7113

PRZEMYSŁOWY INSTYTUT AUTOMATYKI I POMIARÓW

PIAP

440

Al.Jerozolimskie 202 02-486 Warszawa

A

Samodzielna Pracownia Oprogramowania Systemów

Główny wykonawca: mgr inż. Zbigniew Pilat

Wykonawcy: mgr inż. Małgorzata Jacórzyńska Śmigiera

Konsultant:

Nr zlecenia: S1311A

Tytuł: Rozwój oprogramowania podstawowego robotów URP. Etap 9: Opis programu sterującego w zakresie komunikacji jednostki centralnej z urządzeniami peryferyjnymi.

Zleceniodawca: praca wykonywana w ramach działalności statutowej.

Pracę rozpoczęto dnia: 1994.06.01. zakończono dnia: 1994.08.16.

Kierownik Ośrodka

mgr inż. Zbigniew Pilat

Dyrektor Pionu

dr inż. Jan Jabłkowski

Praca zawiera:

stron	37
rysunków	--
fotografii	--
tabel	--
tablic	--
załączników	1 (listing oprogramowania tylko w egz. POS)

Rozdzielnik - ilość egz.:

Egz. 1 BOINTE
Egz. 2 POS
Egz. 3 POS
Egz. 4
Egz. 5
Egz. 6

nr rejestr. 7113

Analiza deskryptorowa:

ROBOTY PRZEMYSŁOWE, UKŁAD STEROWANIA, OPROGRAMOWANIE

Analiza dokumentacyjna:

Sprawozdanie zawiera opis programu sterującego robotów rodziny URP. Jest kontynuacja dokumentacji oprogramowania podstawowego robotów rozpoczętej w spr. PIAP nr rej 6978 i kontynuowanej w spraw. PIAP nr rej. 7064. Niniejszy dokument obejmuje komunikowanie się jednostki centralnej układu sterowania robota w urządzeniami zewnętrznymi, a więc panelem programowania, komputerem, blokami wejść/wyjść dwustanowych, blokami cyfrowych sterowaników osi i pakietem pamięci masowej.

Tytuły poprzednich sprawozdań:

Nr rej. 6978 - Etap 5: Opracowane opisu programu sterującego robotów URP w zakresie organizacji programu, wykorzystania zasobów sprzętowych sterownika, inicjalizacji systemu sterowania i pętli głównej programu. PIAP, lipiec 1993r.

Nr rej. 7064 - Etap 7: Opracowanie opisu programu sterującego robotów URP w zakresie języka programowania, interpretera, wykonywania instrukcji oraz modelu kinematyki i generatora trajektorii.

1. Wstęp

Roboty URP tworzą najnowszą rodzinę robotów przemysłowych oferowanych przez PIAP. W stosunku do poprzednich modeli serii IRb, IRp wyróżnia je przede wszystkim nowoczesny, bazujący na 16-bitowym systemie wieloprocesorowym, układ sterowania. Podstawowe prace konstrukcyjne nad tym układem, zarówno hardware'owe jak i software'owe zakończono na początku 1993 roku. Dokumentacja konstrukcyjna powstawała na bieżąco i po badaniach należało ją tylko zweryfikować. Na bieżąco również było zmieniane i uaktualniane podstawowe oprogramowanie w postaci źródłowej. Aby oprogramowanie to mogło być jednak w pełni czytelne, a w szczególności aby prace nad nim mogły w przyszłości przejąć inne zespoły, niezbędne jest wykonanie opisu, który wyjaśni idee, pomysły, zasady, które twórcy starali się w nim zrealizować. Tworzenie takiej dokumentacji oprogramowania rozpoczęto już w roku 1993. Praca wykonywana jest stopniowo. Do tej pory powstały dwa opracowania, przekazane do archiwum PIAP w formie sprawozdań:

- nr rej. 6978 - lipiec 1993 - zawierające opis następujących zagadnień:
  - konfiguracja sprzętowa sterownika robotów URP,
  - tworzenie programu sterującego,
  - organizacja postaci źródłowej oprogramowania,
  - ogólna struktura programu sterującego,
  - inicjalizacja układu sterowania robota,
  - pętla główna programu sterującego.
- nr rej. 7064 - luty 1994 - zawierające opis następujących zagadnień:
  - zmiany w organizacji postaci źródłowej programu,
  - język programowania robotów URP,
  - interpreter,
  - realizacja instrukcji programu użytkowego,
  - model kinematyki i generator trajektorii.

Niniejsze opracowanie ma być w zamyśle autorów kontynuacją poprzednich, tworząc wraz z nimi integralną całość. Obejmuje ono opis komunikacji jednostki centralnej układu sterowania robota z urządzeniami peryferyjnymi. Należy zwrócić uwagę, że ten dokument został stworzony przy użyciu innego niż poprzednie edytora - Word for Windows. Wydaje się, że w przyszłości należałoby również dwa wspomniane wyżej opracowania przenieść do tego edytora, który ma duże możliwości przeglądania, sortowania, wyszukiwania tekstu, a także umożliwia w prosty spopsób tworzenie różnego rodzaju wykazów, list, skorowidzów itp., co może być bardzo przydatne osobom korzystającym z tego opisu.

2. Środowisko sprzętowe jednostki centralnej układu sterowania robota

Jednostka centralna MV52, bazująca na 16-bitowym mikroprocesorze Intel 80186 (z koprocesorem arytmetycznym) pełni w systemie sterowania robota URP rolę komputera głównego, zarządzającego. Realizując program sterujący jednostka centralna komunikuje się z urządzeniami peryferyjnymi. Możliwe są dwa sposoby wymiany informacji:

- poprzez magistralę systemową AMS-BUS,
- poprzez kanał transmisji szeregowej RS-232.

W standardowej wersji układu sterowania robota URP, jak to przedstawiono na rys. 2.1., występują następujące urządzenia peryferyjne:

- bloki wejść/wyjść dwustanowych MV12,
- cyfrowe sterowniki osi MV20,
- pakiet pamięci masowej MV62,
- panel programowania robota,
- komputer zewnętrzny (lub inne urządzenie np. drukarka wyposażone w kanał transmisji szeregowej RS-232).

Zostaną one teraz pokrótce omówione. Szersze informacje można znaleźć w przywoływanej literaturze i dokumentacji konstrukcyjnej układu sterowania robotOw URP.

Blok wejść/wyjść dwustanowych MV12

W układzie sterowania URP są dwa bloki MV12:

- systemowy - przeznaczony do obsługi lampek i przycisków na panelu operacyjnym robota oraz wewnętrznych dwustanowych sygnałów sterujących (np. Control_Ready - blokujący podtrzymanie zasilania silników robota),
- użytkowy - przeznaczony do wysterowania i odebrania informacji z urządzeń technologicznych stanowiska zrobotyzowanego, dołączonych przez użytkownika.

Każdy blok MV12 ma 16 wejść i 16 wyjść [a].

Cyfrowe sterowniki osi MV20

W układzie sterowania URP do każdej osi musi być jeden cyfrowy sterownik MV20. Sterowniki te są urządzeniami inteligentnymi, wykorzystują procesor Intel 80186. Od strony jednostki centralnej MV20 widziany jest jako dwa porty wejściowe i dwa porty wyjściowe (po 16 bitów każdy). Ich funkcje są następujące:

- wejściowy port danych - jednostka centralna odczytuje z niego aktualne położenie danej osi,
- wejściowy port stanu - jednostka centralna odczytuje z niego aktualny stan osi, np. czy oś jest zsynchronizowana, czy jest w sterfie zerowej itp.,
- wyjściowy port danych - jednostka centralna zapisuje do tego portu słowo zadanego przyrostu pozycji osi wraz z zakodowanym żądanym czasem jrealizacji tego przyrostu,

4

- wyjściowy port sterujący - jednostka centralna zapisując odpowiednie słowo do tego portu wymusza konkretne działanie sterownika osi, np. synchronizację osi, natychmiastowe zatrzymanie ruchu itp.

Dokładne informacje o budowie, działaniu i obsłudze sterownika MV20 można znaleźć w jego dokumentacji [b].

Pakiet pamięci masowej MV62

Jedną z istotnych nowości w układzie sterowania robotów URP było wyeliminowanie magnetofonów i zastosowanie jako pamięci masowej programów użytkowych bloku pamięci półprzewodnikowej MV12. Pakiet ten od strony konstrukcyjnej zawiera 8 par podstawek pod układy pamięci kasowanej elektrycznie EEPROM typu 28256. Każda para ma pojemność 64 kbajtów. Maksymalna pojemność pakietu MV12 wynosi więc 8x64 = 512 kbajtów. W pakiecie muszą być obsadzone przynajmniej pierwsze dwie pary podstawek, czyli minimalna pojemność wynosi 2x64 = 128 kbajtów. Jednostka centralna widzi pakiet jako okno o szerokości 64 kbajtów. Wysyłając do MV12 odpowiednie słowo sterujące można mieć przez to okno dostęp do innego fizycznego obszaru pamięci pakietu. W celu ochrony danych przed zniszczeniem w pakiecie MV62 zastosowano sprzętową protekcję zapisu. Przełącznik zezwolenie/zakaz zapisu, wraz z informacyjną diodą świecącą umieszczono na płycie czołowej pakietu. Więcej informacji o pakiecie pamięci masowej można znaleźć w literaturze [c, d, k].

Panel programowania robota i komputer zewnętrzny

Panel programowania robota jest inteligentnym terminalem realizującym w układzie sterowania robota URP sprzężenie człowiek-maszyna. Sercem panelu jest procesor Intel 8051. Steruje on pracą panelu zgodnie z programem zapisanym w pamięci EPROM. Panel komunikuje się z jednostką centralną za pośrednictwem łącza szeregowego RS232. W dużym uproszczeniu można powiedzieć, że panel wysyła do jednostki centralnej informacje o stanie swych przycisków i wyświetla na dwuwierszowym display'u alfanumerycznym teksty przysłane z jednostki. Panel ma zamontowany również przycisk stopu awaryjnego, dołączony do zasadniczego obwodu w szafie robota i obsługiwany sprzętowo. Oprogramowanie panelu oprócz zasadniczej części sterującej zawiera także funkcje testowe i autodiagnostyczne, działające w trybie off-line, a niektóre (np. Watchdog) w trybie on-line. [e, f].

