ColorForth Assembler/Disassembler

June 3, 2012

Version	Date
0.0.1	22 May 2012

Contents

1 *Purpose*

I created this program to convert colorForth (**cf**) source into ASCII text and then added Albert van der Horst's **ciasdis** to illuminate the **cf** kernel. The goal is to produce a complete round trip, reproducing the **cf** file with an ASCII disassembly (**dsm**) file that can be used in its own Literate Programming document. At this stage, the round trip is produced with the following script:

3a ⟨*OkadWork* 3a⟩≡

```
#!/bin/env bash

gforth cfasdis.frt -e "cidis OkadWork.cf OkadWork.cul bye" > OkadWork.dsm
gforth cfasdis.frt -e "cias OkadWork.dsm OkadWork.cfo bye"
```

However, the assembly portion crashes, at the moment.

The cfasdis source is an extension of Albert's ciasdis as can be seen here:

3b ⟨*cfasdis.frt* 3b⟩≡

⟨*load-ciasdis* 22⟩
⟨*cfdis.f* 5⟩
⟨*cfas.f* 14⟩
⟨*run-ciasdis* 23⟩

2 Color Blindness

Before I start diving much deeper, I should explain that I am red/green color blind. This means that I do not see, or react to color like most people do. 7% of males have this condition, as well as other people who do not perceive color the same way as others do. I use a program called eyePilot (Version 1.0.12 from Tenebraex) so I can figure out what colors ColorForth is using. I don't always need it, but frequently, `yellow` and `green` look far to much alike and on the block I will be using in a few moments, I see that the green component can have an RGB value of 192 or 255. I see that the User's manual explains that these are HEX numbers, but this does not make using colorForth any easier for me.

Additionally, tools that I tend to rely on in my programming, have not caught up with the use of color in source code. HTML editors are getting closer, and colorForth even has an HTML listing utility, but I have not found them good enough yet. Most explicitly, the use of Literate Programming that I will be using in this documentation can not be done with color attributes yet. Some day, the rest of us might catch up with where Chuck wants us to be, but at the moment, I am not there yet.

Therefore, for the rest of this document, I will be using an ASCII translation of the syntax used by colorForth. This makes the code look very similar to ANS Forth, but do not be mistaken, it will not run in any other version of Forth that I know about. Here is the translation matrix:

colorForth	ASCII translation
`red`	`: red`
	`( white )`
`green`	`: ... green ... ;`
`yellow`	`[ yellow ]`
`magenta`	`:# magenta`
`blue`	`{ blue }`
`cyan`	`POSTPONE cyan`
`grey`	`< grey >`

Numbers will be preceded by the base operator that they are in (e.g. **D#** or **H#**, but **B#** and **O#** can not be translated back).

Notice that cyan and grey doesn't show up properly under the colors that LyX uses and I pity the person who is reading this with a black and white ebook reader. The purpose of this exercise is to automate this process.

2.1 License

2.2 ColoForth disassembler

Disassembler support for OkadWork.cf the arrayForth for GA144-1.10.

5 ⟨*cfdis.f* 5⟩≡ (3b)

```
\ Copyright (c) 2010 Dennis Ruffer

: <?> ( n -- )   .S DROP S" <?>" TYPE CR ;    \ a fence post to isolate issues
```

