```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; This is a 430eForth assembler listing based on the original script from
; Dr. Chen Hanson Ting as described in his book: Zen and the Forth Language:
; EFORTH for the MSP430 from Texas Instruments (Kindle Edition). It was first
; written by him in IAR Assembler, then transferred to the CCS and later
; adapted for the naken_asm by Michael Kalus    .
;
; Manfred Mahlow added Flash Tools and support for WORDLISTs, VOCs (VOCABULARY
; Prefixes), ITEMs and STICKY Words (VIS).
;
; Initially the assembler listing was made for the MSP430G2553 LaunchPad. This
; is now a first attempt to make it a template that may be used to also create
; 430eForth for other MSP430 MCUs.
;
; Conditional assembling is used to replace some of the code written for the
; MSP430G2553 with code required for the other target. Assembling is controlled
; by the assembler constant MCU.

; This template supports the MSP430FR5969 (Default, MCU = 0) and the MSP430G2553
; (Reference system, MCU = 1).
;
; !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
;
; The intention of this listing is to encourage other Forth Users to make the
; 430eForth available for more MSP430 MCUs by modifying the MSP430FR5969 code
; sections as required for the other target.
;
; Please send us your listing to be published in the Wiki on forth-ev.de.
;
; Feb. 2022 , Manfred Mahlow  manfred.mahlow@forth-ev.de
;             Michael Kalus   mik.kalus@gmail.com
;
; !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
;
; MIT License
; ------------------------------------------------------------------------
; Copyright (c) 2014 Dr. Chen-Hanson Ting  CCS Version    430eForth4.3
;           (c) 2018 Michael Kalus         Naken Version  430eForth4.3n
;           (c) 2018 Manfred Mahlow        Flash Tools    430eForth4.3n1
;           (c) 2019 Manfred Mahlow        430eForth-g2553lp-43n6vis
;           (c) 2021 Manfred Mahlow        NOAPP function added to P1.3 to
;                                          disable a buggy APP on cold start.
;           (c) 2022 Manfred Mahlow        Code for MSP430FR5969 added
;           (c) 2022 Michael Kalus         FR5969 code modified for FR5739
;
; Permission is hereby granted, free of charge, to any person obtaining a copy
; of this software and associated documentation files (the "Software"), to deal
; in the Software without restriction, including without limitation the rights
; to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
; copies of the Software, and to permit persons to whom the Software is
; furnished to do so, subject to the following conditions:
;
; The above copyright notice and this permission notice shall be included in all
; copies or substantial portions of the Software.
;
; THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
; IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
; FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
; AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
; LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
; OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
; SOFTWARE.
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; 7/7/2012 430eForth1.0, from eForth86.asm and 430uForth
; 7/4/2012 Move 430uForth2.1 from IAR to CCS 5.2
; 8/5/2014 Move 430eForth2.2 to CCS 6.0.  Fix linkage of OVER.
;   Software UART at 2400 baud.
; 8/10/2014 430eForth2.3 9600 baud, thanks to Dirk Bruehl and
;   Michael Kalus of www.4e4th.org
```

```
 73  ; 8/10/2014 430eForth2.4 Restore ERASE and WRITE
 74  ; 8/20/2014 430eForth2.5 Test Segment D
 75  ; 8/25/2014 430eForth2.6 Turnkey
 76  ; 8/26/2014 430eForth2.7 Optimize
 77  ; 9/16/2014 430eForth3.1 Tail recursion, APP!
 78  ; 10/11/2014 430eForth4.1 Direct thread, more optimization
 79  ; 10/23/2014 430eForth4.2 Direct thread, pack lists
 80  ; 11/12/2014 430eForth4.2 Direct thread, final
 81  ;
 82  ; Build for and verified on MSP430G2 LaunchPad from TI
 83  ; Assembled with Code Composer Sudio 6.0 IDE
 84  ; Internal DCO at 8 MHz
 85  ; Hardware UART at 9600 baud. TXD and RXD must be crossed.
 86
 87  ;CCS:   ;ting
 88  ;   .nolist
 89  ;   .title "msp430 eForth 4.3"
 90  ;   .cdecls C,LIST,"msp430g2553.h"  ; Include device header file
 91
 92  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
 93  ;
 94  ; 05/13/2018 Moved 430eForth4.3 from CSS to Michael Kohn's naken_asm (ver.
 95  ;            23 april 2018) - ok                                          ;mk
 96  ; 20180624 $," - bug fixed. Thanks to Manfred Mahlow.                      ;mk
 97  ; 20180628 DIGIT? : bug fix, 0= 0> were handled as numbers                 ;MM
 98  ; 20180629 FSCAN added, sets CP to the lowest free flash addr at BOOT time.
 99  ;          FSCAN is executed before COLD executes QUIT.
100  ;          Version string changed in HI from 43n to 43n1.                 ;MM
101  ; 20180630 ERASE and WRITE renamed to IERASE IWRITE due to name conflict  ;MM
102  ; 20180701 LITERAL and ALIGNED revealed.                                  ;MM
103  ; 20180707 Flash Test QFLASH ( a -- a ) added. Aborts with message ?flash
104  ;          if a > EDM. (EDM = End of Dictionary Memory space in the flash) ;MM
105  ; 20180720 Support for tags based wordlists and VOCs added. Default search
106  ;          order is FORTH ROOT. VOCs search order is <VOC> ROOT. ( new: ROOT
107  ;          FORTH DEFINITIONS VOC )                                        ;MM
108  ; 20180730 Implicit context switching added ( new: CASTED )               ;MM
109  ; 20180826 CASTED renamed to ITEM and moved to a Forth source code file   ;MM
110  ; 20181120 FORTH and ROOT are vocabularies now. vROOT added for VOCS      ;MM
111  ; 20181124 Context Switching is done after a word is executed or compiled. ;MM
112  ; 20181207 FORTH and ROOT are VOCs, vROOT removed,FIRST added             ;MM
113  ; 20190106 VOCs are no longer implemented as ITEMS.
114  ;          ITEM and STICKY added again.                                   ;MM
115  ; 20190131 Macros added to create headers and tags                        ;MM
116  ; 20190208 DOLIT,0 replaced with constant ZERO                           ;MM
117  ; 20190208 DOLIT,1,ANDD replaced with GETB0                              ;MM
118  ; 20190314 Bug fix for UM/MOD and TOKEN                                   ;MM
119  ; 20190330 FIRST removed ALSO added                                      ;MM
120  ; 20190716 FIRST added again, ALSO removed, can be loaded from a file    ;MM
121  ; 20190717 ABORT" made compile only                                      ;MM
122  ; 20191024 INEST changed from call #DOLIT to call R15 with R15 = DOLIT   ;MM
123  ;                requires 2 bytes and 2 clocks less per colon definition,
124  ;                taken from eForth for MSP430FR5969 by David Schultz
125  ; 20210420 CaseSensitive Flag and case-insensitive versions of SAME? and
126  ;          FIND added. Default is now case-insensitive dictionary search.  ;MM
127  ; 20211006 Header of DIGIT made visible                                   ;MM
128  ; 20211207 WORDS changed                                                  ;MM
129  ;
130  ; 20220116 First approach to add support for FRAM based MSP430 MCUs      ;MM
131  ; 20220121 UP added                                                      ;MM
132  ; 20220122 User variable CNTXT (old eForth CONTEXT) renamed to DIC (DICC)  ;MM
133  ; 20220123 MCUID added to support MCU specific uploading of source code  ;MM
134  ;
135  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
136
137  ;naken: ;mk
138      .msp430
139
140  MCU equ 0
141        ; 0 : FR5739 <-- FR5969 : FR5739 generated from FR5969 ;mk
142        ; 1 : G2553
143
144  ; Include MCU specific register data
```

```
145  .if MCU
146     .include "msp430g2553.inc"
147  .else
148  ;   .include "msp430fr5969.inc"
149     .include "msp430fr5739.inc"
150  .endif
151
152  .include "header.inc"  ; add header macros  MM-190130
153
154  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
155
156  ; Direct Thread Model of eForth
157
158  ; CCS: .equ ; naken: equ   ;mk
159  ;;  CPU registers
160
161  tos     equ  R4
162  stack   equ  R5
163  ip      equ  R6
164  temp0   equ  R7
165  temp1   equ  R8
166  temp2   equ  R9
167  temp3   equ  R10
168
169  callR15 equ 128FH  ; MM++191024   call #DOLST is replaced with call R15
170                     ;              2 clocks less, 2 bytes less
171                     ;              R15 is set to DOLST in init
172
173  ;; Macros
174  ; CCS: <name> .macro ; naken: .macro <name>   ;mk
175
176  .macro pops ;DROP
177      mov.w   @stack+,tos
178      .endm
179
180  .macro pushs ;DUP
181      decd.w  stack
182      mov.w   tos,0(stack)
183      .endm;; Constants
184
185  .macro INEXT ;mk renamed (dollar)NEXT to INEXT - inline code for NEXT.
186      mov @ip+,pc ; fetch code address into PC
187      .endm
188
189  .macro INEST ;mk renamed (dollar)NEST to INEST - inline code for NEST.
190      .align 16   ; CCS: 2 bytes align ; naken: 16 bit align. mk
191  ;   call #DOLST ; fetch code address into PC, W=PFA
192      call R15    ; MM-191024
193  .endm
194
195  .macro ICONST ;mk renamed (dollar)CONST to ICONST - inline code calling DOCON.
196      .align 16   ; CCS: 2 bytes align ; naken: 16 bit align. mk
197      call #DOCON ; fetch code address into PC, W=PFA
198      .endm
199
200  .macro IVOC   ; MM++191024
201      .align 16
202      call R15    ; R15 = DOLST, indirect call of DOLST
203      DW DOVP
204  .endm
205
206  ;; Assembler constants
207
208  XTAG   equ  020H    ;lexicon tag bit                       ;MM++180712
209  COMPO  equ  040H    ;lexicon compile only bit
210  IMEDD  equ  080H    ;lexicon immediate bit
211  MASKK  equ  07F1FH  ;lexicon bit mask
212  CELLL  equ   2      ;size of a cell
213  BASEE  equ  10      ;default radix
214  ; VOCSS  equ   8      ;depth of vocabulary stack          ;not used with VIS
215  BKSPP  equ   8      ;backspace
216  LF     equ  10      ;line feed
```

```
217  CRR    equ  13      ;carriage return
218  ERR    equ  27      ;error escape
219  TIC    equ  39      ;tick
220  CALLL  equ  012B0H  ;NOP CALL opcodes
221
222
223  .if MCU ; G2553
224
225  SOR    equ  200H       ;start of SRAM                     ;MM+220119
226  EOR    equ  3FFH       ;end of SRAM                       ;
227
228  UPP    equ  SOR        ;start of USER variables in RAM
229  DPP    equ  UPP+020H   ;start of data memory in RAM
230  SPP    equ  TIBB-2     ;SP0, start of data stack
231  TIBB   equ  RPP-120    ;TIB0, start of terminal input buffer
232  RPP    equ  EOR-7      ;RP0, start of return stack
233
234  CODEE  equ  0C000H     ;code dictionary
235  COLDD  equ  0FFFEH     ;cold start vector
236  EM     equ  0FFFFH     ;top of memory
237
238  INFOD  equ  01000H     ;save area for the user variables   ;MM+220119
239
240  ; .else  ; FR5969                     ;MM+220119
241  .else  ; FR5739                      ;MK+220210
242
243  SOR    equ  01C00H     ;start of SRAM FR5739 == FR5969
244  ;EOR    equ  023FFH     ;end of SRAM FR5969
245  EOR    equ  01FFFH     ;end of SRAM FR5739
246
247  UPP    equ  SOR        ;start of USER variables in RAM
248  DPP    equ  UPP+020H   ;start of data memory in RAM
249  SPP    equ  TIBB-2     ;SP0, start of data stack
250  TIBB   equ  RPP-120    ;TIB0, start of terminal input buffer
251  RPP    equ  EOR-7      ;RP0, start of return stack
252
253  ;CODEE  equ  04400H     ;start of Forth code/data in FRAM FR5969
254  CODEE  equ  0C200H     ;start of Forth code/data in FRAM FR5739
255
256  COLDD  equ  0FFFEH     ;cold start vector
257  EM     equ  0FFFFH     ;end of code/data memory
258
259  INFOD  equ  01800H     ;save area for the user variables
260
261  .endif
262
263
264  .equ CaseSensitive=0  ; 0 = case-insensitive dictionary seearch   ;MM++210420
265                        ; 1 = case-sensitive
266                        ; see find and same?
267
268  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
269
270     .list
271
272  ;; Main entry points and COLD start data
273
274     .org CODEE
275
276  main:
277
278  .set link = 0H        ; Start of the dictionary linked list  MM++190130
279
280  ;; Device dependent I/O
281
282  ; ?KEY  ( -- F | c T )
283  ; Return input character.
284    HEAD( 4,"?key" )
285  QKEY:
286    pushs
287  QKEY1:
288  .if MCU ; G2553
```

```
289     BIT.B #UCA0RXIFG,&IFG2
290   .else   ; FR5969 == FR5739
291     BIT.B #UCRXIFG,&UCA0IFG
292   .endif
293     JZ FALSE              ;return false flag
294     MOV.B &UCA0RXBUF,tos  ; read character into TOS
295     pushs
296     jmp TRUE
297
298   ; KEY   ( -- c )
299   ; Return input character.
300     HEAD( 3,"KEY" )
301   KEY:
302     pushs
303   KEY1:
304   .if MCU ; G2553
305     BIT.B #UCA0RXIFG,&IFG2
306   .else   ; FR5969 == FR5739
307     BIT.B #UCRXIFG,&UCA0IFG
308   .endif
309     JZ KEY1
310     MOV.B   &UCA0RXBUF,tos    ; read character into TOS
311     INEXT
312
313   ; EMIT  ( c -- )
314   ;   Send character c to the output device.