Panel programowania jest dołączony do kanału szeregowego RS232 jednostki MV52. Jednostka centralna ma dwa kanały transmisji szeregowej. Kanał obsługujący panel, oznaczony literą B może pracować jako prądowy z oddzieleniem galwanicznym i ten tryb jest wykorzystywany w układzie sterowania robota, może także, po odpowiedniej zmianie połączeń krosowych, pracować jako interfejs napięciowy (bipolarny). Drugi kanał, oznaczony literą A pracuje zawsze jako napięciowy [g]. Całość zadań obsługi transmisji w obu kanałach realizuje układ scalony Z8530 firmy Thomson [h].Układ musi być wstępnie inicjalizowany, a parametry jego pracy można ustawiać w sposób programowy.

3. Inicjalizacja komunikacji z urządzeniami peryferyjnymi

Obsługa urządzeń peryferyjnych musi być inicjowana zarówno w zakresie sprzętu jak i oprogramowania. Inicjalizacja ta jest realizowana w początkowej fazie działania programu sterującego. Po włączeniu zasilania sterowanie jest przekazywane do procedury restart() (zb. restart.a86 w katalogu CTRLPGM). W niej są wywoływane kolejno podprogramy ustawiania wektora przerwań, inicjalizacji procesora 80186 .oraz koprocesora 8087. Następnie inicjowany jest sterownik komunikacyjny Z8530. Realizuje to procedura ini_z853 (w zb. ini_z853.a86 w katalogu CTRLPGM/HARDWARE). W dalszej kolejności inicjuje się sterownik przerwań, a następnie strumienie formatowanych we/wy i bufory we/wy. Dzięki temu możliwe jest zarówno wysyłanie komunikatów do komputera zewnętrznego jak i na panel programowania przy pomocy standardowych funkcji printf().

Po zakończeniu wstępnej inicjalizacji systemu sterowania program wchodzi do pętli głównej - procedury main() (zb. main.c86 w katalogu CTRLPGM). W niej, na początku wywoływany jest podprogram initmain() (zb. initmain.c86 w katalogu CTRLPGM) - inicjalizacja sterowania robotem. Należy wyraźnie zaznaczyć różnicę między inicjalizacją wstępną w procedurze restart() a inicjalizacją końcową w initmain(). W pierwszym przypadku inicjalizowany jest hardware i podstawowe funkcje, mechanizmy software'owe niezbędne, aby sprzęt ten ożył. W drugim przypadku wykonywane są szczegółowe zabiegi inicjujące pod kątem konkretnych funkcji wymaganych z punktu widzenia wykorzystania układu sterowania do sterowania robotem przemysłowym URP. W procedurze initmain() ustawiane są więc wartości początkowe zmiennych globalnych, zadawany jest stan wyjść (np. sygnał Control Ready podtrzymujący zasilanie stopni końcowych mocy), inicjowany jest także obszar programów użytkowych zgodnie z zaprojektowaną jego strukturą. Wykonywane są również pewne czynności związane z komunikacją z urządzeniami peryferyjnymi. Inicjowany jest sprzęt układu sterowania robota. Programowo zeruje się sterowniki położenia osi MV20. Ustawia się stan wyjść użytkownika na zero. Nawiązywana jest łączność z panelem programowania (wymiana przesyłek - panelini()). Jednostka centralna sprawdza przy okazji, czy panel w ogóle jest dołączony. Inicjowane są również pewne mechanizmy i struktury programowe. Jednostka centralna inicjuje obsługę ręcznego poruszania robotem (initmanm()), inicjowany jest ruch we współrzędnych wewnętrznych (init_intmove()) i zewnętrznych (init_extmove()).g

4. Oprogramowanie realizujące komunikację z urządzeniami peryferyjnymi

4.1. Obsługa wejść/wyjść dwustanowych

Wejścia/wyjścia systemowe

Blok wejść/wyjść MV12 jest podzielony na dwie części: starszą i młodszą, po osiem linii wejściowych i wyjściowych. Obie części są oddzielnie adresowane. Adresy zapisane są w zbiorze hardconst.c86 w katalogu HARDWARE i zadeklarowane są w programie sterującym jako zmienne globalne. Adresy te dla bloku systemowego są następujące:

- const unsigned int operating_panel_in1 = 0xF900 - młodsza część wejść,
- const unsigned int operating_panel_in2 = 0xF904 - starsza część wejść,
- const unsigned int operating_panel_out1 = 0xF908 - młodsza część wyjść,
- const unsigned int operating_panel_out2 = 0xF90C - starsza część wyjść.

Przez długi czas w układzie sterowania robota URP wykorzystywana była tylko młodsza część bloku systemowego MV12. Do obsługi młodszych wyjść dwustanowych systemowych służą trzy procedury z katalogu CTRLPGM/HARDWARE:

- light(n) - włączanie wyjść,
- extingsh(n) - gaszenie wyjść,
- blink(n) - ustawianie wyjść w stan mrugania.

Parmetrem wejściowym wszystkich tych procedur jest liczba całkowita 'n'. Poszczególne bity jej młodszego bajtu odpowiadają za poszczególne linie wyjściowe. W ww procedurach na podstawie wartości wejściowej parametru 'n' modyfikowane są odpowiednio zmienne globalne 'lamps' i 'blink'. Obie te zmienne są typu całkowitego i są zadeklarowane w pliku hardvar.c86, w katalogu CTRLPGM/HARDWARE. Poszczególne bity ich młodszego słowa odpowiadają poszczególnym liniom wyjściowym bloku systemowego MV12, podobnie jak w przypadku zmiennej 'n'. Jeśli konkretny bit zmiennej 'lamps' jest ustawiony na '1', to oznacza, że dane wyjście jest włączone, a jeśli na zero - wyłączone. Jeśli dany bit zmiennej blink jest ustawiony na '1', to oznacza, że to wyjście jest w stanie mrugania (obsługa lampek na panelu operacyjnym szafy robota). Za organizację mrugania (zapalanie i wygaszenie wtyjścia na odpowiedni okres czasu) odpowiedzialna jest funkcja counter2() (w pliku counter2.c86, w katalogu CTRLPGM/HARDWARE). Jest to procedura przerwaniowa. Realizuje ona także uaktualnianie zegara systemowego. Związane z nią przerwanie jest generowane przez wewnętrzny timer procesora 80186 co 20 ms (por. [i], rozdz. 6.1. Inicjacja mikroprocesora 80186). Okres mrugania określa zmienna 'blinking_period'. Podczas inicjalizacji w procedurze inithard() jest ona ustawiana na 15, co przy okresie zegara 20 ms oznacza czas pełnego cyklu mrugnięcia 300 ms. Wypełnienie pojedynczego błysku, tzn. stosunek czasu zapalenia do czasu wygaszenia, jest ustawiany zmienną 'switch_off'. Obecnie jest ona ustawiana w inithard() na 'blinking_period/3', co oznacza, że w pojedynczym cyklu błysku 1/3 czasu trwa świecenie, a 2/3 - wygaszenie.

Wejścia/wyjścia użytkowe

Adresy bloku użytkowego MV12 zapisane są również w zbiorze hardconst.c86 w katalogu HARDWARE i zadeklarowane są w programie sterującym jako zmienne globalne. W obecnej wersji programu sterującego URP obsługiwane są tylko wejścia/wyjścia z jednego bloku MV12.

Konstrukcja elektroniczna przewiduje możliwość opcjonalnego zastosowania drugiego takiego pakietu. W pliku hardconst.c86 zarezerowano już dla niego adresy:

- const unsigned int user_i_address[4]:
  - 0xFA00 - wejście nr: 0 - 7 - młodsza część podstawowego bloku wejść/wyjść użytkownika
  - 0xFA04 - wejście nr: 8 - 15 - starsza część podstawowego bloku wejść/wyjść użytkownika
  - 0xFB00 - wejście nr: 16 - 23 - młodsza część dodatkowego bloku wejść/wyjść użytkownika
  - 0xFB04 - wejście nr: 24 - 31 - starsza część dodatkowego bloku wejść/wyjść użytkownika
- const unsigned int user_o_address[4]:
  - 0xFA08 - wyjście nr: 0 - 7 - młodsza część podstawowego bloku wejść/wyjść użytkownika
  - 0xFA0C - wyjście nr: 8 - 15 - starsza część podstawowego bloku wejść/wyjść użytkownika
  - 0xFB08 - wyjście nr: 16 - 23 - młodsza część dodatkowego bloku wejść/wyjść użytkownika
  - 0xFB0C - wyjście nr: 24 - 31 - starsza część dodatkowego bloku wejść/wyjść użytkownika

Do obsługi wejść/wyjść użytkownika przeznaczone są dwie procedury w katalogu CTRLPGM/HARDWARE:

userin() - w zbiorze userin.c86 - odczyt stanu wejść, procedura zwraca wartość całkowitą, w której poszczególnym bitom odpowiadają poszczególne linie wejściowe,

userout() - w zbiorze userout.c86 - ustawiane wyjść, argumentem jest zmienna całkowita, w której poszczególnym bitom odpowiadają poszczególne linie wyjściowe.

4.2. Komunikacja z cyfrowymi sterownikami położenia osi MV20

Jak już wspomniano wyżej komunikacja pomiędzy jednostką centralną MV52 a cyfrowymi sterownikami osi MV20 odbywa się na zasadzie zapisu i odczytu przez jednostkę odpowiednich słów 16-bitowych. Adresy portów w poszczególnych sterownikach są zapisane w zmiennych globalnych, które są zadeklarowane w zbiorze hardcnst.c86:

- const unsigned int axis_control[9] - adresy wejściowych i wyjściowych słów sterujących,
- const unsigned int axis_data[9] - adresy wejściowych i wyjściowych słów danych,
- const unsigned int axis_restart[9] - adresy restartu - dokonanie zapisu pod ten adres dowolnej wartości powoduje automatycznie restart programu sterownika położenia osi.

