JĘZYK SYMBOLICZNY AWIK-2

Instrukcja programisty

sprawozdanie z realizacji zadania badawczego nr 2 tematu "Kompleksowe sterowanie ruchem ulicznym w podobszarze miejskim" (nr 43-005/13)

Opracował pod kierunkiem
dra inż. Jacka Martinka
mgr inż. LESZEK MASADYŃSKI
z udziałem
mgra Jerzego Bartoszka

Poznań, czerwiec 1977 rok

Zleceniodawca:	Instytut Automatyki Politechniki Poznańskiej jako podzlecenie w temacie nr 43-005/13 "Kompleksowe sterowanie ruchem ulicznym w podobszarze miejskim"
Zleceniobiorca:	Środowiskowy Ośrodek Informatyki Politechniki Poznańskiej
Wykonawcy:	Zespół Środowiskowego Ośrodka Informatyki Politechniki Poznańskiej

Kierownik zadania

dr inż. Jacek Martinek

(1) Kierownik
Środowiskowego Ośrodka
Informatyki
Doc. dr inż. Zbigniew Kierzkowski

SPIS TREŚCI

1. Wstęp

AWIK-2 jest językiem adresów symbolicznych komputera wirtualnego WIK-2. Komputer ten powstał przez programowe rozszerzenie języka wewnętrznego Systemu MERA-300. Organizację i szczegóły realizacji WIK-a-2 opisano w innym opracowaniu 1 .

W Systemie MERA-300 rolę języka adresów symbolicznych pełni język MOTIS, którego translator dostarcza producent Systemu. Ponieważ WIK-2 jest rozszerzeniem Systemu MERA-300, zatem przy konstruowaniu AWIK-a-2 założono, że język ten będzie rozszerzeniem MOTIS-a. Dlatego program napisany w języku MOTIS można tłumaczyć translatorem AWIK-a-2. Oczywiście AWIK-2 ma wiele nowych cech i udogodnień ułatwiających pisanie i tłumaczenie programów dla rozszerzonego komputera wirtualnego jakim jest WIK-2.

Translator AWIK-a jest translatorem skrośnym, to znaczy tłumaczenie odbywa się na innej maszynie (np. typu ODRA 1305). Dlatego przyjęto, że program w wyniku translacji zostaje przekształcony w tekst złożony ze znaków szesnastkowych (heksadecymalnych). Tekst ten jest zapisem stanu pamięci MERY-300 i może być do niej wprowadzony za pomocą prostego programu wczytującego. Dwa znaki heksadecymalne reprezentują jedno słowo 8-bitowe.

Program źródłowy w języku AWIK-2 jest ciągiem jednostek programu. Jednostką programu może być:

- rozkaz
- dyrektywa
- definicja nazwy
- parametr
- tekst
- komentarz

Rozkazy, teksty i parametry stają się po przetłumaczeniu elementami programu wynikowego, natomiast dyrektywy służą do definio-

wania nowych rozkazów oraz sterowania translacją. Nazwa może być etykietą lub zmienną translacji.

AWIK-2 pozwala definiować dowolne rozkazy i usuwać je z "biblioteki rozkazów".

Wprowadzono dwa typy wyrażeń arytmetycznych: o składni zwykłej (typ Z) i wyrażenia zgodne ze składnią języka MOTIS, które będziemy nazywać wyrażeniami typu M.

2. Słowa języka AWIK-2

W języku AWIK-2 istnieją trzy rodzaje słów:

- symbole
- liczby całkowite
- ograniczniki

Symbol jest to sznur złożony z liter i cyfr, rozpoczynający się literą i nie zawierający litery Q. Ilość znaków symbolu jest ograniczona jedynie długością wiersza, znaki powyżej sześćdziesiątego czwartego są zawsze ignorowane.

Przykłady symboli:

BUFOR

ALA2B1

P15793

natomiast nie są symbolami:

QUO

5CV

C-G

Liczby całkowite dzielą się na ósemkowe i dziesiętne. Liczba ósemkowa składa się z co najwyżej ośmiu cyfr ósemkowych (0,1, 2,...,7), z tym, że największą akceptowaną liczbą ósemkową jest 37777777_8.

Przykłady liczb ósemkowych

0

115

37777777

Liczba dziesiętna musi być poprzedzona znakiem # i składa się z cyfr dziesiętnych. Największa akceptowana liczba dziesiętna wynosi # 8388607 ($2^{23}-1$), ponieważ zapis binarny dowolnej liczby nie może przekraczać 23 bitów. Przykłady:

0

15

999999

Ogranicznik rozpoczyna się jednym z następujących znaków:

Q ; ( ) [ ] , * / + - : ⑪

i może zawierać dalej dowolny (także pusty) ciąg utworzony ze znaków =

Przykłady ograniczników

(

;

Q

:=

Jeśli przy czytaniu pojawi się znak, którego wystąpienie w bieżącym miejscu słowa jest niedopuszczalne, to znak ten sygnalizuje koniec słowa (i być może początek słowa następnego).

Zbiór słów można zdefiniować krótko w sposób następujący:

⟨słowo⟩ ::= ⟨symbol⟩ | ⟨liczba całkowita⟩ | ⟨ogranicznik⟩

⟨symbol⟩ ::= ⟨litera⟩ $\left\{\langle\text{litera}\rangle \mid \langle\text{cyfra}\rangle\right\}$

⟨liczba całkowita⟩ ::= ⟨liczba ósemkowa⟩ | ⟨liczba dziesiętna⟩

⟨liczba ósemkowa⟩ ::= $\left\{\langle\text{cyfra ósemkowa}\rangle\right\}_1^8$

⟨liczba dziesiętna⟩ ::= # $\left\{\langle\text{cyfra}\rangle\right\}_1^7$

⟨ogranicznik⟩ ::= Q | ; | (|) | [|] | , | * | / | + | - | : | ⑪ | - ⟨ogranicznik⟩ =

⟨litera⟩ ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|R|S|T|U|V|W|X|Y|Z

⟨cyfra⟩ ::= 0|1|2|3|4|5|6|7|8|9

⟨cyfra ósemkowa⟩ ::= 0|1|2|3|4|5|6|7

W tym zapisie $\{....\}_m^n$ jest skrótem wskazującym, że treść ujęta w nawiasy może wystąpić od m do n razy. Zapis $\{.....\}_0^\infty$ jest równoważny zapisowi $\{.....\}$

3. Ograniczniki jednostki programu, komentarze

Każda jednostka programu (z wyjątkiem definicji etykiety) musi być zakończona tzw. ogranicznikiem jednostki. Jest nim średnik, "zmiana wiersza" lub komentarz.

