Podstawy programowania w BASCOM BASIC AVR

Tworzenie programu

1. Uruchamiamy aplikację BASCOM AVR.
2. Tworzymy plik programu lub otwieramy i ewentualnie edytujemy istniejący.
3. Sprawdzamy, czy ustawienia konfiguracji są zgodne z założonymi.
4. Zapisujemy plik programu.
5. Dokonujemy kompilacji.
6. Jeśli wystąpiły jakieś błędy, poprawiamy je zapisujemy i ponownie kompilujemy program.
7. Uruchamiamy symulację.
8. Jeżeli program działa niezgodnie z oczekiwaniami poprawiamy go i powtarzamy operację kompilacji i symulacji.
9. Programujemy układ mikrokontrolera i testujemy w realizowanym urządzeniu.

Zestaw znaków

- duże litery od A do Z oraz ich odpowiedniki w zestawie małych liter oraz znak _.
- liczby 0 do 9 oraz przy zapisie szesnastkowym litery od A do H
  (zapis szesnastkowy powinien być poprzedzony przedrostkiem &H, a bitowy przedrostkiem &B).

Znaki o szczególnym znaczeniu

spacja - znak rozdzielający
: - rozdziela instrukcje zapisane w jednej linii
, - znak rozdzielający argumenty instrukcji
; - rozdziela argumenty instrukcji wejścia/wyjścia
' - znak początku komentarza
" - rozpoczyna i kończy dane tekstowe
+ - znak operacji dodawania
- - znak operacji odejmowania
* - znak operacji mnożenia
/ - znak operacji dzielenia
\ - znak dzielenia dla liczb całkowitych
^ - znak operacji potęgowania
. - oddziela część całkowitą od ułamkowej
= - występuje w operacjach przypisania oraz porównywania
< - znak operacji porównywania (mniejszy)
> - znak operacji porównywania (większy)

Struktura pojedynczej linii programu

Etykieta: Instrukcja1: Instrukcja2: ... : Instrukcja ostatnia 'Komentarz
(oczywiście nie wszystkie składniki muszą występować jednocześnie).

- **Etykieta - z**aczyna się jako pierwszy znak linii, musi zaczynać się od litery (1 do 32 znaków) i nie może być słowem zastrzeżonym (żadną z instrukcji języka BASCOM BASIC).
- **Komentarz** - rozpoczyna się znakiem ' lub słowem REM (traktowanym jako instrukcja języka BASCOM BASIC). Mogą zawierać dowolne znaki. Komentarze są bardzo wskazane, celem opisu operacji, podprogramów i procedur zawartych w programie.

Instrukcje języka BASCOM BASIC

- Instrukcje "wykonywalne" tworzą logiczną strukturę działania programu (są odzwierciedleniem algorytmu programu).
- Instrukcje "nie wykonywalne" są odpowiedzialne za cześć informacyjną oraz organizacyjną programu:
  - instrukcje komentarza,
  - instrukcje deklaracji (DIM, DECLARE) ,
  - dyrektywy kompilatora, rozpoczynające się znakiem $.

Typy danych

Bit	1/8 bajta	Może przyjmować wartość 0 lub 1
Byte	1 bajt	Może przechowywać wartość binarna od 0 do 255
Word	2 bajty	Może przechowywać wartość w zakresie 0 – 65535
Integer	2 bajty	Może przechowywać dowolną liczbę całkowitą z zakresu –32768 do +32767.
Long	4 bajty	Może przechowywać dowolną liczbę całkowitą z zakresu -2^32 do 2^32-1
Single	4 bajty	Może przechowywać dowolną liczbę stało lub zmiennoprzecinkową pojedynczej precyzji
Double	8 bajtów	Może przechowywać dowolną liczbę stało lub zmiennoprzecinkową podwójnej precyzji
String	max. 254 bajty	Może przechowywać dowolny ciąg znaków o długości nie większej niż 254 znaki

Zmienne

Zmienna to określony obszar pamięci identyfikowany przez nadaną mu nazwę. Wielkość tego obszaru zależy od zmiennej (patrz tabela wyżej). W zmiennych numerycznych można przechowywać tylko dane będące liczbami. W zmiennych typu String tylko dane będące znakami lub ciągiem znaków (ciągiem liczb odpowiadających kodom liter wg ASCII). Nazwa zmiennej musi zaczynać się od litery (max 32 znaki). Nie może być słowem zastrzeżonym.

