VGER

LAST REVISED 6/18/81

INDEX

TRANSITIONS

COORDINATES

WRITING

UTILITIES

CONSTANTS

EXAMPLE

TIMER

Returns to task when timer decrements to zero
16 bit timer with each tic = 1/60th of a second.
Retains overflow time base when timer expires.

TIMER-ON ---
Turn on timer check

TIMER-OFF ---
Turn off timer check

TIMER! n ---
Store timer value n into VECTOR.

TIMER!-ON n ---
Turn on timer check
Store timer value n into VECTOR.

TIMER? --- n
Returns boolean n, true if timer went to zero

FLAG

Returns to task when contents of flag address is non zero.
When flag is detected flag byte is zeroed.

FLAG-ON ---
Turn on flag check.

FLAG-OFF ---
Turn off flag check

FLAG! n ---
Store flag address n into vector

FLAG!-ON n ---
Turn on flag check
Store flag address n into vector

FLAG? --- n
Returns boolean n, true if flag was set

LIMITS

Returns to task when vector hits limit.
Normal limit - object coordinate set to limit
Timebase is lost
Limit with Back Out - object is vectored back away from
limit and Timebase is retained.

~~LIMIT-ON~~ ---
Turn on limit check

LIMIT X-ON
LIMIT Y-ON
LIMIT XY-ON

LIMITBOUT-ON ---
Turn on limit check with backout when limit attained.
Must ~~Set~~ Limit on for this towork

LIMIT-OFF ---
Turn off limit check

Limit x -off
Limit y -off
Limit xy-off

LIMHX! n ---
Store limit value n for high X into vector

LIMLX! n ---
Store limit value n for low X into vector

LIMHY! n ---
Store limit value n for high Y into vector

LIMLY! n ---
Store limit value n for low Y into vector

LIMHXLXHYLY! a b c d ---
Store Limit value a for high X, b for low X, c for high Y and d for low Y into vector.

LIMIT? --- n
Return boolean n, true if any limit attained.

LIMITX? --- n
Return boolean n, true if high X or low X limit attained.

LIMITY? --- n
Returns boolean n, true if high Y or low Y limit attained.

LIMITHX? --- n
Returns boolean n, true if high X limit attained.

LIMITLX? --- n
Returns boolean n, true if low X limit attained.

LIMITHY? --- n
Returns boolean n, true if high Y limit attained.

LIMITLY? --- n
Returns boolean n, true if low Y limit attained.

LIMITS CONT.

LIMITHX@ n --- p
Read from vector n and leave limit high X p

LIMITLX@ n --- p
Read from vector n and leave limit low X p

LIMITHY@ n --- p
Read from vector n and leave limit high Y p

LIMITLY@ n --- p
Read from vector n and leave limit low Y p

DESTINATION

Returns to task when coordinates crosses destination value.
Does not affect the position of the write.

DESTX-ON ---
Turn on destination X check

DESTX-OFF ---
Turn off destination X check

DESTY-ON ---
Turn on destination Y check

DESTY-OFF ---
Turn off destination Y check

DESTX! n ---
Store destination X coordinate n into vector

DESTY! n ---
Store destination Y coordinate n into vector

DESTX!-ON n ---
Turn on destination X check
Store destination X coordinate n into vector

DESTY!-ON n ---
Turn on destination Y check
Store destination Y coordinate n into vector

DESTX? --- n
Returns boolean n, true if object crossed destination X

DESTY? --- n
Returns boolean n, true if object crossed destination X

DEST? --- n
Returns boolean n, true if object crossed destination X or Y

DESTX@ n --- p
Read from vector n and leave destination X p

DESTY@ n --- p
Read from vector n and leave destination Y p

INTERCEPT

Return to task when intercept is detected during a write.

INTERCEPT-ON ---
Turn on Intercept check

INTERCEPT-OFF ---
Turn off intercept check

INTERCEPT? --- n
Returns boolean n, true if intercept was detected

USERS INTERUPT HOOK

User can put in a routine to run at vector management interrupt level.
Registers input and output from Hook routine.

