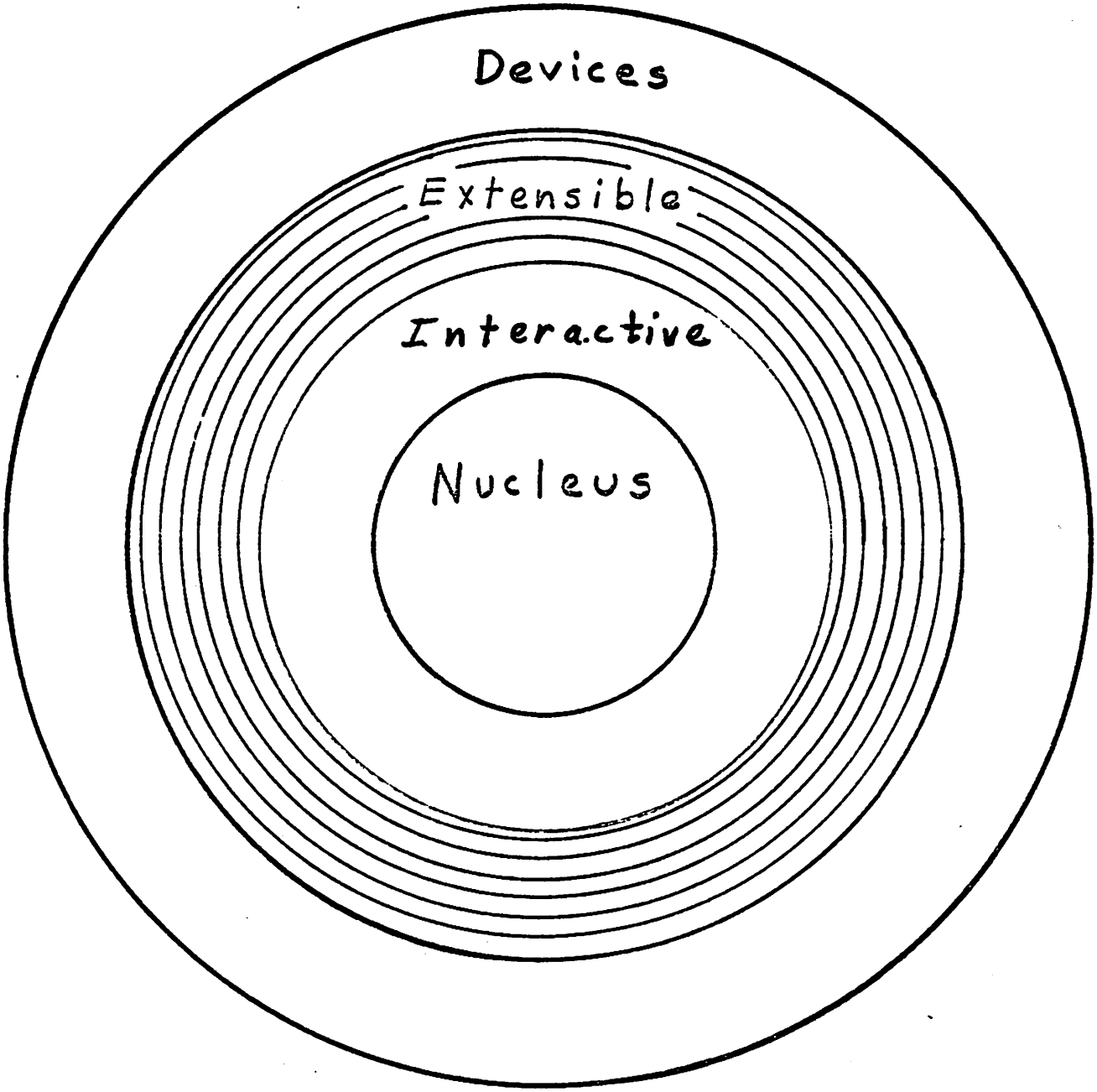
FORTH ASSEMBLER

Application Layers



FORTH ASSEMBLER

ATTRIBUTES:

"CODE" words interface exactly like ":" words
- universal reference
- stack arguments

Allows full machine speed
and full access to hardware details
- carry, overflow flags
- interrupts

Resident vocabulary
Source from keyboard or disk,
Object code to normal memory (normal mode)
or to disk and alternate memory space
(target compiling mode)
Macro capability
Structured programming control structures
"Meta assembler" (table driven) allows
full control over assembly process
All capabilities of FORTH system available during assembly, eg. assembly-time calculations,
dictionary search,
editing.