Komentarz jest ciągiem słów rozpoczynającym się od lewego nawiasu i zakończonym średnikiem lub "zmianą wiersza".

Formalnie można to zapisać następująco:

⟨ogr⟩ ::= ⟨ogr1⟩ | ⟨komentarz⟩

⟨ogr1⟩ ::= ; | (lf)

⟨komentarz⟩ ::= ($\{$⟨słowo 1⟩$\}$ ⟨ogr1⟩

⟨słowo 1⟩ oznacza dowolne słowo różne od ; i (lf)

Treść komentarza jest ignorowana przez translator.

Jeśli między dwoma ogranicznikami jednostki nie ma żadnego słowa, to uważa się, że drugi z tych ograniczników zamyka jednostkę pustą, którą się ignoruje.

4. Definiowanie nazw

Nazwa jest to obiekt języka AWIK-2, któremu nadaje się wartość. Składniowo nazwa jest albo symbolem albo składa się z litery Q i liczby zapisanej bezpośrednio za Q, lub po dowolnej licznie odstępów za Q. Liczbę wchodzącą w skład nazwy zbudowanej z litery Q i liczby będziemy nazywać numerem nazwy.

Ze względu na sposób definiowania będziemy rozróżniać nazwy będące:

- etykietami
- zmiennymi translacji

Specjalną rolę w procesie translacji pełni zmienna Q0 (można ją zapisywać także bez zera - czyli Q0 i Q są symbolami posiadającymi zawsze tę samą wartość). Wartością Q0 jest aktualny adres komórki w pamięci MERY 300, w której powinien zostać umieszczony przekład następnego rozkazu, parametru lub tekstu. Wartość Q0 jest automatycznie uaktualniana w trakcie translacji, można jej także nadawać wartość w definicji zmiennej.

Definicja zmiennej translacji posiada składnię:

⟨definicja zmiennej⟩ ::= ⟨nazwa⟩ := ⟨wyrażenie Z⟩ ⟨ogr⟩ |
⟨numer nazwy⟩ ⟨wyrażenie M⟩⟨ogr⟩

W pierwszym przypadku translator oblicza wartość wyrażenia i przypisuje tę wartość nazwie stojącej po lewej stronie znaku podstawiania. W drugim przypadku, który odpowiada definiowaniu zmiennej w języku MOTIS, translator oblicza wartość wyrażenia, dodaje do niej wartość zmiennej Q0 i wynik przypisuje nazwie o numerze podanym w definicji (czyli nazwie postaci Q numer nazwy). Numer nazwy musi być liczbą z przedziału ⟨0,63⟩ .

Przykłady definicji zmiennych translacji

```
Q1:= 0
GAMMA:= # 20
Q:= Q1 + 1
00  500L - Q
1  Q2 + 7S
2  Q1 + Q2 M; 3
```

W momencie napotkania przez translator definicji nazwy, wartość wyrażenia występującego w tej definicji powinna być możliwa do obliczenia. Wynika stąd, że wyrażenie nie może zawierać nazw, których jeszcze nie zdefiniowano.

Etykieta jest to nazwa miejsca w programie. Definicja etykiety ma postać:

⟨definicja etykiety⟩ ::= ⟨nazwa⟩ :

Nazwie występującej w definicji nazwy nadaje się wartość równą aktualnej wartości Q0.

Uwaga: Wyrażenie typu M może być w szczególnym przypadku puste, czyli w definicji zmiennej może wcale nie występować. Wówczas definicja zmiennej ma postać:

⟨numer nazwy⟩ ⟨ogr⟩

i nazwie postaci Q numer nazwy zostanie nadana wartość Q0. Zatem ten szczególny przypadek definiowania zmiennej jest równoważny definicji etykiety postaci:

Q ⟨numer nazwy⟩ :

Przykłady definiowania etykiet:

Q1:

ALFA:

21 (1f)

5. Wyrażenia

W AWIK-u-2 wprowadzono dwa typy wyrażeń:

- wyrażenia o składni zwykłej (typu Z)
- wyrażenia o składni wymaganej przez MOTIS (typu M).

Wyrażeniem o składni zwykłej, zwanym dalej wyrażeniem typu Z może być:

- liczba
- nazwa
- napis utworzony z wyrażeń typu Z przy pomocy jednego z operatorów + - * / i ewentualnie pary nawiasów ( ). Operatory te oznaczają odpowiednio: dodawanie, odejmowanie, mnożenie i dzielenie.

Wyrażenie typu Z może być poprzedzone znakiem + lub -. Przykłady wyrażeń typu Z:

+ 77

- #2+3/ALFA-Q1

-Q*(ALFA-2/BETA)

Wyrażeniem o składni wymaganej przez MOTIS jest:

- składnik MOTIS-a, który może być liczbą lub nazwą składającą się z litery Q i liczby całkowitej z przedziału 0,63 , będącej numerem nazwy. Jeśli numer nazwy jest równy zero, to można go opuścić,

- czynnik MOTIS-a-jest on ciągiem składników MOTIS-a, z których każdy może być poprzedzony znakiem + lub -. Znak ten obowiązuje dla wszystkich występujących za nim składników, aż do następnego znaku + lub -. Czynnik jest ograniczony jednym z operatorów S,L,M lub słowem niedopuszczalnym w wyrażeniach typu M. Wartość czynnika jest równa sumie wartości składników wziętych z obowiązującymi dla nich znakami, zmodyfikowanej odpowiednio przez operatory S,M,L, których znaczenie jest następujące:

S - sumę dzieli się przez 2^5 i odrzuca resztę

M - sumę dzieli się przez 2^8 i odrzuca resztę

L - wartości sumy nie zmienia się

brak operatora - wartością jest pięć najmniej znaczących bitów sumy

- ciąg czynników MOTIS-a - wartość wyrażenia jest równa sumie wartości czynników. Ciąg ten może być pusty i wówczas wartością wyrażenia jest 0.

