amforth 3.0 Reference Card

Compare

Control Structure

Arithmetics

		P			- 10 1- 11-1 1 1-1
				again	(addr)
				begin	(addr)
1-	(n1 n2)			do	(addr)
1+	(n1 n2)			i	(n)
2/	(n1 n2)	15	(14 10 61)	1	
2*	(n1 n2)	d>	(d1 d2 flag)		; R(loop-sys loop-sys)
abs	(n1 u1)	d<	(d1 d2 flasg)	if	(addr)
><	(n1 n2)	=	(n1 n2 flag)	j	(n)
		0=	(n flag)		; R(loop-sys1 loop-sys2 loop-sys1 loop-sy
cell+	(n1 n2)	>	(n1 n2 flag)	leave	()
cells	(n1 n2)	0>	(n1 flag)		R(loop-sys)
d2/	(d1 d2)	<		laan	(addr)
d2*	(d1 d2)		(n1 n2 flasg)	loop	
dinvert	(d1 d2)	0<	(n1 flag)	+loop	(addr)
dinvert	(d1 d2)	max	(n1 n2 n1 n2)	?do	(addr)
d-	(d1 d2 d3)	min	(n1 n2 n1 n2)	repeat	(addr1 addr2)
		<>	(n1 n2 flag)	unloop	()
${\tt dnegate}$	(d1 d2)	0<>	(n flag)	-	; R(loop-sys)
d+	(d1 d2 d3)	u>	(u1 u2 flag)	while	(dest orig dest)
invert	(n1 n2)			WILLIE	(dest olig dest)
log2	(n1 n2)	u<	(u1 u2 flasg)		
lshift	(n1 n2 n3)			C	
_	(n1 n2 n3)			Conver	rsion
				15 -	(14
mod	(n1 n2 n3)			d>s	(d1 n1)
m*	(n1 n2 d)			s>d	(n1 d1)
*	(n1 n2 n3)				
+	(n1 n2 n3)				
+!	(n addr)			Diction	nary
rshift	(n1 n2 n3)				v
					(n)
/	(n1 n2 n3)			compile	()
/mod	(n1 n2 rem quot)			-	
*/	(n1 n2 n3 n4)			create	()
*/mod	(n1 n2 n3 rem quot)	Compil	er	,	(XT)
ud/mod	(ud1 n rem ud2)	Compi			
um/mod	(ud u2 rem quot)			Except	ions
um/mod um*	(ud u2 rem quot) (u1 u2 d)			Except	ions
um/mod um* u/mod	(ud u2 rem quot) (u1 u2 d) (u1 u2 rem quot)			Except	
um/mod um*	(ud u2 rem quot) (u1 u2 d)			_	(n*x)
um/mod um* u/mod	(ud u2 rem quot) (u1 u2 d) (u1 u2 rem quot)	\	()	abort	(n*x) R(n*y)
um/mod um* u/mod	(ud u2 rem quot) (u1 u2 d) (u1 u2 rem quot)	\ [']	() (XT)	_	(n*x) R(n*y) (n*x)
um/mod um* u/mod	(ud u2 rem quot) (u1 u2 d) (u1 u2 rem quot)	[']	· ·	abort	(n*x) R(n*y) (n*x) R(n*y)
um/mod um* u/mod	(ud u2 rem quot) (u1 u2 d) (u1 u2 rem quot)	['] code	(XT) ()	abort	(n*x) R(n*y) (n*x) R(n*y) (xt)
um/mod um* u/mod	(ud u2 rem quot) (u1 u2 d) (u1 u2 rem quot)	['] code :	(XT) () ()	abort	(n*x) R(n*y) (n*x) R(n*y)
um/mod um* u/mod	(ud u2 rem quot) (u1 u2 d) (u1 u2 rem quot)	['] code : :noname	(XT) () () (xt)	abort abort"	(n*x) R(n*y) (n*x) R(n*y) (xt)
um/mod um* u/mod O	(ud u2 rem quot) (u1 u2 d) (u1 u2 rem quot) (0)	['] code : :noname constant	(XT) () () (xt) (n)	abort abort" catch handler	(n*x) R(n*y) (n*x) R(n*y) (xt) (addr)
um/mod um* u/mod	(ud u2 rem quot) (u1 u2 d) (u1 u2 rem quot) (0)	['] code : :noname constant does>	(XT) () () (xt)	abort abort" catch handler	(n*x) R(n*y) (n*x) R(n*y) (xt) (addr)
um/mod um* u/mod O	(ud u2 rem quot) (u1 u2 d) (u1 u2 rem quot) (0)	['] code : :noname constant	(XT) () () (xt) (n)	abort" catch handler throw	(n*x) R(n*y) (n*x) R(n*y) (xt) (addr) (n)