B = Timebase used
C = Timebase not used
DE = new X — ONLY If vector
HL = new Y
IY = vector address

If used with wait only necessary to return IY.

HOOK-ON ---
Turn on hook check

HOOK-OFF ---
Turn off hook check

HOOK! n ---
Store address of hook routine n into vector

HOOK!-ON n ---
Turn on hook check
Store address of hook routine n into vector

A to B CALCULATION

A->DEST n ---

Calculate the deltas less than n to travel from X Y coordinate to X Y destination.
Use DESTX! DESTY!
Delta maximum n is represented by a hex word with the top byte the whole number and the bottom byte the fraction.
Deltas can range from n to n/2
Stores new deltas into vector
Stores timer value into vector
Turns on timer check
Use TIMER? to determine if destination is reached

A->B p q n ---

Calculates the deltas less than n, to travel from X Y coordinate to p,q destination
Delta maximum n is represented by a hex word with the top byte the whole number and the bottom byte the fraction.
Deltas can range from n to n/2.
Stores new deltas into vector
Stores timer value into vector
Turns on timer check
Use TIMER? to determine if destination is reached

GO AND WAIT

Leave task and stay in vector management at interrupt level until a state transition.
When a state transition is detected task interpreter continues at the following task verb.

GO ---

Start execution with vectoring and writing.

WAIT ---

Turn off vectoring and writing process
Start execution

RECTANGULAR COORDINATE SYSTEM

The coordinate system is zero, centered with the positive x and positive y quadrant in the upper right corner of the screen.
X Range is -160 to 159 or in hex -A0 to 9F
Y Range is -100 to 99 or in hex -64 to 63

```
X!              n ---
    Store X coordinate n into vector

Y!              n --
    Store Y coordinate n into vector

XY!             n m ---
    Store X coordinate n and Y coordinate m into vector

X@              n --- p
    Read from vector n and leave X coordinate p

Y@              n --- p
    Read from vector n and leave Y coordinate p

GETX            n ---
    Read X coordinate from vector n and store into vector

GETY            n ---
    Read Y coordinate from vector n and store into vector

GETXY           n ---
    Read X Y coordinate from vector n and store into vector

ZEROXY          ---
    Zero X Y coordinates of vector
```

Velocity and acceleration must be entered as a hex word with the top byte the whole number and the bottom byte the faction.

```
DX!             n ---
    Store X delta n into vector

DY!             n ---
    Store Y delta n into vector

DXDY!           n ---
    Store X delta and Y delta n into vector

GETDXDY          n ---
    Read delta X and Y from vector n and store into vector.
```

RECTANGULAR COORDINATE SYSTEM cont.

AX! n ---
Store X acceleration n into vector

AY! n ---
Store Y acceleration n into vector

AXAY! n ---
Store X acceleration and Y acceleration n into vector

DX@ n --- p
Read from vector n and leave delta x p

DY@ n --- p
Read from vector n and leave delta y p

AX@ n --- p
Read from vector n and leave acceleration y p

AY@ n --- p
Read from vector n and leave acceleration y p

GETDXDYAXAY n ---
Read delta x delta y acceleration x acceleration y from vector n and store into vector.

ZERODXDYAXAY ---
Zero delta X, delta Y, acceleration X, and acceleration Y of vector

POLAR COORDINATE SYSTEM

GETCOS s a --- d
Calculates delta d from speed s and angle a.

ANGLE ---
Calculates deltas and accelerations for X and Y using polar velocity, polar acceleration and angle.

TURN d t ---
Creates a turn from the current angle to the current angle +d in time t. t is rounded off to the nearest power of 2.
Uses rectangular acceleration and it's own timer.

RADIUSTURN d r --- t
Creates a turn of angle difference d for a constant radius r.
Computes turn time t and calls turn.
t is returned. Speed is in VPLRVEL.
Radius r is unsigned 8 bits (# of pixels)

Angles are represented by a value from 0 to 255. Angle 0 is the direction of no y and positive x. Angle 64 is 90 degrees clockwise. Angles incremet in a clockwise direction.

