

BASCOM BASIC AVR

Wersja 1.11.7.3

Opracowano na podstawie oryginalnego pliku pomocy programu BASCOM-AVR (wersja 1.11.7.3).

Niektóre rysunki pochodzą z oryginalnego pliku pomocy oraz not katalogowych firmy Atmel.

Wszystkie nazwy własne zostały użyte wyłącznie w celu identyfikacji.

Text based on the original BASCOM-AVR help file (version 1.11.7.3).

Some embedded pictures come from MCS Help and Atmel's datasheets.

All brand names used for identification only.

Copyright by Zbigniew Gibek. Poland 2002-2003.

Original English help file is copyrighted by MCS Electronics. All rights reserved.

Visit MCS Electronic Website: www.mcselec.com for more info about BASCOM.

SPIS TREŚCI

Od tłumacza	11
BASCOM AVR - Tworzenie programu	12
BASCOM AVR i pamięć	13
BASCOM AVR - Kody błędów	15
Urządzenia wbudowane w strukturę procesora AVR	19
Licznik-czasomierz TIMER0	21
Licznik-czasomierz TIMER1	22
Układ Watchdog	23
Port B	24
Port D	26
Układ transmisji szeregowej UART	27
Komparator analogowy	30
Urządzenia zewnętrzne	
Używanie SPI	31
Inicjalizacja	38
Rejestry specjalne	39
Alfanumeryczny wyświetlacz LCD	41
Używanie magistrali I ² C	42
Używanie magistrali 1Wire™	43
Topografia wyprowadzeń	47
Podstawy języka BASCOM BASIC	50
Słowa zastrzeżone	62
Różnice w stosunku do BASCOM Basic 8051	68
Dyrektywy kompilatora	
#IF-ELSE-ENDIF	70
\$ASM	71
\$BAUD	71
\$BAUD1 (Nowość w wersji 1.11.6.8)	72
\$BGF	73
\$BOOT (Nowość w wersji 1.11.6.8)	73
\$CRYSTAL	74
\$DATA	75
\$DBG (Nowość w wersji 1.11.6.8)	76
\$DEFAULT	78
\$EEPLEAVE (Nowość w wersji 1.11.7.3)	79
\$EEPROM	79
\$EEPROMHEX (Nowość w wersji 1.11.6.8)	80
\$EXTERNAL	80
\$INCLUDE	81

\$LCD	82
\$LCDRS	83
\$LCDPUTCTRL	83
\$LCDPUTDATA	85
\$LCDVFO (Nowość w wersji 1.11.6.9)	86
\$LIB	86
\$MAP	88
\$NOINIT	88
\$NORAMCLEAR	90
\$REGFILE	91
\$ROMSTART	91
\$SERIALINPUT	92
\$SERIALINPUT1 (Nowość w wersji 1.11.6.8)	93
\$SERIALINPUT2LCD	95
\$SERIALOUTPUT	95
\$SERIALOUTPUT1 (Nowość w wersji 1.11.6.8)	96
\$SIM	96
\$TINY	97
\$WAITSTATE	97
\$XRAMSIZE	97
\$XRAMSTART	98
Elementy języka	
1WIRECOUNT()	98
1WRESET	100
1WREAD()	102
1WSEARCHFIRST()	104
1WSEARCHNEXT()	106
1WVERIFY()	108
1WWRITE	109
ABS()	111
ACOS() (Nowość w wersji 1.11.6.8)	112
ALIAS	112
ASC()	113
ASIN() (Nowość w wersji 1.11.6.8)	114
ATN() (Nowość w wersji 1.11.6.8)	114
ATN2() (Nowość w wersji 1.11.6.8)	115
BAUD	116
BCD()	116
BIN()	117
BINVAL()	118
BIN2GREY()	118
BITWAIT	119
BYVAL, BYREF	120
CALL	120
CHECKSUM()	122

CHR()	122
CIRCLE (Nowość w wersji 1.11.6.8)	123
CLS	125
CLOCKDIVISION	125
CLOSE #	126
CONFIG	128
CONFIG 1WIRE	128
CONFIG ACI (Nowość w wersji 1.11.6.9)	129
CONFIG ADC	129
CONFIG CLOCK	130
CONFIG COM1 (Nowość w wersji 1.11.6.8)	132
CONFIG COM2 (Nowość w wersji 1.11.6.8)	133
CONFIG DATE (Nowość w wersji 1.11.7.3)	133
CONFIG DEBOUNCE	134
CONFIG GRAPHLCD	135
CONFIG I2CDELAY	138
CONFIG INTx	139
CONFIG KBD	140
CONFIG KEYBOARD	140
CONFIG LCD	142
CONFIG LCDBUS	142
CONFIG LCDMODE	143
CONFIG LCDPIN	143
CONFIG PORT, CONFIG PIN	144
CONFIG RC5	145
CONFIG SCL	146
CONFIG SDA	146
CONFIG SERIALIN	147
CONFIG SERIALIN1 (Nowość w wersji 1.11.6.8)	148
CONFIG SERIALOUT	149
CONFIG SERIALOUT1 (Nowość w wersji 1.11.6.8)	150
CONFIG SERVOS	152
CONFIG SPI	153
CONFIG TIMER0	156
CONFIG TIMER1	158
CONFIG TIMER2	160
CONFIG WAITSUART	162
CONFIG WATCHDOG	162
CONFIG X10 (Nowość w wersji 1.11.7.3)	163
COUNTER, CAPTURE, COMPARE i PWM	163
CONST	164
COS() (Nowość w wersji 1.11.6.8)	165
COSH() (Nowość w wersji 1.11.6.8)	165
CRC8()	166
CRC16()	167

CRYSTAL.....	168
CPEEK().....	169
CPEEKH()	170
CURSOR.....	171
DATA.....	171
DATE\$.....	173
DATE() (Nowość w wersji 1.11.7.3).....	175
DAYOFWEEK() (Nowość w wersji 1.11.7.3)	177
DAYOFYEAR() (Nowość w wersji 1.11.7.3).....	178
DBG (Nowość w wersji 1.11.6.8).....	180
DEBOUNCE	180
DECR	181
DECLARE FUNCTION	182
DECLARE SUB	183
DEFxxx.....	184
DEFLCDCHAR	184
DEG2RAD() (Nowość w wersji 1.11.6.8).....	185
DELAY.....	185
DIM.....	186
DISABLE	188
DISPLAY	189
DO...LOOP	189
DTMFOUT.....	190
ECHO	192
ELSE	192
ENABLE	193
END	194
ERR.....	194
EXIT	195
EXP().....	195
FIX() (Nowość w wersji 1.11.6.8)	196
FORMAT().....	196
FOR...NEXT	197
FOURTHLINE	198
FRAC() (Nowość w wersji 1.11.6.8)	199
FUSING()	199
FUNCTION.....	200
GETADC().....	201
GETATKBD().....	202
GETKBD()	204
GETRC()	206
GETRC5()	207
GLCDCMD (Nowość w wersji 1.11.6.9)	209
GLCDDATA (Nowość w wersji 1.11.6.9)	210
GOSUB	210

GOTO.....	211
GREY2BIN().....	211
HEX().....	212
HEXVAL().....	213
HIGH().....	213
HIGHW().....	214
HOME.....	214
I2CINIT (Nowość w wersji 1.11.6.8).....	214
I2CRECEIVE.....	215
I2CSEND.....	216
I2START, I2CSTOP, I2CRBYTE, I2CWBYTE.....	217
IDLE.....	218
IF...THEN...ELSE...END IF.....	218
INCR.....	219
INITLCD.....	220
INKEY().....	220
INP().....	221
INPUT.....	221
INPUTBIN.....	222
INPUTHEX.....	223
INSTR().....	224
INT() (Nowość w wersji 1.11.6.8).....	225
ISCHARWAITING() (Nowość w wersji 1.11.6.9).....	225
LCASE().....	226
LCD.....	227
LCDAT (Nowość w wersji 1.11.6.9).....	229
LEFT().....	229
LEN().....	230
LINE (Nowość w wersji 1.11.6.8).....	230
LOAD.....	232
LOADADR.....	232
LOADLABEL() (Nowość w wersji 1.11.6.9).....	233
LOCAL.....	233
LOCATE.....	235
LOG().....	235
LOG10() (Nowość w wersji 1.11.6.8).....	235
LOOKDOWN().....	236
LOOKUP().....	237
LOOKUPSTR().....	238
LOW().....	238
LOWERLINE.....	238
LTRIM().....	239
MAKEBCD().....	239
MAKEDEC().....	240
MAKEINT().....	240

MAX()	241
MID()	242
MID	242
MIN()	243
ON INTERRUPT	244
ON VALUE	247
OPEN	248
OUT	251
PEEK()	252
POKE	252
POPALL	253
POWER() (Nowość w wersji 1.11.6.8)	253
POWERDOWN	254
POWERSAVE	254
PRINT	254
PRINTBIN	255
PSET	256
PULSEIN	258
PULSEOUT	259
PUSHALL	259
RAD2DEG() (Nowość w wersji 1.11.6.8)	259
RC5SEND (Nowość w wersji 1.11.6.8)	260
RC6SEND (Nowość w wersji 1.11.6.9)	261
READ	263
READEEPROM	264
READMAGCARD	266
REM	268
RESET	268
RESTORE	269
RETURN	270
RIGHT()	270
RND()	271
ROTATE	271
ROUND() (Nowość w wersji 1.11.6.8)	272
RTRIM()	272
SECELAPSED() (Nowość w wersji 1.11.7.3)	273
SECOFDAY() (Nowość w wersji 1.11.7.3)	274
SELECT CASE...CASE...END SELECT	275
SET	276
SETFONT (Nowość w wersji 1.11.6.9)	277
SERIN (Nowość w wersji 1.11.6.9)	277
SEROUT (Nowość w wersji 1.11.6.9)	279
SGN()	281
SHIFT	281
SHIFTCURSOR	282

SHIFTIN	282
SHIFTOUT	284
SHIFTLCD	285
SHOWPIC	285
SHOWPICE (Nowość w wersji 1.11.6.8)	287
SIN() (Nowość w wersji 1.11.6.8)	288
SINH() (Nowość w wersji 1.11.6.8)	288
SONYSEND (Nowość w wersji 1.11.6.8)	289
SOUND	291
SPACE()	291
SPC()	292
SPIIN	292
SPIINIT	293
SPIMOVE()	293
SPIOUT	294
SQR() (Nowość w wersji 1.11.6.8)	294
START	295
STCHECK	296
STOP	300
STR()	302
STRING()	302
SUB	303
SWAP	303
SYSDAY() (Nowość w wersji 1.11.7.3)	303
SYSSEC() (Nowość w wersji 1.11.7.3)	305
SYSSECELAPSED() (Nowość w wersji 1.11.7.3)	306
TAN() (Nowość w wersji 1.11.6.8)	307
TANH() (Nowość w wersji 1.11.6.8)	308
THIRDLIN	308
TIME\$	308
TIME() (Nowość w wersji 1.11.7.3)	310
TOGGLE	311
TRIM()	312
UCASE()	312
UPPERLINE	313
VAL()	313
VARPTR()	314
WAIT	314
WAITKEY()	315
WAITMS	315
WAITUS	316
WHILE...WEND	317
WRITEEEPROM	317
X10DETECT (Nowość w wersji 1.11.7.3)	319
X10SEND (Nowość w wersji 1.11.7.3)	320

Wstawki assemblerowe	322
Lista rozkazów procesorów AVR.....	326
 Biblioteki.....	 330
Biblioteka AT_EMULATOR (Nowość w wersji 1.11.7.3).....	330
CONFIG ATEMU (Nowość w wersji 1.11.7.3)	330
SENDSCANKBD (Nowość w wersji 1.11.7.3).....	332
Biblioteka BCCARD.....	334
CONFIG BCCARD.....	335
BCRESET.....	335
BCDEF.....	336
BCCALL.....	337
Biblioteka DATETIME (Nowość w wersji 1.11.7.3).....	343
Biblioteka EUROTIMEDATE (Nowość w wersji 1.11.6.9).....	343
Biblioteka FP_TRIG (Nowość w wersji 1.11.6.8).....	344
Biblioteka GLCD (Nowość w wersji 1.11.6.8).....	346
Biblioteka GLCDSED (Nowość w wersji 1.11.6.9).....	346
Biblioteka LCD4.....	347
Biblioteka LCD4E2	347
Biblioteka LCD4BUSY	348
Biblioteka MCSBYTE.....	349
Biblioteka MCSBYTEINT.....	349
Biblioteka PS2MOUSE_EMULATOR (Nowość w wersji 1.11.7.3)	350
CONFIG PS2EMU (Nowość w wersji 1.11.7.3)	350
PS2MOUSEXY (Nowość w wersji 1.11.7.3)	351
SENDSCAN (Nowość w wersji 1.11.7.3).....	352
Biblioteka SPISLAVE (Nowość w wersji 1.11.6.8).....	353
Biblioteka TCPIP (Nowość w wersji 1.11.7.3)	354
CONFIG TCPIP (Nowość w wersji 1.11.7.3)	355
BASE64DEC() (Nowość w wersji 1.11.7.3)	356
CLOSESOCKET (Nowość w wersji 1.11.7.3).....	357
GETDSTIP() (Nowość w wersji 1.11.7.3).....	357
GETDSTPORT() (Nowość w wersji 1.11.7.3).....	358
GETSOCKET() (Nowość w wersji 1.11.7.3)	358
SOCKETCONNECT() (Nowość w wersji 1.11.7.3)	359
SOCKETLISTEN (Nowość w wersji 1.11.7.3).....	360
SOCKETSTAT() (Nowość w wersji 1.11.7.3).....	360
TCPREAD() (Nowość w wersji 1.11.7.3)	362
TCPWRITE() (Nowość w wersji 1.11.7.3).....	362
TCPWRITESTR() (Nowość w wersji 1.11.7.3).....	363
UDPREAD() (Nowość w wersji 1.11.7.3).....	367
UDPWRITE() (Nowość w wersji 1.11.7.3)	368
UDPWRITESTR() (Nowość w wersji 1.11.7.3)	369

Od tłumacza.

Jest to finalna wersja, przeznaczona dla kompilatora w wersji 1.11.7.3, który niedawno ukazał się na stronie MCS Electronics www.mcselec.com. W chwili której piszę te słowa nie jest jeszcze dostępna wersja DEMO.

Tekst ten jest w 99% identyczny z tym co zawarto w pliku pomocy. Wersja ta jest przeznaczona dla osób, które wolą mieć całość informacji na papierze. Ponieważ wydruk pliku pomocy jest sprawą dość kłopotliwą, powstał ten skompilowany tekst. Myślę, że format PDF nie sprawi nikomu problemu.

Podziękowania.

Dziękuję za dotychczasową korespondencję jaką otrzymałem w sprawie tłumaczenia. Czekam na dalszą. Zwłaszcza tą dotyczącą zauważonych błędów czy nieścisłości. Będę wdzięczny za wszelkie tego typu informacje.

Korespondencja w innych sprawach dotyczących języka BASCOM także jest mile widziana. Postaram się odpowiedzieć na każdy list.

I would like to thank BASCOM author: Mark Alberts – for support and official released versions which ones published on the MCS Electronics website. Really thanks Mark!

Podziękowania należą się także redakcji miesięcznika „Elektronika dla Wszystkich”, która także umieszcza Moją pracę na łamach swojej witryny internetowej www.edw.com.pl. (Patrz dział FTP).

Marteenez - Tobie także należą się podziękowania za przetłumaczenie niektórych „poważnie zakręconych” zdań.

Jurek M. – Podziękowania za kompilację do formatu PDF.

Zbigniew Gibek
zbeegin@poczta.onet.pl

BASCOM AVR - Tworzenie programu

- Uruchom środowisko BASCOM AVR;
- Otwórz plik programu lub utwórz nowy;
- Sprawdź czy ustawienia konfiguracji są zgodne z założonymi;
- Zapisz plik;
- Dokonaj kompilacji;
- Jeśli wystąpiły jakieś błędy, popraw je i skompiluj ponownie;
- Uruchom symulację;
- Jeśli program nie działa zgodnie z oczekiwaniami popraw tekst programu i powtórz operację kompilacji i symulacji.
- Zaprogramuj układ i przetestuj w budowanym urządzeniu;

BASCOM AVR i pamięć.

Każda ze zmiennych używa pewnego obszaru pamięci. Domyślnie jest to wewnętrzna pamięć danych zwana: **SRAM**. Ilość tej pamięci jest ściśle określona i zależna od konstrukcji poszczególnych procesorów AVR.

Specjalnym obszarem pamięci **SRAM** jest obszar zajmowany przez rejestry uniwersalne. Rejestry te ponumerowane od R0 do R31, zajmują dokładnie pierwsze 32 komórki tej pamięci (adresy 0-31). Rejestry te w różnym stopniu używane są przez instrukcję języka BASCOM BASIC.

Drugim specjalnym obszarem jest przestrzeń **SFR** (*Special Function Registers*), która także rozciąga się od adresu 0 do &H3F. Jest ona „niewidoczna”, gdyż przykryta jest pamięcią SRAM. Tylko specjalne rozkazy mają dostęp do tej pamięci. Niektóre z jej komórek mogą być dostępne również w trybie bitowym. Wtedy każdy bit w bajcie ma swój niepowtarzalny adres.

Do czego używana jest pamięć SRAM.

Reszta pamięci **SRAM** – tzn. ta która nie jest zajęta przez rejestry i zmienne – nie jest w zasadzie używana przez kompilator. Obszar tej pamięci zajmuje jedynie stos sprzętowy i programowy oraz tzw. ramka. Wielkość tych obszarów zmienia się dynamicznie podczas działania programu.

Niektóre z instrukcji mogą używać przestrzeni pamięci **SRAM** na własne potrzeby. Jest to wyraźnie zaznaczone przy opisie konkretnych instrukcji w pliku pomocy.

Wracając do zmiennych, to każda z nich zajmuje pewien obszar pamięci, którego rozmiar jest ściśle określony na podstawie jej typu. I tak:

- każda zmienna bitowa zajmuje jeden bit z bajtu. Gdy jest ich 8 cały bajt jest wypełniony.
- każda zmienna typu Byte zajmuje 1 bajt.
- każda zmienna typu Integer lub Word zajmuje dwa bajty.
- każda zmienna Long lub Single zajmuje 4 bajty.
- każda zmienna typu String zajmuje tyle bajtów, ile przypada na jej długość, plus 1 bajt – znak końca.
- każda zmienna tablicowa zajmuje tyle bajtów z ilu komórek się składa, pomnożonych przez ilość bajtów jaką zajmuje jedna komórka.

By zatem oszczędnie gospodarować pamięcią należy tam gdzie jest to możliwe, używać zmiennych bitowych lub bajtów. Gdy wymagane są liczby ujemne należy używać typu Integer.

Stos programowy.

Stos programowy jest używany do przechowywania adresów parametrów procedur i funkcji oraz ich zmiennych lokalnych.

Dla każdej zmiennej lokalnej lub parametru, używane są 2 bajty do zapamiętania jej adresu w pamięci. Tak więc, gdy procedura lub funkcja posiada 10 parametrów, na stosie odłożonych jest wtedy $10 \cdot 2 = 20$ bajtów. Gdy do tego procedura posiada na przykład 2 zmienne lokalne, to obszar ten powiększany jest o 4 bajty. Co w sumie daje 24 bajty.

Wymagany rozmiar stosu może być łatwo obliczony. Należy policzyć ile maksymalnie parametrów występuje w procedurach lub funkcjach, potem dodać do tego liczbę jej zmiennych lokalnych i pomnożyć otrzymaną liczbę przez 2. Dla bezpieczeństwa należy jeszcze dodać 4 bajty na zapas.

Ramka.

Zmienne lokalne trafiają do obszaru tzw. ramki. Dla przykładu, gdy procedura używa lokalnej zmiennej typu String o długości 40 znaków oraz jedną zmienną typu Long; zapotrzebowanie pamięci na ramkę wynosi: $41 + 4 = 45$ bajtów.

Gdy w programie używane są funkcje dokonujące konwersji liczb na postać tekstową, na przykład: **STR()**, **VAL()** itp.; to wykorzystują one obszar ramki jako pamięć roboczą. Zwykle potrzebują

16 bajtów tej pamięci.

Reszta przestrzeni adresowej ramki, jest wykorzystywana jako dane lokalne – dla zmiennych.

Uwaga! Instrukcja **INPUT** przyjmująca dane liczbowe przez port szeregowy, lub instrukcje **PRINT** czy **LCD** drukujące liczby, także wykorzystują 16 bajtów ramki, podczas wewnętrznej konwersji liczb na ich postać tekstową i odwrotnie.

Pamięć XRAM

Do procesora AT90s8515 (lub jego „młodsze brata” AT90s4414), można w prosty sposób dołączyć zewnętrzną pamięć danych zwaną **XRAM**.

Gdy - przykładowo - dołączona pamięć będzie miała rozmiar 32KB, jej pierwsza komórka będzie miała adres 0, lecz początkowa część tej pamięci zostanie przykryta przez pamięć SRAM. Tak więc pierwsza dostępna komórka pamięci **XRAM** będzie miała adres **&H260** (dla AT90s8515).

Związane jest to z konstrukcją samego procesora, a nie ograniczeniami języka BASCOM AVR. Projektanci z firmy Atmel założyli, że przestrzeń adresowa pamięci danych będzie liniowa. Co pozwoliło jeszcze bardziej zmniejszyć liczbę rozkazów (AVR jest procesorem RISC o architekturze harwardzkiej! *przyp. tłumacza*).

Pamięć ERAM.

Większość procesorów serii AVR posiada wbudowaną pamięć EEPROM. Pamięć ta może przechowywać dane nawet po wyłączeniu zasilania. Jak podaje producent nawet do 10 lat.

Pamięć tą w języku BASCOM AVR oznaczono skrótem **ERAM**.

Pamięć **ERAM** może być używana jak normalna pamięć, w której można umieszczać dane lub zmienne. Jednak należy uważać by nie stosować zmiennych w **ERAM**, do których często zapisywane będą dane - np. zmienna sterująca pętlą. Dzieje się tak dlatego, iż nominalnie pamięć EEPROM ma ograniczoną możliwość przeprogramowywania. Producent gwarantuje tylko 100 tys. operacji zapisu. Łatwo więc w tym przypadku o przekroczenie tej liczby w dość krótkim czasie.

Dlatego nie należy pochopnie używać tej pamięci, i w żadnym wypadku nie w instrukcjach pętli!

Stałe.

Wszystkie stałe są zapamiętane w specjalnie przeznaczonej do tego celu tablicy. Jest ona oczywiście umieszczona w pamięci kodu.

Podczas kompilacji jest dokonywana prosta optymalizacja, polegająca na wykrywaniu powtórzeń stałych. Popatrzmy na przykład:

```
Print "ABCD"  
Print "ABCD"
```

W powyższym przykładzie tylko pierwsza stała ("ABCD") jest zapisana w pamięci, lecz tutaj:

```
Print "ABCD"  
Print "ABC"
```

zapamiętane są obie stałe, gdyż nie są one takie same, choć znacznie podobne.

BASCOM AVR - Kody błędów.

Poniższa tabela zawiera listę błędów mogących się pojawić podczas sprawdzania składni lub kompilacji.

Kod błędu	Opis
1	Nieznana instrukcja
2	Nieznana struktura instrukcji EXIT
3	Spodziewano się WHILE
4	Brak miejsca w pamięci IRAM na zmienną typu Bit
5	Brak miejsca na zmienne typu Bit
6	Spodziewana . (kropka) w nazwie pliku.
7	Spodziewana instrukcja IF..THEN
8	Pliku źródłowego nie odnaleziono
9	Maksymalnie można użyć 128 instrukcji ALIAS
10	Nieznany typ wyświetlacza
11	Spodziewano się INPUT, OUTPUT, 0 lub 1
12	Nieznany parametr instrukcji CONFIG
13	Ta stała już jest zdefiniowana
14	Bajty mogą być tylko w IRAM
15	Błędny typ danych
16	Nieznana definicja
17	Spodziewano się 9 parametrów
18	Zmienne bitowe umieszczone mogą być tylko w pamięci SRAM lub IRAM
19	Spodziewano się określenia długości zmiennej typu String
20	Nieznany typ danych
21	Brak wolnej pamięci IRAM
22	Brak wolnej pamięci SRAM
23	Brak wolnej pamięci XRAM
24	Brak wolnej pamięci EEPROM
25	Ta zmienna już jest zdefiniowana
26	Spodziewano się AS
27	Spodziewano się parametru
28	Spodziewano się IF..THEN
29	Spodziewano się SELECT..CASE
30	Zmienne bitowe są zmiennymi globalnymi, nie można ich usuwać
31	Błędny typ danych
32	Niezdefiniowana zmienna
33	Zmienne globalne nie mogą być usuwane
34	Błędna ilość parametrów
35	Spodziewano się 3 parametrów
36	Spodziewano się THEN
37	Błędny operator relacji
38	Nie można wykonać tej operacji dla zmiennych bitowych
39	Spodziewano się FOR
40	Ta zmienna nie może być parametrem instrukcji RESET
41	Ta zmienna nie może być parametrem instrukcji SET
42	Spodziewano się liczby jako parametru
43	Pliku nie odnaleziono
44	Spodziewano się 2 zmiennych
45	Spodziewano się DO
46	Błędne przypisanie
47	Spodziewano się UNTIL
50	Liczba nie mieści się w zmiennej Integer
51	Liczba nie mieści się w zmiennej Word
52	Liczba nie mieści się w zmiennej Long
60	Ta etykieta już istnieje

Kod błędu	Opis
61	Etykiety nie znaleziono
62	Najpierw SUB lub FUNCTION
63	Parametrem funkcji ABS() może być liczba typu Integer lub Long
64	Spodziewany , (przecinek)
65	Urządzenie nie zostało otwarte
66	Urządzenie już jest otwarte
68	Spodziewano się numeru kanału
70	Ta szybkość transmisji nie może być użyta
71	Typ przekazanych parametrów nie jest zgodny z zadeklarowanym
72	Getclass error. Jest to błąd wewnętrzny.
73	Używanie PRINT w połączeniu z tą funkcją jeszcze nie działa
74	Spodziewano się 3 parametrów
80	Kod nie mieści się w pamięci tego układu
81	Użyj funkcji HEX() zamiast PRINTHEX
82	Użyj funkcji HEX() zamiast LCDHEX
85	Nieznane źródło przerwania
86	Błędny parametr w instrukcji CONFIG TIMER
87	Nazwa podana jako parametr instrukcji ALIAS już jest używana
88	Spodziewano się 0 lub 1
89	Liczba musi zawierać się w przedziale 1 - 4
90	Ten adres jest za duży
91	Spodziewano się INPUT , OUTPUT , BINARY lub RANDOM
92	Spodziewano się LEFT lub RIGHT
93	Niezdefiniowana zmienna
94	Podano zbyt dużo bitów
95	Spodziewano się FALLING albo RISING
96	Stopień podziału preskalera musi być jednym z podanych: 1, 8, 64, 256 lub 1024
97	Procedura lub funkcja musi być wcześniej zadeklarowana przez DECLARE
98	Spodziewano się SET lub RESET
99	Spodziewano się nazwy typu
100	Zmienne tablicowe nie mogą być umieszczone w pamięci IRAM
101	Nie mogę znaleźć takiej nazwy sprzętowego rejestru
102	Błąd w wewnętrznej procedurze
103	Spodziewano się = (znak równości)
104	Nie potrafię załadować rejestru
105	Nie potrafię zapisać wartości bitowej
106	Nieznany rejestr
107	LoadnumValue error
108	Nieznana dyrektywa w pliku definicji rejestrów
109	Spodziewano się znaku = w pseudoinstrukcji .EQU , w dołączanym pliku
110	Nie znaleziono pliku do dołączenia
111	Procedura lub funkcja nie została zadeklarowana przez DECLARE
112	Spodziewano się nazwy procedury lub funkcji
113	Ta procedura jest już zadeklarowana
114	Zmienne lokalne mogą być definiowane tylko w treści procedury lub funkcji
115	Spodziewano się numeru kanału
116	Błędny plik rejestrów
117	Nieznane źródło przerwania
200	Pliku definicji .DEF nie odnaleziono
201	Spodziewano się rejestru wskaźnikowego
202	Nie odnaleziono pseudoinstrukcji .EQU , prawdopodobnie ta funkcja nie jest obsługiwana przez wybrany procesor
203	Błąd w instrukcji LD lub LDD

Kod błędu	Opis
204	Błąd w instrukcji ST lub STD
205	Spodziewano się } (klamra zamykająca)
206	Podanej biblioteki nie odnaleziono
207	Biblioteka została już zarejestrowana
210	Nie znaleziono definicji tego bitu
211	Nie znaleziono zewnętrznej procedury
212	Spodziewano się LOW LEVEL , RISING lub FALLING
213	Spodziewano się ciągu znaków
214	Długość zmiennej String w pamięci XRAM wynosi 0
215	Nieznany skrót mnemoniczny
216	Stała nie została zdefiniowana
217	Zmienne typu BIT lub Boolean nie mogą być łączone w tablice.
218	Rejestr musi być z zakresu R16-R31
219	Przerwania INT0-INT3 są zawsze wyzwalane niskim poziomem logicznym w procesorach MEGA AVR.
220	Skok do przodu poza dozwolonym zakresem
221	Skok do tyłu poza dozwolonym zakresem
222	Błędny znak
223	Spodziewano się * (gwiazdki)
224	Indeks spoza zakresu
225	Nawiasy nie mogą występować w opisach stałych
226	Spodziewano się stałej numerycznej lub znakowej
227	Adres początkowy pamięci SRAM jest większy niż jej adres końcowy
228	Linie DATA muszą być umieszczone poza programem, po instrukcji END
229	Spodziewano się END SUB lub END FUNCTION
230	Nie możesz zapisywać do rejestru wejściowego końcówek portu (PINx)
231	Spodziewano się TO
232	Ta funkcja nie jest obsługiwana w tym procesorze
233	Instrukcja READ nie działa z danymi umieszczonymi w pamięci EEPROM
234	Spodziewano się instrukcji otwarcia bloku komentarza: ')
235	Spodziewano się instrukcji zamknięcia bloku komentarza: '(
236	Liczba nie mieści się w zmiennej typu Byte
238	Ta zmienna nie jest zmienną tablicową
239	Invalid code sequence because of AVR hardware bug
240	Spodziewano się END FUNCTION
241	Spodziewano się END SUB
242	Wystąpił brak zgodności zmiennych
243	Numer bitu wykracza poza liczbę dopuszczalną dla tej zmiennej
244	Nie możesz używać wskaźnika Y
245	Zmienne tablicowe nie mogą być w pamięci IRAM
246	Brak miejsca na definicję w pliku .DEF
247	Spodziewano się kropki
248	Powinien być użyty argument BYVAL w tej deklaracji
249	Procedura obsługi przerwania jest już zdefiniowana
250	Spodziewano się GOSUB
251	Ta etykieta musi być nazwana SECTIC
252	Spodziewano się zmiennej Integer lub Word
253	Ta zmienna nie może być w pamięci ERAM
254	Spodziewana zmienna
255	Spodziewano się Z lub Z+
256	Spodziewano się zmiennej typu Single
257	Spodziewano się ""
258	Spodziewano się SRAM
259	Zmienne typu Byte nie mogą przyjmować wartości ujemnych
260	Ciąg znaków nie zmieści się w tej zmiennej typu String

Kod błędu	Opis
261	Spodziewano się tablicy
262	Spodziewano się ON lub OFF
263	Indeks tablicy poza zakresem
264	Zamiast tego użyj ECHO OFF i ECHO ON
265	Spodziewano się offsetu w rozkazie LDD lub STD . Np. Z+1
266	Spodziewano się TIMER0 , TIMER1 lub TIMER2
267	Spodziewano się stałej liczbowej
268	Parametr musi zawierać się w granicach 0 – 3
269	Spodziewano się END SELECT
270	Ten adres już jest zajęty
322	Ten typ danych nie jest obsługiwany przez tą instrukcję
232	Etykieta posiada zbyt dużo znaków
234	Ten układ nie jest obsługiwany przez bibliotekę I2C w trybie Slave
325	Stopień podziału preskalera musi wynosić: 1, 8, 32, 128, 256 lub 1024
326	Spodziewano się #ENDIF
327	Maksymalna wielkość to 255
328	Nie działa z programowym układem UART
999	Wersje DEMO lub BETA generują kod tylko do 2 KB

Wszystkie inne kody, nie wymienione powyżej są rezultatem błędów wewnętrznych. Jeśli takie się pojawią proszę o stosowną informację.

Uwaga! Często zdarza się że kompilator raportuje błąd „**File not found**”, który jest zwykle spowodowany przez błędne określenie parametrów instrukcji - zwłaszcza **CONFIG**. (przyp. tłumacza)

Urządzenia wbudowane w strukturę procesora AVR

Wszystkie procesory serii AVR posiadają pewną ilość wbudowanych urządzeń, mogących być użytych przez program.

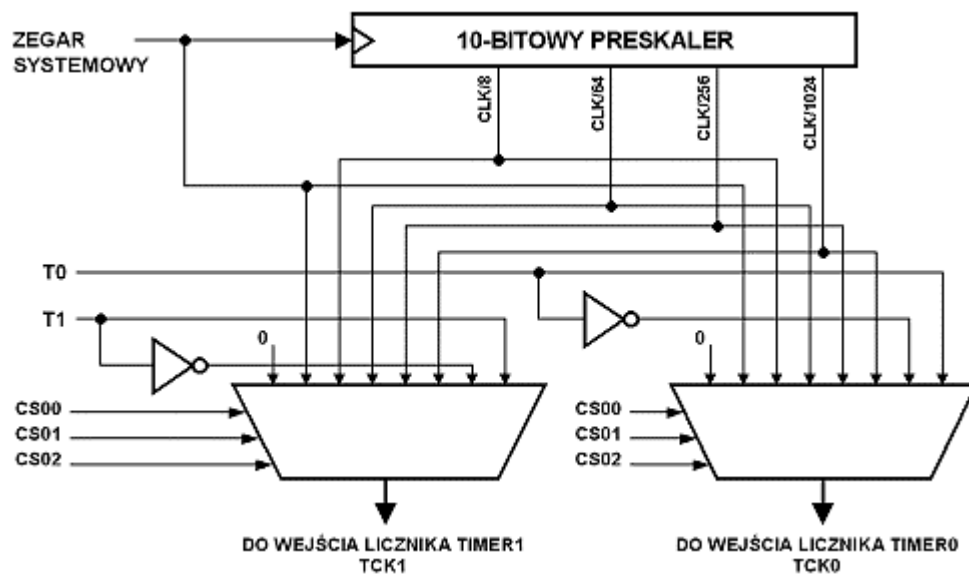
Podstawą opisu jest procesor AT90s8515, dlatego też niektóre elementy mogą nie występować w procesorach o mniejszych możliwościach, jak na przykład AT90s1200 czy AT90s2313.

Uniwersalne Liczniki – Czasomierze.

Kontroler AT90s8515 posiada dwa liczniki-czasomierze, które mogą być użyte w dowolny z możliwych sposobów. Licznik TIMER0 jest 8-bitowy i jest prostszy, drugi z nich TIMER1 jest już 16 bitowy i posiada kilka zaawansowanych funkcji.

Sygnał zegarowy dołączany do wejść liczników, przechodzi przez 10-bitowy preskaler (dzielnik wejściowy). Liczniki też mogą być dołączone do końcówek portów, co pozwala na zliczanie impulsów zewnętrznych.

Układ przełączający i preskaler jest rozrysowany poniżej:



Rysunek 1 Układ przełączający i preskaler.

Układ Watchdog.

Konstruktorzy wychodząc naprzeciw najnowszym trendom w konstruowaniu mikrokontrolerów, wbudowali do procesorów AVR układ Watchdog.

Jest to specjalny licznik, zliczający impulsy zegarowe 1MHz. Gdy nastąpi przepełnienie tego licznika, generowany jest sygnał reset by wyzerować procesor.

Do programisty zatem należy umieszczenie w programie rozkazów powodujących zerowanie tegoż licznika. Jest to jeden z elementów zabezpieczenia przed zapętleniem się programu lub błędami w jego działaniu.

Porty.

Prawie wszystkie procesory z serii AVR posiadają porty nazwane **PORTB** i **PORTD**. Układy w obudowach 40 końcówkowych (i większych) mają także porty **PORTA** i **PORTC**, które są używane także jako systemowa szyna danych i adresowa (multipleksowana). Porty te zasadniczo nie różnią się od portów PORTB lub PORTD.

Ponieważ końcówki portów PORTB i PORTD posiadają jeszcze specjalne (alternatywne) funkcje, zostaną one opisane dokładniej.

Układ transmisji szeregowej.

Procesory serii AVR (nie wszystkie!) posiadają wbudowany układ transmisji szeregowej

UART. Może on pracować w trybie full-duplex oraz posiada niezależny układ taktujący. Zwalnia to liczniki-czasomierze z generowania tego sygnału (porównaj z procesorami 8051).

Do zalet należy także zaliczyć układ eliminacji zakłóceń – przez wielokrotne próbkowanie bitów – oraz układ wykrywania błędów transmisji: błąd ramki (przepełnienie rejestru przesuwającego), błąd bitu startu.

Interfejs SPI.

Procesory AVR (także nie wszystkie!) oraz dwa z rodziny 8051 posiadają układ SPI służący do szybkiej, szeregowej transmisji danych pomiędzy procesorami lub też procesorami i urządzeniami zewnętrznymi. Pracuje on na zasadzie wymiany danych, tj. nadaje i jednocześnie odbiera jeden bajt.

Jest on także używany do programowania wewnętrznej pamięci Flash oraz EEPROM w trybie ISP (*In System Programming*).

Komparator analogowy.

Procesor AVR podobnie jak procesory z rodziny 8051 produkowanej przez Atmel-a posiada wewnętrzny komparator analogowy. Może on służyć do porównywania dwóch napięć, a przy zastosowaniu odpowiedniego programu i prostego układu RC także do ich pomiaru.

Licznik-czasomierz TIMER0

Licznik TIMER0 jest 8 bitowy. Może on zliczać impulsy zegara taktującego procesor doprowadzone do jego wejścia bezpośrednio lub przez preskaler. Może też z powodzeniem zliczać impulsy doprowadzone do jednej z końcówek portów. Zliczanie można w każdej chwili zatrzymać i wznowić.

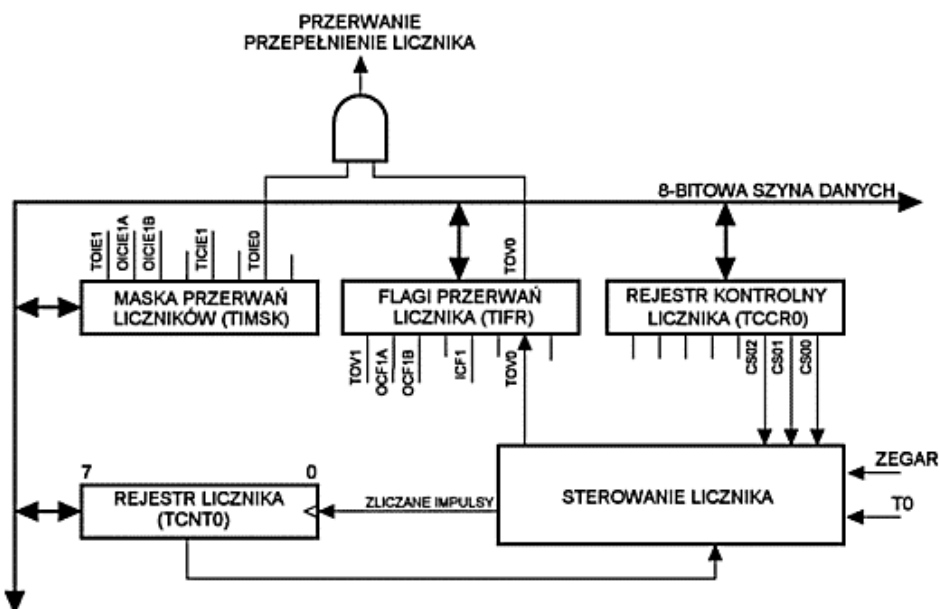
Licznik może być źródłem przerwań. Flaga **TOV0** jest ustawiana jeśli zostało stwierdzone przepełnienie licznika. Znajduje się ona w rejestrze **TIFR** (*Timer/Counter Interrupt Flag Register*). Aby wyłączyć generowanie przerwań licznika, należy ustawić odpowiedni bit w rejestrze **TIMSK** (*Timer/Counter Interrupt Mask Register*).

Tryb pracy licznika ustala się ustawiając odpowiednie bity w rejestrze **TCCR0** (*Timer0/Counter0 Control Register*).

Gdy zliczane są zewnętrzne impulsy, ich próbkowanie jest zsynchronizowane z zegarem systemowym. Zatem aby impulsy te nie były gubione, czas pomiędzy kolejnymi impulsami musi być nie krótszy niż dwa pełne takty zegara systemowego.

Próbkowanie odbywa się podczas narastającego zbocza sygnału zegarowego.

Poniżej znajduje się poglądowy schemat licznika-czasomierza TIMER0.



Rysunek 2 Licznik-czasomierz TIMER0.

Na zakończenie.

Licznik czasomierz TIMER0 cechuje się dużą rozdzielczością i wysoką dokładnością gdy używany jest przy małych stopniach podziału preskalera. Podobnie, przy dużym podziale preskalera licznik staje się użyteczny przy odmierzaniu krótkich odcinków czasu.

Konfigurację pracy licznika zajmuje się instrukcja **CONFIG TIMER0**. Do sterowania licznikiem przewidziano instrukcje **START** oraz **STOP**.

Uproszczono także dostęp do rejestrów licznika definiując w języku BASCOM BASIC specjalną zmienną **COUNTER0**. W celu załadowania wartości do licznika można użyć specjalnej instrukcji **LOAD**, która dokonuje niezbędnego przeliczenia tej wartości, tak aby licznik przepełnił się po podanej liczbie impulsów.

Przewidziano także stosowanie przez użytkownika przerwań jakie generuje licznik. Można je łatwo obsłużyć stosując instrukcję **ON INTERRUPT** i odpowiedni program obsługi.

Licznik-czasomierz **TIMER1**

Licznik **TIMER1** jest 16 bitowy i może zliczać impulsy zegara taktującego procesor doprowadzone do jego wejścia bezpośrednio lub przez preskaler. Może też z powodzeniem zliczać impulsy doprowadzone do jednej z końcówek portów. Zliczanie można w każdej chwili zatrzymać i wznowić.

Licznik ten podobnie jak **TIMER0** może generować przerwania, ustawiana jest wtedy flaga **TOV1**. Włączenie przerwań licznika **TIMER1** spowoduje ustawienie odpowiedniej flagi w rejestrze **TIMSK** (*Timer1/Counter1 Interrupt Mask Register*).

Konfiguracji pracy licznika dokonujemy ustawiając odpowiednie bity w rejestrach **TCCR1A** i **TCCR1B** (*Timer1/Counter1 Control Register*).

Gdy zliczane są zewnętrzne impulsy, ich próbkowanie jest zsynchronizowane z zegarem systemowym. Zatem aby impulsy te nie były gubione, czas pomiędzy kolejnymi impulsami musi być nie krótszy niż dwa pełne takty zegara systemowego.

Próbkowanie odbywa się podczas narastającego zbocza sygnału zegarowego.

Uwaga! Poniższe funkcje opisano na podstawie licznika **TIMER1** procesora AT90s8515, w innych procesorach AVR, mogą się one różnić.

Licznik **TIMER1** posiada dwa rejestry porównywania. Nazwane są one *Output Compare Register A* (**COMPARE1A** lub **OCR1A**) i *Output Compare Register B* (**COMPARE1B** lub **OCR1B**). Gdy w wyniku porównania stwierdzono, że zawartość któregoś z nich jest identyczna z zawartością licznika, ustawiane są znaczniki **OCR1A** lub **OCR1B**.

Z rejestrami porównania jest związana jeszcze jedna funkcja: Gdy zawartość rejestru **COMPARE1A** będzie odpowiadać zawartości licznika, licznik może zostać wyzerowany.

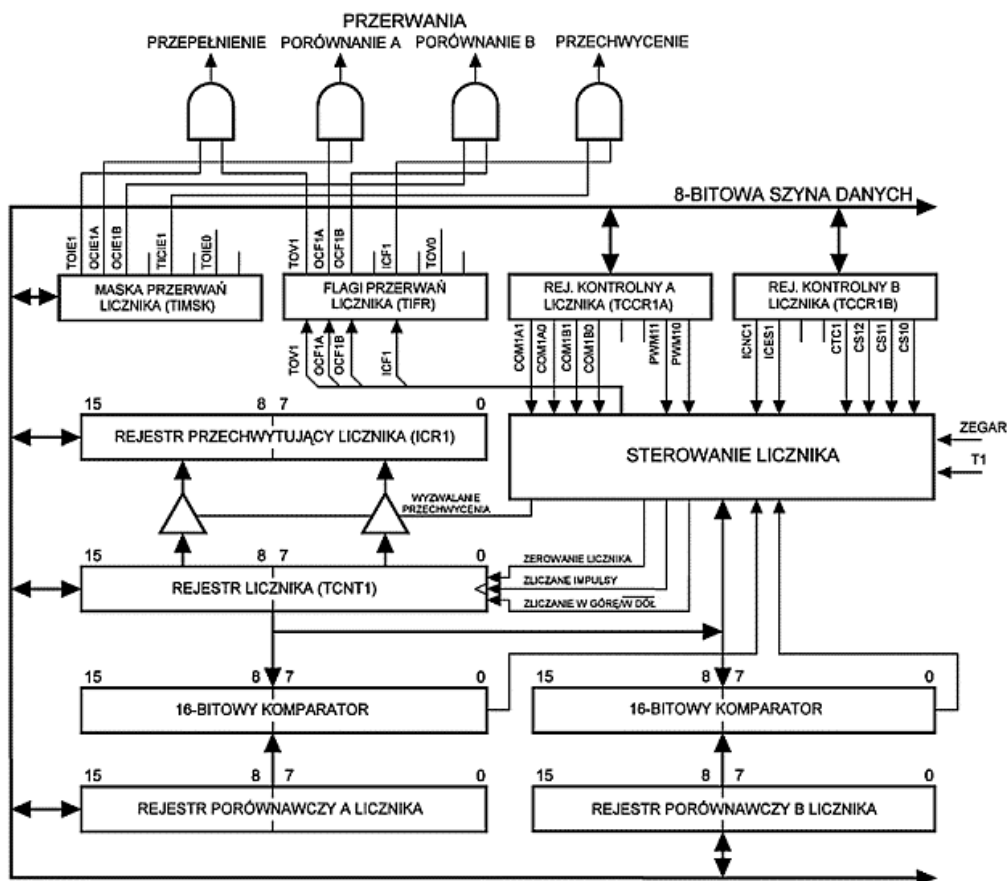
Licznik **TIMER1** może być także używany jako generator impulsów o modulowanej szerokości. Rozdzielczość generatora może być wtedy ustawiona jako 8, 9 lub 10 bitowa. W trybie tym licznik oraz rejestry **OCR1A/OCR1B** tworzą razem układ generatora impulsów PWM (*Pulse Width Modulation*).

Zmiana wypełnienia (szerokości) impulsów polega na wpisaniu odpowiednich wartości do rejestrów **OCR1A** i **OCR1B**. Wyjściem impulsów generatora PWM jest wtedy końcówka OC1.

TIMER1 posiada także rejestr oraz tryb przechwytywania. W trybie tym można w dowolnej chwili przechwycić zawartość licznika do specjalnego rejestru – **ICR1** (*Input Capture Register*). Sterowaniem przechwytywania zajmuje się specjalna końcówka ICP. Ustawienia trybu przechwytywania dokonuje się w rejestrze **TCCR1B**.

Dodatkowo można użyć wbudowanego komparatora analogowego, do kontroli stanu linii ICP.

Poniżej przedstawiono poglądowy schemat licznika-czasomierza **TIMER1**.



Rysunek 3 Licznik-czasomierz TIMER1

Na zakończenie.

Licznik czasomierz TIMER1 cechuje się wysoką rozdzielczością i dokładnością gdy używany jest przy małych stopniach podziału preskalera. Podobnie, przy dużym podziale preskalera licznik staje się użyteczny przy dokładnym odmierzeniu nawet dość długich odcinków czasu.

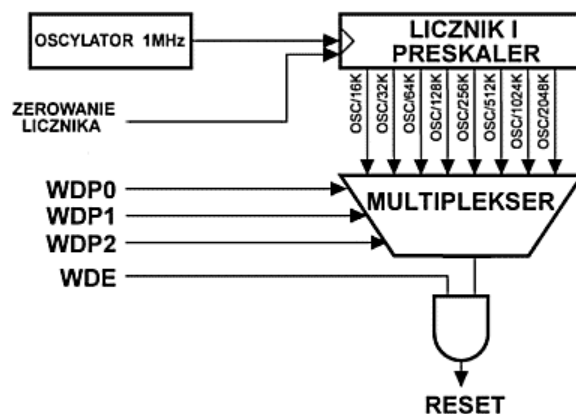
Konfigurację pracy licznika zajmuje się instrukcja **CONFIG TIMER1**. Do sterowania licznikiem przewidziano instrukcje **START** oraz **STOP**.

Uproszczono także dostęp do rejestrów licznika definiując w języku BASCOM BASIC kilka specjalnych zmiennych (**COUNTER1**, **PWM1A**, **CAPTURE1**, **COMPARE1A**). W celu załadowania wartości do licznika można użyć specjalnej instrukcji **LOAD**, która dokonuje niezbędnego przeliczenia tej wartości, tak aby licznik przepełnił się po podanej liczbie impulsów.

Przewidziano także stosowanie przez użytkownika przerwań jakie generuje licznik. Można je łatwo obsłużyć stosując instrukcję **ON INTERRUPT** w połączeniu z odpowiednim programem obsługi.

Układ Watchdog

Układ Watchdog składa się ze specjalnego licznika, zliczającego impulsy pochodzące z odrębnego generatora o częstotliwości 1MHz (przy napięciu zasilania $V_{cc}=5V$), przechodzących przez dodatkowy preskaler.



Rysunek 4 Układ Watchdog.

Stopień podziału preskalera można ustawiać w dość szerokim zakresie począwszy od dzielnika przez 16k do dzielnika przez 2048k, co daje czas opóźnienia przepełnienia licznika od 16ms do około 2 sekund. Gdy licznik się przepełni generowany jest sygnał RESET zerujący mikroprocesor.

Ustawienie stopnia podziału preskalera układu Watchdog jest możliwe za pomocą instrukcji **CONFIG WATCHDOG**. Aby wyzerować licznik układu Watchdog należy użyć instrukcji **RESET WATCHDOG**.

Sterowaniem pracą układu Watchdog zajmują się instrukcje **START** oraz **STOP**.

Port B

Port PB jest dwukierunkowym 8 bitowym portem. W przestrzeni rejestrów specjalnych są umieszczone aż trzy rejestry do obsługi portu. Pierwszy z nich to rejestr danych nazwany **PORTB** (adres **&H18** i **&H38**). Drugi rejestr **DDRB** (*Data Direction Register*, adres **&H17** i **&H37**) służy do określenia kierunku działania poszczególnych linii portu. Trzeci to rejestr wejściowy **PINB** (*Port Input Pins*, adres **&H16** i **&H36**), który odwzorowuje bezpośrednio stan końcówek portu. Rejestr **PINB** jest przeznaczony tylko do odczytu, nie można zatem wpisać tam żadnej wartości. Reszta rejestrów jest o dostępie swobodnym.

Wszystkie końcówki portu posiadają rezystory podciągające, mogące być włączane osobno dla każdej z końcówek. Port PB może także bezpośrednio sterować diodami LED, gdyż prąd wpływający może mieć wartość nawet do 20mA.

Gdy końcówki portu PB0-PB7 pracując jako wejścia są zewnętrznym ściągnięte do masy, to przy włączonym wewnętrznym podciąganiu będą źródłem prądu wypływającego.

Port PB jako wejście-wyjście funkcji alternatywnych.

Końcówki portu B mogą także pełnić alternatywne funkcje, których opis przedstawiono w tabeli:

Funkcje alternatywne końcówek portu B

Port	Końcówka	Funkcja alternatywna
PORTB.0	T0	(Timer/Counter 0 wejście impulsów zewnętrznych)
PORTB.1	T1	(Timer/Counter 1 wejście impulsów zewnętrznych)
PORTB.2	AIN0	(Komparator analogowy wejście nieodwracające)
PORTB.3	AIN1	(Komparator analogowy wejście odwracające)
PORTB.4	SS	(SPI linia Slave Select – SS)
PORTB.5	MOSI	(SPI Wejście w trybie MASTER/Wyjście w trybie SLAVE)
PORTB.6	MISO	(SPI Wyjście w trybie MASTER/Wejście w trybie SLAVE)
PORTB.7	SCK	(SPI Zegar taktujący)

Gdy końcówki portu mają pełnić rolę wejścia-wyjścia funkcji alternatywnej, rejestry **DDRB** i **PORTB** muszą być ustawione zgodnie z opisem funkcji.

Port wejściowy PINB nie jest właściwie rejestrem, a operacja odczytu włącza tylko specjalny tryb pracy portu, gdzie stan końcówek jest odczytywany wprost z wyprowadzenia. Natomiast odczytywanie zawartości rejestru **PORTB**, spowoduje odczytanie tylko stanu wewnętrznych zatrząsków.

Opis funkcji alternatywnych.

T0 – PORTB, Bit 0

Końcówka ta może być używana jako wejście zewnętrznych impulsów, których zliczaniem zajmuje się licznik TIMER0. Licznik może wtedy reagować na określony poziom logiczny lub na zmianę stanu końcówki.

T1 – PORTB, Bit 1

Końcówka ta może być używana jako wejście zewnętrznych impulsów, których zliczaniem zajmuje się licznik TIMER1. Licznik może wtedy reagować na określony poziom logiczny lub na zmianę stanu końcówki.

AIN0 (+) – PORTB, Bit 2

AIN1 (-) – PORTB, Bit 3

Końcówki te są dołączone do wejść wbudowanego w procesor komparatora analogowego, który jest źródłem przerwania. Można go wykorzystać do porównywania napięć, a stosując odpowiedni fragment programu także do ich pomiaru.

Komparator może być użyty do wyzwalania funkcji przechwytywania licznika TIMER1.

SS – PORTB, Bit 3

Wejście uaktywniające układ SPI, stosowane w systemach z kilkoma procesorami AVR komunikującymi się przez interfejs SPI.

Jest wykorzystywana także podczas programowania procesora w trybie ISP.

MOSI – PORTB, Bit 2

MISO – PORTB, Bit 1

Wejścia/wyjścia układu SPI. Są one dołączone do jego wewnętrznego rejestru przesuwającego, służącego do wymiany danych.

Są wykorzystywane także podczas programowania procesora w trybie ISP.

SCK – PORTB, Bit 0

Wejście/Wyjście impulsów taktujących transmisję przez układ SPI.

Końcówka ta wykorzystywana jest także podczas programowania procesora w trybie ISP.

Port PB jako uniwersalny port wejścia wyjścia.

Wszystkie 8 linii portu są równorzędne, gdy używane są jako zwykłe wejścia-wyjścia. Bity **DDB_n** w rejestrze **DDRB** określają kierunek działania końcówki portu. Gdy bit jest ustawiony (stan 1) końcówka pełni rolę wyjścia, gdy jest wyzerowany (stan 0) końcówka pełni rolę wejścia. Jeśli dodatkowo określony bit w rejestrze **PORTB_n** jest ustawiony (stan 1) a końcówka pełni rolę wejścia, włączany jest rezystor podciągający. By wyłączyć rezystor podciągający należy w rejestrze **PORTB** wyzerować odpowiedni bit, lub też skonfigurować linię portu jako wyjście:

<i>DDB_n</i>	<i>PORTB_n</i>	<i>Tryb pracy</i>	<i>Podciąganie</i>	<i>Komentarz</i>
0	0	Wejście	Nie	Trójstanowe (Hi-Z)
0	1	Wejście	Tak	Z linii PB _n może wypływać prąd, gdy końcówka będzie ściągnięta do masy.
1	0	Wyjście	Nie	Stopień wyjściowy typu Push-Pull, stan 0
1	1	Wyjście	Nie	Stopień wyjściowy typu Push-Pull, stan 1

Na zakończenie.

Zarówno porty jak i pojedyncze linie tychże portów mogą pracować jako wejścia lub wyjścia. Ustalenie trybu pracy jest możliwe za pomocą instrukcji **CONFIG PORT** jak i **CONFIG PIN**.

Port D

Sposób działania i konfiguracji portu PD jest prawie identyczny jak w porcie PB, więc nie będzie opisany powtórnie. Adresy rejestrów portu są następujące: **PORTD** - &H12 i &H32, **DDRD** - &H11 i &H31, **PIND** - &H10 i &H30.

Port PB jako wejście-wyjście funkcji alternatywnych.

Końcówki portu PB mogą także pełnić alternatywne funkcje, których opis przedstawiono w tabeli:

Funkcje alternatywne końcówek portu B

Port	Końcówka	Funkcja alternatywna
PORTD.0	RDX	(UART Wejście)
PORTD.1	TDX	(UART Wyjście)
PORTD.2	INT0	(Wejście zewnętrznego przerwania INT0)
PORTD.3	INT1	(Wejście zewnętrznego przerwania INT1)
PORTD.5	OC1A	(Wyjście sygnału porównania licznika TIMER1)
PORTD.6	WR	(Wyjście strobuujące zapis do zewnętrznej pamięci)
PORTD.7	RD	(Wyjście strobuujące odczyt z zewnętrznej pamięci)

Gdy końcówki portu mają pełnić rolę wejścia-wyjścia funkcji alternatywnej, rejestry **DDRB** i **PORTB** muszą być ustawione zgodnie z opisem funkcji.

Opis funkcji alternatywnych.

RD – PORTD, Bit 7

Sygnał RD służy do aktywowania układu odczytu zewnętrznej pamięci RAM.

WR – PORTD, Bit 6

Sygnał WR służy do aktywowania układu zapisu zewnętrznej pamięci RAM.

OC1 – PORTD, Bit 5

Wyjście sygnału przechwytywania licznika TIMER1. Stan tej końcówki zmienia się, gdy w wyniku porównania zawartości licznika z rejestrem porównawczym, stwierdzono ich zgodność.

Końcówka PD5 musi być ustawiona jako wyjście by pełniła tą funkcję, tzn. **DDD5** ustawiony (stan 1). Zobacz opis licznika-czasomierza TIMER1 by dowiedzieć się więcej.

Końcówka ta też może być wyjściem impulsów licznika w trybie PWM.

INT1 – PORTD, Bit 3

Wejście zewnętrznego sygnału przerwania. Końcówka PD3 może z powodzeniem obsługiwać sygnały pochodzące z urządzeń zewnętrznych, pojawiających się okresowo lub systematycznie, a wymagających akcji ze strony procesora. Zobacz informacje na temat przerw, aby uzyskać dodatkowe informacje.

INT0 – PORTD, Bit 2

Wejście zewnętrznego sygnału przerwania. Końcówka PD2 może z powodzeniem obsługiwać sygnały pochodzące z urządzeń zewnętrznych, pojawiających się okresowo lub systematycznie, a wymagających akcji ze strony procesora. Zobacz informacje na temat przerw, aby uzyskać dodatkowe informacje.

TXD – PORTD, Bit 1

Wyjście danych przesyłanych przez wewnętrzny UART. Gdy działa układ UART, końcówka portu jest automatycznie konfigurowana dla celów transmisji bitów, niezależnie od stanu bitu **DDR1**.

RXD – PORTD, Bit 0

Wejście danych odbieranych przez wewnętrzny UART. Gdy działa odbiornik układu UART, końcówka portu jest automatycznie konfigurowana dla celów transmisji bitów, niezależnie od stanu bitu **DDR1**. Dodatkowo gdy układ UART ustawił tryb pracy końcówki jako wejście, ustawienie jedynki w bicie **PORTD0**, powoduje włączenie wewnętrznego podciągania.

Gdy końcówki **TXD** oraz **RXD** nie są używane do transmisji przez UART, mogą być normalnie używane jako uniwersalne porty wejścia-wyjścia. Wtedy jednak nie można używać instrukcji **PRINT** i **INPUT**, oraz innych operujących na wbudowanym układzie UART.

Uwaga! W rejestrze **UCR** (*UART Control Register*) bity 3 i 4 nie są standardowo ustawiane, co oznacza że końcówki **TXD** i **RXD** nie są używane przez UART. Nie jest to jednak regułą. W tym wypadku lepiej w programie wyzerować te bity, stosując na przykład taki blok instrukcji:

```
Reset UCR. 3
```

```
Reset UCR. 4
```

Port PD jako uniwersalny port wejścia wyjścia.

Wszystkie 8 linii portu są równorzędne, gdy używane są jako zwykłe wejścia-wyjścia. Bity **DDb_n** w rejestrze **DDRD** określają kierunek działania końcówki portu. Gdy bit jest ustawiony (stan 1) końcówka pełni rolę wyjścia, gdy jest wyzerowany (stan 0) końcówka pełni rolę wejścia. Jeśli dodatkowo określony bit w rejestrze **PORTD_n** jest ustawiony (stan 1) a końcówka pełni rolę wejścia, włączany jest rezystor podciągający. By wyłączyć rezystor podciągający należy w rejestrze **PORTD** wyzerować odpowiedni bit, lub też skonfigurować linię portu jako wyjście:

<i>DDB_n</i>	<i>PORTD_n</i>	<i>Tryb pracy</i>	<i>Podciąganie</i>	<i>Komentarz</i>
0	0	Wejście	Nie	Trójstanowe (Hi-Z)
0	1	Wejście	Tak	Z linii PD _n może wypływać prąd, gdy końcówka będzie ściągnięta do masy.
1	0	Wyjście	Nie	Stopień wyjściowy typu Push-Pull, stan 0
1	1	Wyjście	Nie	Stopień wyjściowy typu Push-Pull, stan 1

Na zakończenie.

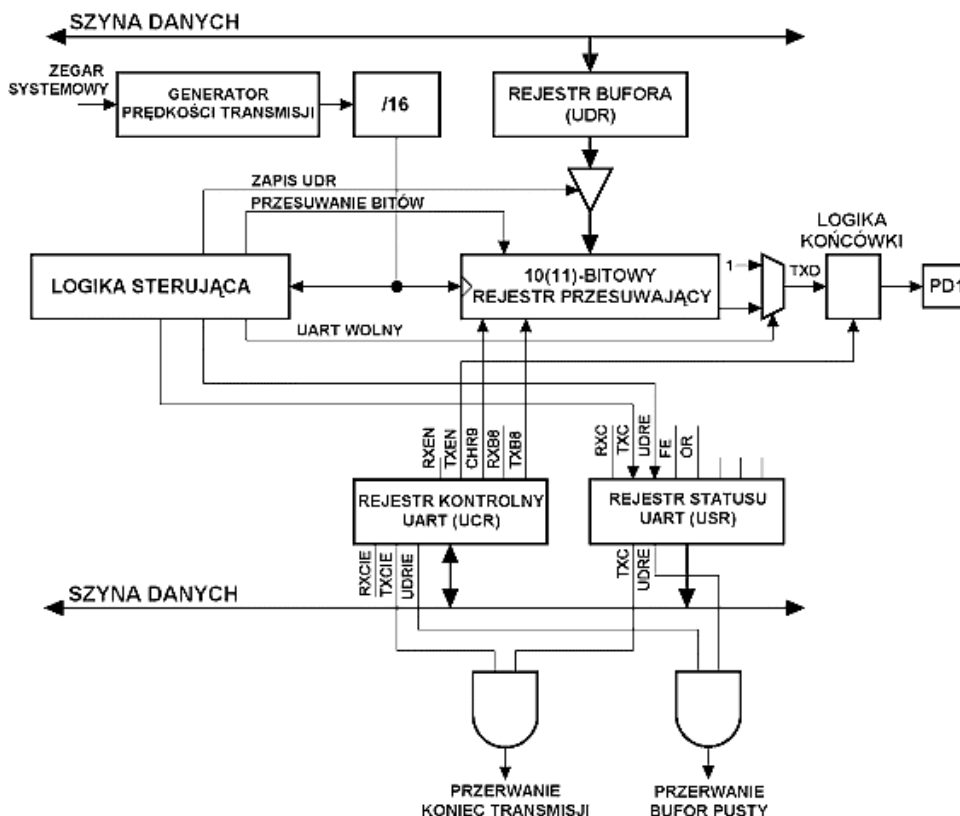
Zarówno porty jak i pojedyncze linie tychże portów mogą pracować jako wejścia lub wyjścia. Ustalenie trybu pracy jest możliwe za pomocą instrukcji **CONFIG PORT** jak i **CONFIG PIN**.

Układ transmisji szeregowej UART

Mikrokontrolery rodziny AVR posiadają układ uniwersalnego portu szeregowego, mogącego służyć do transmisji w standardzie RS232. Transmisja może się odbywać w trybie full-duplex, gdyż układ ten posiada dwa niezależne rejestry transmisyjne. Układ posiada także własny układ taktujący, co zwalnia liczniki-czasomierze z generowania tego sygnału. Jest to znaczne rozszerzenie możliwości układu UART w stosunku do popularnej rodziny kontrolerów 8051.

Cześć nadawcza.

Główną częścią układu nadajnika transmisji jest rejestr przesuwający, połączony z rejestrem bufora **UDR** (*UART Data Register*) do którego należy wpisać transmitowany bajt. Po stwierdzeniu że rej. przesuwający jest pusty, zawartość rej. **UDR** jest przepisywana do rejestru przesuwającego co rozpoczyna transmisję tego bajtu.



Rysunek 5 UART - Część nadawcza.

Ponieważ układ UART może także transmitować dane 9-bitowe, w rejestrze **UCR** (*UART Control Register*) są umieszczone bity **TX8** oraz **CHR9**. Ustawienie bitu **CHR9** służy do włączenia tego trybu, co powoduje, że równocześnie wraz z bajtem z rej. **UDR** przepisany do rej. przesuwającego będzie także stan bitu **TX8**.

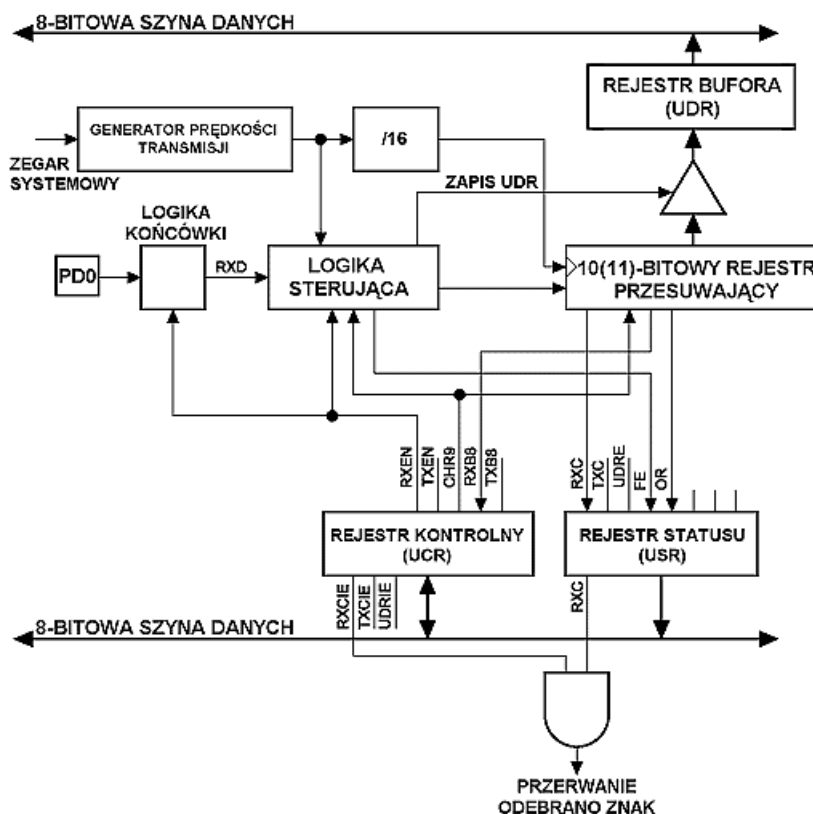
Uwaga! Stan tego bitu musi być ustalony **przed** wpisaniem wartości do rejestru **UDR**.

Przepisanie wartości z rejestru **UDR** do rej. przesuwającego powoduje także ustawienie bitu **UDRE** (*UART Data Register Empty*) co jest sygnałem, że można wpisać nową wartość do rejestru **UDR**. Zostanie ona przepisana automatycznie po zakończeniu transmisji bieżącego bajtu (wysunięciu bitu stopu). Skasowanie bitu **UDRE** może odbyć się tylko sprzętowo, poprzez zapis do rejestru **UDR**.

Jeśli po ustawieniu bitu **UDRE** nie nastąpił zapis do rejestru **UDR** to po zakończeniu transmisji bieżącego bajtu jest ustawiona flaga **TXC** (*UART Transmission Complete*). Oznacza to, że część nadawcza układu UART zakończyła transmisję i jest wolna.

Część odbiorcza.

Podobnie jak w przypadku nadajnika, główną częścią części odbiorczej jest rejestr przesuwający oraz połączony z nim rejestr bufora **UDR**. Niebagatelną rolę pełni także układ logiki, w którym umieszczono układ eliminacji zakłóceń.



Rysunek 6 UART - Część odbiorcza.

Rejestr **UDR** to właściwie dwa rejestry umieszczone pod tym samym adresem. Jeden z nich jest udostępniony tylko do zapisu (dla nadajnika), a drugi tylko do odczytu (dla odbiornika). Przy transmisji 9 bitowej ostatni odebrany bit danych (MSB) trafia do rejestru **UCR** – bit **RX8**.

Jak wspomniano na początku część odbiorcza jest wyposażona w układ eliminujący zakłócenia jakie mogą wystąpić podczas transmisji bitów. Mechanizm jego działania jest prosty. Polega on na wielokrotnym próbkowaniu stanu linii **RxD**. Jeśli logika sterująca stwierdzi, że co najmniej dwie ostatnie z trzech próbek – w środku „okna” dla każdego z bitów – są identyczne, stan linii jest uznawany za ważny i bit trafia do rejestru przesuwającego.

Podobnie dzieje się z bitem stopu, choć tutaj stwierdzenie rozbieżności podczas próbkowania powoduje ustawianie flagi **FE** (*Frame Error*), co jest sygnałem o niedostosowaniu szybkości transmisji. Ustawienie tej flagi nie powoduje jednak zaniechania odbioru tego znaku, i trafia on do rejestru **UDR**.

Odebranie znaku jest sygnalizowane przez ustawienie flagi **RXC** (*UART Receive Complete*). Wtedy do bufora **UDR** przepisywane jest 8 bitów z rejestru przesuwającego. Przy transmisji 9 bitowej ostatni odebrany bit trafia do rejestru **UCR** – bit **RX8**.

Jeśli nie odczytamy (opróżnimy) bufora przed zakończeniem odbioru następnego znaku, dane w nim umieszczone zostaną utracone, gdyż aktualnie odebrany znak trafi do bufora. W takim przypadku ustawiana jest flaga **OR** (*Buffer OverRun*). Flaga ta jest kasowana tylko sprzętowo, przez odczytanie bajtu z rej. **UDR**.

Uwaga! Flaga **RXC** jest ustawiana mimo stwierdzenia błędu ramki i przepełnienia bufora. Aby ustrzec się przed błędami transmisji najlepiej przetestować stan bitów **FE** oraz **OR**.

Sterowanie układem UART.

Obie części: nadawcza i odbiorcza korzystają z wspólnego układu generatora, służącego do taktowania transmisji. Jak już wspomniano jest on układem niezależnym, choć nadal jest zsynchronizowany impulsami zegarowymi.

Częstotliwość wyjściowa tego układu - a co za tym idzie szybkość transmisji - zależy od stopnia podziału, który jest ustalany przez wpisanie odpowiedniej liczby do 8-bitowego rejestru **UBRR**.

(*UART Baud Rate Register*). Wartości od 0 do 255 są w zupełności wystarczające dla uzyskania większości z standardowych szybkości: od 2400 do 115200 bodów. Należy jednak pamiętać o tym, że znaczny wpływ na możliwość uzyskania dokładnie(!) żądanej szybkości ma częstotliwość kwarcu. Najlepszym rozwiązaniem jest stosowanie kwarcu o częstotliwościach: 11.059MHz, 3.6864MHz lub 7.3728MHz.

Ogólny wzór na obliczenie tej wartości to: $\text{szybkość} = f_{\text{kwarcu}} / (16 * (\text{UBRR} + 1))$. Gdzie *szybkość* to żądana szybkość transmisji w bodach, a f_{kwarcu} to częstotliwość kwarcu podana w Hz.

Aby umożliwić nadawanie lub odbieranie danych, należy ustawić także odpowiednie bity w rejestrze **UCR**. Bit **TXEN** (*UART Transmitter Enable*) jest odpowiedzialny za włączenie możliwości nadawania. Ustawienie go powoduje dołączenie końcówki PD1 (TxD) do wyjścia układu nadajnika. Natomiast bit **RXEN** (*UART Receiver Enable*) jest odpowiedzialny za możliwość odbierania danych. Jego ustawienie powoduje dołączenie końcówki PD0 (RxD) do części odbiorczej układu UART.

Uwaga! Skasowanie bitu **TXEN** podczas trwania transmisji nie powoduje jej przerwania. Jeśli w rejestrze **UDR** znajduje się oczekujący bajt on także zostanie wysłany przed odłączeniem końcówki PD1 od układu UART.

Układ UART może być źródłem aż trzech przerwań: stwierdzenie opróżnienia bufora (przy nadawaniu!), zakończenie nadawania lub zakończenie odbioru nowego bajtu. Każde źródło przerwania może być indywidualnie włączane i wyłączane. Służą do tego bity **RXCIE**, **TXCIE** oraz **UDRIE** umieszczone w rejestrze **UCR**.

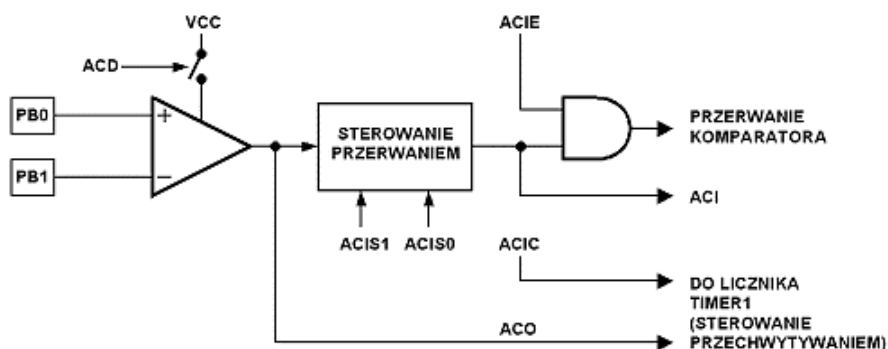
Uwaga! Włączenie przerwania opróżnienia bufora, gdy nie wykonano zapisu do bufora **UDR** powoduje natychmiastowe jego zgłoszenie.

Na zakończenie.

Układ UART jest konfigurowany automatycznie jeśli w programie wykorzystano instrukcje **PRINT** lub **INPUT**. Można oczywiście stworzyć własne procedury sterujące pracą układu UART, jeśli program nie korzysta z wymienionych instrukcji. W tym wypadku jest też możliwe dowolne skorzystanie z przerwań jakie generuje układ UART.

Komparator analogowy

Wejścia komparatora są wyprowadzone na zewnątrz za pomocą końcówek PB0 (AIN0, wejście nieodwracające) i PB1 (AIN1, wejście odwracające). Poniżej znajduje się uproszczony schemat układu komparatora.



Rysunek 7 Komparator analogowy.

Sterowanie pracą komparatora odbywa się za pomocą ustawienia odpowiednich bitów w rejestrze **ACSR** (*Analog Comparator Control and Status Register*) – adres **&H08 (&H28)**. Najważniejszy z nich to bit **ACD** (*Analog Comparator Disable*), którego ustawienie powoduje wyłączenie zasilania komparatora.

Informację o aktualnym stanie komparatora można uzyskać testując stan bitu **ACO** (*Analog*

Comparator Out). Bit ten jest sterowany bezpośrednio z wyjścia komparatora.

Komparator może być źródłem przerwań. Ustawienie bitu **ACIE** (*Analog Comparator Interrupt Enable*) powoduje włączenie przerwań z komparatora.

Wyzwolenie przerwania może nastąpić w wyniku stwierdzenia opadającego lub narastającego zbocza sygnału wyjściowego komparatora, albo oba te zbocza naraz – komparator zgłasza wtedy przerwania zarówno przy opadającym i narastającym zboczu. Wybór pomiędzy tymi trzema trybami dokonuje się ustawiając odpowiednio bity **ACIS0** i **ACIS1** (*Analog Comparator Interrupt Mode Select*) w rejestrze **ACSR**.

Po wystąpieniu jednego z warunków zgłoszenia przerwania, jest ustawiany znacznik przerwania **ACI**.

Uwaga! Przed zmianą stanu bitów **ACD**, **ACIS0** i **ACIS1** należy wyłączyć przerwania komparatora (bit **ACIE**), gdyż zmiana tych bitów może doprowadzić do niepotrzebnego zgłoszenia przerwania.

Ponieważ komparator analogowy może służyć do wyzwalania funkcji przechwytywania zawartości licznika **TIMER1**, w rejestrze **ACSR** znajduje się jeszcze bit **ACIC** (*Analog Comparator Input Capture Enable*), którego ustawienie powoduje skierowanie wyjścia komparatora także do układu licznika **TIMER1**.

Na zakończenie.

Sterowaniem układem komparatora analogowego uproszczono do niezbędnego minimum w języku BASCOM BASIC. Konfiguracją pracy komparatora zajmują się instrukcje: **CONFIG ACI**, a sterowaniem instrukcje **START** oraz **STOP**.

Dla użytkownika dostępne jest też przerwanie jakie generuje komparator. Stosując instrukcję **ON INTERRUPT** można reagować na zmiany stanu wyjściowego komparatora.

Używanie SPI

Informacja ogólne.

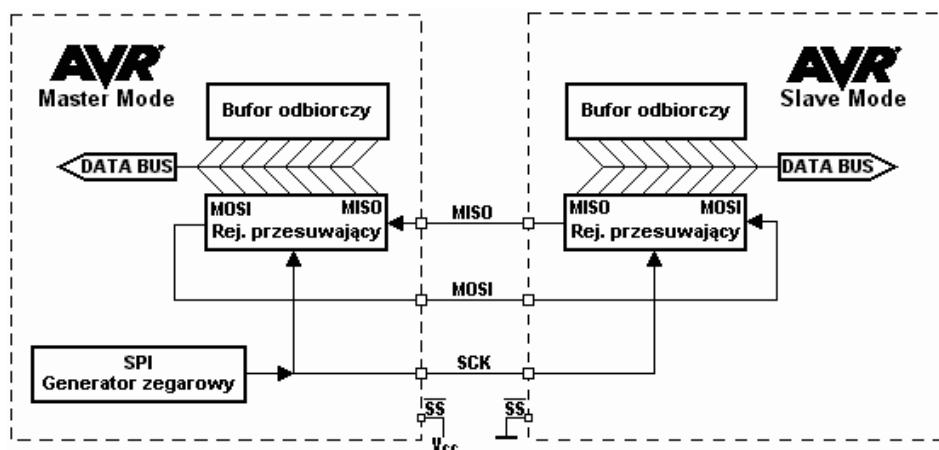
Interfejs SPI pozwala na bardzo szybkie synchroniczne przesyłanie danych pomiędzy procesorami AVR (i dwoma z rodziny 8051 *przyp. tłumacza*) a urządzeniami zewnętrznymi. W wielu procesorach SPI jest wykorzystane do celów programowania przez ISP (*In System Programming*).

Transmisja pomiędzy dwoma urządzeniami komunikującymi się przez SPI, odbywa się na zasadzie Master-Slave. Porównując to do urządzeń zewnętrznych, np. do różnorodnych czujników, mogących pracować tylko w trybie Slave - procesory połączone przez SPI mogą być skonfigurowane zarówno jako Master lub jako Slave.

Tryb pracy urządzenia jest ustalany przez odpowiednie ustawienie bitu **MSTR** w rejestrze kontrolnym SPI (**SPCR** – *SPI Control Register*). Specjalne znaczenie ma także stan końcówki /SS, które zostanie opisane dalej.

Procesor pracujący w trybie Master, jest aktywnym urządzeniem w systemie. Generuje on sygnał zegarowy, służący do synchronizacji transmisji (podobnie jak w I²C *przyp. tłumacza*). Urządzenie pracujące w trybie Slave nie może inicjować transmisji jak i generować sygnału zegarowego.

Urządzenie Slave nadaje i odbiera dane jeśli urządzenie Master nadaje sygnał zegarowy i robi to tylko podczas wysyłania danych. Znaczy to, że Master wysyła dane po to by odebrać inne dane od urządzenia Slave.



Rysunek 8 Połączenia pomiędzy układami.

Transmisja danych pomiędzy układami Master i Slave.

Połączenia pomiędzy dwoma urządzeniami komunikującymi się przez SPI pokazano na rysunku 8. Pokazano tam dwa identyczne urządzenia SPI. Lewy jest skonfigurowany jako Master, a prawy jako Slave. Końcówki MISO, MOSI oraz SCK są połączone z tymi samymi końcówkami w drugim układzie.

Tryb pracy każdego z elementów określa które końcówki będą wejściami, a które wyjściami. Ponieważ bity są przesuwane z urządzenia Master do Slave i z urządzenia Slave do Master równocześnie, w jednym cyklu zegarowym oba 8 bitowe rejestry mogą być traktowane jako jeden 16 bitowy rejestr przesuwający. Oznacza to iż po wygenerowaniu 8 impulsów zegarowych na końcówce SCK, połączone urządzenia wymieniają się danymi.

Podczas wysyłania danych stosowane jest pojedyncze buforowanie, a podczas odbierania danych podwójne. Wpływa to na sposób obsługi danych następująco:

1. Nowe bajty, które należy wysłać nie mogą być wpisane do rejestru danych (**SPDR** – *SPI Data Register*) / rejestru przesuwającego przed zakończeniem bieżącego cyklu wysyłania.
2. Odebrane bajty trafiają do bufora odbiorczego zaraz po zakończeniu transmisji bajtu.
3. Dana w buforze odbiornika musi być odczytana przed kolejną transmisją inaczej zostanie utracona.
4. Odczytywanie z rejestru **SPDR** powoduje de facto odczyt z bufora odbiorczego.

Po zakończeniu transmisji bajtu ustawiana jest flaga zgłoszenia przerwania *SPI Interrupt Flag (SPIF)* w rejestrze *SPI Status Register (SPSR)*. Spowoduje to, że procesor przejdzie do wykonywania obsługi tego przerwania, o ile zostaną one włączone. Ustawienie bitu *SPI Interrupt Enable (SPIE)* w rejestrze **SPCR** włącza zgłaszanie przerw z układu SPI, przy czym ustawienie bitu **I** w rejestrze **SREG** włącza globalny system przerw.

Końcówki układu SPI.

Układ SPI posiada cztery różne linie sygnałowe. Linie te to: sygnał zegara (SCK), *Master Out Slave In* (MOSI), *Master In Slave Out* (MISO) oraz wybór układu *Slave Select (/SS)*, aktywny w stanie niskim. Gdy układ SPI jest włączany, kierunek linii MOSI, MISO, SCK oraz /SS zostaje ustawiony zgodnie z poniższą tabelą.

Tabela 1. Ustawienia końcówek SPI

Kierunek linii	Kierunek dla układu Master	Kierunek dla układu Slave
MOSI	Decyduje użytkownik	Wejście
MISO	Wejście	Decyduje użytkownik
SCK	Decyduje użytkownik	Wejście
SS	Decyduje użytkownik	Wejście

Można zaobserwować, że linie pracujące jako wejścia są ustawiane w ten tryb automatycznie. Linie wyjściowe muszą być ustawiane „ręcznie” w programie. Powodem takiego zachowania jest zabezpieczenie przed uszkodzeniami, wynikłymi na przykład z budowy sterowników.

System z wieloma układami Slave – działanie końcówki SS.

Końcówka *Slave Select* (/SS) odgrywa główną rolę w konfiguracji układu SPI. W zależności od trybu w jakim pracuje układ i konfiguracji tej linii interfejsu, końcówka ta może być używana do włączania lub wyłączania SPI. Działanie końcówki /SS można porównać do końcówki /CS (*Chip Select*) występującej w pamięciach statycznych, która wybiera układ. W trybie Master stan końcówki /SS musi być utrzymywany na poziomie wysokim by układ poprawnie pracował w tym trybie, gdyż końcówka ta pełni wtedy rolę wejścia. Podanie niskiego poziomu, przełącza układ SPI w tryb Slave, a sterownik SPI wykonuje następujące operacje:

1. Bit **MSTR** w rejestrze *SPI Control Register (SPCR)* jest zerowany i układ SPI przełącza się w tryb Slave. Kierunki działania linii zostają zmienione jak to pokazano w Tabeli 1.
2. Ustawiana jest flaga *SPI Interrupt Flag (SPIF)* w rejestrze *SPI Status Register (SPSR)*. Jeśli w tym czasie przerwania od SPI jak i cały system przerwań jest włączony, procesor przechodzi do wykonywania odpowiedniej procedury jego obsługi. Może to być użyteczne w systemach z wieloma urządzeniami pracującymi w trybie Master, by zabezpieczyć się przed skutkami możliwych konfliktów gdy kilka urządzeń próbuje używać SPI w tym samym czasie. Jeśli końcówka /SS jest ustawiona jako wyjście, może być używana do innych celów, nie powodując zakłóceń w pracy SPI.

Uwaga! W przypadkach gdzie kontroler AVR jest skonfigurowany do pracy jako urządzenie Master musi być pewne, że stan końcówki /SS będzie w stanie H pomiędzy dwoma kolejnymi transmisjami, gdyż stan bitu **MSTR** jest sprawdzany przed wpisaniem nowego bajtu. Kiedy stan bitu **MSTR** został wyzerowany – przez pojawienie się stanu L na końcówce /SS – bit ten musi być ustawiony przez program, aby z powrotem przełączyć SPI w tryb Master.

W trybie Slave końcówka /SS pracuje zawsze jako wejście. Gdy stan końcówki /SS jest utrzymywany w stanie niskim, SPI jest aktywne i końcówka MISO staje się wyjściem jeśli tylko zostało to skonfigurowane przez użytkownika. Wszystkie inne końcówki są wejściami. Gdy na końcówce /SS jest utrzymywany stan wysoki, wszystkie końcówki są wejściami. Lecz SPI nie jest wtedy aktywne, co oznacza, że nie odbiera przychodzących danych.

Tabela 2 ukazuje znaczenie końcówki /SS w poszczególnych trybach.

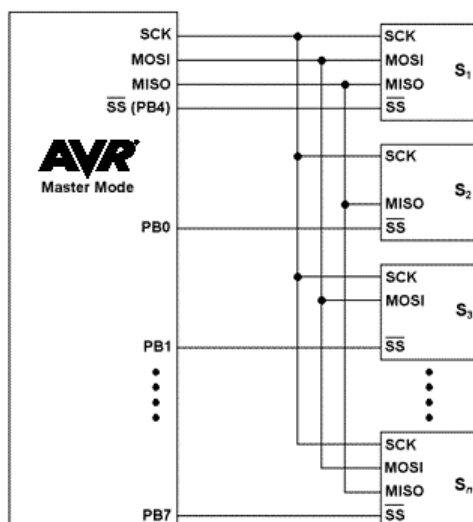
Uwaga! W trybie Slave, logika układu SPI zostaje wyzerowana zaraz po tym jak na końcówce /SS wykryto stan wysoki. Jeśli wysoki stan na końcówce /SS zostanie wykryty podczas transmisji, SPI natychmiast przerywa wysyłanie oraz odbieranie danych, co w konsekwencji prowadzi, że bieżący bajt zostaje utracony.

Tabela 2. Znaczenie końcówki /SS.

Tryb	Konfiguracja /SS	Stan końcówki /SS	Opis
Slave	Zawsze wejście	H	Slave nieaktywny
		L	Slave aktywny
Master	Wejście	H	Master aktywny
		L	Master nieaktywny
	Wyjście	H	Master aktywny
		L	

Jak to pokazano w powyższej tabelce, końcówka /SS w trybie Slave pełni zawsze rolę wejścia. Stan niski na tej końcówce aktywuje układ SPI, a stan wysoki powoduje jego dezaktywowanie.

System mikroprocesorowy z jednym układem Master i kilkoma układami Slave jest przedstawiony na rysunku 9. Układ Master jest skonfigurowany tak, że końcówka /SS pełni rolę wyjścia. Liczba układów Slave w tym systemie jest tylko ograniczona liczbą linii wejścia-wyjścia mikrokontrolera, służących do wyboru układu.



Rysunek 9 Praca z wieloma układami Slave.

Możliwość podłączenia wielu urządzeń do tej samej magistrali SPI opiera się na prostym fakcie: Tylko jedno urządzenie Master i jedno urządzenie Slave może być aktywne w tym samym czasie. Linie MISO, MOSI oraz SCK wszystkich innych urządzeń Slave są w stanie wysokiej impedancji (są trójstanowe i nie posiadają rezystorów podciągających). Błędy w projekcie takiego systemu (np.: gdy dwa układy Slave są aktywne w tym samym czasie) mogą spowodować kolizję na liniach, co może doprowadzić do powstania na końcówce stanu zabronionego (pomiędzy „pewną” jedynką, a „pewnym” zerem) dla układów CMOS i trzeba temu zapobiec. Włączenie w szereg z końcówkami rezystorów o wartościach 1-10 k jest pewnym rozwiązaniem. Jednakże rezystancje te połączone z pojemnościami wejściowymi poszczególnych linii, mają wpływ na szybkość transmisji przez SPI.

Można zauważyć, że jednokierunkowe urządzenia SPI wymagają tylko linii zegarowej i jednej linii danych. Czy urządzenie takie używa końcówki MISO lub MOSI zależy tylko od zadań jakie spełnia. Dla przykładu, proste czujniki tylko wysyłają dane (zobacz S2 na rysunku 9), podczas gdy zewnętrzny układ DAC tylko odbiera dane (zobacz S3 na rysunku 9).

Wykresy czasowe.

Układ SPI procesora może pracować w czterech trybach: od 0 do 3. Tryby te generalnie określają w jaki sposób dane są taktowane podczas transmisji z i do układu SPI. Wyboru trybu dokonujemy przez odpowiednie ustawienie dwóch bitów w rejestrze *SPI Control Register (SPCR)*.

Polaryzacja sygnału zegarowego jest kontrolowana przez stan bitu **CPOL**, służącego do wyboru: czy stan aktywny linii zegara jest stan wysoki lub niski. Bit kontrolujący fazę sygnału zegarowego (**CPHA**) wybiera pomiędzy dwoma podstawowymi, różniącymi się formatami transmisji. By zapewnić poprawną komunikację pomiędzy układem Master a Slave, oba muszą pracować w tym samym trybie. W niektórych przypadkach musi być wykonana rekonfiguracja urządzenia Master, by był on zgodny z specyficznymi ustawieniami układów Slave.

Stan bitów **CPOL** i **CPHA** określa tryb w jakim pracuje SPI. Tryby te są wyszczególnione w tabeli 3 (poniżej). Ponieważ nie jest to standardem i zostało to określone inaczej w innych źródłach, konfiguracja układu SPI musi być wykonana ze szczególną ostrożnością.

Tabela 3. Tryby pracy SPI

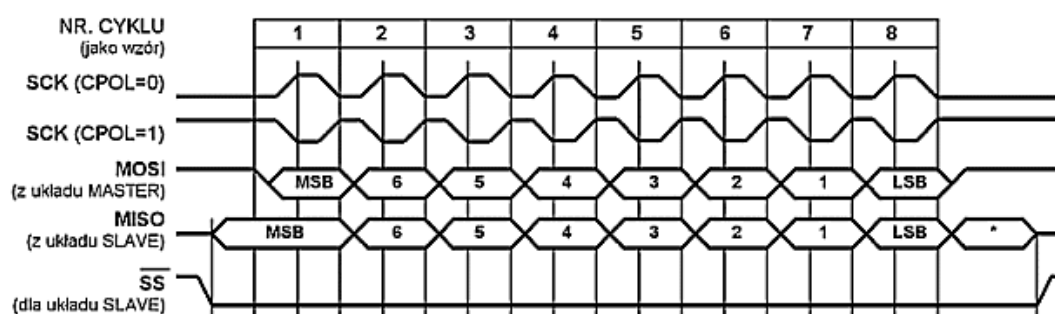
Tryb pracy SPI	CPOL	CPHA	Przesuwanie przy zboczu SCK	Przechwytywanie przy zboczu SCK
0	0	0	Opadającym	Narastającym
1	0	1	Narastającym	Opadającym
2	1	0	Narastającym	Opadającym
3	1	1	Opadającym	Narastającym

Zmiana polaryzacji sygnału zegarowego nie jest aż tak znacząca dla formatu transmisji. Zmiana tego bitu powoduje, że sygnał zegarowy zostanie zanegowany (stan aktywny niski stanie się stanem aktywnym wysokim i a stan wysoki nieaktywny stanem niskim nieaktywnym). Jednakże zmiana stanu bitu odpowiedzialnego za fazę tego sygnału, wybiera jeden z dwóch różniących się od siebie sposobów taktowania transmisji, które to zostaną opisane dokładnie w dwóch następnych rozdziałach.

Ponieważ linie MOSI i MISO układów Master i Slave są połączone bezpośrednio, rysunki przedstawiają przebiegi na każdej z nich. Linia /SS pełni rolę wejścia wybierającego dla układu pracującego jako Slave. Stan linii /SS dla układu Master nie jest tutaj pokazany. Linia ta jest nieaktywna, gdyż jest w stanie wysokim (jeśli została skonfigurowana jako wejście) lub jest ona skonfigurowana jako linia wyjściowa.

CPHA = 0 i CPOL = 0 (Tryb 0) oraz **CPHA = 0 i CPOL = 1 (Tryb 1)**

Wykresy czasowe układu SPI, w którym **CPHA** jest w stanie niskim pokazano na rysunku 3. Dwa przebiegi linii SCK odnoszą się do uzyskanych przebiegów gdy **CPOL = 0** i gdy **CPOL = 1**.



Rysunek 10 Przebiegi gdy **CPHA = 0**.

Gdy SPI jest skonfigurowane jako Slave, transmisja rozpoczyna się gdy zostanie stwierdzone opadające zbocze na linii /SS. To zaś uaktywnia SPI układu Slave oraz powoduje, że stan bitu MSB bajtu umieszczanego w rejestrze **SPDR** zostaje wystawiony na linii MISO.

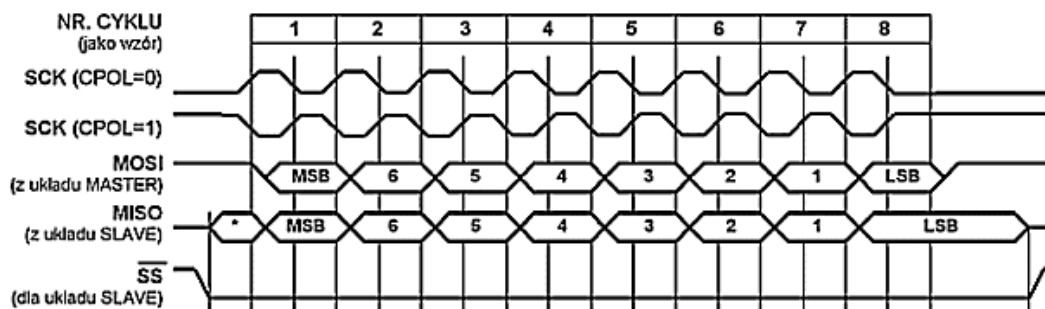
Transmisja bieżącego bajtu rozpoczyna się, gdy zostaje on wpisany - przez program - do rejestru **SPDR**. Powoduje to także, że układ rozpoczyna generowanie sygnału zegarowego. W sytuacjach gdzie **CPHA = 0**, stan linii SCK pozostaje zerem przez pierwszą połowę czasu pierwszego cyklu zegarowego. To pozwala, by dane były stabilne zarówno na linii wejściowej układu Master jak i Slave.

Dane z linii wejściowych są odczytywane przy zmianie stanu linii SCK z nieaktywnego na aktywny (zbocze narastające gdy **CPOL = 0**, zbocze opadające gdy **CPOL = 1**). Zmiana stanu linii SCK ze stanu aktywnego na nieaktywny (zbocze opadające gdy **CPOL = 0**, zbocze narastające gdy **CPOL = 1**) powoduje, że dane są przesuwane o jeden bit, co pozwala na wystawienie stanu następnego bitu na liniach MOSI i MISO.

Po ośmiu taktach zegara transmisja jednego bajtu jest zakończona. W obu układach: Master i Slave, ustawiany jest bit *SPI Interrupt Flag* (**SPIF**) oraz odebrany właśnie bajt trafia do bufora odbiorczego.

CPHA = 1 i CPOL = 0 (Tryb 2) oraz **CPHA = 1 i CPOL = 1 (Tryb 3)**

Wykresy czasowe układu SPI, w którym **CPHA** jest w stanie wysokim pokazano na rysunku 4. Dwa przebiegi linii SCK odnoszą się do uzyskanych przebiegów gdy **CPOL = 0** i gdy **CPOL = 1**.



Rysunek 11 Przebiegi gdy **CPHA** = 1.

Tak jak w poprzednim przypadku, opadające zbocze na linii /SS wybiera i aktywuje układ Slave. Porównując jednak oba te przypadki, w momencie gdy **CPHA** jest zerem, transmisja nie rozpoczyna się i bit MSB nie jest wystawiany przez układ Slave w tej fazie. Transmisja rozpoczyna się gdy w programie następuje zapis do rejestru **SPDR** układu Master, co powoduje iż generowany jest sygnał zegarowy. Pierwsze zbocze – czyli zmiana ze stanu nieaktywnego na aktywny – na linii SCK (zbocze narastające gdy **CPOL** = 0 lub opadające gdy **CPOL** = 1) powoduje, że zarówno układ Master jak i układ Slave wystawiają bit MSB bajtu umieszczonego w rejestrze **SPDR**.

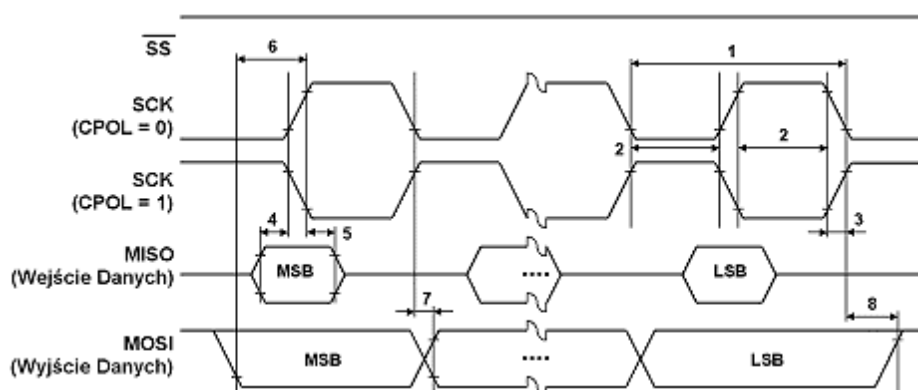
Jak to pokazano na rysunku 11, nie ma tutaj opóźnienia o połowę cyklu zegarowego jak to ma miejsce w trybach 0 i 1. Stan linii SCK zmienia się natychmiast na początku pierwszego cyklu zegarowego. Dane z linii wejściowych są odczytywane przy zmianie stanu linii SCK z aktywnego na nieaktywny (zbocze opadające gdy **CPOL** = 0 lub narastające gdy **CPOL** = 1).

Po ośmiu taktach zegara transmisja jednego bajtu jest zakończona. W obu układach: Master i Slave, ustawiany jest bit **SPI Interrupt Flag (SPIF)** oraz odebrany właśnie bajt trafia do bufora odbiorczego.

Uwagi dotyczące szybkich transmisji.

Układy które pracują przy wyższych częstotliwościach zegara i moduły SPI dostosowane do pracy przy szybkościach dochodzących połowy tej częstotliwości, wymagają bardziej szczegółowego taktowania by dochodziło do zgodności pomiędzy układem wysyłającym i odbierającym dane.

Poniższe dwa przebiegi czasowe ukazują sposób taktowania układu w trybie Master oraz Slave pracujących w trybach 0 i 1. Dokładne zależności czasowe na pokazanych tu przebiegach zmieniają się w różnych układach i nie znajdują się w tej nocie aplikacyjnej. Jednakże funkcjonowanie wszystkich układów jest w zasadzie takie same, więc poniższe uwagi mają zastosowanie dla większości układów.



Rysunek 12 Przebiegi w trybie Master.

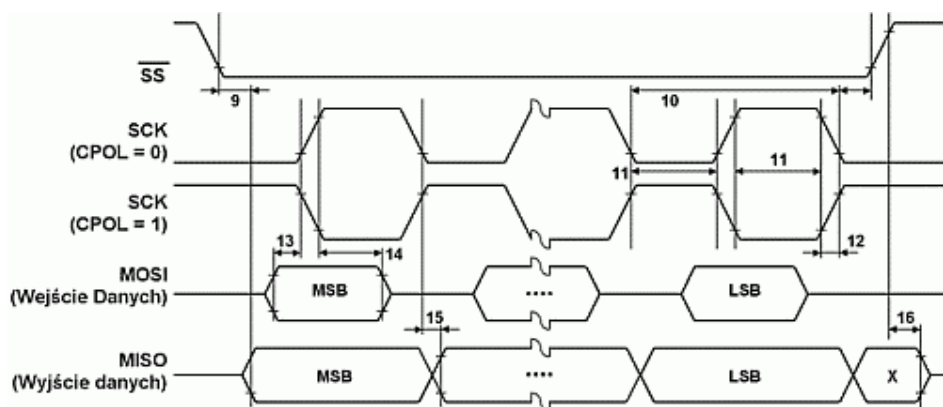
Minimalne zależności czasowe sygnału zegarowego są określone odcinkami „1” i „2”. Odcinek „1” określa okres sygnału SCK, a zaś odcinek „2” określa czas trwania stanu wysokiego / niskiego sygnału zegarowego. Maksymalny czas narastania i opadania sygnału SCK jest pokazany tutaj jako odcinek „3”. Odcinki te powinny być traktowane jako podstawowe, przy sprawdzaniu czy stawiane wymagania odpowiadają układowi Slave.

Czas ustalania (*Setup Time*) „4” oraz czas podtrzymania (*Hold Time*) „5” są szczególnie ważne, gdyż to one określają wymagania układu AVR połączonego z układem Slave. Te zależności czasowe decydują, jak długo przed pojawieniem się zbocza sygnału zegarowego układ Slave jest w stanie wystawić stabilny stan linii danych oraz jak długo po jego przejściu dane te pozostają stabilne.

Jeśli powyższe dwa odcinki czasu są wystarczająco długie, można powiedzieć, że układ Slave sprosta wymaganiom procesora AVR. Lecz czy AVR sprosta wymaganiom układu Slave?

Odcinek „6” (*Out to SCK*) określa minimalny czas, w którym AVR wystawia ważne dane na linii danych przed pojawieniem się pierwszego zbocza sygnału SCK. Czas ten może być porównywany z czasem ustalania „4” układu Slave.

Odcinek „7” (*SCK to Out*) określa maksymalny czas, w którym AVR musi wystawić kolejny bit na linii danych. Odcinek „8” zaś (*SCK to Out high*) to minimalny czas w którym AVR musi utrzymać stan ostatniego transmitowanego bitu na linii MOSI po tym, jak linia SCK wraca do stanu nieaktywnego.



Rysunek 13 Przebiegi w trybie Slave.

W zasadzie przebiegi są takie same przy pracy układu jako Slave, jak te opisane wcześniej. Ponieważ układy wtedy „zamieniają się rolami”, wymagania zostają jakby odwrócone. Minimalny czas przy pracy w trybie Master staje się maksymalnym czasem przy pracy w trybie Slave i vice versa.

Konflikty w transmisji.

Konflikty w transmisji występują gdy rejestr **SPDR** jest zapisywany nową wartością w trakcie trwania transmisji. Rejestr ten podczas nadawania jest pojedynczo buforowany i zapis do rejestru **SPDR** w rezultacie powoduje bezpośredni zapis do rejestru przesuwającego SPI. Ponieważ operacja zapisu może uszkodzić transmitowany bajt, ustawiany jest bit *Write Collision* (**WCOL**) w rejestrze **SPSR**. Operacja zapisu w tym wypadku nie zostanie wykonana i transmisja będzie przebiegać dalej bez zakłóceń.

Kolizje przy zapisie generalnie dotyczą układów pracujących jako Slave, gdyż nie mają one kontroli nad tym, czy układ Master zainicjuje transmisję czy nie. Układ Master jest w stanie określić kiedy trwa transmisja. Tak więc układ pracujący w trybie Master nie powinien zatem generować kolizji przy zapisie – chociaż logika układu SPI może wykryć ten błąd, również podczas pracy w trybie Slave.

Na zakończenie.

Jeśli ustawisz opcję SPI w menu [Options | Compiler | SPI](#) do rejestru **SPCR** zostanie wpisane 01010100, co oznacza: włącz SPI, tryb Master, **CPOL** = 1. Jeśli zależy ci na kontroli nad wszystkimi opcjami sprzętowego układu SPI, możesz użyć instrukcji [CONFIG SPI](#).

Transmisja za pomocą SPI także jest prosta, gdyż w języku BASCOM BASIC przewidziano odpowiednie instrukcje: [SPIINIT](#), [SPIMOVE](#), [SPIIN](#) oraz [SPIOUT](#).

Inicjalizacja

Po włączeniu zasilania wszystkie końcówki portów pracują jako trójstanowe, i mogą być używane jako wejścia.

Jeśli port ma pracować jako wyjście, należy ustawić odpowiedni kierunek instrukcją **CONFIG PORT**:

```
Config Portb = Output
```

Można także ustawiać kierunek odpowiednich linii portu za pomocą ustawienia rejestru DDRB. Dla przykładu instrukcja:

```
DDRB = &B00001111
```

spowoduje wpisanie do liczby 15 do rejestru kierunku portu B, i 4 starsze bity (PORTB.7 to PORTB.4) będą wyjściami, a 4 młodsze (PORTB.3 to PORTB.0) wejściami.

Można także określić kierunki działania dla każdej końcówki portu osobno. W tym celu należy wykorzystać instrukcję **CONFIG PIN**. Na przykład:

```
Config PinB.0 = Input
```

Inicjalizacja programu języka BASCOM.

Kompilator języka BASCOM umieszcza specjalną procedurę inicjalizacyjną, która jest odpowiedzialna za zerowanie wszystkich komórek pamięci RAM. Można to wyłączyć umieszczając na początku programu dyrektywę **\$NORAMCLEAR**.

Procedura ta, wykonuje także inicjalizację wyświetlacza, układu UART jeśli w programie występują instrukcje **INPUT**, **PRINT**, **LCD** etc.

Rejestry specjalne

Poniżej znajduje się tabela adresów wszystkich rejestrów specjalnych procesorów AVR. Wszystkie nazwy rejestrów i bitów są rozpoznawane przez język BASCOM BASIC.

Dane tutaj zamieszczone odpowiadają procesorowi AT90s8515.