USEAGE:

CODE name	body	ending
CREATEs dictionary head for "name" and invokes ASSEMBLER vocabulary (as the CONTEXT vocabulary)	assembler words literals FORTH words	jump to address interpreter or multitasker

Examples are for LSI-11 poly FORTH

Endings:

NEXT	(macro)	jump to address interpreter
POP JMP		discard top of stack, jump to NEXT
PUSH JMP		push register 0 onto stack, jump to NEXT
PUT JMP		store register 0 into top of stack, jump to NEXT
WAIT	(macro)	jump to multitasker (like PAUSE)

Assembler words:

1 operand instructions:

operand mnemonic

eg. CLR, NEG, ASL

2 operand instructions:

destination-operand source-operand mnemonic

eg. ADD MOV XOR

(Note: operand order is opposite on 8080's.)

Operands may be registers, numeric values (eg, immediate data, addresses), and addressing mode modifiers.

Registers:

number	name	(assignment)
0		scratch
1		scratch
2	W	
3	U	User variables base
4	I	Interpreter
5	S	Stack pointer
6	R	Return stack pointer
7	PC	processor's Program Counter

Immediate data, addresses:

value #

eg. CODE ONE 0 1 # MOV PUSH JMP

(Traditional assembler syntax: MOV #1,0)

Addressing mode modifiers:

Relative addressing reg)

eg. CODE MINUS S) NEG NEXT

(Traditional syntax: NEG (S))

Relative, post increment reg)+

eg. CODE DROP S)+ TST NEXT

(Traditional syntax: TST (S)+)

Relative, pre decrement reg -)

eg. LABEL PUSH S -) 0 MOV NEXT

(Traditional syntax: MOV 0,(S-))

Indexed displacement reg)

eg. CODE SWAP 0 2 S) MOV
2 S) S) MOV
PUT JMP

Conditional control structures:

result-flag IF true-phrase ELSE false-phrase THEN

optional

BEGIN loop-body result-flag END

result flags:

0<	Negative flag set
0>	Negative flag clear
0=	zero flag set
CS	Carry flag Set
VS	oVerflow flag Set

Macros: while in ~~the~~ ASSEMBLER DEFINITIONS

: macro-name assembler words ;

eg.

: NEXT W I)+ MOV
W)+) JMP ;

Interrupts:

address-interrupt-code address-interrupt-vector INTERRUPT

installs interrupt
interrupt-code must be CPU code (not code field addr)

Interrupt routine form:

LABEL A/D ... RTI

To install this code at address 177777_8:

A/D 177777 INTERUPT

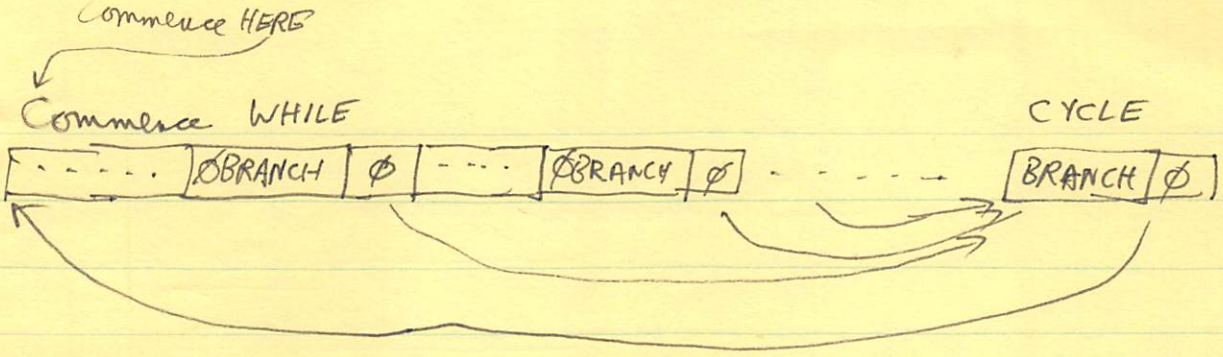
PS-E6
fig

```
 0 ( Solution:  Multiexit loop structure )
 1
 2 : (-BRANCH   HERE - ,   ;
 3 : ->BRANCH   HERE OVER -   SWAP !   ;
 4
 5 : COMMENCE   HERE   0   ;   IMMEDIATE
 6 : &WHILE   COMPILE OBRANCH   HERE 0 ,   ;   IMMEDIATE
 7 : CYCLE   COMPILE BRANCH   0 ,
 8    BEGIN   -DUP WHILE   ->BRANCH   ?STACK   REPEAT
 9    -2 ALLOT   (-BRANCH   ;   IMMEDIATE
10
11 : MULTI-TEST   COMMENCE   DUP .   1 - DUP &WHILE
12    DUP .   1 - DUP &WHILE   DUP .   1 - DUP &WHILE
13    CR   CYCLE   DROP   ;
14

10 MULTI-TEST 10 9 8
7 6 5
4 3 2
1 OK
9 MULTI-TEST 9 8 7
6 5 4
3 2 1 OK
6 MULTI-TEST 6 5 4
3 2 1 OK
2 MULTI-TEST 2 1 OK
. 46 .?  Empty Stack


 0 ( Multi-exit sequence structure )
 1
 2 : &IF   [COMPILE] &WHILE   ;   IMMEDIATE
 3 : FIN   BEGIN   -DUP WHILE   ->BRANCH   ?STACK   REPEAT
 4    DROP   ;   IMMEDIATE
 5
 6
 7 : SEQ-TEST   COMMENCE   DUP .   1 -   DUP &IF
 8    DUP .   1 -   DUP &IF   DUP .   1 -   DUP &IF
 9    DUP .   FIN   DROP   ;
10
11
12
13
14
15
OK
10 SEQ-TEST 10 9 8 7 OK
4 SEQ-TEST 4 3 2 1 OK
3 SEQ-TEST 3 2 1 OK
2 SEQ-TEST 2 1 OK
1 SEQ-TEST 1 OK
0 SEQ-TEST 0 -1 -2 -3 OK
. 46 .?  Empty Stack
```

7/5/80


Commence HERE
Commence WHILE
CYCLE
ØBRANCH
Ø
ØBRANCH
Ø
BRANCH
Ø

CVECTOR for virtual array

start block # ← decided initially

& alternatively could keep a variable DISK-HERE & wire around DISK-ALLOT

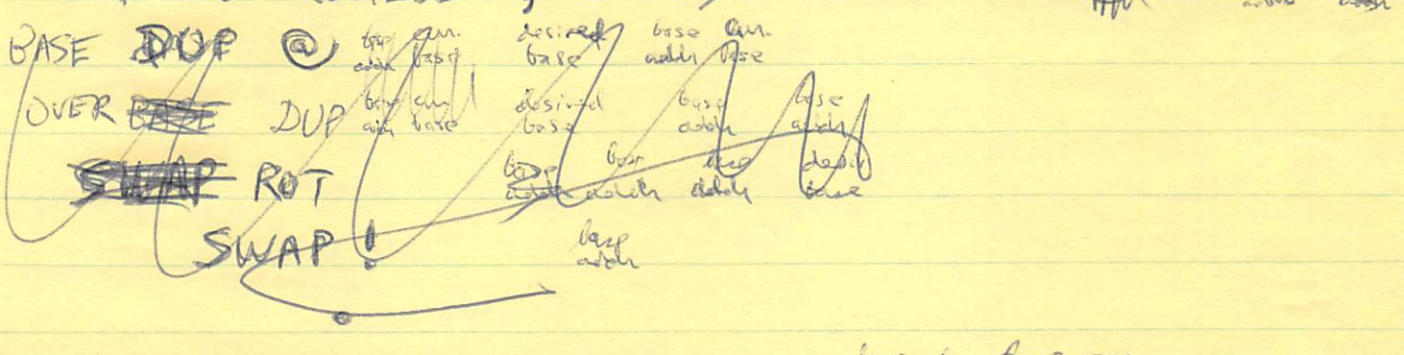
write BASED.

(2) 16 BASED. H.

(2) w H.

exec. time
1) save BASE
2) set BASE to value
3) restore

: BASED. <BUILDS , DOES> ~~BASE DUP~~



~~BASE DUP~~ @
ROT

BASE @ SWAP @ BASE ! / BASE ! ~~SWAP~~ ;

note: this is same as CONSTANT

class ---

: LOADED-BY <BUILDS DOES>

@ LOAD ;

The F16 FORTH "Kludge" for <BUILDS DOES>