um/mod um* u/mod O	(ud u2 rem quot) (u1 u2 d) (u1 u2 rem quot) (0)	['] code : :noname constant does>	(XT) () () (xt) (n) ()	abort" catch handler throw	(n*x) R(n*y) (n*x) R(n*y) (xt) (addr)
um/mod um* u/mod O	(ud u2 rem quot) (u1 u2 d) (u1 u2 rem quot) (0)	['] code : :noname constant does> ."	(XT) () () (xt) (n) ()	abort" catch handler throw	(n*x) R(n*y) (n*x) R(n*y) (xt) (addr) (n)
um/mod um* u/mod 0	(ud u2 rem quot) (u1 u2 d) (u1 u2 rem quot) (0)	['] code : :noname constant does> ." Edefer else	(XT) () () (xt) (n) () () (n <name>) (addr1 addr2)</name>	abort abort" catch handler throw Extend	(n*x) R(n*y) (n*x) R(n*y) (xt) (addr) (n)
um/mod um* u/mod 0	(ud u2 rem quot) (u1 u2 d) (u1 u2 rem quot) (0)	['] code : :noname constant does> ." Edefer else end-code	(XT) () () (xt) (n) () () (n <name>) (addr1 addr2) ()</name>	abort catch handler throw Extend a0 a0-	(n*x) R(n*y) (n*x) R(n*y) (xt) (addr) (n)
um/mod um* u/mod 0	(ud u2 rem quot) (u1 u2 d) (u1 u2 rem quot) (0)	['] code : :noname constant does> ." Edefer else	(XT) () () (xt) (n) () () (n <name>) (addr1 addr2) () ()</name>	abort catch handler throw Extend a0 a0- a0+	(n*x) R(n*y) (n*x) R(n*y) (xt) (addr) (n) led VM (n) (n) (n)
um/mod um* u/mod 0	(ud u2 rem quot) (u1 u2 d) (u1 u2 rem quot) (0)	['] code : :noname constant does> ." Edefer else end-code exit	(XT) () () (xt) (n) () () (n <name>) (addr1 addr2) () () R(xt)</name>	abort catch handler throw Extend a0 a0- a0+ a!	(n*x) R(n*y) (n*x) R(n*y) (xt) (t addr) (n) led VM (n) (n) (n) (n)
um/mod um* u/mod 0	(ud u2 rem quot) (u1 u2 d) (u1 u2 rem quot) (0)	['] code : :noname constant does> ." Edefer else end-code	(XT) () () (xt) (n) () () (n <name>) (addr1 addr2) () () R(xt)</name>	abort catch handler throw Extend a0 a0- a0+	(n*x) R(n*y) (n*x) R(n*y) (xt) (addr) (n) led VM (n) (n) (n) (n) (n)
um/mod um* u/mod 0	(ud u2 rem quot) (u1 u2 d) (u1 u2 rem quot) (0)	['] code : :noname constant does> ." Edefer else end-code exit	(XT) () () (xt) (n) () () (n <name>) (addr1 addr2) () () R(xt)</name>	abort catch handler throw Extend a0 a0- a0+ a!	(n*x) R(n*y) (n*x) R(n*y) (xt) (t addr) (n) led VM (n) (n) (n) (n)
um/mod um* u/mod 0	(ud u2 rem quot) (u1 u2 d) (u1 u2 rem quot) (0) etics (d ud)	['] code : :noname constant does> ." Edefer else end-code exit immediate	(XT) () () (xt) (n) () () (n <name>) (addr1 addr2) () () R(xt) e ()</name>	abort catch handler throw Extend a0 a0- a0+ a! a!-	(n*x) R(n*y) (n*x) R(n*y) (xt) (addr) (n) led VM (n) (n) (n) (n) (n)
