August 12, 1985

Augment Engine Console Reference Manual

V2.6

-CONTENTS-

OVERVIEW

The console program is an interactive front-panel for the Augment Engine. It provides the engineer with access to all internal registers, memories and flags as well as giving an operator simple access to booting and diagnostic procedures. It serves as the CTY once a system is in full operation.

An IBM-PC runs the console program. A relatively simple hardware interface connects to the engine. This reflects the design goal of keeping the complex aspects of the engine in the software, which is more cost-effective to maintain and distribute than hardware.

Major parameters of the console are kept in a small text file, CONSOLE.INI. This allows simple on-site customization. More customization is supported by the console's ability to expand command macros and process command files.

It is possible to access a console from a remote location via a telephone modem. Three levels of security protect a system from unauthorized remote entry. A remote console is completely similar to the local console it is connected to. This provides a means by which engineers may evaluate and correct problems without having to travel to a customer site.

The remote configuration also allows for communication between the local operator and the remote engineer. These talk sessions can be recorded in the log file for a permanent record. All other commands can be logged as well.

The documentation for the console consists of this manual, a help file, and the program source files. This manual contains general information concerning the installation, use and construction of the console. The help file contains detailed information about each command, function and diagnostic test. The source files are the final arbitrator for any question on "how it works" and are appropriately commented.

Interactively, it is possible to lookup and display topics from the help file. It is also possible to display other files, such as sources or command files. The console can display a quick reminder about any command and its parameters.

The console contains a set of diagnostic tests which will quickly verify the correct operation of the console, and to a certain extent, the engine. There are also various debugging aids built into the console code itself.

The screen display of the console is a dynamic reflection of the engine's registers and flags. The micro instruction register is parsed into individual fields. The screen display can be recorded on the printer for a permanent record.

Booting and loading operations are implemented in their full generality. Standard command macros and command files hide advanced parameters from the operator.

Micro code development and debugging is simplified by the starting and stopping commands. Break points may be set at both the macro and micro code locations. Single stepping and clock stepping are provided and a detailed history analysis is available for the last 1000 machine cycles executed.

Resets are provided for the console program, engine hardware and system software. Deposits and examines are available for all engine registers and flags. In many cases there is more than one way to access an item.

The console program is organized as a frontend command interpreter which makes calls on a backend function processor. The backend is the actual interface to the engine. In a remote configuration, a remote frontend makes calls on the local backend. Each backend function can be called interactively.

The console program is written entirely in TURBO-PASCAL. TURBO provides overlay management and the insertion of inline machine code. Important code modules are the interrupt driven I/O, the screen display manager, a constant table loader and the general purpose bit converter.

The files needed for execution are:

CONSOLE.COM -- main program, execute this from PC-DOS
CONSOLE.000, CONSOLE.001,
CONSOLE.002, CONSOLE.003 -- overlays
CONSOLE.INI -- configuration parameters
CONSOLE.HLP -- on-line documentation file
INSTRS.BIN, KLINSTRS.BIN -- special micro instructions
FAST.BIN, KLFAST.BIN -- fast loader
BOOT.BIN, KLBOOT.BIN -- system booter
DIAG.BIN, KLDIAG.BIN -- diagnostics
SYSTEM.BIN, KLSYSTEM.BIN -- system micro code

The source files are:

CONSOLE.PAS -- main source file, display manager
CONSOLE0.PAS -- utilities (bit converter)
CONSOLE1.PAS -- backend functions
CONSOLE2.PAS -- frontend commands
CONSOLE3.PAS -- table loader
CONSOLE.MAN -- text of console reference manual
INSTRS.MLD, KLINSTRS.MLD -- octal instructions
FAST.MLD, KLFAST.MLD -- octal fast loader code
FAST.SLO, KLFAST.SLO -- fast loader sources
BOOT.MLD, KLBOOT.MLD -- octal booter code
BOOT.SLO, KLBOOT.SLO -- booter sources
DIAG.MLD, KLDIAG.MLD -- octal diagnostics code
DIAG.SLO, KLDIAG.SLO -- diagnostic sources
SYSTEM.MLD, KLSYSTEM.MLD -- octal system micro code
SYSTEM.SLO, KLSYSTEM.SLO -- system micro code sources

SETUP

The standard console configuration consists of an IBM-PC with a special Augment Engine Interface adapter and an AST memory adapter. The AST board also provides a time-of-day clock and the CTY serial port. The CTY connection from the interface board to the AST board is internal.

The external connections are power, monitor, printer, modem and the Augment Engine Interface. These are explained in the PC documentation and training materials.

Never connect or disconnect the Interface when the PC is ON. When the PC os off, the Interface may be connected or disconnected without disturbing the Augment Engine.

There are at least three floppy disks need to operate a console. These are:

SYSTEM -- DOS/utilities, drive A

CONSOLE -- CONSOLE executables, micro code

LOG -- log file, must be in drive A

Other disks may be provided which contain documentation and source files.

The SYSTEM disk contains the PC-DOS and other utility programs, such as the side-kick and the ram-disk. The PC should be powered on, or reset, with the system disk in drive A.

The system will automaticaly set up a ram disk and side kick. When the PC is ready, replace the system disk by the console disk and use the INSTALL batch file to set up the console in the ram disk. Replace the console disk by the log disk. The console is ready for operation. Follow the directions displayed.

The LOG disk should be replaced periodically BEFORE it gets full. The procedure is to QUIT from the console program, exchange the old LOG disk with a new one, and return to the console. The console program will not start up if there is not enough room on the log disk.

When the console program is started from PC-DOS, it is possible to specify certain parameters on the command line. They are: REMOTE, to make a remote console, DEBUG, to turn the debug flag on initially, and SIMULATE, to interactively simulate the backend functions. In normal operations, only REMOTE will be used by an operator.

Additionally, the name of a command file may be specified on the PC-DOS command line. Two such files are BOOT.CMD, which boots up the whole system, and CTY.CMD which returns to the console CTY mode.

```
A>REM THIS IS THE CONSOLE SOURCE DISK

A>DIR

 Volume in drive A is FROB-SOURCE
 Directory of  A:\

 )DTU     CMD        72   8-01-85  10:43a
CONSOLE2 PAS     59285   8-01-85  10:25a
CONSOLE1 PAS     49952   8-01-85  10:18a
CONSOLE0 PAS     11744   8-01-85  10:12a
CONSOLE  HLP     18090   7-11-85   2:39p
UTOBJ    CMD        64   8-01-85  10:43a
MBOOT    CMD       118   8-01-85  10:43a
DBGO     CMD       113   8-01-85  10:43a
DBEDDT   CMD       123   8-01-85  10:43a
CONSOLE  INI       171   1-01-80  12:10a
INSTRS   BIN       767   5-13-85   3:43p
CONSOLE3 PAS     15335   7-11-85   2:32p
FAST     BIN      1079  11-14-84   8:12a
BOOT     BIN      5083   1-01-80  12:15a
CTY      CMD        58   8-01-85  10:42a
KLINSTRS BIN       767   5-14-85   2:57p
BOOT     CMD       247   8-01-85  10:42a
KLFAST   BIN      1079  11-14-84   8:12a
KLBOOT   BIN      5486   4-03-85   9:15a
INSTRS   MLD      2957   5-13-85   3:40p
KLINSTRS MLD      2957   5-14-85  12:55p
LIST     PAS      6695   7-08-85  11:43a
CONSOLE  PAS     49927   8-01-85  10:09a
LIST     COM     15443   7-09-85  12:51p
INSTALL  BAT       297   1-01-80  12:09a
 1DS     LST      1079   8-01-85  10:40a
CONSOLE  LOG       193   8-09-85  12:21p
CONDISK  BAT       589   8-09-85  12:24p
REBOOT   CMD        54   8-01-85  10:42a
        29 File(s)     95232 bytes free

A>TYPE CONSOLE.INI
;CONSOLE INIT FILE (NO LOG)
LOG: OFF
REMOTE: OFF
CTY: 2 1200 0 2 8
MODEM: 1 1200 0 2 8
IDLE: MIC ORUS DMA
DELAYS: 4 7 4 7 5 7 5 3 0 12 0 11 0 7
DMA: 30
AMEM: 16

A>
```

SYSBOOT.CMD
@SYSBOOT.CMD
CRASH.CMD

```
A>REM THIS IS THE CONSOLE USER DISK.

A>DIR

 Volume in drive A is CONSOLE3
 Directory of  A:\

INSTALL  BAT       296   8-09-85  12:19p
CTY      CMD        58   8-01-85  10:42a
BOOT     CMD       247   8-01-85  10:42a
REBOOT   CMD        54   8-01-85  10:42a
DBGO     CMD       113   8-01-85  10:43a
DBEDDT   CMD       123   8-01-85  10:43a
BOOTU    CMD        72   8-01-85  10:43a
MBOOT    CMD       118   8-01-85  10:43a
UTOBJ    CMD        64   8-01-85  10:43a
CONSOLE  COM     19621   8-09-85   1:28p
CONSOLE  000     44800   8-09-85   1:28p
CONSOLE  001     18432   8-09-85   1:25p
CONSOLE  002     11008   8-09-85   1:26p
CONSOLE  003     35584   8-09-85   1:27p
CONSOLE  HLP     18090   7-11-85   2:39p
CONSOLE  INI       171   1-01-80  12:10a
INSTRS   BIN       767   5-13-85   3:43p
KLINSTRS BIN       767   5-14-85   2:57p
FAST     BIN      1079  11-14-84   8:12a
KLFAST   BIN      1079  11-14-84   8:12a
BOOT     BIN      5083   1-01-80  12:15a
KLBOOT   BIN      5486   4-03-85   9:15a
DIAG     BIN     47203  11-08-84   8:39a
CONSOLE  SUP       193   8-09-85  12:21p
CONSOLE  PAS     49927   8-01-85  10:09a
CONSOLE  LOG       193   8-09-85  12:21p
       26 File(s)      84992 bytes free

A>TYPE INSTALL.BAT
REM INSTALL CONSOLE TO SUPER-DRIVE D:
COPY CONSOLE.* D:
COPY *.CMD D:
COPY *.BIN D:
REM SPECIAL CONSOLE.INI FOR LOG FILE
COPY CONSOLE.SUP D:CONSOLE.INI
D:
REM CONSOLE INSTALLED, PLACE LOG DISK IN DRIVE A:
REM USE "CONSOLE BOOT" TO BOOT SYSTEM
REM USE "CONSOLE CTY" TO RETURN TO SYSTEM

A>
```

```
A>TYPE QSYSBOOT.CMD
;boot up system
SET QUIET OFF
TEST
SET QUIET ON
RESET SYSTEM
 loading boot routine
 LOAD FAST
LOAD BOOT
;booting micro code from disk
BOOT MICRO
;booting monitor from disk
RESET
BOOT MACRO
;starting system -- going to CTY
CTY BOOT

A>TYPE CRASH.CMD
;RECORD HISTORY AFTER A SYSTEM CRASH
STOP
SET QUIET OFF
HISTORY 40 A:CRASH
TEST

A>TYPE 26DISK.BAT
REM COPY FILES TO 26 CONSOLE DISK ON DRIVE A
DEL A:*.*
COPY INSTALL.BAT A:
COPY CTY.CMD A:
COPY SYSBOOT.CMD A:
COPY QSYSBOOT.CMD A:
COPY REBOOT.CMD A:
 OPY DBGO.CMD A:
OPY DBEDDT.CMD A:
COPY BOOTU.CMD A:
COPY MBOOT.CMD A:
COPY UTOBJ.CMD A:
COPY CRASH.CMD A:
COPY CONSOLE.COM A:
COPY CONSOLE.000 A:
COPY CONSOLE.001 A:
COPY CONSOLE.002 A:
COPY CONSOLE.003 A:
COPY CONSOLE.HLP A:
COPY CONSOLE.INI A:
COPY CONSOLE.LOG A:
COPY INSTRS.BIN A:
COPY FAST.BIN A:
COPY BOOT.BIN A:
COPY DIAG.BIN A:
COPY SYSTEM.BIN A:

A>
```

.type memo.txt
8/30/85

Here is the current status of the console project...

The basic design/documentation/coding/installation is complete.

Known bugs are:

1) The idle DMA address does not reset to the value specified in CONSOLE.INI. Be careful not to disturb the fast loader which uses DMA addres 30 in all cases.
2) The reset system function in the backend needs to zero out the first 20 or so locations of main memory (SRI bug)
3) The macro breakpoint stuff should be fixed. The problem seems to be in the backend function. This will require detailed exploration of the schematics to determine what it is supposed to do in the first place.

A maintenance log has been established to track modifications of the CONSOLE program. Whenever changes are made, I.E. fixing bugs, the new edit number and a brief description of the change is to be recorded by the person making the change. This is a physical pad of paper in the sources folder.

The installation process has been automated to a certain extent. There are three batch/command files which copy to a floppy in drive A. They are 26DISK.BAT, KLDISK.BAT and DISK.BAT. Use them to build CONSOLE disks for 26, KL or both. Note that while DISK.BAT will make a console which is both a 26 and a KL, the system micro code is not included on the disk since there is no room!

The CONSOLE may or may not make a log file as follows. If the CONSOLE is executed from the floppy, it will not make a log file. If the INSTALL.BAT file is used to install a CONSOLE to the ram disk, the CONSOLE will make a log file on a floppy in drive A. This can be confused if the CONSOLE disk does not have a write-protect tab in place! (Look at INSTALL.BAT to see how it does it and all will be understood...)

The user documentation is off to a good start. Several copies are on the shelf and a master is in the source folder. More work should be done on the CONSOLE.HLP file, but if it gets too big, there will be problems with the floppies. I have explained this to Rhett.

I have spent quite a bit of time with Mitchell going over the source code. I feel that he has a good understanding of the program and how it is intended to operate.

Over the next few days, we should all meet to cover any material that needs to be. Perhaps Virgil should be involved also.

Zon

HELP

The normal console/operator interaction consists of typing commands in the command area and inspecting the results in the register display and/or the status message line. A command consists of an initial keyword followed by any appropriate arguments.

A simple macro processor analyzes each command, replacing each macro name with its defined body. The MACRO command is used to DEFINE and ERASE macro definitions.

Macro definitions can be inspected with the "?" command. When the DEBUG flag is on, the result of the macro replacement is displayed.

Macros may be pre-defined in the CONSOLE.INI setup file with the "MACRO:" option. Example: "MACRO: FOO BODY-OF-FOO"

Several command lines may be stored in a text file and then may be submitted to the console for processing. If the keyword on a command line is not a usual command, the console will attempt to open such a command file.

Up to two command line arguments may be passed to a command file via the "*1" and "*2" macros.

Note that the QUIET flag is set whenever commands are being taken from a command file. This prevents the display of intermediate command results. This flag can be turned ON or OFF with the SET command.

Comment command lines begin with ";" and are always displayed on the status message line.

The following command files are supplied with the console:

DBEDDT.CMD -- disk boot to EDDT

DBGO.CMD -- disk boot and go

BOOTU.CMD -- boot micro code from disk

UTOBJ.CMD -- boot micro code from tape

MBOOT.CMD -- boot monitor from tape

The following command files are to be specified when the console program is started from PC-DOS, since the names are similar to real commands.

BOOT.CMD -- boot up system

CTY.CMD -- return to CTY

The default file extension for a command file is ".CMD". In the previous example, the boot file could be submitted by actually typing "BOOT.CMD" on the console command line.

On-line documentation for commands and macros can be displayed with the "?" command. A command line consisting of just "?" will display the names of all available commands and macros. If "?" is followed by a keyword, a short description will be displayed on the status message line. "?*" will display descriptions for all commands and macros.

The "?" command is available as an argument to the CALL command to display documentation on the backend functions.

When documentation is requested for a single item, it is possible to get a more detailed explanation from the help file CONSOLE.HLP. Answer "Y" to the question "Explain?" to see this.

The TYPE command can be used to display other text files. An optional second argument is a keyword to search for in the file.

The side-kick notepad can also be used to wander about in any documentation file. See the side-kick documentation for more information on this procedure.

REMOTE

The console program supports a data connection between two PC's. This allows operators at a remote site to manipulate a local console. The PC-DOS command "CONSOLE REMOTE" is used to start up such a remote console. The data connection is usually via a phone modem.

The local console must allow the access. This can be specified by the REMOTE command. "REMOTE OFF" will terminate any connection and disallow any new connection. "REMOTE ON" allows connections and will attempt to make a connection. Typing any key will abort the attempt. A second argument may specify a password.

The "REMOTE:" option in the CONSOLE.INI file will establish a default remote state. The syntax is: "REMOTE: ON/OFF [password]".

The MODEM command can be used to set parameters such as the baud rate and parity for the MODEM or CTY ports. The parameters are decimal integers in the sequence: baud rate, parity, stop bits and word length.

The default values can be established in CONSOLE.INI with the "MODEM:" and "CTY:" options. These take an additional initial integer parameter to specify the port number to use. An example is: "MODEM: 2 1200 0 2 8". This sets the modem to be COM2 at 1200 bauds, no parity, 2 stop bits and 8-bit words.

If the modem is able to dial out, the "DIAL: ON" option may be used in the CONSOLE.INI file. The console will request and dial a phone number whenever a MODEM connection is established.

The MODEM command, without the oprt parameters, can be used for interactive access to the MODEM and CTY ports. This can be usefull in debugging port connections.

The TALK command allows communication between remote and local operators. The register screen is replaced by message lines labeled "LOC:" or "REM:". Type a blank line to exit talk mode.

DIAGNOSTICS

The console program has a set of 45 internal diagnostic routines which verify the console's ability to control the Augment Engine. When all else fails, use the TEST command to execute these diagnostics.

When a test fails, the RECEIVED/EXPECTED values are displayed along with the test name. An additional explanation will be displayed if "Y" is answered to the question "Explain?".

Tests are labeled by number. Use "?" to display documentation. For example, "?TEST3" will tell about test #3.

If no arguments are supplied, the TEST command will execute every test in sequence. An argument of "*" will repeat this indefinitly.

A numeric argument may be supplied to TEST, in which case that particular test is executed. If "*" is then supplied as a second argument, that test will be executed indefinitly.

There are 45 tests. They start with a verification of the physical connection between the PC and Engine and work their way up to reading and writing registers, micro instruction and main memory.

The console program also contains an internal DEBUG flag which controls the display of debugging information. This may be turned ON or OFF with the SET command. The DEBUG flag can be turned on initially by the DEBUG argument on the PC-DOS command line. (DEBUG can be turned on sooner than initially by compiling the console with INITIAL_DEBUGGING =TRUE)

The SET command controls two other debugging flags. The COUNT flag controls the internal activity counter. The UPDATE flag controls the idle updating.

The PC-DOS DEBUG utility can be used on the console program with the following considerations. Before the code in an overlay can be twiddled, it must actually be in memory. This can be done by first determining the address of the non-overlayed DEBUG_BREAK procedure. Compile the console where the DEBUG_BREAK is called in the routine to be debugged. Use DEBUG to set a break at the DEBUG_BREAK procedure. When the break point is hit, the desired routine is in memory!

BOOTING

Micro and macro code can be loaded from the PC with the LOAD command. The DUMP command is the inverse of the LOAD command.

Micro code files must be in a special binary format. The "MLD" octal format can be converted to "BIN" binary format with the BUILD command.

The LOAD can be FAST or SLOW. FAST requires that the Augment Engine be in reasonable operating condition with the FAST loader routine in micro memory. Use "LOAD FAST" to set up the FAST loader.

Code can be VERIFIED as it is loaded or CHECKED afterwards. For example, use "LOAD BOOT VERIFY" to load and verify the boot code. Use "LOAD FAST CHECK" the check out the previously loaded fast code.

The BOOT command is used to interface to the BOOT routine running on the Engine. Make sure BOOT has been LOADED. The options to the BOOT command are grouped as follows:

MICRO (defaults to disk boot)
 DISK
  BOOTU/BOOTD/MEMIMG/BOOT
   disk number (not KL)
    PRIMARY/SECONDARY (home block)
     NEW/OLD (MMB board)
 TAPE, SLOEXP/MBOOT/OBJ/CCL, file number
MACRO (defaults to disk boot)
 DISK
  START/LOAD/EDDT, bootstrap disk, monitor disk (not KL)
   PRIMARY/SECONDARY (home block)
    monitor file number(KL)
 TAPE, file number

The default arguments for the BOOT command are such that "BOOT MICRO" will boot micro code from disk and "BOOT MACRO" will boot a monitor. Use just "BOOT" to be prompted.

Whenever a monitor is booted, the starting macro address is calculated. The CTY command can then be used to start the monitor by "CTY BOOT". Alternately, the MSTART or MGO commands will also accept "BOOT" as an address.

FEATURES

The console display usually consists fo, from top to bottom, an identification banner, the machine registers, a status message line, the command area and an alert message line. The identification banner displays the current date and time, the console version number and the Augment Engine type and number.

The middle of the screen is devoted to the various registers. This area will sometimes be replaced by documentation, history tables, etc. but will be restored whenever a command is finished.

LOAD FILE	-- name of last micro code loaded
STATE	-- running or stopped
REMOTE	-- state of remote access
MBRK,UBRK	-- macro/micro break address
M	-- main memory / ecc
DMA	-- dma access to main memory
PC,MA,AR,MEM,etc.	-- Augment Engine registers
JMEM,AMEMP,AMEM	-- amem and jmem access
MIC,OBUS,EOBUS	-- micro engine status
LC,TC,BRANCHING	-- condition flags
MI,EEAL,EEFO,etc.	-- micro instruction and fields

The DD command will read new values for all registers. The IDLE command can select the MIC, OBUS and DMA for continuous updating. In CONSOLE.INI use "AMEM:", "DMA:" and "IDLE:" options for initial settings.

Below the register display is the single status message line. Usually this will begin with the name of the current command followed by any interesting data the command returns.

The next four lines are the command area. Commands are entered at the bottom and scrolled up. Any error message is displayed next to the command on the same line.

The ALERT line at the bottom of the screen displays various trivia, such as changes in COM port status or debug information. Note that a debug alert ends with a "*". Type just a carriage return to continue or enter the number of debug calls to skip.

When in CTY or MODEM mode, the entire screen is devoted to the interaction. The PC supports a weak VT100 emulation. Exit the console with the ports active and use other terminal emulation software (such as AUGTERM or TYMCOMM) for more accuracy.

The RESET command performs four functions. The default function is to execute the micro code reset routine ("BOOT" must be loaded).

"RESET SYSTEM" will reset the Augment Engine hardware and internal console interface parameters.

"RESET INITS" will reset all of the delay line values. The operator has the option of changing individual delay settings. The initial values of the delays can be specified in CONSOLE.INI with the "DELAYS:" option followed by octal values in sequence.

"RESET CONSOLE" will reset the console program on the PC. Arguments may be specified just like on the PC-DOS command line.

When the console program is executed, it sets up its own internal state, but does not write anything to the Engine. Thus, entering and exiting the console does not affect the operation of the main system.

It is good practice to do a "RESET SYSTEM" whenever the hardware is powered up. Use "RESET INITS" to twiddle the delay values. Use the basic "RESET" before booting monitors and such.

The Micro Engine is startted and stopped by the RUN and STOP commands. Alternately, the UBRK command can set a break point in the micro code, at which point the Engine will stop.

The USTART command points the MIC to a specific micro address to begin execution. The UGO is a combination USTART and RUN.

The SS command will single step through micro code. The next instruction to be executed is displayed in the MI register. The CS command will step the current micro instruction without advancing the MIC.

The last 1000 Micro Engine states are saved in the micro history. Use the HISTORY command to display the table starting at any offset.

The interpretation of macro code can be controlled by the MSTART, MGO, and MBRK commands which correspond to the USTART, UGO and UBRK commands for micro code.

Macro history information is also displayed with the HISTORY command. Note that the HISTORY command can dump its display to a text file.

When the Engine is running, all commands which could cause the system to crash are locked to prevent accidents. Use the SET command to change the LOCK flag only if you really have to.

The registers are displayed after each single step (like the DD command). This feature may be changed by using the SET command to turn the DD flag off. This can speed up the operation of a remote console.

Register and memory values are inspected and changed with a variety of deposit and examine commands.

The following commands deposit values:

DE,DMWRT -- main memory

USTART,UI,MILOAD,MMLOAD -- micro memory

DAC,LDAC -- ac registers

DAM,DAMI,DAMP,LDAMEMP -- amem

LDJMEMP -- jmem

LDAR,LDDEV,LDHOLD,LDIR,LDMA,LDPC,LDPCF,LDQ -- registers

The following commands examine values:

EX,DMRD -- main memory

MIC,MIREAD,MMREAD -- micro memory

XAC -- ac registers

XAM,XAMI,XAMP,XAMEMP -- amem

XJMEM,XJMEMP -- jmem

XAR,XDEV,XIR,XLP,XMA,XMEM,XPC,XPCF,XQ -- registers

Note that MEM is examined but HOLD is deposited.

The console program is organized as a frontend command/display processor which does all of its work via a backend function processor. When in a remote configuration, the remote frontend makes its calls to the local backend.

The backend interfaces to the Augment Engine. In it are functions which read and write the control flags, such as IORA and IORB, and higher level functions such as RDOB and WROB which read and write the OBUS. The highest level functions in the backend read and write macro and micro memory. All of the test routines are in the backend.

Each function in the backend is accessable via the CALL command. The command arguments are similar to the source code procedure and function arguments. Since the CALL command can do very subtle things, remember to do a "RESET SYSTEM" if things get strange.

A quick access to the obus and status loop is provided by the RDOBUS, WROBUS and RDSTATUS commands.

It is possible to put various CALL commands in a command file to perform complex procedures directly.

```
04-Aug-89          Augment Engine (KL) #122 Console V2.61          12:57:45

        Load: KLBOOT            State: Stopped            Remote:Off
 MBRK=OFF          MI[00000301=000000,,017645/000    DMA[00000030]=000000,,017645
 PC=000000,,000000   MA=400000,,005000   AR=450530,,230000 MEM=000000,,000000
 IR=000000,,017645    0=000000,,000000   LP=177777        AC[00]=000000,,000000
        JMEM[0000]=17777       DEV=00   AMEMP=0000     AMEM[0016]=000000,,000000
  UBRK=OFF     MIC=17645    OBUS=000000,,000000            EOBUS=777777,,777777

 LC=ON      TC=ON      Branching      MI=00204707600005413303434505302O
 EEAL=0   EEFO=0    LDMA=0      IDISP=0      MWT=0   SAAFMA=1      PO=0   SP1=0 CRY=0
 ASRC=2   AFUN=3    ADST=1       ALU1=0    LDAR=1    JCOND=74
 MAPF=00   SP0=00   JADR=13026   ROT=60   MASK=70
 DEST=71   SP2=0   JCODE=10    ACSEL=5        D=30   CYLEN=04   IFU=0   DFU=0

LOAD: KLBOOT loaded.

! load fast
! load boot
! g
```

COMMANDS

The following is a listing of the various command files and the on-line documentation file CONSOLE.HLP.

CONSOLE.HLP is organized to allow access by the TYPE command, seraching for keywords. Note that command keywords are extended by "_CMD" and functions by "_FUN". This is consistant with the internal sysmbols used in the source code.

Following the commands and functions is the TEST documentation. Keywords are "TESTn" where "n" is the test number.

After the tests are the internal micro instructions, ending with "_MI". Note that these symbolic values are available to commands such as MILOAD and MMLOAD. (Other usefull symbolic values are ON, OFF, ODD and EVEN.)

The CONSOLE.HLP file ends with micro instruction fields, "_FLD" and machine registers "_REG".

Use the "?" command to display these other items. For example, use "? MEM_REG" to display the information about the MEM register or "? DFQ_FLD" to tell about the DFQ field.

```
FILE: CTY.CMD
  ;usage: >CONSOLE CTY -- connect to cty from PC-DOS
  CTY

FILE: REBOOT.CMD
  ;load fast/boot
  set quiet off
  reset system
  load fast
  load boot

FILE: BOOT.CMD
  ;usage: >CONSOLE BOOT -- boot up system from PC-DOS
  RESET SYSTEM
  ;loading boot routine
  LOAD FAST
  LOAD BOOT
  ;booting micro code from disk
  BOOT MICRO
  ;booting monitor from disk
  RESET
  BOOT MACRO
  ;starting system -- going to CTY
  CTY BOOT

FILE: BOOTU.CMD
  ;bootu <drive> -- boot micro code from disk
  BOOT MICRO DISK BOOTU #1

FILE: DBGO.CMD
  ;dbgo <drive> -- disk boot monitor
  RESET
  BOOT MACRO DISK START #1
  ;start system -- go to CTY mode
  CTY BOOT

FILE: DBEDDT.CMD
  ;dbeddt <drive> -- disk boot monitor to eddt
  RESET
  BOOT MACRO DISK EDDT #1
  ;start up EDDT -- go to CTY mode
  CTY BOOT

FILE: MBOOT.CMD
  ;mboot <file> -- boot monitor from tape
  RESET
  BOOT MACRO TAPE #1
  ;user must now start monitor and go to CTY mode

FILE: UTOBJ.CMD
  ;utobj <file> -- boot ucode from tape
  BOOT MICRO TAPE OBJ #1
```

; CONSOLE HELP FILE V2.6

; COMMANDS

BOOT_CMD BO/OT [MICRO/MACRO] -- Bootstrap operation.
Use BOOT MICRO to boot micro code, and BOOT MACRO to boot a monitor. This will set the "BOOT" address for the CTY, MSTAT or MGO commands. If no args are supplied, then all options will be solicited as follows:
MICRO,
  DISK, BOOTU/BOOTD/MEMIMG/BOOT, disk number (not KL),
    PRIMARY/DECONDARY (home block), NEW/OLD (mmb board)
  TAPE, SLOEXP/MBOOT/OBJ/CCL, file number
MACRO,
  DISK, START/LOAD/EDDT, bootstrap disk, monitor disk (not KL),
    PRIMARY/SECONDARY (home block), monitor file number (KL)
  TAPE, file number

BUILD_CMD BU/ILD <file name> -- Build BIN file from MLD.
Convert a binary ".BIN" micro code file from an octal ".MLD" file. All micro code files to be loaded must be built to binary format.

CALL_CMD CA/LL <function>,[parameter(s)] -- Call backend function.
This is the interactive access to all backend functions. Note that "?" may be used after CALL. For example, "CALL ?" will display the names of all backend functions. Arguments to "?" in CALL are similar to a normal "?".

CS_CMD CS [count/*] -- Clock step.
A clock step does not load the micro instruction register. The default count is one. A count of "*" signifies continuous clocking until a key is pressed.

CTY_CMD CT [mgo] -- Connect to CTY.
The screen display will be replaced by the CTY screen. The CTY command can also do an MGO, with the corresponding arguments. Use CTY BOOT to start up a monitor after booting. All CTY interactions are logged.

DAC_CMD DAC <AC number>,<value> -- Deposit into AC.
Deposit value into specified AC register.

DAM_CMD DAM <address>,<value> -- Deposit into AMEM.
Deposit value into AMEM at specified address.

DAMI_CMD DAMI <address>,<value> -- Deposit into AMEM(I).
Deposit value into AMEM indexed at specified address.

DAMP_CMD DAMP <value> -- Deposit into AMEM(P).
Deposit value into AMEM at AMEMP address.

DD_CMD DD -- Display all registers.
A DD command is done automatically after a STOP or SS command. The DD command will only update the MIC, OBUS and MI if SET DD OFF.

DE_CMD DE <address>,<value> -- Deposit main memory.
Deposit value into main memory at specified address.

DMRD_CMD DMR/D <address> -- DMA memory read.
DMA memory read at specified address. This will set the address used for the IDLE DMA reads.

DMWRT_CMD DMW/RT <address>,<value> -- DMA memory write.
DMA memory write value at specified address.

DUMP_CMD DU/MP <file>,[MICRO/MACRO] -- Dump micro or macro memory.

EX_CMD EX <address> -- Examine memory.

?_CMD ? [command] -- Describe command.

HISTORY_CMD HI/STORY [offset],[file] -- History analysis.

IDLE_CMD ID/LE OFF,MIC,OBUS,DMA -- Idle dpy.

LDAC_CMD LDAC <value> -- Load AC(DEV).

LDAMEMP_CMD LDAM/P <value> -- Load AMEMP.

LDAR_CMD LDAR <value> -- Load AR.

LDDEV_CMD LDD/EV <value> -- Load DEV.

LDJMEMP_CMD LDJ/MEMP <value> -- Load JMEMP.

LDHOLD_CMD LDH/OLD <value> -- Load HOLD.

LDIR_CMD LDI/R <value> -- Load IR.

LDMA_CMD LDMA <value> -- Load MA.

LDPC_CMD LDPC <value> -- Load PC.

LDPCF_CMD LDPCF <value> -- Load PCF.

LDQ_CMD LDQ <value> -- Load Q.

LOAD_CMD LOA/D <file>,[MICRO/MACRO,CHECK/VERIFY] -- Load memory.

MACRO_CMD MA/CRO DEFINE/ERASE -- Macro facility.

MBRK_CMD MBR/K <address>/OFF,[count,delay]] -- Macro break.

MGO_CMD MG/O <address>,[break] -- Macro start and go.

MIC_CMD MIC -- Display MIC from status.

MILOAD_CMD MIL/OAD [A,B,C],[instr] -- Load MI(section).

MIREAD_CMD MIR/EAD -- Read current MI.

MMLOAD_CMD MML/OAD [A,B,C],[address [instr]] -- Load UMEM.

MMREAD_CMD MMR/EAD <address>,[DISPLAY] -- Read UMEM.

MODEM_CMD MO/DEM [CTY],[control] -- Communications port.

MSTART_CMD MS/TART <address>,[break] -- Set macro start address.

QUIT_CMD	Q/UIT -- Exit console program.
RDOBUS_CMD	RDO/BUS [TRUE/FALSE] -- Read(load) obus.
RDSTATUS_CMD	RDS/TATUS -- Read status loop.
REMOTE_CMD	REM/OTE ON/OFF,[password] -- Remote access.
RESET_CMD	RES/ET [address/INITS/SYSTEM/CONSOLE] -- Reset system.
RUN_CMD	RU/N -- Run system.
SET_CMD	SE/T [DEBUG/LOCK/LOG/QUIET],[ON/OFF] -- Set flags.
SS_CMD	SS [count] -- Single step micro instruction.
STOP_CMD	ST/OP -- Stop system.
TALK_CMD	TA/LK -- Enter talk mode.
TEST_CMD	TE/ST [number/*] -- Execute test routine(s).
TYPE_CMD	TY/PE <file name>,[match] -- Display contents of a file.
UBRK_CMD	UB/RK <address/OFF>,[count] -- Micro break point.
UGO_CMD	UG/O <address>,[break] -- Start micro execution.
UI_CMD	UI <field>,<value> -- Set micro instr field.
USTART_CMD	US/TART <address>,[break] -- Set micro start address.
WROBUS_CMD	WR/OBUS <value> -- Write obus.
XAC_CMD	XAC <AC number> -- Examine AC.
XAM_CMD	XAM <address> -- Examine AMEM.
XAMI_CMD	XAMI <address> -- Examine AMEM(I).
XAMP_CMD	XAMP -- Examine AMEM(P).
XAMEMP_CMD	XAME/MP -- Examine AMEMP.
XAR_CMD	XAR -- Examine AR.
XDEV_CMD	XD/EV -- Examine DEV.
XIR_CMD	XI/R -- Examine IR.
XJMEM_CMD	XJMEM -- Examine JMEM(P).
XJMEMP_CMD	XJMEMP -- Examine JMEMP.
XLP_CMD	XL/P -- Examine LP.

```
XMA_CMD         XMA -- Examine MA.

XMEM_CMD        XMEM -- Examine MEM.

XPC_CMD         XPC -- Examine PC.

XPCF_CMD        XPCF -- Examine PCF.

XQ_CMD          XQ -- Examine Q.

;FUNCTIONS

BITS_FUN        BI/TS <bits>,<siz>,<bit_siz>,LEFT/RIGHT -- Convert bits.

BREAK_FUN       BREA/K <micro?>,<addr/ON/OFF>,[count,delay] -- Break point.

BUILD_FUN       BU/ILD <file name> -- Build binary load file.

CPUOB_FUN       CP/UOB <yes?> -- CPU controls OBUS.

CTY_FUN         CT/Y [yes?] -- Remote CTY mode.

DUMP_FUN        D/UMP <micro?>,[file/begin,end] -- Dump memory.

ENAHST_FUN      EN/AHST <yes?> -- Enable history recording.

EXEC_FUN        EX/EX <instr>,<clock?> -- Execute(and clock) instr.

FLD_FUN         F/LD <field name> -- Convert field name.

INIT_FUN        INIT <send?> -- Initialize (and send) flags.

INIT2_FUN       INIT2 [on],[off] -- Read/write INIT2.

INITS_FUN       INITS <all?> -- Send (all) inits.

INSTRS_FUN      INS/TRS <file name> -- Load constant instructions.

IORA_FUN        IORA [on],[off] -- Read/write IORA.

IORB_FUN        IORB [on],[off] -- Read/write IORB.

JUMP_FUN        J/UMP <address> -- Build jump instr.

LOAD_FUN        L/OAD <micro?>,<file>,<code> -- Load slow,verify,check,fast.

MI_FUN          MI <instr> -- Translate instruction.

MMACTRL_FUN     MM/ACTRL [on],[off] -- Read/write MMACTRL.

RDAC_FUN        RDAC <AC number> -- Read AC.

RDAM_FUN        RDAM <address>,<indexed?> -- Read AMEM.

RDDMA_FUN       RDDMA <address> -- DMA memory read.

RDECC_FUN       RDEC/C -- Read memory ECC.
```

RDHST_FUN RDH/ST <micro?>,[index,relative?] -- Read history.

RDHSTI_FUN RDHSTI <relative?>,<micro?> -- Read history index.

RDM_FUN RDM <address> -- Read memory.

RDMI_FUN RDMI -- Read micro instruction.

RDMIC_FUN RDMIC -- Read micro instruction counter.

RDOB_FUN RDOB <load?>,<eobus?> -- (Load and) read OBUS.

RDPORT_FUN RDP/ORT <port> -- Read port.

RDREG_FUN RDR/EG <register> -- Read register.

RDST_FUN RDST -- Read status items.

RDSTLP_FUN RDSTL/P -- Read status loop.

RDUM_FUN RDUM <address> -- Read micro memory.

REG_FUN REG <register> -- Register name conversion.

RESET_FUN RESE/T -- Reset system after board change.

RUN_FUN RU/N <yes?> -- Run/Stop CPU.

SAVE_FUN SA/VE <yes?>,<force?> -- Save MIC and MI.

SETFLD_FUN SETF/LD <instr>,<field>,<value> -- Set field value.

SHIFT_FUN SH/IFT <count> -- Shift load registers.

STEP_FUN STE/P <load?>,<count> -- Step (and load) micro instr.

SYSID_FUN SY/SID <system number?> -- Read system number or type.

TALK_FUN TA/LK <remote?>,[message] -- Talk message mover.

TEST_FUN TE/ST <number> -- Execute test routine.

WRAC_FUN WRAC <AC number>,<value> -- Write AC.

WRAM_FUN WRAM <address>,<value>,<indexed?> -- Write AMEM.

WRDMA_FUN WRDMA <address>,<value> -- DMA wr memory.

WRHSTI_FUN WRHS/TI <index>,<relative?>,<micro?> -- Write history index.

WRM_FUN WRM <address>,<value> -- Write memory.

WRMI_FUN WRMI <instr>,<section> -- Write micro instruction.

