AKADEMIA EKONOMICZNA W KRAKOWIE

PODSTAWY UŻYTKOWANIA SYSTEMU MINIKOMPUTEROWEGO MERA-400

Praca zbiorowa pod redakcją
Jerzego Kolendowskiego

KRAKÓW 1988

AKADEMIA EKONOMICZNA W KRAKOWIE

PODSTAWY UŻYTKOWANIA SYSTEMU MINIKOMPUTEROWEGO MERA-400

Praca zbiorowa pod redakcją
Jerzego Kolendowskiego

KRAKÓW 1988

Poszczególne rozdziały opracowali:

Jerzy Kolendowski - 1
Ryszard Tadeusiewicz - 6
Marian Kuraś - 3
Tadeusz Wilusz - 4
Jacek Wołoszyn - 5
Jan Dudek - 2 i 7

RECENZENT

Stanisław Białas

Wydano za zgodą Rektora
Akademii Ekonomicznej w Krakowie

Skrypt przeznaczony jest dla studentów
IV roku studiów dziennych kierunku
cybernetyki ekonomicznej i informatyki

Powielono z dostarczonych oryginałów
w Drukarni Uniwersytetu Jagiellońskiego
w Krakowie, ul. Manifestu Lipcowego 13

Wydanie I. Nakład 250 + 22 egz.
Zam. 320/88
Oddano do produkcji w czerwcu 1988 r.
Produkcję ukończono we wrześniu 1988 r.

Ark. wyd. 14,8
Ark. druk. 19 2/16
D-13-1484

Cena zł 296,-

Spis treści

1. WSTĘP

1.1. Aktualny stan minikomputeryzacji

W działalności wielu instytucji Mera-400 oddawała i oddaje ogromne usługi. Stąd też zamysł przygotowania skryptu pozwalającego zapoznać się z elementarnymi właściwościami budowy komputera, z jego systemem operacyjnym, oprogramowaniem użytkowym a w tym z dwoma językami programowania: FORTRAN oraz BASIC.

W dzisiejszym świecie oszałamiającego rozwoju elektroniki (często teraz mówimy - mikroelektroniki) wyrobienie sobie właściwego poglądu na sprzęt informatyczny nie jest sprawą łatwą. Błędy popełniają i specjaliści. Świadczą o tym między innymi mylne decyzje przy zakupach. Na przykład obecnie zdarza się, że instytucje zakupują w nadmiarze mikrokomputery stawiając im pochopnie wymagania przerastające ich możliwości. Bywają też chybione decyzje producentów sprzętu.

Podejmowanie mylnych decyzji naraża kupujących na pewne straty. Skutki błędów popełnianych przez producentów są dla nich dotkliwsze. Ale z drugiej strony trafne decyzje powodują spektakularny rozwój tych co produkują sprzęt komputerowy.

Niektórzy producenci po wpadnięciu w kłopoty są ratowani przez subwencje rządowe. Wynikają one ze świadomości znaczenia własnej produkcji komputerów dla utrzymywania należytej koniunktury kraju, dla podnoszenia ogólnego poziomu technologii i narodowej cywilizacji.

Tak uratowana została angielska firma ICL (International Computer Limited). Znana jest ona u nas - wszak na jej licencji ELWRO produkowało swoje bardzo dobre i cenione ODRY.

Wśród trudności naszego kryzysu nie ostała się produkcja Mery-400. Projektowana z myślą o wieloletnim rozwoju, posiadające elastyczną budowę modułową stwarzająca możliwości dołączania przyśpieszających jej działania matematyczne dodatkowych procesorów - nie przetrwała kryzysu mimo swych zalet.

Ma ona też i wady, które w latach jej projektowania nie były tak oczywiste jak obecnie. Wówczas panowała inna technologia, inne trendy, a minikomputery nie osiągnęły tego miana jakie nadaje się im obecnie: supermini.

Minikomputery zaczęły się pojawiać na światowym rynku komputerowym około dwadzieścia lat temu. Osiągnięto już wtedy w technologii elektronicznej pewne postępy w miniaturyzacji.

W czasie poprzedzającym pojawienie się minikomputerów produkowana wyłącznie duże i kosztowne komputery. Na ich zakup nie mogły sobie pozwalać mniejsze instytucje mimo ich zainteresowania dla stosowania informatyki.

Wtedy pojawia się trafna koncepcja. W roku 1985 firma DEC (Digital Equipment Corporation) wprowadza na rynek pierwszy minikomputer PDP-8 (Programmed Data Processor). Miał on pamięć operacyjną o pojemności 4096 komórek, każda o długości 12 bitów. Był komputerem szybkim. Zaspokajał zapotrzebowanie wielu potencjalnych użytkowników, a kosztował przy tym poniżej 18 000 dolarów USA.

Odniósłszy sukces firma DEC pielęgnowała opanowany przez siebie kierunek. W roku 1970 pojawia się na rynku PDP-11 z wprowadzoną do architektury systemu szyną UNIBUS. Do szyny podłączone są wszystkie elementy systemu komputerowego:procesor, pamięć operacyjna, urządzenia wejścia-wyjścia.

Wkrótce pojawiają się na rynku następne systemy szeregu PDP. Z kolei postępując zgodnie z właściwą sobie linią rozwoju firma opracowała systemy serii VAX.

Wybrano dane dotyczące niektórych komputerów - głównie mini podano w tablicy 1.1.

Natomiast dane dotyczące szeregów minikomputerów produkcji CDC (Control Data Corporation), BASF, IBM-43 podano w tablicy 1.2.

Tablica 1.1

Wybrane dane niektórych komputerów

Producent typ		Mips	RP	PAO	PAN
BASF	7/38	1,0	45	1:8	32
	7/63	1,5	69	2:8	64
IBM	370/158-3	1,0	45	1:6	16
	4361-3	0,38	22	2:4	8
	4361-4	0,8	49	2:8	8
	4361-5	1,1	66	2:8	16
RIAD	1055	0,45	25	1:2	-
ICL	29-37	0,14	11	0,5:2	-
	29-45	0,2	14	0,5:2	-

Mips - miliony operacji na sekundę,
RP - względna wydajność Relative Performance,
PAO - pojemność pamięci operacyjnej w milionach bajtów,
PAN - pojemność buforowych pamięci notatnikowych w tysiącach (kilo) bajtów.
Źródło: Data World. Auerbach Publishers Inc. Vol. 7 and 8.

Minikomputery z tablicy 1.2 noszą też nazwę supermini. Należy rozumieć, że supermini oznaczają komputery budowane w zaawansowanej technologii LSI lub VLSI i zastępujące swe odpowiedniki ze starszych generacji. Różnią się one od poprzednich znacznie mniejszymi gabarytami, ciężarem i poborem mocy. Równocześnie ich wydajność mierzona w milionach operacji na sekundę Mips (Millions of instructions par second) odpowiada lub przewyższa wydajność dużych lub bardzo dużych komputerów drugiej (tranzystory) lub trzeciej (układy scalone) generacji.

Tablica 1.2

Podstawowe dane szeregów minikomputerowych (supermini) produkcji firm CDC, BASF, IBM

Szereg Cyber 180

Typ	810	830	830 Dual	835	845	855	855 Dual	990	990 Dual
Mips	1,0	1,6	2,9	3,8	8,5	12,5	22,5	32,3	58,1
RP	61	97	175	230	506	743	1338	1914	3408
PAO	2:16	2:16	2:16	4:16	4:16	4:16	4:16	8:32	8:32
PAN	-	-	-	16:32	16:32	16:32	16:32	32	32

Szereg BASF

Typ	7/63	7/65	7/68	7/73	7/75	7/78	7/88
Mips	1,5	1,8	2,2	4,5	6,2	8,5	11,0
RP	69	83	101	215	285	390	540
PAO	2:8	2:16	4:16	8:16	8:32	8:32	8:32
PAN	64	64	84	32	64	64	256

Szereg IBM-43

Typ	4321	4331-11	4331-2	4341-9	4341-10	4341-1	4341-11	4341-12
Mips	0,2	0,26	0,38	0,4	0,58	0,72	0,88	1,2
RP	11	18	22	24	34	40	50	76
PAO	1	1:4	1:4	1:4	2:4	2:4	2:8	2:16
PAN	-	8	8	4	4	8	8	16

Źródło: Katalogi firmowe.

Poza Mips (milion czyli mega instrukcji na sekundę), innym powszechnie obecnie stosowanym wskaźnikiem jest tzw. względna wydajność RP (Relative Performance). Ma ona dla szeroko znanego komputera firmy IBM 370/158-3 wartość 45.

Ten ostatni komputer 370/158 należy do komputerów starszej generacji, które około 10 lat temu zaliczano do komputerów dużych.

Pojemności pamięci operacyjnych PAO podawane są w milionach bajtów - czyli w mega bajtach (MB), natomiast pojemności buforowych pamięci notatnikowych (PAN) w tysiącach (kilo) bajtów (KB).

Poprzednio podkreślono, że wprowadzenie przez DEC komputerów serii PDP, a następnie VAX należy zaliczyć do trafnych koncepcji. Następną trafną koncepcją było utworzenie sieci komputerowej DECNET. W tym przypadku specjaliści firmy potrafili słusznie i wcześnie przewidzieć polaryzowanie się podstawowych problemów informatyki wokół:

- dużych systemów, a takimi są superkomputery (np. CRAY), zespoły supermini (np. VAX), układy minikomputerów oraz mikroprocesorów,
- sieci komputerowych.

Od roku 1975 usługi sieciowe stanowią ważny czynnik utrzymania dochodów DEC na wysokim poziomie mimo silnej konkurencji sieci IBM, Control Data, Norsk Data i innych. Wśród konkurencji IBM stosuję zachęcanie do korzystania z usług jednej ze sieci IBM zwanej BITNET polegające na możliwości posługiwania się zasobami BITNET bezpłatnie w ciągu pierwszego roku użytkowania.

Ostatecznie DEC - światowy producent minikomputerów, utrzymuje bardzo wysoką pozycję na liście dochodów. Jest na drugim miejscu tuż za światowym gigantem International Business Machines (tablica 1.3). DEC zatrudnia obecnie 90.000 pracowników, założył też bardzo aktywną organizację pod nazwą DECUS - Digital Equipment Computer User's Society.

Wybrane dane ze światowej listy producentów sprzętu informatycznego podaje tablica 1.3. Niektóre firmy produkują sprzęt informatyczny nie stanowiący ich głównego zakresu specjalności. Lecz udział tych firm w produkcji informatycznej ma znaczące miejsce w skali światowej (np. Fujitsu).

Tablica 1.3, z której tak wiele odczytać mogą znawcy informatyki zawiera następujące kolumny:

Tablica 1.3

Wybrane dane ze światowej listy producentów sprzętu informatycznego

Pozycja r.1984	Nazwa producenta	Doch.glob 1984 w tys. dol.	Doch.inf. 1984 w tys. dol.	Doch.inf. 1983 w tys. dol.	% doch. inform. 1983-84
1.	International Business Machines	45.937	44.292	36.503	21,3
2.	Digital Equipment Corp.	6.230	6.230	4.827	29,0
3.	Burroughs Corp.	4.876	4.500	4.000	12,5
4.	Control Data Corp.	5.027	3.756	3.508	7,0
5.	NCR Corp.	4.074	3.670	3.333	10,1
6.	Fujitsu Ltd.	6.440	3.499	2.800	24,96
14.	Apple Computer Corp.	1.898	1.898	1.085	74,9
18.	AT & T Corp.	33.200	1.340	1.080	23,96
20.	ICL	1.223	1.223	1.283	- 4,68
23.	Commodore International Ltd	1.1895	1.130	927	21,88
26.	N.V.Philips Gloelampen-fabricken	5.221	1.090	1.095	-0,46
32.	Texas Instruments Inc.	5.742	860	800	7,5
35.	General Motors Corp.	83.890	786	719	9,32
36.	Amdahl Corp.	779	779	778	0,22
40.	Prime Computer Inc.	643	643	517	24,4
43.	Motorola	5.534	618	514	20,2
47.	National Semiconductor	1.818	550	425	29,4

Źródło: Wg Datamation 100 - zamieszczone w "Cyberline" Vol. 7. Issne 8, August 1985, p. 3-3.

- miejsce w świecie w roku 1984 w zakresie produkcji informatycznej,
- nazwa firmy,
- globalne dochody w roku 1984,

- dochody z produkcji informatycznej w roku 1984,
- dochody z produkcji informatycznej w roku 1983,
- procent zmian dochodów z produkcji informatycznej 1983 - 1984.

Niesłabnące zapotrzebowanie na minikomputery skłania wielu producentów do pójścia w ślady DEC. Podobną produkcję podjęły między innymi ZSRR, Polska, Czechosłowacja, Węgry, próbowała Rumunia. Są to między innymi dobrze znane systemy SM-3, SM-4, SM-4a, SM-5, TPA. Indywidualnym polskim rozwiązaniem była Mera-400.

W okresie jej tworzenia istniały już zaawansowane rozwiązania hardware'owe, software'owe, sieciowe. Działo się tak mimo upływu bardzo krótkiego czasu od powstania epoki komputerowej. W epoce tej co kilka lat zmieniały się (i zmieniają się nadalę "ery" poszczególnych generacji rozwiązań informatycznych. Do lat pięćdziesiątych pierwsza generacja komputerów lampowych, do lat sześćdziesiątych druga generacja tranzystosowa, do lat siedemdziesiątych generacja układów scalonych: od IC - Integrated Circuits po LSI - Large Scale Integration - duża skala integracji. A do lat osiemdziesiątych generacja VLSI - Very Large Scale Integration - bardzo duża skala integracji.

W różnych erach generacji komputerowych pojawiły się i takie osiągnięcia jak opracowanie w 1963 roku pierwszego urządzenia o nazwie modem (MOdulator - DEModulator). Wtedy już cała technika komputerowa dojrzała do realizacji powszechnego zdalnego dostępu. Ale nie był jeszcze rozwiązany problem taniej teledacji. Z tej naglącej potrzeby rodzi się modem - urządzenie zmieniające (modulujące) sygnały cyfrowe na analogowe. Takie właśnie sygnały przesyłane są w sieciach telefonicznych i sygnał ten może być przesyłany na dowolne odległości. W sąsiedztwie zdalnego terminala drugi modem demoduluje sygnał z sieci telefonicznej na sygnał cyfrowy właściwy terminalowi. W dwa lata później intensywne prace nad oprogramowaniem sieciowym pozwalają utworzyć pierwszą sieć komputerową łączącą różne komputery i ogromne ilości terminali.

Wiele z warunków ówczesnej nowoczesności spełnia MERA-400. Ale nie wszystkie. Na przykład w okresie jej powstawania istniało już utrwalone pojęcie rodziny komputerów.

Komputery z początku swej epoki miały różne organizacje logiczne, czyli były wzajemnie niezgodne. Posiadały różne struktury danych i i zbiorów oraz odmienne zasady działania. Oprogramowanie systemowe i użytkowe - pisane zwykle w językach niskiego poziomu - było także wzajemnie niezgodne.

W celu zmniejszenia kosztów przenoszenia oprogramowania wprowadzono koncepcję rodziny komputerów, która składa się zwykle z kilku do kilkunastu modeli o zróżnicowanej wydajności - np. od 0,01 do 10 Mips - i mających wzajemnie zgodne struktury danych oraz zgodne zasady działania: ODRA 1300 (1325, 1304, 1305), ICL 1900, IBM 360, IBM 370, CDC 6000, CYBER-70 (72, 73, 74).

Wprowadzanie rodzin komputerów było sprawą jednego producenta zabiegającego o możliwość oferowania kupującym maszyn o różnej wydajności opracowywanych z możliwie małym nakładem kosztów. Wkrótce jednak wprowadzanie coraz to nowych, zaskakujących rozwiązań technicznych i technologicznych zaostrzało konkurencję na rynku i powodowało ze strony użytkowników systemów cyfrowych chęć dokonywania zmian elementów systemu (hardware i software) i to przy nabywaniu ich u różnych producentów - przy zachowaniu w stanie nienaruszonym pozostałych zasobów systemu tak hardware'owych jak i software'owych. Ten nowy trend uzyskał miano migracji - migration.

Migration zaostrzyła wymagania dotyczące kompatybilności sprzętowej, standaryzacji, doprowadziła do zmniejszenia trudności przy przenoszeniu oprogramowania oraz narzuciła konieczność współpracy różnych systemów operacyjnych różnych maszyn i oprogramowania sieciowego.

W okresie powstania zjawiska migracji wszyscy producenci widzieli konieczność dokonywania daleko idącej standaryzacji. Zachęca do tego

również powstanie wyspecjalizowanych producentów podzespołów OEM (Original Equipment Manufacturer).

Na początku określano tak producentów urządzeń peryferyjnych: jednostek dyskowych, drukarek, przewijaków taśm magnetycznych, czytników, ekranów, terminali.

Następnie zaczęto używać tego określenia dla producentów podstawowych elementów: np. ramiona prowadzące głowic dla pamięci dyskowych, głowice i silniki dla przewijaków taśm magnetycznych.

Określenie OEM używa się w odniesieniu do producentów jednostek centralnych, oprogramowania, mikroprocesorów, a stąd znaczna część obrotów całego przemysłu informatycznego przypada na OEM. Natomiast jak już wspomniano stymuluje to postępy standaryzacji.

Standaryzacja jest w sposób naturalny związana z koniecznością dokonywania połączeń różnych elementów systemów cyfrowych. Ułatwia ona opracowywanie sprzęgów w systemach cyfrowych (por. np. s. 19).

Przez sprzęg (interface) rozumie się na ogół granicę między współpracującymi modułami systemu. Sprzęg obejmuje zbiór informacji o rodzaju stosowanych łącz oraz zasad określających sposób łączenia a ponadto wykaz i parametry wymienianych sygnałów oraz protokoły przesyłania informacji umożliwiające współpracę różnych elementów systemu.

Standaryzacja obejmuje również obszary softrare'u. Chcąc ułatwić przenoszenie oprogramowania między różnymi systemami oraz zapewnić użytkownikom długotrwałe korzystanie z niego mimo zmiany komputera ANSI (American National Standard Institute) z pewnym opóźnieniem oddała do użytku swoją wersję Fortran 77. Jest ona szeroko rozpowszechniana wśród zachodnich prodzcentów emc (elektronicznych maszyn cyfrowych). Zmniejsza ona uciążliwości przenoszenia oprogramowania we Fortranie. Przenoszalność (portability) była poprzednio znacznie trudniejszym zagadnieniem.

Innym przykładem standardu jest GKS (Graphical Kernal System) -

bazowy software graficzny uznany przede wszystkim przez DIN (Deutsche Industrie Normen) oraz ISO (International Standards Organization). GKS jest stosowany dla wszystkich urządzeń graficznych i ułatwia korzystanie z różnych pakietów graficznych wysokiego poziomu.

Migracja wymusza na producentach również i to, że starają się oni o to aby nabywcy posiadający już pewien system obliczeniowy z określonym systemem operacyjnym mogli swój system rozszerzać przez zakup nowego komputera. Taką konieczność dostosowywania narzuca np. szerokie rozpowszechnienie się systemu operacyjnego UNIX.

Wymieniony w tablicy 1.2 i 1.3 firma CDC (Control Data Corporation) jest znana z wyposażenia swych systemów cyfrowych w bardzo dobre oprogramowanie systemowe. Już dla kilku serii komputerów tej firmy stosowany jest system NOS (Network Operating System). Firma stale i starannie wprowadza poprawki do systemu oraz uzupełnia go. Kolejną wersją NOS jest system NOS/BE, gdzie BE jest akronimem: Batch Enviroment.

Ostatnią wersją jest system NOS/VE gdzie VE jest akronimem: Virtual Enviroment. Ten system wprowadzono dla serii supermini Cyber 180. Umożliwia to oferowanie ich tym posiadaczom rozbudowanych systemów komputerowych pracujących pod UNIX'em, którzy zachcą podnieść wydajność przez dołączenie komputera z szeregu Cyber 180.

System UNIX opracowany w Bell Laboratories rozpowszechniony został szczególnie dla komputerów serii PDP-11 (11/20; 11/34; 11/40; 11/45; 11/60; 11/70/). Obecnie jest na tyle rozpowszechniony wśród posiadaczy systemów mini i mikrokomputerowych, że wszyscy producenci chcąc oferować swoje komputery z indywidualnie u siebie opracowanymi systemami operacyjnymi - muszą dostosowywać się do UNIX'a.

Minikomputery w ciągu dwudziestu lat swego istnienia znalazły ogromne zastosowanie w najrozmaitszych dziedzinach informatyki:

od EPD lub APD (Elektroniczne Przetwarzanie Danych lub Automatyczne Przetwarzanie Danych) do niezliczonych przykładów użycia ich dla realizacji obliczeń numerycznych.

Ponadto mają one swój liczący się udział w takich wydzielonych obszarach, którym nadaje się ogólną nazwę komputerowo wsparte lub komputerowo wspomagane:

- CAD: Computer Aided Designing - komputerowo wsparte projektowanie,
- CAM: Computer Aided Manufacturing - komputerowo wsparte wytwarzanie,
- CAE: Computer Aided Engineering - komputerowo wsparte działanie inżynierskie,
- COMO: Computer Aided Modelling - komputerowo wsparte modelowanie,
- CAADS: Computer Aided Analysis of Dynamic Systems - komputerowo wsparta analiza dynamicznych systemów,
- CAI: Computer Aided Inspection - komputerowo wsparta kontrola,
- ICM: Interactive Computer Modelling - interaktywne komputerowe (wspomagane) modelowanie stanowiące zawężenie CAMO do trójwymiarowego modelowania.

Minikomputery stanowią narzędziowy składnik wielu specjalnych systemów z zakresu:

- Artificial Intelligence - sztuczna inteligencja,
- Pattern Recognition - rozpoznawanie wzorców zwane też rozpoznawaniem obrazów.

Minikomputery stanowią bazę systemów sterowania eksperymentem lub produkcją. Do tej grupy należy szeroko znany CAMAC (Computer Application for Measurement And Control - zastosowanie maszyn cyfrowych do pomiarów i sterowania).

System CAMAC jest układem standardowego interface'u (sprzęgu - por. s. 17) pomiędzy emc - z reguły minikomputerem - a przetwornikami danych. Obejmuje on zbiór reguł określających sposób projektowa-

nia i użytkowania modułowego elektronicznego systemu przeznaczonego do zbierania i przetwarzania danych.

W chwili obecnej pojawiają się liczne, coraz to nowe dziedziny zastosowań minikomputerów wśród których wymienić można FMS (Flexible Manufacturing Systems) odnoszące się do komputerowego sterowania wytwarzaniem produktów uzyskiwanych w wyniku współdziałania wielu stanowisk. Realizowane przez nie operacje nie muszą być ustawiane w sztywne ciągi. Chcąc uzyskiwać najlepsze efekty należy je elastycznie dostosowywać do chwilowych warunków. I to jest obszarem zagadnień FMS.

Podobnie występuje uogólnienie komputerowego sterowania produkcją w pojęciu CIM (Computer Integrated Manufacturing).

Minikomputery znajdują szerokie zastosowanie w dziedzinie szeroko pojętej grafiki komputerowej. Obejmuje ona wspomniane już poprzednio CAD ale jej zasięg stale się powiększa i można tu wymienić ICONICS czyli Ikonikę - komputerowo wspartą komunikację wzrokową. Tu jednak zauważyć należy, że do produkcji filmów animowanych nie wystarczają zasoby minikomputerów. Obecnie do tych celów wykorzystuje się superkomputery - na przykład CRAY sięgające 100 i 400 Mips. Przy tej okazji można wspomnieć, że aktualnie uznaje się, że superkomputer jest to system mający zdolność wykonywania ponad dwadzieścia mega instrukcji na sekundę. W tablicy 1.2 podany jest szereg supermini Cyber 180 z których typ 955 dwuprocesorowy oraz 990 i 990 dwuprocesorowy dają możność osiągnięcia ponad 20 Mips. I te właśnie supermini mimo, że nie odgrywają wiodącej roli w produkcji pełnometrażowych filmów animowanych, znajdują jednak pewne zastosowania przy produkcji wstawek i trików.

Minikomputery mają również swoją bardzo ważną pozycję w zakresie wspomagania zarządzania. Jednym z jego obszarów jest wizualizacja danych - data visualization, zwana też business graphics lub management graphics.

W systemach data visualization dane z różnych dziedzin gromadzone

są w wielkich bazach danych skąd pobierane są dla rysowania histogramów, sporządzania wykresów skupień itp. Stanowią one odpowiedź na zadane pytania. Dla obsługi bardziej złożonych systemów wizualizacji stosowane są minikomputery ze specjalnym oprogramowaniem dla realizacji dialogowego współdziałania managera ze systemem.

Ogromnym obszarem użycia minikomputerów, szczególnie supermini staje się symulacja stosowana do zróżnicowanych zjawisk; technicznych, ekonomicznych, biologicznych, społecznych.

Symulacja jest wciąż zbyt słabo rozwinięta w Polsce. Za jej rozpowszechnieniem przemawiają względu ekonomiczne. Znacznie taniej i szybciej można przeprowadzić na przykład eksperyment komputerowy - inaczej przeprowadzić symulację zjawiska niż badać je w oparciu o eksperyment rzeczywisty. Oczywiście ten eksperyment nie zostaje na ogół wyeliminowany z opracowania zjawiska, ale przeprowadza się go po dobrym rozeznaniu komputerowym.

W okresie powstawania minikomputera Mera-400 był on pod względem technologii i architektury projektem zaawansowanym. Posiadał między innymi możliwość pracy w układzie dwuprocesorowym (dual processor) oraz umożliwiał dołączenie dodatkowego procesora przyspieszającego działania matematyczne.

Mera-400 nie była projektowana jako jeden z elementów szeregu komputerów. Na przykład typ 180 szeregu Cyber 180 jest jednym z elementów tego szeregu. Jest to powód mniejszego przywiązania wagi do ogólnie rozumianej kompatybilności Mery-400 z innymi systemami niż ma to miejsce we współcześnie produkowanych systemach komputerowych.

1.2. Cel i zakres skryptu

Celem skryptu jest ułatwienie studentom uczestnictwa w zajęciach dydaktycznych prowadzonych przez Zakład Informatyki, oraz przygotowanie do przyszłej pracy zawodowej.

Chcąc stworzyć dobre narzędzia należało przygotować skrypt tak by wykorzystywanie go nie wymagało posługiwania się obszerną literaturą i oczywiście poszukiwań za nią.

Równocześnie poszczególne rozdziały zawierają wiadomości o szerszym zakresie wykorzystania. Dotyczy to szczególnie rozdziału 4: Możliwości programowania i języki dostępne w systemie. Można też podkreślić i to, że w zakresie języka BASIC istnieje niedobór literatury w języku polskim. Tym większy jest pożytek z opracowania przez prof. dra hab. inż. Ryszarda Tadeusiewicza części dotyczącej języka BASIC w systemie MERA-400.

2. CHARAKTERYSTYKA SPRZĘTÓW SYSTEMU MERA-400

MERA-400 jest minikomputerem, który produkowany był do połowy lat osiemdziesiątych w Zakładach Wytwórczych Przyrządów Pomiarowych i Systemów Minikomputerowych w Warszawie. Uniwersalność budowy systemu MERA-400 pozwala na jego zastosowanie do różnorodnych celów: obliczeń naukowo-technicznych, przetwarzania danych, dla celów zarządzania, dla potrzeb dydaktyki, do sterowania procesami technologicznymi itd.

Przewidywany sposób wykorzystania minikomputera decyduje w głównej mierze o konfiguracji sprzętowej oraz o trybie pracy maszyny. Modułowa budowa MERY-400 sprawia, że ewentualne zmiany w wyposażeniu w urządzenia zewnętrzne lub modyfikacje w obrębie samej jednostki centralnej są stosunkowo łatwe do przeprowadzenia i nie nastręczają większych technicznych trudności.

Klasyfikację sprzętową MERY-400 sprowadzić można do trzech zasadniczych klas:

- wersja rdzeniowa,
- wersja podstawowa,
- wersja rozszerzona.

Wersja rdzeniowa przeznaczona jest głównie do prac w układach automatyki, sterowania procesami itd., pracujących w trybie czasu rzeczywistego. Układy takie nie wymagają aby minikomputer posiadał zewnętrzne pamięci magnetyczne. MERA-400 eksploatowana w tym trybie pracy posiada bezdyskowy system operacyjny.

Wersja podstawowa minikomputera MERA-400 jest najszerzej wykorzystywaną przez użytkowników. Maszyny cyfrowa obok jednostki centralnej,

pamięci operacyjnej i tradycyjnych urządzeń wejścia/wyjścia to jest czytnika tasiemki papierowej, perforatora tasiemki papierowej, drukarki mozaikowo-znakowej i konsoli operatorskiej. wyposażona jest w pamięć zewnętrzną - stację dysków sztywnych. Tego typu wersja sprzętowa wykorzystywana jest głównie do pracy wsadowej, ale możliwa jest również praca konwersacyjna. Oprogramowanie systemowe zawiera dyskowy system operacyjny.

Wersja rozszerzona minikomputera MERA-400 jest klasą najbardziej rozbudowaną sprzętowo. Przykład takiej konfiguracji przedstawiono w rozdziale 2.1 na rysunku 2.1. Maszyna w wersji rozszerzonej musi być wyposażona w system operacyjny, który zarządza pracą wszystkich urządzeń zewnętrznych korzystających z zasobów pamięciowych oraz procesorów systemowych.

2.1. Możliwości sprzętowe w ramach systemu MERA-400

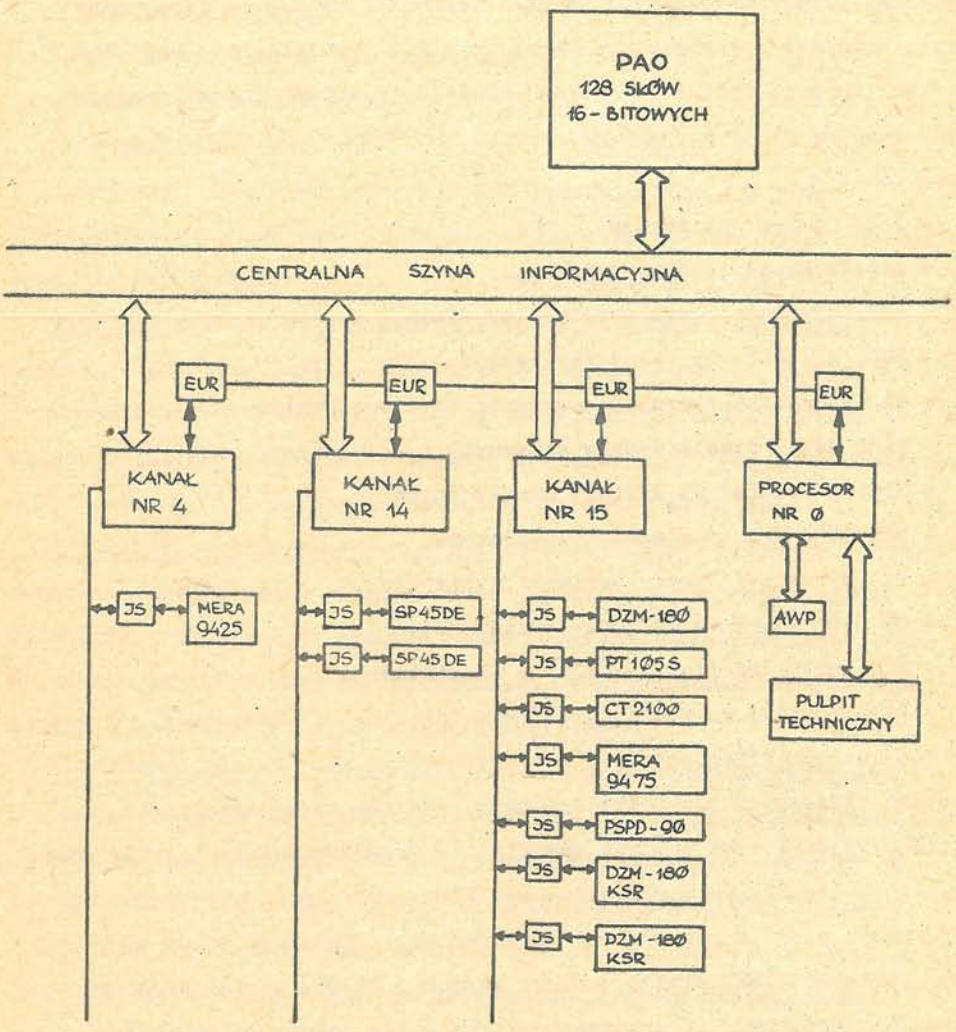
MERA-400 jest minikomputerem, do którego budowy wykorzystano układy średniej i wielkiej skali integracji. Jest to maszyna cyfrowa o architekturze logicznej odpowiadającej wymaganiom stawianym współczesnym małym i średnim systemom cyfrowym.

MERA-400 posiada standardowy interfejs, który łączy poszczególne bloki funkcjonalne (moduły) systemu. Możliwość dołączania nowych modułów do interfejsu sprawia, że struktura sprzętowa jest łatwo modyfikowalna.

Maksymalna konfiguracja jednostki centralnej systemu MERA-400 zawierać może:

- dwa procesory wraz z arytmometrem wielokrotnej precyzji,
- 512 ksłów pamięci operacyjnej,
- 16 kanałów wejścia/wyjścia.

Procesory (lub jeden procesor) sterują pracą jednostki centralnej i komunikują się poprzez centralną szynę informacyjną z pamięcią



Rys. 2.1. Przykładowa konfiguracja sprzętowa systemu MERA-400

Źródło: Opracowanie własne.

operacyjną oraz kanałami wejścia/wyjścia. Z kolei kanały wejścia/wyjścia (kanały transmisji) poprzez jednostki sterujące zarządzają pracą urządzeń zewnętrznych. Na rysunku 2.1 przedstawiono przykładową konfigurację sprzętową systemu MERA-400 w wersji rozszerzonej. Poszczególne bloki opisane na rysunku 2.1 mają następujące znaczenie:

- PAO: pamięć operacyjna,
- Kanał: kanał wejścia/wyjścia,
- AWP: arytmometr wielokrotnej precyzji,
- EUR: elementarny układ rezerwacji,
- JS: jednostka sterująca urządzeniem zewnętrznym,
- MERA 9425: stacja dysków sztywnych (pamięć zewnętrzna),
- SP45DE: pamięć na dyskach elastycznych,
- DZM-180: drukarka mozaikowo-znakowa,
- PT 105 S: perforator tasiemki papierowej,
- CT 2100: czytnik tasiemki papierowej,
- Mera 9475: monitor ekranowy z klawiaturą,
- DZM-180-KSR: konsola operatorska (drukarka z klawiaturą).

Obok urządzeń zewnętrznych zaznaczonych na rys. 2.1 do składu systemu MERA-400 można podłączyć inne niestandardowe urządzenia wejścia/wyjścia oraz pamięci zewnętrzne. Urządzeniami tymi przykładowo mogą być: czytnik kart perforowanych firmy KOVO, drukarka wierszowa typu DW3M, pisak x-y typu KL2, autokreślarka produkcji firmy KOVO typu Digigraf 1208, stacja pamięci taśmowej PT305, pamięć kasetowa typu PK1. Wszystkie te urządzenia łączą się z jednostką centralną za pomocą kanałów wejścia/wyjścia.

2.2. Organizacja jednostki centralnej

Jednostka centralna pod względem konstrukcyjnym stanowi odrębną obudowę zawierającą następujące moduły funkcjonalne:

- procesor,
- pamięć operacyjną,
- arytmometr wielokrotnej precyzji,
- kanały wejścia/wyjścia,
- interfejs systemu.

Podstawową jednostką informaty i w systemie MERA-400 jest 16-bitowe słowo maszynowe. Jednostka centralna posiada osiem 16-bitowych rejestrów uniwersalnych, 32-bitowy rejestr przerwań.

Procesor zawiera układy sterujące pracą jednostki centralnej, układy realizujące funkcje decyzyjne wykonania rozkazu, obsługi przerwania, operacji kluczy pulpitu technicznego, układy dekodera, układy rejestrów. Integralną częścią procesora jest pulpit techniczny (patrz rys. 2.3) widoczny na zewnątrz jednostki centralnej.

Arytmometr wielokrotnej precyzji zwiększa zakres operacji wykonywanych sprzętowo przez mikrokomputer. Operacje te są następujące: dodawanie i odejmowanie oraz mnożenie i dzielenie stałoprzecinkowe, normalizacja liczb zmiennoprzecinkowych oraz dodawanie, odejmowanie, mnożenie i dzielenie liczb zmiennoprzecinkowych.

Pamięć operacyjna przeznaczona jest do przechowywania programów i danych wymagających szybkiego i swobodnego dostępu. Pamięć podzielona jest na dwa obszary: systemowy i użytkowy. System operacyjny ma dostęp do dowolnego miejsca pamięci w obszarze użytkowym. Obszar systemowy, w którym rezyduje system operacyjny, jest niedostępny dla programu użytkowego. W trakcie generowania systemu operacyjnego w minikomputerze dokonywany jest podział pamięci na bloki o stałej wielkości. Podstawowa wielkość bloku pamięci wynosi 32 Kszów (kilo-słów 16 bitowych).

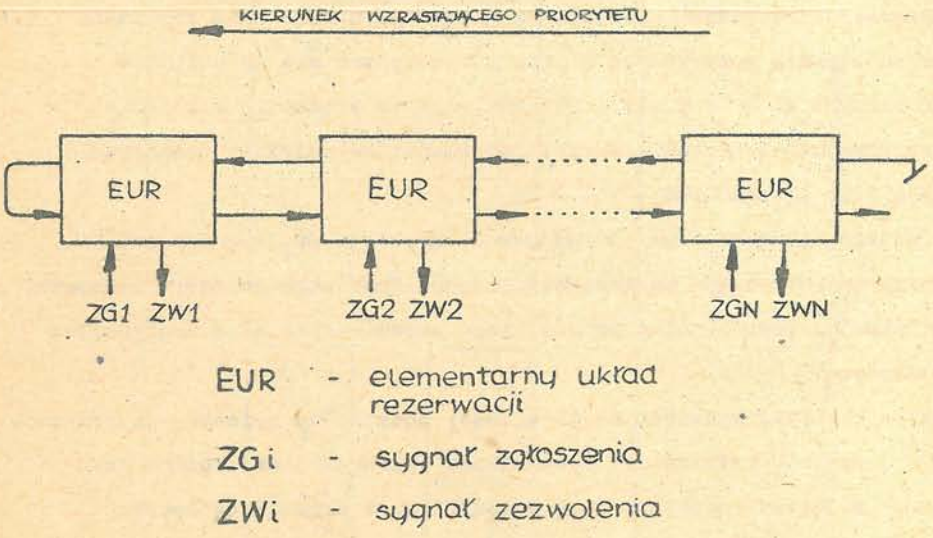
Każdy blok dzielony jest na segmenty z przyrostem co 4 Kszowa. W chwili obecnej najpopularniejszą pamięcią operacyjną dla minikomputera MERA-400 jest pamięć ferrytowa. Fizycznie jeden blok pamięci operacyjnej ferrytowej ma pojemność 8 Kszów. Całkowita długość słowa

dla tego typu pamięci wynosi 18 bitów, przy czym jeden bit jest bitem wolnym (nie wykorzystany), a jeden traktowany jest jako bit parzystości. Pamięć ta charakteryzuje się czasem dostępu ok. 600 μs i czasem cyklu ok. 1100 μs. Aktualnie coraz większą popularność zyskuje sobie pamięć półprzewodnikowa, budowana najczęściej w oparciu o elektroniczne układy wielkiej skali integracji. Przykładowo pamięć półprzewodnikowa firmy Computex charakteryzuje się długością słowa 17 bitów (jeden bit jest bitem parzystości) i porównywalnymi z pamięcią ferrytową czasami cyklu i dostępu. Jednocześnie minimalna pojemność tego typu pamięci wynosi 64 Ksłowa.

Efektywne wykorzystanie zasobów pamięci operacyjnej zapewnia programowy podział tej pamięci. Polega on na tym, że obszar użytkowy pamięci o czym była mowa, jest organizowany przez system operacyjny na drodze programowej. Każdy segment pamięci (o wielkości 4 Ksłów) posiada 4-bitowy rejestr numeru bloku użytkowego oraz 3-bitowy rejestr adresu logicznego segmentu. Rozwiązanie takie determinuje maksymalną wielkość pamięci operacyjnej: 16 bloków po 32 Ksłowa, z których każdy dzielony jest na 8 segmentów po 4 Ksłowa.

Centralna szyna informacyjna służy do przesyłania danych, adresów i sygnałów sterujących. Sterowanie centralnej szyny odbywa się poprzez układ rezerwacji szyny i rozdzielone jest pomiędzy moduły nadawcze systemu, którymi są kanały i procesory. Centralna szyna informacyjna wraz z układem rezerwacji stanowi interfejs systemu. Układ rezerwacji umożliwia dostęp do szyny informacyjnej na zasadzie wzrastającego priorytetu. Układ ten zbudowany jest z elementarnych układów rezerwacji. Komunikacja pomiędzy tymi układami odbywa się przy pomocy dwóch sygnałów: sygnału zgłoszenia i zezwolenia. Moduł żądający transmisji wysyła sygnał do elementarnego układu rezerwacji. Jeżeli układ o wyższym priorytecie nie żąda dostępu do centralnej szyny informacyjnej i szyna ta jest wolna, układ żądający transmisji otrzymuje sygnał zezwolenia na tę transmisję. Jednocześnie blokowane są moduły mające

niższy priorytet. Następny dostęp do centralnej szyny informacyjnej otrzyma moduł o najwyższym priorytecie spośród oczekujących, nawet jeżeli moduł aktualnie prowadzący transmisję natychmiast po jej zakończeniu zgłasza żądanie ponownego zajęcia szyny. Celem takiego rozwiązania jest umożliwienie współzbieżnej pracy modułów systemu. Na rysunku 2.2 przedstawiono schematycznie priorytetowy układ rezerwacji centralnej szyny informacyjnej.



Rys. 2.2. Priorytetowy układ rezerwacji centralnej szyny informacyjnej

Źródło: /4/.

Transmisje interfejsowe przebiegają w sposób asynchroniczny. Każda transmisja poprzedzona musi być rezerwacją centralnej szyny informacyjnej. W danej chwili do centralnej szyny informacyjnej może być dołączony aktywnie tylko jeden moduł nadawczy.

2.3. Pulpit techniczny i uruchamianie maszyny

Pulpit techniczny jest składową częścią procesora minikomputera. Wbudowany jest w przednią część obudowy jednostki centralnej. Umieszczone są na nim wskaźniki świetlne i klucze (przełączniki) służące do obsługi maszyny. Widok pulpitu technicznego przedstawia rysunek 2.3.

Do włączania zasilania minikomputera służy stacyjka umieszczona z prawej strony na pulpicie technicznym. Pozycja OFF klucza świadczy o wyłączeniu zasilania, pozycja ON informuje o włączeniu zasilania i braku blokady pozostałych kluczy znajdujących się na pulpicie technicznym. Klucz w pozycji LOCK świadczy o włączeniu zasilania i jednoczesnym blokowaniu wszystkich kluczy na pulpicie technicznym z wyjątkiem klucza OPRQ.

Przełącznik wybierania rejestrów (pokrętło w środkowej części pulpitu technicznego) ma możliwość wyboru dowolnego rejestru jednostki centralnej). Poszczególne pozycje tego przełącznika mają następujące znaczenie:

R0-R7 - rejestry uniwersalne 16-bitowe; podstawowe rejestry do przechowywania informacji, wykonywania operacji oraz służące jako rejestry indeksowe przy B-modyfikacji argumentów (patrz punkt 2.4).

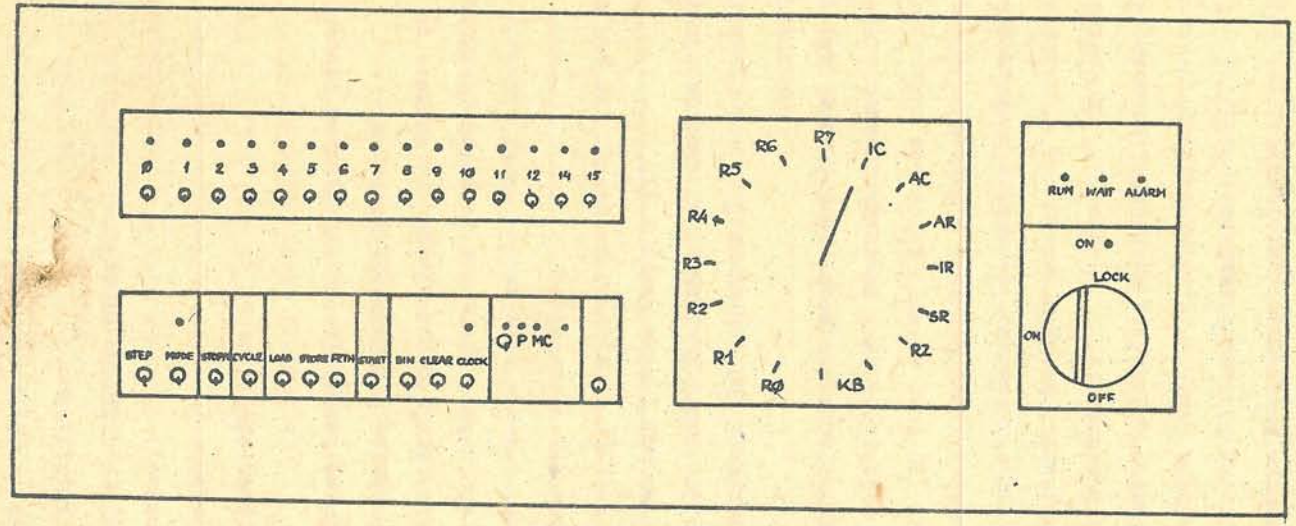
Rejestr R0 dodatkowo wykorzystywany jest jako rejestr stanu programu. Poszczególne pozycje tego rejestru mają następujące znaczenie:

Z (poz. 0) - wskaźnik zera; ustawiany w przypadku otrzymania zera w wyniku działań arytmetycznych i logicznych,

N (poz. 1) - wskaźnik znaku minus; ustawiany w przypadku otrzymania liczby ujemnej w wyniku działań arytmetycznych,

V (poz. 2) - wskaźnik nadmiaru; jedynka na tej pozycji oznacza przekroczenie zakresu liczb,

C (poz. 3) - wskaźnik przeniesienia; ustawiany zgodnie z przeniesie-



Rys. 2.3. Pulpit techniczny minikomputera MERA-400
Źródło: /4/.

niem z zerowej pozycji przy operacjach arytmetycznych,

L (poz. 4) - wskaźnik wyniku porównania (mniejszy),

E (poz. 5) - wskaźnik wyniku porównania (równy),

G (poz. 6) - wskaźnik wyniku porównania (większy),

Y (poz. 7) - wskaźnik przechowujący bit wychodzący poza rejestr przy operacjach przesuwania,

X (poz. 8) - wskaźnik ustawiany programowo,

pozycje 9 - 15 - wskaźniki przeznaczone do użytku programisty.

Zapis na pozycje 0 - 7 rejestru R0 nie jest wykonywany, gdy wykonywany jest program użytkowy (R0 jest wówczas rejestrem wynikowym).

IC - licznik rozkazów; rejestr służący do zaadresowania komórki pamięci operacyjnej z której pobrany będzie następny do wykonania rozkaz,

AR - rejestr adresowy pamięci; rejestr służący do przechowywania adresu przy każdym odwołaniu do interfejsu oraz do przechowywania wartości modyfikatora MOD (patrz punkt 2.4),

IR - rejestr rozkazów; rejestr do którego ładowany jest kod rozkazu po pobraniu go z komórki operacyjnej,

AC - rejestr akumulatora; rejestr zawierający drugi argument operacji arytmetycznych i logicznych, jest on niedostępny programowo,

RZ - rejestr zgłoszeń przerwań; 32-bitowy rejestr służący do zapamiętania 32 różnych przyczyn zgłoszonych przerwań.

Znaczenia poszczególnych pozycji rejestru zgłoszeń przerwań są następujące:

0 alarm zasilania (z procesora),

1 błąd parzystości pamięci,

2 brak pamięci operacyjnej,

3 zgłoszenie przerwania z drugiego procesora,

4 zanik zasilania (z interfejsu),

5 przerwania zegarowe,

6 nieprawidłowy rozkaz,

7 nadmiar dzielenia stałoprzecinkowego,

8 podmiar zmiennoprzecinkowy,

9 nadmiar zmiennoprzecinkowy,

10 błąd danych zmiennoprzecinkowych lub próba dzielenia przez zero,

11 do zastosowań specjalnych,

12 - 27 przerwania kanałowe,

28 zgłoszenie operatora OPRQ,

29 zgłoszenie przerwania z drugiego procesora (zgłoszenie o niższym priorytecie),

30 - 31 przerwania programowe,

SR - rejestr stanu; rejestr, którego poszczególne bity mają następujące znaczenia:

RM (poz. 0 - 9) rejestr masek przerwań.

Poniżej przedstawiono pogrupowanie przerwań w poziomy obsługi.

poziom	0	1	2	3	4	5	6	7	8	9	10
poz.RM	-	0	1	2	3	4	5	6	7	8	9
poz.RZ	0	1	2	3	4	5-11	12-13	14-15	16-21	22-27	28-31

Q (poz. 10) wskaźnik pracy systemu:

Q = 1 odpowiada pracy systemu operacyjnego,

Q = 0 odpowiada pracy programu użytkowego,

BS (poz. 11) wskaźnik zezwolenia na dostęp do bloku systemowggo pamięci innego procesora,

NB (poz. 12 - 15) rejestr numeru bloku pamięci,

KB - wybranie zawartości kluczy informacyjnych na pulpicie technicznym.

Klucze znajdujące się w lewej części pulpitu technicznego mają następujące znaczenie:

KB - 16 kluczy bistabilnych służących do ustawienia dowolnej 16-bitowej informacji.

START - klucz bistabilny "start-stop" służący do uruchomienia i zatrzymania maszyny,

CYCLE - klucz monostabliny "cykl". Naciśnięcie tego klucza powoduje wykonanie jednego cyklu maszynowego. Po wykonaniu operacji maszyna przechodzi do stanu "stop" (przy wyłączonym kluczu START),

OPRQ - klucz monostabilny, którego naciśnięcie powoduje zgłoszenie przerwania od operatora,

BIN - klucz monostabilny "wprowadź binarnie". Naciśnięcie tego klucza powoduje wykonanie operacji wstępnego wprowadzenia programu. Działa tylko w stanie "stop" maszyny.

STORE - klucz monostabilny "pamiętaj". Jego naciśnięcie powoduje zapamiętanie zawartości wybranego rejestru w komórce pamięci wskazanej zawartością rejestrów AR i NB (pozycje 12 - 15 w rejestrze SR), a następnie zwiększenie zawartości AR o 1. Działa tylko w stanie "stop" maszyny.

FETCH - klucz monostabilny "pobierz". Jego naciśnięcie powoduje odczytanie jednego słowa z kómórki pamięci o adresie wskazanym przez rejestr AR i NB. Następnie słowo to umieszczane jest w rejestrze wybranym przełącznikiem wybierania rejestrów, a zawartość AR zwiększana jest o 1. Klucz działa tylko w stanie "stop" maszyny.

MODE - klucz stabilny "reżim pracy" ustalający jeden z następujących trybów pracy:

- praca ciągła w której maszyna wykonuje rozkazy z pełną szybkością (wskaźnik MODE zgaszony),
- praca krokowa, w której maszyna działa start-stopowo wykonując elementarny krok operacji przy każdym naciśnięciu klucza STEP (wskaźnik MODE zapalony).

LOAD - klucz monostabilny "umieść". Jego naciśnięcie powoduje umieszczenie w wybranym rejestrze informacji ustawionej na kluczach informacyjnych KB. Działa tylko w stanie "stop" maszyny.

STEP - klucz monostabilny "krok" działający w trybie pracy krokowej (przy zapalonym wskaźniku MODE). Naciśnięcie tego klawisza powoduje wykonanie elementarnego kroku operacji, a następnie zawieszenie działania do chwili ponownego naciśnięcia tego klucza.

STOP+N - klucz monostabilny "stop na adresie" służący do zatrzymania maszyny przy odwołaniu do komórki pamięci o adresie ustawionym na kluczach informacyjnych KB na pozycjach 1 - 15. Pozycja zerowa wskazuje na blok systemowy (przy ustawieniu w stan 1) lub blok użytkowy pamięci (przy ustawieniu w stan 0).

CLOCK - klucz bistabilny "zegar" którego włączenie powoduje uruchomienie zegara czasu rzeczywistego. Przy włączeniu tego klucza zapala się lampka umieszczona nad nim.

CLEAR - klucz monostabilny "zeruj" którego naciśnięcie powoduje ustawienie wskaźników i sterowanie systemu w stan początkowy.

Wskaźniki świetlne znajdujące się na pulpicie technicznym posiadają następujące znaczenie:

RUN - "praca"; lampka zapalona gdy maszyna jest w stanie "start",

WAIT - "czekaj"; lampka zapalona gdy maszyna jest w stanie "czekaj",

ALARM - lampka zapalona gdy wystąpił błąd parzystości pamięci lub brak pamięci w bloku systemowym pamięci operacyjnej,

Q - lampka wyświetlająca zawartość wskaźnika Q w rejestrze SR,

IRQ - lampka sygnalizująca przyjęcie przerwania,

MC - lampka zapalona gdy ostatnim wykonanym rozkazem był rozkaz modyfikuj (MOD),

P - lampka wyświetlająca zawartość wskaźnika przeskoku P (gdy P = 1 to IC = IC + długość rozkazu + 1). Jest ona zapalona

gdy ostatnim wykonanym rozkazem był rozkaz ustawiający przeskok,

MODE - lampka zapalona gdy maszyna jest w trybie pracy krokowej,

STOP+N - lampka zapalona gdy w maszynie uruchomiony został mechanizm zatrzymania się na adresie wskazanym na kluczach informacyjnych,

CLOCK - lampka zapalona gdy włączony jest zegar czasu rzeczywistego,

ON - lampka zapalona w przypadku prawidłowego zasilania maszyny.

Przy pomocy pulpitu technicznego można uruchomić lub zatrzymać pracę minikomputera. W zależności od systemu operacyjnego proces uruchamiania pracy maszyny jest następujący:

Praca bez systemu operacyjnego

- przekręcić klucz w stacyjne z pozycji OFF w pozację ON,
- włączyć zasilanie urządzeń zewnętrznych współpracujących z jednostką centralną.

Praca z systemem operacyjnym SOM 3

- przekręcić klucz w stacyjce z pozycji OFF w pozycję ON,
- włączyć zasilanie urządzeń zewnętrznych współpracujących z jednostką centralną (dysk sztywny, czytnik tasiemki papierowej, konsolę operatorską, drukarkę),
- wczytać tasiemkę binarną zawierającą bootstrap (ściągaczkę) systemu operacyjnego. Operacja ta wymaga następujących czynności:
  a) włożyć pod ramię czytnika tasiemkę bootstrap,
  b) załadować do rejstru AR wartość 0 (ustawić pokrętło na rejestr AR, ustawić na kluczach informacyjnych KB wartość 0 i nacisnąć klawisz LOAD),
  c) załadować do rejestru IR wartość A0BE (szesnastkowo),
  d) nacisnąć klucz CLEAR,
  e) nacisnąć klawisz BIN,
  f) nacisnąć klawisz CLEAR,
  g) włączyć klawisz START.

Po wczytaniu tasiemki bootstrap (BIN) i załadowaniu systemu operacyjnego z dysku sztywnego do pamięci operacyjnej maszyna zgłasza gotowość do pracy przez wydruk na konsoli operatorskiej znaku apostrofu.

Praca z systemem operacyjnym SOM 3P

Po włączeniu zasilania jednostki centralnej i urządzeń zewnętrznych oraz po wczytaniu tasiemki bootstrap systemu SOM 3P (tak jak było to opisane dla systemu SOM 3) przed włączeniem klucza START na kluczach informacyjnych KB należy dodatkowo ustawić odpowiednią informację. Przy uruchamianiu maszyny z systemem SOM 3P poszczególne klucze informacyjne mają następujące znaczenia:

numer klucza	stan klucza	znaczenie
0		zmiana urządzenia z którego ładowany jest system
	0	dysk sztywny
	1	urządzenie w kanale znakowym
1		zmiana urządzenia
	0	bez zmian
	1	ładowanie z urządzenia 1
2	0	system startuje po załadowaniu
	1	stop po załadowaniu systemu
3	0	LSU startuje po załadowaniu
	1	stop po załadowaniu LSU
4	0	standardowy adres dyskowy
	1	klucze 8 - 15 stanowią adres dyskowy
5	0	nie zeruj pamięci
	1	zeruj pamięć
6		opis urządzenia:
	0	numer urządzenia (bity 8,9,10) i kanału (bity 11-14) na kluczach gdy urządzenie znakowe
	1	numer urządzenia (bity 8,9,10) i kanału (bity 11-14) w rejestrach: R6 gdy urzączenie znakowe R7 gdy urządzenie pamięciowe
7		talerz dysku:
	0	talerz stały
	1	talerz wymienny
8-15		dyskowy adres systemu gdy urządzenie pamięciowe, numer urządzenia (bity 8,9,10) i kanału (bity 11-14) na kluczach gdy urządzenie znakowe

2.4. Maszynowa prezentacja informacji

Podstawową postacią informacji w systemie MERA-400 jest 16-bitowe słowo maszynowe. Informacja traktowana może być jako dana, rozkaz lub adres.

1. Dane

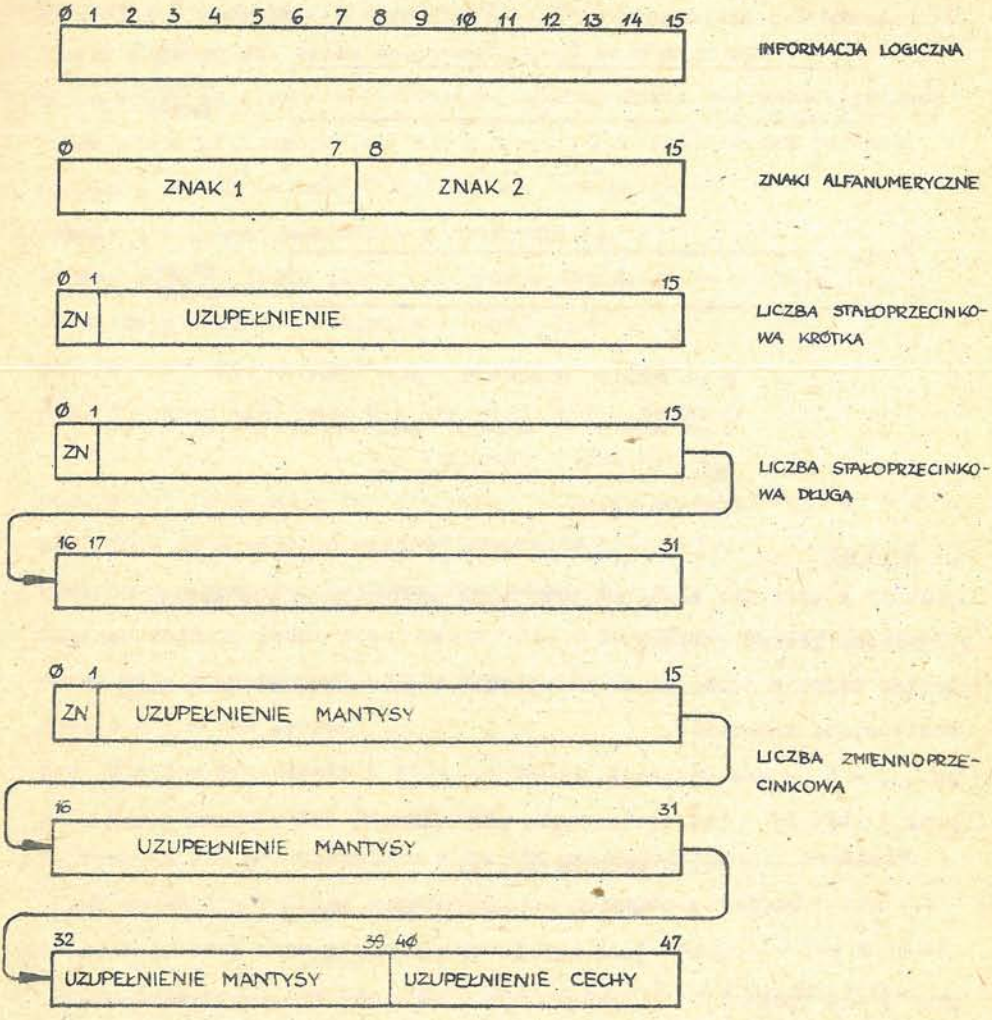
Dane mogą mieć jedną z następujących postaci:

- informacja logiczna zajmująca jedno słowo maszynowe;
- znaki alfanumeryczne podawane standardowo w kodzie ISO-7[1] zajmujące 7 bitów. Oznacza to, że w jednym słowie maszynowym mieszczą się dwa znaki alfanumeryczne. W systemie MERA-400 znaki te mogą być zapisywane w CAN - kodzie umożliwiającym na upakowanie trzech w jednym słowie maszynowym;
- liczba stałoprzecinkowa krótka, która zajmuje jedno słowo maszynowe. Traktowana jest jako liczba całkowita zapisana w notacji uzupełnieniowej do dwóch;
- liczba stałoprzecinkowa długa zajmująca dwa słowa maszynowe. Traktowana jest jako liczba całkowita zapisana w notacji uzupełnieniowej do dwóch;
- liczba zmiennoprzecinkowa zajmująca trzy słowa maszynowe. Standardowa postać liczby zmiennoprzecinkowej jest następująca: pierwszych 40 bitów zajmuje mantysa traktowana jako liczba z przedziału (-1,1) (kropka po pierwszym bicie znaku mantysy), zapisana w notacji uzupełnieniowej do dwóch; cecha zajmuje 8 bitów i traktowana jest jako liczba całkowita zapisana w notacji uzupełnieniowej do dwóch.

Na rysunku 2.4 przedstawiono graficznie postacie danych minikomputera MERA-400.

---

[1] Obecnie znajduje się w użyciu wiele różnych zbiorów znaków. Jednym z takich zbiorów jest 7-bitowy kod ISO-7 zalecany przez ISO (International Standardization Organization).



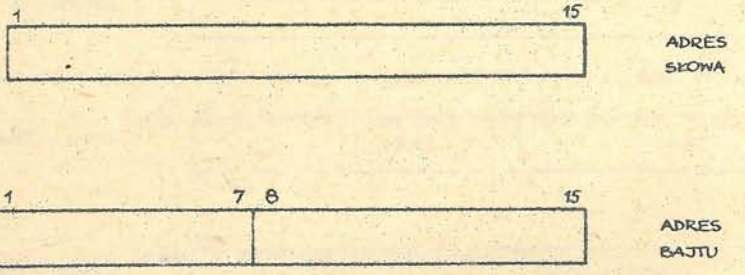
Rys. 2.4. Postacie danych minikomputera MERA-400
Źródło: /4/.

2. Adresy

Adresy przyjmują jedną z postaci przedstawionych poniżej:

- adres słowa będący 15-bitową całkowitą liczbą binarną,
- adres bajtu będący 16-bitową całkowitą liczbą binarną, przy czym adres parzysty oznacza bajt starszy (pozycje 0 - 7), a adres nieparzysty oznacza bajt młodszy (pozycje 8 - 15).

Na rysunku 2.5 przedstawiono graficznie postacie adresów.



Rys. 2.5. Postacie adresów
Źródło: /4/.

3. Rozkazy

Rozkazy w systemie MERA-400 przyjmują następujące postacie:

- podstawowa postać rozkazu.

Rozkaz zajmuje jedno słowo maszynowe, a poszczególne bity mają następujące znaczenie:

poz. 0 - 5 - kod operacji,

poz. 6 (bit D) - bit adresowania pośredniego, lub stanowi przedłużenie kodu operacji:

D = 0 nie ma adresacji pośredniej,

D = 1 jest adresacja pośrednia,

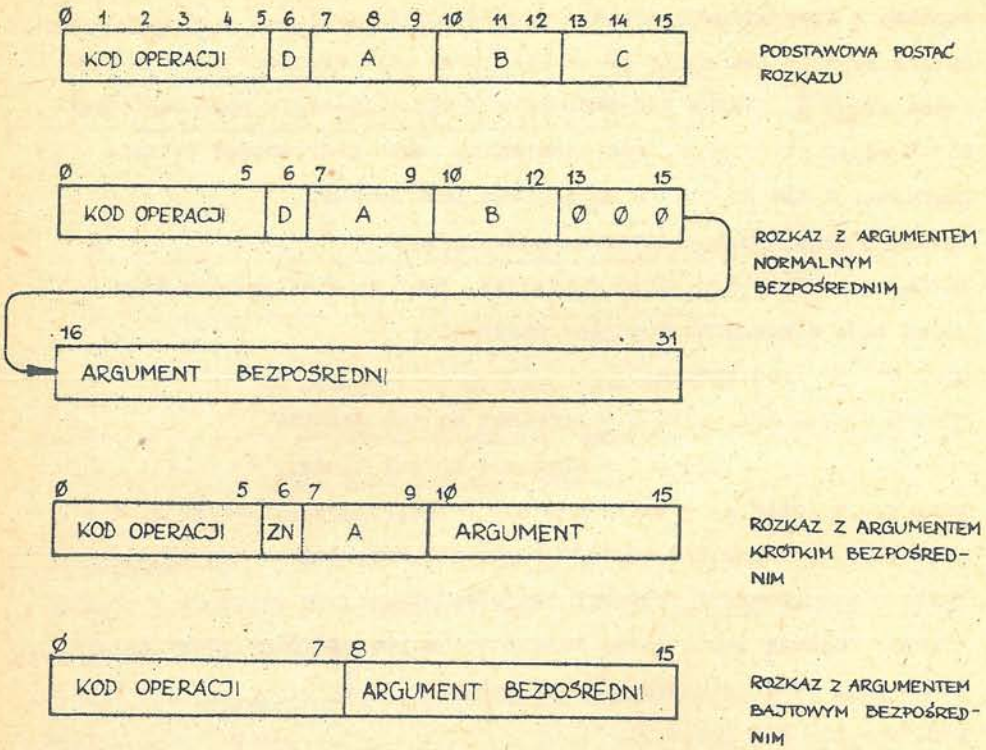
poz. 7 - 9 (pole A) - zawiera numer rejestru uniwersalnego (R1-R7), lub stanowi przedłużenie pola kodu operacji,

poz. 10 - 12 (pole B) - pole B - modyfikacji zawierające numer rejestru (R1-R7) służącego do B-modyfikacji argumentu, lub stanowi przedłużenie pola kodu operacji,

poz. 13 - 15 (pole C) - zawiera numer rejestru uniwersalnego (R1-R7) lub stanowi przedłużenie pola kodu operacji.

- rozkazy z argumentem normalnym bezpośrednim.
  Rozkaz zajmuje dwa słowa maszynowe, przy czym argument zajmuje następne słowo za słowem podstawowym rozkazu. Znaczenie poszczególnych bitów słowa pierwszego jest identyczne jak w podstawowej postaci rozkazów, z tym że pole C wypełnione jest zerami.
- rozkaz z argumentem krótkim bezpośrednim.
  Rozkaz zajmuje jedno słowo maszynowe. Poszczególne pozycje tego słowa mają wówczas następujące znaczenie:
  poz. 0 - 5 - kod operacji,
  poz. 6 - znak argumentu 0 - argument krótki dodatni
  1 - argument krótki ujemny,
  poz. 9 - 9 (pole A) - zawiera numer rejestru uniwersalnego R1 - R7,
  poz. 10 - 15 - zawiera wartość argumentu krótkiego.
- rozkazy z argumentem bajtowym bezpośrednim.
  Rozkazy zajmują jedno słowo maszynowe, a poszczególne pozycje tego słowa mają wówczas następujące znaczenie:
  poz. 0 - 5 - kod operacji,
  poz. 6 - 7 - przedłużenie kodu operacji,
  poz. 8 - 15 - argument bezpośredni.
  Na rysunku 2.6 przedstawiono w sposób graficzny postaci rozkazów.

Cykl wykonania rozkazu rozpoczyna się pobraniem rozkazu z pamięci operacyjnej, przy czym adres miejsca pamięci pierwszego słowa rozkazu określa rejestr IC. Liczba pobranych słów zależy od długości rozkazu. Następnie określana jest efektywność rozkazu. Jest on nieefektywny jeżeli jest rozkazem nieprawidłowym (rozpoznawany ustawieniem wskaźnika przeskoku P na zero). Spowodowane to może być błędnym kodem operacji, użyciem po raz czwarty rozkazu "modyfikuj", nielegalnym rozkazem wykrytym w programie. Wykrycie nielegalnego rozkazu powoduje wyzerowanie modyfikatora (modyfikator MOD stanowi argument rozkazu modyfikuj), wpisanie jedynki na 6 pozycję rejestru zgłoszeń przerwań i natychmiastowe zakończenie rozkazu. Rejestr IC zawiera adres pierwszego



Rys. 2.6. Postacie rozkazów
Źródło: /4/.

słowa tego rozkazu zwiększony o 1. W przypadku gdy wskaźnik przeskoku P przyjmuje wartość 1 następuje wpisanie zera do wskaźnika P, wyzerowanie modyfikatora oraz natychmiastowe zakończenie rozkazu. Rejestr IC zawiera adres pierwszego słowa następnego rozkazu.

W trakcie wykonywania rozkazów wyznaczany jest argument efektywny rozkazu (N). Jest on zawsze liczbą 16-bitową powstałą z argumentu pierwotnego na drodze modyfikacji. Argumentem pierwotnym jest:

- argument normalny to jest zawartość rejestru uniwersalnego, którego adres zawiera pole C słowa rozkazowego lub gdy C = 0 zawartość następnego słowa za słowem podstawowym rozkazu,
- argumentem krótkim, to jest 7-bitową liczbą umieszczoną bezpośrednio w rozkazie z argumentem krótkim bezpośrednim,

- argumentem bajtowym bezpośrednim zajmującym pozycje 8 - 15 słowa podstawowego rozkazu.

Istnieją trzy rodzaje modyfikacji argumentu pierwotnego:

- premodyfikacja,
- B - modyfikacja,
- D - modyfikacja.

Premodyfikacja możliwa dla każdego argumentu pierwotnego dopuszczana jest po wcześniejszym użyciu rozkazu modyfikuj, przy czym możliwe jest użycie co najwyżej trzech rozkazów "modyfikuj" przed danym rozkazem. Argument efektywny rozkazu "modyfikuj" nazywany jest modyfikatorem. Wykonanie tego typu modyfikacji polega na dodaniu do argumentu pierwotnego modyfikatora.

B - modyfikacja możliwa jest w rozkazach z argumentem normalnym, w którym pole B zawiera numer rejestru (R1-R7) i nie stanowi przedłużenia kodu operacji. Tego typu modyfikacja polega na dodaniu do argumentu zawartości rejestru wskazanego przez pole B rozkazu.

D - modyfikacja możliwa jest w rozkazach z argumentem normalnym, jeżeli pole D = 1. Ten typ modyfikacji polega na tym, że uzyskany w wyniku poprzednich operacji modyfikuj argument traktowany jest jako adres argumentu efektywnego, np. w wyniku B - modyfikacji uzyskano argument efektywny 200. Gdyby nie było teraz D - modyfikacji argumentem efektywnym byłoby 200, natomiast w wyniku D - modyfikacji argumentem efektywnym jest wartość znajdująca się w komórce o adresie 200. Można podać ogólne wzory na obliczanie argumentu efektywnego:

$N = R(C) + M + MOD + R(B)$ gdy $D = 0$

$N = S(R(C) + M + MOD + R(B))$ gdy $D = 1$

gdzie:

N - argument efektywny,

R(C) - zawartość rejestru o numerze podanym w polu C rozkazu,

M - argument bezpośredni (gdy C = 0),

MOD - argument efektywny rozkazu "modyfikuj",

R(B) - zawartość rejestru o numerze podanym w polu B rozkazu,

S(...) - zawartość miejsca pamięci o adresie podanym w nawiasie.

Po wyznaczeniu argumentu efektywnego następuje przejście do wykonania operacji określonej kodem rozkazu. Rozkazy mogą być dwuargumentowe, jednoargumentowe i bezargumentowe. Pierwszym argumentem operacji jest zawsze zawartość rejestru uniwersalnego wskazanego polem A rozkazu lub zawartość komórki pamięci operacyjnej. Drugim argumentem jest argument efektywny rozkazu lub zawartość rejestru uniwersalnego wskazanego polem C rozkazu. Rozkazy jednoargumentowe mogą mieć tylko pierwszy lub drugi argument.

2.5. Kanały transmisji

Kanały transmisji zwane również kanałami wejścia/wyjścia służą do komunikowania się procesora systemu MERA-400 z urządzeniami zewnętrznymi i pamięciami zewnętrznymi. Są one dołączone do interfejsu systemu. Dostęp do centralnej szyny informacyjnej kanały otrzymują po wygenerowaniu sygnału zezwolenia przez elementarny układ rezerwacji (rys. 2.2). Ponieważ w systemie MERA-400 kanały traktowane są jako urządzenia aktywne, dlatego same mogą żądać dostępu do interfejsu. W minikomputerze wyróżnia się następujące transmisje dotyczące kanałów:

procesor - kanał		dotyczy przesłania, pobrania,
kanał - procesor		dotyczy przerwania,
kanał - pamięć		dotyczy zapisu, odczytu.