Przykłady wyrażeń typu M

77

7 85

7-4 53Q1M

Q1 + Q2L - 5QM

Składnia wyrażeń jest następująca:

⟨wyrażenie⟩ ::= ⟨wyrażenie Z⟩|⟨wyrażenie M⟩

⟨wyrażenie Z⟩ ::= ⟨term ze znakiem⟩|⟨term ze znakiem⟩ ⟨operator1⟩ ⟨wyrażenie1⟩

⟨term ze znakiem⟩ ::= ⟨operator 2⟩ ⟨term⟩

⟨term⟩ ::= ⟨czynnik⟩| ⟨czynnik⟩ ⟨operator 3⟩ ⟨term⟩

⟨czynnik⟩ ::= ⟨liczba⟩|⟨nazwa⟩ | (⟨wyrażenie Z⟩)

⟨nazwa⟩ ::= ⟨symbol⟩| Q ⟨liczba 1⟩| Q

⟨operator 2⟩ ::= + | - |⟨puste⟩

⟨operator 1⟩ ::= + | -

⟨operator 3⟩ ::= * | /

⟨wyrażenie 1⟩ ::= ⟨term⟩|⟨term⟩⟨operator 3⟩⟨wyrażenie 1⟩

⟨wyrażenie M⟩ ::= ⟨puste⟩|⟨ciąg czynników⟩

⟨ciąg czynników⟩ ::= {⟨czynnik M⟩}⟨czynnik 1 M⟩

⟨czynnik M⟩ ::= ⟨ciąg składników⟩ ⟨operator M⟩

⟨ciąg składników⟩ ::= ⟨operator 2⟩ ⟨składnik M⟩{⟨operator 2⟩ ⟨składnik M⟩}

⟨składnik M⟩ ::= ⟨liczba⟩|Q ⟨liczba 1⟩ | Q

⟨operator M⟩ ::= S | L | M

⟨czynnik 1 M⟩ ::= ⟨czynnik M⟩|⟨ciąg składników⟩

⟨liczba 1⟩ oznacza dowolną liczbę z przedziału ⟨0, 63⟩.

6. Dyrektywy

Dyrektywy są jednostkami programu źródłowego, które sterują działaniem translatora języka AWIK-2.

W punkcie tym omówimy prawie wszystkie dyrektywy języka z wyjątkiem dyrektyw związanych z definiowaniem rozkazów, którym poświęcono punkt 7.

Składnia dyrektyw omawianych w bieżącym punkcie jest następująca:

⟨dyrektywa⟩ ::= * ⟨nazwa 1⟩ ⟨treść dyrektywy⟩ ⟨ogr⟩ | ⟨nazwa 2⟩ ⟨ogr⟩ | QM ⟨wyrażenie M⟩ ⟨ogr⟩

⟨treść dyrektywy⟩ ::= ⟨puste⟩ | , ⟨wyrażenie Z⟩

⟨nazwa 1⟩ ::= MODBEG | MODEND | GLOB | LAB | NLAB | RES | LIBRARY

⟨nazwa 2⟩ ::= ST | KS

(Dyrektywy MODBEG, MODEND, GLOB, LAB, NLAB mają pustą treść).

Omówimy najpierw znaczenie dyrektyw rozpoczynających się od znaku * i działania z nimi związane:

MODBEG - początek modułu programu. Wszystkie definicje nazw występujące wewnątrz modułu mają znaczenie lokalne. (Jeśli dotyczą nazw, które nie są zdefiniowane globalnie).

MODEND - koniec modułu programu.

GLOB - dyrektywa pozwalająca zdefiniować nazwę globalnie wewnątrz modułu. Definicja poprzedzona tą dyrektywą ma charakter globalny.

LAB - polecenie wyprowadzania definicji nazw w pierwszym przebiegu trabslatora. Informacje będą wyprowadzane na drukarkę wierszową

NLAB - unieważnienie polecenia wydanego dyrektywą LAB.

RES - treścią dyrektywy jest wyrażenie typu Z. Następuje obliczenie wartości wyrażenia i zarezerwowanie komórek (bajtów) w ilości określonej tą wartością. Komórki zostają wyzerowane.

LIBRARY - treścią dyrektywy jest wyrażenie typu Z; jego wartość oznaczmy przez n. Dyrektywa służy do odtwarzania starych definicji nazw oraz definicji rozkazów. Definicje występujące w programie są zapisywane w pierwszym przebiegu translacji do uporządkowanego zbioru, który będziemy nazywać tablicą translatora. Pojawienie się w definicji (nazwy lub rozkazu) nazwy już zdefiniowanej uaktualnia tablicę translatora, a więc w danym momencie obliczeń obowiązuje dla danej nazwy tylko jedna definicja. Odtworzenie pierwszych definicji pewnych nazw może się okazać konieczne, szczególnie jeśli są to nazwy rozkazów z "biblioteki". Dyrektywa LIBRARY odtwarza n pierwszych definicji zapamiętanych w tablicy translatora. Stan tablicy translatora (ilość zapamiętanych definicji) jest wyprowadzany na drukarkę na początku i końcu translacji.

Pozostałe dyrektywy

KS - powoduje zatrzymanie pracy translatora (sygnał = AWI2 HALTED:-3 na monitorze). Sposoby uruchomienia translatora omówione są w punkcie 9.

ST - sygnalizuje koniec programu i modułu programu. Każdy program w języku AWIK-2 musi być zakończony tą dyrektywą.

QM - dyrektywa usuwająca z tablicy translatora definiuje nazw typu Q numer nazwy , których wartości są większe od wartości wyrażenia typu M będącego treścią dyrektywy.

7. Dyrektywy związane z definiowaniem rozkazów

W języku AWIK-2 istnieje specjalna dyrektywa definiowania rozkazu. Dzięki niej można rozszerzyć zbiór rozkazów języka akceptowanych przez translator o nowy rozkaz o dowolnej (w ramach pewnego schematu) składni i o znaczeniu określanym przez tą dyrektywę.

