EGMONT INSTRUMENTS

Informacji na temat kart produkowanych dawniej przez Ambex udziela firma Egmont Instruments.

Pod adresem **http://www.ambex.com.pl** powstaje archiwum instrukcji i oprogramowania do kart komputerowych produkowanych dawniej przez Ambex, a obecnie jeszcze w części oferty przez Egmont Instruments. Archiwum to jest systematycznie uzupełniane. Jeśli poszukują Państwo informacji do starych lub aktualnych wyrobów, prosimy kierować się właśnie pod powyższy adres w Internecie. Jeśli nie znajdą tam Państwo potrzebnej informacji, prosimy o bezpośredni kontakt z nami.

Strony http://www.ambex.com.pl są prowadzone bezpośrednio przez firmę Egmont Instruments.

EGMONT INSTRUMENTS	**tel. (0-22) 850-62-05, 850-64-30**
00-867 Warszawa, ul. Chłodna 39, pawilon 11	**fax (0-22) 654-02-48**
http://www.egmont.com.pl	**e-mail egmont@egmont.com.pl**
http://www.ambex.com.pl	**e-mail ambex@ambex.com.pl**

ambex®

D O K U M E N T A C J A D R I V E R A
M O D U Ł U K O N T R O L N O - P O M I A R O W E G O

L C - 0 5 5 - P I O , L C - 0 5 5 - D C U

W E R S J A 2 . 0

Wydanie: lipiec 1992

2 Dokumentacja driver'a modułów wejść/wyjść cyfrowych

1. Informacje ogólne.

Przy tworzeniu oprogramowania modułów przyjęto zasadę, że cała komunikacja z modułem prowadzona jest za pośrednictwem rezydentnego programu dostępnego dla programów użytkowych poprzez przerwanie programowe. Takie rozwiązanie ma następujące zalety:

- użytkownik jest zwolniony ze znajomości szczegółów technicznych tak modułu jak i używanego komputera,
- rozwiązanie to jest niezależne od używanej implementacji języka wyższego poziomu.

Program obsługi został napisany w standardzie driver'ów systemu operacyjnego MS-DOS (wersja 3.1. i wyższe). Główną przyczyną wyboru takiego rozwiązania jest umożliwienie prostego badania obecności driver'a w systemie. Jedyną wykorzystywaną standardową funkcją driver'a jest funkcja inicjalizacji wykonywana w trakcie jego instalacji. Po zainstalowaniu driver służy tylko jako obsługa danego przerwania programowego.

Opisywany driver obsługuje do 4 modułów wejść/wyjść cyfrowych. Każdy z nich może być dowolnego typu: LC-8255, LC-055-PIO, LC-055-DCU.

2. Instalacja driver'a w systemie.

System operacyjny MS-DOS będzie instalował driver po umieszczeniu w pliku CONFIG.SYS poniższej linii:

(w poniższym opisie napisy ujęte w nawiasy trójkątne <> oznaczają nazwy symboliczne, nawiasy kwadratowe [] wydzielają elementy, które mogą lecz nie muszą wystąpić natomiast nawiasy klamrowe {} - oznaczają powtórzenie ich zawartości jeden lub więcej razy; parametry mogą być otaczane spacjami)

DEVICE=[<drive:>][<pathname>]LC055.DRV <parametry instalacji>

<parametry instalacji> różnią się nieco dla poszczególnych typów modułów. I tak:

LC-8255:
<parametry instalacji> ::= {/M=<m>;<typ>[;<i>[;<l>[;<p>[;<ctn>]]]]}

LC-055-PIO:
<parametry instalacji> ::= {/M=<m>;<typ>[;<i>[;<ctn>[;<ctc>]]]}

LC-055-DCU:
<parametry instalacji> ::= {/M=<m>;<typ>[;<i>[;<c>[;<p>[;<ctn>[;<ctc>]]]]]}

gdzie:

M - klucz modułu; wszystkie informacje po nim aż do następnego takiego klucza dotyczą jednego modułu
m - nazwa modułu (A, B, C, D)
typ - **8255** (LC-8255); **PIO** (LC-055-PIO); **DCU** (LC-055-DCU)

każdy z poniżej opisanych parametrów może być pominięty - przybiera wówczas wartość domyślną:

i - numer przerwania IRQ:

LC-8255 : 2..7 (przerwanie programowe 10..15)
LC-055-PIO : 2..4 (przerwanie programowe 10..12)
LC-055-DCU : 2..5 (przerwanie programowe 10..13)

niewystąpienie tego parametru oznacza niepodłączenie modułu do żadnego przerwania

```
c    - konfiguracja źródeł przerwań:
c  ::= [<src1>], [<src2>], [<src2>], [<src2>]
       src1 - nazwa źródła:
              EXT  - sygnał zewnętrzny
              NO   - linia nie jest podłączona
       src2 - nazwa źródła:
              EXT  - sygnał zewnętrzny
              CTC  - wyjście kanału CTC
              NO   - linia nie jest podłączona
```

pominięcie dowolnego z parametrów oznacza podłączenie danej linii przerywającej do sygnału zewnętrznego

```
l    - poziom przerywający:
       1    - poziom wysoki
       0    - poziom niski
```

domyślną wartością jest 1

```
p    - liczba układów 8255 (1 lub 2, domyślnie 2)
ctn  - liczba układów 8253; musi być 1
ctc  - częstotliwość zegara CTC
PIO:   częstotliwość w kHz (0.5, 2, 4)
DCU:   [<f>], [<f>], [<f>]
f      - częstotliwość zegara dla kolejnego kanału (0.5, 1); częstotliwość = 0
       oznacza przyłączenie kanału CTC do zegara zewnętrznego
```

domyślną częstotliwością jest 0 (zegar zewnętrzny)

Przykłady:

```
DEVICE=C:\LC055.DRV /M=A;8255;3 /M=B;PIO;4 /M=C;DCU;5
```

Moduł A: LC-8255, przerwanie IRQ3 poziomem wysokim, 2 układy 8255, 1 układ 8253
Moduł B: LC-055-PIO, przerwanie IRQ4, 1 układ 8253 przyłączony do zegara zewnętrznego
Moduł C: LC-055-DCU, przerwanie IRQ5, wszystkie linie przerywające podłączone do sygnałów zewnętrznych, 2 układy 8255, 1 układ 8253 - wszystkie kanały podłączone do zegarów zewnętrznych

```
DEVICE=C:\LC055.DRV /M=A; DCU ; 3; EXT,NO,CTC,CTC; 2; 1; 0.5, 0, 1; /M=B; PIO; ;
1; 0.5; /M=C; 8255; 4; 0
```

(w rzeczywistości powinna to być jedna linia)

Moduł A: LC-055-DCU, przerwanie IRQ3, linia INT 1 - sygnał zewnętrzny, INT 2 - niepodłączona, INT 3 i INT 4 - podłaczone do kanałów CTC, odpwiednio kanał 1 i kanał 2, 2 układy 8255, 1 układ 8253 - kanał 0 podłączony do zegara wewnętrznego 500 kHz, kanał 1 - do zegara zewnętrznego, kanał 2 - do zegara 1 MHz
Moduł B: LC-055-PIO, żadne przerwanie nie jest podłączone, 1 układ 8253 przyłączony do zegara wewnętrznego 500 kHz
Moduł C: LC-8255, przerwanie IRQ4 poziomem niskim, 1 układ 8255, 1 układ 8253

Jeżeli driver został zainstalowany poprawnie wyświetla informacje o konfiguracji zadeklarowanych modułów. Informacja o pojedynczym module ma postać:

```
LC-8255:     MODULE <m>: LC-8255; <int>; <np> PORTS
```

gdzie:

```
<m>      - kod numeru modułu: A,...,D
<int>    - opis układu generacji przerwania:
           IRQ<i>/<l> - układ podłączony do przerwania IRQ<n>, przerwanie
                        aktywne poziomem <l>
           NO IRQ     - układ nie jest podłączony do żadnego przerwania
<np>     - liczba portów: 3 * liczba układów 8255 (<p>)
```

LC-055-PIO: MODULE <m>: LC-055-PIO; <int>; 6 PORTS; CTC <f>

gdzie:

<m> - kod numeru modułu: A,...,D
<int> - opis układu generacji przerwania:
IRQ<i> - układ podłączony do przerwania IRQ<n>
NO IRQ - układ nie jest podłączony do żadnego przerwania
<f> - 500, 2000 lub 4000 - CTC podłączone do wewnętrznego zegara o częstotliwości <f> kHz
EXT - CTC podłączone do zegara zewnętrznego

LC-055-DCU: MODULE <m>: LC-055-DCU; <int>; <np> PORTS; CTC <f1>/<f2>/<f3>

gdzie:

<m> - kod numeru modułu: A,...,D
<int> - opis układu generacji przerwania:
IRQ<i>/<src1>/<src2>/<src3>/<src4> - układ podłączony do przerwania IRQ<n>; <src_n> opisują źródło przerwania każdej linii przerywającej:
NO - linie nie jest podłączona
EXT - linia podłączona do sygnału zewnętrznego
CTC - linia podłączona do wyjścia kanału CTC
NO IRQ - układ nie jest podłączony do żadnego przerwania
<np> - liczba portów: 3 * liczba układów 8255 (<p>)
<f1>, <f2>, <f3> - opis poszczególnych kanałów CTC:
500 lub 1000 - kanał CTC podłączony do wewnętrznego zegara o częstotliwości <f> kHz
EXT - kanał CTC podłączony do zegara zewnętrznego

W przypadku błędnej deklaracji modułu driver wyświetla odpowiedni komunikat i nie instaluje się. Parametry analizowane są do wystąpienia pierwszego błędu. Błędy mogą być następujące:

MISSING EQUAL SIGN - brak znaku równości po kluczu /M
INCORRECT MODULE NAME - błędna kod modułu (różny od A, B, C, D)
INCORRECT MODULE TYPE - błędny typ modułu (różny od 8255, PIO, DCU)
MISSING MODULE TYPE - brak typu modułu
MULTIPLE MODULE DECLARATION - wielokrotna deklaracja tego samego modułu (np. podwójna deklaracja modułu A)
INCORRECT INTERRUPT NUMBER - błędny numer przerwania (nie jest liczbą naturalną lub jest spoza zakresu dopuszczalnego dla danego typu modułu)
INCORRECT INTERRUPT SOURCE - błędny kod źródła przerwania (różny od NO i EXT dla <src1> lub różny od NO, EXT i CTC dla <src2>)
INCORRECT INTERRUPT LEVEL - błędny poziom przerwania (różny od 0 i 1)
INCORRECT NUMBER OF 8255 CHIPS - błędna liczba układów 8255 (nie jest liczbą naturalną lub jest większa od 2)
INCORRECT NUMBER OF 8253 CHIPS - błędna liczba układów 8253 (nie jest liczbą naturalną lub jest większa od 1)
INCORRECT 8253 CLOCK - błędna częstotliwość zegara CTC (różna od 0, 0.5, 2 i 4 dla LC-055-PIO lub różna od 0, 0.5 i 1 dla LC-055-DCU)
EXTRA PARAMETER(S) - pomiędzy ostatnim parametrem deklaracji a następną deklaracją lub końcem linii wystąpiły znaki różne od spacji i tabulatora

3. Opis driver'a.

3.1. Informacje wstępne.

Przed driver'em postawione zostały następujące zadania:

- pełne wykorzystanie możliwości sprzętowych oferowanych przez obsługiwane moduły
- realizacja pewnych funkcji, dzięki którym możliwe jest pisanie uniwersalnych programów, niezależnych od instalacji konkretnego modułu
- ujednolicenie obsługi różnych typów modułów tak, aby wspólne ich cechy były widziane identycznie

Driver rozpoczyna swoją pracę już w momencie ładowania systemu. Wówczas to pobiera i analizuje parametry instalacji podane w poleceniu "DEVICE=..." z pliku CONFIG.SYS - dzięki tym informacjom możliwe jest pisanie programów niezwiązanych z konkretną instalacją modułu. Następnie wykonywane jest tzw. twarde zerowanie wszystkich zadeklarowanych modułów, w trakcie którego wykonywane jest wstępne programowanie - m.in. ustawianie wszystkich portów na kierunek "wejście".

Driver jest do pewnego stopnia wielowejściowy (re-entrant). Oznacza to, że w trakcie wykonywania jednej z funkcji driver'a można wykonać inną. W chwili obecnej liczba takich równoległych wejść nie może przekroczyć 3, w przeciwnym razie driver zwraca błąd LCO_REJECTED.

Driver został przystosowany do jednolitej obsługi wszystkich modułów cyfrowych produkowanych przez firmę AMBEX. Ponieważ jednak moduły te różnią się między sobą możliwościami sprzętowymi, pewne funkcje dla niektórych typów modułów nie są dostępne, w innych - od typu modułu zależy dokładna interpretacja parametrów funkcji.

3.2. Standardy oznaczeń, numeracja, dane charakterystyczne.

Przy pisaniu tak oprogramowania jak i niniejszej dokumentacji przyjęto następujące zasady:

- wszystkie nazwy pól rekordów, stałych itp. (z wyjątkiem nazw funkcji) opatrzone są przedrostkiem LCO_; występujący w niektórych nazwach przyrostek _55 został wprowadzony dla odróżnienia ich od analogicznych nazw związanych z driver'ami modułów analogowych produkcji firmy AMBEX
- nazwy występujące w dokumentacji są identyczne z nazwami występującymi w plikach źródłowych dla języków C, Pascal, assembler (z dokładnością do rozróżnienie małe / duże litery).

Numer przerwania programowego związanego z driver'em:

nazwa przerwania	numer przerwania szesnastkowy	numer przerwania dziesiętny
LCO11_16	98	152

Driver widziany jest w systemie DOS jako urządzenie. Nazwa tego urządzenia jest następująca: LCO55^^^.

Kodowanie numerów modułów:

moduł	nazwa kodu	wartość
A	LCO_MODA	1
B	LCO_MODB	2
C	LCO_MODC	3
D	LCO_MODD	4

Wszystkie wejścia, wyjścia itp. numerowane są od 1. Wyjątkiem jest numeracja kanałów CTC (ze względu na odniesienie do dokumentacji układu 8253/8254).

3.3. Komunikacja z driver'em.

Funkcje driver'a wywoływane są poprzez przerwanie programowe 98_{16} / 152_{10}. Przesyłanie informacji pomiędzy programem użytkowym a driver'em odbywa się poprzez rekord opisu zlecenia, którego adres przekazywany jest w rejestrach DX:DI. Rekord ten służy do przekazywania informacji zarówno do jak i od driver'a.

Rekord opisu zlecenia ma strukturę zależną od rodzaju zlecenia. Jedynie dwa pierwsze pola są niezmienne i mają następujące znaczenie:

adres rekordu: DX:DI

nazwa	rozmiar w bajtach	znaczenie
LCO_CODE	1	kod funkcji
LCO_STATUS	1	kod zakończenia funkcji

LCO_CODE określa funkcję jaka ma być wykonana przez driver a zarazem sposób interpretacji ciągu bajtów znajdujących się pod adresem DX:DI.

LCO_STATUS informuje program wywołujący driver o poprawności wykonania funkcji:

LCO_STATUS = 0: wykonanie poprawne
LCO_STATUS < 0: wykonanie błędne
LCO_STATUS > 0: wykonanie poprawne z zastrzeżeniami (ostrzeżenia)

W rozdziale 5 podano tabelę wszystkich kodów zwracanych przez LCO_STATUS.

4. Funkcje driver'a.

W poniższych rozdziałach opisano wszystkie funkcje driver'a. Każdy rozdział ma następującą strukturę:

- tabela zawierająca strukturę rekordu opisu zlecenia; w tabeli tej opisano każde pole rekordu w sposób następujący:
- nazwa pola; nazwa ta używana jest konsekwentnie w plikach źródłowych dotyczących języka C, Pascal i asemblera (patrz rozdział 6)
- rozmiar w bajtach; typ danej reprezentowanej przez to pole (np. czy jest to liczba ze znakiem czy bez) wynika ze znaczenia pola; w razie wątpliwości należy porównać z odpowiednim dla danego języka plikiem źródłowym deklarującym struktury danych (dodatki A, C i E)
- znaczenie
- przeznaczenie funkcji
- szczegółowy opis parametrów funkcji (pól rekordu opisu zlecenia); ten punkt został zamieszczony tylko wtedy, gdy uznano, że znaczenie parametru podane w tabeli jest niewystarczająco oczywiste
- ostrzeżenia; lista ostrzeżeń zwracanych przez funkcję w parametrze LCO_STATUS; jeżeli punkt ten nie występuje to oznacza to, że dana funkcja nie zwraca żadnych ostrzeżeń
- błędy; lista błędów zwracanych przez funkcję w parametrze LCO_STATUS; jeżeli punkt ten nie występuje to oznacza to, że dana funkcja nie zwraca żadnych błędów

4.1. Inicjalizacja (MODULE_INIT_55).

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LCO_CODE	1	kod funkcji = 0
LCO_STATUS	1	kod zakończenia funkcji
LCO_IMODULE	1	mapa modułów

Przeznaczenie:

Zainicjalizowanie pracy (zerowanie) wyspecyfikowanych modułów. Moduły podlegające inicjalizacji określa się na bitach parametru LCO_IMODULE:

b8 b1
- - - - D C B A
0 0 0 0 x x x x
x = 1 - zeruj moduł, x = 0 - nie zeruj modułu

W trakcie inicjalizacji wykonywane są następujące czynności:
- porty ustawiane są w kierunku "wejście" (wszystkie typy modułów)
- przerwania są gaszone i maskowane (LC-055-PIO i LC-055-DCU); dodatkowo dla LC-055-DCU aktywne zbocze przerwania ustawiane jest na ujemne
- blokowane są kanały CTC (LC-055-PIO - przez zaprogramowanie wszystkich kanałów na pracę w trybie 3 bez załadowania licznika, dla LC-055-DCU - przez bramkowanie wszystkich kanałów)

Ostrzeżenia:

LCO_NON_EX_MOD - zażądano inicjalizacji nie istniejących modułów ale co najmniej 1 moduł został zainicjalizowany

Błędy:

LCO_NO_MODULE - nie istnieje żaden z żądanych modułów

4.2. Informacja o konfiguracji ogólnej (GET_TOTAL_CONFIGURATION_55).

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LCO_CODE	1	kod funkcji = 1
LCO_STATUS	1	kod zakończenia funkcji = LCO_OK
parametry wyjściowe:		
LCO_TONF	1	konfiguracja modułów
LCO_TCTC	1	liczba dostępnych kanałów CTC
LCO_TPORT	1	liczba dostępnych portów

Przeznaczenie:

Poinformowanie o sumarycznej konfiguracji wszystkich modułów. Bajt konfiguracji modułów ma następujący format:

b8 b1
- - - - D C B A
0 0 0 0 x x x x

x = 1 - moduł zainstalowany, x = 0 - modułu nie ma

4.3. Informacja o konfiguracji modułu (GET_MODULE_CONFIGURATION_55).

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LCO_CODE	1	kod funkcji = 2
LCO_STATUS	1	kod zakończenia funkcji
LCO_MMODULE	1	numer modułu
parametry wyjściowe:		
LCO_MTYPE	1	typ modułu

LCO_MBASE	2	adres bazowy modułu
LCO_MPORT	1	liczba dostępnych portów
LCO_MINTSRC	2	źródło przerwania
LCO_MINTLEV	1	poziom przerywający
LCO_MCTC	1	liczba kanałów CTC
LCO_MCLOCK	12	tablica częstotliwości zegara CTC
LCO_MINT	1	numer przerwania (programowy)

Przeznaczenie:

Poinformowanie o konfiguracji wyspecyfikowanego modułu.

