ROZDZIAŁ DWUDZIESTY PIERWSZY: **PORTY RÓWNOLEGŁE**

Oryginalny projekt IBM'owski dostarczał wsparcia dla trzech portów równoległych drukarki, które IBM nazwał LPT1:,LPT2:, i LPT3:. IBM prawdopodobnie przewidział maszyny, które będą wspierały drukarki mozaikowe, drukarkę z głowicą wirującą i być może inne typy drukarek dla różnych celów, wszystkie na jednej maszynie (drukarki laserowe miały pojawić się dopiero kilka lat później) . Z pewnością IBM nie przewidywał ogólnego zastosowania tych portów równoległych, gdyż prawdopodobnie zaprojektował by je inaczej. Dzisiaj, porty równoległe PC sterują klawiaturą, dyskami, streamerami, kontrolerami SCSI, kontrolerami ethernet (lub innymi sieciowymi), kontrolerem joysticka, pomocniczymi blokami klawiszy i różnymi urządzeniami, no i oczywiście drukarkami. Rozdział ten nie będzie próbował opisywać jak stosować port równoległy dla tych wszystkich różnych celów – ta książka już jest dość duża. Jednak gruntowne omówienie jak interfejs równoległy steruje drukarką i aplikacją portu równoległego (komunikacja krzyżowa) powinno dostarczyć nam dosyć pomysłów na implementację kolejnego wielkiego urządzenia równoległego.

21.1 PODSTAWOWE INFORMACJE O PORCIE RÓWNOLEGŁYM

Są dwa podstawowe metody transmisji danych nowoczesnych obliczeń: równoległa transmisja danych szeregowa transmisja danych. Przy szeregowej transmisji danych (zobacz „Port szeregowy PC") jedno urządzenie wysyła dane do innego jako pojedynczy bit w czasie po jednej linii. W transmisji równoległej, jedno urządzenie wysyła dane do innego jako kilka bitów w czasie (równolegle) kilkoma różnymi liniami. Na przykład, port równoległy PC dostarcza ośmiu lini danych w porównaniu do jednej lini danych portu szeregowego. Dlatego też, wydawałoby się, że port równoległy mógłby transmitować dane osiem razy szybciej ponieważ jest osiem razy więcej linii w kablu. Podobnie wydawałoby się, że kabel szeregowy, w takiej samej cenie jak kabel równoległy, mógłby iść osiem razy wolniej ponieważ jest mnij lini w kablu. Mamy kolejny problem z metodami komunikacji równoległej kontra szeregowej: szybkość kontra koszt.

W praktyce, komunikacja równoległa nie jest osiem razy szybsza niż komunikacja szeregowa, także kable równoległe nie kosztują osiem razy więcej. Generalnie, ci, którzy projektowali kable szeregowe (np. kable ethernetowe) użyli najlepszych materiałów i ekranowania. Podnosi to koszt kabli ale pozwala transmitować dane, bit w czasie, dużo szybciej. Co więcej, lepsze kable pozwalają na większy dystans pomiędzy urządzeniami. Kable równoległe, z drugiej strony, są generalnie tańsze i zaprojektowane dla bardzo krótkich połączeń (mniej więcej od sześciu do dziesięciu stóp). Problemy świata rzeczywistego z szumem elektrycznym i przesłuchem tworzy problemy kiedy używamy długich kabli równoległych i ogranicza szybkość systemu przy transmisji danych. Faktycznie, oryginalna specyfikacja portu drukarki Centronics wskazuje na nie więcej niż 1000 znaków/ sekundę częstotliwości transmisji danych, więc wiele drukarek zaprojektowano do obsługi danych dla tej częstotliwości transmisji. Większość portów równoległych może łatwo przewyższyć osiągami tą wartość; jednakże czynnikiem ograniczającym jest jeszcze kabel, żadne wrodzone ograniczenie w nowoczesnych komputerach.

Chociaż system komunikacji równoległej może używać różnej liczby linii do transmisji danych, większość systemów równoległych używa ośmiu lini danych do transmisji bajtu w czasie. Jest kilka godnych uwagi wyjątków. Na przykład interfejs SCSI jest interfejsem równoległym, nowsze wersje standardu SCSI pozwalają na ośmio- szesnasto- a nawet trzydziesto dwu bitowy transfer danych. W rozdziale tym skoncentrujemy się na transferach o rozmiarze bajta ponieważ port równoległy PC dostarcza ośmiobitowej danej
.

Typowy system komunikacji równoległej może być jednokierunkowy (unidirectional) lub dwu kierunkowy (bidirectional). Port równoległy PC wspiera komunikację jednokierunkowa (z PC do drukarki), więc rozpatrzymy najpierw ten najprostszy przypadek.

W systemie jednokierunkowej komunikacji równoległej są dwie rozpoznawalne węzły: węzeł transmisji i węzeł odbioru. Węzeł transmisji umieszcza dane na lini danych i informuje węzeł odbioru , że dana jest dostępna; wtedy węzeł odbioru odczytuje linię danych i informuje węzeł transmisji ,że pobrał dane. Odnotujmy jak te dwa węzły synchronizują swój dostęp do lini danych – węzeł odbioru nie odczytuje lini danych dopóki węzeł transmisji nie przekaże mu tego, węzeł transmisji nie umieszcza nowej wartości na lini danych dopóki węzeł odbioru nie usunie danych i nie przekaże węzłowi transmisji, że ma dane. Uzgodnienie (handshaking) jest terminem, które opisuje jak te dwa węzły koosdynuja transfer danych.

Właściwa implementacja uzgodnienia wymaga dwóch dodatkowych linii. Linia strobe (lub strobowanie danych) jest tym czego używa węzeł transmisji do przekazania węzłowi odbioru , że dana jest dostępna. Linia acknowledge (potwierdzenia) jest tym czego węzeł odbioru używa do przekazania węzłowi transmisji, że pobrał już dane i jest gotów na więcej. W rzeczywistości port równoległy dostarcza trzeciej lini uzgodnienia, busy, której węzeł odbiorczy może użyć do przekazania węzłowi transmisji, że jest zajęty i węzeł transmisji nie próbował wysyłać danych. Typowa sesja transmisji danych wygląda podobnie jak następuje:

Węzeł transmisji:

1) Węzeł transmisji sprawdza linię busy aby sprawdzić czy odbiór jest zajęty. Jeśli linia busy jest aktywna, nadajnik czeka w pętli dopóki linia busy stanie się nieaktywna
2) Węzeł transmisji umieszcza dane na lini danych
3) Węzeł transmisji aktywuje linię strobe
4) Węzeł transmisji czeka w pętli aż linia potwierdzenia stanie się aktywna
5) Węzeł transmisji ustawia nieaktywna strobe
6) Węzeł transmisji czeka w pętli aż linia potwierdzenia stanie się nieaktywna
7) Tan transmisji powtarza kroki od jeden do sześć dla każdego bajtu jaki musi przesłać

Węzeł odbiorczy:

1) Węzeł odbiorczy ustawia linię busy nieaktywną (zakładając gotowość do akceptacji danej)
2) Węzeł odbiorczy oczekuje w pętli dopóki linia strobe nie stanie się aktywna.
3) Węzeł odbiorczy odczytuje daną z lini danych (i przetwarza tą daną, jeśli to konieczne)
4) Węzeł odbiorczy aktywuje linię potwierdzenia
5) Węzeł odbiorczy oczekuje w pętli dopóki linia strobe nie stanie się nieaktywna
6) Węzeł odbiorczy ustawia nieaktywną linię potwierdzenia
7) Węzeł odbiorczy powtarza kroki od jeden do sześć dla każdego dodatkowego bajtu jaką musi odebrać

Ostrożnie korzystajmy z tych kroków, węzły odbiorczy i transmisji starannie koordynują swoje działania więc węzeł transmisji nie próbuje odłożyć kilku bajtów na linie danych zanim węzeł odbiorczy nie skonsumuje ich a węzeł odbiorczy nie próbuje czytać danych, których nie wysłał węzeł transmisji.

Dwukierunkowa transmisja danych jest często niczym więcej niż dwom jednokierunkowymi transmisjami danych z rolą węzła transmisji i odbiorczego odwróconą dla drugiego kanału komunikacji. Niektóre porty równoległe PC (szczególnie w systemach PS/2 i wielu notebookach) dostarczają dwukierunkowego portu równoległego. Dwukierunkowa transmisja danych na takim sprzęcie nieco bardziej złożona niż w systemach, które implementują komunikację dwukierunkową z dwóch portów jednokierunkowych. Komunikacja dwukierunkowa w dwukierunkowym porcie równoległym wymaga dodatkowego zbioru lini sterujących, więc te dwa węzły mogą określić kto zapisuje do wspólnej lini danych w czasie.

