ALTOII CODE3.MU

Copyright Xerox Corporation 1979

*** Derived from ALTOII CODE2.MU, as last modified by
*** Tobol, August 5, 1976 12:13 PM -- fix DIOG2 bug
*** modified by Ingalls, September 6, 1977
*** BitBLT fixed (LREG bug) and extended for new memory
*** modified by Boggs and Taft September 15, 1977 10:10 PM
*** Modified MRT to refresh 16K chips and added XMSTA and XMLDA.
*** Fixed two bugs in DEXCH and a bug in the interval timer.
*** Moved symbol and constant definitions into AltoConsts23.mu.
*** MRT split and moved into two 'get' files.
*** modified by Boggs and Taft November 21, 1977 5:10 PM
*** Fixed a bug in the Ethernet input main loop.
*** modified by Boggs November 28, 1977 3:53 PM
*** Mess with the information returned by VERS

FOUND ON:

<ALTO SOURCE> ALTOII CODE3.DM
; Get the symbol and constant definitions
; #AltoConsts23.mn

; LABEL PREDEFINITIONS
; The reset locations of the tasks:
; [17, 20, NOVEM, ..., KSEC, ..., EREST, MR1, DMT, CURT, DHT, DVT, PART, KMDX, ...]
; [0, 4, 7, 9, 2, 9, P, B < 0, E]
; Locations which may need to be accessible from the Ram or Ram:
; locations which are accessed from the Rom (TRAP1):
; [137, 20, START, RAMRET, RAMCYCX, ..., TRAP1; ]

; Macro-op dispatch table:
; [137, 20, DOINS, DOIND, ENCYCLE, NOPAR, JSR11, U5, U6, U7, ..., RAMTRAP, TRAP; ]

; Parameterless macro-op sub-table:
; [137, 40, DIR, EIR, BRI, CLK, SIO, BLT, BLKS, SIT, JMPR, RDRM, WTRM, DIRS, VERS, DREAD, DWRITE, DEXCH, MUL, DIV, DIOG, **2, BITBLT, XMLDA, XMSTA, ...]

; Cycle dispatch table:
; [137, 20, L0, L1, L2, L3, L4, L5, L6, L7, R7, R6, R5, R4, R3, R2, R1; ]

; Some global R-Registers
$NWW $SR4; State of interrupt system
$SR37 $SR37; Used by MRI, IOMT, interval timer and EIA
$NTEMP $SR25; Public temporary R-Register

MAT ABOUT EVERY 220 INSTRUCTIONS
; The Display Controller

; its R-Registers:
$CBR  $R22;
$CAEC $R23;
$SSLC $R24;
$SBNT  $R26;
$YPOS $R27;
$DOWA $R30;
$CRUX $R28;
$CURD $R21;

; its task specific functions:
$SEVENFIELD  $L024010,000000,000000; F2 = 10 DHT DVT
$SEITMODE $L024011,000000,000000; F2 = 11 DHT
$DDO $L024010,000000,124100; F2 = 10 DWT
!1.2,DVT1,DVT11;
!1.2,NORM,NOMORE;
!1.2,NORMX,HALFX;
!1.2,NORM2,XNORM2;
!1.2,HALFT,HALFT1;
!1.2,HALFX,HALFX2;
!1.2,DVTZ,DVTY;
!1.2,DOTAB,NOTAB;
!1.2,XNOMORE,DUMORE;

; Display Vertical Task

DVT:  MAR= L= DASTART+1;
      CBA= L, L= 0;
      CURD= L;
      SLG= L;
      T= MD;
      L= NNW OR T;
      MAR= CURLOD;
      NNW= L, T= 0-1;
      L= MD XOR T;
      T= MD, EVENFIELD;
      CURX= L, :DVT1;

DVT1: L= BIAS-T-1, TASK, :DVT2;   BIAS THE Y COORDINATE
DVT2: L= BIAS-T, TASK;
DVT2: YPOS= L, :DVT;
; Display Horizontal Task.
; 11 cycles if no block change, 17 if new control block.

DHT:  MAR+ CBA-1;
      L+ SLC -1, BUS=0;
      SLC+ L, :DHT0;

DHT0:  T+ 37400;
       SINK+ MD;
       L+ T+ MD AND T, SETMODE;
      HTAB+ L LCY 8, :NORMODE;

NORMODE:  L+ T+ 377 . T;
          AECI+ L, :REST;

HALFMODE:  L+ T+ 377 . T;
          AECI+ L, :REST, T= 0;

REST:  L+ DWA + T,TASK;
      INCREMENT DWA BY 0 OR NWRDS

NDNX:  DWA+ L, :DHT;

DHT1:  L+ T+ MD+1, BUS=0;
       CBA+ L, MAR= T, :MOREB;

NOMORE:  BLOCK, :DNX;

MOREB:  T+ 37400;
       L+ T+ MD AND T, SETMODE;
       MAR+ CBA+1, :NORMX, EVENFIELD;

NORMX:  HTAB+ L LCY 8, :NODD;

HALFX:  HTAB+ L LCY 8, :NEVEN;

NODD:  L+ T+ 377 . T;
       AECI+ L, :XREST;
       ODD FIELD, FULL RESOLUTION

NEVEN:  L+ 377 AND T;
       AECI=L, T=0;
       EVEN FIELD OR HALF RESOLUTION

XREST:  L+ MD+T;
       T+MD-1;

DNX:  DWA=L, L+T, TASK;
      SLC+L, :DHT;
; Display Word Task

DWT:     T = DWA;
        T = T + 3 + T + 1;
        L = AECL + T, BUS = 0, TASK;
        AECL = L, : DWTZ;

DWTY:    BLOCK;
        TASK, : DWTF;

DWTZ:    L = HTAB - 1, BUS = 0, TASK;
        HTAB = L, : DOTAB;

DOTAB:   DDR = 0, : DWTZ;

NOTAB:   MAR = T + DWA;
        L = AECL - T - 1;
        ALUCY, L = L + 2 + T;
        DWA = L, : XNOMORE;

DOMORE:  DDR = MD, TASK;
        DDR = MD, : NOTAB;

XNOMORE: DDR = MD, BLOCK;
        DDR = MD, TASK;

DWTF:    : DWT;
Alto Ethernet Microcode, Version III, Boggs and Metcalfe

4-way branches using NEXT6 and NEXT7
117, 20, EIFD0, EODOK, EOEOK, EOEOKD, EIFB01, EODPST, EOEPST, EOEST, EIFB10, EODCOL, EOEOC1, EOREST, EIFB11, EODOUGH, **EOEUGH, EERBES;

2-way branches using NEXT7
EODW1, EODWX, and EIGO are all related. Be careful!
117, 10, , EIFF0, EODW1, EIFBAD, EODWX, EIGO;

Miscellaneous address constraints
117, 10, EOFDG0, EODATA, EIDFUL, EIDZ4, EOCORS, EIDATA, EPOST;
117, 10, EIDOK, , EIDMOR, EOFDST;
111, 1, EIFB1;
111, 1, EIFIRST;

2-way branches using NEXT9
111, 2, EOID1P, EOINPN;
111, 2, EOIDMOR, EODEND;
111, 2, EOLODK, EOLOKD;
111, 2, EIDCHT, EIFPRM;
111, 2, EOCDW1, EOCGDG;
111, 2, ECI00K, ECTZT2;
111, 2, EITBGN, EIS3T;
111, 2, EIFNDC, EIFBC;

A Memory Locations
$ECNTR $SR12: Remaining words in buffer
$EPNTR $SR13: points BEFORE next word in buffer

Ethernet microcode Status codes
$ESIDON $377: Input Done
$EOIDON $777: Output Done
$EISFUL $1377: Input Buffer full - words lost from tail of packet
$EOLOAD $1777: Load location overflowed
$ESCZER $2377: Zero word count for input or output command
$ESABRT $2777: Abort - usually caused by reset command
$SNELVR $3377: Never Happen - Very bad if it does

Main memory locations in page 1 reserved for Ethernet
$EPLOC $600: Post location
$EBLOC $601: Interrupt bit mask
$EELOC $602: Ending count location
$ELLOC $603: Load location
$ECLOC $604: Input buffer Count
$EIPLOC $605: Input buffer Pointer
$EOPLOC $606: Output buffer Count
$EOLOC $607: Output buffer Pointer
$EILLOC $610: Host Address

Function Definitions
$EIDFCT $0000000,014004,000100; BS = 4, Input data
$EILFCT $0106013,070013,000100; F1 = 13, Input Look
$EIPFCT $0106014,070014,000100; F1 = 14, Post
$EWFCT $0106015,000000,000000; F1 = 16, Wake-Up
$EODFCT $0206010,000000,124000; F2 = 10, Output data
$EOSFCT $0204011,000000,000000; F2 = 11, Start output
$EBRFCT $0204012,000000,000000; F2 = 12, Reset branch
$EEFCT $0204013,000000,000000; F2 = 13, End of output
$EBFCT $0204014,000000,000000; F2 = 14, Branch
$EODFCT $0204015,000000,000000; F2 = 15, Countdown branch
$EISFCT $0204016,000000,000000; F2 = 16, Start input
; - Whenever a label has a pending branch, the list of possible
; destination addresses is shown in brackets in the comment field.
; - Special functions are explained in a comment near their first use.
; - To avoid naming conflicts, all labels and special functions
; have "E" as the first letter.

; Top of Ethernet Task loop

; Ether Rest Branch Function - ERBFCT
; merge ICMOD and OCMOD flip flops into NEXT6 and NEXT7
; ICMOD and OCMOD are set from ACO [14:15] by the SIO instruction
; 00 neither
; 01 OCMOD - Start output
; 10 ICMOD - Start input
; 11 Both - Reset interface

; in preparation for a hack at EREST, zero EPNTR

EREST: L=O,ERBFCT;  What's happening?
EPNTR= L,;ENOCMD;  [ENOCMD,EOREST,EIREST,ERBRES]
ENOCMD: L=ESNEVR,;EPOST;  Shouldn't happen
ERBRES: L=ESABRT,;EPOST;  Reset Command

; Post status and halt. Microcode status in L.
; Put microstatus,hardstatus in EPLOC, merge c(EBLOC) into NWW.
; Note that we write EPLOC and read EBLOC in one operation

; Ether Post Function - EPFCT. Gate the hardware status
;(LOW TRUE) to Bus [10:15], reset interface.

EPOST:  MAR= EELOC;
        EPNTR= L,TASK;
        MD= ECNTR;
        Save microcode status in EPNTR
MAR= EPLOC;
T= NWW;
MD= EPNTR,EPFCT;
L= MD OR T,TASK;
NWW= L,;EREST;
        double word reference
        BUS AND EPNTR with Status
        Done. Wait for next command

; This is a subroutine called from both input and output (ECQDAO
; and EISET). The return address is determined by testing EBFCB,
; which will branch if the buffer has any words in it, which can
; only happen during input.