Znaczenie parametrów:

a. LCO_MTYPE
Kod typu badanego modułu:

kod	nazwa	typ modułu
1	LC_8255	LC-8255
2	LC_055_PIO	LC-055-PIO
3	LC_055_DCU	LC-055-DCU

b. LCO_MPORT
Liczba dostępnych równoległych, 8-bitowych portów cyfrowych wejściowo - wyjściowych. Parametr dla modułu LC-055-PIO przybiera zawsze wartość 6, natomiast dla modułów LC-8255 i LC-055-DCU 3 lub 6 - zależnie od liczby zainstalowanych układów 8255.

c. LCO_MINTSRC
Na poszczególnych bitach parametru LCO_MINTSRC zakodowane są źródła przerwań związane z poszczególnymi liniami przerywającymi modułu (moduł LC-055-DCU ma 4 linie przerywające, pozostałe moduły - tylko 1 linię):

nr bitów	nr linii przerywającej	uwagi
1,2	INT 1	------
3,4	INT 2	tylko LC-055-DCU
5,6	INT 3	tylko LC-055-DCU
7,8	INT 4	tylko LC-055-DCU

Kody źródeł przerwań:

kod	nazwa	źródło przerwania	uwagi
0	LCO_NO_INT	brak przerwania	------
1	LCO_INTSRC_EXT	sygnał zewnętrzny	------
2	LCO_INTSRC_CTC	układ CTC	tylko LC-055-DCU

W module LC-055-DCU poszczególne kanały CTC są jednoznacznie przyporządkowane liniom przerywającym:

kanał CTC	nr linii przerywającej
0	INT 2
1	INT 3
2	INT 4

d. LCO_MINTLEV

Na poszczególnych bitach podany jest poziom sygnału przychodzącego do linii przerywającej, który powoduje przerwanie:

moduł LC-8255:

b8 ... b1

- - - - - - - x

moduł LC-055-PIO:

przerwanie generowane zawsze niskim poziomem

moduł LC-055-DCU:

b8 ... b1

- - - - 4 3 2 1 numer linii przerywającej

- - - - x x x x

x = 1 - poziom wysoki, x = 0 - poziom niski

e. LCO_MCTC

Liczba dostępnych kanałów CTC. Dla modułów LC-055-PIO i LC-055-DCU 0 (CTC niezainstalowane) lub 3. Dla modułu LC-8255 zawsze 0 (moduł nie posiada układu CTC).

f. LCO_MCLOCK

Tablica (6 liczb całkowitych) określająca częstotliwość przebiegu zegarowego podanego na poszczególne kanały układu CTC (w kHz). Dla modułu LC-055-PIO wszystkie elementy tablicy mają tę samą wartość (przebieg zegarowy jest wspólny dla wszystkich kanałów): 500, 2000 lub 4000 kHz. Dla modułu LC-055-DCU przebieg zegarowy jest ustawiany niezależnie dla każdego kanału: 500 lub 1000 kHz.

Wartość 0 oznacza przyłączenie odpowiedniego kanału CTC do zegara zewnętrznego.

Dla modułu LC-8255 zawartość tablicy jest nieokreślona (moduł nie posiada układu CTC).

Tablica przeznaczona jest perspektywicznie dla 6 kanałów CTC.

g. LCO_MINT

Parametr podaje programowy numer przerwania związanego z badanym modułem:

nr sprzętowy	nr programowy	uwagi
2 (IRQ2)	10	AT - kaskada 8259
3 (IRQ3)	11	COM2:
4 (IRQ4)	12	COM1:
5 (IRQ5)	13	XT - dysk twardy; AT - LPT2:

Błędy:

LCO_NO_MODULE - nie ma takiego modułu

4.4. Obsługa portów.

4.4.1. Programowanie kierunku pracy (PORT_DIRECTION).

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LCO_CODE	1	kod funkcji = 3
LCO_STATUS	1	kod zakończenia funkcji
LCO_PMODULE	1	numer modułu
LCO_PDIR	2	zakodowane kierunki pracy portów

Przeznaczenie:

Funkcja służy do zaprogramowania kierunku pracy portów równoległych. Kierunek pracy portu kodowany jest na dwóch bitach:

nr bitu	nazwa portu	nr portu
1, 2	A	1
3, 4	B	2
5, 6	C	3
7, 8	D	4
9,10	E	5
11,12	F	6

Kierunek jest zakodowany na dwóch bitach:

nr bitu	wartość	nazwa	znaczenie
1	0	LCO_IN_DIRECTION	port wejściowy
	1	LCO_OUT_DIRECTION	port wyjściowy
2	0	-----	nie zmieniaj kierunku
	1	LCO_CHANGE_DIR	zmień kierunek

Błędy:

LCO_NO_MODULE - dany moduł nie istnieje

4.4.2. Zapis do portu (PORT_WRITE).

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LCO_CODE	1	kod funkcji = 4
LCO_STATUS	1	kod zakończenia funkcji
LCO_PWMODULE	1	numer modułu
LCO_PWPORT	1	numer portu
LCO_PWDATA	1	dana do wysłania

Przeznaczenie:

Bajt LCO_PWDATA jest wysyłany do zadanego portu.

Błędy:

LCO_NO_MODULE - dany moduł nie istnieje
LCO_NONEX_DEV - port o danym numerze nie istnieje
LCO_NOT_OUTPORT - dany port nie jest zaprogramowany jako wyjściowy

4.4.3. Odczyt z portu (PORT_READ).

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LCO_CODE	1	kod funkcji = 5
LCO_STATUS	1	kod zakończenia funkcji

LCO_PRMODULE	1	numer modułu
LCO_PRPORT	1	numer portu
LCO_PRLATCH	1	zatrzaśnięcie danych przed odczytem
parametry wyjściowe:		
LCO_PRDATA	1	odczytany bajt

Przeznaczenie:

Zadany port jest odczytywany, odczytany bajt umieszczany jest w polu LCO_PRDATA.

W przypadku modułu LC-055-PIO dodatkowo istotny jest parametr LCO_PRLATCH. Jeżeli ma on wartość 0 to odczyt przebiega standardowo. Natomiast jeżeli ma wartość 1 to na czas odczytu dane są zatrzaskiwane w rejestrze typu "latch". Zatrzaśnięcie to jest efektywne tylko wtedy gdy linia strobująca GATE jest podłączona do "1" (patrz również dokumentacja techniczno-ruchowa i opis funkcji PORT_LATCH).

Ostrzeżenia:

LCO_NO_LATCH_WARN - dany moduł nie ma możliwości zatrzaskiwania danych - odczytano w sposób standardowy

Błędy:

LCO_NO_MODULE - dany moduł nie istnieje
LCO_NONEX_DEV - port o danym numerze nie istnieje
LCO_NOT_INPORT - dany port nie jest zaprogramowany jako wejściowy

4.4.4. Zatrzaśnięcie danych (PORT_LATCH).

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LCO_CODE	1	kod funkcji = 6
LCO_STATUS	1	kod zakończenia funkcji
LCO_PLMODULE	1	numer modułu
LCO_PLMODE	1	tryb pracy

Przeznaczenie:

Sterowanie zatrzaskiwaniem danych w rejestrze typu "latch". Zrealizowana tylko dla modułu LC-055-PIO.

Działanie funkcji określone jest precyzyjnie przez tryb pracy funkcji LCO_PLMODE i bieżący stan linii GATE modułu:

LCO_PLMODE	nazwa	linia GATE	znaczenie
1	LCO_LATCH_ON	1	zatrzaśnęcie danych w rejestrze
		0	zezwolenie na zatrzaśnięcie linią GATE
0	LCO_LATCH_OFF	1	zwolnienie rejestru do dalszej pracy
		0	zakaz zatrzaskiwania danych linią GATE

Błędy:

LCO_NO_MODULE - dany moduł nie istnieje
LCO_NONEX_DEV - port o danym numerze nie istnieje
LCO_NOT_INPORT - dany port nie jest portem wejściowym
LCO_NO_LATCH - dany moduł nie ma możliwości zatrzaskiwania danych

4.4.5. Zerowanie (PORT_RESET).

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LCO_CODE	1	kod funkcji = 7
LCO_STATUS	1	kod zakończenia funkcji
LCO_PRSMODULE	1	numer modułu

Przeznaczenie:

Wszystkie porty modułu ustawiane są jako wejściowe. W przypadku modułów LC-8255 i LC-055-PIO wykonywane jest to programowo, w przypadku modułu LC-055-DCU wykorzystywany jest odpowiedni sygnał sprzętowy.

W przypadku modułu LC-055-PIO funkcja dodatkowo zwalnia rejestry do dalszej pracy (odpowiednik funkcji PORT_LATCH z trybem pracy LCO_PLMODE = 0);

Błędy:

LCO_NO_MODULE - dany moduł nie istnieje

4.5. Obsługa układu czasowego 8253.

4.5.1. Programowanie i/lub ładowanie licznika (CTC_WRITE_55).

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LCO_CODE	1	kod funkcji = 8
LCO_STATUS	1	kod zakończenia funkcji
LCO_CMODULE	1	numer modułu
LCO_CMODE	1	tryb pracy funkcji
LCO_CCTRL	1	wartość bajtu sterującego
LCO_CWDATA	2	wartość licznika

Przeznaczenie:

Przeznaczeniem funkcji jest zapisanie trybu pracy licznika i/lub jego nowej wartości. Zależy to od ustawionych bitów trybu pracy LCO_CMODE:

nr bitu	nazwa	znaczenie
1	LCO_SET_CTC_MODE	zaprogramuj tryb pracy kanału
2	LCO_SET_COUNTER_VALUE	załaduj nową wartość licznika

Pozostałe bity są ignorowane.

Błędy:

LCO_NO_MODULE - dany moduł nie istnieje
LCO_BAD_CTC_MODE - błędny parametr LCO_CCTRL (część dotycząca trybu pracy kanału CTC)

LCO_NONEX_DEV - w parametrze LCO_CCTRL podano numer nieistniejącego kanału CTC
LCO_CTC_NOT_PROGRAMMED - zlecono zapis wartości licznika lecz nie zaprogramowano trybu pracy kanału
LCO_NOT_SUPPORTED - funkcja nie jest realizowana (dla modułu LC-8255)

4.5.2. Bramkowanie (CTC_GATE).

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LCO_CODE	1	kod funkcji = 9
LCO_STATUS	1	kod zakończenia funkcji
LCO_CGMODULE	1	numer modułu
LCO_CGGATE	1	maska aktywnych kanałów

Przeznaczenie:

Zezwolenie na pracę lub zakazanie pracy poszczególnych kanałów CTC. Funkcja jest dostępna tylko dla modułu LC-055-DCU.

Znaczenie poszczególnych bitów parametru LCO_CGATE:

b_8 b_1
- - - - - 3 2 1
- - - - - x x x
x = 0 - kanał aktywny, x = 1 - kanał nieaktywny (bramkowany)

Błędy:

LCO_NO_MODULE - dany moduł nie istnieje
LCO_NOT_SUPPORTED - funkcja nie jest realizowana (dla modułów LC-8255 i LC-055-PIO)

4.5.3. Odczyt (CTC_READ_55).

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LCO_CODE	1	kod funkcji = 10
LCO_STATUS	1	kod zakończenia funkcji
LCO_CRMODULE	1	numer modułu
LCO_CRCOUNT	1	numer kanału CTC (0, 1, 2)
parametry wyjściowe:		
LCO_CRDATA	2	odczytana wartość licznika

Przeznaczenie:

Funkcja odczytuje bieżącą wartość zadanego kanału CTC.

Błędy:

LCO_NO_MODULE - dany moduł nie istnieje
LCO_NONEX_DEV - podano numer nieistniejącego kanału CTC
LCO_NOT_SUPPORTED - funkcja nie jest realizowana (dla modułu LC-8255)

4.6. Obsługa przerwań.

Poniżej opisano zestaw funkcji driver'a umożliwiających wykorzystywanie przerwań generowanych przez moduł cyfrowy. Na początek kilka ogólnych informacji opisujących ideologię obsługi przerwań przez driver.

Do obsługi przerwania wykorzystywane są zawsze dwie procedury:

- procedura przyjęcia przerwania - dostarczana przez driver
- procedura wykonawcza - dostarczana przez program użytkowy

Procedura przyjęcia przerwania jest procedurą wewnętrzną driver'a i jej zadaniem jest:

- odłożenie na stos rejestrów,
- przełączenie na własny stos o długości 128 bajtów,
- wpisanie do rejestru DS właściwej wartości (jest to wartość, jaką miał ten rejestr w momencie wywoływania przez program użytkowy funkcji INTERRUPT_SERVICE; należy zwrócić na to uwagę, jeżeli wywołuje się driver nie bezpośrednio lecz za pośrednictwem procedur bibliotecznych - patrz przykłady w dodatkach B i D),
- wywołanie procedury wykonawczej,
- powiadomienie kontrolera przerwań o obsłużeniu przerwania,
- przywrócenie poprzedniego stosu i wartości rejestrów

Procedura wykonawcza musi być procedurą daleką, tzn kończyć się instrukcją "retf". W przypadku języka C można to osiągnąć przez skompilowanie programu w modelu "large" lub zadeklarowanie procedury jako "far". W przypadku Turbo Pascala należy użyć dyrektywy kompilatora $F+. Procedura ta musi być procedurą bez parametrów.

Uwaga dotycząca programów napisanych w języku C:
W związku z tym, że driver przełącza stos, należy zwrócić uwagę na model pamięci, w jakim skompilowany jest program. W modelach "tiny", "small" i "compact" zakłada się, że dane i stos znajdują się w tym samym segmencie. Jeżeli więc program jest skompilowany w jednym z powyższych modeli pamięci to procedura wykonawcza nie może korzystać z żadnych funkcji bibliotecznych. W przypadku modeli "medium", "large" i "huge" ograniczenie to oczywiście nie występuje.

W przypadku modułów LC-055-PIO i LC-055-DCU procedura wykonawcza obowiązana jest wykonać funkcję driver'a INTERRUPT_RESET. W przypadku modułu LC-055-DCU powinna przedtem odczytać status modułu (funkcja READ_STATUS) w celu stwierdzenia przyczyny przerwania.

Jeżeli przerwanie przyjdzie przed zakończeniem obsługi poprzedniego to jest po prostu ignorowane.

4.6.1. Maskowanie przerwań (INTERRUPT_MASK).

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LCO_CODE	1	kod funkcji = 11
LCO_STATUS	1	kod zakończenia funkcji
LCO_IMMODULE	1	numer modułu
LCO_IMASK	1	maska przerwań

Przeznaczenie:

Zezwolenie / zakaz przerywania przez poszczególne linie przerywające. Po wykonaniu funkcji wykonywane jest zerowanie linii przerywających w module.

Znaczenie poszczególnych bitów parametru LC0_IMASK:

moduł LC-055-PIO:

```
b8              b1
- - - - - - - 1
- - - - - - - x
```

x = 1 - zezwolenie na przerwanie , x = 0 - zakaz przerwania

moduł LC-055-DCU:

```
b8              b1
- - - - 4 3 2 1
- - - - x x x x
```

x = 1 - zezwolenie na przerwanie od danej linii przerywającej
x = 0 - zakaz przerywania przez daną linię przerywającą

Błędy:

LC0_NO_MODULE - dany moduł nie istnieje
LC0_NOT_SUPPORTED - funkcja nie jest realizowana (dla modułu LC-8255)

4.6.2. Poziom sygnału przerywającego (INTERRUPT_LEVEL).

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LC0_CODE	1	kod funkcji = 12
LC0_STATUS	1	kod zakończenia funkcji
LC0_ILMODULE	1	numer modułu
LC0_ILEVEL	1	poziom sygnału przerywającego na poszczególnych liniach

Przeznaczenie:

Zadanie poziomu sygnału przerywającego dla poszczególnych linii przerywających. Tylko dla modułu Lc-055-DCU.

Znaczenie poszczególnych bitów parametru LC0_ILEVEL:

```
b8              b1
- - - - 4 3 2 1
- - - - x x x x
```

x = 1 - poziom wysoki, x = 0 - poziom niski

Błędy:

LC0_NO_MODULE - dany moduł nie istnieje
LC0_NOT_SUPPORTED - funkcja nie jest realizowana (dla modułów LC-8255 i LC-055-PIO)

4.6.3. Obsługa przerwań (INTERRUPT_SERVICE).

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LC0_CODE	1	kod funkcji = 13
LC0_STATUS	1	kod zakończenia funkcji
LC0_ISMODULE	1	numer modułu
LC0_IPROC	4	adres procedury wykonawczej: offset-segment

Przeznaczenie:

Funkcja służy do instalowania i wyinstalowywania procedur obsługi przerwania przychodzącego z danego modułu. Działanie funkcji zależy od wartości parametru LCO_IPROC:

LCO_IPROC	działanie
0:0	wyinstalowanie procedury obsługi przerwania przychodzącego z modułu i zakazanie tego przerwania w kontrolerze przerwań
inne	zainstalowanie procedury obsługi przerwania i zezwolenie na przerwanie w kontrolerze przerwań

Bliższe informacje na temat procedury LCO_IPROC - patrz wstęp do punktu 4.6.

Procedura obsługi przerwania wyinstalowywana jest również przez funkcję LEAVE_DRIVER_55 (patrz).

Błędy:

LCO_NO_MODULE - dany moduł nie istnieje
LCO_BAD_PROC - błędny adres procedury obsługi przerwania
LCO_NO_IRQ - z danym modułem nie jest związane żadne przerwanie
LCO_INTR_INST - procedura obsługi przerwania jest już zainstalowana

4.6.4. Kasowanie zgłoszenia przerwania (INTERRUPT_RESET).

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LCO_CODE	1	kod funkcji = 14
LCO_STATUS	1	kod zakończenia funkcji
LCO_IRMODULE	1	numer modułu

Przeznaczenie:

Zgaszenie przerzutnika zgłoszenia przerwania.

Błędy:

LCO_NO_MODULE - dany moduł nie istnieje
LCO_NOT_SUPPORTED - funkcja nie jest realizowana (dla modułu LC-8255)

4.7. Odczyt statusu (READ_STATUS).

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LCO_CODE	1	kod funkcji = 15
LCO_STATUS	1	kod zakończenia funkcji = LCO_OK
LCO_SMODULE	1	numer modułu
parametry wyjściowe:		
LCO_SDATA	1	odczytany status
LCO_SRESRVD	1	pole zarezerwowane

Przeznaczenie:

Odczytanie statusu - stanu poszczególnych linii.