Przypisanie zmiennej wartości liczbowej lub tekstowej:

```
B = 1.1             'zmienna C jest typu Single
C = "BASCOM"        'zmienna D jest typu String
s = "8051"
ALFA = &H1af4
Delta = &B1010
```

Przypisanie zmiennej wartości innej zmiennej:

```
Abc = Def
K = G
```

Przypisanie zmiennej wyniku operacji matematycznej:

```
Temp = A + 5
```

Przypisanie zmiennej wyniku działania funkcji:

```
Temp = Asc(s)       'zmienna s jest typu String o długości jednego znaku
```

Definiowanie zmiennych

Aby używać zmiennej w programie należy ją najpierw zdefiniować instrukcją DIM. Jest to wymagane, gdyż kompilator musi określić rozmiar pamięci przeznaczony na każdą ze zmiennych. Zmienne typu String muszą mieć dodatkowy parametr określający przewidywaną długość w znakach.

```
Dim B1 As Bit, I As Integer, K As Byte , S As String * 10
```

Można także użyć specjalnych instrukcji z zestawu: DEFINT, DEFBIT, DEFBYTE, DEFWORD, DEFLNG lub DEFSNG. Instrukcje te odgórnie 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.

Kwalifikatory

Do każdej zmiennej numerycznej (oprócz typu Bit) można stosować kwalifikatory - odnośniki do konkretnych bitów zmiennych. Przykład dla zmiennej typu Byte:

```
A = 7               ’bitowo: &b00000111
Print A.0           'wydrukuje 1
Print A.3           'wydrukuje 0
```

Liczby zmiennoprzecinkowe

Liczby zmiennoprzecinkowe można przechowywać jedynie jako typu 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 (cechę) i 23 bitową mantysę:

```
31 30_________23 22_____________________________0
zn    eksponent                  mantysa
```

Bit znaku określa znak liczby przechowywanej w zmiennej. Wartość 0 oznacza liczbę dodatnią, 1 zaś ujemną. Eksponent jest zapisany jako liczba 8 bitowa ze znakiem w kodzie U2. Jeśli najbardziej znaczący bit eksponentu (zwanego też cechą) jest ustawiony (eksponent > 128) to eksponent jest traktowany jako ujemny. Mantysa jest przechowywana w znormalizowanym formacie “ukrytego bitu”.
Wszystkie operacje matematyczne mogą być przeprowadzane wyłącznie 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
```

Ciągi

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

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

Rozkaz powyższy 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{kod}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 elementów. Każdy element tablicy posiada swój własny indeks (numer) jednoznacznie go identyfikujący. Zmiana dowolnego elementu tablicy nie wpływa zatem na zmianę pozostałych 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.

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

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

Wyrażenia i operatory

Wyróżniamy:

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

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

```
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

+	- dodawanie,
-	- odejmowanie,
*	- mnożenie,
\	- dzielenie całkowite,
Mod	- reszta z dzielenia modulo 2,
/	- dzielenie zmiennoprzecinkowe,
^	- potęgowanie.

```
A = 5 / 2
Print A
```

Wynikiem działania tego programu jest liczba 2,5.

```
A = 5 \ 2
Print A
```

Wynikiem działania tego programu jest liczba 2.

```
A = 5 Mod 2
Print A
```

Wynikiem działania tego programu jest 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

=	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 logiczna prawdę jeśli wyrażenie jest prawdziwe lub fałsz jeśli wyrażenie jest fałszywe.

