MDOS PROGRAMMING REFERENCE

Current Versions:

MDOS 5.0
TASM 3.5
LDR 4.1

User Manual
v3.5

Index

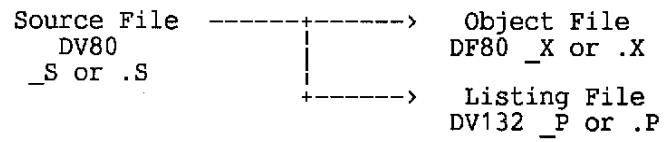
1. Introduction

TASM is an Assembler for the TMS99xx series of TI processors that runs on several machines, including:

9640 Geneve
IBM PC or Compatible
AMIGA

Actually, since TASM is written entirely in C, it can be ported to any machine supporting a good C compiler. In the Geneve version, TASM is generated using the TIC C Compiler.

TASM will take an input source file that contains Assembly source statements, and create from that file an Object File and a Listing File, i.e.:



The listing file can be printed; the object file is compatible with the standard loaders available for the TI-99/4A and Geneve.

Features of TASM are quite extensive, including:

- o Two-Pass Assembler
- o Extensive Symbol Table Size (1000 Symbols on Geneve)
- o Conditional Assemblies
- o MACRO Facility
- o Support for TMS9900, TMS9995, & TMS99000 Instruction sets
- o Versions available for several different machines
- o Extensive error checking
- o Fast hash table storage of symbols
- o Supports special "link chains" for GENEVE paging

TASM has its roots in a99, a cross-assembler originally developed at the University of Arizona by Dr. Bruce Wampler. This assembler was converted to the present TI-99/4A cross-assembler by Alan Beard (LGMA Products) and Ron Lepine (Moderator of the TI Conference on Byte Information Exchange). TASM was recoded to operate specifically with TIC, the Full-C compiler for the Geneve and TI-99/4A; the resulting program is TASM, this assembler.

TASM gets around several problems with using other assemblers with TIC, including:

- o Runs in MDOS Native Mode (not GPL mode)
- o Two-Pass, so backward REF chains are allowed
- o Special Geneve Paging chains supported (-X Option)
- o Supports 31-character external names
- o MACRO capability
- o Extensions

TASM does NOT support the debug code generation mode of GenASM. Also, because TASM is a two-pass assembler (GenASM is one-pass) and because it is written in C, it is slower than GenASM.

==========

2. Running TASM

Regardless of what machine environment you are using, TASM is always started in the same manner:

```
TASM [+/-options] sourcefile
```

A source file on the GENEVE is a normal Display/Variable 80 file. Under MSDOS and Amiga, it is a normal TEXT type of file. The source file must have an extension of:

```
MYARC GENEVE:       _S
MSDOS, AMIGA:       .S or .s
```

TASM optionally produces as output the following files:

```
Object File:        _X, .X, or .x
Listing File:       _P, .P, or .p
```

TASM allows you to specify parameters that allows options to be turned ON and OFF for the subsequent assemblies. These options are specified with a + to turn ON the option, a - to turn OFF the option. The options are:

```
O -  generate object file _X or .X                    (default = yes)
L -  generate listing file _L or .L                   (default = no )
```

R -	use register definitions (e.g. R1 instead of 1)	(default = yes)
C -	generate compressed object	(default = yes)
G -	allow TMS9995 instructions	(default = yes)
M -	generate long name (>6 characters) references	(default = yes)
X -	generate special GENEVE X-REF chains	(default = no)
H -	allow TMS99105A & TMS99110A extensions	(default = no)

2.1 Examples

For example, to assemble a source file, called A99P1_S, you could type:

```
TASM A99P1_S
```

This will create the file:

```
A99P1_X
```

and will NOT create a listing file, will assemble using register definitions, TMS9995 instruction set, long name references, no special REF chains, and no TMS99XXX instructions.

This is equivalent to:

```
TASM -LXH +ORCC A99P1_S
```

TASM is a two pass assembler. At the end of the first pass, it displays the value of the last memory location used. It displays the number of errors detected at the end of the second pass.

2.2 Errors

TASM will detect most syntax errors. Offending lines are displayed on the terminal screen, followed by the error message. In addition, the offending lines are marked in the '.P' listing file (if generated).

2.3 Instruction set

TASM supports the complete instruction set as described in the TI-99/4A Editor/Assembler User's Guide.

If the +G parameter is chosen (default=yes), then four additional opcodes are recognized for the TMS9995 microprocessor, as follows:

```
MPYS -  Signed Multiply
DIVS -  Signed Divide
LST  -  Load Status Register
LWP  -  Load Workspace Pointer
```

If the +H parameter is chosen, then 24 new opcodes for the TMS99105A/ TMS99110A processors, as well as the four additional TMS9995 instructions are recognized:

```
BIND - Branch Auto-Increment
BLSK - Branch and Link through Stack
TMB  - Test Memory Bit
TCMB - Test and Clear Memory Bit
TSMB - Test and Set Memory Bit
AM   - Add Double
SM   - Subtract Double
SLAM - Shift Left Arithmetic Double
SRAM - Shift Right Arithmetic Double
LDS  - Long Distance Source
LDD  - Long Distance Destination
```

```
AR   - Add Real
SR   - Subtract Real
MR   - Multiply Real
DR   - Divide Real
LR   - Load Real
STR  - Store Real
CIR  - Convert Integer to Real
CRI  - Convert Real to Int
NEGR - Negate Real
CRE  - Convert Real to Ext Int
CER  - Convert Ext Int to Real
CR   - Compare Reals
MM   - Multiply Multiple
```

Refer to the TI TMS9995 Data Manual, and the TMS99105A/TMS99110A manuals for a description of the new instructions..

2.4 Operand Addressing Modes

TASM supports all TI-990 addressing modes. The symbols R0-R15 are predefined for registers (if the -R option was not chosen). Any symbols used as operands may use the rules for labels described below. In addition, expressions as described below may also be used. For example, "CLR @VAR+4" is a legal operation which refers to the contents of the 2nd word following the label VAR. (VAR+0 is the word at VAR, VAR+2 is the 1st word, VAR+4 the 2nd following word.)

2.5 Upper and Lower Case

TASM is totally case insensitive. Thus 'ABC' is treated as being identical to 'abc'. This is true for opcodes, operands, and labels.

2.6 Labels

TASM allows labels to be up to 128 characters long. Labels can begin with any character except: 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, >, @, (,), *, ;, -, ', ", comma, space,tab, or :. Following characters may use 0 to 9 and in addition. Labels are defined by appearing in the label field, which is a symbol beginning in column 1. Labels are terminated by a space or tab, as well as a colon. If a label is followed by a colon, it need not begin in column 1, but must be the first symbol on the line. The following are valid labels:

```
SHORT:
A_LABEL_WITH_UNDERSCORE
TEMP$
```

2.7 Comments

Comment lines are indicated by the first character in the line being a semi-colon or an asterick. Comments may be included on the same line as an instruction by preceding them by a blank or a semi-colon.

There are certain cases were you must precede the comment with a semi-colon. In particular, if the first character of your comment beings with a comma (,) or colon (:), then it must be preceded by a semi-colon.

2.8 Expressions

More general expressions are allowed in TASM. The operations +, -, *, /, | (logical or), and & (logical and) are supported. Expressions may not use parens, and are evaluated strictly left to right. (* does not have precedence

over +, for example.) In general, expressions may use any symbol defined anywhere in the program, as well as constants. Hex constants begin with >.

Restrictions: the current version of TASM does not allow spaces between expression symbols and operators. Symbols used in an expression in an "EQU" directive must not be forward references.

2.9 Psuedo-Instructions

AORG <address>
==========

The AORG statement works exactly the same as in the 99/4 assembler. If AORG is omitted, the assembly is assumed to be relocatable as if the statement RORG 0 preceded any object code generation.

RORG <address>
==========

The RORG statement works exactly the same as in the 99/4 assembler. If RORG is omitted, TASM defaults to a starting relocatable address of >0000.

You should not mix AORG mode with RORG mode.

BSS <blocksize>
==========

This reserves a block of of memory of <blocksize> bytes. If <blocksize> is odd, it will NOT be rounded up to an even size.

DATA <valuelist>
==========

This reserves a word of storage for each item in the list. Items are separated by commas. Unlike the UNIBUG assembler, you may have several values following the DATA statement, each taking one word of memory.

END <label>
==========

Denotes the end of the program. Every program must have an END statement.

The <label> field is optional, and if specified, generates an object tag '1' for absolute entry point, or an object tag '2' for relative entry point, and denotes the entry point into the program for the loader.

<label> EQU <value>
==========

This directive will give <value> to the symbol <label>. The value may be any expression, but must not use any symbols defined after the EQU statement (a forward reference is not allowed, in other words). It is possible, however, to use the <label> of the EQU anywhere in the program.

TEXT /string/
==========

This reserves storage for the given string. The string is delimeted

by two matching characters, e.g.:

"string" or /string/ or 'string'

If double single quotes are included in the string, they will be interpreted as a single quote, i.e.:

'don''t do it'

BYTE <valuelist>
==========

This reserves a byte of storage for each item in the list. Items are separated by commas. Note it is possible to leave the location counter odd with this statement (as with the TEXT directive). This allows you to freely mix TEXT and BYTE expressions.

DORG <address>
==========

The DORG statement works exactly the same as in the 99/4 assembler. It differs from AORG in that object code generation is suppressed.

DXOP <symbol>,<term>
==========

The DXOP statement allows you to define an XOP; exactly the same as defined in the 99/4 assembler, page 233 of E/A manual. The DXOP actually defines internally a MACRO, the following two code segments are similar:

```
SYSC  MACRO
      XOP   %1,0
      ENDM

      ...and...

      DXOP  SYSC,0
```

You can then use the defined XOP in following statements, i.e.:

```
      SYSC  @VID
VID   DATA  6
```

DEF <label>[,<label> ... [,<label>]
==========

Provides an external definition. Most useful to indicate an entry point into the program for the load and run option. Multiple labels are allowed per one DEF statement. Each label must be sepereated by a comma, and the DEF list must be terminated by a blank or tab.

IDT <ident>
==========

An up to six character program identifier can be provided. Merely includes it in the object file.

PAGE
====

Causes a page eject on the listing file.

TITL <up-to-50-character-title>
==========

An up to 50 character title line can be included within quotes. This line will be put at the top of each listing page.

REF <label>[,<label>] ... [,<label>]
==========

Provides an external reference. The <label> field must be specified in another program using the DEF statement. Multiple labels are allowed per one REF statement. Each label must be sepereated by a comma, and the REF list must be terminated by a blank or tab.

RT
==

The RT psuedo instruction provides a convenient means of executing a standard TI return from subroutine. It actually generates the same code as the instruction:

B *R11

COPY
====

The COPY psuedo instruction provides a means for including source files with the compilation. This is useful say, to create a small module which "INCLUDES" or "COPIES" other modules in its address space. It is also useful to COPY in common equate files (e.g. what was created for the MYARC GENEVE).

The SYNTAX of the command is:

COPY "<filename>"

where <filename> is the DOS full file name (not the TI-99 file name). For example, on the Amiga, an equates file called "vid.eq9" exists in directory "df2:equates/", so the COPY directive would be:

COPY "df2:equates/vid.eq9"

A COPY'd file may not contain another COPY command, or an error will result.

IFxx <expression> / ELSE / ENDIF
==========

The IFxx/ELSE/ENDIF statements allow you to create modules with conditionally assembled code. The <expression> is evaluated, and if the expression is non-zero, then the following code is assembled. If the expression is zero, then the following code is not assembled, until an ELSE or ENDIF statement is encountered.

This is especially useful in writing code for two different machines or operating systems. Only one source module need be maintained, and the proper code for each operating system can be generated by merely setting an equate in the program or in an equate file.

For example, to generate code for MDOS or GPL, one could create a flag

set to one or zero, e.g.:

```
MDOS    EQU  0              ; MDOS=1, GPL=0
```

and then generate two sets of code, i.e.:

```
        IFEQ  MDOS
        XOP  @DKEYOP,0      ; In MDOS, call keyboard XOP
        ELSE
        BLWP @KSCAN         ; In GPL, call keyboard scan
        ENDIF
```

Note that if the MDOS flag is set to non-zero, then the XOP code will be generated, else the KSCAN routine will be called. The ELSE condition is optional, however the ENDIF statement must be present.

The following conditionals, when TRUE, will cause any following code to be assembled up to any following ELSE or ENDIF statement:

```
IF      - if expression NOT 0
IFEQ    - if expression 0
IFNE    - if expression not 0 (same as IF)
IFLT    - if expression less than 0
IFLE    - if expression less than or equal to 0
IFGT      if expression greater than 0
IFGE    - if expression greater than or equal to 0
```

==========

3. Macros

A macro is a facility for text replacement. A single line of source assembly code that uses a macro will be replaced by possibly many lines of assembly code defined in the body of the macro definition. Thus, to use a macro, it must first be defined, then "called" from within the assembly language program. While a macro may be considered to be somewhat like a subroutine, there is a major difference. When a subroutine is called, the PC changes value, and the program begins executing code in a different place in memory. When a macro is called, the code from the macro body is actually substituted at the place of the call. This substitution process is called expanding the macro. Macros may have parameters that can be substituted within the body when the macro is expanded. Figure 1 illustrates a macro definition and call:

```
          SOURCE CODE                          EXPANDED CODE
          AORG    >300                         AORG    >300
JLSEQ     MACRO           ; "Jump <="
          JLT     %1      ; %1 is param
          JEQ     %1
          ENDM            ; end
;                                  ;
          C       R1,R2   ; compare            C       R1,R2
          JLSEQ   LBL1    ; jmp <=             JLT     LBL1    [expansion]
                                               JEQ     LBL1
          A       R1,R2                        A       R1,R2
LBL1      ...                          LBL1    ...
```

Figure 1. A Macro Example

This macro allows a jump on arithmetic less than or equal using one line of assembly source code. It is expanded at assembly time to two actual instructions, JLT and JEQ, using the parameter provided in the call: LBL1. The macro definition starts with the directive MACRO and ends with the directive ENDM. Parameters in the definition are denoted with a %, and provided in the call just like an ordinary operand.

3.1 Macro Definition

A macro is defined with an initial line of the form:

```
MACRONAME   MACRO.
```

The macro name may be any standard label, and must be present with the MACRO directive. When assembling, the macro definition table is checked last. Thus, it is not legal to define a macro with the same name as a valid instruction such as MOV or INC since TASM will match the regular MOV instruction first and never get to the macro definition table.

3.2 Macro body

The body of a macro consists of any number of legal TI assembly language instructions. There must not be calls to other macros within the body of a macro definition. Nested macros are not allowed. The macro is ended with the ENDM directive. Macros definitions appear in the .P file with dashes in place of the usual address/value fields.

