Basic Stamp "Elektroniczny Znaczek", część 2

Opis sprzętu i języka

Kontynuujemy prezentację tajników języka programowania "Basic Stampa". Omówienie to daje niezbędne podstawy do efektywnego korzystania z możliwości oferowanych przez ten doskonały mikrokontroler.

Poniżej szczegółowo opiszemy instrukcje języka PBasic. Zostały one posegregowane alfabetycznie.

Nawiasy klamrowe {} w składni instrukcji oznaczają parametry opcjonalne, czyli takie, które nie muszą wystąpić.

BRANCH

Składnia:

```
BRANCH przesunięcie, (adres0,
adres1,... adresN)
```

Skok warunkowy do jednego z adresów określonego przez wartość argumentu przesunięcie. W innych odmianach języka Basic odpowiada od instrukcji ON x GOTO. BRANCH odpowiada sekwencji instrukcji warunkowych:

```
if przesuniecie=0 then adres0
if przesuniecie=1 then adres1
                  . . .
if przesuniecie=N then adresN
```

Jeśli wartość przesunięcie wykracza poza liczbę zdefiniowanych etykiet skoku, a skok nie jest wykonywany i program przechodzi do wykonywania instrukcji następującej po BRANCH.

BUTTON

Składnia:

```
BUTTON pin, stan_nacisniecia,
opoznienie, przerwa, zmienna,
stan_skoku, adres
```

Instrukcja obsługi włącznika podłączonego do jednej z linii portu P. Instrukcja BUTTON symuluje działanie klawisza w podobny sposób, z jakim mamy do czynienia w komputerze PC. Dłuższe przytrzymanie naciśniętego klawisza powoduje wygenerowanie serii znaków. Znaczenie argumentów instrukcji BUTTON jest następujące:

I. **pin** - określenie nóżki portu P, do której został podłączony przycisk. Może to być zmienna albo stała, która reprezentuje liczbę całkowitą z zakresu 0..7;
II. **stan_nacisniecia** - określa poziom logiczny stanu naciśnięcia przycisku. Może to być wysoki (1) albo niski (0) stan naciśnięcia (**rys. 3**);
III. **opoznienie** - zmienna albo stała reprezentująca liczbę z zakresu 0..255, która określa jak długo przycisk musi być naciśnięty, zanim rozpocznie się proces samopowtarzania instrukcji BUTTON. Liczba ta wyraża ilość powtórzeń sprawdzania stanu przycisku. Opóźnienie to ma dwie skrajne wartości: 0 i 255. Wartość 0 oznacza brak anulowania drgań zestyków lub tylko samopowtarzanie. Wartość 255 oznacza z kolei, że nie będzie procesu samopowtarzania;
IV. **przerwa** - zmienna albo stała reprezentująca liczbę z zakresu, która określa jak długo trwa przerwa między powtarzaniem instrukcji BUTTON, gdy jest ona w trybie samopowtarzania. Jest to liczba powtórzeń wykonania procedury BUTTON;
V. **zmienna** - adres komórki pamięci przeznaczony dla procedury BUTTON. Przed jej wywołaniem zmienna ta musi być zerowana;
VI. **stan_skoku** - jest to stan logiczny, który powoduje wykonanie skoku do adresu zdefiniowanego w parametrze adres. Zero logiczne oznacza zwolnienie klawisza, zaś jedynka logiczna skok po naciśnięciu klawisza. Takie podejście oznacza, że niezależnie od aktywnego stanu przycisku, o skoku decyduje położenie przycisku (wciśnięty-zwolniony).
VII. **adres** - adres, do którego następuje skok, gdy zostanie wykryty prawidłowy stan_skoku.

Instrukcja BUTTON anuluje zakłócenia, jakie zawsze powstają na mechanicznych zestykach klawiatury. Wykonuje to automatycznie, zwalniając z tego obowiązku programistę.

