```
version equ     3

        include defs.asm

; PC/FTP Packet Driver source, conforming to version 1.09 of the spec
; Katie Stevens (dkstevens@ucdavis.edu)
; Computing Services, University of California, Davis
; Portions (C) Copyright 1988 Regents of the University of California

; This program is free software; you can redistribute it and/or modify
; it under the terms of the GNU General Public License as published by
; the Free Software Foundation, version 1.
;
; This program is distributed in the hope that it will be useful,
; but WITHOUT ANY WARRANTY; without even the implied warranty of
; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
; GNU General Public License for more details.

; Change History
;
; 05-90         ks      Katie Stevens (dkstevens@ucdavis.edu):
;                       - Created
; 07-19-91      jgn     Jim Noble (noble_jim@po.gis.prc.com):
;                       - Added Change History and documented changes
;                       - Fixed a bug that caused AppleTalk network numbers to
;                         be byte-swapped when printed
;                       - Fixed a bug that caused the wrong buffer address to
;                         be passed to the packet receiver upcall routine
;                       - Made changes in the driver information structure to
;                         match LocalTalk:  size of MAC-layer address, MTU,
;                         and multicast buffer size
;                       - Changed the packet buffer sizes from 1024 to 586,
;                         the maximum amount of data in a DDP packet
;                       - Commented-out the noop_upcall completion routine and
;                         changed the completion routine address passed to
;                         the AppleTalk driver to 0000:0000 for all driver
;                         calls
;                       - Set the status field to -1 prior to the GetNetInfo
;                         AppleTalk driver call as per the recommendation of
;                         the driver interface spec.
;                       - Added structure definitions for short and long
;                         format DDP packets and used them when adding a
;                         check to ensure that received packets are IP packets
; 07-22-91      jgn     Jim Noble (noble_jim@po.gis.prc.com):
;                       - Fixed a bug that would prevent the packet driver
;                         from initializing with a static address if there
;                         were no dynamic addresses available
; 07-23-91      jgn     Jim Noble (noble_jim@po.gis.prc.com):
;                       - Added the optional IP address to the usage message
;                       - Incremented version to 3
;

code    segment word public
        assume  cs:code, ds:code

; Definitions specific to the ATALK.SYS driver for PC LocalTalk cards:
; these include Apple LocalTalk PC Card, Sun/TOPS FlashCard
; For a complete description of the LocalTalk commands, structures and
; methods used in this driver, please refer to Apple APDA document #M7055,
; LocalTalk PC Card and Driver Preliminary Notes.

driverstring    db      'AppleTalk', 0          ; ATALK.SYS signature string
dot_char        db      '.', 0                  ; for IP address display

AT_INT          equ     5CH                     ; Software int# for ATALK.SYS

; General ATALK.SYS driver commands
AT_INIT                 equ     01H             ; Initialize driver software
AT_GETNETINFO           equ     03H             ; Get driver info

; Datagram Delivery Protocol commands for ATALK.SYS driver
DDP_OPENSOCKET          equ     20H
DDP_CLOSESOCKET         equ     21H
DDP_WRITE               equ     22H
DDP_READ                equ     23H
DDP_CANCEL              equ     24H

; Name Binding Protocol commands for ATALK.SYS driver
NBP_REGISTER            equ     30H
NBP_REMOVE              equ     31H
NBP_LOOKUP              equ     32H
NBP_CONFIRM             equ     33H
NBP_CANCEL              equ     34H

; AppleTalk Transaction Protocol commands for ATALK.SYS driver
ATP_SEND_REQUEST        equ     42H

; ATALK.SYS command qualifiers
ASYNC_MASK      equ     8000H           ; Start command, then return
INTR_MASK       equ     4000H           ; Wait for intr service to complete

XO_BIT          equ     20H             ; ATP - exactly once transaction

; Structure for short format DDP packet [jgn]
DDPshort struc
        lap_dst         db      ?       ; destination node
        lap_src         db      ?       ; source node
        lap_type        db      ?       ; 1 for short format DDP packet
        ddps_len        dw      ?       ; packet length
        ddps_dskt       db      ?       ; destination socket
        ddps_sskt       db      ?       ; source socket
        ddps_type       db      ?       ; 22 for IP packets
DDPshort ends

; Structure for long format DDP packet [jgn]
DDPlong struc
        lap_dst         db      ?       ; destination node
        lap_src         db      ?       ; source node
        lap_type        db      ?       ; 2 for long format DDP packets
        ddpl_len        dw      ?       ; packet length
        ddpl_chksum     dw      ?       ; checksum
        ddpl_dnet       dw      ?       ; destination network
        ddpl_snet       dw      ?       ; source network
        ddpl_dnode      db      ?       ; destination node
        ddpl_snode      db      ?       ; source node
        ddpl_dskt       db      ?       ; destination socket
        ddpl_sskt       db      ?       ; source socket
        ddpl_type       db      ?       ; 22 for IP packets
DDPlong ends

; Structure for AppleTalk node addressing
AddrBlk struc
        ablk_network    dw      0
        ablk_nodeid     db      0
        ablk_socket     db      0
AddrBlk ends

; Structure for general calls to AppleTalk driver (ATALK.SYS)
InfoParams struc
```

```
        atd_command     dw              AT_GETNETINFO
        atd_status      dw              0
        atd_compfun     segmoffs        <>
        inf_network     dw              0
        inf_nodeid      db              0
        inf_abridge     db              0
        inf_config      dw              0
        inf_buffptr     segmoffs        <>
        inf_buffsize    dw              0
InfoParams ends