Adres	Nazwa	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
\$3F	SREG	I	T	H	S	V	N	Z	C
\$3E	SPH	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8
\$3D	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0
\$3C		-							
\$3B	GIMSK	INT1	INT0	-	-	-	-	-	-
\$3A	GIFR	INTF1	INTF0	-	-	-	-	-	-
\$39	TIMSK	TOIE1	OCIE1A	OCIE1B	-	TICIE1	-	TOIE0	-
\$38	TIFR	TOV1	OCF1A	OCF1B	-	CF1	-	TOV0	-
\$37		-							
\$36		-							
\$35	MCUCR	SRE	SRW	SE SM	ISC11	ISC10	ISC01	ISC00	-
\$34		-							
\$33	TCCR0	-	-	-	-	-	CS02	CS01	CS00
\$32	TCNT0	Timer1/Counter0 - Rejestr licznika							
\$31		-							
\$30		-							
\$2F	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	-	-	PWM11	PWM10
\$2E	TCCR1B	ICNC1	ICES1	-	-	CTC1	CS12	CS11	CS10
\$2D	TCNT1H	Timer1/Counter1 - Rejestr licznika - MSB							
\$2C	TCNT1L	Timer1/Counter1 - Rejestr licznika - LSB							
\$2B	OCR1AH	Timer1/Counter1 - Rejestr porównania A - MSB							
\$2A	OCR1AL	Timer1/Counter1 - Rejestr porównania A - LSB							
\$29	OCR1BH	Timer/Counter1 - Rejestr porównania B - MSB							
\$28	OCR1BL	Timer/Counter1 - Rejestr porównania B - LSB							
\$27		-							
\$26		-							
\$25	ICR1H	Timer/Counter1 - Rejestr przechwytyjący - MSB							
\$24	ICR1L	Timer/Counter1 - Rejestr przechwytyjący - LSB							
\$23		-							
\$22		-							
\$21	WDTCR	-	-	-	WDTOE	WDE	WDP2	WDP1	WDP0
\$20		-							
\$1F		-	-	-	-	-	-	-	EEAR8
\$1E	EEARL	EEPROM - Rejestr adresowy LSB							
\$1D	EEDR	EEPROM - Rejestr danych							
\$1C	EECR	-	-	-	-	-	EEMWE	EWE	EERE
\$1B	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0
\$1A	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0
\$19	PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0
\$18	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
\$17	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
\$16	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
\$15	PORTC	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	PORTC7
\$14	DDRC	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0
\$13	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0
\$12	PORTD	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	PORTD7
\$11	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
\$10	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
\$0F	SPDR	SPI - Rejestr danych							
\$0E	SPSR	SPIF	WCOL	-	-	-	-	-	-
\$0D	SPCR	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0
\$0C	UDR	UART - Bufor transmisji							
\$0B	USR	RXC	TXC	UDRE	FE	OR	-	-	-
\$0A	UCR	RXCIE	TXCIE	UDRIE	RXEN	TXEN	CHR9	RXB8	TXB8
\$09	UBRR	UART - Rejestr szybkości transmisji							

Adres	Nazwa	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
\$08	ACSR	ACD	–	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0
\$00		–							

Nazwy rejestrów i bitów, oraz ich adresy w przestrzeni adresowej SFR, są zdefiniowane w poszczególnych plikach DAT. Pliki te znajdują się w katalogu głównym programu BASCOM AVR.

Rejestry mogą być używane jako normalne zmienne typu Byte. Na przykład:

```
Portb = 40
```

spowoduje umieszczenie liczby 40 w rejestrze portu B.

Uwaga! Nazwy rejestrów i bitów są zastrzeżone, zatem nie mogą stać się nazwą jakiegokolwiek definiowanej przez użytkownika zmiennej! Na przykład:

```
Dim SREG As Byte
```


Alfanumeryczny wyświetlacz LCD

Wyświetlacz alfanumeryczny LCD może być dołączony na dwa sposoby:

- Łącząc bezpośrednio końcówki wyświetlacza LCD z końcówkami portów mikrokontrolera. Tryb ten nazywany jest *Pin Mode*. W trybie tym pozostawiono pełną swobodę przy łączeniu końcówek LCD do portów procesora. Można w ten sposób uprościć połączenia na płycie, lecz okupione jest to zwiększeniem rozmiaru potrzebnego kodu.
- Łącząc końcówki wyświetlacza (dane) do systemowej szyny danych, która jest dostępna w systemach posiadających zewnętrzną pamięć danych XRAM. Jest to tzw. tryb *Bus Mode*.

Standardowo wyświetlacz LCD, jest podłączony wg danych z poniższej tabeli:

Nazwa końcówki LCD	Port mikrokontrolera	Numer końcówki LCD
DB7	PORTB.7	14
DB6	PORTB.6	13
DB5	PORTB.5	12
DB4	PORTB.4	11
E	PORTB.3	6
RS	PORTB.2	4
RW	masa	5
Vss	masa	1
Vdd	+5 Volt	2
Vo	0-5 Volt	3

Ten sposób połączeń pozostawia końcówki PORTB.1, PORTB.0 oraz cały PORTD do innych celów. Oczywiście, jak wspomniano na wstępie, sposób połączeń jest dowolny. Można go określić w menu [Options | Compiler](#), lub w programie instrukcją [CONFIG LCDPIN](#).

Gdy wyświetlacz jest dołączony przez systemową szynę danych – drugi sposób – wszystkie końcówki danych wyświetlacza (DB7-DB0) muszą być dołączone do szyny danych. Transmisja odbywa się wtedy w trybie 8 bitowym.

Sterowaniem liniami **E** i **RS** zajmuje się wtedy dekodер adresów, tak aby nie było konfliktów z zewnętrzną pamięcią. Adresy, których wystawienie na szynie adresowej, powoduje ustawienie lub wyzerowanie linii **E** oraz **RS**, należy ustalić za pomocą dyrektyw [\\$LCDRS](#) i [\\$LCD](#).

Język BASCOM BASIC oferuje wiele instrukcji obsługujących wyświetlacz LCD. Aby jednak mieć całkowitą kontrolę nad wyświetlaczem można stosować procedury w języku assembler. Poniżej znajduje się przykład:

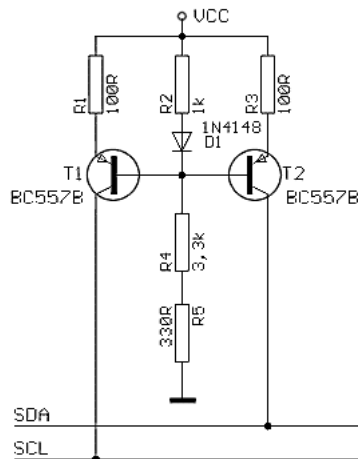
\$ASM

```
Ldi _temp1, 5      'załadowanie kodu rozkazu do R24
Rcall _Lcd_control  'prześlemy do LCD jako komendę

Ldi _temp1, 65      'załadowanie do R24 kodu znaku (litera A)
Rcall _Write_lcd     'prześlemy do LCD jako daną
$END ASM
```

Uwaga! Nazwy `_Lcd_control` oraz `_Write_lcd` są adresami procedur w assemblerze, możliwych do wywołania z języka BASCOM BASIC. Aby były one dostępne należy w programie skonfigurować wyświetlacz lub użyć jakiejkolwiek instrukcji z nim związanej.

Dalszych informacji na temat programowania wyświetlacza LCD na niskim poziomie należy szukać w dokumentacji jego producenta.



Rysunek 15 Układ aktywnego obciążenia.

Używanie magistrali 1Wire™

Magistrala 1wire została zaprojektowana przez firmę DALLAS Semiconductor. Używa tylko jednego przewodu do przesyłania danych. Oczywiście nie należy pomijać linii masy, która występuje jak zwykle.

Poniższy opis został napisany przez Göte Haluza. Przetestował on nowe procedury przeszukiwania magistrali podczas budowy stacji meteorologicznej. Dzięki!

Magistrala 1Wire zaprojektowana przez Dallas Semiconductor.

Znajdują się tu podstawowe informacje na temat magistrali 1Wire obsługiwanej m.in. przez BASCOM. Aby uzyskać więcej bardziej szczegółowych informacji należy przejść na firmową stronę WWW firmy Dallas: <http://www.dalsemi.com>.

Używanie języka BASCOM, stwarza zupełnie nowe możliwości i powoduje, że świat staje się łatwiejszy. Informacje tu zawarte mają przybliżyć zagadnienie magistrali 1Wire z punktu widzenia użytkownika.

Magistrala 1Wire korzysta z transmisji szeregowej, używanej przez wiele elementów z oferty firmy Dallas. Magistrala może być zaimplementowana na dwa sposoby:

- Stosując dwa przewody – wtedy używane są tylko linia DQ oraz masa. Zasilanie elementu jest dostarczane przez linię DQ. Gdy magistrala jest wolna na linii panuje +5V, co powoduje ładowanie wewnętrznego kondensatora elementu. Zgromadzone napięcie jest wtedy wykorzystywane przez element podczas komunikacji. Takie rozwiązanie nazywane jest właśnie 1Wire.
- Stosując 3 przewody – wtedy także linia Vdd jest używana, po której podawane jest zasilanie +5V. Reszta linii jest połączona jak przy komunikacji 1 przewodowej. To rozwiązanie zwane jest 2Wire.

Linia masy nie jest zaliczana jako element magistrali przez Dallas-a. Dlatego będziemy tutaj używać nomenklatury firmy Dallas.

Jak to działa (1wire).

Wówczas gdy magistrala jest wolna, na linii DQ utrzymywany jest stan wysoki. Poprzez linię DQ urządzenie zatem pobiera napięcie zasilania, i wykonuje swoje wewnętrzne operacje.

Kiedy kontroler-nadzorca (*host*, np. mikrokontroler) rozpoczyna transmisję po magistrali,

wysyła wtedy rozkaz *Reset*. Nie jest to skomplikowane. Po prostu ustawia linię DQ w stan niski na czas 480µs (wg definicji magistrali firmy Dallas). Spowoduje to przejście elementu w tryb reset, który odpowiada wysyłając impuls sygnału gotowości i nasłuchuje magistrali.

Sygnał gotowości to nic innego jak przejście linii DQ w stan niski na krótki czas. Czas ten jest określony przez element.

Teraz, kontroler wie, że przynajmniej jedno urządzenie jest podpięte do magistrali, choć nie wie jakie dokładnie.

Ważne! Każdą transmisję po magistrali 1Wire inicjuje kontroler, który nakłada w odpowiednich przedziałach czasowych stan aktywny (niski) na linii DQ, która normalnie jest w stanie wysokim. Wewnętrzny kondensator każdego z elementów dostarcza mu napięcia zasilania podczas wystawienia stanu niskiego.

Jak komunikować się przez magistralę 1Wire.

Teraz można już rozpocząć odczyt lub zapis danych do elementów. Jeśli do magistrali dołączony jest tylko jeden element, lub chcemy zaadresować wszystkie naraz, można wysłać rozkaz *Skip Rom*. Oznacza to: nie potrzebne są numery identyfikacyjne elementów - opuść ten fragment komunikacji.

Po wysłaniu komendy *Reset*, wszystkie elementy prowadzą nasłuch magistrali. Jeśli zaadresowane będzie tylko jedno z nich, reszta wyłącza nasłuch, do czasu ponownego wysłania komendy *Reset*.

Nie będę opisywał tutaj rozkazów języka BASCOM BASIC – tłumaczą się one same. Lecz zasada jest taka: kontroler najpierw wysyła rozkaz na magistralę i potem odczytuje odpowiedź. Dane w postaci rozkazów jakie należy przesłać do elementu, zależą od rodzaju tegoż elementu – oraz tego co chcesz aby element „zrobił”. Każdy z elementów firmy Dallas posiada notę katalogową, którą znajdziesz na stronie <http://www.dalsemi.com/datasheets/pdfindex.html>. Możesz tam znaleźć wszystko na temat rozkazów konkretnego elementu.

Jest parę spraw, wartych przemyślenia, przy decydowaniu o sposobie realizacji magistrali.

Instrukcje zaimplementowane w języku BASCOM BASIC są zawsze takie same. Nie stanowi to zatem problemu.

Napięcie zasilania Vdd (+5V), przy realizacji 2Wire, powinno być brane z osobnego zasilacza według Dallas-a. Lecz wszystko będzie działać dalej, jeśli zasilanie będzie pobierane z tego samego źródła co procesor, na przykład bezpośrednio ze stabilizatora. Nie udało mi się doprowadzić do działania magistrali, pobierając zasilanie bezpośrednio z końcówki zasilania.

Niektóre elementy pobierają dość dużo prądu podczas specjalnych operacji. Na przykład układ DS1820 pobiera znacznie więcej energii podczas operacji pomiaru temperatury (*Convert Temperature*). Ponieważ czujnik wie kiedy jest zasilany (jest możliwe odczytanie i sprawdzenie czy tak jest rzeczywiście) - niektóre operacje, jak pomiar temperatury zajmuje różną ilość czasu. Na przykład rozkaz *Convert T*, potrzebuje ~200ms przy magistrali 2Wire, lecz przy magistrali 1Wire – zajmuje to ~700ms. Powinno to być brane pod uwagę podczas programowania.

Ponadto zasilanie też musi być doprowadzone w jakiś sposób.

Jeśli ma być używana magistrala 2Wire, możesz opuścić tą część tekstu. Możesz wtedy raz za razem wysyłać rozkaz *Convert T* do wszystkich urządzeń podłączonych do magistrali. I w dodatku skrócić czas. Rozkaz ten jest najbardziej „prądożerny”, możliwy do wykonania przez różne czujniki, jestem tego pewny.

Jeśli ma być używana magistrala 1Wire, jest kilka spraw do przemyślenia. Na przykład o tym jak nie pobierać zbyt dużo prądu. Ale jak zasilać? To określa typ urządzeń (ich pobór mocy) oraz rodzaj operacji jakie mają wykonać (pobór mocy przy określonych zadaniach).

Krótki i nieścisły opis sposobu zasilania oraz refleksje nad długością przewodów.

Stosując tylko końcówkę portu jako źródło zasilania, działały maksymalnie 4 układy (procesor AVR, używając wewnętrznego podciągania. W 8051 nie testowałem). Nie myśl sobie, że można zatem przysyłać rozkazy jeden po drugim do różnych czujników.

Gdy zasilanie doprowadzone było linią DQ, połączoną z +5V przez rezystor 4k7; sprawdziłem, iż może być dołączonych około 70 czujników. Lecz ostrożnie, wysyłanie non-stop rozkazu *Convert T*, może w rezultacie dać fałszywe odczyty. Około ~15 czujników to maksimum, gdy bez przerwy wykonywać będą jakieś operacje. W nomenklaturze firmy Dallas zwane jest to "*pull-up resistor*".

Oznacza to, że 70 urządzeń podłączonych do magistrali może być z powodzeniem zasilana w ten sposób.

Wspomniany rezystor 4k7, może mieć mniejszą wartość. Dallas twierdzi, że minimum to 1k5. Ja przetestowałem nawet 500Ω – poniżej tej wartości, magistrala nie była zdatna do użytku (dla AVR). Zmniejszając rezystancję zwiększamy możliwości zasilania – oraz zwiększamy odporność linii na zakłócenia. Naturalnie minimalna rezystancja jest zależna od właściwości portu kontrolera. Najlepiej zostawić 4k7 – jest to wartość rekomendowana jako standardowa.

Firma Dallas podaje jeszcze jedną możliwość zasilania, zwaną „*strong pull-up*”. Polega ona (w skrócie) na zastąpieniu rezystora tranzystorem MOSFET, który zasila linię DQ w wystarczającą mocą, by dalej magistrala pracowała w konfiguracji 1Wire. Nawet podczas zadań pobierających dużą ilość prądu. Nie testowałem tego rozwiązania, ale powinno ono działać. Ponieważ funkcjonalność tego rozwiązania jest naprawdę ograniczona; BASCOM nie posiada wsparcia dla takiego rozwiązania. Jednak, mówię o tym, byś wiedział. „*Strong pull-up*” wymaga użycia jeszcze jednej końcówki procesora do sterowania bramką tranzystora MOSFET.

Długość połączeń

Dla krótkich połączeń - do 30 metrów, dobór przewodu nie jest krytyczny. Dwużyłowy płaski przewód telefoniczny, działa z powodzeniem dla ograniczonej liczby elementów na magistrali. Jednakże, im dłuższe połączenia, wtedy dadzą o sobie znać właściwości przewodów i należy zastanowić się nad ich wyborem.

Dla długich połączeń, firma Dallas poleca dwuprzewodową-skrętkę (np. pochodzącą z przewodu UTP kategorii 5).

Według standardu połączenie może mieć do 100 metrów, więc chyba to nie problem. Choć widziałem przykłady z kablami o długości 300 metrów. A nawet, o ile sobie przypominam, widziałem gdzieś 600 metrów (nie mogę jednak tego potwierdzić).

Szumy i korekcja CRC.

Dłuższe połączenia jak i zakłócenia zewnętrzne, spowodują, że pojawi się więcej błędnych odczytów. W celu ich eliminacji, elementy wyposażono w generator CRC. Ostatni wysyłany bajt to obliczona suma kontrolna CRC. Zerknij do przykładów by dowiedzieć się więcej o obliczaniu tej sumy przez mikrokontroler.

Jeśli stwierdzisz zbyt dużo błędnych transmisji – zrób coś z przewodami transmisyjnymi (ekranowanie, może i rezystor podciągający ma mieć mniejszą rezystancję).

Szybkość transmisji.

W oryginalnej aplikacji magistrali 1Wire, firma Dallas twierdzi, że szybkość transmisji nie przekracza 14Kbit/s. Jeśli to zbyt mało, to niektóre z urządzeń posiadają możliwość jej „podkręcenia”. Wtedy szybkość wzrasta 10-krotnie. Dzieje się tak, gdyż przedziały komunikacyjne zostają skrócone (z 60μs do 6μs), co oczywiście sprawia, że błędy transmisji występują częściej. Szybkość około 140Kbit/s jest zatem możliwa do osiągnięcia, i jak pewnie wiesz to dość dużo. (Jest to około 2 razy szybciej niż modem 56kbps! *przyp. tłumacza*).

Procedury języka BASCOM przeszukujące magistralę, potrafią znaleźć nawet do 50 elementów w ciągu sekundy, a odczytywanie danych z różnych sensorów powinno być możliwe nawet dla 13 urządzeń w ciągu sekundy.

Topologia.

Jeśli chodzi o 1w-net - jest to temat, którym nie będziemy się tu zbytnio zajmować. Star-net, bus-net? Wygląda na to, że możesz to łączyć.

Korzyści ze stosowania magistrali 1Wire.

Każdy z elementów jest indywidualny – możesz się z nimi komunikować tylko poprzez dwa przewody. A i tak, można zaadresować jedno z nich, jeśli trzeba. Policzmy ile ich może być. Istnieje aż 2^{64} (2 do potęgi 64!) unikalnych adresów elementów.

Naturalnie, jeśli większa ilość przewodów połączeniowych jest niewskazana, to jest to największa korzyść. Dodatkowo potrzebna będzie tylko jedna końcówka portu!

Firma Dallas ma w swojej ofercie wiele elementów, przystosowanych do bezpośredniego połączenia z mikrokontrolerem. Żadnych dodatków. Są czujniki - za pomocą których można mierzyć wartości nieelektryczne, są także potencjometry oraz przekaźniki. Można z tego zrobić coś sensownego. A wszystko to na jednej magistrali.

Jeszcze na dokładkę IButton firmy Dallas (Słyszeliście o tym?) oparty na technologii 1Wire. Może jest to coś co warto poznać.

No i BASCOM, pozwalający na używanie układów 1Wire w prosty sposób; który (jak na razie) jest także zaliczany w poczet korzyści – może i jedną z największych. ;-)

Wady magistrali 1Wire.

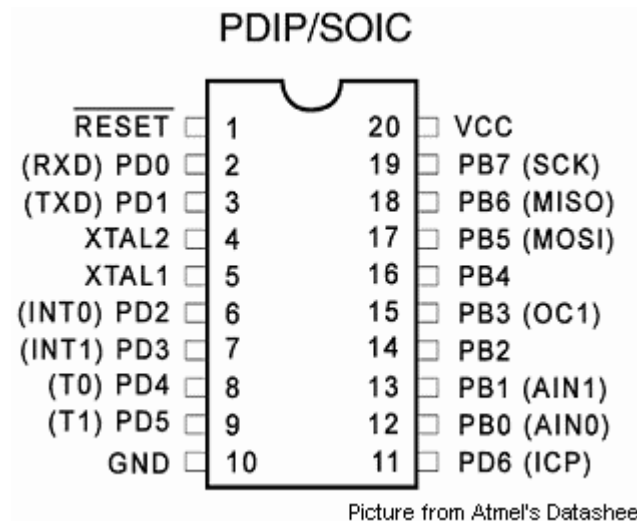
O ile mi wiadomo, firma Dallas jest jedynym producentem układów współpracujących z magistralą 1Wire. Niektórzy zatem twierdzą, że ich układy są kosztowne.

Do tej pory prawdziwą trudność sprawia komunikacja z urządzeniami za pomocą magistrali. Szczególnie, jeśli dołączono do niej wiele różnorodnych urządzeń.

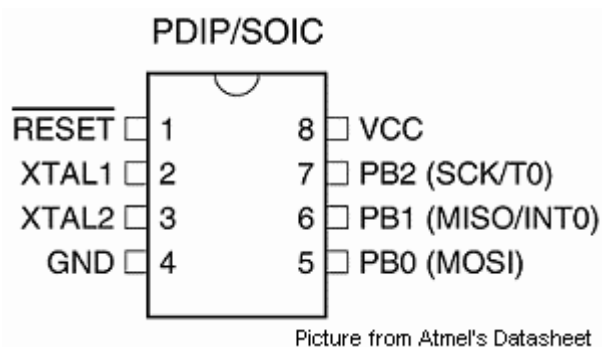
Nadal wielu użytkowników twierdzi, iż magistrala 1Wire jest powolna - lecz ja tak nie myślę.

Göte Haluza
System engineer

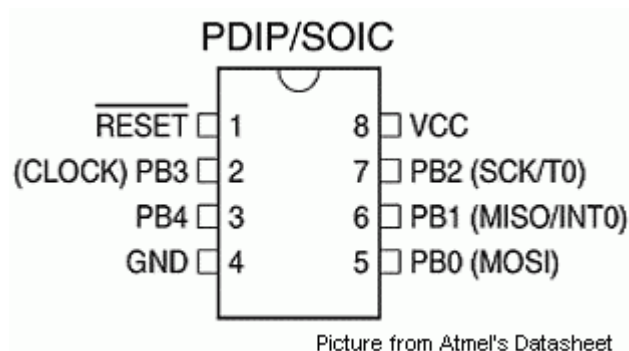
AT90s2313



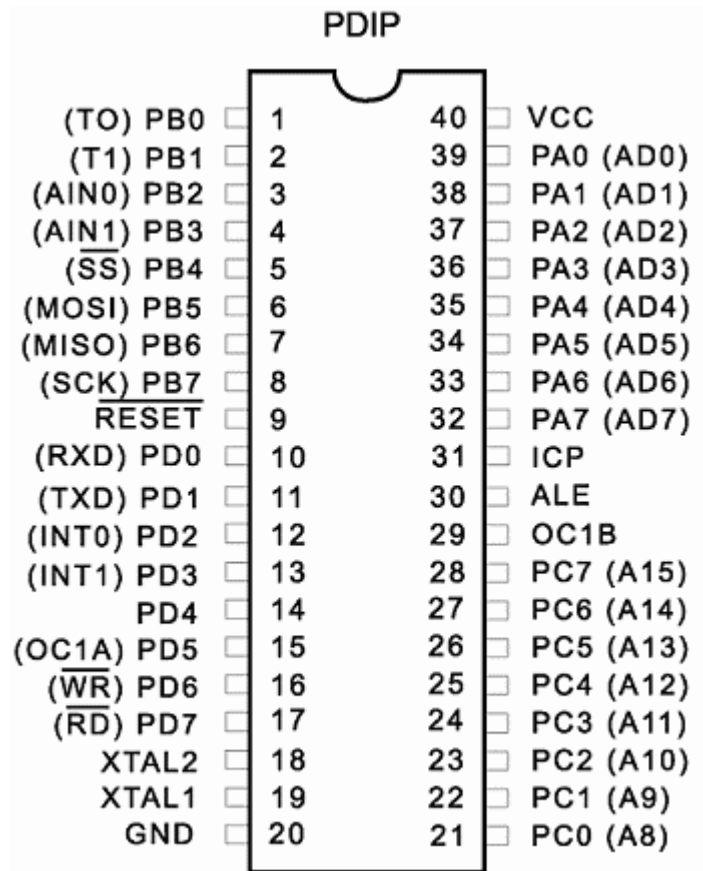
AT90s2323



AT90s2343

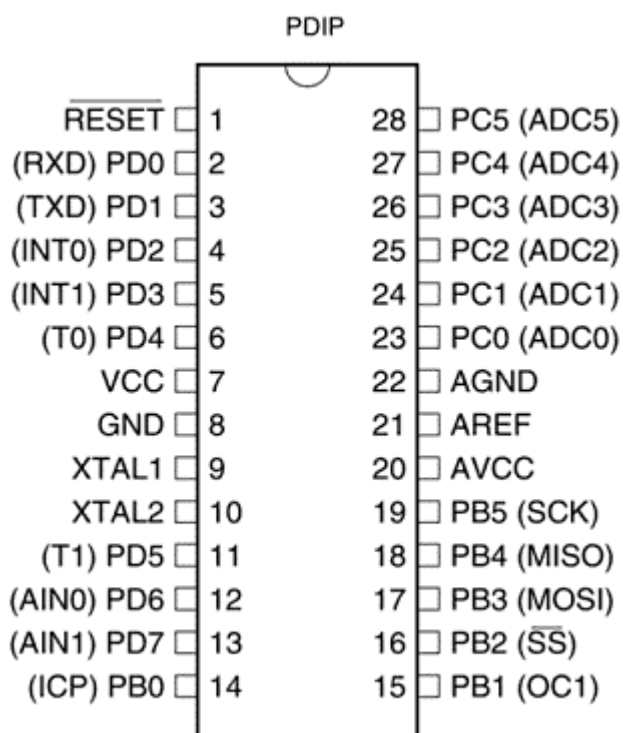


AT90s4414 / AT90s8515



Picture from Atmel's Datasheet

AT90s4433



Picture from Atmel's Datasheet

Podstawy języka BASCOM BASIC

Rozdział ten przedstawia kilka niezbędnych wiadomości, które są podstawą przy tworzeniu programów w języku BASCOM BASIC. W szczególności omówione zostaną następujące kwestie:

- Przeznaczenie poszczególnych symboli z zestawu znaków języka,
- Format linii programu,
- Tworzenie i przeznaczenie etykiet w programie,
- Ograniczenia w długości linii oraz zmiennych,
- Typy danych,
- Tworzenie wyrażeń i używanie operatorów.

Zestaw znaków

Zestaw znaków BASCOM BASIC obejmuje znaki podstawowego alfabetu, liczby oraz znaki specjalne.

Do zestawu znaków podstawowego alfabetu w języku BASCOM BASIC należą duże litery od A do Z oraz ich odpowiedniki w zestawie małych liter. Można także stosować znak dolnej kreski `_`. Polskie znaki diakrytyczne, czyli: `ĄĆĘŁŃÓŚŻąćęłńóśż`; nie wchodzi w skład tego zestawu. Mogą być używane tylko w treści zmiennych tekstowych oraz komentarzy.

W skład liczb języka BASCOM BASIC wchodzi liczby z zakresu 0 do 9 oraz przy zapisie szesnastkowym litery od A do H. Zapis szesnastkowy powinien być poprzedzony przedrostkiem `&H`, a bitowy przedrostkiem `&B`.

Poniżej zestawiono znaki które posiadają szczególne znaczenie w programach w języku BASCOM BASIC:

Tabela 1 Zestaw znaków charakterystycznych.

ENTER	Oznacza koniec linii (Znak ten w nomenklaturze ASCII nazywa się CR)
	Pusty (lub spacja) – znak rozdzielający
'	Apostrof – oznacza początek komentarza
*	Gwiazdka – znak operacji mnożenia
+	Plus – znak operacji dodawania
,	Przecinek – znak rozdzielający argumenty instrukcji
-	Minus – znak operacji odejmowania
.	Kropka – oddziela część całkowitą od ułamkowej
/	Kreska ukośna – znak operacji dzielenia
:	Dwukropek – rozdziela instrukcje zapisane w jednej linii
"	Cudzysłów – rozpoczyna i kończy dane tekstowe
;	Średnik – rozdziela argumenty instrukcji wejścia/wyjścia
<	Mniejszy niż – znak operacji porównywania
=	Znak równości – występuje w operacjach przypisania oraz porównywania
>	Większy niż – znak operacji porównywania
\	Odwrotna kreska ukośna – znak dzielenia dla liczb całkowitych
^	Daszek – znak operacji potęgowania

Struktura pojedynczej linii programu

Linia programu w języku BASCOM BASIC składa się z następujących części:

```
[[identyfikator:]] [[instrukcja]] [[:instrukcja]] ... [[' komentarz]]
```

Identyfikatory i ich używanie

W języku BASCOM BASIC używany jest tylko jeden typ identyfikatora, tak zwana etykieta.

Etykieta taka może się składać z dozwolonych liter i cyfr. Jej długość może zawierać się w granicach od 1 do 32 znaków i musi zaczynać się od litery, a kończyć dwukropkiem. Etykietą nie może być żadna z instrukcji języka, choć nazwy instrukcji mogą zawierać się w treści etykiety. Poniżej przedstawiono kilka poprawnych etykiet:

```
Alpha:
ScreenSUB:
Test3A:
```

Kompilator BASCOM nie rozróżnia dużych i małych liter, więc poniższe etykiety są w rozumieniu kompilatora takie same:

```
alpha:
Alpha:
ALPHA:
```

Etykieta może zaczynać się na dowolnej pozycji linii programu, byle by była umieszczona jako pierwsza w tej linii:

```
Gosub Pisz_Nie
```

```
                                Pisz_Nie: Print "NIE!"
                                Return
```

Instrukcje języka BASCOM BASIC.

Instrukcje języka BASCOM BASIC mogą być typu „wykonywalnego” bądź też „nie wykonywalnego”.

Instrukcje „wykonywalne” tworzą logiczną strukturę działań programu – algorytm. Instrukcje „nie wykonywalne” są odpowiedzialne za część informacyjną oraz organizacyjną programu. Do najważniejszych instrukcji tego typu należą:

- Instrukcja komentarza – rozpoczynająca się słowem REM lub znakiem apostrofu (’),
- Instrukcje deklaracji – rozpoczynające się od słów DIM, DECLARE,
- Dyrektywy kompilatora – rozpoczynające się znakiem dolara (\$).

Instrukcje komentarza rozpoczynające się słowem REM lub znakiem apostrofu są sobie równoważne. Jedyna różnica między nimi to taka, że komentarz rozpoczynający się słowem REM nie będący jedyną instrukcją w linii programu, musi być rozdzielony znakiem dwukropka. Ilustruje to poniższy przykład:

```
Print " Cześć, co jest? " : Rem Rozweselmy użytkownika
Print " Cześć, co jest? " 'Rozweselmy użytkownika
```

Uwaga! Instrukcja REM musi być ostatnią instrukcją w tej linii programu!

Rozdzielanie znakiem dwukropka stosujemy również, gdy w linii programu występuje więcej niż jedna instrukcja:

```
For I = 1 To 5 : Print " Pięć razy TAK! " : Next I
```

Nie zaleca się jednak stosować takich konstrukcji, gdzie nie są wymagane. Zapis taki znacznie utrudnia późniejszą analizę programu. O wiele bardziej przejrzyste wygląda ten sam fragment zapisany z wykorzystaniem wcięć:

```
For I = 1 To 5
    Print " Pięć razy TAK! "
Next I
```

Począwszy od wersji 1.11.7.2 możliwe jest także dzielenie długich linii. Wystarczy na końcu

umieścić znak dolnej kreski: _ oznaczającej przeniesienie do następnej linii. Przykładowo:

```
Print " To jest test " _  
Print " A to dalsza część powyższej linii "
```

Uwaga! Po znaku przeniesienia nie mogą występować inne znaki niż spacje.

Długość linii w języku BASCOM BASIC

Jeśli do tworzenia programu używamy edytora środowiska BASCOM, długość linii nie jest ograniczona. Choć jednym z dobrych nawyków, poprawiających czytelność programu, jest nie przekraczanie 80 znaków w jednej linii. W edytorze środowiska BASCOM wyświetlana jest w tym celu linia pomocnicza.

Typy danych

Każda ze zmiennych użytych w programie posiada określony typ, związany z treścią przechowywanych w niej danych. Tutaj zawarto zbiorcze informacje o tzw. typach elementarnych.

Elementarne typy danych.

W języku BASCOM BASIC zdefiniowano kilka podstawowych typów:

Tabela 2 Elementarne typy danych.

Typ	Rozmiar	Opis
Bit	$\frac{1}{8}$ bajta	Bit może przyjmować tylko dwie wartości: 0 i 1.
Byte	1 bajt	Bajt może przechowywać dowolną dodatnią liczbę całkowitą z zakresu od 0 do 255.
Integer	2 bajty	Typ Integer może przechowywać dowolną liczbę całkowitą z zakresu -32768 do +32767.
Long	4 bajty	Typ Long może przechowywać dowolną liczbę całkowitą z zakresu -2^{32} do $2^{32}-1$.
Word	2 bajty	Typ Word może przechowywać dowolną dodatnią liczbę całkowitą z zakresu od 0 do 65535.
Single	4 bajty	Typ Single może przechowywać dowolną liczbę stałą lub zmiennoprzecinkową.
String	max. 254 bajty	Typ String przechowuje dowolny ciąg znaków o długości nie większej niż 254 znaków. Ciąg ten zakończony jest zawsze znakiem 0. Każdy znak to jeden bajt. Tak więc tekst o długości 10 znaków zajmuje 11 bajtów.

Ogólnie zmienne typu Byte, Integer, Long, Word i Single są nazywane zmiennymi numerycznymi, a typu String: zmiennymi tekstowymi lub ciągami znaków. Taka konwencja zostanie przyjęta w tym dokumencie.

Obszar pamięci przeznaczony na zmienne znajduje się w pamięci RAM procesora. Można to zmienić stosując dyrektywy kompilatora lub określając dla każdej zmiennej obszar pamięci. Informacje na ten temat znajdują się przy opisie instrukcji DIM.

Zmienne

Zmienna to określony obszar pamięci identyfikowany przez nadaną przez programistę nazwę. Wielkość tego obszaru jest ściśle powiązana z ilością pamięci potrzebnej na zapamiętanie zmiennej (patrz poprzednia tabela).

Zmienna zawsze jest kojarzona z konkretnym typem danych. W zmiennych numerycznych można przechowywać tylko dane będące liczbami. Typ String zaś tylko dane będące znakami lub ich ciągiem (który i tak jest ciągiem liczb – kodów liter *przyp. tłumacza*). Poniższe przykłady obrazują kilka możliwości przypisywania zmiennym wartości:

Przypisanie zmiennej wartości:

```
A = 5  
C = 1.1      `zmienna C jest typu Single
```

```
D = "BASCOM" `zmienna D jest typu String
```

Przypisanie zmiennej zawartości innej zmiennej:

```
abc = def  
k = g
```

Przypisanie zmiennej wyniku operacji matematycznej:

```
Temp = a + 5  
Temp = C + 5
```

Przypisanie zmiennej wyniku działania funkcji:

```
Temp = Asc(S) `zmienna S jest typu String o długości 1!
```

Nazwy zmiennych

Nazwy zmiennych w języku BASCOM BASIC, są objęte tymi samymi ograniczeniami co przy nazywaniu etykiet. Ich długość nie powinna przekraczać 32 znaków. Nazwa może składać się z dowolnych liter i liczb – oprócz polskich znaków, oraz rozpoczynać się od litery.

Nazwa zmiennej nie może być którymś z zastrzeżonych słów języka BASCOM BASIC, tj. nazwą instrukcji, rejestru procesora czy dyrektywy. Dla przykładu poniższa zmienna nie jest poprawna, gdyż w języku BASCOM BASIC istnieje instrukcja (operator) AND:

```
AND = 8
```

choć poniższa nazwa jest całkowicie poprawna:

```
ToAND = 8
```

Listę wszystkich zastrzeżonych słów języka BASCOM BASIC można zobaczyć wybierając temat [Słowa zastrzeżone](#).

W instrukcji przypisywania wartości zmiennej można używać zapisu heksadecymalnego (szesnastkowy) bądź dwójkowego (bitowy). Liczby zapisane szesnastkowo muszą być poprzedzone przedrostkiem **&H**, a zapis dwójkowy przedrostkiem **&B**. Poniższy przykład ukazuje tą samą liczbę w zapisie szesnastkowym, dwójkowym i dziesiętnym:

```
a = &HA  
a = &B1010  
a = 10
```

Aby używać zmiennej w programie należy ją najpierw zdefiniować instrukcją **DIM**. Jest to wymagane, gdyż kompilator musi określić rozmiar pamięci przeznaczony na zmienną.

```
Dim b1 As Bit, I As Integer, k As Byte, s As String * 10
```

Zmienne typu String muszą mieć dodatkowy parametr określający przewidywaną długość przechowywanego tekstu - w znakach. W operacjach przypisywania trzeba stosować znaki cudzysłowu:

```
s = "ATMEL"  
s = "8051"
```

Można także użyć specjalnych instrukcji z zestawu: **DEFINT**, **DEFBIT**, **DEFBYTE**, **DEFWORD**, **DEFLNG** lub **DEFSNG**. Instrukcje te ogólnie definiują typ użytych zmiennych w programie. Dla przykładu instrukcja:

```
Defint A
```

spowoduje, że wszystkie zmienne użyte w programie, których nazwa zaczyna się literą A, a nie były zdefiniowane instrukcją DIM, będą zmiennymi typu Integer.

Liczby zmiennoprzecinkowe (programy obsługi dostarczył Jack Tidwell)

Jedynym typem jaki może przechowywać liczby zmiennoprzecinkowe jest typ Single. Sposób reprezentowania zmiennych typu Single w pamięci, został oparty na standardzie IEEE. Liczba taka zajmuje 32 bity (4 bajty). Bity te są podzielone na bit znaku, 8 bitowy eksponent i 23 bitową mantysę:

31	30	23	22	0
zn		eksponent		mantysa

Eksponent jest zapisany jako liczba 8 bitowa ze znakiem w kodzie U2. Jeśli najbardziej znaczący bit jest ustawiony (eksponent > 128) to eksponent jest traktowany jako ujemny. Bit znaku określa znak liczby przechowywanej w zmiennej. Wartość 0 oznacza liczbę dodatnią, 1 zaś ujemną. Mantysa jest przechowywana w znormalizowanym formacie „ukrytego bitu”, dzięki czemu 24 bitowa precyzja może zostać osiągnięta.

Wszystkie operacje matematyczne mogą być przeprowadzane na liczbach typu Single.

Można także dokonywać konwersji (niejawnej) z typu Single na Integer lub Word, i odwrotnie:

```
Dim I As Integer, S As Single
```

```
S = 100.1      'przypisujemy liczbę zmiennoprzecinkową  
I = S          'ta instrukcja zmieni liczbę typu single na Integer
```

Poniżej znajduje się fragment *Microsoft Knowledge Base* dotyczący liczb zmiennoprzecinkowych.

Liczby zmiennoprzecinkowe to złożony temat, który zagmatwał już wielu programistów. Poniższy opis powinien być pomocny w rozpoznawaniu sytuacji w których mogą wystąpić błędy i umożliwi ich uniknięcie. Pozwoli także rozpoznać te przypadki, które spowodowane są przez ograniczenia jakie stawia zapis liczb zmiennoprzecinkowych, w przeciwieństwie do faktycznych błędów kompilatora.

System dziesiętny i dwójkowy

Normalnie obliczenia prowadzone są w systemie dziesiętnym (o podstawie 10), gdzie każdej cyfrze odpowiada kolejna potęga liczby 10. System taki został jakby narzucony przez naturę. Jedynym sensownym powodem że jest on używany to fakt, że ludzie posiadają właśnie 10 palców (u rąk), które tworzą praktyczne narzędzie do liczenia.

Liczbę 532.25 w systemie dziesiętnym można zatem rozłożyć na sumę iloczynów:

$$\begin{aligned} & (5 * 10^2) + (3 * 10^1) + (2 * 10^0) + (2 * 10^{-1}) + (5 * 10^{-2}) \\ = & 500 + 30 + 2 + 2/10 + 5/100 \\ = & 532.25 \end{aligned}$$

W systemie dwójkowym (binarnym - o podstawie 2), każda pozycja reprezentuje jedną z kolejnych potęg liczby 2, podobnie jak w systemie dziesiętnym. Dla przykładu liczba (dwójkowa!) 101.01 oznacza:

$$\begin{aligned} & (1 * 2^2) + (0 * 2^1) + (1 * 2^0) + (0 * 2^{-1}) + (1 * 2^{-2}) \\ = & 4 + 0 + 1 + 0 + 1/4 \\ = & 5.25 \quad (\text{dziesiętnie}) \end{aligned}$$

W jaki sposób zapisywane są liczby całkowite

Ponieważ liczby całkowite nie posiadają części ułamkowej, są one zapisywane w sposób o wiele prostszy niż liczby zmiennoprzecinkowe.

Normalnie liczby całkowite w komputerach klasy PC są zapisywane na dwóch bajtach (16 bitów). Długie liczby całkowite są zaś zapisywane na 4 bajtach. Liczby dodatnie są prostymi liczbami binarnymi (w tzw. kodzie NKB *przyp. tłumacza*). Przykładowo:

```
1 Dec =      1 Bin  
2 Dec =     10 Bin
```

22 Dec = 10110 Bin, itd.

Natomiast liczby ujemne są zapisywane w trochę inny sposób, z wykorzystaniem systemu U2 (uzupełnienia do dwóch). By otrzymać liczbę ujemną w systemie U2 należy wziąć część absolutną (bez znaku) tej liczby, wykonać operację negacji logicznej wszystkich bitów i dodać 1. Na przykład:

```
4 Dec = 0000 0000 0000 0100
        1111 1111 1111 1011      Po negacji
-4      = 1111 1111 1111 1100      Po dodaniu 1
```

Zapis U2 ma jeszcze tą zaletę, że operacja dodawania dowolnej kombinacji dwóch liczb w zapisie U2, daje prawidłowy rezultat.

Komplikacje związane z liczbami zmiennoprzecinkowymi

Każda liczba całkowita może zostać zapisana w notacji binarnej. Liczba zmiennoprzecinkowa już nie. Faktycznie, każda liczba niewymierna o podstawie 10, będzie także niewymierna w każdym systemie o podstawie mniejszej niż 10.

Szczególnie dla systemu dwójkowego, tylko liczby ułamkowe mogące być zapisane w formie ilorazu p/q (gdzie q jest potęgą o podstawie 2), mogą zostać wyrażone dokładnie w skończonej liczbie bitów.

Nawet powszechnie występujące ułamki dziesiętne, takie jak: 0.0001, nie mogą być dokładnie „opisane” w systemie dwójkowym (0.0001 jest powtarzającym się dwójkowym ułamkiem z okresem 104 bitów!).

Wyjaśnia to dlaczego taki prosty przykład jak:

```
SUM = 0
FOR I% = 1 TO 10000
  SUM = SUM + 0.0001
NEXT I%
PRINT SUM                                'Teoretycznie = 1.0
```

wydrukuje w rezultacie 1.000054, gdyż błąd w zapisie liczby 0.0001 w systemie dwójkowym skumuluje się.

Z tego samego powodu, powinno się zawsze uważać przy porównywaniu liczb zmiennoprzecinkowych. Poniższy przykład ilustruje pospolity błąd programistów:

```
item1# = 69.82#
item2# = 69.20# + 0.62#
IF item1# = item2# THEN PRINT "Równe!"
```

Ten fragment programu NIE wydrukuje słowa `Równe!`, ponieważ liczba 69.82 nie może zostać dokładnie zapisana w systemie dwójkowym. Jest to spowodowane tym, że wartość będąca wynikiem przedstawionego działania jest NIEZNACZNIE różna (w systemie dwójkowym) od wartości, która powinna być prawdziwym rezultatem tegoż wyrażenia. W praktyce, powinno się zawsze opisywać takie porównania w sposób uwzględniający pewną tolerancję.

Podstawowa koncepcja liczb zmiennoprzecinkowych

Ważną rzeczą jest by zrozumieć, iż dowolny system zapisu liczb zmiennoprzecinkowych w postaci dwójkowej, może reprezentować w dokładnej formie tylko skończoną liczbę zmiennoprzecinkową. Wszystkie inne wartości muszą być aproksymowane przez najbliższą liczbę dającą się przedstawić w danym systemie. Standard IEEE proponuje właśnie jedną z metod zaokrąglania tychże wartości.

Jako że język BASCOM BASIC używa właśnie tego standardu, stosuje zaokrąglenia liczb według reguł opracowanych przez IEEE.

Tak samo należy pamiętać, że liczby mogące być zapisane w standardzie IEEE są rozkładane w szerokim zakresie. Można to sobie wyobrazić rysując linię z zaznaczonymi liczbami. Duże zagęszczenie panuje wokół liczb 1.0 i -1.0 lecz gęstość ta spada, w kierunku 0 lub nieskończoności.

Celem powstania standardu IEEE, było stworzenie jednolitego systemu - dla obliczeń inżynierskich - który charakteryzował by się maksymalną precyzją (tak aby liczby były maksymalnie dokładnym odzwierciedleniem liczb naturalnych). Precyzja ta odnosi się do ilości liczb, które możliwe są do przedstawienia w tymże standardzie.

Standard IEEE stara się zachować równowagę pomiędzy ilością bitów przeznaczoną na wykładnik a ilością bitów przeznaczoną na część ułamkową liczby; tak by utrzymać zarówno dokładność jak i precyzję w zadowalającym stopniu.

Standard IEEE

Liczby zmiennoprzecinkowe są zapisywane w następującej formie:

$$X = \text{Fraction} * 2^{(\text{Exponent} - \text{Bias})}$$

gdzie:

Exponent to dwójkowy wykładnik,

Fraction to znormalizowana część ułamkowa liczby. Znormalizowana dlatego, iż wykładnik jest „poprawiany” tak by początkowy bit miał wartość 1. W ten sposób, nie jest zapisywana a otrzymywana dodatkowa precyzja. Także dlatego, iż bit ten jest traktowany jako bit niejawny. Można pomyśleć, że ma to związek z notacją naukową, gdzie dokonywana jest manipulacja wykładnikiem, tak by miał jedną cyfrę z lewej strony przecinka; wyjątkowo w systemie binarnym, można zawsze dokonać manipulacji wykładnikiem, tak aby pierwszy bit był jedynką - w tym systemie po prostu są tylko zera i jedynki.

Bias to polaryzacja wartością stałą, stosowana w celu umożliwienia zapamiętywania także ujemnych wykładników. Wartość tej polaryzacji dla liczb pojedynczej precyzji wynosi 127, a dla liczb podwójnej precyzji 1023 (dziesiętnie).

Wartości w których wszystkie bity mają wartość 0 i 1 (binarnie) są zarezerwowane. Występują także inne wyjątki, które wskazują różnorodne błędy.

Przykłady liczb pojedynczej precyzji

$$2 = 1 * 2^1 = 0100\ 0000\ 0000\ 0000 \dots 0000\ 0000 = 4000\ 0000\ \text{hex}$$

Należy zwrócić uwagę że bit znaku jest równy 0, a wykładnik to 128, lub 100 0000 0 w zapisie binarnym, co wynika z operacji 127+1 (dodano 1 do polaryzacji *przyp. tłumacza*). Mantysa zaś to (1.) 000 0000 ... 0000 0000, zapisana z początkowym bitem równym 1; i przecinek, więc mantysa to liczba 1 (dziesiętnie).

$$-2 = -1 * 2^1 = 1100\ 0000\ 0000\ 0000 \dots 0000\ 0000 = C000\ 0000\ \text{hex}$$

Tak samo jak liczba +2 za wyjątkiem bitu znaku, który jest tutaj ustawiony. Jest to prawdziwe dla wszystkich formatów IEEE dla liczb zmiennoprzecinkowych.

$$4 = 1 * 2^2 = 0100\ 0000\ 1000\ 0000 \dots 0000\ 0000 = 4080\ 0000\ \text{hex}$$

Taka sama mantysa, lecz eksponent został zwiększony o 1 (polaryzacja to 129, lub 100 0000 1 binarnie).

$$6 = 1.5 * 2^2 = 0100\ 0000\ 1100\ 0000 \dots 0000\ 0000 = 40C0\ 0000\ \text{hex}$$

Ten sam eksponent, zaś mantysa zwiększona o połowę – to znaczy: (1.) 100 0000 ... 0000 0000, i - podobnie jak w ułamku binarnym – wynosi 1 i ½ (inne wartości czynników to 1/2, 1/4, 1/8, itd.).

$$1 = 1 * 2^0 = 0011\ 1111\ 1000\ 0000 \dots 0000\ 0000 = 3F80\ 0000\ \text{hex}$$

Taki sam eksponent jak inne potęgi 2, ale mantysa jest mniejsza o jeden, przy polaryzacji 127, lub (jak kto woli) 011 1111 1 w systemie binarnym.

$$.75 = 1.5 * 2^{-1} = 0011\ 1111\ 0100\ 0000 \dots 0000\ 0000 = 3F40\ 0000\ \text{hex}$$

Wykładnik przesunięty do 126, czyli 011 1111 0 w systemie binarnym, a mantysa to (1.) 100 0000 ... 0000 0000, co oznacza 1 i ½.

$$2.5 = 1.25 * 2^1 = 0100\ 0000\ 0010\ 0000 \dots 0000\ 0000 = 4020\ 0000\ \text{hex}$$

Dokładnie tak samo jak liczba +2 za wyjątkiem ustawionego bitu reprezentującego wartość ¼ w mantysie.

$$0.1 = 1.6 * 2^{-4} = 0011\ 1101\ 1100\ 1100 \dots 1100\ 1101 = 3DCC\ CCCC\ \text{hex}$$

0.1 to liczba okresowa w zapisie binarnym. Mantysa po prostu unika wartości 1.6, a przesunięty wykładnik wskazuje, że wartość 1.6 musi być podzielona przez 16 (jest to 011 1101 1 w binarnym, a 123 w dziesiętnym). Właściwy eksponent to $123 - 127 = -4$, co oznacza że czynnik przez który należy mnożyć to $2^{-4} = 1/16$. Należy zwrócić uwagę, że mantysa jest zaokrąglona w górę na ostatnim bicie. Jest to próba zapisania liczby nie dającej się zapisać z jak największą dokładnością. (Powodem tego, że liczby 1/10 oraz 1/100 nie są dokładnie odwzorowywane w systemie binarnym jest podobny przypadek z liczbą 1/3 w systemie dziesiętnym, która także nie może być dokładnie odwzorowana.)

$$0 = 1.0 * 2^{-128} = \text{wszystkie bity 0 -- wyjątek.}$$

Inne błędy związane z liczbami zmiennoprzecinkowymi

Poniżej znajduje się lista błędów, wspólnych dla liczb zmiennoprzecinkowych:

1. Błąd zaokrąglenia.

Błąd ten występuje gdy wszystkie bity liczby dwójkowej nie mogą być użyte podczas obliczeń.

Przykładowo:

Należy dodać 0.0001 do 0.9900 (stosując pojedynczą precyzję).

Dziesiętne 0.0001 zostanie zapisane jako:

$$(1.)10100011011011100010111 * 2^{(-14+\text{Bias})} \quad (\text{binarnie: 13 początkowych 0!})$$

a 0.9900 zostanie zapisane w formie:

$$(1.)11111010111000010100011 * 2^{(-1+\text{Bias})}$$

Teraz, aby dodać te dwie wartości, przecinek dziesiętny (w systemie binarnym) musi zostać przesunięty. Dlatego zapis dwójkowy musi pozostać nie znormalizowany. Poniżej znajduje się przeprowadzane działanie:

$$\begin{array}{r} .000000000000011010001101 * 2^0 \quad \leftarrow \text{tylko 11 z 23 bitów jest zachowanych} \\ + .111111010111000010100011 * 2^0 \\ \hline .111111010111011100110000 * 2^0 \end{array}$$

Nazwano to błędem zaokrąglenia, gdyż niektóre komputery dokonują zaokrąglenia podczas przesuwania w operacji dodawania, a inne zaś po prostu obcinają. Błędy zaokrąglenia są bardzo ważne, i zwracają na siebie uwagę ilekroć wykonywana jest operacja dodawania lub mnożenia dwóch bardzo różnych wartości.

2. Odejmowanie dwóch prawie identycznych liczb.

$$\begin{array}{r} .1235 \\ - .1234 \\ \hline .0001 \end{array}$$

Zostanie to znormalizowane. Należy zauważyć, że liczby poddawane operacji posiadają aż cztery cyfry znaczące, za to rezultat posiada tylko jedną cyfrę znaczącą.

3. Przepelnienie i niedopełnienie.

Występuje gdy rezultat działania jest zbyt duży lub zbyt mały by zapisać go w konkretnym typie danych.

4. Błąd kwantyzacji.

Występuje gdy liczby nie mogą zostać zapisane w oryginalnej postaci w przyjętym standardzie zapisu liczb zmiennoprzecinkowych.

Ciągi

Ciągi używane są przede wszystkim do zapamiętywania tekstu. Przy definiowaniu ciągów musi być podana przewidywana długość takiego ciągu.

```
Dim S As String * 5
```

spowoduje utworzenie ciągu który może pomieścić maksymalnie 5 znaków. Zmienna ta zajmie dokładnie 6 bajtów, gdyż na końcu ciągu jest zapamiętywany zawsze znak końca (znak o kodzie zero).

Przypisanie ciągu znaków do zmiennej może odbywać się w ten sposób:

```
s = "abcd"
```

Jeśli nie ma możliwości bezpośredniego wprowadzenia z klawiatury jakiegoś znaku, to można skorzystać z możliwości wprowadzenia jego kodu:

```
s = "AB{027}cd"
```

Znacznik {kod} wstawia do ciągu znak z zestawu ASCII o podanym kodzie. Należy pamiętać, że kod musi być trzycyfrowy. Zapis:

```
s = "{27}"
```

nie spowoduje umieszczenia znaku Escape do ciągu!

Tablice

Tablica to nic innego jak zbiór ponumerowanych jednakowych elementów. Każdy element tablicy posiada swój własny indeks (numer) jednoznacznie identyfikujący go. Zmiana jednego elementu tablicy nie wpływa zatem na resztę elementów przechowywanych w tablicy.

Indeks tablicy jest liczbą całkowitą (bez znaku) i musi się zawierać w przedziale od 1 do 65535. Pierwszy element tablicy ma zawsze numer 1.

Przykład:

```
Dim a(10) As Byte 'definiujemy tablicę a z dziesięcioma elementami
Dim c As Integer

For C = 1 To 10
    a(C) = c        'przypisanie wartość
    Print a(C)      'drukujemy
Next
a(c + 1) = a        'można także obliczać indeksy
```

Wyrażenia i operatory

Przy wszystkich obliczeniach prowadzonych w programie, używane są wyrażenia i operatory. Ten punkt opisuje jak formułować wyrażenia z użyciem operatorów w języku BASCOM BASIC. W szczególności opisowi poddane będą:

- Operacje arytmetyczne, używane podczas obliczeń,
- Operatory relacji, używane podczas operacji porównywania,
- Operatory logiczne, używane przy formułowaniu warunków i manipulacji bitami.

Wyrażenia w języku BASCOM BASIC.

Wyrażenie może składać się z liczb, zmiennych i ich kombinacji połączonych za pomocą operatorów. Operatory używane w wyrażeniach stanowią element działań matematycznych lub logicznych.

Ogólnie można zdefiniować format wyrażenia jako:

```
[[zmienna1][liczba1]] [operator1] [[zmienna2][liczba2]] [operatorN] ...
```

Dla przykładu kilka poprawnych konstrukcji wyrażień:

```
A = A + 1           'dodaj 1 do zawartości zmiennej A
B = 2 + A * 3
If b <> 6 Then b = 6 'są tu dwa wyrażenia!
```

Operacje arytmetyczne

W skład zbioru operacji arytmetycznych wchodzi następujące operacje: + - * \ / oraz ^. Należy rozróżnić operację dzielenia na dwie możliwości:

/ Jest dzieleniem, przy którym otrzymujemy liczbę zmiennoprzecinkową.

```
Dim a As Single
```

```
a = 5 / 2
Print a
```

Wynikiem działania tego programu jest liczba 2.5.

\ Jest dzieleniem, przy którym otrzymujemy liczbę całkowitą. Operator ten powinno się używać tylko i wyłącznie dla zmiennych typu całkowitego (Byte, Integer, Word, Long). Jeśli w powyższym przykładzie zmienna a będzie typu Integer, a dzielenie zastąpimy dzieleniem całkowitym to wynik jaki uzyskamy będzie liczbą 2.

Z dzieleniem całkowitym jest związany inny operator tzw. reszty modulo - MOD. Operator ten pozwala obliczyć resztę z dzielenia całkowitego. Gdyby w przykładowym programie dopisano instrukcję:

```
b = 5 Mod 2
```

to w zmiennej b znalazła by się liczba 1, gdyż $5 = 2 * 2 + 1$

Przekroczenie zakresu i dzielenie przez zero.

Operacja dzielenia przez zero spowoduje błąd. Aktualnie, błąd ten nie generuje stosownego komunikatu, dlatego należy zadbać by nie doszło do dzielenia przez zero.

Operatory relacji

Operatory relacji są używane przy porównywaniu dwóch wartości. Stosowane są przede wszystkim w instrukcjach mających wpływ na sposób działania programu. Lista operatorów używanych w języku BASCOM BASIC znajduje się poniżej:

Tabela 3 Operatory relacji.

Operator	Funkcja	Wyrażenie
=	Równość	$X = Y$
<>	Nierówność	$X <> Y$
<	Mniejszy	$X < Y$
>	Większy	$X > Y$
<=	Mniejszy lub równy	$X <= Y$
>=	Większy lub równy	$X >= Y$

Operacje relacji zawsze zwracają logiczną prawdę jeśli wyrażenie jest prawdziwe lub fałsz jeśli wyrażenie jest fałszywe.

Operatory logiczne

Operatory logiczne stosuje się najczęściej jako łącznik w operacjach relacji. Język BASCOM BASIC operuje czterema poniższymi typami operacji logicznych:

Tabela 4 Operatory logiczne.

Operator	Znaczenie
NOT	Logiczna negacja
AND	Iloczyn logiczny
OR	Suma logiczna
XOR	Suma symetryczna

Operatory logiczne można używać także przy operacjach na bitach. Dla przykładu operator AND jest często wykorzystywany przy maskowaniu bitów. Operator OR przy ustawianiu w stan 1 poszczególnych bitów, a operator XOR może zmieniać stan pojedynczych bitów na przeciwny.

Przykład:

```
A = 63 And 19
Print A
A = 10 Or 9
Print A
A = &B0110 Xor &B0010
Print A
```

Wynikiem działania programu będzie:

```
16
11
4   'czyli &B0100
```

Konwersja typów (jawna i niejawna)

Podczas wykonywania operacji na zmiennych w języku BASCOM BASIC AVR każda zmienna musi być tego samego typu. Dla przykładu:

```
Long = Long1 * Long2
```

gdzie wszystkie składniki wyrażenia są typu Long.

Typy zmiennych poddawane operacji, decydują jakiego modelu operacji matematycznych należy użyć. Na przykład jeśli wartość jest przypisywana zmiennej typu Long, to używany jest fragment biblioteki operacji matematycznych związanych z typem Long.

Tu pojawia się mały problem. Jeśli rezultat będzie typu Long, a przypisanie będzie odbywać dla zmiennej typu Byte:

```
Byte = Long
```

wtedy tylko część LSB zmiennej Long trafi do zmiennej typu Byte.

Gdy zmienna Long będzie zawierała wartość 256, to nie zmieści się ona w zmiennej Byte. Rezultatem tego działania będzie zatem operacja $256 \text{ AND } 255 = 0$.

Oczywiście pozostawiono dowolność użycia dowolnej kombinacji typów. Prawidłowy rezultat takiego działania gwarantowany jest tylko przy używaniu identycznych typów lub gdy rezultat działania zmieści się w zmiennej do której wartość będzie przypisywana.

Przy używaniu typu String obowiązują te same zasady, lecz należy wspomnieć o jednym wyjątku:

```
Dim b As Byte
b = 123      'ok. to jest normalne
b = "A"      'b = 65
```

Jeśli zmienna docelowa (ta do której przypisywana jest wartość *przyp. tłumacza*) jest typu Byte a przypisywana wartość jest stałą typu String – umieszczoną w znakach cudzysłowu – zmiennej docelowej przypisana będzie wartość kodu ASCII tego znaku. Dotyczy to także testowania zmiennych:

```
If b = "A" Then ... 'gdy b = 65
End If
```

Jest to inne podejście niż to stosowane w dialekcie QBasic/VisualBasic, gdzie nie jest możliwa operacja przypisania wartości znakowej do zmiennej typu Byte.

Konwersja dla typu Single

Gdy wystąpi potrzeba zamiany wartości typu Single na typ Byte, Integer, Word bądź Long, wtedy kompilator automatycznie zamieni podaną liczbę jeśli tylko będzie ona typu Single (będzie ułamkowa *przyp. tłumacza*). Dla przykładu operacja przypisania:

```
zmienna_Integer = 10.69
```

spowoduje, że podanej zmiennej typu Integer przypisana zostanie wartość 10. Część ułamkowa będzie odcięta.

Konwersja może być także odwrotna, gdy zmiennej typu Single przypisywana będzie wartość typu Byte, Word, Integer czy Long.

```
zmienna_Single = wartość_Long
```

Słowa zastrzeżone

Poniżej przedstawiono listę wszystkich słów które są zastrzeżone w języku BASCOM BASIC.

Znaki specjalne

!

(

)

\$

\$BAUD
\$BAUD1
\$BGF
\$BOOT
\$CRYSTAL
\$DATA
\$DBG
\$DEFAULT
\$END
\$EEPROM
\$EXTERNAL
\$INCLUDE
\$LCD
\$LCDRS
\$LCDPUTCTRL
\$LCDPUTDATA
\$LCDVFO
\$LIB
\$MAP
\$NOINIT
\$NORAMCLEAR
\$REGFILE
\$ROMSTART
\$SERIALINPUT
\$SERIALINPUT2LCD
\$SERIALOUTPUT
\$SIM
\$TINY
\$WAITSTATE
\$XRAMSIZE
\$XRAMSTART

1

1WIRECOUNT()
1WRESET
1WREAD
1WSEARCHFIRST()
1WSEARCHNEXT()
1WWRITE
1WVERIFY

A

ACK
ABS()
ALIAS
AND
AS

ASC()
AT

B

BAUD
BCD()
BIT
BIN()
BINVAL()
BIN2GREY()
BITWAIT
BLINK
BOOLEAN
BYTE
BYVAL

C

CALL
CAPTURE1
CASE
CHR()
CLS
CLOSE
COMPARE1A
COMPARE1B
CONFIG
CONST
COUNTER
COUNTER0
COUNTER1
COUNTER2
CPEEK()
CPEEKH()
CRC8()
CRC16()
CRYSTAL
CURSOR

D

DATA
DATE\$
DEBOUNCE
DECR
DECLARE
DEFBIT
DEFBYTE
DEFLNG
DEFWORD
DEGSNG
DEFLCDCHAR
DEFINT
DEFWORD
DELAY
DIM
DISABLE
DISPLAY
DO
DOWNT0
DTMFOUT

E

ECHO
ELSE
ELSEIF
ENABLE
END
ERAM
ERASE
ERR
EXIT
EXP()
EXTERNAL

F

FOR
FORMAT()
FOURTH
FOURTHLINE
FUNCTION
FUSING()

G

GATE
GETAD()
GETRC5()
GOSUB
GOTO
GRAPHLCD
GREY2BIN()

H

HEXVAL()
HIGH()
HIGHW()
HOME

I

I2CRECEIVE
I2CSEND
I2CSTART
I2CSTOP
I2CRBYTE
I2CWBYTE
IDLE
IF
INCR
INITLCD
INKEY
INP()
INPUT
INPUTBIN
INPUTHEX
INT0
INT1
INTEGER
INTERNAL
INSTR
IS

L

LCASE()
LCD
LEFT
LEFT()
LEN()
LOAD
LOCAL
LOCATE
LOG()
LONG
LOOKUP()
LOOKUPSTR()
LOOKDOWN()
LOOP
LTRIM()
LOW()
LOWER
LOWERLINE

M

MAKEBCD()
MAKEDEC()
MAKEINT()
MAX()
MID
MID()
MIN()
MOD
MODE

N

NACK
NEXT
NOBLINK
NOSAVE
NOT

O

OFF
ON
OR
OUT
OUTPUT

P

PEEK()
POKE
POPALL
PORTA
PORTB
PORTC
PORTD
PORTE
PORTF
POWERDOWN
PRINT
PRINTBIN
PSET

PULSEIN
PULSEOUT
PUSHALL
PWM1A
PWM1B

R

READ
READEEPROM
REM
RESET
RESTORE
RETURN
RIGHT
RIGHT()
ROTATE
RTRIM()

S

SELECT
SERIAL
SERIALIN
SERIALOUT
SERVOS
SET
SHIFT
SHIFTLCD
SHIFTCURSOR
SHIFTIN
SHIFTOUT
SHOWPIC
SOUND
SPACE()
SPC()
SPIINIT
SPIIN
SPIMOVE
SPIOUT
START
STEP
STCHECK
STR()
STRING()
STOP
STOP TIMER
SUB
SWAP

T

THEN
THIRD
THIRDLINE
TIME\$
TIMER0
TIMER1
TIMER2
TO
TOGGLE
TRIM()

U

UCASE()
UNTIL
UPPER
UPPERLINE

V

VAL()
VARPTR()

W

WAIT
WAITKEY()
WAITMS
WAITUS
WATCHDOG
WRITEEEPROM
WEND
WHILE
WORD

X

XOR
XRAM

Różnice w stosunku do BASCOM Basic 8051

BASCOM AVR został zaprojektowany w taki sposób by był - o ile to możliwe - kompatybilny z BASCOM-8051. Niestety niektóre z instrukcji musiały zostać usunięte, niektóre zmienione a jeszcze inne dodane.

Należy jednak uważać, gdyż wprowadzone zmiany mogą w przyszłości „dosięgnąć” także BASCOM-a 8051.

Instrukcje które usunięto.

Tabela 5 Lista usuniętych instrukcji.

Instrukcja	Opis
\$LARGE	Nie potrzebna.
\$ROMSTART	Kod zawsze rozpoczyna się od adresu 0. Przywrócono w wersji 1.11.6.2
\$LCDHEX	Użyj konstrukcji LCD HEX(zmienna)
\$NOINIT	Nie potrzebna. Przywrócona w wersji 1.11.6.2
\$NOSP	Nie potrzebna
\$NOBREAK	Nie może być używana, gdyż nie ma zarezerwowanych kodów w liście rozkazów
\$OBJ	Usunięta.
BREAK	Nie może być używana, gdyż nie ma zarezerwowanych kodów w liście rozkazów
PRIORITY	Procesory AVR nie pozwalają na ustawianie priorytetu przerw
PRINTHEX	Można używać PRINT HEX(zmienna)
LCDHEX	Można używać LCDT HEX(zmienna)

Instrukcje i funkcje które dodano.

Tabela 6 Lista dodanych instrukcji i funkcji.

Instrukcja	Opis
FUNCTION	Można definiować własne funkcje
LOCAL	Można definiować zmienne lokalne w procedurach i funkcjach
^	Nowy operator matematyczny: potęgowanie. 2^3 , czyli $2 * 2 * 2$
SHIFT	Ponieważ instrukcja ROTATE została zmieniona, dodano instrukcję SHIFT. SHIFT działa podobnie do ROTATE, lecz jeśli zawartość zmiennej będzie przesuwana w lewo, bit LSB będzie zerowany i znacznik przeniesienia nie zostanie wsunięty na miejsce bitu LSB
LTRIM()	LTRIM usuwa wszystkie wiodące spacje z podanego ciągu znaków
RTRIM()	RTRIM usuwa wszystkie końcowe spacje z podanego ciągu znaków
TRIM()	TRIM usuwa zarówno wszystkie wiodące jak i końcowe spacje

Instrukcje których działanie zmieniono.

Tabela 7 Instrukcje które zmieniono.

Instrukcja	Opis
ROTATE	ROTATE działa teraz jak w języku assemblera. Oznacza to, że znacznik przeniesienia jest wprowadzany do zmiennej jako bit MSB lub bit LSB

<i>Instrukcja</i>	<i>Opis</i>
CONST	Można definiować ciągi znaków jako stałe w instrukcji CONST. Zmiana dodatkowo zwiększa kompatybilność z językiem QBasic
DECLARE	Dodano argument BYVAL gdyż obsługiwane są teraz „prawdziwe” procedury
DIM	Można podawać bezwzględny adres definiowanej zmiennej

#IF-ELSE-ENDIF

Przeznaczenie:

Dyrektywy preprocesora, dające możliwość kompilacji warunkowej.

Składnia:

```
#IF warunek
    ciąg_instrukcji
#ELSE
    ciąg_instrukcji
#ENDIF
```

gdzie:

<i>warunek</i>	wyrażenie, od którego zależy czy instrukcje po słowie IF będą skompilowane,
<i>ciąg_instrukcji</i>	dowolny ciąg instrukcji.

Opis:

Dyrektywy te pozwalają na użycie w programie tzw. kompilacji warunkowej.

Co to jest kompilacja warunkowa?

Normalnie poddany kompilacji jest cały tekst programu. Można jednak sprawić, by w zależności od podanego warunku, kompilator nie brał pod uwagę jakiegoś fragmentu tekstu. W tym celu należy użyć dyrektyw preprocesora, które spełnią to zadanie.

Dyrektywy kompilacji warunkowej opierają się na stałych, których wartość jest testowana zgodnie z podanym warunkiem podczas kompilacji. Dlatego przed instrukcjami kompilacji warunkowej, należy ją zdefiniować. Poniżej znajduje się prosty przykład, wyjaśniający całą ideę:

```
Const test = 1
#if test
    Print "To będzie skompilowane"
#else
    Print "A to nie"
#endif
```

Uwaga! Nie ma THEN w dyrektywie #IF, oraz #ENDIF to nie to samo co END IF.

Użycie #ELSE jest opcjonalne, lecz należy pamiętać o dyrektywie #ENDIF.

Język BASCOM BASIC posiada kilka wewnętrznych stałych, które mogą być użyte w charakterze składników warunku. Przedstawione poniżej są generowane automatycznie przez kompilator (wartości przykładowe!):

```
_CHIP      = 0
_RAMSIZE   = 128
_ERAMSIZE  = 128
_SIM       = 0
_XTAL      = 4000000
_BUILD     = 11162
```

_CHIP przechowuje liczbę typu Integer określającą rodzaj procesora, w tym wypadku AT90s2313.
_RAMSIZE określa ilość pamięci SRAM w bajtach.
_ERAMSIZE określa ilość pamięci EEPROM w bajtach.
_SIM jest ustawiana jako 1 gdy użyto w programie dyrektywy \$SIM.
_XTAL zawiera informacje o częstotliwości użytego rezonatora kwarcowego w Hz.
_BUILD zawiera numer wersji kompilatora BASCOM.

Numer wersji można z powodzeniem użyć, gdy napisany program ma być kompilowany przez

tą wersję języka BASCOM, która obsługuje daną instrukcję. Na przykład:

```
#if _BUILD >= 11162
    s = Log(1.1)
#else
    Print "Niestety ta funkcja może być użyta w wersji 1.11.6.2 lub
    nowszej"
#endif
```

\$ASM

Przeznaczenie:

Rozpoczyna blok wstawki assemblerowej.

Składnia:

```
$ASM
...
$END ASM
```

Opis:

Dyrektywa \$ASM używana jest razem z dyrektywą \$END ASM, podczas tworzenia bloków kodu w języku assembler, umieszczanych jako wstawki w programach języka BASCOM BASIC. Zamiast używać tych dyrektyw, można także przed każdą linią zawierającą mnemonik postawić znak !.

Większość mnemoników assemblera nie potrzebuje przedrostka !, gdyż są one rozpoznawane automatycznie przez kompilator BASCOM. Choć istnieją instrukcje języka BASCOM BASIC, brzmiące tak samo jak mnemoniki: OUT, SWAP; które dla odróżnienia muszą być poprzedzone wykrzyknikiem.

Zobacz także: [Wstawki assemblerowe](#) , [Lista rozkazów](#)

Przykład:

```
Dim c as Byte

Loadadr c,X    'załaduj adres zmiennej C do rejestru X
$asm
    Ldi R24,1    ;załaduj do rejestru R24 liczbę 1
    St X,R24     ;wpisz zawartość rejestru R24 do zmiennej C
$end asm
Print c

End
```

\$BAUD

Przeznaczenie:

Zmienia ustawienie szybkości pracy wbudowanego układu UART. Użycie tej dyrektywy w programie zastępuje wartość podaną w opcjach kompilatora.

Składnia:

\$BAUD = wartość

gdzie:

wartość

Liczba określająca szybkość pracy łącza RS 232 w bitach na sekundę (bod). Wartość nie może być zmienną, lub liczbą obliczaną podczas kompilacji.

Opis:

Szybkość transmisji jest określana w zakładce [Options | Compiler | Communication](#) i jest

zapisywana w pliku konfiguracyjnym. Dyrektywę \$BAUD pozostawiono dla kompatybilności z kompilatorem BASCOM-8051.

W generowanym raporcie kompilacji, znajduje się informacja o aktualnej szybkości transmisji oraz o procentowej różnicy między ustawioną a rzeczywiście wygenerowaną szybkością transmisji.

Uwaga! Podczas symulacji programu, nie jest możliwe wykrycie problemów z szybkością transmisji, jaka została ustawiona w programie. W docelowym systemie błędna szybkość transmisji może doprowadzić do pojawienia się „krzaczków” w oknie terminala. Aby temu zapobiec należy używać standardowo przyjętych wartości szybkości transmisji oraz stosować rezonator kwarcowy o częstotliwości będącej wielokrotnością tej szybkości.

Zobacz także: [\\$CRYSTAL](#), [BAUD](#)

Przykład:

```
$baud = 2400
$crystal = 14000000      'zegar 14 MHz

Print "Hello"
'Teraz zmieniamy programowo szybkość transmisji
Baud = 9600

Print "Czy zmieniłeś szybkość transmisji w programie terminala?"

End
```

[\\$BAUD1](#) (Nowość w wersji 1.11.6.8)

Przeznaczenie:

Zmienia ustawienie szybkości pracy drugiego wbudowanego układu UART. Użycie tej dyrektywy w programie zastępuje wartość podaną w opcjach kompilatora.

Składnia:

\$BAUD1 = wartość

gdzie:

wartość	Liczba określająca szybkość pracy łącza RS 232 w bitach na sekundę (bod). Wartość nie może być zmienną, lub liczbą obliczaną podczas kompilacji.
---------	--

Opis:

W niektórych kontrolerach firma Atmel wbudowała drugi sprzętowy układ UART. Dyrektywa \$BAUD1 zmienia szybkość transmisji tego układu.

Szybkość transmisji jest określana w zakładce [Options | Compiler | Communication](#) i jest zapisywana w pliku konfiguracyjnym.

W generowanym raporcie kompilacji, znajduje się informacja o aktualnej szybkości transmisji oraz o procentowej różnicy między ustawioną a rzeczywiście wygenerowaną szybkością transmisji.

Uwaga! Podczas symulacji programu, nie jest możliwe wykrycie problemów z szybkością transmisji, jaka została ustawiona w programie. W docelowym systemie błędna szybkość transmisji może doprowadzić do pojawienia się „krzaczków” w oknie terminala. Aby temu zapobiec należy używać standardowo przyjętych wartości szybkości transmisji oraz stosować rezonator kwarcowy o częstotliwości będącej wielokrotnością tej szybkości.

Zobacz także: [\\$CRYSTAL](#) , [BAUD](#) , [\\$BAUD](#)

Przykład:


```

$baud1 = 2400
$crystal = 14000000      'zegar 14 MHz

Open "COM2:" For Binary As #1
Print #1 , "Hello"
'Teraz zmieniamy programowo szybkość transmisji
Baud #1 , 9600

Print #1 , "Czy zmieniłeś szybkość transmisji w programie terminala?"
Close #1

End

```

\$BGF

Przeznaczenie:

Dołącza plik graficzny w formacie BASCOM Graphics File (BGF).

Składnia:

\$BGF "nazwa_pliku"

gdzie:

nazwa_pliku Nazwa (ze ścieżką) pliku graficznego do dołączenia.

Opis:

Aby wyświetlić tak zapisany obrazek należy użyć instrukcji SHOWPIC.

Zobacz także: **SHOWPIC** , **PSET** , **CONFIG GRAPHLCD**

Przykład:

```

Dim X As Byte, Y As Byte
Showpic 1 , 1 , Graphdata      'pokażemy logo
For X = 0 To 10
  For Y = 0 To 10
    Pset X , Y , 1              'i narysujemy jeszcze fajny kwadrat
  Next
Next

End

Graphdata:
  $bgf "logo.bgf"

```

\$BOOT (Nowość w wersji 1.11.6.8)

Przeznaczenie:

Instruuje kompilator by dołączył obsługę bootloadera.

Składnia:

\$BOOT = *adres*

gdzie:

adres adres procedury obsługi bootloadera.

Opis:

Niektóre z nowych procesorów AVR posiadają specjalny fragment pamięci, znajdujący się na

końcu pamięci Flash, przeznaczony na procedurę obsługi bootloadera.

Ustawiając niektóre bity zabezpieczające (*fusebits*), można wybrać jak duży obszar pamięci przeznaczony będzie na procedurę bootloadera. Rozmiar kodu ma także wpływ na adres początkowy bootloadera.

Za pomocą bootloadera można przeprogramować układ, gdy wystąpi pewien specyficzny warunek. W przytoczonym przykładzie sprawdzany jest stan określonej końcówki portu, by stwierdzić czy uruchomiona ma być procedura obsługi bootloadera. Gdy końcówka ta będzie w stanie niskim, wykonywany jest skok pod odpowiedni adres.

Kod programu bootloadera musi być umieszczony na końcu programu. Musi być także napisany w języku assemblera, gdyż nie powinien mieć dostępu do programu w pamięci Flash. Inaczej można by było uszkodzić działający program!

Przykład został napisany dla procesora M163. Można użyć opcji **Upload** emulatora terminala by załadować nowy plik w formacie HEX. Emulator terminala musi mieć ustawioną identyczną szybkość transmisji co układ. W menu **Options | Monitor**, należy ustawić odpowiednią szybkość dla operacji *Upload* oraz opóźnienie dla monitora na 20. Zapisywanie pamięci Flash trwa dość długo, gdyż po każdej linii należy wprowadzić stosowne opóźnienie, podczas operacji *Upload*.

Zobacz także: Programy BOOT128.BAS oraz BOOT128X.BAS umieszczone w katalogu SAMPLES.

Przykład (częściowy):

Zobacz kompletny przykład umieszczony w pliku BOOT.BAS.

```
'-----  
'                                BOOT.BAS  
'  
'      Przykład Bootloadera dla procesora M163  
'Ustaw fusebit o adresie $FE by 512 bajtów przeznaczyć na bootloader  
'Po restarcie wyświetli się komunikat. Jeśli końcówka PIND.7 będzie  
'w stanie niskim program przejdzie do procedury bootloadera.  
'Ten program jest tylko przykładem. Może zostać zmieniony dla innych  
procesorów.  
'W szczególności zmianie podlegać będzie adres wejściowy do procedury  
bootloadera  
'-----  
  
'Ustawienia komunikacji  
$crystal = 4000000  
$baud = 19200  
  
Print "Sprawdzanie..."  
Portd.7 = 1  
If Pind.7 = 0 Then  
    Print "Przechodzę do Bootloadera"  
    Jmp $1e00  
'Skocz pod adres gdzie umieszczono procedurę bootloadera.  
'Zajrzyj do noty katalogowej by poznać więcej szczegółów  
End If  
Print "Bootloader nie aktywny"  
  
'twój kod możesz umieścić tutaj  
End
```

\$CRYSTAL

Przeznaczenie:

Zmienia ustawienie częstotliwości zegara taktującego procesor. Użycie tej dyrektywy w programie zastępuje wartość podaną w opcjach kompilatora.

Składnia:

\$CRYSTAL = wartość

gdzie:

wartość

Liczba określająca częstotliwość (w Hz!) kwarcu lub oscylatora taktującego procesor. Wartość nie może być zmienną, lub liczbą obliczaną podczas kompilacji.

Opis:

Częstotliwość ta jest określana w menu [Options | Compiler | Communication](#) i jest zapisywana w pliku konfiguracyjnym. Dyrektywę \$CRYSTAL pozostawiono dla kompatybilności z kompilatorem BASCOM-8051.

Zobacz także: [\\$BAUD](#), [BAUD](#)

Przykład:

```
$baud = 2400
$crystal = 14000000
Print "Hello"

End
```

\$DATA

Przeznaczenie:

Określa, że dane umieszczone w liniach DATA mają trafić do pamięci kodu.

Składnia:

\$DATA

Opis:

Standardowo dane umieszczone w liniach DATA, trafiają w trakcie kompilacji do pamięci kodu.

Procesory serii AVR mają także wbudowaną pamięć EEPROM, której obsługą zajmują się instrukcje [WRITEEEPROM](#) oraz [READEEPROM](#).

By umieścić dane w pamięci EEPROM, można także zastosować w programie ciąg instrukcji [DATA](#), poprzedzony dyrektywą \$EEPROM. Podczas kompilacji jest tworzony plik z rozszerzeniem .EEP, do którego trafiają te dane. Plik ten należy wykorzystać podczas programowania pamięci EEPROM procesora.

Aby przywrócić standardowe działanie instrukcji DATA – tj. dane mają trafić do pamięci kodu, należy poprzedzić te linie dyrektywą \$DATA.

Uwaga! Instrukcja [READ](#) służąca do odczytywania danych z linii DATA, nie może być stosowana do odczytu danych umieszczanych w pamięci EEPROM.

Zobacz także: [\\$EEPROM](#) , [WRITEEEPROM](#) , [READEEPROM](#)

Przykład:

```
'-----
'                                     READDATA.BAS
'                                     Copyright 2000 MCS Electronics
'-----

Dim A As Integer , B1 As Byte , Count As Byte
Dim S As String * 15
Dim L As Long

Restore Dta1                                'ustawiamy wskaźnik
```

```

For Count = 1 To 3                                'czytamy trzy dane
    Read B1 : Print Count ; " " ; B1
Next

Restore Dta2                                        'ustawiamy wskaźnik
For Count = 1 To 2                                'czytamy dwie dane
    Read A : Print Count ; " " ; A
Next

Restore Dta3
Read S : Print S
Read S : Print S

Restore Dta4
Read L : Print L                                'dane typu Long

End

Dta1:
    Data &B10 , &HFF , 10

Dta2:
    Data 1000% , -1%

Dta3:
    Data "Witaj" , "Świecie"
'Uwaga! Dane typu Integer (>255 lub <0) muszą być zakończone znakiem %
'Typ danych odczytywanych instrukcją READ i zapisanych w liniach DATA
'musi się zgadzać.

Dta4:
    Data 123456789&
'Uwaga! Dane typu Long muszą być zakończone znakiem &
'
'Należy także pamiętać o zgodności typów umieszczonych danych i
'zmiennej podanej w instrukcji READ

```

\$DBG (Nowość w wersji 1.11.6.8)

Przeznaczenie:

Włącza możliwość śledzenia odwołań procedur do stosu.

Składnia:

\$DBG

Opis:

Obliczanie wielkości sprzętowego i programowego stosu, oraz wielkości tzw. „ramki” może być problematyczne. Dlatego w tej wersji języka BASCOM BASIC wbudowano mechanizm ułatwiający obliczenie tych wartości oraz śledzenia odwołań do stosu.

Podanie na początku programu dyrektywy \$DBG spowoduje, że kompilator wypełni wspomniane obszary specjalnymi znacznikami – znakami:

- w obszarze tzw. ramki wpisane będą znaki **F**. Gdy ramka będzie miała wielkość 4 bajtów wpisane będzie **FFFF**.
- w obszarze sprzętowego stosu wpisane będą znaki **H**. Gdy stos będzie miał wielkość 8 bajtów wpisane będzie **HHHHHHHH**.
- w obszarze programowego stosu wpisane będą znaki **S**. Gdy stos będzie miał wielkość 6 bajtów wpisane będzie **SSSSSS**.

Idea mechanizmu śledzenia jest prosta: Gdy znaki te zostaną nadpisane, oznaczać to będzie, że jakaś część stosu bądź „ramki” była używana. Dlatego przeglądając przestrzeń zajmowaną przez stos i „ramkę”, można się zorientować czy obszary te były używane i w jakim stopniu.

Specjalna instrukcja **DBG** przesyła przez sprzętowy układ UART rekordy, z obrazem zawartości obszaru stosu i „ramki” tzw. *dump*. Rekordy te należy zapisać do pliku, gdyż można je później poddać analizie narzędziem **Tools | Stack Analyzer**.

Aby dokonać obliczenia właściwych wartości rozmiaru „ramki” i stosu, wykonaj podane niżej kroki:

- Określ rozmiar „ramki” na 40, programowy stos na 20 oraz sprzętowy stos na 50 bajtów.
- Dodaj na początku do programu dyrektywę **\$DBG**.
- Dodaj na początku każdej procedury i funkcji instrukcję **DBG**.
- Otwórz okno emulatora terminala i otwórz nowy plik zdarzeń (*log file*). Normalnie posiada on taką samą nazwę jak bieżący program. Oczywiście z rozszerzeniem **.LOG**.
- Uruchom program i zobacz czy przesyła on rekordy, sprawdzając czy pojawiają się one w oknie emulatora terminala.
- Gdy program wykona wszystkie procedury lub opcje przewidziane w programie, wyłącz opcję przekazywania danych do pliku zdarzeń oraz zatrzymaj program.
- Wybierz opcję menu: **Tools | Stack analyzer**.
- Powinno się pojawić okienko z danymi pochodzącymi z pliku zdarzeń.
- Naciśnij teraz przycisk *Advise* by obliczone zostały odpowiednie wartości. Upewnij się, że co najmniej jeden ze znaków **H**, **S** oraz **F** znajduje się w danych. Jeśli ich nie ma, oznacza to, że zostały one nadpisane i należy zwiększyć liczby określające rozmiar stosu lub „ramki”. Należy także uruchomić ponownie cały proces.
- Naciśnij teraz przycisk *Use* by użyć wyliczonych ustawień.

Alternatywnym rozwiązaniem jest obserwacja przestrzeni stosu w jednym z okien symulatora i śledzenie czy znaki są zmieniane czy nie.

Instrukcja **DBG** używa specjalnie utworzonej zmiennej tekstowej: `__SUBROUTINE`. Ponieważ nazwa procedury może mieć maksymalnie 32 znaki, zmienna ta ma rozmiar 33 bajtów!

Do zmiennej `__SUBROUTINE` wpisywana jest nazwa aktualnie wykonywanej procedury lub funkcji. Gdy najpierw wywoływana będzie procedura `Test1234`, jej nazwa zostanie przypisana do zmiennej `__SUBROUTINE`. Gdy wykonywana będzie następna instrukcja **DBG** – na przykład w procedurze `Test` - do zmiennej trafi słowo `Test`.

Uwaga! Znaki `234` (pochodzące z poprzedniej nazwy procedury!) pozostaną w przestrzeni zajmowanej przez zmienną i znajdują się w pliku zdarzeń.

Sub	FS	SS	HS	Frame space	Soft stack	HW stack
TEST	1	4	4	a	SSSS0x0x	Wx0x
TEST	1	4	4	I	SSSS0x0x	Wx0x
TEST	1	4	4	a	SSSS0x0x	Wx0x
TEST	1	4	4	I	SSSS0x0x	Wx0x
TEST	1	4	4	a	SSSS0x0x	Wx0x
TEST	1	4	4	I	SSSS0x0x	Wx0x
TEST	1	4	4	a	SSSS0x0x	Wx0x
TEST	1	4	4	I	SSSS0x0x	Wx0x
TEST	1	4	4	a	SSSS0x0x	Wx0x

Hardware stack
 Frame space

Software stack

Rysunek 16 Okno analizatora z wpisanymi rekordami.

Każdy rekord wysyłany przez instrukcję DBG, jest wyświetlany jako jeden rząd w tablicy. Kolumny zaś oznaczają:

Kolumna	Znaczenie:
Sub	Nazwa procedury lub funkcji gdzie używana była instrukcja DBG.
FS	Używana przestrzeń ramki.
SS	Używana przestrzeń programowego stosu.
HS	Używana przestrzeń sprzętowego stosu.
Frame space	Zawartość ramki.
Soft stack	Zawartość programowego stosu.
HW stack	Zawartość sprzętowego stosu.

Przestrzeń zwana „ramką” służy do przechowywania tymczasowych i lokalnych (w procedurach i funkcjach) zmiennych. Przechowuje także zawartość zmiennych przekazywanych do procedur i funkcji przez wartość (argument **BYVAL**).

Ponieważ instrukcje PRINT, INPUT oraz konwersja liczb zmiennoprzecinkowych na ich postać tekstową (przy drukowaniu), potrzebują pewnego obszaru pamięci jako bufor, kompilator przydziela im także 16 bajtów z puli „ramki”.

Dlatego, gdy *Stack Analyzer* określi zapotrzebowanie na 2 bajty ramki, w rzeczywistości ustawi jej rozmiar na 18 (16 + 2). Dla przykładu, gdy w programie użyta zostanie instrukcja:

```
Print zmienna, zmienna
```

wymagana będzie konwersja zawartości zmiennej (liczby) na jej postać tekstową. Dotyczy to w tym samym stopniu instrukcji **LCD**.

Alternatywą dla bufora mogłoby być ustawienie bufora tymczasowego i zwolnienie go po zakończeniu. Wymagałoby to oczywiście więcej zasobów dla kodu programu

W poprzednich wersjach języka BASCOM BASIC jako bufor używany był początek przestrzeni ramki, lecz powodowało to konflikty, gdy zawartość zmiennych była drukowana w procedurach obsługi przerwań.

Rozwiązanie dotyczące bufora będzie zmienione w przyszłych wersjach języka BASCOM BASIC, gdy zostanie zastosowane inne podejście do tego problemu.

Zobacz także: **DBG**

\$DEFAULT

Przeznaczenie:

Określa rodzaj pamięci w której umieszczone będą zmienne.

Składnia:

```
$DEFAULT typ_pam
```

gdzie:

typ_pam rodzaj pamięci: SRAM, XRAM, ERAM

Opis:

Każda ze zmiennych standardowo jest umieszczona w pamięci RAM procesora tzw. SRAM. Można jednak zmienić typ pamięci przeznaczonej na zmienną, określając typ tej pamięci. Na przykład:

```
Dim B As XRAM Byte
```

spowoduje umieszczenie zmiennej B w pamięci zewnętrznej (XRAM). Gdyby wszystkie zmienne miały być umieszczone w innym obszarze pamięci, zamiast dla każdej zmiennej określać obszar pamięci, można zastosować dyrektywę **\$DEFAULT**:

```
$default XRAM
```

Wtedy zmienne określane instrukcją **DIM** będą umieszczane w określonym obszarze pamięci.

Aby przywrócić domyślny obszar pamięci na zmienne, należy użyć dyrektywy \$END \$DEFAULT.

Uwaga! Zmienne bitowe są umieszczane zawsze w pamięci SRAM.

Zobacz także: DIM

Przykład:

```
$default XRAM
Dim A As Byte, b As Byte, C As Byte
'zmienne a,b oraz c będą umieszczone w pamięci XRAM

$default SRAM
Dim D As Byte
'zmienna D będzie umieszczona w pamięci SRAM
```

\$EEPLEAVE (Nowość w wersji 1.11.7.3)

Przeznaczenie:

Informuje kompilator by nie zmieniał lub usuwał pliku .EEP.

Składnia:

\$EEPLEAVE

Opis:

Jeśli istnieje potrzeba umieszczenia danych w pamięci EEPROM procesora, a dane te zostały utworzone przez inne narzędzie, można w programie umieścić dyrektywę \$EEPLEAVE.

Normalnie plik .EEP (gdzie znajdują się te dane) jest tworzony przez BASCOM. Podanie właśnie tej dyrektywy jest sygnałem dla kompilatora by pozostawił istniejący plik .EEP. W innym wypadku plik ten mógłby być usunięty lub zmieniony.

\$EEPROM

Przeznaczenie:

Informuje kompilator by dane umieszczone po tej dyrektywie w instrukcjach DATA, były zapisane w pliku EEPROM.

Składnia:

\$EEPROM

Opis:

Procesory z serii AVR, mają wewnętrzną pamięć typu EEPROM, do odczytu i zapisu której służą instrukcje WRITEEEPROM oraz READEEPROM.

Dane można umieścić w pamięci EEPROM za pomocą ciągu instrukcji DATA. Aby kompilator wiedział o takim zamiarze, należy przed pierwszą instrukcją DATA użyć dyrektywy \$EEPROM. W ten sposób dane z linii DATA trafią do pliku z rozszerzeniem .EEP, który należy wykorzystać podczas programowania pamięci EEPROM procesora. Wbudowany program obsługi płytki prototypowej STK200/300 pozwala na stosowanie plików EEPROM.

Aby przywrócić standardową obsługę instrukcji DATA, należy użyć dyrektywy \$DATA. Spowoduje to umieszczenie danych z linii DATA w pamięci kodu.

Uwaga! Należy pamiętać, że instrukcje READ i RESTORE nie działają z danymi umieszczonymi w pamięci EEPROM za pomocą instrukcji DATA. Te instrukcje odczytują dane z linii DATA trafiających do pamięci kodu.

Dyrektywa \$EEPROM została dodana tylko w celu ułatwienia tworzenia plików binarnych, których przeznaczeniem jest pamięć EEPROM.

By zapisać i odczytać jakąś wartość z/do pamięci EEPROM, można także użyć zmiennych umieszczonych w ERAM:

```
Dim Store As ERAM Byte , B As Byte
B = 10                                'przypisujemy wartość zmiennej b
Store = B                             'trafia ona do pamięci EEPROM!
B = Store                             'a tu ją odczytujemy
```

Zobacz także: \$DATA , READEEPROM , WRITEEEPROM

Przykład:

```
Dim B As Byte

Restore Lbl 'wskaźnik na dane
Read B
Print B
Restore Lbl2
Read B
Print B

End

Lbl:
Data 100

$eeprom 'dane z poniższych linii DATA trafiają do pliku EPP
Data 200

$data 'przełącz na pamięć kodu
Lbl2:
Data 300
```

\$EEPROMHEX (Nowość w wersji 1.11.6.8)

Przeznaczenie:

Instruuje kompilator by generował plik w formacie HEX z zapisaną w nim zawartością wewnętrznej pamięci EEPROM.

Składnia:

\$EEPROMHEX

Opis:

Procesory z serii AVR, mają wewnętrzną pamięć typu EEPROM, do odczytu i zapisu które służą instrukcje **WRITEEEPROM** oraz **READEEPROM**.

Dane można umieścić w pamięci EEPROM za pomocą ciągu instrukcji **DATA**. Zobacz opis dyrektywy **\$EEPROM** by dowiedzieć się więcej. W ten sposób dane z linii DATA trafiają do pliku z rozszerzeniem .EEP, który należy wykorzystać podczas programowania pamięci EEPROM procesora.

Pliki .EEP mają postać binarną. Gdy używana jest płyta prototypowa STK500, wymaga ona by zawartość tej pamięci była umieszczona w pliku HEX. Umieszczenie w programie dyrektywy \$EEPROMHEX to umożliwia.

\$EXTERNAL

Przeznaczenie:

Informuje kompilator by dołączył procedury lub funkcje w języku assembler z biblioteki.

Składnia:

\$EXTERNAL *Procedura1* [, *Procedura2*]

gdzie:

<i>Procedura1</i>	Nazwy procedur umieszczonych w bibliotece i wykorzystanych
<i>Procedura2</i>	w programie.

Opis:

Fragmenty napisane w języku assembler można umieszczać w pliku biblioteki procedur. Stosując dyrektywę \$EXTERNAL, informujemy kompilator, które procedury lub funkcje muszą być dołączone do programu w języku BASCOM BASIC.

W przyszłości odnajdywanie procedur będzie automatyczne i stosowanie dyrektywy \$EXTERNAL nie będzie konieczne.

Zobacz także: **\$LIB**

Przykład:

```
'-----  
'  
'                                LIBDEMO.BAS  
'                                (c) 2000 MCS Electronics  
'By uruchomić program musisz umieścić plik mylib.lib w katalogu LIB  
'oraz skompilować go na format LBX  
'-----  
'-----  
'definicja użytej biblioteki  
$lib "mylib.lib"  
'możesz także użyć oryginalnej wersji w ASM:  
'$LIB "mylib.LIB"  
  
'trzeba także zdefiniować użyte procedury  
$external Test  
  
'poniższa linia jest wymagana ponieważ informuje kompilator o  
'parametrach umieszczanych na stosie  
Declare Sub Test(byval X As Byte , Y As Byte)  
  
'rezerwacja zmiennej  
Dim Z As Byte  
  
'wywołujemy procedurę z biblioteki  
Call Test(1 , Z)  
  
'z powinno zwrócić 2 w tym przykładzie  
End
```

\$INCLUDE

Przeznaczenie:

Wstawia zawartość pliku ASCII (np. fragment programu) na bieżącej pozycji tekstu źródłowego.

Składnia:

\$INCLUDE "nazwa_pliku"

gdzie:

nazwa_pliku Nazwa dołączanego pliku ze ścieżką dostępu. Plik musi zawierać tekst programu w języku BASCOM BASIC.

Opis:

Ta dyrektywa pozwala na dołączenie innego programu lub pliku z tekstem źródłowym najczęściej używanych fragmentów programu.

Pozwala to na stworzenie biblioteki sprawdzonych procedur, których użycie jest niezbędne w aktualnym programie głównym. Można w ten sposób uporządkować tworzony program lub ukryć niepotrzebne szczegóły. Jeśli zaś wymagane są zmiany w programie, można zmienić tylko treść dołączanego programu, a nie programu głównego.

Uwaga! Można dołączać tylko pliki z tekstem, tzw. pliki ASCII.

Przykład:

```
'-----  
'                               (c) 1997-2000 MCS Electronics  
'-----  
'  file: INCLUDE.BAS  
'  demo: $INCLUDE  
'-----  
  
Print "INCLUDE.BAS"  
' UWAGA! Plik 123.bas zawiera błąd  
$include "123.bas" 'dołącz plik który drukuje tekst HELLO  
Print "Powracamy do INCLUDE.BAS"  
  
End
```

By uruchomić program zmień nazwę pliku `a_rename.bas` na `a.bas`. Plik `a.bas` jest umieszczony w katalogu `SAMPLES`.

\$LCD

Przeznaczenie:

Informuje kompilator by generował kod dla wyświetlacza LCD podłączonego przez szynę danych.

Składnia:

\$LCD = [&H]adres

gdzie:

adres Adres, który należy ustawić na szynie adresowej by sterować linią E wyświetlacza. Adres zapisany szesnastkowo musi mieć przedrostek **&H**.

Opis:

W systemach z zewnętrzną pamięcią RAM, można umieścić wyświetlacz w jej przestrzeni adresowej. Wtedy za pomocą dekodera adresów sterujemy linią E i RS wyświetlacza.

Szyna danych wyświetlacza (linie *db0-db7*) musi być wtedy podłączona do szyny danych procesora D0-D7. Gdyby wyświetlacz pracował w trybie komunikacji 4 bitowej, podłączone są tylko linie D4-D7. Adres sterujący sygnałem RS wyświetlacza należy ustawić dyrektywą \$LCDRS.

Uwaga! Użycie tej dyrektywy, ma sens tylko przy zdefiniowaniu także adresu sterującego wyświetlaczem dyrektywą \$LCDRS.

Zobacz także: **\$LCDRS**

Przykład:

```
Rem Używamy płytki prototypowej STK200 więc adresy są następujące
$lcd = &HC000      'zapisywanie wartości pod ten adres ma powodować
                   'ustawienie linii E i RS wyświetlacza LCD.
$lcdrs = &H8000     'zapisywanie wartości pod ten adres ma powodować
                   'ustawienie linii E wyświetlacza LCD.

Cls
Lcd "Witaj świecie!"
```

\$LCDRS

Przeznaczenie:

Informuje kompilator by generował kod dla wyświetlacza LCD podłączonego przez szynę danych.

Składnia:

\$LCDRS = [&H]adres

gdzie:

adres Adres, który należy ustawić na szynie adresowej by sterować linią RS wyświetlacza. Adres zapisany szesnastkowo musi mieć przedrostek **&H**.

Opis:

W systemach z zewnętrzną pamięcią RAM, można umieścić wyświetlacz w jej przestrzeni adresowej. Wtedy za pomocą dekodera adresów sterujemy linią E i RS wyświetlacza.

Szyna danych wyświetlacza (linie *db0-db7*) musi być podłączona do szyny danych procesora D0-D7. Gdyby wyświetlacz pracował w trybie 4 bitowym, podłączone są tylko linie D4-D7. Adres sterujący sygnałem E wyświetlacza należy ustawić dyrektywą \$LCD.

Uwaga! Użycie tej dyrektywy, ma sens tylko przy zdefiniowaniu także adresu sterującego wyświetlaczem dyrektywą \$LCD.

Zobacz także: **\$LCD**

Przykład:

```
Rem Używamy płytki prototypowej STK200 więc adresy są następujące
$lcd = &HC000      'zapisywanie wartości pod ten adres ma powodować
                   'ustawienie linii E i RS wyświetlacza LCD.
$lcdrs = &H8000     'zapisywanie wartości pod ten adres ma powodować
                   'ustawienie linii E wyświetlacza LCD.

Cls
Lcd "Witaj świecie!"
```

\$LCDPUTCTRL

Przeznaczenie:

Ustala adres procedury użytkownika sterującej wyświetlaczem LCD.

Składnia:

\$LCDPUTCTRL = nazwa

gdzie:

nazwa

Adres procedury, przekazany przez etykietę, która przesyła bajt sterujący do wyświetlacza LCD.

Opis:

Gdyby wyświetlacz LCD był dołączony w niestandardowy sposób lub jego sterowanie było inne niż w standardzie Hitachi, należy zapewnić jego obsługę za pomocą procedur użytkownika. Nazwę procedury wysyłającą bajt sterujący, należy umieścić w dyrektywie \$LCDPUTCTRL.

Po umieszczeniu tej dyrektywy w programie, instrukcja **LCD** przy sterowaniu wyświetlacza będzie korzystać z procedury użytkownika.

Procedura ta nie powinna niszczyć żadnego z rejestrów. Przesyłany rozkaz jest umieszczony w rejestrze R24.

Uwaga! Użycie tej dyrektywy, ma sens tylko przy zdefiniowaniu także adresu procedury wysyłającej dane do wyświetlacza, za pomocą dyrektywy \$LCDPUTDATA.

Zobacz także: **\$LCDPUTDATA**

Przykład:

```
'definicja rejestrów
$regfile = "8535def.dat"

'częstotliwość kwarcu
$crystal = 4000000

'definicje zmiennych
Dim S As String * 10
Dim W As Long

'ustalamy adresy procedur obsługi wyświetlacza
$lcdputdata = Myoutput
$lcdputctrl = MyoutputCtrl
'pętla bez końca
Do
    Lcd "test"
Loop

End

'Procedura obsługi wyświetlacza
'by nie dopuścić do modyfikacji rejestrów używamy instrukcji
'Pushall i Popall zapamiętujących i odtwarzających rejestry
'$LCDPUTDATA wymaga by dane przesyłane pobrać z rejestru R24

Myoutput:
    Pushall                'zapamiętaj rejestry
    'kod umieść tutaj
    Popall                 'odtwórz rejestry
    Return

MyoutputCtrl:
    Pushall                'zapamiętaj rejestry
    'kod umieść tutaj
    Popall                 'odtwórz rejestry
    Return
```

\$LCDPUTDATA

Przeznaczenie:

Ustala adres procedury użytkownika wysyłającej dane do wyświetlacza LCD.

Składnia:

\$LCDPUTDATA = *nazwa*

gdzie:

<i>nazwa</i>	Adres procedury, przekazany poprzez etykietę, która przesyła bajt danych do wyświetlacza LCD.
--------------	---

Opis:

Gdyby wyświetlacz LCD był dołączony w niestandardowy sposób lub jego sterowanie było inne niż w standardzie Hitachi, należy zapewnić jego obsługę za pomocą procedur użytkownika. Nazwę procedury wysyłającej dane, należy umieścić w dyrektywie \$LCDPUTDATA.

Po umieszczeniu tej dyrektywy w programie, instrukcja **LCD** przy wysyłaniu danych do wyświetlacza będzie korzystać z procedury użytkownika.

Procedura ta nie powinna niszczyć żadnego z rejestrów. Dana przesyłana jest umieszczona w rejestrze R24.

Uwaga! Użycie tej dyrektywy, ma sens tylko przy zdefiniowaniu także adresu procedury wysyłającej rozkazy sterujące dla wyświetlacza, za pomocą dyrektywy \$LCDPUTCTRL.

Zobacz także: \$LCDPUTCTRL

Przykład:

```
'definicja rejestrów
$regfile = "8535def.dat"

'częstotliwość kwarcu
$crystal = 4000000

'definicje zmiennych
Dim S As String * 10
Dim W As Long

'ustalamy adresy procedur obsługi wyświetlacza
$lcdputdata = Myoutput
$lcdputctrl = MyoutputCtrl
'pętla bez końca
Do
    Lcd "1,2,3 Test"
Loop

End

'Procedura obsługi wyświetlacza
'by nie dopuścić do modyfikacji rejestrów używamy instrukcji
'Pushall i Popall zapamiętujących i odtwarzających rejestry
'$LCDPUTDATA wymaga by dane przesyłane pobrać z rejestru R24

Myoutput:
    Pushall                'zapamiętaj rejestry
    'kod umieść tutaj
    Popall                 'odtwórz rejestry
Return
```

```

MyoutputCtrl:
    Pushall                'zapamiętaj rejestry
    'kod umieść tutaj
    Popall                 'odtwórz rejestry
    Return

```

\$LCDVFO (Nowość w wersji 1.11.6.9)

Przeznaczenie:

Instruuje kompilator by generował bardzo krótki impuls na linii E dla wyświetlaczy lampowych typu VFO.

Składnia:

\$LCDVFO

Opis:

Wyświetlacze VFO wymagają bardzo krótkiego impulsu na linii E, dlatego wprowadzono tą dyrektywę, by umożliwić także stosowanie tych wyświetlaczy.

Uwaga! Aby używać takiego wyświetlacza, należy upewnić się czy jest on kompatybilny z wyświetlaczami LCD pracującymi w standardzie Hitachi (sterownik HD44780).

\$LIB

Przeznaczenie:

Informuje kompilator o wykorzystanych bibliotekach.

Składnia:

\$LIB "biblioteka1" [, "biblioteka2"]

gdzie:

<i>biblioteka1</i>	Nazwy bibliotek w których należy szukać procedur
<i>biblioteka2</i>	zdefiniowanych przez dyrektywę \$EXTERNAL .

Opis:

Po dyrektywie \$LIB należy podać nazwy bibliotek, w których przechowywane są procedury lub funkcje, używane przez program. Można określić kilka nazw rozdzielając je przecinkami. Kompilator w trakcie dołączania procedur określonych w dyrektywie \$EXTERNAL przeszukuje kolejno biblioteki, w porządku jakim są one podane w dyrektywie \$LIB.

Podstawowa biblioteka **MCS.LBX** jest dołączana na końcu i przeszukiwana jako ostatnia. Nie ma zatem potrzeby umieszczania jej jawnie w dyrektywie \$LIB.

