ruslansh@i.com.ua