WRMIC_FUN WRMIC <address>,<first?>,<load?> -- Write MIC.

```
WROB_FUN        WROB <value> -- Write obus.

WRPORT_FUN      WRP/ORT <port>,<value> -- Write port.

WRREG_FUN       WRR/EG <register>,<value> -- Write register.

WRUM_FUN        WRUM <address>,<instr>,<section> -- Write micro memory.

;DIAGNOSTIC TESTS
codes 1 to 28 crash initial values
codes > 28 must have stable initial values
what about the remote parity error?
how should we react to parity errors in general?

TEST1
BEGIN {cable test, bit set if installed}
  PORT[iorb_port]:=cc_res; byte_test(PORT[iorb_port],cc_res)
END;

TEST2
BEGIN {reset local,clear local parity error}
  PORT[loc_res_port]:=0; byte_test(PORT[loc_sts_port] AND loc_par_err,$00)
END;

TEST3
BEGIN {test 3 should have cleared diag dat bits}
  byte_test(PORT[diag_port],0)
END;

TEST4
FOR i:=1 TO 4 DO BEGIN {load and check diag dat}
  PORT[diag_port]:=test_byte[i]; byte_test(PORT[diag_port],test_byte[i])
END;

TEST5
BEGIN {force local parity error}
  PORT[loc_err_port]:=0;
  byte_test(PORT[rem_sts_port] AND loc_par_err,loc_par_err)
END;

TEST6
FOR i:=1 TO 4 DO BEGIN {dat bank 0}
  PORT[port_0]:=test_byte[i]; byte_test(PORT[port_0],test_byte[i])
END;

TEST7
FOR i:=1 TO 4 DO BEGIN {dat bank 1}
  PORT[port_1]:=test_byte[i]; byte_test(PORT[port_1],test_byte[i])
END;

TEST8
FOR i:=1 TO 4 DO BEGIN {dat bank 2}
  PORT[port_2]:=test_byte[i]; byte_test(PORT[port_2],test_byte[i])
END;

TEST9
FOR i:=1 TO 4 DO BEGIN {dat bank 3}
```

```
  PORT[port_3]:=test_byte[i]; byte_test(PORT[port_3],test_byte[i])
END;

TEST10
FOR i:=1 TO 4 DO BEGIN {dat bank 4}
  PORT[port_4]:=test_byte[i]; byte_test(PORT[port_4],test_byte[i])
END;

TEST11
FOR i:=1 TO 4 DO BEGIN {dat bank 5}
  PORT[port_5]:=test_byte[i]; byte_test(PORT[port_5],test_byte[i])
END;

TEST12
FOR i:=1 TO 4 DO BEGIN {dat bank 6}
  PORT[port_6]:=test_byte[i]; byte_test(PORT[port_6],test_byte[i])
END;

TEST13
FOR i:=1 TO 4 DO BEGIN {dat bank 7}
  PORT[port_7]:=test_byte[i]; byte_test(PORT[port_7],test_byte[i])
END;

TEST14
BEGIN {rd zeros from shifts out bits}
  PORT[port_0]:=0; PORT[port_1]:=0; PORT[port_2]:=0; PORT[port_3]:=0;
  PORT[port_4]:=0; PORT[port_5]:=0; PORT[port_6]:=0; PORT[port_7]:=0;
  byte_test(PORT[rd_shifts_port],$00)
END;

TEST15
BEGIN {rd ones from shifts out bits}
  PORT[port_0]:=1; PORT[port_1]:=1; PORT[port_2]:=1; PORT[port_3]:=1;
  PORT[port_4]:=1; PORT[port_5]:=1; PORT[port_6]:=1; PORT[port_7]:=1;
  byte_test(PORT[rd_shifts_port],$FF)
END;

TEST16
BEGIN {load shift bank 0 to bank 1 with count of 1}
  PORT[port_0]:=$01; PORT[port_1]:=0;
  shift(1); byte_test(PORT[port_1],$80)
END;

TEST17
BEGIN {load shift bank 1 to bank 2 with count of 2}
  PORT[port_1]:=$02; PORT[port_2]:=0;
  shift(2); byte_test(PORT[port_2],$80)
END;

TEST18
BEGIN {load shift bank 2 to bank 3 with count of 4}
  PORT[port_2]:=$08; PORT[port_3]:=0;
  shift(4); byte_test(PORT[port_3],$80)
END;

TEST19
BEGIN {load shift bank 3 to bank 4 with count of 8}
```

```
  PORT[port_3]:=$80; PORT[port_4]:=0;
  shift(8); byte_test(PORT[port_4],$80)
END;

TEST20
BEGIN {load shift bank 4 to bank 5 with count of 1}
  PORT[port_4]:=$01; PORT[port_5]:=0;
  shift(1); byte_test(PORT[port_5],$80)
END;

TEST21
BEGIN {load shift bank 5 to bank 6 with count of 2}
  PORT[port_5]:=$02; PORT[port_6]:=0;
  shift(2); byte_test(PORT[port_6],$80)
END;

TEST22
BEGIN {load shift bank 6 to bank 7 with count of 4}
  PORT[port_6]:=$08; PORT[port_7]:=0;
  shift(4); byte_test(PORT[port_7],$80)
END;

TEST23
BEGIN {load shift bank 7 to bank 0 with count of 8}
  PORT[port_7]:=$80; PORT[port_0]:=0;
  PORT[diag_loop_port]:=0;
  shift(8); byte_test(PORT[port_0],$80);
  PORT[rem_res_port]:=0
END;

TEST24
BEGIN {load shift bank 0 to bank 2 with count of 16}
  PORT[port_0]:=$80; PORT[port_1]:=0; PORT[port_2]:=0;
  shift(16); byte_test(PORT[port_2],$80)
END;

TEST25
BEGIN {load shift bank 0 to bank 4 with count of 32}
  PORT[port_0]:=$80; PORT[port_1]:=0; PORT[port_2]:=0;
  PORT[port_3]:=0; PORT[port_4]:=0;
  shift(32); byte_test(PORT[port_4],$80)
END;

TEST26
BEGIN {load shift bank 0 to bank 0 with count of 64}
  PORT[port_0]:=$80; PORT[port_1]:=0; PORT[port_2]:=0; PORT[port_3]:=0;
  PORT[port_4]:=0; PORT[port_5]:=0; PORT[port_6]:=0; PORT[port_7]:=0;
  PORT[diag_loop_port]:=0;
  shift(64); byte_test(PORT[port_0],$80);
  PORT[rem_res_port]:=0
END;

TEST27
FOR i:=1 TO 4 DO BEGIN {iora}
  PORT[iora_port]:=test_byte[i]; byte_test(PORT[iora_port],test_byte[i])
END;
```

```
TEST28
FOR i:=1 TO 4 DO BEGIN {iorb}
  PORT[iorb_port]:=test_byte[i]; byte_test(PORT[iorb_port],test_byte[i])
END;

TEST29
FOR i:=1 TO 4 DO BEGIN {obus}
  get_bits(test_bits[i],exp,0,36);
  wrob(exp,TRUE); rdob(rec,FALSE,FALSE); bits_test(rec,exp)
END;

TEST30
BEGIN {dma}
  oct_to_bits('30',addr,24);
  FOR i:=1 TO 4 DO BEGIN
    get_bits(test_bits[i],exp,0,36);
    wrdma(addr,exp); rddma(addr,rec); bits_test(rec,exp)
  END
END;

TEST31
FOR i:=1 TO 4 DO BEGIN {mi}
  get_bits(test_bits[i],exp,0,88);
  wrmi(exp,sect_ABC); rdmi(rec); bits_test(rec,exp)
END;

TEST32
FOR i:=1 TO 4 DO BEGIN {mic}
  get_bits(test_bits[i],exp,0,14); fit_bits(exp,16,1,right_fit);
  rec:=exp; wrmic(exp.bit[0],TRUE,FALSE); rec:=exp; rec.bit[0]:=rdmic;
  bits_test(rec,exp)
END;

TEST33
BEGIN {umem}
  oct_to_bits('4043',addr,14); fit_bits(addr,1,16,right_fit);
  FOR i:=1 TO 4 DO BEGIN
    get_bits(test_bits[i],exp,0,88);
    first:=TRUE; wrum(addr.bit[0],exp,sect_ABC,first);
    rdum(addr.bit[0],rec); bits_test(rec,exp)
  END
END;

TEST34
FOR i:=1 TO 4 DO BEGIN {PC register}
  get_bits(test_bits[i],exp,0,36);
  wrreg(pc_reg,exp); rdreg(pc_reg,rec); bits_test(rec,exp)
END;

TEST35
FOR i:=1 TO 4 DO BEGIN {MA register}
  get_bits(test_bits[i],exp,0,36);
  wrreg(ma_reg,exp); rdreg(ma_reg,rec); bits_test(rec,exp)
END;

TEST36
FOR i:=1 TO 4 DO BEGIN {AR register}
```

```
  get_bits(test_bits[i],exp,0,36);
  wrreg(ar_reg,exp); rdreg(ar_reg,rec); bits_test(rec,exp)
END;

TEST37
FOR i:=1 TO 4 DO BEGIN {MEM register}
  get_bits(test_bits[i],exp,0,36);
  wrreg(hold_reg,exp); rdreg(mem_reg,rec); bits_test(rec,exp)
END;

TEST38
FOR i:=1 TO 4 DO BEGIN {IR register}
  get_bits(test_bits[i],exp,0,36);
  wrreg(ir_reg,exp); rdreg(ir_reg,rec); bits_test(rec,exp)
END;

TEST39
FOR i:=1 TO 4 DO BEGIN {Q register}
  get_bits(test_bits[i],exp,0,36);
  wrreg(q_reg,exp); rdreg(q_reg,rec); bits_test(rec,exp)
END;

TEST40
FOR i:=1 TO 4 DO BEGIN {JMEMP register}
  get_bits(test_bits[i],exp,0,12); fit_bits(exp,36,1,right_fit);
  wrreg(jmemp_reg,exp); rdreg(jmemp_reg,rec); bits_test(rec,exp)
END;

TEST41
FOR i:=1 TO 4 DO BEGIN {DEV register}
  get_bits(test_bits[i],exp,0,5); fit_bits(exp,36,1,right_fit);
  wrreg(dev_reg,exp); rdreg(dev_reg,rec); bits_test(rec,exp)
END;

TEST42
FOR i:=1 TO 4 DO BEGIN {AMEMP register}
  get_bits(test_bits[i],exp,0,10); fit_bits(exp,36,1,right_fit);
  wrreg(amemp_reg,exp); rdreg(amemp_reg,rec); bits_test(rec,exp)
END;

TEST43
BEGIN {ac register}
  oct_to_bits('0',addr,4);
  FOR i:=1 TO 4 DO BEGIN
    get_bits(test_bits[i],exp,0,36);
    wrac(addr,exp); rdac(addr,rec); bits_test(rec,exp)
  END
END;

TEST44
BEGIN {amem}
  oct_to_bits('16',addr,10);
  FOR i:=1 TO 4 DO BEGIN
    get_bits(test_bits[i],exp,0,36);
    wram(addr,exp,FALSE); rdam(addr,rec,FALSE); bits_test(rec,exp)
  END
END;
```

```
TEST45
BEGIN {memory}
  oct_to_bits('30',addr,18);
  FOR i:=1 TO 4 DO BEGIN
    get_bits(test_bits[i],exp,0,36);
    wrm(addr,exp); rdm(addr,rec); bits_test(rec,exp)
  END
END

;MICRO INSTRUCTIONS

CLCPMOD_MI

CLEXCTX_MI

CLJMEMP_MI

CLMERGE_MI

CLUSCTX_MI

CLVAMOD_MI

DAM_MI

JUMP_MI

KLRES1_MI

KLRES2_MI

LDAC_MI

LDAMAR_MI

LDAMEMP_MI

LDAR_MI

LDDEV_MI

LDHOLD_MI

LDJMEMP_MI

LDIR_MI

LDLPQ_MI

LDMA_MI

LDPC_MI

LDPCF_MI

LDQ_MI
```

LDQONES_MI

LMHADR_MI

LMHBKAD_MI

LMHCNT_MI

LMHCTL_MI

LUHADR_MI

LUHCTL_MI

MAPDIS_MI

MEMST_MI

MHOFF_MI

MHON_MI

MRD_MI

UHOFF_MI

UHON_MI

XAC_MI

XAM_MI

XAMEMP_MI

XAR_MI

XCOND_MI

XDEV_MI

XIR_MI

XJMEM_MI

XJMEMP_MI

XLOOP_MI

XMA_MI

XMBST_MI

XMEM_MI

XMHADR_MI

XMHDATO_MI

XMHDAT1_MI

XPC_MI

XPCF_MI

XQ_MI

XUHADR_MI

XUHDAT_MI

ZERO_MI

;FIELDS

EEAL_FLD 0,1

EEFO_FLD 1,1

LDMA_FLD 2,1

IDISP_FLD 3,1

MWT_FLD 4,

SAAFMA_FLD 5,1

PO_FLD 6,1

SP1_FLD 7,1

CRY_FLD 8,1

ASRC_FLD 9,3

AFUN_FLD 12,3

ADST_FLD 15,3

ALU1_FLD 18,1

LDAR_FLD 19,1

JCOND_FLD 20,6

MAPF_FLD 26,4

SPC_FLD 30,6

JADR_FLD 36,14

ROT_FLD 50,6

MASK_FLD 56,6

```
DEST_FLD 62,6

SP2_FLD 68,1

JCODE_FLD 69,4

ACSEL_FLD 73,3

D_FLD 76,6

CYLEN_FLD 82,4

IFQ_FLD 86,1

DFQ_FLD 87,1

LIT_FLD 26,36

;REGISTERS

AMEM_REG 36,dam_mi,xam_mi

AMEMP_REG 10,ldamemp_mi,xamemp_mi

AR_REG 36,ldar_mi,xar_mi

DEV_REG 5,lddev_mi,xdev_mi

IR_REG 36,ldir_mi,xir_mi

JMEMP_REG 10,ldjmemp_mi,xjmemp_mi

JMEM_REG 14,zero_mi,xjmem_mi

LP_REG 16,zero_mi,xloop_mi

MA_REG 36,ldma_mi,xma_mi

MEM_REG 36,ldhold_mi,xmem_mi

PC_REG 36,ldpc_mi,xpc_mi

PCF_REG 36,ldpcf_mi,xpcf_mi

Q_REG 36,ldq_mi,xq_mi

MBST_REG 36,zero_mi,xmbst_mi

AC_REG 36,ldac_mi,xac_mi

HOLD_REG 36,ldhold_mi,xmem_mi

AM_REG 36,dam_mi,xam_mi
```

SOURCES

The console program is written in TURBO-PASCAL. This is a subset of standard PASCAL. TURBO does provide an OVERLAY capability and an I/O interface, but that's about all.

The sources consist of 5 files:

CONSOLE.PAS -- main source file, display manager

CONSOLE0.PAS -- utilities (bit converter)

CONSOLE1.PAS -- backend functions

CONSOLE2.PAS -- frontend commands

CONSOLE3.PAS -- table loader

The code takes up a lot of memory. Overlays are used where possible, but every addition of a new "feature" usually requires a restructuring to get it all to fit. As it is now, backend functions are overlayed with each other, as are frontend commands. The table loader is well overlayed with everything.

The source listing which follows is linear with all include files included. An index to functions and procedures is provided, along with the sources to the program which makes the listing.