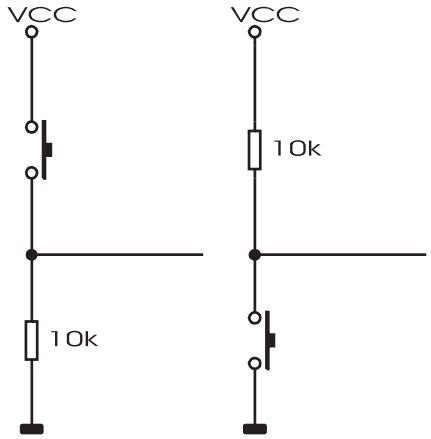
Instrukcja BUTTON pozwala na umieszczenie jej w pętli programowej. Do tego celu są wykorzystywane jej dwa ostatnie parametry. Jeśli BUTTON wykrywa stan_nacisniecia, to może zostać wykonany skok do adresu adres (stan_skoku=1) albo nie (stan_skoku=0). Jeśli przycisk pozostaje ciągle wciśnięty, BUTTON wykonuje określoną liczbę sprawdzeń jego stanu. Jeśli ta liczba zrówna się z zadeklarowaną w parametrze opoznienie, BUTTON ponownie sprawdza czy istnieje możliwość skoku do adresu zdefiniowanego w parametrze adres. Kiedy takiej możliwości nie ma, BUTTON odczekuje w pętli tyle cykli, ile zostało określonych w parametrze przerwa. Na tym kończy się działanie instrukcji BUTTON.

Przykładem działania instrukcji BUTTON niech będzie poniższy programik.

```
b3=0
petla:
button 4,0,240,120,b3,0,przeskocz
toggle 1
przeskocz:
goto petla
```

Do linii P4 dołączono włącznik monostabilny w taki sposób, jak na rys.3a. Musimy rozpatrzyć kilka możliwych sytuacji, które mogą wystąpić.

Jak napisano wyżej, zmienna, która będzie używana przez BUTTON, musi być wstępnie wyzerowana. Pierwszy argument instrukcji BUTTON oznacza, że będzie testowana nóżka P4 portu I/O. Z drugim argumentem jest związany stan aktywny przy-



Rys. 3 Dwa sposoby podłączenia przycisku do linii we/wy układu Basic Stamp:
a) stan aktywny wysoki (stan naciśnięcia = 1),
b) stan aktywny niski (stan naciśnięcia = 0)

cisku, w naszym przypadku jest to stan niski. Argument trzeci określa, że 240 razy zostanie sprawdzony stan nóżki P4, i jeśli w tym czasie nie zostanie wykryte zwolnienie przycisku, to nastąpi przejście do odliczania opóźnienia równego 120 pętlom sprawdzenia stanu nóżki P4 (argument czwarty). Zwolnienie przycisku w czasie odliczania 240 pętli czy 120 pętli potrzebnych do samopowtarzania kończy działanie instrukcji i przejście do instrukcji następnej, czyli do TOGGLE 1. TOGGLE 1 oznacza inwersję stanu nóżki P1 portu I/O.

Rozpatrzmy przypadek, gdy przycisk jest zwolniony. Instrukcja BUTTON ma zapisane w swoim szóstym argumencie, że wykonuje skok, gdy przycisk jest zwolniony. Wykrycie tego faktu nastąpi na początku. W naszym programie skok do etykiety przeskocz jest równoznaczny z ominięciem instrukcji TOGGLE, czyli tak naprawdę z tworzeniem martwej pętli, która nic nie wykonuje.

Teraz wyobraźmy sobie, że przycisk został wciśnięty na stałe. Po 240 pętlach zostanie wykryty stan wciśnięcia przycisku, czyli nie ma podstaw do wykonania skoku do etykiety przeskocz i będzie odliczane z kolei 120 pętli. Potem znów będzie sprawdzona zgodność stanu przycisku ze względu na wykonanie skoku do powyższej etykiety. Oczywiście nie jest to możliwe, ponieważ piąty argument jest ustawiony na zwolnienie przycisku. Po 120 pętlach instrukcja BUTTON zostanie zakończona, potem będzie wykonana instrukcja TOGGLE i z powrotem zostanie wykonany skok do instrukcji BUTTON. Nasz prosty programik wykonuje przełączanie stanu diody LED dołączonej do P1.

Po tym przykładzie należy zauważyć, że ważną właściwością instrukcji BUTTON jest jej dynamiczny charakter. Instrukcja ta **nigdy nie zatrzymuje** działania programu w oczekiwaniu na naciśnięcie klawisza. Nie jest ona równoważna procedurze odczytu klawiatury, choć niewątpliwie jest ona bardzo pomocna w jej tworzeniu. Instrukcja BUTTON wykrywa fakt naciśnięcia przycisku i reaguje nań stosownie do czasu jego naciśnięcia.

