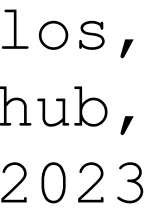


The words in a MITC Forth

Alvaro G. S. Barcellos,
agsb at github,
version 0.9, 2023





the Minimal Indirect Thread Code

1. *Any primitive word contains only a machine code*
2. *All primitive words are performed in a similar way to 'link and jump'*
3. *Any compound word contains only a list of word references*
4. *All compound words have the references 'pushed and pulled' onto the return stack*
5. *Any compound word should only access the return stack using >R R> R@*

Notes

The 'jump and link' concept was used by Charles H. Moore in 70's when --IP holds the address to next instruction--.

In the follow examples:

The names NEXT, NEST (aka doCOL), UNNEST (aka SEMIS), are classic names for Forth inner interpreter, LINK and JUMP are traditional.

Using reference ITC from FIG-Forth for PDP-11

Using PDP-11, Instruction Set Architecture (ISA)

By convention, stacks grows downward, (X) indirect access, increment+ (pós), -decrement (pré)

the MITC dictionary

compound: a list of references, ends with ENDS

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| LINK  | 6  | D  | O  | U  | B  | L  | E  | spc | DUP | PLUS | ENDS |
+LFA-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+CFA-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

primitive: starts with NULL and ends with a jump

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| LINK  | 3  | D  | U  | P  | NULL | code | code | code | LINK |
+LFA-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+CFA-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

PS. NULL is 0x0000, LINK is 'jump link:', spc is space (0x20)

call or link

from PDP-11 ISA

```
JSR Ri, Src // Jump into subroutine  
-(SP) ← Ri; Ri ← +PC; PC ← Src;  
RTS // Return from subroutine  
PC ← Ri; Ri ← (SP)+;
```

```
push and pull // same, but do not use Ri  
CALL Src  
-(SP) ← Ri; Ri ← +PC; PC ← Src;  
RETURN  
PC ← Ri; Ri ← (SP)+;
```

```
link and jump // same, but do not use SP  
JAL Ri, Src  
-(SP) ← Ri; Ri ← +PC; PC ← Src;  
JR Ri  
PC ← Ri; Ri ← (SP)+;
```

the code of MITC

```
unnest: MOV (RP)+, IP    // .pull
next:   MOV (IP)+, W     // .cast
        TST W
        BEQ jump
nest:   MOV IP, -(RP)    // .push
link:   MOV W, IP        // .link
        JMP next
jump:   MOV (RP)+, W     // .jump
        JMP (IP)
```

reference version,
marked is same code as in
FIG-Forth PDP-11 1.3.3.1

W is the next caller
IP is the last callee

```
HEADER "ENDS", 'ends'
.word    0x000
unnest: ...
```

the code of MITC

```
unnest: MOV (RP)+, IP    // .pull
next:   MOV (IP)+, W     // .cast
        TST W
        BEQ jump
nest:   MOV IP, -(RP)    // .push
link:   MOV W, IP        // .link
        JMP next
jump:   MOV (RP)+, W     // .jump
        JMP (IP)
```

from FIG-Forth, 1.3.3.1:

```
SEMIS : MOVE (RP)+, IP
        NEXT

NEXT :  MOV (IP)+, W
        JMP @(W)+

DOCOL : MOV IP, -(RP)
        MOV W, IP
        NEXT
```

```
HEADER "ENDS", 'ends'
.word   0x000
unnest: ...
```


ITC and MITC

In ITC

1. uses two registers, both are scratch
2. CFA is always a reference to a machine code routine
3. must jump twice for every word

In MITC

1. uses two registers, one must be preserved (link register)
2. CFA is always a reference to a word or a NULL
3. must test the CFA of each word but only jump when it is NULL
4. *performs a deep-first search for primitive words*

inside words



FIG-Forth ITC, inside words

```
doVAR: MOV W, -(SP) ; NEXT ;  
doCON: MOV (W), -(SP) ; NEXT ;  
LIT:    MOV (IP)+, -(SP) ; NEXT ;  
BRANCH: ADD (IP), IP ; NEXT ;  
EXEC: MOV (SP)+, W ; JMP @(W)+ ;  
(VALUE): MOV (SP)+, (IP)+ ; NEXT ;
```

MITC, inside words

```
doVAR: MOV W+, -(SP) ; LINK ;
```

```
doCON: MOV (W)+, -(SP) ; LINK ;
```

```
LIT:    MOV (W)+, -(SP) ; LINK ;    // same as doCON
```

```
BRANCH: ADD (W), W ; LINK ;
```

```
EXEC: MOV W, -(RP) ; MOV (SP)+, W ; LINK ;
```

```
(VALUE): MOV (SP)+, (W)+ ; LINK ;
```



