EXTENDED MEMORY OPERATIONS FOR F83

ABSTRACT

A set of extended memory operators based on a 32 bit address or pointer is added to the F83 virtual FORTH machine. This permits accessing all memory in the host microcomputer while still retaining the basic 16 bit model. As an application of using the additional accessable memory, a simple technique for storing FORTH Word definition bodies in extended memory (outside dictionary segment) and dynamically retrieving them is described. When executed, a copy of the definition body is created on an 'execution stack' and execution proceeds as with any other word. For longer definitions or especially for those with significant looping, the runtime retrieval overhead is a relatively minor price to pay to conserve limited dictionary space.

EXTENDED MEMORY OPERATORS

The F83 FORTH model by Laxen and Perry supports only 16 bit addressing of a 64 kilobyte memory space. When a larger memory space is available on the underlying hardware, as with the Intel 8086 series microcomputers, some sort of extended addressing mode is needed. The approach described herein adds a set of extended memory operators based on a 32 bit address or pointer to the F83 virtual machine. These operators, which are designated by prefixing an 'X' to the 16 bit address memory operators (X@, X!, etc.), permit access to all memory in the host computer. These operators expect a 32 bit address where the normal @ and ! expect a 16 bit address. Additional operators XMOVE and XFILL are defined as analogs to CMOVE and FILL except that the extended memory versions have 16 bit instead of byte arguments as well as 32 bit addressing.

The 32 bit address is defined as a pointer, not a FORTH double number, and may therefore be implementation dependent in format. For the Intel machines it is convenient to define a pointer as a segmented address compatible with the hardware. This dictates that the offset is on top of the segment on the stack. For this reason it is desirable to include an operator IXAD for incrementing a pointer by a specified number of bytes.

When F83 is run under MS-DOS(TM), the remainder of available memory from the end of the COM file on is allocated by the system to the F83 task. This area is considered to be a memory resource named HEAP and is alloted in a manner similar to the Dictionary by the word HEAPALLOT.

HEAP COMPILER

In addition to the obvious use of HEAP memory for large data structures, some additional program space can be obtained by storing the executable lists of larger FORTH words in HEAP. The approach presented here defines a mode flag HPMODE which determines whether the body of a new definition is to reside in the dictionary or in HEAP. The compiler is modified to test the HPMODE flag, and if enabled, to allocate space in HEAP, move the executable body to HEAP and set up for dynamic retrieval of the body at execution time. Since the body must be within the Dictionary segment at runtime to execute properly, the runtime action of all HEAP resident words is to copy the body of the word onto an 'execution' stack before beginning normal execution of the word. For small, simple definitions this overhead would be severe, so the HEAP compilation mode should only be enabled for longer definitions and/or for those with extensive looping.

The parameter field of a HEAP based colon definition contains a pointer to a 'counted' list in HEAP. The counted list consists of a 16 bit word count followed by the executable body of the definition. The code field of the HEAP based word is modified to execute a procedure which fetches the pointer, obtains the word count, transfers the body to the execution stack, and finally executes the body.

The obvious benefit of storing the executable list of a new definition in HEAP is the savings in Dictionary space for systems with limited Dictionary resources. However, the fact that a word's definition is not intimately attached to its header allows dynamic redefinition by merely replacing the HEAP pointer with a pointer to a different execution list. Unfortunately, since the executable body of the definition must be preceeded by a word count, the current implementation does not permit redirection of a HEAP based word to an existing Dictionary word body.

LIMITATIONS

One limitation is that the approach does not work on the F83 kernel as normally distributed. The only complication is that the creators of F83 chose to use absolute addresses for branch arguments, and thus F83 bodies will not execute properly if moved. Fortunately it is trivial to modify the KERNEL86.BLK and META86.BLK source and regenerate a kernel with relative branch arguments. My own opinion is that relocatability is of sufficient value to be worthwhile over and above its application to this package.

There is a moderately constraining practical limit on the depth of nesting permissible on the execution stack since it requires some of the very Dictionary space salvaged by storing word bodies in HEAP. The problem only becomes critical, however, when recursion is used, since the retrieval algorithm blindly adds a new copy to the execution stack for each level of recursion. Simple solution: don't store recursive words in HEAP.