315     HEAD( 4,"EMIT" )
316   EMIT:
317   EMIT1:
318   .if MCU ; G2553
319       BIT.B   #UCA0TXIFG,&IFG2
320   .else   ; FR5969 == FR5739
321       BIT.B   #UCTXIFG,&UCA0IFG
322   .endif
323       JZ  EMIT1
324       MOV.B   tos,&UCA0TXBUF
325       pops
326       INEXT
327
328   ; !IO   ( -- )
329   ;   Initialize the I/O devices.
330   ;   .dw EMIT-6
331   ;   .db 3,"!IO"
332   ; HEAD( 3,"!IO" )
333   STOIO:
334   .if MCU ; G2553
335
336     ; 8MHz
337       mov.b   &CALBC1_8MHZ, &BCSCTL1   ; Set DCO
338       mov.b   &CALDCO_8MHZ, &DCOCTL    ; to 8 MHz.
339       mov.b   #006h, &P1SEL    ; Use P1.1/P1.2 for USCI_A0
340       mov.b   #006h, &P1SEL2   ; Use P1.1/P1.2 for USCI_A0
341     ; Configure UART (Koch)
342       bis.b   #UCSSEL_2,&UCA0CTL1 ;db2 SMCLK
343       mov.b   #65,&UCA0BR0    ;db3 8MHz 9600 Insgesamt &833 = $341
344       mov.b   #3,&UCA0BR1 ;db4 8MHz 9600
345       mov.b   #UCBRS_2,&UCA0MCTL  ;db5 Modulation UCBRSx = 2
346       bic.b   #UCSWRST,&UCA0CTL1  ;db6 **Initialize USCI
347
348       bis.b #BIT0,&P1DIR  ; init P1.0 as output
349       bis.b #BIT0,&P1OUT  ; set LED0 after I/O init to signal eForth startup
350
351   .else  ; FR5739 <-- FR5969
352
353    .if 1 ; == B9600 with SMCLK 8MHz ==
354
355       mov #CSKEY,&CSCTL0   ; unlock CS registers
356
357   ;    mov #0x000C,&CSCTL1  ; DCO=8MHz, default FR5969
358       mov #0x0006,&CSCTL1  ; DCO=8MHz, default FR57xx
359       mov #0x0033,&CSCTL2  ; DCO for MCLK and SMCLk and LFXTCLK|VLOCLK for ACLK
360       mov #0x0000,&CSCTL3  ; ACLK=LFXT|VLO , SMCLK=DCO , MCLK=DCO
```

```
361
362       clr.b  &CSCTL0_H     ; lock CS registers
363
364       mov #3000,temp0      ; wait for stable clocks, otherwise the boot message
365     clkwait:               ; starts with garbidge when COLD is executed from the
366       sub #1,temp0         ; command line.
367       jnz clkwait
368
369       bis #UCSWRST,&UCA0CTLW0  ; reset UART
370       mov #(UCSSEL_2|UCSWRST),&UCA0CTLW0  ;  use SMCLK, 8N1
371       mov #52,&UCA0BRW         ; set divider  8MHz 9600
372       mov #0x4911,&UCA0MCTLW   ; UCBRSx=0x49, UCBRFx=1, UCOS16=1
373
374    .else  ; == B9600 with SMCLK = 1MHz ==
375
376       mov #CSKEY,&CSCTL0   ; unlock CS registers
377
378       mov #0x000C,&CSCTL1  ; DCO=8MHz, default  ????mk
379       mov #0x0033,&CSCTL2  ; DCO for MCLK and SMCLk and LFXTCLK|VLOCLK for ACLK
380       mov #0x0030,&CSCTL3  ; ACLK=LFXT|VLO , SMCLK=DCO/8 , MCLK=DCO
381
382       clr.b &CSCTL0_H      ; lock CS registers  MM-220119
383
384       mov #3000,temp0      ; wait for stable clocks, otherwise the boot message
385    clkwait:                 ; starts with garbidge when COLD is executed from the
386       sub #1,temp0         ; command line.
387       jnz clkwait
388
389       bis #UCSWRST,&UCA0CTLW0  ; reset UART
390       mov #(UCSSEL_2|UCSWRST),&UCA0CTLW0  ;  use SMCLK, 8N1
391       mov #6,&UCA0BRW          ; set divider  1MHz 9600
392       mov #0x2081,&UCA0MCTLW   ; UCBRSx=0x20, UCBRFx=8, UCOS16=1
393
394    .endif
395
396       bis.b   #3,&P2SEL1       ; assign pins to TXD and RXD
397       bic.b   #3,&P2SEL0
398       bic #UCSWRST,&UCA0CTLW0 ; bring UART out of reset
399
400    ;   bis.b #BIT0,&P1DIR  ; init P1.0 as output
401    ;    bis.b #BIT0,&P1OUT  ; set LED0 after I/O init to signal eForth startup
402
403    ;mk Enable and set LED1..8 of MSP-EXP430FR5739 LP
404       BIS.B   #0x0f,&PJDIR ; LED1..4
405       BIC.B   #0x0f,&PJOUT ; LEDs off
406       BIS.B   #0x08,&PJOUT ; LED4 on
407       BIS.B   #0xf0,&P3DIR ; LED5..8
408       BIC.B   #0xF0,&P3OUT ; LEDs off
409       BIS.B   #0x10,&P3OUT ; LED5 on
410
411       bic #LOCKLPM5,&PM5CTL0   ;  activate i/o port settings
412
413    .endif
414       INEXT
415
416
417    ;; The kernel
418
419    ; doLIT ( -- w )
420    ;   Push an inline literal.
421    ; HEAD( COMPO+5,"doLIT" )
422    DOLIT:
423       pushs
424       mov @ip+,tos
425       INEXT
426
427    ; doCON ( -- a )
428    ;   Run time routine for CONSTANT, VARIABLE and CREATE.
429    ; HEAD( COMPO+5,"doCON" )
430    DOCON:
431       pushs
432       pop tos
```

```
433      mov @tos,tos
434      INEXT
435
436   ; doLIST    ( -- )
437   ;   Process colon list..
438   ;   HEAD( 6,"doLIST" )
439   DOLST:
440      mov ip,temp0    ;exchange pointers
441      pop ip  ;push return stack
442      push    temp0   ;restore the pointers
443      INEXT
444
445   ; EXIT  ( -- )
446   ;   Terminate a colon definition.
447     HEAD( 4,"EXIT" )
448   EXIT:
449      mov @sp+,ip
450      INEXT
451
452   ; EXECUTE   ( ca -- )
453   ;   Execute the word at ca.
454      HEAD( 7,"EXECUTE" )
455   EXECU:
456      mov tos,temp0
457      pops
458      br  temp0
459
460   ; @EXECUTE  ( a -- )
461   ;   Execute vector stored in address a.
462      HEAD( 8,"@EXECUTE" )
463   ATEXE:
464      mov @tos,temp0
465      pops
466      br  temp0
467
468   ; branch    ( -- )
469   ;   Branch to an inline address.
470   ;   HEAD( COMPO+6,"BRANCH" )
471   BRAN:
472      mov @ip+,ip
473      INEXT
474
475   ; ?branch   ( f -- )
476   ;   Branch if flag is zero.
477   ;   HEAD( COMPO+7,"?BRANCH" )
478   QBRAN:
479      tst tos
480      pops
481      jz  BRAN
482      jmp SKIP
483
484   ; next  ( -- )  MM-210904
485   ; ?next ( -- )
486   ;   Run time code for the single index loop.
487   ;   : next ( -- ) \ hilevel model
488   ;   r> r> dup if 1 - >r @ >r exit then drop cell+ >r ;
489   ;   HEAD( COMPO+4,"?next" )
490   DONXT:
491      dec 0(sp)   ;decrement index
492      jge BRAN    ;loop back
493      incd.w  sp  ;discard index
494   SKIP:
495      incd.w  ip  ;exit loop
496      INEXT
497
498   ; ! ( w a -- )
499   ;   Pop the data stack to memory.
500      HEAD( 1,"!" )
501   STORE:
502      mov.w   @stack+,0(tos)
503      pops
504      INEXT
```

```
505
506   ; @ ( a -- w )
507   ;   Push memory location to the data stack.
508       HEAD( 1,"@" )
509   AT:
510       mov.w   @tos,tos
511       INEXT
512
513   ; C!    ( c b -- )
514   ;   Pop the data stack to byte memory.
515       HEAD( 2,"C!" )
516   CSTOR:
517       mov.b   @stack+,0(tos)
518       inc stack
519       pops
520       INEXT
521
522   ; C@    ( b -- c )
523   ;   Push byte memory location to the data stack.
524       HEAD( 2,"C@" )
525   CAT:
526       mov.b   @tos,tos
527       INEXT
528
529   ; RP! ( -- )
530   ;   init return stack pointer.
531   ;   HEAD( 3,"RP!" )
532   RPSTO:
533       mov #RPP,SP ;init return stack
534       INEXT
535
536   ; R> ( -- w )
537   ;   Pop the return stack to the data stack.
538     HEAD( 2,"R>" )
539   RFROM:
540       pushs
541       pop tos
542       INEXT
543
544   ; R@ ( -- w )
545   ;   Copy top of return stack to the data stack.
546       HEAD( 2,"R@" )
547   RAT:
548       pushs
549       mov 0(sp),tos
550       INEXT
551
552   ; >R ( w -- )
553   ;   Push the data stack to the return stack.
554       HEAD( COMPO+2,">R" )
555   TOR:
556       push    tos
557       pops
558       INEXT
559
560   ; SP! ( -- )
561   ;   Init data stack pointer.
562   ;   HEAD( 3,"SP!" )
563   SPSTO:
564       mov #SPP,stack  ;init parameter stack
565       clr tos
566       INEXT
567
568   ; DROP ( w -- )
569   ;   Discard top stack item.
570       HEAD( 4,"DROP" )
571   DROP:
572       pops
573       INEXT
574
575   ; DUP ( w -- w w )
576   ;   Duplicate the top stack item.
```

```
577      HEAD( 3,"DUP" )
578  DUPP:
579      pushs
580      INEXT
581
582  ; SWAP ( w1 w2 -- w2 w1 )
583  ;   Exchange top two stack items.
584      HEAD( 4,"SWAP" )
585  SWAP:
586      mov.w   tos,temp0
587      mov.w   @stack,tos
588      mov.w   temp0,0(stack)
589      INEXT
590
591  ; OVER ( w1 w2 -- w1 w2 w1 )
592  ;   Copy second stack item to top.
593      HEAD( 4,"OVER" )
594  OVER:
595      mov.w   @stack,temp0
596      pushs
597      mov.w   temp0,tos
598      INEXT
599
600  ; 0< ( n -- t )
601  ;   Return true if n is negative.
602      HEAD( 2,"0<" )
603  ZLESS:
604      tst tos
605      jn  TRUE
606  FALSE:
607      clr tos
608      INEXT
609  TRUE:
610      mov #0x-1,tos
611      INEXT
612
613  ; AND ( w w -- w )
614  ;   Bitwise AND.
615      HEAD( 3,"AND" )
616  ANDD:
617      and @stack+,tos
618      INEXT
619
620  ; OR ( w w -- w )
621  ;   Bitwise inclusive OR.
622      HEAD( 2,"OR" )
623  ORR:
624      bis @stack+,tos
625      INEXT
626
627  ; XOR ( w w -- w )
628  ;   Bitwise exclusive OR.
629      HEAD( 3,"XOR" )
630  XORR:
631      xor @stack+,tos
632      INEXT
633
634  ; UM+ ( w w -- w cy )
635  ;   Add two numbers, return the sum and carry flag.
636      HEAD( 3,"UM+" )
637  UPLUS:
638      add @stack,tos
639      mov tos,0(stack)
640      clr tos
641      rlc tos
642      INEXT
643
644  ;; Common functions
645
646  ; ?DUP ( w -- w w | 0 )
647  ;   Dup tos if its is not zero.
648      HEAD( 4,"?DUP" )
```

```
649  QDUP:
650      tst tos
651      jnz DUPP
652      INEXT
653
654  ; ROT ( w1 w2 w3 -- w2 w3 w1 )
655  ;   Rot 3rd item to top.
656      HEAD( 3,"ROT" )
657  ROT:
658      mov.w   0(stack),temp0
659      mov.w   tos,0(stack)
660      mov.w   2(stack),tos
661      mov.w   temp0,2(stack)
662      INEXT
663
664  ; 2DROP ( w w -- )
665  ;   Discard two items on stack.
666      HEAD( 5,"2DROP" )
667  DDROP:
668      incd.w  stack
669      pops
670      INEXT
671
672  ; 2DUP ( w1 w2 -- w1 w2 w1 w2 )
673  ;   Duplicate top two items.
674      HEAD( 4,"2DUP" )
675  DDUP:
676      mov.w   @stack,temp0
677      pushs
678      decd.w  stack
679      mov.w   temp0,0(stack)
680      INEXT
681
682  ; + ( w w -- sum )
683  ;   Add top two items.
684      HEAD( 1,"+" )
685  PLUS:
686      add @stack+,tos
687      INEXT
688
689  ; D+ ( d d -- d )
690  ;   Double addition, as an example using UM+.
691      HEAD( 2,"D+" )
692  DPLUS:
693      mov.w   @stack+,temp0
694      mov.w   @stack+,temp1
695      add.w   temp0,0(stack)
696      addc    temp1,tos
697      INEXT
698
699  ; NOT   ( w -- w )
700  ;   One's complement of tos.
701      HEAD( 3,"NOT" )
702  INVER:
703      inv tos
704      INEXT
705
706  ; NEGATE ( n -- -n )
707  ;   Two's complement of tos.
708      HEAD( 6,"NEGATE" )
709  NEGAT:
710      inv tos
711      inc tos
712      INEXT
713
714  ; DNEGATE ( d -- -d )
715  ;   Two's complement of top double.
716      HEAD( 7,"DNEGATE" )
717  DNEGA:
718      inv tos
719      inv 0(stack)
720      inc 0(stack)
```

```
721       addc    #0,tos
722       INEXT
723
724   ; - ( n1 n2 -- n1-n2 )
725   ;   Subtraction.