MITC as macros

```
HEADER, 'ENDS', ends,  
.word    0x0000  
unnest:  PULL wp, rp  
next:    MOVE ip, (wp)  
          ADD  wp, #cell  
          BEQ  ip, zero, jump  
nest:    PUSH wp, rp  
link:    MOVE wp, ip  
          JUMP next  
jump:    PULL ip, rp  
          JUMP (wp)
```

syntax to, from

wp, work pointer, scratch
ip, link pointer, reserved
rp, return stack pointer, reserved

#cell is the cell size of Forth

do not use any default link register

primitive code starts with 0x0000
primitive code ends with **JUMP link:**

```

unnest: lw W, RP
        addi RP, RP, 1 * #CELL
next:   add IP, W, zero
        addi W, W, 1 * #CELL
        BEQ IP, zero, jump
nest:   addi RP, RP, -1 * CELL
        sw RP, W
link:   add W, IP, zero
        jal zero, next
jump:   lw IP, RP
        addi RP, RP, 1 * #CELL
        jalr zero, 0(W)

```

MITC, Risc-V

pull, push, move, cast, jump

syntax to, from

not use default link register

let assembler decide offsets for
(jalr zero, zero, link)

```

// ends is unnest
HEADER "ENDS", ends,
.word    0x000
unnest: ...

```

MITC, PDP-11

```
unnest: MOV (RP)+, WP    // .pull
next:   MOV (WP)+, IP    // .cast
        TST IP
        BEQ jump
nest:   MOV WP, -(RP)    // .push
link:   MOV IP, WP      // .link
        JMP next
jump:   MOV (RP)+, IP    // .jump
        JMP (WP)
```

pull, push, move, cast, jump

syntax from, to

wp, work pointer, scratch

ip, link pointer, reserved

rp, return stack pointer, reserved

(use of IP and WP has been swapped)

```
// ends is unnest
HEADER"ENDS",ends,
.word    0x000
unnest:
```


Notes

1. need tweak

```
:NONAME DOES> DEFER TO IS EXIT
```

```
VALUE ASSIGN
```

2. need a ;CODE for primitive words

conclusion

When reinvent the wheel ?

MITC is a faster inner interpreter;

MITC is more effective than ITC;

MITC needs little changes at few primitive words;

MITC uses less memory and less jumps;

references

https://library.nrao.edu/public/memos/comp/CDIR_17.pdf

<https://pdos.csail.mit.edu/6.828/2005/readings/pdp11-40.pdf>

<http://www.stackosaurus.com/figforth-1.3.3.1/FORTH.MAC>

<http://www.complang.tuwien.ac.at/forth/threaded-code.html>

<http://www.bradrodriguez.com/papers/moving1.htm>

<https://muforth.nimblemachines.com/threaded-code/>

<http://git.annexia.org/?p=jonesforth.git;a=tree>

<https://home.hccnet.nl/a.w.m.van.der.horst/lina.html>

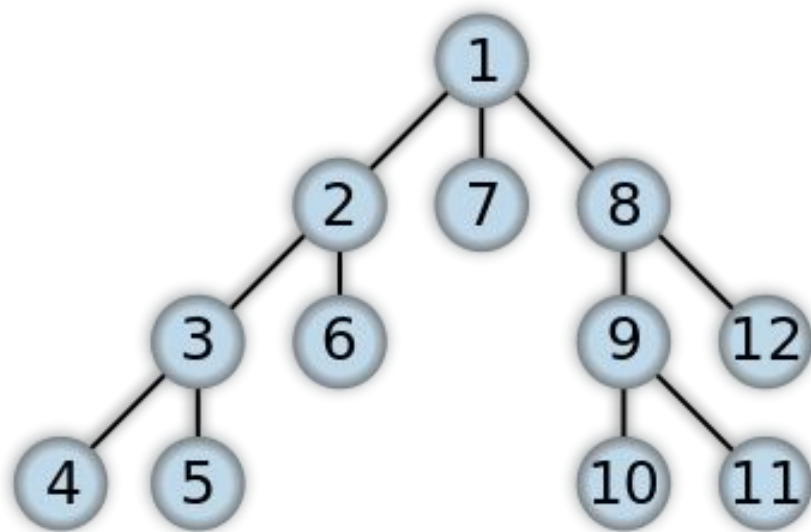
<https://github.com/simh/simh>



Why PDP-11 ?

the **Pi**DP-11 is a replica of the PDP-11/70, with a Raspberry Pi running the simh simulator for PDP-11/70, in a Linux Debian.

deep-first search



Compound words (aka twig)

1, 2, 3, 8, 9

Primitive words (aka leaf)

4, 5, 6, 7, 10, 11, 12

https://en.wikipedia.org/wiki/Depth-first_search