W maksymalnej konfiguracji sprzętowej w systemie MERA-400 zainstalować można 16 różnego typu kanałów. Standardowe wyposażenie maszyny zawiera jedynie kanał znakowy i pamięciowy. Każdy z podłączonych kanałów posiada swój fizyczny numer (od 0 do 15). Typowo kanał znakowy posiada numer 15, a kanał pamięciowy numer 4.

Podłączenie do systemu dodatkowego kanału np. znakowego wymaga

przypisania mu numeru innego, niż numer już zarezerwowany. W kanale znakowym do jednostki centralnej podłączone są urządzenia zewnętrzne pracujące wolno ("start-stopowo"). Dla urządzeń tych w czasie transmisji procesor ciągle musi być zatrudniony pobieraniem rozkazu "czytaj-pisz". W kanale pamięciowym pracują urządzenia zewnętrzne szybkie. W przypadku tych urządzeń procesor jedynie inicjuje transmisję, która następnie prowadzona jest autonomicznie.

Kanał znakowy umożliwia podłączenie co najwyżej ośmiu urządzeń zewnętrznych. Każde urządzenie zewnętrzne posiada swoją jednostkę sterującą z pomocą której komunikuje się z kanałem. Praktycznie górna szybkość transmisji w kanale znakowym nie przekracza 20 tys. znaków na sekundę (przy pracy z kontrolą systemu operacyjnego). Dane przesyłane są znakami 8-bitowymi, słowami 16-bitowymi lub dowolnym formatem zapisu informacji nie przekraczającym 16 bitów. Każde urządzenie zewnętrzne podłączone do kanału posiada swój fizyczny numer (od 0 do 7). W tablicy 2.1 przedstawiono przykładowe podłączenie urządzeń zewnętrznych do jednostki centralnej minikomputera poprzez kanały znakowe o numerach 14 i 15.

Tablica 2.1

Numery urządzeń zewnętrznych w kanałach znakowych

kanał 15		kanał 14	
urządzenie	nr urządzenia	urządzenie	nr urządzenia
DZM-180	7	-	7
PT-105 S	6	-	6
CT 2100	5	-	5
Mera 9475	4	-	4
PSPD-90	3	-	3
DZM-180-KSR	2	-	2
DZM-180-KSR	1	FA	1
-	0	FX	0

Oznaczenia urządzeń tablicy 2.1:

DZM-180 - drukarka mozaikowo-znakowa,
PT-105 S - perforator tasiemki papierowej,
CT 2100 - czytnik tasiemki papierowej,
Mera 9475 - monitor ekranowy z klawiaturą,
PSPD-90 - Programowana Stacja Gromadzenia i Przetwarzania Danych,
DZM-180-KSR- konsola operatorska (drukarka mozaikowo-znakowa z klawiaturą)
FA, FX - jednostki pamięci na dyskach elastycznych (dwie komory dyskowe urządzenia SP 45 DE).

Źródło: Opracowanie własne.

Kanał pamięciowy może sterować pracą ośmiu urządzeń zewnętrznych. Maksymalna szybkość transmisji danych poprzez ten kanał nie przekracza 500 tys. 16-bitowych słów w ciągu sekundy. Kanał pamięciowy jest kanałem selektorowym i może prowadzić transmisję tylko z jednym urządzeniem. Standardowo w kanale pamięciowym pracuje jedna stacja pamięci zewnętrznej na dyskach sztywnych typu Mera-9425.

2.6. Pamięci zewnętrzne

W systemie MERA-400 standardowym urządzeniem pamięci zewnętrznej jest stacja dysków sztywnych. Coraz częściej stosuje się również pamięci na dyskach elastycznych. Do minikomputera można podłączyć również pamięci taśmowe typu PT-305 oraz pamięć kasetową typu PK-1. Ze względu na największą popularność dysków sztywnych i elastycznych poniżej dokonano charakterystyki tych typów urządzeń.

2.6.1. D y s k i s z t y w n e

Do podstawowego wyposażenia MERY-400 należy pamięć zewnętrzna typu Mera-9425. Jest to pamięć dysków sztywnych o krótkim czasie dostępu służąca do przechowywania dużych zbiorów danych i programów. Jest ona podłączona do jednostki centralnej poprzez kanał pamięciowy o numerze 4. Każda stacja dysków sztywnych wyposażona jest w dwa talerze dyskowe (stały i wymienny). Talerz stały zwany systemowym umieszczony jest wewnątrz obudowy stacji dyskowej. Talerz wymienny znajduje się w kasecie z tworzywa sztucznego. Powierzchnie dysków pokryte są warstwą tlenku magnetycznego. Zapis i odczyt informacji możliwy jest z obu stron talerza, tak więc stacja dysków sztywnych posiada cztery głowice. W czasie pracy dysk wiruje z szybkością 2400 obrotów na minutę.

Na powierzchni dysku możliwy jest zapis informacji na 204 ścieżkach (w tym 4 zapasowe). Każda ścieżka dzieli się na 24 sektory. Nominalna

pojemność całej stacji pamięci dysków sztywnych wynosi 50000 000 bitów. Programowo w jednym sektorze na ścieżce można zapisać 512 bajtów informacji.

Powierzchnie dysków wymiennego i stałego są systemowo dzielone na sekcje dyskowe. Podział ten zależy od systemu operacyjnego. System operacyjny SOM 3 bez procesorów systemowych FMC i COS posiada sztywny (stały) podział na sekcje dyskowe obu talerzy. Procesory systemowe FMC i COS pozwalają na dokonywanie dowolnego podziału talerza wymiennego, przy stałym podziale talerza stałego. W tablicy 2.2 przedstawiono podział dysku stałego na sekcje dyskowe.

Tablica 2.2

Sekcje dyskowe dysku stałego - podział dla systemu operacyjnego SOM-3 (bez i z procesorami FMC i COS)

Nazwa sekcji	Wykorzystanie sekcji
ASG	do użytku systemu operacyjnego
GRA	do użytku systemu operacyjnego
GOM	do użytku systemu operacyjnego
GLB	systemowa biblioteka binarna (strumień LB)
GXB	biblioteka binarna procesorów systemowych (strumień LBM)
GMC	biblioteka źródłowa (strumień MC)
ASA	obszar roboczy (strumienie BI,BO, SCA)
ASB	obszar roboczy (strumień SO)
ASC	zbiór notatnikowy (strumień SC)
ASW	obszar roboczy (strumień WRK)
AJC	do użytku wewnętrznego JOB CONTROLA (strumień JC)
AJW	do użytku wewnętrznego JOB CONTROLA (strumień JW)

Uwaga: sekcje dyskowe ASA, ASB, ASC łącznie tworzą obszar o nazwie ASL.
Źródło: /4/.

Podane w tablicy 2.2 w nawiasach nazwy są nazwami logicznych strumieni przy pomocy których można (standardowo) komunikować się z podanymi sekcjami dyskowymi.

Dyski wymienne pracujące w systemie operacyjnym SOM-3 bez procesorów

FMC i COS mają stały podział powierzchni użytkowej na następujące sekcje:

- 4 sekcje 10-cylindrowe o nazwach AMi (i = 1,2,3,4),
- 4 sekcje 40-cylindrowe o nazwach ALi (i = 1,2,3,4),

lub podział drugi:

- 4 sekcje 50-cylindrowe o nazwach ADi (i = 1,2,3,4).

W przypadku podziału drugiego możliwe jest korzystanie z sekcji o nazwie ADO utworzonej z połączenia obszarów AD1 i AD2.

Procesory systemowe FMC i COS systemu operacyjnego SOM-3 pozwalają na dowolne nazewnictwo zarówno samej kasety dyskowej jak i sekcji dyskowych. Sekcje te mogą mieć dowolną wielkość i dowolne nazwy nadawane przez użytkownika.

System operacyjny SOM-3.P zapewnia zupełnie inną organizację zarówno dysku stałego jak i wymiennego. W tablicy 2.3 przedstawiono podział dysku stałego na sekcje dyskowe.

Kaseta dysku wymiennego posiada następującą budowę. Począwszy od trzeciego cylindra następnych 40 cylindrów dzielonych może być na jeden ze sposobów:

AMi - podział po 5 cylindrów (i = 1,...,8),

BMi - podział po 10 cylindrów (i = 1,2,3,4),

CMi - podział po 20 cylindrów (i = 1,2),

DM1 - sekcja 40 cylindrowa.

Pozostałe cylindry na dysku wymiennym przeznaczone są na organizację zbiorów systemu plikowego.

Tablica 2.3

Sekcje dyskowe dysku stałego - podział dla systemu operacyjnego SOM 3.P

Nazwa sekcji	Wykorzystanie sekcji
GXB	biblioteka programów (procesorów systemowych)
GLB	biblioteka podprogramów (strumień LB)
GMC	biblioteka makrodefinicji makroprocesora MAC (strumień MC)
GAS	biblioteka makrodefinicji asemblera GAS
AJC	biblioteka makrodefinicji interpretera komend IK (strumień JC)
SYi	dla systemu operacyjnego (i = A, B, C, D)
OMi	nakładki systemu operacyjnego (i = A, B, C, D)
ASA	sekcja plików binarnych (strumienie BI, BO)
ASB	sekcja plików źródłowych (strumienie SI, SO)
AJW	sekcja robocza makroprocesora IK
ASC	sekcja robocza
PRO	sekcja modułu głównego
SUB	sekcja biblioteki chwilowej

Źródło: /9/.

2.6.2. Dyski elastyczne

Jednostki pamięci na dyskach elastycznych (typu SP 45 DE) podłączone są do jednostki centralnej MERY-400 w kanale znakowym. Urządzenie to posiada cztery komory dyskowe, które połączone są w dwa moduły (po dwie komory). Pracą każdego modułu steruje jedna jednostka sterująca.

Dysk elastyczny jest krążkiem celulojdowej folii na powierzchni której z obu stron naniesiony jest materiał magnetyczny. Powierzchnia dyskietki chroniona jest przed mechanicznymi uszkodzeniami przez sztywną kopertę. Stosowane w systemie MERA-400 dyski elastyczne są dyskami 8-calowymi (wymiar przekątnej koperty).

W trakcie pracy możliwy jest zapis i odczyt informacji na jednej stronie dyskietki. Dostęp do danych na drugiej stronie możliwy jest po przełożeniu dyskietki drugą stroną do komory dyskowej. Wynika to z faktu,

że każda komora dyskowa posiada tylko jedną głowicę zapisu/odczytu.

Podczas pracy dysk elastyczny obraca się wewnątrz nieruchomej koperty z szybkością 360 obrotów na minutę. Średni czas dostępu do danych wynosi 205 msek.

Dyskietka posiada 77 ścieżek, z których każda podzielona jest na 26 sektorów. Podział ścieżki na sektory nie jest stały, a zależy od sposobu formatowania. Sektory leżące fizycznie obok siebie na powierzchni dysku nie muszą mieć kolejnych logicznych numerów. Dyskietkę można formatować na 13 sposobów (kolejność sektorów od 01 do 13). Ścieżki na dysku elastycznym numerowane są od 0 do 76.

W jednym sektorze użytkownik może zapisać 128 bajtów informacji. Pojemność jednej strony dyskietki odpowiada pojemności standardowej roboczej sekcji dyskowej dysku stałego np. ASB.

2.7. Urządzenia wejścia/wyjścia

Urządzenia wejścia/wyjścia w systemie MERA-400 pracują w kanale znakowym. Wszystkie urządzenia posiadają swoje nazwy dzięki którym system operacyjny identyfikuje je. W tablicy 2.4 zestawiono nazwy typowych urządzeń zewnętrznych dla systemów operacyjnych SOM-3 i SOM-3.P.

Standardowym urządzeniem do wprowadzania informacji w systemie MERA-400 jest czytnik tasiemki papierowej typu CT-2100. Jest to urządzenie przystosowane do odczytu informacji zapisanej na tasiemce 5- lub 8-ścieżkowej. W standardzie MERY-400 tasiemka jest 8-ścieżkowa. Informacha na tasiemce zajmuje 7 bitów (ścieżek), a bit 8 traktowany jest jako bit parzystości. Czytnik tasiemki perforowanej zamienia informację zapisaną mechanicznie na tasiemce perforowanej na informację cyfrową, która jest dogodna do dalszej obróbki przez maszynę cyfrową.

Perforator tasiemki papierowej typu DT-105 S jest urządzeniem służącym do przekształcania informacji cyfrowej na odpowiednie kombinacje dziurek w tasiemce. DT-105 S umożliwia na drukowanie tasiemki 5- i 8-

ścieżkowej. Dziurkowanie jest sterowane z procesora jednostki centralnej.

Tablica 2.4

Urządzenia zewnętrzne pracujące w kanale znakowym

Urządzenie	Nazwa urządzenia w systemie		
	System SOM-3		System SOM-3.P
	Bez procesorów FMC i COS	Z procesorami FMC i COS	
DZM-180	CSL	CSL	CSL
PT-105 S	PP	PP	PP
CT-2100	PR	PR	PR
DZM-180-KSR:			
klawiatura	CK	CK	CKi (i=A,B..)
drukarka	CS	CS	CSi (i=A,B..)
Mera 9475:			
klawiatura	CK	CK	CKi (i=A,B..)
monitor	CS	CS	CSi (i=A,B..)
PSPD-90:			
klawiatura	-	-	CKi (i=A,B..)
monitor	-	-	CSi (i=A,B..)
SP 45 DE	-	F0,F1,F2,F3	FA0,FA1,FA2, FA3 lub FB0,FB1,FB2, FB3

Uwaga: System operacyjny komunikuje się oddzielnie z każdym dyskiem elastycznym i dlatego wszystkie komory dyskowe posiadają inne nazwy. Nazwy FBi (i = 1,2,3,4) odpowiadają stronie B dyskietki. System operacyjny SOM-3.P jest systemem wielodostępnym i dlatego konieczne jest rozróżnienie kilku urządzeń tego samego typu. Dlatego po nazwie urządzenie podawana jest dodatkowo litera odpowiadająca fizycznemu umiejscowieniu urządzenia sterującego w kanale znakowym.
Źródło: opracowanie własne.

Drukarka mozaikowo-znakowa typu DZM-180 jest w systemie MERA-400 standardowym urządzeniem służącym do otrzymywania wydruków. Maksymalna

szybkość wydruku wynosi 18 znaków/sek. Podstawowa wersja DZM-180 posiada możliwość wydruku 64 różnych znaków. Ich kody zapamiętane są w dwóch pamięciach stałych typu ROM. Istnieje możliwość dołączenia dodatkowych dwóch ROM-ów, co daje zestaw 128 znaków. W czasie pracy głowica drukująca przesuwa się przed nieruchomym papierem i tasiemką barwiącą. W trakcie przesuwu w zależności od kodu znaku uruchamiane są igły drukujące. Igieł jest siedem, a każda uruchamiana jest oddzielnym elektromagnesem. Uderzenie pojedynczej igły pozostawia na papierze ślad kropki (punkt). Drukowane znaki powstają jako mozaika kropek i mieszczą się w matrycy o siedmiu wierszach i siedmiu kolumnach. Uruchamianie drukarki polega na przyciśnięciu przełącznika zasilającego i włączeniu klawisza SEL. Niegotowość drukarki do druku sygnalizuje czerwona lampka umieszczona pod przyciskiem HA.

Stan niegotowości drukarki może wystąpić w następujących przypadkach:

- podniesienie osłony górnej głowicy drukującej,
- zadziałanie mikroprzełącznika końca przesuwu głowicy drukującej,
- przeciążenie napięcia zasilającego,
- wykrycie końca papieru.

W systemie MERA-400 urządzeniami służącymi do komunikowania się operatora z systemem operacyjnym są konsole operatorskie typu DZM-180-KSR lub monitory ekranowe u klawiaturą np. typu Mera 9725.

Konsola operatorska DZM-180-KSR jest urządzeniem wyposażonym w drukarkę mozaikowo-znakową typu DZM-180 i układ klawiatury. Klawiatura ma możliwość generowania znaków alfanumerycznych i specjalnych. Znaki generowane są w kodzie ISO-7. W tablicy 2.5 zebrano kody znaków możliwych do generacji na konsoli DZM-180-KSR. Dodatkowo obok klawiatury znajdują się cztery przyciski, których znaczenie jest następujące:

SPEED SELECT - służy do wyboru jednej z dwu prędkości transmisji: przy wyciśniętym przełączniku szybkość transmisji wynosi 300 bodów, a przy wciśniętym szybkość ta zależna jest od prędkości transmisji ustawionej w urządzeniu (V-24),

Tablica 2.5

Kody znaków dostępnych na klawiaturze DZM-180-KSR

	0	1	2	3	4	5	6	7
0	NUL	DEL	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	!
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	↑	n	~
F	SI	US	/	?	O	←	o	=

Źródło: /4/

FULL DUPLEX - klawisz wciśnięty: praca w pełnym duplexie, występuje wyraźne rozgraniczenie informacji wchodzącej i wychodzącej. Z klawiatury dane wysyłane są tylko w linię.

RO - klawisz wciśnięty sprawia, że DZM-180-KSR staje się tylko odbiornikiem (praca jak DZM-180),

ON-LINE - klawisz nie wciśnięty umożliwia pracę lokalną: informacja zapisywana z klawiatury nie jest przesyłana do jednostki centralnej.

Klawisze SHIFT i CTRL nie umieszczone w tablicy służą do wyboru znaków z odpowiednich kolumn.

Bez klawiszy SHIFT i CTRL wybierane są znaki z kolumn:

- 3 od wiersza 0 do wiercza B,
- 2 od wiersza C do wiersza F,

- 4 cała,
- 5 cała.

Z klawiszem SHIFT bez klawisza CTRL wybierane są znaki z kolumn:

- 2 od wiersza 0 do wiersza B,
- 3 od wiersza C do wiersza F,
- 6 cała,
- 7 cała.

Z klawiszem CTRL, bez klawisza SHIFT wybierane są znaki z kolumn 0 i 1.

W DZM-180-KSR obok klawiszy klawiatury głównej istnieje blok klawiatury cyfrowej i dodatkowo klawisze 0, SP, HT, LF, CR, ESC, które nie są objęte układem SHIFT i CTRL. Dodatkowy klawisz BREAK (nie opisany w tabeli) służy do przerwania transmisji.

Znaczenie najczęściej wykorzystywanych klawiszy specjalnych jest następujące:

CR - Cariagge Return; po odebraniu kodu tego znaku następuje powrót głowicy na początek wiersza,

LP - LINE Feed; powoduje przesunięcie tekstu o jeden wiersz do góry z jednoczesnym powrotem głowicy na początek wiersza,

BS - Back Space; kod tego znaku powoduje kasowanie ostatniego wprowadzonego znaku,

CAN - Cancel; kod tego znaku powoduje kasowanie ostatniego wprowadzonego wiersza.

Analogiczną funkcję w systemie MERA-400 jaką pełni DZM-180-KSR może spełniać monitor ekranowy z klawiaturą (np. typu Mera 9475). W urządzeniu tym informacja jest wyświetlana na ekranie w 24 wierszach. W wierszu mieści się 80 znaków. Klawiatura przyłączona do monitora nie ma możliwości wyboru znaków z kolumn 6 oraz 7 opisanych w tablicy 2.5. Pozostałe znaki wybiera się identyczne jak dla DZM-180-KSR.

2.8. Stacja PSPD-90 jako terminal w systemie MERA-400

Programowana Stacja Gromadzenia i Przetwarzania Danych typu PSPD-90 jest mikrokomputerem, którego budowę oparto o wykorzystanie możliwości mikroprocesora INTEL 8080. Istnienie pamięci operacyjnej pamięci zewnętrznych na dyskach elastycznych, klawiatury i monitora pokazuje na możliwości autonomicznej pracy tego urządzenia. Ze względu na fakt istnienia złącza transmisji V-24 można PSPD-90 łączyć z innymi maszynami cyfrowymi. Stacja PSPD-90 podłączona jest do jednostki centralnej MERY-400 w kanale znakowym. Pełni ona funkcję analogiczną do konsoli operatorskiej. Transmisja pomiędzy MERA-400 a PSPD-90 (i na odwrót) realizowana jest dzięki programom pracującym na obu maszynach pod systemem SOM-3.P oraz MICRODOS. Programy te mają wspólną nazwę TRANS.

Wykorzystując własne zasoby pamięci zewnętrznych stacji PSPD-90 można korzystając z dodatkowych komend transmisji plików uruchomić taką transmisję między obiema maszynami. Jest to znaczne rozszerzenie funkcji realizowanych na konsoli operatorskiej.

Monitor stacji może wyświetlać informację w 16 wierszach. W jednym wierszu mieszczą się 32 znaki.

Rozbudowana klawiatura stacji, która przystosowana jest głównie do potrzeb systemów PSPD-90, nie jest w całości wykorzystywana przez program transmisji. Wykorzystywane są jedynie następujące klawisze funkcyjne:

- w trakcie wprowadzania komend:
  - CHR - cofanie kursora,
  - SHIFT i CHR - posuw kursora do przodu,
  - FLD - kasowanie zawartości bufora,
  - SHIFT i FLD - kasowanie bufora od kursora do końca.
- w trakcie wprowadzania rekordu, który nie jest komendą:

CHR - kasowanie znaku,

FLD - kasowanie bufora,

≡ - znak nowej linii,

← - ESCAPE.

Niestandardową postać klawiatury PSPD-90 w stosunku do klawiatury DZM-180-KSR sprawia, że kody niektórych znaków przedstawionych w tablicy 2.5 odpowiadają innym znakom opisanym na stacji. Różniące się klawisze przedstawiono w tablicy 2.6.

Tablica 2.6

Odpowiedniki znaków klawiatury DZM-180-KSR na klawiaturze stacji PSPD-90

Klawiatura DZM-180-KSR	Klawiatura PSPD-90
^	Ś
[	Ź
]	Ń
\	Ł
@	Ę
ESC	strzałka w prawo
ER	strzałka w lewo
LF	≡
$	Ą
BS	CHR
CAN	FLD

Źródło: /9/.

3. SYSTEMY OPERACYJNE MINIKOMPUTERA MERA-400

Minikomputery MERA-400 są oferowane przez producenta w podstawowej konfiguracji sprzętowej łącznie z systemem operacyjnym o nazwie SOM-3 w wersji jednozadaniowej. Konfiguracja ta może być przez użytkownika rozbudowana do granic określonych przez architekturę logiczną komputera poprzez powiększenie pamięci operacyjnej, dołączenie dodatkowej stacji pamięci dyskowej i innych urządzeń peryferyjnych. Przykład rozbudowanej konfiguracji minikomputera MERA-400 jest pokazany na rysunku 1 w rozdziale 2. Efektywne wykorzystanie minikomputera w rozszerzonej konfiguracji (zwłaszcza jeżeli wyposażonego w pewną liczbę terminali) automatycznie wymaga zmiany systemu operacyjnego.

Z powodów wymienionych wyżej w skrypcie zostaną omówione dwa systemy operacyjne. Pierwszy z nich jest wspomnianym na wstępie systemem SOM-3 umożliwiającym w pełnym zakresie pracę interakcyjną z jednym tylko użytkownikiem[1] zestawu, który w stosunku do konfiguracji rozszerzonej pokazanej na rysunku 2.1 w rozdziale 2 wykazuje następujące różnice:

1. Pamięć operacyjna ma pojemność tylko 32 K słów szesnastobitowych.
2. W charakterze konsoli operatorskiej występuje drukarka z klawiaturą DZM-180-KSR w miejsce monitora ekranowego Mera 9475.
3. Brak jednostek pamięci SP45DE na dyskach elastycznych.
4. Brak terminali, którymi mogą być drukarki z klawiaturą DZM-180-KSR, lub mikrokomputery PSPD-90.

---

[1] Za wyjątkiem podsystemu programowania w języku BASIC, który jest programem wielodostępnym i pozwala pracować jednocześnie czterem użytkownikom pod warunkiem dołączenia odpowiedniej liczby terminali.

Drugi z omawianych w skrypcie systemów operacyjnych o nazwie SOM 3.P został opracowany w Uniwersytecie Jagiellońskim[2] i ogólnie rzecz biorąc umożliwia eksploatację minikomputera w rozszerzonej konfiguracji z liczbą jednocześnie pracujących terminali nie przekraczającą czterech.

3.1. Nieformalne wprowadzenie w problematykę systemów operacyjnych

Jak do tej pory terminy informatyczne takie jak "system operacyjny", "podział czasu", "wieloprogramowość", czy też "zadanie" nie mają szeroko przyjętych, precyzyjnych definicji z wyjątkiem być może kontekstu studiów teoretycznych, ograniczonych do pewnych niewielkich aspektów systemów liczących. Zamiast tego terminy te oznaczają pewne typy organizacji, funkcje, zachowanie się bądź metody działania. Pamiętając o tym nieformalnie zdefiniowano wiele ważnych terminów powszechnie używanych do opisu funkcjonowania systemów cyfrowych.

Dla potrzeb opisu pracy systemu MERA-400 od strony użytkowej przyjmiemy, że system operacyjny jest to zorganizowany zespół programów, które pełnią rolę pośredniczącą pomiędzy sprzętem a użytkownikami dostarczając tym ostatnim zestawu środków ułatwiających projektowanie, programowanie, uruchamianie i eksploatację programów użytkowych sterując w tym samym czasie przydziałem zasobów dla zapewnienia efektywnego działania systemu cyfrowego jako całości.

W literaturze przedmiotu wymienia się trzy kategorie "czystych" systemów operacyjnych. Każdą z nich można scharakteryzować podając typ możliwego oddziaływania pomiędzy użytkownikiem a jego pracą w systemie oraz tolerancję czasu odpowiedzi systemu:

1. Systemy operacyjne dla przetwarzania wsadowego są to systemy,

---

[2] Własności systemu SOM 3.P opisane w niniejszym podręczniku odnoszą się do wersji wygenerowanej dla konfiguracji minikomputera pokazanej na rysunku 1 w rozdziale 2. Wersje tego systemu wygenerowane dla innych konfiguracji mogą wykazywać pewne różnice w odniesieniu do przedstawionego dalej opisu.

w których prace użytkownika są wprowadzane w postaci sekwencyjnych wsadów przez urządzenia wejścia. Z chwilą wprowadzenia pracy do systemu użytkownik traci z nią jakikolwiek kontakt i nie ma możliwości oddziaływania na proces jej realizacji. W tej sytuacji użytkownik za czas reakcji systemu uważa czas jaki upłynął od momentu w którym dostarczył swoją pracę do chwili odebrania wyników obliczeń. Innymi słowy utożsamia on w tym przypadku czas reakcji systemu z czasem obrotu swojej pracy. Ten ostatni, w zależności od organizacji pracy ośrodka realizującego obliczenia użytkowników w trybie pracy wsadowej, jak i w zależności od czasochłonności samej pracy waha się od kilku minut do kilkunastu godzin.

2. System operacyjny z podziałem czasu jest to system, który równocześnie (w skali czasu zewnętrznego) obsługuje prace obliczeniowe wielu użytkowników na swoje obliczenia za pośrednictwem tzw. terminali czyli zewnętrznych urządzeń znakowych typu monitor ekranowy z klawiaturą lub ulepszony dalekopis. Efekt jednoczesnego dostępu osiągnięto przez podział czasu procesora oraz innych zasobów pomiędzy wielu użytkowników w sposób gwarantujący odpowiedź na pytanie użytkownika w czasie kilku sekund.

3. System operacyjny dla działania w czasie rzeczywistym jest to system, który obsługuje bezpośrednio proces zewnętrzny, mający ściśle określone ograniczenia czasowe na odpowiedź. Sygnały przerwań z procesów zewnętrznych kierują działaniem systemu. Jeśli nie są one obsłużone w czasie wymaganym przez proces zewnętrzny, to podlegają degradacji albo przekształceniu. Systemy tego typu są często projektowane dla szczególnych zastosowań takich jak przykładowo sterowanie procesami technologicznymi.

Systemy operacyjne współczesnych maszyn cyfrowych najczęściej łączą w sobie cechy wymienionych wyżej kategorii. Przykładowo systemy z podziałem czasu pracujące w trybie interakcyjnym z wieloma użytkownikami wykonują zadania wsadowe jako tak zwane prace drugoplanowe. Polega to

na tym, że jeżeli w danym momencie czasu żaden proces użytkownika pracującego w trybie interakcyjnym nie angażuje procesora maszyny, to system operacyjny uaktywnia proces wykonania jednej z prac wsadowych znajdujących się w systemie w stanie oczekiwania na wykonanie. Pozwala to w ewidentny sposób zwiększyć efektywność wykorzystywania zasobów systemu na drodze eliminacji ewentualnych przestojów.

Powszechną cechą współczesnych systemów cyfrowych jest wieloprogramowość. Pierwsze systemy wieloprogramowe pojawiły się na przełomie lat pięćdziesiątych i sześćdziesiątych. W okresie tym wprowadzono do użytku wiele wynalazków z dziedziny sprzętu, które stymulowały postęp w systemach operacyjnych. Najważniejszą, z punktu widzenia naszych rozważań, innowacją sprzętową był kanał danych - prymitywna maszyna cyfrowa wyposażona we własny zestaw rozkazów, rejestry i sterowanie, zarządzająca komunikacją i transmisją informacji pomiędzy jednostką centralną a urządzeniami zewnętrznymi. Jednostka centralna wysyła żądanie transmisji do kanału. Kanał wykonuje i steruje transmisją informacji w sposób asynchroniczny, współbieżnie z ciągłą pracą jednostki centralnej. Jednostka centralna i kanał współdzielą pamięć operacyjną zawierającą ich programy. Początkowo tylko jednostka centralna mogła sprawdzać stan kanału, ale szybko przekonano się, że nowa architektura będzie działać znacznie efektywniej, gdy również kanał będzie mógł przerywać proces przetwarzania realizowany przez jednostkę centralną celem podania wiadomości, najczęściej sygnału zakończenia zleconej uprzednio kanałowi obsługi transmisji informacji. W ten sposób na arenę informatyczną wkroczył podstawowy we współczesnych maszynach cyfrowych mechanizm przerwań. Najogólniej rzecz biorąc przerwanie polega na wytworzeniu, przy zaistnieniu pewnych zdarzeń wymagających bezpośredniej reakcji jednostki centralnej, sygnału, który zawiesza program realizowany w danym momencie przez jednostkę centralną i powoduje przejście do realizacji specjalnej procedury noszącej nazwę obsługi przerwania. Sygnały przerwań mogą być generowane zarówno przez sprzęt (np. układ zasilania,

zegar maszyny czy kanał) jak i realizowane programy (np. zgłoszenie potrzeby realizacji transmisji informacji przez kanał).

Reasumując można stwierdzić, że przed wprowadzeniem kanałów, maszyny realizowały wszystkie procesy przetwarzania sekwencyjnie. Kanały uwalniając procesor maszyny od konieczności obsługi transmisji informacji udostępniły go innym procesom przetwarzania informacji. W ten sposób pojawiła się nowa jakość w pracy maszyny cyfrowej - współbieżna praca jej składowych sprzętowych. Mechanizmy przerwań umożliwiły między innymi synchronizację współbieżnych procesów. O operacjach wejścia /wyjścia realizowanych przez kanał i prowadzonych w tym samym czasie przez procesor obliczeniach można powiedzieć, że są one współbieżne sprzętowo. Na bazie sprzętowej współbieżności pracy kanału z pracą jednostki centralnej wprowadzono współbieżność logiczną procesów w maszynie cyfrowej. Współbieżność logiczna jest to pojęciowa koncepcja współbieżności wynikła z braku możliwości rozróżnienia przez człowieka czy dany proces jest realizowany przez maszynę sekwencyjnie czy też współbieżnie. Logiczna współbieżność leży u podstaw zasad działania wieloprogramowych systemów operacyjnych. W pamięci operacyjnej maszyny cyfrowej pracującej pod kontrolą wieloprogramowego systemu operacyjnego znajduje się kilka programów użytkowych gotowych do wykonania. W danym momencie czasu tylko jeden z nich wykorzystuje procesor do prowadzenia obliczeń. Pozostałe z nich znajdują się w stanie oczekiwania. Wśród tych ostatnich można wyróżnić programy gotowe do wykonania, to znaczy takie, które oczekują na przydzielenie im procesora oraz programy niegotowe, to jest takie, które oczekują na zajście określonego zdarzenia, na przykład zakończenia wczytywania przez kanał danych niezbędnych do dalszej realizacji programu. Z punktu widzenia użytkownika program angażujący w danym momencie procesor i program znajdujący się w fazie oczekiwania na zakończenie przez kanał potrzebnych danych są wykonywane przez maszynę równocześnie. Z chwilą pojawienia się sygnału przerwania, aktualnie wykonywany program przez jednostkę centralną zostaje

przerwany i przeniesiony w stan oczekiwania, a po obsłudze przerwania podejmuje pracę systemową procedura zarządzająca kolejnością wykonywania programów w maszynie cyfrowej. Najczęściej procedura ta pracuje w ten sposób, że przydziela procesor temu spośród programów oczekujących gotowych do wykonania, który ma najwyższy priorytet. Priorytet jest pewną liczbą całkowitą z określonego w danym systemie przedziału wartości i jest traktowany przez system jako atrybut ważności programu. Każda praca w systemie ma określony priorytet. Sposób określania priorytetu jest związany z założoną w określonym systemie strategią obsługi. Różne systemy, a nawet różne instalacje tego samego systemu mogą się różnić realizowanymi strategiami obsługi. Najprostszą zarówno w sensie koncepcyjnym jak i w realizacji jest strategia FIFO (First Input First Output), która polega na tym, że każdy program wstępnie otrzymuje ten sam priorytet, a następnie, co ustalony kwant czasu system zwiększa wszystkim programom zarejestrowanym w systemie priorytet o tę samą wartość. Zgodnie z tą strategią najwyższy priorytet będą miały, a tym samym będą najbardziej uprzywilejowane programy najdłużej przebywające w systemie. Zwykle jednakże procedury planujące kolejność prac w systemie realizują znacznie bardziej złożone strategie obsługi uwzględniając przy ustalaniu priorytetu zadania w systemie takie elementy jak zapotrzebowanie na pamięć operacyjną i czas centralnego procesora, przy czym zależność priorytetu od tych parametrów jest odwrotnie proporcjonalna to znaczy im mniejsza pamięć i krótszy czas procesora są niezbędne do realizacji określonego programu, tym wyższy jest jego priorytet.

Przedstawiony wyżej schemat pracy wieloprogramowej jest charakterystyczny dla wieloprogramowych systemów operacyjnych przetwarzania wsadowego. Strategia pracy wieloprogramowej systemu z podziałem czasu opiera się na zasadzie równoprawności wszystkich użytkowników w systemie. Użytkownicy są dołączeni do systemu za pomocą terminali typu monitor ekranowy lub drukarka z klawiaturą. W systemie użytkownik jest repre-

zentowany przez proces obsługi jego terminalu. Liczba terminali, a co za tym idzie liczba jednocześnie, w sensie logicznym, pracujących z systemem użytkowników jest ustalana w fazie generacji systemu dla określonej konfiguracji sprzętowej. W danym momencie czasu tylko jeden proces użytkownika może dysponować procesorem maszyny. Pozostałe znajdują się w stanie oczekiwania. Strategia obsługi polega w tym przypadku na tym, że poszczególne procesy przechodzą kolejno w stan aktywny (to znaczy otrzymują do dyspozycji procesor maszyny) na ściśle określony kwant czasu jednakowy dla wszystkich procesów.

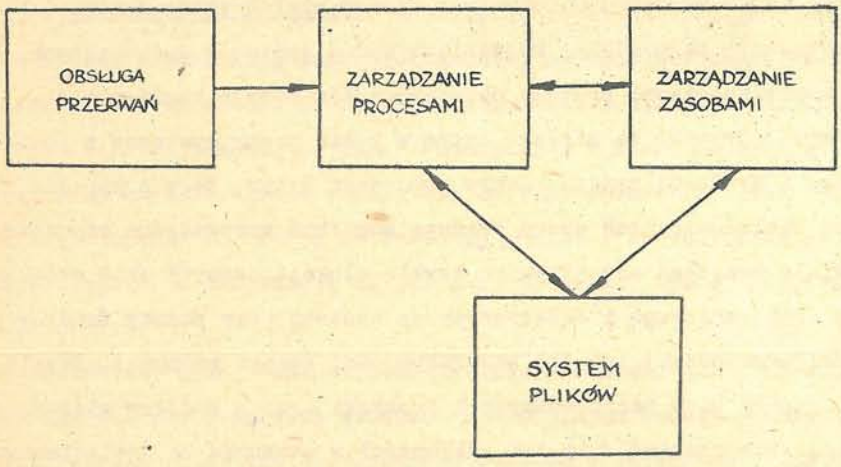
Domykając problematykę systemów wieloprogramowych od strony podstawowych pojęć warto może jeszcze objaśnić jeden termin, który nader często jest utożsamiany z wieloprogramowością: wieloprocesorowość. Termin ten odnosi się do konfiguracji sprzętowej i oznacza kompleks sprzętu zawierający więcej aniżeli jeden z niezależnych procesorów pracujących ze wspólną pamięcią operacyjną. Procesy przetwarzania w zestawie wieloprocesorowym mogą być współbieżne na poziomie sprzętowym i dla podkreślenia tego faktu przyjęło się pracę wieloprogramową takich zestawów sprzętowych określać mianem wieloprzetwarzania.

Współcześnie najbardziej rozwiniętą formę systemów operacyjnych stanowią wielodostępne systemy operacyjne dla dużych konfiguracji sprzętowych. Systemy te niejako łączą w sobie cechy systemów z podziałem czasu i systemów przetwarzania wsadowego z tym, że w przypadku systemów wielodostępnych prace wsadowe mogą być wprowadzane równocześnie z wielu urządzeń zewnętrznych zwykle zlokalizowanych poza obrębem ośrodka obliczeniowego i dołączonych do systemu przy pomocy środków łączności przewodowej lub też bezprzewodowej (drogą radiową). Zwykle są to czytniki kart perforowanych i drukarki wraz z modułem sterującym ich pracą. Dla podkreślenia ich odrębności w stosunku do analogicznych urządzeń zlokalizowanych w obrębie ośrodka obliczeniowego przyjęło się je określać mianem terminali wsadowych, a prace realizowane przy ich udziale - zdalnym przetwarzaniem wsadowym. W tej konwencji uprzednio

wymienione terminale typu monitor ekranowy z klawiaturą można określić mianem terminali interakcyjnych albo dialogowych z uwagi na sposób użytkowania systemu cyfrowego za ich pośrednictwem. W chwili obecnej, w dobie burzliwego rozwoju mikroelektroniki i mikroinformatyki coraz częściej w charakterze terminali dużych systemów cyfrowych są wykorzystywane mini- oraz mikrokomputery, które tworzą klasę tzw. terminali inteligentnych. Cechą charakterystyczną terminali inteligentnych jest to, w zakresie zadań użytkowników nie przekraczających ich możliwości przetwarzania można je uważać za niezależnie działające systemy cyfrowe, a w przeciwnym przypadku angażują do realizacji pracy użytkownika centralny komputer, do którego są dołączone, przy czym mogą pełnić zarówno funkcję terminali interakcyjnych jak i obsługiwać zdalne przetwarzanie wsadowe.

3.1.1. Podstawowe elementy wieloprogramowego systemu operacyjnego

Na rysunku 3.1 pokazano podstawowe elementy oprogramowania wieloprogramowego systemu operacyjnego. Konkretny system może nie mieć dokład-



Rys. 3.1. Podstawowe elementy oprogramowania wieloprogramowego systemu
Źródło: /8/.

nie takiej jak pokazano organizacji, jednakże funkcje każdego z bloków można odnaleźć w funkcjach innych modułów systemu.

Centralnym zadaniem wieloprogramowego systemu operacyjnego jest zarządzanie procesami - tworzenie, usuwanie. współpraca, sterowanie oraz szeregowanie procesów. Jak już było mówione, system może być bardzo wrażliwy na zastosowaną strategię szeregowania (obsługi), tzn. algorytmy decydujące, który z wielu procesów znajdujących się w gotowości do wykonania będzie kontynuowany przez przydzielenie mu jednostki centralnej. Przydzielanie wszystkich innych zasobów jest zadaniem systemu zarządzania zasobami, który administruje takimi kluczowymi zasobami jak przestrzeń pamięci operacyjnej i pomocniczej. Podobnie jak w poprzednim przypadku efektywność pracy całego systemu jest silnie uzależniona od wybranej strategii przydzielania zasobów. Trzecim elementem jest system plików, który odpowiada za tworzenie, aktualizację oraz wyszukiwanie informacji w pamięci pomocniczej. Wymienione trzy składowe są ściśle uzależnione i komunikują się między sobą. Przykładowo procesy zarządzania zasobami muszą współpracować z systemem plików w celu sterowania zarządzaniem pamięcią pomocniczą lub przydzielaniem buforów. Wreszcie, elementy te, w przypadku ogólnym, są aktywowane przez zbiór podstawowych programów obsługi przerwań, gdyż utworzenie, zakończenie, przerwanie i aktywowanie procesu jest zazwyczaj następstwem wystąpienia określonego sygnału przerwania.

Nie wszystkie programy składające się na system operacyjny przebywają stale w pamięci operacyjnej. Niektóre z nich umieszczone są w bibliotece systemowej w pamięci pomocniczej (najczęściej dyskowej) i są ściągane do pamięci przez rezydentną część systemu operacyjnego nazywaną ze względów historycznych jądrem systemu. Rozwiązanie takie jest podyktowane chęcią minimalizacji pamięci operacyjnej zajmowanej przez programy systemu operacyjnego, a tym samym niedostępnej dla programów użytkownika. O tym, które programy systemu operacyjne mają status rezydentnych a które nie, rozstrzygają parametry instalacyjne systemu.

Można jednak wskazać takie, które zawsze wchodzą w skład jądra systemu. Są to: zbiór wielodostępnych procedur definiujących operacje elementarne (tzw. ekstrakodów) oraz programy zarządzające procesami, zasobami systemu oraz operacjami wejścia/wyjścia.

Z punktu widzenia użytkownika system operacyjny można uważać za poszerzenie sprzętu. Użytkownik nie ma bowiem możliwości rozróżnienia, które operacje są wykonywane przez rzeczywisty sprzęt, a które są efektem działania programów i procedur systemowych. Taką maszynę "pozorną" widzianą oczyma użytkownika, a składającą się w rzeczywistości z dwóch części - rzeczywistego sprzętu oraz oprogramowania pośredniczącego pomiędzy zleceniami użytkownika a działaniem sprzętu, przyjęło się w literaturze określać mianem maszyny wirtualnej. Z tego ostatniego stwierdzenia wynika, że użytkownik pracujący w określonym systemie cyfrowym ma do dyspozycji pewną liczbę maszyn wirtualnych. Przykładowo, użytkownik pracujący w systemie MERA-400 pod kontrolą podsystemu programowania w języku BASIC może uważać, że pracuje z maszyną, której językiem programowania jest język BASIC. Tenże sam użytkownik chcąc uruchamiać swoje programy napisane w języku Fortran ma do dyspozycji odpowiednie oprogramowanie systemowe, które pozwoli mu "przekształcić" system cyfrowy MERA-400 w maszynę wirtualną "rozumiejącą" język Fortran.

Patrząc zatem na system operacyjny od strony użytkownika można stwierdzić, że podstawową funkcją systemu operacyjnego jako całości jest prawidłowe odwzorowanie (translacja) działania maszyny wirtualnej, którą on definiuje na działanie maszyny rzeczywistej, tzn. sprzętu realnie istniejącego w określonej konfiguracji. Maszyna wirtualna zdefiniowana przez działanie programów systemu operacyjnego jest reprezentowana na zewnątrz przez tzw. język zleceń stanowiący podstawowe narzędzie komunikowania się użytkownika z systemem.

3.1.2. Współpraca użytkownika z systemem

Języki komunikowania się człowieka z systemami operacyjnymi nie mają przyjętego w literaturze jednolitego nazewnictwa. W odniesieniu do nich używa się takich określeń jak: języki zleceń (ang. Command and Control Languages), języki dyrektyw albo języki komend (ang. Command Languages), czy też języki sterowania pracami (ang. Job Control Languages).

Użytkownicy systemu stosują język zleceń do przekazywania poleceń i opisu swoich prac systemowi operacyjnemu. Operatorzy systemu mają do dyspozycji analogiczny język, który pozwala im sterować pracą systemu, otrzymywać informacje o stanie zasobów sprzętowych i programowych oraz reagować na komunikaty i błędy sygnalizowane przez system. Zatem kolejnym elementem każdego systemu operacyjnego musi być interpreter języka zleceń, który analizuje kolejne instrukcje i aktywuje odpowiednie moduły systemu. Historycznie język zleceń użytkownika ma swój rodowód w języku komunikowania się operatora z systemem operacyjnym. We wczesnej fazie rozwoju systemów operacyjnych rolę interpretera wymagań i poleceń użytkownika pod adresem systemu pełnił operator (albo użytkownik musiał mieć kwalifikacje operatora). Konieczność pośrednictwa operatora w komunikowaniu się użytkownika z systemem przy wzrastającej mocy obliczeniowej systemów i lawinowym powiększaniu się liczby użytkowników powodowały, że sytuacja ta stanowiła swoistego rodzaju wąskie gardło w procesie użytkowania systemów liczących. Opracowanie efektywnych wieloprogramowych i wielodostępnych systemów operacyjnych wymagało w pierwszym rzędzie rozwiązania tego problemu. Tym rozwiązaniem są właśnie języki zleceń.

Użytkownik współczesnych systemów cyfrowych ma w zasadzie do wyboru dwa tryby współpracy z systemem: pracę wsadową i pracę interakcyjną w trybie wielodostępu. Historycznie pierwszy sposób użytkowania maszyny

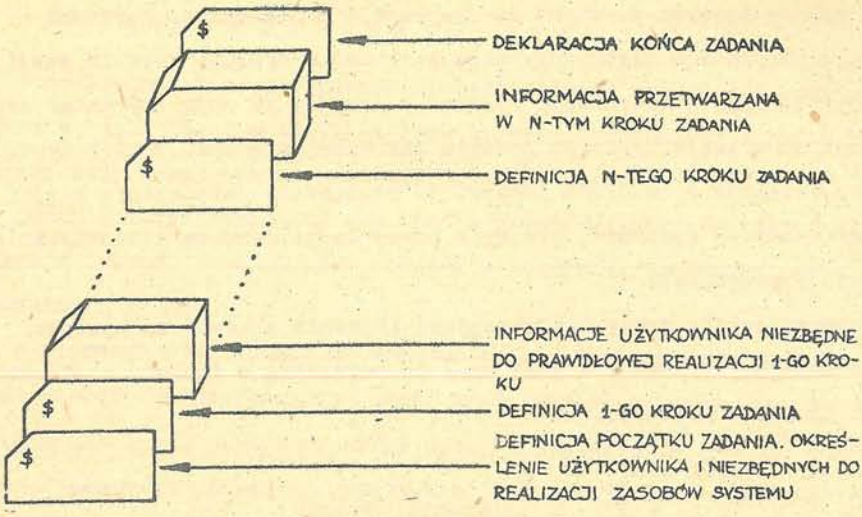
cyfrowej - praca w trybie tradycyjnym, która polegała na tym, że system łącznie ze wszystkimi zasobami był oddawany we wyłączne władanie jednego użytkownika na określony przeciąg czasu, jest dzisiaj stosowany w przypadku tzw. komputerów osobistych i niektórych, niewielkich zestatów minikomputerowych.

Realizację problemu obliczeniowego użytkownika w systemie cyfrowym polegającą na kolejnym (sekwencyjnym) wykonaniu różnych programów przy zapewnieniu odpowiedniego przekazywania danych i wyników pomiędzy nimi przyjęło się określać mianem zadania.

Można zatem uważać, że pojedyńczy krok zadania jest rezultatem wykonania określonego programu dla zdefiniowanych plików danych i zbiorów wynikowych.

W przypadku pracy wsadowej użytkownik, korzysta z języka zleceń, wstępnie przygotowuje szczegółowy opis poszczególnych kroków zadania, który przez analogię do definicji programu można nazwać algorytmem realizacji zadania zapisanym w języku zleceń i przeznaczonym do wykonania przez system operacyjny. Opis kroku musi precyzować program, który ma być wykonany jak i zawierać wszystkie informacje niezbędne zarówno do stworzenia przez system prawidłowych warunków wykonaniu programu jak i do prawidłowego przebiegu jego wykonania. Tak przygotowana definicja zadania ma postać pokazanego na rys. 3.2 pliku kart perforowanych (najczęściej) lub równoważnego mu odwzorowania na innym nośniku informacji. Zwykle języki zleceń do prac wsadowych zawierają instrukcje identyfikacji początku i końca definicji zadania. Dzięki temu zadanie użytkownika dołączone do innych zadań stanowi wsad, który zostaje wprowadzony do systemu. Od tego momentu użytkownik traci kontrolę nad przebiegiem realizacji swojej pracy w systemie.

W przypadku pracy interakcyjnej użytkownik mając do dyspozycji terminal umożliwiający z systemem dialog w interakcyjnym języku komend definiuje kroki swojej pracy na bieżąco i również na bieżąco otrzymuje wyniki realizacji każdego kroku. Innymi słowy elementarny cykl pracy



Znak $ wyróżnia karty zawierające instrukcje języka zleceń systemu operacyjnego

Rys. 3.2. Ogólna struktura zadania wsadowego
Źródło: opracowanie własne.

interakcyjnej użytkownika z systemem ma następującą postać:

1° Definicja kroku przez użytkownika.

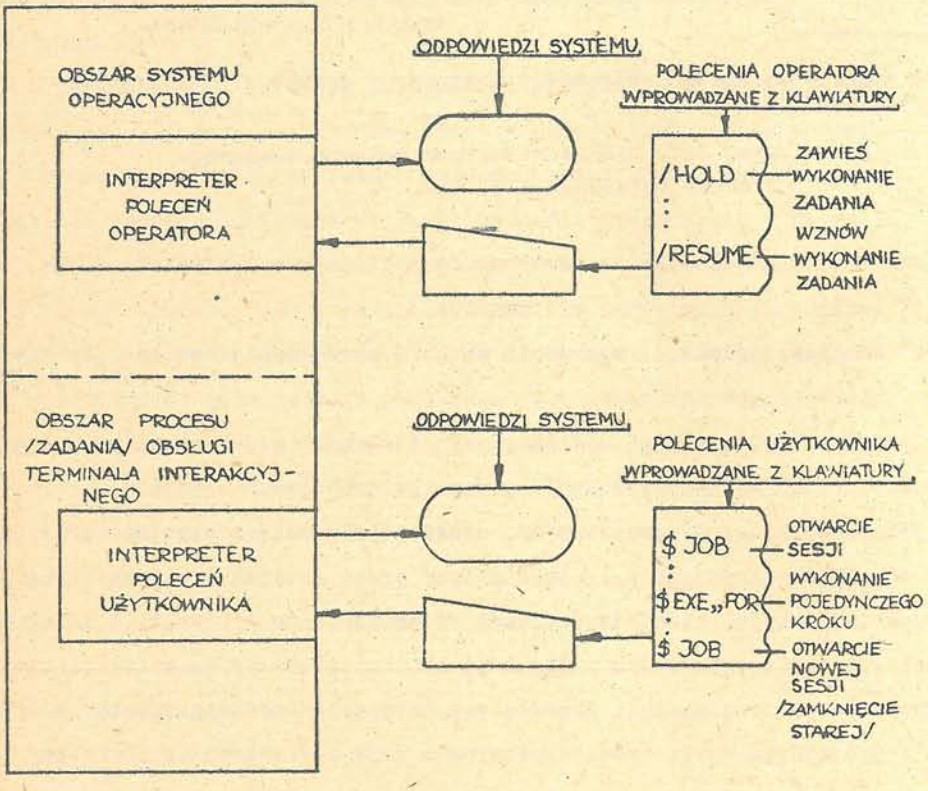
2° Analiza definicji, wykonanie kroku i przekazanie wyników użytkownikowi przez system.

Zatem, w trybie pracy interakcyjnej użytkownik z definicji ma pełny nadzór i kontrolę nad realizacją poszczególnych kroków zadania.

Jednorazowy seans interakcyjnej pracy użytkownika z systemem przyjęło się określać mianem sesji. Języki zleceń pracy interakcyjnej zawierają zawsze instrukcję, która sygnalizuje rozpoczęcie nowej sesji. W kategoriach systemu operacyjnego instrukcja ta jest sygnałem do uaktywnienia w systemie procesu obsługi określonego terminala (zadania interakcyjnego) i ustawienia parametrów jego opisu w stan początkowy zdefiniowany przez zbiór tzw. "założonych" (ang. default) wartości parametrów opisu zadania w systemie. Najogólniej rzecz biorąc parametry opisu zadania de-

finiują zasoby systemu stawiane do dyspozycji użytkownika. Wartości niektórych parametrów użytkownik może zmieniać w trakcie trwania sesji, ale z poprzedniego stwierdzenia wynika, że zmiany te mają charakter czasowy, gdyż obowiązują tylko do momentu zakończenia sesji. W niektórych systemach istnieje wyróżniona instrukcja służąca do zakończenia sesji, natomiast w każdym systemie, otwarcie nowej sesji oznacza automatycznie zamknięcie poprzedniej.

Podstawową zasadę pracy interakcyjnej objaśnia schemat na rysunku 3.3 odpowiadający sytuacji z jaką mamy do czynienia w trakcie użytkowania minikomputera MERA-400.



Rys. 3.3. Zasada pracy interakcyjnej na przykładzie systemu SOM-3 minikomputera MERA-400

Źródło: opracowanie własne.

3.1.3. Zadanie w systemie

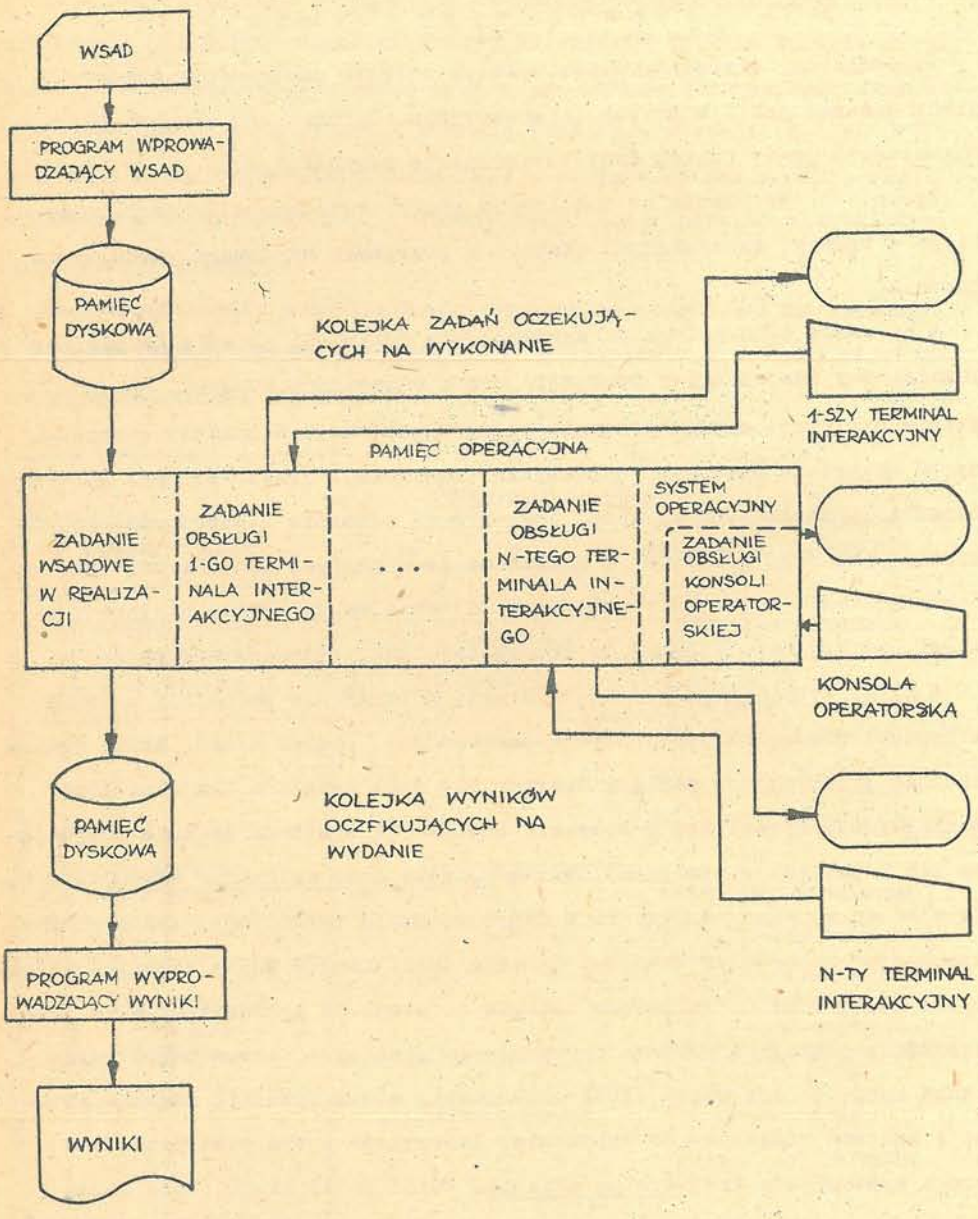
Współczesne, wieloprogramowe systemy cyfrowe umożliwiają zarówno pracę wsadową jak i w trybie interakcyjnym. Uproszczony schemat ideowy organizacji pracy takich systemów pokazuje rysunek 3.4.

Zgodnie z tym rysunkiem realizacja pracy użytkownika (zadania) przebiega w trzech, zasadniczych etapach w przypadku wsadowego trybu użytkowania:

- pierwszy etap polega na wprowadzeniu definicji zadania do pamięci pomocniczej (najczęściej dyskowej) przez odpowiedni program. Na tym etapie następuje zarejestrowanie pracy w systemie. W oparciu o informacje zawarte w pierwszej dyrektywie (instrukcji języka zleceń) system określa identyfikator zadania, jego wymogi odnośnie niezbędnych zasobów systemowych oraz priorytet i umieszcza je w pamięci pomocniczej (najczęściej dyskowej) w puli zadań oczekujących na przydział miejsca w pamięci operacyjnej. Zadania te tworzą tzw. kolejkę wejściową zadań.

- drugi etap rozpoczyna się z chwilą przepisania definicji zadania z pamięci pomocniczej do pamięci operacyjnej. Liczba zadań, które jednocześnie przebywają w pamięci operacyjnej jest ustalana dla określonej konfiguracji sprzętowej w momencie instalacji systemu. Zadania znajdujące się w pamięci operacyjnej tworzą kolejkę prac aktualnie wykonywanych. Kolejka ta wynika z faktu, że w danym momencie tylko jedno zadanie może dysponować procesorem maszyny. Zadanie takie nazwać można zadaniem aktywnym w danej chwili. Pozostałe zadania oczekują na przydzielenie im przez procedurę planującą systemu operacyjnego procesora centralnego. Wśród zadań znajdujących się w fazie oczekiwania można wyróżnić zadania gotowe i zadania niegotowe do wykonania. Informacja o tym jest pamiętana przez system jako tzw. status zadania. Wśród zadań niegotowych można znowu dokonać podziału na dwie grupy zadań:

1. Zadania zawieszone to jest takie, które oczekują na zajście określonego zdarzenia, którym przykładowo może być interwencja



Rys. 3.4. Schemat ideowy organizacji pracy współczesnych systemów cyfrowych

Źródło: opracowane własne.

operatora albo sygnał zakończenia transmisji danych przez kanał wejścia/wyjścia.

2. Zadania zakończone, które oczekują na usunięcie ich definicji z pamięci operacyjnej przez odpowiednie procedury systemowe. Warto może w tym miejscu zwrócić uwagę, że zadanie może mieć status "zakończonego" również w przypadku, gdy w trakcie jego dotychczasowej realizacji zaistniał błąd, który uniemożliwia dalszą realizację zadania.

Zmiany statusu zadania może dokonywać system operacyjny, operator, a w szczególnym przypadku również inne zadanie. Przerwanie zadania aktualnie wykonywanego powoduje zapamiętanie stanu jego wykonania tak, aby można go było później reaktywować i sterowanie przejmuje system operacyjny, który spośród zadań oczekujących i mających status gotowych do wykonania wybiera w myśl określonej strategii jedno zadanie, któremu nadaje status aktualnie wykonywanego, to jest przydziela do jego wyłącznej dyspozycji procesor maszyny. W ten sposób pojedyńcze zadanie w trakcie jego realizacji w systemie może wielokrotnie zmieniać swój status przechodząc z fazy oczekiwania do fazy wykonania i na odwrót. Stwarza to określone problemy związane z transmisją wyników obliczeń na urządzenia znakowe typu drukarka, których zwykle jest mniej aniżeli zadań jednocześnie wykonywanych. W praktyce problem ten jest rozwiązywany w dwojaki sposób. W małych systemach, które nie mają dostatecznie pojemnych pamięci dyskowych obowiązuje kolejkowanie operacji wejścia/wyjścia kierowanych do jednego urządzenia przez różne zadania. Polega to na tym, że jeżeli jedno zadanie angażuje określone urządzenie, to transmisje z innych zadań oczekują na przydzielenie im przez system operacyjny danego urządzenia na analogicznych zasadach jak zadania oczekują na przydział centralnego procesora. Rozwiązanie takie występuje między innymi w przypadku minikomputera MERA-400 pracującego pod systemami SOM-3 oraz SOM 3.P.

W dużych systemach natomiast wszelkie transmisje są realizowane za

pośrednictwem zbiorów dyskowych. W ten sposób wyniki, które są kierowane przez zadanie na przykład na drukarkę, są wstępnie gromadzone w zbiorach dyskowych i dopiero po zakończeniu zadania wyprowadzane na urządzenie docelowe. W takich systemach trzeci etap realizacji zadania polega właśnie na wyprowadzeniu wyników zadania przez specjalizowany program systemowy tak, jak to zostało schematycznie pokazane na rysunku 3.4.

Nieco inaczej przedstawia się sprawa interakcyjnych zadań obsługi terminali. Definicja tych zadań jest utworzona na etapie instalacji systemu operacyjnego i dlatego jest dostępna z chwilą załadowania systemu operacyjnego. W pamięci operacyjnej jest wydzielony obszar pod zadania interakcyjne, a liczba takich zadań odpowiada liczbie terminali interakcyjnych, które mogą równocześnie pracować w systemie. Każde zadanie interakcyjne, podobnie jak zadanie wsadowe, ma swoją nazwę oraz określoną pulę zasobów pozostających do dyspozycji zadania. Rozpoczęcie sesji interakcyjnej odpowiada wprowadzeniu zadania interakcyjnego do puli zadań aktualnie wykonywanych. Pewna różnica pomiędzy zadaniem wsadowym, a interakcyjnym ujawnia się w odniesieniu do zadań, które uzyskują status zakończonych. W przypadku jeśli jest to zadanie interakcyjne, to nie zostaje ono usunięte z systemu, a jedynie zostaje odtworzona jego definicja początkowa, co umożliwia użytkownikowi otwarcie nowej sesji interakcyjnej pracy z systemem.

Zarówno zadania wsadowe jak i interakcyjne można uważać za zadania użytkowe. Traktując zadania jako zamknięte i wzajemnie niezależne procesy przetwarzania informacji w maszynie można oprócz zadań użytkowych wskazać na zadania, które nazwiemy systemowymi. Na rysunku 3.4 zaznaczono jedno z takich zadań, a mianowicie zadanie komunikacji, które przeznaczone jest do interakcyjnej obsługi konsoli operatorskiej. Warto może jeszcze wspomnieć o jednym zadaniu systemowym, które występuje w każdym systemie, a mianowicie o tzw. zadaniu tracenia czasu. Jest ono traktowane przez system operacyjny jako zadanie użytkowe

o najniższym priorytecie i posiadające zawsze status gotowego do wykonania. Dzięki temu będzie ono aktywizowane tylko wtedy, gdy w systemie nie ma żadnego zadania użytkowego w stanie gotowości do wykonania. Treścią zadania tracenia czasu jest zwykle pojedyncza instrukcja stopu. W ten sposób okres bezczynności systemu z punktu widzenia użytkownika w kategoriach systemu operacyjnego oznacza wykonywanie zadania tracenia czasu.

3.2. Zadanie użytkownika w systemie MERA-400

Jak już wspominaliśmy we wstępie trzeciego rozdziału minikomputery MERA-400 mogą pracować pod różniącymi się między sobą systemami operacyjnymi SOM-3 oraz SOM 3.P. W dalszym ciągu tam, gdzie nie będzie zachodzić potrzeba rozróżniania obydwu wymienionych systemów, będziemy mówić o systemie operacyjnym SOM co będzie oznaczać, że prezentowane własności są wspólne dla SOM-3 oraz SOM 3.P.

3.2.1. Struktura zadania w pamięci operacyjnej

Systemy operacyjne SOM umożliwiają interakcyjny tryb użytkowania minikomputera MERA-400. Z chwilą wczytania systemu operacyjnego do pamięci operacyjnej w sposób omówiony w rozdziale drugim ma miejsce automatyczna definicja i uruchomienie zadań interakcyjnych obsługujących terminale użytkownika w ilości zależnej od systemu. Dla sytemu SOM-3 jest to jedno zadanie, a w przypadku systemu SOM 3.P mamy do czynienia z czterema zadaniami. Każde z tych zadań jest identyfikowane przez trzyliterową nazwę. Jedyne zadanie w systemie SOM-3 nazywa się zawsze JOB. System SOM 3.P dopuszcza jednoczesną pracę czterech końcowek interakcyjnych obsługiwanych przez zadania o nazwach JOA, JOB, JOC oraz JOD.

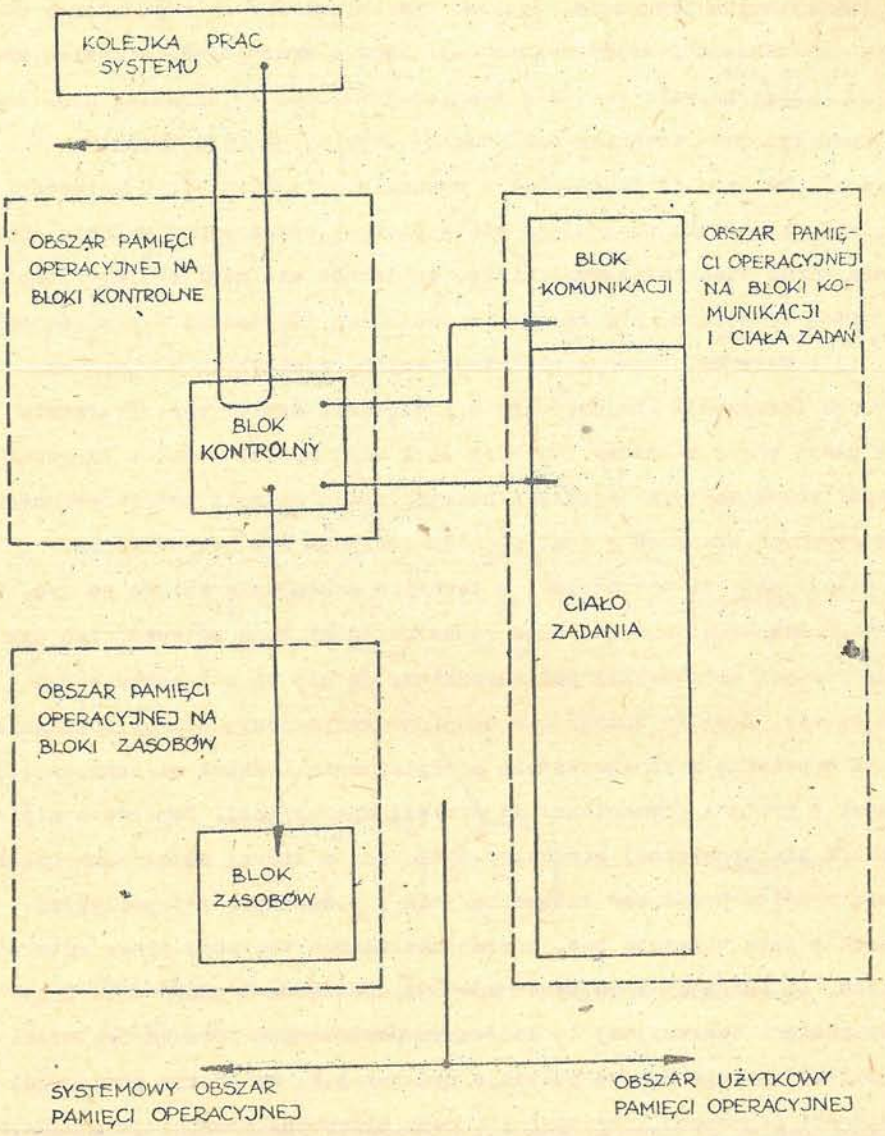
Charakterystyczną cechą systemów operacyjnych SOM jest statyczny podział pamięci operacyjnej na część systemową oraz część przeznaczoną na zadania użytkownika. Poszczególne zadania interakcyjne mają przydzielone rozłączne obszary pamięci o rozmiarach ustalonych na etapie instalacji systemu. W omawianym przypadku jedno zadanie zajmuje 32 K słów szesnastobitowych pamięci operacyjnej minikomputera MERA-400.

Z punktu widzenia sposobu obsługi zadania przez system operacyjny można pamięć zajętą przez pojedyncze zadanie podzielić na cztery, funkcjonalnie różne obszary:

- blok kontrolny, który zawiera opis stanu i parametry zadania,
- blok zasobów zawierający opis zasobów przydzielonych przez system operacyjny dla potrzeb realizacji zadania,
- blok komunikacji stanowiący wydzielony obszar pamięci operacyjnej zajmowanej przez zadanie i przeznaczony na bufory i miejsca robocze procedur systemowych,
- obszar przeznaczony na tzw. ciało zadania to jest na programy i dane wykorzystywane w trakcie realizacji zadania.

Dyslokację wymienionych elementów składowych zadania użytkowego w pamięci operacyjnej pokazuje schemat na rysunku 3.5.

Z rysunku tego wynika, że blok kontrolny i blok zasobów, które łącznie stanowią kompletny opis zadania, w oparciu o który system operacyjny sprawuje nad nim kontrolę, są lokowane w systemowym obszarze pamięci. Dwa pozostałe z wymienionych elementów zadania są umieszczane w jednym z użytkowych bloków pamięci operacyjnej. Nie dotyczy to zadań systemowych, których bloki komunikacji i ich ciała są również umieszczane w systemowym obszarze pamięci operacyjnej. Stwierdziliśmy uprzednio, że wykonanie zadania polega na sekwencyjnym wykonywaniu kolejnych kroków zadania, z których każdy polega na wykonaniu programu lub procedury systemowej. Program, którego wykonanie stanowi treść określonego kroku zadania jest ładowany w obszar ciała zadania. Z ograniczonego obszaru pamięci na ciało zadania wynika ograniczenie na roz-

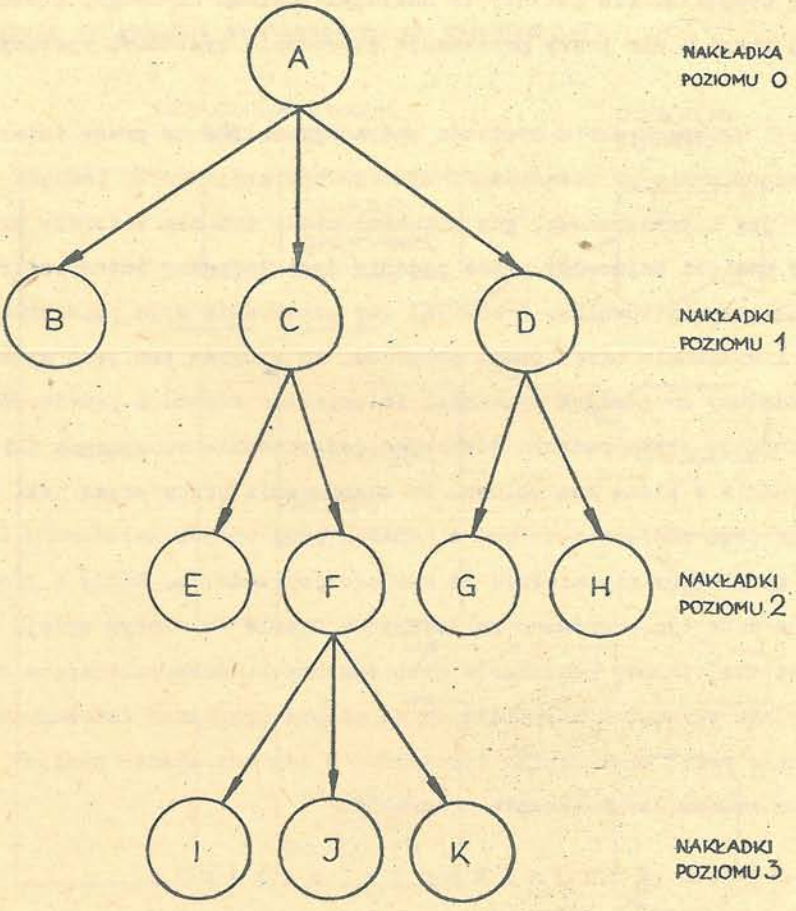


Rys. 3.5. Dyslokacja elementów zadania użytkownika w pamięci operacyjnej minikomputera MERA-400

Źródło: /4/.

miar wykonywanych programów. Systemy operacyjne SOM są wyposażone, podobnie jak większość współczesnych systemów operacyjnych, w pewien mechanizm, który pozwala obejść w pewnych granicach to ograniczenie. Mechanizmem tym jest technika nakładania (overlayowania). Technika ta opiera się na tym, iż aczkolwiek w momencie osiągania stanu gotowości do wykonania, zadanie znajdujące się w pamięci operacyjnej w fazie oczekiwania winno mieć załadowane ciało, to jednak nie musi ono być kompletne. System dopuszcza, aby fragmenty ciała nie rezydowały w pamięci operacyjnej a były umieszczone na zewnętrznych, dostępnych dla systemu nośnikach informacji (najczęściej w pamięciach dyskowych). Fragmenty takie noszą nazwę nakładek (overlayów) i mają postać modułów ładowania (na ogół relokowalnych) identyfikowanych przez nazwę i przechowywanych w zewnętrznych zbiorach o dostępie sekwencyjnym lub bezpośrednim. Oszczędność pamięci operacyjnej w technice nakładania polega na tym, że dwa różne moduły stanowiące dwie różne nakładki mogą zajmować ten sam obszar pamięci operacyjnej pod warunkiem, że nie są potrzebne w tym samym czasie. Jest to oczywiście okupione zwiększonym czasem wykonania zadania w związku z koniecznością przepisywania (czasem wielokrotnego) nakładek z pamięci pomocniczej do pamięci operacyjnej. Dopuszcza się istnienie hierarchicznej struktury nakładek, w której załadowana nakładka może z kolei powodować ładowanie jednej z nakładek jej podległej. Hierarchię taką obrazuje tzw. drzewo nakładania tworzone przez autora problemu, na barki którego spada również obowiązek zaplanowania przydziału pamięci operacyjnej do ładowania utworzonych przezeń nakładek.