; Parameter block for general calls to AppleTalk driver (ATALK.SYS)
MyInfo  InfoParams      <>

; Address block for our gateway
MyGateway       AddrBlk         <>

; Structure for calls to AppleTalk driver (ATALK.SYS) for Datagram
; Delivery Protocol (DDP) service
DDPParams struc
        ddp_command     dw              0
        ddp_status      dw              0
        ddp_compfun     segmoffs        <>
        ddp_addr        AddrBlk         <>
        ddp_socket      db              0
        ddp_type        db              0
        ddp_buffptr     segmoffs        <>
        ddp_buffsize    dw              0
        ddp_chksum      db              0
DDPParams ends
; Parameter blocks for AppleTalk DDP access
DDPio           DDPParams       <>              ; Write on DDP socket
; 2 buffers for packet receive from ATALK.SYS
DDP1inuse       db      0                       ; Buffer occupied flag
DDP1buffsize    dw      0                       ; Buffer length during reads
DDP1buffer      db      586 dup (0)             ; Buffer for DDP read [jgn]
DDP2inuse       db      0                       ; 2nd Buffer occupied flag
DDP2buffsize    dw      0                       ; 2nd Buffer length during read
s
DDP2buffer      db      586 dup (0)             ; 2nd Buffer for DDP read [jgn]

; Structure for calls to AppleTalk driver (ATALK.SYS) for Name
; Binding Protocol (NBP) service
NBPParams struc
        nbp_command     dw              0
        nbp_status      dw              0
        nbp_compfun     segmoffs        <>
        nbp_addr        AddrBlk         <>
        nbp_toget       dw              0
        nbp_buffptr     segmoffs        <>
        nbp_buffsize    dw              0
        nbp_interval    db              0
        nbp_retry       db              0
        nbp_entptr      segmoffs        <>
NBPParams ends
; Parameter block for AppleTalk NBP access
NBP             NBPParams       <>

; Structure for name-to-address bind entries
NBPTuple struc
        tup_address     AddrBlk         <>
        tup_enum        db              0
        tup_name        db              99 dup(0)
NBPTuple ends
; Name Binding Tuple for our IP gateway
NBPt            NBPTuple        <>

; Structure for name-to-address table
NBPEntry struc
        tab_next        segmoffs        <>
        tab_entry       NBPTuple        <>
NBPEntry ends
NBPtable        NBPEntry        <>

; Structure for calls to AppleTalk driver (ATALK.SYS) for AppleTalk
; Transaction Protocol (ATP) service
ATPParams struc
        atp_command     dw              0
        atp_status      dw              0
        atp_compfun     segmoffs        <>
        atp_addrblk     AddrBlk         <>
        atp_socket      db              0
        atp_fill        db              0
        atp_buffptr     segmoffs        <>
        atp_buffsize    dw              0
        atp_interval    db              0
        atp_retry       db              0
        atp_flags       db              0
        atp_seqbit      db              0
        atp_tranid      dw              0
        atp_userbytes   db              4 dup(0)
        atp_bdsbuffs    db              0
        atp_bdsresps    db              0
        atp_bdsptr      segmoffs        <>
ATPParams ends
; Parameter block for AppleTalk ATP access
ATP             ATPParams       <>

; Structure for BDS elements
BDSElement struc
        bds_buffptr     segmoffs        <>
        bds_buffsize    dw              0
        bds_datasize    dw              0
        bds_userbytes   db              4 dup(0)
BDSElement ends
; Parameter block for our BDS element
BDS             BDSElement      <>

; Struct for IP gateway information
IPGInfo struc
        ipg_opcode      db              0,0,0,1 ; IPGP_ASSIGN
        ipg_ipaddress   dd              0       ; our IP address
        ipg_ipname      dd              0       ; nameserver IP address
        ipg_ipbroad     dd              0       ; broadcast IP address
        ipg_ipfile      dd              0       ; file server IP address
        ipg_ipother     dd              4 dup (0)
        ipg_string      db              128 dup (0), '$'
IPGInfo ends
; Parameter block for info about our IP gateway
IPG             IPGInfo         <>

IPG_ERROR       equ     -1

static_address  db      0, 0, 0, 0
use_static      db      0
test_address    db      0, 0, 0, 0
temp_4bytes     db      0, 0, 0, 0

; End of Appletalk parameter definitions

; The following values may be overridden from the command line.
```

```
; If they are omitted from the command line, these defaults are used.

        public  int_no
int_no  db      0,0,0,0                 ;must be four bytes long for get_number
.

        public  driver_class, driver_type, driver_name, driver_function, parame
ter_list
driver_class    db      5,0             ;from the packet spec
driver_type     db      1               ;from the packet spec
driver_name     db      'LocalTalk',0   ;name of the driver.
driver_function db      2
parameter_list  label   byte
        db      1       ;major rev of packet driver
        db      9       ;minor rev of packet driver
        db      14      ;length of parameter list
        db      4       ;length of MAC-layer address [jgn]
        dw      586     ;MTU, including MAC headers [jgn]
        dw      0       ;buffer size of multicast addrs [jgn]
        dw      0       ;(# of back-to-back MTU rcvs) - 1
        dw      0       ;(# of successive xmits) - 1
int_num dw      0       ;Interrupt # to hook for post-EOI
                        ;processing, 0 == none,

        public  rcv_modes
rcv_modes       dw      4               ;number of receive modes in our table.
                dw      0,0,0,rcv_mode_3

        public bad_command_intercept
bad_command_intercept:
;called with ah=command, unknown to the skeleton.
;exit with nc if okay, cy, dh=error if not.
        mov     dh,BAD_COMMAND
        stc
        ret

        public  as_send_pkt
; The Asynchronous Transmit Packet routine.
; Enter with es:di -> i/o control block, ds:si -> packet, cx = packet length,
;   interrupts possibly enabled.
; Exit with nc if ok, or else cy if error, dh set to error number.
;   es:di and interrupt enable flag preserved on exit.
as_send_pkt:
        ret

        public  drop_pkt
; Drop a packet from the queue.