um/mod um* u/mod 0	(ud u2 rem quot) (u1 u2 d) (u1 u2 rem quot) (0) etics (d ud)	['] code : :noname constant does> ." Edefer else end-code exit immediate [literal	(XT) () () (xt) (n) () (n) (n <name>) (addr1 addr2) () () R(xt) e () (n)</name>	abort catch handler throw Extend a0 a0- a0+ a! a!- a!+ a>	(n*x) R(n*y) (n*x) R(n*y) (xt) (addr) (n) led VM (n) (n) (n) (n) (n) (n) (n) (n)
um/mod um* u/mod 0	(ud u2 rem quot) (u1 u2 d) (u1 u2 rem quot) (0) etics (d ud)	['] code : :noname constant does> ." Edefer else end-code exit immediate [literal ((XT) () () (xt) (n) () (n <- name>) (addr1 addr2) () () R(xt) e () (n) (n)	abort catch handler throw Extend a0 a0- a0+ a! a!- a!+ a> b0	(n*x) R(n*y) (n*x) R(n*y) (xt) (addr) (n) led VM (n) (n) (n) (n) (n) (n) (n) (n) (n)
um/mod um* u/mod 0	(ud u2 rem quot) (u1 u2 d) (u1 u2 rem quot) (0) etics (d ud)	['] code : :noname constant does> ." Edefer else end-code exit immediate [literal (]	(XT) () () (xt) (n) () (n) (n <name>) (addr1 addr2) () (() R(xt) e () (n) () (n)</name>	abort catch handler throw Extend a0 a0- a0+ a! a!- a!+ a> b0 b0-	(n*x) R(n*y) (n*x) R(n*y) (xt) (xt) (addr) (n) led VM (n) (n) (n) (n) (n) (n) (n) (n) (n) (n) (n)
um/mod um* u/mod 0 Arithm dabs	(ud u2 rem quot) (u1 u2 d) (u1 u2 rem quot) (0) etics (d ud)	['] code : :noname constant does> ." Edefer else end-code exit immediate [literal (] Rdefer	(XT) () () () (xt) (n) () (n <name>) (addr1 addr2) () () R(xt) e () (n) (n) (n) (n) (n)</name>	abort catch handler throw Extend a0 a0- a0+ a! a!- a!+ a> b0 b0- b0+	(n*x) R(n*y) (n*x) R(n*y) (xt) (xt) (addr) (n) led VM (n) (n) (n) (n) (n) (n) (n) (n) (n) (n) (n) (n) (n)
um/mod um* u/mod 0 Arithm dabs Charact	(ud u2 rem quot) (u1 u2 d) (u1 u2 rem quot) (0) etics (d ud) ter IO	['] code : :noname constant does> ." Edefer else end-code exit immediate [literal (] Rdefer recurse	(XT) () () (xt) (n) () (n <- name>) (addr1 addr2) () () R(xt) e () (n) (n) (n) () (n) () (n <- name>)	abort catch handler throw Extend a0 a0- a0+ a! a!- a!+ a> b0 b0- b0+ b!	(n*x) R(n*y) (n*x) R(n*y) (xt) (xt) (addr) (n) led VM (n) (n) (n) (n) (n) (n) (n) (n) (n) (n) (n) (n) (n) (n) (n)
um/mod um* u/mod 0 Arithm dabs	(ud u2 rem quot) (u1 u2 d) (u1 u2 rem quot) (0) etics (d ud)	['] code : :noname constant does> ." Edefer else end-code exit immediate [literal (] Rdefer	(XT) () () () (xt) (n) () (n <name>) (addr1 addr2) () () R(xt) e () (n) (n) (n) (n) (n)</name>	abort catch handler throw Extend a0 a0- a0+ a! a!- a!+ a> b0 b0- b0+	(n*x) R(n*y) (n*x) R(n*y) (xt) (xt) (addr) (n) led VM (n) (n) (n) (n) (n) (n) (n) (n) (n) (n) (n) (n) (n) (n) (n) (n) (n) (n)