3.3 Macro Formal Parameters

Any macro definition may use up to 9 parameters. Parameters are implicitly defined, and can appear in the body of the macro definition as the symbols %1 to %9. Any given parameter may be used as often as needed within the macro body. Parameters will be expanded whenever they are found within the macro body, regardless of the context. Thus, you may not use an unprotected % in a macro body (See Protected values, below). Parameters must be used in order from %1 to %9.

3.4 Local Labels

It is sometimes necessary to use labels within a macro definition. If a standard label were used, it would result in a multiple definition of a label error whenever the macro was called more than once. TASM allows local labels to be defined within the body of a macro that will be expanded to unique labels whenever the macro is called. Local labels have the form ?x.

The ? denotes a local label, and the x is any letter from a to z that the programmer chooses. Thus, any one macro may have up to 26 local labels. Labels may be re-used in different macro definitions - ?a can be used in more than one macro definition. When local labels are expanded, the ?x portion remains constant, and two additional letters are automatically appended to generate a unique label of the form ?xyz. Each time any macro is called, the label generator produces a new label, whether labels are used or not. Thus, the very first macro call will have generated labels of the form ?xaa, the second macro call ?xab, and so on. Thus up to 676 macro calls may appear in any one program -- far more than will ever be needed for this class. The ? is a special character, just like the % and may require protection.

3.5 Protected Values

Sometimes you may want to protect portions of the macro body. For example, you may have a string with a ? or % within the macro body definition. If such values are left as is, they would be confused with parameters or local labels. Square brackets ([and]) may be used to protect any portion of a macro body. Thus a string text[?] would be expanded to text? with no label expansion of the ?.

3.6 Invoking a Macro

A macro is invoked by using the macro name followed by any parameters. A macro call will look just like a regular instruction in the source code, and may be followed by up to 9 operands. The operands in the order given are then

substituted for the formal parameters %1 to %9 in the macro body as the macro is expanded.

3.7 LIST Directive

The LIST assembler directive is used to control the listing of generated code for macro calls and TEXT directives. These directives affect only the .P file, and in no way change the code generated in the .x object code file.

```
LIST    TEXT
LIST    NOTEXT  ; default
```

The TEXT and NOTEXT options control how generated code for TEXT directives is listed in the .P file. With NOTEXT, the default, the starting address of the string is given, along with the number of bytes the string takes in the value field. Using the LIST TEXT option has TASM list the address and value of each word of data generated by the TEXT directive in the .P file. The NOTEXT option makes the .P file much smaller.

```
LIST    MOBJECT
LIST    MNEVER
LIST    MONCE   ; default
LIST    MALWAYS
```

These LIST options control the .P expansion of macro calls. The MOBJECT option is the defaulted option and causes the MACRO not to be expaned in the listing, only the original source line and the resultant object are listed. The MNEVER option forces TASM to never list the expanded code produced by a macro call. The default MONCE option will have TASM list the full expansion of a macro the first time it is called. Subsequent calls to the same macro will not be expanded in the listing. The MALWAYS option will have TASM show the full macro expansion every time a macro is called. This option is most useful for debugging to make sure the parameter expansion and label generation works as expected. Comments present in the original macro definition will be removed from the expanded code. Expanded code is also formatted on tab stops. This automatic provision means that strings may have some spaces replaced with tab stops. This can be avoided by being sure string defined with TEXT directives inside the macro body use double quotation marks:

```
"STRING".
```

Other forms of the LIST statement:

```
LIST
LIST "file_name"
```

These directives serve two distinct purposes. The LIST statement with no arguments restores output to the listing file which had been previously disabled using the UNL (unlist) command. The list statement with a file name enclosed in double quotation marks will cause the listed output to be redirected from the file specified on the -l command line switch to the file_name specified.

==========

4.0 Sysgen Limits

The TASM assembler has certain limitations that are changed only during sysgen of the TASM program. These limits are currently set to:

	MSDOS/Amiga	Geneve
Symbol Table Space	12000 BYTES	10000 BYTES
Maximum # of Symbols	2000	1000
Maximum # of Macros	80	25
Maximum Chars in a Macro	250 BYTES	250 BYTES

5.0 Include (COPY) Files

Include files or COPY files are used extensively in most modern operating systems to provide the user with a symbolic way of interfacing to other programs or operating system calls. The MYARC DOS package provides a very structured method of interfacing to system services, the XOP interface. COPY files are provided to give the user a symbolic mechinism of using the XOP interface.

For example, the following assembly statements within a program set the video mode to graphics mode 6:

```
            COPY "HDS1.INCLUDE.VIDEO_I"

            LI   R0,SETVID
            LI   R1,GRAPH6
            XOP  @DVIDXP,0

    DVIDXP  DATA VIDXOP
```

This provides readable source. The following source is equivalent to the previous code, but does not use symbols and therefore is not very readable:

```
           LI   R0,0
           LI   R1,8
           XOP  @SIX,0

   SIX     DATA 6
```

The TASM package includes a number of COPY files that provide definition of the Geneve XOP Libraries. For a definition of what these libraries are and how they work, the most definitive document is contained within Paul Charlton's GenASM program. The 9640 Windows library is described in Beery Miller's 9640 Windows user manual. The video library is currently only defined as postings on various BBS's, and has non-working sections.

The header files provided are:

```
            keyboard_i  :     Keyboard Library
            video_i     :     Video Library
            memory_i    :     Memory Library
            io_i        :     Input/Output Library
            utility_i   :     Utility Library
            math_i      :     Math Library
            windows_i   :     9640 Windows Library
```

These files can be used by placing the files in an "INCLUDE" directory. For example, suppose you place all six files in the "HDS1.INCLUDE" directory on your hard disk. You can then utilize the header files you need by placing the statement:

```
            COPY "HDS1.INCLUDE.FILENAME_I"
```

in your Assembly program.

The following example program sets the video mode to text mode (40 column mode) and writes a string of text to it:

```
       COPY "HDS1.INCLUDE.VIDEO_I"
       DEF  START
START  EQU  $
       LI   R0,SETVID
       LI   R1,TEXT1
       XOP  @DVIDXP,0
*
```

```
       LI   R0,WRITET
       LI   R1,STRING
       XOP  @DVIDXP,0
*
       BLWP @0                    ; EXIT
*
STRING TEXT 'Set Video Mode to 40 Columns'
       BYTE >0D,>0A,>00
       EVEN
DVIDXP DATA VIDXOP
       END
```

6. Known Bugs/Deficiencies/Differences

a. The following code acts differently if assembled under TIC rather than the standard TI assembler:

```
       RORG  0
ADDRS  EQU   4
NEWADR EQU   $+ADDRS
```

The symbol "NEWADR" is calculated as an absolute symbol. In the TI assembler, it is treated as a relative symbol.

b. Under unknown circumstances, occasionally the disk directory that is being used for assemblies will "wipe-out", and be not accessable to later disk operations (a DEVICE ERROR results). I wonder if this is related to MDOS hard disk wipeouts in general.

The following was changed in version 3.5:

TASM - Changes for Version v3.5

1. The problem with the "BYTE" directive being first has been fixed.

2. The startup library changed to have a control ^c handler (you can now control ^c out of an assembly).

3. fputc() references in the source were changed to putc() (functionally identical).

a.l.beard 29-Feb-1992

==========end of document==========

LDR v4.1 Release Notes 92/09/10

LDR v4.1 is fuctionally identical to LDR v4.0 as released with the TICRUN package. The documentation for LDR contained in that package is still applicable. The only change in v4.1 is the addition of code to handle the "D" (byte data) tag generated by Paul Charlton's GenASM assembler. It is now possible to assemble TIC object files with GenASM provided that TIC was invoked with the "-a" option to produce GenASM-compatible output.

LDR v4.1 is an update to TICRUN. The fairware terms and conditions which apply to TICRUN also apply to LDR v4.1.

Clint Pulley

MEMORY

```
*****************************************************************************
=
**
**    Filename:            geneve_i
^^    Release:             version 1.5
**    Date:                26/July/1991
**
**    This file contains miscellaneous definitions for the 9640 GENEVE
**    that are hardware dependent.
**
**    Copyright 1991 by LGMA Products
**
**    Revision History:
**
**    26/Jul/91       Initial Release
**
WSP     EQU  >F000              WORKSPACE OFFSET (USUAL USER WORKSPACE)
CLOCK1  EQU  >F130              REAL - TIME CLOCK
VDPOR0  EQU  >F100              9938 CHIP PORT 0
VDPOR1  EQU  >F102              9938 CHIP PORT 1
VDPOR2  EQU  >F104              9938 CHIP PORT 2
VDPOR3  EQU  >F106              9938 CHIP PORT 3
BANKCD  EQU  >8E00              BANK CODE FOR VDP BANKING
*
*****************************************************************************
```

```
*******************************************************************************
=
*
* MEMORY MANAGEMENT ROUTINES:
*       XOP @seven,ZERO         seven is the system routine code for memory **
                                management.
*                               ZERO is the system XOP code
*
*       OPCODE : FUNCTION
*       00      -  return number of free pages in system
*       01      -  get    # pages @ local page #, speed
*       02      -  return # pages @ local page #
*       03      -  map local page # @ execution page
*       04      -  get address map (execution address, size of area
*                      to build list in)
*       05      -  declare shared pages (type,# pages,local page #)
*       06      -  release shared pages (type)
*       07      -  get shared pages (type, local page #)
*       08      -  size of shared page group (type)
*       09      -  move memory from LOCAL1 to LOCAL2 With LENGTH
*
* errors :
*       00  no error!
*       01  not enough free pages
*       02  can't remap execution page zero
*       03  no page at LOCAL address
*       04  user area not large enough for list
*       05  shared type already defined
*       06  shared type doesn't exist
*       07  can't overlay shared and private memory
*       08  out of table space (actually, OS should request more space,
*              but we are going to staticly allocate a table because
*              we don't have a lot of time)
*
*******************************************************************************
* REGISTER USAGE
*
*       in all cases, R0 of caller's WS has the opcode zero through nine
*       also, user's equal flag is set by contents of error flag
*
*  OP#0      IN:           nil
*            OUT:          R0=error code
*                          R1=number of free pages in system
*                          R2=number of fast free pages in system
*                          R3=total number of system pages
*
*  OP#1      IN:           R1=number of pages to get
*                          R2=local page address
*                          R3=speed flag      non-zero => fast
*            OUT:          R0=error code
*                          R1=# of pages actually fetched
*                          R2=# of fast pages actually fetched
*
*  OP#2      IN:           R1=# of pages to return to system
*                          R2=local page address
*            OUT:          R0=error code
*
*  OP#3      IN:           R1=local page #
*                          R2=execution page #
*            OUT:          R0=error code
*
*  OP#4      IN:           R1=execution address
*                          R2=size of area for map
*            OUT:          R0=error code
```

```
*
*  OP#5      IN:          R1=# of pages to be declared shared
*                         R2=local page address
*                         R3=type number
*            OUT:         R0=error code
*
*  OP#6      IN:          R1=type
*            OUT:         R0=error code
*
*  OP#7      IN:          R1=type
*                         R2=local page # for start of shared area
*            OUT:         R0=error code
*
*  OP#8      IN:          R1=type
*            OUT:         R0=error code
*                         R1=number of pages in shared group
*
*  OP#9      IN:          R1= >00 <MSB of 24 bit local address DESTINATION>
*                         R2= <LSWord of 24 bit local address  DESTINATION>
*                         R3= >00 <MSB of 24 bit local address SOURCE>
*                         R4= <LSWord of 24 bit local address  SOURCE>
*                         R5= byte count
*
*******************************************************************************
**
*  XOPS for use by OS only (not accessible to user task)
*
* opcode #A : page get
*             in:         r0=>000A
*                         r1=page number to get,> 255 means first available
*                         r2=speed flag          <>  0 means fast
*             out:        r0=error code
*                         r1=pointer to node
*
* opcode #B : add page to free pages in system
*             in:         r0=>000B
*                         r1=page number
*             out:        r0= error if no free nodes available
*
* opcode #C : add a node to the list of free nodes
*             in:         r0=>000C
*                         r1=pointer to node
*             out:        no errors
*
* opcode #D : link a node to the specified node
*             in:         r0=>000D
*                         r1=pointer to node
*                         r2=pointer to node to link to
*             out:        no errors
*
*  opcode #E : get address map  (system)
*             out:        R0= count of valid pages
*                          system >1F00 has map
*                          system >1FFE has count of pages
*
================================================================================
```

==========

==========
= memory management:
==========

definitions:

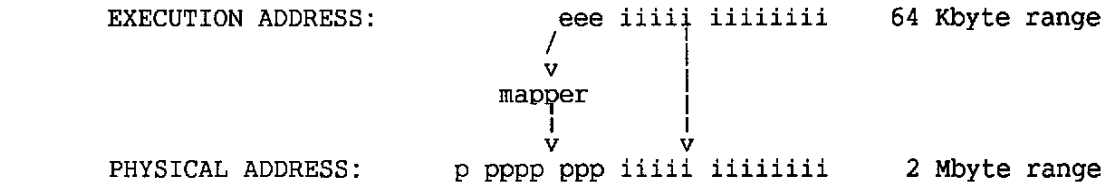
page: 8k bytes of memory, addressed by the 13 least significant address bits

execution page: these are numbered 0 to 7, indexed solely by the
3 most significant bits of the cpu's address

physical page: these are numbered from 0 to 255

function of the mapper: to translate an execution page into a physical page

example:



"eee" is the execution page
"pppppppp" is the physical page

++++++++++

in our case:

local page = virtual page
virtual addressing: a method of allowing a task to access more memory than
is directly accessible through the address bits
of the CPU. allowing a task to own more physical pages
than the 8 execution pages which are directly accessible
through the mapper

in the mapping of MDOS, the execution pages for a task are a subset of the virtual pages belonging to the task.

for each task, MDOS maintains a list of physical pages which can be used by the task. If the pages in this list are numbered, beginning with 0, the number assigned to each position in the list is the same as the virtual page number within the task. The list is allowed to have holes it, which correspond to no physical page.

An example of the virtual page list maintained by MDOS for a task with 128k of memory:

physical page numbers:
>3f >3e >3d >b8 >b8 >b8 >b8 >3c >3b >3a >39 >38 >37 >36 >35 >34

virtual (local) page numbers:
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

```
          <--- HOLE ----->  (physical page >b8 does not correspond
                             to any device which can be located in the
                             PE BOX )
```

the virtual address is:

```
      >2000 * (virtual page number) + ( index into page <13 bits> )
```

note: the virtual page list is limited to 256 bytes in length. therefore, no task may have more than 2 Mbyte of virtual memory

since this task has a hole containing 4 pages, it really only uses 96k of physical memory, even though it has 128k of virtual memory.