Ponieważ biblioteka **MCS.LBX** jest przeszukiwana jako ostatnia, można pokryć (zastąpić) pewne jej procedury tymi z własnej biblioteki. Jest to dobra metoda na rozszerzenie właściwości **MCS.LBX**.

Można także dostosować procedury z biblioteki **MCS.LIB**, modyfikując jej oryginalną treść i zapisując te zmiany jako inny plik biblioteczny. Kiedy MCS Electronics opracuje nową wersję biblioteki standardowej, zmiany jakie wprowadził użytkownik, zostaną uwzględnione poprzez opisany mechanizm pierwszeństwa.

Tworzenie własnych bibliotek.

Biblioteka jest to zwykły plik ASCII. Można go stworzyć za pomocą edytora środowiska BASCOM, czy innego preferowanego przez użytkownika edytora zapisującego ten typ plików.

Plik musi rozpoczynać się nagłówkiem (Aktualnie jest on pomijany, lecz w przyszłości będzie wykorzystywany) zawierającym następujące pola:

copyright = nazwisko i imię twórcy, informacja o prawach autorskich
www = opcjonalnie adres internetowy gdzie można uzyskać aktualną wersję
email = e-mail twórcy biblioteki
comment = użyte biblioteki kompilatora AVR
libversion = wersja pliku w formacie *wersja.podwersja*, np.:1.00, 2.04
date = data ostatniej modyfikacji pliku
statement = informacje o sposobie użycia biblioteki

Poszczególne procedury muszą rozpoczynać się nazwą zawartą w nawiasach kwadratowych **[nazwa]** i zakończone przez **[END]**.

Poniższy przykład procedury pochodzi z oryginalnej biblioteki `MCS.LIB`.

```

[_ClockDiv]

; MEGA chips only
; _temp1 holds the division in the range from 0-129
; 0 will set the division to 1

_ClockDiv:
Cpi _temp1,0          ; is it zero?
Breq _ClockDivX       ; yes so turn off the division
Subi _temp1,2         ; subtract 2
Com _temp1            ; complement
Clr _temp2
Out XDIV,_temp2       ; enable write by writing zeros
_ClockDivX:
Out XDIV,_temp1       ; write new division
Ret                  ; return

[END]

```

Po umieszczeniu pliku własnej biblioteki w katalogu LIB, należy taką bibliotekę skompilować używając narzędzia **LIB Manager** z menu **Tools**. Można także używać biblioteki w postaci nieskompilowanej, wtedy w dyrektywie podajemy nazwę pliku o rozszerzeniu LIB.

Kilka słów o asemblerze.

Jeśli w kodzie programu w języku asembler stosowane są odwołania do stałych, które są definiowane w programie w BASCOM Basic-u, wtedy przed rozkazem należy umieścić znak gwiazdki (*). Ilustruje to prosty przykład:

```

`program w BASCOM Basic-u
Const myconst = 7

`kod biblioteki w asm
* Sbi PortB , myconst

```

Linie w których występuje znak gwiazdki, są kompilowane dopiero podczas kompilacji programu w BASCOM Basic-u. Nie są one zmieniane w kod obiektowy umieszczony w skompilowanych bibliotekach LBX.

Gdy w kodzie są używane stałe należy używać tylko dopuszczalnych form ich zapisu:

```

Ldi r24,12
Ldi r24, 1+1
Ldi r24, &B001
Ldi r24, 0b001
Ldi r24, &HFF
Ldi r24, $FF
Ldi r24, 0xFF

```

Inne formy składni NIE są obsługiwane.

Zobacz także: **\$EXTERNAL**

Przykład:

```
'-----  
'  
'                                LIBDEMO.BAS  
'                                (c) 2000 MCS Electronics  
'By uruchomić program musisz umieścić plik mylib.lib w katalogu LIB  
'oraz skompilować go na format LBX  
'-----  
'  
'definicja użytej biblioteki  
$lib "mylib.lib"  
'używamy tutaj oryginalnej wersji w ASM, nie pliku skompilowanego LBX  
  
'trzeba także zdefiniować użyte procedury  
$external Test  
  
'poniższa linia jest wymagana ponieważ informuje kompilator o  
'parametrach umieszczanych na stosie  
Declare Sub Test(byval X As Byte , Y As Byte)  
  
'rezerwacja zmiennej  
Dim Z As Byte  
  
'wywołujemy procedurę z biblioteki  
Call Test(1 , Z)  
  
'z powinno zwrócić 2 w tym przykładzie  
End
```

\$MAP

Przeznaczenie:

Informuje kompilator by wygenerował adresy linii programu w raporcie kompilacji.

Składnia:

\$MAP

Opis:

Dyrektywa \$MAP powoduje dołączenie adresów poszczególnych linii w pamięci kodu, do generowanego raportu kompilacji. Informacja ta może być użyta podczas usuwania błędów w programie przez inne narzędzia.

Przykład:

```
$map  
  
Print "Ala ma kota"  
Print "Kot ma Alę"
```

\$NOINIT

Przeznaczenie:

Składnia:

\$NOINIT

Opis:

Dyrektywa \$NOINIT powinna być używana razem z dyrektywą \$ROMSTART. Można w ten sposób stworzyć program typu boot-loader.

Zobacz także: [\\$ROMSTART](#)

Asembler:

W prostych programach jest generowany następujący kod – dla AT90s2313:

```
Rjmp _BASICSTART  
Reti  
Reti  
Reti  
Reti  
Reti  
Reti  
Reti  
Reti  
Reti  
  
_BASICSTART:  
;wyłączenie licznika Watchdog  
Ldi _temp1,$1F  
Out WDTCSR,_temp1  
Ldi _temp1,$17  
Out WDTCSR,_temp1  
;Ldi R24,$DF ;stos sprzętowy  
Out SPL,R24  
ldi YL,$C8 ;stos programowy  
ldi ZL,$98  
Mov _SPL,ZL ;wskaźnik na dane ramki  
  
Clr YH  
Mov _SPH,YH  
Ldi ZL,$7E ;liczba bajtów  
Ldi ZH,$00  
Ldi XL,$60 ;początek RAM  
Ldi XH,$00  
Clr R24  
_ClearRAM:  
St X+,R24 ;zerowanie  
Sbiw ZL,1  
Brne _ClearRAM  
Clr R6 ;zerowanie wewnętrznych flag języka  
  
BASCOM  
;##### Dim X As Byte  
;##### X = 1  
Ldi _temp1,$01  
Sts $0060,R24 ;wpisz wartość do pamięci
```

Na początku ustawiane są wektory przerwań, układ Watchdog jest wyłączany, ustawiany jest stos oraz kasowana jest pamięć RAM i znaczniki w rejestrze R6.

Następnie jest wykonywany program główny:

$$X = 1.$$

Teraz ten sam program gdy umieszczono dyrektywę \$NOINIT:

```
; wyłączenie licznika Watchdog
Ldi _temp1,$1F
Out WDTCR,_temp1
Ldi _temp1,$17
Out WDTCR,_temp1
;inicjalizacja wskaźnika stosu
Ldi R24,$DF ;stos sprzętowy
Out SPL,R24
Ldi YL,$C8 ;stos programowy
Ldi ZL,$98
Mov _SPL,ZL ;wskaźnik na dane ramki

Clr YH
Mov _SPH,YH
Ldi ZL,$7E ;liczba bajtów
Ldi ZH,$00
Ldi XL,$60 ;początek RAM
Ldi XH,$00
Clr R24
_ClearRAM:
St X+,R24 ;zerowanie
Sbiw ZL,1
Brne _ClearRAM
Clr R6 ;zerowanie wewnętrznych flag języka
BASC0M
;##### Dim X As Byte
;##### X = 1
Ldi _temp1,$01
Sts $0060,R24 ;wpisz wartość do pamięci
```

Różnica polega na tym, że brak jest tu konfiguracji procedur obsługi przerwań.

Intencją wprowadzenia dyrektywy \$NOINIT jest możliwość napisania programu typu boot-loader w języku BASCOM BASIC. Ponieważ istota programów tego typu nie została jeszcze dokładnie przestudiowana, działanie dyrektywy może zostać zmienione w najbliższej przyszłości.

\$NORAMCLEAR

Przeznaczenie:

Instruuje kompilator by nie umieszczał w procedurze inicjalizacji fragmentu czyszczącego zawartość pamięci.

Składnia:

\$NORAMCLEAR

Opis:

Normalnie pamięć SRAM jest kasowana podczas inicjalizacji. Gdy zawartość pamięci nie powinna być kasowana (zapisywana zerami), można użyć tej dyrektywy.

Ponieważ podczas kasowania pamięci zawartość wszystkich zmiennych jest ustawiana jako 0 lub "" (dla zmiennych String), umieszczenie dyrektywy \$NORAMCLEAR spowoduje, że ich zawartość będzie nieznana. Prawdopodobnie będą one wypełnione wartością &H0FF, choć nie jest to pewne.

Zobacz także: **\$NOINIT**

\$REGFILE

Przeznaczenie:

Informuje kompilator, by użył podanego pliku definicji rejestrów, zamiast określonego w opcjach kompilatora.

Składnia:

\$REGFILE = "nazwa"

gdzie:

nazwa Nazwa pliku z definicją rejestrów.

Opis:

Pliki z definicjami rejestrów są umieszczone w katalogu programu BASCOM-AVR i mają rozszerzenie .DAT. Plik definicji zawiera informacje o nazwach rejestrów wewnętrznych, ich adresów w konkretnym typie kontrolera AVR oraz o adresach przerwań.

Aktualnie definicje rejestrów są zapisywane w pliku <nazwa_projektu>.CFG. Dyrektywę pozostawiono ze względu na kompatybilność z kompilatorem BASCOM-8051.

Uwaga! Dyrektywa \$REGFILE musi być pierwszą dyrektywą umieszczoną w programie.

Przykład:

```
$regfile = "8515DEF.DAT"
```

\$ROMSTART

Przeznaczenie:

Instruuje kompilator by umieścił kod w pliku HEX od podanego adresu.

Składnia:

\$ROMSTART = *adres*

gdzie:

adres adres bazowy jaki ma przyjąć kompilator podczas generowania kodu..

Opis:

Standardowo adres ten jest ustawiany na 0, więc pierwsza instrukcja umieszczona będzie pod adresem 0. Zmiana tego adresu spowoduje, że kod umieszczony będzie od podanego adresu.

Uwaga! W pliku binarnym wygenerowany kod będzie umieszczony na początku, należy zatem pamiętać by zaprogramować pamięć Flash od ustalonego w dyrektywie adresu. Inaczej adresy bezwzględnych skoków będą przesuwane.

Dyrektywa \$ROMSTART może być używana do przemieszczenia kodu, w celu utworzenia programu typu boot-loader. Należy wtedy jeszcze dodać dyrektywę **\$NOINIT**.

Zobacz także: **\$NOINIT**

Przykład:

```
$noinit  
$romstart = &H4000
```

\$SERIALINPUT

Przeznaczenie:

Informuje kompilator by użył procedury użytkownika dla instrukcji INPUT.

Składnia:

\$SERIALINPUT = *nazwa*

gdzie:

<i>nazwa</i>	Adres procedury, przekazany przez etykietę, która ma być wywołana by odebrać znak.
--------------	--

Opis:

Za pomocą tej dyrektywy można zmienić standardową procedurę odczytywania znaków przez instrukcję **INPUT**, by korzystała z innego urządzenia niż domyślnie wbudowany sprzętowy UART.

Uwaga! Odebrany znak musi być umieszczony w rejestrze R24. Instrukcja **INPUT** jest wykonywana do czasu odebrania znaku <CR> (kod 13).

Normalnie instrukcje **INPUT** lub funkcja **INKEY()**, oczekują na transmisję z portu COM (sprzętowy UART). Jeśli w programie jako urządzenie wprowadzania danych używana jest klawiatura lub układ zdalnego sterowania, należy napisać własne procedury obsługi, które umieszczają odebrane znaki w rejestrze R24.

Zobacz także: \$SERIALOUTPUT

Przykład:

```
'-----  
'                               $myserialinput.bas  
'                               (c) 2000 MCS Electronics  
'demonstruje przekierowanie instrukcją $SERIALINPUT  
'-----  
'użyty procesor  
$regfile = "8535def.dat"  
  
'częstotliwość kwarcu  
$crystal = 4000000  
  
'definiujemy zmienne  
Dim S As String * 10  
Dim W As Long  
  
'informujemy kompilator która procedura musi być wywołana  
'by odebrać znak przez łącze szeregowie  
$serialinput = Myinput  
  
'tworzymy nieskończoną pętlę  
Do  
    'pytamy o nazwisko  
    Input "Imię " , S  
    Print S  
    'zmienna Err przyjmuje kod gdy przekroczono czas oczekiwania  
    Print "Err = " ; Err  
Loop  
  
End  
  
'własna procedura odbioru znaku
```

'Żeby nie niszczyć zawartości z żadnego rejestru oraz używania
'asemblera, użyjemy instrukcji Pushall i PopAll by zapamiętać i
'odtworzyć wszystkie rejestry. Możemy zatem używać także instrukcji
'BASCOM BASIC
'\$SERIALINPUT wymaga by odebrany znak był zwracany w rejestrze R24

```
Myinput:
    Pushall                'zapamiętujemy rejestry
    W = 0                  'zerujemy licznik
Myinput1:
    Incr W                 'zwiększamy licznik o jeden
    Sbis USR, 7            'sprawdzamy czy odebrał znak
    Rjmp Myinput2          'jeśli nie no czekamy

    Popall                 'coś otrzymaliśmy więc
    Err = 0                'kasujemy zmienną Err
    In _temp1, UDR         'odczytujemy odebrany znak
    Return                 'i koniec
Myinput2:
    If W > 1000000 Then    'przy 4 MHz będzie to około 10 sekund
        Rjmp Myinput_exit 'za długo czekaliśmy!
    Else
        Goto Myinput1     'spróbujemy ponownie, a nuż...
    End If
Myinput_exit:
    Popall                 'odtwarzamy rejestry
    Err = 1                'ustawiamy, że był błąd
    Ldi R24, 13            'wpisujemy CR by zakończyć działanie INPUT
    Return
```

\$SERIALINPUT1 (Nowość w wersji 1.11.6.8)

Przeznaczenie:

Informuje kompilator by użył procedury użytkownika dla instrukcji INPUT korzystającej z drugiego sprzętowego układu UART.

Składnia:

\$SERIALINPUT1 = nazwa

gdzie:

nazwa Adres procedury (etykieta!), która ma być wywołana by odebrać znak.

Opis:

Za pomocą tej dyrektywy można zmienić standardową procedurę odczytywania znaków przez instrukcję **INPUT**, by korzystała z innego urządzenia niż domyślnie wbudowany sprzętowy UART.

Uwaga! Odebrany znak musi być umieszczony w rejestrze R24. Instrukcja **INPUT** jest wykonywana do czasu odebrania znaku <CR> (kod 13).

Normalnie instrukcje **INPUT #** lub funkcja **INKEY(#)**, oczekują na transmisję z portu COM2 (drugi sprzętowy UART, otwarty wcześniej instrukcją **OPEN** *przyp. tłumacza*). Jeśli w programie jako urządzenie wprowadzania danych używana jest klawiatura lub układ zdalnego sterowania, należy napisać własne procedury obsługi, które umieszczają odebrane znaki w rejestrze R24.

Zobacz także: **\$SERIALOUTPUT1**

Przykład:

'-----

```

'                                     $myserialinput.bas
'                                     (c) 2000 MCS Electronics
'demonstruje przekierowanie instrukcją $SERIALINPUT1
'-----
'użyty procesor
$regfile = "8535def.dat"

'częstotliwość kwarcu
$crystal = 4000000

'definiujemy zmienne
Dim S As String * 10
Dim W As Long

'informujemy kompilator która procedura musi być wywołana
'by odebrać znak przez łącze szeregowo
$serialinput = Myinput

'tworzymy nieskończoną pętlę
Open "COM2:9600" For Input As #1
Do
    'pytamy o nazwisko
    Input #1, "Imię ", S
    Print S
    'zmienna Err przyjmuje kod gdy przekroczono czas oczekiwania
    Print "Err = " ; Err
Loop
Close #1

End

'własna procedura odbioru znaku

'Żeby nie niszczyć zawartości z żadnego rejestru oraz używania
'asemblera, użyjemy instrukcji Pushall i PopAll by zapamiętać i
'odtworzyć wszystkie rejestry. Możemy zatem używać także instrukcji
'BASCOM BASIC
'$SERIALINPUT wymaga by odebrany znak był zwracany w rejestrze R24

Myinput:
    Pushall                                'zapamiętujemy rejestry
    W = 0                                  'zerujemy licznik
Myinput1:
    Incr W                                  'zwiększamy licznik o jeden
    Sbis USR1, 7                            'sprawdzamy czy odebrał znak
    Rjmp Myinput2                            'jeśli nie no czekamy

    Popall                                  'coś otrzymaliśmy więc
    Err = 0                                  'kasujemy zmienną Err
    In _temp1, UDR1                          'odczytujemy odebrany znak
    Return                                    'i koniec
Myinput2:
    If W > 1000000 Then                      'przy 4 MHz będzie to około 10 sekund
        Rjmp Myinput_exit                    'za długo czekaliśmy!
    Else
        Goto Myinput1                        'spróbujemy ponownie, a nuż...
    End If
Myinput_exit:
    Popall                                  'odtworzamy rejestry
    Err = 1                                  'ustawiamy, że był błąd
    Ldi R24, 13                              'wpisujemy CR by zakończyć działanie INPUT

```

Return

\$SERIALINPUT2LCD

Przeznaczenie:

Dyrektywa włącza przekierowanie odebranych znaków z urządzenia UART na ekran LCD.

Składnia:

\$SERIALINPUT2LCD

Opis:

Umieszczenie tej dyrektywy, spowoduje wyświetlanie odebranych znaków na ekranie LCD – tzw. lokalne echo. Dane są odbierane przez instrukcje **INPUT**.

Można stworzyć własne procedury wejścia/wyjścia i za pomocą dyrektyw **\$SERIALINPUT** oraz **\$SERIALOUTPUT** oraz przekierować działanie instrukcji **INPUT** czy **PRINT**. Dyrektywa **\$SERIALINPUT2LCD** może być bardzo pomocna przy testowaniu tych procedur.

Zobacz także: **\$SERIALINPUT** , **\$SERIALOUTPUT**

Przykład:

```
$serialinput2lcd
Dim v As Byte

Cls
Input "Number ", v      'to zostanie także wyświetlone na ekranie LCD
```

\$SERIALOUTPUT

Przeznaczenie:

Informuje kompilator by użył procedury użytkownika dla instrukcji **PRINT**.

Składnia:

\$SERIALOUTPUT = nazwa

gdzie:

nazwa Nazwa procedury (etykieta!), która ma być wywołana by wysłać znak.

Opis:

Za pomocą tej dyrektywy można zmienić standardową procedurę przesyłania znaków przez instrukcję **PRINT**, by korzystała z innego urządzenia niż domyślni przez wbudowany sprzętowy UART. Procedura obsługi urządzenia musi być napisana przez użytkownika.

Uwaga! Wysłany znak jest umieszczony w rejestrze R24.

Zobacz także: **\$SERIALINPUT** , **\$SERIALINPUT2LCD**

Przykład:

```
$serialoutput = MyOutput
'twój program będzie tutaj
Do
  Print "Hello"
Loop
End
```

```

Myoutput:
'umieść treść procedury tutaj
'dane są wpisywane do rejestru R24
'którego zawartość wpiszemy do PORTB
!Out Portb , r24
Ret

```

\$SERIALOUTPUT1 (Nowość w wersji 1.11.6.8)

Przeznaczenie:

Informuje kompilator by użył procedury użytkownika dla instrukcji PRINT korzystającej z drugiego sprzętowego układu UART.

Składnia:

\$SERIALOUTPUT1 = nazwa

gdzie:

nazwa Nazwa procedury (etykieta!), która ma być wywołana by wysłać znak.

Opis:

Za pomocą tej dyrektywy można zmienić standardową procedurę przesyłania znaków przez instrukcję **PRINT**, by korzystała z innego urządzenia niż domyślnie przez wbudowany sprzętowy UART. Procedura obsługi urządzenia musi być napisana przez użytkownika.

Uwaga! Wysyłany znak jest umieszczony w rejestrze R24.

Zobacz także: **\$SERIALINPUT**

\$SIM

Przeznaczenie:

Instruuje kompilator by nie generował pętli opóźniających dla instrukcji WAIT czy WAITMS. Spowoduje to zwiększenie szybkości symulacji.

Składnia:

\$SIM

Opis:

Symulacja instrukcji **WAIT** może trwać dość długo, zwłaszcza gdy otwarte jest okno podglądu zawartości pamięci. Umieszczenie tej dyrektywy spowoduje, że kompilator nie wygeneruje pętli opóźniających dla instrukcji WAIT czy **WAITMS**. Przyspieszy to oczywiście symulację programu.

Uwaga! Gdy program będzie gotowy, należy usunąć (lub umieścić w linii komentarza) dyrektywę \$SIM, inaczej instrukcje WAIT i WAITMS nie będą działać poprawnie.

Jeśli wystąpi próba zaprogramowania pamięci Flash, programem skompilowanym z dyrektywą \$SIM, środowisko BASCOM pokaże stosowne ostrzeżenie.

Przykład:

```

$sim

Do
  Wait 1
Loop

```


\$TINY

Przeznaczenie:

Instruuje kompilator by pomiął fragment ustawiający stos w procedurze inicjalizacji.

Składnia:

\$TINY

Opis:

Układ ATTiny11 (przykładowo) jest dość ciekawym układem. Niestety nie posiada on pamięci SRAM. BASCOM używa tej pamięci na potrzeby programowego i sprzętowego stosu.

Niektóre instrukcje języka BASCOM BASIC będą działać, choć znaczna ich ilość niestety nie. BASCOM BASIC posiada kilka instrukcji i funkcji, które z powodzeniem będą działać na procesorach nie posiadających SRAM. Istnieje specjalna biblioteka `tiny.lib`, której procedury używają mało rejestrów oraz obsługują trzy poziomowy stos, jaki posiadają kontrolery Tiny AVR.

Uwaga! Generowany kod nie jest jeszcze zoptymalizowany. Dyrektywa \$TINY to dopiero pierwszy krok przy wprowadzaniu obsługi tych procesorów!. Dlatego nie jest udzielana jakakolwiek pomoc techniczna, dopóki biblioteka `tiny.lib` nie będzie ukończona.

Gdyby użytkownik chciał pisać program w języku assemblera, może użyć języka BASCOM BASIC stosując dyrektywę \$TINY. Oczywiście kod assemblerowy najlepiej umieścić w bloku `$ASM..$END ASM`.

Przykład:

\$tiny

```
Dim X As Iram Byte, y As Iram Byte
X = 1 : Y = 2 : X = x + y
```

\$WAITSTATE

Przeznaczenie:

Dyrektywa kompilatora aktywująca zewnętrzną pamięć RAM i powodująca wstawienie taktu opóźniającego by wydłużyć sygnał ALE.

Składnia:

\$WAITSTATE

Opis:

Dyrektywa \$WAITSTATE może być użyta do lokalnej zmiany ustawień w menu [Compiler | Chip Options](#).

Przykład:

\$waitstate

\$XRAMSIZE

Przeznaczenie:

Informuje kompilator o wielkości zewnętrznej pamięci RAM.

Składnia:

\$XRAMSIZE = [&H]wielkość

gdzie:

wielkość Całkowity rozmiar zewnętrznej pamięci RAM. Wielkość musi być określona jako stała.

Opis:

Rozmiar tej pamięci można określić w menu [Options | Compiler Chip](#). Wartość przypisana dyrektywie zmienia tą wartość.

Ważne jest by dyrektywa \$XRAMSTART poprzedzała dyrektywę \$XRAMSIZE, jak w przykładzie.

Zobacz także: [\\$XRAMSTART](#)

Przykład:

```
$xramstart = &H300
$xramsize = &H1000
Dim x As XRAM Byte 'zmienna x zostanie umieszczona w XRAM
```

[\\$XRAMSTART](#)

Przeznaczenie:

Informuje kompilator od jakiego adresu zaczyna się zewnętrzna pamięć RAM.

Składnia:

\$XRAMSTART = [&H]*wielkość*

gdzie:

wielkość Adres początkowy zewnętrznej pamięci RAM. Wielkość musi być określona jako stała.

Opis:

Standardowo przestrzeń adresowa zewnętrznej pamięci RAM (tzw. XRAM), zaczyna się tuż za pamięcią wewnętrzną.

Za pomocą tej dyrektywy, można także zabezpieczyć określony obszar pamięci XRAM. Wtedy podajemy adres powiększony o rozmiar zabezpieczanej pamięci. Dla przykładu: określenie adresu &H400 spowoduje że pierwsza zmienna umieszczona w pamięci XRAM będzie umieszczona od adresu &H400, a nie od adresu &H260. Obszar od &H260 do &H3FF jest zabezpieczony przed umieszczaniem tam zmiennych przez kompilator.

Ważne jest by dyrektywa \$XRAMSTART poprzedzała dyrektywę \$XRAMSIZE, jak w przykładzie.

Zobacz także: [\\$XRAMSIZE](#)

Przykład:

```
$xramstart = &H300
$xramsize = &H1000
Dim x As XRAM Byte. 'jako obszar docelowy zmiennej określamy XRAM
```

[1WIRECOUNT\(\)](#)

Przeznaczenie:

Zwraca liczbę urządzeń podłączonych do magistrali 1Wire.

Składnia:

zmienna = **1WIRECOUNT()**

zmienna = 1WIRECOUNT(port , pin)

gdzie:

zmienna	dowolna zmienna typu Word lub Integer, do której wpisana będzie liczba urządzeń
port	nazwa rejestru. np.: PINB, PIND,
pin	numer bitu portu z zakresu 0-7.

Opis:

Można użyć tej funkcji by dowiedzieć się ile razy należy wywołać funkcję **1WSEARCHNEXT()** by odczytać wszystkie numery identyfikacyjne dołączonych układów.

Funkcja korzysta z 4 bajtów pamięci SRAM. Są to:

__1w_bitstorage	W którym zapisano bity:
	lastdeviceflag bit 0
	id_bit bit 1
	mp_id_bit bit 2
	search_dir bit 3
__1wid_bit_number	
__1wlast_zero	
__1wlast_discrepancy	

Asembler:

Wywołuje procedurę z biblioteki `mcs.lib`:

```
_1wire_Count : (wywołuje _1WIRE, _1WIRE_SEARCH_FIRST, _1WIRE_SEARCH_NEXT)
```

Przekazywane parametry:

R24 : numer końcówki,
R30 : port,
Y+0, Y+1 : 2 bajty na stosie,
X : wskaźnik do przestrzeni w ramce,

Zwraca:

Y+0, Y+1 : adres zmiennej do której wpisany jest wynik działania funkcji.

Zobacz także: **1WRITE** , **1WRESET** , **1WREAD** , **1WSEARCHFIRST** , **1WSEARCHNEXT**

Przykład:

```
'-----  
'                                     '1wireSearch.bas  
'                                     (c) 2000 MCS Electronics  
'                                     wersja b, 27 dec 2000  
'-----  
  
Config 1wire = Portb.0                'używamy tej końcówki  
'Na płytce STK200 „jumper” B.0 musi być zwarty  
  
'Następujące bajty są używane przez procedurę przeszukiwania  
magistrali  
'__1w_bitstorage , przechowuje bity:  
'    lastdeviceflag bit 0  
'    id_bit          bit 1  
'    cmp_id_bit      bit 2  
'    search_dir      bit 3  
'__1wid_bit_number, Byte  
'__1wlast_zero,   Byte  
'__1wlast_discrepancy , Byte  
'__1wire_data , string * 7 (8 bajtów)  
  
'[definiowanie zmiennych]
```

```

'potrzebne jest 8 bajtów by przechowywać numery ID
Dim Reg_no(8) As Byte

'Potrzebny będzie licznik pętli i zmienna Word/Integer jako licznik
układów
Dim I As Byte , W As Word

'Sprawdzamy pierwsze z urządzeń
Reg_no(1) = lwsearchfirst()

For I = 1 To 8                                'drukujemy numer ID
    Print Hex(reg_no(i));
Next
Print

Do
    'Teraz szukamy następnych urządzeń
    Reg_no(1) = lwsearchnext()
    For I = 1 To 8
        Print Hex(reg_no(i));
    Next
    Print
Loop Until Err = 1

'Gdy zmienna ERR będzie równa 1, znaczy to że nie znaleziono
następnych urządzeń
'można także policzyć ile ich jest
W = lwirecount()
'Ważne jest by zmienna była typu Integer/Word!
'Rezultat działania tej funkcji jest typu Integer/Word
Print W

'Jako bonus następne procedury:
'Najpierw wpisujemy do tablicy jakiś istniejący numer ID
Reg_no(1) = lwsearchfirst()
'usuń komentarz z następnej linii, by zmienić bajt dla sprawdzenia
flagi ERR
'Reg_no(1) = 2
'teraz sprawdzamy czy numer się zgadza
lwverify Reg_no(1)
Print Err
'err =1 gdy nie ma takiego numeru
'Można także podać numer portu i końcówki: lwverify reg_no(1),pinb,1
'To samo można zrobić z innymi instrukcjami dla magistrali 1Wire:
'W = lwirecount(pinb , 1)                'podaj ile jest dołączonych do
PINB.1
End

```

1WRESET

Przeznaczenie:

Instrukcja ta przywraca poprawny stan magistrali 1Wire, oraz wysyła rozkaz reset do urządzeń podłączonych do magistrali.

Składnia:

1WRESET

1WRESET *port , pin*

gdzie:

<i>port</i>	nazwa rejestru. np.: PINB, PIND,
<i>pin</i>	numer bitu portu z zakresu 0-7.

Opis:

Instrukcja 1WRESET zeruje stan szyny 1Wire. Jeśli operacja się nie powiedzie, zmienna ERR zwraca 1 jako kod błędu.

Nowością jest możliwość pracy z kilkoma urządzeniami 1Wire dołączonymi do różnych końcówek portu mikrokontrolera. By używać tego rozszerzonego trybu, w każdej operacji na szynie 1Wire musi być określony port i końcówka portu (pin).

Instrukcje 1WRESET, 1WWRITE i 1WREAD mogą nadal pracować razem jeśli są używane według starej składni. Wtedy końcówkę portu będącą szyną 1Wire określa się w opcjach kompilatora lub instrukcją **CONFIG 1WIRE**.

Zobacz także: **1WREAD**, **1WWRITE**

Przykład:

```
'-----
'          1WIRE.BAS (c) 2000 MCS Electronics
' Demonstracja lwreset, lwwrite i lwread()
' Wymagany jest rezystor podciągający 4k7 na PORTB.1
' Pastylkę DS2401 podłączono do końcówki PORTB.1
'-----
'Gdy w programie używane są tylko zmienne typu Byte, można skorzystać
z
'poniższej biblioteki by zmniejszyć rozmiar kodu
$lib "mcsbyte.lib"

Config lwire = Portb.0          'użyjemy tej końcówki
'Na płytce STK200 musi być założona zworka B.0
Dim Ar(8) As Byte , A As Byte , I As Byte

Do
  Wait 1
  lwreset                      'wyślij rozkaz Reset
  Print Err                    'jeśli się nie powiodło Err=1
  lwwrite &H33                  'odczytaj ROM
  For I = 1 To 8
    Ar(i) = lwread()            'wpisz nr ID do tablicy
  Next

  'Możesz także odczytać wszystkie 8 bajtów na raz, usuń w tym celu
  komentarz
  'z następnej linii i usuń całą pętlę FOR..NEXT znajdującą się powyżej
  'Ar(1) = lwread(8)            'czytaj 8 bajtów

  For I = 1 To 8
    Print Hex(ar(i));           'wypiszemy co odebrał
  Next
  Print                        'i CR
Loop

'UWAGA! JEŚLI SKOMPILUJESZ PROGRAM DALSZE INSTRUKCJE NIE ZOSTANĄ
WYKONANE,
'GDYŻ POWYŻSZA PĘTLA DO..LOOP NIE ZOSTANIE NIGDY PRZERWANA!!!
```

```

'Nowością jest używanie kilku magistral 1Wire podłączonych do różnych
końcówek portów
'Wtedy instrukcje wyglądają tak:
For I = 1 To 8
    Ar(i) = 0                                'wyczyść tablicę by zobaczyć jak
    działa
Next

lwreset Pinb , 2                            'używamy tego portu i tej końcówki
jako drugiej magistrali
lwwrite &H33 , 1 , Pinb , 2                  'Uwaga! Musi zostać podana liczba
odczytywanych bajtów!
'lwwrite Ar(1) , 5 , Pinb , 2

'Odczyt także wygląda nieco inaczej
Ar(1) = lwread(8 , Pinb , 2)                 'odczytaj 8 bajtów z PortB na
końcówce 2 (PB2)

For I = 1 To 8
    Print Hex(ar(i));
Next

'Możesz numer bitu umieścić w pętli, by odczytać dane z kilku
magistral sekwencyjnie!
For I = 0 To 3                                'końcówki od 0 (PB0) do 3 (PB3)
    lwreset Pinb , I
    lwwrite &H33 , 1 , Pinb , I
    Ar(1) = lwread(8 , Pinb , I)
    For A = 1 To 8
        Print Hex(ar(a));
    Next
    Print
Next

End

```

1WREAD()

Przeznaczenie:

Instrukcja odczytuje dane z urządzeń podłączonych do magistrali 1Wire.

Składnia:

zmienna = 1WREAD([*il_bajtów*])

zmienna = 1WREAD(*il_bajtów* , *port* , *pin*)

gdzie:

<i>zmienna</i>	zmienna do której odczytana zostanie dana,
<i>il_bajtów</i>	opcjonalnie; określa ilość odbieranych bajtów,
<i>port</i>	nazwa rejestru. np.: PINB, PIND,
<i>pin</i>	numer bitu portu z zakresu 0-7.

Opis:

Instrukcja odczytuje dane z urządzenia podłączonego do magistrali 1Wire. Gdy *il_bajtów* nie jest podana jako parametr, instrukcja odczytuje jeden bajt danych.

Odczytywanie kilku bajtów danych, ma sens tylko gdy *zmienna* jest zmienną tablicową. Wtedy kolejne dane trafiają do kolejnych komórek tablicy.

Nowością jest możliwość pracy z kilkoma urządzeniami 1Wire dołączonymi do różnych

końcówek portu mikrokontrolera. By używać tego rozszerzonego trybu, w każdej operacji na szynie 1Wire musi być określona ilość odczytywanych bajtów, port oraz końcówka portu (pin).

Instrukcje 1WRESET, 1WWRITE i 1WREAD mogą nadal pracować razem jeśli są używane według starej składni. Wtedy końcówkę portu będącą szyną 1Wire określa się w opcjach kompilatora lub instrukcją **CONFIG 1WIRE**.

Zobacz także: **1WWRITE** , **1WRESET**

Przykład:

```
'-----
'               1WIRE.BAS (c) 2000 MCS Electronics
' Demonstracja lwreset, lwwrite i lwread()
' Wymagany jest rezystor podciągający 4k7 na PORTB.1
' Pastylkę DS2401 podłączono do końcówki PORTB.1
'-----

'Gdy w programie używane są tylko zmienne typu Byte, można skorzystać
z
'poniższej biblioteki by zmniejszyć rozmiar kodu
$lib "mcsbyte.lib"

Config lwire = Portb.0           'użyjemy tej końcówki
'Na płytce STK200 musi być założona zworka B.0
Dim Ar(8) As Byte , A As Byte , I As Byte

Do
  Wait 1
  lwreset                       'wyślij rozkaz Reset
  Print Err                     'jeśli się nie powiodło Err=1
  lwwrite &H33                  'odczytaj ROM
  For I = 1 To 8
    Ar(i) = lwread()           'wpisz nr ID do tablicy
  Next

  'Możesz także odczytać wszystkie 8 bajtów na raz, usuń w tym celu
  komentarz
  'z następnej linii i usuń całą pętlę FOR..NEXT znajdującą się powyżej
  'Ar(1) = lwread(8)           'czytaj 8 bajtów

  For I = 1 To 8
    Print Hex(ar(i));           'wypiszemy co odebrał
  Next
  Print                         'i CR
Loop

'UWAGA! JEŚLI SKOMPILUJESZ PROGRAM DALSZE INSTRUKCJE NIE ZOSTANĄ
WYKONANE,
'GDYŻ POWYŻSZA PĘTLA DO..LOOP NIE ZOSTANIE NIGDY PRZERWANA!!!

'Nowością jest używanie kilku magistral 1Wire podłączonych do różnych
końcówek portów
'Wtedy instrukcje wyglądają tak:
For I = 1 To 8
  Ar(i) = 0                     'wyczyść tablicę by zobaczyć jak
  działają                      działają
Next

lwreset Pinb , 2               'używamy tego portu i tej końcówki
jako drugiej magistrali
```

```

lwwrite &H33 , 1 , Pinb , 2      'Uwaga! Musi zostać podana liczba
odczytywanych bajtów!
'lwwrite Ar(1) , 5 , Pinb , 2

'Odczyt także wygląda nieco inaczej
Ar(1) = lread(8 , Pinb , 2)      'odczytaj 8 bajtów z PortB na
końcówce 2 (PB2)

For I = 1 To 8
    Print Hex(ar(i));
Next

'Możesz numer bitu umieścić w pętli, by odczytać dane z kilku
magistral sekwencyjnie!
For I = 0 To 3                  'końcówki od 0 (PB0) do 3 (PB3)
    lreset Pinb , I
    lwwrite &H33 , 1 , Pinb , I
    Ar(1) = lread(8 , Pinb , I)
    For A = 1 To 8
        Print Hex(ar(a));
    Next
Print
Next

End

```

1WSEARCHFIRST()

Przeznaczenie:

Funkcja zwraca pierwszy odczytany numer identyfikacyjny z magistrali 1Wire.

Składnia:

```

zmienna = 1WSEARCHFIRST( )
zmienna = 1WSEARCHFIRST( port , pin )

```

gdzie:

<i>zmienna</i>	dowolna zmienna lub tablica gdzie zapisane będzie 8 bajtów numeru ID,
<i>port</i>	nazwa rejestru. np.: PINB, PIND,
<i>pin</i>	numer bitu portu z zakresu 0-7.

Opis:

Funkcja 1WIRESEARCHFIRST() musi być wywołana w celu zainicjowania procedury przeszukiwania magistrali. Potem, aby odczytać następne numery ID należy użyć funkcji 1WSEARCHNEXT().

Uwaga! Zmienna typu String nie może być używana jako zmienna w której umieszczony będzie numer ID elementu. Jeśli pierwszy bajt numeru ID byłby zerem, powodowało by to, że zmienna typu String zwracała by pusty ciąg.

Najlepiej użyć tablicy z 8 komórkami, jak to pokazano w przykładzie.

Funkcja korzysta z 4 bajtów pamięci SRAM. Są to:

__lw_bitstorage	W którym zapisano bity:
	lastdeviceflag bit 0
	id_bit bit 1
	mp_id_bit bit 2
	search_dir bit 3
__lwid_bit_number	
__lwlast_zero	

___lwlast_discrepancy

Asembler:

Wywoływana jest procedura z biblioteki `mcs.lib`:

`_lwire_Search_First`: (wywołuje `_lwire`, `_ADJUST_PIN`, `_ADJUST_BIT_ADDRESS`)

Przekazywane parametry:

R24 : numer końcówki

R30 : port

X : adres zmiennej

Nic nie zwraca w rejestrach

Zobacz także: `1WRITE` , `1WRESET` , `1WREAD` , `1WIRECOUNT` , `1WSEARCHNEXT`

Przykład:

```
'-----
'                                     'lwireSearch.bas
'                                     (c) 2000 MCS Electronics
'                                     wersja b, 27 dec 2000
'-----

Config lwire = Portb.0                'używamy tej końcówki
'Na płytce STK200 „jumper” B.0 musi być zwarty

'Następujące bajty są używane przez procedurę przeszukiwania
magistrali
'___lw_bitstorage , przechowuje bity:
'___lastdeviceflag bit 0
'___id_bit          bit 1
'___cmp_id_bit      bit 2
'___search_dir      bit 3
'___lwid_bit_number, Byte
'___lwlast_zero,    Byte
'___lwlast_discrepancy , Byte
'___lwire_data , string * 7 (8 bajtów)

'[definiowanie zmiennych]
'potrzebne jest 8 bajtów by przechowywać numery ID
Dim Reg_no(8) As Byte

'Potrzebny będzie licznik pętli i zmienna Word/Integer jako licznik
układów
Dim I As Byte , W As Word

'Sprawdzamy pierwsze z urządzeń
Reg_no(1) = lwsearchfirst()

For I = 1 To 8                        'drukujemy numer ID
    Print Hex(reg_no(i));
Next
Print

Do
    'Teraz szukamy następnych urządzeń
    Reg_no(1) = lwsearchnext()
    For I = 1 To 8
        Print Hex(reg_no(i));
    Next
    Print
Loop Until Err = 1
```

```

'Gdy zmienna ERR będzie równa 1, znaczy to że nie znaleziono
następnych urządzeń
'można także policzyć ile ich jest
W = 1wirecount()
'Ważne jest by zmienna była typu Integer/Word
'Rezultat działania tej funkcji jest typu Integer/Word
Print W

'Jako bonus następne procedury:
'Najpierw wpisujemy do tablicy jakiś istniejący numer ID
Reg_no(1) = 1wsearchfirst()
'unremark next line to chance a byte to test the ERR flag
'Reg_no(1) = 2
'teraz sprawdzamy czy numer się zgadza
1wverify Reg_no(1)
Print Err
'err =1 gdy nie ma takiego numeru
'Można także podać numer portu i końcówki: 1wverify reg_no(1),pinb,1
'To samo można zrobić z innymi instrukcjami dla magistrali 1Wire:
'W = 1wirecount(pinb , 1)           'podaj ile jest dołączonych do
PINB.1
End

```

1WSEARCHNEXT()

Przeznaczenie:

Zwraca numer ID następnego znalezionej urządzenia podłączonego do magistrali 1Wire.

Składnia:

```

zmienna = 1WSEARCHNEXT( )
zmienna = 1WSEARCHNEXT( port , pin )

```

gdzie:

<i>zmienna</i>	dowolna zmienna lub tablica gdzie zapisane będzie 8 bajtów numeru ID,
<i>port</i>	nazwa rejestru. np.: PINB, PIND,
<i>pin</i>	numer bitu portu z zakresu 0-7.

Opis:

Funkcja **1WSEARCHFIRST()** musi być wywołana w celu zainicjowania procedury przeszukiwania magistrali. Potem, aby odczytać następne numery ID należy użyć funkcji **1WSEARCHNEXT()**.

Uwaga! Zmienna typu String nie może być używana jako zmienna w której umieszczony będzie numer ID elementu. Jeśli pierwszy bajt numeru ID byłby zerem, powodowało by to, że zmienna typu String, zwracała by pusty ciąg.

Najlepiej użyć tablicy z 8 komórkami, jak to pokazano w przykładzie.

Funkcja korzysta z 4 bajtów pamięci SRAM. Są to:

___1w_bitstorage	W którym zapisano bity:
	lastdeviceflag bit 0
	id_bit bit 1
	mp_id_bit bit 2
	search_dir bit 3
___1wid_bit_number	
___1wlast_zero	
___1wlast_discrepancy	

Asembler:

Wywoływana jest procedura z biblioteki `mcs.lib`:

`_lwire_Search_Next` : (wywołuje `_1WIRE`, `_ADJUST_PIN`, `_ADJUST_BIT_ADDRESS`)

Przekazywane parametry:

R24 : numer końcówki

R30 : port

X : adres zmiennej

Nic nie zwraca

Zobacz także: `1WRITE` , `1WRESET` , `1WREAD` , `1WIRECOUNT` , `1WSEARCHFIRST`

Przykład:

```
'-----
'                                     'lwireSearch.bas
'                                     (c) 2000 MCS Electronics
'                                     wersja b, 27 dec 2000
'-----

Config lwire = Portb.0                'używamy tej końcówki
'Na płycie STK200 „jumper” B.0 musi być zwarty

'Następujące bajty są używane przez procedurę przeszukiwania
magistrali
'__lw_bitstorage , przechowuje bity:
'    lastdeviceflag bit 0
'    id_bit          bit 1
'    cmp_id_bit      bit 2
'    search_dir      bit 3
'__lwid_bit_number, Byte
'__lwlast_zero,     Byte
'__lwlast_discrepancy , Byte
'__lwire_data , string * 7 (8 bajtów)

'[definiowanie zmiennych]
'potrzebne jest 8 bajtów by przechowywać numery ID
Dim Reg_no(8) As Byte

'Potrzebny będzie licznik pętli i zmienna Word/Integer jako licznik
układów
Dim I As Byte , W As Word

'Sprawdzamy pierwsze z urządzeń
Reg_no(1) = lwsearchfirst()

For I = 1 To 8                        'drukujemy numer ID
    Print Hex(reg_no(i));
Next
Print

Do
    'Teraz szukamy następnych urządzeń
    Reg_no(1) = lwsearchnext()
    For I = 1 To 8
        Print Hex(reg_no(i));
    Next
    Print
Loop Until Err = 1

'Gdy zmienna ERR będzie równa 1, znaczy to że nie znaleziono
następnych urządzeń
'można także policzyć ile ich jest
W = lwirecount()
```

```

'Ważne jest by zmienna była typu Integer/Word!
'Rezultat działania tej funkcji jest typu Integer/Word
Print W

'Jako bonus następne procedury:
'Najpierw wpiszemy do tablicy jakiś istniejący numer ID
Reg_no(1) = lwsearchfirst()
'unremark next line to chance a byte to test the ERR flag
'Reg_no(1) = 2
'teraz sprawdzamy czy numer się zgadza
lwverify Reg_no(1)
Print Err
'err =1 gdy nie ma takiego numeru
'Można także podać numer portu i końcówki: lwverify reg_no(1),pinb,1
'To samo można zrobić z innymi instrukcjami dla magistrali lWire:
'W = lwirecount(pinb , 1)           'podaj ile jest dołączonych do
PINB.1
End

```

1WVERIFY()

Przeznaczenie:

Sprawdza czy urządzenie o podanym ID, jest dostępne na magistrali 1Wire.

Składnia:

1WVERIFY *tablica*(1)

gdzie:

tablica dowolna tablica składająca się z 8 bajtów, z zapisanym numerem ID.

Opis:

Wpisuje do zmiennej ERR liczbę 1 jeśli urządzenie nie odpowiada – nie znaleziono go. Jeśli urządzenie jest dołączone do magistrali zwraca 0.

Asembler:

Wywoływana jest procedura z biblioteki mcs.lib: _lwire_Search_First : (wywołuje _1WIRE, _ADJUST_PIN, _ADJUST_BIT_ADDRESS)

Zobacz także: **1WRITE** , **1WRESET** , **1WREAD** , **1WIRECOUNT** , **1WSEARCHFIRST**

Przykład:

```

'-----
'                               'lwireSearch.bas
'                               (c) 2000 MCS Electronics
'                               revision b, 27 dec 2000
'-----

Config lwire = Portb.0           'używamy tej końcówki
'Na płytce STK200 „jumper” B.0 musi być zwarty

'Następujące bajty są używane przez procedurę przeszukiwania
magistrali
'__lw_bitstorage , przechowuje bity:
'__lastdeviceflag bit 0
'   id_bit         bit 1
'   cmp_id_bit     bit 2
'   search_dir     bit 3
'__lwid_bit_number, Byte
'__lwlast_zero,   Byte

```

```

'__lwlast_discrepancy , Byte
'__lwire_data , string * 7 (8 bajtów)

'[definiowanie zmiennych]
'potrzebne jest 8 bajtów by przechowywać numery ID
Dim Reg_no(8) As Byte

'Potrzebny będzie licznik pętli i zmienna Word/Integer jako licznik
układów
Dim I As Byte , W As Word

'Sprawdzamy pierwsze z urządzeń
Reg_no(1) = lwsearchfirst()

For I = 1 To 8                                'drukujemy numer ID
    Print Hex(reg_no(i));
Next
Print

Do
    'Teraz szukamy następnych urządzeń
    Reg_no(1) = lwsearchnext()
    For I = 1 To 8
        Print Hex(reg_no(i));
    Next
    Print
Loop Until Err = 1

'Gdy zmienna ERR będzie równa 1, znaczy to że nie znaleziono
następnych urządzeń
'można także policzyć ile ich jest
W = lwirecount()
'Ważne jest by zmienna była typu Integer/Word!
'Rezultat działania tej funkcji jest typu Integer/Word
Print W

'Jako bonus następne procedury:
'Najpierw wpisujemy do tablicy jakiś istniejący numer ID
Reg_no(1) = lwsearchfirst()
'unremark next line to chance a byte to test the ERR flag
'Reg_no(1) = 2
'teraz sprawdzamy czy numer się zgadza
lwverify Reg_no(1)
Print Err
'err =1 gdy nie ma takiego numeru
'Można także podać numer portu i końcówki: lwverify reg_no(1),pinb,1
'To samo można zrobić z innymi instrukcjami dla magistrali lWire:
'W = lwirecount(pinb , 1)                'podaj ile jest dołączonych do
PINB.1
End

```

1WRITE

Przeznaczenie:

Instrukcja wysyła dane do urządzeń podłączonych do magistrali 1Wire.

Składnia:

1WRITE zmienna
1WRITE zmienna , il_bajtów

1WWRITE *zmienna* , *il_bajtów* , *port* , *pin*

gdzie:

<i>zmienna</i>	zmienna w której umieszczono wysyłane dane. Można także podawać stałe liczbowe.
<i>il_bajtów</i>	określa ilość wysyłanych bajtów. Musi być użyta gdy podany jest <i>port</i> i <i>pin</i> .
<i>port</i>	nazwa rejestru. np.: PINB, PIND,
<i>pin</i>	numer bitu portu z zakresu 0-7.

Opis:

Nowością jest możliwość pracy z kilkoma urządzeniami 1Wire dołączonymi do różnych końcówek portu mikrokontrolera. By używać tego rozszerzonego trybu, w każdej operacji na szynie 1Wire musi być określony port i końcówka portu (pin).

Instrukcje 1WRESET, 1WWRITE i 1WREAD mogą nadal pracować razem jeśli są używane według starej składni. Wtedy końcówkę portu będącą szyną 1Wire określa się w opcjach kompilatora lub instrukcją **CONFIG 1WIRE**.

Zobacz także: **1WREAD** , **1WRESET**

Przykład:

```
'-----
'               1WIRE.BAS (c) 2000 MCS Electronics
' Demonstracja lwreset, lwwrite i lwread()
' Wymagany jest rezystor podciągający 4k7 na PORTB.1
' Pastylkę DS2401 podłączono do końcówki PORTB.1
'-----
'Gdy w programie używane są tylko zmienne typu Byte, można skorzystać
z
'poniższej biblioteki by zmniejszyć rozmiar kodu
$lib "mcsbyte.lib"

Config lwire = Portb.0           'użyjemy tej końcówki
'Na płytce STK200 musi być założona zworka B.0
Dim Ar(8) As Byte , A As Byte , I As Byte

Do
  Wait 1
  lwreset                       'wyślij rozkaz Reset
  Print Err                     'jeśli się nie powiodło Err=1
  lwwrite &H33                  'odczytaj ROM
  For I = 1 To 8
    Ar(i) = lwread()            'wpisz nr ID do tablicy
  Next

  'Możesz także odczytać wszystkie 8 bajtów na raz, usuń w tym celu
  komentarz
  'z następnej linii i usuń całą pętlę FOR..NEXT znajdującą się powyżej
  'Ar(1) = lwread(8)             'czytaj 8 bajtów

  For I = 1 To 8
    Print Hex(ar(i));           'wypiszemy co odebrał
  Next
  Print                         'i CR
Loop

'UWAGA! JEŚLI SKOMPILUJESZ PROGRAM DALSZE INSTRUKCJE NIE ZOSTANĄ
WYKONANE,
'GDYŻ POWYŻSZA PĘTLA DO..LOOP NIE ZOSTANIE NIGDY PRZERWANA!!!
```

```

'Nowością jest używanie kilku magistral 1Wire podłączonych do różnych
końcówek portów
'Wtedy instrukcje wyglądają tak:
For I = 1 To 8
    Ar(i) = 0                                'wyczyść tablicę by zobaczyć jak
    działają
Next

lwreset Pinb , 2                            'używamy tego portu i tej końcówki
jako drugiej magistrali
lwwrite &H33 , 1 , Pinb , 2                  'Uwaga! Musi zostać podana liczba
odczytywanych bajtów!
'lwwrite Ar(1) , 5 , Pinb , 2

'Odczyt także wygląda nieco inaczej
Ar(1) = lwread(8 , Pinb , 2)                 'odczytaj 8 bajtów z PortB na
końcówce 2 (PB2)

For I = 1 To 8
    Print Hex(ar(i));
Next

'Możesz numer bitu umieścić w pętli, by odczytać dane z kilku
magistral sekwencyjnie!
For I = 0 To 3                                'końcówki od 0 (PB0) do 3 (PB3)
    lwreset Pinb , I
    lwwrite &H33 , 1 , Pinb , I
    Ar(1) = lwread(8 , Pinb , I)
    For A = 1 To 8
        Print Hex(ar(a));
    Next
    Print
Next

End

```

ABS()

Przeznaczenie:

Zwraca wartość absolutną zmiennej numerycznej.

Składnia

zmienna1 = ABS(*zmienna2*)

gdzie:

<i>zmienna1</i>	zmienna której przypisuje się wartość absolutną,
<i>zmienna2</i>	zmienna której wartość absolutną trzeba obliczyć, musi być typu Integer, Long lub Single.

Opis:

Wartość absolutna jest zawsze liczbą dodatnią.

Różnice w stosunku do QBasic-a:

Nie jest możliwe stosowanie stałych jako parametru *zmienna2* w funkcji ABS().

Asembler:

	<i>zmienne Integer</i>	<i>zmienne Long</i>
Wywołanie:	<code>_abs16</code>	<code>_abs32</code>

We:	R16-R17	R16-R19
Wy:	R16-R17	R16-R19

Wywołuje procedurę `_Fltabsmem` z biblioteki `fp_trig` dla zmiennych typu `Single`.

Przykład:

```

Dim a As Integer, c As Integer

a = -1000
c = Abs(a)
Print c

End

```

ACOS() *(Nowość w wersji 1.11.6.8)*

Przeznaczenie:

Zwraca wartość arcuscosinusa liczby zapisanej w radianach. Jest odwrotnością funkcji `COS()`.

Składnia:

`zmienna = ACOS(wartość)`

gdzie:

<code>zmienna</code>	zmienna typu <code>Single</code> której przypisuje się wynik działania funkcji,
<code>wartość</code>	liczba typu <code>Single</code> , której wartość arcuscosinusa należy obliczyć.

Opis:

Podana jako parametr funkcji wartość powinna zawierać się w granicach -1 do 1 . Funkcja zwraca wtedy wartości od π do 0 .

Jeśli podana wartość jest mniejsza niż -1 wtedy funkcja zwraca π . Jeśli zaś większa niż 1 wtedy funkcja zwraca 0 .

Uwaga! Jeśli podana jako parametr funkcji wartość jest trochę większa niż $+1$ lub -1 , może wystąpić efekt zaokrąglania liczb zmiennoprzecinkowych i funkcja zwróci wartość kąta jak dla wartości 1.0 (lub -1.0). To jest właśnie przyczyną, że zwracana jest wartość graniczna (π lub 0), jeśli parametr funkcji jest spoza limitu. Generalnie, użytkownik powinien zachować ostrożność, gdy podana wartość znajduje się pomiędzy -1 a $+1$.

Wszystkie funkcje trygonometryczne używają radianów. Aby przekształcić radiany na stopnie i odwrotnie należy używać funkcji `DEG2RAD` oraz `RAD2DEG`.

Zobacz także: `RAD2DEG` , `DEG2RAD` , `COS` , `SIN` , `TAN` , `ATN` , `ASIN` , `ATN2`

Przykład:

```

Dim S As Single , x As Single , y As Single

x = 0.5 : S = Acos(x)
Print S

```

ALIAS

Przeznaczenie:

Pozwala na zdefiniowanie przyjaznej nazwy zmiennej.

Składnia:

nowa_nazwa **ALIAS** *stara_nazwa*

gdzie:

<i>stara_nazwa</i>	Bieżąca nazwa zmiennej,
<i>nowa_nazwa</i>	Nowa nazwa zmiennej.

Opis:

Instrukcję ALIAS stosuje się do zmiany nazw zmiennych na *przyjazne*. Jest to szczególnie użyteczne przy nazywaniu rejestrów, czy portów procesora. Nie zaleca się stosować do zmiennych definiowanych instrukcją DIM.

Zobacz także: **CONST**

Przykład:

```

Config Pinb.1 = Output
Direction Alias Portb.1      'można teraz używać direction zamiast
PORTB.1
Do
    Set Direction              'to samo co SET PORTB.1
    Waitms 1
    Reset Direction
Loop

End

```

ASC()**Przeznaczenie:**

Zwraca kod ASCII podanego znaku.

Składnia:

zmienna = **ASC**(*tekst*)

gdzie:

<i>zmienna</i>	nazwa zmiennej której przypisywany jest wynik. Może być typu Byte, Integer, Word lub Long.
<i>tekst</i>	zmienna tekstowa lub stała tekstowa, do konwersji.

Opis:

Funkcja zwraca kod ASCII znaku zawartego w zmiennej tekstowej. Dla zmiennych typu String dłuższych niż jeden znak, funkcja przekształca tylko pierwszy znak ciągu. Jeśli ciąg był pusty (""), funkcja zwraca kod zero.

Zobacz także: **CHR**

Przykład:

```

Dim a As Byte, s As String * 10

s = "ABC"
a = Asc(s)
Print a          'wydrukuję liczbę 65

End

```

ASIN() (Nowość w wersji 1.11.6.8)

Przeznaczenie:

Zwraca wartość arcussinusa liczby zapisanej w radianach. Jest odwrotnością funkcji SIN().

Składnia:

zmienna = **ASIN**(*wartość*)

gdzie:

<i>zmienna</i>	zmienna typu Single której przypisuje się wynik działania funkcji,
<i>wartość</i>	liczba typu Single, której wartość arcussinusa należy obliczyć.

Opis:

Podana jako parametr funkcji wartość powinna zawierać się w granicach -1 do 1 . Funkcja zwraca wtedy wartości od $-\pi/2$ do $+\pi/2$.

Jeśli podana wartość jest mniejsza niż -1 wtedy funkcja zwraca $-\pi/2$. Jeśli zaś większa niż 1 wtedy funkcja zwraca $+\pi/2$.

Uwaga! Jeśli podana jako parametr funkcji wartość jest trochę większa niż $+1$ lub -1 , może wystąpić efekt zaokrąglania liczb zmiennoprzecinkowych i funkcja zwróci wartość kąta jak dla wartości 1.0 (lub -1.0). To jest właśnie przyczyną, że zwracana jest wartość graniczna ($-\pi/2$ lub $+\pi/2$), jeśli parametr funkcji jest spoza limitu. Generalnie, użytkownik powinien zachować ostrożność, gdy podana wartość znajduje się pomiędzy -1 a $+1$.

Wszystkie funkcje trygonometryczne używają radianów. Aby przekształcić radiany na stopnie i odwrotnie należy używać funkcji DEG2RAD oraz RAD2DEG.

Zobacz także: **RAD2DEG** , **DEG2RAD** , **COS** , **SIN** , **TAN** , **ATN** , **ACOS** , **ATN2**

Przykład:

```
Dim S As Single , x As Single , y As Single

x = 0.5 : S = Asin(x)
Print S
```

ATN() (Nowość w wersji 1.11.6.8)

Przeznaczenie:

Zwraca wartość arcustangensa liczby zapisanej w radianach. Jest odwrotnością funkcji TAN().

Składnia:

zmienna = **ATN**(*wartość*)

gdzie:

<i>zmienna</i>	zmienna typu Single której przypisuje się wynik działania funkcji,
<i>wartość</i>	liczba typu Single, której wartość arcustangensa należy obliczyć.

Opis:

Wszystkie funkcje trygonometryczne używają radianów. Aby przekształcić radiany na stopnie i odwrotnie należy używać funkcji DEG2RAD oraz RAD2DEG.

Zobacz także: **RAD2DEG** , **DEG2RAD** , **COS** , **SIN** , **TAN** , **ASIN** , **ACOS** , **ATN2**

Przykład:

```
Dim S As Single , x As Single , y As Single

S = Atn(1) * 4
Print S
```

'Powinno wydrukować 3.141593

ATN2() (Nowość w wersji 1.11.6.8)

Przeznaczenie:

ATN2 zwraca wartość arcustangensa z pełnego zakresu kąta.

Składnia:

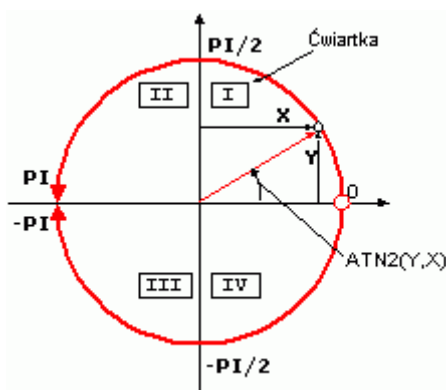
zmienna = ATN2(*x* , *y*)

gdzie:

zmienna zmienna typu Single której przypisuje się wynik działania funkcji,
x zmienna typu Single określająca odległość na osi OX.
y zmienna typu Single określająca odległość na osi OY.

Opis:

Funkcja ATN działa tylko dla liczb w zakresie od $-\pi/2$ (-90°) do $\pi/2$ (90°). Funkcja ATN2 w pełnym zakresie od $-\pi$ (-180°) do $+\pi$ (180°). Rezultat działania funkcji jest zależny od wartości oraz znaku parametrów *x* i *y*. Ilustruje to rysunek:



Rysunek 17 Wyjaśnienia działania funkcji ATN2.

Tabela 8 Zakres wartości zwracanych przez funkcję ATN2.

Ćwiartka	Znak Y	Znak X	Wynik funkcji ATN2()
I	+	+	0 do $\pi/2$
II	+	-	$\pi/2$ do π
III	-	-	$-\pi/2$ do $-\pi$
IV	-	+	0 do $-\pi/2$

Jeśli podasz stosunek *x/y* jak parametr funkcji ATN, otrzymasz taki sam rezultat jak funkcja ATN2, której parametr *x* jest większy niż zero (prawa strona na układzie współrzędnych). ATN2 używa dwóch parametrów *x* oraz *y*, co pozwala na obliczenie wartości z pełnego zakresu kąta ($0 - 360^\circ$).

Wszystkie funkcje trygonometryczne używają radianów. Aby przekształcić radiany na stopnie i odwrotnie należy używać funkcji DEG2RAD oraz RAD2DEG.

Zobacz także: RAD2DEG , DEG2RAD , COS , SIN , TAN , ATN

Przykład:

```
Dim S As Single , x As Single , y As Single

S = Atn2(x,y)
Print S
```

BAUD

Przeznaczenie:

Zmienia szybkość transmisji urządzenia typu UART.

Składnia:

BAUD = *szybkość*

BAUD #kanał , *stała*

gdzie:

<i>szybkość</i>	szybkość transmisji jaka ma obowiązywać od tej pory,
<i>kanał</i>	numer kanału transmisji w programowym urządzeniu typu UART,
<i>stała</i>	liczba określająca szybkość transmisji w danym kanale.

Opis:

Można zmienić szybkość pracy urządzenia UART w trakcie działania programu. Format pierwszy **BAUD** = odnosi się do wbudowanego w procesor łącza szeregowego. Drugi format **BAUD #** odnosi się do programowej transmisji w symulowanym urządzeniu typu UART.

Nie należy mylić instrukcji **BAUD** z dyrektywą **\$BAUD**. Dyrektywa **\$BAUD** zmienia domyślne ustawienia w opcjach kompilatora, a instrukcja **BAUD** programowo zmienia bieżącą szybkość transmisji.

Na marginesie dodam by nie mylić także dyrektywy **\$CRYSTAL** ze zmienną **CRYSTAL**.

Zobacz także: **\$CRYSTAL**, **\$BAUD**

Przykład:

```
$baud = 2400
$crystal = 14000000      'kwarc 14 MHz

Print "Hello"

'teraz zmienimy szybkość transmisji
Baud = 9600
Print "Czy zmieniłeś szybkość także w terminalu?"

End
```

BCD()

Przeznaczenie:

Zamienia wartość zapisaną w kodzie BCD na tekst.

Składnia:

```
PRINT BCD( zmienna )
LCD BCD( zmienna )
```

gdzie:

<i>zmienna</i>	wartość poddana konwersji. Może być typu Byte, Integer,
----------------	---

Word, Long lub stałą liczbową.

Opis:

Gdyby zaistniała konieczność przedstawienia wartości BCD jako tekstu można użyć funkcji BCD(). Funkcja ta zwraca wartość z dodanym na początku zerem.

Uwaga! Funkcja ta działa tylko w połączeniu z instrukcjami **PRINT** i **LCD**.

Aby stosować konwersję do innych celów niż drukowanie wartości, należy używać funkcji MAKEBCD. Konwersję odwrotną zapewnia funkcja MAKEDEC.

Zobacz także: **MAKEDEC** , **MAKEBCD**

Asembler:

Wywołanie: `_BcdStr`

We: X – zawiera adres zmiennej

Wy: R0 – liczba bajtów, odłożonych na „ramce”

Przykład:

```
Dim a as Byte
a = 65
Print a           '65
Print Bcd(a)      '41

End
```

BIN()

Przeznaczenie:

Zwraca w formie tekstu binarną reprezentację liczby.

Składnia:

`zmienna = BIN(liczba)`

gdzie:

<i>zmienna</i>	zmienna tekstowa, w której znajdzie się liczba zapisana w formacie binarnym,
<i>liczba</i>	liczba poddana konwersji, może być stałą lub zmienną typu Integer, Word, Long czy Byte.

Opis:

Funkcja BIN() może być użyta podczas wyświetlania stanu końcówek portu.

Gdy zmienna zawiera wartość zapisaną binarnie **&B10100011**, do zmiennej tekstowej zostaje wpisany ciąg: "10100011", który może być łatwo pokazany na wyświetlaczu lub przesłany przez port szeregowy.

Zobacz także: **HEX** , **STR** , **VAL** , **HEXVAL** , **BINVAL()**

Przykład:

```
Dim A As Byte
A = &H14
Print Bin(a)      'wydrukuj 00010100
```

BINVAL()

Przeznaczenie:

Zamienia tekstową reprezentację liczby binarnej na odpowiadającą jej wartość.

Składnia:

zmienna = **BINVAL**(*liczba*)

gdzie:

zmienna
liczba

zmienna do której zapisana będzie liczba po konwersji,
liczba binarna poddana konwersji, zapisana w zmiennej
tekstowej.

Zobacz także: **BIN** , **VAL** , **STR**

Przykład:

```
Dim a As Byte, s As String * 10

s = "1100"
a = Val(s)           'konwersja
Print a              'wydrukuje 12

End
```

BIN2GREY()

Przeznaczenie:

Zwraca wartość jaką reprezentuje zmienna w kodzie Greya.

Składnia:

zmienna1 = **BIN2GREY**(*zmienna2*)

gdzie:

zmienna1
zmienna2

zmienna do której zapisana będzie liczba po konwersji,
zmienna której zawartość poddawana jest konwersji.

Opis:

Kod Greya jest używany w koderach obrotowych.

Funkcja BIN2GREY() może konwertować zmienne typu Byte, Integer, Word lub Long. Typ podanej jako parametr zmiennej określa także typ zwracanej wartości.

Zobacz także: **GREY2BIN**

Asembler:

W zależności od typu podanej zmiennej, jest wywoływana odpowiednia procedura z biblioteki mcs.lbx: _grey2Bin dla typu Byte, _grey2bin2 dla typów Integer/Word oraz _grey2bin4 dla Long.

Przykład:

```
'-----
'
'                                     (c) 2001-2002 MCS Electronics
' Przykład ten pokazuje jak używać funkcji Bin2Grey i Grey2Bin
```

```

' Podziękowania dla Josef Franz Vögel za poprawki i za wersję dla
typów Word/Long
'-----
-----

'Bin2Grey() zamienia bajt, Integer, Word, Long na zapisany w kodzie
Greya.
'Grey2Bin() zamienia wartość zapisana w kodzie Greya na wartość
binarną

Dim B As Byte                                     'może być także Word, Integer,
Long

Print "BIN" ; Spc(8) ; "GREY"
For B = 0 To 15
    Print B ; Spc(10) ; Bin2grey(b)
Next

Print "BIN" ; Spc(8) ; "GREY"
For B = 0 To 15
    Print B ; Spc(10) ; Grey2bin(b)
Next

End

```

BITWAIT

Przeznaczenie:

Instrukcja oczekująca na ustawienie lub wyzerowanie bitu.

Składnia:

BITWAIT *bit* , **SET** | **RESET**

gdzie:

bit nazwa zmiennej bitowej lub bitu w porcie mikrokontrolera, np.:
PINB.1

Opis:

Instrukcja powoduje wstrzymanie działania programu, dopóki określony bit nie zostanie ustawiony w stan 1 (parametr **SET**) lub w stan 0 (parametr **RESET**).

Gdy jako bit została podana nazwa zdefiniowanej zmiennej bitowej, należy zadbać by była ona modyfikowana przez program – np. w procedurze obsługującej przerwanie. Jeśli będą używane rejestry procesora ustawiane bądź zerowane sprzętowo jak np. PINB.0, wtedy wszystko zależy od reakcji urządzenia zewnętrznego.

Asembler:

```

label1:
    Sbic PINB.0 , label2
    Rjmp label1
label2:

```

Przykład:

```

Dim a as Bit

Bitwait a , Set      'czekaj aż bit będzie ustawiony
Bitwait Pinb.7, Reset 'czekaj aż 7 linia portu B będzie w stanie 0.

```

End

BYVAL, BYREF

Przeznaczenie:

Definiują sposób przekazywania parametrów procedurom i funkcjom użytkownika.

Składnia:

```
SUB Nazwa_Procedury( [ BYVAL | BYREF ] parametr)  
FUNCTION Nazwa_Procedury( [ BYVAL | BYREF ] parametr)
```

gdzie:

parametr nazwa zmiennej lub stała.

Opis:

Argumenty te występują w deklaracjach parametrów procedur lub funkcji użytkownika.

Normalnie parametry są przekazywane jako adresy zmiennych w pamięci. Dlatego każda operacja przypisująca nową wartość parametrowi wewnątrz procedury jest „widoczna” na zewnątrz w programie głównym, co może doprowadzić do różnych nieprawidłowości w działaniu programu.

Jeśli przy deklaracji parametru użyty zostanie argument **BYVAL**, jako parametr zostanie przekazana wartość jaką reprezentuje zmienna. Dzieje się to za pomocą tymczasowo utworzonej zmiennej, do której przepisana jest wartość parametru. Zmienna tymczasowa jest potem przekazywana jako właściwy parametr procedury. Dlatego zmiana wartości parametru w procedurze nie będzie „widoczna” na zewnątrz, gdyż dotyczy tylko jej kopii.

Standardowo przekazywanie parametrów następuje przez adres zmiennej, z wszystkimi tego konsekwencjami. Stosowanie słowa **BYREF** nie jest zatem obowiązkowe i jest to argument domyślny. Ponadto przekazywanie przez adres zmniejsza nieco zapotrzebowanie na kod i przestrzeń ramki.

Zobacz także: **CALL** , **DECLARESUB** , **SUB** , **FUNCTION**

Przykład:

```
Declare Sub Test(Byval X As Byte, Byref Y As Byte, Z As Byte)
```

CALL

Przeznaczenie:

Wywołuje i wykonuje procedurę.

Składnia:

```
CALL nazwa [( parametr1 [, parametrn] )]
```

gdzie:

nazwa nazwa wywoływanej procedury,
parametr1 dowolna zmienna lub stała.
parametrn

Opis:

Instrukcja CALL wywołuje procedurę napisaną przez użytkownika. Ilość parametrów zależy od zadeklarowanej w instrukcji **DECLARE SUB**. Można także wywoływać procedury bezparametrowe.

W przeciwieństwie do wywoływania podprogramów instrukcją **GOSUB**, stosowanie instrukcji CALL i procedur, pozwala na przejrzyste przekazanie parametrów.

Uwaga! Jest szczególnie ważne by procedura została zadeklarowana przez **DECLARE SUB**, przed jej używaniem w programie. Ilość parametrów podanych podczas wywołania musi się zgadzać z ilością parametrów zadeklarowanych.

Nie mniej ważną rzeczą przy wywoływaniu procedury jest sposób przekazywania stałych jako parametrów. Gdy parametr ma być stałą konieczne należy użyć słowa **BYVAL** w deklaracji procedury.

Przewidziano także możliwość wywoływania procedur bez słowa **CALL**. Przy tego typu wywołaniu nie stosuje się zamykania parametrów w nawiasach:

```
Call MojaProc(x,y,z)
```

jest równoważne:

```
MojaProc x , y , x
```

Jest to także sposób na lokalne utworzenie nowych instrukcji języka BASCOM BASIC.

Zobacz także: **DECLARE SUB** , **SUB** , **EXIT** , **FUNCTION** , **LOCAL**

Przykład:

```
Dim A As Byte, B As Byte 'określimy kilka zmiennych

'deklaracja procedury
Declare Sub Test(b1 As Byte, Byval b2 As Byte)

A = 65 'przypisujemy wartość zmiennej A
Call Test (A , 5) 'wywołujemy proc. Test z parametrem A i
stałą
Test A , 5 'inny sposób wywołania
Print A 'drukujemy zawartość A

End

Sub Test(b1 As byte, Byval b2 As byte)
'parametry muszą być takie same jak zadeklarowano!!
Lcd b1 'piszemy na LCD
Lowerline
Lcd Bcd(b2)
B1 = 10 'zmieniamy parametr
B2 = 15 'i tą też

End Sub
```

Szczegóły dotyczące argumentów **BYREF** i **BYVAL**.

W powyższym przykładzie pokazano, że przypisanie nowej wartości parametrowi **B2** procedury, nie zmieni nic w programie głównym. Inaczej się ma zmiana parametru **B1**, gdyż spowoduje to zmianę zawartości zmiennej **A** głównego programu!

Jest to ilustracja różnicy pomiędzy argumentami **BYVAL** i **BYREF** używanymi przy deklaracji procedury. Kiedy użyto słowa **BYVAL**, znaczy to, że jako parametr przekazywana jest tylko wartość zmiennej. Mechanizm ten wykorzystuje stworzoną tymczasowo zmienną, do której przepisywana jest ta wartość, a sama zmienna tymczasowa jest przekazywana jako parametr. Wtedy procedura może wykonywać wszelkie operacje na parametrze, bez żadnych ujemnych skutków dla programu głównego.

Jeśli słowo **BYVAL** nie występuje, domyślnie jest używany argument **BYREF**, który powoduje przekazanie adresu zmiennej użytej jako parametr w programie głównym. Dlatego zmiana w procedurze wartości parametru **B1**, spowodowała także zmianę zawartości zmiennej **A**.

CHECKSUM()

Przeznaczenie

Oblicza sumę kontrolną zmiennej typu String.

Składnia:

```
PRINT CHECKSUM( tekst )  
zmienna = CHECKSUM( tekst )
```

gdzie:

<i>tekst</i>	zmienna tekstowa (typu String)
<i>zmienna</i>	zmienna numeryczna do której przepisana będzie suma kontrolna.

Opis:

Suma kontrolna jest obliczana jako suma kodów znaków składających się na zmienna typu String. Tak obliczoną sumę kontrolną można stosować przy transmisji szeregowej, jako kontrolę jej poprawności.

Suma kontrolna jest obliczana jako bajt. Poniższy fragment (w języku Basic) jest równoważny funkcji CHECKSUM:

```
DIM Check AS Byte  
  
Check = 255  
FOR x = 1 TO LEN(s$)  
    Check = Check - ASC(MID$(s$, x, 1))  
NEXT
```

Zobacz także: [CRC8](#) , [CRC16](#)

Przykład:

```
Dim s As String * 10 'jakaś zmienna  
  
s = "test"           'wpisujemy jakiś tekst  
Print Checksum(s)    'wynikiem będzie (192)  
  
End
```

CHR()

Przeznaczenie:

Zamienia liczbowy kod na odpowiadający mu znak zestawu ASCII.

Składnia:

```
PRINT CHR( zmienna2 )  
zmienna1 = CHR( zmienna2 )
```

gdzie:

<i>zmienna2</i>	zmienna numeryczna lub stała,
<i>zmienna1</i>	zmienna tekstowa.

Opis:

Funkcja ta zamienia wartość zapisaną w zmiennej na odpowiadający tej wartości kod ASCII. Jest często używana do wypisywania znaków, których nie można uzyskać bezpośrednio z klawiatury podczas pisania programu. Dotyczy to szczególnie znaków wyświetlacza LCD zdefiniowanych przez użytkownika (kody 0-7).

Może być także użyteczna gdy kod znaku jest dopiero obliczany w programie.

Zobacz także: **ASC()**

Przykład:

```
Dim a As Byte      'jakaś zmienna

a = 65              'teraz ma 65
LCD a              'pisz wartość (65)
Lowerline
Lcd Hex (a)         'teraz wartość w Hex (41)
Lcd Chr (a)         'a teraz znak czyli A

End
```

CIRCLE (Nowość w wersji 1.11.6.8)

Przeznaczenie:

Rysuje okrąg na ekranie graficznego wyświetlacza LCD.

Składnia:

CIRCLE (*x0* , *y0*) , *promień* , *kolor*

gdzie:

<i>x0</i> , <i>y0</i>	punkt środkowy okręgu,
<i>promień</i>	promień okręgu,
<i>kolor</i>	kolor linii okręgu.

Opis:

Instrukcja CIRCLE jest odpowiednikiem tej samej instrukcji występującej w dialekcie QBasic. Jedyną różnicą jest sposób określania koloru. W języku BASCOM BASIC kolor może być ustawiony tylko jako 0 lub 255. Kolor 0 kasuje okrąg, a kolor 255 rysuje go.

Przykład:

```
'-----
'                                     (c) 2001-2002 MCS Electronics
'Demonstruje instrukcje graficzne dla wyświetlaczy LCD z kontrolerem
T6963C
'-----
'Sposób podłączenia wyświetlacza LCD:
'końcówka                podłączona do
'1          GND           GND
'2          GND           GND
'3          +5V           +5V
'4          -9V           -9V potencjometr
'5          /WR           PORTC.0
'6          /RD           PORTC.1
'7          /CE           PORTC.2
'8          C/D           PORTC.3
'9          NC            nie podłączone
'10         RESET        PORTC.4
'11-18     D0-D7         PA
'19        FS            PORTC.5
'20        NC            nie podłączone

$crystal = 8000000
'Najpierw definiujemy, że używany będzie graficzny wyświetlacz LCD
Config Graphlcd = 240 * 128 , Dataport = Porta , Controlport = Portc ,
Ce = 2 , Cd = 3 , Wr = 0 , Rd = 1 , Reset = 4 , Fs = 5 , Mode = 8
'Dataport określa port do jakiego podłączono szynę danych LCD
'Controlport określa który port jest używany do sterowania LCD
```

```

'CE, CD itd. to numery bitów w porcie Controlport
'Na przykład CE = 2 gdyż jest podłączony do końcówki PORTC.2
'Mode = 8 daje 240 / 8 = 30 kolumn , Mode=6 daje 240 / 6 = 40 kolumn

'Deklaracje (y nie używane)
Dim X As Byte , Y As Byte

'Kasujemy ekran wyświetlacza (tekst oraz grafikę)
Cls
'Inne opcje to:
' CLS TEXT   kasuje tylko stronę tekstową
' CLS GRAPH  kasuje tylko stronę graficzną

Cursor Off

Wait 1
'Instrukcja Locate działa normalnie jak dla wyświetlaczy tekstowych
' LOCATE LINIA , KOLUMNA  numer linii w zakresie 1-8, kolumny 0 - 30

Locate 1 , 1

'Teraz napiszemy jakiś tekst
Lcd "MCS Electronics"
'i jeszcze jakieś inne w linii 2 i 3
Locate 2 , 1 : Lcd "T6963c support"
Locate 3 , 1 : Lcd "1234567890123456789012345678901234567890"
Locate 16 , 1 : Lcd "A to będzie w linii 16 (najniższej)"

Wait 2

Cls Text

'Użyj nowej instrukcji LINE by narysować pudełko
'LINE(X0 , Y0) - (X1 , Y1), on/off
Line(0 , 0) -(239 , 127) , 255      ' poprzeczne linie
Line(0 , 127) -(239 , 0) , 255
Line(0 , 0) -(240 , 0) , 255        ' górna linia
Line(0 , 127) -(239 , 127) , 255    ' dolna linia
Line(0 , 0) -(0 , 127) , 255        ' lewa linia
Line(239 , 0) -(239 , 127) , 255    ' prawa linia

Wait 2
'Można także rysować linie za pomocą PSET X ,Y , on/off
'parametr on/off to: 0 - skasowanie piksela, inna - postawienie
piksela
For X = 0 To 140
    Pset X , 20 , 255                'narysuj punkt
Next

For X = 0 To 140
    Pset X , 127 , 255               'narysuj punkt
Next

Wait 2

'Teraz czas na Circle
'Circle(X , Y) , radius, color
'X , y to środek okręgu, color musi być ustawiony na 255 by narysować,
a 0 by skasować okrąg
For X = 1 To 10
    Circle(20 , 20) , 20 , 255      'rysujemy okrąg

```

```

    Wait 1
    Circle (20 , 20) , 20 , 0          'usuwamy okrąg
    Wait 1
Next

Wait 2

'A teraz czas najwyższy na obrazek
'SHOWPIC X , Y , etykieta
'Etykieta określa początek danych gdzie zapisany jest obrazek
Showpic 0 , 0 , Plaatje
Showpic 0 , 64 , Plaatje              'Pokaż 2, przecież mamy duży
ekran
Wait 2
Cls Text                             'skasuj tekst
End

'Tutaj znajduje się obrazek
Plaatje:
'Dyrektywa $BGF spowoduje umieszczenie danych obrazka w tym miejscu
$bgf "mcs.bgf"

'Możesz dodać inne obrazki tutaj

```

CLS

Przeznaczenie

Usuwa znaki z ekranu wyświetlacza LCD.

Składnia (dla wyświetlaczy alfanumerycznych):

CLS

Składnia (dla wyświetlaczy graficznych):

CLS [TEXT | GRAPH]

Opis:

Wykonanie tej instrukcji powoduje wyczyszczenie zawartości wyświetlacza LCD, oraz ustawienie kursora na początek pierwszej linii. Instrukcja ta nie narusza pamięci CG-RAM gdzie są umieszczone znaki zdefiniowane przez użytkownika.

Dla wyświetlaczy graficznych instrukcja CLS kasuje stronę tekstową i graficzną. Podanie jednak argumentu **TEXT** lub **GRAPH**, spowoduje że tylko dane z określonej „strony graficznej” zostaną usunięte.

Zobacz także: \$LCD , LCD , SHIFTLCD , SHIFTCURSOR

Przykład:

```

Cls          'kasuj wyświetlacz
Lcd "Hello"  'wypisz to co zwykle :)

End

```

CLOCKDIVISION

Przeznaczenie:

Zmienia stopień podziału dzielnika częstotliwości zegarowej w kontrolerach serii AVR MEGA.

Składnia:

CLOCKDIVISON = *wartość*

gdzie:

wartość zmienna lub stała liczba określająca wartość dzielnika.
Dozwolone są wartości podziału od 2 do 127. Wpisanie zera
wyłącza dzielnik.

Opis:

W procesorach serii MEGA 103 i 603, częstotliwość taktowania może być podzielona w celu zmniejszenia poboru mocy.

Należy uważać, gdyż działanie niektórych instrukcji zależy od szybkości pracy procesora. Dla przykładu instrukcja **WAITMS** przy dzielniku ustawionym na 2, będzie odmierzać czas dwa razy dłuższy!

Zobacz także: **POWERSAVE**

Przykład:

```
$baud = 2400
ClockDivision = 2
```

End

CLOSE

Przeznaczenie:

Zamyka otwarty kanał transmisji programowego lub sprzętowego urządzenia UART.

Składnia:

CLOSE #*kanal*

gdzie:

kanal numer otwartego kanału.

Opis:

Instrukcja ta zamyka otwarty wcześniej kanał instrukcją **OPEN**. Numer kanału jest nadawany przy operacji otwarcia.

Uwaga! Każdy otwarty kanał powinien zostać zamknięty gdy nie jest już wykorzystywany.

Zobacz także: **OPEN** , **PRINT**

Przykład:

```
'-----
'                (c) 2000 MCS Electronics
'                OPEN.BAS
'                demonstruje programowy UART
'-----
$crystal = 10000000          'zmień jeśli używasz innego kwarcu

Dim B As Byte

'Opcjonalnie można dostroić szybkość transmisji.
'Dlaczego powinno się to zrobić?
'Niektóre procesory mają wbudowany generator który może nie
'pracować na wymaganej częstotliwości.
'Zależy ona od napięcia zasilania, temperatury itp.
'Można oczywiście zmienić wartość podaną w dyrektywie $CRYSTAL lub
używając:
```

```

'BAUD #1,9610

'W tym przykładzie jest używana płytką DT006 z www.simmstick.com
'Pozwala to na łatwe przetestowanie z wykorzystaniem istniejącego
'portu szeregowego.
'Używany jest tam układ MAX232.
'Ponieważ używane są końcówki z których korzysta sprzętowy UART
'NIE MOŻE on być używany.
'Sprzętowy UART jest używany wraz z instrukcjami PRINT, INPUT i
innymi.

'Używany jest programowy UART.
Waitms 100

'otwieramy kanał wyjściowy
Open "comd.1:19200,8,n,1" For Output As #1
Print #1 , "Transmisja szeregową"

'Teraz otwieramy kanał wejściowy
Open "comd.0:19200,8,n,1" For Input As #2
'ponieważ nie ma połączenia pomiędzy końcówką wejściową a wyjściową
'nie jest wysyłane echo podczas naciskania klawiszy
Print #1 , "Liczba"
'odczytaj liczbę
Input #2 , B
'i wydrukuj
Print #1 , B

'Teraz odczytujemy znaki aż otrzymamy kod Esc
'Funkcją INKEY() można sprawdzić czy jakiś znak jest dostępny.
'By użyć tej funkcji w połączeniu z programowym UART-em należy podać
numer kanału
Do
'czytamy
B = Inkey(#2)
'jeśli kod > 0 wtedy odebraliśmy znak
If B > 0 Then
Print #1 , Chr(b) 'drukujemy go
End If
Loop Until B = 27

Close #2
Close #1

'Możesz także używać sprzętowego układu UART
'Wtedy jednak dla programowego układu UART należy określić inne
kończówki portów.
'Na przykład wydrukujemy sobie coś
'Print B

'Jeśli nie chcesz używać inwertera poziomów jakim jest np. MAX-232
'możesz dodać parametr ,INVERTED:
'Open "comd.0:300,8,n,1,inverted" For Input As #2
'Wtedy poziomy logiczne są odwracane i nie potrzebna jest ich
dodatkowa inwersja.
'Jednak w tym wypadku długość przewodów transmisyjnych musi być
krótsza.

End

```

CONFIG

Instrukcja CONFIG w połączeniu z nazwą urządzenia, używana jest do konfiguracji urządzeń sprzętowych. Lista wszystkich możliwych form instrukcji jest przedstawiona poniżej:

CONFIG 1WIRE
CONFIG ACI
CONFIG ADC
CONFIG ATEMU
CONFIG BCCARD
CONFIG CLOCK
CONFIG COM1
CONFIG COM2
CONFIG DEBOUNCE
CONFIG DATE
CONFIG GRAPHLCD
CONFIG I2CDELAY
CONFIG INTx
CONFIG KBD
CONFIG KEYBOARD
CONFIG LCD
CONFIG LCDBUS
CONFIG LCDMODE
CONFIG LCDPIN
CONFIG PORT
CONFIG PS2EMU
CONFIG RC5
CONFIG SCL
CONFIG SDA
CONFIG SERIALIN
CONFIG SERIALIN1
CONFIG SERIALOUT
CONFIG SERIALOUT1
CONFIG SERVOS
CONFIG SPI
CONFIG TCPIP
CONFIG TIMER0
CONFIG TIMER1
CONFIG TIMER2
CONFIG WATCHDOG
CONFIG WAITSUART
CONFIG X10

CONFIG 1WIRE

Przeznaczenie:

Konfiguruje sposób podłączenia magistrali 1Wire do mikrokontrolera.

Składnia:

CONFIG 1WIRE = *końcówka_portu*

gdzie:

końcówka_portu nazwa końcówki portu, będąca magistralą 1Wire.

Opis:

Instrukcja CONFIG 1WIRE pokrywa tylko standardowe ustawienie w opcjach kompilatora.

W podstawowym układzie, tylko jedna końcówka portu może pełnić rolę magistrali 1Wire. Idea działania tej magistrali, opiera się na dołączeniu wielu urządzeń za pomocą tylko jednej linii

sygnałowej (magistrali) i oczywiście masy.

Można jednak stworzyć kilka magistral 1Wire. Informacje na ten temat znajdują się przy opisie instrukcji korzystających z magistrali.

Instrukcje i funkcje dla magistrali 1Wire powodują automatyczne ustawienie wymaganych stanów na odpowiednich bitach w rejestrach DDRx i PORTx. Nie trzeba zatem ustawiać ich stanu w programie.

Uwaga! Ważne jest by linia pełniąca rolę magistrali była „podciągnięta” do szyny zasilania przez rezystor 4,7k lub też przez układ aktywny. Wbudowane w strukturę procesora rezystory nie są w stanie dostarczyć odpowiedniej wartości prądu.

Niektóre z układów podłączonych do magistrali będą jednak wymagać dodatkowo linii zasilającej +5V.

Zobacz także: [1WRESET](#) , [1WREAD](#) , [1WWRITE](#)

Przykład:

```
Config 1wire = PORTB.0    'PORTB.0 będzie pełnił rolę magistrali 1Wire
1wreset                  'Reset magistrali
```

CONFIG ACI *(Nowość w wersji 1.11.6.9)*

Przeznaczenie:

Konfiguruje sposób pracy wbudowanego komparatora analogowego.

Składnia:

CONFIG ACI = ON | OFF , COMPARE = ON | OFF , TRIGGER = OUTPUT | RISING | FALLING

Opis:

Pierwszy parametr **ACI** pozwala na włączenie (**ON**) lub wyłączenie (**OFF**) zasilania komparatora. Standardowo zasilanie jest włączone.

Drugi z parametrów **COMPARE** określa czy wyzwalanie przechwycenia zawartości licznika TIMER1 ma się odbywać także za pomocą zmiany stanu wyjścia komparatora.

Trzeci parametr **TRIGGER** pozwala na określenie jakie zbocze sygnału na wyjściu komparatora powoduje wygenerowanie przerwania. Dostępne są trzy możliwości:

OUTPUT	jeśli przerwanie ma być generowane zarówno przy opadającym i narastającym zboczu,
RISING	jeśli przerwanie ma być generowane po stwierdzeniu narastającego zbocza,
FALLING	jeśli przerwanie ma być generowane po stwierdzeniu opadającego zbocza

Zobacz także: [Komparator analogowy](#)

CONFIG ADC

Przeznaczenie:

Konfiguruje sposób pracy wbudowanego przetwornika A/D.

Składnia:

CONFIG ADC = SINGLE | FREE , PRESCALER = dzielnik | AUTO , REFERENCE = OFF | AVCC | INTERNAL

Opis:

Niektóre kontrolery AVR posiadają wbudowany przetwornik analogowo-cyfrowy. Instrukcja CONFIG ADC pozwala na ustawienie trybu pracy tego przetwornika:

ADC	Tryb pracy. Może być SINGLE lub FREE .
PRESCALER	Określa stopień podziału zegara systemowego. Dozwolone są wartości: 2, 4, 8, 15, 32, 64 lub 128. Podanie AUTO spowoduje, że kompilator dostosuje wartość dzielnika od bieżącej częstotliwości taktowania (ustawionej dyrektywą \$CRYSTAL).
REFERENCE	Niektóre kontrolery (np. M163) mają dodatkowe opcje dotyczące napięcia odniesienia. Można ustawić: OFF , AVCC lub INTERNAL .

Dalszych informacji należy szukać w dokumentacji technicznej kontrolerów.

Zobacz także: **GETADC()**

Asembler:

```
;W trakcie kompilacji jest generowany poniższy kod
In _temp1,ADCSR ;odczytaj ustawienia
Ori _temp1,XXX ;ustaw bieżące
Out ADCSR,_temp1 ;prześlij z powrotem
```

Przykład:

```
'-----
Config Adc = Single , Prescaler = Auto, Reference = Internal
```

CONFIG CLOCK

Przeznaczenie:

Konfiguruje sposób odmierzania czasu rzeczywistego, który jest zwracany przez specjalne zmienne systemowe TIME\$ oraz DATE\$.

Składnia:

CONFIG CLOCK = SOFT | USER [, GOSUB = SECTIC]

Opis:

Pierwszy parametr **CLOCK** definiuje sposób odmierzania czasu przez mikrokontroler. Przy podaniu **SOFT**, zliczaniem czasu zajmuje się specjalna procedura obsługi przerwania. Określenie tego parametru jako **USER** pozwala na utworzenie procedury odmierzania czasu przez użytkownika. Może on skorzystać z rozwiązania programowego lub sprzętowego, np. wykorzystując scalony zegar RTC.

Uwaga! Jeśli wybrano opcję **USER**, tworzone są tylko specjalne zmienne (opis poniżej). Użytkownik musi wtedy skonstruować procedurę obsługi zegara samodzielnie.

Drugi parametr **GOSUB = SECTIC** pozwala na skonstruowanie przez użytkownika dodatkowego podprogramu, który będzie wywoływany co sekundę. Ważne jest by podprogram miał nazwę (rozpocynał się etykietą) **SECTIC** i był zakończony instrukcją **Return**. Podanie tego parametru powoduje również, że wzrasta o 30 bajtów zapotrzebowanie na stos.

Jeśli w programie użyto instrukcji CONFIG CLOCK, kompilator automatycznie generuje specjalne zmienne nazwane: **_sec**, **_min**, **_hour**, **_day**, **_month**, **_year**.

W ten sam sposób są definiowane specjalne zmienne **TIME\$** oraz **DATE\$**. Jest w nich pamiętany zliczany czas i data, w postaci tekstowej. Zobacz opis **TIME\$** oraz **DATE\$** by poznać więcej szczegółów.

Zmienne **_sec**, **_min** i inne definiowane przez tą instrukcję, mogą być odczytywane i modyfikowane przez program. Oczywiście zmiana jednej z nich powoduje także zmianę zmiennych specjalnych **DATE\$** i **TIME\$**.

Kompilator generuje także specjalną procedurę obsługi przerwania, wykonywaną co sekundę. Procedura ta jest generowana dla układów: AT90s8535, M163, M103 oraz M603, lub innych z rodziny AVR; posiadających w swej strukturze czasomierz mogący pracować w trybie asynchronicznym.

Uwaga! Jeśli używany jest kontroler AT90s8535, zegar wykorzystuje licznik TIMER2. Zatem nie może on być używany w tym układzie.

Zmiany w wersji 1.11.7.3

Skonstruowanie biblioteki **DATETIME** pozwoliło na bardziej elastyczną obsługę daty oraz czasu. Kod zawarty w bibliotece udostępnia dwie podstawowe funkcje **DATE** i **TIME**, które w zależności od oczekiwanego rezultatu mogą zwracać dane w postaci liczbowej (ciąg bajtów) lub tekstowej (ciąg znaków).

Dodatkowo udostępniono kilka dodatkowych funkcji:

- **SecOfDay ()**: (rezultat typu Long) Liczba sekund jaka upłynęła od północy. Dla 0:00:00 zwraca 0 ; dla 23:59:59 zwraca 85399,
- **SysSec ()**: (rezultat typu Long) Liczba sekund jaka upłynęła od początku wieku (począwszy od 2000-01-01!). Dla 00:00:00 w dniu 2000-01-01 zwraca 0, a dla 03:14:07 w dniu 2068-01-19 zwraca 2147483647 (przepełnienie zmiennej typu Long),
- **DayOfYear ()**: (rezultat typu Word) Liczba dni jaka upłynęła od 1 stycznia bieżącego roku. Zwraca 0 w dniu 1 stycznia, a 364 w dniu 31 grudnia (lub 365 w roku przestępnym),
- **SysDay ()**: (rezultat typu Word) Liczba dni jaka upłynęła od początku wieku (dzień 2000-01-01!). Zwraca 0 w dniu 2000-01-01, a 36524 w dniu 2099-12-31,
- **DayOfWeek ()**: (rezultat typu Byte) Liczba dni jakie upłynęły od poniedziałku bieżącego tygodnia. Zwraca 0 w poniedziałek, a 6 w niedzielę.

Dane dotyczące czasu i daty są zwracane w postaci tekstowej jak i liczbowej, można zatem obrabiać te dane korzystając z operacji matematycznych.

Dane Typu 1 (ciąg 3 bajtów) i typu 2 (ciąg znaków) mogą być zamienione na odpowiednią wartość liczbową. Potem można dodać lub odjąć prawie dowolną liczbę sekund (dla **SecOfDay ()** i **SysSec ()**) lub dni (dla **DayOfYear ()**, **SysDay ()**), a rezultat skonwerterować ponownie na tekst lub ciąg bajtów.

Zobacz także: **TIME\$**, **DATE\$**, **CONFIG DATE**, **TIME**, **DATE**, **Biblioteka DATETIME**

Asembler:

Wywoływana jest następująca procedura z biblioteki `mcs.lib: _Soft_Clock`. Jest to procedura obsługi przerwania wywoływana przez system przerw co sekundę.

Przykład:

```
'-----
'                                     MEGACLOCK.BAS
'                                     (c) 2000-2001 MCS Electronics
'-----
'Ten przykład pokazuje jak używać specjalnych zmiennych TIME$ i DATE$
'Użycie procesora AT90s8535 (i licznika TIMER2) oraz Mega103 (licznika
TIMER0)
'pozwala na łatwe zaimplementowanie zegara czasu rzeczywistego,
'dołączając zewnętrzny rezonator 32.768KHz do licznika.
'Potrzebny będzie także pewien fragment kodu.

'Ten przykład jest napisany dla płytki STK300 z procesorem M103
Enable Interrupts

'[konfiguracja LCD]
$lcd = &HC000                                'adres dla E i RS
$lcdrs = &H8000                               'adres tylko dla E
```

```

Config Lcd = 20 * 4           'fajny wyświetlacz dla wielkiego
procesora
Config Lcdbus = 4           'działamy w trybie BUS z 4
liniami danych (db4-db7)
Config Lcdmode = Bus        'właśnie go ustawiamy

' [teraz inicjalizacja zegara]
Config Clock = Soft         'O! Takie to jest proste!
' Powyższa instrukcja definiuje procedurę obsługi przerwania TIMER0,
' więc licznik nie może być już używany do innych celów!

' Format daty: MM/DD/YY (miesiąc/dzień/rok)
Config Date = MDY , Separator = /

' Ustawiamy datę
Date$ = "11/11/00"

' Ustawiamy czas, format to HH:MM:SS (24 godzinny)
' Nie można podać 1:2:3!! Może będzie to możliwe w przyszłości.
' Chyba, że zmienisz sobie kod w bibliotece.

Time$ = "02:20:00"

' wyczyść pole LCD
Cls

Do
    Home           'kursor na początek
    Lcd Date$ ; " " ; Time$      'pokaż czas i datę
Loop

' Procedura zegara używa specjalnych zmiennych:
' _day , _month, _year , _sec, _hour, _min
' Wszystkie są typu Byte. Można je modyfikować bezpośrednio:
_day = 1
' Dla zmiennej _year zapisywane są tylko dwie cyfry oznaczające rok.

End

```

CONFIG COM1 (Nowość w wersji 1.11.6.8)

Przeznaczenie:

Konfiguruje rozszerzony układ UART, występujący w niektórych kontrolerach – jak np. M8.

Składnia:

CONFIG COM1 = , SYNCHROME = 0 | 1 , PARITY = NONE | DISABLED | EVEN | ODD , STOPBITS = 1 | 2 , DATABITS = 4 | 6 | 7 | 8 | 9 , CLOCKPOL = 0 | 1

Opis:

SYNCHROME	0 – jeśli działanie jest synchroniczne (standardowo) 1 – jeśli działanie jest nie synchroniczne, Można ustawić:
PARITY	NONE – brak bitu DISABLED – EVEN – nieparzystość ODD – parzystość,
STOPBITS	Liczba bitów stopu 1 lub 2
DATABITS	Ilość bitów danych: 4, 5, 7, 8 lub 9.
CLOCKPOL	Polaryzacja zegara: 0 lub 1.

Uwaga! Nie wszystkie kontrolery posiadają rozszerzony układ UART.

CONFIG COM2 (Nowość w wersji 1.11.6.8)

Przeznaczenie:

Konfiguruje drugi rozszerzony układu UART, występujący w niektórych kontrolerach – jak np. M128.

Składnia:

CONFIG COM2 = , SYNCHROME = 0 | 1 , PARITY = NONE | DISABLED | EVEN | ODD , STOPBITS = 1 | 2 , DATABITS = 4 | 6 | 7 | 8 | 9 , CLOCKPOL = 0 | 1

Opis:

SYNCHROME 0 – jeśli działanie jest synchroniczne (standardowo)
1 – jeśli działanie jest nie synchroniczne,
Można ustawić:
PARITY **NONE** – brak bitu
DISABLED –
EVEN – nieparzystość
ODD – parzystość,
STOPBITS Liczba bitów stopu 1 lub 2
DATABITS Ilość bitów danych: 4, 5, 7, 8 lub 9.
CLOCKPOL Polaryzacja zegara: 0 lub 1.

Uwaga! Nie wszystkie kontrolery posiadają podwójny rozszerzony układ UART.

CONFIG DATE (Nowość w wersji 1.11.7.3)

Przeznaczenie:

Określa sposób formatowania daty dla funkcji daty i czasu.

Składnia:

CONFIG DATE = DMY | MDY | YMD , SEPARATOR = znak_oddzielający

gdzie:

znak_oddzielający znak oddzielający poszczególne składniki daty. Można używać: / , - (minus) i . (kropka).

Opis:

Poniższa tabela pokazuje kilka stosowanych formatów przedstawiających datę.

Tabela 9 Przykłady konfiguracji formatu zapisu daty.

Kraj/Norma	Format	Postać instrukcji
Ameryka	mm/dd/yy	Config Date = MDY, Separator = /
ANSI	yy.mm.dd	Config Date = YMD, Separator = .
W. Brytania/Francja	dd/mm/yy	Config Date = DMY, Separator = /
Niemcy	dd.mm.yy	Config Date = DMY, Separator = .
Włochy	dd-mm-yy	Config Date = DMY, Separator = -
Japonia/Taiwan	yy/mm/dd	Config Date = YMD, Separator = /
USA	mm-dd-yy	Config Date = MDY, Separator = -

Gdy przykładowo mieszkasz w Holandii powinieneś użyć:

Config Date = DMY , Separator = -

spowoduje to, że dla 24 kwietnia 2002 wydrukowana zostanie data 24-04-02. A jeśli mieszkasz w US powinienes użyć:

```
Config Date = MDY , Separator = /
```

co z kolei wydrukuje 04/24/02 dla tej samej daty.

Zobacz także: [CONFIG CLOCK](#) , [Biblioteka DATETIME](#)

Przykład:

```
Config Clock = Soft
Config Date = YMD , Separator = . 'format ANSI
```

CONFIG DEBOUNCE

Przeznaczenie:

Określa czas opóźnienia kolejnych odczytów w instrukcji DEBOUNCE.

Składnia:

```
CONFIG DEBOUNCE = czas
```

gdzie:

czas	opóźnienie w milisekundach, pomiędzy kolejnymi odczytami stanu portu.
------	---

Opis:

Standardowo opóźnienie to wynosi 25ms. Można je zmienić podając inny czas opóźnienia w instrukcji CONFIG DEBOUNCE.

Zobacz także: [DEBOUNCE](#)

Przykład:

```
Config Debounce = 30 'kiedy nie będzie użyta ta opcja, domyślnym
                      'opóźnieniem jest 25ms

Debounce Pind.0 , 0 , Pr , Sub
Debounce Pind.0 , 0 , Pr , Sub
'          ^----- etykieta określająca gdzie skoczyć
'          ^----- skocz gdy P1.0 przejdzie w stan 0
'          ^----- testuj P1.0

'Debounce Pind.0 , 1 , Pr 'a tu aż przejdzie w stan 1
'kiedy Pind.0 przejdzie w stan niski skacze do Pr
'Pind.0 musi przejść w stan wysoki przed ponownym skokiem
'do Pr gdy Pind.0 przejdzie w stan niski

Debounce Pind.0 , 1 , Pr 'nie będzie skoku
Debounce Pind.0 , 1 , Pr 'spowoduje błąd Return without gosub

End

Pr:
  Print "PIND.0 przeszedł w stan niski"
Return
```

CONFIG GRAPHLCD

Przeznaczenie:

Konfiguruje sposób dołączenia graficznego wyświetlacza LCD.

Składnia:

CONFIG GRAPHLCD = *typ* , **DATAPORT** = *port* , **CONTROLPORT** = *port* , **CE** = *pin* , **CD** = *pin* , **WR** = *pin* , **RD** = *pin* , **RESET** = *pin* , **FS** = *pin* , **MODE** = *tryb*

gdzie:

GRAPHLCD	Określa organizację wyświetlacza. Dla wyświetlacza opartego na kontrolerze T6963C, możliwe są tryby: 240 * 64 , 240 * 128 , 160 * 48 , 128 * 128 (w zależności od budowy wyświetlacza!). Dla wyświetlacza opartego na kontrolerze SEDxxxx dostępny jest tylko jeden tryb: 128*64
DATAPORT	Nazwa portu będącego szyną danych. Np. PORTA,
CONTROLPORT	Nazwa portu będącego szyną sterującą. Np. PORTC
CE	Numer końcówki (bitu) w porcie sterującym, pełniącym rolę linii Enable,
CD	Numer końcówki (bitu) w porcie sterującym, pełniącym rolę sygnału Rozkazy/Dane,
WR	Numer końcówki (bitu) w porcie sterującym, pełniącym rolę sygnału wyboru zapisu,
RD	Numer końcówki (bitu) w porcie sterującym, pełniącym rolę sygnału wyboru odczytu,
RESET	Numer końcówki (bitu) w porcie sterującym, pełniącym rolę linii zerowania,
FS	Numer końcówki (bitu) w porcie sterującym, pełniącym rolę linii wyboru czcionki. Linia ta nie występuje w wyświetlaczach opartych na kontrolerze SEDxxxx.
MODE	Liczba kolumn dla wyświetlania tekstu. Gdy podano 8 wtedy przy ilości pikseli równej 240 uzyska się 30 kolumn ($240/8 = 30$), dla wartości 6 - liczba kolumn wynosi 40.

Opis:

Jest to pierwsza wersja kompilatora BASCOM-AVR (1.11.6.4 dla porządku *przyp. tłumacza*) która obsługuje graficzne wyświetlacze LCD. Procedury komunikacji zostały oparte na wyświetlaczu z kontrolerem T6963C, który jest używany w wielu wyświetlaczach. Na razie połączenie wyświetlacza z mikrokontrolerem rozwiązano w trybie PIN. Tak jest! Sterowanie wyświetlaczem to nic innego jak ustawianie odpowiednich stanów na końcówkach.

Połączenia w trybie BUS zostaną wprowadzone w przyszłości. Tryb PIN może być używany w połączeniu z każdym mikrokontrolerem AVR, dlatego został zaimplementowany jako pierwszy.

