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

Ośrodek Automatyzacji Procesów Produkcji (OAP)

0714 A

Główny wykonawca mgr inż. Stanisław Wółtański

Wykonawcy mgr inż. Jacek Dunaj, mgr inż. Kazimierz Maliszewski, mgr inż. Waldemar Wieremiejczyk

Konsultant

Nr zlecenia K-109.1

Opracowanie i uruchomienie systemu operacyjnego dla sterownika grupy robotów i elastycznych systemów produkcyjnych
Etap nr 2: Wykonanie wersji PROM-owej systemu SIRTOS (obejmującej koprocesor arytmetyczny) dla jednostki AMS-M17-A8. Rozbudowa systemu SIRTOS do pracy z dużym modelem pamięci i instalacja na pakiecie jednostki centralnej PIAP.

Zleceniodawca KBN

Pracę rozpoczęto dnia 1991.09.15 **zakończono dnia** 1991.12.20

Z-ca Dyrektora d/s Badawczo-Rozwojowych
dr inż. J. Jabłkowski

Kierownik Ośrodka
dr inż. M. Wrzesień

Praca zawiera:

stron 8
rysunków
fotografii
tabel 1
tablic
załączników 2 (12+5 stron)

Nr rejestr. 6792

Rozdzielnik - ilość egz: 3

Egz. 1 BOINTE
Egz. 2 OAP-6
Egz. 3 OAP-6
Egz. 4
Egz. 5
Egz. 6

1

Analiza deskryptorowa SYSTEM OPERACYJNY CZASU RZECZYWISTEGO
MONITOR OPERATORSKI
STEROWANIE PROCESEM PRZEMYSŁOWYM
URZĄDZENIA AUTOMATYCZNEJ REGULACJI I STEROWANIA:
KSAP + STEROWNIK + MIKROPROCESOR

Analiza dokumentacyjna Praca zawiera zarys koncepcji realizacji wersji PROM-owej systemu operacyjnego czasu rzeczywistego SIRTOS wraz z monitorem operatorskim dla jednostki centralnej AMS-M17-A8 oraz MV-52. Zamieszczono również opis funkcji wewnętrznego sterownika przerwań mikroprocesora 80186.

Tytuły poprzednich sprawozdań

I Opracowanie i uruchomienie systemu operacyjnego dla sterownika grupy robotów i elastycznych systemów produkcyjnych:
Zadanie nr 1: Wykonanie wersji PROM-owej systemu SIRTOS dla jednostki MM86.
Sprawozdanie MERA-PIAP, nr rejestr. 6649, 1991

II W ramach zlecenia RP16.1 (CPBR 7.1) wykonano następujące prace:
Wersja systemu SIRTOS do sterownika grupy robotów i ESP.
Etap 1. (Zad. 4.2.1) Opracowanie i uruchomienie monitora operatorskiego czasu rzeczywistego systemu R-T SIRTOS. Dołączenie monitora wraz z dezasemblerem do systemu w małym modelu pamięci.
Sprawozdanie MERA-PIAP, nr rejestr. 6497, 1990
Etap 2. (Zad. 4.2.2) Generacja systemu SIRTOS dla sterownika grupy robotów i ESP na podstawie dokumentacji techniczno-ruchowej sterownika.
Sprawozdanie MERA-PIAP, nr rejestr. 6519, 1990

III Praca własna OAP
Dezasembler instrukcji procesorów Intel 8086/88/186/188.
Sprawozdanie MERA-PIAP, nr rejestr. 6548, 1990

UKD

PIAP 41/88 10000

Spis treści

1. WPROWADZENIE I UWAGI OGÓLNE

W ramach kontynuacji tematu "Opracowanie i uruchomienie systemu operacyjnego dla sterownika grupy robotów i elastycznych systemów produkcyjnych" dokonano próby instalacji systemu SIRTOS na jednostkach centralnych AMS-M17-A8 (produkcji firmy SIEMENS) oraz MV-52 (produkcji PIAP); obie jednostki posiadają mikroprocesor INTEL 80186. Przedmiotem pracy była wersja PROM-owa systemu zainstalowana wcześniej na jednostkach MM86 i MM16 systemu INTELDIGIT-PROWAY, zbudowanych na mikroprocesorze INTEL 8086. Podstawowym narzędziem do przygotowanie zarówno systemu jak i aplikacji jest AZTEC C, wersja 3.40 i następne. Podobnie jak i w poprzednich wersjach system SIRTOS zawiera:

- koordynator zadań użytkowych,
- monitor wielopoziomowych przerwań maskowalnych i logicznych,
- bibliotekę systemowych funkcji aktywacji i synchronizacji zadań oraz wymiany informacji między nimi, a także inicjacji sprzęgów szeregowych, równoległych, sterowników przerwań i timerów,
- monitor operatorski (debugger) wraz z dezasemblerem (również w zakresie procesora 80186).

Metodyka przygotowania zadań użytkowych stanowiących aplikację została omówiona w poprzednich sprawozdaniach.

2. PODSTAWOWE ZAŁOŻENIA I KONCEPCJA REALIZACJI SYSTEMU

Przypomnijmy podstawowe założenia realizacji wersji PROM-owej systemu SIRTOS

1. System, tzn. główne funkcje systemowe, monitor przerwań, funkcje inicjacji, monitor operatorski z dezasemblerem i obsługą przerwań INT 1 -praca krokowa- oraz INT 3 (praca z 'breakpoint'-em) znajdują się w oddzielnym segmencie kodu oznaczonym symbolem EP. Fizycznie segment ten znajduje się w stronie (64 kB) pola adresacji procesora przeznaczonej na pamięć EPROM (EP=F000).

2. Aplikacja, tzn. zadania użytkowe, funkcje obsługi przerwań (zewnętrzne i wewnętrzne), moduł inicjujący 'srom.o' ('rom.o') z biblioteki AZTEC-a znajduje się w segmencie kodu CS.W ramach aplikacji znajdują się dodatkowo dwa moduły do konsolidacji ('srt.o' oraz 'usr.o') konieczne do realizacji komunikacji aplikacja - system.
3. Dane systemowe oraz użytkowe (w tym wektor przerwań i stosy zadań) znajdują się we wspólnym segmencie danych DS. Z uwagi na obecność wektora przerwań segment ten został umieszczony w zerowej stronie pola adresacji procesora (DS=1).
4. Programy źródłowe stanowiące system powinny być jednolite dla obu jednostek, których wyboru dokonujemy definiując odpowiednią wartość parametru kompilacji warunkowej (ZSS=1 dla MV-52 oraz ZSS=0 dla AMS-M17-A8).

Z uwagi na fakt wykorzystania niskich numerów wektora przerwań przez wewnętrzny sterownik przerwań procesora 80186 przyporządkowano przerwania logiczne (programowe) następującym pozycjom:

INT 20 - wejście do monitora operatorskiego,
INT 21 - obsługa głównych i pomocniczych funkcji systemowych
INT 22 - wejście z systemu do aplikacji w celu dokonania jej inicjacji (instalacja zadań, inicjacja wewnętrznych układów we/wy sterownika),
INT 23 - wejście do monitora przerwań.

3. WEKTOR PRZERWAN

Tablicę "vint", stanowiącą wektor przerwań systemu, należy zainicjować adresami zewnętrznych funkcji obsługi przerwań. Kolejność adresów wynika z przyporządkowania przerwaniom priorytetów przez sprzętowe podłączenie ich źródeł do poszczególnych wejść sterowników przerwań (PIC). W przypadku nieciągłości w numeracji przerwań należy uwzględnić ten fakt przez wpisanie do wektora przerwań stałej NULLFP (lub nazwy funkcji 'un' reprezentującej obsługę niezidentyfikowanego przerwania). Obsługa przerwań nie wykorzystująca uniwersalnego monitora przerwań jest również możliwa. Należy wówczas pamiętać m.

