# ROZDZIAŁ DWUDZIESTY DRUGI: **PORTY SZEREGOWE**

Standard komunikacji szeregowej RS-232 jest prawdopodobnie najpopularniejszym schematem komunikacji szeregowej na świecie. Chociaż cierpi z powodu wielu wad, szybkość jest jedną z nich , jego zastosowanie jest rozpowszechnione i są dosłownie tysiące urządzeń jakie możemy podłączyć do PC przy użyciu interfejsu RS-232. PC wspiera do czterech urządzeń kompatybilnych z RS-232 używając COM1:, COM2:, COM3: i COM4: dla tych, którzy potrzebują więcej urządzeń szeregowych (np. sterowanie elektronicznej komputerowej tablicy ogłoszeń [BBS], możemy kupić urządzenia, które pozwalają dodać 16 lub więcej ,portów szeregowych do PC. Ponieważ większość PC ma tylko jeden lub dwa porty szeregowe, skoncentrujemy się w tym rozdziale jak użyć COM1: i COM2:.

Chociaż, teoretycznie, oryginalnie zaprojektowany PC pozwolił projektantom systemu zaimplementować porty komunikacji szeregowej używając żądanego sprzętu, wiele z dzisiejszego oprogramowania wykonuje komunikację szeregową poprzez bezpośredni kontakt z chipem komunikacji szeregowej 8250 SCC. Wprowadza to takie same komplikacje z kompatybilnością jakie były kiedy komunikowaliśmy się bezpośrednio ze sprzętem portu równoległego. Jednakże, podczas gdy BIOS dostarcza doskonałego interfejsu dla portu równoległego, wspierając wszystko co życzylibyśmy sobie zrobić poprzez bezpośrednie dotarcie do sprzętu, wsparcie szeregowe nie jest dobre. Dlatego też jest powszechną praktyką omijanie funkcji BIOS 1h i sterowanie chipem 8250 SCC bezpośrednio tak aby program mógł uzyskać dostęp do każdego bitu każdego rejestru w 8250.

Być może największym problemem z kodem BIOS jest to, że nie wspiera przerwań. Chociaż oprogramowanie sterujące portami równoległymi rzadko używa układów I/O sterowanych przerwaniami, często znajduje się oprogramowanie, które dostarcza podprogramów obsługi przerwań dla portów szeregowych. Ponieważ BIOS nie dostarcza takich podprogramów, oprogramowanie które zechce użyć układów I/O sterowanych przerwaniami będzie musiało komunikować się bezpośrednio z 8250 i omijać BIOS. Dlatego też pierwsza część tego rozdziału będzie omawiała chip 8250.

Manipulowanie portem szeregowym nie jest trudne. Jednakże, 8250 SCC zawiera wiele rejestrów i dostarcza wielu cech. Dlatego też, zabiera wiele kodu sterowanie każdą cechą tego chipu. Na szczęście nie musimy pisać tego kodu sami. Biblioteka Standardowa UCR dostarcza doskonałego zbioru podprogramów, które pozwalają sterować 8250. Druga część tego rozdziału będzie przedstawiać kod z Biblioteki Standardowej jako przykład jak oprogramować każdy rejestr w 8250 SCC.

## 22.1 CHIP KOMUNIKACJI SZEREGOWEJ 8250

8250 i kompatybilne chipy (jak urządzenia 16450 i 16550) dostarczają dziewięciu rejestrów I/O. Pewne urządzenia kompatybilne w górę(np. 16450 i 16550) dostarczają również dziesięciu rejestrów. Rejestry te konsumują osiem portów I/O adresowanych w przestrzeni adresowej PC. Sprzęt i lokacje adresów dla tych urządzeń jest następująca:

| Port | Fizyczny adres bazowy (w hex) | Zmienna BIOS zawierająca adres fizyczny |
|---|---|---|
| COM1: | 3F8 | 40:0 |
| COM2: | 2F8 | 40:2 |

Tablica 81; Adresy portów COM

Podobnie jak w portach równoległych PC możemy wymieniać COM1: i COM2: z poziomu programu poprzez wymianę ich adresów bazowych w zmiennych BIOS 40:0i 40:2. Jednakże, oprogramowanie które zmierza bezpośrednio do sprzętu, zwłaszcza podprogramy obsługi przerwań dla portów szeregowych, muszą działać z adresami sprzętowymi, nie adresami logicznymi. Dlatego też zawsze będziemy oznaczać adres bazowy I/O 3F8h kiedy omawiamy COM1: w tym rozdziale. Podobnie zawsze oznaczamy adres bazowy 2F8h kiedy omawiamy COM2: w tym rozdziale.

Adres bazowy jest pierwszym z ośmiu lokacji I/O konsumowanych przez 8250 SCC. Dokładne zadanie tych ośmiu lokacji I/O pojawiają się w poniższej tablicy:

| Adresy I/O (w hex) | Opis |
|---|---|
| 3F8/2F8 | Rejestr danych odbiór/ przekazanie. Również najmniej znaczący bajt rejestru zatrzasku dzielenia szybkości transmisji |
| 3F9/2F9 | Rejestr zezwalający na przerwania/ Również bardziej znaczący bajt rejestru zatrzasku dzielenia szybkości transmisji |
| 3FA/2FA | Rejestr identyfikacji przerwań (tylko odczyt) |
| 3FB/2FB | Rejestr sterowania łączem |
| 3FC/2FC | Rejestr sterowania modemem |
| 3FD/3FD | Rejestr stanu linii (tylko odczyt) |
| 3FE/2FE | Rejestr stanu modemu (tylko odczyt) |
| 3FF/2FF | Rejestr rzutowania odbioru (tylko odczyt, nie dostępny na oryginalnym PC) |

Tablica 82: Rejestry 8250 SCC

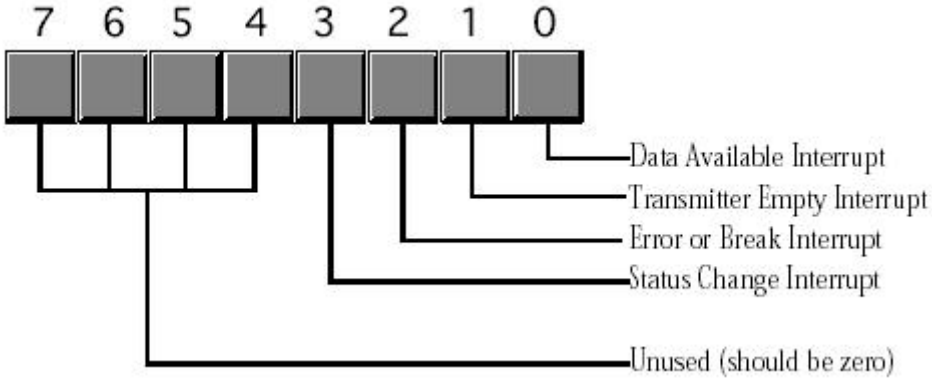
Kolejne sekcje opisują zadania każdego z tych rejestrów

### 22.1.1 REJESTR DANYCH (REJESTR PRZESŁANIA / ODBIORU)

Rejestr danych jest w rzeczywistości dwoma oddzielnymi rejestrami: rejestr przesłania i rejestr odbioru. Wybieramy rejestr przesłania poprzez wpisanie adresu I/O 3F8h lub 2F8h, wybieramy rejestr odbioru poprzez odczyt z tych adresów. Zakładając, że rejestr przesłania jest pusty zapis do rejestru przesłania zaczynamy transmisję danych poprzez linie szeregową. Zakładając, że rejestr odbioru jest pełny, odczytanie rejestru odbioru zwraca tą daną. Aby określić czy nadajnik jest pusty lub odbiornik jest pełny, zobacz rejestr stanu lini. Zauważmy, że rejestr dzielenia szybkości transmisji dzieli ten adres I/O z rejestrami odbioru i przesłania. Proszę zobaczyć „Dzielenie szybkości transmisji" i „Rejestr sterowania łączem" po więcej informacji o podwójnym zastosowaniu tych lokacji I/O.

### 22.1.2 REJESTR ZEZWALAJĄCY NA PRZERWANIA (IER)

Kiedy działamy w trybie przerwań, 8250 SCC dostarcza czterech źródeł przerwań: przerwanie odbioru znaku, przerwanie pustego nadajnika, przerwanie błędu komunikacji i przerwanie zmiany statusu. Możemy indywidualnie zezwalać lub blokować te źródła przerwań poprzez wpisanie jedynek lub zer do 8250 IER (Rejestr zezwalający na przerwania). Wpisując zero do odpowiedniego bitu blokujemy to szczególne przerwanie. Wpisując jedynkę zezwalamy na to przerwanie. Rejestr ten jest do odczytu / zapisu, więc możemy przepytywać aktualne ustawienie w dowolnym czasie (na przykład, jeśli chcemy zamaskować poszczególne przerwanie bez wpływania na inne). Rozkład tego rejestru jest następujący :




Lokacja rejestru zezwalającego na przerwania jest również wspólna z rejestrem dzielenia szybkości transmisji.

## 22.13 DZIELNIK SZYBKOŚCI TRANSMISJI

Rejestr dzielnika szybkości transmisji jest 16 bitowym rejestrem który dzieli lokacje I/O 3F8h/2F8h i 3F9h/2F9h z rejestrem danych i zezwolenia na przerwania. Bit siedem rejestru sterowania łączem wybiera rejestr dzielenia lub rejestr danych / zezwolenia na przerwanie.