ESETUP: NOP;
L= MD,BUS=0;
check for zero length
T= MD-1,;ECNTOK;
[ECNTOK,ECNTZR] start-1

ECNTZR: L= ESCZR,;EPOST;
Zero word count. Abort

; Ether Countdown Branch Function - EBFCB.
; NEXT7 = Interface buffer not empty.

ECNTOK: ECNTR= L,L,T,EBFCB,TASK;
        EPNTR= L,;EODATA;
[EO DATA,EIDATA]
;Ethernet Input

;It turns out that starting the receiver for the first time and
;restarting it after ignoring a packet do the same thing.

23  EIREST: :EIFIGN;        Hack

;Address filtering code.

;When the first word of a packet is available in the interface
;buffer, a wakeup request is generated. The microcode then
;decides whether to accept the packet. Decision must be reached
;before the buffer overflows, within about 14*5.44 usec.
;if EHLOC is zero, machine is 'promiscuous' - accept all packets
;if destination byte is zero, it is a 'broadcast' packet, accept.
;if destination byte equals EHLOC, packet is for us, accept.

;EIREST is really a subroutine that can be called from EIREST
;or from EIGO, output countdown wait. If a packet is ignored
;and EPNTR is zero, EIREST loops back and waits for more
;packets, else it returns to the countdown code.

;Ether Branch Function - EBFCT
;NEXT7 = IDL % OCMD % ICMD % OUTGONE % INGONE (also known as POST)
;NEXT6 = COLLISION - Can't happen during input

153  EIREST: MAR= EHLOC; Get Ethernet address
      T= 377,EBFCT; What's happening?
      L= MD AND T,BUS=0,:EIFOK,:EIFBAD promiscuous?

22  EIFOK: MTEMP= LLCY8,:EIFCHK; [EIFCHK,EIFPRM] Data wakeup

   EIFBAD: ERBFTCT,TASK,:EIFB1;
            [EIFB1] POST wakeup; xCMD FF set?
   EIFB1: :EIFB00;
            [EIFB00,EIFB01,EIFB10,EIFB11]

   EIFB00: :EIFIGN;
   EIFB01: L= ESABRT,:EPOST;
            OCMODE, abort
   EIFB10: L= ESABRT,:EPOST;
            ICMD, abort
   EIFB11: L= ESABRT,:EPOST;
            ICMD and OCMODE, abort

247  EIFPRM: TASK,:EIFBC;
       Promiscuous. Accept

;Ether Look Function - EILFCT; Gate the first word of the
;data buffer to the bus, but do not increment the read pointer.

246  EILFCT: L= 177400,EIFLCT;
         L= MTEMP,T,SH=0; Mask off src addr byte (BUS AND)
         Broadcast?
         SH=0,TASK,:EIFNBC;
         [EIFNBC,EIFBC] Our Address?

   EIFNBC: :EIFIGN;
            [EIFIGN,EISET]

257  EIFBC: :EISET;
        [EISET] Enter input main loop

;Ether Input Start Function - EISFCT. Start receiver. Interface
;will generate a data wakeup when the first word of the next
;packet arrives, ignoring any packet currently passing.

253  EIFIGN: SINK= EPNTR,BUS=0,EPFCR;Reset; Called from output?
        [EISFCT,TASK,:EOCDWX; [EOCDWX,EIGO] Restart rcvr

252  EOCDWX: EWFC,:EOCDWT;

255  EISET: MAR= EICLOC,:ESETUP; Double word reference
        [ESETUP]

;Input Main Loop

;Ether Input Data Function - EIDFCT. Gate a word of data to
;the bus from the interface data buffer, increment the read ptr.

;** WARNING **

;The delay from decoding EIDFCT to gating data to the bus is
;surprising. Some logic in the interface detects the situation
;(which only happens occasionally) and stops SysClk for one cycle.
;Since memory data must be available during cycle 4, and SysClk
;may stop for one cycle, this means that the MD= EIDFCT must
;happen in cycle 3. There is a bug in this logic which occasionally


; stops the clock in the instruction following the EIDFCT, so
; the EIDFCT instruction should not be the last one of the task,
; or it may screw up someone else (such as RDRAM).

; EIDOK, EIDMOR, and EIDPST must have address bits in the pattern:
; xxx1   xxx4   xxx5
; ECBFCT is used to force an unconditional branch on NEXT7

EIDATA: T+ ECNTR-1, BUS=0;
MAR= L+ EPNTR+1, ECBFCT; [EIDMOR,EIDPST] What's happening
EIDMOR: EPNTR+ L, L+ T, ECBFCT; [EIDOK,EIDPST] Guaranteed to branch
EIDOK: MD= EIDFCT, TASK; [EIDZ4] Read a word from the interface
EIDZ4: ECNTR= L, :EIDATA;

; We get to EIDPST for one of two reasons:
; (1) The buffer is full. In this case, an EBFCT (NEXT[7]) is pending.
;    We want to post "full" if this is a normal data wakeup (no branch)
;    but just "input done" if hardware input terminated (branch).
; (2) Hardware input terminated while the buffer was not full.
;    In this case, an unconditional branch on NEXT[7] is pending, so
;    we always terminate with "input done".

EIDPST: L+ ESIDON, :EIDFUL; [EIDFUL,EPOST] Presumed to be INGONE
EIDFUL: L+ ESIFUL, :EPOST; Input buffer overrun
;Ethernet output

;It is possible to get here due to a collision. If a collision happened, the interface was reset (EPFT) to shut off the transmitter. EOSFCF is issued to guarantee more wakeups while generating the countdown. When this is done, the interface is again reset, without really doing an output.

EOREST:  MAR= ELOC;
               Get load
L= R57;
   Use clock as random # gen
EPNTR= LRSH1;
   Use bits [6:13]
MD= EOSFCF;
SHC0,ECNTR= L;
   L= current load
L= OVERFLOWED?
MTEMP= LLSH1,:EOLDK;
   [EOLDK,EOLDDB]

EOLDDB:  L= ESLOAD,:EPOST;
          Load overflow

EOLDK:   L= MTEMP+1;
          Write updated load
MAR= ELOC;
MTEMP= L,TASK;
MD= MTEMP,:EORST1;
   New load = (old lshift 1) + 1

EORST1:  L= EPNTR;
          Continue making random #
EPNTR= LRSH1;
T= 377;
L= EPNTR AND T,TASK;
EPNTR= L,:EORST2;

;At this point, EPNTR has 0.,random number, ENCTR has old load.

EORST2:  MAR= EICLOC;
          Has an input buffer been set up?
T= ECNTR;
L= EPNTR AND T;
SINK= MD,BUS=0;
ECNTR= L,SH=0,EPFCT,:EOINPR,:EOINPR,:EOINPR

EOINPR:  EISFCF,:EOCDWT;
          [EOCDWT,EOCDGO] Enable in under out
EOINPN:  :EOCDWT;
          [EOCDWT,EOCDGO] No input.

;Countdown wait loop. MRT will generate a wakeup every 37 usec which will decrements ECNTR. When it is zero, start the transmitter.

;Ether Wake Function - EWFCT. Sets a flip flop which will cause a wakeup to this task the next time MRT wakes up (every 37 usec). Wakeup is cleared when Ether task next runs. EWFCT must be issued in the instruction AFTER a task.

EOCDWT:  L= 177400,EBFCT;
          What's happening?
EPNTR= L,EBFCT,:EOCDW0,[EOCDW0,EOCDRS] Packet coming in?
EOCDW0:  L= ECNTR-1,BUS=0,TASK,:EOCDW1,[EOCDW1,EGO]
EOCDW1:  ECNTR= L,EWFCT,:EOCDWT,[EOCDWT,EOCDGO]
EOCDRS:  L= ESABRT,:EPOST;
          [EPOST] POST event

EGO:     :EIFRST;
          [EIFRST] Input under output
; Output main loop setup

EOCGO: MAR+EOCLOC;  Double word reference
       EPFCT;      Reset Interface
       EOSFCT, ESETUP;  Start Transmitter

; Ether Output Start Function - EOSFCT. The interface will generate
;a burst of data requests until the interface buffer is full or the
;memory buffer is empty, wait for silence on the Ether, and begin
;transmitting. Thereafter it will request a word every 5.44us.

; Ether Output Data Function - EODFCT. Copy the bus into the
;interface data buffer, increment the write pointer, clears wakeup
;request if the buffer is now nearly full (one slot available).

; Output main loop

EODATA: L+MAR+EPTR+1,EBFCT;  What's happening?
       T+ECNTR-1,BUS=0,:EODOK;  [EODOK,EODPST,EODCOL,EODUGH]
EODOK:  EPTR=L,T+:EODMOR;  [EODMOR,EODEND]
EODMOR: ECNTR=L,TASK;
       EODFCT-MD,:EODATA;  Output word to transmitter
EODPST: L+ESABRT,:EPOST;  [EPOST] POST event
EODCOL: EPFCT,:EOREST;  [EOREST] Collision
EODUGH: L+ESABRT,:EPOST;  [EPOST] POST + Collision

; Ether EOT Function - EEFCT. Stop generating output data wakeups,
;the interface has all of the packet. When the data buffer runs
;dry, the interface will append the CRC and then generate an
;OUTGONE post wakeup.

EODEND: EEFCT;
       TASK;  Disable data wakeups
       :EEOET;  Wait for EEFCT to take
       :EEOET;  Wait for Outgone

; Output completion. We are waiting for the interface buffer to
;empty, and the interface to generate an OUTGONE Post wakeup.

EEOET:  EBFCT;
       :EEOOK;  What's happening?
       [EEOOK,EEOEPST,EEOECOL,EEOUGH]
EEOOK:  L+ESNEVR,:EPOST;  Runaway Transmitter. Never Never.
EEOEPST: L+ESDODN,:EPOST;  POST event. Output done
EEOECOL: EPFCT,:EOREST;  Collision
EEOUGH: L+ESABRT,:EPOST;  POST + Collision
Memory Refresh Task,
Mouse Handler,
EIA Handler,
Interval Timer,
Calendar Clock, and
part of the cursor.