⟨*Restore.f* 11⟩

⟨*endians* 10⟩

```
: ?? ( "name" -- flag )  BL WORD FIND SWAP DROP 0= 0= ;
: uses ( flag -- )  0= IF POSTPONE \ THEN ;

?? CTRL 0= uses : CTRL CHAR 31 AND ;
?? [CTRL] 0= uses : [CTRL] CTRL POSTPONE LITERAL ; IMMEDIATE

?? ForGForth uses ALSO
ASSEMBLER

: .ICON-ROW ( x -- )   BASE @ >R  2 BASE !  S>D
    <#  16 0 DO  #  LOOP  #>  TYPE  R> BASE ! ;

16 CONSTANT ICON-COLUMNS
24 CONSTANT ICON-ROWS

: |ICON| ( -- n )   ICON-COLUMNS 8 / ICON-ROWS * ;

: IH. ( n -- a n )   0 <HEX  <# # #>  HEX>  TYPE SPACE ;

: DUMP-ICONS ( a2 a1 -- )   DUP S" icons{ " CR+GENERIC  2DUP - |ICON| /
    0 DO  I  ICON-ROWS 0 DO  CR  4 0 DO  DUP I +
                J 0= IF  16 / IH.  ELSE
                    J 1 = IF  16 MOD IH.  ELSE
                        DROP 2 SPACES  THEN  THEN
                OVER |ICON| I * + J 2* + W@-BE .ICON-ROW  SPACE
        LOOP  LOOP  CR  DROP  4 |ICON| * +  4 +LOOP
    2DUP - IF  (DUMP-B)  ELSE  2DROP
    THEN  S" }icons" TYPE CR ;

: DIS-ICONS ( a1 a2 -- )   TARGET>HOST SWAP TARGET>HOST
    DUP NEXT-CUT ! DUMP-ICONS ;
: -icons ( a n -- )   2>R ['] DIS-ICONS 2R> RANGE ;
' -icons RANGE: -icons: ( -name- )
: -icons- ( -- )   NONAME$ -icons ;
```

```
' DIS-ICONS  ' -icons:  ARE-COUPLED

\ unpack cf word to ascii text

\ 4-bit 0.xxx 0-7
\ 5-bit 10.xxx 8-15
\ 7-bit 11xx.xxx 16-47
: UNPACK ( n -- n' chr )   DUP  DUP 0<
    IF       1 LSHIFT DUP 0<
        IF   6 LSHIFT SWAP 25 RSHIFT 63 AND  16 -    \ 11xxxxx.. 16-47
        ELSE 4 LSHIFT SWAP 27 RSHIFT  7 AND  8 XOR   \ 10xxx..    8-15
        THEN
    ELSE     4 LSHIFT SWAP 28 RSHIFT  7 AND          \ 0xxx..     0-7
    THEN ;

: PRESHIFT ( n -- n' )   32 0 DO  [ HEX ]
        DUP F0000000 AND IF
            UNLOOP EXIT
        THEN  2*
    LOOP ;

: s, ( a n -- )   DUP C,  0 ?DO  COUNT C,  LOOP  DROP ;

S"  rtoeanismcylgfwdvpbhxuq0123456789j-k.z/;'!+@*,?"
CREATE cf-ii ( -- adr)   s,  0 cf-ii 1+ C!

: CH ( n -- n' chr )   0FFFFFFF0 AND UNPACK DUP cf-ii COUNT
    ROT < ABORT" invalid character" + C@ ;
DECIMAL

VARIABLE PHERE
: PAD| ( -- )   0 PAD C! ;
: PAD+ ( a n -- )   PAD APPEND ;
: PAD+BL ( -- )   S"  " PAD+ ;
: PADTYPE ( -- )   PAD COUNT TYPE  PAD| ;
: PAD, ( chr -- )   PHERE @ C!  1 PHERE +! ;
: PADDECODE ( n -- )   BEGIN  CH DUP WHILE PAD,  REPEAT  2DROP ;
: PADCOUNT ( n -- adr len )   PAD COUNT + DUP >R PHERE !
    PADDECODE  R> PHERE @ OVER -  DUP PAD C@ + PAD C! ;

: DUMP-NAMES ( a2 a1 -- )   DO  I DUP S" names " CR+$
        @-LE PRESHIFT PAD| PADCOUNT ?DUP IF  TYPE SPACE
        ELSE  DROP S" _ " TYPE  THEN  PAD|
    0 CELL+ +LOOP  CR ;
: DIS-NAMES ( a1 a2 -- )   TARGET>HOST SWAP TARGET>HOST
    DUP NEXT-CUT !  DUMP-NAMES ;
```

```
: -names ( a n -- )   2>R ['] DIS-NAMES  2R> RANGE ;
' -names RANGE: -names: ( -name- )
: -names- ( -- )   NONAME$ -names ;
' DIS-NAMES  ' -names:  ARE-COUPLED

65 CONSTANT TRIM#
VARIABLE #OUT  0 #OUT !
VARIABLE #CRS  1 #CRS !
VARIABLE capext   0 capext !
VARIABLE curcolor   0 curcolor ! \ color of current token

: CRS ( -- )   PADTYPE  #CRS @ ?DUP IF  ABS 0 DO  CR  LOOP
        #CRS @ 0< IF  S"      " PAD+  THEN  ELSE  PAD+BL  THEN ;
: |CR ( -- )   #CRS @ IF  #OUT @ IF  CR  THEN  THEN  1 #CRS ! ;
: PAD?TYPE ( -- )   PAD C@ #OUT @ + TRIM# > IF  PADTYPE |CR  THEN ;

: TRANSITION ( new -- x )   \ check against multiple transitions
    ( new    <-- )  curcolor @
    OVER 14 <>  OVER 14 =  AND IF  S"  }" PAD+ CRS THEN   \  b -> ~b
    OVER 13 <>  OVER 13 =  AND IF  S"  }" PAD+     THEN   \  g -> ~g
    OVER  9 <>  OVER  9 =  AND IF  S"  )" PAD+     THEN   \  w -> ~w
    OVER  1 <>  OVER  1 =  AND IF  S"  ]" PAD+     THEN   \  y -> ~y
    OVER  7 <>  OVER  7 =  AND IF  S"  >" PAD+     THEN   \  c -> ~c
    PAD?TYPE
    OVER  7 =   OVER  7 <> AND IF  S"  <" PAD+     THEN   \ ~c ->  c
    OVER  1 =   OVER  1 <> AND IF  S"  [" PAD+     THEN   \ ~y ->  y
    OVER  9 =   OVER  9 <> AND IF  S"  (" PAD+     THEN   \ ~w ->  w
    OVER 13 =   OVER 13 <> AND IF  S"  {" PAD+     THEN   \ ~g ->  g
    OVER 14 =   OVER 14 <> AND IF  S"  {" PAD+     THEN   \ ~b ->  b
    SWAP curcolor ! ;

: NEWC ( new -- )  ( DUP curcolor @ XOR IF ) TRANSITION ( THEN ) DROP ;

: gnn ( a -- a' n )   DUP >R  CELL+  R> @-LE ;
: n32 ( a x -- a' n )   DROP gnn ;
: n27 ( n -- n' )   2/ 2/ 2/ 2/ 2/ ;

HEX
: .NUMBER ( n -- )   DUP 1F AND
    DUP 02 = IF DROP PAD?TYPE S"  D# " PAD+ n32 (.)  PAD+ EXIT THEN \  y: execute 32-bit de
    DUP 12 = IF DROP PAD?TYPE S"  H# " PAD+ n32 (H.) PAD+ EXIT THEN \ dy: execute 32-bit he
    DUP 05 = IF DROP PAD?TYPE S"  D# " PAD+ n32 (.)  PAD+ EXIT THEN \  g: compile 32-bit de
    DUP 15 = IF DROP PAD?TYPE S"  H# " PAD+ n32 (H.) PAD+ EXIT THEN \ dg: compile 32-bit he
    DUP 06 = IF DROP PAD?TYPE S"  d# " PAD+ n27 (.)  PAD+ EXIT THEN \  g: compile 27-bit de
    DUP 16 = IF DROP PAD?TYPE S"  h# " PAD+ n27 (H.) PAD+ EXIT THEN \ dg: compile 27-bit he
    DUP 08 = IF DROP PAD?TYPE S"  d# " PAD+ n27 (.)  PAD+ EXIT THEN \  y: execute 27-bit de
```

```
    DUP 18 = IF DROP PAD?TYPE S"  h# " PAD+ n27 (H.) PAD+ EXIT THEN \ dy: execute 27-bit he
    DROP ;
DECIMAL

: 1CAP ( addr -- )   DUP C@ [CHAR] a [CHAR] z 1+ WITHIN
    IF  DUP C@ 32 - SWAP C!  ELSE  DROP  THEN ;
: CAPS ( addr len -- )   0 ?DO  DUP 1CAP 1+  LOOP  DROP ;

: .WORD ( n -- )   0 capext !  PAD+BL  PADCOUNT 2DROP ;
: .BLUE ( n -- )   0 capext !  PAD+BL  PADCOUNT
    2DUP S" cr" COMPARE 0= IF  1  ELSE
    2DUP S" br" COMPARE 0= IF  2  ELSE
    2DUP S" -cr" COMPARE 0= IF  0  ELSE
    2DUP S" indent" COMPARE 0= IF  -1  ELSE
        S"  UNKNOWN" TYPE  1
    THEN THEN THEN THEN  #CRS !  2DROP ;

: .CAPWORD ( n -- )   0 capext !  PADCOUNT DROP 1CAP ;
: .ALLCAPS ( n -- )   -1 capext !  PADCOUNT CAPS ;
: .EXTENSION ( n -- )   PADCOUNT capext @ IF  CAPS  ELSE  2DROP  THEN ;
: .COLONDEF ( -- )   PADTYPE  |CR S" :" PAD+ .WORD ;
: .VARIABLE ( -- )   S"  :#" PAD+ .WORD gnn  PAD+BL  (.) PAD+ ;
: .COMMENT# ( n -- )   S"  { " PAD+  (H.) PAD+  S"  }" PAD+ ;

HEX
: .TOKEN ( n -- )   DUP 0F AND
    DUP  0 = IF DROP         .EXTENSION EXIT THEN \ --- extension word
    DUP  1 = IF DROP  1 NEWC .WORD      EXIT THEN \ yel execute word
    DUP  2 = IF DROP  1 NEWC .NUMBER    EXIT THEN \ yel execute 32-bit
    DUP  3 = IF DROP  3 NEWC .COLONDEF  EXIT THEN \ red define word
    DUP  4 = IF DROP  4 NEWC .WORD      EXIT THEN \ gre compile word
    DUP  5 = IF DROP  4 NEWC .NUMBER    EXIT THEN \ gre compile 32-bit
    DUP  6 = IF DROP  4 NEWC .NUMBER    EXIT THEN \ gre compile 27-bit
    DUP  7 = IF DROP  7 NEWC .WORD      EXIT THEN \ cya compile a macro
    DUP  8 = IF DROP  1 NEWC .NUMBER    EXIT THEN \ yel execute 27-bit
    DUP  9 = IF DROP  9 NEWC .WORD      EXIT THEN \ whi comment word
    DUP 0A = IF DROP  9 NEWC .CAPWORD   EXIT THEN \ whi Capitalized Word
    DUP 0B = IF DROP  9 NEWC .ALLCAPS   EXIT THEN \ whi ALL CAPS WORD
    DUP 0C = IF DROP 0C NEWC .VARIABLE  EXIT THEN \ mag variable + number
    DUP 0D = IF DROP 0D NEWC PAD+BL (H.) PAD+  EXIT THEN \ gre compiler feedback
    DUP 0E = IF DROP 0E NEWC .BLUE      EXIT THEN \ blu display word
    DROP         .COMMENT# ;          \ $F commented number
DECIMAL

: ABLOCK ( a -- )   DUP 1020 + SWAP  0 curcolor !
    BEGIN  2DUP >  OVER @ 0= 0=  AND WHILE  gnn .TOKEN
```

```
    REPEAT  6 NEWC  2DROP ;   \ dummy color to mark end of block

: DUMP-BLOCKS ( a2 a1 -- )   \ display blocks ready to be translated back
    CUT-SIZE @ >R  1024 CUT-SIZE !  BASE @ >R  DECIMAL
    DO  I CODE-SPACE - 1024 /  S" D# " PAD $!  DUP S>D <# #S #> PAD $+!
        1 AND IF  S"  shadow{ "  ELSE  S"  code{ "  THEN  PAD $+!
        I PAD $@ CR+$  CR  PAD|  I ABLOCK  PADTYPE  S"  }block" TYPE  CR
    1024 +LOOP  R> BASE !  R> CUT-SIZE !  CR ;

?? ForGForth uses : GET-TYPE   'TYPE @ ;
?? ForGForth uses : SET-TYPE   'TYPE ! ;

?? ForGForth 0= uses : GET-TYPE   'TYPE >DFA @ ;
?? ForGForth 0= uses : SET-TYPE   'TYPE >DFA ! ;

: TRIM-EMIT ( c -- )   GET-TYPE  'TYPE RESTORED  SWAP
    DUP BL = IF \ #OUT @ TRIM# > IF
            \    CR  SPACE  1 #OUT !
        (    THEN ) BL EMIT  1 #OUT +!  ELSE
        DUP 10 = IF  10 EMIT  0 #OUT !  ELSE
            DUP EMIT  1 #OUT +!
    THEN THEN  DROP  SET-TYPE ;

: TRIM-TYPE ( a n -- )   0 ?DO  COUNT TRIM-EMIT  LOOP  DROP ;

?? ForGForth uses : GET-TRIM   ['] TRIM-TYPE ;
?? ForGForth 0= uses : GET-TRIM   'TRIM-TYPE >DFA @ ;

: DUMP-TRIM-BLOCKS   GET-TRIM SET-TYPE  DUMP-BLOCKS  'TYPE RESTORED ;

: DIS-BLOCKS ( a1 a2 -- )   TARGET>HOST SWAP TARGET>HOST
    DUP NEXT-CUT !  DUMP-TRIM-BLOCKS ;
: -blocks ( a n -- )   2>R ['] DIS-BLOCKS  2R> RANGE ;
' -blocks RANGE: -blocks: ( -name- )
: -blocks- ( -- )   NONAME$ -blocks ;
' DIS-BLOCKS  ' -blocks:  ARE-COUPLED

PREVIOUS
```