Zakłada się, że rozkaz posiada swoją własną, odrębną nazwę oraz składa się z ciągu parametrów. Parametry te określają treść pól, których rozmiar i rozmieszczenie wynikają z informacji podanych w definicji rozkazu.

Składnia dyrektywy definicji rozkazu jest następująca:

⟨dyrektywa definicji rozkazu⟩ ::=

* DEF, ⟨nazwa rozkazu⟩ := ⟨wyrażenie Z⟩ { , ⟨długość pola⟩ , ⟨typ pola⟩ }$_1^\infty$ ⟨ogr⟩

⟨nazwa rozkazu⟩ ::= ⟨symbol⟩

⟨długość pola⟩ ::= ⟨wyrażenie Z⟩

⟨typ pola⟩ ::= MT | IT | XT | CT | LT | [⟨wyrażenie Z⟩] | ⟨puste⟩

Definicja wiąże nazwę rozkazu z wartością (wyrażenie typu Z za znakiem :=), która jest kodem rozkazu. Następnie podaje się kolejno określenia długości pola w bitach i typu pola dla każdego pola przekładu rozkazu. Znaczenie słów określających typ pola jest następujące:

MT - parametr rozkazu związany z tym polem jest wyrażeniem typu M

IT - pole, w którym umieszcza się bit modyfikacji pośredniej rozkazu. Modyfikację tę wykonuje się nazwą I. Jeśli I pojawi się w rozkazie po jego nazwie, to bit modyfikacji jest równy 1, w przeciwnym razie równa się 0

XT - parametrem związanym z tym polem jest jedna z nazw X0, X1, X2, X3 (lub brak tej nazwy). Jeśli w rozkazie pojawi się jedna z tych nazw, to podczas tłumaczenia rozkazu w pole wpisuje się odpowiednio 4,5,6,7 (lub w przeciwnym razie 0)

CT - pole, w które wpisuje się kod rozkazu

LT - wartość wpisywana w to pole powstaje przez obliczenie wartości wartości wyrażenia typu Z podanego w rozkazie i przesunięcie cyklicznie trzynastu najmniej znaczących bitów tej wartości o pięć pozycji w prawo.

Ponadto, jeśli jako typ pola pojawi się napis [⟨wyrażenie Z⟩] - to podczas translacji w pole wpisuje się wartość wyrażenia ujętego w nawiasy kwadratowe

⟨puste⟩ - to znaczy, że pole związane jest z parametrem określanym wyrażeniem typu Z.

Rozkaz zdefiniowany dyrektywą DEF ma składnię:

⟨rozkaz zdefiniowany⟩ ::= ⟨nazwa rozkazu⟩ {,⟨parametr⟩} ⟨ogr⟩

⟨parametr⟩ ::= ⟨wyrażenie M⟩ | ⟨wyrażenie Z⟩ | I | X0 | X1 | X2 | X3

Oczywiście nazwa rozkazu oraz rodzaj parametru związanego z

każdym polem muszą być zgodne z dyrektywą definiującą ten rozkaz. Na przykład po zdefiniowaniu rozkazu INSTRUKCJA dyrektywą:

```
*DEF INSTRUKCJA:= 5, 3CT, 4, 1 [0] , 1IT, 3XT, 4
```

rozkaz

```
INSTRUKCJA , 15, I, X2, 2*3
```

będzie miał następujący przekład w postaci binarnej

5_8 15_8

1	0	1	1	1	0	1	0
1	1	1	0	0	1	1	0

IT XT 2*3 = 6

czyli translator wygeneruje jako przekład rozkazu ciąg znaków:

BA

E6

Rozkaz

```
INSTRUKCJA 15, X2, 2*3
```

zostanie przetłumaczony na

BA

66

8. Standardowa biblioteka rozkazów

Użytkownik translatora AWIK-a-2 może wykorzystywać standardowy zestaw dyrektyw definiujących rozkazy, zwanych biblioteką. Biblioteka ta zawiera definicje rozkazów MERY 300 oraz podstawowych rozkazów wirtualnych komputera WIK-2 [1].

8.1. Rozkazy MERY 300

Dla rozkazów MERY 300 wprowadzono dwa rodzaje składni. Pierwszy z nich jest zgodny ze składnią rozkazów języka MOTIS, drugi różni się nazwami rozkazów oraz składnią wyrażeń arytmetycznych. Przy omawianiu semantyki rozkazów zastosujemy następujące oznaczenia:

A - akumulator (8 bitów)

Z - rejestr strony zerowej (1 bit)

CI - rejestr skoku warunkowego (1 bit)

P - rejestr przeniesienia (1 bit)

S - rejestr strony (8 bitów)

PAO(N) - komórka pamięci operacyjnej (8 bitów) o adresie równym N

LR - licznik rozkazów (13 bitów)

KL - klucze klawiatury MERY 300 (8 bitów)

WP - rejestr maski przerwań (4 bity)

$|X|$ - zawartość rejestru o nazwie X

X_{N-M} - bity o N do M rejestru o nazwie X

$|X| \rightarrow Y$ - prześlij zawartość rejestru o nazwie X do rejestru o nazwie Y

Rozkazy MERY 300 można podzielić na cztery grupy:

a) rozkazy adresowe krótkie

b) rozkazy adresowe długie

c) rozkazy bezadresowe krótkie

d) rozkazy bezadresowe długie

Przekład rozkazów krótkich zajmuje 1 komórkę (1 bajt) a rozkazów długich 2 komórki.

a. Rozkazy adresowe krótkie MERY 300

Składnia rozkazów adresowych krótkich MERY 300 jest następująca

⟨rozkaz adresowy krótki⟩ ::= ⟨nazwa 1⟩⟨wyrażenie M⟩ ⟨ogr⟩ |
⟨nazwa 2⟩,⟨wyrażenie Z⟩ ⟨ogr⟩

⟨nazwa 1⟩ ::= PS | PZ | ML | DP | WP | WW

⟨nazwa 2⟩ ::= AM | SZ | LM | IM | EP | IO

Z wartości wyrażenia M (lub wyrażenia Z) występującego w rozkazie bierze się pięć najmniej znaczących bitów. Bity te określają numer słowa na stronie (oznaczany przez D). Adres ADR słowa pamięci, które bierze udział w wykonywaniu rozkazu oblicza się według wzoru:

$ADR = |S| * 2^5 + |PAO(D)|$ gdy $20_8 \leqslant D \leqslant 27_8$ i $Z = 1$

lub

$ADR = (1 - Z) * |S| * 2^5 + D$ w przeciwnym wypadku. Semantykę rozkazów adresowych krótkich podano w poniższej tabeli:

nazwa 1	nazwa 2	funkcja
DS	AM	$\|A\| + \|PAO(ADR)\| \longrightarrow A,P$; jeśli nadmiar to $0 \longrightarrow CI$, w przeciwnym wypadku $1 \longrightarrow CI$
PZ	SZ	$\|A\| \longrightarrow PAO(ADR)$; $0 \longrightarrow A$
ML	LM	$\|A\| \wedge \|PAO(ADR)\| \longrightarrow A$
DP	IM	$\|PAO(ADR)\| + 1 \longrightarrow PAO(ADR)$; jeśli nadmiar to $0 \longrightarrow CI$, w przeciwnym wypadku $1 \longrightarrow CI$
WP	EP	Wykonaj rozkaz (podprogram)
WW	IO	Rozkaz inicjujący transmisję we-wy. Jeśli transmisja została przyjęta, to $1 \longrightarrow CI$, w przeciwnym wypadku $0 \longrightarrow CI$

b. Rozkazy adresowe długie

W skład tej grupy wchodzi tylko jeden rozkaz.

Składnia:

⟨rozkaz adresowy długi⟩ ::= SK ⟨wyrażenie M⟩ ⟨ogr⟩ XX ⟨wyrażenie M⟩ ⟨ogr⟩ | JP , ⟨wyrażenie Z⟩ ⟨ogr⟩

Jeśli

$$|CI| = 1$$

to

$$|LR| + 2 \longrightarrow LR$$

W przeciwnym wypadku

$$ADRES \longrightarrow LR$$

gdzie ADRES oblicza się następująco:

- składnia z nazwą SK:

  ADRES = "osiem najmniej znaczących bitów drugiego argumentu" $* 2^5$ + "pięć najmniej znaczących bitów pierwszego argumentu"

- składnia z JP:

  ADRES = "trzynaście najmłodszych bitów argumentu"

c. Rozkazy bezadresowe krótkie

Składnia:

⟨rozkaz bezadresowy krótki⟩ ::= ⟨nazwa 3⟩ ⟨ogr⟩ | ⟨nazwa 4⟩ ⟨ogr⟩

⟨nazwa 3⟩ ::= PR | WW | PS | PO | SD | ZA | NA | DA | DN | LC | AL | AP | KA | AS | SA | SS | ZZ | LZ | ZC | UW | PP | AZ | AD | NN | SP

⟨nazwa 4⟩ ::= PI | IR | SC | RT | WT | ZA | NA | IA | AO | CL | LL | LR | KA | AS | SA | IS | ZZ | LZ | ZC | LW | TP | TZ | TN | NN | SP

Sematyka rozkazów bezargumentowych podana jest w tabeli:

nazwa 3	nazwa 4	funkcja
PR	PI	przerwanie programowe
PW	IR	powrót z przerwania: $\|PAO(0)_{3-7}\| \longrightarrow LR_{8-12}$; $\|PAO(1)\| \longrightarrow LR_{0-7}$; $\|PAO(0)_0\| \longrightarrow P$; $\|PAO(0)_1\| \longrightarrow CI$; $\|PAO(0)_2\| \longrightarrow Z$
PS	SC	pamiętaj ślad: $\|LR_{8-12}\| \longrightarrow PAO(5)_{3-7}$; $\|LR_{0-7}\| \longrightarrow PAO(6)$; $\|P\| \longrightarrow PAO(5)_0$; $\|CI\| \longrightarrow PAO(5)_1$; $\|Z\| \longrightarrow PAO(5)_2$
PO	RT	powrót na podstawie śladu: $\|PAO(5)_{3-7}\| \longrightarrow LR_{8-12}$; $\|PAO(6)\| \longrightarrow LR_{0-7}$; $\|PAO(5)_0\| \longrightarrow P$; $\|PAO(5)_1\| \longrightarrow CI$; $\|PAO(5)_2\| \longrightarrow Z$;
SD	WT	czekaj na przerwanie
ZA	ZA	$0 \longrightarrow A$
NA	NA	$\|A\| \longrightarrow A$
DA	IA	$\|A\| + 1 \longrightarrow A$
DN	AO	$\|A\| + \|P\| \longrightarrow A,P$; gdy po wykonaniu rozkazu wystąpi nadmiar, to $0 \longrightarrow CI$, w przeciwnym wypadku $1 \longrightarrow CI$
LC	CL	przesuń $\|A\|$ cyklicznie w lewo o jedną pozycję
AL	LL	przesuń $\|A\|$ arytmetycznie w lewo o jedną pozycję
AP	LR	przesuń $\|A\|$ w prawo o jedną pozycję: $\|A\|/2 \longrightarrow A_{1-7}$; $\|P\| \longrightarrow A_0$; $0 \longrightarrow P$
KA	KA	$\|KL\| \longrightarrow A$
AS	AS	$\|A\| \longrightarrow S$
SA	SA	$\|S\| \longrightarrow A$

c.d. tabeli

nazwa 3	nazwa 4	funkcja
SS	IS	$\|S\| + 1 \longrightarrow S$
ZZ	ZZ	$0 \longrightarrow Z$
LZ	LZ	$1 \longrightarrow Z$

d. Rozkazy bezadresowe długie

Rozkazy te mają składnię:

⟨rozkaz bezadresowy długi⟩ ::=

⟨nazwa 5⟩ ⟨ogr⟩ XX ⟨wyrażenie M⟩ ⟨ogr⟩ |

⟨nazwa 6⟩ , ⟨wyrażenie Z⟩ ⟨ogr⟩

⟨nazwa 5⟩ ::= US | TM

⟨nazwa 6⟩ ::= LS | TA

Niech OB oznacza wartość ośmiu najmniej znaczących bitów wyrażenia występującego w rozkazie. Znaczenie rozkazów tej grupy podano w poniższej tabeli:

nazwa 5	nazwa 6	funkcja
US	LS	$\|OB\| \longrightarrow S$; $\|LR\| + 2 \longrightarrow LR$
TM	TA	$\|OB\| \oplus \|A\| \longrightarrow A$; $\|LR\| + 2 \longrightarrow LR$
		jeśli po wykonaniu rozkazu $\|A\| = 0$
		to $0 \longrightarrow CI$, w przeciwnym wypadku
		$1 \longrightarrow CI$