117,20,TX0,TX6,TX3,TX2,TX8,TX5,TX1,TX7,TX4,.......;
11,2,DTIMER,NOTIMER;
11,2,NOTIMERINT,TIMERINT;
11,2,DOCUR,NOCUR;
11,2,SHOWC,WAITC;
11,2,SPCHK,NOSPC;

11,2,NOCLK,CLOCK;
11,1,MRTLAST;
11,2,CNOTLAST,CLAST;

$CLOCKTEMP $R11;
$REFIIMSK $7777;

; * * * A T T E N T I O N * * *
; There are two versions of the Memory refresh code:
; AltoiIMRT4K.mu for refreshing 4K chips
; AltoiIMRT16K.mu for refreshing 16K chips
; You must name one or the other 'AltoiIMRT.mu'.
; I suggest the following convention for naming the resulting .MB file:
; AltoiICode3.MB for the 4K version
; AltoiICode3XM.MB for the 16K version

#AltoiIMRT.mu;

CLOCK: MAR= CLOCLOC;
NPO; L= MD+1;
MAR= CLOCLOC;
MTEMP= L, TASK;
MD= MTEMP, :NOCLK;

DOCUR: L= T= YPOS;
CHECK FOR VISIBLE CURSOR ON THIS SCAN
SHKC0, L= 20-T-1; ***x13 change: the constant 20 was 17
SHKC0, L= 2+T, :SHOWC: [SHOWC,WAITC]

WAITC: YPOS= L, L= 0, TASK, :MRTLAST; SQUASHES PENDING BRANCH
SHOWC: MAR= CLOCLOC+T+1, :CNOTLAST;

CNOTLAST: T= CURX, :CURF;
CLAST: T= 0;
CURF: YPOS= L, L= T;
CURX= L;
L= MD, TASK;
CURDATA= L, :MRT;
;AFTER THIS DISPATCH, T WILL CONTAIN XCHANGE, L WILL CONTAIN YCHANGE-1

TX1:  L+ T+ ONE +T, :M00;  Y=0, X=1
TX2:  L+ T+ ALLONES, :M00;  Y=0, X=-1
TX3:  L+ T+ 0, :M00;  Y=1, X=0
TX4:  L+ T+ ONE AND T, :M00;  Y=1, X=1
TX5:  L+ T+ ALLONES XOR T, :M00;  Y=1, X=-1
TX6:  T+ 0, :M00;  Y=-1, X=0
TX7:  T+ ONE, :M00;  Y=-1, X=1
TX8:  T+ ALLONES, :M00;  Y=-1, X=-1

M00:  MAR= MOUSELOC;
START THE FETCH OF THE COORDINATES
MTEMP= L;
YCHANGE -1
L= MD+ T;
X+ XCHANGE
T= MD;
Y
T+ MTEMP+ T+1;
Y+ (YCHANGE-1) + 1
MTEMP= 1, L+ T;
MAR= MOUSELOC;
NOW RESTORE THE UPDATED COORDINATES
CLOCKTEMP= L;
MD= MTEMP, TASK;
MD= CLOCKTEMP, :MRTA;
; CURSOR TASK

; Cursor task specific functions
$XREG   SL026010,000000,124000; F2 = 10
$CSR    SL026011,000000,124000; F2 = 11

CURT:   XREG= CURX, TASK;
        CSR= CURDATA, :CURT;

; PREDEFINITION FOR PARITY TASK.
; THE CODE IS AT THE END OF THE FILE
!17,20,PR0,,PR2,PR3,PR4,PR5,PR6,PR7,PR8,........
; NOVA EMULATOR

$SSAD  $SR5;
$SPC  $SR6;

; USED BY MEMORY INIT

$17,00,01,02,03,04,05,06,07;
$11,2,FINSTO,INCPC;
$11,2,ERERead,FINJMP;
$11,2,EREReadDone,FContRead;
$11,2,EtherBoot,DiskBoot;

; EOVMEN: IR=LR+MAR+0, :INXB, SAD= L; LOAD SAD TO ZERO THE BUS. STORE PC AT 0
Q0:  L= ONE, :INXA;
Q1:  L= TOTUC, :INXA;
Q2:  L=402, :INXA;
Q3:  L= 402, :INXA;
Q4:  L= ONE, :INXA;
Q5:  L=377+1, :INXE;
Q6:  L=ONE, :INXE;
Q7:  MAR= DSTART;

R37= L;
MAR= 177034;
L= 100000;
WW= L, T= 0-1;
L= MD XOR T, BUSOOG; \(*** X21 \) change.
MAR= BBAD, :EtherBoot; \[EtherBoot, DiskBoot\] \(*** X21 \) change.

DiskBoot: SAD= L, L= 0+1;
MD= SAD;
MAR= KBKLKADR, :FINSTO;

; Ethernet boot section added in X21.
$NegBreathM1  $177175;
$ThNovaGo  $3; First data location of incoming packet

; EtherBoot: L=EthNovaGo, :ERERead; [ERERead, FINJMP]

; EContRead: MAR= ELOC; See if status is still 0
T= 377; Status for correct read
L= MD XOR T, TASK, BUS=0;
SAD= L, :EREReadDone; [EREReadDone, EContRead]

; EReadDone: MAR= 2;
T= NegBreathM1; -(Breath-of-life)-1
T=MD+T+1;
L=SAD OR T;
SH=0, :EtherBoot;

; SUBROUTINE USED BY INITIALIZATION TO SET UP BLOCKS OF MEMORY
$SOFFset  $57B;

; INAX: T=ONE, :INXCom; \(*** X21 \) change.
INXE: T=SOFFset, :INXCom; \(*** X21 \) addition.
INXcom: MAR= T=IR= SAD+T;
PC= L, L= 0+T+1;

INXB: MD= PC;
SINK= DISP, BUS,TASK;
SAD= L, :Q0;
; REGISTERS USED BY NOVA EMULATOR
SAC0 $R3; AC'S ARE BACKWARDS BECAUSE THE HARDWARE SUPPLIES THE
SAC1 $R2; COMPLEMENT ADDRESS WHEN ADDRESSING FROM IR
SAC2 $R1;
SAC3 $R0;
$XREG $R7;

; PREDEFINITIONS FOR NOVA
117, 20, GETAD, G1, G2, G3, G4, G5, G6, G7, G10, G11, G12, G13, G14, G15, G16, G17;
117, 20, XCTAB, XJSR, XISZ, XDSZ, XLDA, XSTA, CONVERT, ........;
13, 4, SHIFT, SH1, SH2, SH3;
1, 2, MAYBE, NOINT;
1, 2, DINPUT, DISO;
1, 2, SOMEACTIVE, NOACTIVE;
1, 2, EXIT, NEXIT;
117, 1, ODOCX;
1, 2, EIR, EIR1;
17, 1, INTCODE;
1, 2, INTSOFF, INTSON; **$X21 addition for DIRS
17, 10, ENCYCRET, RAMCYCRET, CYX2, CYX3, CYX4, CONVCYCRET,;
17, 2, MOREBLT, FINTBLT;
1, 2, DOUTPUT, DISABLED;

; ALL INSTRUCTIONS RETURN TO START WHEN DONE

START: T ← MAR+PC+SKIP;
STARTI: L ← NW, BUS=0; BUS# 0 MEANS DISABLED OR SOMETHING TO DO
         MAYBE, SHC0, L ← 0+T+1; SHC0 MEANS DISABLED
         MAYBE: PC ← L, L ← T, :DOINT;
         NOINT: PC ← L, :DISO;

DOINT: MAR ← WWLOC; INTCODE; TRY TO CAUSE AN INTERRUPT

; DISPATCH ON FUNCTION FIELD IF ARITHMETIC INSTRUCTION,
; OTHERWISE ON INDIRECT BIT AND INDEX FIELD

DISO: L ← T ← IR= MD; SKIP CLEARED HERE

; DISPATCH ON SHIFT FIELD IF ARITHMETIC INSTRUCTION,
; OTHERWISE ON THE INDIRECT BIT OR IR[3-7]

DIS1: T ← ACSOURCE, :GETAD;

; GETAD MUST BE 0 MOD 20
GETAD: T ← 0, :DOINS;
G1: T ← PC ← 1, :DOINS;
G2: T ← AC2, :DOINS;
G3: T ← AC3, :DOINS;
G4: T ← 0, :DOINS;
G5: T ← PC ← 1, :DOINS;
G6: T ← AC2, :DOINS;
G7: T ← AC3, :DOINS;
G10: L ← 0+T-1, TASK, :SHIFT;
G11: L ← 0-1, TASK, :SHIFT;
G12: L ← 0+1, TASK, :SHIFT;
G13: L ← 0+1, TASK, :SHIFT;
G14: L ← ACDEST-T-1, TASK, :SHIFT;
G15: L ← ACDEST-T, TASK, :SHIFT;
G16: L ← ACDEST+T, TASK, :SHIFT;
G17: L ← ACDEST AND T, TASK, :SHIFT;

SHIFT: DNS ← L LCY 8, :START; SWAP BYTES
SH1: DNS ← L RSH 1, :START; RIGHT 1
SH2: DNS ← L RSH 1, :START; LEFT 1
SH3: DNS ← L, :START; NO SHIFT

DOINS: L ← DISP + T, TASK, :SAVAD, IDISP;
DOIND: L ← MAR+DISP+T;
XREG ← L;
L ← MD, TASK, IDISP, :SAVAD;

BRI: L ← MAR+PCLOC ; INTERRUPT RETURN BRANCH
BRI0: T ← 77777;
L ← NWW AND T, SH < 0;
NWW ← L, :EIRO; BOTH EIR AND BRI MUST CHECK FOR INTERRUPT
; REQUESTS WHICH MAY HAVE COME IN WHILE INTERRUPTS WERE OFF

EIRO: L ← MD, :DOINT;
EIRI: L ← PC, :DOINT;

;***X21 addition
; DIRS - 61013 - Disable Interrupts and Skip if they were On
DIRS: T ← 100000;
      L ← NWW AND T;
      L ← PC+1, SH = 0;

; DIR - 61000 - Disable Interrupts
DIR: T ← 100000, :INTOFF;
    INTOFF: L ← NWW OR T, TASK, :INTZ;

INTSON: PC ← L, :INTOFF;

; EIR - 61001 - Enable Interrupts
EIR: L ← 100000, :BRI0;

; SIT - 61007 - Start Interval Timer
SIT: T ← ACO;
     L ← R37 OR T, TASK;
     R37 ← L, :START;

FINJSR: L ← PC;
       AC3 ← L, L ← T, TASK;
FINJMP: PC ← L, :START;
SAVAD: SAD ← L, :XCRAB;

; JSRII - 64400 - JSR double indirect, PC relative. Must have X=1 in opcode
; JSRII - 65000 - JSR double indirect, AC2 relative. Must have X=2 in opcode
JSRII: MAR ← DISP+T; FIRST LEVEL
       IR ← JSRCX;
       T ← MD, :DOIND; THE IR+ INSTRUCTION WILL NOT BRANCH
;TRAP ON UNIMPLEMENTED OP-CODES. SAVES PC AT
;TRAPPIC, AND DOES A JMP@ TRAPVEC ! OPCODE.
TRAP: XREG+ L LCY 8: THE INSTRUCTION
TRAP1: MAR= TRAPPIC;***X13 CHANGE: TAG 'TRAP1' ADDED
JR= T- 37;
MD= PC;
T= XREG.T;
T= TRAPCON+T+1, :DOIND; T NOW CONTAINS 4+OPCODE
; THIS WILL DO JMP@ 530+OPCODE

;***X21 CHANGE: ADDED TAG RAMTRAP
RAMTRAP: SWMODE, :TRAP;

; Parameterless operations come here for dispatch.
!1,2,NPNOTRAP,NPTRAP;

NOPAR: XREG+ L LCY 8;  ***X21 change. Checks < 27.
T=27;
L<DISP-T;
ALUCY;
SINK+DISP, SINK+X37, BUS, TASK, :NPNOTRAP;

NPNOTRAP: :DIR;

NPTRAP: :TRAP1;