Wymienione tablice zawierająadresy dla pięciu osi robota URP-6/60 (FI, TETA, ALFA, T, V) oraz adresy dla sterowników kolejnych osi, o które opcjonalnie można rozszerzyć konstrukcję robota.

Bezpośrednio po starcie systemu sterowania, w procedurze inithard() sterowniki osi są restartowane. Następnie program odczekuje 0,5 sekundy i sprawdza czy sterowniki nie zgłaszają sygnału Watchdog (oznacza to, że program sterownika nie pracuje - zła inicjalizacja). Zabieg taki może być powtórzony trzykrotnie. Jeśli po trzeciej próbie którykolwiek ze sterowników zgłasza błąd inicjalizacji, sygnalizowany jest błąd (jeśli błędnie nicjalizuje się więcej niż jeden sterownik, na panelu wyświetlany jest komunikat informujący o pierwszym sterowniku, wg kolejności osi, licząc od podstawy robota). Stan sterowników jest kontrolowany cyklicznie przez program sterujący w trakcie normalnej pracy systemu (procedura axes_controller() w zbiorze axescntr.c86, w katalogu CTRLPGM. Wykrycie jakiegokolwiek błędu jest natychmiast sygnalizowane. Do przekazywania informacji o zaistnieniu błędu sprzętowego służy zmienna globalna 'hardware_error'. Zmienna ta może być ustawiana w różnych miejscach programu sterującego. W cyklicznie wywoływanej procedurze system() zmienna ta jest kontrolowana i jeśli jej wartość jest różna od '0', wywoływana jest procedura sygnalizacji błędu - error()..

4.3. Obsługa pamięci masowej

Programy użytkowe w robocie URP są przechowywane w pamięci półprzewodnikowej typu EEPROM, w specjalizowanym pakiecie MV62. Procedury odpowiedzialne za realizację obsługi pamięci masowej zostały zgromadzone w odrębnym katalogu CTRLPGM/STDIO/MASMEM. Pamięć zainstalowana na pakiecie MV62 jest zorganizowana podobnie jak to ma miejsce w systemach plików np. pod kontrolą systemu operacyjnego DOS. Cała pamięć jest podzielona na sektory, każdy o pojemności 256 bajtów. Podczas formatowania pakietu (wtedy sprawdzana jest obsada) system sprawdza kolejne sektory, czy są sprawne. Realizuje to procedura format w pliku format.a86. W pamięci wyróżniona jest tzw. 'zerowa strona'. W niej zapisane są informacje o rzeczywistej obsadzie pakietu, czyli ile faktycznie pamięci zostało na pakiecie zainstalowane, o zapisanych programach użytkowych robota i ewentualnych uszkodzeniach sektorów wykrytych w trakcie formatowania i normalnej pracy. Informacja ta zapisana jest w dwóch tablicach zadeklarowanych w zbiorze tablice.a86. Pierwsza z nich, o nazwie SEKTOR_TABLE[] ma 2024 elementy 2-bajtowe. Każdy element odpowiada jednemu sektorowi w przestrzeni 512 kbajtów. W elementach tablicy SEKTOR_TABLE[] zakodowane są następujące informacje:

- czy dany sektor jest sprawny, czy uszkodzony - drugi przypadek obejmuje także sytuację fizycznego braku sektora {podstawki nieobsadzone układami EEPROM),
- czy dany sektor jest zajęty (tzn. informacja w nim zapisana jest istotna, jest częścią jakiegoś programu użytkowego przechowywanego w pamięci masowej), czy wolny,
- do jakiego pliku (programu użytkowego) należy wskazywany sektor.

Druga tablica, NAME_TABLE[] ma elementy 8-bajtowe i zawiera nazwy programów zapisanych w pamięci masowej. W pakiecie może być przechowywanych do 256 programów użytkowych (plików). Tablica NAME_TABLE ma więc 256 elementów. Długość pojedynczego programu, z punktu widzenia organizacji pakietu MV-62, ograniczona jest tylko pojemnością wolnej pamięci masowej. Praktycznie aplikacyjny program robotowy jest nie dłuższy niż 48 bajty, co wynika z przyjętej koncepcji programu sterującego robota. Każdy program ma swoją nazwę, będącą kombinacją 8 znaków wprowadzonych z klawiatury numerycznej panelu programowania robota podczas zapisu. Wszelkie odwołania do programu zapisanego w pamięci masowej (zapis, odczyt, zmiana nazwy etc) użytkownik wykonuje podając nazwę tego programu. Każdej nazwie programu użytkowego program obsługi pamięci masowej przyporządkowuje numer pliku. Ten numer jest wpisywany do elementu w tablicy

SEKTOR_TABLE, co w sposób jednoznaczny określa przynależność sektora do konkretnego pliku (programu użytkowego).

Poniżej zostaną wyszczególnione funkcje dostępne dla operatora robota, które służa do obsługi pamięci masowej zrealizowanej na bazie pakietu MV-62. Wyszczególniono także procedury odpowiedzialne za realizację poszczególnych funkcji.

- Przepisywanie programu z pamięci wewnętrznej do pamięci masowej - funkcja ZAPIS, procedura write_mas(), w pliku write_ma.a86 - przeznaczona jest do zapamiętania w pamięci masowej programów użytkowych. Każdy program zapisywany do pamięci masowej ma swoją nazwę, którą to nazwę operator musi wprowadzić przed wykonaniem zapisu.
- Przepisywanie programu z pamięci masowej do pamięci wewnętrznej -funkcja ODCZYT, procedura read_mas(), w pliku read_mas.a86 - przeznaczona jest do odczytania, uprzednio zapamiętanego w pamięci masowej programu pracy robota. Podobnie jak przy zapisie trzeba podać nazwę odczytywanego programu.
- formatowanie pakietu pamięci masowej - funkcja FORMAT, procedura format(), w pliku format.a86 - sprawdzana jest obsada pakietu układami EEPROM , testowane są wszystkie komórki pamięci masowej na zapis/odczyt, sporządzana jest mapa dostępnej pamięci masowej i zapamiętywany jest jej rozmiar, inicjowana jest tablica SEKTOR_TABLE[].
- przeglądanie pamięci masowej - funkcja DIR, procedura dir(), w pliku dir.a86 - wyświetlane są nazwy i długości poszczególnych programów zapisanych w pakiecie pamięci masowej. Dla celów diagnostycznych podawany jest również numer sektora pamięci masowej, w którym zapisany jest początek programu. Przewidziano opcję jednoczesnego wysyłania tych informacji poprzez kanał szeregowy RS232 do zewnętrznego komputera lub drukarki. Po przekazaniu danych o wszystkich programach, na panelu pokazuje się napis, informujący o wielkości wolnego jeszcze obszaru pamięci masowej.
- zmiana nazwy programu w pamięci masowej - funkcja RENAME, procedura rename(), w pliku rename.a86.
- usunięcie programu z pamięci masowej - funkcja DELETE, procedura delete(), w pliku delete.a86.

4.4. Komunikacja z panelem programowania

Panel programowania komunikuje się z jednostką centralną za pośrednictwem kanału B łącza szeregowego RS232, obsługiwanego przez sterownik transmisji Z8530. Wyjście przerywające tego układu jest dołączone do wejścia kontrolera przerwań 8259. Za jego pośrednictwem sygnał z Z8530 jest doprowadzony do wejścia przerywajacego procesora. Przerwanie pochodzące od kontrolera Z8530 ma numer 33. W procedurze inicjalizacji wektora przerwań set_interrupt_vector() (w zb. set_vect.a86, w katalogu CTRLPGM/HARDWARE), jako adres obsługi tego przerwania jest wstawiony adres procedury int_z8530() (w zb. int_z853.a86). Sterownik Z8530 jest inicjowany w procedurze ini_8530(), zapisanej w zbiorze ini_z853.a86, w katalogu CTRLPGM/HARDWARE. Procedura ta jest wywoływana po wystartowaniu systemu, w podprogramie restart(). W ini_z8530 programowane są parametry transmisji w obu kanałach oraz sposób dzialania tych kanałów. W obecnej wersji programu sterującego kanał A (obsługujący komputer) pracuje przerwaniowo, a kanał B (obsługujący panel - bezprzerwaniowo). Parametry transmisji są zadeklarowane w zbiorze paramtr.a86 jako stałe i podtsawiane w procedurze inicjującej ini_z8530(). Dlatego, jeśli zajdzie potrzeba zmiany

parametrów, nie należy ingerować w program inicjalizacji, a tylko zmienić deklaracje parametrów. Obecnie są ustawione następujące wartości:

- kanał A - prędkość 9600, kontrola parzystości, długość słowa 8 bitów, dwa bity stopu,
- kanał B - prędkość 9600, kontrola parzystości, długość słowa 8 bitów, jeden bit stopu.

Po otrzymaniu przesyłki z zewnątrz do bufora odbiorczego kanału B, lub po wpisaniu przez inną procedurę programu sterującego przesyłki do wysłania do bufora nadawczego kanału B, układ Z8530 zgłasza przerwanie. Na podstawie informacji zawartej w słowie odczytanym z portu kontrolnego sterownika komunikacyjnego (adres Z8530_A_CONTROL) procedura int_z8530() rozpoznaje przyczynę przerwania i wywołuje podprogram odczytu przesyłki z panelu (rd_panel()) lub nadania przesyłki do panelu (wr_panel()). Obie te procedury są zapisane w plikach o takich samych nazwach w katalogu CTRLPGM/STDIO/PANEL. W tym samym katalogu zostały zgrupowane inne procedury obsługujące komunikację jednostki centralnej z panelem programowania, odpowiedzialne m.in. za wstępną interpretację przesyłek, zachowywanie obrazu panelu przed wejściem robota w pracę automatyczną (aby po zatrzymaniu robota na panelu pojawiły się te same napisy, co przedtem) itp.

