PRZEMYSŁOWY INSTYTUT AUTOMATYKI I POMIARÓW
MERA-PIAP
Al. Jerozolimskie 202 02-222 Warszawa Telefon 23-70-81

Samodzielna Pracownia Oprogramowania Systemów

LHO

A

Główny wykonawca mgr inż. Zbigniew Pilat

Wykonawcy mgr inż. Jacek Dunaj
mgr inż. Małgorzata Jacórzyńska-Śmigiera

Konsultant

Nr zlecenia S131TA

Rozwój oprogramowania podstawowego robotów UAP

Etap 5. Opracowanie opisu programu sterującego robotów UAP w zakresie organizacji programu, wykorzystania zasobów sprzętowych sterownika, inicjalizacji systemu sterowania i pętli głównej programu.

Zleceniodawca praca statutowa PIAP

Pracę rozpoczęto dnia kwiecień 1993 zakończono dnia 30.06.93

Kierownik Pracowni
mgr inż. Zbigniew Pilat

Z-ca Dyrektora
d/s Badawczo-Rozwojowych
dr inż. Jan Żabkowski

Praca zawiera:
stron
rysunków
fotografii
tabel
tablic
załączników 3 dodatki

Nr rejestr. 6978

Rozdzielnik - ilość egz:
Egz. 1 ZOINTE
Egz. 2 POS
Egz. 3 POS
Egz. 4 ZBS
Egz. 5
Egz. 6

Analiza deskrypiorowa

ROBOTY PRZEMYSŁOWE: UKŁAD STEROWANIA+OPROGRAMOWANIE

Analiza dokumentacyjna

Sprawozdanie zawiera opis programu sterującego robotów URP w zakresie organizacji programu, wykorzystania zasobów sprzętowych sterownika, inicjalizacji systemu sterowania i pętli głównej programu.

Tytuły poprzednich sprawozdań

UKD

PIAP 41/88 10000

2

SPIS TRESCI

1. WPROWADZENIE

Przemysłowy Instytut Automatyki i Pomiarów PIAP prowadzi prace w dziedzinie robotyki od połowy lat 70-tych. W tym czasie przejęto, produkowano i rozwijano licencyjne roboty IRb szwedzkiej firmy ASEA (obecnie ABB). Biegły również własne, orginalne prace naukowo-badawcze. Wykonano wiele aplikacji robotów w przemyśle. Rozwijano nowe, własne konstrukcje, tak w zakresie mechaniki jak i sterowania. W połowie lat 80-tych opracowano nowy, bazowy układ sterowania dla robotów przemysłowych. Była to na owe czasy nowoczesna konstrukcja wieloprocesorowa. Jednostka centralna, wykorzystująca 16-bitowy mikroprocesor Intel8086, zapewniała moc obliczeniową wystarczającą do realizacji sterowania ruchem typu CP oraz wielu innych funkcji robota, korzystnych z punktu widzenia wdrożeń. Panel programowania sterowany mikroprocesorem Intel8080 i wyposażony w wyświetlacz tekstowy umożliwił zorganizowanie systemu 'menu' oraz wprowadzenie konwersacyjnego trybu pracy z robotem. Rodzina robotów pracujących z tym układem sterowania została oznaczona symbolem IRp. Produkcji modeli o udźwigu 6 i 60 kg. podjęły się zakłady ZAP w Ostrowiu Wielkopolskim, w ramach licencji kupionej od PIAP.

Od początku 1991 roku w Instytucie PIAP są prowadzone prace nad nową generacją układów sterowania dla robotów przemysłowych, oznaczonych symbolem URP. Wśród wprowadzanych rozwiązań należy szczególnie podkreślić:

- zastosowanie magistrali AMS - standard firmy Siemens,
- zastosowanie nowych układów i podzespołów elektronicznych,
- wykorzystanie w konstrukcji pakietów sterownika technologii druku wielowarstwowego,
- wprowadzenie nowej konstrukcji jednostki centralnej,
- realizację nowej koncepcji pamięci masowej programów użytkowych robota,
- wprowadzenie cyfrowych sterowników położenia osi.

Przeprowadzone badania prototypów w połączeniu z częścią manipulacyjną o udźwigu 6 kg., wykazały znaczny wzrost niezawodności i odporności na zakłócenia elektromagnetyczne nowego układu sterowania w porównaniu z poprzednimi modelami. Wydaje się, że uzyskano poziom, który pozwala optymistycznie patrzeć na pracę sterownika w warunkach przemysłowych z punktu widzenia bezawaryjności. O zdolnościach aplikacyjnych robotów obok cech mechanicznych ich manipulatorów i jakości układu sterowania w równym stopniu decydują możliwości funkcjonalne realizowane przez program sterujący. Naturalną więc koleją rzeczy równolegle z pracami nad konstrukcją hardware'ową wykonywane było oprogramowanie podstawowe nowego sterownika. W pierwszej kolejności powstało oprogramowanie układu sterowania dla części manipulacyjnej robota o udźwigu 6kg. - URP-6. W pracach tych bazowano na doświadczeniach nabytych przy tworzeniu i pielęgnowaniu programu sterującego robotów IRp. Adaptacja wprost poprzedniego oprogramowania była niemożliwa chociażby z uwagi na różnice konstrukcyjne samego sterownika oraz zastosowanie nowych narzędzi programowych (kompilatory, linker, lokator). Zmieniono także algorytmy realizacji niektórych funkcji. Bardzo istotną zmianą była przebudowa całego oprogramowania w kierunku uzyskania struktury modułowej, aby można było w przyszłości łatwo konfigurować program sterujący dla konkretnego typu manipulatora i dla określonego zakresu realizowanych funkcji. Praktycznie powstał nowy program. Niezbędnym stało się więc opracowanie jego dokumentacji. Składa się ona z dwóch zasadniczych części: programu w postaci źródłowej (tabulogramu) i jego opisu. W skład dokumentacji programu sterującego zalicza się także podręcznik programowania robotów URP.

Opis programu sterującego robotów URP jest wykonywany etapowo. Niniejsza praca stanowi jego pierwszą część. Zawiera ona opis oprogramowania w zakresie jego organizacji, ogólnej struktury, inicjalizacji systemu sterowania i pętli głównej programu. Całość jest poprzedzona syntetycznym opisem konstrukcji hardware'owej układu sterowania oraz omówieniem zasad tworzenia programu sterującego (języki programowania, oprogramowanie narzędziowe).

2. KONFIGURACJA SPRZĘTOWA STEROWNIKA ROBOTÓW URp

Układ sterowania robotów URP jest zabudowany w szafie sterowniczej. Część cyfrowa jest umieszczona w kasecie z magistralą AMS. Sterownik ma budowę modułową. Poszczególne pakiety realizują określony zakres funkcji. Całością steruje jednostka centralna, która poprzez magistralę komunikuje się z pozostałymi modułami. W pamięci EPROM jednostki centralnej jest zapisane oprogramowanie systemowe układu sterowania, czyli program sterujący robota. Z punktu widzenia programu sterującego wszystkie pozostałe podzespoły, poza częścią cyfrową, są urządzeniami peryferyjnymi. Moduły znajdujące się w kasecie pełnią rolę interfejsów, niekiedy inteligentnych, pomiędzy jednostką centralną, a tymi urządzeniami. Bezpośrednio do jednostki jest podłączony, poprzez kanał transmisji szeregowej V-24, panel programowania. Poniżej przedstawiono konfigurację sprzętu umieszczonego w kasecie AMS. Poza zasilaczem kasety i zasilaczem rezolwerów wszystkie wymienione pakiety są bezpośrednio obsługiwane przez program sterujący. Dotyczy to także panelu programowania. Bloki te zostały w dalszej części bliżej przedstawione.

Pakiet	Pozycja w kasecie	Charakterystyka pakietu	Opis w pkt.
MV-70	01 - 05	zasilacz kasety	***
MV-12	06	systemowy pakiet wejść/wyjść dwustanowych	2.3
MV-12	07	pakiet wejść/wyjść dwustanowych użytkownika (1-16)	2.3
MV-12	08	pakiet wejść/wyjść dwustanowych użytkownika (17-32) - opcja	2.3
MV-62	10	pakiet pamięci masowej	2.4
MV-52	11	jednostka centralna z koprocesorem arytmetycznym	2.1
MV-20	12	sterownik położenia osi 9 - opcja	2.2
MV-20	13	sterownik położenia osi 8 - opcja	2.2
MV-20	14	sterownik położenia osi 7 - opcja	2.2
MV-20	15	sterownik położenia osi Z (6) - opcja	2.2
MV-20	16	sterownik położenia osi V	2.2
MV-20	17	sterownik położenia osi T	2.2
MV-20	18	sterownik położenia osi ALFA	2.2
MV-20	19	sterownik położenia osi TETA	2.2
MV-20	20	sterownik położenia osi FI	2.2
MV-21	21	zasilacz rezolwerów	***
*****	----	panel programowania robota	2.5

2.1. Pakiet jednostki centralnej MV-52

Pakiet jednostki centralnej nadzoruje i koordynuje pracę pozostałych pakietów znajdujących się w kasecie. Wykorzystano w nim mikroprocesor Intel 80186, pracujący z zegarem 8MHz. Jako koprocesor arytmetyczny jest używany model 8087-1, również firmy Intel. Oba te układy są dokładnie opisane w literaturze zawartej w Dodatku A.

2.1.1. Zasoby pamięci

MV-52 ma bardzo duże możliwości jeśli idzie o dostępne zasoby pamięci. Na pakiecie zamontowano jedną parę układów RAM po 32 kB każda (typ 62256) i pięć par podstawek pod układy pamięci. Dzięki wykorzystaniu pól krosowych oraz programowalnych dekoderów PAL, możliwe jest konfigurowanie pamięci, stosownie do wymagań konkretnej aplikacji, tak pod względem obsady poszczególnych podstawek jak i przypisanych im adresów. Dla potrzeb programu sterującego robotów URP przyjęto wykorzystanie następujących obszarów pamięci:

- 00000 0FFFFH 64 kB RAM (z podtrzymaniem bateryjnym)
- 10000 1FFFFH 64 kB RAM
- E0000 FFFFFH 128 kB EPROM

Dodatkowo okno o szerokości 64 kB, od adresu 70000H do 7FFFFH jest wykorzystywane do adresowania pakietu pamięci masowej MV-62 (p. 2.4). Na rys. 2.1. przedstawiono wykorzystanie pamięci przez program sterujący robotów URP, w obszarze adresowania 1 MB jednostki centralnej. Konstrukcja pakietu dopuszcza obsadzenie jednej pary podstawek układami pamięci stałej, kasowalnej elektrycznie - EEPROM. Możliwość ta nie jest obecnie wykorzystana. W przyszłości, wraz z rozbudową programu sterującego, w pamięci EEPROM przewiduje się przechowywanie parametrów, ustalanych dla konkretnej aplikacji. Mogą to być na przykład zakresy i prędkości ruchów poszczególnych osi, dane o konfiguracji układu sterowania (liczba pakietów wejść/wyjść) lub o języku dialogu z operatorem (wprowadzenie komunikatów i opisu menu w języku angielskim, niemieckim). Należy także zaznaczyć, że konstrukcja MV-52 pozwala na wykorzystanie bitów adresowych A20...A23, czyli dostępu do pamięci 16 MB. Możliwość tę wprowadzono perspektywicznie, pod kątem włączenia układu sterowania robota w lokalną sieć komputerową.

00000H 003FFH	Wektor przerwań
00400H 008FFH	Niewykorzystane
00900H 02FFFH	Stos
03000H 1FFEFH	Dane i obszar programów użytkowych
20000H 6FFFFH	Niewykorzystane
70000H 7FFFFH	Okno dostępu do bloku MV-62 pamięci masowej programów użytkowych
80000H DFFFFH	Niewykorzystane
E0000H F6FFFH	Kod programu sterującego
FA000H FFFEFH	Program monitora operatorskiego
FFFF0H FFFFFH	BOOTSTRAP

Rys.2.1. Wykorzystanie pamięci przez program sterujący robotów URP.

2.1.2. Układ przerwań

Jednostka centralna MV-52 zawiera dwa programowalne kontrolery przerwań typu 8259A. Są one połączone wewnętrznym sterownikiem przerwań procesora poprzez linie INT0, INT1, INT2, INT3. Taka konfiguracja sprzętowa umożliwia obsługę maksymalnie 17 przerwań:

- przerwanie niemaskowalne - na wejście NMI procesora podawana jest suma logiczna sygnału PFIN (zanik zasilania) i BTMO (brak potwierdzenia przekazu po magistrali kasety),

- 8 przerwań z pierwszego sterownika 8259A

- 8 przerwań z drugiego sterownika 8259A

Program sterujący wykorzystuje dwa przerwania:

- przerwanie 19 - zgłaszane przez licznik 2 wewnętrznego timera mikroprocesora 80186,

- przerwanie 33 - zgłaszane przez kanał B układu transmisji szeregowej Z8530 na linię IR10 pierwszego kontrolera.

Każde inne przerwanie jest traktowane jako zdarzenie nieoczekiwane, a więc będące wynikiem błędu w pracy systemu. W takim przypadku uaktywniana jest specjalna procedura wyświetlająca na panelu programowania numer zaistniałego przerwania - potwierdzenie przez operatora przyjęcia tej informacji (przycisk na panelu) powoduje programowy restart systemu.

W Dodatku B opisano najistotniejsze, z punktu widzenia oprogramowania, cechy układu wewnętrznego sterownika przerwań mikroprocesora 80186.

2.1.3. Kanały transmisji szeregowej

Jednostka centralna MV-52 ma dwa kanały transmisji szeregowej, oznaczone A i B. Oba one są obsługiwane przez układ scalony podwójnego nadajnika/odbiornika Z8530 (firmy THOMSON). Kanał A pracuje zawsze w standardzie napięciowym i jest przeznaczony do komunikacji układu sterowania robota z zewnętrznym komputerem. Poprzez kanał B jednostka centralna komunikuje się z panelem programowania. Kanał ten może być przełączany, przez zmianę połączeń na polach krosowych, w tryb napięciowy lub prądowy wykorzystywany do wymiany informacji między programem sterującym a programem obsługi panelu programowania.

2.2. Pakiety cyfrowych sterowników położenia osi MV-20

W opisywanej wersji programu sterującego robota URP-6 obsługiwanych jest 5 cyfrowych sterowników położenia osi typu MV-20. W maksymalnej konfiguracji, w kasecie układu sterowania można zainstalować 9 takich pakietów. Program sterujący widzi sterowniki położenia jako porty wejścia/wyjścia:

- port wejściowy DATA - odczyt położenia osi,

- port wyjściowy DATA - zapis przyrostu położenia zadanego do realizacji,

- port wejściowy CTRL - odczyt słowa stanu sterownika MV-20,

- port wyjściowy CTRL - zapis rozkazu zmiany stanu sterownika MV-20,

- port wyjściowy RESINT - kasowanie przerwania, które może być wykorzystywane w sterowniku MV-20 - w obecnej postaci układu sterowania robotów URP nie jest używane,

- port wyjściowy PROGRST - programoway restart sterownika MV-20, zainicjowany z poziomu jednostki centralnej.

Przyporządkowanie adresów I/O poszczególnym sterownikom jest następujące:

Nazwa pakietu MV-20:	Adres portu DATA:	Adres portu CTRL:	Adres portu RESINT:	Adres portu PROGRST:
Sterownik osi FI	0F000H	0F004H	0F008H	0F00CH
Sterownik osi TETA	0F100H	0F104H	0F108H	0F10CH
Sterownik osi ALFA	0F200H	0F204H	0F208H	0F20CH
Sterownik osi T	0F300H	0F304H	0F308H	0F30CH
Sterownik osi V	0F400H	0F404H	0F408H	0F40CH
Sterownik osi 6 (Z)	0F500H	0F504H	0F508H	0F50CH
Sterownik osi 7	0F600H	0F604H	0F608H	0F60CH
Sterownik osi 8	0F700H	0F704H	0F708H	0F70CH
Sterownik osi 9	0F800H	0F804H	0F808H	0F80CH

2.2.1. Odczyt portu DATA w pakiecie MV-20

Odczyt portu DATA jest wykorzystywany przez program sterujący do odczytywania rzeczywistej pozycji w jakiej znajduje się sterowana przez dany pakiet MV-20. Odczyty takie są dokonywane:

- dla sprawdzenia, czy sterowniki położenia pracują poprawnie, tzn czy nie gubią pozycji; odczytaną pozycję porównuje się z pozycją zapamiętaną w wewnętrznych rejestrach położenia jednostki centralnej,
- dla uaktualnienia pozycji w wewnętrznych rejestrach położenia, gdy ramiona robota poruszane są ręcznie (przy wyłączonych napędach); rejestry są uaktualniane tylko wtedy, gdy robot był uprzednio zsynchronizowany.

Położenie odczytywane jest jako liczba 16-to bitowa ze znakiem. Gdy robot jest zsynchronizowany i znajduje się w pozycji synchronizacji, to wartości położeń odczytywane z portu DATA są zerowe. Jeśli robot nie jest zsynchronizowany, to odczytane z portu DATA wartości są przypadkowe.

2.2.2. Zapis do portu DATA w pakiecie MV-20

Zapis do portu DATA wykorzystywany jest w programie sterującym do wpisywania słowa przyrostu, dla uzyskania przemieszczenia danej osi robota o zadaną wartość inkrementów, w zadanym czasie i w zadanym kierunku. Oprogramowanie podstawowe sterownika MV-20 umożliwia zadawanie czterech różnych czasów realizacji przyrostów: 8, 16, 32, 64 ms. Formaty danych zapisywanych do portu DATA różnią się w zależności od wymaganego okresu realizacji i wygladają następująco:

okres realiz. przyr.	Słowo wysyłane do portu DATA															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
8 ms	Z	0	0	0	1	x	x	x	x	p	r	z	y	r	o	s t
16 ms	Z	0	0	1	x	x	x	x	p	r	z	y	r	o	s	t
32 ms	Z	0	1	x	x	x	x	p	r	z	y	r	o	s	t	
64 ms	Z	1	x	x	x	x	p	r	z	y	r	o	s	t		

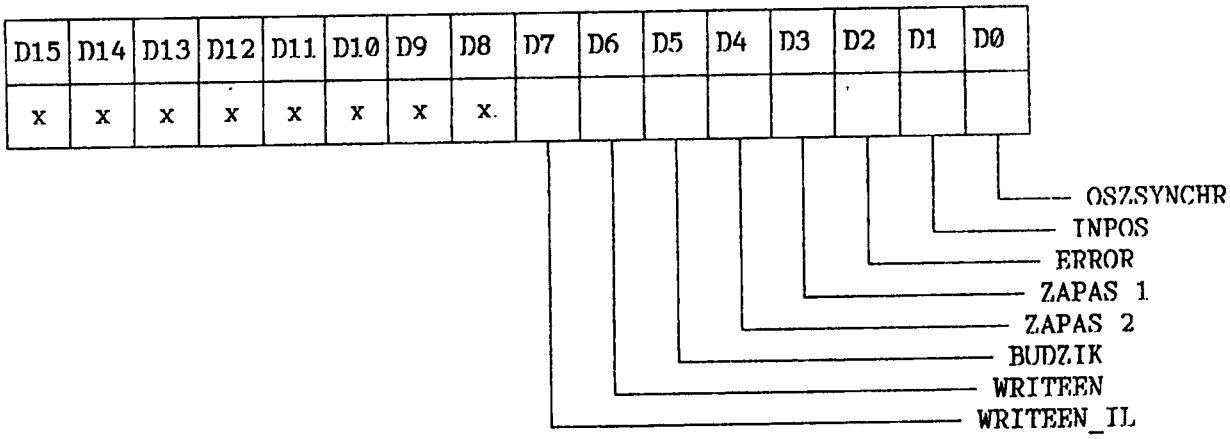
gdzie: Z – jest znakiem przyrostu (1 - ruch "+", 0 - ruch "-"),
x – wartość bitu nieistotna,
przyrost – bezwzględna wartość przyrostu położenia.

2.2.3. Odczyt z portu CTRL w pakiecie MV-20

Słowo stanu, odczytane z portu CTRL dostarcza informacji o aktualnym stanie pracy sterownika MV-20. Na poszczególnych bitach jest zakodowana następująca informacja:

- bit D0 - OSZSYNCHR – wartość "1" informuje, że oś jest zsynchronizowana (odpowiednia dioda na płycie czołowej sterownika MV-20 jest zgaszona),
- bit D1 - INPOS – wartość "1" informuje, że błąd położenia osi znajduje się w strefie zerowej (odpowiednia dioda na płycie czołowej sterownika MV-20 jest zgaszona),
- bit D2 - ERROR – wartość "1" informuje, że błąd położenia osi jest zbyt duży, przekracza linit określony przez konstruktorów (odpowiednia dioda na płycie czołowej sterownika MV-20 jest zaświecona),
- bit D3 - ZAPAS 1 – bit rezerwowy, obecnie niewykorzystany,
- bit D4 - ZAPAS 2 – bit rezerwowy, obecnie niewykorzystany,
- bit D5 - BUDZIK – wartość "1" informuje, że oprogramowanie podstawowe sterownika MV-20 zawiesiło się, potrzebny jest programowy reset z poziomu jednostki centralnej układu sterowania,
- bit D6 - WRITEEN – wartość "1" informuje, że sterownik jest gotowy do przyjęcia następnej porcji przyrostu do realizacji,
- bit D7 - WRITEEN_IL – wartość "1" informuje, że wszystkie sterowniki MV-20, pracujące aktualnie w kasecie układu sterowania są gotowe do przyjęcia następnej porcji przyrostu do realizacji,
- bity D8 do D15 - niewykorzystane.

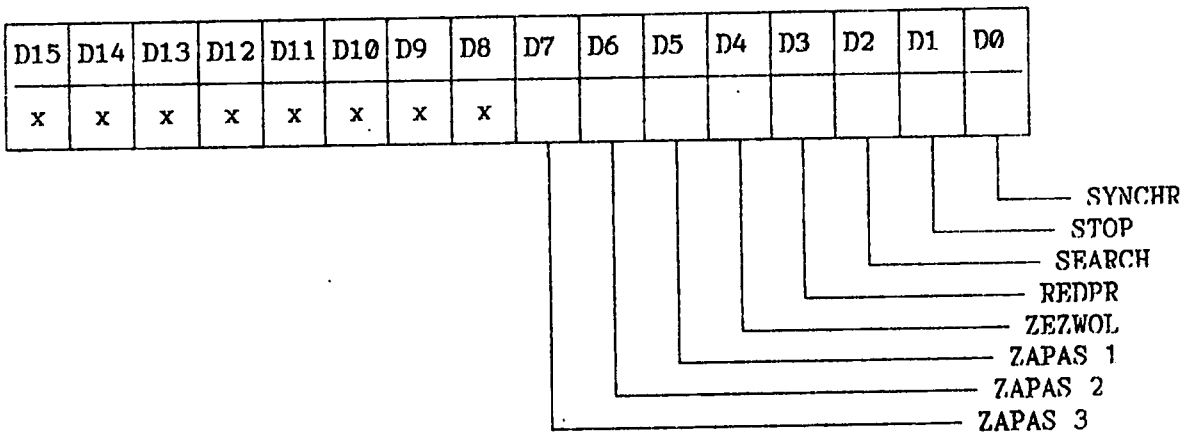
Poniżej przedstawiono w sposób schematyczny postać słowa stanu sterownika MV-20:



gdzie: x – wartość bitu nieistotna.

2.2.4. Zapis do portu CTRL w pakiecie MV-20

Zapis do portu CTRL wykorzystywany jest przez program sterujący do zmiany stanu pracy cyfrowych sterowników położenia osi. Format przesyłanego z jednostki centralnej słowa sterującego jest następujący:



gdzie: x – bity niewykorzystywane przez program sterujący MV-20 i przy wysyłaniu słowa stanu do sterownika ustawiane na zero.

- bit D7 - SYNCHR - zapis słowa stanu z bitem SYNCHR równym 1, powoduje rozpoczęcie synchronizacji danej osi.

- bit D7 - STOP – zapis słowa stanu z bitem STOP równym 1 powoduje zerowanie stanu sterownika położenia, tzn. wyzerowanie błędu położenia, co powoduje szybkie zatrzymanie osi, rozsynchronizowanie osi i zerowanie rejestrów wewnętrznych sterownika.