2.2.1 Endian handlers

The following are used to fetch data that is in known Endian format. E.g. in file system structures or network packets. These words work on un-aligned entities.

10 ⟨*endians* 10⟩≡ (5)

```
: 1C!-LE ( x a n -- )   BEGIN ?DUP WHILE
        1- ROT DUP 8 RSHIFT SWAP 2OVER DROP C! ROT 1+ ROT
    REPEAT 2DROP ;

: 1C!-BE ( x a n -- )   BEGIN ?DUP WHILE
        1- ROT DUP 8 RSHIFT SWAP 2OVER +    C! ROT    ROT
    REPEAT 2DROP ;

: 1C@-LE ( a n -- x )   0 SWAP BEGIN ?DUP WHILE
        1- ROT 2DUP +      C@ >R ROT 8 LSHIFT R> + ROT
    REPEAT  SWAP DROP ;

: 1C@-BE ( a n -- x )   0 SWAP BEGIN ?DUP WHILE
        1- ROT DUP 1+ SWAP C@ >R ROT 8 LSHIFT R> + ROT
    REPEAT  SWAP DROP ;

: SIGN-EXTEND ( x n -- x' )  32 SWAP - DUP >R
    LSHIFT  R> 0 DO  2/  LOOP ;

: @-LE ( a -- x )   4 1C@-LE ;
: !-LE ( x a -- )   4 1C!-LE ;

: W@-BE ( a -- x )   2 1C@-BE ;
: W!-BE ( x a -- )   2 1C!-BE ;
```