4.5. Obsługa komunikacji z komputerem zewnętrznym przez kanał transmisji szeregowej

Jak już wspomniano drugi kanał transmisji szeregowej jednostki centralnej przewidziany jest do współpracy robota z komputerem zewnętrznym. Za pośrednictwem tego łącza można także przesyłać do komputera programy zapisane w pamięci masowej. Możliwe jest również przegrywanie programów z komputera do pamięci MV-62. Dzięki tym mechanizmom użytkownik może tworzyć archiwum robotowych programów użytkowych na komputerze osobistym. Tą drogą możliwe jest także przenoszenie programów między robotami. Do obsługi współpracy z komputerem zorganizowano w systemie komunikacji człowiek-maszyna robotów URP tzw. menu drugiego poziomu, nazwane KOMPUTER. Obejmuje ono trzy funkcje. Procedury realizujące te funkcje są zapisane w katalogu obsługi pamięci masowej CTRLPGM/STDIO/MASMEM:

- ustawienie parametrów transmisji łącza - funkcja PARAMETR - możliwa jest zmiana następujących parametrów transmisji szeregowej: prędkości przesyłania, parzystości, długości słowa i liczby bitów stopu. Umożliwia to współpracę układu sterowania robota praktycznie z dowolnym urządzeniem posiadającym kanał typu V-24. Funkcję realizuje procedura par_tran() w pliku par_tran.a86.
- przesłanie programu użytkowego do komputera zewnętrznego - funkcja SEND - można przesłać program będący aktualnie w pamięci RAM jednostki centralnej lub wskazany, dowolny program z pamięci masowej. Funkcję realizuje procedura send() w pliku send.a86.
- załadowanie programu użytkowego z komputera zewnętrznego - funkcja LOAD - można przesłać program z komputera zewnętrznego do pamięci RAM jednostki centralnej. Funkcję realizuje procedura load() w pliku load.a86.

Funkcje SEND i LOAD umożliwiają przenoszenie programów użytkowych między różnymi robotami. Aby program przesłany z komputera zachować w pamięci masowej należy go tam przegrać z pamięci RAM (po użyciu funkcji LOAD) funkcją ZAPIS.

Program sterujący robotów URP obejmuje także katalog CTRTLPGM/STDIO/KONSOLA. Zawiera on procedury pomocnicze obsługi komunikacji pomiędzy jednostką MV52 a

komputerem np. wstawienie nowej linii na ekranie, wyświetlenie liczby w konktretnym formacie itp.

5. Podsumowanie

Przdstawiony opis powinien ułatwić programistom poruszanie się po programie sterującym robotów URP w zakresie komunikacji z urządzeniami peryferyjnymi. Oczywiście do pełnego zrozumienia oprogramowania niezbędne jest zapoznanie się z jego postacią źródłową. Dlatego też do opracowania dołączono wydruk oprogramowania wg stanu na dzień 31.07.1994r. Jako załącznik natomiast dołączono specyfikację tego oprogramowania. Ten dokument bardzo pomaga przeszukiwać program, zapoznawać się z funkcjami poszczególnych procedur, bez konieczności sięgania od razu do pełnej postaci źródłowej. Jego przeglądanie jest dodatkowo ułatwione dzięki zastosowaniu edytora Word for Windows. Wydaje się, że w perspektywie dobrze byłoby wykonać specyfikację dla całego programu sterującego.

LITERATURA

1. Pilat Z., Dunaj J., Jacórzyńska M. Opracowanie opisu programu sterującego robotów URP w zakresie organizacji programu, wykorzystania zasobów sprzętowych sterownika, inicjalizacji systemu sterowania i pętli głównej programu. PIAP nr rej. 6978, Warszawa 1993.
2. Pilat Z., Jacórzyńska M. Opracowanie opisu programu sterującego robotów URP w zakresie języka programowania, interpretera, wykonywania instrukcji oraz modelu kinematyki i generatora trajektorii. PIAP nr rej. 7064, Warszawa 1994.
3. Słodczyk M., Syryczyński A. DTR pakietu jednostki centralnej MV52. PIAP nr rej. 6760, Warszawa 1991.
4. Jabłoński P., Pachuta M. DTR układu sterowania robotów URP-6 i URP-3. PIAP nr rej. 6843, Warszawa 1992.
5. Słodczyk M., Syryczyński A., Kisiel A. DTR pakietu pamięci masowej MV12. PIAP nr rej. 6761, Warszawa 1991.
6. Dunaj J., Pilat Z. Nowa koncepcja pamięci masowej programów użytkowych w robotach przemysłowych URP. IV Kraj. Konf. Robotyki, Wrocław 1993.
7. Dunaj J. Pakiet oprogramowania pamięci masowej zrealizowanej na bazie modułu MV-62. Biuletyn PIAP nr 5-163/92, Warszawa 1992.
8. Hernik W. Panel programowania - podręcznik użytkownika. PIAP nr rej. 6561, Warszawa 1991.
9. DTR panelu programowania. PIAP nr rej. 6527, Warszawa 1990.
10. DTR pakietu MV12. PIAP nr arch. 8135, Warszawa 1991.
11. Z8539 - Serial Communications Controller. SGS-Thomson Microelectronics, 1989.

DODATEK

SPECYFIKACJA OPROGRAMOWANIA REALIZUJĄCEGO KOMUNIKACJĘ UKŁADU STEROWANIA ROBOTA URP Z URZĄDZENIAMI PERYFERYJNYMI.

A. Katalog CTRLPGM/HARDWARE

A.1. Procedura AXIS_STOP()

Natychmiastowe zatrzymanie ruchu wszystkich osi. Jednostka centralne wysyła sygnał STOP do wszystkich sterowników położenia osi.

A.2. Procedura BLINK()

Miganie lampki na panelu operacyjnym.

A.3 .Procedura COUNTER2()

Procedura obsługi przerwania zegarowego generowanego przez timer procesora 80186 (przerwanie nr 19):

1) zegar systemowy (20 ms),

2) wskaźnik aktywności ręcznego sterowania ruchem

3) miganie lampek na panelu operacyjnym.

A.4. Procedura csra()

Inicjowanie rejestru stronicowania adresów do obsługi pamięci masowej MV62..

A.5. Procedura CTRLRDY()

Procedura ta powinna być wykonana na samym początku. Zakłada się, że przerwania są zablokowane. Realizuje ona:

- wyzerowanie wszystkich wyjść użytkownika
- wysłanie sygnału CONTROLREADY (w celu wyłączenia stopu awaryjnego - odblokowanie zasilania stopni końcowych mocy) ustawienie zmiennej globalnej lamps = 0001$0000B (co oznacza, ze jest ustawiony sygnał CONTROLREADY)

A.6. Procedura disable_interrupts()

Zablokowanie przerwań maskowalnych.

A.7. Proceduraenable_interrupt()

Odblokowanie przerwań maskowalnych.

A.8. Procedura extingsh()

Zgaszenie lampki na panelu operacyjnym.

A.9. Procedura getclock()

Procedura podaje stan zegara systemowego.

A.10. Procedura grippers()

Sterowanie chwytakami.

A.11. Plik **hardcons.c86**

Deklaracje stałych globalnych procedur obsługujących bezpośrednio hardware:

-Adresy portów sterujących sterowników osi na pakietach MV20:

-Adresy portów danych sterowników osi na pakietach MV20:

-Adresy portów sterujących sterowników osi na pakietach MV20:

-Adresy generacji hardware'owego resetu procesora sterowników osi MA20:

-Adres systemowego pakietu WE/WY dwustanowych MV12:

-Adresy pakietów MV12 dwustanowych użytkownika - WEJŚCIA:

-Adresy pakietów MV12 dwustanowych użytkownika - WYJŚCIA:

A.12. Plik hardvar.c86

Zmienne i stale globalne procedur obsługujących bezpośrednio hardware.

A.13. Nagłówek hardware.h

Definicja wielkości związanymi z hardwarem:

- sygnały wyjściowe systemowego bloku MV12:
- sygnały wejściowe systemowego bloku MV12:
- maski dla poszczególnych bitów
- słowa czytanego ze sterownika osi MV20:
- maski dla poszczególnych bitów
- słowa wysyłanego do sterownika osi MV20:

A.14. Plik hard_con.a86

Plik zawiera definicje nazw określających adresy I/O wewnętrznych układów mikroproce-sora 80186 oraz adresy pozostałych układów WE/WY na pakiecie sterownikaMV52. Wszystkie nazwy są typu PUBLIC i zostały zdefiniowane przy pomocy dyrektyw EQU asemblera ASM186. Są one dostępne wyłącznie dla procedur napisanych w języku asemblera, gdzie występują jako nazwy (po zadeklarowaniu jako 'EXTRN nazwa_stałej: ABS").

Deklaracje pól pamięci zawierających adresy I/O wewnętrznych układów mikroprocesora 80186 oraz adresy pozostałych układów WE/WY na pakiecie sterownika MW-52. Pola pamięci zostały zadeklarowane przy pomocy dyrektyw DW asemblera ASM186 i nosza one te same nazwy z dopiskiem "_" na końcu co odpowiadające im definicje opisane w p.A. Pola te są dostępne z poziomu języka C lub PLM-86 (mogą być także dostępne z poziomu asemblera, ale w odwołaniach do nich musi wystąpić znak "_".

A.15. Procedura inbyte()

Odczytanie wartości 1-bajtowej z portu o danym adresie.

A.16. Procedura inithard()

Inicjalizacja hardware'u i procedur bezpośrednio obsługujących hardware

poza układami i zmiennymi bezpośrednio związanymi z funkcjami formatowanego

I/O pakietu iC-86

A.17. Procedura ini_186()

Procedura inicjująca wewnętrzny Timer i wewnętrzny sterownik przerwań mikroprocesora 80186. Zakłada się, że mikroprocesor będzie pracował w trybie non-iRMX86.

A.18. Procedura ini_8259a()

Inicjowanie obu zewnetrznych sterownikow przerwan PIC 8259A na pakiecie MV52.

A.19. Procedura ini_z853()

Inicjowanie dwukanałowego układu transmisji szeregowej Z8530A na pakiecieMV52.

A.20. Procedura ini_kanal_a()

Procedura "ini_kanal_a" ustawia parametry transmisji kanału A układu Z8530, obsługującego transmisje z urządzeniem zewnętrznym (komputer, drukarka etc). Zadane parametry transmisji są przekazywane poprzez stos w kolejności jak poniżej.