Rejestr dzielenia szybkości transmisji pozwala nam wybrać szybkość transmisji danych (poprawnie nazywanej bity na sekundę lub bps lub baud) Poniższa tablica pokazuje wartości jakie powinniśmy zapisać do tych rejestrów aby sterować szybkością transmisji / odbioru:

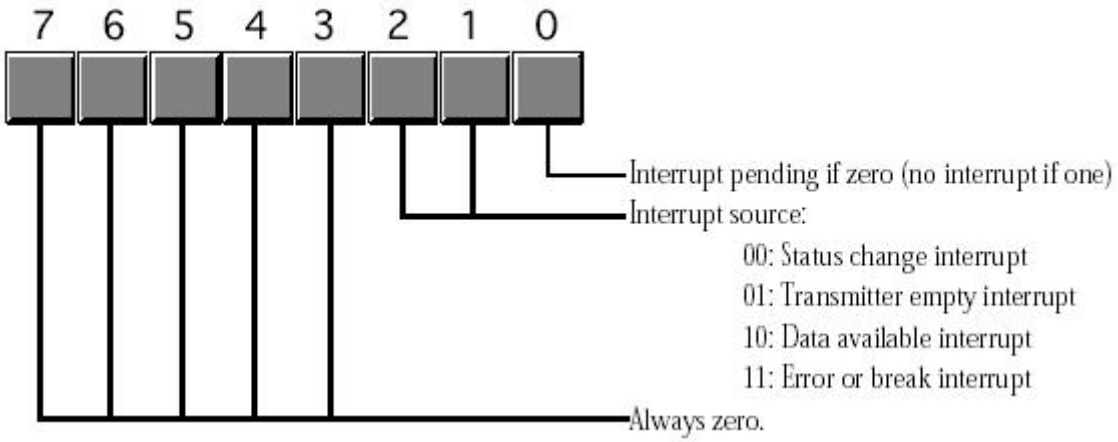
| Bity na sekundę | Wartość 3F9/2F9 | Wartość 3F8/2F8 |
|---|---|---|
| 110 | 4 | 17h |
| 300 | 1 | 80h |
| 600 | 0 | C0h |
| 1200 | 0 | 60h |
| 1800 | 0 | 40h |
| 2400 | 0 | 30h |
| 3600 | 0 | 20h |
| 4800 | 0 | 18h |
| 9600 | 0 | 0Ch |
| 19,2K | 0 | 6 |
| 38,4K | 0 | 3 |
| 56K | 0 | 1 |

Tablica 83 Wartości rejestru dzielenia szybkości transmisji

Powinniśmy działać przy szybkościach większych niż 19,2K na szybkich PC z wysoko wydajnym SCC'esem (np. 16450 lub 16550). Na szczęście powinniśmy użyć kabli najwyższej jakości i trzymać nasze kable bardzo krótkie kiedy działamy przy wysokich szybkościach.

## 22.1.4 REJESTR IDENTYFIKACJI PRZERWANIA

Rejestr identyfikacji przerwania jest rejestrem tylko do odczytu, który określa czy przerwanie oczekuje i które z czterech źródeł wymaga uwagi. Ten rejestr ma następujący rozkład:




Ponieważ IIR może tylko raportować o jednym przerwaniu w czasie, a jest prawdopodobieństwo, że są dwa lub więcej przerwania oczekujące, 8250 SCC wprowadza priorytety przerwań. Przerwanie źródła 00 (zmiana statusu) ma najniższy priorytet a przerwanie źródła 11 (błąd lub przerwa) mają priorytet najwyższy ,tj .numer źródła przerwania dostarcza priorytetu (z trzema będącymi najwyższego priorytetu)

Poniższa tablica opisuje źródła przerwań i jak „wyczyścić" wartość przerwania w IIR. Jeśli dwa przerwania oczekują a my obsługujemy najwyższe przerwanie, 8250 SCC zamienia wartość IIR z identyfikatorem kolejnego źródła przerwania o najwyższym priorytecie

| Priorytet | Wartość ID | Przerwanie | Powód | Wyzerowanie |
|---|---|---|---|---|
| Najwyższy | 11b | Błąd lub przerwanie | Błąd przepełnienia, parzystości, synchronizacji ramki lub przerwanie break | Odczytanie rejestru stanu łącza |
| Kolejny z najwyższych | 10b | Dostępna dana | | Odczyt rejestru odbioru |
| Kolejny z najniższych | 01b | Pusty przekaźnik | | Odczytanie IIR (z ID przerwania 01b) lub zapis rejestru danych |
| Najniższy | 00 | Stan modemu | | Odczyt rejestru stanu modemu |

Tablica 84: Powody przerwania i funkcje zwalniające

Jest jeden interesujący punkt do odnotowania o organizacji IIR: rozkład bitów dostarcza dogodnego sposobu do przekazania sterowania do właściwej sekcji podprogramu obsługi przerwania SCC. Rozważmy poniższy kod:

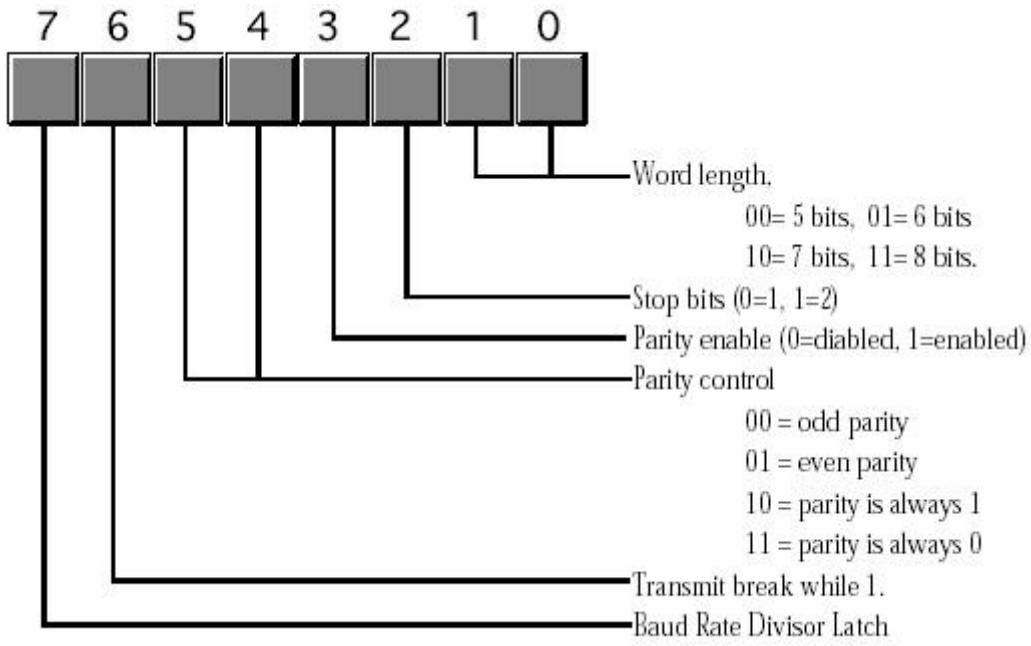
```
                -
                -
                -
                in      al., dx                 ;odczyt IIR
                mov     bl, al.
                mov     bh, 0
                jmp     HnadlerTbl [bx]
HandlerTbl      word    RLSHandler, RDHandler,TEHandler, MSHandler
```

Kiedy wystąpi przerwanie, bit zero IIR będzie zerem. Kolejne dwa bity zawierają numer źródła przerwania a bardziej znaczące pięć bitów jest zerami. Pozwala to nam użyć wartości IIR jako indeksu do tablicy wskaźników właściwego podprogramu obsługi, jak demonstruje powyższy kod.

---

22.1.5 REJESTR STEROWANIA ŁĄCZEM

Rejestr sterowani łączem pozwala nam określić parametry transmisji dla SCC. Obejmuje to ustawienie rozmiaru danej, liczby bitów zatrzymujących, parzystości, wymuszenia break i wybrania rejestru dzielnika szybkości transmisji . Rejestr sterowania łączem wygląda następująco:




8250 SCC może transmitować daną szeregową jako grupy pięciu, sześciu, siedmiu lub ośmiu bitów. Większość nowoczesnych systemów komunikacji używa siedmiu lub ośmiu bitów do transmisji ( my potrzebujemy siedmiu bitów do przekazania ASCII, ośmiu bitów do przekazania danej binarnej). Domyślnie, większość aplikacji przekazuje dane używając ośmio bitowej danej. Oczywiście, zawsze odczytujemy osiem bitów z rejestru odbiorczego; 8250 SCC ustawia wszystkie bardziej znaczące bity na zero jeśli odbieramy mniej niż osiem bitów. Zauważmy, że jeśli przekazujemy tylko znaki ASCII,, komunikacja szeregowa będzie działała około 10% szybciej z siedmio bitową transmisją zamiast transmisji ośmio bitowej. Jest to ważna rzecz do zapamiętania, jeśli sterujemy oboma końcami kabla szeregowego . Z drugiej strony, zazwyczaj będziemy łączyć urządzenia które maja stałą długość słowa, więc będziemy musieli oprogramować SCC specjalnie do dopasowania tego urządzenia.

Szeregowa transmisja danych składa się z bitu startowego, pięciu do ośmiu bitów danych i jednego lub dwóch bitów stopu. Bit startu jest specjalnym sygnałem, który informuje SCC (lub inne urządzenie), że dana przebywa na lini szeregowej. Bity stopu są, w gruncie rzeczy, pod nieobecność bitu startowego, dostarczając małą ilość czasu pomiędzy przybyciem kolejnych znaków na linię szeregową .Przez wybranie dwóch bitów stopu wprowadzamy dodatkowy czas pomiędzy transmisję kolejnego znaku. Niektóre inne starsze urządzenia mogą wymagać tego dodatkowego czasu lub się pogubią. Jednakże, prawie wszystkie nowoczesne urządzenia szeregowe zadowalają się pojedynczym bitem stopu. Dlatego też, powinniśmy zawsze oprogramować chip tylko jednym bitem stopu. Dodatkowy druki bit stopu zwiększa czas transmisji o około 10%.

Bity parzystości pozwalają nam włączyć lub wyłączyć parzystość lub wybrać tryb parzystości. Parzystość jest schematem detekcji błędu. Kiedy włączmy parzystość, SCC dodaje dodatkowy bit (bit parzystości) do transmisji. Jeśli wybieramy kontrolę nieparzystości, bit parzystości zawiera zero lub jeden aby suma mniej znaczącego bitu danej i bitu parzystości dało jeden. Jeśli wybieramy kontrolę parzystości, SCC tworzy bit parzystości taki, że mniej znaczący bit sumy parzystości i bitu danej to zero Wartość „parzystości zablokowanej" (10b i 11b) zawsze tworzy bit parzystości na zero lub jeden. Głównym celem bitu parzystości jest wykrycie możliwego błędu transmisji. Jeśli mamy długi, hałaśliwy lub zły kanał komunikacji szeregowej, jest możliwe zgubienie informacji podczas transmisji. Kiedy się to zdarzy, jest mało prawdopodobne, że suma bitów będzie pasowała do wartości parzystości. Węzeł odbiorczy może wykryć ten „błąd parzystości" i zaraportować błąd transmisji.

Możemy również użyć wartości zablokowanej parzystości (10b i 11b)do usunięcia tych ośmiu bitów i zawsze zamienić na zera lub jedynki podczas transmisji. Na przykład, kiedy przesyłamy osiem bitów znaków PC/ASCII do różnych systemów komputerowych, jest możliwe ,że zbiór rozszerzonych znaków PC (znaki których kod to 128 i większy) nie odwzorowywuje takiego samego znaku na maszynie przeznaczenia. Istotnie, wysyłając takie znaki możemy stworzyć problem na tej maszynie. Poprzez ustawienie rozmiaru słowa na

siedem bitów i włączenia parzystości i zablokowania dla zera możemy automatycznie usunąć wszystkie bardziej znaczące bity podczas transmisji, zamieniając je na zera. Oczywiście jeśli pojawiają się rozszerzone znaki, SCC będzie odwzorowywał je do możliwie nie związanych znaków ASCII, ale jest to użyteczna sztuczka, czasami.