*** this example is the basis of the examples which follow ***

memory mangement opcodes:

++

#0 this returns the number of unassigned physical pages in the system

++

#1 this routines allows you to fill holes in your virtual page list, it will also create a new hole if the first page you are allocating has a higher virtual page number than any which have been previously assigned.

in the above example, if this routine were passed the arguments:

```
r1=2                    # of pages to get
r2=4                    Virtual page #
r3=don't care
```

- then the routine would take unassigned physical pages (if available) and assign them to virtual pages #4 & #5 creating a virtual page map like:

physical page numbers:

```
>3f >3e >3d >b8 >33 >32 >b8 >3c >3b >3a >39 >38 >37 >36 >35 >34
```

virtual (local) page numbers:

```
 0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15
          |               |
        <hole>          <hole>
```

the routine could create a new hole if you gave it arguments like:

```
r1=1
r2=17
r3=don't care
```

results:

physical page numbers:

```
>3f >3e >3d >b8 >b8 >b8 >b8 >3c >3b >3a >39 >38 >37 >36 >35 >34 >b8 >33
```

virtual (local) page numbers:

```
 0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17
          <---- hole ------>                                    |
                                                              <hole>
```

notice that this routine only _fills_ holes, it does not assign
a new physical page to a virtual page which is already assigned

example:

r1=5
r2=1
r3=don't care

results:

physical page numbers:
>3f >3e >3d >33 >32 >31 >b8 >3c >3b >3a >39 >38 >37 >36 >35 >34

virtual (local) page numbers:
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
    <--- 5 pages ---> |
    <had><new pages ><hole>

notice that the arguments asked for 5 pages, but only 3 were actually
assigned, since two of the target pages had already been assigned

++

#2 this routine releases pages which have been assigned to the
    virtual address space of a task, thus creating holes in the list
    it also ensures that there is no hole at the end of the list which
    is not followed by an assigned page
    note: you can not release page #0 (it isn't really owned by the task,
          it belongs to MDOS and is also the "header" page for the task)

    arguments:
    r1=9
    r2=2

results:

physical page numbers:
>3f >3e >b8 >b8 >b8 >b8 >b8 >b8 >b8 >b8 >b8 >38 >37 >36 >35 >34

virtual (local) page numbers:
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

note that some of the pages released were already unassigned, this is
quite ok...

    arguments:
    r1=10
    r2=12

physical page numbers:
>3f >3e >3d >b8 >b8 >b8 >b8 >3c >3b >3a >39 >38

virtual (local) page numbers:
 0 1 2 3 4 5 6 7 8 9 10 11

the list was truncated, since all of the pages at the tail of the list
were unassigned. Also note that we really told it to release
pages 12 to 21, but we only had pages up to 15 to begin with...this is
ok.

```
        arguments:
        r1=12
        r2=2

physical page numbers:
>3f >3e >b8 >b8 >b8 >b8 >b8 >b8 >b8 >b8 >b8 >b8 >b8 >b8 >35 >34


virtual (local) page numbers:
 0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15
        <--- giant hole ------------------------------>

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#3      map a virtual page into the execution (cpu visible) address space

        arguments:
        r1=virtual page number (index into list of physical pages)
        r2=execution page number (index into the mapper)

physical page numbers:
>3f >3e >3d >b8 >b8 >b8 >b8 >3c >3b >3a >39 >38 >37 >36 >35 >34

virtual (local) page numbers:
 0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15

mapper:

>3f >b8 >b8 >b8 >b8 >b8 >b8 >b8


example arguments:

        r1=14
        r2=2

result---new contents of mapper:

>3f >b8 >35 >b8 >b8 >b8 >b8 >b8
       |
       |
        virtual page # 14 is now located at >4000 (cpu address)

        another way to see this is to say that data at a virtual address of
        >0001_c000 is addressable at >4000


++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#4      return a list of virtual pages

        r1= execution address at which to place the list
        r2= length reserved for list

        you'll get an error code if the list is longer than the reserved area

returns (  *R2 ), physical pages numbers:

>3f >3e >3d >b8 >b8 >b8 >b8 >3c >3b >3a >39 >38 >37 >36 >35 >34


        getting such a list is useful if your application does
        _lots_ of paging around to get at the data it needs to use.
        It is quite useful if your application also needs to use
```

pointers longer than 16 bits to maintain some complex data
structure, if you are maintaining 32 bit pointers, here is
how to get at the data addressed by the pointer:

assume: r1,r2 = 32 bit pointer, @paglst are bytes from opcode #4

```
        movb r2,r1              ok, since only low 5 bits of r1 are used
        andi r1,>e01f           keep 8 bits, zap the others
        src  r1,13              rotate them to make an index into the page list
        andi r2,>1fff           mask off the high three bits, they're now
*                               in R1 ...
*
        movb @paglst(r1),@mapper+4          put it at >8000
        movb @paglst+1(r1),@mapper+5        put next page at >a000
*
* it is not necessary to place two pages into the mapper if you
* know for certain that the record accessed by the pointer does not
* cross page boundaries, the above code is just a method of playing it
* safe
*
        mov  @>8000+field_offset(r2),r3
*
* this of course assumes that there is some record addressed by the
* initial pointer, and that the record contains fields of some
* data structure.  fields are easy to set up with a DORG statement
* for each record type in use by an application
*
```

++

#5 declare shared pages

any task may declare a subset of its virtual address space to be
"shared"

each set of shared pages is given a "type" so that other applications
can later assign a certain "type" of shared pages into their
virtual address spage without knowlege of which physical pages
belong to a "shared" group

it is recommended that "types" be assigned by the distributor of MDOS,
so that incompatible applications do not try to use the same "type"
if you decide to use a "type" please correspond with the distributor
of MDOS to coordinate your development efforts with others.

a "shared" type may only be declared once, and always resides in a
group of consecutive virtual pages. If all applications using a
"shared" group of pages release those pages, the "type" may be
redeclared

(MDOS keeps a count of the number of applications using a shared group,
and if the count ever becomes zero, the type is made free for re-use)

*** it is not possible to declare page 0 to be part of a shared group
*** page 0 is _always_ private, since it contains the information
*** which MDOS uses to distinguish between tasks.

arguments:
r1=5 number of pages to be shared in this "type" (40 Kbytes)
r2=8 beginning virtual page number
r3=1 type

results:

physical page numbers:

```
>3f >3e >3d >b8 >b8 >b8 >b8 >3c >3b >3a >39 >38 >37 >36 >35 >34

virtual (local) page numbers:
 0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15
                                 <-- shared pages ->
                                 <-- type #1      ->

let us call the above page list "task #1", for the next few examples
```

++

#7 get "shared" pages (we'll come back to opcode #6 later)

1) the "type" must have be declared by a previous call to #5 by another task
2) if shared type has 5 pages, there must be a hole of at least 5 pages in the virtual page list for the shared type to map into it is not possible to overlay shared and non-shared pages, nor is it possible to overlap different "types" of shared pages

task #2, initially 24k

```
vp:  1   2   3   4   5   6   7   8   9
pp: >20 >21 >22 >b8 >b8 >b8 >b8 >b8 >23
```

task #2 calls opcode #7

arguments:
r1=1
r2=3

results, task #2

```
vp:  1   2   3   4   5   6   7   8   9
pp: >20 >21 >22 >3b >3a >39 >38 >37 >23

              <-- shared pages -->
```

note that arguments of: r1=1, r2=4 would generate an error, since page #9 has already been assigned

also note that it is ok to have the shared group at

>1 0000 in task #1, but at
>0 6000 in task #2

also note that physical pages >37 to >3b can now be accessed by both tasks.

also: a task may use more than one "shared type" simultaneously

shared pages can be a good way to communicate between tasks, it is also a good way to reduce memory usage for an application which is used by more than one user at a time, since you could "share" the object code of the application between the users, and have more memory available for each users' data.

++

#6 release shared pages

only the "type" argument is needed, since MDOS keeps track of

the number of pages in each "type", and its location within the virtual address space of the tasks which are using the "type"

if task #2 were to call opcode #6 with
r1=1

the following would result:

```
vp:  1   2   3   4   5   6   7   8   9
pp: >20 >21 >22 >b8 >b8 >b8 >b8 >b8 >23
                <- the shared pages are no longer here ->
```

++

#8 get size of shared group

if any task were to call this, they can obtain the number of pages used by each active type

example: task #3

argument:
r1=1

return:
r0=0 (assuming task #1, above, still had shared pages)
r1=5 size of shared type #1 in above example