The code assumes that the reader knows PASCAL well.

```
{CONSOLE.PAS -- source file for Augment Engine Console (1 of 5) zw}

{primary source file, contains definitions, variables, frontend utilities}

{other source files included are:
CONSOLE0.PAS -- utility routines
CONSOLE1.PAS -- the backend functions
CONSOLE2.PAS -- the frontend commands
CONSOLE3.PAS -- the table loaders}

{TURBO-PASCAL compiler switches, IBM-PC version}
{$R- subscript checks}
{$V- value checks}
{$U- xon/xoff}
{$I+ i/o check}
{$C- control-c}
{$K- stack check}

PROGRAM console;

CONST

{increment this for each edit}
edit='5';

{set this true if debugging initial start-up problems}
initial_debugging=FALSE;

{major version must match between local and remote, edit can be different}
version='2.6';

{the main command prompt}
prompt='!';

{colors for various display items}
box_color=BROWN; banner_color=GREEN; time_color=CYAN; modem_color=WHITE;
dat_color:ARRAY[BOOLEAN] OF BYTE=(YELLOW,CYAN); hole_color=LIGHTRED;
msg_color=YELLOW; scroll_color=LIGHTGRAY; ask_color=WHITE;
alert_color=LIGHTRED;

{maximum number of items allocated}
number_counters=100; number_tests=45; number_macros=100;

{allocation sizes of items}
max_str_siz=128; max_bits_index=127; max_bits_siz=128; max_bit_siz=16;

{com port codes}
con=$00; com1=$01; com2=$02;

{micro instruction section codes -- binary bits}
sect_A=4; sect_B=2; sect_C=1; sect_ABC=7;

{micro/macro loader function codes}
load_slow=0; load_verify=1; load_check=2; load_fast=3;

TYPE
```

```
{8088 cpu register type for calls to MSDOS or INTR}
regs_typ=RECORD ax,bx,cx,dx,bp,si,ds,es,flags:INTEGER END;

{general purpose string type}
str_typ=STRING[max_str_siz];

{remote/local talk buffer of lines}
talk_typ=ARRAY[1..12] OF str_typ;

{binary record type for micro load files}
bin_load_typ=RECORD
  addr:INTEGER; {address of micro instruction}
  dat:ARRAY[0..10] OF BYTE {ten 8-bit bytes (88 bits) of micro instruction}
END;

{display format type}
dpy_fmt_typ=(line_fmt,reg_fmt,screen_fmt);

{set of codes for idle display}
idle_typ=SET OF (idle_mic,idle_obus,idle_dma);

{string and bit justification codes}
fit_typ=(left_fit,center_fit,right_fit);

{general purpose bit structure}
bits_typ=RECORD
  bit:ARRAY [0..max_bits_index] OF INTEGER; {value of general bit}
  siz:1..max_bits_siz; {number of general bits in structure}
  bit_siz:1..max_bit_siz {number of binary bits in general bit}
END;

{micro instruction identifiers to index table}
mi_typ=(
  clcpmod_mi,clexctx_mi,cljmemp_mi,clmerge_mi,clusctx_mi,clvamod_mi,dam_mi,
  jump_mi,klres1_mi,klres2_mi,ldac_mi,ldamar_mi,ldamemp_mi,ldar_mi,lddev_mi,
  ldhold_mi,ldir_mi,ldjmemp_mi,ldlpq_mi,ldma_mi,ldpc_mi,ldpcf_mi,ldq_mi,
  ldqones_mi,lmhadr_mi,lmhbkad_mi,lmhcnt_mi,lmhctl_mi,luhadr_mi,luhctl_mi,
  mapdis_mi,memst_mi,mhoff_mi,mhon_mi,mrd_mi,uhoff_mi,uhon_mi,xac_mi,xam_mi,
  xamemp_mi,xar_mi,xcond_mi,xdev_mi,xir_mi,xjmem_mi,xjmemp_mi,xloop_mi,xma_mi
  xmbst_mi,xmem_mi,xmhadr_mi,xmhdat0_mi,xmhdat1_mi,xpc_mi,xpcf_mi,xq_mi,
  xuhadr_mi,xuhdat_mi,zero_mi);

{micro instruction field identifiers to index table}
fld_typ=(
  fld_err,eeal_fld,eefo_fld,ldma_fld,idisp_fld,mwt_fld,saafma_fld,po_fld,
  sp1_fld,cry_fld,asrc_fld,afun_fld,adst_fld,alu1_fld,ldar_fld,jcond_fld,
  mapf_fld,spc_fld,jadr_fld,rot_fld,mask_fld,dest_fld,sp2_fld,jcode_fld,
  acsel_fld,d_fld,cylen_fld,ifq_fld,dfq_fld,lit_fld);

{machine register identifiers to index table}
reg_typ=(
  reg_err,amem_reg,amemp_reg,ar_reg,dev_reg,ir_reg,jmemp_reg,jmem_reg,lp_reg,
  ma_reg,mem_reg,pc_reg,pcf_reg,q_reg,mbst_reg,ac_reg,hold_reg,am_reg);

{display item identifiers}
dpy_typ=(
  line1_dpy,line2_dpy,line3_dpy,line4_dpy,line5_dpy,line6_dpy,line7_dpy,
```

```
  line8_dpy,line9_dpy,line10_dpy,line11_dpy,line12_dpy,load_dpy,run_dpy,
  remote_dpy,mbrk_dpy,mad_dpy,m_dpy,ecc_dpy,dmaad_dpy,dma_dpy,pc_dpy,
  ma_dpy,ar_dpy,mem_dpy,ir_dpy,q_dpy,lp_dpy,acad_dpy,ac_dpy,jmemp_dpy,
  jmem_dpy,dev_dpy,amemp_dpy,amemad_dpy,amem_dpy,ubrk_dpy,mic_dpy,obus_dpy,
  eobus_dpy,lc_dpy,tc_dpy,br_dpy,mi_dpy,eeal_dpy,eefo_dpy,ldma_dpy,idisp_dpy,
  mwt_dpy,saafma_dpy,po_dpy,sp1_dpy,cry_dpy,asrc_dpy,afun_dpy,adst_dpy,
  alu1_dpy,ldar_dpy,jcond_dpy,mapf_dpy,spc_dpy,jadr_dpy,rot_dpy,mask_dpy,
  dest_dpy,sp2_dpy,jcode_dpy,acsel_dpy,d_dpy,cylen_dpy,ifq_dpy,dfq_dpy,
  lit_dpy,dd_dpy,dpy_init);

{function/command identifiers}
cmd_typ=(
  bits_fun,break_fun,build_fun,cpuob_fun,cty_fun,dump_fun,enahst_fun,
  exec_fun,fld_fun,init_fun,init2_fun,inits_fun,instrs_fun,iora_fun,
  iorb_fun,jump_fun,load_fun,mi_fun,mmactrl_fun,rdac_fun,rdam_fun,rddma_fun,
  rdecc_fun,rdhst_fun,rdhsti_fun,rdm_fun,rdmi_fun,rdmic_fun,rdob_fun,
  rdport_fun,rdreg_fun,rdst_fun,rdstlp_fun,rdum_fun,reg_fun,reset_fun,
  run_fun,save_fun,setfld_fun,shift_fun,step_fun,sysid_fun,talk_fun,
  test_fun,wrac_fun,wram_fun,wrdma_fun,wrhsti_fun,wrm_fun,wrmi_fun,
  wrmic_fun,wrob_fun,wrport_fun,wrreg_fun,wrum_fun,
  boot_cmd,build_cmd,call_cmd,cs_cmd,cty_cmd,dac_cmd,dam_cmd,dami_cmd,
  damp_cmd,dd_cmd,de_cmd,dmrd_cmd,dmwrt_cmd,dump_cmd,ex_cmd,help_cmd,
  history_cmd,idle_cmd,ldac_cmd,ldamemp_cmd,ldar_cmd,lddev_cmd,ldjmemp_cmd,
  ldhold_cmd,ldir_cmd,ldma_cmd,ldpc_cmd,ldpcf_cmd,ldq_cmd,load_cmd,
  macro_cmd,mbrk_cmd,mgo_cmd,mic_cmd,miload_cmd,mird_cmd,mmload_cmd,
  mmrd_cmd,modem_cmd,mstart_cmd,quit_cmd,rdobus_cmd,rdstatus_cmd,remote_cmd,
  reset_cmd,run_cmd,set_cmd,ss_cmd,stop_cmd,talk_cmd,test_cmd,type_cmd,
  ubrk_cmd,ugo_cmd,ui_cmd,ustart_cmd,wrobus_cmd,xac_cmd,xam_cmd,
  xami_cmd,xamp_cmd,xamemp_cmd,xar_cmd,xdev_cmd,xir_cmd,xjmem_cmd,
  xjmemp_cmd,xlp_cmd,xma_cmd,xmem_cmd,xpc_cmd,xpcf_cmd,xq_cmd);

VAR

{constant tables -- loaded from overlay}

{on/off/odd/even test data patterns}
test_byte:ARRAY[1..4] OF BYTE; test_bits:ARRAY[1..4] OF bits_typ;

{table of micro instructions -- constant for backend}
mi_tab:ARRAY[mi_typ] OF RECORD
  name:STRING[8]; {text name}
  bits:bits_typ {88 bits -- optimal is ten 8-bit bytes}
END;

{table of micro instruction fields -- constant for backend}
fld_tab:ARRAY[fld_typ] OF RECORD
  name:STRING[6]; {text name}
  bit_zero:0..87; {position of field bit zero within instruction}
  bits_siz:1..36; {number of bits in field}
  dpy:dpy_typ {display item for field}
END;

{table of machine registers -- constant for backend}
reg_tab:ARRAY[reg_typ] OF RECORD
  name:STRING[5]; {text name}
  bits_siz:1..36; {number of bits in register}
  mi_to_wr,mi_to_rd:mi_typ {micro instructions to write and read register}
```

```
END;

{table of commands -- constant for frontend}
cmd_tab:ARRAY[cmd_typ] OF RECORD
  name:STRING[8]; eq_siz:1..8; {name and eqiated minimum size}
  arg1_desc,arg2_desc,cmd_desc:STRING[40] {descriptions}
  {see procedure command_ok for interpretation of descriptions}
END;

{dable of display items -- constant for frontend}
dpy_tab:ARRAY[dpy_typ] OF RECORD {table of display items -- semi-constant}
  x_pos:1..80; y_pos:1..12; {position of item in data display}
  desc:STRING[7]; {description text}
  value:STRING[80]; {value text -- not constant}
  value_siz:0..80; {value field width, in characters}
END;

{global variables -- to be accessed by both backend and frontend}

{digit and character tables for conversion between bits and strings}
digit_tab:ARRAY[CHAR] OF BYTE; {convert character to value}
char_tab:ARRAY[0..16] OF CHAR; {convert value to character}

{activity counter}
activity_counter:ARRAY[1..number_counters] OF INTEGER; {the counters}
counting:BOOLEAN; {if true then increment counters}

{debugger}
debugging:BOOLEAN; {if true then display debug information}
debug_counter:INTEGER; {number of debug calls to skip}

{digit/bit conversion errors}
bad_digit, {if true then invalid digit was encountered}
digit_overflow:BOOLEAN; {if true then justification overflowed}

{system type flag -- if true then KL version}
kl:BOOLEAN;

{remote access state -- remote_cty set by backend}
remote:(we_are,hello,entry,login,remote_cty,active);

{log file}
log_file:TEXT; {the log file}
log_open, {if true then log file is open}
log_ok:BOOLEAN; {if true then write log messages}

PROCEDURE debug_break; BEGIN END; {a place to set IBM-PC DEBUG break point}
{set break point to address obtained by: debug(hex_word(OFS(debug_break));
hit the break point with: debug(hex_word(OFS(routine_name)); debug_break;
the routine address is then displayed and the correct overlay is loaded}

{$I CONSOLE0.PAS utility routines}
{CONSOLE0.PAS -- source file for Augment Engine Console (2 of 5) zw}

{contains global utilities}

FUNCTION time(get_time: BOOLEAN): str_typ;
```

```
LISTING: CONSOLE.PAS  01-JAN-80  00:00:00  CONSOLE.PAS;

{if get_time then return time of day else return current date}
CONST
time_code=$2C; date_code=$2A;
month: ARRAY[1..12] OF STRING[3] =
  ('Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec');
VAR regs:regs_typ; d,h,m,s:STRING[2]; y:STRING[4];
BEGIN
  WITH regs DO BEGIN
    IF get_time THEN ax:=time_code SHL 8 ELSE ax:=date_code SHL 8;
    MSDOS(regs);
    IF get_time THEN BEGIN
      STR(cx SHR 8:0, h); STR(cx AND $FF:0, m); STR(dx SHR 8:0, s);
      IF LENGTH(h)<2 THEN h:='0'+h;
      IF LENGTH(m)<2 THEN m:='0'+m;
      IF LENGTH(s)<2 THEN s:='0'+s;
      time:=h+':'+m+':'+s
    END
    ELSE BEGIN
      STR(cx:4,y); STR(dx AND $FF:0, d); y:=COPY(y,3,2);
      IF LENGTH(d)<2 THEN d:='0'+d;
      time:=d+'-'+month[dx SHR 8]+'-'+y
    END
  END
END;

PROCEDURE count(c:BYTE);
{increment specified activity counter if counting}
BEGIN IF counting THEN activity_counter[c]:=activity_counter[c]+1 END;

PROCEDURE dpy_str(x,y:BYTE; s:str_typ; color:BYTE);
{display colored string at x,y location on screen, upper left is 1,1}
BEGIN TEXTCOLOR(color); GOTOXY(x,y); WRITE(s) END;

PROCEDURE alert(m:str_typ);
{display alert message at bottom of screen}
BEGIN dpy_str(1,24,'',alert_color); CLREOL; WRITE(m) END;

PROCEDURE log(m:str_typ);
{if log file open, write message to it}
BEGIN
  IF log_ok THEN BEGIN
    IF log_open THEN BEGIN
      {$I-} WRITELN(log_file,time(TRUE)+' '+m); {$I+}
      log_open:=IORESULT=0; IF NOT log_open THEN CLOSE(log_file)
    END;
    IF NOT log_open THEN alert('LOG FULL. '+^G)
  END
END;

PROCEDURE debug(m:str_typ);
{if debugging and debug_counter not zero then display debug message}
BEGIN
  IF debugging THEN BEGIN
    IF debug_counter>0 THEN debug_counter:=debug_counter-1 ELSE BEGIN
      log(m); alert(m+' #'); {$I-} READLN(debug_counter); {$I+} alert('')
    END
  END
```

```
LISTING: CONSOLE.PAS  01-JAN-80  00:00:00  (CONSOLE.PAS)

END;

FUNCTION debug_bits(VAR b:bits_typ):str_typ;
{debug bits structure by converting all info to string for display}
VAR i:0..max_bits_index; siz,bit_siz,bit,bits:str_typ;
BEGIN
  STR(b.siz:0,siz); STR(b.bit_siz:0,bit_siz); STR(b.bit[0]:0,bit); bits:=bit;
  FOR i:=1 TO b.siz-1 DO BEGIN STR(b.bit[i]:0,bit); bits:=bits+' '+bit END;
  debug_bits:='('+siz+' '+bit_siz+' ('+bits +'))'
END;

FUNCTION min(x,y:INTEGER):INTEGER;
{return the minimum of two integers}
BEGIN IF x<y THEN min:=x ELSE min:=y END;

FUNCTION max(x,y:INTEGER):INTEGER;
{return the maximum of two integers}
BEGIN IF x>y THEN max:=x ELSE max:=y END;

FUNCTION uppercase(s:str_typ):str_typ;
{return string converted to uppercase}
VAR i:1..max_str_siz;
BEGIN FOR i:=1 TO LENGTH(s) DO s[i]:=UPCASE(s[i]); uppercase:=s END;

FUNCTION blanks(n:BYTE):str_typ;
{return the specified number of blanks}
VAR s:str_typ; i:1..max_str_siz;
BEGIN s:=''; FOR i:=1 TO min(n,max_str_siz) DO s:=s+' '; blanks:=s END;

FUNCTION eq(s1,s2:str_typ; n:BYTE):BOOLEAN;
{return true if string s1 is an eqiation of s2 of minimum size n}
BEGIN eq:=(LENGTH(s1)>=n) AND (s1=COPY(s2,1,LENGTH(s1))) END;

FUNCTION switch(x:BOOLEAN; s1,s2:str_typ):str_typ;
{switch strings -- if true then return s1 else return s2}
BEGIN IF x THEN switch:=s1 ELSE switch:=s2 END;

FUNCTION parse(VAR a:str_typ):str_typ;
{parse input string, return token, remove token from input string}
CONST delimiters:SET OF CHAR=[' ',',','/'];
VAR i:1..max_str_siz;
BEGIN
  i:=1; WHILE (i<=LENGTH(a)) AND (a[i] IN delimiters) DO i:=i+1;
  DELETE(a,1,i-1);
  i:=1; WHILE (i<=LENGTH(a)) AND NOT (a[i] IN delimiters) DO i:=i+1;
  parse:=COPY(a,1,i-1);
  DELETE(a,1,i-1);
  i:=1; WHILE (i<=LENGTH(a)) AND (a[i] IN delimiters) DO i:=i+1;
  DELETE(a,1,i-1)
END;

FUNCTION unparse(a1,a2,a3:str_typ):str_typ;
{return appended tokens suitable for parse}
VAR a:str_typ;
BEGIN unparse:=a1+switch(a2<>'',',',a2+switch(a3<>'',',',a3,''),'') END;

PROCEDURE default(VAR a:str_typ; v:str_typ);
```

```
{set default value, v, if input string is null or '.'}
BEGIN IF (a='') OR (a='.') THEN a:=v END;

FUNCTION fit(s:str_typ; n:BYTE; f:fit_typ):str_typ;
{fit string into specified size with justification}
BEGIN
  DELETE(s,n+1,max_str_siz);
  CASE f OF
    center_fit:s:=blanks((n-LENGTH(s)) DIV 2)+s;
    right_fit:s:=blanks(n-LENGTH(s))+s
  END;
  fit:=s+blanks(n-LENGTH(s))
END;

PROCEDURE fit_bits(VAR bits:bits_typ; new_siz,new_bit_siz:BYTE;
  justify:fit_typ);
{fit bits structure into specified configuration with justification}
VAR
bin:ARRAY [0..max_bits_index] OF 0..1;
i,j:0..max_bits_index;
lsb,new_lsb,bits_given,bits_used,difference,value:INTEGER;
BEGIN
  {it works so don't mess with it}
  IF (bits.siz<>new_siz) OR (bits.bit_siz<>new_bit_siz) THEN BEGIN
    bits_given:=0;
    FOR i:=0 TO bits.siz-1 DO BEGIN
      value:=bits.bit[i];
      IF bits_given<=max_bits_index THEN
      FOR j:=bits_given+bits.bit_siz-1 DOWNTO bits_given DO BEGIN
        IF j<=max_bits_index THEN bin[j]:=value AND 1;
        value:=value SHR 1
      END
      ELSE digit_overflow:=TRUE;
      bits_given:=bits_given+bits.bit_siz
    END;
    lsb:=bits_given-1; new_lsb:=(new_siz*new_bit_siz)-1;
    IF lsb>max_bits_index THEN lsb:=max_bits_index;
    IF new_lsb>max_bits_index THEN new_lsb:=max_bits_index;
    difference:=new_lsb-lsb;
    IF justify=right_fit THEN BEGIN {align least significant bit}
      IF difference>0 THEN BEGIN {new_lsb to right of lsb}
        {shift right}
        FOR i:=new_lsb DOWNTO difference DO bin[i]:=bin[i-difference];
        {pad left with zero's}
        FOR i:=0 TO difference-1 DO bin[i]:=0
      END
      ELSE IF difference<0 THEN BEGIN {new_lsb to left of lsb}
        {verify no one's lost on left}
        {
        FOR i:=0 TO -difference-1 DO IF bin[i]<>0 THEN digit_overflow:=TRUE;
        }
        {shift left}
        FOR i:=0 TO lsb DO bin[i]:=bin[i-difference]
      END
    END
    ELSE BEGIN {align most significant bit}
      IF difference>0 THEN BEGIN {new_lsb to right of lsb}
```

```
        {pad right with zero's}
        FOR i:=lsb+1 TO new_lsb DO bin[i]:=0
      END
      ELSE IF difference<0 THEN BEGIN {new_lsb to left of lsb}
        {verify no one's lost on right}
        {
        FOR i:=new_lsb+1 TO lsb DO IF bin[i]<>0 THEN digit_overflow:=TRUE
        }
      END
    END;
    bits.siz:=new_siz; bits.bit_siz:=new_bit_siz; bits_used:=0;
    FOR i:=0 TO bits.siz-1 DO BEGIN
      value:=0;
      FOR j:=bits_used TO bits_used+bits.bit_siz-1
      DO value:=(value SHl 1) OR bin[j];
      bits.bit[i]:=value; bits_used:=bits_used+bits.bit_siz
    END
  END
END;



FUNCTION equal_bits(VAR b1,b2:bits_typ):BOOLEAN;
{return true if bit structures are equivalent -- does not fit_bits}
VAR ok:BOOLEAN; i:0..max_bits_index;
BEGIN
  ok:=(b1.siz=b2.siz) AND (b1.bit_siz=b2.bit_siz);
  IF ok THEN FOR i:=0 TO b1.siz-1 DO ok:=ok AND (b1.bit[i]=b2.bit[i]);
  equal_bits:=ok
END;

PROCEDURE put_bits(VAR b,bb:bits_typ; p,l:BYTE);
{put l bits from bb into b at position p}
VAR i:0..max_bits_index;
BEGIN
  fit_bits(b,b.siz*b.bit_siz,1,left_fit);
  fit_bits(bb,bb.siz*bb.bit_siz,1,left_fit);
  FOR i:=0 TO l-1 DO b.bit[p+i]:=bb.bit[bb.siz-1+i]
END;

PROCEDURE get_bits(VAR b,bb:bits_typ; p,l:BYTE);
{get l bits from b at position p into bb}
VAR i:0..max_bits_index;
BEGIN
  fit_bits(b,b.siz*b.bit_siz,1,left_fit);
  bb.siz:=l; bb.bit_siz:=1;
  FOR i:=p TO p+l-1 DO bb.bit[i-p]:=b.bit[i]
END;

FUNCTION char_to_digit(c:CHAR; s:BYTE):BYTE;
{convert character to digit of specified bit size}
VAR i:BYTE;
BEGIN
  i:=digit_tab[c];
  IF (i<0) OR (i>=(1 SHL s)) THEN BEGIN i:=0; bad_digit:=TRUE END;
  char_to_digit:=i
END;
```

```
FUNCTION digit_to_char(i:BYTE; s:BYTE):CHAR;
{convert digit of specified bit size to character}
BEGIN
  IF (i>=0) AND (i<(1 SHL s)) THEN digit_to_char:=char_tab[i]
  ELSE BEGIN digit_to_char:='0'; bad_digit:=TRUE END
END;

FUNCTION bits_to_oct(b:bits_typ; s:BYTE):str_typ;
{return s bits, b, converted to octal digit string}
VAR i,j:0..max_bits_index; digits:str_typ;
BEGIN
  fit_bits(b,s,1,left_fit);
  fit_bits(b,(s+2) DIV 3,3,right_fit);
  j:=0; WHILE (j<b.siz-1) AND (b.bit[j]=0) DO j:=j+1;
  digits:=''; bad_digit:=FALSE;
  FOR i:=max(j,b.siz-max_str_siz) TO b.siz-1 DO
    digits:=digits+char_tab[b.bit[i]];
  IF b.siz>max_str_siz THEN digit_overflow:=TRUE;
  bits_to_oct:=digits
END;

PROCEDURE oct_to_bits(d:str_typ; VAR b:bits_typ; s:BYTE);
{convert octal digit string to s bits, b}
VAR i:0..max_bits_index; special:INTEGER;
BEGIN
  special:=770; bad_digit:=FALSE;
  IF d='OFF' THEN special:=0
  ELSE IF d='ON' THEN special:=-1
  ELSE IF d='-1' THEN special:=-1
  ELSE IF d='ODD' THEN special:=21845
  ELSE IF d='EVEN' THEN special:=-21846;
  IF special=770 THEN BEGIN
    b.siz:=min(LENGTH(d),max_bits_siz);
    b.bit_siz:=3;
    FOR i:=0 TO b.siz-1 DO b.bit[i]:=digit_tab[d[i+1]];
    IF LENGTH(d)>max_bits_siz THEN digit_overflow:=TRUE;
    fit_bits(b,s,1,right_fit)
  END
  ELSE BEGIN
    b.siz:=max_bits_siz DIV max_bit_siz; b.bit_siz:=max_bit_siz;
    FOR i:=0 TO b.siz-1 DO b.bit[i]:=special;
    fit_bits(b,s,1,right_fit); digit_overflow:=FALSE
  END
END;

FUNCTION int_oct(i:INTEGER):str_typ;
{return octal digit string converted from integer}
VAR b:bits_typ; digits:str_typ;
BEGIN b.siz:=i; b.bit_siz:=16; b.bit[0]:=i; int_oct:=bits_to_oct(b,16) END;

FUNCTION oct_int(d:str_typ):INTEGER;
{return integer converted from octal digit string}
VAR b:bits_typ;
BEGIN oct_to_bits(d,b,16); fit_bits(b,1,16,left_fit); oct_int:=b.bit[0] END;

FUNCTION int_dec(i:INTEGER):str_typ;
```

```
{return decimal digit string converted from integer}
VAR digits:str_typ; BEGIN STR(i:0,digits); int_dec:=digits END;

FUNCTION dec_int(d:str_typ):INTEGER;
{return integer converted from decimal digit string}
VAR i,e:INTEGER;
BEGIN
  VAL(d,i,e); IF e<>0 THEN BEGIN i:=0; bad_digit:=TRUE END; dec_int:=i
END;

FUNCTION incr_oct(d:str_typ):str_typ;
{return incremented octal digit string}
VAR i:0..max_str_siz; carry,new_value:BYTE;
BEGIN
  carry:=1; i:=LENGTH(d); bad_digit:=FALSE;
  WHILE (i>=1) AND (carry>0) DO BEGIN
    new_value:=digit_tab[d[i]]+carry;
    IF new_value>=8 THEN new_value:=0 ELSE carry:=0;
    d[i]:=char_tab[new_value]; i:=i-1
  END;
  IF carry=1 THEN BEGIN
    IF LENGTH(d)<max_str_siz THEN d:='1'+d ELSE digit_overflow:=TRUE
  END;
  incr_oct:=d
END;

FUNCTION oct_bits(d:str_typ; s:BYTE):str_typ;
{return verified octal digit string of specified bit size}
VAR b:bits_typ; BEGIN oct_to_bits(d,b,s); oct_bits:=bits_to_oct(b,s) END;

FUNCTION oct_digits(d:str_typ; s:BYTE):str_typ;
{return octal digit string of specified size, maybe add leading zeros}
VAR i:1..max_str_siz;
BEGIN
  IF s<LENGTH(d) THEN DELETE(d,1,LENGTH(d)-s)
  ELSE FOR i:=1 TO s-LENGTH(d) DO d:='0'+d;
  FOR i:=1 TO LENGTH(d) DO d[i]:=digit_to_char(char_to_digit(d[i],3),3);
  oct_digits:=d
END;

FUNCTION oct_word(d:str_typ):str_typ;
{if octal word, return halfwords seperated by double comma}
BEGIN IF LENGTH(d)=12 THEN INSERT(',,',d,7); oct_word:=d END;

FUNCTION hex_byte(i:BYTE):str_typ;
{convert byte to hex digit string}

FUNCTION nibble(i:BYTE):str_typ;
{convert nibble to hex digit string}
BEGIN
  i:=i AND $0F;
  IF i<10 THEN nibble:=CHR(ORD('0')+i) ELSE nibble:=CHR(ORD('A')+i-10)
END;

BEGIN {hex_byte}
  hex_byte:=nibble(i SHR 4)+nibble(i)
END;
```

```
FUNCTION hex_word(i:INTEGER):str_typ;
{convert 16-bit word to hex digit string}
BEGIN hex_word:=hex_byte(i SHR 8)+hex_byte(i) END;

{the following routines make up the interrupt driven communications package}

VAR {communications stuff}
com1_seg,com1_ofs,com2_seg,com2_ofs:INTEGER; {saved values}
com1_id,com2_id:BYTE; {from last interrupt}
com1_input,com2_input:ARRAY[0..$FF] OF BYTE; {input buffers}
com1_input_in,com2_input_in,com1_input_out,com2_input_out:0..$FF; {pointers}
com1_input_siz,com2_input_siz:0..255; {current size of buffers}
cty,modem:BYTE; {configured port codes}

PROCEDURE com1_server;
{IRQ4 server for com1}
CONST
int_control=$20; eoi_bits=$20;
com1_data=$3F8; com1_int_id=$3FA; receive_int=$04;
BEGIN
  {
  ;pushes for entry
  PUSH AX ! PUSH BX ! PUSH CX ! PUSH DX ! PUSH DI ! PUSH SI ! PUSH ES ! PUSH D
  MOV BX,2700
  CS: ! MOV AX,[0110] ! MOV DS,AX
  }
  INLINE($50/$53/$51/$52/$57/$56/$06/$1E/$BB/$00/$27/$2E/$A1/$10/$01/$8E/$D8);
  com1_id:=PORT[com1_int_id];
  IF (com1_id AND receive_int)<>0 THEN BEGIN
    IF com1_input_siz>=$FF THEN BEGIN
      com1_input_out:=(com1_input_out+1) AND $FF;
      com1_input_siz:=com1_input_siz-1
    END
    ELSE IF com1_input_siz=0 THEN com1_input_in:=com1_input_out;
    com1_input[com1_input_in]:=PORT[com1_data];
    com1_input_in:=(com1_input_in+1) AND $FF;
    com1_input_siz:=com1_input_siz+1
  END;
  PORT[int_control]:=eoi_bits;
  {
  ;pops for exit
  POP DS ! POP ES ! POP SI ! POP DI ! POP DX ! POP CX ! POP BX ! POP AX
  POP BP ! POP BP ! IRET
  }
  INLINE($1F/$07/$5E/$5F/$5A/$59/$5B/$58/$5D/$5D/$CF)
END;

PROCEDURE com2_server;
{IRQ3 server for com2}
CONST
int_control=$20; eoi_bits=$20;
com2_data=$2F8; com2_int_id=$2FA; receive_int=$04;
BEGIN
  INLINE($50/$53/$51/$52/$57/$56/$06/$1E/$BB/$00/$27/$2E/$A1/$10/$01/$8E/$D8);
  com2_id:=PORT[com2_int_id];
  IF (com2_id AND receive_int)<>0 THEN BEGIN
```

```
    IF com2_input_siz>=$FF THEN BEGIN
      com2_input_out:=(com2_input_out+1) AND $FF;
      com2_input_siz:=com2_input_siz-1
    END
    ELSE IF com2_input_siz=0 THEN com2_input_in:=com2_input_out;
    com2_input[com2_input_in]:=PORT[com2_data];
    com2_input_in:=(com2_input_in+1) AND $FF;
    com2_input_siz:=com2_input_siz+1
  END;
  PORT[int_control]:=eoi_bits;
  INLINE($1F/$07/$5E/$5F/$5A/$59/$5B/$58/$5D/$5D/$CF)
END;

FUNCTION receive(p:BYTE; VAR ch:CHAR):BOOLEAN;
{return true if character received from specified port}
VAR regs:regs_typ;
BEGIN
  receive:=FALSE;
  CASE p OF
    con:WITH regs DO BEGIN
      ax:=$0600; dx:=$FF; MSDOS(regs); ch:=CHR(ax AND $7F); receive:=ch<>#0
    END;
    com1:IF com1_input_siz>0 THEN BEGIN
      ch:=CHR(com1_input[com1_input_out] AND $7F);
      com1_input_out:=(com1_input_out+1) AND $FF;
      com1_input_siz:=com1_input_siz-1; receive:=TRUE
    END;
    com2:IF com2_input_siz>0 THEN BEGIN
      ch:=CHR(com2_input[com2_input_out] AND $7F);
      com2_input_out:=(com2_input_out+1) AND $FF;
      com2_input_siz:=com2_input_siz-1; receive:=TRUE
    END
  END
END;

PROCEDURE send(p:BYTE; ch:CHAR);
{send character to specified port}
CONST
com1_data=$3F8; com1_line_status=$3FD; com2_data=$2F8; com2_line_status=$2FD;
transmit_ready=$20;
VAR regs:regs_typ;
BEGIN
  CASE p OF
    con:WITH regs DO BEGIN ax:=$0600; dx:=ORD(ch); MSDOS(regs) END;
    com1:BEGIN
      REPEAT UNTIL (PORT[com1_line_status] AND transmit_ready)<>0;
      PORT[com1_data]:=ORD(ch)
    END;
    com2:BEGIN
      REPEAT UNTIL (PORT[com2_line_status] AND transmit_ready)<>0;
      PORT[com2_data]:=ORD(ch)
    END
  END
END;

PROCEDURE com_init(p,baud,parity,stopbit,wordlength:INTEGER);
{set up a port with baud, parity, stop bit and word length parameters}
```

```
CONST
rs232_code=$14;
com1_int_enable=$3F9; com2_int_enable=$2F9; receive_int=$01;
com1_modem_control=$3FC; com2_modem_control=$2FC; out2=$08;
VAR regs:regs_typ;
BEGIN
  WITH regs DO CASE p OF com1:dx:=0; com2:dx:=1; ELSE dx:=0 END;
  CASE baud OF
    110:baud:=0; 150:baud:=1; 300:baud:=2;
    600:baud:=3; 1200:baud:=4; 2400:baud:=5;
    4800:baud:=6; 9600:baud:=7; ELSE baud:=2
  END;
  CASE parity OF 0,1:; 2:parity:=3 ELSE parity:=0 END;
  CASE stopbit OF 1:stopbit:=0; 2:stopbit:=1; ELSE stopbit:=0 END;
  CASE wordlength OF 7:wordlength:=2; 8:wordlength:=3; ELSE wordlength:=2 END;
  regs.ax:=(baud SHL 5) OR (parity SHL 3) OR (stopbit SHL 2) OR wordlength;
  INTR(rs232_code,regs);
  CASE p OF
    com1:BEGIN
      PORT[com1_int_enable]:=PORT[com1_int_enable] OR receive_int;
      PORT[com1_modem_control]:=PORT[com1_modem_control] OR out2
    END;
    com2:BEGIN
      PORT[com2_int_enable]:=PORT[com2_int_enable] OR receive_int;
      PORT[com2_modem_control]:=PORT[com2_modem_control] OR out2
    END
  END
END;

PROCEDURE connect(p:BYTE; initiate:BOOLEAN);
{if initiate, then set up connection else break connection}
CONST
dtr_bit=$01; rts_bit=$02; cts_bit=$10; dsr_bit=$20; ri_bit=$40; rlsd_bit=$80;
VAR c,s:INTEGER; x:CHAR;
BEGIN
  CASE p OF
    com1:BEGIN c:=$3FC; s:=$3FE END; com2:BEGIN c:=$2FC; s:=$2FE END
  END;
  alert('COM'); WRITE(p:0);
  IF initiate THEN BEGIN
    PORT[c]:=PORT[c] AND NOT (rts_bit OR dtr_bit); DELAY(1000);
    PORT[c]:=PORT[c] OR dtr_bit; WRITE(' DSR?');
    REPEAT UNTIL KEYPRESSED OR ((PORT[s] AND dsr_bit)<>0);
    PORT[c]:=PORT[c] OR rts_bit; WRITE(' CTS?');
    REPEAT UNTIL KEYPRESSED OR ((PORT[s] AND cts_bit)<>0);
    WRITE(' RLDS?');
    REPEAT UNTIL KEYPRESSED OR ((PORT[s] AND rlsd_bit)<>0);
    IF KEYPRESSED THEN BEGIN READ(KBD,x); WRITE(' ABORT') END ELSE WRITE(' ON'
  END
  ELSE BEGIN
    WRITE(' OFF');
    PORT[c]:=PORT[c] AND NOT dtr_bit; DELAY(1000);
    PORT[c]:=PORT[c] OR dtr_bit
  END
END;

FUNCTION connected(p:BYTE):BOOLEAN;
```

```
{return true if port is connected}
CONST cts_bit=$10; dsr_bit=$20; ri_bit=$40; rlsd_bit=$80;
VAR s,l:INTEGER; foo:BYTE;
BEGIN
  CASE p OF
    com1:BEGIN l:=$3FD; s:=$3FE END; com2:BEGIN l:=$2FD; s:=$2FE END
  END;
  foo:=PORT[l];
  connected:=((NOT PORT[s]) AND (rlsd_bit OR dsr_bit OR cts_bit))=0
END;

OVERLAY PROCEDURE interrupts(enable:BOOLEAN);
{if enable then set up communications else restore before program exit}
CONST
irq3_code=$0B; irq4_code=$0C; int_control=$20; eoi_bits=$20;
int_enable=$21; irq3_bit=$08; irq4_bit=$10;
com1_data=$3F8; com2_data=$2F8;
com1_int_enable=$3F9; com2_int_enable=$2F9; receive_int=$01;
com1_int_id=$3FA; com2_int_id=$2FA;
com1_line_control=$3FB; com2_line_control=$2FB; dlab=$80;
com1_modem_control=$3FC; com2_modem_control=$2FC; out2=$08;
BEGIN
  IF enable THEN BEGIN
    {set up the input buffer pointers}
    com1_id:=0; com1_input_in:=0; com1_input_out:=0; com1_input_siz:=0;
    com2_id:=0; com2_input_in:=0; com2_input_out:=0; com2_input_siz:=0;
    {save DS register value in TURBO copyright notice area,CS:110}
    {MOV AX,DS ! CS: ! MOV [0110],AX}
    INLINE($8C/$D8/$2E/$A3/$10/$01);
    {initially make sure irq3 and irq4 are ignored while servers set}
    PORT[int_enable]:=PORT[int_enable] OR irq3_bit OR irq4_bit;
    {save current server addresses, set up local servers}
    com1_seg:=MEMW[0:irq4_code*4+2]; com1_ofs:=MEMW[0:irq4_code*4];
    com2_seg:=MEMW[0:irq3_code*4+2]; com2_ofs:=MEMW[0:irq3_code*4];
    MEMW[0:irq4_code*4+2]:=CSEG; MEMW[0:irq4_code*4]:=OFS(com1_server);
    MEMW[0:irq3_code*4+2]:=CSEG; MEMW[0:irq3_code*4]:=OFS(com2_server);
    {tell the interrupt controller to listen for irq3 and irq4}
    PORT[int_enable]:=PORT[int_enable] AND NOT (irq3_bit OR irq4_bit);
    PORT[int_control]:=eoi_bits;
    {enable serial controller to generate receive interrupts}
    PORT[com1_line_control]:=PORT[com1_line_control] AND NOT dlab;
    PORT[com1_int_enable]:=PORT[com1_int_enable] OR receive_int;
    PORT[com1_modem_control]:=PORT[com1_modem_control] OR out2;
    PORT[com2_line_control]:=PORT[com2_line_control] AND NOT dlab;
    PORT[com2_int_enable]:=PORT[com2_int_enable] OR receive_int;
    PORT[com2_modem_control]:=PORT[com2_modem_control] OR out2;
    {get initial interrupt identification, un-hangs the ports if hung}
    DELAY(100); com1_id:=PORT[com1_int_id]; com2_id:=PORT[com2_int_id]
  END
  ELSE BEGIN
    {tell controller to ignore irq3 and irq4 interrupts}
    PORT[int_enable]:=PORT[int_enable] OR irq3_bit OR irq4_bit;
    {disable serial controller interrupts}
    PORT[com1_line_control]:=PORT[com1_line_control] AND NOT dlab;
    PORT[com1_int_enable]:=PORT[com1_int_enable] AND NOT receive_int;
    PORT[com1_modem_control]:=PORT[com1_modem_control] AND NOT out2;
    PORT[com2_line_control]:=PORT[com2_line_control] AND NOT dlab;
```

```
    PORT[com2_int_enable]:=PORT[com2_int_enable] AND NOT receive_int;
    PORT[com2_modem_control]:=PORT[com2_modem_control] AND NOT out2;
    {restore original interrupt server addresses}
    MEMW[0:irq4_code*4+2]:=com1_seg; MEMW[0:irq4_code*4]:=com1_ofs;
    MEMW[0:irq3_code*4+2]:=com2_seg; MEMW[0:irq3_code*4]:=com2_ofs
  END
END;


OVERLAY PROCEDURE frontend;
{main procedure -- overlay with table constants}

VAR {backend variables -- accessed only by backend}
{these variables have to be here to remain static between calls to backend}

{mi_tab constant table}
{fld_tab constant table}
{reg_tab constant table}

{files for micro/macro loader, dumper and binary builder}
micro_file:FILE OF bin_load_typ; macro_file:TEXT;

{name of files for micro/macro loader, dumper and binary builder}
dump_name,load_name,build_name:str_typ;

{state information for micro/macro loader}
load_code:BYTE; load_errors:TEXT; load_error:BOOLEAN;

{obus bytes from last read for extraction of e-obus data}
obus_bytes:bits_typ;

{micro/macro history base values -- used to calculate offsets}
hist_base:RECORD micro,macro:bits_typ END;

{block of data to be saved and restored}
saved:RECORD count:BYTE; mic:INTEGER; mi:bits_typ; last_cond:BOOLEAN END;

{system state extracted from last status loop read}
state:RECORD running,last_cond,this_cond,branching:BOOLEAN END;

{delay values to be loaded into flag bytes}
delays:RECORD
  row_enb_dly,cas_dly,wrt_dly,ea_mpx_dly_ctrl,sts_clk_dly,
  mid_cy_clk_dly_ctrl,main_clk_dly_ctrl,mc_mdm_go_dly,mc_mm_enb_dly,
  sm_cylen,sm_disp_cylen,fm_cylen,fm_disp_cylen,mc_mem_go_post_dly:BYTE
END;

{local copies of flag bytes -- since these can not be read like iora/b}
local_flags:RECORD
  mma_ctrl,init0,init1,init2,init3,init4,init5,init6,init7:BYTE;
END;

{state of backend -- if not true then init must be done}
init_ok:BOOLEAN;

{talk buffers for remote/local communications}
local_talk,remote_talk:talk_typ;
```

```
FUNCTION backend(arg1s:str_typ):str_typ;
{backend functions interface to the actual system -- no display}

CONST

{data ports rd/wr}
port_0=$300; port_1=$301; port_2=$302; port_3=$303;
port_4=$304; port_5=$305; port_6=$306; port_7=$307;
rem_sts_port=$308; {rd} rem_err_port=$308; {wr}
iora_port=$309; {rd/wr} iorb_port=$30A; {rd/wr}
dma_strb_port=$30B; {rd/wr}
rd_shifts_port=$30C; {rd}
ld_shft_cnt_port=$30D; {wr}
diag_loop_port=$30E; {wr}
rem_res_port=$30F; {wr} loc_res_port=$318; {wr}
loc_err_port=$319; {wr} loc_sts_port=$31E; {rd}
diag_port=$31F; {rd/wr}

{local status bits}
rem_par_err=2; loc_par_err=1;

{iora bits}
cc_act_strb=128; cc_ctrl_shift_enb=64; n_cc_enb_fbus_ctrl=32;
cc_dat_loop_ctrl_2=16; cc_dat_loop_ctrl_1=8; n_cc_sel_mma_ctrl=4;
cc_sts_a_ld_enb=2; n_cc_sts_a_ld_enb=1;

{iorb bits}
n_cc_stb_mma_ctrl=128; cc_res=64; n_cc_res=32; n_cc_dat_loop_oe=16;
n_cc_enb_mi_b=8; cc_mm_wrt_enb=4; cc_cpu_run_sw=2; cc_mc_ctrl_shift_enb=1;

{mma_ctrl bits}
low_mem=64; cc_mm_sctn_2=32; cc_mm_sctn_1=16; cc_wrt_mm=8; cc_mm_wrt_go=4;
cc_mm_res=2; cc_force_mm_b_sel=1;

{init0 bits}
hs_brk_stop_sw=64; muhs_brk_stop_sw=32;
mi_par_stop_sw=16; {???} hardecc_stop_sw=8; {???} cpu_cont_sw=2;

{init2 bits}
cc_enb_mi_a=16; cc_dis_mi_ld=8; cc_enb_mi_c=4; cc_sq_disable=2;

VAR {error values}
timed_out:BOOLEAN; {set true if shift timed out}
bad_reg, {name of bad register or ''}
bad_fld:str_typ; {name of bad field or ''}

{$I CONSOLE1.PAS the backend functions}
{CONSOLE1.PAS -- source file for Augment Engine Console (3 of 5) zw}

{contains backend functions}

PROCEDURE shift(n:BYTE);
{shift data loop n times}
VAR busy:BYTE;
BEGIN
  {start the load shifter}
```

```
  PORT[ld_shft_cnt_port]:=n; busy:=0;
  {wait until shift completed or timed out}
  WHILE NOT timed_out AND NOT ODD(PORT[rem_sts_port]) DO BEGIN
    busy:=busy+1; timed_out:=(busy>=100)
  END;
END;

PROCEDURE iora(on,off:BYTE);
{set iora bits on/off}
BEGIN
  PORT[iora_port]:=(PORT[iora_port] OR on) AND NOT off;
  {may be replaced replace by the following inline code
  MOV DX,309 ! IN AL,DX ! OR <on> ! AND <not off> ! OUT DX,AL
  INLINE($BA/$09/$03/$EC/$0C/<on>/$24/<not off>/$EE)}
END;

PROCEDURE iorb(on,off:BYTE);
{set iorb bits on/off}
BEGIN
  PORT[iorb_port]:=(PORT[iorb_port] OR on) AND NOT off;
  {may be replaced replace by the following inline code
  MOV DX,30A ! IN AL,DX ! OR <on> ! AND <not off> ! OUT DX,AL
  INLINE($BA/$0A/$03/$EC/$0C/<on>/$24/<not off>/$EE)}
END;

PROCEDURE mmactrl(on,off:BYTE);
{set mmactrl bits on/off}
BEGIN
  WITH local_flags DO BEGIN
    {adjust the local copy}
    mma_ctrl:=(mma_ctrl OR on) AND NOT off;
    {write out the local value -- first open the gate}
    iora(0,n_cc_sel_mma_ctrl);
    {INLINE($BA/$09/$03/$EC/$24/$F6/$EE);}
    {output the data, shift it in}
    PORT[port_7]:=mma_ctrl; shift(7);
    {close the gate}
    iora(n_cc_sel_mma_ctrl,0);
    {INLINE($BA/$09/$03/$EC/$0C/$04/$EE);}
    {an strobe makes it official}
    iorb(0,n_cc_stb_mma_ctrl); iorb(n_cc_stb_mma_ctrl,0)
    {INLINE($BA/$0A/$03/$EC/$24/$7F/$EE/$0C/$80/$EE)}
  END
END;

PROCEDURE inits(all:BOOLEAN);
{set inits from local flags, if all then send all else just 0,1,2}
BEGIN
  {open the gate}
  iora(cc_ctrl_shift_enb,0);
  {INLINE($BA/$09/$03/$EC/$0C/$40/$EE);}
  {certain operations, like init2, do not require all inits to be written}
  WITH local_flags DO IF NOT all THEN BEGIN
    {make sure this flag is off}
    iorb(0,cc_mc_ctrl_shift_enb);
    {INLINE($BA/$0A/$03/$EC/$24/$FE/$EE);}
    {output the data and shift it in}
```

```
LISTING: CONSOLE.TAB  01 JAN 80  08:30:00  (CONSOLE.TAB)

    PORT[port_5]:=init0; PORT[port_6]:=init1; PORT[port_7]:=init2; shift(24);
    {close the gate}
    iora(0,cc_ctrl_shift_enb)
    {INLINE($BA/$09/$03/$EC/$24/$6F/$EE)}
  END
  ELSE BEGIN
    {flag on so all inits will be written}
    iorb(cc_mc_ctrl_shift_enb,0);
    {output data and shift it in}
    PORT[port_0]:=init0; PORT[port_1]:=init1;
    PORT[port_2]:=init2; PORT[port_3]:=init3;
    PORT[port_4]:=init4; PORT[port_5]:=init5;
    PORT[port_6]:=init6; PORT[port_7]:=init7;
    shift(64);
    {close the gate}
    iora(0,cc_ctrl_shift_enb);
    {clean up}
    iorb(0,cc_mc_ctrl_shift_enb)
  END;
  {strobe it and its official}
  iora(cc_act_strb,0); iora(0,cc_act_strb)
  {INLINE($BA/$09/$03/$EC/$0C/$80/$EE/$24/$7F/$EE)}
END;

PROCEDURE init2(on,off:BYTE);
{set init2 bits on/off}
BEGIN
  WITH local_flags DO init2:=(init2 OR on) AND NOT off; inits(FALSE)
END;

PROCEDURE cpuob(yes:BOOLEAN);
{enable/disable cpu control of obus}
BEGIN
  IF yes THEN init2(0,cc_sq_disable) ELSE init2(cc_sq_disable,0)
END;

PROCEDURE run(yes:BOOLEAN);
{set/clear system run flag}
BEGIN
  IF yes THEN iorb(cc_cpu_run_sw,0) ELSE iorb(0,cc_cpu_run_sw);
  iora(cc_act_strb,0); iora(0,cc_act_strb)
END;

PROCEDURE step(load_mi:BOOLEAN; count:BYTE);
{step (and load mi) micro engine specified number of times}
VAR i:BYTE;
BEGIN
  FOR i:=1 TO count DO BEGIN
    WITH local_flags DO init0:=init0 AND NOT cpu_cont_sw;
    IF NOT load_mi THEN init2(cc_dis_mi_ld,0) ELSE inits(FALSE);
    WITH local_flags DO init0:=init0 OR cpu_cont_sw; inits(FALSE);
    IF NOT load_mi THEN init2(0,cc_dis_mi_ld)
  END
END;

PROCEDURE wrob(VAR obus:bits_typ; disable_cpu:BOOLEAN);
{write data to obus, might have to disable cpu access}
```

```
VAR bytes:bits_typ;
BEGIN
  fit_bits(obus,5,8,left_fit); bytes.siz:=9; bytes.bit_siz:=8;
  bytes.bit[0]:=obus.bit[0];
  bytes.bit[1]:=obus.bit[1] AND $F0;
  bytes.bit[2]:=0;
  bytes.bit[3]:=((obus.bit[1] AND $F) SHL 4) OR (obus.bit[2] SHR 4);
  bytes.bit[4]:=(obus.bit[2] AND $F) SHL 4;
  bytes.bit[5]:=0;
  bytes.bit[6]:=obus.bit[3];
  bytes.bit[7]:=obus.bit[4];
  bytes.bit[8]:=0;
  IF disable_cpu THEN cpuob(FALSE);
  iora(cc_dat_loop_ctrl_2,cc_dat_loop_ctrl_1);
  {INLINE($BA/$09/$03/$EC/$0C/$10/$24/$F7/$EE);}
  PORT[port_7]:=$FF; shift(8);
  PORT[port_7]:=bytes.bit[7] OR $0F;
  PORT[port_6]:=bytes.bit[6];
  PORT[port_5]:=$FF;
  PORT[port_4]:=bytes.bit[4] OR $0F;
  PORT[port_3]:=bytes.bit[3];
  PORT[port_2]:=$FF;
  PORT[port_1]:=bytes.bit[1] OR $0F;
  PORT[port_0]:=bytes.bit[0];
  shift(64);
  iora(0,cc_dat_loop_ctrl_2);
  {INLINE($BA/$09/$03/$EC/$24/$EF/$EE);}
  IF disable_cpu THEN cpuob(TRUE)
END;

PROCEDURE rdob(VAR obus:bits_typ; from_alu,eobus:BOOLEAN);
{read obus data, maybe from alu, maybe e-obus data from previous read}
BEGIN
  obus.siz:=5; obus.bit_siz:=8;
  IF NOT eobus THEN BEGIN
    obus_bytes.siz:=9; obus_bytes.bit_siz:=8;
    IF from_alu THEN BEGIN
      iora(cc_dat_loop_ctrl_1 OR cc_dat_loop_ctrl_2,0);
      {INLINE($BA/$09/$03/$EC/$0C/$18/$EE);}
      shift(1)
    END;
    iora(cc_dat_loop_ctrl_2,cc_dat_loop_ctrl_1);
    {INLINE($BA/$09/$03/$EC/$0C/$10/$24/$F7/$EE);}
    shift(8); obus_bytes.bit[8]:=PORT[port_0];
    shift(64);
    obus_bytes.bit[7]:=PORT[port_7]; obus_bytes.bit[6]:=PORT[port_6];
    obus_bytes.bit[5]:=PORT[port_5]; obus_bytes.bit[4]:=PORT[port_4];
    obus_bytes.bit[3]:=PORT[port_3]; obus_bytes.bit[2]:=PORT[port_2];
    obus_bytes.bit[1]:=PORT[port_1]; obus_bytes.bit[0]:=PORT[port_0];
    iora(0,cc_dat_loop_ctrl_2);
    {INLINE($BA/$09/$03/$EC/$24/$EF/$EE);}
    obus.bit[0]:=obus_bytes.bit[0];
    obus.bit[1]:=(obus_bytes.bit[1] AND $F0) OR (obus_bytes.bit[3] SHR 4);
    obus.bit[2]:=((obus_bytes.bit[3] AND $F) SHL 4) OR
      (obus_bytes.bit[4] SHR 4);
    obus.bit[3]:=obus_bytes.bit[6];
    obus.bit[4]:=obus_bytes.bit[7] AND $F0;
```

```
  END
  ELSE BEGIN
    obus.bit[0]:=(obus_bytes.bit[1] SHL 4) OR (obus_bytes.bit[2] SHR 4);
    obus.bit[1]:=(obus_bytes.bit[2] SHL 4) OR (obus_bytes.bit[4] AND $0F);
    obus.bit[2]:=obus_bytes.bit[5];
    obus.bit[3]:=(obus_bytes.bit[7] SHL 4) OR (obus_bytes.bit[8] SHR 4);
    obus.bit[4]:=obus_bytes.bit[8] SHL 4;
  END;
  fit_bits(obus,36,1,left_fit)
END;

PROCEDURE wrdma(VAR addr,dat:bits_typ);
{dma write main memory -- duplicated in fast load}
BEGIN
  fit_bits(dat,5,8,left_fit);
  PORT[port_3]:=dat.bit[0]; PORT[port_4]:=dat.bit[1]; PORT[port_5]:=dat.bit[2
  PORT[port_6]:=dat.bit[3]; PORT[port_7]:=dat.bit[4]; PORT[port_2]:=0;
  shift(4);
  fit_bits(addr,3,8,left_fit);
  PORT[port_0]:=addr.bit[0] AND $7F;
  PORT[port_1]:=addr.bit[1]; PORT[port_2]:=addr.bit[2];
  PORT[dma_strb_port]:=1
END;

PROCEDURE rddma(VAR addr,dat:bits_typ);
{dma read main memory}
VAR x:BYTE;
BEGIN
  fit_bits(addr,3,8,left_fit);
  PORT[port_0]:=addr.bit[0] AND $7F;
  PORT[port_1]:=addr.bit[1]; PORT[port_2]:=addr.bit[2];
  dat.siz:=5; dat.bit_siz:=8; x:=PORT[dma_strb_port];
  dat.bit[0]:=((PORT[port_3] AND $0F) SHL 4) OR (PORT[port_4] SHR 4);
  dat.bit[1]:=((PORT[port_4] AND $0F) SHL 4) OR (PORT[port_5] SHR 4);
  dat.bit[2]:=((PORT[port_5] AND $0F) SHL 4) OR (PORT[port_6] SHR 4);
  dat.bit[3]:=((PORT[port_6] AND $0F) SHL 4) OR (PORT[port_7] SHR 4);
  dat.bit[4]:=(PORT[port_7] AND $0F) SHL 4
END;

PROCEDURE rdstlp(VAR dat:bits_typ);
{read status loop data, 128 bits -- set state flags}
BEGIN
  dat.siz:=16; dat.bit_siz:=8;
  iora(cc_sts_a_ld_enb,n_cc_sts_a_ld_enb);
  {INLINE($BA/$09/$03/$EC/$0C/$02/$24/$FE/$EE);}
  shift(1);
  iora(n_cc_sts_a_ld_enb,cc_sts_a_ld_enb);
  {INLINE($BA/$09/$03/$EC/$0C/$01/$24/$FD/$EE);}
  shift(64);
  dat.bit[8]:=PORT[port_0];
  dat.bit[9]:=PORT[port_1];
  dat.bit[10]:=PORT[port_2];
  dat.bit[11]:=PORT[port_3];
  dat.bit[12]:=PORT[port_4] AND $FC;
  dat.bit[13]:=PORT[port_5] AND $FC;
  dat.bit[14]:=PORT[port_6] AND $0;
  dat.bit[15]:=PORT[port_7];
```

```
  shift(64);
  dat.bit[0]:=PORT[port_0];
  dat.bit[1]:=PORT[port_1];
  dat.bit[2]:=PORT[port_2];
  dat.bit[3]:=PORT[port_3];
  dat.bit[4]:=PORT[port_4] AND $1F;
  dat.bit[5]:=PORT[port_5];
  dat.bit[6]:=PORT[port_6];
  dat.bit[7]:=PORT[port_7];
  fit_bits(dat,128,1,left_fit);
  WITH state DO BEGIN
    running:=(dat.bit[89]=1); this_cond:=(dat.bit[71]=0);
    last_cond:=(dat.bit[88]=1); branching:=(dat.bit[86]=0)
  END
END;

PROCEDURE rdmi(VAR mi:bits_typ);
{read micro instruction register -- 88 bits}
VAR bytes,bits:bits_typ; i,j,k:BYTE;

PROCEDURE copy_bits(count,get_pos,put_pos:BYTE);
{copy bits into returned value}
VAR i:BYTE;
BEGIN
  count:=count-1;
  FOR i:=0 TO count DO mi.bit[put_pos+i]:=bits.bit[get_pos+i]
END;

BEGIN {rdmi}
  rdstlp(bytes);
  mi.siz:=88; mi.bit_siz:=1; fit_bits(bytes,128,1,left_fit); bits:=bytes;
  FOR i:=0 TO 4 DO BEGIN
    j:=i*8; FOR k:=0 TO 7 DO bits.bit[j+k]:=bytes.bit[j+7-k]
  END;
  copy_bits(26,0,0); copy_bits(4,124,26); copy_bits(6,26,30);
  copy_bits(14,42,36); copy_bits(6,104,50); copy_bits(6,96,56);
  copy_bits(6,56,62); copy_bits(1,32,68); copy_bits(4,33,69);
  copy_bits(9,62,73); copy_bits(4,120,82); copy_bits(2,40,86)
END;

PROCEDURE wrmi(VAR mi:bits_typ; sect:BYTE);
{write micro instruction register -- only specified sections}
VAR obus:bits_typ;
BEGIN
  cpuob(FALSE);
  fit_bits(mi,11,8,left_fit); obus.siz:=5; obus.bit_siz:=8; obus.bit[4]:=0;
  mmactrl(cc_mm_wrt_go OR cc_wrt_mm OR cc_mm_res OR cc_force_mm_b_sel,
    cc_mm_sctn_1 OR cc_mm_sctn_2);
  IF (sect AND sect_A)<>0 THEN BEGIN
    mmactrl(0,cc_mm_res);
    obus.bit[0]:=mi.bit[0]; obus.bit[1]:=mi.bit[1];
    obus.bit[2]:=mi.bit[2]; obus.bit[3]:=mi.bit[3]; wrob(obus,FALSE);
    mmactrl(cc_mm_sctn_1,0);
    iorb(0,n_cc_dat_loop_oe);
    {INLINE($BA/$0A/$03/$EC/$24/$EF/$EE);}
    init2(cc_enb_mi_a,cc_dis_mi_ld OR cc_enb_mi_c); init2(0,cc_enb_mi_a)
  END;
```

```
  IF (sect AND sect_B)<>0 THEN BEGIN
    obus.bit[0]:=mi.bit[4]; obus.bit[1]:=mi.bit[5];
    obus.bit[2]:=mi.bit[6]; obus.bit[3]:=mi.bit[7]; wrob(obus,FALSE);
    mmactrl(cc_mm_sctn_2,cc_mm_sctn_1);
    iorb(0,n_cc_dat_loop_oe); iorb(0,n_cc_enb_mi_b); iorb(n_cc_enb_mi_b,0)
    {INLINE($BA/$0A/$03/$EC/$24/$EF/$EE/$24/$F7/$EE/$0C/$08/$EE)}
  END;
  IF (sect AND sect_C)<>0 THEN BEGIN
    obus.bit[0]:=mi.bit[8]; obus.bit[1]:=mi.bit[9];
    obus.bit[2]:=mi.bit[10] XOR $03; obus.bit[3]:=0; wrob(obus,FALSE);
    mmactrl(cc_mm_sctn_1 OR cc_mm_sctn_2,0);
    iorb(0,n_cc_dat_loop_oe);
    {INLINE($BA/$0A/$03/$EC/$24/$EF/$EE);}
    init2(cc_enb_mi_c,0); init2(0,cc_enb_mi_c)
  END;
  init2(0,cc_dis_mi_ld);
  mmactrl(0,cc_force_mm_b_sel OR cc_mm_sctn_1 OR cc_mm_sctn_2 OR
    cc_wrt_mm OR cc_mm_wrt_go);
  iorb(n_cc_dat_loop_oe,0);
  {INLINE($BA/$0A/$03/$EC/$0C/$10/$EE);}
  cpuob(TRUE)
END;

PROCEDURE exec(VAR mi:bits_typ; clock_it:BOOLEAN);
{execute a micro instruction, maybe clock it}
BEGIN wrmi(mi,sect_ABC); IF clock_it THEN step(FALSE,1) END;

PROCEDURE mi(digits:str_typ; VAR mi:bits_typ);
{convert octal digit string to micro instruction value}
VAR i,j:mi_typ; ok:BOOLEAN;
BEGIN
  ok:=FALSE;
  FOR i:=dam_mi TO zero_mi DO IF mi_tab[i].name=digits THEN BEGIN
    ok:=TRUE; j:=i
  END;
  IF ok THEN mi:=mi_tab[j].bits ELSE oct_to_bits(digits,mi,88)
END;

FUNCTION fld(f:str_typ):fld_typ;
{convert micro instruction field name to field code}
VAR i,j:fld_typ;
BEGIN
  j:=fld_err;
  FOR i:=fld_err TO lit_fld DO IF fld_tab[i].name=f THEN j:=i;
  IF j=fld_err THEN bad_fld:=f;
  fld:=j
END;

PROCEDURE setfld(f:fld_typ; VAR mi,b:bits_typ);
{set field value in a micro instruction}
VAR x:BYTE;
BEGIN
  fit_bits(mi,88,1,left_fit); fit_bits(b,36,1,right_fit);
  IF f=adst_fld THEN IF k1 THEN BEGIN
    x:=b.bit[35]; b.bit[35]:=b.bit[33]; b.bit[33]:=x
  END;
  IF f<>fld_err THEN WITH fld_tab[f] DO put_bits(mi,b,bit_zero,bits_siz)
```

```
Listing: CONSOLE.PAS  01-JAN-80  00:00:00  (CONSOLE1.PAS)

END;

FUNCTION rdmic:INTEGER;
{read micro instruction counter}
VAR bytes,mic:bits_typ; i:BYTE;
BEGIN
  rdstlp(bytes);
  fit_bits(bytes,128,1,left_fit); mic.siz:=14; mic.bit_siz:=1;
  FOR i:=0 TO 13 DO mic.bit[i]:=bytes.bit[i+72];
  fit_bits(mic,1,16,right_fit); rdmic:=(mic.bit[0]-1) AND $3FFF
END;

PROCEDURE jump(mic:INTEGER; VAR mi:bits_typ);
{build a jump micro instruction to specified address}
BEGIN
  mi.siz:=11; mi.bit_siz:=8; mi.bit[0]:=132;
  IF k1 THEN BEGIN mi.bit[1]:=119; mi.bit[2]:=15 END
  ELSE BEGIN mi.bit[1]:=118; mi.bit[2]:=79 END;
  IF state.last_cond THEN mi.bit[3]:=0 ELSE mi.bit[3]:=64;
  mi.bit[4]:=(mic SHR 10) AND $0F;
  mi.bit[5]:=(mic SHR 2) AND $FF;
  mi.bit[6]:=(mic AND $3) SHL 6;
  mi.bit[7]:=227;
  mi.bit[8]:=144;
  mi.bit[9]:=86;
  mi.bit[10]:=16
END;

PROCEDURE wrmic(mic:INTEGER; first_time,load_mi:BOOLEAN);
{write micro instruction counter, maybe load instruction from there}
VAR mi:bits_typ;
BEGIN
  jump(mic,mi);
  IF first_time THEN wrmi(mi,sect_ABC) ELSE wrmi(mi,sect_B);
  step(load_mi,1)
END;

PROCEDURE do_save(yes,forced:BOOLEAN);
{save/restore varous items}
BEGIN
  WITH saved DO BEGIN
    IF yes THEN BEGIN
      IF forced OR (count=0) THEN BEGIN
        mic:=rdmic; rdmi(mi); last_cond:=state.last_cond
      END;
      count:=count+1
    END
    ELSE BEGIN
      IF count>0 THEN count:=count-1;
      IF forced OR (count=0) THEN BEGIN
        state.last_cond:=last_cond; wrmic(mic,TRUE,FALSE); wrmi(mi,sect_ABC)
      END
    END
  END
END;

{wrappers for do_save}
```

```
LISTING: CONSOLE.PAS  01-JAN-80  00:00:00  CONSOLE.PAS

PROCEDURE save; BEGIN do_save(TRUE,FALSE) END;
PROCEDURE restore; BEGIN do_save(FALSE,FALSE) END;

PROCEDURE wrum(addr:INTEGER; dat:bits_typ; sect:BYTE;
  VAR first_time:BOOLEAN);
{write micro memory}
VAR obus:bits_typ;

PROCEDURE strb(bits_on,bits_off:BYTE);
{action strobe bits on/off}
BEGIN
  wrob(obus,FALSE);
  iorb(0,n_cc_dat_loop_oe);
  {INLINE($BA/$0A/$03/$EC/$24/$EF/$EE);}
  IF addr < 2048 THEN mmactrl(cc_wrt_mm OR cc_mm_wrt_go OR low_mem OR
    bits_on,bits_off)
  ELSE IF addr >= 4096 THEN mmactrl(cc_wrt_mm OR cc_mm_wrt_go OR
    cc_force_mm_b_sel OR bits_on,bits_off)
  ELSE mmactrl(cc_wrt_mm OR cc_mm_wrt_go OR bits_on,cc_force_mm_b_sel OR
    bits_off);
  iorb(cc_mm_wrt_enb,0); iorb(0,cc_mm_wrt_enb);
  {INLINE($BA/$0A/$03/$EC/$0C/$04/$EE/$24/$F6/$EE);}
  mmactrl(0,cc_mm_sctn_2 OR cc_mm_sctn_1 OR cc_wrt_mm OR cc_mm_wrt_go);
  iorb(n_cc_dat_loop_oe,0)