Operatory logiczne

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

```
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
```

Innym sposobem zmieniania stanu bitów zmiennej jest skorzystanie z wspomnianych wcześniej kwalifikatorów i operatora logicznego NOT:

```
A = &B0110
(...)
A.1 = Not A.1
Print A
```

Konwersja typów (jawna i niejawna)

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

```
Long = Long1 * Long2
```

gdzie wszystkie składniki wyrażenia są typu Long.

Typy zmiennych występujące w 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 AND 255 = 0.
```

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
B = "A"     'B jest kodem litery A, czyli B = 65
```

Jeśli zmienna docelowa 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
```

Gdy wystąpi potrzeba zamiany wartości typu Single na typ Byte, Integer, Word bądź Long, wtedy kompilator automatycznie zamieni podana liczbę jeśli tylko będzie ona typu Single (będzie ułamkowa)

```
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
```

Łączenie kodu asemblera z instrukcjami BASCOM BASIC

Fragment w języku asemblera możemy umieścić w programie 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
```

Język BASCOM BASIC pozwala na wplatanie w treść programu bezpośrednio bloków kodu napisanych w asemblerze. Prawie wszystkie mnemoniki są rozpoznawane przez kompilator automatycznie. 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).

```
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 przykładnie wykorzystano instrukcję SWAP poprzedzoną znakiem ! .

Operacje na portach i pojedynczych pinach procesora

W przypadku mikrokontrolerów AVR porty nazywane są A,B,C,D,E,F i posiadają 3 rejestry związane z każdym portem. Port zwykle składa się z ośmiu końcówek. Wszystkie mikrokontrolery posiadają co najmniej PORTB. Jeśli chcemy ustawić jedno z wyjść portu w stan niski bądź wysoki to musimy użyć instrukcji RESET lub SET. Przedtem jednak musimy skonfigurować to wyjście. Do tego celu służą rejestry DDRx (dla każdego portu oddzielny rejestr). Jeśli na odpowiednim bicie w tym rejestrze wpiszemy 0 wtedy skojarzona z nim końcówka będzie pełnić rolę wejścia. Jeżeli wpiszemy 1, odpowiednia końcówka będzie pracować jako wyjście. Po ustawieniu kierunku pracy końcówki, aby ustawiać stan na końcówkach należy użyć rejestru PORTx. Odczytując rzeczywisty stan końcówki, należy użyć rejestru PINx.

Uwaga!
Odczytanie jakichkolwiek danych z portu lub z pojedynczego pinu takiego portu jest możliwe dopiero po programowym ustawieniu na całym porcie lub wybranym pinie stanu wysokiego. Wyprowadzenia portów mikroprocesora możemy z pewnym przybliżeniem traktować jako wyjścia typu OPEN DRAIN wyposażone w bufory, w których zatrzaskuje się podana z wewnątrz informacja. Jeżeli np. po podaniu polecenia: PORTC=0 chcielibyśmy odczytać z wejść tego portu jakiekolwiek dane, to niezależnie od stanów logicznych na dołączonych do nich układach peryferyjnych, zawsze odczytywalibyśmy same zera. Dopiero po wydaniu polecenia: PORTC= 255 uzyskujemy dostęp do wejść portu C.

Ustawianie:

```
DDRB = &B11110000            'cztery starsze jako wyjścia,
                             'cztery młodsze jako wejścia
Set PORTB.7                  'ustaw PB.7 w stan wysoki
Reset PORTB.6                'ustaw PB.6 w stan niski
PORTB = 0                    'zapis 0 do rejestru, spowoduje ustawienie
                             'wszystkich końcówek w stan niski
```

Odczyt:

```
PORTB = 255
Print PINB.0                 'wyślij przez RS-232 odczytany stan
                             'końcówki 0 portu B
A = PINB                     'przypisz stan portu do zmiennej A
Print PINB                   'wyślij przez RS-232 stany panujące na
                             ‘końcówkach portu B
```