**nextFORTH**

The main problem with FORTH on the Z80 is the two stacks ; or more accurately the lack of them. This describes a FORTH system which maintains most of the features of FORTH except:

* Functions are not re-entrant and cannot be recursed (except possibly tail recursion as in Color Forth)
* Functions cannot be multi-tasked (because they are not re-entrant)
* The stack cannot be used as a temporary storage using R> and >R. These have three uses ; temporary store in an operation, index values in loops, and manipulating return addresses. All are done differently.

How the second stack is avoided varies from feature to feature. Examples are given below.

The FORTH is loosely based on CMFORTH but doesn’t have direct compatibility with it. The main change FORTH users will find is FOR/NEXT replacing DO/LOOP and FORTH/COMPILER replacing IMMEDIATE.

**Standard Call**

The standard call routine operates as follows. It is my opinion that this is about 3 times faster than a standard call compiled using normal threading.

EntryPoint: pop hl ; get return addr

Ld (ExitPoint+1),hl ; update the jump address

ExecutePoint:

<word code>

ExitPoint: jp $0000 ; address overwritten.

This shows why functions cannot be re-entered ; it will overwrite the return address stored at ExitPoint+1.

Tail end recursion can be done by jmp ExecutePoint.

Each routine can only have one exit point. Thus EXIT is slightly more complicated as the address ‘ExitPoint’ may not be known. Therefore you need to read the address at EntryPoint+2 (the store address), which will be correct at run time, and jump to that address – 1.

**Short Call**

Short calls are used on machine code instructions. If the routine only affects the stack top and doesn’t change the stack depth then you can simply use RET.

So, for example, c@ could be implemented as a word as :

Read8Bit: ld a,(de)

Ld e,a

Ld d,0

Ret

And as a macro, it would just generate

ExecutePoint:

Ld a,(de)

Ld e,a

Ld d,0

If words manipulate the stack the return address has to be preserved. By convention this is done with either HL or IX – not IY, for example, definitions for DUP could be.

EntryPoint: pop hl

Push de

Jp (hl)

EntryPoint: pop ix

Push de

Jp (ix)

Obviously HL should be used in preference to IX as it is faster, but some simple words will need HL, for example if you are using ADD HL,DE

**Loops**

FOR/NEXT is Chuck Moore’s replacement for DO LOOP. It has a single parameter, the number of executions and executes the Code counting down to zero.

10 FOR I . NEXT will print 10 9 8 7 6 5 4 3 2 1

Implementation is done by a similar code modification.

StartPoint: ex de,hl ; put count in HL

Pop de ; restore TOS.

Loop: ld (LoopEnd+1),hl ; update the count.

<code>

LoopEnd: ld hl,$0000 ; this is overwritten

Dec hl ; dec count

Ld a,h ; check zero and loop.

Or l

Jr nz,Loop

I can be accessed in a similar manner to exit e.g. read the value at Loop+1, which is the address LoopEnd+1, then use that to read the current count value.

**Macros**

Macros can initially be generated from the code words, so there are two functions (one for FORTH and one for COMPILER) for many functions. All you have to do is strip one of the three entry/exit codes off – x->ret pop hl->jp (hl) and pop ix-> jp (ix)

You will be left with a block of code of a given length, which can be copied as a macro.

**Dictionaries**

There are four dictionaries 0-3, FORTH is dictionary 1 and COMPILER dictionary 2. These are physically separate from the code, and occupy the contended memory space from $6000 - $8000.

Each is a pointer to the head of the list, which is constructed backwards (e.g. most recent entry first)

|  |  |
| --- | --- |
| Offset | Contents |
| +0,+1 | Address of previous dictionary entry, or $0000 if it is the last element |
| +2,+3 | Address of word in memory |
| +4 | Bits 0..4 : Page number  Bit 5: Not currently used (zero)  Bit 6: Not currently used (zero)  Bit 7: Speed optimisation flag. If this is set in COMPILER then it is only compiled if speed optimisation is on. Otherwise it will compile as a CALL which for some is slightly more compact. Should not be set if the code generated is three bytes or fewer, obviously 😊 |
| +5 | First character of word (ASCII lower case) |
| +6 | Second character of word (ASCII lower case) |
| +n | Last character of word (ASCII lower case), bit 7 set. |