  {INLINE($BA/$0A/$03/$EC/$0C/$10/$EE)}
END;

BEGIN {wrum}
  save; wrmic(addr,first_time,FALSE); cpuob(FALSE);
  fit_bits(dat,11,8,left_fit); obus.siz:=5; obus.bit_siz:=8; obus.bit[4]:=0;
  IF (sect AND sect_A)<>0 THEN BEGIN
    obus.bit[0]:=dat.bit[0]; obus.bit[1]:=dat.bit[1]; obus.bit[2]:=dat.bit[2]
    obus.bit[3]:=dat.bit[3]; strb(cc_mm_sctn_1,cc_mm_sctn_2)
  END;
  IF (sect AND sect_B)<>0 THEN BEGIN
    obus.bit[0]:=dat.bit[4]; obus.bit[1]:=dat.bit[5]; obus.bit[2]:=dat.bit[6]
    obus.bit[3]:=dat.bit[7]; strb(cc_mm_sctn_2,cc_mm_sctn_1)
  END;
  IF (sect AND sect_C)<>0 THEN BEGIN
    obus.bit[0]:=dat.bit[8]; obus.bit[1]:=dat.bit[9];
    obus.bit[2]:=dat.bit[10] XOR $03; obus.bit[3]:=0;
    strb(cc_mm_sctn_1 OR cc_mm_sctn_2,0)
  END;
  cpuob(TRUE); first_time:=FALSE; restore
END;

PROCEDURE rdum(addr:INTEGER; VAR dat:bits_typ);
{read micro memory}
BEGIN save; wrmic(addr,TRUE,TRUE); rdmi(dat); restore END;

FUNCTION reg(r:str_typ):reg_typ;
{convert register name to register code}
VAR i,j:reg_typ;
BEGIN
  j:=reg_err; FOR i:=reg_err TO am_reg DO IF reg_tab[i].name=r THEN j:=i;
  IF j=reg_err THEN bad_reg:=r;
  reg:=j
```

```
END;

PROCEDURE rdreg(reg:reg_typ; VAR dat:bits_typ);
{read machine register}
VAR i:BYTE;
BEGIN
  save;
  exec(mi_tab[reg_tab[reg].mi_to_rd].bits,FALSE); rdob(dat,TRUE,FALSE);
  restore
END;

PROCEDURE wrreg(reg:reg_typ; VAR dat:bits_typ);
{write machine register}
VAR mi:bits_typ;
BEGIN
  save;
  mi:=mi_tab[reg_tab[reg].mi_to_wr].bits;
  setfld(lit_fld,mi,dat); exec(mi,TRUE);
  restore
END;

PROCEDURE wrhsti(index:bits_typ; relative,micro:BOOLEAN);
{write history index value}
VAR mi:bits_typ;
BEGIN
  save;
  IF micro THEN BEGIN
    mi:=mi_tab[luhadr_mi].bits;
    IF relative THEN WITH hist_base DO BEGIN
      fit_bits(index,1,16,right_fit); fit_bits(micro,1,16,right_fit);
      index.bit[0]:=(micro.bit[0]-index.bit[0]) AND $03FF
    END;
    fit_bits(index,10,1,right_fit); put_bits(mi,index,34,10);
    exec(mi,TRUE)
  END
  ELSE BEGIN
    mi:=mi_tab[lmhadr_mi].bits;
    IF relative THEN WITH hist_base DO BEGIN
      fit_bits(index,1,16,right_fit); fit_bits(macro,1,16,right_fit);
      index.bit[0]:=(macro.bit[0]-index.bit[0]) AND $03FF
    END;
    fit_bits(index,10,1,right_fit); put_bits(mi,index,52,10);
    exec(mi,TRUE)
  END;
  restore
END;

PROCEDURE rdhsti(VAR index:bits_typ; relative,micro:BOOLEAN);
{read history index value}
VAR mi:bits_typ;
BEGIN
  IF relative THEN BEGIN
    {"relative" means return the current base value}
    IF micro THEN index:=hist_base.micro ELSE index:=hist_base.macro
  END
  ELSE BEGIN
    save;
```

```
    {read 10 bit value, right justified and masked}
    IF micro THEN exec(mi_tab[xuhadr_mi].bits,FALSE)
    ELSE exec(mi_tab[xmhadr_mi].bits,FALSE);
    rdob(index,TRUE,FALSE);
    restore
  END;
  {make sure we have 10 bits! -- probably not needed???}
  fit_bits(index,10,1,right_fit)
END;

PROCEDURE rdhst(VAR dat:bits_typ; micro:BOOLEAN);
{read micro/macro history data}
VAR mi,b0,b1:bits_typ;
BEGIN
  save;
  {note that clock takes place after reading data from obus}
  IF micro THEN BEGIN
    {read 36 bit micro history data}
    {note that this instruction, when clocked, clocks the micro index}
    exec(mi_tab[xuhdat_mi].bits,FALSE); rdob(dat,TRUE,FALSE)
  END
  ELSE BEGIN
    {read 40 bit macro history data}
    {first read 8 bits 32 to 39, right justified and masked}
    exec(mi_tab[xmhdat1_mi].bits,FALSE); rdob(b1,TRUE,FALSE);
    fit_bits(b1,8,1,right_fit);
    {next read 32 bits 0 to 31, right justified and masked}
    {note that this instruction, when clocked, clocks the macro index}
    exec(mi_tab[xmhdat0_mi].bits,FALSE); rdob(b0,TRUE,FALSE);
    fit_bits(b0,32,1,right_fit);
    {assemble the 8 and 32 bit chunks into one 40 bit value to be returne
    dat.siz:=40; dat.bit_siz:=1; put_bits(dat,b0,0,32); put_bits(dat,b1,32,8)
  END;
  {this clock increments the history index counter}
  step(FALSE,1);
  restore
END;

PROCEDURE enahst(yes:BOOLEAN);
{enable/disable micro/macro histories}
BEGIN
  save;
  {note that a "hole" is inserted in the history}
  {this is because it takes two instructions to do the job}
  WITH hist_base DO IF yes THEN BEGIN
    {write micro history index, forward to next empty history location}
    fit_bits(micro,1,16,right_fit); micro.bit[0]:=(micro.bit[0]+1) AND $03FF;
    wrhsti(micro,FALSE,TRUE);
    {write macro history index, forward to next empty history location}
    {note extra forward to correct for extra micro clock}
    fit_bits(macro,1,16,right_fit); macro.bit[0]:=(macro.bit[0]+2) AND $03FF;
    wrhsti(macro,FALSE,FALSE);
    {enable micro history recording}
    exec(mi_tab[uhon_mi].bits,TRUE);
    {enable macro history recording -- note that micro is clocked here}
    exec(mi_tab[mhon_mi].bits,TRUE)
  END
```

```
  ELSE BEGIN
    {disable micro history recording -- note that macro is clocked here}
    exec(mi_tab[uhoff_mi].bits,TRUE); exec(mi_tab[mhoff_mi].bits,TRUE);
    {read micro history index, back up to last valid history location}
    rdhsti(micro,FALSE,TRUE);
    fit_bits(micro,1,16,right_fit); micro.bit[0]:=(micro.bit[0]-1) AND $03FF;
    {read macro history index, back up to last valid history location}
    {note the extra backup to correct for the extra macro clock}
    rdhsti(macro,FALSE,FALSE);
    fit_bits(macro,1,16,right_fit); macro.bit[0]:=(macro.bit[0]-2) AND $03FF
  END;
  restore
END;

PROCEDURE rdm(VAR addr,dat:bits_typ);
{read main memory via micro instruction}
VAR mi,saved_ma,saved_mem:bits_typ;
BEGIN
  save; rdreg(ma_reg,saved_ma); rdreg(mem_reg,saved_mem);
  fit_bits(addr,36,1,right_fit); wrreg(ma_reg,addr);
  mi:=mi_tab[mrd_mi].bits; exec(mi,TRUE); rdreg(mem_reg,dat);
  wrreg(mem_reg,saved_mem); wrreg(ma_reg,saved_ma); restore
END;

PROCEDURE wrm(VAR addr,dat:bits_typ);
{write main memory via micro instruction}
VAR mi,saved_ma,saved_mem:bits_typ;
BEGIN
  save; rdreg(ma_reg,saved_ma); rdreg(mem_reg,saved_mem);
  fit_bits(addr,36,1,right_fit); wrreg(ma_reg,addr);
  mi:=mi_tab[memst_mi].bits; setfld(lit_fld,mi,dat); exec(mi,TRUE);
  mi:=mi_tab[jump_mi].bits; exec(mi,TRUE);
  wrreg(mem_reg,saved_mem); wrreg(ma_reg,saved_ma); restore
END;

FUNCTION rdecc:BYTE;
{read main memory ecc code}
VAR obus,ecc:bits_typ; i:BYTE;
BEGIN
  save;
  exec(mi_tab[xmbst_mi].bits,FALSE); rdob(obus,TRUE,FALSE);
  ecc.siz:=8; ecc.bit_siz:=1;
  FOR i:=28 TO 35 DO ecc.bit[i-28]:=obus.bit[i];
  fit_bits(ecc,1,8,left_fit); rdecc:=ecc.bit[0] XOR $3F;
  restore
END;

PROCEDURE wram(VAR addr,dat:bits_typ; indexed:BOOLEAN);
{write amem, maybe indexed}
VAR mi,saved_ar:bits_typ;
BEGIN
  save;
  IF indexed THEN BEGIN
    mi:=mi_tab[memst_mi].bits;
    fit_bits(addr,1,16,right_fit);
    addr.bit[0]:=(addr.bit[0] AND $0F)+$20;
    setfld(dest_fld,mi,addr); setfld(lit_fld,mi,dat);
```

```
    exec(mi,TRUE);
  END
  ELSE BEGIN
    rdreg(ar_reg,saved_ar);
    wrreg(ar_reg,dat);
    mi:=mi_tab[ldamar_mi].bits; setfld(jadr_fld,mi,addr);
    exec(mi,TRUE);
    wrreg(ar_reg,saved_ar)
  END;
  restore
END;

PROCEDURE rdam(VAR addr,dat:bits_typ; indexed:BOOLEAN);
{read amem, maybe indexed}
VAR mi,sixty:bits_typ;
BEGIN
  save;
  IF indexed THEN BEGIN
    mi:=mi_tab[jump_mi].bits;
    fit_bits(addr,1,16,right_fit);
    addr.bit[0]:=(addr.bit[0] AND $0F)+$20;
    setfld(d_fld,mi,addr)
  END
  ELSE BEGIN
    mi:=mi_tab[jump_mi].bits;
    oct_to_bits('60',sixty,36); setfld(d_fld,mi,sixty);
    setfld(jadr_fld,mi,addr);
  END;
  exec(mi,FALSE); rdob(dat,TRUE,FALSE);
  restore
END;

PROCEDURE wrac(VAR ac,dat:bits_typ);
{write specified ac register}
VAR mi,seven:bits_typ;
BEGIN
  save;
  mi:=mi_tab[ldac_mi].bits;
  oct_to_bits('7',seven,36); setfld(acsel_fld,mi,seven);
  setfld(dest_fld,mi,ac); setfld(lit_fld,mi,dat);
  exec(mi,TRUE);
  restore
END;

PROCEDURE rdac(VAR ac,dat:bits_typ);
{read specified ac register}
VAR saved_ir,new_ir:bits_typ;
BEGIN
  save; rdreg(ir_reg,saved_ir);
  oct_to_bits('0',new_ir,36); put_bits(new_ir,ac,9,4);
  wrreg(ir_reg,new_ir); rdreg(ac_reg,dat);
  wrreg(ir_reg,saved_ir); restore
END;

OVERLAY FUNCTION cty(on:BOOLEAN):str_typ;
{set remote cty mode}
BEGIN
```

```
  cty:='OK';
  IF on THEN BEGIN
    IF remote=active THEN remote:=remote_cty ELSE cty:='?Remote not active.'
  END
  ELSE BEGIN
    IF remote=remote_cty THEN remote:=active ELSE cty:='?Remote not cty.'
  END
END;

OVERLAY PROCEDURE init(send:BOOLEAN);
{set up initial values -- maybe send to system}
VAR i:BYTE;
BEGIN
  dump_name:=''; load_name:=''; build_name:=''; init_ok:=send;
  FOR i:=1 TO 12 DO BEGIN local_talk[i]:=''; remote_talk[i]:='' END;
  PORT[loc_res_port]:=0; PORT[rem_res_port]:=0;
  WITH saved DO BEGIN
    count:=0; mic:=0; last_cond:=TRUE; oct_to_bits('0',mi,88)
  END;
  oct_to_bits('0',obus_bytes,36);
  WITH hist_base DO BEGIN
    oct_to_bits('0',micro,10); oct_to_bits('0',macro,10)
  END;
  WITH state DO BEGIN
    last_cond:=TRUE; this_cond:=TRUE; branching:=TRUE; running:=TRUE;
  END;
  PORT[iora_port]:=
    n_cc_enb_fbus_ctrl OR n_cc_sel_mma_ctrl OR n_cc_sts_a_ld_enb;
  PORT[iorb_port]:=
    n_cc_stb_mma_ctrl OR n_cc_res OR n_cc_dat_loop_oe OR n_cc_enb_mi_b;
  WITH local_flags DO BEGIN mma_ctrl:=0; init0:=0; init2:=init2 AND $E0 END;
  IF send THEN BEGIN
    mmactrl(0,0); inits(TRUE); iorb(cc_res,n_cc_res); iorb(n_cc_res,cc_res)
  END
END;

OVERLAY FUNCTION sysid(num:BOOLEAN):BYTE;
{return system identification -- number or type}
BEGIN
  IF num
  THEN sysid:=((PORT[rem_sts_port] AND $F0) SHL 4) OR
    (PORT[ld_shft_cnt_port] AND $FF)
  ELSE sysid:=(PORT[rem_sts_port] SHR 1) AND 1
END;

OVERLAY FUNCTION talk(remote:BOOLEAN; message:str_typ):str_typ;
{remote/local talk tranceiver}

PROCEDURE send(VAR buffer:talk_typ);
{send message to specified buffer}
VAR i,j:1..12;
BEGIN
  FOR i:=2 TO 12 DO buffer[i]:=buffer[i-1];
  buffer[1]:=message;
  j:=0; FOR i:=1 TO 12 DO IF buffer[i]<>'' THEN j:=j+1;
  talk:=int_dec(j)
END;
```

```
PROCEDURE receive(VAR buffer:talk_typ);
{receive message from specified buffer}
VAR i:1..12;
BEGIN
  i:=12; WHILE (i>1) AND (buffer[i]='') DO i:=i-1;
  talk:=buffer[i]; buffer[i]:=''
END;

BEGIN {talk}
  IF remote THEN BEGIN
    IF message<>'' THEN send(remote_talk) ELSE receive(local_talk)
  END
  ELSE BEGIN
    IF message<>'' THEN send(local_talk) ELSE receive(remote_talk)
  END
END;

OVERLAY FUNCTION parse_inits(i:str_typ):str_typ;
{parse inits (delay values), return current values}

PROCEDURE parse_them;
{convert from digit strings}
BEGIN
  WITH delays DO BEGIN
    row_enb_dly:=oct_int(parse(i)) AND $07;
    cas_dly:=oct_int(parse(i)) AND $07;
    wrt_dly:=oct_int(parse(i)) AND $07;
    ea_mpx_dly_ctrl:=oct_int(parse(i)) AND $07;
    sts_clk_dly:=oct_int(parse(i)) AND $07;
    mid_cy_clk_dly_ctrl:=oct_int(parse(i)) AND $07;
    main_clk_dly_ctrl:=oct_int(parse(i)) AND $07;
    mc_mdm_go_dly:=oct_int(parse(i)) AND $07;
    mc_mm_enb_dly:=oct_int(parse(i)) AND $07;
    sm_cylen:=oct_int(parse(i)) AND $0F;
    sm_disp_cylen:=oct_int(parse(i)) AND $07;{not used}
    fm_cylen:=oct_int(parse(i)) AND $0F;
    fm_disp_cylen:=oct_int(parse(i)) AND $0F;
    mc_mem_go_post_dly:=oct_int(parse(i)) AND $07
  END
END;

PROCEDURE unparse_them;
{convert to digit strings}
BEGIN
  WITH delays DO BEGIN
    i:=int_oct(row_enb_dly);
    i:=i+', '+int_oct(cas_dly);
    i:=i+', '+int_oct(wrt_dly);
    i:=i+', '+int_oct(ea_mpx_dly_ctrl);
    i:=i+', '+int_oct(sts_clk_dly);
    i:=i+', '+int_oct(mid_cy_clk_dly_ctrl);
    i:=i+', '+int_oct(main_clk_dly_ctrl);
    i:=i+', '+int_oct(mc_mdm_go_dly);
    i:=i+', '+int_oct(mc_mm_enb_dly);
    i:=i+', '+int_oct(sm_cylen);
    i:=i+', '+int_oct(sm_disp_cylen);
```

```
    i:=i+', '+int_oct(fm_cylen);
    i:=i+', '+int_oct(fm_disp_cylen);
    i:=i+', '+int_oct(mc_mem_go_post_dly)
  END
END;

BEGIN {parse_inits}
  IF i<>'' THEN parse_them;
  WITH delays,local_flags DO BEGIN
    init1:=(row_enb_dly SHL 5) OR (cas_dly SHL 2);
      init2:=(wrt_dly SHL 5) OR (init2 AND $1F);
    init3:=(ea_mpx_dly_ctrl SHL 5) OR (sts_clk_dly SHL 2)
      OR ((mid_cy_clk_dly_ctrl AND $06) SHR 1);
    init4:=((mid_cy_clk_dly_ctrl AND $01) SHL 7)
      OR (main_clk_dly_ctrl SHL 4) OR (mc_mdm_go_dly SHL 1) OR
      ((mc_mm_enb_dly AND $04) SHR 2);
    init5:=((mc_mm_enb_dly AND $03) SHL 6) OR (sm_cylen SHL 2)
      OR ((sm_disp_cylen AND $06) SHR 1);
    init6:=((sm_disp_cylen AND $01) SHL 7) OR (fm_cylen SHL 2)
      OR ((fm_disp_cylen AND $0C) SHR 2);
    init7:=((fm_disp_cylen AND $03) SHL 6) OR (mc_mem_go_post_dly SHL 3)
  END;
  unparse_them; parse_inits:=i
END;

OVERLAY FUNCTION bits(args:str_typ):str_typ;
{debugging access to the bit converter}
VAR b:bits_typ; siz,bit_siz,i:BYTE;
BEGIN
  IF args='' THEN bits:=''
  ELSE BEGIN
    b.siz:=dec_int(parse(args)); b.bit_siz:=dec_int(parse(args));
    FOR i:=0 TO b.siz-1 DO b.bit[i]:=dec_int(parse(args));
    debug('BITS='+debug_bits(b));
    siz:=dec_int(parse(args)); bit_siz:=dec_int(parse(args));
    IF eq(args,'LEFT',1) THEN fit_bits(b,siz,bit_siz,left_fit)
    ELSE fit_bits(b,siz,bit_siz,right_fit);
    bits:=debug_bits(b)
  END
END;

OVERLAY FUNCTION build(name:str_typ):str_typ;
{build binary micro load file from text mld format -- one cycle}
VAR s:str_typ; r:bin_load_typ; dat:bits_typ; i,c:BYTE; err:BOOLEAN;
BEGIN
  IF name<>'*' THEN BEGIN
    IF POS('.',name)=0 THEN name:=name+'.MLD';
    build_name:=name;
    ASSIGN(macro_file,build_name); {$I-} RESET(macro_file); {$I+}
    IF IORESULT<>0 THEN build_name:=''
  END;
  IF build_name='' THEN build:='?No File: '+name
  ELSE BEGIN
    IF name<>'*' THEN BEGIN
      ASSIGN(micro_file,COPY(name,1,POS('.',name)-1)+'.BIN');
      REWRITE(micro_file)
    END;
```

```
    c:=0; err:=FALSE;
    WHILE NOT (EOF(macro_file) OR (c>16)) AND NOT err DO BEGIN
      READLN(macro_file,s); c:=c+1;
      IF COPY(s,1,4)<>'MMLD' THEN err:=TRUE;
      r.addr:=oct_int(COPY(s,6,6)); oct_to_bits(COPY(s,13,30),dat,88);
      {NOTE: no umem 40-3777 and 20000-27777}
      IF r.addr<32 THEN BEGIN {fill in holes 0-37}
        fit_bits(dat,88,1,left_fit);
        FOR i:=16 TO 35 DO dat.bit[i]:=1;
        FOR i:=50 TO 61 DO dat.bit[i]:=1;
        FOR i:=86 TO 87 DO dat.bit[i]:=0
      END;
      fit_bits(dat,11,8,left_fit);
      dat.bit[10]:=dat.bit[10] XOR $03;
      FOR i:=0 TO 10 DO r.dat[i]:=dat.bit[i];
      WRITE(micro_file,r)
    END;
    IF err THEN BEGIN
      build:='?Invalid MLD file: '+build_name; build_name:=''
    END
    ELSE IF EOF(macro_file) THEN BEGIN
      CLOSE(macro_file); CLOSE(micro_file); build:='OK'; build_name:=''
    END
    ELSE build:=int_oct(r.addr)
  END
END;

OVERLAY FUNCTION instrs(name:str_typ):str_typ;
{load micro instruction table from binary load file}
VAR f:FILE OF bin_load_typ; r:bin_load_typ; i:mi_typ; j:BYTE; err:BOOLEAN;
BEGIN
  IF POS('.',name)=0 THEN name:=name+'.BIN';
  ASSIGN(f,name); {$I-} RESET(f); {$I+}
  IF IORESULT<>0 THEN instrs:='?No File: '+name
  ELSE BEGIN
    err:=FALSE;
    FOR i:=clcpmod_mi TO zero_mi DO BEGIN
      IF EOF(f) THEN err:=TRUE ELSE READ(f,r);
      WITH mi_tab[i].bits DO BEGIN
        siz:=11; bit_siz:=8; FOR j:=0 TO 10 DO bit[j]:=r.dat[j]
      END
    END;
    CLOSE(f);
    IF err THEN instrs:= '?EOF in: '+name+'.BIN'
    ELSE instrs:='Instructions: '+name
  END
END;

OVERLAY FUNCTION dump(micro:BOOLEAN; a1,a2:str_typ):str_typ;
{dump micro/macro memory to binary/text file -- one cycle}

PROCEDURE micro_dump(addr,ending:INTEGER);
{dump micro memory to binary file}
VAR i,j:BYTE; d:bits_typ; r:bin_load_typ; done:BOOLEAN;
BEGIN
  done:=FALSE;
  FOR i:=1 TO 16 DO IF NOT done THEN BEGIN
```

```
    rdum(addr,d); fit_bits(d,11,8,left_fit);
    r.addr:=addr; FOR j:=0 TO 10 DO r.dat[j]:=d.bit[j];
    WRITE(micro_file,r);
    done:=addr=ending; IF NOT done THEN addr:=addr+1
  END;
  IF addr<>ending THEN dump:=int_oct(addr)
END;

PROCEDURE macro_dump(addr,ending:str_typ);
{dump macro memory to text file}
VAR i:BYTE; a,d:bits_typ; done:BOOLEAN;
BEGIN
  done:=FALSE;
  FOR i:=1 TO 16 DO IF NOT done THEN BEGIN
    oct_to_bits(addr,a,24); rddma(a,d);
    WRITELN(macro_file,addr,',',bits_to_oct(d,36));
    done:=addr=ending; IF NOT done THEN addr:=incr_oct(addr)
  END;
  IF addr<>ending THEN dump:=addr
END;

BEGIN {dump}
  dump:='OK';
  IF a1='' THEN BEGIN
    IF dump_name<>'' THEN BEGIN
      IF micro THEN CLOSE(micro_file) ELSE CLOSE(macro_file);
      dump_name:=''
    END
  END
  ELSE IF a2='' THEN BEGIN
    IF dump_name<>'' THEN dump:='?File open.'
    ELSE BEGIN
      dump_name:=a1+switch(POS('.',a1)=0,switch(micro,'.BIN','.OCT'),'');
      IF micro THEN BEGIN
        ASSIGN(micro_file,dump_name); REWRITE(micro_file)
      END
      ELSE BEGIN
        ASSIGN(macro_file,dump_name); REWRITE(macro_file)
      END
    END
  END
  ELSE IF dump_name='' THEN dump:='?File not open.'
  ELSE BEGIN
    IF micro THEN micro_dump(oct_int(a1),oct_int(a2))
    ELSE macro_dump(a1,a2)
  END
END;

OVERLAY FUNCTION load(micro:BOOLEAN; name:str_typ; code:BYTE):str_typ;
{load micro/macro memory from text/binary file -- one cycle}

PROCEDURE macro_err(a,d1,d2:bits_typ);
{error in loading macro memory}
BEGIN
  WRITE(load_errors,'Bad memory at ',bits_to_oct(a,24));
  WRITELN(load_errors,' is: ',bits_to_oct(d1,36));
  WRITELN(load_errors,'              Should be: ',bits_to_oct(d2,36));
```

```
  WRITELN(load_errors);
  load_error:=TRUE
END;

PROCEDURE micro_err(r:bin_load_typ; dat:bits_typ);
{error in loading micro memory}
VAR i:BYTE;
BEGIN
  WRITE(load_errors,'Bad memory at ',
    oct_digits(int_oct(r.addr),6));
  WRITELN(load_errors,' is: ',oct_digits(bits_to_oct(dat,88),30));
  FOR i:=0 TO 10 DO dat.bit[i]:=r.dat[i];
  WRITELN(load_errors,'                Should be: ',
    oct_digits(bits_to_oct(dat,88),30));
  WRITELN(load_errors);
  load_error:=TRUE
END;

PROCEDURE macro_load;
{load macro memory}
VAR a,d:bits_typ; c:BYTE; l:str_typ;
BEGIN
  c:=0;
  WHILE NOT EOF(macro_file) AND (c<32) DO BEGIN
    READ(macro_file,l); c:=c+1;
    oct_to_bits(parse(l),a,24); oct_to_bits(parse(l),d,36);
    wrdma(a,d)
  END;
  IF c>0 THEN load:=bits_to_oct(a,24)
END;

PROCEDURE macro_verify;
{load and varify macro memory}
VAR a,d1,d2:bits_typ; c:BYTE; l:str_typ;
BEGIN
  c:=0;
  WHILE NOT EOF(macro_file) AND (c<16) DO BEGIN
    READ(macro_file,l); c:=c+1;
    oct_to_bits(parse(l),a,24); oct_to_bits(parse(l),d1,36);
    wrdma(a,d1); rddma(a,d2);
    fit_bits(d2,36,1,left_fit); fit_bits(d1,36,1,left_fit);
    IF NOT equal_bits(d2,d1) THEN macro_err(a,d2,d1)
  END;
  IF c>0 THEN load:=bits_to_oct(a,24)
END;

PROCEDURE macro_check;
{check previously loaded macro memory}
VAR a,d1,d2:bits_typ; c:BYTE; l:str_typ;
BEGIN
  c:=0;
  WHILE NOT EOF(macro_file) AND (c<32) DO BEGIN
    READ(macro_file,l); c:=c+1;
    oct_to_bits(parse(l),a,24); oct_to_bits(parse(l),d1,36);
    rddma(a,d2);
    fit_bits(d2,36,1,left_fit); fit_bits(d1,36,1,left_fit);
    IF NOT equal_bits(d2,d1) THEN macro_err(a,d2,d1)
```

```
  END;
  IF c>0 THEN load:=bits_to_oct(a,24)
END;

PROCEDURE micro_load;
{slow load of micro memory}
VAR first:BOOLEAN; i,c:BYTE; d:bits_typ; r:bin_load_typ;
BEGIN
  c:=0; first:=TRUE;
  WHILE NOT EOF(micro_file) AND (c<31) DO BEGIN
    READ(micro_file,r); c:=c+1;
    WITH d,r DO BEGIN
      siz:=11; bit_siz:=8; FOR i:=0 TO 10 DO bit[i]:=dat[i]
    END;
    wrum(r.addr,d,sect_ABC,first)
  END;
  IF c>0 THEN load:=int_oct(r.addr)
END;

PROCEDURE micro_verify;
{load and verify micro memory}
VAR dat:bits_typ; i,c:BYTE; first,ok:BOOLEAN; r:bin_load_typ;
BEGIN
  c:=0;
  WHILE NOT EOF(micro_file) AND (c<17) DO BEGIN
    READ(micro_file,r); c:=c+1;
    WITH dat,r DO BEGIN
      siz:=11; bit_siz:=8; FOR i:=0 TO 10 DO bit[i]:=dat[i]
    END;
    first:=TRUE; wrum(r.addr,dat,sect_ABC,first);
    rdum(r.addr,dat);
    fit_bits(dat,11,8,left_fit); ok:=TRUE;
    FOR i:=0 TO 10 DO ok:=ok AND (dat.bit[i]=r.dat[i]);
    IF NOT ok THEN micro_err(r,dat)
  END;
  IF c>0 THEN load:=int_oct(r.addr)
END;

PROCEDURE micro_check;
{check previously loaded micro memory}
VAR dat:bits_typ; i,c:BYTE; ok:BOOLEAN; r:bin_load_typ;
BEGIN
  c:=0;
  WHILE NOT EOF(micro_file) AND (c<31) DO BEGIN
    READ(micro_file,r); c:=c+1;
    rdum(r.addr,dat); fit_bits(dat,11,8,left_fit); ok:=TRUE;
    FOR i:=0 TO 10 DO ok:=ok AND (dat.bit[i]=r.dat[i]);
    IF NOT ok THEN micro_err(r,dat);
  END;
  IF c>0 THEN load:=int_oct(r.addr)
END;

PROCEDURE micro_fast;
{fast load of micro memory}
VAR r:bin_load_typ; c:BYTE;
BEGIN
  c:=0;
```

```
  WHILE NOT EOF(micro_file) AND (c<167) DO BEGIN
    READ(micro_file,r); c:=c+1;
    WITH r DO BEGIN
      PORT[port_0]:=0; PORT[port_1]:=10; PORT[port_2]:=1; {addr is 5001}
      PORT[port_3]:=0;
      PORT[port_4]:=dat[0]; PORT[port_5]:=dat[1];
      PORT[port_6]:=dat[2]; PORT[port_7]:=dat[3];
      PORT[dma_strb_port]:=1; {wr first four bytes}
      PORT[port_2]:=2; {addr is 5002}
      PORT[port_4]:=dat[4]; PORT[port_5]:=dat[5];
      PORT[port_6]:=dat[6]; PORT[port_7]:=dat[7];
      PORT[dma_strb_port]:=1; {wr next four bytes}
      PORT[port_2]:=3; {addr is 5003}
      PORT[port_4]:=0; PORT[port_5]:=dat[8]; PORT[port_6]:=dat[9];
      PORT[port_7]:=dat[10] XOR $03;
      PORT[dma_strb_port]:=1; {wr last three bytes}
      PORT[port_2]:=0; {addr is 5000}
      PORT[port_3]:=0; PORT[port_4]:=0;
      PORT[port_5]:=4; {control flag bit 17 set}
      PORT[port_6]:=addr SHR 8;
      PORT[port_7]:=addr AND $FF; {addr right justified}
      PORT[dma_strb_port]:=1 {wr addr and control flag}
    END
  END;
  IF c>0 THEN load:=int_oct(r.addr)
END;

VAR done:BOOLEAN;
BEGIN {load}
  IF name<>'' THEN BEGIN
    load_error:=FALSE; load_code:=code;
    IF POS('.',name)=0 THEN name:=name+switch(micro,'.BIN','.OCT');
    IF micro THEN BEGIN
      ASSIGN(micro_file,name); {$I-} RESET(micro_file) {$I+}
    END
    ELSE BEGIN
      ASSIGN(macro_file,name); {$I-} RESET(macro_file) {$I+}
    END;
    IF IORESULT=0 THEN load_name:=COPY(name,1,POS('.',name)-1)
    ELSE load_name:=''
  END;
  IF load_name='' THEN load:='?No file: '+name
  ELSE BEGIN
    IF (load_code IN [load_verify,load_check]) AND (name<>'') THEN BEGIN
      ASSIGN(load_errors,load_name+'.ERR'); REWRITE(load_errors)
    END;
    IF micro THEN BEGIN
      CASE load_code OF
        load_slow:micro_load; load_verify:micro_verify;
        load_check:micro_check; load_fast:micro_fast
      END;
      done:=EOF(micro_file); IF done THEN CLOSE(micro_file)
    END
    ELSE BEGIN
      CASE load_code OF
        load_slow:macro_load; load_verify:macro_verify;
        load_check:macro_check
```

```
      END;
      done:=EOF(macro_file); IF done THEN CLOSE(macro_file)
    END;
    IF done THEN BEGIN
      IF load_error THEN BEGIN
        CLOSE(load_errors); load:='?Error(s) recorded in: '+load_name+'.ERR'
      END
      ELSE BEGIN
        IF load_code IN [load_verify,load_check] THEN BEGIN
          CLOSE(load_errors); ERASE(load_errors)
        END;
        load:='OK'
      END;
      load_name:=''
    END
  END
END;

OVERLAY PROCEDURE reset_function;
{reset the system}
VAR zero,addr:bits_typ;
BEGIN
  exec(mi_tab[clexctx_mi].bits,TRUE); exec(mi_tab[clusctx_mi].bits,TRUE);
  exec(mi_tab[clmerge_mi].bits,TRUE); exec(mi_tab[clvamod_mi].bits,TRUE);
  exec(mi_tab[clcpmod_mi].bits,TRUE); exec(mi_tab[mapdis_mi].bits,TRUE);
  exec(mi_tab[cljmemp_mi].bits,TRUE); exec(mi_tab[ldqones_mi].bits,TRUE);
  exec(mi_tab[ldlpq_mi].bits,TRUE); exec(mi_tab[luhctl_mi].bits,TRUE);
  exec(mi_tab[lmhctl_mi].bits,TRUE); exec(mi_tab[luhadr_mi].bits,TRUE);
  exec(mi_tab[lmhadr_mi].bits,TRUE);
  IF kl THEN BEGIN
    exec(mi_tab[klres1_mi].bits,TRUE);
    exec(mi_tab[klres2_mi].bits,TRUE)
  END;
  oct_to_bits('0',zero,36);
  wrreg(pc_reg,zero); wrreg(ma_reg,zero); wrreg(ir_reg,zero);
  wrreg(ar_reg,zero); wrreg(hold_reg,zero); wrreg(q_reg,zero);
  wrreg(dev_reg,zero); wrreg(ac_reg,zero); {ac[dev]}
  wrreg(amemp_reg,zero); wrreg(jmemp_reg,zero);
  wrmi(mi_tab[zero_mi].bits,sect_ABC);
  wrob(zero,TRUE); wrmic(oct_int('4043'),TRUE,TRUE);
  oct_to_bits('0',addr,24); wrdma(addr,zero);
  oct_to_bits('1',addr,24); wrdma(addr,zero);
  oct_to_bits('2',addr,24); wrdma(addr,zero);
  oct_to_bits('3',addr,24); wrdma(addr,zero);
  oct_to_bits('4',addr,24); wrdma(addr,zero);
  oct_to_bits('5',addr,24); wrdma(addr,zero);
  oct_to_bits('6',addr,24); wrdma(addr,zero);
  oct_to_bits('7',addr,24); wrdma(addr,zero);
  oct_to_bits('10',addr,24); wrdma(addr,zero);
  oct_to_bits('11',addr,24); wrdma(addr,zero);
  oct_to_bits('12',addr,24); wrdma(addr,zero);
  oct_to_bits('13',addr,24); wrdma(addr,zero);
  oct_to_bits('14',addr,24); wrdma(addr,zero);
  oct_to_bits('15',addr,24); wrdma(addr,zero);
  oct_to_bits('16',addr,24); wrdma(addr,zero);
  oct_to_bits('17',addr,24); wrdma(addr,zero);
  do_save(TRUE,TRUE); do_save(FALSE,FALSE)
```

```
LISTING: CONSOLE.PAS  01-JAN-80  00:00:00  CONSOLE.PAS;

END;

OVERLAY FUNCTION break(micro:BOOLEAN; args:str_typ):str_typ;
{set/clear a micro/macro break point}

PROCEDURE enable_break(micro,yes:BOOLEAN);
{enable/disable micro/macro break}
BEGIN
  WITH local_flags DO IF micro THEN BEGIN
    IF yes THEN init0:=init0 OR muhs_brk_stop_sw
    ELSE init0:=init0 AND NOT muhs_brk_stop_sw
  END
  ELSE BEGIN
    IF yes THEN init0:=init0 OR hs_brk_stop_sw
    ELSE init0:=init0 AND NOT hs_brk_stop_sw
  END;
  inits(FALSE)
END;

PROCEDURE set_macro_break(VAR addr,macro_count,delay:bits_typ);
{set up a macro break point}
VAR mi,b:bits_typ;
BEGIN
  save;
  enable_break(FALSE,FALSE);
  mi:=mi_tab[lmhbkad_mi].bits; setfld(lit_fld,mi,addr);
  exec(mi,TRUE);
  mi:=mi_tab[lmhctl_mi].bits;
  oct_to_bits('20141140000',b,36); setfld(lit_fld,mi,b);
  exec(mi,TRUE);
  mi:=mi_tab[lmhcnt_mi].bits;
  oct_to_bits('0',b,36); put_bits(b,macro_count,4,8); put_bits(b,delay,16,_0)
  setfld(lit_fld,mi,b);
  exec(mi,TRUE);
  enable_break(FALSE,TRUE);
  restore
END;

PROCEDURE set_micro_break(VAR addr,micro_count:bits_typ);
{set up a micro break point}
VAR mi,b:bits_typ;
BEGIN
  save;
  enable_break(TRUE,FALSE);
  mi:=mi_tab[luhctl_mi].bits; fit_bits(addr,18,1,right_fit);
  oct_to_bits('704340',b,36); put_bits(b,addr,0,18);
  setfld(lit_fld,mi,b); oct_to_bits('2',b,36); setfld(mapf_fld,mi,b);
  exec(mi,TRUE);
  oct_to_bits('0',b,36); put_bits(b,micro_count,24,8);
  setfld(lit_fld,mi,b); oct_to_bits('0',b,36); setfld(mapf_fld,mi,b);
  exec(mi,TRUE);
  setfld(lit_fld,mi,b); oct_to_bits('1',b,36); setfld(mapf_fld,mi,b);
  exec(mi,TRUE);
  enable_break(TRUE,TRUE);
  restore
END;
```

```
VAR addr,brk_count,delay:bits_typ;
BEGIN {break}
  IF args='OFF' THEN enable_break(micro,FALSE)
  ELSE IF args='ON' THEN enable_break(micro,TRUE)
  ELSE IF micro THEN BEGIN
    oct_to_bits(parse(args),addr,14);
    default(args,'1'); oct_to_bits(args,brk_count,8);
    args:=bits_to_oct(addr,14);
    set_micro_break(addr,brk_count)
  END
  ELSE BEGIN
    oct_to_bits(parse(args),addr,18);
    default(args,'1'); oct_to_bits(parse(args),brk_count,8);
    default(args,'0'); oct_to_bits(parse(args),delay,20);
    args:=bits_to_oct(addr,18);
    set_macro_break(addr,brk_count,delay)
  END;
  break:=args
END;

OVERLAY FUNCTION test(code:BYTE):str_typ;
{test various console operations}
VAR failed:BOOLEAN;

PROCEDURE test_ret(ok:BOOLEAN; rec,exp:str_typ);
{build up return message}
BEGIN
  IF NOT failed THEN BEGIN
    failed:=NOT ok; IF failed THEN test:=unparse(rec,exp,'')
  END
END;

PROCEDURE byte_test(rec,exp:BYTE);
{test bytes for equality}
BEGIN test_ret(rec=exp,int_oct(rec),int_oct(exp)) END;

PROCEDURE bits_test(rec,exp:bits_typ);
{test bits for equality}
BEGIN
  fit_bits(rec,exp.siz,exp.bit_siz,left_fit);
  test_ret(equal_bits(rec,exp),
    bits_to_oct(rec,rec.siz*rec.bit_siz),
    bits_to_oct(exp,exp.siz*exp.bit_siz))
END;

VAR i:1..4; addr,exp,rec:bits_typ; first:BOOLEAN;
BEGIN {test}
  test:='OK'; failed:=FALSE;
  {codes 1 to 28 crash initial values}
  {codes > 28 must have stable initial values}
  CASE code OF
    {what about the remote parity error???}
    {how should we react to parity errors in general???}
    1:BEGIN {cable test, bit set if installed}
      PORT[iorb_port]:=cc_res; byte_test(PORT[iorb_port],cc_res)
    END;
    2:BEGIN {reset local,clear local parity error}
```

```
    PORT[loc_res_port]:=0; byte_test(PORT[loc_sts_port] AND loc_par_err,$00
  END;
  3:BEGIN {test 3 should have cleared diag dat bits}
    byte_test(PORT[diag_port],0)
  END;
  4:FOR i:=1 TO 4 DO BEGIN {load and check diag dat}
    PORT[diag_port]:=test_byte[i]; byte_test(PORT[diag_port],test_byte[i
  END;
  5:BEGIN {force local parity error}
    PORT[loc_err_port]:=0;
    byte_test(PORT[rem_sts_port] AND loc_par_err,loc_par_err)
  END;
  6:FOR i:=1 TO 4 DO BEGIN {dat bank 0}
    PORT[port_0]:=test_byte[i]; byte_test(PORT[port_0],test_byte[i])
  END;
  7:FOR i:=1 TO 4 DO BEGIN {dat bank 1}
    PORT[port_1]:=test_byte[i]; byte_test(PORT[port_1],test_byte[i])
  END;
  8:FOR i:=1 TO 4 DO BEGIN {dat bank 2}
    PORT[port_2]:=test_byte[i]; byte_test(PORT[port_2],test_byte[i])
  END;
  9:FOR i:=1 TO 4 DO BEGIN {dat bank 3}
    PORT[port_3]:=test_byte[i]; byte_test(PORT[port_3],test_byte[i])
  END;
  10:FOR i:=1 TO 4 DO BEGIN {dat bank 4}
    PORT[port_4]:=test_byte[i]; byte_test(PORT[port_4],test_byte[i])
  END;
  11:FOR i:=1 TO 4 DO BEGIN {dat bank 5}
    PORT[port_5]:=test_byte[i]; byte_test(PORT[port_5],test_byte[i])
  END;
  12:FOR i:=1 TO 4 DO BEGIN {dat bank 6}
    PORT[port_6]:=test_byte[i]; byte_test(PORT[port_6],test_byte[i])
  END;
  13:FOR i:=1 TO 4 DO BEGIN {dat bank 7}
    PORT[port_7]:=test_byte[i]; byte_test(PORT[port_7],test_byte[i])
  END;
  14:BEGIN {rd zeros from shifts out bits}
    PORT[port_0]:=0; PORT[port_1]:=0; PORT[port_2]:=0; PORT[port_3]:=0;
    PORT[port_4]:=0; PORT[port_5]:=0; PORT[port_6]:=0; PORT[port_7]:=0;
    byte_test(PORT[rd_shifts_port],$00)
  END;
  15:BEGIN {rd ones from shifts out bits}
    PORT[port_0]:=1; PORT[port_1]:=1; PORT[port_2]:=1; PORT[port_3]:=1;
    PORT[port_4]:=1; PORT[port_5]:=1; PORT[port_6]:=1; PORT[port_7]:=1;
    byte_test(PORT[rd_shifts_port],$FF)
  END;
  16:BEGIN {load shift bank 0 to bank 1 with count of 1}
    PORT[port_0]:=$01; PORT[port_1]:=0;
    shift(1); byte_test(PORT[port_1],$80)
  END;
  17:BEGIN {load shift bank 1 to bank 2 with count of 2}
    PORT[port_1]:=$02; PORT[port_2]:=0;
    shift(2); byte_test(PORT[port_2],$80)
  END;
  18:BEGIN {load shift bank 2 to bank 3 with count of 4}
    PORT[port_2]:=$08; PORT[port_3]:=0;
    shift(4); byte_test(PORT[port_3],$80)
```

```
    END;
    19:BEGIN {load shift bank 3 to bank 4 with count of 8}
      PORT[port_3]:=$80; PORT[port_4]:=0;
      shift(8); byte_test(PORT[port_4],$80)
    END;
    20:BEGIN {load shift bank 4 to bank 5 with count of 1}
      PORT[port_4]:=$01; PORT[port_5]:=0;
      shift(1); byte_test(PORT[port_5],$80)
    END;
    21:BEGIN {load shift bank 5 to bank 6 with count of 2}
      PORT[port_5]:=$02; PORT[port_6]:=0;
      shift(2); byte_test(PORT[port_6],$80)
    END;
    22:BEGIN {load shift bank 6 to bank 7 with count of 4}
      PORT[port_6]:=$08; PORT[port_7]:=0;
      shift(4); byte_test(PORT[port_7],$80)
    END;
    23:BEGIN {load shift bank 7 to bank 0 with count of 8}
      PORT[port_7]:=$80; PORT[port_0]:=0;
      PORT[diag_loop_port]:=0;
      shift(8); byte_test(PORT[port_0],$80);
      PORT[rem_res_port]:=0
    END;
    24:BEGIN {load shift bank 0 to bank 2 with count of 16}
      PORT[port_0]:=$80; PORT[port_1]:=0; PORT[port_2]:=0;
      shift(16); byte_test(PORT[port_2],$80)
    END;
    25:BEGIN {load shift bank 0 to bank 4 with count of 32}
      PORT[port_0]:=$80; PORT[port_1]:=0; PORT[port_2]:=0;
      PORT[port_3]:=0; PORT[port_4]:=0;
      shift(32); byte_test(PORT[port_4],$80)
    END;
    26:BEGIN {load shift bank 0 to bank 0 with count of 64}
      PORT[port_0]:=$80; PORT[port_1]:=0; PORT[port_2]:=0; PORT[port_3]:=0;
      PORT[port_4]:=0; PORT[port_5]:=0; PORT[port_6]:=0; PORT[port_7]:=0;
      PORT[diag_loop_port]:=0;
      shift(64); byte_test(PORT[port_0],$80);
      PORT[rem_res_port]:=0
    END;
    27:FOR i:=1 TO 4 DO BEGIN {iora}
      PORT[iora_port]:=test_byte[i]; byte_test(PORT[iora_port],test_byte[i])
    END;
    28:FOR i:=1 TO 4 DO BEGIN {iorb}
      PORT[iorb_port]:=test_byte[i]; byte_test(PORT[iorb_port],test_byte[i])
    END;
    29:FOR i:=1 TO 4 DO BEGIN {obus}
      get_bits(test_bits[i],exp,0,36);
      wrob(exp,TRUE); rdob(rec,FALSE,FALSE); bits_test(rec,exp)
    END;
    30:BEGIN {dma}
      oct_to_bits('30',addr,24);
      FOR i:=1 TO 4 DO BEGIN
        get_bits(test_bits[i],exp,0,36);
        wrdma(addr,exp); rddma(addr,rec); bits_test(rec,exp)
      END
    END;
    31:FOR i:=1 TO 4 DO BEGIN {mi}
```

```
      get_bits(test_bits[i],exp,0,88);
      wrmi(exp,sect_ABC); rdmi(rec); bits_test(rec,exp)
    END;
    32:FOR i:=1 TO 4 DO BEGIN {mic}
      get_bits(test_bits[i],exp,0,14); fit_bits(exp,16,1,right_fit);
      rec:=exp; wrmic(exp.bit[0],TRUE,FALSE); rec:=exp; rec.bit[0]:=rdmic;
      bits_test(rec,exp)
    END;
    33:BEGIN {umem}
      oct_to_bits('4043',addr,14); fit_bits(addr,1,16,right_fit);
      FOR i:=1 TO 4 DO BEGIN
        get_bits(test_bits[i],exp,0,88);
        first:=TRUE; wrum(addr.bit[0],exp,sect_ABC,first);
        rdum(addr.bit[0],rec); bits_test(rec,exp)
      END
    END;
    34:FOR i:=1 TO 4 DO BEGIN {PC register}
      get_bits(test_bits[i],exp,0,36);
      wrreg(pc_reg,exp); rdreg(pc_reg,rec); bits_test(rec,exp)
    END;
    35:FOR i:=1 TO 4 DO BEGIN {MA register}
      get_bits(test_bits[i],exp,0,36);
      wrreg(ma_reg,exp); rdreg(ma_reg,rec); bits_test(rec,exp)
    END;
    36:FOR i:=1 TO 4 DO BEGIN {AR register}
      get_bits(test_bits[i],exp,0,36);
      wrreg(ar_reg,exp); rdreg(ar_reg,rec); bits_test(rec,exp)
    END;
    37:FOR i:=1 TO 4 DO BEGIN {MEM register}
      get_bits(test_bits[i],exp,0,36);
      wrreg(hold_reg,exp); rdreg(mem_reg,rec); bits_test(rec,exp)
    END;
    38:FOR i:=1 TO 4 DO BEGIN {IR register}
      get_bits(test_bits[i],exp,0,36);
      wrreg(ir_reg,exp); rdreg(ir_reg,rec); bits_test(rec,exp)
    END;
    39:FOR i:=1 TO 4 DO BEGIN {Q register}
      get_bits(test_bits[i],exp,0,36);
      wrreg(q_reg,exp); rdreg(q_reg,rec); bits_test(rec,exp)
    END;
    40:FOR i:=1 TO 4 DO BEGIN {JMEMP register}
      get_bits(test_bits[i],exp,0,12); fit_bits(exp,36,1,right_fit);
      wrreg(jmemp_reg,exp); rdreg(jmemp_reg,rec); bits_test(rec,exp)
    END;
    41:FOR i:=1 TO 4 DO BEGIN {DEV register}
      get_bits(test_bits[i],exp,0,5); fit_bits(exp,36,1,right_fit);
      wrreg(dev_reg,exp); rdreg(dev_reg,rec); bits_test(rec,exp)
    END;
    42:FOR i:=1 TO 4 DO BEGIN {AMEMP register}
      get_bits(test_bits[i],exp,0,10); fit_bits(exp,36,1,right_fit);
      wrreg(amemp_reg,exp); rdreg(amemp_reg,rec); bits_test(rec,exp)
    END;
    43:BEGIN {ac register}
      oct_to_bits('0',addr,4);
      FOR i:=1 TO 4 DO BEGIN
        get_bits(test_bits[i],exp,0,36);
        wrac(addr,exp); rdac(addr,rec); bits_test(rec,exp)
```

```
      END
    END;
    44:BEGIN {amem}
      oct_to_bits('16',addr,10);
      FOR i:=1 TO 4 DO BEGIN
        get_bits(test_bits[i],exp,0,36);
        wram(addr,exp,FALSE); rdam(addr,rec,FALSE); bits_test(rec,exp)
      END
    END;
    45:BEGIN {memory}
      oct_to_bits('30',addr,18);
      FOR i:=1 TO 4 DO BEGIN
        get_bits(test_bits[i],exp,0,36);
        wrm(addr,exp); rdm(addr,rec); bits_test(rec,exp)
      END
    END
  END
END;

VAR
code,arg1,arg2,arg2s,arg3:str_typ; {arg1s is input parameter to backend}
ret_ok,first:BOOLEAN;
field:fld_typ;
mic:INTEGER;
sect:BYTE;
eobus,obus,ac,bytes,index,ui,addr,dat:bits_typ;

PROCEDURE ret(msg:str_typ);
{set return value for backend}
BEGIN
  IF timed_out THEN backend:='?shift'
  ELSE IF bad_digit THEN backend:='?digit1'
  ELSE IF digit_overflow THEN backend:='?digit2'
  ELSE IF bad_reg<>'' THEN backend:='?register"'+bad_reg+'"'
  ELSE IF bad_fld<>'' THEN backend:='?field"'+bad_fld+'"'
  ELSE IF NOT ret_ok THEN BEGIN backend:=msg; ret_ok:=TRUE END
END;

{utility for parsing binary arguments}
FUNCTION barg(arg:str_typ):BOOLEAN; BEGIN barg:=eq(arg,'TRUE',1) END;
FUNCTION barg1:BOOLEAN; BEGIN barg1:=barg(arg1) END;
FUNCTION barg2:BOOLEAN; BEGIN barg2:=barg(arg2) END;

BEGIN {backend}
  bad_digit:=FALSE; digit_overflow:=FALSE; timed_out:=FALSE;
  bad_reg:=''; bad_fld:=''; ret_ok:=FALSE; code:=parse(arg1s);
  arg2s:=arg1s; arg1:=parse(arg2s); arg3:=arg2s; arg2:=parse(arg3);
  CASE cmd_typ(dec_int(code)) OF
    bits_fun:ret(bits(arg1s));
    break_fun:ret(break(barg1,arg2));
    build_fun:ret(build(arg1));
    cpuob_fun:cpuob(barg1);
    cty_fun:ret(cty(barg1));
    dump_fun:ret(dump(barg1,arg2,arg3));
    enahst_fun:enahst(barg1);
    exec_fun:BEGIN mi(arg1,ui); exec(ui,barg2) END;
    fld_fun:ret(int_dec(ORD(fld(arg1))));
```

```
  init_fun:BEGIN ret(parse_inits(arg2s)); init(barg1) END;
  inits_fun:BEGIN ret(parse_inits(arg2s)); inits(barg1) END;
  init2_fun:BEGIN
    IF arg1<>'' THEN init2(oct_int(arg1) AND $FF,oct_int(arg2) AND $FF);
    ret(int_oct(local_flags.init2))
    ;debug('INIT2: '+hex_word(OFS(init2))); debug_break
  END;
  instrs_fun:ret(instrs(arg1));
  iora_fun:BEGIN
    IF arg1<>'' THEN iora(oct_int(arg1) AND $FF,oct_int(arg2) AND $FF);
    ret(oct_digits(int_oct(PORT[iora_port]),3))
  END;
  iorb_fun:BEGIN
    IF arg1<>'' THEN iorb(oct_int(arg1) AND $FF,oct_int(arg2) AND $FF);
    ret(oct_digits(int_oct(PORT[iorb_port]),3))
  END;
  jump_fun:BEGIN jump(oct_int(arg1),ui); ret(bits_to_oct(ui,88)) END;
  load_fun:ret(load(barg1,arg2,dec_int(arg3)));
  mi_fun:BEGIN mi(arg1,ui); ret(bits_to_oct(ui,88)) END;
  mmactrl_fun:BEGIN
    IF arg1<>'' THEN mmactrl(oct_int(arg1) AND $FF,oct_int(arg2) AND $FF);
    ret(int_oct(local_flags.mma_ctrl))
  END;
  rdac_fun:BEGIN
    oct_to_bits(arg1,ac,4); rdac(ac,dat); ret(bits_to_oct(dat,36))
  END;
  rdam_fun:BEGIN
    oct_to_bits(arg1,addr,14); rdam(addr,dat,barg2); ret(bits_to_oct(dat,36
  END;
  rddma_fun:BEGIN
    oct_to_bits(arg1,addr,24); rddma(addr,dat); ret(bits_to_oct(dat,36)
  END;
  rdecc_fun:ret(int_oct(rdecc));
  rdhst_fun:BEGIN
    rdhst(dat,barg1);
    IF barg1 THEN ret(bits_to_oct(dat,36)) ELSE ret(bits_to_oct(dat,40))
  END;
  rdhsti_fun:BEGIN rdhsti(index,barg1,barg2); ret(bits_to_oct(index,10)) EN
  rdm_fun:BEGIN
    oct_to_bits(arg1,addr,18); rdm(addr,dat); ret(bits_to_oct(dat,36))
  END;
  rdmi_fun:BEGIN rdmi(ui); ret(bits_to_oct(ui,88)) END;
  rdmic_fun:ret(int_oct(rdmic));
  rdob_fun:BEGIN rdob(obus,barg1,barg2); ret(bits_to_oct(obus,36)) END;
  rdport_fun:ret(int_oct(PORT[oct_int(arg1)]));
  rdreg_fun:BEGIN rdreg(reg(arg1),dat); ret(bits_to_oct(dat,36)) END;
  rdst_fun:WITH state DO BEGIN
    rdstlp(dat);
    ret(version+', '+switch(running,'TRUE','FALSE')+', '+
      switch(last_cond,'TRUE','FALSE')+', '+
      switch(this_cond,'TRUE','FALSE')+', '+
      switch(branching,'TRUE','FALSE'))
  END;
  rdstlp_fun:BEGIN rdstlp(dat); ret(bits_to_oct(dat,128)) END;
  rdum_fun:BEGIN rdum(oct_int(arg1),dat); ret(bits_to_oct(dat,88)) END;
  reg_fun:ret(int_dec(ORD(reg(arg1))));
  reset_fun:BEGIN init(TRUE); reset_function END;
```

```
    run_fun:run(barg1);
    save_fun:BEGIN do_save(barg1,barg2); ret(int_dec(saved.count)) END;
    setfld_fun:BEGIN
      mi(arg1,ui); field:=fld(arg2); oct_to_bits(arg3,dat,36);
      setfld(field,ui,dat); ret(bits_to_oct(ui,88))
    END;
    shift_fun:shift(dec_int(arg1));
    step_fun:step(barg1,dec_int(arg2));
    sysid_fun:BEGIN
      IF barg1 THEN ret(int_dec(sysid(TRUE)))
      ELSE ret(switch(sysid(FALSE)=1,'KL',''))
    END;
    talk_fun:ret(talk(barg1,arg2s));
    test_fun:BEGIN
      IF dec_int(arg1)<29 THEN init_ok:=FALSE
      ELSE IF NOT init_ok THEN init(TRUE);
      ret(test(dec_int(arg1)))
    END;
    wrac_fun:BEGIN
      oct_to_bits(arg1,ac,4); oct_to_bits(arg2,dat,36); wrac(ac,dat)
    END;
    wram_fun:BEGIN
      oct_to_bits(arg1,addr,14); oct_to_bits(arg2,dat,36);
      wram(addr,dat,barg(arg3))
    END;
    wrdma_fun:BEGIN
      oct_to_bits(arg1,addr,24); oct_to_bits(arg2,dat,36); wrdma(addr,dat)
    END;
    wrhsti_fun:BEGIN
      oct_to_bits(arg1,index,10); wrhsti(index,barg2,barg(arg3))
    END;
    wrm_fun:BEGIN
      oct_to_bits(arg1,addr,18); oct_to_bits(arg2,dat,36); wrm(addr,dat)
    END;
    wrmi_fun:BEGIN
      mi(arg1,ui);
      IF arg2='' THEN sect:=sect_ABC ELSE sect:=dec_int(arg2);
      wrmi(ui,sect)
    END;
    wrmic_fun:BEGIN mic:=oct_int(arg1); wrmic(mic,barg2,barg(arg3)) END;
    wrob_fun:BEGIN oct_to_bits(arg1,dat,36); wrob(dat,barg2) END;
    wrport_fun:PORT[oct_int(arg1)]:=oct_int(arg2);
    wrreg_fun:BEGIN oct_to_bits(arg2,dat,36); wrreg(reg(arg1),dat) END;
    wrum_fun:BEGIN
      first:=TRUE; mi(arg2,dat);
      IF arg3='' THEN sect:=sect_ABC ELSE sect:=dec_int(arg3);
      wrum(oct_int(arg1),dat,sect,first)
    END;
    ELSE ret('?code"'+code+'"')
  END;
  ret('OK')
END;

VAR {frontend variables -- accessed only by frontend}

{cmd_tab constant table}
```

```
{screen display stuff}
{dpy_tab constant table}
current_fmt:dpy_fmt_typ;
holes, {if true then flag micro instruction hole fields}
dd_ok, {if true then ok to do a "dd" display}
quiet:BOOLEAN; {if true then messages and registers not displayed}

{macro processor stuff}
macro_count:0..number_macros; {number of macros defined so far}
macro_ok:BOOLEAN; {if true then expand macros in function ask}
macro_tab:ARRAY[1..number_macros] OF RECORD {table of macro definitions}
  name:STRING[8]; body:str_typ
END;

{command file stuff}
cmdfil:TEXT; {current command file}
cmdfil_name:str_typ; {name of command file, '' if none}
cmdfil_arg1,cmdfil_arg2:str_typ; {last command file arguments}

{idle display stuff}
idle_count:BYTE; {count of idle seconds}
idle_codes, {current items to idle display}
old_idle_codes:idle_typ; {copy of previous idle_codes}
updating, {if true then inside update procedure -- prevents recursion}
update_ok, {if true then ok to do idle updates}
abort_ok:BOOLEAN; {if true then abort if key pressed during update}
init_dma,init_amem:str_typ; {initial address values from CONSOLE.INI}

{remote frontend processor}
password_count:BYTE; {counts invalid password attempts}
password, {remote password, '' if none}
request:str_typ; {current remote request buffer}
remote_ok:BOOLEAN; {if true then remote access is allowed}

{random frontend variables}
fast_mode, {if true then default micro load is fast}
aborted, {set true when abort called}
quit, {if true then command loop will exit}
simulate, {if true then backend functions are interactively simulated}
dialing, {if true then modem connections will dial phone number}
locked, {if true then "*" commands not allowed}
running:BOOLEAN; {if true then console thinks system is running}
old_date,old_time, {previous date and time -- used to sense update time}
old_key,old_arg1,old_arg2:str_typ; {previous command -- for repeat}
boot, {boot address if monitor loaded, '' otherwise}
banner, {console identification banner}
last_msg, {last message displayed -- so won't be re-displayed}
command_name:str_typ; {name of current command}
scroll_line:ARRAY [1..4] OF str_typ; {command area scroll buffer}
talk_line:talk_typ; {current talk screen}
ugo_diag,ugo_lit,ugo_jmem,ugo_amem,ugo_mem,ugo_cty:str_typ; {entry points}

FUNCTION do_dpy(item:dpy_typ; dat:str_typ; force_dpy:BOOLEAN):str_typ;
{display an item, return displayed value}

PROCEDURE dat_dpy(x,y:BYTE; s:str_typ; dim:BOOLEAN);
{display data in data area, bright or dim}
```

```
BEGIN dpy_str(max(1,min(x,79)),4+max(1,min(y,12))-1,s,dat_color[dim]) END;

PROCEDURE clear_dat_dpy(y,n:BYTE);
{clear the data display area}
VAR i,yy,nn:BYTE;
BEGIN
  yy:=max(1,min(y,12)); nn:=min(n,12-yy+1);
  FOR i:=4+yy-1 TO 4+yy-1+nn-1 DO BEGIN GOTOXY(1,i); CLREOL END;
  GOTOXY(1,4+yy-1)
END;

PROCEDURE dpy_value(i:dpy_typ);
{display item from display table}
VAR value_bright:BOOLEAN; value_x_pos:BYTE;
BEGIN
  WITH dpy_tab[i] DO IF value_siz>0 THEN BEGIN
    IF i IN [line1_dpy..line12_dpy] THEN clear_dat_dpy(y_pos,1);
    value_x_pos:=x_pos+LENGTH(desc);
    CASE i OF
      mic_dpy:value_bright:=idle_mic IN idle_codes;
      obus_dpy,eobus_dpy:value_bright:=idle_obus IN idle_codes;
      dmaad_dpy,dma_dpy:value_bright:=idle_dma IN idle_codes;
      ubrk_dpy,mbrk_dpy:value_bright:=value<>fit('OFF',value_siz,left_fit);
      ELSE value_bright:=FALSE;
    END;
    IF i IN [amem_dpy,obus_dpy,eobus_dpy,ir_dpy,mem_dpy,ma_dpy,
      ac_dpy,ar_dpy,q_dpy,pc_dpy,m_dpy,dma_dpy,lit_dpy]
    THEN dat_dpy(value_x_pos,y_pos,oct_word(value),NOT value_bright)
    ELSE dat_dpy(value_x_pos,y_pos,value,NOT value_bright);
    IF i IN [adst_dpy,alu1_dpy,ldar_dpy,jcond_dpy,lit_dpy,mapf_dpy,spc_dpy,
      mask_dpy,rot_dpy,ifq_dpy,dfq_dpy]
    THEN BEGIN
      IF holes THEN BEGIN TEXTCOLOR(hole_color); WRITE('*') END
      ELSE WRITE(' ')
    END;
  END
END;

PROCEDURE format(new_fmt:dpy_fmt_typ);
{format the display area}

PROCEDURE format_line;
{line format}
VAR i:dpy_typ; BEGIN FOR i:=line1_dpy TO line12_dpy DO dpy_value(i) END;

PROCEDURE format_reg(code:BYTE);
{register format}

PROCEDURE format(item:dpy_typ);
{set up format for register item}
BEGIN
  WITH dpy_tab[item] DO dat_dpy(x_pos,y_pos,desc,FALSE); dpy_value(item)
END;

VAR i:dpy_typ;
BEGIN {format_reg}
  CASE code OF
```

```
    1:FOR i:=load_dpy TO dfq_dpy DO format(i);
    2:BEGIN
      clear_dat_dpy(dpy_tab[mapf_dpy].y_pos,1);
      FOR i:=mapf_dpy TO mask_dpy DO format(i)
    END;
    3:BEGIN
      clear_dat_dpy(dpy_tab[lit_dpy].y_pos,1);
      format(lit_dpy)
    END
  END
END;

BEGIN {format}
  IF new_fmt<>current_fmt THEN BEGIN
    clear_dat_dpy(1,12); current_fmt:=new_fmt;
    CASE new_fmt OF line_fmt:format_line; reg_fmt:format_reg(1) END
  END;
  IF current_fmt=reg_fmt THEN BEGIN
    IF dpy_tab[d_dpy].value<>dpy_tab[dd_dpy].value THEN BEGIN
      dpy_tab[dd_dpy].value:=dpy_tab[d_dpy].value;
      IF dpy_tab[d_dpy].value='32' THEN format_reg(3) ELSE format_reg(2)
    END
  END
END;

PROCEDURE dpy_flds;
{display fields of micro instruction}
VAR mi:bits_typ; f:fld_typ;

PROCEDURE dpy_field(f:fld_typ);
{display a micro instruction field}
VAR bb:bits_typ; x:BYTE;
BEGIN
  WITH fld_tab[f],dpy_tab[dpy] DO BEGIN