;***X21 addition for debugging w/ expanded DISP Prom
U5: :RAMTRAP;
U6: :RAMTRAP;
U7: :RAMTRAP;
; MAIN INSTRUCTION TABLE. GET HERE:
; (1) AFTER AN INDIRECTION
; (2) ON DIRECT INSTRUCTIONS

XCTAB: L= SAD, TASK, :FINJMP; JMP
XJSR:  T= SAD, :FINJSR; JSR
XISZ:  MAR= SAD, :ISZ1; ISZ
XDSZ:  MAR= SAD, :DSZ1; DSZ
XLDA:  MAR= SAD, :FINLOAD; LDA 0-3
XSTA:  MAR= SAD; /*NORMAL
XSTA1: L= ACDEST, :FINSTO; /*NORMAL

; BOUNDS-CHECKING VERSION OF STORE
; SUBST ";**<CR>" TO "<CR>**" TO ENABLE THIS CODE:
;** 11,2,XSTA1,XSTA2;
;** 11,2,DOSTA,TRAPSTA;
;**XSTA:  MAR= 10;
;** L= MD-T;
;** T= SAD;
;** T= MD, ALUCY;
;** L= SAD-T, :XSTA1;
;**L= MD-T;
;** L= MD, ALUCY;
;** L= MA-SAD, :XSTA3;
;**L= 177, :DOSTA;
;**TRAPSTA:  MAR= SAD;  CAUSE A SWAT
;**DOSTA:  MAR= SAD;  DO THE STORE NORMALLY
;** L= ACDEST, :FINSTO;

DSZ1:  T= ALLONES, :FINISZ;
ISZ1:  T= ONE, :FINISZ;

FINSTO: SAD= L,TASK;
FINST1: MD= SAD, :START;

FINLOAD: NOP;
LOADX: L= MD, TASK;
LOADD: ACDEST= L, :START;

FINISZ: L= MD+T;
   MAR= SAD, SH=0;
   SAD= L, :FINSTO;

INCP:  MD= SAD;
   L= PC+1, TASK;
   PC= L, :START;
DIVIDE. THIS DIVIDE IS IDENTICAL TO THE NOVA DIVIDE EXCEPT THAT
IF THE DIVIDE CANNOT BE DONE, THE INSTRUCTION FAILS TO SKIP, OTHERWISE
IT DOES. CARRY IS UNDISTURBED.

11,2,DODIV,NODIV;
11,2,DIVL,ENDDIV;
11,2,NOOVF,OVF;
11,2,DX0,DX1;
11,2,NOSUB,DOSUB;

DIV: T+ AC2;
DIVX: L+ AC0 - T;  DO THE DIVIDE ONLY IF AC2>AC0
       ALUCY, TASK, SAD+ L, L= 0+1;
       :DODIV, SAD+ L LSH 1;
       SAD= 2. COUNT THE LOOP BY SHIFTING

NODIV: :FINBLT;  ***X21 change.
DODIV: L+ AC0, :DIV1;

DIVL: L+ AC0;
DIV1: SHC0, T+ AC1;  WILL THE LEFT SHIFT OF THE DIVIDEND OVERFLOW?
       :NOOVF, AC0= L MLSH 1, L= T+ 0+1;  L+ AC1, T= 0

OVF: AC1= L LSH 1, L= 0+INCT, :NOV1;  L= 1. SHIFT OVERFLOWED

NOOVF: AC1= L LSH 1, L= T;  L= 0. SHIFT OK

NOV1: T+ AC2, SH=0;
       L+ AC0-T, :DX0;

DX1: ALUCY;
     DO THE TEST ONLY IF THE SHIFT DIDN'T OVERFLOW. IF
         IT DID, L IS STILL CORRECT, BUT THE TEST WOULD GO
         THE WRONG WAY.
         :NOSUB, T+ AC1;

DX0: :DOSUB, T+ AC1;

DOSUB: AC0= L, L= 0+INCT;  DO THE SUBTRACT
       AC1= L;  AND PUT A 1 IN THE QUOTIENT

NOSUB: L= SAD, BUS=0, TASK;
       SAD+ L LSH 1, :DIV1;

ENDDIV: L+ PC+1, TASK, :DOIT; ***X21 change. Skip if divide was done.
;MULTIPLY. THIS IS AN EXACT EMULATION OF NOVA HARDWARE MULTIPLY.
;AC2 IS THE MULTIPLIER, AC1 IS THE MULTIPLICAND.
;THE PRODUCT IS IN AC0 (HIGH PART), AND AC1 (LOW PART).
;PRECISELY: AC0,AC1 = AC1*AC2 + AC0

;11,2,DOMUL,NOMUL;
;11,2,MPYL,MPYA;
;11,2,NADDIER,ADDIER;
;11,2,NOSPILL,SPILL;
;11,2,NOADDDX,ADDX;
;11,2,NOСПИL,СПИЛ;

MUL: L ← AC2-1; BUS=0;
MPYX: XREG=L;L ← 0; :DOMUL; GET HERE WITH AC2-1 IN L. DON'T MUL IF AC2=0
DOMUL: TASK; L ← 10+1;
        SAD+ L; COUNT THE LOOP IN SAD

MPYL: L ← AC1; BUSODD;
       T ← AC0; :NOADDIER;

NADDIER: AC1 ← L MRSH 1; L ← T; T ← 0; :NOSPILL;
ADDIER: L ← T ← XREG+INCT;
        L ← AC1; ALUCY; :NOADDIER;

SPILL: T ← ONE;
NOSPILL: AC0 ← L MRSH 1;
        L ← AC1; BUSODD;
        T ← AC0; :NOADDDX;

NOADDDX: AC1 ← L MRSH 1; L ← T; T ← 0; :NOСПИЛ;
ADDX: L ← T ← XREG+INCT;
        L ← AC1; ALUCY; :NOADDDX;

SPИЛ: T ← ONE;
NOSPИЛ: AC0 ← L MRSH 1;
        L ← SAD+1; BUS=0; TASK;
        SAD+ L; :MPYL;

NOMUL: T ← AC0;
        AC0 ← L; L ← T; TASK; CLEAR AC0
        AC1 ← L; AND REPLACE AC1 WITH AC0

MPYA: :FINBLT; ***X21 change.
; CYCLE ACO LEFT BY DISP MOD 20B, UNLESS DISP=0, IN WHICH
; CASE CYCLE BY AC1 MOD 20B
; LEAVES AC1=CYLE COUNT-1 MOD 20B

$CYRET $R6; Shares space with SAD.
$CYCOUT $R7; Shares space with XREG.

I1,2,EMCYCX,ACCYCLE;
I1,1,Y1;
I1,1,Y2;
I1,1,Y3;
I1,1,Z1;
I1,1,Z2;
I1,1,Z3;

EMCYCLE: L= DISP, SINK+ X17, BUS=0; CONSTANT WITH BS=7
CYCP: T= AC0, :EMCYCX;

ACCYCLE: T= AC1;
L= 17 AND T, :CYCP;

EMCYCX: CYCOUT+L, L=0, :RETCYCX;

RMCYCX: CYCOUT+L, L=0+1;

RETCYCX: CYRET+L, L=0+T;
SINK+CYCOUT, BUS;
TASK, :L0;

; TABLE FOR CYCLE
R4: CYCOUT= L MRLSH 1;
Y3: L= T= CYCOUT, TASK;
R3X: CYCOUT= L MRLSH 1;
Y2: L= T= CYCOUT, TASK;
R2X: CYCOUT= L MRLSH 1;
Y1: L= T= CYCOUT, TASK;
R1X: CYCOUT= L MRLS 1, :ENDCYCLE;

L4: CYCOUT= L MLUSH 1;
Z3: L= T= CYCOUT, TASK;
L3: CYCOUT= L MLUSH 1;
Z2: L= T= CYCOUT, TASK;
L2: CYCOUT= L MLUSH 1;
Z1: L= T= CYCOUT, TASK;
L1: CYCOUT= L MLUSH 1, :ENDCYCLE;
L0: CYCOUT= L, :ENDCYCLE;

L8: CYCOUT= L LCY 8, :ENDCYCLE;
L7: CYCOUT= L LCY 8, :Y1;
L6: CYCOUT= L LCY 8, :Y2;
L5: CYCOUT= L LCY 8, :Y3;

R7: CYCOUT= L LCY 8, :Z1;
R6: CYCOUT= L LCY 8, :Z2;
R5: CYCOUT= L LCY 8, :Z3;

ENDCYCLE: SINK+ CYRET, BUS, TASK;
:EMCYCRET;

EMCYCRET: L=CYCOUT, TASK, :LOAD;
RMCYCRET: T=PC, BUS, SWNODE, :TORAM;
; Scan convert instruction for characters. Takes DWAX (Destination ; word address) - NWRDS in ACO, and a pointer to a .AL-format font ; in AC3. AC2+displacement contains a pointer to a two-word block ; containing NWRDS and DBA (Destination Bit Address).

$XH  $R10;
$SDWAX $R36;
$SMASK $R36;

11.2, HDLOOP, HDE2IT;
11.2, MERGE, STORE;
11.2, NFIN, FIN;
117.2, DOBOTH, MOVELoop;

CONVERT: MAR=XREG+1;  Got here via indirect mechanism which
           left first arg in SAD, its address in XREG.
           T=17;
           L+MD AND T;
           T=MAR=AC3;
           AC1+L;
           L+MD T= TASK;
           AC3+L;
           AC3+Character descriptor block address(Char)
           MAR=AC3+1;
           T=17400;
           IR=L+MD AND T;  IR=XH
           XH=L LCY B, :ODDCX;
           XH register temporarily contains HD ODCCX:
           L=ACO, :HDE2ETER;

HDLOOP:  T=SAD;
         L=DWAX+T;

HDE2ETER:  DWAX=L;
           DWAX + AC0+HD*NWRDS
L=XH-1, BUS=0, TASK;
XH=L, :HDLOOP;

HDE2IT:  T=MASKTAB;
         MAR=T=AC1+T;
         L=DISP;
         XH=L;
         XH register now contains XH
         L+MD;
         MASK+L, L+0+T=1, TASK;
         AC1+L;
***X21. AC1 = (DBA&17)+1
L=5;
         IR=SAD, TASK;
         CYRET=L, :MOVELoop;
         CYRET+CALL5

MOVELoop:  L=T+XH-1, BUS=0;
            MAR=AC3-T-1, :NFIN;

NFIN:  XH=L;
        T=DISP;
        L=DWAX+T;
        Update destination address
        T+MD;
        SINK+AC1, BUS;
        DWAX=L, L=T, TASK, :L0; Call Cycle subroutine

CONVCYRET:  MAR=DWAX;
            T=MASK, BUS=0;
            T=CYCOUT.T, :MERGE;
            Data for first word. If MASK=0
            ; then store the word rather than
            ; merging, and do not disturb the
            ; second word.
            Data for second word. First word now merged.
MERGE:  L=XREG AND NOT T;
        T+MD OR T;
        XREG=L, L=T;
        MTEMP+L;
        MAR=DWAX;
        SINK+XREG, BUS=0, TASK;
        MD+MTEMP, :DOBOTH;
        XREG=0 means only one word
        ; is involved.