Bit break transmisji, przerywa sygnał systemu zdalnego tak długo jak jest oprogramowany bit na tej pozycji. Nie powinniśmy pozostawić włączonego break podczas prób transmisji danej. Sygnał break pochodzi z czasów dalekopisów .Break jest podobne do ctrl-C lub ctrl-break z klawiatury PC. Powinien przerwać program uruchomiony na systemie zdalnym. Zauważmy, że SCC może wykryć nadchodzący sygnał break i wygenerować właściwe przerwanie, ale ten sygnał break pochodzi z systemu zdalnego, nie jest (bezpośrednio) połączony do wyjściowego sygnału break LCR.

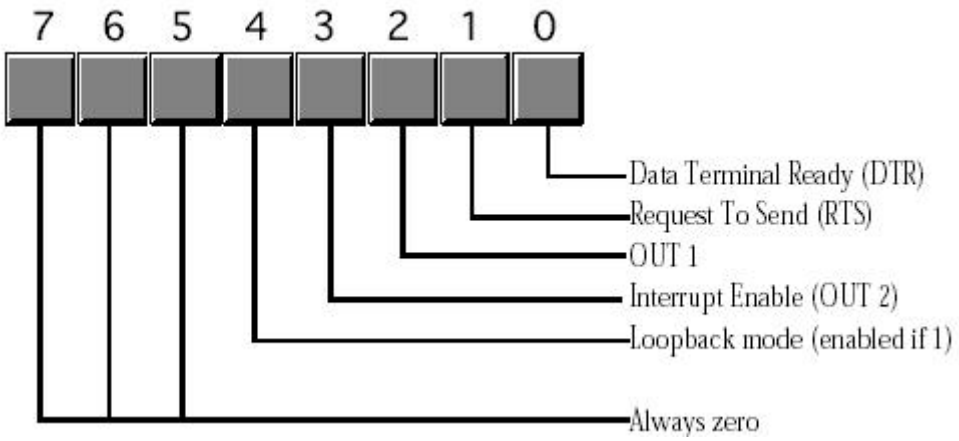
Bit siedem LCR jest bitem zatrzasku rejestru dzielnika szybkości transmisji. Kiedy bit ten zawiera jeden, lokacje 3F8h/2F8h i 3F9h/2F9h stają się rejestrem dzielnika szybkości transmisji. Kiedy zawiera zero, te lokacje I/O odpowiadają rejestrowi danych i rejestrowi zezwolenia na przerwania. Powinniśmy zawsze oprogramować ten bit zerem poza tym kiedy inicjalizujemy szybkość SCC

LCR jest rejestrem odczyt / zapis. Odczytując ten rejestr, LCR zwraca ostatnią wartość zapisaną do niego.

---

## 22.1.6 REJESTR STEROWANIA MODEMEM

Rejestr sterowania modemem 8250 zawiera pięć bitów, które pozwalają nam bezpośrednio sterować różnymi końcówkami wyjściowymi w 8250, jak również włączyć tryb pętli sprzężenia zwrotnego. Poniżej mamy ten rejestr:




8250 wyznacza drogę bitów DTR i RTS bezpośrednio do lin DTR i RTS w chipie 8250. Kiedy bity te są jedynkami, odpowiednie wyjścia są aktywne. Linie te są dwoma oddzielnymi liniami uzgodnień dla komunikacji RS-232.

Sygnał DTR jest porównywalny do sygnału busy. Kiedy węzeł lini DTR jest nie aktywny, inny węzeł nie ma prawa transmitować danej do niego. Linia DTR jest linią ręcznego uzgodnienia. Pojawia się jako linia DSR (gotowość do odbioru danych) po drugiej stronie kabla. Inne urządzenia musza wyraźnie sprawdzić swoje linie DSR aby zobaczyć czy mogą przesyłać dane. Schemat DTR / DSR jest głównie zaplanowany do wymiany potwierdzeń pomiędzy komputerami a modemami.

Linia RTS dostarcza drugiej postaci uzgodnienia. Odpowiadający jej sygnał to CTS (gotowość nadawcza). Protokół uzgodnień RTS/CTS jest zaplanowany głównie do bezpośredniego połączenia urządzeń takich jak komputery i drukarki. Możemy zapytać „dlaczego są dwa oddzielne, ale ortogonalne protokoły uzgodnień?" Powód jest taki, że RS-232C rozwija się od ponad stu lat (od dnia pierwszego telegrafu) i jest wynikiem łączenia różnych schematów przez te lata.

Out1 jest wyjściem ogólnego przeznaczenia w SCC, które mam bardzo małe zastosowanie w IBM PC. Niektóre złącza płyt łączą ten sygnał , inne pozostawiają go niepołączonymi. Generalnie bit ten nie ma funkcji w PC.

Bit zezwolenia na przerwania jest specyficzną pozycją w PC Normalnie jest to wyjście ogólnego przeznaczenia (OUT2) w 8250 SCC. Jednakże, IBM zaprojektował to wyjście jako przyłączone do zewnętrznej bramki włączającej lub wyłączającej wszystkie przerwania z SCC. Bit ten musi być oprogramowany jedynka, włączającą przerwania .Podobnie, musimy zapewnić, że ten bit zawiera zero jeśli nie używamy przerwań.

Bit pętli sprzężenia zwrotnego łączy rejestr przesyłający z rejestrem odbiorczym. Wszystkie dane wysyłane z przekaźnika bezpośrednio przychodzą do rejestru odbiorczego .Jest to użyteczne przy diagnostyce,

testowaniu oprogramowania i wykrywaniu chipu szeregowego. Zauważmy, niestety, że obwód sprzężenia zwrotnego nie będzie generował żadnego przerwania. Możemy użyć tej techniki tyko z odpytywaniem I/O.

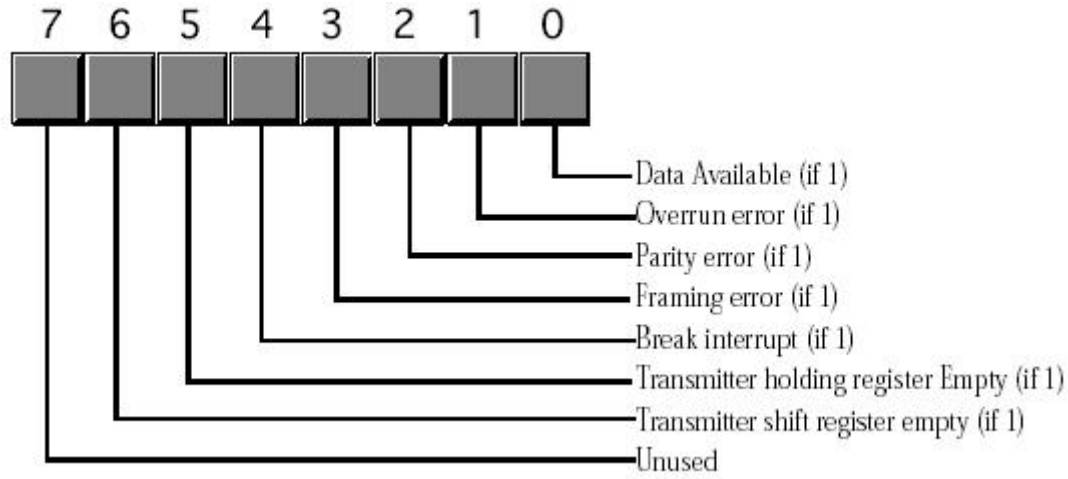
Pozostałe bity w MCR są zarezerwowane, powinny zawsze zawierać zero. Przyszłe wersje SCC (lub chipów kompatybilnych) mogą używać tych bitów do innych celów, ze stanem zero domyślnie (symulacja 8250).

MCR jest rejestrem odczyt/zapis. Odczytując ten rejestr, MCR zwraca ostatnią ,zapisaną do niego wartość.

---

22.1.7 REJESTR STANU ŁĄCZA (LSR)

Rejestr stanu łącza (LSR) jest rejestrem tylko do odczytu, który zwraca bieżący stan komunikacji. Rozkład bitów rejestrów:




Bit dostępnej danej jest ustawiony jeśli jest dostępna dana w rejestrze odbiorczym. Również generuje przerwanie. Odczyt danej w rejestrze odbioru ,czyści ten bit.

Rejestr odbiorczy 8250 może tylko przechowywać jedne bajt w czasie. Jeśli nadszedł bajt a program go nie odczytał a potem nadszedł drugi bajt, 8250 zatrze pierwszy bajt drugim .8250 SCC ustawi bit błędu przepełnienia kiedy to nastąpi/ Odczyt LSR czyści ten bit (po odczytaniu LSR). Błąd ten będzie generował błąd przerwania najwyższego priorytetu

8250 ustawia bit parzystości jeśli wykryje błąd parzystości kiedy odbiera bajt. Ten błąd wystąpi tylko jeśli mamy włączoną operację parzystości w LCR. 8250 resetuje ten bit po odczytaniu LSR. Kiedy wystąpi ten błąd, 8250 wygeneruje błąd przerwania.

Bit trzy jest bitem błędu ramkowania. Błąd ramkowania wystąpi jeśli 8250 odbiera znak bez prawidłowego bitu stopu. 8250 czyści ten bit po odczytaniu LSR. Błąd ten będzie generował błąd przerwania o najwyższym priorytecie.

8250 ustawia bit przerwania break kiedy odbiera sygnał break z urządzenia przesyłającego. Również generuje błąd przerwania. Odczyt LSR czyści ten bit.

