◎FIG-FORTH v64の構造の簡単なメモ

ここから先の説明は、ToUnderstandFORTHは作成途中で整理できていないので、FIG-FORTHのソースコードで説明しています。

FIG-FORTHは主にワードが集まった「辞書」と機械語で記述された「FORTH\_INNER\_INTERPRETER」で成り立っている。

１）「辞書」は登録されている全ワードが集まったものである。それぞれのワードにはそのワードに必要な全情報が収められている。FIG-FORTHでは、それぞれのワードをNF、LF、CF、PFの四つの部分に分けている。詳細は図を見てもらいたいが、大まかに分けると次のようになる。

　NF（Name Field）

：このワードのワード名やその文字数、IMMEDIATE/SMUDGEビットが入っている

　LF（Link Field）

　　：一つ前に登録されたワードのNFAが入っている。

　CF（Compilation Field）

　　：このワードが実行されるときの最初に実行される機械語（ワード「：」に格納されている機械語の先頭アドレスDOCOLから始まる機械語）のアドレスが入っている。

　PF（Parameter Field）

　　：CFで述べたように、このワードの実行時にはまずCFでの処理が行われる。その後にPFが実行される。PFでは、ワードの情報が順番に入っている。詳しく言うと、並んだそれぞれのワードの順番に従って、それぞれのCFAが並んで入っている。

これとは別にPFの最初から機械語で始まる場合は、PFの最初には実行する機械語の開始アドレスが入っており、その後から機械語の列が並んで入っている。最初がワードのアドレスの列で途中から機械語の場合もあるが、ここでは割愛する。

ここでNFAは（Name Field Address）、CFAは（Compilation Field Address）の略である。

LF、CF、PFに格納されているデータ長はFIG-FORTHでは１６ビットを扱うので２バイト（ToUnderstandFORTHでは６４ビットを扱うので８バイト）である。

・以下はあるワード「CALC」の例（一部分のみ）である。

![ ... ](/JPN/Documents/03\_01\_v0.20.01\_JPN/NFR.png)

２）FORTH\_INNER\_INTERPRETERはFORTHの内部インタプリタ（今流ならVM(-->Virtual Machine)）です。それは次に実行すべきワード（の実行アドレス）を判断し、そのワードに処理を移す（ジャンプする）。そのワードの処理が終わればまた起動して、次のワードに処理を移します。基本的には次の機械語コードとなります。FORTH初心者の私はこのコードを見て、「たったこれだけの命令たちでFORTHの内部インタプリタはできている！」と心底驚きました。

\*\*\*\*FORTH\_INNER\_INTERPRETER\*\*\*\*

DPUSH: PUSH RDX ; 注意２）

APUSH: PUSH RAX

NEXT: LODSQ ; RAX <- (RSI) 　　注意１）

; RSI <- RSI+8 IP<-IP+8

MOV RBX,RAX

NEXT1: MOV RDX,RBX

ADD RDX,8 ; SET W

JMP QWORD PTR [RBX] ; JUMP TO (IP)

各レジスタの役割：

FORTHレジスタ x64レジスタ　　　用途

IP RSI 内部インタプリタの命令ポインタ

SP RSP パラメータ・スタック・ポインタ

RP RBP リターン・スタック・ポインタ

W RDX 作業用レジスタ

RAX,RBXなどをこれらのレジスタの作業の補助レジスタとして使用

注意１）FIG-FORTHではIP (Instruction Pointer　命令ポインタ)レジスタとしてSIレジスタを使用していました。ToUnderstandFORTHは６４ビット幅で処理するインテル製x64で作成されています。

そのため、LODSQ命令ではレジスタであるRSIの内容をレジスタRAXに移動（コピー）して、RSIの値を次のワードのあるアドレスへと加算されます。

ここではRSI,RAXは６４ビット幅なので、次のワードのアドレスに進むにはレジスタに８バイトが加えられます。

注意２）あるワードから次のワードに処理を受け渡すのに必要となるデータをスタックを使って受け渡します。その際の受け渡すパラメータ数が０，１，２個のスタックが使われます。

具体的には、ワード「CALC」はその実行時の処理の流れが様々なワードの列の順番で定義されています。基本的なワードはほとんどが実行部分は機械語でできています。そのため、FIG-FORTHでは一つのワードの処理が終わって次のワードに引き継ぐときは、大抵はRAXレジスタやRDXレジスタにそのデータを入れたままにしておいて、FORTH\_INNER\_INTERPRETERでそれらのレジスタの値をスタックに積み上げています。そして、次のワードが始まるときにはスタック上のレジスタの値を必要分だけそれぞれのレジスタに取り出します。

あるワードを終了してFORTH\_INNER\_INTERPRETERに飛び込む（JMP命令の実行)時に、その開始アドレスをラベルのNEXT、APUSH、DPUSHから選んで飛び込むと、スタックに積まれるデータは０個（NEXTから入って何もスタックから移し替えない），１個（APUSHから入ってRAXレジスタにスタックから１個分を移し替える），２個（DPUSHから入ってRDXレジスタとRAXレジスタにスタックから２個分を移し替える）になります。どこから実行するかで、スタックから移し替えるデータの数を変えています。