Znaczenie poszczególnych bitów parametru LC0_SDATA:

nr bitu	nazwa	znaczenie
1	LC0_INT1_INTERRUPTED	stan linii przerywającej INT 1
2	LC0_INT2_INTERRUPTED	stan linii przerywającej INT 2
3	LC0_INT3_INTERRUPTED	stan linii przerywającej INT 3
4	LC0_INT4_INTERRUPTED	stan linii przerywającej INT 4
5	LC0_COMMON_INTERRUPT	suma linii przerywających 1..4
6	LC0_CTC_OUT0	stan wyjścia kanału 0 CTC
7	LC1_CTC_OUT1	stan wyjścia kanału 1 CTC
8	LC0_CTC_OUT2	stan wyjścia kanału 2 CTC

Błędy:

LC0_NO_MODULE - dany moduł nie istnieje
LC0_NOT_SUPPORTED - funkcja nie jest realizowana (dla modułów LC-8255 i LC-055-PIO)

4.8. Zakończenie pracy z driver'em (LEAVE_DRIVER_55).

Rekord opisu zlecenia:

nazwa	rozmiar w bajtach	znaczenie
LC0_CODE	1	kod funkcji = 16
LC0_STATUS	1	kod zakończenia funkcji = LC0_OK

Przeznaczenie:

Funkcja, którą należy wywołać przed zakończeniem programu. Powoduje "zapomnienie" przez driver wszystkiego co zostało mu przekazane w trakcie pracy programu. W szczególności wyinstalowywuje procedury obsługi przerwań zadeklarowane dla poszczególnych modułów za pomocą funkcji INTERRUPT_SERVICE (patrz).

5. Zestawienie kodów zakończenia funkcji.

nazwa	kod	znaczenie
LC0_OK	0	poprawne zakończenie funkcji

Błędy:

nazwa	kod	znaczenie
LC0_UNKN_FUNC	-1	nieznany kod funkcji
LC0_NO_MODULE	-2	nie istnieje żaden z żądanych modułów; nie ma takiego modułu
LC0_NOT_OUTPORT	-3	port nie jest zaprogramowany jako wyjściowy
LC0_NONEX_DEV	-4	nie istnieje urządzenie o tym numerze
LC0_NOT_OUTPORT	-5	port nie jest zaprogramowany jako wejściowy

LCO_NO_LATCH	-7	dany moduł nie ma możliwości zatrzaskiwania danych
LCO_BAD_PROC	-19	błędny adres procedury obsługi przerwania (spoza pamięci podstawowej)
LCO_NOT_SUPPORTED	-24	dla danego modułu funkcja nie jest realizowana
LCO_BAD_CTC_MODE	-25	błędny tryb pracy CTC
LCO_NO_IRQ	-29	z danym modułem nie jest związane żadne przerwanie
LCO_NOT_FULLY_SUP	-30	żądany tryb wykonania funkcji nie jest dla danego typu modułu realizowany lub jest w opracowaniu
LCO_INTR_INST	-33	procedura pbsługi przerwania jest już zainstalowana
LCO_CTC_NOT_PROGRAMMED	-37	zażądano zapisu do niezaprogramowanego kanału CTC
LCO_REJECTED	-38	za duzo równoczesnych odwołań do driver'a

Ostrzeżenia:

nazwa	kod	znaczenie
LCO_NON_EX_MOD	1	zażądano inicjalizacji nie istniejących modułów ale co najmniej 1 moduł został zainicjalizowany
LCO_NO_LATCH_WARN	2	dany moduł nie ma możliwości zatrzaskiwania danych

6. Projektowanie programów użytkowych.

W poniższym rozdziale zostanie omówiony sposób komunikacji programów użytkowych z driver'em. Na wstępie opisane zostaną ogólne zasady komunikacji a w dalszych podrozdziałach - komunikacja z driver'em z poziomu programów napisanych w C, Pascalu i asemblerze.

Wykonanie dowolnej z funkcji driver' wymaga następujących czynności:

- wypełnienie odpowiedniego rekordu opisu zlecenia; odpowiednie struktury danych dostarczane są przez producenta w postaci zbiorów źródłowych
- wpisanie adresu rekordu do rejestrów DX i DI
- wykonanie przerwania programowego 98_{16} / 152_{10}

Dobrze napisany program powinien składać się z następujących części:

- część wstępna:
- stwierdzenie, czy driver jest zainstalowany; metodą wykrycia obecności driver'a w pamięci systemu jest próba otwarcia urządzenia o nazwie określonej przez driver (LCO55^^^, patrz p.3.2.); powodzenie tej próby świadczy o zainstalowamiu driver'a, niepowodzenie - o jego braku
- rozpoznanie konfiguracji modułu (GET_TOTAL_CONFIGURATION_55 - ile i jakich modułów jest zainstalowanych, GET_MODULE_CONFIGURATION_55 - czy moduł ma zainstalowany układ CTC, czy jest podłączony do któregoś z przerwań itp., ten etap jest szczególnie ważny, gdy projektowanyjest program uniwersalny, mający operować na kilku rodzajach modułów
- inicjalizacja modułów;

- część wykonawcza:
- tu powinny się znaleźć funkcje wykonujące właściwe operacje modułu; należy zwrócić uwagę na to, że każda funkcja może dostać błędne parametry i zasygnalizować to w kodzie odpowiedzi (LCO_STATUS); należy koniecznie sprawdzać tę odpowiedź, szczególnie w dwóch sytuacjach: gdy program jest na etapie uruchamiania i gdy parametry funkcji są dostarczane interakcyjnie przez użytkownika
- część końcowa:
- przed zakończeniem programu należy koniecznie wykonać funkcję LEAVE_DRIVER_55; jest to szczególnie ważne wtedy, gdy z komputera korzysta kilku użytkowników posługujących się różnymi programami korzystającymi z driver'a.

6.1. Programowanie w języku C.

Z poziomu języka C komunikacja z driver'em odbywa się za pośrednictwem procedur bibliotecznych **int86** lub **int86x**. W przypadku implementacji firmy Borland (Turbo C) można korzystać również z pseudozmiennych _DX i _DI oraz makroinstrukcji **geninterrupt** (dos.h). Jeżeli program korzysta z przerwań to należy pomiętać o zależności od modelu pamięci (patrz wstęp do punktu 4.6.).

Producent dostarcza dwa pliki źródłowe:

- AMBEX-55.H: definicje wszystkich struktur danych i stałych potrzebnych do współpracy z driver'em
- TEST55.C: program przykładowy

W Dodatku A znajduje się wydruk pliku AMBEX-55.H, natomiast w Dodatku B - wydruk pliku TEST55.C.

6.2. Programowanie w języku Pascal.

Z poziomu języka Pascal komunikacja z driver'em odbywa się za pośrednictwem procedury bibliotecznej **intr**. Jeżeli program korzysta z przerwań to należy pomiętać o odpowiednim zastosowaniu dyrektywy kompilatora $F (Turbo Pascal - patrz wstęp do punktu 4.6.).

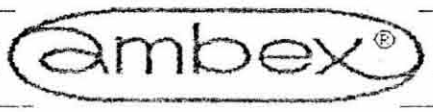
Producent dostarcza dwa pliki źródłowe:

- AMBEX-55.PAS: definicje wszystkich struktur danych i stałych potrzebnych do współpracy z driver'em
- TEST55.PAS: program przykładowy

W Dodatku C znajduje się wydruk pliku AMBEX-55.PAS, natomiast w Dodatku D - wydruk pliku TEST55.PAS.

6.3. Programowanie w języku assemblera.

Producent dostarcza pliki źródłowy AMBEX-55.ASM zawierający definicje wszystkich struktur danych i stałych potrzebnych do współpracy z driver'em. Wydruk tego pliku znajduje się w Dodatku E.



D O D A T E K A

do dokumentacji driver´ów modułów

LC-055-PIO, LC-055-DCU i LC-8255

AMBEX-55.H

struktury danych i stałe dla języka C

Wydanie: lipiec 1992

```
/*****************************************************************************/
/*                                                                           */
/*   Zestaw deklaracji struktur danych i definicji stalych do komunikacji    */
/*   z driver'em LC055.DRV z poziomu jezyka C.                               */
/*                                                                           */
/*****************************************************************************/

#define LC0_MAX_CTC             6       /* maksymalna liczba kanalow CTC    */
#define LC0_MAX_PORTS           6       /* maksymalna liczba portow         */
#define LC0_MAX_INT_LINES       4       /* maksymalna liczba linii          */
                                        /* przerywajacych                   */

/*************** REKORDY OPISOW ZLECEN FUNKCJI DRIVER'A ******************/

/* Rekord opisu zlecenia funkcji  MODULE_INIT_55 ========================== */
struct lc0_init_55
        {
        unsigned char   LC0_CODE;       /* numer funkcji (0)                */
        char            LC0_STATUS;     /* kod odpowiedzi driver'a          */
        unsigned char   LC0_IMODULE;    /* mapa modulow                     */
        };

/* Rekord opisu zlecenia funkcji  GET_TOTAL_CONFIGURATION_55 ============== */
struct lc0_total_55
        {
        unsigned char   LC0_CODE;       /* numer funkcji (1)                */
        char            LC0_STATUS;     /* kod odpowiedzi driver'a          */
        unsigned char   LC0_TONF;       /* inf. o zainstalowanych modulach  */
        unsigned char   LC0_TCTC;       /* liczba dostepnych kanalow CTC    */
        unsigned char   LC0_TPORT;      /* liczba dostepnych portow         */
        };

/* Rekord opisu zlecenia funkcji   GET_MODULE_CONFIGURATION_55 ============ */
struct lc0_module_55
        {
        unsigned char   LC0_CODE;       /* numer funkcji (2)                */
        char            LC0_STATUS;     /* kod odpowiedzi driver'a          */
        unsigned char   LC0_MMODULE;    /* numer modulu                     */
        unsigned char   LC0_MTYPE;      /* typ modulu                       */
        unsigned int    LC0_MBASE;      /* adres bazowy rejestrow modulu    */
        unsigned char   LC0_MPORT;      /* liczba dostepnych portow         */
        unsigned char   LC0_MCTC;       /* liczba dostepnych kanalow CTC    */
        unsigned int    LC0_MCLOCK[LC0_MAX_CTC];    /* tabl. czestotliwosci*/
                                                    /* zegara CTC w kHz    */
        unsigned char   LC0_CTCGATE;    /* bramki kanalow CTC               */
        unsigned char   LC0_MINT;       /* numer przerwania (programowy)    */
        unsigned int    LC0_MINTSRC;    /* okreslenie zrodel przerwan       */
        unsigned char   LC0_MINTLEV;    /* poziom przerywajacy              */
        unsigned char   LC0_MINTMSK;    /* maski przerwan                   */
        };
```

```
/* Rekord opisu zlecenia funkcji   PORT_DIRECTION ======================== */
struct lc0_port_dir
        {
        unsigned char   LC0_CODE;       /* numer funkcji (3)              */
        char            LC0_STATUS;     /* kod odpowiedzi driver'a        */
        unsigned char   LC0_PMODULE;    /* numer modulu                   */
        unsigned long   LC0_PDIR;       /* kierunki pracy portow          */
        };
/* Rekord opisu zlecenia funkcji   PORT_WRITE ========================== */
struct lc0_port_write
        {
        unsigned char   LC0_CODE;       /* numer funkcji (4)              */
        char            LC0_STATUS;     /* kod odpowiedzi driver'a        */
        unsigned char   LC0_PWMODULE;   /* numer modulu                   */
        unsigned char   LC0_PWPORT;     /* numer portu                    */
        unsigned char   LC0_PWDATA;     /* dana do wyslania               */
        };

/* Rekord opisu zlecenia funkcji   PORT_READ =========================== */
struct lc0_port_read
        {
        unsigned char   LC0_CODE;       /* numer funkcji (5)              */
        char            LC0_STATUS;     /* kod odpowiedzi driver'a        */
        unsigned char   LC0_PRMODULE;   /* numer modulu                   */
        unsigned char   LC0_PRPORT;     /* numer portu                    */
        unsigned char   LC0_PRLATCH;    /* LC0_LATCH_ON/LC0_LATCH_OFF     */
        unsigned char   LC0_PRDATA;     /* dana odczytana                 */
        };

/* Rekord opisu zlecenia funkcji   PORT_LATCH ========================== */
struct lc0_port_latch
        {
        unsigned char   LC0_CODE;       /* numer funkcji (6)              */
        char            LC0_STATUS;     /* kod odpowiedzi driver'a        */
        unsigned char   LC0_PLMODULE;   /* numer modulu                   */
        unsigned char   LC0_PLMODE;     /* tryb pracy                     */
        };

/* Rekord opisu zlecenia funkcji   PORT_RESET ========================== */
struct lc0_port_reset
        {
        unsigned char   LC0_CODE;       /* numer funkcji (7)              */
        char            LC0_STATUS;     /* kod odpowiedzi driver'a        */
        unsigned char   LC0_PRSMODULE;  /* numer modulu                   */
        };

/* Rekord opisu zlecenia funkcji   CTC_WRITE_55 ======================== */
struct lc0_ctc_write_55
        {
        unsigned char   LC0_CODE;       /* numer funkcji (8)              */
        char            LC0_STATUS;     /* kod odpowiedzi driver'a        */
        unsigned char   LC0_CMODULE;    /* numer modulu                   */
        unsigned char   LC0_CMODE;      /* tryb pracy funkcji             */
        unsigned char   LC0_CCTRL;      /* bajt sterujacy                 */
        unsigned int    LC0_CWDATA;     /* wartosc licznika               */
        };
```

```
/* Rekord opisu zlecenia funkcji   CTC_GATE ================================ */
struct lc0_ctc_gate
        {
        unsigned char   LC0_CODE;       /* numer funkcji (9)                 */
        char            LC0_STATUS;     /* kod odpowiedzi driver'a           */
        unsigned char   LC0_CGMODULE;   /* numer modulu                      */
        unsigned char   LC0_CGGATE;     /* maska aktywnych kanalow           */
        };

/* Rekord opisu zlecenia funkcji   CTC_READ_55 ============================= */
struct lc0_ctc_read_55
        {
        unsigned char   LC0_CODE;       /* numer funkcji (10)                */
        char            LC0_STATUS;     /* kod odpowiedzi driver'a           */
        unsigned char   LC0_CRMODULE;   /* numer modulu                      */
        unsigned char   LC0_CRCOUNT;    /* numer kanalu                      */
        unsigned int    LC0_CRDATA;     /* odczytana dana                    */
        };

/* Rekord opisu zlecenia funkcji   INTERRUPT_MASK ========================== */
struct lc0_int_mask
        {
        unsigned char   LC0_CODE;       /* numer funkcji (11)                */
        char            LC0_STATUS;     /* kod odpowiedzi driver'a           */
        unsigned char   LC0_IMMODULE;   /* numer modulu                      */
        unsigned char   LC0_IMASK;      /* maska przerwan                    */
        };

/* Rekord opisu zlecenia funkcji   INTERRUPT_LEVEL ========================= */
struct lc0_int_level
        {
        unsigned char   LC0_CODE;       /* numer funkcji (12)                */
        char            LC0_STATUS;     /* kod odpowiedzi driver'a           */
        unsigned char   LC0_ILMODULE;   /* numer modulu                      */
        unsigned char   LC0_ILEVEL;     /* poziom przerywajacy na            */
                                        /* poszczegolnych liniach            */
        };

/* Rekord opisu zlecenia funkcji   INTERRUPT_SERVICE_55 ==================== */
struct lc0_int_service
        {
        unsigned char   LC0_CODE;       /* numer funkcji (13)                */
        char            LC0_STATUS;     /* kod odpowiedzi driver'a           */
        unsigned char   LC0_ISMODULE;   /* numer modulu                      */
        void    far (*LC0_IPROC)(void); /* adres procedury obslugi           */
        };

/* Rekord opisu zlecenia funkcji   INTERRUPT_RESET ========================= */
struct lc0_int_reset
        {
        unsigned char   LC0_CODE;       /* numer funkcji (14)                */
        char            LC0_STATUS;     /* kod odpowiedzi driver'a           */
        unsigned char   LC0_IRMODULE;   /* numer modulu                      */
        };
```

```
/* Rekord opisu zlecenia funkcji   READ_STATUS ============================ */
struct lc0_read_status
        {
        unsigned char   LC0_CODE;       /* numer funkcji (15)              */
        char            LC0_STATUS;     /* kod odpowiedzi driver'a         */
        unsigned char   LC0_SMODULE;    /* numer modulu                    */
        unsigned char   LC0_SDATA;      /* odczytany status                */
        unsigned char   LC0_SRESRVD;    /* pole zarezerwowane              */
        };

/* Rekord opisu zlecenia funkcji  LEAVE_DRIVER_55 ======================== */
struct lc0_leave_55
        {
        unsigned char   LC0_CODE;       /* numer funkcji (16)              */
        char            LC0_STATUS;     /* kod odpowiedzi driver'a         */
        };

#define LC055   0x98    /* numer przerwania obslugiwanego przez driver     */

/* ****************** kody funkcji driver'a ********************************/
#define MODULE_INIT_55                  0       /* inicjalizacja modulow   */
#define GET_TOTAL_CONFIGURATION_55      1       /* odczyt konfig. modulow  */
#define GET_MODULE_CONFIGURATION_55     2       /* odczyt konfig. modulu   */
#define PORT_DIRECTION                  3       /* ustawienie kierunku     */
                                                /* pracy portow            */
#define PORT_WRITE                      4       /* zapis do portu          */
#define PORT_READ                       5       /* odczyt portu            */
#define PORT_LATCH                      6       /* zatrzasniecie danych    */
#define PORT_RESET                      7       /* zerowanie ukladow 8255  */
                                                /* (ustawienie wszystkich  */
                                                /* portow w kierunku IN)   */
#define CTC_WRITE_55                    8       /* zaprogramowanie i/lub   */
                                                /* zapisanie wartosci do   */
                                                /* kanalu CTC              */
#define CTC_GATE                        9       /* bramkowanie kanalu CTC  */
#define CTC_READ_55                     10      /* odczyt kanalu CTC       */
#define INTERRUPT_MASK                  11      /* maskowanie przerwan     */
#define INTERRUPT_LEVEL                 12      /* ustawianie poziomu      */
                                                /* przerywajacego          */
#define INTERRUPT_SERVICE_55            13      /* deklarowanie procedury  */
                                                /* obslugi przerwania      */
#define INTERRUPT_RESET                 14      /* gaszenie zgloszenia     */
                                                /* przerwania              */
#define READ_STATUS                     15      /* odczyt statusu          */
#define LEAVE_DRIVER_55                 16      /* zakonczenie pracy z     */
                                                /* driver'em               */


/********************* kody odpowiedzi funkcji driver'a **********************/

#define LC0_OK                  0       /* OK                              */

/* ===============    ostrzezenia ================    */

#define LC0_NON_EX_MOD          1       /* nie istniejacy(e) modul(y)      */
#define LC0_NO_LATCH_WARN       2       /* modul nie ma mozliwosci         */
                                        /* zatrzaskiwania danych           */
```

```
/* ==================    bledy ====================    */

#define LC0_UNKN_FUNC           -1      /* nieznany kod funkcji             */
#define LC0_NO_MODULE           -2      /* brak modulu(ow)                  */
#define LC0_NOT_OUTPORT         -3      /* port nie jest wyjsciowy          */
#define LC0_NONEX_DEV           -4      /* nie istnieje urzadzenie o tym    */
                                        /* numerze                          */
#define LC0_NOT_INPORT          -5      /* port nie jest wejsciowy          */
#define LC0_BAD_COUNTER         -6      /* zly numer kanalu CTC             */
#define LC0_NO_LATCH            -7      /* dany modul nie moze zatrzaskiwac*/
#define LC0_BAD_PROC            -19     /* bledny adres procedury obslugi   */
                                        /* przerwan                         */
#define LC0_NOT_SUPPORTED       -24     /* dla danego modulu funkcja nie    */
                                        /* jest realizowana                 */
#define LC0_BAD_CTC_MODE        -25     /* bledny tryb pracy CTC            */
#define LC0_NO_IRQ              -29     /* z danum modulem nie jest         */
                                        /* zwiazane zadne przerwanie        */
#define LC0_NOT_FULLY_SUP       -30     /* zadany tryb funkcji nie          */
                                        /* jest realizowany dla danego typu*/
                                        /* modulu lub funkcja w opracowaniu*/
#define LC0_INTR_INST           -33     /* proba powtornej instalacji       */
                                        /* przerwania                       */
#define LC0_CTC_NOT_PROGRAMMED  -37     /* zapis wartosci do                */
                                        /* niezaprogramowanego licznika     */
#define LC0_REJECTED            -38     /* za duzo jednoczesnych odwolan do*/
                                        /* driver'a                         */

/************************** kody numerow modulow **************************/
#define LC0_MODA        1       /* modul A                                  */
#define LC0_MODB        2       /* modul B                                  */
#define LC0_MODC        3       /* modul C                                  */
#define LC0_MODD        4       /* modul D                                  */

/************************** maski modulow w mapie *************************/
#define LC0_MODAMAP     1       /* modul A                                  */
#define LC0_MODBMAP     2       /* modul B                                  */
#define LC0_MODCMAP     4       /* modul C                                  */
#define LC0_MODDMAP     8       /* modul D                                  */

/************************** kody typow modulow ****************************/
#define LC_8255         1       /* LC-8255                                  */
#define LC_055_PIO      2       /* LC-055-PIO                               */
#define LC_055_DCU      3       /* LC-055-DCU                               */

/************************** kody zrodla przerwania ************************/
#define LC0_NO_INT      0       /* brak przerwania                          */
#define LC0_INTSRC_EXT  1       /* sygnal zewnetrzny                        */
#define LC0_INTSRC_CTC  2       /* CTC                                      */

/********************** kody kierunkow pracy portow ***********************/
/* bit kierunku : */
#define LC0_IN_DIRECTION        0       /* port wejsciowy                   */
#define LC0_OUT_DIRECTION       1       /* port wyjsciowy                   */
/* bit trybu : */
#define LC0_CHANGE_DIR          2       /* zmien kierunek pracy             */
```

```
/************* PORT_LATCH/LC0_PLMODE, PORT_READ/LC0_PRLATCH ******************/
#define LC0_LATCH_ON     0              /* zatrzasnij                          */
#define LC0_LATCH_OFF    1              /* zwolnij                             */

/*********************** kody trybu pracy funkcji CTC_MODE *******************/
#define LC0_SET_CTC_MODE        1       /* zaprogramuj tryb pracy kanalu      */
#define LC0_SET_COUNTER_VALUE   2       /* zaladuj nowa wartosc licznika      */

/************************ bity statusu (READ_STATUS) *************************/
#define LC0_INT1_INTERRUPTED    1       /* linia INT 1 zglosila przerwanie */
#define LC0_INT2_INTERRUPTED    2       /* linia INT 2 zglosila przerwanie */
#define LC0_INT3_INTERRUPTED    4       /* linia INT 3 zglosila przerwanie */
#define LC0_INT4_INTERRUPTED    8       /* linia INT 4 zglosila przerwanie */
#define LC0_COMMON_INTERRUPT    16      /*jedna z linii zglosila przerwanie*/
#define LC0_CTC_OUT0            32      /* stan linii OUT kanalu 0           */
#define LC1_CTC_OUT1            64      /* stan linii OUT kanalu 1           */
#define LC0_CTC_OUT2            128     /* stan linii OUT kanalu 2           */

/************** wartosci fragmentow bajtu sterujacego 8253/8254 **************/
#define CTC_NB          0       /* kod naturalny binarny                     */
#define CTC_BCD         1       /* kod BCD                                   */

#define CTC_MODE0       0       /* tryb 0                                    */
#define CTC_MODE1       2       /* tryb 1                                    */
#define CTC_MODE2       4       /* tryb 2                                    */
#define CTC_MODE3       6       /* tryb 3                                    */
#define CTC_MODE4       8       /* tryb 4                                    */
#define CTC_MODE5       10      /* tryb 5                                    */

#define CTC_LSB         0x10    /* tylko mlodszy bajt                        */
#define CTC_MSB         0x20    /* tylko starszy bajt                        */
#define CTC_BOTH        0x30    /* mlodszy - starszy                         */

#define CTC_COUNT0      0x00    /* licznik 0                                 */
#define CTC_COUNT1      0x40    /* licznik 1                                 */
#define CTC_COUNT2      0x80    /* licznik 2                                 */
```