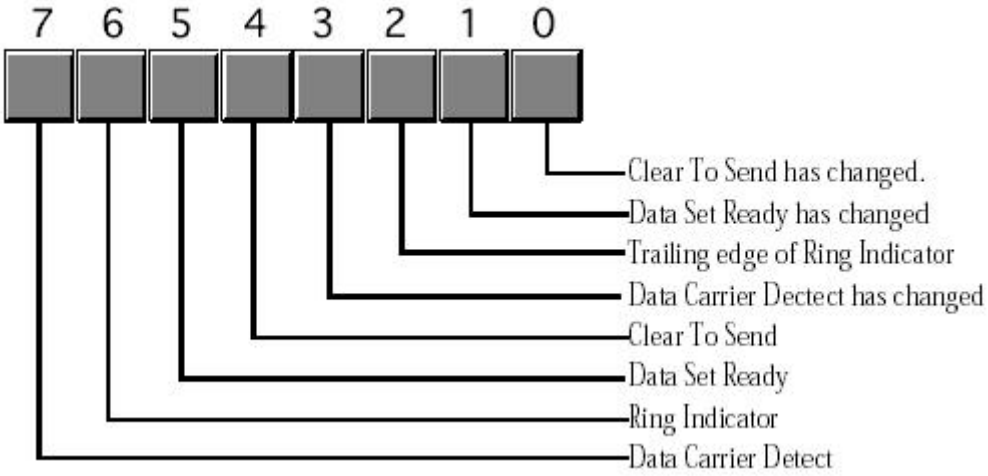
8250 ustawia bit piąty, przekaźnik przechowuje bit pustego rejestru, kiedy można zapisać inny znak do rejestru danych. Zauważmy, że 8250 w rzeczywistości ma dwa rejestry związane z przekaźnikiem. Rejestr przesunięcia przekaźnika zawiera dane aktualnie przesuwane na linię szeregową. Rejestr przechowujący przekaźnika przechowuje wartości, które 8250 zapisuje do rejestru przesunięcia, kiedy ten kończy przesunięcie znaku. Bit pięć wskazuje ,że rejestr przechowujący jest pusty a 8250 może zaakceptować inny bajt. Zauważmy, że 8250 może jeszcze przesuwać znak równolegle z tą operacją. 8250 może generować przerwanie kiedy rejestr przechowujący przekaźnika jest pusty. Odczyt LSR lub zapis do rejestru danych czyści ten bit.

8250 ustawia bit sześć kiedy oba, rejestr przechowujący i rejestr przesunięcia są puste. Bit ten jest czyszczony kiedy inny rejestr zawiera daną

---

22.1.8 REJESTR STANU MODEMU (MSR)

Rejestr stanu modemu (MSR) raportuje stan sygnału uzgodnienia i innych sygnałów modemu. Cztery bity dostarczają wartości natychmiastowych tych sygnałów, 8250 ustawia inne cztery bity jeśli któryś z tych sygnałów zmienił się od ostatniego czasu odpytania MSR przez CPU. Rozkład MSR:




Bit gotowości do wysłania (bit # 4) jest sygnałem uzgodnienia. Jest zazwyczaj podłączony do sygnału RTS (zgłoszenie gotowości do nadawania) w urządzeniu odbiorczym. Kiedy to zdalne urządzenie zaznaczy swoją linię RTS, transmisja danych może mieć miejsce.

Bit gotowości do odbioru (bit # 5) jest jedynka jeśli urządzenie zdalne nie jest zajęte. To wejście jest generalnie podłączone do lini gotowości do wysłania danych (DTR) w urządzeniu zdalnym

Chip 82250 ustawia bit wskaźnika pierścieniowego (bit #6) kiedy modem zaznacza linię wskaźnika pierścieniowego. Będziemy rzadko używali tego sygnału, chyba ,że piszemy nowoczesne oprogramowanie sterujące, które automatycznie odpowiada na telefon.

Bit wykrywania sygnału nośnego (DCD, bit # 7) jest innym specyficznym sygnałem modemu. Bit ten zawiera jeden podczas gdy modem wykrywa sygnał nośny na lini telefonicznej.

Bity od zera do trzy MSR są bitami „delta". Bity te zawierają jeden, jeśli ich odpowiadający sygnał stanu modemu zmienia się. Takie wystąpienie również generuje stan przerwania modemu. Odczyt MSR będzie czyścił te bity.

---

22.1.9 POMOCNICZE REJESTR WEJŚCIOWY

Pomocniczy rejestr wejściowy jest dostępny tylko na późniejszych modelach 8250 kompatybilnych urządzeń. Jest to rejestr tylko odczyt, który zwraca taką samą wartość jaka odczytuje rejestr danych. Różnica pomiędzy odczytem tego rejestru a odczytem rejestru danych jest taka, że odczyt pomocniczego rejestru wejściowego nie wpływa na bit dostępnej danej w LSR. Pozwala to nam przetestować nadchodzące wartości danej bez usuwania jej z rejestru wejściowego. Jest to użyteczne, na przykład kiedy tworzymy łańcuch podprogramów obsługi przerwań chipu szeregowego i chcemy obsłużyć pewne „gorące" wartości w jednym ISR'ze i przekazać wszystkie inne znaki do różnych szeregowych ISR'ów.

---

22.2 PROGRAMY WSPIERAJĄCE KOMUNIKACJĘ SZEREGOWĄ BIBLIOTEKI STANDARDOWEJ UCR

Chociaż oprogramowanie 8250 SCC nie wydaje się być rzeczywiście dużym problemem, niezmiennie jest trudnym ,niewdzięcznym (i nużącym) pisanie całego koniecznego oprogramowania uzyskując działający system komunikacji szeregowej. Jest to szczególnie prawdziwe kiedy używamy układów szeregowych I/O sterowanych przerwaniami. Na szczęście nie musimy pisać tego oprogramowania od podstaw, Biblioteka Standardowa UCR dostarcza 21 podprogramów, które trywializują zastosowanie portów szeregowych w PC. Jedyną wadą tych podprogramów jest to ,że zostały napisane specjalnie dla COM1:, chociaż nie jest zbyt dużo pracy z modyfikacją ich do współpracy z COM2: Poniższa tablica pokazuje dostępne podprogramy:

| Nazwa | Wejścia | Wyjścia | Opis |
|---|---|---|---|
| ComBaud | AX: bps (baud rate) = 110,150, 300, 600, 1200, 2400, 4800, 9600 lub 19200 | | Ustawia szybkość komunikacji portu szeregowego. ComBaud wspiera tylko określone szybkości. Jeśli ax zawiera jakąś inną wartość na wejściu, ComBaud ignoruje ją |
| ComStop | AX: 1 lub 2 | | Ustawia liczbę bitów stopu. Rejestr ax zawiera liczbę bitów |

| | | | |
|---|---|---|---|
| | | | stop (1 lub 2) |
| ComSize | AX: rozmiar słowa (5,6,7 lub 8) | | Ustawia liczbę bitów danej. Rejestr ax zawiera liczbę bitów do przesłania dla każdego bajtu na lini szeregowej. |
| ComParity | AX: selektor parzystości. Jeśli bit zero jest zerem, parzystość wyłączona, jeśli jeden, bity jeden i dwa to:<br>00-kontorla nieparzystości<br>01-kontroal parzystości<br>10-zablokowana parzystość przy 0<br>11-zablokowana parzystość przy 1 | | Ustawia parzystość dla komunikacji szeregowej |
| ComRead | | AL- Odczyt znaku z portu | Czeka dopóki znak jest dostępny z rejestru danych i zwraca ten znak. Używana dla odpytywania I/O na porcie szeregowym. Ni używane jeśli aktywujemy przerwania szeregowe. |
| ComWrite | AL – znak do zapisu | | Oczekuje dopóki rejestr przechowujący przekaźnika jest pusty, wtedy zapisuje znak w al. do rejestru wyjściowego. Używana przy odpytywaniu I/O w porcie szeregowym. Nie używamy przy aktywnych przerwaniach. |
| ComTstIn | | AL.=0 jeśli brak znaku<br>AL.=1 jeśli znak dostępny | Testuje by sprawdzić czy znak jest dostępny w porcie szeregowym. Używamy tylko do odpytywania I/O , nie używamy z aktywnymi przerwaniami |
| ComTstOut | | AL.=0 jeśli przekaźnik zajęty, AL.=1 jeśli nie zajęty | Testuje by sprawdzić czy można zapisać znak do rejestru wyjściowego. Używany tylko z odpytywaniem I/O |
| ComGetLSR | | AL= Bieżąca wartość LSR | Zwraca bieżącą wartość LSR w rejestrze al. |
| ComGetMSR | | AL.= bieżąca wartość MSR | Zwraca bieżącą wartość MSR w rejestrze al. |
| ComGetMCR | | AL= bieżąca wartość MCR | Zwraca bieżącą wartość MCR w rejestrze al. |
| ComSetMCR | AL.= nowa wartość MCR | | Przechowuje wartość al. w rejestrze MCR |
| ComGetLCR | | AL= bieżąca wartość LCR | Zwraca bieżącą wartość LCR w rejestrze al. |
| ComSetLCR | AL.= nowa wartość LCR | | Przechowuje wartość al. w rejestrze LCR |
| ComGetIIR | | AL.= bieżąca wartość IIR | Zwraca bieżącą wartość IIR w rejestrze al. |
| ComGetIER | | AL.= bieżąca wartość IER | Zwraca bieżącą wartość IER w rejestrze al. |
| ComSetIER | AL. = nowa wartość IER | | Przechowuje wartość al. w rejestrze IER |
| ComInitIntr | | | Inicjalizuje system wspierając układy szeregowe I/O sterowane przerwaniami. |
| ComDisIntr | | | Resetuje system z powrotem do szeregowego odpytywania I/O |
| ComIn | | | Czyta znak portu szeregowego kiedy działa z układami I/O sterowanymi przerwaniami |
| ComOut | | | Zapisuje znak do portu szeregowego używając układów I/O sterowanych przerwaniami |

Tablica 85: Wsparcie portu szeregowego przez Bibliotekę Standardową

Cecha podprogramów Biblioteki Standardowej , sterowanie układami I/O poprzez przerwania zasługuje na późniejsze wyjaśnienia. Kiedy wywołujemy podprogram ComInitIntr, aktualizujemy wektor przerwań COM1: (int 0Ch), włączamy IRQ 4 w PIC 8259A i włączamy odczyt i zapis przerwań w 8250 SCC. Jedyna rzecz jakiej to wywołanie nie robi to to, że powinniśmy zaktualizować wektory wyjątków break i błędu krytycznego (int 23h i int 24h) obsługujące przerwanie każdego programu jaki nadchodzi. Kiedy nasz program kończy się, albo normalnie , albo przez jeden z powyższych wyjątków , musi wywołać ComDisIntr do zablokowania przerwań .W przeciwnym wypadku, następnym razem znak przychodzący do portu szeregowego maszyny może spowodować krach ponieważ będzie próbował skoczyć do podprogramu obsługi przerwania, którego może tam nie być.

Podprogramy ComIn i ComOut obsługują szeregowe układy I/O sterowane przerwaniami. Biblioteka Standardowa dostarcza stosownych buforów wejściowego i wyjściowego (podobnych do bufora roboczego klawiatury), abyśmy nie musieli martwić się o zgubione znaki chyba ,że program jest rzeczywiście, rzeczywiście wolny lub rzadko odczytuje dane z portu szeregowego.

