Metody cyfrowej analizy sygnałów
wibroakustycznych — Ossolineum 1979
ISBN 83-04-00172-1

Ryszard Tadeusiewicz

2. ELEMENTY PROGRAMOWANIA MASZYN CYFROWYCH

Maszyny cyfrowe można programować na dwu poziomach. Pierwszy poziom, bardziej abstrakcyjny, polega na posługiwaniu się językiem maksymalnie zbliżonym do kodów, jakie są przesyłane wewnątrz samej maszyny cyfrowej między poszczególnymi jej podzespołami. Będziemy go nazywać poziomem maszynowo zorientowanym, gdyż w tym przypadku człowiek musi nauczyć się języka wygodnego dla maszyny. Drugi poziom, na ogół wygodniejszy, polega na posługiwaniu się językiem maksymalnie zbliżonym do naturalnego, a zatem program komputera zawiera zdania języka potocznego i fragmenty wzorów matematycznych. W tym przypadku możemy mówić o programowaniu zorientowanym na użytkownika, albo częściej, o programowaniu problemowo zorientowanym.

Wydaje się, że nawet w nauce podstaw programowania nie można skupić się wyłącznie na jednym z wymienionych wyżej poziomów, nie wspominając o drugim. Niewątpliwie w wielu praktycznych zagadnieniach korzystniejsze jest posługiwanie się wyższym poziomem problemowo zorientowanego programowania. Niemniej w przypadku, kiedy zależy nam na napisaniu programu odznaczającego się wysoką efektywnością, to znaczy dużą szybkością liczenia, małą pamięcią zajętą przez program, zwłaszcza jeśli potrzebujemy oprogramować minikomputer współpracujący z aparaturą pomiarową - jest konieczne zejście na poziom programowania maszynowo zorientowanego.

W dalszym ciągu naszych rozważań skupimy się na dwu konkretnych przykładach, dotyczących wymienionych wyżej metod programowania. Przykładem maszynowo zorientowanej metody programowania będzie programo-

wanie w assemblerze DAS (podstawowy język programowania dla minikomputerów Varian), w które są wyposażone zestawy pomiarowe firmy Brüel i Kjaer. Natomiast jako język problemowo zorientowany zostanie omówiony bardzo obecnie popularny język FORTRAN. Jest to język ogólny, czyli może być używany zarówno do obliczeń naukowych oraz technicznych, jak i do przetwarzania danych. W translatory FORTRAN-u wyposażone są obecnie niemal wszystkie maszyny cyfrowe, w szczególności można programować posługując się tym językiem w maszynach cyfrowych serii Odra 1300, Riad, a także w większych minikomputerach (np. Mera 400 lub Varian, w konfiguracji z powiększoną pamięcią).

2.1. Programowanie w assemblerze DAS

2.1.1. Informacje podstawowe

Program napisany w języku DAS tłumaczony jest na język wewnętrzny maszyny przez firmowy program nazywany assemblerem. Większość instrukcji języka DAS ma swoje bezpośrednie odpowiedniki wśród kodów wewnętrznych komputera, w związku z czym wspomniany proces tłumaczenia jest stosunkowo prosty. Instrukcja języka DAS dzieli się na cztery części: etykietę, kod operacji, operand i (ewentualnie) komentarz. Poszczególne części mogą być różnie wykorzystywane, zależnie od typu rozkazu (co będzie szczegółowo dalej omówione), ogólnie jednak (dla większości instrukcji) mają one następujące znaczenie.

Etykieta jest oznacznikiem instrukcji, pozwalającym na wyodrębnienie tej instrukcji spośród wszystkich innych. Z tego powodu etykieta musi być unikalna. Może składać się z jednego lub więcej znaków alfanumerycznych, przy czym maksymalnie do czterech znaków, a pierwszy zawsze musi być literą. Nie wszystkie instrukcje programu muszą mieć etykiety, możliwe jest pominięcie etykiet w tych instrukcjach, które nie są obiektami zewnętrznych odwołań lub skoków, chociaż na przykład w celu uczytelnienia programu możliwe jest dopisanie etykiet zbędnych z merytorycznego punktu widzenia.

Kod operacji określa rodzaj czynności, jakie maszyna ma wykonać. Najczęściej jest on w języku DAS skrótem angielskich słów, opisujących czynność nakazaną daną instrukcją.

Operand może mieć różne znaczenie w zależności od kodu operacji. Ogólnie można powiedzieć, że operand określa obiekt, na którym ma być wykonana czynność nakazana kodem operacji i zazwyczaj podaje bezpośrednio lub pośrednio adres tego obiektu w pamięci maszyny.
Komentarz nie jest rozpatrywany przez maszynę cyfrową i może zawierać dowolne zapisy, ułatwiające zrozumienie roli danej instrukcji w całości programu; w szczególności komentarz może być napisem pustym.
W polu operanda można umieszczać nazwę (etykietę), stałą lub wyrażenie względnie kilka tego rodzaju elementów rozdzielonych przecinkami.
Stałe mogą być dziesiętnymi lub oktalnymi (wyrażone w systemie ósemkowym), przy czym te drugie wyróżnia cyfra 0, umieszczona na początku (przykładowo 1689 jest stałą dziesiętną, zaś 012765 jest stałą oktalną).
W przypadku jeśli stała ma zawierać część ułamkową (stała zmiennoprzecinkowa lub rzeczywista), zapisuje się ją w ogólnej postaci

$$)\ \pm\ \text{cz. całkowita} \cdot \text{cz. ułamkowa}\ E\ \pm\ \text{wykładnik},$$

przykładowo:)-76.82E+12, co jest rozumiane jako $-76{,}82\ 10^{12}$. W języku DAS występują także stałe alfanumeryczne, będące tekstami ujętymi w apostrofy.
Wyrażenia tworzone są z nazw i stałych połączonych operatorami +, -, * (znak mnożenia) i / (znak dzielenia). Mnożenie i dzielenie wykonywane jest przed dodawaniem i odejmowaniem, w kolejności od lewej do prawej. W wyrażeniach można używać oznaczenia * dla oznaczenia adresu aktualnie wykonywanego rozkazu, tak więc, na przykład, zapis * + 2 oznacza komórkę pamięci odległą o 2 od aktualnie wykonywanego rozkazu.

Obok stałych numerycznych i alfanumerycznych w języku DAS używane są stałe adresowe. Są one ujmowane w nawiasy okrągłe i oznaczają adres określonej nazwy lub wyrażenia. Stałe adresowe są pośrednie i bezpośrednie (znaczenie tych terminów zostanie wyjaśnione dalej).
Czasami zdarza się, że chcemy wartość operanda użyć bezpośrednio w obliczeniach; taki typ operanda nazywa się literałem. Literał jest stałą dowolnego typu, poprzedzoną znakiem =, przykładowo =29 lub =(E+2).

2.1.2. Rozkazy (kody operacji) języka DAS

Rozkazy języka DAS podzielić można na 19 grup funkcjonalnych, które omówimy kolejno. Podamy uwagi na temat trybu adresacji, a bliższe wyjaśnienia używanych terminów zawarliśmy w rozdziale 2.1.3.

Pierwszą grupą rozkazów jest grupa ładowania/pamiętania. Rozkazy tej grupy pozwalają przesyłać informacje z komórek pamięci do rejestrów i odwrotnie przy użyciu sposobów adresacji: normalnej, rozszerzonej i bezpośredniej. Wykaz kodów operacji dla tej grupy rozkazów zebrano w tabeli 1, w której zastosowano notację wyjaśnioną niżej. W maszynie

Tabela 1. Rozkazy ładowania/pamiętania assemblera DAS

Funkcja	Postać kodu dla adresacji:			Nazwa rozkazu
	normalnej	rozszerzonej	bezpośredniej	
a' = n	LDA	LDAE	LDAI	ładuj rejestr A
b' = n	LDB	LDBE	LDBI	ładuj rejestr B
x' = n	LDX	LDXE	LDXI	Ładuj rejestr X
n' = a	STA	STAE	STAI	zapamiętaj A
n' = b	STB	STBE	STBI	zapamiętaj B
n' = x	STX	STXE	STXI	zapamiętaj X

Varian są trzy rejestry: A, B oraz X. W trakcie procesów obliczeniowych jest konieczne przesyłanie informacji z komórki pamięci o określonym adresie (oznaczmy ten adres przez N) do określonego rejestru względnie zapamiętanie w komórce o adresie N zawartośći określonego rejestru. Zabieg taki zmienia więc zawartość określonego rejestru lub określonej komórki pamięci, co będziemy oznaczali symbolicznie w ten sposób, że zawartość po wykonaniu rozkazu (nową, zmienioną) oznaczymy apostrofem. Równocześnie przyjmujemy konwencję, na mocy której odpowiednie małe litery oznaczają zawartości określonych elementów maszyny. Przykładowo a oznacza zawartość rejestru A, zaś n oznacza zawartość komórki pamięci o wyspecyfikowanym adresie N. Tak więc zapis

a' = n

należy czytać: do rejestru A prześlij zawartość komórki o podanym adresie N. Adres N ustalany jest zgodnie z zasadami przedstawionymi dalej.

Kolejną grupą rozkazów języka DAS jest zbiór rozkazów arytmetycznych. Wykaz kodów tych rozkazów podano w tabeli 2, w której oznaczenia i struktura są takie jak w tabeli 1. Przez AB (i odpowiednio ab) oznaczono połączenie rejestrów A i B oraz ich łączną zawartość.

Tabela 2. Rozkazy operacji arytmetycznych assemblera DAS

Funkcja	Postać kodu dla adresacji			Nazwa rozkazu
	normalnej	rozszerzonej	bezpośredniej	
n' = n+1	INR	INRE	INRI	zwiększ o jeden
a' = a+n	ADD	ADDE	ADDI	dodaj
a' = a-n	SUB	SUBE	SUBI	odejmij
ab'=a+b*n	MUL	MULE	MULI	pomnóż
b' =ab/n	DIV	DIVE	DIVI	podziel (reszta z dzielenia w A)

Rozkazy grupy operacji logicznych działają na pojedynczych bitach słowa maszynowego, przy czym wyróżnia się trzy rodzaje możliwych operacji: alternatywę (oznaczaną $\vee$), koniunkcję (oznaczaną $\wedge$) i różnicę symetryczną (oznaczaną $\oplus$). Do zilustrowania działania tych operacji podano niżej prosty, czterobitowy przykład. W minikomputerze Varian odpowiednie działania wykonywane są oczywiście na pełnych słowach maszynowych, liczących 16 bitów.

$$
\begin{aligned}
p &= 0\ 0\ 1\ 1\\
q &= 0\ 1\ 0\ 1\\
p \vee q &= 0\ 1\ 1\ 1\\
p \wedge q &= 0\ 0\ 0\ 1\\
p \oplus q &= 0\ 1\ 1\ 0.
\end{aligned}
$$

Opis kodów rozkazów operacji logicznych zebrano w tabeli 3.

Tabela 3. Rozkazy operacji logicznych assemblera DAS

Funkcja	Postać kodu dla adresacji			Nazwa rozkazu
	normal-nej	rozsze-rzonej	bezpo-średniej	
$a' = a \vee n$	ORA	ORAE	ORAI	alternatywa
$a' = a \oplus n$	ERA	ERAE	ERAI	różnica symetryczna
$a' = a \wedge n$	ANA	ANAE	ANAI	koniunkcja

Kolejną grupą rozkazów języka DAS są rozkazy przesunięć. Przez przesunięcie rozumiemy takie rprzemieszczenie kolejnych bitów zawartości określonego rejestru, które powoduje skutki pokazane przykładowo poniżej:

zawartość przed przesunięciem	1000111	
przesunięcie logiczne w lewo	0001110	(o jedno miejsce)
	1100000	(o pięć miejsc)
przesunięcie logiczne w prawo	0010001	(o dwa miejsca)
przesunięcie arytmetyczne w lewo	1001110	(o jedno miejsce)
przesunięcie arytmetyczne w prawo	1000001	(o dwa miejsca)
przesunięcie cykliczne w lewo	1111000	(o pięć miejsc)
przesunięcie cykliczne w prawo	1110001	(o dwa miejsca).

Należy zwrócić uwagę, że przesunięcie logiczne od arytmetycznego różni się zachowaniem skrajnie lewego bitu: przy przesunięciu arytmetycznym jest on "nieruchomy". Przy przesunięciu cyklicznym zawartość "wysuwająca się" z rejestru jedną stroną jest wprowadzana drugą stroną. Omawiane wyżej przesunięcia były przesunięciami krótkimi, tj. dotyczyły pojedynczego rejestru (np. tylko A lub tylko B). Istnieją także przesunięcia długie, w których uczestniczą rejestry A i B, traktowane jako jeden rejestr o podwójnej długości. Zestawienie rozkazów przesunięcia podano w tabeli 4.

Tabela 4. Rozkazy przesunięć assemblera DAS

Rejestr przesuwany	Typ przesunięcia	Kierunek	Postać kodu	Uwagi
A	logiczne	w prawo	LSRA	krótkie
B	logiczne	w prawo	LSRB	krótkie
A	cykliczne	w lewo	LRLA	krótkie
B	cykliczne	w lewo	LRLB	krótkie
AB	logiczne	w prawo	LLSR	długie
AB	cykliczne	w lewo	LLRL	długie
A	arytmetyczne	w prawo	ASRA	krótkie
B	arytmetyczne	w prawo	ASRB	krótkie
A	arytmetyczne	w lewo	ASLA	krótkie
B	arytmetyczne	w lewo	ASLB	krótkie
AB	arytmetyczne	w prawo	LASR	długie
AB	arytmetyczne	w lewo	LASL	długie

Działania wymagające odwoływania się do pamięci maszyny cyfrowej trwają zawsze stosunkowo długo, znacznie szybsze są operacje wykonywane z użyciem samych tylko rejestrów. Grupa rozkazów obsługujących rejestry zestawiona jest w tabeli 5. Przy opisywaniu tych rozkazów użyto

Tabela 5. Rozkazy modyfikacji zawartości rejestrów w assemblerze DAS

Funkcja	Postać kodu	Nazwa rozkazu
1	2	3
b' = a	TAB	prześlij zawartość rejestru A do B
x' = a	TAX	prześlij zawartość rejestru A do X
a' = b	TBA	prześlij zawartość rejestru B do A
x' = b	TBX	prześlij zawartość rejestru B do X
a' = x	TXA	prześlij zawartość rejestru X do A
b' = x	TXB	prześlij zawartość rejestru X do B
a' = 0	TZA	zeruj rejestr A
b' = 0	TZB	zeruj rejestr B
x' = 0	TZX	zeruj rejestr X

c.d. tab. 5

a' = s	TSA	prześlij do rejestru A wartość odpowiadającą układowi przełączników na pulpicie operatora
a' = a+1	IAR	powiększ zawartość rejestru A o 1
b' = b+1	IBR	powiększ zawartość rejestru B o 1
x' = x+1	IXR	powiększ zawartość rejestru X o 1
a' = a-1	DAR	zmniejsz zawartość rejestru A o 1
b' = b-1	DBR	zmniejsz zawartość rejestru B o 1
x' = x-1	DXR	zmniejsz zawartość rejestru X o 1
a' = $\bar{a}$	CPA	dopełnij zawartość rejestru A
b' = $\bar{b}$	CPB	dopełnij zawartość rejestru B
x' = $\bar{x}$	CPX	dopełnij zawartość rejestru X
a' = a+0F	AOFA	do zawartości rejestru A dodaj wskaźnik nadmiaru
b' = b+0F	AOFB	do zawartości rejestru B dodaj wskaźnik nadmiaru
x' = x+0F	AOFX	do zawartości rejestru X dodaj wskaźnik nadmiaru
a' = a - 0F	SOFA	od zawartości rejestru A odejmij wskaźnik nadmiaru
b' = b - 0F	SOFB	od zawartości rejestru B odejmij wskaźnik nadmiaru
x' = x - 0F	SOFX	od zawartości rejestru X odejmij wskaźnik nadmiaru
	MERC c d	prześlij wynik logicznej alternatywy zawartości wyspecyfikowanych w polu c rejestrów wejściowych do wyspecyfikowanych w polu d rejestrów wyjściowych
	INCR c d	powiększ wynik alternatywy rejestrów wyspecyfikowanych w polu c o 1 i prześlij go do rejestrów wyjściowych określonych w polu d
	DECR c d	jak w rozkazie INCR, tylko ze zmniejszeniem o 1
	COMPL c d	jak w rozkazie INCR, tylko z wykorzystaniem dopełnienia jedynkowego
	ZERO d	zeruj wyspecyfikowane w polu d rejestry wyjściowe

nowych oznaczeń: jeżeli a (zgodnie z dotychczasowymi ustaleniami) oznacza zawartość rejestru A, to przez $\bar{a}$ oznaczać będziemy dopełnienie jedynkowe zawartości tego rejestru. Dalej znak OF jest tzw. wskaźnikiem nadmiaru. Wskaźnik ten normalnie ma wartość 0 i ustawiany jest w stan 1 w przypadku, kiedy wynik operacji na zawartości któregoś z rejestrów spowodował powstanie wyniku większego od największej liczby możliwej do przedstawienia w maszynie (liczbą tą jest 077777). Oznaczenie s zostało użyte w tabeli 5 do wyrażenia wartości binarnej, ustawionej na przełącznikach znajdujących się na pulpicie operatora. Wyjaśnienia wymagają też oznaczenia c i d, stosowane w rozkazach przesłań zbiorowych (MERG, INCR, DECR, COMPL, ZERO). Oznaczenia te wskazują na trzybitowe pola, służące do specyfikacji rejestrów wejściowych i wyjściowych, przy czym stosuje się oznaczenia:

001 - rejestr A ,
010 - rejestr B ,
100 - rejestr X .

Chcąc przykładowo wyspecyfikować jako rejestry wejściowe rejestry A oraz X, a jako rejestry wyjściowe B oraz X, piszemy c = 101 i d = 110.

Ważną grupą są rozkazy skoków, zmieniające kolejność wykonywania rozkazów programu. Kolejność ta narzucona jest przez zawartość licznika rozkazów P, który w przypadku wykonywania rozkazów nieskokowych zwiększa się systematycznie po każdym wykonanym rozkazie, w sposób powodujący wykonanie w kolejności następnego rozkazu, natomiast rozkazy skoków zmieniają tę kolejność według tabeli 6. Należy zwrócić uwagę, że poza rozkazem JMP, będącym skokiem bezwarunkowym, wszystkie dalsze rozkazy przewidują wykonanie skoku w przypadku kiedy jest spełniony określony warunek. W każdym przypadku wykonywania skoku zachodzi operacja

$$p' = N,$$

w przypadku skoku nie wykonanego

$$p' = p + 1.$$

Obok skoków zwyczajnych, których zadaniem jest przeniesienie sterowania do innego punktu wykonywanego programu, wykorzystywać można

Tabela 6. Rozkazy skoków w assemblerze DAS

Warunek wykonania skoku	Postać kodu	Nazwa rozkazu
brak	JMP	skok bezwarunkowy
OF = 1	JOF	skok przy nadmiarze
OF = 0	JOFN	skok przy braku nadmiaru
a > 0	JAP	skok przy dodatniej zawartości rejestru A
a < 0	JAN	skok przy ujemnej zawartości rejestru A
a = 0	JAZ	skok przy zerowej zawartości rejestru A
b = 0	JBZ	skok przy zerowej zawartości rejestru B
x = 0	JXZ	skok przy zerowej zawartości rejestru X
a ≠ 0	JANZ	skok przy niezerowej zawartości A
b ≠ 0	JBNZ	skok przy niezerowej zawartości B
x ≠ 0	JXNZ	skok przy niezerowej zawartości X
s1= 1	JSS1	skok przy ustawionym przełączniku 1
s2= 1	JSS2	skok przy ustawionym przełączniku 2
s3= 1	JSS3	skok przy ustawionym przełączniku 3
s1= 0	JS1N	skok przy wyłączonym przełączniku 1
s2= 0	JS2N	skok przy wyłączonym przełączniku 2
s3= 0	JS3N	skok przy wyłączonym przełączniku 3
tabela 7	JIF	skok przy warunku złożonym

tzw. skoki ze śladem. Skoki te tym różnią się od omówionych wyżej, że powodują nie tylko przeniesienie sterowania do wskazanego adresu, ale dodatkowo jest zapamiętywany adres, z którego nastąpił skok jako "ślad" dla późniejszego powrotu. Skoku ze śladem używa się w celu programowania z wykorzystaniem podprogramów. Postaci kodów rozkazów skoku ze śladem są w zasadzie takie jak dla rozkazów skoku (patrz tabela 8), z tym że kod uzupełniany jest literą M dopisaną na końcu kodu. Tak więc bezwarunkowy skok ze śladem ma kod JMPM, skok ze śladem przy ustawionym wskaźniku nadmiaru ma kod JOFM. Ułatwia to znacznie naukę programowania.