in. o przechowaniu rejestrów procesora. Pełny wektor przerwań dla obu jednostek zawiera poniższa struktura:

```
struct vinp vint [] = {
                { bo,  CS },  { uo,  CS },  { eo,  CS }, /* 5 - 7 */
 { tr0, CS },  { unr, CS },  { dma0,CS },  { dma1,CS }, /* 8 -11 */
 { int0,CS },  { int1,CS },  { int2,CS },  { int3,CS }, /* 12-15 */
 { unr, CS },  { unr, CS },  { tr1, CS },  { tr2, CS }, /* 16-19 */
#if MSRTS
 { BE,  EP },                                          /* 20 */
#else
 { un,  CS },
#endif
                { BE+9,EP },  { api, CS },  { BE+3,EP }, /* 21-23 */

 { un,  CS },  { un,  CS },  { un,  CS },  { un,  CS }, /* 24-31 */
 { un,  CS },  { un,  CS },  { un,  CS },  { un,  CS }, /* vacat */

 { coa, CS },  { zs,  CS },  { zk,  CS },  { ee,  CS }, /* master */
 { un,  CS },  { un,  CS },  { un,  CS },  { un,  CS }, /* 32-39 */
#if ZSS
 { im0, CS },  { im1, CS },  { im2, CS },  { im3, CS }, /* master_B*/
 { im4, CS },  { im5, CS },  { im6, CS },  { im7, CS }  /* 40-47 */
#else
 { un,  CS },  { un,  CS },  { un,  CS },  { un,  CS }, /* 40-47 */
 { un,  CS },  { un,  CS },  { un,  CS },  { un,  CS }  /* vacat */
#endif
} ;
```

4. MAPA PAMIECI SYSTEMU URUCHOMIENIOWEGO I DOCELOWEGO

Adres (hex)	Model pamięci w systemie uruchomieniowym	Model pamięci w systemie docelowym	Adres (hex)
0000 RAM 03FF	wektor przerwań: monitora operatorskiego, SIRTOS-a i aplikacji	wektor przerwań monitora operatorskiego, SIRTOS-a i aplikacji	0000
RAM	dane zainicjowane systemowe i użytkowe stos podstawowy 2kB użytkowe dane niezainicjowane	dane zainicjowane systemowe i użytkowe stos podstawowy 2 kB użytkowe dane niezainicjowane	RAM
DS: FFFF	dodatkowy obszar stosu	dodatkowy obszar stosu	DS: FFFF
CS: 0000 RAM CS: FFFF	kod zadań użytkowych kopia użytkowych danych zainicjowanych	kod zadań użytkowych kopia użytkowych danych zainicjowanych	CS: 0000 ROM max 64 kB CS: FFFF
EP: 0000 ROM max 32 kB EP: 7FFF	jądro systemu SIRTOS kopia systemowych danych zainicjowanych	jądro systemu SIRTOS kopia systemowych danych zainicjowanych	EP: 0000 ROM max 32 KB EP: 7FFF
F8000 ROM 32 kB FFFF0 FFFFF	kod monitora operatorskiego OAP argument długiego skoku do monitora OAP	argument długiego skoku do aplikacji	ROM FFFF0 FFFFF

5. PODSUMOWANIE I WNIOSKI

Realizacja wersji PROM-owej SIRTOS-a dla jednostek AMS-M17-A8 i MV-52 wymagała - w porównaniu do poprzednio wykonanej dla MM86 i MM16 - zmiany funkcji inicjującej interfejs szeregowy zbudowany na układzie Z8530A. Układ ten jest znacznie trudniejszy do obsługi

programowej od używanego w tamtych jednostkach układu USART (8251A). Wielość rejestrów we/wy i różnorodność trybów pracy stanowiła dodatkową trudność w wyborze właściwych parametrów do pracy w środowisku systemu czasu rzeczywistego. W pracy przerwaniowej obecność tylko jednego źródła przerwania dla nadawania jak i odbioru przesyłanej informacji powoduje dodatkową komplikację funkcji obsługi przerwania. Obie realizacje wymagają w obecnej postaci dodatkowego testowania dla bardziej złożonej aplikacji (np. z większą liczbą źródeł przerwań sprzętowych).

Załącznik A : stron 12

1. WEWNĘTRZNY STEROWNIK PRZERWAŃ MIKROPROCESORA INTEL-80186 (80188)

Procesory 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 może 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.

1.1. 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).