21.2 SPRZĘT PORTU RÓWNOLEGŁEGO

Standardowy jednokierunkowy port równoległy w PC dostarcza więcej niż 11 lini opisanych w poprzedniej sekcji (osiem lini danych , trzy linie uzgodnienia). Port równoległy PC dostarcza następujących sygnałów:

Numer końcówki złącza	Kierunek I/O	Aktywność Biegunowość	Opis sygnałów
1	wyjście	0	Strobe (sygnał dostępnej danej
2 –9	wyjście	-	Linia danych (bit 0 to pin 2, bit 7 to pin 9
10	wejście	0	Linia potwierdzenia (aktywna kiedy zdalny system pobrał daną)
11	wejście	0	Linia busy (aktywna kiedy system

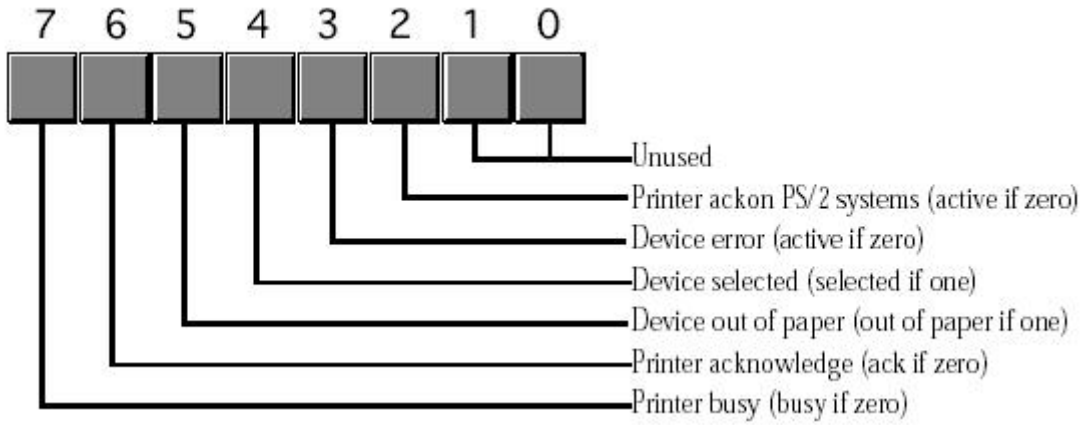
			zdalny jest zajęty i nie można zaakceptować danej
12	wejście	1	Brak papieru (aktywna kiedy w drukarce brak papieru)
13	wejście	1	Wybór. Aktywna kiedy jest wybrana drukarka.
14	wyjście	0	Autoprzesuw. Aktywna kiedy drukarka automatycznie przesuwa linię po każdym powrocie karetki
15	wejście	0	Błąd. Aktywna kiedy mamy błąd drukarki
16	wyjście	0	Inicjalizacja. Sygnał ten powoduje, że drukarka sam się inicjalizuje.
17	wyjście	0	Wybór wejścia. Sygnał ten, kiedy jest nieaktywny, wymusza autonomiczną drukarkę
18 - 25	-	-	Sygnał uziemienia

Tablica 79; Sygnały portu równoległego

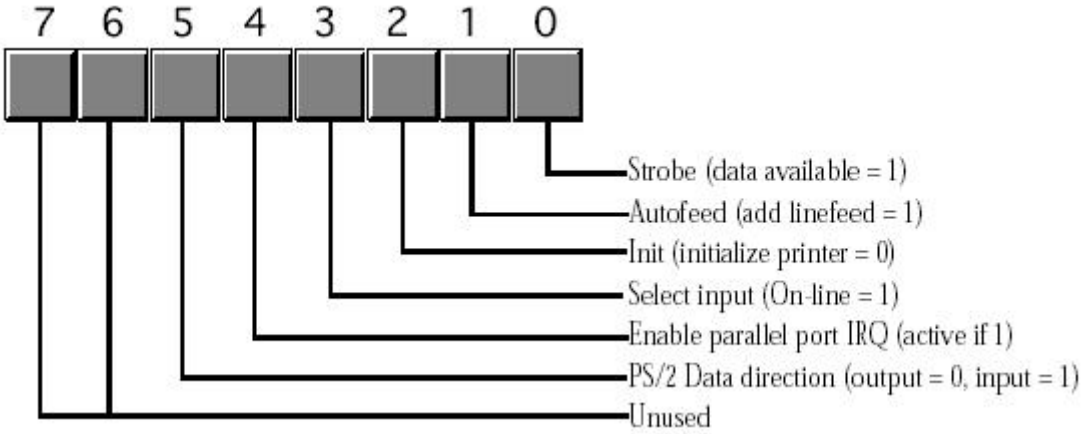
Zauważmy, że port równoległy dostarcza 12 lini wyjściowych (osiem lini danych, strobe, autoprzesuwania, inicjalizacji i wyboru wejścia0 i pięć lini wejściowych (potwierdzenia, busy, brak papieru, wyboru i błędu). Pomimo, że port jest jednokierunkowy, jest dobrą mieszanką dostępnych lini wejściowych i wyjściowych w porcie. Wiele urządzeń (jak dysk lub streamer), które wymagają dwukierunkowego transferu danych używają tych dodatkowych lini do wykonania dwukierunkowego transferu danych

W dwukierunkowym porcie równoległym (system PS/2 i laptopy), linia danych i strobe, oba , są liniami wejściowymi i wejściowymi. Jest bit w rejestrze sterującym powiązanym z portem równoległym, który jest wybrany, który steruje kierunkiem w danym momencie (nie możemy przekazać danych w obu kierunkach równocześnie).

Są trzy adresy I/O powiązane z typowym PC porcie równoległym. Adres te należą do rejestru danych, rejestru statusu i rejestru sterującego. Rejestr danych jest ośmiobitowym portem odczyt / zapis. Odczytując rejestr danych (w trybie jednokierunkowym) zwracamy wartość ostatnio zapisaną do rejestru danych. Rejestr sterujący i statusu dostarcza interfejsu do innych lini I/O. Organizacja tego portu jest następująca:



Bit dwa (potwierdzenie drukarki) jest dostępny tylko na PS/2 i innych systemach, które wspierają dwukierunkowy port drukarki. Inne systemy nie używają tego bitu



Rejestr sterujący portu równoległego jest rejestrem wyjściowym. Odczytując tą lokację zwracamy ostatnią wartość zapisaną do rejestru sterującego z wyjątkiem bitu pięć, który jest tylko do zapisu. Bit pięć, bit kierunku danej, jest dostępny tylko w PS/2 i innych systemach, które wspierają dwukierunkowy port równoległy. Jeśli zapiszemy zero do tego bitu, linia strobe i danej są bitami wyjściowymi, podobnie jak jednokierunkowy port równoległy. Jeśli zapisujemy jeden do tego bitu, wtedy linie strobe i danej są wejściowe. Odnotujmy, że w trybie wejściowym (bit 5 =1), bit zero rejestru sterującego jest w rzeczywistości wejściowym. Notka: zapiszmy jeden do bitu cztery rejestru sterującego odblokowującego IRQ drukarki (IRQ 7). Jednakże , cecha ta nie działa na wszystkich systemach , więc bardzo mało programów próbuje używać przerwań z portu równoległego. Kiedy jest aktywny, port równoległy będzie generował int 0Fh kiedy drukarka potwierdza transmisję danych.

Ponieważ PC wspiera do trzech oddzielnych portów równoległych, może być nie mniej niż trzy zbiory tych rejestrów portów równoległych w systemie w tym samym czasie. Są trzy adresy bazowe portu równoległego powiązane z trzema możliwymi portami równoległymi :3BCh, 378h i 278h. Będziemy się odnosili do tego jako adresów bazowych dla LPT1: , LPT2:, i LPT3:, odpowiednio. Rejestr danych portu równoległego jest zawsze ulokowany pod adresem bazowym dla portu równoległego, rejestr stanu pojawia się pod adresem bazowym plus jeden a rejestr sterujący pojawia się pod adresem bazowym plus dwa. Na przykład , dla LPT1:, rejestr danych jest pod adresem I/O 3BCh, rejestr statusu pod adresem I/O 3BDh a rejestr sterujący pod adresem I/O 3BEh.

Jest jedno ważne zakłócenie. Adresy I/O dla LPT1:, LPT2:, i LPT3: dane powyżej są adresami fizycznymi dla portu równoległego. BIOS dostarcza również adresów logicznych dla tych portów równoległych. Pozwala to użytkownikom ponownie odwzorować ich drukarki (ponieważ większość programów tylko zapisuje do LPT1. Wykonując to, BIOS rezerwuje osiem bajtów w przestrzeni zmiennej BIOS (40:8, 40:0A, 40:0C i 40:0E). Lokacja 40:8 zawiera adres bazowy dla logicznego LPT1: , lokacja 40:0A zawiera adres bazowy dla logicznego LPT2:, itd. Kiedy oprogramowanie uzyskuje dostęp do LPT1:, LPT2:, itd., generalnie uzyskuje dostęp do portu równoległego, którego adres bazowy pojawia się w jednej z tych lokacji.

21.3 STEROWANIE DRUKARKI POPRZEZ PORT RÓWNOLEGŁY

Chociaż jest wiele urządzeń które przyłącza się do portu równoległego PC, drukarki wykonują największą ilość takich połączeń. Dlatego też, opisanie jak sterować drukarką z portu równoległego PC jest prawdopodobnie najlepszym pierwszym przykładem do przedstawienia. Jak przy klawiaturze, nasze oprogramowanie może działać na trzech różnych poziomach: może drukować dane stosując DOS, stosując BIOS lub przez zapisanie bezpośrednio do sprzętu portu równoległego. Podobnie jak przy interfejsie klawiatury , stosowanie DOS lub BIOS jest najlepszym podejściem jeśli chcemy utrzymać kompatybilność z innymi urządzeniami podłączonymi do portu równoległego. Oczywiście, jeśli sterujemy jakimś innym typem urządzenia, przejście bezpośrednio do sprzętu jest tylko naszym wyborem. Jednakże, BIOS dostarcza dobrego wsparcia dla drukarek, więc podejście bezpośrednio do sprzętu jest rzadko koniecznością jeśli po prostu chcemy wysłać dane do drukarki.

21.3.1 DRUKOWANIE POPRZEZ DOS

MS-DOS dostarcza dwóch funkcji jakie możemy użyć wysyłając dane do drukarki. Funkcja DOS'a 05h zapisuje znak w rejestrze dl bezpośrednio do drukarki. Funkcja 40h, z logicznym numerem pliku 04h, również wysyła daną do drukarki. Ponieważ rozdział o DOS i BIOS dokładnie opisał te funkcje, nie będziemy ich omawiać dalej tutaj.

21.3.2 DRUKOWANIE POPRZEZ BIOS

Chociaż DOS dostarcza stosownego zbioru funkcji dla wysyłania znaków do drukarki, nie dostarcza funkcji, która pozwoli nam zainicjalizować drukarki lub uzyskać bieżącego stanu drukarki. Dlatego też DOS tylko drukuje do LPT1:. Podprogram BIOS'a PC int 17h dostarcza trzech funkcji, drukuj, inicjalizuj i status. Możemy zastosować te funkcje do każdego portu równoległego w systemie. Funkcja drukuj jest przybliżonym odpowiednikiem funkcji DOS'a drukowani znaku. Funkcja inicjalizacji inicjalizuje drukarkę przy użyciu systemowej informacji zależnej czasowo. Status drukarki zwraca informację z portu stanu drukarki wraz z informacją limitu czasu.