Tabela 7. Wykaz kodów warunków przy realizacji skoku JIF

Warunek oktalny	Treść warunku
0001	OF = 1
0002	a > 0
0006	skok, jeśli podany warunek nie spełniony
0004	A < 0
0010	A = 0
0020	B = 0
0040	X = 0
0100	s1 = 1
0200	s2 = 1
0400	s3 = 1

Uwaga: ostateczny kod warunków określany jest jako suma kodów wybranych z powyższej tabeli. Skok wykona się, jeżeli wszystkie wyspecyfikowane warunki będą spełnione. Przykładowo kod warunku: "wszystkie warunki zerowe i wszystkie przełączniki ustawione" ma postać = 0010+0020+0040+0100+0200+0400=0770.

Kolejną grupą rozkazów są rozkazy trybu wykonania. Rozkazy te umożliwiają nakazanie wykonania pojedynczych rozkazów, umieszczonych poza głównym programem. Adres wykonywanego w trybie "wyskoku" rozkazu podawany jest jako N, czyli operand odpowiedniego rozkazu trybu wykonania. Rozkazy trybu wykonania, podobnie jak rozkazy skoków, wykonywane są bezwarunkowo lub warunkowo, przy czym postaci warunków są we wszystkich przypadkach jednakowe. Dlatego te rozkazy omawiamy tylko w tabeli 8, w której podano odpowiedniości między kodami rozkazów trybu wykonania a rozkazami skoków.

Rozkazy skoków i rozkazy trybu wykonania służą do sterowania przebiegiem wykonania programu w obrębie dostępnych instrukcji programu. Rozkazy, które teraz omówimy, nie zmieniają kolejności wykonywania innych rozkazów, lecz pełnią ogólne funkcje sterujące. Są to cztery rozkazy:

HLT - zatrzymanie programu;

NOP - instrukcja pusta;

Tabela 8. Wykaz odpowiedników pośród kodów rozkazów skoków, rozkazów skoków ze śladem i instrukcji trybu wykonania. Odpowiedniość rozumiana jest jako zgodność warunków, przy których odpowiednie rozkazy są wykonywane (por. tab. 6 i 7)

dla skoku	Postać kodu dla skoku ze śladem	dla rozkazu trybu wykonania
JMP	JMPM	XEC
JOF	JOFM	XOF
JOFN	JOFNM	XOFN
JAP	JAPM	XAP
JAN	JANM	XAN
JAZ	JAZM	XAZ
JBZ	JBZM	XBZ
JXZ	JXZM	XXZ
JANZ	JANZM	XANZ
JBNZ	JBNZM	XBNZ
JXNZ	JXNZM	XXNZ
JSS1	JS1M	XS1
JSS2	JS2M	XS2
JSS3	JS3M	XS3
JS1N	JS1NM	XS1N
JS2N	JS2NM	XS2N
JS3N	JS3NM	XS3N
JIF	JIFM	XIF

ROF - zeruj wskaźnik nadmiaru OF;

SOF - ustaw wskaźnik nadmiaru.

Ostatnią grupą rozkazów assemblera DAS jest grupa rozkazów wejścia/wyjścia. Funkcje realizowane przez te rozkazy są na tyle różnorodne, że omówimy je bez zestawiania w tabele.

EXC - rozkaz nakazuje wysłanie do kanału wejścia/wyjścia (we/wy) dziewięciobitowego kodu operacji, w celu realizacji na urządzeniu peryferyjnym wyspecyfikowanym w polu a funkcji określonej zawartością pola m. Znaczenia kodów podawanych w polach a i m podano w tabeli 9.

Tabela 9. Znaczenie kodów m i a w rozkazach EXC m a i SEN m a

a	Urządzenie	m	Znaczenie dla rozkazu EXC	SEN
01	dalekopis	1	przyłączenie rejestru wejściowego do bufora pośredniego	gotów do pisania
		2	przyłączenie rejestru wyjściowego do bufora pośredniego	gotów do czytania
		4	uruchomienie jednostki sterującej dalekopisu	- - - -
37	czytnik perforator tasiemki papierowej	0	przyłączenie perforatora do bufora pośredniego	- - - -
		4	zatrzymanie czytnika i uruchomienie jednostki sterującej	- - - -
		5	uruchomienie czytnika	gotów do transmisji
		6	przygotowanie bufora perforatora	- - - -
		7	przeczytanie jednego znaku do bufora	- - - -
77	analizator tercjowy i wąskopasmowy	2	- - - -	wartość poniżej przyjętego zakresu
		3	- - - -	wartość powyżej przyjętego zakresu
		4	- - - -	przesterowanie analizatora
		5	wyłączenie pamięci automatycznej	wartość spoza przyjętego zakresu
		6	włączenie pamięci automatycznej	widmo gotowe do odczytu
		7	żądanie przygotowania widma do odczytu	koniec odczytu widma
63	zegar czasu rzeczywistego		start zegara do odmierzania odcinka czasu	
		0	500 ms	- - - -
		1	10 ms	- - - -
		2	20 ms	- - - -
		3	50 ms	- - - -
		4	100 ms	- - - -
		5	200 ms	- - - -
		6	500 ms	- - - -
		7	1000 ms	upłynięcie żądanego odcinka czasu

Uwaga: Część m oznacza urządzenie wybrane do transmisji (wykorzystywane także w rozkazach zebranych w tabeli 10), zaś część a oznacza funkcję realizowaną przez rozkaz. Przy rozkazie SEN podano warunek skoku.

EXC2 - dodatkowy do wyżej podanego, umożliwia wprowadzenie ośmiu innych funkcji dla każdego z urządzeń we/wy.

SEN - formuje i wysyła do kanału we/wy dziewięciobitowy kod, w celu rozpoznania statusu urządzenia. Jest to rozkaz skokowy: jeśli status urządzenia odpowiada aktualnym potrzebom, dokonywany jest skok do instrukcji o etykiecie podanej w polu operanda. Bliższych informacji na temat statusów urządzeń szukać należy w dokumentacji tych urządzeń.

Dalsze rozkazy we/wy są już typowe i służą do wprowadzenia określonych informacji do maszyny lub ich wyprowadzenia na urządzenie zewnętrzne. Oznaczając przez we informację na określonym wejściu, a przez wy informację na określonym wyjściu, możemy operacje dokonywane przez pozostałe rozkazy zestawić w tabeli 10.

Tabela 10. Niektóre rozkazy wejścia wyjścia assemblera DAS

Postać kodu	Treść rozkazu	Nazwa
CIA	a' = we	wczytaj do A
CIB	b' = we	wczytaj do B
CIAB	ab' = we	wczytaj do połączonych rejestrów A i B
INA	a' = a $\vee$ we	wprowadź do A
INB	b' = b $\vee$ we	wprowadź do B
INAB	ab' = ab $\vee$ we	wprowadź do AB
OAR	wy' = a	wyprowadź z A
OBR	wy' = b	wyprowadź z B
OAB	wy' = a $\vee$ b	wyprowadź alternatywę zawartości rejestrów AB
IME	n' = we	wczytaj do pamięci
OME	wy' = n	wyprowadź z pamięci

2.1.3. Adresacja (operandy) w języku DAS

Omówiliśmy kody operacji dla wszystkich rozkazów możliwych do wykonania w języku DAS. Pojawiały się jednak pewne terminy, dotyczące

adresacji, które należy wyjaśnić. Adrsacja, określana wyżej jako normalna, może być: podstawowa, pośrednia, względna i indeksowa. Adrsacja podstawowa polega na tym, że w polu operanda rozkazu podana jest bezpośrednio jawna lub symboliczna postać adresu komórki pamięci, do której powinno nastąpić odwołanie. Oznaczając przez OP zawartość pola operanda, możemy zapisać dla tego typu adresacji

$$N = OP,$$

gdzie N jest, wykorzystywanym przy omawianiu poszczególnych rozkazów, efektywnym adresem operanda.

Adresacja pośrednia polega na tym, że pole operanda zawiera adres komórki pamięci, w której (w najprostszym przypadku) jest adres operanda. Oznaczając przez op zawartość komórki pamięci o adresie OP, możemy więc zapisać

$$N = op.$$

W bardziej złożonych przypadkach komórka wskazana w OP może zawierać adres nie samego operanda, lecz adres adresu itd.

Adresacja względna polega na tym, że zawartość pola operanda rozkazu OP określa adres operanda po dodaniu do zawartości licznika rozkazów p, powiększonego o 1, zatem

$$N = OP + p + 1.$$

Adresacja indeksowa polega na obliczaniu adresu operanda jako sumy zawartości pola operanda i zawartości rejstru indeksowego, którym w komputerach VARIAN może być rejestr X lub B. Zatem

$$N = OP + x, \quad \text{gdy } m = 101,$$

względnie

$$N = OP + b, \quad \text{gdy } m = 110.$$

Adresacja rozszerzona, stosowana w rozkazach zajmujących dwie komórki pamięci, pozwala na adresację pamięci o pojemności do 32 768 komórek, a więc o maksymalnej pojemności dostępnej w komputerach Varian. W przypadku innych metod adresacji objęcie adresacją tak dużego obszaru możliwe jest jedynie przy użyciu indeksowania.

Adresacja bezpośrednia nie jest w rzeczywistości adresacją, lecz polega na umieszczeniu w polu operanda bezpośrednio argumentu, który zostanie użyty do wykonania czynności nakazanej kodem rozkazu. Stosowanie tego typu "adresacji" upraszcza niektóre programy i przyspiesza ich wykonanie.

2.1.4. Postacie instrukcji języka DAS

Ze względu na format instrukcje języka DAS podzielić można na 5 grup, wymienionych w tabeli 11. Ogólna charakterystyka tych grup jest podana w tabeli 12.

Rozkazy grupy 1 mają ogólną postać

kod wyrażenie

w przypadku adresacji podstawowej,

kod wyrażenie

w przypadku adresacji pośredniej względnej,

kod wyrażenie, wyrażenie

w przypadku adresacji indeksowej. W drugim przypadku drugie (po przecinku) wyrażenie określa numer rejestru indeksowego na zasadzie: 1 - rejestr X, 2 - rejestr B. Rozkazy te zajmują 1 komórkę pamięci.

Rozkazy grupy 2 zajmują dwie komórki pamięci, przy czym adres (np. skoku) podany jest w drugim słowie.

Rozkazy grupy 3 są także dwusłowowe. W polu operanda tych rozkazów umieszcza się dwa wyrażenia, z których pierwsze jest tzw. mikrokodem rozkazu (patrz niżej), a drugie adresem operanda. Mikrokod jest stosowany w instrukcjach JIF, JIFM i XIF, określając warunek skoku (patrz tabela 7), a także w rozkazach IME i OME jako adres urządzenia.

Rozkazy grupy 4 są jednosłowowe i nie zawierają odwołań do pamięci. W polu operanda może być ewentualnie podany mikrokod, będący sumą liczb określanych w opisie jako c i d (rozkazy MERG i dalsze) lub kodów mnemonicznych.

Rozkazy grupy 5 są dwusłowowe i mogą być adresowane we wszystkich wymienionych wyżej postaciach. Rozkazy tej grupy mają od 1 do 3 argumentów, zależnie od typu rozkazu. Należy pamiętać, że adresacja bezpośrednia jest sygnalizowana gwiazdką w polu operanda bezpośrednio po kodzie instrukcji.

Tabela 11. Lista rozkazów assemblera DAS z podziałem na grupy

Grupa	Wykaz kodów rozkazów należących do danej grupy
1	ADD ANA DIV ERA INR LDA LDB LDX MUL ORA STA STB STX SUB
2	ADDI ANAI DIVI ERAI INRI JAN JANM JANZ JANZM JAP JAPM JAZ JAZM JBZ JBZM JMP JMPM JOF JOFM JOFN JOFNM JSS1 JSS2 JSS3 JS1M JS2M JS3M JS1NM JS2NM JS3NM JXZ JXZM LDAI LDBI LDXI MULI ORAI STAI STBI STXI SUBI XAN XANZ XAP XAZ XBNZ XBZ XEC XOF XOFN XS1 XS1N XS2 XS2N XS3 XS3N XXZ XXNZ
3	JIF JIFM JMIF OME SEN IME XIF
4	AOFA AOFB AOFX ASLA ASLB ASRA ASRB CIA CIAB CIB COMP CPA CPB CPX DAR DBR DECR DXR EXC EXC2 HLT IAR IBR INA INAB INB INCR LASL IXR LASR LLRL LLSR LRLA LRLB LSRA LSRB MERG NOP OAB OAR OBR ROF SEL SEL2 SOF SOFA SOFB SOFX TAB TAX TBA TBX TXA TXB TZA TZB ZERO TZX
5	ADDE ANAE DIVE ERAE IJMP INRE JSR LDAE LDBE LDXE MULE ORAE SRE STAE STBE STXE SUBE

Tabela 12. Niektóre własności grup rozkazów assemblera DAS

Własność	Grupa 1	2	3	4	5	Uwagi
Długość rozkazu	1	2	2	1	2	słów maszyn.
Odwołania do pamięci	tak	tak	tak	nie	tak	o ile nie używana adresacja bezpośrednia
czy dozwolona adresacja pośrednia	tak	tak	tak	nie	tak	
czy dozwolone indeksowanie	tak	nie	nie	nie	tak	
liczba wyrażeń w polu operanda	1-2	1	2	1	1-3	
mikrokodowanie	nie	nie	tak	tak	nie	

Programy napisane w języku DAS mogą być wprowadzane do maszyny cyfrowej przez czytnik kart perforowanych lub przez czytnik tasiemki perforowanej. Ze względu na częstsze występowanie konfiguracji maszyny Varian nie zawierającej czytnika kart omówimy tu metodę pisania instrukcji na taśmie papierowej. Maksymalna długość instrukcji wynosi 80 znaków, przy czym znakiem końca jednej i początku następnej instrukcji jest zmiana linii (nowy wiersz), oznaczony na tasiemce znakami CR LF. Poszczególne pola: etykiety, kodu operacji i operanda oddzielone są od siebie przecinkami, pole komentarza rozpoczyna się natomiast blankiem (odstępem). Jeśli w instrukcji występuje kilka operandów, to są one także separowane przecinkami. W przypadku kart dziurkowanych nie stosuje się przecinków, a pola etakiety, kodu operacji i operanda mają ustaloną pozycję na karcie.

2.1.5. Dyrektywy języka DAS

Obok wyżej omówionych instrukcji języka DAS stosowane są w programach tzw. dyrektywy, które stanowią instrukcje dla samego assemblera. Można wyróżnić 13 typów dyrektyw:

- definicji symboli;

- definicji instrukcji;
- sterowania pozycją;
- definicji danych;
- rezerwacji pamięci;
- warunkowej translacji;
- sterowania pracą assemblera;
- sterowania podprogramami;
- sterowania wydrukiem i perforacją;
- łączności z programami w FORTRANIE;
- łączenia fragmentów programu;
- sterowania operacjami we/wy;
- definicji makrosów.

Omawianie wszystkich dyrektyw języka DAS zajmie zbyt wiele miejsca, przeto wymienimy te z nich, które są częściej używane, i to w takiej postaci, w jakiej są najbardziej przydatne. Obszerniejsze informacje znaleźć można w podręczniku [1]. Prawie każdy program rozpoczyna się dyrektywą

ORG adres.

Dyrektywa ta deklaruje początkowy adres programu, wskazując assemblerowi, jak ma rozmieścić wygenerowane kody. Ma ona także obszerniejszą postać, której jednak rzadko się używa. Jeżeli zamierzamy w programie odwoływać się do jakiejś zmiennej przez jej nazwę (etykietę), to musimy zarezerwować dla tej zmiennej pamięć. Dokonuje się tego między innymi dyrektywą

etykieta BSS ilość,

gdzie etykieta jest rozważaną nazwą zmiennej, zaś ilość podaje liczbę komórek pamięci, jakie zostaną przydzielone do tej nazwy. Przypadek, kiedy ilość jest większa od jeden, odpowiada mechanizmowi zmiennych indeksowanych w językach wyższych rzędów. Sposób wykorzystania takich obszarów pamięci, oparty na adresacji indeksowej, zilustrowany będzie niżej na przykładzie. Rezerwując pamięć dla zmiennych, możemy równocześnie nadawać im wartości początkowe. Służy do tego dyrektywa DATA

etykieta DATA wartość, wartość, ...

Zostanie zarezerwowanych tyle komórek pamięci, ile podano wartości, a poszczególnym komórkom nadane będą, jako wartości początkowe, wartości wymienione w dyrektywie DATA. Ważnym mechanizmem, ułatwiającym programowanie w języku DAS, są podprogramy. Początek podprogramu sygnalizowany jest dyrektywą

etykieta ENTR.

Po tej dyrektywie etykieta wymieniona w ENTR może być traktowana jako nazwa podprogramu. Koniec podprogramu sygnalizowany jest dyrektywą

RETU nazwa podprogramu.

Dyrektywa ta powoduje zakończenie podprogramu i powrót do miejsca wywołania. Wywołanie podprogramu następuje w wyniku wykonania dyrektywy

CALL nazwa podprogramu, argument, argument, ...,

gdzie argumenty są parametrami aktualnymi, przenoszonymi do podprogramu w obręb początkowych komórek w okolicy wejścia do podprogramu. Dyrektywą kończącą każdy program jest dyrektywa

END.

Pozostałe dyrektywy nie są tak często używane i dlatego nie będziemy ich tu przytaczali.

2.1.6. Przykładowy program w języku DAS

Praktyczna umiejętność programowania w assemblerze DAS jest możliwa do osiągnięcia po odpowiednio długiej praktyce. Do zilustrowania podanych wyżej informacji na temat organizacji i elementów języka DAS przytaczamy niżej (za podręcznikiem firmy Brüel and Kjaer "7504, 6401, 7102 - instructions and aplications") przykład programu napisanego w assemblerze DAS. Program ten oblicza pierwiastki kwadratowe 40_8 liczb ósemkowych i zapamiętuje wyniki. Zgodnie z omówionym wyżej formatem instrukcji języka DAS wyróżnimy w opisie cztery pola:

etykieta	kod operacji	operand	komentarz
	,ORG	,0500	Dyrektywa ustalająca pierwszy adres programu – jest nim 0500.
	,LDXI	,037	Ładowanie bezpośrednie liczby ósemkowej 037 do rejestru X.
DALEJ	,LDB	,POCZ,1	Ładowanie do rejestru b zawartości komórki pamięci o adresie obliczonym jako suma adresu odpowiadającego etykiecie początkowej POCZ i zawartości rejestru indeksowego X. Należy zwrócić uwagę, że jeśli zawartość X będzie ulegała zmianie, to równocześnie zmieniany będzie adres ładowanej komórki.
	,CALL	,PIERW,0777	Dyrektywa wywołująca podprogram o nazwie PIERW (pierwiastek), tłumaczona na trzy rozkazy, z których pierwszy jest skokiem ze śladem.
	,SBT	,WYNIK,1	Do tej instrukcji następuje powrót z podprogramu. Powoduje ona zapamiętanie zawartości rejestru B (w rejestrze tym znajduje się obliczona w podprogramie PIERW wartość pierwiastka). Adres komórki, w której zapamiętany jest wynik, obli-

			czany jest jako suma adresu odpowiadającego etykiecie WYNIK i zawartości rejestru indeksowego X.
	,JXZ	,STOP	Skok warunkowy do etykiety STOP w przypadku, kiedy rejestr X zawiera zero. Oznacza to, że wszystkie liczby z obszaru zaczynającego się etykietą POCZ zostały spierwiastkowane, a ich pierwiastki umieszczono w odpowiednich komórkach obszaru, zaczynającego się od etykiety WYNIK.
	,DXR		Zmniejszenie zawartości rejestru X o jeden.
	,JMP	,DALEJ	Powrót do instrukcji o etykiecie DALEJ, czyli pobranie i obliczenie pierwiastka kolejnej następnej liczby z obszaru POCZ.
STOP	,HLT		Do tej instrukcji wykonywany jest skok w przypadku, kiedy X=0. Następuje wtedy przerwanie akcji programu.
POCZ	,DATA	,25,30, ...	Dyrektywa języka DAS, rezerwująca pamięć i nadająca początkowe wartości liczbom, z których będą wyciągane pierwiastki. Powinno tu zostać wymienione (w polu operanda) 40_8 liczb rozdzielonych przecinkami.

WYNIK	,BSS	,040	Dyrektywa rezerwująca 40_8 komórek pamięci na wyniki obliczeń.
PIERW	,ENTR		Dyrektywa definiująca punkt wejścia (początek) podprogramu o nazwie PIERW.
	,JBZ	,KONIEC+1	W rejestrze B jest liczba, z której ma być wyciągany pierwiastek. Jeśli liczbą tą jest zero, następuje skok do instrukcji następującej bezpośrednio po instrukcji o etykiecie KONIEC
	,TBA		Przesłanie zawartości rejestru B do rejestru A. W tej chwili oba rejestry mają tę samą zawartość, równą liczbie pierwiastkowanej.
	,JAN*	,PIERW	Jeśli zawartość rejestru A jest ujemna, oznacza to błędne wywołanie podprogramu. Następuje wówczas skok do instrukcji, której adres jest zawarty pod etykietą PIERW. Ze względu na sposób wywołania programu (por. dyrektywę CALL) instrukcja ta ma postać 0777 i jest instrukcją zatrzymania programu. Gwiazdka po instrukcji JAN sygnalizuje adresację bezpośrednią.
	,STB	,LICZ	Zapamiętanie zawartości rejestru B w komórce pamięci o etykiecie LICZ.