726       HEAD( 1,"-" )
727   SUBB:
728       sub @stack+,tos
729       jmp NEGAT
730
731   ; ABS ( n -- n )
732   ;   Return the absolute value of n.
733       HEAD( 3,"ABS" )
734   ABSS:
735       tst.w   tos
736       jn  NEGAT
737       INEXT
738
739   ; = ( w w -- t )
740   ;   Return true if top two are equal.
741       HEAD( 1,"=" )
742   EQUAL:
743       xor @stack+,tos
744       jnz FALSE
745       jmp TRUE
746
747   ; U< ( u u -- t )
748   ;   Unsigned compare of top two items.
749       HEAD( 2,"U<" )
750   ULESS:
751       mov @stack+,temp0
752       cmp tos,temp0
753       subc    tos,tos
754       INEXT
755
756   ; < ( n1 n2 -- t )
757   ;   Signed compare of top two items.
758       HEAD( 1,"<" )
759   LESS:
760       cmp @stack+,tos
761       jz  FALSE
762       jge TRUE
763       jmp FALSE
764
765   ; > ( n1 n2 -- t )
766   ;   Signed compare of top two items.
767       HEAD( 1,">" )
768   GREAT:
769       cmp @stack+,tos
770       jge FALSE
771       jmp TRUE
772
773   ; MAX ( n n -- n )
774   ;   Return the greater of two top stack items.
775       HEAD( 3,"MAX" )
776   MAX:
777       cmp 0(stack),tos
778   MAX1:
779       jl  DROP
780       incd.w  stack
781       INEXT
782
783   ; MIN ( n n -- n )
784   ;   Return the smaller of top two stack items.
785       HEAD( 3,"MIN" )
786   MIN:
787       cmp tos,0(stack)
788       jmp MAX1
789
790
791   ;; Divide
792
```

```
793   .if 0  ; this code is buggy, fails for ud > 16383               ;MM-190314
794
795   ; UM/MOD ( udl udh u -- ur uq )
796   ;   Unsigned divide of a double by a single. Return mod and quotient.
797       HEAD( 6,"UM/MOD" )
798   UMMOD:
799       mov tos,temp0
800       pops
801       mov #17,temp1
802   UMMOD2:
803       cmp temp0,tos
804       jnc UMMOD3
805       sub temp0,tos
806       setc
807       jmp UMMOD4
808   UMMOD3:
809       clrc
810   UMMOD4:
811       rlc 0(stack)
812       rlc tos
813       dec temp1
814       jnz UMMOD2
815       rra tos
816       mov tos,temp0
817       mov 0(stack),tos
818       mov temp0,0(stack)
819       INEXT
820
821   .else  ; bug fix MM-190314
822
823   ;   UM/MOD  ( udl udh un -- ur uq )
824   ;       Unsigned divide of a double by a single. Return mod and quotient.
825       HEAD( 6,"UM/MOD" )
826   UMMOD:
827     mov.w @stack+,temp0
828     mov @stack,temp1
829     call #UMMOSR
830     mov temp0,0(stack)
831     mov temp1,tos
832     INEXT
833
834   UMMOSR:
835     cmp tos,temp0
836     jnc UMMO1
837     mov #-1,temp0
838     mov temp0,temp1
839     ret
840   UMMO1:
841     mov #1,temp2
842   UMMO2:
843     add temp1,temp1
844     addc temp0,temp0
845     jc UMMO3
846     cmp tos,temp0
847     jnc UMMO4
848   UMMO3:
849     sub tos,temp0
850     add #1,temp1
851   UMMO4:
852     add temp2,temp2
853     jnc UMMO2
854     ret
855
856   .endif
857
858
859   ; M/MOD ( d n -- r q )
860   ;   Signed floored divide of double by single. Return mod and quotient.
861       HEAD( 5,"M/MOD" )
862   MSMOD:
863       INEST
864       .dw DUPP,ZLESS,DUPP,TOR,QBRAN,MMOD1
```

```
865       .dw NEGAT,TOR,DNEGA,RFROM
866   MMOD1:
867       .dw TOR,DUPP,ZLESS,QBRAN,MMOD2
868       .dw RAT,PLUS
869   MMOD2:
870       .dw RFROM,UMMOD,RFROM,QBRAN,MMOD3
871       .dw SWAP,NEGAT,SWAP
872   MMOD3:
873       .dw EXIT
874
875   ; /MOD ( n n -- r q )
876   ;   Signed divide. Return mod and quotient.
877       HEAD( 4,"/MOD" )
878   SLMOD:
879       INEST
880       .dw OVER,ZLESS,SWAP,MSMOD,EXIT
881
882   ; MOD ( n n -- r )
883   ;   Signed divide. Return mod only.
884       HEAD( 3,"MOD" )
885   MODD:
886       INEST
887       .dw SLMOD,DROP,EXIT
888
889   ; / ( n n -- q )
890   ;   Signed divide. Return quotient only.
891       HEAD( 1,"/" )
892   SLASH:
893       INEST
894       .dw SLMOD,SWAP,DROP,EXIT
895
896   ;; Multiply
897
898   ; UM* ( u u -- ud )
899   ;   Unsigned multiply. Return double product.
900       HEAD( 3,"UM*" )
901   UMSTA:
902       clr temp0
903       mov #16,temp1
904   UMSTA2:
905       bit #1,0(stack)
906       jz  UMSTA3
907       add tos,temp0
908       jmp UMSTA4
909   UMSTA3:
910       clrc
911   UMSTA4:
912       rrc temp0
913       rrc 0(stack)
914       dec temp1
915       jnz UMSTA2
916       mov temp0,tos
917       INEXT
918
919   ; * ( n n -- n )
920   ;   Signed multiply. Return single product.
921       HEAD( 1,"*" )
922   STAR:
923       INEST
924       .dw UMSTA,DROP,EXIT
925
926   ; M* ( n n -- d )
927   ;   Signed multiply. Return double product.
928       HEAD( 2,"M*" )
929   MSTAR:
930       INEST
931       .dw DDUP,XORR,ZLESS,TOR
932       .dw ABSS,SWAP,ABSS,UMSTA,RFROM
933       .dw QBRAN,MSTA1
934       .dw DNEGA
935   MSTA1:
936       .dw EXIT
```

```
 937
 938  ; */MOD ( n1 n2 n3 -- r q )
 939  ;   Multiply n1 and n2, then divide by n3. Return mod and quotient.
 940      HEAD( 5,"*/MOD" )
 941  SSMOD:
 942      INEST
 943      .dw TOR,MSTAR,RFROM,MSMOD,EXIT
 944
 945  ; */ ( n1 n2 n3 -- q )
 946  ;   Multiply n1 by n2, then divide by n3. Return quotient only.
 947      HEAD( 2,"*/" )
 948  STASL:
 949      INEST
 950      .dw SSMOD,SWAP,DROP,EXIT
 951
 952  ;; Miscellaneous
 953
 954  ; 1+ ( a -- a+1 )
 955  ;   Increment.
 956      HEAD( 2,"1+" )                                          ;MM+210908
 957  ONEP:
 958      add #1,tos
 959      INEXT
 960
 961  ; 1- ( a -- a-1 )
 962  ;   Decrement
 963      HEAD( 2,"1-" )                                          ;MM+210908
 964  ONEM:
 965      sub #1,tos
 966      INEXT
 967
 968  ; 2+ ( a -- a+2 )
 969  ;   Add cell size in byte to address.
 970  ;   HEAD( 2,"2+" )
 971      HEAD( 5,"CELL+" )                                       ;MM-191024
 972  CELLP:
 973      add #2,tos
 974      INEXT
 975
 976  ; 2- ( a -- a-2 )
 977  ;   Subtract cell size in byte from address.
 978  ;   HEAD( 2,"2-" )
 979      HEAD( 5,"CELL-" )                                       ;MM-191024
 980  CELLM:
 981      sub #2,tos
 982      INEXT
 983
 984  ; 2* ( n -- 2*n )
 985  ;   Multiply tos by cell size in bytes.
 986      HEAD( 2,"2*" )
 987  CELLS:
 988      rla tos
 989      INEXT
 990
 991  ; 2/ ( n -- n/2 )
 992  ;   Divide tos by cell size in bytes.
 993      HEAD( 2,"2/" )
 994  TWOSL:
 995      rra tos
 996      INEXT
 997
 998  ; ALIGNED ( b -- a )
 999  ;   Align address to the cell boundary.
1000      HEAD( 7,"ALIGNED" )
1001  ALGND:
1002      add #1,tos
1003      bic #1,tos
1004      INEXT
1005
1006  ; >CHAR ( c -- c )
1007  ;   Filter non-printing characters.
1008  ;   HEAD( 5,">CHAR" )
```

```
1009  TCHAR:
1010      INEST
1011      .dw BLANK,MAX   ;mask msb
1012      .dw DOLIT,126,MIN   ;check for printable
1013      .dw EXIT
1014
1015  ; DEPTH ( -- n )
1016  ;   Return the depth of the data stack.
1017      HEAD( 5,"DEPTH" )
1018  DEPTH:
1019      mov stack,temp0
1020      pushs
1021      mov #SPP,tos
1022      sub temp0,tos
1023      rra tos
1024      INEXT
1025
1026  ; PICK ( ... +n -- ... w )
1027  ;   Copy the nth stack item to tos.
1028      HEAD( 4,"PICK" )
1029  PICK:
1030      rla tos
1031      add stack,tos
1032      mov @tos,tos
1033      INEXT
1034
1035  ;; Memory access
1036
1037  ; +! ( n a -- )
1038  ;   Add n to the contents at address a.
1039      HEAD( 2,"+!" )
1040  PSTOR:
1041      add @stack+,0(tos)
1042      pops
1043      INEXT
1044
1045  ; COUNT ( b -- b +n )
1046  ;   Return count byte of a string and add 1 to byte address.
1047      HEAD( 5,"COUNT" )
1048  COUNT:
1049      mov.b   @tos+,temp0
1050      pushs
1051      mov temp0,tos
1052      INEXT
1053
1054  ; CMOVE ( b1 b2 u -- )
1055  ;   Copy u bytes from b1 to b2.
1056      HEAD( 5,"CMOVE" )
1057  CMOVE:
1058      mov @stack+,temp0   ;destination
1059      mov @stack+,temp1   ;source
1060      jmp CMOVE2
1061  CMOVE1:
1062      mov.b   @temp1+,0(temp0)
1063      inc temp0
1064  CMOVE2:
1065      dec tos
1066      jn  CMOVE3  ;I need a jp.  Oh, well.
1067      jmp CMOVE1
1068  CMOVE3:
1069      JMP DROP
1070
1071  ; FILL ( b u c -- )
1072  ;   Fill u bytes of character c to area beginning at b.
1073      HEAD( 4,"FILL" )
1074  FILL:
1075      mov @stack+,temp0   ;count
1076      mov @stack+,temp1   ;destination
1077      jmp FIL2
1078  FIL1:
1079      mov.b   tos,0(temp1)
1080      inc temp1
```

```
1081  FIL2:
1082      dec temp0
1083      jn  FIL3
1084      jmp FIL1
1085  FIL3:
1086      JMP DROP
1087
1088  ; UP ( -- a )                                          MM-220121
1089  ;   Start address of the user variable area
1090      HEAD( 2,"UP" )
1091  UVP:
1092      ICONST
1093      .dw UPP+000H  ;
1094
1095  ;; User variables
1096
1097  ; 'BOOT ( -- a )
1098  ;   The application startup vector.
1099      HEAD( 5,"'BOOT" )
1100  TBOOT:
1101      ICONST
1102      .dw UPP+000H  ;
1103
1104  ; BASE ( -- a )
1105  ;   Storage of the radix base for numeric I/O.
1106      HEAD( 4,"BASE" )
1107  BASE:
1108      ICONST
1109      .dw UPP+002H
1110
1111  ; tmp ( -- a )
1112  ;   A temporary storage location used in parse and find.
1113  ;   HEAD( COMPO+3,"tmp" )
1114  TEMP:
1115      ICONST
1116      .dw UPP+004H
1117
1118  ; #TIB ( -- a )
1119  ;   Hold the character pointer while parsing input stream.
1120  ;   HEAD( 4,"#TIB" )
1121  NTIB:
1122      ICONST
1123      .dw UPP+006H
1124
1125  ; >IN ( -- a )
1126  ;   Hold the character pointer while parsing input stream.
1127  ;   HEAD( 3,">IN" )
1128  INN:
1129      ICONST
1130  ;   .dw 208H
1131      .dw UPP+008H
1132
1133  ; HLD ( -- a )
1134  ;   Hold a pointer in building a numeric output string.
1135  ;   HEAD( 3,"HLD" )
1136  HLD:
1137      ICONST
1138      .dw UPP+00AH
1139
1140  ; 'EVAL ( -- a )
1141  ;   A area to specify vocabulary search order.
1142  ;   HEAD( 5,"'EVAL" )
1143  TEVAL:
1144      ICONST
1145      .dw UPP+00CH
1146
1147  ; CONTEXT ( -- a )                                      MM-220122
1148  ;   A area to specify vocabulary search order.
1149  ; For VIS renamed to
1150  ;   HEAD( 7,"CONTEXT" )
1151  ;
1152  ; superseded by
```

```
1153
1154  ; DIC ( -- a )                                        MM-220122
1155  ;   Pointer to the last word in the dictionary.
1156      HEAD( 3,"DIC" )
1157  DICC:
1158      ICONST
1159      .dw UPP+00EH
1160
1161  ; CP ( -- a )
1162  ;   Point to the top of the code dictionary.
1163      HEAD( 2,"CP" )
1164  CP:
1165      ICONST
1166      .dw UPP+010H
1167
1168  ; DP ( -- a )
1169  ;   Point to the bottom of the free ram area.
1170      HEAD( 2,"DP" )
1171  DP:
1172      ICONST
1173      .dw UPP+012H
1174
1175  ; LAST ( -- a )
1176  ;   Point to the last name in the name dictionary.