2.3. Pakiety wejść/wyjść cyfrowych MV-12

Pakiet MV-12 daje użytkownikowi do dyspozycji 16 wejść i 16 wyjść dwustanowych. Mają one, z punktu widzenia jednostki centralnej, organizację bajtową. Oznacza to, że program sterujący odczytuje osiem młodszych wejść spod

jednego adresu, a osiem starszych spod innego. Podobnie jest przy zapisywaniu zadanego stanu wyjść. Program sterujący widzi więc pakiet MV-12 jako dwa wejściowe (CSA, CSB) i dwa wyjściowe (CSC, CSD) porty danych. Dodatkowo jednostka centralna może odczytywać bajt informacyjny z portu stanu pakietu - CSSTAT. Wyjścia pakietu MV-12 są zabezpieczone przed przeciążeniem i zwarciem. Gdy taki stan nastąpi, informacja o tym jest przesyłana przez port stanu do jednostki centralnej. Łącząc odpowiednio piny na polach krosowych, można spowodować, aby po wykryciu zwarcia na wyjściach, pakiet MV-12 generował przerwanie na wybranej linii (INT0 do INT7) magistrali AMS. Do zerowania tych przerwań na pakiecie służy wyjście ZERINT.

Przyporządkowanie adresów portów I/O poszczególnym pakietom MV-12 w układzie sterowania robota URP jest następujące:

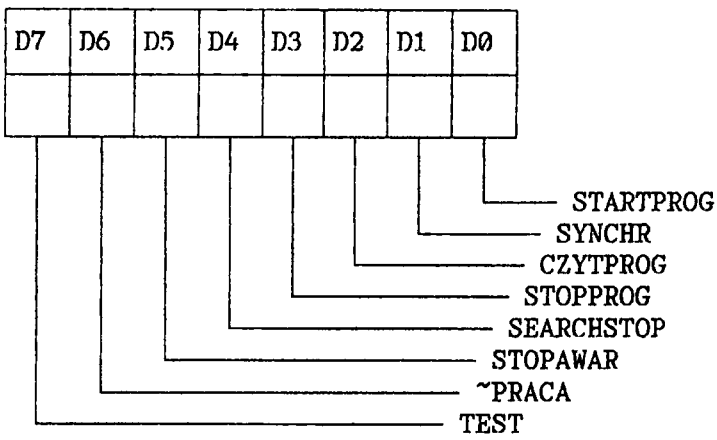
Przeznaczenie pakietu MV-12	Adres portu					
	CSA	CSB	CSC	CSD	CSSTAT	ZERINT
Systemowy	0F000H	0F004H	0F008H	0F008H	0F008H	0F00CH
Użytkownika (1-16)	0F100H	0F104H	0F108H	0F108H	0F108H	0F10CH
Użytkownika (17-32)	0F100H	0F104H	0F108H	0F108H	0F108H	0F10CH

2.3.1. Systemowy pakiet wejść/wyjść cyfrowych MV-12

Systemowy pakiet wejść/wyjść jest wykorzystywany przez program sterujący do:

- odczytu stanu przycisków panelu operacyjnego,
- odczytu sygnałów określających stan w jakim znajduje się sprzęt,
- zapalania i gaszenia lampek na panelu operacyjnym,
- wysyłania wewnętrznych jednobitowych sygnałów sterujących.

Format bajtu odczytywanego z portu CSA systemowego pakietu MV-12 jest następujący:



Znaczenie poszczególnych wejść (bitów w odczytanym bajcie) przedstawiono

poniżej:

- bit D0 - STARTPROG - wartość wejścia równa "1" oznacza wciśnięcie przycisku START na panelu operacyjnym,
- bit D1 - SYNCHR - wartość wejścia równa "1" oznacza wciśnięcie przycisku SYNCHRONIZACJA na panelu operacyjnym,
- bit D2 - CZYTPROG - wartość wejścia równa "1" oznacza wciśnięcie przycisku PK1 (odczyt programu z pamięci masowej) na panelu operacyjnym,
- bit D3 - STOPPROG - wartość wejścia równa "1" oznacza wciśnięcie przycisku STOP PROGRAMU na panelu operacyjnym,
- bit D4 - SEARCHSTOP - wejście niewykorzystane,
- bit D5 - STOPAWAR - wartość wejścia równa jeden oznacza ze układ sterowania znajduje się w stanie "stopu awaryjnego",
- bit D6 - ~PRACA - zerowa wartość wejścia oznacza, że układ sterowania znajduje się w "praca" - zasilane są silniki robota,
- bit D7 - TEST - wejścia określa stan przełącznika testowego w zespole bezpieczników i styczników (z przodu szafy sterowniczej).

Format bajtu odczytywanego z portu CSB systemowego pakietu MV-12 jest następujący:

D7	D6	D5	D4	D3	D2	D1	D0
x	x	x	x	x	x		

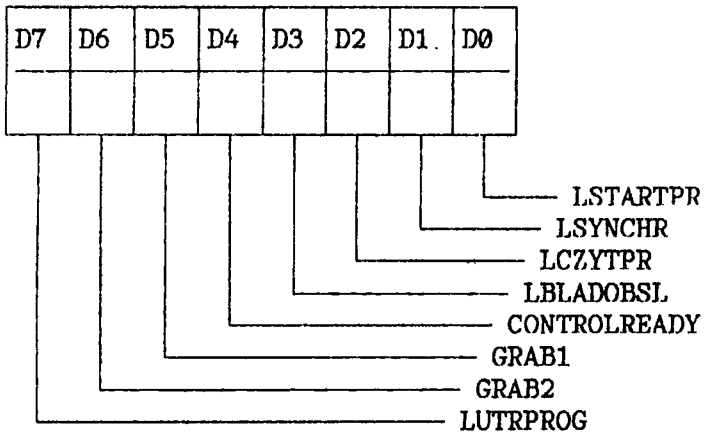
cz. ob. p. p.
INPOS

gdzie: x - bity nieinterpretowane przez program sterujący.

Znaczenie poszczególnych wejść istotnych (bitów w odczytanym bajcie) przedstawiono poniżej:

- bit D0 - cz. ob. p. p. - czujnik obecności panelu programowania w drzwiach szafy sterowniczej,
- bit D1 - INPOS - wartość wejścia równa "1" oznacza, że błędy położenia osi we wszystkich sterownikach MV-20, które aktualnie pracują w szafie, znajdują się w strefie zerowej.

Format bajtu zapisywanego do portu CSC systemowego pakietu MV-12 jest następujący:



Znaczenie poszczególnych wyjść (bitów w zapisywanym bajcie) przedstawiono poniżej:

- bit D0 - LSTARTPR - wartość wyjścia równa "1" oznacza zaświecenie lampki w przycisku START na panelu operacyjnym,

- bit D1 - LSYNCHR - wartość wyjścia równa "1" oznacza zaświecenie lampki w przycisku SYN (trwa synchronizacja robota) na panelu operacyjnym,

- bit D2 - CZYTPROG - wartość wyjścia równa "1" oznacza zaświecenie lampki w przycisku PK1 (odczyt programu z pamięci masowej) na panelu operacyjnym,

- bit D3 - LBLADOBSL - wartość wyjścia równa "1" oznacza zaświecenie lampki przycisku BLA (sygnalizacja błędu obsługi) na panelu operacyjnym,

- bit D4 - CONTROLREADY - wartość wyjścia równa "1" oznacza programowe załączenie stanu STOP AWARYJNY,

- bit D5 - GRAB1 - wartość wyjścia równa jeden oznacza, że został wysterowany przekaźnik elektropneumatyczny chwytaka A robota,

- bit D6 - GRAB2 - wartość wyjścia równa jeden oznacza, że został wysterowany przekaźnik elektropneumatyczny chwytaka B robota,

- bit D7 - LUTRPROG - *wartość wyjścia równa "1" oznacza zaświecenie lampki przycisku RAM (brak programu użytkowego w pamięci RAM) na panelu operacyjnym,

Format bajtu zapisywanego do portu CSD systemowego pakietu MV-12 jest następujący:

D7	D6	D5	D4	D3	D2	D1	D0
x	x	x	x	x	x		

ZEWSTRT

gdzie: x - bity nieinterpretowane przez program sterujący.

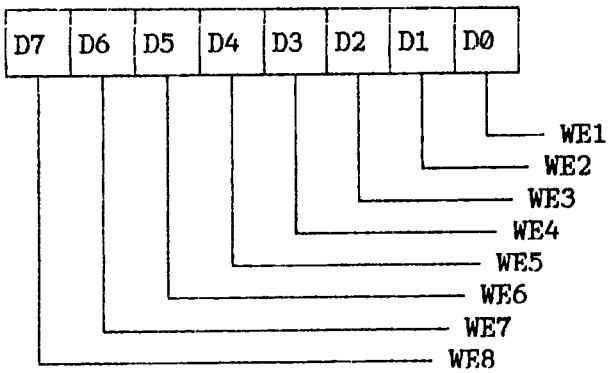
Bit D0 - ZEWSTRT - jest odpowiedzialny za sygnalizację na zewnątrz stanu automatycznego wykonywania programu użytkowego. Wartość wyjścia równa "1" oznacza, że załączony jest zewnętrzny sygnał informacyjny.

2.3.2. Pakiety wejść/wyjść cyfrowych użytkownika MV-12

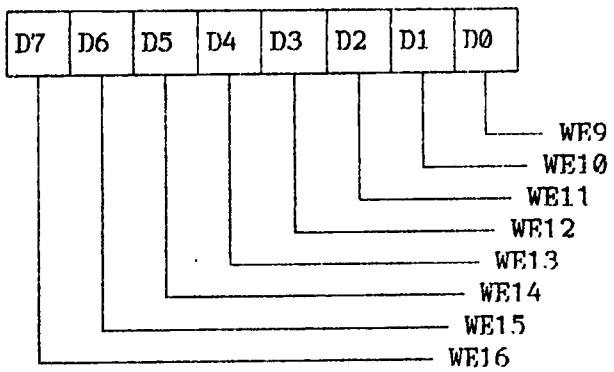
Pakiety wejść/wyjść cyfrowych użytkownika wykorzystywane są do komunikacji układu sterowania robota z urządzeniami współpracującymi. Sygnały ze złączy tych pakietów są doprowadzone do skrzynki rozgałęzienia wejść/wyjść. Wszystkie przewody kończą się na listwie zaciskowej, do której z drugiej strony dołącza się przewody od urządzeń zewnętrznych.. Przyporządkowanie numerów wejść/wyjść pakietom i bitom w bajtach czytanych z pakietu jest następujące:

Pakiet MV-12 (1-16)

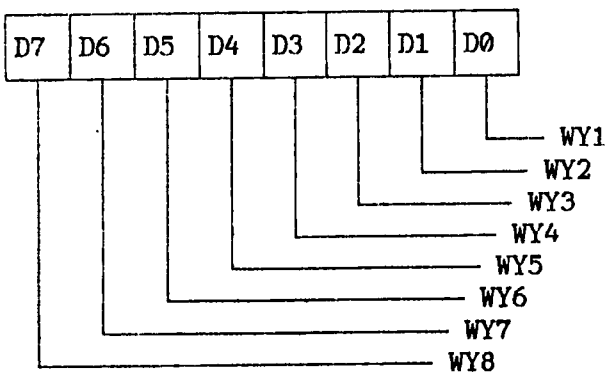
CSA - wejścia 1-8



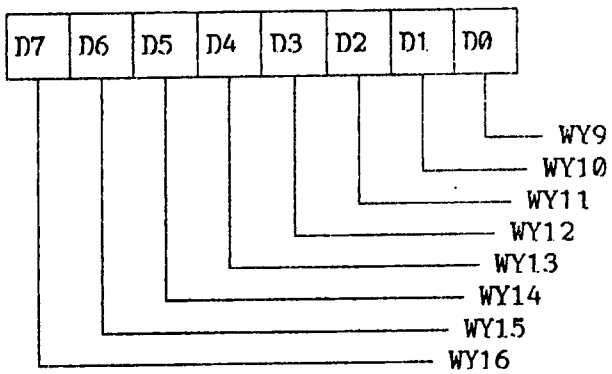
CSB - wejścia 9-16



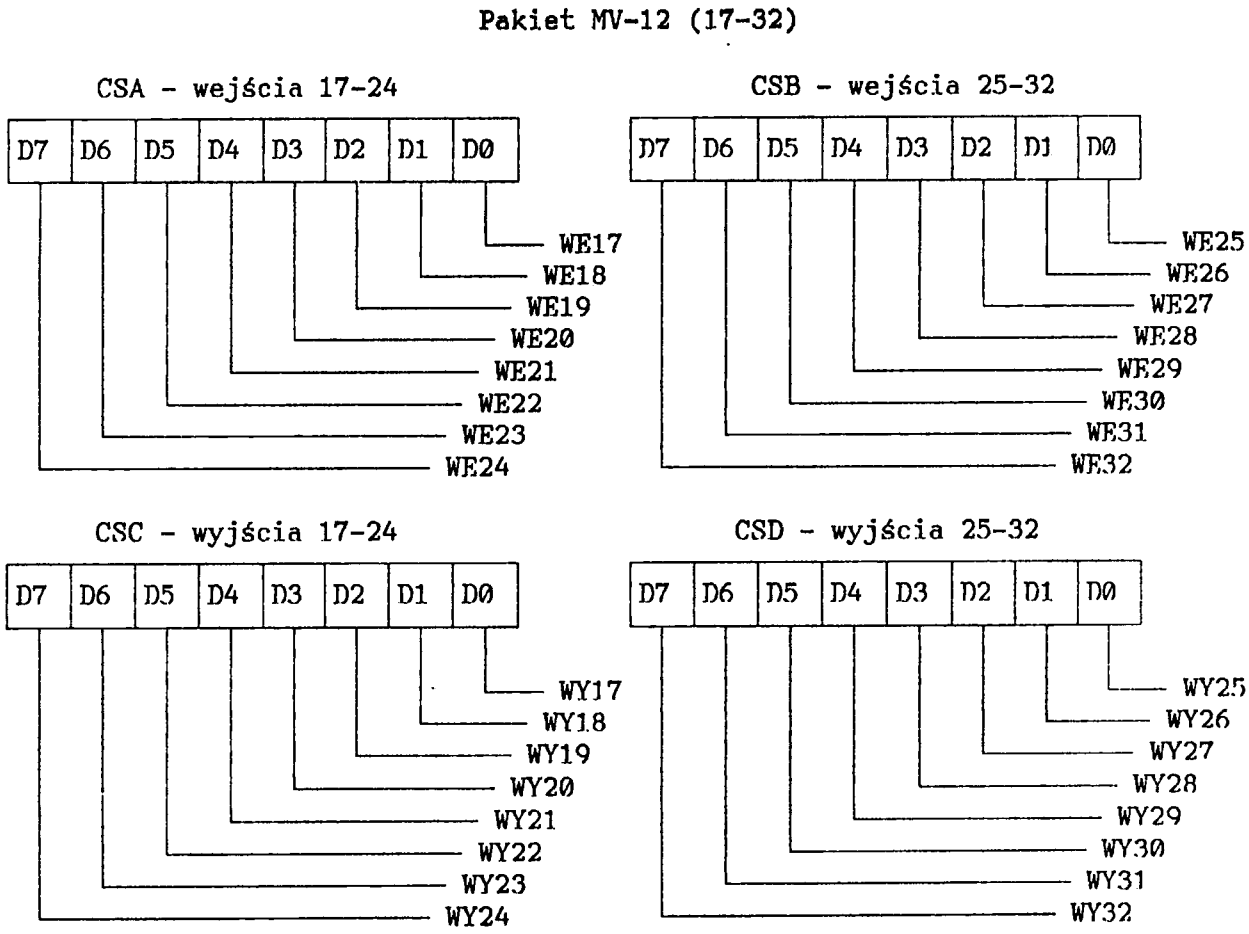
CSC - wyjścia 1-8



CSD - wyjścia 9-16



17



2.3.3. Pakiet pamięci masowej MV-62

Pakiet pamięci masowej MV-62, przeznaczony do przechowywania programów użytkowych robota, wykorzystuje układy scalone pamięci stałej EEPROM elektrycznie zapisywanej i kasowanej. Pakiet zawiera podstawki pod 8 par tych układów. Każda para obejmuje jeden blok pamięci o pojemności 64 kilobajtów, maksymalna pojemność pamięci zainstalowanej w pakiecie MV-62 wynosi więc 512 kilobajtów. Z punktu widzenia programu obsługi pamięci masowej nie jest konieczne obsadzenie wszystkich par podstawek, ponieważ aktualna konfiguracja zostaje ustalona i zapisana na etapie programowego inicjowania pakietu MV-62. Wymagane jest jednak obsadzenie zerowego bloku pamięci (pierwsza para podstawek), ponieważ właśnie w nim jest przechowywana i uaktualniana informacja o konfiguracji pakietu, położeniu wszystkich zapisanych w nim plików (programów robotowych) i ewentualnych uszkodzeniach pamięci wykrytych w trakcie jej eksploatacji.

Pakiet pamięci masowej zawiera pojedynczy, 8-bitowy rejestr wejściowy i pojedynczy, 16-bitowy rejestr wyjściowy - oba dostępne pod tym samym adresem w przestrzeni I/O sterownika MV-52. Formaty obu tych rejestrów są następujące:

- rejestr wejściowy:

D7	D6	D5	D4	D3	D2	D1	D0

Znaczenie poszczególnych bitów:

D7 - zezwolenie na przerwanie

D6, D5, D4, D3 - kod zabezpieczający:
1100 - zezwolenie na wpis
0011 - zakaz wpisu

D2, D1, D0 - numer bloku:
000 - blok zerowy
.....
111 - blok siódmy

- rejestr wyjściowy, odczytywany przez jednostkę centralną sterownika:

D15	D14	D13	D12	D11	D10	D9	D8

D7	D6	D5	D4	D3	D2	D1	D0

Znaczenie poszczególnych bitów:

D15 - blokada zapisu. Jest to aktualny stan sygnału ochrony danych na pakiecie MV-62, ustalany za pomocą dwupołożeniowego, zewnętrznego przełącznika na płycie czołowej modułu MV-62

D14, D13, D12, D11, D10 - nieużywane

D9 - gotowość do wpisu

D8 - zezwolenie przerwania do jednostki centralnej pakietu MV-52

D7 - zezwolenie na przerwanie

D6, D5, D4, D3 - kod zabezpieczający:
1100 - zezwolenie na wpis
0011 - zakaz wpisu

D2, D1, D0 - numer bloku pamięci:
000 - blok zerowy
.....
111 - blok siódmy

Pakiet pamięci masowej MV-62 może być adresowany w podstawowej przestrzeni 1 megabajtowej sterownika MV-52 lub też w obszarze powyżej 1 MB, do 16 MB. O wyborze decyduje odpowiednie zainicjowanie adresu stronicowania adresów pakietu sterownika MV-52, dostępnego pod adresem I/O 120H. W programie robota adres ten zainicjowano tak, że pakiet pamięci masowej MV-62 jest dostępny w podstawowej przestrzeni adresowej mikroprocesora 80186. Dodatkowo sprzętowo wprowadzono do dekodera adresu (układ typu PAL) pakietu pamięci masowej MV-62 taką wartość, że pakiet ten jest dostępny dla jednostki centralnej od adresu fizycznego 70000H.

Pakiet pamięci masowej MV-62 zajmuje 64 KB obszaru adresowego (a więc od 70000H do 7FFFFH). Aby na tym obszarze "zmieścić" pamięć o maksymalnej pojemności 0,5 megabajta, należy najpierw wybrać właściwy blok pamięci pakietu MV-62 (co wykonuje się przez wpisanie odpowiedniej wartości do rejestru wejściowego pakietu MV-62), a następnie odczytać (lub zapisać) żądaną komórkę (lub słowo) wewnątrz tego bloku w taki sam sposób, w jaki wykonywany jest odczyt/zapis pamięci w obszarze adresowalności mikroprocesora 80186.

3. TWORZENIE PROGRAMU STERUJĄCEGO ROBOTÓW URp

W rozdziale tym podano zasady tworzenia i wzajemnego wykorzystania procedur programu sterującego robotów URp, a także opisano kolejne etapy tworzenia obrazu tego programu od kompilacji jego poszczególnych części do uzyskania gotowej postaci w formacie INTEL-HEX przeznaczonej do załadowania do pamięci RAM sterownika robota lub przepalenia do pamięci stałej typu EPROM. Kolejne etapy tworzenia programu sterującego robotów URp pokazano na rysunku zamieszczonym na następnej stronie.

3.1. Języki programowania programu sterującego robotów URp

Program sterujący dla robotów URp został napisany w dwóch językach programowania: języku C i języku asemblera mikroprocesora 8086/80186 i uruchomiony w środowisku oprogramowania narzędziowego firmy INTEL. Ze względu na to, że procedury pisane w obu tych językach wzajemnie się wykorzystują muszą być zachowane zasady ich wzajemnego odwoływania się, przekazywania parametrów, zwracania wartości po wykonaniu tych procedur, a także ewentualnej modyfikacji wskaźnika stosu po przekazaniu sterowania do podprogramu nadrzędnego. Do kompilowania plików źródłowych programu sterującego robotów URp wykorzystano dwa następujące programy firmy INTEL:

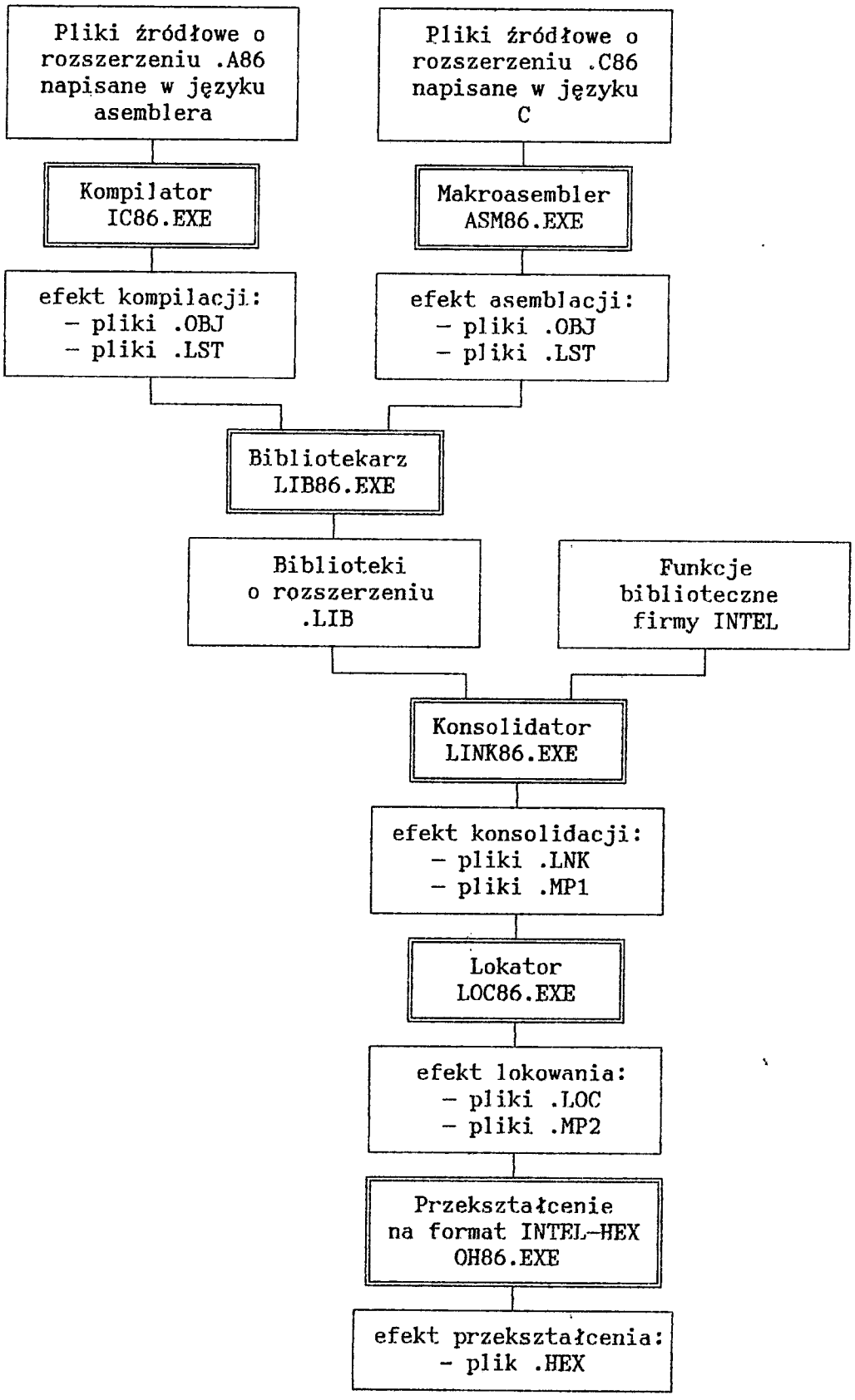
- kompilator języka C o nazwie iC-86,
- asembler mikroprocesora 8086/80186 o nazwie ASM-86.