	,STB	,APR	Zapamiętanie zawartości B w komórce o etykiecie APR.
	,STX	,PAM	Zapamiętanie zawartości rejestru X w komórce o etykiecie PAM.
	,LDXI	,7	Załadowanie do rejestru X wartości 7 (podprogram liczy pierwiastek na drodze określania siedmiu kolejnych przybliżeń).
ZNOWU	,TZA		Zerowanie rejestru A.
	,LDB	LICZ	Załadowanie do rejestru B zawartości komórki LICZ.
	,DIV	,APR	Dzielenie rejestrów A i B przez zawartość komórki APR. Wynik dzielenia umieszczony jest w rejestrze B.
	,TBA		Przesłanie wyniku z rejestru B do A.
	,ADD	,APR	Dodanie do zawartości rejestru A zawartości komórki pamięci APR.
	,TAB		Przesłanie zawartości rejestru A do rejestru B.
	,ASRB	,1	Przesunięcie arytmetyczne zawartości rejestru B o jedną pozycję w lewo. Działanie to jest równoważne z dzieleniem przez dwa, ale jest szybciej wykonywane.
	,STB	,APR	Zawartość rejestru B zapamiętana jest w komórce o etykiecie APR.

	,DXR		Zmniejszenie zawartości rejestru X o jeden.
	,JXZ	,KONIEC	Jeśli zawartość X jest zerowa (obliczono już siedem kolejnych przybliżeń), następuje skok do etykiety KONIEC.
	,JMP	,ZNOWU	Powrót do początku obliczeń kolejnego przybliżenia.
KONIEC	,LDX	,PAM	Odtworzenie zawartości rejestru X.
	,INR	,PIERW	Przygotowanie do powrotu z podprogramu do miejsca, w którym go wywołano.
	,RETU	,PIERW	Powrót z podprogramu do miejsca wywołania.
LICZ	,BSS	,1	Zarezerwowanie komórki pamięci.
APR	,BSS	,1	Zarezerwowanie komórki pamięci.
PAM	,BSS	,1	Zarezerwowanie komórki pamięci.
	,END		Dyrektywa oznaczająca koniec programu.

Struktura programu jest dość przejrzysta, jakkolwiek jego prześledzenie może zająć trochę czasu, szczególnie jeśli konfrontujemy używane instrukcje z ich wcześniejszym opisem. (Jest to zresztą ze wszech miar wskazane, gdyż pozwala wyjątkowo łatwo przyswoić sobie reguły pisania programów w assemblerze). Pewnych wyjaśnień wymaga być może algorytm obliczania pierwiastka, zastosowany w podprogramie. Jest to metoda iteracyjna, bazująca na wzorze Newtona. Wzór ten pozwala obliczyć kolejne przybliżenie y_{n+1} pierwiastka liczby pierwiastkowanej, a opierając się na poprzednim (gorszym) przybliżeniu zgodnie z zależnością

$$y_{n+1} = (a/y_n + y_n)/2.$$

Jako pierwsze przybliżenie przyjmowane jest $y_o = a$. W podprogramie podany wyżej wzór stosowany jest siedmiokrotnie (z założenia), przy czym y_{n+1} i y_n przechowywane są w komórce APR (aproksymacja), a liczba pierwiastkowana a przechowywana jest w komórce LICZ. Algorytm ten działa nader skutecznie. Przyglądnijmy się jego działaniu dla LICZ=30. Kolejne wartości APR będą wówczas wynosiły (zaleca się czytelnikowi sprawdzenie podanych obliczeń na podstawie przytoczonego wzoru!):

APR = 30.0 15.5 8.7177415 6.0794995 5.507058 5.477306
5.4772255 5.4772255

Jak widać przytoczony algorytm jest zbieżny, gdyż po upływie 7 iteracji wyniki zaczęły się powtarzać, co oznacza, że otrzymano wynik poprawny. Istotnie $(5,4772255)^2 = 29,999999$, czyli z dokładnością do ostatniej uwzględnianej w obliczeniach pozycji osiągnięto pierwiastek.

Przedstawiony program miał za zadanie wyliczanie określonych wyników na podstawie danych wstępnie (dyrektywą DATA) umieszczonych w pamięci. Dzięki tym założeniom program był prosty i stosunkowo przejrzysty. Jak jednak przekonamy się z następnego rozdziału, program taki można było znacznie łatwiej napisać posługując się językiem FORTRAN. Niewątpliwie program napisany w assemblerze DAS będzie znacznie bardziej wydajny niż ten sam program napisany w języku FORTRAN, gdyż zajmować będzie znacznie mniej miejsca w pamięci maszyny i będzie wyraźnie szybciej wykonywał obliczenia. Wątpliwe jest jednak, czy te zalety równoważą trud pisania programu w tak trudnym języku jak DAS i czy ktokolwiek w praktyce dałby się namówić na programowanie tą metodą. Zalety assemblera DAS widać jednak wyraźnie przy programowaniu zadań, w których minikomputer ma współpracować z zewnętrzną aparaturą pomiarową, na przykład z analizatorami widma. W tym momencie posługując się FORTRANEM, czy jakimkolwiek innym językiem wysokiego poziomu, nie będziemy mogli napisać programu zdolnego do przyjmowania, przekodowywania i przetwarzania sygnałów z analizatora, względnie program ten będzie tak niewygodny i niezgrabny, że praktycznie stanowić

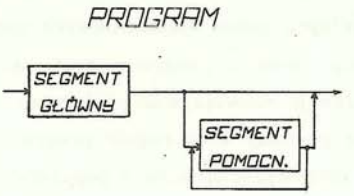
będzie ograniczenie i opóźnienie w procesie pomiarowym. Podsumowując należy więc stwierdzić, że większość typowych zadań obliczeniowych nie wymaga stosowania tak trudnego i złożonego systemu programowania, jakim jest assembler DAS, i doskonale można je rozwiązywać posługując się FORTRANEM. W następnym rozdziale poznamy więc reguły programowania w tym najbardziej współcześnie rozpowszechnionym i popularnym języku.

2.2. Język FORTRAN

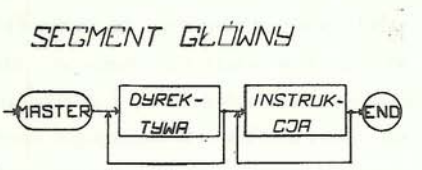
Jak wykazano poprzednio, programowanie w językach niższych rzędów (np. w assemblerze DAS) ma wiele wad. Znacznie wygodniejsze jest programowanie z wykorzystaniem tzw. języków algorytmicznych, których wspólną cechą jest to, że bazując na słowach języka naturalnego (zwykle angielskiego) i zapisach maksymalnie upodobnionych do zwykłych wzorów matematycznych, zapewniają wystarczającą jednoznaczność i precyzję zapisu, tak aby mogły służyć do formułowania algorytmów dla elektronicznej maszyny cyfrowej.

Z pwodów, które zostały omówione we wstępie rozdziału 2, wybrano do przedstawienia w niniejszym opracowaniu język FORTRAN, a konkretnie wersję FORTRAN 1900, w którą wyposażone są maszyny cyfrowe serii Odra 1300 oraz ICL 1900, a więc niewątpliwie najpopularniejszy obecnie sprzęt obliczeniowy. Podkreślić przy tym należy, że różnice pomiędzy poszczególnymi wersjami FORTRANU są bardzo nieznaczne i po opanowaniu techniki programowania w języku FORTRAN 1900 można bez trudu w ciągu krótkiego czasu przyswoić sobie programowanie w dowolnej innej konkretnej realizacji języka FORTRAN.

Do przedstawienia reguł budowy programu w języku FORTRAN wykorzystano technikę rysunkową, zastosowaną po raz pierwszy przez N. Wirtha przy omawianiu gramatyki języka PASCAL. Technika ta nie była dotychczas zbyt powszechnie stosowana do prezentacji języka FORTRAN, niemniej pozwala na bardzo zwięzły i czytelny zapis, co predystynuje ją do zastosowań diagnostycznych. Opis każdego elementu programu ma kształt schematu blokowego z jednym wejściem i jednym wyjściem. Na



Rys. 1



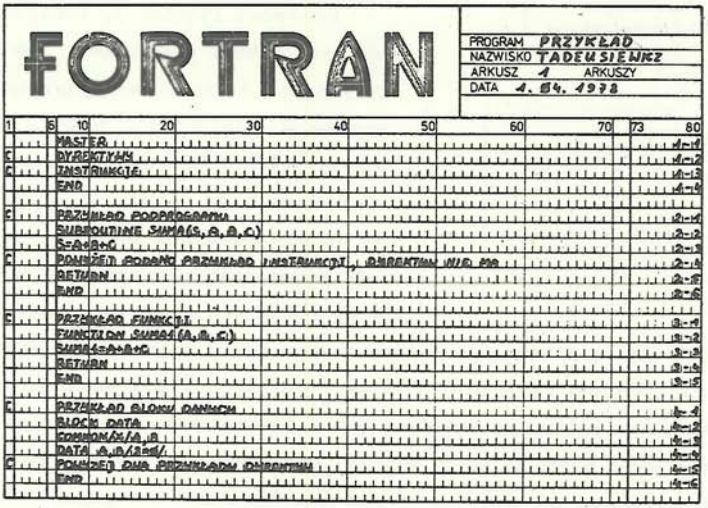
Rys. 2

schemacie jest widocznych szereg bloków, połączonych strzałkami. Poruszając się po schemacie w sposób, na jaki pozwalają strzałki, otrzymujemy poprawny zapis w języku FORTRAN. Stosowane są przy tym dwa rodzaje bloków. Bloki okrągłe zawierają treści ostateczne, dalej już nie definiowane, bloki prostokątne zawierają natomiast opisy, dla których wcześniej lub później, w stosunku do miejsca, w którym się pojawiły, wystąpią szczegółowe wyjaśnienia. Rozważmy to na kolejnych rysunkach.

Rysunek 1 przedstawia strukturę programu w języku FORTRAN. Widać, że możemy się na nim poruszać wieloma sposobami, tworząc rozmaite konfiguracje, które są poprawnymi programami. Na program może składać się sam segment główny, poza tym segmentem może jednak wystąpić, po nim, jeden lub więcej segmentów pomocniczych. Zarówno pojęcie segmentu głównego, jak i segmentu pomocniczego będą dalej definiowane, gdyż są podane w blokach prostokątnych.

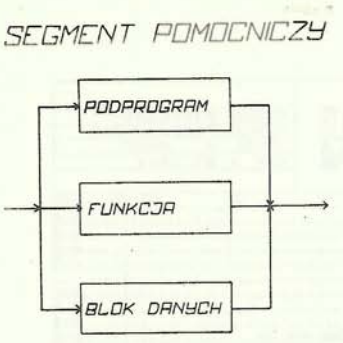
Rysunek 2 dostarcza informacji na temat segmentu głównego. Widać z niego, że na segment ten składają się: słowo MASTER, które go rozpoczyna, pewna liczba dyrektyw i pewna liczba instrukcji, a całość kończy się słowem END. Zarówno MASTER, jak i END, podane zostały w ostatecznej postaci, tak jak muszą występować w programie. Widać dodatkowo, że instrukcji i dyrektyw (oba te pojęcia będą dalej wyjaśnione) może być dowolnie dużo, ale dyrektywy zawsze poprzedzają instrukcje, a słowa MASTER i END są pojedyncze i zawsze muszą być pierwszym i ostatnim słowem segmentu.

Należy tutaj wprowadzić dodatkową informację, dotyczącą sposobu wprowadzania program- w języku FORTRAN do maszyny cyfrowej. Typowym nośnikiem przy wprowadzaniu programów w języku FORTRAN są karty papierowe, na których dziurkuje się wszystkie elementy programu.



Rys. 3

Na jednej karcie mieści się dokładnie 80 znaków, przy czym położenie znaku na karcie nie może być dowolne, gdyż w obrębie karty wyróżnia się tzw. pola. Pierwsze z nich, liczące 5 znaków może zawierać jedynie etykietę. Jeśli dana instrukcja czy dyrektywa nie ma etykiety, wówczas pole to musi być puste, tj. zapełnione spacjami. Drugie ma szerokość tylko jednego znaku (jest to szósty w kolejności znak) i w większości kart powinno być puste (zawierać spację). Jeśli w polu tym pojawi się jakikolwiek znak, oznaczać to będzie, że cała karta nie zawiera nowej instrukcji, lecz stanowi kontynuację karty poprzedniej. Jest to bardzo wygodne w przypadku pisania długich instrukcji, mieszczących się na kilku czy nawet kilkunastu kartach. Najważniejsze pole rozciąga się od siódmego do siedemdziesiątego drugiego znaku każdej karty. Tu wpisywana jest treść instrukcji lub dyrektywy. Pole od siedemdziesiątego trzeciego znaku do końca karty ma charakter identyfikacyjny, gdyż wpisuje się tam numery kolejne kart, potrzebne przy identyfikacji błędu. Szczególne znaczenie ma także pierwszy znak każdej karty. Jeśli jest nim li-

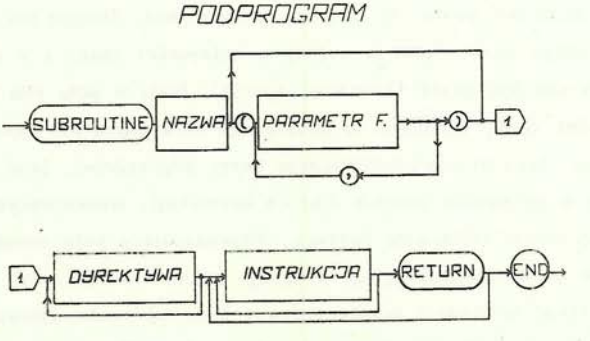


Rys. 4.

tera C, to cała zawartość karty traktowana jest jako komentarz.

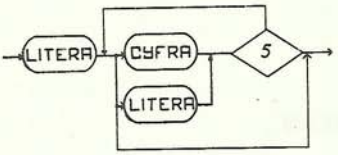
Rozważmy rysunek 3. Pokazano na nim tzw. arkusz programowy, zawierający rubryki, które ułatwiają pisanie programów w FORTRANIE. Widać, że każdy wiersz zawiera 80 miejsc do wpisywania znaków oraz że miejsca te są grubszymi liniami podzielone na pola, o których była wyżej mowa. Przykładowe wiersze o numerach od 1-1 do 1-4 pokazują sposób zapisu poznanych już elementów segmentu głównego. Wiersze 1-2 i 1-3 zawierają komentarz. Podczas wprowadzania informacji do maszyny cyfrowej na podstawie zapisów z arkusza programowego perforuje się karty: jedną kartę na każdy wiersz. Powstały w ten sposób plik kart jest programem dla maszyny cyfrowej, który (jeśli jest poprawny) może być wykonany.

Na rysunku 4 pokazano, czym może być segment pomocniczy[1], a na rysunku 5 podano strukturę podprogramu. Niektóre elementy występujące na rysunku 5 wyjaśniono na rysunkach 6 i 7. Ważnym elementem jest NAZWA, której struktura została przedstawiona na rysunku 6, gdyż będzie się ona pojawiać w niemal wszystkich dalszych definicjach. Jak wi-

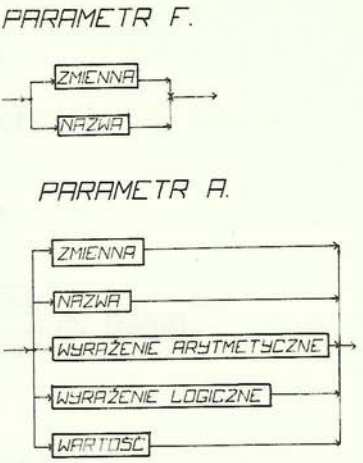


Rys. 5

[1] Nazwy stosowane w tym rozdziale są skrótami nazw powszechnie używanych.

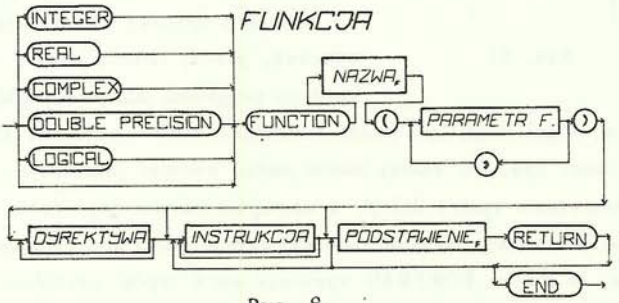


Rys. 6



Rys. 7

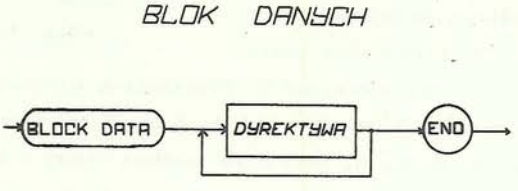
dać, nazwą może być dowolny zespół od 1 do 6 znaków, zaczynający się koniecznie od litery. Romb z cyfrą wewnątrz określa liczbę przejść przez rozgałęzienie: w rozważanym przypadku nie może ich być więcej niż 5. Przykładową strukturę podprogramu podano na rysunku 3 (wiersze 2-1 do 2-6). Nazwami w tym przypadku są: SUMA, A, B, C, S, przy czym ostatnie cztery z nich są parametrami f. (por. rys. 5 i 7). Na temat parametrów f. i parametrów a. podamy więcej informacji przy omawianiu pojęcia wyrażenia arytmetycznego. Strukturę segmentu pomocniczego, jakim jest funkcja, przedstawia rysunek 8, uzupełniony wyjaśnieniami na rysunku 9. Przykład funkcji po-



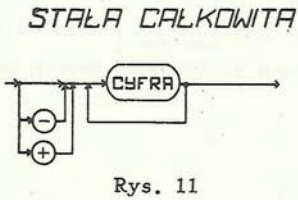
Rys. 8

dano na rysunku 3 w wierszach o numerach 3-1 do 3-5. Ostatnim możliwym typem segmentu jest blok danych (rys. 10 i wiersze 4-1 do 4-6 na rysunku 3). Poszczególne wymienione wyżej segmenty programu pełnią

Rys. 9



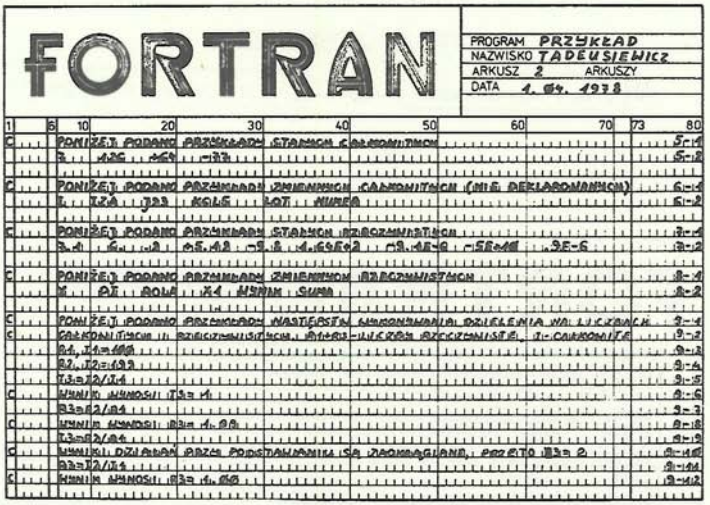
Rys. 10



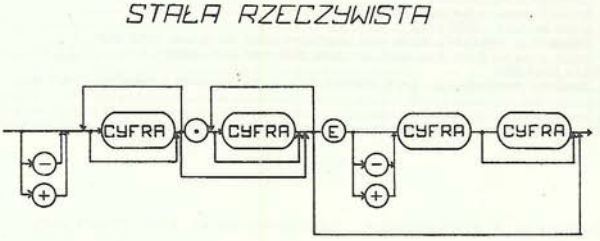
Rys. 11

różne funkcje podczas pracy maszyny cyfrowej, co będzie dalej szczegółowo omówione i zilustrowane na przykładach.