++

this is _fun_ stuff...<sic>

==========

```
*******************************************************************************
**
**    Filename:             memory_i
**    Release:              Version 1.5
**    Date:                 26/July/1991
**
**    This file contains the definitions for the MDOS Memory library
**
**    Copyright 1991 by LGMA Products
**
**    Revision History:
**
**    26/Jul/91       Re-released as memory_i
**    20/Dec/87       Updated, fixed opcodes & added mapper register address
**    30/Nov/87       Initial Release
MEMXOP  EQU     7               ; MEMORY MANAGEMENT XOP NUMBER
**
**  Opcodes
**
RETFRE  EQU     0               ; RETURN FREE PAGES IN SYSTEM
**
GETLCP  EQU     1               ; GET # PAGES @LOCAL PAGE #, SPEED
RETLCP  EQU     2               ; RETURN # PAGES @LOCAL PAGE #
MAPLCP  EQU     3               ; MAP LOCAL PAGE @ EXECUTION PAGE
GETMAP  EQU     4               ; GET ADDRESS MAP
DECSHP  EQU     5               ; DECLARE SHARED PAGES (TYPE,#PAGES,LOCAL P#)
RELSHP  EQU     6               ; RELEASE SHARED PAGES (TYPE)
GETSHP  EQU     7               ; GET SHARED PAGES
SIZSHP  EQU     8               ; SIZE OF SHARED PAGE GROUP (TYPE)
MOVMML  EQU     9               ; MOVE MEMORY LOCAL1 TO LOCAL2 WITH LENGTH
LIBSET  EQU     10              ; SET LIBRARY PAGES
**
**  Error Returns
**
MEMNON  EQU     0               ; NO ERROR
NOTENF  EQU     1               ; NOT ENOUGH FREE MEMORY
CANREM  EQU     2               ; CAN'T REMAP EXECUTION PAGE 0
NOPAGE  EQU     3               ; NO PAGE AT LOCAL ADDRESS
USEFUL  EQU     4               ; USER AREA NOT LARGE ENUF FOR LIST
SHATYP  EQU     5               ; SHARED TYPE ALREADY DEFINED
SHAEXI  EQU     6               ; SHARED TYPE DOESN'T EXIST
CANOVR  EQU     7               ; CAN'T OVERLAY SHARED & PRIVATE MEMORY
OUTTAB  EQU     8               ; OUT OF TABLE SPACE
**
**  MAPPER REGISTER ADDRESS
**
MMRREG  EQU     >F110           ; MAPPER REGISTER START ADDRESS (8 PAGES)
*******************************************************************************
```

```
*****************************************************************************
*
* I/O library ...  XOP @eight,0
*
*      PASS: r0=pointer to PAB in local memory (64k addressable by CPU)
*
*      PAB = peripheral access block
*
*      PAB format:
*
*      byte  0      i/o opcode
*            1      mode flags
*            2      returned error code
*            3      buffer address, high byte       |
*            4      buffer address, middle byte     | task-virtual
*            5      buffer address, low byte        |      address
*            6      \
*            7       \ record number
*            8      \
*            9       \ logical record length
*           10      memory type flag, >00 is CPU, non-zero is VDP
*           11      character count, high byte
*           12      character count, middle byte
*           13      character count, low byte
*           14      status byte
*           15      name length <byte count>
*           16+     text of name
*
*         Byte 1, mode flags:
*
*         7 6 5 4 3 2 1 0    meaning
*         | | | | | | | +--- 0=sequential file access
*         | | | | | | |      1=relative file access <fixed files only>
*         | | | | | | +------ 00=update mode, r/w access, fixed files only
*         | | | | | |         01=output mode, write only, erase old file *
*         | | | | | |          contents
*         | | | | | |         10-input mode, read only
*         | | | | | |         11=append mode, write only to EOF, not for fixed
*         | | | | | +-------- 0=display format data
*         | | | | |           1=internal format data
*         | | | | +---------- 0-fixed record lengths
*         | | | |             1=variable record lengths
*         +-+-+-------------- not used, set to zero!
*
*         Byte 2, error codes:
*
*         7 6 5 4 3 2 1 0
*         | | | | | | | |
*         | | | +-+-+-+-+--- extended code for error #7, mask otherwise
*         | | |
*         +-+-+- 000 non-existent device name
*                001 operation aborted due to write protection
*                010 bad open attribute (filetype,record length,mode)
*                011 illegal operation (bad opcode on this device)
*                100 out of table space, no free buffers
*                101 attempt to read past end of file
*                110 low-level device error (parity, bad media)
*                111 catch-all for other errors
*                    (mismatch between program and data file,
*                     non-existent file opened for input, etc)
*
* PAB usage, by opcode:
*
*
*      OPEN          pab in
```

```
*
*       byte  0       00
*             1       mode flags
*             2       -
*             3       -
*             4       -
*             5       -
*             6       \  number of records to reserve space for
*             7        \ on an open which creates a new file
*             8       \  logical record length to use. if this is zero and the
*             9        \ file exists, uses file's record length, otherwise
*                       + if zero, and file doesn't exist, defaults to 80
*            10
*            11       -
*            12       -
*            13       -
*            14       -
*            15       name length <byte count>
*            16+      text of name
*
*       OPEN           pab out
*
*       byte  0       -
*             1       -
*             2       error code
*             3       -
*             4       -
*             5       -
*             6       -
*             7       -
*             8       \
*             9        \ returned record length, different only if zero was *
                        passed
*            10       -
*            11       0
*            12       \  true record length of file, will not agree with bytes *
                         8,9
*            13        \ if there was a mismatch on the OPEN
*            14       -
*            15       name length <byte count>
*            16+      text or name
*
*
*       CLOSE          pab in          requires prior OPEN
*
*       byte  0       01
*             1       mode flags
*             2       -
*             3       -
*             4       -
*             5       -
*             6       -
*             7       -
*             8       -
*             9       -
*            10       -
*            11       -
*            12       -
*            13       -
*            14       -
*            15       name length <byte count>
*            16+      text of name
*
*       CLOSE          pab out
*
```

```
*       byte  0       -
*             1       -
*             2       error code
*             3       -
*             4       -
*             5       -
*             6       -
*             7       -
*             8       -
*             9       -
*            10       -
*            11       -
*            12       -
*            13       -
*            14       -
*            15       name length <byte count>
*            16+      text of name
*
*
*       READ           pab in          requires prior OPEN
*
*       byte  0       02
*             1       mode flags
*             2       -
*             3       \
*             4        \
*             5         \ buffer address
*             6       \
*             7        \ on FIXED files, record number to read from
*             8       \
*             9        \ logical record length, as returned by OPEN
*            10       transfer flag: 0 is cpu, non-zero is VDP
*            11       -
*            12       -
*            13       -
*            14       -
*            15       name length <byte count>
*            16+      text of name
*
*       READ           pab out
*
*       byte  0       -
*             1       -
*             2       error code
*             3       -
*             4       -
*             5       -
*             6       \  for fixed files
*             7        \ (passed record number)+1
*             8       -
*             9       -
*            10       -
*            11       0
*            12       \
*            13        \ number of chars transfered to buffer
*            14       -
*            15       name length <byte count>
*            16+      text of name
*
*
*       WRITE          pab in          requires prior OPEN
*
*       byte  0       03
*             1       mode flags
*             2       -
```

```
*             3       \
*             4        \
*             5         \ buffer address
*             6       \
*             7        \ for fixed files, record number to write to
*             8       \
*             9        \ logical record length, as returned by OPEN
*            10       transfer flag: 0 is cpu, non-zero is VDP
*            11       \
*            12        \
*            13         \ number of chars to write from buffer into record
*            14       -
*            15       name length <byte count>
*            16+      text of name
*
*       WRITE          pab out
*
*       byte  0        -
*             1        -
*             2        error code
*             3        -
*             4        -
*             5        -
*             6        \  for fixed files,
*             7         \ (passed record number)+1
*             8        -
*             9        -
*            10        -
*            11        -
*            12        -
*            13        -
*            14        -
*            15       name length <byte count>
*            16+      text of name
*
*
*       RESTORE        pab in          requires prior OPEN
*
*       byte  0        04
*             1        mode flags
*             2        -
*             3        -
*             4        -
*             5        -
*             6        \  for fixed files in relative access mode, record number
*             7         \ to position the r/w pointer-all others go to file start
*             8        -
*             9        -
*            10        -
*            11        -
*            12        -
*            13        -
*            14        -
*            15       name length <byte count>
*            16+      text of name
*
*       RESTORE        pab out
*
*       byte  0        -
*             1        -
*             2        error code
*             3        -
*             4        -
*             5        -
*             6        \
```

```
*             7       \ updated record number for fixed files
*             8       -
*             9       -
*            10       -
*            11       -
*            12       -
*            13       -
*            14       -
*            15       name length <byte count>
*            16+      text of name
*
*
*       LOAD          pab in
*
*       byte  0       05
*             1       -
*             2       -
*             3       \
*             4        \
*             5         \ buffer address
*             6       -
*             7       -
*             8       -
*             9       -
*            10       transfer flag: 0 is cpu, non-zero is VDP
*            11       \
*            12        \
*            13         \ maximum number of chars to allow from IMAGE file
*            14       -
*            15       name length <byte count>
*            16+      text of name
*
*       LOAD          pab out
*
*       byte  0       -
*             1       -
*             2       error code
*             3       -
*             4       -
*             5       -
*             6       0
*             7       \
*             8        \  number of bytes in IMAGE file, returned even when
*             9         \ LOAD fails due to buffer size
*            10       -
*            11       -
*            12       -
*            13       -
*            14       -
*            15       name length <byte count>
*            16+      text of name
*
^
*       SAVE          pab in
*
*       byte  0       06
*             1       -
*             2       -
*             3       \
*             4        \
*             5         \ buffer address
*             6       -
*             7       -
*             8       -
*             9       -
```

```
*            10      transfer flag: 0 is cpu, non-zero is VDP
*            11      \
*            12       \
*            13        \ number of bytes to save as an IMAGE
*            14      -
*            15      name length <byte count>
*            16+     text of name
*
*      SAVE          pab out
*
*      byte  0       -
*            1       -
*            2       error code
*            3       -
*            4       -
*            5       -
*            6       -
*            7       -
*            8       -
*            9       -
*           10       -
*           11       -
*           12       -
*           13       -
*           14       -
*           15       name length <byte count>
*           16+      text of name
*
*
*      DELETE file   pab in
*
*      byte  0       07
*            1       -
*            2       -
*            3       -
*            4       -
*            5       -
*            6       -
*            7       -
*            8       -
*            9       -
*           10       -
*           11       -
*           12       -
*           13       -
*           14       -
*           15       name length <byte count>
*           16+      text of name
*
*      DELETE file   pab out
*
*      byte  0       -
*            1       -
*            2       error code
*            3       -
*            4       -
*            5       -
*            6       -
*            7       -
*            8       -
*            9       -
*           10       -
*           11       -
*           12       -
*           13       -
```

```
*            14      -
*            15      name length <byte count>
*            16+     text of name
*
*
*       DELETE rec   pab in          reserved for key-indexed files
*                                    not yet supported
*       byte  0      08
*             1      -
*             2      -
*             3      -
*             4      -
*             5      -
*             6      -
*             7      -
*             8      -
*             9      -
*            10      -
*            11      -
*            12      -
*            13      -
*            14      -
*            15      name length <byte count>
*            16+     text of name
*
*       DELETE rec   pab out
*
*       byte  0      -
*             1      -
*             2      error code= >60, bad opcode
*             3      -
*             4      -
*             5      -
*             6      -
*             7      -
*             8      -
*             9      -
*            10      -
*            11      -
*            12      -
*            13      -
*            14      -
*            15      name length <byte count>
*            16+     text of name
*
*
*       STATUS       pab in    can be used on open or closed file
*
^       byte  0      09
*             1      -
*             2      -
*             3      -
*             4      -
*             5      -
*             6      -
*             7      -
*             8      -
*             9      -
*            10      -
*            11      -
*            12      -
*            13      -
*            14      -
*            15      name length <byte count>
*            16+     text of name
```

```
*
*       STATUS          pab out
*
*       byte  0         -
*             1         -
*             2         error code
*             3         -
*             4         -
*             5         -
*             6         -
*             7         -
*             8         -
*             9         -
*            10         -
*            11         -
*            12         -
*            13         -
*            14         flag byte
*
*      lsb bit 0        0=not at end of file
*                       1=at end of open file, read not possible, write is
*               1       0=there is space to expand file
*                       1=media is full, no room to expand file
*               2       0=fixed records
*                       1=variable records
*               3       0=data
*                       1=program image
*               4       0=Display format data
*                       1=internal format data
*               5       currently unused, always zero
*               6       0=file is not protected against writes
*                       1=file is protected against writes
*               7       0=file exists
*                       1=file does not exist
*
*            15         name length <byte count>
*            16+        text of name
*
*
*       BREAD           pab in          read sectors from any disk file
*
*       byte  0         0A
*             1         -
*             2         -
*             3         \
*             4          \
*             5           \ buffer address
*             6          \
*             7           \ sector offset within file to begin read
*             8         -
*             9         -
*            10         transfer flag, 0 is cpu, non-zero is VDP
*            11         0
*            12         \  number of sectors to read from file
*            13          \ if zero, transfers files ID info into buffer
*            14         -
*            15         name length <byte count>
*            16+        text of name
*
*       BREAD           pab out
*
*       byte  0         -
*             1         -
*             2         error code
*             3         \ updated to point to location in memory after the
```

```
*            4       \ last byte successfully read from disk
*            5        \
*            6       \  (sector offset of last sector read)+1
*            7        \ if read error, points to the bad sector
*            8       -
*            9       -
*           10       -
*           11       0
*           12       \  count of number of sectors
*           13        \ not read due to error condition
*           14       -
*           15       name length <byte count>
*           16+      text of name
*
* @buffer is file ID info as follows:
*
*      byte:
=           0,1      extended record length, used for files with records
=                    longer than 255 bytes
=           2        file status flags
=
= BIT:  Msb 7 6 5 4 3 2 1 0   MEANING
=           | | | | | | | +-> 0-DATA 1-PROGRAM
=           | | | | | | +---> 0-ASCII 1-BINARY
-           | | | | | +-----> 2              RESERVED
=           | | | | +-------> 3              0-NOT PROTECTED 1-PROTECTED
=           | +-+-+---------> 4-6            RESERVED
=           +---------------> 7              0-FIXED RECORD 1-VARIABLE LENGTH
-
=           3        number of records which can fit in a sector (256 bytes)
=                    0 for program image, extended record lengths
=           4,5      number of sectors reserved for file
=           6        number of bytes used in last sector of file
=                    (0 means all 256 are used)
=           7        logical record length
=                    0 for program image, extended record lengths
-           8,9      (bytes reversed)
=                    for fixed files, (highest used record number)+1
=                    for program image, variable files, number of sectors
=                    actually used
=           10,11    date of creation     bits: yyyy yyyM MMMd dddd
=           12,13    time of creation           hhhh hmmm mmms ssss
=                                               seconds are / 2, with remainder
=                                               discarded
-           14,15    date of last change   "
=           16,17    time of last change   "
=
*
*
*
*      BWRITE       pab in         write sectors to any disk file
*
*      byte  0      0B
*            1      -
*            2      -
*            3      \
*            4       \
*            5        \ buffer address
*            6       \
*            7        \ sector offset within file to begin write
*            8       -
*            9       -
*           10       transfer flag, 0 is cpu, non-zero is VDP
*           11       0
*           12       \  number of sectors to write to file
```

```
*            13        \ if zero, creates file with ID info from buffer
*                       \ buffer info described in BREAD
*            14        -
*            15        name length <byte count>
*            16+       text of name
*
*       BWRITE         pab out
*
*       byte  0        -
*             1        -
*             2        error code
*             3        \  points to location in memory after last byte
*             4        \ successfully written to disk
*             5        \
*             6        \  (sector offset of last sector written)+1
*             7        \ if write error, points to the bad sector
*             8        -
*             9        -
*            10        -
*            11        0
*            12        \  count of number of sectors not
*            13        \ written due to error condition
*            14        -
*            15        name length <byte count>
*            16+       text of name
*
*
*       PROTECT        pab in          change file protection
*
*       byte  0        0C
*             1        -
*             2        protect flag, zero is unprotect file
*                                    non-zero is protect file
*             3        -
*             4        -
*             5        -
*             6        -
*             7        -
*             8        -
*             9        -
*            10        -
*            11        -
*            12        -
*            13        -
*            14        -
*            15        name length <byte count>
*            16+       text of name
*
*       PROTECT        pab out
*
*       byte  0        -
*             1        -
*             2        error code
*             3        -
*             4        -
*             5        -
*             6        -
*             7        -
*             8        -
*             9        -
*            10        -
*            11        -
*            12        -
*            13        -
*            14        -
```

```
*            15      name length <byte count>
*            16+     text of name
*
*
*      RENAME        pab in          rename a file
*
*      byte  0       0D
*            1       -
*            2       -
*            3       \
*            4        \
*            5         \ buffer address
*            6       -
*            7       -
*            8       -
*            9       -
*           10       transfer flag, 0 is cpu, non-zero is VDP
*           11       -
*           12       -
*           13       -
*           14       -
*           15       name length <byte count>
*           16+      text of file to rename
*
* @buffer = new 10 character filename
*
*      RENAME        pab out
*
*      byte  0       -
*            1       -
*            2       error code
*            3       -
*            4       -
*            5       -
*            6       -
*            7       -
*            8       -
*            9       -
*           10       -
*           11       -
*           12       -
*           13       -
*           14       -
*           15       name length <byte count>
*           16+      text of name
*
*
*      FORMAT        pab in          format a floppy diskette
*
*      byte  0       0E
*            1       -
*            2       -
*            3       tracks per side
*            4       skew between adjacent tracks
*            5       interlace of individual tracks
*            6       density  (2=double, other is single)
*            7       number of sides to format 2=double, other is single
*            8       -
*            9       -
*           10       -
*           11       -
*           12       -
*           13       -
*           14       -
*           15       name length <byte count>
```

```
*            16+     physical drive spec, ie: "DSK1."
*
*       FORMAT       pab out
*
*       byte  0      -
*             1      -
*             2      error code
*             3      -
*             4      -
*             5      -
*             6      -
*             7      -
*             8      \
*             9       \ total number of sectors formatted
*            10      -
*            11      number of sides formatted  (1,2)
*            12      sectors per track
*            13      -
*            14      -
*            15      name length <byte count>
*            16+     text of name
*
*
*       available devices under MDOS:
*
*       DSK1 \
*       DSK2  \
*       DSK3   \
*       DSK4    \  floppy volumes
*       DSK5       internal ramdisk
*       DSK6       Horizons ramdisk at >1400 cru base
*       DSK7       Horizons ramdisk at >1600 cru base
*
*       WDSx personality card winchester device
*
*       PIO   \
*       PIO/1  \
*       PIO/2   \  parallel printer spooler devices
*
*       RS232   \
*       RS232/1  \
*       RS232/2   \
*       RS232/3    \
*       RS232/4     \ serial port input and output spoolers
*
*       the following switch extensions may be used with the PIO ports
*
*       .CR  turn off cr/lf after each variable record sent
*       .LF  turn off lf    after each variable record sent
*       .NU  print nulls after each variable record to allow for a low
*            slew-rate printer
*       .IB  reconfigure the spooler to recognize a printer with an
*            inverted-busy handshake signal
*       .HS  reconfigure the spooler to perform a full handshake with
*            the printer for each byte sent (instead of just strobe)
*
*
*       the following switch extensions may be used with the RS232 ports
*
*       .CR  turn off cr/lf after each variable record sent
*       .LF  turn off lf    after each variable record sent
*       .NU  print nulls after each variable record to allow for a low
*            slew-rate printer
*       .BA=baudrate   (110,300,600,1200,2400,4800,9600,19200)
*       .DA=databits   (7,8)
```

```
*       .PA=O,E,N
*       .TW  use two stopbits on transmission
*       .CH  check parity
*
*       Note that none of the RS232 switches take effect unless they are
*       found as a special record within spooler itself
*
*       The only way to place the switches into the spooler is to perform
*       an OPEN call to the RS232 with the high bit of the flag byte set to 1
*       ie: the first two bytes of the PAB could be >0080, which would append
*       the switch record onto the end of the spooler, so as not to affect
*       characters which may have been placed into the spooler while other
*       switches may have been extant
*
********************************************************************************
```

```
*******************************************************************************
**
**   Filename:            Io_i
**   Release:             Version 1.5
**   Date:                26/July/1991
**
**   This file contains the definitions for the MDOS I/O library
**
**   Copyright 1991 by LGMA Products
**
**   Revision History:
**
**   26/Jul/91       Re-Release as Io_i
**   23/Feb/88       Change Program Size parameter (16k too large)
**   09/Jan/88       Corrected PABCHC offset
**   20/Dec/87       Updated, added PRGSIZ equate
**   30/Nov/87       Initial Release
**

IOXOP   EQU     8               ; INPUT/OUTPUT XOP NUMBER
**
**  PAB FORMAT:
**
PABOPC  EQU     0               ; OPCODE
PABMOD  EQU     1               ; MODE
PABERR  EQU     2               ; ERROR
PABBUF  EQU     3               ; BUFFER ADDRESS (3 BYTES)
PABREC  EQU     6               ; RECORD NUMBER (2 BYTES)
PABLOG  EQU     8               ; LOGICAL RECORD LENGTH (2 BYTES)
PABMEM  EQU     10              ; PAB MEMORY FLAG (0=CPU, <>0=VDP)
PABCHS  EQU     11              ; PAB CHARACTER COUNT (2 BYTES)
PABCHC  EQU     12              ; CHARACTER COUNT (3 BYTES)
PABSTA  EQU     14              ; PAB STATUS BYTE
PABFLN  EQU     15              ; FILE NAME LENGTH
PABTXT  EQU     16              ; FILE NAME TEXT START
**
**  FILE MODE BITS:
**
FVARIA  EQU     >10             ; VARIABLE FILE
FINTER  EQU     >08             ; INTERNAL
FINPUT  EQU     >04             ; INPUT
FOUTPU  EQU     >02             ; OUTPUT
FAPPEN  EQU     >06             ; APPEND
FRELAT  EQU     >01             ; RELATIVE MODE
**
**  PAB OPCODES:
**
OPCOPE  EQU     >00             ; OPEN
OPCCLO  EQU     >01             ; CLOSE
OPCREA  EQU     >02             ; READ
OPCWRI  EQU     >03             ; WRITE
OPCRES  EQU     >04             ; RESTORE
OPCLOA  EQU     >05             ; LOAD (PROGRAM FILE)
OPCSAV  EQU     >06             ; SAVE (PROGRAM FILE)
OPCDEF  EQU     >07             ; DELETE FILE
OPCDER  EQU     >08             ; DELETE RECORD
OPCSTA  EQU     >09             ; RETURN FILE STATUS
OPCBRE  EQU     >0A             ; BINARY READ
OPCBWR  EQU     >0B             ; BINARY WRITE
OPCPRO  EQU     >0C             ; PROTECT BIT
OPCREN  EQU     >0D             ; RENAME FILE
OPCFOR  EQU     >0E             ; FORMAT A FLOPPY
**
**  ERROR CODES RETURNED:
**
```

```
NONEXI  EQU     >00             ; NON-EXISTANT DEVICE NAME
WRIPRO  EQU     >01             ; OPERATION ABORTED DUE TO WRITE PROTECTION
BADOPE  EQU     >02             ; BAD OPEN ATTRIBUTE
ILLOPE  EQU     >03             ; ILLEGAL OPERATION (E.G. BAD OPCODE)
OUTBUF  EQU     >04             ; OUT OF TABLE OR BUFFER SPACE
REAEOF  EQU     >05             ; ATTEMPT TO READ PAST EOF
LLDEVE  EQU     >06             ; LOW LEVEL DEVICE ERROR (E.G. PARITY)
CATCHA  EQU     >07             ; CATCH ALL FOR OTHER ERRORS
**
**  Other misc. equates
**
PRGSIZ  EQU     >3E00           ; MAXIMUM SIZE OF A PROGRAM LOADER FILE
********************************************************************************
```