DEBUG

Składnia:

```
DEBUG zmienna{, zmienna}
```

Instrukcja ta jest używana w trybie uruchamiania, gdy Basic Stamp jest połączony z komputerem PC. Umożliwia ona przesłanie do PC stanu wybranej grupy zmiennych. Argumentem tej funkcji jest bit, bajt albo słowo. Dopuszczalny format zapisu tych zmiennych jest następujący:

I. **bez modyfikatorów** - wyświetli "zmienna=" i tu poda wartość zmiennej;
II. **#** przed nazwą zmiennej - wartość zmiennej zostanie podana dziesiętnie, bez dodatkowego komentarza, jak to miało miejsce wyżej;
III. **$** przed nazwą zmiennej - wartość zmiennej zostanie podana szesnastkowo;
IV **%** przed nazwą zmiennej - wartość zmiennej zostanie podana binarnie;
V. **@** przed nazwą zmiennej - podany zostanie jawnie znak danego formatu;
VI. tekst pomiędzy znakami cudzysłowu " " - ten tekst ukaże się na ekranie;
VII. **polecenie cr** - kursor przeniesie się na początek nowej linii;
VIII. **polecenie cls** - ekran zostanie wyczyszczony.

Przykłady:

`DEBUG b3` -> Na ekranie pojawi się „b3=" oraz jej wartość

`DEBUG #b3` -> dziesiętnie zapisana wartość zmiennej b3

`DEBUG "Zliczono ",b3,"przypadków"` ->Będzie „Zliczono"+wartość b3+„przypadków"

`DEBUG #%b3` ->Wartość zmiennej b3 zapisana binarnie

`DEBUG #@b3` ->Wartość zmiennej b3 zapisana jako znak ASCII

EEPROM

Składnia:

```
EEPROM {położenie},(dana,dana,...)
```

Instrukcja EEPROM jest używana do składowania danych do pamięci EEPROM układu Basic Stamp. Interpretacja argumentów jest następująca:

I. **położenie** - zmienna albo stała, która reprezentuje liczbę z zakresu 0-255 wskazującą położenie pierwszej danej w pamięci EEPROM.
II. **dana** - zmienna albo stała reprezentująca liczbę z zakresu 0-255, która będzie zapisana do EEPROM. Jeśli liczba danych jest większa od jednej, następne dane są zapisywane sekwencyjnie pod następne adresy pamięci EEPROM.

END

Składnia:

```
END
```

Instrukcja wprowadzenia procesora w stan uśpienia. Pobór prądu spada do poziomu 20µA, nie licząc poboru prądu przez dołączone obciążenia.

FOR...NEXT

Składnia:

```
FOR zmienna = początek TO koniec
{STEP {-} krok } ... NEXT {zmienna}
```

Instrukcja ta tworzy licznikową pętlę programową. Składa się ona z dwóch fraz, pomiędzy którymi znajduje się powtarzana sekwencja instrukcji, w powyższej definicji zaznaczona wielokropkiem. Znaczenie parametrów instrukcji FOR... NEXT jest następujące:

I. **zmienna** - bit, bajt lub słowo używane jako licznik pętli. Zakres wartości jest ograniczony przez samą definicję zmiennej. Zmienna bitowa przyjmuje dwie wartości (0 albo 1), zmienna bajtowa jest z zakresu 0..255, a słowo ma pojemność od 0..65535. Parametr zmienna jest konieczny we frazie FOR.... We frazie ...NEXT parametr ten jest opcjonalny i służy do identyfikacji końca pętli w grupie pętli zagnieżdżonych;
II. **początek** - zmienna lub stała, która określa wartość początkową licznika pętli;
III. **koniec** - zmienna lub stała, która określa wartość końcową licznika pętli;
IV. **krok** - zmienna lub stała, definiująca krok zmiany licznika pętli oraz kierunek tej zmiany. Licznik jest zwiększany o wartość krok, gdy jest to liczba dodatnia, a zmniejszany, kiedy jest to liczba ujemna. Oczywiście pętla o kroku równym 0 jest martwą pętlą. Parametr krok jest parametrem opcjonalny, czyli nie musi on obowiązkowo wystąpić. Brak tego parametru we frazie FOR... oznacza jego domyślną wartość równą 1.

