ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1987

Seria: AUTOMATYKA z. 83 Nr kol. 888

Janusz JEŻEWSKI

JĘZYK SAPL - STRUKTURALNY ASEMBLER DO PROGRAMOWANIA MIKROPROCESORÓW

Streszczenie. Prezentowany w pracy język programowania mikroprocesorów SAPL (Structural Assembly Programming Language) umożliwia tworzenie programów strukturalnych z pełnym wykorzystaniem cech języka maszynowego.
Tworzenie programów zgodnych z zasadami strukturalnego programowania umożliwia:

- hierarchiczna budowa języka,
- uporządkowanie danych,
- odrębne konstrukcje organizujące przebieg wykonywania programu.

Wyeliminowanie mnemonik jako zapisu prostych instrukcji na korzyść notacji uogólnionej zapewnia uniwersalność sprzętową narzędzia. Istotne elementy gramatyki języka SAPL podano w postaci formalnej oraz wyjaśniono na przykładach.
W zakończeniu ustosunkowano się do problemów implementacyjnych oraz wytyczono kierunki dalszych prac nad językiem .

1. Wprowadzenie

Gwałtowny postęp w technologii wytwarzania układów mikroprocesorowych oraz ciągły rozwój zastosowań systemów mikrokomputerowych wymagają coraz lepszych metod i narzędzi ich efektywnego oprogramowania. Jednoznaczna odpowiedź na pytanie, czy stosować wygodne w użytkowaniu języki wysokiego poziomu, czy też bardziej uciążliwe, lecz efektywniejsze języki poziomu asemblera, staje się coraz trudniejsza. Bezkrytyczne wyciąganie wniosków z doświadczeń przy konstrukcji oprogramowania dużych systemów komputerowych może prowadzić do zbyt pochopnych stwierdzeń.

Wydaje się bowiem, iż naturalny rozwój i postęp to wykorzystanie języków wysokiego poziomu. Budowa oprogramowania na bazie języków wysokiego poziomu zapewnia łatwość jego późniejszej modyfikacji. Modyfikacje w celu poprawy wydajności czy też lepszego dostosowania do zmieniającego się środowiska (np. sprzętowego) są konieczne dla powstrzymania procesu starzenia się oprogramowania. Jeżeli programy oparto na językach symbolicznych, to zapewnienie właściwej ich pielęgnacji jest bardzo kosztowne i czasochłonne.

Dokładna analiza zagadnienia dostarcza równie ważkich dowodów na korzyści płynące ze stosowania języków poziomu asemblera. Do najbardziej oczywistych należy efektywne gospodarowanie czasem i zasobami pamięci. Jednakże zdaniem autora najistotniejszym powodem tego, że języki symbo-

liczne, pomimo licznych swych wad, są i w najbliższej przyszłości będą chętnie stosowane, jest czynnik ludzki. Bazuje on na przyzwyczajeniu i mentalności dużej rzeszy czynnych projektantów oprogramowania, którzy jeszcze wczoraj byli konstruktorami sprzętu, dla których oprogramowanie w asemblerze stanowi naturalne przejście od projektowania logiki konwencjonalnej. Im właśnie, nie będącym zawodowymi programistami, trudno jest opanować bardziej abstrakcyjne techniki programowania zalecane przy stosowaniu języków wysokiego poziomu.

W celu pogodzenia powyższych sprzeczności proponowane są rozwiązania pośrednie, z których jedno opiera się na koncepcji języka poziomu pośredniego (ang. a medium - level language) [4]. Język ten łączy w sobie konstrukcje i notację języków wysokiego poziomu z mechanizmami właściwymi językom poziomu asemblera.

2. Asembler strukturalny jako język poziomu pośredniego

Głównym celem projektantów języków pośrednich było stworzenie narzędzia łączącego w sobie wybrane pozytywne cechy zarówno języków wysokiego poziomu, jak i języków symbolicznych. Z powszechnie stosowanych strukturalnych języków algorytmicznych zaczerpnięto konstrukcje bardziej ogólne, a ułatwiające kojarzenie programu z samym algorytmem. Umożliwia to nowa notacja zwiększająca czytelność i przejrzystość programu oraz wprowadzenie specjalnych mechanizmów dla strukturalizacji.