DOBOTH:  MAR=DWAX+1;
         T=XREG;
         T=XREG;
L=MD OR T;
MAR=DWAX+1;
XREG=L, TASK;
STORE: MD=XREG, :MOVELOOP;

***X21. TASK added.

FIN: L=AC1-1;
AC1=L;
IR=SH3CONST;
L=MD, TASK, :SH1;

***X21. Return AC1 to DBA#17.
*** ... bletch ...
;RCLK - 61003 - Read the Real Time Clock into AC0, AC1
RCLK: MAR+ CLOCKLOC;
    L+ R37;
    AC1+ L ;LOADX;

;SIO - 61004 - Put AC0 on the bus, issue STARTF to get device attention,
;Read Host address from Ethernet interface into AC0.
SIO:    L+ AC0, STARTF;
    T+ 77777;           ***XZ1 sets AC0[0] to 0
    L+ RSNF AND T;
LTOACO: AC0+ L, TASK, ;TOSTART;

;EngNumber is a constant returned by VERS that contains a description
;of the Alto and it's Microcode. The composition of EngNumber is:
;    bits 0-3    Alto engineering number
;    bits 4-7    Alto build
;    bits 8-16   Version number of Microcode
;Use of the Alto Build number has been abandoned.
;the engineering number (EngNumber) is in the MRT files because it
; it different for Altos with and without Extended memory.
VERS:   T+ EngNumber;
        L+ 3+T, ;LTOACO;        ***V3 change

;XMLDA - Extended Memory Load Accumulator.
;    AC0 + 0AC1 in the alternate bank
XMLDA: XMAR+ AC1, ;FINLOAD;     ***V3 change

;XMSTA - Extended Memory Store Accumulator
;    0AC1 + AC0 in the alternate bank
XMSTA: XMAR+ AC1, ;XSTA1;        ***V3 change
;BLT - 61005 - Block Transfer
;BLKS - 61006 - Block Store

; Accepts
; Acc/ BLT: Address of first word of source block-1
; BLKS: Data to be stored
; AC1/ Address of last word of destination block
; AC3/ NEGATIVE word count

; Leaves
; AC0/ BLT: Address of last word of source block+1
; BLKS: Unchanged
; AC1/ Unchanged
; AC2/ Unchanged
; AC3/ 0

; These instructions are interruptable. If an interrupt occurs,
; the PC is decremented by one, and the ACs contain the intermediate
; so the instruction can be restarted when the interrupt is dismissed.

!1,2,PERHAPS, NO;

BLT:  L= MAR= AC0+1;
      AC0= L;
      L= MD, :BLKSA;
BLKS: L= AC0;
BLKSA: T= AC3+1, BUS=0;
       MAR= AC1+T, :MOREBLT;
MOREBLT: XREG= L, L= T;
          AC3= L, TASK;
          MD= XREG;        STORE
          L= NWW, BUS=0;    CHECK FOR INTERRUPT
          SHC0, :PERHAPS, L= PC-1;  Prepare to back up PC.
NO:   SINK= DISP, SINK= M7, BUS, :DISABLED;
PERHAPS: SINK= DISP, SINK= M7, BUS, :DOIT;
DISABLED: :DIR; GOES TO BLT OR BLKS
FINBLT: T=777;  ***X21. PC in [177000-177777] means Ram return
          L=PC+T;1;
          L=PC AND T, TASK, ALUCY;
TOSTART: XREG= L, :START;
RAMRET: T=XREG, BUS, SWMODE;
TORAM:  :NOVEM;
; PARAMETERLESS INSTRUCTIONS FOR DIDDLING THE WCS.

; JMPRAM - 61010 - JUMP TO THE RAM ADDRESS SPECIFIED BY AC1
; JMPHR:  T+AC1, BUS, SNODE, :TORAM;

; RDRAM - 61011 - READ THE RAM WORD ADDRESSED BY AC1 INTO AC0
; RDRM:  T+ AC1, RDRAM;
;       L+ ALLONES, TASK, :LOADD;

; WRTRAM - 61012 - WRITE AC0, AC3 INTO THE RAM LOCATION ADDRESSED BY AC1
; WRTRM:  T+ AC1;
;        L+ AC0, WRTRAM;
;        L+ AC3, :FINBLT;
; DOUBLE WORD INSTRUCTIONS

; DREAD - 61015
; AC0← rv(AC3); AC1← rv(AC3 xor 1)

DREAD:  MAR← AC3;  START MEMORY CYCLE
        NOP;  DELAY
DREAD1:  L← MD;  FIRST READ
        T← MD;  SECOND READ
        AC0← L, L+T, TASK;  STORE MSW
        AC1← L, :START;  STORE LSW

; DWRITE - 61016
; rv(AC3)← AC0; rv(AC3 xor 1)← AC1

DWRITE:  MAR← AC3;  START MEMORY CYCLE
         NOP;  DELAY
         MD← AC0, TASK;  FIRST WRITE
         MD← AC1, :START;  SECOND WRITE

; DEXCH - 61017
; t← rv(AC3); rv(AC3)← AC0; AC0← t
; t← rv(AC3 xor 1); rv(AC3 xor 1)← AC1; AC1← t

DEXCH:  MAR← AC3;  START MEMORY CYCLE
        NOP;  DELAY
        MD← AC0;  FIRST WRITE
        MD← AC1, :DREAD1;  SECOND WRITE, GO TO READ

; DIOGNose INSTRUCTIONS

; DIOG1 - 61022
; Hamming Code← AC2
; rv(AC3)← AC0; rv(AC3 xor 1)← AC1

DIOG1:  MAR← ERRCTRL;  START WRITE TO ERROR CONTROL
         NOP;  DELAY
         MD← AC2, :DWRITE;  WRITE HAMMING CODE, GO TO DWRITE

; DIOG2 - 61023
; rv(AC3)← AC0
; rv(AC3)← AC0 xor AC1

DIOG2:  MAR← AC3;  START MEMORY CYCLE
        T← AC0;  SETUP FOR XOR
        L← AC1 XORT;  DO XOR
        MD← AC0;  FIRST WRITE
        MAR← AC3;  START MEMORY CYCLE
        AC0← L, TASK;  STORE XOR WORD
        MD← AC0, :START;  SECOND WRITE
; INTERRUPT SYSTEM. TIMING IS 0 CYCLES IF DISABLED, 18 CYCLES
; IF THE INTERRUPTING CHANNEL IS INACTIVE, AND 36+6*N CYCLES TO CAUSE
; AN INTERRUPT ON CHANNEL N

INTCODE: PC< L, IR< 0;
           T< NW;
           T< MD OR T;
           L< MD AND T;
           SAD< L, L< T, SH<0;
           SAD HAD POTENTIAL INTERRUPTS
           NW< L, L<0+1, SOMEACTIVE;
           NW< HAS NEW WW

NOACTIVE: MAR< WWLOC;
           L< SAD;
           MD< NW< WW, TASK;

INTZ: NW< L, :START;

SOMEACTIVE: MAR< PLC< LOC;
           STORE PC AND SET UP TO FIND HIGHEST PRIORITY REQUEST
           XREG< L, L< 0;
           MD< PC, TASK;

ILPA: PC< L;

ILP: T< SAD;
       L< T< XREG AND T;
       SH<0, L< T, T< PC;
       :IEXIT, XREG< L LSH 1;

NIEXIT: L< 0+T+1, TASK, :ILPA;

IEXIT: MAR< PLC< LOC+T+1;
       FETCH NEW PC. T HAS CHANNEL #, L HAS MASK
           XREG< L;
           T< XREG;
           L< NW< XOR T; TURN OFF BIT IN WW FOR INTERRUPT ABOUT TO HAPPEN
           T< MD;
           NW< L, L< T;
           PC< L, L< T< 0+1, TASK;
           SAD< L MRS< L, :NOACTIVE;
           SAD< 1B5 TO DISABLE INTERRUPTS
***************
* BIT-BLT - 61024 *
***************

Modified September 1977 to support Alternate memory banks
Last modified Sept 6, 1977 by Dan Ingalls

/* NOVA REGS
AC2 -> BLT DESCRIPTOR TABLE, AND IS PRESERVED
AC1 CARRIERS LINE COUNT FOR RESUMING AFTER AN
INTERRUPT. MUST BE 0 AT INITIAL CALL
AC0 AND AC3 ARE SMASHED TO SAVE S-REGS

/* ALTO REGISTER USAGE
;DISP CARRIES: TOPLD(100), SOURCEBANK(40), DESTBANK(20),
; SOURCE(14), OP(3)
$MASK1 $R0;
$YMUL $R2;  HAS TO BE AN R-REG FOR SHIFTS
$RETN $R2;
$SKEW $R3;
$TEMP $R5;
$WIDTH $R7;
$PLIER $R7;  HAS TO BE AN R-REG FOR SHIFTS
$DESTY $R10;
$WORD2 $R10;
$STARTBITSM1 $R35;
$SWA $R36;
$DESTX $R36;
$LREG $R40;  HAS TO BE R40 (COPY OF L-REG)
$NLINES $R41;
$RAST1 $R42;
$SRCX $R43;
$SKMSK $R43;
$SRCY $R44;
$RAST2 $R44;
$SCONST $R46;
$TWICE $R46;
$HCNT $R46;
$VINC $R46;
$HINC $R47;
$NWORDS $R50;
$SMASK2 $R51;  WAS $R40;

$LASTMASK1 $500;  MASKTABLE+021
$170000 $170000;
$CALL3 $3;  SUBROUTINE CALL INDICES
$CALL4 $4;
$DWAOFF $2;  BLT TABLE OFFSETS
$DXOFF $4;
$DWOFF $6;
$DHOFF $7;
$SWAOFF $10;
$SXOFF $12;
$GRAYOFF $14;  GRAY IN WORDS 14-17
$LASTMASK $477;  MASKTABLE+020 **NOT IN EARLIER PROMS!
BITBLT SETUP - CALCULATE RAM STATE FROM AC2'S TABLE

/* FETCH COORDINATES FROM TABLE */
1.2, FDBX, BLITX;
1.2, FDBL, BBNORAM;
17,20, FDBX, ..., FDX, FDW, ..., FSX, ...:
FDBL RETURNS (BASED ON OFFSET)

BITBLT: L= 0:
SINK+ LREG, BUSODD; SINK+ -1 IFF NO RAM
L= T= DWOFF, :FDBL;
BBNORAM: TASK, :NPTRAP;
TRAP IF NO RAM

FDB: T= MD;
WIDTH= L, L= T, TASK, :NZWID;
NZWID: NLINES= L;
T= AC1;
L= NLINES-T;
NLINES= L, SH<0, TASK;
:FDX;

FDBX: L= T= DWOFF, :FDBL;
PICK UP DEST X AND Y
FDX: T= MD;
DESTX= L, L= T, TASK;
DESTY= L;

FSX: T= MD;
SRCX= L, L= T, TASK;
SRCY= L, :CSHI;

/* FETCH DOUBLEWORD FROM TABLE (L= T= OFFSET, :FDBL) */
FDBL: MAR= AC2+T;
SINK= LREG, BUS;