1177      HEAD( 4,"LAST" )                                      ;MM-191024
1178  LAST:
1179      ICONST
1180      .dw UPP+014H
1181
1182  ; VIS CURRENT
1183  CURR: ; ( -- a )  MM++180712
1184    ICONST
1185    .dw UPP+016H
1186
1187  ; VIS sCONTEXT
1188  CONT: ; ( -- a )  MM++181003
1189    ICONST
1190    .dw UPP+018H
1191
1192  ; VIS tCONTEXT
1193  VOCP: ; ( -- a )  MM++180714
1194    ICONST
1195    .dw UPP+01AH
1196
1197  CSR: ; ( -- a )  MM++180726
1198    ICONST
1199    .dw UPP+01CH
1200
1201  LBB: ; ( -- a )  MM++181208
1202    ICONST
1203    .dw UPP+01EH
1204
1205
1206  ; DOLIT,0 is used so often, that it should be substituted with ZERO
1207
1208    HEAD( 1,"0" )
1209  ZERO: ; ( -- 0 )  MM--190208
1210    ICONST
1211    .dw 0
1212
1213  ; so is DOLIT,1
1214
1215  ONE: ; ( -- 1 )  MM--190208
1216    ICONST
1217    .dw 1
1218
1219
1220  ; HERE ( -- a )
1221  ;   Return the top of the code dictionary.
1222      HEAD( 4,"HERE" )
1223  HERE:
1224      INEST
```

```
1225      .dw DP,AT,EXIT
1226
1227  ; PAD ( -- a )
1228  ;   Return the address of a temporary buffer.
1229      HEAD( 3,"PAD" )
1230  PAD:
1231      INEST
1232      .dw HERE,DOLIT,80,PLUS,EXIT
1233
1234  ; TIB ( -- a )
1235  ;   Return the address of the terminal input buffer.
1236      HEAD( 3,"TIB" )
1237  TIB:
1238      ICONST
1239      .dw TIBB
1240
1241  ;; Numeric output, single precision
1242
1243  ; DIGIT ( u -- c )
1244  ;   Convert digit u to a character.
1245      HEAD( 5,"DIGIT" )
1246  DIGIT:
1247      cmp #10,tos
1248      jl  DIGIT1
1249      add #7,tos
1250  DIGIT1:
1251      add #"0",tos
1252      INEXT
1253
1254  ; EXTRACT ( n base -- n c )
1255  ;   Extract the least significant digit from n.
1256  ;   HEAD( 7,"EXTRACT" )
1257  EXTRC:
1258      INEST
1259      .dw ZERO,SWAP,UMMOD
1260      .dw SWAP,DIGIT,EXIT
1261
1262  ; <# ( -- )
1263  ;   Initiate the numeric output process.
1264      HEAD( 2,"<#" )
1265  BDIGS:
1266      INEST
1267      .dw PAD,HLD,STORE,EXIT
1268
1269  ; HOLD ( c -- )
1270  ;   Insert a character into the numeric output string.
1271      HEAD( 4,"HOLD" )
1272  HOLD:
1273      INEST
1274      .dw HLD,AT,ONEM
1275      .dw DUPP,HLD,STORE,CSTOR,EXIT
1276
1277  ; # ( u -- u )
1278  ;   Extract one digit from u and append the digit to output string.
1279      HEAD( 1,"#" )
1280  DIG:
1281      INEST
1282      .dw BASE,AT,EXTRC,HOLD,EXIT
1283
1284  ; #S ( u -- 0 )
1285  ;   Convert u until all digits are added to the output string.
1286      HEAD( 2,"#S" )
1287  DIGS:
1288      INEST
1289  DIGS1:
1290      .dw DIG,DUPP,QBRAN,DIGS2
1291      .dw BRAN,DIGS1
1292  DIGS2:
1293      .dw EXIT
1294
1295  ; SIGN ( n -- )
1296  ;   Add a minus sign to the numeric output string.
```

```
1297      HEAD( 4,"SIGN" )
1298  SIGN:
1299      INEST
1300      .dw ZLESS,QBRAN,SIGN1
1301      .dw DOLIT,"-",HOLD
1302  SIGN1:
1303      .dw EXIT
1304
1305  ; #> ( w -- b u )
1306  ;   Prepare the output string to be TYPE'd.
1307      HEAD( 2,"#>" )
1308  EDIGS:
1309      INEST
1310      .dw DROP,HLD,AT
1311      .dw PAD,OVER,SUBB,EXIT
1312
1313  ; str ( n -- b u )
1314  ;   Convert a signed integer to a numeric string.
1315  ;   HEAD( 3,"str" )
1316  STR:
1317      INEST
1318      .dw DUPP,TOR,ABSS
1319      .dw BDIGS,DIGS,RFROM
1320      .dw SIGN,EDIGS,EXIT
1321
1322  ; HEX ( -- )
1323  ;   Use radix 16 as base for numeric conversions.
1324      HEAD( 3,"HEX" )
1325  HEX:
1326      mov #16,UPP+02
1327      INEXT
1328
1329  ; DECIMAL ( -- )
1330  ;   Use radix 10 as base for numeric conversions.
1331      HEAD( 7,"DECIMAL" )
1332  DECIM:
1333      mov #10,UPP+02
1334      INEXT
1335
1336  ;; Numeric input, single precision
1337
1338  ; DIGIT? ( c base -- u t )
1339  ;   Convert a character to its numeric value. A flag indicates success.
1340  ;   HEAD( 6,"DIGIT?" )
1341  DIGTQ:
1342      mov @stack,temp0
1343      sub #"0",temp0
1344      jl  FALSE1
1345      cmp #10,temp0
1346      jl  DIGTQ1
1347      sub #7,temp0
1348  ; -- bug fix MM-180628 --
1349    cmp #10,temp0
1350    jl FALSE1
1351  ; ------------------------
1352  DIGTQ1:
1353      cmp tos,temp0
1354      mov temp0,0(stack)
1355      jl  TRUE1
1356  FALSE1:
1357      clr tos
1358      INEXT
1359  TRUE1:
1360      mov #-1,tos
1361      INEXT
1362
1363  ; NUMBER? ( a -- n T | a F )
1364  ;   Convert a number string to integer. Push a flag on tos.
1365      HEAD( 7,"number?" )
1366  NUMBQ:
1367      INEST
1368      .dw BASE,AT,TOR,ZERO,OVER,COUNT
```

```
1369        .dw OVER,CAT,DOLIT,'$',EQUAL,QBRAN,NUMQ1
1370        .dw HEX,SWAP,ONEP,SWAP,ONEM
1371    NUMQ1:
1372        .dw OVER,CAT,DOLIT,'-',EQUAL,TOR
1373        .dw SWAP,RAT,SUBB,SWAP,RAT,PLUS,QDUP
1374        .dw QBRAN,NUMQ6
1375        .dw ONEM,TOR
1376    NUMQ2:
1377        .dw DUPP,TOR,CAT,BASE,AT,DIGTQ
1378        .dw QBRAN,NUMQ4
1379        .dw SWAP,BASE,AT,STAR,PLUS,RFROM,ONEP
1380        .dw DONXT,NUMQ2
1381        .dw RAT,SWAP,DROP,QBRAN,NUMQ3
1382        .dw NEGAT
1383    NUMQ3:
1384        .dw SWAP
1385        .dw BRAN,NUMQ5
1386    NUMQ4:
1387        .dw RFROM,RFROM,DDROP,DDROP,ZERO
1388    NUMQ5:
1389        .dw DUPP
1390    NUMQ6:
1391        .dw RFROM,DDROP,RFROM,BASE,STORE,EXIT
1392
1393
1394    ;; Terminal output
1395
1396    ; BL ( -- 32 )
1397    ;   Return 32, the blank character.
1398        HEAD( 2,"BL" )
1399    BLANK:
1400        ICONST
1401        .dw 20H
1402
1403    ; SPACE ( -- )
1404    ;   Send the blank character to the output device.
1405        HEAD( 5,"SPACE" )
1406    SPACE:
1407        INEST
1408        .dw BLANK,EMIT,EXIT
1409
1410    ; SPACES ( +n -- )
1411    ;   Send n spaces to the output device.
1412        HEAD( 6,"SPACES" )
1413    SPACS:
1414        INEST
1415        .dw ZERO,MAX,TOR,BRAN,CHAR2
1416    CHAR1:
1417        .dw SPACE
1418    CHAR2:
1419        .dw DONXT,CHAR1,EXIT
1420
1421    ; TYPE ( b u -- )
1422    ;   Output u characters from b.
1423        HEAD( 4,"TYPE" )
1424    TYPEE:
1425        INEST
1426        .dw TOR,BRAN,TYPE2
1427    TYPE1:
1428        .dw DUPP,CAT,TCHAR,EMIT
1429        .dw ONEP
1430    TYPE2:
1431        .dw DONXT,TYPE1
1432        .dw DROP,EXIT
1433
1434    ; CR ( -- )
1435    ;   Output a carriage return and a line feed.
1436        HEAD( 2,"CR" )
1437    CR:
1438        INEST
1439        .dw DOLIT,CRR,EMIT
1440        .dw DOLIT,LF,EMIT,EXIT
```

```
1441
1442  ; do$ ( -- a )
1443  ;   Return the address of a compiled string.
1444  ;   HEAD( COMPO+3,"do$" )
1445  DOSTR:
1446      INEST
1447      .dw RFROM,RAT,RFROM,COUNT,PLUS
1448      .dw ALGND,TOR,SWAP,TOR,EXIT
1449
1450  ; $"| ( -- a )
1451  ;   Run time routine compiled by $". Return address of a compiled string.
1452  ;   HEAD( COMPO+3,"$\"|" )
1453  STRQP:
1454      INEST
1455      .dw DOSTR,EXIT  ;force a call to do$
1456
1457  ; ."| ( -- )
1458  ;   Run time routine of ." . Output a compiled string.
1459  ;   HEAD( COMPO+3,".\"|" )
1460  DOTQP:
1461      INEST
1462      .dw DOSTR,COUNT,TYPEE,EXIT
1463
1464  ; .R ( n +n -- )
1465  ;   Display an integer in a field of n columns, right justified.
1466      HEAD( 2,".R" )
1467  DOTR:
1468      INEST
1469      .dw TOR,STR,RFROM,OVER,SUBB
1470      .dw SPACS,TYPEE,EXIT
1471
1472  ; U.R ( u +n -- )
1473  ;   Display an unsigned integer in n column, right justified.
1474      HEAD( 3,"U.R" )
1475  UDOTR:
1476      INEST
1477      .dw TOR,BDIGS,DIGS,EDIGS
1478      .dw RFROM,OVER,SUBB,SPACS,TYPEE,EXIT
1479
1480  ; U. ( u -- )
1481  ;   Display an unsigned integer in free format.
1482      HEAD( 2,"U." )
1483  UDOT:
1484      INEST
1485      .dw BDIGS,DIGS,EDIGS,SPACE,TYPEE,EXIT
1486
1487  ; . ( w -- )
1488  ;   Display an integer in free format, preceeded by a space.
1489      HEAD( 1,"." )
1490  DOT:
1491      INEST
1492      .dw BASE,AT,DOLIT,10,XORR   ;?decimal
1493      .dw QBRAN,DOT1
1494      .dw UDOT,EXIT   ;no, display unsigned
1495  DOT1:   .dw STR,SPACE,TYPEE,EXIT    ;yes, display signed
1496
1497  ; ? ( a -- )
1498  ;   Display the contents in a memory cell.
1499      HEAD( 1,"?" )
1500  QUEST:
1501      INEST
1502      .dw AT,DOT,EXIT
1503
1504  ;; Parsing
1505
1506  ; parse ( b u c -- b u delta ; <string> )
1507  ;   Scan string delimited by c. Return found string and its offset.
1508  ;   HEAD( 5,"parse" )
1509  PARS:
1510      INEST
1511      .dw TEMP,STORE,OVER,TOR,DUPP,QBRAN,PARS8
1512      .dw ONEM,TEMP,AT,BLANK,EQUAL,QBRAN,PARS3
```

```
1513      .dw TOR
1514  PARS1:
1515      .dw BLANK,OVER,CAT  ;skip leading blanks ONLY
1516      .dw SUBB,ZLESS,INVER,QBRAN,PARS2
1517      .dw ONEP,DONXT,PARS1
1518      .dw RFROM,DROP,ZERO,DUPP,EXIT
1519  PARS2:
1520      .dw RFROM
1521  PARS3:
1522      .dw OVER,SWAP,TOR
1523  PARS4:
1524      .dw TEMP,AT,OVER,CAT,SUBB   ;scan for delimiter
1525      .dw TEMP,AT,BLANK,EQUAL,QBRAN,PARS5
1526      .dw ZLESS
1527  PARS5:
1528      .dw QBRAN,PARS6
1529      .dw ONEP,DONXT,PARS4
1530      .dw DUPP,TOR,BRAN,PARS7
1531  PARS6:
1532      .dw RFROM,DROP,DUPP,ONEP,TOR
1533  PARS7:
1534      .dw OVER,SUBB,RFROM,RFROM,SUBB,EXIT
1535  PARS8:
1536      .dw OVER,RFROM,SUBB,EXIT
1537
1538  ; PARSE ( c -- b u ; <string> )
1539  ;   Scan input stream and return counted string delimited by c.
1540      HEAD( 5,"PARSE" )                                    ;MM-191024
1541  PARSE:
1542      INEST
1543      .dw TOR,TIB,INN,AT,PLUS ;current input buffer pointer
1544      .dw NTIB,AT,INN,AT,SUBB ;remaining count
1545      .dw RFROM,PARS,INN,PSTOR,EXIT
1546
1547  ; .( ( -- )
1548  ;   Output following string up to next ) .
1549      HEAD( IMEDD+2,".(" )
1550  DOTPR:
1551      INEST
1552      .dw DOLIT,")",PARSE,TYPEE,EXIT
1553
1554  ; ( ( -- )
1555  ;   Ignore following string up to next ) . A comment.