Maszyny cyfrowe wykorzystywane są do obliczeń, przeto istotną rolę w budowie każdego programu odgrywać będą wartości liczbowe, które mogą być dwóch rodzajów: stałe i zmienne. Zmienna jest nazwą (por. rys. 6), której można nadać wartość instrukcją czytania lub podstawienia (patrz dalej), a następnie używać jej w obliczeniach, podobnie jak stałej. Pojęcie zmiennej będzie dalej nieco dokładniej zdefiniowane. W języku FORTRAN wyróżnia się 6 typów wartości. Wartości całkowite są to stałe zbudowane według zasad podanych na rysunku 11 (por. także wiersze 5-1 i 5-2 na rysunku 12) względnie zmienne zaczynające się od jednej z liter: I, J, K, L, M, N, a także zmienne zadeklarowane jako należące do typu INTEGER (patrz dalej, por. także rys. 12, wiersze 6-1 oraz 6-2). W odróżnieniu od wartości całkowitych wyróżnia-

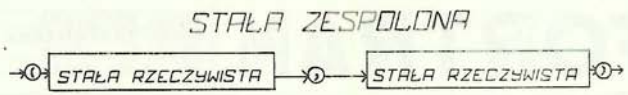


Rys. 12

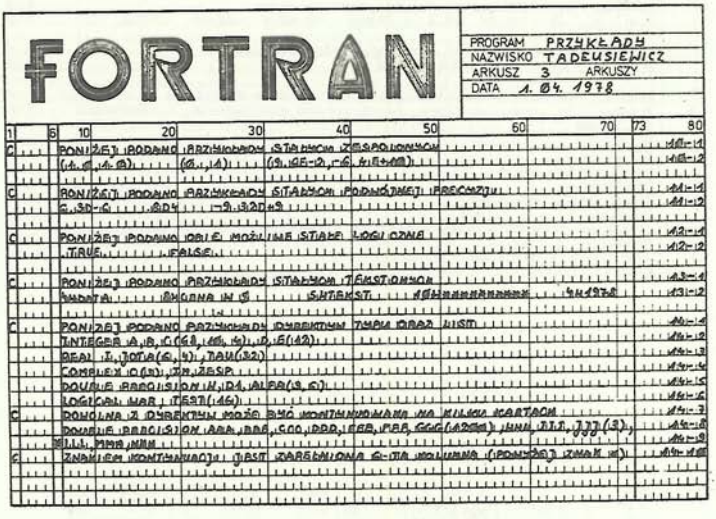


Rys. 13

ne są w FORTRANIE wartości rzeczywiste, których strukturę (dla stałych) podaje rysunek 13, a przykłady podano na rysunku 12 w wierszach 7-1 i 7-2. Zmienne rzeczywiste mogą zaczynać się na dowolną literę, z wyjątkiem I, J, K, L, M, N, lub muszą być deklarowane jako należące do typu REAL (patrz dalej, a także rys. 12 wiersze 8-1 i 8-2). Jak łatwo zauważyć z porównania przykładów stałych całkowitych i rzeczywistych, te ostatnie wyróżniają się posiadaniem części ułamkowej (której wartość może zresztą być zerowa, tak jak w przykładzie stałej 6., co jest równoważne zapisowi 6.0). Rozróżnienie między wartościami całkowitymi i rzeczywistymi jest dla osób rozpoczynających praktyczne progra-



Rys. 14

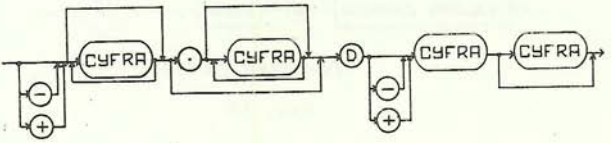


Rys. 15

mowanie sztuczne i niewygodne, niemniej musi być rygorystycznie przestrzegane. Wynika to z faktu, że maszyna przyjmuje zawsze taki sam typ wyniku operacji arytmetycznej jak typ argumentów wchodzących w jej skład. Jeśli zatem zaprogramujemy działanie, w ramach którego dochodzi do dzielenia liczb całkowitych, to wówczas ewentualna część ułamkowa wyniku dzielenia jest obcinana (bez przybliżenia!). Fakt ten jest niekiedy przydatny, może jednak prowadzić do błędów przy niefrasobliwym mieszaniu typów. Na rysunku 12 w wierszach 9-1 do 9-12 pokazano, jakie skutki ma dzielenie liczb całkowitych, a jakie - dzielenie liczb rzeczywistych. Należy zauważyć także, że przy podstawieniu wartości rzeczywistej za zmienną typu całkowitego następuje prawidłowe przybliżenie wyniku.

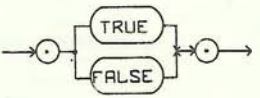
W języku FORTRAN możliwe jest także operowanie liczbami zespolo-

STAŁA PODWÓJNEJ PRECYZJI



Rys. 16

STAŁA LOGICZNA

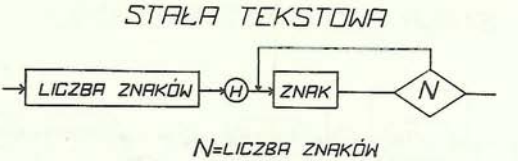


Rys. 17

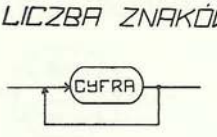
nymi. Postać stałej zespolonej podana jest na rysunku 14, natomiast przykłady stałych zespolonych – na rysunku 15 (wiersz 10-1 i 10-2). Zmienne typu zespolonego muszą być deklarowane w dyrektywie typu jako COMPLEX. Niekiedy obliczenia trzeba wykonywać z większą dokładnością niż ta, która wynika z możliwości używanej maszyny cyfrowej (dokładność taka dla Odry serii 1300 wynosi około 10^{-11}). Można wówczas posługiwać się wartościami podwójnej precyzji, których dokładność jest około dwukrotnie większa (dla Odry 10^{-21}), chociaż możliwości tej nie należy nadużywać, gdyż czas liczenia wydłuża się wówczas niewspółmiernie. Strukturę stałych podwójnej precyzji zdefiniowano na rysunku 16, a przykłady podano na rysunku 15 w wierszach 11-1 i 11-2.

Obok obliczeń wykonywanych na liczbach maszyna cyfrowa może przeprowadzać kalkulacje logiczne, wykorzystując wartości logiczne. Postać stałej logicznej podano na rysunku 17 oraz na rysunku 15 (wiersze 12-1 i 12-2). Zmienne logiczne i podwójnej precyzji, podobnie jak zespolone, muszą być deklarowane w dyrektywie typu jako LOGICAL i DOUBLE PRECISION, odpowiednio.

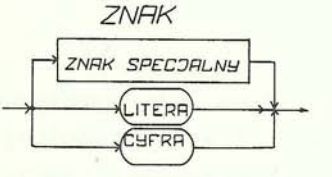
Ostatnim typem wartości występującym w języku FORTRAN są teksty. Stała tekstowa jest ciągiem znaków poprzedzonym przez literę H i liczbę znaków (por. rys. 18 i rys. 15, wiersze 13-1 i 13-2). W FORTRANIE nie istnieją zmienne typu tekst, gdyż teksty można umieszczać w dowolnych zmiennych (całkowitych, rzeczywistych, zespolonych itd.). Pamiętać tylko trzeba, że różne zmienne mają rozmaitą pojemność. W przypadku Odry 1300 pojemności te wynoszą odpowiednio: 4 znaki dla zmiennych



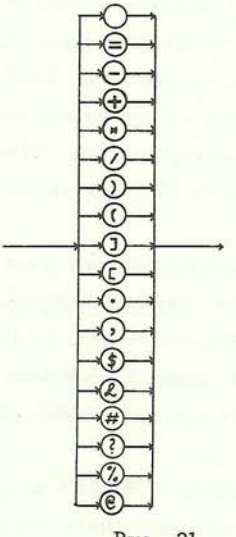
Rys. 18



Rys. 19



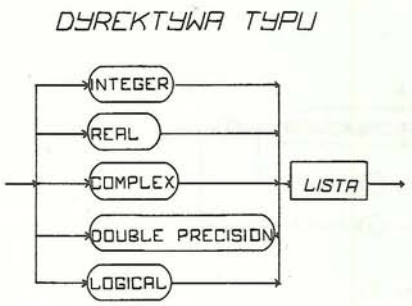
Rys. 20



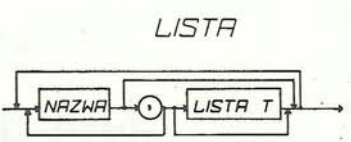
Rys. 21

Rys. 22

całkowitych i logicznych, 8 dla zmiennych rzeczywistych i 16 dla zespolonych czy podwójnej precyzji. Wartości typu tekstowego używane są do redagowania wydruków w sposób przedstawiony dalej. Pojęcia użyte w definicji stałej tekstowej zilustrowano na rysunku 19, 20, 21.



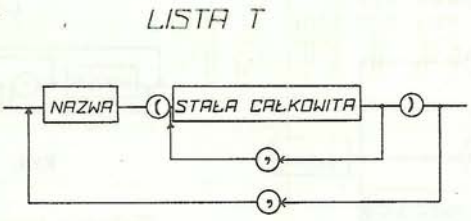
Rys. 23



Rys. 24

Omówimy teraz poszczególne elementy wchodzące w skład segmentów programu w języku FORTRAN. Pamiętamy (por. rys. 2, 5 i 8), że poza elementami odmiennymi w każdym segmencie (MASTER, RETURN, FUNCTION itp.), segment składa się z instrukcji i dyrektyw. Rodzaje dyrektyw używanych w języku FORTRAN podano na rysunku 22. Pierwszą w kolejności omawianą dyrektywą jest dyrektywa typu, służąca do deklarowania typu zmiennych (rys. 23 i rys. 15, wiersze 14-1 do 14-10). Pozwala ona nadać zmiennym wchodzącym w skład listy odpowiedni typ: całkowity (INTEGER), rzeczywisty (REAL), zespolony (COMPLEX), podwójnej precyzji (DOUBLE PRECISION) lub logiczny (LOGICAL). Zwróćmy uwagę, że słowa INTEGER, REAL, COMPLEX itd. wystąpiły już wcześniej (patrz rys. 8). Użyte mogły być w nagłówku segmentu funkcji, nadając tej funkcji typ. Zwróćmy uwagę, że w nagłówku funkcji słowa te można było pominąć (linia ciągła poniżej słowa LOGICAL na rys. 8). Taki skrócony wariant zapisu funkcji zastosowano w przykładzie na rysunku 3 (wiersze 3-1 do 3-5). Typ takiej funkcji nadawany jest przez samą maszynę na podstawie pierwszej litery nazwy funkcji: nazwy zaczynające się na literę I, J, K, L, M, N zostaną potraktowane jako nazwy funkcji całkowitej, a pozostałe - funkcji rzeczywistej. Tak więc funkcja SUMA1 z przykładu na rysunku 3 będzie potraktowana jako funkcja rzeczywista. Wspomniano wyżej, że identyczne zasady obowiązują w odniesieniu do zmiennych: jeśli jakaś zmienna pojawi się w instrukcjach dowolnego segmentu, a nie była deklarowana w dyrektywie typu należącej do tegoż segmentu, to zostanie jej nadany typ automatycznie.

Wracając do dyrektywy typu musimy wyjaśnić, co to jest lista (rys. 24). Jak widać, lista musi zawierać przynajmniej jedną nazwę, może jed-



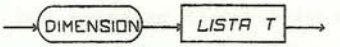
Rys. 25

nak zawierać ich dowolnie dużo, a także może zawierać "listę t". Na liście t (rys. 25) są nazwy tych zmiennych, których w programie zamierzamy używać jako tablice. Zmienne te muszą być deklarowane, także w przypadku, kiedy ich nazwa (pierwsza litera) jednoznacznie definiuje ich typ, gdyż dla zmiennych tych trzeba podać maksymalne rozmiary tablic i liczbę indeksów. Maksymalne wymiary podaje się po nazwie zmiennej w nawiasach, rozdzielając je przecinkami. Tak więc zapis

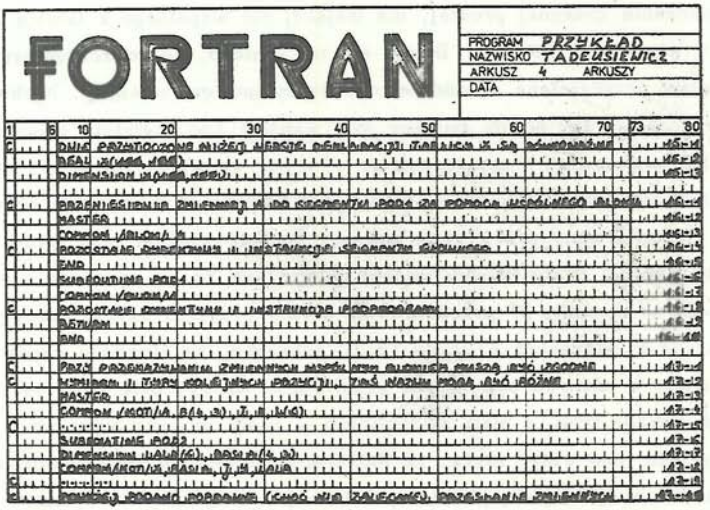
REAL A(4), B(3,4,5), C(12,2)

oznacza nie tylko nadanie zmiennym A, B i C typu rzeczywistego, ale deklaruje dodatkowo, że będą to zmienne traktowane jako tablice, przy czym pierwsza z nich ma 4 elementy: A(1), A(2), A(3) i A(4), druga ma 60 elementów (trzy "warstwy", cztery kolumny i pięć wierszy), a trzecia ma 24 elementy (12 kolumn, dwa elementy w każdym wierszu). Zmiennych, będących tablicami, będziemy często używali w dalszych przykładach i tam zostaną podane szczegółowe informacje na ich temat. Tablice deklarować można także bez podania typu (określa go wtedy pierwsza litera) w dyrektywie wymiaru podanej na rysunku 26 oraz na rysunku 27 (wiersze 15-1 do 15-3). Ten drugi sposób, z użyciem dyrektywy wymiaru, jest często stosowany w praktyce, gdyż programiści piszący swoje programy w języku FORTRAN nader często posługują się w oznaczaniu typów swoich zmiennych konwencją pierwszej litery nazwy (zmiennych typu in-

DYREKTYWA WYMIARU

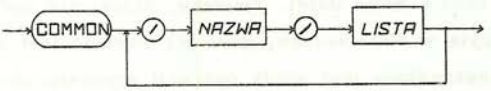


Rys. 26



Rys. 27

DYREKTYWA PAMIĘCI



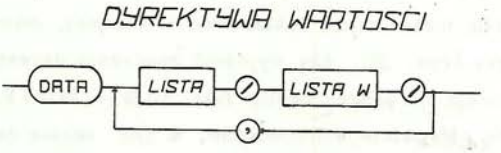
Rys. 28

nego niż całkowitego i rzeczywistego używa się rzadko). Jest to praktyka godna polecenia, gdyż zabezpiecza przed pomyłkami.

Rozmiary tablic można także deklarować w kolejnej, omawianej teraz, dyrektywie pamięci (rys. 28). Aby wyjaśnić znaczenie dyrektywy pamięci, trzeba zwrócić uwagę na pewien ogólny fakt. Otóż w FORTRANIE poszczególne segmenty są całkowicie autonomiczne, w tym sensie że wszystkie oznaczenia użyte w obrębie jednego segmentu nie mają żadnego wpływu na inne segmenty programu. W szczególności nazwy zmiennych mogą być w każdym segmencie nadawane niezależnie i jeśli w segmencie głównym (MASTER) używano zmiennej X jako, na przykład, tablicy, to w segmen-

cie podprogramu np. SUBROUTINE A tej samej nazwy X można używać do oznaczania zmiennej prostej, nie mającej nic wspólnego z tablicą używaną w segmencie głównym. Dzieje się tak dlatego, że poszczególnym segmentom przydzielane są oddzielne obszary pamięci maszyny. Niekiedy jednak chcemy, aby pewne zmienne były wspólne (po angielsku common) dla kilku segmentów. Umieszczamy je wówczas we wspólnym bloku, który zlokalizowany jest w pamięci maszyny poza obszarami wszystkich segmentów i identyfikowany przez swoją nazwę. Do tego właśnie celu służy dyrektywa pamięci o strukturze podanej na rysunku 28. W wierszach 16-1 do 16-10 na rysunku 27 przedstawiono prosty przykład użycia dyrektywy pamięci, a w wierszach 17-1 do 17-10 pokazano, że elementy wspólnego bloku mogą w różnych segmentach nosić różne nazwy, byle tylko były takiego samego typu, i jeśli są tablicami - miały te same wymiary. Z przykładu tego widać także, że deklarację rozmiarów tablicy można umieszczać w dyrektywie pamięci podobnie jak w dyrektywie wymiaru czy typu, nie wolno jedynie żadnej tablicy zadeklarować dwukrotnie, gdyż jest to zawsze wykrywane jako błąd.

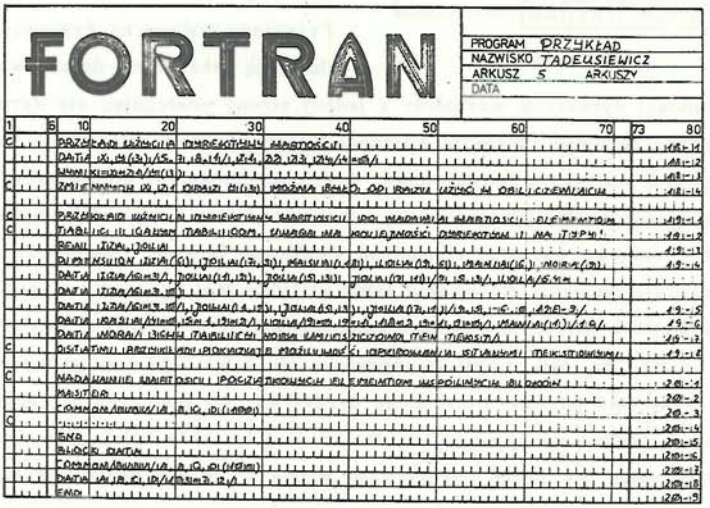
Zmiennym nadaje się w trakcie wykonywania programu wartości i następnie wartości tych używa się w dalszych obliczeniach. Nadanie wartości zmiennej możliwe jest za pomocą instrukcji podstawienia lub czytania, o których będzie mowa dalej. Zmienna, której nie nadano wartości, nie może być użyta w obliczeniach, gdyż jej wartość jest nieokreślona, a tym samym nieokreślony jest wynik operacji arytmetycznych, wykonywanych z użyciem tej zmiennej. Możliwe jest jednak nadanie zmiennej wartości przed rozpoczęciem wykonywania programu. Do takiego właśnie nadawania zmiennym wartości początkowych służy dyrektywa wartości o struk-



Rys. 29

turze podanej na rysunkach 29 i 30. Na rysunku 31 podano trzy przykłady zastosowania dyrektywy wartości.

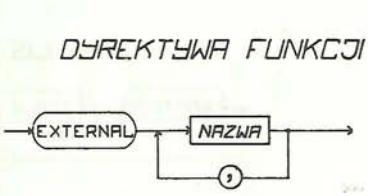
Rys. 30



Rys. 31

Analizując te przykłady, warto zwrócić uwagę na następujące elementy: W dyrektywie wartości lista zmiennych i lista wartości umieszczona pomiędzy znakami / muszą dokładnie sobie odpowiadać zarówno co do liczby, kolejności, jak i typów poszczególnych wartości. W dyrektywie wartości można nadawać wartości dowolnym wybranym elementom tablic, a także wszystkim elementom tablicy, w tym ostatnim przypadku w liście zmiennych umieszcza się nazwę tablicy bez indeksów w nawiasach, podając wartość dowolnego ustalonego elementu tablicy wymieniamy jego koordynaty. Tak więc Y(3) oznacza trzeci element tablicy Y, zaś zapis JOLA(5,3) (patrz wiersz 19-5 na rysunku 31) oznacza element na przecięciu piątej kolumny i drugiego wiersza tablicy JOLA, przy czym element ten należy do tablicy, gdyż jej maksymalne wymiary (por. wiersz 19-4)

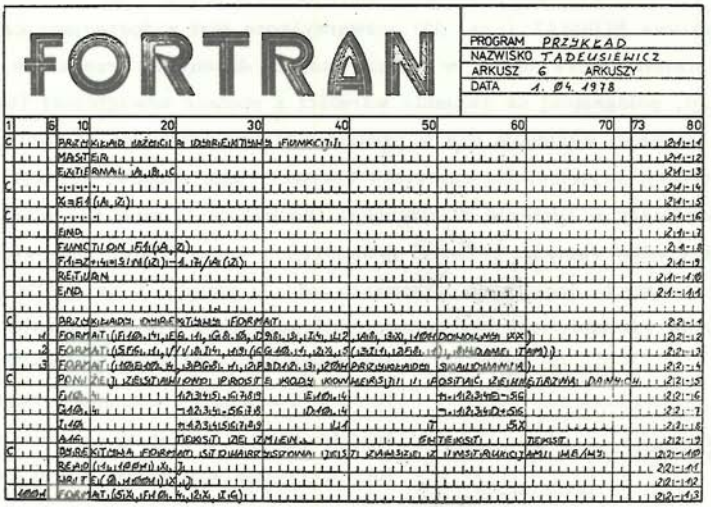
Rys. 32



Rys. 33

wynoszą 7 kolumn i 3 wiersze. Przykłady podane na rysunku 31 ilustrują także dwie dalsze ważne własności dyrektywy wartości: z jednej strony posługując się dyrektywą można nadawać zmiennym wartości tekstowe, czego nie można robić na przykład instrukcją podstawienia, z drugiej natomiast strony istnieje istotne ograniczenie, polegające na zakazie nadawania wartości początkowych dyrektywą wartości elementom wspólnych bloków, zadeklarowanym w dyrektywie pamięci. Nadawanie takie możliwe jest wyłącznie w segmencie BLOCK DATA, a równocześnie segment BLOCK DATA jedynie służy właśnie do tego. Czyli w segmencie BLOCKA DATA mogą występować wyłącznie poznane dotychczas dyrektywy, co jest zresztą zgodne z definicją podaną na rysunku 10. Ważna jest tu także kolejność: dyrektywy typu mogą być przed dyrektywami wymiaru, te zaś przed dyrektywami pamięci i wartości.