; Enter with es:di -> iocb.
drop_pkt:
        assume  ds:nothing
        ret

        public  xmit
; Process a transmit interrupt with the least possible latency to achieve
;   back-to-back packet transmissions.
; May only use ax and dx.
xmit:
        assume  ds:nothing
        ret

        public  send_pkt
send_pkt:
;enter with es:di->upcall routine, (0:0) if no upcall is desired.
;  (only if the high-performance bit is set in driver_function)
;enter with ds:si -> packet, cx = packet length.
;if we're a high-performance driver, es:di -> upcall.
;exit with nc if ok, or else cy if error, dh set to error number.
        assume  ds:nothing
; send packet to AppleTalk/DDP/IP gateway
        ; load info about the packet we are sending
        mov     DDPio.ddp_buffptr.offs, si
        mov     DDPio.ddp_buffptr.segm, ds      ; DDPio.buffptr -> IP packet
        mov     DDPio.ddp_buffsize, cx          ; DDPio.buffsize = packet len

        ; send all packets to the IP gateway
        mov     cx, (size AddrBlk)              ; DDPio.ddp_addr = MyGateway
        mov     ax, cs
        mov     ds, ax
        mov     es, ax
        mov     si, offset MyGateway
        mov     di, offset DDPio.ddp_addr
        rep     movsb

        mov     bx, offset DDPio
        call    doATint                         ; Ask ATALK.SYS to send packet

        cmp     DDPio.ddp_status, 00H           ; Packet sent okay?
        je      send_ret                        ; Yes, status is good
                                                ; No, status gives error
send_err:
        call    count_out_err
        mov     dh, CANT_SEND                   ; set error flag
        stc
        ret

send_ret:
        clc                                     ; packet sent successfully
        ret

        public  set_address
set_address:
;enter with ds:si -> Ethernet address, CX = length of address.
;exit with nc if okay, or cy, dh=error if any errors.
        assume  ds:nothing
        mov     dh, BAD_COMMAND
        stc
        ret


rcv_mode_3:
;receive mode 3 is the only one we support, so we don't have to do anything.
        ret


        public  set_multicast_list
set_multicast_list:
;enter with ds:si ->list of multicast addresses, ax = number of addresses,
;  cx = number of bytes.
;return nc if we set all of them, or cy,dh=error if we didn't.
        mov     dh,NO_MULTICAST
        stc
        ret


        public  terminate
terminate:
        push    ds
        push    cs
        pop     ds
```

```
terminate_write:
        ; close the DDP socket
        mov     DDPio.ddp_command, DDP_CLOSESOCKET
        mov     bx, offset DDPio
        call    doATint

        mov     NBP.nbp_command, NBP_REMOVE
        mov     NBP.nbp_entptr.offs, offset NBPtable.tab_entry.tup_name
        mov     NBP.nbp_entptr.segm, ds
        mov     bx, offset NBP
        call    doATint

        pop     ds
        ret

        public  reset_interface
reset_interface:
;reset the interface.
        assume  ds:code
        ret


;called when we want to determine what to do with a received packet.
;enter with cx = packet length, es:di -> packet type, dl = packet class.
        extrn   recv_find: near

;called after we have copied the packet into the buffer.
;enter with ds:si ->the packet, cx = length of the packet.
        extrn   recv_copy: near

;call this routine to schedule a subroutine that gets run after the
;recv_isr.  This is done by stuffing routine's address in place
;of the recv_isr iret's address.  This routine should push the flags when it
;is entered, and should jump to recv_exiting_exit to leave.
;enter with ax = address of routine to run.
        extrn   schedule_exiting: near

;recv_exiting jumps here to exit, after pushing the flags.
        extrn   recv_exiting_exit: near

        extrn   count_in_err: near
        extrn   count_out_err: near

        public  recv
recv:
;called from the recv isr.  All registers have been saved, and ds=cs.
;Upon exit, the interrupt will be acknowledged.
;NOTE: this packet driver merely makes calls to another hardware
;driver, ATALK.SYS. ATALK.SYS handles the hardware interrupt service;
;ATALK.SYS then calls this packet driver with FAR subroutine calls.
;the ATALK.SYS FAR subroutine is recv_at_upcall
        assume  ds:nothing
        ret

;*******************************************

; [jgn] Null completion routine is not needed.  Just specify 0000:0000 as the
; address of the completion routine for no call.
;
; NULL completion routine for ATALK.SYS drivers calls
;noop_upcall proc far
;       ret
;noop_upcall endp

;First half routine for DDP socket.
;ATALK.SYS calls this routine when a packet is received.
;ATALK.SYS assumes we are a far procedure.
;       AH = socket
;       CX = size of data packet
;       DS:BX = address of buffer
preview_upcall proc far
        assume  ds:nothing

;       maximum packet we can receive is 586 bytes [jgn]
        cmp     cx, 586                         ; Max size of DDP data field [j
gn]
        ja      preview_drop

;       Is it an IP packet?  DDP type should be 22 [jgn]
        cmp     byte ptr ds:[bx].lap_type, 1    ; Is it a short DDP packet?
        jne     ddp_long                        ; No, must be long DDP
        cmp     byte ptr ds:[bx].ddps_type, 22  ; Is it an IP packet?
        jne     preview_drop                    ; No, refuse packet
        jmp     preview_buff1
ddp_long:
        cmp     byte ptr ds:[bx].ddpl_type, 22  ; Is it an IP packet?