1556      HEAD( IMEDD+1,"(" )
1557  PAREN:
1558      INEST
1559      .dw DOLIT,")",PARSE,DDROP,EXIT
1560
1561  ; \ ( -- )
1562  ;   Ignore following text till the end of line.
1563  ;  HEAD( IMEDD+1,"\" )
1564  ;   .dw link
1565  ;   .set link = $
1566  ;   .db IMEDD+1,92,
1567    HEAD( IMEDD+1,92 )
1568  BKSLA:
1569      INEST
1570      .dw NTIB,AT,INN,STORE,EXIT
1571
1572  ; CHAR ( -- c )
1573  ;   Parse next word and return its first character.
1574      HEAD( 4,"CHAR" )
1575  CHAR:
1576      INEST
1577      .dw BLANK,PARSE,DROP,CAT,EXIT
1578
1579  ; TOKEN ( -- a ; <string> )
1580  ;   Parse a word from input stream and copy it to name dictionary.
1581      HEAD( 5,"TOKEN" )
1582  TOKEN:
1583      INEST
1584      .dw BLANK,PARSE,DOLIT,31,MIN
```

```
1585  TOKEN1:
1586  .if 0  ;MM-190314
1587    ; bug: search fails if HERE returns an unaligned address
1588      .dw HERE,DDUP,CSTOR,ONEP
1589      .dw SWAP,CMOVE,HERE
1590      .dw ZERO,HERE,COUNT,PLUS,CSTOR,EXIT
1591  .else  ; move string to 'HERE ALIGNED"
1592    .dw HERE,ALGND,DUPP,TOR,DDUP,CSTOR,ONEP
1593    .dw   SWAP,CMOVE,RAT
1594    .dw   ZERO,RFROM,COUNT,PLUS,CSTOR,EXIT
1595  .endif
1596
1597
1598  ; WORD ( c -- a ; <string> )
1599  ;   Parse a word from input stream and copy it to code dictionary.
1600      HEAD( 4,"WORD" )
1601  WORDD:
1602      INEST
1603      .dw PARSE,BRAN,TOKEN1
1604
1605  ;; Dictionary search
1606
1607  ; NAME> ( na -- ca )
1608  ;   Return a code address given a name address.
1609    HEAD( 5,"NAME>" )
1610  NAMET:
1611      mov.b   @tos+,temp0
1612      and #0x1F,temp0
1613      add temp0,tos
1614      inc tos
1615      bic #1,tos
1616      INEXT
1617
1618
1619  .if CaseSensitive                                          ;MM++210420
1620
1621  ; SAME? ( a a u -- a a f \ -0+ )
1622  ;   Compare u cells in two strings. Return 0 if identical.
1623  ;   HEAD( 5,"SAME?" )
1624  SAMEQ:
1625      pushs
1626      mov 2(stack),tos
1627      mov.b   @tos,tos
1628  SAME1:
1629      mov 2(stack),temp0
1630      add tos,temp0
1631      mov.b   0(temp0),temp0
1632      mov 0(stack),temp1
1633      add tos,temp1
1634      mov.b   0(temp1),temp1
1635      sub temp1,temp0
1636      jnz SAME2
1637      dec tos
1638      jnz SAME1
1639      INEXT
1640  SAME2:
1641      jmp TRUE1
1642
1643  .else
1644
1645  ;MM++210420
1646  ;
1647  ; ?LC ( c -- c )
1648  ;   Convert upper case c to lower case, let lowercase unchanged.  No eForth word.
1649  ;   HEAD( 3,"?lc" )
1650  QLC:
1651      INEST
1652      .dw DOLIT,40H,OVER,LESS
1653      .dw QBRAN,QLC1,DUPP,DOLIT,5BH,LESS
1654      .dw QBRAN,QLC1,DOLIT,20H,PLUS
1655  QLC1:
1656      .dw EXIT
```

```
1657
1658  ;    : ?lc ( c -- c )
1659  ;     $40 over ( c @ c ) < if ( c ) dup $5B ( c c [ ) < if $20 + then then
1660  ;    ;
1661
1662
1663  ; MM++210420
1664  ; ----------------------------
1665  ; SAME? ( a a u -- a a f|0 )
1666  ;   Compare u cells in two strings. Return 0 if identical.
1667  ;   Case insensitive version !!!!!!!!!!
1668  ;   HEAD( 5,"SAME?" )
1669  SAMEQ:
1670  NSAMEQ:
1671      INEST
1672      .DW TOR,BRAN,NMSQ2
1673  NMSQ1:
1674      .dw OVER,RAT,PLUS,CAT,QLC
1675      .dw OVER,RAT,PLUS,CAT,QLC,SUBB,QDUP
1676      .dw QBRAN,NMSQ2,RFROM,DROP,EXIT,
1677  NMSQ2:
1678      .dw DONXT,NMSQ1,DOLIT,0,EXIT
1679
1680  ; : same? ( a a u -- a a f|0 )
1681  ;  for aft over r@ + c@ ?lc
1682  ;           over r@ + c@ ?lc - ?dup
1683  ;    if r> drop exit then then
1684  ;  next 0 ;
1685  ;
1686  ; ---------------------------------
1687
1688  .endif
1689
1690
1691  ; id? ( wid na -- f )  MM++180714
1692  ; Return true if the word at na is a member of the wordlist wid.
1693  ; HEAD( 3,"id?" )
1694  IDQ:
1695    INEST
1696    .dw DUPP,CAT,DOLIT,20H,ANDD
1697    .dw QBRAN,IDQ1
1698  ; tagged word, check if tag = wid
1699    .dw DOLIT,4,SUBB,AT,CLRB0
1700    .dw BRAN,IDQ2
1701  IDQ1:
1702  ; untagged word, return true if wid = wid(FORTH) = 0
1703    .dw DROP,ZERO
1704  IDQ2:
1705    .dw EQUAL,EXIT
1706
1707
1708  ; find-in-wordlist ( a wid -- xt na | a ff )  ;MM++180714
1709  WLFND:
1710    INEST
1711    .dw TOR,DICC
1712  WLFN1:  ; ( a va )
1713    .dw OVER,SWAP,FIND,DUPP,ZERO,EQUAL
1714    .dw QBRAN,WLFN2
1715    .dw SWAP,RFROM,DDROP,EXIT
1716  WLFN2: ; (a xt na )
1717    .dw RAT,OVER,IDQ,
1718    .dw QBRAN,WLFN3
1719    .dw ROT,RFROM,DDROP,EXIT
1720  WLFN3: ; ( a xt na )
1721    .dw SWAP,DROP,CELLM
1722    .dw BRAN,WLFN1
1723
1724
1725  .if CaseSensitive                                                ;MM-210420
1726
1727  ; NAME? ( a -- ca na | a F )
1728  ;   Search all context vocabularies for a string.
```

```
;   HEAD( 5,"NAME?" )
;NAMEQ:                          ;MM--180714
;   INEST                         ;MM--
;   .dw DICC,AT                   ;MM--  ( a va )
FIND: ;( a va -- ca na | a F )  ;MM++
  INEST                         ;MM++
  .dw AT                        ;MM++  ( a na )
FIND1:
    .dw DUPP,QBRAN,FIND3    ;end of dictionary
    .dw OVER,AT,OVER,AT,DOLIT,MASKK,ANDD,EQUAL
    .dw QBRAN,FIND4
    .dw SAMEQ,QBRAN,FIND2   ;match
FIND4:
    .dw CELLM,AT,BRAN,FIND1
FIND2:
    .dw SWAP,DROP,DUPP,NAMET,SWAP,EXIT
FIND3:
    .dw EXIT

.else

;MM++210420
; -------------------------------
; FIND ( a va -- xt na | a F )
; Case-insensitive find.
;   HEAD( 4,"find" )
FIND:
    INEST
    .dw SWAP
    .dw DUPP,CAT,TOR
    .dw ONEP,SWAP
  ;begin
   NFND1:
    .dw AT,DUPP
    ;if
    .dw QBRAN,NFND2
      .dw DUPP,CAT,DOLIT,1FH,ANDD,RAT,XORR
      ;if
      .dw QBRAN,NFND3
        .dw ONEP,DOLIT,-1
        .dw BRAN,NFND4
      ;else
       NFND3:
        .dw ONEP,RAT,NSAMEQ
      ;then
       NFND4:
      .dw BRAN,NFND5
    ;else
     NFND2:
      .dw RFROM,DROP,SWAP,ONEM,SWAP,EXIT
    ;then
     NFND5:
  ;while
   .dw QBRAN,NFND6
   .dw ONEM,CELLM
  ;repeat
   .dw BRAN,NFND1
   NFND6:
   .dw SWAP,DROP,RFROM,DROP,ONEM,DUPP,NAMET,SWAP
   .dw EXIT


; : find ( a va -- xt na | a F )     .s
;  swap              \ va a
;  dup c@ >r         \ va a \ get byte count
;  1+ swap           \ a+1 va
;  begin
;    @ dup           \ a+1 na na
;    if dup c@ $1F and r@ xor        \ a+1 na f
;       if 1+ -1 else 1+ r@ same? ( a+1 na+1 f ) then
;    else ( a+1 na ) r> drop swap 1 - swap ( a 0 ) exit
;    then ( a+1 na+1 f )
```

```
1801  ;  while 1- cell-  \ a' lfa
1802  ;  repeat ( a+1 na+1 ) swap drop r> drop ( na+1 ) 1- dup name> swap ;
1803
1804  ; ------------------------------
1805
1806  .endif
1807
1808
1809  ; NAME? ( a -- ca na | a F )                                    ;MM~~181124
1810  ;   Search all context vocabularies for a string.
1811  ;   .dw WORDD-6
1812  ;   HEAD( 5,"NAME?" )
1813  NAMEQ:
1814      INEST
1815    .dw VOCP,AT,GETB0,QBRAN,NAMQ1       ; ( a )
1816     ; search temporary|transient VOC search order
1817    .dw VOCP,AT,CLRB0,WLFND,QDUP,QBRAN,NAMQ2,EXIT
1818  NAMQ1:
1819     ; search permanent|static FORTH search order        ; ( a )
1820    .dw CONT,AT,QDUP,QBRAN,NAMQ3
1821    .dw WLFND,QDUP,QBRAN,NAMQ3,EXIT     ; TOSO ?
1822  NAMQ3:
1823    .dw ZERO
1824    .dw WLFND,QDUP,QBRAN,NAMQ2,EXIT     ; NOSO ?
1825  NAMQ2:
1826    .dw DOLIT,NROOT,WLFND,EXIT          ; ROOT ?
1827
1828
1829  ; ?CSR ( na|0 -- )
1830  ; If the word at na has a ctag <> NSTKY store the wid of the ctag to CSR. Other-
1831  ; wise store 0 to CSR (no context switch request).
1832  QCSR:
1833    INEST
1834    .dw DUPP,QBRAN,QCSR1
1835    ; ( na )
1836    .dw DUPP,CAT,DOLIT,XTAG,ANDD,QBRAN,QCSR2 ; ( na )
1837     ; ( na ) XTAG=true ?
1838    .dw DUPP,DOLIT,4,SUBB,AT,GETB0,QBRAN,QCSR2  ; ( na )
1839     ; ctag = NSTKY ?
1840    .dw DOLIT,6,SUBB,AT,DUPP,DOLIT,NSTKY,EQUAL,QBRAN,QCSR1 ;( ctag )
1841     ; ctag = NSTKY , no context switch ( ctag )
1842    .dw DROP,EXIT
1843  QCSR2: ;( na )
1844    .dw DUPP,SUBB ;( 0 )
1845  QCSR1: ; ( ctag|0 )
1846    .dw CLRB0
1847    .dw CSR,STORE,EXIT
1848
1849  ; ?CS ( -- )
1850  ; Do a context switch if CSR <> 0 = wid(ctag)
1851  QCS:
1852    INEST
1853    .dw CSR,AT  ; ( wid|0 )
1854    .dw QDUP,QBRAN,QCS1
1855    ; set VOCP     ( VOC search order )
1856    .dw SETB0,BRAN,QCS2
1857  QCS1:
1858    ; reset VOCP.0 ( FORTH search order )
1859    .dw VOCP,AT,CLRB0
1860  QCS2:
1861    .dw VOCP,STORE
1862    .dw EXIT
1863
1864
1865  ;; Terminal input
1866
1867  ; ^H ( bot eot cur -- bot eot cur )
1868  ;   Backup the cursor by one character.
1869  ;   HEAD( 2,"^H" )
1870  BKSP:
1871      INEST
1872      .dw TOR,OVER,RFROM,SWAP,OVER,XORR
```

```
1873      .dw QBRAN,BACK1
1874      .dw DOLIT,BKSPP,EMIT,ONEM
1875      .dw BLANK,EMIT,DOLIT,BKSPP,EMIT
1876  BACK1:
1877      .dw EXIT
1878
1879  ; TAP ( bot eot cur c -- bot eot cur )
1880  ;   Accept and echo the key stroke and bump the cursor.
1881  ;   HEAD( 3,"TAP" )
1882  TAP:
1883      INEST
1884      .dw DUPP,EMIT,OVER,CSTOR,ONEP,EXIT
1885
1886  ; kTAP ( bot eot cur c -- bot eot cur )
1887  ;   Process a key stroke, CR or backspace.
1888  ;   HEAD( 4,"kTAP" )
1889  KTAP:
1890      INEST
1891      .dw DUPP,DOLIT,CRR,XORR,QBRAN,KTAP2
1892      .dw DOLIT,BKSPP,XORR,QBRAN,KTAP1
1893      .dw BLANK,TAP,EXIT
1894  KTAP1:
1895      .dw BKSP,EXIT
1896  KTAP2:
1897      .dw DROP,SWAP,DROP,DUPP,EXIT
1898
1899  ; accept ( b u -- b u )
1900  ;   Accept characters to input buffer. Return with actual count.