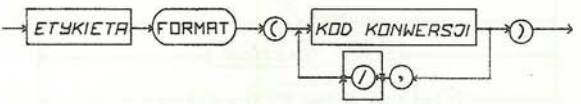
Następna w kolejności omawiania jest dyrektywa funkcji o strukturze podanej na rysunku 33. Dyrektywa ta służy do wskazania, że pewne nazwy (wymienione w niej po słowie EXTERNAL) powinny być traktowane jako nazwy segmentów (podprogramów lub funkcji), a nie jako nazwy zmiennych. Dyrektywy funkcji używa się zazwyczaj wtedy, gdy potrzeba określoną nazwę podprogramu lub funkcji przekazać przez parametry aktualne do innego podprogramu lub funkcji (patrz dalej, a także por. przykład w wierszach 21-1 do 21-11 na rysunku 34). Warto chwilę zastanowić się nad przykładem z rysunku 34, pomimo że nie wszystkie elementy na nim są znane. Zwróćmy uwagę, że w wierszu 21-3 w segmencie głównym użyto dyrektywy funkcji, która uprzedziła maszynę, że nazwy A, B i C zarezerwowano dla segmentów. W tej sytuacji gdy w wierszu 21-5 przy wywoływaniu funkcji F1 użyto jako parametru nazwy A,



Rys. 34

wiadomo było że chodzi o nazwę segmentu, a nie nazwę zmiennej. Zwróćmy uwagę, że gdyby nie było dyrektywy funkcji, nazwa A bez wątpienia potraktowana byłaby jako nazwa zmiennej, gdyż nic nie wskazuje, że może być inaczej. Rozważmy teraz sytuację w segmencie funkcji F1 (wiersze 21-8 do 21-11). W tym przypadku nie potrzeba deklarować A jako nazwy funkcji w dyrektywie funkcji, gdyż użycie tej nazwy (wiersz 21-9)

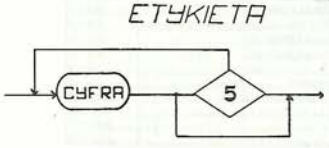
DYREKTYWA FORMAT



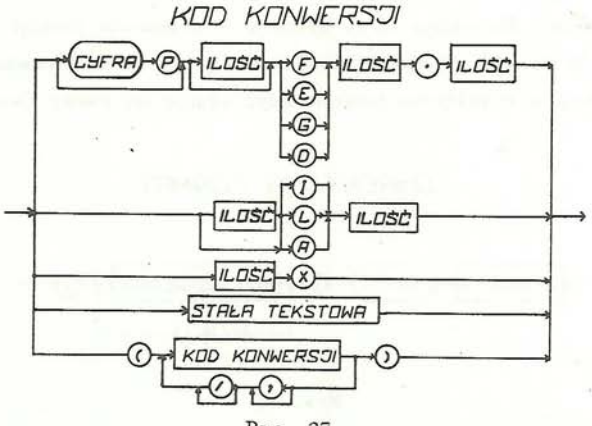
Rys. 35

zupełnie jasno wskazuje, że chodzi o funkcję. Niemniej gdyby w segmencie tym użyto dyrektywy funkcji deklarującej nazwę A, nie byłoby to błędem, lecz zbytecznym instruowaniem maszyny o oczywistych faktach. Podobnie w segmencie głównym nie deklarowano nazwy F1, jakkolwiek można to zrobić.

Kolejna dyrektywa wymagać będzie znacznie szerszego omówienia. Dyrektywa FORMAT (rys. 35) stowarzyszona jest z operacjami czytania lub pisania (patrz dalej), w trakcie których dokonywać trzeba tzw. konwersji, polegającej na zamianie wartości z postaci wewnętrznej (binarnej, w jakiej występują one w pamięci maszyny cyfrowej) do postaci zewnętrznej, czytelnej dla człowieka (postaci ciągu znaków, będących zapisem liczby w systemie dziesiętnym). Konwersja ta musi być inna w przypadku wartości różnych wymienionych wyżej typów: całkowitych, rzeczywistych, logicznych tekstowych itd., a ponadto w przypadku drukowania wyników może być celowe zredagowanie wydruku w sensie odpowiedniego rozmieszczenia poszczególnych liczb w wierszach i kolumnach, zaopatrzenie wydruku w teksty informacyjne itd. Wszystkie wymienione wyżej czynności, związane z operacjami czytania lub pisania, sterowane są przez dyrektywę FORMAT. Jak widać z rysunku 35 dyrektywa FORMAT musi mieć etykietę (patrz rys. 36) i zawiera w nawiasach ciąg kodów konwersji (rys. 37), przy czym każdej drukowanej lub czytanej wartości odpowia-



Rys. 36

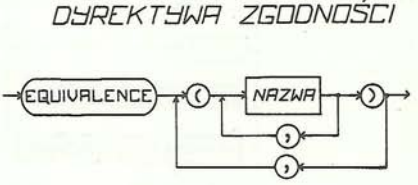


Rys. 37

dać musi w dyrektywie FORMAT stosowany do jej typu kod konwersji. Idąc od góry na rysunku 37 możemy stwierdzić, że kody konwersji zawierające literę F, E lub G służą do konwersji wartości rzeczywistych,

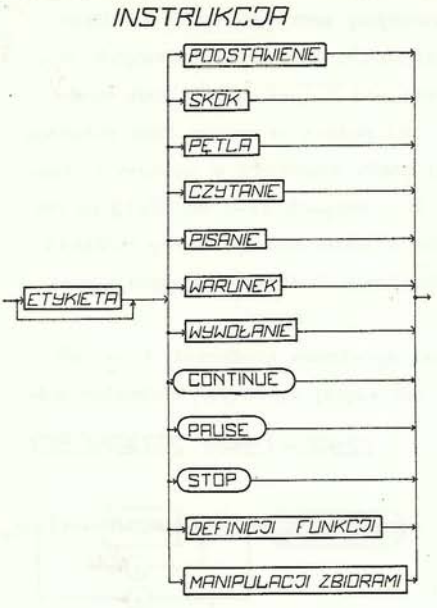
kod D przeznaczony jest dla liczb podwójnej precyzji, kod I dla liczb całkowitych, kod L dla zmiennych logicznych, kod A dla zmiennych zawierających teksty, stałe tekstowe oraz kod X (nakazujący druk odstępów) służą do redagowania wydruku, zaś podana na samym dole rekursja przewiduje możliwość ujmowania grup kodów konwersji w nawiasy i traktowanie ich jak kodów pojedynczych. W wierszach 22-1 do 22-13 na rysunku 34 zilustrowano niektóre aspekty użytkowania dyrektywy FORMAT. Do zagadnień tych powrócimy przy omawianiu instrukcji czytania i pisania.

Ostatnią omawianą dyrektywą jest dyrektywa zgodności (rys. 38). Pozwala ona utożsamić ze sobą dwie lub więcej zmiennych w obrębie jednego segmentu. Przykładowo zapis EQUIVALENCE (A, B) oznacza, że w całym segmencie zarówno nazwa A, jak i nazwa B, będą odnosiły się do jednej i tej samej komórki pamięci. W jednej dyrektywie zgodności można deklarować kilka takich kompletów, na przykład EQUIVALENCE (X, Y, Z), (G, H), (C(1,2),C1) powoduje, że do jednej i tej samej komórki pamięci odnoszą się nazwy X, Y i Z, do innej, ale także wspólnej - nazwy G i H, oraz pozwala używać elementu C(1,2) tablicy C (tablica ta musi być wcześniej deklarowana!) pod nazwą C1.

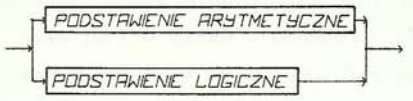


Rys. 38

Dyrektywy pełnią rolę pomocniczą, zasadnicze obliczenia prowadzone są jednak na podstawie instrukcji, których zestawienie podano na rysunku 39. Jak widać każda instrukcja może, lecz nie musi mieć etykiety. (Z dyrektyw jedynie dyrektywa FORMAT mogła mieć etykietę, ale tam nie było wyboru - etykieta musiała być.) Na rysunku 39 podano trzy instrukcje w ostatecznej postaci. Ich znaczenie jest bardzo proste. Instrukcja CONTINUE jest instrukcją pustą ("nic nie rób przez okres jednego taktu") i używaną najczęściej jako adresowalny, neutralny punkt w programie, do którego można wykonywać skok. Liczne przytaczane dalej przykłady użycia instrukcji CONTINUE wyjaśnią niewątpliwie tę sprawę do końca. Dwie dalsze instrukcje STOP i PAUSE zatrzymują wykonanie



Rys. 39

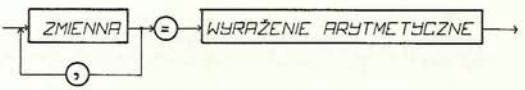


Rys. 40

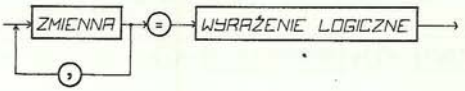
z pamięci, zaś PAUSE zatrzymuje wykonanie, ale umożliwia start programu jeszcze raz od początku, albo dalej od miejsca zatrzymania po wykonaniu odpowiednich czynności (na przykład po założeniu nowego pudła kart do czytnika). W przypadku używania PAUSE trzeba do programu dołączyć instrukcję dla operatora maszyny cyfrowej, aby wiedział, co zrobić po zatrzymaniu maszyny. Ze względu na oszczędność czasu zaleca się stosować w miarę możności instrukcję STOP, która zwalnia maszynę zaraz po wykonaniu programu.

Najpowszechniej używaną instrukcją w każdym programie jest instrukcja podstawienia. Jak wynika z rysunków 40 i 41, może to być podstawienie arytmetyczne lub podstawienie logiczne. Struktura instrukcji podstawienia jest taka (por. rys. 41), że po lewej stronie znaku równości występuje zmienna (lub kilka zmiennych rozdzielonych przecinkami), a po prawej stronie - wyrażenie arytmetyczne. Pojęcie zmiennej jest nam już znane, będzie zresztą jeszcze dokładniej opisane na rysunku 43, wiemy jednak, że zmiennej można nadawać wartość. W instrukcji podstawienia jest to wartość obliczona z wyrażenia stojącego po prawej stronie znaku równości. Pojęcie wyrażenia arytmetycznego (rys. 42) jest w FORTRANIE nadzwyczaj pojemne (rys. 42), gdyż dowolny wzór matematyczny można zapisać w postaci wyrażenia. Od tej własności pochodzi nazwa języka: FORTRAN to skrót od formula translator (tłumacz wzorów). Jak widać z rysunku 42

PODSTAWIENIE ARYTMETYCZNE

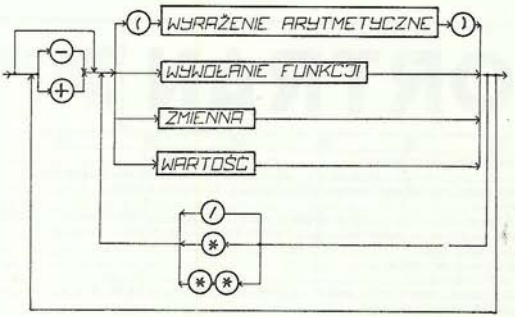


PODSTAWIENIE LOGICZNE



Rys. 41

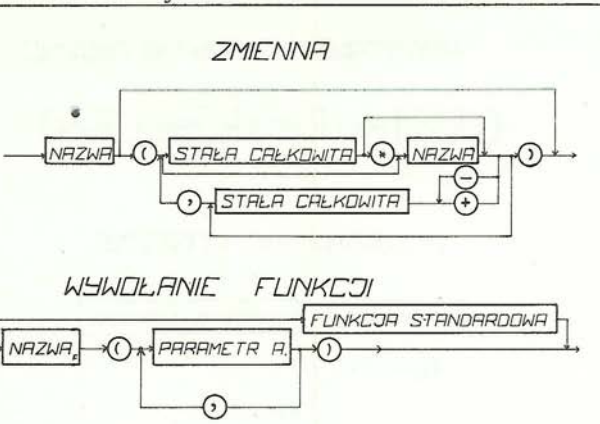
WYRAŻENIE ARYTMETYCZNE



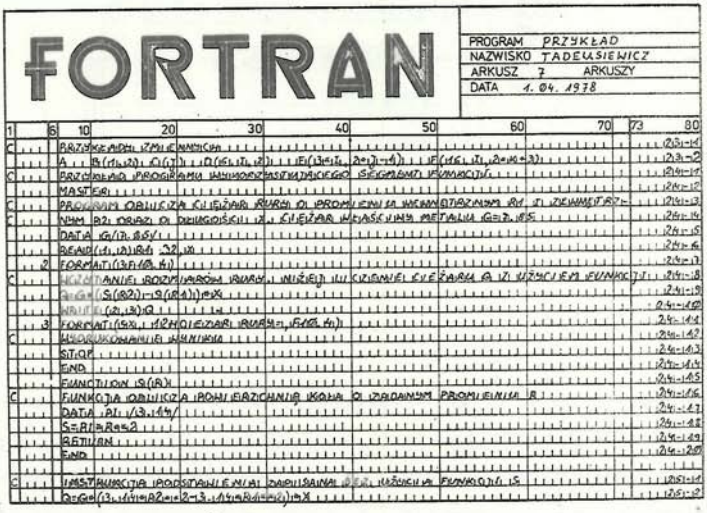
Rys. 42

elementami wyrażenia arytmetycznego są: stałe (wartości), zmienne, wywołania funkcji lub całe wyrażenia arytmetyczne ujęte w nawiasy, połączone znakami działań: + (dodawanie), - (odejmowanie), * (mnożenie), / (dzielenie), ** (potęgowanie). Kolejność wykonywania działań narzucona jest w naturalny sposób: na początku wykonywane są działania w nawiasach, potem wywołania funkcji, z kolei potęgowanie, mnożenie i dzielenie oraz dodawanie i odejmowanie.

Zanim przejdziemy do przykładów wyrażeń arytmetycznych i całych instrukcji podstawienia, sprecyzujemy pojęcie zmiennej i wywołania funkcji (rys. 43). Zmienna może być zmienną prostą i wówczas używamy wyłącznie jej nazwy, może jednak być także elementem tablicy i wówczas obok

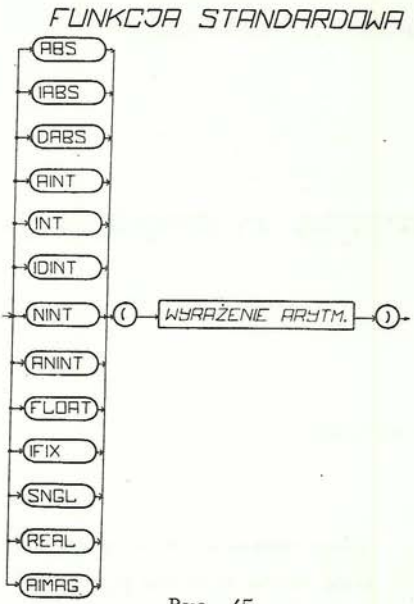


Rys. 43

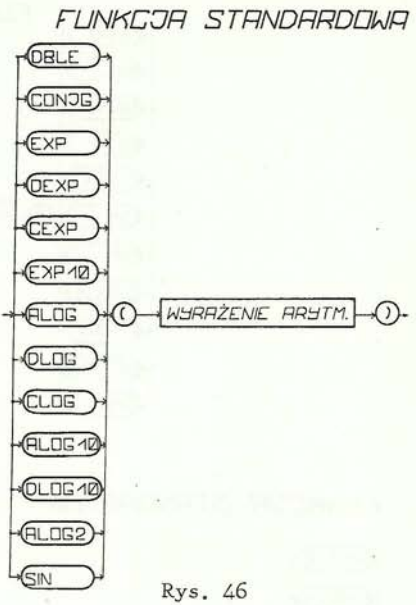


Rys. 44

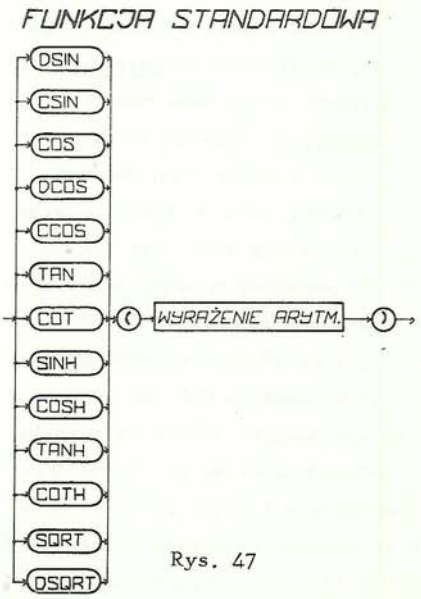
jej nazwy trzeba podać konkretne wartości kolejnych indeksów. Wartości te mogą być stałymi całkowitymi (najczęściej), zmiennymi typu INTEGER lub prostymi wyrażeniami, utworzonymi zgodnie z regułą widoczną na rysunku 43. Na rysunku 44 w wierszach 23-1 i 23-2 podano przykłady zmiennych odpowiadających definicji z rysunku 43. Wywołanie funkcji do-



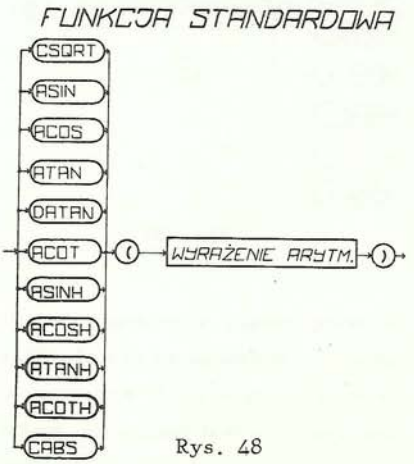
Rys. 45



Rys. 46

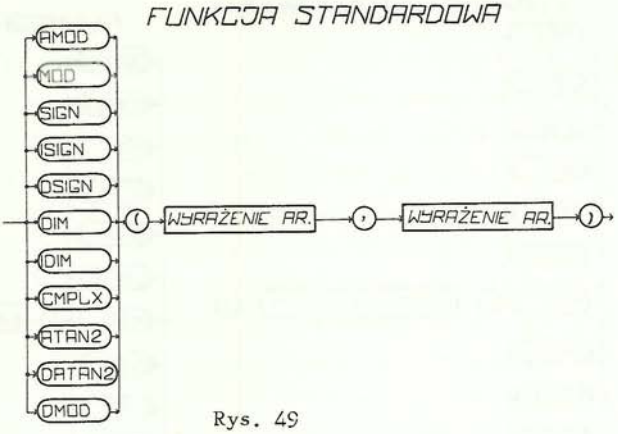


Rys. 47

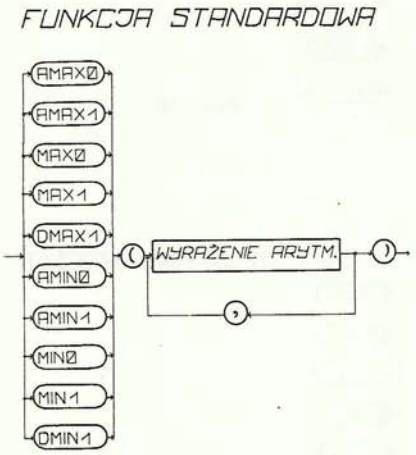


Rys. 48

tyczyć może funkcji standardowej lub funkcji napisanej przez programistę i dołączonej w postaci segmentu funkcji (por. rys. 7 i 8). W przypadku



Rys. 49



Rys. 50

funkcji napisanej przez programistę liczba oraz typ parametrów a. podanych przy wywołaniu funkcji musi być zgodna z liczbą i typami odpowiadających im parametrów f., użytych w nagłówku funkcji. Parametry a. stanowią argumenty funkcji i muszą reprezentować wartości znane w momencie wywołania funkcji (por. rys. 7). Na ich podstawie w segmencie funkcji następuje obliczenie wartości funkcji i ta właśnie wartość zostaje przekazana do obliczeń wyrażenia arytmetycznego, w którym wystąpiło wołanie funkcji. Rozważmy przykład programu podanego na rysunku 44 w wierszach 24-1 do 24-20, ilustrujący użycie funkcji. W wyrażeniu arytmetycznym (wiersz 24-9) dwukrotnie wywołana jest funkcja S, służąca do obliczania powierzchni koła, przy czym parametrami a. są odpowiednio R1 i R2. Zwróćmy uwagę, że zmienne te mają w segmencie głównym nadaną wcześniej wartość przez wykonanie instrukcji wczytania (wiersz 24-6), dzięki czemu wywołanie

jest sensowne. Podobnie pozostałe elementy wyrażenia arytmetycznego (G, X) mają wcześniej nadane wartości. W wyniku wykonania instrukcji podstawienia zostaje nadana wartość zmiennej Q, która zostaje potem wydrukowana (wiersz 24-10). W segmencie pomocniczym używany jest parametr f. o nazwie R, który występuje w wyrażeniu arytmetycznym 24-18. W trakcie obliczeń po wywołaniu funkcji obliczenia są prowadzone z podstawieniem za parametr f., użytego w wywołaniu, parametru a., tak więc w rzeczywistości wartość funkcji liczona jest dla R = R2 i R = R1, odpowiednio dla pierwszego i drugiego wywołania. Odnotujmy tu także zysk, jaki odnosimy stosując segment pomocniczy. Oczywiście możliwe byłoby zapisanie instrukcji obliczania wartości Q bez użycia funkcji S (patrz rys. 44 wiersze 25-1 i 25-2), ale jest to mniej wygodne i mniej czytelne, a także powoduje zajęcie większej pamięci maszyny.