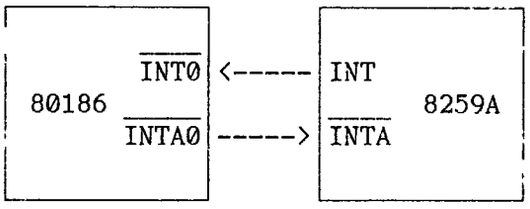
W trybie MASTER (non-iRMX_86) wewnętrzny sterownik przerwań może pracować w trzech różnych "podtrybach" pracy: fully nested, cascade lub special fully nested.

1.1.1. 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).

1.1.2. 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 zgłoszenia przerwania 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.

1.1.3. 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.

1.1.4. 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.

1.1.5. 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.

1.2. 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).

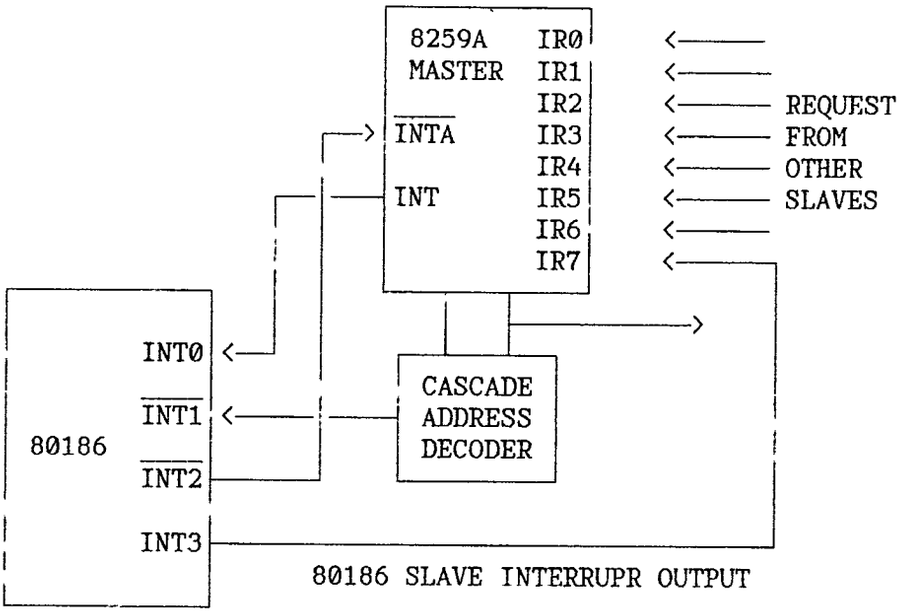
W trybie pracy iRMX_86 wewnętrzny sterownik przerwań mikroprocesora 80186 (80188) pracuje jako sterownik podrzędny (SLAVE) w stosunku do zewnętrznego układu PIC 8259A, pracującego jako sterownik MASTER. Przerwania od innych wewnętrznych układów mikroprocesora 80186 (80188) tzn. od układów DMA i timerów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

Tryb 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.

1.2.1. 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 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).

1.2.2. 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.

1.3. 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

1.3.1. 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).

1.3.2. 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.

1.3.3. 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.

1.3.4. 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.

1.3.5. 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.

1.3.6. 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.

1.3.7. 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.

1.3.8. 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.

1.3.9. 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

1.4. 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).