ANGLE! n ---
Store polar angle n into vector.

ANGLE@ v --- n
Read angle n from vecotr V.

Polar velocity and acceleration must be entered as a hex word with the top byte the whole number and the bottom byte the fraction.

POLARVEL! n ---
Store polar velocity n into vector.

POLARACC! n ---
Store polar acceleration n into vector.

POLARVEL@ v --- n
Read polar velocity n from vector v.

POLARACC@ v --- n
Read polar acceleration n from vector v.

PATTERNS

```
PATTERN!      n ---
    Store pattern address n into the current vector
    All patterns must have a 4 byte header
         X offset B,
         Y offset B,
         X byte size B,
         Y line size B,
    Followed by pattern source
```

If xpand is used the last 3 pixels of the pattern have to be 0 for flush. Non expanded patterns are automatically flushed.

Pattern creation helpers are provided.

```
PATTERN       --- n
    Sets base to decimal, makes name n a data statement

~             ---
    Marks stack sets base

^             ---
    Store bytes on stack from ~ in RAM as pattern, resets base

QUADPAT       ---
    Specifies base 4 to be used between ~ and ^
    Does not change current base

BINPAT        ---
    Specifies base 2 to be used between ~ and ^
    Does not change current base
```

Example: Base 4 pattern

```
    PATTERN DEMO-PAT-QUAD 0 B, 0 B, 2 B, 2 B, QUADPAT
        ~ 3321  1233 ^
        ~ 3321  1233 ^
```

Each number represents a pixel with base 4

Base 2 PATTERN

```
    PATTERN DEMO-PAT-BIN 0 B, 0 B, 2 B, 2 B, BINPAT
        ~ 11111001   01101111 ^
        ~ 11111001   01101111 ^
```

BASE 16 PATTERN

```
    PATTERN DEMO-PAT-HEX 0 B, 0 B, 2 B, 2 B, HEX
        F9 B,  6F B,
        F9 B,  6F B,
```

All of the above examples represent the same pattern.

```
PDUMP         n ---
    Dump pattern n in Pattern Base, (ie Quadpat or
    Binpat) Assumes pattern has a header of
    x offset B, y offset B, x byte size B, y line size B,
```

ANIMATION TYPES

VGER supports animation, rotation and perspective alone or in combinations. All patterns are assumed to have a header of x offset B, y offset B, x byte size B, y line size B, .

PERROTANM-OFF ---
Turn off all animation types

ROT!-ON n s ---
Turn on rotation
Store rotation table address n into current vector
Store rotation shift amount s into vector, shift value takes maximum angle (256) and shifts it down until it equals the number of rotation patterns.

128 patterns =	1
64 patterns =	2
32 patterns =	3
16 patterns =	4
8 patterns =	5
4 patterns =	6

Rotation Options
Flip [redacted]
Flop [redacted]
Flip-Flop [redacted]
NORMAL
Rotation uses angle to determine rotation table index.

Example:

```
    DATA ROT-TBL          (Rotation table)
    (Flip Flop Options) (Pattern address)
NORMAL    [redacted] B,          ROT-PAT-1
FLIP-FLOP [redacted] B,          ROT-PAT-2
FLOP      [redacted] B,          ROT-PAT-1
FLIP      [redacted] B,          ROT-PAT-2

    ROT-TBL 6 ROT!-ON
```

ANIM!-ON n ---
Turn on animaton
Store animation table address n into current vector.