Używane są następujące połączenia:

Tabela 10 Sposób dołączenia graficznego wyświetlacza LCD.

Końcówka procesora	Końcówka wyświetlacza	Przeznaczenie
PORTA.0 – PORTA.7	DB0 - DB7	Szyna danych
PORTC.5	FS	Wybór czcionki
PORTC.2	CE	Sygnał Enable
PORTC.3	CD	Wybór Rozkazy/DaneLCD
PORTC.0	WR	Zapis do sterownika
PORTC.1	RD	Odczyt ze sterownika
PORTC.4	RESET	Zerowanie

Graficzny wyświetlacz LCD pochodzący z oferty f-my Konrad (<http://www.konrad.de/>) wymaga

dodatkowo ujemnego zasilania potrzebnego do regulacji kontrastu. Dwie baterie 9V i potencjometr wystarczą. Niektóre wyświetlacze mają wyjście Vout, które może być używane do regulacji kontrastu (Vo).

Wyświetlacz T6963C posiada zarówno stronę graficzną jak i tekstową. Mogą one być używane razem. Procedury używają trybu zapisu XOR by wyświetlać tekst i grafikę nałożoną na siebie.

Instrukcje które mogą być używane w połączeniu z graficznym wyświetlaczem LCD:

CLS kasuje zawartość ekranu wyświetlacza.

CLS GRAPH kasuje tylko dane graficzne (stronę graficzną).

CLS TEXT kasuje tylko dane tekstowe (stronę tekstową).

LOCATE rząd , kolumna ustawia kursor na podanej pozycji. Rząd może przyjąć wartości od 1 do 16, kolumna zaś od 1 do 40. Wartości te ściśle zależą od trybu w jakim pracuje wyświetlacz oraz organizacji pikseli wyświetlacza.

CURSOR [ON|OFF] [BLINK|NOBLINK] może być używane tak samo jak w wyświetlaczach alfanumerycznych.

LCD również działa w tak samo jak dla wyświetlacza alfanumerycznego.

SHOWPIC x , y , etykieta wyświetla obrazek na pozycji (x,y), którego dane znajdują się pod adresem określonym podaną etykietą.

PSET x , y , kolor zapala lub kasuje określony piksel. X może przyjąć 0-239 a Y z zakresu 0-63. Gdy kolor jest określony jako 0, piksel jest kasowany. Jeśli kolor jest określony jako 1 piksel jest zapalany.

\$BGF "plik.bgf" wstawia dane graficzne z pliku BGF na bieżącej pozycji.

LINE (x0 , y0) – (x1 , y1) , kolor rysuje linię od punktu (x0,y0) do (x1,y1). Kolor musi być ustawiony jako 0 by skasować linię, a 255 by ją narysować.

Procedury graficzne znajdują się w bibliotece `glib.lib` lub `glib.lbx`.

Możesz połączyć razem FS i RESET, oraz zmienić kod znajdujący się w bibliotece `glib.lib`, co pozwoli na używanie tych końcówek do innych celów.

Zobacz także: **SHOWPIC** , **PSET** , **\$BGF** , **LINE** , **LCD**

Przykład:

```
'-----
'                                     (c) 2001-2002 MCS Electronics
' Przykład obsługi graficznego wyświetlacza LCD z kontrolerem T6963C
'-----

'Sposób podłączenia wyświetlacza LCD:
'końcówka                podłączona do
'1          GND            GND
'2          GND            GND
'3          +5V            +5V
'4          -9V            -9V potencjometr
'5          /WR            PORTC.0
'6          /RD            PORTC.1
'7          /CE            PORTC.2
'8          C/D            PORTC.3
'9          NC             nie podłączone
```



```

'10      RESET      PORTC.4
'11-18   D0-D7      PA
'19      FS         PORTC.5
'20      NC         nie podłączone

$crystal = 8000000
'Najpierw definiujemy, że używany będzie graficzny wyświetlacz LCD
Config Graphlcd = 240 * 128 , Dataport = Porta , Controlport = Portc ,
Ce = 2 , Cd = 3 , Wr = 0 , Rd = 1 , Reset = 4 , Fs = 5 , Mode = 8
'Dataport określa port do jakiego podłączono szynę danych LCD
'Controlport określa który port jest używany do sterowania LCD
'CE, CD itd. to numery bitów w porcie CONTROLPORT
'Na przykład CE = 2 gdyż jest podłączony do końcówki PORTC.2
'Mode = 8 daje 240 / 8 = 30 kolumn , Mode=6 daje 240 / 6 = 40 kolumn

'Deklaracje (y nie używane)
Dim X As Byte , Y As Byte

'Kasujemy ekran wyświetlacza (tekst oraz grafikę)
Cls
'Inne opcje to:
' CLS TEXT   kasuje tylko stronę tekstową
' CLS GRAPH  kasuje tylko stronę graficzną

Cursor Off

Wait 1
'Instrukcja Locate działa normalnie jak dla wyświetlaczy tekstowych
' LOCATE LINIA , KOLUMNA numer linii w zakresie 1-8, kolumny 0 - 30
Locate 1 , 1

'Teraz napiszemy jakiś tekst
Lcd "MCS Electronics"
'i jeszcze jakieś inne w linii 2
Locate 2 , 1 : Lcd "T6963c support"
Locate 16 , 1 : Lcd "A to będzie w linii 16 (najniższej)"

Wait 2

Cls Text

'Użyj nowej instrukcji LINE by narysować pudełko
'LINE(X0 , Y0) - (X1 , Y1), on/off
Line(0 , 0) -(239 , 127) , 255      ' poprzeczne linie
Line(0 , 127) -(239 , 0) , 255
Line(0 , 0) -(240 , 0) , 255        ' górna linia
Line(0 , 127) -(239 , 127) , 255   ' dolna linia
Line(0 , 0) -(0 , 127) , 255       ' lewa linia
Line(239 , 0) -(239 , 127) , 255   ' prawa linia

Wait 2
'Można także rysować linie za pomocą PSET X ,Y , on/off
'parametr on/off to: 0 - skasowanie piksela, inna - postawienie
piksela
For X = 0 To 140
    Pset X , 20 , 255                'narysuj punkt
Next

Wait 2

'A teraz czas najwyższy na obrazek

```

```

'SHOWPIC X , Y , etykieta
'Etykieta określa początek danych gdzie zapisany jest obrazek
Showpic 0 , 0 , Plaatje
Wait 2
Cls Text                               'skasuj tekst
End

'Tutaj znajduje się obrazek
Plaatje:
'Dyrektywa $BGF spowoduje umieszczenie danych obrazka w tym miejscu
$bgf "mcs.bgf"

'Możesz dodać inne obrazki tutaj

```

CONFIG I2CDELAY

Przeznaczenie:

Dyrektywa kompilatora określająca szybkość pracy magistrali I2C.

Składnia:

CONFIG I2CDELAY = wartość

gdzie:

wartość Liczba z zakresu od 1 do 255. Wyższa wartość oznacza zmniejszenie szybkości zegara magistrali I2C.

Opis:

Dla instrukcji używających magistrali I2C jest obliczane opóźnienie, by częstotliwość taktująca procesor nie miała wpływu na szybkość magistrali. Opóźnienie to jest dostosowane tak by każde urządzenie korzystające z I2C mogło działać poprawnie. Gdy pewne jest, że używane będą urządzenia mogące pracować w trybie szybkim (częstotliwość SCL max. 400kHz *przyp. tłumacza*), można ustawić niższą wartość.

Standardowo jest używana wartość 5, co oznacza, że częstotliwość sygnału SCL wynosi 200kHz. Podanie wartości 10 spowoduje zmniejszenie tej częstotliwości do 100kHz.

Zobacz także: **CONFIG SCL** , **CONFIG SDA**

Asembler:

Procedury obsługi magistrali I2C są zawarte w bibliotekach `i2c.lib/i2c.lbx`.

Przykład:

```

Config Sda = PORTB.7           'PORTB.7 będzie pełnił rolę linii SDA
Config I2cdelay = 5

```

Zobacz także przykład zawarty w pliku `i2c.bas` (listing poniżej)

```

'-----
' (c) 1999-2000 MCS Electronics
'-----
' plik: I2C.BAS
' demo: I2CSEND oraz I2CRECEIVE
'-----
Declare Sub Write_eeeprom(byval Adres As Byte , Byval Value As Byte)
Declare Sub Read_eeeprom(byval Adres As Byte , Value As Byte)

Const Addressw = 174           'adres do zapisu
Const Addressr = 175           'adres do odczytu

```

```

Dim B1 As Byte , Adres As Byte , Value As Byte 'definicja bajtów

Call Write_eeprom(1 , 3) 'wpisujemy 3 do komórki 1
                             pamięci EEPROM

Call Read_eeprom(1 , Value) : Print Value 'oczytamy ją
Call Read_eeprom(5 , Value) : Print Value 'i zowu adres 5

'----- zapisujemy dane do kostki PCF8474 -----
I2csend &H40 , 255 'wszystkie wyjścia w stan H
I2creceive &H40 , B1 'odczytanie wejść
Print "Dane odczytane " ; B1 'i wydruk stanu
End

Rem UWAGA! Bit adresowy R/W jest ustawiany automatycznie przez
instrukcje I2csend & I2creceive
Rem oznacza to, że jako adres należy podać tylko adres bazowy układu.

'przykład zapisu bajtu do pamięci AT2404
Sub Write_eeprom(byval Adres As Byte , Byval Value As Byte)
    I2cstart 'warunek startu
    I2cwbyte Addressw 'adres podrzędny
    I2cwbyte Adres 'adres EEPROM
    I2cwbyte Value 'zapisywana wartość
    I2cstop 'warunek stopu
    Waitms 10 'czekaj 10ms
End Sub

'przykład odczytu bajtu z pamięci AT2404
Sub Read_eeprom(byval Adres As Byte , Value As Byte)
    I2cstart 'warunek startu
    I2cwbyte Addressw 'adres podrzędny
    I2cwbyte Adres 'adres EEPROM
    I2cstart 'powtórka warunku startu
    I2cwbyte Addressr 'adres podrzędny (odczyt)
    I2crbyte Value , Nack 'czytanie bajtu
    I2cstop 'warunek stopu
End Sub

```

CONFIG INTx

Przeznaczenie:

Konfiguruje sposób wyzwalania przerwania INTx.

Składnia:

CONFIG INTx = LOW LEVEL | RISING | FALLING [| CHANGE]

gdzie:

x numer przerwania zewnętrznego 0 lub 1, a dla AVR MEGA 4
 – 7.

Opis:

Przerwania z końcówek INTx może być wywoływane na skutek pojawienia się niskiego poziomu logicznego lub też przez detekcję zbocza.

Gdy jest wybrana detekcja stanu logicznego **LOW LEVEL**, przerwanie z linii INT zostanie wygenerowane przez podanie na tą linię stanu niskiego. Utrzymywanie tego stanu spowoduje, że przerwanie będzie generowane raz za razem.

Można także określić, czy przerwanie ma być generowane po stwierdzeniu opadającego zbocza sygnału (**FALLING**) lub też narastającego (**RISING**).

Uwaga! W układach AVR Mega wyzwalanie przerwania linii INT0-INT3 odbywa się na skutek niskiego poziomu (**LOW LEVEL**) i nie jest on konfigurowalny.

Zmiany w wersji 1.11.6.9

Stosując niektóre z nowszych układów serii Mega można także użyć **CHANGE**. Przerwanie będzie wtedy wywoływane po stwierdzeniu dowolnego zbocza sygnału.

Uwaga! Tryb ten jest dostępny tylko dla niektórych końcówek przerywających. Dla przykładu układ ATmega128 pozwala na taką konfigurację tylko dla **INT4-INT7**.

Przykład:

```
'-----  
'przykład dla ATmega103  
  
Config Int4 = Low Level
```

CONFIG KBD

Przeznaczenie:

Definiuje nazwę portu, który używa instrukcja GETKBD().

Skład:

CONFIG KBD = PORTx , DEBOUNCE = czas [, DELAY = czas_op]

gdzie:

PORTx	nazwa portu wejściowego użytego do podłączenia klawiatury, np. PORTB lub PORTD.
czas	czas pomiędzy kolejnymi odczytami stanu klawisza
czas_op	czas opóźnienia w milisekundach jaki zostanie wprowadzony po wykryciu naciśniętego klawisza.

Opis:

Za pomocą funkcji **GETKBD** można odczytywać stan prostej klawiatury matrycowej dołączonej wprost do portu mikrokontrolera. Nazwa portu musi być określona właśnie instrukcją CONFIG KBD.

Jako dodatkową opcję można skonfigurować klawiaturę o matrycy posiadającej 6 rzędów:

CONFIG KBD = PORTx , DEBOUNCE = czas , ROWS = 6 , ROW5 = PIND.6 , ROW6 = PIND.7

Powyższa instrukcja określa, że rząd 5 jest podpięty do końcówki PIND.6, a 7 rząd do PIND.7.

Uwaga! Można podać tylko **ROWS = 6**. Inne wartości nie będą działać.

Podanie opcjonalnego parametru **DELAY=czas_op** spowoduje, że po wykryciu naciśnięcia klawisza funkcja GETKBD wstrzyma działanie programu na czas jaki został podany. Parametr ten może być użyteczny w tych sytuacjach gdzie funkcja GETKBD jest wykonywana w pętli. W takim wypadku mogą o sobie dać znać szumy styków czy zakłócenia elektrostatyczne, co w rezultacie da fałszywe odczyty. Czas opóźnienia około 100 ms powinien wyeliminować ten problem.

Zobacz także: **GETKBD()**

CONFIG KEYBOARD

Przeznaczenie:

Konfiguruje działanie funkcji GETATKBD() oraz określa końcówki portów do których podłączono klawiaturę komputera PC AT.

Składnia:

CONFIG KEYBOARD = PINx.y , **DATA** = PINx.y , **KEYDATA** = tabela_kodów

gdzie:

KEYBOARD Końcówka portu do której dołączony będzie sygnał CLOCK klawiatury.
DATA Końcówka portu do której dołączony będzie sygnał DATA klawiatury.
KEYDATA Etykieta określająca adres tablicy przeliczeniowej kodów klawiszy.

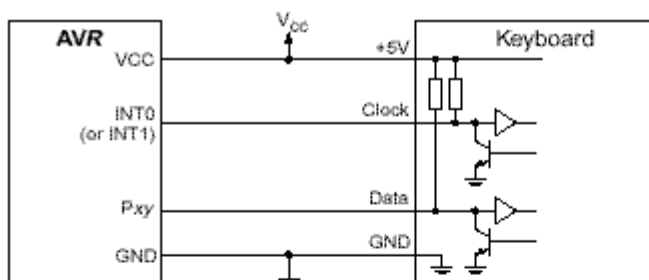
Opis:

Klawiatura PCAT nie przesyła kodów ASCII naciśniętego klawisza, tylko zwraca numer umowny tego klawisza (Na przykład klawisz **Esc** zwraca kod 1. *przyp. tłumacza*). Dlatego w instrukcji CONFIG KEYBOARD należy podać adres (poprzez podanie etykiety) tablicy przeliczeniowej, zapisanej w instrukcjach DATA.



Uwaga! Instrukcja GETATKBD() zwraca także kody klawiszy z Shift-em. Klawisze specjalne nie są rozpoznawane.

Klawiatura PCAT jest podłączana za pomocą tylko czterech linii: Clock, Data, GND oraz Vcc. Krótka informacja o wyprowadzeniach jest przedstawiona poniżej. Jest to fragment pochodzący z informacji technicznych firmy Atmel. Końcówka INT0 lub INT1 jest tylko przykładowym miejscem podłączenia linii zegarowej. Każda końcówka ustawiona jako wejście może pełnić tą rolę.

Program przykładowy opisany w nocie aplikacyjnej dostępnej na stronach WWW firmy Atmel, działa w oparciu o przerwania. Dla języka BASCOM BASIC zmieniono ją tak, aby nie korzysta z systemu przerwań.



AT Keyboard Connector Pin Assignments

AT Computer		
Signals	DIN41524, Female at Computer, 5-pin DIN 180°	6-pin Mini DIN PS2 Style Female at Computer
Clock	1	5
Data	2	1
nc	3	2,6
GND	4	3
+5V	5	4
Shield	Shell	Shell

Rysunek 18 Sposób dołączenia klawiatury do mikrokontrolera.

Zobacz także: GETATKBD()

CONFIG LCD

Przeznaczenie:

Ustawia typ wyświetlacza LCD, pokrywając ustawiony typ w opcjach kompilatora.

Składnia:

CONFIG LCD = *typ_LCD*

gdzie:

typ_LCD typ dołączonego wyświetlacza. W języku BASCOM BASIC zdefiniowano następujące typy:

- 40 * 4,
- 20 * 4,
- 20 * 2,
- 16 * 4,
- 16 * 2 (wartość domyślna),
- 16 * 1,
- 16 * 1a.

Opis:

Standardowo jest zdefiniowany wyświetlacz 16 * 2 (chyba że zostało to zmienione w opcjach kompilatora), więc jeśli używany jest taki typ wyświetlacza, można pominąć instrukcję CONFIG LCD.

Wyświetlacz typu 16 * 1a jest odpowiednikiem wyświetlacza dwie linie po 8 znaków, w którym pierwszy znak drugiej linii zaczyna się pod adresem 8.

Przykład:

```
Config Lcd = 40 * 4
Lcd "Witam!"      'coś piszemy
Fourthline        'wybieramy linię 4
Lcd "4"           'i drukujemy 4
End
```

CONFIG LCDBUS

Przeznaczenie:

Określa sposób komunikacji z wyświetlaczem LCD podłączonym w trybie BUS.

Składnia:

CONFIG LCDBUS = *ilość_bitów*

gdzie:

ilość_bitów 4 dla transmisji 4-bitowej, 8 dla transmisji 8-bitowej.

Opis:

Instrukcji tej powinno się używać w połączeniu z dyrektywą \$LCD.

Gdy komunikacja z wyświetlaczem LCD odbywa się za pomocą systemowej szyny danych (tryb BUS), standardowo używana jest transmisja 8 bitowa. Aby przełączyć tryb komunikacji na 4-bitowy, należy użyć tej instrukcji.

W trybie 4-bitowym wykorzystywane są tylko końcówki danych od DB7 do DB4.

Zobacz także: CONFIG LCD , CONFIG LCDMODE

Przykład:

```
$lcd = &HC000      'adres sygnału E i RS
$lcdrs = &H8000     'adres sygnału E
```

```
Config Lcdbus = 4      'komunikacja 4 bitowa
Lcd "Tu jestem!"
```

CONFIG LCDMODE

Przeznaczenie:

Konfiguruje sposób komunikacji z wyświetlaczem LCD, pokrywając domyślne ustawienia w opcjach kompilatora.

Składnia:

CONFIG LCDMODE = PORT | BUS

Opis:

Sterowanie wyświetlacza LCD w języku BASCOM BASIC może się odbywać za pomocą wybranych portów mikrokontrolera lub też przez szynę danych dostępną w systemach z zewnętrzną pamięcią RAM.

W trybie **PORT** (domyślnym, 4-bitowym) można określić, które z końcówek procesora mają służyć do komunikacji i sterowania. Można używać dowolnej kombinacji tych końcówek (w ramach portów!). Jest to bardzo elastyczne rozwiązanie, gdyż można użyć końcówek nie wykorzystywanych przez inne urządzenia oraz dostosować połączenia na płytce, by były one prostsze. Z drugiej strony zwiększa się ilość kodu jaki potrzebny jest do sterowania wyświetlaczem.

W trybie **BUS** do sterowania wyświetlacza używana jest szyna danych i adresowa przeznaczona do komunikacji z zewnętrzną pamięcią RAM (XRAM). Sterowaniem liniami RS i E zajmuje się wtedy dekodery adresów. Należy wtedy określić dwa adresy, do których zapis będzie powodował wysłanie kodu sterującego lub danej. W tym trybie wszystkie linie danych są dołączone do szyny danych wyświetlacza, chyba że zostało to zmienione instrukcją **CONFIG LCDBUS**.

Adresy sterujące należy ustawić dyrektywami \$LCD oraz \$LCDRS.

Zobacz także: **CONFIG LCD** , **\$LCD** , **\$LCDRS**

Przykład:

```
Config Lcdmode = Port      'w pliku raportu będzie informacja o trybie
pracy
Config Lcdbus = 4          'komunikacja 4 bitowa
Lcd "Witaj!"
```

CONFIG LCDPIN

Przeznaczenie:

Pokrywa ustawienia dotyczące sposobu podłączenia wyświetlacza.

Składnia:

CONFIG LCDPIN = PIN , DB4 = pin , DB5 = pin , DB6 = pin , DB7 = pin , E = pin , RS = pin

CONFIG LCDPIN = PIN , PORT = port , E = pin , RS = pin

gdzie:

pin nazwa końcówki portu pełniąca przypisaną funkcję.

port nazwa portu do którego podłączono szynę danych wyświetlacza (komunikacja 8-bitowa).

Opis:

Można za pomocą tej instrukcji lokalnie zmienić sposób dołączenia wyświetlacza LCD pokrywając ustawienia w menu [Compiler | Settings](#).

Zmiany począwszy od wersji 1.11.6.9

Drugi format stosuje się gdy korzystamy z transmisji 8 bitowej w trybie PORT. W tym wypadku wszystkie linie szyny danych wyświetlacza dołączone są do odpowiednich końcówek portu.

Uwaga! Ważne jest by zachować kolejność bitów! Np. PORTx.0 - DB0, PORTx.1 - DB1 itd.

Zobacz także: **CONFIG LCD**

Przykład:

```
Config Lcdpin = PIN , DB4=PORTB.1, DB5=PORTB.2 , DB6=PORTB.3 ,  
DB7=PORTB.4 , E=PORTB.5 , RS=PORTB.6  
' instrukcja musi być umieszczona w jednej linii
```

CONFIG PORT, CONFIG PIN

Przeznaczenie:

Ustawia kierunek działania portu lub jego końcówki.

Składnia:

```
CONFIG PORTx = tryb  
CONFIG PINx.y = tryb
```

gdzie:

tryb

Możliwe jest podanie: **INPUT** gdy port (końcówka) ma być wejściem lub **OUTPUT** gdy port (końcówka) ma być wyjściem. Można także użyć wartości bitowej aby szybko skonfigurować poszczególne końcówki portu.

Opis:

Aby zmienić tryb pracy portu lub końcówki należy użyć instrukcji CONFIG PORT lub CONFIG PIN.

Najlepszą metodą na ustawienie kilku linii portu jako wejścia lub wyjścia, jest użycie stałej liczbowej w instrukcji CONFIG PORT. Jedynka na odpowiedniej pozycji oznacza, że ta końcówka będzie wyjściem, zero zaś że wejściem. Na przykład podanie wartości **&B00001111** spowoduje, że 4 pierwsze linie portu (7-4) będą wejściami, a 4 następne (3-0) wyjściami.

Uwaga! Instrukcje ustawiają jedynie kierunek działania linii portu, ustawiając odpowiednio rejestr DDRx. Aby skorzystać z możliwości podciągania (pull-up) podczas pracy portu jako wejście, należy w programie odpowiednio ustawić stan bitów w rejestrze PORTx.

Uwaga! Niektóre końcówki pełnią jeszcze dodatkową rolę, np. końcówka PORTB.3 jest także sygnałem OC1. Żeby poprawnie pełniła ona funkcję sygnału OC1, musi być ustawiona jako WYJŚCIE (OUTPUT)!

Zobacz także: Opis portów: **Port B** , **Port D**

Przykład:

```
'-----  
'                                     (c) 2000 MCS Electronics  
'-----  
'  plik: PORT.BAS  
'  demo: PortB oraz PortD  
'-----  
  
Dim A As Byte , Count As Byte  
  
'Używamy portu B jako wyjścia  
Config PortB = Output
```



```

A = Portb                                'odczytujemy rejestr portu B
A = A And Portb
A = Pinb

Print A                                  'drukujemy

Portb = 10                               'wpisujemy do rejestru portu 10
Portb = Portb And 2

Set Portb.0                              'ustawiamy bit 0 w porcie B na 1
Bitwait Pinb.0 , Set                     'czekamy aż będzie ustawiony(1)

Incr Portb

'A teraz lightshow z cyklu Zagraj Światłem (płytką STK200)
Count = 0
Do
  Incr Count
  Portb = 1
  For A = 1 To 8
    Rotate Portb , Left                  'przesuwamy bity w lewo
  Next
  'wykonujemy to samo ale bez instrukcji pętli For..Next
  Portb = 1
  Rotate Portb , Left , 8
Loop Until Count = 10
Print "Gotowe"

'Porty w kontrolerach AVR mają specjalny rejestr w którym ustawiany
'jest kierunek poszczególnych jego końcówek:

'DDRB = &B11110000 'spowoduje to że portb1.0,portb.1,portb.2 i
'portb.3 będą wejściami.

'Jeśli chcesz używać portu jako wejście najpierw ustaw
'zera w rejestrze DDRx! Dopiero potem możesz odczytywać stan końcówek
PINx
'Jeśli chcesz używać portu jako wyjście wpisz do DDRx jedynki.
'Możesz wtedy ustawiać stan końcówek PORTx

End

```

CONFIG RC5

Przeznaczenie:

Zmienia domyślne ustawienia dla linii wejściowej odbiornika RC5.

Składnia:

CONFIG RC5 = pin [, TIMER = 2]

gdzie:

pin

nazwa końcówki portu podłączonej z wyjściem odbiornika podczerwieni.

Opis:

Parametr **TIMER=2** jest opcjonalny. Powoduje on, że do generowania wymaganych opóźnień podczas odbierania danych będzie wykorzystany licznik TIMER2 (w procesorach w których występuje!) zamiast domyślnego TIMER0.

Instrukcja ta pokrywa domyślne ustawienia w zakładce [Options | Compiler Settings](#), które są

zapisywane w plikach .CFG poszczególnych projektów.

Zobacz także: [GETRC5](#)

Przykład:

```
Config Rc5 = Pind.5      'PORTD.5 będzie końcówką wejściową
sygnałów RC5
```

CONFIG SCL

Przeznaczenie:

Pokrywa domyślne ustawienia dotyczące linii SCL interfejsu I2C.

Składnia:

CONFIG SCL = *końcówka_portu*

gdzie:

końcówka_portu nazwa końcówki portu pełniącego rolę sygnału SCL.

Opis:

Gdy sposób dołączenia magistrali I2C zmienia się w zależności od projektu, można używać tej instrukcji by lokalnie zmienić domyślne ustawienia z menu [Options | Compiler](#). W ten sposób będziesz pamiętał jakie końcówki portu zostały użyte w bieżącym projekcie, i nie trzeba będzie za każdym razem zmieniać ustawień w opcjach kompilatora.

W środowisku BASCOM-AVR ustawienia te są zapisywane do pliku konfiguracji projektu.

Zobacz także: [CONFIG SDA](#) , [CONFIG I2CDELAY](#)

Przykład:

```
Config Scl = PORTB.5      'PORTB.5 będzie linią SCL
```

CONFIG SDA

Przeznaczenie:

Pokrywa domyślne ustawienia dotyczące linii SDA interfejsu I2C.

Składnia:

CONFIG SDA = *końcówka_portu*

gdzie:

końcówka_portu nazwa końcówki portu pełniącego rolę sygnału SDA.

Opis:

Gdy sposób dołączenia magistrali I2C zmienia się w zależności od projektu, można używać tej instrukcji by lokalnie zmienić domyślne ustawienia z menu [Options | Compiler](#). W ten sposób będziesz pamiętał jakie końcówki portu zostały użyte w bieżącym projekcie, i nie trzeba będzie za każdym razem zmieniać ustawień w opcjach kompilatora.

W środowisku BASCOM-AVR ustawienia te są zapisywane do pliku konfiguracji projektu.

Zobacz także: [CONFIG SCL](#) , [CONFIG I2CDELAY](#)

Przykład:

```
Config Sda = Portb.7      'PORTB.7 będzie linią SDA

'Zobacz także przykład dotyczący magistrali I2C.
```

CONFIG SERIALIN

Przeznaczenie:

Konfiguruje sprzętowy układ UART by używał bufora wejściowego.

Składnia:

CONFIG SERIALIN = BUFFERED , SIZE = *rozmiar*

gdzie:

rozmiar

liczba określająca ile bajtów SRAM przeznaczyć na bufor wejściowy.

Opis:

Gdy w programie użyta zostanie instrukcja CONFIG SERIALIN, automatycznie zostaną stworzone następujące zmienne:

<code>_RS_HEAD_PTR0</code>	bajt – wskaźnik, określający gdzie znajduje się pierwszy jeszcze nie przetworzony a odebrany bajt,
<code>_RS_TAIL_PTR0</code>	bajt – wskaźnik określający gdzie znajduje się ostatni jeszcze nie przetworzony a odebrany bajt,
<code>_RS232INBUF0</code>	tablica bajtów pełniąca rolę wejściowego bufora kołowego.

Zobacz także: CONFIG SERIALOUT

Asembler:

Wywoływana jest procedura z biblioteki MCS.LIB: `_GotChar`. Jest to procedura obsługi przerwania, wywoływana po każdym odebraniu znaku. Gdy nie ma miejsca w buforze, odebrany znak nie jest umieszczony w buforze. Dlatego bufor musi być opróżniany cyklicznie, podczas odczytywania portu normalnymi instrukcjami `INKEY()` lub `INPUT`.

Ponieważ procedura `_GotChar` wykorzystuje przerwanie URXC, nie jest możliwe używanie tego źródła przerwania w programie użytkownika. Chyba, że zmodyfikuje on procedurę `_GotChar` znajdującą się w bibliotece.

Przykład:

```
'-----
'                               RS232BUFFER.BAS
'                               (c) 2000-2002, MCS Electronics
' Przykład ten pokazuje różnicę pomiędzy pracą UART z buforem i bez
' bufora.
'-----
$crystal = 4000000
$baud = 9600

'Najpierw spróbuj skompilować program, z umieszczoną w komentarzu
'poniższą linią
Config Serialin = Buffered , Size = 20

'definiujemy zmienne
Dim Name As String * 10

'Włączenie przerwań nie jest wymagane dla normalnej pracy układu UART
'Więc najpierw umieść ją jako komentarz, dla testów
Enable Interrupts

Print "Start"
Do
    'odczytamy znak z UART
    Name = Inkey()
```

```

If Err = 0 Then                                'była jakiś znak?
  Print Name                                    'wydrukujemy go
End If

Wait 1                                          'czekaj 1 sekundę
Loop

'Powinieneś zaobserwować efekty gdy powoli będziesz wprowadzał znaki
'przez terminal. Wszystkie wprowadzone znaki powinny być odegnane.
'Teraz jeśli w szybkim tempie będziesz wprowadzał znaki, powinno dać
'się zauważyć, że niektóre z nich będą gubione.

'TERAZ USUŃ ZNAKI KOMENTARZA Z OPISANYCH LINII.
'Skompiluj i uruchom program ponownie.
'Teraz wszystkie wprowadzone znaki będą odbierane przez procedurę
'obsługi przerywania i wprowadzane do bufora transmisji.
'W ten sposób żaden ze znaków, przy szybkim ich wprowadzaniu, nie
'powinien być zgubiony.
'Więc jeśli szybko wpiszesz abcdefg, będą one wydrukowane jeden po
'drugim z 1 sekundowym opóźnieniem.

```

CONFIG SERIALIN1 *(Nowość w wersji 1.11.6.8)*

Przeznaczenie:

Konfiguruje drugi sprzętowy układ UART by używał bufora wejściowego.

Składnia:

CONFIG SERIALIN1 = BUFFERED , SIZE = *rozmiar*

gdzie:

rozmiar liczba określająca ile bajtów SRAM przeznaczyć na bufor wejściowy.

Opis:

Gdy w programie użyta zostanie instrukcja CONFIG SERIALIN1, automatycznie zostaną stworzone następujące zmienne:

<code>_RS_HEAD_PTR1</code>	bajt – wskaźnik, określający gdzie znajduje się pierwszy jeszcze nie przetworzony a odebrany bajt,
<code>_RS_TAIL_PTR1</code>	bajt – wskaźnik określający gdzie znajduje się ostatni jeszcze nie przetworzony a odebrany bajt,
<code>_RS232INBUF1</code>	tablica bajtów pełniąca rolę wejściowego bufora kołowego.

Uwaga! Instrukcja działa tylko w kontrolerach posiadających dwa układy UART.

Zobacz także: CONFIG SERIALOUT1

Asembler:

Wywoływana jest procedura z biblioteki `MCS.LIB: _GotChar1`. Jest to procedura obsługi przerywania, wywoływana po każdym odebrany znaku. Gdy nie ma miejsca w buforze, odebrany znak nie jest umieszczony w buforze. Dlatego bufor musi być opróżniany cyklicznie, podczas odczytywania portu normalnymi instrukcjami `INKEY()` lub `INPUT`.

Ponieważ procedura `_GotChar1` wykorzystuje przerywanie `URXC1`, nie jest możliwe używanie tego źródła przerywania w programie użytkownika. Chyba, że zmodyfikuje on procedurę `_GotChar1` znajdującą się w bibliotece.

Przykład:

```
'-----
```

```

'               RS232BUFFER.BAS
'               (c) 2000-2002, MCS Electronics
' Przykład ten pokazuje różnicę pomiędzy pracą UART z buforem i bez
' bufora.
' Działa tylko z układami posiadającymi dwa sprzętowe układy UART!!!
'-----
$regfile = "m161def.dat"
$crystal = 4000000
$baud1 = 9600

'Najpierw spróbuj skompilować program, z umieszczoną w komentarzu
'poniższą linią
Config Serialin1 = Buffered , Size = 20

'definiujemy zmienne
Dim Name As String * 10

Open "com2:" For Binary As #1
'Włączenie przerwań nie jest wymagane dla normalnej pracy układu UART
'Więc najpierw umieść ją jako komentarz, dla testów
Enable Interrupts

Print "Start"
Do
    'odczytamy znak z UART
    Name = Inkey(#1)

    If Err = 0 Then
        Print #1 , Name
    End If

    Wait 1
Loop
Close #1

'Powinieneś zaobserwować efekty gdy powoli będziesz wprowadzał znaki
'przez terminal. Wszystkie wprowadzone znaki powinny być odegnane.
'Teraz jeśli w szybkim tempie będziesz wprowadzał znaki, powinno dać
'się zauważyć, że niektóre z nich będą gubione.

'TERAZ USUŃ ZNAKI KOMENTARZA Z OPISANYCH LINII.
'Skompiluj i uruchom program ponownie.
'Teraz wszystkie wprowadzone znaki będą odbierane przez procedurę
'obsługi przerwania i wprowadzane do bufora transmisji.
'W ten sposób żaden ze znaków, przy szybkim ich wprowadzaniu, nie
'powinien być zgubiony.
'Więc jeśli szybko wpiszesz abcdefg, będą one wydrukowane jeden po
'drugim z 1 sekundowym opóźnieniem.

```

CONFIG SERIALOUT

Przeznaczenie:

Konfiguruje sprzętowy układ UART by używał bufora wyjściowego.

Składnia:

CONFIG SERIALOUT = BUFFERED , SIZE = rozmiar

gdzie:

rozmiar

liczba określająca ile bajtów SRAM przeznaczyć na bufor wyjściowy.

Opis:

Gdy w programie użyta zostanie instrukcja CONFIG SERIALOUT, automatycznie zostaną stworzone następujące zmienne:

_RS_HEAD_PTRW0	bajt – wskaźnik, określający gdzie znajduje się pierwszy jeszcze nie przetworzony a odebrany bajt,
_RS_TAIL_PTRW0	bajt – wskaźnik określający gdzie znajduje się ostatni jeszcze nie przetworzony a odebrany bajt,
_RS232OUTBUF0	tablica bajtów pełniąca rolę wyjściowego bufora kołowego.

Zobacz także: CONIFG SERIALIN

Asembler:

Wywoływana jest procedura z biblioteki MCS.LIB: _CheckSendChar. Jest to procedura obsługi przerwania, wywoływana gdy rejestr transmisji jest pusty.

Ponieważ używane jest przerwanie UDRE, nie jest możliwe korzystanie przez program użytkownika tego źródła przerwania.

Gdy do wysyłania znaków używana jest instrukcja PRINT, jest włączane źródło przerwania UDRE. Pozwala to na wysyłanie znaków z bufora przez procedurę _CheckSendChar.

Przykład:

```
'-----  
'                                     RS232BUFFEROUT.BAS  
'                                     (c) 2000-2002 MCS Electronics  
' Przykład pokazuje jak używać bufora wyjściowego transmisji  
' szeregowej.  
'-----  
  
$baud = 9600  
$crystal = 4000000  
  
'ustawienie bufora transmisji dla wysyłanych znaków o długości 20  
znaków  
Config Serialout = Buffered , Size = 20  
  
'Jest szczególnie ważne by włączyć globalny system przerw, gdyż  
używane jest przerwanie UDRE  
Enable Interrupts  
  
Print "Witaj świecie!"  
Do  
    Wait 1  
    'UWAGA! Procedura obsługi przerwania spowoduje zwolnienie działania  
    instrukcji opóźniających  
    Print "Test"  
Loop  
  
End
```

CONFIG SERIALOUT1 (Nowość w wersji 1.11.6.8)

Przeznaczenie:

Konfiguruje drugi sprzętowy układ UART by używał bufora wyjściowego.

Składnia:

CONFIG SERIALOUT1 = BUFFERED , SIZE = rozmiar

gdzie:

<i>rozmiar</i>	liczba określająca ile bajtów SRAM przeznaczyć na bufor wyjściowy.
----------------	--

Opis:

Gdy w programie użyta zostanie instrukcja CONFIG SERIALOUT1, automatycznie zostaną stworzone następujące zmienne:

<code>_RS_HEAD_PTRW1</code>	bajt – wskaźnik, określający gdzie znajduje się pierwszy jeszcze nie przetworzony a odebrany bajt,
<code>_RS_TAIL_PTRW1</code>	bajt – wskaźnik określający gdzie znajduje się ostatni jeszcze nie przetworzony a odebrany bajt,
<code>_RS232OUTBUF1</code>	tablica bajtów pełniąca rolę wejściowego bufora kołowego.

Uwaga! Instrukcja działa tylko w kontrolerach posiadających dwa układy UART.

Zobacz także: [CONIFG SERIALIN1](#)

Asembler:

Wywoływana jest procedura z biblioteki MCS.LIB: `_CheckSendChar1`. Jest to procedura obsługi przerwania, wywoływana gdy rejestr transmisji jest pusty.

Ponieważ używane jest przerwanie UDRE1, nie jest możliwe korzystanie przez program użytkownika tego źródła przerwania.

Gdy do wysyłania znaków używana jest instrukcja PRINT, jest włączane źródło przerwania UDRE. Co pozwala na wysyłanie znaków z bufora przez procedurę `_CheckSendChar1`.

Przykład:

```
'-----
'
'               RS232BUFFEROUT.BAS
'               (c) 2000-2002 MCS Electronics
' Przykład pokazuje jak używać bufora wyjściowego transmisji
' szeregowej.
' Działa tylko w kontrolerach posiadających podwójny układ UART
'-----

$regfile = "m16ldef.dat"
$baud1 = 9600
$crystal = 4000000

'ustawienie bufora transmisji dla wysyłanych znaków o długości 20
'znaków
Config Serialout1 = Buffered , Size = 20

Open "com2:" For Binary As #1
'Jest szczególnie ważne by włączyć globalny system przerwania, gdyż
używane jest przerwanie UDRE1
Enable Interrupts

Print #1 , "Witaj świecie!"
Do
  Wait 1
  'UWAGA! Procedura obsługi przerwania spowoduje zwolnienie działania
  instrukcji opóźniających
  Print #1 , "Test"
Loop
Close #1

End
```

CONFIG SERVOS

Przeznaczenie:

Określa ile serwomechanizmów będzie kontrolowanych przez mikrokontroler.

Składnia:

CONFIG SERVOS = *liczba* , **SERVO1** = *końcówka* [, **SERVO***n* = *końcówka*] , **RELOAD** = *czas*

gdzie:

<i>liczba</i>	liczba serwomechanizmów,
<i>końcówka</i>	nazwa końcówki portu sterującego odpowiednim serwomechanizmem,
<i>czas</i>	czas opóźnienia pomiędzy kolejnymi przerwaniem licznika TIMER0.

Opis:

Serwomechanizmy wymagają impulsów o zmiennym wypełnieniu by były sterowane poprawnie. Instrukcja CONFIG SERVOS inicjalizuje specjalną tablicę bajtów, w której zapisywany będzie czas, przez który odpowiednia końcówka będzie włączona. Instrukcja konfiguruje także procedurę przerwania licznika TIMER0.

Dla przykładu instrukcja:

```
Config Servos = 2 , Servo1 = Portb.0 , Servo2 = Portb.1 , Reload = 100
```

spowoduje że:

- procedura przerwania będzie wywoływana co 100 μ s.
- zostanie utworzona tablica nazwana SERVO() składająca się z dwóch komórek: SERVO(1) i SERVO(2).

Ustawiając odpowiednią wartość w jednej z komórek, kontrolowany jest czas podawania stanu wysokiego na skojarzonej końcówce. Po upływie tego czasu zawartość komórki zostanie wyzerowana.

Podana końcówka portu musi być wcześniej ustawiona jako wyjście. Przykładowo w ten sposób:

```
Config Pinb.0 = Output
```

Uwaga! Instrukcja używa przerwania licznika TIMER0.

Przykład:

```
'-----  
'                                     (c) 2001 MCS Electronics  
'                               servo.bas demonstruje instrukcję SERVO  
'-----  
  
'Serwomechanizmy wymagają impulsów sterujących dla poprawnego  
działania  
'Stosując instrukcję CONFIG SERVOS ustalamy iloma serwomechanizmami  
'chcemy sterować, oraz które końcówki będą do tego używane  
'Maksymalnie można używać 16 serwomechanizmów  
'Instrukcja SERVO potrzebuje jednego bajta z pamięci SRAM i  
'wykorzystuje licznik TIMER0. Dlatego używanie licznika TIMER0 nie  
jest możliwe.  
'Wartość podawana jako parametr RELOAD, określa czas pomiędzy  
'poszczególnymi przerwaniem (w mikrosekundach).  
Config Servos = 2 , Servo1 = Portb.0 , Servo2 = Portb.1 , Reload = 100  
'Używamy 2 serwomechanizmy sterowane z precyzją 100 mikrosekund
```



```

'Musimy określić sposób działania portu
Config Portb = Output

'Na koniec musimy także włączyć system przerwań
Enable Interrupts

'Tablica Servo() jest tworzona automatycznie. Można jej użyć do
'sterowania czasem włączenia
Servo(1) = 10                                'włącz na 1000 mikrosekund
Servo(2) = 20                                'włącz na 2000 mikrosekund

Dim I As Byte
Do
  For I = 1 To 20
    Servo(1) = I
    Waitms 1000
  Next

  For I = 20 To 1 Step -1
    Servo(1) = I
    Waitms 1000
  Next
Loop

End

```

CONFIG SPI

Przeznaczenie:

Konfiguruje tryb pracy interfejsu SPI.

Składnia (emulacja programowa):

CONFIG SPI = SOFT, DIN = pin, DOUT = pin, SS = [pin | NONE], CLOCK = pin

gdzie:

pin nazwa końcówki pełniącej rolę sygnału.

Składnia (interfejs sprzętowy):

CONFIG SPI = HARD, DATA ORDER = LSB | MSB, MASTER = YES | NO, POLARITY = HIGH | LOW, PHASE = 0 | 1, CLOCKRATE = 4 | 16 | 64 | 128, NOSS = 0 | 1

Opis:

Procesory AVR mają specjalny interfejs szeregowy służący do komunikacji między procesorami tej serii. Interfejs ten jest wykorzystywany także podczas programowania pamięci kodu i EEPROM.

W języku BASCOM BASIC oprócz wbudowanego **sprzętowego interfejsu SPI**, istnieje także jego emulacja programowa. Gdy wybrano tą opcję: **SPI = SOFT**; należy określić, które końcówki będą pełnić rolę sygnałów interfejsu SPI.

W trybie sprzętowym: **SPI = HARD**; interfejs używa przydzielonych mu fabrycznie końcówek:

DIN	Wejście danych lub MISO. Końcówka PINB.0
DOUT	Wyjście danych lub MOSI. Końcówka PORTB.1
SS	Slave Select. Końcówka PORTB.2.
CLOCK	Użycie NONE pozwala na wyłączenie sygnału SS. Taktowanie. Końcówka PORTB.3

Dodatkowo można określić następujące cechy interfejsu:

DATA ORDER	Wybiera czy najpierw ma być przesyłany bit najstarszy (MSB) czy najmłodszy (LSB).
MASTER	Definiuje czy interfejs ma być urządzeniem nadrzędnym

	(MASTER) czy podporządkowanym (SLAVE).
POLARITY	Wybiera czy stan linii CLOCK ma być ustawiona w stan wysoki (HIGH) czy niski (LOW), gdy interfejs nie jest aktywny.
PHASE	Przeczytaj notę katalogową, by dowiedzieć się więcej o ustawieniach związanych z tą opcją.
CLOCKRATE	Wybiera stopień podziału sygnału zegarowego, który jest potem dostarczony jako sygnał taktujący interfejs SPI. Jeśli wybrany będzie dzielnik 4 to przy częstotliwości kwarcu równej 4MHz, sygnał taktujący będzie miał częstotliwość 1MHz.
NOSS	Podanie jako parametru 1 pozwala na wyłączenie generowania sygnału SS w trybie Master.

Standardowo sprzętowy interfejs SPI jest ustawiony na: DATA ORDER = MSB , POLARITY = HIGH, MASTER = YES, PHASE = 0, CLOCKRATE = 4.

Gdy zostanie użyte samo **Config SPI = Hard**, bez dodatkowych parametrów, interfejs SPI zostanie tylko włączony. Pracować będzie wtedy w trybie **SLAVE**, z wyzerowanymi bitami **CPOL** i **CPH.s**

W trybie **SPI = HARD**, instrukcja **SPIINIT** ustawia odpowiednie końcówki następująco:

```
Sbi DDRB, 7           ; SCK jako wyjście
Cbi DDRB, 6           ; MISO jako wejście
Sbi DDRB, 5           ; MOSI jako wyjście
```

W trybie **SPI = SOFT**, instrukcja **SPIINIT** ustawia stan końcówek (podanych jako przykład!) w taki sposób:

```
Sbi PORTB, 5          ;ustawienie zatrzasku w stan H (nieaktywny) SS
Sbi DDRB, 5           ;końcówka odpowiadająca za SS jako wyjście
Cbi PORTB, 4          ;ustawienie linii zegara w stan niski
Sbi DDRB, 4           ;jako wyjście

Cbi PORTB, 6          ;ustawienie linii MOSI w stan L
Sbi DDRB, 6           ;jako wyjście MOSI
Cbi DDRB, 7           ;MISO jako wejście
Ret
```

Gdy występuje potrzeba zaadresowania jednego z wielu urządzeń Slave przez programowy SPI, potrzebne są końcówki, które będą wybierać/aktywować odpowiednie układy. W tym wypadku należy podać parametr **SS = NONE**. Oznacza to także, że przed każdą transmisją wysyłaną przez SPI, należy ustawić odpowiednią końcówkę w stan niski, a po wysłaniu bajtu ustawić ją z powrotem w stan wysoki.

Sprzętowy SPI także posiada taką opcję. Gdy parametr **NOSS** jest ustawiony na 1 powoduje to, że końcówka SS nie jest ustawiana w stan niski gdy SPI rozpoczyna transmisję. Trzeba wtedy samodzielnie ustawić odpowiednią końcówkę SS lub inną wykorzystywaną w tym celu. Po zakończeniu pracy SPI należy ją z powrotem ustawić w stan 1, by dezaktywować wybrany układ.

Wszystkie procedury dotyczące SPI działają w trybie Master. Przykład 2 pokazuje jak stworzyć programowy SPI pracujący w trybie Slave. W katalogu **SAMPLES**, powinieneś znaleźć przykłady programów gdzie sprzętowy SPI pracuje w trybie Master i Slave.

Zobacz także: **SPIIN** , **SPIOUT** , **SPIINIT**

Przykład1:

```

Config Spi = SOFT, DIN = PINB.0 , DOUT = PORTB.1, SS = PORTB.2, CLOCK
= PORTB.3
Dim var As Byte

Spiinit           'Inicjalizacja SPI oraz portów.
Spiout var , 1    'transmisja 1 bajtu

```

Przykład2:

```

'-----
'                               SPI-SOFTSLAVE.BAS
'                               (c) 2002 MCS Electronics
'Przykład pokazujący jak zaimplementować programowy SPI w trybie SLAVE
'-----

'Niektóre układy jak na przykład 90s2313 nie posiadają portu SPI.
'Wszystkie procedury wbudowane w BASCOM BASIC pracują w trybie master.
'Przykład ten pokazuje jak pracować w trybie Slave używając 90s2313

'używamy 90s2313
$regfile = "2313def.dat"

'XTAL
$crystal = 4000000

'szybkość UART
$baud = 19200

'definicja stałych używanych przez SPI Slave
Const _softslavespi_port = Portd      'używamy portu D
Const _softslavespi_pin = Pind        'używamy PIND jako rejestr
wejściowy
Const _softslavespi_ddr = Ddrd        'kierunek działania portu D

Const _softslavespi_clock = 5         'pD.5 jako CLOCK
Const _softslavespi_miso = 3          'pD.3 jako MISO
Const _softslavespi_mosi = 4          'pd.4 jako MOSI
Const _softslavespi_ss = 2            'pd.2 jako SS
'Wszystkie końcówki mogą być wybrane dowolnie, jedynie jako
'końcówka SS musi być użyta INT0.
'W 90s2313 jest to PD.2

'PD.3(7), MISO musi być wyjściem
'PD.4(8), MOSI
'Pd.5(9) , Clock
'PD.2(6), SS /INT0

'definicja używanej biblioteki
$lib "spislave.lbx"
'która procedura jest używana
$external _spisoftslave

'Używamy przerwania INT0 by określić kiedy układ jest wybierany
On Int0 Isr_sspi Nosave
'włączamy przerwanie
Enable Int0
'INT0 będzie reagować na zmianę z wysokiego na niski stanu końcówki
Config Int0 = Falling
'teraz włączamy globalny system przerwań
Enable Interrupts

```

```

'
Dim _ssspdr As Byte 'to będzie rejestr SPI SLAVE
SPDR
Dim _ssspif As Bit 'bit przerwania SPI
Dim Bsend As Byte , I As Byte , B As Byte 'inne przykładowe
zmienne

_ssspdr = 0 'wpisujemy 0, najpierw Master
wysyła dane
Do
    If _ssspif = 1 Then
        Print "Odebrano: " ; _ssspdr
        Reset _ssspif
        _ssspdr = _ssspdr + 1 'wyślemy to następnym razem
    End If
Loop

```

CONFIG TIMER0

Przeznaczenie:

Ustala sposób działania sprzętowego licznika-czasomierza TIMER0.

Składnia:

CONFIG TIMER0 = COUNTER , EDGE= RISING | FALLING

CONFIG TIMER0 = TIMER , PRESCALE= 1 | 8 | 64 | 256 | 1024

Opis:

Licznik-czasomierz TIMER0 jest pojedynczym 8 bitowym licznikiem. Dalsze informacje zawarto w temacie [Licznik-czasomierz TIMER0](#).

Kiedy TIMER0 jest skonfigurowany do pracy w charakterze licznika (parametr **TIMER0 = COUNTER**) wtedy należy określić parametr **EDGE**:

EDGE	Można określić czy licznik będzie zwiększany wraz z pojawieniem się narastającego zbocza sygnału (RISING) lub opadającego (FALLING).
-------------	--

Gdy TIMER0 jest skonfigurowany do pracy w charakterze czasomierza (parametr **TIMER0 = TIMER**) wtedy należy także określić parametr **PRESCALE**:

PRESCALE	Źródłem impulsów zwiększających licznik jest przebieg taktujący mikroprocesor. Przebieg ten zanim trafi na wejście licznika TIMER0 jest dzielony w preskalerze, którego stopień podziału może być wybrany spośród podanych wartości: 1 , 8, 64, 256 lub 1024.
-----------------	---

Jeśli w programie użyto instrukcji CONFIG TIMER0, tryb pracy licznika jest zapamiętywany przez kompilator oraz ustawiany jest rejestr sprzętowy **TCCR0**.

Licznik może być w dowolnej chwili zatrzymany instrukcją **STOP TIMER0**. Ponowne uruchomienie licznika następuje przez wydanie polecenia **START TIMER0**, wtedy to do rejestru **TCCR0** jest wpisywana zapamiętana wartość. Dlatego przed użyciem w programie instrukcji **START TIMER** lub **STOP TIMER**, należy skonfigurować działanie licznika.

Przykład:

```

'Na początku musisz skonfigurować pracę licznika Timer0, by pracował
'jako licznik lub czasomierz

'Skonfigurujemy go jako licznik. Na początek

```

```

'musisz także określić czy zmiana stanu licznika ma się odbywać po
'stwierdzeniu narastającego lub opadającego zbocza

Config Timer0 = Counter , Edge = Rising
'Config Timer0 = Counter , Edge = Falling
'usuń komentarz w powyższej linii by zmiana następowała wraz z
'opadającym zboczem.

'By odczytać lub zapisać dane do licznika użyjemy specjalnej zmiennej
'systemowej
'Najpierw ją zerujemy
Tcnt0 = 0

Do
    Print Tcnt0
Loop Until Tcnt0 >= 10
'Po zliczeniu 10 impulsów program przestanie drukować stan licznika

'Teraz skonfigurujemy go jako czasomierz

'Sygnałem wejściowym w tym przypadku będzie przebieg zegarowy
'podzielony w preskalerze przez 8,64,256 lub 1024
'Parametr PRESCALE akceptuje tylko poniższe wartości: 1,8,64,256 lub
1024
Config Timer0 = Timer , Prescale = 1

'Licznik jest teraz uruchamiany automatycznie
'możesz go jednak zatrzymać w dowolnej chwili, o tak:
Stop Timer0

'Teraz licznik jest zatrzymany
'by wznowić działanie licznika użyjemy:
Start Timer0

'znowu możemy odczytywać zawartość licznika
Print Tcnt0

'kiedy licznik się przepełni, ustawiona zostanie flaga TOV0 w
'rejestrze TIFR (będzie miała stan 1 logicznej)

'Można użyć poniższej instrukcji do wywoływania procedury obsługi
przerwania licznika.
Set Tifr.1

'Poniższy fragment pokaże jak używać licznika TIMER0 do generowania
'przerwań zegarowych
'Fragment ten jest umieszczony w bloku komentarza '( )

' (
    'konfigurujemy licznik by używał dzielnika 1024
    Config Timer0 = Timer , Prescale = 1024

    'Definiujemy procedurę obsługi przerwania licznika
    On OvF0 Tim0_Isr
    'możesz także użyć TIMER0 jako OVFO, jest to to samo

    'Możesz załadować do licznika wartość odpowiadającą liczbie
    'impulsów po zliczeniu których licznik się przepełni.
    'UWAGA! W tym wypadku instrukcja LOAD musi być umieszczona także

```

```

'na początku procedury obsługi przerwania!
'Load Timer0, 100          'załaduj liczbę impulsów

Enable Timer0              'włączamy przerwania licznika
Enable Interrupts          'oraz globalny system przerw

Do
  'Twój program będzie tutaj
Loop

'Poniższy kod będzie wykonany gdy licznik się przepełni
Tim0_Isr:
  'Load Timer0, 100          'ponownie załaduj liczbę impulsów
  Print "*";
  Return

')

End

```

CONFIG TIMER1

Przeznaczenie:

Ustala sposób działania sprzętowego licznika-czasomierza TIMER1.

Składnia:

**CONFIG TIMER1 = COUNTER , EDGE = RISING | FALLING , NOICE CANCEL = 0 | 1 ,
CAPTURE EDGE = RISING | FALLING , COMPARE A = SET | CLEAR | TOGGLE | DISCONNECT ,
COMPARE B = SET | CLEAR | TOGGLE | DISCONNECT**

**CONFIG TIMER1 = TIMER , PRESCALE = 1 | 8 | 64 | 256 | 1024 , CAPTURE EDGE =
RISING | FALLING , COMPARE A = SET | CLEAR | TOGGLE | DISCONNECT , COMPARE B =
SET | CLEAR | TOGGLE | DISCONNECT**

**CONFIG TIMER1 = PWM , PWM = 8 | 9 | 10 , COMPARE A PWM = CLEAR UP | CLEAR
DOWN | DISCONNECT , COMPARE B PWM = CLEAR UP | CLEAR DOWN | DISCONNECT**

Opis:

Licznik-czasomierz TIMER1 jest 16-bitowym zliczającym w górę. Zobacz opis wybierając temat [Licznik-czasomierz TIMER1](#).

Jeśli TIMER1 jest skonfigurowany do pracy w charakterze licznika (**TIMER1 = COUNTER**) wtedy należy określić następujące parametry:

EDGE	Można wybrać czy licznik ma być zwiększany po wykryciu narastającego zbocza sygnału (RISING) lub opadającego zbocza (FALLING).
CAPTURE EDGE	Można określić czy przechwytywać zawartość licznika do rejestru <i>INPUT CAPTURE</i> wraz z narastającym (RISING) lub opadającym zboczem (FALLING) sygnału pojawiającego się na końcówce ICP.
NOICE CANCEL	Można określić czy licznik ma być odporny na drobne zakłócenia sygnału wejściowego. Wpisanie 1 włącza tą opcję.

Gdy TIMER1 jest skonfigurowany do pracy w charakterze czasomierza (parametr **TIMER1 = TIMER**) wtedy należy także określić parametr **PRESCALE**:

PRESCALE	Źródłem impulsów zwiększających licznik jest przebieg taktujący mikroprocesor. Przebieg ten zanim trafi na wejście licznika TIMER1 jest dzielony w preskalerze, którego stopień podziału może być wybrany spośród podanych wartości: 1 , 8, 64, 256 lub
-----------------	---

1024.

Licznik-czasomierz TIMER1 posiada także dwa rejestry – COMPARE A i COMPARE B, służące do porównywania zawartości licznika. Kiedy zawartość licznika zgadza się z zawartością któregoś licznika, wywoływane zostanie przerwanie OC1A lub OC1B.

COMPAREx	Określa tryb pracy rejestrów COMPAREA lub COMPAREB:
SET	bit OC1x zostanie ustawiony,
CLEAR	bit OC1x zostanie skasowany,
TOGGLE	stan bitu OC1x zostanie zmieniony na przeciwny
DISCONNECT	porównywanie nie będzie przeprowadzane i bit OC1x nie będzie zmieniany.

Uwaga! Nie wszystkie kontrolery posiadają rejestr COMPARE B (OC1B) licznika TIMER1.

Można także używać licznika-czasomierza TIMER1 jako generatora impulsów PWM (o regulowanym wypełnieniu *przyp. tłumacza*):

PWM	Określa rozdzielczość generatora w bitach. Możliwe są wartości 8, 9 i 10 bitów.
COMPARE A PWM	Określa czy licznik ma zliczać w górę (CLEAR UP) czy w dół (CLEAR DOWN) po stwierdzeniu zgodności stanu licznika z zawartością rejestru COMPAREx.

Użycie **COMPARE A**, **COMPARE B**, **COMPARE A PWM** lub **COMPARE B PWM** spowoduje ustawienie odpowiedniej końcówki jako wyjście. Gdy nie jest to pożądane można dodać argument **NO_OUTPUT**, co spowoduje że tryb pracy końcówki nie zostanie zmieniony.

Dla przykładu: **COMPARE A NO_OUTPUT** , **COMPARE A PWM NO_OUTPUT**

Przykład:

```
'-----
'                                     TIMER1.BAS
'-----

Dim W As Word

'TIMER1 jest 16-bitowym licznikiem
'Ten program przykładowy pokazuje jak skonfigurować licznik TIMER1

'Tak samo jak przy liczniku TIMER0 , może służyć jako licznik lub
'czasomierz
'Najpierw skonfigurujemy licznik jako czasomierz, zliczający impulsy
'zegarowe

'Wewnętrzny zegar może być podzielony przez preskaler przez 1,8,64,256
'lub 1024
Config Timer1 = Timer , Prescale = 1024

'Możesz odczytywać i zapisywać dane z licznika TIMER1, za pomocą
'specjalnej zmiennej systemowej:
W = Timer1
Timer1 = W

'By używać licznika jako licznik impulsów zewnętrznych musisz
'określić, jakie zbocze sygnału będzie powodowało zwiększenie
'licznika.
Config Timer1 = Counter , Edge = Falling
'Config Timer1 = Counter , Edge = Rising

'Możesz także określić czy ma następować zatrzaśnięcie stanu licznika
```

```

'w rejestrze INPUT CAPTURE
'W tym celu za pomocą parametru CAPTURE EDGE =, należy określić
'czy zatrzaśnięcie ma nastąpić po stwierdzeniu zbocza narastającego
'lub opadającego sygnału na końcówce ICP

Config Timer1 = Counter , Edge = Falling , Capture Edge = Falling
'Config Timer1 = Counter , Edge = Falling , Capture Edge = Rising

'Aby pozbyć się mogących się pojawić zakłóceń sygnału wyzwalającego
'można użyć parametru NOICE CANCEL
Config Timer1 = Counter , Edge = Falling , Capture Edge = Falling ,
Noice Cancel = 1

'aby odczytać zawartość rejestru CAPTURE:
W = Capture1
'czy wpisać wartość do tego rejestru:
Capture1 = W

'Licznik ma także dwa rejestry porównawcze A i B
'Kiedy zawartość licznika będzie taka sama jak rejestru COMPARE,
'ustawiany będzie specjalny znacznik. Przy czym zmianę tą można
'skonfigurować następująco:
' SET , OC1X przybierze stan 1
' CLEAR , OC1X przybierze stan 0
' TOGGLE , nastąpi zmiana stanu OC1X
' DISCONNECT, OC1X nie będzie zmieniany w żaden sposób
Config Timer1 = Counter , Edge = Falling , Compare A = Set , Compare B
= Toggle

'Aby zapisać i odczytać rejestr COMPAREA lub COMPAREB należy posłużyć
'się zmiennymi systemowymi:
Compare1a = W
W = Compare1a

'Na koniec można także wykorzystać TIMER1 jako generator PWM. Licznik
'może mieć wtedy 8, 9 lub 10 bitów.
'Można ustawić czy licznik ma zliczać w górę (UP) czy w dół (DOWN)
'po stwierdzeniu zgodności „wskazań” licznika i rejestru COMPARE.
'W trybie tym są dwa rejestry COMPARE!
Config Timer1 = Pwm , Pwm = 8 , Compare A Pwm = Clear Up , Compare B
Pwm = Clear Down

'Tu także można zmieniać zawartość licznika COMPARE za pomocą
specjalnych zmiennych systemowych.
Compare1a = 100
Compare1b = 200

'albo dla polepszenia czytelności:
Pwm1a = 100
Pwm1b = 200

End

```

CONFIG TIMER2

Przeznaczenie:

Określa tryb pracy licznika TIMER2.

Składnia: (dla AT90s8535)

CONFIG TIMER2 = TIMER | PWM , ASYNC = ON | OFF , PRESCALE = 1 | 8 | 32 | 64 | 128 |

256 | 1024 , COMPARE = CLEAR | SET | TOGGLE | DISCONNECT , PWM = ON | OFF , COMPARE PWM = CLEAR UP | CLEAR DOWN | DISCONNECT , CLEAR TIMER = 0 | 1

Składnia: (dla M103)

CONFIG TIMER2 = COUNTER | TIMER | PWM , EDGE = FALLING | RISING , PRESCALE = 1 | 8 | 64 | 256 | 1024 , COMPARE = CLEAR | SET | TOGGLE | DISCONNECT , PWM = ON | OFF , COMPARE PWM = CLEAR UP | CLEAR DOWN | DISCONNECT , CLEAR TIMER = 0 | 1

Opis:

Czasomierz TIMER2 ma rozdzielczość 8 bitów. W zależności od konstrukcji konkretnego procesora może pracować także jako licznik.

Oto efekt działania poszczególnych opcji:

EDGE	Można wybrać czy licznik ma być zwiększany po stwierdzeniu narastającego (RISING) lub opadającego (FALLING) zbocza sygnału zewnętrznego. Tylko w trybie COUNTER .
PRESCALE	Gdy licznik pracuje jako czasomierz (TIMER2 = TIMER) wejście licznika jest dołączone do generatora taktującego procesor przez preskalera. Można wtedy wybrać stopień podziału tego preskalera. Dostępne wartości to: 1, 8, 32, 64, 128, 256 i 1024, a dla układu M103 tylko 1, 8, 64, 256 lub 1024.

Licznik TIMER2 posiada także rejestr porównania. Gdy zawartość licznika będzie zgodna z zawartością tego rejestru, stan końcówki OC2 może się zmienić następująco:

COMPARE	Określa jaka ma być reakcja przy zgodności rejestru COMPARE2 i zawartości licznika:
SET	końcówka OC2 zostanie ustawiona w stan wysoki,
CLEAR	końcówka OC2 zostanie ustawiona w stan niski,
TOGGLE	stan końcówki OC2 zostanie zmieniony na przeciwny
DISCONNECT	porównywanie nie będzie przeprowadzane i stan końcówki OC2 nie będzie zmieniany.

Licznika TIMER2 może pracować także jako 8 bitowy generator PWM (**TIMER2 = PWM**). Można określić czy licznik ma zliczać w górę lub w dół po stwierdzeniu zgodności rejestru COMPARE2 z zawartością licznika.

COMPARE PWM	Określa tryb pracy generatora PWM. Można ustawić: CLEAR UP lub CLEAR DOWN
--------------------	---

Uwaga! Instrukcja CONFIG TIMER2 musi być zapisana w jednej linii. Nie wszystkie jej składniki muszą występować.

Przykład:

```
Dim W As Byte

Config Timer2 = Timer , Async = 1 , Prescale = 128
On Timer2 MyIsr

Enable Interrupts
Enable Timer2

Do
Loop

MyIsr:
    'procesor będzie „wskakiwać” co sekundę przy kwarcu 32768KHz
```

Return

```
'Można odczytywać zawartość licznika jak normalną zmienną COUNTER2 lub  
TIMER2  
W = Timer2  
Timer2 = W
```

CONFIG WAITSUART

Przeznaczenie:

Dyrektywa kompilatora określająca opóźnienie jakie ma być wstawione po wysłaniu ostatniego znaku przez programowy UART.

Składnia:

CONFIG WAITSUART = wartość

gdzie:

wartość liczba z zakresu 1-255. Większa wartość określa dłuższe opóźnienie w ms.

Opis:

Gdy programowy interfejs UART używany jest do komunikacji z szeregowym wyświetlaczem LCD, może być potrzebne krótkie opóźnienie po wysłaniu ostatniego znaku, by wyświetlacz miał czas na przetworzenie danych.

Przykład: Zobacz przykład dla instrukcji **OPEN**.

CONFIG WATCHDOG

Przeznaczenie:

Konfiguruje opóźnienie zadziałania układu WATCHDOG.

Składnia:

CONFIG WATCHDOG = czas

gdzie:

czas czas w milisekundach po jakim układ WATCHDOG uaktywni sygnał reset procesora. Możliwe są wartości: 16 , 32, 64 , 128 , 256 , 512 , 1024 oraz 2048.

Opis:

Mikrokontrolery AVR mają wbudowany **układ WATCHDOG**, który zabezpiecza przed skutkami zakłóceń w działaniu programu, mogącymi się pojawić przez czynniki zewnętrzne.

Działanie układu WATCHDOG polega na zliczaniu impulsów zegarowych przez specjalny licznik. Gdyby program główny w odpowiednim czasie nie wyzerował licznika, układ WATCHDOG uaktywni sygnał reset mikrokontrolera. Spowoduje to rozpoczęcie działania programu od początku. W ten sposób można zapobiec niekontrolowanemu skokom czy przeciwdziałać zapętleniom się programu.

Do kasowania licznika służy instrukcja RESET WATCHDOG. Przy wartości 2048 czas pomiędzy kolejnymi instrukcjami RESET WATCHDOG jest dość długi i wynosi 2 sekundy.

Zobacz także: **START WATCHDOG** , **STOP WATCHDOG** , **RESET WATCHDOG**

Przykład:

```
'-----  
'                      (c) 2000 MCS Electronics
```

```

' WATCHD.BAS demonstruje działanie układu Watchdog
'-----
Config Watchdog = 2048          'reset wystąpi po 2048ms
Start Watchdog                  'uruchamiamy układ watchdog
Dim I As Word

For I = 1 To 10000
  Print I                       'drukujemy coś
  'Reset Watchdog
  'Pętla For Next nie zakończy się gdyż układ Watchdog zresetuje
  'mikrokontroler.
  'Gdyby instrukcja Reset Watchdog nie została usunięta z pętli
  'wtedy układ Watchdog nie spowodowałby wyzerowania procesora
  'gdyż nie zdążyłby odliczyć ustawionego czasu.
Next

End

```

CONFIG X10 (Nowość w wersji 1.11.7.3)

Przeznaczenie:

Konfiguruje końcówki portu użyte do transmisji protokołem X10.

Składnia:

CONFIG X10 = *końcówka_ZC* , **TX** = *końcówka_TX*

gdzie:

końcówka_ZC nazwa końcówki portu połączonej z wyjściem ZERO CROSS (ZC) modułu TW-523
końcówka_TX nazwa końcówki portu połączonej z końcówką TX modułu TW-523. Końcówka ta służy do przekazywania danych do modułu TW-523.

Opis:

Moduł TW-523 posiada złącze RJ-11 którego końcówki spełniają następujące funkcje:

Nr. końc.	Opis	Połączenie z procesorem
1	Zero Cross	Końcówka wejściowa <i>końcówka_ZC</i> . Należy dodać rezystor podciągający 5,1k.
2	GND	Masa
3	RX	Nie używana
4	TX	Końcówka wyjściowa <i>końcówka_TX</i>

Zobacz także: **X10DETECT** , **X10SEND**

COUNTER, CAPTURE, COMPARE i PWM

Przeznaczenie:

Zmienne te służą do modyfikacji rejestrów sprzętowych liczników.

Składnia:

COUNTER0 = *wartość*
zmienna = **COUNTER0** zawartość licznika TIMER0
COUNTER1 = *wartość*
zmienna = **COUNTER1** zawartość licznika TIMER1

Można używać także **TIMER0** i **TIMER1** zamiast **COUNTER0** i **COUNTER1**.

CAPTURE1 = wartość
zmienna = **CAPTURE1** zawartość rejestru przechwytyującego licznika TIMER1

COMPARE1A = wartość
zmienna = **COMPARE1A** zawartość rejestru COMPARE A licznika TIMER1

COMPARE1B = wartość
zmienna = **COMPARE1B** zawartość rejestru COMPARE B licznika TIMER1

PWM1A = wartość
zmienna = **PWM1A** rejestr COMPARE A licznika TIMER1 (używany w trybie PWM)

PWM1B = wartość
zmienna = **PRM1B** rejestr COMPARE B licznika TIMER1 (używany w trybie PWM)

gdzie:

wartość zmienna typu Byte, Integer lub Word, albo stała liczbowa określająca nową zawartość licznika,
zmienna zmienna do której przekazana będzie aktualna zawartość licznika.

Opis:

Zmienne te są odzwierciedleniem sprzętowych rejestrów 16 bitowych i muszą być traktowane w trochę inny sposób niż normalne zmienne. Jedynym wyjątkiem jest **TIMER0/COUNTER0**, gdyż jest to 8 bitowy rejestr.

Ponieważ po uruchomieniu liczniki pracują non-stop, to przy wpisywaniu nowej wartości do licznika, powstać może błąd polegający na zmianie zawartości licznika pomiędzy wpisami kolejnych połówek do 16 bitowego rejestru. Eliminacja tego błędu polega na wykorzystaniu następujących mechanizmów:

Kiedy procesor wpisuje dane do starszej części rejestru (MSB) 16 bitowego, zapisywane dane trafiają najpierw do rejestru tymczasowego. Następnie procesor zapisuje młodszą część rejestru (LSB), jednocześnie ze starszą częścią z rejestru tymczasowego. W wyniku tego cała wartość 16 bitowa trafia do właściwego rejestru w tym samym czasie.

Podczas operacji odczytu sytuacja jest podobna, z tym że kolejność odczytu rejestrów jest odwrotna – najpierw młodszą część, potem starszą.

Cały mechanizm odczytu i zapisu rejestrów jest obsługiwany przez **BASCOM** automatycznie.

Trzeba zaznaczyć, że dostępność do niektórych rejestrów jest zależna od konstrukcji procesorów AVR. Dokumentacja języka **BASCOM**, przy opisie różnic między sprzętowymi rejestrami, opiera się na właściwościach procesora **AT90s8515**.

CONST

Przeznaczenie:

Deklaruje nazwę stałej.

Składnia:

CONST nazwa = liczba
CONST nazwa = "tekst"
CONST nazwa = wyrażenie

gdzie:

nazwa nazwa pod jaką będzie identyfikowana stała,
liczba wartość liczbowa przypisana stałej,
tekst wartość tekstowa przypisana stałej.
wyrażenie dowolne wyrażenie języka **BASCOM BASIC**

Opis:

Stałe nie używają pamięci przeznaczanej na zmienne, gdyż podczas kompilacji w miejsce nazwy stałej jest przypisywana jej wartość.

Różnice w stosunku do BASCOM-8051

W języku BASCOM-8051 tylko stałe numeryczne mogą być definiowane. Inna jest także składnia instrukcji.

Zobacz także: **ALIAS**

Przykład:

```
Dim Z As String * 10
Dim B As Byte

'utworzymy jakieś stałe
'nie używają one pamięci danych
Const s = "Siemano"           'teraz stałą tekstową
Const A = 5                   'definiujemy stałą liczbową

'a tą na przykład dwójkową
Const B1 = &B1001

'można teraz używać wyrażeń
Const X = (b1 * 3) + 2
Const Ssingle = Sin(1)

End
```

COS() (Nowość w wersji 1.11.6.8)

Przeznaczenie:

Zwraca wartość kosinusa kąta podanego w radianach.

Składnia:

zmienna = **COS**(*liczba*)

gdzie:

<i>zmienna</i>	dowolna zmienna typu single, do której wpisany będzie wynika działania funkcji,
<i>liczba</i>	liczba której wartość kosinusa należy obliczyć.

Opis:

Wszystkie funkcje trygonometryczne używają zapisu wartości kątów w radianach. Aby przeliczyć stopnie na radiany należy użyć funkcji DEG2RAD. Zamianę odwrotną zapewnia instrukcja RAD2DEG.

Zobacz także: **RAD2DEG** , **DEG2RAD** , **ATN** , **SIN**

Przykład: Zobacz temat **Biblioteka FP_TRIG**

COSH() (Nowość w wersji 1.11.6.8)

Przeznaczenie:

Zwraca wartość kosinusa hyperbolicznego kąta podanego w radianach.

Składnia:

zmienna = COSH(liczba)

gdzie:

<i>zmienna</i>	dowolna zmienna typu single, do której wpisany będzie wynik działania funkcji,
<i>liczba</i>	liczba której wartość kosinusa hyperbolicznego należy obliczyć.

Opis:

Wszystkie funkcje trygonometryczne używają zapisu wartości kątów w radianach. Aby przeliczyć stopnie na radiany należy użyć funkcji DEG2RAD. Zamianę odwrotną zapewnia instrukcja RAD2DEG.

Zobacz także: RAD2DEG , DEG2RAD , ATN , SIN , COS , TANH , SINH

Przykład: Zobacz temat **Biblioteka FP_TRIG**

CRC8()

Przeznaczenie:

Oblicza bajt sumy kontrolnej wg algorytmu CRC.

Składnia:

zmienna = CRC8(źródło , ile_bajtów)

gdzie:

<i>zmienna</i>	dowolna zmienna typu Byte, do której wpisany będzie wynik działania funkcji,
<i>źródło</i>	tablica w której zawarte są bajty, których sumę kontrolną trzeba obliczyć,
<i>il_bajtów</i>	ile elementów podanej tablicy należy brać pod uwagę.

Opis:

Suma kontrolna obliczana według algorytmu CRC8, jest używana do kontroli poprawności transmisji. Na przykład urządzenia współpracujące z magistralą 1Wire zwracają jako ostatni bajt numeru ID bajt sumy kontrolnej CRC.

Zobacz także: CHECKSUM , CRC16

Asembler:

Wywoływana jest następująca procedura z biblioteki `mcs.lib: _CRC8`.

Jako parametry procedury jest przekazywany wskaźnik do tablicy w rejestrze Z, oraz w rejestrze R24 ilość bajtów branych pod uwagę. Suma kontrolna jest zwracana w rejestrze R16. Procedura używa R16-R19 i R25.

```
##### X = Crc8(ar(1) , 7)
ldi R24,$07                ;liczba bajtów
ldi R30,$64                ;adres ar(1)
ldi R31,$00                ;załadowany do rejestru Z
rcall _Crc8                ;wywołanie
ldi R26,$60                ;adres X
st X,R16                   ;wpisuje crc8
```

Przykład:

```
Dim Ar(8) As Byte , X As Byte

'inicjalizacja tablicy
Ar(1) = &H2
```

```

Ar(2) = &H1C
Ar(3) = &HB8
Ar(4) = 1
Ar(5) = 0
Ar(6) = 0
Ar(7) = 0

'odczytanie bajtu CRC z tablicy. Skanuj 7 bajtów
X = Crc8(ar(1) , 7)

```

CRC16()

Przeznaczenie:

Oblicza słowo sumy kontrolnej wg algorytmu CRC.

Składnia:

zmienna = CRC16(źródło , ile_bajtów)

gdzie:

zmienna	dowolna zmienna typu Word lub Integer, do której wpisany będzie wynik działania funkcji,
źródło	tablica w której zawarte są bajty, których sumę kontrolną trzeba obliczyć,
il_bajtów	ile elementów podanej tablicy należy brać pod uwagę.

Opis:

Suma kontrolna obliczana według algorytmu CRC16, jest używana do kontroli poprawności transmisji. Na przykład urządzenia współpracujące z magistralą 1Wire zwracają jako ostatni bajt numeru ID bajt sumy kontrolnej CRC. Dla celów transmisji magistralą 1Wire, należy używać funkcji CRC8.

Istnieje wiele różnych algorytmów obliczania sumy kontrolnej CRC16. Dotychczas nie ustalono jednego standardu jej obliczania.

Zobacz także: **CHECKSUM** , **CRC8**

Asembler:

Wywoływana jest następująca procedura z biblioteki `mcs.lib: _CRC16`.

Jako jeden z parametrów procedury jest przekazywany wskaźnik do tablicy w rejestrze X. Liczba bajtów branych pod uwagę jest przekazywana przez stos programowy. Suma kontrolna jest zwracana w rejestrach R16 i R17. Procedura używa R16-R19 i R25.

```

;##### X = Crc16(ar(1) , 7)
Ldi R24,$07                ;liczba bajtów
St -y, R24
Ldi R26,$64                ;adres ar(1)
Ldi R27,$00                ;załaduj do X
Rcall _Crc16               ;wywołaj procedurę
Ldi R26,$60                ;adres zmiennej X
St X+,R16                  ;zapisz młodszy bajt
St X ,R17                  ;i starszy sumy CRC16

```

Przykład:

```

Dim Ar(8) As Byte , X As Word

'inicjalizacja tablicy
Ar(1) = &H2
Ar(2) = &H1C
Ar(3) = &HB8

```

```
Ar(4) = 1
Ar(5) = 0
Ar(6) = 0
Ar(7) = 0
```

```
'odczytanie słowa CRC z tablicy. Skanuj 7 bajtów
X = Crc16(ar(1) , 7)
```

CRYSTAL

Przeznaczenie:

Specjalna zmienna systemowa określająca szybkość pracy programowego urządzenia UART.

Składnia:

CRYSTAL = *wartość* Starsza wersja składni, nie powinna być używana!!
__CRYSTAL*x* = *wartość*

gdzie:

<i>x</i>	numer kanału programowego urządzenia UART,
<i>wartość</i>	liczba określająca szybkość pracy.

Opis:

Dla programowego urządzenia UART można ustawić dowolną szybkość transmisji, bez błędów wynikających z niedopasowania częstotliwości generatora taktującego. Jedynie seria ATtiny22 posiada wewnętrzny generator 1MHz. Częstotliwość ta może ulegać zmianie pod wpływem temperatury lub zmian napięcia zasilającego.

Zawartość zmiennej CRYSTAL może być zmieniona w trakcie działania programu, co spowoduje zmianę szybkości transmisji.

Zmiany wprowadzone do wersji 1.11 i wyższych kompilatora BASCOM.

Nadal można zmieniać szybkość transmisji posługując się zawartością zmiennej CRYSTAL, która jest definiowana automatycznie. Jednakże nazwa tej zmiennej została zmieniona, gdyż jest możliwe otwarcie kilku kanałów programowego urządzenia UART:

__CRYSTAL*x*, gdzie *x* określa numer kanału.

Kiedy w programie zostanie otwarty kanał #1, wtedy zmienna dla tego kanału będzie miała nazwę: **__CRYSTAL1**.

Lepszą jednak metodą na zmianę szybkości transmisji jest używanie rozszerzonej instrukcji **BAUD**. Na przykład:

```
Baud #1 , 2400
```

zmienia szybkość transmisji kanału nr 1 na 2400 bodów.

Gdy w programie użyto rozszerzonej instrukcji **BAUD #**, szybkość transmisji musi zostać określona wcześniej, zanim można będzie korzystać z transmisji tym kanałem. Użycie tej instrukcji spowoduje automatycznie zdefiniowanie zmiennej **__CRYSTAL***x* oraz umieszczenie w niej poprawnej wartości.

Gdy instrukcja **BAUD #** nie występuje, jest przyjmowana wartość domyślna. Oszczędza się tym samym dwie komórki z pamięci SRAM.

Zmienne **__CRYSTAL***x* nie występują na liście zmiennych w raporcie kompilacji, gdyż są to zmienne systemowe.

Uwaga! Zmienna CRYSTAL występująca w starszych wersjach kompilatora została zlikwidowana!

Niektóre wartości przy częstotliwości wewnętrznego zegara 1MHz:

Crystal = 66 'dla 2400 bodów

Crystal = 31 'dla 4800 bodów

Crystal = 14 'dla 9600 bodów

Zobacz także: **OPEN** , **CLOSE** , **BAUD**

Przykład:

```
Dim B As byte

Open "comd.1:9600,8,n,1,inverted" For Output As #1
Print #1 , B
Print #1 , "Transmisja szeregową"
Baud #1, 4800 'teraz użyjemy 4800 bodów
Print #1, "Transmisja szeregową"
__Crystal1 = 255
Close #1

End
```

CPEEK()

Przeznaczenie:

Zwraca zawartość wskazanej komórki pamięci kodu.

Składnia:

zmienna = **CPEEK**(*adres*)

gdzie:

<i>zmienna</i>	zmienna liczbowa której przypisana będzie zawartość komórki,
<i>adres</i>	adres pamięci kodu.

Opis:

Można odczytać zawartość dowolnej komórki pamięci kodu za pomocą funkcji CPEEK. Nie istnieje funkcja CPOKE, gdyż zapis do pamięci kodu nie jest możliwy podczas działania programu.

Wielkość przestrzeni adresowej pamięci kodu jest zależna od typu mikrokontrolera. Pierwsza komórka jest identyfikowana adresem 0.

Zobacz także: **PEEK** , **CPEEKH** , **POKE** , **INP** , **OUT**

Przykład:

```
'-----
'                               (c) 1998-2000 MCS Electronics
'                               PEEK.BAS
'  demonstracja PEEK, POKE, CPEEK, INP oraz OUT
'-----

Dim I As Integer , B1 As Byte

'zrzut pamięci RAM kontrolera
For I = 0 To 31 'AVR-y mają tylko 32 rejestry
    B1 = Peek(i) 'czytamy bajt
    Print Hex(b1) ; " ";
    'Poke I , 1 'można zmieniać zawartość
    pamięci(rejestrów)
Next
Print 'piszemy CR
'Ostrożnie z bezpośrednią manipulacją pamięci RAM!!
```

```

'A teraz podejrzmy sobie pamięć kodu(programu)
For I = 0 To 255

    B1 = Cpeek(i)                                'czytamy bajt z ROM
    Print Hex(b1) ; " ";
Next
'Nie można zapisywać do pamięci ROM!!
'(A szkoda... Powstały by wirusy! przyp. tłumacza)

Out &H8000 , 1                                'zapisujemy 1 do XRAM pod adresem 8000
B1 = Inp(&H8000)                              'a teraz czytamy

```

CPEEKH()

Przeznaczenie:

Zwraca zawartość wskazanej komórki z górnej połówki pamięci kodu układu M103.

Składnia:

zmienna = CPEEKH(*adres*)

gdzie:

<i>zmienna</i>	zmienna liczbowa której przypisana będzie zawartość komórki,
<i>adres</i>	adres pamięci kodu.

Opis:

Funkcja CPEEKH(0) zwraca wartość zapisaną w pierwszej komórce drugiej połówki (drugie 64KB) pamięci kodu dostępnej w kontrolerach M103. Układy M103 posiadają drugie 64KB pamięci kodu, które przy liniowym odwzorowaniu pamięci kodu funkcja CPEEK nie potrafi odczytać.

Uwaga! Funkcja ta powinna być używana tylko dla kontrolerów M103.

Zobacz także: CPEEK , PEEK , POKE , INP , OUT

Przykład:

```

'-----
'                                (c) 1998-2000 MCS Electronics
'                                PEEK.BAS
'  demonstracja PEEK, POKE, CPEEK, INP oraz OUT
'-----
Dim I As Integer , B1 As Byte

'zrzut pamięci RAM kontrolera
For I = 0 To 31                                'AVR-y mają tylko 32 rejestry
    B1 = Peek(i)                                'czytamy bajt
    Print Hex(b1) ; " ";
    'Poke I , 1                                'można zmieniać zawartość
    pamięci(rejestrów)
Next
Print                                           'piszemy CR
'Ostrożnie z bezpośrednią manipulacją pamięci RAM!!

'A teraz podejrzmy sobie pamięć kodu(programu)
For I = 0 To 255

    B1 = Cpeek(i)                                'czytamy bajt z pierwszej
    połówki ROM
    Print Hex(b1) ; " ";

```

```

    B1 = Cpeekh(i)                'czytamy bajt z drugiej
połówki ROM
    Print Hex(b1) ;

Next
'Nie można zapisywać do pamięci ROM!!
'(A szkoda... Powstały by wirusy! przyp. tłumacza)

Out &H8000 , 1                    'zapisujemy 1 do XRAM pod adresem 8000
B1 = Inp(&H8000)                  'a teraz czytamy
Print B1

End

```

CURSOR

Przeznaczenie:

Ustawia stan kursora wyświetlacza LCD.

Składnia:

CURSOR [**ON** | **OFF**] [**BLINK** | **NOBLINK**]

Opis:

Instrukcja pozwala na sterowanie atrybutami kursora sprzętowego wyświetlacza LCD.

Można stosować razem parametry **ON/OFF** i **BLINK/NOBLINK**. W czasie inicjalizacji wyświetlacza LCD stan kursora jest ustawiany na **ON** i **NOBLINK**.

Zobacz także: **DISPLAY** , **LCD**

Przykład:

```

Dim a As Byte

a = 255
Lcd a
Cursor Off      'schowaj kursor
Wait 1          '1 sekunda pauzy
Cursor Blink    'teraz migaj

End

```

DATA

Przeznaczenie:

Służy do umieszczania danych, odczytywanych później instrukcją **READ** w trakcie działania programu.

Składnia:

DATA wartość1 [, wartośćn]

gdzie:

wartość stała numeryczna lub tekstowa.

Opis:

Instrukcja wykorzystywana jest do definiowania ciągu danych, umieszczonych w pamięci kodu lub pamięci EEPROM mikrokontrolera. Dane te można wykorzystać w dowolny sposób. Odczyt danych umieszczonych w pamięci kodu umożliwia instrukcja **READ**. Dane umieszczone w pamięci EEPROM należy odczytywać instrukcją **READEEPROM**.

Dane mogą być dowolnego typu z dostępnych w języku BASCOM BASIC. Dane tekstowe należy umieszczać w cudzysłowie:

```
Data "Ala ma kota" 'ulubione tłumacza :)
```

Aby umieścić w liniach data znak " należy użyć takiego zapisu:

```
Data $34
```

Znak \$ oznacza daną w postaci kodu znaku ASCII. W ten sam sposób można umieszczać dane tekstowe niemożliwe do wprowadzenia bezpośrednio z klawiatury.

Można także umieszczać dane heksadecymalne z przedrostkiem &H i bitowe z &B.

Ponieważ w instrukcjach DATA można umieszczać dane, które mają być zapisane do pamięci EEPROM, dodano dyrektywy kompilatora \$DATA oraz \$EEPROM.

Linie DATA nie mogą być umieszczane pomiędzy blokami wykonywalnego kodu – oprócz tych, które mają trafić do pamięci EEPROM – ponieważ dane tam zawarte są zamieniane na ciąg bajtów. Kiedy więc skompilowany program natrafi na te dane, może to wywołać nieprzewidziane skutki, z zawieszeniem się procesora włącznie.

Najlepszym miejscem na umieszczenie linii DATA, jest koniec programu. Można wpleść dane pomiędzy instrukcje, lecz należy zadbać by program ominął te instrukcje.

Na przykład, to jest poprawny kod:

```
Print "Hello"
Goto jump
Data "test"

Jump:
'ponieważ przeskoczyliśmy linię DATA,
'wszystko jest w porządku
```

Poniższy przykład może sprawiać pewne kłopoty:

```
Dim S As String * 10

Print "Hello"
Restore lbl
Read S
Data "test"
Print S
```

Uwaga! Jeśli użyto instrukcji END, to musi być ona umieszczona **przed** instrukcjami DATA.

Instrukcje ściśle związane z instrukcją DATA używają pary rejestrów R8 i R9, jako wskaźnika danych.

Różnice w stosunku do QBasic-a.

Wartości typu Integer i Word muszą być zakończone znakiem %. Wartości Long, znakiem &, a wartości Single zaś znakiem !.

Zobacz także: READ , RESTORE , \$DATA , \$EEPROM

Przykład:

```
'-----
'                                READDATA.BAS
'                                Copyright 2000 MCS Electronics
'-----
```

```

Dim A As Integer , B1 As Byte , Count As Byte
Dim S As String * 15
Dim L As Long

Restore Dta1
For Count = 1 To 3
    Read B1 : Print Count ; " " ; B1
Next
'ustawiamy wskaźnika
'czytamy trzy dane

Restore Dta2
For Count = 1 To 2
    Read A : Print Count ; " " ; A
Next
'ustawiamy wskaźnik
'czytamy dwie dane

Restore Dta3
Read S : Print S
Read S : Print S

Restore Dta4
Read L : Print L
'dane typu Long

End

Dta1:
    Data &B10 , &HFF , 10

Dta2:
    Data 1000% , -1%

Dta3:
    Data "Hello" , "World"
'Uwaga! Dane typu Integer (>255 lub <0) muszą być zakończone znakiem %
'Typ danych odczytywanych instrukcją READ i zapisanych w liniach DATA
'musi się zgadzać.

Dta4:
    Data 123456789&
'Uwaga! Dane typu Long muszą być zakończone znakiem &

```

DATE\$

Przeznaczenie:

Specjalna zmienna przechowująca aktualną datę.

Składnia:

```

DATE$ = "mm/dd/yy"
zmienna = DATE$

```

Opis:

Zmienna DATE\$ jest używana w połączeniu z instrukcją **CONFIG CLOCK**.

Instrukcja CONFIG CLOCK używa licznika TIMER0 lub TIMER2, pracującego w trybie asynchronicznym, do generowania przerwań co 1 sekundę. Procedura obsługi tego przerwania zwiększa odpowiednio zawartość zmiennych `_sec`, `_min` oraz `_hour`. Jeśli ten „licznik” się przepełni zostaje zwiększony kolejny „licznik” składający się ze zmiennych `_day`, `_month` oraz `_year`.

Kiedy zawartość zmiennej DATE\$ zostaje przypisana zmiennej tekstowej, wspomniane zmienne „liczników” są najpierw przekazywane zmiennej DATE\$. Gdy wystąpi operacja odwrotna – tj. zostanie zmieniona wartość zmiennej DATE\$, wtedy jej zawartość zostaje rozkodowana i zmieniane

są zmienne „liczników”.

Format zapisu daty jest taki sam jak w dialektach Qbasic/VB. Jedyną różnicą w stosunku do QBasic/VisualBasic jest to, że wszystkie dane muszą być podane gdy przypisywana jest wartość daty. Jest to podyktowane zmniejszeniem ilości potrzebnego kodu. Można to oczywiście zmienić, modyfikując treść biblioteki.

Uwaga! Praca liczników (TIMER0 bądź TIMER2) w trybie asynchronicznym jest możliwa tylko w procesorach M103, AT90s8535, M164 oraz M32(3). Inne procesory nie posiadają tej możliwości.

By używać daty w zapisie europejskim DD-MM-YY, należy użyć biblioteki `eurotimedata.lbx`:

```
$lib "eurotimedata.lbx"
```

Zmiany w wersji 1.11.7.3

Biblioteka EUROTIMEDATE jest obecnie wycofywana. Język BASCOM Basic AVR udostępnia bibliotekę DATETIME która oferuje znacznie więcej możliwości.

Zobacz także: [TIME\\$](#) , [CONFIG CLOCK](#)

Asembler:

Wywoływane są następujące procedury:

- podczas przypisywania wartości zmiennej DATE\$: `_set_date` (wywołuje `_str2byte`)
- podczas odczytywania zmiennej DATE\$: `_make_dt` (wywołuje `_byte2str`)

Przykład:

```
'-----  
'                               MEGACLOCK.BAS  
'                               (c) 2000-2001 MCS Electronics  
'-----  
'Ten przykład pokazuje jak używać specjalnych zmiennych TIME$ i DATE$  
'Użycie procesora AT90s8535 (i licznika TIMER2) oraz Mega103 (licznika  
TIMER0)  
'pozwala na łatwe zaimplementowanie zegara czasu rzeczywistego,  
'dołączając zewnętrzny rezonator 32.768KHz do licznika.  
'Potrzebny będzie także pewien fragment kodu.  
  
'Ten przykład jest napisany dla płytki STK300 z procesorem M103  
Enable Interrupts  
  
'[konfiguracja LCD]  
$lcd = &HC000                                'adres dla E i RS  
$lcdrs = &H8000                               'adres tylko dla E  
Config Lcd = 20 * 4                           'fajny wyświetlacz dla wielkiego  
procesora  
Config Lcdbus = 4                             'działamy w trybie BUS z 4  
liniami danych (db4-db7)  
Config Lcdmode = Bus                          'właśnie go ustawiamy  
  
'[teraz inicjalizacja zegara]  
Config Clock = Soft                           'O! Takie to jest proste!  
'Powyższa instrukcja definiuje procedurę obsługi przerwania TIMER0,  
' więc licznik nie może być już używany do innych celów!  
  
'Format daty: MM/DD/YY (miesiąc/dzień/rok)  
Config Date = MDY , Separator = /  
  
'Ustawiamy datę
```

```

Date$ = "11/11/00"

'Ustawiamy czas, format to HH:MM:SS (24 godzinny)
'Nie można podać 1:2:3!! Może będzie to możliwe w przyszłości.
'Chyba, że zmienisz sobie kod w bibliotece.

Time$ = "02:20:00"

'wyczyść pole LCD
Cls

Do
    Home                                'kursor na początek
    Lcd Date$ ; " " ; Time$            'pokaż czas i datę
Loop

'Procedura zegara używa specjalnych zmiennych:
'_day , _month, _year , _sec, _hour, _min
'Wszystkie są typu Byte. Można je modyfikować bezpośrednio:
_day = 1
'Dla zmiennej _year zapisywane są tylko dwie cyfry oznaczające rok.

End

```

DATE() (Nowość w wersji 1.11.7.3)

Przeznaczenie:

Dokonyje konwersji daty na postać znakową lub na ciąg 3 bajtów przechowujących dzień, miesiąc i rok.

Składnia:

```

bDMR = DATE( ISysSec )
bDMR = DATE( wSysDay )
bDMR = DATE( strData )

strData = DATE( ISysSec )
strData = DATE( wSysDay )
strData = DATE( bDMR )

```

gdzie:

<i>strData</i>	ciąg znaków z zapisaną datą w formacie określonym przez instrukcję CONFIG DATE ,
<i>ISysSec</i>	zmienna typu Long która przechowuje liczbę sekund jaka upłynęła od początku wieku, (SysSec = TimeStamp)
<i>wSysDay</i>	zmienna typu Word która przechowuje liczbę dni jaka upłynęła od początku wieku,
<i>bDMR</i>	nazwa zmiennej typu Byte, w której przechowywany jest bieżący dzień. Zaraz po tym bajcie powinny znajdować się jeszcze dwa bajty zawierające kolejno miesiąc i rok (tylko 2 ostatnie cyfry!).

Opis:

Funkcja ta w zależności od typu zmiennej do której ma być wpisany jej wynik zwraca rozkodowaną datę w postaci liczbowej lub znakowej. Zmiana danych jest przeprowadzana automatycznie.

Konwersja na ciąg znaków:

Ciąg znaków, któremu funkcja ma przypisać datę musi mieć co najmniej 8 znaków. Inaczej bajty następujące po tym ciągu w pamięci zostaną nadpisane.

Konwersja do formatu programowego zegara (CONFIG CLOCK):

Trzy bajty do których funkcja ma zwrócić dzień, miesiąc i rok muszą być umieszczone w pamięci jeden za drugim. Jako parametr funkcji podaje się tylko pierwszy bajt gdzie funkcja umieszcza numer dnia.

Zobacz także: Biblioteka **DATETIME** , **DAYOFYEAR** , **SYSDAY**

Przykład:

```
Enable Interrupts
Config Clock = Soft
Config Date = YMD , Separator = . ' format ANSI

Dim strDate As String * 8
Dim bDay As Byte , bMonth As Byte , bYear As Byte
Dim wSysDay As Word
Dim lSysSec As Long

' Przykład 1: Konwersja danych - 3 Bajty (Dzień / Miesiąc / Rok) na
' Datę w postaci znakowej
bDay = 29 : bMonth = 4 : bYear = 12
strDate = Date(bDay)
Print "Składniki daty: Dzień="; bDay ; " Miesiąc="; bMonth ; " Rok=" ;
bYear ; " skonwerterowane na ciąg znaków " ; strDate
' Składniki daty: Dzień=29 Miesiąc=4 Rok=12 skonwerterowane na ciąg
' znaków 12.04.29

' Przykład 2: Konwersja liczby odpowiadającej liczbie dni
' od początku wieku na datę w postaci znakowej
wSysDay = 1234
strDate = Date(wSysDay)
Print wSysDay ; " dni to " ; strDate
' 1234 dni to 03.05.19

' Przykład 3: Konwersja liczby odpowiadającej liczbie sekund
' od początku wieku na datę w postaci znakowej
lSysSec = 123456789
strDate = Date(lSysSec)
Print lSysSec ; " sekund to " ; strDate
' 123456789 sekund to 03.11.29

' Przykład 4: Konwersja liczby odpowiadającej liczbie dni
' od początku wieku na 3 bajty (Dzień / Miesiąc / Rok)
wSysDay = 2000
bDay = Date(wSysDay)
Print wSysDay ; " dni skonwerterowano na Dzień="; bDay ; " Miesiąc=";
bMonth ; " Rok=" ; bYear
' 2000 dni skonwerterowano na Dzień=23 Miesiąc=6 Rok=5

' Przykład 5: Konwersja daty w postaci znakowej na 3 bajty (Dzień /
' Miesiąc / Rok)
strDate = "04.08.31"
bDay = Date(strDate)
Print "Data " ; strDate ; " skonwerterowana na Dzień="; bDay ; "
Miesiąc="; bMonth ; " Rok=" ; bYear
' Data 04.08.31 skonwerterowana na Dzień=31 Miesiąc=8 Rok=4

' Przykład 6: Konwersja liczby odpowiadającej liczbie sekund na 3
' bajty (Dzień / Miesiąc / Rok)
lSysSec = 123456789
bDay = Date(lSysSec)
```



```
Print lSysSec ; " sekund skonwerterowano na Dzień="; bDay ; "
Miesiąc="; bMonth ; " Rok=" ; bYear
' 123456789 sekund skonwerterowano na Dzień=29 Miesiąc=11 Rok=3
```

DAYOFWEEK() (Nowość w wersji 1.11.7.3)

Przeznaczenie:

Zwraca numer dnia tygodnia dla podanej daty.

Składnia:

```
rezultat = DAYOFWEEK()
rezultat = DAYOFWEEK( bDMY )
rezultat = DAYOFWEEK( strData )
rezultat = DAYOFWEEK( wSysDay )
rezultat = DAYOFWEEK( lSysSec )
```

gdzie:

<i>rezultat</i>	dowolna zmienna gdzie funkcja zwróci wynik,
<i>strData</i>	ciąg znaków z zapisaną datą w formacie określonym przez instrukcję CONFIG DATE ,
<i>lSysSec</i>	zmienna typu Long która przechowuje liczbę sekund jaka upłynęła od początku wieku, (SysSec)
<i>wSysDay</i>	zmienna typu Word która przechowuje liczbę dni jaka upłynęła od początku wieku (SysDay),
<i>bDMY</i>	nazwa zmiennej typu Byte, w której przechowywany jest bieżący dzień. Zaraz po tym bajcie powinny znajdować się jeszcze dwa bajty zawierające kolejno miesiąc i rok (tylko 2 ostatnie cyfry!).

Opis:

Funkcja ta może być wywołana aż z pięcioma wariantami parametrów:

1. Bez parametrów. Wtedy funkcja zwraca numer dnia tygodnia na podstawie zmiennych **programowego zegara** (*_day*, *_month*, *_year*).
2. Ze zdefiniowanym przez użytkownika ciągiem 3 bajtów. Muszą one być zorganizowane podobnie jak zmienne programowego zegara. W pierwszym bajcie należy umieścić dzień, w następnym miesiąc i ostatnim rok (tylko 2 cyfry, bez wieku). Pozwala to na bezproblemowe określenie dnia tygodnia dla dowolnej daty.
3. Z ciągiem znaków, w którym data przedstawiona jest w formacie określonym przez instrukcję **CONFIG DATE**.
4. Ze zmienną typu Word przechowującą liczbę dni jakie upłynęły od początku wieku.
5. Ze zmienną typu Long przechowującą liczbę sekund jaka upłynęła od początku wieku.

Funkcja zwraca liczbę od 0 do 6 która reprezentuje dzień tygodnia. Zero oznacza poniedziałek.

Uwaga! Funkcja zwraca poprawne wartości tylko w XXI wieku (od 2000-01-01 do 2099-12-31).

Zobacz także: **CONFIG DATE** , **CONFIG CLOCK** , **SYSDAY** , **SYSSEC** , **Biblioteka DATETIME**

Przykład:

```
Enable Interrupts
Config Clock = Soft
Config Date = YMD , Separator = . ' format ANSI

Dim bWeekDay As Byte , strWeekDay As String * 10
Dim strDate As String * 8
```

```

Dim bDay As Byte , bMonth As Byte , bYear As Byte
Dim wSysDay As Word
Dim lSysSec As Long

' Przykład 1 z wewnętrznym zegarem RTC
_Day = 24 : _Month = 10 : _Year = 2      ' ustaw zegar - dla przykładu
i testów
bWeekDay = DayOfWeek()
strWeekDay = Lookupstr(bWeekDay , WeekDays)
Print "Dzień " ; Date$ ; " zwraca " ; bWeekday ; " czyli " ;
strWeekday
' Dzień 02.10.24 zwraca 3 czyli Czwartek

' Przykład 2 z zdefiniowanym ciągiem 3 bajtów (Dzień / Miesiąc / Rok )
bDay = 26 : bMonth = 11 : bYear = 2
bWeekDay = DayOfWeek(bDay)
strWeekDay = Lookupstr(bWeekDay , WeekDays)
Print "Dla składników Dzień="; bDay ; " Miesiąc="; bMonth ; " Rok=" ;
bYear ; " zwraca " ; bWeekday ; " czyli " ; strWeekday
' Dla składników Dzień=26 Miesiąc=11 Rok=2 zwraca 1 czyli Wtorek

' Przykład 3 z liczbą dni od początku wieku
wSysDay = 2000                          ' to jest 2005-06-23
bWeekDay = DayOfWeek(wSysDay)
strWeekDay = Lookupstr(bWeekDay , WeekDays)
Print "Dla " ; wSysDay ; " dnia wieku zwraca " ; bWeekday ; " czyli "
; strWeekday
' Dla 2000 dnia wieku zwraca 3 czyli Czwartek

' Przykład 4 z liczbą sekund od początku wieku
lSysSec = 123456789
bWeekDay = DayOfWeek(lSysSec)
strWeekDay = Lookupstr(bWeekDay , WeekDays)
Print "Dla liczby " ; lSysSec ; " sekund zwraca " ; bWeekday ; " czyli "
; strWeekday
' Dla liczby 123456789 sekund zwraca 5 czyli Sobota

' Przykład 5 z ciągiem zawierającym datę
strDate = "02.11.26"                   ' używamy formatu daty wg ANSI
bWeekDay = DayOfWeek(strDate)
strWeekDay = Lookupstr(bWeekDay , WeekDays)
Print "Dla dnia " ; strDate ; " zwraca " ; bWeekday ; " czyli " ;
strWeekday
' Dla dnia 02.11.26 zwraca 1 czyli Wtorek

End

WeekDays:
Data "Poniedziałek", "Wtorek", "Środa", "Czwartek", "Piątek",
"Sobota", "Niedziela"

```

DAYOFYEAR() *(Nowość w wersji 1.11.7.3)*

Przeznaczenie:

Zwraca numer dnia liczonego od początku roku.

Składnia:

```
rezultat = DAYOFYEAR()  
rezultat = DAYOFYEAR( bDMY )  
rezultat = DAYOFYEAR( strData )  
rezultat = DAYOFYEAR( wSysDay )  
rezultat = DAYOFYEAR( lSysSec )
```

gdzie:

<i>rezultat</i>	zmienna typu Integer gdzie funkcja zwróci wynik,
<i>strData</i>	ciąg znaków z zapisaną datą w formacie określonym przez instrukcję CONFIG DATE ,
<i>lSysSec</i>	zmienna typu Long która przechowuje liczbę sekund jaka upłynęła od początku wieku, (SysSec)
<i>wSysDay</i>	zmienna typu Word która przechowuje liczbę dni jaka upłynęła od początku wieku (SysDay),
<i>bDMR</i>	nazwa zmiennej typu Byte, w której przechowywany jest bieżący dzień. Zaraz po tym bajcie powinny znajdować się jeszcze dwa bajty zawierające kolejno miesiąc i rok (tylko 2 ostatnie cyfry!).

Opis:

Funkcja ta może być wywołana aż z pięcioma wariantami parametrów:

1. Bez parametrów. Wtedy funkcja zwraca numer dnia w roku na podstawie zmiennych **programowego zegara** (*_day*, *_month*, *_year*).
2. Ze zdefiniowanym przez użytkownika ciągiem 3 bajtów. Muszą one być zorganizowane podobnie jak zmienne programowego zegara. W pierwszym bajcie należy umieścić dzień, w następnym miesiąc i ostatnim rok (tylko 2 cyfry, bez wieku).
3. Z ciągiem znaków, w którym data przedstawiona jest w formacie określonym przez instrukcję **CONFIG DATE**.
4. Ze zmienną typu Word przechowującą liczbę dni jakie upłynęły od początku wieku.
5. Ze zmienną typu Long przechowującą liczbę sekund jaka upłynęła od początku wieku.