FDBX: L= MD, :FDBX;

/* CALCULATE SKEW AND HINC */
1.2, LTOR, RTOL;

CSHI: T= DESTX;
L= SRCX-T-1;
T= LREG+1, SH<0;
TEST HORIZONTAL DIRECTION
L= 17.T, :LTOR; SKEW= (SRCX - DESTX) MOD 16

RTOL: SKEW= L, L= 0-1, :AH, TASK; HINC = -1
LTOR: SKEW= L, L= 0+1, :AH, TASK; HINC = 1
AH: HINC= L;

CALCULATE MASK1 AND MASK2
1.2, IFRTOL, LNWORDS;
1.2, POSWID, NEGWID;

CMASKS: T= DESTX;
T= 17.T;
MAR= LASTMASKP1-T-1;
L= 17-T;
STARTBITS = 16 - (DESTX.17)
STARTBISM1= L;
L= MD, TASK;
MASK1= L;
MASK1 = @ (MASKLOC + STARTBITS)
L= WIDTH-1;
T= LREG-1, SH<0;
T= DESTX+T+1, :POSWID;

POSWID: T= 17.T;
MAR= LASTMASK-T-1;
T= ALLONES;
MASK2 = NOT
L= HINC-1;
L= MD XOR T, SH<0, TASK; @ (MASKLOC + ((DESTX + WIDTH-1).17))
MASK2= L, :IFRTOL;
/* IF RIGHT TO LEFT, ADD WIDTH TO X'S AND EXCH MASK1, MASK2 */

IFRTOL: T= WIDTH-1;
L= SRCX+T;
SRCX= L;
L= DESTX+T;
DESTX= L;
DESTX = DESTX + (WIDTH-1)
T= DESTX;
L= 17.T, TASK;
STARTBISM1= L; STARTBITS = (DESTX.17) + 1
T= MASK1;
L = MASK2;
MASK1 = L, L = T, TASK; EXCHANGE MASK1 AND MASK2
MASK2 = L;

/* CALCULATE NWORDS*/
11.2, LNWI, THIN;
LNWORDS = T + STARTBITSM1 + 1;
L = WIDTH - T - 1;
T = 177700, SHC0;
T = RREG.T, LNWI;
LNWI;
L = CALL4;
NWORDS = (WIDTH - STARTBITS) / 16
CYRET = L, L = T, R4, TASK; CYRET = CALL4

**WIDTH REG NOW FREE**
CYX4:
L = CYCOUT, LNWI;
THIN:
T = MASK1; SPECIAL CASE OF THIN SLICE
L = MASK2.T;
L = MASK1.L, L = 0 - 1;
MASK1 = MASK1.MASK2, NWORDS = - 1
LNWI:
L = CALL5;
LOAD NWORDS
**STARTBITSM1 REG NOW FREE**

/* DETERMINE VERTICAL DIRECTION*/
11.2, BTO1, TTOB;
T = SRCY;
L = DESTY - T;
T = NLINES - 1, SHC0;
L = 0, T = BTOT;
VINC = 0 IFF TOP-TO-BOTTOM
BTOT:
L = ALLONES; ELSE - 1
BTOT1:
VINC = L;
L = SRCY + T;
GOING BOTTOM TO TOP
SRCY = L; ADD NLINES TO STARTING Y'S
L = DESTY + T;
DESTY = L, L = 0 + 1, TASK;
TWICE = L, : CWA;

TTOB:
T = AC1, : TTOB1; TOP TO BOT, ADD NDONE TO STARTING Y'S
**AC1 REG NOW FREE**

/* CALCULATE WORD ADDRESSES - DO ONCE FOR SWA, THEN FOR DWAX*/
CWA:
L = SRCY; Y HAS TO GO INTO AN R-REG FOR SHIFTING
VMUL = L;
T = SWAOFF; FIRST TIME IS FOR SWA, SRCX
L = SRCH;
**SRCX, SRCY REG NOW FREE**
DOBWA:
MAR = AC2 + T, FETCH BITMAP ADDR AND RASTER
XREG = L;
L = CALL3;
CYRET = L; CYRET = CALL3
L = MD;
T = MD;
DWAX = L, L = T, TASK;
RST2 = L;
T = 177700;
L = T = XREG.T, : R4, TASK; SWA = SWA + SRCX / 16
CYX3:
T = CYCOUT;
L = DWAX + T;
DWAX = L;

11.2, NOADD, DOADD;
11.2, MULP, CDLT;
SWA = SWA + SRCY * RAST1
L = RAST2;
SINK = VMUL, BUS = 0, TASK; NO MUL IF STARTING Y = 0
PLIER = L, MULP;
MULP:
L = PLIER, BUSDD;
PLIER = L RSH 1, : NOADD;
NOADD:
L = VMUL, SH = 0, TASK;
TEST NO MORE MULTIPLIER BITS
SHIFTB:
VMUL = L LSH 1, : MULP
DOADD:
T = VMUL;
L = DWAX + T;
DWAX = L, L = T, : SHFTB, TASK;
**PLIER, VMUL REG NOW FREE**

11.2, HNEG, HPOS;
11.2, VPOS, VNEG;
11.1, CD0;
CALCULATE DELTAS = -(NWORDS + 2)[HINC] + RASTER[VINC]
CDLT:
L = T + HINC - 1; (NOTE T = - 2 OR 0)
L = T + NWORDS - T, SH = 0; (L = NWORDS + 2 OR T = NWORDS)
CD1:  SINK+ VINC, BUSODD, :HNEG;
HNEG:  T= RAST2, :VPOS;
HPOS:  L= -2-T, :CD1;  (MAKES L=-(NWORDS+2))
VPOS:  L= LREG+T, :GDelt, TASK;  BY NOW, LREG = +(NWORDS+2)
VNEG:  L= LREG-T, :GDelt, TASK;  AND T= RASTER
GDelt:  RAST2= L;

;  /* END WORD ADDR LOOP
!1,2,ONEMORE,CTOPL;
  L= TWICE-1;
  TWICE= L, SHO;
  L= RAST2, :ONEMORE;  USE RAST2 2ND TIME THRU
ONEMORE:  RAST1= L;
  L= DESTY, TASK; USE DESTY 2ND TIME THRU
  YMUL= L;
  L= DWAX;  USE DWAX 2ND TIME THRU
  T= DESTX;  CAREFUL - DESTX=SWA!!
SWA= L, L= T;  USE DESTX 2ND TIME THRU
  T= DWAOFF, :DOSWA;  AND DO IT AGAIN FOR DWAX, DESTX
;  **TWICE, VINC REGS NOW FREE**
;
;  /* CALCULATE TOPLD
!1,2,CTOPL,CSKEW;
!1,2,HM1,H1;
!1,2,NOTOPL,TOPL;
CTOPL:  L= SKEW, BUS=0, TASK;  IF SKEW=0 THEN 0, ELSE
CTX:   IR= 0, :CTOPL;
CTOPL1: T= SRCH;  (SKEW GR SRCX.17) XOR (HINC EQ 0)
  L= HINC-1;
  T= 17.T, SH=0;  TEST HINC
  L= SKEW-T-1, :HM1;
HM1:   T= LNEG;  IF HINC=-1, THEN FLIP
  L= 0-T-1, :H1;  THE POLARITY OF THE TEST
NOTOPL: SINK= HINC, BUSODD, TASK, :CTX;  HINC FORCES BUSODD
TOPL:  SWA= L, TASK;
  (DISP = 100 FOR TOPLD)
;  **HINC REG NOW FREE**

;  /* CALCULATE SKEW MASK
!1,2,THINC,BCOM1;
!1,2,COMSK,NOCOM;
CSKEW:  T= SKEW, BUS=0; IF SKEW=0, THEN COMP
MAR= LASTMASK1-T-1, :THINC;
THINC: L=HINC-1;
        SH=0;
        IF HINC=-1, THEN COMP
BCOM1:  T= ALLONES, :COMSK;
COMSK:  L= MD XOR T, :GFM;
NOCOM:  L= MD, :GFM;
;
;  /* GET FUNCTION
GFM:    MAR= AC2;
    SKMSK= L;
    T= MD;
    L= DISP+T, TASK;
    IR+ LREG, :BENTR;
    DISP = DISP .OR. FUNCTION
BITBLT WORK - VERT AND HORIZ LOOPS WITH 4 SOURCES, 4 FUNCTIONS

/* VERTICAL LOOP: UPDATE SWA, DWAX */
!1.2.DDO0.VLOOP;
VLOOP:  T= SWA;
        L= RASI+T;  INC SWA BY DELTA
        SWA= L;
        T= DWAX;
        L= RAST2+T, TASK;  INC DWAX BY DELTA
        DWAX= L;

/* TEST FOR DONE, OR NEED GRAY */
!1.2.MOREV,DONEV;
!1.2.BMAYBE,BNOINT;
!1.2.BDOINT, BDISO;
!1.2.DOGRAY, NOGRAY;
BENTR:  L= T= Nelines-1;  DECR NINES AND CHECK IF DONE
        Nines= L, SHO;
        L= NWW, BUS=0, :MOREV;  CHECK FOR INTERRUPTS
        MOREV= L= 3.T, :BMAYBE, SHO;  CHECK DISABLED ***V3 change
        BNOINT: SINK+ DISP, SINK+ 1gm10, BUS=0, :BDISO, TASK;
        BMAYBE: SINK+ DISP, SINK+ 1gm10, BUS=0, :BDOINT, TASK;  TEST IF NEED GRAY(FUNC=8,12)
        BDISO: CONST= L, :DOGAY;  ***V3 change

/* INTERRUPT SUSPENSION (POSSIBLY) */
!1.1.DOII:  MAY GET AN OR-1
BDOINT:  :DOII;  TASK HERE
DOII:  T= AC2;
        MAR= DHOFF+T;  NINES DONE = HT-NINES-1
        T= NINES;
        L= PC-1;  BACK UP THE PC, SO WE GET RESTARTED
        PC= L;
        L= MD-T-1, :BLITX, TASK;  ...WITH NO LINES DONE IN AC1

/* LOAD GRAY FOR THIS LINE (IF FUNCTION NEEDS IT) */
!1.2.PRELD,NOPLD;
DOGRAY:  T= CONST-1;
        T= GRAYOFF+T+1;
        MAR= AC2+T;
        NOP;  UGH
        L= MD;
NOGRAY:  SINK+ DISP, SINK+ 1gm100, BUS=0, TASK;  TEST TOPLD
        CONST= L, :PRELD;

/* NORMAL COMPLETION */
NEGWI:  L= 0, :BLITX, TASK;
DONEV:  L= 0, :BLITX, TASK;  MAY BE AN OR-1 HERE!
BLITX:  AC1= L, :FINBLIT;

/* PRELOAD OF FIRST SOURCE WORD (DEPENDING ON ALIGNMENT) */
!1.2.AB1.NB1;
PRELD:  SINK+ DISP, SINK+ 1gm40, BUS=0;  WHICH BANK
        T= HINC, :AB1;
NB1:  MAR= SWA-T, :XB1;  (NORMAL BANK)
AB1:  XMAR= SWA-T, :XB1;  (ALTERNATE BANK)
XB1:  NOP;
        L= MD, TASK;
        WORD2+ L, :NOPLD;