Oba te programy pozwalają na utworzenie postaci wyjściowej .OBJ kompilowanego zbioru w zależności od narzuconego przez użytkownika typu mikroprocesora w jaki wyposażono dany sterownik. Językiem "narzucającym" zasady wzajemnych zależności pomiędzy procedurami programu sterującego robotów URp jest język C i dlatego ustalenia kompilatora iC-86 zostaną tutaj szerzej omówione.

3.2. Kompilator iC-86

Wszystkie kompilatory języków wysokiego poziomu firmy INTEL, za wyjątkiem jednej z opcji zawartej w kompilatorze iC-86 języka C, korzystają z tej samej sekwencji wywoływania podprogramów i przekazywania parametrów. Parametry wywołania podprogramów są przekazywane poprzez stos mikroprocesora (lub koprocesora) w porządku określonym ich wystąpieniem w instrukcji wywołania. Każdy parametr zajmuje zawsze wielokrotność dwóch bajtów na stosie (nawet jeśli jest to parametr jednobajtowy). Stosowane przez kompilatory różnych języków programowania sposoby przekazywania argumentów z programu nadrzędnego do funkcji pomocniczej różnią się między sobą w następujących punktach:

- porządkiem w jakim argumenty przekazywane do wywoływanej funkcji są umieszczane na stosie mikroprocesora,
- czy program nadrzędny przekazuje argumenty przez wartość (tzn. że na stos jest odkładana aktualna wartość danego argumentu) czy przez "referencję" (tzn. że na stos jest odkładany pointer do danego argumentu),
- formatem parametrów przekazywanych przez wartość.



Kolejne etapy tworzenia programu sterującego robotów URp

Kompilator iC-86 zawsze przekazuje parametry poprzez wartość, za wyjątkiem przypadku gdy argumentem wywołania jest tablica (wtedy przekazuje pointer do tablicy).

Kompilator iC-86/286 umożliwia stosowanie dwóch rodzajów odwołań z poziomu programu napisanego w języku C do funkcji pomocniczych. Są to:

- odwołanie ze stałą listą parametrów (FPL - fixed parameter list),
- odwołanie ze zmienną listą parametrów (VPL - varied parameter list).

Odwołanie ze stałą listą parametrów jest domyślnym odwołaniem kompilatora iC-86 i jest najczęściej stosowanym odwołaniem kompilatorów innych języków programowania.

3.2.1. Odwołania ze stałą listą parametrów - FLP

W przypadku odwołania ze stałą listą parametrów (FLP) funkcja nadrzędna przekazuje argumenty do procedury wywoływanej w następujący sposób:

- na stos mikroprocesora odkładane są kolejno, licząc od lewej strony, wszystkie argumenty wywołania podprogramu, które nie są argumentami zmiennoprzecinkowymi,
- na stos koprocesora odkładane są kolejno, licząc od lewej strony, pierwsze siedem argumenty zmiennoprzecinkowe wywołania podprogramu,
- jeśli liczba parametrów zmiennoprzecinkowych jest większa niż siedem, to wszystkie argumenty o numerach powyżej siedmiu, licząc od lewej strony, są odkładane na stos mikroprocesora. W takim przypadku argumenty zmiennoprzecinkowe nie muszą występować na stosie w sposób ciągły, lecz mogą być rozdzielone argumentami nie-zmiennoprzecinkowymi w sposób, jaki ustala kolejność występowania wszystkich argumentów w linii wywołania podprogramu.

Należy jeszcze raz podkreślić, że przy odwołaniu typu FPL argumenty wywołania podprogramu są umieszczane na stosie mikroprocesora w kolejności od lewej do prawej strony ich wystąpienia w instrukcji wywołującej podprogram (tzn. lewy skrajny parametr zostaje umieszczony na stosie jako pierwszy, a więc posiada najwyższy adres). Argumenty wielobajtowe są odkładane na stos w ten sam sposób, w jakim występują w segmencie danych, tzn. najpierw jest odkląd̨ane starsze słowo, potem młodsze. Każdy argument wywołania podprogramu zajmuje na stosie parzystą liczbę bajtów. Jeśli jest to argument jednobajtowy, to dopełnieniem do pełnego słowa jest nieokreślona wartość jednobajtowa, występująca jako bardziej znacząca. Podobnie kompilator dopełnia argumenty złożone z nieparzystej liczby bajtów.

Argumenty zmiennoprzecinkowe zajmują na stosie koprocesora 80 bitów każdy. Dla zgodności ze standardem ANSI C z 1988 roku deklaracja typu parametru określa zajętość pamięci przez argument zmiennoprzecinkowy na stosie mikroprocesora.

Jeśli wywoływany podprogram ma zwracać strukturę (lub unię) to funkcja nadrzędna jako ostatni argument umieszcza na stosie wartość, która jest adresem, pod którym wywoływany podprogram ma umieścić tę strukturę (lub unię).

3.2.2. Odwołanie ze zmienną listą parametrów - VPL

W przypadku odwołania ze zmienną listą parametrów (VLP) funkcja nadrzędna przekazuje wszystkie argumenty wywołania poprzez stos mikroprocesora (nie jest wykorzystywany stos koprocesora, jak ma to miejsce w przypadku odwołania FPL). Przy odwołaniu typu VPL argumenty wywołania podprogramu są umieszczane na stosie mikroprocesora w kolejności prawej do lewej strony ich wystąpienia w instrukcji wywołującej podprogram (tzn. prawy skrajny parametr zostaje umieszczony na stosie jako pierwszy, a więc posiada najwyższy adres - jest to na odwrót jak dla odwołania FPL). Argumenty wielobajtowe są odkładane na stos w ten sam sposób, w jakim występują w segmencie danych, tzn. najpierw jest odkładane starsze słowo, potem młodsze. Każdy argument wywołania podprogramu zajmuje na stosie parzystą liczbę bajtów,. Jeśli jest to argument jednobajtowy, to dopełnieniem do pełnego słowa nie jest nieokreślona wartość jednobajtowa, jak dla odwołania FLP, lecz, w zależności od typu argumentu, albo bajt wypełniony zerami, albo bajt zawierający informacje o znaku. Argumenty wielobajtowe, o nieparzystej liczbie bajtów są dopełniane bajtem o niekreślonej wartości.

Dla zgodności ze standardem ANSI C z 1988 roku deklaracja typu parametru określa zajętość pamięci przez argument zmiennoprzecinkowy na stosie.

Jeśli wywoływany podprogram ma zwracać strukturę (lub unię) to funkcja nadrzędna jako ostatni argument umieszcza na stosie wartość, która jest adresem, pod którym wywoływany podprogram ma umieścić tę strukturę (lub unię).

3.2.3. Wartości zwracane przez wywoływaną funkcję

W obu rodzajach odwołań FPL i VPL sposób zwracania wartości przez wywoływany podprogram jest podobny:

- wartość skalarna jest zwracana za pośrednictwem rejestru (rejestrów) mikroprocesora,
- wartość zmiennoprzecinkowa jest zwracana poprzez wierzchołek stosu koprocesora,
- struktura lub unia jest zwracana poprzez jej skopiowanie do pamięci pod adres określony ostatnim parametrem wywołania podprogramu.

W przypadku zwracania wartości za pośrednictwem rejestru (rejestrów) mikroprocesora oba odwołania FPL i VPL część wartosci zwracają poprzez ten sam rejestr (rejestry), ale różnice występują jeśli zwracaną wartością jest pointer. Poniższa tabela określa jakie wartości są zwracane przy poszczególnych rodzajach odwołań:

Zwracana wartość:	FPL:	VPL:
8-bit integer	AL	AL
16-bit integer	AX	AX
32-bit integer	DX:AX	DX:AX
NEAR-pointer	BX	AX
FAR-pointer	ES:BX	DX:AX

3.2.4. Zachowywanie i odzyskiwanie zawartości rejestrów mikroprocesora

Dana procedura, skompilowana przy pomocy kompilatora iC-86 zachowuje (odkłada na stos mikroprocesora) zawartość niektórych rejestrów, które po jej zakończeniu są następnie odtwarzane. O tym, zawartość których rejestrów będzie zachowana decyduje przyjęty rodzaj odwołania do danej procedury, a więc jeden z parametrów jej kompilacji (FPL lub VPL). Przy użyciu kompilatora z opcją VLP (zmienna lista parametrów) zachowane będą m.in. rejestry DI i SI, dla zachowania zgodności z wcześniejszymi kompilatorami języka C firmy Intel.

1. Konwencja ze stałą listą parametrów - FLP

   - rejestry, których zawartość jest zachowywana: BP, SP, CS, DS, SS
   - rejestry, których zawartość jest niszczona: AX, BX, CX, DX, DI, SI, ES

2. Konwencja ze zmienną listą parametrów - VLP

   - rejestry, których zawartość jest zachowywana: BP, SP, DI, SI, CS, DS, SS
   - rejestry, których zawartość jest niszczona: AX, BX, CX, DX, ES

3.2.5. Zakończenie procedury skompilowanej przez kompilator iC-86

W przypadku funkcji skompilowanej z opcją FLP wywoływana funkcja zdejmuje ze stosu wszystkie parametry (bilansuje stos) przed przekazaniem sterowania do programu nadrzędnego.

W przypadku funkcji skompilowanej z opcją VLP bilansowanie stosu o wartość wynikającą z liczby parametrów przekazanych do procedury wywoływanej wykonuje procedura nadrzędna, po przekazaniu do niej sterowania z podprogramu wywoływanego.

W obu przypadkach funkcja wywoływana zdejmuje wszystkie argumenty zmiennoprzecinkowe ze stosu koprocesora arytmetycznego i zachowuje je jako zmienne lokalne. Jeśli funkcja wywoływana zwraca wartość zmiennoprzecinkową do programu nadrzędnego, to umieszcza ją na szczycie stosu koprocesora.

3.2.6. Podział programu pomiędzy poszczególne obszary pamięci

Każda procedura napisana w języku C składa się z kilku elementów, które mogą być umieszczane w różnych miejscach pamięci operacyjnej sterownika. Elementami tymi są:

- kod programu w postaci instrukcji wykonywanych przez mikroprocesor realizujących czynności wykonywanych przez daną procedurę,
- zmienne przechowywane w pamięci inicjowane i modyfikowane przez daną procedurę,
- stałe przechowywane w pamięci, których wartość nie ulega zmianie podczas wykonywania programu (np. różne stałe tekstowe),
- elementy służące do transferu danych pomiędzy procedurami, do przechowywania stanu poszczególnych rejestrów mikroprocesora etc.

W przypadku uruchamiania programu składającego się tylko z procedur napisanych w języku C w środowisku sterownika (komputera) wyposażonego w system operacyjny użytkownik w zasadzie nie potrzebuje informacji o miejscu, w którym kompilator umieści wymienione elementy, gdyż wykonuje to za niego system operacyjny. W przypadku "gołego" sterownika i programu złożonego z procedur pisanych w różnych językach należy pamiętać o kryteriach podziału poszczególnych elementów całego programu. Kryteria te zależą od przyjętego modelu pamięci, w jakim napisano dany program.

Kompilator iC-86, podobnie jak inne kompilatory języka C, może używać czterech rodzajów modeli pamięci. Są to: SMALL, COMPACT, MEDIUM i LARGE. Charakteryzują się one m.in.

- w modelu SMALL cały program zajmuje nie więcej 128 kilobajtów pamięci, z czego jeden 64-kilobajtowy segment jest przeznaczony na kod programu, a drugi segment - na stałe, zmienne i stos. W modelu SMALL kompilator iC-86 do zmiany sterowania wewnątrz programu używa tylko instrukcji skoków i odwołań wewnątrzsegmentowych (czyli typu NEAR).
- w modelu COMPACT cały program zajmuje nie więcej 192 kilobajtów pamięci, z czego jeden 64-kilobajtowy segment jest przeznaczony na kod programu, drugi - na stałe i zmienne, a trzeci - na stos. W modelu COMPACT kompilator iC-86 do zmiany sterowania wewnątrz programu używa tylko instrukcji skoków i odwołań wewnątrzsegmentowych (czyli typu NEAR).
- w modelu MEDIUM kod programu może zajmować cały dostępny obszar pamięci, ale stałe, zmienne i stos - tylko pojedynczy, 64-kilobajtowy segment. W modelu MEDIUM tym kompilator iC-86 do zmiany sterowania wewnątrz programu używa tylko instrukcji skoków i odwołań międzysegmentowych (czyli typu FAR).
- w modelu LARGE długość kodu programu oraz obszaru przeznaczonego na stałe, zmienne i stos jest ograniczona tylko wielkością dostępnej pamięci. W modelu LARGE kompilator iC-86 do zmiany sterowania wewnątrz programu używa tylko instrukcji skoków i odwołań międzysegmentowych (czyli typu FAR).

Przy tworzeniu programu złożonego z procedur pisanych w języku C i asemblerze umieszczanych w wielu plikach możliwe jest "mieszanie" modeli pamięci, ale nie jest to zalecane z uwagi na duże prawdopodobieństwo popełnienia błędu. Z punktu widzenia szybkości działania i długości kodu wynikowego najbardziej efektywny jest model SMALL i COMPACT, ale złożoność programu sterującego robotów URp nie

pozwala na ich zastosowanie (model SMALL był stosowany w programie sterującym robotów IRp, wyposażonych w pierwszy panel programowania opracowany w PIAP w połowie lat osiemdziesiątych). Ponadto długość obszaru przeznaczonego na program aplikacyjny robota nie pozwala na stosowanie modelu MEDIUM. Dlatego program sterujący robotów URp w obecnej wersji został wykonany w modelu LARGE.

3.2.7. Ustalenia w zakresie procedúr pisanych w języku C przyjęte w programie sterującym robotów URp

Program sterujący robotów URp został napisany w tzw. dużym (LARGE) modelu pamięci z przyjęciem odwołań typu FLP (Fixed List Parameter) pomiędzy poszczególnymi podprogramami. Programista tworzący program tylko na poziomie języka C w zasadzie jest zwolniony z przestrzegania zasad przekazywania parametrów do procedury wywoływanej, bilansowania stosu przez lub po powrocie do programu nadrzędnego, pamiętania w którym rejestrze są zwracane wyniki działania podprogramu etc, ponieważ wszystkie te problemy rozwiązuje za niego kompilator. Także ewentualna zmiana modelu pamięci lub sposobu wykonywania odwołań jest wykonywana na etapie samej kompilacji pliku źródłowego bez potrzeby ingerowania w samą treść programu. Tym niemniej należy pamiętać o kilku szczegółach wymuszonych przez ustalenia istotne na etapie konsolidacji (ang. linking) i lokowania (ang. locating) całego programu. Są nimi:

1. Kompilator iC-86 kompilując w dużym (LARGE) modelu pamięci podprogram (podprogramy) zawarte w danym pliku o nazwie *module* (i rozszerzeniu .C lub .C86) tworzy trzy segmenty o następujących nazwach:

   - segment *module*_CODE w którym umieszcza kod programu i ewentualnie stałe (o ile kompilator uruchomiono z opcją ROM),

   - segment *module*_DATA w którym umieszcza wszystkie zmienne i ewentualnie stałe (o ile kompilator uruchomiono z opcją RAM),

   - segment STACK przeznaczony na stos programu.

   Ponieważ w przypadku programu zapisanego w wielu modułach byłoby kłopotliwe "indywidualne" lokowanie w pamięci poszczególnych segmentów (patrz opis programu lokatora LOC86), toteż kompilator iC-86 segmentowi *module_CODE* nadaje dodatkowo klasę 'CODE', a segmentowi *module_DATA* - klasę 'DATA'.

2. Definiując stałą w programie źródłowym napisanym w języku C (w dużym modelu pamięci) można bez względu na parametr kompilacji (RAM lub ROM) umieszczać ją w segmencie kodu programu deklarując tę stałą jako *"const"* (lub *"readonly"*). Od zasady tej jest jednak wyjątek: Załóżmy, że w danym programie wykonano następującą deklarację:

```
typedef int (*PROCEDURY_OBSLUGI)();
const static PROCEDURY_OBSLUGI *tablica_procedur_obslugi[] = {
    procedura_1,
    procedura_2};
```

   Postać ta jest wygodna z punktu widzenia przejrzystosci i prostoty zapisu, ale niestety kompilator iC-86 bez względu na deklarację *"const"* i parametr kompilacji (ROM lub RAM) zawsze umieszcza deklarowaną tablicę w segmencie danych. Pociąga to za sobą konieczność dopisywania dodatkowego podprogramu inicjujacego stosowny fragment pamięci RAM w przypadku umieszczenia kodu programu w pamięci EPROM. Wyjściem z tego jest wykonanie deklaracji tablicy pointerów do funkcji w języku asemblera (patrz pliki decinspr.a86,

edit_tab.a86, mano_tab.a86, othe_tab.a86 w katalogu CTRLPGM\PROGPNL jego podkatalogach).

3. Konsolidator LINK86 (opisany w dalszej części tego rozdziału) jest programem bardzo rygorystycznym ze względu na przestrzeganie zgodności typów wszystkich procedur i zmiennych, które scala ("linkuje"). W języku C wszystkie procedury nie zdefiniowane w danym pliku, a występujące (wywoływane) w treści programu definiowanego w tym pliku są traktowane jako procedury typu *extern* i w zasadzie nie wymagają umieszczenia ich deklaracji. W takim przypadku są jednak traktowane jako podprogramy zwracające wartość typu int, a zatem jeśli uprzednio zdefiniowano, że zwracają inną wartość (np. unsigned int), to program konsolidatora LINK86 wyświetla informację o błędzie (tzw. warning). Dlatego ze względu na potencjalne problemy z konsolidatorem LINK86 zaleca się umieszczanie deklaracji danej funkcji (zdefiniowanej w innym pliku) w module (zbiorze) w którym jest ona wykorzystywana wraz z określeniem wartości jaką ona zwraca. Np. jeśli funkcja "func()" została zdefiniowana w jednym pliku jako funkcja:

```
unsigned char func()
```

to w pliku w którym jest wykorzystywana musi zostać zadeklarowana jako:

```
extern unsigned char func();
```

Podobnie należy przestrzegać zgodności typów definicji i deklaracji zmiennych globalnych definiowanych i wykorzystywanych w różnych plikach.

4. Wywołanie dowolnego podprogramu (funkcji) napisanej w innym języku niż C z poziomu programu napisanego w języku C jest identyczne jak wywołanie podprogramów pisanych w C. Pewne wątpliwości mogą się pojawić przy deklarowaniu (patrz punkt 3) zewnętrznego podprogramu napisanego w języku asemblera, ponieważ w asemblerze nie określa się typu wartości jaką funkcja ta ma zwracać. Należy wówczas konsekwentnie we wszystkich plikach źródłowych, napisanych w języku C w których funkcja ta jest wykorzystywana jednakowo ją deklarować, bez względu na to, czy jest to praktycznie uzasadnione. Np. jeśli dana funkcja asemblerowa "func()" w każdym przypadku zwraca znak o kodzie ASCII nie większym niż 127, to z punktu widzenia nadrzędnego programu napisanego w języku C deklaracje:

```
extern char func();
extern unsigned char func();
```

mają praktyczne to samo znaczenie, ale umieszczone w osobnych plikach spowodują wyświetlenie informacji o błędzie podczas konsolidowania całego programu (ang. linking, patrz opis programu LINK86.EXE).

3.2.8. Wywołanie kompilatora iC-86

Wszystkie pliki programu sterującego robotów URp zawierającego teksty procedur pisanych w języku C mają rozszerzenie .C86. Kompilator iC-86, uruchamiany komendą o formacie jak poniżej, tworzy dwa zbiory wyjściowe, oba o takich samych nazwach jak plik źródłowy, ale o następujących rozszerzeniach:

- .OBJ - zawierający skompilowaną postać zbioru źródłowego. Plik o rozszerzeniu .OBJ nie jest tworzony (w przeciwieństwie do działania asemblera ASM-86), jeśli kompilowany zbiór zawierał jakiekolwiek błędy formalne (składniowe),

- .LST - zawierający informacje o wykonanej kompilacji (także o ewentualnych błędach) i dodatkowo (opcjonalnie) listing programu z możliwością jego rozwinięcia na postać asemblerową (patrz opis parametru CODE w podręczniku opisującym kompilator iC-86).

Format linii wywołania kompilatora iC-86, używany do przekształcania wszystkich procedur programu sterującego robotów URp napisanych w języku C ma następującą postać:

ic86 *nazwa*.c86 LARGE ROM MOD186 FIXEDPARAMS NOALIGN

Poszczególne parametry występujące w linii wywołania mają następujące znaczenie:

nazwa - określa nazwę wraz z rozszerzeniem pliku źródłowego zawierającego podprogram napisany w języku C,

LARGE - oznacza wybór dużego (LARGE) modelu pamięci,

ROM - oznacza, że stałe będą umieszczane w segmencie kodu programu. Nie należy sugerować się nazwą tego parametru w przypadku gdy program w celu testowania ma być w całości ładowany do pamięci RAM (także należy używać opcji ROM),

MOD186 - oznacza zezwolenie na umieszczanie w kodzie programu instrukcji mikroprocesora 80186,

FIXEDPARAMS - wymusza kompilację programu źródłowego z odwołaniami do funkcji pomocniczych wykonywanych ze stałą listą parametrów (FPL - fixed parameter list),

NOALIGN - domyślnie kompilator iC-86 rezerwując miejsce w pamięci na poszczególne elementy danej struktury umieszcza je w ten sposób, aby adres najmniej znaczącego bajtu każdego takiego elementu był parzysty, nawet jeśli poprzedni element zawierał nieparzystą liczbę bajtów. Pozwala to na szybsze odczytywanie zawartości pamięci, ale jednocześnie struktura może zawierać niewykorzystywane bajty pomiędzy swoimi elementami. Opcja NOALIGN zakazuje kompilatorowi przyporządkowywania poszczególnym elementom struktur parzystych adresów, przez co elementy te bezpośrednio sąsiadują ze sobą. Z punktu widzenia programu pisanego w języku C efektywniejsze byłoby pominięcie tej opcji, a głównym powodem jej stosowania jest potrzeba uniknięcia rozbudowanych konstrukcji i potencjalnych błędów w adresowaniu wykonywanym z poziomu procedur napisanych w języku asemblera.

Opisane parametry wywołania kompilatora iC-86 nie wyczerpują listy wszystkich możliwych parametrów. Za pozostałe są więc przyjmowane ich wartości domyślne lub nie mają one znaczenia z punktu widzenia kompilacji poszczególnych procedur programu sterującego robotów URp. Dodatkowo w niektórych programach przetwarzania wsadowego, które kompilują zawartość całego katalogu mogą być używane następujące parametry:

NOPRINT - zakazuje tworzenia zbioru wyjściowego .LST,

NOLIST - zakazuje umieszczania w zbiorze wyjściowym .LST listingu programu źródłowego (plik ten będzie zawierał tylko informacje o wykonanej kompilacji).

3.3. Asembler ASM-86

3.3.1. Zasady pisania procedur programu sterującego robotów URp w języku asemblera ASM-86

Procedury napisane w języku asemblera mikroprocesora 8086/80186 muszą uwzględniać już na etapie ich tworzenia przyjęty sposób przekazywania parametrów do podprogramu i model pamięci. Innymi słowy: zmiana modelu pamięci i konwencji przekazywania parametrów nie jest wykonywana przez zmianę parametrów wywołania programu asemblera ASM-86 lecz wymaga zmian w tekście źródłowym samej procedury. Jedyną opcją, jaką można sterować (jednak w ograniczonym zakresie) formatem postaci wynikowej jest wybór mikroprocesora na którym procedura ta będzie wykonywana.