Funkcja zwraca liczbę od 0 do 364 (lub 365 dla lat przestępnych) która reprezentuje numer dnia w roku.

Uwaga! Funkcja zwraca poprawne wartości tylko w XXI wieku (od 2000-01-01 do 2099-12-31).

Zobacz także: **SYSDAY** , **SYSSEC** , **Biblioteka DATETIME**

Przykład:

```
Enable Interrupts  
Config Clock = Soft  
Config Date = YMD , Separator = . ' format ANSI  
  
Dim strDate As String * 8  
Dim bDay As Byte , bMonth As Byte , bYear As Byte  
Dim wSysDay As Word  
Dim lSysSec As Long  
Dim wDayOfYear As Word  
  
' Przykład 1 z wewnętrznym zegarem RTC  
_Day = 20 : _Month = 11 : _Year = 2 ' ustaw zegar - dla przykładu  
i testów  
wDayOfYear = DayOfYear()  
Print "Dla dnia " ; Date$ ; " zwraca " ; wDayOfYear  
' Dla dnia 02.11.20 zwraca 323
```

```

' Przykład 2 z zdefiniowanym ciągiem 3 bajtów (Dzień / Miesiąc / Rok )
bDay = 24 : bMonth = 5 : bYear = 8
wDayOfYear = DayOfYear(bDay)
Print "Dla składników Dzień="; bDay ; " Miesiąc="; bMonth ; " Rok=" ;
bYear ; " zwraca " ; wDayOfYear
' Dla składników Dzień=24 Miesiąc=5 Rok=8 zwraca 144

' Przykład 3 z ciągiem zawierającym datę
strDate = "04.10.29"
wDayOfYear = DayOfYear(strDate)
Print "Dla dnia " ; strDate ; " zwraca " ; wDayOfYear
' Dla dnia 04.10.29 zwraca 302

' Przykład 4 z liczbą dni od początku wieku
wSysDay = 3000
wDayOfYear = DayOfYear(wSysDay)
Print "Dla " ; wSysDay ; " dnia wieku zwraca " ; wDayOfYear
' Dla 3000 dnia wieku zwraca 78

' Przykład 5 z liczbą sekund od początku wieku
lSysSec = 123456789
wDayOfYear = DayOfYear(lSysSec)
Print "Dla liczby " ; lSysSec ; " sekund zwraca " ; wDayOfYear
' Dla liczby 123456789 sekund zwraca 332

End

```

DBG (Nowość w wersji 1.11.6.8)

Przeznaczenie:

Przesyła informacje śledzenia stosu przez sprzętowy UART.

Składnia:

DBG

Opis:

Zobacz opis dyrektywy **\$DBG** by poznać więcej szczegółów.

DEBOUNCE

Przeznaczenie:

Eliminuje drgania styków przełącznika dołączonego bezpośrednio do linii portu.

Składnia:

DEBOUNCE *pin_portu* , *stan* , *etykieta* [, **SUB**]

gdzie:

<i>pin_portu</i>	sprawdzana linia portu (np.: PINB.0),
<i>stan</i>	0 gdy sygnałem aktywnym ma być stan niski, 1 gdy stan wysoki,
<i>etykieta</i>	etykieta określająca miejsce skoku,
SUB	skacze jak do podprogramu (GOSUB).

Opis:

Instrukcja DEBOUNCE oczekuje na zmianę stanu linii portu do określonego stanu. Kiedy określony stan pojawi się na linii portu, instrukcja czeka 25ms i sprawdza ponownie stan portu (eliminuje to drganie styków). Jeśli stan się nie zmienił, skacze do podanej etykiety.

Przed kolejną instrukcją DEBOUNCE stan linii portu musi być nieaktywny, by zapobiec powtórnemu skokowi.

Gdy użyty jest opcjonalny parametr **SUB**, instrukcja skacze do podprogramu jak instrukcja GOSUB.

Każda instrukcja DEBOUNCE korzysta z 1 bitu pamięci RAM, by zapamiętać stan portu. Dla każdego użytego portu jest przydzielany osobny bit.

Zobacz także: CONFIG DEBOUNCE

Przykład:

```
'-----
'                                     DEBOUN.BAS
'                               Demonstracja DEBOUNCE
'-----
Config Debounce = 30 'gdy nie użyjemy tej opcji, domyślnym
                     'opóźnieniem będzie 25ms

Debounce Pind.0 , 0 , Pr , Sub
Debounce Pind.0 , 0 , Pr , Sub
'
'           ^----- etykieta określająca gdzie skoczyć
'           ^----- skocz gdy PIND.0 przejdzie w stan 0
'           ^----- testuj PIND.0

'Debounce Pind.0 , 1 , Pr 'a tu aż przejdzie w stan 1

'kiedy PIND.0 przejdzie w stan niski skacze do Pr
'Pind.0 musi przejść w stan wysoki przed ponownym skokiem
'do Pr gdy Pind.0 przejdzie w stan niski

Debounce Pind.0 , 1 , Pr 'nie będzie skoku
Debounce Pind.0 , 1 , Pr 'spowoduje błąd Return without gosub

End

Pr:
  Print "PIND.0 przeszedł w stan niski"
Return
```

DECR

Przeznaczenie:

Zmniejsza o jeden zawartość zmiennej.

Składnia:

DECR *zmienna*

gdzie:

zmienna

zmienna typu Byte, Integer, Word, Long lub Single

Opis:

Instrukcja zmniejsza o jeden zawartość zmiennej podanej jako parametr. Instrukcję DECR pozostawiono w ramach kompatybilności z językiem BASCOM-8051.

Zobacz także: INCR

Przykład:

```
'-----  
'                                     (c) 1997,1998 MCS Electronics  
'-----  
'   plik: DECR.BAS  
'   demo: DECR  
'-----  
  
Dim A As Byte  
  
A = 5                                'niech w A będzie 5  
Decr A                              'zmniejszamy (o jeden)  
Print A                             'drukujemy  
  
End
```

DECLARE FUNCTION

Przeznaczenie:

Definiuje nagłówek funkcji użytkownika.

Składnia:

DECLARE FUNCTION *nazwa* [([BYREF | BYVAL] *parametr* AS *typ*)] AS *typ_rez*

gdzie:

<i>nazwa</i>	nazwa deklarowanej funkcji,
<i>parametr</i>	nazwa parametrów,
<i>typ</i>	typ przekazywanych parametrów,
<i>typ_rez</i>	typ zwracanego rezultatu.

Opis:

Każda funkcja musi być zdefiniowana instrukcją DECLARE FUNCTION przed jej pierwszym użyciem w programie. Jest to konieczne, by kompilator określił jakie parametry powinny być przekazane funkcji, oraz jaki rezultat funkcja zwraca.

Nie trzeba używać argumentów **BYREF** lub **BYVAL** jeśli nie są wymagane, w takim przypadku przekazywanie parametrów następuje przez adres.

Uwaga! Zmienne typu Bit są zmiennymi globalnymi. Nie można ich zatem umieszczać jako parametry funkcji.

Zobacz także: **CALL**, **SUB**, **FUNCTION**

Przykład:

```
'-----  
'                                     (c) 2000 MCS Electronics  
'                                     demonstracja funkcji użytkownika  
'-----  
'Funkcje użytkownika muszą być zadeklarowane przed pierwszym użyciem  
'Funkcja zawsze zwraca daną określonego typu  
Declare Function Myfunction(Byval I As Integer , S As String) As  
Integer  
'Argument Byval przekazuje funkcji wartość tego parametru, więc  
'nie nastąpi modyfikacja parametru w treści funkcji.  
  
Dim K As Integer  
  
Dim Z As String * 10  
Dim T As Integer
```

```

'przypisujemy jakieś dane
K = 5
Z = "123"

T = Myfunction(k , Z)
Print T
End

Function Myfunction(Byval I As Integer , S As String) As Integer
'można używać lokalnych zmiennych we własnych procedurach i
funkcjach
Local P As Integer

P = I
'ponieważ parametrowi I jest przekazywana jego wartość, zmiana
'nie będzie możliwa
I = 10
P = Val(s) + I

'Na koniec zwracamy rezultat działania funkcji
'Uwaga! Typ danych musi się zgadzać z zadeklarowanym!
Myfunction = P
End Function

```

DECLARE SUB

Przeznaczenie:

Definiuje nagłówek procedury użytkownika.

Składnia:

DECLARE SUB *nazwa* [([BYREF | BYVAL] *parametr* AS *typ*)]

gdzie:

<i>nazwa</i>	nazwa deklarowanej funkcji,
<i>parametr</i>	nazwa parametrów,
<i>typ</i>	typ przekazywanych parametrów,

Opis:

Każda procedura musi być zdefiniowana instrukcją DECLARE SUB przed jej pierwszym użyciem w programie. Jest to konieczne, by kompilator określił jakie parametry powinny być przekazane procedurze.

Nie trzeba używać argumentów BYREF lub BYVAL jeśli nie są wymagane, w takim przypadku przekazywanie parametrów następuje przez adres.

Uwaga! Zmienne typu Bit są zmiennymi globalnymi. Nie można ich zatem umieszczać jako parametry procedury.

Zobacz także: CALL, SUB, FUNCTION

Przykład:

```

Dim a As Byte, b1 As Byte, c As Byte

Declare Sub Test(a As Byte)
a = 1 : b1 = 2 : c = 3

Print a ; b1 ; c

Call Test(b1)

```

```
Print a ; b1 ; c

End

Sub Test(a as byte)
  Print a ; b1 ; c
End Sub
```

DEFxxx

Przeznaczenie

Odgórnie deklaruje zmienne nie określone w instrukcjach DIM.

Składnia:

DEFBIT <i>litera</i>	zmienne Bit
DEFBYTE <i>litera</i>	zmienne Byte
DEFINT <i>litera</i>	zmienne Integer
DEFWORD <i>litera</i>	zmienne Word
DEFLNG <i>litera</i>	zmienne Long
DEFSNG <i>litera</i>	zmienne Single

Opis:

Instrukcje te deklarują typ zmiennych nie zdefiniowanych przez DIM, a zaczynające się od podanej litery.

Różnice w stosunku do QBasic-a.

QBasic pozwala na określenie zakresu liter początkowych definiowanych zmiennych, jak na przykład DEFINT A - D. BASCOM BASIC pozwala tylko określić jedną literę.

Przykład:

```
Defbit b : DefInt c      'zmienne bitowe zaczynać się będą od b
                          'a Integer od c
Set b1                   'na przykład taka zmienna
c = 10                   'oraz c = 10
```

DEFLCDCHAR

Przeznaczenie:

Definiuje znak użytkownika dla wyświetlacza LCD.

Składnia:

DEFLCDCHAR *nr_znaku*, *r1*, *r2*, *r3*, *r4*, *r5*, *r6*, *r7*, *r8*

gdzie:

<i>nr_znaku</i>	numer znaku którego dotyczy definicja (0-7),
<i>r1-r8</i>	kolejne bajty odpowiadające matrycy znaku.

Opis:

Wyświetlacze kompatybilne z standardem Hitachi posiadają pamięć CG-RAM, gdzie można zdefiniować 8 znaków użytkownika. Mają one kody od 0 do 7 (oraz 8 - 15 zdublowane *przyp. tłumacza*). Znaki te można wyświetlać używając instrukcji LCD w połączeniu z funkcją CHR().

W środowisko BASCOM wbudowano narzędzie projektowe dla tych znaków. Znajduje się ono w menu: [Tools | Character Designer](#).

Uwaga! Instrukcja CLS powinna być użyta po instrukcjach DEFLCDCHAR.

Zobacz także: [Tools - LCD designer](#) (w oryginalnym pliku pomocy)

Przykład:

```
Deflcdchar 0,0,28,28,28,28,28,28,0    'definiujemy znak 0
Cls                                     'wybieramy LCD DATA RAM
Lcd Chr(0)                             'wyświetlamy

End
```

DEG2RAD() (Nowość w wersji 1.11.6.8)

Przeznaczenie:

Zamienia stopnie na radiany.

Składnia:

zmienna = DEG2RAD(*stopnie*)

gdzie:

<i>zmienna</i>	zmienna typu Single, do której wpisana będzie wartość w radianach podanego kąta w stopniach,
<i>stopnie</i>	kąt określony w stopniach.

Opis:

Wszystkie funkcje trygonometryczne używają radianów. Konwersji pomiędzy stopniami a radianami można dokonać używając funkcji DEG2RAD oraz RAD2DEG.

Zobacz także: **RAD2DEG**

Przykład:

```
Dim S As Single

S = 90
S = Deg2Rad(s)
Print S
```

DELAY

Przeznaczenie:

Opóźnia działanie programu na krótki czas.

Składnia:

DELAY

Opis:

Można używać instrukcji DELAY by opóźnić działanie programu. Czas opóźnienia wynosi około 1000µs.

Zobacz także: **WAIT** , **WAITMS**

Przykład:

```
Portb = 5
Delay    'czekaj aż urządzenie będzie gotowe
```

Przeznaczenie:

Definiuje zmienną.

Składnia:

DIM *nazwa* [(*rozmiar*)] **AS** [**XRAM** | **ERAM** | **SRAM**] *typ* [**AT** *adres*] [**OVERLAY**]

gdzie:

<i>nazwa</i>	nazwa pod jaką identyfikowana będzie zmienna,
<i>rozmiar</i>	parametr dla zmiennych tablicowych – ilość elementów tablicy,
<i>typ</i>	typ zmiennej,
<i>adres</i>	opcjonalnie – przypisuje adres absolutny zmiennej, będzie ona umieszczona pod wskazanym adresem

Opis:

Każda zmienna powinna być zdefiniowana instrukcją DIM. Zmienne domyślnie są umieszczane w pamięci RAM procesora, jeśli nie zastosowano argumentów **XRAM** lub **ERAM**.

Gdy zastosowano argument **XRAM**, zmienna zostanie umieszczona w pamięci zewnętrznej procesora. Argument **ERAM** spowoduje, że zmienna zostanie umieszczona w wewnętrznej pamięci EEPROM mikrokontrolera.

Zmienne typu String wymagają dodatkowego parametru określającego ich przewidywaną długość w znakach:

```
Dim s As XRAM String * 10
```

W powyższym przykładzie zmienna tekstowa s może zapamiętać ciąg znaków o długości nie większej niż 10 znaków. Będzie ona umieszczona w pamięci zewnętrznej.

Uwaga! Zmienne bitowe mogą być umieszczone tylko w pamięci SRAM (wewnętrznej).

Parametr opcjonalny **AT** pozwala na umieszczenie zmiennej w ściśle określonej komórce pamięci. Gdyby pod podanym adresem znajdowała się już jakaś zmienna, zmiennej definiowanej zostanie przydzielony pierwszy wolny adres.

Parametr **OVERLAY** powoduje, że zmienna umieszczona pod wskazanym adresem wykorzystuje bajty zajmowane przez inną zmienną, która także została umieszczona arbitralnie pod podanym adresem. Dla takiej zmiennej tworzony jest wskaźnik. Wyjaśnia to przykład:

```
Dim x As Long At $60           'używa adresów 60,61,62 i 63 w SRAM
Dim b1 As Byte At $60 OVERLAY
Dim b2 As Byte At $61 OVERLAY
```

Zmienne B1 oraz B2 nie są tutaj prawdziwymi zmiennymi! Ich nazwy tylko wskazują określony bajt w pamięci. W tym wypadku &H60 i &H61. Zapisując coś pod adres wskazywany przez B1, tak naprawdę dane trafiają do przestrzeni zajmowanej przez zmienną x. Można oczywiście także odczytać zawartość bajtu wskazywanego przez B1:

```
Print B1
```

Wydrukowana wartość znajduje się pod adresem &H60. Używając wskaźników można manipulować danymi umieszczonymi w prawdziwych zmiennych.

Jeszcze inny przykład:

```
Dim L As Long At &H60
Dim W As Word At &H62 OVERLAY
```

W wskazuje tutaj na dwa bajty starszego słowa zmiennej typu Long.

Różnice w stosunku do QBasic-a.

W języku QBasic nie trzeba definiować zmiennych przed ich użyciem. W języku BASCOM obszar przeznaczony na zmienne jest ograniczony. Należy więc poinformować kompilator o zamiarze używania zmiennej. Kompilator zaś powiadomi jeśli brakuje miejsca w pamięci na zmienną. Tworzy to mechanizm bezpiecznego gospodarowania pamięcią RAM, oraz służy dodatkowemu zabezpieczeniu kodu przed błędami.

Dodatkowo, argumenty XRAM, SRAM, ERAM nie występują w składni języka QBasic.

Zobacz także: **CONST** , **LOCAL**

Przykład:

```
'-----  
'                               (c) 2000 MCS Electronics  
'-----  
'   plik: DIM.BAS  
'   demo: DIM  
'-----  
Dim B1 As Bit           'bit może mieć wartość 0 lub 1  
Dim A As Byte           'bajt 0-255  
Dim C As Integer        'integer -32767 - +32768  
Dim L As Long  
Dim W As Word  
Dim S As String * 10     'tekst może mieć do 10 znaków  
  
'NOWOŚĆ: można określić adres zmiennej w pamięci  
Dim K As Integer At 120  
  
'następna zmienna będzie umieszczona w pamięci za zmienną S  
Dim Kk As Integer  
  
'przypisanie bitu  
B1 = 1                     'lub  
Set B1                    'używając Set  
  
'przypisanie bajtu  
A = 12  
A = A + 1  
  
'przypisanie integer  
C = -12  
C = C + 100  
Print C  
  
W = 50000  
Print W  
  
'liczba long  
L = 12345678  
Print L  
  
'i jeszcze tekst  
S = "Witaj świecie"  
Print S  
  
End
```

DISABLE

Przeznaczenie:

Wyłącza źródła przerwań.

Składnia

DISABLE *źródło*

gdzie:

źródło symboliczna nazwa źródła przerwania:

Tabela 11 Lista źródeł przerwań.

Nazwa	Źródło przerwania
INTERRUPTS	Globalna flaga statusu przerwań
INT0	Przerwanie zewnętrzne (końcówka INT0)
INT1	Przerwanie zewnętrzne (końcówka INT1)
OVF0, TIMER0, COUNTER0	Przerwanie przepełnienia licznika TIMER0
OVF1, TIMER1, COUNTER1	Przerwania przepełnienia licznika TIMER1
CAPTURE1, ICP1	Przerwanie z rejestru INPUT CAPTURE w liczniku TIMER1
COMPARE1A, OC1A	Przerwanie z rejestru OUTPUT COMPARE A licznika TIMER1
COMPARE1B, OC1B	Przerwanie z rejestru OUTPUT COMPARE B licznika TIMER1
SPI	Przerwanie z interfejsu SPI
URXC	Przerwanie urządzenia UART – odebrano znak
UDRE	Przerwanie urządzenia UART – pusty rejestr nadajnika
UTXC	Przerwanie urządzenia UART – wysłano znak
SERIAL	Blokuje jednocześnie URXC, UDRE oraz UTXC
ACI	Przerwanie z wewnętrznego komparatora
ADC	Przerwanie z przetwornika A/D

Opis:

Za pomocą instrukcji **DISABLE** można wyłączać pojedyncze źródła przerwań lub całość systemu przerwań. Standardowo wszystkie przerwania są wyłączone.

Aby wyłączyć cały system przerwań należy użyć nazwy: **INTERRUPTS**. Aby ponownie włączyć system przerwań należy użyć instrukcji **ENABLE INTERRUPTS**.

Uwaga! Liczba dostępnych przerwań zależy od typu użytego mikrokontrolera.

Zobacz także: **ENABLE**

Przykład:

```
'-----  
'                               SERINT.BAS  
'           przerwania układu UART w kontrolerach AVR  
'-----  
  
Dim B As Bit                               'flaga odebrania znaku  
  
Dim Bc As Byte                             'licznik znaków  
Dim Buf As String * 20                     'bufor transmisji  
'Buf = Space(20)  
'usuń komentarz w powyższej linii dla funkcji MID() w ISR  
'trzeba wstawić spację, gdyż inaczej bufor zawierałby śmieci  
  
On Urxrc Rec_isr                           'definiujemy procedurę obsługi  
przerwania
```

Enable Urx	'włączamy przerwanie UART'
Enable Interrupts	'włączamy przerwania'
Do	
If B = 1 Then	'otrzymaliśmy coś?'
Disable Serial	
Print Buf	
Print Bc	
Reset B	
Enable Serial	
End If	
Loop	
Rec_isr:	
If Bc < 20 Then	'czy zmieści się w buforze?'
Incr Bc	'zwiększamy licznik znaków'
Buf = Buf + Chr (udr)	'dopisujemy do bufora'
' Mid(buf , Bc , 1) = Udr	
'usuń komentarz i wstaw go na początku poprzedniej linii programu	
'by umieszczać znaki w dowolne miejsce w buforze	
B = 1	'ustaw flagę'
End If	
Return	

DISPLAY

Przeznaczenie:

Włącza lub wyłącza ekran LCD.

Składnia:

DISPLAY ON | OFF

Opis:

Ekran LCD jest włączany automatycznie, gdyż kompilator BASCOM umieszcza w procedurze zerowania stosowne instrukcje.

Zobacz także: [LCD](#)

Przykład:

```
Dim a as Byte

a = 255
Lcd a
Display Off
Wait 1
Display On

End
```

DO...LOOP

Przeznaczenie:

Powtarza blok programu dopóki warunek końcowy nie będzie spełniony.

Składnia:

DO
 ciąg_instrukcji
LOOP [**UNTIL** *warunek*]

Opis:

Instrukcja tworzy blok zwany pętlą. Instrukcje zawarte pomiędzy DO a LOOP, będą się wykonywać dopóki *warunek* nie będzie spełniony – tj. będzie miał wartość logicznego fałszu. Gdy warunek nie jest podany pętla będzie się wykonywać w nieskończoność. Można wyjść z pętli w dowolnym momencie stosując instrukcję **EXIT DO**.

Uwaga! Warunek pętli jest obliczany na końcu, więc instrukcje w pętli wykonają się co najmniej jeden raz.

Zobacz także: **EXIT** , **WHILE - WEND** , **FOR - NEXT**

Przykład:

```
'-----  
'                                     (c) 1999 MCS Electronics  
'-----  
'   plik: DO_LOOP.BAS  
'   demo: DO, LOOP  
'-----  
  
Dim A As Byte  
  
Do                               'początek pętli  
    A = A + 1                    'zwiększamy A  
    Print A                      'drukujemy  
Loop Until A = 10                'powtarzamy aż A = 10  
Print A  
  
'można tworzyć puste i niekończące się pętle:  
Do  
    'inne instrukcje w pętli będą tutaj  
Loop
```

DTMFOUT

Przeznaczenie:

Generuje tony w systemie DTMF.

Składnia:

DTMFOUT *liczba* , *czas*
DTMFOUT *ciąg* , *czas*

gdzie:

<i>liczba</i>	liczba określająca kod klawisza na klawiaturze telefonicznej,
<i>ciąg</i>	ciąg znaków określający numer, który będzie wygenerowany w systemie DTMF,
<i>czas</i>	czas w milisekundach przypadający na poszczególne cyfry.

Opis:

Instrukcja DTMFOUT oparta została na nocie aplikacyjnej AN314 firmy Atmel.

Instrukcja używa podczas generacji licznika TIMER1. W konsekwencji czego, licznik ten nie może być używany do generacji przerwań. Choć nadal można go używać w innym celu.

Ponieważ licznik TIMER1 jest używany przy generowaniu przerwań, należy włączyć globalny system przerwań instrukcją **ENABLE INTERRUPTS**. Kompilator może to zrobić automatycznie, lecz gdy używane są inne przerwania, lepiej jest to zrobić jawnie w programie.

Instrukcja będzie działać poprawnie jeśli zegar systemowy będzie miał częstotliwość w granicach od 4 do 10MHz.

Wyjściem impulsów DTMF jest końcówka OC1A. W układzie AT90s2313 jest to PB.3.

UWAGA! Dołączanie jakichkolwiek urządzeń nie posiadających homologacji Ministerstwa Łączności jest zabronione w Polsce. Ponadto bezpośrednie połączenie do linii może stwarzać niebezpieczeństwo, gdyż „na otwartej” linii panuje ponad 50V!

Asembler:

Wywoływana jest procedura _DTMFOUT z biblioteki mcs.lib. Rejestr R16 zawiera numer tonu, a R24-R25 czas jego trwania. Procedura używa R9, R10, R16-R23.

Przykład:

```
'-----
'                                     DTMFOUT.BAS
'   demonstruje instrukcję DTMFOUT opartą na nocie AN314 f-my Atmel
'       min. częstotliwość oscylatora 4 MHz, max. 10 MHz
'-----

'Ponieważ DTMFOUT używa przerwań licznika TIMER1, musisz włączyć
globalny system przerwań
'Nie jest to wykonywane automatycznie, gdyż program może posiadać
więcej procedur obsługi przerwań.
Enable Interrupts

'Będziemy wysyłać kolejne kody w pętli
Dim Btmp As Byte

Do
'jest tylko 16 możliwych tonów
For Btmp = 0 To 15
    Dtmfout Btmp , 500                                'generuj DTMF na końcówce PORTB.3
                                                         '(dla 2313) o czasie trwania 500
ms                                                         'wyjściem jest końcówka OC1A
                                                         'czekaj 1 sekundę
    Waitms 1000
Next
Loop

End

'klawiatura większości telefonów wygląda tak:
'1  2  3      opcjonalnie: A
'4  5  6      B
'7  8  9      C
'*  0  #      D

'instrukcja DTMFOUT przekształca liczbę z zakresu 0-15 na kod
klawisza:
'liczba      klawisz
'  0          0
'  1          1
'  2          2
'  3          3
' itd.
'  9          9
' 10          *
' 11          #
```

' 12	A
' 13	B
' 14	C
' 15	D

ECHO

Przeznaczenie:

Włącza i wyłącza echo dla instrukcji INPUT.

Składnia:

ECHO ON | OFF

Opis:

Do wprowadzania danych służy instrukcja **INPUT**, która wszystkie odebrane znaki przesyła z powrotem do terminala, by użytkownik widział co pisze. Gdy taka akcja nie jest wymagana lub w terminalu włączono lokalne echo, w programie można umieścić ECHO z parametrem OFF. Spowoduje to wyłączenie „odbijania” wpisywanych znaków.

Uwaga! W starszych wersjach (<1.11.6.2) parametr ECHO był kontrolowany lokalnie dla każdej instrukcji INPUT. Jak w przykładzie:

```
Input "Echo?" , var NOECHO
```

Podanie parametru **NOECHO** wyłączało echo dla wprowadzanych znaków. Według nowej składni należy używać instrukcji ECHO z parametrem **ON** lub **OFF**.

Standardowo ECHO jest włączone (**ON**).

Zobacz także: INPUT

Asembler:

Wywoływana jest jedna z procedur biblioteki `mcs.lib: _ECHO_ON` lub `_ECHO_OFF`. Gdy w programie występuje instrukcja ECHO OFF, kompilator generuje następujący kod:

```
Rcall _Echo_Off
```

Spowoduje to ustawienie 3 bitu w rejestrze R6, który przechowuje flagę stanu instrukcji ECHO. Gdy ECHO jest włączane to kod wygląda tak:

```
Rcall _Echo_On
```

Przykład:

```
Dim Var As Byte

'wyłącz echo
Echo Off
'gdy będziesz wpisywał dane to ich nie zobaczysz
Input Var
'włącz echo
Echo On
'teraz będziesz widział co piszesz!
Input Var
```

ELSE

Przeznaczenie:

Wykonuje instrukcje po słowie ELSE, jeśli wyrażenie w instrukcji IF-THEN lub SELECT-CASE jest fałszywe

Składnia:

ELSE instrukcja

gdzie:

instrukcja dowolna instrukcja języka BASCOM BASIC.

Opis:

Instrukcja ELSE może być używana tylko w konstrukcjach **IF – THEN** lub **SELECT**. Jeśli warunek podany jako wyrażenie w instrukcji IF-THEN lub SELECT CASE będzie fałszywe, to program przejdzie do instrukcji występującej po słowie ELSE.

Po słowie ELSE można także umieścić kilka instrukcji oddzielonych dwukropkami. Ważne jest, by były one umieszczone w jednej linii programu.

Jeśli fragment programu wykonywany po instrukcji ELSE jest dość złożony, należy użyć konstrukcji IF-THEN-ELSE-END IF.

Aby sprawdzić kolejny warunek można użyć instrukcji **ELSEIF**. Na przykład:

```
If a = 1 Then
...
Elseif a = 2 Then
...
Elseif b1 > a Then
...
Else
...
End If
```

Zobacz także: **IF-THEN-ELSE-ENDIF** , **SELECT CASE - END SELECT**

Przykład:

```
A = 10
If A > 10 Then
    Print "A > 10"
Else
    Print "A nie większe niż 10"
End If
```

'niech a = 10
'podejmujemy decyzję
'to się nie wyświetli
'alternatywa
'to będzie wyświetlone

ENABLE

Przeznaczenie:

Włącza źródła przerwań.

Składnia

ENABLE źródło

gdzie:

źródło symboliczna nazwa źródła przerwania:

Tabela 12 Lista źródeł przerwań.

Nazwa	Źródło przerwania
INTERRUPTS	Globalna flaga systemu przerwań
INT0	Przerwanie zewnętrzne (końcówka INT0)
INT1	Przerwanie zewnętrzne (końcówka INT1)
OVF0, TIMER0, COUNTER0	Przerwanie przepełnienia licznika TIMER0

Nazwa	Źródło przerwania
OVF1, TIMER1, COUNTER1	Przerwanie przepełnienia licznika TIMER1
CAPTURE1, ICP1	Przerwanie z rejestru INPUT CAPTURE w liczniku TIMER1
COMPARE1A, OC1A	Przerwanie z rejestru OUTPUT COMPARE A licznika TIMER1
COMPARE1B, OC1B	Przerwanie z rejestru OUTPUT COMPARE B licznika TIMER1
SPI	Przerwanie z interfejsu SPI
URXC	Przerwanie urządzenia UART – odebrano znak
UDRE	Przerwanie urządzenia UART – pusty rejestr nadajnika
UTXC	Przerwanie urządzenia UART – wysłano znak
SERIAL	Blokuje jednocześnie URXC, UDRE oraz UTXC
ACI	Przerwanie z wewnętrznego komparatora
ADC	Przerwanie z przetwornika A/D

Opis:

Za pomocą instrukcji **ENABLE** można włączać pojedyncze źródła przerwania lub całość systemu przerwania. Standardowo wszystkie przerwania są wyłączone.

Aby włączyć cały system przerwania należy użyć nazwy: **INTERRUPTS**. Aby ponownie wyłączyć system przerwania należy użyć instrukcji **DISABLE INTERRUPTS**.

Uwaga! Liczba dostępnych przerwania zależy od typu użytego mikrokontrolera. Niektóre procesory mogą mieć jeszcze inne źródła przerwania, np.: INT2, INT3.

Zobacz także: **DISABLE**

Przykład:

```
Enable Interrupts  'włączamy system przerwania
Enable Timer1     'włączamy przerwanie przepełniania licznika TIMER1
```

END

Przeznaczenie:

Kończy działanie programu.

Składnia

END

Opis:

Po wykonaniu instrukcji **END**, wszystkie przerwania są wyłączane i wykonywana jest tzw. nieskończona pętla.

Zamiast instrukcji **END** można użyć instrukcji **STOP**. Różnica jest taka, że instrukcja **STOP** nie powoduje wyłączenia przerwania przed przejściem do nieskończonej pętli.

Zobacz także: **STOP**

Przykład:

```
Print "Hello"      'drukujemy
End                'kończymy program i wyłączamy przerwania
```

ERR

Przeznaczenie:

Zwraca 1 gdy wystąpił błąd.

Opis:

Zmienna **ERR** odzwierciedla stan znacznika błędu, który mógł pojawić się podczas ostatnio wykonywanej instrukcji. Nie wszystkie instrukcje ustawiają znacznik błędu.

Procedury lub funkcje użytkownika mogą ustawiać znacznik błędu, wpisując 1 do zmiennej ERR. Także procedury umieszczone we własnych bibliotekach mogą modyfikować zmienną ERR.

EXIT

Przeznaczenie:

Wychodzi z instrukcji: FOR..NEXT, DO..LOOP, WHILE..WEND, SUB..END SUB lub FUNCTION..END FUNCTION.

Składnia:

```
EXIT FOR
EXIT DO
EXIT WHILE
EXIT SUB
EXIT FUNCTION
```

Opis:

Za pomocą instrukcji EXIT można w każdej chwili wyjść z dowolnej instrukcji strukturalnej.

Przykład:

```
If a >= b1 Then      'jakiś kod
  Do                 'zaczynamy pętlę DO..LOOP
    A = A + 1         'jak Incr a
    If A = 100 Then   'czy a = 100?
      Exit Do         'wychodzimy!
    End If            'koniec instrukcji IF..THEN
  Loop               'koniec DO..LOOP
End If                'koniec pierwszej IF..THEN
```

EXP()

Przeznaczenie:

Zwraca wartość liczby **e** (podstawy logarytmu naturalnego) podniesioną do podanej potęgi.

Składnia:

zmienna = **EXP**(*wykładnik*)

gdzie:

zmienna dowolna zmienna typu Single, do której wpisany będzie wynik działania funkcji,
wykładnik zmienna lub stała typu Single, określająca wartość wykładnika podstawy liczby **e**.

Zobacz także: **LOG**, **LOG10**

Przykład:

```
Dim X As Single

X = Exp(1.1)
Print X                'wydrukuje 3.004166124
```

```
X = 1.1
X = Exp(x)
Print X
```

'wydrukuje 3.004164931

FIX() (Nowość w wersji 1.11.6.8)

Przeznaczenie:

Zwraca dla wartości większej niż zero następną mniejszą liczbę, a dla wartości większych niż zero następną większą liczbę.

Składnia:

zmienna = **FIX**(*wartość*)

gdzie:

<i>zmienna</i>	dowolna zmienna typu Single, do której wpisany będzie wynik działania funkcji,
<i>wartość</i>	zmienna lub stała typu Single, określająca wartość poddawaną korekcji.

Zobacz także: **INT** , **ROUND** , **SGN**

Przykład:

```
'-----
'                                     ROUND_FIX_INT.BAS
'-----
Dim S As Single , Z As Single

For S = -10 To 10 Step 0.5
    Print S ; Spc(3) ; Round(s) ; Spc(3) ; Fix(s) ; Spc(3) ; Int(s)
Next

End
```

FORMAT()

Przeznaczenie:

Formatuje tekst zawierający liczbę według podanego wzorca.

Składnia:

rezultat = **FORMAT**(*zmienna* , "*maska*")

gdzie:

<i>rezultat</i>	zmienna tekstowa, w której umieszczony zostanie sformatowany tekst,
<i>zmienna</i>	zmienna tekstowa zawierająca formatowaną liczbę,
<i>maska</i>	wzorzec formatowania.

Opis:

Instrukcja formatuje tekst zawierający liczbę według podanego wzorca, który składa się z określonych znaków:

Gdy w masce występują znaki spacji, przed liczbą zostaną wstawione spacje by liczba została wyrównana do lewej. Dla przykładu gdy maska zawiera 8 spacji " ", to do liczby 123 zostanie dodanych 5 spacji wiodących " 123".

Gdy w masce wystąpi znak plus "+" to przed liczbą dodatnia zostanie dodany normalnie nie występujący znak plus. Na przykład maska "+" spowoduje że liczba 123 zostanie sformatowana do "+123".

Umieszczenie w masce znaku zero "0" spowoduje, że przed liczbą umieszczone zostaną

wiodące zera. Na przykład: maska " +00000" da rezultat " +00123".

Można także w masce umieścić znak kropki ".", wtedy zostanie ona wstawiona na odpowiadającej jej pozycji. Dla przykładu: maska "000.00" da w rezultacie "001.23".

Można w masce umieścić wszystkie znaki formatujące, byle by zachowana była odpowiednia kolejność: spacje, +, 0 i opcjonalnie razem z zerami kropka.

Przykład:

```
'-----  
'                                     (c) 2000 MCS Electronics  
'-----  
  
Dim S As String * 10  
Dim I As Integer  
  
S = "12345"  
S = Format(s , "+")  
Print S  
  
S = "123"  
S = Format(s , "00000")  
Print S  
  
S = "12345"  
S = Format(s , "000.00")  
Print S  
  
S = "12345"  
S = Format(s , " +000.00")  
Print S  
  
End
```

FOR...NEXT

Przeznaczenie:

Wykonuje blok instrukcji pomiędzy FOR a NEXT określoną ilość razy, jednocześnie modyfikując podaną zmienną.

Składnia:

```
FOR zmienna = liczba_początkowa TO liczba_końcowa [STEP krok]  
...  
NEXT [zmienna]
```

gdzie:

<i>zmienna</i>	dowolna zmienna numeryczna, będąca licznikiem przejść pętli,
<i>liczba_początkowa</i>	liczba startowa licznika pętli,
<i>liczba_końcowa</i>	liczba końcowa licznika pętli,
<i>krok</i>	krok co jaki będzie zmieniany licznik pętli po dotarciu do instrukcji NEXT.

Opis:

W przeciwieństwie do innych pętli DO-LOOP czy WHILE-WEND, pętla FOR-NEXT wykona się tylko z góry określoną liczbę razy. Można to określić wzorem: $(liczba_końcowa - liczba_początkowa) / krok$.

Uwaga! Istniejący w wersji 1.11a format instrukcji z **DOWNTO**, nadal jest obsługiwany. Instrukcja FOR-NEXT została jednak przekonstruowana w celu zwiększenia kompatybilności. Zaleca się by używać instrukcji FOR-NEXT według nowej składni.

Każda pętla FOR musi się kończyć instrukcją **NEXT**. Użycie **STEP** jest opcjonalne dla zwiększania zmiennej, gdyż standardowo krok wynosi 1. **STEP** stosuje się gdy zmienna musi zostać zmniejszana po każdym przejściu pętli. Wtedy po słowie STEP musi być umieszczona liczba ujemna, jak to pokazano w przykładzie.

Uwaga! Nie należy mieszać ze sobą pętli, jak to pokazano tutaj:

```
For A = 1 To 10
  For B = 1 To 20
    Print A ; ":" ; B
  Next A
Next B
```

Program będzie działał poprawnie, lecz konstrukcja ta może stać się nieczytelna (*przyp. tłumacza*).

Zobacz także: **EXIT FOR**

Przykład:

```
'-----
'                                     (c) 2000 MCS Electronics
'-----
'   plik: FOR_NEXT.BAS
'   demo: FOR, NEXT
'-----

Dim A As Byte , B1 As Byte , C As Integer

For A = 1 To 10 Step 2
  Print "A jest równe " ; A
Next A

Print "teraz liczenie w dół"
For C = 10 To -5 Step -1
  Print "C jest równe " ; C
Next

Print "Możesz zagnieźdzać kilka pętli FOR..NEXT"
For A = 1 To 10
  Print "To jest A " ; A
  For B1 = 1 To 10
    Print "A to jest B1 " ; B1
  Next
  'nie musisz umieszczać nazwy zmiennej, gdyż kompilator wie że chodzi
  'o zmienną B1
Next A

End
```

FOURTHLINE

Przeznaczenie:

Ustawia kursor wyświetlacza LCD w czwartej linii.

Składnia:

FOURTHLINE

Opis:

Instrukcja ta działa tylko na wyświetlaczach posiadających cztery linie.

Zobacz także: **HOME** , **UPPERLINE** , **LOWERLINE** , **THIRDLINE** , **LOCATE**

Przykład:

```
Dim a As byte

a = 255
Lcd a
Fourthline
Lcd a
Upperline

End
```

FRAC() *(Nowość w wersji 1.11.6.8)*

Przeznaczenie:

Zwraca ułamkową część podanej liczby.

Składnia:

zmienna = **FRAC**(*wartość*)

gdzie:

<i>zmienna</i>	dowolna zmienna typu Single, do której wpisana będzie ułamkowa część podanej liczby,
<i>wartość</i>	liczba której ułamkowa część powinna być zwrócona.

Opis:

Część ułamkowa to wszystkie liczby znajdujące się po przecinku z prawej strony.

Zobacz także: **INT**

Przykład:

```
Dim A As Single

A = 9.123
A = Frac(A)
Print A           'wydrukuje 0.123
```

FUSING()

Przeznaczenie:

Formatuje liczbę zmiennoprzecinkową według podanego wzorca.

Składnia:

rezultat = **FUSING**(*zmienna* , "*maska*")

gdzie:

<i>rezultat</i>	zmienna tekstowa, w której umieszczony zostanie sformatowany tekst,
<i>zmienna</i>	zmienna typu Single zawierająca formatowaną liczbę,
<i>maska</i>	wzorzec formatowania.

Opis:

Maska jest ciągiem znaków, który musi zaczynać się od znaku # (hash). Po kropce należy umieścić tyle znaków #, ile ma pojawić się znaków po przecinku. Na przykład podanie maski jako "#.###" sformatuje liczbę do postaci 123.456. Gdy liczba cyfr po przecinku jest większa niż

określona w masce, nastąpi zaokrąglenie ostatniej cyfry. Na przykład liczba 123.4567 zostanie sformatowana do 123.457.

Jeśli operacja zaokrąglania nie powinna być przeprowadzana zamiast umieszczonych po kropce znaków # należy umieścić znak & (ampersand). Dla powyższej liczby maska "#. &&&" spowoduje sformatowanie liczby do postaci 123.456.

Gdy liczba jest zerem funkcja zwraca zawsze 0.0, niezależnie od podanej maski.

Zobacz także: **FORMAT** , **STR**

Przykład:

```
'-----  
'                                     FUSING.BAS  
'                                     (c) 2001 MCS Electronics  
'-----  
  
Dim S As Single , Z As String * 10  
  
'najpierw przypiszemy zmiennej jakąś wartość  
S = 123.45678  
'Użycie funkcji Str() pozwala na konwersję liczby na jej  
'reprezentację w postaci tekstu  
Z = Str(s)  
Print Z                                'wydrukuj 123.456779477  
  
Z = Fusing(s , "#.##")  
  
'Teraz sformatujemy liczbę według wzorca z zaokrągleniem  
Print Fusing(s , "#.##")              'wydrukuj 123.46  
  
'A teraz to samo tylko bez zaokrąglania  
Print Fusing(s , "#.&&")              'wydrukuj 123.45  
  
'Maska musi rozpoczynać się od znaku #.  
'Należy także umieścić choć jeden znak # lub & po kropce.  
'Nie można mieszać znaków # i & w masce po kropce.  
  
End
```

FUNCTION

Przeznaczenie:

Rozpoczyna treść funkcji użytkownika.

Składnia:

```
FUNCTION nazwa [ ( parametr AS typ [, parametr AS typ] ) ] AS typ_rezultatu  
    instrukcje funkcji  
END FUNCTION
```

gdzie:

<i>nazwa</i>	nazwa funkcji,
<i>parametr</i>	nazwa parametru funkcji,
<i>typ</i>	typ przekazywanego parametru,
<i>typ_rezultatu</i>	typ danych zwracanych przez funkcję.

Opis:

Każda funkcja musi być zakończona instrukcją END FUNCTION.

Najlepiej skopiować deklarację funkcji (instrukcja DECLARE FUNCTION na początku programu) a następnie skasować słowo DECLARE. Można w ten sposób ustrzec się od błędów, polegających na różnicy pomiędzy częścią deklaracyjną a treścią funkcji.

Zobacz temat **DECLARE FUNCTION** by poznać więcej szczegółów.

GETADC()

Przeznaczenie:

Pobiera wynik przetwarzania z wbudowanego układu A/D.

Składnia:

zmienna = **GETADC**(*nr_kanału*)

gdzie:

zmienna zmienna do której wpisana będzie przetworzona wartość,
nr_kanału numer kanału przetwornika A/D. Może być z zakresu 0 – 7.

Opis:

Funkcja GETADC() działa jedynie na mikrokontrolerach AVR90s8535 i innych posiadających wbudowany przetwornik A/D.

Końcówki wejściowe przetwornika A/D działają także jako normalne linie portów. Jest szczególnie ważne, by podczas pracy przetwornika ich stan nie był zmieniany.

Należy się upewnić czy przetwornik AD został wcześniej włączony instrukcją **START ADC**, lub też przez ustawienie odpowiednich bitów w rejestrze kontrolnym przetwornika.

Przykład:

```
'-----  
'                               ADC.BAS  
'   demonstracja funkcji GETADC() dla układu AVR Mega163  
'-----  
$regfile = "m163def.dat"  
  
'Konfigurujemy tryb pracy na SINGLE i prescaler na AUTO  
'Tryb SINGLE musi być wybrany dla poprawnego działania GETADC()  
  
'Prescaler dzieli sygnał zegarowy przez 2,4,8,15,32,64 lub 128,  
'ponieważ przetwornik ADC działa z częstotliwością 50-200kHz  
'Ustawienie AUTO spowoduje, że dzielnik zostanie tak ustawiony aby  
'wybrana częstotliwość była najwyższa z możliwych.  
Config Adc = Single , Prescaler = Auto  
'teraz włączamy przetwornik (włącza zasilanie!)  
Start Adc  
  
'Instrukcją STOP ADC, można wyłączyć przetwornik  
'Stop Adc  
  
Dim W As Word , Channel As Byte  
  
Channel = 0  
  
'odczytujemy przetworzoną wartość z kolejnych kanałów  
Do  
  W = Getadc(channel)  
  Print "Kanał " ; Channel ; " wartość " ; W  
  Incr Channel  
  If Channel > 7 Then Channel = 0
```

Loop

End

```
'Nowy układ M163 posiada pewne opcje dotyczące napięcia odniesienia
'Dla tego układu można stosować dodatkową opcję:
'Config Adc = Single , Prescaler = Auto, Reference = Internal
'Dla parametru Reference można podać:
' OFF      : napięcie trzeba dostarczyć do końcówki AREF;
'          : wewnętrzne napięcie odniesienia jest wyłączone.
' AVCC     : napięcie jest pobierane z końcówki AVCC;
'          : należy dołączyć kondensator do końcówki AREF.
' INTERNAL : napięcie wewnętrzne 2,56V; kondensator do końcówki AREF.

'Użycie tego dodatkowego parametru dla układów nie posiadających
wewnętrznego napięcia
'odniesienia powoduje zignorowanie tej opcji.
```

GETATKBD()

Przeznaczenie:

Odczytuje dane z klawiatury PC AT podłączonej do kontrolera.

Składnia

zmienna = GETATKBD()

gdzie:

zmienna zmienna typu Byte lub String, do której wpisany będzie odczytany znak lub jego kod. Funkcja zwraca zero gdy nie naciśnięto żadnego z klawiszy.

Opis:

Funkcja ta pozwala na odczyt danych z klawiatury PC AT, podłączonej bezpośrednio do mikrokontrolera. Sposób dołączenia jest opisany przy opisie instrukcji CONFIG KEYBOARD.

Funkcja oczekuje na odebranie znaku z klawiatury. Przerwanie pętli oczekiwania może nastąpić przez ustawienie zmiennej ERR, na przykład w procedurze obsługi przerwania.

GETATKBD używa dwóch bitów w rejestrze R6: bit 4 i 5 jest używany jako flagi stanu klawiszy Shift i Control.

Zobacz także: CONFIG KEYBOARD

Przykład:

```
'-----
'                               PC AT-KEYBOARD Sample
'                               (c) 2000 MCS Electronics
'-----
'Aby uruchomić ten przykładowy program:
'dołącz sygnał CLOCK z klawiatury PCAT do PIND.2 w AT90s8535
'dołącz sygnał DATA z klawiatury PCAT do PIND.4 w AT90s8535
$regfile = "8535def.dat"

'Funkcja GetATKBD() nie korzysta z systemu przerw.
'Oczekuje jednak na naciśnięcie jednego z klawiszy!

'Konfigurujemy linie portów do podłączenia klawiatury
'mogą być dowolne byle by pracowały jako wejścia
'Keydata jest to adres tablicy przeliczeniowej
```

```

Config Keyboard = Pind.2 , Data = Pind.4 , Keydata = Keydata

'definicja niezbędnych zmiennych
Dim S As String * 12
Dim B As Byte

'Użyjemy tu także przekierowania instrukcji INPUT
$serialinput = Kbdinput

'Pokaż że program zadziałał
Print "Cześć"

Do
    'poniższy fragment znajduje się w komentarzach, gdyż nie
    'jest używany pokazuje jednak jak używać funkcji GetATKBD()
    ' B = Getatkbd()      'odczytaj dane z klawiatury

    'Kiedy nie naciśnięto klawisza rezultat funkcja zwraca zero
    'więc trzeba testować kiedy zwróci wartość większą od zera
    ' If B > 0 Then
    '     Print B ; Chr(b)
    ' End If

    'Przeznaczeniem tego programu jest pokazanie jak używać klawiatury
    PC AT
    'Dane odbierane instrukcją Input, pochodzą z klawiatury (a nie
    'jak normalnie z łącza szeregowego).
    'Należy więc naciskać klawisze na dołączonej klawiaturze!
    Input "Jak ci na imię?" , S
    'pokazujemy echo
    Print S
Loop

End

'Ponieważ użyliśmy przekierowania, trzeba napisać procedurę która
'odbierze znak z klawiatury.
'
Kbdinput:
    'Tutaj program wskoczy gdy będą odczytywane znaki przez Input.
    'Zrobimy to tak by naciśnięty klawisz trafił do rejestru R24, i
    'wykorzystamy do tego funkcję GetATkbd z assemblera
    'Teraz fragment zabezpieczający rejestry
$asm
    Push r16          ; zapamiętujemy je na stosie
    Push r25
    Push r26
    Push r27

    Kbdinput1:
    Rcall _getatkbd    ; wywołujemy funkcję
    Tst r24            ; czy zwróciła zero
    Breq Kbdinput1    ; jeśli tak to czekamy dalej
    Pop r27           ; coś przyszło więc wracamy
    Pop r26           ; przedtem odtwarzamy rejestry
    Pop r25
    Pop r16
$end asm
    'teraz naprawdę wracamy
Return

```

```

'Skorzystamy z programistycznej sztuczki, gdyż TRZEBA! wykonać
'normalny skok do procedury inaczej wystąpił by błąd
'Nie jest to zbyt jasna metoda i będzie zmieniona w przyszłości
B = Getatkbd()

'Poniżej znajduje się tabela konwersji

Keydata:
'klawisze normalnie - małe litery
Data 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , &H5E , 0
Data 0 , 0 , 0 , 0 , 0 , 113 , 49 , 0 , 0 , 0 , 122 , 115 , 97 , 119 ,
50 , 0
Data 0 , 99 , 120 , 100 , 101 , 52 , 51 , 0 , 0 , 32 , 118 , 102 , 116
, 114 , 53 , 0
Data 0 , 110 , 98 , 104 , 103 , 121 , 54 , 7 , 8 , 44 , 109 , 106 ,
117 , 55 , 56 , 0
Data 0 , 44 , 107 , 105 , 111 , 48 , 57 , 0 , 0 , 46 , 45 , 108 , 48 ,
112 , 43 , 0
Data 0 , 0 , 0 , 0 , 0 , 92 , 0 , 0 , 0 , 0 , 13 , 0 , 0 , 92 , 0 , 0
Data 0 , 60 , 0 , 0 , 0 , 0 , 8 , 0 , 0 , 49 , 0 , 52 , 55 , 0 , 0 , 0
Data 48 , 44 , 50 , 53 , 54 , 56 , 0 , 0 , 0 , 43 , 51 , 45 , 42 , 57
, 0 , 0

'klawisze z Shift - wielkie litery
Data 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
Data 0 , 0 , 0 , 0 , 0 , 81 , 33 , 0 , 0 , 0 , 90 , 83 , 65 , 87 , 34
, 0
Data 0 , 67 , 88 , 68 , 69 , 0 , 35 , 0 , 0 , 32 , 86 , 70 , 84 , 82 ,
37 , 0
Data 0 , 78 , 66 , 72 , 71 , 89 , 38 , 0 , 0 , 76 , 77 , 74 , 85 , 47
, 40 , 0
Data 0 , 59 , 75 , 73 , 79 , 61 , 41 , 0 , 0 , 58 , 95 , 76 , 48 , 80
, 63 , 0
Data 0 , 0 , 0 , 0 , 0 , 96 , 0 , 0 , 0 , 0 , 13 , 94 , 0 , 42 , 0 , 0
Data 0 , 62 , 0 , 0 , 0 , 8 , 0 , 0 , 49 , 0 , 52 , 55 , 0 , 0 , 0 , 0
Data 48 , 44 , 50 , 53 , 54 , 56 , 0 , 0 , 0 , 43 , 51 , 45 , 42 , 57
, 0 , 0

```

GETKBD()

Przeznaczenie:

Sprawdza stan klawiatury matrycowej (4x4 lub 4x6) i zwraca numer wciśniętego klawisza.

Składnia:

zmienna = GETKBD()

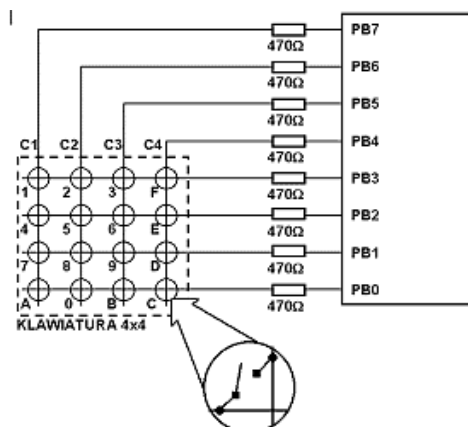
gdzie:

zmienna zmienna do której wpisany będzie umowny numer naciśniętego klawisza.

Opis:

Funkcja GETKBD() zwraca numer naciśniętego klawisza, w klawiaturze matrycowej, podłączonej do portu mikrokontrolera. Gdy nie naciśnięto żadnego z klawiszy funkcja zwraca liczbę 16. Dla matrycy 4x6 funkcja zwraca wtedy 24.

Port do którego podpięto klawiaturę określa się instrukcją **CONFIG KBD**. Schemat połączenia klawiatury z portem PORTB pokazano poniżej:



Rysunek 19 Sposób dołączenia klawiatury matrycowej do portu mikrokontrolera.

Końcówki portu mogą również pełnić inną rolę, gdy klawiatura nie jest odczytywana. Należy jednak pamiętać, że użycie funkcji GETKBD powoduje przekonfigurowanie używanego portu jako wejście. Jeśli port ten pracował również jako wyjście, należy przywrócić odpowiedni stan rejestru DDRx, lub też użyć ponownie instrukcji **CONFIG PORT** lub **CONFIG PIN**.

Ponieważ funkcja zwraca numer umowy klawisza w matrycy, można dokonać konwersji kodów klawiszy (za pomocą danych w linii **DATA** w połączeniu z funkcją **LOOKUP()**) by były one zgodne z posiadaną klawiaturą.

Uwaga! Obecnie funkcja nie wprowadza standardowego opóźnienia 100ms po odczycie numeru naciśniętego klawisza. Opóźnienie to należy określić podając odpowiedni parametr w instrukcji **CONFIG KBD**.

Na płytce prototypowej STK200 funkcja może nie działać, gdyż inne urządzenia są podłączone do tego samego portu.

Uwaga! Zdarza się czasami, że funkcja zwraca numer naciśniętego klawisza choć żaden z nich nie jest naciśnięty. W takim wypadku należy dołączyć rezystory 1kΩ, połączone z jednej strony do końcówek portu PORTx.4 – PORTx.7, a z drugiej do masy.

Zobacz także: **CONFIG KBD**

Przykład:

```
'-----
'                               GETKBD.BAS
'                               (c) 2000 MCS Electronics
'-----
'określamy do którego portu dołączono klawiaturę,
'wszystkie jego linie są używane
Config Kbd = Portb , Delay = 100

'określamy zmienną do której trafiać będą umowne numery klawiszy
Dim B As Byte

'pętla bez końca
Do
  B = Getkbd()
  'sprawdź w pliku pomocy jak dołączyć klawiaturę
  Print B
  'Jeśli żaden z klawiszy nie jest naciśnięty funkcja zwraca 16
  'Użyj funkcji Lookup() by zamienić numery klawiszy na odpowiednie
  kody
  'gdyż funkcja zwraca umowne numery klawiszy w matrycy
```

```

Loop
Lcd B

End

```

GETRC()

Przeznaczenie:

Określa wartość zmiany rezystancji lub pojemności.

Składnia:

zmienna = GETRC(*port* , *numer*)

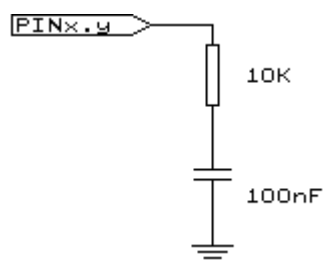
gdzie:

<i>zmienna</i>	zmienna typu Word w której znajdzie się wynik pomiaru,
<i>port</i>	nazwa portu procesora, np.: PIND,
<i>numer</i>	numer końcówki do której podpięty został obwód RC.

Opis:

Nazwa końcówki portu (na przykład PIND) musi zostać podana, nawet gdy inne końcówki portu są skonfigurowane jako wyjścia. Dlatego, należy również podać numer bitu odpowiadającej tejże końcówce. Może być on określony zmienną lub stałą liczbową.

Poniżej pokazano sposób podłączenia obwodu RC:



Rysunek 20 Przykładowa gałąź RC.

Zasada działania jest dość prosta. Kondensator jest ładowany przez ustawienie 1 na końcówce portu. Następnie mierzony jest czas rozładowania kondensatora, który jest zwracany jako wynik funkcji. Zmiana wartości rezystancji lub pojemności spowoduje zatem zmianę tego czasu.

Celem tej funkcji jest określenie relatywnej zmiany rezystancji, podłączonego np.: potencjometru, czy innego rezystancyjnego czujnika. Oczywiście można za pomocą dodatkowych obliczeń określić wartość rezystancji, lecz pomiar taki byłby obciążony dość dużym błędem.

Przykład:

```

'-----
'                               GETRC.BAS
' Demonstruje jak odczytać rezystancję jaką reprezentuje dołączony
rezystor
'-----
'Działanie instrukcji jest następujące:
' Na początku kondensator jest ładowany.
' Następnie jest on rozładowywany w krótkich odcinkach czasu.
' Przy czym czas rozładowania jest liczony i zwracany w zmiennej.
'Dlatego rezultat działania tej funkcji powinien być traktowany jako
'zmianę pozycji np. suwaka potencjometru, a nie jako aktualną wartość
'rezystancji

```

```

'Przykład ten używa kontrolera 90s8535 z dołączonym obwodem RC do
końcówki PIND.4
'Rezystor ma wartość 10k, a kondensator 100nF.

'Funkcja ta różni się od tej w dialekcie BASCOM-8051!
'Spowodowane jest to różnicami w architekturze tych procesorów.

'Funkcja Getrc() zwraca liczbę typu Word więc deklarujemy taką
Dim W As Word

Do
'Pierwszym parametrem jest nazwa rejestru PIN.
'Drugim jest numer końcówki tego portu, gdzie dołączono układ RC.
'Drugi parametr może być zmienną!
W = Getrc(Pind , 4)
Print W
Wait 1
Loop

```

GETRC5()

Przeznaczenie:

Odbiera i rozkodowuje sygnał pilota pracującego w standardzie RC5.

Składnia:

GETRC5(*adres* , *rozkaz*)

gdzie:

<i>adres</i>	adres urządzenia, dla którego przeznaczony jest rozkaz,
<i>rozkaz</i>	bajt określający rozkaz.

Opis:

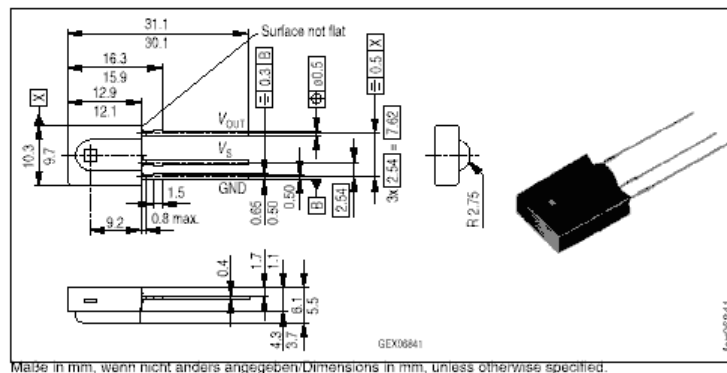
Funkcję tą opracowano na podstawie noty aplikacyjnej AVR410 firmy Atmel.

W czasie działania funkcji używany jest licznik TIMER0 i związane z nim przerwanie. Licznik ten jest służy do generacji wymaganych opóźnień podczas próbkowania sygnału. TIMER0 może być dalej używany przez program, lecz należy się liczyć z wynikłymi opóźnieniami, gdy wykonywana będzie funkcja GETRC5(). Jest tylko jeden warunek: przerwania licznika TIMER0 nie mogą być używane.

Zmiany wprowadzone od wersji 1.11.6.9.

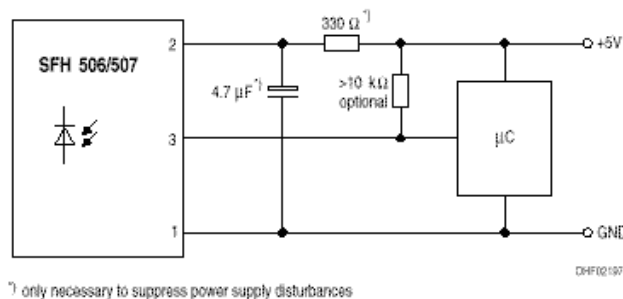
Funkcja GETRC5 odbiera rozszerzone kody RC5. Podziękowania dla **Gert Boer**, który jest autorem tej implementacji.

Jako odbiornik podczerwieni używany jest scalony detektor **SFH506-36** produkowany przez Siemens-a lub jego odpowiednik pracujący na częstotliwości 36kHz.



Rysunek 21 Scalony odbiornik podczerwieni.

Aby detektor pracował bez zakłóceń należy odpowiednio odfiltrować jego zasilanie:



Rysunek 22 Sposób dołączenia odbiornika do mikrokontrolera.

Wiele urządzeń audio-video jest wyposażonych w system zdalnego sterowania pracujący w podczerwieni. Dość duża grupa tego typu urządzeń (szczególnie produkowanych przez europejskie firmy *przyp. tłumacza*) używa transmisji w kodzie RC5.

Nadajniki pracujące według standardu RC5 transmitują 14-bitowe słowa danych, kodowane w formacie bi-phase, zwanym także kodem Manchester.

Pierwsze dwa bity słowa są zawsze jedyнкami i tworzą razem sygnał startu. Następny bit jest bitem kontrolnym (*toggle bit*), zmieniańm w kolejnych nadawanych słowach, gdy użytkownik przytrzyma klawisz pilota – umożliwia to powtarzanie komend. Kolejne 5 bitów reprezentuje adres urządzenia, które ma być właściwym odbiornikiem transmisji. Dla przykładu: odbiorniki telewizyjne mają zazwyczaj adres 0, a magnetowidy adres 5. Ostatnie 6 bitów reprezentuje jedną z 64 możliwych komend.

Dla rozszerzonego kodu RC5, dodatkowy bit jest transmitowany jako bit 6. Dlatego też bit kontrolny (*toggle bit*) został przeniesiony do bitu 7.

Zobacz także: **CONFIG RC5**

Przykład:

```

'-----
'                               RC5.BAS
'                               (c) 2000 MCS Electronics
'                               oparte na nocie aplikacyjnej AVR410 f-my Atmel
'-----

'Ten przykład pokazuje jak odczytać sygnały w kodzie RC5
'odbierane przez detektor SFH506-35.

```



```

'Wyjście detektora podłącz do PIND.2
'Funkcja GETRC5 używa licznika TIMER0 i jego przerwania.
'Ustawienia licznika są zapamiętywane i odtwarzane po wykonaniu
funkcji.
'Nie dotyczy to jednak obsługi przerw!

'ustawiamy linię wejściową dla danych RC5
Config Rc5 = Pind.2

'Procedura przerwania TIMER0 jest wstawiana automatycznie.
'Należy tylko włączyć system przerw!
Enable Interrupts

'określamy zmienne
Dim Address As Byte , Command As Byte

Do
    'teraz sprawdzamy czy jakiś klawisz pilota został naciśnięty
    'Uwaga! Po włączeniu zasilania wszystkie linie portów pracują
    'jako wejścia. Dlatego nie używamy tutaj instrukcji określających
    'kierunki portów.
    'Jeśli wymagane jest określenie kierunku końcówki portu usuń
    'komentarz z poniższej linii.
    'Config Pind.2 = Input
    Getrc5(address , Command)

    'sprawdzamy czy adres jest równy 0 (korzystamy z pilota TV)
    If Address = 0 Then
        'zerujemy znacznik w słowie RC5
        'bit znacznika jest zmieniany podczas każdej transmisji
        Command = Command And &B10111111

        Print Address ; " " ; Command
    End If
Loop

End

```

GLCDCMD (Nowość w wersji 1.11.6.9)

Przeznaczenie:

Wysyła rozkaz dla graficznego wyświetlacza LCD opartego na kontrolerze SEDxxxx.

Składnia:

GLCDCMD *rozkaz*

gdzie:

rozkaz zmienna lub stała bajtowa określający rozkaz.

Opis:

Za pomocą instrukcji GLCDCMD można przesłać rozkaz który ma wykonać kontroler wyświetlacza. W ten sposób można manipulować wyświetlaczem gdy w zestawie instrukcji BASCOM Basic brak odpowiedniej instrukcji.

Uwaga! Aby używać tej instrukcji należy zadeklarować w programie użycie biblioteki GLCDSED używając instrukcji: `$lib "glcdsed.lbx"`

Zobacz także: **CONFIG GRAPHLCD** , **LCDAT** , **GLCDDATA**

GLCDDATA *(Nowość w wersji 1.11.6.9)*

Przeznaczenie:

Wysyła dane dla graficznego wyświetlacza LCD opartego na kontrolerze SEDxxxx.

Składnia:

GLCDDATA *dana*

gdzie:

dana dowolna zmienna lub stała bajtowa przeznaczona dla wyświetlacza.

Opis:

Za pomocą instrukcji GLCDDATA można przesłać dowolne dane, które towarzyszą wysyłanemu przedtem rozkazowi. W ten sposób można manipulować wyświetlaczem gdy w zestawie instrukcji BASCOM Basic brak odpowiedniej instrukcji.

Uwaga! Aby używać tej instrukcji należy zadeklarować w programie użycie biblioteki GLCDSED używając instrukcji: `$lib "glcdsed.lbx"`

Zobacz także: **CONFIG GRAPHLCD** , **LCDAT** , **GLCDCMD**

GOSUB

Przeznaczenie:

Wykonuje skok to podprogramu.

Składnia:

GOSUB *etykieta*

gdzie:

etykieta nazwa etykiety do której skacze program.

Opis:

Za pomocą instrukcji GOSUB można wykonać skok pod adres określony etykietą i wykonać umieszczone tam instrukcje. Pozwala to na podzielenie programu na fragmenty – podprogramy, które mogą być wykorzystane wielokrotnie, bez potrzeby pisania ich za każdym razem.

Gdy w podprogramie umieszczona będzie instrukcja RETURN, nastąpi powrót do instrukcji następnej po instrukcji GOSUB (tej która wykonała skok). Adresy powrotne są zapamiętywane na specjalnym stosie.

Gdy w programie wystąpią dwie etykiety o tej samej nazwie, kompilator zasygnalizuje ten fakt.

Zobacz także: **GOTO** , **CALL** , **RETURN**

Przykład:

```
Gosub Routine      'skok do podprogramu
Print "Cześć"      'powrót z podprogramu nastąpi tutaj

End                'koniec programu

Routine:           'tutaj jest podprogram
  x = x + 2        'obliczymy coś
  Print x          'wydrukujemy wynik
Return            'i wracamy
```

GOTO

Przeznaczenie:

Wykonuje skok do określonej etykiety.

Składnia:

GOTO *etykieta*

gdzie:

etykieta nazwa etykiety do której nastąpi skok

Opis:

Instrukcja skacze bezwarunkowo do określonej etykiety. Nie istnieje instrukcja powrotu jak w instrukcji GOSUB. Etykieta może mieć maksimum 32 znaki.

Gdy w programie wystąpią dwie etykiety o tej samej nazwie, kompilator zasygnalizuje ten fakt.

Zobacz także: **GOSUB**

Przykład:

```
'-----  
'                               (c) 1999 MCS Electronics  
'-----  
'   plik: GOSUB.BAS  
'   demo: GOTO, GOSUB and RETURN  
'-----  
  
Goto Continue  
Print "Ten tekst się nie pokaże"  
  
Continue:                               'etykiety kończymy dwukropkiem  
  Print "Rozpoczynamy tutaj"  
  Gosub Routine  
  Print "Powrót z podprogramu"  
  
End  
  
Routine:                               'początek podprogramu  
  Print "To będzie wykonane"  
Return                               'powrót z podprogramu
```

GREY2BIN()

Przeznaczenie:

Przekształca liczbę zapisaną w kodzie Greya na jej wartość binarną.

Składnia:

zmienna1 = **GREY2BIN**(*zmienna2*)

gdzie:

zmienna1 zmienna do której zapisana będzie liczba po konwersji,
zmienna2 liczba w kodzie Greya poddawana konwersji.

Opis:

Kod Greya jest używany w koderach obrotowych.

Funkcja GREY2BIN() może konwertować zmienne typu Byte, Integer, Word lub Long. Typ podanej jako parametr zmiennej określa także typ zwracanej wartości.

Zobacz także: **BIN2GREY**

Asembler:

W zależności od typu podanej zmiennej, jest wywoływana odpowiednia procedura z biblioteki mcs.lbx: `_bin2grey` dla typu `Byte`, `_bin2grey2` dla typów `Integer/Word` oraz `_bin2grey4` dla `Long`.

Przykład:

```
'-----  
'                                     (c) 2001-2002 MCS Electronics  
' Przykład ten pokazuje działanie funkcji Bin2Grey oraz Grey2Bin  
' Podziękowania dla Josef Franz Vögel za poprawienie działania funkcji  
' i wersji dla zmiennych typu word/long  
'-----  
  
'Bin2Grey() zamienia bajt, Integer, Word, Long na zapisany w kodzie  
Greya.  
'Grey2Bin() zamienia wartość zapisana w kodzie Greya na wartość  
binarną  
  
Dim B As Byte                                     'może być także Word/Integer,  
Long  
  
Print "BIN" ; Spc(8) ; "GREY"  
For B = 0 To 15  
    Print B ; Spc(10) ; Bin2grey(b)  
Next  
  
Print "BIN" ; Spc(8) ; "GREY"  
For B = 0 To 15  
    Print B ; Spc(10) ; Grey2bin(b)  
Next  
  
End
```

HEX()

Przeznaczenie:

Zwraca w formie tekstu szesnastkową reprezentację liczby.

Składnia:

`zmienna = HEX(liczba)`

gdzie:

<code>zmienna</code>	zmienna tekstowa, w której znajdzie się liczba zapisana w formacie szesnastkowym,
<code>liczba</code>	liczba poddana konwersji, może być stałą lub zmienną typu Integer, Word, Long czy Byte.

Zobacz także: **HEXVAL** , **VAL** , **STR** , **BIN**

Przykład:

```
Dim a As Byte, S As String * 10, Sn As Single  
  
a = 123  
s = Hex(a)  
Print s  
Print Hex(a)
```

```

Sn = 1.2
Print Hex (Sn)

End

```

HEXVAL()

Przeznaczenie:

Zamienia tekstową reprezentację liczby szesnastkowej na odpowiadającą jej wartość.

Składnia:

zmienna = HEXVAL(*liczba*)

gdzie:

<i>zmienna</i>	zmienna do której zapisana będzie liczba po konwersji,
<i>liczba</i>	liczba szesnastkowa poddana konwersji, zapisana w zmiennej tekstowej.

Opis:

Różnice w stosunku do QBasic-a.

W języku QBasic można używać funkcji VAL() do konwersji liczb szesnastkowych. Określenie czy liczba jest rzeczywiście w zapisie szesnastkowym, odbywa się za pomocą testu czy rozpoczyna się od przedrostka &H.

W języku BASCOM BASIC by uniknąć testowania wprowadzono osobną instrukcję HEXVAL.

Zobacz także: HEX , VAL , STR , BIN

Przykład:

```

Dim a As Integer, s As string * 15

s = "A"
a = Hexval(s)
Print a ; Spc(10) ; Hex(a)

End

```

HIGH()

Przeznaczenie:

Zwraca starszą część (bajt MSB) podanej zmiennej.

Składnia:

rezultat = HIGH(*zmienna*)

gdzie:

<i>rezultat</i>	zmienna do której zapisana będzie starszy bajt zmiennej,
<i>zmienna</i>	zmienna której starsza część ma być określona.

Zobacz także: LOW , HIGHW

Przykład:

```

Dim I As Integer , Z As Byte

I = &H1001
Z = High(I) 'zwróci 16

```

End

HIGHW()

Przeznaczenie:

Zwraca starsze słowo podanej zmiennej typu Long.

Składnia:

rezultat = **HIGHW**(*zmienna*)

gdzie:

rezultat

zmienna

zmienna do której zapisana będzie starsze słowo zmiennej,
zmienna typu Long której starsza część ma być określona.

Opis:

Nie istnieje funkcja LOWW, gdyż przypisanie wartości zmiennej typu Long do zmiennej typu Word powoduje przepisanie tylko młodszej części zmiennej Long. Dlatego zrezygnowano z osobnej funkcji LOWW.

Zobacz także: **LOW** , **HIGH**

Przykład:

```
Dim X As Word , L As Long
```

```
L = &H12345678
```

```
X = Highw(L)
```

```
Print Hex(x)
```

HOME

Przeznaczenie:

Ustawia pozycję kursora LCD na początku określonej linii.

Składnia:

HOME [**UPPER** | **LOWER** | **THIRD** | **FOURTH**]

Opis:

Jeśli nie podano nazwy linii, kursor ustawiany jest na początku pierwszej linii wyświetlacza.

Można także użyć skrótowego zapisu nazwy linii, np.: HOME U, co odpowiada instrukcji HOME UPPER.

Zobacz także: **CLS** , **LOCATE**

Przykład:

```
Lowerline
```

```
Lcd "Under"
```

```
Home Upper
```

```
Lcd "Lepper"
```

I2CINIT (Nowość w wersji 1.11.6.8)

Przeznaczenie:

Przywraca odpowiedni stan końcówek pełniących rolę linii SCL i SDA.

Składnia:
I2CINIT

Opis:

Standardowo stan końcówek pełniących rolę linii SCL i SDA jest poprawny po wyzerowaniu procesora. Oba rejestry PORTx i DDRx zawierają zera.

Gdy potrzebna jest modyfikacja rejestrów DDRx oraz(lub) PORTx, można użyć instrukcji I2CINIT by przywrócić poprawny stan odpowiednich końcówek.

Zobacz także: **I2CSEND** , **I2CSTART** , **I2CSTOP** , **I2CRBYTE**

Asembler:

Procedury obsługi magistrali I2C są umieszczone w bibliotece `I2C.LIB/I2C.LBX`.

Przykład:

```
Config Sda = Portb.5
Config Scl = Portb.7
I2cinit

Dim X As Byte , Slave As Byte

X = 0                                'zerowanie zmiennej
Slave = &H40                         'adres układu PCF8574
I2creceive Slave , X                 'odczytaj wartość
Print X                              'i wydrukuj
```

I2CRECEIVE

Przeznaczenie:

Odczytuje dane z urządzenia dołączonego do magistrali I2C.

Składnia:

I2CRECEIVE *adres, zmienna*
I2CRECEIVE *adres, zmienna, bajty_zapisywane, bajty_odczytywane*

gdzie:

<i>adres</i>	zmienna lub stała określająca adres urządzenia,
<i>zmienna</i>	zmienna z której zapisywane i do której odczytywane będą dane,
<i>bajty_zapisywane</i>	ilość bajtów które należy przelać do urządzenia,
<i>bajty_odczytywane</i>	ilość bajtów które należy odebrać z urządzenia.

Opis:

Jako *adres* należy podać adres bazowy, gdyż instrukcja sama ustawia bit R/W w bajcie adresowym urządzenia I2C.

Uwaga! Przy podawaniu ilości bajtów zapisanych/odczytywanych należy uważać by nie określić ich za dużo.

Jeśli wystąpił jakiś błąd, zmienna ERR zwraca 1. Inaczej jest ona zerowana.

Zobacz także: **I2CSEND** , **I2CRBYTE**

Asembler:

Procedury i funkcje dotyczące magistrali I2C są umieszczone w bibliotece `i2c.lib / i2c.lbx`.

Przykład:

```
x = 0                                'zerowanie zmiennej
slave = &H40                         'określamy adres np.: PCF 8574 I/O
I2creceive slave, x                  'odczytujemy stan końcówek
Print x                             'i drukujemy

Dim buf(10) As Byte
buf(1) = 1 : buf(2) = 2
I2creceive slave, buf(), 2, 1        'zapisujemy dwa bajty i czytamy jeden
Print buf(1)                         'drukujemy co odebraliśmy
```

I2CSEND

Przeznaczenie:

Wysyła ciąg danych do urządzenia podłączonego do magistrali I2C.

Składnia:

```
I2CSEND adres, zmienna
I2CSEND adres, zmienna, bajty_zapisywane
```

gdzie:

<i>adres</i>	zmienna lub stała określająca adres urządzenia,
<i>zmienna</i>	zmienna będąca z której pobrane będą dane do wysłania,
<i>bajty_zapisywane</i>	ilość bajtów które należy przesłać do urządzenia.

Opis:

Większość układów komunikujących się za pomocą magistrali I2C, posiada możliwość odbioru większej ilości danych. Do najpopularniejszych należą pamięci EEPROM, które posiadają tryb zapisu wielobajtowego. Po podaniu początkowego adresu można kolejno przysyłać bajty, które umieszczone zostaną jeden za drugim w strukturze pamięci. Pamięć sama wtedy będzie zwiększać adres, po odebraniu bajtu.

Jeśli wystąpił jakiś błąd, zmienna ERR zwraca 1. Inaczej jest ona zerowana.

Zobacz także: **I2CRECEIVE** , **I2CWBYTE**

Asembler:

Procedury i funkcje dotyczące magistrali I2C są umieszczone w bibliotece `i2c.lib` / `i2c.lbx`.

Przykład:

```
x = 5                                'wpisujemy 5
Dim ax(10) As Byte

Const slave = &H40                   'adres układu PCF 8574 I/O
I2csend slave, x                     'przesyłamy liczbę 5

For a = 1 To 10
    ax(a) = a                        'wypełniamy tablicę danych
Next
bytes = 10
I2csend slave , ax(), bytes          'transmitujemy dane

End
```


I2START, I2CSTOP, I2CRBYTE, I2CWBYTE

Przeznaczenie:

I2CSTART generuje sygnał startu wymagany przy inicjacji transmisji po magistrali I2C.

I2CSTOP generuje sygnał stopu wymagany po zakończeniu transmisji magistralą I2C.

I2CRBYTE odbiera jeden bajt z zaadresowanego urządzenia.

I2CWBYTE wysyła jeden bajt do określonego urządzenia.

Składnia:

I2CSTART

I2CSTOP

I2CRBYTE *zmienna*, **ACK** | **NACK**

I2CWBYTE *zmienna*

Opis:

Instrukcje wprowadzono jako dodatkowe dla instrukcji I2CSEND oraz I2CRECEIVE.

Instrukcja I2CRBYTE posiada dodatkowy parametr, którego znaczenie jest następujące:

ACK	Parametr ACK występuje, gdy bieżący odbierany bajt nie jest ostatnim.
NACK	Parametr NACK występuje, gdy bieżący bajt jest ostatnim z odczytywanych.

Jeśli wystąpił jakiś błąd, zmienna ERR zwraca 1. Inaczej jest ona zerowana.

Zobacz także: **I2CRECEIVE** , **I2CSEND**

Asembler:

Procedury i funkcje dotyczące magistrali I2C są umieszczone w bibliotece `i2c.lib` / `i2c.lbx`.

Przykład:

```
'----- Zapis i odczyt danych z pamięci EEPROM 2404 -----'
Dim a As Byte
Const adresW = 174 'adres do zapisu układu 2404
Const adresR = 175 'adres do odczytu układu 2404

I2cstart          'warunek startu
I2cwbyte  adresW  'wyślij adres układu
I2cwbyte  1        'wyślij adres w pamięci EEPROM
I2cwbyte  3        'wyślij daną
I2cstop           'warunek stopu

Waitms 10         'czekamy 10ms ponieważ 2404 potrzebuje czasu na
                  'zapisanie danych w pamięci

'----- teraz odczytamy to co zapisaliśmy -----'
I2cstart          'warunek startu
I2cwbyte  adresW  'zaadresujemy urządzenie
I2cwbyte  1        'wyślij adres komórki EEPROM
I2cstart          'znów generujemy warunek startu
I2cwbyte  adresR  'adresujemy układ EEPROM do odczytu
I2crbyte  a, Nack  'odczytujemy komórkę pamięci.
                  'Nack oznacza że nie będzie następnego odczytu
I2cstop           'warunek stopu
Print a          'i drukujemy zawartość komórki
End
```

IDLE

Przeznaczenie:

Wprowadza procesor w stan beczynności – Idle.

Składnia:

IDLE

Opis:

W stanie beczynności, jednostka centralna zostaje zatrzymana i nie wykonuje żadnych rozkazów. Działa jednak system przerwań, układ transmisji szeregowej i liczniki.

Wyprowadzenie procesora ze stanu beczynności następuje po zgłoszeniu przerwania (np.: z układu Watchdog, liczników, wejść przerywających czy przetwornika A/D) lub na skutek wyzerowania procesora – aktywny stan końcówki RESET.

Zobacz także: **POWERDOWN**

Przykład:

```
Idle
Print "Pobudka!"           'O! Było przerwanie...
```

IF...THEN...ELSE...END IF

Przeznaczenie:

Tworzy tzw. blok decyzyjny.

Składnia:

```
IF wyrażenie THEN
    ciąg_instrukcji
    [ ELSEIF wyrażenie THEN ciąg_instrukcji ]
    [ ELSE ciąg_instrukcji ]
END IF

IF wyrażenie THEN ciąg_instrukcji [ ELSE ciąg_instrukcji]
```

gdzie:

wyrażenie	testowane wyrażenie logiczne,
ciąg_instrukcji	dowolny ciąg instrukcji, wykonywanych po spełnieniu lub nie warunku określonego w wyrażeniu.

Opis:

Instrukcja IF..THEN oblicza logiczną wartość podanego wyrażenia. Jeśli będzie ono prawdziwe (wynikiem będzie logiczna prawda) wykonany zostanie blok instrukcji umieszczony po instrukcji THEN. Jeśli będzie ono fałszywe, to instrukcje po słowie THEN nie zostaną wykonane. Wykonane za to będą instrukcje po słowie ELSE, jeśli ono występuje.

Aby nie stosować dość złożonych zagnieżdżonych bloków instrukcji IF..THEN przewidziano też instrukcję ELSEIF, w której testowane może być dodatkowe wyrażenie – po stwierdzeniu fałszywego pierwszego wyrażenia (występującego po słowie IF).

Nowością jest możliwość testowania stanu bitów:

```
Dim A As Byte

If A.3 = 1 Then ...
```

Można to zrobić także w ten sposób:

```
Dim A As Byte, idx As Byte

idx = 3
If A.IDX = 1 Then ...
```

Numer bitu może być zawierać się w przedziale 0-255.

Zobacz także: ELSE , SELECT - CASE

Przykład:

```
Dim A As Integer

A = 10
If A = 10 Then                                'jeśli wyrażenie będzie
    Print "Ta część zostanie wykonana."      'to zostanie wydrukowane
    'prawdziwe

Else
    Print "Ta instrukcja się nie wykona."    'jeśli będzie fałszywe,
    'wydrukujemy to
End If
If A = 10 Then Print "Nowość w języku BASCOM BASIC"
If A = 10 Then Goto LABEL1 Else Print "A<>10"
LABEL1:

Rem Poniższy fragment pokazuje rozszerzenia instrukcji IF THEN
If A.15 = 1 Then                               'testowanie stanu bitu
    Print "Bit 15 jest ustawiony"
End If

Rem a tu przykład instrukcji IF THEN [ELSE] umieszczonej w jednej
linii.
If A.15 = 0 Then Print "BIT 15 jest wyzerowany" Else Print "BIT 15
jest ustawiony"
```

INCR

Przeznaczenie:

Zwiększa zawartość zmiennej o jeden.

Składnia:

INCR *zmienna*

gdzie:

zmienna

zmienna typu Byte, Integer, Word, Long lub Single

Opis:

Zawartość zmiennej jest zwiększana o jeden. Instrukcja jest szybsza niż taka sama operacja z wykorzystaniem operatora +.

Zobacz także: DECR

Przykład:

```
Dim A As Byte
Do
    Incr A
    Print A
Loop Until a > 10
```

'pętelka
'zwiększamy a o jeden
'drukujemy
'powtarzamy aż a będzie większe niż 10

Print A

INITLCD

Przeznaczenie:

Inicjalizuje wyświetlacz LCD.

Składnia:

INITLCD

Opis:

Inicjalizacja wyświetlacza LCD jest przeprowadzana podczas startu systemu, gdy program korzysta z wyświetlacza LCD.

Jeśli z jakiegoś powodu musi nastąpić powtórna inicjalizacja wyświetlacza, należy użyć instrukcji INITLCD.

Asembler:

Generowany jest następujący kod:

```
Rcall _Init_LCD
```

Zobacz także: [LCD](#)

INKEY()

Przeznaczenie:

Zwraca kod ASCII pierwszego znaku znajdującego się w buforze transmisji szeregowej.

Składnia:

zmienna = **INKEY**()

zmienna = **INKEY**(#kanał)

gdzie:

<i>zmienna</i>	dowolna zmienna typu Byte, Integer, Word, Long lub Single,
<i>kanał</i>	numer otwartego kanału, przy odczytywaniu kodów z programowego lub sprzętowego urządzenia UART, otwartego instrukcją OPEN.

Opis:

Jeśli w buforze nie ma żadnego znaku funkcja zwraca zero. Można także najpierw sprawdzić czy w buforze transmisji znajduje się jakikolwiek znak. W tym celu należy skorzystać z funkcji [ISCHARWAITING](#).

Funkcja INKEY może być użyteczna gdy projektowany system mikroprocesorowy posiada interfejs RS-232. Interfejs ten może być połączony z komputerem PC wyposażonym w port COM.

Zobacz także: [WAITKEY](#) , [ISCHARWAITING](#)

Przykład:

```
Do                                'jakaś pętelka
  A = Inkey()                    'odczytujemy znak
  If A > 0 Then                   'czy był jakiś (> 0)?
    Print A                      'jeśli tak to piszemy
  End If
Loop                             'to będzie pętelka nieskończona
```

```
'Powyższy przykład dotyczy SPRZĘTOWEGO układu UART!  
'Program zapisany jako OPEN.BAS zawiera przykład użycia INKEY()  
'w połączeniu z programowym układem UART.
```

INP()

Przeznaczenie:

Zwraca bajt odczytany z portu mikrokontrolera lub z podanego adresu w przestrzeni adresowej mikrokontrolera.

Składnia:

zmienna = INP(*adres*)

gdzie:

<i>zmienna</i>	dowolna zmienna typu Byte,
<i>adres</i>	adres w przestrzeni adresowej (0-&HFFFF) z którego odczytany zostanie bajt.

Opis:

Funkcja PEEK() potrafi odczytać tylko początkowe 32 bajty pamięci RAM, gdzie umieszczono zestaw rejestrów R0-R31. Funkcja INP() może odczytywać z dowolnego adresu pamięci RAM, gdyż w kontrolerach AVR organizacja tej pamięci jest liniowa. Znaczy to, że pamięć zewnętrzna jest przedłużeniem wewnętrznej pamięci RAM.

Zobacz także: OUT, PEEK

Przykład:

```
Dim a As Byte  
  
a = Inp(&H8000) 'odczytanie stanu szyny danych d0-d7  
'gdy na szynie adresowej jest wystawiony adres 8000h  
Print a  
  
End
```

INPUT

Przeznaczenie

Pozwala na wprowadzanie danych, za pomocą zewnętrznego terminala.

Składnia:

INPUT ["tekst_zachęty"] , *zmienna1* [, *zmiennan*]

gdzie:

<i>tekst_zachęty</i>	opcjonalnie - tekst pojawiający się w oknie terminala,
<i>zmienna1</i>	zmienna której przypisana zostanie odebrana wartość.
<i>zmiennan</i>	

Opis:

Instrukcja INPUT może być używana gdy system mikroprocesorowy posiada interfejs RS-232, połączony z portem COM komputera osobistego. W ten sposób za pomocą programu emulatora terminala, mogą być wprowadzane dane z klawiatury. Środowisko BASCOM posiada wbudowany emulator terminala.

Uwaga! Począwszy od wersji 1.11.6.4 wprowadzono specjalną instrukcję ECHO, kontrolującą wysyłanie echa.

Odebranie znaku <CR> (kod 13) kończy wprowadzanie danych dla bieżącej zmiennej.

Zobacz także: **INPUTHEX** , **INPUTBIN** , **PRINT** , **ECHO**

Przykład:

```
'-----  
'                               (c) 1997,1998 MCS Electronics  
'-----  
'   plik: INPUT.BAS  
'   demo: INPUT, INPUTHEX  
'-----  
'Gdy używane będą inną szybkość pracy kontrolera i UART, zmień  
'wartości w dyrektywach $BAUD =   i $CRYSTAL =  
$baud = 1200                               'spróbujmy 1200 bodów  
  
$crystal = 12000000                         '12MHz  
  
Dim V As Byte , B1 As Byte  
Dim C As Integer , D As Byte  
Dim S As String * 15                       'tylko dla uP z pamięcią XRAM  
  
Input "Użyj tej konstrukcji by wypisać tekst zachęty " , V  
Input B1                                     'opuszczamy tekst zachęty  
  
Input "Podaj liczbę typu Integer " , C  
Print C  
  
Inputhex "Podaj liczbę szesnastkową (4 bajty) " , C  
Print C  
Inputhex "Podaj liczbę szesnastkową (2 bajty) " , D  
Print D  
  
Input "Czekam na trzy dane " , C , D  
Print C ; " " ; D  
  
Echo Off  
Input C                                     'bez lokalnego echa  
Echo On  
  
Input "Podaj swoje imię " , S  
Print "Witaj " ; S  
  
Input S  
Print S  
  
End
```

INPUTBIN

Przeznaczenie:

Odczytuje dane przesyłane przez port szeregowy.

Składnia:

INPUTBIN *zmienna1* [, *zmienna2*] [, *il_bajtów*]

INPUTBIN #*kanal* , *zmienna1* [, *zmienna2*] [, *il_bajtów*]

gdzie:

<i>zmienna1</i>	dowolna zmienna do której wpisane będą odebrane znaki,
<i>zmienna2</i>	opcjonalnie - następna zmienna (i dalsze) do której wpisane będą odebrane znaki,
<i>kanal</i>	numer kanału przy korzystaniu z programowego lub sprzętowego układu UART.
<i>il_bajtów</i>	ilość wymaganych bajtów do odczytu

Opis:

Przy odbieraniu znaków przez programowy UART należy podać numer otwartego wcześniej kanału. Dotyczy to także transmisji przez drugi sprzętowy UART, występujący w niektórych kontrolerach.

Liczba odczytywanych bajtów jest określana na podstawie typu podanej zmiennej. Gdy jako parametr będzie użyta zmienna typu Byte, instrukcja odbierze jeden znak. Przy zmiennych typu Integer odczytane zostaną dwa znaki. To samo dotyczy zmiennych tablicowych – instrukcja INPUTBIN zakończy się dopiero gdy **wszystkie** elementy tablicy zostaną odebrane!

Uwaga! Instrukcja INPUTBIN nie kończy się gdy zostanie odebrany znak <CR> (kod 13), tylko gdy wszystkie bajty zostaną odebrane.

Można także sztywno określić ile bajtów ma zostać odczytanych. Może to być pomocne podczas odbioru danych trafiających do tablicy.

```
Inputbin ar(1) , 4    'do tablicy trafia 4 bajty począwszy od komórki o nr.1
```

Zobacz także: PRINTBIN

Przykład:

```
Dim a As Byte, c As Integer

Inputbin a, c          'oczekuje na 3 znaki

End
```

INPUTHEX

Przeznaczenie

Pozwala na wprowadzanie danych w zapisie szesnastkowym, za pomocą zewnętrznego terminala.

Składnia:

```
INPUTHEX ["tekst_zachęty" ] , zmienna1 [ , zmiennan ]
```

gdzie:

<i>tekst_zachęty</i>	opcjonalnie - tekst pojawiający się w oknie terminala,
<i>zmienna1</i>	zmienna której przypisana zostanie odebrana wartość,
<i>zmiennan</i>	

Opis:

Instrukcja INPUTHEX może być używana gdy system mikroprocesorowy posiada interfejs RS-232, połączony z portem COM komputera osobistego. W ten sposób za pomocą programu emulatora terminala, mogą być wprowadzane dane z klawiatury. Środowisko BASCOM posiada wbudowany emulator terminala.

Uwaga! Począwszy od wersji 1.11.6.4 wprowadzono specjalną instrukcję **ECHO**, kontrolującą wysyłanie echa.

Należy uważać by podawać tylko liczby 0-9 i znaki A-F (lub a-f) używane w zapisie szesnastkowym. Ilość przyjmowanych znaków jest różna w zależności od podanej zmiennej. Jeśli zmienna jest typu Byte pod uwagę brane są tylko dwa znaki. Gdy zmienna jest typu Integer lub Word – 4 znaki, a typ Long przyjmuje 8 znaków.

Odebranie znaku <CR> (kod 13) kończy wprowadzanie danych dla bieżącej zmiennej.

Różnice w stosunku do QBasic-a.

W języku QBasic można podać przedrostek &H co spowoduje, że instrukcja INPUT rozpozna daną w zapisie szesnastkowym.

W języku BASCOM BASIC zaimplementowano w tym celu specjalną instrukcję: INPUTHEX.

Zobacz także: **INPUT** , **ECHO**

Przykład:

```
Dim x As Byte

Inputhex "Podaj liczbę " , x      'wydrukuj tekst zachęty
```

INSTR()

Przeznaczenie:

Zwraca pozycję szukanego ciągu znaków w podanym tekście.

Składnia:

zmienna = INSTR(*start* , *tekst* , *szukany_ciąg*)

zmienna = INSTR(*tekst* , *szukany_ciąg*)

gdzie:

<i>zmienna</i>	zmienna numeryczna, której przypisana będzie pozycja szukanego ciągu w przeszukiwanym tekście,
<i>start</i>	opcjonalny parametr określający znak w tekście od którego rozpoczęte zostaną poszukiwania; normalnie poszukiwania rozpoczynają się od początku tekstu,
<i>tekst</i>	przeszukiwany tekst,
<i>szukany_ciąg</i>	szukany ciąg znaków.

Opis:

Funkcja zwraca zero, gdy poszukiwany ciąg znaków nie występuje w podanym tekście.

Uwaga! Tekst w którym poszukiwany jest ciąg znaków nie może być stałą. Tylko szukany ciąg może być stałą znakową.

Przykład:

```
Dim S As String * 10 , Z As String * 5
Dim bP As Byte

S = "To jest test"
Z = "st"

bP = Instr(S,Z) : Print bP      'szukamy st
                                'powinno być 6
bP = Instr(7,S,Z) : Print bP   'powinno być 11

End
```


INT() (Nowość w wersji 1.11.6.8)

Przeznaczenie:

Zwraca całkowitą część podanej liczby.

Składnia:

zmienna = INT(wartość)

gdzie:

zmienna	dowolna zmienna, do której wpisana będzie całkowita część podanej liczby,
wartość	liczba której całkowita część powinna być zwrócona.

Opis:

Część ułamkowa to wszystkie liczby znajdujące się po kropce z prawej strony. Część całkowita to ta znajdująca się przed kropką.

Dla przykładu liczba 1234.567 posiada część całkowitą: 1234 i ułamkową: 567.

Zobacz także: **FRAC** , **FIX** , **ROUND**

Przykład:

```
'-----  
'                                     ROUND_FIX_INT.BAS  
'-----  
  
Dim S As Single , Z As Single  
  
For S = -10 To 10 Step 0.5  
    Print S ; Spc(3) ; Round(S) ; Spc(3) ; Fix(S) ; Spc(3) ; Int(S)  
Next  
  
End
```

ISCHARWAITING() (Nowość w wersji 1.11.6.9)

Przeznaczenie:

Zwraca wartość 1 gdy w buforze transmisji sprzętowego układu UART znajduje się oczekujący znak.

Składnia:

zmienna = ISCHARWAITING()
zmienna = ISCHARWAITING(#kanał)

gdzie:

zmienna	dowolna zmienna typu Byte, Integer, Word lub Long,
kanał	liczba określająca numer otwartego kanału.

Opis:

Funkcja zwraca wartość 1 gdy w buforze odbiorczym znajduje się odebrany znak. Jeśli nie ma znaku funkcja zwraca 0.

Uwaga! Funkcja nie powoduje odebrania tego znaku, tylko sprawdza czy znak jest dostępny. Odebranie oczekującego znaku powinno się odbyć za pomocą funkcji **INKEY**.

Ponieważ funkcja **INKEY** pobiera znak z bufora sprzętowego układu UART, może ona zwrócić 0 gdy odebrany znak będzie znak o kodzie 0. Powoduje to, że nie nadaje się ona do odbierania danych binarnych, które mogą przecież zawierać bajty zerowe.

Można temu zaradzić stosując funkcję **ISCHARWAITING**, która sprawdzi czy znak rzeczywiście został odebrany. Jeśli funkcja ta zwróci 1, można odebrać znak przez **INKEY** lub

WAITKEY.

Uwaga! Funkcja nie działa jeśli skorzystano z możliwości buforowania odebranych znaków z układu UART. Zobacz **CONFIG SERIALIN**

Zobacz także: **WAITKEY** , **INKEY**

Przykład:

```
'-----  
'                                     (c) 1997-2002 MCS Electronics  
'-----  
'   plik: INKEY.BAS  
'   demo: INKEY , WAITKEY  
'-----  
Dim A As Byte , S As String * 2  
Do  
    A = Inkey()                'odczytaj kod odebranego znaku  
    's = Inkey()  
    If A > 0 Then                'odebraliśmy coś?  
        Print "Odebrano znak o kodzie ASCII " ; A  
    End If  
Loop Until A = 27                'powtarzaj aż stwierdzisz  
wciśnięcie Esc  
  
A = Waitkey()                'czekaj na klawisz  
's = waitkey()  
Print Chr(a)  
  
'czekaj aż naciśnie Esc  
Do  
Loop Until Inkey() = 27  
  
'Gdy będziesz odbierał dane binarne które będą zawierać bajty zerowe,  
'możesz zastosować funkcję ISCHARWAITING().  
'Zwróci ona 1 jeśli w buforze transmisji jest dostępny znak.  
'Możesz go potem odebrać przez INKEY() lub WAITKEY(), i być pewny że  
znak rzeczywiście został odebrany.  
  
End
```

LCASE()

Przeznaczenie:

Zamienia wszystkie znaki z ciągu na ich odpowiedniki w zestawie małych liter.

Składnia:

zmienna = **LCASE**(*ciąg*)

gdzie:

<i>zmienna</i>	zmienna typu String, do której wpisany będzie wynik działania funkcji,
<i>ciąg</i>	ciąg którego znaki należy zamienić.

Opis:

Funkcja ta nie rozpoznaje polskich znaków diakrytycznych.

Zobacz także: **UCASE**

Asembler:

Wywoływana jest procedura `_LCASE` z biblioteki `mcs.lib`. Jest generowany następujący kod (może być inny w zależności od kontrolera):

```
;##### Z = Lcase(s)
Ldi R30,$60
Ldi R31,$00          ;załadowanie stałej do rejestru
Ldi R26,$6D
Rcall _Lcase
```

Przykład:

```
Dim S As String * 12 , Z As String * 12

S = "Witaj świecie"
Z = Lcase(s)
Print Z
Z = Ucase(s)
Print Z

End
```

LCD

Przeznaczenie:

Pokazuje zawartość zmiennej lub stałej na wyświetlaczu LCD.

Składnia:

LCD x

gdzie:

x zmienna lub stała, która zostanie wyświetlona na wyświetlaczu LCD,

Opis:

Można wyświetlać więcej niż jedną zmienną, oddzielając każda z nich średnikiem:

```
LCD a ; b1 ; "jakiś tekst"
```

Instrukcja LCD zachowuje się tak samo jak instrukcja PRINT, dlatego tutaj także możliwe jest stosowanie funkcji **SPC**.

Zobacz także: **\$LCD** , **\$LCDRS** , **CONFIG LCD**

Przykład:

```
'-----
'                               (c) 2000 MCS Electronics
'-----
'   plik: LCD.BAS
'   demo: LCD, CLS, LOWERLINE, SHIFTLCD, SHIFTCURSOR, HOME
'         CURSOR, DISPLAY
'-----

'Uwaga : Testowano z wyświetlaczem podłączonym za pomocą 4-bitów

Config Lcdpin = Pin , Db4 = Portb.1 , Db5 = Portb.2 , Db6 = Portb.3 ,
Db7 = Portb.4 , E = Portb.5 , Rs = Portb.6
Rem Używając tej instrukcji można zmienić domyślne ustawienia w
opcjach kompilatora
```

```

Config Lcd = 16 * 2                                'ustawiamy typ wyświetlacza
'inne wartości: 16 * 4 , 20 * 4, 20 * 2 , 16 * 1a
'Nie umieszczenie tej instrukcji spowoduje że wybrany będzie
wyświetlacz 16 * 2
'16 * 1a to wyświetlacz 2 linie po 8 znaków z liniowym odwzorowaniem
pamięci

'$LCD = address uaktywnia tryb komunikacji z wyświetlaczem za pomocą
'          szyny danych dostępnej w systemach z pamięcią zewnętrzną.
Dim A As Byte

Cls                                                    'kasujemy LCD
Lcd "Witaj swiecie"                                   'wydrukujemy tekst w górnej linii
Wait 1
Lowerline                                             'wybieramy drugą linię
Wait 1
Lcd "  Przesuwamy  "                                'wydrukujemy to w drugiej linii

Wait 1
For A = 1 To 10
    Shiftlcd Right                                   'przesuniemy tekst w prawo
    Wait 1                                             'czekamy aż wykona
Next

For A = 1 To 10
    Shiftlcd Left                                    'przesuniemy tekst w lewo
    Wait 1                                             'czekamy aż wykona
Next

Locate 2 , 1                                         'ustawiamy pozycje kursora
Lcd "*"                                              'wydrukujemy to
Wait 1                                              'czekamy aż wykona

Shiftcursor Right                                  'przesuniemy kursor w prawo
Lcd "@"                                              'drukujemy to
Wait 1                                              'czekamy aż wykona

Home Upper                                           'wybieramy 1 linię i wracamy na
początek
Lcd "Zastapiono"                                    'zastępujemy tekst
Wait 1                                              'czekamy

Cursor Off Noblink                                  'chowamy kursor
Wait 1                                              'czekamy
Cursor On Blink                                    'pokazujemy kursor

Wait 1                                              'czekamy
Display Off                                           'wyłączamy wyświetlacz
Wait 1                                              'czekamy
Display On                                           'i włączamy z powrotem

'-----NOWOŚĆ! Obsługa czteroliniowych wyświetlaczy-----
Thirdline
Lcd "Linia 3"
Fourthline
Lcd "Linia 4"
Home Third                                           'wracamy na początek trzeciej
linii
Home Fourth

```

```

Home F                                     'skróty też działają

Locate 4 , 1 : Lcd "Linia 4"
Wait 1

'Teraz zdefiniujemy znak użytkownika
'pierwszy bajt to kod znaku (0-7)
'rezsta to postać znaku
'Użyj narzędzia LCD Designer

Deflcdchar 1 , 225 , 227 , 226 , 226 , 226 , 242 , 234 , 228
Deflcdchar 0 , 240 , 224 , 224 , 255 , 254 , 252 , 248 , 240
Cls                                     'wybieramy Data RAM
Rem Jest szczególnie ważne by po zdefiniowaniu znaków wykonać
instrukcję
Rem CLS, ponieważ przełączy to kontroler wyświetlacza na Data RAM

Lcd Chr(0) ; Chr(1)                     'drukujemy zdefiniowane znaki

'----- Teraz użyjemy wewnętrznej procedury -----
_templ = 1                             'dana do rejestru ACC
Rcall _write_lcd                        'wyświetlamy

End

```

LCDAT (Nowość w wersji 1.11.6.9)

Przeznaczenie:

Wysyła stałą lub zmienną, która ma być wyświetlona na określonej pozycji graficznego wyświetlacza LCD opartego na kontrolerze SEDxxxx.

Składnia:

LCDAT *x* , *y* , *zmienna* [, *inwersja*]

gdzie:

<i>x</i>	pozycja X, zawierająca się w granicach 0-63, kolumny w wyświetlaczu SED są rozmiaru 1 piksela,
<i>y</i>	pozycja Y,
<i>zmienna</i>	zmienna lub stała do wyświetlenia,
<i>inwersja</i>	parametr opcjonalny, 0 – oznacza normalne wyświetlanie, inna wartość spowoduje odwrócenie danych.

Opis:

Uwaga! By stosować tą instrukcję należy zadeklarować użycie biblioteki GLCDSED: `$lib "glcdsed.lbx"`

Zobacz także: **CONFIG GRAPHLCD** , **SETFONT** , **GLCDCMD** , **GLCDDATA**

LEFT()

Przeznaczenie:

Zwraca określoną liczbę znaków z tekstu począwszy od lewej strony.

Składnia:

zmienna = **LEFT**(*tekst* , *il_znaków*)

gdzie

<i>zmienna</i>	zmienna tekstowa, do której przepisane będą skopiowane
----------------	--

<i>tekst</i>	znaki,
<i>il_znaków</i>	tekst z którego skopiowane będą znaki, ilość kopiowanych znaków.

Zobacz także: **RIGHT** , **MID**

Przykład:

```
Dim s As XRAM String * 15, z As String * 15

s = "ABCDEFGH"
z = Left(s , 5)
Print z           'wydrukuj ABCDE

End
```

LEN()

Przeznaczenie:

Zwraca liczbę znaków znajdujących się w zmiennej tekstowej.

Składnia:

zmienna = **LEN**(*tekst*)

gdzie

<i>zmienna</i>	zmienna numeryczna, do której wpisana będzie długość tekstu,
<i>tekst</i>	zmienna tekstowa której długość będzie obliczona.

Opis:

Ciągi znaków w języku BASCOM BASIC mogą mieć długość do 254 znaków.

Przykład:

```
Dim S As String * 12
Dim A As Byte

S = "test"
A = Len(s)
Print A           'wydrukuj 4
Print Len(s)
```

LINE (Nowość w wersji 1.11.6.8)

Przeznaczenie:

Rysuje linię na ekranie graficznym wyświetlacza LCD.

Składnia:

LINE(*x0* , *y0*) – (*x1* , *y1*) , *kolor*

gdzie

<i>x0</i> , <i>y0</i>	współrzędne punktu początkowego linii,
<i>x1</i> , <i>y1</i>	współrzędne punktu końcowego linii,
<i>kolor</i>	kolor linii.