A.21. Procedura reset_dsr()

Procedura "RESET_DTR" ustawia w stan "0" logicznej linie sterującą DTR kanału A układu transmisji szeregowej Z8530, obsługującego transmisje do/z urządzenia zewnętrznego (komputer, drukarka).

A.22. Procedura set_dsr()

Procedura "SET_DTR" ustawia w stan "1" logicznej linie sterującą DTR kanału A układu transmisji szeregowej Z8530, obsługującego transmisje do/z urzadzenia zewnętrznego (komputer, drukarka).

A.23. Procedura inpos()

Sprawdzenie czy jest stan "INPOS"

A.24. Procedura int_time();

Obsługa przerwania zgłaszanego przez licznik nr 2 wewnętrznego timera mikroprocesora 80186. Przerwania są zgłaszane co 20 milisekund.

A.25. Procedura int_z8530();

Obsługa przerwania zgłaszanego przez kanał B układu transmisji szeregowej Z8530 pakietu sterownika MV-52.

A.26. Procedura inword();

Odczytanie wartości 1-bajtowej z portu o danym adresie.

A.27. Procedura light()

Zapalenie lampki na panelu operacyjnym

A.28. Procedura opoznienie();

Zrealizowanie programowego opóźnienia polegającego na wykonywaniu stałej ilości razy pustej pętli programowej.

A.29. Procedura outbyte();

Wysłanie wartości 1-bajtowej do portu o danym adresie.

A.30. Procedura outword();

Wysłanie wartości 2-bajtowej do portu o danym adresie.

A.31. Plik paramtr.a86

Plik zawiera definicje stałych, wykorzystywanych w czasie inicjowania układu transmisji szeregowej Z8530 do ustawiania parametrów transmisji obu kanałów A i B.

A.32. Procedura read_actual_position();

Odczytanie ze sterowników rzeczywistego położenia.

A.33. Procedura sendinc()

Procedura send_increments() wysyła inkrementy do sterowników osi. Sprawdzana jest gotowość sterowników osi i jeśli sterowniki nie są gotowe to procedura czeka aż będą gotowe.

A.34. Procedura set_interrupt_vector();

Plik zawiera procedurę, wypełniająca cały obszar wektora przerwań adresami do podprogramu, który wyświetla na panelu programowania numer przerwania. Po ustawieniu całego wektora zostaje ustawiony adres dla przerwania nr 19 (obsługuje licznik 2 wewnętrznego timera mikroprocesora 80186), i do przerwania nr 33 (obsługuje zgłaszenia kanału B układu transmisji szeregowej Z8530).

A.35. Procedura set_unexpected_interrupt_vectors();

Ustawienie wektora przerwań do procedur obsługi przerwań od numeru 0 do numeru 255.

A.36. Procedura send_correction_increments();

Wystartowanie synchronizacji hardware'owej

A.39. Procedura synchrnd()

Badanie zsynchronizowania sterowników osi.

A.40. Procedura unexpected_interrupt();

Obsługa nieoczekiwanych przerwań od numeru 0 do numeru 255.

A.41. Procedura userin();

Odczytanie wejścia użytkownika

A.42. Procedura userout()

Wysłanie wartości na wyjście użytkownika.

B. Katalog CTRLPGM/UTILITY

B.1. Procedura add_to_memory_address()

Zwiększenie adresu o zadana ilość razy.

B.2. Procedura address_on_console()

Wyświetlenie na ekranie konsoli adresu hexadecymalnego, przekazanego jako parametr wywołania procedury.

B.3. Procedura char_to_int()

Wpisanie 2-bajtowej wartości od komórki pamięci, której adres określa parametr wywołania procedury do rejestru AX. Celem tego podprogramu jest umożliwienie "złożenia" wartości typu (unsigned) int z dwóch wartości typu (unsigned) char. Zawartość komórki o młodszym adresie (przekazanym jako parametr wywołania procedury) jest wstawiana do rejestru AL, zawartość komórki następnej (w kierunku rosnących adresów) - do rejestru AH.

B.4. Procedura dump()

Wyświetlenie na ekranie konsoli zawartości obszaru pamięci od komórki, której adres określa pierwszy parametr. Drugi parametr określa liczbę komórek pamięci, których zawartość będzie wyświetlona.

B.5. Procedura increment_memory_address()

Zwiększenie adresu pamięci o jeden.

B.6. Procedura int_to_char()

Wpisanie 2-bajtowej wartości od komórki pamięci, której adres określa parametr wywołania procedury. Celem tego podprogramu jest możliwość wypełnienia wartościami typu (unsigned) int tablic o elementach zadeklarowanych jako (unsigned) char. Młodszy bajt 2-bajtowej wartości jest wstawiany do komórki pamięci o podanym adresie, starszy - do komórki następnej (w kierunku rosnących adresów), czyli do elementu tablicy o kolejnym indeksie.

B.7. Procedura lampki()

Procedura "lampki()" powoduje sekwencyjne zapalanie i gaszenie dwóch zielonych lampek na czołowej listwie pakietu MV-52. . Procedura ta nie jest wykorzystywana w programie sterującym robota, ale może być wykorzystywana podczas jego testowania.

B.8. Procedura monitor_reset()

Skok do początku monitora z zablokowaniem przerwań maskowalnych.

B.9. Procedura mov_byte()

Przepisanie bajt po bajcie zawartości obszaru pamięci o długości "count" bajtów, rozpoczynającego się od adresu "source" do obszaru pamięci rozpoczynającego się od adresu "destination". Przepisywanie odbywa się w kierunku rosnących adresów.

B.10. Procedura parametr()

Wyświetlenie od nowej linii na ekranie monitora konsoli następującego komunikatu:

PARAMETR = xxxx

gdzie xxxx jest hexadecymalna wartością parametru wywołania procedury.

B.11. Procedura pause()

Procedura PAUSE wykonuje określoną ilość razy pustą pętlę programową. Czas wykonania jednego obiegi pętli wynosi 804 cykle, co po przemnożeniu przez okres zegara procesora (125 nanosekund) odpowiada w przybliżeniu okresowi około 100 mikrosekund (0.1 milisekundy).

B.12. Procedura pointer_to_text()

Procedura "pointer_to_text()" wyznacza wskaźnik (long pointer) to początku tekstu o numerze określonym parametrem wywołania tej procedury. Tekst jest zawarty w jednym z elementów tablicy znakowej, przy czym separatorami pomiędzy kolejnymi tekstami są wartości 00H:

B.13. Procedura rejestry()

Wyświetlenie na ekranie konsoli zawartości wszystkich rejestrów fizycznych z miejsca, w którym wywoływana jest procedura REJESTRY.

C. Katalog CTRLPGM\STDIO\KONSOLA

C.1. Procedura blank();

Wyświetlenie na ekranie konsoli pojedynczego znaku spacji.

C.2. Procedura byte_on_console()

Wyświetlenie na ekranie konsoli 2-cyfrowej liczby hexadecymalnej, odpowiadajacej danej wartości 1-bajtowej, przekazanej jako parametr wywołania procedury.

C.3 Plik constans.a86

Definicje nazw używanych w procedurach asemblerowych. Wszystkie nazwy są typu PUBLIC i zostały zdefiniowane przy pomocy dyrektyw EQU asemblera ASM186. Są one dostępne wyłącznie dla procedur napisanychw języku asemblera, gdzie występują jako nazwy po zadeklarowaniu jako:

EXTRN nazwa_stałej: ABS

C.4. Procedura decimal_double_word_on_console()

Wyświetlenie na ekranie konsoli liczby dziesiętnej, odpowiadającej danej

C.5. Procedura decimal_byte_on_console()

Wyświetlenie na ekranie konsoli liczby dziesiętnej, odpowiadającej danej wartości 1-bajtowej, przekazanej jako parametr wywołania procedury.

C.6. Procedura decimal_word_on_console()

Wyświetlenie na ekranie konsoli liczby dziesiętnej, odpowiadającej danej wartości 2-bajtowej, przekazanej jako parametr wywołania procedury.

C.7. Procedura getchar_external()

Odebranie znaku z urządzenia zewnętrznego. Jeśli bufor układu Z8530 obsługującego urządzenie zewnętrzne jest pusty, to do programu wywołującego w rejestrze AX zwracana jest wartość EOF, jeśli odebrano znak to rejestr AL zawiera ten znak a w rejestrze AH jest wartość 00H.

C.8. Procedura getchar_external_with_wait()

Odebranie znaku z urządzenia zewnętrznego, ale program pracuje w pętli aż do nadejścia znaku.

C.9. Procedura message()

Wysłanie do kanału obsługującego konsolę operatorską tekstu znakowego, do którego long pointer jest parametrem wywołania procedury. Tekst musi kończyć się znakiem NULL (00H).

C.10. Procedura new_line()

Wysłanie znaku nowej linii i znaku przesunięcia karetki do kanału obsługującego konsolę operatorska.

C.11. Procedura putchar_external()

Wysłanie znaku na urządzenie zewnętrzne (konsole operatorska).

C.12. Procedura word_on_console()

Wyświetlenie na ekranie konsoli 4-cyfrowej liczby hexadecymalnej, odpowiadającej danej wartości 2-bajtowej, przekazanej jako parametr wywołania procedury.

D. Katalog CTRLPGM\STDIO\RETARGER

D.1. Procedura _conio_create() - w pliku conio.c86

Funkcja przydziela n-bajtów pamięci z "puli" pamięci zarezerwowanej na potrzeby zadania, które wywołało funkcję _conio_create() i zwraca wskaźnik do początku przydzielonego obszaru. Inna funkcja, a mianowicie _conio_ptr() wywoływana z poziomu tego samego zadania powinna zwracać ten sam wskaźnik. Jeśli standardowe strumienie są identyczne dla kilku zadań program nadrzędny powinien wywoływać funkcję _conio_create() dokładnie jeden raz.