8.2. Rozkazy wirtualne

a. Utwórz rozkaz wirtualny - CVI

Składnia:

⟨rozkaz utworzenia rozkazu⟩ ::= CVI, ⟨kod⟩ , ⟨adr⟩ ⟨ogr⟩

⟨kod⟩ ::= ⟨wyrażenie Z⟩

⟨adr⟩ ::= ⟨wyrażenie Z⟩

Rozkaz CVI tworzy nowy rozkaz wirtualny o kodzie równym wartości wyrażenia ⟨kod⟩ , natomiast wyrażenie ⟨adr⟩ określa adres początku programu realizującego ten rozkaz.
Rozkaz CVI zajmuje cztery komórki.

b. Usuń rozkaz wirtualny - DVI

Składnia:

⟨rozkaz usuwania rozkazu⟩ ::= DVI, ⟨kod⟩ ⟨ogr⟩

Rozkaz DVI usuwa rozkaz wirtualny o kodzie równym wartości wyrażenia ⟨kod⟩

Rozkaz DVI zajmuje dwie komórki.

c. Utwórz procesor - CRP

Składnia:

⟨rozkaz tworzenia procesora⟩ ::= CRP, ⟨ni⟩ , ⟨prior⟩ , ⟨w⟩ , ⟨adr⟩ ⟨ogr⟩

⟨ni⟩ ::= ⟨wyrażenie Z⟩

⟨prior⟩ ::= ⟨wyrażenie Z⟩

⟨w⟩ ::= ⟨wyrażenie Z⟩

Rozkaz CRP tworzy nowy procesor w komputerze WIK. Nazwa (numer) tego procesora jest równa wartości wyrażenia ⟨ni⟩ Utworzony procesor otrzymuje priorytet równy wartości wyrażenia ⟨prior⟩ . Pozostałe argumenty rozkazu określają stany początkowe rejestrów WP i LR tworzonego procesora. Stanem początkowym procesora jest stan zatrzymania.
Rozkaz CRP zajmuje pięć komórek.

d. Usuń procesor - DEP

Składnia:

⟨rozkaz usuwania procesora⟩ ::= DEP, ⟨ni⟩ ⟨ogr⟩

Rozkaz DEP usuwa procesor o nazwie równej wartości wyrażenia ⟨ni⟩ . Przed usunięciem procesor musi być w stanie

zatrzymania lub zatrzymania i czekania.

Rozkaz DEP zajmuje dwie komórki.

e. Startuj procesor - ST1

Składnia:

⟨rozkaz startuj procesor⟩ ::= ST1, ⟨ni⟩ ⟨ogr⟩

Rozkaz ten powoduje przejście procesora o nazwie ⟨ni⟩ w stan przetwarzania (jeżeli był w stanie zatrzymania) lub w stan czekania (jeżeli był w stanie zatrzymania i czekania)

Rozkaz ST1 zajmuje dwie komórki.

f. Startuj procesor od podanego adresu - ST2

Składnia:

⟨rozkaz ST2⟩ ::= ST2, ⟨ni⟩ , ⟨adr⟩ ⟨ogr⟩

Rozkaz ten powoduje umieszczenie w rejestrze LR procesora o nazwie ⟨ni⟩ adresu ⟨adr⟩ i przejście procesora w stan przetwarzania. Procesor o tej nazwie musi być w stanie zatrzymania.

Rozkaz ST2 zajmuje cztery komórki.

g. Zatrzymaj procesor - STP

Składnia:

⟨rozkaz zatrzymania procesora⟩ ::= STP, ⟨ni⟩ , ⟨adr⟩ ⟨ogr⟩

Po wykonaniu rozkazu procesor o nazwie ⟨ni⟩ przechodzi w stan zatrzymania lub zatrzymania i czekania (jeżeli był w stanie czekania).

Rozkaz STP zajmuje dwie komórki.

h. Generuj przerwanie - GIN

Składnia:

⟨rozkaz generuj przerwanie⟩ ::= GIN

Procesor wykonujący rozkaz generuje przerwanie.

Rozkaz zajmuje dwie komórki.

i. Czekaj na przerwanie - WIN

Składnia:

⟨rozkaz czekaj na przerwanie⟩ ::= WIN, ⟨ik⟩ , ⟨nik⟩ ⟨ogr⟩.

⟨ik⟩ ::= ⟨wyrażenie Z⟩

⟨nik⟩ ::= ⟨wyrażenie Z⟩

Po wykonaniu tego rozkazu procesor zostaje umieszczony w kolejce, w której oczekuje na k-krotne (gdzie k jest wartością wyrażenia ⟨nik⟩) przerwanie o numerze określonym przez ⟨ik⟩ .

Rozkaz WIN zajmuje cztery komórki.

j. Wyślij komunikat - SME

Składnia:

⟨rozkaz wyślij komunikat⟩ ::= SME, ⟨ni⟩ , ⟨f⟩ , ⟨adr⟩ ⟨ogr⟩

⟨f⟩ ::= ⟨wyrażenie Z⟩

Rozkaz SME umożliwia wysłanie komunikatu ze znacznikiem równym wartości ⟨f⟩ do procesora określonego przez ⟨ni⟩ . Ilość znaków w treści komunikatu jest określona zawartością słowa pamięci o adresie wskazanym przez ⟨adr⟩ . Treść komunikatu znajduje się w kolejnych słowach pamięci poczynając od adresu ⟨adr⟩ + 1.

Rozkaz SME zajmuje cztery komórki.

k. Czekaj na następny komunikat - WNM

Składnia:

⟨rozkaz czekaj na następny komunikat⟩ ::= WNM, ⟨ni⟩ , ⟨f⟩⟨ogr⟩

Procesor, wykonując rozkaz WNM sprawdza, czy w kolejce skierowanych do niego komunikatów znajduje się komunikat ze znacznikiem równym wartości ⟨f⟩ od procesora określonego przez ⟨ni⟩ . Jeśli komunikat nadesłano, to wprowadza się go

do kolejki komunikatów zbadanych. W przeciwnym razie procesor przechodzi w stan czekania.

