COS computes cosine ARITHMETIC

Synopsis: cos: proc (floating) returns (floating);

Usage: result = cos (num);

COS returns the cosine of NUM, where NUM is the angle in radians.

Example:

```
dcl (x, y) floating;

x = cos (y);
```

See also: SIN
TAN
ATN

EXIT terminates program execution PROGRAM TERMINATION

Synopsis: exit: proc (fixed);

Usage: call exit (status);

where STATUS is the program termination status returned to the Monitor. STATUS should be a zero if the program completed successfully or -1 if the program aborted. Control is returned to the Monitor following the EXIT statement.

Example:

```
when break then call exit (0); /* terminate on BREAK */
...

if find_device (dev) = 0 then do; /* DEV not configured */
   print 'Device is not configured in system';
   call exit (-1); /* abort program */
end;
```

See also: STOP

EXP — computes natural anti-logarithm — ARITHMETIC

Synopsis: `exp: proc (floating) returns (floating);`

Usage: `result = exp (num);`

EXP returns the natural anti-logarithm of NUM. This is equal to e (e = 2.71828...) raised to the power of NUM.

XPL does not provide functions for raising 10 to the power of X or Y to the power of X, but these functions can be derived from EXP as follows:

```
ten_to_the_x (x)  = exp (log (10)*x)
x_to_the_y (x, y) = exp (log (x)*y)
```

Example:

```
dcl (x, y) floating;

y = exp (x)*10;
```

See also: LOG
SQR

EXPORT — copies an array into external memory — BLOCK MANIPULATION

Synopsis: `export: proc (fixed, fixed, fixed array, fixed);`

Usage: `call export (sector, word, buffer, length);`

This routine copies the first LENGTH words from BUFFER into external memory starting at the address specified by SECTOR and WORD (WORD is not restricted to 256).

The LOCATION and ADDR functions can be used to create pointers to specific array or memory locations.

Example:

```
call export (0, 0, buf, 256);
call export (sector, wd, loc (addr (buf (ptr))), len);
```

See also: EXTSET
IMPORT

Synopsis: extread: proc (fixed, fixed, fixed array);

Usage: call extread (ms_sector, ls_sector, ext_data);

where MS_SECTOR and LS_SECTOR form a 32-bit word pair that identifies the device and sector number that is to be read. The upper eight bits (byte) of this word pair specify the storage device number and the lower 24-bits specify the sector number on that device. EXT_DATA is a fixed array which contains the following information:

```
ext_data (0) = base sector in external memory
ext_data (1) = word offset from base sector
ext_data (2) = number of sectors to read
ext_data (3) = number of words beyond last sector
               to read
```

Data will be read from a storage device at MS_SECTOR and LS_SECTOR into external memory. The location in external memory to write to and the amount of data to transfer are both specified in the array EXT_DATA. Neither the word offset nor the word length are restricted to 256.

NOTE: If the device you are using with EXTREAD is a SCSI device, you must insert :-XPL:SCSISWAP into your program.

Example:

```
dcl info (3) fixed; /* array for using EXTREAD */

info (0) = 104; /* sector 104 of external memory */
info (1) = 0;   /* no word offset */
info (2) = 0;   /* no sectors */
info (3) = file_len; /* read entire file */

/* read file from disk into external memory */

call extread (ms_file_start, ls_file_start, info);
```

See also: EXTWRITE
READDATA
WRITEDATA

Synopsis: extset: proc (fixed, fixed, fixed, fixed);

Usage: call extset (sector, word, length, value);

This routine assigns VALUE to LENGTH words of external memory starting at the address specified by SECTOR and WORD. WORD is not restricted to 256.