そして、次に実行すべきワードの実行部分のアドレスをRBXレジスタに、そのワードの実行部分が必要とするデータのアドレス（RBXから８バイト先になる）をRDXレジスタに書込んで、JMP QWORD PTR [RBX]で次のワードのCFAに制御を渡します。こうやってあるワードの終了時のレジスタの値がスタックに積まれて、次のワードで必要な数のスタック上のデータが内部で機械語により取り出されて利用されます。

・ToUnderstandFORTHのVisual C++プログラムとMASMプログラムの関係

キーの入出力で、FIG-FORTHではキーの１文字入力処理が必要でした。しかし、MASMプログラム上のcall \_getch()ではENTERキーを押すまでキーを読み続けてしまい、１文字入力になりませんでした。

そのため、キーの入出力処理を行う時は、一旦MASMプログラムから抜けて、Visual C++プログラム上で関数\_getch()を使ってキーの入出力処理を行い、またMASMプログラムに戻って実行の続きを行うようにしています。

◎ToUnderstandFORTH実行時の全体処理の流れ

・最初に機械語で書かれた初期設定処理が行われる

・次にFORTH\_INNER\_INTERPRETERにより、最初のワード「COLD」が実行される。

・ワード「COLD」はその処理を行うための一連のワード群でできている。それらは初期設定に必要な様々なワード群であり、それらのワードを実行する時にはFORTH\_INNER\_INTERPRETERが実行される。

・ひとつのワードの処理が終わると、またFORTH\_INNER\_INTERPRETERが実行されて、次のワードの処理が始まる。

・ワード「COLD」から呼び出される一連のワード群の最後のワードとして、ワード「ABORT」がある。

・ワード「ABORT」から呼び出される一連のワード群の最後のワードとして、ワード「QUIT」がある。

・ワード「QUIT」から呼び出される一連のワード群の中で、キーボードから入力された一連の文字列（ワードの列）の処理が行われる。

・ToUnderstandFORTHの初期状態はキー入力はないので、何も処理しないで画面上に「OK」が表示され、キー入力待ちとなる。

次の手順はToUnderstandFORTHの初期状態でなかった場合に行う処理です。

・ワード「QUIT」から呼び出される一連のワード群の中で、キーボードから入力された一連の文字列（ワードの列）の処理が行われる。

・その処理の後で、ワード「INTERPRET」で文字列が示す数値やワードが解釈／実行される。

・これらの一連の処理は入力された文字列が最後まで実行されるまで繰り返される。

・入力された文字列が最後まで実行されると（もしくはエラーが発生すると）、画面上に「OK」（もしくはエラー情報）が表示され、キー入力待ちとなる。

・参考のためにINOUE-FORTH（機械語はｘ６４版ですが）でのソースコードの一部を載せておきます。これはあまり手を加えていないときのソースコードで、多少バグがあります。。。

; COLD STATE VECTOR COMES HERE

CLD0: MOV AX,CS

MOV DS,AX

MOV ES,AX

MOV SS,AX

CLD

MOV SI,OFFSET CLD1 ; SET UP IP

MOV SP,WORD PTR ORIG+10H ; SET UP SP

MOV BP,WORD PTR ORIG+12H ; SET UP RP

JMP NEXT

CLD1: DW COLD

; COLD

DB 84H

DB 'COL'

DB 'D'+80H

DW WARM-7

COLD: DW DOCOL

DW LIT,UVR ; set user variables

DW LIT,UP,ATT

DW LIT,50

DW CMOVE

DW EMPBUF ; EMPTY-BUFFERS

DW ABORT ; ABORT

( # 336 )

: ABORT ( clear stacks, warm start )

SP! DECIMAL DR0 [COMPILE] FORTH DEFINITION QUIT ;

: QUIT ( restart, interpret from terminal )

0 BLK ! [COMPILE] [

BEGIN RP! CR QUERY INTERPRET

STATE @ 0= IF ." OK" THEN

AGAIN ;

（私は普通はQUITって言ったら、そこでコマンドプロンプト画面の状態まですべての処理が終わってしまうのかと思いました。ですが、実はQUITの処理の中でキー入力状態となり、読み込んだFORTHの文字列を処理して、それが終わったらまたキー入力状態になってと、永遠に命令の入力、実行を繰り返しています。このFIG-FORTHでの実行処理をQUITが行っています。

私はプログラムの処理はメインルーチンからサブルーチンへと呼び出すもの（CALLの処理に相当)で、サブルーチンが終われば必ずメインルーチンに帰るものだと思っていました。これはソフトウェア開発が進化して得た知恵なのかと思います。FORTHは別の道を歩んできたように思います。FORTHはワードからワードへ処理が移るときは、一旦FORTH\_INNER\_INTERPRETERに飛び込んで（JUMPの処理に相当)、そこから次のワードに飛び込む（JUMPの処理に相当）を繰り返します。ワードとワードの間にはメインとサブのような上下関係はなく、対等で独立しているように私は感じています。私はそれを面白いと感じています。）