Zobacz także: **PSET** , **CIRCLE** , **CONFIG GRAPHLCD**

Przykład:

```
'Sposób podłączenia wyświetlacza LCD:
'końcówka          podłączona do
'1          GND          GND
'2          GND          GND
'3          +5V          +5V
'4          -9V          -9V potencjometr
'5          /WR          PORTC.0
'6          /RD          PORTC.1
'7          /CE          PORTC.2
'8          C/D          PORTC.3
'9          NC          nie podłączone
'10         RESET        PORTC.4
'11-18      D0-D7        PA
'19         FS          PORTC.5
'20         NC          nie podłączone

$crystal = 8000000
'Najpierw definiujemy, że używany będzie graficzny wyświetlacz LCD
Config Graphlcd = 240 * 128 , Dataport = Porta , Controlport = Portc ,
Ce = 2 , Cd = 3 , Wr = 0 , Rd = 1 , Reset = 4 , Fs = 5 , Mode = 8
'Dataport określa port do jakiego podłączono szynę danych LCD
'Controlport określa który port jest używany do sterowania LCD
'CE, CD itd. to numery bitów w porcie CONTROLPORT
'Na przykład CE = 2 gdyż jest podłączony do końcówki PORTC.2
'Mode = 8 daje 240 / 8 = 30 kolumn , Mode=6 gda je 240 / 6 = 40 kolumn

'Deklaracje (y nie używane)
Dim X As Byte , Y As Byte

'Kasujemy ekran wyświetlacza (tekst oraz grafikę)
Cls
'Inne opcje to:
' CLS TEXT   kasuje tylko stronę tekstową
' CLS GRAPH  kasuje tylko stronę graficzną

Cursor Off

'Teraz napiszemy jakiś tekst
Lcd "MCS Electronics"
'i jeszcze jakieś inne w linii 2
Locate 2 , 1 : Lcd "T6963c support"
Locate 16 , 1 : Lcd "A to będzie w linii 16 (najniższej)"

Wait 2
Cls Text

'Użyj nowej instrukcji LINE by narysować pudełko
'LINE(X0 , Y0) - (X1 , Y1), on/off
Line(0 , 0) -(239 , 127) , 255      ' poprzeczne linie
Line(0 , 127) -(239 , 0) , 255
Line(0 , 0) -(240 , 0) , 255        ' górna linia
Line(0 , 127) -(239 , 127) , 255    ' dolna linia
Line(0 , 0) -(0 , 127) , 255        ' lewa linia
Line(239 , 0) -(239 , 127) , 255    ' prawa linia

Wait 2
'Można także rysować linie za pomocą PSET X ,Y , on/off
```

```

'parametr on/off to: 0 - skasowanie piksela, inna - postawienie
piksela
For X = 0 To 140
    Pset X , 20 , 255          'narysuj punkt
Next

Wait 2

End

```

LOAD

Przeznaczenie:

Powoduje wpisanie wartości do rejestru licznika.

Składnia:

LOAD *licznik* , *wartość*

gdzie

<i>licznik</i>	nazwa licznika; może być: TIMER0 , TIMER1 lub TIMER2 ,
<i>wartość</i>	liczba potrzebnych impulsów.

Opis:

Instrukcja powoduje wpisanie do licznika określonej liczby zliczanych impulsów. Przed załadowaniem wartości do licznika, instrukcja dokonuje niezbędnego przeliczenia w postaci: 256-*wartość*. Tak więc instrukcja:

```
Load Timer0, 10
```

spowoduje, że do licznika trafi liczba 246, więc licznik przepełni się właśnie po 10 impulsach.

Licznik TIMER0 nie posiada trybu zliczania z funkcją automatycznego przeładowywania zawartości. Jeśli byłby on wykorzystywany do generowania periodycznych przerwań zegarowych, można użyć właśnie instrukcji LOAD.

Licznik TIMER1 jest 16 bitowy więc, przeliczenie jest wykonywane w postaci 65536-*wartość*.

LOADADR

Przeznaczenie:

Powoduje załadowanie adresu zmiennej do pary rejestrów wskaźnikowych.

Składnia:

LOADADR *zmienna* , *rejestr*

gdzie

<i>zmienna</i>	nazwa zmiennej, której adres ma być określony,
<i>rejestr</i>	nazwa specjalnego rejestru wskaźnikowego; rejestrem tym może być: X , Y lub Z .

Opis:

Instrukcję LOADADR przewidziano jako pomoc przy pisaniu wstawek assemblerowych, odwołujących się do zadeklarowanych zmiennych.

Przykład:

```

Dim S As String * 12
Dim A As Byte

```



```

$asm
  Loadadr S , X 'załaduj adres zmiennej do pary R26/R27
  Ld _templ, X 'załaduj bajt z adresu podanego R26/R27
                  'do R24(zmienna _templ)
$end asm

```

LOADLABEL() *(Nowość w wersji 1.11.6.9)*

Przeznaczenie:

Przypisuje zmiennej adres podanej etykiety.

Składnia:

`zmienna = LOADLABEL(etykieta)`

gdzie

<code>zmienna</code>	nazwa zmiennej, do której przepisany będzie adres określony podaną etykietą,
<code>etykieta</code>	nazwa etykiety.

Opis:

W niektórych przypadkach, może być potrzebny adres określonego punktu w programie. Na przykład jako adres dla funkcji **CPEEK**. Można w tym celu umieścić w tym miejscu etykietę oraz użyć instrukcji LOADLABEL w celu określenia jej adresu.

LOCAL

Przeznaczenie:

Deklaruje zmienne lokalne w procedurach lub funkcjach.

Składnia:

`LOCAL zmienna AS typ_zmiennej`

gdzie

<code>zmienna</code>	nazwa definiowanej zmiennej,
<code>typ_zmiennej</code>	typ definiowanej zmiennej.

Opis:

Zmienne lokalne mogą być typu Byte, Integer, Word, Long, Single lub String. Nie jest możliwe definiowanie lokalnych tablic, ani zmiennych typu Bit.

Zmienne zdefiniowane instrukcją LOCAL są zmiennymi tymczasowymi umieszczanymi na stosie. Po wyjściu z procedury lub funkcji, miejsce przez nie zajmowane jest zdejmowane ze stosu.

Uwaga! Argumenty **AT**, **ERAM**, **SRAM**, **XRAM** nie mogą być używane w deklaracjach zmiennych lokalnych.

Zobacz także: **DIM**

Przykład:

```

'-----
'               (c) 2000 MCS Electronics
'               DECLARE.BAS
'  Procedury działają na innej zasadzie niż w BASCOM-8051
'-----
'Na początku zadeklarujemy nagłówki procedur SUB

```

```

'Przykładowo procedura bez parametrów
Declare Sub Test2

'A teraz procedura z parametrami. Parametr pierwszy(A) nie może być
' nie może być zmieniony w procedurze. Parametr drugi(B1) może być
' zmieniony w treści procedury.
' Kiedy jest określony argument BYVAL zawartość parametru jest
' przekazywana procedurze.
' Kiedy jest określony argument BYREF lub nie występuje żaden z nich,
' wtedy do procedury przekazywany jest adres parametru, co oznacza, że
' może być zmieniony w procedurze.
Declare Sub Test(byval A As Byte , B1 As Byte)

'Parametry podprogramów mogą być dowolnego typu, oprócz typu Bit.
'Zmienne bitowe są widoczne w całym programie, gdyż są to zmienne
globalne.
Declare Sub Testvar(b As Byte , I As Integer , W As Word , L As Long ,
S As String)

Dim Bb As Byte , I As Integer , W As Word , L As Long , S As String *
10                                'zdefiniujemy zmienne
Dim Ar(10) As Byte

Call Test2                                'wywołujemy procedurę
Test2                                'lub to samo bez CALL
'Uwaga! W procedurach z parametrami przy stosowaniu wywołania bez CALL
'nie należy umieszczać parametrów w nawiasach.
Bb = 1
Call Test(1 , Bb)                        'wywołujemy procedurę z
parametrami
Print Bb                                'drukujemy zmieniony parametr

'przetestujemy wszystkie możliwe typy zmiennych

Call Testvar(bb , I , W , L , S )
Print Bb ; I ; W ; L ; S

'teraz przekazujemy tablicę
'musi być przekazana jako adres (zobacz BYREF)
Test 2 , Ar(1)
Print "ar(1) = " ; Ar(1)

End

'Koniec programu, teraz treść procedur
'Uwaga! Parametry muszą być takie same jak zadeklarowane!

Sub Test(byval A As Byte , B1 As Byte)    'początek procedury
Print A ; " " ; B1                        'wydrukujemy wartości
parametrów
    B1 = 3                                'teraz coś zmienimy

    'Możesz zmienić zawartość A, lecz nie da to żadnego efektu poza
    procedurą,
    'gdyż podczas przekazywania parametru została utworzona zmienna
    tymczasowa
End Sub

Sub Test2                                'procedura bezparametrowa
Print "Nie ma parametrów"
End Sub

```

```

Sub Testvar(b As Byte , I As Integer , W As Word , L As Long , S As
String)
    Local X As Byte
    X = 5                                     'przypisujemy wartość
zmiennej lokalnej
    B = X
    I = -1
    W = 40000
    L = 20000
    S = "test"
End Sub

```

LOCATE

Przeznaczenie:

Przesuwa kursor wyświetlacza LCD na określoną pozycję.

Składnia:

LOCATE y , x

gdzie

x	pozycja kursora w linii; można podać pozycję z zakresu od 1 do 64 - zależnie od posiadanego wyświetlacza,
y	numer linii wyświetlacza; może być z zakresu 1 - 4 – zależnie od typu używanego wyświetlacza.

Zobacz także: **CONFIG LCD** , **LCD** , **HOME** , **CLS**

Przykład:

```

Lcd "Witaj"
Locate 1,10
Lcd "*"

```

LOG()

Przeznaczenie:

Zwraca wartość logarytmu naturalnego podanej liczby.

Składnia:

zmienna = **LOG**(*liczba*)

gdzie

<i>zmienna</i>	dowolna zmienna typu Single, do której wpisany będzie wynik działania funkcji,
<i>liczba</i>	liczba lub zmienna której logarytm należy obliczyć.

Zobacz także: **EXP** , **LOG10**

Przykład: Zobacz temat **Biblioteka FP_TRIG**

LOG10() *(Nowość w wersji 1.11.6.8)*

Przeznaczenie:

Zwraca wartość logarytmu dziesiętnego (o podstawie 10) podanej liczby.

Składnia:

zmienna = **LOG10**(*liczba*)

gdzie

<i>zmienna</i>	dowolna zmienna typu Single, do której wpisany będzie wynik działania funkcji,
<i>liczba</i>	liczba lub zmienna której logarytm należy obliczyć.

Zobacz także: **EXP** , **LOG**

Przykład: Zobacz temat **Biblioteka FP_TRIG**

LOOKDOWN()

Przeznaczenie:

Przeszukuje ciąg danych w poszukiwaniu określonej wartości.

Składnia:

zmienna = **LOOKDOWN**(*wartość* , *etykieta* , *il_danych*)

gdzie

<i>zmienna</i>	dowolna zmienna do której trafi indeks odnalezionej wartości,
<i>wartość</i>	zmienna której zawartość należy odnaleźć w danych,
<i>etykieta</i>	etykieta określająca adres linii DATA,
<i>il_danych</i>	liczba elementów w liniach DATA, poddanych przeszukiwaniu.

Opis:

Funkcja LOOKDOWN przeszukuje dane, umieszczone w liniach **DATA**, sprawdzając czy nie znajduje się tam podana liczba. Jeśli liczba ta zostanie odnaleziona, funkcja zwróci indeks to tego elementu. Jeśli jednak podana liczba nie znajduje się w liniach DATA, funkcja zwróci -1.

Uwaga! Ważna jest zgodność typów. Gdy dane zawarte w liniach DATA są typu Integer lub Word, zmienna określająca wartość musi być także tego typu. Dotyczy to w takim samym stopniu danych typu Byte oraz Long.

Zobacz także: **LOOKUPSTR** , **LOOKUP**

Przykład:

```
' -----
'                               LOOKDOWN.BAS
'                               (c) 2001 MCS Electronics
' -----

Dim Idx As Integer , search As Byte , entries As Byte

'będziemy szukać liczby 3
Search = 3
'jest 5 danych w linii DATA
Entries = 5

'przeszukaj i zwróć indeks
Idx = Lookdown(search , Label , Entries)
Print Idx

Search = 1
Idx = Lookdown(search , Label , Entries)
Print Idx
```

```

Search = 100
Idx = Lookdown(search , Label , Entries)
Print Idx                                'wydrukuje -1 jeśli nie znalazł

'Przeszukiwanie danych typu Integer lub Word wymaga by szukana
'liczba była także typu Integer lub Word
Dim Isearch As Integer

Isearch = 400
Idx = Lookdown(isearch , Label2 , Entries)
Print Idx                                'wydrukuje 3

End

Label1:
  Data 1 , 2 , 3 , 4 , 5

Label2:
  Data 1000% , 200% , 400% , 300%

```

LOOKUP()

Przeznaczenie:

Zwraca wartość wybranego elementu w liniach DATA.

Składnia:

zmienna = **LOOKUP**(*nr_elementu* , *etykieta*)

gdzie

<i>zmienna</i>	zmienna, której przypisana będzie wartość podanego elementu,
<i>nr_elementu</i>	numer elementu,
<i>etykieta</i>	etykieta określająca adres danych umieszczonych w liniach DATA.

Opis:

Elementy są numerowane od zera. Numer elementu nie może być większy niż 65535.

Zobacz także: LOOKUPSTR

Przykład:

```

Dim b1 As Byte , I As Integer

b1 = Lookup(1, DTA)
Print b1                                'wydrukuje 2 (elementy liczone od zera)

I = Lookup(0, DTA2)
End

DTA:
  Data 1,2,3,4,5

DTA2:                                'dane typu Integer
  Data 1000% , 2000%

```

LOOKUPSTR()

Przeznaczenie:

Zwraca wartość wybranego tekstu umieszczonego w liniach DATA.

Składnia:

zmienna = LOOKUPSTR(*nr_elementu* , *etykieta*)

gdzie

<i>zmienna</i>	zmienna typu String, której przypisana będzie wartość podanego elementu,
<i>nr_elementu</i>	numer elementu,
<i>etykieta</i>	etykieta określająca adres danych umieszczonych w liniach DATA.

Opis:

Elementy są numerowane od zera. Numer elementu nie może być większy niż 255.

Zobacz także: LOOKUP

Przykład:

```
Dim s As String, idx As Byte

idx = 3 : s = Lookupstr(idx, Sdata)
Print s                                     'wydrukuj 'tylko'

End

Sdata:
Data "To" , "jest" , "tylko" , "test"
```

LOW()

Przeznaczenie:

Zwraca młodszą część (bajt MSB) podanej zmiennej.

Składnia:

rezultat = LOW(*zmienna*)

gdzie:

<i>rezultat</i>	zmienna do której zapisana będzie młodszy bajt zmiennej,
<i>zmienna</i>	zmienna której młodsza część ma być określona.

Opis:

Zobacz także: HIGH , HIGHW

Przykład:

```
Dim I As Integer , Z As Byte

I = &H1001
Z = Low(I)                       'zwróci 1
```

LOWERLINE

Przeznaczenie:

Ustawia kursor wyświetlacza LCD w drugiej linii.

Składnia:

LOWERLINE

Opis:

Instrukcja ustawia kursor wyświetlacza na początek dolnej linii (wyświetlacze 2 liniowe) lub w drugiej linii (wyświetlacze 4 liniowe).

Zobacz także: **HOME** , **UPPERLINE** , **THIRDLINE** , **FOURTHLINE** , **LOCATE**

Przykład:

```
$config Lcd = 16 * 2

Dim a As byte
a = 255
Lcd a
Lowerline
Lcd a

End
```

LTRIM()

Przeznaczenie:

Obcina wiodące spacje znajdujące się z lewej strony.

Składnia:

zmienna = **LTRIM**(*tekst*)

gdzie:

<i>zmienna</i>	zmienna tekstowa, do której zapisany będzie rezultat działania funkcji,
<i>tekst</i>	tekst z którego usunięte mają być wiodące spacje.

Zobacz także: **RTRIM** , **TRIM**

Przykład:

```
Dim S As String * 6

S = "  AB "
Print Ltrim(s)
Print Rtrim(s)
Print Trim(s)

End
```

MAKEBCD()

Przeznaczenie:

Dokonyuje konwersji liczby na format BCD.

Składnia:

zmienna1 = **MAKEBCD**(*zmienna2*)

gdzie:

<i>zmienna1</i>	zmienna, do której zapisany będzie rezultat działania funkcji,
-----------------	--

zmienna2 zmienna zawierająca liczbę dziesiętną.

Opis:

Większość zegarów czasu rzeczywistego (np. PCF8535) korzysta z formatu BCD. Dlatego podczas ustawiania jego rejestrów należy najpierw przekształcić daną dziesiętną na BCD.

Aby wydrukować liczbę w formacie BCD najlepiej użyć funkcji BCD(), która powoduje przekształcenie jej wartości na postać tekstu.

Zobacz także: [MAKEDEC](#) , [BCD](#)

Przykład:

```
Dim a As Byte

a = 65
Lcd a
Lowerline
Lcd Bcd(a)
a = Makebcd(a)
Lcd " " ; a

End
```

MAKEDEC()

Przeznaczenie:

Dokonyje konwersji liczby w formacie BCD na jej postać dziesiętną.

Składnia:

zmienna1 = MAKEDEC(*zmienna2*)

gdzie:

<i>zmienna1</i>	zmienna, do której zapisany będzie rezultat działania funkcji,
<i>zmienna2</i>	zmienna zawierająca liczbę w formacie BCD.

Opis:

W niektórych przypadkach wymagana jest konwersja odwrotna z formatu BCD na postać dziesiętną. Jest to szczególnie użyteczne przy sterowaniu dekodery wyświetlaczy 7-segmentowych, czy odczytywaniu danych z rejestrów zegarów RTC.

Zobacz także: [MAKEBCD](#)

Przykład:

```
Dim a As Byte

a = 65
Print a
Lowerline
Print Bcd(a)
a = Makedec(a)
Print Spc(3) ; a

End
```

MAKEINT()

Przeznaczenie:

Łączy dwa bajty w zmienną typu Integer lub Word.

Składnia:

zmienna = **MAKEINT**(*część_LSB* , *część_MSB*)

gdzie:

<i>zmienna</i>	zmienna, do której zapisany będzie rezultat działania funkcji,
<i>część_LSB</i>	bajt stający się częścią LSB zmiennej wynikowej,
<i>część_MSB</i>	bajt stający się częścią MSB zmiennej wynikowej.

Opis:

Funkcja ta odpowiada następującej operacji matematycznej: *zmienna* = (256 * *część_MSB*) + *część_LSB*.

Zobacz także: **LOW** , **HIGH**

Przykład:

```
Dim a As Integer, I As Integer

a = 2
I = Makeint(a , 1)      'I = (1 * 256) + 2 = 258

End
```

MAX()

Przeznaczenie:

Zwraca największą wartość znalezioną w tablicy z elementami typu Word.

Składnia:

zmienna1 = **MAX**(*zmienna2*)
MAX(*tablica*(1) , *max* , *indeks*)

gdzie:

<i>zmienna1</i>	zmienna, do której zapisana będzie odnaleziona maksymalna wartość,
<i>zmienna2</i>	adres pierwszego elementu tablicy,
<i>tablica</i>	nazwa zmiennej tablicowej,
<i>max</i>	zmienna do której wpisana będzie odnaleziona maksymalna wartość,
<i>indeks</i>	numer elementu tablicy gdzie odnaleziono maksymalną wartość.

Opis:

Funkcja MAX występuje w dwóch formach. Pierwsza z nich odnajduje tylko samą maksymalną wartość i działa jak normalna funkcja. Drugi format pozwala oprócz odnalezienia maksymalnej wartości, określenie w której komórce tablicy się ona znajduje.

Jeśli wartość maksymalna nie występuje (np. gdy elementy tablicy są sobie równe) funkcja zwraca indeks równy 0.

Zobacz także: **MIN**

Przykład:

```
' -----
'                                     (c) 2001-2002 MCS
'                                     minmax.bas
' Przykład ten pokazuje działanie funkcji MIN i MAX
```

```

' Funkcje te działają tylko dla tablic złożonych z typu Word!!!!
'-----
'Określamy zmienne
Dim Wb As Word , B As Byte
Dim W(10) As Word

'wypełniamy tablicę wartościami od 1 do 10
For B = 1 To 10
    W(b) = B
Next

Print "Max liczba " ; Max(w(1))
Print "Min liczba " ; Min(w(1))

End

```

MID()

Przeznaczenie:

Funkcja MID() zwraca określony fragment tekstu.

Składnia:

zmienna = MID(*tekst* , *start* [, *ilość*])

gdzie:

<i>zmienna</i>	zmienna, do której zapisany będzie rezultat działania funkcji,
<i>tekst</i>	tekst którego fragment będzie zwrócony,
<i>start</i>	pozycja w tekście od której tekst będzie zwrócony,
<i>ilość</i>	parametr opcjonalny - ilość zwracanych znaków.

Opis:

Gdy nie jest podana ilość znaków, funkcja zwraca wszystkie znaki od pozycji startowej do końca tekstu.

Zobacz także: LEFT , RIGHT , MID

Przykład:

```

Dim s As XRAM String * 15, z As XRAM String * 15

s = "ABCDEFGH"
z = Mid(s, 2, 3)
Print z           'BCD
z="12345"
Mid(s, 2, 2) = z
Print s           'A12DEFG

End

```

MID

Przeznaczenie:

Instrukcja MID zamienia określoną część tekstu.

Składnia:

MID(*tekst* , *start* [, *ilość*]) = *zmienna*

gdzie:

<i>zmienna</i>	zmienna zawierająca wstawiany ciąg,
----------------	-------------------------------------

<i>tekst</i>	tekst na którym przeprowadzone będą operacje,
<i>start</i>	pozycja w tekście od której ma być wstawiony ciąg,
<i>ilość</i>	parametr opcjonalny, ilość wstawianych znaków.

Opis:

Gdy nie jest podana ilość znaków, instrukcja wstawia wszystkie znaki znajdujące się w zmiennej.

Zobacz także: **LEFT** , **RIGHT** , **MID()**

Przykład:

```
Dim s As XRAM String * 15, z As XRAM String * 15

s = "ABCDEFGH"
z = Mid(s, 2, 3)
Print z           'BCD
z = "12345"
Mid(s, 2, 2) = z
Print s           'A12DEFG

End
```

MIN()

Przeznaczenie:

Zwraca najmniejszą wartość znaną w tablicy z elementami typu Word.

Składnia:

```
zmienna1 = MIN( zmienna2 )
MIN( tablica(1) , min , indeks )
```

gdzie:

<i>zmienna1</i>	zmienna, do której zapisana będzie odnaleziona minimalna wartość,
<i>zmienna2</i>	adres pierwszego elementu tablicy,
<i>tablica</i>	nazwa zmiennej tablicowej,
<i>min</i>	zmienna do której wpisana będzie odnaleziona minimalna wartość,
<i>indeks</i>	numer elementu tablicy gdzie odnaleziono minimalną wartość.

Opis:

Funkcja MIN występuje w dwóch formach. Pierwsza z nich odnajduje tylko samą minimalną wartość i działa jak normalna funkcja. Drugi format pozwala oprócz odnalezienia minimalnej wartości, określić w której komórce tablicy się ona znajduje.

Jeśli wartość minimalna nie występuje (np. gdy elementy tablicy są sobie równe) funkcja zwraca indeks równy 0.

Zobacz także: **MAX**

Przykład:

```
'-----
'                                     (c) 2001-2002 MCS
'                                     minmax.bas
' Przykład ten pokazuje działanie funkcji MIN i MAX
' Funkcje te działają tylko dla tablic złożonych z typu Word!!!!
'-----
'Określamy zmienne
```

```

Dim Wb As Word , B As Byte
Dim W(10) As Word

'wypełniamy tablicę wartościami od 1 do 10
For B = 1 To 10
    W(b) = B
Next

Print "Max liczba " ; Max(w(1))
Print "Min liczba " ; Min(w(1))

End

```

ON INTERRUPT

Przeznaczenie:

Wykonuje skok do podprogramu gdy wystąpiło określone przerwanie.

Składnia:

ON *źródło_przerwania* *nazwa_podprogramu* [**NOSAVE**]

gdzie:

źródło_przerwania symboliczna nazwa źródła przerwania,
nazwa_podprogramu etykieta określająca gdzie znajduje się podprogram obsługi przerwania.

Opis:

Gdy wystąpi określone przerwanie – określone symbolicznie po słowie ON – instrukcja skacze do podanego podprogramu.

Język BASCOM BASIC rozpoznaje następujące źródła przerwania (po przecinku podano nazwy w konwencji stosowanej przy opisie procesorów AVR):

Tabela 13 Lista źródeł przerwania dla instrukcji ON Interrupt.

Nazwa	Źródło
INT0	przerwanie zewnętrzne końcówka INT0
INT1	przerwanie zewnętrzne końcówka INT1
INT2	przerwanie zewnętrzne końcówka INT2 (tylko niektóre kontrolery)
INT3	przerwanie zewnętrzne końcówka INT3 (tylko niektóre kontrolery)
INT4	przerwanie zewnętrzne końcówka INT4 (tylko niektóre kontrolery)
INT5	przerwanie zewnętrzne końcówka INT5 (tylko niektóre kontrolery)
TIMER0, OVFO	przerwanie przepełnienia licznika TIMER0
TIMER1, OVFI	przerwanie przepełnienia licznika TIMER1
TIMER2, OVFI	przerwanie przepełnienia licznika TIMER2 (tylko niektóre kontrolery)
ADC, ADSC	przerwanie przetwornika A/D (tylko niektóre kontrolery)
EEPROM, ERDY	przerwanie - gotowość wbudowanej pamięci EEPROM
CAPTURE1, ICP1	przerwanie rejestru przechwycenia licznika TIMER1
COMPARE1A, OC1A	przerwanie pierwszego rejestru porównania licznika TIMER1
COMPARE1B, OC1B	przerwanie drugiego rejestru porównania licznika TIMER1
COMPARE1	przerwanie rejestru porównania licznika TIMER1
OC2	przerwanie rejestru porównania licznika TIMER2 (tylko niektóre kontrolery)
SPI	przerwanie interfejsu SPI

Nazwa	Źródło
URXC	przerwanie układu sprzętowego UART – odebranie znaku
UTXC	przerwanie układu sprzętowego UART – wysłanie znaku
UDRE	przerwanie układu sprzętowego UART – bufor pusty
ACI	przerwanie wewnętrznego komparatora

Ilość i źródła przerw są ściśle związane z budową użytego procesora AVR. Nazwy możliwych przerw można znaleźć w pliku rejestrów wybranego procesora. Na przykład w pliku 2313def.dat podano, że istnieje źródło przerwania nazwane COMPARE1 (patrz na końcu pliku).

Procedura obsługi musi się kończyć instrukcją **RETURN**, co spowoduje, że na końcu procedury zostanie umieszczona instrukcja maszynowa RETI.

Instrukcja RETURN może występować w kilku miejscach podprogramu. W przypadku podprogramów obsługi przerw **tylko** pierwsza instrukcja RETURN zostanie przetłumaczona na sekwencję powrotu z przerwania (tj. odtworzenie stanu rejestrów i wykonanie RETI). Przy czym nie może ona występować w połączeniu z instrukcją warunkową. Wszystkie dalsze instrukcje RETURN zostaną skompilowane do zwykłego RET.

Określenie parametru **NOSAVE** spowoduje, że zawartość rejestrów nie będzie zapamiętana przed wejściem do procedury przerwania i odtworzona na jej końcu. Dlatego też, gdy podano **NOSAVE** należy zapewnić by zawartość rejestrów została zapamiętana i odtworzona. Służą do tego instrukcje **PUSHALL** i **POPALL**.

Gdy parametr **NOSAVE** nie występuje, zawartość wszystkich rejestrów zostanie automatycznie zapamiętana i odtworzona. Dotyczy to tylko rejestrów użytkowych R31-R16, R11-R10 i rejestru SREG.

Jeśli w procedurze obsługi przerwania wykonywane są operacje na liczbach typu Single (nie zalecane!), należy zapamiętać a potem odtworzyć zawartość rejestrów R12-R15. Na przykład:

```

Moj_ISR:
    Push R12          'zapamiętaj rejestry na stosie
    Push R13
    Push R14
    Push R15
    Single = single + 1 'używamy operacji zmiennoprzecinkowej
    Pop R15           'odtworzymy zawartość rejestrów
    Pop R14
    Pop R13
    Pop R12
    Return

```

Zrozumieć przerwanie

By przybliżyć idee działania przerw, można sobie wyobrazić następującą sytuację: W dość dużym domu z ogrodem jest zatrudniona osoba zajmująca się pracą w ogrodzie jak i w roli kamerdynera. Kamerdyner gdy nie ma co robić strzyże żywopłot lub pracuje w ogrodzie. Gdy ktokolwiek zadzwoni do drzwi frontowych obowiązkiem kamerdynera jest zostawienie nożyc w ogrodzie, zrzucenie ubrania roboczego i przystąpienie do obowiązków przyjmowania gości.

Gdy wszystkie grzeczności zostaną już zakończone, kamerdyner znów wraca do miejsca w którym przerwał pracę ogrodnika i zostawił nożyce.

W powyższym przykładzie (*przytoczonym przez tłumacza*) źródłem przerwania jest dzwonek u drzwi, a procedurą obsługi są wszystkie działania kamerdynera począwszy od usłyszenia sygnału dzwonka, a skończywszy na powrocie do pracy ogrodnika.

W przełożeniu na język BASCOM BASIC, wygenerowanie sygnału przerwania spowoduje wstrzymanie działania programu głównego, gdyż procesor przechodzi do wykonywania procedury obsługi przerwania. Po zakończeniu procedury procesor wraca do miejsca w którym zatrzymał program główny. Można na przykład stworzyć pustą pętlę **DO..LOOP**, która będzie przerywana, gdy na określonej końcówce pojawi się stan niski.

Najlepszym przykładem wykorzystania przerw jest program zegara. Można zaprogramować

któryś z liczników, by co 10ms generował przerwanie. Procedura jego obsługi będzie zliczać te impulsy. Naliczenie 100 impulsów oznacza, że upłynęła właśnie jedna sekunda i trzeba licznik sekund zwiększyć. Gdy przepełni się licznik sekund (>59), zwiększany będzie licznik minut, itd.

Program główny będzie się zajmował tylko obsługą klawiatury i wyświetlacza, nie zważając na to co robi procedura przerwania.

Używanie przerw w języku BASCOM BASIC.

By używać przerw we własnych programach należy obowiązkowo włączyć globalny system przerw instrukcją **ENABLE INTERRUPTS**, oraz włączyć odpowiednie źródło przerwania. Na przykład:

```
Enable Timer0
```

spowoduje, że będą generowane przerwania przepełnienia licznika TIMER0. Instrukcją **DISABLE** można wyłączyć system przerw jak i poszczególne ich źródła.

Gdy procesor stwierdzi, że wystąpiło przerwanie, skacze pod ustalony adres, umieszczony na początku pamięci FlashROM. Adresy te i powiązane z nimi źródła przerw są zdefiniowane przez projektantów z firmy Atmel. Przeglądając plik .DAT można dowiedzieć się, o które adresy dokładnie chodzi.

Normalnie pod tymi adresami kompilator BASCOM BASIC umieszcza instrukcje RETI, więc po wygenerowaniu przerwania, nic się nie dzieje i procesor natychmiast wraca do programu głównego.

Gdy jednak w programie umieszczona będzie instrukcja ON INTERRUPT, wtedy pod odpowiednim adresem procedury obsługi przerwania zostanie umieszczony skok do podprogramu, którego adres określa podana etykieta.

Przed wykonaniem jednak zawartego pod podaną etykietą kodu, zostanie zapamiętana zawartość rejestru SREG i rejestrów użytkowych. Zostaną one później przed wykonaniem właściwej instrukcji RETURN odtworzone. Chodzi o to, by po powrocie z przerwania stan procesora był dokładnie taki sam jak tuż przed wystąpieniem przerwania. Inaczej mogłoby dojść do zakłóceń w działaniu programu głównego. Zakończyło by się to pewnie – w żargonie programistów - „pójściem w maliny” programu.

Gdy procesor jest w trakcie wykonywania procedury przerwania, następne przerwanie nie będzie przyjęte, gdyż procesor (nie kompilator!) zeruje flagę globalnego zezwolenia na przerwania. Tak samo flaga bieżącego przerwania (tego, którego procedura obsługi jest wykonywana) zostaje automatycznie wyzerowana. Po zakończeniu przerwania flaga globalnego zezwolenia jest na powrót ustawiana i procesor może przyjąć następne przerwanie.

W procesorach AVR nie jest zatem możliwe swobodne ustawienie priorytetu źródeł przerw. Standardowo priorytet jest ustawiany według adresów programów obsługi. Dlatego przerwania o niższym adresie stoją wyżej w hierarchii (mają wyższy priorytet)!

Wskazówki dotyczące obsługi przerw

- Gdy wykorzystano przerwania w celu generacji impulsów, np. co 10µs, należy się upewnić czy procedura obsługi będzie w stanie zakończyć się w czasie krótszym niż 10µs. Inaczej można doprowadzić do zapętlenia się przerw!
- Polecane jest utworzenie flagi (widocznej w programie głównym), która byłaby ustawiana w trakcie działania podprogramu przerwania i zerowana przed wyjściem z procedury. Pozwala to na podanie parametru **NOSAVE**, co w rezultacie spowoduje mniejsze rozrastanie się stosu oraz zredukuje użytą pamięć programu. Należy jednak zadbać by zapamiętać i odtworzyć zawartość rejestrów R24 i SREG.

Zobacz także: **ENABLE** , **DISABLE**

Przykład:

```
Enable Interrupts  
Enable Int0           'włączenie przerwania
```

```

On Int0 Label2 Nosave      'nastąpi skok do Label2 gdy wystąpi
przerwanie
Do                          'nieskończona pętla
Loop

End

Label2:
Dim a As Byte
If a > 1 Then
    Return                  'zastąpione będzie przez RET, gdyż
                            'jest powiązanie z instrukcją IF
End if

Return                      'zostanie przetłumaczone na RETI, gdyż
                            'jest to pierwsza instrukcja RETURN

Return                      'zostanie przetłumaczone na RET, gdyż
                            'jest to następna instrukcja RETURN

```

ON VALUE

Przeznaczenie:

Wykonuje skok do jednej z podanych etykiet, w zależności od zawartości zmiennej.

Składnia:

ON *zmienna* [**GOTO** | **GOSUB**] *etykieta1* [, *etykietaN*] [**CHECK**]

gdzie:

<i>zmienna</i>	testowana zmienna,
<i>etykieta1</i> ,	etykieta lub lista etykiet określająca miejsca skoków.
<i>etykietaN</i>	

Opis:

Jako *zmienna* można podać także któryś z rejestrów specjalnych np.: PORTB.

Uwaga! Skok do pierwszej etykiety na liście nastąpi gdy zmienna będzie miała wartość zero.

Stosowanie parametru **CHECK** pozwala na zabezpieczenie się przed błędnym działaniem ON VALUE. Jeśli zatem wartość przekazana przez *zmienną* będzie większa niż liczba podanych *etykiet* (pamiętaj o numeracji od zera!) wtedy program zachowa się tak jakby ON VALUE nie było i wykonana zostanie następna instrukcja.

Asembler:

Instrukcja ON..GOTO w procesorach nie należących do rodziny AVR MEGA jest tłumaczona na poniższy kod:

```

Ldi R26,$60                ;adres zmiennej
Ldi R27,$00                ;load constant in register
Ld R24,X
Clr R25
Ldi R30, Low(ON_1_ * 1)    ;rejestr Z zawiera adres etykiety
Ldi R31, High(ON_1_ * 1)
Add z1,r24                 ;dodaj offset

Adc zh,r25
Ijmp                       ;skocz pod adres zawarty w rej. Z
ON_1_:
Rjmp lb11                  ;tablica skoków
Rjmp lb12

```

Rjmp lb13

Instrukcja ON..GOSUB w procesorach nie należących do rodziny AVR MEGA jest tłumaczona następująco:

```
##### On X Gosub L1 , L2
  Ldi R30,Low(ON_1_EXIT * 1)
  Ldi R31,High(ON_1_EXIT * 1)
  Push R30
;umieść adres powrotu
  Push R31
  Ldi R30,Low(ON_1_ * 1)
;załaduj adres tabeli skoków
  Ldi R31,High(ON_1_ * 1)

  Ldi R26,$60
  Ld R24,X
  Clr R25
  Add z1,r24
;dodaj offset do adresu tabeli skoków
  Adc zh,r25
  Ijmp                                     ;instrukcja skoku!!!
ON_1_:
  Rjmp L1
  Rjmp L2
ON_1_EXIT:
```

Jak można zauważyć jako wywołanie podprogramu używana jest instrukcja skoku. Wcześniej jednak jest zapamiętywany adres powrotny na stosie.

Przykład:

```
x = 2
On x Gosub lb11, lb12, lb13      'przypisujemy wartość
x = 0                           'nastąpi zatem skok do lb13
On x Goto lb11, lb12, lb13

End

lb13:
  Print "lb13"
Return

lb11:
  Print "lb11"

lb12:
  Print "lb12"
```

OPEN

Przeznaczenie:

Otwiera kanał urządzenia typu UART.

Składnia:

OPEN "urządzenie" FOR tryb AS #kanał

gdzie:

<i>urządzenie</i>	nazwa urządzenia,
<i>tryb</i>	tryb pracy urządzenia,

kanal numer przypisanego urządzeniu kanału.

Opis:

Język BASCOM BASIC oferuje oprócz wbudowanych w procesor układów UART, także programowy UART. Posiada on kilka wirtualnych kanałów transmisji, mogących być używanych niezależnie. Każdy z kanałów przed użyciem musi zostać otwarty, a po zakończeniu transmisji zamknięty.

Domyślnym urządzeniem transmisyjnym jest wbudowany w mikroprocesor układ transmisji szeregowej, nazywany COM1. Jest on automatycznie otwierany, tak więc może być używany przez instrukcje **INPUT** oraz **PRINT**, bez potrzeby używania instrukcji **OPEN**.

Programowy UART, korzysta z dowolnie wybranych końcówek portów, gdyż transmisją zajmuje się procesor. Dlatego w instrukcji **OPEN** należy określić, która z końcówek portu ma być użyta jako linia transmisyjna. Ponadto należy także określić format przesyłania danych. Na przykład, podanie jako nazwy urządzenia:

```
COMB.0:9600,8,N,2
```

informuje, że linią transmisyjną stanie się końcówka PORTB.0; transmisja odbywać się będzie z szybkością 9600 bodów; 8 bitów danych; bez bitu parzystości oraz z 2 bitami stopu.

Ogólnie format nazwy urządzenia wygląda następująco:

```
COMpin:szybkość,8,N,bity_stopu
```

gdzie:

<i>pin</i>	nazwa końcówki portu mającego pełnić rolę linii transmisyjnej: B.x lub D.x ; gdzie x określa umowny bit portu,
<i>szybkość</i>	szybkość transmisji w bodach,
<i>bity_stopu</i>	1 lub 2 bity stopu.

Jest także możliwe ustawienie trybu transmisji jako 7 bitowej, przy czym ostatni bit wykorzystywany jest wtedy jako bit parzystości:

COMpin:szybkość,7,O,bity_stopu	bit parzystości określa nieparzystość (ODD)
COMpin:szybkość,7,E,bity_stopu	bit parzystości określa parzystość (EVEN)

Można dodać także parametr opcjonalny: **INVERTED**, który spowoduje że nastąpi odwrócenie polaryzacji (zanegowanie) sygnałów programowego urządzenia UART. Na przykład instrukcja:

```
Open "COMD.1:9600,8,N,1,INVERTED" For Output As #1
```

spowoduje, że końcówka PORTD.1 stanie się wyjściem programowego urządzenia UART. Transmisja będzie przebiegać w trybie 8 bitowym, z prędkością 9600 bodów i 1 bitem stopu. Nastąpi także odwrócenie polaryzacji sygnału.

Można także otwierać kanały dla sprzętowego układu UART, zwłaszcza że niektóre z kontrolerów posiadają dwa takie układy. Format nazwy urządzenia dla sprzętowego UART-u to:

COM1 :	dla pierwszego układu UART,
COM2 :	dla drugiego układu UART,

Szybkość pracy łącza jest określany dyrektywami **\$BAUD** dla COM1: oraz **\$BAUD1** dla COM2:. Można także lokalnie zmieniać częstotliwość w trakcie działania programu stosując instrukcje: **BAUD**.

Uwaga! Nazwa **COM2**: jest dostępna tylko jeśli określony zostanie odpowiedni procesor w opcjach kompilatora lub za pomocą dyrektywy **\$REGFILE**.

Tryb pracy dla sprzętowego urządzenia COM1/COM2 może być określony następująco (Dwa poniższe tryby nie mają aktualnie znaczenia. Tryby dodano ze względów kompatybilności z językiem QBasic *przyp. tłumacza*):

BINARY
RANDOM

Dla urządzenia programowego UART, możliwe są tylko tryby (Muszą występować gdyż korzystamy tylko z jednej linii transmisyjnej *przyp. tłumacza*):

INPUT	kanal będzie używany jako wejściowy,
OUTPUT	kanal będzie używany jako wyjściowy.

Instrukcje korzystające z kanałów to: **PRINT**, **INPUT**, **INPUTHEX**, **INKEY** oraz **WAITKEY**.

Uwaga! Instrukcja **INPUT#**, korzystająca z otwartego kanału programowego urządzenia UART wysyła echo, gdyż nie ma domyślnych ustawień dotyczących linii sygnałowych.

Zobacz także: **CLOSE** , **CRYSTAL**

Przykład:

```
'-----  
'          (c) 2000 MCS Electronics  
'          OPEN.BAS  
'          demonstruje programowy UART  
'-----  
$crystal = 10000000          'zmień jeśli używasz innego kwarcu  
  
Dim B As Byte  
  
'Opcjonalnie można dostroić szybkość transmisji.  
'Dlaczego powinno się to zrobić?  
'Niektóre procesory mają wbudowany generator który może nie  
'pracować na wymaganej częstotliwości.  
'Zależy ona od napięcia zasilania, temperatury itp.  
'Można oczywiście zmienić wartość podaną w dyrektywie $CRYSTAL lub  
używając:  
'BAUD #1,9610  
  
'W tym przykładzie jest używana płytką DT006 (www.simmstick.com)  
'Pozwala to na łatwe przetestowanie z wykorzystaniem istniejącego  
'portu szeregowego.  
'Używany jest tam układ MAX232.  
'Ponieważ używane są końcówki z których korzysta sprzętowy UART  
'NIE MOŻE on być używany.  
'Sprzętowy UART jest używany wraz z instrukcjami PRINT, INPUT i  
innymi.  
  
'Używamy zatem programowy UART.  
Waitms 100  
  
'otwieramy kanał wyjściowy  
Open "comd.1:19200,8,n,1" For Output As #1  
Print #1 , "Transmisja szeregową"  
  
'Teraz otwieramy kanał wejściowy
```

```

Open "comd.0:19200,8,n,1" For Input As #2
'ponieważ nie ma połączenia pomiędzy końcówką wejściową a wyjściową
'nie jest wysyłane echo podczas naciskania klawiszy
Print #1 , "Liczba"
'odczytaj liczbę
Input #2 , B
'i wydrukuj
Print #1 , B

'Teraz odczytujemy znaki aż otrzymamy kod Esc
'Funkcją INKEY() można sprawdzić czy jakiś znak jest dostępny.
'By użyć tej funkcji w połączeniu z programowym UART-em należy podać
numer kanału
Do
  'czytamy
  B = Inkey(#2)
  'jeśli kod > 0 wtedy odebraliśmy znak
  If B > 0 Then
    Print #1 , Chr(b)          'drukujemy go
  End If
Loop Until B = 27

Close #2
Close #1

'Możesz także używać sprzętowego układu UART
'Wtedy jednak dla programowego układu UART należy określić inne
kończówki portów.
'Na przykład wydrukujemy sobie coś
'Print B

'Jeśli nie chcesz używać inwertera poziomów jakim jest np. MAX-232
'możesz dodać parametr ,INVERTED:
'Open "comd.0:300,8,n,1,inverted" For Input As #2
'Wtedy poziomy logiczne są odwracane i nie potrzebna jest ich
dodatkowa inwersja.
'Jednak w tym wypadku długość przewodów transmisyjnych musi być
krótsza.

End

```

OUT

Przeznaczenie:

Wysyła bajt do określonego portu lub do zewnętrznej pamięci.

Składnia:

OUT adres , dana

gdzie:

adres	dowolny adres w przestrzeni adresowej (0-&HFFFF) procesora,
dana	dowolna stała lub zmienna typu Byte.

Opis:

Instrukcja OUT może zapisać dowolną liczbę mieszczącą się w bajcie pod dowolny adres w przestrzeni adresowej kontrolerów AVR.

Jako adres należy używać liczb typu Word. Liczby typu Integer mogą przyjmować wartości ujemne, które owszem są zapisywane jako 16 bitowa liczba. W liczbach Integer wskaźnikiem znaku

jest najbardziej znaczący bit. Dlatego adresy powyżej 32767 należałoby przeliczać do wartości ujemnych.

Uwaga! Aby zapisać bajt w pamięci XRAM należy włączyć opcję **External RAM access** w opcjach kompilatora.

Zobacz także: **INP**

Przykład:

```
Out &H8000 , 16      'wystawienie na szynie danych d0-d7 liczby 16
                        'gdy na szynie adresowej jest wystawiony adres 8000h
End
```

PEEK()

Przeznaczenie:

Odczytuje daną z rejestru.

Składnia:

zmienna = PEEK (adres)

gdzie:

<i>zmienna</i>	dowolna zmienna, do której zwrócona będzie zawartość rejestru,
<i>adres</i>	adres w przestrzeni rejestrów użytkowych (0-31) procesora.

Opis:

Funkcja PEEK działa podobnie jak funkcja **INP**, lecz zakres adresów jest ograniczony do adresów zajmowanych przez rejestry użytkowe R0-R31.

By odczytać bajt spod większego adresu, należy użyć funkcji **INP**.

Zobacz także: **POKE** , **CPEEK** , **CPEEKH** , **INP** , **OUT**

Przykład:

```
Dim A As Byte
a = Peek(0)          'zwraca bajt spod adresu 0 (rejestr R0)
End
```

POKE

Przeznaczenie:

Zapisuje daną do rejestru.

Składnia:

POKE adres , dana

gdzie:

<i>adres</i>	dowolny adres w przestrzeni adresowej rejestrów (0-31) procesora,
<i>dana</i>	dowolna stała lub zmienna typu Byte.

Opis:

Instrukcja jest dokładną odwrotnością funkcji PEEK.

Aby zapisać daną pod większym adresem należy użyć instrukcji OUT.

Zobacz także: PEEK , CPEEK , OUT

Przykład:

```
Poke 1 , 1      'zapisze 1 do rejestru R1

End
```

POPALL

Przeznaczenie:

Przywraca zawartość wszystkich rejestrów odłożonych na stosie.

Składnia:

POPALL

Opis:

Gdy użytkownik tworzy własne procedury w języku assemblera i wstawia je w treść programu języka BASCOM BASIC, nie potrafi określić, które z rejestrów są używane przez program w języku BASCOM BASIC. Zapotrzebowanie na rejestry jest ściśle związane z konkretnymi instrukcjami i funkcjami. Także procedury obsługi przerwań, wykonywane „w tle” mogą używać rejestrów w trakcie ich działania.

Aby zabezpieczyć się przed skutkami zniszczenia ważnych danych z rejestrów, można użyć instrukcji PUSHALL, która umieszcza zawartość rejestrów na stosie. Mogą potem mogą być z niego zdjęte instrukcją POPALL.

Zobacz także: PUSHALL

POWER() (Nowość w wersji 1.11.6.8)

Przeznaczenie:

Dokonuje operacji podniesienia do potęgi liczby typu Single.

Składnia:

zmienna = **POWER**(*liczba* , *wykładnik*)

gdzie:

<i>zmienna</i>	dowolna zmienna typu Single do której wpisany będzie wynik działania funkcji,
<i>liczba</i>	zmienna typu Single w której umieszczono liczbę poddaną operacji potęgowania,
<i>wykładnik</i>	wykładnik potęgi.

Opis:

Funkcja POWER wykonuje operacje tylko dla liczb dodatnich. Gdy używana jest normalna operacja potęgowania: $a \wedge b$, znak liczby jest zachowywany. Dla przykładu arkusz Excel nie potrafi podnieść do potęgi liczby zmiennoprzecinkowej ujemnej, a QBasic pozwala na to.

Funkcja POWER zużywa mniej kodu niż normalna operacja potęgowania \wedge , której argumentem jest liczba zmiennoprzecinkowa.

Zobacz także: EXP , LOG , LOG10

Przykład: Zobacz temat Biblioteka FP_TRIG

POWERDOWN

Przeznaczenie:

Zatrzymuje procesor do czasu jego wyzerowania – Power Down Mode.

Składnia:

POWERDOWN

Opis:

W trybie Power Down, zewnętrzny oscylator zostaje zatrzymany. Użytkownik może przywrócić działanie procesora tylko za pomocą: układu WATCHDOG, zewnętrznego sygnału reset lub za pomocą zewnętrznego sygnału przerwania.

Zobacz także: **IDLE** , **POWERSAVE**

Przykład:

```
Powerdown
```

POWERSAVE

Przeznaczenie:

Wprowadza procesor w tryb obniżonego poboru mocy.

Składnia:

POWERSAVE

Opis:

Tryb Power Save jest dostępny tylko w kontrolerze AT90s8535

Zobacz także: **IDLE** , **POWERDOWN**

Przykład:

```
Powersave
```

PRINT

Przeznaczenie:

Wysyła dane przez sprzętowy lub programowy UART.

Składnia:

PRINT [*zmienna* | *stała*] [; [*zmienna* | *stała*]]

PRINT #*nr_kanału* , [*zmienna* | *stała*] [; [*zmienna* | *stała*]]

gdzie:

zmienna,

dowolna stała lub zmienna,

stała

nr_kanału

numer otwartego wcześniej kanału programowego lub sprzętowego układu UART

Opis:

Można umieszczać więcej zmiennych lub stałych w instrukcji PRINT, rozdzielając je znakiem średnika (;). Gdy średnik zostanie umieszczony na końcu listy zmiennych lub stałych, nie zostanie wysłany znak przejścia do następnej linii.

Instrukcje PRINT można używać gdy projektowany system mikroprocesorowy wyposażony jest w interfejs RS-232. Interfejs ten może być połączony z portem COM komputera osobistego, gdzie za pomocą dowolnego emulatora terminala, lub wbudowanego w środowisko BASCOM AVR - można odbierać znaki z mikrokontrolera.

Zobacz także: **INPUT** , **OPEN** , **CLOSE**

Przykład:

```
'-----  
'                               (c) 2000 MCS Electronics  
'-----  
'   plik: PRINT.BAS  
'   demo: PRINT  
'-----  
  
Dim A As Byte , B1 As Byte , C As Integer  
  
A = 1  
Print "Drukujemy wartość zmiennej A " ; A  
Print                                     'przejdźcie do nowej linii  
Print "Jakiś tam tekst"                 'drukowanie stałej  
  
B1 = 10  
Print Hex(B1)                           'drukowanie w formacie HEX  
C = &HA000  
Print Hex(C)                             'drukowanie w formacie HEX  
Print C                                  'drukowanie w formacie  
decymalnym  
  
C = -32000  
Print C  
Print Hex(C)  
Rem Uwaga! Liczby Integer mają zakres -32767 To 32768  
  
End
```

PRINTBIN

Przeznaczenie:

Przesyła binarnie zawartość dowolnej zmiennej przez sprzętowy lub programowy UART.

Składnia:

```
PRINTBIN zmienna [; zmiennaN]  
PRINTBIN #nr_kanału ; zmienna [; zmiennaN]
```

gdzie:

<i>zmienna</i> ,	dowolna zmienna,
<i>zmiennaN</i>	
<i>nr_kanału</i>	numer otwartego wcześniej kanału programowego lub sprzętowego układu UART

Opis:

Instrukcja PRINTBIN jest jakby ekwiwalentem instrukcji **Print Chr**(*zmienna*); z tym, że można przesyłać zmienne dowolnego typu. Na przykład podanie jako parametru zmiennej typu Long, spowoduje wysłanie 4 bajtów. Można także przesyłać całą zawartość zmiennej tablicowej.

Gdy używany jest drugi format instrukcji, należy podać numer otwartego wcześniej (instrukcją **OPEN**) kanału transmisji urządzenia UART. W innym wypadku kompilator zasygnalizuje błąd.

Zobacz także: **INPUTBIN**

Przykład:

```
Dim a(10) As Byte, c As Byte

For c = 1 To 10
    a(c) = a          'wpiszemy jakieś dane
Next

Printbin a(1)        'i wydrukujemy
```

PSET

Przeznaczenie:

Zapala lub gasi określony piksel na ekranie graficznego wyświetlacza LCD.

Składnia:

PSET *x, y, kolor*

gdzie:

<i>x, y</i>	współrzędne piksela,
<i>kolor</i>	kolor jaki ma mieć podany piksel.

Opis:

Współrzędna X piksela może się zawierać w granicach 0-239. Pozycja Y zaś w granicach 0-63. Kolor piksela może przyjąć wartości 0 – zgaszony i 1 – zapalony.

Instrukcja PSET może być użyteczna przy rysowaniu skomplikowanych wzorów lub do rysowania przebiegów w programie prostego oscyloskopu.

Zobacz także: **LINE** , **CIRCLE** , **SHOWPIC** , **CONFIG GRAPHLCD**

Przykład:

```
'-----
'                                     (c) 2001-2002 MCS Electronics
' Demo graficznego wyświetlacza LCD ze sterownikiem T6963C
'-----

'Sposób podłączenia wyświetlacza LCD:
'końcówka                podłączona do
'1          GND           GND
'2          GND           GND
'3          +5V           +5V
'4          -9V           -9V potencjometr
'5          /WR           PORTC.0
'6          /RD           PORTC.1
'7          /CE           PORTC.2
'8          C/D           PORTC.3
'9          NC            nie podłączone
'10         RESET        PORTC.4
'11-18      D0-D7        PA
'19         FS           PORTC.5
'20         NC            nie podłączone

$crystal = 8000000
'Najpierw definiujemy, że używany będzie graficzny wyświetlacz LCD
'Na razie tylko wyświetlacze 240*64 są obsługiwane
```



```

Config Graphlcd = 240 * 128 , Dataport = Porta , Controlport = Portc ,
Ce = 2 , Cd = 3 , Wr = 0 , Rd = 1 , Reset = 4 , Fs = 5 , Mode = 8
'Dataport określa port do jakiego podłączono szynę danych LCD
'Controlport określa który port jest używany do sterowania LCD
'CE, CD itd. to numery bitów w porcie CONTROLPORT
'Na przykład CE = 2 gdyż jest podłączony do końcówki PORTC.2
'Mode = 8 daje 240 / 8 = 30 kolumn , Mode=6 gda je 240 / 6 = 40 kolumn

'Deklaracje (y nie używane)
Dim X As Byte , Y As Byte

'Kasujemy ekran wyświetlacza (tekst oraz grafikę)
Cls
'Inne opcje to:
' CLS TEXT   kasuje tylko stronę tekstową
' CLS GRAPH  kasuje tylko stronę graficzną

Cursor Off

Wait 1
'Instrukcja Locate działa normalnie jak dla wyświetlaczy tekstowych
' LOCATE LINIA , KOLUMNA  numer linii w zakresie 1-8, kolumny 0 - 30
Locate 1 , 1

'Teraz napiszemy jakiś tekst
Lcd "MCS Electronics"
'i jeszcze jakieś inne w linii 2
Locate 2 , 1 : Lcd "T6963c support"
Locate 16 , 1 : Lcd "A to będzie w linii 16 (najniższej)"

Wait 2

Cls Text

'Użyj nowej instrukcji LINE by narysować pudełko
'LINE(X0 , Y0) - (X1 , Y1), on/off
Line(0 , 0) -(239 , 127) , 255      ' poprzeczne linie
Line(0 , 127) -(239 , 0) , 255
Line(0 , 0) -(240 , 0) , 255      ' górna linia
Line(0 , 127) -(239 , 127) , 255  ' dolna linia
Line(0 , 0) -(0 , 127) , 255      ' lewa linia
Line(239 , 0) -(239 , 127) , 255  ' prawa linia

Wait 2
'Można także rysować linie za pomocą PSET X ,Y , on/off
'parametr on/off to: 0 - skasowanie piksela, inna - postawienie
piksela
For X = 0 To 140
    Pset X , 20 , 255              'narysuj punkt
Next

Wait 2
'A teraz czas najwyższy na obrazek
'SHOWPIC X , Y , etykieta
'Etykieta określa początek danych gdzie zapisany jest obrazek
Showpic 0 , 0 , Plaatje
Wait 2
Cls Text                          'skasuj tekst
End

```

```
'Tutaj znajduje się obrazek
Plaatje:
'Dyrektywa $BGF spowoduje umieszczenie danych obrazka w tym miejscu
$bgf "mcs.bgf"

'Możesz dodać inne obrazki tutaj
```

PULSEIN

Przeznaczenie:

Zwraca ilość jednostek czasowych, które upłynęły pomiędzy dwoma zboczami impulsów.

Składnia:

PULSEIN *zmienna* , PINx , *nr_końcówki* , *zbocze*

gdzie:

<i>zmienna</i>	zmienna do której wpisany będzie czas, wyrażony w jednostkach czasowych,
PINx	nazwa rejestru wejściowego portu, np. PIND,
<i>nr_końcówki</i>	numer testowanej końcówki w podanym porcie PINx
<i>zbocze</i>	określa przejście stanu na końcówce jakie ma być brane pod uwagę. Podanie 0 określa, że chodzi o przejście z 0 do 1; podanie zaś 1 , że o przejście z 1 na 0.

Opis:

Instrukcja PULSEIN oczekuje na pojawienie się określonego zbocza na podanej końcówce. Gdy jako ostatni parametr podane jest **0**, instrukcja oczekuje, aż stan końcówki też będzie 0. Wtedy dopiero zostanie uruchomiona procedura zliczania czasu, zakończona dopiero gdy na końcówce pojawi się stan 1, lub też gdy upłynie maksymalny czas oczekiwania na zmianę stanu.

Gdy czas przekroczy z góry określony limit, zmienna **ERR** przyjmie stan 1. Limit ten jest ustawiony na 65535 jednostek, co przy jednej jednostce wynoszącej 10µs da czas 655.35ms.

Można dodać instrukcję **BITWAIT**, by mieć pewność, że instrukcja PULSEIN odczeka na pojawienie się określonego stanu początkowego. Z drugiej jednak strony, jeśli stan ten się nie pojawi program zatrzyma się w tym miejscu na stałe.

Uwaga! Do zliczania nie jest używany żaden z liczników-czasomierzy. Wewnętrzna zmienna pracująca jako licznik (16 bitowy) jest zwiększana co 10µs. Czas ten jednak zależy od częstotliwości taktującego kwarcu. Można zmodyfikować procedurę z biblioteki by zmienić podstawową jednostkę czasu.

Zobacz także: PULSEOUT

Asembler:

Wywoływana jest procedura `_PULSE_IN` (korzystająca z procedury `_ADJUST_PIN`) z biblioteki `MCS.LIB`. Jako parametry przekazywane są: ZL wskazuje rejestr PINx, R16 zawiera stan końcówki a R24 przechowuje numer końcówki portu. Procedura zwraca w parze rejestrów X odmierzony czas.

Przykład:

```
Dim w As Byte

Pulsein w , PIND , 1 , 0 'detekcja zmiany stanu z 0 na 1
Print w

End
```

PULSEOUT

Przeznaczenie:

Generuje impuls na określonej końcówce portu.

Składnia:

PULSEOUT *port* , *pin* , *okres*

gdzie:

<i>port</i>	nazwa użytego portu mikrokontrolera,
<i>pin</i>	stała lub zmienna określająca numer bitu w porcie,
<i>okres</i>	czas trwania impulsu.

Opis:

Impuls jest generowany poprzez dwukrotną zmianę stanu końcówki portu. Stan początkowy końcówki określa polaryzacja impulsu. Okres jest liczony w jednostkach po 1µs (przy częstotliwości zegara systemowego równej 4MHz).

Uwaga! Używana przez PULSEOUT końcówka portu musi być wcześniej ustawiona jako wyjście.

Przykład:

```
Dim A As Byte

Config Portb = Output      'PORTB jako wyjście
Portb0 = 0                 'wszystkie końcówki na 0
Do
  For A = 0 To 7
    Pulseout Portb , A , 60000  'generuj impuls
    Waitms 250                 'czekaj
  Next
Loop                        'i powtarzaj
```

PUSHALL

Przeznaczenie:

Umieszcza na stosie zawartość wszystkich rejestrów.

Składnia:

PUSHALL

Opis:

Gdy użytkownik tworzy własne procedury w języku assemblera i wstawia je w treść programu języka BASCOM BASIC, nie potrafi określić, które z rejestrów są używane przez program w języku BASCOM BASIC. Zapotrzebowanie na rejestry jest ściśle związane z konkretnymi instrukcjami i funkcjami. Także procedury obsługi przerwań, wykonywane „w tle” mogą używać rejestrów w trakcie ich działania.

Instrukcja PUSHALL umieszcza zawartość rejestrów na stosie, które potem mogą być zdjęte instrukcją POPALL.

Zobacz także: [POPALL](#)

RAD2DEG() *(Nowość w wersji 1.11.6.8)*

Przeznaczenie:

Zamienia radiany na stopnie.

Składnia:

`zmienna = RAD2DEG(stopnie)`

gdzie:

<code>zmienna</code>	zmienna typu Single, do której wpisana będzie wartość w stopniach podanego kąta w radianach,
<code>stopnie</code>	kąt określony w radianach.

Opis:

Wszystkie funkcje trygonometryczne używają radianów. Konwersji pomiędzy stopniami a radianami można dokonać używając funkcji DEG2RAD oraz RAD2DEG.

Zobacz także: [DEG2RAD](#)

Przykład:

```
Dim S As Single

S = 3.14 / 2
S = Rad2deg(s)
Print S
```

RC5SEND (Nowość w wersji 1.11.6.8)

Przeznaczenie:

Wysyła sygnały zdalnego sterowania w standardzie RC5.

Składnia:

`RC5SEND bit_zmienny , adres , rozkaz`

gdzie:

<code>bit_zmienny</code>	podanie 0 zeruje, a podanie 32 ustawia bit zwany. <i>toggle bit</i> ,
<code>adres</code>	adres urządzenia dla którego przeznaczony jest rozkaz,
<code>rozkaz</code>	kod rozkazu.

Opis:

Wiele urządzeń audio-video jest wyposażonych w system zdalnego sterowania pracujący w podczerwieni. Dość duża grupa tego typu urządzeń (szczególnie produkowanych przez europejskie firmy *przyp. tłumacza*) używa transmisji w kodzie RC5.

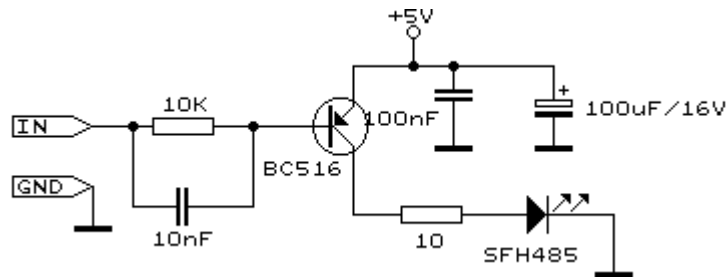
Nadajniki pracujące według standardu RC5 transmitują 14-bitowe słowa danych, kodowane w formacie bi-phase, zwanym także kodem Manchester.

Pierwsze dwa bity słowa są zawsze jedynkami i tworzą razem sygnał startu. Następny bit jest bitem kontrolnym, zmieniającym w kolejnych nadawanych słowach, gdy użytkownik przytrzyma klawisz pilota – umożliwia to powtarzanie komend. Kolejne 5 bitów reprezentuje adres urządzenia, które ma być właściwym odbiornikiem transmisji. Dla przykładu: odbiorniki telewizyjne mają zazwyczaj adres 0, a magnetowidy adres 5. Ostatnie 6 bitów reprezentuje jedną z 64 możliwych komend.

Opis implementacji nadajnika.

W przykładzie wykorzystywany jest kontroler AT90s2313, który używa końcówki PortB.3 jako wyjście OC1A. Zajrzyj do not katalogowych, by określić numer końcówki dla innych układów.

Schemat przykładowego wzmacniacza sygnału podczerwieni pokazano poniżej:



Rysunek 23 Wzmacniacz wyjściowy nadajnika.

Zobacz także: **SONYSEND** , **CONFIG RC5** , **GETRC5**

Przykład:

```

'-----
'
'               RC5SEND.BAS
'               (c) 2002 MCS Electronics
'Kod w oparciu o notę aplikacyjną, którą napisał Ger Langezaal
'+5V <---[A Led K]---[220 Ohm]---> PB.3 dla 90s2313.
'RC5SEND używa licznika TIMER1, nie są używane przerwania
'Rezystor musi być dołączony do końcówki OC1(A), w tym wypadku PB.3
'-----

$regfile = "2313def.dat"
$crystal = 4000000

Dim Togbit As Byte , Command As Byte , Address As Byte

Command = 12                                'kod włącz/wyłącz
Togbit = 0                                  'wyzeruj toggle bit
Address = 0

Do
    Waitms 500
    Rc5send Togbit , Address , Command
Loop

End

```

RC6SEND (Nowość w wersji 1.11.6.9)

Przeznaczenie:

Wysyła sygnały zdalnego sterowania w standardzie RC6.

Składnia:

RC6SEND *bit_zmienny* , *adres* , *rozkaz*

gdzie:

<i>bit_zmienny</i>	podanie 0 zeruje, a podanie 1 ustawia bit zwany. <i>toggle bit</i> ,
<i>adres</i>	adres urządzenia dla którego przeznaczony jest rozkaz,
<i>rozkaz</i>	kod rozkazu.

Opis:

Wiele urządzeń audio-video jest wyposażonych w system zdalnego sterowania pracujący w podczerwieni. Nadajniki pracujące według standardu RC6 transmitują 16-bitowe słowa danych, kodowane w formacie bi-phase, zwanym także kodem Manchester.

Nagłówek składa się z 20bitów w których umieszczone są bity *toggle bit*. Pięć bitów systemowych zawiera adres urządzenia, co pozwala by tylko odpowiednie urządzenia mogły odebrać transmitowany kod. Rozkazy są 8 bajtowe, więc jest możliwe przesłanie do 256 komend dla jednego

urządzenia.

Przeważnie odbiorniki telewizyjne posiadają adres 0, magnetowidy adres 5, odbiorniki SAT adres 8 a odtwarzacze DVD adres 4. Poniżej znajduje się lista (niekompletna) rozkazów i ich kodów:

Tabela 14 Niektóre kody dla pilotów pracujących w kodzie RC6.

Rozkaz	Kod	Rozkaz	Kod
Klawisz 0	0	Balans w prawo	26
Klawisz 1	1	Balans w lewo	27
Klawisz 2-9	2-9	Przeszukiwanie kanałów +	30
Program -	10	Przeszukiwanie kanałów -	31
Włącz/Wyłącz	12	Następny	32
Wyciszanie	13	Poprzedni	33
Pamięć nastaw	14	External 1	56
Pokaż OSD	15	External 2	57
Głośność +	16	Tryb teletekstu	60
Głośność -	17	Stanby	61
Jasność +	18	Pokaż Menu	84
Jasność -	19	Ukryj Menu	85
Nasycenie +	20	Pomoc	129
Nasycenie -	21	Powiększenie -	246
Tony niskie +	22	Powiększenie +	247
Tony niskie -	23		
Tony wysokie +	24		
Tony wysokie -	25		

Uwaga! Informacje opublikowane w sieci Internet na temat RC6 są aktualnie dość skąpe. Używasz ich na własne ryzyko!

Opis implementacji nadajnika.

W przykładzie wykorzystywany jest kontroler AT90s2313, który używa końcówki PB.3 jako wyjście OC1A. Zajrzyj do not katalogowych, by określić numer końcówki dla innych układów.

Schemat przykładowego wzmacniacza sygnału podczerwieni jest taki sam jaki pokazano przy omawianiu instrukcji **RC5SEND**.

Zobacz także: **SONYSEND** , **RC5SEND** , **CONFIG RC5** , **GETRC5**

Przykład:

```
'-----
'
'                      RC5SEND.BAS
'                      (c) 2002 MCS Electronics
'Kod w oparciu o notę aplikacyjną, którą napisał Ger Langezaal
'+5V <---[A Led K]---[220 Ohm]---> PB.3 dla 90s2313.
'RC5SEND używa licznika TIMER1, nie są używane przerwania
'Rezystor musi być dołączony do końcówki OC1(A), w tym wypadku PB.3
'-----

$regfile = "2313def.dat"
$crystal = 4000000

Dim Togbit As Byte , Command As Byte , Address As Byte

Command = 12
Togbit = 0
Address = 0
Do
    'kod włącz/wyłącz
    'wyzeruj toggle bit
```

```

    Waitms 500
    Rc6send Togbit , Address , Command
Loop

End

```

READ

Przeznaczenie:

Odczytuje daną z linii DATA i przypisuje je podanej zmiennej.

Składnia:

READ *zmienna*

gdzie:

zmienna dowolna zmienna.

Opis:

W programie można umieszczać ciągi danych, które mogą być potem odczytywane instrukcją READ. Dane umieszczone w liniach DATA mogą być dowolnego typu (oprócz typu Bit!) i mogą być odczytywane wielokrotnie.

Instrukcja READ współpracuje z specjalnym wskaźnikiem danych i dlatego należy go najpierw ustawić za pomocą instrukcji RESTORE.

Różnice w stosunku do QBasic-a.

Należy zwracać uwagę na zgodność typów odczytywanych a umieszczonych w liniach DATA.

Zobacz także: DATA , RESTORE

Przykład:

```

'-----
'                                READDATA.BAS
'                                Copyright 2000 MCS Electronics
'-----

Dim A As Integer , B1 As Byte , Count As Byte
Dim S As String * 15
Dim L As Long

Restore Dta1                                'ustawiamy wskaźnik
For Count = 1 To 3                          'czytamy trzy dane
    Read B1 : Print Count ; " " ; B1
Next

Restore Dta2                                'ustawiamy wskaźnik
For Count = 1 To 2                          'czytamy dwie dane
    Read A : Print Count ; " " ; A
Next

Restore Dta3
Read S : Print S
Read S : Print S

Restore Dta4
Read L : Print L                            'dane typu Long

End

```

```

Dta1:
  Data &B10 , &HFF , 10

Dta2:
  Data 1000% , -1%

Dta3:
  Data "Witaj" , "Świecie"
'Uwaga! Dane typu Integer (>255 lub <0) muszą być zakończone znakiem %
'Typ danych odczytywanych instrukcją READ i zapisanych w liniach DATA
'musi się zgadzać.

Dta4:
  Data 123456789&
'Uwaga! Dane typu Long muszą być zakończone znakiem &

```

READEEPROM

Przeznaczenie:

Odczytuje zawartość wbudowanej pamięci EEPROM.

Składnia:

READEEPROM *zmienna* , *adres*

gdzie:

<i>zmienna</i>	zmienna, do której wpisana będzie zawartość adresowanej komórki EEPROM,
<i>adres</i>	adres komórki pamięci EEPROM.

Opis:

Procesory serii AVR mają wbudowaną pamięć EEPROM, której można używać z poziomu języka BASCOM BASIC do przechowywania zmiennych lub dowolnych danych.

Na przykład można umieścić dowolną zmienną – oprócz typu Bit – we wbudowanej pamięci EEPROM:

```

Dim V As Eram Byte      'zmienna V będzie umieszczona w EEPROM
Dim B As Byte           'zmienna będzie w pamięci SRAM

B = 10
V = B                   'można normalnie korzystać z takich zmiennych
B = V                   'na przykład odczytać zapisana wcześniej dana

```

Uwaga! Bardzo ważna jest zgodność typów.

Można także używać zmiennych tablicowych:

```
Dim ar(10) As Eram Byte
```

Uwaga! Opierając się na nacie katalogowej firmy Atmel, pierwsza komórka pamięci EEPROM (o adresie 0) może zostać nadpisana podczas operacji zerowania mikroprocesora. Zaleca się nie używać tej komórki pamięci dla ważnych danych, szczególnie przechowywanych po zaniku zasilania.

Instrukcję tą wprowadzono ze względu na kompatybilność z kompilatorem BASCOM-8051.

Jeśli opuszczona będzie etykieta przy kolejnym odczycie, należy użyć nowej wersji instrukcji READEEPROM. Nie działa ona jednak w pętli:

```

Readeeprom B , Label1
Print B

```



```

Do
  Readeeprom B
  Print B
Loop Until B = 5

```

Powyższa konstrukcja nie zadziała, gdyż nie jest zachowywany wskaźnik. Działa jednak ten sposób:

```

ReadEEProm B , Label1      'określ etykietę
ReadEEProm B                'czytaj następny adres w EEPROM
ReadEEProm B                'czytaj następny adres w EEPROM

```

Zobacz także: **WRITEEEPROM**

Przykład1:

```

Dim B As Byte

WriteEeprom B , 0          'zapis do pamięci EEPROM
ReadEeprom B , 0           'odczytanie wartości

```

Przykład2:

```

'-----
'                                     EEPROM2.BAS
'przykład ten pokazuje jak używać nowej wersji instrukcji READEEPROM
'-----
'Najpierw określimy zmienne
Dim B As Byte
Dim Yes As String * 1

'Składnia Readeeprom oraz Writeeprom:
'Readeeprom var, address

'Nowością jest wprowadzenie obsługi odczytywania danych umieszczonych
w liniach DATA
'Ponieważ dane te są umieszczone w osobnym pliku a nie w pamięci kodu,
'należy się przełączyć na pamięć EEPROM i umieścić je na początku
programu
'Taka jest różnica między normalnymi liniami DATA, które muszą być
'umieszczone poza wykonywalną częścią programu!!

'przełączamy się zatem na pamięć EEPROM
$eeprom
'określamy etykiety (adresy)
Label1:
  Data 1 , 2 , 3 , 4 , 5
Label2:
  Data 10 , 20 , 30 , 40 , 50

'przełączamy się na pamięć kodu
$data
'Cały powyższy fragment w istocie nie generuje kodu.
'Tworzy on tylko plik z danymi umieszczonymi w pliku .EEP

'Używamy nowej składni instrukcji
Readeeprom B , Label1
Print B                                'wydrukuj 1
'Następna instrukcja spowoduje odczyt danych spod następnego adresu

```

```

'Lecz pierwsza instrukcja musi określić skąd odczytać dane - adres
startowy
Readeeprom B
Print B 'wydrukuje 2

Readeeprom B , Label2
Print B 'wydrukuje 10
Readeeprom B
Print B 'wydrukuje 20

'Działa to także podczas zapisu danych:
'zatrzymamy się jednak na chwilę
Input "Gotów?" , Yes
B = 100
Writeeprom B , Label1
B = 101
Writeeprom B

'odczytujemy je z powrotem
Readeeprom B , Label2
Print B 'wydrukuje 1
'Następna instrukcja spowoduje odczyt danych spod następnego adresu
'Lecz pierwsza instrukcja musi określić skąd odczytać dane - adres
startowy
Readeeprom B
Print B 'wydrukuje 2

End

```

READMAGCARD

Przeznaczenie:

Odczytuje dane zapisane na karcie magnetycznej.

Składnia:

READMAGCARD *zmienna* , *licznik* , 5 | 7

gdzie:

<i>zmienna</i>	zmienna tablicowa do której wpisane będą odczytane bajty,
<i>licznik</i>	zmienna zawiera ilość rzeczywiście odczytanych bajtów

Opis:

Karta magnetyczna (wyglądająca tak samo jak karta kredytowa czy bankomatowa), posiada szeroki magnetyczny pasek, na którym umieszczono trzy ścieżki danych. Na pierwszej ścieżce zapisywany jest ciąg 7 bitowych danych włącznie z bitem parzystości. Jest ona przewidziana do umieszczenia na niej dowolnych danych alfanumerycznych. Dwie pozostałe ścieżki zawierają dane zakodowane w postaci 5 bitowej.

Instrukcja READMAGCARD działa z kartami zgodnymi z standardem ISO7811-2 z 5 oraz 7 bitowym dekodowaniem. Dane zwracane są w kodzie 5 bitowym:

Tabela 15 Znaki w kodzie 5 bitowym.

Zwracana liczba	Znak wg normy ISO
0	0
1	1
2	2
3	3
4	4

Zwracana liczba	Znak wg normy ISO
5	5
6	6
7	7
8	8
9	9
10	hardware control
11	bajt startu
12	hardware control
13	separator
14	hardware control
15	bajt stopu

Przykład:

```

'-----
'                                     (c) 2000 MCS Electronics
'                                     MAGCARD.BAS
'Przykład ten pokazuje w jaki sposób odczytać dane z karty
magnetycznej
'Testowany był na płytce DT006 SimmStick.
'-----

'[definicja zmiennych]
Dim Ar(100) As Byte , B As Byte , A As Byte

'Czytnik kart posiada 5 wyprowadzeń (kabelków):
'czerwony - +5V
'czarny   - GND
'żółty    - sygnał Card Inserted (CS)
'zielony  - sygnał zegarowy
'niebieski - linia danych

'Jeśli twój czytnik posiada inne wyprowadzenia, to możesz znaleźć
'poszczególne dołączając zasilanie i wkładając kartę do czytnika.
'Przesuwanie karty powinno spowodować, że:
' - linia CS ustawiana będzie w stan niski gdy będziesz wkładał kartę,
' - linia zegarowa będzie generować równomierne impulsy,
' - na linii danych pojawiały będą się przypadkowo jedynki i zera.
'Pamiętaj także o prawidłowym włożeniu karty, mnie się czasem zdarza
'włożyć ją odwrotnie :-).

'Niestety mam znikome pojęcie o kartach magnetycznych, więc proszę o
'nie zasypywanie mnie pytaniami na ten temat.
'W DT006 usuń wszystkie zworki które połączone są z diodami LED.

'[używamy aliasów dla określenia nazw portów]
_mport Alias Pinb           'wszystkie linie podpięte do PINB
_mdata Alias 0               'dane (niebieski) PORTB.0
_mcs Alias 1                 'linia CS (żółty)PORTB.
_mclock Alias 2              'zegar (zielony) PORTB.2

Config Portb = Input         'potrzebujemy tylko linii 0, 1 i 2
Portb = 255                  'ustawiamy je w stan wysoki

Do
  Print "Wprowadź kartę"     'drukujemy tekst zachęty
  Readmagcard Ar(1) , B , 5   'czytamy dane
  Print B ; " odczytanych bajtów"
  For A = 1 To B
    Print Ar(a);              'i drukujemy
  Next

```

Print
Loop

'Podając 7 zamiast 5 w instrukcji Readmagcard, można odczytać 7 bitowe dane

REM

Przeznaczenie:

Instruuje kompilator by potraktował dalszy tekst jako komentarz.

Składnia:

REM lub '

Opis:

W tekście programu można (a nawet powinno się! *przyp. tłumacza*) umieszczać komentarze. Pozwoli to później zrozumieć jakie operacje wykonuje program co uprości później ewentualne modyfikacje.

Tekst umieszczony po słowie REM lub znaku apostrofu do końca linii programu, nie będzie brany pod uwagę przez kompilator. Dlatego też instrukcję komentarza należy umieszczać jako ostatnią w linii.

Można zamiennie używać instrukcji komentarza REM lub ' (apostrofu). Istnieje także komentarz w postaci bloku:

```
'(  początek bloku komentarza  
Print "To nie zostanie skompilowane"  
)  koniec bloku komentarza
```

Uwaga! Początkowy znak apostrofu gwarantuje kompatybilność z językiem QBasic/Visual Basic.

Przykład:

```
Rem TEST.BAS  wersja 1.00  
  
Print a      \ ; " to jest komentarz" : PRINT "Cześć"  
\           ^----- to nie zostanie wykonane! -----^
```

RESET

Przeznaczenie:

Ustawia określony bit w stan 0.

Składnia:

RESET *bit*
RESET WATCHDOG

RESET *zmienna.x*

gdzie:

bit

nazwa bitu; określonego w przestrzeni rejestrów specjalnych czy jako zmienna bitowa

zmienna

dowolna zmienna,

x numer określający bit z zmiennej; 0-7 dla bajtów, 0-15 dla Integer/Word, 0-31 dla Long.

Opis:

Instrukcję RESET WATCHDOG stosuje się do wyzerowania licznika układu Watchdog. (Zostanie ona zastąpiona rozkazem mnemonicznym **Wdr** - *przyp. tłumacza*).

Zobacz także: SET

Przykład:

```
Dim b1 As Bit, b2 As Byte, I As Integer

Reset Portb.3      'ustaw bit 3 w porcie B
Reset b1            'albo zmienną bitową
Reset b2.0          'lub też określony bit w b2
Reset I.15          'albo bit MSB w zmiennej I
```

RESTORE

Przeznaczenie:

Ustawia wewnętrzny wskaźnik danych dla instrukcji READ.

Składnia:

RESTORE *etykieta*

gdzie:

etykieta

etykieta określająca adres danych na które ma być ustawiony wskaźnik.

Opis:

Aby ponownie odczytać dane z linii DATA rozpoczynając od początku lub zmienić wskazanie na inne dane, należy zastosować instrukcję RESTORE. Ustawi ona wewnętrzny wskaźnik na pozycję określoną podaną jako parametr etykiety.

Zobacz także: DATA, READ

Przykład:

```
Dim a As Byte, I As Byte

Restore dta1
For a = 1 To 3
    Read a : Print a
Next

Restore dta2
Read I : Print I
Read I : Print I

End

dta1:
    Data 5, 10, 100

dta2:
    Data -1%, 1000%
'Dane typu Integer (<0 lub >255) muszą być zakończone znakiem %
```

RETURN

Przeznaczenie:

Wykonuje powrót z podprogramu.

Składnia:

RETURN

Opis:

Podprogramy wywoływane przez **GOSUB** muszą kończyć się instrukcją **RETURN**. Nastąpi wtedy powrót do instrukcji następnej po instrukcji **GOSUB** która wykonała skok.

Instrukcja służy także do zakończenia podprogramów obsługi przerwań. Skompilowana będzie wtedy na instrukcję powrotu z przerwania **Reti**. Dalsze informacje przy opisie instrukcji **ON INTERRUPT**

Zobacz także: **GOSUB** , **ON INTERRUPT**

Przykład:

```
Gosub Pr          'skoczmy do podprogramu
Print result      'wydrukujemy rezultat jego działania

End              'koniec programu głównego

Pr:              'podprogram musi rozpoczynać się etykietą
    result = 5 * y 'robimy coś głupiego
    result = result + 100 'a może jeszcze coś dodamy...
Return          'i wracamy
```

RIGHT()

Przeznaczenie:

Zwraca określoną liczbę znaków z tekstu licząc od prawej strony.

Składnia:

zmienna = RIGHT(tekst , il_znaków)

gdzie

<i>zmienna</i>	zmienna tekstowa, do której przepisane będą skopiowane znaki,
<i>tekst</i>	tekst z którego skopiowane będą znaki,
<i>il_znaków</i>	ilość kopiowanych znaków.

Zobacz także: **LEFT** , **MID()**

Przykład:

```
Dim s As XRAM String * 15, z As String * 15

s = "ABCDEFGH"
z = Right(s , 3)
Print z          'wydrukuje EFG

End
```

RND()

Przeznaczenie:

Zwraca pseudolosową liczbę z określonego zakresu.

Składnia:

zmienna = **RND**(*zakres*)

gdzie

<i>zmienna</i>	dowolna zmienna numeryczna, w której znajdzie się wygenerowana liczba pseudolosowa,
<i>zakres</i>	liczba określająca górną granicę generowanych liczb.

Opis:

Funkcja RND zwraca liczbę typu Integer/Word oraz współpracuje z specjalną zmienną `__RSEED` (zajmującą dwa bajty). Każde wywołanie funkcji RND powoduje wygenerowanie nowej pseudolosowej liczby.

Uwaga! Liczby są generowane przez odpowiedni kod programu. Każdy restart systemu, powoduje, że sekwencja tych liczb będzie taka sama.

Można zmienić podstawę generatora definiując jawnie w programie zmienną `__RSEED` i przypisując jej nową wartość:

```
Dim __rseed As Word
__rseed = 10234
```

```
Dim I As word
I = Rnd(10)
```

Jeśli aplikacja wykorzystuje jeden ze sprzętowych liczników-czasomierzy, można przypisywać zawartość jego licznika do zmiennej `__RSEED`. Poprawi to sposób generowania liczb pseudolosowych.

Przykład:

```
Dim I As Integer

Do
  I = Rnd(100)           'zwraca liczbę pseudolosową z zakresu 0-99
  Print I
  Wait 1
Loop

End
```

ROTATE

Przeznaczenie:

Przesuwa wszystkie bity w wybranym kierunku.

Składnia:

ROTATE *zmienna* , **LEFT** | **RIGHT** [, *liczba_przesunięć*]

gdzie

<i>zmienna</i>	dowolna zmienna stałoprzecinkowa, która podlega przesuwaniu,
<i>liczba_przesunięć</i>	liczba określająca ilość kroków przesunięć.

Opis:

Instrukcja ROTATE przesuwca wszystkie bity zmiennej w prawo lub lewo. Żaden z bitów nie wychodzi poza zmienną, gdyż przesuwane są one w pętli. Oznacza to, że wychodzący bit z jednej strony zostaje automatycznie wpisany z drugiej. Tak więc, przesunięcie bajtu zawierającego liczbę 1 osiem razy nie zmieni zawartości bajtu – będzie on dalej zawierał 1.

Gdy potrzebna jest funkcja, która usunie bit(y) z prawej lub lewej strony, należy użyć instrukcji SHIFT.

Zobacz także: **SHIFT** , **SHIFTIN** , **SHIFTOUT**

Przykład:

```
Dim a As Byte

a = 128
Rotate a , Left , 2

Print a          'wydrukuje 2
```

ROUND() *(Nowość w wersji 1.11.6.8)***Przeznaczenie:**

Zwraca liczbę zaokrągloną do najbliższej wartości całkowitej.

Składnia:

zmienna = **ROUND**(*wartość*)

gdzie:

<i>zmienna</i>	dowolna zmienna typu Single, do której wpisana będzie wynik działania funkcji,
<i>wartość</i>	liczba którą całkowita część powinna być zwrócona.

Opis:

Funkcja dokonuje zaokrąglenia podanej wartości do najbliższej wartości całkowitej. Jeśli część ułamkowa będzie mniejsza niż 0.5 zwracana będzie tylko część całkowita. Jeśli natomiast będzie większa – zwracana będzie część całkowita powiększona o jeden.

Zobacz także: **INT** , **FIX** , **SGN**

Przykład:

```
'-----
'                                     ROUND_FIX_INT.BAS
'-----
Dim S As Single , Z As Single

For S = -10 To 10 Step 0.5
    Print S ; Spc(3) ; Round(S) ; Spc(3) ; Fix(S) ; Spc(3) ; Int(S)
Next

End
```

RTRIM()**Przeznaczenie:**

Obcina spacje znajdujące się z prawej strony.

Składnia:

`zmienna = RTRIM(tekst)`

gdzie:

<code>zmienna</code>	zmienna tekstowa, do której zapisany będzie rezultat działania funkcji,
<code>tekst</code>	tekst z którego usunięte mają być końcowe spacje.

Zobacz także: [LTRIM](#) , [TRIM](#)

Przykład:

```
Dim S As String * 6

S = "  AB  "
Print Ltrim(s)
Print Rtrim(s)
Print Trim(s)

End
```

SECELAPSED() *(Nowość w wersji 1.11.7.3)*

Przeznaczenie:

Zwraca liczbę sekund jaka upłynęła od poprzednio zapamiętanej liczby sekund od początku dnia.

Składnia:

`rezultat = SECELAPSED(TimeStamp)`

gdzie:

<code>rezultat</code>	zmienna typu Long gdzie funkcja zwróci wynik,
<code>TimeStamp</code>	zmienna typu Long która przechowuje uprzednio zapamiętaną liczbę sekund jaka upłynęła od początku dnia, (funkcja SECOFDAY)

Opis:

Funkcja ta korzysta ze zmiennych programowego zegara czasu rzeczywistego (patrz [CONFIG CLOCK](#)): `_sec`, `_min` i `_hour`. Na podstawie tych danych oblicza różnicę pomiędzy czasem aktualnym a uprzednio zapamiętanym.

Uwaga! Funkcja zwraca poprawne wartości tylko w obrębie 24 godzin.

Zwracana wartość mieści się w granicach 0 do 86399.

Zobacz także: [SYSSECELAPSED](#) , [SECOFDAY](#) , [Biblioteka DATETIME](#)

Przykład:

```
Enable Interrupts
Config Clock = Soft

Dim lTimeStamp As Long
Dim lSecondsElapsed As Long

lTimeStamp = SecOfDay()
Print "Teraz jest " ; lTimeStamp ; " sekund po północy"
```

```
' wykonaj cokolwiek
' a kilka chwil później

lSecondsElapsed = SecElapsed(lTimeStamp)
Print "Teraz jest " ; lSecondsElapsed ; " sekund później"
```

SECOFDAY() (Nowość w wersji 1.11.7.3)

Przeznaczenie:

Zwraca liczbę sekund jaka upłynęła od początku dnia.

Składnia:

```
rezultat = SECOFDAY()
rezultat = SECOFDAY( bSMG )
rezultat = SECOFDAY( strCzas )
rezultat = SECOFDAY( lSysSec )
```

gdzie:

<i>rezultat</i>	zmienna typu Long gdzie funkcja zwróci wynik,
<i>strCzas</i>	ciąg znaków z zapisaną godziną w formacie „gg:mm:ss”,
<i>lSysSec</i>	zmienna typu Long która przechowuje liczbę sekund jaka upłynęła od początku wieku, (SysSec)
<i>bSMG</i>	nazwa zmiennej typu Byte, w której przechowywany jest bieżąca liczba sekund. Zaraz po tym bajcie powinny znajdować się jeszcze dwa bajty zawierające kolejno.

Opis:

Funkcja ta może być wywołana z czterema wariantami parametrów:

1. Bez parametrów. Wtedy funkcja zwraca liczę sekund na podstawie zmiennych programowego zegara (*_sec*, *_min*, *_hour*).
2. Ze zdefiniowanym przez użytkownika ciągiem 3 bajtów. Muszą one być zorganizowane podobnie jak zmienne programowego zegara. W pierwszym bajcie należy umieścić liczbę sekund, w następnym liczbę minut i ostatnim liczbę godzin.
3. Z ciągiem znaków, w którym data przedstawiona jest w formacie „ss:mm:gg”.
4. Ze zmienną typu Long przechowującą liczbę sekund jaka upłynęła od początku wieku.

Funkcja zwraca liczbę od 0 do 86399. Zero oznacza północ, a 86399 godzinę 23:59:59.

Uwaga! Funkcja nie sprawdza poprawności podawanych jako parametr danych.

Zobacz także: [SYSSEC](#) , [Biblioteka DATETIME](#)

Przykład:

```
Enable Interrupts
Config Clock = Soft

Dim strTime As String * 8
Dim bSec As Byte , bMin As Byte , bHour As Byte
Dim lSecOfDay As Long
Dim lSysSec As Long

' Przykład 1 z wewnętrznym zegarem RTC
_Sec = 12 : _Min = 30 : _Hour = 18          ' ustaw zegar - dla
przykładu i testów
```

```

lSecOfDay = SecOfDay ()
Print "Liczba sekund dla " ; Time$ ; " równa się " ; lSecOfDay
' Liczba sekund dla 18:30:12 równa się 66612

' Przykład 2 z zdefiniowanym ciągiem 3 bajtów (Sekundy / Minuty /
Godziny )
bSec = 20 : bMin = 1 : bHour = 7
lSecOfDay = SecOfDay(bSec)
Print "Liczba sekund dla Sekundy="; bSec ; " Minuty="; bMin ; "
Godziny=" ; bHour ; " równa się " ; lSecOfDay
' Liczba sekund dla Sekundy= 20 Minuty=1 Godziny=7 równa się 25280

' Przykład 3 z ciągiem zawierającym czas
strTime = "04:58:37"
lSecOfDay = SecOfDay(strTime)
Print "Dla godziny " ; strTime ; " zwraca " ; lSecOfDay
' Dla godziny 04:58:37 zwraca 17917

' Przykład 4 z liczbą sekund od początku wieku
lSysSec = 1234456789
lSecOfDay = SecOfDay(lSysSec)
Print "Dla liczby " ; lSysSec ; " sekund zwraca " ; lSecOfDay
' Dla liczby 1234456789 sekund zwraca 59989

End

```

SELECT CASE...CASE...END SELECT

Przeznaczenie:

Specjalna instrukcja wykonująca odpowiedni ciąg instrukcji na podstawie wartości zmiennej.

Składnia:

```

SELECT CASE zmienna

    CASE [ IS < | IS > ] test1 : instrukcje
    [ CASE [ IS < | IS > ] test2 : instrukcje ]
    [ CASE wart_pocz TO wart_końcowa : instrukcje ]

    [ CASE ELSE : instrukcje ]
END SELECT

```

gdzie:

<i>zmienna</i>	zmienna numeryczna, której wartość będzie przedmiotem testowania,
<i>test1</i> , <i>test2</i>	wartości jakie powinna zawierać zmienna by wykonać podane instrukcje, przy podaniu IS należy obowiązkowo podać > lub < przed liczbą,
<i>instrukcje</i>	dowolny ciąg instrukcji.
<i>wart_pocz</i>	dolna liczba zakresu wartości <i>zmiennej</i>
<i>wart_końcowa</i>	górną liczbą zakresu wartości <i>zmiennej</i>

Opis:

Działanie instrukcji strukturalnej SELECT..CASE jest następujące: Jeśli wartość zmiennej podanej po **SELECT CASE** odpowiada którejkolwiek wartości podanej po klauzurze **CASE**, wtedy wykonywane są wszystkie instrukcje związane z tą właśnie klauzulą CASE (podane po dwukropku). Po wykonaniu tych instrukcji program wychodzi z konstrukcji SELECT..CASE, to znaczy do dalszej części programu umieszczonej po słowie **END SELECT**.

Jeśli testowana wartość ma zawierać się w przedziale od do, to zamiast tworzyć kilka klauzul CASE można stworzyć jedną. W tym celu należy użyć słowa **TO**, przed którym podajemy najmniejszą liczbę z zakresu, a po słowie **TO** największą liczbę z zakresu. Należy przy tym uważać, gdyż dla liczb ujemnych konstrukcja CASE -1 TO -5 nie jest poprawna. Prawidłowa konstrukcja to: CASE -5 TO -1.

Można także użyć testu w postaci relacji. Wtedy podajemy słowo **IS** po którym należy umieścić znak większości (>) lub mniejszości (<) oraz liczbę do której relacja ma się odnosić.

Jeśli na końcu umieścimy **CASE ELSE** to podane po tej klauzurze instrukcje zostaną wykonane jeśli żadna z wartości podanej po słowie CASE nie pasuje do testowanej zmiennej. Najlepiej jest umieszczać CASE ELSE w każdej instrukcji SELECT..CASE, można w ten sposób zabezpieczyć program przed wartościami których się nie spodziewaliśmy.

Gdy nie umieścimy CASE ELSE, a żadna z wartości zmiennej nie pasuje do tych określonych po klauzulach CASE, program zachowa się tak jakby instrukcji SELECT..CASE nie było.

Uwaga! Jeśli wartość zmiennej odpowiada dwóm klauzulom CASE wtedy wykonana zostanie tylko pierwsza z nich.

Przykład:

```
Dim X As Integer

Do
  Input "Podaj wartosc X ? " , X
  Select Case X
    Case 1 To 3 : Print "1 , 2 lub 3 są w porządku"
    Case 4 : Print "O! Wpisałeś 4"
    Case Is > 10 : Print "X jest większe niż 10"
    Case Else : Print "Taka liczba nie jest poprawna!"
  End Select
Loop

End
```

SET

Przeznaczenie:

Ustawia określony bit w stan 1.

Składnia:

SET *bit*

SET *zmienna.x*

gdzie:

<i>bit</i>	nazwa bitu; określonego w przestrzeni rejestrów specjalnych czy jako zmienna bitowa
<i>zmienna</i>	dowolna zmienna,
<i>x</i>	numer określający bit z zmiennej; 0-7 dla bajtów, 0-15 dla Integer/Word, 0-31 dla Long.

Zobacz także: **RESET**

Przykład:

```
Dim b1 As Bit, b2 As Byte, I As Integer

Set Portb.3      'ustaw bit 3 w porcie B
Set b1           'albo zmienną bitową
```

```
Set b2.0           'lub też określony bit w b2
Set I.15           'albo bit MSB w zmiennej I
```

SETFONT *(Nowość w wersji 1.11.6.9)*

Przeznaczenie:

Ustawia czcionki używane przez graficzny wyświetlacz LCD oparty na kontrolerze SEDxxxx.

Składnia:

SETFONT *nazwa*

gdzie:

nazwa

nazwa czcionki używanej przez instrukcję LCDAT.

Opis:

Ponieważ wyświetlacze z kontrolerem SED nie posiadają własnego generatora znaków (czcionek), należy samodzielnie zdefiniować te czcionki. Czcionki można utworzyć i modyfikować używając dodatku (plug-in) [FontEditor](#).

Instrukcja SETFONT ustawia specjalny wskaźnik do miejsca w pamięci gdzie umieszczono dane czcionek. Nazwa to ta sama nazwa jaka została użyta przy definiowaniu czcionek.

Plik z czcionkami należy dołączyć do programu korzystając z dyrektywy **\$INCLUDE**:

```
$include "font8x8.font"
```

Aby wykorzystać wyświetlacz z kontrolerem SED i instrukcje z nim związane, należy skorzystać z biblioteki GLCDSER:

```
$lib "glcdsed.lib"
```

Zobacz także: **CONFIG GRAPHLCD** , **LCDAT** , **GLCDCMD** , **GLCDDATA**

SERIN *(Nowość w wersji 1.11.6.9)*

Przeznaczenie:

Odbiera dane transmitowane przez dynamiczny układ programowego urządzenia UART.

Składnia:

SERIN *zmienna* , *il_bajtów* , *port* , *końcówka* , *szybkość* , *parzystość* , *bity_danych* , *bity_stopu*

gdzie:

<i>zmienna</i>	zmienna do której trafiać będą dane,
<i>il_bajtów</i>	liczba odbieranych bajtów; zmienne tekstowe będą wypełniane znakami, aż do odebrania kodu CR (13).
<i>port</i>	jednoznakowa symboliczna nazwa użytego portu; dla portu PORTA literą musi być A,
<i>końcówka</i>	numer końcówki w podanym porcie; może się zawierać w granicach 0 – 7 ,
<i>szybkość</i>	szybkość transmisji w bodach; na przykład: 19200,
<i>parzystość</i>	liczba określająca sposób obliczania bitu parzystości: 0 = NONE (brak), 1 = EVEN (parzystość), 2 = ODD (nieparzystość)
<i>bity_danych</i>	liczba bitów danych; można podać 7 lub 8 ,
<i>bity_stopu</i>	liczba bitów stopu; można podać od 1 do 2 .

Opis:

Używanie instrukcji **OPEN** i **CLOSE** w połączeniu z programowym urządzeniem UART, nie

pozwała na użycie tych samych końcówek portu jako wejścia i wyjścia, gdyż format tych instrukcji wymaga jasnego określenia trybu pracy końcówki. Także podczas działania programu nie jest możliwa zmiana trybu pracy otwartego kanału.

Instrukcje SERIN oraz **SEROUT** korzystają z programowego urządzenia UART pozwalając na dynamiczną zmianę pracy podanych końcówek, które mogą być zarówno wyjściem jak i wejściem, choć nie w tym samym czasie. Można zatem przysyłać dane instrukcją SEROUT i odebrać dane zwrotne instrukcją SERIN.

Ponieważ procedury obsługujące instrukcje SERIN oraz SEROUT mogą używać dowolnej z dostępnych końcówek portu oraz różnych ustawień transmisji, kod wynikowy tych procedur musi być obszerniejszy.

Uwaga! Instrukcje te nie wyposażono w mechanizm kontroli, czy podana zmienna będzie w stanie pomieścić odbierane dane.

Użycie instrukcji SERIN powoduje automatyczne utworzenie specjalnej zmiennej `__SER_BAUD`. Jest to zmienna typu Long. Jest szczególnie ważne, by za pomocą dyrektywy `$CRYSTAL` określić rzeczywistą częstotliwość użytego rezonatora kwarcowego, gdyż czasy opóźnień są obliczane na podstawie tej wartości.

Uwaga! Zmienna `__SER_BAUD` nie przechowuje liczby określającej szybkość pracy łącza. Zmienna ta zawiera czas opóźnienia pomiędzy kolejnymi transmitowanymi bitami.

Ponieważ programowy UART jest dynamiczny, jest możliwa zmiana każdego z parametrów podczas działania programu. Jest na przykład możliwa zmiana szybkości transmisji z wykorzystaniem zmiennej. Zawartość tej zmiennej może być kontrolowana z zewnątrz, przez odczytanie – jak w to podano na końcu przykładu – stanu końcówek portów.

Jest też ważne aby podczas transmisji nie zakłócać jej przebiegu. Dowolne przerwanie, które pojawi się podczas działania instrukcji SERIN, spowoduje trudne do oszacowania opóźnienia w transmisji, co może doprowadzić do błędów. Dlatego lepiej wyłączyć przerwania podczas działania instrukcji SERIN.

Zobacz także: **SEROUT**

Asembler:

Instrukcja wykorzystuje procedurę `_serin` umieszczoną w bibliotece `mcs.lib`. Podczas obliczania szybkości transmisji jest wykorzystywana procedura `_calc_baud`.

Przykład:

```
'-----
'                               serin_out.bas
'                               (c) 2002 MCS Electronics
'                               Demonstracja dynamicznego programowego UART-a
'-----
'Wskazówka : zobacz także OPEN oraz CLOSE

'określ częstotliwość rezonatora kwarcowego
$crystal = 4000000

'określamy model mikrokontrolera
$regfile = "2313def.dat"

'parę używanych zmiennych
Dim S As String * 10
Dim Mybaud As Long
'Jeśli szybkość pracy łącza jest przekazywana przez zmienną
'należy zadbać, by była ona typu Long
```


zwrotne instrukcją SERIN.

Ponieważ procedury obsługujące instrukcje SERIN oraz SEROUT mogą używać dowolnej z dostępnych końcówek portu oraz różnych ustawień transmisji, kod wynikowy tych procedur musi być obszerniejszy.

Uwaga! Instrukcje te nie wyposażono w mechanizm kontroli, czy podana zmienna będzie w stanie pomieścić odbierane dane.

Użycie instrukcji SEROUT powoduje automatyczne utworzenie specjalnej zmiennej `__SER_BAUD`. Jest to zmienna typu Long. Jest szczególnie ważne, by za pomocą dyrektywy `$CRYSTAL` określić rzeczywistą częstotliwość użytego rezonatora kwarcowego, gdyż czasy opóźnień są obliczane na podstawie tej wartości.

Uwaga! Zmienna `__SER_BAUD` nie przechowuje liczby określającej szybkość pracy łącza. Zmienna ta zawiera czas opóźnienia pomiędzy kolejnymi transmitowanymi bitami.

Ponieważ programowy UART jest dynamiczny, jest możliwa zmiana każdego z parametrów podczas działania programu. Jest na przykład możliwa zmiana szybkości transmisji z wykorzystaniem zmiennej. Zawartość tej zmiennej może być kontrolowana z zewnątrz, przez odczytanie – jak w to podano na końcu przykładu – stanu końcówek portów.

Jest też ważne aby podczas transmisji nie zakłócać jej przebiegu. Dowolne przerwanie, które pojawi się podczas działania instrukcji SEROUT, spowoduje trudne do oszacowania opóźnienia w transmisji, co może doprowadzić do błędów. Dlatego lepiej wyłączyć przerwania podczas działania instrukcji SEROUT.

Zobacz także: [SERIN](#)

Asembler:

Instrukcja wykorzystuje procedurę `_serout` umieszczoną w bibliotece `mcs.lib`. Podczas obliczania szybkości transmisji jest wykorzystywana procedura `_calc_baud`.

Przykład:

```
'-----
'                                     serin_out.bas
'                               (c) 2002 MCS Electronics
'                   Demonstracja dynamicznego programowego UART-a
'-----
'Wskazówka : zobacz także OPEN oraz CLOSE

'określ częstotliwość rezonatora kwarcowego
$crystal = 4000000

'określamy model mikrokontrolera
$regfile = "2313def.dat"

'parę używanych zmiennych
Dim S As String * 10
Dim Mybaud As Long
'Jeśli szybkość pracy łącza jest przekazywana przez zmienną
'należy zadbać, by była ona typu Long

Mybaud = 19200
Do
    'najpierw coś odbierzemy
    Serin S , 0 , D , 0 , Mybaud , 0 , 8 , 1
    'now send it
    Serout S , 0 , D , 1 , Mybaud , 0 , 8 , 1
    '                                     ^ 1 bit stopu
```



```

'                                     ^----- 8 bitów danych
'                                     ^----- parzystość (0=N, 1=E, 2=0)
'                                     ^----- szybkość transmisji
'                                     ^----- numer końcówki
'                                     ^----- port (używamy PORTA.0 oraz
PORTA.1)
'                                     ^----- dla zmiennych tekstowych
podaj 0
'                                     ^----- właściwa zmienna

Wait 1
Loop

End

'ponieważ szybkość łącza jest w tym przykładzie przekazywana przez
zmienną
'można umożliwić użytkownikowi określenie tej częstotliwości z
zewnątrz
'na przykład kontrolując stan DIP-switch-a

```

SGN()

Przeznaczenie:

Zwraca znak podanej liczby.

Składnia:

```
zmienna = SGN( liczba )
```

gdzie:

zmienna

dowolna zmienna, do której zwrócony będzie wynik działania funkcji,

liczba

liczba lub zmienna której znak należy określić.

Opis:

Dla liczb mniejszych niż zero funkcja zwraca liczbę -1 , a dla liczb większych niż zero funkcja zwraca liczbę 1 . Jeśli podana liczba jest zerem funkcja także zwraca zero.

Zobacz także: **INT** , **FIX** , **ROUND**

Przykład:

```
Dim S As Single , x As Single , y As Single

x = 2.3
S = Sgn(x)
Print S      '1

x = -2.3
S = Sgn(x)
Print S      '-1
```

SHIFT

Przeznaczenie:

Przesuwa wszystkie bity zmiennej o jedną (lub więcej) pozycję w prawo lub lewo.

Składnia:

SHIFT *zmienna*, **LEFT** | **RIGHT** [, *il_przesunięć*]

gdzie:

<i>zmienna</i>	dowolna zmienna typu Byte, Integer, Word czy Long, której zawartość będzie przesuwana,
<i>il_przesunięć</i>	opcjonalnie liczba przesunięć.

Opis:

Instrukcja SHIFT przesuwa wszystkie bity składające się na podaną zmienną w prawo lub lewo.

Gdy podany jest parametr **LEFT** bit MSB zostaje usunięty, a reszta bitów jest przesuwana w lewo. Na pozycji LSB zostaje umieszczone zero. Przesuwanie wartości zmiennej w lewo, jest równoznaczne z pomnożeniem jej zawartości przez dwa.

Gdy podany jest parametr **RIGHT** bit LSB zostaje usunięty, a reszta bitów jest przesuwana w prawo. Na pozycji MSB zostaje umieszczone zero. Przesuwanie wartości zmiennej w prawo, jest równoznaczne z podzieleniem jej zawartości przez dwa.

Dzielenie lub mnożenie przez dwa wykonywane przez SHIFT jest o wiele szybsze niż ta sama operacja wykonywana przez operatory matematyczne.

Zobacz także: **ROTATE** , **SHIFTIN** , **SHIFTOUT**

Przykład:

```
Dim a As Byte

a = 128
Shift a, Left , 2
Print a          'wydrukuje 0
```

SHIFTCURSOR

Przeznaczenie:

Przesuwa kursor wyświetlacza LCD o jedną pozycję w prawo lub lewo.

