
NEW DICTIONARY
RSC Forth

```
SCR # 1
  0 ( Utility Words )
  1 : U. ( print unsigned number from stack )
  2   S->D DROP 0 D. ;
  3 CODE LSHIFT ( x,n---x' ; Shift word x left by n bits )
  4   TOP LDY,  BEGIN,  DEY, 0< NOT WHILE,  CLC, SEC ROL,
  5   SEC 1+ ROL,   REPEAT,  POP JMP,
  6 END-CODE
  7 : 2* 1 LSHIFT ;
  8 CODE RSHIFT ( x,n---x' ; Shift word x right by n bits )
  9   TOP LDY,   BEGIN,   DEY, 0< NOT WHILE,
 10   SEC 1+ LSR, SEC ROR,   REPEAT,   POP JMP,
 11 END-CODE
 12
 13 CODE BSWAP ( x---x' ; Swap bytes in x )
 14   TOP LDY, TOP 1+ LDA, TOP STA, TOP 1+ STY, NEXT JMP,
 15 END-CODE
T
 SCR # 2
  0 ( Masc --- data structures 1 ) HEX
  1 FORTH DEFINITIONS
  2 : %TO 36A ;    : TO 1 %TO ! ;
  3
  4 : FROM/TO
  5   %TO @ IF !  0 %TO !  ELSE @ ENDIF ;
  6
  7 : PARAMETER
  8   <BUILDS ,   DOES> FROM/TO  ;
  9
 10 : PARAMETERS
 11   <BUILDS  DUP , 0 DO 0 , LOOP
 12   DOES>  SWAP 2* + 2+ FROM/TO   ;
 13 : XYPARAMETERS
 14   <BUILDS 2DUP SWAP , , * 0 DO 0 , LOOP
 15   DOES> DUP >R @ * + 2+ 2* R> + FROM/TO ;
U
 SCR # 3
  0 ( Masc --- data structures 2 )
  1 : USERPARAM
  2   <BUILDS DUP , !   ( n, user addr --- name )
  3   DOES>  @ FROM/TO  ;
  4
  5 : TASK ;
  6
  7 : EXPARAMETERS ( size --- name )
  8   <BUILDS DUP , 0 DO ' TASK , LOOP
  9   DOES> SWAP 2* + 2+ %TO @
 10   IF ! 0 %TO !
 11   ELSE @ CFA EXECUTE THEN ;
 12
 13 : INPUT QUERY INTERPRET ;
 14
 15
```

```
SCR # 4
 0 ( Masc --- Randoms and delays )
 1 DECIMAL
 2 : SEED 870 ;
 3 : RANDOM ( --- SEED ) SEED @ 31421 * 6927 + DUP SEED ! ;
 4 : CHOOSE ( n -- 0 =< r < n ) RANDOM U* SWAP DROP ;
 5 : RND    ( n m -- n =< r =< m ) SWAP OVER - 1+ CHOOSE + ;
 6
 7
 8
 9
10
11
12
13
14
15

SCR # 5
 0 ( Time Queue Words )
 1 HEX
 2 35E CONSTANT T> ( n --- ! location of time queue pointer )
 3 8000 360 USERPARAM T_HEAD ( Start of Time Queue )
 4 : INC_T>       T> @ + T> ! ;  ( n--- ! Increment T> by n )
 5 : T_INIT     T_HEAD T> ! ; ( Initialize pointer to Start )
 6 : T,        ( n--- ! Load n into Queue increment pointer )
 7          T> @ ! 2 INC_T> ;
 8 : @T      T> @ @ ; ( ---n ! value n found at pointer loc )
 9 : !!    -1 T> @ ! ; ( --- ! mark end of Queue with -1 )
10
11 : FIND_!! ( --- ! leave pointer at end of queue )
12           T_INIT BEGIN
13           T> @ T_HEAD 1FFF + > DUP   @T -1 =  OR NOT
14           WHILE  DROP 6 INC_T> REPEAT
15           IF CR ." !! NOT FOUND" CR T_INIT THEN ;
V
 SCR # 6
  0 ( Data Queue Words )
  1 : DEPTH S0 @ SP@ - 2- 2 / ; ( --- stack depth )
  2 HEX
  3 362 CONSTANT D>     ( holds data queue pointer )
  4 A000 364 USERPARAM D_HEAD  ( start of data Q  )