Jak wynika z rysunku 43, obok wywołań funkcji, zdefiniowanych przez programistę, można używać funkcji standardowych, czyli funkcji mających w FORTRANIE ustaloną nazwę i ustalone znaczenie, nie wymagających definiowania. Funkcji standardowych jest dużo (rys. 45 do 50), przy czym w celu poprawnego wykorzystania konkretnej funkcji trzeba poza wywołaniem, którego strukturę podają wymienione rysunki, zapewnić właściwy typ argumentu i wyniku. Poniżej omówimy wszystkie funkcje, używając oznaczeń: I - wartości całkowite, R - wartości rzeczywiste, D - wartości podwójnej precyzji, C - wartości zespolone. Funkcje omawiamy w kolejności występowania na rysunkach.

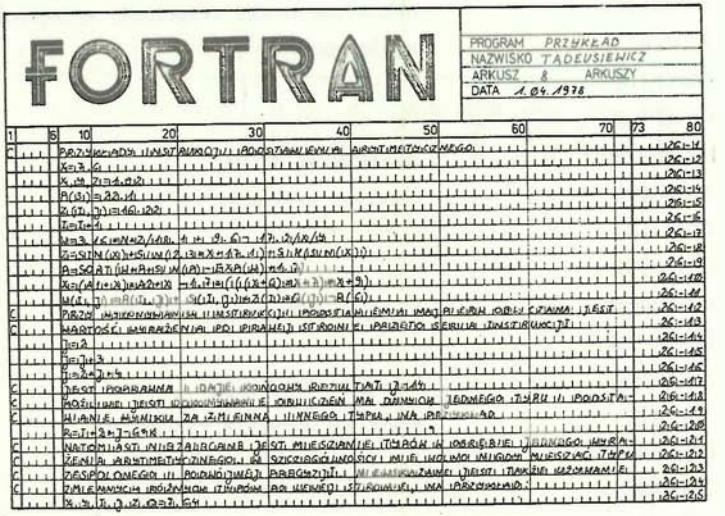
R = ABS(R)	wartość bezwzględna;
I = IABS(I)	"
D = DABS(D)	"
R = AINT(R)	część całkowita argumentu;
I = INT(R)	"
D = IDINT(D)	"
I = NINT(R)	najbliższa całkowita (zaokrąglenie!);
R = ANINT(R)	"
R = FLOAT(I)	zamiana całkowitej na rzeczywistą;
R = ASIN(R)	arkus sinus;
R = ACOS(R)	arkus kosinus;

R = ATAN(R)	arkus tangens;
D = DATAN(D)	"
R = ACOT(R)	arkus kotangens;
R = ASINH(R)	arkus sinus hiperboliczny;
R = ACOSH(R)	arkus kosinus hiperboliczny;
R = ATANH(R)	arkus tangens hiperboliczny;
R = ACOTH(R)	arkus kotangens hiperboliczny;
R = CABS(C)	moduł liczby zespolonej;
R3 = AMOD(R1,R2)	reszta z dzielenia R1 przez R2 (modulo);
I3 = MOD(I1,I2)	"
R3 = SIGN(R1,R2)	R2 ze znakiem R1 (przeniesienie znaku);
I3 = ISIGN(I1,I2)	"
D3 = DSIGN(D1,D2)	"
R3 = DIM(R1,R2)	moduł różnicy R1 - R2;
I3 = IDIM (I1,I2)	"
C = CMPLX(R1,R2)	utworzenie liczby zespolonej o części rzeczywistej R1 i urojonej R2;
R3 = ATAN2(R1,R2)	arkus tangens ilorazu R1/R2;
D3 = DATAN2(D1,D2)	"
D3 = DMOD(D1,D2)	funkcja modulo dla liczb podwójnej precyzji;
R = AMAXO(I1,I2, ...)	wartość maksymalna;
R = AMAX1(R1,R2, ...)	"
I = MAXO(I1, I2, ...)	"
I = MAX1(R1, R2, ...)	"
D = DMAX1(D1, D2, ...)	"
R = AMINO(I1,I2, ...)	wartość minimalna;
R = AMIN1(R1,R2, ...)	"
I = MINO(I1, I2, ...)	"
I = IFIX(R)	zamiana rzeczywistej na całkowitą;
R = SNGL(D)	zamiana podwójnej precyzji na rzeczywistą;
R = REAL(C)	zamiana zespolonej na rzeczywistą (cz. rzecz.);
R = AIMAG(C)	zamiana zespolonej na rzeczywistą (cz. uroj.);

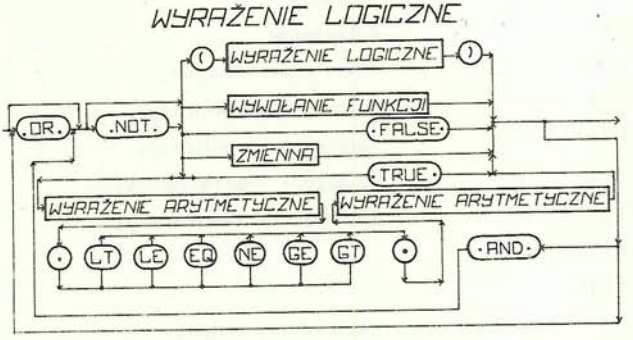
C = CONJG(C)	obliczenie liczby zespolonej sprzężonej;
R = EXP(R)	wykładnicza (e^x);
D = DEXP(D)	"
C = CEXP(C)	"
R = EXP10(R)	wykładnicza (10^x);
R = ALOG(R)	logarytm naturalny;
D = DLOG(D)	"
C = CLOG(C)	"
R = ALOG10(R)	logarytm dziesiętny;
D = DLOG10(D)	"
R = ALOG2(R)	logarytm dwójkowy;
R = SIN(R)	sinus (argument w radianach);
D = DSIN(D)	"
C = CSIN(C)	"
R = COS(R)	kosinus (argument w radianach);
D = DCOS(D)	"
C = CCOS(C)	"
R = TAN(R)	tangens;
R = COT(R)	kotangens;
R = SINH(R)	sinus hiperboliczny;
R = COSH(R)	kosinus hiperboliczny;
R = TANH(R)	tangens hiperboliczny;
R = COTH(R)	kotangens hiperboliczny;
R = SQRT(R)	pierwiastek kwadratowy;
D = DSQRT(D)	"
C = CSQRT(C)	"
I = MIN1(R1, R2, ...)	wartość minimalna;
D = DMIN1(D1, D2, ...)	"

Na rysunku 51 podano przykłady instrukcji podstawienia, wykorzystujących rozmaite możliwe postacie wyrażenia arytmetycznego. Wykaz ten jest niewątpliwie niepełny, lecz liczne dalsze przykłady instrukcji podstawienia arytmetycznego napotykać będziemy we wszystkich dalszych, a także w niektórych wcześniej podanych przykładach.

Instrukcje podstawienia logicznego spotyka się w programach raczej rzadko, niemniej wyrażenie logiczne, opisane na rysunku 52, jest ważnym

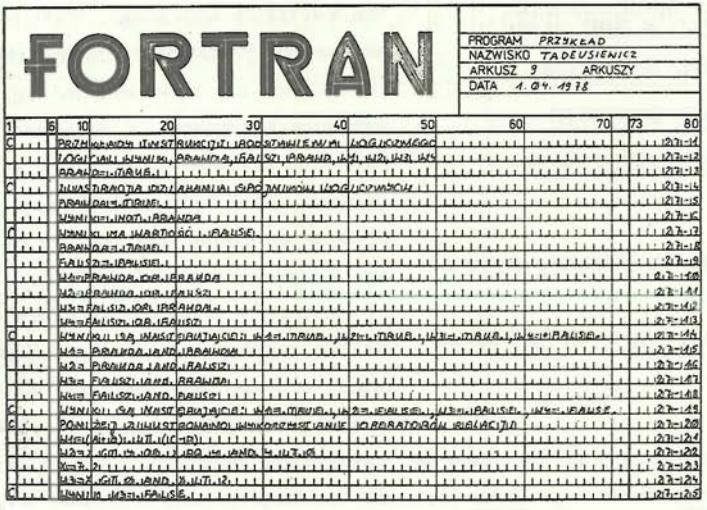


Rys. 51



Rys. 52

elementem także i innych instrukcji, dlatego warto je poznać. Dla zrozumienia znaczenia zapisów wyrażeń logicznych trzeba znać znaczenie słów i skrótów, występujących na rysunku 52. Poświęcimy im teraz nieco uwagi. Zapisy .TRUE. i .FALSE. są znanymi już stałymi logicznymi, oznaczającymi odpowiednio prawdę i fałsz (por. rys. 17). Spójniki logiczne .NOT. , .OR. oraz .AND. oznaczają w kolejności negację, alter-



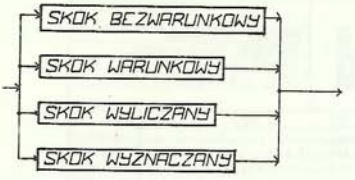
Rys. 53

natywę i koniunkcję. Ich znaczenie wyjaśniono w przykładach na rysunku 53. Bardzo często używane są operatory relacji .LT. , .LE. , itd., występujące na rysunku 52 pomiędzy dwoma wyrażeniami arytmetycznymi. Ich znaczenia są następujące:

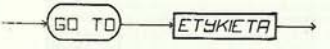
operator	znaczenie	przykład zastosowania
.LT.	$<$	A.LT.ϕ
.LE.	$\leqslant$	A+B .LE. C-D
.EQ.	$=$	A .EQ. B
.NE.	$\neq$	B .NE. C + C/A
.GE.	$\geqslant$	7.6 .GE. Z
.GT.	$>$	Z . GT. (A + B)/SIN(A).

Zapis: wyrażenie arytmetyczne – operator – wyrażenie arytmetyczne jako całość stanowi wartość logiczną, gdyż operator relacji stwierdza istnienie określonej relacji między wyrażeniami, co może być prawdą lub nie. Przykładowo podany wyżej zapis A.LT.ϕ stwierdza, że aktualna wartość zmiennej A jest ujemna. Może to być prawda lub nie, zależnie

SKOK



SKOK BEZWARUNKOWY



Rys. 54

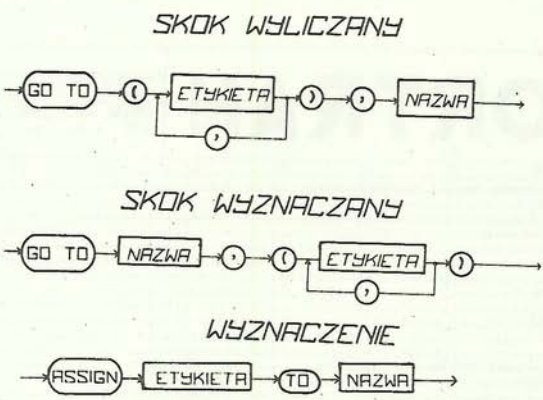
od tego, czy istotnie A jest ujemne czy nie, a zatem cały ten zapis jest elementem wyrażenia logicznego. Występujące na rysunku 52 dalsze elementy, takie jak zmienna czy wywołanie funkcji nie wymagają komentarza, z wyjątkiem odnotowania oczywistego faktu, że zarówno zmienna, jak i funkcja, jeśli mają być użyte w wyrażeniu logicznym to muszą być deklarowane jako LOGICAL.

Instrukcje programu FORTRANOWSKIEGO wykonywane są w takiej kolejności, w jakiej były napisane. Nazwane jest to pracą sekwencyjną. Ten tryb pracy może zostać zmieniony po zastosowaniu instrukcji skoku lub instrukcji pętli. Instrukcja skoku (rys. 54) nakazuje zmianę kolejności w ten sposób, że wskazuje bezpośrednio etykietę instrukcji, która ma być wykonana jako bezpośrednio następna. Wyróżnia się kilka typów skoków, przy czym najprostszy jest skok bezwarunkowy (rys. 54). Przykłady skoków bezwarunkowych podano na rysunku 57 w wierszach 28-1 do 28-11. Należy zwrócić uwagę, że możliwe są zarówno skoki do przodu

SKOK WARUNKOWY

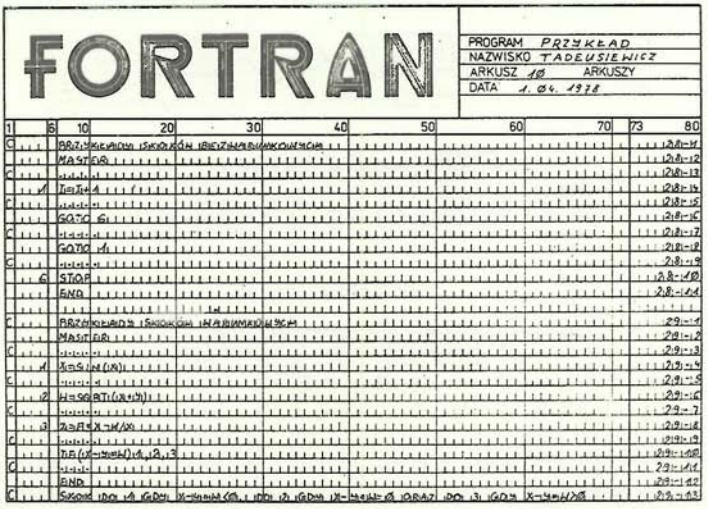
Rys. 55

(instrukcja z etykietą wymienioną po GOTO jest dalej od samej instrukcji GO TO), jak i do tyłu. Etykieta wymieniona w instrukcji GO TO musi występować w segmencie, gdyż inaczej sygnalizowany jest błąd i program nie nadaje się do wykonania. Bogatsze możliwości niż skok bezwarunkowy ma skok warunkowy, w którym podane są trzy etykiety (rys. 55), a skok

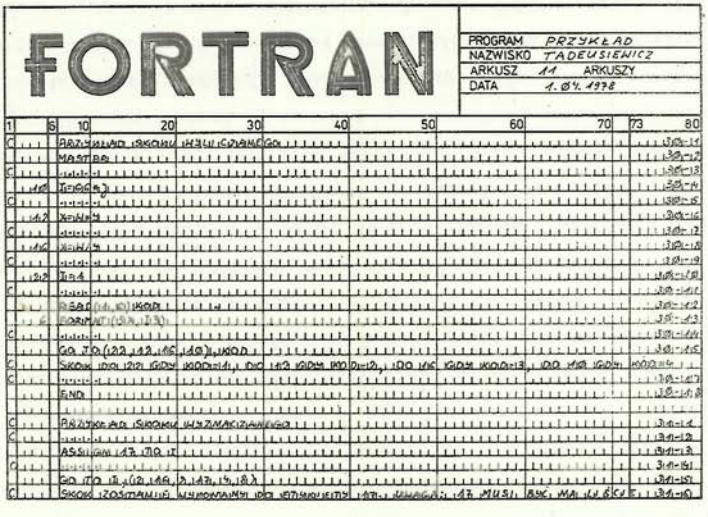


Rys. 56

wykonywany jest do jednej z nich, zależnie od wartości wyrażenia arytmetycznego podanego w nawiasie. Jeśli wyrażenie to (po obliczeniu) ma wartość ujemną, to skok następuje do pierwszej w kolejności etykiety, jeśli wartość wynosi zero, to skok wykonywany jest do drugiej etykiety, jeśli wartość jest dodatnia, to skok wykonywany jest do trzeciej etykiety. Przykłady skoków warunkowych podano na rysunku 57 w wierszach od 29-1 do 29-13. Trzy etykiety występujące w instrukcji skoku warunkowego niekoniecznie muszą być różne, koniecznie jednak muszą być trzy i koniecznie muszą odpowiadać konkretnym etykietom, występującym w rozważanym fragmencie programu. Na rysunku 56 podano definicję dwu dalszych typów skoków. Skok wyliczany, nazywany także przełącznikiem, służy do wskazywania etykiety, do której ma nastąpić skok za pomocą zmiennej całkowitej, występującej po ujętej w nawiasy liście etykiet (rys. 58, wiersze 30-1 do 30-18). Skok następuje do tej listy, w kolejności etykiety, której numer odpowiada aktualnej wartości zmiennej. W odróżnieniu od tego skok wyznaczany, stowarzyszony z instrukcją wyznaczania, która musi go poprzedzać, pozwala na wykonanie skoku do tej etykiety, którą nadano (ASSIGN = nadaj) zmiennej występującej przed listą etykiet. Wydaje się, że w praktyce znacznie wygodniejszy jest skok wyliczany, gdyż obserwując setki programów pisanych przez różnych programistów autor ani razu nie spotkał użycia instrukcji skoku wyznaczanego.

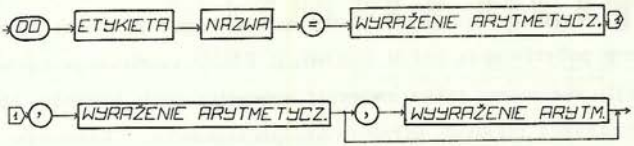

FORTRAN
PROGRAM PRZYKŁAD
NAZWISKO TADEUSIEWICZ
ARKUSZ 10 ARKUSZY
DATA 1.04.1978

Rys. 57

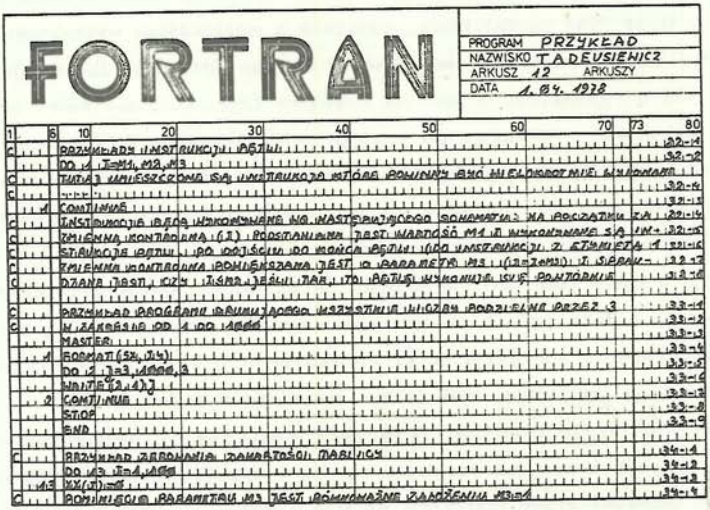

FORTRAN
PROGRAM PRZYKŁAD
NAZWISKO TADEUSIEWICZ
ARKUSZ 11 ARKUSZY
DATA 1.04.1978

Rys. 58

PĘTLA



Rys. 59

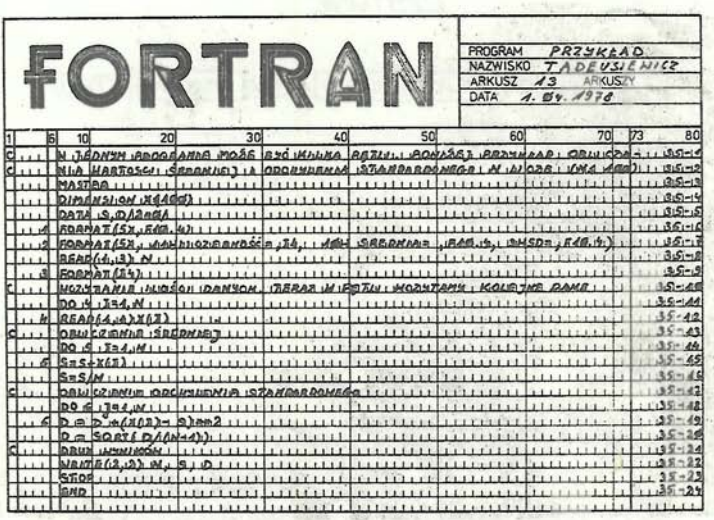


Rys. 60

Niezwykle istotną konstrukcją przy programowaniu w dowolnym języku programowania jest pętla. W FORTANIE do programowania pętli służy specjalna instrukcja (rys. 59). Pętlą jest w istocie grupa instrukcji, które powinny być przez maszynę wykonane zadaną ilość razy. Instrukcja pętli wskazuje, które instrukcje mają być wielokrotnie wykonane, ile razy mają być wykonane oraz w jaki sposób mają się kolejne wykonania różnić od siebie. Rozważmy przykłady podane na rysunku 60. Instrukcjami wykonywanymi wielokrotnie są wszystkie instrukcje zawarte pomiędzy

instrukcją pętli a instrukcją, której etykieta podana została w instrukcji pętli (np. 1 CONTINUE, por. rys. 60 wiersz 32-3). Zmiennej, której nazwa występuje w instrukcji pętli (tzw. zmiennej kontrolnej pętli) wolno używać wewnątrz pętli (por. rys. 60 wiersz 33-6), ale nie wolno zmieniać jej wartości, zatem nie może ona wystąpić po lewej stronie w instrukcji podstawienia ani w instrukcji READ (instrukcja czytania, patrz dalej). Nie wolno także zmieniać wewnątrz pętli wartości M1, M2 i M3 (nazywanych niekiedy parametrami: początkowym, końcowym i kroku). Możliwe jest umieszczanie jednej pętli wewnątrz drugiej (por. rys. 62), z tym że nie wolno, aby pętle się przecinały, czyli pętla wewnętrzna musi się kończyć wcześniej niż zewnętrzna. Przy konstruowaniu pętli istnieją także inne ograniczenia, związane z możliwością wykonywania skoków. Wolno wykonywać skoki między dwiema dowolnymi instrukcjami pętli, wolno wyskakiwać z pętli na zewnątrz (rys. 62), nie wolno natomiast wskakiwać z zewnątrz do wnętrza pętli. Odnośnie do ostatniej instrukcji, zawartej wewnątrz pętli (tej instrukcji, której etykieta figuruje w instrukcji pętli), nie może nią być żadna instrukcja typu startującego, a więc: IF, GO TO (we wszystkich postaciach), STOP, RETURN, PAUSE, a także inna instrukcja pętli. W przypadku kiedy ograniczenie to utrudnia budowę pętli o zamierzonym działaniu można zawsze posłużyć się, jako ostatnią instrukcją pętli, instrukcją CONTINUE, która niczego nie zmienia w programie, a może być zaopatrzona etykietą.