ambex®

D O D A T E K B

do dokumentacji driver'a modułów

LC-055-PIO, LC-055-DCU i LC-8255

TEST55.C

przykładowy program w C

Wydanie: lipiec 1992

```
/*****************************************************************************/
/* Program przykladowy pokazujacy sposob wykorzystania drivera modulow       */
/* cyfrowych: LC-8255, LC-055-PIO i LC-055-DCU                               */
/*****************************************************************************/
/* W katalogu zawierajacym standardowe naglowki C musi sie znajdowac plik    */
/* AMBEX-55.H.                                                               */
/*****************************************************************************/
/* Program musi byc kompilowany w modelu "large" lub "compact" ze wzgledu    */
/* na to, ze w trakcie obslugi przerwan driver przelacza stos (dla procedur*/
/* newintPIODCU i newint8255) - patrz dokumentacja driver'a.                 */
/* UWAGA: Przed rozpoczeciem kompilacji nalezy sprawdzic ustawienie opcji    */
/*        kompilatora:                                                       */
/*              - alingment = byte,                                          */
/*              - default char type = signed.                                */
/*****************************************************************************/

#include <ambex-55.h>           /* struktury danych i definicje stalych do */
                                /* komunikacji z driver'em                 */
#include <conio.h>              /* funkcja getch                           */
#include <dos.h>                /* funkcja int86                           */
#include <io.h>                 /* dla funkcji driverinstalled             */
#include <stdio.h>              /* wyswietlanie                            */
#include <stdlib.h>             /* funkcja exit                            */

/********************* deklaracje procedur ***********************************/
void    askdriver(void);
void    checkdriver(void);
void    displayconfiguration(void);
void    displayinterrupts(int x, int y);
void    drivererror(char status);
int     driverinstalled(char *name);
void    initdelay(void);
void    far newint8255(void);
void    far newintPIODCU(void);
void    pressanykey(void);
void    readport(void);
void    test8255int(void);
void    testDCUint(void);
void    testinterrupts(void);
void    testPIOint(void);
void    writeportfail(void);
void    quit(void);

/********************* definicje typow ***************************************/
typedef unsigned char   byte;           /* bajt bez znaku                    */
typedef unsigned int    word;           /* slowo bez znaku                   */

/********************* definicje stalych *************************************/
#define TRUE    1
#define FALSE   0

/********************* deklaracje zmiennych **********************************/
                                /* rekordy opisu poszczegolnych zlecen     */
struct  lc0_init_55             s_init        = {MODULE_INIT_55};
struct  lc0_total_55            s_total       = {GET_TOTAL_CONFIGURATION_55};
struct  lc0_module_55           s_module      = {GET_MODULE_CONFIGURATION_55};
struct  lc0_port_dir            s_port_dir    = {PORT_DIRECTION};
struct  lc0_port_write          s_port_write  = {PORT_WRITE};
```

```
struct  lc0_port_read           s_port_read   = {PORT_READ};
struct  lc0_ctc_write_55        s_ctc_write   = {CTC_WRITE_55};
struct  lc0_ctc_gate            s_ctc_gate    = {CTC_GATE};
struct  lc0_int_mask            s_int_mask    = {INTERRUPT_MASK};
struct  lc0_int_level           s_int_level   = {INTERRUPT_LEVEL};
struct  lc0_int_service         s_int_service = {INTERRUPT_SERVICE_55};
struct  lc0_int_reset           s_int_reset   = {INTERRUPT_RESET};
struct  lc0_leave_55            s_leave       = {LEAVE_DRIVER_55};
union   REGS    r;              /* parametr dla funkcji int86                */
int     modulenum;              /* numer badanego modulu                     */
word    interrupt_counter;      /* licznik przerwan                          */
                                /* ------- teksty bledow ----------------- */
char    *DriverErrors[] =
{
                                        /* LC0_UNKN_FUNC                -1 */
"LC0_UNKN_FUNC: Nieznany kod funkcji",
                                        /* LC0_NO_MODULE                -2 */
"LC0_NO_MODULE: Bledny numer modulu",
                                        /* LC0_NOT_OUTPORT              -3 */
"LC0_NOT_OUTPORT: Port nie jest portem wyjsciowym",
                                        /* LC0_NONEX_DEV                -4 */
"LC0_NONEX_DEV: Bledny numer urzadzenia",
                                        /* LC0_NOT_INPORT               -5 */
"LC0_NOT_INPORT: Port nie jest portem wejsciowym",
"",
                                        /* LC0_NO_LATCH                 -7 */
"LC0_NO_LATCH: Module nie moze zatrzaskiwac danych",
""
"",
"",
"",
"",
"",
"",
"",
"",
"",
"",
"",
"",
                                        /* LC0_BAD_PROC                -19 */
"LC0_BAD_PROC: Bledny adres procedury obslugi przerwania",
""
"",
"",
"",
"",
                                        /* LC0_NOT_SUPPORTED           -24 */
"LC0_NOT_SUPPORTED: Funkcja nie jest realizowana",
                                        /* LC0_BAD_CTC_MODE            -25 */
"LC0_BAD_CTC_MODE: Bledny tryb pracy CTC",
""
"",
"",
"",
                                        /* LC0_NO_IRQ                  -29 */
"LC0_NO_IRQ: Z modulem nie jest zwiazane zadne przerwanie",
                                        /* LC0_NOT_FULLY_SUP           -30 */
"LC0_NOT_FULLY_SUP: Funkcja w opracowaniu",
""
"",
"",
                                        /* LC0_INTR_INST               -33 */
```

```
"LCO_INTR_INST: Proba powtornej instalacji przerwania",
"",
"",
"",
                                        /* LCO_CTC_NOT_PROGRAMMED      -37 */
"LCO_CTC_NOT_PROGRAMMED: Zapis licznika przy niezaprogramowanym trybie pracy",
                                        /* LCO_REJECTED                -38 */
"LCO_REJECTED: Za duzo jednoczesnych odwolan do driver'a",
};
char    *DriverWarnings[] =
{
                                        /* LCO_NON_EX_MOD               1 */
"LCO_NON_EX_MOD: Zazadano inicjalizacji nieistniejacych modulow",
                                        /* LCO_NO_LATCH_WARN            2 */
"LCO_NO_LATCH_WARN: Module nie moze zatrzaskiwac danych",
};

/*****************************************************************************/
/*****************************************************************************/
void main(void)
{
        checkdriver();          /* rozpoznanie zainstalowanego driver'a    */
        initdelay();            /* zainicjalizowanie funkcji delay; istotne*/
                                /*   tylko dla implementacji Turbo C       */
        askdriver();            /* odpytanie driver'a o konfiguracje       */
        displayconfiguration(); /* wyswietlenie konfiguracji badanego      */
                                /* modulu                                  */
        readport();             /* odczyt portu cyfrowego                  */
        writeportfail();        /* zapis na port wejsciowy - blad          */
        testinterrupts();       /* test przerwan przychodzacych z modulu   */
        quit();                 /* zakonczenie programu                    */
}

/*****************************************************************************/
/* Przeznaczenie:                                                            */
/*   Rozpoznanie, czy driver jest zainstalowany.                             */
/* Sposob:                                                                   */
/*   Za pomoca ogolnej funkcji driverinstalled. Jezeli zaden driver nie      */
/*   jest zainstalowany to po wyswietleniu wlasciwego komunikaru program     */
/*   konczy prace.                                                           */
/*****************************************************************************/
void checkdriver(void)
{
        clrscr();
        gotoxy(1, 1);
        if(!driverinstalled("LC055^^^"))
            {
            printf("Driver nie jest zainstalowany!\n");
            exit(1);
            }
}
```

```
/*****************************************************************************/
/* Przeznaczenie:                                                            */
/*   Sprawdzenie obecnosci drivera.                                          */
/* Sposob:                                                                   */
/*   Przez probe otwarcia urzadzenia o nazwie okreslonej przez driver.       */
/* Parametry:                                                                */
/*   name - nazwa urzadzenia                                                 */
/* Wartosc:                                                                  */
/*   TRUE  - driver jest zainstalowany                                       */
/*   FALSE - driver nie jest zainstalowany                                   */
/*****************************************************************************/
int driverinstalled(char *name)
{
        int     hd;

        hd = open(name, 0);
        if(hd == -1)
            return(FALSE);
        else
            {
            close(hd);
            return(TRUE);
            }
}

/*****************************************************************************/
/* Przeznaczenie:                                                            */
/*   Zainicjalizowanie funkcji delay - istotne tylko dla implementacji       */
/*   Turbo C.                                                                */
/*****************************************************************************/
void initdelay(void)
{
        printf("Prosze czekac . . .");
        delay(1);
        clrscr();
        gotoxy(1, 1);
}

/*****************************************************************************/
/* Przeznaczenie:                                                            */
/*   Rozpoznanie konfiguracji badanego modulu.                               */
/* Sposob:                                                                   */
/*   Przez wykorzystanie funkcji driver'a GET_TOTAL_CONFIGURATION,           */
/*   GET_MODULE_CONFIGURATION.                                               */
/* Uwagi:                                                                    */
/*   Funkcja nadaje wartosc zmiennej modulenum (numer badanego modulu)       */
/*   wypelnia struktury total, module, info i inicjalizuje zainstalowane     */
/*   moduly. Jezeli zainstalowanych jest wiecej niz jeden modulow to         */
/*   program bedzie analizowal tylko pierwszy z brzegu.                      */
/*****************************************************************************/
void askdriver(void)
{
        r.x.dx = FP_SEG(&s_total);
        r.x.di = FP_OFF(&s_total);
        int86(LC055, &r, &r);           /* GET_TOTAL_CONFIGURATION_55      */
```

```
                                /* inicjalizacja zainstalowanych modulow   */
        s_init.LC0_IMODULE = s_total.LC0_TONF & 0xF;
        r.x.dx = FP_SEG(&s_init);
        r.x.di = FP_OFF(&s_init);
        int86(LC055, &r, &r);                           /* MODULE_INIT_55  */

                                        /* sprawdzenie, ktory modul jest   */
                                        /*   zainstalowany: A, B, C czy D  */
        for(modulenum = 1; modulenum <= 4; modulenum++)
            if(s_total.LC0_TONF & (1 << (modulenum - 1)))
                break;

                                        /* spytanie o konfiguracje modulu  */
        s_module.LC0_MMODULE = modulenum;
        r.x.dx = FP_SEG(&s_module);
        r.x.di = FP_OFF(&s_module);
        int86(LC055, &r, &r);           /* GET_MODULE_CONFIGURATION_55     */

}

/*****************************************************************************/
/* Przeznaczenie:                                                            */
/*   Wyswietlenie konfiguracji badanego modulu.                              */
/* Sposob:                                                                   */
/*   Przez wykorzystanie informacji zawartych w strukturze s_module.         */
/*****************************************************************************/
void displayconfiguration(void)
{
        int     i;
                                        /* konfiguracja modulu             */
        printf("\n---------- Konfiguracja modulu ");
        switch(s_module.LC0_MTYPE)
            {
            case LC_8255:
                printf("LC-8255");
                break;
            case LC_055_PIO:
                printf("LC-055-PIO");
                break;
            case LC_055_DCU:
                printf("LC-055-DCU");
                break;
            }
        printf(" (%c) :\n\n", 'A' + modulenum - 1);
        printf("Adres modulu: %04X (hex)\n", s_module.LC0_MBASE);
        printf("Liczba portow we/wy: %d\n", s_module.LC0_MPORT);
        printf("Liczba kanalow CTC: %d\n", s_module.LC0_MCTC);
        if(s_module.LC0_MCTC != 0)
            {
            if(s_module.LC0_MTYPE == LC_055_PIO)
                {                               /* wspolna czestotliwosc   */
                if(s_module.LC0_MCLOCK[0] == 0)
                    printf("Wszystkie kanaly CTC podlaczone "
                                "do zegara zewnetrznego\n");
                else
                    printf("Czestotliwosc zegara CTC: %d kHz\n",
                            s_module.LC0_MCLOCK[0]);
                }
```

```
            else                    /* LC-055-DCU gdyz LC-8255 nie ma CTC      */
                {
                        /* indywidualna czestotliwosc dla kazdego kanalu   */
                    for(i = 0; i < s_module.LCO_MCTC; i++)
                        if(s_module.LCO_MCLOCK[i] == 0)
                            printf("Kanal %d CTC podlaczony "
                                                "do zegara zewnetrznego\n", i);
                        else
                            printf("Czestotliwosc zegara kanalu %d CTC: %d kHz\n",
                                    i, s_module.LCO_MCLOCK[i]);
                }
            }
        if(s_module.LCO_MINT == 0xFF)
            printf("Modul nie jest podlaczony do zadnego przerwania\n");
        else
            printf("Numer przerwania zwiazanego z modulem: %d\n",
                    s_module.LCO_MINT);
        pressanykey();          /* oczekiwanie na operatora                */
}

/*************************************************************************/
/* Przeznaczenie:                                                        */
/*   Przerwa miedzy kolejnymi czesciami programu - oczekiwanie na reakcje */
/*   operatora.                                                          */
/*************************************************************************/
void pressanykey(void)
{
        printf("\nNacisnij dowolny klawisz . . .");
        getch();
        clrscr();
        gotoxy(1, 1);
}

/*************************************************************************/
/* Przeznaczenie:                                                        */
/*   Odczyt portu 1 (A) zaprogramowanego jako wejsciowy i wyslanie       */
/*   odczytanej wartosci na port 2 (B) zaprogramowany jako wyjsciowy.    */
/*************************************************************************/
void readport(void)
{
        int     i,
                j;

        printf("---------- "
               "Przesylanie danych z portu wejsciowego na wyjsciowy\n\n");
                            /* zaprogramowanie kierunku pracy portow:  */
                            /*   1 (A) : wejscie                       */
                            /*   2 (B) : wyjscie                       */
                            /*   pozostale - bez zmiany                */
        s_port_dir.LCO_PMODULE = modulenum;
        s_port_dir.LCO_PDIR = (LCO_IN_DIRECTION | LCO_CHANGE_DIR) |
                              ((LCO_OUT_DIRECTION | LCO_CHANGE_DIR) << 2);
        r.x.dx = FP_SEG(&s_port_dir);
        r.x.di = FP_OFF(&s_port_dir);
        int86(LC055, &r, &r);                   /* PORT_DIRECTION          */

                                    /* przygotowanie stalych pol           */
                                    /* rekordow opisow zlecen              */
```

```
        s_port_read.LCO_PRMODULE = modulenum;
        s_port_read.LCO_PRPORT = 1;
        s_port_read.LCO_PRLATCH = LCO_LATCH_OFF;
        s_port_write.LCO_PWMODULE = modulenum;
        s_port_write.LCO_PWPORT = 2;
                                /* 20-krotny odczyt / wyswietlenie / zapis */
        for(i = 0; i < 20; i++)
            {
            r.x.dx = FP_SEG(&s_port_read);
            r.x.di = FP_OFF(&s_port_read);
            int86(LCO55, &r, &r);                   /* PORT_READ              */
            for(j = 0; j < 8; j++)      /* wyswietlenie kolejnych bitow       */
                printf("%d ", (s_port_read.LCO_PRDATA >> (7 - j)) & 1);
            printf("\n");
            s_port_write.LCO_PWDATA = s_port_read.LCO_PRDATA;
            r.x.dx = FP_SEG(&s_port_write);
            r.x.di = FP_OFF(&s_port_write);
            int86(LCO55, &r, &r);                   /* PORT_WRITE             */
            delay(200);
            }

        pressanykey();
}

/*****************************************************************************/
/* Przeznaczenie:                                                            */