21.3.3 PODPROGRAM OBSŁUGI PRZERWANIA INT 17H

Być może najlepszym sposobem zobaczenia jak funkcje BIOS działają jest napisanie zastępczego ISR'a 17h dla drukarki. Sekcja ta wyjaśni protokół uzgodnienia i zmienne używane przez drukarkę. Opisuje również działanie i zwrot wyniku powiązanego z każdą maszyną.

Jest osiem zmiennych w przestrzeni zmiennych BIOS (segment 40h) jakich używa drukarka. Poniższa tabela opisuje każdą z tych zmiennych.

Adres	Opis
40:08	Adres bazowy urządzenia LPT1:
40:0A	Adres bazowy urządzenia LPT2:
40:0C	Adres bazowy urządzenia LPT3:
40:0E	Adres bazowy urządzenia LPT4:
40:78	Wartość ograniczenia czasu LPT1:. Oprogramowanie portu drukarki powinno zwracać błąd jeśli drukarka nie odpowiada w stosownej ilości czasu. Zmienna ta (jeśli nie zero) określa jak wiele pętli z 65,536 iteracji urządzenie będzie oczekiwało na potwierdzenie z drukarki. Jeśli zero, urządzenie będzie czekało zawsze
40:79	Wartość ograniczenia czasu LPT2: .Jak wyżej
40:7A	Wartość ograniczenia czasu LPT3: .Jak wyżej
40:7B	Wartość ograniczenia czasu LPT4: .Jak wyżej

Tablica 80: Zmienne BIOS portu równoległego

Zwrócimy uwagę na drobne odchylenie w protokole uzgodnienia w powyższym kodzie. Sterownik drukarki nie oczekuje na potwierdzenie z drukarki po wysłaniu znaku. Zamiast tego, sprawdza aby zobaczyć czy drukarka wysłała potwierdzenie dla poprzedniego znaku zanim wyśle znak. Zajmuje to małą ilość czasu ponieważ program drukując znaki może kontynuować działanie równolegle z odebraniem potwierdzenia z drukarki. Odnotujmy również, że to szczególne urządzenie nie monitoruje lini busy. Prawie każda istniejąca drukarka pozostawia tą linie nieaktywną (not busy), więc nie musimy jej sprawdzać. Jeśli napotkamy drukarkę, która manipuluje linią busy, modyfikacja tego kodu jest banalna. Poniższy kod implementuje usługę int 17h

```
; INT17.ASM
;
; Krótki bierny TSR, który zamienia program obsługi BIOS'a int 17h. Ten podprogram demonstruje
; funkcjonowanie każdej z funkcji int 17h, której standardowo dostarcza BIOS.
;
; Zauważmy, że kod ten nie aktualizuje int 2Fh (przerwania równoczesnych procesów) ani też nie możemy
; usunąć tego kodu z pamięci z wyjątkiem przeładowania. Jeśli chcemy móc zrobić te dwie rzeczy (jak również
; sprawdzić poprzednią instalację), zobacz rozdział o programach rezydentnych. Kod taki zostanie pominięty
; w tym programie z powodu ograniczenia długości
;
; cseg i EndResident muszą pojawić się przed segmentem biblioteki standardowej!

cseg            segment para public 'code'
cseg            ends
```

```
; Oznaczamy segment znajdując koniec sekcji rezydentnej

EndResident     segment para public 'Resident'
EndResident     ends

                .xlist
                include         stdlib.a
                includelib      stdlib.lib
                .list
byp             equ     < byte ptr >

cseg            segment para public 'code'
                assume  cs:cseg, ds:cseg

OldInt17        dword   ?

; Zmienne BIOS:

PrtrBase        equ     8
PrtrTimeOut     equ     78h

; Kod ten  obsługuje działanie INT 17H. INT 17h jest podprogramem BIOS wysyłającym dane do drukarki i
; i raportują  status drukarki. Są trzy różne funkcje dla tego podprogramu , w zależności od zawartości rejestru
; AH. Rejestr DX zawiera numer portu drukarki
;
; DX=0 –Używamy LPT1:
; DX=1 – Używamy LPT2:
; DX=2 – Używamy LPT3:
; DX=3 – Używamy LPT4:
;
; AH=0 --       Drukujemy znak w AL na drukarce. Status drukarki jest zwracany w AH. Jeśli bit #0 = 1 wtedy
;               wystąpi błąd limitu czasu
;
; AH=1 --       Inicjalizacja drukarki. Status zwracany w AH
;
; AH=2 --       Zwrot statusu drukarki w AH
;
; Bity statusu zwracane w AH są takie jak następuje
; Bit   Funkcja                                 Wartość bez błędu
; ----  ----------                              -----------------------
; 0     1 = błąd limitu czasu                           0
; 1     nie używane                                     x
; 2     nie używane                                     x
; 3     1 = błąd I/O                                    0
; 4     1 = wybrany, 0 = nie wybrany                    1
; 5     1 = brak papieru                                0
; 6     1 = potwierdzenie                               x
; 7     1 = nie zajęta                                  x
;
; Zauważmy, że sprzęt zwraca bit 3 z zerem jeśli wystąpił błąd, z jedynką jeśli nie  ma błędu. Program zazwyczaj
; odwraca ten bit przed zwróceniem go do programu wywołującego
;
; Lokacje sprzętowego portu drukarki:
;
; PrtrPortAdrs --       Port wyjściowy  gdzie dana jest wysyłana do drukarki (8 bitów)
; PrtrPortAdrs+1 --     Port wejściowy gdzie może być  odczytany status drukarki (8 bitów)
; PrtrPortAdrs+2 --     Port wyjściowy gdzie informacja sterująca jest wysyłana  do drukarki
```

```
;
; Port wyjściowy danych – 8 – bitowa dana jest transmitowana do drukarki przez ten port
;
; Status portu wejściowego:
;               bit 0:          nie używany
;               bit 1:          nie używany
;               bit 2:          nie używany
;               bit 3;          - Błąd, zazwyczaj ten bit oznacza, że drukarka napotkała błąd. Jednakże z
;                               zainstalowanym P101 jest to dana lini sygnału zwrotnego dla skanowania
;                               klawiatury
;
;               bit 4:          +SLTC, zazwyczaj bit ten jest używany do określenia czy drukarka jest
;                               wybrana czy nie. Z zainstalowanym P101 jest to dana lini sygnału zwrotnego
;                               skanowanej klawiatury
;
;               bit 5:          +PE, 1 w tym bicie oznacza ,że drukarka wykryła koniec papieru. Na wielu
;                               portach drukarek , bit ten jest nieczynny
;
;               bit 6:          -ACK, zero w tym bicie oznacza, ze drukarka zaakceptowała ostatni znak i
;                               jest gotowa do przyjęcia kolejnego. Bit ten nie jest zazwyczaj używany przez
;                               BIOS ponieważ bit 7 również spełnia  taką funkcję (lub więcej)
;
;               bit 7:          -Busy, kiedy ten sygnał  jest aktywny (0) wtedy drukarka jest zajęta i nie
;                               może zaakceptować danej. Kiedy bit ten  jest ustawiony na jeden, drukarka
;                               może zaakceptować kolejny znak
;
; Wyjściowy port sterujący:
;               bit 0:          +Strobe, 0,5 μs (minimum) aktywny impuls na tym bicie zegarowym
;                               zamyka dane portu wyjściowego danych drukarki przed drukarką
;               bit 1:          +Auto FD  XT – 1 przechowywane pod tym bitem powoduje, że drukarka
;                               przesuwa linię po lini wydrukowanej. W pewnych interfejsach
;                               drukarek (np. karta graficzna Hercules) bit ten jest nieczynny
;
;               bit 2:          -INIT, zero w tym bicie (dla minimum 50 us) bezie powodował, że
;                               drukarka sama będzie się (re)inicjalizowała
;
;               bit 3:          +SLCT, jeden w tym bicie wybiera  drukarkę. Zero spowoduje przejście
;                               drukarki do trybu off –line
;
;               bit 4:          +IRQ ENABLE, jeden w tym bicie pozwala na wystąpienie przerwań
;                               kiedy –ACK zmieni się z jeden na zero
;               bit 5:          Sterowanie kierunkiem w porcie dwukierunkowym. 0 =
;                               wyjście, 1= wejście
;
;               bit 6:          zarezerwowane, musi być zerem
;               bit 7:          zarezerwowane, musi być zerem
;

MyInt17         proc    far
                assume  ds.:nothing

                push    ds
                push    bx
                push    cx
                push    dx

                mov     bx, 40h                 ;DS wskazuje zmienne BIOS
                mov     ds., bx
```

```
                cmp     dx, 3                   ;musi być LPT1...LPT4
                ja      InvalidPrtr

                cmp     ah, 0                   ;skocz do właściwego kodu dla funkcji drukowania
                jz      PrtChar
                cmp     ah, 2
                jb      PrtrInit
                je      PrtrStatus

; Jeśli przekazano nam opcod jakiego nie znaliśmy, wracamy

InvalidPrtr:    jmp     ISR17Done

;Inicjalizujemy drukarkę poprzez impulsowanie lini init dla przynajmniej 50 µs. Poniższa pętla
; opóźniająca będzie opóźniała dobrze ponad 50 µs nawet na szybszych maszynach

PrtrInit:       mov     bx, dx              ;pobranie wartości portu drukarki
                shl     bx, 1               ;konwersja do bajtu indeksu
                mov     dx, PrtrBase[dx]    ;pobranie adresu bazowego drukarki
                test    dx, dx              ;czy ta drukarka istnieje?
                je      InvalidPrtr         ;wyjście jeśli nie ma takiej  drukarki
                add     dx, 2               ;dx wskazuje na rejestr sterujący
                in      al., dx             ;odczyt bieżącego statusu
                and     al., 11011011b      ; zerowanie bitów INIT/BIDIR
                out     dx, al              ;reset drukarki
                mov     cx, 0               ; to będzie tworzyło przynajmniej 50 µs opóźnienie
PIDelay:        loop    PIDelay
                or      al., 100b           ;zatrzymanie resetu drukarki
                out     dx, al.
                jmp     ISR17Done

; Zwracamy bieżący status drukarki. Kod odczytuje status portu drukarki i formatuje bity do zwrotu do kodu
; wywołującego

PrtrStatus:     mov     bx, dx                  ;pobranie wartości portu drukarki
                shl     bx, 1                   ; konwersja do bajtu indeksu
                mov     dx, PrtrBase[bx]        ;adres bazowy portu drukarki
                mov     al., 00101001b          ;Domyślnie: każdy możliwy błąd
                test    dx, dx                  ;czy ta drukarka istnieje?
                je      InvalidPrtr             ;wychodzimy jeśli nie
                inc     dx                      ;wskazuje status portu
                in      al., dx                 ;odczyt statusu portu
                and     al., 11110000b      ;zerowanie bitów nieużywanych / przekroczenia czasu
                jmp     ISR17Done

;Druk znaku w akumulatorze!

PrtChar:        mov     bx, dx
                mov     cl, PrtrTimeOut[bx]     ;pobranie wartości przekroczenia czasu
                shl     bx, 1                   ;konwersja do bajtu indeksu
                mov     dx, PrtrBase[bx]        ;pobranie adresu portu drukarki
                or      dx, dx                  ;wskaźnik nie zerowy?
                jz      NoPrtr2                 ;skok jeśli zerowy

;Poniższy kod sprawdza aby zobaczyć czy z drukarki zostało odebrane potwierdzenie. Jeśli ten kod czeka zbyt
; długo, jest zwracany błąd przekroczenia czasu. Potwierdzenie jest dostarczane w bicie 7 portu statusu drukarki
; (który jest kolejnym adresem po porcie danych drukarki)

                push    ax
```

```
                inc     dx                      ;wskazuje status portu
                mov     bl, cl                  ;włożenie wartości przekroczenia czasu do bl I bh
                mov     bh, cl
WaitLp1:        xor     cx, cx                  ;inicjalizacja licznika 65536
WaitLp2:        in      al., dx                 ;odczyt statusu portu
                mov     ah, al                  ;zachowanie statusu
                test    al., 80h                ;potwierdzenie drukarki?
                jnz     GotAck                  ;skok jeśli potwierdzenie
                loop    WaitLp2                 ;powtarzanie 65536 razy
                dec     bl                      ;zmniejszanie wartości przekroczonego czasu
                jnz     WaitLp1                 ;powtarzanie 65536*TimeOut razy

;Zobaczmy czy użytkownik wybrał czas przekroczenia:

                cmp     bh, 0
                je      WaitLp1
; WYSTĄPIŁ BŁĄD PRZEKROCZENIA CZASU!
;
; Przekroczenie czasu – błąd I/O jest zwracany do systemu przez ten port. Albo nie dochodzi do skutku
; ten punkt  (błąd przekroczenia czasu) albo odnośny port drukarki nie istnieje. W innym przypadku zwraca
; błąd

NoPrtr2:        or      ah, 9                   ;ustawienie flagi błędu I/O – przekroczenia czasu
                and     ah, 0F9h                ;wyłączenie nie używanych flag
                xor     ah, 40h                 ;

;Okay, przywracamy rejestry i wracamy do kodu wywołującego
                pop     cx                      ;usunięcie starego ax
                mov     al., cl                 ;przywrócenie starego al
                jmp     ISR17Done

;Jeśli port drukarki istnieje i odebraliśmy potwierdzenie, wtedy jest możliwość przekazania danych do drukarki.

GotAck:         mov     cx, 16                  ;krótkie opóźnienie jeśli drukarka potrzebuje czasu
                loop    GALp                    ; po potwierdzeniu
                pop     ax                      ;pobranie znaku na wyjściu i ponowne zachowanie
                push    ax
                dec     dx                      ;DX wskazuje port drukarki
                pushf                           ;wyłączenie przerwań
                cli
                out     dx, al.                 ;dane wyjściowe do drukarki

; Poniższe krótkie opóźnienie daje danym czas na przeniesienie przez linie równoległe. To zapewnia, że dane
; przybywają do drukarki przed strobe (czas ten może zależeć od przepustowości kabli lini równoległej)

                mov     cx, 16                          ; czas danej przed wysłaniem strobe
DataSettleLp:   loop    DataSettleLp

; Teraz dana została zamknięta w porcie wyjściowym drukarki, strobe musi zostać wysłany do drukarki. Linia
; strobe jest połączona do bitu zero portu sterującego. Odnotujmy również, że czyści to bit 5 portu sterującego.
; Zapewnia to, że port  kontynuuje działania porcie wyjściowym jeśli jest to urządzenie dwukierunkowe. Kod
ten
; również czyści bity sześć i siedem, które IBM zaleca ustawiać na zero

                inc     dx                      ;DX wskazuje na wyjściowy port
drukarki
                inc     dx
                in      al., dx                 ;pobranie bieżących bitów sterujących
                and     al., 01eh               ;wymuszenie lini strobe na zero
```

```
                out     dx, al.                         ;i upewnienie się ,że to port wyjściowy

                mov     cx, 16                          ;krótkie opóźnienie pozwalające danym
Delay0:         loop    Delay0                          ;stać się dobrymi

                or      al., 1                          ;wys³¹nie (+) strobe
                out     dx, al                          ; wyjściowy (+) strobe do bitu 0

                mov     cx, 16                          ;krótkie opóźnienie wydłużające strobe
StrobeDelay:    loop    StrobeDelay

                and     al., 0Feh                       ;zerowanie bitu strobe
                out     dx, al.                         ; wyjście do portu sterującego
                popf                                    ;przywrócenie przerwań

                pop     dx                              ;pobranie starej wartości AX
                mov     al., dl                         ‘przywrócenie starej wartości AL.

ISR17Done:      pop     dx
                pop     cx
                pop     bx
                pop     ds.
                iret
MyInt17         endp

Main            proc

                mov     ax, cseg
                mov     ds, ax

                print
                byte    “INT 17 Replacement”,cr, lf
                byet    “Installing…”, cr,lf,0
```