Zerowa wartość wyrażenia ⟨f⟩ lub ⟨ni⟩ oznacza czekanie na komunikat odpowiednio z dowolnym znacznikiem lub od dowolnego nadawcy.

Rozkaz WNM zajmuje trzy komórki.

l. Czytaj komunikat - RME

Składnia:

⟨rozkaz czytaj komunikat⟩ ::= RME, ⟨ni⟩ , ⟨f⟩ , ⟨adr⟩⟨ogr⟩

Rozkaz RME powoduje, że nazwa nadawcy komunikatu, wskaźnik określający ilość znaków w treści oraz treść zostają wpisane do kolejnych słów pamięci poczynając od adresu wskazanego przez ⟨adr⟩ . Przed wykonaniem rozkazu RME komunikat wskazywany tym rozkazem musi znajdować się w kolejce komunikatów, które nie były badane, skierowanych do procesora wykonującego rozkaz. W przeciwnym razie zostanie wykryty błąd. Zerowe wartości wyrażeń ⟨f⟩ i ⟨ni⟩ mają identyczne znaczenie jak w rozkazie WNM.

Rozkaz RME zajmuje cztery komórki.

m. Sprawdź czy nadesłano komunikat - TME

Składnia:

⟨rozkaz sprawdź komunikat⟩ ::= TME, ⟨ni⟩ , ⟨f⟩ , ⟨adr⟩ ⟨ogr⟩

Rozkazem TME sprawdza się, czy w kolejkach komunikatów skierowanych do procesora wykonującego rozkaz (w kolejce komunikatów, których nie badano i w kolejce komunikatów zbadanych) znajduje się komunikat ze znacznikiem równym wartości ⟨f⟩ nadesłany przez procesor określany przez ⟨ni⟩ . Jeśli tak jest, to w rejestrze LR procesora umieszcza się adres wskazany przez ⟨adr⟩ . W przeciwnym razie

zawartość LR zostaje powiększona o cztery, czyli przechodzi się do wykonania następnego rozkazu.

Testowanie kolejek komunikatów rozpoczyna się od kolejki komunikatów zbadanych i wykonuje się do znalezienia odpowiedniego komunikatu lub osiągnięcia końca kolejki komunikatów niezbadanych. Komunikaty z tej ostatniej kolejki, które się testuje, zostają z niej usunięte a wprowadza się je do kolejki komunikatów zbadanych. Zerowe wartości wyrażeń ⟨f⟩ , ⟨ni⟩ mają identyczne znaczenie jak w rozkazie WNM.
Rozkaz TME zajmuje cztery komórki.

n. Wirtualne rozkazy adresowe

Składnia:

⟨wirtualny rozkaz adresowy⟩ ::= ⟨nazwa rozkazu adresowego⟩ $\{,I\}_0^1$ $\{,\langle X\rangle\}_0^1$, ⟨adr⟩ ⟨ogr⟩

⟨nazwa rozkazu adresowego⟩ ::= VAM | VSZ | VLM | VIM | VJP | VEP

⟨X⟩ ::= X0 | X1 | X2 | X3

Wirtualne rozkazy adresowe mają to samo znaczenie co odpowiednie rozkazy MERY 300. Różnica polega na sposobie adresacji pamięci operacyjnej, który nie uwzględnia podziału na strony, umożliwia natomiast stosowanie modyfikacji pośredniej (I) oraz indeksowej (X0, X1, X2, X3). Jeżeli w rozkazie należy wykonać obie modyfikacje, to najpierw wykonywana jest modyfikacja indeksowa. Nazwy X0...X3 oznaczają wykorzystywany rejestr X-modyfikacji.
Wirtualny rozkaz adresowy zajmuje cztery komórki.

o. Wykonaj rozkaz wirtualny z pośrednio wskazanymi argumentami - EXI

Składnia:

⟨rozkaz EXI⟩ ::= EXI, ⟨kod⟩$\{,⟨X⟩\}_0^1$ ⟨adr⟩

⟨kod⟩ ::= ⟨wyrażenie Z⟩

Rozkaz ten umożliwia wykonanie rozkazu wirtualnego o kodzie równym wartości wyrażenia ⟨kod⟩ z argumentami zapamiętanymi w kolejnych słowach pamięci operacyjnej poczynając od adresu określanego przez ⟨adr⟩ (ewentualnie modyfikowanego). Dla rozkazów opisanych powyżej przyjęto, że w jednym słowie znajduje się tylko jeden argument. Nie dotyczy to argumentu ⟨adr⟩ który pamiętany jest w dwu słowach, oraz argumentów I,X0,X1, X2,X3, które zawsze są pamiętane w jednym słowie na bitach 4 do 7.

Rozkaz EXI zajmuje cztery komórki.

9. Parametry i teksty

Jednostka programu zwana parametrem posiada następującą składnię:

⟨parametr⟩ ::= ⟨parametr 1-bajtowy⟩ | ⟨parametr 2-bajtowy⟩

⟨parametr 1-bajtowy⟩ ::= XX ⟨wyrażenie M⟩ ⟨ogr⟩

PAR1, ⟨wyrażenie Z⟩ ⟨ogr⟩

⟨parametr 2-bajtowy⟩ ::= PAR2, ⟨wyrażenie Z⟩ ⟨ogr⟩

Umieszczenie parametru w programie źródłowym powoduje wpisanie wartości wyrażenia występującego w tej jednostce programu do jednej lub dwóch komórek pamięci.