SPIS TRESCI

Załącznik B: stron 5

```
/*********************************************************************************************
               The interrupts constants definitions: "gic.h" file                                */
#define ZSS         1              /* 1- j.c. MV-52,    0- SIEMENS */
#define IC_186      0
#define MASTER      1
#if ZSS
#define MASTER_B    2
#endif
/*******************************************************************************************/
          /* Stale do programowania sterownika przerwan - IC_186 */
               /* 80_186 Interrupt Controller Registers */
#define MRV       0xEC
#define EOIR      0x22
#define PR        0xFF24        /* PSR = 0xFF26 */
#define MR        0x28          /* PMR = 0xFF2A */
#define ISR       0x2C
#define IRR       0x2E
#define ICSR      0x30
#define TCR       0x32
#define DCR       0xFF34        /* D1CR= 0xFF36 */
#define ICR       0xFF38        /* I1CR= 0xFF3A, I2CR=0xFF3C, I3CR=0xFF3E */
        /* Stale do programowania sterownika przerwan 8259A - MASTER */

#define PICM    0x0100         /* MASTER port address */
#define ICW1M    0x11          /* edge, master/slave, icw4 */
#define ICW2M    0x20          /* int type 32-39 */
#define ICW3M    0x00          /* no slaves */
#define ICW4M    0x01          /* non-specific EOI, 86/88 mode */
#define MMR      0xFD          /* OCW1, unmask 'zs' in master */
      /* Stale do programowania sterownika przerwan 8259A - MASTER_B */

#define PICB    0x0130         /* MASTER_B port address */
#define ICW1B    0x13          /* edge, single, icw4 */
#define ICW2B    0x28          /* int type 40-47 */
#define ICW3B    0x00          /* never */
#define ICW4B    0x01          /* non-specific EOI, 86/88 mode */
#define BMR      0xFF          /* OCW1, mask ints in single */
/*******************************************************************************************/
                      /* 80_186 Timer 0 Control Registers */
#define T0CR    0xFF50          /* Count Register */
#define T0MCA   0x0052          /* Max Count A */
#define T0MCB   0x0054          /* Max Count B */
#define T0CW    0xC005          /* Mode/Control Word: EN,~INH, EXT, CONT */

                      /* 80_186 Timer 1 Control Registers */
#define T1CR    0xFF58          /* Count Register */
#define T1MCA   0x005A          /* Max Count A */
#define T1MCB   0x005C          /* Max Count B */
#define T1CW    0xC001          /* Mode/Control Word: EN,~INH, CONT */

                      /* 80_186 Timer 2 Control Registers */
#define T2CR    0xFF60          /* Count Register */
#define T2MCA    40000          /* Max Count A: 10ms/500ns=1/4 CPU clock rate */
#define T2CW    0xE001          /* Mode/Control Word: EN,~INH,INT, CONT */
/*******************************************************************************************/
                      /* Inicjacja jednostki AMS-M17-A8 oraz MV-52
                        -stale do programowania Z8530A */
#define UCAZ_B  0x110          /* adres slowa sterujacego ukladu Z8530_B */
#define UDAZ_B  0x114          /* adres slowa danych ukladu Z8530_B */
#define UCAZ_A  0x112          /* adres slowa sterujacego ukladu Z8530_A */
#define UDAZ_A  0x116          /* adres slowa danych ukladu Z8530_A */
#define CTRL_C UCAZ_B          /* konsola systemowa- adres slowa sterujacego */
#define DATA_C UDAZ_B          /* konsola systemowa- adres slowa danych */
#define CTRL_M UCAZ_A          /* konsola operatorska- adres slowa sterujacego */
#define DATA_M UDAZ_A          /* konsola operatorska- adres slowa danych */
#if ZSS
#define LBRG       24          /* MV-52 dla 4800 */
#else
#define LBRG       30          /* dla transmisji 4800 */
#endif
#define UL1     0x14C          /* prawa, dolna lampka - green; 0x14E- lewa */
#define HRTS     0xEA          /* RTS- High */
#define LRTS     0xE8          /* RTS- Low */
/*******************************************************************************************/
                        /* Inicjacja jednostki MV-52 */
#define SW      0x140          /* adres slowa stanu j.c.; rej we/wy */
#define WD      0x150          /* Watchdog- podtrzymanie budzika */
/*******************************************************************************************/
                            /* Maski przerwan */
#define TxRDY    0x02          /* maska przerwania TxRDY w MMR */
#define RxRDY    0x02          /* maska przerwania RxRDY w MMR */
#if ZSS