; Aktualizujemy wektor przerwania INT 17. Zauważmy, że powyższe instrukcje uczyniły cseg bieżącym
;segmentem danych, więc możemy przechować starą wartość INT 17 bezpośrednio w zmiennej OldInt17

```
                cli                                     ;wyłączenie przerwań
                mov     ax, 0
                mov     es, ax
                mov     ax, es:[17h*4]
                mov     word  ptr OldInt17, ax
                mov     ax, es:[17h*4+2]
                mov     word  ptr OldInt17+2, ax
                mov     es:[17h*4], offset MyInt17
                mov     es:[17h*4+2], cs
                sti                                     ;Ok, załączamy przerwania
```

; Jedyne co pozostało to zakończyć i pozostawić w pamięci

```
                print
                byte    „Installed.”,cr, lf,0

                mov     ah, 62h                         ;pobranie wartości PSP programu
                int     21h

                mov     dx, EndResident                 ;obliczamy rozmiar programu
                sub     dx, bx
                mov     ax, 3100h                       ;polecenie TSR DOS’a
```

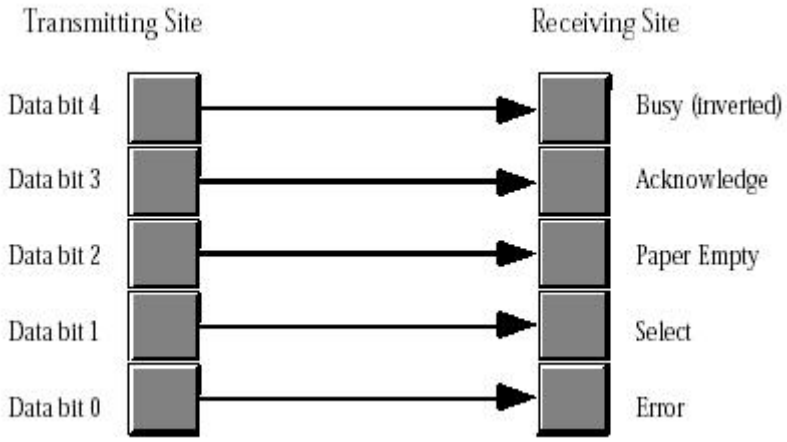
```
                int     21h
Main            endp
cseg            ends

sseg            segment para stack 'stack'
stk             byte    1024 dup ('stack')
sseg            ends

zzzzzzseg       segemnt para public 'zzzzzz'
LastBytes       byte    16  dup  (?)
zzzzzzseg       ends
                end     Main
```

21.4 MIĘDZY KOMPUTEROWA KOMUNIKACJA PORTEM RÓWNOLEGŁYM

Chociaż drukowanie jest najpopularniejszym zastosowaniem dla portu równoległego na PC, wiele urządzeń stosuje port równoległy dla innych celów, jak wspomniano wcześniej. Nie pasowałoby zamknąć tego rozdziału bez przynajmniej jednego przykładu aplikacji nie drukarkowej portu równoległego .Sekcja ta bezie opisywała jak ustawić dwa komputery do przekazywania plików z jednego do drugiego z wykorzystaniem portu równoległego.

Program Laplink™ firmy Travelling Software jest dobrym przykładem produktu komercyjnego, który może przekazać dane przez port równoległy PC; chociaż poniższe oprogramowanie nie jest tak silne jak Laplink, demonstruje podstawowe zasady takiego oprogramowania.