        jne     preview_drop                    ; No, refuse packet

preview_buff1:
        cmp     DDP1inuse, 00H
        jne     preview_buff2
        mov     DDP1inuse, 01H

;       repeat buffer size back to ATALK.SYS in CX
;       ask ATALK.SYS driver to pass us the buffer at DS:BX
;       tell ATALK.SYS address of 2nd half routine in ES:DX
        movseg  ds,cs
        mov     bx, offset DDP1buffer           ; ds:bx->buffer
        movseg  es,cs
        mov     dx, offset recv_at_upcall       ; es:dx->2nd half routine
        jmp     preview_ret

preview_buff2:
        cmp     DDP2inuse, 00H
        jne     preview_drop
        mov     DDP2inuse, 01H

;       repeat buffer size back to ATALK.SYS in CX
;       ask ATALK.SYS driver to pass us the buffer at DS:BX
;       tell ATALK.SYS address of 2nd half routine in ES:DX
        movseg  ds,cs
        mov     bx, offset DDP2buffer           ; ds:bx->buffer
        movseg  es,cs
        mov     dx, offset recv_at_upcall       ; es:dx->2nd half routine
        jmp     preview_ret


preview_drop:
;       ask ATALK.SYS to drop the packet
        call    count_in_err
        mov     cx, 00h

preview_ret:
        ret
preview_upcall endp

;Second half routine for DDP socket.
;ATALK.SYS calls this routine when the packet has been copied to our buffer.
;ATALK.SYS assumes we are a far procedure.
;       CX = size of data packet
```

```
;       DS:BX = address of buffer
recv_at_upcall proc far
        assume  ds:nothing

recv_buff1:
        cmp     bx, offset DDP1buffer
        jne     recv_buff2

;       check if we have a client waiting for packets
;       pass to recv_find    es:di->driver_type, cx=#bytes in packet
        mov     DDP1buffsize, cx
        mov     di, offset driver_type
        movseg  es,cs
        mov     dl,cs:driver_class
        call    recv_find

;       es:di->client buffer, or es:di=0 means drop the packet
        mov     ax, es
        or      ax, di
        je      recv_pass1

;       copy ds:si->es:di for cx bytes
        push    es                              ; save the client buffer
        push    di                              ; address (es:di) [jgn]
        movseg  ds,cs
        mov     si, offset DDP1buffer
        mov     cx, DDP1buffsize
        rep     movsb

;       tell receiver copy has been made; ds:si->the packet, cx=length
        pop     si                              ; restore the client buffer
        pop     ds                              ; address into ds:si [jgn]
        mov     cx, DDP1buffsize
        call    recv_copy

recv_pass1:
;       first buffer is free for use again
        mov     DDP1inuse, 00H
        jmp     recv_ret


recv_buff2:
        cmp     bx, offset DDP2buffer
        jne     recv_ret

;       check if we have a client waiting for packets
;       pass to recv_find    es:di->driver_type, cx=#bytes in packet
        mov     DDP2buffsize, cx
        mov     di, offset driver_type
        movseg  es,cs
        call    recv_find

;       es:di->client buffer, or es:di=0 means drop the packet
        mov     ax, es
        or      ax, di
        je      recv_pass2

;       copy ds:si->es:di for cx bytes
        push    es                              ; save the client buffer
        push    di                              ; address (es:di) [jgn]
        movseg  ds,cs
        mov     si, offset DDP2buffer
        mov     cx, DDP2buffsize
        rep     movsb
```

```
;       tell receiver copy has been made; ds:si->the packet, cx=length
;       push    es
        pop     si                              ; restore the client buffer
        pop     ds                              ; address into ds:si [jgn]
;       mov     si, offset DDP2buffer
        mov     cx, DDP2buffsize
        call    recv_copy
recv_pass2:
;       second buffer is now free for use
        mov     DDP2inuse, 00H

recv_ret:
        ret
recv_at_upcall endp

;*******************************************

; Call DOS software interrupt for AppleTalk
; caller must set ds:bx -> parameter block for ATALK.SYS
doATint:
        int     AT_INT                          ; Interrupt ATALK.SYS driver
        ret



        public  timer_isr
timer_isr:
;if the first instruction is an iret, then the timer is not hooked
        iret

;any code after this will not be kept after initialization. Buffers
;used by the program, if any, are allocated from the memory between
;end_resident and end_free_mem.
        public end_resident,end_free_mem
end_resident    label   byte
end_free_mem    label   byte
;*****************************************************************************


        public  usage_msg
usage_msg       db      "usage: localtlk [options] <packet_int_no> [ <IP addres
s> ]",CR,LF,'$' ; [jgn]

        public  copyright_msg
copyright_msg   db      "Packet driver for Apple LocalTalk PC Card, version ",'
0'+(majver / 10),'0'+(majver mod 10),".",'0'+version,CR,LF
                db      "Portions Copyright 1990, Regents of the University of
California",CR,LF,'$'

no_atalk_sys_msg:
        db      "Couldn't locate ATALK.SYS -- packet driver not installed",CR,L
F,'$'
atalk_sys_found_msg:
        db      "ATALK.SYS driver located at software interrupt ",'$'
inf_nodeid_name:
        db      "Attaching to AppleTalk network as node ",'$'
inf_abridge_name:
        db      "AppleTalk network bridge is node ",'$'
ddp_failed_msg:
        db      "Datagram Delivery Protocol socket open failed; return status:
",'$'
ddp_wrong_socket_msg:
        db      "Datagram Delivery Protocol failed; unable to aquire requested
socket",CR,LF,'$'
ddp_open_msg:
        db      "Datagram Delivery Protocol open on socket ",'$'
```

```
atalk_open_msg:
        db      "Attached to AppleTalk network as (net:node:sock): ",'$'
nbp_no_gateway_msg:
        db      "NBP: IPGATEWAY lookup failed; return status: ",'$'
nbp_ipg_addr_msg:
        db      "IPGATEWAY located on AppleTalk network as (net:node:sock): ",'
$'
atp_no_gateway_msg:
        db      "ATP: IPGATEWAY transport setup failed; return status: ",'$'
ipg_gateway_err_msg:
        db      "IP Gateway error: ",'$'
myip_addr_msg:
        db      "My IP address: ",'$'
ns_ip_addr_msg:
        db      "Name Server IP address: ",'$'
bd_ip_addr_msg:
        db      "Broadcast IP address: ",'$'
fs_ip_addr_msg:
        db      "File Server IP address: ",'$'
opcode_msg:
        db      "IPG opcode: ",'$'
nbp_no_register_msg:
        db      "NBP: failed, couldn't register our name; return status: ",'$'
ddp_cant_recv:
        db      "DDP: couldn't initiate read on socket; return status: ",'$'
test_arg_msg:
        db      "Test IP arg parsing: ",'$'
null_msg        db      '$'

ipgateway_name:
        db      01H, '=', 09H, "IPGATEWAY", 01H, '*', '0'
myip_name:
        db      09H, "IPADDRESS", 01H, '*', '0'
myip_name_len   equ     12

; Temporary storage for calls to print_number
dtemp   dw      ?,0

        extrn   set_recv_isr: near

;enter with si -> argument string, di -> wword to store.