D.2. Procedura _conio_ptr() - w pliku conio.c86

Funkcja _conio_ptr() zwraca wskaźnik do struktury danych określających strumienie na konsolĘ wywołującego te funkcje zadania.

D.3. Procedura _exit_create

Funkcja _exit_create() przydziela nbyte bajtów.pamięci z "puli" pamięci zarezerwowanej na potrzeby zadania, które wywołało funkcje _exit_create() i zwraca wskaźnik do początku przydzielonego obszaru. Inna funkcja, a mianowicie _exit_ptr(), wywoływana z poziomu tego samego zadania powinna zwracać ten sam wskaźnik. Jeśli funkcja exit() przerywa wykonywanie wszystkich zadań korzystających z danych kontekstu to funkcja _exit_create() może być wywoły wana dokładnie jeden raz. Funkcja _exit_create() jest wykorzystywana przez funkcje _exit_init(), zdefiniowana w bibliotekach CLIBx.LIB (x=C,L,M,L).

D.4. Procedura _exit_ptr

Funkcja zwraca wskaźnik do listy otwartych strumieni i handlerów wyjściowych zadania, z którego została wywołana. Jeśli funkcja exit() przerywa wykonywanie wszystkich zadan korzystających z danych kontekstu to funkcja _exit_ptr() nie potrzebuje zwracać wskaźnika związanego tylko z tym zadaniem. Funkcja _exit_ptr() jest wykorzystywana przez następujące funkcje (wszystkie zdefiniowane w bibliotekach CLIBx.LIB (x=C,L,M,L):

- exit()
- atexit()
- fopen()
- close()
- stdio_init()

D.5. Procedura isatty();

Użycie procedury isatty() spowoduje sprawdzenie czy deskryptor pliku jest skojarzony z takim rodzajem urządzenia jak port szeregowy, drukarka, terminal.

D.6. Procedura lseek()

Use LSEEK to alter the read/write position pointer for a disk file. This pointer defines the character to be read or written on the next I/O operation. The file must be open to be read or read/write mode. Allowable base are:

- SEEK_SET - seek relative to the beginning of the file. The offset must be positive,
- SEEK_CUR - seek relative to the current position in the file,
- SEEK_END - seek relative to the end of the file.

The call "lseek (fd, 0L, SEEK_SET)" causes the next I/O operation to process the first byte in the file. The call "lseek (fd, -1L, SEEK_CUR)" causes the next I/O operation to process again the most recently processed byte. Avoid seek operations beyond the end of the file, since such operations can result in files with missing sectors and can cause incorrect end_of_file indications in subsequent processing. Although you can perform LSEEK on an ASCII file, the presence of carriage return and line-feed (CR-LF) pair makes determining the seek offset difficult.

D.7. Procedura malloc()

The function MALLOC allocates a region of memory large enough to hold an object whose size (as measured by sizeof operator) is size. A pointer to the first element of the region is returned. If it is impossible for some reason to perform the requested allocation, a null pointer is returned. If the requested size is 0, either a null pointer or an implementation-defined unique pointer is returned, at the implementation's discretion. The region of memory is not specially initialized in any way, and the caller must assume that it will contain garbage information

D.8. Procedura _map_length()

Funkcja _MAP_LENGTH - wzajemne dopasowanie strumienia do pliku. Funkcja _MAP_LENGTH dopasowuje znaki miedzy strumieniami i plikami. Przy pomocy funkcji ftell() można otrzymać wskaźnik do aktualnie czytanego lub zapisywanego znaku w danym pliku. Funkcja _MAP_LENGTH dopasowuje aktualną pozycję w danym pliku do liczby danych znajdujących się w buforze. Wynika to z następującego powodu: Znaki w strumieniu nie muszą dokładnie odpowiadać znakom w pliku, ponieważ są różne opcje przechowywania niektórych znaków takich jak new-line w pliku. Funkcja _MAP_LENGTH jest używaną przez funkcje ftell().

D.9. Procedura semaphore_init()

Funkcja _SEMAPHORE_INIT tworzy i inicjuje semafor (jest to rodzaj flagi) związany z obszarem pamięci, którego początek określa parametr wywołania tej funkcji. Pozwala to sterować wyłącznością dostępu do określonego obszaru pamięci przez inne funkcje. Dany program musi wywołać funkcję _semaphore_ _init() przed użyciem jakiejkolwiek funkcji związanej z czasowa wyłącznością dostępu do określonych danych. Funkcja _semaphore_init() jest wykorzystywana przez następujące funkcje (wszystkie zdefiniowane w bibliotekach CLIBx.LIB (x=C,L,M,L):

- fopen(),
- _exit_init(),
- stdio_init().

D.10. Procedura semaphore_signal()

Funkcja _SEMAPHORE_SIGNAL zwalnia (gasi ?) semafor (jest to rodzaj flagi) związany z obszarem pamięci, którego początek określa parametr wywołania tej funkcji. Pozwala to na wykorzystanie tego obszaru pamięci przez inne zadania.

D.11. Procedura semaphore_wait()

Funkcja _SEMAPHORE_WAIT sprawdza, czy obszar pamięci, którego początek określa parametr wywołania tej funkcji jest wolny (sprawdza semafor regulujący wyłączność dostępu do tego obszaru pamięci). Jeśli obszar ten jest wykorzys- tywany przez inne zadanie, to funkcja _SEMAPHORE_WAIT zawiesza wykonywanie zadania z którego ja wywołano do momentu, aż pamięć będzie zwolniona przez funkcje _SEMAPHORE_SIGNAL uruchomiona z zadania aktualnie posiadającego wyłączność dostępu do "spornego" obszaru. Jeśli więcej niż jedno zadanie wywołuje funkcje _SEMAPHORE_WAIT z tym samym parametrem w czasie, gdy dany obszar pamięci jest zajęty, to dodatkowym zadaniem użytkownika jest wykonanie mechanizmu regulującego priorytet dostępu do tej pamięci przez poszcze- gólne zadania.

D.12. Procedura _stdio_create

Funkcja _stdio_create()przydziela n-bajtów pamięci z "puli" pamięci zarezerwowanej na potrzeby zadania, które wywołało funkcje _stdio_create() i zwraca wskaźnik do początku przydzielonego obszaru. Inna funkcja, a miano- wicie _stdio_ptr() wywoływana z poziomu tego samego zadania powinna zwracać ten sam wskaźnik. Jeśli standardowe strumienie są identyczne dla kilku zadan program nadrzędny powinien wywoływać funkcje _stdio_create() dokładnie jeden raz. Funkcja _stdio_create() jest wykorzystywana przez funkcje _stdio_init(),zdefiniowana w bibliotekach CLIBx.LIB (x=C,L,M,L).

D.13. Procedura _stdio_ptr

Funkcja _stdio_ptr() zwraca wskaźnik do struktury danych określających standardowe strumienie wywołującego te funkcje zadania. Jeśli więcej niż jedno zadanie wykorzystuje te same standardowe strumienie, to funkcja _stdio_ptr() nie musi zwracać określonego tylko dla danego zadania wskaźnika. Funkcja _stdio_ptr() jest wykorzystywana przez cały szereg funkcji zadeklarowanych w pliku <stdio.h>.

D.14. Procedura _stdio_stdopen()

Funkcja _STDIO_STDOPEN otwiera standardowe strumienie. Funkcja ta otwiera wybrany strumień i zwraca związany z nim deskryptor pliku. Argument wywołania tej funkcji określa w następujący sposób strumień do otworzenia:

Wartość parametru	Otwierany strumień
0	stdin
1	stdout
2	stderr

Funkcja _stdio_stdopen() jest wywoływana przez funkcje _stdio_init() zdefiniowana w bibliotekach CLIBx.LIB (x=C,L,M,L).

D.15. Procedura thread_create();

Funkcja _thread_create() przydziela nbyte bajtów pamięci z "puli" pamięci zarezerwowanej na potrzeby zadania, które wywołało funkcje _thread_create i zwraca wskaźnik do początku przydzielonego obszaru. Inna funkcja, a mianowi cie _thread_ptr() wywoływana z poziomu tego samego zadania powinna zwracać ten sam wskaźnik. Funkcja _thread_create() jest wykorzystywana przez funkcje _thread_init(), zdefiniowana w bibliotekach CLIBx.LIB (x=C,L,M,L).

D.16. Procedura _thread_ptr()

Funkcja _thread_ptr() zwraca wskaźnik do struktury danych wywołującego te funkcje zadania.

E. Katalog CTRLPGM\STDIO

E.1. Nagłówek buffers.h

Plik zawiera deklaracje wielkości buforów I/O i definicje struktury sterującej wypełnieniem tych buforów. Wielkość bufora wejściowego przechowywującego informacje odbierane z panelu programowania robota jest obecnie ustawiona na 256, a wielkość bufora wyjściowego przechowywującego informacje wysyłane do panelu programowania robota jest - na 512 (#definePNLINS256, #definePNLOUS512).

E.2. Nagłówek charcode.h

Nagłówek zawiera kody znaków sterujących wykorzystywanych przy obsłudze strumieni we/wy w programie sterującym robota.

E.3. Procedura ioinit();

Inicjacja pól danych związanych z obsługa strumieni we/wy.

E.4. Procedura iovariab.c86

Deklaracje pól roboczych dla funkcji we/wy.

E.5. Procedura read();

Odczytanie co najwyżej "count" znaków z pliku opisanego deskryptorem "file_descriptor" i umieszczenie ich w pamięci od adresu opisanego parametrem "buffer". Znaki są odczytywane do momentu pojawienia się EOF, albo odczytania "count" znaków. Do procedury tej w programie sterującym robota URP nie występują jawne odwołania, lecz jest ona wykorzystywana przez podprogramy biblioteczne I/O (printf() etc.) dostarczone z pakietem kompilatora iC-86.

E.6. Nagłówek serial.h

Nagłówek zawiera definicje stałych wykorzystywanych do obsługi układu transmisji szeregowej Z8530.

E.7. Nagłówek stdio.h

Ten nagłówek zawiera definicje typów, struktur i makrosów dla standardowej biblioteki wejścia/wyjścia dla programu sterującego robota z panelem RP66. Do obsługi standardowego wejścia/wyjścia wykorzystywane są procedury 'printf' i 'fprintf' z biblioteki 'lclib' dla MWC86.