Pomiędzy funkcjami ComInitIntr a ComDisIntr nie powinniśmy wywoływać żadnego innego podprogramu szeregowego z wyjątkiem ComIn i ComOut. Inne podprogramy są zaplanowane do odpytywania I/O i inicjalizacji. Oczywiście powinniśmy wykonać konieczną inicjalizację przed włączeniem przerwań i nie wolno odpytywać I/O podczas gdy działają przerwania. Zauważmy, że nie ma odpowiedników dla ComTstIn i ComTstOut w czasie działania w trybie przerwania. Podprogramy te są łatwe do napisania, instrukcja pojawi się w następnej sekcji.

---

## 22.3 OPROGRAMOWANIE 8250 (PRZYKŁADY Z BIBLIOTEKI STANDARDOWEJ)

Podprogramy komunikacji szeregowej Biblioteki Standardowej UCR dostarczają doskonałego przykładu jak oprogramować bezpośrednio 8250 SCC, ponieważ używają prawie wszystkich cech tego chipu z PC. Dlatego też, sekcja ta wylistuje każdy z tych podprogramów i opisze dokładnie co dany podprogram robi. Poprzez studiowanie tego kodu możemy nauczyć się o wielu szczegółach związanych z SCC i odkryć jak rozszerzyć lub zmodyfikować podprogramy Biblioteki Standardowej.

```
;Użyteczne równania:

BIOSvars        =       40h                     ;adres segmentu BIOS'a
Com1Adrs        =       0                       ;offset adresu COM1: zmiennych BIOS
Com2Adrs        =       2                       ;offset adresu COM2: zmiennych BIOS

BufSize         =       256                     ;# bajtów w buforze

; Przyrównania portu szeregowego. Jeśli chcemy wesprzeć COM2: zamiast COM1:, po prostu zmieniamy
; przyrównanie na 2F8h, 2F9h....

ComPort         =       3F8h
ComIER          =       3F9h
ComIIR          =       3FAh
ComLCR          =       3FBh
ComMCR          =       3FCh
ComLSR          =       3FDh
ComMSR          =       3Feh

; Zmienne itd., Kod ten zakłada, że DS=CS. To znaczy, wszystkie zmienne są w segmencie kodu.
;
;Wskaźnik do wektora przerwań dla int 0Ch w tablicy wektorów przerwań. Notka: zmieniamy te wartości na
; 0Bh*4 i 0Bh*4+2 jeśli chcemy wesprzeć port COM2:



int0Cofs        equ     es:[0Ch*4]
int0Dseg        equ     es:[0Ch*4+2]
```

```
OldInt0C        dword   ?

;Bufor wejściowy dla nadchodzących znaków (tylko działanie przerwań). Zobacz rozdział o strukturach danych
i
; opis kolejki cyklicznej po szczegóły jak działa bufor. Działa w sposób inny niż bufor roboczy klawiatury.

InHead          word    InpBuf
InTail          word    InpBuf
InpBuf          byte    BufSize  dup  (?)
InpBufEnd       equ     this byte

;Bufor wyjściowy dla znaków oczekujących na przesłanie

OutHead         word    OutBuf
OutTail         word    OutBuf
OutBuf          byte    BufSize  dup  (?)
OutBufEnd       equ     this byte

; Zmienna 18259a przechowuje kopię PIC IER'a więc możemy przywrócić ją po usunięciu naszego
; podprogramu obsługi przerwań z pamięci.

18259a          byte    0                               ;rejestr zezwalający na przerwania 8259a

;Zmienna TestBuffer mówi nam czy mamy bufor pełen znaków lub czy możemy przechować kolejny znak
; bezpośrednio w rejestrze wyjściowym 8250

TestBuffer      db      0
```

Pierwszy zbiór podprogramów dostarczony przez Bibliotekę Standardową pozwala nam inicjalizować 8250 SCC. Podprogramy te dostarczają interfejsu „przyjaznego programiście" rejestrów dzielnika szybkości transmisji i starowania łączem .Pozwalają nam ustawić szybkość transmisji, rozmiar danej liczbę bitów stopu i opcję parzystości w SCC

Podprogram ComBaud ustawia szybkość transferu 8250 (w bitach na sekundę). Dostarcza przyjemnego „interfejsu programisty" 8250 SCC. Zamiast obliczania dzielnika szybkości transmisji samemu, możemy po prostu załadować ax wartością bps jaką chcemy i po prostu wywołujemy ten podprogram. Oczywiście jeden problem jest taki, że musimy wybrać wartość bps, jaką wspiera ten podprogram lub zignoruje żądanie zmiany szybkości transmisji. Na szczęście podprogram ten wspiera wszystkie popularne szybkości bps; jeśli potrzebujemy jakiejś innej, łatwo jest zmodyfikować ten kod na pozwalający na inne szybkości.

Kod ten skalda się z dwóch części. Pierwsza część porównuje wartość w ax ze zbiorem poprawnych wartości bps. Jeśli dopasuje, ładuje ax odpowiednią 16 bitową stałą dzielnika. Druga część tego kodu przełącza rejestry dzielnika szybkości transmisji i przechowuje wartość ax w tych rejestrach. W końcu przełącza pierwsze dwa rejestry I/O 8250 z powrotem na rejestr danych i zezwolenia na przerwanie.

Notka: Ten podprogram wywołuje kilka podprogramów, zwłaszcza ComSetLCR i ComGetLCR, które zdefiniujemy trochę później. Podprogramy te wykonują oczywiste funkcje, odczyt i zapis rejestru LCR

```
ComBaud         proc
                push    ax
                push    dx
                cmp     ax, 9600
                ja      Set19200
                je      Set9600
                cmp     ax, 2400
                ja      Set4800
                je      Set2400
                cmp     ax, 600
                ja      Set1200
                je      Set600
                cmp     ax, 150
```

```
                ja      Set300
                je      Set150
                mov     ax, 1047                ;domyślnie 110 bps
                jmp     SetPort

Set150:         mov     ax, 768                 ;wartość dzielnika dla 150 bps
                jmp     SetPort
Set300:         mov     ax, 384                 ;wartość dzielnika dla 300 bps
                jmp     SetPort
Set600:         mov     ax, 192                 ;wartość dzielnika dla 600 bps
                jmp     SetPort
Set1200:        mov     ax, 96                  ;wartość dzielnika dla 1200 bps
                jmp     SetPort
Set2400         mov     ax, 48                  :wartość dzielnika dla 2400 bps
                jmp     SetPort
Set4800:        mov     ax, 24                  ;wartość dzielnika dla 4800 bps
                jmp     SetPort
Set9600:        mov     ax, 12                  ;wartość dzielnika dla 9600 bps
                jmp     SetPort
Set19200        mov     ax, 6                   ;wartość dzielnika dla 19,2 kbps
SetPort:        mov     dx, ax                  ;zachowanie wartości szybkości
                call    GetLCRCom               ;pobranie wartości LCR
                push    ax                      ;zachowanie starej wartości bitu dzielnika
                ot      al., 80h                ;ustawienie bitu wyboru dzielnika
                call    SetLCRCom               ;zapis wartości LCR
                mov     ax, dx                  ;Pobranie wartości dzielnika szybkości transmisji
                mov     dx, ComPort             ;wskazuje mniej znaczący bajt rejestru dzielnika
                out     dx, al.                 ;wyprowadzenie mniej znaczącego bajtu dzielnika
                inc     dx                      ;wskazuje bardziej znaczący bajt
                mov     al., ah                 ;włożenie bardziej znaczącego bajtu do AL.
                out     dx, al.                 ;wyprowadzenie bardziej znaczącego bajtu
                pop     ax                      ;przywrócenie starej wartości LCR
                call    SetLCRCom1              ;przywrócenie wartości bitu dzielnika
                pop     dx
                pop     ax
                ret
ComBaud         endp
```

Podprogram ComStop oprogramowuje LCR dostarczając określoną liczbę bitów stopu. Na wyjściu, ax powinien zawierać albo jeden albo dwa (liczbą żadnych bitów stopu) Kod ten konwertuje to do zera lub jeden i zapisuje wynikowy mniej znaczący bit do pola bitu stopu LCR. Zauważmy ,że kod ten ignoruje inne bity w rejestrze ax. Odczytuje LCR, maskuje pole bitu stopu a potem odwraca wartość kodu wywołującego określonego w tym polu. Odnotujmy zastosowanie instrukcji shl ax, 2; wymaga to procesora 80286 lub późniejszego.

```
ComStop         proc
                push    ax
                push    dx
                dec     ax                      ;konwersja 1lub 2 na 0 lub 1
                and     al., 1                  ; usuwamy inne bity
                shl     ax, 2                   ;pozycja w bicie #2
                mov     ah, al.                 ;zachowujemy naszą wartość wyjściową
                call    ComGetLCR               ;odczyt wartości LCR
                and     al., 11111011b          ;maskowanie bitu stopu
                or      al., ah                 ;podział nowych # bitów stopu
                call    ComSetLCR               ;zapis wyniku ponownie do LCR
                pop     dx
                pop     ax
                ret
```

```
ComStop         endp
```

ComSize ustawia rozmiar słowa dla transmisji danych. Zazwyczaj kod ten dostarcza „przyjacielskiego programiście" interfejsu do 8250 SCC. Na wejściu określamy liczbę bitów (5,6,7 lub 8) w rejestrze ax, nie musimy martwić się o właściwy wzorzec bitów dla rejestru LCR 8250. Podprogram ten oblicza właściwy wzorzec bitów dla nas. Jeśli wartość w rejestrze ax nie jest właściwa, kod ten domyślnie ustawi rozmiar słowa na osiem bitów.

```
ComSize         proc
                push    ax
                push    dx
                sub     al., 5                  ;odwzorowanie 5..8 → 00b, 01b, 10b, 11b
                cmp     al., 3
                jbe     Okay
                mov     al., 3                  ;domyślnie osiem bitów
Okay:           mov     ah, al.                 ;zachowanie nowego rozmiaru bitów
                call    ComGetLCR               ;odczyt bieżącej wartości LCR
                and     al., 11111100b          ;maskowanie starego rozmiaru słowa
                or      al., ah                 ;dzielenie nowego rozmiaru słowa
                call    ComSetLCR               ;zapis nowej wartości LCR
                pop     dx
                pop     ax
                ret
ComSize         endp
```

Podprogram ComParity inicjalizuje opcje parzystości w 8250. Niestety jest mniejsza możliwość „przyjaznego programiście" interfejsu dl tego podprogramu, więc ten kod wymaga aby przekazać mu jedna z następujących wartości w rejestrze ax:

| Wartość w AX | Opis |
|---|---|
| 0 | Parzystość wyłączona |
| 1 | Włączona kontrola nieparzystości |
| 3 | Włączona kontrola parzystości |
| 5 | Włączony bit zablokowanej parzystości wartość jeden |
| 7 | Włączony bit zablokowanej parzystości wartość zero |

Tablica 86: Parametry wejœciowe ComParity

```
ComParity       proc
                push    ax
                push    dx

                shl     al ,3                   ;przesunięcie na końcową pozycję w LCR
                and     al., 00111000b          ;maskowanie innej danej
                mov     ah, al.                 ;zachowanie na później
                call    ComGetLCR               ;pobranie bieżącej wartości LCR
                and     al., 11000111b          ;maska istniejących bitów parzystości
                or      al., ah                 ;podział nowych bitów
                call    ComSetLCR               ;zapis wyniku do LCR
                pop     dx
                pop     ax
                ret
ComParity       endp
```

Kolejny zbiór podprogramów komunikacji szeregowej dostarcza wsparcia dla odpytywania I/O. Podprogramy te pozwalają nam odczytać znaki z portu szeregowego, zapisać znaki do portu szeregowego i sprawdzić czy jest dostępna dana w porcie wejściowym lub zobaczyć czy można zapisać dane do portu

wyjściowego. Pod żadnym pozorem nie powinniśmy używać tych podprogramów kiedy mamy aktywny system przerwań szeregowych. Robiąc to możemy zamieszać w systemie i stworzyć niepoprawne dane lub zagubić je.