Przykład takiego drzewa pokazuje rysunek 3.6, na którym poszczególne nakładki oznaczono kolejnymi literami alfabetu. Obowiązuje zasada, że w danym momencie czasu mogą być w pamięci tylko nakładki zaliczane do różnych poziomów, które leżą na jednej gałęzi drzewa obrazującego hierarchię nakładek. Dla przykładu z rysunku 3.6 możliwa jest zatem sytuacja gdy w obszarze pamięci zajmowanym przez ciało programu znajdują się nakładki oznaczone literami A, C oraz E, natomiast przykładowo



Rys. 3.6. Przykładowa struktura programu nakładanego
Źródło: Opracowanie własne.

nie mogą być równocześnie załadowane nakładki B oraz D. Należy podkreślić, że istnieje zawsze dokładnie jednak nakładka poziomu zerowego (tzw. nakładka albo overlay główny), która jako jedyna musi być załadowana jeżeli zadanie ma osiągnąć stan gotowości do wykonania. Zakończenie pracy przez nakładkę powoduje przekazanie sterowania do nakładki,

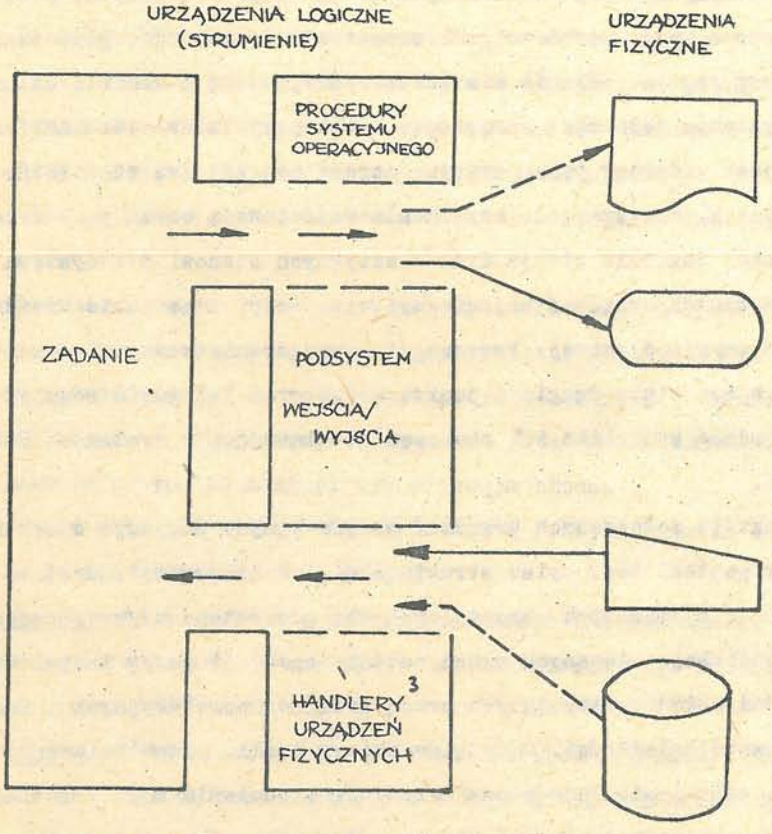
która ją wywołała. Nie dotyczy to nakładki poziomu zerowego, która z chwilą zakończenia pracy przekazuje sterowanie systemowi operacyjnemu.

Z racji ukierunkowania systemów operacyjnych SOM na pracę interakcyjną bezpośrednio po załadowaniu systemu operacyjnego do pamięci operacyjnej jak i każdorazowo, gdy aktualne ciało zadania zakończy pracę w obszar pamięci zajmowany przez zadanie jest ładowany interpreter języka zleceń użytkownika. Z chwilą, gdy użytkownik wyda polecenie załadowania i wykonania określonego programu, to program ten jest wyszukiwany, ładowany do pamięci wymazując interpreter komend i przejmuje na siebie funkcję ciała zadania dokonując jednocześnie stosownych zmian opisu zadania w bloku kontrolnym. Po zakończeniu pracy przez taki program w to samo miejsce może być w analogiczny sposób załadowany inny program działający niezależnie od swojego poprzednika. Każdy z tych programów może być programem nakładanym w sensie omówionym wyżej. Natomiast wielofazowa realizacja problemu użytkownika, polegająca na sekwencyjnym wykonaniu niezależnych od siebie programów ładowanych w ustalonej przez użytkownika kolejności w ten sam obszar pamięci, jest określana mianem łańcuchowania programów.

3.2.2. Komunikacja zadania z otoczeniem

Wszelka komunikacja zadania z otoczeniem odbywa się za pośrednictwem tzw. strumieni wejścia/wyjścia. Z punktu widzenia użytkownika strumienie we/wy stanowią urządzenia logiczne, którymi może się on wygodnie posługiwać przy programowania wymiany informacji pomiędzy opracowywanym programem a rzeczywistymi urządzeniami peryferyjnymi systemu. Istnienie strumieni we/wy, a co za tym idzie możliwość korzystania za ich pośrednictwem z peryferii systemu w zunifikowany sposób jest efektem pracy odpowiednich składowych systemu operacyjnego, które organi-

zują i pośredniczą w wymianie informacji pomiędzy zadaniem a otoczeniem. Ilustruje to schemat zamieszczony na rysunku 3.7.



Rys. 3.7. Zasada komunikacji zadania z otoczeniem

Źródło: Opracowanie własne.

Zgodnie z tym schematem strumień dla zadania stanowi pewien abstrakcyjny kanał przesyłowy informacji z i do bliżej niesprecyzowanych urzą-

---

[3] Mianem "handlera" przyjęło się określać program dopasowujący sposób pracy określonego fizycznego urządzenia peryferyjnego do standardu wyznaczanego przez dany system operacyjny. Program taki jest dołączony do puli procedur systemowych. Dzięki takiemu rozwiązaniu zapewniona zostaje duża łatwość podmiany określonego urządzenia na inne bez konieczności dokonywania istotnych zmian w samym systemie operacyjnym.

dzeń zewnętrznych. Dołączenie strumieni we/wy odbywa się na poziomie systemu operacyjnego w oparciu o informacje zawarte w bloku zasobów zadania do fizycznie zainstalowanych urządzeń peryferyjnych. Należy w tym miejscu sprecyzować pojęcie urządzenia fizycznego, gdyż zakres znaczeniowy tego określenia nie zawsze pokrywa się z realnie zainstalowaną w systemie jednostką sprzętową. I tak przykładowo terminal użytkownika jest widziany przez system operacyjny jako dwa niezależne urządzenia: klawiatura jako urządzenie wejściowe i ekran jako urządzenie wyjścia. Podobnie stacja dysków sztywnych stanowi dla systemu zespół niezależnych urządzeń wejścia/wyjścia. Pojedyncze takie urządzenie odpowiada sekcji dyskowej. Poszczególne urządzenia zewnętrzne zostały omówione w rozdziale drugim w punktach 2.6 oraz 2.7 gdzie również została podana symbolika ich oznaczeń obowiązująca w systemach SOM-3 i SOM 3.P.

Całokształt dołączonych urządzeń peryferyjnych znajduje w systemie odbicie w postaci tzw. opisu strukturalnego konfiguracji. Jest to zbiór tablic systemowych obrazujących dla podsystemu we/wy zarządzającego realizacją zlecanych przez zadanie operacji we/wy indywidualne właściwości układów sterujących pracą urządzeń peryferyjnych i ich wzajemne współzależności. Oddzielne zbiory tablic przewidziane są na definicje strumieni. Pojedyncza definicja strumienia musi określać jego nazwę, która może liczyć maksimum 3 znaki będące literami bądź cyframi oraz powinna zawierać dwa odnośniki do tablic opisu strukturalnego konfiguracji, które precyzują tzw. przydziały strumienia: aktualny i standardowy.

Przydział aktualny dokonywany za pomocą procedury systemowej ASSIGN określa urządzenie lub inny strumień, którego dotyczą wszystkie zlecane przez zadanie operacje we/wy.

Przydział standardowy dokonywany procedurą DEFAULT określa urządzenie(lub inny zdefiniowany w systemie strumień) używane w standardowych warunkach pracy.

Przydziały: standardowy i aktualny mogą się różnić, przy czym powrót do standardu następuje w przypadku:

a) odtwarzania stanu początkowego zadania. Sytuacja taka ma miejsce bezpośrednio po załadowaniu systemu operacyjnego do pamięci lub jest wywołana poleceniem użytkownika inicjującym pracę interpretera komend (dyrektywa $JOB w przypadku systemu SOM-3 lub dyrektywa Q przy pracy pod kontrolą systemu SOM 3.P);

b) stanu alarmowego w urządzeniu na które opiewa aktualny przydział strumienia.

Użytkownik programując operacje we/wy zdefiniować strumień, którego dana operacja dotyczy. Należy przy tym pamiętać, że system przyjmie zleconą operację we/wy do realizacji jedynie wtedy, gdy strumień jest przydzielony do zdefiniowanego w systemie urządzenia fizycznego. W przeciwnym przypadku nastąpi sygnalizacja błędu.

Z punktu widzenia dostępności strumieni z poziomu zadania użytkownika można je podzielić na dwa rodzaje:

a) lokalne to jest takie, których definicje umieszczone są w bloku zasobów zadania i pozostające całkowicie pod jego kontrolą, a niedostępne dla innych zadań;

b) globalne, tzn. umieszczone w bloku zasobów systemowego zadania komunikacji (i pozostające pod kontrolą tegoż zadania, ale udostępnione do wykorzystania przez inne zadania).

Lokalność definicji strumieni we/wy pozwala używać identycznych nazw strumieni w różnych zadaniach. Każdy strumień zadeklarowany przy wykonywaniu operacji we/wy traktowany jest początkowo jako lokalny i poszukiwanie jego opisu dokonywane jest we własnym bloku zasobów zadania. Dopiero w przypadku braku lokalnej definicji system przechodzi do przeszukiwania bloku zasobów zadania komunikacji (globalnego bloku zasobów). Brak strumienia o zadanej nazwie spowoduje sygnalizację błędu.

Wszystkie zlecone przez programy operacje we/wy ustawiane są w kolej-

ce do autonomicznego podsystemu wejścia/wyjścia. Autonomia tego podsystemu polega na tym, że może on realizować zlecone operacje nie angażując centralnego procesora systemu. Dzięki temu operacje we/wy mogą być wykonywane równolegle z olbiczeniami realizowanymi przez procesor. Systemy operacyjne SOM dopuszczają dwa reżimy wykonywania operacji we/wy:

1. Wait - zadanie ulega zawieszeniu po przekazaniu zlecenia podsystemowi we/wy i jest wznawiane po jej zakończeniu,
2. Quick-Return - ten reżim polega na tym, że po przekazaniu zlecenia podsystemowi we/wy zadanie jest kontynuowane (i w szczególności może zlecać wykonanie następnych operacji we/wy).

Podsystem wejścia/wyjścia stanowi zbiór wielodostępnych procedur systemowych, które jako miejsc roboczych używają odpowiednich tablic systemowych oraz zdefiniowanych przez zadania buforów. Można w nim wyróżnić trzy zasadnicze elementy pełniące określoną rolę w procesie obsługi zlecanych operacji we/wy:

1. Koordynator nadzoruje całość komunikacji z peryferiami systemu, planuje kolejność obsługi poszczególnych operacji na urządzeniach fizycznych, kontroluje stan kolejek we/wy, dokonuje wyboru operacji we/wy, inicjuje ich rozpoczęcie oraz wykonuje czynności związane z zakończeniem zleconej operacji, organizuje dostęp do buforów we/wy i komunikację z zadaniem.
2. Dystrybutor przyjmuje i obsługuje przerwania peryferyjne, interpretuje zachodzące w peryferiach zdarzenia, synchronizuje pracę koordynatora i kontroluje poprawność pracy urządzeń sygnalizując sytuacje alarmowe lub podejmując próby skorygowania błędów.
3. Handlery są indywidualnymi procedurami obsługi urządzeń we/wy i stanowią opisy realizacyjne każdej operacji wejścia/wyjścia. Każdy istniejący w systemie typ urządzeń wymaga istnienia odrębnego handlera. Wykazując zasadnicze różnice w algorytmach obsługi konkretnych urządzeń poszczególne handlery unifikują swoje czyn-

ności w ramach komunikacji z zadaniem stwarzając tym samym z jego punktu widzenia jednolity aparat obsługi urządzeń peryferyjnych.

Unifikacja obsługi urządzeń peryferyjnych wprowadzana przez oprogramowanie systemowe jako całość polega na określeniu listy możliwych operacji we/wy i określeniu dopuszczalnych formatów informacji transmitowanych do oraz z urządzeń peryferyjnych.

3.2.2.1. Repertuar operacji wejścia/wyjścia

Informacja płynąca poprzez strumienie we/wy może mieć postać znakową lub binarną i zorganizowana jest w jednostki zwane rekordami. Rekord stanowi dla systemu najmniejszą porcję informacji jaka może podlegać procesowi transmisji. System dopuszcza możliwość operowania czterema typami rekordów, które zostaną omówione w następnym punkcie. Większą od rekordu jednostką informacji jest zbiór (plik) rozumiany jako sekwencja rekordów zakończona znacznikiem końca zbioru (EOF). Dla zbiorów zbudowanych z rekordów znakowych znacznikiem tym jest rekord zawierający jedynie dwa znaki dolara ($$). Wszystkie dopuszczalne w systemie operacje wejścia/wyjścia odnoszą się do tych dwóch jednostek informacji i ich pełna lista przedstawia się następująco /4/:

- READ - wczytanie pojedynczego rekordu,
- WRITE - zapis pojedynczego rekordu,
- REWIND - ustawienie na początek informacji w strumieniu,
- ADVANCE FILE - pomijanie w strumieniu rekordów aż do napotkania rekordu-znacznika końca zbioru (EOF),
- BACKSPACE FILE - wycofywanie rekordów aż do napotkania znacznika końca zbioru,
- ADVANCE RECORD - pominięcie rekordu w strumieniu,
- BACKSPACE RECORD - cofnięcie się na początek poprzedniego rekordu w strumieniu,
- WRITE EOF - wyprowadzenie rekordu-znacznika końca zbioru,

- HOME - normalizacja położenia w strumieniu.

Poszczególne operacje są zlecane podsystemowi we/wy do wykonania przez program użytkowy na drodze wywołania odpowiedniej procedury systemowej.

3.2.2.2. Podstawowe typy rekordów informacji

Autonomia podsystemu we/wy i jego pełna niezależność od współpracy zlecającego operację we/wy zadania wymaga wzajemnego przekazywania transmitowanych danych za pośrednictwem specjalnie w tym celu wydzielonych obszarów pamięci dostępnych zarówno dla programu jak i podsystemu we/wy. Obszary te (tzw. bufory we/wy) są zlokalizowane w obszarze pamięci operacyjnej zajmowanej przez zadanie użytkownika. Każdorazowo podsystem we/wy musi być informowany o adresie bufora i jego długości liczonej w bajtach przez program zlecający wykonanie transmisji danych.

Podsystem we/wy jest zorientowany na obsługę transmisji danych w dwóch podstawowych trybach: znakowym i binarnym.

Znakowy tryb pracy podsystemu we/wy gwarantuje użytkownikowi jednolitą postać informacji w buforze (wszystkie znaki w kodzie ISO-7) niezależnie od wewnętrznego kodu fizycznych urządzeń we/wy. Wszystkie ewentualnie niezbędne konwersje kodów są realizowane automatycznie przez podsystem we/wy.

W przypadku binarnego typu pracy podsystemu we/wy transmitowana informacja nie podlega żadnej konwersji. Tym samym ten reżim pracy umożliwia operowanie na oryginalnej postaci informacji zewnętrznej. Jej organizacja i sposób rejestracji na nośniku są indywidualną sprawą poszczególnych urządzeń peryferyjnych.

W każdym z wymienionych trybów pracy podsystem wejścia/wyjścia mogą brać udział dwa typy rekordów: standardowy i niestandardowy. Z tego powodu informacja zawarta w buforze we/wy może być zawsze uważana za jeden z następujących, podstawowych typów rekordu zdefiniowanych w systemie:

1. Standardowy rekord znakowy.

2. Niestandardowy rekord znakowy.

3. Standardowy rekord binarny.

4. Niestandardowy rekord binarny.

Zanim przystąpimy do krótkiej charakterystyki poszczególnych rekordów zwróćmy uwagę, że z tej racji iż słowo pamięci operacyjnej komputera MERA 400 liczy 16 bitów sprzęt maszyny jest nastawiony na operowanie tą jednostką informacji maszynowej. Oznacza to, że w przypadku rekordów znakowych kody znaków będą upakowywane po dwa na jedno słowo. Z tego powodu w przypadku rekordów liczących nieparzystą ilość znaków, długość rekordów przy standardowym reżimie pracy będzie zaokrąglana w górę do całkowitej liczby słów. Zgodnie z tabelą 6 z rozdziału 2 kody znaków w standardzie ISO-7 liczą siedem bitów. W słowie maszynowym kod pojedynczego znaku zajmuje osiem bitów, przy czym ósmy, najstarszy bit ciągu kodowego jest zawsze wyzerowany. Odstępstwem od tej zasady jest tzw. skompresowana postać informacji znakowej, którą omówimy oddzielnie. Przy omawianiu struktury poszczególnych typów rekordów będziemy posługiwać się szesnastkowymi kodami znaków oraz ich nazwami zgodnie z tabelą 6. Szesnastkowe wartości kodów będą dla wyróżnienia zawsze poprzedzane znakiem '#'.

Ad 1. Standardowy rekord znakowy

Umieszczony w buforze standardowy rekord znakowy składa się z ciągu kodów, znaków w standardzie ISO-7 pakowanych po dwa do jednego 16 bitowego słowa pamięci operacyjnej. Początkowy znak rekordu nie może mieć wartości # 03 (ETX) ani # 07 (BEL), gdyż są to wyróżniki standardowego rekordu binarnego. Ograniczniki informacji użytecznej w buforze stanowi znak NUL o kodzie # 00 (tzn. bajt zerowy).

Wprowadzany rekord tego typu jest zakończony słowem zawierającym dwa znaki NUL (wszystkie bity słowa są wyzerowane). Jeżeli wprowadzany rekord liczy nieparzystą liczbę bajtów, to podsystem we/wy automatycznie uzupełnia go na ostatniej pozycji znakowej znakiem odstępu SP

o kodzie # 20. Dzięki temu standardowy rekord znakowy po wprowadzeniu zawsze zajmuje całkowitą liczbę słów maszynowych.

Wyprowadzany standardowy rekord znakowy powinien być zakończony zerowym bajtem. Jeśli informacja użyteczna rekordu wypełnia dokładnie cały bufor i nie ma miejsca na umieszczenie tam tego znacznika końca rekordu, to znak NUL jest automatycznie generowany przez podsystem we/wy na zakończenie transmisji.

Handlery urządzeń drukujących (do tej kategorii zaliczany jest również znakowy monitor ekranowy) intepretując w specjalny sposób dwa pierwsze znaki rekordu. Bajt początkowy służy do sterowania położeniem karetki przed każdą operacją WRITE. Po jego rozpoznaniu handler generuje odpowiednią sekwencją znaków kontrolnych sterujących pracą urządzenia. Na tej pozycji mogą się pojawić następujące znaki:

- znak spacji SP o kodzie #20 powodujący wydruk rekordu od początku następnej linii (powrót karetki + wysuw),
- znak 0 o kodzie #30 powodujący wydruk rekordu z odstępem (powrót karetki + dwa wysuwy),
- znak 1 o kodzie #31 powodujący wydruk rekordu od nowej strony (powrót karetki + przejście do nowej strony lub 16 wysuwów),
- znak + o kodzie # 2B powodujący nadruk rekordu w tej samej linii (powrót karetki).

Następny bajt buforu jest traktowany jako pierwszy użytkowy znak rekordu o ile nie jest to znak STX o kodzie # 02. W tym bowiem przypadku bajt ten jest ignorowany, a początkowy znak rekordu mieści się w następnym słowie. Taka interpretacja umożliwia dodawanie specjalnego słowa sterującego przed właściwą treścią rekordu i ułatwia redagowanie wyprowadzanych tekstów.

Handlery pozostałych (nie drukujących) urządzeń peryferyjnych nie sprawdzają zawartości bajtów sterujących i transmitują je jako normalne znaki.

Ad 2. Niestandardowy rekord znakowy

Umieszczony w buforze niestandardowy rekord znakowy jest traktowany jako ciąg znaków w kodzie ISO-7. Normalnym ograniczeniem transmisji niestandardowych rekordów znakowych jest zadeklarowany rozmiar buforu, aczkolwiek istnieje opcjonalna możliwość zgłoszenia podsystemowi we/wy kodu znaku, który będzie pełnił funkcję znacznika końca niestandardowego rekordu znakowego. W takim przypadku wykrycie terminatora jest powodem do zakończenia transmisji rekordu, a on sam jest ostatnim ze znaków przesłanych.

Wprowadzany rekord omawianego typu jest automatycznie uzupełniany kodem znaku spacji o wartości #20 jeżeli zajmuje w buforze nieparzystą liczbę bajtów. W trakcie wprowadzania rekordu podsystem we/wy zlicza wprowadzane znaki i informacja ta jest dostępna dla programu po ukończeniu transmisji. Liczba wprowadzonych znaków stanowi podstawę do oddzielenia w buforze informacji użytecznej od nieistotnej. Należy nadmienić, że wykrycie znacznika końca rekordu fizycznego na nośniku zewnętrznym powoduje zakończenie transmisji niestandardowego rekordu znakowego. Możliwość korzystania z omawianego reżimu transmisji pozwala wykorzystać w systemie MERA 400 zbiory informacji znakowych utworzone w standardach innych systemów liczących.

Ad 3. Standardowy rekord binarny

Każdy standardowy rekord binarny składa się z czołówki i danych. Czołówka zawiera zawsze trzy słowa o następującej treści:

słowo 1 - zerowy bajt/bity 0-7 słowa) zawiera zawsze znak ETX o kodzie #03 lub znak BEL o kodzie #07.
- pierwszy bajt (bity 8 - 15) nie ma znaczenia, ale najczęściej jest wykorzystywany przez procedury systemowe jako licznik modulo 128 rekordów w zbiorze,

słowo 2 - zawiera ogólną długość rekordu w bajtach,

słowo 3 - zawiera sumę kontrolną rekordu w oparciu o którą jest weryfi-

kowana poprawność transmisji rekordu. Suma kontrolna jest tworzona i weryfikowana przez podsystem we/wy.

W powyższym schemacie założono numerację bitów i bajtów w słowie od lewej strony.

Treść pozostałych słów rekordu jest wyłączną sprawą programów użytkowych.

Ad 4. Niestandardowy rekord binarny

Umieszczony w buforze niestandardowy rekord binarny ma (w zależności od sfecyfiki urządzenia we/wy) postać ciągu 16-bitowych słów binarnych lub też 8-bitowych znaków upakowanych po dwa w słowo maszynowe. Nie wyróżnia się żadnych ograniczników informacji w buforze, a jedynym realnym ograniczeniem staje się zadeklarowana długość buforu.

Wprowadzenie takiego znaku prowadzi zawsze do umieszczenia w buforze całkowitej liczby słów informacji. Niektóre urządzenia (np. pamięci dyskowe) prowadzą transmisję słowami 16-bitowymi. Większość urządzeń znakowych przesyła 8-bitowe znaki binarne i przy transmisji nieparzystej liczby znaków ostatnie słowo jest uzupełniane zerowym bajtem o wartości #00.

Wyprowadzenie niestandardowego rekordu binarnego na urządzenie z natury transmitujące słowa 16-bitowe prowadzi do zaokrąglenia wzwyż nieparzystej (w bajtach) długości bufora. Na urządzenia transmitujące 8-bitowe znaki binarne wyprowadzeniu podlega dokładnie podana ilość informacji.

W trakcie transmisji podsystem we/wy zlicza przesyłane bajty informacji dzięki czemu istnieje możliwość wykrycia wcześniejszego zakończenia transmisji (najczęściej w wyniku sygnalizacji końca rekordu fizycznego na nośniku urządzenia przypisanego do danego strumienia) poprzez porównanie liczby przesłanych bajtów z długością bufora. Jest to jednocześnie metoda rozdzielenia w buforze informacji użytecznej od nieistotnej.

3.2.2.3. Skompresowana postać informacji znakowej

Standardowe rekordy znakowe są przez podsystem we/wy umieszczane na zewnętrznym nośniku informacji w ten sposób, że początek rekordu pokrywa się z początkiem rekordu fizycznego zdefiniowanym dla określonego nośnika (patrz rozdział 2). W przypadku, gdy długość rekordu znakowego jest istotnie mniejsza od długości rekordu fizycznego prowadzi to do nieefektywnego wykorzystania nośnika informacji.

Sytuacja ta ma miejsce w przypadku pamięci dyskowych, dla których sprzętowo zdefiniowany rozmiar bloku (rekordu fizycznego) wynosi 512 bajtów. Tymczasem wiele programów zarówno systemowych (np. translatory) jak i użytkowych posługuje się rekordami liczącymi 80 i mniej znaków w kodzie ISO - 7. Ponadto analiza rekordów znakowych wskazuje na to, że nade często zawierają one ciągi powtarzających się znaków (najczęściej dotyczy to znaku spacji). Wymienione przesłanki doprowadziły do utworzenia formy zapisu kolejnych rekordów znakowych w postaci jednego, lub większej ilości rekordów skompresowanych w jednym bloku fizycznym. Rekord skompresowany na format odpowiadający formatowi standardowego rekordu binarnego: zerowy bajt rekordu zawiera kod #03 lub #07, w bajcie pierwszym jest zapisany numer rekordu, a bajty: czwarty oraz piąty są zarezerwowane na wpisanie sumy kontrolnej, jeżeli tego wymaga konkretny program. Bajty: drugi i trzeci zawierają długość rekordu w znakach, a skompresowana informacja znakowa jest wpisywana począwszy od szóstego bajtu rekordu. Określenie "skompresowany" wywodzi się stąd, że ciągi kolejnych identycznych znaków są kodowane przy użyciu dwóch bajtów bez względu na długość takiego ciągu. Wykorzystany jest tutaj fakt, że zapamiętanie znaku w kodzie ISO-7 wymaga tylko siedmiu bitów. Pierwszy z dwóch bajtów kodujących ciąg znaków zawiera kod powtarzającego się znaku. Drugi bajt zawiera liczbę ujemną, której moduł określa liczbę powtórzeń znaku (nie licząc pierwszego wystąpienia ciągu w znaku). Zapisanie liczby powtórzeń jako wartości ujemnej gwarantuje, że

najstarszy bit takiego bajtu jest zawsze ustawiony na jeden. Pozwala to rozróżniać bajty zawierające znaki w kodzie ISO-7 od bajtów pamiętających liczbę powtórzeń.

Pamiętanie informacji w postaci skompresowanej zapewnia oszczędność miejsca na nośniku pamięci pomocniczej przez to, że:

1. Tylko znacząca część rekordu (znaki przed bajtem zerowym, który jest znacznikiem końca rekordu przy niezapełnionym buforze) jest przepisywana do rekordu skompresowanego.
2. Ciągi kolejnych identycznych znaków są pamiętane tylko przy użyciu dwóch bajtów.
3. Skompresowana informacja znakowa jest blokowana do rozmiaru fizycznego rekordu na nośniku pamięci pomocniczej wykorzystywanej do przechowywania informacji.

Programy systemowe posługujące się informacją skompresowaną same dokonują w razie potrzeby kompresji i dekompresji informacji znakowej (np. translator Fortranu czy też edytor EDM).

Użytkownik, który chce posługiwać się w swoich programach tą postacią informacji znakowej ma do dyspozycji dwa podprogramy znajdujące się w systemowej bibliotece podprogramów użytkowych, które umożliwiają programowanie przetwarzania rekordów skompresowanych. Pierwszy z nich o nazwie LSCMR analizuje rekordy skompresowane, dekompresuje je i przekazuje do programu użytkownika, który go wywołał. Podprogram ten przekaże każdy rekord, którego pierwszy bajt nie zawiera wartości #03 lub #07 w niezmienionej postaci.

Drugi podprogram o nazwie LSCMW realizuje proces odwrotny, to znaczy analizuje dostarczane przez program użytkownika standardowe rekordy znakowe, kompresuje je i tworzy rekordy skompresowane w omówionej postaci.

3.2.2.4. Strumienie zadania użytkownika

W punkcie 3.2.2 omawiając zasadę komunikowania się zadania z otoczeniem powiedzieliśmy, że wymiana informacji pomiędzy zadaniem i peryferiami systemu odbywa się za pomocą abstrakcyjnych kanałów nazywanych strumieniami wejścia wyjścia. W momencie załadowania systemu operacyjnego dla każdego zadania obsługi terminala użytkownika tworzona jest odpowiednia tablica definiująca strumienie we/wy. Jeżeli w momencie instalacji systemu przewidziano w takiej tablicy puste pozycje, to oznacza, że użytkownik może dla potrzeb własnych programów definiować dodatkowe strumienie o nazwach różniących się od już zdefiniowanych. Nazw strumieni zdefiniowanych przez system użytkownik nie może zmienić. Może natomiast zredefiniować ich przydziały. Przydział aktualny można zmienić komendą ASSIGN, a przydział standardowy komendą DEFAULT. Wydruk aktualnej definicji strumieni zadania można uzyskać za pośrednictwem zadania komunikacji komendą (dyrektywą) operatorską /INFORM STR. Efektem użycia tej komendy będzie wyświetlenie (lub wydruk) tabelki, której trzy pierwsze kolumny podają odpowiednio: nazwę strumienia, jego przydział aktualny oraz przydział standardowy. W kolumnie przydziału aktualnego może pojawić się obowiązujący w systemie symbol urządzenia peryferyjnego (w tym sekcji dyskowej) lub nazwa innego istniejącego strumienia. Symbole urządzeń fizycznych obowiązujące w systemach SOM-3 i SOM 3.P zostały przedstawione w rozdziale 2. Tutaj należy dodać, że w obydwu systemach istnieje definicja fikcyjnego urządzenia o symbolu NO (tzw. zaślepki). Jego istnienie pozwala wyłączyć (poprzez przydzielenie go do odpowiedniego strumienia) w standardowo napisanych programach pewne procedury we/wy(np. wprowadzanie danych, wyprowadzanie wyników) bez potrzeby dokonywania zmian w tekstach programów. Urządzenie to, zdefiniowane przez fakt istnienia odpowiedniego handlera, akceptuje bez zastrzeżeń wszystkie operacje we/wy i potwierdza ich wykonanie

systemowi nie wywołując przy tym żadnej akcji na żadnym z rzeczywistych urządzeń peryferyjnych. Puste miejsce na pozycji któregoś z przydziałów strumienia oznacza, że nie jest on zdefiniowany (dotyczy to zwykle strumieni definiowanych lokalnie przez użytkownika).

W przypadku konfiguracji podstawowej systemu MERA-400 pracującej pod kontrolą jednozadaniowego systemu SOM-3 jest zainstalowany tylko jeden terminal użytkownika, który pełni również funkcję konsoli operatorskiej. Terminal taki jest traktowany jako dwa niezależne urządzenia oznaczane przez CK oraz CS (klawiatura i ekran lub drukarka). W przypadku systemu SOM 3.P pracującego w rozszerzonej konfiguracji urządzenia wejścia i wyjścia poszczególnych terminali mają oznaczenia trzyliterowe i te same urządzenia różnych terminali różnią się ostatnią literą oznaczenia. Klawiatury terminali w tym systemie mają oznaczenie CKx, a urządzenia listujące (monitory ekranowe lub drukarki) CSx gdzie x oznacza jedną z liter: A,B,C, lub D. Oznaczenia te korelują z nazwami zadań obsługujących prace terminali w tym systemie, których nazwy są również trzyliterowe o postaci JOx, przy czym ostatni, trzeci symbol ma znaczenie jak wyżej.

Istnieje pewien standard wykorzystywania poszczególnych strumieni przez system operacyjny i przez systemowe programy użytkowe (translatory, edytory tekstowe itp.). Ten standard wykorzystania przez oprogramowanie systemowe w przypadku większości strumieni jest sygnalizowany ich nazwą, która jest zwykle skrótem odpowiedniego terminu angielskiego wyjaśniającego standardowe wykorzystanie strumienia. Tablica 3.1 zawiera wykaz najważniejszych strumieni zadania użytkownika oraz przydzielonych im urządzeń w systemach SOM-3 oraz SOM 3.P. Z uwagi na to, że w momencie załadowania systemu operacyjnego przydziały: aktualny i standardowy strumieni nie różnią się między sobą w tablicy podano tylko jeden przydział. W kolumnie "wyjaśnienie" dla większości strumieni podano terminy angielskie, od których strumienie wzięły swoją nazwę oraz ich polskie tłumaczenie.

Tablica 3.1

Lokalne strumienie zadania użytkownika

Strumień	Przydział		Wyjaśnienie	
	SOM-3	COM 3.P		
CI	CK	CKx	Command Input	Wprowadzanie komend
CO	CS	CSx	Command Output	Wyjście dla komunikatów systemowych
SI	PR	ASB	Source Input	Wejście informacji źródłowej
SO	ASB	ASB	Source Output	Wyjście informacji źródłowej
LO	CSL	CSx	List Output	Wyjście listingowe
BI	ASA	ASA	Binary Input	Wejście binarne
BO	ASA	ASA	Binary Output	Wyjście binarne
JC	AJC	AJC	Job Control	Strumień definicji makroinstrukcji języka komend
JW	AJW	AJW	Job Work	Strumień roboczy makroinstrukcji języka komend
SC	ASC	ASC	Scratch Stream	Strumień roboczy
USL	NO	NO	USer Library	Biblioteka użytkownika
AI	PR	PRO	Auxiliary Input	Pomocnicze wejście komend
1	CK	CKx		
2	CS	CSx		
3	PR	PR		strumienie wykorzystywane jako urządzenia logiczne w fortranowskich instrukcjach READ oraz WRITE
4	PP	PP		
5	CSL	NO		

Źródło: /4/.

Oprócz wymienionych w tablicy strumieni, których definicje są lokalne dla poszczególnych zadań, każde zadanie ma również dostęp do puli tzw. strumieni globalnych, którymi posługuje się system operacyjny, a ściślej zadanie komunikacji. Najważniejszymi z punktu widzenia potrzeb użytkownika strumieniami globalnymi zdefiniowanymi w systemie są niżej wymienione i objaśnione strumienie:

- OC (Operator Commands) - używany przez systemowe zadanie komunikacji do przyjmowania poleceń operatora,
- CO (Communications Output) - służy do wyprowadzania komunikatów przeznaczonych dla operatora systemu,
- LMB (Load Modules Background) - jest to strumień przyłączony do sekcji dyskowej, na której jest przechowywana biblioteka systemowych programów użytkowych,
- LM (Load Modules) - przyłączony do sekcji dyskowej na której systemowo jest umieszczona biblioteka nakładek zadań rezydujących w pamięci operacyjnej. Wykorzystywany przez program systemowy LIB,
- LB (LiBrary) - przyłączony do sekcji dyskowej zawierającej systemową bibliotekę podprogramów.

Należy podkreślić, że to, w jaki sposób oprogramowanie systemowe wykorzystuje poszczególne strumienie zdefiniowane w systemie w niczym nie ogranicza użytkownika, który może je wykorzystywać w swoich programach do innych, aniżeli wyżej wymienione, celów.

3.2.3. Opcje sterujące sposobem wykonania zadania

Wszystkie zadania działające pod kontrolą systemów operacyjnych SOM mają dostęp do jednego ze słów umieszczonych w bloku kontrolnym zadania, które może być łatwo zmienione przez operatora lub użytkownika przy pomocy odpowiednich dyrektyw. Poszczególne bity tego słowa nazywanego słowem opcji mają ustalone w systemie dwuznakowe oznaczenia. Jeżeli w określonej komendzie, w której dopuszczalne jest wymienienie opcji, nazwa opcji wystąpi z dwuliterowym przedrostkiem NO to oznacza on wyzerowanie odpowiadającego jej bitu w słowie opcji. Użycie nazwy opcji bez tego przedrostka ustawia odpowiedni bit słowa opcji na wartość jeden. Główne wykorzystanie słowa opcji przez użytkownika polega na wyborze odpowiedniego wariantu wykonania systemowych programów użytkowych. Rzecz w tym, że poszczególne programy systemowe interpre-

tują określone bity słowa opcji w jednolity, a tym samym standardowy sposób.

Znaczenie istotnych dla określonego programu systemowego opcji będzie podawane każdorazowo przy opisie poszczególnych programów w rozdziale czwartym. Tutaj ograniczymy się do podania symboli poszczególnych opcji oraz krótkiego omówienia ich standardowego znaczenia.

Tablica 3.2 podaje nazwy poszczególnych bitów słowa opcji oraz ich znaczenie ze względu na sposób interpretacji przez systemowe programy użytkowe.

Poza słowem opcji istnieje w bloku kontrolnym zadania tzw. słowo komunikacji - FLAG, służące do informowania systemu o błędach zaistniałych w trakcie wykonania programu. W wielu wypadkach błąd powstały w trakcie pracy jednego programu uniemożliwia działanie innego programu, który korzysta z produktu wyjściowego poprzedniego. System bada stan tego słowa i jeżeli wskazuje on na to, że poprzedni program nie zakończył się prawidłowo, to dalszy ciąg pracy jest uzależniony od stanu opcji GO: jeśli opcja GO jest nieustawiona, to system nie dopuszcza do wykonania kolejnych programów (dyrektywy użytkownika EXE stają się nieaktywne). W takiej sytuacji użytkownik musi albo ustawić opcję GO, albo zainicjować prace interpretera komend systemowych (dyrektywa Q dla systemu SOM 3.P lub dyrektywa $JOB w przypadku systemu SOM-3) co powoduje zgaszenie wskaźnika błędu. Przy zapalonej opcji GO stan wskaźnika błędu w słowie komunikacji nie ma wpływa na kontynuację pracy w ramach bieżącej sesji.

Oprócz wymienionej funkcji słowo komunikacji pozwala operatorowi przekazywać kody poleceń dla realizowanego zadania przy pomocy dyrektywy /LET co ma istotne znaczenie w trakcie pracy pod kontrolą podsystemu programowania w języku BASIC.

Standardowo, po zainicjowaniu pracy interpretera komend systemowych ustawione są opcje LO, BO, SC, MA oraz JO. Bity pozostałych opcji są wyzerowane.

Tablica 3.2

Znaczenie bitów słowa opcji zadania

Bit	Nazwa opcji	Interpretacja - znaczenie
0 1 2 3 4 5	U0 U1 U2 U3 U4 U5	Opcje użytkownika. Brak jednolitego, standardowego sposobu interpretacji opcji od U0 do U5. Poszczególne programy mogą je interpretować w różny, charakterystyczny dla siebie sposób.
6	U6	Opcja ustawiana i interpretowana przez system
7	LO	Opcja listingu. Ustawienie tej opcji w przypadku translatorów pozwala uzyskać wydruk tekstu źródłowego tłumaczonego programu.
8	BO	Opcja sterująca generowaniem kodu wynikowego
9	SC	Opcja strumienia roboczego
10	HO	Opcja zatrzymania zadania. Wskazuje, czy działanie programu ma być zatrzymane w określonym miejscu.
11	MA	Określa, czy ma być drukowana mapa pamięci.
12	GO	Opcja kontynuowania pracy
13	AO	Określa z którego strumienia (CI czy AI) mają być czytane dyrektywy sterujące pracą programu.
14	JO	Opcja używana przez system. Programy użytkowe nie mogą jej używać.
15	DU	Opcja wydruku zawartości pamięci operacyjnej przy awaryjnym (błędnym) zakończeniu zadania. Interpretowana jest jedynie przez system operacyjny.

Źródło: /4/.

3.3. Zadanie Komunikacji. Język zleceń operatorskich

Zadanie komunikacji jest zadaniem rezydującym w obszarze systemowym pamięci operacyjnej minikomputera MERA-400. Jego podstawowym celem jest zapewnienie obustronnej komunikacji pomiędzy systemem operacyjnym a ope-

ratorem. Poprzez zadanie komunikacji system operacyjny informuje o wszystkich błędach zaistniałych podczas realizacji operacji wejścia/wyjścia, wyświetla na konsoli operatorskiej komunikaty kierowane pod adresem operatora przez zadania użytkownika oraz wczytuje i interpretuje komendy zleceń operatorskich. Zadanie komunikacji w systemie ma nazwę składającą się z trzech znaków spacji. Normalnie znajduje się ono w stanie zawieszenia i może być uaktywnione przez system operacyjny lub przez operatora systemu. W tym drugim przypadku uaktywnienie zadania komunikacji ma miejsce każdorazowo po naciśnięciu przycisku OPRQ na pulpicie technicznym minikomputera MERA 400. Po naciśnięciu tego przycisku zadanie komunikacji zgłasza się na terminalu, który w trakcie instalacji systemu został zdefiniowany jako konsola operatorska* poprzez wypisanie w nowej linii znaku "/" (slash) i przejście na nasłuch dyrektyw operatorskich. Dyrektywy (inaczej komendy) operatorskie mają ustaloną składnię, zbliżoną do typowej składni instrukcji języków symbolicznych. W ogólnym przypadku w pojedynczej dyrektywie można wyróżnić trzy, kolejno po sobie następujące pola:

1. Pole adresowe. Jeżeli pole to występuje, to musi zawierać poprawną nazwę zadania, do którego odnosi się dana dyrektywa. Interpretowane są tylko rzy pierwsze znaki, stąd też nazwy zadania JOA oraz JOALA są uważane za tożsame i odnoszą się do zadania o nazwie JOA. Większość dyrektyw operatorskich wymaga określenia nazwy. Z tego powodu, aby uniknąć konieczności powtarzania nazwy zadania w kolejnych komendach odnoszących się do tego samego zadania wprowadzono zasadę, że pominięcie nazwy zadania oznacza odwołanie się do nazwy zadania ostatnio użytej. Co więcej, możliwe jest użycie pustej komendy, która zawiera tylko pole adresowe określające poprawną nazwę zadania.

---

* W czterozadaniowym systemie SOM 3.P funkcję konsoli operatorskiej pełni terminal obsługiwany przez zadanie użytkownika JOA. W przypadku jednozadaniowego SOM-3 jedyny terminal użytkownika jest równocześnie konsolą operatorską.

Pozwala to ustawić nazwę zadania dla następnych komend, w których w związku z tym pole adresowe może być pominięte. Przykładowo komenda postaci

(JOA)

nie spowoduje żadnej akcji, poza tym, że ustawi opcjonalną wartość pola adresowego dla następujących po niej dyrektyw.

2. Słowo kluczowe dyrektywy. Zadanie komunikacji dysponuje zbiorem procedur, których wykonanie definiuje treść poszczególnych komend. Słowo kluczowe dyrektywy zawiera nazwę, albo jej dopuszczalny skrót, procedury, która będzie wywołana w procesie interpretacji zlecenia operatorskiego.
3. Pole parametrów dyrektywy zawiera niezbędne informacje do prawidłowego wykonania określonej słowem kluczowym procedury. Każda komenda ma ściśle określoną liczbę, postać i znaczenie parametrów. Niektóre parametry mają charakter obligatoryjny, inne mogą występować opcjonalnie. Istnieją dyrektywy, które w ogóle nie posiadają parametrów.

Znakami, które są interpretowane jako ograniczniki poszczególnych pól w obrębie dyrektywy są:

- przecinek ","
- znak równości "="
- znak spacji (odstęp) " "
- slash "/"

Znaki spacji (odstępy) poprzedzające określone pole są ignorowane. Zwykle, jako ogranicznika pola adresowego używa się znaku "/", gdyż jest to znak, którym zadanie komunikacji zgłasza się na konsoli operatorskiej, a w pozostałych przypadkach w charakterze ogranicznika najczęściej używany jest znak odstępu. Jeżeli parametrem określonej komendy może być liczba, to dopuszczalne jest podawanie jej wartości w systemie dziesiętnym lub szesnastkowym. W tym drugim przypadku zapis wartości szesnastkowej jest poprzedzany znakiem # (hash).

Zadanie komunikacji kontroluje poprawność informacji zawartych w polu adresowym i w polu słowa kluczowego dyrektywy, sygnalizując wykryte błędy operatorowi przy pomocy komunikatów:

! ILN - błąd w polu adresowym lub nieznana nazwa zadania

! ILD - błąd w słowie kluczowym lub nieznana nazwa dyrektywy

Poprawność pola parametrów jest sprawdzana przez procedurę interpretacyjną i w tym przypadku błędy są sygnalizowane poprzez wyświetlenie poprawnej części dyrektywy zakończonej znakiem zapytania. W obydwu przypadkach analiza dyrektywy zostaje przerwana, a zadanie komunikacji zaprzestaje wczytywania kolejnych komend przechodząc w stan zawieszenia. Jego reaktywowanie wymaga w takim przypadku ponownego naciśnięcia przycisku OPRQ.

3.3.1. Zasady prezentacji komend

Zbiór wszystkich poprawnych dyrektyw akceptowanych przez zadanie komunikacji przyjęło się określać mianem języka systemu operacyjnego. Dodatkowym uzasadnieniem takiej nazwy jest to, że analogiczną składnię i znaczenie jak wiele komend operatorskich mają komendy języka pracy interakcyjnej użytkownika pod kontrolą systemów operacyjnych SOM jak i komendy sterujące pracą systemowych programów użytkowych. Dotyczy to zwłaszcza komend sterujących operacjami wejścia/wyjścia i komend kontroli urządzeń zewnętrznych. Z tego powodu przy prezentacji poszczególnych dyrektyw zadania komunikacji będziemy pomijać część adresową pamiętając, że każda dyrektywa w podanej postaci może być poprzedzona trzyliterową nazwą zadania, do którego się odnosi, po której należy napisać znak "/". Ponadto, dla ujednolicenia syntaktycznego opisu dyrektyw przyjmiemy następującą konwencję metajęzykową:

- wszystkie elementy definicji pisane dużymi literami oznaczają symbole terminale i muszą pojawić się we wprowadzanej dyrektywie bez jakichkolwiek zmian. Jeżeli w symbolach tych są dopuszczalne skróty to znaki, które muszą wystąpić będą podkreślane,

- symbole metajęzykowe, za które należy podstawić odpowiednie, zgodne z wyjaśnieniem wartości będą pisane małymi literami i ujmowane w ostre nawiasy. Znaczenie najczęściej używanych oznaczeń metajęzykowych jest następujące:

  ⟨s⟩ nazwa strumienia,

  ⟨d⟩ nazwa urządzenia,

  ⟨sd⟩ nazwa strumienia lub nazwa urządzenia,

  ⟨n⟩ liczba całkowita z przedziału od 1 do 32767 dziesiętnie. Wartość taką wolno zapisać w systemie szesnastkowym, ale wtedy należy ją poprzedzić znakiem #,

  ⟨f⟩ nazwa pliku w komendach systemu plikowego systemu SOM 3.P.,

  ⟨ol⟩ lista opcji (patrz punkt 3.2.3),

  ⟨prog⟩ nazwa programu,

  ⟨prior⟩ priorytet zadania,

  ⟨FSD⟩ nazwa pliku, strumienia lub urządzenia.

  Inne symbole metajęzykowe użyte do opisu dyrektyw będą objaśniane w tekście.

- elementy definicji ujęte w nawiasy kwadratowe są opcjonalne i w związku z tym w konkretnym przypadku można je opuścić,
- użycie nawiasów klamrowych { } sygnalizować będzie konieczność wyboru jednego z wymienionych w nich elementów,
- wielokropek ... sygnalizuje możliwość wielokrotnego powtórzenia poprzedzającego go fragmentu dyrektywy; fragmentem tym jest zawartość poprzedzających go nawiasów kwadratowych, a w przypadku ich braku poprzedzający go bezpośrednio ciąg.

Z podanych reguł wynika, że nawiasy ostre, kwadratowe i klamrowe podobnie jak wielokropek i podkreślenia są symbolami metajęzykowymi i w związku z tym nie powinny pojawić się w rekordzie wyprowadzanej dyrektywy.

Warto jeszcze nadmienić, że wszystkie parametry dyrektyw powinny być zapisywane przy użyciu znaków należących do tak zwanego kodu CAN

którymi posługują się systemy operacyjne SOM. Kod CAN jest podzbiorem kodu ISO-7 i należą do niego następujące znaki:

- 26 liter alfabetu łacińskiego,
- 10 znaków cyfr,
- 4 znaki specjalne: dolar, kropka, dwukropek oraz znak spacji.

3.3.2. Dyrektywy sterujące operacjami wejścia/wyjścia

Dyrektywy zadania komunikacji zaliczane do kategorii wymienionej w tytule umożliwiają operatorowi:

- manipulowanie położeniem nośnika informacji na urządzeniu przydzielonym do strumienia wskazanego w dyrektywie,
- przerwanie wykonywania operacji wejścia/wyjścia na określonym urządzeniu,
- wyprowadzenie na określony strumień rekordu (znacznika) końca zbioru.

Składnia i znaczenie dyrektyw sterujących operacjami wejścia/wyjścia jest następująca:

REWIND ⟨s⟩ [⟨s⟩]	Przewiń do początku i znormalizuj położenie nośnika
AVFILE ⟨s⟩ [⟨n⟩]	Przesuń nośnik do przodu poza n-ty znacznik końca zbioru
BSFILE ⟨s⟩ [⟨n⟩]	Przesuń nośnik do tyłu o podaną liczbę znaczników końca zbioru
AVRECORD ⟨s⟩ [⟨n⟩]	Przesuń nośnik do przodu o podaną liczbę rekordów
BSRECORD ⟨s⟩ [⟨n⟩]	Jak AVR ale do tyłu
HOME ⟨s⟩[⟨s⟩]...	Znormalizuj położenie nośnika
EOFILE ⟨s⟩[⟨n⟩]	Na wskazany strumień wyprowadź podaną liczbę znaczników końca zbioru

TERMINATE ⟨s⟩[⟨s⟩]...

Przerwij wszystkie operacje we/wy zakolejkowane przez zadanie wskazane w części adresowej na urządzeniach przydzielonych do wymienionych strumieni.

We wszystkich wymienionych dyrektywach parametr określający nazwę strumienia może dotyczyć strumieni lokalnych bądź globalnych. Dostęp do lokalnego strumienia wymaga podania w części adresowej nazwy zadania, w którego bloku zasobów mieści się definicja danego strumienia. Dostęp do strumieni globalnych jest możliwy przez podanie nazwy // zadania komunikacji. Wszystkie dyrektywy manipulowania położeniem nośnika zakładają standardową organizację informacji na nośniku.

Wykrycie punktów skrajnych nośnika przed poprawnym zakończeniem operacji jest sygnalizowane wydrukami na konsoli operatorskiej:

! BOM s - początek nośnika na strumieniu s

! EOM s - koniec nośnika na strumieniu s

Poszczególne operacje związane z pozycjonowaniem nośnika są wykonywane w sposób charakterystyczny dla danego urządzenia, które jest aktualnie przydzielone do strumienia wymienionego w dyrektywie. Należy pamiętać, że nie wszystkie operacje wejścia/wyjścia są zdefiniowane na każdym z urządzeń. Przykładowo nie można żądać przewinięcia strumienia przypisanego do takich urządzeń jak klawiatura, perforator tasiemki papierowej itp.

Przykłady:

/JOA/REW SI SO - Na urządzeniach przypisanych do lokalnych strumieni SI oraz SO zadania JOA przewiń nośniki do początku.

/JOB/EOF BO - Wyprowadź rekord - znacznik końca zbioru na lokalny strumień BO zadania JOB. Opuszczenie parametru n oznacza zawsze przyjęcie jego wartości równej jeden.

/TER LO - Przerwij wykonywanie operacji we/wy na strumieniu LO zadania, którego nazwa była ostatnio wymieniona w dotychczas wprowadzonych dyrektywach. Operacje aktualnie wykonywane zostaną bezzwłocznie przerwane, a operacje oczekujące na dostęp do urządzenia zostaną usunięte z kolejki.

3.3.3. Dyrektywy kontroli strumieni wejścia/wyjścia

Cztery dyrektywy zadania komunikacji pozwalają operatorowi dokonywać zmian w tablicach definiujących strumienie zadania użytkownika. Są to dyrektywy OPEN, CLOSE oraz ASSIGN i DEFAULT.

Dyrektywa OPEN postaci

OPEN ⟨s⟩

pozwala utworzyć definicję nowego strumienia dla zadania określonego częścią adresową dyrektywy, o ile w tablicy strumieni, który znajduje się w bloku zasobów tego zadania, przewidziano taką ewentualność na etapie instalacji systemu pozostawiając w niej wolne miejsca. Jeśli tablica taka wolnych miejsc nie posiada to sygnalizowany jest błąd przekroczenia liczby dopuszczalnych strumieni. Jeżeli strumień o podanej nazwie istnieje już w bloku zasobów, to operacja OPEN niczego nie zmienia, czyli jest operacją pustą w takim przypadku. Strumień utworzony dyrektywą OPEN jest uważany przez system za strumień tymczasowy w odróżnieniu od strumieni stałych, których definicje istnieją w bloku zasobów od momentu załadowania zadania. Ma to znaczenie w przypadku wykonywania operacji zlecanej dyrektywą CLOSE o ogólnej postaci:

CLOSE ⟨s⟩[⟨s⟩] ...

Wykonanie tej dyrektywy spowoduje, że na urządzeniach aktualnie przydzielonych do każdego z wymienionych w dyrektywie strumieni zostanie cofnięta rezerwacja, o ile takowa miała uprzednio miejsce (patrz dyrektywa TAKE w następnym punkcie). W przypadku strumieni stałych działanie dyrektywy CLOSE na tym się kończy. Dla strumieni tymczasowych dodatkowo odłączane są obiekty aktualnego i standardowego przydziału a sama definicja strumienia w bloku zasobów ulega likwidacji. Z tego tytułu za podstawową funkcję tejże dyrektywy można uznać kaso-

wanie definicji strumieni tymczasowych. Należy przy tym pamiętać, że kasowanie definicji strumienia jest niedopuszczalne, jeżeli pośredniczy on w przydziale innego strumienia lub nie uległy zakończeniu zakolejkowane za jego pośrednictwem operacje wejścia/wyjścia.

Dwie ostatnie dyrektywy z omawianej grupy komend zadania komunikacji umożliwiają zmianę definicji przydziałów strumieni zadania użytkownika. Są to dyrektywy:

ASSIGN ⟨s⟩ ⟨sd⟩ [⟨s⟩ ⟨sd⟩]...

która umożliwia dokonanie aktualnego przydziału strumienia oraz

DEFAULT ⟨s⟩ ⟨sd⟩ [⟨s⟩ ⟨sd⟩]...

której użycie spowoduje zmianę definicji standardowego przydziału dla wymienionych w dyrektywie strumieni. Jak widać, składnia obydwu dyrektyw jest podobna. W obydwu przypadkach liczba użytych parametrów musi być parzysta. Pierwszy element każdej pary określa istniejący w bloku zasobów zadania, wskazanego w części adresowej dyrektywy, strumień, a drugi - obiekt przydziału. Obiektem przydziału może być nazwa urządzenia lub nazwa innego strumienia mającego określony przydział. W tym drugim przypadku mówimy, że strumień taki pośredniczy w definicji przydziału strumienia wymienionego na pozycji parametru s. Jeżeli w dyrektywie ASSIGN zamiast nazwy obiektu przydziału umieścimy parametr pusty*, to w efekcie uzyskamy określenie aktualnego przydziału strumienia zgodne z jego przydziałem standardowym.

3.3.4. Dyrektywy kontroli urządzeń wejścia/wyjścia

Każde z rzeczywistych urządzeń może, z punktu widzenia podsystemu wejścia/wyjścia, znajdować się w jednym z dwóch stanów logicznych:

* Parametr pusty jest definiowany przez dwa bezpośrednio po sobie występujące ograniczniki. Oznacza to, że w takim przypadku nie możemy używać w charakterze ogranicznika znaku spacji a na przykład posłużyć się w tym celu przecinkiem, np. /ASS SI,,SO,PR.

W stanie operatywnym /ON-LINE/ lub w stanie nieoperatywnym /OFF-LINE/. Operator ma możliwość dokonania zmiany stanu urządzenia przy pomocy dyrektyw postaci:

OFFLINE ⟨d⟩ oraz ONLINE ⟨d⟩ [NORETRY]

W przypadku użycia dyrektywy /OFF wymienione w parametrze urządzenie zostaje postawione w stan nieoperatywny. Polega to na tym, że po zakończeniu obsługi operacji, która była wykonywana na urządzeniu w momencie podania komendy, wszystkie inne operacje zakolejkowane do tego urządzenia zostają wstrzymane aż do momentu przywrócenia urządzenia w stan operatywny, co umożliwia komenda /ONL.

Opcjonalny parametr NO komendy /ONL ma znaczenie w przypadku, gdy stan nieoperatywny urządzenia został wywołany przez błąd operacji wejścia/wyjścia. Przywrócenie stanu operatywnego w takim przypadku przy pomocy komend /ONL bez parametru NO spowoduje, że operacja, która wywołała błąd, zostanie ponowiona w pierwszej kolejności. Użycie parametru NO spowoduje, że operacja taka nie będzie ponowiona, a system przyjmie, że została ona zakończona (bez względu na stan faktyczny).

Standardowo systemy operacyjne SOM umożliwiają wykorzystywanie pojedynczego urządzenia (np. drukarki) przez wiele zadań. Operacje we/wy kierowane do danego urządzenia przez różne zadania są kolejkowane i wykonywane w kolejności określonej przez system prioryterów. Operator może ten stan rzeczy zmienić przy pomocy dyrektywy

TAKE ⟨s⟩ [⟨s⟩] ...

która powoduje, że urządzenia przydzielone wskazanym w dyrektywie strumieniom zostają zarezerwowane na wyłączny użytek podanego w części adresowej zadania. Operacje zakolejkowane do danego urządzenia ulegają wstrzymaniu aż do momentu odwołania rezerwacji przy pomocy dyrektywy

GIVE ⟨s⟩ [⟨s⟩] ...

która przywraca standardowy tryb pracy. Należy pamiętać, że zarówno próba rezerwacji urządzenia już zarezerwowanego jak i próba odwołania rezerwacji urządzenia, które nie było zarezerwowane uprzednio dla podanego w dyrektywie zadania, są sygnalizowane jako błąd.

3.3.5. Dyrektywy sterujące wykonaniem zadań

Dyrektywą, przy pomocy której operator może dokonać aktywacji określonego zadania jest dyrektywa postaci:

{ACTIVATE / EXECUTE / E} [⟨s⟩ [⟨prior⟩]]

Aktywacji podlega zadanie wymienione w części adresowej dyrektywy. Jeżeli zadanie to było już zarejestrowane w kolejce prac systemu to parametry dyrektywy nie mają żadnego znaczenia. W przeciwnym przypadku zadanie zostanie zarejestrowane z podanym poziomem priorytetu i system rozpocznie jego ładowanie ze strumienia określonego pierwszym parametrem. Jeżeli w tej drugiej sytuacji operator pominie podanie parametrów, to system nada zadaniu priorytet równy 128 i podejmie próbę jego załadowania z globalnego strumienia LM. Należy zwrócić uwagę, że podana ogólna postać dyrektywy definiuje w praktyce trzy dyrektywy, które pozwalają osiągnąć opisany efekt. Są to dyrektywy /ACT, /EXE oraz /E. Jeżeli operator chce, aby aktywacja zadania nie kończyła procesu wczytywania komend powinien użyć dyrektywy /ACT gdyż dyrektywy /EXE oraz /E powodują zawieszenie zadania komunikacji. Analogiczna sytuacja ma miejsce w przypadku trzech komend, którymi operator może posłużyć się celem przerwania i zakończenia wykonywania zadania przez system, których ogólną postać można zapisać w postaci jednej definicji:

W tym przypadku dyrektywy /ABO oraz /A powodują dodatkowo zakończenie wprowadzania dyrektyw przez zadanie komunikacji. Natomiast podstawową funkcją każdej z nich jest spowodowanie odrzucenia zadania przez system operacyjny SOM. Opcjonalny parametr dyrektywy ustawia w pożądany stan bit DUMP słowa opcji zadania, który steruje listingiem usuwanego zadania wyprowadzanym na globalny strumień DO. Brak tego parametru pozostawia bit tej opcji niezmieniony. Usuwając określone zadanie operator tym samym przerywa działanie programu aktualnie wykonywane w ramach tego zadania bez możliwości jego wznowienia. Specyfiką systemów SOM jest to, że pozwalają one użytkownikowi na ingerencję w proces już rozpoczętego wykonywania programu. Tym samym wymienione wyżej dyrektywy w tych systemach stanowią jedyną możliwość przerwania i zakończenia realizacji błędnie działającego programu użytkownika (który np. się "zapętlił"). Czasowe zawieszenie wykonywania określonego zadania (a tym samym i zawieszenie wykonywania określonego programu użytkownika) umożliwia dyrektywa operatorska postaci:

HOLD

Wykonanie tej dyrektywy powoduje dodatkowo zawieszenie zadania komunikacji z jednoczesną rezerwacją urządzenia przydzielonego do strumienia globalnego OC aż do momentu naciśnięcia przycisku OPRQ. Wskazane w części adresowej tej dyrektywy zadanie pozostaje zawieszone (w tym czasie możliwe jest wykonanie określonych czynności - np. zmiana papieru w drukarce) aż do momentu jego wznowienia przez operatora komendami

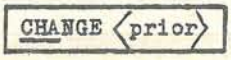
RELEASE lub RESUME

Potrzeba wznowienia wykonania zadania zachodzi również w sytuacji,

gdy zostało ono z określonych powodów zawieszone przez system operacyjny. Różnica pomiędzy komendami /REL oraz /R jest ponownie taka, że /R kończy wprowadzanie komend przez zadanie komunikacji, a /REL nie.

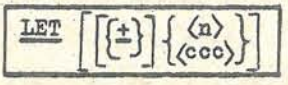
W prezentowanej grupie dyrektyw zadania komunikacji pozostały nam jeszcze do omowienia komendy, które umożliwiają operatorowi ingerencję w sposób realizacji zadania przez system.

Pierwsza z nich postaci



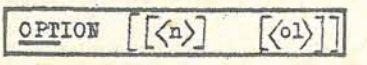
pozwala operatorowi zmienić priorytet zadania wskazanego w części adresowej na wartość wyliczaną w oparciu o podany parametr.

Poza priorytetem operator może zmieniać wartość słowa komunikacji i słowa opcji w bloku kontrolnym określonego zadania. Zmianę słowa komunikacji umożliwia dyrektywa /LET o ogólnej postaci:



Opcjonalny parametr tej dyrektywy może mieć postać liczby całkowitej o wartości od 0 do 32676 w zapisie dziesiętnym lub szesnastkowym albo ciągu znaków należących do kodu CAN oznaczonego w powyższej definicji jako ⟨ccc⟩ . Trzy pierwsze znaki tego ciągu stanowią podstawę do wyliczenia nowej wartości słowa komunikacji. Tak więc użycie komendy /LET z parametrem spowoduje zmianę dotychczasowej zawartości słowa komunikacji na podaną. W praktyce, dyrektywa ta najczęściej służy do ingerowania w proces wykonywania programu pod kontrolą interpretera języka BASIC.

Ustawienie bitów słowa opcji na pożądaną wartość umożliwia komenda



Dyrektywa ta umożliwia ustawienie słowa opcji na wartość określoną parametrem numerycznym ⟨n⟩ albo zmianę stanu poszczególnych bitów

opcji zgodnie z listą opcji <ol> . Na liście opcji wymieniane są nazwy bitów opcji zgodnie z konwencją opisaną w punkcie 3.2.3.

Przykłady:

/JOB/OPT FFFF NOMA - ustawienie wszystkich opcji za wyjątkiem opcji MA;

/JOB/OPT LO - w słowie opcji zadania JOB bit opcji LO będzie miał wartość jeden.

Pominięcie parametrów w dyrektywach /LET oraz /OPT spowoduje wypisanie na konsoli operatorskiej aktualnej wartości, odpowiednio, słowa komunikacji lub słowa opcji. W obydwu przypadkach wydruk będzie miał następującą postać:

- ddddd - wartość słowa w postaci liczby całkowitej,
  hhhh - wartość słowa w postaci liczby szesnastkowej,
  ccc - interpretacja wartości słowa jako trzech znaków w kodzie CAN.

3.3.6. Uzyskiwanie informacji o zadaniach

Za pośrednictwem zadania komunikacji operator może uzyskać określone informacje dotyczące zadań oraz zasobów systemu. Do tego celu służą dwie komendy, a mianowicie komenda /REP oraz komenda /INF. Komenda /REP służy do otrzymywania informacji o zasobach globalnych systemu i jej składnia wygląda następująco:

REPORT <podmiot raportu>

Część adresowa w przypadku tej komendy nie ma żadnego znaczenia, a jedyny parametr jest słowem określającym podmiot raportu. Postać i znaczenie tego parametru zdefiniowane w systemie jest następujące:

TASKS - raport o zadaniach zarejestrowanych w systemie,
CORESTORAGE - raport o podziale pamięci operacyjnej,

DEVICES - raport o urządzeniach,

TIMERS - raport o zegarach systemowych.

Analogiczną składnię ma dyrektywa /INF, która umożliwia uzyskanie informacji o zasobach zadania określonego przez część adresową komendy.

INFORM ⟨podmiot raportu⟩

Parametrem określającym podmiot raportu dla tej dyrektywy może być jedno ze słów (istotne tylko podkreślone znaki):

PARTITIONS - uzyskanie informacji o urządzeniach i sekcjach dyskowych,

STREAMS - uzyskanie informacji o strumieniach zadania aktualnie zdefiniowanych w bloku jego zasobów.

Przykłady:

/REP TAS - na strumień CO (konsolę operatorską) zostanie wypisana informacja o zadaniach użytkowych za rejestrownych w kolejce prac systemu,

/JOB/INF STR - operator uzyska wydruk zawierający wykaz strumieni zdefiniowanych w bloku zasobów zadania JOB. Dla każdego strumienia podana będzie jego nazwa oraz określony jego przydział aktualny i standardowy. Wolne pozycje w tablicy definicji strumieni sygnalizowane będą napisem VACANT.

3.3.7. Sygnalizacja błędów systemowych

Poprzez zadanie komunikacji system operacyjny informuje operatora o trzech kategoriach błędów. Należą do nich:

1. Błędy zaistniałe w trakcie realizacji dyrektyw operatorskich.

W tym przypadku komunikat o błędzie ma postać:

! <err> [<s>]

gdzie: err - trzyliterowy kod przyczyny błędu, s - nazwa strumienia,

która pojawia się w komunikacie, jeżeli błąd wystąpił w dyrektywie związanej z operacjami we/wy. Wykaz kodów err i ich znaczenie zamieszczany jest w tablicy 3.3.

2. Błędy zaistniałe w trakcie realizacji operacji wejścia/wyjścia.

W tym przypadku możliwe są dwie sytuacje:

a) Urządzenie przechodzi w nieoperatywny stan OFF-LINE, o czym operator jest powiadamiany komunikatem postaci

! ttt (ooo) sss=ddd OFF-LINe rrr ccc aaa bbb

gdzie: ttt - nazwa zadania, ooo - nazwa ostatniej nakładki łańcuchowej zadania (programu), sss - nazwa strumienia ddd - nazwa obiektu przydzielonego do strumienia, ccc - identyfikator operacji w trakcie, której zaistniał błąd /REA, WRI, REW, AVF, BSF, AVR, BSR, EOF, HOM/, aaa - adres absolutny ekstrakodu wejścia/wyjścia, bbb - adres bazowy zadania, rrr - adres słowa stanu operacji. Adresy: aaa, bbb oraz rrr są podawane szesnastkowo.

Wszystkie operacje zakolejkowane do wymienionego w komunikacie urządzenia ddd zostają wstrzymane, aż do momentu, gdy operator przywróci temu urządzeniu stan operatywny komendą /ONL.

b) Sytuacja druga ma miejsce w przypadku, gdy błąd współpracy z urządzeniem zaistniał w trakcie realizacji jednej z komend sterujących operacjami we/wy opisanymi w punkcie 3.3.2.

W tym przypadku stan operatywności urządzenia nie ulega zmianie, wykonywana operacja zostaje zakończona, a operator jest powiadamiany o zaistnieniu błędu komunikatem:

! sss HOLD rrrr

gdzie: sss - nazwa strumienia, rrr szesnastkowa wartość słowa odpowiedzi podsystemu wejścia/wyjścia.

Równocześnie zadanie komunikacji zostaje zawieszone, co automatycznie oznacza przerwanie aktualnie wykonywanej dyrektywy.

Tablica 3.3

Wykaz kodów przyczyny powstania błędów systemowych

Kod przyczyny błędu	Objaśnienie
ARG	Nieznormalizowany argument operacji zmiennoprzecinkowej lub dzielenie przez zero
ASG	Próba przydziału strumienia do nieistniejącego w systemie lub do niedostępnego dla danego zadania obiektu
BUL	Błędna próba allokacji pamięci dyskowej. Niedozwolona lub powtórzona nazwa sekcji dyskowej
CHS	Błąd sumy kontrolnej w trakcie ładowania programu do pamięci
COR	Błędna identyfikacja obszaru roboczego pamięci operacyjnej przy próbie allokacji
CRE	Niedopuszczalna nazwa pakieru dyskowego
DIV	Nadmiarowy iloraz w wyniku dzielenia stałoprzecinkowego
ENG	Próby zmiany warunków pracy podsystemu we/wy przy niezakończonej operacji we/wy (np. próba zmiany przydziałów strumienia)
EXC	Próba przekroczenia ilości zasobów w bloku zasobów zadania
FNC	Błędny kod słowa kontrolnego w ładowanym programie
IBR	Niepoprawny fizyczny rekord binarny w ładowanym programie
ILA	Próba przydziału strumienia do nieprzydzielonego (lub przydzielonego do innego z kolei) strumienia we/wy
ILF	Niedopuszczalna lub nieistniejąca nazwa strumienia
ILM	Błędna struktura modułu ładowania
ILS	Nieistniejący numer przerwania
ILT	Nieistniejąca nazwa zegara programowego
IOR	Niedozwolona operacja na strumieniu we/wy
LEV	Niedopuszczalna wartość priorytetu
MEM	Odwołanie poza granice bloku pamięci
OPR	Zadanie odrzucone przez operatora
OVF	Wystąpienie nadmiaru zmiennoprzecinkowego
PAR	Błąd parzystości przy odczycie z pamięci operacyjnej
PRI	Próba wykonania zastrzeżonego ekstrakodu przez nieuprawnione do tego zadanie
RES	Nielegalna próba rezerwacji urządzenia
REX	Nielegalny rozkaz lub ekstrakod
SEQ	Błąd sekwencji numeracji rekordów ładowanego modułu

cd. tablicy 3.3

Kod przyczyny błędu	Objaśnienie
TIM	Błędne parametry czasowe zegara programowego
UAS	Operacja na nieprzydzielonym strumieniu we/wy
UNF	Utrata dokładności (podmiar) w trakcie operacji zmiennoprzecinkowej
ZAP	Błędny adres negatywnego wyjścia z ekstrakodu

Źródło: /4/.

3. Systemowa kontrola zadania prowadzi do "odrzucenia" zadania przez system w przypadku gdy nie istnieje możliwość jego kontynuacji. Operator w takim przypadku jest powiadamiany o zaistniałym błędzie poprzez systemowy wydruk na konsoli operatorskiej mający postać:

! ttt (ooo) ABORT err aaaa bbbb

gdzie: ttt - nazwa odrzuconego zadania, ooo - nazwa ostatnio realizowanego programu w trakcie tegoż zadania, err - trójliterowy kod przyczyny błędu, aaaa - szesnastkowy, absolutny adres miejsca powstania błędu, bbbb - szesnastkowy, absolutny adres bazowy ciała zadania.