Dobierając konstrukcje z języków wysokiego poziomu uwzględniono następujące założenia:

a) zapewnienie hierarchicznego projektowania i tworzenia programu,
b) wyraźne wydzielenie części deklaracyjnych i wykonawczych,
c) uporządkowanie danych,
d) wprowadzenie zakresów działania identyfikatorów obiektów,
e) zwiększenie klarowności sterowania w części wykonawczej,
f) umożliwienie tworzenia programów bez konieczności stosowania etykiet i instrukcji skoku,
g) zabezpieczenie się przed zbytnim dublowaniem samych języków wysokiego poziomu.

Uwzględnienie powyższych założeń wpłynie na podniesienie dyscypliny programowania poprzez wymuszenie lub chociażby nakłonienie przyszłych użytkowników języka do stosowania bardziej efektywnej metody programowania - metody stopniowego uszczegółowiania.

Analizując użyteczne własności języków poziomu sprzętu jako nieodzowne w programowanym języku uznano:

a) pełne panowanie programisty nad generowanym kodem wynikowym.

b) bezpośredni dostęp do własności sprzętu, pomimo iż semantyka prostych instrukcji definiowana będzie w mniejszym stopniu poprzez sprzęt,
c) umożliwienie tradycjonalistom stosowania etykietowania i instrukcji skoku.

Ad a) Przekład powinien być tworzony w skali 1 do n, gdzie n jest znane i stałe dla danego typu konstrukcji. Obowiązuje dążenie do zapewnienia n równego 1 i to zarówno przy tłumaczeniu konstrukcji wyższego poziomu, jak i instrukcji prostych.

Ad b) W tradycyjnym języku poziomu asemblera semantyka prostych instrukcji opisywanych mnemoniką jest definiowana bezpośrednio przez odpowiedni fragment sprzętu. Języki pośrednie z reguły preferują bardziej uogólniony opis rozkazów bazujący na standaryzowanych mnemonikach [1] lub też oparty na klasycznej symbolice matematycznej [4] z instrukcją przypisania i wyrażeniem warunkowym.

Kryterium dodatkowym przy opracowywaniu podstaw języka SAPL było zapewnienie dużej prostoty gramatyki ułatwiającej szybkie wprowadzenie języka do praktyki w procesie dydaktycznym oraz zapewniającej szybkie opanowanie narzędzia przez studentów.

3. Opis języka SAPL

3.1. Słownik języka

Program w języku SAPL stanowi skończony ciąg symboli ze skończonego słownika. W skład słownika wchodzą:

- identyfikatory,
- liczby,
- teksty,
- słowa kluczowe,
- operatory,
- ograniczniki,
- separatory.

Przez pojęcie identyfikatora (nazwy) rozumie się ciąg liter i cyfr rozpoczynający się od litery.

Liczby traktowane są jako całkowite, ich zapis stanowi ciąg cyfr. Wyróżniono cztery postacie zapisu: dwójkową, ósemkową, dziesiętną i szesnastkową. Pierwszym znakiem zapisu jest cyfra.

Tekst stanowi ciąg znaków ograniczony obustronnie podwójnymi apostrofami. Słowa kluczowe budowane są z dużych liter i stanowią nazwy zastrzeżone. Operatory i ograniczniki reprezentowane są pojedynczym lub podwójnym znakiem różnym od cyfry i litery.

```
? Przykład ilustruje ogólną strukturę programu w języku ?
? SAPL. Moduł o nazwie : Komunikacja_z_operatorem bada ?
? tryb pracy systemu i informuje o tym użytkownika wy- ?
? prowadzając odpowiedni komunikat na drukarkę. Samym ?
? drukiem zajmuje się procedura Druk_komunikatu, która ?
? korzysta z procedury wewnętrznej Wysłanie_znaku odpo- ?
? wiedzialnej za druk pojedynczego znaku. ?

Komunikacja_z_operatorem MODULE;
   INTERNAL   Adr_aktual_hasła WORD,
              Bufor [1Ø] BYTE,
              Zbiór_haseł [2ØØ] BYTE := ["BREAK ON",ØFF H, ...
                                         ,"READY",ØFF H, ... ,"WAIT FOR",
                                         ØFF ] ;
   INTERNAL  Druk_komunikatu PROCEDURE;
   Druk_komunikatu PROCEDURE;
      CONSTANT  Znacznik_końca := ØFF H;
      LOCAL  Znak BYTE;
      Wysłanie_znaku PROCEDURE;
         CONSTANT  Adr_sterow := 1A H,
                   Adr_danych_wyj := 1B H,
                   Maska := ØØØØØ1ØØ B;
         BEGIN
               REPEAT  A := IN(Adr_sterow); A := A & Maska
            UNTIL  A<>Ø;          ? Czekaj na gotowość ?
               A := Znak;
               OUT(Adr_danych_wyj) := A
      END  Wysłanie_znaku;
      BEGIN
            HL :=@ Bufor;
            LOOP
               A := (HL);
               Znak := A;
               IF A = Znacznik_końca
                                          THEN  EXIT
               FI;
               CALL Wysłanie_znaku;
               HL := HL + 1;
            POOL
   END  Druk_komunikatu;
      .
      .
   ? Kopiowanie ze Zbioru_haseł do Bufora wg Adresu_aktual_hasła ?
      .
      .
   CALL Druk_komunikatu;
      .
      .
END  Komunikacja_z_operatorem;
```