2.3 Restore uncompressed version

Pulled out of OkadWork.cf blocks

11 ⟨*Restore.f* 11⟩≡ (5)

```
\ Copyright (c) 2010 Dennis Ruffer

CREATE cfca 0 , \ address of compressed allocation
CREATE ebx 0 ,
CREATE ecx 0 ,

: 2*d ( n -- n )   DUP 32 ecx @ - RSHIFT  ebx @ ecx @ LSHIFT  + ebx ! ;
: 2*c ( n -- n' )   ecx @ LSHIFT ;

CREATE [na] 26 , \ bits remaining in source word
CREATE [nb] -6 , \ bits remaining in ebx
CREATE [h] 67510272 , \ destination address
CREATE [an] 0 ,
CREATE [aa] 67977026 ,
CREATE [nz] 4 ,

: NEW ( 32-bits in current word )   [aa] @ @ [an] !
    1 CELLS [aa] +!  32 [na] ! ;
: ?NEW ( fetch new word if necessary )   [na] @ 0= IF  NEW  THEN ;
: SHIFT ( n -- n ) ( into ebx, decrement nb )
    DUP NEGATE DUP [nb] +!  [na] +!  ecx !
    [an] @ 2*d 2*c [an] ! ;
: BITS ( n -- ) ( shift bits into ebx. overflow into next word )
    ?NEW DUP NEGATE [na] @ +  DUP 0< IF
        DUP >R + SHIFT NEW R> NEGATE SHIFT
    ELSE  DROP SHIFT  THEN ;

: h, ( n -- ) ( store at destination )   [h] @ !  1 CELLS [h] +! ;
: TBITS ( n n -- ) ( fill ebx with tag )   [nb] @ 8 + ecx !  2*c OR h, ;

: TZ ( n n -- n ? )   OVER [nz] !  DUP NEGATE >R + ebx @
    R> 0 DO  DUP 1 AND IF
            2DROP  UNLOOP  [nz] @ 0 EXIT
        THEN  2/
    LOOP  ebx ! DUP [nz] @ INVERT + INVERT [nb] +!  1 ;

: ?FULL ( n -- n ) ( is there room in ebx? )
    [nb] @ DUP AND DUP 0< IF
        TZ IF  EXIT  THEN
        DUP >R  4 - [nb] +!  TBITS
        0 DUP R> DUP INVERT 29 + [nb] !
    ELSE  DROP  THEN ;
```

```
: CHR ( -- n 1 | 0 ) \ examine high bits; shift 4, 5 or 7 bits
    0 ebx ! ( ?NEW )  4 BITS ebx @ 8 AND IF
        ebx @ 4 AND IF
            3 BITS 7 1 EXIT
        THEN  1 BITS 5 1 EXIT
    THEN  4 ebx @ 15 AND IF  1 EXIT
    THEN  DROP 0 ;
: CHRS ( n -- n ) \ shift characters until 0
    CHR IF  ?FULL ecx !  2*c ebx @ OR RECURSE  THEN ;
: WRD ( n -- ) \ shift characters, then tag
    28 [nb] !  DUP CHRS TBITS ;

: t, ( -- )   -4 [nb] !  ebx @ TBITS ;
: SHORT ( n -- ) ( 28-bit value+tag )   28 BITS t, ;
: 32BITS ( -- ) ( for values )   16 BITS  16 BITS  ebx @ h, ;
: LITRAL ( n -- ) \  1-bit base base, tag. value in next word
    0 ebx !  1 BITS t,  32BITS ;
: VAR ( n -- ) ( word, value )   WRD 32BITS ;

: TAG ( -- n 1 | 0 ) \ vector
    ebx @ 15 AND DUP
    DUP  0 = IF  2DROP          0 EXIT  THEN
    DUP  1 = IF   DROP  WRD     1 EXIT  THEN
    DUP  2 = IF   DROP  LITRAL  1 EXIT  THEN
    DUP  3 = IF   DROP  WRD     1 EXIT  THEN
    DUP  4 = IF   DROP  WRD     1 EXIT  THEN
    DUP  5 = IF   DROP  LITRAL  1 EXIT  THEN
    DUP  6 = IF   DROP  SHORT   1 EXIT  THEN
    DUP  7 = IF   DROP  WRD     1 EXIT  THEN
    DUP  8 = IF   DROP  SHORT   1 EXIT  THEN
    DUP  9 = IF   DROP  WRD     1 EXIT  THEN
    DUP 10 = IF   DROP  WRD     1 EXIT  THEN
    DUP 11 = IF   DROP  WRD     1 EXIT  THEN
    DUP 12 = IF   DROP  VAR     1 EXIT  THEN
    DUP 13 = IF   DROP  SHORT   1 EXIT  THEN
    DUP 14 = IF   DROP  WRD     1 EXIT  THEN
    DUP 15 = IF   DROP  SHORT   1 EXIT  THEN ;

: WRDS ( ?new -- ) \ examine tags
    4 BITS TAG IF  RECURSE  THEN ;

: BLOCKS ( blks -- bytes)   1024 * ;
: CFBLOCK ( blk -- addr)   BLOCKS CODE-SPACE + ;
: ERASEBLKS ( b n -- )   >R CFBLOCK R> BLOCKS ERASE ;
```

```
: BLOCK-RANGE ( a n n -- ) \ process each block
    OVER CFBLOCK [h] !  DUP >R ERASEBLKS  [aa] !  0 [na] !
    R> 0 DO  WRDS
        [h] @ CODE-SPACE - 1024 + -1024 AND
        CODE-SPACE + [h] !
    LOOP ;

: ns ( -- n )   18 CFBLOCK 1 CELLS + ;    \ compressed if negative
: cfc ( -- n )   CP @ CODE-SPACE - ;      \ size of compressed file
: nblk ( -- n )   18 CFBLOCK 3 CELLS + ;  \ size of uncompressed file

: RESTORE ( -- ) \ restore compressed blocks
    ns @ 0< IF  nblk @ BLOCKS CODE-LENGTH @ > ABORT" Too big!"
        36 CFBLOCK  HERE DUP cfca !  cfc 36 BLOCKS - DUP ALLOT
        MOVE  cfca @ 36 nblk @ OVER - BLOCK-RANGE
        nblk @ BLOCKS CODE-SPACE + CP !
    THEN ;
```