Składnia:

SHIFTCURSOR LEFT | RIGHT

Zobacz także: **SHIFTLCD**

Przykład:

```
Cursor On
Lcd "Cześć"

Wait 1
Shiftcursor Left

End
```

SHIFTIN

Przeznaczenie:

Wsusza ciąg bitów do zmiennej.

Składnia:

SHIFTIN *pin_danych* , *pin_zegarowy* , *zmienna* , *opcje* [, *il_bitów* , *opóźnienie*]

gdzie:

pin_danych nazwa końcówki portu będącą linią wejściową strumienia

<i>pin_zegarowy</i>	bitów,
<i>zmienna</i>	nazwa końcówki portu będąca linią zegarową,
<i>opcje</i>	zmienna do której wsuwane będą bity,
<i>il_bitów</i>	opcje,
<i>opóźnienie</i>	ilość wsuwanych bitów,
	opóźnienie w mikrosekundach między kolejnymi bitami.

Opis:

Instrukcja ta może być używana, do szybkiej transmisji szeregowej pomiędzy dwoma procesorami.

Pin_danych powinien być połączony z końcówką wyjściową drugiego procesora, który jest nadajnikiem strumienia bitów. *Pin_zegarowy* w trybie MASTER generuje sygnał zegarowy synchronizujący transmisję. W trybie SLAVE jest on wejściem sygnału zegarowego pochodzącego z drugiego procesora.

Podana jako parametr zmienna może być dowolną – oprócz zmiennych bitowych! - zmienną języka BASCOM BASIC. Bity odczytywane trafią właśnie do tej zmiennej.

Można podać ilość wsuwanych bitów – maksymalnie 255. Jeśli ten parametr nie występuje lub jest określony jako **NULL**, liczba bitów jest automatycznie określana na podstawie ilości bitów składających się na zmienną. Dla typu Byte będzie to 8 bitów a dla typu Word – 16.

Znaczenie liczby podanej jako parametr *opcje*, jest następująca:

- 0 najpierw bit MSB jest wpisywany przy wystawieniu niskiego poziomu logicznego na końcówce zegarowej
- 1 najpierw bit MSB jest wpisywany przy wystawieniu wysokiego poziomu logicznego na końcówce zegarowej
- 2 najpierw bit LSB jest wpisywany przy wystawieniu niskiego poziomu logicznego na końcówce zegarowej
- 3 najpierw bit LSB jest wpisywany przy wystawieniu wysokiego poziomu logicznego na końcówce zegarowej

Gdy liczba określająca opcję zostanie powiększona o 4, wtedy sygnał zegarowy nie będzie generowany i lina zegarowa będzie pełnić rolę wejścia zewnętrznego sygnału zegarowego (tryb SLAVE).:

- 4 najpierw bit MSB jest wpisywany przy niskim poziomie logicznym na końcówce zegarowej
- 5 najpierw bit MSB jest wpisywany przy wysokim poziomie logicznym na końcówce zegarowej
- 6 najpierw bit LSB jest wpisywany przy niskim poziomie logicznym na końcówce zegarowej
- 7 najpierw bit LSB jest wpisywany przy wysokim poziomie logicznym na końcówce zegarowej

Jako opóźnienie normalnie używane są dwie instrukcje maszynowe NOP. Gdy częstotliwość zegara jest zbyt duża, można podać czas opóźnienia w mikrosekundach.

Zobacz także: **SHIFTOUT** , **SHIFT**

Przykład:

```
Dim a As Byte

Shiftin Pinb.0 , Portb.1 , A , 4 , 4 , 10 'wsuwa 4 bity oraz używa
                                          'zewnętrznego sygnału
zegarowego
Shift A , Right , 4                      'przesuwamy
Shiftin Pinb.0 , Portb.1 , A              'odczytuje 8 bitów
```

SHIFTOUT

Przeznaczenie:

Wysuwa ciąg bitów pochodzący z określonej zmiennej.

Składnia:

SHIFTOUT *pin_danych* , *pin_zegarowy* , *zmienna* , *opcje* [, *il_bitów* , *opóźnienie*]

gdzie:

<i>pin_danych</i>	nazwa końcówki portu będącą linią wyjściową strumienia bitów,
<i>pin_zegarowy</i>	nazwa końcówki portu będącą linią zegarową,
<i>zmienna</i>	zmienna z której wysuwane będą bity,
<i>opcje</i>	opcje,
<i>il_bitów</i>	ilość wysuwanych bitów,
<i>opóźnienie</i>	opóźnienie w mikrosekundach między kolejnymi bitami.

Opis:

Instrukcja ta podobnie jak SHIFTIN, może być używana do szybkiej transmisji szeregowej pomiędzy dwoma procesorami.

Pin_danych powinien być połączony z końcówką wejściową drugiego procesora, który jest odbiornikiem strumienia bitów. *Pin_zegarowy* w trybie MASTER generuje sygnał zegarowy synchronizujący transmisję. W trybie SLAVE jest on wejściem sygnału zegarowego pochodzącego z drugiego procesora.

Podana jako parametr *zmienna* może być dowolną – oprócz zmiennych bitowych! - zmienną języka BASCOM BASIC. Bity nadawane będą wysuwane właśnie z tej zmiennej.

Można podać ilość wysyłanych bitów, maksymalnie 255. Jeśli ten parametr nie występuje lub jest określony jako **NULL**, liczba bitów jest automatycznie określana na podstawie ilości bitów składających się na zmienną. Dla typu Byte będzie to 8 bitów a dla typu Word – 16.

Znaczenie liczby podanej jako parametr *opcje*, jest następująca:

- | | |
|---|--|
| 0 | najpierw bit MSB jest podawany przy niskim poziomie logicznym na końcówce zegarowej |
| 1 | najpierw bit MSB jest podawany przy wysokim poziomie logicznym na końcówce zegarowej |
| 2 | najpierw bit LSB jest podawany przy niskim poziomie logicznym na końcówce zegarowej |
| 3 | najpierw bit LSB jest podawany przy wysokim poziomie logicznym na końcówce zegarowej |

Gdy liczba określająca opcję zostanie powiększona o 4, wtedy sygnał zegarowy nie będzie generowany i lina zegarowa będzie pełnić rolę wejścia zewnętrznego sygnału zegarowego (tryb SLAVE).

Jako opóźnienie normalnie używane są dwie instrukcje maszynowe NOP. Gdy częstotliwość zegara jest zbyt duża, można podać czas opóźnienia w mikrosekundach.

Zobacz także: **SHIFTIN** , **SHIFT**

Przykład:

Dim a **As** Byte

```
Shiftout Portb.0, Portb.1, A, 3, 4, 10 'nadanie 4 bitów
Shiftout Pinb.0, Portb.1 , A, 3         'nadanie 8 bitów
```

SHIFTLCD

Przeznaczenie:

Przesuwa zawartość wyświetlacza LCD o jedną pozycję w prawo lub w lewo.

Składnia:

SHIFTLCD LEFT | RIGHT

Zobacz także: **SHIFTCURSOR**

Przykład:

```
Lcd "Bardzo dłuuuuuuuuuuuugi tekst"
Shiftlcd Left
Wait 1
Shiftlcd Right

End
```

SHOWPIC

Przeznaczenie:

Wyświetla obrazek zapisany w formacie BGF na ekranie graficznego wyświetlacza LCD.

Składnia:

SHOWPIC x , y , etykieta

gdzie:

x , y
etykieta

punkt w którym ma się znaleźć lewy górny róg obrazka,
adres określony etykietą, pod jakim znajdują się dane
obrazka.

Opis:

Instrukcja SHOWPIC potrafi wyświetlić bitmapy monochromatyczne poddane konwersji do pliku .BGF narzędziem znajdującym się w menu [Tools | Graphic Converter](#).

Parametry x oraz y określają gdzie obrazek ma być umieszczony na ekranie. Muszą to być wartości podzielne przez 8, albo równe 0. Wysokość obrazka jak i jego szerokość także muszą być wielokrotnością liczby 8.

Podana etykieta określa gdzie w pamięci kodu znajdują się dane obrazka. Nazwę pliku obrazka należy umieścić w dyrektywie **\$BGF** pod wspomnianą etykietą. Najlepiej dane umieścić na końcu programu po słowie **END**.

Można umieścić wiele obrazków, przy czym dla każdego z nich należy określić osobną etykietę.

Uwaga! Dane obrazka w pliku BGF są spakowane metodą RLE (*Run Lenght Encoded* – opartą na eliminacji powtórzeń bajtów – *przyp. tłumacza*) by zmniejszyć rozmiar zajmowanych danych.

Zobacz także: **PSET** , **\$BGF** , **CONFIG GRAPHLCD** , **LINE** , **CIRCLE** , **SHOWPICE**

Przykład:

```
'-----
'                               (c) 2001-2002 MCS Electronics
'-----

'Sposób podłączenia wyświetlacza LCD:
```

'końcówka		podłączona do
'1	GND	GND
'2	GND	GND
'3	+5V	+5V
'4	-9V	-9V potencjometr
'5	/WR	PORTC.0
'6	/RD	PORTC.1
'7	/CE	PORTC.2
'8	C/D	PORTC.3
'9	NC	nie podłączone
'10	RESET	PORTC.4
'11-18	D0-D7	PA
'19	FS	PORTC.5
'20	NC	nie podłączone

\$crystal = 8000000

'Najpierw definiujemy, że używany będzie graficzny wyświetlacz LCD

'Na razie tylko wyświetlacze 240*64 są obsługiwane

Config Graphlcd = 240 * 128 , Dataport = Porta , Controlport = Portc ,
Ce = 2 , Cd = 3 , Wr = 0 , Rd = 1 , Reset = 4 , Fs = 5 , Mode = 8

'Dataport określa port do jakiego podłączono szynę danych LCD

'Controlport określa który port jest używany do sterowania LCD

'CE, CD itd. to numery bitów w porcie CONTROLPORT

'Na przykład CE = 2 gdyż jest podłączony do końcówki PORTC.2

'Mode = 8 daje 240 / 8 = 30 kolumn , Mode=6 daje 240 / 6 = 40 kolumn

'Deklaracje (y nie używane)

Dim X As Byte , **Y As** Byte

'Kasujemy ekran wyświetlacza (tekst oraz grafikę)

Cls

'Inne opcje to:

' CLS TEXT kasuje tylko stronę tekstową

' CLS GRAPH kasuje tylko stronę graficzną

Cursor Off

'Teraz napiszemy jakiś tekst

Lcd "MCS Electronics"

'i jeszcze jakieś inne w linii 2

Locate 2 , 1 : **Lcd** "T6963c support"

Locate 16 , 1 : **Lcd** "A to będzie w linii 16 (najniższej)"

Wait 2

'SHOWPIC X , Y , etykieta

'Etykieta określa początek danych gdzie zapisany jest obrazek

Showpic 0 , 0 , **Plaatje**

Wait 2

Cls Text

'skasuj tekst

End

'Tutaj znajduje się obrazek

Plaetje:

'Dyrektywa \$BGF spowoduje umieszczenie danych obrazka w tym miejscu

\$bfg "mcs.bgf"

'Możesz dodać inne obrazki tutaj

Przeznaczenie:

Wyświetla obrazek zapisany w formacie BGF i umieszczony w pamięci EEPROM na ekranie graficznego wyświetlacza LCD.

Składnia:

SHOWPICE *x* , *y* , *etykieta*

gdzie:

x , *y*
etykieta

punkt w którym ma się znaleźć lewy górny róg obrazka,
adres określony etykietą, pod jakim znajdują się dane
obrazka.

Opis:

Instrukcja SHOWPICE potrafi wyświetlić bitmapy monochromatyczne poddane konwersji do pliku .BGF narzędziem znajdującym się w menu **Tools | Graphic Converter**. Dane obrazka są pobierane z wewnętrznej pamięci EEPROM.

Parametry *x* oraz *y* określają gdzie obrazek ma być umieszczony na ekranie. Muszą to być wartości podzielne przez 8, albo równe 0. Wysokość obrazka jak i jego szerokość także muszą być wielokrotnością liczby 8.

Podana etykieta określa gdzie w pamięci kodu znajdują się dane obrazka. Nazwę pliku obrazka należy umieścić w dyrektywie **\$BGF** pod wspomnianą etykietą.

Uwaga! Przed podaniem dyrektywy **\$BGF**, należy najpierw określić dyrektywą **\$EEPROM**, że dane mają trafić do pamięci EEPROM.

Można umieścić wiele obrazków – o ile pozwala na to rozmiar pamięci EEPROM – przy czym dla każdego z nich należy określić osobną etykietę.

Uwaga! Dane obrazka w pliku BGF są spakowane metodą RLE (Run Length Encoded – opartą na eliminacji powtórzeń bajtów – *przyp. tłumacza*) by zmniejszyć rozmiar zajmowanych danych.

Zobacz także: **PSET** , **\$BGF** , **CONFIG GRAPHLCD** , **LINE** , **CIRCLE** , **SHOWPIC**

Przykład:

```
'-----  
'                                     (c) 2001-2002 MCS Electronics  
'-----  
  
'Sposób podłączenia wyświetlacza LCD:  
'końcówka          podłączona do  
'1          GND          GND  
'2          GND          GND  
'3          +5V          +5V  
'4          -9V          -9V potencjometr  
'5          /WR          PORTC.0  
'6          /RD          PORTC.1  
'7          /CE          PORTC.2  
'8          C/D          PORTC.3  
'9          NC          nie podłączone  
'10         RESET        PORTC.4  
'11-18      D0-D7        PA  
'19         FS          PORTC.5  
'20         NC          nie podłączone  
  
$crystal = 8000000  
$regfile = "8535def.dat"
```

```

'Najpierw definiujemy, że używany będzie graficzny wyświetlacz LCD
'Na razie tylko wyświetlacze 240*64 są obsługiwane
Config Graphlcd = 240 * 128 , Dataport = Porta , Controlport = Portc ,
Ce = 2 , Cd = 3 , Wr = 0 , Rd = 1 , Reset = 4 , Fs = 5 , Mode = 8
'Dataport określa port do jakiego podłączono szynę danych LCD
'Controlport określa który port jest używany do sterowania LCD
'CE, CD itd. to numery bitów w porcie CONTROLPORT
'Na przykład CE = 2 gdyż jest podłączony do końcówki PORTC.2
'Mode = 8 daje 240 / 8 = 30 kolumn , Mode=6 daje 240 / 6 = 40 kolumn

'Musimy załadować dane obrazka do pamięci EEPROM i dlatego musimy
'umieścić dyrektywę $EEPROM
'UWAGA! Dane muszą być umieszczone wcześniej w pamięci EEPROM i należy
'to zrobić przed instrukcją SHOWPICE.
$eeprom
Plaatje:
'Dyrektywa $BGF załaduje dane obrazka do EEPROM lub FLASH w zależności
'od dyrektyw $DATA oraz $EEPROM.
$bgf "mcs.bgf"
'przełączamy z powrotem na dane w pamięci kodu (FLASH)
$data

'SHOWPICE X , Y , etykieta
'Etykieta określa początek danych gdzie zapisany jest obrazek
Showpice 0 , 0 , Plaatje
Wait 2
End

```

SIN() (Nowość w wersji 1.11.6.8)

Przeznaczenie:

Zwraca wartość sinusa kąta podanego w radianach.

Składnia:

zmienna = **SIN**(*liczba*)

gdzie:

<i>zmienna</i>	dowolna zmienna typu single, do której wpisany będzie wynik działania funkcji,
<i>liczba</i>	liczba której wartość sinusa należy obliczyć.

Opis:

Wszystkie funkcje trygonometryczne używają zapisu wartości kątów w radianach. Aby przeliczyć stopnie na radiany należy użyć funkcji DEG2RAD. Zamianę odwrotną zapewnia instrukcja RAD2DEG.

Zobacz także: **RAD2DEG** , **DEG2RAD** , **ATN** , **COS**

Przykład: Zobacz temat **Biblioteka FP_TRIG**

SINH() (Nowość w wersji 1.11.6.8)

Przeznaczenie:

Zwraca wartość sinusa hyperbolicznego kąta podanego w radianach.

Składnia:

zmienna = **SINH**(*liczba*)

gdzie:

zmienna dowolna zmienna typu single, do której wpisany będzie
wynika działania funkcji,
liczba liczba której wartość sinusa hyperbolicznego należy obliczyć.

Opis:

Wszystkie funkcje trygonometryczne używają zapisu wartości kątów w radianach. Aby przeliczyć stopnie na radiany należy użyć funkcji DEG2RAD. Zamianę odwrotną zapewnia instrukcja RAD2DEG.

Zobacz także: RAD2DEG , DEG2RAD , ATN , SIN , COS , TANH , COSH

Przykład: Zobacz temat Biblioteka FP_TRIG

SONYSEND (Nowość w wersji 1.11.6.8)

Przeznaczenie:

Wysyła sygnały pilota zdalnego sterowania w standardzie Sony.

Składnia:

SONYSEND adres

gdzie:

adres rozkaz (adres) jaki ma być wysłany.

Opis:

Poniżej znajduje się tabela z przykładowymi kodami (adresami) funkcji realizowanymi przez wybrane urządzenia.

Odtwarzacz CD SONY (pilot RM-DX55)

Funkcja	Hex	Bin
Power	&HA91	1010 1001 0001
Play	&H4D1	0100 1101 0001
Stop	&H1D1	0001 1101 0001
Pause	&H9D1	1001 1101 0001
Continue	&HB91	1011 1001 0001
Shuffle	&HAD1	1010 1101 0001
Program	&HF91	1111 1001 0001
Disc	&H531	0101 0011 0001
1	&H011	0000 0001 0001
2	&H811	1000 0001 0001
3	&H411	0100 0001 0001
4	&HC11	1100 0001 0001
5	&H211	0010 0001 0001
6	&HA11	1010 0001 0001
7	&H611	0110 0001 0001
8	&HE11	1110 0001 0001
9	&H111	0001 0001 0001
0	&H051	0000 0101 0001
>10	&HE51	1110 0101 0001
Enter	&HD11	1101 0001 0001
Clear	&HF11	1111 0001 0001
Repeat	&H351	0011 0101 0001
Disc -	&HBD1	1011 1101 0001
Disc +	&H7D1	0111 1101 0001
<<	&H0D1	0000 1101 0001
>>	&H8D1	1000 1101 0001
<<	&HCD1	1100 1101 0001
>>	&H2D1	0010 1101 0001

Magnetofon SONY (pilot RM-J901)

Deck A		
Stop	&H1C1	0001 1100 0001
Play >	&H4C1	0100 1100 0001
Play <	&HEC1	1110 1100 0001
>>	&H2C1	0010 1100 0001
<<	&HCC1	1100 1100 0001
Record	&H6C1	0110 1100 0001
Pause	&H9C1	1001 1100 0001
Deck B		
Stop	&H18E	0001 1000 1110
Play >	&H58E	0101 1000 1110
Play <	&H04E	0000 0100 1110
>>	&H38E	0011 1000 1110
<<	&HD8E	1101 1000 1110
Record	&H78E	0111 1000 1110
Pause	&H98E	1001 1000 1110

Telewizor SONY (pilot RM-694)

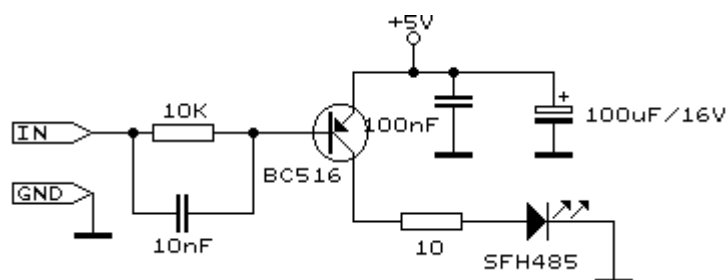
Program +	&H090	0000 1001 0000
Program -	&H890	1000 1001 0000
Volume +	&H490	0100 1001 0000
Volume -	&HC90	1100 1001 0000
Power	&HA90	1010 1001 0000
Mute	&H290	0010 1001 0000
1	&H010	0000 0001 0000
2	&H810	1000 0001 0000
3	&H410	0100 0001 0000
4	&HC10	1100 0001 0000
5	&H210	0010 0001 0000
6	&HA10	1010 0001 0000
7	&H610	0110 0001 0000
8	&HE10	1110 0001 0000
9	&H110	0001 0001 0000
0	&H910	1001 0001 0000
-/--	&HB90	1011 1001 0000

Więcej informacji na temat zdalnego sterowania w standardzie Sony można znaleźć na stronie: <http://www.fet.uni-hannover.de/purnhage/>

Implementacja nadajnika

W przykładzie wykorzystywany jest kontroler AT90s2313, który używa końcówki PortB.3 jako wyjście OC1A. Zajrzyj do not katalogowych, by określić numer końcówki dla innych układów.

Schemat przykładowego wzmacniacza sygnału podczerwieni pokazano poniżej:



Rysunek 24 Przykładowy wzmacniacz wyjściowy nadajnika.

Zobacz także: **RC5SEND**

Przykład:

```
'-----  
'  
'                               SONYSEND.BAS  
'                               (c) 2002 MCS Electronics  
'Kod w oparciu o notę aplikacyjną, którą napisał Ger Langezaal  
'+5V <---[A Led K]---[220 Ohm]---> PB.3 dla 90s2313.  
'RC5SEND używa licznika TIMER1, nie są używane przerwania  
'Rezystor musi być dołączony do końcówki OC1(A), w tym wypadku PB.3  
'-----  
$regfile = "2313def.dat"  
$crystal = 4000000  
  
Do  
    Waitms 500  
    Sonysend &HA90  
Loop  
  
End
```

SOUND

Przeznaczenie:

Wysyła ciąg impulsów na wybraną końcówkę portu.

Składnia:

SOUND *pin* , *il_impulsów* , *czas_impulsu*

gdzie:

<i>pin</i>	nazwa końcówki portu będącą linią wyjściową,
<i>il_impulsów</i>	ilość generowanych pełnych impulsów,
<i>czas_impulsu</i>	czas trwania pojedynczego impulsu.

Opis:

Gdy do podanej linii portu dołączony będzie głośniczek lub buzzer, można użyć instrukcji SOUND w celu generacji pojedynczych tonów.

Końcówka portu jest ustawiana w stan 0 i 1 w czasie trwania pełnego impulsu. Czas trwania tej sekwencji jest określony parametrem *czas_impulsu*. Liczba tych impulsów jest określona jako drugi parametr.

Uwaga! Instrukcja SOUND nie jest przeznaczona do generowania przebiegów o określonej częstotliwości. W tym celu najlepiej użyć przerwań jednego z sprzętowych liczników.

Przykład:

```
Sound Portb.1 , 10000, 10      'generuj jakiś dźwięk  
  
End
```

SPACE()

Przeznaczenie:

Zwraca ciąg znaków wypełniony znakiem spacji.

Składnia:

zmienna = **SPACE**(*il_spacji*)

gdzie:

zmienna
il_spacji

zmienna typu String,
długość generowanego ciągu.

Opis:

Podanie zera jako parametru funkcji spowoduje utworzenie ciągu o długości 255 znaków. Spowodowane jest to brakiem testowania czy parametr jest równy 0.

Zobacz także: **STRING** , **SPC**

Przykład:

```
Dim s As String * 15, z As String * 15

s = Space(5)
Print " { " ; s ; " } "           '{      }

Dim A As Byte

A = 3
S = Space(a)
```

SPC()

Przeznaczenie:

Drukuje określona liczbę spacji.

Składnia:

PRINT **SPC**(*il_spacji*)

gdzie:

il_spacji

ilość drukowanych spacji.

Opis:

Funkcja SPC może być używana w połączeniu z instrukcją **PRINT** oraz **LCD**.

Istnieje podobna funkcja **SPACE**. Różnicą jest, iż funkcja **SPACE** zwraca ciąg znaków składających się ze spacji, który można przypisać zmiennej tekstowej. Funkcja **SPC** może jedynie drukować spacje.

Używanie funkcji **SPACE** do drukowania spacji, powoduje, że tworzona jest tymczasowa zmienna w pamięci. Zaś funkcja **SPC** nie potrzebuje tej zmiennej gdyż działa bezpośrednio.

Uwaga! Podanie jako ilość spacji liczby zero spowoduje wydrukowanie 255 znaków spacji. Jest to spowodowane brakiem testowania czy podana liczba jest zerem.

Zobacz także: **SPACE**

Przykład:

```
Dim s As String * 15 , z As String * 15

Print "{ " ; Spc(5) ; " } "           '{      }
Lcd "{ " ; Spc(5) ; " } "             '{      }
```

SPIIN

Przeznaczenie:

Odczytuje dane przesłane przez interfejs SPI.

Składnia:

SPIIN *zmienna* , *il_bajtów*

gdzie:

zmienna
il_bajtów

zmienna do której wpisane będą odebrane bajty,
ilość odczytanych bajtów.

Zobacz także: **SPIOUT** , **SPIINIT** , **CONFIG SPI** , **SPIMOVE**

Przykład:

```
Dim a(10) As Byte

Config SPI = Soft, DIN = Pinb.0, DOUT = Portb.1, SS = Portb.2, CLOCK =
Portb.3

Spiinit                                     'inicjalizacja interfejsu
Spiin a(1) , 4                             'odczytuje 4 bajty
```

SPIINIT

Przeznaczenie:

Inicjalizuje programowy lub sprzętowy interfejs SPI.

Składnia:

SPIINIT

Opis:

Po skonfigurowaniu nazw końcówek używanych przez SPI, należy dokonać ich inicjalizacji, by ustalić kierunki ich działania. Gdy końcówki interfejsu SPI nie są używane w inny sposób w programie, instrukcję SPIINIT należy wykonać tylko raz.

Gdy inne procedury zmieniają stan końcówek interfejsu SPI, należy powtórnie użyć instrukcji SPIINIT, przed użyciem SPIIN lub SPIOUT.

Zobacz także: **SPIIN** , **SPIOUT** , **CONFIG SPI** , **SPIMOVE**

Asembler:

Wywołuje procedurę `_init_spi`

Przykład:

```
Dim a(10) As Byte

Config SPI = Soft, DIN = Pinb.0, DOUT = Portb.1, SS = Portb.2, CLOCK =
Portb.3

Spiinit                                     'inicjalizacja interfejsu
Spiin a(1) , 4                             'odczytujemy 4 bajty
```

SPIMOVE()

Przeznaczenie:

Przesyła i odbiera dane przez interfejs SPI.

Składnia:

zmienna = **SPIMOVE**(*bajt*)

gdzie:

zmienna
bajt

zmienna do której wpisany będzie odebrany bajt,
bajt którego zawartość ma być wysłana.

Zobacz także: **SPIIN** , **SPIINIT** , **CONFIG SPI**

Przykład:

```
Config SPI = Soft, DIN = Pinb.0, DOUT = Portb.1, SS = Portb.2, CLOCK =  
Portb.3  
  
Dim a(10) As Byte , X As Byte  
  
Spiout a(1) , 5           'wyślemy 5 bajtów  
Spiout X , 1              'teraz jeden  
A(1) = Spimove(5)         'i wyślemy liczbę 5 oraz odbierzemy bajt  
                           'który trafi do zmiennej tablicowej  
  
End
```

SPIOUT

Przeznaczenie:

Wysyła dane przez interfejs SPI.

Składnia:

SPIOUT *zmienna* , *il_bajtów*

gdzie:

zmienna
il_bajtów

zmienna z której pochodzą wysyłane bajty,
ilość wysyłanych bajtów.

Zobacz także: **SPIIN** , **SPIINIT** , **CONFIG SPI** , **SPIMOVE**

Przykład:

```
Config SPI = Soft, DIN = Pinb.0, DOUT = Portb.1, SS = Portb.2, CLOCK =  
Portb.3  
  
Dim a(10) As Byte , X As Byte  
  
Spiout a(1) , 5           'wyślemy 5 bajtów  
Spiout X , 1              'i jeszcze jeden
```

SQR() *(Nowość w wersji 1.11.6.8)*

Przeznaczenie:

Zwraca wartość pierwiastka kwadratowego podanej liczby.

Składnia:

zmienna = **SQR**(*wartość*)

gdzie:

zmienna
wartość

zmienna w której umieszczony będzie wynik działania
funkcji,
zmienna której wartość pierwiastka należy obliczyć.

Opis:

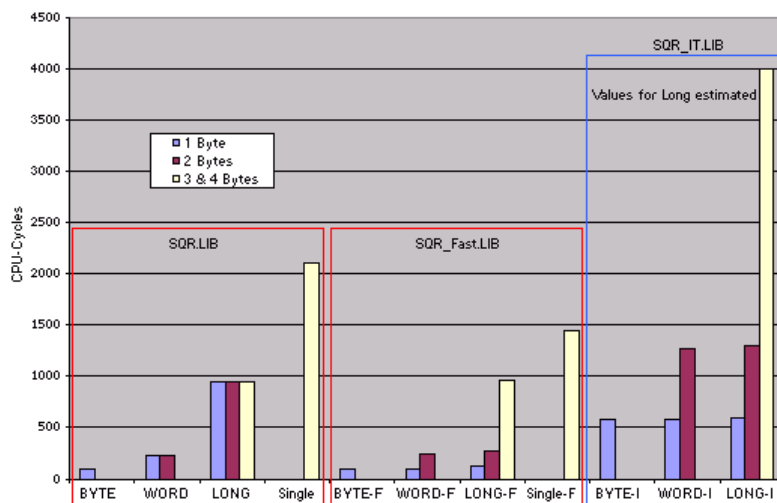
Gdy jako parametr funkcji SQR podano liczbę typu Single, należy koniecznie w programie

dodać dyrektywę:

```
$lib = "FP_TRIG.LBX"
```

Gdy funkcja SQR jest wywoływana z parametrem typu Byte, Integer, Word lub Long, używana jest wtedy procedura SQR pochodząca z biblioteki MCS.LBX. Jako alternatywę można użyć biblioteki SQR_IT.LBX lub SQR.LBX.

Do obliczania wartości funkcji SQR można wykorzystać różne algorytmy. Standardowo używany jest najszybszy. Poniższy rysunek pokazuje różnice czasowe odnośnie do trzech metod:



Rysunek 25 Porównanie czasu działania różnych metod obliczania funkcji SQR().

Przykład:

```
Dim A As Single
A = 9.0
A = Sqr(A)
Print A
```

'wydrukuje 3.0

START

Przeznaczenie:

Uruchamia określone urządzenie.

Składnia:

START urządzenie

gdzie:

urządzenie

nazwa symboliczna uruchamianego urządzenia.

Opis:

Instrukcja START powoduje włączenie (uruchomienie) podanego urządzenia. Jako parametr należy podać nazwę symboliczną urządzenia. W języku BASCOM BASIC zdefiniowano następujące nazwy urządzeń:

Tabela 16 Lista urządzeń dla instrukcji START i STOP.

Nazwa	Typ urządzenia
TIMER0	Licznik TIMER0
TIMER1	Licznik TIMER1

Nazwa	Typ urządzenia
COUNTER0 ^{*)}	Licznik TIMER0
COUNTER1 ^{*)}	Licznik TIMER1
WATCHDOG	Licznik układu WATCHDOG
AC	Wbudowany komparator analogowy
ADC	Wbudowany przetwornik A/D

^{*)} Nazwa COUNTERx może być stosowana zamiennie z TIMERx.

Uwagi!

Liczniki-czasomierze muszą zostać włączone by mogły generować przerwania (przy wyłączonym zewnętrznym bramkowaniu).

Instrukcja START z parametrem **AC** lub **ADC** włącza zasilanie urządzenia co spowoduje jego gotowość do pracy.

Zobacz także: STOP

Przykład:

```

'-----
'               ADC.BAS
'   demonstruje funkcję GETADC() działającą z procesorem 90s8535
'-----

'Konfigurujemy tryb pracy na SINGLE i prescaler na AUTO
'Tryb SINGLE musi być wybrany dla poprawnego działania GETADC()

'Prescaler dzieli sygnał zegarowy przez 2,4,8,15,32,64 lub 128,
'ponieważ przetwornik ADC działa z częstotliwością 50-200KHz
'Ustawienie AUTO spowoduje, że dzielnik zostanie tak ustawiony aby
'wybrana częstotliwość była najwyższa z możliwych.
Config Adc = Single , Prescaler = Auto
'teraz włączamy przetwornik (włącza zasilanie)
Start Adc

'Instrukcją STOP ADC, można wyłączyć przetwornik
'Stop Adc

Dim W As Word , Channel As Byte

Channel = 0

'odczytujemy przetworzoną wartość z kolejnych kanałów
Do
  W = Getadc(channel)
  Print "Kanał " ; Channel ; " wartość " ; W
  Incr Channel
  If Channel > 7 Then Channel = 0
Loop

End

```

STCHECK

Przeznaczenie:

Wywołuje procedurę sprawdzającą różnorodne błędy związane z przepełnieniem stosu. Procedura ta służy do wskazania tych błędów, które należy usunąć.

Składnia:

STCHECK

Opis:

Ponieważ BASCOM BASIC używa aż trzech stosów, może to stanowić nie lada problem. Instrukcja STCHECK pomaga w ustaleniu czy nie są one powodem błędnej pracy programu. Na podstawie przykładu zawartego w pliku STACK.BAS, postaram się wytłumaczyć jak ustawić poprawne wartości.

UWAGA! Instrukcja STCHECK powinna być usunięta w finalnej wersji programu. Po stwierdzeniu, że wszystko działa poprawnie, powinno się usunąć instrukcje STCHECK z programu. Zmniejszy to czas jego wykonywania oraz rozmiar kodu wynikowego.

Używane przez program ustawienia to:

Stos sprzętowy – 8 bajtów,
Stos programowy – 2 bajty,
Obszar „ramki” – 14 bajtów.

Poniżej znajduje się fragment zawartości pamięci układu AT90s2313 który posłuży nam jako przykład.:

```
C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF
FR FR FR FR FR FR FR FR

D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF
FR FR FR FR FR FR YY YY SP SP SP SP SP SP SP SP
```

Ostatnia komórka pamięci SRAM ma adres **&H0DF** (szesnastkowo), sprzętowy stos (oznaczony tutaj jako **SP**) zajmuje tutaj 8 bajtów od **&H0D8** do **&H0DF**.

Gdy wykonywany jest skok do podprogramu lub na stos odkładane są zawartości rejestrów, zostają one umieszczone na wierzchołku stosu, wskazywanym przez rejestr wskaźnika stosu (**SP**). Odkładanie wartości na stos powoduje jego zmniejszanie.

Wywołanie podprogramu odkłada na stosie 2 bajty (adres powrotny) więc wskaźnik stosu zostaje zmniejszony o 2 i wskazuje **&H0DD**. Gdy na stos odłożone będzie już 8 bajtów, wskaźnik stosu będzie wskazywał na bajt **&H0D7**. Jeśli zatem na stos trafi kolejny bajt, fragment programowego stosu (oznaczony tutaj jako **YY**) zostanie zniszczony.

Programowy stos jest umieszczony zawsze „po” sprzętowym stosie, i także rozrasta się w dół (w kierunku mniejszych adresów). Jako wskaźnik jest tu używany rejestr **Y** (para rejestrów R28 i R29).

Ponieważ wskaźnik stosu **Y** jest zmniejszany przed zapisem danych, wskaźnik musi wskazywać na wierzchołek stosu plus jedna komórka. Tutaj wskazuje na bajt **&H0D8**, czyli koniec przestrzeni stosu sprzętowego. Na przykład instrukcja:

```
St -Y,R24
```

zmieni wartość wskaźnika stosu na **&H0D8 - 1 = &H0D7** i umieści zawartość R24 pod adresem **&H0D7**. Gdyby na stosie odłożono dwa bajty zajęły by one komórki pod adresami **&H0D7** i **&H0D6**.

Gdy wskaźnik stosu wskazywałby na **&H0D6** i inna instrukcja `St -Y,R24` była by użyta, wtedy dana trafiłaby pod adres **&H0D5**, który jest końcem przestrzeni zajmowanej przez obszar „ramki” używanej jako pamięć tymczasowa.

Obszar ramki (oznaczony tutaj jako **FR**) zajmuje adresy od **&H0C8** do **&H0D5**. Przepełnienie „ramki” spowodowałoby nieuchronne zniszczenie danych odłożonych na programowym stosie. Gdyby stos programowy nie jest używany i ma rozmiar 0 bajtów, przepełnienie „ramki” powodowałoby nadpisanie bajtów zajmowanych przez sprzętowy stos.

Jak określić poprawne wartości?

Procedura sprawdzająca stan stosu może być używana do określenia czy nie występuje przepełnienie któregośkolwiek ze stosów. W szczególności sprawdza czy:

- SP wskazuje poniżej przestrzeni sprzętowego stosu, w tym wypadku poniżej **&H0D8**,
- Y wskazuje poniżej przestrzeni programowego stosu, w tym wypadku poniżej **&H0D5**,

- czy dane z ramki nie próbują wyjść poza jej obszar, w tym wypadku poza adres &H0D6 (wskaźnik „ramki” przesuwają się w kierunku większych adresów!).

Kiedy używany jest programowy stos?

Programowy stos jest używany przede wszystkim przez procedury i funkcje napisane przez użytkownika.

Na stosie tym odkładane są adresy przekazywanych parametrów – każdy parametr to 2 bajty – także te przekazywane przez wartość (argument **BYVAL**). Tak samo każda zmienna lokalna zajmuje dwa bajty z puli stosu. Dla przykładu:

```
Call MySub(x , y) użyje czterech bajtów (2 parametry * 2 bajty)
Local z As Byte użyje kolejne 2 bajty
```

Bajty te zostaną oczywiście zwolnione i przekazane do puli stosu, przy wyjściu z procedury lub funkcji.

Gdy procedura lub funkcja wywołuje jeszcze inną funkcję lub procedurę, zapotrzebowanie na stos zwiększa się:

```
Sub mysub(x As Byte , y As Byte)
    Call testsub(r As Byte) 'wykorzystuje jeszcze 2 bajty stosu.
```

Procedury bezparametrowe, jak na przykład:

```
Call mytest()
```

nie używają przestrzeni programowego stosu.

Kiedy używana jest przestrzeń „ramki”?

Najczęściej „ramka” używana jest podczas procedur wewnętrznych konwersji z liczby na jej reprezentację tekstową, jak w przykładzie:

```
Print b (gdzie b jest zmienną typu Byte)
```

Różne typy zmiennych używają różną ilość bajtów z puli „ramki”:

- typ Byte używa maksymalnie 4 bajtów (123+0),
- typ Integer używa maksymalnie 7 bajtów (-12345+0),
- typ Long używa maksymalnie 16 bajtów,
- typ Single używa maksymalnie 24 bajty.

„Ramka” jest także używana, gdy wykonywane są operacje łączenia dwóch ciągów tekstowych, przy czym jeden z nich jest używany jako zmienna docelowa:

```
s = "abcd" + s
```

W powyższym przykładzie przypisywana jest nowa wartość zmiennej *s*, lecz oryginalna jej zawartość musi zostać zapamiętana przed wykonaniem operacji dodawania i przypisania. Kopia zmiennej *s* jest zatem umieszczana w przestrzeni „ramki”. Jeśli więc przewidywana długość zmiennej *s* została zdefiniowana na 20 znaków, wymagane jest zapamiętanie 21 znaków (20 znaków plus znak końca ciągu).

Przebieg „ramki” jest używana często przy procedurach i funkcjach użytkownika. I tak:

- gdy jako parametr jest przekazywana wartość zmiennej (argument **BYVAL**), kopia zawartości zmiennej jest umieszczana w przestrzeni „ramki”, a adres kopii jest przekazywany procedurze,
- gdy przekazywane są bajty używany jest jeden bajt ramki, a zmienne typu Long zajmują już 4 bajty,
- gdy używana jest zmienna lokalna typu Long (dla przykładu) także używane są 4 bajty „ramki”.

Przebieg ramki jest używana ponownie i to nawet przez programowy jak i sprzętowy stos. Dlatego tak trudno jest określić właściwie obszary stosów i ramki.

Instrukcja STCHECK musi być wywoływana z najniższego poziomu zagnieżdżonych funkcji, procedur lub podprogramów.

```
Gosub test

test:
  Gosub test1
Return

test1:
  'jest to najniższy poziom, więc umieścimy instrukcje tutaj
  Stcheck
Return
```

Instrukcja STCHECK używa jednej zmiennej nazwanej `Error`, którą należy zdefiniować osobno:

```
Dim Error As Byte
```

Zmienna `Error` przyjmuje wartość:

- 1 : jeśli sprzętowy stos rozrósł się za bardzo w dół i może zniszczyć obszar programowego stosu,
- 2 : jeśli programowy stos rozrósł się za bardzo w dół i może zniszczyć obszar zajmowany przez „ramkę”
- 3 : jeśli przestrzeń „ramki” powiększyła się w górę i może zniszczyć obszar programowego stosu.

Ostatnie dwa błędy nie są aż takie złe, jeśli weźmiesz pod uwagę fakt, że gdy przestrzeń programowego stosu nie jest wykorzystywana do przekazywania danych, może zostać użyta jako przestrzeń „ramki”. Zamieszałem, prawda?

Asembler:

Instrukcja STCHECK wywołuje procedurę: `_StackCheck`. Używa ona rejestrów R24 i R25 które są zapamiętywane i odtwarzane przy wyjściu z procedury.

Ponieważ wywołanie procedury zajmuje 2 bajty z puli sprzętowego stosu, a także 2 bajty na zapamiętanie zawartości R24 i R25; program używa o 4 bajty więcej niż finalny program, w którym nie powinno się używać instrukcji STCHECK.

Przykład:

Poniżej znajduje się listing pliku `STACK.BAS`, znajdujący się w katalogu `SAMPLES`. Używa on kompilacji warunkowej, która umożliwia przetestowanie różnorodnych błędów.

```
'Ten przykład pokazuje jak sprawdzić rozmiar używanych stosów.

'Uwaga! Wywoływana procedura (_STACKCHECK) używa także 4 bajtów
'z przestrzeni sprzętowego stosu.
'Więc jeśli twój program będzie działać, możesz zmniejszyć o 4 bajty
'rozmiar sprzętowego stosu w wersji finalnej, z której należy usunąć
'instrukcje STCHECK.
'
' testmode =0   działa normalnie
' testmode =1   używa za dużo sprzętowego stosu
' testmode =2   używa za dużo programowego stosu
' testmode =3   używa za dużo przestrzeni „ramki”
Const Testmode = 0
'Skompiluj i przetestuj program z różnymi ustawieniami Testmode (0-3)

'Musisz zdefiniować zmienną ERROR typu Byte!!
Dim Error As Byte

#if Testmode = 2
```

```

    Declare Sub Pass(z As Long , Byval K As Long)
#else
    Declare Sub Pass()
#endif

Dim I As Long
I = 2
Print I
'wywołanie procedury w programie na najniższym poziomie
'normalnie wewnątrz funkcji lub procedury

#if Testmode = 2
    Call Pass(i , 1)
#else
    Call Pass()
#endif
End

#if Testmode = 2
    Sub Pass(z As Long , Byval K As Long)
#else
    Sub Pass()
#endif
    #if Testmode = 3
        Local S As String * 13
    #else
        Local S As String * 8
    #endif

    Print I
    Gosub Test
End Sub

Test:
#if Testmode = 1
    Push r0 ;zabierze trochę sprzętowego stosu
    Push r1
    Push r2
#endif

    ' *** teraz wywołujemy procedurę ***
    Stcheck
    ' *** jeśli zmienna Error <> 0 to wystąpiły problemy ***
#if Testmode = 1
    Pop r2
    Pop r1
    Pop r0
#endif

Return

```

STOP

Przeznaczenie:

Zatrzymuje działanie programu lub określone urządzenie.

Składnia:

STOP

STOP *urządzenie*

gdzie:

urządzenie

nazwa symboliczna uruchamianego urządzenia.

Opis:

Pierwszy format instrukcji powoduje zatrzymanie działania programu. W tym celu jest wykonywana pusta pętla, podobnie jak w instrukcji END. Jednak przy instrukcji STOP nie są wyłączane przerwania.

Drugi format instrukcji STOP powoduje wyłączenie (zatrzymanie) podanego urządzenia. Jako parametr należy podać nazwę symboliczną urządzenia. W języku BASCOM BASIC zdefiniowano następujące nazwy urządzeń:

Tabela 17 Lista urządzeń dla instrukcji START i STOP.

Nazwa	Typ urządzenia
TIMER0	Licznik TIMER0
TIMER1	Licznik TIMER1
COUNTER0 ^{*)}	Licznik TIMER0
COUNTER1 ^{*)}	Licznik TIMER1
WATCHDOG	Licznik układu WATCHDOG
AC	Wbudowany komparator analogowy
ADC	Wbudowany przetwornik A/D

^{*)} Nazwa COUNTERx może być stosowana zamiennie z TIMERx.

Uwaga! Instrukcja STOP z parametrem **AC** lub **ADC** wyłącza zasilanie urządzenia.

Zobacz także: **START** , **END**

Przykład:

```
'-----  
'                               ADC.BAS  
'  demonstruje funkcję GETADC() działającą z procesorem 90s8535  
'-----  
  
'Konfigurujemy tryb pracy na SINGLE i prescaler na AUTO  
'Tryb SINGLE musi być wybrany dla poprawnego działania GETADC()  
  
'Prescaler dzieli sygnał zegarowy przez 2,4,8,15,32,64 lub 128,  
'ponieważ przetwornik ADC działa z częstotliwością 50-200KHz  
'Ustawienie AUTO spowoduje, że dzielnik zostanie tak ustawiony aby  
'wybrana częstotliwość była najwyższa z możliwych.  
Config Adc = Single , Prescaler = Auto  
'teraz włączamy przetwornik (włącza zasilanie)  
Start Adc  
  
'Instrukcją STOP ADC, można wyłączyć przetwornik  
'Stop Adc  
  
Dim W As Word , Channel As Byte  
  
Channel = 0  
  
'odczytujemy przetworzoną wartość z kolejnych kanałów  
Do  
  W = Getadc(channel)  
  Print "Kanał " ; Channel ; " wartość " ; W  
  Incr Channel  
  If Channel > 7 Then Channel = 0  
Loop
```

End

STR()

Przeznaczenie:

Zwraca tekstową reprezentację podanej liczby.

Składnia:

zmienna = **STR**(*x*)

gdzie:

<i>zmienna</i>	zmienna typu String do której wpisany będzie wynik działania funkcji,
<i>x</i>	dowolna zmienna lub stała, poddana konwersji.

Opis:

Zmienna String musi mieć odpowiednią długość, by pomieścić wszystkie znaki.

Zobacz także: **VAL** , **HEX** , **HEXVAL**

Różnice w stosunku do QBasic-a.

W języku QBasic funkcja STR() dodaje na początku spację, w języku BASCOM BASIC spacja ta nie występuje.

Przykład:

```
Dim a As Byte, S As XRAM String * 10

a = 123
s = Str(a)
Print s

End
```

STRING()

Przeznaczenie:

Zwraca ciąg znaków wypełniony znakiem o podanym kodzie ASCII.

Składnia:

zmienna = **STRING**(*il_znaków* , *kod_znaku*)

gdzie:

<i>zmienna</i>	zmienna typu String,
<i>il_znaków</i>	długość generowanego ciągu,
<i>kod_znaku</i>	kod ASCII znaku wypełniającego ciąg.

Opis:

Ponieważ ciągi znaków w języku BASCOM BASIC zakończone są zawsze znakiem o kodzie zero, jako kod znaku nie można podać 0. Gdyż tak wygenerowany ciąg w rozumieniu języka BASCOM BASIC będzie pusty.

Podanie zera jako parametru *il_znaków* funkcji STRING, spowoduje utworzenie ciągu o długości 255. Jest to wynikiem braku testowania czy parametr jest równy 0.

Zobacz także: **SPACE**

Przykład:

```
Dim s As String * 15

s = String(5, 65)
Print " { " ; s ; " } "      '{AAAAA}'
```

SUB

Przeznaczenie:

Rozpoczyna treść procedury użytkownika.

Składnia:

```
SUB nazwa [ ( parametr AS typ [, parametr AS typ] ) ]
    instrukcje procedury
END SUB
```

gdzie:

<i>nazwa</i>	nazwa procedury,
<i>parametr</i>	nazwa parametru procedury,
<i>typ</i>	typ przekazywanego parametru.

Opis:

Każda procedura musi być zakończona instrukcją END SUB.

Najlepiej skopiować deklarację procedury (instrukcja DECLARE SUB na początku programu) a następnie skasować słowo DECLARE. Można w ten sposób ustrzec się od błędów, polegających na różnicy pomiędzy częścią deklaracyjną a treścią procedury.

Zobacz temat **DECLARE SUB** by poznać więcej szczegółów.

SWAP

Przeznaczenie:

Zamienia między sobą wartości podanych zmiennych.

Składnia:

```
SWAP zmienna1 , zmienna2
```

gdzie:

<i>zmienna1</i>	zmienna dowolnego typu, oprócz typu Single oraz tablicowego,
<i>zmienna2</i>	zmienna takiego samego typu co <i>zmienna1</i> .

Opis:

Po wykonaniu tej instrukcji *zmienna1* będzie miała wartość *zmiennej2* i na odwrót.

Przykład:

```
Dim a As Integer , b1 As Integer

a = 1 : b1 = 2      'przypisujemy wartości
Swap a, b1          'zamieniamy
Print a ; b1        'wydrukuje 21
```

SYSDAY() (Nowość w wersji 1.11.7.3)

Przeznaczenie:

Zwraca numer dnia liczonego od początku wieku.

Składnia:

```
rezultat = SYSDAY()  
rezultat = SYSDAY( bDMY )  
rezultat = SYSDAY( strData )  
rezultat = SYSDAY( lSysSec )
```

gdzie:

<i>rezultat</i>	zmienna typu Long gdzie funkcja zwróci wynik,
<i>strData</i>	ciąg znaków z zapisaną datą w formacie określonym przez instrukcję CONFIG DATE ,
<i>lSysSec</i>	zmienna typu Long która przechowuje liczbę sekund jaka upłynęła od początku wieku, (SysSec)
<i>bDMR</i>	nazwa zmiennej typu Byte, w której przechowywany jest bieżący dzień. Zaraz po tym bajcie powinny znajdować się jeszcze dwa bajty zawierające kolejno miesiąc i rok (tylko 2 ostatnie cyfry!).

Opis:

Funkcja ta może być wywołana z czterema wariantami parametrów:

1. Bez parametrów. Wtedy funkcja zwraca numer dnia od początku wieku na podstawie zmiennych **programowego zegara** (*_day*, *_month*, *_year*).
2. Ze zdefiniowanym przez użytkownika ciągiem 3 bajtów. Muszą one być zorganizowane podobnie jak zmienne programowego zegara. W pierwszym bajcie należy umieścić dzień, w następnym miesiąc i ostatnim rok (tylko 2 cyfry, bez wieku).
3. Z ciągiem znaków, w którym data przedstawiona jest w formacie określonym przez instrukcję **CONFIG DATE**.
4. Ze zmienną typu Long przechowującą liczbę sekund jaka upłynęła od początku wieku.

Funkcja zwraca liczbę od 0 – dla 2000-01-01 - do 36524 – dla 2099-12-31.

Uwaga! Funkcja zwraca poprawne wartości tylko w XXI wieku (od 2000-01-01 do 2099-12-31).

Zobacz także: **CONFIG DATE** , **CONFG CLOCK** , **SYSSEC** , **Biblioteka DATETIME**

Przykład:

```
Enable Interrupts  
Config Clock = Soft  
Config Date = YMD , Separator = . ' format ANSI  
  
Dim strDate As String * 8  
Dim bDay As Byte , bMonth As Byte , bYear As Byte  
Dim wSysDay As Word  
Dim lSysSec As Long  
  
' Przykład 1 z wewnętrznym zegarem RTC  
_Day = 20 : _Month = 11 : _Year = 2 ' ustaw zegar - dla przykładu  
i testów  
wSysDay = SysDay()  
Print "Dzień " ; Date$ ; " zwraca " ; wSysDay  
' Dzień 02.11.20 zwraca 1054  
  
' Przykład 2 z zdefiniowanym ciągiem 3 bajtów (Dzień / Miesiąc / Rok )  
bDay = 24 : bMonth = 5 : bYear = 8  
wSysDay = SysDay(bDay)  
Print "Dla składników Dzień=" ; bDay ; " Miesiąc=" ; bMonth ; " Rok=" ;  
bYear ; " zwraca " ; wSysDay
```



```

' Dla składników Dzień=24 Miesiąc=5 Rok=8 zwraca 3066

' Przykład 3 z ciągiem zawierającym datę
strDate = "04.10.29"
wSysDay = SysDay(strDate)
Print "Dla dnia " ; strDate ; " zwraca " ; wSysDay
' Dla dnia 04.10.29 zwraca 1763

' Przykład 4 z liczbą sekund od początku wieku
lSysSec = 123456789
wSysDay = SysDay(lSysSec)
Print "Dla liczby " ; lSysSec ; " sekund zwraca " ; wSysDay
' Dla liczby 123456789 sekund zwraca 1428

End

```

SYSSEC() (Nowość w wersji 1.11.7.3)

Przeznaczenie:

Zwraca liczbę sekund jaka upłynęła od początku wieku.

Składnia:

```

rezultat = SYSSEC()
rezultat = SYSSEC( bSMG )
rezultat = SYSSEC( strCzas , strData )
rezultat = SYSSEC( wSysDay )

```

gdzie:

<i>rezultat</i>	zmienna typu Long gdzie funkcja zwróci wynik,
<i>strCzas</i>	ciąg znaków z zapisaną godziną w formacie „gg:mm:ss”,
<i>strData</i>	ciąg znaków z zapisaną datą w formacie ustalonym instrukcją CONFIG DATE ,
<i>wSysDay</i>	zmienna typu Word która przechowuje liczbę dni jaka upłynęła od początku wieku (SysDay),
<i>bSMG</i>	nazwa zmiennej typu Byte, w której przechowywany jest bieżąca liczba sekund. Zaraz po tym bajcie powinno znajdować się jeszcze pięć bajtów zawierających kolejno: minuty, godziny, dzień, miesiąc i rok (2 cyfrowy).

Opis:

Funkcja ta może być wywołana z czterema wariantami parametrów:

1. Bez parametrów. Wtedy funkcja zwraca liczbę sekund na podstawie zmiennych **programowego zegara** (*_sec, _min, _hour, _day, _month, _year*).
2. Ze zdefiniowanym przez użytkownika ciągiem 6 bajtów. Muszą one być zorganizowane podobnie jak zmienne programowego zegara. W pierwszym bajcie należy umieścić liczbę sekund, a w następnych kolejno liczbę minut, godzin oraz dzień, miesiąc i rok. Dlatego funkcja może zwracać liczbę sekund dla każdej kombinacji daty i godziny (z pewnymi ograniczeniami *przyp. tłumacza*).
3. Z dwoma ciągami znaków. W pierwszym czas przedstawiony w formacie „ss:mm:gg”. W drugim datę zapisaną w formacie takim jaki określono przez **CONFIG DATE**.
4. Ze zmienną typu Word przechowującą liczbę dni jaka upłynęła od początku wieku. Funkcja zwraca wtedy liczbę sekund jaka upłynęła od początku wieku do północy (00:00:00) tego dnia.

Funkcja zwraca liczbę od 0 do 2147483647. Zero oznacza północ w dniu 2000-01-01, a 2147483647 godzinę 03:14:07 w dniu 2068-01-19 gdyż zmienna Long nie potrafi pomieścić większej liczby.

Zobacz także: **SYSSECELAPSED** , **SYSDAY** , **SECELAPSED** , **Biblioteka DATETIME**

Przykład:

```
Enable Interrupts
Config Clock = Soft
Config Date = YMD , Separator = . ' format ANSI

Dim strTime As String * 8
Dim strDate As String * 8
Dim bSec As Byte , bMin As Byte , bHour As Byte
Dim bDay As Byte , bMonth As Byte , bYear As Byte
Dim wSysDay As Word
Dim lSysSec As Long

' Przykład 1 z wewnętrznym zegarem RTC
' Ustaw zegar - dla przykładu i testów
_Sec = 17 : _Min = 35 : _Hour = 8 : _Day = 16 : _Month = 4 : _Year = 3
lSysSec = SysSec()
Print "Liczba sekund dla " ; Time$ ; " w dniu " ; Date$ ; " równa się "
; lSysSec
' Liczba sekund dla 08:35:17 w dniu 03.04.16 równa się 103797317

' Przykład 2 z zdefiniowanym ciągiem 6 bajtów (Sekundy / Minuty /
Godziny / Dzień / Miesiąc / Rok)
bSec = 20 : bMin = 1 : bHour = 7 : bDay = 22 : bMonth = 12 : bYear = 1
lSysSec = SysSec(bSec)
strTime = Time(bSec) : strDate = Date(bDay)
Print "Liczba sekund dla " ; strTime ; " w dniu " ; strDate ; " równa
się " ; lSysSec
' Liczba sekund dla 07:01:20 w dniu 01.12.22 równa się 62319680

' Przykład 3 z ciągiem zawierającym czas i ciągiem zawierającym datę
strTime = "04:58:37"
strDate = "02.09.18"
lSysSec = SysSec(strTime , strDate)
Print "Dla godziny " ; strTime ; " w dniu " ; strDate ; " zwraca " ;
lSysSec
' Dla godziny 04:58:37 w dniu 02.09.18 zwraca 85640317

' Przykład 4 z liczbą dni od początku wieku
wSysDay = 2000
lSysSec = SecOfDay(wSysDay)
Print "Dla dnia " ; wSysDay ; " o godz. 00:00:00 zwraca " ; lSysSec
' Dla dnia 2000 o godz. 00:00:00 zwraca 172800000

End
```

SYSSECELAPSED() *(Nowość w wersji 1.11.7.3)*

Przeznaczenie:

Zwraca liczbę sekund jaka upłynęła od poprzednio zapamiętanej liczby sekund od początku wieku.

Składnia:

rezultat = **SYSSECELAPSED**(*TimeStamp*)

gdzie:

<i>rezultat</i>	zmienna typu Long gdzie funkcja zwróci wynik,
<i>TimeStamp</i>	zmienna typu Long która przechowuje uprzednio zapamiętaną liczbę sekund jaka upłynęła od początku wieku, (funkcja

SYSSEC)

Opis:

Funkcja ta korzysta ze zmiennych programowego zegara czasu rzeczywistego (patrz **CONFIG CLOCK**): `_sec`, `_min`, `_hour`, `_day`, `_month`, `_year`. Na podstawie tych danych oblicza różnicę pomiędzy czasem aktualnym a zapamiętanym uprzednio.

Funkcja ta jest podobna do funkcji **SECELAPSED**. Różnią się one tylko czasem jaki może upłynąć pomiędzy czasem zapamiętanym i bieżącym (dokładnie maksymalną liczbą sekund).

Uwaga! Funkcja zwraca poprawne wartości tylko w obrębie: od 2000-01-01 do 2068-01-19 godz. 03:14:07. W 2068 przepełnieniu ulega zmienna typu Long.

Zwracana wartość mieści się w granicach 0 do 2147483647.

Zobacz także: **SECELAPSED** , **SYSSEC** , **Biblioteka DATETIME**

Przykład:

```
Enable Interrupts
Config Clock = Soft

Dim lSystemTimeStamp As Long
Dim lSystemSecondsElapsed As Long

lSystemTimeStamp = SecOfDay()
Print "Teraz upłynęło " ; lSystemTimeStamp ; " sekund po 2000-01-01
00:00:00"

' wykonaj cokolwiek
' a kilka chwil później

lSystemSecondsElapsed = SysSecElapsed(lSystemTimeStamp)
Print "Teraz jest już " ; lSystemSecondsElapsed ; " sekund później"
```

TAN() (Nowość w wersji 1.11.6.8)

Przeznaczenie:

Zwraca wartość tangensa kąta podanego w radianach.

Składnia:

zmienna = **TAN**(*liczba*)

gdzie:

<i>zmienna</i>	dowolna zmienna typu single, do której wpisany będzie wynik działania funkcji,
<i>liczba</i>	liczba której wartość tangensa należy obliczyć.

Opis:

Wszystkie funkcje trygonometryczne używają zapisu wartości kątów w radianach. Aby przeliczyć stopnie na radiany należy użyć funkcji DEG2RAD. Zamianę odwrotną zapewnia instrukcja RAD2DEG.

Zobacz także: **RAD2DEG** , **DEG2RAD** , **ATN** , **SIN** , **COS**

Przykład: **Zobacz przykład**

TANH() (Nowość w wersji 1.11.6.8)

Przeznaczenie:

Zwraca wartość tangensa hyperbolicznego kąta podanego w radianach.

Składnia:

zmienna = **TANH**(*liczba*)

gdzie:

<i>zmienna</i>	dowolna zmienna typu single, do której wpisany będzie wynik działania funkcji,
<i>liczba</i>	liczba której wartość tangensa hyperbolicznego należy obliczyć.

Opis:

Wszystkie funkcje trygonometryczne używają zapisu wartości kątów w radianach. Aby przeliczyć stopnie na radiany należy użyć funkcji DEG2RAD. Zamianę odwrotną zapewnia instrukcja RAD2DEG.

Zobacz także: **RAD2DEG** , **DEG2RAD** , **ATN** , **SIN** , **COS** , **TAN** , **SINH** , **COSH**

Przykład: Zobacz temat **Biblioteka FP_TRIG**

THIRDLINE

Przeznaczenie:

Ustawia kursor wyświetlacza LCD w trzeciej linii.

Składnia:

THIRDLINE

Opis:

Instrukcja ustawia kursor wyświetlacza na początek trzeciej linii.

Uwaga! Instrukcja działa tylko na wyświetlaczach posiadających cztery linie.

Zobacz także: **UPPERLINE** , **LOWERLINE** , **FOURTHLINE** , **LOCATE**

Przykład:

```
Dim a As byte
a = 255
Thirdline
Lcd a
Home Upper
Lcd a

End
```

TIME\$

Przeznaczenie:

Specjalna zmienna przechowująca aktualny czas.

Składnia:

TIME\$ = "*hh:mm:ss*"
zmienna = **TIME\$**

Opis:

Zmienna TIME\$ jest używana w połączeniu z instrukcją CONFIG CLOCK.

Instrukcja CONFIG CLOCK używa licznika TIMER0 lub TIMER2, pracującego w trybie asynchronicznym, do generowania przerwań co 1 sekundę. Procedura obsługi tego przerwania zwiększa odpowiednio zawartość zmiennych _sec, _min oraz _hour. Godziny są zliczane w trybie 24 godzinny.

Kiedy zawartość zmiennej TIME\$ zostaje przypisana zmiennej tekstowej, powyższe zmienne „liczników” są najpierw przekazywane zmiennej TIME\$. Gdy wystąpi operacja odwrotna – tj. zostanie zmieniona wartość zmiennej TIME\$, wtedy jej zawartość zostaje rozkodowana i zmieniane są zmienne „liczników”.

Format zapisu czasu jest taki sam jak w dialektach Qbasic/VB. Jediną różnicą w stosunku do QBasic/VisualBasic jest to, że wszystkie liczby muszą być 2 cyfrowe. Jest to podyktowane zmniejszeniem ilości potrzebnego kodu. Można to oczywiście zmienić.

Zobacz także: DATE\$, CONFIG CLOCK

Asembler:

Wywoływane są następujące procedury z biblioteki mcs.lib:

- podczas przypisywania wartości zmiennej TIME\$: _set_time (wywołuje _str2byte)
- podczas odczytywania zmiennej DATE\$: _make_dt (wywołuje _byte2str)

Przykład:

```
'-----  
'                               MEGACLOCK.BAS  
'                               (c) 2000-2001 MCS Electronics  
'-----  
'Ten przykład pokazuje jak używać specjalnych zmiennych TIME$ i DATE$  
'Użycie procesora AT90s8535 (i licznika TIMER2) oraz Mega103 (licznika  
TIMER0)  
'pozwala na łatwe zaimplementowanie zegara czasu rzeczywistego,  
'dołączając zewnętrzny rezonator 32.768KHz do licznika.  
'Potrzebny będzie także pewien fragment kodu.  
  
'Ten przykład jest napisany dla płytki STK300 z procesorem M103  
Enable Interrupts  
  
'[konfiguracja LCD]  
$lcd = &HC000                                'adres dla E i RS  
$lcdrs = &H8000                              'adres tylko dla E  
Config Lcd = 20 * 4                          'fajny wyświetlacz dla wielkiego  
procesora  
Config Lcdbus = 4                            'działamy w trybie BUS z 4  
liniami danych (db4-db7)  
Config Lcdmode = Bus                         'właśnie go ustawiamy  
  
'[teraz inicjalizacja zegara]  
Config Clock = Soft                          'O! Takie to jest proste!  
'Powyższa instrukcja definiuje procedurę obsługi przerwania TIMER0,  
' więc licznik nie może być już używany do innych celów!  
  
'Format daty: MM/DD/YY (miesiąc/dzień/rok)  
Config Date = MDY , Separator = /  
  
'Ustawiamy datę  
Date$ = "11/11/00"
```

```

'Ustawiamy czas, format to HH:MM:SS (24 godzinny)
'Nie można podać 1:2:3!! Może będzie to możliwe w przyszłości.
'Chyba, że zmienisz sobie kod w bibliotece.

Time$ = "02:20:00"

'wyczyść pole LCD
Cls

Do
    Home                                'kursor na początek
    Lcd Date$ ; " " ; Time$            'pokaż czas i datę
Loop

'Procedura zegara używa specjalnych zmiennych:
'_day , _month, _year , _sec, _hour, _min
'Wszystkie są typu Byte. Można je modyfikować bezpośrednio:
_day = 1
'Dla zmiennej _year zapisywane są tylko dwie cyfry oznaczające rok.

End

```

TIME() (Nowość w wersji 1.11.7.3)

Przeznaczenie:

Dokonuje konwersji czasu na postać znakową lub na ciąg 3 bajtów przechowujących sekundy, minuty i godziny.

Składnia:

```

bSMG = TIME( /SecOfDay )
bSMG = TIME( /SysSec )
bSMG = TIME( strCzas )

strCzas = TIME( /SecOfDay )
strCzas = TIME( /SysSec )
strCzas = TIME( bSMG )

```

gdzie:

<i>strCzas</i>	ciąg znaków z zapisanym czasem w formacie „gg:mm:ss”,
<i>/SysSec</i>	zmienna typu Long która przechowuje liczbę sekund jaka upłynęła od początku wieku,
<i>/SecOfDay</i>	zmienna typu Word która przechowuje liczbę sekund jaka upłynęła od północy,
<i>bSMG</i>	nazwa zmiennej typu Byte, w której przechowywany jest bieżąca liczba sekund. Zaraz po tym bajcie powinny znajdować się jeszcze dwa bajty zawierające kolejno liczbę minut i godzin.

Opis:

Funkcja ta w zależności od typu zmiennej do której ma być wpisany jej wynik zwraca rozkodowany czas w postaci liczbowej lub znakowej. Zmiana danych jest przeprowadzana automatycznie.

Konwersja na ciąg znaków:

Ciąg znaków, któremu funkcja ma przypisać datę musi mieć co najmniej 8 znaków. Inaczej bajty następujące po tym ciągu w pamięci zostaną nadpisane.

Konwersja do formatu programowego zegara (CONFIG CLOCK):

Trzy bajty do których funkcja ma zwrócić kolejno liczbę sekund, minut i godzin. Bajty te muszą być umieszczone w pamięci jeden za drugim. Jako parametr funkcji podaje się tylko pierwszy bajt gdzie funkcja umieszcza liczbę sekund dnia.

Zobacz także: Biblioteka DATETIME , SECOFDAY , SYSSEC

Przykład:

```
Enable Interrupts
Config Clock = Soft

Dim strTime As String * 8
Dim bSec As Byte , bMin As Byte , bHour As Byte
Dim lSecOfDay As Long
Dim lSysSec As Long

' Przykład 1: Konwersja 3 bajtów zawierających Sekundy / Minuty /
Godziny na czas w postaci tekstu
bSec = 20: bMin = 1: bHour = 7
strTime = Time(bSec)
Print "Wartości: Sek="; bSec ; " Min="; bMin ; " Godz=" ; bHour ; "
skonwerterowano do " ; strTime
' Wartości: Sek=20 Min=1 Godz=7 skonwerterowano do 07:01:20

' Przykład 2: Konwersja liczby sekund na czas w postaci tekstu
lSysSec = 123456789
strTime = Time(lSysSec)
Print "Liczba " ; lSysSec ; " sekund to " ; strTime
' Liczba 123456789 sekund to 21:33:09

' Przykład 3: Konwersja liczby sekund od początku dnia na czas w
postaci tekstu
lSecOfDay = 12345
strTime = Time(lSecOfDay)
Print "Liczba " ; lSecOfDay ; " sekund od początku dnia to " ; strTime
' Liczba 12345 sekund od początku dnia to 03:25:45

' Przykład 4: Konwersja liczby sekund na 3 bajty zawierające
' składniki Sekundy / Minuty / Godziny
lSysSec = 123456789
bSec = Time(lSysSec)
Print "Liczba " ; lSysSec ; " sekund skonwerterowana na Sek="; bSec ;
" Min="; bMin ; " Godz=" ; bHour
' Liczba 123456789 sekund skonwerterowana na Sek=9 Min=33 Godz=21

' Przykład 5: Konwersja liczby sekund od początku dnia na 3 bajty
' zawierające Sekundy / Minuty / Godziny
lSecOfDay = 12345
bSec = Time(lSecOfDay)
Print "Liczba " ; lSecOfDay ; " sekund od początku dnia
skonwerterowana na Sek="; bSec ; " Min="; bMin ; " Godz=" ; bHour
' Liczba 12345 sekund od początku dnia skonwerterowana na Sec=45
Min=25 Hour=3
```

TOGGLE

Przeznaczenie:

Zamienia na przeciwny stan końcówki portu lub zawartość zmiennej bitowej.

Składnia:

TOGGLE *pin*

gdzie:

pin zmienna typu Bit, lub nazwa końcówki portu.

Opis:

Za pomocą instrukcji TOGGLE można szybko zmienić stan końcówki lub zawartość zmiennej bitowej na przeciwny. Dla przykładu, gdy końcówka portu steruje przekaźnikiem, który aktualnie jest w stanie OFF – wykonanie instrukcji TOGGLE spowoduje przełączenie stanu przekaźnika na ON. Ponowne użycie instrukcji TOGGLE, przełączy stan z powrotem na OFF.

Uwaga! Kończówka portu musi być wcześniej ustawiona jako wyjście.

Zobacz także: **CONFIG PORT**

Przykład:

```
Dim Var As Byte

Config Pinb.0 = Output           'kończówka PortB.0 będzie wyjściem

Toggle Portb.0                  'zmiana stanu
Waitms 1000                     'czekaj 1 sekundę
Toggle Portb.0                  'zmień stan ponownie
```

TRIM()

Przeznaczenie:

Zwraca kopię ciągu z usuniętymi wiodącymi i końcowymi spacjami.

Składnia:

zmienna = TRIM(*tekst*)

gdzie:

<i>zmienna</i>	zmienna tekstowa do której wpisany będzie rezultat działania funkcji,
<i>tekst</i>	stała tekstowa lub zmienna, z której usunięte zostaną wiodące jak i końcowe spacje.

Zobacz także: **RTRIM** , **LTRIM**

Przykład:

```
Dim S As String * 6

S = "  AB  "
Print Ltrim(s)
Print Rtrim(s)
Print Trim(s)

End
```

UCASE()

Przeznaczenie:

Zamienia wszystkie znaki z ciągu na ich odpowiedniki w zestawie dużych liter.

Składnia:

zmienna = UCASE(*ciąg*)

gdzie:

<i>zmienna</i>	zmienna typu String, do której wpisany będzie wynik działania funkcji,
<i>ciąg</i>	ciąg którego znaki należy zamienić.

Opis:

Uwaga! Funkcja nie rozpoznaje polskich znaków diakrytycznych, ani innych znaków narodowych o kodach większych niż 127.

Zobacz także: **LCASE**

Asembler:

Wywoływana jest procedura `_UCASE` z biblioteki `mcs.lib`. Rejestr X wskazuje na ciąg docelowy, rejestr Z wskazuje na ciąg źródłowy.

Jest generowany następujący kod (może być inny w zależności od kontrolera):

```
;##### Z = Ucase(s)
Ldi R30,$60
Ldi R31,$00          ;załadowanie stałej do rejestru
Ldi R26,$6D
Rcall _Ucase
```

Przykład:

```
Dim S As String * 12 , Z As String * 12

S = "Witaj świecie"
Z = Lcase(s)
Print Z
Z = Ucase(s)
Print Z

End
```

UPPERLINE

Przeznaczenie:

Ustawia kursor wyświetlacza LCD w pierwszej linii.

Składnia:

UPPERLINE

Zobacz także: **LOWERLINE** , **THIRDLINE** , **FOURTHLINE**

Przykład:

```
Lowerline
Lcd "Under"
Upperline
Lcd "Upper"
```

VAL()

Przeznaczenie:

Zamienia tekstową reprezentację liczby na jej postać dziesiętną.

Składnia:

zmienna = **VAL**(*tekst*)

gdzie:

<i>zmienna</i>	zmienna numeryczna do której wpisany będzie wynik działania funkcji,
<i>tekst</i>	zmienna tekstowa, zawierająca liczbę w postaci tekstu.

Opis:

Funkcja ta jest odwrotnością funkcji STR.

Zobacz także: STR , HEXVAL , HEX , BIN

Przykład:

```
Dim a As Byte , s As String * 10

s = "123"
a = Val(s)           'konwersja ciągu
Print a

End
```

VARPTR()

Przeznaczenie:

Zwraca adres pod jakim umieszczono podaną zmienną.

Składnia:

zmienna = VARPTR(*zmienna2*)

gdzie:

<i>zmienna</i>	zmienna typu Integer lub Word do której wpisany będzie adres <i>zmiennej2</i> ,
<i>zmienna2</i>	dowolna zmienna, której adres należy obliczyć.

Zobacz także: LOADADR

Przykład:

```
Dim B As XRAM Byte At &H300 , I As Integer , W As Word

W = Varptr(b)
Print Hex(w)           'wydrukuje &H0300

End
```

WAIT

Przeznaczenie:

Przerywa działanie programu na określony czas.

Składnia:

WAIT *il_sekund*

gdzie:

<i>il_sekund</i>	liczba określająca czas opóźnienie w sekundach.
------------------	---

Opis:

Instrukcja wstrzymuje działanie programu na podaną ilość sekund. Odmierzany czas jest

wartością przybliżoną, więc nie należy stosować tej instrukcji do dokładnego odmierzenia czasu. Używanie przerw, może znacznie wydłużyć działanie instrukcji.

Zobacz także: **DELAY** , **WAITMS**

Przykład:

```
Wait 3          'wstrzymanie działania programu na około 3 sekundy
Print "*"      
```

WAITKEY()

Przeznaczenie:

Funkcja wstrzymuje działanie programu do czasu pojawienia się w buforze transmisji szeregowej odebranego znaku.

Składnia:

```
zmienna = WAITKEY()
```

```
zmienna = WAITKEY( #nr_kanału )
```

gdzie:

<i>zmienna</i>	zmienna tekstowa lub numeryczna, do której wpisany będzie kod ASCII odebranego znaku,
<i>nr_kanału</i>	numer otwartego wcześniej kanału programowego lub sprzętowego układu UART.

Opis:

Funkcja działa podobnie do funkcji INKEY. Różnica polega na tym, że INKEY nie czeka na pojawienie się znaku w buforze transmisji szeregowej i w przypadku jego braku zwraca ciąg pusty (zero). WAITKEY natomiast będzie oczekiwać tak długo, aż znak się pojawi.

Można przetestować czy jest dostępny jakikolwiek znak w buforze stosując funkcję ISCHARWAITING.

Zobacz także: **INKEY** , **ISCHARWAITING**

Przykład:

```
Dim A As Byte

A = Waitkey()      'czekamy na znak
Print A            
```

WAITMS

Przeznaczenie:

Przerywa działanie programu na określony czas.

Składnia:

```
WAITMS czas
```

gdzie:

<i>czas</i>	liczba określająca czas opóźnienie w milisekundach, maksymalnie 255.
-------------	--

Opis:

Instrukcja wstrzymuje działanie programu na podaną ilość milisekund. Odmierzany czas jest

wartością przybliżoną, więc nie należy stosować tej instrukcji do jego dokładnego odmierzenia. Używanie przerw, może znacznie wydłużyć działanie instrukcji.

Instrukcję przewidziano jako generowanie opóźnień przy transmisji magistralą I2C. Dla przykładu: układ pamięci szeregowej potrzebuje 10ms by zapisać dane. Instrukcja WAITMS pozwala na łatwe odmierzenie tego czasu.

Zobacz także: [DELAY](#) , [WAIT](#) , [WAITUS](#)

Asembler:

Instrukcja WAITMS wywołuje procedurę `_waitms`. Do rejestrów R24 oraz R25 załadowana jest liczba podanych milisekund. Instrukcja używa i zapamiętuje zawartość R30 oraz R31.

W zależności od zastosowanego kwarcu, kod w języku asembler wygląda tak:

```
_WaitMS:
_WaitMS1F:
    Push R30                      ;zapamiętaj Z
    Push R31
_WaitMS_1:
    Ldi R30,$E8                   ;opóźnienie o 1 ms
    Ldi R31,$03
_WaitMS_2:
    Sbiw R30,1                    ;-1
    Brne _WaitMS_2                ;dopóki odliczysz 1 ms
    Sbiw R24,1
    Brne _WaitMS_1                ;tyle razy ile podano mS
    Pop R31
    Pop R30
    Ret
```

Przykład:

```
Waitms 10      'wstrzymanie działania programu na około 10 milisekund
Print " *"
```

WAITUS

Przeznaczenie:

Przerywa działanie programu na określony czas.

Składnia:

WAITUS czas

gdzie:

czas

liczba określająca czas opóźnienie w mikrosekundach,
maksymalnie 65535.

Opis:

Instrukcja wstrzymuje działanie programu na podaną ilość mikrosekund. Odmierzany czas jest wartością przybliżoną, więc nie należy stosować tej instrukcji do jego dokładnego odmierzenia. Używanie przerw, może znacznie wydłużyć działanie instrukcji.

Minimalny czas możliwy do odmierzenia jest zależny od użytej częstotliwości kwarcu. Liczba cykli potrzebnych na ustawienie i zapamiętanie rejestrów wynosi 17. Jeśli pętla jest ustawiona na 1, minimalny czas wynosi 21μs. W tym wypadku lepiej użyć instrukcji NOP, która „odmierzy” 1 cykl zegarowy.

Przy częstotliwości 4MHz minimalne opóźnienie to 5 μs, więc instrukcja **Waitus** 3 odmierzy właśnie 5 μs. Poniżej tych wartości, czas opóźnienia nie będzie dokładny.

Kiedy naprawdę wymagany jest dokładny czas opóźnienia, najlepiej użyć do tego celu któregoś z czasomierzy. Należy wtedy ustawić odpowiednią wartość w rejestrze licznika, włączyć zliczanie i poczekać aż zostanie ustawiona flaga przepełnienia licznika. Wadą tego rozwiązania, jest niemożliwość używania tego licznika do innych celów, podczas odmierzania czasu.

Filozofia języka BASCOM BASIC zakłada, że tam gdzie tylko to możliwe, nie są używane zasoby sprzętowe procesora, jeśli tylko te same zadania można wykonać programowo.

Zobacz także: **DELAY** , **WAIT** , **WAITMS**

Przykład:

```
Waitus 10          'wstrzymanie działania programu na około 10
mikrosekund
Print "*"          'wyświetlenie gwiazdki
```

WHILE...WEND

Przeznaczenie:

Wykonuje określony ciąg instrukcji dopóki warunek jest spełniony.

Składnia:

```
WHILE warunek
    ciąg_instrukcji
WEND
```

gdzie:

warunek	wyrażenie obliczane jako warunek pętli,
ciąg_instrukcji	dowolny ciąg instrukcji języka BASCOM BASIC.

Opis:

Instrukcja służy do konstruowania pętli programowych. Gdy podany warunek jest prawdziwy (zwraca wartość logicznej prawdy) wtedy wykonywany będzie ciąg instrukcji umieszczony pomiędzy WHILE a WEND.

Wyjście z pętli jest możliwe tylko gdy warunek nie będzie spełniony (zwróci wartość logicznego fałszu), lub też za pomocą instrukcji **EXIT WHILE**

Jest to instrukcja podobna do instrukcji DO..LOOP, z tą różnicą, że warunek jest obliczany przed wejściem do pętli. Gdy warunek nie będzie spełniony ciąg instrukcji wewnątrz pętli nie będzie wykonany.

Zobacz także: **DO..LOOP**

Przykład:

```
Dim a As Byte

a = 0

While a <= 10      'jeśli a jest mniejsze lub równe 10 to
    Print a        'drukujemy wartość a
    Incr a         'i zwiększamy o jeden
Wend
```

WRITEEEPROM

Przeznaczenie:

Zapisuje dane do wbudowanej pamięci EEPROM.

Składnia:

WRITEEEPROM *zmienna* , *adres*

gdzie:

<i>zmienna</i>	zmienna, której wartość wpisana będzie do adresowanej komórki EEPROM,
<i>adres</i>	adres komórki pamięci EEPROM.

Opis:

Procesory serii AVR mają wbudowaną pamięć EEPROM, której można używać z poziomu języka BASCOM BASIC do przechowywania zmiennych lub dowolnych danych.

Na przykład można umieścić dowolną zmienną – oprócz typu Bit – we wbudowanej pamięci EEPROM:

```
Dim V As Eram Byte      'zmienna V będzie umieszczona w EEPROM
Dim B As Byte           'zmienna będzie w pamięci SRAM

B = 10
V = B                   'można normalnie korzystać z takich zmiennych
B = V                   'na przykład odczytać zapisana wcześniej daną
```

Uwaga! Bardzo ważna jest zgodność typów.

Można także używać zmiennych tablicowych:

```
Dim ar(10) As Eram Byte
```

Uwaga! Opierając się na nocie katalogowej firmy Atmel, pierwsza komórka pamięci EEPROM (o adresie 0) może zostać nadpisana podczas operacji zerowania mikroprocesora. Zaleca się nie używać tej komórki pamięci dla ważnych danych, szczególnie przechowywanych po zaniku zasilania.

Uwaga! Jako zabezpieczenie, wartość zapisana w rejestrze R23 jest ustawiana na „magiczną”(?) przed zapisem danych do pamięci EEPROM

Instrukcje tą wprowadzono ze względu na kompatybilność z kompilatorem BASCOM-8051.

Zobacz także: **READEEPROM**

Przykład:

```
Dim B As Byte

WriteEeprom B , 0      'zapis do pamięci EEPROM
ReadEeprom B , 0       'odczytanie wartości
```

Przykład2:

```
'-----
'                                     EEPROM2.BAS
'przykład ten pokazuje jak używać nowej wersji instrukcji READEEPROM
'-----
'Najpierw określimy zmienne
Dim B As Byte
Dim Yes As String * 1

'Składnia Readeeprom oraz Writeeprom:
'Readeeprom var, address
```

```

'Nowością jest wprowadzenie obsługi odczytywania danych umieszczonych
w liniach DATA
'Ponieważ dane te są umieszczone w osobnym pliku a nie w pamięci kodu,
'należy się przełączyć na pamięć EEPROM i umieścić je na początku
programu
'Taka jest różnica między normalnymi liniami DATA, które muszą być
'umieszczone poza wykonywalną częścią programu!!

'przełączamy się zatem na pamięć EEPROM
$eeprom
'określamy etykiety (adresy)
Label1:
    Data 1 , 2 , 3 , 4 , 5
Label2:
    Data 10 , 20 , 30 , 40 , 50

'przełączamy się na pamięć kodu
$data
'Cały powyższy fragment w istocie nie generuje kodu.
'Tworzy on tylko plik z danymi umieszczonymi w pliku .EEP

'Używamy nowej składni instrukcji
Readeeprom B , Label1
Print B                                'wydrukuj 1
'Następna instrukcja spowoduje odczyt danych spod następnego adresu
'Lecz pierwsza instrukcja musi określić skąd odczytać dane - adres
startowy
Readeeprom B
Print B                                'wydrukuj 2

Readeeprom B , Label2
Print B                                'wydrukuj 10
Readeeprom B
Print B                                'wydrukuj 20

'Działa to także podczas zapisu danych:
'zatrzymamy się jednak na chwilę
Input "Gotowy?" , Yes
B = 100
Writeeeprom B , Label1
B = 101
Writeeeprom B

'odczytujemy je z powrotem
Readeeprom B , Label2
Print B                                'wydrukuj 1
'Następna instrukcja spowoduje odczyt danych spod następnego adresu
'Lecz pierwsza instrukcja musi określić skąd odczytać dane - adres
startowy
Readeeprom B
Print B                                'wydrukuj 2

End

```

X10DETECT (Nowość w wersji 1.11.7.3)

Przeznaczenie:

Składnia:

gdzie:

Opis:

Zobacz także: **CONFIG X10** , **X10SEND**

X10SEND (Nowość w wersji 1.11.7.3)

Przeznaczenie:

Wysyła kod domu i kod rozkazu za pomocą protokołu X10.

Składnia:

X10SEND *kod_domu* , *kod_rozkazu*

gdzie:

kod_domu kod domu to jedna litera z zakresu od **A** do **P**. Można użyć stałej lub zmiennej zawierającej jedną literę.
kod_rozkazu kod funkcji jaką ma wykonać urządzenie końcowe. Jest to liczba z zakresu 1-32.

Opis:

Instrukcja X10SEND wymaga modułu interfejsu TW-523. Transmisja pomiędzy procesorem a modulem odbywa się za pomocą 3 linii: masy, **TX** oraz **Zero Cross**. Końcówki procesora określa się instrukcją **CONFIG X10**.

X10 to popularny protokół transmisji danych przez domową sieć energetyczną. Wykorzystuje on modulowany sygnał 110 KHz, nakładany na przebieg domowej sieci energetycznej 50/60 Hz, 220/110 V.

Uwaga! Eksperymentowanie z siecią 110V-240V może być bardzo niebezpieczne!!!

Gdy moduł TW-523 jest podłączony do sieci, wykonanie instrukcji X10SEND powinno spowodować że zamrugają kontrolki LED.

W poniższej tabeli umieszczono dostępne kody:

Kod Funkcji	Opis
1-16	Używane do adresowania urządzeń końcowych. Protokół X10 pozwala na korzystanie z 16 urządzeń przypisanych jednemu kodowi domu.
17	Wyłącz wszystkie urządzenia.
18	Włącz wszystkie światła.
19	WŁĄCZ
20	WYŁĄCZ
21	CIEMIEJ
22	JAŚNIEJ
23	Wyłącz wszystkie światła
24	Kod rozszerzający
25	Żądanie powitania
26	Potwierdzenie powitania
27	Predefiniowane ściemnienie
28	Predefiniowane ściemnienie
29	Rozszerzone dane analogowe
30	Włącz status
31	Wyłącz status

Urządzenia X10 są bardzo popularne i rozpowszechnione w Stanach Zjednoczonych. W Europie trudno jest znaleźć moduł TW-523 przystosowany do napięcia 220/230/240 V.

Osobiście zmodyfikowałem moduł 110V tak by pracował przy napięciu 220V. Informacje te można znaleźć w Internecie. Pamiętaj jednak aby **MODYFIKACJA BYŁA PRZEPROWADZONA TYLKO GDY WIESZ CO ROBISZ**. Zmiany mogą doprowadzić do ryzyka powstania spięć czy nawet pożaru! Zmodyfikowane urządzenie może nie przejść testów zgodności z normami **CE** lub innych wymaganych przez energetykę.

Pod adresem www.x10.com można znaleźć wszystkie informacje na temat urządzeń X10. Z powodu objętości niemożliwe jest umieszczenie tych informacji w pomocy.

Zobacz także: **CONFIG X10** , **X10DETECT**

Wstawki assemblerowe

Język BASCOM BASIC pozwala na wpłatanie w treść programu bloków kodu napisanych w języku assembler. Jest to użyteczne w tych sytuacjach gdy wymagana jest pełna kontrola nad generowanym kodem.

Łączenie kodu assemblera z instrukcjami BASCOM BASIC.

Prawie wszystkie mnemoniki są rozpoznawane przez kompilator automatycznie. Listę tych mnemoników możesz zobaczyć wybierając temat [Lista rozkazów procesorów AVR](#). Wyjątkiem są mnemoniki: **SUB**, **SWAP** oraz **OUT**, gdyż w języku BASCOM BASIC istnieją jako **słowa zastrzeżone**. Aby kompilator wiedział, że należy interpretować je jako mnemoniki należy umieścić przed nimi znak ! (wykrzyknik).

Dla przykładu:

```
Dim a As Byte At &H60 'zmienna A znajduje się pod adresem &H60

Ldi R27 , $00          ;w R27 zapisujemy część starszą adresu (MSB)
Ldi R26 , $60          ;w R26 zaś młodszą część (LSB)
Ld R1, X               ;teraz ładujemy daną spod adresu $60 do R1
!Swap R1               ;zamieniamy połówki bajtów
```

W tym przykładzie wykorzystano instrukcję SWAP poprzedzoną znakiem !.

Można także umieścić fragment w języku assemblera stosując dyrektywę \$ASM:

```
$ASM
Ldi R27 , $00          ;w R27 zapisujemy część starszą adresu (MSB)
Ldi R26 , $60          ;w R26 zaś młodszą część (LSB)
Ld R1, X               ;teraz ładujemy daną spod adresu $60 do R1
Swap R1                ;zamieniamy połówki bajtów
$END ASM
```

By ułatwić wpisywanie adresu do pary rejestrów oznaczonej jako X czy Z, można użyć specjalnej instrukcji języka BASCOM BASIC. Instrukcja ta nie może być użyta do wpisywania adresu do rejestru Y, gdyż jest on używany jako programowy wskaźnik stosu.

```
Dim A As Byte          'rezerwujemy bajt w pamięci
Loadadr a, X            'ładujemy adres zmiennej A do pary rejestrów X
```

Instrukcję LOADADR można było zastąpić dwoma instrukcjami przesłania:

```
Ldi R26, $60            'adres przykładowy!
Ldi R27, $00
```

Rejestry procesora używane przez instrukcje języka BASCOM BASIC.

Rejestry R4 oraz R5 są używane jako wskaźnik stosu dla przechowywania tymczasowych zmiennych.

Rejestr R6 zawiera kilka specjalnych zmiennych bitowych:

- R6 bit 0 = flaga konwersji Integer - Word,
- R6 bit 1 = tymczasowy bit używany przy zamianie bitów,
- R6 bit 2 = wskaźnik błędu (zmienna ERR)
- R6 bit 3 = flaga stanu echa dla instrukcji INPUT

Para R8 i R9 używana jest jako wskaźnik dla instrukcji READ.

Reszta rejestrów jest używana w zależności od treści instrukcji.

Dostęp do zmiennych języka BASCOM BASIC z poziomu asemblera.

Istnieje możliwość używania zmiennych zdefiniowanych w programie przez kod asemblera. By załadować adres zmiennej należy jej nazwę umieścić w nawiasach klamrowych { }. Jak w poniższym przykładzie:

```
Dim A As Byte, B As Bit

Lds R16, {A}      '{A} zostanie zastąpione adresem zmiennej A w pamięci
```

By mieć dostęp do zmiennych bitowych, ich nazwę należy poprzedzić kwalifikatorem BIT.

```
Sbrs R16, BIT.B      'uwaga na kropkę!
```

Zmienne bitowe są umieszczane w bajtach, a kolejność ich jest taka, że pierwszy z nich zajmuje najbardziej znaczący bit. Następne zajmują kolejno następne bity w kierunku najmniej znaczącego bitu.

Aby załadować adres etykiety należy użyć takiej konstrukcji:

```
Ldi ZL, Low(lbl * 2)
Ldi ZH, High(lbl * 2)
```

gdzie:

ZL to rejestr R30 a ZH rejestr R31 które tworzą razem rejestr wskaźnikowy Z.
Analogicznie R26 i R27 tworzą rejestr X, a para R28 i R29 rejestr Y.

Ponieważ procesory AVR korzystają z 16bitowego modelu pamięci kodu (każda instrukcja zajmuje dwa bajty) należy użyć * 2. LBL to nazwa przykładowej etykiety.

Niektóre sprawy szczególnego znaczenia:

- Wszystkie instrukcje których parametrem jest stała liczbowa, działają tylko w połączeniu z górną połówką rejestrów (R16 – R31). Tak więc instrukcja `LDI R15,12` NIE BĘDZIE DZIAŁAĆ!
- W instrukcji `SBR register, K`, stała K może zawierać się w zakresie 0-255. Można zatem ustawiać kilka bitów naraz!
- W instrukcji `SBI port, K`, stała K może mieć zakres 0-7 co oznacza, że tylko JEDEN bit może być ustawiany w rejestrach portów.
- Dwie ostatnie uwagi tak samo dotyczą instrukcji CBR i CBI.

Tworzenie własnych bibliotek procedur i funkcji.

Opisy zawarte tutaj bazują na przykładach umieszczonych w plikach: `libdemo.bas` umieszczonego w katalogu `SAMPLES` oraz `mylib.lib` znajdującego się w katalogu `LIB`.

Procedury.

Na początku trzeba określić jakie parametry są przekazywane w odwołaniu do procedury (lub funkcji) z biblioteki. Należy także określić jak są one przekazywane, chodzi tu o argumenty **BYVAL** i **BYREF**.

Dla przykładu, procedura `test` posiada dwa parametry:

- × do którego odwołanie jest przez wartość (tworzona jest tymczasowa zmienna),
- y do którego odwołanie jest przez adres.

W obu przypadkach adres zmiennej (parametru) jest umieszczany na stosie, którego wskaźnik

znajduje się w rejestrze Y (para R28 i R29).

Adresy parametrów są umieszczane w odwrotnej kolejności – najpierw *y* potem *x*. By więc uzyskać dostęp do adresu pierwszego parametru, można wykonać to w ten sposób:

```
Ldd R26 , Y + 2
Ldd R27 , Y + 3
```

Spowoduje to umieszczenie adresu pierwszego parametru we wskaźniku X (para R26 i R27).

Adres drugiego parametru także jest umieszczony w na stosie i zajmuje kolejne dwa bajty. Należy pamiętać, że stos rozrasta się w dół, więc by otrzymać adres drugiego parametru *y*, można to zrobić tak:

```
Ldd R26 , Y + 0
Ldd R27 , Y + 1
```

Mając te informacje można utworzyć procedurę `test`. Należy pamiętać by nazwę procedury umieścić w nawiasach kwadratowych `[]`, oraz zakończyć przez `[end]`.

```
[test]
test:
    ldd r26,Y+2      ;załaduj adres parametru x
    ldd r27,Y+3
    ld r24,X         ;jego wartość do rejestru r24
    inc r24          ;dodaj 1

    st X,r24         ;zwróć z powrotem
    ldd r26,Y+0      ;adres parametru y
    ldd r27,Y+1
    st X,r24         ;zapisz
    ret             ;i koniec
[end]
```

Funkcje.

Definiowanie funkcji jest analogiczne. Różnicą jest tylko to, że funkcja zwraca zawsze wynik swojego działania, więc jest wykorzystywana dodatkowy parametr.

Parametr ten jest generowany automatycznie i posiada nazwę taką jak nazwa funkcji. Na przykład:

```
Declare Function Test (Byval x As Byte , y As Byte) As Byte
```

Spowoduje utworzenie wirtualnej zmiennej `test`, gdzie umieszczony będzie wynik działania funkcji.

Adres zwracanej zmiennej także jest umieszczony na stosie. Dla funkcji nie posiadającej parametrów adres tej zmiennej zajmuje bajty: *y* + 0 oraz *y* + 1 (*y* jest wskaźnikiem stosu).

Dla przykładowej funkcji `test` rozkład parametrów na stosie jest następujący:

Rezultat (zmienna `test`)

y + 4
y + 5

Parametr *x*

y + 2
y + 3

Parametr *y*

y + 0

```
Y + 1
```

Kiedy wymagane jest wyjście z procedury lub funkcji (odpowiednik EXIT SUB lub EXIT FUNCTION), tworzona jest specjalna etykieta rozpoczynająca się przez `sub_` i zakończona nazwą procedury lub funkcji. W przypadku procedury `test` ma ona postać:

```
sub_test:
```

Zmienne lokalne.

Gdy w treści funkcji lub procedury użyte zostały zmienne lokalne, ich adresy także są umieszczone na stosie. Adresy te są umieszczone przed adresami parametrów.

Dla przykładu gdy procedura korzysta z jednej zmiennej lokalnej, jej adres znajduje się na wierzchołku stosu:

```
Y + 0  
Y + 1
```

I dlatego adres pierwszego z parametrów lub w przypadku funkcji zmiennej zawierającej rezultat, znajdują się pod:

```
Y + 2  
Y + 3
```

Im więcej zmiennych lokalnych, tym niżej (o 2) umieszczone są adresy parametrów.

Kompilacja biblioteki.

Wersje źródłowe bibliotek muszą być skompilowane za pomocą Menedżera Bibliotek ([Library manager](#)) do formatu pliku .LBX. Deklaracje procedur lub funkcji muszą być umieszczone w programie głównym, by kompilator BASCOM poprawnie umieścił parametry na stosie.

Tutaj znajduje się przykład pochodzący z pliku `libdemo.bas`:

```
'definiujemy użytą bibliotekę  
$lib "mylib.lib"  
  
'definiujemy także używane procedury  
$external Test  
  
'deklaracja jest wymagana by parametry umieszczone były na stosie  
Declare Sub Test (Byval X As Byte , Y As Byte)  
  
'jakaś zmienna  
Dim Z As Byte  
  
'wywołujemy własną procedurę  
Call Test (1 , Z)  
  
'Z zaiwierać będzie 2 w tym przykładzie  
End
```

Rozdział ten nie jest próbą nauczania programowania w języku asembler. Jeśli coś zostało pominięte w powyższym opisie, proszę o e-mail na adres: mark@mcselec.com.

Od tłumacza

Proszę nie kierować pytań do mojej osoby, gdyż nie będę w stanie udzielić tych informacji.

Lista rozkazów procesorów AVR

BASCOM BASIC rozpoznaje tylko te mnemoniki (rozkazy), które zdefiniowane zostały oficjalnie przez firmę Atmel.

Kompletne zestawienie mnemoników rozpoznawanych przez BASCOM BASIC, znajduje się w poniższej tabeli. Dalszych informacji na ten temat należy szukać w publikacjach „AVR Data Book” dostępnych na witrynie internetowej firmy Atmel: <http://www.atmel.com/>.

Mnemonic	Operand	Opis działania	Operacja	Ustawiane flagi	Taktów
OPERACJE ARYTMETYCZNE I LOGICZNE					
ADD	Rd, Rr	Add without Carry	$Rd = Rd + Rr$	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry	$Rd = Rd + Rr + C$	Z,C,N,V,H	1
SUB	Rd, Rr	Subtract without Carry	$Rd = Rd - Rr$	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Immediate	$Rd = Rd - K$	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry	$Rd = Rd - Rr - C$	Z,C,N,V,H	1
SBCI	Rd, K	Subtract Immediate with Carry	$Rd = Rd - K - C$	Z,C,N,V,H	1
AND	Rd, Rr	Logical AND	$Rd = Rd \wedge Rr$	Z,N,V	1
ANDI	Rd, K	Logical AND with Immediate	$Rd = Rd \wedge K$	Z,N,V	1
OR	Rd, Rr	Logical OR	$Rd = Rd \vee Rr$	Z,N,V	1
ORI	Rd, K	Logical OR with Immediate	$Rd = Rd \vee K$	Z,N,V	1
EOR	Rd, Rr	Exclusive OR	$Rd = Rd (+) Rr$	Z,N,V	1
COM	Rd	Ones Complement	$Rd = \$FF - Rd$	Z,C,N,V	1
NEG	Rd	Twos Complement	$Rd = \$00 - Rd$	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	$Rd = Rd \vee K$	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	$Rd = Rd \cdot (\$FFh - K)$	Z,N,V	1
INC	Rd	Increment	$Rd = Rd + 1$	Z,N,V	1
DEC	Rd	Decrement	$Rd = Rd - 1$	Z,N,V	1
TST	Rd	Test for Zero or Minus	$Rd = Rd \cdot Rd$	Z,N,V	1
CLR	Rd	Clear Register	$Rd = Rd (+) Rd$	Z,N,V	1
SER	Rd	Set Register	$Rd = \$FF$	Brak	1
ADIW	RdI, K	Add Immediate to Word	$Rdh:Rdl = Rdh:Rdl + K$	Brak	1
SBIW	RdI, K	Subtract Immediate from Word	$Rdh:Rdl = Rdh:Rdl - K$	Brak	1
MUL	Rd,Rr	Multiply Unsigned	$R1, R0 = Rd * Rr$	C	2 *
INSTRUKCJE SKOKÓW					
RJMP	k	Relative Jump	$PC = PC + k + 1$	Brak	2
IJMP		Indirect Jump to (Z)	$PC = Z$	Brak	2
JMP	k	Jump	$PC = k$	Brak	3
RCALL	k	Relative Call Subroutine	$PC = PC + k + 1$	Brak	3
ICALL		Indirect Call to (Z)	$PC = Z$	Brak	3
CALL	k	Call Subroutine	$PC = k$	Brak	4
RET		Subroutine Return	$PC = STACK$	Brak	4
RETI		Interrupt Return	$PC = STACK$	I	4
CPSE	Rd,Rr	Compare, Skip if Equal	if $(Rd = Rr)$ $PC = PC + 2$ or 3	Brak	1 / 2
CP	Rd,Rr	Compare	$Rd - Rr$	Z,C,N,V,H	1
CPC	Rd,Rr	Compare with Carry	$Rd - Rr - C$	Z,C,N,V,H	1
CPI	Rd,K	Compare with Immediate	$Rd - K$	Z,C,N,V,H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	If $(Rr(b)=0)$ $PC = PC + 2$ or 3	Brak	1 / 2
SBR	Rr, b	Skip if Bit in Register Set	If $(Rr(b)=1)$ $PC = PC + 2$ or 3	Brak	1 / 2
SBIC	P, b	Skip if Bit in I/O Register Cleared	If $(I/O(P,b)=0)$ $PC = PC + 2$ or 3	Brak	2 / 3

Mnemonik	Operand	Opis działania	Operacja	Ustawiane flagi	Taktów
SBIS	P, b	Skip if Bit in I/O Register Set	If(I/O(P,b)=1) PC = PC + 2 or 3	Brak	2 / 3
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC=PC+k + 1	Brak	1 / 2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC=PC+k + 1	Brak	1 / 2
BREQ	k	Branch if Equal	if (Z = 1) then PC = PC + k + 1	Brak	1 / 2
BRNE	k	Branch if Not Equal	if (Z = 0) then PC = PC + k + 1	Brak	1 / 2
BRCS	k	Branch if Carry Set	if (C = 1) then PC = PC + k + 1	Brak	1 / 2
BRCC	k	Branch if Carry Cleared	if (C = 0) then PC = PC + k + 1	Brak	1 / 2
BRSH	k	Branch if Same or Higher	if (C = 0) then PC = PC + k + 1	Brak	1 / 2
BRLO	k	Branch if Lower	if (C = 1) then PC = PC + k + 1	Brak	1 / 2
BRMI	k	Branch if Minus	if (N = 1) then PC = PC + k + 1	Brak	1 / 2
BRPL	k	Branch if Plus	if (N = 0) then PC = PC + k + 1	Brak	1 / 2
BRGE	k	Branch if Greater or Equal, Signed	if (N V= 0) then PC = PC+ k + 1	Brak	1 / 2
BRLT	k	Branch if Less Than, Signed	if (N V= 1) then PC = PC + k + 1	Brak	1 / 2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then PC = PC + k + 1	Brak	1 / 2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC = PC + k + 1	Brak	1 / 2
BRTS	k	Branch if T Flag Set	if (T = 1) then PC = PC + k + 1	Brak	1 / 2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC = PC + k + 1	Brak	1 / 2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC = PC + k + 1	Brak	1 / 2
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC = PC + k + 1	Brak	1 / 2
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC = PC + k + 1	Brak	1 / 2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC = PC + k + 1	Brak	1 / 2
INSTRUKCJE PRZESYŁANIA DANYCH					
MOV	Rd, Rr	Copy Register	Rd = Rr	Brak	1
LDI	Rd, K	Load Immediate	Rd = K	Brak	1
LDS	Rd, k	Load Direct	Rd = (k)	Brak	3
LD	Rd, X	Load Indirect	Rd = (X)	Brak	2
LD	Rd, X+	Load Indirect and Post-Increment	Rd = (X), X = X + 1	Brak	2
LD	Rd, -X	Load Indirect and Pre-Decrement	X = X - 1, Rd =(X)	Brak	2
LD	Rd, Y	Load Indirect	Rd = (Y)	Brak	2
LD	Rd, Y+	Load Indirect and Post-Increment	Rd = (Y), Y = Y + 1	Brak	2
LD	Rd, -Y	Load Indirect and Pre-Decrement	Y = Y - 1, Rd = (Y)	Brak	2
LDD	Rd,Y+q	Load Indirect with Displacement	Rd = (Y + q)	Brak	2
LD	Rd, Z	Load Indirect	Rd = (Z)	Brak	2
LD	Rd, Z+	Load Indirect and Post-Increment	Rd = (Z), Z = Z+1	Brak	2
LD	Rd, -Z	Load Indirect and Pre-Decrement	Z = Z - 1, Rd = (Z)	Brak	2

Mnemonik	Operand	Opis działania	Operacja	Ustawiane flagi	Taktów
LDD	Rd, Z+q	Load Indirect with Displacement	$Rd = (Z + q)$	Brak	2
STS	k, Rr	Store Direct	$(k) = Rr$	Brak	3
ST	X, Rr	Store Indirect	$(X) = Rr$	Brak	2
ST	X+, Rr	Store Indirect and Post-Increment	$(X) = Rr, X = X + 1$	Brak	2
ST	-X, Rr	Store Indirect and Pre-Decrement	$X = X - 1, (X) = Rr$	Brak	2
ST	Y, Rr	Store Indirect	$(Y) = Rr$	Brak	2
ST	Y+, Rr	Store Indirect and Post-Increment	$(Y) = Rr, Y = Y + 1$	Brak	2
ST	-Y, Rr	Store Indirect and Pre-Decrement	$Y = Y - 1, (Y) = Rr$	Brak	2
STD	Y+q,Rr	Store Indirect with Displacement	$(Y + q) = Rr$	Brak	2
ST	Z, Rr	Store Indirect	$(Z) = Rr$	Brak	2
ST	Z+, Rr	Store Indirect and Post-Increment	$(Z) = Rr, Z = Z + 1$	Brak	2
ST	-Z, Rr	Store Indirect and Pre-Decrement	$Z = Z - 1, (Z) = Rr$	Brak	2
STD	Z+q,Rr	Store Indirect with Displacement	$(Z + q) = Rr$	Brak	2
LPM		Load Program Memory	$R0 = (Z)$	Brak	3
IN	Rd, P	In Port	$Rd = P$	Brak	1
OUT	P, Rr	Out Port	$P = Rr$	Brak	1
PUSH	Rr	Push Register on Stack	$STACK = Rr$	Brak	2
POP	Rd	Pop Register from Stack	$Rd = STACK$	Brak	2
OPERACJE NA BITACH					
LSL	Rd	Logical Shift Left	$Rd(n+1) = Rd(n), Rd(0) = 0, C = Rd(7)$	Z,C,N,V,H	1
LSR	Rd	Logical Shift Right	$Rd(n) = Rd(n+1), Rd(7) = 0, C = Rd(0)$	Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry	$Rd(0) = C, Rd(n+1) = Rd(n), C = Rd(7)$	Z,C,N,V,H	1
ROR	Rd	Rotate Right Through Carry	$Rd(7) = C, Rd(n) = Rd(n+1), C = Rd(0)$	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	$Rd(n) = Rd(n+1), n=0..6$	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	$Rd(3..0) \leftrightarrow Rd(7..4)$	Brak	1
BSET	s	Flag Set	$SREG(s) = 1$	SREG(s)	1
BCLR	s	Flag Clear	$SREG(s) = 0$	SREG(s)	1
SBI	P, b	Set Bit in I/O Register	$I/O(P, b) = 1$	Brak	2
CBI	P, b	Clear Bit in I/O Register	$I/O(P, b) = 0$	Brak	2
BST	Rr, b	Bit Store from Register to T	$T = Rr(b)$	T	1
BLD	Rd, b	Bit load from T to Register	$Rd(b) = T$	Brak	1
SEC		Set Carry	$C = 1$	C	1
CLC		Clear Carry	$C = 0$	C	1
SEN		Set Negative Flag	$N = 1$	N	1
CLN		Clear Negative Flag	$N = 0$	N	1
SEZ		Set Zero Flag	$Z = 1$	Z	1
CLZ		Clear Zero Flag	$Z = 0$	Z	1
SEI		Global Interrupt Enable	$I = 1$	I	1
CLI		Global Interrupt Disable	$I = 0$	I	1
SES		Set Signed Test Flag	$S = 1$	S	1
CLS		Clear Signed Test Flag	$S = 0$	S	1
SEV		Set Twos Complement Overflow	$V = 1$	V	1
CLV		Clear Twos Complement Overflow	$V = 0$	V	1
SET		Set T in SREG	$T = 1$	T	1
CLT		Clear T in SREG	$T = 0$	T	1

Mnemonic	Operand	Opis działania	Operacja	Ustawiane flagi	Taktów
SHE		Set Half Carry Flag in SREG	H = 1	H	1
CLH		Clear Half Carry Flag in SREG	H = 0	H	1
NOP		No Operation		Brak	1
SLEEP		Sleep		Brak	1
WDR		Watchdog Reset		Brak	1

*) Działa tylko w niektórych mikrokontrolerach.