Zmienna przyjmuje wartość początek. Sekwencja instrukcji umieszczona pomiędzy frazami FOR... i ...NEXT będzie wykonana co najmniej jeden raz. Potem nastąpi zmiana licznika pętli o wartość krok (przy braku określenia wartości krok zmienna będzie powiększona o 1). Jeśli po zmianie wartość zmiennej jest nie mniejsza (większa lub równa) od wartości koniec, pętla będzie zakończona. W przypadku przeciwnym zachodzi skok do początku pętli i sekwencja instrukcji pomiędzy frazami FOR... i ...NEXT zostanie wykonana ponownie.

Ze względu na ograniczony zakres zmiennych może powstać sytuacja, że pętla tylko z pozoru będzie wyglądać na pętlę skończoną. Na przykład zapis

```
for b3=0 to 220 step 100
```

na pierwszy rzut oka definiuje pętlę, która powinna się zakończyć po czterech krokach. Tak się nie stanie, ponieważ zmienna b3 jest bajtem, czyli jej zakres wynosi od 0 do 255. Zmienna b3 będzie przyjmować kolejno następujące wartości: 0, 100, 200, 44, 144, 244. Dopiero po szóstym wykonaniu pętli nastąpi spełnienie warunku jej zakończenia. Dzieje się tak dlatego, że powiększanie zmiennej b3 odbywa się modulo 256. Spodziewaną po 200 liczbę 300 można zapisać jako 256+44. Dodawanie 200+100 jako jednobajtowe powoduje, że przeniesienie na następną pozycję jest tracone. Pozostaje reszta równa 44.

GOSUB

Składnia:

```
GOSUB etykieta
```

Jest to instrukcja skoku do podprogramu. Jej parametrem jest etykieta. Liczba możliwych zagnieżdżeń wynosi 16.

Podprogram kończy instrukcją RETURN, po której następuje skok do instrukcji znaj-

dującej się po GOSUB, wywołującej ten podprogram.

Podprogramy są stosowane w celu skrócenia wielkości programu, poprzez odwoływanie się do ciągów powtarzających się instrukcji.

GOTO

Składnia:

```
GOTO etykieta
```

Instrukcja skoku do miejsca określonego etykietą. Program jest wykonywany dalej od linii, którą ta etykieta wskazuje.

HIGH

Składnia:

```
HIGH pin
```

Instrukcja ustawiania jedynki logicznej na wybranym pinie. Jeśli ów pin został wcześniej zaprogramowany jako wejście, następuje zmiana deklaracji i staje się on wyjściem. Wartość pin musi być liczbą lub zmienną reprezentującą liczbę z zakresu 0..7, która określa numer pinu.

Jeśli określimy argument tej instrukcji jako zmienną, oznacza to wskazanie, gdzie tej wartości należy szukać. Zapis, który dla programistów jest oczywisty

```
HIGH pin2
```

wcale nie oznacza ustawienia pinu 2 portu we/wy. Powyższy zapis oznacza ustawienie pinu 0, gdy pin 2 jest wyzerowany albo ustawienie pinu 1, kiedy pin 2 jest ustawiony. O tym należy pamiętać.

IF..THEN

Składnia:

```
IF warunek THEN etykieta
```

Instrukcja skoku warunkowego. Jeśli warunek jest logicznym zdaniem oznaczającym prawdę, nastąpi skok do linii oznaczonej etykieta.

Warunek ma następującą składnię:

zmienna ?? wartość {AND/OR zmienna ?? wartość}

Znaczenie powyższych argumentów jest następujące:

zmienna - zmienna, która jest porównywana z wartość;

wartość - zmienna lub stała;

?? - jeden z następujących operatorów relacji: = (równy), <> (różny), > (większy), < (mniejszy), >= (większy lub równy, nie mniejszy), <= (mniejszy lub równy, nie większy).

Warunek może zawierać więcej niż jedną z relacji porównania połączone ze sobą operatorem AND albo OR.

Operator iloczynu logicznego AND oznacza, że wynik jest prawdą wtedy i tylko wtedy, gdy oba argumenty tego iloczynu są prawdziwe, w pozostałych przypadkach wynikiem operacji iloczynu logicznego jest fałsz.