;if there is no number, don't change the number.
        extrn   get_number: near

;enter with ds:dx -> argument string, ds:di -> dword to print.
        extrn   print_number: near

;enter with al = char to display
        extrn   chrout: near
;enter with ax,dx holding 32 bits to display in decimal (ax holds low word)
        extrn   decout: near
        extrn   byteout: near
        extrn   wordout: near

        extrn   skip_blanks: near

;enter with si -> argument string, di -> word to store.
;if there is no number, don't change the number.
        extrn   get_number: near

;called with ds:si -> immediately after the entry_point
        public  parse_args
parse_args:
        call    skip_blanks
        lodsb
        cmp     al, CR
        je      no_more_args
        cmp     al, '['                         ; check for square brackets
        je      past_brackets
        dec     si                              ; not a bracket, back up
past_brackets:
        mov     di, offset temp_4bytes          ; get first IP address byte
        call    get_number
        mov     byte ptr test_address, cl
        lodsb
        cmp     al, '.'
        jne     no_more_args

        mov     di, offset temp_4bytes          ; get second IP address byte
        call    get_number
        mov     byte ptr test_address+1, cl
        lodsb
        cmp     al, '.'
        jne     no_more_args

        mov     di, offset temp_4bytes          ; get third IP address byte
        call    get_number
        mov     byte ptr test_address+2, cl
        lodsb
        cmp     al, '.'
        jne     no_more_args

        mov     di, offset temp_4bytes          ; get first IP address byte
        call    get_number
        mov     byte ptr test_address+3, cl

;       mov     dx, offset test_arg_msg
;       mov     di, offset test_address
;push   si
;       call    print_ip_addr
;pop    si

        mov     ax, word ptr test_address
        mov     word ptr static_address, ax
        mov     ax, word ptr test_address+2
        mov     word ptr static_address+2, ax
        mov     use_static, 01H

        lodsb
        cmp     al, ']'
        je      arg_return

;exit with nc if all went well, cy otherwise.
no_more_args:
        dec     si
arg_return:
        clc
        ret


; Initialize our interface to the ATALK.SYS driver.
; NOTE: this initialization code is modeled after the PC/IP LocalTalk
; driver written by Dan Lanciani (ddl@harvard.harvard.edu); the PCIP
; software package can found at husc6.harvard.edu:pub/pcip/pcip.tar.Z
        public  etopen
etopen:
        assume  ds:code

; ATALK.SYS driver may be loaded at a software interrupt somewhere
; between 5CH and 70H. Locate ATALK.SYS driver by scanning for signature.
```

```
isATLoaded:                             ; Look for ATALK.SYS driver
        cld
        call    ATGetInt                ; Load start of intr range
        mov     dx, ax                  ; Save start value in DX
chkloop:
        cmp     dx, 70H                 ; Scanned all possible vectors?
        jne     checkstring             ; No, check this vector
        xor     ax, ax                  ; Yes, driver not found
        jmp     chksplit                ; Skip ahead to return
checkstring:
        mov     bx, dx                  ; Load intr# for scan
        shl     bx, 1                   ; Multiply by 2 (for seg bytes)
        shl     bx, 1                   ; Multiply by 2 (for off bytes)
        xor     ax, ax
        mov     es, ax                  ; Lowest page of memory
        lds     si, es:[bx]             ; Load vector for scan intr#
        mov     ax, ds                  ; Load segment this scan intr#
        or      ax, si                  ; OR with off this scan intr#
        jz      keepchecking            ; Keep checking if no bits
        sub     si, 16                  ; Signature is just before code
        mov     di, offset driverstring ; Load compare string
        mov     cx, 9                   ; Load length of compare string
        movseg  es,cs
        repe    cmpsb                   ; Compare ds:si to es:di
        jne     keepchecking            ; Keep checking if not matched
        call    ATGetInt                ; Matched, get INT# again
        cmp     ax, dx                  ; INT# already set properly?
        jz      chksplit                ; Yes, use this INT#
                                        ; No, we found INT# by scanning
        call    ATPatch                 ; Modify code to match scan
        call    ATGetInt                ; Retrieve final INT#
        jmp     chksplit                ; Skip ahead to return

keepchecking:                           ; Havent found ATALK.SYS driver
        inc     dx                      ; Check next possible INT#
        jmp     chkloop                 ; Loop back to check next INT#

chksplit:                               ; Done with scan for ATALK.SYS
        cmp     ax, 00H                 ; ATALK.SYS driver found?