1901      HEAD( 6,"ACCEPT" )
1902  ACCEP:
1903      INEST
1904      .dw OVER,PLUS,OVER
1905  ACCP1:
1906      .dw DDUP,XORR,QBRAN,ACCP4
1907      .dw KEY,DUPP,BLANK,SUBB,DOLIT,95,ULESS
1908      .dw QBRAN,ACCP2
1909      .dw TAP,BRAN,ACCP1
1910  ACCP2:
1911      .dw KTAP
1912  ACCP3:
1913      .dw BRAN,ACCP1
1914  ACCP4:
1915      .dw DROP,OVER,SUBB,EXIT
1916
1917  ; QUERY ( -- )
1918  ;   Accept input stream to terminal input buffer.
1919      HEAD( 5,"QUERY" )
1920  QUERY:
1921      INEST
1922  ;   .dw TIB,DOLIT,80,ACCEP,NTIB,STORE
1923      .dw TIB,DOLIT,80,ACCEP,NTIB,STORE,SPACE    ; MM-211102
1924      .dw DROP,ZERO,INN,STORE,EXIT
1925
1926  ;; Error handling
1927
1928  ; QUIT inits return stack. ERROR inits both stacks.
1929
1930  ; ERROR ( a -- )
1931  ;   Return address of a null string with zero count.
1932  ;   HEAD( 5,"ERROR" )
1933  ERROR:
1934      INEST
1935      .dw SPACE,COUNT,TYPEE,DOLIT
1936      .dw 3FH,EMIT,CR,SPSTO,QUIT
1937
1938
1939  ; ABORT" ( f -- )
1940  ;   Run time routine of ABORT" . Abort with a message.
1941  ;   HEAD( COMPO+6,"ABORT\"" )
1942  ABORQ:
1943      INEST
1944      .dw QBRAN,ABOR1 ;text flag
```

```
1945      .dw DOSTR,COUNT,TYPEE,SPSTO,QUIT    ;pass error string
1946  ABOR1:
1947      .dw DOSTR,DROP,EXIT
1948
1949
1950  ;; Text interpreter
1951
1952  ; $INTERPRET ( a -- )                                        ;MM~~181124
1953  ;   Interpret a word. If failed, try to convert it to an integer.
1954  ;   HEAD( 10,"$INTERPRET" )
1955  INTER:
1956      INEST
1957  ;  .dw NAMEQ,QDUP  ;?defined
1958    .dw NAMEQ,DUPP,QCSR,QDUP  ;?defined
1959      .dw QBRAN,INTE1
1960      .dw AT,DOLIT,COMPO,ANDD ;?compile only lexicon bits
1961      .dw ABORQ
1962      .db 13," compile only"
1963      .dw EXECU,QCS,EXIT  ;execute defined word  ;
1964  INTE1:
1965      .dw NUMBQ   ;convert a number
1966      .dw QBRAN,INTE2,EXIT
1967  INTE2:
1968      .dw ERROR   ;error
1969
1970
1971  ; [ ( -- )
1972  ;   Start the text interpreter.
1973      HEAD( IMEDD+1,"[" )
1974  LBRAC:
1975      INEST
1976      .dw DOLIT,INTER,TEVAL,STORE,EXIT
1977
1978  ; .OK ( -- )
1979  ;   Display 'ok' only while interpreting.
1980  ;   HEAD( 3,".OK" )
1981  DOTOK:
1982      INEST
1983      .dw DOLIT,INTER,TEVAL,AT,EQUAL
1984      .dw QBRAN,DOTO1
1985      .dw DOTQP
1986      .db 3," ok"
1987  DOTO1:  .dw CR,EXIT
1988
1989
1990  ; ?STACK ( -- )
1991  ;   Abort if the data stack underflows.
1992  ;   HEAD( 6,"?STACK" )
1993  QSTAC:
1994      INEST
1995      .dw DEPTH,ZLESS ;check only for underflow
1996      .dw ABORQ
1997      .db 10," underflow",0
1998      .dw EXIT
1999
2000  ; EVAL ( -- )
2001  ;   Interpret the input stream.
2002  ;   HEAD( 4,"EVAL" )
2003  EVAL:
2004      INEST
2005  EVAL1:
2006      .dw TOKEN,DUPP,CAT  ;?input stream empty
2007      .dw QBRAN,EVAL2
2008      .dw TEVAL,ATEXE,QSTAC   ;evaluate input, check stack
2009      .dw BRAN,EVAL1
2010  EVAL2:
2011      .dw DROP,DOTOK,EXIT ;prompt
2012
2013  ; QUIT ( -- )
2014  ;   Reset return stack pointer and start text interpreter.
2015      HEAD( 4,"QUIT" )
2016  QUIT:
```

```
2017      INEST
2018      .dw RPSTO,LBRAC ;start interpretation
2019    .dw VOCP,AT,CLRB0,VOCP,STORE  ; reset search context   MM++181124
2020  QUIT1:
2021      .dw QUERY,EVAL  ;get input
2022      .dw BRAN,QUIT1  ;continue till error
2023
2024  ;; Compiler utilities
2025
2026  ; ALLOT ( n -- )
2027  ;   Allocate n bytes to the RAM dictionary.
2028      HEAD( 5,"ALLOT" )
2029  ALLOT:
2030      INEST
2031      .dw DP,PSTOR,EXIT   ;adjust code pointer
2032
2033  ; IALLOT ( n -- )
2034  ;   Allocate n bytes to the code dictionary.
2035      HEAD( 6,"iALLOT" )
2036  IALLOT:
2037      INEST
2038      .dw CP,PSTOR,EXIT   ;adjust code pointer
2039
2040
2041  ; I! ( n a -- )
2042  ;   Store n to address a in code dictionary.
2043      HEAD( 2,"i!" )
2044  ISTORE:
2045  .if MCU ; G2553
2046      mov #FWKEY,&FCTL3 ; Clear LOCK
2047      mov #FWKEY+WRT,&FCTL1 ; Enable write
2048  ;   call    #STORE
2049      mov.w   @stack+,0(tos)
2050      pops
2051      mov #FWKEY,&FCTL1 ; Done. Clear WRT
2052      mov #FWKEY+LOCK,&FCTL3 ; Set LOCK
2053  .else  ; FR5969 == FR5739
2054      mov.w   @stack+,0(tos)
2055      pops
2056  .endif
2057      INEXT
2058
2059  ;------------------------------------ ;MM++180707
2060  ; ?I! ( w a -- )
2061  ;   Store w to address a in code dictionary.
2062  ; Abort if user dictionary space is full.
2063  QISTOR:
2064    INEST
2065    .dw QFLASH,ISTORE,EXIT
2066  ; ------------------------------------
2067
2068  .if MCU ; G2553
2069
2070  ; IERASE ( a -- )
2071  ;   Erase a segment at address a.
2072  ;   HEAD( 5,"ERASE" )   ;MM--180630
2073      HEAD( 6,"iERASE" )  ;MM++
2074  IERASE:
2075  .if MCU ; G2553
2076      mov #FWKEY,&FCTL3 ; Clear LOCK
2077      mov #FWKEY+ERASE,&FCTL1 ; Enable erase
2078      clr 0(tos)
2079      mov #FWKEY+LOCK,&FCTL3 ; Set LOCK
2080  .endif
2081      pops
2082      INEXT
2083
2084  .endif
2085
2086
2087  .if MCU ; G2553   only show the head for G2553      MM-220121
2088
```

```
2089  ; IWRITE ( src dest n -- )
2090  ;   Copy n bytes from src to dest.  Dest is in flash memory.
2091  ;   HEAD( 5,"WRITE" )    ;MM--
2092      HEAD( 6,"iWRITE"  )  ;MM++
2093  .endif
2094
2095  IWRITE:
2096      INEST
2097      .dw TWOSL,TOR
2098  IWRITE1:
2099  ;   .dw OVER,AT,OVER,ISTORE    ;MM--180707
2100      .dw OVER,AT,OVER,QISTOR    ;MM++
2101      .dw CELLP,SWAP,CELLP,SWAP
2102      .dw DONXT,IWRITE1
2103      .dw DDROP,EXIT
2104
2105
2106  ; , ( w -- )
2107  ;   Compile an integer into the code dictionary.
2108      HEAD( 1,"," )
2109  COMMA:
2110      INEST
2111  ;   .dw CP,AT,DUPP,CELLP    ;cell boundary                      ;MM--180707
2112      .dw CP,AT,QFLASH,DUPP,CELLP ;cell boundary + flash test  ;MM++
2113      .dw CP,STORE,ISTORE,EXIT
2114
2115  ; call, ( w -- )
2116  ;   Compile a call instruction into the code dictionary.
2117  ;   HEAD( 5,"call," )
2118  CALLC:
2119      INEST
2120      .dw DOLIT,CALLL,COMMA
2121      .dw COMMA,EXIT
2122
2123  ; [COMPILE] ( -- ; <string> )
2124  ;   Compile the next immediate word into code dictionary.
2125  ;   HEAD( IMEDD+9,"[COMPILE]" )
2126  BCOMP:
2127      INEST
2128      .dw TICK,COMMA,EXIT
2129
2130  ; COMPILE ( -- )
2131  ;   Compile the next address in colon list to code dictionary.
2132  ;   HEAD( COMPO+7,"COMPILE" )
2133  COMPI:
2134      INEST
2135      .dw RFROM,DUPP,AT,COMMA ;compile address
2136      .dw CELLP,TOR,EXIT  ;adjust return address
2137
2138  ; LITERAL ( w -- )
2139  ;   Compile tos to code dictionary as an integer literal.
2140      HEAD( IMEDD+7,"LITERAL" )
2141  LITER:
2142      INEST
2143      .dw COMPI,DOLIT,COMMA,EXIT
2144
2145  ; $," ( -- )
2146  ;   Compile a literal string up to next " .
2147  ;   HEAD( 3,"$,\"" )
2148  STRCQ:
2149      INEST
2150  ;   .dw     DOLIT,"""      ;MM-180624 assembles to |DOLIT|0000H|
2151      .dw     DOLIT,0022H     ;              should be |DOLIT|ASCII(")|
2152      .dw WORDD    ;move string to code dictionary
2153    .dw ZERO  ;MM++ ( a 0 ) no count byte flags
2154      .dw STRCQ1,EXIT
2155
2156  ; STRCQ1:  ;( a -- ) called from $," or $,,n           ;MM--180713
2157    STRCQ1:  ;( a f -- )                                 ;MM++
2158             ;( a 0 -- ) when called from $,"
2159             ;( a COMPO|IMEDD|XTAG -- )when called from $,,n
2160      INEST
```

```
2161  ;   .dw DUPP,CAT,TWOSL  ;MM-- ;calculate aligned end of string
2162    .dw OVER,CAT,TWOSL  ;MM++ ( a f cnt )
2163      .dw TOR
2164    .dw OVER,PSTOR      ;MM++ ( store f to count byte at a )
2165  STRCQ2:
2166      .dw DUPP,AT,COMMA,CELLP
2167      .dw DONXT,STRCQ2
2168      .dw DROP,EXIT
2169
2170  ;; Structures
2171
2172  ; FOR ( -- a )
2173  ;   Start a FOR-NEXT loop structure in a colon definition.
2174      HEAD( IMEDD+3,"FOR" )
2175  FOR:
2176      INEST
2177      .dw COMPI,TOR,BEGIN,EXIT
2178
2179  ; BEGIN ( -- a )
2180  ;   Start an infinite or indefinite loop structure.
2181      HEAD( IMEDD+5,"BEGIN" )
2182  BEGIN:
2183      INEST
2184      .dw CP,AT,EXIT
2185
2186  ; NEXT ( a -- )
2187  ;   Terminate a FOR-NEXT loop structure.
2188      HEAD( IMEDD+4,"NEXT" )
2189  NEXT:
2190      INEST
2191      .dw COMPI,DONXT,COMMA,EXIT
2192
2193  ; UNTIL ( a -- )
2194  ;   Terminate a BEGIN-UNTIL indefinite loop structure.
2195      HEAD( IMEDD+5,"UNTIL" )
2196  UNTIL:
2197      INEST
2198      .dw COMPI,QBRAN,COMMA,EXIT
2199
2200  ; AGAIN ( a -- )
2201  ;   Terminate a BEGIN-AGAIN infinite loop structure.
2202      HEAD( IMEDD+5,"AGAIN" )
2203  AGAIN:
2204      INEST
2205      .dw COMPI,BRAN,COMMA,EXIT
2206
2207  ; IF ( -- A )
2208  ;   Begin a conditional branch structure.
2209      HEAD( IMEDD+2,"IF" )
2210  IFF:
2211      INEST
2212      .dw COMPI,QBRAN,BEGIN
2213      .dw DOLIT,2,IALLOT,EXIT
2214
2215  ; AHEAD ( -- A )
2216  ;   Compile a forward branch instruction.
2217  ;   HEAD( IMEDD+5,"AHEAD" )
2218  AHEAD:
2219      INEST
2220      .dw COMPI,BRAN,BEGIN
2221      .dw DOLIT,2,IALLOT,EXIT
2222
2223  ; REPEAT ( A a -- )
2224  ;   Terminate a BEGIN-WHILE-REPEAT indefinite loop.
2225      HEAD( IMEDD+6,"REPEAT" )
2226  REPEA:
2227      INEST
2228      .dw AGAIN,BEGIN,SWAP,ISTORE,EXIT
2229
2230  ; THEN ( A -- )
2231  ;   Terminate a conditional branch structure.
2232      HEAD( IMEDD+4,"THEN" )
```

```
2233  THENN:
2234      INEST
2235      .dw BEGIN,SWAP,QISTOR,EXIT
2236
2237  ; AFT ( a -- a A )
2238  ;   Jump to THEN in a FOR-AFT-THEN-NEXT loop the first time through.
2239      HEAD( IMEDD+3,"AFT" )
2240  AFT:
2241      INEST
2242      .dw DROP,AHEAD,BEGIN,SWAP,EXIT
2243
2244  ; ELSE ( A -- A )
2245  ;   Start the false clause in an IF-ELSE-THEN structure.