/*   Zapis na port zaprogramowany jako wejsciowy - blad.                     */
/*****************************************************************************/
void writeportfail(void)
{
        printf("---------- "
               "Przesylanie danych do portu wejsciowego (blad!):\n\n");
                                /* zaprogramowanie kierunku pracy portow:    */
                                /*   1 (A) : wejscie                         */
                                /*   pozostale - bez zmiany                  */
        s_port_dir.LCO_PMODULE = modulenum;
        s_port_dir.LCO_PDIR = (LCO_IN_DIRECTION | LCO_CHANGE_DIR);
        r.x.dx = FP_SEG(&s_port_dir);
        r.x.di = FP_OFF(&s_port_dir);
        int86(LCO55, &r, &r);                   /* PORT_DIRECTION         */

        s_port_write.LCO_PWMODULE = modulenum;
        s_port_write.LCO_PWPORT = 1;
        s_port_write.LCO_PWDATA = 0;
        r.x.dx = FP_SEG(&s_port_write);
        r.x.di = FP_OFF(&s_port_write);
        int86(LCO55, &r, &r);                   /* PORT_WRITE             */
        if(s_port_write.LCO_STATUS != LCO_OK)
            drivererror(s_port_write.LCO_STATUS);

        pressanykey();
}
```

```
/*****************************************************************************/
/* Przeznaczenie:                                                            */
/*   Testowanie obslugi przerwan przychodzacych z modulu.                    */
/*****************************************************************************/
void testinterrupts(void)
{
        printf("---------- Testowanie przerwan przychodzacych z modulu:"
               "\n\n");
        if(s_module.LC0_MINT == 0xFF)
            {
            printf("Z modulem nie jest zwiazane zadne przerwanie!\n");
            return;
            }
                                /* w zaleznosci od typu modulu procedury   */
                                /* testujace maja rozna postac             */
        switch(s_module.LC0_MTYPE)
            {
            case LC_8255:
                test8255int();
                break;
            case LC_055_PIO:
                testPIOint();
                break;
            case LC_055_DCU:
                testDCUint();
                break;
            }
}

/*****************************************************************************/
/* Przeznaczenie:                                                            */
/*   Testowanie przerwan z modulu LC-8255.                                   */
/* Sposob:                                                                   */
/*   Pod wlasciwe przerwanie podkladana jest procedura newint8255.           */
/*   Przerwanie musi byc generowane z zewnatrz. Procedura konczona jest      */
/*   przez nacisniecie dowolnego klawisza.                                   */
/*****************************************************************************/
void test8255int(void)
{
        int     x,
                y;

        printf("(aby przerwac nacisnij dowolny klawisz . . .)\n");
        printf("Sumaryczna liczba przerwan: ");
        x = wherex();
        y = wherey();           /* biezace polozenie kursora - za tekstem  */

        interrupt_counter = 0;  /* wartosc poczatkowa licznika przerwan    */

                                /* podlozenie procedury obslugi przerwania */
        s_int_service.LC0_ISMODULE = modulenum;
        s_int_service.LC0_IPROC = newint8255;
        r.x.dx = FP_SEG(&s_int_service);
        r.x.di = FP_OFF(&s_int_service);
        int86(LC055, &r, &r);                   /* INTERRUPT_SERVICE_55    */

        displayinterrupts(x, y);
```

```
                                /* przywrocenie poprzedniej procedury      */
                                /* obslugi przerwania                      */
        s_int_service.LC0_IPROC = NULL;
        r.x.dx = FP_SEG(&s_int_service);
        r.x.di = FP_OFF(&s_int_service);
        int86(LC055, &r, &r);                   /* INTERRUPT_SERVICE_55    */

        getch();                        /* oproznienie bufora klawiatury   */
        pressanykey();                  /* oczekiwanie na operatora        */
}

/*****************************************************************************/
/* Przeznaczenie:                                                            */
/*   Procedura wywolywana przez driver'owa procedure obslugi przerwania z    */
/*   modulu LC-8255.                                                         */
/* Uwagi:                                                                    */
/*   Procedura musi byc "far"!.                                              */
/*   Procedura dostaje od driver'a stos dlugosci 128 bajtow (patrz           */
/*   dokumentacja driver'a). Jezeli jest to za malo to procedura powinna     */
/*   przelaczac sie na wlasny stos.                                          */
/*****************************************************************************/
void far newint8255(void)
{
        interrupt_counter++;
}

/*****************************************************************************/
/* Przeznaczenie:                                                            */
/*   Testowanie przerwan z modulu LC-055-PIO.                                */
/* Sposob:                                                                   */
/*   Pod wlasciwe przerwanie podkladana jest procedura newintPIODCU.         */
/*   Przerwanie musi byc generowane z zewnatrz. Procedura konczona jest      */
/*   przez nacisniecie dowolnego klawisza.                                   */
/*****************************************************************************/
void testPIOint(void)
{
        int     x,
                y;

        printf("(aby przerwac nacisnij dowolny klawisz . . .)\n");
        printf("Sumaryczna liczba przerwan: ");
        x = wherex();
        y = wherey();           /* biezace polozenie kursora - za tekstem  */

        interrupt_counter = 0;  /* wartosc poczatkowa licznika przerwan    */

                                /* podlozenie procedury obslugi przerwania */
        s_int_service.LC0_ISMODULE = modulenum;
        s_int_service.LC0_IPROC = newintPIODCU;
        r.x.dx = FP_SEG(&s_int_service);
        r.x.di = FP_OFF(&s_int_service);
        int86(LC055, &r, &r);                   /* INTERRUPT_SERVICE_55    */

                                        /* zezwolenie na przerwania        */
        s_int_mask.LC0_IMMODULE = modulenum;
        s_int_mask.LC0_IMASK = 1;
        r.x.dx = FP_SEG(&s_int_mask);
        r.x.di = FP_OFF(&s_int_mask);
```

```
        int86(LC055, &r, &r);                       /* INTERRUPT_MASK          */

        displayinterrupts(x, y);

                                            /* zakaz przerwan                  */
        s_int_mask.LC0_IMMODULE = modulenum;
        s_int_mask.LC0_IMASK = 0;
        r.x.dx = FP_SEG(&s_int_mask);
        r.x.di = FP_OFF(&s_int_mask);
        int86(LC055, &r, &r);                       /* INTERRUPT_MASK          */

                                  /* przywrocenie poprzedniej procedury        */
                                  /* obslugi przerwania                        */
        s_int_service.LC0_IPROC = NULL;
        r.x.dx = FP_SEG(&s_int_service);
        r.x.di = FP_OFF(&s_int_service);
        int86(LC055, &r, &r);                       /* INTERRUPT_SERVICE_55    */

        getch();                                /* oproznienie bufora klawiatury */
        pressanykey();                          /* oczekiwanie na operatora      */
}

/*****************************************************************************/
/* Przeznaczenie:                                                            */
/*   Testowanie przerwan z modulu LC-055-DCU.                                */
/* Sposob:                                                                   */
/*   Pod wlasciwe przerwanie podkladana jest procedura newintPIODCU.         */
/*   W przypadku linii przerywajacych polaczonych z CTC program odpowiednio  */
/*   steruje CTC tak, aby osiagnac generowanie przerwan z czestotliwoscia    */
/*   20 Hz. W pozostalych przypadkach przerwanie musi byc oczywiscie         */
/*   generowane z zewnatrz. Procedura konczona jest przez nacisniecie        */
/*   dowolnego klawisza.                                                     */
/*****************************************************************************/
void testDCUint(void)
{
        int     i,
                x,
                y;

                        /* rozpoznawanie zrodel przerwan w poszczegolnych    */
                        /* liniach przerywajacych                            */
        for(i = 0; i < LC0_MAX_INT_LINES; i++)
            switch((s_module.LC0_MINTSRC >> (2 * i)) & 3)
                {
                case LC0_NO_INT:
                    printf("Linia %d : niepodlaczona\n", i + 1);
                    break;
                case LC0_INTSRC_EXT:
                    printf("Linia %d : przerwania zewnetrzne\n", i + 1);
                    break;
                case LC0_INTSRC_CTC:
                    printf("Linia %d : przerwania od CTC\n", i + 1);
                    break;
                }
```

```
                        /* zaprogramowanie wszystkich kanalow CTC  */
                        /* na prace w trybie fali prostokatnej o   */
                        /* czestotliwosci 20 Hz                    */
                        /* jezeli ktorakolwiek z linii             */
                        /* przerywajacych jest przylaczona do CTC  */
                        /* to przerwania beda generowane           */
                        /* samoczynnie                             */
s_ctc_write.LC0_CMODULE = modulenum;
s_ctc_write.LC0_CMODE = LC0_SET_COUNTER_VALUE | LC0_SET_CTC_MODE;

s_ctc_write.LC0_CCTRL = CTC_COUNT0 | CTC_NB | CTC_MODE3 | CTC_BOTH;
s_ctc_write.LC0_CWDATA = (long)(s_module.LC0_MCLOCK[0]) * 50;
r.x.dx = FP_SEG(&s_ctc_write);
r.x.di = FP_OFF(&s_ctc_write);
int86(LC055, &r, &r);                   /* CTC_WRITE_55            */
s_ctc_write.LC0_CCTRL = CTC_COUNT1 | CTC_NB | CTC_MODE3 | CTC_BOTH;
s_ctc_write.LC0_CWDATA = (long)(s_module.LC0_MCLOCK[1]) * 50;
r.x.dx = FP_SEG(&s_ctc_write);
r.x.di = FP_OFF(&s_ctc_write);
int86(LC055, &r, &r);                   /* CTC_WRITE_55            */
s_ctc_write.LC0_CCTRL = CTC_COUNT2 | CTC_NB | CTC_MODE3 | CTC_BOTH;
s_ctc_write.LC0_CWDATA = (long)(s_module.LC0_MCLOCK[2]) * 50;
r.x.dx = FP_SEG(&s_ctc_write);
r.x.di = FP_OFF(&s_ctc_write);
int86(LC055, &r, &r);                   /* CTC_WRITE_55            */

                        /* zezwolenie na prace wszystkich  */
                        /* kanalow CTC                     */
s_ctc_gate.LC0_CGMODULE = modulenum;
s_ctc_gate.LC0_CGGATE = 0;
r.x.dx = FP_SEG(&s_ctc_gate);
r.x.di = FP_OFF(&s_ctc_gate);
int86(LC055, &r, &r);                   /* CTC_GATE                */

printf("(aby przerwac nacisnij dowolny klawisz . . .)\n");
printf("Sumaryczna liczba przerwan: ");
x = wherex();
y = wherey();           /* biezace polozenie kursora - za tekstem  */

interrupt_counter = 0;  /* wartosc poczatkowa licznika przerwan    */

                        /* podlozenie procedury obslugi przerwania */
s_int_service.LC0_ISMODULE = modulenum;
s_int_service.LC0_IPROC = newintPIODCU;
r.x.dx = FP_SEG(&s_int_service);
r.x.di = FP_OFF(&s_int_service);
int86(LC055, &r, &r);                   /* INTERRUPT_SERVICE_55    */

                        /* zaprogramowanie aktywnych poziomow      */
                        /* przerywajacych                          */
s_int_level.LC0_ILMODULE = modulenum;
s_int_level.LC0_ILEVEL = 0xF;   /* wszystkie aktywne poziomy - 1   */
r.x.dx = FP_SEG(&s_int_level);
r.x.di = FP_OFF(&s_int_level);
int86(LC055, &r, &r);                   /* INTERRUPT_LEVEL         */
```

```
                        /* zdjecie masek ze wszystkich linii          */
                        /* przerywajacych                             */
        s_int_mask.LCO_IMMODULE = modulenum;
        s_int_mask.LCO_IMASK = 0xF;     /* zezwolenie na przerwania   */
        r.x.dx = FP_SEG(&s_int_mask);
        r.x.di = FP_OFF(&s_int_mask);
        int86(LC055, &r, &r);                   /* INTERRUPT_MASK     */

        displayinterrupts(x, y);

                        /* zablokowanie wszystkich przerwan           */
        s_int_mask.LCO_IMASK = 0;
        s_int_mask.LCO_IMMODULE = modulenum;
        r.x.dx = FP_SEG(&s_int_mask);
        r.x.di = FP_OFF(&s_int_mask);
        int86(LC055, &r, &r);                   /* INTERRUPT_MASK     */
                        /* zgaszenie zgloszenia przerwania            */
        s_int_reset.LCO_IRMODULE = modulenum;
        r.x.dx = FP_SEG(&s_int_reset);
        r.x.di = FP_OFF(&s_int_reset);
        int86(LC055, &r, &r);                   /* INTERRUPT_RESET    */
                        /* przywrocenie poprzedniej procedury         */
                        /* obslugi przerwania                         */
        s_int_service.LCO_ISMODULE = modulenum;
        s_int_service.LCO_IPROC = NULL;
        r.x.dx = FP_SEG(&s_int_service);
        r.x.di = FP_OFF(&s_int_service);
        int86(LC055, &r, &r);                   /* INTERRUPT_SERVICE_55 */
                        /* zablokowanie wszystkich kanalow CTC        */
        s_ctc_gate.LCO_CGMODULE = modulenum;
        s_ctc_gate.LCO_CGGATE = 0;
        r.x.dx = FP_SEG(&s_ctc_gate);
        r.x.di = FP_OFF(&s_ctc_gate);
        int86(LC055, &r, &r);                   /* CTC_GATE           */

        getch();                        /* oproznienie bufora klawiatury */
        pressanykey();                  /* oczekiwanie na operatora   */
}

/*****************************************************************************/
/* Przeznaczenie:                                                            */
/*   Procedura wywolywana przez driver'owa procedure obslugi przerwania z    */
/*   modulow LC-055-DCU i LC-055-PIO.                                        */
/* Uwagi:                                                                    */
/*   Procedura musi byc "far"!.                                              */
/*   Procedura dostaje od driver'a stos dlugosci 128 bajtow (patrz           */
/*   dokumentacja driver'a). Jezeli jest to za malo to procedura powinna     */
/*   przelaczac sie na wlasny stos.                                          */
/*****************************************************************************/
void far newintPIODCU(void)
{
        interrupt_counter++;            /* zwieksz licznik przerwan   */
                                        /* zgaszenie przerwania w module */
        s_int_reset.LCO_IRMODULE = modulenum;
        r.x.dx = FP_SEG(&s_int_reset);
        r.x.di = FP_OFF(&s_int_reset);
        int86(LC055, &r, &r);                   /* INTERRUPT_RESET    */
}
```

```
/*****************************************************************************/
/* Przeznaczenie:                                                            */
/*   Oczekiwanie na przerwania i wyswietlanie biezacej liczby przerwan.      */
/* Parametry:                                                                */
/*   x, y - miejsce wyswietlania liczby przerwan                             */
/*****************************************************************************/
void displayinterrupts(int x, int y)
{
        word    old_counter = 0;

        while(TRUE)
            {
                        /* czekanie na klawisz lub kolejne przerwanie      */
            while(!kbhit() && old_counter == interrupt_counter)
                ;
            if(kbhit())
                break;                              /* nacisnieto klawisz      */
            else
                {                           /* przyszlo kolejne przerwanie     */
                old_counter = interrupt_counter;
                gotoxy(x, y);
                printf("%d", old_counter);
                }
            }
}

/*****************************************************************************/
/* Przeznaczenie:                                                            */
/*   Wypisanie komunikatow o bledzie / ostrzezeniu.                          */
/* Parametry:                                                                */
/*   status   - LCO_STATUS                                                   */
/*****************************************************************************/
void drivererror(char status)
{
        if(status > 0)
            printf("--------- Ostrzezenie:\n%s\n",
                   DriverWarnings[status - 1]);
        else
            printf("--------- Blad:\n%s\n", DriverErrors[-status - 1]);
}

/*****************************************************************************/
/* Przeznaczenie:                                                            */
/*   Zakonczenie pracy programu.                                             */
/* Sposob:                                                                   */
/*   Przez wykonanie funkcji exit. Wczesniej - rozstanie sie z driver'em     */
/*   za pomoca funkcji LEAVE_DRIVER_55.                                      */
/*****************************************************************************/
void quit(void)
{
        r.x.dx = FP_SEG(&s_leave);
        r.x.di = FP_OFF(&s_leave);
        int86(LCO55, &r, &r);                           /* LEAVE_DRIVER    */
        printf("Dziekuje, to wszystko!\n");
        exit(0);
}
```