Zauważmy, że nie możemy połączyć dwóch portów równoległych komputerów prostym kablem, który ma łącza DB25 na każdym końcu. Faktycznie robiąc tak możemy uszkodzić porty równoległe komputerów ponieważ połączylibyśmy cyfrowe wyjście do cyfrowego wyjścia (w rzeczywistości nie –nie) . Jednakże, zakupując kable „kompatybilne z Laplinkiem" (lub rzeczywisty kabel Laplink do tego celu) mamy poprawne połączenie pomiędzy portami równoległymi dwóch komputerów. Jak możemy sobie przypomnieć z sekcji o sprzęcie portu równoległego, port równoległy jednokierunkowy dostarcza pięć sygnałów wejściowych. Kabel Laplink wyznacza drogę czterech lini danych do czterech lini wejściowych w obu kierunkach. Połączenia w kompatybilnym z Laplink kablu pokazano jak następuje:



Dane zapisywane w bitach od zero do trzy rejestru danych węzła transmisji pojawiają się , nie zmienione, w bitach od trzy do sześć portu stanu węzła odbiorczego. Bit cztery węzła transmisji pojawia się, odwrócony, w bicie siedem węzła odbiorczego. Zauważmy ,że kable kompatybilne z Laplink są dwukierunkowe. To znaczy, możemy przekazywać dane z jednego węzła do innego używając powyższego połączenia. Jednakże, ponieważ jest tylko pięć bitów w porcie równoległym, musimy przekazać cztery bity danych w jednym czasie 9potrzebujemy jeden bit na daną strobującą). Ponieważ węzeł odbiorczy musi potwierdzić transmisję danych, nie możemy zasymulować transmisji danych w obu kierunkach. Musimy użyć jednej z lini wyjściowych węzła odbiorczego danych do potwierdzenia przychodzącej danej.

Ponieważ dwa węzły współpracują w transferze danych przez kabel równoległy, muszą jedna po drugiej przesyłać i odbierać dane, muszą utworzyć protokół, aby każdy uczestnik wymiany danych wiedział

kiedy następuje przesył i odbiór danych. Nasz protokół będzie bardzo prosty – węzeł jest albo przekaźnikiem albo odbiorcą, ich role nie będą przełączane. Zaprojektowanie bardziej złożonego protokołu nie jest trudne, ale ten prosty protokół będzie wystarczający dla tego przykładu. Później w tym rozdziale omówimy sposób stworzenia protokołu, który pozwala na transmisję dwukierunkową.

Poniższy przykład będzie przekazywał i odbierał pojedynczy plik przez port równoległy. Używając tego programu, uruchamiamy program przekaźnika w węźle transmisji i program odbiorcy w węźle odbiorczym. Program transmitera pobiera nazwę pliku z lini poleceń DOS i otwiera ten plik do odczytu (generując błąd i wychodząc, jeśli plik nie istnieje). Zakładając, że plik istnieje, program przekaźnika odpytuje węzeł odbiorczy aby zobaczyć czy jest dostępny. Przekaźnik sprawdza obecność węzła odbiorczego poprzez kolejne zapisywanie zer i jedynek do wszystkich bitów wyjściowych potem odczytując jego bity wejściowe. Węzeł odbiorczy będzie odwracał te wartości i zapisywał je z powrotem kiedy będzie on-line. Zauważmy, że porządek wykonania (najpierw przesłanie lub najpierw odbiór) nie ma znaczenia. Te dwa programy będą próbowały uzgadniać dopóki nie nadejdzie coś innego on-line. Kiedy oba węzły dokonają inwersji trzy razy, zapiszą wartość 05h do swoich portów wyjściowych, mówiąc innym węzłom ,że są gotowe do kontynuacji. Funkcja przekroczenia czasu przerywa program jeśli jakiś inny węzeł nie odpowiada prze stosowną ilość czasu.

Ponieważ te dwa węzły są zsynchronizowane, węzeł transmisji określa rozmiar pliku a potem przekazuje nazwę pliku i rozmiar do węzła odbiorczego. Węzeł odbiorczy zaczyna oczekiwanie na odbiór danej.

Węzeł transmisji wysyła 512 bajtów danych do węzła odbiorczego. Po przekazaniu 512 bajtów, węzeł odbiorczy opóźnia wysłanie potwierdzenia i zapisuje 512 bajtów danych na dysk. Potem węzeł odbiorczy wysyła potwierdzenie a węzeł transmisji zaczyna wysyłanie kolejnych 512 bajtów. Proces ten powtarza się , dopóki węzeł odbiorczy nie zaakceptuje wszystkich bajtów z pliku.