Finally, no mechanism is included in the current package to save the contents of HEAP when SAVE is invoked. Since part of the executable code is in HEAP if the HEAP compiler has been enabled, SAVE should be augmented to save and reload the allocated portion of the HEAP memory. On my own FORTH system I have saved HEAP data as a separate (from the .COM file) disk file which is automatically loaded when the .COM image is loaded. An alternate possibility would be to create an .EXE image with both the Dictionary and HEAP areas included.

SOURCE CODE

The source code for the extended memory primitives and the HEAP compiler are available on the East Coast Forth Board (telephone # 703-442-8695) in file ROHDAF83.BLK.

```
     1
\ EXTENDED MEMORY PRIMITIVES                         11DEC86RHD
ONLY FORTH ALSO DEFINITIONS HEX
CODE X@  (S ptr > n )
   BX POP  ES POP  ES: 0 [BX] PUSH  NEXT  END-CODE
CODE X!  (S n ptr > )
   BX POP  ES POP  ES: 0 [BX] POP  NEXT  END-CODE
CODE X2@  (S ptr > d )
   BX POP ES POP  ES: 2 [BX] PUSH ES: 0 [BX] PUSH NEXT END-CODE
CODE X2!  (S d ptr > )   BX POP  ES POP
   ES: 0 [BX] POP  ES: 2 [BX] POP  NEXT  END-CODE
CODE XC@  (S ptr > b )   BX POP  ES POP
   ES: 0 [BX] AL MOV  AH AH XOR  AX PUSH  NEXT  END-CODE
CODE XC!  (S b ptr > )   BX POP  ES POP
   AX POP  AL ES: 0 [BX] MOV  NEXT  END-CODE
CODE CS@  (S > sg )  CS PUSH  NEXT  END-CODE
DECIMAL
```

```
     33
\ EXTENDED MEMORY PRIMITIVES                         14JAN87RHD

The extended memory operators are based on a 32 bit memory
address (pointer).  For the Intel 8086 series computers, the
pointer is stored with the low order (offset) preceeding the
high order (segment) in memory.  This provides compatibility
with the hardware.  Since a pointer is NOT a double number,
there is no conflict with FORTH conventions.

The X@ and X! operators fetch and store 16bit data.
The X2@ and X2! operators fetch and store 32bit data.
The XC@ and XC! operators fetch and store 8bit data.
The CS@ operator provides access to the dictionary segment.
```

```
     2
\ EXTENDED MEMORY PRIMITIVES                         14JAN87RHD
HEX
CODE XMOVE  (S ptr1 ptr2 wct > )
    SI AX MOV  DS DX MOV  CX POP  DI POP ES POP  SI POP DS POP
    CLD  REP  MOVS  AX SI MOV  DX DS MOV  NEXT  END-CODE
CODE XFILL  (S ptr wct n > )
    AX POP  CX POP  DI POP ES POP
    CLD  REP  AX STOS  NEXT  END-CODE
CODE IPTR  (S ptr n > ptr' )
    AX POP  DX POP  AX SHL
    U< IF  BX POP  F000 # BX ADD  BX PUSH  THEN  AX DX ADD
    U< IF  BX POP  1000 # BX ADD  BX PUSH  THEN
    DX PUSH   NEXT  END-CODE
CREATE DOHP  (S > ad )   ( runtime ) (S > ptr )  ASSEMBLER
    CS DX MOV   2 [W] AX MOV   4 [W] DX ADD   10 # DH ADD
    2PUSH   END-CODE     DECIMAL
```

```
     34
\ EXTENDED MEMORY PRIMITIVES                         09FEB87RHD
XMOVE and XFILL are similar to CMOVE and FILL except that
they utilize 32bit memory pointers and 16bit word arguments.

IPTR increments a 32bit pointer by n words.  Overflow from
the low-order computation is used to appropriately increase
the high-order part of the pointer, but no segment realignment
is performed.  With care, . R may be used to increment
dictionary (16bit) addresses by n words.

DOHP is a variable header for the runtime entrypoint for an
extended memory referencing word.
```