Podprogram ComRead jest porównywalny z getc – oczekuje dopóki dana jest dostępna w porcie szeregowym, odczytuje tą dana i zwraca ją w rejestrze al. Ten program zaczyna się poprzez upewnienie się, ze możemy uzyskać dostęp do rejestru odbioru danych (poprzez wyczyszczenie bitu zatrzasku dzielnika szybkości transmisji w LCR)

```
ComRead         proc
                push    dx
                call    GetLCRCom
                 push   ax                      ;zachowanie bitu zatrzasku dzielnika
                and     al., 7fh                ;wybór normalnego portu
                call    SetLCRCom               ;zapis LCR wyłączając rejestr dzielnika
WaitForChar:    call    GetLSRCom               ;pobranie bitu dostępności danej z LSR
                test    al., 1                  ;dana dostępna?
                jz      WaitForChar             ;pętla dopóki dana dostępna
                mov     dx, ComPort             ;odczyt danej z portu wejściowego
                in      al., dx
                mov     dl, al.                 ;zachowanie znaku
                pop     ax                      ;przywrócenie bitu dostępu dzielnika
                call    SetLCRCom               ;zapis z powrotem do LCR
                mov     al., dl                 ;przywrócenie wyprowadzonego znaku
                pop     dx
                ret
ComRead         endp
```

Podprogram ComWrite wyprowadza znak z al. do portu szeregowego. Najpierw czeka dopóki rejestr przechowywani przekaźnika jest pusty, potem zapisuje daną wyjściową do rejestru wyjściowego

```
ComWrite        proc
                push    dx
                push    ax
                mov     dl, al.                 ;zachowanie znaku do wyprowadzenia
                call    GetLCRCom               ;przełączenie na rejestr wyjściowy
                push    ax                      ;zachowanie bitu zatrzasku dzielnika
                and     al., 7fh                ;wybranie zwykłego portu wejścia / wyjścia
                call    SetLCRCom               ;zamiast rejestru dzielnika
WaitForXmtr:    call    GetLSRCom               ;odczyt LSR z bitu pustego transmitera
                test    al., 00100000b          ;bufor przekaźnika pusty?
                jz      WaitForXmtr             ;pętla dopóki pusty
                mov     al., dl                 ;pobranie znaku do wyprowadzenia
                mov     dx, ComPort             ;przechowanie w porcie wyjściowym
                out     dx, al                  ;przywrócenie bitu dzielnika
                pop     ax
                call    SetLCRCom
                pop     ax
                pop     dx
                ret
ComWrite        endp
```

Podprogramy ComTstIn I ComTstOut pozwalają nam sprawdzić czy znak jest dostępny w porcie wejściowym (ComTstIn) lub czy można wysłać znak do portu wyjściowego (ComTstOut) . ComTstIn zwraca zero lub jeden w al., odpowiednio, jeśli dana nie jest dostępna lub jest dostępna. ComTstOut zwraca zero lub jeden w al., odpowiednio, jeśli rejestr przekaźnika jest pełny lub pusty

```
ComTstIn        proc
                call    GetComLSR
                and     ax, 1                   ;przetrzymujemy tylko bit dostępności
                ret
```

```
ComTstIn        endp
ComTstOut       proc
                push    dx
                call    ComGetLSR               ;pobranie stanu łącza
                test    al., 00100000b          ;maskujemy bit pustego przekaźnika
                mov     al., 0                  ;zakładamy, że nie pusty
                jz      tocl                    ;skok jeśli nie pusty
                inc     ax                      ;ustawiamy jeden jeśli pusty
tocl:           ret
ComTstOut       endp
```

Kolejny zbiór podprogramów Biblioteki Standardowej dostarcza ładowania i przechowywania różnych rejestrów w 8250 SCC. Chociaż są to trywialne podprogramy, pozwalają programiście uzyskać dostęp do tego rejestru poprzez nazwę bez poznania jego adresu. Co więcej podprogramy te zachowują wszystkie wartości w rejestrze dx, zachowując jakiś kod w programie wywoływanym jeśli rejestr dx jest już używany.

Poniższe podprogramy pozwalają nam odczytać („Get") wartość w rejestrach LSR, MSR, MCR, IIR i IER, zwracając wartość w rejestrze al. Pozwalają nam zapisać („Set") wartość z al do każdego z rejestrów LCR, MCR i IER. Ponieważ te podprogramy są tak proste nie ma potrzeby ich omawiania ich pojedynczo. Zauważmy, że powinniśmy unikać wywoływania tych podprogramów na zewnątrz SCC ISR podczas trybu przerwań, ponieważ robiąc to możemy wpłynąć na system przerwań 8250 SCC.

```
ComGetLSR       proc                            ;zwraca wartość LSR w rejestrze AL.
                push    dx
                mov     dx, ComLSR              ;wybór rejestru LSR
                in      al., dx                 ;odczyt i zwrot wartości LSR
                pop     dx
                ret
ComGetLSR       endp

ComGetMSR       proc                            ;zwraca wartość MSR w rejestrze AL.
                push    dx
                mov     dx, ComMSR              ;wybór rejestru MSR
                in      al, dx                  ;odczyt i zwrot wartości MSR
                pop     dx
                ret
ComGetMSR       endp

ComSetMCR       proc                            ;zachowanie wartości AL. w rejestrze MCR
                push    dx
                mov     dx, ComMCR              ;wskazuje rejestr MCR
                out     dx, al                  ;wyprowadzenie wartości z AL. do MCR
                pop     dx
                ret
ComSetMCR       endp

ComGetMCR       proc                            ;przechowanie wartości AL. w rejestrze MCR
                push    dx
                mov     dx, ComMCR              ;wybór rejestru MCR
                in      al, dx                  ;odczyt wartości z rejestru MCR do AL
                pop     dx
                ret
ComGetMCR       endp

ComGetLCR       proc                            ; zwraca wartość LCR w rejestrze AL.
                push    dx
                mov     dx, ComLCR              ;wskazuje rejestr LCR
                in      al., dx                 ;odczyt i zwrot wartości LCR
                pop     dx
```

```
                ret
ComGetLCR       endp
ComSetLCR       proc                            ;zapis nowej wartości do LCR
                push    dx
                mov     dx, ComLCR              ;wskazuje rejestr LCR
                out     dx, al.                 ;zapis wartości w AL do LCR
                pop     dx
                ret
ComSetLCR       endp

ComGetIIR       proc                            ;zwraca wartość w IIR
                push    dx
                mov     dx, ComIIR              ;wybór rejestru IIR
                in      al, dx                  ;odczyt  wartości IIR do AL. i powrót
                pop     dx
                ret
ComGetIIR       endp

ComGetIER       proc                            ;zwraca wartość IER w AL.
                push    dx
                call    ComGetLCR               ;musimy wybrać rejestr IER poprzez zachowanie
                push    ax                      ;wartości LCR a potem wyczyszczenie bitu
                and     al., 7fh                ;zatrzasku dzielnika szybkości transmisji
                call    ComSetLCR
                mov     dx, ComIER              ;adres IER
                in      al, dx                  ;odczyt bieżącej wartości IER
                mov     dl, al.                 ;zachowanie teraz
                pop     ax                      ;odzyskanie starej wartości LCR
                call    ComSetLCR               ;przywrócenie zatrzasku dzielnika
                mov     al., dl                 ;przywrócenie wartości IER
                pop     dx
                ret
ComGetIER       endp

ComSetIER       proc                            ;zapis wartości AL. do IER
                push    dx
                push    ax                      ;zachowanie wartości AX
                mov     ah, al.                 ; zachowanie wartości IER do wyprowadzenia
                call    ComGetLCR               ;pobranie i zachowanie bitu dzielnika
                push    ax
                and     al., 7fh                ;czyszczenie bitu dzielnika
                call    ComSetLCR
                mov     al., ah                 ;odzyskanie nowej wartości IER
                mov     dx. ComIER              ;wybór rejestru IER
                out     dx, al.                 ; wartość wyjściowa IER
                pop     ax                      ;przywrócenie bitu dzielnika
                call    ComSetLCR
                pop     ax
                pop     dx
                ret
ComSetIER       endp
```

Ostatni zbiór szeregowych podprogramów pojawiających się w Bibliotece Standardowej dostarcza wsparcia dla układów I/O sterowanych przerwaniami. Jest pięć podprogramów w tej sekcji: ComInitIntr, ComDisIntr, ComIntISR, ComIn i ComOut. ComInitIntr inicjalizuje system przerwań portu szeregowego . Zachowuje stary wektor [przerwania 0Ch, inicjalizuje wektor wskazywanym przez ComIntISR podprogramem obsługi przerwania i stosownie inicjalizuje PIS 8259A i 8250 SCC dla operacji opartych na przerwaniu. ComDisIntr niszczy wszystko co ustawi ComDisIntr; musimy wywołać ten podprogram wyłączając przerwania zanim nasz pogram się skończy. ComOut i ComIn przekazują dane do i z bufora opisanego w sekcji zmiennych;

Podprogram ComIntISR jest odpowiedzialny za usunięcie danych z kolejki transmisji i wysłanie przez linię szeregową , jak również buforowanie nadchodzących danych z lini szeregowej.