Zwykle, w przypadku odrzucenia zadania obsługi terminala użytkownika równocześnie ma miejsce przełączenie urządzeń terminala (klawiatura, ekran lub drukarka) w stan nieoperatywny. Po przywróceniu urządzeń w stan operatywny, operator może wznowić działanie takiego zadania komendą /R.

4. WYBRANE ZAGADNIENIA UŻYTKOWANIA

Podstawowym celem niniejszego rozdziału jest dostarczenie użytkownikowi niezbędnego kompendium wiedzy umożliwiającego Mu samodzielne prowadzenie prac obliczeniowych w systemie MERA-400 przy wykorzystaniu języków programowania Fortran oraz BASIC. Języki te są jedynymi językami wyższego rzędu, których translatory producent systemu włączył w skład oprogramowania podstawowego. Obydwa języki w wersji odpowiadającej ich translatorom w systemie MERA-400 są opisane w rozdziałach piątym i szóstym.

W tym rozdziale zostaną omówione te programy systemowe, z którymi każdy użytkownik realizujący swe prace przy użyciu wyżej wymienionych języków musi umieć się posługiwać. Należą do nich:

- interpreter języka zleceń użytkownika nazywany dalej interpreterem komend (w skrócie IK),
- edytor tekstowy,
- translatory,
- program łączący,
- programy tworzenia i obsługi bibliotek programów użytkownika.

Z uwagi na to, że praca interakcyjna jest podstawowym trybem użytkowania minikomputera MERA-400 wymienione wyżej programy za wyjątkiem translatorów interpretują pewien zbiór komend sterujących ich pracą. Uwzględniając ten fakt można stwierdzić, że zakres wiedzy na temat każdego z procesorów systemowych, bo takim mianem będziemy je dalej określać, jaką użytkownik winien opanować musi obejmować:

- przeznaczenie procesora,

- sposób jego wywołania,
- sposób korzystania przez procesor ze strumieni we/wy zadania,
- postać i znaczenie komend sterujących jego pracą,
- znaczenie komunikatów diagnostycznych emitowanych przez procesor.

Powyższe wyliczenie określa schemat prezentacji poszczególnych procesorów przyjęty w niniejszym rozdziale.

Ogólna postać komend sterujących pracą poszczególnych procesorów (za wyjątkiem edytora tekstowego EDM) zbliżona jest do postaci komend operatorskich akceptowanych przez zadanie komunikacji i omówionych w rozdziale trzecim. Podobnie jak w tamtym przypadku można w niej wyróżnić dwie główne składowe: słowo kluczowe identyfikujące w sposób jednoznaczny komendę oraz listę parametrów, która w jednych komendach musi zostać określona, w innych wolno ją opuścić, a są również i takie komendy, które z definicji nie wymagają określenia żadnych parametrów. W charakterze separatorów oddzielających słowo kluczowe od listy parametrów jak również i poszczególne parametry na liście wolno używać zamiennie znaków: odstępu, przecinka, kreski ukośnej "/" lub znaków równości. Z oczywistych powodów w komendach tych, inaczej aniżeli w przypadku komend operatorskich, nie będzie nigdy występowało pole adresowe określające zadanie, jako że zadanie do którego odnoszą się komendy jest zdeterminowane przez terminal, z którego są one wprowadzane.

Przy prezentacji poszczególnych komend będziemy posługiwać się regułami opisu analogicznymi do podanych w punkcie 3.3.1, które tutaj pokrótce powtórzymy raz jeszcze. I tak:

1. Symbolami metajęzykowymi, które służyć będą tylko do zapisu ogólnej definicji komendy będą:

   < > - ograniczniki określeń metajęzykowych,
   [ ] - nawiasy wskazujące opcjonalne elementy dyrektywy,
   { } - nawiasy wyboru jednego z podanych w nich elementów,
   ... - wielokrotne powtórzenie poprzedzającego wielokropek elementu definicji.

2. Wszystkie znaki nie ujęte w ograniczniki określeń metajęzykowych winny być powtórzone bez żadnych zmian we wprowadzanej dyrektywie. Jedyne dopuszczalne odstępstwo od tej zasady dotyczy jedynie separatorów. W definicjach w charakterze separatorów będziemy używać przecinków, które mogą zostać zamienione na inne znaki separujące, które zostały wymienionej wyżej.
3. Większość procesorów identyfikuje komendę na podstawie trzech pierwszych znaków słowa kluczowego. Dlatego też przy definiowaniu komend słowa kluczowe będą podawane w minimalnej postaci dopuszczalnej przez poszczególne procesory.
4. Dla najczęściej pojawiających się określeń metajęzykowych przyjmiemy następujące oznaczenie, pozwalające zapisywać definicje komend w bardziej zwartej postaci:

⟨s⟩ - nazwa strumienia wejścia/wyjścia,
⟨d⟩ - nazwa urządzenia,
⟨f⟩ - nazwa pliku,
⟨sd⟩ - nazwa urządzenia lub nazwa strumienia,
⟨fsd⟩ - nazwa pliku, strumienia lub urządzenia,
⟨n⟩ - liczba całkowita z przedziału od -32676 do +32676 w zapisie dziesiętnym lub szesnastkowym (w tym drugim przypadku zapis wartości należy poprzednić znakiem #),
⟨ol⟩ - lista nazw opcji zadania (patrz punkt 3.2.3),
⟨p⟩ - nazwa programu,
⟨u⟩ - nazwa użytkownika,
⟨txt⟩ - dowolny tekst (najczęściej komentarz).

Należy pamiętać, że wszystkie nazwy (identyfikatory) pojawiające się w komendach muszą być zbudowane ze znaków należących do zbioru znaków kodu CAN (patrz punkt 3.3.1).

Użytkownik w trakcie pracy interakcyjnej wszystkie informacje, w tym również komendy sterujące pracą procesorów, wprowadza przy pomocy klawiatury terminala. Wprowadzany rekord informacji jest wstępnie

umieszczany w buforze wejściowym, którego zawartość system wypisuje na urządzeniu listującym terminala. Jeżeli urządzeniem listującym jest monitor ekranowy, to aktualna pozycja wprowadzania w obrębie bufora wejściowego jest zaznaczana tak zwanym kursorem, którym jest albo znak podkreślenia albo plamka świetlna w zależności od typu monitora.
W trakcie wprowadzania informacji z klawiatury specjalną interpretację mają następujące znaki:

"CTRL H" - cofanie kursora o jeden znak (kasowanie ostatnio wprowadzonego znaku),

"CTRLX" - kasowanie całej zawartości bufora wejściowego,

"CR" - przesłanie zawartości bufora wejściowego do systemu (np. akceptacja wprowadzanego rekordu dyrektywy).

Uzyskanie znaku specjalnego opisanego jako "CTRL y" wymaga jednoczesnego naciśnięcia dwóch klawiszy: klawisza oznaczonego symbolem "CTRL" na klawiaturze oraz klawisza znaku "y". Klawisz akceptacji "CR" jest na niektórych klawiaturach oznaczany napisem "Ret" lub "Return".

W przypadku, gdy terminalem jest stacja PSPD-90, która posiada niestandardowy ekran oraz niestandardową klawiaturę należy dodatkowo uwzględnić następujące uwagi:

1. Opisane wyżej znaki specjalne są dostępne za pośrednictwem następujących klawiszy:

   "CTRL H" - klawisz oznaczony symbolem "CH",

   "CTRL X" - klawisz "FL",

   "CR" - klawisz opisany jako  .

   Dodatkowo, w trakcie wprowadzania komend pod kontrolą interpretera języka zleceń użytkownika (interpretera komend) specjalne znaczenie mają następujące znaki:

   "SHF CH" - przesunięcie kursora o jeden znak do przodu w obrębie bufora wejściowego,

   "SHF FL" - kasowanie zawartości bufora wejściowego od kursora do końca.

Podobnie jak uprzednio opis "SHF CH" oznacza konieczność naciśnięcia dwóch klawiszy "SHF" oraz "CH". Podobna uwaga odnosi się do oznaczenia "SHF FL".

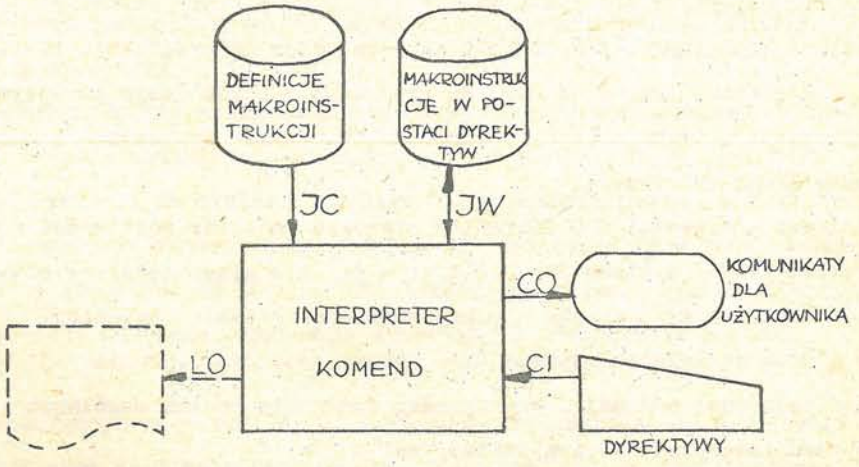
2. Przytrzymanie naciśniętego klawisza powoduje, że po krótkiej chwili znak będzie wysyłany wielokrotnie, aż do momentu zwolnienia klawisza.
3. Rekord logiczny, z uwagi na wąski ekran, może się składać z kilku linii fizycznych ekranu. Przejście do następnej linii fizycznej następuje automatycznie w trakcie pisania.
4. Z uwagi na niestandardową klawiaturę należy w trakcie pracy uwzględniać odpowiedniość kodów znaków określoną w rozdziale drugim w punkcie 2.8.

4.1. Interpretery komend systemowych

Interpreter komend systemowego języka zleceń użytkownika jest uprzeywilejowanym procesorem systemowym w tym sensie, że jest ona ładowany automatycznie z chwilą wczytania do pamięci systemu operacyjnego. Zasadniczym przeznaczeniem tego procesora jest umożliwienie użytkownikowi sterowanie wykonaniem programów (zarówno własnych jak i systemowych) w obszarze pamięci przydzielonym dla zadania obsługującego jego terminal. Działanie interpretera komend kończy się z chwilą polecenia użytkownika wykonania określonego programu. Wywołany program jest ładowany do pamięci operacyjnej jako nakładka łańcuchowa ciała zadania użytkownika i zostaje mu przekazane sterowanie. Po zakończeniu pracy wywołanego programu system operacyjny automatycznie wznawia działanie interpretera komend. Wynika stąd, że dyrektywy akceptowane przez interpreter komend stanowią podstawową i najważniejszą składową języka pracy interakcyjnej użytkownika pracującego pod kontrolą systemów SOM minikomputera MERA-400.

Interpretery komend systemów SOM-3 oraz SOM 3.P różnią się między sobą zarówno co do ilości dostępnych komend (a co za tym idzie oferowanych przez system użytkownikowi możliwości działania) jak i ich

postaci oraz sposobie sygnalizacji gotowości do przyjęcia kolejnej dyrektywy. Interpreter komend systemu SOM-3 nosi nazwę JOB CONTROL podczas gdy analogiczny procesor systemu SOM 3.P nie ma specjalnie określonej nazwy i dlatego w dalszym ciągu będziemy go określać mianem interpretera komend (w skrócie IK).



Rys. 4.1. Przepływ informacji w trakcie pracy interpretera komend Źródło: /4/.

Podstawowymi objawami zewnętrznymi, które różnią obydwa interprety pomiędzy sobą są:

1. JOB CONTROL zgłasza swoją gotowość do przyjęcia kolejnej komendy użytkownika poprzez wypisanie na urządzeniu listującym terminala pojedynczego znaku ' (apostrof) w nowej linii podczas gdy IK w analogicznej sytuacji wypisuje w nowej linii pojedynczy znak kropki.
2. Słowo kluczowe każdej komendy JOB CONTROL'a musi być zawsze poprzedzone znakiem dolara ($). W przypadku interpretera systemu SOM 3.P poza dyrektywą komentarza $NOP pozostałych komend nie wolno poprzedzać znakiem dolara.
3. JOB CONTROL każdą przyjętą dyrektywę listuje na strumieniu LO

poprzedzając ją, w przypadku stwierdzenia błędu syntaktycznego, jednym z następujących komunikatów:

ERR R - dyrektywa nie jest poprzedzona znakiem $,

ERR N - nieznane słowo kluczowe dyrektywy,

ERR S - niedozwolony znak w słowie kluczowym (spoza kodu CAN),

podczas gdy IK nie korzysta ze strumienia LO a błędy składniowe kwituje komunikatem BAD COMMAND wyprowadzanym na urządzenie listujące terminala (strumień CO). Na rysunku 4.1 pokazującym przepływ informacji w trakcie pracy interpreterów różnica ta została zaznaczona linią przerywaną.

Ponieważ możliwości JOC CONTROL'a stanowią podzbiór możliwości interpretera komend systemu SOM 3.P więc w dalszym ciągu będziemy omawiać je wspólnie zaznaczając w miarę zaistnienia takiej potrzeby, które z nich są wyłączną domeną tego ostatniego. Jednocześnie, aby uniknąć podwójnej narracji, w przypadku omawiania komend wspólnych dla obydwu interpreterów przyjmiemy, że:

a) Postać komendy będzie podawana w wersji odpowiadającej interpreterowi systemu SOM 3.P.

b) Zamiast pełnych słów kluczowych podawane będą jedynie ich trzy pierwsze znaki, istotne dla obydwu interpreterów. Znaki tworzące dopuszczalny skrót słowa kluczowego dla interpretera systemu SOM 3.P będą podkreślane.

4.1.1. Standaryzacja stanu pracy interpretera komend

Pierwszą dyrektywą użytkownika rozpoczynającego pracę powinna być dyrektywa standaryzująca stan interpretera komend. W przypadku JOB CONTROL'a komenda ta ma postać

$ JOB [⟨txt⟩]

gdzie opcjonalny parametr ⟨txt⟩ jest dowolnym komentarzem użytkowni-

ka, a analogiczne w działaniu polecenie interpretera SOM 3.P ma postać

Q lub Q!

Efektem wykonania wymienionych dyrektyw będzie:

- ustawienie standardowej wartości słowa opcji zadania,
- wyzerowanie słowa komunikacji,
- przyłączenie strumieni we/wy do ich standardowych obiektów przydziału,
- przywrócenie warunków początkowych interpretera komend, a tym samym i zadania użytkownika,
- przejście na nasłuch dyrektyw wprowadzanych przez użytkownika.

W przypadku interpretera JOB CONTROL dyrektywa $JOB jest odrukowywana na strumieniach LO oraz CO. Interpreter komend systemu SOM 3.P potwierdza natomiast wykonanie dyrektywy Q komunikatem postaci

SYS. INIT

wyprowadzanym na strumień CO przyłączony standardowo do urządzenia listującego terminala użytkownika. W przypadku tego interpretera użycie wersji dyrektywy Q z wykrzyknikiem powoduje dodatkowo fizyczne przeładowanie interpretera komend z dysku systemowego.

Nawiązując do charakterystyki współpracy użytkownika z systemem przedstawionej w punkcie 3.12 należy stwierdzić, że wymienione wyżej komendy powodują otwarcie nowej sesji pracy interakcyjnej (a tym samym zamknięcie poprzedniej). Użytkownik może reinicjować pracę interpretera komend w dowolnym momencie. Niemniej jednak konieczność otwarcia nowej sesji przy pomocy komendy $JOB (lub Q) ma miejsce zawsze wtedy, gdy w wyniku błędnej pracy jakiegoś programu nastąpi odrzucenie zadania przez system, o czym użytkownik jest powiadamiany komunikatem ABORT (patrz punkt 3.3.7). Odrzucenie zadania przez system jest równoznaczne bowiem z zakończeniem sesji interakcyjnej i dopóki użytkownik nie otworzy nowej, interpreter komend będzie ignorował wprowadzane dyrektywy.

4.1.2. Ładowanie i wykonanie programu

Polecenie, przy pomocy którego użytkownik może spowodować załadowanie do pamięci operacyjnej określonego programu i rozpoczęcie jego wykonania ma postać komendy o ogólnej definicji:

EXE, ⟨p⟩ [, [⟨s⟩][, ⟨ol⟩]]

Wykonanie dyrektywy EXE polega na załadowaniu do pamięci operacyjnej programu ⟨p⟩ ze strumienia wskazanego przez parametr ⟨s⟩ i przekazanie sterowania załadowanemu programowi, czyli rozpoczęcie jego wykonania. Jeżeli w strumieniu zadeklarowanym w dyrektywie nie ma programu, o podanej nazwie, to sygnalizowany jest błąd komunikatem ERR E poprzedzającym wydruk przyjętej dyrektywy na listingu, a jednocześnie zapalony zostaje wskaźnik FLAG słowa komunikacji i sterowanie powraca do interpretera komend. Jeżeli w takiej sytuacji opcja GO jest nieaktywna, to następna dyrektywa EXE spowoduje odrzucenie (abort) zadania, a tym samym automatycznie zamknięcie bieżącej sesji pracy interakcyjnej i system przejdzie w stan oczekiwania na otwarcie nowej sesji przy pomocy dyrektywy $JOB lub Q (w zależności od systemu operacyjnego).
W przypadku poprawnego załadowania programu do pamięci operacyjnej przed rozpoczęciem jego wykonania może ulec zmianie stan słowa opcji stosownie do listy nazw opcji ⟨ol⟩ wymienionej w dyrektywie przez użytkownika. Jeżeli lista opcji ⟨ol⟩ nie zostanie podana to stan słowa opcji nie ulega zmianie. Pominięcie nazwy strumienia ⟨s⟩ w dyrektywie EXE spowoduje że wymieniony program poszukiwany będzie w strumieniu LMB, który standardowo jest dołączony do sekcji dyskowej przechowującej bibliotekę procesorów systemowych.

Przykłady:

EXE,TEST1,BI,U1,U2,NOLO - Ze strumienia BI zostanie załadowany program o nazwie TEST1. Przed rozpoczęciem jego wykonania zostaną zapalone opcje U1 i U2 oraz zostanie zgaszona opcja LO

EXE,FOR,,NOLO,NOMA - Ze strumienia LMB zostanie załadowany program FOR i rozpocznie się jego wykonanie przy wyzerowanych (zgaszonych) bitach opcji LO oraz MA

E,EDM - Załadowanie i wykonanie programu EDM, który winien znajdować się na strumieni LMB. Żadna z opcji przed wykonaniem programu nie zostanie zmieniona.

Stosownie do uwag ogólnych podanych na początku punktu 4.1 postać komend w powyższych przykładach odpowiada interpreterowi komend systemu SOM 3.P. Prawidłowa postać tych komend w systemie SOM-3 jest następująca:

```
$EXE,TEST1,BI,U1,U2,NOLO
$EXE,FOR,,NOLO,NOMA
$EXE,EDM
```

W obydwu systemach pominięcie nazwy strumienia w sytuacji gdy występuje lista opcji polega na podaniu dwu ograniczników obok siebie (pusty parametr) tak jak to ma miejsce w drugim przykładzie. Jeżeli natomiast lista opcji nie występuje i nazwa strumienia ma być pominięta, to linia komendy EXE kończy się po nazwie programu (przykład trzeci). W praktyce zwykle w charakterze separatora po słowie kluczowym znacznie częściej używa się znaku spacji aniżeli przecinka. Tym samym komendy E,EDM oraz E EDM są dokładnie równoważne.

Program, którego nazwa jest pierwszym parametrem komendy EXE jest standardowo ładowany od początku obszaru pamięci zajętej przez ciało zadania, a więc w miejsce interpretera komend. Jeżeli zachodzi taka potrzeba, to użytkownik może zmienić ten stan rzeczy przy pomocy komendy

LOC [$] ⟨n⟩

gdzie ⟨n⟩ jest adresem w pamięci operacyjnej podanym dziesiętnie lub szesnastkowo, od którego ma być ładowany program przy

pomocy następnej dyrektywy EXE. Użycie przed adresem znaku dolara ($) powoduje, że program będzie ładowany od komórki o numerze określonym przez ⟨n⟩ ale licząc od początku obszaru pamięci zajętej przez ciało zadania (adres względny). W przeciwnym przypadku wartość parametru ⟨n⟩ określa adres bezwzględny w pamięci operacyjnej.

Przykłady:

LOC B000 - Adres absolutny w zapisie szesnastkowym

LOC 1400 - Adres absolutny w zapisie dziesiętnym

LOC $100 - Względny adres ładowania oznaczający setną komórkę od początku obszaru pamięci zajętej przez zadanie.

W typowych pracach obliczeniowych rzadko zachodzi uzasadniona potrzepotrzeba zmiany standardowego adresu ładowania programów. Znacznie częściej użytkownik odczuwa potrzebę ustawiania bitów słowa opcji. Można to zrobić przy okazji polecenia wykonania programu komendą EXE w sposób omówiony wyżej. Nie mniej jednak istnieje również specjalna komenda, która umożliwia ustawienie określonych bitów słowa opcji na pożądaną wartość. Jest nią dyrektywa postaci:

OPT ⟨ol⟩

gdzie ⟨ol⟩ jest listą nazw opcji omówionych w punkcie 3.2.3. Przypomnijmy, że jeżeli nazwa opcji na liście jest ppprzedzona przedrostkiem NO, to odpowiadający jej bit w słowie opcji zostanie wyzerowany (zgaszony). Nazwa opcji bez tego przedrostka spowoduje ustawienie odpowiadającego jej bitu na wartość jeden.

Przykład:

OPT U1,GO,NOLO - ustawione zostaną opcje U1 oraz GO a bit opcji LO zostanie wyzerowany

4.1.3. Komendy odnoszące się do strumieni wejścia/wyjścia

Komenda EXE jest najsilniejszą i najważniejszą dyrektywą interpretera komend. Wywołany nią program przejmuje sterowanie maszyną. Po jego zakończeniu system operacyjny reaktywuje automatycznie działanie interpretera komend, chyba że w trakcie wykonania programu wystąpił błąd uniemożliwiający kontynuację bieżącej sesji pracy interakcyjnej. Przed wykonaniem określonego programu użytkowego zachodzi zwykle potrzeba odpowiedniego ustawienia słowa opcji oraz przygotowania strumieni wejścia wyjścia zgodnie z wymogami programu, który ma być wykonany. Sposoby definiowania opcji z poziomu interpretera komend zostały omówione w poprzednim punkcie. Obecnie zapoznamy się z postaciami komend odnoszącymi się do strumieni wejścia/wyjścia. Ich działanie jest analogiczne do działania odpowiadających im dyrektyw operatorskich, które zostały omówione w punktach 3.3.2 oraz 3.3.3. Z tego powodu ich prezentację tutaj ograniczymy do podania ogólnej postaci oraz krótkiego wyjaśnienia realizowanych funkcji. A oto lista tych komend:

ASS, ⟨s⟩ , ⟨sd⟩[, ⟨s⟩ , ⟨sd⟩] ...	- Zmień aktualny przydział strumienia,
DEF, ⟨s⟩ , ⟨sd⟩ [, ⟨s⟩ , ⟨sd⟩] ...	- Zmień standardowy przydział strumienia (tylko pod systemem SOM 3.P),
AVF, ⟨s⟩ [, ⟨n⟩]	- Przesuń nośnik o ⟨n⟩ znaczników końca zbioru do przodu
BSF, ⟨s⟩ [, ⟨n⟩]	- Jak AVF, ale w odwrotnym kierunku
AVR, ⟨s⟩ [, ⟨n⟩]	- Przesuń nośnik o ⟨n⟩ rekordów do przodu
BSR, ⟨s⟩ [, ⟨n⟩]	- Jak AVR ale w odwrotnym kierunku
REW, ⟨s⟩ [, ⟨s⟩] ...	- Przewiń nośnik do początku

WEO, ⟨s⟩ [, ⟨n⟩]	- Wyprowadź na strumień ⟨n⟩ znaczników końca zbioru

We wszystkich komendach, w których parametr ⟨n⟩ jest opcjonalny pominięcie go jest równoznaczne z założeniem, że jego wartość jest równa jeden.

4.1.4. Komunikacja użytkownika z operatorem systemu

W systemach MERA-400 pracujących w zestawie z kilkoma terminalami może zachodzić potrzeba przesłania określonej informacji lub polecenia pod adresem operatora systemu. Taki transfer informacji umożliwiają dwie komendy postaci:

NOT, ⟨txt⟩	lub	ACT, ⟨txt⟩

Obydwie powodują wypisanie na konsoli operatorskiej komunikatu określonego parametrem ⟨txt⟩ z tym, że po przesłaniu komunikatu dyrektywa NOT nie zawiesza zadania użytkownika podczas gdy komenda ACT powoduje zawieszenie zadania umożliwiając tym samym operatorowi podjęcie określonej akcji (np. wymianę pakietu dyskowego).

Przykład:

```
NOT, KONIEC ZAKLADANIA ZBIORU
ACT, ZMIEN PAKIET DYSKOWY
```

Po komendzie ACT zadanie użytkownika może być wznowione tylko poleceniem operatorskim /RESUME.

4.1.5. Podsystem plikowy systemu SOM 3.P

System operacyjny SOM 3.P został, w stosunku do bazowego systemu SOM-3, poszerzony o bardzo istotny z użytkowego punktu widzenia element, którym jest podsystem plikowy. Tworzą go uprzywilejowane programy systemowe realizujące złożone algorytmy dostępu do zbiorów (plików)

umieszczonych przez użytkownika w obrębie swojego katalogu (albo inaczej kartoteki) zbiorów w obrębie tzw. sekcji plikowej. Sekcja plikowa to obszar w pamięci zewnętrznej przechowujący zbiór kartotek użytkowników. Zwykle sekcja plikowa jest umieszczana na kasecie wymiennej dysku sztywnego, ale nic nie stoi na przeszkodzie, aby zlokalizować ją na jednej ze stron dysku elastycznego. W plikach wchodzących w skład katalogu użytkownika może być przechowywana informacja o dowolnej postaci, tzn. pliki te mogą składać się zarówno z rekordów znakowych jak i binarnych. Jedynym wymogiem stawianym przez omawiany podsystem jest to, aby zbiór, który jest kopiowany na sekcję plikową kończył się standardowym znacznikiem końca zbioru.

Programy realizujące operacje na zbiorach przechowywanych w obrębie sekcji plikowej i dostępne z poziomu interpretera za pomocą odpowiednich, omówionych dalej komend umożliwiają użytkownikowi w prosty sposób:

- Tworzyć nowe pliki w obrębie własnej kartoteki zbiorów;
- Modyfikować pliki w obrębie własnego katalogu;
- Kasować własne pliki;
- Kopiować pliki na dowolny strumień we/wy lub bezpośrednio na urządzenie;
- Uzyskiwać informacje o zawartości swojego katalogu zbiorów;
- Uzyskiwać określone informacje o strumieniach i urządzeniach.

Ponieważ sekcja plikowa przechowuje katalogi zbiorów wielu użytkowników więc została zabezpieczona przed nieumyślnym lub intencjonalnym zniszczeniem za pomocą hasła dostępu do operacji na sekcji plikowej jako całości. Hasło to nie jest znane użytkownikom, co w konsekwencji oznacza, że mogą oni operować w obrębie swojej kartoteki, ale nie mogą na przykład uzyskać informacji o innych kartotekach ani tym bardziej ich usunąć. Możliwości te ma tylko osoba znające hasło (zwykle kierownik systemu cyfrowego). Osoba ta ma dostęp do puli tzw. komend specjalnych podsystemu plikowego, które zostaną wykonane tylko po podaniu prawidłowego, aktualnie obowiązującego hasła. Takimi komendami specjal-

nymi w omawianym systemie są: (podkreślone litery stanowią dopuszczalny skrót słowa kluczowego komendy):

REPORT	- Podaj wielkości kartotek wszystkich użytkowników
REPORT!	- Podaj pełne informacje o wszystkich kartotekach sekcji plikowej
NU, ⟨u⟩, ⟨txt⟩	- Zdefiniuj w obrębie sekcji plikowej pusty katalog nowego użytkownika o identyfikatorze ⟨u⟩ . Tekst określony drugim parametrem jest listowany podczas wykonywania dyrektywy RP i aczkolwiek może być dowolny, to zaleca się, aby go rozpoczynać nazwiskiem właściciela otwieranej kartoteki.
ERASE, ⟨u⟩	- Efektem wykonania tej komendy jest usunięcie z sekcji plikowej kartoteki zbiorów użytkownika o identyfikatorze podanym w parametrze dyrektywy.
DU, ⟨u⟩	- Kartoteka użytkownika o podanym identyfikatorze zostanie zamrożona, to znaczy czasowo niedostępna. Kartoteka taka może być odmrożona komendą NU lub usunięta z systemu na stałe komendą ER.
NEWPSW, ⟨hasło⟩	- Przy pomocy tej komendy można zmienić hasło dostępu do sekcji plikowej w sposób umożliwiający wykonanie wyżej wymienionych komend specjalnych. Nowe hasło, które będzie obowiązywać od momentu wykonania tej komendy jest jej jedynym parametrem.

Użytkownik w danym momencie może korzystać tylko z jednej kartoteki plików. Otwarcie kartoteki umożliwia komenda postaci:

LOGIN, ⟨u⟩

Począwszy od tego momentu komendy zawierające jako parametr nazwę pliku f będą dotyczyły kartoteki ⟨u⟩ aż do momentu ponownego użycia komendy LN, co odpowiada otwarciu nowej kartoteki (i automatycznie zamyka dotąd otwartą), lub komendy postaci:

LOGOUT

która zamyka aktualnie otwartą kartotekę bez otwierania nowej. Od tego momentu nie działają niżej omówione komendy systemu plikowego. Należy pamiętać, że otwarcia kartoteki komendą LOGIN można dokonać tylko wtedy, gdy kartoteka ta wcześniej została zdefiniowana komendą specjalną NU i nie jest to kartoteka zamrożona komendą DU.

Po otwarciu kartoteki komendą LOGIN użytkownik może korzystać z następujących komend systemu plikowego:

MAKEFILE, ⟨sd⟩ ,⟨f⟩	- Plik z ⟨sd⟩ (nazwa strumienia lub urządzenia) umieść w kartotece pod nazwą ⟨f⟩ ,
MAKEFILE, ⟨sd⟩, ⟨f⟩	- Kasuj plik ⟨f⟩ w kartotece (o ile taki istnieje). Następnie plik z ⟨sd⟩ umieść w kartotece pod nazwą ⟨f⟩ .
COPY, ⟨fsd⟩, ⟨sd⟩	- Kopiuj plik z ⟨fsd⟩ na ⟨sd⟩ łącznie ze znacznikiem końca zbioru.
COPYNEOF, ⟨fsd⟩, ⟨sd⟩	- Kopiuj plik z ⟨fsd⟩ na ⟨sd⟩ , ale nie zapisuj na ⟨sd⟩ znacznika końca zbioru.
DELETE, ⟨f⟩	- Kasuj plik o nazwie ⟨f⟩ w otwartej kartotece
NEWNAME, ⟨f⟩, ⟨f⟩	- Plikowi podanemu jako pierwszy parametr zmień nazwę na podaną jako drugi parametr
INFORM, ⟨fsd⟩	- W zależności od tego, czym jest podany parametr podaj informację o wielkości: pliku ⟨f⟩ w otwartej kartotece, pierwszym pliku na urządzeniu ⟨d⟩ lub aktualnie dostępnym pliku w strumieniu ⟨s⟩
INFORM	- Podaj informacje statystyczne o zawartości otwartej kartoteki
INFORM!	- Drukuj pełną informację o zawartości aktualnie otwartej biblioteki
V	- Podaj informację o aktualnej wersji interpretera komend
EDM	- Komenda równoważna dyrektywie EXE EDM. Powoduje załadowanie i rozpoczęcie pracy edytora tekstowego EDM.

W celu odróżnienia nazw strumieni i urządzeń od nazw plików w kartotece użytkownika we wszystkich wyżej wymienionych komendach obowiązuje zasada, że te ostatnie muszą być poprzedzane znakiem @.

4.1.6. Mikrokomputer PSPD-90 jako terminal w systemie MERA-400

Do nawiązania transmisji pomiędzy mikrokomputerem PSPD-90 a minikomputerem MERA-400 pracującym pod kontrolą systemu operacyjnego SOM 3.P należy na PSPD-90 uruchomić program o nazwie TRANS. Program ten winien być uruchomiony pod kontrolą systemu MICRODOS-90, FDOS lub równoważnym. Uruchomienie programu TRANS pod wymienionymi systemami uzyskujemy za pomocą komendy

RUN ⟨komora⟩, TRANS

w której ⟨komora⟩ jest parametrem określającym numer komory, w której umieszczono dyskietkę z programem TRANS. Zakończenie pracy programu TRANS (a tym samym pracy PSPD-90 jako terminala w systemie MERA-400) następuje po podaniu komendy

K

Po wystartowaniu programu TRANS pierwszym, wyświetlonym na ekranie terminala rekordem jest komenda SET ustawiająca proponowane parametry transmisji. Może ona zostać zaakceptowana w proponowanej postaci przy pomocy klawisza "CR" lub dowolnie zmodyfikowana zgodnie z podanym dalej opisem.

Pod kontrolą programu TRANS użytkownik ma do dyspozycji trzy grupy komend:

1. Komendy systemu operacyjnego SOM 3.P.
2. Wybrane komendy systemu operacyjnego mikrokomputera PSPD-90.
3. Komendy transmisji plików tekstowych pomiędzy minikomputerem MERA-400 a PSPD-90.

Ad 1. Komendy systemu operacyjnego SOM 3.P akceptowane są w swojej standardowej postaci opisanej w poprzednich punktach.

Ad 2. Do tej grupy komend zaliczymy komendę SET, która pozwala ustawić parametry interpretera programu TRANS obsługującego transmisję oraz cztery komendy systemu operacyjnego MICRODOS-90 (lub FDOS) związane z operacjami na plikach.

Komenda SET ma postać:

SET ⟨parametry⟩

gdzie ⟨parametry⟩ to oddzielona przecinkami lista następujących parametrów podanych w dowolnej kolejności:

DR = ⟨komora⟩ - definiuj podstawową komorę. Komory dyskietek mikro PSPD-90 pod wymienionymi uprzednio systemami operacyjnymi są ponumerowane od 0 do 3. Komora zdefiniowana tym parametrem jest używana jako standardowa we wszystkich niżej opisanych komendach, w których opuszczono parametr ⟨komora⟩.

PG = {A / B} - definiuj stronę dyskietki (A lub B),

CH = ' ⟨znak⟩ - definiuj znak, którym zgłasza swoją gotowość interpreter systemu SOM 3.P,

{N / W} - rodzaj ekranu PSPD-90 (N-wąski, W-szeroki).

Komendy związane z systemem plikowym PSPD-90 mają następującą postać i znaczenie:

DIR [⟨komora⟩] - Listuj wykaz zbiorów dyskietki umieszczonej w komorze o podanym numerze. Wykaz zawiera nazwę pliku, jego typ, adres pierwszego sektora oraz wielkość określoną przez liczbę zajętych sektorów. Listowanie jest stronicowane. Następną stronę uzyskuje się po naciśnięciu klawisza "CR". Klawisz "ESC" powoduje przerwanie listowania.

CRE [⟨komora⟩ ,] ⟨nazwa⟩ - W komorze o podanym numerze utwórz pusty plik o podanej nazwie. Jeżeli plik o takiej nazwie już istnieje to sygnalizowany jest błąd.

DEL [⟨komora⟩ ,] ⟨nazwa⟩ - Usuń plik o podanej nazwie z dyskietki umieszczonej w komorze określonej pierwszym parametrem.

SCR [⟨komora⟩ ,] ⟨nazwa⟩ - Jak komenda DEL, ale wymieniony zbiór pozostaje w indeksie plików na dyskietce jako zbiór pusty.

W przypadku komend DEL oraz SCR interpreter komunikatem

ARE YOU SURE ?

upewnia się, czy komenda ma być wykonana. Odpowiedzią może być Y- jeśli tak lub N- w przeciwnym przypadku.

Ad 3. Dwie komendy umożliwiają wymianę kompletnych plików pomiędzy obydwoma komputerami.

Komenda

AR [⟨komora⟩,]⟨nazwa⟩

powoduje przesłanie pliku tekstowego ze strumienia SI minikomputera MERA-400 na plik, o nazwie określonej drugim parametrem, zlokalizowany na dyskietce umieszczonej w podanej komorze dyskowej mikro PSPD-90. Komenda zostaje wykonana po upewnieniu się (podobnie jak przy komendach DEL i SCR), że polecenie jest poprawne. Jeżeli plik o podanej nazwie nie istnieje, to jest on automatycznie tworzony. Jeżeli taki plik już jest na dyskietce, to użytkownik jest o tym powiadamiany odpowiednim komunikatem. Błąd w trakcie transmisji powoduje jej unieważnienie, ale strumień SI zostaje przesunięty. W trakcie przepełnienia pojemności dyskietki interpreter pyta użytkownika o dalszą akcję.

Transmisja plików tekstowych w odwrotnym kierunku jest efektem wykonania komendy

RV [⟨komora⟩ ,] ⟨nazwa⟩

która określony parametrami plik przesyła z mikro PSPD-90 na strumień SO minikomputera MERA-400. Przebieg realizacji tej komendy jest analogiczny jak w przypadku komendy AR z tym, że ewentualny błąd w trakcie transmisji powoduje zachowanie dotychczas przetransmitowanego pliku (na strumień SO zostaje wyprowadzony znacznik końca zbioru).

4.1.7. M a k r o i n s t r u k c j e

Interpretery komend systemów operacyjnych SOM minikomputera MERA-400 są wyposażone w makroprocesory, które umożliwiają zastąpienie typowego ciągu komend wywołaniem jednej makroinstrukcji. Makroprocesor interpretera komend analizując rekord dyrektywy wprowadzony przez użytkownika sprawdza najpierw, czy jest to jedna z komend interpretera. Jeśli nie, to przegląda strumień JC w poszukiwaniu odpowiedniej makrodefinicji. Jeśli ją znajdzie, to zaczyna ją przetwarzać korzystając przy tym ze strumienia roboczego JW. Makrodefinicję w najprostszym przypadku można uważać za ciąg komend interpretera, w których niektóre elementy zostały zamienione parametrami formalnymi mającymi postać liczby poprzedzonej znakiem % (procent), co możemy przedstawić jako %n. Liczba występująca w parametrze formalnym określa, który z kolei parametr aktualny w wywołaniu makroinstrukcji zastąpi fragment instrukcji opisany danym parametrem formalnym. Parametrów formalnych, a co za tym idzie również aktualnych może być nie więcej aniżeli 16. Makrodefinicja musi rozpoczynać się wierszem postaci:

$PRD ⟨nazwa⟩ [⟨ps1⟩] , [⟨ps2⟩] , ... , [⟨ps16⟩]

gdzie ⟨nazwa⟩ jest nazwą makroinstrukcji, a ⟨psi⟩ standardowymi wartościami parametrów formalnych, które zostaną podstawione za parametry formalne gdy użytkownik w wywołaniu makroinstrukcji nie poda wartości aktualnej dla określonego parametru.

Wywołanie makroinstrukcji ma postać uzależnioną od wersji systemu SOM. W systemie SOM-3 służy do tego celu komenda postaci:

$DO ⟨nazwa⟩ , [⟨pa1⟩] , [⟨pa2⟩] , ... , [⟨pa16⟩]

gdzie nazwa oznacza nazwę wywoływanej makroinstrukcji, a ⟨pai⟩ - i-ty argument aktualny, którego wartość ma zastąpić w makrodefinicji wszystkie wystąpienia parametru formalnego mającego postać %i.

W przypadku systemu SOM 3.P w celu wywołania makroinstrukcji wystarczy podać jej nazwę i listę parametrów aktualnych, co powoduje że postać takiego odwołania

⟨nazwa⟩[⟨pa1⟩], [⟨pa2⟩] , ... , [⟨pa 16⟩]

jest w ogólnym zapisie identyczna z ogólną postacią komend interpretera. Jest to istotne spostrzeżenie w kontekście wstępnych uwag, gdyż wynika stąd, że użytkownik nie powinien w tym systemie używać słów kluczowych komend interpretera jako nazw swoich makroinstrukcji. Taka makrodefinicja jest niedostępna, gdyż interpreter jej nazwę potraktuje jako wywołanie komendy. Ograniczenie to z oczywistej racji nie występuje w systemie SOM-3, w którym wywołanie makroinstrukcji jest funkcją specjalnej komendy $DO.

Wywołanie makroinstrukcji powoduje, że makroprocesor interpretera komend:

- czyta kolejne linie makrodefinicji ze strumienia JC,
- zastępuje wszystkie wystąpienia parametrów formalnych %i odpowiadającymi im (w sensie kolejności) parametrami aktualnymi pai z wywołania makroinstrukcji, a w przypadku braku któregoś z parametrów aktualnych odpowiednią wartością standardową wziętą z linii definicji makroinstrukcji,
- zapisuje wynikową postać na strumieniu JW po czym, po napotkaniu końca makrodefinicji przewija strumień JW od początku i czyta kolejno rekordy traktując je na tych samych zasadach jak rekordy dyrektyw wprowadzane przez użytkownika z klawiatury.

Należy w tym miejscu dodać, że koniec pojedynczej makrodefinicji

jest wyznaczony przez początek następnej makrodefinicji lub znacznik końca zbioru.

Te nieco zawiłe reguły powinny stać się bardziej czytelne po zapoznaniu się z następującym przykładem. Załóżmy, że na strumieniu JC jest dostępna makrodefinicja o postaci:

```
$PRD LISTUJ,BIBL,@ SORT,CSL
LOGIN,%1
ASS,SO,%3
COPY,%2,SO
```

Wywołanie w systemie SOM 3.P tej makroinstrukcji dyrektywą postaci

```
LISTUJ,TEKSTY,@ SORT,CO
```

da ten sam efekt co wywołanie postaci

```
LISTUJ,TEKSTY,,CO
```

gdyż wartość drugiego parametru aktualnego jest identyczna z wartością standardową z linii $PRD makrodefinicji, oraz spowoduje w pierwszym etapie umieszczenie na strumieniu JW zbioru dyrektyw o następującej treści:

```
LOGIN TEKSTY
ASS,SO,CO
COPY,@ SORT,CO
```

W drugim etapie realizacji makroinstrukcji LISTUJ poszczególne komendy ze strumienia JW będą kolejno czytane i wykonywane tak, jakby były wprowadzane przez użytkownika. Efektem będzie otwarcie kartoteki TEKST i skopiowanie pliku o nazwie @ SORT na urządzenie przypisane do strumienia CO. Wykonanie tej samej makroinstrukcji bez parametrów spowoduje wydruk pliku @SORT z kartoteki BIBL na drukarce (obowiązują standardowe wartości parametrów z linii $PRD makrodefinicji).

Już ten bardzo elementarny przykład pozwala się zorientować, że makroinstrukcje dają użytkownikowi możliwość automatyzacji pracy z systemem w zakresie typowych przebiegów przetwarzania często powtarzających się i wymagających podawania tych samych komend z co najwyżej różnymi parametrami. Należy w tym miejscu podkreślić, że makrodefinicja nie

musi zawierać parametrów formalnych. Oczywiście, w takim przypadku każdej jej wywołanie będzie (przy tych samych warunkach początkowych) dawać dokładnie ten sam efekt.

W przypadku systemu SOM 3.P definiując makroinstrukcje należy dodatkowo uwzględnić następujące uwagi:

1. Na sekcji AJC, do której standardowo przyłączony jest strumień JC znajdują się definicje makroinstrukcji systemowych. Dlatego zaleca się, aby użytkownik definicje własnych makroinstrukcji umieszczał na innej sekcji dyskowej. Będzie się mógł do nich odwołać przez zmianę przydziału strumienia JC komendą ASS.
2. Wśród makroinstrukcji systemowych system SOM 3.P jest makroinstrukcja pozwalająca odwoływać się do makrodefinicji umieszczonych w aktualnie otwartej kartotece użytkownika. Jej wywołanie ma postać:

   DO, ⟨mcr⟩ [, ⟨lista parametrów⟩]

   i spowoduje wykonanie makroinstrukcji o nazwie ⟨mcr⟩ zapisanej w pliku o tej samej nazwie dla podanych (opcjonalnie) parametrów aktualnych.

W odniesieniu do makroinstrukcji systemowych systemu SOM 3.P warto również odnotować, że przy ich wywołaniu nazwa makroinstrukcji może być oddzielona od parametrów spacją, przecinkiem lub znakiem /. Parametry oddzielone są przecinkami (żaden inny separator nie oddziela parametrów). Parametry mogą zawierać podparametry rozdzielane spacjami lub znakiem /.

Oprócz opisanej wyżej makroinstrukcji DO do ważniejszych z użytkowego punktu widzenia makroinstrukcji zdefiniowanych w systemie SOM 3.P należy zaliczyć dwie makroinstrukcje o charakterze organizacyjnym: TERMINAL oraz CLEAR. Pierwsza z nich której wywołanie ma postać

TERMINAL/ [⟨t⟩]

zmienia standardowe i aktualne przydziały strumieni CI,CO,LO,1,2 oraz 5 do urządzeń CK ⟨t⟩ (klawiatura) oraz CS ⟨t⟩ (urządzenie listujące)

terminala obsługiwanego przez zadanie JO ⟨t⟩ . Dla przypomnienia parametr ⟨t⟩ w czterozadaniowym systemie SOM 3.P może być jedną z liter: A,B,C lub D. Jeśli parametr ⟨t⟩ jest pominięty przydział odbywa się do zaślepki (strumień NO). Gdy parametr ten ma wartość X przydział ten odbywa się do terminala pełniącego funkcję konsoli operatorskiej.

Makroinstrukcja CLEAR o postaci wywołania

```
CLEAR, ⟨t⟩
```

ustawia wszystkie przydziały strumieni zadania JO ⟨t⟩ jak po instalacji systemu, a następnie kasuje ekran i wypełnia sekcje robocze zadania rekordami znacznika końca zbioru.

O innych makroinstrukcjach skatalogowanych na sekcji dyskowej AJC i gotowych do użycia w systemie SOM 3.P będzie mowa w punkcie 4.3 poświęconym realizacji prac obliczeniowych z wykorzystaniem języka FORTRAN.

4.2. P o d s y s t e m BASIC

Realizację prac obliczeniowych z wykorzystaniem języka programowania BASIC umożliwia procesor o nazwie BAS umieszczony w bibliotece programów systemowych standardowo dołączonej do strumienia LMB.Procesor ten, po wywołany z poziomu interpretera komend poleceniem

```
EXE BAS
```

izoluje użytkownika od standardowego systemu operacyjnego minikomputera MERA 400 definiując odrębny standard pracy przedstawiony w rozdziale szóstym. Z tego powodu pracę pod kontrolą tego programu nader często określa się mianem pracy w systemie Basic. Jest to tym bardziej uzasadnione, że program BAS jako jedyny z procesów systemowych wchodzących w skład oprogramowania podstawowego umożliwia wielostanowiskową pracę (do czterech terminali) w systemie SOM-3. W systemie tym wywołanie procesora BAS powoduje, że na drukarkę zostaje wstępnie wyprowadzony tekst następującej postaci /4/:

```
                    BASIC MERA-400
STANDARD VERSION
STORE = X KWORDS
USER'S STORAGE = XXXX  WORDS
USERS = X
FILES = XXX XXX XXX XXX
INPUT TERMINALS = XXX XXX XXX XXX
OUTPUT TERMINALS = XXX XXX XXX XXX
WORKFILE = XXX
```

Znaki X zastępują tutaj litery i cyfry parametrów, które tworzą początkową, założoną na etapie generacji, konfigurację pracy dla systemu BASIC. Wydruk o podanej wyżej postaci rozpoczyna proces inicjowania pracy, w trakcie którego operator za pomocą odpowiednich dyrektyw może zmienić konfigurację dla danego uruchomienia systemu. Ogólna postać dyrektyw inicjalizacji pracy systemu BASIC jest następująca:

⟨nazwa⟩ = ⟨lista parametrów⟩

przy czym ⟨nazwa⟩ oznacza nazwę dyrektywy (dekodowane, a więc istotne są tylko trzy pierwsze znaki, które dalej będą podkreślane), a lista parametrów oznacza wykaz parametrów oddzielonych od siebie przecinkami zgodnie z definicją funkcji określonej dyrektywy. Dostępne są następujące dyrektywy inicjalizacji pracy systemu BASIC:

STORE = ⟨n⟩ - Ustala ⟨n⟩ K słów pamięci operacyjnej przeznaczonej na cały system BASIC (minimalna wartość parametru wynosi 8, a maksymalna 20)

USERS = ⟨n⟩ - Ustala ilość użytkowników (terminali) w systemie (minimum 1, maksimum 4)

FILES ⟨lista sekcji⟩ - Ustala nazwy sekcji dyskowych użytkowników w ilości odpowiadającej liczbie użytkowników (każdy musi mieć inną sekcję)

INPUT TERMINALS ⟨lista⟩ - Określa nazwy strumieni przyłączonych do urządzeń wejściowych poszczególnych terminali w ilości odpowiadającej liczbie

użytkowników podanej w dyrektywie USERS.

OUTPUT TERMINALS = ⟨lista⟩	- Jak wyżej, ale w odniesieniu do urządzeń wyjściowych terminali (monitor, drukarka).
WORKFILE = ⟨strumień⟩	- Określa dyskowy zbiór roboczy na którym przechowywane są nakładki systemowe oraz obrazy obszarów roboczych użytkowników.
STATE	- Powoduje wydruk stanu konfiguracji.
END	- Kończy proces inicjowania i powoduje rozpoczęcie pracy systemu dla zdefiniowanej konfiguracji.

System BASIC jest udostępniany w dwóch wersjach: dyskowej oraz bezdyskowej. W przypadku tej drugiej wersji komendy FILES oraz WORKFILE są nielegalne. Pominięcie którejś z komend w procesie inicjowania oznacza, że parametry przez nią definiowane przyjmują wartości standardowe. Jedyną dyrektywą, która musi być podana zawsze jest dyrektywa END kończąca proces inicjowania pracy systemu BASIC.

Przykład:

Następujący ciąg dyrektyw podany w procesie inicjowania spowoduje, że system BASIC będzie dla tego uruchomienia obsługiwał dwa terminale, których urządzenia wejściowe (klawiatury) są dołączone do strumieni CI1 oraz CI2, a urządzenia wyjściowe (monitor ekranowy lub drukarka) do strumieni CO1 oraz CO2.

```
USE=2
INP=CI1,CI2
OUT=CO1,CO2
END
```

Pozostałe parametry (w szczególności obszar roboczy systemu) dla uruchomienia jak w powyższym przykładzie przyjmą wartości standardowe.

Po wprowadzeniu dyrektywy END kończącej proces inicjalizacji system

BASIC zgłasza się na urządzeniach wyjściowych terminali użytkowników tekstem:

BASIC MERA-400

i w nowej linii wyprowadza pojedynczy znak gwiazdki, co sygnalizuje jego gotowość do pracy na zasadach omówionych w rozdziale szóstym.

Podczas pracy systemu BASIC operator może ingerować w jego działanie za pośrednictwem komendy operatorskiej zadania komunikacji /LET. Za pośrednictwem tej komendy operator może przekazać systemowi BASIC trzy polecenia o następującej, ogólnej postaci:

/LET @ ⟨l⟩ ⟨n⟩

gdzie:

⟨l⟩ pierwsza litera identyfikująca jedno z trzech dopuszczalnych poleceń:

HOLD - "zawieś" działalność końcówki,

RESTART - "odwieś" działalność końcówki i przejdź na nasłuch,

ABORT - przerwij wykonanie programu użytkownika i przejdź na nasłuch poleceń systemu BASIC,

⟨n⟩ - numer użytkownika (liczba od 1 do 4) według kolejności określonej w dyrektywach inicjalizacji INP oraz OUT.

W przypadku systemu SOM 3.P każdy z użytkowników może za pośrednictwem dyrektywy EXE uruchomić system BASIC w ramach swojego zadania. Dlatego też w tym przypadku w trakcie inicjalizacji systemu BASIC należy określić liczbę użytkowników 1 oraz podać w dyrektywach INP oraz OUT strumienie CI oraz CO do których są przyłączone urządzenia terminala. Z tego powodu w tym systemie typowy przebieg inicjalizacji systemu ma poastać:

```
USE=1
INP=CI
OUT=CO
END
```

Cała komunikacja użytkownika z systemem BASIC odbywa się za pośrednic-

twem urządzeń terminala. System reaguje na kilka znaków sterujących wprowadzanych z klawiatury, ale nie widocznych na wydruku. Nazwy tych znaków i sposób ich generowania (tzn. jakie klawisze trzeba nacisnąć) podano niżej.

CR - (Carriage Return) klawisz powrotu karetki oznaczany symbolem CR lub Return

BSP - (Backspace) klawisze CTRL oraz H jednocześnie naciśnięte

CAN - (Cancel) klawisze CTRL oraz X

DC4 - (Stop) klawisze CTRL oraz T

Znak powrotu karetki kończy wiersz informacji. Po wprowadzeniu przez użytkownika tego znaku wiersz jest przekazywany systemowi BASIC do analizy. Jeżeli wiersz ten jest poprawnym zleceniem systemu BASIC (np. LIST) to jest realizowany, a jeżeli jest instrukcją programu (zaczyna się numerem linii) to system umieszcza go w pamięci; w obydwu przypadkach w następnym kroku system wyprowadzi w następnym wierszu znak gwiazdki sygnalizujący jego gotowość do przyjmowania następnych zleceń.

Znaki BSP oraz CAN mają znaczenie dla linii tekstu, która nie została jeszcze wysłana do systemu klawiszem CR. Znak BSP powoduje wymazanie znaku ostatnio wprowadzonego. Chcąc skasować kilka znaków ostatnio wprowadzonych musimy wprowadzić kilka znaków BSP. Wymazanie całej aktualnie pisanej linii umożliwia nam wysłanie znaku CAN.

Znak DC4 służy do zatrzymania programu w trakcie jego wykonania po komendzie RUN (patrz rozdział 6). W szczególności, jeżeli chcemy zatrzymać wykonanie programu w chwili, gdy oczekuje ona na wprowadzenie danych po wydrukowaniu znaku zapytania, to należy wprowadzić znak DC4, jeszcze jeden dowolny znak (np. spację) i nacisnąć klawisz CR.

4.3. Realizacja programów napisanych w języku FORTRAN

Oprogramowanie podstawowe minikomputera MERA-400 stawia do dyspozycji użytkownika dwa języki wyższego rzędu: BASIC oraz FORTRAN. Liczne

ograniczenia jakim podlega wersja języka BASIC dostępna w omawianym systemie (dotyczy to zwłaszcza wersji dezdyskowej) jak i sama natura tego języka, sprawiają, że w systemie tym jedynie język FORTRAN stwarza użytkownikowi warunki pełnego wykorzystania możliwości systemu z poziomu języków programowania wyższego rzędu. W przeciwieństwie do programów napisanych w języku BASIC realizacja programów napisanych w języku FORTRAN jest procesem wieloetapowym, w których zaangażowana jest znaczna ilość procesorów systemowych wchodzących w skład oprogramowania podstawowego minikomputera MERA-400. Stawia to użytkownikowi wymóg zapoznania się z zasadami użytkowania tychże procesorów, przynajmniej w ich podstawowym zakresie. Zagadnieniu temu poświęcony jest ten punkt, w którym pokrótce zostaną omówione poszczególne procesory biorące udział w procesie realizacji programów napisanych w języku FORTRAN. Wyjątkiem jest edytor tekstowy EDM, któremu z uwagi jego kluczowego znaczenia w procesie przygotowania programów i danych poświęcono znacznie więcej miejsca. Jego obszerny i szczegółowy opis jest treścią punktu 4.4. Sam język FORTRAN w wersji obowiązującej w systemie MERA 400 jest opisany w rozdziale piątym. Poszczególne etapy realizacji programu użytkownika napisanego w języku FORTRAN pokazuje schemat na rysunku 4.2. Stosownie do tego schematu można powiedzieć, że użytkownik, aby doprowadzić swój program do stanu gotowości do wykonania musi:

1. Przygotować wersję źródłową programu. Może tego dokonać poza systemem przygotowując program na papierowej taśmie perforowanej ale znacznie lepszym i w związku z tym zalecanym rozwiązaniem w tym względzie jest posłużenie się jednym z dostępnych edytorów tekstowych. Zalecanym edytorem, ze względu na swoje walory użytkowe jest edytor EDM.
2. Przetłumaczyć za pośrednictwem kompilatora FOR tekst programu zapisany w języku FORTRAN na równoważny tekst napisany w języku MACROASSEMBLER.
3. Przetłumaczyć za pośrednictwem translatora MAC postać assemble-

rową programu do postaci wynikowej, która umożliwia dołączenie do programu procedur bibliotecznych.

4. Dokonać przy użyciu procesora EDI konsolidacji modułów wynikowych uzupełniając je stosownie do potrzeb modułami bibliotecznymi w wyniku czego powstaje program w postaci gotowej do załadowania i wykonania za pośrednictwem dyrektywy EXE interpretera komend.

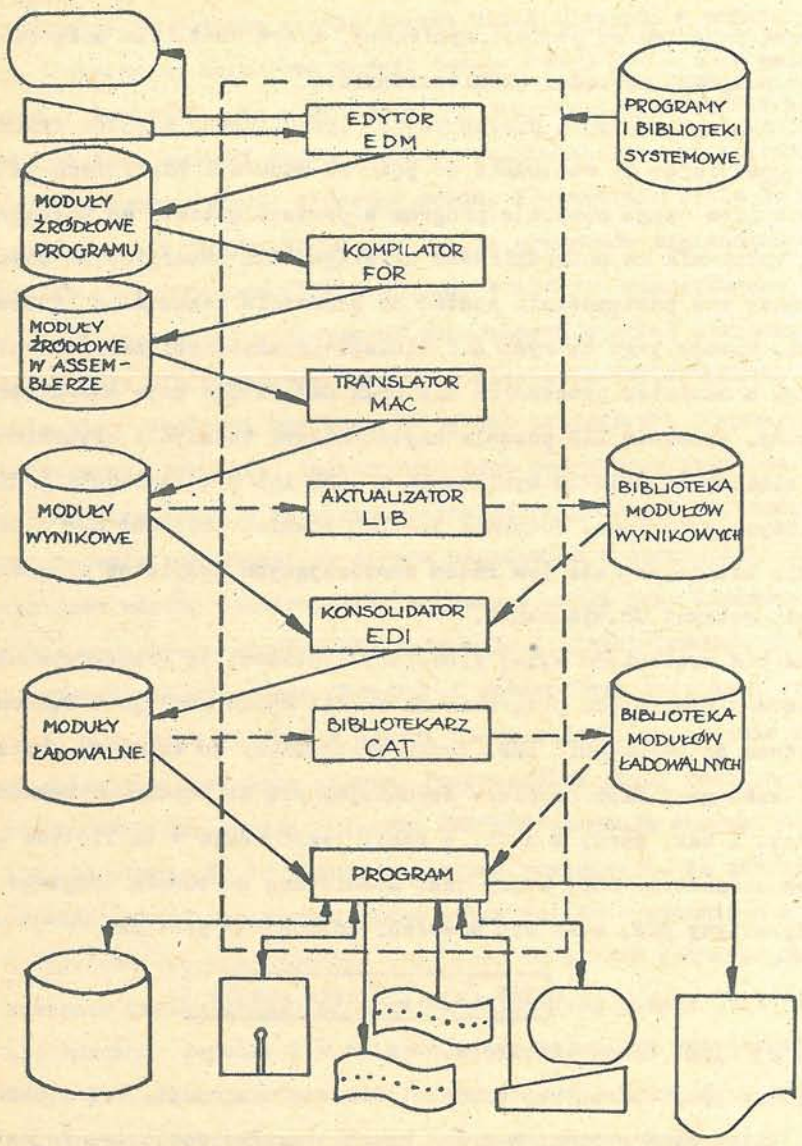
Powyższy tok postępowania został na schemacie zaznaczony liniami ciągłymi. Oprócz tego na rys. 4.2 liniami przerywanymi zaznaczono dwa przebiegi z udziałem procesorów LIB oraz CAT, które mają charakter opcjonalny. Procesor LIB pozwala użytkownikowi tworzyć i aktualizować własne biblioteki modułów wynikowych w celu ich wykorzystania przy realizacji innych programów. Podobnie jak LIB również procesor CAT umożliwia tworzenie bibliotek, ale tym razem zawierających kompletne programy w postaci gotowej do wykonania.

Wszystkie wymienione wyżej procesory systemowe są przechowywane w systemowej bibliotece programów na sekcji dysku stałego dołączonej standardowo do strumienia LMB. Zanim przejdziemy do krótkiej charakterystyki każdego z nich najpierw zapoznajmy się ze wspólnymi zasadami ich pracy. I tak, każdy z nich, z racji rezydowania w bubliotece systemowej na strumieniu LMB, który jest domniemaną wartością drugiego parametru dyrektywy EXE, może być wywołany poleceniem postaci:

EXE ⟨p⟩ ,, ⟨lista opcji⟩

gdzie ⟨p⟩ jest nazwą procesora.

W liście opcji określamy wartości bitów słowa opcji, które określają jeden z założonych wariantów pracy procesora. Przypominamy, że ustawienie określonego bitu słowa opcji wymaga podania jego nazwy (patrz punkt 3.2.3). Poprzedzenie nazwy opcji przedrostkiem NO powoduje, że odpowiadający jej bit będzie ustawiony na zero. W poniższej tablicy zaznaczono symbolami X istotne opcje dla poszczególnych procesorów.



Rys. 4.2. Schemat realizacji programu w języku Fortran (objaśnienia w tekście)

Źródło: opracowanie własne.

Tablica 4.1

Wykaz opcji procesorów systemowych

	UØ	U1	U2	U3	U4	U5	U6	AO	BO	LO	MA	SC	HO
EDM	X						X						
FOR	X	X	X	X	X	X	X		X	X			X
MAC									X	X		X	X
LIB								X		X			X
EDI								X		X	X	X	X
CAT								X		X			X

Źróło: opracowanie własne.

Opcje procesorów EDM oraz FOR są omówione w rozdziałach im poświęconych. Ogólne znaczenie opcji dla pozostałych procesorów można sformułować następująco:

AØ - Ta opcja jest istotna dla procesorów, których pracą użytkownik steruje przy pomocy odpowiednich komend. Standardowo opcja ta ma wartość NOAØ (jest wyzerowana) co powoduje że procesory te akceptują komendy ze strumieni CI standardowo dołączonego do klawiatury terminala użytkownika. Ustawienie tej opcji spowoduje, że procesor będzie akceptował komendy ze strumienia AI. Zalecana wartość - NOAØ.

BO - Istotna dla translatorów FOR oraz MAC i standardowo bit tej opcji jest ustawiony na 1. Wyzerowanie tego bitu powoduje, że dany translator nie generuje przetłumaczonej wersji programu. W związku z tym wywołując wymienione translatory zerujemy bit tej opcji w sytuacji, gdy spodziewamy się, że tłumaczony program nie jest poprawny i chcemy za pośrednictwem translatora otrzymać wykaz błędów składniowych (translacja próbna).

LO - Opcja ta steruje poziomem szczegółowości sprawozdania z przebiegu pracy procesora wyprowadzanego na strumień LO. Wyzerowanie bitu tej opcji powoduje zwykle, że na strumień LO są wyprowadzane żadne informacje za wyjątkiem komunikatów o błędach. Ustawienie tej opcji powoduje wyprowadzenie na LO pełnego

sprawozdania założonego w danym procesorze (np. pełnego listingu translowanego programu).

MA - Opcja ta ma znaczenie wprzypadku konsolidatora EDI i jeśli jest ustawiona, to powoduje wyprowadzenie pełnej mapy skonsolidowanego programu. Ze względu na wyjątkową obszerność tego wydruku i jego małą czytelność dla przeciętnego użytkownika zaleca się ustawianie tej opcji na wartość jeden tylko w przypadkach absolutnej konieczności. Z tych samych powodów w przypadku programu EDI nie zaleca się ustawiania opcji LO.

SC - Ustawienie tej opcji zezwala programom, które z niej korzystają na posługiwanie się strumieniem roboczym SC. Standardowo bit tej opcji jest ustawiony na jeden i zaleca się nie zmieniać jego wartości.

HO - Wartość jeden bitu tej opcji powoduje, że w określonych sytuacjach (różnych dla różnych procesorów) praca procesora jest wstrzymana i może być wznowiona tylko poleceniem operatorskim /R za pośrednictwem zadania komunikacji. Standardowo opcja ta nie jest ustawiona i w typowych pracach nie zachodzi potrzeba zmiany tego stanu rzeczy.

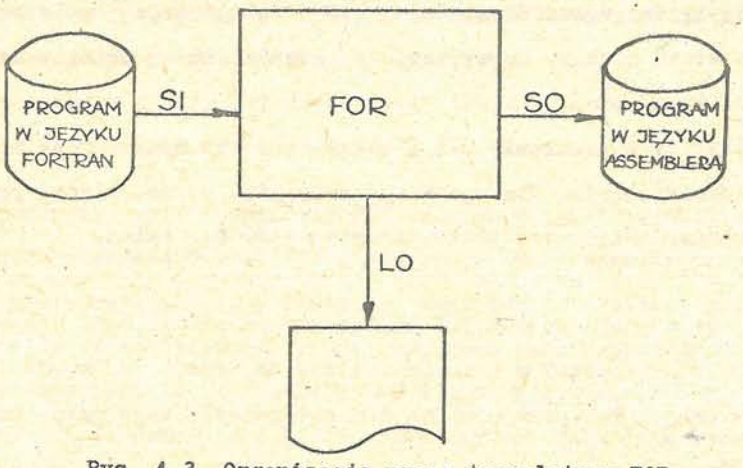
4.3.1. Faza kompilacji

Kompilator FOR realizuje pierwszy etap translacji programu fortranowskiego tłumacząc segmenty źródłowe programu z aktualnie dostępnego zbioru na strumieniu SI na równoważne im segmenty źródłowe w języku symbolicznym minikomputera MERA-400, które są wyprowadzane do zbioru na strumieniu SO. Translator FOR kończy pracę i przekazuje sterowanie interpreterowi komend systemowych z chwilą napotkania na strumieniu SI standardowego znacznika końca zbioru (rekordu składającego się z dwóch znaków dolara). Najczęściej wykorzystywaną opcją w wywołaniu tego translatora jest opcja LO, której ustawienie pozwala uzyskać na urządzeniu dołączonym do strumienia LO listing tłumaczonych segmentów pro-

gramu źródłowego. Schemat ideowy organizacji pracy translatora FOR pokazuje rysunek 4.2, a szczegółowe informacje dotyczące znaczenia pozostałych opcji dla pracy translatora jak i opis diagnostyki błędów wykrytych przez kompilator w tłumaczonym programie zawiera rozdział piąty poświęcony opisowi języka Fortran w wersji dostępnej w systemie MERA-400. Informacji tych nie będziemy tutaj powtarzać a uzupełnimy je jedynie kilkoma uwagami natury systemowej:

1. Należy pamiętać, że kompilator FOR po napotkaniu znacznika końca zbioru na strumieniu SI kończy pracę nie zapisując znacznika końca zbioru na strumieniu SO. Musi to zrobić użytkownik z poziomu interpretera komend dyrektywą WEO SO.

2. Istnieje możliwość kompilacji programu wprowadzanego bezpośrednio z klawiatury dołączając, przed wywołaniem kompilatora, strumień SI do klawiatury komendą ASS SI CI (strumień CI jest standardowo zawsze dołączony do klawiatury danego terminala). Nie jest to zalecany sposób pracy gdyż jakikolwiek błąd we wprowadzanym w ten sposób programie oznacza konieczność powtórzenia całego procesu od nowa. Jeśli jednak z niego korzystamy, to należy pamiętać że koniec wprowadzania programu sygnalizujemy kompilatorowi poprzez wysłanie linii zawierającej dwa znaki dolara na pierwszych dwóch pozycjach znakowych.



Rys. 4.3. Organizacja pracy translatora FOR
Źródło: /4/.

3. W trakcie translacji na urządzenie listujące terminala użytkownika wyprowadzane są komunikaty, które informują go o tym jakie segmenty programu są tłumaczone. Ponadto, bez względu na stan opcji LO wypisywane są informacje o błędach syntaktycznych programu wykrytych przez kompilator. Jeżeli w tłumaczonym programie został wykryty błąd uniemożliwiający poprawne wygenerowanie przetłumaczonej wersji programu (tzw. fatalny błąd kompilacji) to kompilator oprócz stosownego komunikatu ustawia wskaźnik błędu w słowie komunikacji FLAG i kończąc pracę po wczytaniu znacznika końca zbioru ze strumienia SI dodatkowo ostrzega użytkownika, że BYLY BLEDY. Brak takiego komunikatu oznacza, że proces translacji zakończył się prawidłowo. Należy pamiętać, że w przypadku kompilacji z błędami, która powoduje ustawienie wskaźnika FLAG i przy niezapalonej opcji GO zadania interpreter komend systemowych nie będzie akceptował następnych dyrektyw EXE podawanych z terminala. W takiej sytuacji należy przywrócić stan pełnej operatywności zadania użytkownika przez zainicjowanie interpretera komend poleceniem Q (SOM 3.P) lub $JOB (SOM-3).

4. Strumień SO, na który kompilator wyprowadza przetłumaczony program jest standardowo przypisany do sekcji roboczej dysku stałego i w zasadzie trudno wskazać sytuację, która uzasadniałaby potrzebę zmiany tego stanu rzeczy. Korzystanie ze standardowych zbiorów roboczych minimalizuje zarówno ilość niezbędnych informacji wprowadzanych przez użytkownika z terminala jak i związane z tym ryzyko popełnienia przez użytkownika błędu. Dlatego w tym przypadku można zalecać pozostawienie standardowego przydziału strumienia SO bez zmian.

Przykład:

Załóżmy, że w systemie SOM 3.P użytkownik ma skompilować program w języku FORTRAN zapisany w trzecim zbiorze na sekcji AM1 wymiennego pakietu dyskowego. Tok postępowania dla osiągnięcia tego celu będzie następujący:

Q - Inicjowanie pracy interpretera komend. Wszystkie strumienie otrzymają standardowe przydziały.

ASS SI,AM1,LO,CSL - Przypisanie strumienia SI do sekcji dyskowej na której jest program źródłowy oraz strumienia LO do drukarki. Strumień SO zachowuje standardowy przydział do sekcji ASB (patrz punkt 3.2.2.4).

AVF SI,2 - Przewinięcie strumienia SI na początek trzeciego zbioru sekcji AM1.

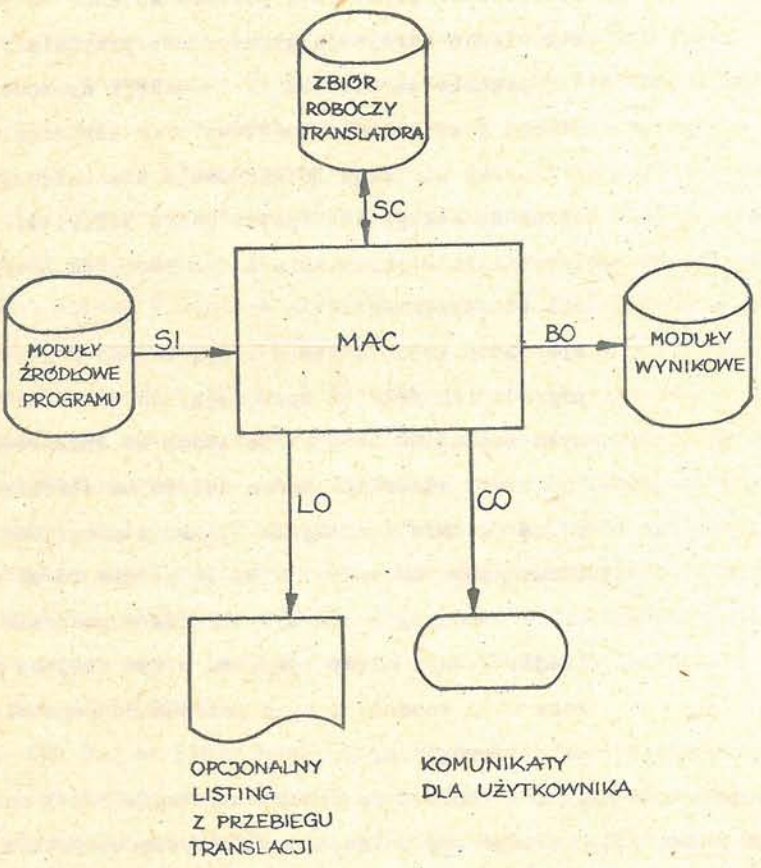
EXE FOR,,LO - Wywołanie translatora z opcją LO, co w konsekwencji poprzednich poleceń spowoduje, że treść tłumaczonych segmentów będzie listowana na drukarce.

WEO SO - Wypisanie znacznika końca zbioru na strumień SO co ma na celu zamknięcie zbioru z programem przetłumaczonym na sekcji ASB. Polecenie to ma sens tylko wtedy, gdy nie było błędów w procesie translacji. Jeżeli błędy były, to w tym miejscu należałoby użyć komendy Q oraz przejść do poprawiania tekstu programu.

Chcąc powyższy przykład odnieść do pracy w systemie SOM-3 należy w miejsce komendy Q posłużyć się poleceniem $JOB oraz wszystkie pozostałe polecenia poprzedzić znakiem dolara.