Podprogram asemblerowy wymaga znacznie większej uwagi w trakcie jego tworzenia niż podprogram pisany w języku C, ponieważ programista musi także zwracać uwagę na zagadnienia, które w przypadku języka C rozwiązuje kompilator. Są nimi:

1. Każdy program asemblerowy musi rozpoczynać się dyrektywą NAME o następującym formacie:

   NAME nazwa_modulu

   Obligatoryjny parametr "nazwa_modulu" jest wykorzystywany jako nazwa identyfikująca zawartość modułu (pliku) na etapie dołączania do danej biblioteki wszystkich obiektów typu PUBLIC zdefiniowanych w tym pliku. Parametr "nazwa_modulu" może być dowolny i nie ma on nic wspólnego z nazwą pliku z punktu widzenia nazewnictwa plików w systemie operacyjnym MS DOS. W programie sterującym robotów URp wskazane jest jednak, aby był on identyczny jak nazwa pliku (bez rozszerzenia) zawierającego dany program asemblerowy. Wynika to z następującej przyczyny: Usuwanie danego modułu z biblioteki (patrz opis programu LIB86.EXE), który zdefiniowano w języku asemblera polega m.in. na podaniu parametru "nazwa_modulu", a nie "nazwa_zbioru". Jeśli "nazwa_modulu" jest identyczna jak "nazwa zbioru" o wiele łatwiej jest utworzyć program przetwarzania wsadowego, który automatycznie usuwa dany moduł z biblioteki, a następnie dołącza jego zmodyfikowaną wersję. Dołączanie modułu do danej biblioteki polega na podaniu "nazwy_zbioru", ale w bibliotece obiekty zdefiniowane w tym pliku będą zapisane pod "nazwą_modulu", a nie "nazwą_zbioru". Ponadto należy unikać określenia "nazwa_modulu" identycznego jak "nazwa_zbioru" jakiegokolwiek pliku zawierającego tekst programu w języku C, ponieważ kompilator iC-86 obligatoryjnie przyjmuje za "nazwę_modulu" "nazwę_zbioru" (do biblioteki

2. Jak wyjaśniono w ustaleniach dotyczących zasadami posługiwania się kompilatorem iC-86 przekształcając program źródłowy w dużym (LARGE) modelu pamięci tworzy on dwa segmenty: segment kodu programu - zawierający zakodowaña postać samego programu i stałe oraz segment danych - zawierający zmnienne. Aby zachować tę zasadę także w stosunku do procedur asemblerowych przyjęto, że w każdym pliku definiowane są co najwyżej tylko dwa segmenty (oprócz pliku restart.a86, gdzie dodatkowo zdefiniowano segment stosu), o następującej postaci:

   ```
   name_DATA   SEGMENT   PUBLIC    'DATA'
   ..........................................
        definicje zmiennych
   ..........................................
   name_DATA   ENDS
   ```

```
name_CODE   SEGMENT   WORD PUBLIC 'CODE'
            ASSUME    CS:name_CODE
.....................................
     definicje stałych
     program
.....................................
name_CODE   ENDS
            END
```

gdzie określenie *"name"* jest identyczne jak nazwa zbioru, w którym zdefiniowano dany segment. Z tych samych powodów, co w przypadku kompilatora iC-86 (prostsza forma wywołania programu lokatora LOC86) dodatkowo każdemu segmentowi nadano osobną klasę, odpowiednio: *name*_DATA - klasę 'DATA', *name*_CODE - klasę 'CODE'. Należy tutaj zwrócić uwagę na jeszcze jeden fakt: Tylko w przypadku segmentu *name*_CODE jest określone jaki rejestr segmentowy adresuje zawartość tego segmentu (dyrektywa ASSUME CS:*name*_CODE). Jest to spowodowane koniecznością określenia, który rejestr segmentowy bierze udział w adresowaniu wszelkich instrukcji skoków wewnątrz procedury. Adresowanie zmiennych i stałych w dużym modelu pamięci jest wykonywane zawsze indywidualnie w odniesieniu do danego obiektu, tzn. przed odwołaniem się do tego obiektu najpierw ustala się zawartość tego rejestru segmentowego, który ma zawierać paragraf adresu (w przypadku stałych umieszczonych w segmencie kodu nie zawsze jest to rejestr CS, ale wynika to z techniki programowania, a nie z narzuconych ustaleń).

3. W języku asemblera mikroprocesora 8086/80186 w przypadku użycia obiektu (podprogramu, zmiennej, stałej) zewnętrznego obligatoryjnie musi wystąpić deklaracja EXTRN tego obiektu. Często popełnianym błędem jest umieszczanie tej deklaracji wewnątrz definicji segmentu, w którym obiekt ten jest wykorzystywany. Błąd ten jest trudny do wykrycia, ponieważ nie jest to błąd składniowy, a więc sygnalizowany przez program asemblera ASM-86. W związku z tym, we wszystkich plikach "asemblerowych" programu sterującego robotów URp konsekwentnie wszelkie deklaracje EXTRN są umieszczane pomiędzy dyrektywą NAME a początkiem definicji pierwszego segmentu.

4. Asembler ASM-86 pozwala na wywoływanie z poziomu programu napisanego w tym języku dowolnego podprogramu napisanego w innym języku. Sposób wywołania wymusza jednak kompilator, którym skompilowano wywoływany podprogram. W dotychczasowych pracach nad oprogramowaniem robotów przemysłowych prowadzonych w PIAP z języków wysokiego poziomu stosowano tylko język C (a także PLM-86) do którego uprzednio wykorzystywano dwa rodzaje kompilatorów: MWC-86 zawartego w systemie GENESIS i Microsoft C v.6.00 (ten ostatni w ograniczonym zakresie). Oba z nich charakteryzowały się tym, że do nazwy definiowanego podprogramu lub zmiennej o zasięgu globalnym dopisywały znak "_" za nazwą (kompilator MWC) lub przez nazwą (kompilator Microsoft C). Ponieważ znak "_" był także automatycznie dopisywany przez kompilator w instrukcji odwołania do danego obiektu pisanej na poziomie języka C, to znak ten nie musiał być umieszczany w odwołaniach w tym języku, ale był wymagany, jeśli odwołanie wykonywano z poziomu asemblera. Na przykład, jeśli w języku C zdefiniowano funkcję "func()" (którą skompilowano przy pomocy kompilatora MWC-86, to jej wywołanie z poziomu aseblera miało następującą postać:

```
EXTRN func_: FAR
.................
CALL FAR PTR func_
```

Obecnie stosowany kompilator iC-86 języka C nie dopisuje żadnego znaku ani przed, ani po nazwie definiowanego obiektu, a zatem wywołanie procedury "func()" skompilowanej przez iC-86 z poziomu asemblera ma postać jak poniżej:

```
EXTRN func: FAR
.....
CALL FAR PTR func
```

5. Parametry do wszystkich procedur asemblerowych programu sterującego robotów URP są zawsze przekazywane poprzez stos mikroprocesora (w programie sterującym nie występuje przypadek przekazywania parametru zmiennoprzecinkowego do procedury asemblerowej, który wymuszałby użycie stosu koprocesora 8087). Ponieważ w przypadku procedur pisanych w języku C przyjęto, że wszelkie odwołania do podprogramów są wykonywane ze stałą listą parametrów (FPL), to ze względu na kompatybilność zasadę tę zachowano także w stosunku do procedur asemblerowych. Ponadto w przypadku programów asemblerowych odwołania typu FLP łatwiej pozwalają unikać błędów związanych z bilansowaniem stosu po przekazaniu sterowania do procedury nadrzędnej.

Załóżmy, że w języku C użyto funkcji func() zadeklarowanej w dużym modelu pamięci jako:

```
unsigned int func (int i, char c);
```

Poniżej przedstawiono sposób wywołania tej funkcji w przypadku odwołania ze stałą (FPL) i zmienną (VPL) listą parametrów, oraz jakie instrukcje powinny zostać umieszczone w ciele podprogramu *func*:

Odwołanie ze stałą listą parametrów (FPL):

```
        PUSH ...              ; Słowo zawierające parametr "i"
        PUSH ...              ; Słowo, którego młodszy bajt zawiera
                              ; parametr "c"
        CALL FAR PTR func     ; Wywołanie podprogramu func
.....................................................

func PROC    FAR  ; Początek definicji procedury func
     PUBLIC  func

     PUSH BP
     MOV  BP, SP
     PUSH ...      ; Ciąg instrukcji odkładających na stos zawartość
                   ; wszystkich rejestrów mikroprocesora
                   ; wykorzystywanych w podprogramie func
     .....
     POP  ...      ; Ciąg instrukcji zdejmujących ze stosu zawartość
                   ; wszystkich rejestrów mikroprocesora
                   ; wykorzystywanych w podprogramie func
     POP  BP
     RET  4        ; Instrukcja przekazania sterowania do podprogramu
                   ; nadrzędnego z jednoczesnym zmodyfikowaniem
                   ; wskaźnika stosu o wartość wynikającą z wielkości
                   ; stosu zajętego przez parametry. Rejestr AX zawiera
                   ; wartość typu unsigned int przekazywaną do
                   ; podprogramu nadrzędnego
func ENDP          ; Koniec definicji procedury func
```

Odwołanie ze zmienną listą parametrów (VPL):

```
        PUSH ...              ; Słowo zawierające parametr "i"
        PUSH ...              ; Słowo, którego młodszy bajt zawiera
                              ; parametr "c"
        CALL FAR PTR func     ; Wywołanie podprogramu func
        ADD  SP, 4            ; Modyfikacja wskaźnika stosu o wartość
                              ; wynikającą z wielkości stosu zajętego
                              ; przez parametry
    ..........................................................

    func PROC    FAR  ; Początek definicji procedury func
         PUBLIC  func

         PUSH BP
         MOV  BP, SP
         PUSH ...      ; Ciąg instrukcji odkładających na stos zawartość
                       ; wszystkich rejestrów mikroprocesora
                       ; wykorzystywanych w podprogramie func
         ........
         POP  ...      ; Ciąg instrukcji zdejmujących ze stosu zawartość
                       ; wszystkich rejestrów mikroprocesora
                       ; wykorzystywanych w podprogramie func
         POP  BP
         RET           ; Instrukcja przekazania sterowania do podprogramu
                       ; nadrzędnego. Rejestr AX zawiera wartość typu
                       ; unsigned int przekazywaną do podprogramu
                       ; nadrzędnego
    func ENDP          ; Koniec definicji procedury func
```

W obu przypadkach rejestr BP pozwala zaadresować na stosie następujące wartości:

```
WORD PTR SS:[BP + 6] - słowo zawierające parametr "i",
WORD PTR SS:[BP + 6] - słowo, którego młodszy bajt zawiera parametr "c",
WORD PTR SS:[BP + 4] - paragraf adresu kolejnej instrukcji po instrukcji
                       wywołania podprogramu func,
WORD PTR SS:[BP + 2] - offset adresu kolejnej instrukcji po instrukcji
                       wywołania podprogramu func,
WORD PTR SS:[BP]     - "stara" zawartość rejestru BP.
```

6. We wszystkich procedurach pisanych w języku asemblera ASM-86 programu sterującego robotów URp są przestrzegane następujące zasady dotyczące zachowywania zawartości rejestrów mikroprocesora:

   - każda procedura zachowuje na stosie, a następnie odtwarza zawartość wszystkich modyfikowanych przez siebie rejestrów, za wyjątkiem rejestru wskaźników, oraz tych rejestrów, które służą do przekazywania informacji do podprogramu nadrzędnego (patrz także rozdział dotyczący kompilatora iC-86 pt. "Wartości zwracane przez wywoływaną funkcję"),

   - jeśli dana procedura asemblerowa wykorzystuje (wywołuje) funkcję zdefiniowaną w języku C lub dowolną funkcję biblioteczną z pakietu oprogramowania narzędziowego firmy INTEL, to przed instrukcją wywołania tej funkcji (ewentualnie przed instrukcjami przekazania parametrów wywołania) zachowuje, a potem odtwarza zawartość wszystkich rejestrów mikroprocesora za wyjątkiem: CS, IP, SS, SP, rejestru wskaźników oraz tych rejestrów, które będą służyć do przekazania informacji z tej funkcji,

   - jeśli dana procedura asemblerowa wykorzystuje (wywołuje) funkcję

zdefiniowaną także w języku asemblera C, to przed przekazaniem sterowania do tej funkcji nie zachowuje zawartości żadnego rejestru (przyjmuje się, że wykona to podprogram wywoływany) - od reguły tej mogą być jednak nieliczne wyjątki.

3.3.2. Wywołanie programu asemblera ASM-86

Wszystkie pliki programu sterującego robotów URp zawierającego teksty procedur pisanych w języku asemblera mikroprocesora 8086/80186 mają rozszerzenie .A86. Asembler ASM-86, uruchamiany komendą o formacie jak poniżej, tworzy dwa zbiory wyjściowe, oba o takich samych nazwach jak plik źródłowy, ale o następujących rozszerzeniach:

- .OBJ - zawierający skompilowaną postać zbioru źródłowego. Plik o rozszerzeniu .OBJ zostaje utworzony zawsze, bez względu za zawartość z zbiorze źródłowym błędów formalnych (składniowych),
- .LST - zawierający informacje o wykonanej asemblacji (także o ewentualnych błędach) i dodatkowo listing programu.

Format linii wywołania kompilatora iC-86, używany do przekształcania wszystkich procedur programu sterującego robotów URp napisanych w języku C ma następującą postać:

asm86 *nazwa*.a86 MOD186

Poszczególne parametry występujące w linii wywołania mają następujące znaczenie:

nazwa - określa nazwę wraz z rozszerzeniem pliku źródłowego zawierającego podprogram napisany w języku C,

MOD186 - oznacza zezwolenie na umieszczanie w kodzie programu instrukcji mikroprocesora 80186.

Oprócz parametru MOD186 program ASM86 zawiera także szereg innych, które w większości służą do modyfikowania pliku wyjściowego .LST. Dodatkowo w niektórych programach przetwarzania wsadowego, które kompilują zawartość całego katalogu mogą być używane następujące inne parametry:

NOPRINT - zakazuje tworzenia zbioru wyjściowego .LST,

NOLIST - zakazuje umieszczania w zbiorze wyjściowym .LST listingu programu źródłowego (plik ten będzie zawierał tylko informacje o wykonanej asemblacji).

3.4. Program LIB86 - tworzenie bibliotek

Pliki źrodłowe programu sterującego robotów URp zostały podzielone pomiędzy kilkadziesiąt podkatalogów, co znacznie ułatwia wprowadzanie modyfikacji, ale jednocześnie utrudnia wykonywanie konsolidacji (ang. linking) całego programu. W celu uproszczenia czynności związanych z konsolidacją zdecydowano się na rozwiązanie w którym wszystkie pliki o rozszerzeniu .OBJ, będące efektem wykonania kompilacji są łączone w ramach danego podkatalogu w pojedynczą bibliotekę o rozszerzeniu .LIB. Do tworzenia biblioteki został wykorzystany

program LIB86.EXE z pakietu oprogramowania narzędziowego firmy INTEL.

Program LIB86.EXE pozwala tworzyć, modyfikować i wyświetlać zawartość danej biblioteki. Może on pracować w trybie wykonywania pojedynczych komend wpisywanych przez użytkownika po zgłoszeniu przez LIB86 znakiem "*" swojej gotowości lub w trybie wykonywania kolejnych komend odczytywanych z osobnego pliku (na zasadzie funkcji systemu MS DOS określanej jako tzw. wtórna definicja standardowego urządzenia wejściowego). W obu tych trybach dostępne są następujące komendy:

ADD - dodawanie modułów do biblioteki,
CREATE - tworzenie biblioteki (utworzenie pliku do którego będą dopisywane kolejne moduły),
DELETE - usunięcie danego modułu z biblioteki,
EXIT - zakończenie wykonywania programu LIB86.EXE i przekazanie sterowania do systemu operacyjnegpo,
LIST - wyświetlenie zawartości danej biblioteki.

Ponieważ "ręczna" modyfikacja biblioteki w danym katalogu jest dość kłopotliwa, toteż utworzono odpowiednie programy przetwarzania wsadowego, które maksymalnie ją upraszczają. Modyfikacja biblioteki może polegać albo na utworzeniu nowej biblioteki ze wszystkich plików źródłowych zawartych w danym podkatalogu, albo na dodaniu jednego lub kilku modułów do biblioteki już istniejącej. W obu przypadkach mechanizm działania jest podobny i polega na wykorzystaniu m.in. edytora ekranowego KEDIT.

3.4.1. Tworzenie nowej biblioteki ze wszystkich plików źródłowych zawartych w danym podkatalogu

W każdym podkatalogu zawierającym pliki źródłowe programu sterującego robotów URp znajduje się program przetwarzania wsadowego LIBRARY.BAT, którego zadaniem jest wywołanie z odpowiednim parametrem innego programu przetwarzania wsadowego, a mianowicie _LIBRARY.BAT, zapisanego w podkatalogu c:\INTEL\UTIL (katalog c:\INTEL zawiera całe oprogramowanie narzędziowe firmy INTEL). Parametrem wywołania programu _LIBRARY.BAT jest nazwa biblioteki, a jego działanie polega na wykonaniu m.in. następujących czynności:

1. Utworzenie pliku _1.BAT zawierającego spis wszystkich zbiorów zawartych o rozszerzeniach .A86, .C86 i .P86 w danym podkatalogu(a więc spisu wszystkich zbiorów zawierających postać źródłową procedur napisanych w języku asemblera, C oraz PL/M, ten ostatni został już wyeliminiowany z programu sterującego robotów URp). Plik _1.BAT jest tworzony komendą DIR systemu operacyjnego MS DOS, ale uruchamianą z opcjami /B/O:ne dostępnymi od wersji 5.00 tego systemu.

2. Automatyczne uruchomienie edytora ekranowego KEDIT.EXE z parametrem _1.BAT. Przedtem do podkatalogu w którym ma być zbudowana biblioteka zostaje skopiowany zbiór c:\INTEL\UTIL_profile.ked. Po zmianie nazwy tego pliku na PROFILE.KED staje się on plikiem konfiguracyjnym dla edytora KEDIT, zawierającym makroinstrukcje umożliwiające utworzenie następujących dwóch zbiorów:

   - zmodyfikowanej wersji zbioru _1.BAT, w której do istniejących nazw zbiorów o rozszerzeniach .A86, .C86 i .P86 zostają dopisane wywołania odpowiednio asemblera ASM-86 i kompilatorów: iC-86 oraz PL/M-86 (wraz z odpowiednimi parametrami),

- zbioru _1.CON zawierającego ciąg komend dla programu bibliotekarza LIB86.EXE.

Zainicjowanie tworzenia tych plików następuje po wciśnięciu klawisza ESC na klawiaturze komputera w momencie, kiedy edytor ekranowy KEDIT otworzy okno z "pierwotną" wersją zbioru _1.BAT.

3. Zainicjowanie wykonania programu przetwarzania wsadowego _1.BAT, który wykona kompilację wszystkich plików o rozszerzeniach: .A86, .C86 i P86.

4. Uruchomienie programu bibliotekarza LIB86.EXE w trybie wykonywania komend zawartych w pliku _1.CON:

   LIB86.EXE < _1.CON

   Tworzy on żądaną bibliotekę.

Opisane powyżej algorytm wymaga od użytkownika uruchomienia programu LIBRARY.BAT i zainicjowania wykonania odpowiednich makroinstrukcji edytora ekranowego KEDIT (niestety w obecnie stosowanej wersji tego edytora nie posiada on mechanizmu automatycznego wykonywania komend zawartych w osobnym pliku bez dodatkowego udziału użytkownika). Należy ponadto zwrócić uwagę na brak mechanizmu przerywającego wykonywanie algorytmu po wystąpieniu błędu kompilacji w jednym z kompilowanych plików. Poniżej przedstawiono przykład obu utworzonych plików dla podkatalogu CTRLPGM\:

Program przetwarzania wsadowego _1.BAT kompilujący
wszystkie zbiory z rozszerzeniem .A86 i .C86:

```
asm86.exe RESTART.A86 MOD186 NOPRINT
ic86.exe AXESCNTR.C86 MOD186 LARGE ROM NOALIGN FIXEDPARAMS NOPRINT
ic86.exe ERROR.C86 MOD186 LARGE ROM NOALIGN FIXEDPARAMS NOPRINT
ic86.exe ERRTXT.C86 MOD186 LARGE ROM NOALIGN FIXEDPARAMS NOPRINT
ic86.exe GLOBVAR.C86 MOD186 LARGE ROM NOALIGN FIXEDPARAMS NOPRINT
ic86.exe GRIPOPER.C86 MOD186 LARGE ROM NOALIGN FIXEDPARAMS NOPRINT
ic86.exe HARDERR.C86 MOD186 LARGE ROM NOALIGN FIXEDPARAMS NOPRINT
ic86.exe INITMAIN.C86 MOD186 LARGE ROM NOALIGN FIXEDPARAMS NOPRINT
ic86.exe MAIN.C86 MOD186 LARGE ROM NOALIGN FIXEDPARAMS NOPRINT
ic86.exe SETSYSST.C86 MOD186 LARGE ROM NOALIGN FIXEDPARAMS NOPRINT
ic86.exe SWTCHEXT.C86 MOD186 LARGE ROM NOALIGN FIXEDPARAMS NOPRINT
ic86.exe SYSTEM.C86 MOD186 LARGE ROM NOALIGN FIXEDPARAMS NOPRINT
ic86.exe TIMELAG.C86 MOD186 LARGE ROM NOALIGN FIXEDPARAMS NOPRINT
ic86.exe VERSION.C86 MOD186 LARGE ROM NOALIGN FIXEDPARAMS NOPRINT
```

Plik _1.CON zawierający zbiór komend dla programu bibliotekarza LIB86.EXE dla utworzenia biblioteki ze wszystkich skompilowanych plików:

```
create library.lib
ADD RESTART.obj TO library.lib
ADD AXESCNTR.obj TO library.lib
ADD ERROR.obj TO library.lib
ADD ERRTXT.obj TO library.lib
ADD GLOBVAR.obj TO library.lib
ADD GRIPOPER.obj TO library.lib
ADD HARDERR.obj TO library.lib
ADD INITMAIN.obj TO library.lib
ADD MAIN.obj TO library.lib
ADD SETSYSST.obj TO library.lib
ADD SWTCHEXT.obj TO library.lib
ADD SYSTEM.obj TO library.lib
ADD TIMELAG.obj TO library.lib
ADD VERSION.obj TO library.lib
exit
```

Program bibliotekarza LIB86.EXE tworzy bibliotekę o nazwie LIBRARY.LIB, której nazwa zostaje następnie zamieniona na MAIN.LIB przez program _LIBRARY.BAT.

3.4.2. Dodawanie jednego lub kilku modułów do biblioteki już istniejącej

Tworzenie nowej biblioteki opisane w poprzednim podrozdziale wymagało skompilowania zawartości wszystkich plików źródłowych zawartych w danym katalogu. Alternatywnym rozwiązaniem jest dodawanie jednego lub kilku modułów do biblioteki już istniejącej w ten sposób, że dopisywane są tylko te moduły, których postać .OBJ została wcześniej utworzona przez użytkownika.

W każdym podkatalogu zawierającym pliki źródłowe programu sterującego robotów URp znajduje się program przetwarzania wsadowego LIB.BAT, którego zadaniem jest wywołanie z odpowiednim parametrem innego programu przetwarzania wsadowego, a mianowicie _LIB.BAT, zapisanego w podkatalogu c:\INTEL\UTIL (katalog c:\INTEL zawiera całe oprogramowanie narzędziowe firmy INTEL). Parametrem wywołania programu _LIB.BAT jest nazwa biblioteki która ma być zmodyfikowana, a jego działanie polega na wykonaniu m.in. następujących czynności:

1. Utworzenie pliku _1.CON zawierającego spis wszystkich zbiorów o rozszerzeniu .OBJ zawartych w danym podkatalogu (a więc spis wszystkich zbiorów będących wynikiem wykonanej kompilacji). Plik _1.CON jest tworzony komendą DIR systemu operacyjnego MS DOS, ale uruchamianą z opcjami /B/O:ne dostępnymi od wersji 5.00 tego systemu.

2. Automatyczne uruchomienie edytora ekranowego KEDIT.EXE z parametrem _1.CON. Przedtem do podkatalogu w którym ma być zbudowana biblioteka zostaje skopiowany zbiór c:\INTEL\UTIL_profi.ked. Po zmianie nazwy tego pliku na PROFILE.KED staje się on plikiem konfiguracyjnym dla edytora KEDIT, zawierającym makroinstrukcje umożliwiające zmodyfikowanie pliku _1.CON w taki sposób, że do istniejących nazw zbiorów o rozszerzeniach .OBJ zostają dopisane komendy dla programu bibliotekarza LIB86.EXE.

3. Uruchomienie programu bibliotekarza LIB86.EXE w trybie wykonywania komend zawartych w pliku _1.CON:

```
LIB86.EXE < _1.CON
```

Modyfikuje on żądaną bibliotekę.