  5
  6 : D_INIT      D_HEAD D> ! ;  ( initialize pointer )
  7 : D,          ( n --- ! load n into Q, increment pointer )
  8               D> @ !    D> @ 2+ D> !  ;
  9 : @D          D> @ @ ;  ( --- n ! value at pointer loc )
 10
 11 : !DATA       ( xyz --- ! move data stack to data Queue )
 12               ( with word count in front of data )
 13               DEPTH DUP 1+ 0 DO D, LOOP ;
```

```
SCR # 7
 0 ( Data Queue word @data )  HEX
 1 ( : @DATA    DUP HERE ! @ 2 * 0 SWAP )
 2 (            DO I  HERE @ + @ ?STACK -2 +LOOP )
 3
 4 ( n --- xyz ! move data Q data to stack       )
 5 ( n is data Q location holding word count     )
 6 CODE @DATA
 7  TOP LDA, N STA, TOP 1+ LDA, N 1+ STA, ( load n into N )
 8  INX, INX,    ( pop n off data stack )
 9  N )Y LDA,    ( load word count into accumulator )
10  SEC, .A ROL, TAY, INY, ( #bytes = #words*2 + 2 )
11  BEGIN, DEX, 6D # CPX, 0< IF, 1 # LDY, THEN, ( stack overflow )
12  N )Y LDA, TOP 1+ STA,  ( move Q byte to stack )
13  DEY, 0= UNTIL,     ( decrement byte count )
14  INX, INX, NEXT JMP,
15 END-CODE

SCR # 8
 0 ( Timer Words )
 1 HEX      0 368 USERPARAM TIME
 2          80 KHZ USERPARAM TEMPO
 3 : TM TO TIME ;  ( n--- ! TIME = n )
 4 : DUR TIME + TO TIME ;  ( n--- ! TIME = TIME + n )
 5
 6 : TEMPO! ( --- start timer B counting from TEMPO val )
 7          28 MCR C!  KHZ C@ 1C C!  KHZ 1+ C@ 1E C!  ;
 8 : WAIT   ( --- wait for timer B to finish counting )
 9        BEGIN 11 C@ 20 AND ?TERMINAL OR UNTIL ;
10 : MS     TO TEMPO TEMPO! WAIT ; ( n--- ! wait n msecs )
11
12
13
14 : >DLY< ( --- ) 50 0 DO LOOP ;
15 : DELAY ( N --- ) 0 DO >DLY< LOOP ;
U
 SCR # 9
  0 ( Time Queue CYCLE )
  1 ( Cycle throush the time Queue from start to end [-1] )
  2 ( Check time Q triplets [ time,data pointer,execute pointer] )
  3 ( if time=TIME, load data into stack and execute )
  4
  5 : CYCLE
  6   TEMPO! T_INIT BEGIN  ( start timer, start at top of Queue )
  7   @T DUP -1 = NOT WHILE  ( check for !!, end of time queue )
  8   TIME = IF              ( if time queue val = time then ... )
  9   2 INC_T> @T DUP 0= NOT ( check data pointer )
 10   IF @DATA ELSE DROP THEN ( if not 0, load data Q into stack )
 11   2 INC_T> @T EXECUTE    ( execute program pointer value )
 12   2 INC_T>
 13   ELSE 6 INC_T> THEN     ( if not time go to next TQ triplet )
 14   REPEAT DROP TIME 1+ TO TIME ;
```

```
 SCR # 10
  0 ( Time Queue Play )
  1 HEX
  2 0 36F USERPARAM PLAY_TILL
  3 0 370 USERPARAM PLAY_START
  4
  5 : PLAY
  6   PLAY_START TO TIME    BEGIN
  7   CYCLE WAIT   PLAY_TILL TIME < ?TERMINAL OR