Operator sumy logicznej OR oznacza, że wynik jest prawdą wtedy i tylko wtedy, gdy co najmniej jeden z argumentów jest prawdziwy. Jeśli oba argumenty są fałszywe, wynik jest też fałszywy.

Interpreter PBasic nie ma zbyt dużych możliwości. W składni warunek nawiasy nie zmieniają kolejności wykonywania operacji logicznych. Obliczanie wartości warunku odbywa się od lewej do prawej.

INPUT

Składnia:

```
INPUT pin
```

Instrukcja definiowania określonego pinu portu we/wy jako wejścia. Argumentem tej instrukcji jest stała lub zmienna reprezentująca liczbę z zakresu 0..7.

LET

Składnia:

```
LET zmienna={-}wartość ?? wartość...
```

Instrukcja przypisania argumentowi zmienna wartości wynikającej z wykonania operacji znajdujących się po znaku równości. Słowo LET jest opcjonalne. Jest ono dopuszczalne jako relikt specyfikacji i równoważnym zapisem jest zapis z jego pominięciem.

Operacja oznaczona tutaj jako ?? może być jedną z następujących:

+ (dodawanie);
- (odejmowanie);
* (mnożenie - wynikiem jest młodszy bajt iloczynu);
** (mnożenie - wynikiem jest starszy bajt iloczynu);
/ (dzielenie - wynikiem jest iloraz);
// (dzielenie - wynikiem jest reszta z dzielenia);
min (przyjęcie wartości nie mniejszej);
max (przyjęcie wartości nie większej);
& (iloczyn logiczny - AND);
| (suma logiczna - OR);
^ (suma modulo 2 XOR);
&/ (inwersja iloczynu logicznego - AND NOT);
|/ (inwersja sumy logicznej - OR NOT);
^/ (inwersja sumy modulo 2 - XOR NOT).

Arytmetyka języka PBASIC nie uznaje liczb innych niż naturalne oraz zero. Zakres tych liczb sięga do 65535. Nie są też uznawane nawiasy, czyli wszystkie operacje są wykonywane od lewej do prawej. Co ważne, kolejność wykonywania działań jest jednakowa dla wszystkich operatorów zapisów.

Zatem działanie

```
2+3*5
```

będzie wykonane inaczej niż my jesteśmy do tego przyzwyczajeni. Zgodnie z zasadami arytmetyki mnożenie będzie wykonane przed dodawaniem i wynik jest równy 17. Te same operacje wykonane w języku PBASIC dadzą wynik równy 30, bowiem najpierw zostanie wykonane dodawanie, a potem mnożenie.

LOOKDOWN

Składnia:

```
LOOKDOWN wzorzec,(wartość0,
wartość1,...,wartośćN),zmienna
```

Instrukcja, która do zmiennej zapisuje liczbę porządkową wartości umieszczonej w nawiasach równej wzorcowi. Znaczenie argumentów tej instrukcji jest następujące:

- ✓ wzorzec - zmienna albo stała, która oznacza poszukiwaną wartość;
- ✓ wartość0, wartość1,...,wartośćN - lista wartości, które są porównywane ze wzorcem;
- ✓ zmienna - do niej jest zapisywana liczba porządkowa wartości (liczba porządkowa pierwszej pozycji jest równa 0) równej wzorcowi.

Zmienna nie ulega zmianie, jeśli wzorzec jest różny od każdej wartości.

LOOKUP

Składnia:

```
LOOKUP przesunięcie,(wartość0, wartość1,..., wartośćN), zmienna
```

Instrukcja zapisu do zmiennej wartości określonej przez przesunięcie. Znaczenie argumentów tej instrukcji jest następujące:

- ✓ przesunięcie - zmienna albo stała, oznaczająca liczbę porządkową wybranej wartości (liczba porządkowa pierwszej pozycji jest równa 0);
- ✓ wartość0, wartość1,...wartośćN - lista wartości;
- ✓ zmienna - do niej zostanie przepisana wartość wskazana przez przesunięcie.

Instrukcja nic nie zmienia, gdy przesunięcie wskazuje liczbę porządkową większą niż długość listy wartości.

Mirosław Lach, AVT