Podprogram ComInitIntr inicjalizuje 8250 SCC i PIC 8259A przerwaniami szeregowym I/O. Również inicjalizuje wektor 0Ch wskazujący podprogram ComIntISR. Jedynej rzeczy jakiej ten kod nie robi to dostarczenie obsługi wyjątków break i błędu krytycznego. Pamiętajmy, jeśli użytkownik naciśnie ctrl-C (lub ctrl-Break) lub wybierze przerwanie przy błędzie I/O, domyślny program obsługi wyjątków po prostu wróci do DOS'a bez przywracania wektora 0Ch. Ważne jest aby nasz program dostarczył obsługi wyjątku, który wywoła ComDisIntr zanim zezwoli systemowi zwrócić sterowanie do DOS'a . W przeciwnym razie system może mieć krach, kiedy DOS załaduje kolejny program do pamięci.

```
ComInitIntr     proc
                pushf                   ;zachowanie flagi wyłączenia przerwań
                push    es
                push    ax
                push    dx
; Wyłączamy przerwania

                cli

;Zachowujemy stary wektor przerwań. Oczywiście musimy zmienić poniższy kod zachowując i ustawiając
;wektor 0Bh. Jeśli chcemy uzyskać dostęp do COM2: zamiast do portu COM1;

                xor     ax, ax          ;wskazuje wektor przerwań
                mov     es, ax
                mov     ax, Int0Cofs
                mov     word ptr OldInd0C, ax
                mov     ax, Int0Seg
                mv      word  ptr OldInt0C+2, ax

;wskazujemy wektor 0Ch naszego programu obsługi przerwań

                mov     ax, cs
                mov     Int0Cseg, ax
                mov     ax, offset ComIntISR
                mov     Int0Cofs, ax

;zerujemy oczekujące przerwania:

                call    ComGetLSR       ;zerujemy stan łącza odbiorczego
                call    ComGetMSR       ;zerujemy przerwania CTS/DSR/RI
                call    ComGetIIR       ;zerujemy przerwanie pustego transmitera
                mov     dx, ComPort
                in      al, dx          ;zerujemy przerwanie dostępnej danej

; Zerujemy bit dostępu dzielnika/. PODCZAS OPEROWANIA W TRYBIE PRZERWAŃ, BIT TEN MUSI
; BYĆ ZEREM. Jeśli z jakiegoś dziwnego powodu musimy zmienić szybkość transmisji w środku transmisji
; lub kiedy przerwania są włączone) zerujemy flagę przerwań, zerujemy bit zatrzasku dzielnika i w końcu
; przywracamy przerwania.

                call    ComGetLCR       ;Pobranie LCR
                and     al, 7fh         ;zerowanie bitu zatrzasku dzielnika
                call    ComSetLCR       ;zapisanie nowej wartości LCR

;Włączamy przerwania odbiornika i nadajnika. Zauważmy, że kod ten ignoruje błędy i przerwania
; zmiany stanu modemu

                mov     al., 3          ;włączenie przerwań odb./nad.
                call    SetIERCom
```

```
;Musimy ustawić linie OUT2 do pracy z przerwaniami. Również ustawiamy aktywne DTR i RTS

                mov     al., 00001011b
                call    ComSetMCR

;Aktywacja bitu COM1 (int 0Ch) w chipie 8259A. Notka: musimy zmienić poniższy kod dla
; wyczyszczenia bitu trzy (zamiast cztery) dla zastosowania tego kodu z portem COM2:.

                in      al.,21h         ;pobranie wartości zezwolenia na przerwanie 8259A
                mov     18259a, al.     ;zapisanie bitów  włączenia przerwania
                and     al., 0efh       ;bit 4 = IRQ 4 = INT 0Ch
                out     21h, al         ;włączenie przerwań

                pop     dx
                pop     ax
                pop     es
                popf                            ;przywrócenie wyłączonej flagi przerwań
                ret
ComInitIntr     endp
```

Podprogram ComDisIntr blokuje przerwania szeregowe. Przywraca oryginalną wartość rejestru zezwolenia na przerwania 8259A, przywraca wektor przerwań int 0Ch i maskuje przerwania w 8250 SCC. Zauważmy, że kod ten zakłada, że nie musimy zmieniać bitów zezwolenia na przerwania w PIC 8259 ponieważ wywołujemy ComInitIntr. Przywraca rejestr zezwolenia na przerwanie 8259A wartością z rejestru zezwolenia na przerwanie 8259A kiedy pierwotnie wywołujemy ComInitIntr.

Byłoby kompletną katastrofą wywołanie tego programu bez wcześniejszego wywołania ComInitIntr. Robiąc to, zaktualizowalibyśmy wektor 0Ch śmieciami i , podobnie, przywrócilibyśmy rejestr zezwolenia na przerwania 8259A ze śmieciami. Upewnijmy się, ze wywołaliśmy ComInitIntr przed wywołaniem tego podprogramu. Generalnie powinniśmy wywoływać ComInitIntr raz, na początku naszego programu, i wywołać ComDisIntr raz, albo na końcu programu albo wewnątrz podprogramu wyjątku break lub błędu krytycznego.

```
ComDisIntr      proc
                pushf
                push    es
                push    dx
                push    ax

                cli                             ;nie zezwalamy na przerwania
                xor     ax, ax
                mov     es, ax                  ;ES wskazuje tablicę wektorów przerwań
; Najpierw wyłączamy źródła przerwań w chipie 8250:

                call    ComGetMCR               ;pobranie bitu OUT2
                and     al., 3                  ;zamaskowanie bitu OUT2
                call    ComSetMCR               ;zapis wyniku do MCR

; Teraz przywracamy bit IRQ 4 w PIC 8259A. Zauważmy, że musimy zmodyfikować ten kod , jeśli chcemy
; wesprzeć COM2: zamiast COM1:

                in      al., 21h                ;pobranie bieżącej wartości IER 8259A
                and     al., 0efh               ;zerowanie bitu IRQ 4 -zmiana na COM2:
                mov     ah, 18259a              ;pobranie naszej zapisanej wartości
                and     ah, 1000b               ;zamaskowanie  bitu COM1 (IRQ 4)
                or      al., ah                 ;odłożenie bitu z powrotem
                out     21h, al.

; przywracanie wektora przerwań:

                mov     ax, word  ptr OldInt0C
```

```
                mov     Int0Cofs, ax
                mov     ax, word  ptr OldInt0C+2
                mov     Int0Cseg, ax

                pop     ax
                pop     dx
                pop     es
                popf
                ret
ComDisIntr      endp
```

Poniższy kod implementuje podprogram obsługi przerwania dla 8250 SCC. Kiedy wystąpi przerwanie, kod ten odczyta IIR 8250 aby określić źródło przerwania. Podprogramy Biblioteki Standardowej dostarczają tylko bezpośredniego starcia dla przerwania dostępnej danej i przerwania pustego rejestru przechowywania nadajnika. Jeśli kod ten wykryje błąd lub zmianę stanu przerwania, zeruje stan przerwania ale nie podejmuje innej akcji. Jeśli wykryje przerwanie odbiornika lub nadajnika, przekazuje sterowanie do właściwego programu obsługi

Program obsługi przerwania odbiornika jest bardzo łatwy do implementacji. Wszystko co ten kod musi robić to odczytać znak z rejestru odbiorczego i dodać ten znak do bufora wejściowego. Jedyny ból, to to, że kod ten musi ignorować każdy nachodzący znak jeśli bufor wejściowy jest pełny. Aplikacja może uzyskać dostęp do tej danej przy zastosowaniu podprogramu ComIn, który usuwa dane z bufora wejściowego.

Pogram obsługi nadajnika jest nieco bardziej złożony. 8250 SCC przerywa 80x86 kiedy możliwe jest zaakceptowanie więcej danych do transmisji. Jednakże, fakt, że 8250 jest gotowy na więcej danych nie gwarantuje, że te dane są gotowe do przesłania. Aplikacja tworzy dane swoją własną szybkością, nie koniecznie z szybkością jaką chce 8250 SCC. Dlatego też, jest całkiem możliwe przy 8250 powiedzenie „daj mi więcej danych" ale aplikacja nie tworzy żadnej. Oczywiście nie powinniśmy przesyłać niczego w tym momencie. Zamiast tego, musimy czekać aby aplikacja stworzyła więcej danych przed ponowieniem transmisji.