    get_bits(mi,bb,bit_zero,bits_siz);
    IF f=adst_fld THEN IF k1 THEN BEGIN
      x:=bb.bit[0]; bb.bit[0]:=bb.bit[2]; bb.bit[2]:=x
    END;
    value:=oct_digits(bits_to_oct(bb,bb.siz),value_siz);
    dpy_value(dpy)
  END
END;

BEGIN {dpy_flds}
  oct_to_bits(dpy_tab[mi_dpy].value,mi,88);
  FOR f:=aeal_fld TO jcond_fld DO dpy_field(f);
  FOR f:=dest_fld TO dfq_fld DO dpy_field(f);
  IF dpy_tab[d_dpy].value='32' THEN dpy_field(lit_fld)
  ELSE FOR f:=mapf_fld TO mask_fld DO dpy_field(f);
  format(reg_fmt)
END;

VAR old_value:str_typ; old_fmt:dpy_fmt_typ; i:dpy_typ;
BEGIN {do_dpy}
  IF item=dpy_init THEN BEGIN
    FOR i:=line1_dpy TO dpy_init DO dpy_tab[i].value:='';
    CLRSCR; current_fmt:=screen_fmt
```

```
  END
  ELSE IF NOT quiet THEN BEGIN
    old_fmt:=current_fmt;
    old_value:=dpy_tab[item].value;
    WITH dpy_tab[item] DO
    IF item IN [line1_dpy..line12_dpy] THEN BEGIN
      value:=dat; format(line_fmt)
    END
    ELSE IF item IN [load_dpy..remote_dpy,lc_dpy..br_dpy] THEN BEGIN
      value:=fit(dat,value_siz,left_fit); format(reg_fmt)
    END
    ELSE IF item IN [mbrk_dpy,ubrk_dpy] THEN BEGIN
      IF (dat='OFF') OR (dat='*HIT*')
      THEN value:=fit(dat,value_siz,left_fit)
      ELSE value:=oct_digits(dat,value_siz);
      format(reg_fmt)
    END
    ELSE BEGIN value:=oct_digits(dat,value_siz); format(reg_fmt) END;
    IF current_fmt=old_fmt THEN BEGIN
      IF force_dpy OR (old_value<>dpy_tab[item].value) THEN BEGIN
        dpy_value(item);
        IF item=mic_dpy THEN BEGIN
          WITH dpy_tab[ubrk_dpy] DO
          IF value=fit('*HIT*',value_siz,left_fit) THEN BEGIN
            value:=fit('OFF',value_siz,left_fit); dpy_value(ubrk_dpy)
          END;
          WITH dpy_tab[mbrk_dpy] DO
          IF value=fit('*HIT*',value_siz,left_fit) THEN BEGIN
            value:=fit('OFF',value_siz,left_fit); dpy_value(mbrk_dpy)
          END
        END
      END
    END;
    {flag holes in mi if mic is less than 40}
    IF item=mic_dpy THEN holes:=oct_int(dat)<32;
    IF item=mi_dpy THEN dpy_flds
  END;
  do_dpy:=dpy_tab[item].value
END;

FUNCTION dpyf(i:dpy_typ; d:str_typ):str_typ;
{display an item, return displayed value}
BEGIN dpyf:=do_dpy(i,d,TRUE) END;

PROCEDURE dpy(i:dpy_typ; d:str_typ);
{display an item}
BEGIN d:=dpyf(i,d) END;

PROCEDURE dpy_quiet(i:dpy_typ; d:str_typ);
{display an item -- update screen only if value changed}
BEGIN d:=do_dpy(i,d,FALSE) END;

PROCEDURE dpy_line(n:BYTE; l:str_typ);
{display data line n-1 -- may reformat data area to line format}
BEGIN dpy_quiet(dpy_typ(ORD(line1_dpy)+(n MOD 12)),l) END;

PROCEDURE line_screen;
```

```
LISTING: CONSOLE.TAB  01-JAN-80  00:00:00

{clear and set up line display in data area}
VAR i:1..12; BEGIN FOR i:=1 TO 12 DO dpy_line(i,'') END;

PROCEDURE register_screen;
{set up register display in data area}
VAR old_quiet:BOOLEAN;
BEGIN
  old_quiet:=quiet; quiet:=FALSE;
  line_screen; dpy_tab[dd_dpy].value:=''; dpy(run_dpy,dpy_tab[run_dpy].value)
  quiet:=old_quiet
END;

PROCEDURE dpy_default(VAR a:str_typ; i:dpy_typ);
{get default value from display table}
BEGIN default(a,dpy_tab[i].value) END;

FUNCTION update_time:BOOLEAN;
{return true if time for update -- every second}
BEGIN
  IF time(TRUE)<>old_time THEN BEGIN
    idle_count:=idle_count+1; old_time:=time(TRUE);
    IF current_fmt<>screen_fmt THEN BEGIN
      dpy_str(69,2,old_time,time_color);
      IF time(FALSE)<>old_date THEN BEGIN
        IF old_date<>'' THEN log(time(FALSE));
        old_date:=time(FALSE); dpy_str(3,2,old_date,time_color)
      END
    END;
    update_time:=TRUE
  END
  ELSE update_time:=FALSE
END;

PROCEDURE dpy_remote(m:str_typ);
{display remote state message}
BEGIN IF current_fmt=reg_fmt THEN dpy_quiet(remote_dpy,m) END;

PROCEDURE remote_frontend;
{cycle for remote frontend -- build request then call backend}
VAR result:str_typ; i:BYTE; c:CHAR; done:BOOLEAN;
BEGIN
  result:='';
  IF connected(modem) THEN BEGIN
    CASE remote OF
      hello: BEGIN {first state when connected}
        log('REMOTE ON: '+time(FALSE));
        dpy_remote('Hello'); remote:=entry; DELAY(2000); password_count:=0;
        result:=#10+#13+banner+#10+#13+'Type RETURN. '+#10+#13
      END;
      entry:BEGIN {connection is established}
        dpy_remote('Entry');
        IF receive(modem,c) THEN IF c=#13 THEN BEGIN
          request:='';
          IF password='' THEN BEGIN remote:=active; result:='Ok. '+#3 END
          ELSE BEGIN remote:=login; result:='Password: ' END
        END
      END;
```

```
    login: BEGIN {check password}
      dpy_remote('Login'); done:=FALSE;
      WHILE receive(modem,c) AND (remote=login) DO BEGIN
        IF c=#13 THEN BEGIN
          password_count:=password_count+1; done:=TRUE;
          IF uppercase(request)=password THEN BEGIN
            remote:=active; result:='Ok.'+#3
          END
          ELSE BEGIN
            remote:=entry; result:='?Invalid password.'+#10+#13
          END;
          request:=''
        END
        ELSE BEGIN
          IF (c>=' ') AND (LENGTH(request)<max_str_siz)
          THEN request:=request+c;
        END
      END
    END;
    remote_cty:BEGIN {do cty data transferr -- break on #0 #1}
      dpy_remote('Cty');
      REPEAT
        c:=#0;
        IF receive(cty,c) THEN send(modem,c);
        IF receive(modem,c) THEN BEGIN
          IF c=#0 THEN BEGIN
            REPEAT UNTIL receive(modem,c);
            IF c=#1 THEN BEGIN remote:=active; send(modem,#2) END
            ELSE send(cty,c)
          END
          ELSE send(cty,c)
        END
      UNTIL (c=#0) OR NOT connected(modem)
    END;
    active:BEGIN {do function requests}
      dpy_remote('Active');
      WHILE receive(modem,c) DO BEGIN
        IF c=#3 THEN request:=''
        ELSE IF c=#13 THEN BEGIN
          result:=backend(request);
          debug('REMOTE:('+request+')=('+result+')');
          request:=''; result:=result+#10+#13
        END
        ELSE BEGIN
          result:=#0;
          IF (c>=' ') AND (LENGTH(request)<max_str_siz)
          THEN request:=request+c
        END
      END
    END
  END;
  {send any result}
  IF NOT remote_ok THEN result:='?No access.';
  FOR i:=1 TO LENGTH(result) DO send(modem,result[i]);
  IF (password_count>3) OR NOT remote_ok THEN connect(modem,FALSE)
END
ELSE IF remote=we_are THEN dpy_remote('Off')
```

```
  ELSE BEGIN {modem not connected, clean up}
    IF remote<>hello THEN BEGIN
      log('REMOTE OFF: '+time(FALSE));
      connect(modem,FALSE); remote:=hello
    END;
    dpy_remote(switch(remote_ok,'Inactive','Off'))
  END
END;

PROCEDURE msg(m:str_typ);
{display a message}
BEGIN
  IF debugging OR NOT quiet THEN BEGIN
    m:=command_name+': '+m;
    IF m<>last_msg THEN BEGIN
      last_msg:=m; dpy_str(3,17,fit(m,76,left_fit),msg_color)
    END
  END
END;

PROCEDURE scroll(l:str_typ; to_next:BOOLEAN);
{scroll line into command area, scroll up if to_next}
VAR i:1..4;
BEGIN
  IF to_next THEN BEGIN
    dpy_str(1,19,'',scroll_color); scroll_line[4]:=l;
    FOR i:=1 TO 3 DO BEGIN
      CLREOL; scroll_line[i]:=scroll_line[i+1]; WRITELN(scroll_line[i])
    END;
    CLREOL {clear ask line}
  END
  ELSE BEGIN
    scroll_line[3]:=l; dpy_str(1,21,scroll_line[3],scroll_color)
  END
END;

PROCEDURE error(m:str_typ);
{display error message on command line}
BEGIN
  quiet:=FALSE;
  IF cmdfil_name<>'' THEN BEGIN CLOSE(cmdfil); cmdfil_name:='' END;
  scroll(scroll_line[4-1]+' > '+m,FALSE)
END;

PROCEDURE update(idle_ok:BOOLEAN); FORWARD; {see below}

PROCEDURE print_screen;
{print the display screen}
BEGIN INLINE($55/$CD/$05/$5D) END;

PROCEDURE wait(p:str_typ);
{display prompt and wait for keypressed -- do idle updates and remote cycles}
BEGIN
  dpy_str(1,22,p,ask_color);
  REPEAT
    {do not let keypressed signal an abort here}
    abort_ok:=FALSE; update(FALSE); abort_ok:=TRUE;
```

```
    {cycle the remote frontend}
    remote_frontend;
  UNTIL KEYPRESSED;
  dpy_str(1,22,p,ask_color) {put cursor back on command line}
END;

FUNCTION ask(p,d:str_typ):str_typ;
{interactive input request -- does macro expansion}
VAR r:str_typ; {the response}

FUNCTION expand_macros:BOOLEAN;
{expand any macro names found, return false if error}
VAR err,old_quiet:BOOLEAN; s,m:str_typ; i:1..number_macros;
BEGIN
  expand_macros:=TRUE;
  {display comment lines}
  IF POS(';',r)=1 THEN BEGIN
    old_quiet:=quiet; quiet:=FALSE; msg(r); quiet:=old_quiet;
    expand_macros:=FALSE
  END
  ELSE IF POS('@',r)=1 THEN BEGIN print_screen; expand_macros:=FALSE END
  {expand macros}
  ELSE IF macro_ok THEN BEGIN
    s:=uppercase(r); r:=''; err:=FALSE;
    WHILE (s<>'') AND NOT err DO BEGIN
      m:=parse(s);
      {replace "#1" and "#2" by command file args}
      IF m='#1' THEN m:=cmdfil_arg1 ELSE IF m='#2' THEN m:=cmdfil_arg2
      {stop expanding if "?" encountered}
      ELSE IF m[1]='?' THEN BEGIN DELETE(m,1,1); m:='? '+m+' '+s; s:='' END
      {otherwise, look up name in macro table}
      ELSE FOR i:=1 TO macro_count DO WITH macro_tab[i] DO
        IF name=m THEN m:=body;
      {append expansion if there is enough room else error}
      err:=(LENGTH(m)+LENGTH(r)+1)>max_str_siz;
      IF err THEN error('Macro error.') ELSE r:=r+' '+m
    END;
    debug('MACRO: '+r); expand_macros:=NOT err
  END;
END;

PROCEDURE read_keyboard;
{take input from keyboard}
BEGIN wait(p); READLN(r) END;

PROCEDURE read_file;
{take input from command file}
BEGIN
  IF EOF(cmdfil) THEN BEGIN
    CLOSE(cmdfil); cmdfil_name:=''; quiet:=FALSE; read_keyboard
  END
  ELSE BEGIN update(FALSE); READLN(cmdfil,r) END
END;

BEGIN {ask}
  {incorporate default response into prompt}
  p:=p+switch(d<>'',' ['+d+']','')+' ';
```

```
Listing: CONSOLE.PAS  01-JAN-80  00:00:00

  {solicit response, process any macros}
  REPEAT
    IF cmdfil_name='' THEN read_keyboard ELSE read_file;
    IF r='' THEN r:=d;
    IF debugging OR NOT quiet THEN scroll(p+r,TRUE);
  UNTIL expand_macros;
  {return the processed response}
  ask:=r
END;

FUNCTION askyn(q:str_typ):BOOLEAN;
{ask yes/no question, return true if answered yes}
VAR c:CHAR;
BEGIN
  q:=q+' '; {it looks nicer this way}
  REPEAT
    wait(q); {wait for key to be pressed}
    {read the key immediatly -- do not wait for carriage return}
    READ(KBD,c); c:=UPCASE(c); scroll(q+c,TRUE);
    IF c='@' THEN print_screen
    ELSE IF NOT (c IN ['Y','N']) THEN error('Options: YES, NO')
  UNTIL c IN ['Y','N'];
  askyn:=c='Y'
END;

FUNCTION continue:BOOLEAN;
{ask if continue, return true if yes}
BEGIN continue:=askyn('Continue?') END;

PROCEDURE abort(m:str_typ);
{display error message and set aborted flag}
BEGIN IF NOT aborted THEN BEGIN error(m); aborted:=NOT continue END END;

FUNCTION callf(f:cmd_typ; a:str_typ):str_typ;
{call backend function, return result}

FUNCTION remote_backend(q:str_typ):str_typ;
{make a remote backend request via the modem}
VAR r:str_typ; c:CHAR; i:1..max_str_siz; t:BYTE;
BEGIN
  {send request string ended with linefeed and carriage return}
  FOR i:=1 TO LENGTH(q) DO send(modem,q[i]);
  send(modem,#10); send(modem,#13);
  {loop to receive result, may time out, checks for disconnected modem}
  t:=0; r:='';
  REPEAT
    IF update_time THEN t:=t+1;
    IF receive(modem,c) THEN
      IF (c>=' ') AND (LENGTH(r)<max_str_siz) THEN r:=r+c
  UNTIL (c=#13) OR (t>5) OR NOT connected(modem);
  {return error message if abnormal situation}
  IF NOT connected(modem) THEN BEGIN dpy_remote('Off'); r:='?Remote lost. ' EN
  ELSE BEGIN dpy_remote('Active'); IF t>5 THEN r:='?Remote hung. ' END;
  remote_backend:=r
END;

VAR cmd,r:str_typ;
```

```
BEGIN {call}
  {build up the backend function request string}
  cmd:=unparse(int_dec(ORD(f)),a,'');
  {if not simulating, then call the backend, maybe a remote backend}
  IF NOT simulate THEN BEGIN
    IF remote=we_are THEN r:=remote_backend(cmd) ELSE r:=backend(cmd)
  END;
  {build up a legible version of the request}
  cmd:=cmd_tab[f].name+'('+a+')';
  {maybe simulate the call}
  IF simulate THEN BEGIN
    bad_digit:=FALSE; digit_overflow:=FALSE; r:=ask(cmd+': ','')
  END;
  debug('CALL: '+cmd+'="'+r+'"');
  {check for any error message -- abort if so}
  IF r<>'' THEN IF r[1]='?' THEN BEGIN
    abort('CALL: '+cmd_tab[f].name+'('+a+')='+r); r:=''
  END;
  callf:=r
END;

PROCEDURE call(f:cmd_typ; a:str_typ);
{call a backend function -- a callf with no result returned}
BEGIN a:=callf(f,a) END;

FUNCTION statusf(check_conds:BOOLEAN):str_typ;
{return system status, maybe check conditions}

PROCEDURE check_for_break;
{check for micro/macro break, turn off micro/macro histories}
BEGIN
  {make sure system is stopped, turn of history recording}
  call(run_fun,'FALSE'); call(enahst_fun,'FALSE'); locked:=FALSE;
  {get the mic and mi where system stopped}
  dpy_quiet(mic_dpy,callf(rdmic_fun,'')); dpy_quiet(mi_dpy,callf(rdmi_fun,''))
  {check if mic/pc matches micro/macro break address, flag any hit}
  WITH dpy_tab[ubrk_dpy] DO
  IF value<>fit('OFF',value_siz,left_fit) THEN BEGIN
    IF value=dpy_tab[mic_dpy].value THEN BEGIN
      dpy_quiet(ubrk_dpy,'*HIT*'); call(break_fun,'TRUE OFF')
    END
  END;
  WITH dpy_tab[mbrk_dpy] DO
  IF value<>fit('OFF',value_siz,left_fit) THEN BEGIN
    dpy_quiet(pc_dpy,callf(rdreg_fun,'PC'));
    IF value=COPY(dpy_tab[pc_dpy].value,7,6) THEN BEGIN
      dpy_quiet(mbrk_dpy,'*HIT*'); call(break_fun,'FALSE OFF')
    END
  END
END;

VAR s:str_typ; was_running:BOOLEAN;
BEGIN {statusf}
  {collect system status from backend}
  s:=callf(rdst_fun,'');
  {backend and frontend versions must match}
  IF parse(s)<>version THEN abort('Wrong version.')
```

```
  ELSE IF NOT aborted THEN BEGIN
    {parse and display running status, check for break if just stopped}
    was_running:=running; running:=parse(s)='TRUE';
    IF was_running AND NOT running THEN check_for_break;
    dpy_quiet(run_dpy,switch(running,'Running','Stopped'));
    {parse and display status conditions}
    IF check_conds THEN BEGIN
      dpy_quiet(lc_dpy,switch(parse(s)='TRUE','ON ','OFF'));
      dpy_quiet(tc_dpy,switch(parse(s)='TRUE','ON ','OFF'));
      dpy_quiet(br_dpy,switch(parse(s)='TRUE','Branching',''))
    END
  END;
  {build a neat status report message}
  statusf:='System is '+
    switch(locked,'locked','unlocked')+' and '+
    switch(running,'running','stopped')+'.'
END;

PROCEDURE status(check_conds:BOOLEAN);
{check system status -- a statusf which does not return anything}
VAR s:str_typ; BEGIN s:=statusf(check_conds) END;

PROCEDURE run(yes:BOOLEAN);
{set run/stop state of system}
BEGIN
  locked:=yes; {automatic lock/unlock}
  IF yes<>running THEN BEGIN
    {change run state of system, turn on histories if about to run}
    IF yes THEN call(enahst_fun,'TRUE');
    call(run_fun,switch(yes,'TRUE','FALSE'))
  END;
  {make sure status routine notices change in state}
  running:=TRUE; status(FALSE)
END;

PROCEDURE idle(do_refresh:BOOLEAN);
{idle display, forces value display if do_refresh}
VAR old_debugging,old_quiet:BOOLEAN;
BEGIN
  {debugging/quiet not applicable to idle display}
  old_debugging:=debugging; old_quiet:=quiet;
  debugging:=FALSE; quiet:=FALSE;
  {check on system status, report any changes}
  status(FALSE);
  {maybe refresh all idle items -- will force bright/dim change}
  IF do_refresh THEN BEGIN
    dpy(mic_dpy,dpy_tab[mic_dpy].value);
    dpy(obus_dpy,dpy_tab[obus_dpy].value);
    dpy(eobus_dpy,dpy_tab[eobus_dpy].value);
    dpy(dmaad_dpy,dpy_tab[dmaad_dpy].value);
    dpy(dma_dpy,dpy_tab[dma_dpy].value)
  END;
  {collect and display idle items if changed -- note use of dpy_quiet}
  IF idle_mic IN idle_codes THEN dpy_quiet(mic_dpy,callf(rdmic_fun,''));
  IF idle_obus IN idle_codes THEN BEGIN
    dpy_quiet(obus_dpy,callf(rdob_fun,'TRUE FALSE'));
    dpy_quiet(eobus_dpy,callf(rdob_fun,'FALSE TRUE'))
```

```
  END;
  IF idle_dma IN idle_codes
  THEN dpy_quiet(dma_dpy,callf(rddma_fun,dpy_tab[dmaad_dpy].value));
  debugging:=old_debugging; quiet:=old_quiet
END;

PROCEDURE update(idle_ok:BOOLEAN);
{idle update -- does idle display every second, maybe abort if keypressed}
VAR c:CHAR;
BEGIN
  {can not re-enter this routine}
  IF NOT updating THEN BEGIN
    updating:=TRUE;
    {ok for idle update if time or forced}
    IF update_time THEN IF update_ok THEN idle_ok:=TRUE;
    {do not want to do idle update if not register format}
    IF idle_ok AND (current_fmt=reg_fmt) THEN idle(FALSE);
    {maybe check for operator abort}
    IF abort_ok AND KEYPRESSED THEN BEGIN READ(KBD,c); abort('Abort.') END;
    updating:=FALSE
  END
END;

{$I CONSOLE2.PAS the frontend commands}
{CONSOLE2.PAS -- source file for Augment Engine Console (4 of 5) zw}

{frontend commands}

PROCEDURE command(args:str_typ);
{do frontend command}
VAR repeat_ok:BOOLEAN;

FUNCTION flag(a:str_typ; VAR f:BOOLEAN):BOOLEAN;
{parse ON or OFF, set flag and return true if success}
BEGIN
  flag:=TRUE;
  IF eq(a,'ON',2) THEN f:=TRUE ELSE IF eq(a,'OFF',2) THEN f:=FALSE
  ELSE flag:=FALSE
END;

PROCEDURE idles(i:idle_typ);
{save idle codes, set new value, refresh idle display}
BEGIN old_idle_codes:=idle_codes; idle_codes:=i; idle(TRUE) END;

FUNCTION unlocked:BOOLEAN;
{return true if unlocked else abort}
BEGIN unlocked:=NOT locked; IF locked THEN abort(statusf(FALSE)) END;

OVERLAY PROCEDURE start(a:str_typ; micro,and_run:BOOLEAN);
{micro/macro start, run}
BEGIN
  IF micro THEN BEGIN
    IF eq(a,'DIAG',1) THEN a:=ugo_diag
    ELSE IF eq(a,'LIT',1) THEN a:=ugo_lit
    ELSE IF eq(a,'JMEM',1) THEN a:=ugo_jmem
    ELSE IF eq(a,'AMEM',1) THEN a:=ugo_amem
    ELSE IF eq(a,'MEM',1) THEN a:=ugo_mem
```

```
    ELSE IF eq(a,'CTY',1) THEN a:=ugo_cty;
    call(wrmic_fun,a+',T,T'); IF and_run THEN run(TRUE)
  END
  ELSE BEGIN
    IF eq(a,'BOOT',1) THEN BEGIN
      IF boot<>'' THEN a:=boot ELSE abort('No boot address.')
    END;
    call(wrreg_fun,'PC,'+a); call(wrmic_fun,'4040,T,F');
    call(step_fun,'T,2'); IF and_run THEN run(TRUE)
  END
END;

OVERLAY PROCEDURE wait(a2:str_typ;timeout:INTEGER);
{wait for micro address(s) to be hit}
VAR a1:str_typ; hit,x:BOOLEAN;
BEGIN
  idles([idle_mic]); idle_count:=0; a1:=parse(a2);
  a1:=oct_digits(a1,5); IF a2<>'' THEN a2:=oct_digits(a2,5);
  IF bad_digit OR digit_overflow THEN abort('Bad address value.')
  ELSE BEGIN
    msg('Waiting for '+a1+switch(a2<>'',',',+a2,''));
    IF NOT update_ok THEN abort('Updating OFF.')
    ELSE REPEAT
      IF (cmdfil_name<>'') AND (idle_count>timeout) THEN abort('Timed out.');
      update(FALSE);
      WITH dpy_tab[mic_dpy] DO hit:=(value=a1) OR (value=a2)
    UNTIL aborted OR hit;
    IF aborted THEN msg('Aborted.') ELSE msg('Hit.');
    x:=aborted; aborted:=FALSE; run(FALSE); aborted:=x
  END;
  idle_codes:=old_idle_codes
END;

OVERLAY PROCEDURE new_screen(b:str_typ);
{paint new screen}

PROCEDURE box(x,y,h,l:BYTE);
{draw box}
VAR i:BYTE; s:str_typ;
BEGIN
  s:=#218; FOR i:=1 TO l DO s:=s+#196; s:=s+#191+#8+#10;
  FOR i:=1 TO h DO s:=s+#179+#8+#10;
  dpy_str(x,y,s,box_color);
  s:=''; FOR i:=1 TO h DO s:=s+#179+#8+#10;
  s:=s+#192; FOR i:=1 TO l DO s:=s+#196; s:=s+#217;
  dpy_str(x,y+1,s,box_color)
END;

PROCEDURE vertical(x,y,h:BYTE);
{draw vertical branch}
VAR i:BYTE; s:str_typ;
BEGIN
  s:=#194+#8+#10; FOR i:=1 TO h DO s:=s+#179+#8+#10; s:=s+#193;
  dpy_str(x,y,s,box_color)
END;

VAR i:1..4;
```

```
BEGIN {new_screen}
  CLRSCR; box(1,1,1,77); vertical(13,1,1); vertical(66,1,1);
  banner:=b; dpy_str(14,2,fit(banner,51,center_fit),banner_color);
  box(1,16,1,77); last_msg:='';
  FOR i:=1 TO 4 DO scroll_line[i]:='';
  old_date:=''; old_time:=''; IF update_time THEN register_screen
END;

OVERLAY PROCEDURE connect_modem(yes:BOOLEAN);
{connect modem port, dial phone number}

PROCEDURE dial(p:BYTE; n:str_typ);
{dial phone number on port}
VAR i,lcr,thr,mcr,lsr:BYTE;
BEGIN
  IF p=com1 THEN BEGIN lcr:=$3FB; mcr:=$3FC; END
  ELSE BEGIN lcr:=$2FB; mcr:=$2FC; END;
  PORT[mcr]:=PORT[mcr] AND $01; PORT[lcr]:=PORT[lcr] AND $07;
  n:='AT M1 V1 S6=9 DT'+n+' ; 0'; FOR i:=1 TO LENGTH(n) DO send(p,n[i]);
  send(p,^M)
END;

VAR c1,c2:CHAR; n:str_typ;
BEGIN {connect_modem}
  IF NOT yes THEN connect(modem,FALSE)
  ELSE IF NOT connected(modem) THEN BEGIN
    IF dialing THEN BEGIN alert('Phone number: '); READLN(n); dial(modem,n) END
    connect(modem,TRUE);
    IF NOT connected(modem) THEN aborted:=TRUE
    ELSE IF remote=we_are THEN BEGIN
      alert('Modem connected, Use F1 to abort. '); WRITELN;
      REPEAT
        IF receive(modem,c1) THEN WRITE(c1)
        ELSE IF KEYPRESSED THEN BEGIN
          READ(KBD,c2);
          IF (c2=#27) AND KEYPRESSED THEN BEGIN
            READ(KBD,c2); aborted:=c2=#59; IF NOT aborted THEN send(modem,#27)
          END;
          IF NOT aborted THEN send(modem,c2)
        END
      UNTIL (c1=#3) OR aborted
    END
    ELSE BEGIN
      REPEAT alert('Type RETURN. '); REPEAT UNTIL KEYPRESSED; READ(KBD,c2)
      UNTIL c2=^M;
      remote:=active; request:=''; send(modem,#3)
    END
  END
  ELSE IF remote=we_are THEN BEGIN send(modem,#0); send(modem,#1) END
END;

OVERLAY PROCEDURE command_repeat(VAR key,arg1,arg2:str_typ);
{repeat previous command}

PROCEDURE args(new_arg1,new_arg2:str_typ);
{copy args from previous command}
BEGIN
```

```
  key:=old_key; arg1:=new_arg1; arg2:=new_arg2;
  scroll(prompt+' '+unparse(key,arg1,arg2),FALSE)
END;

BEGIN {command_repeat}
  IF cmdfil_name='' THEN BEGIN {no repeats in command file}
    IF old_key='SS' THEN args('','')
    ELSE IF old_key='DMRD' THEN args(incr_oct(old_arg1),old_arg2)
    ELSE IF old_key='DMWRT' THEN args(incr_oct(old_arg1),old_arg2)
    ELSE IF old_key='EX' THEN args(incr_oct(old_arg1),old_arg2)
    ELSE IF old_key='DE' THEN args(incr_oct(old_arg1),old_arg2)
    ELSE IF old_key='MMLOAD' THEN args(incr_oct(old_arg1),old_arg2)
    ELSE IF old_key='MMREAD' THEN args(incr_oct(old_arg1),'DISPLAY')
    ELSE IF old_key='CALL' THEN args(old_arg1,'')
  END
END;

OVERLAY PROCEDURE command_file(name,a1,a2:str_typ);
{open command file}
VAR i:BYTE; dot:BOOLEAN;
BEGIN
  dot:=FALSE; FOR i:=1 TO LENGTH(name) DO IF name[i]='.' THEN dot:=TRUE;
  IF NOT dot THEN name:=name+'.CMD';
  IF cmdfil_name<>'' THEN CLOSE(cmdfil);
  cmdfil_name:='';
  ASSIGN(cmdfil,name); {$I-} RESET(cmdfil); {$I+}
  IF IORESULT<>0 THEN abort('No file: '+name)
  ELSE BEGIN
    cmdfil_name:=name; cmdfil_arg1:=a1; cmdfil_arg2:=a2; quiet:=TRUE
  END
END;

OVERLAY FUNCTION command_ok(i:cmd_typ; VAR key,arg1,arg2:str_typ):BOOLEAN;
{if a given command is ok, match key, verify arguments}

PROCEDURE ask_for_args(prompt1,prompt2:str_typ);
{solicit arguments}
BEGIN
  IF (arg1='') AND (prompt1<>'') AND (prompt1<>'*')
  THEN arg1:=ask(prompt1,'');
  IF (arg2='') AND (prompt2<>'') AND (prompt2<>'*')
  THEN arg2:=ask(prompt2,'');
  IF ((arg1<>'') AND (prompt1='')) OR ((arg2<>'') AND (prompt2=''))
  THEN abort('Too many arguments')
END;

FUNCTION arg_prompt(desc:str_typ):str_typ;
{make argument prompt from description}
BEGIN
  IF desc='' THEN arg_prompt:=''
  ELSE IF (desc='*') OR (desc[1]='[') THEN arg_prompt:='*'
  ELSE arg_prompt:='Enter '+desc+':'
END;

PROCEDURE protect_system;
{abort if system is protected, running and/or locked}
BEGIN
```

```
  status(FALSE);
  IF unlocked THEN IF running THEN BEGIN
    IF askyn(statusf(FALSE)+'  Stop?') THEN run(FALSE)
    ELSE abort('Aborted.');
    status(FALSE)
  END
END;

BEGIN {command_ok}
  WITH cmd_tab[i] DO IF eq(key,name,eq_siz) THEN BEGIN
    command_name:=name; msg(cmd_desc);
    IF cmd_desc<>'' THEN IF cmd_desc[1]='*' THEN protect_system;
    IF NOT aborted THEN ask_for_args(
      arg_prompt(arg1_desc),arg_prompt(arg2_desc));
    command_ok:=NOT aborted
  END
  ELSE command_ok:=FALSE
END;

OVERLAY PROCEDURE idle(args:str_typ);
{set idle display}
VAR arg:str_typ;
BEGIN
  REPEAT
    arg:=parse(args);
    IF eq(arg,'OFF',1) THEN idles([])
    ELSE IF eq(arg,'MIC',1) THEN idles(idle_codes+[idle_mic])
    ELSE IF eq(arg,'OBUS',1) THEN idles(idle_codes+[idle_obus])
    ELSE IF eq(arg,'DMA',1) THEN idles(idle_codes+[idle_dma])
    ELSE IF eq(arg,'LOC',4) THEN BEGIN
      arg:=parse(args); IF arg<>'' THEN init_dma:=arg;
      dpy(dma_dpy,init_dma); idles(idle_codes+[idle_dma])
    END
    ELSE IF arg<>'' THEN abort('Options: OFF, MIC, OBUS, DMA')
  UNTIL aborted OR (arg='')
END;

OVERLAY PROCEDURE do_dd;
{display data from all registers}
BEGIN
  IF NOT quiet THEN BEGIN
    call(save_fun,'T,F');
    IF dd_ok THEN BEGIN
      dpy(m_dpy,callf(rdm_fun,dpy_tab[mad_dpy].value));
      dpy(ecc_dpy,callf(rdecc_fun,''));
      dpy(dma_dpy,callf(rddma_fun,dpy_tab[dmaad_dpy].value));
      dpy(pc_dpy,callf(rdreg_fun,'PC'));
      dpy(ma_dpy,callf(rdreg_fun,'MA'));
      dpy(ar_dpy,callf(rdreg_fun,'AR'));
      dpy(mem_dpy,callf(rdreg_fun,'MEM'));
      dpy(ir_dpy,callf(rdreg_fun,'IR'));
      dpy(q_dpy,callf(rdreg_fun,'Q'));
      dpy(lp_dpy,callf(rdreg_fun,'LP'));
      dpy(ac_dpy,callf(rdac_fun,dpy_tab[acad_dpy].value));
      dpy(jmemp_dpy,callf(rdreg_fun,'JMEMP'));
      dpy(jmem_dpy,callf(rdreg_fun,'JMEM'));
      dpy(dev_dpy,callf(rdreg_fun,'DEV'));
```

```
      dpy(amemp_dpy,callf(rdreg_fun,'AMEMP'));
      dpy(amem_dpy,callf(rdam_fun,dpy_tab[amemad_dpy].value+',F'))
    END;
    call(save_fun,'F,T');
    dpy(mic_dpy,callf(rdmic_fun,''));
    dpy(obus_dpy,callf(rdob_fun,'T,F'));
    dpy(eobus_dpy,callf(rdob_fun,'F,T'));
    dpy(mi_dpy,callf(rdmi_fun,''));
    msg(statusf(TRUE))
  END
END;

OVERLAY PROCEDURE type_file(name,match:str_typ);
{display contents of file, search for match}
VAR text_file:TEXT; line_num:BYTE; matched:BOOLEAN; line:str_typ;
BEGIN
  ASSIGN(text_file,name); {$I-} RESET(text_file); {$I+}
  IF IORESULT<>0 THEN abort('No file: '+name)
  ELSE BEGIN
    line_num:=0; matched:=FALSE;
    IF match<>'' THEN BEGIN
      msg('Searching '+name+' for '+match);
      WHILE NOT (aborted OR matched OR EOF(text_file)) DO BEGIN
        READLN(text_file,line); matched:=POS(match,line)>0; update(FALSE)
      END
    END;
    IF (match<>'') AND NOT matched THEN abort('No match: '+match)
    ELSE BEGIN
      msg('File: '+name); line_screen;
      WHILE NOT aborted AND (matched OR NOT EOF(text_file)) DO BEGIN
        IF NOT matched THEN READLN(text_file,line);
        matched:=FALSE; update(FALSE);
        IF aborted OR ((line_num>0) AND ((line_num MOD 12)=0))
        THEN aborted:=NOT continue;
        IF NOT aborted THEN BEGIN
          IF (line_num MOD 12)=0 THEN line_screen;
          dpy_line(line_num,line); line_num:=line_num+1
        END
      END;
      IF NOT aborted THEN aborted:=continue;
      register_screen
    END;
    CLOSE(text_file);
    aborted:=FALSE
  END
END;

OVERLAY FUNCTION help(first,last:cmd_typ; VAR key:str_typ):BOOLEAN;
{display documentation, return true if eplain}
VAR i:cmd_typ; j:1..number_macros; count:BYTE; done:BOOLEAN; line:str_typ;

PROCEDURE dpy_names;
{display command names}

PROCEDURE dpy_name(name:str_typ);
{display command name}
VAR selected:BOOLEAN;
```

```
BEGIN
  IF key='' THEN selected:=TRUE
  ELSE IF LENGTH(key)>LENGTH(name) THEN selected:=FALSE
  ELSE selected:=COPY(name,1,LENGTH(key))=key;
  IF selected THEN BEGIN
    IF (count MOD 8)=0 THEN BEGIN
      IF count > 0 THEN BEGIN
        dpy_line((count-1) DIV 8,line);
        IF (count MOD 96)=0 THEN BEGIN
          done:=NOT continue; IF NOT done THEN line_screen
        END
      END;
      line:='     '
    END;
    line:=line+name+blanks(9-LENGTH(name));
    count:=count+1
  END
END;

BEGIN {dpy_names}
  msg('Commands'+switch(key='','. ',':  '+key+'*'));
  line_screen; count:=0; line:=''; done:=FALSE;
  FOR i:=first TO last DO dpy_name(cmd_tab[i].name);
  FOR j:=1 TO macro_count DO dpy_name(macro_tab[j].name);
  IF NOT done THEN BEGIN dpy_line((count-1) DIV 8,line); done:=continue END;
  done:=TRUE; register_screen;
END;

PROCEDURE dpy_descs;
{display command descriptions}
VAR i:cmd_typ; j:1..number_macros;

PROCEDURE dpy_on_status_line(name,desc:str_typ);
{display description message}
BEGIN command_name:=name; msg(desc) END;

PROCEDURE dpy_on_line(name,desc:str_typ);
{display description line}
BEGIN
  IF (count>0) AND ((count MOD 12)=0) THEN BEGIN
    done:=NOT continue; IF NOT done THEN line_screen
  END;
  IF NOT done THEN dpy_line(count,name+' : '+desc);
  count:=count+1
END;

FUNCTION desc_args(a1,a2:str_typ):str_typ;
{describe arguments}

FUNCTION desc_arg(a:str_typ):str_typ;
{describe argument}
BEGIN
  IF a='*' THEN desc_arg:='[args]'
  ELSE IF a[1]='[' THEN desc_arg:=a
  ELSE desc_arg:='<'+a+'>'
END;
```

```
BEGIN {desc_args}
  IF a1='' THEN desc_args:=desc_arg('no args')
  ELSE IF (a2='') OR (a2='*') THEN desc_args:=desc_arg(a1)
  ELSE desc_args:=desc_arg(a1)+' '+desc_arg(a2)
END;

FUNCTION desc_eq(name:str_typ; siz:BYTE):str_typ;
{note command abbreviation}
BEGIN
  IF siz>=LENGTH(name) THEN desc_eq:=name
  ELSE desc_eq:=COPY(name,1,siz)+'/'+COPY(name,siz+1,8)
END;

BEGIN {dpy_descs}
  IF key='*' THEN BEGIN count:=0; line_screen END;
  msg('Macros.'); done:=FALSE;
  FOR j:=1 TO macro_count DO IF NOT done THEN WITH macro_tab[j] DO BEGIN
    IF key='*' THEN dpy_on_line(name,'"'+body+'"')
    ELSE IF key=name THEN BEGIN
      dpy_on_status_line(name,'"'+body+'"'); done:=TRUE
    END
  END;
  IF NOT done THEN msg('Commands.');
  FOR i:=first TO last DO IF NOT done THEN WITH cmd_tab[i] DO BEGIN
    IF key='*' THEN dpy_on_line(desc_eq(name,eq_siz),
      desc_args(arg1_desc,arg2_desc)+' -- '+cmd_desc)
    ELSE IF eq(key,name,eq_siz) THEN BEGIN
      dpy_on_status_line(desc_eq(name,eq_siz),
        desc_args(arg1_desc,arg2_desc)+' -- '+cmd_desc);
      key:=name; done:=TRUE
    END
  END;
  IF key='*' THEN BEGIN
    IF NOT done THEN done:=continue; done:=TRUE; register_screen
  END;
END;

BEGIN {help}
  IF key='' THEN dpy_names
  ELSE IF (LENGTH(key)>1) AND (key[LENGTH(key)]='*') THEN BEGIN
    DELETE(key,LENGTH(key),1); dpy_names
  END
  ELSE BEGIN
    dpy_descs;
    IF (key<>'*') AND done THEN BEGIN
      done:=NOT askyn('Explain?');
      IF NOT done THEN key:=key+switch(first<boot_cmd,'_FUN','_CMD')
    END
  END;
  help:=NOT done; repeat_ok:=FALSE
END;

PROCEDURE do_command(cmd_code:cmd_typ; args:str_typ);
{do frontend command}
VAR arg1,arg2:str_typ;

OVERLAY PROCEDURE do_clock(a:str_typ);
```

```
LISTING: CONSOLE.TEXT  01-JAN-80  00:00:00  (CONSOLE.TEXT)

{clock micro engine}
BEGIN
  default(a,'1');
  IF a<>'*' THEN call(step_fun,'F,'+a)
  ELSE BEGIN
    msg('infinite clock stepping...');
    REPEAT call(step_fun,'F,512'); update(FALSE) UNTIL aborted;
    aborted:=FALSE
  END
END;

OVERLAY PROCEDURE do_talk;
{communicate remote/local}

PROCEDURE talk_screen;
{display talk lines -- bottom up}
VAR i:1..12; BEGIN FOR i:=12 DOWNTO 1 DO dpy_line(i-1,talk_line[i]) END;

PROCEDURE talk(m:str_typ);
{display line on talk screen at bottom, scroll up}
VAR i:1..12;
BEGIN
  log(m);
  FOR i:=1 TO 11 DO talk_line[i]:=talk_line[i+1]; talk_line[12]:=m;
  talk_screen
END;

VAR m:str_typ; old_macro_ok:BOOLEAN;
BEGIN {do_talk}
  old_macro_ok:=macro_ok; macro_ok:=FALSE; talk_screen;
  REPEAT
    dpy_str(1,22,prompt,ask_color);
    WHILE NOT (KEYPRESSED OR (cmdfil_name<>'') OR aborted) DO BEGIN
      remote_frontend;
      IF update_time THEN BEGIN
        IF NOT connected(modem) THEN msg('Remote disconnected...')
        ELSE BEGIN
          msg('Talking to '+switch(remote=we_are,'local.','remote.'));
          m:=callf(talk_fun,switch(remote=we_are,'T','F'));
          IF m<>'' THEN talk(m)
        END
      END
    END;
    IF NOT aborted THEN BEGIN
      m:=ask(prompt,'');
      IF (m<>'') AND connected(modem) THEN BEGIN
        m:=switch(remote=we_are,'REM: ','LOC: ')+m; talk(m);
        IF dec_int(callf(talk_fun,switch(remote=we_are,'T','F')+',+m))>2
        THEN alert(switch(remote=we_are,'Local','Remote')+' not listening.')
        ELSE alert('')
      END
    END
  UNTIL (m='') OR NOT connected(modem) OR aborted;
  macro_ok:=old_macro_ok; register_screen
END;

OVERLAY PROCEDURE do_history;
```

```
LISTING: CONSOLE.PAS  01-JAN-80  00:00:00  CONSOLE.PAS;

{display micro/macro histories}
CONST
header='Time  uIndex uAddr uFlags       mIndex mAddr       mCycle  mFlags';
VAR old_hist,old_wait:str_typ;

FUNCTION micro_hist(o:INTEGER):str_typ;
{return micro history}
VAR i,b:str_typ; a,d:bits_typ;
BEGIN
  i:=' ['+oct_digits(callf(rdhsti_fun,'F,T'),4)+'] ';
  oct_to_bits(callf(rdhst_fun,'T'),d,36); {clocks index}
  b:=switch((0<>0) AND (d.bit[8]=0),' Branch','')+old_wait;
  micro_hist:=old_hist+fit(b,12,left_fit);
  IF o>0 THEN BEGIN
    IF d.bit[9]=0 THEN BEGIN
      get_bits(d,a,28,4); a.siz:=5; a.bit[4]:=d.bit[27];
      old_hist:=i+'TR-'+oct_digits(bits_to_oct(a,5),2)
    END
    ELSE BEGIN
      get_bits(d,a,12,14); old_hist:=i+oct_digits(bits_to_oct(a,14),5)
    END;
    old_wait:=switch(d.bit[10]=0,' Wait','')
  END
END;

FUNCTION macro_hist:str_typ;
{return macro history}
VAR a,c,d:bits_typ; i,h:str_typ;
BEGIN
  i:=' ['+oct_digits(callf(rdhsti_fun,'F,F'),4)+'] ';
  oct_to_bits(callf(rdhst_fun,'F'),d,40); {clocks address}
  get_bits(d,a,4,32); h:=i+oct_digits(bits_to_oct(a,32),11)+' ';
  get_bits(d,c,0,3); fit_bits(c,1,16,right_fit);
  CASE c.bit[0] OF
    0:h:=h+'Refresh'; 1:h:=h+'DF Req '; 2:h:=h+'DMA Rd ';
    3:h:=h+'DS Req '; 4:h:=h+'DMA Wr '; 5:h:=h+'IF Req ';
    6:h:=h+'JF Req '; 7:h:=h+'No Cy  '
  END;
  h:=h+switch(d.bit[38]>0,' Exec',' User');
  h:=h+switch(d.bit[3]>0,' Mapped',' Nomap ');
  h:=h+switch(d.bit[39]>0,' Wait','      ');
  macro_hist:=h
END;

FUNCTION continue(VAR o:str_typ; VAR ok:BOOLEAN):BOOLEAN;
{continue?, accept new offset}
VAR r,a:str_typ; c:(yes,no,num,err); i:INTEGER;
BEGIN
  REPEAT
    r:=ask('Continue?','YES'); a:=parse(r);
    IF r<>'' THEN c:=err
    ELSE IF eq(a,'YES',1) THEN c:=yes ELSE IF eq(a,'NO',1) THEN c:=no
    ELSE BEGIN i:=dec_int(a); IF NOT bad_digit THEN c:=num ELSE c:=err END;
    IF c=err THEN error('Options: YES, NO, offset')
  UNTIL c<>err;
  continue:=c=yes; ok:=c=num; IF ok THEN o:=a
END;
```

```
VAR f:TEXT; line_num:INTEGER; name,line:str_typ;

PROCEDURE dpy;
{display history line}
BEGIN
  IF name<>'' THEN BEGIN
    msg(int_dec(line_num)); {$I-} WRITELN(f,line); {$I+}
    IF IORESULT<>0 THEN abort('Disk full.')
  END
  ELSE BEGIN
    IF (line_num MOD 12)=0 THEN line_screen; dpy_line(line_num,line)
  END;
  line_num:=line_num+1
END;

VAR j,i:INTEGER; first,ok:BOOLEAN; o:str_typ;
BEGIN {do_history}
  o:=arg1; default(o,'10'); name:=arg2; i:=0; line_num:=0;
  IF name<>'' THEN BEGIN
    IF POS('.',name)<>0 THEN ASSIGN(f,name)
    ELSE REPEAT i:=i+1; ASSIGN(f,name+'.'+int_dec(i)); {$I-} RESET(f) {$I+}
    UNTIL IORESULT<>0;
    REWRITE(f); line:=time(FALSE)+' '+banner+' '+time(TRUE); dpy
  END;
  ok:=TRUE; first:=TRUE;
  REPEAT
    j:=dec_int(o);
    IF bad_digit OR (j<0) OR (j>1023) THEN abort('Options: 0..1023')
    ELSE BEGIN
      line_num:=0; old_hist:=''; old_wait:='';
      call(wrhsti_fun,int_oct(j)+',T,T'); line:=micro_hist(1);
      call(wrhsti_fun,int_oct(j)+',T,F'); line:=header;
      IF first THEN BEGIN first:=FALSE; dpy END;
      ok:=FALSE;
      FOR i:=j DOWNTO 0 DO IF NOT aborted THEN BEGIN
        line:=fit(int_dec(i),4,right_fit)+' '+micro_hist(i)+' '+macro_hist;
        update(FALSE);
        IF aborted OR (((line_num MOD 12)=0) AND (name=''))
        THEN IF line_num>0 THEN aborted:=NOT continue(o,ok);
        IF NOT aborted THEN dpy
      END;
      IF NOT aborted AND (name='') THEN BEGIN
        aborted:=NOT continue(o,ok); ok:=ok OR NOT aborted
      END;
      aborted:=NOT ok
    END
  UNTIL aborted;
  IF name<>'' THEN CLOSE(f) ELSE register_screen
END;

OVERLAY PROCEDURE do_macro;
{define/erase macro definition}
VAR i,index:1..number_macros; name:str_typ; old_macro_ok:BOOLEAN;
BEGIN
  old_macro_ok:=macro_ok; macro_ok:=FALSE;
  IF eq(arg1,'DEFINE',1) THEN BEGIN
```

```
    name:=ask('Define Macro: ',''); index:=0;
    FOR i:=1 TO macro_count DO IF macro_tab[i].name=name THEN index:=i;
    IF (name='') OR (index>0) THEN abort('Already exists: '+name)
    ELSE IF macro_count=number_macros THEN abort('No more space.')
    ELSE IF LENGTH(name)>8 THEN abort('Name too big.')
    ELSE BEGIN
      macro_count:=macro_count+1;
      macro_tab[macro_count].name:=name;
      macro_tab[macro_count].body:=ask('Enter body: ','');
      msg('Defined: '+name)
    END
  END
  ELSE IF eq(arg1,'ERASE',1) THEN BEGIN
    name:=ask('Erase Macro: ',''); index:=0;
    FOR i:=1 TO macro_count DO IF macro_tab[i].name=name THEN index:=i;
    IF index<>0 THEN BEGIN
      FOR i:=index+1 TO macro_count DO macro_tab[i-1]:=macro_tab[i];
      macro_count:=macro_count-1
    END;
    msg(name+' erased.')
  END
  ELSE abort('Options: DEFINE, ERASE');
  macro_ok:=old_macro_ok
END;

OVERLAY PROCEDURE do_build;
{build binary load file from mld format}
VAR res:str_typ;
BEGIN
  res:=callf(build_fun,arg1);
  WHILE (NOT aborted) AND (res<>'OK') DO BEGIN
    msg('Build: '+res); res:=callf(build_fun,'*'); update(FALSE)
  END;
  IF NOT aborted THEN msg('Built: '+arg1+'.BIN')
END;

OVERLAY PROCEDURE do_modem;
{connect/interact communications port}

PROCEDURE send_char(p:BYTE; c:CHAR);
BEGIN
  send(p,c);
  IF log_ok THEN BEGIN
    IF log_open THEN BEGIN
      {$I-} WRITE(log_file,c); {$I+}
      log_open:=IORESULT=0; IF NOT log_open THEN CLOSE(log_file)
    END;
    IF NOT log_open THEN alert('?LOG FILE FULL?')
  END
END;

PROCEDURE local_com(p:BYTE);
{local communications}

PROCEDURE status;
{display port status}
CONST
```

```
com1_base=$300; com2_base=$200;
data_port=$F8; interrupt_enable=$F9; interrupt_id=$FA; line_control=$FB;
modem_control=$FC; line_status=$FD; modem_status=$FE;
VAR b:BYTE;
BEGIN {status}
  CASE p OF
    com1:BEGIN
      WRITELN('--COM1 STATUS--');
      WRITELN('SERVER=',hex_word(OFS(com1_server)));
      b:=$300; WRITELN('BASE=',hex_byte(b));
      WRITELN('ID=',hex_byte(com1_id)); com1_id:=0;
      WRITELN('INPUT_IN=',com1_input_in);
      WRITELN('INPUT_OUT=',com1_input_out);
      WRITELN('INPUT_SIZ=',com1_input_siz);
    END;
    com2:BEGIN
      WRITELN('--COM2 STATUS--');
      WRITELN('SERVER=',hex_word(OFS(com2_server)));
      b:=$200; WRITELN('BASE=',hex_byte(b));
      WRITELN('ID=',hex_byte(com2_id)); com2_id:=0;
      WRITELN('INPUT_IN=',com2_input_in);
      WRITELN('INPUT_OUT=',com2_input_out);
      WRITELN('INPUT_SIZ=',com2_input_siz);
    END;
  END;
  WRITELN('DATA_PORT=',hex_byte(PORT[b+data_port]));
  WRITELN('INT_ENABLE=',hex_byte(PORT[b+interrupt_enable]));
  WRITELN('INT_ID=',hex_byte(PORT[b+interrupt_id]));
  WRITELN('LINE_CONTROL=',hex_byte(PORT[b+line_control]));
  WRITELN('MODEM_CONTROL=',hex_byte(PORT[b+modem_control]));
  WRITELN('LINE_STATUS= ',hex_byte(PORT[b+line_status]));
  WRITELN('MODEM_STATUS=',hex_byte(PORT[b+modem_status]))
END;

VAR q:BOOLEAN; c:CHAR;
BEGIN
  q:=FALSE; WRITELN('F1=EXIT, F2=STATUS'); WRITELN;
  REPEAT
    IF receive(p,c) THEN send_char(con,c);
    IF KEYPRESSED THEN BEGIN
      READ(KBD,c);
      IF c=#27 THEN IF KEYPRESSED THEN BEGIN
        READ(KBD,c);
        IF c=#59 THEN BEGIN q:=TRUE; c:=#0 END
        ELSE IF c=#60 THEN BEGIN status; c:=#0 END
        ELSE send_char(p,#27)
      END;
      IF c<>#0 THEN send_char(p,c)
    END
  UNTIL q OR NOT connected(p)
END;

PROCEDURE remote_cty;
{remote cty communications}
VAR q:BOOLEAN; c:CHAR;
BEGIN
  q:=FALSE; call(cty_fun,'T'); WRITELN('F1=EXIT, F2=QUIT'); WRITELN;
```

```
  REPEAT
    IF receive(modem,c) THEN send_char(con,c);
    IF KEYPRESSED THEN BEGIN
      READ(KBD,c);
      IF c=#27 THEN IF KEYPRESSED THEN BEGIN
        READ(KBD,c);
        IF c=#59 THEN BEGIN
          WRITELN; WRITELN('CTY exit...'); send(modem,#0); send(modem,#1);
          REPEAT REPEAT UNTIL receive(modem,c) OR KEYPRESSED
          UNTIL (c=#2) OR KEYPRESSED; IF KEYPRESSED THEN READ(KBD,c);
          q:=TRUE
        END
        ELSE IF c=#60 THEN BEGIN
          WRITELN; WRITELN('CTY quit.'); q:=TRUE; quit:=TRUE
        END
        ELSE send_char(modem,#27)
      END;
      IF NOT q THEN send_char(modem,c)
    END
  UNTIL q OR NOT connected(modem);
END;

PROCEDURE port_init(c:BYTE; cmd:str_typ);
{set initial port parameters}
VAR b,p,s,w:INTEGER;
BEGIN
  msg('Port init: '+cmd);
  b:=dec_int(parse(cmd)); p:=dec_int(parse(cmd));
  s:=dec_int(parse(cmd)); w:=dec_int(cmd);
  com_init(c,b,p,s,w);
END;

VAR p:BYTE; name,cmd:str_typ; c:CHAR;
BEGIN {do_modem}
  IF eq(arg1,'CTY',1) THEN BEGIN p:=cty; arg1:=arg2; arg2:='' END
  ELSE p:=modem;
  name:=switch(p=cty,'CTY','MODEM');
  IF arg1<>'' THEN port_init(p,unparse(arg1,arg2,''))
  ELSE BEGIN
    IF (name='MODEM') AND NOT connected(p)
    THEN connect_modem(askyn('Connect?'));
    IF (name='CTY') AND (remote=we_are) THEN p:=modem;
    IF connected(p) OR debugging THEN BEGIN
      TEXTCOLOR(modem_color); CLRSCR; current_fmt:=screen_fmt;
      WRITELN(name+' '+time(FALSE)+' '+time(TRUE));
      log('BEGIN '+name);
      IF (name='CTY') AND (remote=we_are) THEN remote_cty ELSE local_com(p);
      log('END '+name);
      new_screen(banner);
      IF (name='MODEM') AND connected(p)
      THEN connect_modem(NOT askyn('Disconnect?'))
    END;
    msg(switch(connected(p),'Connected: ','Disconnected: ')+name);
  END;
  IF KEYPRESSED THEN READ(KBD,c) {clear keyboard buffer}
END;
```

```
LISTING: CONSOLE.FMS  01-JAN-80  00:00:00  (CONSOLE.FMS)

OVERLAY PROCEDURE do_boot;
{interface to system booter}

PROCEDURE boot_macro(args:str_typ);
{boot monitor}
VAR arg,field:bits_typ; bhb,btp,ec,mnp,devn,filen,res:str_typ;
BEGIN
  devn:=parse(args); default(devn,'DISK');
  IF eq(devn,'TAPE',1) THEN BEGIN
    filen:=parse(args); default(filen,'0');
    IF LENGTH(filen)<>1 THEN abort('FILEN/'+filen)
    ELSE IF args<>'' THEN abort('ARGS/'+args)
    ELSE BEGIN
      IF k1 THEN call(wrreg_fun,'AR,40000000000'+filen)
      ELSE call(wrreg_fun,'AR,'+incr_oct(filen));
      start('4044',TRUE,TRUE); wait('4043',300);
      res:=callf(rdreg_fun,'AR');
      IF (NOT k1 AND (res<>'12')) OR
        (k1 AND (res<>'72')) THEN abort('Boot failed: '+res)
      ELSE boot:=switch(k1,'40000','147')
    END
  END
  ELSE IF eq(devn,'DISK',1) THEN BEGIN
    IF NOT k1 THEN BEGIN
      ec:=parse(args); default(ec,'START');
      IF eq(ec,'EDDT',1) THEN BEGIN ec:='0'; call(wrm_fun,'32,0') END
      ELSE IF eq(ec,'START',1) THEN ec:='1'
      ELSE IF eq(ec,'LOAD',1) THEN ec:='2'
      ELSE abort('EC/'+ec);
      btp:=parse(args); default(btp,'0');
      mnp:=parse(args); default(mnp,btp);
      IF eq(mnp,'ANY',1) THEN mnp:='-1'
      ELSE IF eq(mnp,'ASK',1) THEN mnp:='-2'
      ELSE IF mnp='' THEN abort('MNP/'+mnp)
    END;
    bhb:=parse(args); default(bhb,'PRIMARY');
    IF eq(bhb,'PRIMARY',1) THEN bhb:='0'
    ELSE IF eq(bhb,'SECONDARY',1) THEN bhb:='1'
    ELSE abort('BHB/'+bhb);
    IF k1 THEN BEGIN
      filen:=parse(args); default(filen,'3');
      IF LENGTH(filen)<>1 THEN abort('FILEN/'+filen)
      ELSE BEGIN
        IF bhb='0' THEN filen:='40000000000'+filen
        ELSE filen:='60000000000'+filen;
        IF args<>'' THEN abort('ARGS/'+args);
        IF NOT aborted THEN call(wrreg_fun,'AR,'+filen)
      END
    END
    ELSE BEGIN
      IF args<>'' THEN abort('ARGS/'+args);
      IF NOT aborted THEN BEGIN
        oct_to_bits('0',arg,36);
        oct_to_bits(bhb,field,9); put_bits(arg,field,0,9);
        oct_to_bits(btp,field,9); put_bits(arg,field,9,9);
        oct_to_bits(ec,field,9); put_bits(arg,field,18,9);
        oct_to_bits(mnp,field,9); put_bits(arg,field,27,9);
```

```
LISTING: CONSOLE.TAD  01-JAN-00  00:00:00  CONSOLE.TAD;

        call(wram_fun,'16,'+bits_to_oct(arg,36)+',F')
      END
    END;
    IF NOT aborted THEN BEGIN
      start('4045',TRUE,TRUE); wait('4043',5);
      res:=callf(rdreg_fun,'AR');
      IF (NOT k1 AND (res<>'70')) OR
        (k1 AND (res<>'72')) THEN abort('Boot failed: '+res)
      ELSE IF k1 THEN boot:='40000'
      ELSE BEGIN
        boot:=oct_digits(callf(rddma_fun,'3000'),12);
        boot:=int_oct(oct_int(COPY(boot,7,6))-oct_int('1'+COPY(boot,2,5)))
      END
    END
  END
  ELSE abort('DEVN/'+devn)
END;

PROCEDURE boot_micro(args:str_typ);
{boot system micro-code}
VAR arg,field:bits_typ; uhb,mmb,tctl,devn,filen,ucp,res:str_typ;
BEGIN
  devn:=parse(args); default(devn,'DISK');
  IF eq(devn,'TAPE',1) THEN BEGIN
    devn:='1'; uhb:='0'; ucp:='0';
    tctl:=parse(args); default(tctl,'OBJ');
    IF eq(tctl,'OBJ',1) THEN tctl:='0'
    ELSE IF eq(tctl,'SLOEXP',1) THEN tctl:='1'
    ELSE IF eq(tctl,'CCL',1) THEN tctl:='2'
    ELSE IF eq(tctl,'MBOOT',1) THEN tctl:='3'
    ELSE abort('TCTL/'+tctl);
    filen:=parse(args); default(filen,'0')
  END
  ELSE IF eq(devn,'DISK',1) THEN BEGIN
    devn:='0'; tctl:='0';
    filen:=parse(args); default(filen,'BOOTU');
    IF eq(filen,'BOOTU',1) THEN filen:='0'
    ELSE IF eq(filen,'BOOTD',1) THEN filen:='1'
    ELSE IF eq(filen,'MEMIMG',1) THEN filen:='2'
    ELSE IF eq(filen,'BOOT',1) THEN filen:='3'
    ELSE abort('FILEN/'+filen);
    IF k1 THEN ucp:='0'
    ELSE BEGIN ucp:=parse(args); default(ucp,'0') END;
    uhb:=parse(args); default(uhb,'PRIMARY');
    IF eq(uhb,'PRIMARY',1) THEN uhb:='0'
    ELSE IF eq(uhb,'SECONDARY',1) THEN uhb:='1'
    ELSE abort('UHB/'+uhb)
  END
  ELSE abort('DEVN/'+devn);
  mmb:=parse(args); default(mmb,'NEW');
  IF eq(mmb,'OLD',1) THEN mmb:='1'
  ELSE IF eq(mmb,'NEW',1) THEN mmb:='0'
  ELSE abort('MMB/'+mmb);
  IF args<>'' THEN abort('ARGS/'+args);
  IF NOT aborted THEN BEGIN
    oct_to_bits('0',arg,36);
    oct_to_bits(uhb,field,1); put_bits(arg,field,0,1);
```

```
LISTING: CONSOLE.CMD  01-JAN-80  00:00:00  CONSOLE.CMD

    oct_to_bits(mmb,field,1); put_bits(arg,field,1,1);
    oct_to_bits(tctl,field,6); put_bits(arg,field,3,6);
    oct_to_bits(devn,field,9); put_bits(arg,field,9,9);
    oct_to_bits(filen,field,9); put_bits(arg,field,18,9);
    oct_to_bits(ucp,field,9); put_bits(arg,field,27,9);
    call(wram_fun,'16,'+bits_to_oct(arg,36)+',F')
  END;
  IF NOT aborted THEN BEGIN
    start('17000',TRUE,TRUE);
    IF devn='0' THEN wait('4043,17001',5) ELSE wait('4043,17001',300);
    res:=callf(rdreg_fun,'AR');
    IF res<>'72' THEN abort('Boot failed: '+res)
  END
END;

PROCEDURE getargs;
{get boot arguments}
VAR a:str_typ;
BEGIN
  a:=ask('Boot MICRO/MACRO:','MICRO'); args:=a+' ';