Example:

```
    DATA ANIM-TBL         (animation table)
    (timer in 1/60ths)    (patterns)
         5 B,             PAT-1 ,
         10 B,            PAT-2 ,
         0 B,             ANIM-TBL ,
    (0 timer signifies a jump to the next word to start table over)

    ANIM-TBL ANIM!-ON
```

ANIMATION TYPES cont.

```
PER!-ON       n ---
    Turn on perspective
    Store perspective table address n into current vector

PERINX!       n ---
    Store perspective table index number n into vector.
    Used as an index into the perspective table.

Example:
    DATA PER-TBL         (perspective table)
         PAT-1 ,
         PAT-2 ,
         PAT-3 ,

    PER-TBL PER!-ON
```

You can use the animation types in combination by specifying more than one and then building proper tables.

```
Example
    DATA ANIM-TBL-1
         5 B,        PAT-1 ,
         10 B,       PAT-2 ,
         0 B,        ANIM-TBL-1 ,

    DATA ANIM-TBL-2
    etc.

    DATA ANIM-TBL-3
    etc.

    DATA ANIM-TBL-4
    etc.

    DATA ROT-ANIM-TBL
NORMAL    [redacted] B, 0H  ANIM-TBL-1 ,
FLIP-FLOP  , B, 80H ANIM-TBL-2 ,
FLOP       , B, C0H ANIM-TBL-3 ,
FLIP       , B, 40H ANIM-TBL-4 ,

    ANIM-TBL-1 ANIM!-ON
    ROT-ANIM-TBL 6 ROT!-ON
```

NOTE:
ANIMATION TABLES MUST
Be the same Size
A change of ROT does NOT
change ANIMINDEX # or
ANIMTIMER

This will rotate and animate. We can continue this with perspective also by including ROT-ANIM-TBL as an entry in a perspective table.

WRITE OPTIONS

XPAND!   n ---
Store xpand color mask n into vector
Xpand mask bits
Bits 0,1 off color
Bits 2,3 on color
Bits 4,5,6,7, not used

XPAND-ON   ---
Turn on magic pattern xpand

XPAND -OFF   ~~n~~ ---
Turn off magic pattern xpand

XPAND!-ON   n ---
Store xpand color mask n into vector
Turn on magic pattern expand

OR-ON   ---
Turn on magic or write

XOR-ON   ---
Turn on magic xor write

PLOP-ON   ---
Turn on magic plop write
Turns off XOR and OR

FLIP-ON   ---
Turn on magic pattern flip

FLIP-OFF   ---
Turn off magic pattern flip

FLOP-ON   ---
Turn on magic pattern flop

FLOP-OFF   ---

Turn off magic pattern flop

FLIPFLOP-ON   ---
Turn on magic pattern flip and flop

FLIPFLOP-OFF   ---
Turn off magic pattern flip and flop

AREAFILL-ON   ---
Fills the area defined by a pattern using only the first byte of the pattern.

AREAFILL-OFF   ---
Turns off the areafill function.

MAGIC!   n ---
Load vector magic whith value n.
(NOTE: used in place of above magic options)

STRING POSTING

STRING ---

Writes string to screen with software if blow up option used)
~~Writes immediately from backgroun~~d
Set X with X!
Set Y with Y!
Set magic with magic commands (automatically sets magic xpand)
Set xpand colors and options with XPAND!
Xpand bits
Bit 7 = blow 4 (blow up string * 4)
Bit 6 = blow 2 (blow up string * 2)
Bit 5 = small font (small font only)
Bit 4 = not used
Bit 2,3 = xpand color on
Bit 0,1 = xpand color off
Set string address with PATTERN!
1st byte at string address is the string count followed by the ASCII string.
To set string address you can use ,"
Example:

```
     DATA STRING-1 ," HI THERE"
```

Then in the task STRING-1 PATTERN!

To save RAM a vector of length SLENGTH can be used to display strings.
Normal font has both upper and lower case.

BIN->ASC l m a s ---

Takes double precision binary number with least significant word l and most significant word m converts it to an ASCII string of length s and stores it at address a. 1st character of string at address a is a sign of the number followed by the ASCII string.
Length byte must be added by the user.
If the double precision # m and l is larger than size s will allow, all 9's are returned.

OSUPR s ---

Suppress leading 0's of string address s with blanks.
Assumes 1st byte at string address s is the string count followed by the ASCII string. If string is all 0's, the last 0 will be left.