2.4 ColorForth assembler

Assembler support for OkadWork.cf the arrayForth for GA144-1.10

14 ⟨*cfas.f* 14⟩≡ (3b)

```
\ Copyright (c) 2010 Dennis Ruffer

FALSE CONSTANT TESTING

\ ******************************************************************************
\ stack administration * originally from gforth's grey.fs * with GNU v2 license
\ this implementation does not check overflow

\ creates a stack called word with n cells
\ the first cell is the stackpointer

TRUE CONSTANT STACK-DEBUG

: STACK ( n -- )   \ use: n stack word
    CREATE HERE CELL+ , HERE 0 ,
    SWAP CELLS ALLOT
    HERE SWAP ! ;

: STACK-CLEAR? ( stack -- f )   DUP @ [ 1 CELLS ] LITERAL - = ;

: STACK-DUMP ( stack -- )   DUP STACK-CLEAR? 0= IF
        ." >" DUP @ SWAP CELL+
        BEGIN  2DUP - WHILE  CELL+ DUP ?
        REPEAT  2DROP ." <"
    ELSE  DROP  THEN ;

: STACK-PUSH ( n stack -- )   CELL OVER +!
    DUP 2@ < ABORT" stack full"
    DUP -ROT @ !  STACK-DEBUG IF
        CR ." PUSH " STACK-DUMP
    THEN  DROP ;

: STACK-TOP ( stack -- n )   \ returns top of stack
    DUP STACK-CLEAR? ABORT" no items on stack"
    DUP @ @  SWAP STACK-DEBUG IF
        CR ." TOP  " STACK-DUMP
    THEN  DROP ;

: STACK-TOP? ( stack -- n|0 )   \ returns top or 0
    DUP STACK-CLEAR? IF  DROP 0
    ELSE  STACK-TOP  THEN ;
```

```
: STACK-POP ( stack -- )   \ discards top stack item
    DUP STACK-CLEAR? ABORT" no items on stack"
    [ -1 CELLS ] LITERAL OVER +!  STACK-DEBUG IF
        CR ." POP  " STACK-DUMP
    THEN  DROP ;

: STACK-PULL ( stack -- n )   DUP STACK-TOP SWAP STACK-POP ;

: STACK-PULL? ( stack -- n|0 )   \ pull top if available
    DUP STACK-CLEAR? IF  DROP 0
    ELSE  STACK-PULL  THEN ;

: STACK-CLEAR ( stack -- )   DUP CELL+ SWAP ! ;

?? ForGForth uses ALSO ASSEMBLER

\ ***************************************************************************
\ Support for assembling icons

VARIABLE #ICONS      \ Number of icons in the array
VARIABLE #ICON-COL   \ Index into array of icons at HERE
VARIABLE #ICON-ROW   \ Scan row within the icons we are assembling

ICON-COLUMNS 8 / CONSTANT |ICON-ROW|
ICON-ROWS 1+ |ICON-ROW| * CONSTANT |ICON-BUFFER|

: >ICON ( i -- a )   |ICON-BUFFER| * HERE 32 + + ;
: >ICON-ROW ( i r -- a )   1+ |ICON-ROW| * SWAP >ICON + ;

: SAVE-ROW ( str len -- )   2 BASE !
    0 0 2SWAP >NUMBER 2DROP DROP
    #ICON-COL @  #ICON-ROW @
    >ICON-ROW W!-BE ;

: SAVE-ICON ( str len -- )   HEX
    0 0 2SWAP >NUMBER 2DROP DROP
    #ICON-COL @ >ICON >R  #ICON-ROW @ IF
        R@ W@-BE  4 LSHIFT +  R> W!-BE  ELSE
        R@ |ICON-BUFFER| ERASE  R> W!-BE  THEN ;

: SAVE-ICONS ( -- )   HEX  #ICONS @ 0 ?DO
        I >ICON DUP W@-BE >R  |ICON-ROW| +
        |ICON-BUFFER| |ICON-ROW| - DUP R> *
        AS-HERE + SWAP MOVE
    LOOP  0 #ICONS ! ;
```

```
: NEXT-ICON-ROW ( -- )
    REFILL 0= ABORT" End of input before icons finished"
    #ICON-COL @ IF  #ICON-COL @ #ICONS !  0 #ICON-COL !  1 #ICON-ROW +!
    ELSE  SAVE-ICONS  0 #ICON-COL !  0 #ICON-ROW !
    THEN ;

: icons{ ( -- )   BASE @ >R
    0 #ICONS !  0 #ICON-COL !  0 #ICON-ROW !
    BEGIN
        BEGIN
            BEGIN  #ICON-ROW @ 2 <
            WHILE  BL WORD COUNT  DUP 0=
                WHILE  2DROP  NEXT-ICON-ROW
                REPEAT
                2DUP S" }icons" COMPARE
                IF  SAVE-ICON
                ELSE  2DROP  SAVE-ICONS  R> BASE !  EXIT
                THEN
            THEN
            BL WORD COUNT  DUP 0=
        WHILE  2DROP  NEXT-ICON-ROW
        REPEAT
        2DUP S" }icons" COMPARE
        IF  SAVE-ROW  1 #ICON-COL +!
        ELSE  2DROP  SAVE-ICONS  R> BASE !  EXIT
        THEN
    AGAIN ;

\ ***************************************************************************
\ Support for translating ASCII to colorForth

VARIABLE CFEND                    \ address of last colorForth token in block
VARIABLE NBITS                    \ number of bits left in token
VARIABLE WORDLEN  0 WORDLEN !    \ size of current ASCII word name
VARIABLE 1STCAP  TRUE 1STCAP !    \ 1st letter that can be capitalized

4 STACK SAVECOLOR    \ remember colors around extensions and comments

: /BITS ( -- )        \ reset token specific flags
    28 NBITS !  TRUE 1STCAP !
    curcolor CASE
         0 OF  SAVECOLOR STACK-TOP? ?DUP
            IF  curcolor !  THEN  ENDOF
         3 OF  4 curcolor !  ENDOF
        10 OF  9 curcolor !  ENDOF
        11 OF  9 curcolor !  ENDOF
```

```
        12 OF  1 curcolor !  ENDOF
    ENDCASE ;

: .CFPTR ( -- )   BASE @ >R  DECIMAL  CR            \ display block location
    CP @ CODE-SPACE -  1024 /MOD 3 .R  4 / 4 .R  SPACE
    R> BASE ! ;

: .AFPTR ( -- )   SOURCE DUP >R TYPE                \ display text location
    CR 27 SPACES  >IN @ DUP R@ - IF  DUP WORDLEN @ + R@ - 1-
        IF  1-  THEN  THEN  R> DROP  WORDLEN @ - SPACES
    WORDLEN @ 0 DO  S" ^" TYPE  LOOP ;

: 8H.R ( n -- )   BASE @ SWAP  HEX 0
    <#  8 0 DO  #  LOOP  #>  TYPE
    BASE ! ;

: ADDN ( n -- )   TESTING IF  DUP CP @ @-LE -    \ add a fully formed token
        IF  .CFPTR  DUP 8H.R S" <>" TYPE CP @ @-LE 8H.R SPACE .AFPTR
    THEN  THEN  CP @ !-LE  4 CP +!  /BITS ;

: ADDC ( n -- )   NBITS @ 4 + LSHIFT  curcolor @ DUP >R +    \ add text token
    ADDN  R> 0= ( 3 <?> ) IF  SAVECOLOR STACK-POP  THEN ;

: GETWORD ( -- str len )    \ get next ASCII word or abort if none left
    BEGIN  BL WORD COUNT ( 2DUP TYPE SPACE  SAVECOLOR STACK-DUMP ) DUP WORDLEN !  DUP 0=
    WHILE  2DROP REFILL 0= ABORT" end of input before block finished"
    REPEAT ;

: GETN ( -- n )   0 0 GETWORD OVER C@ [CHAR] - = DUP >R   \ get an ASCII number