Przykład 1. Moduł "Komunikacja z operatorem" jako ogólna idea notacji oraz struktury programu w języku SAPL

Example 1. The module "Komunikacja z operatorem" as a general idea of notation and program structure in SAPL language

Rolę separatorów pełnią znak spacji i komentarz. Separatory (w dowolnej liczbie) można umieszczać między dowolnymi dwoma symbolami programu; nie mają one wpływu na wykonanie programu.

3.2. Program

Programem nazywamy kompletny opis danego algorytmu. Program jest najwyższą jednostką programową. Każdy program w języku SAPL jest modułem lub też składa się z kilku modułów, z których każdy jest oddzielnie tłumaczony. Moduły dzielą duży program na logiczne części, a mechanizmy wymiany międzymodułowej definiują specjalne atrybuty.

Podstawową strukturą programową jest blok. Wyróżniono dwa rodzaje bloków: blok modułu i blok procedury. Blok modułu stanowi treść wnętrza modułu.

Do opisu złożonych operacji, z reguły wymagających wprowadzenia dodatkowych obiektów, służą procedury. Wnętrze procedury stanowi blok procedury. Dzięki mechanizmom wielopoziomowego zagnieżdżania procedur umożliwiono stosowanie hierarchicznego projektowania i budowy programu. W bloku wydzielono dwie części: część deklaracyjną opisu obiektów oraz część wykonawczą do opisu czynności. W części deklaracyjnej obiektom przyporządkowuje się identyfikatory, ustala się zakres ich stosowania oraz precyzuje się atrybuty dodatkowe. Część wykonawczą bloku opisuje się za pomocą instrukcji języka. Wyróżniono instrukcje proste i instrukcje strukturalne.

Podstawowe instrukcje proste stanowią: instrukcja przypisania i instrukcja wywołania procedury. Wśród instrukcji strukturalnych wyróżniamy instrukcje: iteracyjne, warunkowe oraz wyboru.

Ideę notacji oraz struktury programu oddaje program zamieszczony w przykładzie 1.

3.3. Moduł

Moduł jest niezależną jednostką kompilacyjną w ramach jednego programu. Podział programu na moduły wiąże się najczęściej z logicznym podziałem złożonego problemu na fragmenty. W pewnych wypadkach podziału dokonuje się wg kryteriów fizycznych, na przykład:

- moduł stanowi fragment programu pisany i uruchomiany przez jednego programistę,
- moduł obejmuje część programu mieszczącą się jeszcze w dysponowanej pamięci.

Ogólna gramatyka modułu przedstawia się następująco:*

<moduł> ::= <identyfikator modułu> $\left\{\underline{\text{MAIN}}\right\}_0^1$ MODULE;
<blok modułu>
END $\left\{\text{<identyfikator modułu>}\right\}_0^1$;

<blok modułu> ::= <część deklaracyjna modułu>
BEGIN
<część wykonawcza modułu>

<część dekl. modułu> ::= $\left\{\text{<definicja stałych>}\right\}_0^n$
$\left\{\text{<deklaracja obiektu>}\right\}_0^n$
$\left\{\text{<definicja procedury>}\right\}_0^n$

<część wyk. modułu> ::= <złożenie instrukcji>

<złożenie instrukcji> ::= <instrukcja> $\left\{\text{; <instrukcja>}\right\}_0^n$

Jeżeli program składa się z kilku modułów, to jeden z nich wyróżniony jest jako główny (MAIN). Niezależnie tłumaczone moduły mogą być połączone, gdyż deklaracje z atrybutami EXTERNAL i GLOBAL rozszerzają zakres deklaracji jednego modułu na inne moduły.