um/mod um* u/mod 0 Arithm dabs Charact	(ud u2 rem quot) (u1 u2 d) (u1 u2 rem quot) (0) etics (d ud) ter IO	['] code : :noname constant does> ." Edefer else end-code exit immediate [literal (] Rdefer recurse	(XT) () () (xt) (n) () (n <- name>) (addr1 addr2) () () R(xt) e () (n) (n) (n) () (n) () (n <- name>)	abort catch handler throw Extend a0 a0- a0+ a! a!- a!+ a> b0 b0- b0+ b!	(n*x) R(n*y) (n*x) R(n*y) (xt) (xt) (addr) (n) led VM (n) (n) (n) (n) (n) (n) (n) (n) (n) (n) (n) (n) (n) (n) (n)
um/mod um* u/mod 0 Arithm dabs Charact	(ud u2 rem quot) (u1 u2 d) (u1 u2 rem quot) (0) etics (d ud) ter IO (32) ()	['] code : :noname constant does> ." Edefer else end-code exit immediate [literal (] Rdefer recurse s,	(XT) () () (xt) (n) () (n < name >) (addr1 addr2) () () R(xt) e () (n) (n) (n) (n) (n < name >) (addr len)	abort abort" catch handler throw Extend a0 a0- a0+ a! a!- a!+ a> b0 b0- b0+ b! b!-	(n*x) R(n*y) (n*x) R(n*y) (xt) (xt) (addr) (n) led VM (n) (n) (n) (n) (n) (n) (n) (n) (n) (n) (n) (n) (n) (n) (n) (n) (n) (n)
um/mod um* u/mod 0 Arithm dabs Charact bl cr emit emit?	(ud u2 rem quot) (u1 u2 d) (u1 u2 rem quot) (0) etics (d ud) ter IO (32) () (c) (f)	['] code : :noname constant does> ." Edefer else end-code exit immediate [literal (] Rdefer recurse s, ; s"	(XT) () () (xt) (n) () (n < name >) (addr1 addr2) () () R(xt) e () (n) (n) (n) (n) (n) (n) (n) (n < name >) (n < name >) () (addr len) () (< cchar >)	abort abort" catch handler throw Extend a@ a@- a@+ a! a!- a!+ a> b@ b@- b@+ b! b!- b!+ b>	(n*x) R(n*y) (n*x) R(n*y) (xt) (xt) (xt) (n) led VM (n)
um/mod um* u/mod 0 Arithm dabs Charact bl cr emit emit? key	(ud u2 rem quot) (u1 u2 d) (u1 u2 rem quot) (0) etics (d ud) (32) () (c) (c) (f) (c)	['] code : :noname constant does> ." Edefer else end-code exit immediate [literal (] Rdefer recurse s, ; s" state	(XT) () () (xt) (n) () (n < name >) (addr1 addr2) () () R(xt) e () (n) (n) (n) (n) (n) (n) (n) (n < name >) (n < name >) () (addr len) () (<cchar>) (addr)</cchar>	abort catch handler throw Extend a0 a0- a0+ a! a!- a!+ a> b0 b0- b0+ b! b!- b!+ b> na0	(n*x) R(n*y) (n*x) R(n*y) (xt) (xt) (addr) (n) led VM (n)
um/mod um* u/mod 0 Arithm dabs Charact bl cr emit emit? key key?	(ud u2 rem quot) (u1 u2 d) (u1 u2 rem quot) (0) etics (d ud) (32) () (c) (f) (c) (f)	['] code : :noname constant does> ." Edefer else end-code exit immediate [literal (] Rdefer recurse s, ; s" state then	(XT) () () () (xt) (n) () (n <name>) () R(xt) e () (n)</name>	abort abort" catch handler throw Extend a0 a0- a0+ a! a!- a!+ a> b0 b0- b0+ b! b!- b!+ b> na0 na!	(n*x) R(n*y) (n*x) R(n*y) (xt) (xt) (addr) (n) led VM (n)