    IF  1 /STRING  THEN  >NUMBER ROT OR
        IF  .CFPTR  S" Invalid number     " TYPE  .AFPTR
        THEN  DROP R>
    IF  NEGATE  THEN ;

: LARGEN ( -- color )   curcolor @ 1 = IF  2 EXIT  THEN  5 ;
: SMALLN ( -- color )   curcolor @ 1 = IF  8 EXIT  THEN  6 ;

HEX

: L/S ( hex -- n )   \ format a numeric token, adding prefix token if large
    >R GETN DUP 80000000 = OVER ABS 2* F8000000 AND OR
    IF  LARGEN R> + ADDN EXIT  THEN
    5 LSHIFT SMALLN R> + + ;

DECIMAL
```

```
: SETCOLOR ( n _ -- )   CREATE , DOES> @ curcolor ! ;   \ define format words

HEX

CREATE II-CF   HERE 60 DUP ALLOT  ERASE   \ ASCII to cf translation table

: /II-CF ( -- )   CF-II COUNT 1
    DO  I 2DUP + C@  DUP [CHAR] a [CHAR] z 1+ WITHIN
        IF  SWAP 40 +  SWAP 20 -  2DUP II-CF + C!        \ lower case alpha
            SWAP 40 +  SWAP 20 -       II-CF + C!        \ upper case alpha
        ELSE               20 -       II-CF + C!        \ non-alpha chars
        THEN
    LOOP  DROP ;

/II-CF   \ fill programatically to reduce maintenance

: CHC ( ascii -- cf )   \ convert ASCII to cf character
    DUP 80 20 WITHIN IF  .CFPTR S" invalid character  " TYPE  .AFPTR  0=
    ELSE  -20 + II-CF + C@  THEN ;

DECIMAL

: HUF ( cf -- cf huf len )   \ convert cf char to huffman, #-bits
    DUP 63 AND
    DUP 16 < IF
        DUP 8 < IF
            4 ( 0xxx 0-7 )
        ELSE  24 XOR 5 ( 10xxx 8-15 )
        THEN
    ELSE  80 + 7 ( 11xxxxx 16-47 )
    THEN ;

\ Chuck's 'trick': since he is shifting the bits left, which fills with zeros,
\ he allows the packed encoded characters to be larger than 28 bits if the
\ trailing bits are zeros.

: SHORTPACK ( n cf huf len -- cf huf' n' len' )   >R ROT R@ NBITS @ >
    IF  OVER DUP  R@ NBITS @ -  DUP >R RSHIFT  R@ LSHIFT XOR
        IF  R> DROP  ADDC  0  curcolor @ SAVECOLOR STACK-PUSH  0 curcolor !
        ELSE  SWAP R> R> OVER - >R RSHIFT SWAP  THEN
    THEN  R> ;

: FIRSTCAP ( -- )   \ process 1st capital character
    curcolor @ 9 =  curcolor @ 0= OR NOT
    IF  .CFPTR  S" Cap not in comment " TYPE  .AFPTR  THEN
    curcolor @ IF  10 curcolor !  FALSE 1STCAP !  THEN ;
```

```
: ALLCAPS ( -- )   \ process folllowing capital characters
    curcolor @ 10 =  curcolor @ 11 = OR NOT
    IF  .CFPTR  S" Not 1st/all caps   " TYPE  .AFPTR  THEN
    curcolor @ IF  11 curcolor !  THEN ;

: NOTCAP ( -- )   \ process lower case characters
    curcolor @ 11 = IF  .CFPTR  S" Must be all caps   " TYPE  .AFPTR  THEN
    FALSE 1STCAP ! ;

: PACK ( n cf huf len -- n' )   SHORTPACK   \ pack huffman characters
    DUP >R ( len )  LSHIFT + >R ( n )  DUP 128 AND
    IF  1STCAP @ IF  FIRSTCAP  ELSE  ALLCAPS  THEN
    ELSE  DUP 64 AND IF  NOTCAP  THEN
    THEN  DROP R> ( n )  R> ( len ) NEGATE NBITS +! ;

: ADDWORD ( str len -- )   \ add ASCII word to image
    0 -ROT  OVER + SWAP ?DO
        I C@ CHC HUF PACK
    LOOP  ADDC ;

: names ( -- )   4 curcolor !  SAVECOLOR STACK-CLEAR  /BITS
    BEGIN  BL WORD COUNT  DUP WORDLEN !  ?DUP WHILE
            S" _" 2OVER COMPARE 0= IF
                2DROP 0 ADDN  ELSE  ADDWORD  THEN
    REPEAT  DROP ;

: ENDBLOCK ( -- )   \ finish up the block by filling it with null tokens
    CFEND @ CP @ - 4 /MOD DUP >R 0< OR ABORT" bad pointers"
    .CFPTR S" tokens processed ok" TYPE  0 curcolor !
    R> 1+ 0 DO  0 ADDN  LOOP ;

\ ***************************************************************************
\ Translate ASCII to colorForth

GET-CURRENT ( * )   WORDLIST DUP CONSTANT AF-VOC   SET-CURRENT

: D# ( _ -- )               0 L/S ADDN ;
: H# ( _ -- )   BASE @ HEX 16 L/S ADDN BASE ! ;

: :# ( _name _value -- )   curcolor @ SAVECOLOR STACK-PUSH \ compile a variable
    CP @  12 curcolor !  GETWORD ADDWORD  CP @ SWAP - 4 -
    IF  .CFPTR  S" Var name too long  " TYPE  .AFPTR  THEN
    GETN ADDN  SAVECOLOR STACK-PULL curcolor ! ;

: }blocks ( -- ) ;  \ terminate blocks
```

```
: }block ( -- )   \ terminate block
    ENDBLOCK  -1 curcolor ! ;

: }   SAVECOLOR STACK-PULL curcolor ! ; \ restore color
: { ( _value -- )   \ compile an error, comment or display token
    curcolor @ SAVECOLOR STACK-PUSH  BASE @  HEX
    0 0 GETWORD 2DUP >R >R  OVER C@ [CHAR] - =
    DUP >R IF  1 /STRING  THEN  >NUMBER ROT OR
    IF  2DROP R> DROP  14 curcolor !  R> R> ADDWORD
    ELSE  DROP R> IF  NEGATE  THEN  ADDN  R> R> 2DROP
    THEN  BASE ! ;

3 SETCOLOR :        \ define
1 SETCOLOR [        \ execute
4 SETCOLOR ]        \ compile
7 SETCOLOR <        \ compile macro
4 SETCOLOR >        \ compile

: )   SAVECOLOR STACK-PULL curcolor ! ;                \ restore color
: (   curcolor @ SAVECOLOR STACK-PUSH  9 curcolor ! ;    \ comment

( * ) SET-CURRENT

: +TOKEN ( str len -- )   \ process ASCII word
    2DUP AF-VOC SEARCH-WORDLIST ( 2 <?> )
    IF  EXECUTE  2DROP  EXIT  THEN
    ADDWORD ;

: CBLOCK ( blk# -- )   \ process ASCII into a block
    CFBLOCK DUP CP !  1020 + CFEND !
    4 curcolor !  SAVECOLOR STACK-CLEAR  /BITS
    BEGIN  curcolor @ 0< NOT  WHILE
        CFEND CP @ > NOT
        IF  .CFPTR  S" Block too long     " TYPE  .AFPTR
            BEGIN  GETWORD S" }block" COMPARE 0=
            UNTIL  -1 curcolor !
        ELSE ( 0 <?> ) GETWORD ( 1 <?> ) +TOKEN  THEN
    REPEAT ;

: code{ ( n -- )   DUP 1 AND   \ start code block
    IF  CR  8 SPACES S" Odd code block     " TYPE
        5 WORDLEN !  .AFPTR  THEN
    CBLOCK ;

: shadow{ ( n -- )   DUP 1 AND 0=   \ start shadow block
    IF  CR  8 SPACES S" Even shadow block  " TYPE
```

```
        7 WORDLEN !  .AFPTR  THEN
    CBLOCK ;

PREVIOUS
```