4.3.2. Faza kodowania

Wygenerowany przez kompilator FOR program w języku symbolicznym minikomputera MERA-400 musi być w następnym kroku przetłumaczony do postaci wynikowej (półskomplikowanej) akceptowanej później przez program łączący (konsolidator). Proces ten jest realizowany przez dostępny w systemie translator o nazwie MAC w sposób pokazany na rysunku 4.4. Ogólna idea pracy tego translatora jest podobna do sposobu pracy kompilatora FOR. Segmenty źródłowe są czytane ze zbioru aktualnie dos-



Rys. 4.4. Organizacja pracy translatora MAC
Źródło: /4/.

tępnego na strumieniu SI i po przetłumaczeniu do postaci wynikowej zapisywane do zbioru na strumieniu BO, który standardowo w obydwu systemach SOM jest przypisany do sekcji dysku stałego ASA. Translator MAC może być wykorzystywany niezależnie od kompilatora FOR przez tych użytkowników, którzy formułują swoje programy bezpośrednio w języku MACROASSEMBLER. W takim przypadku wymagana jest bardziej szczegółowa znajomość sposobu działania procesora MAC, która jest w zasadzie zbędny dla użytkowników piszących swe programy w języku FORTRAN. Ponieważ

programowanie w języku symbolicznym wykracza poza ramy tego opracowania więc poprzestaniemy na omówieniu schematu jego wykorzystania w procesie translacji programów napisanych w języku FORTRAN. Generalnie, faza kodowania programu wygenerowanego przez kompilator FOR traktowana jako niezależne zadanie wymaga użycia następującej sekwencji dyrektyw*:

```
Q
ASS SI= ⟨si⟩, MC= ⟨mc⟩ ,BO= ⟨bo⟩ ,LO= ⟨lo⟩
EXE MAC, ⟨lm⟩ , ⟨opcje⟩
WEO BO
```

gdzie:

⟨si⟩ - strumień z aktualnie dostępnym zbiorem stanowiącym produkt pracy kompilatora FOR,

⟨mc⟩ - strumień zawierający makrodefinicje makroinstrukcji z których korzystał kompilator FOR,

⟨bo⟩ - strumień, na który MAC wyprowadza po translacji moduły wynikowe programu,

⟨lo⟩ - strumień wyjścia listującego,

⟨lm⟩ - strumień, z którego ładowany jest procesor MAC (standardowo LMB).

W praktyce zaleca się użytkownikom programującym w Fortranie maksymalne korzystanie z założonych w systemach SOM definicji z powodów analogicznych do wymienionych przy omawianiu procesora FOR. W szczególności dotyczy to strumieni ⟨lm⟩ oraz ⟨mc⟩. Użytkownik musi w pierwszym rzędzie zadbać o prawidłowe określenie wejścia i wyjścia dla translatora MAC. Dodatkowo zaleca się używanie opcji NOLO a to z tej racji, że tekst programu w języku MACROASSEMBLER jest mało czytelny dla użytkownika, który sformułował go w języku FORTRAN a jego wydruk zabiera zwykle dużo czasu.

---

* W dyrektywie ASSIGN w charakterze separatorów oddzielających nazwę strumienia od obiektu przydziału użyto w celu większej czytelności znaku "=" a nie przecinka. Jest to niezgodne z ogólną definicją podaną dla tej komendy, ale zgodnie z uwagami dotyczącymi ograniczników, które podano w punkcie 3.3.

Przykład:

Kontynuując proces translacji programu z przykładu w poprzednim punkcie dalszy ciąg postępowania może mieć następujący, typowy przebieg:

ASS SI SO - Nie inicjujemy interpretera komend, dzięki czemu najłatwiej można określić, gdzie znajduje się produkt wynikowy pracy translatora FOR, który będzie stanowił wejście dla procesora MAC.

REW SI - Ustawiamy wskaźnik położenia strumienia na początek zbioru zawierającego program do tłumaczenia.

EXE MAC,,NOLO - Procesor MAC jest wywoływany z biblioteki LMB (pusty parametr na pozycji strumienia ⟨1m⟩). Opcja NOLO oznacza, że rezygnujemy z wydruku programu w języku MACROASSEMBLER.

WEO BO - Podobnie jak w przypadku translatora FOR tak i tutaj należy zamknąć zbiór z programem wynikowym wypisując na strumień wyjściowy BO znacznik końca zbioru.

Podobnie jak FOR również translator MAC informuje użytkownika o nazwach translowanych segmentów poprzez stosowne komunikaty. Zakładając, że procesor MAC tłumaczy program będący produktem wyjściowym kompilatora FOR jest rzeczą mało prawdopodobną aby w trakcie fazy kodowania pojawiły się jakiekolwiek komunikaty o błędach. Jeżeli jednak użytkownik otrzyma komunikat o błędzie, to najprawdopodobniej zaistniała jedna z dwóch, następujących możliwości:

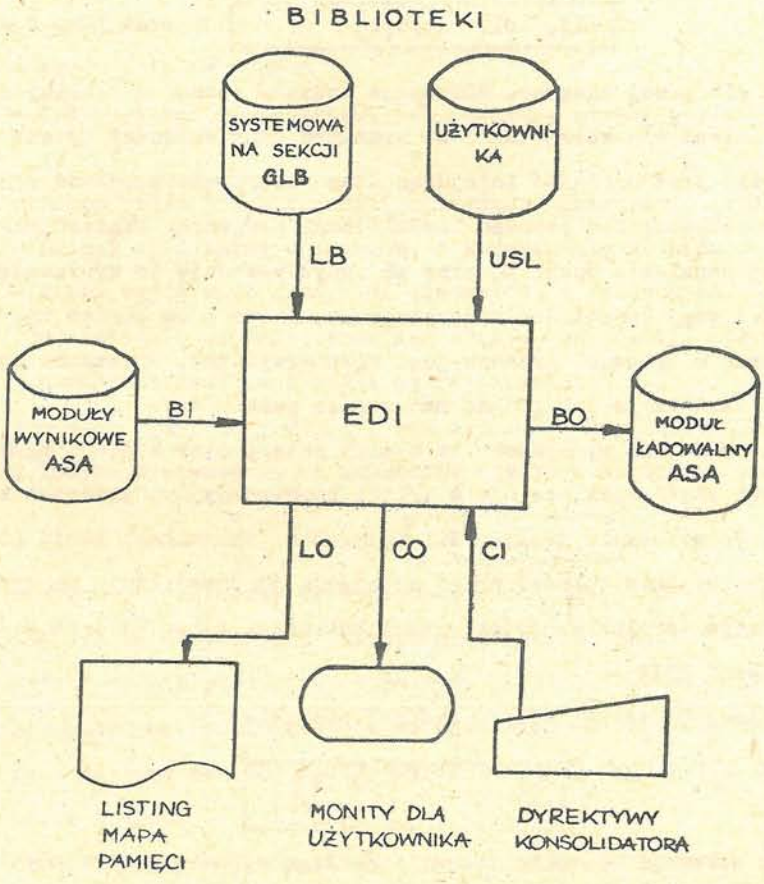
1. Nastąpiło przekłamanie w trakcie transmisji dyskowej.
2. Nieprawidłowo został zdefiniowany zbiór wejściowy do translacji (tłumaczony jest niewłaściwy program).

W obydwu przypadkach najbezpieczniej jest powtórzyć cały proces translacji programu od początku. Istnieje jeszcze trzecia, ale nader mało prawdopodobna ewentualność, że jeden z tłumaczonych segmentów jest tak duży, że powoduje przepełnienie tablic translatora. W takim przypadku ponowna translacja da identyczny rezultat i wyjściem z tej

sytuacji może być jedynie zmiana struktury programu źródłowego (np. rozbicie dużego segmentu na dwa mniejsze).

4.3.3. K o n s o l i d a c j a p r o g r a m u

Program fortranowski napisany przez użytkownika w zasadzie nigdy nie jest kompletny, a to z tego powodu, że translator tłumacząc program źródłowy wiele operacji dostępnych w języku Fortran tłumaczy na odwołania do procedur dostępnych w systemowych bibliotekach podprogramów. Również użytkownik może odwoływać się do wcześniej opracowanych przez siebie podprogramów, które zgromadził we własnych bibliotekach.



Rys. 4.5. Organizacja pracy konsolidatora EDI
Źródło: /4/.

Podprogramy w bibliotekach są przechowywane w postaci modułów wynikowych będących produktem pracy procesora MAC omówionego w poprzednim punkcie. Korzystanie z uprzednio opracowanych i przetestowanych procedur bibliotecznych znacznie ułatwia proces opracowywania programu i czyni go bardziej efektywnym pod każdym względem. Proces łączenia modułów zdefiniowanych w programie źródłowym z modułami, do których w programie źródłowym są tylko odwołania nosi nazwę konsolidacji procesorem systemowym dostępnym w bibliotece na strumieniu LMB za pośrednictwem dyrektywy EXE postaci:

```
EXE,,EDI, ⟨opcje⟩
```

Istotne dla pracy programu EDI opcje zostały podane w tablicy 4.1 z tym, że zaleca się korzystanie ze standardowych wartości tychże opcji, co najłatwiej jest osiągnąć inicjując stan interpretera komend systemowych przed rozpoczęciem procesu konsolidacji programu. Szczególnie nie zaleca się zapalania opcji LO oraz MA, gdyż powoduje to wyprowadzenie szczegółowej mapy konsolidowanego programu, która poza bardzo sporadycznymi okazjami w typowych pracach jest mało przydatna. Znaczenie pozostałych opcji jest takie jak podano na wstępie punktu 4.3.

Procedur EDI jest wyposażony we własny interpreter komend, przy pomocy których użytkownik steruje w trybie konwersacyjnym procesem konsolidacji. Po wywołaniu program EDI zgłasza użytkownikowi swoją gotowość wość do przyjmowania poleceń przez przejście do nowej linii na urządzeniu listującym terminala. Podstawowymi komendami konsolidatora są dyrektywy LINK oraz EDIT.

Przy pomocy dyrektywy LINK o ogólnej postaci:

```
LINK ⟨s⟩ [{MAIN}]
         [{nazwa}]
```

użytkownik wskazuje konsolidatorowi z jakiego strumienia (parametr ⟨s⟩) jakie segmenty (parametr ⟨nazwa⟩) mają wejść w skład modułu ładowalnego. Ładowanie wskazanych dyrektywami LINK modułów wynikowych jest

funkcją pierwszego przebiegu pracy konsolidatora. Drugi przebieg, który polega na uzupełnieniu programu o podprogramy biblioteczne i wyprowadzenie skonsolidowanego programu do zbioru dołączonego do strumienia BO jest efektem polecenia postaci

```
EDIT
```

Jeżeli w wywołaniu konsolidatora EDI było podana (lub wcześniej ustawiona opcja mapy MA to dodatkowym efektem będzie wyprowadzenie w drugim przebiegu szczegółowej mapy modułu ładowalnego na strumień listujący LO. Mapa pamięci zawiera następujące informacje o konsolidowanym programie:

- Adresy bloków COMMON
- Nazwy elementów programu, ich poziomy, adresy początkowe i długości
- Nazwy wejść i ich adresy (poprzedzone znakiem $, jeżeli są relokowalne) oraz nazwy elementów, z których się do nich odwołano,
- Nazwy wyjść wraz z nazwami elementów, w których znajdowało się odpowiednie wejście, oraz ich adresy (wartości), które są poprzedzone znakiem $ jeśli są relokowalne.

Procesor EDI kończy swoją pracę i przekazuje sterowanie interpreterowi komend systemowych na polecenie użytkownika mające postać:

```
EXIT
```

Powyższy, podstawowy opis procesu konsolidacji przy pomocy programu EDI należy uzupełnić o następujące, istotne uwagi:

1. Jeżeli drugim parametrem dyrektywy LINK jest słowo MAIN lub drugi parametr jest pominięty, to ze wskazanego strumienia będą wczytywane i łączone ze sobą moduły wynikowe aż do napotkania znacznika końca zbioru.
2. Zbiorem na strumieniu określonym pierwszym parametrem dyrektywy LINK może być biblioteka podprogramów. W takim przypadku drugi para-

metr (powinien wystąpić, gdyż w przeciwnym razie zostałyby dołączone wszystkie moduły znajdujące się w bibliotece) precyzuje, który z podprogramów zawartych w bibliotece ma zostać dołączony do konsolidowanego programu.

3. Przy przeszukiwaniu bibliotek w drugiej fazie konsolidacji (po dyrektywie EDIT) obowiązuje zasada, że najpierw jest przeszukiwany zbiór na strumieniu USL (biblioteka użytkownika), a dopiero potem biblioteka systemowa ze strumienia LB. Wynika stąd, że jeżeli w obu bibliotekach będą moduły o tych samych nazwach, to zostanie dołączony moduł zawarty w bibliotece użytkownika.

Wszelkie informacje o błędach konsolidator EDI wyprowadza na strumień CO dołączony do urządzenia listującego terminala w postaci tekstu opisującego występującą nieprawidłowość. Przykładowo:

! ZŁA SKŁADNIA DYREKTYWY
! BŁAD STRUKTURY MODUŁU WYNIKOWEGO
itp.

W zależności od opcji HO oraz AØ, po wystąpieniu błędu i wydruku stosownego komunikatu o nim konsolidator podejmie następujące działania:

AØ	HO	
0	0	- przejście do czytania następnych dyrektyw ze strumienia CI
0	1	- przejście do systemu i czekanie na decyzję operatora
1	1	
1	0	- wyjście z programu jak po dyrektywie EXIT. Konsolidowany program może nie być poprawny.

Przykład:

Zakładając, że zarówno w fazie kompilacji jak i kodowania programu korzystaliśmy ze standardowych przydziałów strumieni, to moduły wynikowe programu są dostępne dla procesu konsolidacji w pierwszym zbiorze na sekcji roboczej dysku stałego o nazwie ASB, która stanowi standardowy przydział strumieni BI oraz BO. W takiej sytuacji proces konsolidacji programu najprościej jest przeprowadzić w następujący sposób:

Q - Zainicjowanie pracy interpretera komend systemowych w celu nadania opcjom oraz przydziałom strumieni wartości standardowych,

EXE EDI,,NOMA,NOLO - wywołanie konsolidatora z zerowaniem (na wszelki wypadek) opcji LO oraz MA

LINK BI
EDIT
EXIT
} - dyrektywy konsolidatora omówione wyżej

W efekcie użytkownik (zakładając brak błędów) uzyska moduł ładowalny programu zapisany w pierwszym zbiorze sekcji dyskowej ASA (standardowy przydział strumienia BO, którego nie zmieniliśmy dyrektywą ASS), który jest gotowy do wywołania i wykonania z poziomu interpretera komend systemowych za pośrednictwem dyrektywy EXE.

4.3.4. Wykonanie programów napisanych w języku FORTRAN

Po wykonaniu czynności opisanych w punktach od 4.3.1 do 4.3.3 użytkownik otrzymuje binarny moduł ładowania programu zapisany w zbiorze określonym w procesie konsolidacji przez odpowiedni przydział strumienia BO. W celu wykonania skonsolidowanego programu znajdującego się w zbiorze na strumieniu będącym drugim parametrem dyrektywy EXE należy wprowadzić następujące polecenie:

EXE ⟨nazwa⟩ , ⟨s⟩ , ⟨opcje⟩

przy czym parametr ⟨nazwa⟩ określa nazwę segmentu głównego ładowanego programu lub, w przypadku gdy aktualnie dostępny zbiór na strumieniu ⟨s⟩ jest biblioteką, trzyznakowy identyfikator programu, pod którym jest on przechowywany w bibliotece. Opcje fazy egzekucji mają następujące:

U0=U1=U2=0 - w przypadku błędu wykonania podaj numer błędu na strumieniu CO i zakończ wykonanie programu

U0=1, U1=U2=0 - w przypadku błędu zakończ działanie programu

U0=U2=ϕ, U1=1 - Kontynuuj program po błędzie

U0=ϕ, J1=U2=1 - wypisz numer błędu na strumień CO i zawieś wykonanie programu; zawieszony program może wznowić operator polecenie /R zadania komunikacji.

U3=1 - listuj "ślad" (trace) programu

Błędy wykonania są sygnalizowane wydrukiem na strumień CO komunikatu postaci:

⟨ttt⟩ (⟨pro⟩) ERROR ⟨nr. błędu⟩ [⟨nr. instrukcji⟩]

gdzie: ⟨ttt⟩jest nazwą zadania, ⟨pro⟩- trzema pierwszymi znakami nazwy programu, a numer instrukcji jest wyprowadzany tylko wtedy jeżeli w fazie kompilacji programu procesorem FOR była zapalona opcja U1.

Znaczenie numerów błędów wykonania jest podane w rozdziale 5.

4.3.5. Makroinstrukcje translacji programów fortranowskich w systemie SOM 3.P

W zakresie typowych prac proces translacji programów fortranowskich w znakomity sposób ułatwiają w systemie SOM 3.P (system SOM-3 tych udogodnień nie posiada - ewentualnie użytkownik może zdefiniować własne makrodefinicje) umieszczone na sekcji AJC makrodefinicje typowych przebiegów translacji. Odwołanie się do tych makroinstrukcji odbywa się z poziomu interpretera komend systemowych na zasadach omówionych w punkcie 4.1.7, przy czym zunifikowana postać odwołania się do makroinstrukcji programów fortranowskich da się przedstawić w następujący sposób:

⟨z⟩ FOR[/ [⟨opcje⟩][, ⟨definicje przydziałów strumieni⟩]]

gdzie ⟨z⟩ jest jedną z liter: Q, T, C lub L i precyzuje, o którą makroinstrukcję chodzi, co wiąże się z zakresem translacji programu. Poszczególne makroinstrukcje umożliwiają:

QFOR - Próbna kompilacja translatorem FOR bez generowania programu (np. w celu sprawdzenia, czy program jest wolny od błędów formalnych).

TFOR - Realizuje tylko fazę kompilacji programem FOR jak w punkcie 4.3.1.

CFOR - Wykonywane są fazy kompilacji i kodowania. Wynik translacji ma postać relokowalnych modułów wynikowych programu w zbiorze na strumieniu BO.

LFOR - Pełna translacja programu do postaci modułu ładowalnego programu zapisanego w zbiorze na strumieniu BO.

Pierwszym parametrem odwołania się do makroinstrukcji translacji jest lista opcji fazy kompilacji. Pozostałe fazy translacji (makroinstrukcje CFOR oraz LFOR) są z definicji realizowane z opcjami NOLO, NOMA, a stan pozostałych opcji nie ulega zmianie po fazie kompilacji.

Drugim parametrem odwołania się do makroinstrukcji jest lista przydziałów strumieni mających istotne znaczenie dla procesu translacji i omówionych w punktach od 4.3.1 do 4.3.3, która jest skonstruowana na tych samych zasadach i ma takie samo znaczenie jak lista przydziałów strumieni dyrektywy ASS.

Brak definicji przydziału dla określonego strumienia biorącego udział w procesie translacji powoduje, że w trakcie realizacji makroinstrukcji przydziałem aktualnym dla takiego strumienia jest jego przydział standardowy.

Przykłady wykorzystania makroinstrukcji:

1. Próbną translację programu przygotowanego na taśmce papierowej z wyprowadzeniem listingu na drukarkę uzyskamy przy pomocy makroinstrukcji QFOR wywołanej w następujący sposób:

```
QFOR/LO MA,SI PR LO CSL
```

2. Przetłumaczenie programu źródłowego zapisanego w zbiorze na sekcji AM1 do postaci binarnego modułu ładowalnego, który będzie umieszczony w zbiorze na sekcji AM2 można uzyskać jednym poleceniem postaci:

```
LFOR/NOLO,SI AM1 BO AM2
```

Z przebiegu translacji nie będzie żadnego raportu na strumieniu LO

(obowiązuje opcja NOLO). Wykonanie skompilowanego programu uzyskamy w wyniku podania następującej sekwencji dyrektyw:

```
REW BO
EXE nazwa,BO, opcje
```

gdzie pierwszy parametr określa nazwę segmentu głównego fortranowskiego programu.

4.3.6. Tworzenie i aktualizacja bibliotek podprogramów użytkowych

Sprawdzone i przetestowane podprogramy wielokrotnego użytku (jako przykład może służyć podprogram odwracania macierzy) wygodnie jest przechowywać w zbiorach dyskowych w postaci relokowalnych modułów wynikowych z uwagi na łatwość dołączania ich do różnych programów na etapie konsolidacji. Konsolidator EDI może traktować każdy zbiór będący produktem pracy translatora MAC jako bibliotekę modułów wynikowych użytkownika. Z tego powodu utworzenie biblioteki podprogramów w systemie MERA-400 nie nastręcza większych trudności.

Przykładowo chcąc pod systemem SOM-3 utworzyć bibliotekę podprogramów przygotowanych na taśmce papierowej należy zrealizować następujące zadanie:

```
$JOB
$ASS SI,PR,
$EXE FOR
$WEO SO
$ASS SI,SO,BO,AM1
$REW SI
$EXE MAC,,NOLO
$WEO BO
```

W świetle dotychczasowych przykładów interpretacja przebiegu powyższej sesji pracy nie powinna Czytelnikowi nastręczać żadnych trudności.

W systemie SOM 3.P ten sam efekt można znacznie łatwiej uzyskać z uwagi na dostępność makroinstrukcji translacji CFOR. W tym systemie

przebieg zakładania zbioru podprogramów na sekcji użytkownika AM1 na analogicznych jak wyżej warunkach może mieć następujący przebieg:

```
Q
CFOR/NOLO,SI PR BO AM1
```

Problem powstaje z chwilą zaistnienia potrzeby aktualizacji zbioru biblioteki. Do tego celu w systemie MERA-400 służy procesor o nazwie LIB uruchamiany poleceniem

```
EXE LIB,, ⟨opcje⟩
```

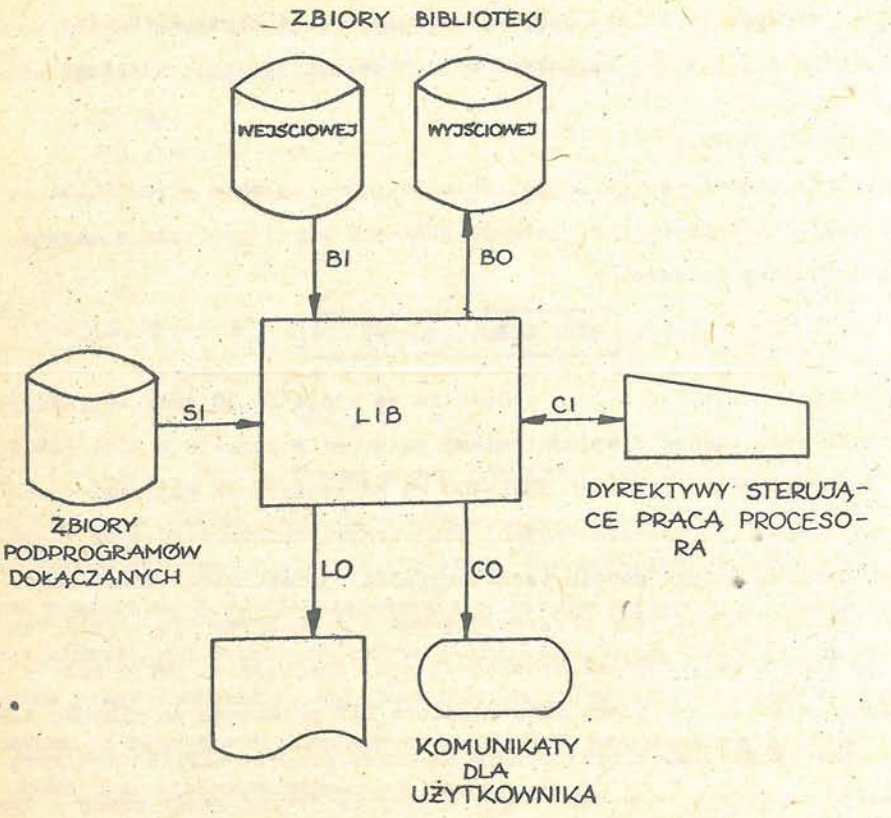
Istotnymi opcjami dla tego procesora są opcje HO,LO oraz AØ, których znaczenie jest zgodne z wyjaśnieniami podanymi w punkcie 4.3.1. Biblioteki podprogramów w systemie MERA-400 są sekwencyjnymi zbiorami modułów wynikowych (półskompilowanych) zbudowanymi ze standardowych rekordów binarnych. Z tego powodu (sekwencyjność) jakakolwiek zmiana zawartości takiego zbioru jest zawsze związana z przepisywaniem, czyli tworzeniem jego nowej, zaktualizowanej wersji. Znajduje to swoje odzwierciedlenie w schemacie pracy aktualizatora LIB pokazanym na rysunku 4.6. Podprogramy w zbiorze bibliotecznym są przez program LIB identyfikowane według swojego numeru kolejnego. W trakcie jednej sesji pracy z tym procesorem istnieje możliwość aktualizacji w zasadzie nieograniczonej liczby zbiorów.

Aktualizacji podlegają zbiory biblioteczne dostępne na strumieniu BI. Poprawione wersje zbiorów wyprowadzane są na strumień BO (biblioteki wyjściowe). Do zaktualizowanej wersji biblioteki mogą być dołączane zbiory modułów wynikowych dostępne na strumieniu SI. Użytkownik steruje pracą aktualizatora LIB za pośrednictwem komend wprowadzanych ze strumienia CI (standardowo klawiatura terminala).

Aktualizację polegającą na dopisaniu zbioru modułów wynikowych do biblioteki umożliwia komenda postaci

```
ADD ⟨n⟩
```

która powoduje, że do aktualnie dostępnego zbioru biblioteki wejścio-



Rys. 4.6. Przepływ informacji w trakcie pracy procesora LIB
Źródło: /4/.

wej zostanie przepisane ⟨n⟩ -1 pierwszych modułów, po których w bibliotece wyjściowej zostanie umieszczona zawartość aktualnie dostępnego zbioru na strumieniu SI. Znacznik końca zbioru ze strumienia SI nie zostanie przekopiowany.

Operacja odwrotna do ADD, która polega na usunięciu z nowej wersji biblioteki określonej ilości podporgramów jest efektem wykonania dyrektywy

DELETE ⟨n⟩ [, ⟨m⟩]

W tym przypadku, podobnie jak przy ADD ze starej biblioteki do nowej

zostanie przekopiowane ⟨n⟩ - 1 początkowych modułów, a następnie z biblioteki wejściowej zostaną wczytane i zignorowane moduły od numeru ⟨n⟩ do ⟨m⟩ włącznie.

Dyrektywa

REPLACE ⟨n⟩ [, ⟨m⟩]

działa tak jak ADD i DELETE łącznie. Powoduje przepisanie początkowych ⟨n⟩ - 1 podprogramów ze zbioru wejściowego do wyjściowego, po czym ignoruje kolejne moduły aż do modułu o numerze ⟨m⟩ a do biblioteki wyjściowej dopisuje zawartość aktualnie dostępnego zbioru ze strumienia SI (tak samo jak ADD).

Przepisanie pojedynczego modułu o numerze porządkowym ⟨n⟩ ze starej biblioteki do nowej umożliwia dyrektywa

GET ⟨n⟩

Określenie przydziału strumieni oraz ich pozycjonowanie umożliwiają komendy ASS, AVF oraz WEO o identycznej definicji i działaniu jak analogiczne komendy interpretera systemu operacyjnego omówione w punkcie 4.1.3.

W trakcie aktualizacji przy aktywnej opcji LO aktualizator wyprowadza na strumień LO informacje o zbiorach biblioteki wyjściowej o różnym stopniu szczegółowości w zależności od tego, która z niżej podanych komend była wprowadzona jako ostatnia:

LALL	LNS	LNA

Najbardziej szczegółowy listing zawierający nazwy modułów, nazwy bloków COMMON i ich wielkości, nazwy punktów wejścia i wyjścia, długości oraz adresy startowe uzyskujemy w trakcie obowiązywania dyrektywy LAL (jest to również opcjonalny poziom). Po dyrektywie LNS listing jest okrojony o informacje dotyczące punktów wejścia i wyjścia. Podanie dyrektywy LNA spowoduje, że od tego momentu będą wyprowadzane tylko nazwy modułów zawartych w bibliotece wyjściowej.

Wydruk informacji o zawartości zbiorów bibliotecznych na strumieniu BI możemy uzyskać przy pomocy dyrektywy

LIST ⟨q⟩

gdzie ⟨q⟩ określa ilość zbiorów. Komendę tę można podać w dowolnym momencie aktualizacji. Spowoduje ona przeczytanie do końca aktualnie przetwarzanego zbioru ze strumienia BI, a następnie wczytanie ⟨q⟩ -1 zbiorów i wydruk informacji o ich zawartości.

Aktualizator LIB kończy pracę na polecenie użytkownika mające postać dyrektywy

EXIT

Diagnostyka aktualizatora obejmuje 7 komunikatów o błędach, których postać i znaczenie jest następujące:

1. STATEMENT ERROR - nierozpoznana lub nieprawidłowa dyrektywa
2. CHECKSUM ERROR - błędna suma kontrolna rekordu binarnego
3. ILLEGAL BINARY RECORD - wczytany rekord nie jest standardowym rekordem binarnym ani znacznikiem końca zbioru
4. SEQUENCE ERROR - nieprawidłowy numer kolejny rekordu
5. FUNCTION ERROR - błąd w rekordzie logicznym
6. END - przedwczesne napotkanie końca zbioru w trakcie wykonywania dyrektyw ADD, DEL, REP lub GET
7. END MISSING - brak logicznego rekordu "END OF PROGRAM" przed rekordem końca zbioru lub błąd końca programu.

4.3.7. Biblioteki programów skonsolidowanych

Procesor systemowy przechowywany w bibliotece dostępnej na strumieniu LMB pod identyfikatorem CAT umożliwia użytkownikowi systemu MERA-400 tworzyć biblioteki zawierające skonsolidowany, tzn. gotowe do wykonania

programy. Biblioteki programów skonsolidowanych w standardzie bibliotekarza są tzw. zbiorami słownikowymi. W słowniku zbioru bibliotecznego przechowywane są identyfikatory programów aktualnie dostępnych w bibliotece i inne niezbędne do łatwego wyszukania określonego programu informacje. Procesor CAT jest ładowany z biblioteki LMB dyrektywą EXE o standardowej postaci. Po załadowaniu przechodzi na nasłuch dyrektyw sterujących jego pracą, które standardowo są akceptowane ze strumienia CI.

Utworzenie biblioteki na określonej sekcji dyskowej użytkownika wymaga najpierw utworzenia pustego zbioru słownikowego. Umożliwia to komenda postaci

INITIALIZE ⟨s⟩

Strumień ⟨s⟩ musi być uprzednio odpowiednio przydzielony za pośrednictwem dyrektywy ASS o postaci i działaniu identycznym z analogiczną komendą interpretera komend systemowych.

Dopisywanie modułów ładowalnych (skonsolidowanych programów fotranowskich) do biblioteki na zbiorze słownikowym jest funkcją komendy

CATALOG ⟨s⟩ , ⟨nazwa⟩ , ⟨ident⟩

Moduł, który chcemy wprowadzić do biblioteki powinien być zawarty w aktualnie dostępnym zbiorze na strumieniu BI. Dyrektywa CAT spowoduje wczytanie ze strumienia programu, którego segment główny ma nazwę określoną drugim parametrem (⟨nazwa⟩) i zapisanie go do biblioteki w zbiorze słownikowym dostępnym na strumieniu ⟨s⟩. Program ten będzie dostępny w bibliotece pod identyfikatorem składającym się z trzech znaków kodu CAN i podanym jako parametr ⟨ident⟩ dyrektywy CAT. Wykonanie programu znajdującego się w bibliotece jest możliwe za pośrednictwem dyrektywy EXE po podaniu identyfikatora i nazwy strumienia z dostępnym zbiorem bibliotecznym,w którym ten program się znajduje.

Jeżeli z określonych powodów użytkownik chce, aby ten sam program w bibliotece był dostępny pod różnymi identyfikatorami, to może to

osiągnąć za pośrednictwem dyrektywy

DUPLICATE ⟨s⟩ , ⟨ident-1⟩ , ⟨ident-2⟩

gdzie ⟨s⟩ jest nazwą strumienia z biblioteką a pozostałe parametry trzyznakowymi identyfikatorami: pierwszy określa identyfikator pod jakim program został umieszczony, a drugi parametr określa dodatkowy identyfikator pod którym, od momentu pomyślnego zakończenia realizacji dyrektywy DUP dany program będzie również dostępny.

Logiczne usunięcie programu polegające na usunięciu informacji o nim ze słownika zbioru (przez co program staje się automatycznie niedostępny) jest efektem wykonania dyrektywy

DELETE ⟨s⟩ , ⟨ident⟩

przy czym ⟨s⟩ jest nazwą strumienia ze zbiorem bibliotecznym, a drugi parametr określa identyfikator programu, który ma być usunięty. Zbiór dyskowy, z którego słownika zostały usunięte nazwy programów zawiera w dalszym ciągu ciała tych programów. Usunięcie takich zbędnych programów i przepakowanie pozostałych programów w bibliotece tak, aby zajmowały jak najmniej miejsca jest funkcją dyrektywy o postaci:

COMPRESS ⟨s⟩, ⟨s⟩

której parametrami są nazwy strumieni: pierwsza nazwa określa strumień ze zbiorem biblioteki podlegającej opisanemu wyżej zabiegowi kompresji, a zbiór, którego nazwa została podana jako drugi parametr będzie używany jako zbiór roboczy w tym procesie.

Użytkownik może za pośrednictwem dyrektywy LIST o postaci

LIST ⟨s⟩

uzyskać na strumieniu LO wydruk zawartości słownika biblioteki ze zbioru na strumieniu określonym parametrem.

Procesor CAT zakończy pracę i przekaże sterowanie interpreterowi komend systemowych po wprowadzeniu komendy

EXIT

Komunikaty o błędach emitowane przez program CAT mają postać tekstów w języku polskim objaśniających zaistniałą, błędną sytuację.

4.4. Edytor tekstowy EDM

Producent systemu MERA-400 dostarcza w ramach oprogramowania podstawowego edytor tekstowy o nazwie UPD. Podstawową wadą tego edytora jest sekwencyjna organizacja aktualizacji tekstu. Jeśli dodać do tego, że edytor UPD nie oferuje żadnych mechanizmów kontekstowego wyszukiwania i poprawiania informacji, to uzasadnionym staje się twierdzenie o jego niskich walorach eksploatacyjnych.

Diametralnie różną opinię można wystawić edytorowi o nazwie EDM, który został opracowany w Instytucie Informatyki Uniwersytetu Warszawskiego w oparciu o sprawdzone wzory edytorów ED oraz EX działających pod systemem operacyjnym UNIX na maszynie PDP-11.

4.4.1. Ogólna charakterystyka pracy z edytorem

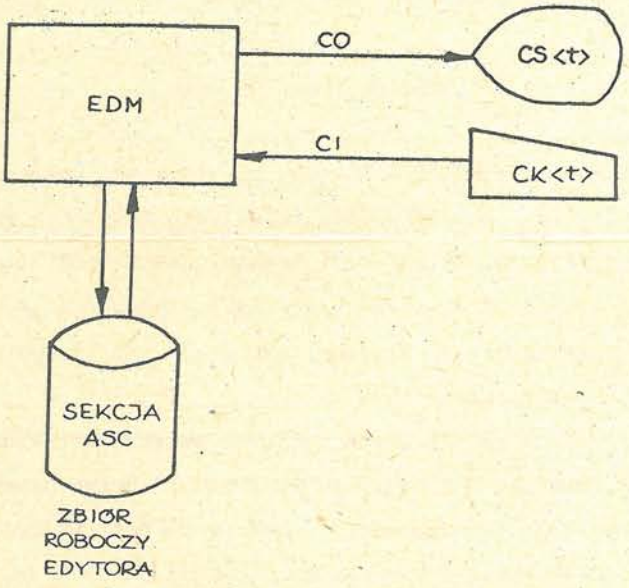
Program EDM jest ładowany do pamięci operacyjnej i uruchamiany tak jak wszystkie programy pod systemem SOM-3 poleceniem EXECUTE. W przypadku systemu operacyjnego SOM-3 polecenie uruchomienia edytora będzie miało następującą, ogólną postać:

$EXE EDM,, ⟨ewentualne opcje⟩

W przypadku pracy pod kontrolą systemu operacyjnego SOM 3.P należy pamiętać, że instrukcje języka zleceń tego systemu nie są poprzedzane znakiem dolara, oraz, że programy z biblioteki systemowej na strumieniu LMB mogą być wywoływane poprzez nazwę. Tak więc w przypadku systemu SOM 3.P wywołanie programu EDM można zlecić systemowi na trzy, alter-

natywne sposoby:

EXE EDM,, ewentualne opcje
E EDM,, ewentualne ppcje
EDM,, ewentualne opcje



Rys. 4.7. Schemat przepływu informacji w trakcie pracy edytora EDM
Źródło: opracowanie własne.

Zwykle wywołanie programu EDM nie będzie zawierać opcji. W takim przypadku należy również opuścić przecinki oddzielające listę opcji od nazwy programu. Dla określenia sposobu pracy programu istotne znaczenie mają jedynie dwie opcje: opcja AI oraz opcja UØ. Autorzy programu EDM nie zalecają używania opcji AI, która powoduje, że dyrektywy sterujące pracą edytora są wczytywane ze strumienia AI, a nie jak to standardowo przyjęte ze strumienia CI, który normalnie oznacza klawiaturę terminala użytkownika.

Istotne znaczenie ma natomiast opcja UØ. Rzecz w tym, że EDM dopuszcza realizację pewnych dyrektyw systemu operacyjnego takich jak ASS, REW, AVP oraz BSF. Wykonanie takiej dyrektywy z błędnymi parametrami (np. przypisanie dyrektywą ASS strumienia do nieistniejącego urządze-

nia) może spowodować "abort" (odrzucenie) programu przez system, co dla użytkownika mogło by oznaczać przykre konsekwencje w postaci utraty potencjalnie wielu wprowadzonych do momentu "awarii" poprawek. Z tej racji, że system operacyjny SOM nie pozwala sprawdzać legalności parametrów swoich dyrektyw EDM nie może mieć wewnętrznych zabezpieczeń przed taką sytuacją. Opcja UØ w wywołaniu programu EDM komendą EXE o podanej postaci stanowi zabezpieczenie użytkownika przed skutkami opisanej sytuacji jak i przed skutkami innych błędów powodujących niespodziewane i awaryjne przerwanie pracy edytora. Sposób postępowania zmierzający do odtworzenia stanu pracy edytora jaki miał miejsce w momencie przerwania jego pracy wywołanego błędem można opisać następująco:

1° Przyciskiem OPRQ na pulpicie technicznym uaktywnić zadanie komunikacji

2° Komendą [/R] wznowić zadanie użytkownika z konsoli operatorskiej

3° Z terminala użytkownika zainicjować pracę interpretera komend komendą [$JOB] w przypadku systemu SOM-3 lub komendą [Q] w przypadku systemu SOM 3.P

4° Wywołać ponownie program EDM komendą [EXE EDM,,UØ] pamiętając o konieczności poprzedzenia słowa EXE znakiem dolara jeżeli pracujemy pod kontrolą systemu SOM-3.

Należy w tym miejscu nadmienić, że opcja UØ jest nieaktywna, jeżeli poprzednia praca z edytorem została zakończona normalnym trybem, to jest podaniem komendy edytora kończącej jego pracę przezwwymazanie programu EDM z pamięci operacyjnej i przekazanie sterowania interpreterowi komend systemu operacyjnego.

Po zainicjowaniu pracy programu EDM w opisany wyżej sposób zgłasza on swoją gotowość na terminalu z którego został wywołany przez przejście kursora do nowej linii (wysuw papieru o jedną linię na terminalu DZM-180-KSR) i przechodzi na nasłuch dyrektyw. Wychodząc z definicji, że program EDM jest przeznaczony do interakcyjnego kreowania oraz modyfikowania tekstów i opierając się na przedstawionym na rysunku 4.7

schemacie przepływu informacji pojedynczy krok pracy z omawianym edytorem można przedstawić w sposób następujący:

- użytkownik z klawiatury terminala przyłączonej do strumienia CI wprowadza dyrektywę edytora zlecającą wykonanie określonej operacji jak i wszystkie niezbędne dane do jej prawidłowego wykonania.
- edytor realizuje dyrektywę po uprzednim sprawdzeniu jej poprawności kierując wszelkie informacje z przebiegu realizacji na strumień CO przyłączony standardowo do urządzenia listującego (monitor ekranowy lub drukarka) terminala użytkownika, po czym przechodzi w stan oczekiwania na następne zlecenie sygnalizując to w analogiczny sposób jak w momencie rozpoczęcia pracy.

Ten tok postępowania obowiązuje do momentu podania dyrektywy QUIT , która powoduje zakończenie pracy programu EDM.

U podstaw filozofii działania edytora EDM leży założenie, że wszystkie zlecane odpowiednimi dyrektywami opracje na tekście odnoszą się do informacji pamiętanej w zbiorze roboczym edytora dołączonym do strumienia SC. Standardowo, zgodnie z rysunkiem strumień SC jest dołączony do sekcji roboczej dysku stałego o nazwie ASC. Z punktu widzenia użytkownika informacja w zbiorze roboczym edytora ma strukturę adresowalnych linii tekstu. Możliwość wygodnego adresowania linii poprawianego tekstu wpływa w najistotniejszy sposób na elegancję i efektywność pracy z edytorem EDM. Użytkownik zostaje tym samym zwolniony z konieczności sekwencyjnego wprowadzania poprawek w oparciu o bezwzględne numery linii tekstu źródłowego co, jak już wspomnieliśmy, jest najpoważniejszą wadą firmowego edytora o nazwie UPD. Możliwość bezpośredniego dostępu do dowolnej linii tekstu w zbiorze roboczym implikuje oczywisty w tym kontekście wymóg, że zbiór ten musi być zlokalizowany w szybkiej pamięci o dostępie bezpośrednim.

Z chwilą wywołania edytora zbiór roboczy otrzymuje status "pusty" to znaczy taki, który nie zawiera ani jednej linii tekstu. Odstępstwo od tej sytuacji ma miejsce tylko w przypadku, gdy EDM jest wywołany

z opcją UØ. Wówczas edytor usiłuje odtworzyć zawartość zbioru roboczego, co jest możliwe tylko w okolicznościach omówionych wyżej przy objaśnianiu znaczenia użycia opcji UØ w wywołaniu edytora.

Zawartość zbioru roboczego użytkownik definiuje przy użyciu odpowiednich komend objaśnionych szczegółowo dalej na dwa sposoby:

- przez wprowadzanie linii tekstu bezpośrednio z klawiatury terminala, co jest typowym sposobem w sytuacji kreowania tekstu,
- przez przepisanie tekstu z zewnętrznego nośnika (dysk elastyczny, sekcja dyskowa dysku sztywnego, taśiemka papierowa) do zbioru roboczego.

Drugi sposób może być wykorzystany zarówno w trakcie kreowania tekstu jak i modyfikacji istniejącego zbioru tekstowego. W tym drugim przypadku przepisanie poprawianego tekstu do zbioru roboczego edytora będzie pierwszą czynnością po zainicjowaniu pracy programu EDM. W trakcie kreowania dyrektywy wczytywania informacji do zbioru roboczego pozwalają nam wykorzystywać fragmenty innych tekstów do utworzenia nowego tekstu. W obydwu przypadkach istnieje potrzeba przeglądania tekstu w zbiorze roboczym i dokonywania odpowiednich poprawek polegających na usuwaniu zbędnych linii, wprowadzaniu nowych, wymianie błędnych fragmentów określonych linii, czy też na zamianie miejscami określonych partii tekstu. Do wymienionych celów EDM oferuje bogaty zestaw dyrektyw, które z racji swojego przeznaczenia można określić mianem dyrektyw do przeglądania i poprawiania tekstu w zbiorze roboczym.

Ponieważ z chwilą zakończenia pracy programu EDM zawartość zbioru roboczego staje się niedostępna, więc użytkownik przed zakończeniem pracy edytora ma możliwość przepisania opracowanego tekstu do określonego zbioru, przyłączonego do jednego ze strumieni zadania użytkownika. Wynika stąd potrzeba definiowania przydziału urządzeń fizycznych do strumieni jak konieczność pozycjonowania nośnika informacji urządzenia dołączonego do strumienia, z którego EDM wczytuje informacje do zbioru roboczego, bądź dokonuje odwrotnego transferu informacji. Autorzy pro-

gramu EDM rozwiązali to w ten sposób, że użytkownik z poziomu edytora ma dostęp do tych komend systemu operacyjnego SOM, które umożliwiają realizację wymienionych operacji.

Domykając ogólną charakterystykę omawianego edytora należy wspomnieć o tym, że ma on wbudowanych szereg rozwiązań umożliwiających dopasowanie sposobu użytkowania i, w pewnym zakresie, możliwości programu EDM do indywidualnych upodobań i potrzeb użytkownika. Do najważniejszych z nich można zaliczyć:

- Możliwość stosowania skrótów w słowach kluczowych. Na ogół zamiast słowa kluczowego dyrektywy można podawać tylko jedną literę.
- Szeroko stosowana zasada wartości domniemanych istotnych parametrów komend. Uwalnia to użytkownika od konieczności podawania dyrektyw w pełnym brzmieniu w tych sytuacjach, gdy wartości jej parametrów są tożsame z założonymi w programie EDM.
- Możliwość korzystania z silnych dyrektyw organizacyjnych, które bardziej wprawnemu użytkownikowi pozwalają w jednym poleceniu zamknąć zlecenie wykonania wielu operacji na tekście.
- Możliwość zmiany pewnych parametrów rzutujących na sposób pracy z programem EDM. Odpowiednie dyrektywy umożliwiają zdefiniowanie tzw. formatu wprowadzanych linii tekstu, zmianę sposobu sygnalizacji błędów wykrywanych przez EDM oraz redefinicję założonego rozmiaru i struktury fizycznej zbioru roboczego.

Pewne utrudnienie w opanowaniu zasad posługiwania się edytorem stanowi istnienie tzw. drugiego wariantu wykonania niektórych dyrektyw, który jest sygnalizowany znakiem wykrzyknika umieszczanym bezpośrednio po słowie kluczowym komendy oraz brak specjalizowanej komendy pozwalającej otrzymać wydruk tekstu na drukarce, co ma istotne znaczenie dla użytkownika korzystającego z terminala typu monitor ekranowy z klawiaturą. Ta druga niedogodność jest możliwa do obejścia poprzez użycie dyrektywy przepisania zawartości zbioru roboczego na strumień uprzednio dołączony do drukarki (dyrektywa WRITE), ale w kontekście ogólnej

elegancji pracy z programem EDM takie rozwiązanie problemu otrzymania drukowanej kopii tekstu jest odczuwane jako swoisty dysonans.

4.4.2. Sposoby adresowania linii tekstu

Jak już nadmieniališmy, jedną z zasadniczych zalet edytora EDM jest możliwość wygodnego adresowania linii tekstu w zbiorze roboczym, na których mają być wykonane operacje zlecane dyrektywami. Ogólnie rzecz biorąc w przypadku omawianego edytora można mówić o dwóch metodach adresowania linii, a co za tym idzie, o dwojakiego rodzaju adresach.

Pierwsza z nich, pojęciowo znacznie prostsza, polega na tym, że adres określa numer bezwzględny linii tekstu w zbiorze roboczym. Adres taki można nazwać adresem bezwzględnym, albo bezkontekstowym i jest on po prostu albo liczbą naturalną, albo prostym wyrażeniem arytmetycznym, którego wartość określa numer linii tekstu. Rzecz w tym, że linie tekstu w zbiorze roboczym są ponumerowane kolejnymi liczbami naturalnymi od 1 do n. Jedna z linii, a mianowicie ta, której dotyczyła ostatnia operacja, ma status linii ostatnio aktualizowanej. Znajomość adresu tej linii jest niezwykle ważna z punktu widzenia organizacji pracy programu EDM, gdyż z jednej strony adres ten wyznacza aktualne położenie punktu pracy w obrębie poprawianego tekstu, a z drugiej strony stanowi on założoną wartość adresu wielu dyrektyw edytora, w sytuacjach, gdy użytkownik przy podawaniu dyrektywy pominie jej część adresową. Z tego drugiego powodu równie ważny jest również adres pierwszej linii tekstu jak i numer linii ostatniej, Z pierwszą linią nie ma kłopotów, gdyż jej adres bezwzględny ma zawsze wartość jeden. Inaczej przedstawia się sprawa z adresem linii ostatnio aktualizowanej i adresem ostatniej linii tekstu, gdyż dyrektywy, które zmieniają ilość linii tekstu (np. dyrektywy kasowania lub wprowadzania nowych linii) powodują zmianę numeracji linii leżących za linią ostatnio aktualizo-

waną. Aby ułatwić użytkownikowi odwoływanie się do tych linii, przyjęto że ich adresy są wartościami dwu znaków specjalnych, jeśli znaki te pojawią się w części adresowej dyrektywy. I tak:

$ (znak dolara) - oznacza adres ostatniej linii tekstu w zbiorze roboczym,

. (znak kropki) - reprezentuje adres linii ostatnio aktualizowanej.

Z racji przyjętego znaczenia symbolu kropki jako adres linii przy omawianiu poszczególnych dyrektyw, dla których adres ten ma istotne znaczenie, zamiast rozwlekłego określenia "adres ostatnio aktualizowanej linii" będziemy posługiwać się niezbyt poprawnie brzmiącym ale znacznie bardziej zwartym terminem "adres kropki".

Przykłady:

7	oznacza siódmą linię tekstu
7+3-4+2	oznacza ósmą linię tekstu
$-5	oznacza piątą linię od końca
.-3+5	oznacza drugą linię za linią ostatnio aktualizowaną
$+1	jest adresem błędnym, gdyż odnosi się do nie istniejącej linii tekstu (formalne znaczenie: następna linia za ostatnią linią tekstu).

W przypadku drugiej metody adresacji, którą należy zaliczyć w poczet kluczowych możliwości edytora, mamy do czynienia z tzw. adresem kontekstowym. Podstawową częścią takiego adresu jest tak zwany wzorzec, który reprezentuje pewien tekst. Wzorzec adresuje linię zawierającą wystąpienie reprezentowanego nim tekstu. Najprostszy adres kontekstowy jest tożsamy z wzorcem i może mieć postać

/ wzorzec /

lub

? wzorzec ?

Różnica pomiędzy obydwoma postaciami adresu kontekstowego zamyka się w sposobie, w jaki EDM wyszukuje linię zawierającą podany w adre-

sie wzorzec. W przypadku użycia jako ograniczników znaku / przebieg wyszukiwania jest następujący:

Przeglądane są linie o numerach .+1, .+2, ... itd. i wybrana zostaje pierwsza linia zawierająca wystąpienie określonego wzorcem tekstu. Jeśli zanim to nastąpi zostanie osiągnięty koniec zbioru, to poszukiwania są kontynuowane od początku zbioru, to znaczy przeglądane są linie o numerach 1, 2, ... itd., aż do ponownego osiągnięcia linii, od której rozpoczęło się przeszukiwanie (jest to zawsze linia o numerze wskazywanym adresem kropki). Jeżeli wskazany wzorcem tekst nie zostanie odnaleziony w żadnej linii aktualnie przetwarzanego tekstu, wówczas EDM sygnalizuje błąd.

W przypadku, gdy jako ogranicznika wzorca w adresie kontekstowym użyjemy znaku ? sposób postępowania jest analogiczny do wyżej opisanego, z tą różnicą, że przeszukiwanie odbywa się w odwrotnym kierunku, tzn. przeglądane są linie .-1, .-2, ... do linii o numerze 1, a dalej, w analogiczny sposób, od ostatniej linii, aż do linii o adresie kropki.

Nieco bardziej rozbudowana forma adresu kontekstowego może oprócz wzorca zawierać proste wyrażenie arytmetyczne, na przykład:

/READ/+10 - oznacza dziesiątą linię za linią, w której po raz pierwszy wystąpił tekst READ

?AS?-1+4 - odnosi się do trzeciej linii po linii zawierającej tekst AS

Z doświadczeń autorów w użytkowaniu programu EDM wynika, że omówione do tej pory postacie adresów i wzorca tekstu są najbardziej przydatne i najczęściej używane. Dalsze, bogate możliwości edytora EDM w tym zakresie umożliwiają szybkie wprowadzanie nieraz bardzo wyrafinowanych poprawek, ale ich opanowanie może w pierwszym momencie nastręczać pewne trudności. Dlatego też można sugerować, aby Czytelnik, który nie miał do tej pory okazji korzystać z usług edytorów tekstowych, pominął przy pierwszym czytaniu dalszy ciąg niniejszego punktu i powrócił do niego z chwilą praktycznego opanowania zasad do tej pory omówionych.

Wzorzec w dotychczas omówionej postaci stanowił bezpośrednio wyszukiwany fragment tekstu. W bardziej ogólnym przypadku wzorzec może odnosić się nie do jednego konkretnego napisu, a do pewnej klasy tekstów, które spełniają podane we wzorcu warunki. W przypadku takiego uogólnionego wzorca, który w edytorze EDM ma postać przypominającą wyrażenie regularne, można zadanie wyszukiwania określonej linii formułować w sposób bardziej ogólny, np. znaleźć linię, która zawiera napis składający się z ciągu cyfr. W takim przypadku nie chodzi o określony ciąg cyfr, ale jakikolwiek ciąg cyfr. Jeżeli polecenie takie dotyczyłoby tekstu programu w Fortranie, to mogło by ono przykładowo wynikać z potrzeby szybkiego dotarcia do pierwszej etykietowanej instrukcji programu.

Przystępując do omawiania sposobu definiowania takiego uogólnionego wzorca należy zacząć od podstawowej informacji, że pewne znaki mogące występować we wzorcu nie reprezentują bezpośrednio znaków, które chcemy wyszukiwać w tekście. Są to tzw. znaki specjalne. Dwa z nich zostały już omówione, a są to znaki pytajnika "?" oraz kreski ukośnej "/", które pełnią funkcję ograniczników wzorca. Do pozostałych znaków, które w definicji wzorca mogą pełnić rolę znaków specjalnych należą*:

*	gwiazdka
.	kropka
[]	nawiasy kwadratowe
$	dolar
^	circumflex (na monitorach ekranowych strzałka w górę)
\	backslash
-	minus

Znak "^" (circumflex) ma specjalne znaczenie tylko wtedy, gdy występuje jako pierwszy znak wzorca. Analogicznie znak dolara "$" jest znakiem specjalnym wtedy i tylko wtedy, gdy bezpośrednio po nim

* W przypadku urządzeń z klawiaturą uwzględniającą istnienie specyficznych znaków języka polskiego takich jak ś. ź. itd., znaki specjalne mają inne oznaczenie. Patrz tablica 7 w punkcie 2.8.

występuje prawy ogranicznik wzorca. Warunkiem, aby prawy nawias "]" kwadratowy był interpretowany jako znak specjalny wzorca jest wcześniejsze wystąpienie lewego nawiasu kwadratowego " [". Pomiędzy parą nawiasów kwadratowych " [...] " gwiazdka i kropka przestają być znakami specjalnymi, a zaczyna nim być znak minus. Te nieco zawiłe związki zaczną być bardziej zrozumiałe, gdy zilustrujemy je odpowiednim zestawem przykładów. Zanim do tego przejdziemy, jeszcze jedna uwaga.

Jeżeli gdziekolwiek pragniemy użyć któregoś z wymienionych znaków specjalnych jako znaku o zwykłym charakterze, wówczas należy go poprzedzić znakiem " \ " (backslash). To samo dotyczy również samego znaku " \ ", który z racji opisanej funkcji jest przy pojedynczym wystąpieniu uważany za znak specjalny.

Pojawienie się znaku gwiazdki "*" we wzorcu reprezentuje wielokrotne powtórzenie znaku, po którym we wzorcu wystąpiła gwiazdka. Przykładowo wzorzec:

JACEK KOF*MAN

reprezentuje tekst zawierający sekwencję znaków "JACEK KO", po której może wielkorotnie pojawić się litera "F" (w szczególności litera "F" może ani raz nie wystąpić), a całość uzupełnia sekwencja "MAN". W ten sposób można powiedzieć, że wzorzec ten pasuje do wszystkich niżej podanych napisów:

JACEK KOMAN
JACEK KOFMAN
JACEK KOFFMAN
JACEK KOFFFFMAN
... itd.

Kropka we wzorcu oznacza, że na jej miejscu może pojawić się dowolny znak. Innymi słowy kropka pasuje do każdego znaku. I tak wzorzec:

B.A

zostanie w trakcie przeszukiwania dopasowany do każdego z poniższych tekstów:

```
B + A
B / A
BYK
BAK ... itd.
```

Znak "^ " ma cpecjalne znaczenie jedynie na początku wzorca. Jego użycie sygnalizuje, że określony po nim wzorzec musi zostać dopasowany do tekstu na początku linii. Na przykład wzorzec:

```
^ALFA
```

może odnosić się do linii zaczynających się tekstem

```
ALFABET
ALFA=X**5 ... itd.
```

ale nie zostanie dopasowany na przykład do linii

```
X=ALFA
```

mimo że linia ta zawiera wystąpienie tekstu "ALFA".

Podobnie znak dolara umieszczony na ostatniej pozycji wzorca reprezentuje wirtualny koniec linii i z tego powodu wzorzec:

```
A)$
```

będzie adresował linie takie jak:

```
X=ALFA/(2-A)
CALL BETA(A)
```

ale nie będzie odnosił się przykładowo do tekstu postaci

```
GAMMA(A) = 1.5
```

Z wymienionych funkcji znaków "^ " oraz "$" jako znaków specjalnych wynika, że wzorzec postaci

```
^$
```

pasuje jedynie do pustej linii tekstu, a wzorzec

```
^C$
```

może adresować tylko takie linie, których cała treść składa się z jednego wystąpienia znaku "C".

Rozwijając ostatni przykład zauważmy, że dzięki własnościom omówionych do tej pory znaków specjalnych mamy możliwość adresowania linii zawierających określoną liczbę znaków, gdyż przykładowo wzorzec:

```
^..... $
```

oznaczać będzie polecenie wyszukania linii, która zawiera dokładnie pięć (obojętne jakich) znaków.

Należy w tym miejscu poinformować użytkownika, że wszystkie odstępy (znaki spacji) są ignorowane, jeżeli występują na końcu linii poprawianego tekstu. W szczególności pusta linia jest rzeczywiście pustą linią i nie zawiera ani jednego znaku. Ponieważ jednak systemy operacyjne SOM ignorują takie linie, więc EDM w momencie przepisywania tekstu ze zbioru roboczego do zbioru zewnętrznego zamienia puste linie na linie zawierające dokładnie jeden znak spacji.

Nawiasy kwadratowe jako znaki specjalne służą do definiowania klasy znaków odpowiadających danej pozycji wzorca. Wypisując sekwencję znaków ograniczoną parą nawiasów kwadratowych "[ ]" wskazujemy, że bieżąca pozycja wzorca ma pasować do dowolnego z wymienionych znaków. Należy w tym miejscu podkreślić, że taka definicja odnosi się do jednej pozycji znakowej wzorca, chyba że bezpośrednio po niej pojawi się znak gwiazdki, o czym będzie mowa dalej. W celu ułatwienia definicji większych liczebnie klas znaków, przy pomocy omawianej konstrukcji wprowadzono dodatkowo dwie reguły:

1. Jeżeli w obrębie nawiasów kwadratowych pojawi się para znaków rozdzielonych znakiem "-" (minus), to jest to równoznaczne z wypisaniem sekwencji wszystkich znaków, których kody w standardzie ISO-7 mają wartość nie mniejszą niż kod znaku poprzedzającego znak minus, i nie większą niż kod znaku wymienionego po znaku minus. Przykładowo zapis [A-D] jest tożsamy z zapisem [ABCD], a definicja [0-9] reprezentuje wszystkie znaki cyfr.
2. Umieszczenie bezpośrednio po lewym nawiasie kwadratowym "[" znaku "^" (circumflex) oznacza, że do danej pozycji znakowej wzorca pasują wszystkie znaki, które nie są reprezentowane przez dalszy ciąg definicji. W ten sposób definicja postaci [^A-Z0-9] definicje klasę znaków nie będących znakami liter ani cyfr.

Przykłady:

Wzorzec:

$$A[+*\backslash -/]B$$

będzie dopasowany do każdego z poniższych napisów

A - B
A + B
A * B
A / B

ale próba dopasowania go do jakiejkolwiek innej sekwencji trzech znaków da rezultat negatywny. Zwróćmy uwagę, że ponieważ normalnie znak minus w obrębie definicji klasy znaków jest znakiem specjalnym to w powyższym przykładzie zaszła konieczność poprzedzenia go znakiem "\" aby nadać mu status zwykłego znaku.

Rozpatrzmy teraz wzorzec postaci:

$$[A-ZO-9]\ ..+..\ [ABCDE]$$

Reprezentuje on tekst składający się z siedmiu znaków, z których:

- pierwszy znak musi być literą lub cyfrą,
- dwa następne znaki mogą być dowolne,
- czwarty znak musi być znakiem "+",
- na piątej i szóstej pozycji mogą znowu wystąpić dowolne znaki,
- ostatni znak musi być jedną z pięciu początkowych liter alfabetu.

Może być on zatem dopasowany na przykład do następujących sekwencji znaków:

B7) + (XA
527 + 2*D
0.3 + 1-C

W analogiczny sposób, stosując wszystkie podane reguły definiowania klasy znaków, można stwierdzić, że wzorzec:

$$[\hat{}0-9AZ]\ XXX$$

reprezentując tekst złożony z trzech liter X poprzedzonych znakiem różnym od cyfry, litery A oraz litery Z.

Po definicji klasy znaków zamkniętej nawiasami kwadratowymi jak i po

znaku kropki, która poza obrębem nawiasów kwadratowych jest równoważna definicji klasy wszystkich możliwych znaków, może we wzorcu pojawić się znak gwiazdki interpretowany w omówiony już sposób. Daje to duże możliwości reprezentowania przez wzprzec rozmaitych klas tekstów, niekoniecznie tej samej długości.

Dla przykładu rozważmy następujący wzorzec:

[A-Z][A-Z0-9] *[+\ - * /][A-Z] [A-Z0-9]*

Zostanie on między innymi dopasowany do każdego z poniższych fragmentów tekstu:

A/B

ALFA-OMEGA

ABCDEFGHIJK*T0123456789XYZ98

ale nie będzie go można dopasować do napisów takich jak:

12*A3

1ALEKSANDER-2WITOLDY

Na tych samych zasadach wzorzec:

[A-F].*)= [^A]*

reprezentuje takie napisy jak

A-B+PI(6)=B

C(X(ALA(12+C+V))-AS)=123.5678

ale nie będzie przykładowo pasował do takiego tekstu jak:

A(I)=A1

Pewne dodatkowe możliwości definiowania wzorca są szczególnie przydatne w operacjach wymiany fragmentów tekstu linii i dlatego zostaną omówione przy okazji objaśniania dyrektywy SUBSTITUTE.

Najważniejszymi ograniczeniami ilościowymi jakie obowiązują przy definiowaniu wzorca są:

- długość wzorca nie może przekraczać 128 znaków,
- liczba gwiazdek użytych we wzorcu w charakterze znaków specjalnych nie może przekroczyć sześciu.

Na zakończenie niniejszego punktu warto jeszcze dodać i zapamiętać,

że użycie pustego wzorca, to jest wzorca postaci

//

lub

??

EDM zawsze traktuje jako odwołanie się do poprzednio użytego wzorca. Stanowi to znaczne ułatwienie w przypadku zajścia potrzeby wielokrotnego wyszukiwania tych samych fragmentów tekstu.

4.4.3. Postać dyrektyw edytora

Ogólny format dyrektyw akceptowanych przez edytor EDM da się opisać następującą formułą:

[⟨adres1⟩[{,;}⟨adres2⟩]] [!]⟨słowo kluczowe⟩[!] [⟨parametry⟩] [⟨opcje⟩]

Znaczenie symboliki użytej w powyższej definicji, a która stanowić będzie również minimalny zestaw środków opisu ogólnej postaci poszczególnych dyrektyw, jest następujące:

< > - ostre nawiasy sygnalizują określenia metajęzykowe (zmienne syntaktyczne),

- nawiasy kwadratowe oznaczają opcjonalne elementy postaci dyrektywy,
- użycie w opisie składni dyrektywy nawiasów klamrowych oznacza, że konkretna postać dyrektywy musi zawierać jeden i tylko jeden z elementów wyspecyfikowanych wewnątrz nawiasów.

Jeżeli dyrektywa ma część adresową, to pojawiające się tam adresy mają postać omówioną w poprzednim punkcie. W przypadku gdy słowo kluczowe dyrektywy jest poprzedzone dwoma adresami, to wyznaczają one zakres działania tej dyrektywy w obrębie poprawianego tekstu. Obowiązuje w związku z tym zasada, że numer linii wyznaczanej przez pierwszy adres nie może być większy niż numer linii reprezentowanej przez drugi adres. Użycie średnika zamiast przecinka jako separatora adresów w takim przypadku powoduje, że przed rozpoczęciem wyszukiwania linii wskazanej dru-

gim adresem status linii ostatnio aktualizowanej otrzymuje linia o numerze reprezentowanym przez pierwszy adres. Innymi słowy wartość symbolu kropki jako adresu symbolicznego bieżącej linii tekstu staje się równa numerowi linii reprezentowanej pierwszym adresem*. Ma to istotne znaczenie w przypadku, gdy drugi adres jest zdefiniowany przy pomocy symbolu kropki jako adresu bieżącej linii. Tak więc w następujących przykładach części adresowej określonej dyrektywy

/ALA/,.+5 oraz /ALA/;.+5

drugi adres na ogół nie musi oznaczać tej samej linii w obydwu definicjach zakresu działania dyrektywy.

Znak wykrzyknika przed słowem kluczowym występuje tylko w przypadku czterech dyrektyw systemu operacyjnego SOM, które są legalne przy pracy pod kontrolą programu EDM i służy właśnie do wyróżnienia tychże dyrektyw z grona pozostałych, jako że są one realizowane nie przez program EDM, a przez odpowiednie procedury (ekstrakody) systemu operacyjnego. W szczególności oznacza to, że EDM nie sprawdza ich legalności i wszelkie błędy popełnione przy ich pisaniu powodują w konsekwencji "abort" programu EDM wskutek błędu systemowego.

Postać parametrów związana jest ze specyfiką konkretnej dyrektywy i dlatego musi być każdorazowo objaśniana w momencie jej prezentacji. Często pole parametrów dyrektywy podawanej przez użytkownika jest puste, co wiąże się zarówno z tym, że wiele dyrektyw nie posiada parametrów jak i z tym, że użytkownik może korzystać z założonych w edytorze wartości parametrów w przypadku dyrektyw, których definicja przewiduje ich wystąpienie.

Znak wykrzyknika po słowie kluczowym dyrektyw może występować w przypadku tych dyrektyw, których definicja przewiduje dwa warianty wykonania zlecanych operacji. Brak lub wystąpienie wykrzyknika w przypadku

* Z tej racji znak kropki w funkcji adresu linii wygodnie jest utożsamiać z pojęciem wskaźnika (kursora) definiującego zawsze aktualne położenie punktu pracy w obrębie linii poprawianego tekstu znajdującego się w zbiorze roboczym.

takich dyrektyw wskazuje który wariant wykonania użytkownik wybiera.

W dyrektywach, które pozwalają zmieniać tekst w zbiorze roboczym mogą wystąpić tzw. opcje, które mają postać pojedynczych liter "P", "L", "C" oraz "G" umieszczanych na końcu dyrektywy. Pominięcie opcji oznacza, że jest ona nieaktywna. Znaczenie tych opcji nie jest jednolite w odniesieniu do wszystkich dyrektyw, w których mogą się one pojawić. Stąd też będziemy je również objaśniać dla konkretnych sytuacji.

Przy prezentacji poszczególnych dyrektyw będziemy stosować następujące zasady:

1. W definicji ogólnej postaci każdej dyrektywy będziemy zaznaczać tylko te elementy, które mają istotne znaczenie dla jej wykonania. Przykładowo, pominięcie w zapisie ogólnej postaci opcjonalnego znaku wykrzyknika po słowie kluczowym oznacza automatycznie, że dana dyrektywa ma jeden wariant wykonania, bądź też, że drugi wariant ma odrębną nieco składnię pokazaną oddzielnie.
2. Z uwagi na to, że EDM dopuszcza stosowanie skrótów w odniesieniu do słów kluczowych będziemy w definicji postaci dyrektywy podkreślać tę część słowa kluczowego, która wystarcza edytorowi do jego identyfikacji.
3. Oprócz postaci ogólnej będzie podawana tzw. wersja standardowa dyrektyw określająca przyjmowane przez EDM wartości poszczególnych elementów definicji w sytuacji, gdy użytkownik nie poda ich explicite. W znakomitej większości przypadków dotyczyć to będzie zakładanych postaci części adresowych dyrektyw. Ogólna postać składni dyrektywy jak i jej wersja standardowa będą podawane w jednym bloku według następującego, jednolitego schematu:

ogólna postać definicji
wersja standardowa

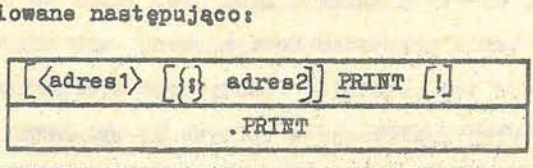
Nie podanie wersji standardowej oznaczać będzie jej brak lub identyczność z postacią ogólną (ta druga sytuacja jest typowa dla dyrektyw, które składają się tylko ze słowa kluczowego).

Poszczególne dyrektywy będą omawiane w pewnych grupach tematycznych, co powinno ułatwić Czytelnikowi posługiwanie się ich opisem.

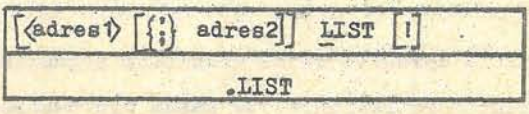
4.4.4. Dyrektywy umożliwiające przeglądanie tekstu

Omówione w tym punkcie dyrektywy pozwalają wypisywać na strumieniu CO, a więc tym samym wyświetlać lub drukować w zależności od typu terminala, pojedyncze linie, lub zadane fragmenty tekstu ze zbioru roboczego.

Dyrektywami najczęściej używanymi do tego celu są dyrektywy LIST oraz PRINT zdefiniowane następująco:



oraz



Różnica w działaniu pomiędzy nimi polega na innymi traktowaniu kodów znaków nie mających drukowalnych odpowiedników (tzw. kodów znaków specjalnych według standardu ISO-7). Dyrektywy PRINT wypisuje szesnastkowe kody takich znaków poprzedzając każdy z nich znakiem "\ ", a dyrektywa LIST wyprowadza je bez zmian. Zakres wyprowadzanych linii, a co za tym idzie ich ilość jest wyznaczana przez część adresową dyrektyw. Ponieważ systemy operacyjne SOM nie dają możliwości przerwania działania żadnego programu przy pomocy znaku wprowadzanego z klawiatury, więc w edytorze założone ograniczenie na maksymalną liczbę jednorazowo wypisywanych linii. Ograniczenie to jest obowiązujące nie tylko w przypadku wyżej wymienionych dyrektyw, ale również w przypadku dyrektyw innych aniżeli PRINT oraz LIST jeżeli zostaną one użyte z opcjami "P" lub "L". Ograniczenie, o którym mowa, jest ustalane w momencie instalacji i standardowo wynosi:

- 21 linii w wersji dla monitora ekranowego jako urządzenia listującego,
- 60 linii w przypadku terminala typu DZM-180-KSR.

Po osiągnięciu ograniczenia EDM zawiesza wykonanie dyrektywy oczekując na reakcję użytkownika, który w takim przypadku może (podkreślono niezbędne znaki odpowiedzi - pozostałe można opuscić):

1. Odpowiedzieć YES lub nacisnąć klawisz "CR" jeśli chce kontynuować wydruk,
2. Odpowiedzieć NO lub nacisnąć klawisz "ESC" by zakończyć wydruk. W przypadku gdy wydruk jest efektem użycia opcji "P" lub "L" innych dyrektyw aniżeli LIST oraz PRINT to działanie dyrektywy jest w takim przypadku kontynuowane, ale bez wydruku. Ważnym wyjątkiem od tej ostatniej zasady jest wykonanie dyrektywy GLOBAL, której realizacja w opisywanej sytuacji zostanie przerwana.

Wariant wykonania dyrektyw PRINT oraz LIST sygnalizowany wykrzyknikiem po słowie kluczowym powoduje, że każda wyprowadzana linia będzie poprzedzona swoim numerem bezwzględnym (adresem bezwzględnym) w obrębie zbioru roboczego.

Zgodnie z postacią standardową dyrektywy PRINT oraz LIST podanie ich bez części adresowej jest równoznaczne z poleceniem wypisania bieżącej linii poprawianego tekstu (linii reprezentowanej adresem kropki).

Po wykonaniu każdej z omawianych dyrektyw adres kropki otrzymuje wartość numeru ostatniej z wyprowadzonych linii tekstu.

Dopuszczalne jest używanie "zdegenerowanych" postaci dyrektywy LIST, które upraszczają zlecenie wypisywania pojedynczych linii tekstu na terminalu. I tak:

1. Wysłanie pustej linii (tylko znak "CR") lub znaku "+" (plus)

powoduje wypisanie następnej po bieżącej linii tekstu (to jest

linii o adresie .+1) oraz ustawienie adresu kropki na tej linii. Umożliwia to krokowe przeglądanie tekstu linia po linii.