um/mod um* u/mod 0 Arithm dabs Charact bl cr emit emit? key key? /key	(ud u2 rem quot) (u1 u2 d) (u1 u2 rem quot) (0) etics (d ud) ter IO (32) () (c) (f) (c) (f) ()	['] code : :noname constant does> ." Edefer else end-code exit immediate [literal (] Rdefer recurse s, ; s" state then until	(XT) () () (xt) (n) () (n) (n < name >) (addr1 addr2) () R(xt) e () (n) (n) (n < name >)	abort catch handler throw Extend a0 a0- a0+ a! a!- a!+ a> b0 b0- b0+ b! b!- b!+ b> na0 na! nb0	(n*x) R(n*y) (n*x) R(n*y) (xt) (xt) (addr) (n) led VM (n) (offs n) (offs n)
um/mod um* u/mod 0 Arithm dabs Charact bl cr emit emit? key key?	(ud u2 rem quot) (u1 u2 d) (u1 u2 rem quot) (0) etics (d ud) (32) () (c) (f) (c) (f)	['] code : :noname constant does> ." Edefer else end-code exit immediate [literal (] Rdefer recurse s, ; s" state then	(XT) () () () (xt) (n) () (n <name>) () R(xt) e () (n)</name>	abort abort" catch handler throw Extend a0 a0- a0+ a! a!- a!+ a> b0 b0- b0+ b! b!- b!+ b> na0 na!	(n*x) R(n*y) (n*x) R(n*y) (xt) (xt) (addr) (n) led VM (n) (n ffs) (offs n) (n offs)
um/mod um* u/mod 0 Arithm dabs Charact bl cr emit emit? key key? /key	(ud u2 rem quot) (u1 u2 d) (u1 u2 rem quot) (0) etics (d ud) ter IO (32) () (c) (f) (c) (f) ()	['] code : :noname constant does> ." Edefer else end-code exit immediate [literal (] Rdefer recurse s, ; s" state then until	(XT) () () (xt) (n) () (n) (n < name >) (addr1 addr2) () R(xt) e () (n) (n) (n < name >)	abort catch handler throw Extend a0 a0- a0+ a! a!- a!+ a> b0 b0- b0+ b! b!- b!+ b> na0 na! nb0	(n*x) R(n*y) (n*x) R(n*y) (xt) (xt) (addr) (n) led VM (n) (offs n) (offs n)
um/mod um* u/mod 0 Arithm dabs Charact bl cr emit emit? key key? /key space	(ud u2 rem quot) (u1 u2 d) (u1 u2 rem quot) (0) etics (d ud) ter IO (32) () (c) (f) (c) (f) () ()	['] code : :noname constant does> ." Edefer else end-code exit immediate [literal (] Rdefer recurse s, ; s" state then until user	(XT) () () (xt) (n) () (n <- name>) (addr1 addr2) () R(xt) e () (n) (n) (n) (n) () (addr len) () (addr) (addr) (n) (n)	abort catch handler throw Extend a0 a0- a0+ a! a!- a!+ a> b0 b0- b0+ b! b!- b!+ b> na0 na! nb0 nb!	(n*x) R(n*y) (n*x) R(n*y) (xt) (xt) (addr) (n) led VM (n) (n offs) (offs n) (n offs)

```
Hardware Access
                                  Numeric IO
                                                                     System
           ( -- c)
                                              ( -- addr )
                                    base
           ( -- f)
 rx0?