3

```
\ ROHDA-FORTH HEAP ALLOCATION                            09FEB87RHD
2VARIABLE FREEHEAP    WARNING OFF
: LITERAL (S n > )  STATE @ IF DUP (.) OVER 1- C! 1- FIND
   IF SWAP ELSE COMPILE (LIT) THEN DROP , THEN ;  IMMEDIATE
: ?IMMED  ( cfa > f )  >NAME @ 64 AND   ;
: IAD1   1 IPTR  ;
: SEGALIGN (S ptr > ptr' )  16 /MOD SWAP IF 1+ THEN + 0  ;
: HEAPALLOT  ( n > rptr )    >R
    FREEHEAP 2@  DUP  16 -  R@ 2* +  0>  IF  SEGALIGN  THEN
    2DUP  R> IPTR  FREEHEAP 2!  ;       HEX
: HREL   CS@ 1000 + 0  D+ ;
: C/X  (S cfa > )   STATE @ IF , ELSE EXECUTE THEN ;
DECIMAL    : BUGFROM [ BUG ] ' HERE (DEBUG) ;
4 5 THRU    ( HEAP COMPILER )

                                   WARNING ON
```

35

```
\ ROHDA-FORTH HEAP ALLOCATION                            09FEB87RHD
LITERAL is redefined to use constants if they have been defined
?IMMED  tests the word represented by cfa for immediate
precedence.

IAD1  advances a pointer or an address (if no wrap) by 1 cell.
SEGALIGN  modifies a pointer to point to the next memory
  location that can be specified with a zero offset.
HEAPALLOT  allocates n words of extended memory and returns
  a relative pointer (from the beginning of allocatable memory)
  to the allocated area.  A more sophisticated allocation and
  recovery scheme could easily be substituted.

HREL provides high-level relocation of an offset pointer.

C/X  compile or execute depending on compilation state.
```

4

```
\ ROHDA-FORTH HEAP COMPILER                              14JAN87RHD
\ This technique cannot work unless F83 branches are
\   implemented in a relocatable manner!!!!
ONLY FORTH ALSO HIDDEN ALSO FORTH DEFINITIONS  WARNING OFF
VARIABLE HPMODE      VARIABLE XOS
HEX  D000 XOS !

: HEAP  HPMODE ON ;
: DICT  HPMODE OFF ;
: EXE   ( ad > )   >R ;
: LST@  ( ptr > ptr' wdct )  2DUP 1 IPTR 2SWAP X@ ;
: EVAL  ( ptr wdct > ? )  DUP NEGATE  XOS @  DUP >R  SWAP IPTR
    DUP XOS !  CS@ SWAP ROT XMOVE  XOS @ EXE  R> XOS !  ;
: @EVAL  ( ad > )  2@ HREL  LST@  EVAL ;
DECIMAL
```

36

```
\ ROHDA-FORTH HEAP COMPILER                              09FEB87RHD
The HEAP compiler stores the body of the definition in HEAP
memory.  The dictionary entry contains the relative heap pointer
at which the body is stored.  At runtime the body is copied onto
an 'execution stack' (in the dictionary segment at present) and
executed.
HPMODE is a flag determining whether to store in DICT or HEAP
XOS is the stack pointer for the 'execution stack'
HEAP or DICT select HEAP or DICTionary compilation, respectively

EXE sets up execution of a colon definition body.
@EVAL, EVAL and LST@ provide tools for the retrival and
execution of LISTS (: def bodies) stored in HEAP as a word
count followed by the list of words (analogous to byte count
followed by character sequence for strings).
```

5

```
\ ROHDA-FORTH HEAP COMPILER                              14JAN87RHD
ALSO HIDDEN DEFINITIONS
: X;  ( h > )  LAST @ NAME> >BODY
   HERE OVER - 1+ 2/ DUP >R  ( CT)  1+ HEAPALLOT
   2DUP HREL  2DUP R@ -ROT X!  2+
   CS@ 5 PICK 2SWAP R> XMOVE
   ROT DUP 2 IPTR DP !  2!  ;

FORTH DEFINITIONS
: :   [COMPILE] :  HPMODE @  IF   DOES>  @EVAL  THEN  ;
  IMMEDIATE
: ;   [COMPILE] ;  HPMODE @  IF  X;   THEN  ;
  IMMEDIATE

ONLY FORTH ALSO DEFINITIONS DECIMAL
  WARNING ON
```

37

```
\ ROHDA-FORTH HEAP COMPILER                              09FEB87RHD

X; sets up the word count, allocates space in HEAP, and moves
the body of the definition to HEAP.  The HEAP offset to the
list is stored in the parameter field of the dictionary entry
and the remainder of the dictionary space is reclaimed.

The fact that the interpret loop is buried INSIDE of : in F83
necessitates the redefinition of both : and ;.  Normally
only redefinition of ; is required.
                                                     HPMODE is
checked in both cases and appropriate additional action is
taken if HEAP compilation is enabled.
```

Forth 83 Model