Opisane powyżej algorytm wymaga od użytkownika uruchomienia programu LIB.BAT i zainicjowania wykonania odpowiednich makroinstrukcji edytora ekranowego KEDIT. Modyfikacja biblioteki polega na usunięciu z niej starego modułu określonego nazwą pliku o rozszerzeniu .OBJ i dodaniu nowego modułu zawartego w tym pliku. Usuwanie modułu z biblioteki polega na podaniu nazwy tego modułu, a nie nazwy zbioru, zawierającego ten moduł, ale plik _1.CON sterujący tą wymianą jest tworzony na podstawie nazw zbiorów. Stąd aby plik _1.CON zawierał poprawny ciąg komend dla bibliotekarza LIB86.EXE wymagane jest, aby nazwa modułu była taka sama jak nazwa zbioru zawierającego ten moduł. W przypadku plików zawierających definicje wykonane w języku C warunek ten jest zawsze spełniony, natomiast w przypadku zbiorów asemblerowych należy użyć odpowiedniej dyrektywy NAME (wyjaśniono to bliżej w podrozdziale pt. "Zasady pisania procedur programu sterującego robotów URp w języku asemblera ASM-86")

Poniżej przedstawiono przykład pliku _1.CON utworzonego dla zmodyfikowania biblioteki w katalogu CTRLPGM\ w przypadku zmian wprowadzonych w plikach RESTART.A86 i SYSTEM.C86. Modyfikowaną biblioteką jest main.lib, ale jej nazwa przed wykonaniem tej modyfikacji zostaje zamieniona przez program _LIB.BAT na LIBRARY.LIB:

```
delete library.lib (RESTART)
delete library.lib (SYSTEM)
add RESTART.OBJ to library.lib
add SYSTEM.OBJ to library.lib
exit
```

3.5. Program LINK86 - konsolidacja programu użytkowego

Program konsolidatora LINK86.EXE służy do łączenia ze sobą plików o rozszerzeniu .OBJ i .LIB w pojedynczy zbiór wyjściowy o rozszerzeniu .LNK, w którym jednocześnie uzupełniane są wzajemne odwołania pomiędzy obiektami zdefiniowanymi w różnych modułach. Oprócz tego program konsolidatora LINK86.EXE tworzy dodatkowy plik, o rozszerzeniu .MP1, zawierający raport o rezultacie wykonania swojej pracy.

Program LINK86 może być uruchamiany jak zwykły rozkaz systemu operacyjnego MS DOS (nie zgłasza on swojej gotowości tak jak LIB86.EXE) za pomocą komendy o następującym formacie:

```
LINK86 zbiór_1, zbiór_2, .. , zbiór_n TO zbiór_WY [parametry]
```

gdzie:

zbiór_i - określa i-ty zbiór wejściowy (plik typu .OBJ lub biblioteka o rozszerzeniu .LIB,

zbiór_WY - określa nazwę, którą będą nosić oba zbiory wyjściowe (o rozszerzeniu .LNK i .MP1),

parametry - określa opcje z jakimi można uruchamiać program LINK86.EXE. W przypadku programu sterującego robotów URp nie są wykorzystywane żadne opcje modyfikujące sposób działania konsolidatora LINK86 (lub są przyjmowane ich wartości domyślne), a więc nie będą one tutaj omawiane.

Dany program może składać się z kilkadziesięciu modułów, których każdorazowe wypisywanie w linii wywołania programu LINK86 jest kłopotliwe, bądź niewykonalne ze względu na ograniczoną długość linii wywołania. Aby tego uniknąć program konsolidatora można uruchamiać z pojedynczym parametrem, określającym nazwę pliku zawierającego specyfikacje wszystkich łączonych zbiorów. Plik taki (o domyślnym rozszerzeniu .CON) może mieć następującą postać:

```
    zbiór_1,  &
    zbiór_2,  &
    zbiór_3,  &
    .....  ,  &
    zbiór_n   &
TO zbiór_WY [parametry]
```

Należy zwrócić uwagę na znaczenie znaku "&". Jest on interpretrowany przez program LINK86.EXE jako znak wskazujący, że dalsza część linii wywołania tego programu będzie kontynuowana od nowego wiersza. Ponadto znak ten musi wystąpić w linii wywołania konsolidatora LINK86.EXE z nazwą zbioru jako parametrem wywołania:

```
LINK86.EXE & < nazwa_zbioru.CON
```

(nie jest to więc "klasyczna" funkcja systemu operacyjnego MS DOS określana jako tzw. wtórna definicja standardowego urządzenia wejściowego. Komenda:

```
LINK86.EXE < nazwa_zbioru.CON
```

jest poprawna jako funkcja systemu MS DOS, ale program LINK86.EXE ze względu na brak znaku kontynuacji "&" zinterpretuje ją jako błędną).

Program LINK86.EXE do zbioru wynikowego .LNK dołącza:

- wszystkie funkcje, zmienne i stałe (w skrócie: obiekty) zdefinowane w plikach źródłowych o rozszerzeniu .OBJ,
- te obiekty, które zdefiniowano w modułach zawartych w zbiorach bibliotecznych .LIB jawnie wyszczególnionych za nazwą biblioteki (patrz przykład),
- te obiekty, do których odwołania wystąpiły w już dołączonych obiektach.

Rozważmy następujący przykład:

```
    restart.obj,                &
    main.lib(version),          &
    ..\PROGPNL\EDIT\edit.lib,   &
    ..\PROGPNL\START\start.lib  &
TO RAMFILE.LNK
```

Program konsolidatora LINK86.EXE do pliku wynikowego, nazwanego RAMFILE.LNK będzie dołączał następujące obiekty (funkcje, zmienne, stałe):

- wszystkie obiekty zdefiniowane w pliku RESTART.OBJ, bez względu na to, czy gdziekolwiek w programie występują odwołania do nich,

39

- obiekty stałe zdefiniowane w module VERSION zawartym w bibliotece MAIN.LIB, bez względu na to, czy gdziekolwiek w programie występują odwołania do nich,

- te obiekty z biblioteki EDIT.LIB do których odwołania wystąpiły w już dołączonych obiektach, zdefiniowanych w RESTART.OBJ i VERSION z MAIN.LIB,

- te obiekty z biblioteki START.LIB do których odwołania wystąpiły w już dołączonych obiektach, zdefiniowanych w RESTART.OBJ, VERSION z MAIN.LIB, oraz w EDIT.LIB.

Jeśli do zbioru wyjściowego z biblioteki START.LIB dołączono obiekt, który zawiera odwołanie do obiektu zdefiniowanego w EDIT.LIB, ale nie dołączonego w trakcie dotychczasowej konsolidacji, to program LINK86.EXE wyświetli komunikat o błędzie. Błąd ten można usunąć dołączając powtórnie bibliotekę EDIT.LIB za START.LIB:

```
    restart.obj,                &
    main.lib(version),          &
    ..\PROGPNL\EDIT\edit.lib,   &
    ..\PROGPNL\START\start.lib, &
    ..\PROGPNL\EDIT\edit.lib    &
TO RAMFILE.LNK
```

Na powyższym przykładzie widać zalety i wady wykorzystywania bibliotek podczas konsolidacji. Wadą jest bardziej skomplikowana i mniej przejrzysta struktura pliku sterującego kolejnością wykonywania konsolidacji, zaletą natomiast to, że skonsolidowany program zawiera tylko te funkcje, zmienne i stałe, które są rzeczywiście wykorzystywane (skonsolidowany program może być krótszy).

Kolejność konsolidacji (ang. linking) programu sterującego robotów URp jest ustalana za pomocą pliku RAMLINK.CON zapisanego w podkatalogu CTRLPRM\TO_RAM. Plik ten jest identyczny dla obu przypadków lokowania programu: do pamięci RAM i do pamięci EPROM (we wcześniejszych wersjach programu sterującego robotów URp, a także IRp tworzonych w środowisku systemu GENESIS występowały dwa różne pliki wymuszające kolejność wykonywania konsolidacji całego programu sterującego w z).

3.6. Program LOC86 - lokowanie programu użytkowego

Lokator LOC86.EXE służy tworzenia obrazu danego programu w postaci, w której ma on zostać umieszczony w pamięci sterownika. Zbiorem wejściowym lokatora jest plik o rozszerzeniu .LNK będący rezultatem działania konsolidatora LINK86.EXE. Lokator LOC86.EXE tworzy dwa zbiory wyjściowe:

- plik o rozszerzeniu .LOC, zawierający binarny obraz przekształcanego programu,

- plik o rozszerzeniu .MP2, zawierający rezultat lokowania wraz z mapą pamięci obejmującej adresy wszystkich procedur, zmiennych i stałych globalnych zdefiniowanych w języku C, oraz wszystkich obiektów typu PUBLIC, zdefiniowanych w języku asemblera ASM-86.

Program LOC86 może być uruchamiany jak zwykły rozkaz systemu operacyjnego MS DOS (nie zgłasza on swojej gotowości tak jak LIB86.EXE) za pomocą komendy o następującym formacie:

LOC86 zbiór_WE TO zbiór_WY [parametry]

40

gdzie:

zbiór_WE - określa zbiór wejściowy (o domyślnym rozszerzeniu .LNK), będący rezultatem działania programu konsolidatora LINK86.EXE,

zbiór_WY - określa nazwę, którą będą nosić oba zbiory wyjściowe (o rozszerzeniu .LOC i .MP2),

parametry - określa opcje z jakimi można uruchamiać program LOC86.EXE.

W przypadku programu sterującego robotów URp wykorzystywane są następujące opcje programu lokatora LOC86.EXE:

Opcja ADDRESSES:

Opcja ta określać miejsce lokowania (adres) w pamięci poszczególnych obiektów (segmentów, grup segmentów o określonej klasie lub segmentów połączonych w grupę o określonej nazwie). Opcja ADDRESSES może mieć następującą postać:

ADDRESSES (SEGMENTS (segment_1 (adres_1),..,(segment_n (adres_n)))

ADDRESSES (CLASSES (klasa_1 (adres_1),..,(klasa_n (adres_n)))

ADDRESSES (GROUPS (grupa_1 (adres_1),..,(grupa_n (adres_n)))

gdzie słowa kluczowe SEGMENTS, CLASSES i GROUPS określają kryterium względem którego będzie wykonywane lokowanie do pamięci od danego adresu. Parametry "segment_i", "klasa_i" oraz "grupa_i" są nazwami własnymi lokowanego obiektu, parametr "adres_i" - określa fizyczny adres pamięci podawany w postaci absolutnego adresu hexadecymalnego.

Opcja ORDER:

Opcja ta pozwala ustalać kolejność umieszczania w pamięci określonych obiektów (segmentów lub grup segmentów o określonej klasie). Opcja ORDER może mieć następującą postać:

ORDER (SEGMENTS (segment_1,..,segment_n))

ORDER (CLASSES (klasa_1,..,klasa_n))

gdzie słowa kluczowe SEGMENTS i CLASSES określają kryterium porządkowania obiektów umieszczanych w pamięci. Parametry "segment_j" i "klasa_i" są nazwami własnymi lokowanego obiektu.

Opcja SEGSIZE:

Opcja ta pozwala określać długość obszaru pamięci przeznaczonego na dany segment. Opcja SEGSIZE może mieć następującą postać:

SEGSIZE (nazwa_segmentu (długość))

gdzie parametr "nazwa_segmentu" określa segment zdefiniowany w pliku wejsciowym, a "długość" - wielkość pamięci, jaką lokator ma zarezerwować na ten segment (nie więcej niż FFFFH).

Opcja SYMBOLCOLUMNS:

Opcja ta określa format tablicy zawierającej informacje o adresach wszystkich obiektów (funkcji, zmiennych, stałych) zdefiniowanych jako obiekty globalne

(język C) lub typu PUBLIC (asembler). Tablica ta stanowi część pliku wynikowego o rozszerzeniu .MP2. Opcja SYMBOLCOLUMS może mieć następującą postać:

```
SYMBOLCOLUMNS (1 ¦ 2 ¦ 3 ¦ 4)
```

gdzie "1", "2", "3", "4" - określa odpowiednio tryb jedno-, dwu-, trzy- lub czterokolumnowy.

Przedstawiony opis niektórych opcji wywołania programu lokatora LOC86.EXE nie zawiera wszystkich dostępnych o nich informacji, lecz tylko te, które są niezbędne dla wykonania prawidłowego lokowania programu sterującego robotów URp. Ponadto dodatkowo może być wykorzystywana opcja BOOTSTRAP, której działanie polega na umieszczeniu od adresu FFFF:0000H (tj. FFFF0H) kodu instrukcji tzw. skoku międzysegmentowego (ang. long jump) do procedury zadeklarowanej jako pierwsza wykonywana procedura (patrz także podrozdział pt. "Tworzenie obrazu programu sterującego robotów URp dla pamięci EPROM").

Podobnie jak w przypadku konsolidatora LINK86, każdorazowe uruchomienie programu LOC86.EXE może być kłopotliwe ze względu na mnogość ewentualnych opcji i ograniczoną długość samej linii wywołania. Dla uniknięcia tego lokator LOC86.EXE można uruchamiać tylko z pojedynczym parametrem, określającym nazwę pliku zawierającego specyfikacje wszystkich opcji wywołania. Plik taki (o domyślnym rozszerzeniu .CON) może mieć następującą (przykładową) postać:

```
LOC86 zbiór_WE TO zbiór_WY                    &
ORDER (SEGMENTS (segment_1 (adres_1),         &
.....                                         &
                (segment_n (adres_n)))        &
SYMBOLCOLUMNS (x)                             &
SEGSIZE (segment_i (xxxxH))                   &
ADDRESSES (SEGMENTS (segment_1 (adres_1),     &
.....                                         &
                (segment_n (adres_n)))
```

przy czym znaczenie znaku "&" jest identyczne jak w przypadku konsolidatora LINK86.EXE tj. jest on interpretrowany przez program LOC86.EXE jako znak wskazujący, że dalsza część linii wywołania tego programu będzie kontynuowana od nowego wiersza. Ponadto znak ten musi wystąpić w linii wywołania lokatora LOC86.EXE z nazwą zbioru jako parametrem wywołania:

```
LOC86.EXE & < nazwa_zbioru.CON
```

O właściwym ulokowaniu danego programu w pamięci sterownika decydują opcje ADDRESSES i ORDER, które mogą wzajemnie uzupełniać się. Rozważmy następujący przykład:

```
RAMFILE.LNK to RAMFILE.LOC                              &
      ORDER (SEGMENTS (PROGRAM_STACK,                   &
                       PROGRAM_DATA,                    &
                       PROGRAM_CODE))                   &
       SYMBOLCOLUMNS(1)                                 &
       SEGSIZE (stack (2B00H))                          &
       ADDRESSES (SEGMENTS (PROGRAM_STACK (500H),       &
                            PROGRAM_CODE (6000H)))
```

Program lokatora LOC86.EXE ulokuje poszczególne segmenty w kolejności (od najniższego adresu) wynikającej z parametrów opcji ORDER tj. PROGRAM_STACK, PROGRAM_DATA i PROGRAM_CODE, przy czym segmenty PROGRAM_STACK i PROGRAM_CODE będą umieszczone od adresów określonych na podstawie opcji ADDRESSES. Segment PROGRAM_DATA będzie umieszczony przez lokator bezpośrednio za segmentem

PROGRAM_STACK.

Adresy dla lokowania programu sterującego robotów URp w pamięci sterownika są ustalane za pomocą dwóch plików:

- pliku RAMLOC.CON zapisanego w podkatalogu CTRLPRM\TO_RAM. Zawiera on adresy dla programu sterującego w wersji przeznaczonej do ładowania do pamięci RAM i uruchamianej przy pomocy programu MONITOR OPERATORSKI:

```
RAMFILE.LNK to RAMFILE.LOC                    &
        ORDER (CLASSES (STACK,                &
                        DATA,                 &
                        CODE))                &
         SYMBOLCOLUMNS(1)                     &
         SEGSIZE (stack (2B00H))              &
         ADDRESSES (CLASSES (STACK(500H), &
                             DATA(3000H), &
                             CODE(6000H)))
```

- pliku ROMLOC.CON zapisanego w podkatalogu CTRLPRM\EPROM. Zawiera on adresy dla programu sterującego w wersji przeznaczonej do zapisania w pamięci stałej typu EPROM i uruchamianej jako samodzielny program sterownika:

```
RAMFILE.LNK to ROMFILE.LOC                    &
        ORDER (CLASSES (STACK,                &
                        DATA,                 &
                        CODE))                &
         SYMBOLCOLUMNS(1)                     &
         SEGSIZE (stack (2B00H))              &
         ADDRESSES (CLASSES (STACK(500H), &
                             DATA(3000H), &
                             CODE(E0000H)))
```

Pliki te różnią się jedynie adresem od którego będzie umieszczany kod programu. W obu przypadkach kryterium ustalającym porządek poszczególnych segmentów, a następnie miejsce ich ulokowania w pamięci sterownika jest klasa segmentów.

U W A G A !

W przypadku tworzenia wersji programu sterującego robotów URp przeznaczonej do umieszczenia w pamięci RAM należy pamiętać o zmniejszeniu obszaru pamięci przeznaczonego na programy użytkowe robota (zmniejszeniu deklarowanego rozmiaru tablicy glob_space[], patrz plik GLOBSIZE.A86 umieszczony w katalogu CTRLPGM\GLOBSPAC). Przyjęcie wymiaru tablicy glob_space[] takiego jak dla wersji przeznaczonej dla pamięci EPROM powoduje zachodzenie obszaru danych programu na obszar kodu, następnie przesunięcie początku obszaru kodu poza deklarowany adres 6000H, a w konsekwencji umieszczenie go w obszarze, w którym fizycznie brak jest pamięci na pakiecie sterownika MV-52.

3.7. Program OH86 - tworzenie obrazu programu użytkowego w postaci INTEL-HEX

Aby uzyskać obraz danego programu w postaci INTEL-HEX możliwy do załadowania do pamięci RAM sterownika lub przeznaczonego do wpisania do pamięci EPROM należy wykorzystać program OH86.EXE. Przekształca on plik o rozszerzeniu .LOC będący rezultatem działania lokatora LOC86.EXE, w pojedynczy zbiór wyjściowy .HEX. Program OH86.EXE jest uruchamiany jak zwykły rozkaz systemu operacyjnego MS DOS (nie zgłasza on swojej gotowości tak jak LIB86.EXE) za pomocą komendy o

następującym formacie:

OH86 zbiór_WE TO zbiór_WY

gdzie:

zbiór_WE - określa zbiór wejściowy (o domyślnym rozszerzeniu .LOC), będący rezultatem działania programu lokatora LOC86.EXE,

zbiór_WY - określa nazwę, którą będzie nosić zbiór wyjściowy (o rozszerzeniu .HEX).

Działanie programu OH86.EXE nie jest modyfikowane żadnymi dodatkowymi opcjami. Program ten nie tworzy dodatkowego pliku zawierającego raport o rezultatach swojej pracy.

3.8. Tworzenie obrazu programu sterującego robotów URp dla pamięci EPROM

Program sterujący robotów URp w wersji przeznaczonej do umieszczenia w pamięci stałej typu EPROM przed zapisaniem w niej musi być połączony z procedurą inicjującą niektóre rejestry mikroprocesora 80186 i programem MONITOR OPERATORSKI. Połączenie takie jest możliwe do wykonania już na etapie konsolidacji programu, ale wiąże się z koniecznością osobnego przygotowywania wersji przeznaczonej do pamięci RAM i EPROM jeszcze na etapie budowania bibliotek.

Procedura inicjująca niektóre rejestry mikroprocesora 80186, wspólna zarówno dla programu sterującego robotów URp i MONITORA OPERATORSKIEGO musi być umieszczona na końcu obszaru adresowego mikroprocesora 80186. Zawiera ona instrukcje odczytujące stan dodatkowego przełącznika umieszczonego wewnątrz szafy sterowniczej robota i w zależności od jego położenia przekazuje sterowanie do programu sterującego robota lub do programu MONITOR OPERATORSKI. Procedurę tę zdefinowano w programie MONITOR i dlatego jakiekolwiek zmiany adresu startowego programu sterującego są związane z modyfikacją odpowiedniej instrukcji skoku wewnątrz tej procedury.

Aby przygotować wspólny obraz programu sterującego robotów URp i programu MONITOR OPERATORSKI w postaci pojedynczego pliku INTEL-HEX należy:

- zmodyfikować obraz programu sterującego robotów URp, będący rezultatem działania programu OH86.EXE, a przeznaczonego do zapisania w pamięci EPROM. Modyfikacja ta polega na usunięciu dowolnym edytorem tekstowym dwóch rekordów, a mianowicie START ADDRESS RECORD i END_OF_FILE RECORD (zwykle rekordy te zajmują dwa ostatnie wiersze w pliku utworzonym przez program OH86.EXE).

- tak spreparowany zbiór połączyć z obrazem programu MONITOR OPERATORSKI zakodowanym w postaci INTEL-HEX. W tym celu można wykorzystać rozkaz COPY systemu operacyjnego MS DOS, pamiętając jednak, że program MONITOR musi być kopiowany jako drugi (przez programem sterującym).

Obraz programu MONITOR OPERATORSKI w formacie INTEL-HEX jest zapisany w podkatalogu CTRLPGM\EPROM\MONITOR w dwóch plikach:

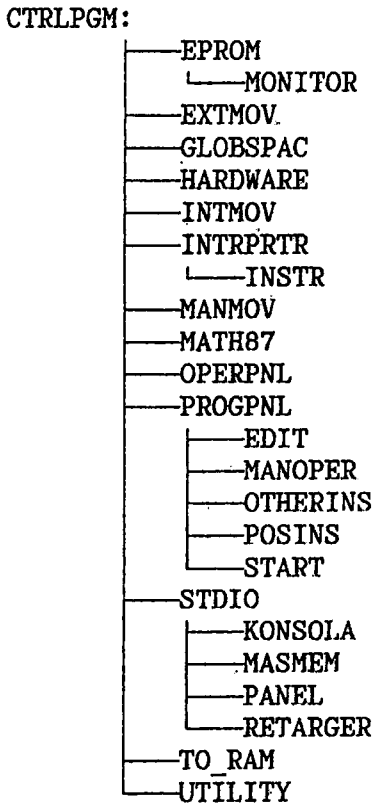
- plik MONITON.HEX zawierający wersję monitora z odblokowanym programem deasemblera instrukcji mikroprocesora 80186 i koprocesora 8087,

- plik MONITOFF.HEX zawierający wersję monitora z zadblokowanym programem deasemblera instrukcji mikroprocesora 80186 i koprocesora 8087.

Uzyskany zbiór można przekształcać programami będącymi na wyposażeniu danego programatora pamięci EPROM.

4. ORGANIZACJA POSTACI ŹRÓDŁOWEJ OPROGRAMOWANIA

Cały program sterujący zawiera się w kilkuset plikach. Jak już wyżej wspomniano jest on napisany w dwóch językach: C i asemblerze procesora 80186. Na postać źródłową oprogramowania składają się więc zbiory asemblerowe, zbiory z treścią programu w języku C i zbiory nagłówkowe. Jedyną możliwością upoprządkowania takiej mnogości plików jest zorganizowanie ich w drzewo katalogów. Przy tej okazji dokonano podziału i selekcji plików tak, aby uzyskać modułową budowę programu już na tym etapie. Chodziło o to, aby każdy katalog był odrębny tematycznie, odpowiadał za określony zakres funkcji programu sterującego. Katalog główny oprogramowania nazwano CTRLPGM. Poniżej przedstawiono jego budowę. Zawartość poszczególnych podkatalogów opisano w dodatku C.