Tu mamy kod przekaźnika:

```
; TRANSMIT.ASM
;
; Program ten jest częścią przekaźnikową programu, który przesyła pliki poprzez kompatybilny z Laplink
; kablem równoległym.
;
; Program ten zakłada, że użytkownik chce użyć LPT1: dla transmisji. Modyfikuje przyrównywanie lub czyta
; port z lini poleceń jeśli jest to niewłaściwe.

                .286
                .xlist
                include         stdlib.a
                includelib      stdlib.lib
                .list

dseg            segment para public 'data'

TimeOutConst    equ     4000                    ;około 1 minuty na 66MHz 486
PrtrBase        equ     10                      ; offset adresu LPT1:

MyPortAdrs      word    ?                       ;przechowuje adres portu drukarki
FileHandle      word    ?                       ;obsługa pliku wyjściowego
FileBuffer      byte    512  dup  (?)           ;bufor dla nadchodzących danych

FileSize        dword   ?                       ;rozmiar nadchodzącego pliku
FileNamePtr     dword   ?                       ;przechowuje wskaźnik do pliku

dseg            ends

cseg            segment para public 'code'
                assume  cs:cseg, ds:dseg

; TestAbort-    sprawdza aby zobaczyć czy użytkownik wcisnął ctrl-C i chce przerwać program. Podprogram
;               wywołuje BIOS dla sprawdzenia czy użytkownik nacisnął klawisz. Jeśli tak, wywołuje DOS
;               do odczytu tego klucza (funkcja AH=8, odczyt klucza w/o echo i sprawdzeniem ctrl-C
```

```
TestAbort       proc    near
                push    ax
                push    cx
                push    dx
                mov     ah, 1
                int     16h                     ;zobacz  czy wciśnięto klawisz
                je      NoKeyPress              ;powrót jeśli nie
                mov     ah, 8                   ;odczyt znaku, sprawdzenie na ctrl-c
                int     21h                     ;przerwanie DOS jeśli ctrl-C
NoKeyPress:     pop     dx
                pop     cx
                pop     ax
                ret
TestAbort       endp

; SendByte-     Przekazanie bajtu w AL do czterech bitów węzła odbiorczego w jednym czasie

SendByte        proc    near
                push    cx
                push    dx
                mov     ah, al.                 ;zachowanie bajtu do transmisji

                mov     dx, MyPortAdrs          ; adres bazowy portu LPT1:

; Najpierw, aby być pewnym, zapisujemy zero do bitu #4. Zostanie odczytane jako jeden w bicie
; busy odbiorcy

                mov     al., 0
                out     dx, al.                 ;dana jeszcze nie gotowa

;Czekamy dopóki odbiornik jest zajęty. Odbiornik zapisze zero do bitu #4 do swojego rejestru danych
; kiedy jest zajęty. Wychodzi on jako jeden w naszym bicie busy (bit 7 rejestru stanu). Pętla oczekuje
; dopóki odbiornik nie powie nam, że jest gotowy odebrać daną poprzez zapisanie jeden do bitu #4 (który my
; odczytujemy jako zero). Zauważmy, że sprawdzamy ctrl-C często w przypadku kiedy użytkownik chce
; przerwać transmisję.

                inc     dx                      ;wskazuje rejestr stanu
W4NBLp:         mov     cx, 10000
Wait4NotBusy:   in      al., dx                 ;odczyt wartości rejestru stanu
                test    al., 80h                ;Bit 7 =1 jeśli zajęty
                loopne  Wait4NotBusy            ;powtarzamy kiedy zajęty, 10000 razy
                je      ItsNotBusy              ;opuszczamy pętlę jeśli nie zajęty
                call    TestAbort               ;sprawdzamy dla Ctrl-C
                jmp     W4NBLp

;Okay, wkładamy daną na linie danych:

ItsNotBusy:     dec     dx                      ;wskazuje rejestr danych
                mov     al., ah                 ;pobranie kopii danej
                and     al., 0Fh                ;usunięcie bardziej znaczącego nibble'a
                out     dx, al.                 '"Pierwsza" linia danych, dana nie dostępna
                or      al., 10h                ;włączenie dostępnej danej
                out     dx, al.                 ;wysłanie danej

;Czekamy na potwierdzenie  z węzła odbiorczego.  Co jakiś czas sprawdzamy ctrl-C, czy użytkownik
; może przerwała transmisję programu z wnętrza tej pętli.

                inc     dx                      ;wskazuje rejestr stanu
W4Alp:          mov     cx, 10000               ;czas pętli pomiędzy sprawdzeniem Ctrl-C
```

```
Wait4Ack:       in      al., dx                 ;odczyt stanu portu
                test    al, 80h                 ;Ack = 1 kiedy odbiorca potwierdza
                loope   Wait4Ack                ;powtarzanie 10000 razy lub jeśli potwierdzenia
                jne     GotAck                  ;skok jeśli mamy potwierdzenie
                call    TestAbort               ;sprawdzenie Ctrl - C użytkownika
                jmp     W4Alp

;Wysyłamy daną niedostępnego sygnału do odbiorcy:

GotAck:         dec     dx                      ;wskazuje rejestr danych
                mov     al., 0                  ;zapis zera do bitu 4, pojawia się jako jeden
                out     dx, al.                 ;w bicie busy odbiorcy

;okay ,w bardziej znaczącym nibble'u:

                inc     cx                      ;wskazuje rejestr stanu
W4NB2:          mov     cx, 10000               ;10000 wywołań pomiędzy sprawdzaniem ctrl-C
Wait4NotBusy2:  in      al., dx                 ;odczyt rejestru stanu
                test    al., 80h                ;;Bit 7 = 1 jeśli zajęty
                loopne  Wait4NotBusy            ;bardziej znaczący bit wyzerowany (nie zajęty)?
                call    TestAbort               ;sprawdzenie ctrl-C
                jmp     W4NB2

;Okay, wkładamy daną na linie danych:

NotBusy2:       dec     dx                      ;wskazuje rejestr danych
                mov     al., ah                 ;odzyskujemy dane bardziej znaczącego nibble'a
                shr     al., 4                  ;przesuwamy bardziej znaczący nibble do mniej
znaczącego
                out     dx, al.                 ;"pierwsza" linia danych
                or      al., 10h                ; dana + dana dostępny strobe
                out     dx, al.                 ;wysłanie danej

;Czekamy na potwierdzenie z węzła odbiorczego:

                inc     dx                      ; wskazuje rejestr stanu
W4A2Lp:         mov     cx, 10000
Wait4Ack2:      in      al., dx                 ;odczyt stanu portu
                test    al, 80h                 ;Ack = 1
                loope   Wait4Ack2               ;kiedy brak potwierdzenia
                jne     GotAck2                 ;bardziej znaczący bit = 1 (potwierdzenie)?
                call    TestAbort               ;sprawdzenie ctrl-C
                jmp     W4A2Lp

;wysłanie danej niedostępnego sygnału do odbiorcy:

GotAck2:        dec     dx                      ;wskazuje rejestr danych
                mov     al., 0                  ;zero w bicie 4 (który staje się busy=1 u odbiorcy)
                out     dx, al.

                mov     al, ah                  ;przywrócenie oryginalnej danej w AL
                pop     dx
                pop     cx
                ret
SendByte        endp

; Podprogram synchronizacji:
;
; Send0s-       Przesłanie zera do węzła odbiorczego a potem oczekiwanie aby zobaczyć  czy została
```

```
;               jedynka. Zwraca ustawioną flagę przeniesienia jeśli to działa, wyczyszczoną jeśli nie
;               ustawia jedynki w rozsądnej ilości czasu.

Send0s          proc    near
                push    cx
                push    dx

                mov     dx, MyPortAdrs

                mov     al, 0                   ;zapis wartości początkowego zera do naszego
                out     dx, al.                 ; portu wyjściowego

                xor     cx, cx                  ;sprawdza jedynkę 10000 razy
Wait41s:        inc     dx                      ;wskazuje stan portu
                in      al., dx                 ;odczyt stanu portu
                dec     dx                      ;wskazuje ponownie port danych
                and     al. 78h                 ;maskuje bity wejściowe
                cmp     al., 78h                ;wszystkie jedynki?
                loopne  Wait41s
                je      Got1s                   ;skok jeśli sukces
                clc                             ;zwraca niepowodzenie
                pop     dx
                pop     cx
                ret

Got1s:          stc                             ;zwraca powodzenie
                pop     dx
                pop     cx
                ret
Send0s          endp

; Send1s-       Przesyła wszystkie jedynki do węzła odbiorczego a potem oczekuje aby zobaczyć czy są
;               ustawione ponownie zera. Zwraca ustawioną flagę przeniesienia jeśli działa, wyczyszczoną
;               jeśli nie ma ustawionych zer w rozsądnej ilości czasu

Send1s          proc    near
                push    cx
                push    dx

                mov     dx, MyPortAdrs          ;adres bazowy LPT1:

                mov     al, 0Fh                 ;zapis  wartości “wszystkie jedynki” do
                out     dx, al.                 ;naszego portu wyjściowego

                mov     cx, 0
Wait40s:        inc     dx                      ;wskazuje port wejściowy
                in      al., dx                 ;odczyt portu stanu
                dec     dx                      ;wskazuje ponownie port danych
                and     al., 78h                ;maska bitów wejściowych
                loopne  Wait40s                 ;Pętla dopóki ustawiamy zera
                je      Got0s                   ;wszystkie zera?  Jeśli tak, skok
                clc                             ;zwraca nie powodzenie
                pop     dx
                pop     cx
                ret

Got0s:          stc                             ;zwraca powodzenie
                pop     dx
                pop     cx
```

```
                ret
Send1s          endp

; Synchronizacja-Procedura ta zapisuje wszystkie zera i wszystkie jedynki do swojego portu wyjściowego i
;               sprawdza stan portu wejściowego aby zobaczyć czy węzeł odbiorczy został zsynchronizowany.
;               Kiedy jest zsynchronizowany, zapisze wartość 05h do swojego portu wyjściowego. Więc kiedy
;               ten węzeł zobaczy wartość 05h na swoim porcie wejściowym, oba węzły zostaną
;               zsynchronizowane. Zwraca ustawioną flagę przeniesienia jeśli operacja zakończyła się
;               powodzeniem, wyczyszczoną jeśli niepowodzeniem

Synchronize     proc    near
                print
                byte    „Srynchronizing with receiver program”
                byte    cr, lf, 0

                mov     dx, MportAdrs

                mov     cx, TimeOutConst            ;opóźnienie przekroczenia czasu
SyncLoop:       call    Send0s                      ;wysłanie bitów zero, czekanie na jedynki
                jc      Got1s                       ; (przeniesienie ustawione = jedynki)

; Jeśli nie mamy to na co oczekiwaliśmy, zapisujemy jedynki w tym punkcie i zobaczymy czy poza
; wprowadzeniem węzła odbiorczego

Retry0:         call    Send1s                  ;wysyłamy jedynki, czekamy na zera
                jc      SyncLoop                ;przeniesienie ustawione = zera

;Cóż, nie dostaliśmy jeszcze odpowiedzi, zobaczmy czy użytkownik nacisnął ctrl-C aby przerwać program

DoRetry:        call    TestAbort

;Okay, węzeł odbiorczy już odpowiedział. Wracamy i próbujemy ponownie

                loop    SyncLoop

; Jeśli przekroczyliśmy czas, drukuje informację o błędzie i zwraca wyczyszczoną flagę przeniesienia
; (oznaczającą błąd przekroczenia czasu)

                print
                byte    „Transmit: Timeout error waiting for receiver”
                byte    cr, lf, 0
                clc
                ret

;Okay, zapisaliśmy zera I mamy jedynki. Zapiszmy jedynki i zobaczymy czy mamy zera. Jeśli nie ponawiamy
; pętlę

Got1s:
                call    Send1s                  ;wysyłamy bity jedynek, czekamy na zera
                jnc     DoRetry                 ;(przeniesieni ustawione = zera)

;Dobrze, wydaje się być zsynchronizowane. Dla pewności zróbmy to raz jeszcze

                call    Send0a                  ;wysyłamy zera , czekamy na jedynki
                jnc     Retry0
                call    Snd1s                   ;wysyłamy jedynki, czekamy na zera
                jnc     DoRetry
```

```
; Zsynchronizowaliśmy .Wyślijmy wartość 05h do węzła odbiorczego co pozwoli poznać ,że wszystko jest w
; porządku:

                mov     al., 05h                ;wysłanie sygnału do odbiorcy
                out     dx, al.                 ;aby powiedzieć, że jest synchronizacja

                xor     cx, cx                  ;długie opóźnienie dające czas odbiorcy
FinalDelay:     loop    FinalDelay              ; na przygotowanie

                print
                byte    „Synchronized with receiving site”
                byet    cr, lf, 0
                stc
                ret
Synchronize     endp

; Podprogramy I/O plików:
;
; GetFileInfo-  Otwiera plik określony przez użytkownika i przekazuje nazwę pliku i jego rozmiar do
;               węzła odbiorczego. Zwraca ustawioną flagę przeniesienia jeśli operacja zakończyła się
;               powodzeniem, wyczyszczoną jeśli niepowodzeniem

GetFileInfo     proc    near

; Pobranie nazwy pliku z lini poleceń DOS:

                mov     ax, 1
                argv
                mov     word  ptr FileNamePtr, di
                mov     word ptr FileNamePtr+2, es

                printf
                byte    “Opening %^s\n”,0
                dword   FileNamePtr

;Otwieramy plik:

                push    ds
                mov     ax, 3D00h               ;otwarcie do odczytu
                lds     dx, FileNamePtr
                int     21h
                pop     ds.
                jc      BadFile
                mov     FileHandle, ax

;obliczamy rozmiar tego pliku (robimy to poprzez pozycjonowanie na ostatniej pozycji w pliku i
; użycie pozycji powrotnej jako długości pliku):

                mov     bx, ax                  ;potrzebna obsługa w BX
                mov     ax, 4202h               ;pozycja końca pliku
                xor     cx, cx                  ;umieszczenie na pozycji zero
                xor     dx, dx                  ; z końca pliku
                int     21h
                jc      BadFile

; Zachowujemy końcową pozycję jako długość pliku:

                mov     word  ptr FileSize, ax
```

```
                mov     word  ptr FileSize+2, dx

;Musimy przewinąć plik na początek (ustawienie pozycji zero):

                mov     bx, FileHandle
                mov     ax, 4202h               ;pozycja początku pliku
                xor     cx, cx                  ;ustawienie pozycji zero
                xor     dx, dx
                int     21h
                jc      BadFile

;Okay, przekazujemy do węzła odbiorczego:

                mov     al., byte ptr FileSize                  ;wysłanie rozmiaru pliku
                call    SendByte
                mov     al., byte  ptr FileSie+1
                call    SendByte
                mov     al,  byte  ptr FileSize+2
                call    SendByte
                mov     al, byte  ptr FileSize+3
                call    SendByte

                les     bx, FileNamePtr                 ;wysyłamy znaki nazwy pliku do
odbiorcy
SendName:       mov     al., es;[bx]                    ;dopóki nie trafimy na bajt zero
                call    SendByte
                inc     bx
                cmp     al., 0
                jne     SendName
                stc                                     ;zwraca powodzenie
                ret

BadFile:        print
                byte    „Error transmitting file infroamtion:”, 0
                puti
                putcr
                clc
                ret
GetFileInfo     endp

;GetFileData-   procedura ta odczytuje dane z pliku I transmituje je do odbiorcy jako bajt w czasie

GetFileData     proc    near
                mov     ah, 3Fh                 ;opcod odczytu DOS
                mov     cx, 512                 ;odczyt 512 bajtów w czasie
                mov     bx, FileHandle          ;plik do odczytu
                lea     dx, FileBuffer          ;bufor do przechowywania danych
                int     21h                     ;odczyt danej
                jc      GFDError                ;wyjście jeśli błąd odczytu danych

                mov     cx, ax                  ; zachowanie # bajta odczytanego
                jcxz    GFDone                  ;wyjście jeśli EOF
                lea     bx, FileBuffer          ;wysłanie bajtów bufora plików do odbiorcy
XmitLoop:       mov     al., [bx]
                call    SendByte
                inc     bx
                loop    XmitLoop
                jmp     GetFileData             ;odczyt reszty pliku
```

```
GFDError:       print
                byte    „DOS error #”, 0
                puti
                print
                byte    , while reading file“, cr, lf, 0
GFDone:         ret
GetFileData     endp

;Okay, tu mamy program główny, któ®y steruje wszystkim

Main            proc
                mov     ax, dseg
                mov     ds, ax
                meminit

;Najpierw pobieramy adres LPT1: z obszaru zmiennych BIOS

                mov     ax, 40h
                mov     es, ax
                mov     ax, es:[PrtrBase]
                mov     MyPortAdrs, ax

; Zobaczmy czy mamy parametr nazwy pliku:

                argc
                cmp     cx, 1
                je      GotName
                print
                byte    „Usage: transmit <filename>”, cr,lf,0
                jmp     Quit

GotName:        call    Synchronize                     ;czekanie na program przekaźnika
                jnc     Quit

                call    GetFileData                     ;pobieranie danych h pliku
Quit:           ExitPgm
Main            endp

cseg            ends

ssego           segment para  stack ‘stack’
stk             byte    1024   dup  (“stack”)
sseg            ends

zzzzzzseg       segment para public ‘zzzzzz’
LastBytes       byte    16  dup  (?)
Zzzzzzseg       ends
                end     Main
```