USER VERBS

1STWRITE ---
Use when introducing an object to the screen - guarantees screen write with no erase

ACTIVE? n --- p
Returns boolean p
True if vector n is active in the system

BIT m n --- p
Checks Bit m at address n
Returns boolean p true if bit on.

BREAK ---
Runs in Background allows user to return control to the terminal by pressing a terminal key.

ERASE ---
Erase vector pattern from the screen through interrupt. Returns when erase accomplished.

EX n ---
Execute verb at address n (ie.' VERB EX)

FILL n a 1 ---
Fill memory with constant n starting at address a fpr byte
length 1

IMM! n ---
Loads vector n as current vector

INVERT-OFF ---
Turn off Invert feature

INVERT-ON ---
INVERTS entire screen
(ie. for cocktail) coordinates all remain the same

NDUP n ---
Duplicate top n elements of the stack

PUP ---
Power up routine.
does a MAP
intializes the system ques and interupts
sets horizontal color boundry at 28h
sets vertical blank at c8h
sets colors

RANDOM --- p
Returns a 16 bit random # p
2 array RND# is the seed

RES m n --- p
Reset bit m at address n

USER VERBS cont.

REVDLIM ---
Reverse X delta if X limit attained
Reverse Y delta if Y limit attained
Does nothing if no limit attained

REVDX ---
Reverse X delta of vector

REVDY ---
Reverse Y delta of vector

RND n --- p
Returns a random # p within the range n-1 and 0
2 array RND# is the seed

SCRERASE ---
Fill screen memory area with zero's 4000H to 7FFF

SELF --- n
Returns current vector address n

SET m n ---
Set bit m at address n

SHUTUP ---
Turns off sounds

SLEEP ---
Puts current vector to sleep taking him out of the system until woken up by another vector. (See WAKEUP)

SLEEP? v --- f
Returns flag f as true if vector V is asleep

SPARKLES - OFF ---
Turns off card rack sparkle and stars.

SYNC ---
Stops TASK execution
Allows all other tasks to execute before resuming execution

SWAN n ---
Swap nibbles in low byte of n

;TASK: ---
Demarcates the following routine to be a vector task.

TASK-MASTER ---
Activates multi tasking background and starts interrupts (also defined as TT)

USER VERBS cont.

TIMEBMAX!   n ---

Store maximum timebase into current vector m.

Maximum timebase that vector is allowed to vector itself.

If 0 it assumes no maximum.

TIME-BARS   ---

Turns on diagnostic time bars to the right of the horizontal color boundry.

PUP sets horizontal color boundry to the far right side of the screen.

Red - background

Green - vector management

Blue - screen update 1

White - screen update 2

Yellow - idle time

Black - changing processes

Rainbow of small colors - ERROR (see ERROR MESSAGES)

TIMEBSCALE!   n ---

Stores timebase scale factor n into vector.

Examble: If timebase scale = 2 then object updated once every 2 timebases, timer would decrement once for every 2 timebases.

If 0 it defaults to 1

TIMEBSCALE@   --- n

Leaves timebase scale n of current vector.

TT   ---

~~abbreviation for TASK-MASTER~~

does MAP START-INTERRUPTS TASK-MASTER

VDUMP   n ---

Dump contents of vector n

WAKEUP   n ---

Wakes vector n, putting him back into the system to resume execution following where he was put to sleep.

WRITE   ---

Writes vector pattern to the screen through interrupt.

Returns when write accomplished.

XADJ   c --- p

Adjust 0 centered x coordinate c to upper left centered value p. (Note; vector contains upper left center value, VGER interface expects 0 center. X! adjusts automatically.)

YADJ   c --- p

Adjust 0 centered y coordinate c to upper left centered value P.(Note; vector contains upper left center value. VGER interface expects 0 center Y! adjusts automatically)

ZEROTIMEB   ---

Gives start over timebase for current vector.

SYSTEM HELPER VERBS

<STKH ---
Does <STK HEX

<STKD ---
Does <STK DECIMAL

STK> ---
Does STK> DECIMAL

XDI ---
Does DI and resets interrupt mode to 0 for disks.