VIDEO

```
*********************************************************************
*
*
* video library ... xop @six,0
*
* NOTE: all opcode numbers in this library are subject to revision,
*       though the functions they perform will remain the same
* note: all parameters, including register set, must reside between
*         >E000 & >FFF8 of the calling task.
*
* all opcodes passed in caller R0
*
* SetVideoMode                  op 00
*
* R1x = Video mode
*
* Video modes:
*       0000    Text 1 mode
*       0001    Text 2 mode
*       0002    MultiColor mode
*       0003    Graphic 1 mode
*       0004    Graphic 2 mode
*       0005    Graphic 3 mode
*       0006    Graphic 4 mode
*       0007    Graphic 5 mode
*       0008    Graphic 6 mode
*       0009    Graphic 7 mode
*
* GetVideoMode                  op 01
*
* -----
* R0x = Returned video mode
*
* SetCurPos                     op 02
*
* R1x = Page number
* R2x = Row number
* R3x = Column number
*
* GetCurPos                     op 03
*
* R1x = Page number
* -----
* R0x = Returned row number
* R1x = Returned column number
*
* SetDisPage                    op 04
*
* R1x = Page number
*
* GetDisPage                    op 05
*
* -----
* R0x = Returned page number
*
* ScrollWinUp                   op 06
*
* R1x = Number of lines to scroll
* R2x = Row number of upper left corner
* R3x = Column number of upper left corner
* R4x = Row number of lower right corner
* R5x = Column number of lower right corner
* R6x = Character value for blank lines
* R7h = Foreground color for blank lines
* R7l = Background color for blank lines
```

```
*
* ScrollWinDown                op 07
*
* R1x = Number of lines to scroll
* R2x = Row number of upper left corner
* R3x = Column number of upper left corner
* R4x = Row number of lower right corner
* R5x = Column number of lower right corner
* R6x = Character value for blank lines
* R7h = Foreground color for blank lines
* R7l = Background color for blank lines
*
* ScrollWinLeft                op 08
*
* R1x = Number of lines to scroll
* R2x = Row number of upper left corner
* R3x = Column number of upper left corner
* R4x = row number of lower right corner
* R5x = Column number of lower right corner
* R6x = Character value for blank lines
* R7h = Foreground color for blank lines
* R7l = Background color for blank lines
*

* ScrollWinRight               op 09
*
* R1x = Number of lines to scroll
* R2x = Row number of upper left corner
* R3x = Column number of upper left corner
* R4x = row number of lower right corner
* R5x = Column number of lower right corner
* R6x = Character value for blank lines
* R7h = Foreground color for blank lines
* R7l = Background color for blank lines
*
* WriteCharColor               op 0A
*
* R1x = ASCII character to write to screen
* R2h = Foreground color for character
* R2l = Background color for character in graphics modes
* R3x = Number of times to write character and color
*
* ReadCharColor                op 0B
*
* -----
* R0x = ASCII character read from screen
* R1h = Foreground color for character
* R1l = Background color for character in graphics modes
*
* SetBorderColor               op 0C
*
* R1x = Color to render border
*
* SetColPalette                op 0D
*
* R1x = Palette register number
* R2x = Color to put into palette register
*
* SetPixelColor                op 0E
*
* R1x = X coordinate of pixel
* R2x = Y coordinate of pixel
* R3h = Foreground color to render pixel
* R3l = Background color to render pixel in graphics 2-3
*
```

```
* GetPixelColor                op 0F
*
* R1x = X coordinate of pixel
* R2x = Y coordinate of pixel
* -----
* R0h = Returned foreground color of pixel
* R0l = Returned background color of pixel in graphics 2-3
*
* SetVectorColor               op 10
*
* R1x = X coordinate of first pixel
* R2x = Y coordinate of first pixel
* R3x = X coordinate of second pixel
* R4x = Y coordinate of second pixel
* R5h = Foreground color to render vector
* R5l = Background color to render vector in graphics 2-3
*
* ColorSRCH                    op 11
*
* R1x = X coordinate of source point
* R2x = Y coordinate of source point
* R3l = Color for search
* R3h = Direction for search (>00=LEFT, >FF=RIGHT)
* -----
* ST  = if EQUAL, color found
* R0x = X coordinate of location where color was found
* R1x = Y coordinate of location where color was found
*
* HBlockMove                   op 12
*
* R1x = Row number of upper left corner of source
* R2x = Column number of upper left corner of source
* R3x = Row number of upper left corner of destination
* R4x = Column number of upper left corner of destination
* R5x = Number of rows
* R6x = Number of columns
* R7l = Pixel color for blank pixels
*
* HBlockCopy                   op 13
*
* R1x = Row number of upper left corner of source
* R2x = Column number of upper left corner of source
* R3x = Row number of upper left corner of destination
* R4x = Column number of upper left corner of destination
* R5x = Number of rows
* R6x = Number of columns
*
* LBlockMove                   op 14
*
* R1x = Row number of upper left corner of source
* R2x = Column number of upper left corner of source
* R3x = Row number of upper left corner of destination
* R4x = Column number of upper left corner of destination
* R5x = Number of rows
* R6x = Number of columns
* R7l = Pixel color for blank pixels
* R7h = Logic operation to be performed on destination
*
* LBlockCopy                   op 15
*
* R1x = Row number of upper left corner of source
* R2x = Column number of upper left corner of source
* R3x = Row number of upper left corner of destination
* R4x = Column number of upper left corner of destination
* R5x = Number of rows
```

```
* R6x = Number of columns
* R7h = Logic operation to be performed on destination
*
* BScrollUp                    op 16
*
* R1x = Number of pixels to scroll
* R2x = Row number of upper left corner
* R3x = Column number of upper left corner
* R4x = Row number of lower right corner
* R5x = Column number of lower right corner
* R6h = Pixel color for blank pixels
*
* BScrollDown                  op 17
*
* R1x = Number of pixels to scroll
* R2x = Row number of upper left corner
* R3x = Column number of upper left corner
* R4x = Row number of lower right corner
* R5x = Column number of lower right corner
* R6h = Pixel color for blank pixels
*
* BScrollLeft                  op 18
*
* R1x = Number of pixels to scroll
* R2x = Row number of upper left corner
* R3x = Column number of upper left corner
* R4x = Row number of lower right corner
* R5x = Column number of lower right corner
* R6h = Pixel color for blank pixels
*
* BScrollRight                 op 19
*
* R1x = Number of pixels to scroll
* R2x = Row number of upper left corner
* R3x = Column number of upper left corner
* R4x = Row number of lower right corner
* R5x = Column number of lower right corner
* R6h = Pixel color for blank pixels
*
* SpriteDef                    op 1A
*
* R1x = Pointer to sprite data
* R2x = # of sprites to define
*
* SpriteDel                    op 1B
*
* R1x = Pointer to list of sprite #'s
* R2x = # of sprites to delete (>FFFF for all)
*
* SpriteLocate                 op 1C
*
* R1x = Pointer to location data
* R2x = # of sprites to locate
*
* SpriteMotion                 op 1D
*
* R1x = Pointer to motion data
* R2x = # of sprites to put in motion
*
* SpriteColor                  op 1E
*
* R1x = Pointer to color data
* R2x = # of sprites to color
*
* SpritePattern                op 1F
```

```
*
* R1x = Pointer to pattern # data
* R2x = # of sprites to pattern
*
* SpriteMagnify                 op 20
*
* R1x = MagFac (1-4, just like XB)
*
* SpritePosition                op 21
*
* R1x = # of sprite to get position of
* -----
* R0x = Returned Row of sprite
* R1x = Returned Column of sprite
*
* SpriteCoinc                   op 22
*
* R1x = Type (0=sprites, 1=locations, 2=any two sprites)
* R2x = # of coincidence checks
* R3x = Pointer to test field
* R4x = Pointer to result field
* -----
* R0x = # of coincidences detected
*
* SpritePatDef                  op 23
*
* R1x = CPU address of sprite pattern definitions
* R2x = # of sprite patterns to define
*
* CharPatDef                    op 24
*
* R1x = CPU address of character pattern definitions
* R2x = # of character patterns to define
*
* SetMargins                    op 25
*
* R1x = Page number
* R2x = Top margin
* R3x = Bottom margin
* R4x = Left margin
* R5x = Right margin
*
* GetMargins                    op 26
*
* R1x = Page number
* -----
* R0x = Returned top margin
* R1x = Returned bottom margin
* R2x = Returned left margin
* R3x = Returned right margin
*
* WriteTTY                      op 27
*
* R1x = CPU address of string
* R2x = # of characters in string (0=NULL terminated)
*
* SetTTYPos                     op 28
*
* R1x = Page number
* R2x = Row number
* R3x = Column number
*
* GetTTYPos                     op 29
*
* R1x = Page number
```

```
* -----
* R0x = Returned row number
* R1x = Returned column number
*
*
* SetMouse                      op 2A
*
* R1x = new Xposition for mouse
* R2x = new Yposition for mouse
* R3x = scale factor for mouse speed  (0 to 7) 0=fastest
* -----
*
* GetMouse                      op 2B
*
* -----
* R1x = Returned Xposition for mouse
* R2x = Returned Yposition for mouse
* R3x = b1 b2 b3 0 xxxx xxxx xxxx   (highest bits)
*       \          b1= left   1=down
*          \       b2= middle 1=down
*             \    b3= right  1=down
*
* GetMouseRel                   op 2C
*
* -----
* R1x = Returned X displacement since last call to GetMouse or GetMouseRel
* R2x = Returned Y displacement since last call to GetMouse or GetMouseRel
*
* MakeSound                     op 2D
*
* R1x = Generator 1 frequency in Hz
* R2x = Generator 2 frequency in Hz
* R3x = Generator 3 frequency in Hz
* R4h = Attenuation for Generator 1 (0-15)
* R4l = Attenuation for Generator 2 (0-15)
* R5h = Attenuation for Generator 3 (0-15)
* R6h = control for noise generator: bits= 0000 0w t1 t2
*                                          |  +--+-- 00= 6991 Hz
*                                          |         01= 3496 Hz
*                                          |         10= 1738 Hz
*                                          |         11- same Hz as Gen
* #3
*                                          +---------- 0= periodic noise
*                                                      1= white noise
* R6l - Attenuation for Noise Generator
* R7x = duration of noise in 60th seconds
* -----
*
* SoundStatus                   op 2E
*
* -----
* EQ bit set if no sound is in progress
*
================================================================================
* SetVideoMode
*
* R1x = Video mode
*
* Video modes:
*       0000    Text 1 mode
*       0001    Text 2 mode
*       0002    MultiColor mode
*       0003    Graphic 1 mode
*       0004    Graphic 2 mode
*       0005    Graphic 3 mode
```

```
*       0006    Graphic 4 mode
*       0007    Graphic 5 mode
*       0008    Graphic 6 mode
*       0009    Graphic 7 mode
*       000A    Text 2 mode -- 26 lines
*
*                0   1   9   C  R

* GetVideoMode
*
* -----
* R0x = Returned video mode
* R1x = Number of columns
* R2x = Number of rows
* R3x = Number of Graphics columns (pixels)
* R4x = Number of Graphics rows (pixels)
* R5x = Current page offset (in pixel rows, use for chip commands)
* R6x = Color of screen border
* R7h = Foreground color of text
* R7l = Background color of text
*
================================================================================
```

```
*****************************************************************
X901VC

* SetCurPos
*
* R1x = Row number
* R2x = Column number
*
X902VC

* GetCurPos
*
* -----
* R0x = Returned row number
* R1x = Returned column number
*
X903VC

* SetDisPage
*
* R1x = Page number
* R2x = Initialize tables? (0=YES !0=NO)
*
X904LS  DATA    X904M0,X904M1,X904M2,X904M3

* GetDisPage
*
* -----
* R0x = Returned page number
*
X905VC
X905L1  MOV     @PAGE,*R13
        RT

* ScrollWinUp
*
* R1x = Number of lines to scroll
* R2x = Row number of upper left corner
* R3x = Column number of upper left corner
* R4x = Row number of lower right corner
* R5x = Column number of lower right corner
* R6x = Character value for blank lines
* R7h = Foreground color for blank lines
* R7l = Background color for blank lines
*
X906LS  DATA    X906M0,X906M0,X906M2,X906M0

* ScrollWinDown
*
* R1x = Number of lines to scroll
* R2x = Row number of upper left corner
* R3x = Column number of upper left corner
* R4x = Row number of lower right corner
* R5x = Column number of lower right corner
* R6x = Character value for blank lines
* R7h = Foreground color for blank lines
* R7l = Background color for blank lines
*
X907LS  DATA    X907M0,X907M0,X907M2,X907M0

* ScrollWinLeft
*
* R1x = Number of lines to scroll
* R2x = Row number of upper left corner
* R3x = Column number of upper left corner
```

```
* R4x = Row number of lower right corner
* R5x = Column number of lower right corner
* R6x = Character value for blank lines
* R7h = Foreground color for blank lines
* R7l = Background color for blank lines
*
X908LS  DATA    X908M0,X908M0,X908M2,X908M0

* ScrollWinRight
*
* R1x = Number of lines to scroll
* R2x = Row number of upper left corner
* R3x = Column number of upper left corner
* R4x = Row number of lower right corner
* R5x = Column number of lower right corner
* R6x = Character value for blank lines
* R7h = Foreground color for blank lines
* R7l = Background color for blank lines
*
X909LS  DATA    X909M0,X909M0,X909M2,X909M0

* CALL COLOR   (ONLY WORKS IN SCREEN CODE 3)
*
* R1h = Foreground color
* R1l = Background color
* R2x = Charset # (if mode 3)
*
X90AVC

* GCharColor
*
* R1x = Row
* R2x = Col
* -----
* R0x = ASCII character read from screen
* R1h = Foreground color for character
* R1l = Background color for character
*
X90BLS  DATA    X90BM0,X90BM1,X90BM2,X90BM3

* SetBorderColor
*
* R1x = Color to render border
*
X90CVC

* SetColPalette
*
* R1x = Palette register number
* R2x = Color to put into palette register
*       (G,R,B -- AS BOOK DESCRIBES)          NIBBLE JUSTIFIED MOST 0-G-R-B
*
X90DVC

* SetPixelColor
*
* R1x = X coordinate of pixel
* R2x = Y coordinate of pixel
* R3h = Foreground color to render pixel
* R3l = Background color to render pixel in graphics 2-3
* R4h = Logic operation to be performed
*
X90ELS  DATA    X90EM0,X90EM0,X90EM0,X90EM0

* GetPixelColor
```

```
*
* R1x = X coordinate of pixel
* R2x = Y coordinate of pixel
* -----
* R0h = Returned foreground color of pixel
* R0l = Returned background color of pixel in graphics 2-3
*
X90FLS  DATA    X90FM0,X90FM0,X90FM0,X90FM0

* SetVectorColor      (DRAWING A LINE)
*
* R1x = X coordinate of first pixel
* R2x = Y coordinate of first pixel
* R3x = X coordinate of second pixel
* R4x = Y coordinate of second pixel
* R5h = Foreground color to render vector
* R5l = Background color to render vector in graphics 2-3
* R6h = Logic operation to be performed
*
X910LS  DATA    X910M0,X910M0,X910M0,X910M0

* ColorSRCH         (FOR FILLS)
*
* R1x = X coordinate of source point
* R2x = Y coordinate of source point
* R3l = Color for search
* R3h = Direction for search (>00=LEFT, >FF=RIGHT)
* -----
* ST  = if EQUAL, color found
* R0x = X coordinate of location where color was found
* R1x = Y coordinate of location where color was found
*
X911VC

* HBlockMove
*
* R1x = Row number of upper left corner of source
* R2x = Column number of upper left corner of source
* R3x = Row number of upper left corner of destination
* R4x = Column number of upper left corner of destination
* R5x = Number of rows
* R6x = Number of columns
* R7l = Pixel color for blank pixels
*
X912VC

* HBlockCopy
*
* R1x = Row number of upper left corner of source
* R2x = Column number of upper left corner of source
* R3x = Row number of upper left corner of destination
* R4x = Column number of upper left corner of destination
* R5x = Number of rows
* R6x = Number of columns
*
X913VC

* LBlockMove
*
* R1x = Row number of upper left corner of source
* R2x = Column number of upper left corner of source
* R3x = Row number of upper left corner of destination
* R4x = Column number of upper left corner of destination
* R5x = Number of rows
* R6x = Number of columns
```

```
* R7l = Pixel color for blank pixels
* R7h = Logic operation to be performed on destination
*
X914VC

* LBlockCopy
*
* R1x = Row number of upper left corner of source
* R2x = Column number of upper left corner of source
* R3x = Row number of upper left corner of destination
* R4x = Column number of upper left corner of destination
* R5x = Number of rows
* R6x = Number of columns
* R7h = Logic operation to be performed on destination
*
X915VC

* BScrollUp
*
* R1x = Number of pixels to scroll
* R2x = Row number of upper left corner
* R3x = Column number of upper left corner
* R4x = Row number of lower right corner
* R5x = Column number of lower right corner
* R6h - Pixel color for blank pixels
*
X916VC

* BScrollDown
*
* R1x = Number of pixels to scroll
* R2x = Row number of upper left corner
* R3x = Column number of upper left corner
* R4x = Row number of lower right corner
* R5x = Column number of lower right corner
* R6h = Pixel color for blank pixels
*
X917VC

* BScrollLeft
*
* R1x = Number of pixels to scroll
* R2x = Row number of upper left corner
* R3x = Column number of upper left corner
* R4x = Row number of lower right corner
* R5x = Column number of lower right corner
* R6h = Pixel color for blank pixels
*
X918VC

* BScrollRight
*
* R1x = Number of pixels to scroll
* R2x = Row number of upper left corner
* R3x = Column number of upper left corner
* R4x = Row number of lower right corner
* R5x = Column number of lower right corner
* R6h = Pixel color for blank pixels
*
X919VC
              ##### SEE ADDENDUM FOR MOR ON SPRITES #####
* SpriteDef
*
* R1x = Pointer to sprite data
* R2x = # of sprites to define
```

```
*
X91AVC

* SpriteDel
*
* R1x = Pointer to list of sprite #'s
* R2x = # of sprites to delete (>FFFF for all)
*
X91BVC

* SpriteLocate
*
* R1x = Pointer to location data
* R2x = # of sprites to locate
*
X91CVC

* SpriteMotion
*
* R1x = Pointer to motion data
* R2x = # of sprites to put in motion
*
X91DVC

* SpriteColor
*
* R1x = Pointer to color data     LIST=SP#,SP COL,....,SP#,SP COL
* R2x = # of sprites to color
*
X91EVC

* SpritePattern
*
* R1x = Pointer to pattern # data  (SET POINTER TO SPRITE TABLE 0-255)
* R2x = # of sprites to pattern
*

* SpriteMagnify
*
* R1x = MagFac (1-4, just like XB)
*
X920SM  BYTE    >00,>01,>02,>03

* SpritePosition / SpriteDistance
*
* R1x = # of sprite to get position of
* R2x = Type of Distance, 0 for none, 1 for Sprite, 2 for Location
* R3x = # of second sprite (type 1), or Pixel row (type 2)
* R4x = Pixel column (type 2)
* -----
* R0x = Returned Row of sprite
* R1x = Returned Column of sprite
* R2x = Distance (if second sprite # was given)
* R3X = Distance squared (if second sprite # was given)
X921VC

* SpriteCoinc
*
* R1x = Type (0=sprites, 1=locations, 2=any two sprites)
* R2x = # of coincidence checks (if1,2) IF 2 PUTS THE RESULT IN R4 ONLY 2
* R3x = Pointer to test field     0   SP#, SP#, TOLER  1  SP#,DR,DC,TOL
* R4x = Pointer to result field   0 one COINC PAIR  row,col  if coinc
* -----                           1 SP#, DR, DC
*                                 2 DR,DC         (2 IS ALL, SP TO SP IS 0)
* R0x = # of coincidences detected                      (DR,DC  )
```

```
*
X922LS  DATA    X922T0,X922T1,X922T2

* SpritePatDefGet
*
* R1x = CPU address of sprite pattern definitions
* R2x = # of sprite patterns to define or get
* R3x = Starting pattern #
* R4x = 0 if Def, >FFFF if Get
*
X923VC

* CharPatDefGet
*
* R1x = CPU address of character pattern definitions
* R2x = # of character patterns to define or get     (SPRITE & CHAR DATA SAME
* R3x = Starting pattern #                                  ONLY IN MODE 1,1)
* R4x = 0 if Def, >FFFF if Get
*
X924LS  DATA    X924M0,X924M0,X924M2,X924M0

* SetTextWindow
*
* R1x = Top row
* R2x = Left column
* R3x = Bottom row
* R4x = Right column
*
X925VC

* GetTextWindow
*
* -----
* R0x = Returned top row
* R1x = Returned left column
* R2x = Returned bottom row
* R3x = Returned right column
*
X926VC

* WriteTTY
*
* R1x = CPU address of string
* R2x = # of characters in string (0=NULL terminated)
*
X927LS  DATA    X92700,X92701,X92700,X92700

* RestoreCharSpritePat
*
* R1x = Restore Chars?   (0=No)
* R2x = Restore Sprites? (0=No)
*
X928VC

* SetTextColor
*
* R1h = Foreground color for text
* R1l = Background color for text
*
X929VC

* WriteCharString
*
* R1x = Address of string
* R2x = # of characters in string
```

```
* R3x = 0 if change cursor position, >FFFF if leave cursor at beginning
*
X92ALS  DATA    X92AM0,X92AM1,X92AM2,X92AM3

* HCharColor
*
* R1x = Row
* R2x = Col
* R3x = ASCII character to write to screen
* R4x = Number of times to write character and color
* R5h = Foreground color for character
* R5l = Background color for character
*
X92CLS  DATA    X92CM0,X92CM1,X92CM2,X92CM3

* VCharColor
*
* R1x = Row
* R2x = Col
* R3x = ASCII character to write to screen
* R4x = Number of times to write character and color
* R5h = Foreground color for character
* R5l = Background color for character
*
X92DLS  DATA    X92DM0,X92DM1,X92DM2,X92DM3

* HChar
*
* R1x = Row
* R2x = Col
* R3x = ASCII character to write to screen
* R4x = Number of times to write character and color
*
X92EVC

* VChar
*
* R1x = Row
* R2x = Col
* R3x = ASCII character to write to screen
* R4x = Number of times to write character and color
*
X92FVC

* VWTR - write to video registers, with register save
*
* R1x = VDP register #
* R2l = Value to put into VDP register
*
X935VC

* VRFR - read from video registers, actually, a read from stored values
*
* R1x = VDP register #
* -----
* R0l = Value read from VDP register
*
X936VC

* GetTables
*
* R1x = Pointer in user data, to put copy of tables (24 bytes)
* -----
* Copies of this data are put into user data:
*
```

```
* CTABLE  DATA    0,0
* PTABLE  DATA    0,0
* SCRIMG  DATA    0,0
* SPRATT  DATA    0,0
* SPRPAT  DATA    0,0
* SPRCOL  DATA    0,0
*
X937VC

* GetPalRegs
*
* R1x = Pointer in user data, to put copy of Palette Registers (32 bytes)
* R2x = FORMAT (0=SQUASHED !0=BY THE BOOK, EXPANDED)
*
X938VC

* PrintScreen
*
* R1x = 0 for shades, 1 for outline
* R2x = 0 for normal density (double), 1 for hi density (quad)
*
X92BVC

MOUSE SAME VIDEO

R0 2A-SET MOUSE
R1   -X POSTION
R2    Y POSITION
R3    SCALE FACTOR 0-7 0 FASTEST

R0 2B-GET ABSOLUTE (WHOLE REGISTER)
R1    RETURNED X
R2    RETURNED Y
R3    MOST SIGNIFICANT 3 BITS L BUTTON, 1 DOWN, 0 NOT DOWN

R0 2C-GET RELATIVE (HOW MUCH MOUSE HAS MOVED SINCE LAST CALL TO GET MOUSE)
R1    RETURNED X
R2    RETURNED Y
R3    MOST 3 BITS L BUTTON,1 DOWN 0 NOT DOWN

##### DEFINE SPRITES

POINTER IS TO WORDS (SPRITE MODE 1)

1ST WORD IN LIST IS SPRITE #  (BASE 0)
2ND WORD IN LIST IS CHAR CODE 0-255
3RD WORD IN LIST IS POSITION
4TH WORD IN LIST IS VOLOCITY
5TH WORD IN LIST IS VOLOCITY
6TH WORD IN LIST IS COLOR
POINTER IS TO WORDS (SPRITE MODE 2)

1ST WORD IN LIST IS SPRITE #  (BASE 0)
2ND WORD IN LIST IS CHAR CODE 0-255
3RD WORD IN LIST IS POSITION
4TH WORD IN LIST IS VOLOCITY
5TH WORD IN LIST IS VOLOCITY
NEXT 16 WORDS IN LIST ARE COLORS

##### SPRITE COLOR

SAME AS COLORS FOR SPRITE MODE 1 OR 2
================================================================================
```

```
********************************************************************************
**
**   Filename:            video_i
**   Release:             Version 1.5
**   Date:                26/July/1991
**
**   This file contains the definitions for the MDOS Video library
**
**   Copyright 1991 by LGMA Products
**
**   Revision History:
**
**   26/Jul/91       Re-release as video_i
**   09/Feb/89       Changed for 1.14 video XOP 6 modifications
**   09/Sep/88       Changed names of VCHAR/HCHAR to VCHARW/HCHARW
**   15/Aug/88       Updated for MDOS 1.06
**   30/Nov/87       Initial Release
**
VIDXOP  EQU     6               ; VIDEO LIBRARY XOP NUMBER
**
**  Graphics Modes:
**
**                                    mode          size         color  sprite
**                                   ------------  ------------  ------ -------
TEXT1   EQU     0               ; TEXT 1      :  40x26      :   2  :  N
TEXT2   EQU     1               ; TEXT 2      :  80x26      :   2  :  N
MULTIC  EQU     2               ; MULTICOLOR  :  64x48      :  16  :  1
GRAPH1  EQU     3               ; GRAPHIC 1   :  32x24      :  16  :  1
GRAPH2  EQU     4               ; GRAPHIC 2   :  32x24      :  16  :  1
GRAPH3  EQU     5               ; GRAPHIC 3   :  256dx212d  :  16  :  2
GRAPH4  EQU     6               ; GRAPHIC 4   :  512dx212d  :  16  :  2
GRAPH5  EQU     7               ; GRAPHIC 5   :  512dx212d  :   4  :  2
GRAPH6  EQU     8               ; GRAPHIC 6   :  512dx212d  :  16  :  2
GRAPH7  EQU     9               ; GRAPHIC 7   :  256dx212d  :  32  :  2
**
**  Opcodes:
**
SETVID  EQU     >00             ; SetVideoMode
GETVID  EQU     >01             ; GetVideoMode
SETCUR  EQU     >02             ; SetCurPos
GETCUR  EQU     >03             ; GetCurPos
SETDIS  EQU     >04             ; SetDisPage
GETDIS  EQU     >05             ; GetDisPage
SCRWUP  EQU     >06             ; ScrollWinUp
SCRWDN  EQU     >07             ; ScrollWinDown
SCRWLE  EQU     >08             ; ScrollWinLeft
SCRWRI  EQU     >09             ; ScrollWinRight
CCOLOR  EQU     >0A             ; CALL COLOR                          | 0.95
^CALLSC EQU     >0A             ; CallScreen                          | 1.14+
GETCCO  EQU     >0B             ; GCharColor                          | 1.06
SETBOR  EQU     >0C             ; SetBorderColor
SETCOP  EQU     >0D             ; SetColPallette
SETPIC  EQU     >0E             ; SetPixelColor
GETPIC  EQU     >0F             ; GetPixelColor
SETVTC  EQU     >10             ; SetVectorColor
COLSRC  EQU     >11             ; ColorSRCH
HBLMOV  EQU     >12             ; HBlockMove
HBLCOP  EQU     >13             ; HBlockCopy
LBLMOV  EQU     >14             ; LBlockMove
LBLCOP  EQU     >15             ; LBlockCopy
BSCRUP  EQU     >16             ; BScrollUP
BSCRDN  EQU     >17             ; BScrollDown
BSCRLE  EQU     >18             ; BScrollLeft
BSCRRI  EQU     >19             ; BScrollRight
SPRDEF  EQU     >1A             ; SpriteDef
```

```
SPRDEL  EQU     >1B             ; SpriteDel
SPRLOC  EQU     >1C             ; SpriteLoc
SPRMOT  EQU     >1D             ; SpriteMotion
SPRCOL  EQU     >1E             ; SpriteColor
SPRPAT  EQU     >1F             ; SpritePattern
SPRMAG  EQU     >20             ; SpriteMagnify
SPRPOS  EQU     >21             ; SpritePosition
SPRCOI  EQU     >22             ; SpriteCoincindence
SPRPAD  EQU     >23             ; SpritePatDefGet                | 1.06
CHAPAD  EQU     >24             ; CharPatDefGet                  | 1.06
SETTWI  EQU     >25             ; SetTextWindow                  | 1.06
GETTWI  EQU     >26             ; GetTextWindow                  | 1.06
WRITET  EQU     >27             ; WriteTTY
RESTCH  EQU     >28             ; RestoreCharacters              | 1.06
SETTCO  EQU     >29             ; SetTextColor                   | 1.06
WRICHS  EQU     >2A             ; WriteCharString                | 1.06
PRISCR  EQU     >2B             ; PrintScreen                    | 1.06
HCHCOL  EQU     >2C             ; HCharColor                     | 1.06
VCHCOL  EQU     >2D             ; VCharColor                     | 1.06
HCHARW  EQU     >2E             ; HChar                          | 1.06
VCHARW  EQU     >2F             ; VChar                          | 1.06
SETMOU  EQU     >30             ; SetMouse                       | 1.06
GETMOU  EQU     >31             ; GetMouse                       | 1.06
GETMOR  EQU     >32             ; GetMouseRel                    | 1.06
MAKSOU  EQU     >33             ; MakeSound                      | 1.06
SNDSTA  EQU     >34             ; SoundStatus                    | 1.06
VVWTR   EQU     >35             ; VDP Write Register             | 1.14+
VVRFR   EQU     >36             ; VDP Read Registers             | 1.14+
GETVTR  EQU     >37             ; Get Video Tables               | 1.14+
GETPAR  EQU     >38             ; Get Pallette Registers         | 1.14+
SETEDG  EQU     >3A             ; Set Edge Color                 | 0.95h
================================================================================
```

```
*********************************************************************
*
* utility library ...  XOP @nine,0
*
*
*
* OP #0     CHECK TIME FOR VALIDITY
*                              PASS:   r0=0
*                              return: EQ bit set if time is valid
*                                      reset if time is invalid
*
* OP #1     CONVERT TIME TO STRING
*                              PASS:   R0=1
*                                      R1=pointer to string in local mem
*                                         (at least 10 bytes)
*                              return: 8 character string
*
* OP #2     CONVERT STRING TO TIME
*                              PASS:   R0=2
*                                      R1=pointer to string in local mem
*                                         (at least 10 bytes)
*                              return: EQ flag set if time was valid
*
* OP #3     CHECK DATE FOR VALIDITY
*                              PASS:   R0-3
*                              return: EQ bit is set if date is valid
*
* OP #4     CONVERT DATE TO STRING
*                              PASS:   R0=4
*                                      R1=pointer to string in local mem
*                                         (at least 10 bytes)
*                              return: 8 character string
*
* OP #5     CONVERT STRING TO DATE
*                              PASS:   R0=5
*                                      R1=pointer to string in local mem
*                                         (at least 10 bytes)
*                              return: EQ flag set if date was valid
*
* OP #6     CONVERT MM,DD,YYYY into julian date
*                              PASS:   R0=6
*                                      R1=month
*                                      R2=day
*                                      R3=year
*
*                              return: R1=MSword of julian date
*                                      R2=LSword of julian date
*
* OP #7     return day of week
*                              PASS:   R0=7
*                              return: R1=day of week, 1-7
*
* OP #8     parse filename from logical descriptor to physical descriptor
*    IN:
*        R1= POINTER TO TEXT OF STRING TO PARSE                exec address
*        R2= ^ TO LENGTH BYTE IN STRING OUT BUFFER             exec address
*            (1st byte is maxlen for the buffer)
*        R3= FLAG TO CONTROL GENERATION OF ALIAS PREFIX
*   OUT:
*        R0= pointer to ending delimiter
*        R1= 0 is no error, <>0 is error
*        EQUAL FLAG set by contents of R1
*
* OP #9     load program image task
*    in:  r0=9
```

```
*         r1=pointer to length byte of image name              exec address
*
*    out: EQ set if no error
*         r0=error code
*              0=no error
*              1=not enough memory to load task
*              2=bad name for image
*              3=invalid header on task
*              4=task not found, hardware error, wrong file type for task
*
================================================================================
```