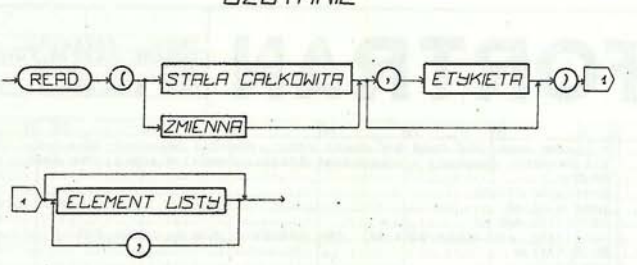
Dotychczas poznawaliśmy instrukcje pozwalające na dokonywanie obliczeń na wartościach już znajdujących się w pamięci maszyny cyfrowej. Obecnie omówimy grupę instrukcji wejścia - wyjścia, czyli instrukcji służących do wczytywania danych do programu oraz do wydruku wyników obliczeń. Instrukcje te występowały już wcześniej w przykładach, chociaż tylko w swojej najprostszej postaci. Pełną postać instrukcji czytania podano na rysunku 63, a pojęcie elementu listy, użyte do definicji instrukcji czytania, wyjaśniono na rysunku 64. Na rysunku 65 przytoczono reprezentatywne przykłady instrukcji czytania i pisania. Do wyjaśnienia poszczególnych elementów instrukcji rozważmy przykład podany w wierszach 38-2 i 38-3 na rysunku 65. Jak wynika ze składni podanej na rysunku 63, po słowie READ musi wystąpić nawias, w którym w najprostszym przypadku wymieniona jest najpierw stała lub zmienna (całko-


FORTRAN
PROGRAM PRZYKŁAD
NAZWISKO TADEUSIEWICZ
ARKUSZ 13 ARKUSZY
DATA 1.04.1978

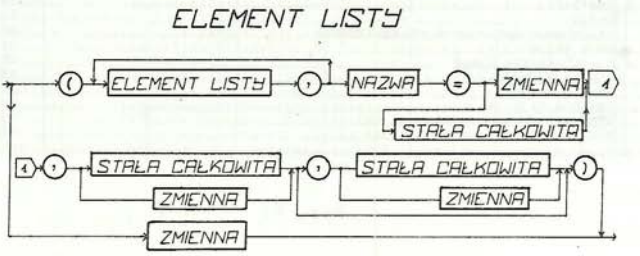
Rys. 61


FORTRAN
PROGRAM PRZYKŁAD
NAZWISKO TADEUSIEWICZ
ARKUSZ 14 ARKUSZY
DATA 1.04.1978

Rys. 62



Rys. 63



Rys. 64

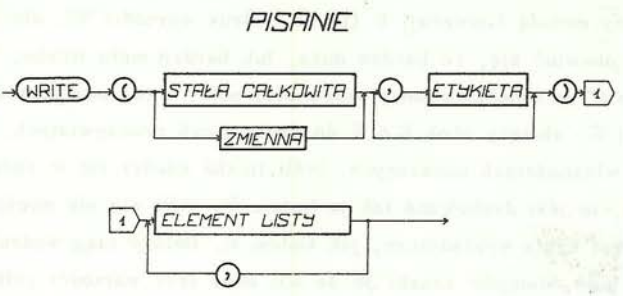
Rys. 65

wita!), a potem etykieta. Stała określa numer urządzenia, z którego będzie dokonywane czytanie, gdyż na ogół przy jednej maszynie cyfrowej jest duży zespół urządzeń czytających: czytnik kart (najczęściej używany), czytnik tasiemki papierowej, taśmy magnetyczne, dyski itp. O tym jaki numer odpowiada jakiemu urządzeniu decyduje sam programista, deklarując stosowne przyporządkowania w kartach sterujących programu (patrz dalej). Chwilowo załóżmy dla wygody, że numer 1 oznacza czytnik kart, a numer 2 - drukarkę wierszową. Etykieta występująca po numerze urządzenia wskazuje dyrektywę FORMAT, opisującą sposób wczytania danych.

Dyrektywa FORMAT została już wstępnie omówiona, obecnie przeanalizujemy związek poszczególnych kodów konwersji w dyrektywie FORMAT z listą zmiennych, których wartości są wczytywane, oraz z postacią wczytywanych danych (rys. 65 wiersze 38-1 do 38-6). W liście zmiennych, których wartości mają być czytane, figurują dwie zmienne rzeczywiste: X i Y. Odpowiada im kod konwersji F10.4 porzedzony cyfrą 2, co oznacza dwukrotne powtórzenie tego samego kodu. Kod F sygnalizuje, że chodzi o konwersje liczby rzeczywistej, cyfra 10 określa szerokość pola w znakach, które zajmuje liczba; innymi słowy na karcie dziurkowanej, z której czytane będą dane znaki od 1 do 10 włącznie, przeznaczone są na zapis wartości X, a znaki od 11 do 20 włącznie - na zapis wartości Y. Cyfra 4 występująca po kropce dziesiętnej oznacza liczbę cyfr po kropce dziesiętnej (liczba miejsc ułamkowych). Istotnie, zauważmy że zapis 1234567 podany w wierszu 38-5 w polu przeznaczonym na zapis wartości dla zmiennej X zinterpretowany został jako liczba 123.4567. Kropka była więc niepotrzebna w zapisie na karcie, gdyż maszyna sama zinterpretowała dane i ustaliła położenie kropki dziesiętnej. Zauważmy jednak, że w polu przeznaczonym dla liczby Y wpisano kropkę, i to niezgodnie z deklaracją wynikłą z FORMATU. Liczba wczytana została zatem tak, jak ją podano, czyli jako 8.2, a nie jako 0.0802, zgodnie z interpretacją liczby, gdyby nie było kropki. (Uwaga: wszystkie nie zapisane znaki wewnątrz pola na karcie dziurkowanej, przydzielonego FORMATEM określonej liczbie traktowane są jako zera!). Analizujemy dalej instrukcję czytania i przypisany jej FORMAT. W liście zmiennych do czytania są teraz kolejno trzy zmienne całkowite: I, J, K, a w FOR-

MACIE jest opis kodu konwersji I2 z oznaczeniem trzykrotnego powtórzenia. Tak więc zadecydowano, że wartość zmiennej I powinna zajmować znaki 21 i 22, wartość zmiennej J znaki 23 i 24, a wartość zmiennej K znaki 25 i 26 na tej samej karcie co wartości dla X i Y. Analizując wiersze 38-5 i 38-6 należy zauważyć, że wartość dla zmiennej K wpisana została jako 25 znak, zatem znak 26 zinterpretowano jako 0 i wczytana wartość zmiennej K wynosi 30. Na marginesie opisanego przykładu warto rozważyć własności innych możliwości zawartych w dyrektywie FORMAT, odnośnie do czytania liczb całkowitych i rzeczywistych. Zauważmy, że w kodzie I nie występuje kropka i liczba po niej (por. kod F), gdyż w liczbie całkowitej z definicji nie ma części ułamkowej. Pamiętamy, że kod konwersji może być zaopatrzony we współczynnik skali zawierający literę P (por. rys. 37). Liczba poprzedzająca P jest wykładnikiem zmiany skali. Przykładowo, gdyby czytanie wartości zmiennej X odbywało się nie formatem F10.4, ale formatem 3PF10.4, to wówczas wczytana wartość byłaby zapamiętana jako 0.1234567, a nie jako 123.4567, podobnie gdyby liczbę Y czytano formatem -4PF10.4, to zapamiętana wartość wynosiłaby 82000.0, a nie 8.2, jak przy kodzie F10.4. Jak pamiętamy do konwersji liczb rzeczywistych mogą służyć, poza kodem F, kody E oraz G, przy czym w przypadku czytania jest najzupełniej obojętne, jaki z nich zostanie użyty, gdyż wszystkie działają jednakowo. Bardziej złożony przykład przedstawiono na rysunku 65 w wierszach 39-1 do 38-8. Zaprezentowano tam użycie elementu listy w postaci cyklu. Jest to jak gdyby połączenie instrukcji czytania i instrukcji pętli. Zapis (X(I), I = 1,3) jest równoważny zapisowi X(1), X(2), X(3), podobnie należy interpretować wszystkie przykłady podane na rysunku 65.

Instrukcje czytania pozwalają wprowadzić informację do maszyny cyfrowej, natomiast instrukcje pisania powodują wydruk wyników. Struktura instrukcji pisania omówiona jest na rysunku 66, a na rysunku 65 w wierszach 40-1 do 40-5 podano przykład wykorzystania instrukcji pisania oraz postać wydruku uzyskiwanego przy użyciu określonego FORMATU. W przypadku druku FORMAT odgrywa bardzo ważną rolę, gdyż za jego pomocą można zredagować postać tabulogramu uzyskanego z maszyny, zaopatrzyć wyniki w wydruki tekstowe, objaśniające znaczenie podanych liczb, a także można ustalić dokładność podawanych danych na poziomie



Rys. 66

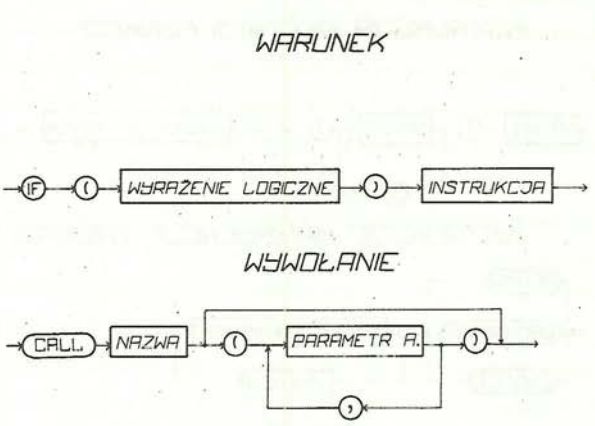
umożliwiającym natychmiastową interpretację. Przeanalizujemy przykład wspomniany wyżej. Drukowane są dwie liczby rzeczywiste X i Y oraz trzy liczby całkowite I, J, K. W wierszu 40-4 podano założenia odnośnie do wartości tych liczb. Przypatrzmy się wydrukowi (przytoczony poniżej wiersza 40-5) i dyrektywie FORMAT. W dyrektywie na początku jest kod 5X. Jest to nakaz wydrukowania pięciu odstępów. Na wydruku pojawiają się jednak jedynie 4 odstępy (znaki od 1 do 4), gdyż pierwszy znak wysyłany w nowej linii na drukarkę steruje wysuwem papieru i nie jest drukowany. Trzeba o tym bezwzględnie pamiętać. Znak spacji (odstępu) powoduje druk od nowej linii, znak + oznacza druk w tej samej linii (nabijanie jednego wydruku na poprzedni), znak 1 powoduje wysuw papieru do początku nowej stronicy. Inne znaki mogą mieć różne znaczenie, zależnie od konkretnej maszyny, należy bezwzględnie wystrzegać się ich stosowania, gdyż można bezproduktywnie zniszczyć sporo kosztownego papieru w drukarce. Typowo zatem FORMAT do druku powinien zaczynać się kodem nX, gdzie n musi wynosić przynajmniej 1.

Rozpatrując dalej przykładowy wydruk widzimy w FORMACIE kod konwersji H (stałą tekstową), której odpowiada tekst X= umieszczony na wydruku jako 5 i 6 jego znak. Następnie dziesięć znaków (od 7 do 16 włącznie) przeznaczonych jest na wydruk wartości zmiennej X kodem konwersji F10.4, dalej kod 3X nakazuje odmierzenie trzech odstępów (znaki 17, 18 i 19) oraz druk tekstu YGREK =. Zwróćmy uwagę na wydruk wartości zmiennej Y, dokonywany według konwersji E10.4. Widać, że liczba została wydrukowana w postaci wykładniczej z użyciem litery E do oznaczenie potęgi dziesiątki. Jest to postać mniej czytelna niż wydruk

uzyskiwany metodą konwersji F (patrz wydruk wartości X), ale nie ma potrzeby obawiać się, że bardzo duża, lub bardzo mała liczba, "nie zmieści się" w zadeklarowanej szerokości pola. Nawiasem mówiąc, kod konwersji G, służący obok E i F do druku liczb rzeczywistych, jest kodem o własnościach mieszanych. Jeśli liczba mieści się w zadeklarowanym polu, to jest drukowana tak ja kodem F, jeśli się nie mieści, stosowany jest zapis wykładniczy, jak kodem E. Dalszy ciąg wydruku to kolejno: pięć odstępów (znaki 36 do 40) oraz trzy wartości całkowite wydrukowane w ten sposób, że dla każdej z nich przeznaczono 4 pola. Naturalnie liczby będące aktualnymi wartościami I, J, K nie wypełniały pola do końca i dlatego pojawiły się na wydruku dodatkowe spacje.

Wydruk uzyskiwany instrukcją pisania, odwołującą się do etykiety dyrektywy FORMAT, ma postać "zredagowaną", podobnie czytanie z wykorzystaniem FORMATU musi odbywać się z nośnika, na którym dane rozmieszczono ściśle według sposobu opisanego w FORMACIE. Natomiast często zachodzi potrzeba zapisania czy odczytania danych w sposób bezformatowy, lub jak się często mówi, niezredagowany. Ma to miejsce szczególnie wtedy, gdy nadawcą i odbiorcą informacji jest ta sama lub inna maszyna cyfrowa, a zwłaszcza w przypadku, kiedy jest użyty zapis i odczyt na taśmie magnetycznej. Stosuje się wówczas instrukcje pisania i czytania, podając jedynie numer urządzenia, na którym piszemy lub czytamy (rys. 65, wiersze 41-1 i 41-2). Taka forma zapisu jest godna zalecenia, gdyż zapis oraz odczyt w tym przypadku dokonywane są w systemie dwójkowym i maszyna nie traci czasu na konwersję kodów. Oszczędności czasu uzyskiwane w ten sposób przy używaniu taśmy magnetycznej mogą sięgać 90%.

Instrukcja warunkowa przedstawiona na rysunku 67 działa w bardzo specyficzny sposób: na początku wyliczane jest wyrażenie logiczne w nawiasie. Jeśli jego wartość jest .FALSE., to działanie instrukcji na tym się kończy. Jeśli natomiast wyrażenie logiczne ma wartość .TRUE., to wówczas wykonywana jest instrukcja, która jest po nawiasie zamykającym wyrażenie logiczne. Instrukcją tą może być dowolna instrukcja języka FORTRAN, dzięki czemu możliwe jest warunkowe wykonywanie dowolnych czynności. Przykłady i komentarze podane na rysunku 68 w wierszach



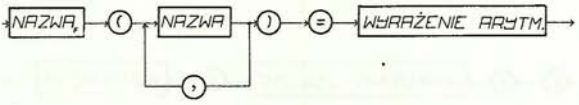
Rys. 67

Rys. 68

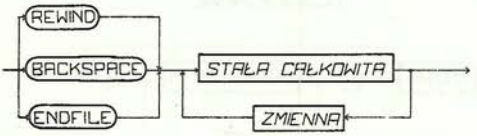
42-1 do 42-15 pozwalają zorientować się w niektórych możliwościach, jakie stwarza instrukcja warunkowa.

Wywołanie podprogramu (rys. 67 oraz rys. 68, wiersze 49-1 do 49-9) pozwala na przeniesienie akcji maszyny cyfrowej z jednego segmentu do innego, wymienionego jako nazwa po słowie CALL segmentu typu

INSTRUKCJA DEFINICJI FUNKCJI



INSTRUKCJE MANIPULACJI ZBIORAMI

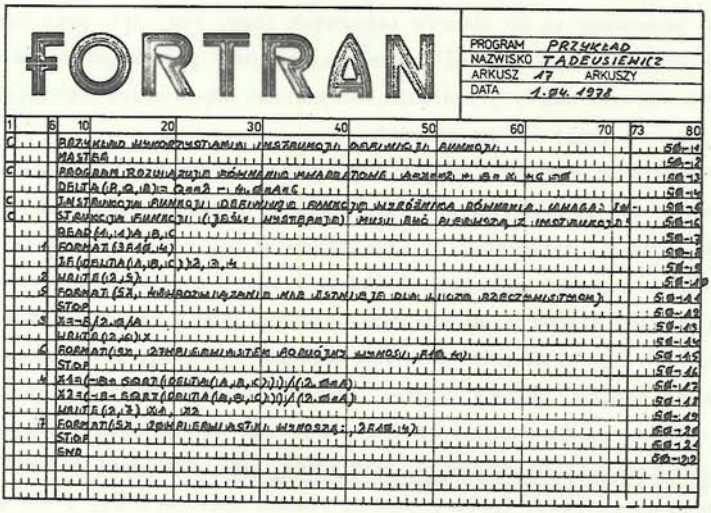


Rys. 69

SUBROUTINE. Przy wywołaniu następuje przekazanie informacji z segmentu wołającego do segmentu wywoływanego za pośrednictwem znanego nam już mechanizmu utożsamiania parametrów a. instrukcji wywołania z parametrami f. nagłówka segmentu podprogramu. Maszyna wykonuje kolejne instrukcje podprogramu aż do napotkania instrukcji RETURN, która powoduje powrót do segmentu wołającego, a konkretnie powrót do wykonywania instrukcji, następującej w segmencie wołającym bezpośrednio po instrukcji wywołania.

Instrukcja definicji funkcji (rys. 69, rys. 70) daje możliwość definiowania przez programistę własnych funkcji w sposób analogiczny do znanego już mechanizmu segmentu funkcji, ale bez konieczności tworzenia osobnego segmentu i przy znacznym uproszczeniu zapisu. Instrukcja definicji funkcji musi poprzedzać wszystkie inne instrukcje danego segmentu i może definiować funkcję, która daje się opisać wzorem, zajmującym pojedynczą instrukcję podstawienia (por. rys. 70). Funkcji tak zdefiniowanej można używać jedynie w segmencie, w którym znalazła się instrukcja definiująca. Niemniej jest to bardzo wygodna metoda definiowania własnych funkcji, warta częstszego stosowania niż to się obecnie obserwuje.