#define CBTMO    0x40          /* maska BTMO w SW */
#define PFIN     0x80          /* maska PFIN -zaniku zasilania w SW */
#endif
```

```
/*************************************************************************
            The application system initialization: "uinit.c"  file
                     Global functions: apinit, mstack
*************************************************************************/
#include "seg.h"
#include "gcc.h"
#include "gsc.h"
#include "gse.h"
#include "ees.h"
#include "guc.h"
#include "gue.h"
#include "gic.h"
#include "gfe.h"
#include "exm.h"
/*************************************************************************/
#define NRTMAX        6                        /* maksymalny numer zadania */
#define STACK      OFF              /* _ins = STACK - adres dodatkowego stosu */
/*************************************************************************/
unsigned int _ins = STACK ;
int _nrtm           = NRTMAX ;
char ** ptsym      = EMPTY ;
int ( ** pteds ) ()     = NULA ;
void ( ** ptentry ) () = NULA ;
char * plst            = NULA ;
struct sicw * picw     = NULA ;
struct tcblock * ptcb  = NULA ;
struct tcblock tcb[ NRTMAX+ 1 ] = OFF ;
/*************************************************************************/
static void ( *tentry [] ) () = { MAIN,
                                   tads, kbord, tmon,
                                   mmc,  ttgt,  NULLFP
} ;
static char lst []           = { 8,
                                   8,    8,     8,
                                   8,    8,     8
} ;                                          /* 56 from 64 portions occupied */
/*************************************************************************/
static struct sicw icw [] =  {
                    { ICR,  TCR,   MR,    MRV,   EOIR,  ICSR },
                    { PICM, ICW1M, ICW2M, ICW3M, ICW4M, MMR }
#if ZSS
                   ,{ PICB, ICW1B, ICW2B, ICW3B, ICW4B, BMR }
#endif
} ;
unsigned int tcw[ 3 ][ 4 ] = {
                    { T0CR, T0CW, T0MCA, T0MCB },
                    { T1CR, T1CW, T1MCA, T1MCB },
                    { T2CR, T2CW, T2MCA,    NO }
} ;
/*************************************************************************/
#if MSRTS
static char *tasksym[ NRTMAX+ 1 ] = {  "SYS",            /* zadanie systemowe */
                                       "ADS",
                                       "KBD", "MON",
                                       "MMC",
                                       "BGT",
                                       "VCT"
} ;
#endif
/*************************************************************************/
unsigned int dsval, halfds ;
static char sdiag [] = "  \33H\33J\33Y70Zniszczenie stosu zadania nn !" ;
/*************************************************************************/
void mstack ()                      /* obsluga zniszczenia stosu zadania _act */
    {
    dec2( _act, sdiag+ 39 ) ;
    wstring( sdiag ) ;
    return ;              /* zatrzymanie pracy procesora po powrocie - halt */
}                                                               /* end mstack */
/*************************************************************************/
void apinit ()       /* INICJACJA APLIKACJI - pod zamknietym ukladem przerwan */
    {
    int nrt ;

    writeword( 0, 8, nmi ) ;        /* interrupt service address INT 2 */
    writeword( 0, 10, CS ) ;        /* for all models in POWER-DOWN */
    _ptentry = tentry ;
    _plst = lst ;
    _picw = &icw[ pic ] ;
#if MSRTS
    _ptsym = tasksym ;
    delay( 3 ) ;                    /* opoznienie 3 ms */
#else
    delay( 3000 ) ;                 /* opoznienie 3 s */
#endif
                       /* Pheripheral initialization */
```

```
/*******************************************************************************/
static void txrdy ( semafor )          /* obsluga przerwania TxRDY */
   char *semafor ;
   {
   static char flag = ON, *adrinf ;

   if ( flag ) {
      adrinf = adrk+ 2 ;
      flag = OFF ;
   }
   if ( *adrinf )                      /* czy koniec komunikatu ? */
      outv( DATA_C, *adrinf++ ) ;      /* wyslanie kol. znaku; no 'sch' */
   else {
      flag = ON ;                      /* koniec komunikatu */
      outv( CTRL_C, 0x28 ) ;           /* reset Ts IP when end of message */
      SIGNAL ( semafor, 1 ) ;          /* koniec oczekiwania: 'screen' ON */
      SIGNAL ( adrk, 1 ) ;             /* komunikat wyslano, entry 'sch' */
   }
   return ;