Rys.4.1 Drzewo katalogów programu sterującego robotów URP

46

5. OGÓLNA STRUKTURA PROGRAMU STERUJĄCEGO ROBOTÓW URp

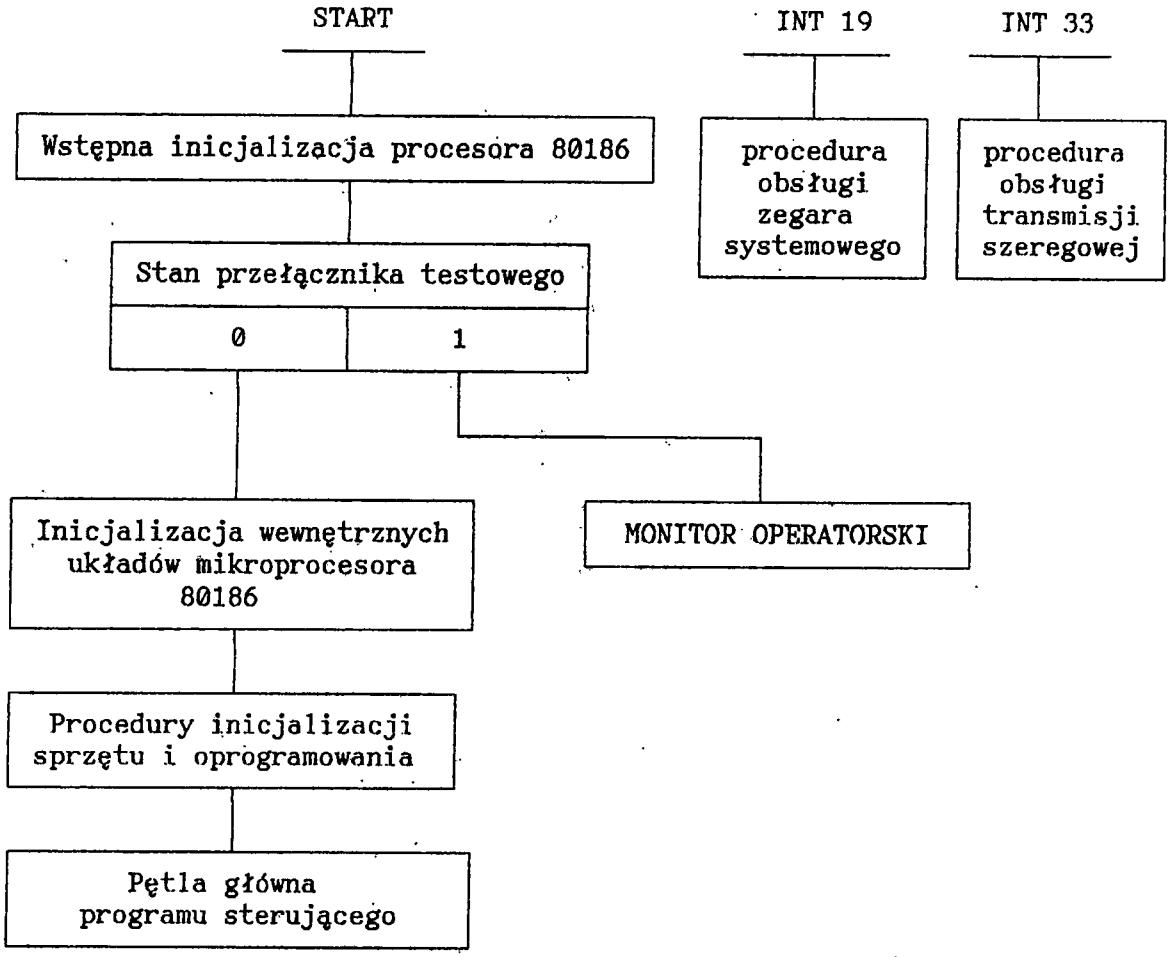
Schemat blokowy ogólnej struktury głównego programu sterującego robotów URP został przedstawiony na rys. 5.1. W procedurze startowej ustawiane są niektóre rejestry procesora, a następnie sprawdzany jest stan przełącznika testowego. Przełącznik ten umieszczony jest wewnątrz szafy sterowniczej, w tzw. zespole bezpieczników i styczników. Dołączony on jest do jednego z wejść systemowego pakietu wejść/wyjść dwustanowych (bit D7, wejście CSA). Jeśli przełącznik testowy jest załączony, to sterowanie jest przekazywane do monitora operatorskiego. Opcja ta ma znaczenie serwisowe i wykorzystywana jest przy naprawach robota oraz uruchamianiu oprogramowania. Monitor komunikuje się z operatorem poprzez kanał A transmisji szeregowej na jednostce centralnej. Można do niego podłączyć dowolny terminal pracujący z łączem V-24. Monitor operatorski umożliwia realizację szeregu funkcji bezpośrednio na sprzęcie układu sterowania. Są to m.in.:

- odczyt i zmiana zawartości rejestrów procesora,
- odczyt obszaru pamięci,
- odczyt i zmiana zawartości pojedynczej komórki pamięci,
- odczyt stanu wejść i ustawianie stanu wyjść dwustanowych zarówno użytkowych jak i systemowych,
- odczyt rejestrów wyjściowych oraz ustawianie rejestrów wejściowych sterowników położenia osi,
- ładowanie programu przez kanał V-24 z komputera zewnętrznego do pamięci RAM jednostki centralnej układu sterowania,
- uruchamianie załadowanego programu od wybranego adresu,
- ustawianie pułapki w uruchamianym programie.

Szerzej o programie monitora operatorskiego traktują pozycje przedstawione w Dodatku A.

Jeśli przełącznik testowy jest wyłączony, sterowanie jest przekazywane do zasadniczej części programu realizującej obsługę robota. Na wstępie są w niej wykonywane procedury pełnej inicjalizacji sprzętu i oprogramowania.

47



Rys.5.1. Ogólna struktura głównego programu sterującego robotów URP.

6. INICJALIZACJA UKŁADU STEROWANIA ROBOTA

Po starcie systemu sterowanie zostaje przekazane do procedury restart(), zdefiniowanej w pliku CTRLPGM\restart.a86. Są w niej po kolei wywoływane procedury inicjujące poszczególne układy sterownika:

SET_INTERRUPT_VECTOR	- ustawienie wektora przerwań,
INI_186	- inicjowanie wewnętrznych układów mikroprocesora 80186,
INIT87	- inicjowanie koprocesora 8087,
INI_Z853	- inicjowanie układu transmisji szeregowej Z8530,
INI_8259A	- inicjowanie sterowników przerwań 8259A,
STREAMS	- inicjowanie strumieni formatowanych WE/WY,
IOINIT	- inicjowanie buforów I/O,

Po wykonaniu tej inicjalizacji sterowanie zostaje przekazane do głównej procedury programu sterującego - do podprogramu MAIN.C86 (w katal. CTRLPGM). Z uwagi naszczególną wagę inicjacji samego procesora, procedura ta zostanie teraz dokładnie omówiona.

6.1. Inicjacja mikroprocesora 80186

Inicjowanie mikroprocesora 80186 na pakiecie sterownika MV-52 (procedura INI_186 zdefiniowana w pliku INI_186.A86 w podkatalogu CTRLPGM\HARDWARE) polega na wykonaniu następujących czynności:

1. Profilaktyczne wykonanie komendy non-specific End-Of-Interrupt do wewnętrznego sterownika przerwan mikroprocesora 80186. Wysłanie tej komendy polega na wpisaniu wartości 8000H do rejestru EOI.

2. Zaprogramowanie rejestrow kontrolnych sterujących pracą mikroprocesora. Poszczególne rejestry kontrolne mikroprocesora 80186 mają następujący format (w opisie podano tylko format młodszych bajtów, starsze w całości zawierają "0"):

   - format rejestrów kontrolnych wewnętrznych układów bezpośredniego dostępu do pamięci (DMA) i układu timera:

Timer / DMA	0	0	0	0	MSK	PR2	PR1	PR0

   - format rejestrów kontrolnych wewnętrznego sterownika przerwań związanych z pinami INT0 i INT1:

INT0 / INT1	0	SFNM	C	LTM	MSK	PR2	PR1	PR0

   - format rejestrów kontrolnych wewnętrznego sterownika przerwań związanych z pinami INT2 i INT3:

INT2 / INT3	0	0	0	LTM	MSK	PR2	PR1	PR0

   gdzie:

SFNM - bajt określający, czy wewnętrzny sterownik przerwań pracuje:

1 - w trybie Special Fully Nested Mode,
0 - w trybie Fully Nested Mode.

C - bajt określający, czy wewnętrzny sterownik przerwań pracuje:

1 - w trybie kaskadowym,
0 - w trybie bezpośrednim.

LTM - bajt określający, czy układ współpracujący z wewnętrznym sterownikiem przerwań zgłasza przerwanie:

1 - wysokim poziomem sygnału,
0 - zmianą sygnału z niskiego na wysoki.

MSK - bajt maski:

1 - zamaskowana obsługa przerwania,
0 - niezamaskowana obsługa przerwania.

PRx - bity określające priorytet.

W programie sterującym robotów URp do poszczególnych rejestrów kontrolnych mikroprocesora 80186 zostają wpisane następujące wartości:

- do rejestru kontrolnego wewnętrznego sterownika przerwań związanego z pinem INT3 - wartość 00001111B
- do rejestru kontrolnego wewnętrznego sterownika przerwań związanego z pinem INT2 - wartość 00000111B,
- do rejestru kontrolnego wewnętrznego sterownika przerwań związanego z pinem INT1 - wartość 00101111B
- do rejestru kontrolnego wewnętrznego sterownika przerwań związanego z pinem INT0 - wartość 00100101B,
- do rejestrów kontrolnych obu wewnętrznych układów DMA bezpośredniego dostępu do pamięci (DMA) - wartości 00001111B,
- do rejestru kontrolnego wewnętrznego układu timera - wartość 00000111B.

2. Ustawienie rejestru maski.
Rejestr maski dla wewnętrznego sterownika przerwań mikroprocesora 80186 ma następujący format (w opisie podano tylko format młodszego bajtu, starszy w całości zawiera "0"):

MASK_REGISTER	I3	I2	I1	I0	DMA_1	DMA_0	0	TIMER

gdzie:

I3 - odpowiada 3 wejściu od przerwania spoza wewnętrznych układów mikroprocesora,
I2 - odpowiada 2 wejściu od przerwania spoza wewnętrznych układów mikroprocesora,
I1 - odpowiada 1 wejściu od przerwania spoza wewnętrznych układów mikroprocesora,

I0 - odpowiada 0 wejściu od przerwania spoza wewnętrznych układów mikroprocesora,
DMA_1 - odpowiada przerwaniu od 1 kanału wewnętrznego modułu DMA,
DMA_0 - odpowiada przerwaniu od 0 kanalu wewnętrznego modułu DMA,
TIMER - odpowiada przerwaniu od jednego z wewnętrznych timerów.

"1" na pozycji danego bitu oznacza, że przerwania z odpowiadającego mu źródła są zamaskowane.

W programie sterującym robotów URp zamaskowane zostają przerwania na wejściach I3, I2 i I1 spoza wewnętrznych układów mikroprocesora 80186 oraz przerwania od obu układów DMA bezpośredniego dostępu do pamięci.

3. Inicjowanie wewnętrznych timerów mikroprocesora 80186.
Inicjowanie wewnętrznych timerów mikroprocesora 80186 polega na ustawieniu rejestru kontrolnego i wpisaniu zliczanej wartości do innego rejestru (rejestrów) związanego z tym timerem. Rejestry kontrolne wewnętrznych timerów mikroprocesora 80186 mają następujący format:

	15	14	13	12	11	10	9	8
Starszy bajt:	EN	INH	INT	RIU	0	0	0	0

gdzie:

EN - "enable bit" pozwalający kontrolować, czy dany timer zlicza, czy został zatrzymany:

- dla EN=1: timer zlicza sygnały (pracując w trybie zliczania sygnałów z wewnętrzego zegara mikroprocesora, bit EXT=0), o ile pozwala mu na to stan na jego linii wejściowej - patrz bit RTG),

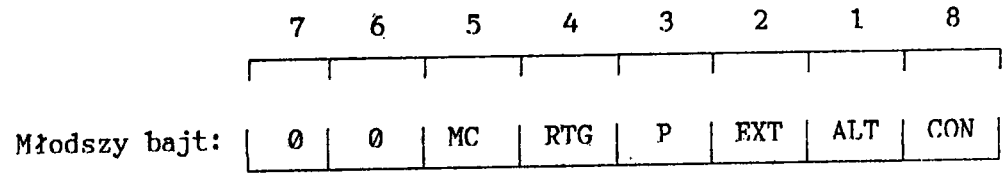
- dla EN=0: zliczanie zostaje zawieszone.

Dla CON=0 bit EN zostaje automatycznie wyzerowany po zliczeniu do zaprogramowanej wartości.

INH - bit ten umożliwia selektywne ustawianie bitu EN. Jeśli podczas programowania słowa kontrolnego danego timera bit INH=1, to wartość bitu EN zostaje ustawiona zgodnie z opisem w słowie kontrolnym, w przypadku INH=0 wartość bitu EN zostaje zignorowana.

INT - bit maskujący przerwania od danego timera. Jeśli INT=1 to timer generuje przerwanie po zliczeniu do zaprogramowanej wartości. Jeśli TIMER został ustawiony w trybie pracy polegający na kolejnym zliczaniu do wartości określonej najpierw zawartością rejestru MAX_COUNT_A, a następnie zawartością MAX_COUNT_B (przypadek dla ktorego bit ALT=1), to przerwanie pojawia się po zliczeniu do MAX_COUNT_A, a następnie po zliczeniu do MAX_COUNT_B.

RIU - bit informacyjny, określający licznik którego zawartość jest aktualnie porównywana ze zliczaną wartością: RIU=0 wskazuje na MAX_COUNT_A, RIU =1 na MAX_COUNT_B. Podczas programowania słowa kontrolnego wartość tego bitu jest ignorowana. W przypadku, gdy ustawiono wartość bitu ALT=0 bit RIU jest zawsze równy "0".



gdzie:

MC - jest to bit informacyjny dla programisty. Jeśli dany timer został ustawiony w trybie pracy polegającym na kolejnym zliczaniu do wartości określonej najpierw zawartością rejestru MAX_COUNT_A, a następnie zawartością rejestru MAX_COUNT_B (przypadek dla którego bit ALT=1), to bit MC zostaje ustawiony w stan "1" - jeśli timer wykonał pełny cykl takiego zliczania, przy czym wartość tego bitu zostaje ustawiona bez względu na stan bitu INT. Bit ten jest bitem "zatrzaskowym", tzn. po zliczeniu zachowuje wartość "1" bez względu na stan kolejnego zliczania, a więc przed jego kolejnym wykorzystaniem musi zostać wyzerowany.

RTG - bit określający funkcje sterującą pinu wejściowego danego timera, ale tylko w przypadku, gdy źródłem sygnalu zliczanego jest wewnętrzny zegar mikroprocesora (EXT =0):

- dla RTG=0: timer zlicza sygnały zegarowe, ale tylko wtedy, gdy na jego pinie wejściowym jest stan HIGH. W przypadku stanu LOW zliczanie zostaje zawieszone, ale wartość zliczona jest zachowywana.

- dla RTG=1: w przypadku zmiany sygnału na pinie wejściowym z LOW na HIGH timer zeruje rejestr zliczający i rozpoczyna zliczanie od początku. Jesli bit CON=0 to po zliczeniu do zaprogramowanej wartości zostaje wyzerowany bit EN.

P - bit "przeskalowujący". Jego stan jest ignorowany, jesli EXT=0. Jeśli EXT=1 to:

0 - sygnałem zegarowym, taktującym timer jest sygnał równy 1/4 częstotliwosci wewnętrzego zegara CPU,

1 - sygnałem zegarowym, taktujacym TIMER jest sygnał z pinu wyjściowego timera 2.

EXT - bit określający źródło zliczanych sygnałów:

1 - źródło zewnętrzne (zliczane są sygnały na pinie wejściowym do timera),

0 - źródło wewnętrzne (zliczane są sygnały z wewnętrznego zegara mikroprocesora). W tym przypadku funkcja pinu wejściowego do timera jest określona stanem bitu RTG.

ALT - bit określający, który z licznikow MAX_COUNT_A lub MAX_COUNT_B jest wykorzystywany do porównywania z rejestrem COUNT_REGISTER:

1 - wykorzystuje się najpierw MAX_COUNT_A, a po zliczeniu do wartości określonej jego zawartością, do kolejnego zliczania wykorzystuje się MAX_COUNT_B, po czym w zależności od wartości bitu CON timer albo zostaje zatrzymany, albo zlicza od początku,

0 - wykorzystuje się MAX_COUNT_A (dodatkowo na pinie wyjściowym z danego timera pojawia się sygnał LOW na 1 cykl zegara po zliczeniu do wartości maksymalnej).

52

CON - bit określający, czy dany timer pracuje w sposób rewersyjny, tj. czy zlicza do wartości określonej zawartością MAX_COUNT_A (a następnie MAX_COUNT_B, jeśli bit ALT=1) i staje, czy zlicza od poczatku:

1 - timer pracuje rewersyjnie,

0 - po zliczeniu timer zostaje zatrzymany.

Dla TIMERA 2 bity ALT, EXT, P, RTG i RIU sa zawsze ustawiane na "0".

W przypadku sterownika MV-52 wejścia timerów: 0 i 1 są zwarte do masy (jest na nich sygnal LOW), stąd nie można ich wykorzystywac do żadnego zliczania (stąd nie są one programowane). Do zliczania sygnału zegarowego będzie zatem wykorzystany timer 2. Przy założeniu, że cykl zegara mikroprocesora wynosi 0.5 mikrosekundy wpisanie do rejestru TIMER_2_MAX_COUNT_A wartości 40000 spowoduje, że przerwanie zegarowe będzie pojawiało się co 20 milisekund. Do rejestru kontrolnego tego timera należy przedtem wpisać następującą wartość:

11100000 00000001B

53

7. PĘTLA GŁÓWNA PROGRAMU STERUJĄCEGO

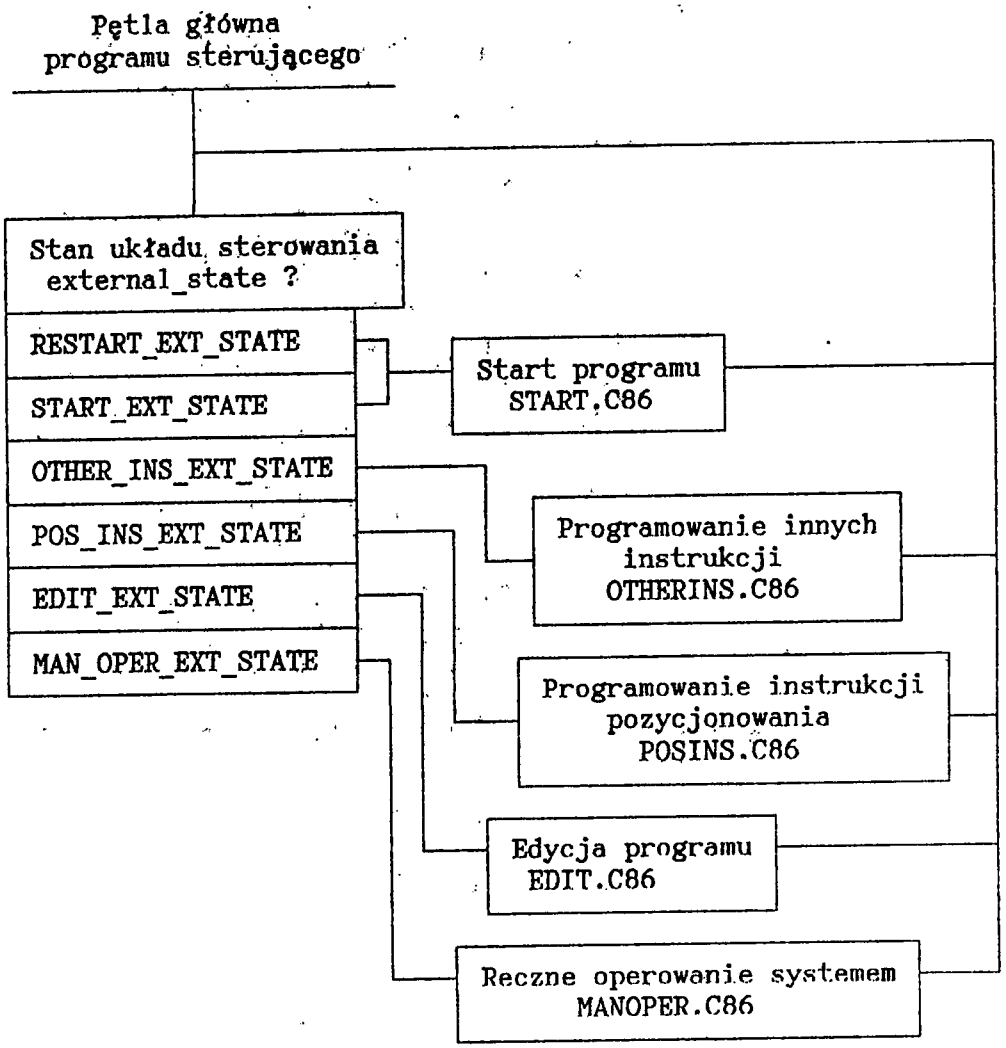
Wykonanie programu sterującvego jest uzależnione od zewnętrznego stanu układu sterowania robota, widzianego od strony operatora. Stan ten określa zmienna globalna 'external_state'. Może ona prezyjmować następujące wartości (określone w pliku EXTWORLD.H):

- RESTART_EXT_STATE = 0 - restart programu po właczeniu zasilania,
- OTHER_INS_EXT_STATE = 1 - programowanie innych instrukcji,
- POS_INS_EXT_STATE = 2 - programowanie instrukcji pozycjonowania,
- EDIT_EXT_STATE = 3 - edycja programu,
- MAN_OPER_EXT_STATE = 4 - ręczne operowanie systemem,
- START_EXT_STATE = 5 - start programu.

Po prawidłowo zakończonej inicjalizacji program ustawia stan zmiennej 'external_state' na RESTART_EXT_STATE i wchodzi do pętli głównej. Jest ona oprogramowana w procedurze MAIN.C86 w katalogu CTRLPGM. W pętli tej sterowanie jest przekazywane do jednej z pięciu procedur obsługujących poszczególne stany zewnętrzne układu sterowania robota. Stany te są zgodne ze stanami ustawianymi przez operatora na panelu programowania - opisanymi w podręczniku programowania robotów URP. Powrót do pętli głównej z procedur obsługi następuje po przełączeniu przez operatora stanu zewnętrznego. Realizowane jest to przy wykorzystaniu dostępnego w języku C mechanizmu "setjmp-longjmp". Pozwoliło to uniknąć długich serii powrotów z zagnieżdżonych podprogramów, przy jednoczesnym zapewnieniu prawidłowego bilansowania stosu.

W obsłudze każdego ze stanów zewnętrznych jest wywoływana cyklicznie procedura SYSTEM.C86, sprawdzająca stan całego układu sterowania robota. Rozpoczyna się ona od sprawdzenia czy nie odebrano z panelu rozkazu przełączenia stanu zewnętrznego. W takiej sytuacji wykonywany jest długi skok powrotny do pętli głównej. Jeśli stan nie został zmieniony, sprawdzane są rejestry stanu sterowników osi. Wszelkie sytuacje awaryjne (np. przekroczenie ograniczenia prądu lub maksymalnego dopuszczalnego błędu położenia osi) są rejestrowane i przechowywane w zmiennej globalnej 'hardware_error', określającej stan sprzętu układu sterowania. Następnie sprawdzane jest, czy operator nie użył przycisku ręcznego poruszania manipulatorem. Jeśli tak, to sterowanie jest przekazywane do odpowiednich procedur realizujących ruch w odpowiednimukładzie współrzędnych (katalog MANMOV). W dalszej części procedury SYSTEM program sprawdza, czy nastąpiła interwencja operatora za pośrednictwem panelu operacyjnego. W takich sytuacjach sterowanie jest przekazywane do procedur obsługi zleceń operatora (katalog OPERPNL). Następnie program sprawdza, czy nie należy zmienić stanu chwytaków. We wszystkich tych procedurach kontrolowane są różne blkoki układu sterowania. Wszelkie informacje o nieprawidłowościach są zapisywane w zmiennej 'hardware_error'. W kolejnej wywoływanej w podprogramie SYSTEM procedurze - HARDERR.C86 - zmienna ta jest analizowana i w zależności od jej wartości podejmowane są odpowiednie działania (sygnalizacja błędu z wyświetleniem komunikatu na panelu programowania). Na zakończenie procedury SYSTEM sprawdzany jest stan bufora przesyłek z panelu priogramowania. Może się zdarzyć, że np. na skutek pomyłki operatora (wciśnięcie złego przycisku) jednostka centralne odbierze z panelu przesyłkę bez znaczenia, nieistitną. Taka przesyłka musi być zignorowan i usunięta z bufora, aby nie blokować miejsca na kolejne przesyłki. Realizuje to podprogram UNLOCK.C86. Jego wywołanie kończy działanie procedury SYSTEM.

Wprowadzenie opisanego mechanizmu z funkcją SYSTEM zapewnia użytkownikowi możliwość ingerencji w przebieg wykonywanego zlecenia. Realizacja procedur obsługi tych zleceń może być bardzo długa. Przykładem niech tu będzie rozkaz wykonywania programu użytkowego robota. Dlatego wprowadzenie możliwości np. zatrzymanie wykonywania instrukcji czekania było niezbędne.



Rys. 7.1 Schemat blokowy pętli głównej programu sterującego.