3 Computer Intelligence Assembler/Disassembler

Albert van der Horst, of the HCC FIG Holland, wrote **ciasdis** between 2000-2008 and released it with a GNU Public License. He wrote it based on **lina** 4.0.6 and I extended it to work with **gforth** 0.7.9. My documentation will be included here as Literate Programming, but Albert's overview was:

> "From day one the reverse engineering assembler had the property that disassembly was based on the same tables as assembly, and that disassembled binaries, could be reassembled to the exact same binary. This is now complemented by consult files that supply the disassembler with information to generate a readable and documented source with label names. Consult files can be built up incrementally."

3.1 License

3.2 Main Build

3.2.1 Main Load

Albert wrote this to be used as Linux executables (**cias** and **cidis**), but I use it in Windows 7 in source form with **gforth** running under **cygwin**. In reality, the two executbles are identical, but the 'd' in the name triggers the disassembler behavior. For ColorForth, I split this into 2 parts so I could load the ColorForth portions in the middle.

21 ⟨*ciasdis.frt* 21⟩≡
⟨*load-ciasdis* 22⟩
⟨*run-ciasdis* 23⟩

Load the ciasdis support files:

22 ⟨*load-ciasdis* 22⟩≡ (3b 21)

```
( $Id: ciasdis.frt,v 1.26 2009/03/26 19:40:39 albert Exp $ )
( Copyright{2000}: Albert van der Horst, HCC FIG Holland by GNU Public License)
( Uses Richard Stallmans convention. Uppercased word are parameters.    )

CR \ Start compile time output on a new line

⟨compat.f 32⟩
⟨noname 27f⟩

[DEFINED] ForSwiftForth
[DEFINED] ForGForth OR [IF]
    ⟨GET-FILE 27g⟩
    ⟨PUT-FILE 28a⟩
[THEN]
[DEFINED] ForCiForth [IF]
    REQUIRE OLD:    REQUIRE $=    REQUIRE class    REQUIRE W/O
    ⟨NEW-PRESENT 28b⟩
    ⟨FAR-at 28c⟩
[THEN]

\ ----------------------------------------------------------------
⟨tools.frt 33⟩
⟨asgen.frt 42⟩
⟨aswrap.frt 68a⟩
⟨asi386.frt 72⟩

\ Tools
⟨access.frt 79d⟩
⟨labelas.frt 80a⟩
⟨labeldis.frt 85⟩
⟨crawl.frt 114⟩
```

This second part defines the run time behavior of ciasdis:

23 ⟨*run-ciasdis* 23⟩≡ (3b 21)

```
[DEFINED] ForCiForth [IF]
    REQUIRE #-PREFIX    \ In behalf of user.
    REQUIRE ARGC    \ In behalf of building an executable.
[THEN]

⟨WRITE-ONE-SECTION 24a⟩
⟨WRITE-SECTIONS 24b⟩
⟨OPEN-IT 24c⟩
⟨CLOSE-IT 24d⟩

[DEFINED] ForSwiftForth
[DEFINED] ForGForth OR [IF]
    ⟨cf-SOURCE-AS 26a⟩
    ⟨cf-TARGET-AS 26b⟩
[THEN]
[DEFINED] ForCiForth [IF]
    ⟨ci-SOURCE-AS 26e⟩
    ⟨ci-TARGET-AS 26f⟩
[THEN]

⟨WRITE-IT 24e⟩
⟨PARSE-ASM 25a⟩
⟨[ASM 25b⟩
⟨cias 25c⟩
⟨FETCHED 25d⟩
⟨FETCH 25e⟩

[DEFINED] ForSwiftForth
[DEFINED] ForGForth OR [IF]
    ⟨cf-TARGET-DIS 26c⟩
[THEN]
[DEFINED] ForCiForth [IF]
    ⟨ci-TARGET-DIS 26g⟩
    REQUIRE DUMP
[THEN]

⟨CONSULTED 25f⟩
⟨CONSULT 25g⟩

[DEFINED] ForSwiftForth
[DEFINED] ForGForth OR [IF]
    ⟨cfdis 26d⟩
[THEN]
[DEFINED] ForCiForth [IF]
```

```
    ⟨cidis 26h⟩
[THEN]

⟨RESTORE-ALL 25h⟩

[DEFINED] ForCiForth [IF]
    ⟨INTERACTIVE 26i⟩
    ⟨HANDLE-ARG 27a⟩
    ⟨CONTAINS-D? 27b⟩
    ⟨BLOCK-FILE 27c⟩
    ⟨SIGNON 27d⟩
    ⟨MAIN 27e⟩
[THEN]
```

Write current section to FILEHANDLE. Leave FILEHANDLE.

24a ⟨*WRITE-ONE-SECTION* 24a⟩≡ (23)

```
: WRITE-ONE-SECTION ( handle -- handle )
    >R  FILE-OFFSET 0 R@ REPOSITION-FILE THROW
    CODE-SPACE CP @ OVER - R@ WRITE-FILE THROW  R> ;
```

Write all sections to FILEHANDLE. Leave FILEHANDLE.

24b ⟨*WRITE-SECTIONS* 24b⟩≡ (23)

```
: WRITE-SECTIONS ( handle -- handle )   SECTION-REGISTRY DO-BAG
        I @ TO CURRENT-SECTION  WRITE-ONE-SECTION
    LOOP-BAG ;
```

Open NAME, return FILEHANDLE.

24c ⟨*OPEN-IT* 24c⟩≡ (23)

```
: OPEN-IT ( a n -- handle )   R/W CREATE-FILE THROW ;
```

Close FILEHANDLE.

24d ⟨*CLOSE-IT* 24d⟩≡ (23)

```
: CLOSE-IT ( handle -- )   CLOSE-FILE THROW ;
```

Write all sections to file NAME.

24e ⟨*WRITE-IT* 24e⟩≡ (23)

```
: WRITE-IT ( a n -- )   OPEN-IT  WRITE-SECTIONS  CLOSE-IT ;
```

Assembler parser.

25a ⟨*PARSE-ASM* 25a⟩≡ (23)
```
: PARSE-ASM ( -- )
    BEGIN BEGIN  BL WORD DUP C@  WHILE  FIND IF  EXECUTE  ELSE
                    COUNT 2DUP S" ASM]" $= IF  2DROP EXIT
                    THEN  OVER C@ [CHAR] : = IF  1 /STRING
                        KNOWN-LABEL? IF  2DROP  ELSE
                            _AP_ -ROT LABELED
                        THEN ELSE  0 0 2SWAP >NUMBER 2DROP DROP
                THEN THEN REPEAT  DROP REFILL 0= UNTIL ;
```

Parse assembler until **ASM]**.