2. Naciśnięcie klawisza "ESC" lub wysłanie znaku "-" (minus)

spowoduje wyświetlenie poprzedniej, w stosunku do bieżącej, linii oraz ustawienie na niej adresu kropki. Daje to te same możliwości co w poprzednim przypadku, tylko przeglądanie odbywa się w kierunku malejących numerów linii.

3. Wysłanie tylko pojedynczego adresu linii, to jest użycie dyrektywy postaci

⟨adres⟩

spowoduje wyświetlenie linii określonej adresem oraz ustawienie na niej adresu kropki.

Uzyskanie bezwzględnego numeru linii reprezentowanej określonym adesem (najczęściej kontekstowym) umożliwia dyrektywa postaci:

⟨adres⟩ =
. =

Efektem jej wykonania jest wypisanie numeru linii reprezentowanej podanym adresem.

Wygodną możliwość obejrzenia wskazanego, spójnego fragmentu tekstu (okienka tekstu) daje dyrektywa postaci:

[⟨adres⟩] Z [I] [{⟨liczba⟩}]
. Z

Efektem jej wykonania jest wyświetlenie (lub wydrukowanie) okienka tekstu zawierającego wskazaną adresem linię tekstu, bądź też redefinicja rozmiarów okienka w sensie ilości linii. Znaczenie parametrów jest następujące:

parametr	znaczenie
pominięty	wyświetlenie okienka ze wskazaną adresem linią pośrodku
+	wyświetlane okienko rozpoczyna się wskazaną linią
-	wskazana adresem linia jest ostatnią linią okienka
liczba	podana jako parametr liczba naturalna jest zaokrąglana w górę do najbliższej nieparzystej i definiuje nowy rozmiar okienka tekstu obowiązujący dla wykonania następnych dyrekty Z
D	powoduje powrót do standardowego rozmiaru okienka, który wynosi 23 linie

Po zrealizowaniu dyrektywy Z adres kropki przyjmuje wartość numeru ostatniej z wyświetlonych linii okienka.

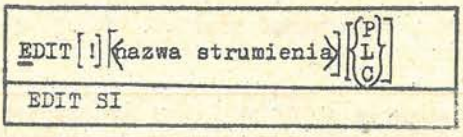
Użycie wariantu dyrektywy z wykrzyknikiem bezpośrednio po literze Z spowoduje, że wyświetlane linie opatrzone będą swoimi numerami.

Przykłady:

postać dyrektywy	opis działania
1,$ LIST	- wypisanie wszystkich linii tekstu
L!	- wydruk numer i treści bieżącej linii
/ALA/;.+5 P!	- wypisanie sześciu kolejnych linii tekstu począwszy od linii, w której po raz pierwszy, licząc od bieżącej linii, pojawi się tekst "ALA". Linie będą opatrzone numerami, a kody znaków nie czerniących papieru będą drukowane w sposób opisany dla dyrektywy PRINT,
$	- wypisanie ostatniej linii tekstu i ustawienie na niej adresu kropki
/[*]/ Z +	- wypisanie okienka tekstu rozpoczynającego się linią, w której po raz pierwszy napotkano znak gwiazdki,
Z 14	- ustawienie rozmiarów okienka tekstu wyświetlanego dyrektywami Z na 15 linii.

4.4.5. Przepisywanie tekstu. Dyrektywy EDIT, READ oraz WRITE

W celu przepisania tekstu ze zbioru zewnętrznego do zbioru roboczego edytora należy posłużyć się jedną z dwóch komend: EDIT lub READ Składnia dyrektywy EDIT jest następująca:



Działanie tej dyrektywy jest następujące:
dotychczasowa zawartość zbioru roboczego jest niszczona i wczytywane są kolejne linie tekstu ze strumienia określonego parametrem aż do napotkania znacznika końca zbioru, którym jest linia zawierająca wyłącznie dwa znaki dolara ($$). Zgodnie z podaną wersją standardową pominięcie parametru jest równoznaczne z wprowadzaniem tekstu z urządzenia przyłączonego do strumienia SI.

Jeżeli w momencie wykonania dyrektywy EDIT zbiór roboczy nie jest pusty, a do tekstu znajdującego się w nim były wnoszone jakiekolwiek poprawki i poprawiona wersja nie została w całości zapisana dyrektywą WRITE to EDM wysyła użytkownikowi ostrzeżenie o możliwości nieumyślnego zniszczenia tekstu i nie wykonuje dyrektywy EDIT. Jeśli użytkownik godząc się na zniszczenie dotychczasowej zawartości zbioru roboczego chce przeforsować dyrektywy EDIT to musi ją podać w wersji z wykrzyknikiem po słowie kluczowym.

Znaczenie opcji, które mogą pojawić się w dyrektywie EDIT jest następujące:

- Opcje P (Print) oraz L (List) powodują, że każda wczytywana linia tekstu jest wypisywana na urządzeniu listującym terminala przyłączonym do strumienia CO. Różnica w działaniu obydwóch opcji jest dokładnie taka sama jak różnica w wykonaniu omówionych już dyrektyw PRINT oraz LIST

- Opcja C (Confirm) powoduje, że EDM przed umieszczeniem każdej wczytanej linii tekstu wypisuje ją na strumieniu CO (podobnie jak w przypadku opcji L) po czym oczekuje na decyzję użytkownika.

Użytkownik w tym momencie może:

1. Wysłać odpowiedź [YES] lub nacisnąć klawisz ["CR"] aby zaakceptować włączenie danej linii do zbioru roboczego.
2. Odpowiedzieć [NO] lub nacisnąć klawisz ["ESC"] aby daną linię odrzucić.
3. Odpowiedzieć poleceniem [CONTINUE] aby zaakceptować daną linię i wszystkie pozostałe jakie jeszcze zostaną wczytane. Od momentu podjęcia takiej decyzji dyrektywa EDIT wykonuje się tak jak gdyny została podana bez opcji C.
4. Przerwać proces wykonywania dyrektyw poleceniem [ABORT].

Jeżeli wczytywana linia liczy więcej znaków aniżeli aktualnie zdefiniowana maksymalna długość linii (porównaj opis dyrektywy FORMAT) to postępowanie programu uzależnione jest od ustawienia wskaźnika przepełnienia OVF (por. opis dyrektywy OVERFLOW). Jeżeli wskaźnik ten nie jest ustawiony, co oznacza, że przepełnienie linii jest niedopuszczalne, to wtedy wprowadzana linia tekstu, obcięta, do ilości znaków wynikającej z aktualnej definicji maksymalnego rozmiaru linii, pojawia się na strumieniu CO i edytor oczekuje na decyzję użytkownika. Użytkownik ma do dyspozycji dokładnie ten sam repertuar możliwości co opisany wyżej z tym, że w tym przypadku odpowiedź CONTINUE dodatkowo powoduje ustawienie wskaźnika OVF.

Jeżeli wskaźnik OVF jest ustawiony, to EDM w razie potrzeby obcina zbyt długie linie nie informując o tym użytkownika.

Przykłady:

Dyrektywa postaci

E SI

spowoduje przepisanie do zbioru roboczego tekstu ze zbioru aktualnie dostępnego na strumieniu SI jeśli wstępnie zbiór roboczy był pusty,

lub jego zawartość ma status posiadającej kopie (np. wczytany uprzednio tekst do zbioru roboczego nie był zmieniany albo przed wykonaniem dyrektywy EDIT został w całości przepisany do zbioru zewnętrznego dyrektywą WRITE). Jeśli tak to dyrektywa zostanie wykonana. W przeciwnym przypadku nie będzie wykonania dyrektywy, a użytkownik zostanie poinformowany o zaistniałej sytuacji stosownym komunikatem.

W przypadku dyrektywy EDIT postaci

E! SO C

poprzednia zawartość zbioru roboczego jest niszczona bez względu na swój status i na jej miejsce jest wprowadzany tekst z aktualnie dostępnego zbioru na strumieniu, tym razem SO. Przed wprowadzeniem do zbioru każdej z wczytywanych linii EDM oczekuje decyzji użytkownika na omówionych wyżej zasadach.

W przypadku, gdy zachodzi potrzeba poszerzenia zbioru roboczego o tekst pochodzący ze zbioru zewnętrznego należy posłużyć się dyrektywą READ postaci:

⟨adres⟩ READ [!] [⟨strumień⟩] [{P, L, C}]
$ READ SI

Sposób działania dyrektywy READ jest analogiczny jak w przypadku dyrektywy EDIT z tą jedną różnicą, że zawartość zbioru roboczego nie jest niszczona, a wczytywane linie są kolejno umieszczane za linią wskazaną adresem poprzedzającym słowo kluczowe.

Przykłady:

Komenda postaci

R

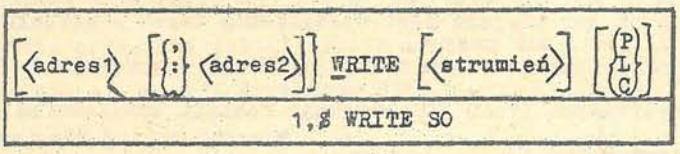
spowoduje wczytywanie linii tekstu ze strumienia SI i umieszczanie ich za ostatnią linią dotychczasowej zawartości zbioru roboczego (zgodnie z założonymi standardowymi wartościami adresu i parametru dyrektywy).

Znaczenie i sposób użycia opcji są dokładnie takie same jak w przypadku EDIT. Dlatego też użycie przykładowo komendy

/END/ R SI P

spowoduje umieszczanie wczytywanego ze strumienia SI tekstu i umieszczanie go za linią w zbiorze roboczym za linią zawierającą tekst "END" ale dodatkowo, każda wczytywana linia będzie wypisywana na strumieniu CO.

Dyrektywą, która realizuje odwrotny transfer poprawionego tekstu jest dyrektywa WRITE.



W wyniku wykonania dyrektywy WRITE linie poprawionego tekstu wskazane adresami są zapisywane jako kolejny zbiór na strumieniu, którego nazwa jest parametrem dyrektywy (standardowo jest to strumień SO). Sposób wyprowadzania, a dokładniej format linii, jest uzależniony od rodzaju urządzenia do którego jest przypisany strumień wymieniony w dyrektywie. Jeśli jest to urządzenie znakowe (np. perforator tasiemki papierowej) to wtedy linie są wyprowadzane jako tzw. rekordy znakowe nieskompresowane. W przypadku, gdy strumień jest przyłączony do sekcji dyskowej to zapisywane w zbiorze zewnętrznym rekordy mają postać rekordów znakowych skompresowanych. Użytkownik nie ma na to żadnego wpływu i stanowi to w związku z tym pewien problem w przypadku potrzeby przygotowania na dysku zbioru danych dla programów napisanych w języku FORTRAN. Zagadnieniem tym zajmiemy się oddzielnie nieco dalej.

Jeżeli przy pomocy dyrektywy WRITE zostaną przepisane na wskazany strumień wszystkie linie zbioru roboczego, to wówczas zbiór ten uzyskuje status "posiadający kopię", co pozwala go zniszczyć dyrektywami EDIT, RESET lub INIT.

Działanie opcji jest identyczne jak w przypadku dyrektywy EDIT

z tym, że dotyczy linii wyprowadzanych ze zbioru roboczego.

Przykłady:

Dyrektywa postaci

W

odpowiada pokazanej w definicji wersji standardowej i w związku z tym spowoduje przepisanie całej aktualnej zawartości zbioru roboczego jako kolejnego zbioru na strumieniu SO.

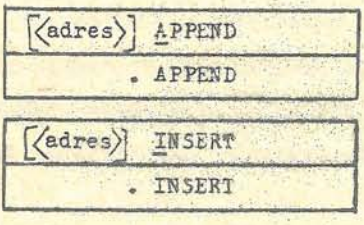
W przypadku dyrektywy postaci

$-50,$ W SO C

kolejny zbiór na strumieniu SO będzie zawierał 51 ostatnich linii zbioru roboczego edytora. Proces zapisywania będzie miał analogiczny przebieg jak proces czytania dyrektywami EDIT oraz READ przy aktywnej opcji C. Innymi słowy przed zapisaniem każdej linii na strumień SO EDM będzie oczekiwał na jedną z czterech możliwych decyzji użytkownika, które zostały omówione przy opisie znaczenia opcji C dla dyrektywy EDIT.

4.4.6. Wprowadzanie, kasowanie i wymiana pełnych linii tekstu. Dyrektywy APPEND, INSERT, DELETE oraz CHANGE

Dyrektywa READ umożliwiała włączanie do zbioru roboczego linii tekstu ze zbioru znajdującego się na określonym strumieniu. Wprowadzanie pełnych linii z klawiatury terminala umożliwiają użytkownikowi dyrektywy APPEND oraz INSERT o następującej składni:



Działanie dyrektywy APPEND jest następujące. Wszystkie następne linie pisane z klawiatury (a ściślej wprowadzane ze strumieni CI, który

standardowo jest przyłączony do klawiatury) i wysyłane przy pomocy klawisza "CR" są umieszczane (wstawiane) za linią wskazaną adresem podanym w komendzie. Proces ten jest kontynuowany, aż do wysłania linii zawierającej wyłącznie znak kropki, co powoduje zakończenie działamia komendy APPEND.

Proces wstawiania tekstu przy pomocy dyrektywy INSERT przebiega identycznie z tym, że wprowadzane linie umieszczane są przed linią określoną częścią adresową komendy.

Warto w tym miejscu dodać, że w części adresowej komendy APPEND można odwołać się do linii o numerze zero. Zerowa linia zbioru roboczego istnieje tylko w sensie logicznym i nie należy do poprawianego tekstu. W szczególności wynika stąd, że linii tej nie można ani usunąć, ani zmienić ani nawet zaadresować poza szczególnym przypadkiem jakim jest właśnie komenda APPEND. Definicja istnienia takiej linii nie związanej z poprawianym tekstem umożliwia przy pomocy instrukcji postaci

Ø APPEND

wprowadzić z klawiatury linie tekstu do pustego zbioru roboczego, czyli innymi słowy utworzyć tekst w tym zbiorze.

Przykłady:

W wyniku następującego fragmentu pracy z edytorem

```
ØA
C PROGRAM ROZWIAZYWANIA UKŁADU ROWNAN
C METODA ELIMINACJI GAUSSA
.
```

na początek poprawianego tekstu w zbiorze roboczym zostaną włączone wypisane w przykładzie dwie linie. Jeśli wstępnie zbiór był pusty to powyższy przykład odpowiada wykreowaniu tekstu składającego się z dwóch linii. Zauważmy, że w przypadku niepustego zbioru roboczego zastąpienie w powyższym przykładzie dyrektywy APPEND przez INSERT w następującej postaci

1 I

da dokładnie ten sam rezultat (dołożenie dwóch linii na początek tekstu).

Przy pomocy dyrektywy INSERT nie można jednak ani utworzyć tekstu w zbiorze roboczym, ani też dopisać linii na końcu tekstu. Z tego względu można twierdzić, że przy pomocy APPEND można zrealizować wszystko to co i przy użyciu dyrektywy INSERT, natomiast odwrotne twierdzenie nie jest prawdziwe. Przykładowo użycie dyrektywy INSERT w następującej sekwencji poleceń

```
?REAL? I
        LOGICAL SWITCH
.
```

można zastąpić dyrektywą APPEND w następujący sposób:

```
?REAL?-1 A
     LOGICAL SWITCH
.
```

W obydwu przypadkach efektem będzie wstawienie linii tekstu
" LOGICAL SWITCH" przed linię zawierającą tekst "REAL".

Jeżeli długość linii tekstu wprowadzanej dyrektywami APPEND i INSERT jest większa niż aktualnie zdefiniowana maksymalna długość linii i wska- i wskaźnik przepełnienia linii OVF nie jest ustawiony, to wtedy EDM wypisuje na strumieniu CO obciętą linię i czeka na decyzję użytkownika, który ma do wyboru następujące możliwości:

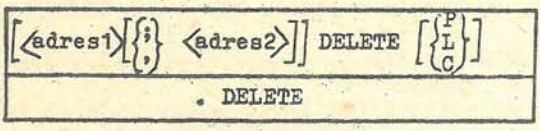
- odpowiedzieć [YES] lub nacisnąć klawisz ["CR"] aby zaakceptować linię,
- odpowiedzieć [NO] , [ABORT] lub nacisnąć klawisz ["ESC"] jeśli linia ma być odrzucona,
- napisać i wysłać słowo [CONTINUE] aby zaakceptować linię, i ustawić wskaźnik przepełnienia OVF.

Jeśli wskaźnik OVF jest ustawiony wówczas EDM automatycznie obcina za długą linię i wprowadza ją do zbioru roboczego nie informując użytkownika o wystąpieniu przepełnienia.

Dużym udogodnieniem przy wprowadzaniu linii tekstu z klawiatury jest możliwość korzystania ze znaku tabulacji definiowanego przy pomocy dyrektywy FORMAT w sposób omówiony w następnym punkcie.

W przypadku obydwóch dyrektyw adres kropki po zakończeniu ich działania jest ustawiany na ostatniej z wprowadzonych linii.

Dyrektywą, która pozwala usuwać zbędne linie tekstu ze zbioru roboczego jest dyrektywa DELETE o następującej definicji:



Powoduje ona usunięcie wskazanych częścią adresową linii tekstu. Użycie opcji L lub P powoduje, że usuwane linie są wypisywane na strumieniu CO. Opcja C pozwala użytkownikowi kontrolować proces usuwania linii. W przypadku, gdy jest ona wymieniona w dyrektywie EDM wypisuje na strumieniu CO każdą linię, którą ma zamiar usunąć i oczekuje od użytkownika potwierdzenia (według reguł opisanych dla realizacji dyrektywy EDIT z aktywną opcją C).

Po wykonaniu dyrektywy DELETE adres bieżącej linii (adres kropki) zostaje ustawiony na pierwszej linii, która nie została usunięta. Jeżeli usunięta została ostatnia linia, to adres ten pokazuje na linię, która jest ostatnia po zakończeniu działania dyrektywy DELETE. W celu usunięcia pojedynczej linii wystarczy podać tylko jeden adres. Podanie tylko słowa kluczowego (lub jego skrótu) spowoduje usunięcie linii o adresie kropki.

Przykłady:

dyrektywa postaci:

$ D P

spowoduje wypisanie i usunięcie ostatniej linii tekstu.

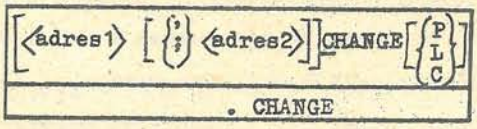
Usunięcie jedenastu linii poczynając od pierwszej linii zawierającej wystąpienie tekstu "ALA" można zlecić edytorowi dyrektywą DELETE w następujący sposób:

/ALA/;.+10D C

Użycie opcji C w tym przykładzie, spowoduje, że przed usunięciem każdej linii edytor będzie ją wypisywał i oczekiwał akceptacji użytkownika.

Odpowiedź twierdząca w tym przypadku oznaczać będzie zgodę na usunięcie wypisanej linii.

Dyrektywą, która łączy w sobie działanie dyrektyw DELETE oraz INSERT jest dyrektywa CHANGE o następującej postaci:

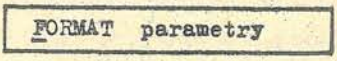


Pozwala ona zastąpić wskazane linie tekstu nowymi liniami wprowadzonymi przez użytkownika z klawiatury terminala. W pierwszym kroku linie wskazane częścią adresową są usuwane według procedury opisanej dla dyrektywy DELETE. W następnym kroku działanie dyrektywy odpowiada procedurze wykonania dyrektywy INSERT.

4.4.7. D y r e k t y w a FORMAT

Dyrektywa FORMAT służy do deklarowania logicznego rozmiaru linii tekstu oraz do definiowania znaku jak i kolumn tabulacji.

Ogólna postać tej dyrektywy jest następująca:



Parametry tej dyrektywy mogą być dwojakiego rodzaju:

1. L ⟨liczba⟩
2. T= ⟨znak⟩,⟨kol1⟩ [, kol2][,⟨kol3⟩][,⟨kol4⟩][,⟨kol5⟩][, ⟨kol6⟩]

Użycie dyrektywy FORMAT z parametrem o pierwszej postaci pozwala nam zmienić maksymalny rozmiar linii w znakach. Z tego powodu liczba pojawiająca się w tym parametrze po literze "L" i oznaczająca nowy rozmiar linii nie może być ujemna, ani większa od 257. Standardowo (co jest ustalane przy instalacji) maksymalny rozmiar linii jest określony na 80 znaków. Chcąc go zmienić na przykład na 72 znaki użyjemy dyrektywy FORMAT następującej postaci:

F L72

Druga postać parametru pozwala nam zdefiniować znak tabulujący oraz kolumny odpowiadające kolejnym pojawieniom się tego znaku. Przykładowo dyrektywa FORMAT postaci:

F T=;,7,72

definiuje średnik jako znak tabulacji oraz kolumny o numerach 7 i 72 jako kolumny tabulacji odpowiadające pierwszemu i drugiemu pojawieniu się znaku średnika w treści linii tekstu wprowadzanego do zbioru roboczego. Tak ustawiony format linii jak w powyższym przykładzie będzie nader użyteczny przy wprowadzaniu z klawiatury tekstu programu w FORTRANIE, w którym pole instrukcji zaczyna się od siódmej kolumny, a pole komentarza od kolumny o numerze 73.

W jednej dyrektywie FORMAT można użyć obydwu rodzajów parametrów w związku z czym zapis

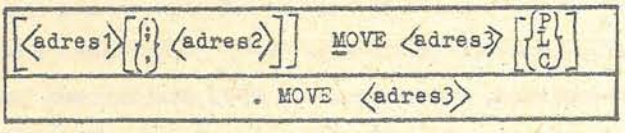
F L110 T=!,20,40,60,80,100

jest poprawnym przykładem użycia dyrektywy FORMAT

Należy pamiętać, że obowiązuje zasada, że zmiana rozmiaru linii przy pomocy dyrektywy FORMAT jest dopuszczalna tylko w sytuacji, gdy zbiór roboczy nie jest pusty.

4.4.8. Przemieszczanie, powielanie, łączenie i markowanie linii. Dyrektywy MOVE, TRANSCRIBE, JOIN oraz MARK

Dyrektywą, kóra umożliwia przeniesienie wskazanego fragmentu tekstu w inne miejsce zbioru roboczego jest dyrektywa MOVE postaci:



Działanie tej dyrektywy polega na przesunięciu linii wskazanych adresami poprzedzającymi słowo kluczowe za linię, której adres jest parametrem dyrektywy MOVE. Adres ten nie może reprezentować żadnej

z przesuwanych linii. Znaczenie opcji P,L oraz C jest analogiczne do opisanego przy komendzie EDIT z tym, że dotyczy przesuwanych linii.

Przykłady:

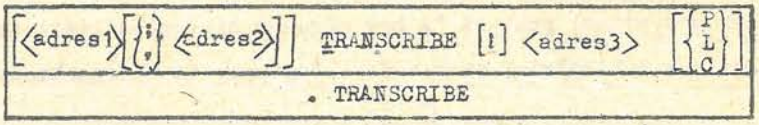
1 M $ P

Pierwsza linia jest wypisywana i przesuwana na koniec zbioru roboczego.

M .+1

Zgodnie ze standardową wersją dyrektywy powyższy przykład oznacza polecenie zamiany miejscami linii bieżącej i następnej po niej.

W sytuacji gdy zchodzi potrzeba powielania określonego fragmentu poprawianego tekstu możemy taką operację zlecić edytorowi następującą dyrektywą:



Dyrektywa TRANSCRIBE powoduje umieszczenie kopii linii wskazanych częścią adresową za linią o adresie będącym jej parametrem. Wszystkie elementy definicji dyrektywy TRANSCRIBE mają analogiczne znaczenie jak w przypadku dyrektywy MOVE. Należy jedynie pamiętać że standardowo nie są tworzone fizyczne kopie powielanych linii. Chcąc ten stan rzeczy zmienić należy użyć wersji dyrektywy z wykrzyknikiem po słowie kluczowym.

Przykłady:

T $

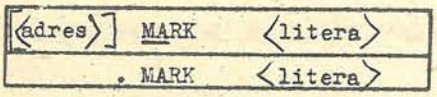
Bieżąca linia jest kopiowana na początek poprawianego tekstu. Należy zauważyć, że TRANSCRIBE jest dyrektywą, w której jako parametru wolno nam użyć fikcyjnej linii o adresie zero.

Chcąc podwoić istniejący tekst z warunkiem utworzenia fizycznych kopii poszczególnych linii należy użyć dyrektywy TRANSCRIBE w postaci:

1,$ T! $

Wymóg zworzenia fizycznych kopii powielanych linii ma istotne znaczenie w sytuacji, gdy mają być one obiektami działania dyrektywy MARK.

Dyrektywa ta ma następującą definicję:



Pozwala ona zdefiniować symbol złożony z litery będącej jej argumentem poprzedzonej znakiem apostrofu ' jako adres linii o adresie podanym przed słowem kluczowym na tych samych zasadach jak znak kropki oznacza adres bieżącej linii. Adres taki wolno nam używać na prawach adresu kontekstowego tak długo, dopóki wyznaczana nim linia nie zostanie usunięta lub zmieniona. Z racji postaci dyrektywy MARK operację przezeń realizowaną utarło się określać jako markowanie linii literą.

Przykład:

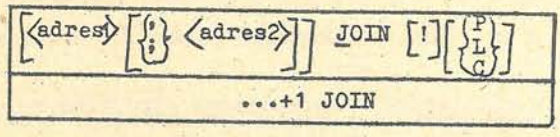
Chcemy aby symbol postaci 'A był adresem pierwszej linii wstecz, która zawiera jakikolwiek znak cyfry. Uzyskamy to pisząc komendę

? [0-9] ? MA A

Jeśli teraz zachodzi potrzeba usunięcia fragmentu tekstu zaczynającego się od dziesiątej linii następującej po zamarkowanej linii do końca tekstu, to wystarczy użyć dyrektywy DELETE w następujący sposób:

'A + 10, $ D

Ostatnią w omawianej grupie dyrektyw operowania na pełnych liniach tekstu w zbiorze roboczym jest dyrektywa JOIN:



Efektem wykonania dyrektywy JOIN jest połączenie w jedną linię linii, które są wskazywane częścią adresową dyrektywy. W przypadku, gdy podany jest tylko adres jednej linii, lub gdy oba adresy poprzedzające słowo kluczowe wskazują na tę samą linię, przyjmuje się, że łączeniu podlega określona pierwszym adresem oraz linia bezpośrednio występująca po niej. Normalnie teksty łączonych linii oddzielane są w linii wynikowej jednym znakiem spacji. Jeśli spacja ta jest zbędna,

to można jej uniknąć używając wersji dyrektywy z wykrzyknikiem po słowie kluczowym. Opcje P, L i C posiadają zwykłe znaczenie opisane wyczerpująco przy poprzednich dyrektywach. W sytuacji, gdy linia wynikowa przekroczy zadeklarowany maksymalny rozmiar linii działania programu EMD zależy w uprzednio opisany sposób od ustawienia wskaźnika przepełnienia OVF. Z postaci wersji standardowej dyrektywy wynika, że podanie samego słowa kluczowego (lub jego skrótu) oznacza polecenie połączenia bieżącej linii tekstu z następną.

Przykład:

$ - 2 ; $ J C

trzy ostatnie linie tekstu zostaną połączone w jedną linię, w obrębie której teksty linii składowych będą oddzielone pojedynczym znakiem spacji. Przed wpisaniem linii wynikowej do zbioru edytor będzie oczekiwał potwierdzenia użytkownika zgodnie z typową zasadą działania opcji C. W wyniku operacji liczba linii w zbiorze zmniejszy się o dwa.

4.4.9. Wymiana fragmentów linii tekstu. Dyrektywa SUBSTITUTE

Dyrektywa SUBSTITUTE jest najsilniejszą dyrektywą korektorską edytora EDM. Pozwala ona wymienić określony wzorcem fragment linii tekstu na inny, zadany parametrem ciąg znaków bez konieczności przepisywania całej treści linii. W połączeniu z dyrektywą GLOBAL omówioną w następnym punkcie dyrektywa SUBSTITUTE pozwala w łatwy i elegancki sposób dokonywać systematycznych zmian w wielu liniach poprawianego tekstu. Podobnie jak wiele innych dyrektyw edytora SUBSTITUTE ma dwie wersje wykonania sygnalizowane wystąpieniem znaku wykrzyknika, lub jego brakiem po słowie kluczowym dyrektywy. Ponieważ w przypadku tej dyrektywy obydwie wersje różnią się dodatkowo składnią, więc omówimy je oddzielnie. Zaczniemy od wersji bez wykrzyknika, której składnia ma następującą definicję:

[<adres1>[{;,}<adres2>]] SUBSTITUTE <znak><wzorzec><znak><tekst><znak>[[{P L C G}]]

. SUBSTITUTE <znak><wzorzec><znak><tekst><znak>

Adres przed słowem kluczowym wskazuje linię, która będzie obiektem działania dyrektywy. Jeżeli pierwszym znakiem po słowie kluczowym (różnym od odstępu) nie jest wykrzyknik, to znak ten jest przyjmowany jako ogranicznik, a jednocześnie separator dwóch parame-trów dyrektywy którymi są wzorzec oraz tekst. Wzorzec, na analogicznych zasadach jak w przypadku adresu kontekstowego, opisuje fragment linii, który będzie podlegał wymianie na tekst określony drugim parametrem. Znaczenie użycia opcji P, L i C w odniesieniu do poprawianej linii jest takie samo jak w przypadku dotychczas omówionych dyrektyw, w których opcje te mogły się pojawić. Opcja G pozwala zdefiniować zakres działania dyrektyw SUBSTITUTE w obrębie poprawianej linii. Standardowo wymianie podlega pierwszy, licząc od lewej strony, fragment linii dopasowany do podanego wzorca. Użycie opcji G spowoduje, że działanie to zostanie rozciągnięte na wszystkie wystąpienia tekstu określonego wzorcem w linii.

Przykłady:

S/ALA/OLA/

W bieżącej linii pierwsze od lewej strony wystąpienie tekstu "ALA" zostanie zamienione na tekst "OLA".

100 S B + B - B G

W linii o numerze 100 wszystkie znaki "+" zostaną zamienione na znaki "-".

/SUBROUTINE/S++FUNCTION+ P

Edytor odszukuje pierwszą (od aktualnego miejsca pracy w obrębie poprawianego tekstu) linię zawierającą tekst "SUBROUTINE" i zastępuje w niej ten tekst (pusty wzorzec oznacza zawsze poprzedni wzorzec mimo tego, że wystąpił on w części adresowej) napisem "FUNCTION". Zmieniona postać linii zostanie wypisana na strumień CO (opcja P).

Należy także wiedzieć, że wzorzec zawierający jedno lub więcej wystąpienie znaku gwiazdki w charakterze znaku specjalnego jest dopasowywany w ten sposób, aby każda gwiazdka licząc kolejno od lewej strony "objęła" jak największą liczbę znaków. Jeżeli przykładowo aktualnie poprawiana linia (linia o adresie kropki) ma postać

A=(A+B)+C+D/B

to po wykonaniu dyrektywy

S/+.*B/-C)/

linia ta będzie wyglądać tak:

A=(A-C)

gdyż podany w dyrektywie SUBSTITUTE wzorzec zostanie dopasowany w następujący sposób:

A=(A+B)+C+D/B
+.....B

Podobnie w przypadku linii

ALFA+BETA+GAMMA=DELTA+4

poprawianej dyrektywą

S/A.*E [A-Z] *+/B/

proces dopasowania będzie wyglądał następująco

ALFA+BETA+GAMMA=DELTA+4
A.............ELTA+

i w rezultacie po dokonaniu wymiany otrzymamy tekst:

B4

W tekście wymienionym w dyrektywie, który ma zastąpić fragment okre określony wzorcem można użyć znaku "&" (ampersand) jako znaku specjalnego. Oznacza on "wszystko to, co zostało dopasowane do wzorca".

Przykład:

Jeżeli dyrektywa postaci

S/.+./(&)/G

odnosi się do linii

Y=A+B/C+D/E+F

to po jej wykonaniu tekst tej linii będzie wyglądał następująco:

Y=(A+B)/(C+D)/(E+F)

Wprawny użytkownik może dokonywać bardzo wyrafinowanych poprawek przy użyciu dyrektywy SUBSTITUTE korzystając z możliwości zdefiniowania w obrębie wzorca zmienianego tekstu tzw. podwyrażeń. Mianem tym określa się fragment wzorca ujęty w dwuznakowe klamry (...). Każde podwyrażenie w obrębie wzorca otrzymuje numer wyznaczony przez kolejny numer rozpoczynającej go klamry. Maksymalnie można zdefiniować 6 podwyrażeń w obrębie pojedynczego wzorca. Jeśli teraz w podstawianym tekście pojawi się dwuznakowy symbol specjalny postaci "\n", gdzie n jest numerem podwyrażenia zdefiniowanego we wzorcu, to symbol ten zostanie przy podstawianiu zastąpiony przez tekst, który został dopasowany do podwyrażenia o podanym numerze. Te nieco zawiłe reguły staną się znacznie bardziej przejrzyste po uważnym przestudiowaniu następującego przykładu:

Załóżmy że poprawiamy trzy kolejne linie tekstu o następującej treści:

```
K=ALFA+BETA/2
K=K+DELTA/3
K=K-U**4
```

Po wykonaniu dyrektywy

/ALFA/;.+3S/K.\([A-Z] * [+\-] [A-Z] *\)/IK=(\1)/

linie te przyjmą postać:

```
IK=(ALFA+BETA)/2
IK=(K+DELTA)/3
IK=(K-U)**4
```

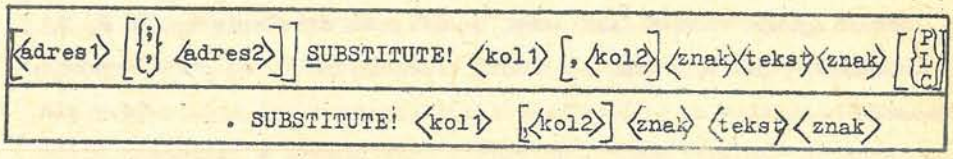
a dopasowanie wzorca do pierwszej linii wyglądać będzie następująco:

```
K=ALFA+BETA/2
K.ALFA+BETA
```

podwyrażenie

Dużym udogodnieniem pozwalającym przykładowo na eleganckie i łatwe formatowanie ramek (komentarze itp.) jest możliwość wpisania podanego

tekstu w miejsce rozpoczynające się wskazanym numerem znaku w linii. Możliwość tę zapewnia drugi wariant wykonania dyrektywy SUBSTITUTE sygnalizowany podaniem wykrzyknika bezpośrednio po słowie kluczowym. Dla tego wariantu składnia dyrektywy SUBSTITUTE jest zdefiniowana w następujący sposób:



Dwie liczby pojawiające się w polu parametrów są numerami kolumn (znaków) poprawianej linii wyznaczającymi miejsce, w które zostanie wpisany tekst, będący drugim parametrem dyrektywy. Tekst ten w dyrektywie musi być ujęty w ograniczniki ⟨znak⟩. Ogranicznikiem może być dowolny znak nie występujący we wstawianym tekście i różny od spacji. W szczególności wstawiany tekst może być pusty (dwa ograniczniki obok siebie) co jest równoznaczne z kasowaniem znaków w poprawianej linii w obszarze wyznaczonym przez numery kolumn. Podanie tylko jednego numeru kolumny powoduje, że program zakłada iż drugi numer jest taki sam. Jeżeli długość poprawianej linii jest mniejsza niż wymagają tego podane numery kolumn, wówczas przed przystąpieniem do wstawiania wskazanego tekstu edytor uzupełni za krótką linię znakami spacji do wymaganej długości.

Przykłady:

W wyniku podania dyrektywy

/^C/;//S!72/*/

w każdej linii występującej pomiędzy najbliższą parą linii zawierających literę "C" na początku zostanie umieszczona gwiazdka w 72 kolumnie

Dyrektywa postaci

1,$ S! 1,6//

spowoduje usunięcie początkowych 6 znaków we wszystkich liniach poprawianego tekstu.

Warto wiedzieć, że wśród znaków wstawianego tekstu może znajować się znak "LF" (LINE FEED) oznaczający wirtualny koniec linii. Należy jednak przy tym pamiętać, że łączna długość wszystkich linii wygenerowanych tym sposobem przez rozbicie pojedynczej linii nie może przekroczyć 256 znaków.

Obsługa przepełnienia logicznego linii oraz działanie opcji P, L, C są analogiczne jak w przypadku innych omówionych do tej pory dyrektyw (np. EDIT).

4.4.10. Dyrektywa organizacyjna GLOBAL

Dyrektywa GLOBAL służy do wykonania listy podanych dyrektyw dla wielu linii poprawianego tekstu. Jej ogólna postać jest następująca:

[⟨adres1⟩ [,;] ⟨adres2⟩] GLOBAL [!] ⟨wzorzec⟩[⟨lista dyrektyw⟩]
1,$ GLOBAL ⟨wzorzec⟩

W typowym przypadku działanie tej dyrektywy jest następujące:

Spośród linii wskazanych parą adresów poprzedzających słowo kluczowe są wybierane (markowane) linie zawierające wystąpienie tekstu opisanego wzorcem będącym pierwszym parametrem dyrektywy. Następnie wszystkie linie z podanego zakresu są przeglądane i dla każdej zaznaczonej linii wykonywana jest lista dyrektyw występująca za wzorcem dyrektywy GLOBAL. Wykonanie listy dyrektyw dla pojedynczej zaznaczonej linii polega na ustawieniu adresu kropki na tej linii i następnie zwykłym wykonaniu podanej listy dyrektyw tak, jakby były one wprowadzane po kolei przez użytkownika. Poszczególne dyrektywy z podanej listy muszą być od siebie oddzielone znakiem "LF" (klawisz LINE FEED).

Ponadto wykonanie dyrektywy GLOBAL podlega następującym zasadom:

1. Jeżeli wzorzec jest poprzedzony znakiem "-" (minus), wówczas wybierane i markowane są te linie, które nie zawierają wystąpienia tekstu określonego wzorcem.

2. Drugi wariant wykonania dyrektywy sygnalizowany wykrzyknikiem po słowie kluczowym polega na tym, że po zakończeniu realizacji listy dyrektyw dla bieżącej linii nowa zamarkowana linia poszukiwana jest od początku zaczynając od pierwszej linii w zbiorze. W wariancie podstawowym (bez wykrzyknika) linia ta jest poszukiwana poczynając od linii o adresie kropki. Należy pamiętać, że adres kropki może być ustawiony na wiele sposobów podczas realizacji listy dyrektyw dla poprzedniej zamarkowanej linii. Informacja ta może mieć szczególne znaczenie w sytuacji, w której na liście dyrektyw do wykonania znajdują się dyrektywy MOVE przesuwające zaznaczone linie (np. na początek poprawianego tekstu).
3. Jeżeli po wzorcu dyrektywy GLOBAL nie pojawia się lista dyrektyw, wówczas EDM przyjmuje, że należy jedynie zaznaczyć wybrane przez część adresową dyrektywy i wzorzec linie tekstu. Pojawienie się kolejnych dyrektyw GLOBAL z pustymi listami dyrektyw powodować będzie kolejne zaznaczenie nowych linii aż do momentu podania dyrektywy GLOBAL zawierającej przynajmniej jedną dyrektywę do wykonania. Wtedy dyrektywy występujące na jej liście zostaną wykonane dla wszystkich sukcesywnie zaznaczonych linii tekstu.

Przykłady:

W wyniku podania dyrektywy postaci

G/SUBROUTINE/P!

zostaną wypisane wraz z numerami (dyrektywa PRINT!) wszystkie linie (standardowe wartości części adresowej) zawierające wystąpienie tekstu "SUBROUTINE".

Przebieg wykonania dyrektywy GLOBAL zapisanej w sposób następujący:

?ALFA?+1;.+50G-/[0-9]/.-1,.+1 P "LF"

.-1S/A/B/GC

odbędzie się według schematu:

- W pierwszym etapie spośród pięćdziesięciu jeden linii wskazanych parą adresów poprzedzających słowo kluczowe wybrane i zaznaczone zostaną te, które nie zawierają wystąpienia cyfry.

- W drugim etapie dla każdej uprzednio zaznaczonej linii zostaną wykonane następujące czynności:
  1. Zostanie wydrukowana trójka kolejnych linii z wybraną linią pośrodku.
  2. Wszystkie wystąpienia litery "A" w wybranej linii zostaną zastąpione przez wystąpienie litery "B" (drugą dyrektywą na liście jest dyrektywa SUBSTITUTE z aktywną opcją G). Przed wpisaniem zmienionej linii do zbioru EDM będzie oczekiwał potwierdzenia użytkownika (aktywna opcja C dyrektywy SUBSTITUTE).

W powyższym przykładzie zapis "LF" symbolizuje naciśnięcie klawisza "LINE FEED" celem oddzielenia dyrektywy PRINT od dyrektywy SUBSTITUTE na liście dyrektyw dyrektyw GLOBAL.

Warto zauważyć, że ewentualna decyzja "ABORT" użytkownika przy realizacji dyrektywy SUBSTITUTE z opcją C w powyższym przykładzie przerwie jedynie aktualną realizację dyrektywy SUBSTITUTE nie przerywając działania dyrektywy GLOBAL.

4.4.11. Kasowanie całego tekstu w zbiorze roboczym. Dyrektywy RESET oraz INIT

Zniszczenie dotychczasowej zawartości zbioru roboczego umożliwiają dwie komendy edytora, których ogólna postać jest następująca:

RESET [!]
INIT [!]

oraz

Jak widać dyrektywy te składają się jedynie ze słowa kluczowego, po którym ewentualnie może wystąpić wykrzyknik sygnalizujący wariant wykonania.

W wyniku wykonania dyrektywy RESET następuje skasowanie tekstu sta-

nowiącego zawartość zbioru roboczego i pełne zainicjowanie programu EDM, tak jak byłby on wywołany od nowa. Jeżeli aktualny tekst w zbiorze roboczym ma status "nie posiadający kopii" to wtedy użytkownik jest ostrzegany o możliwości przypadkowego zniszczenia jedynej wersji poprawianego tekstu i EDM procedury RESET nie wykonuje. Użytkownik może wówczas albo skopiować zawartość zbioru roboczego dyrektywą WRITE i ponownie zlecić wykonanie komendy RESET w normalnym trybie, albo przeforsować, mimo ostrzeżeń edytora, wykonanie dyrektywy umieszczając po słowie kluczowym wykrzyknik.

Dyrektywa INIT jest zbliżona w sposobie działania do dyrektywy RESET z tym, że po jej wykonaniu pozostają obowiązujące poprzednie definicje formatów, opcji itp.

Zauważmy, że skasowanie wszystkich linii poprawianego tekstu jest możliwe przy użyciu dyrektywy DELETE w następujący sposób:

1,$ D

Odpowiada to użyciu dyrektywy INIT z tą różnicą, że edytor zniszczy zawartość zbioru roboczego bez ostrzeżenia również w sytuacji, gdyby poprawiany tekst nie był uprzednio skopiowany.

4.4.12. Zakończenie pracy edytora. Dyrektywa QUIT

Zakończenie pracy programu EDM i przekazanie sterowania systemowi operacyjnemu następuje w wyniku wykonania dyrektywy postaci:

QUIT [!]

Ponieważ zakończenie pracy edytora oznacza bezpowrotną utratę dostępu do informacji zawartej w zbiorze roboczym więc użytkownik jest ostrzegany o możliwości zniszczenia poprawionej wersji tekstu w analogicznej sytuacji i na analogicznych warunkach jak w przypadku przeforsować wykonanie dyrektywy mimo ostrzeżeń ze strony programu EDM.

4.4.13. Dyrektywy pomocnicze

Omówione w tym punkcie dyrektywy mają charakter pomocniczy i pozwalają zmieniać, bądź przywracać standardowe wartości pewnych parametrów mających wpływ na pracę edytora.

Pierwszą z nich jest dyrektywa LIMIT postaci:

LIMIT { ⟨liczba⟩ | D }

Podanie tej dyrektywy z liczbą jako parametrem pozwala ustalić nowe ograniczenie na maksymalną ilość linii wydruku produkowanych przez pojedynczą dyrektywę (patrz opis dyrektyw PRINT oraz LIST). Użycie dyrektywy LIMIT z parametrem "D" przywraca standardową wartość tego ograniczenia.

Dyrektywa postaci

SIZE ⟨liczba⟩

której parametrem może być liczba naturalna mniejsza od 2048 pozwala zdefiniować maksymalny rozmiar zbioru roboczego określając jednocześnie fizyczny format zapisywania informacji w zbiorze.

Aby zdać sobie sprawę z jej działania należy wiedzieć, że istnieje ścisły związek pomiędzy rozmiarem zbioru roboczego a fizycznym formatem zapisywanej w nim informacji. EDM rozróżnia dwa takie formaty:

1. Ograniczenie na rozmiar zbioru roboczego wynosi 256 rekordów po 512 skompresowanych rekordów. Format ten zapewnia dużą szybkość zapisu i odczytu informacji.
2. Ograniczenie na rozmiar zbioru roboczego wynosi 1024 rekordy (0-1023) po 480 skompresowanych bajtów.

Ponadto dopuszczalne jest rozszerzenie drugiego formatu tak, aby maksymalna liczba rekordów wynosiła 2048. Rozszerzenie takie uniemożliwia jednak wykonywanie dyrektywy GLOBAL.

Standardowo przyjętym formatem jest format pierwszy z ograniczeniem

wynoszącym 256 rekordów po 512 bajtów skompresowanej informacji. Dla standardowych zastosowań edytora takich jak pisanie programów, danych itp. daje to możliwość operowania tekstami rzędu kilku tysięcy linii. Jeżeli jednak z pewnych względów okaże się, że standardowy format zbioru roboczego jest zbyt mały, to można go zmienić przy pomocy omawianej dyrektywy SIZE. Należy przy tym pamiętać, że:

1. Parametr dyrektywy określa ile rekordów przeznaczamy na zbiór roboczy. Jeśli nowa podana liczba mieści się w ograniczeniach bieżącego formatu i nie jest mniejsza od aktualnego rozmiaru zbioru roboczego, to wtedy dyrektywa SIZE jest legalna bez względu na to czy zbiór roboczy był pusty czy też nie. Jeżeli parametr jest taki, że spowoduje zmianę aktualnego formatu zbioru roboczego, to takiej dyrektywy wolno użyć tylko przy pustym zbiorze roboczym. Zbiór roboczy jest uważany za pusty bezpośrednio po wywołaniu programu EDM bez opcji Uϕ oraz po wykonaniu dyrektyw RESET lub INIT.
2. Dyrektywa SIZE z parametrem większym aniżeli 1023 rozszerza drugi format wyłączając jednocześnie możliwość korzystania z dyrektywy GLOBAL.
3. Użytkownik jest odpowiedzialny za to aby w sytuacji, w której EDM pracuje z dużym formatem zbioru roboczego strumień SC był przypisany do odpowiednio dużej sekcji dyskowej.

Przy przepełnieniu sekcji dyskowej ze zbiorem roboczym EDM sygnalizuje przepełnienie logiczne tak, jakby przekroczona została zadeklarowana liczba rekordów.

Dyrektywą pomocniczą, która umożliwia ustawianie wskaźnika OVF przekroczenia logicznego rozmiaru linii tekstu ustalonego dyrektywą FORMAT jest dyrektywa postaci:

OVERFLOW [!]

Użycie tej dyrektywy bez wykrzyknika powoduje ustawienie wskaźnika

OVF zezwalając tym samym edytorowi obcinanie za długich linii bez informowania o tym użytkownika. Aby wyłączyć wskaźnik OVF należy w powyższej dyrektywie użyć wykrzyknika bezpośrednio po słowie kluczowym.

Użycie dyrektywy VERSION o ogólnej postaci

VERSION [!]

powoduje wydruk nazwy programu, numeru jego wersji oraz następujących informacji:

1. Aktualne ograniczenie na maksymalny rozmiar linii tekstu.
2. Aktualne ograniczenie na liczbę rekordów zbioru roboczego.
3. Aktualny rozmiar zbioru roboczego.

Jeżeli po słowie kluczowym następuje wykrzyknik, wówczas drukowana jest jedynie informacja o aktualnym rozmiarze zbioru roboczego.

4.4.14. Dyrektywy przypisania i pozycjonowania strumieni

Z poziomu edytora EDM użytkownik może odwołać się do czterech procedur systemu operacyjnego. Są to następujące procedury:

ASSIGN - przypisanie strumienia do urządzenia wejścia/wyjścia.

REWIND - przewinięcie (ustawienie na początek) nośnika na urządzeniu przypisanym do strumienia.

AVFILE - przesunięcie nośnika o zadaną ilość znaczników końca zbioru do przodu.

BSFILE - przesunięcie nośnika o zadaną ilość znaczników końca zbioru do tyłu.

Przyjmując oznaczenia:

s - strumień,

sd - urządzenie lub strumień,

n - liczba naturalna,

format dyrektyw edytora odwołujących się do tych procedur zapiszemy w następujący sposób:

```
!ASS  ⟨s⟩ ⟨sd⟩ [⟨s⟩ ⟨sd⟩] [...][⟨s⟩ ⟨sd⟩]
!REW  ⟨s⟩[⟨s⟩] [...] [⟨s⟩]
!AVF  ⟨s⟩ ⟨n⟩ [...] [⟨s⟩ ⟨n⟩]
!BSF  ⟨s⟩ ⟨n⟩ [...] [⟨s⟩ ⟨n⟩]
```

Przykłady:

!ASS SI AM1 SO PP - przypisanie strumienia SI do sekcji dyskowej AM1 oraz strumienia SO do perforatora tasiemki papierowej

!REW SI - przewinięcie nośnika na strumieniu SI

!AVF SI 1 - "przeskoczenie" jednego zbioru do przodu na strumieniu SI

!BSF SO 3 - cofnięcie nośnika przypisanego do strumienia SO o trzy znaczniki końca zbioru do tyłu.

4.4.15. Diagnostyka edytora. Znaczenie dyrektywy NOICE

Komunikaty diagnostyczne emitowane w trakcie pracy przez program EDM można podzielić na dwie zasadnicze grupy:

- komunikaty o wykrytych błędach,
- uwagi i ostrzeżenia.

Standardowe wystąpienie jakiegokolwiek błędu edytor sygnalizuje wypisaniem pojedynczego znaku zapytania "?". Podobnie uwagi i ostrzeżenia pod adresem użytkownika ograniczają się do wypisania jednego lub dwu znaków. Jeżeli użytkownik pragnie, aby edytor emitował komunikaty diagnostyczne w bardziej rozwiniętej postaci wówczas może zmienić normalnie obowiązujący standard przez użycie dyrektywy postaci

```
NOICE [!]
```

bez podawania wykrzyknika po słowie kluczowym. Użycie wersji dyrektywy NOICE z wykrzyknikiem spowoduje powrót do standardowej (to jest skróconej) postaci komunikatów diagnostycznych.

W tablicach 4.2 i 4.3 zamieszczono postać i znaczenie komunikatów diagnostycznych edytora, przy czym w przypadku uwag i ostrzeżeń obok pełnej wersji odpowiadającej dyrektywie NOICE podano wersję standardową (skróconą). Ta druga wersja nie występuje w wykazie komunikatów o błędach, gdyż jak już było powiedziane wyżej jest ona taka sama dla wszystkich błędów i ma postać jednego znaku zapytania.

Tablica 4.2

Wykaz komunikatów o błędach

Lp.	Postać komunikatu	Przyczyna
1	2	3
1	BAD COMMAND	Złe słowo kluczowe dyrektywy
2	CANT NULL EXP.	Nielegalne użycie pustego wzorca
3	EXP. NO DELIMITER	Brak ogranicznika wzorca
4	EXP. OVERFLOW	Przepełnienie wzorca
5	BAD ASTERRIKS	Niedozwolone użycie gwiazdki
6	BAD BACKSLASH	Zły kontekst użycia znaku "\"
7	BAD BRACKET	Złe zagnieżdżenie podwyrażeń
8	BAD CLASS	Zły format definicji klasy znaków we wzorcu ([...])
9	MARK FMT BAD	Parametr dyrektywy MARK nie jest literą
10	NO MARKED LINE	Odwołanie do zamarkowanej linii, która została usunięta lub zmieniona
11	ADDR. RANGE BAD	Zły zakres adresu lub adresów
12	ADDR. FMT BAD	Zły format adresu
13	NO ADDR. LINE	Odwołanie do nieistniejącej linii
14	NUMBER OVERRLOW	Za duża liczba
15	SUBST. SYNTAX BAD	Zła składnia dyrektywy SUBSTITUTE
16	NO MATCH	Dyrektywa SUBSTITUTE nie znalazła ani jednego wystąpienia tekstu określonego podanym wzorcem
17	BRACKET OFF RANGE	Odwołanie do nieistniejącego podwyrażenia
18	MOVE ADDR. CONFLICT	Parametr dyrektywy MOVE lub TRANSCRIBE odnosi się do jednej z przesuwanych linii
19	CANT JOIN AT $	JOIN dla ostatniej linii
20	BAD WINDOW	Zły parametr dyrektywy Z

cd. tablicy 4.2

1	2	3
21	WINDOW RANGE BAD	Błędna definicja rozmiarów okienka
22	POSITIVE NUM. REQUIRED	Niedozwolone wystąpienie liczby ujemnej lub zera
23	BAD SIZE	Nielegalny parametr dyrektywy SIZE lub FORMAT
24	CANT NEST GLOBALS	Dyrektywa GLOBAL występuje w liście innej dyrektywy GLOBAL
25	GLOBAL FMT BAD	Zły format dyrektywy GLOBAL lub format zbioru roboczego
26	FORMAT SYNTAX BAD	Zła składnia dyrektywy FORMAT
27	CANT CMD ON GLOBAL	Dyrektywa nie może występować na liście GLOBAL
28	BAD ARGUMENTS	Zły znak w polu parametrów dyrektywy
29	SOM ERROR	Zła składnia dyrektywy systemu operacyjnego lub inny błąd wykryty przez system

Źródło: /6/.

Tablica 4.3

Wykaz uwag i ostrzeżeń edytora EDM

Lp.	Postać komunikatu		Przyczyna
	pełna	skrócona	
1	LINE OVERFLOW!	O!	Przepełnienie logicznych rozmiarów linii
2	ACCEPTING??	??	Wymagane potwierdzenie od użytkownika
3	PRINT LIMIT!	P?	Przekroczenie ograniczenia na liczbę linii wypisywanych jedną instrukcją
4	FILE NOT SAVED!	T?	Ostrzeżenie przed możliwością zniszczenia poprawianego tekstu
5	=AT FIRST LINE=	!	Próba wyświetlenia poprzedniej linii dyrektywą "-" lub "ESC" gdy bieżącą linią jest pierwsza linia poprawianego tekstu
6	=AT LAST LINE=	!	Analogiczna sytuacja dotycząca ostatniej linii
7	TMP FILE OVERFLOW!	!!	Przepełnienie rozmiarów zbioru roboczego
8	MEMORY OVERFLOW!	M!	Przepełnienie rozmiarów pamięci

Źródło: /6/.

5. PROGRAMOWANIE W JĘZYKU FORTRAN W SYSTEMIE MERA-400

Niniejszy rozdział poświęcony jest prezentacji języka programowania FORTRAN IV-S będącego implementacyjną wersją języka FORTRAN IV opracowaną dla maszyny MERA-400. Opisując składnię i semantykę tego języka szczególną uwagę zwrócono na odstępstwa od zalecanego przez ISO standardu FORTRANu. Odstępstwa te, na które składają się zarówno ograniczenia, jak i rozszerzenia standardu, zostały odrębnie wyspecyfikowane na końcu tego rozdziału.

Na wstępie omówimy pewne reguły notacyjne wykorzystywane przy opisie składni poszczególnych instrukcji języka FORTRAN IV-S. Duże litery, cyfry oraz inne znaki z wyjątkiem nawiasów kwadratowych [ ] i wielokropka ..... reprezentują w zapisie składni same siebie. Słowa zapisywane małymi literami oznaczają zastępowane przez nie obiekty. Rodzaj i postać zastępowanego obiektu są każdorazowo omówione w tekście wyjaśniającym znaczenie danej konstrukcji składniowej. Nawiasy kwadratowe [ ] zawierają pozycje opcjonalne. Zawarty w nich zapis nie musi wystąpić w instrukcji. Wielkropek ..... wskazuje, że poprzedzająca go pozycja lub ograniczona nawiasami grupa pozycji może być powtórzona wielokrotnie. Dla zwiększenia czytelności poszczególne zapisy składni zostały umieszczone w prostokątnych ramkach.

5.1. Wprowadzenie do FORTRANu IV-S

Program w FORTRANie składa się z instrukcji oraz opcjonalnych komentarzy powiązanych w jednostki logiczne zwane segmentami.

Instrukcje dzielą się na dwie podstawowe klasy: instrukcje wykonywalne opisujące akcje programu i instrukcje niewykonywalne opisujące rozmieszczenie i własności danych, ich redakcję oraz stosowane konwersje. Instrukcje zapisywane są rekordami fizycznymi - wierszami złożonymi z maksimum 80 znaków, z czego 72 pierwsze znaki są analizowane przez translator. Instrukcje mogą posiadać wiersze kontynuacji.

Komentarze nie są analizowane przez translator i służą jedynie programiście do dokładniejszego opisania działania programu. Wiersz komentarza jest identyfikowany za pomocą litery C w pierwszej kolumnie i nie może posiadać wiersza kontynuacji. W programie mogą występować tzw. instrukcje testujące (dowolne instrukcje FORTRANu) zaznaczane za pomocą litery D w pierwszej kolumnie wiersza. Wiersze instrukcji testujących (wiersze kontynuacji tych instrukcji również muszą zawierać literę D w pierwszej kolumnie) są analizowane przez translator jedynie wtedy, gdy przy jego wywołaniu zostały podane odpowiednie opcje. W przeciwnym przypadku translator traktuje je jak wiersze komentarza.

Zbiór znaków translatora FORTRANu IV-S obejmuje następujące znaki, które mogą być używane w instrukcjach:

- litery od A do Z,
- cyfry od Ø do 9,
- znaki specjalne:

spacja	= znak równości
+ plus	- minus
* gwiazdka	/ ukośnik
(nawias lewy	) nawias prawy
, przecinek	. kropka dziesiętna
" cudzysłów	' apostrof
< nawias ostry lewy	> nawias ostry prawy

5.1.1. Format wiersza

Wiersz programu w FORTRANie składa się z następujących pól: pola etykiety instrukcji, pola kontynuacji, pola instrukcji i pola identyfikatora. Tylko trzy pierwsze pola są interpretowane przez translator.

Pole etykiety instrukcji zajmuje pięć pierwszych kolumn wiersza. Etykiety jest ciągiem cyfr dziesiętnych (maksymalnie pięciu), które przedstawiają liczbę całkowitą z przedziału $[1, 32767]$. Wszystkie instrukcje, do których występują w programie odwołania, muszą być opatrzone etykietą. Niedopuszczalne jest oznaczenie dwóch różnych instrukcji w obrębie jednego segmentu tą samą etykietą.

Pole kontynuacji obejmuje szóstą kolumnę wiersza. Jeżeli w kolumnie tej znajduje się znak różny od zera lub spracji, to analizowany wiersz jest traktowany przez translator jako wiersz kontynuacji. Instrukcje można dzielić na wiersze w dowolny sposób z tym, że pojedyncza nazwa, stała (z wyjątkiem stałej tekstowej, łańcucha alfanumerycznego i stałej zespolonej) oraz operator nie mogą być kontynuowane w następnym wierszu.

Pole instrukcji zajmuje kolumny od 7 do 72 i zawiera właściwą treść instrukcji (nie licząc występującej ewentualnie etykiety).

Pole identyfikatora (kolumny 73 - 80) może zawierać dowolne znaki. Informacja zawarta w tym polu nie jest interpretowana przez translator.

5.1.2. Struktura segmentów

W FORTRANie istnieją pewne ogólne wymagania odnośnie kolejności występowania poszczególnych rodzajów instrukcji w segmencie. Zamieszczony niżej schemat podaje rozmieszczenie i zakres możliwych przemieszczeń instrukcji w obrębie segmentu. Schemat składa się z prostokątnych obszarów, w które wpisano nazwy lub określenia dotyczące możliwych do wykorzystania w segmencie instrukcji. Kolejność występowania poszczególnych rodzajów instrukcji w segmencie musi być zgodna z kolejnością występowania poszczególnych obszarów w omawianym schemacie (przy przeglą-

daniu schematu w kierunku pionowym z góry na dół), przy czym sąsiadujące ze sobą na jednym poziomie obszary świadczą o możliwości umieszczenia w danym miejscu segmentu instrukcji należącej do jednego lub drugiego obszaru.

<table>
<tr><td rowspan="4">Wiersz komentarza</td><td colspan="2">Instrukcje PROGRAM, FUNCTION, OVERLAY, SUBROUTINE, BLOCK DATA</td></tr>
<tr><td colspan="2">Instrukcje IMPLICIT</td></tr>
<tr><td rowspan="2">Instrukcje specyfikujące (bez INTERNAL i EXTERNAL) oraz instrukcje DEFINE FILE</td><td>Definicja funkcji lokalnej</td></tr>
<tr><td>Instrukcje wykonywalne oraz instrukcje FORMAT, ENTRY, INTERNAL, EXTERNAL</td></tr>
<tr><td colspan="3">Instrukcje END</td></tr>
</table>

Źródło: /4/.

5.2. Elementy instrukcji języka FORTRAN

Podstawowymi składnikami instrukcji w FORTRANie są: stałe, zmienne, tablice, wyrażenia oraz wywołania funkcji. Wiele obiektów jest w programie (segmencie) identyfikowanych poprzez nazwy symboliczne.

Nazwa symboliczna jest dowolnej długości ciągiem liter i cyfr rozpoczynającym się od litery. Translator rozpoznaje jedynie sześć pierwszych znaków nazwy, pozostałe znaki są ignorowane.

Każdy podstawowy składnik może reprezentować dane kilku różnych typów. Typ danych może wynikać bezpośrednio z zapisu składnika (stałe), może być deklarowany explicite lub może być implikowany umownie.

Omówimy teraz dokładniej wymienione na wstępie składniki instrukcji FORTRANu.

5.2.1. S t a ł e

Stała reprezentuje wartość, która nie ulega zmianie w trakcie wykonania programu. Stała może przedstawiać wartość numeryczną, wartość logiczną lub ciąg znaków.

Stała całkowita ma następującą postać:

gdzie nn jest ciągiem cyfr dziesiętnych. Stała ujemna musi być poprzedzona znakiem minus. Stała dodatnia może być opcjonalnie poprzedzona znakiem plus. Wartością stałej całkowitej jest całkowita liczba dziesiętna, której zapis pozycyjny odpowiada ciągowi cyfr nn. Wartość bezwzględna stałej całkowitej nie może być większa od 32767. Stała całkowita zajmuje jedno słowo pamięci. Poprawnie zapisanymi stałymi całkowitymi są przykładowo:

Ø
-265
885
+31

Stała rzeczywista może przyjmować jedną z trzech postaci:

gdzie nn jest ciągiem cyfr dziesiętnych. Kropka dziesiętna może występować w dowolnym miejscu ciągu cyfr. Ilość cyfr nie jest ograniczona, ale tylko 11 cyfr licząc w lewo od pierwszej niezerowej to cyfry znaczące. Określoną w powyższy sposób stałą nazwiemy podstawową stałą rzeczywistą. W ogólnym przypadku stała rzeczywista może występować jako podstawowa stała rzeczywista lub jako stała całkowita czy podstawowa stała rzeczywista, po której następuje wykładnik dziesiętny postaci:

gdzie cc jest stałą całkowitą o wartości bezwzględnej nie większej od 39. Wykładnik określa potęgę liczby dziesięć, przez którą mnożona

jest poprzedzająca ten wykładnik stała. Stała rzeczywista zajmuje trzy słowa pamięci. Wartość bezwzględna stałej rzeczywistej musi należeć do przedziału ($\emptyset.147 \equiv 10^{-38}$, $\emptyset.17\emptyset \equiv 10^{39}$). Przykłady poprawnego zapisu stałych rzeczywistych:

```
5.76
-.333
88754367439.
+21.8E12
13ØE-6
```

Stała podwójnej precyzji jest podstawową stałą rzeczywistą lub stałą całkowitą, po których następuje wykładnik dziesiętny postaci:

```
Dcc
```

gdzie cc jest stałą całkowitą, której wartość bezwzględna nie może przekraczać 39. Ilość cyfr poprzedzających wykładnik nie jest ograniczona, ale tylko pierwsze 23 cyfry licząc w lewo od pierwszej niezerowej są cyframi znaczącymi. Stała podwójnej precyzji zajmuje sześć słów pamięci. Ograniczenia na wartość bezwzględną stałej podwójnej precyzji są identyczne jak w przypadku stałej rzeczywistej. Poprawnie zapisanymi stałymi podwójnej precyzji są przykładowo:

```
123456789Ø123456789Ø123D9
23.74D-29
2D2
```

Stała zespolona jest parą stałych rzeczywistych oddzielonych od siebie przecinkiem i ujętą w nawiasy:

```
(sr,sr)
```

gdzie sr jest stałą rzeczywistą. Pierwsza stała reprezentuje część rzeczywistą, druga stała część urojoną liczby zespolonej. Stała zespolona zajmuje sześć słów pamięci. Przykłady zapisu stałych zespolonych:

```
(12.75,-4.6)
(Ø.,2222E-3)
```

Stała hexadecymalna jest innym sposobem przedstawienia stałych zajmujących jedno, trzy lub sześć słów pamięci. Typ danej związany ze stałą hexadecymalną jest zależny od ilości słów zajmowanych przez tę

stałą: 1 słowo - typ całkowity, 3 słowa - typ rzeczywisty, 6 słów - typ podwójnej precyzji lub zespolony. Postać stałej hexadecymalnej jest następująca:

nZs

gdzie n jest stałą całkowitą bez znaku, $1 \leq n \leq 24$, natomiast s jest ciągiem znaków ze zbioru cyfr szesnastkowych (Ø,1,2,3,4,5,6,7,8,9,A, B,C,D,E,F), przy czym długość ciągu jest równa dokładnie liczbie n. Każdy znak (cyfra szesnastkowa) zajmuje w pamięci cztery bity. Znaki są ładowane kolejno po cztery w słowie, w jedno, trzy lub sześć słów w taki sposób, że ostatni znak ciągu znajduje się na ostatnich czterech bitach ostatniego słowa. Pozostałe części słowa są uzupełniane zerami. Przykładem są następujące stałe:

stała hexadecymalna	binarna reprezentacja słowa
1ZF	0000000000001111
2Z2C	0000000000101100
4ZACED	1010110011101101

Stała logiczna odpowiada logicznej wartości prawdy lub fałszu. Istnieją jedynie dwie stałe logiczne:

.TRUE.	i	.FALSE.

przy czym pierwsza z nich jest wartością prawdy, druga fałszu.

Stała tekstowa ma postać:

$nHc_1c_2...c_n$

gdzie n jest niezerową stałą całkowitą bez znaku, natomiast $c_1c_2...c_n$ jest n-elementowym ciągiem znaków kodu ISO-7. Ilość znaków nie jest ograniczona. Stałe tekstowe są umieszczane w kolejnych bajtach pamięci po jednym znaku w bajcie. Jeżeli ilość znaków jest nieparzysta, prawy bajt ostatniego słowa jest spacjowany. Przykładem stałej tekstowej może być następujący zapis:

11 HALA MA KOTA

Łańcuch alfanumeryczny jest alternatywną postacią stałej tekstowej mającą następującą formę:

"$c_1c_2....c_n$"

gdzie $c_1c_2....c_n$ ma identyczne znaczenie, jak w przypadku stałej tekstowej, przy czym dodatkowo żaden ze znaków ciągu nie może być cudzysłowem. Łańcuch alfanumeryczny jest przechowywany w pamięci zgodnie z regułami podanymi dla stałej tekstowej. Przykład łańcucha alfanumerycznego:

"ALA MA KOTA".

Stała CAN jest innym sposobem przedstawiania stałych zajmujących jedno, trzy lub sześć słów pamięci. Postać zapisu stałej jest następująca:

nCs

gdzie n jest stałą całkowitą bez znaku, $1 \leqslant n \leqslant 18$, natomiast s jest n-elementowym ciągiem znaków należących do tzw. kodu CAN. Kod ten obejmuje litery, cyfry, dwukropek, kropkę, znak dolara i spację. Znaki występujące w stałej CAN ładowane są do pamięci po trzy w jedno słowo (po odpowiednim zakodowaniu). Jeżeli ilość znaków nie jest wielokrotnością trzech, słowo (lub słowa) pamięci uzupełniane jest (są) w taki sposób, jakby ciąg znaków był przedłużony o odpowiednią ilość spacji. Przykłady stałych CAN:

```
11CALA MA KOTA
 4CDATA
```

Łańcuch CAN ma następującą postać:

$c_1c_2....c_n$

gdzie $c_1c_2....c_n$ jest ciągiem znaków należących do kodu CAN. Znaki występujące w łańcuchu CAN ładowane są do pamięci według zasad omówionych dla stałych CAN. Ilość znaków w łańcuchu CAN nie jest ograniczona.

5.2.2. Zmienne

Zmienna jest nazwą symboliczną związaną z ustalonym miejscem w pamięci. Zawartość tego miejsca pamięci jest równocześnie wartością danej zmiennej. Każda zmienna ma określony typ, który może być: całkowity, rzeczywisty, podwójnej precyzji, zespolony lub logiczny.

Typ zmiennej jest ustalany przez deklarację lub przez implikację. Instrukcje deklaracji typu danych (omówione dokładniej w punkcie dotyczącym instrukcji specyfikujących) określają explicite typ danych związany z występującymi w tych instrukcjach zmiennymi. Przez implikację określany jest typ tych zmiennych, dla których nie został on określony explicite. Zmienne te (jeżeli nie występuje instrukcja IMPLICIT) mają typ całkowity, jeżeli ich nazwy rozpoczynają się od liter I,J,K,L,M,N oraz typ rzeczywisty, jeżeli ich nazwy rozpoczynają się od pozostałych liter alfabetu. Dane tekstowe nie mają określonego typu i mogą być przypisane do zmiennych różnych typów. Ilość znaków danej tekstowej uzależniona jest od typu zmiennej, do której jest przypisane: typ całkowity - 2 znaki, typ rzeczywisty - 6 znaków, typ podwójnej precyzji i typ zespolony - 12 znaków.

5.2.3. Tablice

Tablica jest ciągiem kolejnych miejsc pamięci związanych z nazwą symboliczną będącą nazwą tej tablicy. Poszczególne miejsca pamięci, nazywane elementami tablicy, są określone przez wskaźniki występujące po nazwie tablicy. Liczba wskaźników niezbędnych do określenia położenia elementu w obrębie tablicy jest jej wymiarem. Tablica może mieć jeden, dwa lub trzy wymiary.

Wszystkie wartości występujące w tablicy są danymi tego samego typu. Typ danych dla tablicy jest ustalany w taki sam sposób jak dla zmiennych.

Deklarator tablicy specyfikuje nazwę symboliczną, która identyfikuje tablicę wewnątrz segmentu i określa własności tej tablicy. Ma on

następującą postać:

a(d[,d][,d])

gdzie a jest nazwą tablicy, d jest deklaratorem wymiaru. Deklarator wymiaru określa maksymalną wartość, jaką może osiągnąć wskaźnik występujący po nazwie tablicy na pozycji odpowiadającej temu deklaratorowi. Innymi słowy, jest to ilość elementów tablicy dla danego wymiaru.

Tablice są przechowywane w pamięci jako liniowa sekwencja wartości. Tablica jednowymiarowa przechowywana jest w taki sposób, że jej pierwszy element zajmuje pierwsze miejsce, a ostatni ostatnie miejsce w spójnym obszarze pamięci. Elementy tablicy wielowymiarowej przechowywane są w pamięci w kolejności wynikającej z reguły najszybszego wzrostu najbardziej lewego wskaźnika. Przykładowo tablica trójwymiarowa T(3,3,3) ma w następujący sposób rozmieszczone elementy w pamięci: T(1,1,1), T(2,1,1), T(3,1,1), T(1,2,1), T(2,2,1), T(3,2,1), T(1,3,1), T(2,3,1), T(3,3,1), T(1,1,2), T(2,1,2), T(3,1,2), T(1,2,2), T(2,2,2), T(3,2,2), T(1,3,2), T(2,3,2), T(3,3,2), T(1,1,3), T(2,1,3), T(3,1,3), T(1,2,3), T(2,2,3), T(3,2,3), T(1,3,3), T(2,3,3), T(3,3,3).

Wskaźnik jest listą wyrażeń wskaźnikowych ujętą w nawiasy i występuje zawsze po nazwie tablicy, której dotyczy. Wskaźnik ma postać:

(s[,s][,s])

gdzie s jest wyrażeniem wskaźnikowym będącym wyrażeniem arytmetycznym o jednej z następujących postaci:

c*v+k
c*v-k
c*v
v+k
v-k
v
k

gdzie c oraz k są nieujemnymi stałymi całkowitymi, natomiast v jest zmienną całkowitą.

Wskaźnik służy do odwoływania się do określonego elementu tablicy. W każdym odwołaniu do elementu tablicy musi wystąpić jedno wyrażenie wskaźnikowe dla każdego wymiaru, który był deklarowany dla tej tablicy.