<deklaracja obiektu> ::= <deklaracja wewnętrzna>|
<deklaracja zewnętrzna>|
<deklaracja globalna>

<deklaracja wewnętrzna> ::= INTERNAL $\left\{\text{<obiekt>}\right\}_1^n$;

<deklaracja zewnętrzna> ::= EXTERNAL $\left\{\text{<specyfikacja obiektu>}\right\}_1^n$;

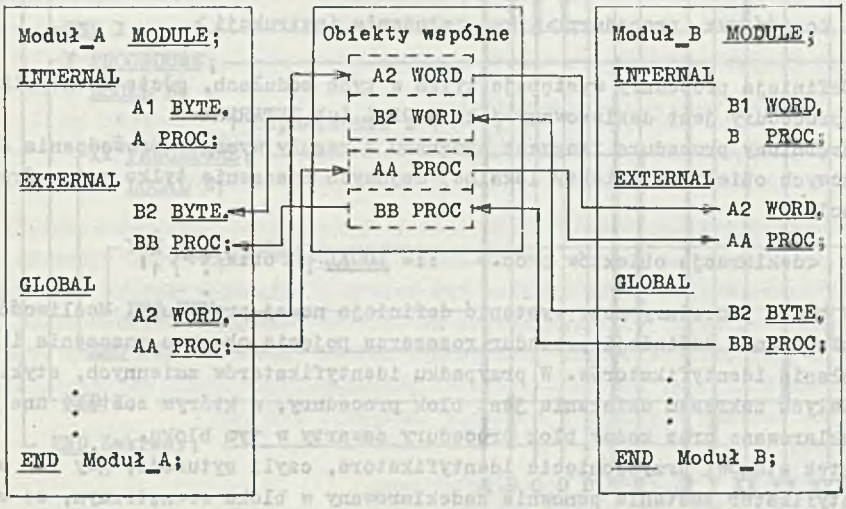
<deklaracja globalna> ::= GLOBAL $\left\{\text{<obiekt>}\right\}_1^n$;

<obiekt> ::= <deklaracja zmiennych>|
<deklaracja etykiet>|
<deklaracja procedur>

<spec. obiektu> ::= <specyfikacja zmiennych>|
<deklaracja etykiet>|
<deklaracja procedur>

*Formalny opis wybranych fragmentów gramatyki języka przedstawiono w postaci Backusa - Naura. Do notacji tej wprowadzono dodatkowe oznaczenie: $\{x\}_k^n$ oznacza i-krotną listę złożoną z symboli X, gdzie $k \leqslant i \leqslant n$. Dodatkowo słowa kluczowe dla wyodrębnienia są podkreślane.

Wszystkie zmienne, etykiety i procedury deklarowane z atrybutem INTERNAL są dostępne (lokalne) tylko w danym module. W deklaracji globalnej deklarowane są te obiekty, których użycie przewiduje się w innych modułach. Natomiast z atrybutem EXTERNAL specyfikuje się te obiekty, które zostały zadeklarowane jako globalne w innym module, lecz będą wykorzystywane również w module bieżącym. Posługując się atrybutami GLOBAL i INTERNAL określono sposób wymiany informacji i komunikację pomiędzy modułami za pomocą wspólnych zmiennych, procedur i etykiet.
Sposób użycia ilustruje przykład 2.



Przykład 2. Komunikacja międzymodułowa za pomocą obiektów z atrybutem GLOBAL i EXTERNAL

Example 2. An intermodule communication by objects with GLOBAL and EXTERNAL atributes

Różnica pomiędzy deklaracją zmiennych a specyfikacją sprowadza się do tego, że zmiennym w specyfikacji (deklaracja zewnętrzna) nie wolno nadawać wartości początkowych.

3.4. Procedura

Procedura obok modułu jest podstawową konstrukcją języka SAPL. Stanowi podstawowy mechanizm umożliwiający podział rozwiązywanego problemu na logicznie spójne fragmenty. Treścią definicji procedury jest blok procedury, który zawiera opis fragmentu algorytmu.

<definicja procedury> ::= <identyfikator procedury> PROCEDURE;
<blok procedury>
END{<identyfikator procedury>}$_0^1$;

<blok procedury> ::= <część deklaracyjna procedury>
BEGIN
<część wykonawcza procedury>

<część dekl. procedury> ::= {<definicja stałych>}$_0^n$
{<deklaracja obiektów proc.>}$_0^n$
{<definicja procedury>}$_0^n$

<część wyk. procedury> ::= <złożenie instrukcji>

Definicja procedury występuje tylko w tych modułach, gdzie identyfikator procedury jest deklarowany jako GLOBAL lub INTERNAL.
Wyodrębniony procedurą fragment programu z reguły wymaga wprowadzenia dodatkowych obiektów (obiekty lokalne) mających znaczenie tylko w tym fragmencie.