Składnia tekstu jest następująca:

⟨tekst⟩ ::= , ⟨znak⟩ ⟨ogr⟩

⟨znak⟩ ::= 0 | 1 ... 9 | : | ; | < | = | > | ? | spacja | ! | " | # |
£ | % | & | ' | (|) | * | + | − | . | / | @ | A | B ... Z |
[| $ |] | ↑ | ←

Każdy znak tekstu zostaje umieszczony w jednym słowie pamięci.
Przykład:

```
,M
,O
,M
,I
,K
,
,8
,-
,B
```

Do kolejnych komórek pamięci zostanie umieszczony tekst

MOMIK 8-B

10. Sygnalizacja błędów wykrywanych przez translator

Wszystkie informacje o błędach są wyprowadzane na drukark wierszową. Translator wykrywa trzy typy błędów:

- błędy składniowe
- błędy leksykalne
- przekroczenia ilościowe translatora

Opis wykrytego błędu składniowego składa się z następujących elementów:

a. treść wiersza, w którym wykryty został błąd

b. napis

SYNTACTIC ERROR n

Liczba n oznacza rodzaj błędu składniowego i należy ją interpretować następująco:

n	rodzaj błędu
1	nierozpoznana jednostka programu
2	zła postać definicji nazwy lub próba użycia niezdefiniowanego rozkazu
3	niedopuszczalna postać ogranicznika jednostki programu
4	błędna postać czynnika w wyrażeniu Z
5	brak nawiasu zamykającego w wyrażeniu Z
6	użyto niezdefiniowanej nazwy w wyrażeniu
7	zarezerwowane
8	zarezerwowane
9	błąd w dyrektywie

Po wykryciu błędu, oraz wydruku napisu translator ignoruje wszystkie słowa aż do najbliższego ogranicznika jednostki programu.

Użycie niewłaściwego znaku w danym miejscu programu jest sygnalizowane przez translator wydrukiem tego znaku poprzedzonego napisem

INCORRECT CHAR

Źle użyty znak jest ignorowany przez translator.

Przekroczenie pojemności tablicy translatora jest sygnalizowane wydrukiem napisu:

WARNING: TO MANY DEFINITIONS

Pojawienie się dalszych definicji może spowodować uszkodzenie programu translatora. Inne przekroczenia ilościowe mogą spowodować wykrycie błędów przez programy śledzące dołączane automatycznie przez kompilatory FORTRAN-u.

Opis tych błędów jest zgodny z odpowiednimi podręcznikami.

11. Sposób wykorzystania translatora

Pracę bezpośrednią programu translatora AWIK-2 zapewnia system operacyjny EXECUTIV. System ten może przyjmować zlecenia od operatora i wyprowadzać na monitor informacje o przyczynach zatrzymania programu. W tym trybie pracy program w AWIK-u-2 powinien być przygotowany na kartach perforowanych (lub taśmie perforowanej). Translator AWIK-a-2 wykorzystuje czytnik kart (lub czytnik taśmy), drukarkę wierszową i perforator taśmy. Poniżej podane są kolejne czynności podejmowane przez operatora.

a) Wprowadzenie do pamięci programu translatora FORTRAN-u zleceniem:

fi # xfat # tape

System powinien odpowiedzieć

XFAT HALTED: - LD

b) Założenie tasiemki z translatorem AWIK-a-2 do czytnika i uruchomienie translacji FORTRAN-u zleceniem:

go # xfat 20

Koniec translacji jest sygnalizowany wyprowadzeniem napisu

AWI2 HALTED: - LD

c) Założenie kart (lub taśmy) z programem w AWIK-u-2 i uruchomienie translatora tego języka zleceniem:

go # awi2 20

Zatrzymanie pracy translatora sygnalizowane jest jednym z napisów podanych w tabeli:

napis	interpretacje
# AWI2 HALTED:-1	Natrafiono w tłumaczonym programie na znak zapytania występujący poza jednostką "tekst". Znak ten może być np. znakiem końca taśmy.
# AWI2 HALTED:-2	Zakończył się pierwszy przebieg i nie zostały wykryte błędy. Można teraz rozpocząć przebieg drugi, oczywiście po powtórnym założeniu kart (taśmy) z programem w AWIK-u-2
# AWI2 HALTED:-3	Natrafiono na dyrektywę KS
# AWI2 HALTED:-4	Zakończono pierwszy przebieg i wykryto błędy. Informacja o nich wyprowadzana została na drukarkę
# AWI2 HALTED:-5	Zakończył się drugi przebieg i nie zostały wykryte błędy
# AWI2 HALTED:-6	Zakończył się drugi przebieg, w trakcie którego wykryto błędy

Po każdym zatrzymaniu można uruchomić translator przy pomocy zlecenia

go # awi2

które w przypadku zatrzymania 4,5 i 6 powoduje usunięcie programu translatora z pamięci.

Po wszystkich zatrzymaniach można uruchomić 6translator od początku zleceniem:

go # awi2 20

Translator w przypadku zatrzymań 1,3,5,6 stnrtuje z tablicą translatora zawierającą wszystkie napotkane w trakcie pierwszego przebiegu definicje. W pozostałych przypadkach stan tablicy translatora odpowiada stanowi tej tablicy na początku pierwszego przebiegu.

12. Przykład programu w języku AWIK-2

Poniższy program czyta znak napisany na klawiaturze drukarki mozaikowej, a następnie ten znak wyprowadza. Zakłada się, że została wczytana "biblioteka" rozkazów MOTIS-a (34 definicje)

```
(PRZYKLAD PROGRAMU W JEZYKU AWIK-2)
QMQ
00 10000-QL
*LAB
*DEF, WIN:=3, #8 [#224] , 4CT, 4 [0], #8, #8
01
WIN, #18, 1 (CZEKAJ NA PRZERWANIE)
WW 15          (CZYTANIE ZNAKU)
SK Q1
XX Q1S
PZ 30
WW:=20
*DEF, IO:=6, 3CT, 5 (DEFINICJA NOWEGO ROZKAZU WE-WY)
02
IO, WW+15 (PISANIE ZNAKU)
SK Q3
XX Q3S
WIN, #18, 1 (CZEKAJ NA PRZERWANIE)
ZC
SK Q1
XX Q1S
03
*LIBRARY, #34
```

```
WW 15    (CZYTANIE SLOWA STANU JEDNOSTKI STERUJACEJ)
ZA
DS 30
ZC
SK Q2
XX Q2S
(KONIEC PRZYKLADU)
ST
```

Literatura

1 J. Bartoszek, Komputer wirtualny WIK-2, Raport Środowiskowego Ośrodka Informatyki Politechniki Poznańskiej, Poznań, czerwiec 1977.