TASKS

```
================================================================================
          General notes on user tasks within MDOS:

address:
     >0000 to >0043      reserved for OS
     >0044 to >007f      can be used for user defined XOPs
     >0080 to >03ff      reserved for OS
     >0400               entry of user program
           to >1fff      execution page 0 for this task

     >2000 to >3fff      execution page 1 for this task
     >4000 to >5fff      execution page 2 for this task
     >6000 to >7fff      execution page 3 for this task
     >8000 to >9fff      execution page 4 for this task
     >a000 to >bfff      execution page 5 for this task
     >c000 to >dfff      execution page 6 for this task
     >e000 to >efff      execution page 7 for this task (part 1)
     >f000 to >f07f      fast register locations for this task
                         (all calls to OS with XOP @op,0 must have
                          registers located here)
     >f080 to >f13f      reserved for OS
     >f140 to >fff9      execution page 7 for this task (part 2)
     >fffa to >ffff      reserved for OS

all system routines are called through XOP zero, as follows:

     .

     XOP @libnum,0       libnum is the address of a memory location
     .                   containing the Library number to call
     .
     .

LIBNUM DATA >0006
      (for video library)
     A subroutine will be provided to allow the task to fetch
     the command line options with which it was invoked
================================================================================
```

```
*******************************************************************************
**
**   Filename:            TaskHead_I
**   Release:             Version 1.5
**   Date:                26/July/1991
**
**   This file contains the definitions for a task header
**
**   Copyright 1991 by LGMA Products
**
**   Revision History:
**
**   26/Jul/91       Initial Release
**
INTREG  EQU  >F080            ; INTERRUPT ROUTINE REGISTERS
XOPREG  EQU  >F0A0            ; XOP REGISTERS
SYS1    EQU  >F0A0            ; SYSTEM 1 REGISTERS
SYS2    EQU  >F0C0            ; SYSTEM 2 REGISTERS
SYSREG  EQU  >F0E0            ; SYSTEM REGISTERS
SCRATC  EQU  >F0E0            ; SCRATCH AREA
MAPPER  EQU  >F110            ; MAPPER REGISTERS
*
RSETVC  EQU  >0000            ; RESET VECTOR
VEC990  EQU  >0004            ; 9901 INTERRUPT
VECMID  EQU  >0008            ; INTERNAL TIMER INTERRUPT
VECINT  EQU  >000C            ; INTERNAL TIMER INTERRUPT
VECEXT  EQU  >0010            ; EXTERNAL BUS INTERRUPT
*
XOPTBL  EQU  >0040            ; START OF XOP TABLE
SYSXOP  EQU  >0040            ; SYSTEM WIDE XOP VECTOR
RSDBUG  EQU  >0044            ; RS232 DEBUG XOP
XOPUS2  EQU  >0048            ; USER #2 XOP
XOPUS3  EQU  >004C            ; USER #3 XOP
XOPUS4  EQU  >0050            ; USER #4 XOP
XOPUS5  EQU  >0054            ; USER #5 XOP
XOPUS6  EQU  >0058            ; USER #6 XOP
XOPUS7  EQU  >005C            ; USER #7 XOP
XOPUS8  EQU  >0060            ; USER #8 XOP
XOPUS9  EQU  >0064            ; USER #9 XOP
XOPU10  EQU  >0068            ; USER #10 XOP
XOPU11  EQU  >006C            ; USER #11 XOP
XOPU12  EQU  >0070            ; USER #12 XOP
XOPU13  EQU  >0074            ; USER #13 XOP
XOPU14  EQU  >0078            ; USER #14 XOP
XOPU15  EQU  >007C            ; USER #15 XOP
XOPCAL  EQU  >0080            ; XOP ENTRY
INTCAL  EQU  >0086            ; INTERRUPT ENTRY
KILLIT  EQU  >00AC            ; INHIBIT INTERRUPTS & RESET
DELON   EQU  >00D8            ; DELETE ON
ESCSEQ  EQU  >00DA            ; ESCAPE SEQUENCE
ESCCNT  EQU  >00DC            ; ESCAPE COUNT
ESCROU  EQU  >00DE            ; ESCAPE ROUTINE
ESCDAT  EQU  >00E0            ; ESCAPE DATA
STDCLR  EQU  >00E2            ; STANDARD CLEAR
LINFLG  EQU  >00E4            ; LINE FLAG
COTASK  EQU  >00E6            ; CO-TASK
PAGE    EQU  >00E8            ; PAGE #
PAGEPX  EQU  >00EC            ; PAGE PX
TURX    EQU  >00EE            ;
TURXMN  EQU  >00F0            ;
TURXMX  EQU  >00F2            ;
TURY    EQU  >00F4            ;
TURYMN  EQU  >00F6            ;
TURYMX  EQU  >00F8            ;
```

```
CTLP    EQU  >00FA            ;
CTLS    EQU  >00FC            ;
BREAK   EQU  >00FE            ; BREAK ON
TSKTBL  EQU  >0100            ; TASK ID #
MAXDRV  EQU  >0101            ; MAXIMUM ALIAS LETTER USER CAN SPECIFY
STATE   EQU  >0102            ; PROCESS STATE
SLICE   EQU  >0103            ; # OF SLICES LEFT TIL SWAPPED OUT
PNAME   EQU  >0104            ; NAME OF THE TASK (8-CHARS)
UWP     EQU  >010C            ; USER WORKSPACE POINTER
UPC     EQU  >010E            ; USER PC
UST     EQU  >0110            ; USER STATUS REGISTER
MEMLST  EQU  >0112            ; POINTER TO MEMORY LIST
TSKMAP  EQU  >0118            ; SAVED MEMORY MAP USED DURING XOP'S
CURDRV  EQU  >011C            ; POINTER TO CURRENT DRIVE ENTRY
PATH#P  EQU  >011E            ; POINTER TO TEXT FOR PATH COMMAND
BLKDEV  EQU  >0120            ; POINTER TO NAMES OF BLOCK DEVICES
FREPTR  EQU  >0122            ; POINTER TO FREE NODES
PMTSTR  EQU  >0124            ; POINTER TO PROMPT STRING'S NODES
GPLPAR  EQU  >0126            ; STRING TO CONTROL SPEED OF GPL INSTRUCTIONS
CMDSTR  EQU  >0128            ; STRING CONTAINING COMMAND LINE OPTIONS
ALIASA  EQU  >012A            ; POINTERS TO THE ALIAS STRINGS
HANDL0  EQU  >016A              ; stdin HANDLE (0=keyboard or screen)
HANDL1  EQU  >016C              ; stdout
HANDL2  EQU  >016E              ; stderr
================================================================================
```