<deklaracja obiektów proc.> ::= LOCAL {<obiekt>}$_1^n$;

W bloku procedury może wystąpić definicja nowej procedury. Możliwość zagnieżdżania definicji procedur rozszerza pojęcie obszaru znaczenia i działania identyfikatorów. W przypadku identyfikatorów zmiennych, etykiet i stałych zakresem działania jest blok procedury, w którym zostały one zadeklarowane oraz każdy blok procedury zawarty w tym bloku.
Wyjątek stanowi przysłonięcie identyfikatora, czyli sytuacja, gdy ten sam identyfikator zostanie ponownie zadeklarowany w bloku wewnętrznym, co wiąże się z nadaniem mu nowego znaczenia w tym bloku.

Zakres działania identyfikatora procedury jest węższy. Obejmuje blok, w którym została procedura zdefiniowana oraz bloki równoległe, to jest bloki procedur, które w hierarchii zagnieżdżonych bloków są w rozpatrywanym nadbloku na tym samym poziomie, co blok danej definicji procedury. Dodatkowo blok definicji tej procedury musi poprzedzać bloki procedur równoległych.
Ilustrację stanowi przykład 3.

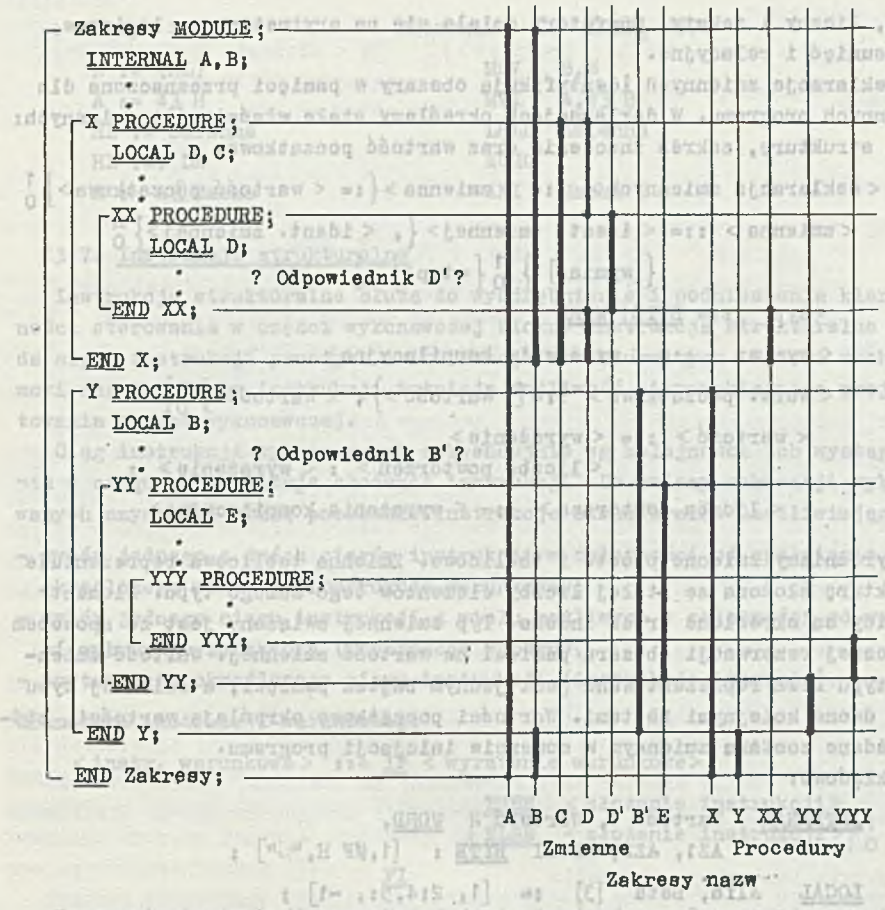
3.5. Stałe i zmienne

Stałe służą do wiązania identyfikatorów z wybranymi wartościami.