Example:

```
call extset (0, 0, 256, 0);
call extset (sector, word, len, i);
```

See also: EXPORT
IMPORT

Synopsis: extwrite: proc (fixed, fixed, fixed array);

Usage: call extwrite (ms_sector, ls_sector, ext_data);

where MS_SECTOR and LS_SECTOR form a 32-bit word pair that identifies the device and sector number where data is to be written. The upper eight bits (byte) of this word pair specify the storage device number and the lower 24-bits specify the sector number on that device. EXT_DATA is a fixed array which contains the following information:

```
ext_data (0) = base sector in external memory
ext_data (1) = word offset from base sector
ext_data (2) = number of sectors to write
ext_data (3) = number of words beyond last sector
               to write
```

Data will be written from external memory to a storage device at MS_SECTOR and LS_SECTOR. The location in external memory to read from and the amount of data to transfer are both specified in the array EXT_DATA. Neither the word offset nor the word length is restricted to 256.

NOTE: If the device you are using with EXTWRITE is a SCSI device, you must insert :-XPL:SCSISWAP into your program.

Example:

```
dcl info (3) fixed; /* array for using EXTWRITE */

info (0) = ptr; /* sector number in external memory */
info (1) = words; /* word offset */
info (2) = 24;    /* read 24 sectors */
info (3) = 0;     /* no extra word length */

/* write data from external memory to disk location */

call extwrite (f#ms_sector, f#ls_sector, info);
```

See also: EXTREAD
READDATA
WRITEDATA

FIND_DEVICE   finds the configuration of a device   SYSTEM

Synopsis: find_device: proc (fixed) returns (pointer);

Usage: ptr = find_device (dev_num);

where DEV_NUM is a valid system device number. If a valid device number is passed, and that device is configured in the system, FIND_DEVICE returns an absolute memory pointer to the storage device table entry for that device. Otherwise, it returns a null pointer.

Example:

```
p = find_device (7);  /* look for the W1 Winchester */
if p <> null then do; /* the device was found */
   ...
end;
else print 'W1 Winchester disk is not configured.';
```

IMPORT   copies from ext memory into an array   BLOCK MANIPULATION

Synopsis: import: proc (fixed, fixed, fixed array, fixed);

Usage: call import (sector, word, buffer, length);

This routine copies LENGTH words into BUFFER from external memory starting at the address specified by SECTOR and WORD. WORD is not restricted to 256.

The LOCATION and ADDR functions can be used to create pointers to specific array or memory locations.

Examples:

```
call import (0, 0, buf, 256);
call import (sector, 0, loc (addr (buf (ptr))), len);
```

See also: EXPORT
EXTSET

Synopsis: int: proc (floating) returns (fixed);

Usage: result = int (num);

The INT function converts the floating point parameter NUM into a fixed point number, which is returned. NUM is rounded towards negative infinity. That is, positive numbers are truncated (the fractional part is dropped), while negative numbers are set to the nearest more negative integer (int (-1.1) = -2).

Example:

```
dcl (x, y) floating;
dcl i      fixed;

i = int (x + y); /* store integer sum in variable I */
```

Synopsis: location: proc (pointer) returns (fixed array);
location: proc (pointer) returns (floating array);

Usage: call proc_name (location (expression));

The LOCATION function is used to reference absolute locations of memory during a procedure call. LOCATION can only appear in an actual parameter list during a procedure call. The absolute memory location is passed to the procedure as the start of an array (either fixed or floating point). LOCATION can be shortened to LOC.

LOCATION is often used in conjunction with the ADDR function in order to read from or write to an array starting at an element other than element zero.

If a program tries to read a location of memory that does not exist, the computer will halt and the program will stop running.

Example:

```
/* Read a sector from disk into memory starting at
   location 8192. */

call readdata (ms_sec, ls_sec, loc (8192), 256);

/* This example fills a string with nulls starting
   at word one of the string, leaving word zero
   (the string length) intact. */

call blockset (loc (addr (string (1))), str_len, 0);
```

See also: ADDR
CORE

Synopsis: log: proc (floating) returns (floating);

Usage: result = log (num);

LOG returns the natural logarithm (i.e., log base e) of NUM. NUM must be a positive non-zero number.