Tu mamy program odbiornika, który akceptuje i przechowuje dany wysyłane przez powyższy program :

```
; RECEIVE.ASM
;
;Program ten jest odbiorczą częścią programu, który przekazuje pliki przez  kompatybilny kabel równoległy z
; Laplink
;
; Program ten zakłada, że użytkownik chce używać LPT1: dla transmisji. Modyfikuje przyrównywanie lub czyta
; port z lini poleceń jeśli jest niewłaściwy
```

```
                .286
                .xlist
                include         stdlib.a
                includelib      stdlib.lib
                .list

dseg            segment para public 'data'

TimeOutConst    equ     100
Prtrbase        equ     8                       ;offset adresu LPT1:

MyPortAdrs      word    ?                       ;przechowuje adres portu drukarki
FileHandle      word    ?                       ;obsługa pliku wyjściowego
FileBuffer      byte    512  dup (?)            ;bufor dla nadchodzących danych

FileSize        dword   ?                       ;rozmiar nadchodzącego pliku
FileName        byte    128  dup  (0)           ; przechowuje nazwę pliku

dseg            ends

cseg            segment para public 'code'
                assume  cs:cseg, ds:dseg

;TestAbort-     Odczytuje klawiaturę i  podaje użytkownikowi sposobność do włączenia klawiszy ctrl-C

TestAbort       proc    near
                push    ax
                mov     ah, 1
                int     16h                     ;zobacz czy coś naciśnięto
                je      NoKeypress
                mov     ah, 8                   ;odczyt znaku, sprawdzenie na ctrl-C
                int     21h
NoKeyPress:     pop     ax
                ret
TestAbort       endp

; GetByte-      odczytuje pojedynczy bajt z portu równoległego (cztery bity w czasie). Zwraca bajt w AL.

GetByte         proc    near
                push    cx
                push    dx

; odbiór mniej znaczącego nibble'a

                mov     dx, MyPortAdrs
                mov     al., 10h                ;sygnał nie zajęty
                out     dx, al.

                inc     dx                      ;wskazuje port stanu

W4DLp:          mov     cx, 10000
Wait4Data:      in      al., dx                 ;zobacz  czy dana dostępna
                test    al., 80h                ; (bit 7 = 0 jeśli dana dostępna)
                loopne  Wait4Data
                je      DataIsAvail             ;czy dana jest dostępna?
                call    TestAbort               ;jeśli nie sprawdzamy ctrl-C
                jmp     W4DLp
```

```
DataIsAvail:    shr     al., 3                  ;zachowujemy pakiet czterech bitów
                and     al., 0Fh                ; (to jest mniej znaczący nibble naszego bajtu)
                mov     ah, al.

                dec     dx                      ;wskazuje rejestr danych
                mov     al., 0
                out     dx, al.

                inc     dx                      ;wskazuje rejestr stanu
W4Alp:          mov     cx, 10000
Wait4Ack:       in      al., dx                 ;czeka aż przekaźnik  cofnie dostępną daną
                test    al., 80h
                loope   Wait4Ack                ;pętla dopóki pętla niedostępna
                jne     NextNibble              ;skok jeśli dana niedostępna
                call    TestAbort               ;zezwala użytkownikowi na ctrl-C
                jmp     W4Alp
; odbiór bardziej znaczącego nibble'a:

NextNibble:     dec     dx                      ;wskazuje rejestr danych
                mov     al., 10h                ; sygnał nie zajęty
                out     dx, al.
                inc     dx                      ;wskazuje stan portu
W4D2Lp:         mov     cx, 10000
Wait4Data2:     in      al., dx                 ;zobaczmy czy dana jest dostępna
                test    al., 80h                ;(bit 7 = 0 jeśli dostępna)
                loopne  Wait4Data2              ;pętla dopóki data jest dostępna
                je      DataAvail2              ;skok jeśli dostępna
                call    TestAbort               ;sprawdzamy ctrl-C
                jmp     W4D2Lp

DataAvail2:     shl     al., 1                  ;dzielimy ten bardziej znaczący nibble z
                and     al., 0FH                ; istniejącym  mniej znaczącym nibble'm
                or      ah, al.
                dec     dx                      ;wskazuje rejestr danych
                mov     al., 0                  ;sygnał danej pobrany
                out     dx, al.

                inc     dx                      ;wskazuje rejestr stanu
W4A2Lp:         mov     cx, 10000
Wait4Ack2:      in      al, dx                  ;czeka na przekaźnik aż cofnie dostępną daną
                test    al., 80h
                loope   Wait4Ack2               ;czekamy aż dana niedostępna
                jne     ReturnData              ;skok jeśli nie dostępna
                call    TestAbort               ;sprawdzenie ctrl-C
                jmp     WrA2Lp

ReturnData:     mov     al., ah                 ;włożenie danej do al
                pop     dx
                pop     cx
                ret
GetByte         endp

; Synchronize-  Procedura ta oczekuje dopóki widzie same zera w bitach wejściowych jakie odebraliśmy z
;               węzła transmisji. Ponieważ odebrano same zera, zapisuje same jedynki do portu wyjściowego .
;               Kiedy mamy same jedynki, zapisujemy same zera. Powtarzamy ten proces dopóki węzeł
;               transmisji zapisze wartość 05h

Synchronize     proc    near
```

```
                print
                byte    „Synchronizing with transmitter program”
                byte    cr, lf, 0

                mov     dx, MyPortAdrs
                mov     al, 0                   ;inicjalizujemy nasz port wyjściowy
                out     dx, al.                 ; zapobiegając pomyłce
                mov     bx, TimeOutConst        ;warunek przekroczenia czasu

SyncLoop:       mov     cx, 0                   ;dla celów przekroczenia czasu
SyncLoop0:      inc     dx                      ;wskazuje port wejściowy
                in      al., dx                 ;odczyt naszych bitów wejściowych
                dec     dx
                and     al., 78h                ;trzymamy tylko bity danej
                cmp     al., 78h                ;sprawdzamy dla wszystkich jedynek
                je      Got1s                   ;skok jeśli same jedynki
                cmp     al. 0                   ;zobacz czy same zera
                loopne  SyncLoop0

;Ponieważ widzieliśmy zero, zapisujemy same jedynki do portu wyjściowego

                mov     al., 0FFh               ;zapisz wszystkie jedynki
                out     dx, al.

;teraz czekamy  aż same jedynki nadejdą z węzła transmisji

SyncLoop1:      inc     dx              ;Wskazuje rejestr stanu
                in      al., dx         ;odczyt stanu portu
                dec     dx              ;wskazuje z powrotem rejestr danych
                and     al., 78h        ;trzymamy tylko bity danej
                cmp     al., 78h        ;czy wszystkie są jedynkami?
                loopne  SyncLoop1
                je      Got1s

;jeśli przekroczyliśmy czas,  sprawdzamy aby zobaczyć czy użytkownik nacisnął ctrl-C aby przerwać

                call    TestAbort               ;sprawdzamy ctrl-C
                dec     dx                      ;zobaczmy czy przekroczyliśmy czas
                jne     SyncLoop                ;powtarzamy jeśli tak

                print
                byte    „Receive:  connected time out during synchronization”
                byte    cr, lf, 0
                clc                             ;sygnał przekroczenia czasu
                ret

; Skaczemy tu jeśli widzieliśmy i zero i jeden. Wysyłamy je dwa w kombinacji dopóki nie dostaniemy 05h z
; węzła transmisji lub użytkownik nie naciśnie ctrl-C

Got1s:          inc     dx              ;wskazuje rejestr stanu
                in      al., dx         ;kopiujemy cokolwiek pojawi się w naszym porcie
                dec     dx              ;wejściowym do portu wyjściowego dopóki węzeł
                shr     al. 3           ;transmisji nie wyśle nam wartości 05h
                and     al., 0Fh
                cmp     al., 05h
                je      Synchronized
                not     al.             ;trzymamy odwrócone to co dostaliśmy i wysyłamy
                out     dx, al.         ; to do tarnsmittera
                call    TestAbort       ;sparwdzamy ctrl-C
```

```
                jmp     Got1s

;Okay, zsynchronizowaliśmy, wracamy do kodu wywołującego

Synchronized:
                and     al., 0Fh                ;upewniamy się czy bit busy to jeden
                out     dx, al.                 ; (bit 4 = 0 dla busy = 1)
                print
                byte    "Synchronized with transmitting site"
                byte    cr, lf, 0
                stc
                ret
Synchronize     endp

;GetFileInfo-   Program przekaźnika wysyła nam długość pliku i nazwę pliku zakończoną zerem
GetFileInfo     proc    near
                mov     dx, MyPortAdrs
                mov     al, 10h                 ;ustawiamy bit busy na zero
                out     dx, al.

;Pierwsze cztery bajty zwierają rozmiar pliku:

                call    GetByte
                mov     byte  ptr FileSize, al
                call    GetByte
                mov     byte  ptr FileSize+1, al
                call    GetByte
                mov     byte  ptr FileSize+2, al
                call    GetByte
                mov     byte ptr FileSize+3, al

; kolejne n bajtów (do bajtu zakończonego zerem) zawierają nazwę pliku

                mov     bx, 0
GetFileName:    call    GetByte
                mov     FileName[bx], al
                call    TestAbort
                inc     bx
                cmp     al, 0
                jne     GetFileName

                ret
GetFileInfo     endp

;GetFileData-   Odbiera plik danych z węzła transmisji I zapisuje go do pliku wyjściowego

GetFileData     proc    near

;najpierw, zobaczymy czy mamy więcej niż 512 bajtów

                cmp     word  ptr FileSize+2, 0     ;jeśli bardziej znaczące słowo nie jest
                jne     MoreThan512                 ;zerem, więcej niż 512.
                cmp     word  ptr FileSize, 512     ;jeśli bardziej znaczące słowo jest zerem,
                jbe     LastBlock                   ;sprawdzamy mnij znaczące słowo

; mamy więcej niż 512 bajtów w tym pliku, odczytujemy 512 bajtów w tum punkcie

MoreThan512:    mov     cx, 512                     ;odbieramy 512 bajtów
                lea     bx, FileBuffer              ;z przekaźnika
```

```
ReadLoop:   call    GetByte                 ;odczyt bajtu
            mov     [bx], al                ;zachowujemy bajt
            inc     bx                      ;przechodzimy do kolejnego elementu
            loop    ReadLoop                ;bufora

;Okay, zapisujemy dane do pliku:

            mov     ah, 40h                 ;opcod zapisu DOS
            mov     bx, FileHandle          ;zapis do tego pliku
            mov     cx, 512                 ;zapis 512 bajtów
            lea     dx, FileBuffer          ; z tego adresu
            int     21h
            jc      BadWrite                ;wyjście jeśli błąd

;zmniejszenie rozmiaru pliku do 512 bajtów:

            sub     word ptr FileSize, 512  ;32 bitowe odejmowanie z 512
            sbb     word  ptr FileSize, 0
            jmp     GetFileData

;Przetwarzamy ostatni blok, który zawiera 1..511 bajtów

LastBlock:
            mov     cx, word  ptr  FileSize ;odbiór ostatnich 1..511 bajtów z
            lea     bx, FileBuffer          ;przekaźnika
ReadLB:     call    GetByte
            mov     [bx], al
            inc     bx
            loop    ReadLB
            mov     ah, 40h                 ;zapis ostatniego bloku do pliku
            mov     bx, FileHandle
            mov     cx, word ptr FileSize
            lea     dx, FileBuffer
            int     21h
            jnc     Closefile

BadWrite:   print
            byte    "DOD error #", 0
            put1
            print
            byte    " while writing data.", cr, lf,0

;Tu zamykamy plik

CloseFile:  mov     bx, FileHandle          ;zamykamy ten plik
            mov     ah, 3Eh                 ;opcod zamykania DOS
            int     21h
            ret
GetFileData endp

; Tu jest program główny

Main        proc
            mov     ax, dseg
            mov     ds, ax
            meminit

;najpierw pobieramy adres LPT1: z obszaru zmiennych BIOS
```

```
                mov     ax, 40h                 ;wskazuje segment zmiennych BIOS
                mov     es, ax
                mov     ax, es:[PrtrBase]
                mov     MyPortAdrs, ax

                call    Synchronize             ;oczekujemy na program przekaźnika
                jnc     Quit

                call    GetFileInfo             ;pobieramy nazwę i rozmiar pliku

                printf
                byte    „Filename: %s\nFile size: %ld\n”, 0
                dword   FileName, FileSize

                mov     ah, 3Ch                 ;stworzenie pliku
                mov     cx, 0                   ;atrybuty standardowe
                lea     dx, Filename
                int     21h
                jnc     GoodOpen
                print
                byte    „Error opening file”, cr,lf,0
                jmp     Quit

GoodOpen:       mov     FileHandle, ax
                call    GetFileData             ;pobranie danych pliku
Quit:           ExitPgm
Main            endp
cseg            ends

sseg            segment para stack ‘stack’
stk             byte    1024  dup (“stack”)
sseg            ends

zzzzzzseg       segment para public ‘zzzzzz’
LastBytes       byte    16  dup  (?)
zzzzzzseg       ends
                end     Main
```