ambex®

D O D A T E K C

do dokumentacji driver´ów modułów

LC-055-PIO, LC-055-DCU i LC-8255

AMBEX-55.PAS

struktury danych i stałe dla Pascala

Wydanie: lipiec 1992

```
(*****************************************************************************)
(*                                                                           *)
(*   Zestaw deklaracji struktur danych i definicji stalych do komunikacji    *)
(*   z driver'em LC055.DRV z poziomu jezyka Pascal.                          *)
(*                                                                           *)
(*****************************************************************************)

const
LC0_MAX_CTC             = 6;    (* maksymalna liczba kanalow CTC          *)
LC0_MAX_PORTS           = 6;    (* maksymalna liczba portow               *)
LC0_MAX_INT_LINES       = 4;    (* maksymalna liczba linii                *)
                                (* przerywajacych                         *)

(*************** REKORDY OPISOW ZLECEN FUNKCJI DRIVER'A ******************)

(* Rekord opisu zlecenia funkcji  MODULE_INIT_55 ========================= *)
type
lc0_init_55 =
    record
        LC0_CODE:       byte;           (* numer funkcji (0)              *)
        LC0_STATUS:     shortint;       (* kod odpowiedzi driver'a        *)
        LC0_IMODULE:    byte;           (* mapa modulow                   *)
    end;

(* Rekord opisu zlecenia funkcji  GET_TOTAL_CONFIGURATION_55 ============= *)
type
lc0_total_55 =
    record
        LC0_CODE:       byte;           (* numer funkcji (1)              *)
        LC0_STATUS:     shortint;       (* kod odpowiedzi driver'a        *)
        LC0_TONF:       byte;           (* inf. o zainstalowanych modulach *)
        LC0_TCTC:       byte;           (* liczba dostepnych kanalow      *)
        LC0_TPORT:      byte;           (* liczba dostepnych portow       *)
    end;

(* Rekord opisu zlecenia funkcji   GET_MODULE_CONFIGURATION_55 =========== *)
type
lc0_module_55 =
    record
        LC0_CODE:       byte;           (* numer funkcji (2)              *)
        LC0_STATUS:     shortint;       (* kod odpowiedzi driver'a        *)
        LC0_MMODULE:    byte;           (* numer modulu                   *)
        LC0_MTYPE:      byte;           (* typ modulu                     *)
        LC0_MBASE:      word;           (* adres bazowy rejestrow modulu  *)
        LC0_MPORT:      byte;           (* liczba dostepnych portow       *)
        LC0_MCTC:       byte;           (* liczba dostepnych kanalow      *)
                                        (* tablica czestotliwosci zegara  *)
                                        (*   CTC w kHz                    *)
        LC0_MCLOCK:     array [1..LC0_MAX_CTC] of word;
        LC0_CTCGATE:    byte;           (* bramki kanalow CTC             *)
        LC0_MINT:       byte;           (* numer przerwania (programowy)  *)
        LC0_MINTSRC:    word;           (* okreslenie zrodel przerwan     *)
        LC0_MINTLEV:    byte;           (* poziom przerywajacy            *)
        LC0_MINTMSK:    byte;           (* maski przerwan                 *)
    end;
```

```
const
                                (* kody typow modulow                    *)
LC_8255          = 1;           (* LC-8255                               *)
LC_055_PIO       = 2;           (* LC-055-PIO                            *)
LC_055_DCU       = 3;           (* LC-055-DCU                            *)

                                (* kody zrodla przerwania                 *)
LC0_NO_INT       = 0;           (* brak przerwania                       *)
LC0_INTSRC_EXT   = 1;           (* sygnal zewnetrzny                     *)
LC0_INTSRC_CTC   = 2;           (* CTC                                   *)

(* Rekord opisu zlecenia funkcji   PORT_DIRECTION ======================= *)
type
lc0_port_dir =
    record
        LC0_CODE:       byte;       (* numer funkcji (3)                 *)
        LC0_STATUS:     shortint;   (* kod odpowiedzi driver'a           *)
        LC0_PMODULE:    byte;       (* numer modulu                      *)
        LC0_PDIR:       longint;    (* kierunki pracy portow             *)
    end;

const
                                (* kody kierunkow pracy portow           *)
                                (* bit kierunku :                        *)
LC0_IN_DIRECTION        = 0;    (* port wejsciowy                        *)
LC0_OUT_DIRECTION       = 1;    (* port wyjsciowy                        *)
                                (* bit trybu :                           *)
LC0_CHANGE_DIR          = 2;    (* zmien kierunek pracy                  *)

(* Rekord opisu zlecenia funkcji   PORT_WRITE ========================== *)
type
lc0_port_write =
    record
        LC0_CODE:       byte;       (* numer funkcji (4)                 *)
        LC0_STATUS:     shortint;   (* kod odpowiedzi driver'a           *)
        LC0_PWMODULE:   byte;       (* numer modulu                      *)
        LC0_PWPORT:     byte;       (* numer portu                       *)
        LC0_PWDATA:     byte;       (* dana do wyslania                  *)
    end;

(* Rekord opisu zlecenia funkcji   PORT_READ =========================== *)
type
lc0_port_read =
    record
        LC0_CODE:       byte;       (* numer funkcji (5)                 *)
        LC0_STATUS:     shortint;   (* kod odpowiedzi driver'a           *)
        LC0_PRMODULE:   byte;       (* numer modulu                      *)
        LC0_PRPORT:     byte;       (* numer portu                       *)
        LC0_PRLATCH:    byte;       (* LC0_LATCH_ON/LC0_LATCH_OFF        *)
        LC0_PRDATA:     byte;       (* dana odczytana                    *)
    end;
```

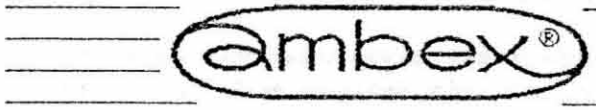
```
(* Rekord opisu zlecenia funkcji   PORT_LATCH ============================== *)
type
lc0_port_latch =
    record
        LC0_CODE:       byte;           (* numer funkcji (6)                  *)
        LC0_STATUS:     shortint;       (* kod odpowiedzi driver'a            *)
        LC0_PLMODULE:   byte;           (* numer modulu                       *)
        LC0_PLMODE:     byte;           (* tryb pracy                         *)
    end;

const
                        (* PORT_LATCH/LC0_PLMODE, PORT_READ/LC0_PRLATCH       *)
LC0_LATCH_ON    = 0;                    (* zatrzasnij                         *)
LC0_LATCH_OFF   = 1;                    (* zwolnij                            *)

(* Rekord opisu zlecenia funkcji   PORT_RESET ============================== *)
type
lc0_port_reset =
    record
        LC0_CODE:       byte;           (* numer funkcji (7)                  *)
        LC0_STATUS:     shortint;       (* kod odpowiedzi driver'a            *)
        LC0_PRSMODULE:  byte;           (* numer modulu                       *)
    end;

(* Rekord opisu zlecenia funkcji   CTC_WRITE_55 ============================ *)
type
lc0_ctc_write_55 =
    record
        LC0_CODE:       byte;           (* numer funkcji (8)                  *)
        LC0_STATUS:     shortint;       (* kod odpowiedzi driver'a            *)
        LC0_CMODULE:    byte;           (* numer modulu                       *)
        LC0_CMODE:      byte;           (* tryb pracy funkcji                 *)
        LC0_CCTRL:      byte;           (* bajt sterujacy                     *)
        LC0_CWDATA:     word;           (* wartosc licznika                   *)
    end;

const
                        (* wartosci fragmentow bajtu sterujacego 8253/8254    *)
CTC_NB          = 0;            (* kod naturalny binarny                      *)
CTC_BCD         = 1;            (* kod BCD                                    *)

CTC_MODE0       = 0;            (* tryb 0                                     *)
CTC_MODE1       = 2;            (* tryb 1                                     *)
CTC_MODE2       = 4;            (* tryb 2                                     *)
CTC_MODE3       = 6;            (* tryb 3                                     *)
CTC_MODE4       = 8;            (* tryb 4                                     *)
CTC_MODE5       = 10;           (* tryb 5                                     *)

CTC_LSB         = $10;          (* tylko mlodszy bajt                         *)
CTC_MSB         = $20;          (* tylko starszy bajt                         *)
CTC_BOTH        = $30;          (* mlodszy - starszy                          *)

CTC_COUNT0      = $00;          (* licznik 0                                  *)
CTC_COUNT1      = $40;          (* licznik 1                                  *)
CTC_COUNT2      = $80;          (* licznik 2                                  *)
```

```
                                    (* kody trybu pracy funkcji CTC_WRITE_55   *)
LC0_SET_CTC_MODE        = 1;            (* zaprogramuj tryb pracy kanalu   *)
LC0_SET_COUNTER_VALUE   = 2;            (* zaladuj nowa wartosc licznika   *)

(* Rekord opisu zlecenia funkcji   CTC_GATE ============================== *)
type
lc0_ctc_gate =
    record
        LC0_CODE:       byte;           (* numer funkcji (9)               *)
        LC0_STATUS:     shortint;       (* kod odpowiedzi driver'a         *)
        LC0_CGMODULE:   byte;           (* numer modulu                    *)
        LC0_CGGATE:     byte;           (* maska aktywnych kanalow         *)
    end;

(* Rekord opisu zlecenia funkcji   CTC_READ_55 =========================== *)
type
lc0_ctc_read_55 =
    record
        LC0_CODE:       byte;           (* numer funkcji (10)              *)
        LC0_STATUS:     shortint;       (* kod odpowiedzi driver'a         *)
        LC0_CRMODULE:   byte;           (* numer modulu                    *)
        LC0_CRCOUNT:    byte;           (* numer kanalu                    *)
        LC0_CRDATA:     word;           (* odczytana dana                  *)
    end;

(* Rekord opisu zlecenia funkcji   INTERRUPT_MASK ======================== *)
type
lc0_int_mask =
    record
        LC0_CODE:       byte;           (* numer funkcji (11)              *)
        LC0_STATUS:     shortint;       (* kod odpowiedzi driver'a         *)
        LC0_IMMODULE:   byte;           (* numer modulu                    *)
        LC0_IMASK:      byte;           (* maska przerwan                  *)
    end;

(* Rekord opisu zlecenia funkcji   INTERRUPT_LEVEL ======================= *)
type
lc0_int_level =
    record
        LC0_CODE:       byte;           (* numer funkcji (12)              *)
        LC0_STATUS:     shortint;       (* kod odpowiedzi driver'a         *)
        LC0_ILMODULE:   byte;           (* numer modulu                    *)
        LC0_ILEVEL:     byte;           (* poziom przerywajacy na          *)
                                        (* poszczegolnych liniach          *)
    end;

(* Rekord opisu zlecenia funkcji   INTERRUPT_SERVICE_55 ================== *)
type
lc0_int_service =
    record
        LC0_CODE:       byte;           (* numer funkcji (13)              *)
        LC0_STATUS:     shortint;       (* kod odpowiedzi driver'a         *)
        LC0_ISMODULE:   byte;           (* numer modulu                    *)
        LC0_IPROC:      pointer;        (* adres procedury obslugi         *)
    end;
```

```
(* Rekord opisu zlecenia funkcji   INTERRUPT_RESET ======================= *)
type
lc0_int_reset =
    record
        LC0_CODE:       byte;           (* numer funkcji (14)              *)
        LC0_STATUS:     shortint;       (* kod odpowiedzi driver'a         *)
        LC0_IRMODULE:   byte;           (* numer modulu                    *)
    end;

(* Rekord opisu zlecenia funkcji   READ_STATUS ============================ *)
type
lc0_read_status =
    record
        LC0_CODE:       byte;           (* numer funkcji (15)              *)
        LC0_STATUS:     shortint;       (* kod odpowiedzi driver'a         *)
        LC0_SMODULE:    byte;           (* numer modulu                    *)
        LC0_SDATA:      byte;           (* odczytany status                *)
        LC0_SRESRVD:    byte;           (* pole zarezerwowane              *)
    end;

const
                                        (* bity statusu (READ_STATUS)      *)
LC0_INT1_INTERRUPTED    = 1;            (* linia INT 1 zglosila przerwanie *)
LC0_INT2_INTERRUPTED    = 2;            (* linia INT 2 zglosila przerwanie *)
LC0_INT3_INTERRUPTED    = 4;            (* linia INT 3 zglosila przerwanie *)
LC0_INT4_INTERRUPTED    = 8;            (* linia INT 4 zglosila przerwanie *)
LC0_COMMON_INTERRUPT    = 16;           (*jedna z linii zglosila przerwanie*)
LC0_CTC_OUT0            = 32;           (* stan linii OUT kanalu 0         *)
LC1_CTC_OUT1            = 64;           (* stan linii OUT kanalu 1         *)
LC0_CTC_OUT2            = 128;          (* stan linii OUT kanalu 2         *)

(* Rekord opisu zlecenia funkcji  LEAVE_DRIVER_55 ======================== *)
type
lc0_leave_55 =
    record
        LC0_CODE:       byte;           (* numer funkcji (16)              *)
        LC0_STATUS:     shortint;       (* kod odpowiedzi driver'a         *)
    end;


const

(* ***************** kody funkcji driver'a ********************************)
MODULE_INIT_55                  = 0;        (* inicjalizacja modulow   *)
GET_TOTAL_CONFIGURATION_55      = 1;        (* odczyt konfig. modulow  *)
GET_MODULE_CONFIGURATION_55     = 2;        (* odczyt konfig. modulu   *)
PORT_DIRECTION                  = 3;        (* ustawienie kierunku     *)
                                            (* pracy portow            *)
PORT_WRITE                      = 4;        (* zapis do portu          *)
PORT_READ                       = 5;        (* odczyt portu            *)
PORT_LATCH                      = 6;        (* zatrzasniecie danych    *)
PORT_RESET                      = 7;        (* zerowanie ukladow 8255  *)
                                            (* (ustawienie wszystkich  *)
                                            (* portow w kierunku IN)   *)
CTC_WRITE_55                    = 8;        (* zaprogramowanie i/lub   *)
                                            (* zapisanie wartosci do   *)
                                            (* kanalu CTC              *)
```

```
CTC_GATE                    = 9;        (* bramkowanie kanalu CTC  *)
CTC_READ_55                 = 10;       (* odczyt kanalu CTC       *)
INTERRUPT_MASK              = 11;       (* maskowanie przerwan     *)
INTERRUPT_LEVEL             = 12;       (* ustawianie poziomu      *)
                                        (* przerywajacego          *)
INTERRUPT_SERVICE_55        = 13;       (* deklarowanie procedury  *)
                                        (* obslugi przerwania      *)
INTERRUPT_RESET             = 14;       (* gaszenie zgloszenia     *)
                                        (* przerwania              *)
READ_STATUS                 = 15;       (* odczyt statusu          *)
LEAVE_DRIVER_55             = 16;       (* zakonczenie pracy z     *)
                                        (* driver'em               *)

(********** numer przerwania obslugiwanego przez driver **********)

LC055   = $98;


(****************** kody odpowiedzi funkcji driver'a ******************)

LC0_OK                  = 0;            (* OK                          *)

(* ==============   ostrzezenia ================   *)

LC0_NON_EX_MOD          = 1;            (* nie istniejacy(e) modul(y)  *)
LC0_NO_LATCH_WARN       = 2;            (* modul nie ma mozliwosci     *)
                                        (* zatrzaskiwania danych       *)


(* ================   bledy ==================   *)

LC0_UNKN_FUNC           = -1;           (* nieznany kod funkcji        *)
LC0_NO_MODULE           = -2;           (* brak modulu(ow)             *)
LC0_NOT_OUTPORT         = -3;           (* port nie jest wyjsciowy     *)
LC0_NONEX_DEV           = -4;           (* nie istnieje urzadzenie o tym *)
                                        (* numerze                     *)
LC0_NOT_INPORT          = -5;           (* port nie jest wejsciowy     *)
LC0_BAD_COUNTER         = -6;           (* zly numer kanalu CTC        *)
LC0_NO_LATCH            = -7;           (* dany modul nie moze zatrzaskiwac*)
LC0_BAD_PROC            = -19;          (* bledny adres procedury obslugi *)
                                        (* przerwan                    *)
LC0_NOT_SUPPORTED       = -24;          (* dla danego modulu funkcja nie *)
                                        (* jest realizowana            *)
LC0_BAD_CTC_MODE        = -25;          (* bledny tryb pracy CTC       *)
LC0_NO_IRQ              = -29;          (* z danum modulem nie jest    *)
                                        (* zwiazane zadne przerwanie   *)
LC0_NOT_FULLY_SUP       = -30;          (* zadany tryb funkcji nie     *)
                                        (* jest realizowany dla danego typu*)
                                        (* modulu lub funkcja w opracowaniu*)
LC0_INTR_INST           = -33;          (* proba powtornej instalacji  *)
                                        (* przerwania                  *)
LC0_CTC_NOT_PROGRAMMED  = -37;          (* zapis wartosci do           *)
                                        (* niezaprogramowanego licznika *)
LC0_REJECTED            = -38;          (* za duzo jednoczesnych odwolan do*)
                                        (* driver'a                    *)
```

```
(***************************** kody numerow modulow ***************************)
LCO_MODA        = 1;            (* modul A                                    *)
LCO_MODB        = 2;            (* modul B                                    *)
LCO_MODC        = 3;            (* modul C                                    *)
LCO_MODD        = 4;            (* modul D                                    *)

(***************************** maski modulow w mapie **************************)
LCO_MODAMAP     = 1;            (* modul A                                    *)
LCO_MODBMAP     = 2;            (* modul B                                    *)
LCO_MODCMAP     = 4;            (* modul C                                    *)
LCO_MODDMAP     = 8;            (* modul D                                    *)
```