/* HORIZONTAL LOOP - 3 CALLS FOR 1ST, MIDDLE AND LAST WORDS */
!1.2.FDISP.LASTH;
%17.17,14, DON0, DON2, DON3;  CALLERS OF HORIZ LOOP
NOTE THIS IGNORES 14-BITS, SO 1gm14 WORKS LIKE L=0 FOR RETN
!14.1.LHI;  IGNORE RESULTING BUS
NOPLD:  L= 3, :FDISP;  CALL #3 IS FIRST WORD
DON3:  L= NWORDS;
        HCNT= L, SHO;  HCNT COUNTS WHOLE WORDS
DON0:  L= HCNT-1, :DOO;  IF NEG, THEN NO MIDDLE OR LAST
DOO:  HCNT= L, SHO;  CALL #0 (OR-141) IS MIDDLE WORDS
        UGLY HACK SQUEEZES 2 INSTRS OUT OF INNER LOOP:
        L= DISP, SINK+ 1gm14, BUS, TASK, :FDISP;  (WORKS LIKE L=0)
LASTH:  :LHI;  TASK AND BUS PENDING
LHI:  L= 2, :FDISP;  CALL #2 IS LAST WORD
DON2:  :VLOOP;

/* HERE ARE THE SOURCE FUNCTIONS

17,20,...,F0,...,F1,...,F2,...,F3;  IGNORE OP BITS IN FUNCTION CODE
17,20,...,FOA,...,F1A,...,F2A,...;

13,4,OP0,OP1,OP2,OP3;
11,2,AB2,NB2;

FDISP:  SINK= DISP, SINK+1gm14, BUS, TASK;
FDISPA:  RETN= L :FO;
F0:  SINK= DISP, SINK+1gm40, BUS=0, :WIND;  FUNC 0 - WINDOW
F1:  SINK= DISP, SINK+1gm40, BUS=0, :WIND;  FUNC 1 - NOT WINDOW
F1A:  T= CYCOUT;
     L= ALLOWES XOR T, TASK, :F3A;
F2:  SINK= DISP, SINK+1gm40, BUS=0, :WIND;  FUNC 2 - WINDOW .AND. GRAY
F2A:  T= CYCOUT;
     L= ALLOWES XOR T;
     SINK= DISP, SINK+1gm20, BUS=0;  WHICH BANK
     TEMP= L :AB2;  TEMP = NOT WINDOW
NB2:  MAR= DWAX, :XB2;  (NORMAL BANK)
AB2:  XMAR= DWAX, :XB2;  (ALTERNATE BANK)
XB2:  L= CONST AND T;  WINDOW .AND. GRAY
T= TEMP;
T= MD .T;  DEST .AND. NOT WINDOW
L= LREG OR T, TASK, :F3A;  (TRANSPARENT)
F3:  L= CONST, TASK, :F3A;  FUNC 3 - CONSTANT (COLOR)

/*/ AFTER GETTING SOURCE, START MEMORY AND DISPATCH ON OP

11,2,AB3,NB3;
F3A:  CYCOUT= L;  (TASK HERE)
FOA:  SINK= DISP, SINK+1gm20, BUS=0;  WHICH BANK
     SINK= DISP, SINK+1gm3, BUS:AB3;  DISPATCH ON OP
NB3:  T= MAR= DWAX, :OP0;  (NORMAL BANK)
AB3:  T= XMAR= DWAX, :OP0;  (ALTERNATE BANK)

/*/ HERE ARE THE OPERATIONS - ENTER WITH SOURCE IN CYCOUT
X16,17,15,STFULL,STMSK;  MASKED OR FULL STORE (LOOK AT 2-BIT)

OP0:  SINK= RETN, BUS;  TEST IF UNMASKED
OPOA:  L= HINC+T, :STFULL;  ELSE:STMSK
OP1:  T= CYCOUT;  OP 1 - SOURCE .OR. DEST
OP2:  T= CYCOUT;  OP 2 - SOURCE .XOR. DEST
L= MD OR T, :OPN;
OP3:  T= CYCOUT;
     L= 0-T-1;
     T= LREG;
     L= MD AND T, :OPN;
OPH:  SINK= DISP, SINK+1gm20, BUS=0, TASK;  WHICH BANK
     CYCOUT= L, :AB3;

/*/ STORE MASKED INTO DESTINATION
11,2,STM2,STM1;
11,2,AB4,NB4;
STMSK:  L= MD;
     SINK= RETN, BUSODD, TASK;  DETERMINE MASK FROM CALL INDEX
     TEMP= L :STM2;  STACHE DEST WORD IN TEMP
STM1:  T=MASK1, :STM3;
STM2:  T=MASK2, :STM3;
STM3:  L= CYCOUT AND T;  ***X24. Removed TASK clause.
     CYCOUT= L, L= 0-T-1;  AND INTO SOURCE
     T= LREG;
     T= TEMP .T;  AND INTO DEST
L= CYCOUT OR T;  OR TOGETHER THEN GO STORE
SINK= DISP, SINK+1gm20, BUS=0, TASK;  WHICH BANK
     CYCOUT= L, :AB4;
NB4:  T= MAR= DWAX, :OP0A;  (NORMAL BANK)
AB4:  T= XMAR= DWAX, :OP0A;  (ALTERNATE BANK)

/*/ STORE UNMASKED FROM CYCOUT (L=NEXT DWAX)
STFULL:  MD+ CYCOUT;
STFULL1:  SINK= RETN, BUS, TASK;
DWAX+ L, :DON0;
/* WINDOW SOURCE FUNCTION

TASKS UPON RETURN, RESULT IN CYCOUT
11,2, DOCY, NOCY;
17,1, WIA;
11,2, NZSK, ZESK;
11,2, AB6, NB6;

WIND: L= T= SKMSK, :AB5; ENTER HERE (8 INST TO TASK)
NB6: MAR= SWA, :XB6; (NORMAL BANK)
AB6: XMAR= SWA, :XB6; (ALTERNATE BANK)
XB6: L= WORD2.T, SH=0;
CYCOUT+= L, L= 0-T-1, :NZSK; CYCOUT+= OLD WORD .AND. MSK
ZESK: L= M6, TASK; ZERO SKEW BYPASSES LOTS
CYCOUT+= L, :NOCY;
NZSK: T= MD;
L= LREG.T;
TEMP= L, L=T, TASK; TEMP= NEW WORD .AND. NOTMSK
WORD2= L;
T= TEMP;
L= T= CYCOUT OR T; OR THEM TOGETHER
CYCOUT+= L, L= 0+1, SH=0; DON'T CYCLE A ZERO ***X21.
SINK= SWK, BUS, :DOCY;
DOCY: CYRET= L LSH 1, L= T, :L0; CYCLE BY SKEW ***X21.
NOCY: T= SWA, :WIA; (MAY HAVE OR-17 FROM BUS)
CYX2: T= SWA;
WIA: L= HINC+T;
SINK= DISP, SINK= lgm14, BUS, TASK; DISPATCH TO CALLER
SWA= L, :F0A;
THE DISK CONTROLLER

ITS REGISTERS:
$DCBR  $R34;
$KNMAR  $R33;
$CKSUMR  $R32;
$KDCT  $R31;
$KNMARC  $R33;
$CKSUMRC  $R32;
$KDCTC  $R31;

ITS TASK SPECIFIC FUNCTIONS AND BUS SOURCES:
$KSTAT  $L020012,014003,124100;  DF1 = 12 (LHS) BS = 3 (RHS)
$KRC  $L024011,000000,000000;  NDF2 = 11
$KRCN  $L024012,000000,000000;  NDF2 = 12
$INIT  $L024010,000000,000000;  NDF2 = 10
$CLRSTAT  $L018014,000000,000000;  NDF1 = 14
$KCOMM  $L020016,000000,124000;  DF1 = 15 (LHS only) Requires bus def
$SWRN  $L024014,000000,000000;  NDF2 = 14
$KDATA  $L020017,014004,124100;  DF1 = 17 (LHS) BS = 4 (RHS)
$STOP  $L016011,000000,000000;  NDF1 = 11
$NFER  $L024015,000000,000000;  NDF2 = 15
$STOPB  $L024016,000000,000000;  NDF2 = 16
$XFBD  $L024013,000000,000000;  NDF2 = 13
$INCREC  $L016013,000000,000000;  NDF1 = 13

THE DISK CONTROLLER COMES IN TWO PARTS. THE SECTOR
TASK HANDLES DEVICE CONTROL AND COMMAND UNDERSTANDING
AND STATUS REPORTING AND THE LIKE. THE WORD TASK ONLY
RUNS AFTER BEING ENABLED BY THE SECTOR TASK AND
ACTUALLY MOVES DATA WORDS TO AND FROM THE DISK.

THE SECTOR TASK

LABEL PREDEFINITIONS:
!1,2,COMM,NOCOMM;
!1,2,COMM2,_IDLE;
!1,2,BADCOMM,COMM3;
!1,2,COMM4,ILSEC;
!1,2,COMMS,WHYNRDY;
!1,2,STROB,CSECT;
!1,2,STALL,CSECT1;
!1,2,KSFINI,CSECT2;
!1,2,IDLEZ,TRANSFER;
!1,2,STALL2,GASP;
!1,2,INVERT,NOINVERT;

KSEC:  MAR+ KBLKADR2;
KPOQ:  CLRSTAT;  RESET THE STORED DISK ADDRESS
MD+L=ALLONES+1, .GCOMZ;  ALSO CLEAR DCB POINTER
GETCOM:  MAR+KBLKADR;  GET FIRST DCB POINTER
GCOM1:  NOP;
L=MD;
GCOM2:  DCBR=L,TASK;
KCOM=TOWT;  IDLE ALL DATA TRANSFERS
MAR+KBLKADR3:  GENERATE A SECTOR INTERRUPT
T+WWM;
L=MD OR T;
MAR+KBLKADR+1;  STORE THE STATUS
WWM+L, TASK;
MD+KSTAT;
MAR+KBLKADR;  WRITE THE CURRENT DCB POINTER
KSTAT=5;  INITIAL STATUS IS INCOMPLETE.
L=DCBR,TASK,BUS=0;
MD=DCBR, .COMM;

BUS=0 MAPS COMM TO NOCOMM

COMM:  T+2;  GET THE DISK COMMAND
MAR+DCBR+T;

T=TOTUNC;
L=MD XOR T, TASK, STROBON;
KDCT=L, :COMM2;

; STROBON MAPS COMM2 TO IDLE1

COMM2: T=10; READ NEW DISK ADDRESS
MAR=DCBR+T+1;
T=KDCT;
L=ONE AND T;
L=400 AND T, SH=0;
T=MD, SH=0, :INVERT;

; SH=0 MAPS INVERT TO NOINVERT

INVERT: L=2 XOR T, TASK, :BADCOMM;
NOINVERT: L=T, TASK, :BADCOMM;