  8   UNTIL   CR ." STOPPED AT TIME " TIME . CR ;
  9
 10
 11
 12
 13
 14
 15
T
 SCR # 11
  0 ( Time Queue / Data Queue Compiling )
  1 HEX
  2 : TLD   ( pfa--- ; load time and data queues )
  3   TIME T,                ( Load time value to T queue )
  4   DEPTH 1 >              ( anything on the data stack? )
  5   IF   D> @ T,           ( if so load data Q pointer )
  6        CFA T,  !DATA ;;  ( also load execute pointer and stack )
  7   ELSE   0 T,  CFA T, ;; ( if not load 0 as data Q pointer )
  8   THEN ;
  9
 10 : ^   ( used as ^ cccc --- pfa of cccc  )
 11   [COMPILE] '  STATE @
 12   IF COMPILE TLD    ( if compiling compile TLD into word )
 13   ELSE TLD          ( if executing just execute TLD )
 14   THEN
 15 ; IMMEDIATE
T
 SCR # 12
  0 ( Dac, Adc, Pulse Interface Devices Words )
  1 HEX
  2 ( n --- x ; x read from analog-to-dig conv #n )
  3 : ADC 7 AND 160 + C@ ;
  4
  5
  6 ( x, n --- ; x loaded into dig-to-analog conv #n )
  7 : DAC 7 AND 160 + C! ;
  8
  9 ( read -- PULSE  /  write -- n TO PULSE )
 10   0 150 USERPARAM PULSE
 11
 12
 13
 14
 15
```

```
 14
 15
T
 SCR # 13
  0 ( Midi words )    HEX
  1 40 372 USERPARAM VEL      0 374 USERPARAM CHNL
  2 ( -- Initialize serial port for midi i/o )
  3 : MIDINIT 15 110 3 OVER C! C! FE FA C! ;
  4 ( n-- ! Midi Load after transmitter is clear )
  5 : MLD BEGIN 110 C@ 2 AND UNTIL  111 C! ;
  6 : M CHNL F AND OR MLD 7F AND MLD 7F AND MLD ;
  7 : ON     VEL SWAP 90 M ; ( key -- ! Note On )
  8 : OFF      0 SWAP 80 M ; ( key -- ! Note Off )
  9 : KPRES           A0 M ; ( vel,key -- ! Poly Key Press )
 10 : CONT     B0 M ; ( value, cont# -- ! Control Change    )
 11 : PWHL     E0 M ; ( msb, lsb -- ! Pitch Wheel Change    )
 12 : MM CHNL F AND OR MLD 7F AND MLD ;
 13 : PROG C0 MM ;    ( program -- ! Program Change )
 14 : CPRES D0 MM ;   ( value   -- ! Channel Pressure )
 15   DECIMAL    ;S
V
 SCR # 14
  0 ( Midi Words - 2 )  DECIMAL
  1 : KLR    ( --- ! Turns off 128 midi keys )
  2          128 0 DO I OFF LOOP ;
  3 : KK     ( dur,key --- ! Monophonic midi note )
  4          DUP ON SWAP 0 DO TEMPO! WAIT LOOP OFF ;
  5
  6 ( System Exclusive for DX7, channel 0 )
  7 : DX7    240 MLD 67 MLD ;
  8 : FPAR   ( data, param# --- ! Function change )
  9          DX7 16 MLD 8 MLD MLD MLD ;
 10 : CPAR   ( data, param# --- ! Common System change )
 11          DX7 16 MLD  DUP 127 > IF 128 - 1
 12          ELSE 0 THEN MLD MLD MLD ;
 13
 14
 15
T
 SCR # 15
  0 ( Midi Words - 3 )
  1 ( midi control changes, n --- ! )
  2 DECIMAL
  3 : MODWHL  1 CONT ;    : BREATH   2 CONT ;