2246      HEAD( IMEDD+4,"ELSE" )
2247  ELSEE:
2248      INEST
2249      .dw AHEAD,SWAP,THENN,EXIT
2250
2251  ; WHILE ( a -- A a )
2252  ;   Conditional branch out of a BEGIN-WHILE-REPEAT loop.
2253      HEAD( IMEDD+5,"WHILE" )
2254  WHILE:
2255      INEST
2256      .dw IFF,SWAP,EXIT
2257
2258  ; ABORT" ( -- ; <string> )
2259  ;   Conditional abort with an error message.
2260    .dw link
2261    .set link = $
2262  ; .db IMEDD+6,"ABORT",34,0           MM-190717
2263    .db IMEDD+COMPO+6,"ABORT",34,0
2264  ABRTQ:
2265      INEST
2266      .dw COMPI,ABORQ,STRCQ,EXIT
2267
2268  ; $" ( -- ; <string> )
2269  ;   Compile an inline string literal.
2270    .dw link
2271    .set link = $
2272    .db IMEDD+2,36,34,0
2273  STRQ:
2274      INEST
2275      .dw COMPI,STRQP,STRCQ,EXIT
2276
2277  ; ." ( -- ; <string> )
2278  ;   Compile an inline string literal to be typed out at run time.
2279    .dw link
2280    .set link = $
2281    .db IMEDD+2,".",34,0
2282  DOTQ:
2283      INEST
2284      .dw COMPI,DOTQP,STRCQ,EXIT
2285
2286  ;; Colon compiler
2287
2288  ; ?UNIQUE ( a -- a )
2289  ;   Display a warning message if the word already exists.
2290  ;   HEAD( 7,"?UNIQUE" )
2291  UNIQU:
2292      INEST
2293  ;   .dw DUPP,NAMEQ           ;?name exists             ;MM--180714
2294    .dw DUPP,CURR,AT,CLRB0,WLFND  ;?name exists        ;MM++
2295      .dw QBRAN,UNIQ1          ;redefinitions are OK
2296      .dw DOTQP
2297      .db 7," reDef "          ;but warn the user
2298      .dw OVER,COUNT,TYPEE     ;just in case its not planned
2299  UNIQ1:
2300      .dw DROP,EXIT
2301
2302
2303  ; $,n ( a -- )
2304  ;   Build a new dictionary name using the string at a.
```

```
2305      HEAD( 3,"$,n" )                                          ;MM-192024
2306    SNAME:
2307      INEST
2308    ; .dw ZERO,SWAP ; ( 0 a )  ;MM++180714
2309      .dw LBB,AT,DOLIT,0E0H,ANDD,SWAP     ;MM-181208
2310      .dw BRAN,FNAM1              ;MM++
2311    ; $,,n ( f a )                ;MM++
2312    ; Build a new dictionary name using string a and flags f.
2313    FNAME:
2314      INEST                       ;MM++
2315    FNAM1:                        ;MM++
2316      .dw ZERO,LBB,STORE       ;MM++181208   reset LBB
2317      .dw CP,AT,ALGND,CP,STORE    ; align CP    MM++181004
2318        .dw DUPP,CAT    ;?null input
2319        .dw QBRAN,SNAM1
2320        .dw UNIQU   ;?redefinition
2321       ; ( f a ) ? compile a ctag and a wtag  ;MM++180726
2322      .dw SWAP,CURR,AT,DUPP,GETB0,QBRAN,SNAM2  ; ( a f wtag )
2323       ; compile ctag
2324      .dw VOCP,AT,CLRB0,COMMA   ; ( a f wtag )
2325    SNAM2:  ; ( a f wtag )
2326      .dw DUPP,QBRAN,SNAM3
2327      ; ( a f wtag ) compile wtag, reset CURR.0
2328      .dw DUPP,COMMA,CLRB0,CURR,STORE  ; ( a f )
2329      .dw DOLIT,XTAG,ORR               ; ( a f ) set XTAG bit in f
2330      .dw BRAN,SNAM0
2331    SNAM3: ; ( a f 0|1 )
2332      .dw DROP
2333    SNAM0: ; ( a f )
2334        .dw LAST,AT,COMMA       ;save na for vocabulary link
2335        .dw CP,AT,LAST,STORE
2336        .dw STRCQ1,EXIT       ;fill name field
2337    SNAM1:
2338        .dw STRQP
2339        .db 5," name"   ;null input
2340        .dw ERROR
2341
2342
2343    ; $COMPILE ( a -- )
2344    ;   Compile next word to code dictionary as a token or literal.
2345    ;   HEAD( 8,"$COMPILE" )
2346    SCOMP:
2347        INEST
2348    ;   .dw NAMEQ,QDUP  ;?defined    ;MM--181124
2349      .dw NAMEQ,DUPP,QCSR,QDUP      ;MM++
2350        .dw QBRAN,SCOM2
2351        .dw AT,DOLIT,IMEDD,ANDD   ;?immediate
2352        .dw QBRAN,SCOM1
2353    ;   .dw EXECU,EXIT            ;its immediate, execute          ;MM--181124
2354        .dw EXECU,QCS,EXIT        ;its immediate, execute          ;MM++
2355    SCOM1:
2356    ;   .dw COMMA,EXIT            ;its not immediate, compile      ;MM--
2357      .dw   COMMA,QCS,EXIT        ;its not immediate, compile      ;MM++
2358    SCOM2:
2359        .dw NUMBQ                 ;try to convert to number
2360        .dw QBRAN,SCOM3
2361        .dw LITER,EXIT            ;compile number as integer
2362    SCOM3:  .dw ERROR           ;error
2363
2364
2365    ; OVERT ( -- )
2366    ;   Link a new word into the current vocabulary.
2367        HEAD( 5,"OVERT" )                                          ;MM-191024
2368    OVERT:
2369        INEST
2370        .dw LAST,AT,DICC,STORE,EXIT
2371
2372    ; ; ( -- )
2373    ;   Terminate a colon definition.
2374        HEAD( IMEDD+COMPO+1,";" )
2375    SEMIS:
2376        INEST
```

```
2377      .dw COMPI,EXIT,LBRAC,OVERT,EXIT
2378
2379  ; ] ( -- )
2380  ;   Start compiling the words in the input stream.
2381      HEAD( 1,"]" )
2382  RBRAC:
2383      INEST
2384      .dw DOLIT,SCOMP,TEVAL,STORE,EXIT
2385
2386  ; : ( -- ; <string> )
2387  ;   Start a new colon definition using next word as its name.
2388      HEAD( 1,":" )
2389  COLON:
2390      INEST
2391      .dw TOKEN,SNAME
2392  COLO1:
2393  ;  .dw DOLIT,DOLST,CALLC,RBRAC,EXIT    ; MM-191024
2394     .dw DOLIT,callR15,COMMA,RBRAC,EXIT
2395
2396
2397  ; :nn ( -- xt )                                          ;MM++181004
2398  ; Start a headerless colon definition.
2399    HEAD( 3,":nn" )
2400  CONN:
2401    INEST
2402    .dw CP,AT,BRAN,COLO1
2403
2404  ;; Defining words
2405
2406  ; HEADER ( -- ; <string> )
2407  ;   Compile a new array entry without allocating code space.
2408  ;   HEAD( 6,"HEADER" )
2409  HEADER:
2410      INEST
2411      .dw TOKEN,SNAME,OVERT
2412      .dw DOLIT,DOCON,CALLC,EXIT
2413
2414
2415  ; CREATE ( -- ; <string> )
2416  ;   Compile a new array entry without allocating code space.
2417      HEAD( 6,"CREATE" )
2418  CREAT:
2419      INEST
2420      .dw HEADER,DP,AT,COMMA,EXIT
2421
2422  ; CONSTANT ( n -- ; <string> )
2423  ;   Compile a new constant.
2424      HEAD( 8,"CONSTANT" )
2425  CONST:
2426      INEST
2427      .dw HEADER,COMMA,EXIT
2428
2429  ; VARIABLE ( -- ; <string> )
2430  ;   Compile a new variable initialized to 0.
2431      HEAD( 8,"VARIABLE" )
2432  VARIA:
2433      INEST
2434      .dw CREAT,DOLIT,2,ALLOT,EXIT
2435
2436  ;; Tools
2437
2438  ; ' ( -- ca )                                              ;MM~~181207
2439  ;   Search context vocabulary for the next word in input stream.
2440    WTAG( NROOT )
2441      HEAD( XTAG+1,"'" )
2442  TICK:
2443      INEST
2444      .dw TOKEN,NAMEQ ; defined ?
2445      .dw QBRAN,TICK1
2446      .dw EXIT        ;yes, push code address
2447  TICK1:
2448      .dw ERROR       ;no, error
```

```
2449
2450  .if 0
2451  ; DUMP( a u -- )
2452  ;   Dump u bytes from a, in a formatted manner.
2453      HEAD( 4,"DUMP" )
2454  DUMP:
2455      INEST
2456      .dw DOLIT,7,TOR ;start count down loop
2457  DUMP1:
2458      .dw CR,DUPP,DOLIT,5,UDOTR
2459      .dw DOLIT,15,TOR
2460  DUMP2:
2461      .dw COUNT,DOLIT,3,UDOTR
2462      .dw DONXT,DUMP2 ;loop till done
2463      .dw SPACE,DUPP,DOLIT,16,SUBB
2464      .dw DOLIT,16,TYPEE  ;display printable characters
2465      .dw DONXT,DUMP1 ;loop till done
2466      .dw DROP,EXIT
2467  .endif
2468
2469  ; .S ( ... -- ... )
2470  ;   Display the contents of the data stack.
2471    WTAG( NROOT )
2472      HEAD( XTAG+2,".S" )
2473  DOTS:
2474      INEST
2475  ;  .dw DOTS,EXIT
2476      .dw CR,DEPTH    ;stack depth
2477      .dw TOR ;start count down loop
2478      .dw BRAN,DOTS2  ;skip first pass
2479  DOTS1:
2480      .dw RAT,PICK,DOT    ;index stack, display contents
2481  DOTS2:
2482      .dw DONXT,DOTS1 ;loop till done
2483      .dw DOTQP
2484      .db 4," <sp",0
2485      .dw EXIT
2486
2487  ; >NAME ( ca -- na | F )
2488  ;   Convert code address to a name address.
2489  ;   HEAD( 5,">NAME" )
2490  TNAME:
2491      INEST
2492      .dw TOR,DICC,AT ;vocabulary link
2493  TNAM1:
2494      .dw DUPP,QBRAN,TNAM2
2495      .dw DUPP,NAMET,RAT,XORR ;compare
2496      .dw QBRAN,TNAM2
2497      .dw CELLM   ;continue with next word
2498      .dw AT,BRAN,TNAM1
2499  TNAM2:
2500      .dw RFROM,DROP,EXIT
2501
2502
2503  ; .ID ( na -- )
2504  ;   Display the name at address.
2505    WTAG( NROOT )
2506  ; HEAD( 3,".ID" )
2507    HEAD( XTAG+3,".ID" )
2508  DOTID:
2509      INEST
2510      .dw COUNT,DOLIT,01FH,ANDD   ;mask lexicon bits
2511      .dw TYPEE,EXIT
2512
2513
2514  CLRB0: ; ( w1 -- w2 ) Clear bit 0 of w1     ;MM++180725
2515    bic #1,tos
2516    INEXT
2517
2518  SETB0: ; ( w1 -- w2 ) set bit 0 of w1       ;MM++180725
2519    bis #1,tos
2520    INEXT
```

```
2521
2522  GETB0: ; ( w -- 0|1 )   return bit 0 of w     ;MM++190208
2523    and #1,tos
2524    INEXT
2525
2526
2527
2528  ; Return the address of the current context pointer.
2529  QCON: ; ( -- VOCP|CONT )                    ;MM++181207
2530    INEST
2531    .dw VOCP,AT,GETB0,QBRAN,QCON1,VOCP,EXIT
2532  QCON1:
2533    .dw CONT,EXIT
2534
2535  ;   Display the name at address na if it's a member of the actual context.
2536  QID: ; ( na -- )                            ;MM++180714~~181207
2537    INEST
2538    .dw QCON,AT,CLRB0,OVER,IDQ
2539    .dw QBRAN,QID1
2540  ; .dw SPACE,DOTID,BRAN,QID2     MM--210904
2541    .dw DOTID,SPACE,SPACE,BRAN,QID2
2542  QID1:
2543    .dw DROP
2544  QID2:
2545    .dw EXIT
2546
2547
2548  ; WORDS ( -- )      ;MM-181118  moved to the ROOT VOC
2549  ; Display the names of the top of the actual search order.   MM~~181207
2550    WTAG( NROOT )
2551      HEAD( XTAG+5,"WORDS" )
2552  WORDS:
2553      INEST
2554    .dw CR,DICC,AT     ; the top of the dictionaries wordlisT
2555  WORS1:
2556      .dw QDUP    ;?at end of list
2557      .dw QBRAN,WORS2
2558      .dw DUPP,QID          ;display a name
2559  ;   .dw CELLM,AT,BRAN,WORS1                           ; MM-211208
2560      .dw CELLM,AT                                      ;
2561  ;   ?key dup if swap drop then not and ( 16 bytes )   ;
2562      .dw QKEY,DUPP,QBRAN,WORS3,SWAP,DROP               ;
2563  WORS3:
2564      .dw INVER,ANDD                                    ;
2565      .dw BRAN,WORS1                                    ;
2566  WORS2:
2567      .dw EXIT
2568
2569
2570  ;; Cold boot
2571
2572  ; HI ( -- )
2573  ;   Display the sign-on message of eForth.