21.5 PODSUMOWANIE

Port równoległy PC , chociaż pierwotnie zaprojektowany do sterownia drukarkami równoległymi , jest uniwersalnym ośmio bitowym portem wyjściowym z kilkoma liniami uzgodnień, jakie możemy zastosować do sterowania różnymi innymi urządzeniami niż drukarki.

Teoretycznie, komunikacja równoległa powinna być wiele razy szybsza niż komunikacja szeregowa. W praktyce, jednak ograniczenia świata rzeczywistego i ekonomia uniemożliwiają taki przypadek. Niemniej jedna możemy połączyć wysoko wydajne urządzenia do portu równoległego PC.

Porty równoległe PC dzielą się na jednokierunkowe i dwukierunkowe. Wersje dwukierunkowe są dostępne tylko na PS/2, pewnych laptopach i kilku innych maszynach. Podczas gdy osiem lini danych jest wyjściowych tylko w portach jednokierunkowych, możemy oprogramować je jako wejściowe lub wyjściowe w porcie dwukierunkowym. Chociaż operacje dwukierunkowe są skierowane do drukarki, można poprawić wydajność innych urządzeń połączonych z portem równoległym, takie jak dysk i streamer, łącza sieciowe, SCSI i tak dalej.

Kiedy system komunikuje si z innym urządzeniem przez port równoległy, potrzebuje jakiegoś sposobu przekazania urządzeniu, że dana jest dostępna na lini danych. Podobnie, urządzenie potrzebuje sposobu w jaki przekaże systemowi, że nie jest zajęte i zaakceptowało dane. Wymaga to dodatkowych sygnałów w porcie równoległym znanych jako linie uzgodnienia. Typowy port równoległy PC dostarcza trzech sygnałów uzgodnienia: strobowanie dostępnej danej, sygnał potwierdzenia pobrania danej i linia zajętości urządzenia. Linie te łatwo sterują przepływem danych pomiędzy PC a jakimś urządzeniem zewnętrznym.

Dodatkowo do lini uzgodnień port równoległy PC dostarcza kilku innych pomocniczych linii I/O. W sumie jest 12 lini wyjściowych i pięć wejściowych w porcie równoległym PC Są trzy porty I/O w przestrzeni adresowej PC powiązane z każdym portem I/O. Pierwszy z nich (pod adresem bazowym portu) jest rejestrem danych. Jest to ośmio bitowy rejestr wyjściowy w porcie jednokierunkowym, jest rejestrem wejściowym / wyjściowym w porcie dwukierunkowym. Drugi rejestr , pod adresem bazowym plus jeden jest rejestrem stanu. Rejestr stanu jest portem wejściowym. Pięć z tych bitów odpowiada pięciu liniom wejściowym w porcie równoległym PC. Rejestr trzeci (pod adresem bazowym plus dwa) jest rejestrem sterującym. Cztery z tych bitów odpowiadają dodatkowym czterem bitom wyjściowym w PC, jeden z tych bitów steruje linią IRQ w porcie równoległym, a szósty bit steruje kierunkiem danych w porcie dwukierunkowym.

*"Podstawowe informacje o porcie równoległym"
*"Sprzęt portu równoległego"

Chociaż wielu producentów używa portu równoległego do sterowania wieloma różnymi urządzeniami, drukarka równoległa jest urządzeniem najczęściej przyłączanym do portu równoległego. Są trzy sposoby do wysyłania danych do drukarki: poprzez wywołanie DOS do wydruku znaku, poprzez wywołanie ISR'a int 17h BIOS do wydruku znaku, lub poprzez komunikację bezpośrednią do portu równoległego. Powinniśmy unikać tej ostatniej techniki z powodu możliwych niekompatybilności programowych z innymi urządzeniami podłączonymi do portu równoległego.

*"Sterowanie drukarką poprzez port równoległy"
*"Drukowanie poprzez DOS"
*"Drukowanie poprzez BIOS"
*"Podprogram obsługi przerwania INT 17h"

Popularnym zastosowaniem portu równoległego jest komunikacja pomiędzy dwoma komputerami; na przykład transfer danych między maszyną stacjonarną a laptopem.. Dla zademonstrowania jak używać portu równoległego do sterowani innym urządzeniem poza drukarką, rozdział ten przedstawia program do transferu danych pomiędzy komputerami poprzez jednokierunkowy port równoległy (działa również z portem dwukierunkowym)

*"Miedzy komputerowa komunikacja portem równoległym"