Instrukcje manipulacji zbiorami pozwalają przewijać taśmę magnetyczną lub zbiór dyskowy do początku danych (REWIND), wycofywać zapis o jeden rekord (BACKSPACE) i zakańczać zapis na taśmie standardowym




```
C     PRZYKŁAD WYKORZYSTANIA INSTRUKCJI DEFINICJI FUNKCJI
      MASTER
C     PROGRAM ROZWIĄZUJE RÓWNANIE KWADRATOWE I OBLICZA JE
C     INSTRUKCJA FUNKCJI DEFINIUJE FUNKCJĘ WYRÓŻNIKA RÓWNANIA
C     STAŁE FUNKCJI (JEDEN WYSTĘPUJE) MUSZĄ BYĆ PRZEDSTAWIONE
      READ(4,1)A,B,C
    1 FORMAT(3F8.4)
      IF(DELTA(A,B,C))2,3,4
    2 WRITE(2,5)
    5 FORMAT(52,'WYRÓŻNIK JEST UJEMNY')
      STOP
    3 X=-B/(2.*A)
      WRITE(2,6)X
    6 FORMAT(52,'DWIE IDENTYCZNE RZECZYWISTE')
      STOP
    4 X1=(-B+SQRT(DELTA(A,B,C)))/(2.*A)
      X2=(-B-SQRT(DELTA(A,B,C)))/(2.*A)
      WRITE(2,7)X1,X2
    7 FORMAT(52,'DWIE RZECZYWISTE:',2F8.4)
      STOP
      END
```

Rys. 70

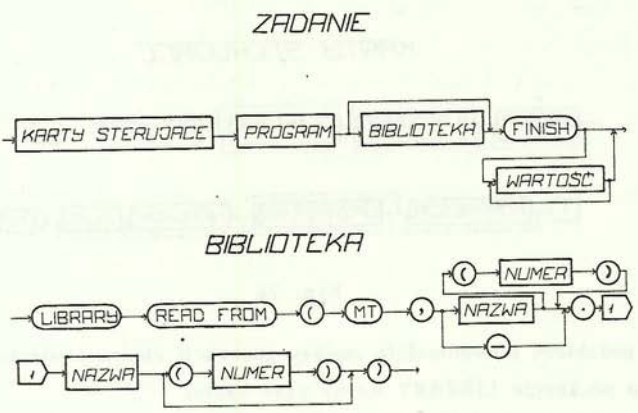

```
C     PRZYKŁADY UŻYCIA INSTRUKCJI MANIPULACJI ZBIORAMI
      MASTER
C     PROGRAM PRZEPISUJE DANE Z MAŁEJ NA TAŚMĘ MAGNETYCZNĄ I DA
      DIMENSION X(40)
    2 READ(4,4)X
    4 FORMAT(40F2.0)
      IF(X(1).LT.0)GOTO 3
C     UJEMNA WARTOŚĆ X(1) SYGNALIZUJE KONIEC DANYCH
      WRITO(3)X
      GOTO 2
    3 BACKSPACE 3
      ENDFILE 3
      REWIND 3
    4 READ(3)X
      WRITE(2,4)X
      GOTO 4
C     PRZEPISYWANIA CZYTANIA NASTĄPI AUTOMATYCZNIE PO ZNALEZIENIU OGRANICZNIKA
      END
```

Rys. 71

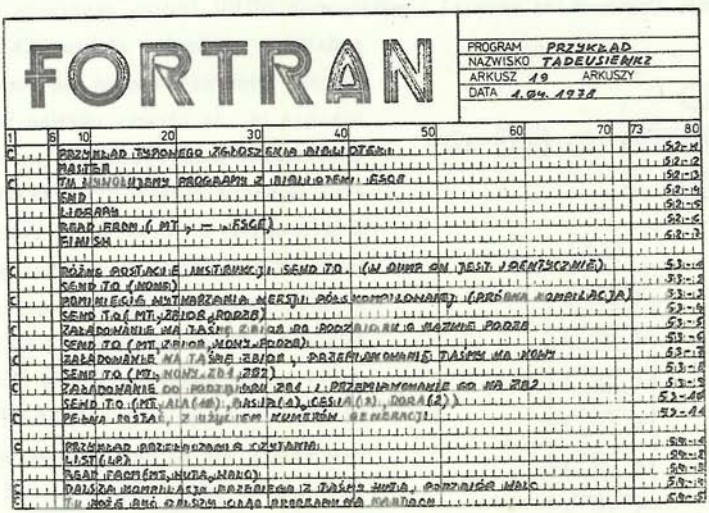
oznacznikiem końca zbioru danych (ENDFILE). Z natury rzeczy instrukcje te stosowane są do zbiorów taśmowych (por. rys. 71), przy czym stała całkowita (lub wartość zmiennej całkowitej) podana po słowie kluczowym określa numer urządzenia, do którego odpowiednia manipulacja (np. przewinięcie) ma się odnosić.

Omawianie elementów języka FORTRAN, zamykamy przedstawieniem instrukcji manipulacji zbiorami służących do programowania. Elementy te (instrukcje i dyrektywy) są uniwersalne w tym sensie, że niezależnie od rodzaju maszyny cyfrowej, na której program ma być wykonywany, mogą one być pisane w podany wyżej sposób. Jednak sam program to za mało, aby w pełni określić działanie maszyny cyfrowej, gdyż całość zadania ma bardziej złożoną budowę. Niestety jednak budowa ta bywa różna, zależnie od typu użytej maszyny, i dlatego dalsze uwagi odnoszą się wyłącznie do maszyn cyfrowych serii Odra 1300. Strukturę zadania dla tej maszyny cyfrowej (przy założeniu obsługi operatorskiej, a nie w warunkach pracy systemu GEORGE) przedstawia rysunek 72. Jak widać, przed programem w strukturze zadania muszą występować karty sterujące, po programie może występować biblioteka (nie koniecznie), potem karta FINISH oraz ewentualnie ciąg kart z wydziurkowanymi wartościami, stanowiącymi dane, czytane instrukcjami READ w programie. Zgłoszenie biblioteki (rys. 72) następuje wtedy, gdy chcemy posłużyć się podprogramem lub funkcją należącą do takiej biblioteki. Zawartości bibliotek są bardzo bogate i można prawie jak pewnik przyjmować, że dla większości typowych zadań obliczeniowych (rozwiązywanie układów równań, całkowanie numeryczne itd.) są w bibliotekach gotowe programy, wystarczy jedynie napisać segment MASTER i odpowiednie CALL. Z tego względu warto zapoznać się ze spisem programów biblioteki używanej w swoim ośrodku obliczeniowym, gdyż można zaoszczędzić sobie sporo pracy. Biblioteką, która w maszynach serii Odra 1300 zawiera bardzo wiele wartościowych podprogramów jest FSCE, nazywana także biblioteką naukową. Przykład zgłoszenia tej właśnie biblioteki podano na rysunku 73 w wierszach 52-1 do 52-7.

Warto zająć się jeszcze składnią opisu taśmy magnetycznej, podaną na rysunku 72 po literach MT. Aby ją dobrze zinterpretować, należy poznać zarys organizacji zbiorów taśmowych w maszynach serii Odra



Rys. 72



Rys. 73

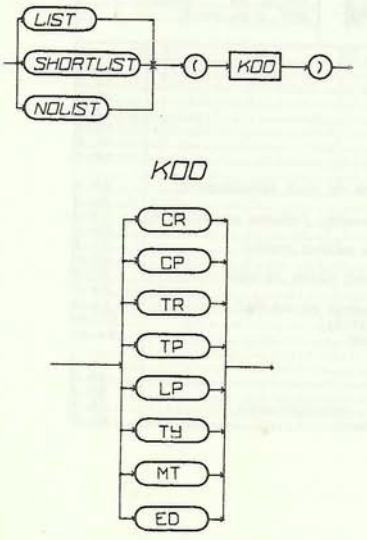
1300. Całość zapisu na taśmie magnetycznej nosi nazwę zbioru i może zawierać jeden lub więcej podzbiorów. Każdy zbiór może mieć poza nazwą także numer generacji, odróżniający od siebie ewentualne kolejne wersje tego zbioru. Podobnie z podzbiorami. Odwołując się do taśmy magnetycznej musimy zatem poza oznaczeniem MT podać: nazwę zbioru,

Rys. 74

nazwę podzbioru i ewentualnie numery generacji zbioru i podzbioru. Przykładowo po karcie LIBRARY można użyć karty:

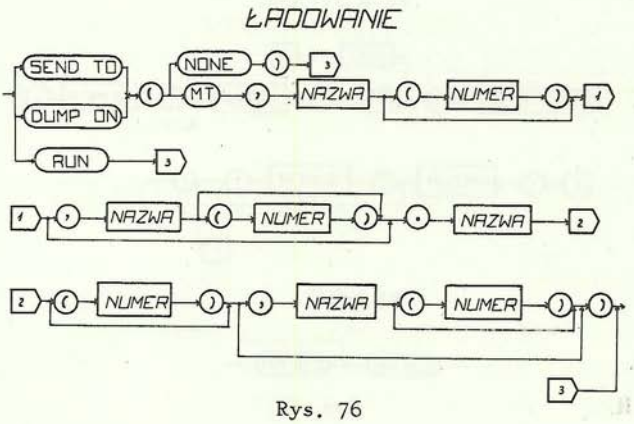
READ FROM (MT, YOGI (3).BUBU(2)).

Oznacza to, że chcemy posłużyć się zbiorem o nazwie YOGI (trzecią generacją zbioru o tej nazwie) i podzbiorem BUBU (druga generacja tego podzbioru w obrębie zbioru YOGI). Numery generacji często się pomija; oznacza to, że chcemy używać zbioru (podzbioru) o najwyższym z istniejących numerze generacji. W przykładzie podanym na rysunku 73 pominięto także nazwę zbioru, zastępując ją kreską. Można tak było postąpić, gdyż biblioteka FSCE znajduje się na tej samej taśmie magnetycznej (o nazwie FORT), co kompilator FORTRANU, nazywający się XFAM. Kompilator zatem wie, na jakiej taśmie szukać biblioteki – na swojej własnej mianowicie. Naturalnie zapis READ FROM(MT, FORT.FSCE) byłby też poprawny, ale czas trwania kompilacji wyraźnie wydłużyłby się.



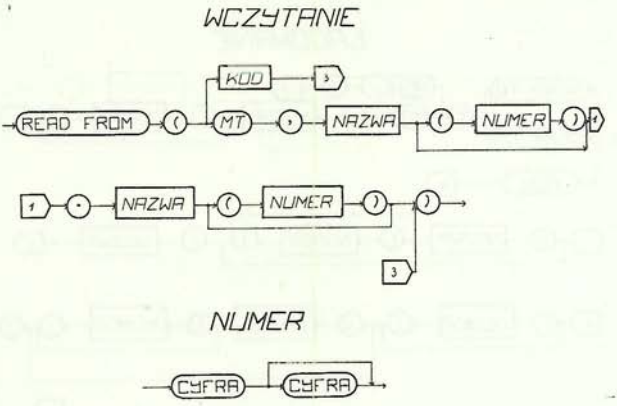
Rys. 75

Omówimy teraz karty sterujące. Jak wynika z rysunku 74, segment kart sterujących zawiera kilka grup kart, z których niektóre mogą być pomijane, inne mogą występować kilkakrotnie, a jedynie grupa nazwana



Rys. 76

DEFINICJA oraz karta END muszą występować zawsze. Pierwszy zespół kart sterujących, nazwany wydruk (rys. 75), określa zakres i miejsce wydruku programu podczas kompilacji. W przypadku podania słowa LIST przedrukowywany jest pełny tekst programu, co ułatwia ewentualne śledzenie błędów. SHORTLIST powoduje drukowanie jedynie nagłówków segmentów, a NOLIST - brak jakiegokolwiek wydruku. Kod następujący po słowie kluczowym określa urządzenie, na którym wydruk jest dokonywany. Najczęściej urządzeniem tym jest LP, ale na rysunku 75 podano wszystkie możliwe kody. Ich znaczenie jest następujące: CR - czytnik kart, CP - perforator kart, TR - czytnik tasiemki perferowanej, TP - perforator taśmy papierowej, LP - drukarka wierszowa, TY - monitor operatora, MT - taśma magnetyczna, ED - jednostka dysków magnetycznych. Kolejny zespół kart dotyczy ładowania (rys. 76). Program po skomplikowaniu może być od razu wykonywany, ale można go załadować na taśmę magnetyczną do późniejszego wykorzystania. Jeżeli program jest już gotowy do natychmiastowego późniejszego wykonywania, to warto go załadować na taśmę w postaci binarnej skonsolidowanej. Służy do tego wiersz DUMP ON. Jeżeli jednak program ma charakter zbioru luźnych segmentów, z których później będziemy zestawiać różne programy zależnie od potrzeb (tworzenie własnej biblioteki podprogramów), to wówczas należy posługiwać się wierszem SEND TO, który ładuje na taśmę program nieskonsolidowany. W wierszu wymieniana jest taśma, na którą na-

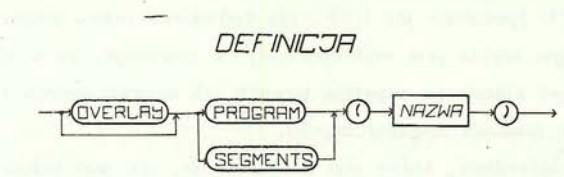


Rys. 77

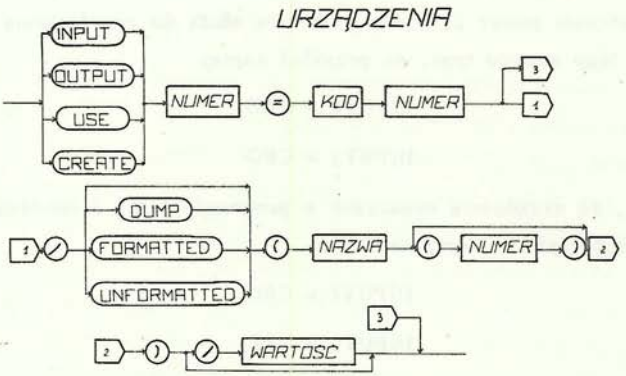
leży załadować program, ewentualna nazwa, na jaką należy taśmę przemianować, nazwa podzbioru, do którego nowy program ma wejść i ewentualna nowa nazwa podzbioru. Wszystkie te elementy mogą oczywiście dodatkowo mieć podane numery generacji. Jak widać, możliwości jest tu sporo, ważniejsze z nich pokazano na przykładzie z rysunku 73. Jeśli użyliśmy któregokolwiek z wierszy SEND TO albo DUMP ON, to program nie będzie wykonywany, chyba że dodatkowo podamy wiersz RUN.

Wiersze strujące wczytaniem (rys. 77) pozwalają na kompilację programu z różnych nośników. Podstawowym nośnikiem są zwykle karty dziurkowane, ale stosując wiersz READ FROM można nakazać kompilację z tasiemki papierowej, taśmy magnetycznej i innych nośników (por. rys. 73, wiersze 54-1 do 54-5). Kompilacja z taśmy magnetycznej może zachodzić w przypadku, kiedy na taśmie zapisano treść programu w FORTRANIE specjalnym programem o nazwie XKYA. Omawianie edytora XKYA wykracza poza ramy niniejszego opracowania, warto jednak zwrócić na niego uwagę, gdyż pozwala on na wygodne poprawianie dużych programów bez konieczności wielokrotnego wczytywania dużej liczby kart. Obowiązkowo występujące w każdym zadaniu karty definicji określają, czy mamy do czynienia z programem czy ze zbiorem luźnych segmentów, nadają mu nazwę i ewentualnie deklarują, że segmenty (program) mają być nakładkowe (OVERLAY). Najczęstsza postać karty definicji to

PROGRAM (nazwa),



Rys. 78



Rys. 79

pozostałe warianty używane są wyjątkowo. Nieczęsto także używa się kart kompresji, które powodują, że zmienne typu całkowitego i logicznego względnie podwójnej precyzji zajmują mniej miejsca w pamięci. Postaci wszystkich tych kart są bardzo proste, i rysunek 78 wyjaśnia je całkowicie. Karty urządzeń (rys. 79) deklarują urządzenia zewnętrzne, używane przez maszynę w czasie wykonywania programu, podają ich numery programowe (numery, które były używane w programie w instrukcjach READ, WRITE, REWIND, BACKSPACE itp.) i określają tryb ich użycia. Urządzenie może być zadeklarowane jako INPUT (służy tylko do czytania), OUTPUT (służy tylko do pisania), USE (służy zarówno do czytania, jak i do pisania, używane w przypadku taśm magnetycznych)

oraz CREATE (podobnie jak USE, ale dodatkowo taśma magnetyczna używana w tym trybie jest etykietowana, co powoduje, że w tym trybie można tworzyć zbiory na zupełnie nowych lub oczyszczonych z poprzednich zapisów taśmach magnetycznych).

Jeśli urządzeniem, które jest deklarowane, nie jest taśma magnetyczna, obowiązuje skrócony format karty (zgodnie ze strzałką 3 na rysunku 79). Przykładami takich deklaracji są występujące niemal w każdym programie

INPUT1 = CRO,

OUTPUT2 = LPO,

definiujące czytnik kart jako urządzenie numer 1 i drukarkę wierszową jako urządzenie numer 2. Cyfry po kodzie służą do rozróżnienia kilku urządzeń tego samego typu, na przykład zapisy

INPUT1 = CRO,

INPUT3 = CRO

deklarują, że urządzenia oznaczane w programie 1 lub 3 odnoszą się do tego samego czytnika kart, natomiast

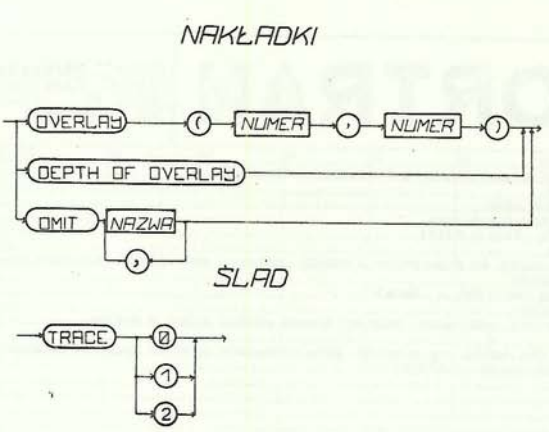
INPUT1 = CRO,

INPUT5 = CR1

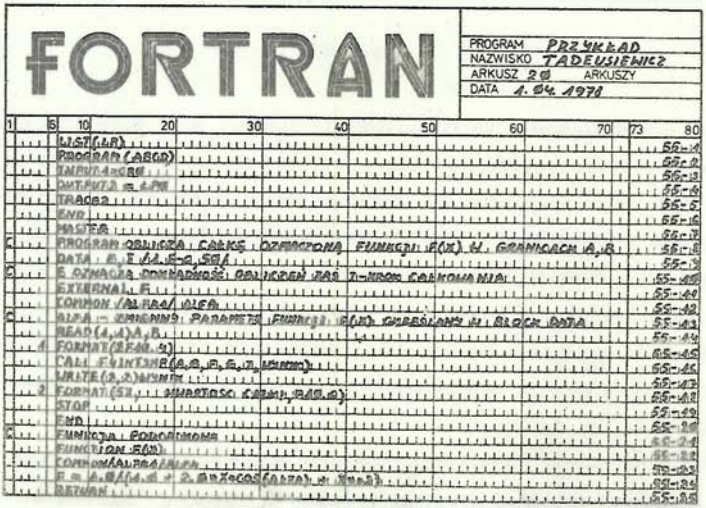
deklarują użycie dwu różnych czytników. W przypadku używania taśmy magnetycznej (kod = MT) konieczne jest uzupełnienie definicji elementami podanymi na rysunku 79 za strzałką 1. FORMATTED oznacza, że taśma służyć będzie do pisania i czytania zredagowanego (z wykorzystaniem dyrektywy FORMAT), UNFORMATTED lub brak tego słowa oznacza zapis niezredagowany, DUMP deklaruje taśmę do zapisu samego programu w razie jego przerwania. Nazwa i numer określają nazwę taśmy i numer jej generacji, a wartość określa sposób zapisu (długość bloków na taśmie). Typowy zapis użycia taśmy ma postać

CREATE7 = MT1/UNFORMATTED(DANE)

lub w postaci równoważnej



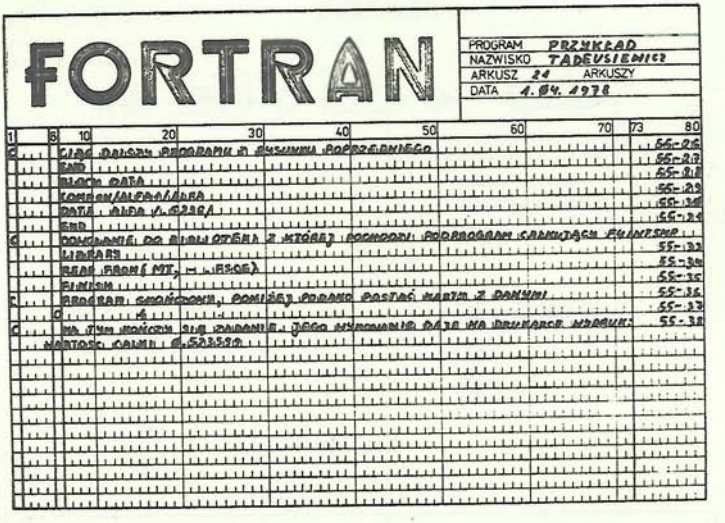
Rys. 80



Rys. 81

CREATE7 = MT1/(DANE).

Na rysunku 80 podano składnię użycia kart nakładek, ich ewentualne użycie wymaga jednak zapoznania się z obszerniejszym opisem nakładkowania programu, co wykracza poza ramy niniejszego opracowania. Warto na-



Rys. 82

tomiast umieć stosować karty sterowania śladem, gdyż pozwalają one łatwo lokalizować ewentualne błędy. Jeśli umieścimy w programie kartę TRACE 2, to w przypadku ewentualnego błędu dostaniemy bardzo obszerny wydruk informacyjny ("ślad"), który umożliwi łatwe znalezienie i poprawę błędu. Niestety jednak za udogodnienie to trzeba płacić, gdyż program kompilowany z TRACE 2 liczy się kilkakrotnie dłużej (maszyna wykonuje wiele czynności związanych z tworzeniem "śladu"), dlatego jeśli mamy program pewny, należy pomijać kartę TRACE lub podawać TRACE 0.

Na zakończenie niniejszego rozdziału przedstawiamy na rysunku 81 i 82 kompletny przykład prostego zadania dla maszyny Odra 1304, pozwalający zorientować się w praktycznym stosowaniu przytoczonych tu zasad.