XPL does not provide functions for log base 10 or log base X, but these functions can be derived from LOG as follows:

```
log 10 (x)    = log (x)/log (10)
log x (x, y) = log (y)/log (x)
```

Example:

```
dcl (x, y) floating;

x = log (y);
```

See also: EXP

Synopsis: pbyte: proc (fixed array, fixed, fixed);

Usage: call pbyte (string, byte_num, value);

where STRING is an XPL string, BYTE_NUM is the byte position in the array (starting from zero), and VALUE is the number to store there. The lower 8 bits of VALUE will be written to the byte location specified by BYTE_NUM. Only the appropriate upper or lower byte of the array element is altered.

PBYTE is usually used with fixed arrays that are XPL strings, as each character is represented by one byte (two characters per word). When using PBYTE, the first byte (byte position 0) corresponds to the lower half of word one of the array. This leaves the string length in the first word (element 0) of the array.

PBYTE can be used with fixed arrays that are not strings, to manipulate data elements by byte rather than by word. In this case the LOCATION and ADDR functions must be used to access the first word (element 0) of the array, as shown below:

```
dcl list (100) fixed array;

call pbyte (loc (addr (list (0)) - 1), byte_num, i);
```

Example:

```
dcl line (8) fixed array; /* string of 16 characters */
dcl i        fixed;

do i = 0 to 15;
   call pbyte (line, i, a.x); /* fill line with X's */
end;

call pbyte (line, 9, a.sp); /* put a space in the middle */
```

See also: BYTE

Synopsis: polyread: proc (fixed, fixed, fixed array, fixed);

Usage: call polyread (ms_sector, ls_sector, poly_data, channel);

where MS_SECTOR and LS_SECTOR form a 32-bit word pair that identifies the device and sector number that is to be read. The upper eight bits (byte) of this word pair specify the storage device number and the lower 24-bits specify the sector number on that device. POLY_DATA is a fixed array which contains the following information:

```
poly_data (0) = base sector in polyphonic memory
poly_data (1) = word offset from base sector
poly_data (2) = number of sectors to read
poly_data (3) = number of words beyond last sector
                to read
```

Data will be read from a storage device at MS_SECTOR and LS_SECTOR into polyphonic sampling memory. The location in polyphonic memory to write to and the amount of data to transfer are both specified in the array POLY_DATA. Neither the word offset nor the word length is restricted to zero. The data will be transferred through the indicated polyphonic CHANNEL (0-31). CHANNEL is usually zero; an interrupt handler could pass the channel being used at the time of interrupt (read ("155")).

NOTE: If the device you are using with POLYREAD is a SCSI device, you must insert :-XPL:SCSISWAP into your program.

Example:

```
dcl info (3) fixed; /* array for using POLYREAD */

info (0) = 10; /* sector 10 of poly memory */
info (1) = 0;  /* no word offset */
info (2) = 0;  /* no sectors */
info (3) = file_len; /* read the whole file */

/* read file from disk into polyphonic memory */

call polyread (ms_file_start, ls_file_start, info, 0);
```

See also: POLYWRITE
READDATA
WRITEDATA

Synopsis: polywrite: proc (fixed, fixed, fixed array, fixed);

Usage: call polywrite (ms_sector, ls_sector, poly_data, channel);

where MS_SECTOR and LS_SECTOR form a 32-bit word pair that identifies the device and sector number where data is to be written. The upper eight bits (byte) of this word pair specify the storage device number and the lower 24-bits specify the sector number on that device. POLY_DATA is a fixed array which contains the following information:

```
poly_data (0) = base sector in polyphonic memory
poly_data (1) = word offset from base sector
poly_data (2) = number of sectors to write
poly_data (3) = number of words beyond last sector
                to write
```

Data will be written from polyphonic sampling memory to a storage device at MS_SECTOR and LS_SECTOR. The location in polyphonic memory to read from and the amount of data to transfer are both specified in the array POLY_DATA. Neither the word offset nor the word length is restricted to 256. The data transfer will occur through the indicated polyphonic CHANNEL (0-31). CHANNEL is usually zero; an interrupt handler could pass the channel being used at the time of interrupt (read ("155")).