DED ---
Does XDI DECIMAL EDIT

.HOPS ---
Gives message ".HOPS? Y or N" and waits for KEYBOARD entry.
If y does .HOU .OPS
Else does nothing

.COPS ---
Does .CEN .OPS

.NLOPS ---
Does CR CR CR CR PAGE PAGE
.NOPS .NLIST

DV= --- vvvv
Creates a double precision VARIABLE with name vvvv

NC= --- vvvv
Does 1+DUP C= vvvv
Use for table creation

SC= --- vvvv
Does DUP C= vvvv
Use for table creation

USER SUBROUTINES

Subroutines that end in RET.

```
write    write pattern with pattern board
         does not flush if expand set

         in-  IX= pattern address (no header on pattern)
              B=  xpand color
              C=  magic with shift
              D=  Y size
              E=  X size
              HL= absolute screen address

         out - nothing
                        (A,B,C,D,E,H,L,IX altered)

relabs   relative X Y to magic absolute address conversion
                 does not invert
         in-  DE=X
              HL= Y

         out- A= shift
         HL= absolute magic screen address
                     (A,D,E,H,L altered)


bwrite   write blow up pattern to screen
          does immediate software write from background

         in-  B= Blow up + Expand
                     Bit 7 = blow up *4
                     Bit 6 = blow up *2
                     Bits 4,5 = not used
                     Bits 2,3 = expand color on
                     Bits 0,1 = expand color off
                 C= Magic + shift
                 D= Y size
                 E= X size
                 HL= Screen addr
                 IX= Pattern addr

         out- BC= same
              E=  blow up factor
                 (A,D,E,H,L,IX,IY altered)
```

USER SUBROUTINES cont.

COMPHL
2's compliment register set HL
saves psw
(H,L altered)

COMPDE
2's compliment register set DE
saves psw
(D,E altered)

COMPBC
2's compliment register set BC
saves psw
(B,C altered)

INDEXW
Index into a word table
(in- HL= table address A=index value)
(out- DE= indexed value HL= address of indexed value
A= index value)
(D,E,H,L altered)

divd16/8
Divides 16 bit dividend by an 8 bit divisor.
Returns 16 bit quotient.
(IN-HL = signed dividend, C= unsigned divisor)
(OUT-HL = signed quotient)
(A,B,H,L altered)

divd16/16
Divides 16 bit dividend by a 16 bit divisor.
Returns 16 bit unsigned.
(IN- A:C = dividend, DE = divisor)
(OUT A:C quotient)
(A,B,C,H,L altered)

mult8*8
Unsigned 8 bit multiply
(IN H = operand 1, E = operand 2)
(OUT HL = product)
(B,D,H,L altered)

vreloffrelabs

ROOT VGER

QUEUES

QUEUE n --- vvvv
Defines queue vvvv with n entries.

EMPTY-QUEUE vvvv ---
Empties queue vvvv. Must also be used after a queue definition and before its use in order to initialize pointers.

QUEUE-IN d vvvv --- f
Puts data d into queue vvvv and returns flag f as true if the queue did not have enough room.

QUEUE-OUT vvvv --- d f l
Gets data d from queue vvvv if it exists. If the queue is empty only flag f (set to true) is returned otherwise data d and flag f (set to false) are returned.

QDUMP ---
Dump contents of all VGER queues

VECTOR RETURN STACK

Each vector can have its own return stack (ie.8 0 DO LOOP) First create a RAM area for the stack. The stack RAM can be attached to the end of a vector. (Note: CAUTION - there is no stack checking so if you overflow you can get into trouble.)