8. Dodatek A. DOKUMENTY ZWIĄZANE

[1] P.Jabłoński, M.Pachuta:
"DTR układu sterowania robotów URP-6 i URP-3"
Archiwum PIAP nr rej. 6843, 1992

[2] M.Słodczyk, A.Syryczyński, A.Kisiel:
"DTR pakietu pamięci masowej MV-62"
Archiwum PIAP nr rej. 6761, grudzień 1991

[3] M.Słodczyk, A.Syryczyński:
"DTR pakietu jednostki centralnej MV-52"
Archiwum PIAP nr rej. 6760, grudzień 1991

[4] SGS-Thomson Microelectronics:
"Z8530 - Serial Communications Controller"
Januar 1989

[5] Siemens Technical Description:
"AMS-M17-A8 CPU Board (SAB 80186, 8MHz) with 256 Kbyte SRAM"
L8451-B40-A45-3-7618, September 1989

[6] J.Dunaj
"Pakiet oprogramowania pamięci masowej zrealizowanej na bazie modułu MV-62"
Biuletyn MERA-PIAP nr 2-3/1991

[7] J.Dunaj
"Monitor operatorski dla sterowników przemysłowych z mikroprocesorem 16-bitowym"
Biuletyn MERA-PIAP nr 2-3/1991

[8] J.Dunaj, W.Hernik, M.Jacórzyńska-Śmigiera, Z.Pilat:
"Podręcznik programowania robotów URP"
Archiwum PIAP nr rej. 6861, wrzesień 1992

[9] W.Hernik
"Panel programowania - podręcznik użytkownika"
Archiwum PIAP nr rej. 6561, luty 1991

[10] INTEL D86-ASM86-NL version 4.3:

Tom I:
1. iAPX 86,88 Family Utilities User's Guide for DOS Systems
2. Operating System Interface Libraries Manual

Tom II:
1. ASM86 Assembly Language Reference Manual
2. ASM86 Macro Assembler Operating Instructions for DOS Systems

Tom III:
1. 8087 Support Library Reference Manual

Tom IV:
1. 80C187 Support Library Reference Manual

Podręcznik:
1. An Introduction To ASM86

[11] INTEL D86-C86-NL version 4.1:

Tom I:
1. iC-86/286 Compiler User's Guide for DOS Systems

Tom II:
1. iC-86/286 Libraries Supplement

Podręcznik:
1. C: A Reference Manual

9. Dodatek B. WEWNĘTRZNY STEROWNIK PRZERWAŃ MIKROPROCESORA 80186

Mikroprocesory Intel 80186 i 80188 posiadają wewnętrzny sterownik przerwań, umożliwiający obsługę przerwań zgłaszanych w typowym systemie mikroprocesorowym. Obsługa ta obejmuje synchronizację zgłoszeń poszczególnych przerwań, ustalanie priorytetów ich obsługi oraz wystawianie na szynę adresową odpowiedniego wektora przerwań po potwierdzeniu jego przyjęcia przez mikroprocesor. Dopuszczalne jest zagnieżdżanie programów obsługi poszczególnych przerwań, tzn. obsługa przerwania o niższym priorytecie moż być zawieszona na czas obsługi przerwania o wyższym priorytecie.

Wewnętrzny sterownik przerwań mikroprocesora 80186 (80188) może obsługiwać przerwania zgłaszane zarówno przez układy zewnętrzne, współpracujące z mikroprocesorem jak też przez wewnętrzne układy samego mikroprocesora tj. przez układy wewnętrznych timerów i sterowników DMA. W tym drugim przypadku obsługa przerwania może zostać zamaskowana albo poprzez odpowiednie ustawienie rejestru sterującego układu będącego źródłem przerwania, albo poprzez ustawienie odpowiednich bitów w rejestrze sterującym pracą samego sterownika przerwań.

Wewnętrzny sterownik przerwań mikroprocesora 80186 (80188) może pracować w dwóch podstawowych trybach pracy: trybie MASTER (lub non-iRMX_86) lub trybie iRMX_86. W trybie MASTER wewnętrzny sterownik przerwań działa jako nadrzędny sterownik przerwań całego systemu mikroprocesorowego, w trybie iRMX_86 - jako sterownik SLAVE w stosunku do zewnętrznego sterownika PIC 8259A, który pełni rolę sterownika nadrzędnego (MASTER). W zależności od wybranego trybu pracy zmienia się znaczenie poszczególnych pinów wewnętrznego sterownika przerwań i jego rejestrów sterujących, ale podstawowe funkcje sterownika pozostają takie same.

9.1. Tryb pracy MASTER (non-iRMX 86) wewnętrznego sterownika przerwań

W trybie MASTER wewnętrzny sterownik przerwań pracuje jako nadrzędny sterownik przerwań dla całego systemu mikroprocesorowego. Jego pięć pinów jest wykorzystywanych dla obsługi przerwań zewnętrznych tzn. spoza wewnętrznych układów samego mikroprocesora. Jeden z tych pinów jest przeznaczony do zgłaszania przerwania niemaskowalnego (NMI), a pozostałe cztery mogą pracować w następujący sposób:

- jako cztery linie wejściowe na których mogą być zgłaszane przerwania z czterech różnych źródeł (brak jest linii potwierdzających przyjęcie przerwania). W tym przypadku wektor przerwań jest generowany przez mikroprocesor,

- jako dwie linie wejściowe na których mogą być zgłaszane przerwania (bez linii potwierdzających ich przyjęcie) i jako para: linia na której może być zgłoszone przerwanie i linia na której pojawia się potwierdzenie jego przyjęcia. W przypadku zgłoszenia przerwania na jednej z linii bez potwierdzenia jego przyjęcia wektor przerwań jest generowany przez mikroprocesor,

- jako dwie pary: linia na której może być zgłoszone przerwanie i linia na której pojawia się potwierdzenie jego przyjęcia.

Wewnętrzny sterownik przerwań jako przerwanie może interpretować sygnał o wartości "1" lub zmianę sygnału z "0" na "1". Interpretacja zależy od ustawienia bitu LTM w odpowiednim rejestrze sterującym. W zależności od przyjętego

"podtrybu" pracy tego sterownika wektor przerwań może być generowany przez mikroprocesor lub przez sam sterownik. Mikroprocesor generuje wektor przerwań zawsze wtedy, gdy źródłem przerwania jest jeden z pozostałych wewnętrznych układów tego mikroprocesora (timer, DMA) lub gdy przerwanie jest zgłaszane na tym pinie wewnętrznego sterownika przerwań, któremu nie odpowiada żadna linia przesyłająca sygnał potwierdzenia przyjęcia przerwania (przypadek taki zachodzi w "podtrybach" pracy: Fully Nested i częściowo w Cascade).

W przypadku zgłoszenia przerwania, po którym mikroprocesor sam generuje wektor przerwań należy pamiętać, że nie ma programowej możliwości ustawiania tego wektora poprzez wykonanie wcześniejszej inicjacji mikroprocesora - każde źródło przerwania ma z góry określony numer przerwania, zgodnie z poniższą tabelą:

Źródło przerwania:	Numer przerwania:	Domyślny priorytet:
NMI	2	1
Timer nr 0	8	2
Timer nr 1	18	2
Timer nr 2	19	2
Reserved	9	3
DMA nr 0	10	4
DMA nr 1	11	5
INT 0	12	6
INT 1	13	7
INT 2	14	8
INT 3	15	9

Wszystkie trzy timery z punktu widzenia sterownika przerwań stanowią jedno źródło przerwań (to mikroprocesor generuje różne wektory). Oznacza to, że przerwania zgłaszane przez poszczególne timery mają jednakowy priorytet obsługi w odniesieniu do innych przerwań, ale ich hierarchię wewnętrzną określa numer timera (przerwanie zgłaszane przez timer nr 0 jest nadrzędne w stosunku do przerwania zgłaszanego przez timer nr 1).

Użytkownik może przyporządkować do każdego źródła przerwań odpowiedni priorytet jego obsługi. Przyporządkowanie to polega na wpisaniu do odpowiedniego rejestru sterującego 3-bitowej liczby określającej priorytet przerwania (przerwanie o priorytecie np. 4 jest obsługiwane przed przerwaniami o priorytetach od 5 do 7). W przypadku jednoczesnego zgłoszenia przerwań o jednakowych priorytetach obsługiwane jest najpierw to przerwanie, które posiada wyższy priorytet domyślny (patrz tabela powyżej).

W trybie MASTER (non-iRMX_86) wewnętrzny sterownik przerwań może pracować w trzech różnych "podtrybach" pracy: fully nested, cascade lub special fully nested.

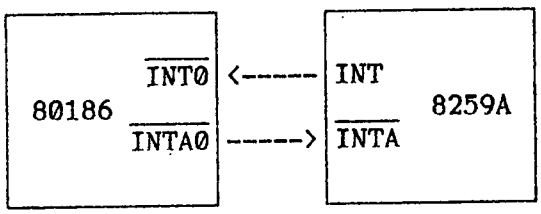
9.1.1. Podstawowy "podtryb" pracy (fully nested mode)

W podstawowym "podtrybie" pracy (fully nested mode) piny INT0, INT1, INT2 i INT3 wewnętrznego sterownika przerwań są używane jako cztery linie wejściowe na których mogą być zgłaszane przerwania z czterech różnych źródeł (brak jest linii potwierdzających przyjęcie przerwania). Wektor przerwań jest generowany przez mikroprocesor. Każdemu przerwaniu jest przyporządkowany pojedynczy bit w

rejestrze obsługi przerwań (IS). Jego ustawienie zapobiega przed podjęciem obsługi przerwania o mniejszym priorytecie podczas wykonywania programu obsługi przerwania o wyższym priorytecie. Po zakończeniu obsługi przerwania bit odpowiadający temu przerwaniu w rejestrze IS musi zostać zgaszony przez programistę odpowiednią komendą EOI (End-Of-Interrupt).

9.1.2. Kaskadowy "podtryb" pracy (cascade mode)

W kaskadowym "podtrybie" pracy (cascade mode) piny INT0, INT1, INT2 i INT3 wewnętrznego sterownika przerwań są używane jako dwie pary: linia na której może być zgłoszone przerwanie i linia na której pojawia się potwierdzenie jego przyjęcia. Wzajemne połączenie z zewnętrznym sterownikiem PIC 8259A pokazano na poniższym rysunku:



Linia INT0 wewnętrznego sterownika przerwań służy do zgłaszenia przerwanie z zewnętrznego układu PIC 8259A, natomiast linia INT2 (INTA0) do potwierdzenia przyjęcia tego przerwania. Podobną rolę pełnią linie INT1 i INT3. Każda taka para może współpracować zarówno z zewnętrznym sterownikiem przerwań PIC 8259A jak też z pojedynczym układem zgłaszającym przerwanie z potwierdzeniem. Zależy to od odpowiedniego zaprogramowania rejestrów kontrolnych dla linii INT0 i INT1.

Kaskadowy "podtryb" pracy wewnętrznego sterownika przerwań pozwala na jednoczesną obsługę do 128 zewnętrznych przerwań przy zastosowaniu dwóch zewnętrznych układów PIC 8259A pracujących jako sterowniki MASTER i szesnastu układów PIC 8259A pracujących w trybie SLAVE. Po zakończeniu obsługi każdego przerwania programista musi zapewnić wysłanie trzech komend EOI (End-Of-Interrupt) do sterowników obsługujących dane przerwanie na każdym poziomie kaskady.

9.1.3. Specjalny podstawowy "podtryb" pracy (special fully nested mode)

Specjalny postawowy "podtryb" pracy (special fully nested mode) można programować poprzez ustawienie bitów SFNM w kontrolnych rejestrach odpowiadających pinom INT0 i INT1 wewnętrznego sterownika przerwań. Podczas normalnej pracy każde przerwanie jest wtedy rozpoznawane, gdy odpowiedni bit IS w rejestrze obsługi przerwań jest zgaszony. W przypadku, w którym do zewnętrznego sterownika przerwań jest dołączonych więcej niż jedno źródło przerwań, wszystkie te przerwania są zgłaszane do mikroprocesora jedną linią. A zatem, jeśli do zewnętrznego sterownika przerwań podczas obsługi jednego przerwania zgłoszono przerwanie o wyższym priorytecie, to nie zostanie ono rozpoznane przez mikroprocesor 80186 do chwili zgaszenia bitu IS odpowiadającego linii, po której są zgłaszane przerwania z zewnętrznego układu PIC 8259A. W specjalnym podstawowym "podtrybie" pracy wewnętrzny sterownik przerwań mikroprocesora 80186 (80188) umożliwia rozpoznawanie przerwań zgłaszanych na pinach zewnętrznego sterownika przerwań bez względu na stan odpowiedniego bitu IS w rejestrze obsługi wewnętrznego sterownika. Należy zwrócić jednak uwagę, że

przy wysyłaniu komendy EOI trzeba zbadać stan wszystkich bitów IS w rejestrze kontrolnym zewnętrznego sterownika. Komendę EOI do wewnętrznego sterownika przerwań można wysłać dopiero wtedy, gdy wszystkie przerwania zgłaszane do sterownika zewnętrznego zostały obsłużone.

9.1.4. Badanie stanu wewnętrznego sterownika przerwań

Podczas pracy we wszystkich opisanych podtrybach możliwe jest badanie stanu wewnętrznego sterownika przerwań, podczas którego procesor blokuje przerwania. Badanie takie polega na odczycie specjalnego, 16-bitowego rejestru tzw. Poll Register. Najstarszy bit tego rejestru wskazuje, czy zgłoszono przerwanie o wyższym priorytecie podczas wykonywania programu obsługi przerwania o niższym priorytecie. Cztery najmłodsze bity tego rejestru określają numer wektora przerwań, który odpowiada przerwaniu o najwyższym priorytecie.

Możliwość odczytu stanu wewnętrznego sterownika przerwań jest użyteczna w przypadku, w którym potrzebna jest informacja o zgłoszonych przerwaniach, ale bez konieczności zapewnienia ich natychmiastowej obsługi. Informacja zawarta w Poll Register jest dodatkowo powielona w tzw. Poll Status Word, ale odczyt Poll Status Word nie ustawia odpowiedniego bitu IS w rejestrze obsługi przerwań.

9.1.5. Komedna End-Of-Interrupt w trybie MASTER

Komenda EOI jest używana przez programistę dla gaszenia odpowiedniego bitu IS w rejestrze obsługi przerwań, po zakończeniu obsługi danego przerwania. Zainicjowanie wykonania tej komendy polega na wpisaniu określonej wartości dwubajtowej do tzw. EOI Register. Rozróżnia się dwa rodzaje komend End-Of-Interrupt:

- nonspecyfic EOI, która gasi ten bit IS w rejestrze obsługi przerwań, który odpowiada przerwaniu o najwyższym priorytecie,
- specyfic EOI, która gasi ten bit IS w rejestrze obsługi przerwań, którego numer określono explicite.

9.2. Tryb pracy iRMX 86 wewnętrznego sterownika przerwań

Wewnętrzny sterownik przerwań mikroprocesora 80186 (80188) ma możliwość pracy w tzw. trybie iRMX_86, który jest wymagany w przypadku wykorzystywania systemu operacyjnego iRMX_86. Tryb ten jest ustawiany poprzez zapalenie bitu 14 w tzw. Peripheral Control Block Relocation Register ustawianego podczas inicjowania mikroprocesora 80186 (80188).

W trybie pracy iRMX_86 wewnętrzny sterownik przerwań mikroprocesora 80186 (80188) pracuje jako sterownik podrzędny (SLAVE) w stosunku do zewnętrznego układu PIC 8259A, pracującego jako sterownik MASTER. Przerwania od innych wewnętrznych układów mikroprocesora 80186 (80188) tzn. od układów DMA i timerów są kontrolowane przez wewnętrzny sterownik przerwań.

Połączenie wewnętrznego sterownika przerwań do zewnętrznego układu PIC 8259A wymaga wykorzystania wszystkich czterech linii tego sterownika (za wyjątkiem

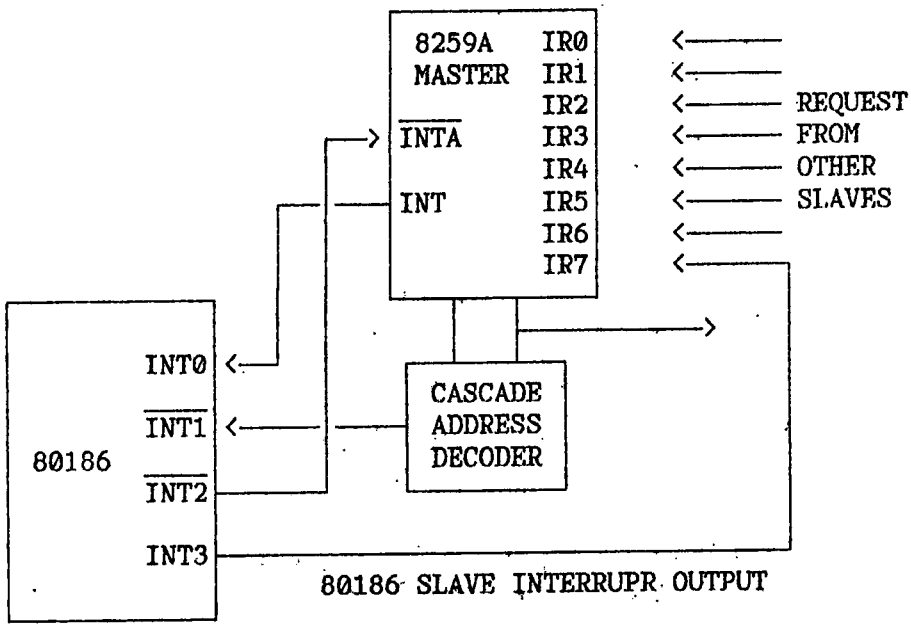
lini przeznaczonej do zgłaszania NMI), a zatem wewnętrzny sterownik przerwań nie może obsługiwać zewnętrznych źródeł przerwań. W odróżnieniu od trybu pracy MASTER (non-iRMX) wszystkie trzy wewnętrzne timery są rozróżniane jako niezależne źródła przerwań i posiadają własne bity IS w rejestrze obsługi przerwań i własne słowa sterujące.

System operacyjny iRMX_86 wymaga, aby wszystkim źródłom przerwań przyporządkować stałe priorytety ich obsługi. A zatem podczas inicjowania całego softwaru programista musi określić priorytet obsługi każdego potencjalnego źródła przerwań. W przypadku wewnętrznych układów mikroprocesora 80186 (80188) priorytety te określa poniższa tabela:

Źródło przerwania:	Priorytet:
Timer nr 0	0
Reserved	1
DMA nr 0	2
DMA nr 1	3
Timer nr 1	4
Timer nr 2	5

Try

b pracy iRMX_86 pozwala na zagnieżdżenie żądań obsługi przerwań. Po potwierdzeniu przerwania zostają zamaskowane wszystkie przerwania o niższych priorytetach obsługi.



Połączenie wewnętrznego sterownika przerwań mikroprocesora 80186 (80188) pracującego w trybie iRMX_86 z zewnętrznym układem PIC 8259A.

9.2.1. Generacja wektora przerwań w trybie iRMX 86

Generacja wektora przerwań w trybie iRMX_86 jest dokładnie taka sama jak dla sterownika PIC 8259A pracującego w kaskadzie w reżimie SLAVE: Sterownik przerwań generuje 8-bitowy numer przerwania, który CPU mnoży przez 4, a następnie używa

62

jako adresu w tablicy wektora przerwań. Pięć najbardziej znaczących bitów tego numeru jest wpisywanych przez programistę podczas inicjowania mikroprocesora 80186 (80188) do rejestru wektora przerwań (Interrupt Vector Register).

9.2.2. Komenda End-Of-Interrupt w trybie iRMX 86

W trybie pracy iRMX_86 komenda tzw. specific End-Of-Interrupt gasi określony explicite przez użytkownika bit IS w rejestrze obsługi przerwań.

9.3. Rejestry wewnętrznego sterownika przerwań

Wewnętrzny sterownik przerwań zawiera 15 rejestrów wykorzystywanych do sterowania jego pracą. Wszystkie one mogą być zapisywane i odczytywane, a funkcje niektórych z nich różnią się od siebie w zależności od przyjętego trybu pracy.

	OFFSET
INT_3 CONTROL REGISTER	3EH
INT_2 CONTROL REGISTER	3CH
INT_1 CONTROL REGISTER	3AH
INT_0 CONTROL REGISTER	38H
DMA_1 CONTROL REGISTER	36H
DMA_0 CONTROL REGISTER	34H
TIMER CONTROL REGISTER	32H
INTERRUPT CONTROLLER STATUS REGISTER	30H
INTERRUPT REQUEST REGISTER	2EH
IN-SERVICE REGISTER	2CH
PRIORITY MASK REGISTER	2AH
MASK REGISTER	28H
POLL STATUS REGISTER	26H
POLL REGISTER	24H
EOI REGISTER	22H

Tryb pracy MASTER (non-iRMX_86)

	OFFSET
LEVEL 5 CONTROL REGISTER (TIMER 2)	3AH
LEVEL 4 CONTROL REGISTER (TIMER 1)	38H
LEVEL 3 CONTROL REGISTER (DMA 1)	36H
LEVEL 2 CONTROL REGISTER (DMA 0)	34H
LEVEL 0 CONTROL REGISTER (TIMER 0)	32H
INTERRUPT REQUEST REGISTER	2EH
IN-SERVICE REGISTER	2CH
PRIORITY-LEVEL MASK REGISTER	2AH
MASK REGISTER	28H
SPECIFIC EOI REGISTER	22H
INTERRUPT VECTOR REGISTER	20H

Tryb pracy RMX86

9.3.1. Rejestry kontrolne (Control Registers)

Wewnętrzny sterownik przerwań zawiera 7 kontrolnych rejestrów, po jednym dla każdego źródła przerwań (poza NMI). W trybie pracy MASTER, cztery z nich (INT0 - INT3) obsługują przerwania od układów zewnętrznych mikroprocesora, dwa są przeznaczone do obsługi obu wewnętrznych układów DMA a jeden jest wspólny dla trzech wewnętrznych timerów. W trybie pracy iRMX_86 rejestry kontrolne INT2 i INT3 nie są używane, rejestry INT0 i INT1 obsługują przerwania od wewnętrznych timerów odpowiednio 1 i 2, natomiast rejestry DMA_0 i DMA_1 obsługują przerwania od wewnętrznych źródeł.

Każdy z wymienionych rejestrów zawiera trzy bity: PR0, PR1 i PR2, określających priorytet obsługi przerwania odpowiadającego danemu rejestrowi kontrolnemu (0 oznacza najwyższy priorytet, 7 - najniższy), oraz pojedynczy bit MSK, maskujący obsługę danego przerwania (MSK = 1 oznacza przerwanie zamaskowane). Bit MSK jest tym samym bitem, co odpowiedni bit w rejestrze maski (Mask Register), więc jego zmiana pociąga za sobą automatyczną zmianę rejestru maski i vice versa.

15	14	. . . . .	4	3	2	1	0
0	0	. . . . .	0	MSK	PR2	PR1	PR0

Timer/DMA Control Registers in MASTER (non-iRMX_86) mode.

15	14	. . . . .	7	6	5	4	3	2	1	0
0	0	. . . . .	0	SFNM	C	LTM	MSK	PR2	PR1	PR0

INT0/INT1 Control Registers in MASTER (non-iRMX_86) mode.

15	14	. . . . .	5	4	3	2	1	0
0	0	. . . . .	0	LTM	MSK	PR2	PR1	PR0

INT2/INT3 Control Registers in MASTER (non-iRMX_86) mode.

15	14	. . . . .	4	3	2	1	0
0	0	. . . . .	0	MSK	PR2	PR1	PR0

Control Word in iRMX_86 mode.

Znaczenie poszczególnych bitów jest następujące:

- bity PR2, PR1, PR0 określają priorytet:

  000 - oznacza najwyższy priorytet,
  111 - oznacza najniższy priorytet.

- bit LTM oznacza jaki sygnał ma być interpretowany jako przerwanie:

  0 - przerwanie wyzwalane zboczem sygnału.
  1 - przerwanie wyzwalane poziomem sygnału,

- bit MSK jest bitem maski:

  0 - niezamaskowana obsługa przerwania,
  1 - zamaskowana obsługa przerwania.

- bit C = 1 oznacza, że wewnętrzny sterownik przerwań pracuje w kaskadzie,

- bit SFNM = 1 określa specjalny podstawowy podtryb pracy (special fully nested mode).

9.3.2. Rejestr zgłoszeń (Interrupt Request Register)

Rejestr zgłoszeń zawiera bity, które są odpowiednio przyporządkowane poszczególnym źródłom przerwań. Każdy bit zostaje automatycznie zapalony wtedy, gdy na odpowiednim wejściu wewnętrznego sterownika przerwań pojawi się przerwanie pochodzące z wewnętrznego układu mikroprocesora 80186 (80188) lub z urządzenia zewnętrznego (w tym ostatnim przypadku tylko w przypadku pracy w trybie MASTER). Bity w rejestrze zgłoszeń są zapalane bez względu na to, czy obsługa danego przerwania jest zamaskowana.