D O D A T E K D

do dokumentacji drivera modułów

LC-055-PIO, LC-055-DCU i LC-8255

TEST55.PAS

przykładowy program w Pascalu

Wydanie: lipiec 1992

```
(*****************************************************************************)
(* Program przykladowy pokazujacy sposob wykorzystania drivera modulow       *)
(* cyfrowych: LC-8255, LC-055-PIO i LC-055-DCU                               *)
(*****************************************************************************)

program test;

uses crt, dos;

{$i ambex-55.pas}

(********************* deklaracje zmiennych ********************************)
var
                                (* rekordy opisu poszczegolnych zlecen      *)
    s_init:         lc0_init_55;
    s_total:        lc0_total_55;
    s_module:       lc0_module_55;
    s_port_dir:     lc0_port_dir;
    s_port_write:   lc0_port_write;
    s_port_read:    lc0_port_read;
    s_ctc_write:    lc0_ctc_write_55;
    s_ctc_gate:     lc0_ctc_gate;
    s_int_mask:     lc0_int_mask;
    s_int_level:    lc0_int_level;
    s_int_service:  lc0_int_service;
    s_int_reset:    lc0_int_reset;
    s_leave:        lc0_leave_55;
    r: registers;                       (* parametr dla funkcji intr       *)
    modulenum: integer;                 (* numer badanego modulu           *)
    interrupt_counter: word;            (* licznik przerwan                *)

(*****************************************************************************)
(* Przeznaczenie:                                                            *)
(*   Inicjalizacja zmiennych.                                                *)
(*****************************************************************************)
procedure initprogram;
begin
    s_init.LC0_CODE         := MODULE_INIT_55;
    s_total.LC0_CODE        := GET_TOTAL_CONFIGURATION_55;
    s_module.LC0_CODE       := GET_MODULE_CONFIGURATION_55;
    s_port_dir.LC0_CODE     := PORT_DIRECTION;
    s_port_write.LC0_CODE   := PORT_WRITE;
    s_port_read.LC0_CODE    := PORT_READ;
    s_ctc_write.LC0_CODE    := CTC_WRITE_55;
    s_ctc_gate.LC0_CODE     := CTC_GATE;
    s_int_mask.LC0_CODE     := INTERRUPT_MASK;
    s_int_level.LC0_CODE    := INTERRUPT_LEVEL;
    s_int_service.LC0_CODE  := INTERRUPT_SERVICE_55;
    s_int_reset.LC0_CODE    := INTERRUPT_RESET;
    s_leave.LC0_CODE        := LEAVE_DRIVER_55;
end;
```

```
(*****************************************************************************)
(* Przeznaczenie:                                                            *)
(*   Zakonczenie pracy programu.                                             *)
(* Sposob:                                                                   *)
(*   Przez wykonanie funkcji exit. Wczesniej - rozstanie sie z driver'em     *)
(*   za pomoca funkcji LEAVE_DRIVER_55.                                      *)
(*****************************************************************************)
procedure quit;
begin
    r.dx := seg(s_leave);
    r.di := ofs(s_leave);
    intr(LC055, r);                                    (* LEAVE_DRIVER    *)

    writeln('Dziekuje, to wszystko!');
end;

(*****************************************************************************)
(* Przeznaczenie:                                                            *)
(*   Wypisanie komunikatow o bledzie / ostrzezeniu / dodatkowej informacji   *)
(*   bledzie.                                                                *)
(* Parametry:                                                                *)
(*   status   - LC0_STATUS                                                   *)
(*****************************************************************************)
procedure drivererror(status: shortint);
begin
    if status > 0 then
        writeln('--------- Ostrzezenie: ', status)
    else
        writeln('--------- Blad: ', status);
end;

(*****************************************************************************)
(* Przeznaczenie:                                                            *)
(*   Przerwa miedzy kolejnymi czesciami programu - oczekiwanie na reakcje    *)
(*   operatora.                                                              *)
(*****************************************************************************)
procedure pressanykey;
var
    c: char;
begin
    writeln;
    writeln('Nacisnij dowolny klawisz . . .');
    c := readkey;
    clrscr;
    gotoxy(1, 1);
end;

(*****************************************************************************)
(* Przeznaczenie:                                                            *)
(*   Oczekiwanie na przerwania i wyswietlanie biezacej liczby przerwan.      *)
(* Parametry:                                                                *)
(*   x, y - miejsce wyswietlania liczby przerwan                             *)
(*****************************************************************************)
procedure displayinterrupts(x, y: integer);
var
    old_counter: word;
```

```
begin
    old_counter := 0;
    while true do
        begin
                        (* czekanie na klawisz lub kolejne przerwanie      *)
        while (not keypressed and (old_counter = interrupt_counter)) do
            ;
        if keypressed then                          (* nacisnieto klawisz      *)
            exit
        else                                  (* przyszlo kolejne przerwanie     *)
            begin
                old_counter := interrupt_counter;
                gotoxy(x, y);
                write(old_counter);
            end;
        end;
end;

{$F+}                               (* procedury obslugi przerwan musza byc    *)
                                    (* procedurami typu 'far'                  *)

(*****************************************************************************)
(* Przeznaczenie:                                                            *)
(*   Procedura wywolywana przez driver'owa procedure obslugi przerwania z    *)
(*   modulu LC-8255.                                                         *)
(* Uwagi:                                                                    *)
(*   Procedura musi byc 'far'!.                                              *)
(*   Procedura dostaje od driver'a stos dlugosci 128 bajtow (patrz           *)
(*   dokumentacja driver'a). Jezeli jest to za malo to procedura powinna     *)
(*   przelaczac sie na wlasny stos.                                          *)
(*****************************************************************************)
procedure newint8255;
begin
                                        (* zwieksz licznik przerwan         *)
    interrupt_counter := interrupt_counter + 1;
end;

(*****************************************************************************)
(* Przeznaczenie:                                                            *)
(*   Procedura wywolywana przez driver'owa procedure obslugi przerwania z    *)
(*   modulow LC-055-DCU i LC-055-PIO.                                        *)
(* Uwagi:                                                                    *)
(*   Procedura musi byc 'far'!.                                              *)
(*   Procedura dostaje od driver'a stos dlugosci 128 bajtow (patrz           *)
(*   dokumentacja driver'a). Jezeli jest to za malo to procedura powinna     *)
(*   przelaczac sie na wlasny stos.                                          *)
(*****************************************************************************)
procedure newintPIODCU;
begin
                                        (* zwieksz licznik przerwan         *)
    interrupt_counter := interrupt_counter + 1;
                                        (* zgaszenie przerwania w module    *)
    s_int_reset.LC0_IRMODULE := modulenum;
    r.dx := seg(s_int_reset);
    r.di := ofs(s_int_reset);
    intr(LC055, r);                             (* INTERRUPT_RESET          *)
end;
{$F-}
```

```
(*****************************************************************************)
(* Przeznaczenie:                                                            *)
(*   Testowanie przerwan z modulu LC-8255.                                   *)
(* Sposob:                                                                   *)
(*   Pod wlasciwe przerwanie podkladana jest procedura newint8255.           *)
(*   Przerwanie musi byc generowane z zewnatrz. Procedura konczona jest      *)
(*   przez nacisniecie dowolnego klawisza.                                   *)
(*****************************************************************************)
procedure test8255int;
var
    x: integer;
    y: integer;
    c: char;
begin
    writeln('(aby przerwac nacisnij dowolny klawisz . . .)');
    write('Sumaryczna liczba przerwan: ');
    x := wherex;
    y := wherey;                    (* biezace polozenie kursora - za tekstem  *)

                                    (* wartosc poczatkowa licznika przerwan    *)
    interrupt_counter := 0;

                                    (* podlozenie procedury obslugi przerwania *)
    s_int_service.LC0_ISMODULE := modulenum;
    s_int_service.LC0_IPROC := @newint8255;
    r.dx := seg(s_int_service);
    r.di := ofs(s_int_service);
                                    (* spowodowanie ze driver (a co za tym     *)
                                    (* idzie procedura newint8255) dostanie    *)
                                    (* wlasciwy segment DS; moze tu byc uzyta  *)
                                    (* dowolna zmienna byle globalna -         *)
                                    (* - umieszczona w segmencie DS            *)
    r.ds := seg(interrupt_counter);
    intr(LC055, r);                                 (* INTERRUPT_SERVICE_55    *)

    displayinterrupts(x, y);

                                    (* przywrocenie poprzedniej procedury      *)
                                    (* obslugi przerwania                      *)
    s_int_service.LC0_ISMODULE := modulenum;
    s_int_service.LC0_IPROC := nil;
    r.dx := seg(s_int_service);
    r.di := ofs(s_int_service);
    intr(LC055, r);                                 (* INTERRUPT_SERVICE_55    *)

    c := readkey;                           (* oproznienie bufora klawiatury   *)

    pressanykey;
end;
```

```
(*************************************************************************)
(* Przeznaczenie:                                                        *)
(*   Testowanie przerwan z modulu LC-055-PIO.                            *)
(* Sposob:                                                               *)
(*   Pod wlasciwe przerwanie podkladana jest procedura newintPIODCU.     *)
(*   Przerwanie musi byc generowane z zewnatrz. Procedura konczona jest  *)
(*   przez nacisniecie dowolnego klawisza.                               *)
(*************************************************************************)
procedure testPIOint;
var
    x: integer;
    y: integer;
    c: char;
begin
    writeln('(aby przerwac nacisnij dowolny klawisz . . .)');
    write('Sumaryczna liczba przerwan: ');
    x := wherex;
    y := wherey;                    (* biezace polozenie kursora - za tekstem  *)

                                    (* wartosc poczatkowa licznika przerwan    *)
    interrupt_counter := 0;

                                    (* podlozenie procedury obslugi przerwania *)
    s_int_service.LC0_ISMODULE := modulenum;
    s_int_service.LC0_IPROC := @newintPIODCU;
    r.dx := seg(s_int_service);
    r.di := ofs(s_int_service);
                                    (* spowodowanie ze driver (a co za tym     *)
                                    (* idzie procedura newintPIODCU) dostanie  *)
                                    (* wlasciwy segment DS; moze tu byc uzyta  *)
                                    (* dowolna zmienna byle globalna -         *)
                                    (* - umieszczona w segmencie DS            *)
    r.ds := seg(interrupt_counter);
    intr(LC055, r);                                 (* INTERRUPT_SERVICE_55    *)

                                            (* zezwolenie na przerwania        *)
    s_int_mask.LC0_IMMODULE := modulenum;
    s_int_mask.LC0_IMASK := 1;
    r.dx := seg(s_int_mask);
    r.di := ofs(s_int_mask);
    intr(LC055, r);                                 (* INTERRUPT_MASK          *)

    displayinterrupts(x, y);

                                    (* zablokowanie wszystkich przerwan        *)
    s_int_mask.LC0_IMASK := 0;
    s_int_mask.LC0_IMMODULE := modulenum;
    r.dx := seg(s_int_mask);
    r.di := ofs(s_int_mask);
    intr(LC055, r);                                 (* INTERRUPT_MASK          *)

                                    (* przywrocenie poprzedniej procedury      *)
                                    (* obslugi przerwania                      *)
    s_int_service.LC0_ISMODULE := modulenum;
    s_int_service.LC0_IPROC := nil;
    r.dx := seg(s_int_service);
    r.di := ofs(s_int_service);
    intr(LC055, r);                                 (* INTERRUPT_SERVICE_55    *)
```

```
    c := readkey;                          (* oproznienie bufora klawiatury   *)

    pressanykey;
end;

(*****************************************************************************)
(* Przeznaczenie:                                                            *)
(*   Testowanie przerwan z modulu LC-055-DCU.                                *)
(* Sposob:                                                                   *)
(*   Pod wlasciwe przerwanie podkladana jest procedura newintPIODCU.         *)
(*   W przypadku linii przerywajacych polaczonych z CTC program odpowiednio*)
(*   steruje CTC tak, aby osiagnac generowanie przerwan z czestotliwoscia  *)
(*   20 Hz. W pozostalych przypadkach przerwanie musi byc oczywiscie         *)
(*   generowane z zewnatrz. Procedura konczona jest przez nacisniecie        *)
(*   dowolnego klawisza.                                                     *)
(*****************************************************************************)
procedure testDCUint;
var
    i: integer;
    x: integer;
    y: integer;
    c: char;
begin

                        (* rozpoznawanie zrodel przerwan w poszczegolnych  *)
                        (* liniach przerywajacych                          *)
    for i := 1 to LC0_MAX_INT_LINES do
      case ((s_module.LC0_MINTSRC shr (2 * i)) and 3) of
        LC0_NO_INT:
            writeln('Linia ', i, ' : niepodlaczona');
        LC0_INTSRC_EXT:
            writeln('Linia ', i, ' : przerwania zewnetrzne');
        LC0_INTSRC_CTC:
            writeln('Linia ', i, ' : przerwania od CTC');
      end;
                                (* zaprogramowanie wszystkich kanalow CTC  *)
                                (* na prace w trybie fali prostokatnej o   *)
                                (* czestotliwosci 20 Hz                    *)
                                (* jezeli ktorakolwiek z linii             *)
                                (* przerywajacych jest przylaczona do CTC  *)
                                (* to przerwania beda generowane           *)
                                (* samoczynnie                             *)
    s_ctc_write.LC0_CMODULE := modulenum;
    s_ctc_write.LC0_CMODE := LC0_SET_COUNTER_VALUE or LC0_SET_CTC_MODE;

    s_ctc_write.LC0_CCTRL := CTC_COUNT0 or CTC_NB or CTC_MODE3 or CTC_BOTH;
    s_ctc_write.LC0_CWDATA := s_module.LC0_MCLOCK[1] * 50;
    r.dx := seg(s_ctc_write);
    r.di := ofs(s_ctc_write);
    intr(LC055, r);                                (* CTC_WRITE_55          *)
    s_ctc_write.LC0_CCTRL := CTC_COUNT1 or CTC_NB or CTC_MODE3 or CTC_BOTH;
    s_ctc_write.LC0_CWDATA := s_module.LC0_MCLOCK[2] * 50;
    r.dx := seg(s_ctc_write);
    r.di := ofs(s_ctc_write);
    intr(LC055, r);                                (* CTC_WRITE_55          *)
    s_ctc_write.LC0_CCTRL := CTC_COUNT2 or CTC_NB or CTC_MODE3 or CTC_BOTH;
    s_ctc_write.LC0_CWDATA := s_module.LC0_MCLOCK[3] * 50;
```

```
r.dx := seg(s_ctc_write);
r.di := ofs(s_ctc_write);
intr(LC055, r);                                  (* CTC_WRITE_55          *)
                                        (* zezwolenie na prace wszystkich  *)
                                        (* kanalow CTC                     *)
s_ctc_gate.LC0_CGMODULE := modulenum;
s_ctc_gate.LC0_CGGATE := 0;
r.dx := seg(s_ctc_gate);
r.di := ofs(s_ctc_gate);
intr(LC055, r);                                  (* CTC_GATE              *)

writeln('(aby przerwac nacisnij dowolny klawisz . . .)');
write('Sumaryczna liczba przerwan: ');
x := wherex;
y := wherey;                    (* biezace polozenie kursora - za tekstem  *)

                                (* wartosc poczatkowa licznika przerwan    *)
interrupt_counter := 0;

                                (* podlozenie procedury obslugi przerwania *)
s_int_service.LC0_ISMODULE := modulenum;
s_int_service.LC0_IPROC := @newintPIODCU;
r.dx := seg(s_int_service);
r.di := ofs(s_int_service);
                                (* spowodowanie ze driver (a co za tym     *)
                                (* idzie procedura newintPIODCU) dostanie  *)
                                (* wlasciwy segment DS; moze tu byc uzyta  *)
                                (* dowolna zmienna byle globalna -         *)
                                (* - umieszczona w segmencie DS            *)
r.ds := seg(interrupt_counter);
intr(LC055, r);                                  (* INTERRUPT_SERVICE_55   *)

                                (* zaprogramowanie aktywnych poziomow      *)
                                (* przerywajacych                          *)
s_int_level.LC0_ILMODULE := modulenum;
s_int_level.LC0_ILEVEL := 15;          (* wszystkie aktywne poziomy - 1    *)
r.dx := seg(s_int_level);
r.di := ofs(s_int_level);
intr(LC055, r);                                  (* INTERRUPT_LEVEL       *)

                                (* zdjecie masek ze wszystkich linii       *)
                                (* przerywajacych                          *)
s_int_mask.LC0_IMMODULE := modulenum;
s_int_mask.LC0_IMASK := 15;            (* zezwolenie na przerwania         *)
r.dx := seg(s_int_mask);
r.di := ofs(s_int_mask);
intr(LC055, r);                                  (* INTERRUPT_MASK        *)

displayinterrupts(x, y);

                                (* zablokowanie wszystkich przerwan        *)
s_int_mask.LC0_IMASK := 0;
s_int_mask.LC0_IMMODULE := modulenum;
r.dx := seg(s_int_mask);
r.di := ofs(s_int_mask);
intr(LC055, r);                                  (* INTERRUPT_MASK        *)
```

```
                                (* zgaszenie zgloszenia przerwania          *)
    s_int_reset.LC0_IRMODULE := modulenum;
    r.dx := seg(s_int_reset);
    r.di := ofs(s_int_reset);
    intr(LC055, r);                              (* INTERRUPT_RESET          *)

                                (* przywrocenie poprzedniej procedury       *)
                                (* obslugi przerwania                       *)
    s_int_service.LC0_ISMODULE := modulenum;
    s_int_service.LC0_IPROC := nil;
    r.dx := seg(s_int_service);
    r.di := ofs(s_int_service);
    intr(LC055, r);                              (* INTERRUPT_SERVICE_55     *)

                                (* zablokowanie wszystkich kanalow CTC      *)
    s_ctc_gate.LC0_CGMODULE := modulenum;
    s_ctc_gate.LC0_CGGATE := 0;
    r.dx := seg(s_ctc_gate);
    r.di := ofs(s_ctc_gate);
    intr(LC055, r);                              (* CTC_GATE                 *)

    c := readkey;                          (* oproznienie bufora klawiatury  *)

    pressanykey;
end;

(*****************************************************************************)
(* Przeznaczenie:                                                            *)
(*   Testowanie obslugi przerwan przychodzacych z modulu.                    *)
(*****************************************************************************)
procedure testinterrupts;
begin
    writeln('---------- Testowanie przerwan przychodzacych z modulu:');
    writeln;
    if s_module.LC0_MINT = $FF then
        writeln('Z modulem nie jest zwiazane zadne przerwanie!')
    else
        begin
                                (* w zaleznosci od typu modulu procedury    *)
                                (* testujace maja rozna postac              *)
            case s_module.LC0_MTYPE of
                LC_8255:
                    test8255int;
                LC_055_PIO:
                    testPIOint;
                LC_055_DCU:
                    testDCUint;
            end;
        end;
end;

(*****************************************************************************)
(* Przeznaczenie:                                                            *)
(*   Rozpoznanie konfiguracji badanego modulu.                               *)
(* Sposob:                                                                   *)
(*   Przez wykorzystanie funkcji driver'a GET_TOTAL_CONFIGURATION,           *)
(*   GET_MODULE_CONFIGURATION.                                               *)
```

```
(* Uwagi:                                                                  *)
(*   Funkcja nadaje wartosc zmiennej modulenum (numer badanego modulu)     *)
(*   wypelnia struktury total, module, info i inicjalizuje zainstalowane   *)
(*   moduly. Jezeli zainstalowanych jest wiecej niz jeden modulow to       *)
(*   program bedzie analizowal tylko pierwszy z brzegu.                    *)
(*************************************************************************)
procedure askdriver;
label   ok;
begin
    r.dx := seg(s_total);
    r.di := ofs(s_total);
    intr(LC055, r);                         (* GET_TOTAL_CONFIGURATION_55    *)

                                (* inicjalizacja zainstalowanych modulow     *)
    s_init.LC0_IMODULE := s_total.LC0_TONF and $F;
    r.dx := seg(s_init);
    r.di := ofs(s_init);
    intr(LC055, r);                                       (* MODULE_INIT_55  *)

                                        (* sprawdzenie, ktory modul jest     *)
                                        (*   zainstalowany: A, B, C czy D    *)
    for modulenum := 1 to 4 do
        if (s_total.LC0_TONF and (1 shl (modulenum - 1))) <> 0 then
            goto ok;

ok:
                                        (* spytanie o konfiguracje modulu    *)
    s_module.LC0_MMODULE := modulenum;
    r.dx := seg(s_module);
    r.di := ofs(s_module);
    intr(LC055, r);                     (* GET_MODULE_CONFIGURATION_55       *)
end;

(*************************************************************************)
(* Przeznaczenie:                                                        *)
(*   Wyswietlenie konfiguracji badanego modulu.                          *)
(* Sposob:                                                               *)
(*   Przez wykorzystanie informacji zawartych w strukturze s_module.     *)
(*************************************************************************)
procedure displayconfiguration;
var
    i: integer;
begin
                                        (* konfiguracja modulu               *)
    writeln;
    write('---------- Konfiguracja modulu ');
    case s_module.LC0_MTYPE of
        LC_8255:
            write('LC-8255');
        LC_055_PIO:
            write('LC-055-PIO');
        LC_055_DCU:
            write('LC-055-DCU');
    end;
    writeln(' (', chr(ord('A') + modulenum - 1), ') :');
    writeln;
    writeln('Adres modulu: ', s_module.LC0_MBASE);
    writeln('Liczba portow we/wy: ', s_module.LC0_MPORT);
```

```
    writeln('Liczba kanalow CTC: ', s_module.LCO_MCTC);
    if s_module.LCO_MCTC <> 0 then
      begin
        if s_module.LCO_MTYPE = LC_055_PIO then
          begin                                    (* wspolna czestotliwosc   *)
            if s_module.LCO_MCLOCK[1] = 0 then
                writeln('Wszystkie kanaly CTC ',
                        'podlaczone do zegara zewnetrznego')
            else
                writeln('Czestotliwosc zegara CTC: ',
                        s_module.LCO_MCLOCK[1], ' kHz');
            end
        else                          (* LC-055-DCU gdyz LC-8255 nie ma CTC      *)
          begin
                          (* indywidualna czestotliwosc dla kazdego kanalu   *)
            for i := 1 to s_module.LCO_MCTC do
                if s_module.LCO_MCLOCK[i] = 0 then
                    writeln('Kanal ', i,
                            ' CTC podlaczony do zegara zewnetrznego')
                else
                    writeln('Czestotliwosc zegara kanalu ', i, ' CTC: ',
                            s_module.LCO_MCLOCK[i], ' kHz');
          end;
      end;
    if s_module.LCO_MINT = $FF then
        writeln('Modul nie jest podlaczony do zadnego przerwania')
    else
        writeln('Numer przerwania zwiazanego z modulem: ',
                s_module.LCO_MINT);

    pressanykey;                  (* oczekiwanie na operatora                  *)
end;

(*****************************************************************************)
(* Przeznaczenie:                                                            *)
(*   Odczyt portu 1 (A) zaprogramowanego jako wejsciowy i wyslanie           *)
(*   odczytanej wartosci na port 2 (B) zaprogramowany jako wyjsciowy.        *)
(*****************************************************************************)
procedure readport;
var
    i: integer;
    j: integer;
begin
    writeln('---------- Przesylanie danych z portu wejsciowego na wyjsciowy');
    writeln;
                                  (* zaprogramowanie kierunku pracy portow:  *)
                                  (*   1 (A) : wejscie                       *)
                                  (*   2 (B) : wyjscie                       *)
                                  (*   pozostale - bez zmiany                *)
    s_port_dir.LCO_PMODULE := modulenum;
    s_port_dir.LCO_PDIR := (LCO_IN_DIRECTION or LCO_CHANGE_DIR) or
                           ((LCO_OUT_DIRECTION or LCO_CHANGE_DIR) shl 2);
    r.dx := seg(s_port_dir);
    r.di := ofs(s_port_dir);
    intr(LC055, r);                           (* PORT_DIRECTION            *)
```

```
                                        (* przygotowanie stalych pol       *)
                                        (* rekordow opisow zlecen           *)
    s_port_read.LCO_PRMODULE := modulenum;
    s_port_read.LCO_PRPORT := 1;
    s_port_read.LCO_PRLATCH := LCO_LATCH_OFF;
    s_port_write.LCO_PWMODULE := modulenum;
    s_port_write.LCO_PWPORT := 2;
                          (* 20-krotny odczyt / wyswietlenie / zapis *)
    for i := 1 to 20 do
      begin
        r.dx := seg(s_port_read);
        r.di := ofs(s_port_read);
        intr(LC055, r);                          (* PORT_READ           *)
        for j := 1 to 8 do                 (* wyswietlenie kolejnych bitow  *)
            write((s_port_read.LCO_PRDATA shr (8 - j)) and 1, ' ');
        writeln;
        s_port_write.LCO_PWDATA := s_port_read.LCO_PRDATA;
        r.dx := seg(s_port_write);
        r.di := ofs(s_port_write);
        intr(LC055, r);                          (* PORT_WRITE          *)
        delay(200);
      end;

    pressanykey;
end;

(*****************************************************************************)
(* Przeznaczenie:                                                            *)
(*   Zapis na port zaprogramowany jako wejsciowy - blad.                     *)
(*****************************************************************************)
procedure writeportfail;
begin
    writeln('---------- Przesylanie danych na port wejsciowy');
    writeln;
                              (* zaprogramowanie kierunku pracy portow:  *)
                              (*   1 (A) : wejscie                       *)
                              (*   pozostale - bez zmiany                *)
    s_port_dir.LCO_PMODULE := modulenum;
    s_port_dir.LCO_PDIR := (LCO_IN_DIRECTION or LCO_CHANGE_DIR);
    r.dx := seg(s_port_dir);
    r.di := ofs(s_port_dir);
    intr(LC055, r);                              (* PORT_DIRECTION      *)

    s_port_write.LCO_PWMODULE := modulenum;
    s_port_write.LCO_PWPORT := 1;
    s_port_write.LCO_PWDATA := 0;
    r.dx := seg(s_port_write);
    r.di := ofs(s_port_write);
    intr(LC055, r);                              (* PORT_WRITE          *)
    if s_port_write.LCO_STATUS <> LCO_OK then
        drivererror(s_port_write.LCO_STATUS);

    pressanykey;
end;

(*****************************************************************************)
(* Przeznaczenie:                                                            *)
(*   Sprawdzenie obecnosci drivera.                                          *)
```

```
(* Sposob:                                                                *)
(*   Przez probe otwarcia urzadzenia o nazwie okreslonej przez driver.    *)
(* Parametry:                                                             *)
(*   name - nazwa urzadzenia                                              *)
(* Wartosc:                                                               *)
(*   TRUE  - driver jest zainstalowany                                    *)
(*   FALSE - driver nie jest zainstalowany                                *)
(************************************************************************)
function driverinstalled(name: string) : boolean;
var
    hd: text;
begin
{$I-}
    assign(hd, name);
    reset(hd);
    if IOResult <> 0 then
        driverinstalled := false
    else
        begin
            close(hd);
            driverinstalled := true
        end;
{$I+}
end;

(************************************************************************)
(* Przeznaczenie:                                                       *)
(*   Rozpoznanie, czy driver jest zainstalowany.                        *)
(* Sposob:                                                              *)
(*   Za pomoca ogolnej funkcji driverinstalled. Jezeli zaden driver nie *)
(*   jest zainstalowany to po wyswietleniu wlasciwego komunikaru program*)
(*   konczy prace.                                                      *)
(************************************************************************)
procedure checkdriver;
begin
    clrscr;
    gotoxy(1, 1);
    if not driverinstalled('LC055^^^') then
        begin
            writeln('Driver nie jest zainstalowany!');
            halt;
        end;
end;

(************************************************************************)

begin
    initprogram;              (* inicjalizacja programu                 *)
    checkdriver;              (* rozpoznanie zainstalowanego driver'a   *)
    askdriver;                (* odpytanie driver'a o konfiguracje      *)
    displayconfiguration;     (* wyswietlenie konfiguracji badanego     *)
                              (* modulu                                 *)
    readport;                 (* odczyt portu cyfrowego                 *)
    writeportfail;            (* zapis na port wejsciowy - blad         *)
    testinterrupts;           (* test przerwan przychodzacych z modulu  *)
    quit;                     (* zakonczenie programu                   *)
end.
```