  IF eq(a,'MACRO',2) THEN BEGIN
    a:=ask('From DISK/TAPE:','DISK'); args:=args+a+' ';
    IF eq(a,'TAPE',1) THEN args:=args+ask('Tape file number:','0')
    ELSE IF eq(a,'DISK',1) THEN BEGIN
      IF NOT k1 THEN BEGIN
        args:=args+ask('Entry code START/LOAD/EDDT:','START')+' ';
        args:=args+ask('Bootstrap disk number:','0')+' ';
        args:=args+ask('Monitor disk number/ANY/ASK:','0')
      END;
      args:=args+ask('Home block PRIMARY/SECONDARY:','PRIMARY');
      IF k1 THEN args:=args+' '+ask('Monitor file number:','3')
    END
  END
  ELSE IF eq(a,'MICRO',2) THEN BEGIN
    a:=ask('From DISK/TAPE:','DISK'); args:=args+a+' ';
    IF eq(a,'TAPE',1) THEN BEGIN
      args:=args+ask('File type SLOEXP/MBOOT/OBJ/CCL:','SLOEXP')+' ';
      args:=args+ask('Tape file number:','0')+' '
    END
    ELSE IF eq(a,'DISK',1) THEN BEGIN
      args:=args+ask('Boot file BOOTU/BOOTD/MEMIMG/BOOT:','BOOTU')+' ';
      IF NOT k1 THEN args:=args+ask('Disk number:','0')+' ';
      args:=args+ask('Home block PRIMARY/SECONDARY:','PRIMARY')+' '
    END;
    args:=args+ask('MMB is NEW/OLD:','NEW')
  END
END;

BEGIN {do_boot}
  boot:=''; IF args='' THEN getargs;
  arg2:=args; arg1:=parse(arg2);
  IF eq(arg1,'MACRO',1) THEN boot_macro(arg2)
  ELSE IF eq(arg1,'MICRO',1) THEN boot_micro(arg2)
  ELSE abort('Options: MACRO, MICRO')
END;

OVERLAY PROCEDURE do_dump;
```

```
{dump micro/macro memory}

PROCEDURE dump(micro:BOOLEAN; name:str_typ);
{dump interface to backend cycles}
VAR a,b,e:str_typ;
BEGIN
  {open dump file, call of two arguments}
  call(save_fun,'T,F'); call(dump_fun,switch(micro,'T,','F,')+name);
  IF NOT aborted THEN REPEAT
    {solicit beginning,ending addresses}
    a:=ask(switch(micro,'Micro','Macro')+' addresses:','');
    b:=parse(a); e:=parse(a);
    IF (a='') AND (e<>'') THEN REPEAT
      {do dump cycle, call of three arguments}
      b:=callf(dump_fun,switch(micro,'T,','F,')+b+',+e);
      IF NOT aborted THEN msg(switch(micro,'Micro','Macro')+' dump: '+b);
      update(FALSE) {keep the clock ticking}
    UNTIL aborted OR (b='OK')
    ELSE IF b<>'' THEN abort('Enter: <beginning>,<ending>')
  UNTIL aborted OR (b='');
  {close dump file, call of one argument}
  call(dump_fun,switch(micro,'T','F')); call(save_fun,'F,F')
END;

BEGIN {do_dump}
  default(arg2,'MICRO');
  IF eq(arg2,'MICRO',2) THEN dump(TRUE,arg1)
  ELSE IF eq(arg2,'MACRO',2) THEN dump(FALSE,arg1)
  ELSE abort('Options: MICRO, MACRO')
END;

OVERLAY PROCEDURE do_load;
{load micro/macro memory}

PROCEDURE macro_load(name,code:str_typ);
{load macro memory}
VAR load:BYTE; res,sys:str_typ;
BEGIN
  IF code='' THEN load:=load_slow
  ELSE IF eq(code,'VERIFY',1) THEN load:=load_verify
  ELSE IF eq(code,'CHECK',1) THEN load:=load_check
  ELSE abort('Options: VERIFY, CHECK');
  IF NOT aborted THEN BEGIN
    sys:=switch(kl,'KL','');
    IF eq(name,'DIAG',1) THEN name:=sys+'DIAG';
    res:=callf(load_fun,'F,'+name+',+int_dec(load));
    WHILE (NOT aborted) AND (res<>'OK') DO BEGIN
      msg('Loading: '+res); res:=callf(load_fun,'F'); update(FALSE)
    END;
    IF NOT aborted THEN CASE load OF
      load_slow:msg(name+' loaded.');
      load_verify:msg(name+' verified.');
      load_check:msg(name+' checked.')
    END
  END
END;
```

```
LISTING: CONSOLE.PAS  01-JAN-80  00:00:00  (CONSOLE.PAS)

PROCEDURE micro_load(name,code:str_typ);
{load micro memory}
VAR load:BYTE; dmaad,ending,res,sys:str_typ;
BEGIN
  IF eq(name,'FAST',1) THEN fast_mode:=FALSE;
  IF code='' THEN BEGIN
    IF fast_mode THEN load:=load_fast ELSE load:=load_slow
  END
  ELSE IF eq(code,'VERIFY',1) THEN load:=load_verify
  ELSE IF eq(code,'CHECK',1) THEN load:=load_check
  ELSE IF eq(code,'FAST',1) THEN load:=load_fast
  ELSE IF eq(code,'SLOW',1) THEN load:=load_slow
  ELSE abort('Options: VERIFY, CHECK, FAST, SLOW');
  IF NOT aborted THEN BEGIN
    sys:=switch(kl,'KL','');
    IF eq(name,'FAST',1) THEN name:=sys+'FAST'
    ELSE IF eq(name,'DIAG',1) THEN name:=sys+'DIAG'
    ELSE IF eq(name,'BOOT',1) THEN name:=sys+'BOOT'
    ELSE IF eq(name,'SYSTEM',1) THEN name:=sys+'SYSTEM';
    IF load=load_fast THEN BEGIN
      dmaad:=dpy_tab[dmaad_dpy].value; dpy_tab[dmaad_dpy].value:='00000030';
      idles([idle_dma]); start('17650',TRUE,TRUE); fast_mode:=TRUE
    END
    ELSE BEGIN call(save_fun,'T,F'); idles([idle_mic]) END;
    dpy(load_dpy,name); res:=callf(load_fun,'T,'+name+','+int_dec(load));
    WHILE (NOT aborted) AND (res<>'OK') DO BEGIN
      msg('Loading: '+res); update(TRUE);
      IF NOT aborted AND (res<>'OK') THEN BEGIN
        IF load=load_fast THEN WITH dpy_tab[dma_dpy] DO BEGIN
          IF value<>oct_digits(res,value_siz)
          THEN abort('Fast load error: '+value+'/'+res)
        END
        ELSE WITH dpy_tab[mic_dpy] DO BEGIN
          IF value<>oct_digits(res,value_siz)
          THEN abort('Slow load error: '+value+'/'+res)
        END;
        IF aborted THEN fast_mode:=FALSE
      END;
      IF NOT aborted THEN res:=callf(load_fun,'T');
    END;
    update(TRUE);
    IF load=load_fast THEN BEGIN
      run(FALSE);
      ending:=dpy_tab[dma_dpy].value; dpy_tab[dmaad_dpy].value:=dmaad
    END
    ELSE BEGIN ending:=dpy_tab[mic_dpy].value; call(save_fun,'F,F') END;
    idle_codes:=old_idle_codes;
    IF NOT aborted THEN BEGIN
      start(ending,TRUE,FALSE); do_command(dd_cmd,'');
      CASE load OF
        load_slow,load_fast:msg(name+' loaded.');
        load_verify:msg(name+' verified.');
        load_check:msg(name+' checked.')
      END;
      IF name=(sys+'FAST') THEN fast_mode:=TRUE
    END
  END
```

```
LISTING: CONSOLE.PAS  01-JAN-80  00:00:00  "CONSOLE.PAS"

END;

VAR c:str_typ;
BEGIN {do_load}
  c:=parse(arg2);
  IF eq(c,'MACRO',2) THEN macro_load(arg1,arg2)
  ELSE BEGIN
    IF NOT eq(c,'MICRO',2) THEN arg2:=unparse(c,arg2,'');
    micro_load(arg1,arg2)
  END
END;

OVERLAY PROCEDURE do_call;
{interactively call backend function}
VAR found:BOOLEAN; i:cmd_typ; key:str_typ;
BEGIN
  key:=arg1; arg1:=parse(arg2);
  IF command_ok(help_cmd,key,arg1,arg2) THEN BEGIN
    IF help(bits_fun,wrum_fun,arg1) THEN type_file('CONSOLE.HLP',arg1)
  END
  ELSE BEGIN
    found:=FALSE;
    FOR i:=bits_fun TO wrum_fun DO
      IF command_ok(i,key,arg1,arg2) THEN BEGIN
        found:=TRUE; msg(key+'='+callf(i,unparse(arg1,arg2,'')));
      END;
    IF NOT found THEN abort('Invalid call: '+key)
  END
END;

OVERLAY PROCEDURE do_test;
{operate internal console test functions in the backend}
VAR pass_count:INTEGER; test_count:1..number_tests;

PROCEDURE test(test_num:BYTE; multiple,infinite:BOOLEAN);
{perform specific test}
VAR err_msg:str_typ;

PROCEDURE failed(test_num:BYTE; dat:str_typ);
{display message when test fails}

FUNCTION name:str_typ;
{return test name}
BEGIN
  CASE test_num OF
    1:name:='CABLE'; 2:name:='LOCRES'; 3:name:='CLDIAG'; 4:name:='LDDIAG';
    5:name:='LOCPAR'; 6:name:='BANK0'; 7:name:='BANK1'; 8:name:='BANK2';
    9:name:='BANK3'; 10:name:='BANK4'; 11:name:='BANK5'; 12:name:='BANK6';
    13:name:='BANK7'; 14:name:='ZEROS'; 15:name:='ONES'; 16:name:='0-1-1';
    17:name:='1-2-2'; 18:name:='2-3-4'; 19:name:='3-4-8'; 20:name:='4-5-1';
    21:name:='5-6-2'; 22:name:='6-7-4'; 23:name:='7-0-8'; 24:name:='0-2-16';
    25:name:='0-4-32'; 26:name:='0-0-64'; 27:name:='IORA'; 28:name:='IORB';
    29:name:='OBUS'; 30:name:='DMA'; 31:name:='MI'; 32:name:='MIC';
    33:name:='UMEM'; 34:name:='PC'; 35:name:='MA'; 36:name:='AR';
    37:name:='MEM'; 38:name:='IR'; 39:name:='Q'; 40:name:='JMEMP';
    41:name:='DEV'; 42:name:='AMEMP'; 43:name:='AC'; 44:name:='AMEM';
    45:name:='MEMORY'
```

```
  END
END;

BEGIN
  err_msg:=int_dec(test_num)+' '+name;
  IF dat<>'' THEN err_msg:=err_msg+' '+parse(dat);
  IF dat<>'' THEN err_msg:=err_msg+'/'+dat;
  msg('Failed: '+err_msg)
END;

VAR result:str_typ; again:BOOLEAN; c:CHAR;
BEGIN {test}
  again:=infinite;
  REPEAT
    aborted:=update_time AND KEYPRESSED;
    IF KEYPRESSED THEN READ(KBD,c)
    ELSE IF NOT aborted THEN BEGIN
      result:=callf(test_fun,int_dec(test_num));
      IF aborted THEN result:='?';
      IF result<>'OK' THEN BEGIN
        failed(test_num,result);
        IF NOT infinite THEN again:=askyn('Again?');
        IF NOT again THEN BEGIN
          aborted:=NOT multiple;
          IF NOT aborted THEN aborted:=NOT continue;
          IF aborted THEN IF askyn('Explain?') THEN BEGIN
            error(err_msg); aborted:=FALSE;
            type_file('CONSOLE.HLP','TEST'+int_dec(test_num));
            aborted:=TRUE
          END
        END
      END
    END
  UNTIL aborted OR NOT again
END;

VAR old_update_ok:BOOLEAN;
BEGIN {do_test}
  old_update_ok:=update_ok; update_ok:=FALSE;
  IF arg1='' THEN BEGIN
    FOR test_count:=1 TO number_tests DO IF NOT aborted THEN BEGIN
      msg(int_dec(test_count)); test(test_count,TRUE,FALSE)
    END;
    IF NOT aborted THEN msg('OK')
  END
  ELSE IF arg1='*' THEN BEGIN
    pass_count:=1;
    REPEAT
      FOR test_count:=1 TO number_tests DO IF NOT aborted THEN BEGIN
        msg(int_dec(test_count)+'  Pass '+int_dec(pass_count));
        test(test_count,TRUE,FALSE)
      END;
      IF pass_count=32767 THEN pass_count:=0;
      pass_count:=pass_count+1
    UNTIL aborted
  END
  ELSE BEGIN
```

```
    test_count:=1+((dec_int(arg1)-1) MOD number_tests);
    msg(int_dec(test_count)+arg2);
    test(test_count,FALSE,arg2='*');
    IF NOT aborted THEN msg('OK')
  END;
  call(reset_fun,'');
  update_ok:=old_update_ok
END;

OVERLAY PROCEDURE do_set;
{set values of flags}

PROCEDURE start_counters;
{turn counters on, zero all}
VAR i:1..number_counters;
BEGIN
  counting:=TRUE; FOR i:=1 TO number_counters DO activity_counter[i]:=0
END;

PROCEDURE stop_counters;
{turn counters off, sort and display}
VAR map:ARRAY[1..number_counters] OF 1..number_counters;

PROCEDURE sort_counters;
{sort activity counters by amount of activity}
VAR i,j,k:1..number_counters;
BEGIN
  msg('Sorting...');
  FOR i:=1 TO number_counters DO map[i]:=i;
  FOR i:=1 TO number_counters DO
    FOR j:=number_counters-1 DOWNTO i DO BEGIN
      IF activity_counter[map[j+1]]>activity_counter[map[j]] THEN BEGIN
        k:=map[j]; map[j]:=map[j+1]; map[j+1]:=k
      END
    END
END;

VAR i:BYTE; n,c:str_typ; done:BOOLEAN;
BEGIN {stop_counters}
  counting:=FALSE; line_screen; sort_counters; i:=0; done:=FALSE;
  WHILE (i<number_counters) AND NOT done DO BEGIN
    IF (i>0) AND ((i MOD 12)=0) THEN BEGIN
      done:=NOT continue; IF NOT done THEN line_screen
    END;
    IF NOT done THEN BEGIN
      i:=i+1; STR(map[i]:0,n); STR(activity_counter[map[i]]:0,c);
      dpy_line(i-1,fit(n,3,left_fit)+' -- '+c)
    END
  END;
  IF NOT done THEN done:=continue;
  register_screen
END;

PROCEDURE set_flag(n:str_typ; VAR f:BOOLEAN);
{set flag on/off}
BEGIN
  default(arg1,switch(f,'OFF','ON'));
```

```
  IF flag(arg1,f) THEN msg(n+' '+switch(f,'ON','OFF'))
  ELSE abort('Options: ON, OFF')
END;

VAR key:str_typ;
BEGIN {do_set}
  key:=arg1; arg1:=parse(arg2);
  IF eq(key,'COUNTERS',1) THEN BEGIN
    set_flag('Counters',counting);
    IF counting THEN start_counters ELSE stop_counters
  END
  ELSE IF eq(key,'DD',2) THEN set_flag('Display',dd_ok)
  ELSE IF eq(key,'DEBUG',1) THEN set_flag('Debug',debugging)
  ELSE IF eq(key,'LOCK',3) THEN set_flag('Lock',locked)
  ELSE IF eq(key,'LOG',3) THEN set_flag('Log',log_ok)
  ELSE IF eq(key,'QUIET',1) THEN set_flag('Quiet',quiet)
  ELSE IF eq(key,'UPDATE',1) THEN set_flag('Update',update_ok)
  ELSE IF eq(key,'SIMULATE',1) THEN set_flag('Simulate',simulate)
  ELSE abort('Options: DD, DEBUG, LOG, QUIET')
END;

OVERLAY PROCEDURE do_reset;
{set remote access, password}
BEGIN
  IF flag(arg1,remote_ok) THEN BEGIN
    password:=arg2;
    IF NOT remote_ok THEN connect_modem(FALSE)
    ELSE IF remote<>we_are THEN connect_modem(TRUE);
    {see call to do_reset below for remote=we_are}
    msg('Remote access: '+switch(remote_ok,'ON','OFF')+
      switch(password<>'',' "'+password+'"',''))
  END
  ELSE abort('Options: ON [password], OFF')
END;

OVERLAY PROCEDURE reset_console;
{reset entire console program, initial values}

PROCEDURE play(m:str_typ);
{play music from specified file}
VAR f:TEXT; hz,c:REAL; i,n,s,o:INTEGER; x:STRING[2];
BEGIN
  ASSIGN(f,m); {$I-} RESET(f); {$I+}
  IF IORESULT=0 THEN BEGIN
    alert('Hello...');
    WHILE NOT (EOF(f) OR KEYPRESSED) DO BEGIN
      READ(f,x);
      IF x<>'' THEN BEGIN
        IF x='C ' THEN n:=1 ELSE IF x='CF' THEN n:=2
        ELSE IF x='D ' THEN n:=3 ELSE IF x='DF' THEN n:=4
        ELSE IF x='E ' THEN n:=5 ELSE IF x='F ' THEN n:=6
        ELSE IF x='FF' THEN n:=7 ELSE IF x='G ' THEN n:=8
        ELSE IF x='GF' THEN n:=9 ELSE IF x='A ' THEN n:=10
        ELSE IF x='AF' THEN n:=11 ELSE IF x='B ' THEN n:=12
        ELSE n:=0;
        READ(f,x);
        IF x='E ' THEN s:=1 ELSE IF x='Q ' THEN s:=2
```

```
        ELSE IF x='DQ' THEN s:=3 ELSE IF x='H ' THEN s:=4
        ELSE IF x='DH' THEN s:=6 ELSE IF x='W ' THEN s:=8
        ELSE s:=0;
        c:=(125/0.9)*s;
        READ(f,o);
        IF n>0 THEN BEGIN
          hz:=32.625; FOR i:=1 TO o DO hz:=hz*2;
          FOR i:=1 TO n-1 DO hz:=hz*1.059463094;
          SOUND(ROUND(hz)); DELAY(ROUND(c)); NOSOUND
        END
        ELSE DELAY(ROUND(c))
      END;
      READLN(f)
    END
  END
END;

PROCEDURE reset_inits;
{reset init (delay) values}
VAR new,old:str_typ;
BEGIN
  new:='';
  old:=callf(inits_fun,'F');
  new:=new+','+ask('ROW ENB DLY:',parse(old));
  new:=new+','+ask('CAS DLY:',parse(old));
  new:=new+','+ask('WRT DLY:',parse(old));
  new:=new+','+ask('EA MPX DLY CTRL:',parse(old));
  new:=new+','+ask('STATUS CLK DLY:',parse(old));
  new:=new+','+ask('MID CY CLK DLY CTRL:',parse(old));
  new:=new+','+ask('MAIN CLK DLY CTRL:',parse(old));
  new:=new+','+ask('MC MEM GO DLY:',parse(old));
  new:=new+','+ask('MC MM ENB DLY:',parse(old));
  new:=new+','+ask('SM CYLEN:',parse(old));
  new:=new+','+ask('SM DISP CYLEN:',parse(old));
  new:=new+','+ask('FM CYLEN:',parse(old));
  new:=new+','+ask('FM DISP CYLEN:',parse(old));
  new:=new+','+ask('MC MEM GO POST-DLY:',parse(old));
  call(inits_fun,'T'+new)
END;

PROCEDURE reset_system;
{reset system}
BEGIN
  do_command(ubrk_cmd,'OFF'); do_command(mbrk_cmd,'OFF'); call(reset_fun,'');
  dpy(mad_dpy,init_dma); dpy(dmaad_dpy,init_dma); do_command(dd_cmd,'')
END;

PROCEDURE reset_console;
{reset console program -- frontend}
VAR init_delays,init_idle:str_typ;

PROCEDURE initial_values;
{set up initial values}
VAR i:BYTE;
BEGIN
  init_delays:='4,7,2,7,5,7,5,3,0,11,1,11,11,4'; init_idle:='';
  {global}
```

```
  counting:=FALSE; debugging:=FALSE; debug_counter:=0;
  aborted:=FALSE; k1:=FALSE; log_open:=FALSE; log_ok:=FALSE;
  {frontend}
  current_fmt:=screen_fmt; dd_ok:=TRUE; fast_mode:=FALSE;
  cty:=com1; modem:=com2;
  FOR i:=1 TO 12 DO talk_line[i]:=''; FOR i:=1 TO 4 DO scroll_line[i]:='';
  dialing:=FALSE; macro_count:=0; macro_ok:=TRUE;
  cmdfil_name:=''; cmdfil_arg1:=''; cmdfil_arg2:='';
  updating:=FALSE; update_ok:=TRUE;
  password_count:=0; password:=''; request:='';
  remote_ok:=FALSE; remote:=hello;
  simulate:=FALSE; quit:=FALSE;  quiet:=FALSE;  locked:=FALSE; running:=FALSE;
  old_date:=''; old_time:=''; old_key:=''; old_arg1:=''; old_arg2:='';
  boot:=''; banner:=''; last_msg:=''; command_name:='';
  idle_codes:=[]; old_idle_codes:=[]; ugo_diag:='';
  ugo_lit:=''; ugo_jmem:=''; ugo_amem:=''; ugo_mem:=''; ugo_cty:='';
  com_init(cty,1200,0,1,8);
  IF NOT connected(modem) THEN com_init(modem,1200,0,1,8)
END;

PROCEDURE command_args;
{arguments from MSDOS command line}

PROCEDURE arg(value:str_typ);
{process argument}
BEGIN
  IF eq(value,'REMOTE',1) THEN remote:=we_are
  ELSE IF eq(value,'DEBUG',2) THEN debugging:=TRUE
  ELSE IF eq(value,'SIMULATE',8) THEN simulate:=TRUE
  ELSE command_file(value,'','')
END;

VAR i:BYTE;
BEGIN {command_args}
  debug('command_args');
  IF arg2='*' THEN FOR i:=1 TO PARAMCOUNT DO arg(uppercase(PARAMSTR(i)))
  ELSE WHILE arg2<>'' DO arg(parse(arg2))
END;

PROCEDURE file_args;
{arguments from console initialization file}

PROCEDURE open_com(VAR com:BYTE; args:str_typ);
{reset communications port}
VAR b,p,s,w:INTEGER;
BEGIN
  com:=dec_int(parse(args)); b:=dec_int(parse(args));
  p:=dec_int(parse(args)); s:=dec_int(parse(args));
  w:=dec_int(parse(args)); com_init(com,b,p,s,w)
END;

PROCEDURE open_log(n:str_typ);
{open log file}
VAR f:TEXT; l,d:str_typ;
BEGIN
  log_ok:=parse(n)='ON';
  IF log_ok THEN BEGIN
```

```
    log_open:=TRUE; IF POS('.',n)=0 THEN n:=n+'.LOG';
    debug('LOG FILE: '+n);
    ASSIGN(log_file,n); {$I-} RESET(log_file); {$I+}
    IF IORESULT=0 THEN BEGIN {append existing log file}
      CLOSE(log_file);
      d:=n; IF POS(':',d)=0 THEN d:='' ELSE DELETE(d,POS(':',d)+1,LENGTH(d
      ASSIGN(f,n); RENAME(f,d+'CONSOLE.TMP'); RESET(f);
      REWRITE(log_file);
      WHILE NOT EOF(f) AND NOT aborted DO BEGIN
        READLN(f,l);
        {$I-} WRITELN(log_file,l); {$I+}
        IF IORESULT<>0 THEN abort('LOG FULL: '+n)
      END;
      CLOSE(f);
      IF NOT aborted THEN ERASE(f) ELSE BEGIN RENAME(f,n); log_open:=FALSE END
    END
    ELSE REWRITE(log_file)
  END
  ELSE log_open:=FALSE
END;

VAR f:TEXT; code,value:str_typ;
BEGIN {file_args}
  ASSIGN(f,'CONSOLE.INI'); {$I-} RESET(f); {$I+}
  IF IORESULT=0 THEN WHILE NOT EOF(f) DO BEGIN
    READLN(f,value); value:=uppercase(value); debug('OPTION: '+value);
    IF value<>'' THEN IF value[1]<>';' THEN BEGIN
      code:=parse(value);
      IF code='CTY:' THEN open_com(cty,value)
      ELSE IF code='DELAYS:' THEN init_delays:=value
      ELSE IF code='DIAG:' THEN BEGIN
        code:=parse(value);
        IF code='LIT' THEN ugo_lit:=value
        ELSE IF code='JMEM' THEN ugo_jmem:=value
        ELSE IF code='AMEM' THEN ugo_amem:=value
        ELSE IF code='MEM' THEN ugo_mem:=value
        ELSE IF code='CTY' THEN ugo_cty:=value
        ELSE ugo_diag:=code
      END
      ELSE IF code='DIAL:' THEN dialing:=value='ON'
      ELSE IF code='IDLE:' THEN init_idle:=value
      ELSE IF code='LOG:' THEN open_log(value)
      ELSE IF code='MACRO:' THEN BEGIN
        IF macro_count<number_macros THEN BEGIN
          macro_count:=macro_count+1;
          macro_tab[macro_count].name:=parse(value);
          macro_tab[macro_count].body:=value
        END
      END
      ELSE IF code='MODEM:' THEN open_com(modem,value)
      ELSE IF code='REMOTE:' THEN BEGIN
        remote_ok:=parse(value)='ON'; password:=value
      END
      ELSE abort('Invalid paramter: '+code)
    END
  END
END;
```

```
PROCEDURE connect_cty(yes:BOOLEAN);
{connect cty port}
BEGIN
  IF NOT yes THEN connect(cty,FALSE)
  ELSE IF NOT connected(cty) THEN connect(cty,yes)
END;

PROCEDURE reset_screen;
{reset display screen}
VAR i:dpy_typ; t:str_typ; old_quiet:BOOLEAN;
BEGIN
  IF kl THEN t:=' (KL)' ELSE t:='';
  new_screen('Augment Engine'+t+' #'+callf(sysid_fun,'T')+
    ' Console V'+version+edit);
  old_quiet:=quiet; quiet:=FALSE;
  FOR i:=mbrk_dpy TO mi_dpy DO dpy(i,'0');
  dpy(ubrk_dpy,'OFF'); dpy(mbrk_dpy,'OFF');
  dpy(mad_dpy,init_dma); dpy(dmaad_dpy,init_dma);
  dpy(amemad_dpy,init_amem);
  status(TRUE); quiet:=old_quiet
END;

BEGIN {reset_console}
  CLRSCR; play('CONSOLE.MUS'); initial_values; dpy(dpy_init,'');
  command_args; file_args;
  IF NOT aborted THEN BEGIN
    IF remote=we_are THEN BEGIN
      connect_cty(FALSE); connect_modem(TRUE)
    END
    ELSE BEGIN
      connect_cty(TRUE); IF connected(modem) THEN connect_modem(TRUE)
    END
  END;
  IF NOT aborted THEN BEGIN
    call(init_fun,'F,'+init_delays); kl:=callf(sysid_fun,'F')='KL';
    init_amem:='16'; init_dma:=switch(kl,'67','30');
    reset_screen; call(instrs_fun,switch(kl,'KL','')+'INSTRS');
    idle(init_idle); log(banner+switch(remote=we_are,'  *REMOTE*',''))
  END
END;

VAR res:str_typ;
BEGIN {reset_console}
  default(arg1,'4042');
  IF eq(arg1,'INITS',1) THEN reset_inits
  ELSE IF eq(arg1,'SYSTEM',1) THEN reset_system
  ELSE IF eq(arg1,'CONSOLE',1) THEN reset_console
  ELSE BEGIN
    default(arg2,'4043,17001');
    arg1:=oct_bits(arg1,14);
    IF NOT (digit_overflow OR bad_digit) THEN BEGIN
      start(arg1,TRUE,TRUE); wait(arg2,5);
      IF (arg1='4042') AND NOT aborted THEN BEGIN
        res:=callf(rdreg_fun,'AR');
        IF res<>'1' THEN abort('Reset failed: '+res)
      END
```

```
    END
    ELSE abort('Options: INITS, SYSTEM, CONSOLE, address')
  END
END;

OVERLAY PROCEDURE examine_deposit;
{overlay examine/deposit commands}

FUNCTION sect:str_typ;
{parse micro instruction section}
VAR s:BYTE;
BEGIN
  s:=sect_ABC;
  IF arg1='A' THEN s:=sect_A
  ELSE IF arg1='B' THEN s:=sect_B
  ELSE IF arg1='C' THEN s:=sect_C;
  sect:=int_dec(s); IF s<>sect_ABC THEN arg1:=parse(arg2)
END;

PROCEDURE dreg(i:dpy_typ; reg:str_typ; c:cmd_typ; a:str_typ);
{deposit register}
BEGIN dpy_default(arg1,i); call(wrreg_fun,reg+', '+arg1); do_command(c,a) END;

FUNCTION xregf(reg:str_typ; item:dpy_typ):str_typ;
{examine register}
BEGIN xregf:=dpyf(item,callf(rdreg_fun,reg)) END;

VAR arg3:str_typ;
BEGIN {examine_deposit}
  CASE cmd_code OF
    dac_cmd:BEGIN
      dpy_default(arg1,acad_dpy); dpy_default(arg2,ac_dpy);
      call(wrac_fun,arg1+', '+arg2); do_command(xac_cmd,arg1);
    END;
    dam_cmd:BEGIN
      dpy_default(arg1,amemp_dpy); dpy_default(arg2,amem_dpy);
      call(wram_fun,arg1+', '+arg2+',F'); do_command(xam_cmd,arg1)
    END;
    dami_cmd:BEGIN
      dpy_default(arg2,amem_dpy);
      call(wram_fun,arg1+', '+arg2+',T'); do_command(xami_cmd,arg1)
    END;
    damp_cmd:dreg(amem_dpy,'AMEM',xamp_cmd,'');
    de_cmd:BEGIN
      dpy_default(arg1,mad_dpy); dpy_default(arg2,m_dpy);
      call(wrm_fun,arg1+', '+arg2); do_command(ex_cmd,arg1)
    END;
    dmrd_cmd:BEGIN
      dpy_default(arg1,dmaad_dpy);
      msg('(DMA) MEMORY['+
        dpyf(dmaad_dpy,oct_bits(arg1,24))+']='+
        oct_word(dpyf(dma_dpy,callf(rddma_fun,arg1))))
    END;
    dmwrt_cmd:BEGIN
      dpy_default(arg1,dmaad_dpy); dpy_default(arg2,dma_dpy);
      call(wrdma_fun,arg1+', '+arg2); do_command(dmrd_cmd,arg1)
    END;
```

```
ex_cmd:BEGIN
  dpy_default(arg1,mad_dpy);
  msg('MEMORY['+
    dpyf(mad_dpy,oct_bits(arg1,18))+']='+
    oct_word(dpyf(m_dpy,callf(rdm_fun,arg1)))+
    '<'+dpyf(ecc_dpy,callf(rdecc_fun,''))+'>')
END;
ldac_cmd:dreg(ac_dpy,'AC',xac_cmd,dpy_tab[dev_dpy].value);
ldamemp_cmd:dreg(amemp_dpy,'AMEMP',xamemp_cmd,'');
ldar_cmd:dreg(ar_dpy,'AR',xar_cmd,'');
lddev_cmd:dreg(dev_dpy,'DEV',xdev_cmd,'');
ldhold_cmd:dreg(mem_dpy,'HOLD',xmem_cmd,'');
ldir_cmd:dreg(ir_dpy,'IR',xir_cmd,'');
ldjmemp_cmd:dreg(jmemp_dpy,'JMEMP',xjmemp_cmd,'');
ldma_cmd:dreg(ma_dpy,'MA',xma_cmd,'');
ldpc_cmd:dreg(pc_dpy,'PC',xpc_cmd,'');
ldpcf_cmd:dreg(pc_dpy,'PCF',xpcf_cmd,'');
ldq_cmd:dreg(q_dpy,'Q',xq_cmd,'');
miload_cmd:BEGIN
  arg2:=sect; dpy_default(arg1,mi_dpy); call(wrmi_fun,arg1+','+arg2);
  do_command(mird_cmd,'')
END;
mird_cmd:BEGIN
  status(TRUE); msg('MI='+dpyf(mi_dpy,callf(rdmi_fun,'')))
END;
mmload_cmd:BEGIN
  arg3:=sect; dpy_default(arg1,mic_dpy);
  dpy_default(arg2,mi_dpy); call(wrum_fun,arg1+','+arg2+','+arg3);
  do_command(mmrd_cmd,arg1)
END;
mmrd_cmd:BEGIN
  dpy_default(arg1,mic_dpy);
  IF (arg2<>'') AND NOT eq(arg2,'DISPLAY',1)
  THEN abort('Options: DISPLAY')
  ELSE IF arg2<>'' THEN msg('UMEM['+oct_digits(arg1,5)+']='+
    dpyf(mi_dpy,callf(rdum_fun,arg1)))
  ELSE msg('UMEM['+oct_digits(arg1,5)+']='+
    oct_digits(callf(rdum_fun,arg1),30))
END;
rdobus_cmd:BEGIN
  default(arg1,'T'); arg1:=arg1+',F';
  msg('OBUS='+oct_word(dpyf(obus_dpy,callf(rdob_fun,arg1))));
  dpy(eobus_dpy,callf(rdob_fun,'F,T'))
END;
rdstatus_cmd:msg('STATUS='+oct_digits(callf(rdstlp_fun,''),43));
wrobus_cmd:BEGIN
  dpy_default(arg1,obus_dpy);
  call(wrob_fun,arg1); do_command(rdobus_cmd,'F')
END;
xac_cmd:BEGIN
  dpy_default(arg1,acad_dpy);
  msg('AC['+dpyf(acad_dpy,oct_bits(arg1,4))+']='+
    oct_word(dpyf(ac_dpy,callf(rdac_fun,arg1))))
END;
xam_cmd:BEGIN
  dpy_default(arg1,amemad_dpy);
  msg('AMEM['+dpyf(amemad_dpy,oct_bits(arg1,14))+']='+
```

```
        oct_word(dpyf(amem_dpy,callf(rdam_fun,arg1+',F'))))
    END;
    xami_cmd:msg('(I) AMEM['+
      dpyf(amemad_dpy,int_oct(oct_int(arg1)+
        (oct_int(dpy_tab[dev_dpy].value) SHL 4)))+']='+
      oct_word(dpyf(amem_dpy,callf(rdam_fun,arg1+' TRUE'))));
    xamp_cmd:BEGIN
      do_command(xamemp_cmd,'');
      msg('AMEM['+dpyf(amemad_dpy,dpy_tab[amemp_dpy].value)+']='+
        oct_word(xregf('AMEM',amem_dpy)))
    END;
    xamemp_cmd:msg('AMEMP='+xregf('AMEMP',amemp_dpy));
    xar_cmd:msg('AR='+oct_word(xregf('AR',ar_dpy)));
    xdev_cmd:msg('DEV='+xregf('DEV',dev_dpy));
    xir_cmd:msg('IR='+oct_word(xregf('IR',ir_dpy)));
    xjmem_cmd:msg('JMEM['+xregf('JMEMP',jmemp_dpy)+']='+
      oct_word(xregf('JMEM',jmem_dpy)));
    xjmemp_cmd:msg('JMEMP='+xregf('JMEMP',jmemp_dpy));
    xlp_cmd:msg('LP='+xregf('LP',lp_dpy));
    xma_cmd:msg('MA='+oct_word(xregf('MA',ma_dpy)));
    xmem_cmd:msg('MEM='+oct_word(xregf('MEM',mem_dpy)));
    xpc_cmd:msg('PC='+oct_word(xregf('PC',pc_dpy)));
    xpcf_cmd:msg('PCF='+callf(rdreg_fun,'PCF'));
    xq_cmd:msg('Q='+oct_word(xregf('Q',q_dpy)));
  END
END;

PROCEDURE diag;
{load/execute micro-code diagnostics}
VAR data,addr:bits_typ; ok:BOOLEAN;
BEGIN
  do_command(reset_cmd,'SYS'); do_command(load_cmd,'FAST');
  IF NOT aborted THEN do_command(load_cmd,'DIAG');
  IF aborted THEN BEGIN
    aborted:=NOT askyn('Slow load?');
    run(FALSE); do_command(reset_cmd,'SYS'); do_command(load_cmd,'DIAG')
  END;
  IF NOT aborted THEN BEGIN {diag loaded ok}
    msg('DIAG'); dpy(dmaad_dpy,'30'); start('DIAG',TRUE,TRUE);
    idles([idle_mic,idle_dma]); idle_count:=0;
    REPEAT ok:=idle_count>10; update(FALSE);
    UNTIL ok OR (dpy_tab[mic_dpy].value='16777');
    aborted:=FALSE; run(FALSE); idle_codes:=old_idle_codes;
    IF NOT ok THEN BEGIN {address from history, t-5}
      call(wrhsti_fun,'3,T,T');
      oct_to_bits(callf(rdhst_fun,'T'),data,36);
      IF data.bit[9]=0 THEN BEGIN
        get_bits(data,addr,28,4); addr.siz:=5; addr.bit[4]:=data.bit[27];
        abort('DIAG=TR'+oct_digits(bits_to_oct(addr,5),2))
      END
      ELSE BEGIN
        get_bits(data,addr,12,14);
        abort('DIAG='+oct_digits(bits_to_oct(addr,14),5))
      END
    END
    ELSE msg('Passed.')
  END;
```

```
END;

VAR saved:BOOLEAN;
BEGIN {command}
  log(cmd_tab[cmd_code].name+'('+args+')');
  arg2:=args; arg1:=parse(arg2);
  saved:=cmd_code IN [dac_cmd,dam_cmd,dami_cmd,damp_cmd,
    dd_cmd,de_cmd,ex_cmd,history_cmd,ldac_cmd,ldamemp_cmd,ldar_cmd,
    lddev_cmd,ldhold_cmd,ldir_cmd,ldjmemp_cmd,ldma_cmd,ldpc_cmd,
    ldpcf_cmd,ldq_cmd,mbrk_cmd,mmload_cmd,mmrd_cmd,ubrk_cmd,
    xac_cmd,xam_cmd,xami_cmd,xamp_cmd,xamemp_cmd,xar_cmd,xdev_cmd,
    xir_cmd,xjmem_cmd,xjmemp_cmd,xlp_cmd,xma_cmd,xmem_cmd,xpc_cmd,
    xpcf_cmd,xq_cmd];
  IF saved THEN call(save_fun,'T,F');
  IF NOT aborted THEN CASE cmd_code OF
    boot_cmd:do_boot;
    build_cmd:do_build;
    cs_cmd:BEGIN do_clock(arg1); do_dd END;
    cty_cmd:BEGIN
      IF arg1<>'' THEN IF unlocked THEN start(arg1,FALSE,TRUE);
      IF NOT aborted THEN BEGIN arg1:='C'; do_modem END
    END;
    dd_cmd:do_dd;
    dump_cmd:do_dump;
    call_cmd:do_call;
    help_cmd:IF help(boot_cmd,xq_cmd,args)
      THEN type_file('CONSOLE.HLP',args);
    history_cmd:do_history;
    idle_cmd:idle(args);
    load_cmd:do_load;
    macro_cmd:do_macro;
    mbrk_cmd:BEGIN
      dpy_default(arg1,mad_dpy);
      dpy(mbrk_dpy,callf(break_fun,unparse('F',arg1,arg2)))
    END;
    mgo_cmd:BEGIN do_command(mstart_cmd,args); run(TRUE) END;
    mic_cmd:msg('MIC='+dpyf(mic_dpy,callf(rdmic_fun,'')));
    modem_cmd:do_modem;
    mstart_cmd:BEGIN
      IF arg2<>'' THEN do_command(mbrk_cmd,arg2);
      dpy_default(arg1,m_dpy); start(arg1,FALSE,FALSE);
      do_command(ex_cmd,arg1)
    END;
    quit_cmd:quit:=TRUE;
    remote_cmd:BEGIN
      do_reset;
      IF remote_ok AND (remote=we_are) AND NOT aborted
      THEN do_command(reset_cmd,'C,R')
    END;
    reset_cmd:reset_console;
    run_cmd:run(TRUE);
    set_cmd:do_set;
    ss_cmd:BEGIN
      IF unlocked THEN BEGIN
        IF running THEN run(FALSE);
        {call(enahst_fun,'T');}
        default(arg1,'1'); call(step_fun,'T,'+arg1);
```

```
        {call(enahst_fun,'F');}
        do_dd
      END
    END;
    stop_cmd:BEGIN run(FALSE); do_dd END;
    talk_cmd:do_talk;
    test_cmd:BEGIN do_test; IF (arg1='') AND NOT aborted THEN diag END;
    type_cmd:type_file(arg1,arg2);
    ubrk_cmd:BEGIN
      dpy_default(arg1,mic_dpy);
      dpy(ubrk_dpy,callf(break_fun,unparse('T',arg1,arg2)))
    END;
    ugo_cmd:BEGIN do_command(ustart_cmd,args); run(TRUE) END;
    ui_cmd:do_command(miload_cmd,
      callf(setfld_fun,unparse(dpy_tab[mi_dpy].value,arg1,arg2)));
    ustart_cmd:BEGIN
      dpy_default(arg1,mic_dpy);
      IF arg2<>'' THEN do_command(ubrk_cmd,arg2);
      start(arg1,TRUE,FALSE); do_command(mic_cmd,''); do_command(mird_cmd,'')
    END;
    ELSE examine_deposit;
  END;
  IF saved THEN call(save_fun,'F,F')
END;

VAR i:cmd_typ; ok:BOOLEAN; k,a1,a2:str_typ;
BEGIN {command}
  alert(''); aborted:=FALSE; repeat_ok:=TRUE; debug_counter:=0;
  IF args=#0 THEN do_command(reset_cmd,'C,*')
  ELSE BEGIN
    k:=parse(args); a2:=args; a1:=parse(a2);
    IF k='' THEN command_repeat(k,a1,a2); ok:=k='';
    IF NOT ok THEN BEGIN
      command_name:=k; msg('');
      FOR i:=boot_cmd TO xq_cmd DO IF NOT ok THEN
        IF command_ok(i,k,a1,a2) THEN BEGIN
          do_command(i,unparse(a1,a2,'')); k:=cmd_tab[i].name; ok:=TRUE
        END;
      IF NOT ok AND NOT aborted THEN command_file(k,a1,a2)
    END;
    IF repeat_ok AND NOT aborted THEN BEGIN
      old_key:=k; old_arg1:=a1; old_arg2:=a2
    END
    ELSE BEGIN old_key:=''; old_arg1:=''; old_arg2:='' END
  END
END;

BEGIN {frontend}
  {initial setup -- express call to reset console}
  command(#0);
  {do not continue if initial reset aborted}
  log('CONSOLE ON:  '+time(FALSE));
  IF debugging OR NOT aborted THEN REPEAT command(ask(prompt,'')) UNTIL quit;
  log('CONSOLE OFF:  '+time(FALSE));
  {close log file}
  IF log_open THEN CLOSE(log_file)
END;
```

```
OVERLAY PROCEDURE tables;
{load up all the constant tables}

{$I CONSOLE3.PAS the table loaders}
{CONSOLE3.PAS -- source file for Augment Engine Console (5 of 5) zw}

{contains routines to load constant tables}

PROCEDURE load_mi_tab;
{load micro instruction table -- load is finished with instrs in backend}
VAR zero:bits_typ;

PROCEDURE x(i:mi_typ; n:str_typ);
BEGIN
  WITH mi_tab[i] DO BEGIN name:=n; bits:=zero END
END;

VAR i:BYTE;
BEGIN {load_mi_tab}
  zero.siz:=11; zero.bit_siz:=8; FOR i:=0 TO 10 DO zero.bit[i]:=0;
  x(clcpmod_mi,'CLCPMOD'); x(clexctx_mi,'CLEXCTX'); x(cljmemp_mi,'CLJMEMP');
  x(clmerge_mi,'CLMERGE'); x(clusctx_mi,'CLUSCTX'); x(clvamod_mi,'CLVAMOD');
  x(dam_mi,'DAM'); x(jump_mi,'JUMP'); x(klres1_mi,'KLRES1');
  x(klres2_mi,'KLRES2'); x(ldac_mi,'LDAC'); x(ldamar_mi,'LDAMAR');
  x(ldamemp_mi,'LDAMEMP'); x(ldar_mi,'LDAR'); x(lddev_mi,'LDDEV');
  x(ldhold_mi,'LDHOLD'); x(ldjmemp_mi,'LDJMEMP');
  x(ldir_mi,'LDIR'); x(ldlpq_mi,'LDLPQ'); x(ldma_mi,'LDMA');
  x(ldpc_mi,'LDPC'); x(ldpcf_mi,'LDPCF'); x(ldq_mi,'LDQ');
  x(ldqones_mi,'LDQONES'); x(lmhadr_mi,'LMHADR');
  x(lmhbkad_mi,'LMHBKAD'); x(lmhcnt_mi,'LMHCNT');
  x(lmhctl_mi,'LMHCTL'); x(luhadr_mi,'LUHADR'); x(luhctl_mi,'LUHCTL');
  x(mapdis_mi,'MAPDIS'); x(memst_mi,'MEMST'); x(mhoff_mi,'MHOFF');
  x(mhon_mi,'MHON'); x(mrd_mi,'MRD'); x(uhoff_mi,'UHOFF');
  x(uhon_mi,'UHON'); x(xac_mi,'XAC'); x(xam_mi,'XAM');
  x(xamemp_mi,'XAMEMP'); x(xar_mi,'XAR'); x(xcond_mi,'XCOND');
  x(xdev_mi,'XDEV'); x(xir_mi,'XIR'); x(xjmem_mi,'XJMEM');
  x(xjmemp_mi,'XJMEMP'); x(xloop_mi,'XLOOP'); x(xma_mi,'XMA');
  x(xmbst_mi,'XMBST'); x(xmem_mi,'XMEM'); x(xmhadr_mi,'XMHADR');
  x(xmhdat0_mi,'XMHDAT0'); x(xmhdat1_mi,'XMHDAT1');
  x(xpc_mi,'XPC'); x(xpcf_mi,'XPCF'); x(xq_mi,'XQ'); x(xuhadr_mi,'XUHADR');
  x(xuhdat_mi,'XUHDAT');
  x(zero_mi,'ZERO')
END;

PROCEDURE load_fld_tab;
{load micro instruction field table}

PROCEDURE x(i:fld_typ; n:str_typ; p,l:BYTE; d:dpy_typ);
BEGIN
  WITH fld_tab[i] DO BEGIN name:=n; bit_zero:=p; bits_siz:=l; dpy:=d END
END;

BEGIN {load_fld_tab}
  x(fld_err,'ERR',0,0,dpy_init); x(eeal_fld,'EEAL',0,1,eeal_dpy);
  x(eefo_fld,'EEFO',1,1,eefo_dpy); x(ldma_fld,'LDMA',2,1,ldma_dpy);
  x(idisp_fld,'IDISP',3,1,idisp_dpy); x(mwt_fld,'MWT',4,1,mwt_dpy);
```

```
  x(saafma_fld,'SAAFMA',5,1,saafma_dpy); x(po_fld,'PO',6,1,po_dpy);
  x(sp1_fld,'SP1',7,1,sp1_dpy); x(cry_fld,'CRY',8,1,cry_dpy);
  x(asrc_fld,'ASRC',9,3,asrc_dpy); x(afun_fld,'AFUN',12,3,afun_dpy);
  x(adst_fld,'ADST',15,3,adst_dpy); x(alu1_fld,'ALU1',18,1,alu1_dpy);
  x(ldar_fld,'LDAR',19,1,ldar_dpy); x(jcond_fld,'JCOND',20,6,jcond_dpy);
  x(mapf_fld,'MAPF',26,4,mapf_dpy); x(spc_fld,'SPC',30,6,spc_dpy);
  x(jadr_fld,'JADR',36,14,jadr_dpy); x(rot_fld,'ROT',50,6,rot_dpy);
  x(mask_fld,'MASK',56,6,mask_dpy); x(dest_fld,'DEST',62,6,dest_dpy);
  x(sp2_fld,'SP2',68,1,sp2_dpy); x(jcode_fld,'JCODE',69,4,jcode_dpy);
  x(acsel_fld,'ACSEL',73,3,acsel_dpy); x(d_fld,'D',76,6,d_dpy);
  x(cylen_fld,'CYLEN',82,4,cylen_dpy); x(ifq_fld,'IFQ',86,1,ifq_dpy);
  x(dfq_fld,'DFQ',87,1,dfq_dpy); x(lit_fld,'LIT',26,36,lit_dpy)
END;

PROCEDURE load_reg_tab;
{load machine register table}

PROCEDURE x(i:reg_typ; n:str_typ; l:BYTE; wr,rd:mi_typ);
BEGIN
  WITH reg_tab[i] DO BEGIN
    name:=n; bits_siz:=l; mi_to_wr:=wr; mi_to_rd:=rd
  END
END;

BEGIN {load_reg_tab}
  x(reg_err,'ERR',0,zero_mi,zero_mi);
  x(amem_reg,'AMEM',36,dam_mi,xam_mi);
  x(amemp_reg,'AMEMP',10,ldamemp_mi,xamemp_mi);
  x(ar_reg,'AR',36,ldar_mi,xar_mi);
  x(dev_reg,'DEV',5,lddev_mi,xdev_mi);
  x(ir_reg,'IR',36,ldir_mi,xir_mi);
  x(jmemp_reg,'JMEMP',10,ldjmemp_mi,xjmemp_mi);
  x(jmem_reg,'JMEM',14,zero_mi,xjmem_mi);
  x(lp_reg,'LP',16,zero_mi,xloop_mi);
  x(ma_reg,'MA',36,ldma_mi,xma_mi);
  x(mem_reg,'MEM',36,ldhold_mi,xmem_mi);
  x(pc_reg,'PC',36,ldpc_mi,xpc_mi);
  x(pcf_reg,'PCF',36,ldpcf_mi,xpcf_mi);
  x(q_reg,'Q',36,ldq_mi,xq_mi);
  x(mbst_reg,'MBST',36,zero_mi,xmbst_mi);
  x(ac_reg,'AC',36,ldac_mi,xac_mi);
  x(hold_reg,'HOLD',36,ldhold_mi,xmem_mi);
  x(am_reg,'AM',36,dam_mi,xam_mi)
END;

PROCEDURE load_dpy_tab;
{load display item table}

{345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678
        Load: LOADFILE        State: Running          Remote: Inactive
  MBRK=000000     MI[000000]=000000,,000000/000   DMA[00000000]=000000,,000000
  PC=000000,,000000  MA=000000,,000000  AR=000000,,000000 MEM=000000,,000000
  IR=000000,,000000   Q=000000,,000000  LP=000000       AC[00]=000000,,000000
        JMEM[0000]=00000      DEV=00  AMEMP=0000     AMEM[0000]=000000,,000000
   UBRK=00000  MIC=00000   OBUS=000000,,000000          EOBUS=000000,,000000

  LC=OFF  TC=OFF   Branching      MI=00000000000000000000000000000000
```

```
LISTING: CONSOLE.PAS  01-JAN-80  00:00:00  CONSOLE.PAS;

  EEAL=0  EEFO=0   LDMA=0    IDISP=0    MWT=0  SAAFMA=0    PO=0  SP1=0 CRY=0
  ASRC=0  AFUN=0   ADST=0     ALU1=0   LDAR=0   JCOND=00
  MAPF=00  SPC=00  JADR=00000  ROT=00  MASK=00     LIT=000000,,000000
  DEST=00  SP2=0  JCODE=00   ACSEL=0       D=00  CYLEN=00  IFQ=0  DFQ=0
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456}

PROCEDURE x(i:dpy_typ; x,y,l:BYTE; d:str_typ);
BEGIN
  WITH dpy_tab[i] DO BEGIN
    x_pos:=x; y_pos:=y; desc:=d; value:=''; value_siz:=l
  END
END;

BEGIN {load_dpy_tab}
  x(line1_dpy,1,1,79,''); x(line2_dpy,1,2,79,''); x(line3_dpy,1,3,79,'');
  x(line4_dpy,1,4,79,''); x(line5_dpy,1,5,79,''); x(line6_dpy,1,6,79,'');
  x(line7_dpy,1,7,79,''); x(line8_dpy,1,8,79,''); x(line9_dpy,1,9,79,'');
  x(line10_dpy,1,10,79,''); x(line11_dpy,1,11,79,''); x(line12_dpy,1,12,79,'').
  x(load_dpy,9,1,8,'Load: '); x(run_dpy,30,1,7,'State: ');
  x(remote_dpy,54,1,10,'Remote: '); x(mbrk_dpy,3,2,6,'MBRK=');
  x(mad_dpy,18,2,6,'MD['); x(m_dpy,26,2,12,']='); x(ecc_dpy,42,2,3,'//');
  x(dmaad_dpy,49,2,8,'DMA['); x(dma_dpy,61,2,12,']=');
  x(pc_dpy,3,3,12,'PC='); x(ma_dpy,22,3,12,'MA='); x(ar_dpy,41,3,12,'AR=');
  x(mem_dpy,59,3,12,'MEM='); x(ir_dpy,3,4,12,'IR='); x(q_dpy,23,4,12,'Q=');
  x(lp_dpy,41,4,6,'LP='); x(acad_dpy,56,4,2,'AC['); x(ac_dpy,61,4,12,']=');
  x(jmemp_dpy,9,5,4,'JMEM['); x(jmem_dpy,18,5,5,']=');
  x(dev_dpy,30,5,2,'DEV='); x(amemp_dpy,38,5,4,'AMEMP=');
  x(amemad_dpy,52,5,4,'AMEM['); x(amem_dpy,61,5,12,']=');
  x(ubrk_dpy,4,6,5,'UBRK='); x(mic_dpy,16,6,5,'MIC=');
  x(obus_dpy,28,6,12,'OBUS='); x(eobus_dpy,57,6,12,'EOBUS=');
  x(lc_dpy,3,8,5,'LC='); x(tc_dpy,12,8,5,'TC='); x(br_dpy,21,8,9,'');
  x(mi_dpy,35,8,30,'MI='); x(eeal_dpy,3,9,1,'EEAL=');
  x(eefo_dpy,11,9,1,'EEFO='); x(ldma_dpy,20,9,1,'LDMA=');
  x(idisp_dpy,30,9,1,'IDISP='); x(mwt_dpy,41,9,1,'MWT=');
  x(saafma_dpy,48,9,1,'SAAFMA='); x(po_dpy,60,9,1,'PO=');
  x(sp1_dpy,66,9,1,'SP1='); x(cry_dpy,72,9,1,'CRY=');
  x(asrc_dpy,3,10,1,'ASRC='); x(afun_dpy,11,10,1,'AFUN=');
  x(adst_dpy,20,10,1,'ADST='); x(alu1_dpy,31,10,1,'ALU1=');
  x(ldar_dpy,40,10,1,'LDAR='); x(jcond_dpy,49,10,2,'JCOND=');
  x(mapf_dpy,3,11,2,'MAPF='); x(spc_dpy,12,11,2,'SPC=');
  x(jadr_dpy,20,11,5,'JADR='); x(rot_dpy,32,11,2,'ROT=');
  x(mask_dpy,40,11,2,'MASK='); x(dest_dpy,3,12,2,'DEST=');
  x(sp2_dpy,12,12,1,'SP2='); x(jcode_dpy,19,12,2,'JCODE=');
  x(acsel_dpy,30,12,1,'ACSEL='); x(d_dpy,43,12,2,'D=');
  x(cylen_dpy,49,12,2,'CYLEN='); x(ifq_dpy,59,12,1,'IFQ=');
  x(dfq_dpy,66,12,1,'DFQ='); x(lit_dpy,51,11,12,'LIT=');
  x(dd_dpy,1,1,0,''); x(dpy_init,1,1,0,'')
END;

PROCEDURE load_cmd_tab;
{load command table}

PROCEDURE x(i:cmd_typ; n:str_typ; a:BYTE; p1,p2,d:str_typ);
BEGIN
  WITH cmd_tab[i] DO BEGIN
    name:=n; eq_siz:=a; arg1_desc:=p1; arg2_desc:=p2; cmd_desc:=d
  END
```

```
END;

BEGIN {load_cmd_tab}
  x(bits_fun,'BITS',2,'bits','siz,bit_siz,LEFT/RIGHT','Convert bits.');
  x(break_fun,'BREAK',4,'micro?,addr/ON/OFF','[count,delay]','Break point.');
  x(build_fun,'BUILD',2,'file name','','Build binary load file.');
  x(cpuob_fun,'CPUOB',2,'yes?','','CPU controls OBUS.');
  x(cty_fun,'CTY',2,'[yes?]','','Remote CTY mode.');
  x(dump_fun,'DUMP',1,'micro?','[file/begin,end]','Dump memory.');
  x(enahst_fun,'ENAHST',2,'yes?','','Enable history recording.');
  x(exec_fun,'EXEC',2,'instr','clock?','Execute(and clock) instr.');
  x(fld_fun,'FLD',1,'field name','','Convert field name.');
  x(init_fun,'INIT',4,'send?','','Initialize (and send) flags.');
  x(init2_fun,'INIT2',5,'[on]','[off]','Read/write INIT2.');
  x(inits_fun,'INITS',5,'all?','','Send (all) inits.');
  x(instrs_fun,'INSTRS',3,'file name','','Load constant instructions.');
  x(iora_fun,'IORA',4,'[on]','[off]','Read/write IORA.');
  x(iorb_fun,'IORB',4,'[on]','[off]','Read/write IORB.');
  x(jump_fun,'JUMP',1,'address','','Build jump instr.');
  x(load_fun,'LOAD',1,'micro?','file,code','Load slow,verify,check,fast.');
  x(mi_fun,'MI',2,'instr','','Translate instruction.');
  x(mmactrl_fun,'MMACTRL',2,'[on]','[off]','Read/write MMACTRL.');
  x(rdac_fun,'RDAC',4,'AC number','','Read AC.');
  x(rdam_fun,'RDAM',4,'address','indexed?','Read AMEM.');
  x(rddma_fun,'RDDMA',5,'address','','DMA memory read.');
  x(rdecc_fun,'RDECC',4,'','','Read memory ECC.');
  x(rdhst_fun,'RDHST',3,'micro?','[index,relative?]','Read history.');
  x(rdhsti_fun,'RDHSTI',6,'relative?','micro?','Read history index.');
  x(rdm_fun,'RDM',3,'address','','Read memory.');
  x(rdmi_fun,'RDMI',4,'','','Read micro instruction.');
  x(rdmic_fun,'RDMIC',5,'','','Read micro instruction counter.');
  x(rdob_fun,'RDOB',4,'load?','eobus?','(Load and) read OBUS.');
  x(rdport_fun,'RDPORT',3,'port','','Read port.');
  x(rdreg_fun,'RDREG',3,'register','','Read register.');
  x(rdst_fun,'RDST',4,'','','Read status items.');
  x(rdstlp_fun,'RDSTLP',5,'','','Read status loop.');
  x(rdum_fun,'RDUM',4,'address','','Read micro memory.');
  x(reg_fun,'REG',2,'register','','Register name conversion.');
  x(reset_fun,'RESET',4,'','','Reset system after board change.');
  x(run_fun,'RUN',2,'yes?','','Run/Stop CPU.');
  x(save_fun,'SAVE',2,'yes?','force?','Save MIC and MI.');
  x(setfld_fun,'SETFLD',4,'instr','field,value','Set field value.');
  x(shift_fun,'SHIFT',2,'count','','Shift load registers.');
  x(step_fun,'STEP',3,'load?','count','Step (and load) micro instr.');
  x(sysid_fun,'SYSID',2,'system number?','','Read system number or type.');
  x(talk_fun,'TALK',2,'remote?','[message]','Talk message mover.');
  x(test_fun,'TEST',2,'number','','Execute test routine.');
  x(wrac_fun,'WRAC',4,'AC number','value','Write AC.');
  x(wram_fun,'WRAM',4,'address,value','indexed?','Write AMEM.');
  x(wrdma_fun,'WRDMA',5,'address','value','DMA wr memory.');
  x(wrhsti_fun,'WRHSTI',4,'index,relative?','micro?','Write history index.');
  x(wrm_fun,'WRM',3,'address','value','Write memory.');
  x(wrmi_fun,'WRMI',4,'instr','section','Write micro instruction.');
  x(wrmic_fun,'WRMIC',5,'address','first?,load?','Write MIC.');
  x(wrob_fun,'WROB',4,'value','','Write obus.');
  x(wrport_fun,'WRPORT',3,'port','value','Write port.');
  x(wrreg_fun,'WRREG',3,'register','value','Write register.');
```

```
x(wrum_fun,'WRUM',4,'address','instr,section','Write micro memory.');
x(boot_cmd,'BOOT',2,'[MICRO/MACRO]','*','Bootstrap operation.');
x(build_cmd,'BUILD',2,'file name','','Build BIN file from MLD.');
x(call_cmd,'CALL',2,'function','[parameter(s)]','Call backend function.');
x(cs_cmd,'CS',2,'[count/*]','','*Clock step.');
x(cty_cmd,'CTY',2,'[mgo]','*','Connect to CTY.');
x(dac_cmd,'DAC',3,'AC number','value','*Deposit into AC.');
x(dam_cmd,'DAM',3,'address','value','*Deposit into AMEM.');
x(dami_cmd,'DAMI',4,'address','value','*Deposit into AMEM(I).');
x(damp_cmd,'DAMP',4,'value','','*Deposit into AMEM(P).');
x(dd_cmd,'DD',2,'','','*Display all registers.');
x(de_cmd,'DE',2,'address','value','*Deposit main memory');
x(dmrd_cmd,'DMRD',3,'address','','DMA memory read.');
x(dmwrt_cmd,'DMWRT',3,'address','value','?*DMA memory write.');
x(dump_cmd,'DUMP',2,'file','[MICRO/MACRO]','*Dump micro or macro memory.');
x(ex_cmd,'EX',2,'address','','*Examine memory.');
x(help_cmd,'?',1,'[command]','','Describe command.');
x(history_cmd,'HISTORY',2,'[offset]','[file]','*History analysis.');
x(idle_cmd,'IDLE',2,'OFF,MIC,OBUS,DMA','*','Idle dpy.');
x(ldac_cmd,'LDAC',4,'value','','*Load AC(DEV).');
x(ldamemp_cmd,'LDAMEMP',4,'value','','*Load AMEMP.');
x(ldar_cmd,'LDAR',4,'value','','*Load AR.');
x(lddev_cmd,'LDDEV',3,'value','','*Load DEV.');
x(ldjmemp_cmd,'LDJMEMP',3,'value','','*Load JMEMP.');
x(ldhold_cmd,'LDHOLD',3,'value','','*Load HOLD.');
x(ldir_cmd,'LDIR',3,'value','','*Load IR.');
x(ldma_cmd,'LDMA',4,'value','','*Load MA.');
x(ldpc_cmd,'LDPC',4,'value','','*Load PC.');
x(ldpcf_cmd,'LDPCF',5,'value','','*Load PCF.');
x(ldq_cmd,'LDQ',3,'value','','*Load Q.');
x(load_cmd,'LOAD',3,'file','[MICRO/MACRO,CHECK/VERIFY]','*Load memory.');
x(macro_cmd,'MACRO',2,'DEFINE/ERASE','','Macro facility.');
x(mbrk_cmd,'MBRK',3,'address/OFF','[count,delay]]','*Macro break.');
x(mgo_cmd,'MGO',2,'address','[break]','*Macro start and go.');
x(mic_cmd,'MIC',3,'','','Display MIC from status.');
x(miload_cmd,'MILOAD',3,'[A,B,C]','[instr]','*Load MI(section).');
x(miird_cmd,'MIREAD',3,'','','*Read current MI.');
x(mmload_cmd,'MMLOAD',3,'[A,B,C]','[address [instr]]','*Load UMEM.');
x(mmrd_cmd,'MMREAD',3,'address','[DISPLAY]','*Read UMEM.');
x(modem_cmd,'MODEM',2,'[CTY]','[control]','Communications port.');
x(mstart_cmd,'MSTART',2,'address','[break]','*Set macro start address.');
x(quit_cmd,'QUIT',1,'','','Exit console program.');
x(rdobus_cmd,'RDOBUS',3,'[TRUE/FALSE]','','Read(load) obus.');
x(rdstatus_cmd,'RDSTATUS',3,'','','Read status loop.');
x(remote_cmd,'REMOTE',3,'ON/OFF','[password]','Remote access.');
x(reset_cmd,'RESET',3,'[address/INITS/SYSTEM/CONSOLE]','*','Reset system.');
x(run_cmd,'RUN',2,'','','*Run system.');
x(set_cmd,'SET',2,'[DEBUG/LOCK/LOG/QUIET]','[ON/OFF]','Set flags.');
x(ss_cmd,'SS',2,'[count]','','Single step micro instruction.');
x(stop_cmd,'STOP',2,'','','Stop system.');
x(talk_cmd,'TALK',2,'','','Enter talk mode.');
x(test_cmd,'TEST',2,'[number/*]','[*]','Execute test routine(s).');
x(type_cmd,'TYPE',2,'file name','[match]','Display contents of a file.');
x(ubrk_cmd,'UBRK',2,'address/OFF','[count]','*Micro break point.');
x(ugo_cmd,'UGO',2,'address','[break]','*Start micro execution.');
x(ui_cmd,'UI',2,'field','value','*Set micro instr field.');
x(ustart_cmd,'USTART',2,'address','[break]','*Set micro start address.');
```

```
  x(wrobus_cmd,'WROBUS',2,'value','','*Write obus.');
  x(xac_cmd,'XAC',3,'AC number','','*Examine AC.');
  x(xam_cmd,'XAM',3,'address','','*Examine AMEM.');
  x(xami_cmd,'XAMI',4,'address','','*Examine AMEM(I).');
  x(xamp_cmd,'XAMP',4,'','','*Examine AMEM(P).');
  x(xamemp_cmd,'XAMEMP',4,'','','*Examine AMEMP.');
  x(xar_cmd,'XAR',3,'','','*Examine AR.');
  x(xdev_cmd,'XDEV',2,'','','*Examine DEV.');
  x(xir_cmd,'XIR',2,'','','*Examine IR.');
  x(xjmem_cmd,'XJMEM',5,'','','*Examine JMEM(P).');
  x(xjmemp_cmd,'XJMEMP',6,'','','*Examine JMEMP.');
  x(xlp_cmd,'XLP',2,'','','*Examine LP.');
  x(xma_cmd,'XMA',3,'','','*Examine MA.');
  x(xmem_cmd,'XMEM',4,'','','*Examine MEM.');
  x(xpc_cmd,'XPC',3,'','','*Examine PC.');
  x(xpcf_cmd,'XPCF',4,'','','*Examine PCF.');
  x(xq_cmd,'XQ',2,'','','*Examine Q.')
END;

VAR i:BYTE; c:CHAR;
BEGIN {tables}
  load_mi_tab; load_fld_tab; load_reg_tab; load_dpy_tab; load_cmd_tab;
  {load character and digit tables}
  FOR i:=0 TO 9 DO char_tab[i]:=CHR(ORD('0')+i);
  FOR i:=10 TO 16 DO char_tab[i]:=CHR(ORD('A')+i-10);
  FOR c:=CHR(0) TO CHR(127) DO digit_tab[c]:=-1;
  FOR c:='0' TO '9' DO digit_tab[c]:=ORD(c)-ORD('0');
  FOR c:='A' TO 'F' DO digit_tab[c]:=ORD(c)-ORD('A')+10;
  {load test data}
  oct_to_bits('OFF',test_bits[1],88); oct_to_bits('ON',test_bits[2],88);
  oct_to_bits('ODD',test_bits[3],88); oct_to_bits('EVEN',test_bits[4],88);
  test_byte[1]:=oct_int('OFF') AND $FF;
  test_byte[2]:=oct_int('ON') AND $FF;
  test_byte[3]:=oct_int('ODD') AND $FF;
  test_byte[4]:=oct_int('EVEN') AND $FF
END;

BEGIN
  log_open:=FALSE; log_ok:=FALSE;
  debugging:=initial_debugging; debug_counter:=0;
  alert('CONSOLE '+version+edit); debug('BREAK: '+hex_word(OFS(debug_break)))
  interrupts(TRUE); tables; frontend; interrupts(FALSE); CLRSCR
END.
```