Asembler nie rozróżnia wielkich i małych liter.

Skróty w opisie operandów oznaczają:

Rd	R0-R31 lub R16-R31 (zależnie od instrukcji)
Rr	R0-R31
b	stała (0-7)
s	stała (0-7)
P	stałą (0-31 / 63)
K	stała (0-255)
k	stała, zakres liczb zależny od instrukcji
q	stała (0-63)
Rdl	R24, R26, R28, R30. W instrukcjach ADIW i SBIW

Od tłumacza:

Pozostawiłem angielskie opisy rozkazów, gdyż po przetłumaczeniu ich na polski nie będą kojarzyć się ze skrótami mnemonicznymi.

Biblioteki

Niektóre z instrukcje czy funkcje są dostępne tylko po dołączeniu do programu odpowiedniej biblioteki. Niektóre z bibliotek są dołączane do podstawowego pakietu BASCOM AVR, inne zaś są dostępne osobno.

Poniżej znajduje się aktualna lista wszystkich bibliotek.

Biblioteka AT_EMUALTOR	(ponadstandardowa)
Biblioteka BCD	
Biblioteka BCCARD	(ponadstandardowa)
Biblioteka DATETIME	
Biblioteka EUROTIMEDATE	(wycofywana)
Biblioteka FP_TRIG	
Biblioteka GLCD	
Biblioteka GLCDSED	
Biblioteka I2C	
Biblioteka I2CSLAVE	(ponadstandardowa)
Biblioteka LCD4	
Biblioteka LCD4BUSY	
Biblioteka LCD4E2	
Biblioteka MCSBYTE	
Biblioteka MCSBYTEINT	
Biblioteka MODBUS	
Biblioteka PS2MOUSE_EMULATOR	(ponadstandardowa)
Biblioteka SPISLAVE	
Biblioteka SQR	
Biblioteka SQR_IT	
Biblioteka TCPIP	(ponadstandardowa)

Uwaga! Do wersji DEMO są dołączone tylko skompilowane wersje bibliotek (pliki .LBX).

Biblioteka **AT_EMULATOR** (Nowość w wersji 1.11.7.3)

Biblioteka **AT_EMULATOR.LIB** jest biblioteką dodatkową, którą można dokupić oddzielnie. Zawiera ona kod obsługujący emulator klawiatury AT PS/2. Wzbogaca ona język BASCOM o następujące instrukcje:

CONFIG ATEMU
SENDESCANKBD

CONFIG ATEMU (Nowość w wersji 1.11.7.3)

Przeznaczenie:

Konfiguruje emulator klawiatury PC AT.

Instrukcja korzysta z biblioteki AT_EMUALTOR która jest rozprawdzana oddzielnie przez MCS Electronics.

Składnia:

CONFIG ATEMU = INT0 | INT1 , DATA = linia_danych , CLOCK = linia_zegara

gdzie:

linia_danych nazwa końcówki portu, która będzie podłączona do linii DATA klawiatury. Musi być taka sama jak numer końcówki wejścia przerwania INT.

linia zegara nazwa końcówki portu, połączonej z linią CLOCK klawiatury.

Opis:

Klawiatura PC wykorzystuje tylko 2 linie sygnałowe do komunikacji komputer-kalwiatura. Są to sygnały: DATA i CLOCK, oraz oczywiście masa. Ponadto na złącze klawiatury wyprowadzone jest także zasilanie +5V, które może być wykorzystane do zasilania procesora. Należy jednak pamiętać by pobór prądu nie przekraczał 1A (*przyp. tłumacza*).

Rozkład sygnałów na wyprowadzeniach klawiatury komputera PC jest przedstawiony przy opisie instrukcji **CONFIG KEYBOARD**.

Instrukcja CONFIG ATEMU korzysta z jednej z dwóch linii przerywających **INT0** lub **INT1**, do której należy także dołączyć linię DATA klawiatury. Instrukcja sam ustawia odpowiednią procedurę przerwania dla wybranej końcówki INT.

Procedura ta jest wykonywana jeśli wykryto stan niski wybranej końcówce INT. Jej treść jest umieszczona w bibliotece `AT_EMULATOR.LIB`.

Uwaga! Instrukcja nie włącza globalnego systemu przerw, należy go zatem włączyć w programie.

Procedura przerwania odbiera dane z komputera PC oraz wysyła odpowiednie rozkazy do komputera PC.

Instrukcja **SENDSCANKBD** pozwala wysłać ustalony ciąg danych jako dane z klawiatury.

W przeciwieństwie do emulatora myszy PS/2 emulator klawiatury jest rozpoznawany przez komputer PC jak normalna klawiatura.

Zobacz także: **SENDSCANKBD** , **Biblioteka AT_EMULATOR**

Przykład:

```
'-----
'
'                                     PS2_KBDEMUL.BAS
'                                     (c) 2002-2003 MCS Electronics
'                                     Emulator klawiatury PS2 AT
'-----
'-----
$regfile = "2313def.dat"
$crystal = 4000000
$baud = 19200

$lib "mcsbyteint.lbx"      'skorzystamy z opcjonalnej biblioteki
                           'gdyż używane są tylko zmienne bajtowe

'konfiguracja końcówek PS2 AT
Enable Interrupts          'przerwania należy włączyć samodzielnie
Config Atemu = Int1 , Data = Pind.3 , Clock = Pinb.0
'                           ^----- używane przerwanie
'                           ^----- końcówka połączona z linią
DATA
'                           ^-- końcówka połączona z linią
CLOCK
'Uwaga! Linia DATA musi być podłączona do końcówki odpowiedniego
przerwania

Waitms 500                  'opcjonalnie

'rcall _AT_KBD_INIT
```

```
Print "Naciśnij t by przetestować oraz przełącz się do okna dowolnego edytora"
```

```
Dim Key2 As Byte , Key As Byte
Do
  Key2 = Waitkey()          ' odbierz znak z terminala
  Select Case Key2
    Case "t" :
      Waitms 1500
      Sendscankbd Mark      ' wyślij ciąg scan code
    Case Else
  End Select
Loop
```

```
Print Hex(key)
```

```
Mark:                          ' wysyła mark
  Data 12 , &H3A , &HF0 , &H3A , &H1C , &HF0 , &H1C , &H2D , &HF0 ,
&H2D , &H42 , &HF0 , &H42
  ' ^ wysyła 12 bajtów
  ' m a r k
```

SENDSCAN (Nowość w wersji 1.11.7.3)

Przeznaczenie:

Przesyła wcześniej ustawiony ciąg scan kodów.

Instrukcja korzysta z biblioteki *AT_EMULATOR* która jest rozprawdzana oddzielnie przez MCS Electronics.

Składnia:

SENDSCAN etykieta

gdzie:

etykieta

etykieta, adres linii DATA gdzie umieszczono ciąg scan kodów.

Opis:

Instrukcja SENDSCANKBKD pozwala na wysłanie ciągu scan kodów do portu klawiatury komputera PC.

Pod podaną etykietą musi znajdować się ciąg bajtów, umieszczony na przykład w liniach DATA, które to reprezentują kolejne kody. Pierwszy bajt musi określać ilość bajtów do wysłania.

W poniższej tabeli umieszczono listę wszystkich kodów:

AT KEYBOARD SCANCODES

Table reprinted with permission of Adam Chapweske

<http://panda.cs.ndsu.nodak.edu/~achapwes>

KEY	MAKE	BREAK	KEY	MAKE	BREAK	KEY	MAKE	BREAK
A	1C	F0,1C	9	46	F0,46	[54	F0,54
B	32	F0,32	`	0E	F0,0E	INSERT	E0,70	E0,F0,70
C	21	F0,21	-	4E	F0,4E	HOME	E0,6C	E0,F0,6C
D	23	F0,23	=	55	F0,55	PG UP	E0,7D	E0,F0,7D
E	24	F0,24	\	5D	F0,5D	DELETE	E0,71	E0,F0,71
F	2B	F0,2B	BKSP	66	F0,66	END	E0,69	E0,F0,69
G	34	F0,34	SPACE	29	F0,29	PG DN	E0,7A	E0,F0,7A
H	33	F0,33	TAB	0D	F0,0D	U ARROW	E0,75	E0,F0,75
I	43	F0,43	CAPS	58	F0,58	L ARROW	E0,6B	E0,F0,6B

KEY	MAKE	BREAK	KEY	MAKE	BREAK	KEY	MAKE	BREAK
J	3B	F0, 3B	L SHFT	12	F0, 12	D ARROW	E0, 72	E0, F0, 72
K	42	F0, 42	L CTRL	14	F0, 14	R ARROW	E0, 74	E0, F0, 74
L	4B	F0, 4B	L GUI	E0, 1F	E0, F0, 1F	NUM	77	F0, 77
M	3A	F0, 3A	L ALT	11	F0, 11	KP /	E0, 4A	E0, F0, 4A
N	31	F0, 31	R SHFT	59	F0, 59	KP *	7C	F0, 7C
O	44	F0, 44	R CTRL	E0, 14	E0, F0, 14	KP -	7B	F0, 7B
P	4D	F0, 4D	R GUI	E0, 27	E0, F0, 27	KP +	79	F0, 79
Q	15	F0, 15	R ALT	E0, 11	E0, F0, 11	KP EN	E0, 5A	E0, F0, 5A
R	2D	F0, 2D	APPS	E0, 2F	E0, F0, 2F	KP .	71	F0, 71
S	1B	F0, 1B	ENTER	5A	F0, 5A	KP 0	70	F0, 70
T	2C	F0, 2C	ESC	76	F0, 76	KP 1	69	F0, 69
U	3C	F0, 3C	F1	05	F0, 05	KP 2	72	F0, 72
V	2A	F0, 2A	F2	06	F0, 06	KP 3	7A	F0, 7A
W	1D	F0, 1D	F3	04	F0, 04	KP 4	6B	F0, 6B
X	22	F0, 22	F4	0C	F0, 0C	KP 5	73	F0, 73
Y	35	F0, 35	F5	03	F0, 03	KP 6	74	F0, 74
Z	1A	F0, 1A	F6	0B	F0, 0B	KP 7	6C	F0, 6C
0	45	F0, 45	F7	83	F0, 83	KP 8	75	F0, 75
1	16	F0, 16	F8	0A	F0, 0A	KP 9	7D	F0, 7D
2	1E	F0, 1E	F9	01	F0, 01]	5B	F0, 5B
3	26	F0, 26	F10	09	F0, 09	;	4C	F0, 4C
4	25	F0, 25	F11	78	F0, 78	'	52	F0, 52
5	2E	F0, 2E	F12	07	F0, 07	,	41	F0, 41
6	36	F0, 36	PRNT SCR	E0, 12, E0, 7C	E0, F0, 7C, E0, F0, 12	/	4A	F0, 4A
7	3D	F0, 3D	SCROLL LOCK	7E	F0, 7E			
8	3E	F0, 3E	PAUSE	E1, 14, 77, E1, F0, 14, F0	-NONE-			

ACPI Scan Codes

Key	Make Code	Break Code
Power	E0, 37	E0, F0, 37
Sleep	E0, 3F	E0, F0, 3F
Wake	E0, 5E	E0, F0, 5E

Windows™ Multimedia Scan Codes

Key	Make Code	Break Code
Next Track	E0, 4D	E0, F0, 4D
Previous Track	E0, 15	E0, F0, 15
Stop	E0, 3B	E0, F0, 3B
Play/Pause	E0, 34	E0, F0, 34
Mute	E0, 23	E0, F0, 23
Volume Up	E0, 32	E0, F0, 32
Volume Down	E0, 21	E0, F0, 21
Media Select	E0, 50	E0, F0, 50
E-Mail	E0, 48	E0, F0, 48
Calculator	E0, 2B	E0, F0, 2B
My Computer	E0, 40	E0, F0, 40
WWW Search	E0, 10	E0, F0, 10
WWW Home	E0, 3A	E0, F0, 3A
WWW Back	E0, 38	E0, F0, 38
WWW Forward	E0, 30	E0, F0, 30
WWW Stop	E0, 28	E0, F0, 28
WWW Refresh	E0, 20	E0, F0, 20
WWW Favorites	E0, 18	E0, F0, 18

Przykładowo by zaemulować operację zwiększenia głośności, linia DATA powinna mieć postać:

```

Data 5 , &HE0, &H32, &HE0, &HF0 , &H32
\      ^ wysyłamy 5 bajtów
\      ^ naciśnij klawisz zwiększ głośność
\      ^ zwolnij klawisz

```

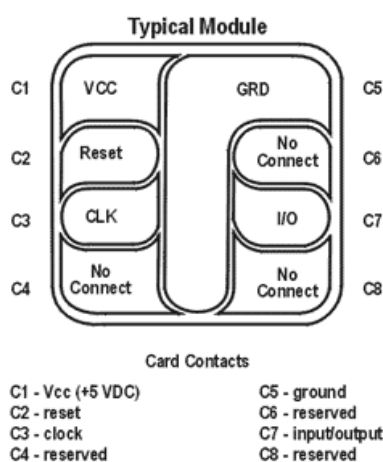
Zobacz także: **CONFIG ATEMU**, **Biblioteka AT_EMULATOR**

Biblioteka BCCARD

Biblioteka `BCCARD.LIB` jest rozprowadzana oddzielnie przez MCS Electronics. Za pomocą niej można komunikować się z inteligentnymi kartami BasicCard, których opis znajduje się na stronie <http://www.basiccard.com/>

Karty typu BasicCards są także rozprowadzane przez MCS Electronics. Karty te można programować w języku BASIC.

Układ umieszczony na karcie (jego pola kontaktowe) wygląda tak:



Rysunek 26 Typowy układ styków kontaktowych na kartach Smart Card.

Pole kontaktowe C6 opisane jako „No connect” może być wykorzystywane do podawania napięcia programującego V_{pp} (przyp. tłumacza).

Aby połączyć pola kontaktowe karty, należy użyć standardowego gniazda SmartCard. W przykładzie są używane następujące połączenia:

Tabela 18 Połączenia między kartą Smart a mikrokontrolerem.

Pole kontaktowe Smart Card	Podłączone do
C1	Vcc (+5 V)
C2	PORTD.4, RESET
C3	końcówka 4 układu 90s2313, CLOCK
C5	Masa
C7	PORTD.5, I/O

Mikroprocesor musi być taktowany kwarcem o częstotliwości 3,579545MHz, gdyż taka jest częstotliwość pracy układu na karcie. Wyjście układu generatora zegarowego należy wtedy połączyć z polem CLOCK karty.

Biblioteka wymaga paru zmiennych globalnych umieszczonych w programie. Są one definiowane automatycznie, gdy w programie umieszczono instrukcję **CONFIG BCCARD**. Te zmienne to:

_Bc_pcb : bajt używany przez protokół komunikacji z kartą,
SW1 i SW2 : obie typu Byte, odpowiedniki zmiennych SW1 i SW2 w BasicCard.

Specjalnie dla kart BasicCard stworzono następujące instrukcje:

CONFIG BCCARD – inicjalizującą bibliotekę
BCRESET – wysyłającą sygnał reset do karty,
BCDEF – definiującą procedurę użytkownika na karcie,
BCCALL – wywołuje funkcję zawartą na karcie.

Uwaga! Kodowanie nie jest obsługiwane przez tą bibliotekę.

CONFIG BCCARD

Przeznaczenie:

Inicjalizuje bibliotekę i ustala końcówki które połączone są z polami stykowymi BasicCard.

Ta instrukcja korzysta z biblioteki BCCARD, która jest rozprawdzana oddzielnie przez MCS Electronics.

Składnia:

CONFIG BCCARD = port , IO = pin_we_wy , RESET = pin_rst

gdzie:

<i>port</i>	określa który z portów jest połączony z kartą, dla większości kontrolerów może być podany PORTB lub PORTD,
<i>pin_we_wy</i>	określa która z końcówek portu ma być linia transmitującą dane (0 – 7),
<i>pin_rst</i>	określa która z końcówek portu ma być linią sterującą końcówką RESET.

Opis:

Instrukcja CONFIG BCCARD automatycznie tworzy zmienne SW1, SW2 oraz _BC_PCB.

Zobacz także: **BCRESET** , **BCDEF** , **BCCALL**

Przykład:

```
'----- konfiguracja końcówek -----  
Config Bccard = D , Io = 5 , Reset = 4  
'                                     ^ PORTD.4  
'                               ^----- PORTD.5  
'               ^----- PORT D
```

BCRESET

Przeznaczenie:

Resetuje kartę wysyłając ATR.

Ta instrukcja korzysta z biblioteki BCCARD, która jest rozprawdzana oddzielnie przez MCS Electronics.

Składnia:

BCRESET
tablica(1) = BCRESET()

gdzie:

tablica Jeśli instrukcja BCRESET jest używana w charakterze

funkcji, zwraca odpowiedź karty po wysłaniu ATR. Dane trafiają wtedy do tablicy, która musi mieć tyle elementów by pomieścić wszystkie dane wysyłane przez kartę. Wystarczy tablica składająca się a 25 bajtów.

Opis:

W przykładzie pokazano przykładowe dane jakie karta zwraca po ATR. Reszty informacji należy szukać w instrukcji obsługi BasicCard.

Jeśli przesyłane dane nie są potrzebne można użyć BCRESET jako instrukcji.

Zobacz także: CONFIG BCCARD , BCDEF , BCCALL , Biblioteka BCCARD.LIB

Przykład: (bez instrukcji inicjalizującej)

```
'-----teraz wyślemy ATR
Dim Buf(25) As Byte , I As Byte

Buf(1) = Bcreset()
For I = 1 To 25
    Print I ; " " ; Hex(buf(i))
Next

'typowe dane zwracane przez kartę:
'TS = 3B
'T0 = EF
'TB1 = 00
'TC1 = FF
'TD1 = 81 T=1 indication
'TD2 = 31 TA3,TB3 follow T=1 indicator
'TA3 = 50 or 20 IFSC -> 50 =Compact Card, 20 = Enhanced Card
'TB3 = 45 BWT block waiting time
'T1 -Tk = 42 61 73 69 63 43 61 72 64 20 5A 43 31 32 33 00 00
'      B a s i c C a r d       Z C 1 2 3
```

BCDEF

Przeznaczenie:

Definiuje nazwę i liczbę parametrów procedury w BasicCard która może być wywołana przez BASCOM BASIC.

Ta instrukcja korzysta z biblioteki BCCARD, która jest rozprowadzana oddzielnie przez MCS Electronics.

Składnia:

BCDEF nazwa([parametr_1 , parametr_n])

gdzie:

nazwa

nazwa procedury. Może się różnić od nazwy procedury zawartej w BasicCard, lecz lepiej użyć tej samej nazwy, opcjonalnie można przekazywać parametry. Dla każdego parametru należy określić jego typ, który może być: Byte, Integer, Word, Long, Single oraz String.

parametr_1

parametr_n

Opis:

Dla przykładu: **Bcdef** Calc(string). Zdefiniuje nazwę procedury Calc z jednym parametrem typu String. Jeśli jest używany typ String, a parametrów jest więcej, musi on być ostatnim na liście: **Bdcef** nazwa(byte , string).

BCDEF nie generuje kodu programu. Informuje tylko kompilator o typie przekazywanych parametrów.

Zobacz także: **CONFIG BCCARD** , **BCCALL** , **BCRESET**

Przykład: (bez instrukcji inicjalizującej)

```
'definicja procedury w programie zawartym w BasicCard
Bcdef Paramtest(Byte , Word , Long )
```

BCCALL

Przeznaczenie:

Wywołuje procedurę lub podprogram zawarty w BasicCard.

Ta instrukcja korzysta z biblioteki BCCARD, która jest rozprowadzana oddzielnie przez MCS Electronics.

Składnia:

BCCALL nazwa(NAD , CLA , INS , p1 , p2 [, param1 , paramn])

gdzie

<i>nazwa</i>	nazwa procedury. Musi być wcześniej zdefiniowana przez BCDEF. Podana nazwa może się różnić od nazwy procedury zawartej w BasicCard, lecz lepiej użyć tej samej nazwy,
<i>NAD</i>	bajt adresu połączenia. BasicCard odpowiada na każdą wartość tego adresu. By użyć domyślnego podaj 0,
<i>CLA</i>	bajt klasy. Pierwszy z dwóch bajtów rozkazu CLA INS. Musi być taki sam jaki ma procedura w BasicCard,
<i>INS</i>	bajt instrukcj. Drugi z dwóch bajtów rozkazu CLA INS. Musi być taki sam jaki ma procedura w BasicCard,
<i>p1</i>	pierwszy parametr nagłówek CLA-INS
<i>p2</i>	drugi parametr nagłówek CLA-INS
<i>parametr_1</i>	opcjonalnie można przekazywać parametry. Dla każdego
<i>parametr_n</i>	parametru należy określić jego typ, który może być: Byte, Integer, Word, Long, Single oraz String.

Opis:

Użycie w programie BCCALL, parametry NAD, CLA, INS, P1 oraz P2 są przesyłane do BasicCard. Parametry użytkownika są także przesyłane do programu w karcie. Procedura zawarta w BasicCard wykonuje określony program (numer identyfikacyjny podany w CLA i INS), oraz zwraca rezultat w zmiennych SW1 i SW2.

Wartości parametrów, które została zmieniona przez BasicCard także zostają zwrócone.

Uwaga! Nie można przesłać stałych, tylko zmienne. Jest to spowodowane tym, iż wartości stałych nie mogą zostać zmienione.

Dla przykładu. Jeśli w programie zawartym w BasicCard zawarto procedurę (przykładowo):

```
'test of passing parameters
Command &hf6 &h01 ParamTest( b as byte, w as integer, l as long)
  b=b+1
  w=w+1
  l=l+1
end command
```

w wywołaniu procedury należy podać **&HF6** jako parametr CLA oraz **1** jako INS:


```

Dim S As String * 15

'szybkość transmisji może zostać zmieniona
$baud = 9600
'częstotliwość kwarcu musi być ustawiona na 3.5795MHz gdyż generator
taktujący
'także wytwarza przebieg zegarowy dla karty
$crystal = 3579545

'Wykonaj ATR
Bcreset

'teraz wywołamy procedurę w BasicCard
'bccall funcname(nad,cla,ins,p1,p2,PRM as TYPE,PRM as TYPE)
S = "1+1+3" 'chcemy aby obliczyła rezultat
tej operacji matematycznej

Bccall Calc(0 , &H20 , 1 , 0 , 0 , S)
'
'                                     ^--- zmienna która zawiera
wyrażenie
'
'                                     ^----- P2
'                                     ^----- P1
'                                     ^----- INS
'                                     ^----- CLA
'                                     ^----- NAD
'By dowiedzieć się czegoś więcej o parametrach NAD, CLA, INS, P1 oraz
P2 zobacz instrukcję do karty BasicCard
'jeśli wystąpił błąd to zmienna ERR jest ustawiona
'BCCALL zwraca także wartości w zmiennych SW1 i SW2
Print "Rezultat działania : " ; S
Print "SW1 = " ; Hex(sw1)
Print "SW2 = " ; Hex(sw2)
'Print Hex(_bc_pcb) 'dla testów można zobaczyć zmiany w tej
zmiennej
Print "Błąd : " ; Err

'Możesz wywołać procedurę obliczającą inne działanie w tej sesji

S = "2+2"
Bccall Calc(0 , &H20 , 1 , 0 , 0 , S)
Print "Rezultat działania : " ; S
Print "SW1 = " ; Hex(sw1)
Print "SW2 = " ; Hex(sw2)
'Print Hex(_bc_pcb) 'dla testów można zobaczyć zmiany w tej
zmiennej
Print "Błąd : " ; Err

'teraz następny ATR
Bcreset
Input "Podaj wyrażenie " , S
Bccall Calc(0 , &H20 , 1 , 0 , 0 , S)
Print "Odpowiedź : " ; S

'-----a teraz odczytamy dane wysłane po ART
Dim Buf(25) As Byte , I As Byte
Buf(1) = Bcreset()
For I = 1 To 25
    Print I ; " " ; Hex(buf(i))

```

Next

```
'typowe dane zwracane przez kartę:
'TS = 3B
'T0 = EF
'TB1 = 00
'TC1 = FF
'TD1 = 81 T=1 indication
'TD2 = 31 TA3,TB3 follow T=1 indicator
'TA3 = 50 or 20 IFSC -> 50 =Compact Card, 20 = Enhanced Card
'TB3 = 45 BWT block waiting time
'T1 -Tk = 42 61 73 69 63 43 61 72 64 20 5A 43 31 32 33 00 00
'      B a s i c C a r d      Z C 1 2 3

'i jeszcze jeden test
'definicja procedury w programie zawartym w BasicCard
Bcdef Paramtest(byte , Word , Long )

'potrzebne zmienne
Dim B As Byte , W As Word , L As Long

'ustalamy wartości
B = 1 : W = &H1234 : L = &H12345678

Bccall Paramtest(0 , &HF6 , 1 , 0 , 0 , B , W , L)
Print Hex(sw1) ; Spc(3) ; Hex(sw2)
'i zobaczymy jak karta zmieniła zawartość parametrów!
Print B ; Spc(3) ; Hex(w) ; " " ; Hex(l)

'próba rozkazu Echotest
Bcdef Echotest(byte)
Bccall Echotest(0 , &HC0 , &H14 , 1 , 0 , B)
Print B
End                                'koniec programu
```

Kod źródłowy programu zawartego w BasicCard.

```
Rem BasicCard Sample Source Code
Rem -----
Rem Copyright (C) 1997-2001 ZeitControl GmbH
Rem You have a royalty-free right to use, modify, reproduce and
Rem distribute the Sample Application Files (and/or any modified
Rem version) in any way you find useful, provided that you agree
Rem that ZeitControl GmbH has no warranty, obligations or liability
Rem for any Sample Application Files.
Rem -----

#include CALCKEYS.BAS

Declare ApplicationID = "BasicCard Mini-Calculator"

Rem This BasicCard program contains recursive procedure calls, so the
Rem compiler will allocate all available RAM to the P-Code stack
Rem unless
Rem otherwise advised. This slows execution, because all strings have
Rem to
Rem be allocated from EEPROM. So we specify a stack size here:
```

```

#Stack 120

' Calculator Command (CLA = &H20, INS = &H01)
'
' Input: an ASCII expression involving integers, and these operators:
'
'      *  /  %  +  -  &  ^  |
'
' (Parentheses are also allowed.)
'
' Output: the value of the expression, in ASCII.
'
' P1 = 0: all numbers are decimal
' P1 <> 0: all numbers are hex

' Constants
Const SyntaxError = &H81
Const ParenthesisMismatch = &H82
Const InvalidNumber = &H83
Const BadOperator = &H84

' Forward references

Declare Function EvaluateExpression (S$, Precedence) As Long
Declare Function EvaluateTerm (S$) As Long
Declare Sub Error (Code@)

'test for passing a string
Command &H20 &H01 Calculator (S$)

    Private X As Long
    S$ = Trim$ (S$)
    X = EvaluateExpression (S$, 0)
    If Len (Trim$ (S$)) <> 0 Then Call Error (SyntaxError)
    If P1 = 0 Then S$ = Str$ (X) : Else S$ = Hex$ (X)

End Command

'test of passing parameters
Command &hf6 &h01 ParamTest( b as byte, w as integer,l as long)

    b=b+1
    w=w+1
    l=l+1
end command

Function EvaluateExpression (S$, Precedence) As Long

    EvaluateExpression = EvaluateTerm (S$)

    Do
        S$ = LTrim$ (S$)
        If Len (S$) = 0 Then Exit Function

        Select Case S$(1)

            Case "*"
                If Precedence > 5 Then Exit Function

```

```

        S$ = Mid$ (S$, 2)
        EvaluateExpression = EvaluateExpression * _
            EvaluateExpression (S$, 6)
    Case "/"

        If Precedence > 5 Then Exit Function
        S$ = Mid$ (S$, 2)
        EvaluateExpression = EvaluateExpression / _
            EvaluateExpression (S$, 6)
    Case "%"
        If Precedence > 5 Then Exit Function
        S$ = Mid$ (S$, 2)
        EvaluateExpression = EvaluateExpression Mod _
            EvaluateExpression (S$, 6)
    Case "+"
        If Precedence > 4 Then Exit Function
        S$ = Mid$ (S$, 2)

        EvaluateExpression = EvaluateExpression + _
            EvaluateExpression (S$, 5)
    Case "-"
        If Precedence > 4 Then Exit Function
        S$ = Mid$ (S$, 2)
        EvaluateExpression = EvaluateExpression - _
            EvaluateExpression (S$, 5)
    Case "&"
        If Precedence > 3 Then Exit Function
        S$ = Mid$ (S$, 2)
        EvaluateExpression = EvaluateExpression And _
            EvaluateExpression (S$, 4)

    Case "^"
        If Precedence > 2 Then Exit Function
        S$ = Mid$ (S$, 2)
        EvaluateExpression = EvaluateExpression Xor _
            EvaluateExpression (S$, 3)
    Case "|"
        If Precedence > 1 Then Exit Function
        S$ = Mid$ (S$, 2)
        EvaluateExpression = EvaluateExpression Or _
            EvaluateExpression (S$, 2)
    Case Else
        Exit Function
    End Select

Loop

End Function

Function EvaluateTerm (S$) As Long

    Do
        S$ = LTrim$ (S$)
        If Len (S$) = 0 Then Call Error (SyntaxError)
        If S$(1) <> "+" Then Exit Do
        S$ = Mid$ (S$, 2)
    Loop

    If S$(1) = "(" Then
        S$ = Mid$ (S$, 2)

```

```

    EvaluateTerm = EvaluateExpression (S$, 0)
    S$ = LTrim$ (S$)
    If S$(1) <> ")" Then Call Error (ParenthesisMismatch)

    S$ = Mid$ (S$, 2)
    Exit Function

ElseIf S$(1) = "-" Then    ' Unary minus
    S$ = Mid$ (S$, 2)
    EvaluateTerm = -EvaluateTerm (S$)
    Exit Function

Else
    ' Must be a number
    If P1 = 0 Then        ' If decimal
        EvaluateTerm = Val& (S$, L@)
    Else
        EvaluateTerm = ValH (S$, L@)
    End If
    If L@ = 0 Then Call Error (InvalidNumber)
    S$ = Mid$ (S$, L@ + 1)

End If

End Function

Sub Error (Code@)
    SW1 = &H64
    SW2 = Code@
    Exit
End Sub

```

Biblioteka DATETIME *(Nowość w wersji 1.11.7.3)*

Bibliotekę DATETIME.LIB napisał **Josef Franz Vögel**. Rozszerza ona procedury dotyczące czasu i daty o możliwość manipulacji tymi danymi. Udostępnia następujące funkcje:

DAYOFWEEK
DAYOFYEAR
SECOFDAY
SECELAPSED
SYSDAY
SYSSEC
SYSSECELAPSED

DATE
TIME

Uwaga! Nie należy mylić funkcji DATE i TIME ze zmiennymi specjalnymi DATE\$, TIME\$.

Biblioteka EUROTIMEDATE *(Nowość w wersji 1.11.6.9)*

Instrukcja **CONFIG CLOCK** używa jednego z liczników pracujących w trybie asynchronicznym (układy AT90s8535, M163, M103 lub M128), w celu programowego odmierzenia czasu rzeczywistego.

Normalnie zmienna DATE\$ zwraca datę podaną w systemie amerykańskim: MM/DD/RR. Dołączając jednak bibliotekę EUROTIMEDATE.LBX :

```
$lib "eurotimedate.lbx"
```

można zmienić format zapisu daty na europejski: DD-MM-RR.

Uwaga! Biblioteka **EUROTIMEDATE** nie powinna być dalej używana. Nowe wersje (>1.11.7.2) dostarczane są z biblioteką **DATETIME** oferującą znacznie więcej możliwości.

Zobacz także: **TIME\$** , **DATE\$**.

Biblioteka FP_TRIG (Nowość w wersji 1.11.6.8)

Bibliotekę **FP_TRIG** napisał **Josef Franz Vögel**. MCS Electronics pragnie podziękować mu za wielki wkład w rozwój języka BASCOM BASIC.

Wszystkie funkcje trygonometryczne są umieszczone w bibliotece `fp_trig.lib`. Plik `fp_trig.lbx` zawiera skompilowany kod biblioteczny.

Poniższy przykład ilustruje sposób działania wszystkich funkcji z biblioteki:

```
'-----  
'  
'                                TEST_FPTRIG2.BAS  
'Demonstruje funkcje zaimplementowane w bibliotece FP_TRIG , której  
autorem jest Josef Franz Vögel  
'-----  
-----  
  
$regfile = "8515def.dat"  
$lib "fp_trig.lbx"  
  
Dim S1 As Single , S2 As Single , S3 As Single , S4 As Single , S5 As  
Single  
Dim Vcos As Single , Vsin As Single , Vtan As Single , Vatan As Single  
Dim Wi As Single , B1 As Byte  
Dim Ms1 As Single  
  
Const Pi = 3.14159265358979  
  
'kalkulacja liczby Pi  
Ms1 = Atn(1) * 4  
  
Testing_Power:  
Print "Testowanie potęgowania X ^ Y"  
Print "X          Y          x^Y"  
For S1 = 0.25 To 14 Step 0.25  
    S2 = S1 \ 2  
    S3 = Power(s1 , S2)  
    Print S1 ; " ^ " ; S2 ; " = " ; S3  
Next  
Print : Print : Print  
  
Testing_EXP_log:  
Print "Testowanie EXP i LOG"  
Print "x      exp(x)      log([exp(x)])      Error-abs      Error-rel"  
Print "Błąd jest obliczany na podstawie obliczenie liczby e i  
powtórne jej zlogartymowanie"  
For S1 = -88 To 88  
    S2 = Exp(s1)  
    S3 = Log(s2)  
    S4 = S3 - S1  
    S5 = S4 \ S1
```



```

    Print S1 ; "      " ; S2 ; "      " ; S3 ; "      " ; S4 ; "      " ; S5 ;
" " ;
    Print
Next
Print : Print : Print

Testing_Trig:
Print "Testowanie COS, SIN oraz TAN"
Print "Kąt w stopniach   Kąt w radianach           Cos           Sin
Tan"
For Wi = -48 To 48
    S1 = Wi * 15
    S2 = Deg2rad(s1)
    Vcos = Cos(s2)
    Vsin = Sin(s2)
    Vtan = Tan(s2)
    Print S1 ; "      " ; S2 ; "      " ; Vcos ; "      " ; Vsin ; "      " ;
Vtan
Next
Print : Print : Print

Testing_ATAN:
Print "Testowanie ARCTAN"
Print "X      ATAN w radianach,      stopniach"
S1 = 1 / 1024
Do
    S2 = Atn(s1)
    S3 = Rad2deg(s2)
    Print S1 ; "      " ; S2 ; "      " ; S3
    S1 = S1 * 2
    If S1 > 1000000 Then
        Exit Do
    End If
Loop

Print : Print : Print

Testing_Int_Fract:
Print "Testowanie Int oraz Fract liczb typu Single"
Print "Value      Int      Fract"
s2 = pi \ 10
For S1 = 1 To 8
    S3 = Int(s2)
    S4 = Frac(s2)
    Print S2 ; "      " ; S3 ; "      " ; S4
    S2 = S2 * 10
Next

Print : Print : Print

Print "Testowania konwersji stopnie - radiany - stopnie"
Print "Stopnie   Radiany   Stopnie   Różnica   Zależność"

For S1 = 0 To 90
    S2 = Deg2rad(s1)
    S3 = Rad2deg(s2)
    S4 = S3 - S1
    S5 = S4 \ S1
    Print S1 ; "      " ; S2 ; "      " ; S3 ; "      " ; S4 ; "      " ; S5
Next

```

```

Testing_Hyperbolicus:
Print : Print : Print
Print "Testowanie SINH, COSH and TANH"
Print "X      sinh(x)      cosh(x)      tanh(x) "
For S1 = -20 To 20
    S3 = Sinh(s1)
    S2 = Cosh(s1)
    S4 = Tanh(s1)
    Print S1 ; "      " ; S3 ; "      " ; S2 ; "      " ; S4
Next
Print : Print : Print

TEsting_LOG10:
Print "Testowanie LOG10"
Print "X      log10(x) "
S1 = 0.01
S2 = Log10(s1)
Print S1 ; "      " ; S2
S1 = 0.1
S2 = Log10(s1)
Print S1 ; "      " ; S2
For S1 = 1 To 100
    S2 = Log10(s1)
    Print S1 ; "      " ; S2
Next

Print : Print : Print
Print "Koniec testów"

End

```

Biblioteka GLCD *(Nowość w wersji 1.11.6.8)*

GLCD.LIB (GLCD.LBX) jest biblioteką procedur dla graficznych wyświetlaczy LCD opartych na kontrolerze T6963C. Zawiera ona kod dla następujących instrukcji:

```

LOCATE
CLS
PSET
LINE
CIRCLE
SHOWPIC
SHOWPICE

```

Biblioteka GLCDSER *(Nowość w wersji 1.11.6.9)*

Biblioteka GLCDSER.LIB (GLCDSER.LBX) zawiera zmodyfikowany kod obsługi dla graficznych wyświetlaczy LCD opartych na kontrolerach z serii SEDXXXX.

W bibliotece zawarto także nowe instrukcje dla tego typu wyświetlacza:

```

LCDAT
SETFONT
GLCDCMD
GLCDDATA

```

Zobacz także przykład zapisany w pliku SED.BAS umieszczonym w katalogu SAMPLES.

Biblioteka LCD4

Wbudowane procedury sterujące wyświetlaczem LCD połączonym w trybie PIN, zostały napisane tak aby mikroprocesor mógł sterować wyświetlaczem, nawet jeśli wybrany został nawet najgorszy model połączeń. Oczywiście w wielu przypadkach zmniejsza to komplikację połączeń na płytce, lecz powoduje zwiększenie zapotrzebowania na kod.

Kiedy wielkość kodu jest dość krytyczna, można użyć na sztywno ustawionych połączeń pomiędzy LCD a procesorem. Taką usługę może spełnić biblioteka LCD4, którą wystarczy użyć instrukcją:

```
$lib "LCD4.LBX"
```

Połączenia są wtedy następujące (używane przez kod w języku assemblera):

RS	PORTB.0	
R/W	PORTB.1	w tej wersji nie jest używana końcówka R/W wyświetlacza, należy ją dołączyć do masy
E	PORTB.2	
E2	PORTB.3	opcjonalnie dla LCD posiadających dwa oddzielne kontrolery
DB4	PORTB.4	
DB5	PORTB.5	szyna danych musi pracować w trybie 4 bitowym, by
DB6	PORTB.6	zmniejszyć ilość kodu
DB7	PORTB.7	

Można zmienić port do którego podpięty jest wyświetlacz. Wystarczy w pliku źródłowym LCD4.LIB zmienić adresy w następujących liniach:

```
.EQU LCDDDR=$17          ;change to another address for DDRD ($11)
.EQU LCDPORT=$18         ;change to another address for PORTD ($12)
```

Zobacz także tekst w pliku lcdcustom4bit.bas umieszczony w katalogu SAMPLES.

Uwaga! Nadal należy wybrać typ wyświetlacza instrukcją **CONFIG LCD**.

Zobacz także opis biblioteki **LCD4E2.LIB** by dowiedzieć się jak używać wyświetlaczy z dwoma liniami E.

Uwaga! Pliki z rozszerzeniem LBX są to skompilowane wersje bibliotek (z rozszerzeniem LIB) Aby cokolwiek zmienić w treści biblioteki należy posiadać komercyjną wersję BASCOM, która zawiera wersje źródłowe.

Po zmianach należy skompilować bibliotekę narzędziem **Library Manager**.

Biblioteka LCD4E2

Wbudowane procedury sterujące wyświetlaczem LCD połączonym w trybie PIN, zostały napisane tak aby mikroprocesor mógł sterować wyświetlaczem, nawet jeśli wybrany został nawet najgorszy model połączeń. Oczywiście w wielu przypadkach zmniejsza to komplikację połączeń na płytce, lecz powoduje zwiększenie zapotrzebowania na kod.

Kiedy wielkość kodu jest dość krytyczna, można użyć na sztywno ustawionych połączeń pomiędzy LCD a procesorem. Taką usługę może spełnić biblioteka LCD4E2, którą wystarczy użyć instrukcją:

```
$lib "LCD4E2.LBX"
```

Połączenia są wtedy następujące (używane przez kod w języku assemblera):

RS	PORTB.0	
R/W	PORTB.1	w tej wersji nie jest używana końcówka R/W wyświetlacza, należy ją dołączyć do masy
E	PORTB.2	linia aktywująca pierwszy kontroler
E2	PORTB.3	linia aktywująca drugi kontroler
DB4	PORTB.4	
DB5	PORTB.5	szyna danych musi pracować w trybie 4 bitowym, by
DB6	PORTB.6	zmniejszyć ilość kodu
DB7	PORTB.7	

Można zmienić port do którego podpięty jest wyświetlacz. Wystarczy w pliku źródłowym `LCD4E2.LIB` zmienić adresy w następujących liniach:

```
.EQU LCDDDR=$17          ;change to another address for DDRD ($11)
.EQU LCDPORT=$18        ;change to another address for PORTD ($12)
```

Zobacz także tekst w pliku `lcdcustom4bit2e.bas` umieszczony w katalogu `SAMPLES`.

Wyświetlacze z dwoma liniami E, posiadają dwa oddzielne kontrolery matrycy punktów. Muszą one być sterowane razem. Ta biblioteka pozwala na wybranie aktywnej linii E przez program. Przed użyciem instrukcji `LCD` należy zatem wybrać jeden z kontrolerów.

Inicjalizacja wyświetlacza obsługuje oba kontrolery.

Zobacz także opis biblioteki `LCD4.LIB` by dowiedzieć się jak używać wyświetlaczy z jedną linią E.

Uwaga! Pliki z rozszerzeniem `LBX` są to skompilowane wersje bibliotek (z rozszerzeniem `LIB`) Aby cokolwiek zmienić w treści biblioteki należy posiadać komercyjną wersję `BASCOM`, która zawiera wersje źródłowe.

Po zmianach należy skompilować bibliotekę narzędziem [Library Manager](#).

Biblioteka `LCD4BUSY`

Biblioteka `LCD4BUSY.LIB` może być używana jeśli istotny jest czas działania programu. Podstawowa biblioteka obsługi LCD używa sztywnych opóźnień podczas komunikacji z wyświetlaczem. Biblioteka `lcd4busy.lib` używa dodatkowej końcówki (linia R/W) by móc odczytać flagę statusu wyświetlacza.

Końcówki `Db4-Db7` wyświetlacza LCD muszą być podłączone do starszej połówki portu. Inne końcówki mogą być określone przez programistę.

```
'-----
'                                     (c) 2002 MCS Electronics
' lcd4busy.bas pokazuje jak używać LCD z sprawdzaniem jego gotowości
'-----
'kod testowano dla 90s8515
$regfile = "8515def.dat"

'stk200 ma kwarc 4 MHz
$crystal = 4000000

'definicja używanej biblioteki
'używa $184 bajtów (liczba hex)
$lib "lcd4busy.lib"

'definicja potrzebnych stałych
'do testów używany jest portA
Const _lcdport = Porta
Const _lcdddr = Ddra
Const _lcdin = Pina
```

```

Const _lcd_e = 1
Const _lcd_rw = 2
Const _lcd_rs = 3

'jak zwykle musimy określić typ wyświetlacza
Config Lcd = 16 * 2

'a teraz prosty kod
Cls
Lcd "test"
Lowerline
Lcd "test"

End

```

Biblioteka MCSBYTE

Konwersja z liczby na reprezentujący jej wartość ciąg znaków jest zoptymalizowana dla zmiennych typu Byte, Integer, Word oraz Long. Konwersja ta jest przeprowadzana:

- Gdy używane są funkcje: **STR()**, **VAL()** lub **HEX()**,
- Podczas drukowania wartości zmiennych numerycznych,
- Gdy używana jest instrukcja **INPUT** dla zmiennych numerycznych.

By obsługiwać wszystkie wymienione typu zmiennych, wbudowane procedury konwersji są efektywne tylko pod względem ilości generowanego kodu. Jeśli używana jest tylko konwersja dla liczb typu Byte, można jeszcze zmniejszyć zapotrzebowanie na rozmiar kodu.

Biblioteka `mcsbyte.lib` jest zoptymalizowaną wersją, która obsługuje tylko liczby typu Byte. Aby użyć tej biblioteki należy umieścić w programie: `$lib "mcsbyte.lbx"`.

Zobacz także opis biblioteki **MCSBYTEINT**.

Uwaga! Pliki z rozszerzeniem LBX są to skompilowane wersje bibliotek (z rozszerzeniem LIB) Aby cokolwiek zmienić w treści biblioteki należy posiadać komercyjną wersję BASCOM, która zawiera wersje źródłowe.

Po zmianach należy skompilować bibliotekę narzędziem [Library Manager](#).

Biblioteka MCSBYTEINT

Konwersja z liczby na reprezentujący jej wartość ciąg znaków jest zoptymalizowana dla zmiennych typu Byte, Integer, Word oraz Long. Konwersja ta jest przeprowadzana:

- Gdy używane są funkcje: **STR()**, **VAL()** lub **HEX()**,
- Podczas drukowania wartości zmiennych numerycznych,
- Gdy używana jest instrukcja **INPUT** dla zmiennych numerycznych.

By obsługiwać wszystkie wymienione typu zmiennych, wbudowane procedury konwersji są efektywne tylko pod względem ilości generowanego kodu. Jeśli nie jest używana konwersja dla liczb typu Long, można jeszcze zmniejszyć zapotrzebowanie na rozmiar kodu.

Biblioteka `MCSBYTEINT.LIB` jest zoptymalizowaną wersją, która obsługuje tylko liczby typu Byte, Integer oraz Word. Aby użyć tej biblioteki należy umieścić w programie: `$lib "mcsbyteint.lbx"`.

Zobacz także opis biblioteki **MCSBYTE**.

Uwaga! Pliki z rozszerzeniem LBX to skompilowane wersje bibliotek (z rozszerzeniem LIB) Aby cokolwiek zmienić w treści biblioteki należy posiadać komercyjną wersję BASCOM, która zawiera

wersje źródłowe.

Po zmianach należy skompilować bibliotekę narzędziem [Library Manager](#).

Biblioteka PS2MOUSE_EMULATOR (Nowość w wersji 1.11.7.3)

Biblioteka `PS2MOUSE_EMULATOR.LIB` jest biblioteką dodatkową, którą można dokupić oddzielnie. Zawiera ona kod obsługujący emulator myszy PS/2. Wzbogaca ona język BASCOM o następujące instrukcje:

CONFIG PS2EMU
PS2MOUSEXY
SENDSCAN

CONFIG PS2EMU (Nowość w wersji 1.11.7.3)

Przeznaczenie:

Konfiguruje końcówki emulatora myszy PS2.

Instrukcja korzysta z biblioteki PS2MOUSE_EMUALTOR która jest rozprowadzana oddzielnie przez MCS Electronics.

Składnia:

CONFIG PS2EMU = INT0 | INT1 , DATA = linia_danych , CLOCK = linia zegara

gdzie:

<i>linia_danych</i>	nazwa końcówki portu, która będzie podłączona do linii DATA w gnieździe myszy. Musi być taka sama jak numer końcówki wykorzystywanego przerwania INT.
<i>linia zegara</i>	nazwa końcówki portu, połączonej z linią CLOCK w gnieździe myszy.

Opis:

Nowsze komputery (i starsze firmowe) posiadają wbudowany interfejs myszy PS2. Mysz tą podłącza się do specjalnego gniazda mini DIN (6 pinowe, żeńskie, kolor zielony).

Instrukcja **CONFIG PS2EMU** korzysta z jednej z dwóch linii przerywających **INT0** lub **INT1**, do której należy także dołączyć linię DATA. Instrukcja sam ustawia odpowiednią procedurę przerwania dla wybranej końcówki INT. Procedura przerwania jest wykonywana jeśli wykryto stan niski na wybranej końcówce INT. Treść tej procedury jest umieszczona w bibliotece `PS2MOUSE_EMULATOR.LIB`.

Uwaga! Instrukcja nie włącza globalnego systemu przerwania, należy go zatem włączyć w programie.

Do dyspozycji są dwie podstawowe instrukcje: **SENDSCAN** która wysyła ustalony ciąg danych do portu myszy; oraz **PS2MOUSEXY** przekazującą dane o przesunięciu myszy lub naciskania klawiszy myszy.

Uwaga! System mikroprocesorowy musi być podłączony przed włączeniem komputera PC. Jest to podyktowane faktem, że emulator myszy jest wykrywany tylko podczas startu (*bootowania*) komputera.

Podłączenie emulatora w trakcie działania komputera może doprowadzić do jego uszkodzenia.

Zobacz także: **PS2MOUSEXY** , **SENDSCAN** , **Biblioteka PS2MOUSE_EMULATOR**

Przykład:

```

'-----
'
'                               PS2_EMUL.BAS
'                               (c) 2003-2003 MCS Electronics
'                               PS2 Mouse emulator
'-----
'-----
$regfile = "2313def.dat"
$crystal = 4000000
$baud = 19200

$lib "mcsbyteint.lbx"           ' użyjemy biblioteki gdyż będą
używane wyłącznie zmienne bajtowe

'konfiguracja emulatora
Config Ps2emu = Int1 , Data = Pind.3 , Clock = Pinb.0
'                               ^----- użyte przerwanie
'                               ^----- końcówka podłączona do
linii DATA
'                               ^-- końcówka podłączona do
linii CLOCK
'Uwaga! Końcówka przypisana linii DATA musi być tą samą końcówką co
użyte przerwanie

Waitms 500                      ' opcjonalne opóźnienie

Enable Interrupts               ' należy włączyć system
przerwań, gdyż używamy przerwania INT

Print "Naciskaj u,d,l,r,b, lub t"
Dim Key As Byte
Do
    Key = Waitkey()             ' odczytamy dane klawisza z
terminala
    Select Case Key
        Case "u" : Ps2mousexy 0 , 10 , 0   ' w górę
        Case "d" : Ps2mousexy 0 , -10 , 0   ' w dół
        Case "l" : Ps2mousexy -10 , 0 , 0    ' w prawo
        Case "r" : Ps2mousexy 10 , 0 , 0     ' w lewo
        Case "b" : Ps2mousexy 0 , 0 , 1      ' naciśniemy lewy klawisz
        Ps2mousexy 0 , 0 , 0                ' i zwolnimy go zaraz
        Case "t" : Sendscan Mouseup         ' wyślij ciąg scan code
        Case Else
    End Select
Loop

Mouseup:
    Data 3 , &H08 , &H00 , &H01           ' mysz w górę o 1 pozycję

```

PS2MOUSEXY (Nowość w wersji 1.11.7.3)

Przeznaczenie:

Wysła informacje o przesunięciu wirtualnej myszy oraz stanie jej klawiszy.

Instrukcja korzysta z biblioteki PS2MOUSE_EMULATOR która jest rozprowadzana oddzielnie przez MCS Electronics.

Składnia:**PS2MOUSEXY** *x* , *y* , *klawisze*

gdzie:

x przesunięcie w osi X z bieżącej pozycji. Możliwe jest podanie **-255** do **255**,

y przesunięcie w osi Y z bieżącej pozycji. Możliwe jest podanie **-255** do **255**,

klawisze stan klawiszy wirtualnej myszy. Podanie:
0 - oznacza że żaden z klawiszy nie jest naciśnięty;
1 - oznacza że naciśnięto lewy klawisz myszy;
2 - oznacza że naciśnięto środkowy klawisz myszy;
4 - oznacza że naciśnięto prawy klawisz myszy.

Opis:

Stan klawiszy jest odzwierciedlany na kolejnych bitach - stąd też wartości przypisywane kolejnym klawiszom. Jedynka na odpowiednim bicie oznacza naciśnięty klawisz.

Można także dodawać kolejne wartości by zasymulować naciśnięcie kilku klawiszy naraz. Dla przykładu wartość 6 oznacza, że naciśnięte są klawisze: środkowy i prawy.

Aby przesłać informacje o pojedynczym kliknięciu należy dwa razy wykonać instrukcję PS2MOUSEXY z odpowiednimi parametrami. Na przykład:

```
Ps2mouseXY 0 , 0 , 1 \ naciśnięto lewy klawisz
Ps2mouseXY 0 , 0 , 0 \ zwolniono lewy klawisz
```

By wysłać całą serię kodów należy użyć instrukcji SENDSCAN.

Zobacz także: **SENDSCAN** , **CONFIG PS2EMU**

SENDSCAN *(Nowość w wersji 1.11.7.3)*

Przeznaczenie:

Przesyła wcześniej ustawiony ciąg scan kodów.

Instrukcja korzysta z biblioteki PS2MOUSE_EMULATOR która jest rozprowadzana oddzielnie przez MCS Electronics.

Składnia:**SENDSCAN** *etykieta*

gdzie:

etykieta etykieta, adres linii DATA gdzie umieszczono ciąg scan kodów.

Opis:

Instrukcja SENDSCAN pozwala na wysłanie ciągu scan kodów do portu myszy (PS2!) komputera PC.

Pod podaną etykietą musi znajdować się ciąg bajtów, umieszczony na przykład w liniach DATA, które to reprezentują kolejne kody. Pierwszy bajt musi określać ilość bajtów do wysłania.

W poniższej tabeli umieszczono listę wszystkich kodów:

Tabela 19 Ciągi kodów przesyłane przez mysz.

Akcja	Ciąg kodów
Przesuń w górę o jeden punkt	08, 00, 01

Akcja	Ciąg kodów
Przesuń w dół o jeden punkt	28,00,FF
Przesuń w prawo o jeden punkt	08,01,00
Przesuń w lewo o jeden punkt	18,FF,00
Naciśnięto lewy klawisz	09,00,00
Zwolniono lewy klawisz	08,00,00
Naciśnięto środkowy klawisz	0C,00,00
Zwolniono środkowy klawisz	08,00,00
Naciśnięto prawy klawisz	0A,00,00
Zwolniono prawy klawisz	08,00,00

Przykładowo by zaemulować kliknięcie lewym klawiszem myszy, linia DATA powinna mieć postać:

```
Data 6 , &H09, &H00, &H00, &H08 , &H00, &H00
\      ^ wysyłamy 6 bajtów
\      ^ naciśnięcie i
\                                     ^ zwolnienie klawisza
```

Zobacz także: [CONFIG PS2EMU](#) , [PS2MOUSEXY](#) , [Biblioteka PS2MOUSE_EMULATOR](#)

Biblioteka SPISLAVE (Nowość w wersji 1.11.6.8)

SPISLAVE.LIB (SPISLAVE.LBX) jest biblioteką, która może być użyta do stworzenia interfejsu SPI pracującego w trybie Slave w kontrolerach nie posiadających sprzętowego interfejsu SPI.

Mimo że wiele kontrolerów AVR posiada interfejs ISP służący do programowania pamięci Flash i EEPROM, to na przykład AT90s2313 nie posiada interfejsu SPI.

Jeśli zachodzi potrzeba kontrolowania kilku takich mikrokontrolerów przez SPI, można użyć w tym celu biblioteki SPISLAVE. Zawarty w spi-softslave.bas przykład (znajdujący się w katalogu SAMPLES), pokazuje przykład użycia tej biblioteki.

Zobacz także przykład zawarty w pliku spi-slave.bas, który przeznaczony jest dla sprzętowego interfejsu SPI.

Plik sendspi.bas umieszczony w katalogu z przykładami, ukazuje jak można użyć sprzętowego SPI pracującego jako Master.

```
'-----
'                               SPI-SOFTSLAVE.BAS
'                               (c) 2002 MCS Electronics
'Przykład pokazujący jak zaimplementować programowy SPI w trybie SLAVE
'-----

'Niektóre układy jak na przykład 90s2313 nie posiadają portu SPI.
'Wszystkie procedury wbudowane w BASCOM BASIC pracują w trybie master.
'Przykład ten pokazuje jak pracować w trybie Slave używając 90s2313

'używamy 90s2313
$regfile = "2313def.dat"

'XTAL
$crystal = 4000000

'szybkość UART
$baud = 19200

'definicja stałych używanych przez SPI Slave
Const _softslavespi_port = Portd      'używamy portu D
```

```

Const _softslavespi_pin = Pind           'używamy PIND jako rejestr
wejściowy
Const _softslavespi_ddr = Ddrd          'kierunek działania portu D

Const _softslavespi_clock = 5           'pD.5 jako CLOCK
Const _softslavespi_miso = 3            'pD.3 jako MISO
Const _softslavespi_mosi = 4            'pd.4 jako MOSI
Const _softslavespi_ss = 2              'pd.2 jako SS
'Wszystkie końcówki mogą być wybrane dowolnie, jedynie jako
'końcówka SS musi być użyta INT0.
'W 90s2313 jest to PD.2

'PD.3(7), MISO musi być wyjściem
'PD.4(8), MOSI
'Pd.5(9) , Clock
'PD.2(6), SS /INT0

'definicja używanej biblioteki
$lib "spislave.lbx"
'która procedura jest używana
$external _spisoftslave

'Używamy przerwania INT0 by określić kiedy układ jest wybierany
On Int0 Isr_sspi Nosave
'włączamy przerwanie
Enable Int0
'INT0 będzie reagować na zmianę z wysokiego na niski stanu końcówki
Config Int0 = Falling
'teraz włączamy globalny system przerwań
Enable Interrupts

,

Dim _ssspdr As Byte                     'to będzie rejestr SPI SLAVE
SPDR
Dim _ssspif As Bit                     'bit przerwania SPI
Dim Bsend As Byte , I As Byte , B As Byte 'inne przykładowe
zmienne

_ssspdr = 0                             'wpisujemy 0, najpierw Master
wysła dane
Do
  If _ssspif = 1 Then
    Print "Odebrano: " ; _ssspdr
    Reset _ssspif
    _ssspdr = _ssspdr + 1               'wyślemy to następnym razem
  End If
Loop

```

Biblioteka TCPIP (Nowość w wersji 1.11.7.3)

Biblioteka TCPIP.LIB pozwala na używanie układu W3100A służącego do komunikacji Internetowej. Więcej informacji znajdziesz na www.i2chip.com.

Firma MCS Electronics stworzyła specjalną płytę prototypową za pomocą której można szybko rozpocząć pracę nad wykorzystaniem komunikacji protokołem TCP/IP. Zobacz http://www.mcselec.com/easy_tcp_ip.htm by poznać więcej szczegółów.

Biblioteka tcpip.lib jest dołączona do pakietu **MCS Easy TCP/IP PCB** oraz modułu IIM7000. Normalnie biblioteka nie jest dostępna.

Biblioteka zawiera kod dla następujących instrukcji i funkcji:

CONFIG TCPIP

BASE64DEC
CLOSESOCKET
GETDSTIP
GETDSTPORT
GETSOCKET
SOCKETCONNECT
SOCKETLISTEN
SOCKETSTAT
TCPREAD
TCPWRITE
TCPWRITESTR

UDPREAD
UDPWRITE
UDPWRITESTR

CONFIG TCPIP (Nowość w wersji 1.11.7.3)

Przeznaczenie

Konfiguruje scalony układ komunikacji TCP/IP - W3100A.

*Instrukcja korzysta z biblioteki TCPIP która jest rozprowadzana oddzielnie przez MCS Electronics. Wchodzi ona m.in. w skład pakietu **MCS Easy TCP/IP**.*

Składnia:

CONFIG TCPIP = INT0 | INT1 , MAC = numer_MAC , IP = adres_IP , SUBMASK = maska , GATEWAY = adres_bramy , LOCALPORT = nr_portu , TX = tx , RX = rx

Opis:

Instrukcja służy do skonfigurowania układu W3100A oraz procedury współpracy z tym układem. Ponieważ układ generuje przerwania należy określić które przerwanie jest podłączone do układu W3100A. Dla płytki prototypowej *MCS Easy TCP/IP* należy podać INT0.

Parametr **MAC=** określa unikalny numer jednoznacznie identyfikujący układ W3100A. Jest to odpowiednik unikalnego numeru każdej karty sieciowej Ethernet.

Każdy z układów podłączony do sieci musi mieć inny numer. Numer ten składa się z ciągu 6 liczb (od 0 do 255) oddzielonej kropkami. Przykładowo: 123.00.12.34.56.78

Parametr **IP=** określa adres IP układu, który także musi być unikalny w sieci. Jeśli posiadasz sieć LAN możesz użyć: 192.168.0.10, gdyż wszystkie adresy rozpoczynające się od 192.168.0.x są zarezerwowane dla sieci LAN i nie występują jako adresy IP w sieci Internet.

Parametr **SUBMASK=** określa tzw. maskę podsieci dla układu W3100A. W większości przypadków maska ta jest ustawiona na 255.255.255.0

Parametr **GATEWAY=** określa adres domyślnej bramy. Adres ten można określić wpisując w linii poleceń systemu Windows polecenie: C:\>ipconfig Rezultat może wyglądać tak:

```
Windows 2000 IP Configuration
Ethernet adapter Local Area Connection 2:

    Connection-specific DNS Suffix  . : 
    IP Address. . . . . : 192.168.0.3
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 192.168.0.1
```

W tym przypadku adres ten to 192.168.0.1.

Wartość (typu Word) przypisana parametrowi **LOCALPORT=** jest przepisywana do wewnętrznej zmiennej **LOCAL_PORT**. Zobacz opis funkcji **GETSOCKET**. Domyślnie możesz podać 5000.

Za pomocą parametru **TX=** można określić liczbę oraz wielkość bufora transmisyjnego. Układ W3100A posiada 4 tzw. *gniazda* (socket). Dla każdego gniazda przypisano osobne dwa bity w bajcie. Stan 00 spowoduje określenie bufora na 1024 bajtów, 01 - 2048 bajtów, 10 - 4096 bajtów a 11 - 8192 bajtów. Dwa młodsze bity określają wielkość gniazda 0, a dwa najstarsze - gniazda 3. Przykładowo **TX = &B01010101** spowoduje, iż wielkość bufora dla każdego gniazda będzie wynosić 2048 bajtów.

Ponieważ bufor transmisyjny ma tylko 8KB, można go podzielić na 4 części po 2048 bajtów (tak jak wyżej). Można oczywiście określić wielkość bufora na 8KB, lecz w tym wypadku możliwe jest użycie tylko gniazda 0 i parametr powinien **TX=** powinien wyglądać tak: **TX = &B11**.

Parametr **RX=** określany jest tak samo jak **TX=** z tą różnicą że dotyczy bufora odbiorczego.

Więcej informacji można znaleźć przeglądając notę aplikacyjną układu W3100A.

Instrukcja CONFIG TCPIP może być użyta tylko raz. Przedtem należy włączyć globalny system przerwań, gdyż wykorzystywane są przerwania INT0/1. Instrukcja automatycznie inicjalizuje układ W3100A.

Po wykonaniu CONFIG TCPIP można już wysłać pakiety kontrolne poleceniem **ping**

Zobacz także: **GETSOCKET** , **SOCKETCONNECT** , **SOCKETSTAT** , **TCPWRITE** , **TCPWRITESTR** , **TCPREAD** , **CLOSESOCKET** , **SOCKETLISTEN**

Przykład:

```
Config TcpiP = Int0 , Mac = 00.00.12.34.56.78 , Ip = 192.168.0.8 ,  
Submask = 255.255.255.0 , Gateway = 192.168.0.1 , Localport = 1000 ,  
Tx = $55 , Rx = $55
```

```
`Teraz użyj polecenia PING z linii poleceń by wysłać pakiet kontrolny  
i sprawdzić czy układ odpowiada:  
'PING 192.168.0.8  
'Możesz użyć także programu EasyTcp.
```

BASE64DEC() (Nowość w wersji 1.11.7.3)

Przeznaczenie:

Wykonuje konwersję danych zapisanych w formacie Base-64 na postać oryginalną.

Ta instrukcja korzysta z biblioteki TCPIP.LIB, która jest rozprowadzana oddzielnie przez MCS Electronics. Wchodzi ona m.in. w skład pakietu MCS Easy TCP/IP.

Składnia:

rezultat = BASE64DEC (dane)

gdzie

rezultat
dane

ciąg znaków do którego wpisane będą przetworzone dane,
ciąg znaków z zakodowanym ciągiem.

Opis:

Format Base-64 nie jest żadnym z protokołem szyfrującym. Przesyła on dane w postaci ciągu 7-bitowych znaków ASCII. MIME, serwery web oraz reszta serwerów i klientów w Internecie używa

kodowania Base-64

Przeznaczeniem funkcji BASE64DEC jest wyłącznie dekodowanie informacji. Wspomaga ona przeprowadzenie operacji autoryzacji do serwera web. Gdy serwer pyta o autoryzację, klient powinien wysłać nazwę i hasło użytkownika w postaci niezaszyfrowanej, zapisanej w formacie Base-64.

Zakodowane ciągi w formacie Base-64 składają się z pary 4 bajtów. Te 4 bajty reprezentują 3 normalne bajty.

Zobacz także: CONFIG TCPIP , GETSOCKET , SOCKETCONNECT , SOCKETSTAT , TCPWRITE , TCPWRITESTR , CLOSESOCKET , SOCKETLISTEN

CLOSESOCKET (Nowość w wersji 1.11.7.3)

Przeznaczenie

Zamyka połączenie z gniazdem.

Instrukcja korzysta z biblioteki TCPIP która jest rozprowadzana oddzielnie przez MCS Electronics. Wchodzi ona m.in. w skład pakietu MCS Easy TCP/IP.

Składnia:

CLOSESOCKET nr_gniazda

gdzie:

nr_gniazda jest to numer gniazda z zakresu **0 - 3** które należy zamknąć.
Jeśli gniazdo zostało już zamknięte instrukcja nie wykonuje żadnych operacji.

Opis:

Należy zamknąć gniazdo jeśli zostanie odebrany komunikat SOCK_CLOSE_WAIT. Można także zamknąć gniazdo jeśli wymaga tego używany protokół komunikacyjny. Komunikat SOCK_CLOSE_WAIT powinien być odebrany w momencie gdy serwer zamyka połączenie.

Użycie instrukcji CLOSESOCKET powoduje zamknięcie aktualnego połączenia.

Uwaga! Nie jest wymagane oczekiwanie na komunikat SOCK_CLOSE_WAIT by zamknąć połączenie od strony klienta.

Po zamknięciu gniazda należy użyć instrukcji GETSOCKET by ponownie skorzystać z tego gniazda.

Zobacz także: CONFIG TCPIP , GETSOCKET , SOCKETCONNECT , SOCKETSTAT , TCPWRITE , TCPWRITESTR , TCPREAD , SOCKETLISTEN

GETDSTIP() (Nowość w wersji 1.11.7.3)

Przeznaczenie:

Zwraca adres IP połączanego użytkownika sieci.

Instrukcja korzysta z biblioteki TCPIP która jest rozprowadzana oddzielnie przez MCS Electronics. Wchodzi ona m.in. w skład pakietu MCS Easy TCP/IP.

Składnia:

rezultat = **GETDSTIP**(gniazdo)

gdzie:

rezultat zmienna typu Long gdzie wpisany będzie odczytany adres IP,
gniazdo zmienna lub stała określająca numer gniazda (**0 – 3**).

Opis:

Jeśli system pracuje jako serwer, może być wymagane odczytanie adresu IP połączonego klienta. Można to wykorzystać do logowania tylko wybranych klientów czy jako element systemu zabezpieczeń.

Uwaga! Bajt MSB adresu IP jest zwracany w bajcie LSB zmiennej Long.

Zobacz także: CONFIG TCPIP , GETSOCKET , SOCKETCONNECT , SOCKETSTAT , TCPWRITE , TCPWRITESTR , CLOSESOCKET , SOCKETLISTEN , GETDSTPORT

Przykład:

```
Dim L As Long
L = GetdstIP(i)      ' pobierz adres IP dla gniazda o numerze i
```

GETDSTPORT() (Nowość w wersji 1.11.7.3)

Przeznaczenie:

Zwraca numer portu połączonego użytkownika sieci.

Instrukcja korzysta z biblioteki TCPIP która jest rozprowadzana oddzielnie przez MCS Electronics. Wchodzi ona m.in. w skład pakietu MCS Easy TCP/IP.

Składnia:

rezultat = GETDSTPORT(gniazdo)

gdzie:

rezultat	zmienna typu Word gdzie wpisany będzie odczytany numer portu,
gniazdo	zmienna lub stała określająca numer gniazda (0 – 3).

Opis:

Jeśli system pracuje jako serwer, może być wymagane odczytanie numeru portu połączonego klienta. Można to wykorzystać do logowania tylko wybranych klientów czy jako element systemu zabezpieczeń.

Zobacz także: CONFIG TCPIP , GETSOCKET , SOCKETCONNECT , SOCKETSTAT , TCPWRITE , TCPWRITESTR , CLOSESOCKET , SOCKETLISTEN , GETDSTIP

Przykład:

```
Dim W As Word
W = GetdstPort(i)    ' pobierz numer portu dla gniazda o numerze i
```

GETSOCKET() (Nowość w wersji 1.11.7.3)

Przeznaczenie:

Otwiera gniazdo do komunikacji protokołem TCP/IP.

Instrukcja korzysta z biblioteki TCPIP która jest rozprowadzana oddzielnie przez MCS Electronics. Wchodzi ona m.in. w skład pakietu MCS Easy TCP/IP.

Składnia:

rezultat = GETSOCKET (gniazdo , tryb , port , opcje)

gdzie:

rezultat	zwraca numer otwartego gniazda lub 255 jeśli operacja otwarcia się nie powiodła,
----------	--

<i>gniazdo</i>	numer gniazda 0 - 3 jakie ma być otwarte,
<i>tryb</i>	określa tryb pracy gniazda,
<i>port</i>	określa port jaki używać będzie gniazdo,
<i>opcje</i>	określa dodatkowe opcje.

Opis:

Funkcja próbuje otworzyć uprzednio zamknięte gniazdo. Jeśli operacja się powiedzie funkcja zwraca numer otwartego właśnie gniazda. Niepowodzenie operacji otwarcia sygnalizowane jest zwróceniem wartości 255 (–1 w notacji U2 *przyp. tłumacza*).

Otwarcie gniazda może się odbyć w kilku predefiniowanych trybach: `sock_stream` (1), `sock_dgrm` (2), `sock_ip1_raw` (3) or `macl_raw` (4). Najlepiej zdefiniować numery trybów jako stałe.

Dla komunikacji za pomocą UDP należy podać `sock_dgrm` lub 2.

Następnym parametrem jaki trzeba określić jest port przez jaki odbywać się będzie komunikacja. Można podać dowolną wartość lecz należy pamiętać by każde gniazdo posiadało własny numer. Można także podać 0, wtedy numer portu będzie pobrany ze zmiennej `LOCAL_PORT` której wartość przypisywana jest podczas konfiguracji instrukcją **CONFIG TCPIP**. Po wykonaniu konfiguracji wartość zmiennej `LOCAL_PORT` jest zwiększana o 1, więc jako parametr *port* można podać 0.

Ostatnim parametrem można określić dodatkowe opcje:

- 0 : brak dodatkowych opcji,
- 128 : send/receive broadcast message in UDP
- 64 : use register value with designated timeout value
- 32 : when not using no delayed ack
- 16 : when not using silly window syndrome.

Dodatkowych informacji należy szukać w nocie katalogowej układu W3100A.

Po poprawnej inicjalizacji i otwarciu gniazda można użyć funkcji **SOCKETCONNECT** by połączyć się z klientem sieci lub **SOCKETLISTEN** by rozpocząć nasłuchiwanie (praca jako serwer).

Zobacz także: **CONFIG TCPIP** , **SOCKETCONNECT** , **SOCKETSTAT** , **TCPWRITE** , **TCPWRITESTR** , **TCPREAD** , **CLOSESOCKET** , **SOCKETLISTEN**

Przykład:

```
I = Getsocket(0 , Sock_stream , 5000 , 0)      'przydziel nowe
gniazdo
```

SOCKETCONNECT() *(Nowość w wersji 1.11.7.3)*

Przeznaczenie:

Nawiązuje połączenie z serwerem TCP/IP.

*Instrukcja korzysta z biblioteki TCPIP która jest rozprowadzana oddzielnie przez MCS Electronics. Wchodzi ona m.in. w skład pakietu **MCS Easy TCP/IP**.*

Składnia:

rezultat = **SOCKETCONNECT**(*gniazdo* , *adres_IP* , *port*)

gdzie:

<i>rezultat</i>	zmienna typu Byte gdzie funkcja zwraca 0 jeśli połączenie zostanie nawiązane, lub 1 gdy wystąpił błąd,
<i>gniazdo</i>	numer z zakresu 0 – 3 określający numer gniazda,
<i>adres_IP</i>	adres IP serwera z którym trzeba się połączyć,
<i>port</i>	parametr określający numer portu.

Opis:

Funkcja przeznaczona jest tylko do nawiązania połączenia z serwerem. Należy przy tym podać adres IP oraz portu, gdyż standardowo serwery posiadają dedykowane numery portów. Na przykład serwer HTTP (web server) używa portu 80.

Adres IP można podać w postaci zwykłego zapisu (4 liczby oddzielone kropkami) lub jako zmienną typu Long, w której bajt MSB określa część LSB numeru IP.

Po nawiązaniu połączenia serwer może odpowiedzieć ciągiem danych. Dane te zależą od używanego protokołu. Większość serwerów wysyła tekst powitalny, zwany banerem.

Można wysyłać lub odebrać dane po nawiązaniu połączenia. Serwer może także zakończyć połączenie lub też można zakończyć połączenie samodzielnie. To także zależy od używanego protokołu.

Zobacz także: CONFIG TCPIP , GETSOCKET , SOCKETLISTEN , SOCKETSTAT , CLOSESOCKET , TCPWRITE , TCPWRITESTR , TCPREAD

Przykład:

```
J = SocketConnect(i , 194.109.6.52 , 25) ' serwer SMTP
```

SOCKETLISTEN (Nowość w wersji 1.11.7.3)

Przeznaczenie:

Otwiera gniazdo w trybie serwera (nasłuch).

*Instrukcja korzysta z biblioteki TCPIP która jest rozprowadzana oddzielnie przez MCS Electronics. Wchodzi ona m.in. w skład pakietu **MCS Easy TCP/IP**.*

Składnia:

SOCKETLISTEN numer

gdzie:

numer numer z zakresu **0 – 3** określający numer gniazda.

Opis:

Wykonanie tej instrukcji powoduje, że wybrane gniazdo będzie nasłuchiwać portu, wybranego wcześniej za pomocą funkcji **GETSOCKET**. Jednocześnie można nasłuchiwać maksimum 4 gniazda.

Po zamknięciu połączenia – przez klienta lub przez serwer – ustalone musi być nowe połączenie by móc ponownie użyć instrukcji **SOCKETLISTEN**.

Zobacz także: CONFIG TCPIP , GETSOCKET , SOCKETCONNECT , SOCKETSTAT , CLOSESOCKET , TCPWRITE , TCPWRITESTR , TCPREAD

Przykład:

```
I = Getsocket(0 , Sock_stream , 5000 , 0) ' otwórz nowe gniazdo
Socketlisten I ' nasłuchuj
#if Debug
Print "Nasłuchuję gniazdo: " ; I
#endif
```

SOCKETSTAT() (Nowość w wersji 1.11.7.3)

Przeznaczenie:

Zwraca informacje o wybranym gnieździe.

Instrukcja korzysta z biblioteki TCPIP która jest rozprowadzana oddzielnie przez MCS

Electronics. Wchodzi ona m.in. w skład pakietu **MCS Easy TCP/IP**.

Składnia:

rezultat = SOCKETSTAT(gniazdo , tryb)

gdzie:

rezultat zmienna typu Word gdzie funkcja zwróci stan gniazda,
gniazdo numer z zakresu **0 – 3** określający numer gniazda.
tryb parametr określający jaką informację ma zwrócić funkcja.

Opis:

Funkcja SOCKETSTAT aktualnie zwraca 3 typy informacji. Jedną z nich wybiera się podając odpowiedni parametr (**tryb**):

SEL_CONTROL (0) Funkcja zwraca wartość z rejestru statusu;
SEL_SEND (1) Funkcja zwraca liczbę bajtów które można umieścić w buforze transmisji;
SEL_RECV (2) Funkcja zwraca liczbę bajtów które są umieszczone w buforze odbiorczym;

Jeśli wybrany zostanie tryb **0 (SEL_CONTROL)** funkcja zwróci jedną z poniższych wartości:

Tabela 20 Wartości zwracane przez funkcję SOCKETSTAT w trybie 0.

Wartość	Stan	Opis
0	SOCK_CLOSED	Połączenie zamknięte
1	SOCK_ARP	Oczekiwanie na odpowiedź po nadaniu ARP
2	SOCK_LISTEN	Oczekiwanie na ustawienie połączenia przez klienta podczas pracy w trybie pasywnym
3	SOCK_SYSENT	Oczekiwanie na SYN, ACK po transmisji SYN dla ustawienia połączenia podczas pracy w trybie aktywnym
4	SOCK_SYSENT_ACK	Zakończenie ustawienia połączenia po odebraniu SYN, ACK i nadanie ACK podczas pracy w trybie aktywnym
5	SOCK_SYNCRCV	SYN, ACK został wysłany po odebraniu SYN przez klienta pracującego w trybie nasłuchu, tryb pasywny
6	SOCK_ESTABLISHED	Ustawienie połączenia zostało zakończone w trybie aktywnym/pasywnym
7	SOCK_CLOSE_WAIT	Połączenie zostało przerwane
8	SOCK_LAST_ACK	Połączenie zostało przerwane
9	SOCK_FIN_WAIT1	Połączenie zostało przerwane
10	SOCK_FIN_WAIT2	Połączenie zostało przerwane
11	SOCK_CLOSING	Połączenie zostało przerwane
12	SOCK_TIME_WAIT	Połączenie zostało przerwane
13	SOCK_RESET	Połączenie zostało przerwane po odebraniu pakietu zerującego
14	SOCK_INIT	Gniazdo jest inicjalizowane
15	SOCK_UDP	Właściwy kanał jest zainicjalizowany w trybie UDP
16	SOCK_RAW	Właściwy kanał jest zainicjalizowany w trybie warstwy RAW
17	SOCK_UDP_ARP	Oczekiwanie na odpowiedź po przesłaniu pakietu ARP do miejsca docelowego dla transmisji UDP
18	SOCK_UDP_DATA	Trwa transmisja w trybie UDP RAW
19	SOCK_RAW_INIT	Układ W3100A zainicjalizowany w trybie MAC warstwy RAW

Zobacz także: **CONFIG TCP/IP** , **GETSOCKET** , **SOCKETCONNECT** , **SOCKETLISTEN** , **CLOSESOCKET** , **TCPWRITE** , **TCPWRITESTR** , **TCPREAD**

Przykład:

```
Tempw = SocketStat(i , 0)                      ' pobierz słowo statusu  
Select Case Tempw
```

```
Case Sock_established
End Select
```

TCPREAD() (Nowość w wersji 1.11.7.3)

Przeznaczenie:

Odczytuje dane z otwartego gniazda.

*Instrukcja korzysta z biblioteki TCPIP która jest rozprowadzana oddzielnie przez MCS Electronics. Wchodzi ona m.in. w skład pakietu **MCS Easy TCP/IP**.*

Składnia:

rezultat = **TCPREAD**(*gniazdo* , *zmienna* [, *bajtów*])

gdzie:

<i>rezultat</i>	zmienna typu Word gdzie funkcja umieści liczbę aktualnie odebranych bajtów z gniazda,
<i>gniazdo</i>	numer gniazda (0 – 3) za pomocą którego funkcja odbiera dane,
<i>zmienna</i>	nazwa zmiennej gdzie funkcja ma umieścić odebrane dane,
<i>bajtów</i>	ilość bajtów jak ma zostać odczytana. Ważna tylko dla zmiennych nie będących ciągami znaków.

Opis:

W przypadku gdzie jako parametr *zmienna* funkcji TCPREAD użyto zmiennej typu String procedura będzie czekać aż odebrany będzie znak końca linii (CR+LF). Nie ma zatem potrzeby podawania ilości odbieranych bajtów.

Uwaga! Należy pamiętać iż odebrany ciąg znaków będzie pozbawiony CR+LF.

Dla innych typów danych należy podać ile bajtów ma być odebranych. Nie jest jednak sprawdzane czy podana ilość bajtów jest większa niż rozmiar zmiennej. Dlatego odebranie 2 bajtów dla zmiennej typu Byte spowoduje nadpisanie następnego bajtu.

Dla zmiennych typu String funkcja nie powoduje nadpisania dalszych bajtów poza ciągiem. Przykładowo: ciąg znaków ma długość 10 bajtów a odbierany posiada ich 80 – odebrane zostanie tylko 10 znaków, a funkcja zakończy swoje działanie po odebraniu CR+LF.

Uwaga! Jeśli w buforze odbiorczym nie ma wystarczającej ilości wolnych bajtów funkcja zaczeka aż w buforze zwolni się miejsce lub gniazdo zostanie zamknięte.

Zobacz także: **CONFIG TCPIP** , **GETSOCKET** , **SOCKETCONNECT** , **SOCKETSTAT** , **TCPWRITE** , **TCPWRITESTR** , **CLOSESOCKET** , **SOCKETLISTEN**

Przykład:

Zobacz przykład dla funkcji **TCPWRITESTR**

TCPWRITE() (Nowość w wersji 1.11.7.3)

Przeznaczenie:

Przesyła dane za pomocą otwartego gniazda.

*Instrukcja korzysta z biblioteki TCPIP która jest rozprowadzana oddzielnie przez MCS Electronics. Wchodzi ona m.in. w skład pakietu **MCS Easy TCP/IP**.*

Składnia:

rezultat = **TCPWRITE**(*gniazdo* , *zmienna* [, *bajtów*])

rezultat = **TCPWRITE**(*gniazdo* , **EEPROM** , *adres* , *bajtów*)

gdzie:

<i>rezultat</i>	zmienna typu Word gdzie funkcja umieści liczbę aktualnie wysłanych bajtów. Jeśli w bufor jest pełen zwraca 0,
<i>gniazdo</i>	numer gniazda (0 – 3) za pomocą którego funkcja ma przesłać dane,
<i>zmienna bajtów</i>	nazwa zmiennej lub stała która ma zostać wysłana, ilość wysłanych bajtów. Ważna tylko dla zmiennych nie będących ciągami znaków i dla danych z pamięci EEPROM,
<i>adres</i>	adres w pamięci EEPROM gdzie umieszczono pierwszy wysyłany bajt.

Opis:

Pierwsza postać funkcji pozwala na wysłanie zawartości dowolnej zmiennej (oprócz zmiennych bitowych). Dla typu String nie podaje się ilości bajtów gdyż funkcja sama określa długość ciągu. Dla innych typów ilość bajtów należy obowiązkowo podać. Funkcja może także przesyłać zawartość tablic, wtedy należy podać także indeks wysyłanego elementu. Przykładowo: `zmienna(2)`.

Uwaga! Dla ciągów znaków funkcja nie dodaje znaków CR+LF.

Druga postać instrukcji pozwala na przesyłanie zawartości wewnętrznej pamięci EEPROM. Należy tylko podać adres pierwszego bajtu i ilość przesyłanych bajtów.

Jeśli w buforze nadawczym jest wystarczająca ilość wolnych bajtów funkcja zwraca wartość taką samą jak została podana.

Zobacz także: **CONFIG TCPIP** , **GETSOCKET** , **SOCKETCONNECT** , **SOCKETSTAT** , **TCPREAD** , **TCPWRITESTR** , **CLOSESOCKET** , **SOCKETLISTEN**

Przykład:

```
Tempw = Tcpwrite(i , "HTTP/1.0 200 OK{013}{010}")
```

TCPWRITESTR() *(Nowość w wersji 1.11.7.3)*

Przeznaczenie:

Przesyła dane tekstowe za pomocą otwartego gniazda.

*Instrukcja korzysta z biblioteki TCPIP która jest rozprowadzana oddzielnie przez MCS Electronics. Wchodzi ona m.in. w skład pakietu **MCS Easy TCP/IP**.*

Składnia:

rezultat = TCPWRITESTR(gniazdo , zmienna , opcje)

gdzie:

<i>rezultat</i>	zmienna typu Word gdzie funkcja umieści liczbę aktualnie wysłanych bajtów. Jeśli w bufor jest pełen zwraca 0,
<i>gniazdo</i>	numer gniazda (0 – 3) za pomocą którego funkcja ma przesłać dane,
<i>zmienna</i>	nazwa zmiennej typu String która ma zostać wysłana,
<i>opcje</i>	można podać 0 jeśli wysłana ma być zawartość zmiennej bez znaku końca linii; lub 255 by funkcja sama dodała CR+LF.

Opis:

Funkcja jest specjalnym wariantem poprzedniej funkcji **TCPWRITE**. Przeznaczona jest przede wszystkim do wysyłania ciągów znaków. W zależności od ostatniego parametru funkcja dodaje znaki CR+LF.

Zobacz także: **CONFIG TCPIP** , **GETSOCKET** , **SOCKETCONNECT** , **SOCKETSTAT** , **TCPREAD** ,

TCPWRITE , CLOSESOCKET , SOCKETLISTEN

Przykład:

```
'-----
'
'                               SMTP.BAS
'                               (c) 2003 MCS Electronics
'   Przykład pokazuje jak wysłać e-mail za pomocą protokołu SMTP
'-----

$regfile = "m16ldef.dat"      ' użyty procesor
$crystal = 4000000            ' użyty kwarc
$baud = 19200                 ' szybkość pracy UART
$lib "tcpip.lib"              ' używamy biblioteki dodatkowej tcpip.lib

'Stałe W3100A
Const Sock_stream = $01      ' Tcp
Const Sock_dgram = $02       ' Udp
Const Sock_ipl_raw = $03     ' Ip Layer Raw Sock
Const Sock_mac1_raw = $04    ' Mac Layer Raw Sock
Const Sel_control = 0        ' Confirm Socket Status
Const Sel_send = 1           ' Confirm Tx Free Buffer Size
Const Sel_recv = 2           ' Confirm Rx Data Size

'Status gniazda
Const Sock_closed = $00      ' Status Of Connection Closed
Const Sock_arp = $01         ' Status Of Arp
Const Sock_listen = $02      ' Status Of Waiting For Tcp Connection
Setup
Const Sock_synsent = $03     ' Status Of Setting Up Tcp Connection
Const Sock_synsent_ack = $04 ' Status Of Setting Up Tcp Connection
Const Sock_synrecv = $05     ' Status Of Setting Up Tcp Connection
Const Sock_established = $06 ' Status Of Tcp Connection Established
Const Sock_close_wait = $07  ' Status Of Closing Tcp Connection
Const Sock_last_ack = $08    ' Status Of Closing Tcp Connection
Const Sock_fin_wait1 = $09   ' Status Of Closing Tcp Connection
Const Sock_fin_wait2 = $0a   ' Status Of Closing Tcp Connection
Const Sock_closing = $0b     ' Status Of Closing Tcp Connection
Const Sock_time_wait = $0c   ' Status Of Closing Tcp Connection
Const Sock_reset = $0d       ' Status Of Closing Tcp Connection
Const Sock_init = $0e        ' Status Of Socket Initialization
Const Sock_udp = $0f         ' Status Of Udp
Const Sock_raw = $10         ' Status of IP RAW

Const Debug = -1              'czy wysyłać informacje o pracy
                               programu

#if Debug
  Print "Początek programu"
#endif

Enable Interrupts              ' włącz przerwania !
'określamy nr MAC, adres IP, maskę podsieci oraz domyślną bramę
'numer portu będzie użyty jeśli nie podano numeru portu przy tworzeniu
połączenia
'TX oraz RX konfiguruja 4 gniazda z buforami po 2 KB
Config Tcpiip = Int0 , Mac = 00.44.12.34.56.78 , Ip = 192.168.0.8 ,
Submask = 255.255.255.0 , Gateway = 192.168.0.1 , Localport = 1000 ,
Tx = $55 , Rx = $55
```

```

'użyte zmienne
Dim S As String * 50 , I As Byte , J As Byte , Tempw As Word
#If Debug
    Print "Zakończono ustawienie W3100A "
#EndIf

'najpierw otwieramy gniazdo
I = Getsocket(0 , Sock_stream , 5000 , 0)
'      ^ numer gniazda      ^ port
#If Debug
    Print "Gniazdo : " ; I
    'funkcja zwróci numer gniazda o który żądamy. Lub zwróci 255 gdy
    wystąpi błąd
#EndIf

If I = 0 Then                                     ' wszystko
w porządku
    'Połącz z serwerem SMTP
    J = Socketconnect(i , 194.000.000.002 , 25)      ' numer IP
serwera i port dla SMTP
    '      ^gniazdo
    '      ^ adres IP serwera
    '      ^ port 25
charakterystyczny dla SMTP
#If Debug
    Print "Połączenie : " ; J
    Print S_status(1)
#EndIf
If J = 0 Then                                     ' wszystko
ok
    #If Debug
        Print "Połączony!"
    #EndIf
    Do
        Tempw = Socketstat(i , 0)                  ' pobierz
status
        Select Case Tempw
            Case Sock_established                    '
nawiązano połączenie
                Tempw = Tcpread(i , S)                ' odczytaj
jedną linię
                #If Debug
                    Print S                            ' pokaż co
zwrócił serwer
                #EndIf
                If Left(s , 3) = "220" Then            ' ok
                    Tempw = Tcpwrite(i , "HELO xxxxxx{013}{010}" )
' nazwa użytkownika
                #If Debug
                    Print "Wysłano" ; Tempw ; " bajtów " ' liczba
bajtów wysłanych
                #EndIf
                Tempw = Tcpread(i , S)                ' odbierz
odpowiedź
                #If Debug
                    Print S                            ' pokaż
                #EndIf
                If Left(s , 3) = "250" Then            ' ok
                    Tempw = Tcpwrite(i , "MAIL
FROM:<tcpip@mcselec.com>{013}{010}")

```

```

        ' wyślij adres nadawcy
        Tempw = Tcpread(i , S)                                ' odbierz
odpowiedź
        #if Debug
        Print S
        #endif
        If Left(s , 3) = "250" Then                            ' ok
            Tempw = Tcpwrite(i , "RCPT
TO:<tcpip@mcselec.com>{013}{010}")
        ' wyślij adres odbiorcy
            Tempw = Tcpread(i , S)                                ' odbierz
odpowiedź
        #if Debug
        Print S
        #endif
        If Left(s , 3) = "250" Then                            ' ok
            Tempw = Tcpwrite(i , "DATA{013}{010}")
        ' teraz informujemy serwer że wysyłamy dane
            Tempw = Tcpread(i , S)                                ' odbierz
odpowiedź
        #if Debug
        Print S
        #endif
        If Left(s , 3) = "354" Then                            ' ok
            Tempw = Tcpwrite(i , "From:
tcpip@mcselec.com{013}{010}")
            Tempw = Tcpwrite(i , "To:
tcpip@mcselec.com{013}{010}")
            Tempw = Tcpwrite(i , "Subject: BASCOM
SMTP test{013}{010}")
            Tempw = Tcpwrite(i , "X-Mailer: BASCOM
SMTP{013}{010}")
            Tempw = Tcpwrite(i , "{013}{010}")
            Tempw = Tcpwrite(i , "To jest testowy e-
mail przesłany przez BASCOM SMTP{013}{010}")
            Tempw = Tcpwrite(i , "Dodaj więcej linii
jeśli trzeba {013}{010}")
            Tempw = Tcpwrite(i , ".{013}{010}")
        ' zakończ za pomocą kropki
            Tempw = Tcpread(i , S)                                ' odbierz
odpowiedź
        #if Debug
        Print S
        #endif
        If Left(s , 3) = "250" Then                            ' ok
            Tempw = Tcpwrite(i ,
"QUIT{013}{010}")        ' koniec połączenia
            Tempw = Tcpread(i , S)
            #if Debug
            Print S
            #endif
        End If
    End If
End If
End If
End If
Case Sock_close_wait
    Print "odebrano CLOSE_WAIT"
    Closesocket I                                                ' zamknij
jeśli serwer zażądał

```

```

                Case Sock_closed
                    Print "odebrano SOCKET_CLOSED"           ' gniazdo
                    zostało zamknięte
                End
            End Select
        Loop
    End If
End If
End                                     'i finał

```

UDPREAD() (Nowość w wersji 1.11.7.3)

Przeznaczenie:

Odczytuje dane za pomocą protokołu UDP.

Instrukcja korzysta z biblioteki TCPIP która jest rozprowadzana oddzielnie przez MCS Electronics. Wchodzi ona m.in. w skład pakietu **MCS Easy TCP/IP**.

Składnia:

rezultat = UDPREAD(gniazdo , zmienna [, bajtów])

gdzie:

rezultat	zmienna typu Word gdzie funkcja umieści liczbę aktualnie odebranych bajtów z gniazda,
gniazdo	numer gniazda (0 – 3) za pomocą którego funkcja odbiera dane,
zmienna	nazwa zmiennej gdzie funkcja ma umieścić odebrane dane,
bajtów	ilość bajtów jak ma zostać odczytana. Ważna tylko dla zmiennych nie będących ciągami znaków.

Opis:

W przypadku gdzie jako parametr *zmienna* funkcji UDPREAD użyto zmiennej typu String procedura będzie czekać aż odebrany będzie znak końca linii (CR+LF). Nie ma zatem potrzeby podawania ilości odbieranych bajtów.

Uwaga! Należy pamiętać iż odebrany ciąg znaków będzie pozbawiony CR+LF.

Dla innych typów danych należy podać ile bajtów ma być odebranych. Nie jest jednak sprawdzane czy podana ilość bajtów jest większa niż rozmiar zmiennej. Dlatego odebranie 2 bajtów dla zmiennej typu Byte spowoduje nadpisanie następnego bajtu.

Dla zmiennych typu String funkcja nie powoduje nadpisania dalszych bajtów poza ciągiem. Przykładowo: ciąg znaków ma długość 10 bajtów a odbierany posiada ich 80 – odebrane zostanie tylko 10 znaków, a funkcja zakończy swoje działanie po odebraniu CR+LF.

Uwaga! Jeśli w buforze odbiorczym nie ma wystarczającej ilości wolnych bajtów funkcja zaczeka aż w buforze zwolni się miejsce lub gniazdo zostanie zamknięte.

Funkcja **SOCKETSTAT** w przypadku protokołu UDP zwraca zawsze wartość o 8 bajtów większą. Spowodowane jest to tym, że protokół UDP zawsze dodaje nagłówek, który zawiera m.in. długość pakietu danych, adres IP i numer portu równorzędnego komputera (połączenie jest typu peer-to-peer).

Funkcja UDPREAD rozkodowywuje nagłówek i poszczególne jego składowe umieszcza w zmiennych: PeerSize, PeerAdress, PeerPort.

Uwaga! Powyższe zmienne należy wcześniej zadeklarować w programie dokładnie w ten sposób:

```
Dim PeerSize As Integer , PeerAdress As Long , PeerPort As Word
```

Zobacz także: CONFIG TCPIP , GETSOCKET , SOCKETCONNECT , SOCKETSTAT , TCPWRITE , TCPWRITESTR , CLOSESOCKET , SOCKETLISTEN , UDPWRITE , UDPWRITESTR ,

Przykład:

```
Result = Socketstat(idx , Sel_recv)      'pobierz liczbę oczekujących
bajtów
If Result > 0 Then
    Print "Ilość oczekujących bajtów : " ; Result
    Temp2 = Result - 8                    '8 pierwszych bajtów zawsze
zawiera nagłówek UDP                      'zawierający długość, adres IP
i numer portu
    Temp = Udpread(idx , S(1) , Result)   ' odczytaj rezultat
    For Temp = 1 To Temp2
        Print S(temp) ; " " ;             ' i wydrukuj
    Next
End If
```

UDPWRITE() (Nowość w wersji 1.11.7.3)

Przeznaczenie:

Przesyła dane za pomocą gniazda protokołem UDP.

Instrukcja korzysta z biblioteki TCPIP która jest rozprowadzana oddzielnie przez MCS Electronics. Wchodzi ona m.in. w skład pakietu **MCS Easy TCP/IP**.

Składnia:

rezultat = UDPWRITE(IP , port , gniazdo , zmienna [, bajtów])

rezultat = UDPWRITE(IP , port , gniazdo , EEPROM , adres , bajtów)

gdzie:

rezultat	zmienna typu Word gdzie funkcja umieści liczbę aktualnie wysłanych bajtów. Jeśli w bufor jest pełen zwraca 0,
IP	adres IP gdzie dane mają być przesłane. Można podać w postaci standardowej (4 wartości oddzielone kropkami) lub jako zmienną typu Long, gdzie część MSB zmiennej zawiera część LSB adresu IP,
port	numer portu do którego przesłane mają być dane,
gniazdo	numer gniazda (0 – 3) za pomocą którego funkcja ma przesyłać dane,
zmienna bajtów	nazwa zmiennej lub stała która ma zostać wysłana, ilość wysyłanych bajtów. Ważna tylko dla zmiennych nie będących ciągami znaków i dla danych z pamięci EEPROM,
adres	adres w pamięci EEPROM gdzie umieszczono pierwszy wysyłany bajt.

Opis:

Pierwsza postać funkcji pozwala na wysłanie zawartości dowolnej zmiennej (oprócz zmiennych bitowych). Dla typu String nie podaje się ilości bajtów gdyż funkcja sama określa długość ciągu. Dla innych typów ilość bajtów należy obowiązkowo podać. Funkcja może także przysyłać zawartość tablic, wtedy należy podać także indeks wysyłanego elementu. Przykładowo: zmienna(2).

Druga postać instrukcji pozwala na przesyłanie zawartości wewnętrznej pamięci EEPROM. Należy tylko podać adres pierwszego bajtu i ilość przesyłanych bajtów.

Jeśli w buforze nadawczym jest wystarczająca ilość wolnych bajtów funkcja zwraca wartość

taką samą jak została podana.

Uwaga! Funkcja UDPWRITE jest prawie taka sama jak TCPWRITE. Ponieważ UDP jest protokołem „bezpółłączeniowym”, należy podać adres IP i numer portu. UDP wymaga tylko otwartego gniazda.

Zobacz także: CONFIG TCPIP , GETSOCKET , SOCKETCONNECT , SOCKETSTAT , TCPREAD , TCPWRITESTR , CLOSESOCKET , SOCKETLISTEN , UDPREAD , UDPWRITESTR

Przykład:

Zobacz przykład dla funkcji UDPWRITESTR

UDPWRITESTR() (Nowość w wersji 1.11.7.3)

Przeznaczenie:

Przesyła dane tekstowe za pomocą protokołu UDP.

Instrukcja korzysta z biblioteki TCPIP która jest rozprowadzana oddzielnie przez MCS Electronics. Wchodzi ona m.in. w skład pakietu MCS Easy TCP/IP.

Składnia:

rezultat = UDPWRITESTR(IP , port , gniazdo , zmienna , opcje)

gdzie:

rezultat	zmienna typu Word gdzie funkcja umieści liczbę aktualnie wysłanych bajtów. Jeśli w bufor jest pełen zwraca 0,
gniazdo	numer gniazda (0 – 3) za pomocą którego funkcja ma przesłać dane,
IP	adres IP gdzie dane mają być przesłane. Można podać w postaci standardowej (4 wartości oddzielone kropkami) lub jako zmienną typu Long, gdzie część MSB zmiennej zawiera część LSB adresu IP,
port	numer portu do którego przesłane mają być dane,
zmienna	nazwa zmiennej typu String która ma zostać wysłana,
opcje	można podać 0 jeśli wysłana ma być zawartość zmiennej bez znaku końca linii; lub 255 by funkcja sama dodała CR+LF.

Opis:

Funkcja jest specjalnym wariantem poprzedniej funkcji UDPWRITE. Przeznaczona jest przede wszystkim do wysyłania ciągów znaków. W zależności od ostatniego parametru funkcja dodaje znaki CR+LF.

Zobacz także: CONFIG TCPIP , GETSOCKET , SOCKETCONNECT , SOCKETSTAT , TCPREAD , TCPWRITESTR , CLOSESOCKET , SOCKETLISTEN , UDPREAD , UDPWRITE

Przykład:

```
'-----
'
'                               UDPTTEST.BAS
'                               (c) 2002-2003 MCS Electronics
' Uruchom program easytcp.exe po zaprogramowaniu procesora i naciśnij
' przycisk UDP
'-----
'-----

$regfile = "m16ldef.dat"      ' użyty procesor
$crystal = 4000000            ' użyty kwarc
$baud    = 19200              ' szybkość pracy UART
```

```

Const Sock_stream = $01          ' Tcp
Const Sock_dgram = $02          ' Udp
Const Sock_ipl_raw = $03        ' Ip Layer Raw Sock
Const Sock_mac1_raw = $04       ' Mac Layer Raw Sock
Const Sel_control = 0           ' Confirm Socket Status
Const Sel_send = 1              ' Confirm Tx Free Buffer Size
Const Sel_recv = 2              ' Confirm Rx Data Size

'Status gniazda
Const Sock_closed = $00         ' Status Of Connection Closed
Const Sock_arp = $01            ' Status Of Arp
Const Sock_listen = $02        ' Status Of Waiting For Tcp Connection
Setup
Const Sock_synsent = $03        ' Status Of Setting Up Tcp Connection
Const Sock_synsent_ack = $04    ' Status Of Setting Up Tcp Connection
Const Sock_synrecv = $05        ' Status Of Setting Up Tcp Connection
Const Sock_established = $06    ' Status Of Tcp Connection Established
Const Sock_close_wait = $07     ' Status Of Closing Tcp Connection
Const Sock_last_ack = $08       ' Status Of Closing Tcp Connection
Const Sock_fin_wait1 = $09      ' Status Of Closing Tcp Connection
Const Sock_fin_wait2 = $0a      ' Status Of Closing Tcp Connection
Const Sock_closing = $0b        ' Status Of Closing Tcp Connection
Const Sock_time_wait = $0c      ' Status Of Closing Tcp Connection
Const Sock_reset = $0d          ' Status Of Closing Tcp Connection
Const Sock_init = $0e           ' Status Of Socket Initialization
Const Sock_udp = $0f            ' Status Of Udp
Const Sock_raw = $10            ' Status of IP RAW

$lib "tcpip.lib"                 ' używamy biblioteki dodatkowej
tcpip.lib
Print "Inicjalizacja, ustaw IP na 192.168.0.8"
Enable Interrupts                ' włącz przerwania !
Config Tcpiip = Int0 , Mac = 12. 128.12.34.56.78 , Ip = 192.168.0.8 ,
Submask = 255.255.255.0 , Gateway = 0.0.0.0 , Localport = 1000 , Tx =
$55 , Rx = $55

'użyj poniższej wersji gdy używana jest brama domyślna
'Config Tcpiip = Int0 , Mac = 12. 128.12.34.56.78 , Ip = 192.168.0.8 ,
Submask = 255.255.255.0 , Gateway = 192.168.0.1 , Localport = 1000 ,
Tx = $55 , Rx = $55

Dim Idx As Byte                  ' numer gniazda
Dim Result As Word               ' rezultat
Dim S(80) As Byte
Dim Sstr As String * 20
Dim Temp As Byte , Temp2 As Byte ' bajty tymczasowe
'-----
'-----
'gdy używany jest UDP należy zdefiniować poniższe zmienne dokładnie
tak jak tutaj!!
Dim Peersize As Integer , Peeraddress As Long , Peerport As Word
'-----
-
Declare Function Ipnum(ip As Long) As String ' a handy function

'like with TCP, we need to get a socket first
'note that for UDP we specify sock_dgram
Idx = Getsocket(idx , Sock_dgram , 5000 , 0) ' get socket for
UDP mode, specify port 5000
Print "Socket " ; Idx ; " " ; Idx

```

'UDP to protokół bezpołączeniowy co oznacza, że nie można nasłuchiwać, łączyć się lub odczytywać statusu
 'Można tylko nadawać lub odbierać dane tak w ten sam sposób jak dla TCP /IP.
 'Lecz ponieważ nie ma protokołu łączenia, należy podać adres IP i port komputera docelowego
 'Porównując do funkcji dla TCP/IP można wysyłać dokładnie tak samo, lecz z dodatkowo określonym adresem IP i numerem portu

```

Do
  Temp = Inkey()                ' czekaj na znak
  If Temp = 27 Then              ' czy klawisz ESC
    Sstr = "Witaj"
    Result = Udpwritestr(192.168.0.3 , 5000 , Idx , Sstr , 255)
  End If
  Result = Socketstat(idx , Sel_recv) 'pobierz liczbę
oczekujących bajtów
  If Result > 0 Then
    Print "Ilość oczekujących bajtów : " ; Result
    Temp2 = Result - 8           '8 pierwszych bajtów zawsze
zawiera nagłówek UDP
                                'zawierający długość, adres IP
i numer portu
    Temp = Udpread(idx , S(1) , Result) ' odczytaj rezultat
    For Temp = 1 To Temp2
      Print S(temp) ; " " ;      ' i wydrukuj
    Next
    Print
    Print Peersize ; " " ; Peeraddress ; " " ; Peerport
zmienne te są wypełniane przez UDPREAD
    Print Ipnum(peeraddress)      ' wydrukuj numer IP w
przyjazny sposób
    Result = Udpwrite(192.168.0.3 , Peerport , Idx , S(1) , Temp2)
    ' prześlij odebrane dane z powrotem
  End If
Loop

```

'Przykład powyższy oczekuje na dane i wysyła je z powrotem dlatego też temp2 jest zmniejszana o 8, czyli rozmiar bufora

'Funkcja ta może być używana do wyświetlania adresu IP w normalny sposób

```

Function Ipnum(ip As Long) As String
  Local T As Byte , J As Byte
  Ipnum = ""
  For J = 1 To 4
    T = Ip And 255
    Ipnum = Ipnum + Str(t)
    If J < 4 Then Ipnum = Ipnum + "."
    Shift Ip , Right , 8
  Next
End Function

End

```