ambex®

D O D A T E K E

do dokumentacji driver´ów modułów

LC-055-PIO, LC-055-DCU i LC-8255

AMBEX-55.ASM

struktury danych i stałe dla assemblera

Wydanie: lipiec 1992

```
                SUBTTL  Struktury parametrow driver'a
                PAGE +

                COMMENT #
Kazde zlecenie zaczyna sie od nizej opisanych pol.
#
LCO_HEADER      STRUC
LCO_CODE        DB      ?               ;kod funkcji
LCO_STATUS      DB      ?               ;kod bledu
LCO_HEADER      ENDS

                SUBTTL  Stale
                PAGE +

;-------------------------------------------------------------------------

LC055           EQU     98h     ;numer przerwania obslugiwanego przez driver

;-------------------------------------------------------------------------

; Kody funkcji driver'a:

MODULE_INIT_55                  EQU     0
GET_TOTAL_CONFIGURATION_55      EQU     1
GET_MODULE_CONFIGURATION_55     EQU     2
PORT_DIRECTION                  EQU     3
PORT_WRITE                      EQU     4
PORT_READ                       EQU     5
PORT_LATCH                      EQU     6
PORT_RESET                      EQU     7
CTC_WRITE_55                    EQU     8
CTC_GATE                        EQU     9
CTC_READ_55                     EQU     10
INTERRUPT_MASK                  EQU     11
INTERRUPT_LEVEL                 EQU     12
INTERRUPT_SERVICE_55            EQU     13
INTERRUPT_RESET                 EQU     14
READ_STATUS                     EQU     15
LEAVE_DRIVER_55                 EQU     16

;-------------------------------------------------------------------------
;
;kody odpowiedzi driver'a karty
;
;-------------------------------------------------------------------------
;
;ostrzezenia:
;
LCO_OK                  EQU     0       ;OK
LCO_NON_EX_MOD          EQU     1       ;nie istniejacy(e) modul(y)
LCO_NO_LATCH_WARN       EQU     2       ;dany modul nie moze zatrzaskiwac

;-------------------------------------------------------------------------
;
```

```
;bledy:
;
LCO_UNKN_FUNC           EQU     -1      ;nieznany kod funkcji
LCO_NO_MODULE           EQU     -2      ;brak modulu(ow)
LCO_NOT_OUTPORT         EQU     -3      ;port nie jest wyjsciowy
LCO_NONEX_DEV           EQU     -4      ;nie istnieje urzadzenie o tym numerze
LCO_NOT_INPORT          EQU     -5      ;port nie jest wejsciowy
LCO_NO_LATCH            EQU     -7      ;dany modul nie moze zatrzaskiwac
LCO_CTC_NOT_PROGRAMMED  EQU     -8      ;zapis licznika do niezaprogramowanego
                                        ;kanalu
LCO_BAD_PROC            EQU     -19     ;bledny adres procedury obslugi przerwan
LCO_NOT_SUPPORTED       EQU     -24     ;dla danego modulu funkcja nie jest
                                        ;realizowana
LCO_BAD_CTC_MODE        EQU     -25     ;bledny tryb pracy CTC
LCO_NO_IRQ              EQU     -29     ;z danum modulem nie jest zwiazane
                                        ;zadne przerwanie
LCO_NOT_FULLY_SUP       EQU     -30     ;zadany tryb wykonania funkcji nie jest
                                        ;realizowany dla danego typu modulu lub
                                        ;funkcja w opracowaniu
LCO_INTR_INST           EQU     -33     ;proba powtornej instalacji przerwania
LCO_REJECTED            EQU     -38     ;za wiele rownoleglych wejsc do driver'a

;-------------------------------------------------------------------------------
;
LCO_MAX_CTC             EQU     6       ;maksymalna mozliwa liczba kanalow CTC
LCO_MAX_PORTS           EQU     6       ;maksymalna mozliwa liczba portow
LCO_MAX_INT_LINES       EQU     4       ;maksymalna liczba linii przerywajacych
;
;-------------------------------------------------------------------------------
;
; rekordy opisu poszczegolnych zlecen
;
;===============================================================================
; MODULE_INIT
;
LCO_INIT_55     STRUC
                DB      TYPE LCO_HEADER DUP ()
LCO_IMODULE     DB      ?               ;mapa modulow
LCO_INIT_55     ENDS
;
; Mapa modulow
;       - - - - D C B A
;       0 0 0 0 x x x x
; x = 1 zeruj modul, = 0 nie zeruj modulu
;

;===============================================================================
; GET_TOTAL_CONFIGURATION
;
LCO_TOTAL_55    STRUC
                DB      TYPE LCO_HEADER DUP ()
                ;PARAMETRY WYJSCIOWE
LCO_TONF        DB      ?       ;informacja o zainstalowanych modulach
LCO_TCTC        DB      ?       ;liczba dostepnych kanalow CTC
LCO_TPORT       DB      ?       ;liczba dostepnych portow
LCO_TOTAL_55    ENDS
;
```

```
; Format bajtu konfiguracji modulow
;       - - - - D C B A
;       0 0 0 0 x x x x
; x = 1 modul zaistalowany, = 0 modulu nie ma
;

;=============================================================================
; GET_MODULE_CONFIGURATION
;
LCO_MODULE_55   STRUC
                DB      TYPE LCO_HEADER DUP ()
LCO_MMODULE     DB      ?               ;numer modulu
                ;PARAMETRY WYJSCIOWE
LCO_MTYPE       DB      ?       ;typ modulu
LCO_MBASE       DW      ?       ;adres bazowy rejestrow modulu
LCO_MPORT       DB      ?       ;liczba dostepnych portow
LCO_MCTC        DB      ?       ;liczba dostepnych kanalow CTC
LCO_MCLOCK      DW      LCO_MAX_CTC DUP (?)     ;tablica czestotliwosci zegara
                                                ;CTC w kHz
LCO_MCTCGATE    DB      ?       ;bramki kanalow CTC
LCO_MINT        DB      ?       ;numer przerwania (programowy)
LCO_MINTSRC     DW      ?       ;okreslenie zrodel przerwan
LCO_MINTLEV     DB      ?       ;poziom przerywajacy
LCO_MINTMSK     DB      ?       ;maski przerwan
LCO_MODULE_55   ENDS
;
;
; kody typow modulow
;
; 1 - LC-8255
; 2 - LC-055-PIO
; 3 - LC-055-DCU
;
LC_8255         EQU     1
LC_055_PIO      EQU     2
LC_055_DCU      EQU     3
;
;
; kody zrodla przerwania
;
; 0 - brak przerwania
; 1 - sygnal zewnetrzny
; 2 - CTC
;
LCO_NO_INT      EQU     0
LCO_INTSRC_EXT  EQU     1
LCO_INTSRC_CTC  EQU     2
;

;=============================================================================
; PORT_DIRECTION
;
LCO_PORT_DIRECTION      STRUC
                        DB      TYPE LCO_HEADER DUP ()
LCO_PMODULE             DB      ?       ;numer modulu
LCO_PDIR                DD      ?       ;kierunki pracy portow
LCO_PORT_DIRECTION      ENDS
;
```

```
;
; kody kierunkow pracy portow
;
; w kazdej parze bitow:
;  b1 - kierunek portu:
;     0 - port wejsciowy
;     1 - port wyjsciowy
;  b2 - tryb:
;     0 - nie zmieniaj kierunku
;     1 - zmien kierunek zgodnie z bitem b1
;
LCO_IN_DIRECTION        EQU     0
LCO_OUT_DIRECTION       EQU     1
LCO_CHANGE_DIR          EQU     2
;

;=============================================================================
; PORT_WRITE
;
LCO_PORT_WRITE  STRUC
                DB      TYPE LCO_HEADER DUP ()
LCO_PWMODULE    DB      ?       ;numer modulu
LCO_PWPORT      DB      ?       ;numer portu
LCO_PWDATA      DB      ?       ;dana do wyslania
LCO_PORT_WRITE  ENDS
;

;=============================================================================
; PORT_READ
;
LCO_PORT_READ   STRUC
                DB      TYPE LCO_HEADER DUP ()
LCO_PRMODULE    DB      ?       ;numer modulu
LCO_PRPORT      DB      ?       ;numer portu
LCO_PRLATCH     DB      ?       ;zatrzask LCO_LATCH_ON/LCO_LATCH_OFF
LCO_PRDATA      DB      ?       ;dana odczytana
LCO_PORT_READ   ENDS
;

;=============================================================================
; PORT_LATCH
;
LCO_LATCH       STRUC
                DB      TYPE LCO_HEADER DUP ()
LCO_PLMODULE    DB      ?       ;numer modulu
LCO_PLMODE      DB      ?       ;tryb pracy
LCO_LATCH       ENDS
;
; Tryby pracy:
;
;    1 - zatrzasnij dane
;    0 - zwolnij
;
LCO_LATCH_ON    EQU     0
LCO_LATCH_OFF   EQU     1
;
```

```
;=============================================================================
; PORT_RESET
;
LCO_PORT_RESET  STRUC
                DB      TYPE LCO_HEADER DUP ()
LCO_PRSMODULE   DB      ?       ;numer modulu
LCO_PORT_RESET  ENDS
;

;=============================================================================
; CTC_MODE
;
LCO_CTC_WRITE_55        STRUC
                        DB      TYPE LCO_HEADER DUP ()
LCO_CMODULE             DB      ?       ;numer modulu
LCO_CMODE               DB      ?       ;tryb pracy funkcji
LCO_CCTRL               DB      ?       ;tryb pracy CTC
LCO_CWDATA              DW      ?       ;wartosc licznika
LCO_CTC_WRITE_55        ENDS
;
; tryb pracy funkcji
;
; bit 1 - zaprogramuj tryb pracy kanalu
; bit 2 - zaladuj nowa wartosc licznika
;
LCO_SET_CTC_MODE        EQU     1
LCO_SET_COUNTER_VALUE   EQU     2
;

;=============================================================================
; CTC_GATE
;
LCO_CTC_GATE    STRUC
                DB      TYPE LCO_HEADER DUP ()
LCO_CGMODULE    DB      ?       ;numer modulu
LCO_CGGATE      DB      ?       ;maska aktywnych kanalow
LCO_CTC_GATE    ENDS
;
;       - - - - - 3 2 1
;       - - - - - x x x
; x = 0 kanal aktywny, x = 1 kanal nieaktywny
;

;=============================================================================
; CTC_READ
;
LCO_CTC_READ_55 STRUC
                DB      TYPE LCO_HEADER DUP ()
LCO_CRMODULE    DB      ?       ;numer modulu
LCO_CRCOUNT     DB      ?       ;numer kanalu
LCO_CRDATA      DW      ?       ;odczytana dana
LCO_CTC_READ_55 ENDS
;
```

```
;=============================================================================
; INTERRUPT_MASK
;
LCO_INT_MASK    STRUC
                DB      TYPE LCO_HEADER DUP ()
LCO_IMMODULE    DB      ?       ;numer modulu
LCO_IMASK       DB      ?       ;maska przerwan
LCO_INT_MASK    ENDS
;
; Maska przerwan:
; LC-055-DCU
;       - - - - 4 3 2 1
;       - - - - x x x x
; x = 1 - zezwolenie na przerwanie przez dana linie przrywajaca
;
; LC-055-PIO
;       - - - - - - - 1
;       - - - - - - - x
; x = 1 - zezwolenie na przerwanie
;

;=============================================================================
; INTERRUPT_LEVEL
;
LCO_INT_LEVEL   STRUC
                DB      TYPE LCO_HEADER DUP ()
LCO_ILMODULE    DB      ?       ;numer modulu
LCO_ILEVEL      DB      ?       ;poziom przerywajacy na poszczegolnych liniach
LCO_INT_LEVEL   ENDS
;
; Poziom przerywajacy
;
;       - - - - 4 3 2 1
;       - - - - x x x x
; x = 1 - poziom wysoki, x = 0 - poziom niski
;

;=============================================================================
; INTERRUPT_SERVICE
;
LCO_INT_SERVICE STRUC
                DB      TYPE LCO_HEADER DUP ()
LCO_ISMODULE    DB              ?       ;numer modulu
LCO_IPROC       DD              ?       ;adres procedury obslugi
LCO_INT_SERVICE ENDS
;

;=============================================================================
; INTERRUPT_RESET
;
LCO_INT_RESET   STRUC
                DB      TYPE LCO_HEADER DUP ()
LCO_IRMODULE    DB      ?       ;numer modulu
LCO_INT_RESET   ENDS
;
```

```
;=============================================================================
; READ_STATUS
;
LCO_READ_STATUS STRUC
                DB      TYPE LCO_HEADER DUP ()
LCO_SMODULE     DB      ?       ;numer modulu
LCO_SDATA       DB      ?       ;odczytany status
LCO_SRESRVD     DB      ?       ;pole zarezerwowane
LCO_READ_STATUS ENDS
;
; bity statusu:
;
LCO_INT1_INTERRUPTED    EQU     1       ;linia INT 1 zglosila przerwanie
LCO_INT2_INTERRUPTED    EQU     2       ;linia INT 2 zglosila przerwanie
LCO_INT3_INTERRUPTED    EQU     4       ;linia INT 3 zglosila przerwanie
LCO_INT4_INTERRUPTED    EQU     8       ;linia INT 4 zglosila przerwanie
LCO_COMMON_INTERRUPT    EQU     16      ;jedna z linii zglosila przerwanie
LCO_CTC_OUT0            EQU     32      ;stan linii OUT kanalu 0
LC1_CTC_OUT1            EQU     64      ;stan linii OUT kanalu 1
LCO_CTC_OUT2            EQU     128     ;stan linii OUT kanalu 2
;

;=============================================================================
; LEAVE_DRIVER
;
LCO_LEAVE_55    STRUC
                DB      TYPE LCO_HEADER DUP ()
LCO_LEAVE_55    ENDS
;=============================================================================

;=============================================================================
;
; maski bitow poszczegolnych modulow
;
LCO_MODAMAP     EQU     1       ;modul A
LCO_MODBMAP     EQU     2       ;modul B
LCO_MODCMAP     EQU     4       ;modul C
LCO_MODDMAP     EQU     8       ;modul D
;

;
; kody modulow
; 1 - modul A, 2 - modul B, 3 - modul C, 4 - modul D
;
LCO_MODA        EQU     1
LCO_MODB        EQU     2
LCO_MODC        EQU     3
LCO_MODD        EQU     4
;
```