  4 : FOOT    4 CONT ;    : P_TIME   5 CONT ;
  5 : D_ENTRY 6 CONT ;    : VOLUME   7 CONT ;
  6 : TOUCH 208 CONT ;
  7
  8 ( midi control switches, --- )
  9 : SUS_ON 127 64 CONT ;   : SUS_OFF 0 64 CONT ;
 10 : P_ON   127 65 CONT ;   :   P_OFF 0 65 CONT ;
 11 : +D     127 60 CONT ;   : -D    127 61 CONT ;
 12
 13
 14
 15
T
```

```
SCR # 16
 0 ( MIDI INPUT Interrupt, data pushed on 4K FIFO, )
 1 HEX
 2 358 CONSTANT POP>  35A CONSTANT PUSH>  35C CONSTANT STAT>
 3 CODE >M
 4   PHA, TYA, PHA, N LDA, PHA, N 1+ LDA, PHA,      ( Save reg's )
 5   FE # LDA, 111 CMP, 0= NOT IF,  ( Filter out FE midi signal )
 6   PUSH> LDA, N STA, PUSH> 1+ LDA, N 1+ STA,    ( ptr to pg 0 )
 7    111 LDA, N )Y STA,    FF # LDA,          ( push onto fifo )
 8    ( stat=ff, increment pointer lo & hi byte with wraparound )
 9   STAT> STA,  PUSH> INC, 0= IF,     PUSH> 1+ LDA,     CLC,
10   1 # ADC, F # AND, 10 # ORA,      PUSH> 1+ STA,     THEN,
11                          ( replace save registers before exiting )
12   THEN, PLA, N 1+ STA, PLA, N STA, PLA, TAY, PLA, RTI,
13 END-CODE
14     DECIMAL    ;S
15
T
SCR # 17
 0 ( Midi In Fifo words - pop, push, fwrap, fclr )
 1 HEX
 2 ( fifo pointers and status - 0 = empty, f = non-empty )
 3 ( does not contain 'fifo full' tests )
 4
 5 : FWRAP    ( Wrap around routine for Fifo memory 1000 to 1FFF )
 6              DUP 2000 < IF ELSE DROP 1000 THEN ;
 7 : POP    ( -- n ! Pop n from fifo )
 8          POP> @  C@      POP> @ DUP PUSH> @ =
 9          IF DROP ELSE 1+ FWRAP DUP POP> !  PUSH> @ =
10                          IF  0 STAT> C!  THEN THEN  ;
11 : PUSH  ( n -- ! Push n onto the fifo )
12          PUSH> @ 1+ FWRAP DUP POP> @ = IF DROP DROP
13          ELSE SWAP PUSH> @  C!  PUSH> ! F STAT> C! THEN ;
14 : FCLR 1000 DUP POP> ! PUSH> ! 0 STAT> C! ;
15 DECIMAL ;S
T
SCR # 18
 0 ( Enable and Disable for Midi In Interrupt )
 1 HEX
 2
 3 : ENB_MIDIN MIDINIT ' >M @ NMIVEC ! FCLR 95 110 C! ;
 4 : DIS_MIDIN 15 110 C!  ;
 5
 6 : .MIDIN  ( Print out midi input )
 7   ENB_MIDIN CR BEGIN STAT> C@ IF POP . THEN ?TERMINAL
 8   UNTIL DIS_MIDIN ;
 9
10 : ^N ( dur, key --- )
11   SWAP >R DUP >R ^ ON R> R> DUR ^ OFF ;
12
13
```

```
 12
 13
 14
 15
V
 SCR # 20
  0 ( Hybrid Synthesizer words )
  1 HEX
  2 148 CONSTANT HMOD! 144 CONSTANT HMOD@
  3 14E CONSTANT HWAVA! 14F CONSTANT HWAVD!
  4 376 CONSTANT HDAT
  5 : HDATA! HDAT @ ;
  6
  7 : POLL  ( 1/0 --- ! resume or stop polling after data load )
  8   IF HDATA! 1 AND   14C OR   HDAT !
  9   ELSE  HDATA! 1 AND  14A OR  HDAT !
 10   THEN  ;
 11 : MODMASK  ( 1/0 --- ! Turn modmask on or off after data ld )