; SH=0 MAPS BADCOMM TO COMM3

COMM3: KNMAR=L;

MAR=KBLKAD2; WRITE THE NEW DISK ADDRESS
T=SECT2CM; CHECK FOR SECTOR > 13
L=T+KDATA+KNMAR+T; NEW DISK ADDRESS TO HARDWARE
KDCT=KDCT, ALUCY; DISK COMMAND TO HARDWARE
L=MD XOR T, TASK, :COMM4; COMPARE OLD AND NEW DISK ADDRESSES

; ALUCY MAPS COMM4 TO ILLSEC

COMM4: CKSUMR=L;

MAR=KBLKAD2; WRITE THE NEW DISK ADDRESS
T=CADM, SWNRDY; SEE IF DISK IS READY
L=CKSUMR AND T, :COMM5;

; SWNRDY MAPS COMM5 TO WHNYRDY

COMM5: MD=KNMAR; COMPLETE THE WRITE
SH=0, TASK;
; STROB;

; SH=0 MAPS STROB TO CKSECT

CKSECT: T=KNMAR, NFER;
L=KSTAT XOR T, :STALL;

; NFER MAPS STALL TO CKSECT1

CKSECT1: CKSUMR=L, XFRDAT;
T=CKSUMR, :KSFINI;

; XFRDAT MAPS KSFINI TO CKSECT2

CKSECT2: L=SECTMSK AND T;
KSLAST: BLOCK, SH=0;
GASP: TASK, :IDLE2;

; SH=0 MAPS IDLE2 TO TRANSFER

TRANSFER: KCOMM=TOTUWC; TURN ON THE TRANSFER

11,2, ERRFND, NOERRFND;
11,2, EF1, NEF1;

DMPSTAT: T=COMERR1; SEE IF STATUS REPRESENTS ERROR
L=KSTAT AND T;
MAR=DCBR+1; WRITE FINAL STATUS
KDCT=L, TASK, SH=0;
MD=KSTAT, :ERRFND;

; SH=0 MAPS ERRFND TO NOERRFND

NOERRFND: T=6; PICK UP NO-ERROR INTERRUPT WORD

INTCOM: MAR=DCBR+T;
T+NNW;
L+MD OR T;
SINK+KWDCT,BUS=0,TASK;
NNW=L,:EFI;

; BUS=0 MAPS EFI TO NEF1
NEF1: MAR=DCBR,.6COM1; FETCH ADDRESS OF NEXT CONTROL BLOCK
ERRFND: T+7,:INTCOM; PICK UP ERROR INTERRUPT WORD
EF1: :KSEC;
NOCOMM: L+ALLONES,CLRSTAT,:KSLAST;
IDLE1: L+ALLONES,:KSLAST;
IDLE2: KSTAT+LOW14,:GETCOM; NO ACTIVITY THIS SECTOR
BADCOMM: KSTAT+7; ILLEGAL COMMAND ONLY NOTED IN KBLK STAT BLOCK;
          TASK,:EF1;
WHYNRDY: NFER;
STALL: BLOCK,:STALL2;
; NFER MAPS STALL2 TO GASP
STALL2: TASK;
        :DMPSTAT;
ILLSEC: KSTAT+7,:STALL; ILLEGAL SECTOR SPECIFIED
STROB: CLRSTAT;
       L+ALLONES,STROBE,:CKSECT1;
KSFNI: KSTAT+4,:STALL; COMMAND FINISHED CORRECTLY
:DISK WORD TASK
:WORD TASK PREDEFINITIONS
:37,37,,RPO,INIPREF,CMP,RPO,,PXFLP,RDCCK0,WRT0,REC1,,REC2,REC3,,RECORC,RECO,RO,,CK0,WO,,R2,,W2,,RE
*CO,,KWD:
:11,2,RW1,RW2;
:11,2,CK1,CK2;
:11,2,CK3,CK4;
:11,2,CKERR,CK5;
:11,2,PXFLP,PXF2;
:11,2,PREFDONE,INIPREF;
:11,2,CK8;
:11,2,CKSMERR,PXFLP0;

KWD: BLOCK,:RECO;

; SH<0 MAPS RECO TO RECO
; ANYTHING=INIT MAPS RECO TO KWD

RECO: L+2, TASK; LENGTH OF RECORD 0 (ALLOW RELEASE IF BLOCKED)
KNMARW=L;

T=KNMARW, BLOCK, RWC; GET ADDR OF MEMORY BLOCK TO TRANSFER
MAR=DCBR+T+1,:RECORC;

; WRITE MAPS RECORC TO RECOW
; INIT MAPS RECORC TO KWD

RECORC: T=MIRRDL,BLOCK,:RECL2A; FIRST RECORD READ DELAY
RECOW: T=MIRRBL,BLOCK,:RECL2A; FIRST RECORD 0'S BLOCK LENGTH

REC1: L=10, INCRECN0; LENGTH OF RECORD 1
T=4,:RECL2;

REC2: L=PAGE1; INCRECN0; LENGTH OF RECORD 2
T=5,:RECL2;

RECL2: MAR=DCBR+T, RWC; MEM BLK ADDR FOR RECORD
KNMARW=L,:RDCCK0;

; RWC=WRITE MAPS RDCCK0 INTO WRT0
; RWC=INIT MAPS RDCCK0 INTO KWD

RDCCK0: T=MIRRDL,:RECL2A;
WRT0: T=MIRRBL,:RECL2A;

RECL2A: L=MD;
KWDCTW=L, T;
COM1: KCOMM=STWNC,:INIPREF0;

INIPREF: L=CKSUMRW+1, INIT, BLOCK;
INIPREF0: CKSUMRW=L, SH<0, TASK,:INIPREF1;

; INIT MAPS INIPREF1 TO KWD

INIPREF1: KDATA=0,:PREFDONE;

; SH<0 MAPS PREFDONE TO INIPREF

PREFDONE: T=KNMARW; COMPUTE TOP OF BLOCK TO TRANSFER
KWDX: L=KWDCTW+T,RWC; (ALSO USED FOR RESET)
KNMARW=L,BLOCK,:RPO;

; RWC=CHECK MAPS RPO TO CMP
; RWC=WRITE MAPS RPO AND CMP TO WPO
; RWC=INIT MAPS RPO, CMP, AND WPO TO KWD

RPO: KCOMM=STRCWFS,:WP1;

CKPO: L=KWDCTW-1; ADJUST FINISHING CONDITION BY 1 FOR CHECKING ONLY
KWDCTW=L,:RPO;

WPO: KDATA=ONE; WRITE THE SYNC PATTERN
WP1: L=KBLKADR,TASK,:RW1; INITIALIZE THE CHECKSUM AND ENTER XFER LOOP

XFLP: T=L=KNMARW-1; BEGINNING OF MAIN XFER LOOP
KNMARW=L;
MAR=KNMAR,WRC; L=KWDCTW-T,:RO;
; RWC=CHECK MAPS RO TO CK0
; RWC=WRITE MAPS RO AND CK0 TO W0
; RWC=INIT MAPS RO, CK0, AND W0 TO KWD
R0: T=CKSUMRW,SH=0,BLOCK;
MD=L+KDATA XOR T,TASK,:RW1;
; SH=0 MAPS RW1 TO RW2
   RW1: CKSUMRW=L,:XFLP;
   W0: T=CKSUMRW,BLOCK;
   KDATA=L+MD XOR T,SH=0;
   TASK,:RW1;
; AS ALREADY NOTED, SH=0 MAPS RW1 TO RW2
CK0: T=KDATA,BLOCK,SH=0;
     L+MD XOR T,BUS=0,:CK1;
; SH=0 MAPS CK1 TO CK2
CK1: L=CKSUMRW XOR T,SH=0,:CK3;
; BUS=0 MAPS CK3 TO CK4
CK3: TASK,:CKERR;
; SH=0 MAPS CKERR TO CK5
CK5: CKSUMRW=L,:XFLP;
CK4: MAR=KNMARW,:CK6;
; SH=0 MAPS CK6 TO CK6
CK6: CKSUMRW=L+0+T;
     MTEMP=L,TASK;
     MD=MTEMP,:XFLP;
CK2: L=CKSUMRW-T,:R2;
; BUS=0 MAPS R2 TO R2
   RK2: CKSUMRW=L;
      T=KDATA+CKSUMRW,RWC; THIS CODE HANDLES THE FINAL CHECKSUM
      L=KDATA-T,BLOCK,:R2;
; RWC=CHECK NEVER GETS HERE
; RWC=WRITE MAPS R2 TO W2
; RWC=INIT MAPS R2 AND W2 TO KWD
R2: L=MRPAL, SH=0; SET READ POSTAMBLE LENGTH, CHECK CKSUM
KCOMM=TOTUWC,:CKSMERR;
; SH=0 MAPS CKSMERR TO PXFLPO
W2: L=MWPAL, TASK; SET WRITE POSTAMBLE LENGTH
CKSUMRW=L,:PXFLP;
CKSMERR: KSTAT=0,:PXFLPO; 0 MEANS CHECKSUM ERROR, CONTINUE
PXFLP: L=CKSUMRW+1, INIT, BLOCK;
PXFLPO: CKSUMRW=L, TASK, SH=0,:PXFLP1;
; INIT MAPS PXFLP1 TO KWD
PXFLP1: KDATA=0,:PXFLP;
; SH=0 MAPS PXFLP TO PXF2
PXF2: RECNO, BLOCK; DISPATCH BASED ON RECORD NUMBER
:REC1;

; RECN0=2 MAPS REC1 INTO REC2
; RECN0=3 MAPS REC1 INTO REC3
; RECN0=INIT MAPS REC1 INTO KWD

REC3:  KSTAT=4,:PXFLP; 4 MEANS SUCCESS!!!

CKERR:  KCOMM=TOTUNC; TURN OFF DATA TRANSFER
        L+KSTAT+6,:PXFLP1; SHOW CHECK ERROR AND LOOP
; The Parity Error Task
; Its label predefined is way earlier
; It dumps the following interesting registers:
; 014/ DCBR  Disk control block
; 016/ KNMAR  Disk memory address
; 016/ DWA  Display memory address
; 017/ CBA  Display control block
; 020/ PC  Emulator program counter
; 021/ SAD  Emulator temporary register for indirection

PART:  T = 10;
        L = ALLOWES;
        MAR = ERRCTRL, :PX1;
PR8:  L = SAD, :PX;
PR7:  L = PC, :PX;
PR6:  L = CBA, :PX;
PR5:  L = DWA, :PX;
PR4:  L = KNMAR, :PX;
PR3:  L = DCBR, :PX;
PR2:  L = NWW OR T, TASK;  T CONTAINS 1 AT THIS POINT
PR0:  NWW = L, :PART;
PX:  MAR = 612+T;
PX1:  MTEMP = L, L = T;
MD = MTEMP;
CURDATA = L;  THIS CLOBBERS THE CURSOR FOR ONE
T = CURDATA-1, BUS;  FRAME WHEN AN ERROR OCCURS
:PR0;