MATH

```
*********************************************************************
*
* math library ...  XOP @TEN,0
*
*      all floating point arguments must be on an even byte boundary
*      calling registers must be in PAD from >f000 to >f060
*
*      floating point representation, radix 100:
*
*      0  = 00 00 xx xx xx xx xx xx
*     +n  = e0 m0 m1 m2 m3 m4 m5 m6
*
*           e0 is int(log[100](n)) + >40
*           m0 - m6 are numbers from >00 to >63 (0 to 99)
*           m0 is most  significant digits of mantissa,
*           m6 is least significant digits of mantissa.
*
*           in normalized numbers, decimal is between m0 and m1
*     -n  is same as "n" except first word is negated ... -(e0 m0)
*
*      examples:
*      decimal                floating point
*
*      7                      >40 >07 >00 >00 >00 >00 >00 >00
*      70                     >40 >46 >00 >00 >00 >00 >00 >00
*      2,345,600              >43 >02 >22 >38 >00 >00 >00 >00
*      23,456,000             >43 >17 >2D >3C >00 >00 >00 >00
*      0                      >00 >00 >xx >xx >xx >xx >xx >xx
*      -7                     >BF >F9 >00 >00 >00 >00 >00 >00
*      -70                    >BF >BA >00 >00 >00 >00 >00 >00
*      -2,345,600             >BC >FE >22 >38 >00 >00 >00 >00
*
*  opcode #0
*  FCOMP    floating point compare
*           input:            r0=0
*                             r2=^float1
*                             r3=^float2
*
*           return:           status reg = AG set iff (float2 > float1)
*                                          EQ set iff (float2 = float1)
*
*  opcode #1
*  FSUB     floating point subtract
*           input:            r0=1
*                             r1=^result
*                             r2=^float1
*                             r3=^float2
*           return:           r0=error code
*                             ^r1 = float2 - float1
*
*  opcode #2
*  FADD     floating point add
*           input:            r0=2
*                             r1=^result
*                             r2=^float1
*                             r3=^float2
*           return:           r0=error code
*                             ^r1 = float2 + float1
*
*  opcode #3
*  FMULT    floating point multiply
*           input:            r0=3
*                             r1=^result
*                             r2=^float1
*                             r3=^float2
```

```
*           return:           r0=error code
*                             ^r1 = float2 * float1
*
*  opcode #4
*  FDIV     floating point divide
*           input:            r0=4
*                             r1=^result
*                             r2=^float1
*                             r3=^float2
*           return:           r0=error code
*                             ^r1 = float2 / float1
*
*  opcode #5
*  PWR      floating point power
*           input:            r0=5
*                             r1=^result
*                             r2=^float1
*                             r3=^float2
*           return:           r0=error code
*                             ^r1 = float2 ^ float1
*
*  opcode #6
*  EXP      floating point e^x
*           input:            r0=6
*                             r1=^result
*                             r2=^float1
*           return:           r0=error code
*                             ^r1 = e ^ float1
*
*  opcode #7
*  LOG      floating point ln(x)
*           input:            r0=7
*                             r1=^result
*                             r2=^float1
*           return:           r0=error code
*                             ^r1 = ln( float1 )
*
*  opcode #8
*  SQR      floating point sqr(x)
*           input:            r0=8
*                             r1=^result
*                             r2=^float1
*           return:           r0=error code
*                             ^r1 = sqr( float1 )
*
*  opcode #9
*  COS      floating point cos(x)
*           input:            r0=9
*                             r1=^result
*                             r2=^float1
*           return:           r0=error code
*                             ^r1 = cos( float1 )
*
*  opcode #10
*  SIN      floating point sin(x)
*           input:            r0=10
*                             r1=^result
*                             r2=^float1
*           return:           r0=error code
*                             ^r1 = sin( float1 )
*
*  opcode #11
*  TAN      floating point tan(x)
*           input:            r0=11
*                             r1=^result
```

```
*                                   r2=^float1
*           return:                 r0=error code
*                                   ^r1 = tan( float1 )
*
*  opcode #12
*  ATN      floating point atn(x)
*           input:                  r0=12
*                                   r1=^result
*                                   r2=^float1
*           return:                 r0=error code
*                                   ^r1 = atn( float1 )
*
*  opcode #13
*  GRI      floating point greatest integer
*           input:                  r0=13
*                                   r1=^result
*                                   r2=^float1
*           return:                 r0=error code
*                                   ^r1 = || float1 ||
*
*  opcode #14
*  CFI      convert floating point to integer
*           input:                  r0=14
*                                   r2=^float1
*           return:                 r0=error code
*                                   r1 = integer
*
*  opcode #15
*  CIF      convert integer to floating point
*           input:                  r0=15
*                                   r1=^result
*                                   r2=integer
*           return:                 r0=error code
*                                   ^r1 = float( integer )
*
*  opcode #16
*  CSINT    convert string to integer
*           input:                  r0=16
*                                   r2=^string
*           return:                 r0=error code
*                                   r1 = integer
*
*  opcode #17
*  CSN      convert string to floating point
*           input:                  r0=17
*                                   r1=^result
*                                   r2=^string
*                                   r3=strlen
*           return:                 r0=error code
*                                   ^r1 = float( string )
*
*  opcode #18
*  CNS      convert float to string
*           input:                  r0=18
*                                   r1=^string
*                                   r2=^float1
*                                   r3=opt1
*                                       bit 0: 0=free form (ignore opt2, opt3)
*                                              1=fixed (opt2, opt3 are field sizes)
*                                       bit 1: 1 for explicit sign
*                                       bit 2: 1 to show sign of positive number
*                                              as a '+' instead of as a space.
*                                              (bit 1 must also be on)
*                                       bit 3: 1 for E-notation output
*                                       bit 4: 1 for extended E-notation
```

```
*                                       (bit 3 must also be 1)
*                               r4=if fixed format, number of places to left
*                                  of decimal point, including explicit sign
*                               r5=if fixed format, number of places to the
*                                  right of decimal point and including decimal
*                                  point
*                               if fixed format, with exponent, r4,r5 exclude *
                                the
*                               3 places for an exponent
*
*          return:              r0=error code
*                               ^r1 = string
*
*
================================================================================
```

```
;===============================================================================
**
**   Filename:            math_i
**   Release:             Version 1.5
**   Date:                26/July/1991
**
**   This file contains the definitions for the MDOS MATH library
**
**   Copyright 1991 by LGMA Products
**
**   Revision History:
**
**   26/Jul/91       Re-released as math_i
**   31/Aug/88       Removed FAC & ARG Addresses
**   22/Feb/88       Added temporary FAC addresses
**   27/Dec/87       Updated FAC & ARG addresses
**   30/Nov/87       Initial Release
MATXOP  EQU     10              ; XOP NUMBER FOR LIBRARY
**
**  Opcodes
**
FCOMP   EQU     0               ; FLOATING COMPARE
FSUB    EQU     1               ; FLOATING SUBTRACT
FADD    EQU     2               ; FLOATING ADD
FMULT   EQU     3               ; FLOATING MULTIPLY
FDIV    EQU     4               ; FLOATING DIVIDE
PWR     EQU     5               ; POWER FUNCTION
EXP     EQU     6               ; EXPONENTIAL
LOG     EQU     7               ; LOGORITHM
SQR     EQU     8               ; SQUARE ROOT
COS     EQU     9               ; COSINE
SIN     EQU     10              ; SINE
TAN     EQU     11              ; TANGENT
ATN     EQU     12              ; ARC-TANGENT
GRI     EQU     13              ; FLOATING POINT TO GREATEST INTEGER
CFI     EQU     14              ; CONVERT FLOATING TO INTEGER
CIF     EQU     15              ; CONVERT INTEGER TO FLOATING
CSINT   EQU     16              ; CONVERT STRING TO INTEGER
CSN     EQU     17              ; CONVERT STRING TO NUMBER
CNS     EQU     18              ; CONVERT FLOAT TO STRING
;===============================================================================
```