Niestety , to komplikuje nieco sterowanie kodem przesyłu. Przy odbiorniku, przerwanie zawsze wskazywało, ze ISR może przesunąć dane z 8250 do bufora. Aplikacja może usunąć tą dana a proces jest zawsze taki sam.: oczekiwanie na nie pusty bufor odbiorczy a potem usunięcie pierwszej pozycji z bufora. Niestety nie możemy po prostu odwrócić działania kiedy przesyłamy dane. To znaczy nie możemy po prostu przechować danej w buforze przesyłu i pozostawić w ISR'ze aż do usunięcia tej danej. Problem jest taki, że 8250 przerywa tylko raz system, kiedy rejestr przechowujący nadajnika jest pusty. Jeśli w tym momencie nie ma danych do przesłania , ISR musi wrócić bez zapisywania czegokolwiek do rejestru przesyłu. Ponieważ nie ma danej w burze przesyłu, nie będzie generowanych dodatkowych przerwań, nawet jeśli będzie dodana dana do bufora przesyłu. Dlatego też, ISR i podprogramy odpowiedzialne za dodawanie danych do bufora wyjściowego (ComOut) musza skoordynować swoją aktywność. Jeśli bufor jest pusty a nadajnik aktualnie nie przesyła niczego, podprogram ComOut musi zapisać swoje dane bezpośrednio do 8250. Jeśli 8250 aktualnie przesyła dane, ComOut musi dołączyć swoje dane do na koniec bufora wyjściowego. ComIntISR i ComOut używają flag. TestBuffer określa czy ComOut powinien zapisać bezpośrednio do portu szeregowego lub dołączyć dane do bufora wyjściowego. Spojrzyjmy na ten kod i kod dla ComOut

```
ComIntISR       proc    far
                push    ax
                push    bx
                push    dx
TryAnother:     mov     dx, ComIIR
                in      al, dx          ;pobranie wartości ID przerwania
                test    al., 1          ;inne przerwania ?
                jnz     IntRtn          ;wyjdź ,jeśli inne przerwania nie oczekują
                cmp     al., 100b       ;ponieważ są tylko przerwania nad./odb.
                jnz     ReadCom1        ; aktywne, sprawdzamy dla przerwań odb.
                cmp     al., 10b        ;sprawdzenie dla przerwania pustego nad.
                jnz     WriteCom1

; Fałszywe przerwanie? Nie powinniśmy nigdy dopuścić do tego kodu ponieważ nie włączyliśmy przerwań
; błędu lub zmiany stanu. Jednak jest możliwe ,że kod aplikacji może wejść i wkręcić IER w 8250. dlatego też,
; musimy dostarczyć domyślnego programu obsługi przerwań dla tych warunków. Poniższy kod odczytuje
; wszystkie właściwe rejestry dla wyzerowania każdego oczekującego przerwania.
```

```
                call    ComGetLSR               ;wyzerowanie stanu łącza odbiorczego
                call    ComGetMSR               ;wyzerowanie stanu modemu
                jmp     TryAnother              ;sprawdzenie niższego priorytetu przerwania

; Kiedy nie ma więcej przerwań oczekujących w 8250, wracamy z tego ISR'a

IntRtn:         mov     al., 20h                ;potwierdzenie przerwania sterownika przerwań
                out     20h, al                 ;8259A
                pop     dx
                pop     bx
                pop     ax
                iret
;  Tu mamy obsługę nadchodzących danych:
;(Ostrzeżenie: To jest region krytyczny. Przerwania MUSZĄ BYĆ WYŁĄCZONE podczas wykonywania tego
; kodu. Domyślnie, przerwania są wyłączane w ISR'ze. NIE WŁĄCZAMY ICH  jeśli modyfikujemy ten kod)

ReadCom1:       mov     dx, ComPort             ;wskazuje rejestr wejściowy danej
                in      al., dx                 ;pobranie znaku wejściowego

                mov     bx, InHead              ;wprowadzenie znaku do szeregowego bufora
                mov     [bx], al.               wejściowego

                inc     bx                      ;zwiększenie wskaźnika bufora
                cmp     bx, offset InpBufEnd
                jb      NoInpWrap
                mov     bx, offset  InpBuf
NoInpWrap:      cmp     bx, InTail              ;jeśli bufor jest pełny, ignorujemy ten znak
                je      TryAnother              ;wejściowy
                mov     InHead, bx
                jmp     TryAnother              ;idziemy do programu obsługi innych przerwań
                                                ;8250

; Program obsługi danych wychodzących (to również jest region krytyczny):

WriteCom1:      mov     bx, OutTail             ;zobaczmy czy bufor jest pusty
                cmp     bx, OutHead
                jne     OutputChar              ; jeśli nie, wyprowadzamy kolejny znak

;Jeśli głowa i ogon są równe, po prostu ustawiamy zmienną TestBuffer na zero  i wychodzimy. Jeśli nie są
równe
; wtedy dana jest w buforze i powinniśmy wyprowadzić kolejny znak.

                mov     TestBuffer, 0
                jmp     TryAnother                      ;obsługa innych przerwań oczekujących

;Wskaźniki bufora nie są równe, wyprowadzamy kolejny znak

OutputChar:     mov     al., [bx]                       ;pobrani kolejnego znaku z bufora
                mov     dx, ComPort                     ;wybór portu wyjściowego
                out     dx, al.                         ;wyprowadzania  znaku

;Okay, wskaźnik wyjściowy

                inc     bx
                cmp     bx, offset OutBufEnd
                jb      NoOutWrap
                mov     bx, offset OutBuf
NoOutWrap:      mov     OutTail, bx
```

```
                jmp     TryAnoter
ComIntISR       endp
```

Ostanie dwa podprogramy odczytują dane z szeregowego bufora wejściowego i zapisują dane do bufora wyjściowego. Podprogram ComIn, który obsługuje sprawy wejściowe, czeka dopóki bufor wejściowy nie będzie pusty. Potem usuwa pierwszy dostępny bajt z bufora wejściowego i zwraca tą wartość do programu wywołującego

```
ComIn           proc
                pushf                           ;zachowanie flagi przerwania
                push    bx
                sti                             ;upewnijmy się, że przerwania są włączone
TstInLoop:      mov     bx, InTail              ;czekamy dopóki jest przynajmniej jeden znak
                cmp     bx, InHead              ; w buforze wejściowym
                je      TstInLoop
                mov     al., [bx]               ;pobranie kolejnego znaku
                cli                             ;wyłączenie przerwań kiedy modyfikujemy
                inc     bx                      ;wskaźniki bufora
                cmp     bx, offset InpBufEnd
                jne     NoWrap2
                mov     bx, offset InpBuf
NoWrap2:        mov     InTail, bx
                pop     bx
                popf                            ;przywrócenie flagi przerwania
                ret
ComIn           endp
```

ComOut musi sprawdzić zmienną TestBuffer aby zobaczyć czy 8250 jest aktualnie zajęty. Jeśli nie (TestBuffer równa się zero) wtedy kod ten musi zapisać znak bezpośrednio do portu szeregowego i ustawić TestBuffer na jeden (ponieważ chip jest teraz zajęty). Jeśli TestBuffer zawiera wartość nie zerową, od ten po prostu dodaje znak z al. na koniec bufora wyjściowego.

```
ComOut          proc    far
                pushf
                cli                             ;żadnych przerwań
                cmp     TestBuffer, 0           ;zapisać bezpośrednio do chipa?
                jnz     BufferItUp              ;jeśli nie, włóż do bufora

;Poniższy kod zapisuje bieżący znak bezpośrednio do portu szeregowego ponieważ 8250 nie przekazuje niczego
; teraz i nigdy ponownie nie dostaniemy przerwania pustego rejestru przechowującego nadajnika (przynajmniej
; doki nie zapiszemy danej bezpośrednio do portu)

                push    dx
                mov     dx, ComPort             ;wybranie rejestru wyjściowego
                out     dx, al.                 ;zapis znaku do portu
                mov     TestBuffer, 1           ;bufor przechodzi do następnego znaku
                pop
                popf                            ;przywrócenie flagi przerwania
                ret

;jeśli 8250 jest zajęty,

BufferItUp:     push    bx
                mov     bx, OutHead             ;wskaźnik do kolejnej pozycji w buforze
                mov     [bx], al.               ;dodanie znaku do bufora

                inc     bx
                cmp     bx, offset  OutBufEnd
                jne     NoWrap3
                mov     bx, offset OutBuf
```

```
NoWrap3:    cmp     bx, OutTail             ;zobacz czy pełny bufor
            je      NoSetTail               ;nie dodawaj znaku jeśli bufor jest pełny
            mov     OutHead, bx             ;jeśli nie, aktualizuj wskaźnik bufora
NoSetTail:  pop     bx
            popf                            ;przywrócenie flagi przerwania
            ret
ComOut      endp
```

Zauważmy ,że Biblioteka Standardowa nie dostarcza żadnego podprogramu , który sprawdza czy dana jest dostępna w buforze wejściowym lub czy bufor wyjściowy jest pełny (porównywalne do podprogramów ComTstIn i ComTstOut). Jednakże jest bardzo łatwo taki podprogram napisać ; wszystko co musimy zrobić to porównać głowę i ogon wskaźników dwóch buforów .Bufory są puste jeśli głowa i ogon wskaźników są równe. Bufory są pełne jeśli głowa wskaźnika jest jeden bajt przed wskaźnikiem ogona (zapamiętajmy ,że wskaźnik zawija na koniec bufora, więc bufor jest również pełny jeśli wskaźnik głowy jest na ostatniej pozycji w buforze a wskaźnik ogona na pierwszej pozycji w buforze).

## 22.4 PODSUMOWANIE

Rozdział ten omawia komunikację szeregową RS-232 w PC. Podobnie jak przy porcie równoległym są trzy poziomy przy jakich możemy uzyskać dostęp do portu szeregowego: poprzez DOS, poprzez BIOS lub oprogramowanie bezpośrednie sprzętu. W odróżnieniu od wsparcia przez DOS i BIOS portu równoległego, wsparcie DOS'a dla szeregowego jest prawie bezwartościowe, a wsparcie BIOS jest dosyć słabe (tj. nie wspiera układów I/O sterowanych przerwaniami) . Dlatego też jest powszechną praktyką programistyczną na PC, sterowanie sprzętem bezpośrednio z aplikacji. Dlatego też , zapoznanie się z chipem komunikacji szeregowej 8250 jest ważne jeśli mamy zamiar korzystać z komunikacji szeregowej . Rozdział ten nie omawia komunikacji szeregowej pod DOS i BIOS, głownie z powodu ich ograniczonego wsparcia.

8250 wspiera 10 rejestrów I/O, które pozwalają nam sterować parametrami komunikacji, sprawdzić stan chipa, sterować potencjałem przerwań i, oczywiście ,wykonania szeregowych I/O, odwzorowania tych rejestrów 8250 jako ośmiu lokacji I/O w przestrzeni adresowej I/O PC.

PC wspiera do czterech urządzeń komunikacji szeregowej: COM1:, COM2:, COM3: i COM4:. Jednak większość oprogramowania działa tylko z portami COM1; i COM2:. Podobnie jak przy port równoległy, BIOS rozróżnia logiczny port komunikacyjny i fizyczny port komunikacyjny. BIOS przechowuje adres bazowe COM1:...COM4: w komórkach pamięci 40:0, 40:2. 40:4 i 40:6. Ten adres bazowy jest jst adresem I/O pierwszego rejestru 8250 dla tego szczególnego portu komunikacyjnego.

*"Chip komunikacji szeregowej 8250"
*"Rejestr danych (rejestr nadawczy / odbiorczy"
*"Rejestr zezwolenia na przerwanie (IER)"
*"Dzielnik szybkości transmisji"
*"Rejestr identyfikacji przerwań (IIR)"
*"Rejestr sterowania łączem"
*"Rejestr sterowanie modemem"
*"Rejestr stanu łącza (LSR)"
*"Rejestr stanu modemu (MSR)"
*"Pomocniczy rejestr wejściowy"

Biblioteka Standardowa UCR dostarcza bardzo rozsądnego zbioru podprogramów jakie możemy użyć do sterowania portem szeregowym w PC. Pakiet ten dostarcza nie tylko zbioru podprogramów odpytujących ,których możemy użyć podobnie jak kodu BIOS'a ale również podprogramów obsługi przerwań wspierających układy I/O sterowane przerwaniami w porcie szeregowym

*"Podprogramy wspierające komunikację szeregową biblioteki standardowej UCR"

Podprogramy szeregowych I/O Biblioteki Standardowej dostarczają doskonałych przykładów jak oprogramować 8250 SCC. Dlatego też, rozdział ten kończy się przedstawieniem i objaśnieniem podprogramów szeregowych I/O Biblioteki Standardowej. W szczególności, kod ten demonstruje pewne subtelne problemy z komunikacją szeregową sterowaną przerwaniami

*"Oprogramowanie 8250 (Przykłady z Biblioteki Standardowej) „