        jne     atalk_sys_found         ; Yes, skip ahead to continue

        mov     dx, offset no_atalk_sys_msg     ; No, ATALK.SYS not loaded
        jmp     error_wrt               ; Skip ahead to report error

atalk_sys_found:                        ; ATALK.SYS driver found
        movseg  ds,cs
        mov     dtemp, dx               ; Report intr# of ATALK.SYS
        mov     di, offset dtemp
        mov     dx, offset atalk_sys_found_msg
        call    print_number

; We need to establish our Appletalk node
get_our_info:                           ; Get info params from ATALK
        mov     MyInfo.atd_command, AT_GETNETINFO
        mov     MyInfo.atd_status, -1           ; As per PC LocalTalk spec. [jg
n]
        mov     MyInfo.atd_compfun.offs, 0      ; No completion routine [jgn]
        mov     MyInfo.atd_compfun.segm, 0
        mov     bx, offset MyInfo
        call    doATint

        cmp     MyInfo.atd_status, 00H          ; Already initialized?
        je      get_ddp_socket                  ; Yes, skip ahead

        mov     MyInfo.atd_command, AT_INIT     ; No, initialize our node
        mov     MyInfo.atd_compfun.offs, 0      ; No completion routine [jgn]
        mov     MyInfo.atd_compfun.segm, 0
        mov     bx, offset MyInfo
        call    doATint

; We need to establish our AppleTalk/DDP socket
get_ddp_socket:                                 ; Open a DDP socket
        mov     DDPio.ddp_command, DDP_OPENSOCKET
        mov     DDPio.ddp_compfun.offs, 0       ; No completion routine [jgn]
        mov     DDPio.ddp_compfun.segm, 0
        mov     DDPio.ddp_buffptr.offs, offset preview_upcall
        mov     DDPio.ddp_buffptr.segm, cs
        mov     DDPio.ddp_socket, 72            ; ask for experimental sock#
        mov     DDPio.ddp_type, 22              ; ask for IP socket type
        mov     bx, offset DDPio                        ; ds:bx-> DDP param blo
ck
        call    doATint                         ; ask ATALK.SYS for a socket

        cmp     DDPio.ddp_status, 00H           ; error return from ATALK.SYS?
        je      chk_ddp_socket                  ; no, skip ahead to continue
                                                ; yes, no socket for us
        mov     ax, DDPio.ddp_status
        mov     dtemp, ax
        mov     di, offset dtemp
        mov     dx, offset ddp_failed_msg
        call    print_number                    ; report error and stat return
        jmp     error_ret

;**** do we really require socket 72?
chk_ddp_socket:                                 ; check the socket we opened
        cmp     DDPio.ddp_socket, 72            ; did we get the one requested?
        je      ddp_ready                       ; yes, socket is as expected
                                                ; no, but must have socket 72
        mov     DDPio.ddp_command, DDP_CLOSESOCKET
        mov     bx, offset DDPio
        call    doATint                         ; close the assigned socket

        mov     dx, offset ddp_wrong_socket_msg ; load error msg
        jmp     error_wrt                       ; skip ahead to display and ret

ddp_ready:                                      ; DDP socket 72 is ready
        mov     DDPio.ddp_command, DDP_WRITE    ; Use param block for WRITE now

; AppleTalk node and DDP socket have been established
        mov     ax, MyInfo.inf_network
        mov     word ptr dtemp, ax
        mov     al, MyInfo.inf_nodeid
        mov     ah, DDPio.ddp_socket
        mov     byte ptr dtemp+2, al
        mov     byte ptr dtemp+3, ah
        mov     di, offset dtemp
        mov     dx, offset atalk_open_msg
        call    print_at_addr                   ; display AppleTalk node info
        mov     ax, 00H
        mov     dtemp+2, ax

; We need an IP gateway node
nbp_ipgateway:                                  ; Locate our IP gateway node
        movseg  ds,cs
        mov     NBP.nbp_command, NBP_LOOKUP
        mov     NBP.nbp_compfun.offs, 0         ; No completion routine [jgn]
        mov     NBP.nbp_compfun.segm, 0
        mov     NBP.nbp_toget, 01H
        mov     NBP.nbp_buffptr.offs, offset NBPt
        mov     NBP.nbp_buffptr.segm, ds
```

```
        mov     NBP.nbp_buffsize, (size NBPTuple)
        mov     NBP.nbp_interval, 5
        mov     NBP.nbp_retry, 12
        mov     NBP.nbp_entptr.offs, offset ipgateway_name
        mov     NBP.nbp_entptr.segm, ds
        mov     bx, offset NBP
        call    doATint                         ; do name-bind lookup

        cmp     NBP.nbp_status, 00H             ; status return=error?
        jne     nbp_no_gateway                  ; yes, report error and exit

        cmp     NBP.nbp_toget, 01H
        je      atp_setup

nbp_no_gateway:                                 ; NBP lookup failed
        mov     ax, NBP.nbp_status
        mov     dtemp, ax
        mov     di, offset dtemp
        mov     dx, offset nbp_no_gateway_msg   ; display error msg
        call    print_number

        mov     DDPio.ddp_command, DDP_CLOSESOCKET
        mov     bx, offset DDPio
        call    doATint                         ; close the assigned socket

        jmp     error_ret                       ; skip ahead to return

; We need a transport layer to the IP gateway
atp_setup:
        mov     cx, (size AddrBlk)              ; MyGateway = NBPt.tup_addr
        movseg  es,cs
        mov     si, offset NBPt.tup_address
        mov     di, offset MyGateway
        rep     movsb

        mov     di, offset NBPt.tup_address     ; Display our gateway node
        mov     dx, offset nbp_ipg_addr_msg
        call    print_at_addr

        cmp     use_static, 00H                 ; Do we have a static address?