                                                                                 ( addr n1 -- n2 )
                                                                       accept
                                              ( d1 -- )
           ( -- )
                                                                                 ( n -- )
 >usart0
                                                                       allot
                                    d.r
                                              ( d1 n -- )
                                                                                 ( -- )
           (c -- )
 tx0
                                                                       cold
                                    decimal
                                              ( -- )
 tx0?
           ( -- f)
                                              ( c base -- number flag )defer@
                                                                                 ( xt1 -- xt2 )
                                    digit
           ( -- )
 +usart0
                                                                       defer!
                                                                                 ( xt1 xt2 -- )
                                              ( n -- )
                                                                       execute
                                                                                 ( xt -- )
                                              ( n1 n2 -- )
                                                                                 ( -- f_cou )
                                                                       f_cpu
                                              ( -- )
                                    hex
                                                                       >in
                                                                                 ( -- addr )
                                              ( -- addr )
                                    hld
IO
                                                                       interpret ( -- )
                                    hold
                                              ( c -- )
                                                                                 ; R(i*x - j*x )
                                              ( -- )
                                    <#
                                                                                 ( xt1 c<char> -- )
                                                                       is
                                              (addr -- n )
 refill
           ( -- f )
                                    number
                                                                       #t.ib
                                                                                 ( -- addr )
                                              ( d1 -- d2 )
                                                                       ?execute ( xt|0 -- )
                                              ( d1 -- addr count )
                                    #>
                                                                       quit
                                                                                 ( -- )
                                              ( d1 -- 0)
                                    #s
                                                                                 ( -- addr n )
                                              ( n -- )
                                                                       source
                                    sign
Interrupt
                                                                       up@
                                                                                 ( -- addr )
                                              ( ud -- )
                                    ud.
                                                                                 ( addr -- )
                                                                       up!
                                    ud.r
                                              ( ud n -- )
           ( i -- xt )
 int@
                                              ( d1 -- )
                                    u.
           ( -- sreg )
 -int
                                              ( d n -- )
                                    u.r
           ( -- )
 +int
                                              (un--)
                                    u0.r
           ( xt i -- )
 int!
 #int
           ( -- n )
                                                                      System Value
                                   Stack
Logic
                                                                                 ( -- v)
                                    depth
                                              ( -- n )
                                                                                 ( -- edp)
                                                                       edp
                                              ( n -- )
                                                                                 ( -- faddr)
           ( n1 n2 -- n3 )
                                    drop
                                                                       head
           ( n1 -- n2 )
                                    dup
                                              ( n -- n n )
                                                                                 ( -- addr)
                                                                       heap
 negate
                                    over
                                              ( n1 n2 -- n1 n2 n1 )
                                                                                 ( -- faddr )
           (flag -- flag')
                                                                       here
           ( n1 n2 -- n3 )
( n1 n2 -- n3)
                                              ( n1 -- [ n1 n1 ] | 0)
                                                                                 ( -- addr )
                                    ?dup
                                                                       pad
 or
                                              ( n1 n2 n3 -- n2 n3 n1) tib
                                                                                 ( -- addr )
                                    rot
 xor
                                              ( -- n)
                                    r@
                                                                                ( -- n )
                                                                       tibsize
                                              R(n -- n)
                                                                       turnkey
                                                                                 ( -- n*y )
                                              ( -- n )
                                              ; R( n --)
MCU
                                              ( n1 n2 -- n2 n1)
                                    swap
                                    >r
                                              ( n -- )
           ( -- )
 -jtag
           ( -- )
                                              ; R( -- n)
 -wdt
                                                                      Time
           ( -- )
           ( txbyte -- rxbyte)
 spirw
           ( -- )
 wdr
                                                                                 ( -- )
                                                                       1ms
                                   Stackpointer
                                    rp0
                                              ( -- addr)
Memory
                                              ( -- n)
                                    rp@
                                              (n --)
                                    rp!