5.2.4. Wyrażenia arytmetyczne

Wyrażenie arytmetyczne zbudowane jest z elementów arytmetycznych i operatorów arytmetycznych. Element arytmetyczny może mieć następującą postać:

- stała arytmetyczna,
- zmienna arytmetyczna,
- element tablicy arytmetycznej,
- funkcja arytmetyczna,
- wyrażenie arytmetyczne ujęte w nawiasy ( ).

Określenie arytmetyczny odnosi się do obiektów mających typ całkowity, rzeczywisty, podwójnej precyzji lub zespolony. Stałe CAN oraz hexadecymalne mogą również wystąpić w wyrażeniu arytmetycznym.

Wyrażenie arytmetyczne reprezentuje pojedynczą wartość arytmetyczną, która jest obliczana poprzez wykonanie występujących w wyrażeniu operacji działających na elementach tego wyrażenia. W wyrażeniu arytmetycznym można używać następujących operatorów arytmetycznych:

** potęgowanie,
* mnożenie,
/ dzielenie
+ dodawanie lub plus unarny,
- odejmowanie lub minus unarny.

W wyrażeniu arytmetycznym nie mogą wystąpić bezpośrednio po sobie dwa elementy arytmetyczne. Muszą być one rozdzielone operatorem. Nie mogą także pojawić się bezpośrednio obok siebie dwa operatory.

Przy obliczaniu wartości wyrażenia arytmetycznego obowiązuje ogólna

zasada wykonywania operacji kolejno od lewej strony do prawej, o ile kolejność ta nie jest jednoznacznie ustalona priorytetami operacji i użyciem nawiasów. Nawiasy, podobnie jak w klasycznej notacji matematycznej, służą do wyodrębniania pewnych grup operacji, które mają być wykonane przed innymi operacjami (występującymi na zewnątrz danej pary nawiasów). Poszczególnym rodzajem operacji przypisane są następujące priorytety:

5 - obliczanie wartości funkcji,
4 - potęgowanie,
3 - jednoargumentowa operacja zmiany znaku,
2 - mnożenie i dzielenie,
1 - dodawanie i odejmowanie.

W części wyrażenia arytmetycznego, w której nie występują nawiasy, operacje mające większy priorytet wykonywane są przed operacjami mającymi mniejszy priorytet. Operacje o jednakowych priorytetach wykonywane są w kolejności ich zapisania w wyrażeniu (od lewej do prawej).

Typ i wartość wyrażenia arytmetycznego są określone przez typy i wartości elementów występujących w wyrażeniu oraz rodzaj i kolejność wykonywania operacji. Proste wyrażenie składające się z jednego elementu ma typ i wartość tego elementu. Wyrażenie zawierające wiele elementów przyjmuje wartość i typ wynikający z ostatniej wykonanej operacji podczas obliczania wartości wyrażenia. Typ wyniku operacji zależy od typów elementów, na których wykonywana jest operacja. Jeżeli operacja wykonywana jest na elementach jednakowych typów, to typ wyniku jest taki jak typ tych elementów. Dopuszczalne jest dowolne mieszanie typów argumentów w wyrażeniu artymetycznym z wyjątkiem potęgowania o podstawie typu zespolonowego, kiedy to wykładnik musi być typu całkowitego. Typ wyniku dowolnej operacji jest ustalony przez typ argumentu o większym priorytecie typu. Priorytet typów określony jest następująco:

4 - typ zespolony,
3 - typ podwójnej precyzji,
2 - typ rzeczywisty,
1 - typ całkowity.

5.2.5. Wyrażenia logiczne i wyrażenia relacji

Wyrażenia logiczne zawierają stałe logiczne, zmienne logiczne, elementy tablic typu logicznego, funkcje logiczne oraz wyrażenia relacji połączone operatorami logicznymi. Wyrażenie logiczne przyjmuje wartość prawdy lub fałszu, która jest reprezentowana w pamięci maszyny wartościami -1 i Ø zajmującymi jedno słowo.

Wyrażenie relacji składa się z dwóch wyrażeń arytmetycznych połączonych operatorem relacji. Nie są dozwolone wyrażenia typu zespolonego. Pozostałe typy wyrażeń arytmetycznych mogą być dowolnie ze sobą mieszane. Niżej podana jest postać poszczególnych operatorów relacji oraz ich znaczenie.

.GT.	większy
.GE.	większy lub równy
.LT.	mniejszy
.LE.	mniejszy lub równy
.EQ.	równy
.NE.	różny

Wartość wyrażenia relacji (prawda lub fałsz) wyznaczana jest w ten sposób, że najpierw obliczane są wartości obu wyrażeń arytmetycznych, a następnie dokonywane jest wsprawdzenie, czy spełniona jest odpowiednia relacja określona operatorem. Spełnienie relacji daje wartość prawdy, brak spełnienia - wartość fałszu.

Istnieją następujące operatory logiczne:

.NOT.	Jeżeli x ma wartość prawdy, to .NOT.x ma wartość fałszu. Jeżeli x ma wartość fałszu, to .NOT.x ma wartość prawdy.
.AND.	Jeżeli x oraz y mają wartość prawdy, to x.AND.y ma również wartość prawdy. W pozostałych przypadkach .AND.y ma wartość fałszu.
.OR.	Jeżeli x oraz y mają wartość fałszu, to x.OR.y ma

również wartość fałszu. W pozostałych przypadkach x.OR.y ma wartość prawdy.

Dwa operatory logiczne lub relacji nie mogą występować obok siebie, z wyjątkiem przypadku gdy drugim z nich jest operator .NOT. Dla przykładu wyrażenie:

A.AND..NOT.B.OR.C

jest poprawne i równoważne wyrażeniu:

A.AND.(.NOT.B).OR.C

Operator arytmetyczny nie może występować bezpośrednio po .NOT., co ilustruje następujący przykład:

.NOT.-3.5.GT.X jest wyrażeniem błędnym

.NOT.(-3.5.GT.X) jest wyrażeniem poprawnym

Ogólną zasadą jest wykonywanie operacji logicznych w wyrażeniu logicznym od strony lewej do prawej, przy uwzględnieniu następujących priorytetów w przypadku, gdy kolejność obliczeń nie jest określona nawiasami:

5 - obliczanie wartości funkcji,
4 - operatory relacji,
3 - .NOT.,
2 - .AND.,
1 - .OR..

Na przykład wyrażenie logiczne:

.NOT.Z**2+Y**MA.GT.K-2.OR.PA.AND.X.EQ.Y

będzie obliczane analogicznie jak wyrażenie:

(NOT.((Z**2+Y**MA).GT.(K-2))).OR.(PA.AND.(X.EQ.Y))

5.3. Instrukcje podstawienia

Instrukcje podstawienia ustalają lub zmieniają wartość zmiennej lub elementu tablicy poprzez wyliczenie wyrażenia i przypisanie wartości wynikowej odpowiedniej zmiennej lub elementowi tablicy.

Istnieją następujące typy instrukcji podstawienia: arytmetyczna

instrukcja podstawienia, logiczna instrukcja podstawienia oraz instrukcja ASSIGN.

5.3.1. Arytmetyczna instrukcja podstawienia

Arytmetyczna instrukcja podstawienia ma postać:

```
v=wyr
```

gdzie v jest nazwą zmiennej arytmetycznej, a wyr jest dowolnym wyrażeniem arytmetycznym. Wykonanie tej instrukcji polega na obliczeniu wartości wyrażenia wyr i przypisaniu tej wartości zmiennej v występującej po lewej stronie znaku równości (operatora podstawienia).

Typ wyrażenia arytmetycznego po prawej stronie instrukcji podstawienia może być inny niż typ zmiennej (elementu tablicy) występującej po lewej stronie. W takim przypadku po obliczeniu wartości wyrażenia wynik zostanie przekształcony do typu zgodnego z typem lewej strony instrukcji podstawienia.

Rozszerzeniem arytmetycznej instrukcji podstawienia jest wielokrotna arytmetyczna instrukcja podstawienia mająca postać:

$$v_1=v_2=\ldots=v_n=wyr$$

gdzie $v_1,\ldots,v_n$ są elementami lewej strony, a wyr jest dowolnym wyrażeniem arytmetycznym. Elementem lewej strony może być: nazwa zmiennej arytmetycznej, element tablicy arytmetycznej lub nazwa tablicy arytmetycznej. Ilość elementów lewej strony nie jest ograniczona. Mogą to być elementy różnych rodzajów i różnych typów. Wykonanie zdefiniowanej wyżej wielokrotnej arytmetycznej instrukcji podstawienia daje dokładnie taki sam efekt, jak wykonanie następującej sekwencji instrukcji podstawienia:

$v_n=wyr$
$v_{n-1}=v_n$
$\vdots$
$v_1=v_2$

Jeżeli po lewej stronie instrukcji podstawienia występuje nazwa tablicy, to wartością wyrażenia po prawej stronie zostaną wypełnione wszystkie elementy tej tablicy.

5.3.2. Logiczna instrukcja podstawienia

Logiczna instrukcja podstawienia ma postać:

```
v=wyr
```

gdzie v jest nazwą zmiennej logicznej lub elementem tablicy logicznej, a wyr jest dowolnym wyrażeniem logicznym. Instrukcja ta nadaje obiektowi po lewej stronie instrukcji wartość logiczną prawdy lub fałszu określoną wyrażeniem logicznym występującym po prawej stronie tej instrukcji.

Podobnie jak w przypadku arytmetycznej instrukcji podstawienia, istnieje możliwość wykorzystywania wielokrotnej logicznej instrukcji podstawienia mającej postać:

$v_1=v_2=\ldots=v_n=wyr$

gdzie $v_1,\ldots.v_n$ są elementemi lewej strony, a wyr jest wyrażeniem logicznym. Elementami lewej strony mogą być: zmienna logiczna, element tablicy logicznej lub nazwa tablicy logicznej. Ilość elementów lewej strony nie jest ograniczona.

Wielokrotna logiczna instrukcja podstawienia nadaje wartość wyrażenia logicznego wszystkim elementom lewej strony. Jeżeli elementem lewej strony jest nazwa tablicy, to wartość wyrażenia logicznego jest nadawana wszystkim elementom tablicy.

5.3.3. Instukcja ASSIGN

Instrukcja ASSIGN ma następującą postać:

```
ASSIGN s TO v
```

gdzie s jest etykietą instrukcji wykonywalnej lub instrukcji FORMAT, v jest zmienną całkowitą.

Instrukcja ASSIGN przypisuje wartość (etykietę) do zmiennej w podobny sposób jak w przypadku arytmetycznej instrukcji podstawienia. Zmienna, której przypisana została etykieta jest zdefiniowana tylko dla odwołań do etykiety, nie jest natomiast zdefiniowana jako zmienna całkowita. Powtórne zdefiniowanie jej jako zmiennej całkowitej może nastąpić w wyniku wykonania instrukcji podstawienia nadającej tej zmiennej pewną wartość całkowitą.

Zmienna występująca w instrukcji ASSIGN może być wykorzystana do przekazywania sterowania przez wykonywaną w dalszej kolejności instrukcję GO TO lub też może zostać użyta w instrukcjach WE/WY wskazując odpowiedni format.

5.4. Instrukcje sterujące

Przekazywanie sterowania w obrębie tego samego segmentu lub do innego segmentu, a także przerwanie realizacji programu dokonywane jest przy pomocy instrukcji sterujących. Różne typy instrukcji sterujących pozwalają dokonywać:

- bezwarunkowego przekazania sterowania innym instrukcjom w programie,
- warunkowego przekazania sterowania na podstawie wyników badania pewnego wyrażenia lub zmiennej,
- przekazania sterowania w zależności od ustawionych uprzednio "przełączników" określających jedną lub kilka dróg, którymi może przebiegać program,
- powtarzania sekwencji instrukcji określoną ilość razy,
- zawieszenia lub zatrzymania programu,
- przekazania sterowania innemu segmentowi.

Opisane wyżej czynności są realizowane przez następujące instrukcje:

GO TO, IF, DO, CONTINUE, CALL, RETURN, PAUSE, STOP, END. Instrukcje CALL i RETURN omówione zostaną w punkcie opisującym wykorzystanie podprogramów.

5.4.1. I n s t r u k c j e GO TO

Instrukcja GO TO służy do przekazywania sterowania wewnątrz segmentu, w którym występuje. Istnieją trzy typy tej instrukcji:

- bezwarunkowa instrukcja GO TO,
- przełączana instrukcja GO TO,
- podstawiana instrukcja GO TO.

Zapis GO TO, w którym nie występuje spacja jest równoważny zapisowi GO TO.

Bezwarunkowa instrukcja GO TO ma postać:

```
GO TO k
```

gdzie k jest etykietą instrukcji wykonywalnej. Wynikiem wykonania tej instrukcji GO TO jest kontynuowanie programu począwszy od instrukcji wskazanej przez etykietę k.

Przełączana instrukcja GO TO pozwala na wybór instrukcji, która ma być wykonana jako następna. Postać tej instrukcji GO TO jest następująca:

```
GO TO (k1, k2, ... kn), i
```

gdzie $k_1, k_2, \ldots k_n$ są etykietami lub pseudoetykietami Ø, i jest zmienną całkowitą. Wykonanie instrukcji przełączanej GO TO powoduje przekazanie sterowania do instrukcji opatrzonej etykietą k_j, przy czym j jest wartością zmiennej i w czasie wykonywania instrukcji GO TO. Jeżeli etykieta k_j jest pseudoetykietą Ø, to sterowanie zostanie przekazane następnej instrukcji wykonywalnej występującej w programie po instrukcji GO TO. Wartość j zmiennej i w trakcie wykonywania instrukcji GO TO powinna spełniać warunek $1 \leqslant j \leqslant n$.

Dla przykładu rozważmy instrukcję:

GO TO (11, 12, Ø, 11), KR

po której wykonywana będzie instrukcja następująca po niej w przypadku, gdy KR=3, instrukcja z etykietą 11 w przypadku KR=1 lub KR=4 oraz instrukcja z etykietą 12 dla KR=2.

Podstawiana instrukcja GO TO przekazuje sterowanie do instrukcji, której etykieta jest reprezentowana przez zmienną. Postać instrukcji jest następująca:

GO TO v[,(k_1, k_2, ... k_n)]

gdzie v jest zmienną całkowitą, k_1, k_2, ..., k_n są etykietami instrukcji wykonywalnych lub pseudoetykietami Ø. Zmienna v musi mieć przed wykonaniem instrukcji GO TO podstawioną wartość przez instrukcję ASSIGN (nie przez arytmetyczną instrukcję podstawienia). Instrukcja GO TO i instrukcja ASSIGN muszą występować w tym samym segmencie lub w różnych segmentach w przypadku, gdy zmienna v zlokalizowana jest we wspólnym dla obu segmentów bloku COMMON.

5.4.2. Instrukcja IF

Instrukcja IF służy do warunkowego przekazania sterowania lub warunkowego wykonania instrukcji. Istnieją dwa typy instrukcji IF:

- arytmetyczna instrukcja IF,
- logiczna instrukcja IF.

Arytmetyczna instrukcja IF ma postać:

IF(wyr)e_1,e_2,e_3

gdzie wyr jest wyrażeniem arytmetycznym, natomiast każdy z elementów e_1, e_2, e_3 jest etykietą lub pseudoetykietą Ø. Pseudoetykieta Ø identyfikuje następną instrukcję wykonywalną występującą w programie bezpośrednio po instrukcji IF.

Podczas wykonywania arytmetycznej instrukcji IF obliczana jest

najpierw wartość wyrażenia w nawiasach, a następnie sterowanie zostaje przekazane do jednej z trzech instrukcji identyfikowanych przez etykiety e_1, e_2, e_3. Etykieta e_2 jest wybierana, jeżeli wartość wyrażenia jest równa zeru, etykieta e_1, jeżeli wartość ta jest mniejsza od zera i etykieta e_3, jeżeli wartość wyrażenia jest większa od zera. Etykiety występujące w instrukcji IF nie muszą być różne.

Przykłady arytmetycznych instrukcji IF:

```
IF(SUMA-1ØØ)12,45,9
IF(A**2+B/1Ø-5Ø)1Ø,Ø,1Ø
```

Logiczna instrukcja IF ma następujące dwie postacie:

IF(wyr)st

IF(wyr)e_1,e_2

gdzie wyr jest wyrażeniem logicznym, st jest dowolną instrukcją wykonywalną różną od instrukcji DO i innej logicznej instrukcji IF, e_1 i e_2 są etykietami lub pseudoetykietą Ø. Druga z przedstawionych postaci instrukcji IF nazywana jest logiczną instrukcją IF typu arytmetycznego.

W trakcie wykonywania logicznej instrukcji IF obliczana jest najpierw wartość wyrażenia logicznego wyr. Jeżeli wartością tego wyrażenia jest prawda, to dla pierwszej postaci instrukcji IF następuje wykonanie instrukcji uwarunkowanej st, a dla logicznej instrukcji IF typu arytmetycznego następuje przekazanie sterowania do instrukcji opatrzonej etykietą e_1. Jeżeli wartością wyrażenia jest fałsz, to sterowanie zostaje przekazane do następnej instrukcji wykonywalnej w przypadku pierwszej postaci instrukcji logicznej IF oraz do instrukcji z etykietą tą e_2 w przypadku logicznej instrukcji IF typu arytmetycznego. Pseudoetykieta Ø określa pierwszą instrukcję wykonywalną występującą bezpośrednio po logicznej instrukcji IF. Przykłady logicznych instrukcji IF:

```
IF(K.GT.5)GO TO 34
IF(Z.EQ.Y.AND.8.LT.E+4)R=R+H
IF(A.NE.B)IF(P*EQ*Ø.75+15)3,5,17
IF(X.LE.44)13,12
```

5.4.3. I n s t r u k c j a DO

Instrukcja DO służy do organizowania pętli programowych. Jej postać jest następująca:

```
DO e k=m1,m2 [,m3]
```

gdzie e jest etykietą instrukcji wykonywalnej występującej po instrukcji DO w tym samym segmencie, k jest zmienną całkowitą nazywaną zmienną sterującą pętli DO, m_1, m_2 i m_3 są nazywane parametrami (odpowiednio: początkowym, końcowym i kroku) i mogą być stałymi, zmiennymi lub wyrażeniami arytmetycznymi dowolnego typu. Jeżeli parametr kroku m_3 nie występuje w instrukcji DO, jest ona wykonywana tak, jakby jego wartość była równa 1. Instrukcje wykonywalne następujące po instrukcji DO aż do instrukcji z etykietą e włącznie stanowią zakres instrukcji DO, a instrukcja mająca etykietę e nazywana jest instrukcją graniczną cyklu.

Wykonanie instrukcji DO przebiega w następujący sposób: Najpierw zmiennej sterującej k zostaje przypisana wartość parametru początkowego m_1. W każdym kolejnym kroku wykonywania instrukcji DO aktualna wartość zmiennej sterującej jest porównywana z wartością parametru końcowego m_2 i jeżeli wartość zmiennej sterującej nie jest większa od tego parametru przy dodatnim kroku m_3 lub nie jest od niego mniejsza przy kroku ujemnym, to wykonywane są kolejno instrukcje wchodzące w zakres instrukcji DO, natomiast w przeciwnym przypadku następuje przekazanie sterowania do pierwszej instrukcji wykonywalnej umieszczonej w programie po instrukcji granicznej. Za każdym razem po wykonaniu instrukcji granicznej wartość zmiennej sterującej jest zwiększana o wartość kroku m_3, po czym następuje powtórne jej porównanie z parametrem końcowym m_2 rozpoczynające kolejny krok wykonywania instrukcji DO. Każdprazowo przed użyciem parametru m_1, m_2 lub m_3 obliczana jest jego aktualna wartość, którą stanowi część całkowita wyrażenia m_1, m_2 lub

m_3. Należy brać pod uwagę, że jeżeli pewna instrukcja z zakresu instrukcji DO spowoduje zmianę wartości zmiennej sterującej lub paramertu m_2 czy m_3, to w konsekwencji może ulec zmianie ilość cykli wykonania instrukcji stanowiących zakres pętli DO.

Instrukcją graniczną cyklu nie może być żadna z form instrukcji GO TO, arytmetyczna instrukcja IF, RETURN, STOP, PAUSE, logiczna instrukcja IF typu arytmetycznego lub inna instrukcja DO. Logiczna instrukcja IF typu niearytmetycznego jest dozwolona, jeżeli nie zawiera żadnej z wymienionych instrukcji.

Każda pętla DO może zawierać jedną lub więcej wewnętrznych pętli DO. Zakres wewnętrznej (zagnieżdżonej) pętli DO musi w całości znajdować się wewnątrz zakresu zewnętrznej pętli. Tworzone w ten sposób pętle mogą posiadać tę samą instrukcję graniczną. Przykładem prawidłowo umieszczonych wewnętrznych pętli DO jest następujący fragment programu:

```
      DO 45 K = 1,10
      .
      .
      .
      DO 35 L=2,50,2
      .
      .
      .
35    CONTINUE
      .
      .
      .
      DO 45 M=1,20
      .
      .
      .
45    CONTINUE
```

Podamy teraz pewne uwagi odnoszące się do przekazywania sterowania w pętlach DO. Niedozwolone jest przekazywanie sterowania do zakresu pętli DO z zewnątrz tej pętli. Wykonanie pętli DO może zostać zakończone przed wyczerpaniem pełnej ilości cykli, jaka wynika z wartości parametrów m_1, m_2 i m_3, jeżeli w trakcie wykonywania kolejnego cyklu zostanie wykonana instrukcja sterująca przekazująca sterowanie instrukcji spoza zakresu DO. Przy takim zakończeniu pętli DO zmienna sterująca zachowuje swoją aktualną wartość, natomiast przy normalnym zakończeniu pętli pozostaje ona formalnie niezdefiniowana.

Jeżeli dwie (lub więcej) pętle DO posiadają tę samą instrukcję

graniczną, to sterowanie do instrukcji granicznej może być przekazane jedynie z zakresu pętli najbardziej wewnętrznej.

5.4.4. I n s t r u k c j e CONTINUE, PAUSE, STOP i END

Instrukcja CONTINUE służy jedynie do przekazania sterowania do następnej instrukcji wykonywalnej. Ma ona postać:

```
CONTINUE
```

Instrukcja PAUSE powoduje czasowe zawieszenie programu w celu umożliwienia operatorowi wykonania pożądanych czynności przy pomocy zadania komunikacji. Instrukcja ta ma następującą postać:

```
PAUSE [n]
```

gdzie n jest stałą całkowitą bez znaku lub stałą tekstową.

Wykonanie instrukcji PAUSE polega na wyprowadzeniu na globalny strumień CO informacji podanej w tej instrukcji (liczba całkowita lub stała tekstowa - 40 pierwszych znaków albo ciąg znaków do pierwszego przecinka), następnie zawieszeniu wykonania programu. Operator może wznowić wykonanie programu odpowiednią dyrektywą zadania komunikacji. Jako pierwsza zostanie wówczas wykonana instrukcja wykonywalna następująca bezpośrednio po instrukcji PAUSE.

Instrukcja STOP jest używana do zakończenia wykonywania programu i ma następującą postać:

```
STOP [n]
```

gdzie n jest stałą całkowitą bez znaku lub stałą tekstową.

W wyniku wykonania instrukcji STOP następuje przekazanie sterowania systemowi operacyjnemu i wyprowadzenie informacji podobnie, jak w przypadku omówionej wcześniej instrukcji PAUSE.

Instrukcja END służy do oznaczenia końca segmentu i ma postać:

```
END
```

Instrukcja END musi występować zawsze w ostatnim wierszu każdego segmentu i nie może być opatrzona etykietą.

5.4.5. I n s t r u k c j a ASSEMBLER

Kompilator FORTRANu generuje w wyniku swego działania program w języku symbolicznym MACROASSEMBLER. Instrukcja ASSEMBLER mająca postać:

ASSEMBLER

pozwala na umieszczenie instrukcji napisanych w języku symbolicznym w ciągu instrukcji programu fortranowskiego.

Po instrukcji ASSEMBLER następuje dowolna liczba wierszy zakodowanych w języku symbolicznym. Sekwencję tę kończy wiersz mający znaki gwiazdki w dwóch początkowych kolumnach.

We wstawkach w języku symbolicznym można używać odwołań do zmiennych, tablic i etykiet zdefiniowanych za pomocą normalnych instrukcji FORTRANu. Normalne instrukcje FORTRANu nie mogą się jednak odwoływać do żadnych nazw i etykiet zdefiniowanych w części kodowanej symbolicznie.

Translator po napotkaniu instrukcji ASSEMBLER umieszcza w generowanym programie wszystkie wiersze występujące po tej instrukcji aż do wiersza rozpoczynającego się dwiema gwiazdkami.

5.5. Instrukcje wejścia/wyjścia

Przesyłanie danych pomiędzy pamięcią operacyjną a urządzeniami zewnętrznymi odbywa się przy pomocy instrukcji READ i WRITE. Przesłania te mogą dotyczyć informacji nieredagowanej (binarnej) lub redagowanej (znakowej). Ponadto istnieją jeszcze dwie instrukcje INREC i OUTREC realizujące przesłania sekwencyjne pojedynczych rekordów redagowanych.

Oprócz instrukcji wymienionych wyżej istnieją pomocnicze instrukcje WE/WY, które nie transmitują żadnej informacji. Są to instrukcje: BACKSPACE, REWIND, ENDFILE, DEFINE FILE oraz FIND.

Instrukcjami DECODE i ENCODE można realizować przesłania i konwersję danych wewnątrz pamięci operacyjnej.

Instrukcje WE/WY odwołują się do strumienia logicznego, z którego lub do którego transmitowane są dane. Strumień logiczny powinien być dołączony do fizycznego urządzenia WE/WY, strumienia WE/WY lub zbioru o dostępie bezpośrednim. W tym ostatnim przypadku zbiór o dostępie bezpośrednim musi być zdefiniowany przy pomocy instrukcji DEFINE FILE. Odwołanie do strumienia logicznego następuje poprzez jego identyfikator, który dalej oznaczać będziemy przez u. Identyfikatorem strumienia logicznego u może być:

- stała całkowita bez znaku z zakresu od 1 do 999,
- stała CAN typu całkowitego identyfikująca explicite strumień,
- zmienna typu całkowitego, której nadano wartość jednej z powyższych stałych.

Redagowane instrukcje WE/WY zawierają odwołanie do specyfikacji wzorca, według którego dokonywana jest konwersja danych. Odwołanie to, oznaczane dalej przez f, może być:

- stałą całkowitą Ø,
- etykietą instrukcji FORMAT,
- zmienną całkowitą, której przypisano instrukcją ASSIGN etykietę instrukcji FORMAT,
- nazwą tablicy wypełnionej tekstem, który będzie interpretowany jako specyfikacja wzorca.

W trakcie wykonywania instrukcji WE/WY mogą zostać wykryte błędy. Jeżeli w instrukcji występował indykator obsługi błędu (mający postać ERR=e, gdzie e jest etykietą instrukcji wykonywalnej), to wykrycie błędu spowoduje przekazanie sterowania instrukcji identyfikowanej przez indykator obsługi błędu. Jeżeli taki indykator nie występował w instrukcji, to w przypadku błędu sterowanie zostanie przekazane standardowej procedurze obsługi błędu, która spowoduje sygnalizację błędu i uzależni kontynuację zadania od ustawionych opcji.

5.5.1. Listy wejścia/wyjścia

Listy WE/WY specyfikują dane i określają kolejność, w jakiej mają być one przetransmitowane przy pomocy instrukcji zawierającej daną listę WE/WY.

Prosta lista WE/WY może zawierać zmienną, element tablicy, tablicę albo ciąg tych elementów oddzielonych od siebie przecinkami. Każdy pojedynczy element listy lub ich grupę można ująć w nawiasy. W procesie transmisji biorą udział dane związane z elementami listy rozpatrywanymi w kolejności od lewej do prawej. Jeżeli na liście pojawi się nazwa tablicy (bez wskaźników) to transmitowana jest cała tablica, a kolejność jej elementów określa omówiona wcześniej zasada lokalizowania elementów tablicy w pamięci.

Lista ukrytego cyklu DO ma postać:

(plis,k=m_1,m_2[,m_3])

gdzie plis jest dowolną prostą listą WE/WY, natomiast k, m_1, m_2 i m_3 są zdefiniowane tak samo, jak dla instrukcji DO z tym, że parametry te nie mogą być wyrażeniami arytmetycznymi, a parametr m_3 musi mieć wartość różną od Ø.

Lista ukrytego cyklu DO jest traktowana przez translator jak pojedynczy element, co pozwala umieszczać listę ukrytego cyklu DO na innej liście ukrytego cyklu DO.

Realizacja ukrytego cyklu DO przebiega w ten sposób, że dane są transmitowane do lub z elementów listy wielokrotnie, zgodnie ze specyfikacją ukrytego cyklu DO. Dla przykładu lista ukrytego cyklu DO:

(X,Y,Z,K=1,4)

jest równoważna liście prostej:

X,Y,Z,X,Y,Z,X,Y,Z,X,Y,Z

Jeżeli lista ukrytego cyklu DO zawiera inną listę ukrytego cyklu DO, to dla każdej iteracji zewnętrznej pętli wykonywane są wszystkie iteracje pętli wewnętrznej.

Dowolna lista WE/WY może zawierać pojedynczy element, ciąg elementów lub listę ukrytego cyklu DO.

5.5.2. Nieredagowane sekwencyjne wejście/wyjście

Instrukcje nieredagowanego WE/WY służą do transmisji danych binarnych z pominięciem konwersji. Nieredagowane sekwencyjne instrukcje READ i WRITE mają postać:

```
READ(u [,ERR=s] [,END=t] ) [lista]
WRITE(u [,ERR=s] [,END=t] ) [lista]
```

gdzie u jest identyfikatorem strumienia logicznego, lista jest listą WE/WY, a s,t są etykietami instrukcji wykonywalnych lub pseudoetykietami Ø identyfikującymi następną instrukcję wykonywalną.

Omawiana instrukcja READ czyta ze wskazanego strumienia logicznego dokładnie jeden binarny rekord logiczny i podstawia uzyskane dane do kolejnych elementów listy WE/WY. Niewykorzystana część rekordu jest pomijana. Instrukcja READ bez listy WE/WY powoduje zignorowanie (przeskoczenie) jednego rekordu w zbiorze skojarzonym ze strumieniem u.

Instrukcja WRITE wyprowadza na wskazany strumień logiczny wartości wszystkich elementów listy WE/WY w postaci jednego binarnego rekordu logicznego. Jeżeli instrukcja WRITE nie zawiera listy, to wyprowadzany jest jeden pusty rekord.

Występujący w obu instrukcjach indykator końca zbioru mający postać END=t powoduje przekazanie sterowania instrukcji z etykietą t, jeżeli w trakcie odczytu lub zapisu danych napotkany został znacznik końca zbioru (nośnika). Jeżeli instrukcja nie zawiera indykatora końca zbioru, sterowanie przekazywane jest standardowej procedurze obsługi błędów.

Przykłady:

```
READ(2),X,Y,Z
```

```
READ(2CBI,END=5Ø)R,(A(K),K=7,8Ø)
WRITE(1,ERR=42)NUM,KR
WRITE(4Ø)
```

5.5.3. Redagowane sekwencyjne wejście/wyjście

Redagowane instrukcje WE/WY służą do transmisji danych w postaci znakowej i mają następującą postać:

READ(u,f [,ERR=s] [,END=t]) [lista]
WRITE(u,f [,ERR=s] [,END=t]) [lista]
INREC(u,f [,ERR=s] [,END=t]) [lista]
OUTREC(u,f [,ERR=s] [,END=t]) [lista]

gdzie f jest odwołaniem do specyfikacji wzorca, a pozostałe oznaczenia są identyczne jak w punkcie 5.5.2.

Instrukcja READ wczytuje rekordy znakowe ze wskazanego strumienia logicznego, przekształca dane z postaci znakowej na binarną zgodnie ze skojarzoną z instrukcją specyfikacją wzorca i podstawia uzyskane wartości do kolejnych elementów listy. Instrukcja WRITE pobiera wartości kolejnych elementów listy, przeprowadza konwersję z postaci binarnej na znakową i wyprowadza logiczne rekordy znakowe na wskazany strumień. Poza tym wykonanie omawianych instrukcji przebiega podobnie, jak w przypadku nieredagowanych instrukcji WE/WY.

Instrukcja INREC działa podobnie do instrukcji READ z tym, że transmituje tylko jeden rekord znakowy i jeżeli do strumienia u dołączona jest klawiatura alfanumeryczna, to przed operacją czytania nie jest dokonywana zmiana linii oraz pojedynczy znak powrotu karetki kończy rekord. Daje to możliwość wypisania przez operatora treści rekordu znakowego w tej samej linii co wyprowadzony uprzednio tekst.

Instrukcja OUTREC wykonywana jest identycznie jak instrukcja redagowana WRITE z tą różnicą, że transmituje tylko jeden rekord.

5.5.4. Instrukcje wejścia/wyjścia bezpośredniego dostępu

Przedstawione niżej instrukcje służą do transmisji danych pomiędzy pamięcią a zbiorami o dostępie bezpośrednim. Zbiory te muszą być wyspecyfikowane przy pomocy instrukcji DEFINE FILE. Instrukcje WE/WY bezpośredniego dostępu mają postać:

```
READ(u'r [,ERR=s] [,END=t ] ) [lista]
WRITE(u'r [,ERR=s] [,END=t ] ) [lista]
READ(u'r,f [,ERR=s] [,END=t ] ) [lista]
WRITE(u'r,f [,ERR=s ] [,END=t ] ) [lista]
```

gdzie r jest wyrażeniem typu całkowitego, które specyfikuje numer transmitowanego rekordu logicznego. Pozostałe oznaczenia są identyczne jak w przypadku sekwencyjnych instrukcji WE/WY. Pierwsze dwie instrukcje są nieredagowanymi instrukcjami WE/WY, dwie pozostałe są redagowanymi instrukcjami WE/WY.

Wykonanie instrukcji WE/WY bezpośredniego dostępu przebiega podobnie jak wykonanie omówionych uprzednio instrukcji sekwencyjnego wprowadzania i wyprowadzania informacji z tą różnicą, że praca ze zbiorem bezpośredniego dostępu wymaga podawania numeru rekordu biorącego udział w transmisji. Numer rekordu podawany w instrukcji nie może być mniejszy od 1.

5.5.5. Instrukcje DECODE i ENCODE

Opisane niżej instrukcje dokonują konwersji danych z postaci binarnej na znakową i odwrotnie bez przeprowadzania transmisji z urządzeniami zewnętrznymi. Postać instrukcji jest następująca:

```
DECODE(c,f,b [,ERR=s ] ) [lista]
ENCODE(c,f,b [,ERR=s ] ) [lista]
```

gdzie b jest nazwą tablicy stanowiącej bufor, a c jest wyrażeniem ty-

pu całkowitego określającym długość bufora w znakach. Pozostałe oznaczenia są takie same, jak w redagowanych instrukcjach WE/WY.

Wykonanie instrukcji DECODE przebiega podobnie jak wykonanie instrukcji redagowanej READ, z tym że odczytywana informacja nie jest pobierana z urządzenia zewnętrznego, ale z tablicy będącej buforem. Analogicznie działa instrukcja ENCODE (odpowiadająca instrukcji WRITE) przesyłając dane z listy WE/WY do tablicy - bufora.

Działanie opisanych instrukcji ilustruje przykład:

```
   DIMENSION A(3), K(3)
   DATA A/" 345 12345    1Ø"/
   DECODE(18,1ØØ,A)K
1ØØFORMAT(3I6)
```

Po wykonaniu tego fragmentu programu K(1)=345, K(2)=12345, K(3)=1Ø.

5.5.6. Pomocnicze instrukcje wejścia/wyjścia

Instrukcje mające postać:

```
REWIND u
BACKSPACE u
ENDFILE u
```

gdzie u jest identyfikatorem strumienia logicznego działają na zbiorach sekwencyjnych i powodują odpowiednio: REWIND - cofnięcie do początku zbioru, BACKSPACE - cofnięcie zbioru o jeden rekord logiczny, ENDFILE - zapisanie w zbiorze znacznika końca.

Instrukcja DEFINE FILE nie jest instrukcją wykonywalną i służy do definiowania zbiorów bezpośredniego dostępu. Ma ona postać:

```
DEFINE FILE u(m,n,t,v)  ,u(m,n,t,v)  ....
```

gdzie u jest identyfikatorem strumienia logicznego, m jest liczbą rekordów w zbiorze (stała całkowita z zakresu 1 - 32767), n jest długością rekordu (stała całkowita z zakresu 1 - 32767), v jest zmienną stowarzyszoną ze zbiorem (zmienna całkowita), t jest pojedynczą literą

określającą typ zbioru: F - zbiór rekordów redagowanych (długość rekordu podawana w bajtach), A - zbiór rekordów redagowanych lub nieredagowanych (długość rekordu podawana w bajtach), U - zbiór rekordów nieredagowanych (długość rekordu podawana w słowach).

Po zakończeniu wykonywania danej instrukcji WE/WY, zmienna stowarzyszona zawiera numer rekordu następnego za ostatnio przetransmitowanym.

W odniesieniu do zbiorów o dostępie bezpośrednim można wykorzystywać następującą instrukcję:

```
FIND(u'r)
```

gdzie u jest identyfikatorem strumienia logicznego, a r jest numerem rekordu logicznego (wyrażenie typu całkowitego). Instrukcja FIND realizuje wstępną fazę przetwarzania rekordu polegającą na odszukaniu odpowiedniego rekordu fizycznego w zbiorze i wczytaniu go do bufora w pamięci. Pozwala to na przyspieszenie wykonania kolejnej instrukcji WE/WY.

5.6. Instrukcja FORMAT

Instrukcja FORMAT opisuje wzorzec, według którego dokonywana jest konwersja danych w czasie realizacji redagowanych instrukcji WE/WY. Instrukcja FORMAT jest niewykonywalna i ma następującą postać:

```
FORMAT(q1f1s1f2s2....fnsnqn)
```

gdzie f_i jest opisem pola albo grupą opisów pól ujętą w nawiasy, s_i jest separatorem pól, q_1 i q_n jest ciągiem ukośników lub jest puste.

Lista opisów i separatorów pól jest nazywana specyfikacją wzorca. Opis pola w specyfikacji wzorca ma jedną z następujących postaci:

```
[r] kw.d   [r] kw   .nk   kn   k
```

gdzie k jest jednoliterowym kodem opisu pola, w jest długością pola w znakach, d jest ilością znaków części ułamkowej liczby, r jest licz-

nikiem powtórzeń, który oznacza, że opis pola określa r kolejnych jednakowych pól, n specyfikuje liczbę znaków lub pozycję znaku w rekordzie. Parametry r,w,d,n są stałymi całkowitymi z zakresu: dla r oraz w 1 - 255, dla d Ø - 255, dla n zakres zależy od typu opisu pola.

Separatorem pól może być przecinek lub ukośnik. Ten ostatni ma dodatkową funkcję: inicjuje nowy rekord logiczny.

Dowolny z kodów opisu pól F, E, D, G może być poprzedzony współczynnikiem skali [±] nP gdzie n jest stałą całkowitą z zakresu Ø - 255. Współczynnik skali określa przesunięcie kropki dziesiętnej w prawo (+) lub w lewo (-).

5.6.1. O p i s y p ó l

Omówimy teraz opisy pól, które mogą występować w specyfikacji wzorca.

Opis pola [Iw] służy do konwersji danych całkowitych. Odpowiadający mu element listy WE/WY musi być typu całkowitego. Dla instrukcji wejścia następuje pobranie dokładnie w znaków kolejnych z wczytywanego rekordu i po dokonaniu konwersji na postać binarną umieszczenie tak uzyskanej wartości w odpowiednim elemencie listy WE/WY. Pobrany ciąg znaków musi mieć postać stałej całkowitej. Dla instrukcji wyjścia następuje zamiana wartości elementu listy WE/WY na ciąg długości w znaków i umieszczenie tego ciągu w odpowiednim polu rekordu wyjściowego. Utworzone pole ma postać: spacje wiodące, znak minus (jeżeli liczba jest ujemna), ciąg cyfr wyprowadzanej liczby. Jeżeli pole jest zbyt krótkie, aby pomieścić daną, to pierwszym znakiem pola jest # (gdy dana była dodatnia) lub = (gdy była ujemna), po czym występuje w-1 mniej znaczących cyfr liczby.

Opis pola [Lw] służy do konwersji danych logicznych. Jeżeli wśród w kolejnych znaków wczytywanego rekordu pierwszym różnym od spacji znakiem jest litera T, to do odpowiedniego elementu listy WE/WY podstawiana jest wartość prawdy. W przeciwnym przypadku podstawiany jest fałsz. Na wyjściu opis pola Lw powoduje wyprowadzenie w-1 spacji oraz litery T

(jeżeli zmienna ma wartość prawdy) lub litery F (jeżeli zmienna ma wartość fałszu).

Opis pola Fw.d związany jest z elementami listy WE/WY mającymi typ rzeczywisty, podwójnej precyzji lub zespolony. Na wejściu kolejnych w znaków zostaje przekształconych do postaci binarnej odpowiedniego typu (zgodnego z typem elementu listy WE/WY) i umieszczonych w korespondującym elemencie listy wejściowej. Pobrany z rekordu ciąg znaków musi mieć postać stałej całkowitej, rzeczywistej lub podwójnej precyzji. Jeżeli pole wejściowe nie zawiera kropki dziesiętnej, to d znaków z prawej strony pola stanowi część ułamkową liczby. Na wyjściu zamieniana jest wartość danego elementu listy WE/WY na ciąg w znaków: spacje wiodące (o ile są konieczne), znak minus (jeżeli liczba jest ujemna), cyfry części całkowitej, kropka dziesiętna, d cyfr części ułamkowej. Parametry w i d muszą spełniać relację $w \geqslant d+2$. Jeżeli pole jest zbyt krótkie, aby pomieścić część całkowitą, to następuje podobna reakcja jak w przypadku opisu pola Iw.

Opis pola Ew.d działa na wejściu identycznie jak opis Fw.d, natomiast na wyjściu tworzone jest pole mające postać: spacje wiodące (o ile są konieczne), znak minus (jeżeli liczba jest ujemna), cyfra Ø (o ile się mieści), kropka dziesiętna, d cyfr części ułamkowej, wykładnik o postaci E±nn, gdzie n jest cyfrą dziesiętną. Parametry w i d muszą spełniać relację $w \geqslant d+6$.

Opis pola Dw.d działa identycznie jak opis pola Ew.d, z tym, że wykładnik na wyjściu zawiera literę D zamiast litery E.

Opis pola Gw.d służy do konwersji danych rzeczywistych, podwójnej precyzji, zespolonych, całkowitych i logicznych. O wyborze konwersji decyduje typ elementu listy WE/WY. Opis pola Gw.d działa na wejściu dla danych rzeczywistych, podwójnej precyzji i zespolonych jak opis Fw.d, dla danych całkowitych jak opis Iw, dla danych logicznych jak opis Lw. Działanie opisu Gw.d na wyjściu dla danych całkowitych i logicznych nie różni się od działania opisów Iw i Lw, natomiast dla pozos-

tałych typów danych jest równoważne działaniu opisów pól Fw.d i Ew.d, które są wybierane na podstawie wartości wyprowadzanej liczby.

Opis pola Zw przeznaczony jest do wprowadzania lub wyprowadzania danych dowolnego typu zapisanych przy pomocy cyfr hexadecymalnych. Na wejściu kolejnych w znaków powinno zawierać ciąg cyfr hexadecymalnych reprezentujących wczytywaną wartość. Na wyjściu wartość odpowiedniego elementu listy WE/WY wyprowadzana jest w postaci: w-m spacji wiodących, dokładnie m cyfr hexadecymalnych odpowiadających wyprowadzanej wartości.

Opis pola Aw służy do transmisji danych tekstowych. Na wejściu pobieranych jest z rekordu kolejnych w znaków i umieszczonych w elemencie listy WE/WY. Ilość znaków m umieszczonych w jednym elemencie zależy od jego typu i wynosi: 2 dla typu całkowitego i logicznego, 6 dla typu rzeczywistego, 12 dla typu podwójnej precyzji i zespolonego. Jeżeli $w > m$, to tylko m ostatnich znaków będzie umieszczonych w elemencie listy WE/WY. Jeżeli $w < m$, to wczytane znaki uzupełnione zostaną spacjami. Na wyjściu opis Aw powoduje wyprowadzenie w pierwszym znaków z elementu listy WE/WY, przy czym jeżeli $m < w$, to pojawi się w-m spacji wiodących.

Opis pola Cw służy do transmisji danych tekstowych zapisanych w pamięci w kodzie CAN 1 działa podobnie jak opis pola Aw. Różnica dotyczy ilości znaków m umieszczanych w jednym elemencie listy WE/WY. Ilości te wynoszą: 3 dla typu całkowitego i logicznego, 9 dla typu rzeczywistego, 18 dla typu podwójnej precyzji i zespolonego.

Omówimy teraz opisy pól, które nie są związane z żadnym elementem listy WE/WY.

Opis pola H ma postać identyczną z postacią stałej $nHc_1c_2....c_n$. Na wyjściu n znaków w opisie pola po literze H zostaje umieszczonych w rekordzie wyjściowym. W analogiczny sposób na wejściu n znaków z rekordu wejściowego zostanie umieszczonych w opisie pola po literze H. Opis pola H może być zastąpiony równoważnym mu łańcuchem alfanumerycznym.

Opis pola nX powoduje opuszczenie n znaków w rekordzie wejściowym i umieszczenie n spacji w rekordzie wyjściowym.

Opis pola | Tn | zwany tabulatorem powoduje, że występujący po nim opis pola odnosić się będzie w rekordzie wejściowym lub wyjściowym do ciągu znaków rozpoczynającego się na pozycji n. Dla przykładu instrukcje:

```
      WRITE(1,4)
    4 FORMAT(T8,"KOTA",T1,"ALA MA")
```

wyślą na strumień logiczny 1 następujący rekord: ALA MA KOTA.

Jako parametrów opisów pól (z wyjątkiem opisu H) można używać zmiennych typu całkowitego ujętych w ostre nawiasy < > . Przykładowo opis pola Fw.d może mieć postać ⟨I⟩ F ⟨J⟩ .⟨K⟩ .

5.6.2. Zewnętrzne separatory pól i konwersja swobodna

Jeżeli pobierany przez instrukcję wprowadzania redagowanego ciąg znaków zawiera przecinek, to wczytanych zostanie jedynie tyle znaków, ile zawartych jest od początku pola do pozycji przecinka. Przecinek, zwany zewnętrznym separatorem pól, jest interpretowany jako ogranicznik tylko dla opisów z kodami I, L, F, E, D, G, Z i C. Przecinek tylko wtedy jest ogranicznikiem pola, kiedy ogranicza ciąg krótszy niż w znaków (w jest długością pola).

Istnieje możliwość wykorzystywania tzw. swobodnej konwersji danych polegającej na tym, że dane są wprowadzane lub wyprowadzane na odpowiednie strumienie logiczne w postaci znakowej, ale nie pod kontrolą specyfikacji wzorca. Instrukcje READ i WRITE będą realizować swobodną konwersję danych, jeżeli w miejscu odwołania do specyfikacji wzorca wystąpi pseudoetykieta Ø. Przykładem mogą być instrukcje:

```
READ(1,Ø)A,B,C
WRITE(5,Ø)(X(K),K=4,L)
```

Za pomocą konwersji swobodnej można transmitować tylko dane numeryczne. Na wejściu dane powinny być oddzielone od siebie przecinkami.

Na wyjściu dane przedstawiane są z maksymalną dla danego typu dokładnością i również oddzielane od siebie przecinkami.

5.7. Instrukcje specyfikujące

Instrukcje specyfikujące nie są wykonywalne i zawierają informacje niezbędne dla właściwego przydzielenia pamięci i inicjacji zmiennych i tablic oraz określają inne własności nazw występujących w programie.

5.7.1. Deklaracje typu i rozmiaru tablic

Instrukcje deklaracji typu określają explicite typy specyfikowanych nazw i mają następującą postać:

```
typ v [,v] ....
```

gdzie typ jest jedną z nazw typów: LOGICAL, INTEGER, REAL, DOUBLE PRECISION, COMPLEX, natomiast v jest nazwą symboliczną zmiennej, tablicy, funkcji lub deklaracją tablicy. Deklaracja typu przysłania typ implikowany przez pierwszą literę nazwy.

Instrukcja IMPLICIT umożliwia zmianę implikacji typu nazw symbolicznych poprzez związanie pewnych liter z określonymi typami danych. Postać tej instrukcji jest następująca:

```
IMPLICIT typ (a [,a] ....) [,typ (a [,a] .....)]....
```

gdzie typ jest jedną z nazw: LOGICAL, INTEGER, REAL, DOUBLE PRECISION lub COMPLEX, natomiast a jest pojedynczą literą lub przedziałem liter postaci $l_1 - l_2$, gdzie l_1 i l_2 są literami.

Instrukcja DIMENSION określa liczbę wymiarów tablicy i ilość elementów każdego wymiaru. Postać instrukcji jest następująca:

```
DIMENSION dek ,dek ....
```

gdzie dek jest deklaratorem tablicy opisanym w punkcie 5.2.3. Wymiary mogą być definiowane w instrukcji deklaracji typu. Jeżeli tablica zos-

tała zdefiniowana, to nie może być definiowana powtórnie przez żadną inną instrukcję.

5.7.2. I n s t r u k c j e COMMON i EQUIVALENCE

Instrukcja COMMON definiuje jeden lub kilka spójnych bloków obszaru wspólnego pamięci. Każdy blok jest identyfikowany przez nazwę symboliczną lub jest tzw. blokiem nieetykietowanym. Dane zawarte w bloku obszaru wspólnego mogą być dostępne z każdego segmentu, w którym występuje instrukcja COMMON opisująca blok o tej samej nazwie lub nieetykietowany. Instrukcja COMMON ma następującą postać:

```
COMMON /cb/nlista [ /cb/nlista ] ....
```

gdzie cb jest nazwą bloku lub jest puste (oznacza wtedy blok nieetykietowany),nlista jest niepustą listą nazw zmiennych, nazw tablic, deklaracji tablic, które to elementy są oddzielone przecinkami. Elementom listy w bloku obszaru wspólnego o tej samej nazwie jest przydzielana pamięć w kolejności ich występowania na liście, w każdym segmencie od początku bloku.

Instrukcja EQUIVALENCE przydziela dwóm lub więcej obiektom występującym w jednym segmencie ten sam obszar pamięci. Ma ona postać:

```
EQUIVALENCE (nlista) [,(nlista)] ....
```

gdzie nlista jest listą zmiennych, tablic, elementów tablic, oddzielonych od siebie przecinkami. Obiekty występujące w tej instrukcji na pojedynczej ujętej w nawiasy liście są umieszczane w pamięci poczynając od tego samego miejsca (niezależnie jaki typ danych reprezentują). Jeżeli element tablicy został przy pomocy instrukcji EQUIVALENCE uczyniony równoważnym elementowi innej tablicy, to narzucona zostaje równoważność pozostałych odpowiadających sobie elementów obu tablic. Nie można przypisywać tego samego miejsca w pamięci dwóm różnym elementom tej samej tablicy.

5.7.3. Instrukcje EXTERNAL i INTERNAL

Instrukcja EXTERNAL ma następującą postać:

```
EXTERNAL v [,v] ....
```

gdzie v jest nazwą podprogramu lub etykietą zdefiniowaną poza danym segmentem. Wszystkie nazwy podprogramów zewnętrznych, które są używane w segmencie jako argumenty innych podprogramów zewnętrznych, muszą być zadeklarowane w tym segmencie w instrukcji EXTERNAL. Jeżeli w instrukcji EXTERNAL występują etykiety zdefiniowane w segmencie różnym od danego (tzw. etykiety zewnętrzne), to możliwe jest przekazywanie sterowania z jednego segmentu do drugiego przy pomocy instrukcji GO TO lub IF. Etykiety, do których przekazywane jest sterowanie z innych segmentów, muszą być w segmencie przyjmującym sterowanie umieszczone w instrukcji INTERNAL mającej postać:

```
INTERNAL s [,s] ....
```

gdzie s jest etykietą występującą w danym segmencie. Przykład:

```
SUBROUTINE A                    SUBROUTINE B
EXTERNAL 15                     INTERNAL 15
   :                               :
GO TO 15                        15 CONTINUE
   :                               :
```

5.7.4. Instrukcja DATA

Instrukcja DATA umożliwia przypisanie wartości początkowych zmiennym i elementom tablic przed wykonaniem programu. Ma ona postać:

```
DATA nlista/clista/ [,nlista/clista/] ....
```

gdzie nlista jest listą nazw zmiennych, nazw tablic lub elementów tablic, oddzielonych od siebie przecinkami (wyrażenia wskaźnikowe muszą być stałymi całkowitymi), natomiast clista jest listą stałych oddzielonych przecinkami. Każda stała na liście clista może być poprzedzona gwiazdką i stałą całkowitą bez znaku, która jest interpretowana

jako krotność powtórzeń danego elementu listy. Instrukcja DATA przypisuje wartości kolejnych stałych z clista kolejnym elementom nlista. Wystąpienie w nlista nazwy tablicy (bez indeksów) jest traktowane jak wypisanie wszystkich jej elementów. Jako element nlista może wystąpić konstrukcja z ukrytym DO równoważna liście elementów tablicy podanych w kolejności wynikającej z realizacji ukrytego cyklu DO. Opisywana konstrukcja w najbardziej rozwiniętym przypadku ma postać:

$$(((a(i,j,k),i=a_1,b_1[,c_1]),j=a_2,b_2\ [,c_2]\ ),k=a_3,b_3\ [,c_3])$$

gdzie a jest nazwą tablicy, i,j,k są nazwami zmiennych sterujących ukrytego cyklu DO, a_1,a_2,a_3 (stałe całkowite dodatnie) są parametrami początkowymi zmiennych sterujących, b_1,b_2,b_3 (stałe całkowite dodatnie) są parametrami końcowymi zmiennych sterujących, c_1,c_2,c_3 (stałe całkowite dodatnie) są parametrami kroku. Jeżeli parametr kroku nie jest podany, to jego wartość jest równa 1. Realizacja ukrytego cyklu DO przebiega podbonie, jak w przypadku listy ukrytego cyklu DO w instrukcjach WE/WY.

5.8. Podprogramy i funkcje

Program składa się z segmentu głównego, funkcji i podprogramów dostarczanych przez kompilator lub definiowanych przez użytkownika. Pierwsza instrukcja segmentu, który jest segmentem głównym, ma postać:

```
PROGRAM name
```

gdzie name jest nazwą programu głównego. Jeżeli name nie występuje, kompilator przydziela nazwę MAIN.

5.8.1. Funkcje standardowe

Argumentem aktualnym wywołania funkcji może być dowolna stała, zmienna, element tablicy, odwołanie do innej funkcji lub dowolne wyrażenie. Ilość i typ argumentów aktualnych musi być zgodny z ilością

i typem argumentów określonych dla danej funkcji standardowej. W zamieszczonej niżej tablicy omówione są poszczególne funkcje standardowe dostępne poprzez kompilator FORTRANu.

Postać funkcji	Typ arg.	Typ fun.	Definicja funkcji
1	2	3	4
ABS(X) IABS(I) DABS(X)	rzecz. całk. p.prec.	rzecz. całk. p.prec.	wartość bezwzględna argumentu
CABS(Z)	zesp.	rzecz.	moduł liczby zespolonej
AINT(X) INT(X) DINT(X)	rzecz. rzecz. p.prec	rzecz. całk. p.prec.	znak argumentu pomnożony przez największą liczbę całkowitą $\leqslant$ X
AMOD(X,Y) MOD(I,J) DMOD(X,Y)	rzecz. całk. p.prec.	rzecz. całk. p.prec.	argument pierwszy modulo argument drugi
AMAXØ(I,J,...) AMAX1(X,Y,...) MAXØ(I,J,...) MAX1(X,Y,...) DMAX1(X,Y,...)	całk. rzecz. całk. rzecz. p.prec.	rzecz. rzecz. całk. całk. p.prec.	maksimum z argumentów, ilość argumentów $\geqslant 2$
AMINØ(I,J,...) AMIN1(X,Y,...) MINØ(I,J,...) MIN1(X,Y,...) DMIN1(X,Y,...)	całk. rzecz. całk. rzecz. p.prec.	rzecz. rzecz. całk. całk. p.prec.	minimum z argumentów, ilość argumentów > 2
FLOAT(I) IFIX(X) SNGL(X) DBLE(X)	całk. rzecz. p.prec. rzecz.	rzecz. całk. rzecz. p.prec.	zamiana typu całk.-rzecz. zamiana typu rzecz.-całk. zamiana typu p.pr.-rzecz. zamiana typu rzecz.-p.pr.
REAL(Z) AIMAG(Z) CMPLX(X,Y)	zesp. zesp. rzecz.	rzecz. rzecz. zesp.	cz. rzecz. liczby zesp. cz.urojona liczby zesp. utworzenie liczby zespolonej z dwóch rzeczywistych
SIGN(X,Y) ISIGN(I,J) DSIGN(X,Y)	rzecz. całk. p.prec.	rzecz. całk. p.prec.	znak argumentu drugiego razy wartość bezwzględna pierwszego argumentu

1	2	3	4
DIM(X,Y) IDIM(I,J)	rzecz. całk.	rzecz. całk.	argument pierwszy minus minimum z obu argumentów
CONJG(Z)	zesp.	zesp.	zespolona liczba sprzężona
EXP(X) DEXP(X) CEXP(Z)	rzecz. p.prec. zesp.	rzecz. p.prec. zesp.	e do potęgi argument
ALOG(X) DLOG(X) CLOG(Z)	rzecz. p.prec. zesp.	rzecz. p.prec. zesp.	logarytm naturalny z argumentu
ALOG10(X) DLOG10(X)	rzecz. p.prec.	rzecz. p.prec.	logarytm o podstawie 10 z argumentu
SQRT(X) DSQRT(X) CSQRT(Z)	rzecz. p.prec. zesp.	rzecz. p.prec. zesp.	pierwiastek kwadratowy z argumentu
SIN(X) DSIN(X) CSIN(Z)	rzecz. p.prec. zesp.	rzecz. p.prec. zesp.	sinus argumentu
COS(X) DCOS(X) CCOS(Z)	rzecz. p.prec. zesp.	rzecz. p.prec. zesp.	cosinus argumentu
TANH(X)	rzecz.	rzecz.	tangens hiperb. argumentu
ATAN(X) DATAN(X)	rzecz. p.prec.	rzecz. p.prec.	arcus tangens argumentu
ATAN2(X,Y) DATAN2(X,Y)	rzecz. p.prec.	rzecz. p.prec.	arcus tangens ilorazu argumentu pierwszego przez drugi argument
IOR(X,Y)	całk.	całk.	suma logiczna argumentów
IAND(X,Y)	całk.	całk.	iloczyn logiczny argum.
NOT(X)	całk.	całk.	negacja logiczna argum.
IEOR(X,Y)	całk.	całk.	różnica symetryczna arg.
ISHFT(X,Y)	całk.	całk.	przesunięcie argumentu X o Y pozycji w lewo gdy Y > 0, w prawo gdy Y < 0

1	2	3	4
BTEST(X,Y)	całk.	log.	sprawdzenie czy bit nr Y w argumencie X jest = 1, jeżeli tak, BTEST=prawda, jeżeli nie, BTEST=fałsz

Źródło: /4/.

5.8.2. Podprogramy FUNCTION i SUBROUTINE

Podprogram stanowi oddzielny segment programu. Nazwy zmiennych, tablic oraz etykiety (z wyjątkiem etykiet w instrukcji EXTERNAL i INTERNAL) definiowane w podprogramie są niezależne od tych samych obiektów występujących w innych podprogramach. Podprogram zaczyna się instrukcją FUNCTION lub SUBROUTINE z listą parametrów formalnych i zakończony jest instrukcją END. Wewnątrz podprogramu musi wystąpić co najmniej jedna instrukcja RETURN, która powoduje powrót do segmentu wywołującego. Komunikacja pomiędzy segmentem wywołującym a segmentem wywoływanym odbywa się poprzez parametry formalne i aktualne oraz przez bloki COMMON.

Parametrem formalnym podprogramu może być: nazwa zmiennej, nazwa tablicy, nazwa podprogramu, nazwa ENTRY, nazwa funkcji standardowej. W podprogramie dopuszcza się, aby wskaźniki graniczne w deklaracji tablicy były zmiennymi (tablica dynamiczna). W takim przypadku lista parametrów formalnych musi zawierać nazwę tablicy oraz nazwy zmiennych całkowitych, które będą użyte jako wskaźniki graniczne.

Parametrem aktualnym wywołania podprogramu może być: dowolna stała, zmienna, element tablicy, wyrażenie arytmetyczne, logiczne lub relacji, nazwa tablicy, nazwa podprogramu, nazwa ENTRY, nazwa funkcji standardowej. Kolejność i typ parametrów aktualnych muszą być zgodne z kolejnością i typem odpowiadających im parametrów formalnych. Jeżeli nazwa podprogramu jest użyta jako parametr aktualny, to musi ona wystąpić również w instrukcji EXTERNAL.

Podprogram SUBROUTINE rozpoczyna się instrukcją:

```
SUBROUTINE nazwa
```

lub

```
SUBROUTINE nazwa (p1,p2,...,pn)
```

gdzie nazwa służy do identyfikacji podprogramu, a $p_1,p_2,\ldots,p_n$ jest listą parametrów formalnych.

Podprogram SUBROUTINE jest wywoływany z innego podprogramu lub programu odpowiednio instrukcjami:

```
CALL nazwa
```

lub

```
CALL nazwa (a1,a2,...,an)
```

gdzie nazwa jest nazwą wywoływanego podprogramu, natomiast $a_1,a_2,\ldots,a_n$ jest listą parametrów aktualnych.

Podprogram FUNCTION zwraca wartość związaną z nazwą funkcji. Rozpoczyna go instrukcja:

```
FUNCTION nazwa(p1,p2,...,pn)
```

lub

```
typ FUNCTION nazwa(p1,p2,...,pn)
```

gdzie nazwa jest nazwą podprogramu FUNCTION, typ określa typ funkcji, $p_1,p_2,\ldots,p_n$ jest niepustą listą parametrów formalnych. Podprogram FUNCTION musi zawierać co najmniej jedną instrukcję podstawienia ustalającą wartość funkcji.

Wywołanie podprogramu FUNCTION ma postać:

```
nazwa(a1,a2,...,an)
```

gdzie nazwa jest nazwą wywoływanego podprogramu, natomiast $a_1,a_2,\ldots,a_n$ jest listą parametrów aktualnych.

Instrukcja ENTRY umożliwia tworzenie kilku punktów wejścia do podprogramu FUNCTION lub SUBROUTINE. Ma ona następującą postać:

```
ENTRY nazwa
```

lub

```
ENTRY nazwa(p1,p2,...,pn)
```

gdzie nazwa jest nazwą wejścia ENTRY, $p_1,p_2,\ldots,p_n$ jest listą parametrów formalnych. Instrukcja ENTRY bez parametrów formalnych może wystąpić tylko w podprogramie SUBROUTINE.

5.8.3. S t a n d a r d o w e p o d p r o g r a m y

W systemie FORTRAN IV MERA-4ØØ programista może korzystać z wielu standardowych podprogramów. Niektóre z nich zostały opisane niżej.

Podprogram OVERFL bada błędy operacji arytmetycznych. Postać wywołania jest następująca `CALL OVERFL(m)` gdzie m jest zmienną lub elementem tablicy typu całkowitego, w którym umieszczony jest wynik badania. Jeżeli wystąpiły błędy arytmetyczne operacji zmiennoprzecinkowych lub błędy dzielenia stałoprzecinkowego, to na bitach od Ø do 3 parametru m zostaną umieszczone jedynki oznaczające:

bit Ø - nadmiar dzielenia stałoprzecinkowego,
bit 1 - nadmiar operacji zmiennoprzecinkowej,
bit 2 - podmiar operacji zmiennoprzecinkowej,
bit 3 - nieznormalizowane argumenty operacji zmiennoprzecinkowych lub dzielnik równy Ø.

Podprogram DATA podaje aktualną datę kalendarzową. Postać wywołania jest następująca `CALL DATA(j)` gdzie j stanowi trzysłowową tablicę typu całkowitego. Kolejne słowa tablicy zawierają:

j(1) - miesiąc
j(2) - dzień
j(3) - rok

Podprogram TIME podaje bieżący czas zegarowy. Postać wywołania jest następująca `CALL TIME(j)` gdzie j stanowi trzysłowową tablicę

typu całkowitego, w której zostaje umieszczona informacja:

j(1) - godzina zegara 24-godzinnego

j(2) - minuty

j(3) - sekundy

Podprogram MESSAG drukuje wiadomość dla operatora. Postać wywołania jest następująca `CALL MESSAG(i,j,k)` gdzie i jest tablicą zawierającą drukowany tekst, j jest zmienną lub stałą całkowitą (j=Ø wykonanie zadania będzie kontynuowane po wydruku, j=1 po wydruku wiadomości wykonanie zadania może być wznowione przez operatora), k jest zmienną lub elementem tablicy typu całkowitego (pod ten argument zostanie zwrócone słowo komunikacji zadania). Drukowany tekst powinien zawierać nie więcej niż 62 znaki i powinien być zakończony słowem Ø.

5.9. Kompilacja i wykonanie programu

Program źródłowy w języku FORTRAN musi być przygotowany zgodnie z definicją składni tego języka. Na końcu programu umieszczany jest znacznik końca zbioru (znaki $$ na początku linii).

W celu wykonania fazy kompilacji, kodowania i łączenia programu należy wykonać następującą procedurę:

```
$JOB
$EXE FORC
$WEO SO
$ASS SI SO
$REW SI
$EXE MAC,,NOLO
$WEO BO
$JOB
$EXE EDI,,NOMA,NOLO
LINK BI
EXI
```

W fazie kompilacji badane są przez translator opcje, które mają następujące znaczenie:

Opcja	Stan	Znaczenie opcji
UØ	1	Generuj wywołanie procedury śledzącej kolejność wykonywania się instrukcji programu
U1	1	Generuj wywołanie procedury informującej o numerze instrukcji w przypadku błędu
U2	1	Generuj sprawdzenie poprawności wskaźników przy odwoływaniu do elementu tablicy
U3	1	Generuj instrukcje testujące
U4		Gdy opcja BO jest zapalona, to: jeżeli U4=Ø, generowany jest kompilat skompresowany, jeżeli U4=1, generowany jest kompilat nieskompresowany
U5	1	Drukuj identyfikator zamiast numeru instrukcji przy błędzie w fazie kompilacji
U6	1	Listuj kompilat
LO	1	Listuj program źródłowy
BO	1	Generuj kompilat
HO	1	Zawieś pracę translatora po napotkaniu znacznika końca zbioru lub po wystąpieniu instrukcji END

Wykrycie błędu w programie źródłowym przez translator jest w fazie kompilacji sygnalizowane na zbiorze LO komunikatem o postaci:

<nr porządkowy instrukcji> ERROR NR<nr błędu> [<parametry>]

Spis numerów błędów wykrywanych w trakcji kompilacji oraz ich znaczenie zamieszczone są w Dodatku A.

Wykonanie programu inicjowane jest następującymi dyrektywami:

ØJOB
ØEXE nazwa,BI,opt_1,...,opt_n

gdzie nazwa jest nazwą pierwszego segmentu programu źródłowego (segmentu głównego), a opt_1,...,opt_n są opcjami fazy egzekucji.

Błędy wykryte w trakcie wykonywania programu sygnalizowane są na zbiorze CO w postaci komunikatów:

JOB (nazwa) ERROR <nr błędu> [<nr instrukcji>]

gdzie nazwa jest nazwą wykonywanego programu. Znaczenie poszczególnych numerów błędów podane zostało w Dodatku A.

5.10. Rozszerzenia i odstępstwa implementacji języka FORTRAN IV MERA-400 od standardu ISO

W translatorze FORTRAN IV-S wprowadzono następujące rozszerzenia i odstępstwa od standardu /4/:

- zestaw znaków interpretowanych przez translator został rozszerzony o następujące znaki " ' < >
- wprowadzono instrukcje testujące,
- ilość wierszy kontynuacji nie jest ograniczona,
- nie można kontynuować w następnym wierszu żadnego obiektu z wyjątkiem stałej tekstowej, łańcucha alfanumerycznego i stałej zespolonej,
- obiekty instrukcji powinny być oddzielone od siebie spacjami, separatorami lub operatorami,
- instrukcje bierne poza instrukcją FORMAT nie mogą być etykietowane,
- wartość liczby reprezentującej etykietę nie może być większa od 32767,
- wprowadzono pseudoetykietę Ø,
- nazwy symboliczne oraz stałe muszą być zapisywane bez spacji między znakami,
- długość nazwy może być dowolna, z tym, że translator interpretuje sześć pierwszych znaków,
- nazwy wspólnych bloków COMMON powinny różnić się na pierwszych pięciu znakach,
- wprowadzono stałe hexadecymalne i stałe CAN,
- wprowadzono instrukcję IMPLICIT zmieniającą naturalną implikację typu zmiennej, tablicy lub funkcji,
- dopuszcza się postać instrukcji: GOTO i
- dopuszcza się IF logiczne typu arytmetycznego: IF(wyr)e_1,e_2,
- parametr w instrukcji STOP i PAUSE może być stałą całkowitą lub tekstową.

- nie nakłada się żadnych ograniczeń na wartości parametrów pętli DO,
- zezwala się na redefiniowanie zmiennej sterującej wewnątrz pętli DO,
- jeżeli parametry pętli DO definiują cykl pusty, zostaje on pominięty,
- dopuszcza się wyrażenia arytmetyczne dla parametrów pętli DO,
- identyfikator specyfikacji wzorca we wszystkich instrukcjach WE/WY może być zmienną całkowitą,
- dopuszcza się swobodną konwersję danych w instrukcjach WE/WY,
- wprowadza się instrukcje READ i WRITE transmitując dane z i do zbiorów bezpośredniego dostępu,
- parametry ukrytego cyklu DO mogą mieć wartości ujemne,
- zmienna sterująca oraz parametry ukrytego cyklu DO (jeżeli są zmiennymi) mogą występować w listach WE/WY jako samodzielne elementy,
- instrukcje READ i WRITE uzupełniono o opcjonalne indykatory błędu i znacznika końca zbioru,
- wprowadzono instrukcje ENCODE i DECODE,
- wprowadzono instrukcje INREC i OUTREC,
- wprowadzono instrukcje DEFINE FILE i FIND,
- dopuszcza się etykiety w instrukcji EXTERNAL,
- dopuszcza się nazwy tablic na listach zmiennych instrukcji DATA,
- wprowadza się listę ukrytego cyklu DO w instrukcji DATA,
- w instrukcji FORMAT wprowadzono opisy pól Z, C, T, Q,
- nie narzuca się żadnych ograniczeń na głębokość struktury nawiasowej instrukcji FORMAT,
- wprowadzono zewnętrzne separatory pól, dla READ i DECODE,
- wprowadzono zmienne parametry dla wszystkich opisów pól (oprócz H),
- wprowadzono instrukcję ENTRY,
- rozszerzono zbiór funkcji standardowych,
- wprowadzono instrukcję PROGRAM,
- wprowadzono instrukcję ASSEMBLER,
- dopuszczono mieszanie typów argumentów w wyrażeniu arytmetycznym,
- wprowadzono wielokrotną instrukcję podstawienia.