RSTACK!-ON   s v ---
Loads vector return stack address s into vector v
(Note: do before ;TASK:)

RSTACK-OFF   v ---
Turn off vector return stack option for vector v (Note: do before ;TASK:)

ERROR MESSAGES

ERROR? --- n

Returns error number n
Use when diagnostic colors go to rainbow

Error Messages

1. Playaction management queue overflow
2. Vector management queue overflow
3. Screen update queue 1 overflow
4. Screen update queue 2 overflow

User can add error messages.

error# is byte variable containing error message numbers.
error-addr is a variable containing the address of error handler routine initialized at diagnostic rainbow loop.
When error detected load error number in error# and jump to the contents of error-addr.

INTERRUPTS

Look at PUP in Edible VGER to see how interrupts are set up.

There are 3 interrupts

line 50	BAKI (background)
line 100	SUI1 (screen update 1)
line 200	SUI2 (screen update 2)

To set up your own interrupt routine (ie. guarantee coin and IO check) do so by loading your interrupt into one of the V variables (see PUP). Your routine must then jump to the appropriate VGER Interrupt routine.

Example:

```
    SUBR MY-INTR
         -
         -
         SUI2 JMP,

    MY-INTR SUI1V!
```

This sets up user interrupt routine at line 200 without destroying VGER.

To change interrupt lines change L variables (see PUP)

Example:

```
    20 SUI2L B!
```

Changes background interrupt routine to line 20 from
VGER default of 50.

NOTE: Interrupt varables always contain line # and vector address
for the interrupt following there own.

VECTOR CONSTANTS

```
Task Header Block
(W) TPAPC            (playaction program counter)
(W) TOPAPC           (old playaction program counter)
(B) TSTAT            (task status)
       Bits:     0 = TBNEWTASK (New task)
                 1 = TBACT (Vector active)
                 3 = TBSLEEP (Vector asleep)
                 4 = TBSTACK (user supplied return stack)
(B) TPRI             (task priority)
(W) TVMR             (vector management routine)
(W) TSUR             (screen update routine)
(B) TSUCNT           (screen update count)
(B) TTIMEB           (time base)
(B) TSCALE           (time base scaler)
(W) TTIMER           (timer)
(B) TVMROPT          (vmr options)
       Bits:     0 = TBINTCPT-CHK (Intercept check)
                 1 = TBFLAG-CHK (Flag check)
                 2 = TBLIMIT-CHK (Limit check)
                 3 = TBDEST-CHK (Destination check)
                 4 = TBANGVECT (Angle vector)
                 5 = TBNOVECT (No vector)
                 6 = TBHOOK (Hook check)
                 7 = TBTIMER-CHK (Timer check)
(B) TVMROPT2         (vmr options # 2)
       Bits:     0 = TBLIMBOUT-CHK (Limit with back out)
                 1 = TBDESTX-CHK (Destination x check)
                 2 = TBDESTY-CHK (Destination y check)
                 3 = TBLIMX-CHK
                 4 = TBLIMY-CHK
(B) TCHGSTAT         (state trans feedback)
       Bits:     0 = TBLIMIT (Limit attained)
                 1 = TBINTCPT (Intercept detected)
                 2 = TBDEST (Destination reached)
                 3 = TBTIMEDOUT (Timer went to 0)
                 4 = TBFLAG (Flag detected)
(B) TCHGSTAT2        (state trans feedback # 2)
       Bits:     0 = TBLIMHY (Limit Y high attained)
                 1 = TBLIMLY (Limit Y low attained)
                 2 = TBLIMHX (Limit X high attained)
                 3 = TBLIMLX (Limit X low attained)
                 4 = TBDESTX (Destination X reached)
                 5 = TBDESTY (Destination Y reached)
(W) TFLAGADR         (flag address)
(W) THOOKADR         (vmr hook address)
(W) TRSTACK               (return stack address)
    TLENGTH          (length of task header)
```