           ( addr - c1 )
                                              ; R( -- xy)
                                                                      Tools
           (addr-from addr-to n --
                                   )
sp0
 cmove
                                              ( -- addr)
 cmove>
           (addr-from addr-to n --
                                              ( -- addr)
           ( c addr -- )
 c!
                                    sp@
                                              ( -- n)
                                                                       [char]
                                                                                 ( -- c )
           (addr - n)
 e@
                                    sp!
                                              ( addr -- i*x)
                                                                                 ( -- c )
                                                                       char
 e!
           ( n addr -- )
                                                                                 ( c-addr search
start -- [ addr 0 ] | [ xt [-:
                                                                       (find)
           ( addr -- n )
 @
                                                                                 ( -- )
                                                                       .s
 fill
           ( c-addr u c -- )
                                                                                 ( addr -- [ addr 0 ] | [ xt [-1|1]] )
                                                                       find
 i@
           ( addr -- n1 )
                                                                                 ( addr-ram addr-flash -- f)
                                                                       icompare
           ( n addr -- )
                                   String
 i!
                                                                                 ( adr -- adr n )
                                                                       icount
           ( n addr -- )
                                                                                 ( addr n -- )
                                                                       itype
                                              ( addr -- addr+1 n)
                                                                                 ( -- )
                                    count
                                                                       noop
                                                                                 (n < name > --)
                                              ( addr1 n1 c -- addr1 n2 t)o
                                    cscan
                                              ( addr1 n1 c -- addr2 n2 whused
( char "ccc" -- c-addr u wer
                                    cskip
                                                                                 ( -- n )
Multitasking
                                                                                 ( -- )
                                    parse
                                              ( addr1 len1 addr2 -- ) word
                                                                                 ( c -- addr )
                                    place
```

(addr1 u1 n-- addr2 u2)words

(--)

(--)

pause

internal/hidden

```
(branch) (-- )
(?branch) (f -- )
(constant)(-- addr )
(create) (-- )
         (limit counter -- )
(do)
           R(-- limit counter)
(does>) (-- )
(defer) (i*x -- j*x )
(literal) (-- n1 )
         (-- )
(loop)
           R(limit counter -- limit counter+1|)
(+loop) (n1 -- )
           R(llimit counter -- limit counter+n1|)
           (limit counter -- )
(?do)
           R(-- limit counter| )
(rp0)
          ( -- addr)
( -- addr)
(sp0)
           (spmcsr x addr -- )
(spm)
           ( n -- )
R(IP -- IP+1)
(to)
(user)
          (-- addr )
(variable)(-- addr )
Edefer@ ( xt1 -- xt2 )
Edefer! ( xt1 xt2 -- )
>mark
         ( -- addr )
>resolve ( addr -- )
hiemit (w -- )
int_restor(e sreg -- )
<mark ( -- addr )
<resolve ( addr -- )
Rdefer@ ( xt1 -- xt2 )
Rdefer! ( xt1 xt2 -- )
(sliteral)( -- addr n)
spmbuf (x addr -- )
spmerase (addr -- )
spmpageloa(daddr -- )
spmrww (-- )
spmrww? (--)
spmwrite (spmcsr x addr -- )
Udefer@ ( xt1 -- xt2 )
Udefer! ( xt1 xt2 -- )
```