<definicja stałych> ::= CONSTANT <stała> {, <stała>}$_0^n$

<stała> ::= <identyfikator stałej> := <wartość stałej>

<wartość stałej> ::= <wyrażenie kompilacyjne> |
<synonim obiektu>



Przykład 3. Zakresy działania identyfikatorów zmiennych i procedur w obrębie modułu

Example 3. The activity range of variable and procedure identifiers in a module body

Wyróżniamy stałe reprezentujące wartości liczb i teksty oraz stałe pełniące funkcje synonimów obiektów (np. rejestrów lub bitów stanu mikroprocesora).

```
CONSTANT   Maska := ØØØØ Ø1ØØ B, Rozmiar := 12,
           KRES := Rozmiar * 3 + 1ØØ, NAWIAS_IF := "IF",
           bufor := HL, Przeniesienie := AC;
```

Wyrażenie kompilacyjne określa wartości wyliczone w czasie kompilacji. Składa się z argumentów i operatorów. Argumentami są identyfikatory sta-

łych, liczby i teksty. Operatory dzielą się na arytmetyczne, logiczne, przesunięć i relacyjne.

Deklaracje zmiennych identyfikują obszary w pamięci przeznaczone dla zmiennych programu. W deklaracjach określamy stałe właściwości zmiennych: typ, strukturę, zakres znaczenia oraz wartość początkową.

< deklaracja zmiennych > ::= < zmienna > $\left\{ := < \text{wartość początkowa} > \right\}_{0}^{1}$

< zmienna > ::= < ident. zmiennej > $\left\{ , < \text{ident. zmiennej} > \right\}_{0}^{n}$
$\left\{ [\text{wymiar}] \right\}_{0}^{1} \left\{ < \text{typ} > \right\}_{0}^{1}$

< typ ::= BYTE | WORD

< wymiar ::= < wyrażenie kompilacyjne >

< wart. początkowa > ::= $\left[< \text{wartość} > \right\}, < \text{wartość} > \Big\}_{0}^{n} \Big]$

< wartość > ::= < wyrażenie >
< liczba powtórzeń > : < wyrażenie > :

< liczba powtórzeń > ::= < wyrażenie kompilacyjne >

Wyróżniamy zmienne proste i tablicowe. Zmienna tablicowa reprezentuje strukturę złożoną ze stałej liczby elementów tego samego typu. Elementy tablicy są określone przez indeks. Typ zmiennej związany jest ze sposobem fizycznej rezerwacji obszaru pamięci na wartość zmiennej. Wartość zmiennej typu BYTE reprezentowana jest jednym bajtem pamięci, a zmiennej typu WORD dwoma kolejnymi bajtami. Wartości początkowe określają wartości, które nadane zostaną zmiennym w momencie inicjacji programu.
Przykładowo:

```
INTERNAL   Wartość 1, Wartość 2  WORD.
           AZ1, AZ2, ASCII  BYTE :  [1,ØF H,"%"] ;

LOCAL   Alfa, Beta  [3]  :=  [1, 2:4,5:, -1] ;
```

3.6. Instrukcje proste

Instrukcje proste stanowią odpowiedniki podstawowych operacji mikrokomputera. Każda pojedyncza instrukcja prosta jest tłumaczona dokładnie na pojedynczy rozkaz maszynowy. Proponowany zapis instrukcji jest bardziej uogólniony niż zapis za pomocą mnemonik. Wzorowany jest na instrukcji przypisania (podstawienia) przejętej z języków wysokiego poziomu. Wyróżniono następujące instrukcje proste:

- instrukcje przypisania,
- instrukcje sterujące (np. instrukcja wywołania procedury),
- instrukcje dodatkowe (np. operacje na układzie przerwań).

Ogólna postać instrukcji przypisania:

< instrukcja przypisania > ::= < obiekt przeznaczenia > := < obiekt źródłowy >

Przykłady:

Język SAPL	Mnemonika μP INTEL 8080
B := (HL)	MOV B,M
A := 43 H	MVI A,43 H
HL := Zmienna	LHLD Zmienna
HL :=: DE	XCHG
A := A&Maska	ANI Maska

3.7. Instrukcje strukturalne

Instrukcje strukturalne służą do wyodrębnienie i podniesienia klarowności sterowania w części wykonawczej bloku. Instrukcja strukturalna składa się z instrukcji prostych i instrukcji strukturalnych. Dzięki mechanizmowi zagnieżdżania instrukcji istnieje możliwość hierarchicznego projektowania części wykonawczej.

Ciąg instrukcji wykonywanych sekwencyjnie wg kolejności ich wystąpienia w ciągu reprezentuje złożenie instrukcji. Do zmiany sekwencji wykonywanych czynności służą pozostałe instrukcje strukturalne umożliwiające:

- wybór jednego z dwóch ciągów instrukcji w zależności od spełnienia określonego warunku (instrukcja warunkowa),
- wybór jednego ciągu instrukcji z wielu możliwych w zależności od wartości wybranego selektora (instrukcja wyboru),
- powtarzanie określonego ciągu instrukcji (instrukcje iteracji).