ABORT	3
DO_INCLUDE	3
FINISH_PAGE	2
INCLUDE_FILE	3
INDEX_NAME	2
LIST_INDEX	3
PAGE_CHECK	2
PRINT	2
TIME	1
UPPERCASE	1
ZAP	2
ZAP	3

```
PROGRAM list;

{TURBO-PASCAL compiler switches, IBM-PC version}
{$R- subscript checks}
{$V- value checks}
{$U- xon/xoff}
{$I+ i/o check}
{$C- control-c}
{$K- stack check}

CONST names_size=1000;

TYPE
string_type=STRING[80];
register_type=RECORD ax,bx,cx,dx,bp,si,ds,es,flags:INTEGER END;

VAR
printer:TEXT;
aborted:BOOLEAN;
side:INTEGER;
lines_printed:INTEGER;
page_number:INTEGER;
start_page:INTEGER;
print_on:BOOLEAN;
name_count:INTEGER;
names:ARRAY[1..names_size] OF RECORD name:STRING[20]; number:INTEGER END;

PROCEDURE uppercase(VAR s:string_type);
VAR i:INTEGER;
BEGIN
  FOR i:=1 TO LENGTH(s) DO s[i]:=UPCASE(s[i])
END;

FUNCTION time(get_time:BOOLEAN): string_type;
CONST
time_code=$2C; date_code=$2A;
month: ARRAY[1..12] OF STRING[3] =
  ('Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec');
VAR regs:register_type; d,h,m,s:STRING[2]; y:STRING[4];
BEGIN
  WITH regs DO BEGIN
    IF get_time THEN ax:=time_code SHL 8 ELSE ax:=date_code SHL 8;
    MSDOS(regs);
    IF get_time THEN BEGIN
      STR(cx SHR 8:0, h); STR(cx AND $FF:0, m); STR(dx SHR 8:0, s);
      IF LENGTH(h)<2 THEN h:='0'+h;
      IF LENGTH(m)<2 THEN m:='0'+m;
      IF LENGTH(s)<2 THEN s:='0'+s;
      time:=h+':'+m+':'+s
    END
    ELSE BEGIN
      STR(cx:4,y); STR(dx AND $FF:0, d); y:=COPY(y,3,2);
      IF LENGTH(d)<2 THEN d:='0'+d;
      time:=d+'-'+month[dx SHR 8]+'-'+y
    END
  END
END;
```

```
PROCEDURE finish_page;
VAR i:INTEGER;
BEGIN
  IF print_on THEN BEGIN
    FOR i:=1 TO 61-lines_printed DO WRITELN(printer);
    IF side<>2 THEN WRITE(printer,'   ');
    WRITE(printer,' ':35,'Page ',page_number:0,^M,^L);
  END;
  page_number:=page_number+1; print_on:=page_number>=start_page;
  WRITE(^M,page_number:3); lines_printed:=-1
END;

PROCEDURE page_check(title:string_type);
BEGIN
  IF lines_printed=60 THEN finish_page;
  IF lines_printed<0 THEN BEGIN
    CASE side OF
      0:;
      1:IF (page_number MOD 2)=0 THEN print_on:=FALSE;
      2:IF (page_number MOD 2)=1 THEN print_on:=FALSE;
    END;
    IF print_on THEN BEGIN
      WRITELN(printer);
      IF side<>2 THEN WRITE(printer,'   '); WRITELN(printer,title);
      WRITELN(printer)
    END;
    lines_printed:=3
  END
END;

PROCEDURE print(line,title:string_type);
BEGIN
  page_check(title);
  IF print_on THEN BEGIN
    IF side<>2 THEN WRITE(printer,'   '); WRITELN(printer,line)
  END;
  lines_printed:=lines_printed+1
END;

PROCEDURE index_name(line:string_type; page_number:INTEGER);
PROCEDURE zap(i:INTEGER); BEGIN IF i>0 THEN DELETE(line,i,LENGTH(line)) END;
VAR i,j:INTEGER;
BEGIN
  IF POS('OVERLAY ',line)=1 THEN DELETE(line,1,8);
  IF POS('PROCEDURE ',line)=1 THEN DELETE(line,1,10)
  ELSE IF POS('FUNCTION ',line)=1 THEN DELETE(line,1,9)
  ELSE line:='';
  zap(POS('(',line)); zap(POS(':',line)); zap(POS(';',line));
  zap(POS('*',line)); zap(POS('{',line));
  IF line<>'' THEN BEGIN
    uppercase(line);
    IF name_count=0 THEN i:=1
    ELSE BEGIN
      i:=0; j:=1;
      WHILE (j<=name_count) AND (i=0) DO BEGIN
        IF line<names[j].name THEN i:=j;
```

```
        j:=j+1
      END;
      IF i=0 THEN i:=name_count+1
    END;
    FOR j:=name_count DOWNTO i DO names[j+1]:=names[j];
    name_count:=name_count+1;
    names[i].name:=line; names[i].number:=page_number
  END
END;

FUNCTION abort:BOOLEAN;
VAR c:CHAR;
BEGIN
  IF KEYPRESSED AND NOT aborted THEN BEGIN
    WRITELN; READ(KBD,c);
    REPEAT
      WRITE('Continue? '); REPEAT UNTIL KEYPRESSED;
      READ(KBD,c); c:=UPCASE(c); WRITELN(c)
    UNTIL c IN ['Y','N'];
    aborted:=c='N'; IF aborted THEN print('***ABORTED***','')
  END;
  abort:=aborted
END;

PROCEDURE include_file(line,title: string_type);
VAR input: TEXT;
PROCEDURE do_include;
PROCEDURE zap(i:INTEGER); BEGIN IF i>0 THEN DELETE(line,i,LENGTH(line)) END;
BEGIN
  IF line<>'' THEN IF UPCASE(line[1])='I' THEN BEGIN
    line:=line+' ?'; REPEAT DELETE(line,1,1) UNTIL line[1]<>' ';
    zap(POS(' ',line)); zap(POS('}',line)); zap(POS('*)',line));
    IF POS('.',line)=0 THEN line:=line+'.PAS';
    WRITE(^M,'    ',line);
    ASSIGN(input, line); (*$I-*) RESET(input); (*$I+*)
    IF IORESULT=0 THEN BEGIN
      title:=title+'  ('+line+')'; uppercase(title);
      WHILE NOT EOF(input) AND NOT abort DO BEGIN
        READLN(input,line); print(line,title); index_name(line,page_number);
      END;
      CLOSE(input)
    END
    ELSE BEGIN WRITE(' NO FILE'); DELAY(1000) END;
    WRITE(^M); CLREOL
  END
END;
BEGIN
  IF COPY(line,1,3)='(*$' THEN BEGIN DELETE(line,1,3); do_include END
  ELSE IF COPY(line,1,2)='{$' THEN BEGIN DELETE(line,1,2); do_include END
END;

PROCEDURE list_index(title:string_type);
VAR i,j,p:INTEGER;
BEGIN
  WRITE(^M); CLREOL; WRITE('INDEX...');
  IF side=2 THEN i:=101 ELSE i:=1;
  WHILE (i<name_count) AND NOT abort DO BEGIN
```

```
    WRITELN(printer);
    IF side<>2 THEN WRITE(printer,'   '); WRITELN(printer,title);
    WRITELN(printer);
    FOR j:=1 TO 50 DO BEGIN
      IF side<>2 THEN WRITE(printer,'   ');
      IF i<=name_count
      THEN WRITE(printer,' ':5,names[i].name:20,'   ',names[i].number:3);
      i:=i+50;
      IF i<=name_count
      THEN WRITE(printer,' ':10,names[i].name:20,'   ',names[i].number:3);
      WRITELN(printer);
      i:=i+1-50;
    END;
    i:=i+50; IF side>0 THEN i:=i+100;
    WRITE(printer,^L);
  END;
  WRITE(^M); CLREOL
END;

VAR input:TEXT; line,name,title:string_type; err:INTEGER;
BEGIN
  aborted:=FALSE;
  ASSIGN(printer,'LST:'); REWRITE(printer);
  WRITELN('PROGRAM LISTER');
  lines_printed:=-1; page_number:=1; name_count:=0;
  start_page:=1; side:=0;
  IF PARAMCOUNT>0 THEN BEGIN
    name:=PARAMSTR(1);
    IF PARAMCOUNT>1 THEN VAL(PARAMSTR(2),start_page,err);
    IF PARAMCOUNT>2 THEN VAL(PARAMSTR(3),side,err)
  END
  ELSE BEGIN
    WRITE('FILE: '); READLN(name);
    WRITE('STARTING PAGE: '); READLN(start_page);
    WRITE('SIDE (0,1,2): '); READLN(side)
  END;
  WRITELN('LIST: ',name,'  START: ',start_page:0,'  SIDE: ',side:0);
  print_on:=start_page=1;
  IF POS('.',name)=0 THEN name:=name+'.PAS';
  ASSIGN(input,name); (*$I-*) RESET(input); (*$I+*)
  IF IORESULT<>0 THEN WRITELN('NO FILE.')
  ELSE BEGIN
    title:='LISTING: '+name+'  '+time(FALSE)+'  '+time(TRUE);
    uppercase(title); WRITELN(title); WRITE(1:3);
    WHILE NOT EOF(input) AND NOT abort DO BEGIN
      READLN(input,line); print(line,title);
      include_file(line,title); index_name(line,page_number)
    END;
    CLOSE(input); finish_page;
    WRITELN('READY TO LIST INDEX.'); READLN;
    list_index(title);
  END;
  CLOSE(printer)
END.
```

```
| 11-Dec-84 |        Augment Engine (KL) #117 Console V2.610        | 15:33:07 |

      Load: KLBOOT          State: Stopped          Remote:Off
 MBRK=OFF        M[000067]=000005,,000004/000   DMA[00000030]=000000,,017645
 PC=000000,,000035  MA=400000,,005000  AR=450530,,230000 MEM=000000,,000000
 IR=000000,,017645   Q=000000,,000000  LP=002734       AC[00]=000000,,002400
      JMEM[0001]=12146      DEV=37  AMEMP=0335    AMEM[0016]=001600,,160016
  UBRK=OFF     MIC=17645   OBUS=450530,,230000          EOBUS=777777,,777777

 LC=ON    TC=OFF    Branching      MI=00316777777767376377744153023
 EEAL=0  EEFO=0   LDMA=0     IDISP=0     MWT=0  SAAFMA=1     PO=1  SP1=0 CRY=0
 ASRC=7  AFUN=3   ADST=7      ALU1=1   LDAR=1   JCOND=77
 MAPF=17  SPC=77  JADR=16774  ROT=77  MASK=77
 DEST=71  SP2=0  JCODE=01   ACSEL=5       D=30  CYLEN=04  IFQ=1  DFQ=1

| MMLOAD: UMEM[07070]=00316777777767376377744153023 |

Continue? N
!
! MMLOAD 7070
!

| 11-Dec-84 |        Augment Engine (KL) #117 Console V2.610        | 15:34:08 |

      Load: KLBOOT          State: Stopped          Remote:Off
 MBRK=OFF        M[000067]=000005,,000004/000   DMA[00000030]=000000,,017645
 PC=000000,,000035  MA=400000,,005000  AR=450530,,230000 MEM=000000,,000000
 IR=000000,,017645   Q=000000,,000000  LP=002734       AC[00]=000000,,002400
      JMEM[0001]=12146      DEV=37  AMEMP=0335    AMEM[0016]=001600,,160016
  UBRK=OFF     MIC=17645   OBUS=450530,,230000          EOBUS=777777,,777777

 LC=ON    TC=OFF    Branching      MI=00316777777767376377744153023
 EEAL=0  EEFO=0   LDMA=0     IDISP=0     MWT=0  SAAFMA=1     PO=1  SP1=0 CRY=0
 ASRC=7  AFUN=3   ADST=7      ALU1=1   LDAR=1   JCOND=77
 MAPF=17  SPC=77  JADR=16774  ROT=77  MASK=77
 DEST=71  SP2=0  JCODE=01   ACSEL=5       D=30  CYLEN=04  IFQ=1  DFQ=1

| MMLOAD: UMEM[07071]=00316777777767376377744153023 |

!
! MMLOAD 7070
! MMLOAD 7071
!
```