 12   IF  HDATA! FFFE AND         HDAT !
 13   ELSE  HDATA! 1 OR           HDAT !
 14   THEN  ;
 15 DECIMAL
T
 SCR # 21
  0 ( Hybrid Synthesizer Words - 2 )
  1 HEX
  2 : HRAMP   <BUILDS ,     ( Module# HRAMP name )
  3             DOES>       ( Level, speed ---   )
  4   @ HMOD! C!      F AND SWAP 4 LSHIFT OR   HDATA! C! ;
  5
  6 : HPITCH  <BUILDS ,     ( Module# HPITCH name )
  7           DOES>         ( Pitch byte ---      )
  8   @ HMOD! C!    HDATA! C!  ;
  9
 10 : HWAVE   <BUILDS ,     ( Module# HWAVE name )
 11           DOES>         ( Memory A/B, Wave# --- )
 12   @ HMOD! C!   3F AND SWAP 6 LSHIFT OR    HDATA! C! ;
 13
 14
 15 DECIMAL
T
 SCR # 22
  0 ( Hybrid Synthesizer words - 3 )
  1 HEX ( Mix, Lo pass filt, Transpostion, X-loc, Y-loc, Vol )
  2  8 HRAMP M0      B HRAMP X0
  3 18 HRAMP M1     1B HRAMP X1
  4 28 HRAMP M2     2B HRAMP X2
  5 38 HRAMP M3     3B HRAMP X3
  6
  7  9 HRAMP L0      C HRAMP Y0
  8 19 HRAMP L1     1C HRAMP Y1
  9 29 HRAMP L2     2C HRAMP Y2
 10 39 HRAMP L3     3C HRAMP Y3
 11
 12  A HRAMP T0      D HRAMP V0
 13 1A HRAMP T1     1D HRAMP V1
 14 2A HRAMP T2     2D HRAMP V2
 15 3A HRAMP T3     3D HRAMP V3       DECIMAL
T
```

```
 15 3A HRAMP T3      3D HRAMP V3        DECIMAL
T
 SCR # 23
  0 ( Hybrid Synthesizer Words - 4 )
  1 HEX
  2  7 HPITCH P0       5 HWAVE W0
  3 17 HPITCH P1      15 HWAVE W1
  4 27 HPITCH P2      25 HWAVE W2
  5 37 HPITCH P3      35 HWAVE W3
  6
  7 : HTIMER  <BUILDS , DOES> ( 16-bit timer val --- )
  8   @ DUP 1+ HMOD! C!
  9   SWAP DUP HDATA! C!
 10   SWAP HMOD! C!
 11   BSWAP HDATA! C!
 12 ;
 13  2 HTIMER TM0       22 HTIMER TM2
 14 12 HTIMER TM1       32 HTIMER TM3
 15 DECIMAL
T
 SCR # 24
  0 ( Hybrid Synthesizer words - 5 )
  1 HEX
  2 : FMATRIX <BUILDS
  3   80 C, 4 C, 2 C, 1 C,   4 C, 80 C, 2 C, 1 C,
  4   4 C, 2 C, 80 C, 1 C,   4 C, 2 C, 1 C, 80 C,
  5   DOES> SWAP 2 LSHIFT + + C@ ;  ( x,y --- byte )
  6 FMATRIX FX
  7
  8 378 CONSTANT HFM0      379 CONSTANT HFM1
  9 37A CONSTANT HFM2      37B CONSTANT HFM3
 10
 11 : FM ( x,y --- ; voice x frequency modulates voice y )
 12   SWAP OVER FX
 13   3 LSHIFT SWAP HFM0 + DUP C@
 14   ROT OR SWAP C! ;
 15 DECIMAL
T
 SCR # 25
  0 ( Hybrid Synthesizer words - 6 )
  1 HEX
  2 : MXM ( x,y --- ; voice x mixer modulates voice y )
  3   SWAP OVER FX
  4   SWAP HFM0 + DUP C@
  5   ROT OR SWAP C! ;
  6 : MODULATE ( voice# --- ; start mod, set up with FM & MXM )
  7   DUP 2 LSHIFT 4 OR HMOD! C!
  8   HFM0 + C@          HDATA! C! ;
  9 : MODULATE_OFF ( --- )
 10   4 0 DO 0 HFM0 I + C!   I MODULATE    LOOP ;
 11 : H_INIT      0 POLL 0 MODMASK
 12   MODULATE_OFF  3 3E W0 3 3E W1 3 3E W2 3 3E W3
 13   3E 8 DO    I HMOD! C!   0 148 C!  LOOP
 14        9A P0 9A P1 9A P2 9A P3  ;
 15 DECIMAL
```