E.8. Procedura stream()

Plik ten zawiera procedurę streams() składającą się z następujących części:

- części pierwszej, zawierającej wywołania trzech procedur:
  - malloc_init() - jest to procedura inicjująca tablice wykorzystywane przez samodzielnie napisaną funkcję malloc()
  - _thread_init() - jest to procedura biblioteczna, której wywołania żąda instrukcja źródłowa Intela przed rozpoczęciem korzystania z innych podprogramów bibliotecznych związanych z obsługą WE/WY,

  - _exit_init() - jest to procedura biblioteczna, której wywołania żąda instrukcja źródłową Intela przed rozpoczęciem korzystania z innych podprogramów bibliotecznych związanych z obsługą WE/WY,

- części drugiej, która inicjuje strukturę związaną z obsługa strumieni stdio (panel programowania). W tym miejscu wykonywane są czynności, które wykonuje program biblioteczny, a mianowicie _stdio_init(). Powodem samodzielnego napisania tej części (zamiast wykorzystania gotowej procedury bibliotecznej) była potrzeba zapanowania nad inicjacją w/w struktury, a sam kod został napisany na podstawie zdeasemblowanej postaci funkcji bibliotecznej. W stosunku do pierwowzoru (tj. funkcji bibliotecznej _stdio_init()) poczyniono pewne uproszczenia, ale w komentarzach za niektórymi instrukcjami (zaznaczonych dodatkowo znakiem "#") przedstawiono w jaki sposób dana czynność była wykonywana przez funkcje _stdio_init(). Podprogramy wywoływane w tej zakomentowanej części są napisane i uruchomione, ale nie są dołączane przez konsolidator LINK86.EXE.
- części trzeciej, inicjującej strukturę związaną z obsługa strumieni na konsole operatorska, wykorzystywana do wydruków kontrolnych. Struktura ta jest inicjowana identycznie jak struktura obsługi strumieni stdio (a wiec po prostu przekopiowano zawartość jednej struktury na druga) z jedynym wyjątkiem dotyczącym deskryptorów (nazw) obsługiwanych strumieni.

E.9. Procedura unlock();

Procedura wyczyszczenia bufora fp->_ungetc dla wskazanego strumienia.. Procedura powinna być użyta wtedy, gdy chcemy mieć pewność, że nieobsłużony przez nikogo znak zostanie pominięty, a nie będzie stale siedział w buforze blokując tym samym odbiór następnych znaków (jako niewłaściwy byłby stale zwracany przez procedurę 'ungetc').

E.10. Procedura write();

Przepisanie "count" znaków z pamięci od adresu opisanego parametrem buffer do pliku opisanego deskryptorem "file_descriptor". Do procedury tej w programie sterującym robota URP nie występują jawne odwołania,lecz jest ona wyko-rzystywana przez podprogramy biblioteczne we/wy dostarczone z pakietem kompilatoraiC86.

F. Katalog CTRLPGM\STDIO\MASMEM

F.1. Procedura brak_pliku()

Procedura brak_pliku()wyświetla na panelu programowania komunikat o niewykryciu w tablicy NAME_TABLE[] pliku o nazwie zapisanej w tablicy pomocniczej NAZWA_ZBIORU[] i oczekuje na wciśnięcie jednego z klawiszy funkcyjnychPF4 lub PF5. Kod tego klawisza jest zwracany do programu nadrzędnego w rejestrze AX.

F.2. Procedura calculate_sector_address()

Procedura oblicza adres fizyczny początku danego sektora pamięci masowej EEPROM pakietu MV62 na podstawie numeru tego sektora.

F.3. Procedura del_prog()

Procedura realizuje funkcję "DELETE".

F.4. Plika dir.c86

Plik zawiera komplet procedur realizujących funkcje "DIR" (poza podprogramami ogólnymi, zadeklarowanymi jako EXTRN). Procedurą główną, realizującą funkcję "DIR" jest podprogram dir().

F.5. Procedura informacja()

Wyświetlenie na panelu programowania lub wydrukowanie na drukarce linii zawierającej dane danego programu zapisanego w pamięci masowej EEPROM.

F.6. Procedura format()

Procedura format() realizuje funkcje "FORMAT".

F.7. Procedura interp_klawisz()

Panel programowania posiada na swojej klawiaturze szereg klawiszy oznaczonych mnemonikami. Wciśnięcie tych klawiszy powoduje wysłanie z panelu programowania kodu klawisza, który niestety nie jest kodemASCII mnemonika opisującego ten klawisz.. Procedura interpr_klawisz() bada stan klawiatury i wysyła do programu nadrzędnego kod ASCII mnemonika opisującego wciśnięty klawisz. Dotyczy to jednak tylko następujących klawiszy:

F.8. Procedura load()

Plik zawiera komplet procedur realizujących funkcje "LOAD" (poza podprogramami ogólnymi, zadeklarowanymi jako EXTRN). . Procedura główna, realizująca funkcje "LOAD" jest podprogram load().

F.9. Procedura interp_byte()

Procedura czyta dwa kolejne znaki odbierane z zewnętrznego urządzenia I/O, a następnie składa z nich jednobajtowa wartość hexadecymalna. Wartość ta jest zwracana do procedury nadrzędnej w rejestrze AL. Procedura jest wywoływana tylko przez podprogramy zawarte w tym pliku, stad nie przepisano jej do osobnego zbioru.

F.10. Procedura interp_word()

Procedura czyta cztery kolejne znaki odbierane z zewnętrznego urządzenia I/O, a następnie składa z nich dwubajtowa wartość hexadecymalna. Wartość ta jest zwracana do procedury nadrzędnej w rejestrze AX. Procedura jest wywoływana tylko przez podprogramy zawarte w tym pliku, stad nie przepisano jej do osobnego zbioru.

F.11. Procedura interp_number()

Procedura interpretuje cyfrę hexadecymalna przekazywana jako parametr i zwraca w rejestrze AX jej wartość liczbowa. W przypadku, gdy parametr nie jest cyfra hexadecymalna, to zwracany jest jego kod ASCII czyli parametr wywołania).

F.12. Procedura suma_kontrolna()

Odczytuje bajt sumy kontrolnej z kanału transmisyjnego, porównuje go z obliczona suma kontrolna i jeśli wszystko jest O.K. następuje powrót do procedury wywołującej. W przypadkubłęduwyświetlony zostaje stosowny komunikat i po wciśnięciu na panelu programowania klawisza PF5 następuje wyskok z procedury LOAD.

F.13. Procedura getchar_with_wait_for_load()

Ustawia linie sterującą DTR kanału A układu Z8530 w stan "1" (zezwolenie dla urządzenia zewnętrznego na transmisję) i oczekuje na nadejście znaku badając jednocześnie stan klawiatury panelu programowania robota. Odebrany znak jest odczytywany, a następnie linia sterującą DTR zostaje ustawiona w stan "0" (blokada transmisji dla urzadzenia zewnętrznego).

F.14.Procedura record_information()

Wyświetlenie na panelu programowania numeru rekordu, zawartego w rejestrze SI.

F.15.Procedura calculate_address();

Procedura CALCULATE_ADDRESS na podstawie paragrafu i offsetu adresu wyznacza jego wartość fizyczna i zwraca:

- w rejestrze DX starsze słowo adresu fizycznego,
- w rejestrze AX młodsze słowo adresu fizycznego.

F.16.Procedura exit_load()

Procedura ta pozwala na wyskok z programu realizującego funkcje LOAD (lub z wewnątrz wywoływanej przez ten program procedur), korzystając z zawartości rejestru BP.

F.17.Procedura nazwa()

Procedura ta pozwala na wczytanie z panelu programowania do tablicy pomocniczej NAZWA_ZBIORU ciągu znaków, które następnie będą interpretowane przez procedurę nadrzędną. Procedura działa w ten sposób, że najmniej znaczący znak nazwy jest zapisany zawsze w elemencie NAZWA_ZBIORU [0] (który ma także najniższy adres). Jeśli nazwa zbioru składa się z mniejszej ilości znaków niż wynosi rozmiartablicy NAZWA_ZBIORU [], to doniewykorzystanych elementów tej tablicy będą wpisane znaki 00H.

F.18. Procedura par_tran()

Procedura par_tran() realizuje funkcje "PARAMETR" - programowe ustawianie parametrów transmisji kanału A układu Z8530 na pakiecie MV-52, które służy do wymiany informacji z urządzeniem zewnętrznym.

F.19. Procedura pojemnosc()

Wypisanie na panelu programowania robota danych dotyczących wolnej przestrzeni w pamięci masowej.

F.20. Procedura protection()

Wyświetlenie na panelu programowania robota komunikatu o ustawionej protekcji zapisu pamięci EEPROM na pakiecie MV62 i oczekiwanie na wciśnięcie jednego z klawiszyfunkcyjnych PF4 lub PF5. Kod tego klawisza jest zwracany do programu nadrzędnego w rejestrze AL, rejestr AH jest zerowany.

F.21. Procedura read_mas()

Procedura read_mas() realizuje funkcje "ODCZYT".

F.22. Procedury: read_byte_from_eeprom() read_word_from_eeprom()

Procedura "read_byte_from_eeprom()" ("read_word_from_eeprom()") odczytuje wartość 1-bajtowa (1 słowo) z pamięci EEPROM pakietu MV62. Parametrami wejściowymi są:

- paragraf adresu,
- offset adresu.

Funkcja zwraca w rejestrze AX (poprzez swoja nazwę) odczytana wartość.

F.23. Procedura readmem();

Procedura READMEM przepisuje zawartość pamięci masowej (pamięć EEPROM od adresu 4000:0000H do tablicy glob_space[glob_space_length], a także obsługuje błąd spowodowany ewentualnym błędnym zapisem.

F.24. Procedura rename()

Procedura rename() realizuje funkcje RENAME.

F.25. Plik send.c86

Plik zawiera komplet procedur realizujących funkcje "SEND" (poza podprogramami ogólnymi, zadeklarowanymi jako EXTRN). Procedurą główną, realizująca funkcje "SEND" jest podprogram send().