MACRO - 18 bits Adder
MICRO - 18 bits Adder

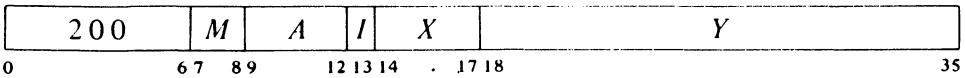
LOCATION			
3,000	1000	Move 10, 2000	200 400,, 002000 10 ← 3000
12,100	1001	Move 10,(1)2000	200 401,, 002000 10 ← 12,400
12,000	1002	Move 10, @2000	200 420,, 002000 10 ←
2,000	1003	move 10,@(1)2000	200 421,, 002000 10 ← 2000
	1004		0 — 0,, 0 — 0

2000		0 — 0,,	003000
3000	@	000020,,	004000
4000		0 — 0,,	0 12000
12000		0 — 0,,	0 12100
12100		0 — 0,,	0 02000
12200		000400,,	00 1000

1	0 — 0, 0 10000
4	7 — 7, 7 — 7
8	0 — 0,, 7 — 7

Mode	*Suffix*	*Source*	*Destination*
Basic		*E*	AC
Immediate	I	The word 0,*E*	AC
Memory	M	AC	*E*
Self	S	*E*	*E*, but also AC if *A* is nonzero

MOVE **Move**

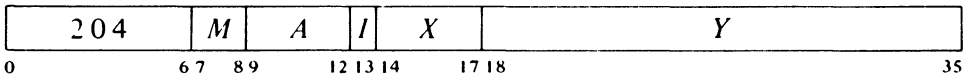


Move one word from the source to the destination specified by *M*. The source is unaffected, the original contents of the destination are lost.

MOVE	Move	200
MOVEI	Move Immediate	201
MOVEM	Move to Memory	202
MOVES	Move to Self	203

Notes. MOVEI loads the word 0,*E* into AC. If *A* is zero, MOVES is a no-op that writes in memory; otherwise it is equivalent to MOVE except that it writes in memory.

MOVS **Move Swapped**

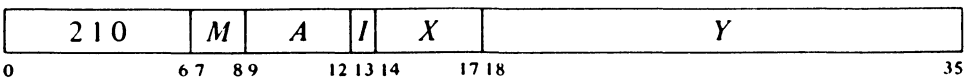


Interchange the left and right halves of the word from the source specified by *M* and move it to the specified destination. The source is unaffected, the original contents of the destination are lost.

MOVS	Move Swapped	204
MOVSI	Move Swapped Immediate	205
MOVSM	Move Swapped to Memory	206
MOVSS	Move Swapped to Self	207

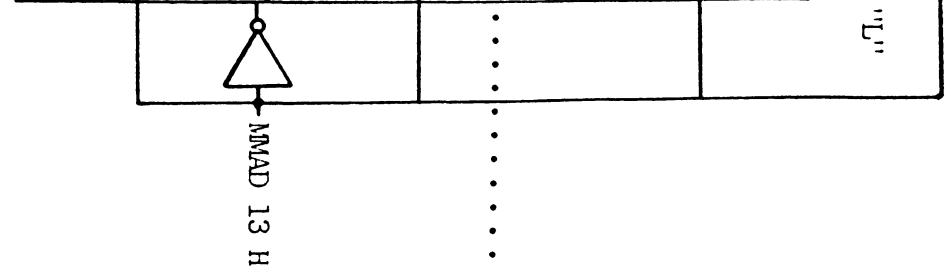
Notes. Swapping halves in immediate mode loads the word *E*,0 into AC.

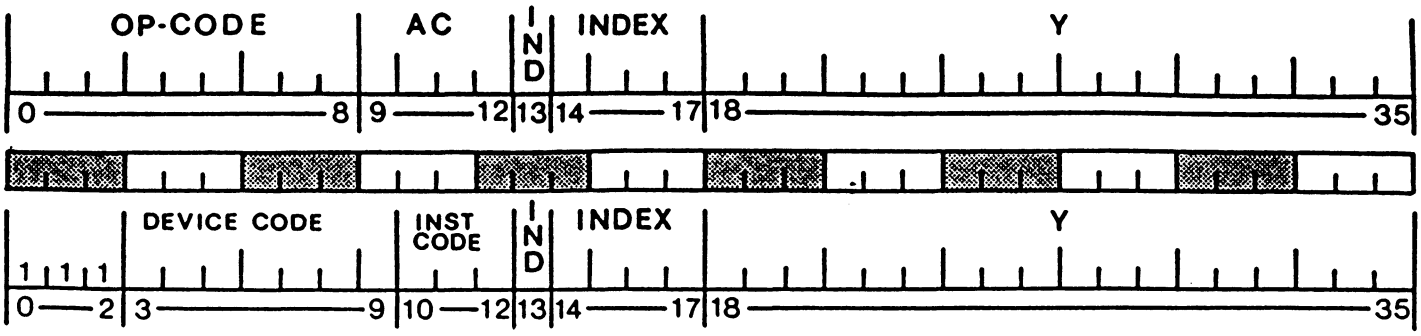
MOVN **Move Negative**

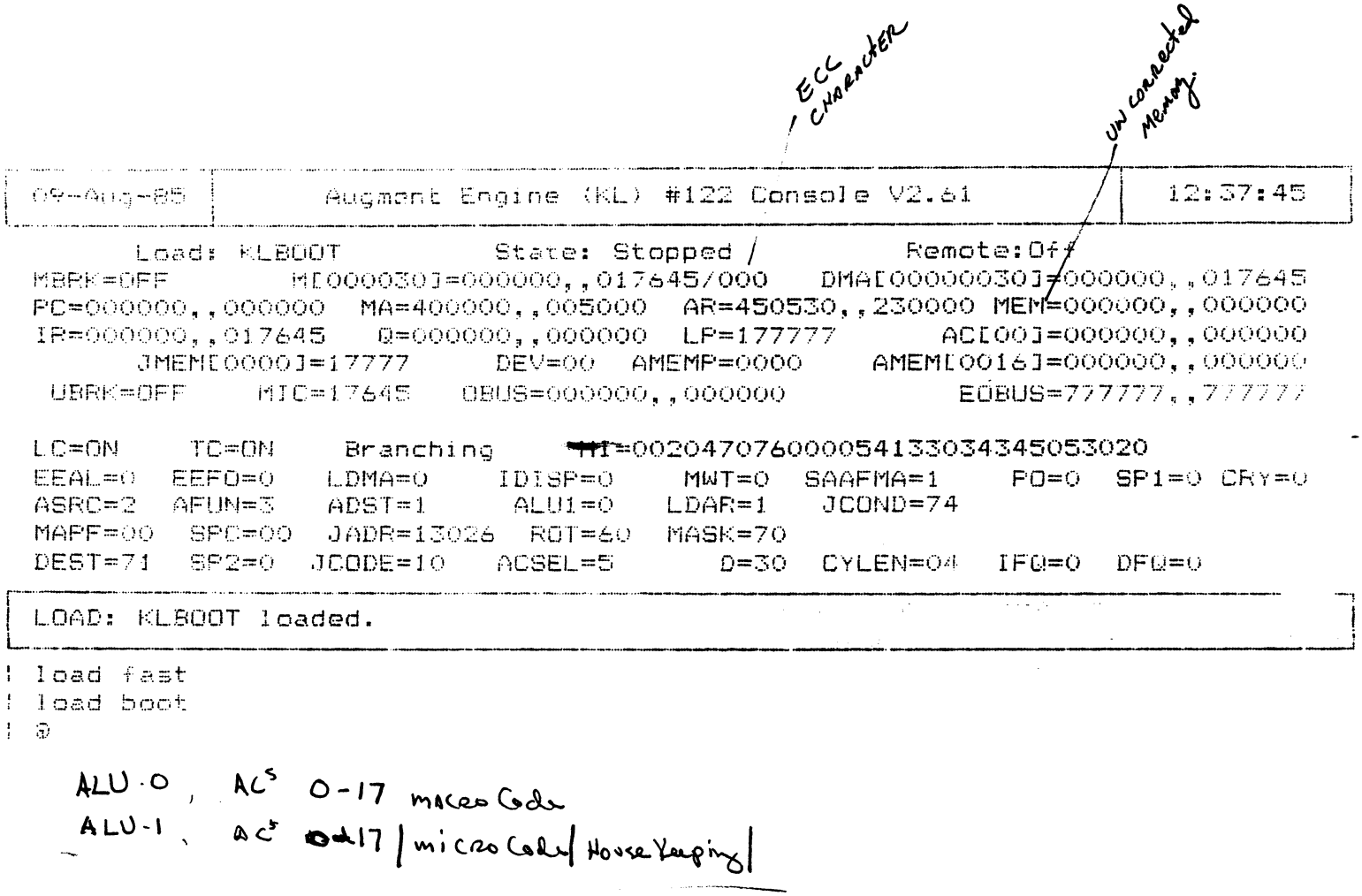


Negate the word from the source specified by *M* and move it to the specified destination. If the source word is fixed point -2^{35} (400000 000000) set the

	MMB				MMA			"R"
JADR 12,13 DEST[20,40]	13-46	13-35	13-24	13-13	9-46	9-36	9-18	9-9
JADR 8,9,10,11	15	15	15	15	19	11	11-18	11-9
JADR 4,5,6,7	17	17	17	17	13	13	13-18	13-9
JADR 0,1,2,3	19	19	19	19	15	15	15-18	15-9
MASK SIZE [10,4,2,1]	21	21	21	21	17	17		
MASK SIZE[40,20] ROT SIZ[2,1]	23	23	23	23	19	19		
ROT SIZ[40,20,10,4]	25	25	25	25	21	21		
SPC [10,4,2,1]	27	27	27	27	23	23		
~~IF, DF~~ CYLEN [2,1]	31	31	31	31	25	25	25-18	25-09
D [2,1] CYLEN[10,4]	33	33	33	33	27	27	27-18	27-09
D [40,20,10,4]	36	36	36	36	31	31	31-18	31-09
AGSEL [4,2,1] DEST [10]	40-46	40-35	40-24	40-13	33-46	33-36	33-18	33-09
IF DF					05-37 05-28	07-37 07-28		
BANK #	7	6	5	4	3	2	1	Ø
JCOND [4,2,1,R]	50-46	50-35	50-24	50-13	50-46	50-36	~~54-18~~	~~54-09~~
JCOND [20,10] SPC[40,20]	52	52	52	52	52	52	~~56-18~~	~~56-09~~
JCODE [10,4,2,1]	54	54	54	54	54	54	56-18	56-09
DEST [4,2,1] SPARE	56	56	56	56	56	56	54-18	~~54~~-09
MAPF [10,4,2,1]	58	58	58	58	58	58		
ALUDST [2,1] ALU1 SEL, LD AR	60	60	60	60	60	60		
ALUDST [4] ALU FUN[4,2,1]	62	62	62	62	62	62	62-18	62-09
ALU SRC[4,2,1] CARRY IN	64	64	64	64	64	64	64-18	64-09
AA from MA-MEM WAIT-PAR-SP1	66	66	66	66	66	66	66-18	66-09
IDISP RQ,LD MA, ENB EA LEFT	68-48	68-35	68-24	68-13	68-46	68-36	68-18	68-09
EA FROM OBUS								




OP-CODE
AC
IND
INDEX
Y
0
8
9
12
13
14
17
18
35
1 1 1
DEVICE CODE
INST CODE
IND
INDEX
Y
0
2
3
9
10
12
13
14
17
18
35




```
! load fast
! load boot
! @
```

ALU-0, AC's 0-17 macro Code

ALU-1, AC's 0-17 | micro Code | Housekeeping |

```
09-Aug-85 |   Augment Engine (KL) #122 Console V2.61   | 12:37:45

        Load: KLBOOT          State: Stopped          Remote:Off
  MBRK=OFF        MI[000030]=000000,,017645/000   DMA[00000030]=000000,,017645
  PC=000000,,000000  MA=400000,,005000  AR=450530,,230000 MEM=000000,,000000
  IR=000000,,017645   Q=000000,,000000  LP=177777       AC[00]=000000,,000000
        JMEM[0000]=17777      DEV=00  AMEMP=0000     AMEM[0016]=000000,,000000
   UBRK=OFF     MIC=17645   OBUS=000000,,000000          EOBUS=777777,,777777

  LC=ON     TC=ON     Branching     MI=00204707600005413303434505302O
  EEAL=0  EEFO=0   LDMA=0     IDISP=0     MWT=0  SAAFMA=1     PO=0  SP1=0 CRY=0
  ASRC=2  AFUN=3   ADST=1     ALU1=0    LDAR=1    JCOND=74
  MAPF=00  SPC=00  JADR=13026  ROT=60  MASK=70
  DEST=71  SP2=0  JCODE=10   ACSEL=5       D=30  CYLEN=04  IFQ=0  DFQ=0

 LOAD: KLBOOT loaded.

! load fast
! load boot
! @
```