NOTE: If the device you are using with POLYWRITE is a SCSI device, you must insert :-XPL:SCSISWAP into your program.

Example:

```
dcl info (3) fixed; /* array for using POLYWRITE */

info (0) = ptr; /* sector number in polyphonic memory */
info (1) = words; /* word offset */
info (2) = 12;    /* read 12 sectors */
info (3) = 0;     /* no extra word length */

/* write data from polyphonic memory to disk location */

call polywrite (f#ms_sector, f#ls_sector, info, 0);
```

See also: POLYREAD
READDATA
WRITEDATA

RCVDCHARACTER gets last character from terminal interrupt SYSTEMS

Synopsis: rcvdcharacter: proc returns (fixed);

Usage: ch = rcvdcharacter;

RCVDCHARACTER returns the last character that was received from a terminal interrupt. It is normally used in a WHEN TTIINT statement (the terminal input interrupt).

Example:

```
when ttiint then begin;
   dcl ch fixed;  /* character typed bu user */

   ch = rcvdcharacter;  /* get the character */
   print 'You just typed: ', char (ch);
end;
```

READ reads a word from an interface device HARDWARE

Synopsis: read: proc (fixed) returns (fixed);

Usage: i = read (device_number);

The READ function reads a value from the interface module specified by DEVICE_NUMBER and returns that value to the program. DEVICE_NUMBER can be a constant expression or a variable expression, although the READ will be much slower in the latter case.

If an attempt is made to read a device that is not in the system, the computer will halt.

Example:

```
dcl timer literally '"03"';
dcl i     fixed;

i = read (timer);
```

See also: WRITE

Synopsis: readdata: proc (fixed, fixed, fixed array, fixed);

Usage: call readdata (ms_sector, ls_sector, buffer, length);

where MS_SECTOR and LS_SECTOR form a 32-bit word pair that identifies the device and sector number that is to be read. The upper eight bits (byte) of this word pair specify the storage device number and the lower 24-bits specify the sector number on that device. LENGTH words of data will be read into BUFFER from this device and sector location.

When using READDATA to access Winchester systems with more than one drive attached to a device, XPL will automatically determine which physical Winchester disk contains the specified logical sector.

NOTE: If the device you are using with READDATA is a SCSI device, you must insert either :-XPL:SCSI or :-XPL:SCSISWAP into your program.

Example:

```
dcl buf (256) fixed;

call readdata (shl (2, 8), 84, buf, 256);
```

In this example, one sector (256 words) of the floppy disk in the F0 drive is read. The number 2 is in the upper byte (device specifier for F0), and the sector address is 84.

See also: WRITEDATA

Synopsis: rot: proc (fixed, fixed) returns (fixed);

Usage: result = rot (number, bit_count);

ROT returns a value that is equal to NUMBER rotated left BIT_COUNT bit positions. BIT_COUNT must be in the range 0-15. Each bit of VALUE will be shifted to the left BIT_COUNT positions, with the most significant bit rotating into the least significant position. Notice the following example, where the number 5000 is rotated 4 positions:

```
i = 5000;         /* i = 0001001110001000 */
j = rot (i, 4);   /* j = 0011100010000001 */
```

Example:

```
dcl (i, j) fixed;

i = rot (j, 8); /* swap the upper and lower bytes */
```

See also: SHL
SHR

Synopsis: set_curdev: proc (fixed) returns (boolean);

Usage: if set_curdev (dev_num) then ...

where DEV_NUM is a valid system device number. SET_CURDEV sets the current device to be the passed device number. If the operation is successful, a TRUE is returned. If an invalid device number is passed, or if the device is not configured in the system, a FALSE is returned and the current device is not changed.