15	14	. . . . .	8	7	6	5	4	3	2	1	0
0	0	. . . . .	0	I3	I2	I1	I0	DMA1	DMA0	0	Timer 0

In-Service, Interrupt Request and Mask Register Format in MASTER mode.

15	14	. . . . .	6	5	4	3	2	1	0
0	0	. . . . .	0	Timer 2	Timer 1	DMA1	DMA0	0	Timer 0

In-Service, Interrupt Request and Mask Register Format in iRMX_86 mode.

W obu przypadkach bity Timer_2, Timer_1, Timer_0, DMA_1, DMA_0 mogą być odczytywane i zapisywane, natomiast bity: I3, I2, I1 i I0 związane z zewnętrznymi źródłami przerwań (tylko w trybie pracy MASTER) mogą być tylko odczytywane.

9.3.3. Rejestr maski (Mask Register)

Rejestr maski (jego format jest taki sam jak rejestru zgłoszeń) zawiera bity, których zapalenie powoduje zamaskowanie obsługi przerwania odpowiadającego danemu bitowi. Poszczególne bity w rejestrze maski odpowiadają dokładnie bitom MSK w rejestrach kontrolnych przyporządkowanych poszczególnym źródłom przerwań, a zatem zmiana danego bitu w rejestrze maski pociąga za sobą zmianę bitu MSK w rejestrze kontrolnym i vice versa.

9.3.4. Rejestr maski priorytetu (Priority Mask Register)

Rejestr maski priorytetu pozwala na maskowanie obsługi wszystkich przerwań o

priorytetach obsługi poniżej określonego poziomu. Jego zawartość określa najniższy priorytet przerwań, które mogą być obsługiwane, np. wpisanie wartości 100 (binarnie) spowoduje, że obsługa przerwań o priorytetach (binarnie) 101, 110 i 111 będzie zamaskowana. Podczas restartu mikroprocesora 80186 (80188) do rejestru maski jest wpisywana wartość 111 (binarnie) i może ona ulec zmianie tylko na skutek działania użytkownika (rejestr ten nie jest modyfikowany działaniem sprzętu).

15	14	13		3	2	1	0
0	0	0	. . . . .	0	m2	m1	m0

Priority Level Mask Register Format in MASTER (non-iRMX_86) mode.

15	14	13		3	2	1	0
0	0	0	. . . . .	0	PRM2	PRM1	PRM0

Priority Level Mask Register Format in iRMX_86 mode.

9.3.5. Rejestr obsługi (In-Service Register)

Rejestr obsługi (jego format jest taki sam jak rejestru zgłoszeń) zawiera bity odpowiadające poszczególnym źródłom przerwań, wskazujące, czy dane przerwanie zostało przyjęte do obsługi. Ustawienie bitu IS dla danego przerwania powoduje, że przerwania o niższych priorytetach nie będą przyjmowane do obsługi.

W trybie pracy iRMX_86 bity na pozycjach 0, 4 i 5 odpowiadają wewnętrznym timerom mikroprocesora 80186 (80188). W trybie pracy MASTER przerwaniom od wszystkich trzech wewnętrznych timerów odpowiada ten sam, pojedynczy bit w rejestrze obsługi IS, natomiast bity oznaczone jako I0, I1, I2 i I3 są przyporządkowane przerwaniom od zewnętrznych źródeł.

Poszczególne bity w rejestrze IS są ustawiane zawsze po potwierdzeniu przez mikroprocesor przyjęcia przerwania (albo przez ustawienie właściwego stanu na odpowiedniej linii sterującej, albo poprzez programowe badanie stanu wewnętrznego sterownika przerwań) i są gaszone komendą EOI (End-Of-Interrupt) po zakończeniu obsługi danego przerwania. Rejestr obsługi może być zarówno odczytywany jak i zapisywany przez CPU, tzn. poszczególne bity można ustawiać bez przyjęcia przerwania, jak też gasić - bez wykonania komendy EOI.

9.3.6. Rejestry stanu wewnętrznego sterownika przerwań (Poll Register i Poll Status Register) - tylko w trybie MASTER

Wewnętrzny sterownik przerwań zawiera dwa rejestry stanu: tzw. Poll Register i Poll Status Register, oba służące do przechowywania tej samej informacji. Format tych rejestrów przedstawiono poniżej:

15	14	13	. . . . .	5	4	3	2	1	0
INTREQ	0	0	. . . . .	0	S4	S3	S2	S1	S0

Poll Register Format

Bit INTREQ wskazuje, że przerwanie jest w trakcie obsługi. Zostaje on zapalony, jeśli zarejestrowano przerwanie o wystarczająco dużym priorytecie, a zgaszony - po potwierdzeniu tego przerwania. Podczas obsługi danego przerwania bity S4-S0 wskazują numer przerwania o najwyższym priorytecie obsługi spośród przerwań oczekujących na obsługę.

Odczyt Poll Register powoduje wysłanie do wewnętrznego sterownika przerwań potwierdzenia przyjęcia przerwania oczekującego na obsługę, ale nie modyfikuje żadnego rejestru kontrolnego. Oba rejestry: Poll Register i Poll Status Register można tylko odczytywać, jakiekolwiek dane wpisywane do nich nie powodują żadnego działania i nie są zapamiętywane.

Chociaż rejestry te nie są obsługiwane w trybie iRMX_86, odwołanie się do nich w tym trybie powoduje, że wewnętrzny sterownik przerwań "potwierdza" przyjęcie przerwania, tzn. ustawia odpowiedni bit w rejestrze obsługi i w rejestrze maski priorytetu.

9.3.7. Rejestr End-Of-Interrupt (EOI Register)

Rejestr EOI jest używany przez programistę do wysyłania komendy End-Of-Interrupt do wewnętrznego sterownika przerwań. Po odebraniu tej komendy wewnętrzny sterownik przerwań automatycznie gasi odpowiedni bit w rejestrze obsługi i bity w rejestrze maski priorytetu. W trybie pracy iRMX_86 dozwolona jest tylko komenda tzw. specific End-Of-Interrupt, która gasi określony explicite przez użytkownika bit IS w rejestrze obsługi przerwań.

Rejestr EOI może być tylko zapisywany, przy czym zapisana informacja nie jest zapamiętywana.

15	14	13	. . . . .	5	4	3	2	1	0
SPEC/NSPEC	0	0	. . . . .	0	S4	S3	S2	S1	S0

EOI Register Format in MASTER (non-iRMX_86) mode.

15	14	13	. . . . .	5	4	3	2	1	0
0	0	0	. . . . .	0	0	0	L2	L1	L0

EOI Register Format in iRMX_86 mode.

9.3.8. Rejestr stanu przerwania (Interrupt Status Register)

Rejestr ten zawiera ogólne informacje dotyczące wewnętrznego sterownika przerwań. Wszystkie znaczące bity tego rejestru można zapisywać i odczytywać.

15	14	13	. . . . .	5	4	3	2	1	0
DHLT	0	0	. . . . .	0	0	0	IRT2	IRT1	IRT0

Interrupt Status Register format.

Trzy bity w tym rejestrze (IRT0-IRT2) są wykorzystywane do określenia, który wewnętrzny timer w trybie pracy MASTER jest źródłem przerwania, przy czym bit związany z danym timerem zostaje automatycznie zgaszony po potwierdzeniu przyjęcia przerwania zgłoszonego przez ten timer.

Bit DHLT (DMA Halt Transfer) zabezpiecza gotowość obsługi niemaskowanych przerwań przez zawieszenie transmisji modułem DMA. Jest on automatycznie zapalany zawsze wtedy, gdy wystąpiło niemaskowane przerwanie i gaszony po wykonaniu instrukcji IRET. Bit ten może być również ustawiany przez programistę. Za wyjątkiem przypadku wykonania instrukcji IRET bit DHLT nie może być zgaszony przez automatycznie działanie mikroprocesora, a zatem dla odblokowania transmisji kanałami DMA po wystąpieniu każdego przerwania NMI programista musi go zgasić.

9.3.9. Rejestr wektora przerwań (Interrupt Vector Register) - tylko w trybie iRMX 86

Rejestr wektora przerwań jest używany do określania pięciu najbardziej znaczących bitów numeru wektora przerwań umieszczonego na szynie CPU w odpowiedzi na potwierdzenie przerwania. Wewnętrzny sterownik przerwań sam "dopisuje" trzy najmniej znaczące bity tego numeru, zgodnie z priorytetami obsługi poszczególnych przerwań. Format tego rejestru pokazano poniżej:

15	14	13	. . . . .	8	7	6	5	4	3	2	1	0
0	0	0	. . . . .	0	t4	t3	t2	t1	t0	0	1	0

Interrupt Vector Register Format

9.4. Restart mikroprocesora a wewnętrzny sterownik przerwań

Restart mikroprocesora 80186 (80188) powoduje, że jego wewnętrzny sterownik przerwań wykonuje następujące operacje:

- gasi (ustawia na "0") wszystkie bity SFNM, programując podstawowy podtryb pracy (fully nested mode),
- zapala (ustawia na "1") wszystkie bity PR2, PR1 i PR0 we wszystkich rejestrach sterujących wewnętrznego sterownika przerwań. Powoduje to ustalenie najniższego priorytetu obsługi przerwań odpowiadających tym rejestrom sterującym,
- gasi (ustawia na "0") wszystkie bity LTM, co powoduje, że przerwanie będzie wyzwalane zboczem sygnału,
- gasi (ustawia na "0") wszystkie bity w rejestrze zgłoszeń (Interrupt

Request Register) i rejestrze obsługi (In-Service Register), ustawia natomiast wszystkie "znaczące" bity w rejestrze maski (Mask Register),

- gasi (ustawia na "0") bity C w rejestrach sterujących (non-cascade),
- zapala (ustawia na "1") wszystkie "znaczące" bity w rejestrze maski priorytetu (Priority Mask Register), co oznacza, że nie jest maskowana obsługa żadnego poziomu przerwań,
- ustawia wewnętrzny sterownik przerwań mikroprocesora 80186 (80188) w trybie pracy MASTER (non-iRMX_86).

10. Dodatek C. ZAWARTOSC KATALOGOW PROGRAMU STERUJACEGO ROBOTA URp

Katalog CTRLPGM:

--- RESTART.A86
--- COP.BAT
--- GO.BAT
--- LIB.BAT
--- LIBRARY.BAT
--- AXESCNTR.C86
--- ERROR.C86
--- ERRTXT.C86
--- GLOBVAR.C86
--- GRIPOPER.C86
--- HARDERR.C86
--- INITMAIN.C86
--- MAIN.C86
--- SETSYSST.C86
--- SWTCHEXT.C86
--- SYSTEM.C86
--- TIMELAG.C86
--- VERSION.C86
--- ERRORS.H
--- EXTWORLD.H
--- MAIN.H
--- SETJMP.H
--- MAIN.LIB

Katalog CTRLPGM\EPROM:

--- ROMFILE.BAT
--- ROMLOC.CON
--- PROFILE.KED

Katalog CTRLPGM\EPROM\MONITOR:

--- MONITOFF.HEX
--- MONITON.HEX

Katalog CTRLPGM\EXTMOV:

--- LIB.BAT
--- LIBRARY.BAT
--- CALCEXT.C86
--- DEFTOOL.C86
--- DTOI.C86
--- DTOL.C86
--- EXTCONST.C86
--- INITEXTM.C86
--- INTORAD.C86
--- INWERS.C86
--- IT_BACK1.C86
--- LINEAR.C86
--- MAKE_PP.C86
--- MOVE.C86
--- M_URP6_2.C86
--- PARLIN.C86
--- RADTOIN.C86
--- STINTERP.C86
--- EXTMOV.H
--- INDEKS.H
--- LINEAR.H
--- MOVE.H
--- EXTMOV.LIB

Katalog CTRLPGM\GLOBSPAC:

--- GLOBSIZE.A86
--- LIB.BAT
--- LIBRARY.BAT
--- CLRPRLMP.C86
--- DELINS.C86
--- DELTOOL.C86
--- FINDINS.C86
--- FINDTOOL.C86
--- GETCURR.C86
--- GETEXT.C86
--- GETFRST.C86
--- GETINTR.C86
--- GETLAST.C86
--- GETMIDD.C86
--- GETNEXT.C86
--- GETPREV.C86
--- GLOBSINI.C86
--- GLOBSVAR.C86
--- INIT_GS.C86
--- PUTINSNO.C86
--- SEARCH.C86
--- SEARCHIN.C86
--- SEARPREV.C86
--- SETPRLMP.C86
--- SHIFT.C86
--- STOREINS.C86
--- STORTOOL.C86
--- UPDATIDX.C86
--- VELCOEFF.C86
--- CONST.H
--- GLOBSPAC.H
--- GLOBSPAC.LIB

Katalog CTRLPGM\HARDWARE:

--- CSRA.A86
--- DIS_IN.A86
--- ENABL_IN.A86
--- HARD_CON.A86
--- INBYTE.A86
--- INI_186.A86
--- INI_8259.A86
--- INI_Z853.A86
--- INT_TIME.A86
--- INT_Z853.A86
--- INWORD.A86
--- OPOZNIEN.A86
--- OUTBYTE.A86
--- OUTWORD.A86
--- PARAMTR.A86
--- SET_VECT.A86
--- UNEX_INT.A86
--- LIB.BAT
--- LIBRARY.BAT
--- AXISSTOP.C86
--- BLINK.C86
--- COUNTER2.C86
--- CTRLRDY.C86
--- EXTINGSH.C86
--- GETCLOCK.C86
--- GRIPPERS.C86
--- HARDCONS.C86
--- HARDVAR.C86
--- INITHARD.C86
--- INPOS.C86
--- LIGHT.C86
--- READAPOS.C86
--- SENDINC.C86
--- SNDCORIN.C86
--- STOPHSYN.C86
--- STRTHSYN.C86
--- SYNCHRND.C86
--- USERIN.C86
--- USEROUT.C86
--- HARDWARE.H
--- HARDWARE.LIB

Katalog CTRLPGM\INTMOV:

--- DIVIDE.A86
--- LIB.BAT
--- LIBRARY.BAT
--- INITINTM.C86
--- INRANGE.C86
--- INTMCONS.C86
--- INTMVAR.C86
--- MICROSTP.C86
--- QUASILIN.C86
--- SENDABS.C86
--- SIMPLSTP.C86
--- SIVCNSTR.C86
--- STOP.C86
--- ARM.H
--- INTMOV.H
--- MICROSTP.H
--- QUASILIN.H
--- SENDABS.H
--- SIVCNSTR.H
--- INTMOV.LIB

Katalog CTRLPGM\INTRPRTR:

--- LIB.BAT
--- LIBRARY.BAT
--- AUTOSERV.C86
--- CLEAN.C86
--- DOEXTRA.C86
--- DOINSTR.C86
--- INIINTRP.C86
--- INTRPCNS.C86
--- INTRPRTR.C86
--- INTRPVAR.C86
--- INTRRPTD.C86
--- IS.C86
--- JUMPCOND.C86
--- VALUE1B.C86
--- WAITCOND.C86
--- WAITSTAT.C86
--- WTFINPOS.C86
--- INSCODES.H
--- INSTRUCT.H
--- INTRPRTR.H
--- TYPES.H
--- WAITCOND.H
--- INTRPRTR.LIB

Katalog CTRLPGM\INTRPRTR\INSTR:

--- LIB.BAT
--- LIBRARY.BAT
--- CALL.C86
--- DEFV.C86
--- EMPTY.C86
--- ENDPGM.C86
--- GRIP.C86
--- JUMP.C86
--- JUMPEQ.C86
--- JUMPNE.C86
--- LOOP.C86
--- LPEND.C86
--- NOOP.C86
--- POSL.C86
--- POSQ.C86
--- RET.C86
--- SET.C86
--- STRPGM.C86
--- SUBR.C86
--- TOOL.C86
--- WAIT.C86
--- WAITEQ.C86
--- WAITNE.C86
--- INSTR.H
--- INSTR.LIB

Katalog CTRLPGM\MANMOV:

--- LIB.BAT
--- LIBRARY.BAT
--- CARFRAME.C86
--- GETDEFL.C86
--- INITMANM.C86
--- INTFRAME.C86
--- MANMCONS.C86
--- MANMOV.C86
--- MANMVAR.C86
--- NEWORNT0.C86
--- NEWPOS0.C86
--- NEWTCP0.C86
--- PGMBUTT.C86
--- SWITCHFR.C86
--- MANMOV.H
--- MANMOV.LIB

Katalog CTRLPGM\MATH87:

--- FARCCOS.A86
--- FATAN2.A86
--- FCOSSIN.A86
--- FRNDINT.A86
--- FSQRT.A86
--- MATHCONS.A86
--- VECTCOPY.A86
--- LIB.BAT
--- LIBRARY.BAT
--- VECTMOD.C86
--- MATH87.H
--- MATH87.LIB

Katalog CTRLPGM\OPERPNL:

--- OPER_TAB.A86
--- LIB.BAT
--- LIBRARY.BAT
--- AUTOWORK.C86
--- DONOTHNG.C86
--- EMERSTOP.C86
--- GO_ON.C86
--- LOADPGM.C86
--- MANUWORK.C86
--- OPERPNL.C86
--- PGMSTART.C86
--- STOPERR.C86
--- STOPPGM.C86
--- STOPSYN.C86
--- SYNCHRNZ.C86
--- OPERPNL.LIB

Katalog CTRLPGM\PROGPNL:

--- DECINSPR.A86
--- LIB.BAT
--- LIBRARY.BAT
--- ATOD.C86
--- CHKEYS.C86
--- CLRLINE.C86
--- DECINSTR.C86
--- DECNPAR.C86
--- DIODEOFF.C86
--- DIODEON.C86
--- DTOA.C86
--- D_EMPTY.C86
--- D_STRANG.C86
--- ESDOFF.C86
--- NUMBER.C86
--- OPTCHNG.C86
--- OPTREAD.C86
--- PANELINI.C86
--- PANELVAR.C86
--- READNUM.C86
--- RESTCURS.C86
--- SAVECURS.C86
--- SETCURS.C86
--- TCBTAB.C86
--- WAITFKEY.C86
--- BUTTONS.H
--- CHKEY.H
--- CURSOR.H
--- DECINSTR.H
--- DIODES.H
--- NUMBER.H
--- PROGPNL.H
--- PROGPNL.LIB

Katalog CTRLPGM\PROGPNL\EDIT:

--- EDIT_TAB.A86
--- LIB.BAT
--- LIBRARY.BAT
--- BACKWARD.C86
--- DELETE.C86
--- EDIT.C86
--- FORWARD.C86
--- PAREDIT.C86
--- RENUMER.C86
--- SETINSNO.C86
--- DELETE.H
--- EDIT.LIB

Katalog CTRLPGM\PROGPNL\MANOPER:

--- MANO_TAB.A86
--- LIB.BAT
--- LIBRARY.BAT
--- ACTPOS.C86
--- FREEMEMO.C86
--- KOMPUTER.C86
--- MANOPER.C86
--- MEMSERV.C86
--- SETMVELO.C86
--- SETREFPO.C86
--- STOREREF.C86
--- TOOLDEF.C86
--- USER_IO.C86
--- MANOPER.LIB

Katalog CTRLPGM\PROGPNL\OTHERINS:

--- OTHE_TAB.A86
--- LIB.BAT
--- LIBRARY.BAT
--- DECOBO.C86
--- DECREL.C86
--- D_CALL.C86
--- D_ENDPGM.C86
--- D_ENDREP.C86
--- D_GRIPP.C86
--- D_JUMP.C86
--- D_OUFLAG.C86
--- D_REPEAT.C86
--- D_RET.C86
--- D_STRPGM.C86
--- D_SUBR.C86
--- D_TOOL.C86
--- D_VELOC.C86
--- D_WAIT.C86
--- OTHERINS.C86
--- PGMOBO.C86
--- PGMREL.C86
--- P_CALL.C86
--- P_ENDPGM.C86
--- P_ENDREP.C86
--- P_GRIPP.C86
--- P_JUMP.C86
--- P_OUFLAG.C86
--- P_REPEAT.C86
--- P_RET.C86
--- P_STRPGM.C86
--- P_SUBR.C86
--- P_TOOL.C86
--- P_VELOC.C86
--- P_WAIT.C86
--- OTHERINS.H
--- OTHERINS.LIB

43

Katalog CTRLPGM\PROGPNL\POSINS:

--- LIB.BAT
--- LIBRARY.BAT
--- D_POS.C86
--- POSINS.C86
--- P_POS.C86
--- POS.H
--- POSINS.LIB

Katalog CTRLPGM\PROGPNL\START:

--- LIB.BAT
--- LIBRARY.BAT
--- CONTPGM.C86
--- SETAVEL.C86
--- SETSYMUL.C86
--- START.C86
--- STARTPER.C86
--- STARTPGM.C86
--- STEPPGM.C86
--- START.LIB

Katalog CTRLPGM\STDIO:

--- LIB.BAT
--- LIBRARY.BAT
--- GETC.C8
--- GETCHAR.C8
--- UNGETC.C8
--- UNGETCH.C8
--- IOINIT.C86
--- IOVARIAB.C86
--- READ.C86
--- STREAMS.C86
--- UNLOCK.C86
--- WRITE.C86
--- BUFFERS.H
--- CHARCODE.H
--- SERIAL.H
--- STDIO.H
--- STDIO.LIB

Katalog CTRLPGM\STDIO\KONSOLA:

--- BLANK.A86
--- BYTE_CON.A86
--- CONSTANS.A86
--- DECDOUBL.A86
--- DEC_BYTE.A86
--- DEC_WORD.A86
--- GETCH_EX.A86
--- GETCH_WA.A86
--- MESSAGE.A86
--- NEW_LINE.A86
--- PUTCH_EX.A86
--- WORD_CON.A86
--- LIB.BAT
--- LIBRARY.BAT
--- KONSOLA.LIB

Katalog CTRLPGM\STDIO\MASMEM:

--- BRAK_PLK.A86
--- CALCADDR.A86
--- DEL_PROG.A86
--- DIR.A86
--- FORMAT.A86
--- INTRKLAW.A86
--- LOAD.A86
--- NAZWA.A86
--- PAR_TRAN.A86
--- POJEMN.A86
--- PROTECT.A86
--- READEEPR.A86
--- READ_MAS.A86
--- RENAME.A86
--- SEND.A86
--- SRCHNUMB.A86
--- STALE.A86
--- TABLICE.A86
--- WRITEEPR.A86
--- WRITE_MA.A86
--- LIB.BAT
--- LIBRARY.BAT
--- READMEM.C86
--- WRITEMEM.C86
--- MASMEM.LIB

Katalog CTRLPGM\STDIO\PANEL:

```
--- BLANKPNL.A86        --- LIB.BAT
--- BYTEPNL.A86         --- LIBRARY.BAT
--- CLRLINE.A86         --- ESC_SERV.C86
--- CURSLEFT.A86        --- GC_PANEL.C86
--- DECBYTPN.A86        --- MKIMAGE.C86
--- DECDPNL.A86         --- MOVEKPAD.C86
--- DECWORPN.A86        --- PC_PANEL.C86
--- GC_PNL_.A86         --- POPIMAGE.C86
--- GC_WAIT_.A86        --- PR_CHAR.C86
--- GETCHPNL.A86        --- PR_STR.C86
--- MESSPNL.A86         --- PSHIMAGE.C86
--- PRCHAR_.A86         --- PUTTOBUF.C86
--- PRSTR_.A86          --- RD_PANEL.C86
--- PUTCHPNL.A86        --- WR_PANEL.C86
--- REJESPNL.A86        --- IMAGE.H
--- WORDPNL.A86         --- PANEL.LIB
```

Katalog CTRLPGM\STDIO\RETARGER:

```
--- LSEEK.A86           --- EXIT.C86
--- MALLOC.A86          --- ISATTY.C86
--- MAP_LENG.A86        --- STDIO.C86
--- SEMAPHOR.A86        --- STDOPEN.C86
--- LIB.BAT             --- THREAD.C86
--- LIBRARY.BAT         --- RETARGER.LIB
--- CONIO.C86
```

Katalog CTRLPGM\TO_RAM:

```
--- RAMFILE.BAT         --- RAMLINK.CON
--- RAMLINK.BAT         --- RAMLOC.CON
--- RAMLOC.BAT          --- PROFILE.KED
```

Katalog CTRLPGM\UTILITY:

```
--- ADDMEM.A86          --- MOV_BYTE.A86
--- ADDRESS.A86         --- PARAMETR.A86
--- CHAR_INT.A86        --- PAUSE.A86
--- DUMP.A86            --- POINTEXT.A86
--- INCRMEM.A86         --- REJESTRY.A86
--- INT_CHAR.A86        --- LIB.BAT
--- LAMPKI.A86          --- LIBRARY.BAT
--- MONITOR.A86         --- UTILITY.LIB
```