F.26. Procedura code_byte_hex()

Przekodowanie liczby przekazywanej jako parametr wywołania podprogramu CODE_BYTE_HEX na dwie cyfry hexadecymalne, zwracane do procedury wywołującej wrejestrach AH i AL.

F.27. Procedura search_number_file()

Nazwy wszystkich zbiorów (programów robotowych) zapisanych w pamięci masowej EEPROM na pakiecie MV62 znajdują się w tablicy NAME_TABLE[] (pamięć EEPROM), natomiast nazwa aktualnego programu potrzebnego do jakiś celów (kasowanie, zapis do obszaru glob_space etc) w tablicy pomocniczej NAZWA_ZBIORU[] (pamięć RAM). Procedura SEARCH_NUMBER_FILE przeszukuje tablicę NAME_TABLE[] w poszukiwaniu jej elementu (ciągu znaków), który jest identyczny z tym wpisanym do NAME_TABLE[] i zwraca do podprogramu nadrzędnego (w rejestrze AX) numer tego elementu (jest to liczba z przedziału [0,255]. Jeśli poszukiwania nie zakończyły się sukcesem to zwracana jest wartość FFFFH.

F.28. Plik: stale.a86

Plik zawiera definicje nazw używanych w procedurach obsługi pamięci masowej napisanych w języku asemblera ASM-86. Wszystkie nazwy są typuPUBLIC i zostały zdefiniowane pomocy dyrektyw EQU asemblera ASM186. Są one dostępne wylacznie dla procedur napisanychw języku asemblera.

F.29. Plik: tablice.a86

Plik zawiera następujące definicje i deklaracje:

- LICZBA_SEKTOROW - określająca liczbę 256-bajtowych sektorów pamięci masowej,
- SIZE_NAZWA_ZBIORU - rozmiar tablicy NAZWA_ZBIORU, czyli maksymalna liczba znaków, z których może składać się nazwa zbioru,
- MASS_MEMORY - etykieta, od adresu której rozpoczyna się ta część pamięci masowej, która jest przeznaczona bezpośrednio do zapisywania informacji użytkowej,
- NAME_TABLE - tablica zawierająca nazwy plików zapisanych w pamięci masowej,
- NAZWA_ZBIORU- tablica zawierająca nazwę pliku, który ma zostać poddany dalszej "obróbce" (kasowanie, czytanie zapisanie etc),

- SEKTOR_TABLE- tablica, której dany element opisuje pojedynczy sektor pamięci masowej (czy jest on zajęty, wolny, uszkodzony, ktorego pliku informacje zawiera etc).

F.30. Procedura write_mas()

Procedura write_mas() realizuje funkcje "ZAPIS".

F.31. Procedury: write_byte_to_eeprom(), write_word_to_eeprom()

Procedura WRITE_BYTE_TO_EEPROM (WRITE_WORD_TO_EEPROM) wpisuje wartość 1-bajtową (1 słowo) do pamięci EEPROM pod wskazany adres. Parametrami wejściowymi są:

- - bajt (słowo),
- - paragraf adresu,
- - offset adresu.

F.32. Procedura writemem()

Procedura przepisuje zawartość programu użytkowego, czyli tablicy o nazwie glob_space[glob_space_length] do pamięci masowej (pamięć EEPROM na jednostce centralnej od adresu 4000:0000H), a także obsługuje błąd spowodowany ewentualnym błędnym zapisem.

G. Katalog CTRLPGM\STDIO\PANEL

G.1. Procedura blank_on_panel()

Wyswietlenie na panelu programowania pojedynczego znaku spacji.

G.2. Procedura byte_on_panel()

Wyswietlenie na panelu programowania 2-cyfrowej liczby hexadecymalnej, odpowiadajacej danej wartosci 1-bajtowej, przekazanej jako parametr wywolania procedury.

G.3. Procedura clear_linel()

Wyczyszczenie linii wyswietlacza panelu programowania robota od kursora do ostatniej kolumny.

G.4. Procedura cursor_left_on_panel()

Przesuniecie kursora o jedna pozycje w lewo na wyswietlaczu panelu programowania.

G.5. Procedura decimal_byte_on_panel()

Wyswietlenie na panelu programowania liczby dziesietnej, odpowiadajacej danej wartosci 1-bajtowej, przekazanej jako parametr wywolania procedury.

G.6. Procedura decimal_double_word_panel()

Wyswietlenie na panelu programowania liczby dziesietnej, odpowiadajacej danej wartosci 4-bajtowej, przekazanej jako parametr wywolania procedury.

G.7. Procedura decimal_word_on_panel()

Wyswietlenie na panelu programowania liczby dziesietnej, odpowiadajacej danej wartosci 2-bajtowej, przekazanej jako parametr wywolania procedury.

G.8. Procedura gc_panel_()

Wprocedury gc_panel() napisanej w jezyku C z poziomu programu napisanego w asemblerze ASM-86. Zasadniczym celem tego "posrednictwa" jest zachowanie zawartosci rejestrow, ktore moze zmienic procedura gc_panel().

G.9. Procedura gc_panel_with_wait_()

Oczekiwanie na nadejscie znaku z panelu programowania.

G.10. Procedura getchar_panel()

Bezprzerwaniowe odczytanie znaku z panelu programowania. Jesli bufor odbiorczy kanalu B ukladu Z8530 jest pusty to do procedury nadrzednej w rejestrze AX zwracana jest wartosc EOF, w przeciwnym razie rejestr AL zawiera kod wcisnietego klawisza, a rejestr AH jest wyzerowany.

G.11. Procedura message_na_panel()

Wyslanie do panelu programowania tekstu znakowego, do ktorego long pointer jest parametrem wywolania procedury. Teskt musi konczyc sie znakiem NULL (00H).

G.12. Procedura printf_char_()

Procedura wpisania pojedynczego znaku do bufora nadawczego kanalu, do ktorego dolaczony jest panel programowania. Jest to procedura posredniczaca pomiedzy programem nadrzednym napisanym w jezyku asemblera ASM86, a procedura printf_char() napisana w jezyku C. Zasadniczym celem tego "posrednictwa" jest zachowanie zawartosci rejestrow, ktore moze zmienic procedura printf_char().

G.13. Procedura printf_string_()

Wpisanie ciagu znakow, zakonczonych znakiem o kodzie 00H, do bufora nadawczego kanalu, do ktorego dolaczony jest panel programowania. Jest to procedura posredniczaca pomiedzyprogramemnadrzednymnapisanym w jezyku asemblera ASM86 a procedura printf_string() napisana w jezyku C. Zasadniczym celem tego "posrednictwa" jest zachowanie zawartosci rejestrow, ktore moze zmienic procedura printf_string().

G.14. Procedura putchar_panel()

Bezprzerwaniowe wyslanie znaku do panelu programowania.

G.15. Procedura rejestry_na_panel()

Wyswietlenie na panelu programowania robota zawartosci wszystkich rejestrow fizycznych mikroprocesora z miejsca, w ktorym wywolywana jest procedura REJESTRY_NA_PANEL.

G.16. Procedura word_on_panel()

Wyswietlenie na ekranie panelu programowania 4-cyfrowej liczby hexadecymalnej, odpowiadajacej danej wartosci 2-bajtowej, przekazanej jako parametr wywolania procedury.

G.17. Procedura esc_serv(c)

Obsługa odbioru sekwencji sterującej z panelu programowania.

G.18. Procedura gc_panel()

Udostepnienie kolejnej, nieodczytanej 1-bajtowej informacji z bufora pnlin[] posredniczacego pomiedzy procedurami zwiazanymi ze strumieniem stdin a ukladem transmisji szeregowej obslugujacego transmisje z/do panelu programowania robota.

G.19. Procedura mkimage()

Wpisanie kolejnego znaku do wskazanej strony obrazu panelu.

G.20. Procedura movekpad()

Obsluga przyciskow klawiatury do recznego sterowania.

G.21. Procedura pc_panel()

Wpisanie kolejnego bajtu do bufora nadawczego związanego ze zlaczem szeregowym pakietu MV-52, do ktorego jest dolaczony panel programowania.

G.22. Procedura popimage()

Przelaczenie na poprzedni obraz i odtworzenie go na wyswietlaczu panelu programowania (odczyt ze stosu).

G.23. Procedura printf_char()

Wpisanie pojedynczego znaku do bufora nadawczego kanalu, do ktorego dolaczony jest panel programowania.

G.24. Procedura printf_string()

Wpisanie ciagu znakow, zakonczonych znakiem o kodzie 00H, dobufora nadawczego kanalu, do ktorego dolaczony jest panel programowania.

G.25. Procedura pshimage()

Przelaczenie panelu na nastepny obraz (zapamiętanie aktualnego na stosie).

G.26. Procedura put_to_buf()

Wstawienie 1-bajtowej danej do bufora wyjściowego pnlout[], z którego informacja jest bezpośrednio wysyłana na złącze szeregowe, do którego podłączono panel programowania robota.

G.27. Procedura rd_panel()

Odczyt 1-bajtowej danej z bufora wejsciowego układu transmisji szeregowej (obslugujacego panel programowania robota) do programowego bufora pnlin[]. Procedura ta w zasadzie jest przeznaczona do wywolania z poziomu programu sluzacego do obslugi przerwania sygnalizujacego pelny bufor odbiorczy ukladu transmisji szeregowej, poniewaz nie sprawdza, czy bufor ten jest pelny.

G.28. Procedura wr_panel()

Wyslanie kolejnej, 1-bajtowej danej z bufora wyjściowego na złącze szeregowe do którego jest dołączony panel programowania robota. Procedura ta w zasadzie jest przeznaczona do wywolania z poziomu programu sluzacego do obslugi przerwania sygnalizujacego pusty bufor nadawczy ukladu transmisji szeregowej z/do panelu programowania robota, poniewaz nie sprawdza, czybufor ten jest pusty.

G.29. Nagłówek image.h

Plik zawiera definicje buforow zwiazanych z tworzeniem obrazu

SPIS TREŚCI