2574      HEAD( 2,"hi" )
2575  HI:
2576      INEST
2577      .dw CR,DOTQP
2578  .if MCU ; G2553
2579      .db 23,"430eForth-g2553-43n7vis"     ;model
2580  .else   ; fr5739 ;mk
2581      .db 24,"430eForth-fr5739-43n7vis",0  ;model
2582  .endif
2583      .dw EXIT
2584
2585
2586  ; APP! ( a -- ) Turnkey
2587  ;   HEX : APP! 200 ! 1000 IERASE 200 1000 20 IWRITE ;
2588      HEAD( 4,"app!" )
2589  APPST:
2590      INEST
2591  .if MCU ; G2553
2592  ;   .dw TBOOT,STORE,DOLIT,0x1000,IERASE     ;MM-220119
```

```
2593      .dw TBOOT,STORE,DOLIT,INFOD,IERASE
2594  ;   .dw TBOOT,DOLIT,0x1000,DOLIT,0x20       ;MM-220119
2595      .dw TBOOT,DOLIT,INFOD,DOLIT,0x20,IWRITE
2596  .else  ; FR5969 == FR5739
2597      .dw TBOOT,STORE
2598      .dw TBOOT,DOLIT,INFOD,DOLIT,0x20,IWRITE
2599  .endif
2600      .dw EXIT
2601
2602
2603  ;; Flash tools                              ;MM-180629
2604
2605  EDM equ 0FFC0H-2  ; top of users dictionary space in the flash memory
2606
2607  .if MCU ; G2553
2608
2609  ; fscan ( -- )  ;MM++180629
2610  ; Scan the Flash memory from EDM downwards and set CP to the next free cell
2611  ; above the last used one.
2612  ; HEAD( 5,"fscan" )
2613  FSCAN:
2614    INEST
2615     .dw DOLIT,EDM
2616  FSCN1:
2617     .dw DOLIT,2,SUBB,DUPP,AT,DOLIT,EM,SUBB
2618     .dw QBRAN,FSCN1
2619     .dw DOLIT,2,PLUS,CP,AT,ALGND,MAX,CP,STORE,EXIT
2620
2621  .endif
2622
2623  ; ?flash ( a -- a )  ;MM++180707
2624  ; Abort if user dictionary is full ( a > EDM )
2625  ; HEAD( COMPO+6,"?flash" )
2626  QFLASH:
2627  ; EDM OVER U< ABORT"  Flash?"
2628    INEST
2629    .dw DOLIT,EDM,OVER,ULESS,ABORQ,
2630    .db 7," Flash?"
2631    .dw EXIT
2632
2633  ;; VOCs and ITEMs  ( VIS )                              ;MM~~190105
2634
2635  ; doVP ( -- ) runtime for VOCS
2636  ; HEAD( COMPO+4,"doVP" )
2637  DOVP:
2638    INEST
2639  ; .dw RFROM,AT,CSR,STORE,QCS,EXIT     ;MM~~190208
2640    .dw RFROM,AT                        ;
2641  DOVP1:                                ;
2642    .dw CSR,STORE,QCS,EXIT              ;
2643
2644
2645  ; ROOT ( -- )                                          ;MM~~191004
2646  ; The root VOCabulary for the default and the transient VOCabulary search order.
2647    HEAD( IMEDD+4,"root" )
2648    .set NROOT = link
2649  ROOT:
2650  ;  INEST
2651  ;  .dw DOVP,NROOT
2652    IVOC
2653    .dw NROOT
2654
2655
2656  ; FORTH ( -- )                                          ;MM~~181120
2657  ; The FORTH VOCabulary. The default search order is FORTH ROOT.
2658    HEAD( IMEDD+5,"forth" )
2659    .set NFORTH = link
2660  FORTH:
2661  ;  INEST                  ;MM~~191024
2662  ;  .dw DOVP,1
2663    IVOC
2664    .dw 1
```

```
2665
2666
2667  ; DEFINITIONS ( -- )                                          ;MM~~181207
2668  ; Make the actual VOC the current compilation context.
2669    WTAG(NROOT)
2670    HEAD( IMEDD+XTAG+11,"DEFINITIONS" )
2671  DEFS:
2672    INEST
2673    .dw QCON
2674  DEFS1:
2675  ;  .dw AT,CLRB0,CURR,STORE,EXIT                      ;MM~~190129
2676    .dw AT,CLRB0,CURR,STORE,oDDI,EXIT                ;
2677
2678
2679  ; VOC ( "name" -- )                                  ;MM~~190105
2680  ; Create a vocabulary prefix.
2681    HEAD( 3,"voc" )
2682  VOC:
2683    INEST
2684  ;   .dw DOLIT,IMEDD,TOKEN,FNAME,DOLIT,DOLST,CALLC,OVERT   ; MM-191024
2685      .dw DOLIT,IMEDD,TOKEN,FNAME,DOLIT,callR15,COMMA,OVERT
2686      .dw DOLIT,DOVP,COMMA,LAST,AT,COMMA,EXIT
2687
2688
2689  ; item ( -- )                                      ;MM~~190105
2690  ; Make the next created word a context switching one that restores the VOC
2691  ; context it was created in.
2692    WTAG( NROOT )
2693    HEAD( XTAG+4,"item" )
2694  ITEM:
2695    INEST
2696    .dw CURR,AT,SETB0,CURR,STORE,EXIT
2697
2698
2699  ; .. ( -- ) ; IMMEDIATE                              ;MM~~190105
2700  ; Switch back from a VOC context to the default search order.
2701    WTAG( NROOT )
2702    HEAD( IMEDD+XTAG+2,".." )
2703  oDDI:
2704    INEST
2705  ; .dw ZERO,CSR,STORE,QCS,EXIT         ;M~~190208
2706    .dw ZERO,BRAN,DOVP1
2707
2708
2709  ; @voc ( -- ) ; IMMEDIATE                            ;MM~~191024
2710  ; Use CURRENT as vocabulary prefix.
2711    WTAG( NROOT )
2712    HEAD( IMEDD+XTAG+4,"@voc" )
2713  ATVOC:
2714    INEST
2715    .dw CURR,AT,CLRB0,BRAN,DOVP1
2716  ;  .dw ZERO,BRAN,DOVP1
2717
2718
2719  ; FIRST ( -- )                                        ;MM~~181207
2720  ; Overwrite the top of the Forth search order with the wid of current VOC.
2721    WTAG( NROOT )
2722    HEAD( IMEDD+XTAG+5,"first" )
2723  FIRST:
2724    INEST
2725    .dw QCON,AT,CLRB0,CONT,STORE,oDDI,EXIT
2726
2727
2728  ; STICKY ( – )                                      ;MM++180826  181003
2729  ; Make the next created word a context switching ITEM that restores the VOC
2730  ; context it was found in.
2731  ; NSTKY:
2732    HEAD( 6,"sticky" )
2733    .set NSTKY = link
2734  STKY:
2735    INEST
2736    .dw DOLIT,NSTKY,VOCP,STORE,ITEM,EXIT
```

```
2737
2738
2739  ; ONLY ( -- )
2740  ; Reset all context related pointers.   ;MM++190208
2741    WTAG( NROOT )
2742    HEAD( IMEDD+XTAG+4,"ONLY" )
2743  ONLY:
2744    INEST
2745  ; .dw CURR,DOLIT,0AH,ZERO,FILL,EXIT     ;MM~~190211
2746    .dw CONT,DOLIT,08H,ZERO,FILL,EXIT     ;
2747
2748
2749  ; NOAPP? ( -- f )                                          :MM++210417
2750  ; Return true if P1.3 is low
2751  NOAPP:
2752  .if 0 ; G2553
2753      bic.b   #008h,&P1DIR    ; setup P1.3 as input with pull-up resistor
2754      bis.b   #008h,&P1OUT    ;
2755      bis.b   #008h,&P1REN    ;
2756      pushs
2757      mov.w #-1,tos
2758      bit.b #8,&P1IN
2759      jz  NOAPP1
2760      add #1,tos
2761  NOAPP1:
2762      bic.b   #008h,&P1REN    ; P1.3 floating input
2763  .else  ; FR5969
2764      pushs
2765      mov.w #0,tos
2766  .endif
2767      INEXT
2768
2769  ;; init is now placed ahead of COLD             ;MM-220117  FR5969
2770
2771  init:                ;mk Put it closer to cold : jmp was out of range.
2772      mov #RPP,SP ; set up stack
2773      mov #SPP,stack
2774      clr tos
2775      mov #DOLST,R15    ; MM-191024
2776  .if MCU ; G2553
2777      mov.w   #WDTPW+WDTHOLD,&WDTCTL  ; Stop watchdog timer
2778      bis.b   #041h,&P1DIR    ; P1.0/6 output
2779  .else  ; FR5969 == FR5739
2780      mov.w   #WDTPW|WDTHOLD,&WDTCTL ; disable watchdog
2781  .endif
2782      jmp COLD
2783
2784  ; COLD  ( -- )
2785  ;   The hilevel cold start sequence.
2786      HEAD( 4,"COLD" )
2787  COLD:
2788      INEST
2789      .dw STOIO
2790      .dw DOLIT,UZERO,DOLIT,UPP
2791      .dw DOLIT,ULAST-UZERO,CMOVE    ;initialize user area
2792  ;   .dw CURR,DOLIT,0AH,DOLIT,0,FILL  ; init contexts and CURRENT ;MM~~181207
2793      .dw ONLY,DEFS
2794
2795  .if MCU ; G2553
2796  ;
2797  ; MM++210417
2798  ; If P1.3 is low at cold start, reset 'boot to hi,so that an app is not started.
2799  ; With save or reset you can make this change permanent. Otherwise the app will
2800  ; be started again at the next cold start, if P1.3 is no longer low.
2801  ;
2802      .dw NOAPP,QBRAN,COLD1,DOLIT,HI,TBOOT,STORE
2803  COLD1:
2804  ;
2805  .endif
2806      .dw TBOOT,ATEXE ;application boot
2807  .if MCU ; G2553
2808      .dw FSCAN,CR                                              ;MM++180629
```

```
2809  .else   ; FR5969 == FR5739
2810      .dw CR
2811  .endif
2812      .dw QUIT    ;start interpretation
2813
2814
2815  ; MCU-ID ( -- )                                                 ;MM++220123
2816  MCUID:
2817  .if MCU ; G2553
2818      HEAD(IMEDD+7,"-G2553-" )
2819  .else   ; FR5739
2820      HEAD(IMEDD+8,"-FR5739-" )
2821  .endif
2822      INEXT
2823
2824  CTOP:
2825
2826  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
2827
2828  ;; COLD start moves the following user variable init values from NVM to RAM.
2829
2830  .if MCU ; G2553
2831  ;   .sect   ".infoD"  ; CCS: .sect ; naken: .org   ;mk
2832  ;   .org 01000H ;mk
2833      .org INFOD  ;MM-220119
2834  ;   1000-10FF = 256B information memory   ;mk
2835  ;   INFO+000h (INFOD):  RAM save area, user variables ;mk
2836  ;   INFO+040h (INFOC):  RAM save area ??? ;mk
2837  ;   INFO+080h (INFOB):  user interrupt vectors ??? ;mk
2838  ;   INFO+0C0h (INFOA):  configuration data - do not use!! ;mk
2839  .else  ; FR5969 == FR5739
2840    .org INFOD  ; (INFOD 128 Bytes):  COLD start USER area data, 32 Bit used
2841  .endif
2842
2843  ;; User variable init values:
2844
2845  UZERO:
2846  ;  init value  ; offset to UPP
2847  ;--------------;-------------------
2848    .dw HI       ; 000H boot routine
2849    .dw BASEE    ; 002H BASE
2850    .dw 0        ; 004H tmp
2851    .dw 0        ; 006H #TIB
2852    .dw 0        ; 008H >IN
2853    .dw 0        ; 00AH HLD
2854    .dw INTER    ; 00CH 'EVAL
2855  ; .dw COLD-6   ; 00EH CONTEXT pointer                          ;MM--180713
2856    .dw MCUID+2  ; 00EH DIC    ; points to na of last word in the dictionary
2857    .dw CTOP+8   ; 010H CP
2858    .dw DPP      ; 012H DP
2859  ; .dw COLD-6   ; 014H LAST
2860    .dw MCUID+2  ; 014H LAST
2861    .dw 0        ; 016H CURR   ;MM++180712  new CURRENT pointer
2862    .dw 0        ; 018H CONT   ;MM++181003  new CONTEXT pointer
2863    .dw 0        ; 01AH VOCP   ;MM++180713  new VOC context pointer
2864    .dw 0        ; 01CH CSR    ;MM++181124  context switch request, wid(voc) or 0
2865    .dw 0        ; 01EH LBB    ;MM++181208  buffer to collect lexicon bits
2866  ULAST:
2867
2868  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
2869
2870  ;   .sect   ".reset"    ; MSP430 RESET Vector  ;mk
2871
2872  ; Interrupt vectors are located in the range FFE0-FFFFh.
2873  ;       .org 0FFE0h
2874  ; intvecs:
2875  ;        DC16  VECAREA+00     ; FFE0 - not used
2876  ;        DC16  VECAREA+04     ; FFE2 - not used
2877  ;        DC16  VECAREA+08     ; FFE4 - IO port P1
2878  ;        DC16  VECAREA+12     ; FFE6 - IO port P2
2879  ;        DC16  VECAREA+16     ; FFE8 - not used
2880  ;        DC16  VECAREA+20     ; FFEA - ADC10
```

```
;        DC16  VECAREA+24     ; FFEC - USCI A0/B0 tx, I2C tx/rx
;        DC16  VECAREA+28     ; FFEE - USCI A0/B0 rx, I2C status
;        DC16  VECAREA+32     ; FFF0 - Timer 0_A3
;        DC16  VECAREA+36     ; FFF2 - Timer 0_A3
;        DC16  VECAREA+40     ; FFF4 - Watchdog
;        DC16  VECAREA+44     ; FFF6 - Comparator A
;        DC16  VECAREA+48     ; FFF8 - Timer 1_A3
;        DC16  VECAREA+52     ; FFFA - Timer 1_A3
;        DC16  VECAREA+56     ; FFFC - NMI, osc.fault, flash violation

         .org 0FFFEh
         DC16  init           ; FFFE - Reset

    .end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```