Gramatyka instrukcji warunkowej:

< instr. warunkowa > ::= IF < wyrażenie warunkowe >

THEN < złożenie instrukcji >
{ ELSE < złożenie instrukcji > }$^{1}_{0}$

FI

Znaczenie instrukcji warunkowej jest standardowe. Opracowując zapis wyrażenia warunkowego uwzględniono ograniczenia powodowane koniecznością zapewnienia minimalnej skali przekładu na rozkazy maszynowe.
Założono, że przekład wyrażenia warunkowego musi być ograniczony do jednej instrukcji skoku warunkowego oraz jednej instrukcji skoku bezwarunkowego poprzedzonych opcjonalnie instrukcją, która uaktywnia bit stanu (np. instrukcją porównania). Fragment gramatyki wyrażenia warunkowego w odniesieniu dla mikroprocesora INTEL 8080 przedstawia się następująco:

< wyrażenie warunkowe > ::= < identyfikator bitu stanu > |
< akumulator > < op. relacji > < argument >

< argument > | ::= < rejestr > | < identyfikator stałej > | (HL)

Ogólna postać instrukcji wyboru:

ON < selektor >

$\{$CASE < argument > $\{$, < argument >$\}_0^n$ DO < złożenie instrukcji>$\}_1^n$

$\{$ELSE < złożenie instrukcji>$\}_0^1$

NO

Przykład:

```
ON  A
      CASE 1,B     DO  HL = HL+1  ? Jeżeli A=1 lub  A=B to zwiększ?
      CASE 2,Maks  DO  HL = HL+1; HL = HL+1  ? HL o 1. Jeżeli A=2 ?
    ELSE   HL = HL-1   ? lub A=Maks to zwiększ HL o 2, w innych   ?
NO:                    ? wypadkach zmniejsz wartość HL o 1        ?
```

Efektem instrukcji wyboru jest wykonanie złożenia instrukcji tego członu CASE, w którym wartość co najmniej jednego argumentu równa się wartości selektora z nagłówka ON. Jeżeli w żadnym z członów CASE nie wystąpi taka równość, to wykonane zostanie złożenie instrukcji po słowie ELSE. Nawias kończący NO (FI) pozwala zagnieżdżać instrukcje wyboru (warunkowe) bez narażania się na niejednoznaczność.

Język SAPL wyróżnia trzy rodzaje instrukcji iteracji:

- instrukcję WHILE,
- instrukcję REPEAT,
- instrukcję LOOP.

Różnią się pomiędzy sobą miejscem sprawdzania warunku zakończenia procesu iterowania.
W instrukcji WHILE warunek jest sprawdzany na początku, natomiast w instrukcji REPEAT na końcu złożenia instrukcji przeznaczonego do cyklicznego powtarzania..
Gramatyka instrukcji iteracji:

< instr. z warunkiem początk.> ::= WHILE < wyrażenie warunkowe >
DO < złożenie instrukcji > OD

< instr. z warunkiem końcowym > ::= REPEAT < złożenie instrukcji >
UNTIL < wyrażenie warunkowe >

< instrukcja pętli > ::= LOOP
< złożenie instrukcji >
POOL

Złożenie instrukcji ograniczone nawiasami LOOP ... POOL jest wykonywane cyklicznie w sposób ciągły. Sterowanie zakończeniem iteracji umożliwiają instrukcje posiłkowe EXIT i CONTINUE sprzęgnięte z instrukcją warunkową. Badanie warunku może wystąpić w dowolnym miejscu instrukcji pętli. Instrukcja EXIT przekazuje sterowanie na zewnątrz, instrukcja CONTINUE zaś do początku pętli.

Przykład:

```
LOOP
        A := IN (Bufor_wej)
        IF A = ØFF H  THEN  EXIT,
        (HL) := A,
         B := B-1,
        HL := HL+1,
        IF B<>Ø  THEN  CONTINUE
rOOL;
```

4. Uwagi o implementacji

Pierwotną wersję języka SAPL opracowano w Instytucie Elektroniki w 1982 roku z przeznaczeniem dla systemu uruchomieniowego z mikroprocesorem TMS 9900. Słabą stroną tej wersji było oparcie notacji instrukcji na symbolice mnemonik. Drugie istotne ograniczenie to założenie, że blok procedury nie może zawierać definicji nowej procedury. Program tłumaczący i redagujący tekst źródłowy w postaci skrośnej został opracowany na m.c. ODRA 1305.