[jgn]
        je      bds_setup                       ; No, don’t change anything
        mov     IPG.ipg_opcode+3, 3             ; Yes, change to IPG_SERVER

bds_setup:
        mov     BDS.bds_buffptr.offs, offset IPG
        mov     BDS.bds_buffptr.segm, ds
        mov     BDS.bds_buffsize, (size IPGInfo)

        mov     ATP.atp_command, ATP_SEND_REQUEST
        mov     ATP.atp_compfun.offs, 0         ; No completion routine [jgn]
        mov     ATP.atp_compfun.segm, 0
        mov     cx, (size AddrBlk)              ; ATP.atp_addr = NBPt.tup_addr
        movseg  es,cs
        mov     si, offset NBPt.tup_address
        mov     di, offset ATP.atp_addrblk
        rep     movsb
        mov     ATP.atp_buffptr.offs, offset IPG
        mov     ATP.atp_buffptr.segm, ds
        mov     ATP.atp_buffsize, (size IPGInfo)
        mov     ATP.atp_interval, 05H
        mov     ATP.atp_retry, 05H
        mov     ATP.atp_flags, XO_BIT
        mov     ATP.atp_bdsbuffs, 01H
        mov     ATP.atp_bdsptr.offs, offset BDS
        mov     ATP.atp_bdsptr.segm, ds
        mov     bx, offset ATP
        call    doATint

        cmp     ATP.atp_status, 00H             ; status return=error?
        jne     atp_no_gateway                  ; yes, report error and exit

        cmp     ATP.atp_bdsbuffs, 01H
        je      chk_ip_opcode

atp_no_gateway:                                 ; ATP setup failed
        mov     ax, ATP.atp_status
        mov     dtemp, ax
        mov     di, offset dtemp
        mov     dx, offset atp_no_gateway_msg   ; display error msg
        call    print_number

        mov     DDPio.ddp_command, DDP_CLOSESOCKET
        mov     bx, offset DDPio
        call    doATint                         ; close the assigned socket

        jmp     error_ret                       ; skip ahead to return

chk_ip_opcode:
        cmp     IPG.ipg_opcode.offs, IPG_ERROR  ; opcode is 32 bit
        jne     save_ipaddr                     ; check one word at a time
        cmp     IPG.ipg_opcode.segm, IPG_ERROR  ; error from IP gateway?
        jne     save_ipaddr                     ; no, transport established
                                                ; yes, ATP setup failed
        mov     dx, offset ipg_gateway_err_msg  ; display IPG error msg
        mov     ah, 9
        int     21H
        mov     dx, offset IPG.ipg_string
        mov     ah, 9
        int     21H
        mov     al, 13                          ; display CR-LF
        call    chrout
        mov     al, 10
        call    chrout

        mov     DDPio.ddp_command, DDP_CLOSESOCKET
        mov     bx, offset DDPio
        call    doATint                         ; close the assigned socket

        jmp     error_ret

; AppleTalk/IP transport layer established
save_ipaddr:
        mov     dx, offset myip_addr_msg
        cmp     use_static, 00H
        jne     show_static
show_dynamic:
        mov     di, offset IPG.ipg_ipaddress
        jmp     show_ipaddr
show_static:
        mov     di, offset static_address
show_ipaddr:
        call    print_ip_addr

        mov     dx, offset ns_ip_addr_msg
        mov     di, offset IPG.ipg_ipname
        call    print_ip_addr

        mov     dx, offset bd_ip_addr_msg
        mov     di, offset IPG.ipg_ipbroad
```

```
        call    print_ip_addr

        mov     dx, offset fs_ip_addr_msg
        mov     di, offset IPG.ipg_ipfile
        call    print_ip_addr

; We need to register ourself with the AppleTalk Name Binding Agent
nbp_register_ourself:
        mov     al, MyInfo.inf_nodeid
        mov     NBPtable.tab_entry.tup_address.ablk_nodeid, al
        mov     al, DDPio.ddp_socket
        mov     NBPtable.tab_entry.tup_address.ablk_socket, al

        ; print our IP address in our NBP table entry
        mov     bx, offset NBPtable.tab_entry.tup_name
        inc     bx
        xor     dx, dx
        cmp     use_static, 00H
        jne     reg_static1
reg_dynamic1:
        mov     dl, byte ptr IPG.ipg_ipaddress
        jmp     reg_format1
reg_static1:
        mov     dl, byte ptr static_address
reg_format1:
        call    decstr
        mov     al, dot_char
        mov     byte ptr ds:[bx], al
        inc     bx
        cmp     use_static, 00H
        jne     reg_static2
reg_dynamic2:
        mov     dl, byte ptr IPG.ipg_ipaddress+1
        jmp     reg_format2
reg_static2:
        mov     dl, byte ptr static_address+1
reg_format2:
        call    decstr
        mov     al, dot_char
        mov     ds:[bx], al
        inc     bx
        cmp     use_static, 00H
        jne     reg_static3
reg_dynamic3:
        mov     dl, byte ptr IPG.ipg_ipaddress+2
        jmp     reg_format3
reg_static3:
        mov     dl, byte ptr static_address+2
reg_format3:
        call    decstr
        mov     al, dot_char
        mov     ds:[bx], al
        inc     bx
        cmp     use_static, 00H
        jne     reg_static4
reg_dynamic4:
        mov     dl, byte ptr IPG.ipg_ipaddress+3
        jmp     reg_format4
reg_static4:
        mov     dl, byte ptr static_address+3
reg_format4:
        call    decstr

        mov     ax, bx
        sub     ax, offset NBPtable.tab_entry.tup_name
        sub     ax, 1
        mov     NBPtable.tab_entry.tup_name, al

        mov     cx, myip_name_len       ; append IPADDR command to our IP
        movseg  es,cs
        mov     si, offset myip_name    ; ds:si -> source
        mov     di, bx                  ; es:di -> dest
        rep     movsb

        ; Register our name with NBP agent
        mov     NBP.nbp_command, NBP_REGISTER
        mov     NBP.nbp_compfun.offs, 0         ; No completion routine [jgn]
        mov     NBP.nbp_compfun.segm, 0
        mov     NBP.nbp_buffptr.offs, offset NBPtable
        mov     NBP.nbp_buffptr.segm, ds
        mov     NBP.nbp_interval, 01H
        mov     NBP.nbp_retry, 03H
        mov     bx, offset NBP
        call    doATint

        cmp     NBP.nbp_status, 00H
        je      atinit_done

nbp_no_register:
        mov     ax, NBP.nbp_status
        mov     dtemp, ax
        mov     di, offset dtemp
        mov     dx, offset nbp_no_register_msg  ; display error msg
        call    print_number

        mov     DDPio.ddp_command, DDP_CLOSESOCKET
        mov     bx, offset DDPio
        call    doATint                         ; close the assigned socket

        jmp     error_ret                       ; skip ahead to return

;**** LocalTalk PC Card initialized, ready to TSR
atinit_done:
        movseg  ds,cs
        clc
        ret

;**** Got an error while initializing LocalTalk PC Card
error_wrt:                                      ; Display an error message
        movseg  ds,cs
        stc
        ret
error_ret:                              ; Board not initialized
        mov     dx,offset null_msg      ;error message already printed.