}                                                              /* end txrdy */
/*******************************************************************************/
void rxrdy ( semafor )          /* obsluga przerwania RxRDY */
   char *semafor ;
   {
   static char tmp ;
   static int pwm ;

   tmp = getch_wait () ;
#if SPEED
   if ( tmp == ' ' ) {
      SIGNAL ( &reflex, 1 ) ;     /* znak ' ' zostal przyjety */
      return ;
   }
#endif
#if MSRTS
   if ( tmp == ETX ) {       /* M O N I T O R   O P E R A T O R S K I   SRTS */
      msrts () ;
      return ;
   }
#endif
   if ( *semafor == DBW )
      return ;
   *( bufwe+ pwm ) = tmp ;              /* wpisanie znaku do bufora zadania */
   if ( ++pwm == DBW )                  /* czy koniec bufora ? */
      pwm = 0 ;        /* aktualizacja wskaznika pierwszego wolnego miejsca */
   SIGNAL ( semafor, 1 ) ;              /* znak zostal przyjety */
   return ;
}                                                              /* end rxrdy */
/*******************************************************************************/
static void zxrdy ( trick, semafor )    /* obsluga przerwania z Z_8530A */
   char *semafor, *trick ;
   {
   static char rr3 ;

   while () {
      rr3 = inr( UCAZ_A, 3 ) ;          /* RR3 only in channel A */
      if ( rr3 & 6 ) {
         if ( rr3 & 4 )                 /* channel B: Rx- IP; pusty b. odb. */
            rxrdy( semafor+ 1 ) ;       /* keyboard */
         else
            if ( rr3 & 2 )              /* channel B: Tx- IP; pusty b. nad. */
               txrdy( semafor ) ;       /* screen */
         continue ;
      }
      else {
         pic = MASTER ;
         retfar () ;                    /* far return to interrupted task */
      }
   }                                    /* end while */
}                                                              /* end zxrdy */
/*******************************************************************************/
void zs ()                       /* wywolanie obslugi przerwania TxRDY i RxRDY */
   {
   savereg () ;
   servints( zxrdy, &screen ) ;
}                                                                 /* end zs */
/*******************************************************************************/
```

```
/*******************************************************************************
                    The segment registers file: "seg.h"
*******************************************************************************/
#define DS          1           /* system and application DS */
#define CS      0x500           /* application CS */
#define EP      0x800           /* system CS */
/*#define EP    0xF000          /* system CS */
#define BE      0x9B0           /* "seprom" address */
#define MAIN    0x62            /* SIRTOS 'main+ 3' address */
/*******************************************************************************/
#define IC 186      0
#define CZAS        0           /* 1- test zegara, 0- bez */
/*******************************************************************************/
```

```
/*******************************************************************************
                 Warunkowe sterowanie kompilacja: "exm.h"
*******************************************************************************/
#define MSRTS   1          /* Czy jest monitor operatorski SRTS? 1-tak; 0-nie */
extern void msrts () ;     /* wywolanie mon. op. przez przerwanie programowe */
```

```
*=sirtos.
R=srom.o
L=srt.o usr.o
W=usersfs.o user.o
U=uinit.o uints.o
I=vi.o
T=tkbd.o tmon.o tmmc.o
D=tbgt.o ram.o tads.o
M=-c 500 -d 1
K=10

$*hex: $*exe
        @hex86 -z -s$K $*.exe
        @echo SIRTOS OK!
        @echo ap makefile end

$*exe: $I $R $L $W $U $T $D
        @ln -T $M -o $@ $I $R $L $W $U $T $D -lc

srt.o:      gcc.h gsc.h gse.h       guc.h gue.h             exm.h seg.h

vi.o:       gcc.h                                 gfe.h exm.h seg.h
uinit.o:    gcc.h gsc.h gse.h ees.h       gue.h gic.h gfe.h exm.h seg.h
uints.o:    gcc.h gsc.h gse.h ees.h guc.h gue.h gic.h       exm.h
usersfs.o:  gcc.h                   guc.h       gic.h       exm.h

tkbd.o:     gcc.h             ees.h guc.h gue.h
tmon.o:     gcc.h             ees.h guc.h gue.h gic.h
tmmc.o:     gcc.h             ees.h guc.h gue.h                         mmc.h
tads.o:     gcc.h                               gic.h
tbgt.o:     gcc.h             ees.h       gue.h       gfe.h
ram.o:      gcc.h             ees.h       gue.h                   seg.h
```