67

```
Motion Information Block
(B) VMAGIC          (magic register)
       Bits:     0 = Shift amount
                 1 = Shift amount
                 2 = MRAREAFILL (Area fill)
                 3 = MREXP (Expand)
                 4 = MROR (OR)
                 5 = MRXOR (XOR)
                 6 = MRFLOP (FLOP)
                 7 = MRFLIP (FLIP)
(B) VXPAND          (xpand color use bottom nibble)
(W) VX              (x pixel value)
(W) VY              (y pixel value)
(W) VPAT            (pattern for write)
    SLENGTH         (length of string header)
(B) VOMAGIC         (erase magic register)
(B) VOXPAND         (erase xpand color use bottom nibble)
(B) VLOGICSTAT      (vector logic status byte)
       Bits:     0 = VBSUPDATE (Do screen update)
                 1 = VBNOWRITE (No write)
                 2 = VBNOEARSE (No erase)
                 3 = VBNOSU (No screen update)
(B) VLOGICSTAT2     (vector logic status byte #2)
       Bits:     0 = VBANIM-CHK (Animation check)
                 1 = VBANIM (Animation)
                 2 = VBROT (Rotation)
                 3 = VBPERS (Perspective)
(B) VTBMAX          (max time base)
(W) VDX             (x speed)
(W) VAX             (x acceleration)
(W) VLIMLX          (x limit low)
(W) VLIMHX          (x limit high)
(W) VDY             (y speed)
(W) VAY             (y acceleration)
(W) VLIMLY          (y limit low)
(W) VLIMHY          (y limit high)
(W) VOPAT           (old pattern for erase)
(W) VSCRADR         (screen address for write)
(W) VOSCRADR        (screen address for erase)
(W) VDESTX          (destination x)
(W) VDESTY          (destination y)
(W) VPRATBL         (per rot anm table)
(B) VANIMTIMER      (animation timer)
(B) VANIMINX        (animation index #)
(B) VROTINXSHF      (rotation inc shift amount)
(B) VPERINX         (perspective index #)
(W) VPLRVEL         (polar velocity)
(W) VPLRACC         (polar acceleration)
(B) VANGLE          (polar angle)
(B) VPLRTIMER       (polar angle timer)
    VLENGTH         (length of motion vector)
```

PORTS: PORT EQUATES

INFBK	(Interupt feed back port)
INMOD	(Interupt mode port)
INLIN	(Interupt line port)
MAGIC	(Magic port)
XPAND	(Expand color mask port)
INCPT	(Intercept port)
VERBL	(Vertical blanking line port)
HORCB	(Horizontal color boundry port)

TASK EXAMPLE

```
<STK
    RAMMARK
    VLENGTH R= DEMOVECT
    RAMLEN C= DEMO-RAM-LENGTH
    VARHERE C= DEMO-RAM-START
: ZERO-DEMO-RAM 0 DEMO-RAM-START DEMO-RAM-LENGTH FILL ;

    PATTERN DEMO-PAT  0 B, 0 B, 2 B, 2 B, QUADPAT
      ~ 3322 1133 ^
      ~ 3322 1133 ^

HEX ( set base to hex )

: DEMOPA ;TASK:
    DEMO-PAT PATTERN! ( set pattern)
    ZEROXY          ( set coordinates at 0 center of screen)
    XOR-ON          ( set magic for XOR)
    1STWRITE        ( start writing with no erase)
    100 DX!         ( set X delta to 1 pixel /60th of a sec.)
    50 RND          ( get Random # between 0 and 49)
    TIMER!-ON       ( store random # in timer and turn it on)
    WAIT            ( start up timer and wait until it goes to 0)
    BEGIN           ( start of loop)
    50 TIMER!-ON    ( store 50 into timer and turn it on)
    GO              ( start up timer, vectoring and writing)
                    ( return when timer goes to 0)
    REVDX           ( reverse delta x)
    0 END ;         ( go back to begin)


: TEST PUP          ( power up)
    BREAK           ( key board break)
    SCRERASE        ( erase screen)
    ZERO-DEMO-RAM   ( zero demo ram area )
    TIME-BARS       ( turns on diagnostic time bars)
    DEMOVECT DEMOPA ( load vector in background que with task DEMOPA)
    TT ;            ( start up system)


STK> ;S
```

(The above task will 1st wait a random period of time 0 -> 50/60 of a second then move, after each 50/60 of a second it will reverse direction.)