Kolejna wersja języka opracowana na początku 1984 roku umożliwiała już zapis instrukcji prostych na podstawie notacji przejętej z języków algorytmicznych oraz wprowadzała hierarchię definiowania procedur. Ponadto dla ułatwienia praktycznego stosowania metodologii strukturalnego projektowania i programowania wprowadzono dodatkową wizualną strukturalizację programu w języku źródłowym. Wizualizacja obejmuje implementację formatowanego wydruku postaci źródłowej oraz prezentację programu opierając się na struktogramach [7].

Program tłumaczący oraz pełne oprogramowanie towarzyszące opracowano najpierw na komputer osobisty SINCLAIR ZX81, a później na ZX Spectrum. W chwili obecnej język oraz oprogramowanie umożliwiają generowanie kodu wynikowego dla mikroprocesorów INTEL 8080 oraz Z80.

W najbliższej przyszłości przewidywane są prace nad zwiększeniem uniwersalności narzędzia poprzez rozszerzenie repertuaru mikroprocesorów (m.in. o 16-bitowe), dla których będzie możliwe generowanie kodu maszynowego na podstawie tekstu źródłowego wyrażonego w języku SAPL.

Podziękowanie

Autor pragnie złożyć serdeczne podziękowanie dr inż. Adamowi Pawlakowi za merytoryczną pomoc przy poszczególnych etapach pracy nad językiem oraz wnikliwą analizę samego artykułu.

Równie gorąco dziękuję studentom: Jackowi Burghardowi, Stanisławowi Purolnikowi, Zdzisławowi Białemu i Wojciechowi Rozparze za ogromny wysi-

łek i poświęcenie włożone w implementację programu tłumaczącego oraz oprogramowania towarzyszącego.

LITERATURA

[1] Baldwin G.: Towards an Assembly Language Standard. IEEE Micro, 1984, Vol. 4, pp. 81-85.

[2] Biały Z., Rozpara W.: Asembler strukturalny dla mikrokomputera Sinclair ZX 81, Praca dyplomowa, Instytut Elektroniki, 1984.

[3] Burghard J., Purolnik S.: Asembler strukturalny dla systemu mikrokomputerowego z mikroprocesorem TMS 9900, Praca dyplomowa, Instytut Elektroniki, 1982.

[4] De Man J.A., Boute R.T: MISTRAL M , A Medium Level Programming Language for Microcomputers, Proceedings of the Euromicro Conference, 1978.

[5] Grabowski J., Koślacz S.: Podstawy i praktyka programowania mikroprocesorów, Wyd. I, WNT, Warszawa 1980.

[6] Muhlbacher J.R.: A tutorial introduction into structured microprocessor software development, Journal of Microcomputer Applications, 1982, nr 5, ss. 67-86.

[7] Nassi I., Schneiderman B.: Flowchart Techniques for Structuerd Programming, SIGPLAN Notices of the ACM, 1973, Vol. 8, nr 8, ss. 12-26.

Recenzent: Doc. dr inż. Ferdynand WAGNER

Wpłynęło do Redakcji 16.05.86

ЯЗЫК САПЛ - СТРУКТУРНЫЙ АССЕМБЛЕР ДЛЯ ПРОГРАММИРОВАНИЯ МИКРОПРОЦЕССОРОВ

Р е з ю м е

Представленный в настоящей работе язык программирования микропроцессоров САПЛ (Структураль Ассемблы Программинг Лангуич) даёт возможность разработки структурных программ с полным использованием свойств машинного языка.

Отказ от мнемоник как формы записи простых команд обеспечивает универсальность языка независимо от типа оборудования. Существенные элементы грамматики языка САПЛ даны формальным способом и разъяснены на примерах.

В заключении содержутся замечания по поводу вопросов программного обеспечения а также описание главных направлений по дальнейшим исследованиям языка.

SAPL - A STRUCTURAL ASSEMBLY LEVEL PROGRAMMING LANGUAGE FOR MICROPROCESSORS

S u m m a r y

In the paper the description of the microprocessors programming language SAPL is presented. This language enables structured programming with utilization of an assembly level features.

Structured programming is facilitated by:

- the hierarchy of problem description due to the language block structure,
- strongly typed data,
- a set of special language constructs for program execution control.

The elimination of mnemonic code as a notation of simple instructions, which results in more generalised notation assures equipment independence. The essential elements of SAPL language syntax have been presented in a formal manner and explained in examples.

In the concluding part a number of problems connected with the implementation is discussed, and the further ways of activities are marked out.