        stc
        ret

        public  print_parameters
print_parameters:
;echo our command-line parameters
        ret

;*******************************************

; Modify ATALK.SYS interrupt number in doATint code (self-modifying code!)
ATPatch:
        mov     al, dl                          ; Load new interrupt number
        movseg  es,cs
        lea     bx, doATint                     ; es:bx=offset of doATint code
        inc     bx                              ; skip to operator for INT
```

```
        mov     es:[bx], al                     ; modify the code
        ret

; Get ATALK.SYS interrupt number
ATGetInt:
        movseg  es,cs
        lea     bx, doATint                     ; es:bx=offset of doATint code
        inc     bx                              ; skip to operator for INT
        mov     al, es:[bx]                     ; load operator for INT
        xor     ah, ah                          ; zero high byte
        ret                                     ; return INT# to caller

;********************************************

;********************************************

; caller must set ds:si -> dest for string, dx 16-bit value to sprint
decstr:
        mov     di,dx
        cmp     dx, 0
        jne     decstr_nonzero
        mov     al,'0'                  ;yes - easier to just print it, than
        jmp     chrstr                  ;  to eliminate all but the last zero.
decstr_nonzero:

        xor     ax,ax                   ;start with all zeroes in al,bx,bp
        mov     bp,ax

        mov     cx,16                   ;16 bits in one 16 bit registers.
decstr_1:
        rcl     di,1
        xchg    bp,ax
        call    addbit
        xchg    bp,ax
        adc     al,al
        daa
        loop    decstr_1

        mov     cl,'0'                  ;prepare to eliminate leading zeroes.
        call    bytestr                 ;output the first two.
        mov     ax,bp
        jmp     wordstr                 ;output the next four.

addbit: adc     al,al
        daa
        xchg    al,ah
        adc     al,al
        daa
        xchg    al,ah
        ret

;print the char in al at ds:bx
chrstr:
        mov     byte ptr [bx], al
        inc     bx
        ret

wordstr:
        push    ax
        mov     al,ah
        call    bytestr
        pop     ax
bytestr:
        mov     ah,al
        shr     al,1
        shr     al,1
        shr     al,1
        shr     al,1
        call    digstr
        mov     al,ah
digstr:
        and     al,0fh
        add     al,90h  ;binary digit to ascii hex digit.
        daa
        adc     al,40h
        daa
        cmp     al,cl                   ;leading zero?
        je      digstr_1
        mov     cl,-1                   ;no more leading zeros.
        jmp     chrstr
digstr_1:
        ret

; caller must set ds:dx -> argument string, ds:di -> AddrBlk struct
print_at_addr:
;enter with dx -> dollar terminated name of number, di ->dword.
;exit with the number printed and the cursor advanced to the next line.
        mov     ah,9                    ;print the name of the number.
        int     21h
        mov     ax, [di].ablk_network   ;print the network number
        xchg    ah, al                  ; byte-swap network number [jgn]
        mov     dx, 00H
        push    di
        call    decout
        pop     di
        mov     al, ':'
        call    chrout
        xor     ax, ax
        mov     al, [di].ablk_nodeid    ; print the nodeid number
        push    di
        call    decout
        pop     di
        mov     al, ':'
        call    chrout
        xor     ax, ax
        mov     al, [di].ablk_socket    ; print the socket number
        call    decout
        mov     al,CR
        call    chrout
        mov     al,LF
        call    chrout
        ret

; caller must set ds:dx -> argument string, ds:di -> 32 bit ip address
print_ip_addr:
;enter with dx -> dollar terminated name of number, di ->dword.
;exit with the number printed and the cursor advanced to the next line.
        mov     ah,9                    ;print the name of the number.
        int     21h
        mov     al, '['
        call    chrout
        xor     ax, ax
        mov     al, [di]                ;print first byte in decimal.
        mov     dx, 00H
        push    di
        call    decout
        pop     di
        mov     al, '.'
        call    chrout
        xor     ax, ax
```

```
        mov     al, [di+1]              ; print second byte in decimal
        push    di
        call    decout
        pop     di
        mov     al, '.'
        call    chrout
        xor     ax, ax
        mov     al, [di+2]              ; print third byte in decimal
        push    di
        call    decout
        pop     di
        mov     al, '.'
        call    chrout
        xor     ax, ax
        mov     al, [di+3]              ; print fourth byte in decimal
        call    decout
        mov     al, ']'
        call    chrout
        mov     al,CR
        call    chrout
        mov     al,LF
        call    chrout
        ret

code    ends

        end
```