5.11. Dodatek A. Wykaz błędów kompilacji i egzekucji programów fortranowskich

B ł ę d y f a z y k o m p i l a c j i

1 - Pierwszy wiersz instrukcji jest wierszem kontynuacji
2 - Niedozwolony obiekt w polu etykiety
3 - Niedozwolony obiekt na początku instrukcji
4 - Etykieta bez instrukcji
5 - Definicja etykiety równej Ø
6 - Etykieta przed instrukcją bierną
7 - Powtórna definicja etykiety
8 n W instrukcji wykryto niedozwolone znaki albo nadmiarowe lub błędnie zbudowane stałe; n - ilość wykrytych błędów
9 - Niezidentyfikowana instrukcja
10 - Instrukcja niedozwolona w translowanym segmencie lub instrukcja IMPLICIT w niedozwolonym miejscu
11 - Niedozwolona instrukcja uwarunkowana w IF logicznym
12 - Błędna składnia instrukcji IMPLICIT
13 - Błędny deklarator typu w instrukcji IMPLICIT
20 - Niewłaściwy obiekt w instrukcji DO zamiast =, brak separatora lub niedozwolony separator w instrukcji DO
21 - Znak + lub - przed zmienną sterującą w instrukcji DO
22 - Obiekt różny od etykiety po słowie DO
23 - Pętla DO do wcześniejszej instrukcji
24 - Przecinające się pętle DO
25 - Redefinicja zmiennej sterującej występującej w poprzednim niezakończonym DO
26 - Obiekt różny od oczekiwanej stałej lub zmiennej typu całkowitego, także niedozwolona stała ze znakiem
27 - Niewłaściwa składnia instrukcji GO TO
28 - Obiekt różny od oczekiwanej zmiennej typu całkowitego
29 - Niewłaściwy obiekt w miejscu etykiety
30 - Oczekiwano słowa TO (w instrukcji GO TO lub ASSIGN)
31 - Błąd składni w instrukcji STOP lub PAUSE
32 - Oczekiwano końca instrukcji, instrukcja "za długa"
33 - Błąd składni w instrukcji IF
34 - Niewłaściwy typ wyrażenia w IF arytmetycznym
35 - Niewłaściwe wyrażenie w IF logicznym lub wyrażenie logiczne w DO
36 - Niedozwolona instrukcja jako graniczna instrukcja DO
37 - Niedozwlone wystąpienia Ø jako etykiety

40 - Wystąpienie w instrukcji EXTERNAL obiektu z INTERNAL
41 - Powtórna definicja obiektu w EXTERNAL
42 - Brak separatora lub separator zamiast nazwy
43 - Typ zmiennej niezgodny z poprzednią specyfikacją
44 - Niedozwolony obiekt w deklaracji tablicy jako nazwa tablicy
45 - Zbyt wiele indeksów w deklaracji tablicy
46 - Niedozwolony obiekt w deklaracji tablicy jako indeks
47 - Zmienna nie będąca parametrem formalnym występuje jako indeks
48 - Indeks tablicy jest większy od rozmiaru tablicy
49 - Niewłaściwa wartość wskaźnika granicznego tablicy
50 - Nadmiarowy rozmiar w deklaracji tablicy
51 - Niedozwolony obiekt jako nazwa bloku obszaru wspólnego
52 - Niedozwolony obiekt w bloku obszaru wspólnego
53 - Powtórna deklaracja nazwy w bloku obszaru wspólnego
54 - Niedozwolony obiekt na liście DATA lub EQUIVALENCE
55 - Zmienna będąca parametrem formalnym na liście DATA lub EQUIVALENCE
56 - Niedozwolony obiekt na liście EXTERNAL
57 - Nazwa zamiast stałej na liście stałych w DATA
58 - Niewłaściwy obiekt lub niewłaściwa ilość wskaźników w konstrukcji z ukrytym DO
60 - Obiekty różnych bloków obszaru wspólnego na tej samej liście EQUIVALENCE
61 - Równoważność dwóch różnych elementów tego samego bloku obsz. wsp.
62 - Błąd struktury EQUIVALENCE
63 - Przekroczenie lewej granicy bloku obszaru wspólnego
64 n Długość listy stałych większa od długości listy zmiennych w instrukcji DATA; n - kolejny numer listy DATA
65 n Długość listy zmiennych większa od długości listy stałych w instrukcji DATA; n - kolejny numer listy DATA
66 n Niezgodność typów stałej i zmiennej w instrukcji DATA; n - kolejny numer listy DATA
67 n Obliczony w trakcie realizacji ukrytego DO w instrukcji DATA indeks globalny jest większy od rozmiaru tablicy; n - kolejny numer listy DATA
70 n Niedozwolone odwołanie do etykiety n
71 n Otworzone niezamknięte DO; n - etykieta instrukcji granicznej
72 n Niezdefiniowana etykieta n
73 - W podprogramie brak instrukcji RETURN
74 - Instrukcja END na początku segmentu
75 - Brak instrukcji czynnej w segmencie różnych od BLOCK DATA

80 - Błąd struktury lewej strony instrukcji podstawienia
81 - Niedozwolona sekwencja operatorów
82 - Niedozwolony separator
83 - Błąd struktury nawiasowej
84 - Brak operatora
85 - .NOT. wystąpił jako operator binarny
86 - Niewłaściwie zakończona instrukcja CALL
87 - Niedozwolony obiekt po lewej stronie instrukcji podstawienia
88 - Niedozwolony obiekt przed "(" w wyrażeniu
89 - Brak nazwy podprogramu w instrukcji CALL
90 - Brak "(" po nazwie tablicy lub funkcji albo stała tekstowa, hexadecymalna lub CAN w wyrażeniu
91 - Niedozwolona nazwa w instrukcji CALL
92 - Niewłaściwa ilość argumentów aktualnych
93 - Nazwa funkcji lokalnej lub standardowej jako parametr aktualny
94 - Niezgodność wywołań tego samego podprogramu
100 - Błąd syntaktyczny w wyrażeniu wskaźnikowym
101 - Niedozwolony obiekt w wyrażeniu wskaźnikowym
102 - Brak nazwy zmiennej całkowitej po "="
103 - Brak stałej całkowitej po "+" lub "-"
104 - Przekroczenie wartości wskaźnika tablicy
105 - Zbyt mała ilość wskaźników tablicy
106 - Przekroczenie ilości wskaźników tablicy
107 - Przekroczenie rozmiaru tablicy
108 - Zerowa wartość wskaźnika
110 - Niedopuszczalne mieszanie typów argumentów
111 - Niewłaściwy typ argumentu operatora binarnego
112 - Niewłaściwy typ parametru funkcji standardowej jednoargumentowej
113 - Niewłaściwy typ parametru funkcji standardowej wieloargumentowej
114 - Niezgodne typy argumentów w kolejnych wywołaniach tego samego podprogramu
120 - Brak nazwy definiowanego podprogramu
121 - Niedozwolona nazwa funkcji lokalnej lub podprogramu
122 - Brak argumentu formalnego lub brak "(" po nazwie podprogramu
123 - Niedozwolony parametr formalny lub brak parametru
124 - Brak ")" lub "," na liście parametrów formalnych
125 - Powtórzony argument formalny w definicji funkcji lokalnej
126 - Definicja funkcji lokalnej w niedozwolonym miejscu programu
130 - Błędna struktura nawiasów instrukcji FORMAT
131 - Brak etykiety przed instrukcją FORMAT

132 n W instrukcji FORMAT wykryto: niedozwolone znaki, wartość parametrów opisów pól spoza dopuszczalnego zakresu, brak separatorów między opisami pól lub opisy pól o nieprawidłowej budowie;
n - ilość wykrytych błędów
133 - Błędny zmienny parametr opisu pola
134 - Błędny separator, brak separatora albo innego obiektu
135 - Błędna składnia instrukcji
136 - Błędne elementy opisu zbioru w instrukcji DEFINE FILE
137 - Błędne, wielokrotne lub niedozwolone ERR/END
138 - Błędny typ wyrażenia specyfikującego długość lub numer rekordu
139 - Błędna specyfikacja identyfikatora urządzenia, bufora lub zbioru bezpośredniego dostępu
140 - Błędna specyfikacja identyfikatora specyfikacji wzoru
141 - Błędny separator lub brak separatora na liście WE/WY
142 - Zła struktura nawiasów listy WE/WY lub ukryty cykl DO bez listy
143 - Niedozwolony obiekt na liście WE/WY
144 - Zbyt wiele, zbyt mało lub błędne parametry ukrytego cyklu DO

Źródło: /4/.

B ł ę d y f a z y e g z e k u c j i

Błędy funkcji standardowych i pomocniczych

Numer błędu	Funkcja	Określenie błędu
1	EXP	argument nadmiarowy
2	IFIX	argument nadmiarowy
3	SQRT	argument ujemny
4	SIGN	argument drugi = Ø
5	ISIGN	argument drugi = Ø
6	AMOD	argument drugi = Ø
7	MOD	argument drugi = Ø
8	DIM	różnica nadmiarowa
9	IDIM	różnica nadmiarowa
10	ALOG	argument $\leqslant$ Ø
11	ALOG1Ø	argument $\leqslant$ Ø
12	AMIN1	liczba argumentów < 2
13	MIN1	liczba argumentów < 2
14	MINØ	liczba argumentów < 2
15	AMINØ	liczba argumentów < 2
16	MAXØ	liczba argumentów < 2
17	AMAXØ	liczba argumentów < 2

18		AMAX1	liczba argumentów < 2
19		MAX1	liczba argumentów < 2
20		ATAN2	nadmiar w trakcie obliczeń
21		ATAN2	podmiar w trakcie obliczeń
22	logarytm naturalny liczby zesp.		nadmiar w trakcie obliczeń
23	logarytm naturalny liczby zesp.		podmiar w trakcie obliczeń
24	dzielenie		podmiar w trakcie obliczeń
25	liczb		nadmiar w trakcie obliczeń
26	zespolonych		dzielnik = ∅
27		CSQRT	podmiar w trakcie obliczeń
28		CSQRT	nadmiar w trakcie obliczeń
29	mnożenie		podmiar w trakcie obliczeń
30	liczb zesp.		nadmiar w trakcie obliczeń
31		CABS	nadmiar w trakcie obliczeń
32		CABS	podmiar w trakcie obliczeń
33		CEXP	podmiar w trakcie obliczeń
34		CSIN	podmiar w trakcie obliczeń
35	sinus hip. liczby podw. prec.		nadmiar w trakcie obliczeń
36	cosinus hip. liczby podw. prec.		nadmiar w trakcie obliczeń
37		TANH	nadmiar w trakcie obliczeń
38		CONJG	podmiar w trakcie obliczeń
39		CONJG	nadmiar w trakcie obliczeń
40	negacja		nadmiar w trakcie obliczeń
41	liszby zesp.		podmiar w trakcie obliczeń
42		DMIN1	ilość argumentów < 2
43		DSIGN	argument drugi = ∅
44		DSQRT	argument ujemny
60		IDINT	argument nadmiarowy
61		IABS	argument nadmiarowy
62		ABS	argument nadmiarowy
63		DLOG	argument ≤ ∅
64		DEXP	argument nadmiarowy
65		DMAX1	liczba argumentów < 2
66		DLOG1∅	argument ≤ ∅
67		ATAN	podmiar lub nadmiar w trakcie obliczeń
68		DATAN2	niepoprawne dane (oba arg. = ∅)

Źródło: /4/.

Błędy potęgowania

Numer błędu	Typ potęgowania	Określenie błędu
69	R**I	podstawa < Ø i wykładnik < Ø
70	R**I	nadmiar w trakcie obliczeń
71	D**I	podstawa < Ø i wykładnik < Ø
72	D**I	nadmiar w trakcie obliczeń
73	I**I	podstawa = Ø i wykładnik < Ø
74	I**I	nadmiar w trakcie obliczeń
75	D**D	podstawa < Ø i wykładnik < Ø
76	D**D	nadmiar w trakcie obliczeń
77	D**R	podstawa < Ø i wykładnik < Ø
78	D**R	nadmiar w trakcie obliczeń
79	R**D	podstawa < Ø i wykładnik < Ø
80	R**D	nadmiar w trakcie obliczeń
81	R**R	podstawa < Ø i wykładnik < Ø
82	R**R	nadmiar w trakcie obliczeń

Źródło: /4/

Pozostałe błędy

100 - nadmiar ilorazu liczb typu całkowitego

101 - nadmiar sumy, różnicy, iloczynu i ilorazu liczb typu rzeczywistego i podwójnej precyzji

102 - podmiar sumy, różnicy, iloczynu i ilorazu liczb typu rzeczywistego i podwójnej precyzji

103 - nieznormalizowana dana typu rzeczywistego lub podwójnej precyzji albo dzielnik = Ø

110 - brak zadanej wartości zmiennej całkowitej na liście instrukcji GO TO podstawianej z listą

111 - wartość wskaźnika wykracza poza długość listy instrukcji GO TO przełączanej

112 - obliczone wartości wskaźników elementu tablicy wskazują na element spoza sumarycznego rozmiaru tablicy

120 - numer strumienia logicznego spoza zakresu 1 - 999

121 - przepełnienie stosów listy i specyfikacji wzorca albo brak miejsca na bufory

122 - identyfikator specyfikacji wzorca w redagowanej instrukcji WE/WY nie jest etykietą instrukcji FORMAT

123 - brak opisu pola lub opis pola skojarzony z elementem listy WE/WY

jest niewłaściwego typu
124 - zerowa wartość kroku ukrytego cyklu DO
125 - błędna specyfikacja wzorca zapisana w tablicy
126 - żądana zmiana rekordu w instrukcji ENCODE i DECODE
127 - brak żądanej specyfikacji zbioru o dostępie bezpośrednim
128 - słownik lub zbiór o dostępie bezpośrednim nie jest umieszczony w pamięci dyskowej
129 - niedozwolony typ transmisji na lub ze zbioru o dostępie bezpośrednim
130 - niedozwolony numer rekordu
140 - przekroczony rozmiar rekordu logicznego
141 - błędny rekord danych
142 - błędne parametry opisu pola
144 - nadmiarowa dana wejściowa lub wyjściowa
145 - napotkanie znacznika końca zbioru albo przekroczenie rozmiaru zbioru

Źródło: /4/.

6. JĘZYK BASIC W SYSTEMIE MERA-400

6.1. Uwagi wstępne

Język BASIC jest popularnym, łatwym do nauki i stosowania środkiem programowania nieskomplikowanych zadań sprowadzających się do obliczeń numerycznych. Stosując BASIC można używać dużego komputera w podobny sposób jak używa się podręcznego kalkulatora, mając oczywiście w razie potrzeby możność wykorzystania pełnej mocy obliczeniowej maszyny. W systemie MERA-400 BASIC jest jednym z najchętniej używanych języków ze względu zarówno na wspomnianą prostotę użycia, jak i z powodu możliwości pracy z kilku końcówek równocześnie, co powoduje że kilku użytkowników może wykorzystywać maszynę w tym samym czasie nie przeszkadzając sobie wzajemnie. Cenną własnością języka BASIC jest też konwersacyjny sposób pracy: maszyna po każdym wprowadzonym przez użytkownika wierszu analizuje jego poprawność i sygnalizuje ewentualne usterki, co daje możliwość natychmiastowego wprowadzania poprawek i w efekcie szybkiego uruchomienia programów; analogiczne czynności i etapy w przypadku innych języków (ASSEMBLER, FORTRAN) zajmują więcej czasu i wymagają większego wysiłku użytkownika maszyny.

Wyliczając zalety języka BASIC nie należy tracić z pola widzenia jego wad. Należą do nich ograniczony zakres zastosowań i niska sprawność obliczeniowa konstruowanych programów. Język BASIC nadaje się do programowania zagadnień z zakresu obliczeń numerycznych. Inne zadania, na przykład popularne w zastosowaniach gospodarczych przetwarzanie danych nie mogą być realizowane z jego wykorzystaniem. Szybkość obliczeń wykonywanych programem w języku BASIC jest mniejsza, niż szybkość tych samych obliczeń wykonywanych przez program napisany w języku

FORTRAN, przeto dla zadań powtarzalnych, wymagających dużej liczby obliczeń - warto posługiwać się innymi niż BASIC językami programowania.

6.2. Postać najprostszych instrukcji i struktura programu

Program nakazujący maszynie wykonanie określonej sekwencji obliczeń budowany jest w postaci zbioru wierszy pisanych na końcówce, przy czym każdy wprowadzany wiersz musi zaczynać się swoim numerem, zaś ostatnim (w sensie prowadzonej numeracji) wierszem musi być wiersz END kończący program. Na szczególną uwagę zasługują numery wierszy, gdyż dzięki ich użyciu można w języku BASIC niezwykle łatwo korygować błędy programu. Maszyna automatycznie porządkuje wprowadzane wiersze według ronsącej kolejności numerów, w związku z czym można uzupełnić program w dowolnym momencie i w dowolnym miejscu poprzez podawanie odpowiednich numerów wierszy. Dla ułatwienia sobie ewentualnych przyszłych poprawek programista powinien prowadzić numerację wierszy "luźno" - typowy i zalecany sposób numeracji polega na nadaniu pierwszemu wierszowi programu numeru 100, a dalszym kolejnych numerów w odstępach co 10:, 110, 120, 130, itd. Dzięki temu przypomniawszy sobie o konieczności dokonania jakichś uzupełnień lub wstawek będziemy mogli w czasie dalszej pracy bez trudu wprowadzać poprawki. Powtórne wpisanie wiersza o tym samym numerze, co uprzednio wprowadzony powoduje skasowanie starej zawartości wiersza i wprowadzenie jego nowej wersji. W szczególności wypisanie wiersza złożonego z samego tylko numeru kasuje starą zawartość wiersza o podanym numerze. Przykładowo:

Pierwotny tekst programu ma postać (znaczenie instrukcji będzie dalej wyjaśnione):

```
100 X = A + B
110 PRINT A,B
120 END
```

Program ten ma usterki i powinien być poprawiony. Programista pisze

więc poprawki w następującej postaci:

```
 80 A = 5
 90 INPUT B
110 PRINT X
```

i otrzymuje nową wersję programu:

```
 80 A = 5
 90 INPUT B
110 X = A + B
110 PRINT X
120 END
```

Każdy wprowadzony przez programistę wiersz jest natychmiast kontrolowany pod kątem jego poprawności i logicznej zgodności z dotychczas wprowadzonymi wierszami. Wykrycie sprzeczności lub błędu powoduje, że maszyna pod napisanym przez programistę wierszem wyprowadzi tekst

ERROR xx IN LINE nn

gdzie xx jest numerem określającym rodzaj błędu (patrz zamieszczony na końcu rozdziału wykaz) a nn jest numerem błędnej linii (zwykle jest to numer ostatnio wprowadzonej linii, lecz nie zawsze, ponieważ ostatnio wprowadzona linia może umożliwić wykrycie błędu w innej, uprzednio już akceptowanej linii). W przypadku gdy linia jest bezbłędna wyprowadzany jest znak * stanowiący zgłoszenie gotowości maszyny do przyjęcia kolejnego wiersza. Tak więc na końcówce każdy wprowadzony wiersz poprzedzany jest gwiazdką stanowiącą zgłoszenie maszyny, dzięki czemu można odróżnić wiersze nowo wprowadzane od przepisywanych przez komputer z pamięci (na przykład w trakcie wykonywania komendy LIST - patrz dalej).

Programowanie obliczeń opiera się na konstrukcji instrukcji podstawienia. Jej ogólna postać jest następująca:

⟨numer wiersza⟩ zmienna = wyrażenie

Między numerem wiersza a zmienną może (lecz nie musi) występować słowo kluczowe LET. Wyjaśnimy teraz pojęcia użyte w definicji instrukcji podstawienia. Zmienna w języku BASIC może być zmienną prostą lub elementem tablicy. Pozostawiając sprawę tablic i ich elementów do późniejszego omówienia zdefiniujemy zmienną prostą jako oznaczenie literowe (poje-

dyncza litera lub litera po której następuje cyfra), któremu w trakcie wykonywania programu można nadawać wartości - używając ich potem w obliczeniach lub zmieniając wedłe potrzeby tak jak wartości zmiennych w algebrze. Liter jest w BASICu 27 (A - Z oraz znak @ traktowany również jak litera). Każda z nich może występować jako zmienna sama, lub być uzupełniona cyfrą - od 0 do 9. Przeto różnych zmiennych można zdefiniować w programie najwyżej 297.

Wyrażenie jest zapisem wzoru matematycznego, według którego mają być wykonane określone obliczenia. Występują w nim stałe (wartości liczbowe o jawnie podanej wartości), zmienne oraz funkcje połączone operatorami arytmetycznymi: dodawania (+), odejmowania (-), mnożenia (*), dzielenia (/) oraz potęgowania (↑). Działania nakazane operatorami wykonywane są w kolejności wynikającej z priorytetów: najpierw potęgowania, potem mnożenia i dzielenia a na końcu dodawania i odejmowania. Kolejność tę można jednak dowolnie modyfikować przez stosowanie nawiasów "(" i ")", przy czym liczba użytych nawiasów i stopień ich "zanurzenia" nie są limitowane, co daje możliwość budowania wyrażeń o dużej złożoności.

Stałe występujące w wyrażeniach mogą mieć postać liczby całkowitej, liczby z częścią ułamkową (oddzielaną od części całkowitej kropką) lub liczby w zapisie wykładniczym, z zastosowaniem litery "E" oznaczającej "razy dziesięć do potęgi". Przykłady stałych:

3 -17 345.76 -123.04 7E-3 4.6E4 -6.71E12

Funkcje występujące w wyrażeniach dzielą się na standardowe i definiowane. Każda funkcja posiada trzyliterową nazwę i jest zależna od jednego argumentu, który może być dowolnym wyrażeniem. Przyjmując, że argumentem funkcji ma być zmienna X możemy wymienić następujące funkcje standardowe:

ABS(X)	wartość bezwzględna
ATN(X)	arcus tangens
COS(X)	cosinus (argument w radianach)
COT(X)	cotangens (argument w radianach)
EXP(X)	funkcja wykładnicza e^X

FRA(X)	największa całkowita nie większa niż X
LOG(X)	logarytm naturalny
ODD(X)	równe 0 dla X nieparzystych i 1 dla parzystych
SGN(X)	znak X (przyjmuje wartości -1, 0, +1)
SIN(X)	sinus (argument w radianach)
SQR(X)	pierwiastek kwadratowy
TAN(X)	tangens (argument w radianach)

Programista może ten zestaw uzupełnić pisząc i definiując własne funkcje, których nazwa musi mieć postać:

FNy(X)

gdzie y jest dowolną literą. Możliwe jest więc zdefiniowanie najwyżej 27 własnych funkcji, nazwanych FNA, FNB, FNC, ... Sposób definicji funkcji będzie dalej dyskutowany.

Obok funkcji z argumentami możliwe jest używanie standardowych funkcji bez argumentów:

COL	ilość kolumn ostatnio używanej macierzy
DET	wyznacznik ostatnio odwracanej macierzy
EPS	najmniejsza liczba dla której 1 + EPS > 1
INF	największa przedstawialna w maszynie liczba
@PI	stała (=3.1415926535)
RND	liczba przypadkowa z przedziału [0,1]
ROW	ilość wierszy ostatnio używanej macierzy

Zadaniem instrukcji podstawienia jest obliczenie podanego wyrażenia i nadanie obliczonej wartości zmiennej stojącej po lewej stronie znaku "z". Przykładowo:

```
10 X = 5
20 Y = X + 3
30 Z = SIN(X)/(COS(Y) - SIN(Y))
```

W wyniku wykonania przytoczonych instrukcji zmienna X będzie miała wartość 5, Y=8 a Z = -1.179.

Stosując instrukcję podstawienia można nakazać maszynie wykonanie dowolnych obliczeń; w istocie jest to najczęściej stosowana i najważniejsza ze wszystkich instrukcji. Aby jednak mieć pożytek z obliczeń komputera trzeba mieć możność nakazania wydrukowania wyników, gdyż inaczej nie ma możliwości ich wykorzystania. Instrukcja drukowania ma

postać:

⟨numer wiersza⟩ PRINT ⟨lista drukowanych wartości⟩

Drukowane wartości są najczęściej zmiennymi, wówczas wykonanie instrukcji PRINT polega na wydrukowaniu wartości wymienionych zmiennych. Elementy listy drukowanych wartości mogą być rozdzielone znakiem przecinka "," lub średnika ";" co nie jest bez znaczenia z punktu widzenia postaci uzyskiwanego wydruku. Używając średnika uzyskujemy wydruk "gęsty", to znaczy kolejne drukowane wartości (zajmujące w zależności od swojej wartości większą lub mniejszą ilość miejsca na wydruku stosownie do posiadanej liczby cyfr znaczących) rozmieszczane są jedna za drugą z zachowaniem minimalnych odstępów niezbędnych dla zapewnienia czytelności wydruku. Taka forma jest wygodna, gdy program powinien jednorazowo wydrukować pewną (niewielką) liczbę wyników. Często w obliczeniach komputerowych tworzone są zbiory wyników liczące setki pozycji. W tym przypadku wydruk "gęsty", jakkolwiek przyczania się do oszczędności papieru, utrudnia analizę i interpretację wyników. Wówczas korzystniejszy jest wydruk "tabelaryczny" w którym kolejne wyniki zajmują ustalone pozycje w wierszu co ułatwia ich analizę. Wydruk tego typu uzyskuje się wykorzystując jako separator w liście znak przecinka ",". Powoduje on wydruk kolejnej danej na początku kolejnej "strefy", przy czym zazwyczaj strefy mają ustaloną szerokość (15 znaków) i dają efekt wyrównanych kolumn wydruku. Znak "," może być używany za każdym razem, kiedy chcemy przemieścić punkty drukowania o kolejną strefę, zatem możliwe jest stosowanie go w dowolnym miejscu, przykładowo zapis

```
10 PRINT,,A,,,B
```

spowoduje wydrukowanie wartości zmiennej A poczynając od 30-tej pozycji, zaś wartości zmiennej B poczynając od 75-tej pozycji, co daje możliwość dość elastycznego formatowania wydruku. Dalsze możliwości w tym zakresie wiążą się ze stosowaniem wzorców wydruku ustalanych przez programistę, co będzie omówione nieco dalej.

Rozważmy przykład:

```
100 R = 50
110 S = @PI * R↑2
120 PRINT R; S
130 END
```

Program podany w przykładzie jest bardzo prosty, ale może być wykonany przez maszynę i dostarczy w wyniku dwu liczb: wartość zmiennej R (wynoszącej 50 i interpretowanej jako promień koła) i wartości zmiennej S, interpretowanej jako powierzchnia koła o podanym promieniu R. Zgodnie z wcześniejszymi rozważaniami zauważmy, że wyniki zostaną wydrukowane jeden obok drugiego z minimalnym odstępem, jednak ich interpretacja może napotykać na trudności, gdyż nie będzie wiadomo, co która liczba oznacza. Aby temu przeciwdziałać można na liście drukowanych wartości obok zmiennych, których wartości zamierzamy drukować, umieścić teksty objaśniające. Teksty takie ujmuje się w znaki cudzysłowu "
i traktuje podobnie, jak elementy podlegające drukowaniu, to znaczy separuje przecinkiem lub średnikiem od pozostałych elementów listy. Przykładowo:

```
100 R = 50
110 S = @PU * R↑2
120 PRINT "DLA KOŁA O PROMIENIU R=";R;" POWIERZCHNIA S=";S
130 END
```

Wydruk uzyskany z tego programu będzie znacznie łatwiej czytelny, niż wydruk z poprzedniej wersji, gdyż każda ukazująca się na drukarce liczba będzie opisana w zrozumiały sposób.

Organizacja wydruku za pomocą instrukcji PRINT może być udoskonalona przez zastosowanie tabulatora. Tabulator pozwala ustawić punkt, w którym drukowany będzie kolejny element listy wartości, przy czym pozycja początku drukowanego pola zadawana może być w postaci dowolnego wyrażenia, będącego argumentem funkcji TAB (tabulatora). Daje to bardzo ciekawe możliwości, na przykład instrukcja

```
200 PRINT TAB(X);"*"
```

pozwala umieścić znak gwiazdki * w punkcie odległym od lewego marginesu

wydruku o odległość odpowiadającą aktualnej wartości zmiennej X (o ile wartość ta jest dodatnia i mniejsza od maksymalnej liczby znaków w wierszu), co pozwala łatwo "rysować" na drukarce dowolne wykresy.

Instrukcja druku posiada jeszcze kilka interesujących możliwości. Pierwsza z nich polega na możliwości "sklejania" w jednej linii wydruku pochodzącego z kilku instrukcji PRINT. Wystarczy w tym celu po ostatnim elemencie drukowanej listy umieścić odpowiedni separator ("," lub ";"), a kolejna instrukcja druku będzie kontynuowała wydruk w tej samej linii, zamiast zaczynać do od nowego wiersza. Druga możliwość polega na tym, że elementem drukowanej listy może być dowolne wyrażenie, którego wartość obliczana jest w momencie dokonywania wydruku i ukazuje się na drukarce bez konieczności używania osobnej zmiennej dla przechowania tej wartości. Dla ilustracji rozważmy przykład tego samego (merytorycznie) programu, co wyżej rozważany, ale inaczej zapisanego:

```
100 R = 50
110 PRINT "PROMIEN  KOŁA WYNOSI "; R;
120 PRINT "ZAS JEGO POWIERZCHNIA WYNOSI "; @PI * R↑2
130 END
```

Czy telnik zapewne zirytowany jest już dyskutowaniem trywialnego przykładu programu obliczającego powierzchnię koła, szczególnie że program ten pozwala wyznaczać powierzchnię tylko jednego koła o z góry ustalonej średnicy R = 50. Przy całej swojej trywialności program ten byłby odrobinę bardziej sensowny, gdyby liczył powierzchnię dowolnego koła. Istotnie, większość programów komputerowych konstruowanych jest w ten sposób, aby rozwiązywały one pewną klasę zadań. Do tego nieodzowne jest jednak poznanie instrukcji pozwalającej w trakcie funkcjonowania programu wprowadzać do tego programu wartości liczbowe dla zmiennych, czyli tzw. instrukcji czytania. W języku BASIC dostępne są instrukcje czytania READ i INPUT, przy czym wygoda stosowania tej drugiej jest na tyle duża, że warto głównie ją omówić. Jej postać jest następująca:

⟨numer wiersza⟩ INPUT ⟨lista zmiennych, dla których czytamy wartości⟩

Po napotkaniu instrukcji INPUT maszyna przerywa obliczenia i domaga się od użytkownika programu (którym może być programista lub inna osoba, wykorzystująca napisany program) aby podał on wartości liczbowe dla wszystkich zmiennych znajdujących się na liście po słowie INPUT. Kolejno wprowadzane dane podstawiane są przez maszynę jako wartości kolejnych zmiennych. "Domaganie się" wartości dla zmiennych polega na wyprowadzaniu przez komputer znaku "?" na drukarkę i nie podejmowaniu obliczeń, dopóki dla wszystkich zmiennych z listy nie zostaną podane wartości. Możliwe jest wprowadzanie od razu większej liczby wartości w odpowiedzi na pojedynczy monit maszyny ("?"), przy czym kolejno wprowadzane wartości rozdzielać należy przecinkami. Jeśli wprowadzonych wartości będzie mniej, niż elementów listy w instrukcji INPUT pojawi się kolejny monit ("?"), jeśli ich będzie za dużo - nastąpi zapamiętanie nadmiarowych wartości i wykorzystanie ich przy obsłudze (ewentualnej) następnej wykonywanej instrukcji INPUT. Oto przykład wykorzystania instrukcji INPUT:

```
100 INPUT R
110 S = @ PI ≡ R↑2
120 PRINT "R = ";R;" S= ";S
130 END
```

Podana wyżej wersja programu umożliwia obliczenie powierzchni dowolnego koła, o promieniu podawanym dopiero w momencie wykonywania programu. Ma ona jednak charakterystyczny mankament. Otóż użytkownik programu przystępując do jego wykorzystania zaskakiwanyjest znakiem "?" będącym żądaniem podania wartości dla promienia obliczanego koła i bez wyjaśnień ze strony twórcy programu nie będzie umiał tego żądania zinterpretować. Problem ma charakter generalny: każdorazowe stosowanie instrukcji INPUT powoduje, że maszyna oczekuje od użytkownika podania pewnych danych o czym powinien być on poinformowany. Dlatego do zasad poprawnego programowania w języku BASIC należy poprzedzanie instrukcji

INPUT instrukcją PRINT, zawierającą tekst wyjaśniający, czego maszyna żąda. Oto przykład tak skonstruowanego programu:

```
100 PRINT "PODAJ PROMIEN KOŁA"
110 INPUT R
120 PRINT "POWIERZCHNIA WYNOSI"; @PI * R ↑ 2
130 END
```

6.3. Instrukcje sterujące

Omówiony wyżej zestaw instrukcji pozwala na programowanie dowolnie złożonych obliczeń, wprowadzanie danych i drukowanie wyników. Jednak poważnym ograniczeniem możliwości programisty jest brak możliwości zmieniania kolejności wykonywania instrukcji. Istotnie, każdy z omawianych dotychczas programów wykonywany był przez maszynę w ten sposób, że każda instrukcja wykonywana była dokładnie jeden raz w kolejności wynikającej z jej numeru wiersza. Zmiana omówionego trybu pracy możliwa jest w przypadku wykorzystania instrukcji sterujących. Najprostszą jest skok bezwarunkowy:

⟨numer wiersza⟩ GO TO ⟨numer wiersza, który ma być wykonany⟩

Instrukcja tej postaci powoduje, że jako kolejny wykonany będzie wiersz wymieniony po słowach GO TO, a nie kolejny w sensie sekwencji numerów. Oto przykład wykorzystania:

```
110 PRINT "R="
120 INPUT R
130 PRINT "S= "; @PI * R ↑ 2
140 GO TO 110
150 END
```

Nowa wersja programu pozwala policzyć powierzchnię dowolnie wielu kół, gdyż program po wykonaniu obliczeń i wydrukowaniu wyniku każdorazowo wraca na początek i ponawia pytanie o promień. Program został uformowany w postaci tzw. pętli, to znaczy wielokrotnie, cyklicznie wykonuje te same instrukcje (za każdym razem z innymi danymi). Pętla jest bardzo ważną techniką programowania, gdyż pozwala łatwo programować

wszelkie iteracyjne obliczenia, jednak postać pętli uzyskiwana z użycia instrukcji GO TO ma wadę: jest to pętla która nigdy się nie skończy i programista będzie musiał przerywać działanie programu specjalnym sygnałem z klawiatury.

Jednym z możliwych sposobów zaradzania tej niedogodności jest wykorzystanie instrukcji skoku warunkowego. Instrukcja ta ma postać:

⟨numer wiersza⟩ IF ⟨warunek⟩ THEN ⟨numer wiersza docelowego⟩

Instrukcja ta powoduje, że jako następny będzie wykonany wiersz o podanym po słowie THEN numerze docelowym (będzie wykonany skok), jednak jedynie wtedy, gdy spełniony będzie umieszczony po słowie IF warunek. W przeciwnym przypadku (to znaczy przy nie spełnieniu warunku) instrukcja nie powoduje żadnego skutku - skok nie zostaje wykonany i wykonywana jest instrukcja sekwencyjnie następna w stosunku do instrukcji IF. Kluczowym pojęciem w omawianiu instrukcji warunkowej (skoku warunkowego) jest pojęcie warunku. Tworzą go dwa wyrażenia pomiędzy którymi ustalono interesującą relację za pomocą znaków "=", ">", "<" oraz ich kombinacji (na przykład "większy lub równy" można zapisać jako >= lub =>, zaś "nie równy" zapisuje się >< lub <>). Możliwość użycia przy konstrukcji warunku dowolnych wyrażeń daje programiście bardzo ciekawe możliwości, wymagające jednak dużej wprawy i wyobraźni. Oto przykład zastosowania instrukcji IF:

```
110 PRINT "R="
120 INPUT R
130 IF R  = 0 THEN 200
140 PRINT "S= "; @ PI * R↑2
150 GO TO 110
200 PRINT "KONIEC OBLICZEN"
210 END
```

Nowa wersja programu daje programiście możliwość wyjścia z pętli poprzez podanie nonsensownej danej i zabezpiecza przed prowadzeniem obliczeń dla niewłaściwych danych.

Kolejnym udoskonaleniem poznanych instrukcji sterujących jest instrukacja skoku liczonego. Umożliwia ona wskazanie kilku alternatyw-

nych numerów wierszy, do których wykonany będzie skok - w zależności od wartości wyrażenia wchodzącego w skład instrukcji. Postać instrukcji jest następująca:

⟨nr.wiersza⟩ ON ⟨wyrażenie⟩GO TO ⟨nr.wiersza1 , nr.wiersza2 ,...⟩

Jeśli wyrażenie występujące po słowie ON przyjmie wartość 1, nastąpi skok do wiersza, którego numer wymieniono na pierwszym miejscu listy po słowach GO TO, gdy wyrażenie przyjmie wartość 2 - skok nastąpi do wiersza o numerze n.wiersza2 itd. W przypadku kiedy wartość wyrażenia jest mniejsza od 1 lub większa od ilości wymienionych numerów wierszy - sygnalizowany jest błąd. Oto przykład programu wykorzystującego instrukcję ON:

```
100 PRINT "PROGRAM WYKONUJE NASTEPUJACE OBLICZENIA:"
110 PRINT ,,"1 - POWIERZCHNI KOŁA"
120 PRINT,, "2 - OBWODU KOŁA"
130 PRINT ,,"3 - CIECIWY ŁUKU KOŁA"
140 PRINT ,,"4 - STRZAŁKI ŁUKU KOŁA"
150 PRINT "PODAJ NUMER POTRZEBNYCH AKTUALNIE OBLICZEN"
160 INPUT K
170 PRINT "PODAJ PROMIEN KOŁA"
180 INPUT R
190 ON K GO TO 200, 300, 400, 500
200 PRINT "POWIERZCHNIA = "; @ PI * R↑2
210 GO TO 600
300 PRINT "OBWOD = "; 2 * @ PI * R
310 GO TO 600
400 PRINT "PODAJ KAT SRODKOWY (W RADIANACH)"
410 INPUT A
420 PRINT "CIECIWA = "; 2 * R * SIN(0.5 * A)
430 GO TO 600
500 PRINT "PODAJ CIĘCIWE ŁUKU"
510 INPUT H
520 PRINT "STRZAŁKA WYNOSI "; R - SQR(R↑2 - 0.25 *ˆH↑2)
600 END
```

W rozważanym wyżej programie wielokrotnie stosowano instrukcję skoku do wiersza END celem zakończenia działania poszczególnych wariantów programu. Zamiast tego można było przerywać obliczenia wierszem STOP

o ogólnej postaci

⟨numer wiersza⟩ STOP

Użycie instrukcji STOP ma w przypadkach podobnych do rozważanego liczne zalety. Po pierwsze program staje się bardziej zrozumiały i czytelny, po drugie w trakcie obliczeń otrzymuje się informację o tym, która z licznych instrukcji STOP zadziałała. Zatrzymaniu programu towarzyszy bowiem komunikat postaci

FINISH ⟨numer wiersza⟩

gdzie numer wiersza odpowiada instrukcji przerywającej pracę programu. W podanym wyżej programie komunikat zawsze będzie wskazywał na wiersz o numerze 600, natomiast zastępując odpowiednio skoki GO TO 600 instrukcjami STOP możemy mieć komunikaty o zatrzymaniu w wierszach 210, 310, 430 lub 600, zależnie od wersji obliczeń. Udogodnienie takie pozwala śledzić i wykrywać niektóre błędy przy uruchamianiu programów.

6.4. Instrukcja pętli

Pojęcie pętli było już wcześniej wprowadzone a instrukcja organizująca pętlę jest w istocie instrukcją sterującą jak GO TO, IF czy ON. Fakt jej wyodrębnienia wynika ze znaczenia pętli jako nadzwyczaj przydatnej konstrukcji programowej oraz większej (w stosunku do poznanych już konstrukcji) złożoności instrukcji pętli.

Instrukcja pętli nakazuje wielokrotne wykonywanie zespołu instrukcji poczynając od instrukcji następującej bezpośrednio (w sensie numeracji wierszy) po instrukcji pętli a kończąc na instrukcji domykającej pętlę, zawierającej słowo kluczowe NEXT. W trakcie wielokrotnego wykonywania zawartych w pętli instrukcji pewna wyróżniona (wymieniona w instrukcji pętli) zmienna przyjmuje kolejno wartości z zadanego przedziału z zadanym krokiem. Zmienna ta może "zliczać" kolejne obiegi pętli powodując jej zakończenie po osiągnięciu założonej liczby wykonań, może także być wykorzystywana do bardziej wyrafinowanych celów. Zadanie instrukcji

pętli polega w głównej mierze na zdefiniowaniu wyróżnionej zmiennej i określeniu sposobu jej zmian przy kolejnych wykonaniach pętli. Postać tej instrukcji jest następująca:

⟨numer wiersza⟩ FOR ⟨zmienna⟩ = ⟨początek⟩ TO ⟨koniec⟩ STEP ⟨przyrost⟩

Parametry nazwane wyżej "początek", "koniec" i "przyrost" są dowolnymi wyrażeniami, których wartości określają odpowiednio: początkową wartość wskazanej zmiennej (taką wartość ma zmienna przy pierwszym obiegu pętli), końcowej wartość (jej osiągnięcie lub przekroczenie powoduje zakończenie wykonywania pętli) i wartość o którą zwiększana jest wartość zmiennej za każdym kolejnym obiegiem pętli. Końcową instrukcją pętli, wskazującą jej zakres w ciele programu jest instrukcja NEXT postaci:

⟨numer wiersza⟩ NEXT ⟨zmienna⟩

przy czym zmienna musi być tą samą, co wyróżniona w instrukcji FOR otwierającej pętlę. Oto przykładowe zastosowanie instrukcji pętli:

```
100 PRINT "PROGRAM DRUKUJE TABELE WARTOSCI POWIERZCHNI KOŁA"
110 PRINT "PODAJ ZAKRES ZMIENNOSCI PROMIENIA DLA KTOREGO MA"
120 PRINT "BYC PROWADZONA TABELARYZACJA ORAZ PRZYROST R"
130 INPUT P,K,D
140 FOR R = P TO K STEP D
150 PRINT "R=";R;" S= "; @ PI ≡ R↑2
160 NEXT R
170 END
```

Dla dokładniejszego zrozumienia sposobu wykonywania pętli podamy niżej wersję programu z pętlą zbudowaną w oparciu o wcześniej poznane instrukcje skoku, a bez użycia instrukcji FOR. Oba programy wykonywać się będą identycznie:

```
130 INPUT P,K,D
140 R0 = P
150 R1 = K
160 R3 = D
170 IF (R0 - R1) ≡ R2 > 0 THEN 300
180 R = R0
190 PRINT "R=";R;" S= ";@ PI ≡ R↑2
200 R0 = R + R2
210 GO TO 170
300 END
```

Pętle mogą być pogrążane jedna wewnątrz drugiej, nie mogą się natomiast przecinać. Przyrost zmiennej w pętli (parametr podawany po słowie kluczowym STEP) najczęściej wynosi 1 ("pętla z licznikiem") i wówczas można pomijać w instrukcji FOR podawanie tej wartości oraz słowa STEP.

6.5. Podprogramy

Przy pisaniu programów w licznych językach algorytmicznych chętnie stosowane są podprogramy, które mogą być wydzielone z tekstu programu głównego (jak na przykład w FORTRANIE) lub mogą być w nim zawarte (w Algolu i językach na nim wzorowanych). W języku BASIC jako podprogram może być wykorzystany dowolny fragment napisanego programu, rozpoczynający się wierszem wskazanym w instrukcji wywołania podprogramu (patrz dalej) a zakończony wierszem powrotu o postaci

⟨numer wiersza⟩ RETURN

Wywołanie podprogramu polega na użyciu instrukcji GOSUB ⟨numer wiersza⟩, przy czym instrukcja ta może występować samodzielnie i wtedy działa podobnie jak instrukcja skoku bezwarunkowego, ale może także być wbudowana do instrukcji warunkowej IF lub ON - w instrukcjach tych wystarczy zmienić słowo THEN lub GO TO na GOSUB aby otrzymać w miejscu skoku wywołanie podprogramu. Wykonanie podprogramu polega na tym, że zapamiętany jest wiersz następny w stosunku do wiersza z instrukcji GOSUB a następnie wykonywany jest skok do wiersza o numerze wskazanym w GOSUB. Napotkanie instrukcji RETURN powoduje powrotny skok do zapamiętanego miejsca, a więc bezpośrednio za wywołaniem podprogramu. Oto przykład:

```
110 PRINT "PODAJ PROMIEN"
120 INPUT R
130 IF R>0 GOSUB 200
140 STOP
200 PRINT "S = "; @ PI * R↑2
210 RETURN
220 END
```

Specyficznym rodzajem podprogramu jest funkcja. Wspomniano na początku podrozdziału 6.2, że programista może budować własne funkcje,

używane potem w wyrażeniach; podano także reguły budowy nazw tych funkcji, które muszą być trzyliterowe i zaczynać się od FN. Sposób definicji takiej funkcji w ogólnym przypadku jest następujący:
⟨numer wiersza⟩ DEF FN ⟨litera⟩(⟨argument⟩) = ⟨wyrażenie definiujące⟩
Jak widać możliwe jest definiowanie funkcji niezbyt złożonych (wyrażenie definiujące musi mieścić się w jednym wierszu) i przewidujących użycie tylko jednego argumentu. Instrukcja DEF może być umieszczona w dowolnym miejscu programu i jest pomijana przy normalnym sekwencyjnym wykonywaniu instrukcji. Litera użyta jako argument w instrukcji DEF może być używana w wyrażeniu definiującym, nie ma ona jednak nic wspólnego ze zmienną o takiej samej nazwie, występującej (ewentualnie) w programie. Oto przykładowy program wykorzystujący definicję funkcji:

```
110 PRINT "R="
120 INPUT R
130 PRINT "S = "; FNS(R)
140 DEF FNS(X) = @ PI ≡ X↑2
150 END
```

6.6. Operacje macierzowe

W języku BASIC, podobnie, jak w innych językach algorytmicznych, możliwe jest używanie zmiennych indeksowych, przy czym dostępne są jedynie zmienne z jednym lub najwyżej z dwoma indeksami. Nazwa zmiennej indeksowej musi składać się jedynie z litery, możliwe jest więc używanie najwyżej 27 zmiennych indeksowych (tablic). Deklarowanie rozmiarów dla tablic odbywa się w instrukcji DIM o postaci
⟨numer wiersza⟩ DIM ⟨nazwa⟩ (⟨granica indeksu⟩),⟨nazwa⟩ (⟨granica ⟩),..
Deklarując rozmiary tablic dwuindeksowych trzeba podać oczywiście nie jedną, lecz dwie granice. Instrukcja DIM może występować w dowolnym punkcie programu i jest traktowana jako niewykonywalna. Zmienne indeksowe mogą być wykorzystywane w większości tych zastosowań, w których wolno użyć zmiennej prostej. Wyjątki to zmienne wyróżnione w pętlach

i argumenty funkcji w instrukcji DEF. Oto przykład wykorzystania zmiennych indeksowych:

```
100 PRINT "PROGRAM OBLICZA SREDNIA I ODCH.STAND.DLA N LICZB"
110 PRINT "PODAJ N (MAKSYMALNIE 100)"
120 INPUT N
130 DIM X(100)
140 PRINT "TERAZ PODAWAJ KOLEJNE LICZBY"
150 FOR I = 1 TO N
160 INPUT X(I)
170 NEXT I
180 S = 0
190 FOR I = 1 TO N
200 S = S + X(I)
210 NEXT I
220 S = S/N
230 K = 0
240 FOR I = 1 TO N
250 K = K + (X(I) - S)↑2
260 NEXT I
270 K = SQR(K/N-1)
280 PRINT "SREDNIA= "; S;" ODCHYLENIE STANDARDOWE= ";K
290 END
```

Przytoczony wyżej program nie jest optymalny z punktu widzenia organizacji zdefiniowanych w nim obliczeń (w istocie nie zachodziła konieczność posługiwania się tablicą, gdyż można było uśredniać dane na bieżąco), natomiast ilustruje sposób posługiwania się zmiennymi indeksowymi oraz uwidacznia typowe związki, jakie zwykle zachodzą pomiędzy używaniem tablic i stosowaniem instrukcji pętli.

Osobliwością języka BASIC jest możliwość traktowania tablic (zmiennych indeksowanych) jako macierzy i wykonywania na nich operacji zgodnie z regułami rachunku macierzowego. Służy do tego grupa operacji z przedrostkiem MAT. Zakładając, że A,B i C są macierzami (tablicami z dwoma indeksami), zaś U, V i W są wektorami (tablicami z jednym indeksem) możemy zapisać następujące operacje macierzowe:

a) mnożenie przez skalar (który może być dowolnym wyrażeniem zamkniętym w nawiasy):

```
100 MAT A = (Q) * B
```

b) mnożenie wektora przez wektor (iloczyn skalarny)

```
110 MAT Q = U * V
```

c) mnożenie wektora przez macierz, macierzy przez wektor i macierzy przez macierz

```
120 MAT U = V * A

130 MAT W = B * U

140 MAT C = A * B
```

d) dodawanie i odejmowanie wektorów i macierzy

```
150 MAT A = B - C
```

e) transpozycja macierzy

```
160 MAT A = TRN(B)
```

f) odwracanie macierzy

```
170 MAT B = INV(C)
```

g) wypełnianie macierzy zerami

```
180 MAT A = ZER
```

h) wypełnianie macierzy jedynkami

```
190 MAT B = CON
```

i) wypełnianie macierzy do postaci macierzy jednostkowej

```
200 MAT C = IDN
```

W omawianych wyżej działaniach wymaga się, aby spełnione były warunki zgodności wymiarów odpowiednich argumentów (macierzy i wektorów) zgodnie z zasadami rachunku macierzowego. Macierz wynikowa może mieć inne wymiary, niż deklarowane dla macierzy stojącej po lewej stronie znaku "=". Jeśli wynikowe wymiary zawierają się w deklarowanych (są od nich mniejsze) wówczas następuje automatyczna zmiana wymiarów macierzy. Liczbę wierszy i kolumn przewymiarowej macierzy można uzyskać za pomocą funkcji ROW i COL (patrz podrozdział 6.2). Reguły zmiany wymiarów są w ogólnym przypadku dość złożone i nie będą tu z braku miejsca rozważane. W praktycznych przypadkach programista dba o zgodność wymiarów poszczególnych macierzy z potrzebami i do zmiany wymiarów nie dochodzi. Macierze można czytać instrukcją o postaci:

⟨numer wiersza⟩ MAT INPUT ⟨lista nazw macierzy⟩

oraz drukować instrukcją postaci

⟨numer wiersza⟩ MAT PRINT ⟨lista nazw macierzy⟩

Macierze wczytywane i drukowane są wierszami, przy czym w przypadku druku rozmieszczenie elementów każdej macierzy ("gęste" lub w strefach po 15 znaków na strefę) zależy od separatora napisanego po nazwie danej macierzy (";" determinuje wydruk gęsty, "," wydruk w strefach). Brak separatora po nazwie wektora powoduje wydruk kolejnych jego elementów w oddzielnych wierszach (wektor kolumnowy). Możliwa jest zmiana wymiarów macierzy przy czytaniu; nazwę macierzy podaje się wówczas w liście instrukcji MAT INPUT wraz z wartościami granicznych indeksów i wartości wprowadzane są tylko do takiego na bieżąco zawężonego podobszaru macierzy, niezależnie od jej generalnych rozmiarów.

Oto przykłady programu wykorzystującego operacje macierzowe:

```
100 PRINT "PROGRAM ROZWIAZUJE UKŁAD N RÓWNAŃ LINIOWYCH"
110 DIM A(10,10), B(10), X(10)
120 PRINT "PODAJ LICZBE ROWNAN (MAKSYMALNIE 10)"
130 INPUT N
140 "PODAWAJ KOLEJNO WSPOŁCZYNNIKI ROWNAN"
150 MAT INPUT A(N,N)
160 "PODAJ KOLEJNO WARTOSCI PRAWYCH STRON ROWNAN
170 MAT INPUT B(N)
180 MAT A = INV(A)
190 IF DET = 0 THEN 300
200 MAT X = A * B
210 PRINT "WYNIKI:"
220 FOR I = 1 TO ROW
230 PRINT X(I),
240 NEXT I
250 STOP
300 PRINT "MACIERZ OSOBLIWA - BRAK ROZWIAZANIA
310 END
```

6.7. Uzupełniające wiadomości na temat operacji czytania i pisania

Do czytania danych używano dotychczas instrukcji INPUT, jako eksponującej konwersacyjne (bardzo wygodne z użytkowego punktu widzenia) własności języka BASIC. Dane mogą być jednak czytane przy pomocy instrukcji o ogólnej postaci

⟨numer wiersza⟩ READ ⟨lista zmiennych dla których czytamy wartości⟩

Instrukcja taka powoduje czytania (i umieszczanie w kolejnych zmiennych) wartości umieszczonych w treści samego programu za pomocą instrukcji o ogólnej postaci

⟨numer wiersza⟩ DATA ⟨lista wartości które podlegają czytaniu⟩

W programie instrukcja DATA może być umieszczona w dowolnym miejscu (zarówno po instrukcji READ, jak i przed nią), a także możliwe jest "składanie" listy danych przez wypisanie kilku instrukcji DATA. Po przeczytaniu każdej kolejnej danej udostępniana jest dana następna na liście instrukcji DATA, jednak można użyć instrukcji ogólnej postaci

⟨numer wiersza⟩ RESTORE

w celu "przewinięcia" danych - dostępna staje się wówczas ponownie pierwsza dana z listy, a za nią kolejne dalsze.

Instrukcja READ ma swój odpowiednik macierzowy w postaci

⟨numer wiersza⟩ MAT READ ⟨lista nazw macierzy⟩

Drukując wyniki obliczeń możemy je formatować używając wzorca. Instrukcja druku musi wówczas odwoływać się do tego wzorca w postaci:

⟨numer wiersza⟩ PRINT USING ⟨numer wiersza wzorca:⟩: ⟨lista wartości⟩

Wiersz wzorca ma następującą postać:

⟨numer wiersza⟩ : ⟨ciąg znaków stanowiących wzorzec wydruku⟩

We wzorcu wydruku można używać następujących znaków "#" (miejsce dla wydrukowania cyfry), "-" (miejsce dla wydrukowania znaku -), "+" (miejsce dla wydrukowania znaku + lub - zależnie od drukowanej wartości), "." (oznacza pozycję dla wydrukowania kropki dziesiętnej), "↑↑↑↑" (oznacza pole - pięcioznakowe - dla wydrukowania wykładnika liczby).

Inne znaki niż wymienione wyżej (między innymi odstępy " ") są traktowane jako nie znaczące i są kopiowane z wzorca na wydruk. Natomiast na pozycjach oznaczonych symbolami "#", "." oraz ewentualnie "+", "-" i "↑↑↑↑↑" drukowane są wartości wymienione w instrukcji PRINT USING, przy czym postać wyniku ściśle odpowiada podanemu wzorcowi. Jeśli nie używa się pola wykładnika "↑↑↑↑↑" to możliwa jest sytuacja, w której ilość miejsc na wydruku (oznaczonych przez "#") nie wystarczy do wydrukowania uzyskanej w wyniku obliczeń wartości. Całe pole odpowiadające wydrukowi takiej wartości zostaje wypełnione gwiazdkami.

6.8. Komentarze

Programista piszący program w dowolnym języku powinien dbać o jego czytelność, będącą warunkiem łatwego używania programu - zarówno przez innych użytkowników, jak i przez samego twórcę. Dlatego zalecane jest a nawet można to uznać za kategoryczny nakaz, aby program zaopatrywać w odpowiednią liczbę komentarzy, które bezużyteczne dla komputera i ignorowane w trakcie realizacji programu, znakomicie jednak ułatwiają jego interpretację przez człowieka. W języku BASIC możliwe jest wstawienie w dowolnym punkcie wiersza komentarza o postaci:

⟨numer wiersza⟩ REM ⟨dowolny tekst objaśniający⟩

Na ogół przyjmuje się, że komentarz powinien znajdować się na początku programu (wyjaśniając, jakie jest jego przeznaczenie, kto i po co go napisał, jakie są ograniczenia stosowalności itd.), a następnie w każdym istotnym z punktu widzenia logiki programu punkcie, nie rzadziej jednak, niż w proporcji 1 wiersz komentarza na 10 wierszy instrukcji.

6.9. Zlecenie systemu BASIC w komputerze MERA-400

Programista pracujący z wykorzystaniem języka BASIC ma możliwość wydawania maszynie szeregu poleceń, dotyczących konstruowanego programu. Oto niektóre z nich.

Rozpoczynając pracę należy usunąć z maszyny ewentualne pozostałości

działań innych programistów. Dokonuje się tego zleceniem postaci

NEW

W przypadku, kiedy chcemy sprawdzić, jak komputer zapamiętał nasz program, można posłużyć się zleceniem

LIST

po którym komputer wydrukuje obraz wszystkich zapamiętanych wierszy programu. Można też zażądać wylistowania fragmentu programu pisząc

LIST ⟨numer wiersza początku⟩ - ⟨numer wiersza końca⟩

Pominięcie numeru wiersza początku spowoduje listowanie od początku programu do wskazanego wiersza końca, pominięcie wiersza końca - spowoduje listowanie od wskazanego wiersza początku do końca programu. Należy zwrócić uwagę na rolę łącznika "-", gdyż jego pominięcie nie pozwala zinterpretować, czy podany numer wiersza jest początkowy, czy końcowy i maszyna listuje tylko jeden wskazany wiersz. Listowanie może być dokonane na innym urządzeniu, niż końcówka z której korzysta programista, co między innymi pozwala łatwo wytwarzać kopie potrzebnego programu na taśmie perforowanej. Takie listowanie zewnętrzne wykonywane jest na polecenie

LIST # ⟨numer⟩

gdzie numer odpowiada 1 - drukarce, 2 - perforatorowi taśmy, 3 - drukarce z poszerzonym zapisem (150 znaków w wierszu).

Program zeskładowany na taśmie papierowej można wprowadzić do maszyny zleceniem

LOAD # ⟨numer⟩

gdzie "numer" jest numerem czytnika taśmiki (1, 2 lub 3).
Zlecenia systemu BASIC umożliwiają wprowadzanie poprawek w programie. Przykładowo zlecenie

REJECT ⟨numer wiersza początku⟩ - ⟨numer wiersza końca⟩

pozwala usunąć wskazany fragment programu (obowiązują tu omówione przy dyskusji zlecenia LIST uproszczone formy zapisu zakresu w przypadku obejmowania usuwaniem początku lub końca programu). Zlecenie

RESEQUENCE ⟨stary numer początkowy⟩,⟨nowy numer początkowy⟩,⟨przyrost⟩ pozwala przenumerować wiersze w całym programie lub jego fragmencie (przy czym logicznie przenumerowane są też numery w instrukcjach skoków). Pominięcie parametru "stary numer początkowy" oznacza, że przenumerowywany będzie cały program od początkowego wiersza (jakikolwiek miałby początkowo numer), pominięcie parametru "nowy numer początkowy" spowoduje przyjęcie numeru 100, pominięcie parametru "przyrost" (czyli skok numeracji od wiersza do wiersza) spowoduje przyjęcie wartości 10.

Zredagowany i opracowany program trzeba uruchomić. Służy do tego zlecenie

RUN

po którym maszyna zaczyna wykonywać napisany program. Niekiedy w celach testowych chcemy spowodować wykonanie jedynie fragmentu programu, wówczas można posłużyć się wersją

RUN ⟨numer wiersza początku⟩ - ⟨numer wiersza końca⟩

Programista piszący program w języku BASIC ma możliwość skorzystać w każdej chwili z komputera jak z kalkulatora. Służy do tego zlecenie

PRINT ⟨wyrażenie⟩

gdzie wyrażenie jest konstruowane według reguł obowiązujących w BASICu. Jeśli w wyrażeniu użyć zmiennych to będą one reprezentowały wartości, jakie im nadawano w ostatnio wykonywanym programie.

Istnieje jeszcze obszerna grupa zleceń związanych z manipulacjami na dyskach magnetycznych, pominięto jednak ich opis, gdyż nie w każdej konfiguracji komputera mogą być używane.

7. PRZYKŁADY PRACY W SYSTEMIE MERA-400

W niniejszym rozdziale przedstawiono przykłady eksploatacji systemu minikomputerowego MERA-400. Poszczególne przykłady prezentują:

a) uruchamianie programów w języku BASIC

Po wywołaniu procesora BAS dyrektywami JOB CONTROL'a ($JOB i $EXE) interpreter żąda od operatora zdefiniowania nazw strumieni (urządzeń) wejścia i wyjścia, zgłaszając pytania ?INPUT i następnie ?OUTPUT. Po każdym pytaniu należy wprowadzić z klawiatury nazwy odpowiednich strumieni (w przykładzie kolejno CI oraz CO). Prawidłowo wywołanie procesora BAS powoduje wydruk w nowej linii znaku gwiazdki * co świadczy o oczekiwaniu na przyjmowanie zleceń i instrukcji programu.

Przykładowe programy wprowadzane były bezpośrednio z klawiatury konsoli operatorskiej. Po wykonaniu programu pierwszego został on wymazany z pamięci operacyjnej.

b) praca z procesorem EDM

Po wywołaniu procesora EDM (dyrektywy $JOB i $EXE EDM) wprowadzono z klawiatury (dyrektywa !ASS SI CK procesora edytora tekstowego) źródłowy program fortranowski. Program ten celowo zawierał kilka błędów, które następnie usunięto wykorzystując do tego celu instrukcje procesora EDM (w przykładzie instrukcje I, A, S). Poprawiony tekst programu zapisano na sekcję AM1 dysku wymiennego.

c) uruchamianie programów w języku FORTRAN

W przykładzie tym wykorzystano tekst źródłowy programu wygene-

rowanego w przykładzie poprzednim. W celu uzyskania programu wynikowego (możliwego do wykonania) program źródłowy skompilowano, a następnie dołączono. W procesie tym korzystano z dyrektyw JOB CONTROL'a i procesorów systemowych FOR, MAC i EDI.

Program źródłowy znajdował się na sekcji AM1 dysku wymiennego, natomiast program wynikowy umieszczony został na sekcji ASA (strumień BO) dysku systemowego. Dyrektywy wprowadzane były przez operatora bezpośrednio z klawiatury konsoli operatorskiej. W procesie kompilacji tekst programu został wylistowany (wraz z nazwami zmiennych prostych i nazwami tablic wykorzystanych w programie).

Uruchomienie programu wynikowego o nazwie DODMAC poprzedzone zostało przywiązaniem strumienia 5 do urządzenia drukującego konsoli operatorskiej (strumień ten wykorzystywany jest w instrukcji WRITE skompilowanego programu).

Przykładowy program tworzy sumę dwóch macierzy i drukuje macierz wynikową.

d) katalogowanie programów

W przykładzie pokazano sposób katalogowania fortranowskich programów wynikowych (program wytworzony w przykładzie poprzednim). W tym celu wykorzystano możliwości procesora CAT. Program przeznaczony do zakatalogowania znajdował się na sekcji ASA dysku systemowego. Po zakatalogowaniu zapisany został pod nazwą DMC na sekcję AL1 dysku wymiennego.

e) uruchamianie zakatalogowanych programów fortranowskich

W przykładzie przedstawiono sposób wykorzystania do obliczeń programów fortranowskich umieszczonych w bibliotece. Biblioteka znajdowała się na sekcji AL1 dysku wymiennego. Wywołany i uruchomiony program jest programem utworzonym w przykładzie c.

f) tworzenie własnych procedur

W celu utworzenia procedury o nazwie FORTR, której użycie powoduje skompilowanie i docalenie źródłowego programu fortranowskiego

wykorzystano procesor EDM. Tekst procedury umieszczono na strumieniu JC (sekcja AJC dysku systemowego).

g) uruchamianie programów z wykorzystaniem własnych procedur

W przykładzie pokazano sposób uruchamiania programów fortranowskich z wykorzystaniem procedury utworzonej w przykładzie poprzednim. Wywołanie tej procedury dyrektywą JOB CONTROL'a ($DO FORTR ...) powoduje wytworzenie programu wynikowego (na podstawie programu źródłowego wprowadzanego z klawiatury) i jego uruchomienie. Wykorzystana procedura zastępuje konieczność wprowadzania ciągu dyrektyw z przykładu c.

```
URUCHAMIANIE PROGRAMOW W JEZYKU BASIC.

'$JOB
      $JOB
!JOB(JOB)$JOB
'$EXE BAS
      $EXE BAS

                BASIC     MERA-400



STANDARD VERSION
STORE =  20 [ KWORDS ]
USER'S STORAGE = 9771 [ WORDS ]
USERS =  1
INPUT TERMINALS =TC1
OUTPUT TERMINALS = TC1
MATRIX = YES
?INPUT CI
?OUTPUT CO
?END
STORE =  20 [ KWORDS ]
USER'S STORAGE = 9771 [ WORDS ]
USERS =  1
INPUT TERMINALS =CI
OUTPUT TERMINALS = CO
MATRIX = YES

                BASIC     MERA-400
*10 REM "SUMA DWOCH MACIERZY"
*20 DIM A(10,10),B(10,10),X(10,10)
*30 INPUT N,M
*40 MAT INPUT A(N,M)
*50 MAT INPUT B(N,M)
*60 MAT X=A+B
*70 PRINT "SUMA MACIERZY:"
*80 MAT PRINT X
*90 END
```

```
*RUN
RUNNING  1213
?2,3
?1,2,3
?7,8,9
?1,1,1
?1,1,1
SUMA MACIERZY:

 2              3              4
 8              9              10
FINISH 90
*NEW
*LIST

*10 REM "PROGRAM TEST"
*20 INPUT A,B
*30 C=A+B
*40 PRINT "LICZBY:",A,B,C
*50 END
*RUN
RUNNING  71
?2,3
LICZBY:        2              3              5
FINISH 50
*
```

PRACA Z PROCESOREM EDM.

```
'$JOB
      $JOB
!JOB(JOB)$JOB
'$EXE EDM
      $EXE EDM
!ASS SI CK
E


      PROGRAM DODMAC
      READ(1,9) N,M
      DO 100 I=1,N
      READ(1,0)(A(I,J),J=1,M)
      READ(1,0)(B(I,J),J=1,M)
 100  COMTINE
      DO 200 I=1,N
      DO 200 J=1,M
 200  X(I,J)=A(I,J)+B(I,J)
      WRITE(5,52)
      DO 300 I=1,N
 300  WRITE(5,53)(X(I,J),J=1,M)
STOP

 51   FORMAT(/1X,"PODAJ DANE:",/1X)
 52   FORMAT(/1X,"WYDRUK SUMY MACIERZY:")
 55   FORMAT(/1X,10(F8.2,2X))
$$
00000*512
2S/9/0/
6S/COMTIN/CONTINU/
1,$S/53/55/
2I
```

```
      DIMENSION A(10,10),B(10,10),C(10,10)
      WRITE(5,51)
.
$A
      END
.
2S/C/X/
1,$L!
00001=        PROGRAM DODMAC
00002=        DIMENSION A(10,10),B(10,10),X(10,10)
00003=        WRITE(5,51)
00004=        READ(1,0) N,M
00005=        DO 100 I=1,N
00006=        READ(1,0)(A(I,J),J=1,M)
00007=        READ(1,0)(B(I,J),J=1,M)
00008=  100   CONTINUE
00009=        DO 200 I=1,N
00010=        DO 200 J=1,M
00011=  200   X(I,J)=A(I,J)+B(I,J)
00012=        WRITE(5,52)
00013=        DO 300 I=1,N
00014=  300   WRITE(5,55)(X(I,J),J=1,M)
00015= STOP
00016=   51   FORMAT(/1X,"PODAJ DANE:",/1X)
00017=   52   FORMAT(/1X,"WYDRUK SUMY MACIERZY:")
00018=   55   FORMAT(/1X,10(F8.2,2X))
00019=        END
1,$S/STOP/      STOP/
!ASS SO AM1
W
00001*512
Q
'$JOB
      $JOB
!JOB(JOB)$JOB
```

URUCHAMIANIE PROGRAMOW W JEZYKU FORTRAN.

```
'$JOB
      $JOB
!JOB(JOB)$JOB
'$ASS SI AM1
      $ASS SI AM1
'$EXE FOR
      $EXE FOR
```

```
FORTRAN IV-S                        20. 11. 1978            STRONA

     1        1        PROGRAM DODMAC
     2        2        DIMENSION A(10,10),B(10,10),X(10,10)
     3        3        WRITE(5,51)
     4        4        READ(1,0) N,M
     5        5        DO 100 I=1,N
     6        6        READ(1,0)(A(I,J),J=1,M)
     7        7        READ(1,0)(B(I,J),J=1,M)
     8        8   100  CONTINUE
```

```
 9        9          DO 200 I=1,N
10       10          DO 200 J=1,M
11       11    200   X(I,J)=A(I,J)+B(I,J)
12       12          WRITE(5,52)
13       13          DO 300 I=1,N
14       14    300   WRITE(5,55)(X(I,J),J=1,M)
15       15          STOP
16       16    51    FORMAT(/1X,"PODAJ DANE:",/1X)
17       17    52    FORMAT(/1X,"WYDRUK SUMY MACIERZY:")
18       18    55    FORMAT(/1X,10(F8.2,2X))
19       19          END
```

```
          INTEGER   NAME     SPECIFICATION   LENGTH
                    I           VARIABLE      1
                    J           VARIABLE      1
                    M           VARIABLE      1
                    N           VARIABLE      1

          REAL      NAME     SPECIFICATION   LENGTH
                    A           ARRAY         300
                    B           ARRAY         300
                    X           ARRAY         300
!JOB(FT4)DODMAC KONIEC TRANSLACJI
'$WEO SO
      $WEO SO
'$ASS SI SO
      $ASS SI SO
'$REW SI
      $REW SI
'$EXE MAC,,NOLO
      $EXE MAC,,NOLO
!JOB(Z1 )DODMAC -KONIEC PRZEBIEGU 1
!JOB(Z2 )KONIEC TRANSLACJI
'$WEO BO
      $WEO BO
'$JOB
      $JOB
!JOB(JOB)$JOB
'$EXE EDI,,NOMA,NOLO
      $EXE EDI,,NOMA,NOLO
LINK BI
EXI
```

URUCHAMIANIE PROGRAMOW W JEZYKU FORTRAN.

```
'$JOB
      $JOB
!JOB(JOB)$JOB
'$ASS 5 CS
      $ASS 5 CS
'$EXE DODMAC,BI
      $EXE DODMAC,BI

PODAJ DANE:

3,5
1,2,3,4,5,
1,1,1,1,1,
-6,2,3,1,4,
8,8,8,8,8,
12,1,1,3.5,1.5,
-5,-6.43,1,1,0,

WYDRUK SUMY MACIERZY:

    2.00      3.00      4.00      5.00      6.00

    2.00     10.00     11.00      9.00     12.00

    7.00     -5.43      2.00      4.50      1.50
!JOB(DOD)STOP
'
```

KATALOGOWANIE PROGRAMOW.

```
'$JOB
      $JOB
!JOB(JOB)$JOB
'$EXE CAT
      $EXE CAT
ASS STRX AL1
 ASS STRX AL1
INI STRX
 INI STRX
CAT STRX DODMAC DMC 0
 CAT STRX DODMAC DMC 0
LIS STRX
 LIS STRX
NAZWA WEJSCIA       NAZWA PROGRAMU      INTERNAL            EXTERNAL
---------------     ---------------     ----------------    --------
DMC                 DODMAC
EXIT
 EXIT
EXIT
'$JOB
      $JOB
!JOB(JOB)$JOB
'
```

URUCHAMIANIE ZAKATALOGOWANYCH PROGRAMOW FORTRANOWSKICH.

PROGRAM DODMAC ZNAJDUJE SIF W BIBLIOTECE NA SFKCJI AL1
I POSIADA NAZWE DMC.

```
'$JOB
      $JOB
!JOB(JOB)$JOB
'$ASS XX AL1 5 CS
      $ASS XX AL1 5 CS
'$EXE DMC XX
      $EXE DMC XX

PODAJ DANE:

2,3
-5,3,4
0,0,0,
7,5,2,
1,1,1,

WYDRUK SUMY MACIERZY:

   -5.00       3.00       4.00

    8.00       6.00       3.00
!JOB(DMC)STOP
'
```

TWORZENIE WLASNYCH PROCEDUR.

```
'$JOB
      $JOB
!JOB(JOB)$JOB
'$EXE EDM
      $EXE EDM
!ASS SI CK
E

$PRD FORTR,PR,LO,LO,NOMA,NOLO
$JOB
$ASS SI X1
$EXE FOR,,X2
$WEO SO
$ASS SI SO
$REW SI
$EXE MAC,,X3
$WEO BO
$JOB
$EXE EDI,,X4,X5
LINK BI
EXI
$JOB
$$
00000*512
!ASS SO JC
W
00000*512
Q
'
```

URUCHAMIANIE PROGRAMOW Z WYKORZYSTANIEM WLASNYCH PROCEDUR.

```
'$JOB
      $JOB
!JOB(JOB)$JOB
'$DO FORTR,CK,NOLO,NOLO
      $DO FORTR,CK,NOLO,NOLO
      $JOB
!JOB(JOB)$JOB
      $ASS SI CK
      $EXE FOR,,NOLO
      PROGRAM TEST
      READ(1,0) A,B
      C=A+B
      WRITE(5,500) A,B,C
      STOP
 500  FORMAT(/1X,"LICZBY:",3(F9.2,2X))
      END
!JOB(FT4)TEST   KONIEC TRANSLACJI
$$
      $WEO SO
      $ASS SI SO
      $REW SI
      $EXE MAC,,NOLO
!JOB(Z1 )TEST   -KONIEC PRZEBIEGU 1
!JOB(Z2 )KONIEC TRANSLACJI
      $WEO BO
      $JOB
!JOB(JOB)$JOB
      $EXE EDI,,NOMA,NOLO
      $JOB
!JOB(JOB)$JOB
      $ASS CI=CK
'
```

```
'$JOB
      $JOB
!JOB(JOB)$JOB
'$ASS 5 CS
      $ASS 5 CS
'$EXE TEST,BI
      $EXE TEST,BI
2.56,-789.5

LICZBY:     2.56    -789.50    -786.94
!JOB(TES)STOP
'
```

L i t e r a t u r a

1. Bańkowski J., Fiałkowski K., Odrowąż-Stępniewski Z.: Programowanie w języku Fortran. Wersja standardowa ODRA 1300, ICL 1900, CDC 70, PWN, Warszawa 1981.
2. Cyber Line, August 1985, The Monthly Newsletter - Control Data.
3. Czech Z., Nałęcki K., Wołek St.: Programowanie w języku Basic, WNT, Warszawa 1977.
4. DTR-MERA-400, Centrum Naukowo Produkcyjne Technik Komputerowych i pomiarów, Warszawa 1980.
5. Eckhouse R.H. /jr./: Systemy minikomputerowe. Organizacja i oprogramowanie, WNT, Warszawa 1980.
6. EDM, Instytut Informatyki Uniwersytetu Warszawskiego (na prawach rękopisu).
7. Kolbus A., Szyller J.: Mikroprocesory, Wiedza Powszechna, Warszawa 1984.
8. Shaw A.C.: Projektowanie logiczne systemów operacyjnych, WNT, Warszawa 1980.
9. SOM 3.P - Dokumentacja, Uniwersytet Jagielloński, Kraków 1983 (na prawach rękopisu).