Key + mouse

```
*****************************
*
*
* keyboard    XOP @FIVE,0
*
*      IN:     R0Lsb =  scan mode   (like ti modes, 0-5)
*
*      OUT:    R0Lsb =  returned mode  (same as >8374 in 99/4a)
*              R1Msb =  returned scan code (same as >8375)
*              R2Msb =  joystick Y value     (same as >8376)
*              R2Lsb =  joystick X value     (same as >8377)
*           EQ bit in Status register set if there's a new key in R0
*
*              mode 7 = break key check  return EQ=1 if Break on
*              mode 8 = raw scan code returned in R1h
*                       raw code returned is >FF if there was no code in **
                        buffer
*
```

```
*******************************************************************************
**
**   Filename:            keyboard_i
**   Release:             Version 1.5
**   Date:                26/July/1991
**
**   This file contains the definitions for the MDOS Keyboard library
**
**   Copyright 1991 by LGMA Products
**
**   Revision History:
**
**   26/Jul/91       Re-release as keyboard_i
**   20/Dec/87       Added command line address
**   30/Nov/87       Initial Release
**
KEYXOP  EQU     5               ; XOP NUMBER FOR LIBRARY
**
**  KEYBOARD MODES
**
MOLAST  EQU     0               ; USE LAST KEYBOARD MODE (MUST BE GOOD!)
MOLEFT  EQU     1               ; USE LEFT SIDE OF KEYBOARD
MRIGHT  EQU     2               ; USE RIGHT SIDE OF KEYBOARD
MSTAND  EQU     3               ; STANDARD MODE
MPASCA  EQU     4               ; PASCAL MODE
MBASIC  EQU     5               ; BASIC MODE
MBREAK  EQU     7               ; BREAK KEY CHECK
MRAW    EQU     8               ; RAW KEY CHECK
**
**  Command Line (MDOS)
**
CMDADR  EQU     >0128           ; COMMAND LINE ADDRESS/ADDRESS
*******************************************************************************
```

```
*****************************************************************************
**
**   Filename:            mouse_i
**   Release:             Version 1.5
**   Date:                26/August/1991
**
**   This file contains the definitions for Bruce Hellstrom's
**   Mouse Driver
**
**   Copyright 1991 by LGMA Products
**
**   Revision History:
**
**   26/AUG/91       Initial Release
**

COMREG  EQU >F200            ; COMMAND REGISTER
MSPD    EQU >F201            ; SPEED
MSX     EQU >F202            ; MOUSE X COORD
MSY     EQU >F204            ; MOUSE Y COORD
BUT1    EQU >F206            ; BUTTON 1 (LEFT)
BUT2    EQU >F208            ; BUTTON 2 (MIDDLE)
BUT3    EQU >F20A            ; BUTTON 3 (RIGHT)
MSSETX  EQU >F20C            ;
MSSETY  EQU >F210            ;

**
** Function Codes for driver (in command register)
**

DEACTIVE EQU >FFFF           ; DEACTIVATE THE MOUSE
ACTIVE   EQU >0001           ; ACTIVATE THE MOUSE
HIDEMOUS EQU >0002           ; HIDE THE MOUSE
SHOWMOUS EQU >0003           ; SHOW THE MOUSE
SETSPEED EQU >0400           ; SET MOUSE SPEED
DISABLE  EQU >0005           ; DISABLE MOUSE COMMAND
ALTBITMP EQU >0600           ; TURN ON ALTERNATE BITMAP

**
** Bit Maps for various sprite definitions:
**
ARROWP   EQU >0000           ; NORMAL ARROW
SLEEPYP  EQU >0001           ; SLEEPY POINTER
TARGETP  EQU >0002           ; TARGET POINTER
MENUP    EQU >0003           ; MENU POINTER

**
** Task page 0 equates for mouse
**

ENABLEB  EQU >0062           ; BYTE TO REENABLE THE MOUSE
LOADEDB  EQU >0063           ; WHETHER MOUSE IS LOADED OR NOT
*****************************************************************************
```

MISC

```
;=============================================================================
;*   Filename:            Assembly_s
;*   Release:             Version 1.5
;*   Date:                26/July/1991
;*
;*   This file contains the definitions for Interfacing TIC with
;*   Assembly Language Subroutines
;*
;*   Copyright 1991 by LGMA Products
;*
;*   Define the Stack Frame:
;*
TIC_LOCAL    EQU        0          /* Start of Local Variable Frame */
TIC_PARAM1   EQU       -12         /* Parameter 1 */
TIC_PARAM2   EQU       -14         /* Parameter 2 */
TIC_PARAM3   EQU       -16         /* Parameter 3 */
TIC_PARAM4   EQU       -18         /* Parameter 4 */
TIC_PARAM5   EQU       -20         /* Parameter 5 */
TIC_PARAM6   EQU       -22         /* Parameter 6 */
TIC_PARAM7   EQU       -24         /* Parameter 7 */
TIC_PARAM8   EQU       -26         /* Parameter 8 */
TIC_PARAM9   EQU       -28         /* Parameter 9 */
;*
;*  Register Definitions:
;*
TEMP#REG1    EQU       1           /* 1st Temporary Register */
TEMP#REG2    EQU       2           /* 2nd Temporary Register */
TEMP#REG3    EQU       3           /* 3rd Temporary Register */
SAVD#REG1    EQU       4           /* 1st Saved Temporary Register */
SAVD#REG2    EQU       5           /* 2nd Saved Temporary Register */
SAVD#REG3    EQU       6           /* 3rd Save Temporary Register */
FRAME#PTR    EQU       7           /* Pointer to start of Frame */
RETURN#RG    EQU       8           /* Return Register */
CALL#SUBR    EQU       12          /* Call a Subroutine */
RETURN#SB    EQU       13          /* Return from a Subroutine */
STACK#PTR    EQU       14          /* Stack Pointer */
C#SWITCH     EQU       15          /* C$SWCH Routine */
;=============================================================================
```

MISC. NOTES

```
================================================================================
COLLECTED NOTES FROM PAUL CHARLTON ON 9640 ASSEMBLY AND LINKER
(Collected on CompuServe in October/November 1987)
--------------------------------------------------------------------------------
DESCRIPTION OF LINKER PROGRAM:

    this is a linker which runs from MDOS and creates MDOS program images
    invoke with either "LINK" or "LINK <command filename>"
      command file> is redirected keyboard input
    commands are:
    "filename" <-object file to load
    "@filename" <-save program image stating with "filename"
    "!"   exit to mdos
    "?"   list "ref/def" table. "*" before unresolved refs
    have fun....$15 donation to author suggested
    Paul Charlton 1599 Tibbits Ave Troy, Ny 12180-3723
--------------------------------------------------------------------------------
REPLY TO QUERY BY DAVE RAMSEY ON USING THE LINKER:

RORG the object files...AORG files don't set the max address counter
which is used by the SAVE routine (it only looks at the RELATIVE load
address defined by the loader)
also, RORG files will automatically start at >0400 and there
is no need to define such things as "sfirst,sload,slast"
load the file with the entry point 1st...
LZW compression should _never_ increase file size if the bit-codes
are placed into variable fields within the files bytes
(as opposed to making all codes 12bit regardless of whether all codes
 are used ...CIS addressed this quite well in the GIF spec)
--------------------------------------------------------------------------------
REPLY TO QUERY ON 9640 PROGRAM HEADER BYTES:

the header for Geneve programs is quite similar to TI header...
1st byte >00 or non->00, more to load flag if non-zero
2nd byte, 'F' or 'G' (this is what tells the loader it's a 9640 mode pgm)
  'F' means to load into fast memory if possible
  'G' means slow memory
bytes 3 & 4 : length of current pgm image (not including 6 header bytes!)_
bytes 5,6   : load address of current pgm image
note that the entry address for all programs is at >0400 and that no
application should need to turn off interrupts.
================================================================================
L.D.O.M. October 13, 1996
<EOF>
```