Example:

```
if not set_curdev (dev_num) /* could not change device */
then do;
   print '*** System Error!';
   print 'Could not change current device to ', dev_num;
end;
else do;  /* current device was set to DEV_NUM */
   ...
end;
```

Synopsis: shl: proc (fixed, fixed) returns (fixed);

Usage: result = shl (number, bit_count);

SHL returns a value that is equal to NUMBER shifted to the left BIT_COUNT bit positions. BIT_COUNT must be in the range 0-15. Bits shifted off the left end will be lost, and bits shifted into the right end will be zeros. Notice the following example, where the number 5000 is shifted left 4 positions:

```
i = 5000;          /* i = 0001001110001000 */
j = shl (i, 4);    /* j = 0011100010000000 */
```

Example:

```
dcl dev         fixed; /* device number */
dcl ms_sector   fixed; /* MS word of starting sector */
dcl ls_sector   fixed; /* LS word of starting sector */
dcl sec         fixed; /* number of sectors to read */
dcl buf (2048)  fixed; /* data buffer */

/* The following code sets up a word pair that has a
   device number in the upper byte and the starting
   sector of a file in the lower 24 bits.  This is the
   standard way of specifying a file location. */

dev = shl (dev, 8); /* put device in the upper byte */

/* Use AND to make sure that MS_SECTOR has only the
   bottom 8 bits, then OR the device into it.  */

ms_sector =  (dev or (ms_sector and "377"));

/* Now read the first SEC sectors of the file.
   Notice the word length is given by multiplying the
   sector length by 256, or SHL by 8. */

call readdata (ms_sector, ls_sector, buf, shl (sec, 8));
```

See also: SHR
ROT

Synopsis: shr: proc (fixed, fixed) returns (fixed);

Usage: result = shr (number, bit_count);

SHR returns a value that is equal to NUMBER shifted to the right BIT_COUNT bit positions. BIT_COUNT must be in the range 0-15. Bits shifted off the right end will be lost, and bits shifted into the left end will be zeros. Notice the following example, where the number 5000 is shifted right 4 positions:

```
i = 5000;         /* i = 0001001110001000 */
j = shr (i, 4);   /* j = 0000000100111000 */
```

Example:

```
dcl device  fixed; /* device number */
dcl words   fixed; /* number of words */
dcl sectors fixed; /* number of sectors */

/* extract the device from the upper byte of word pair
   identifying file location */

device = shr (f#ms_sector, 8);

/* sectors is words divided by 256, or SHR of 8 */

sectors = shr (words, 8);
```

See also: SHL
ROT

SIN computes sine ARITHMETIC

Synopsis: `sin: proc (floating) returns (floating);`

Usage: `result = sin (num);`

SIN returns the sine of NUM, where NUM is the angle in radians.

Example:

```
dcl (x, y) floating;

x = sin (y)*y;
```

See also: COS
TAN
ATN

SQR computes square root ARITHMETIC

Synopsis: `sqr: proc (floating) returns (floating);`

Usage: `result = sqr (num);`

SQR returns the square root of NUM, where NUM must be a positive number.

XPL does not provide functions for cube roots or the Xth root of Y, but these functions can be derived from LOG and EXP as follows:

```
cube_root (x)         = exp (log (x)/3.0)
xth_root_of_y (x, y)  = exp (log (y)/x)
```

Example:

```
dcl (x, y) fixed;

print sqr (x);
y = sqr (abs (x));
```

See also: ABS
LOG
EXP

Usage:

```
stop;
stop (value);
```

The STOP statement is used to halt the computer at a predetermined point in a program. If STOP is used with a fixed point parameter, that VALUE will be written to the Hand Operated Processor (HOP) upon execution of the STOP statement. Pressing the SYNC button on the HOP will cause the program to continue from that point, or control can be returned to the Monitor by pressing the Load button. The STOP statement is most often used for debugging purposes.

Example:

```
dcl status fixed;

status = test_device; /* call testing procedure */

if status <> 0 then do;
   stop ("10"); /* write a value to HOP to mark this */
end;
```

See also: EXIT