25b ⟨*[ASM* 25b⟩≡ (23)
```
: [ASM ( -- )   BASE @ >R  GET-ORDER
    POSTPONE ONLY POSTPONE FORTH POSTPONE ALSO POSTPONE ASSEMBLER
    SAVE-INPUT  FIRSTPASS  2 0 DO  DEPTH >R  PARSE-ASM  DEPTH R> -
        IF  .S  TRUE ABORT" Stack depth error"  THEN
        I 0= IF  SECONDPASS  RESTORE-INPUT THROW  THEN
    LOOP  SET-ORDER  R> BASE ! ;
```

Perform the action of the program as per the spec's of '**cias**'.

25c ⟨*cias* 25c⟩≡ (23)
```
: cias ( -- )   SOURCE-AS INCLUDED   TARGET-AS WRITE-IT ;
```

Fetch file NAME to the code buffer.

25d ⟨*FETCHED* 25d⟩≡ (23)
```
: FETCHED ( a n -- )   GET-FILE DUP CODE-LENGTH @ > ABORT" Too big!"
    CODE-SPACE SWAP  2DUP + CP !  MOVE ;
```

Fetch file "name" to the code buffer.

25e ⟨*FETCH* 25e⟩≡ (23)
```
: FETCH ( -- )   BL WORD COUNT FETCHED ;
```

Using (only) information from file with NAME, disassemble the current program as stored in the '**CODE-BUFFER**'.

25f ⟨*CONSULTED* 25f⟩≡ (23)
```
: CONSULTED ( a n -- )   INIT-ALL  HEX  INCLUDED ( file)  SORT-ALL
    PLUG-HOLES  ALL-L-LABELS  DISASSEMBLE-TARGET  DECIMAL ;
```

Consult "file" as per '**CONSULT**'.

25g ⟨*CONSULT* 25g⟩≡ (23)
```
: CONSULT ( -- )   BL WORD COUNT CONSULTED ;
```

Restore all revectoring done while compiling to stand alone.

25h ⟨*RESTORE-ALL* 25h⟩≡ (23)
```
: RESTORE-ALL ( -- )   '?ERROR RESTORED  'SECTION RESTORED
    'TYPE RESTORED ;
RESTORE-ALL
```

ANS Forth specific

Return the NAME of the source file.

26a ⟨*cf-SOURCE-AS* 26a⟩≡ (23)
```
: SOURCE-AS ( -- a n )   BL WORD COUNT ;
```

Return the NAME of the target file.

26b ⟨*cf-TARGET-AS* 26b⟩≡ (23)
```
: TARGET-AS ( -- a n )   BL WORD COUNT ;
```

Return the NAME of the target file.

26c ⟨*cf-TARGET-DIS* 26c⟩≡ (23)
```
: TARGET-DIS ( -- a n )   BL WORD COUNT ;
```

Perform the action of the program as per the spec's of '**cidis**'.

26d ⟨*cfdis* 26d⟩≡ (23)
```
: cidis ( -- )   BL WORD COUNT FETCHED TARGET-DIS CONSULTED ;
```

ciForth specific

Return the NAME of the source file.

26e ⟨*ci-SOURCE-AS* 26e⟩≡ (23)
```
: SOURCE-AS ( -- a n )   1 ARG[] ;
```

Return the NAME of the target file.

26f ⟨*ci-TARGET-AS* 26f⟩≡ (23)
```
: TARGET-AS ( -- a n )   2 ARG[] ;
```

Return the NAME of the target file.

26g ⟨*ci-TARGET-DIS* 26g⟩≡ (23)
```
: TARGET-DIS ( -- a n )   2 ARG[] ;
```

Perform the action of the program as per the spec's of '**cidis**'.

26h ⟨*cidis* 26h⟩≡ (23)
```
: cidis ( -- )   1 ARG[] FETCHED TARGET-DIS CONSULTED ;
```

Start an interactive session or a filter. The startup code has changed '**OK**' for a filter. In that case suppress the splat screen. Note that '**QUIT**' is the command interpreter.

26i ⟨*INTERACTIVE* 26i⟩≡ (23)
```
: INTERACTIVE    'OK DUP >DFA @ SWAP >PHA = IF 0 LIST OK THEN
    ASSEMBLER   0 ORG   QUIT ;
```

Handle arguments, start interactive system if no arguments. The second argument is still obligatory for the moment.

27a ⟨*HANDLE-ARG* 27a⟩≡ (23)
```
: HANDLE-ARG   ARGC 1 = IF INTERACTIVE THEN
    ARGC ( 2) 3 4 WITHIN 0= 13 ?ERROR ;
```

For **`STRING:`** "It CONTAINS a 'd' or a 'D' "

27b ⟨*CONTAINS-D?* 27b⟩≡ (23)
```
: CONTAINS-D?    2DUP &D $I >R  &d $I R>  OR ;
```

Fetch the library file from the current directory. We can't assume **`lina`** has been installed so **`forth.lab`** is supplied with the **`ciasdis`** program.

27c ⟨*BLOCK-FILE* 27c⟩≡ (23)
```
"forth.lab" BLOCK-FILE $!
```

Make a cold start silent.

27d ⟨*SIGNON* 27d⟩≡ (23)
```
'TASK >DFA @   '.SIGNON >DFA !
```

The name determines what to do.

27e ⟨*MAIN* 27e⟩≡ (23)
```
: MAIN   RESTORE-ALL  DEFAULT-SEGMENT HANDLE-ARG
    0 ARG[] CONTAINS-D? IF cidis ELSE cias THEN ;
```

3.2.2 Disgraceful adaptations

Put here to draw attention.

This name might later be changed.

27f ⟨*noname* 27f⟩≡ (22)
```
: NONAME$ ( -- a n )   s" NONAME" ;
```

ANS Forth specific

27g ⟨*GET-FILE* 27g⟩≡ (22)
```
: GET-FILE ( a1 n1 -- a2 n2 )
    r/o open-file throw
    dup >r file-size throw
    abort" file too large"
    dup allocate throw
    swap 2dup r@ read-file throw
    over <> abort" could not read whole file"
    r> close-file throw ;
```

28a ⟨*PUT-FILE* 28a⟩≡ (22)
```
: PUT-FILE ( a1 n1 a2 n2 -- )
    r/w create-file throw
    dup >r write-file throw
    r> close-file throw ;
```

ciForth specific

Patch the word '**PRESENT**' such that no name words are no longer considered present. This prevents a zillion error messages.

28b ⟨*NEW-PRESENT* 28b⟩≡ (22)
```
: NEW-PRESENT   OLD: PRESENT DUP IF DUP >NFA @ $@ NONAME$ $= 0= AND THEN ;
' NEW-PRESENT ' PRESENT 3 CELLS MOVE
```

Patch the word '**L@**' with the name '**FAR@**'. Such that it no longer conflicts with the '**L@**' we have.

28c ⟨*FAR-at* 28c⟩≡ (22)
```
: FAR@ L@ ;    HIDE L@
```

3.3 Compatibility

I have created similar compatibility test systems before, but Josh Grams (**josh@qualdan.com**) created this one, which is smaller and covers more systems than I had even envisioned earlier. He wrote this code to detect forth system and use appropriate prelude, between 2009 and 2010-03-09. He gifted it to the public domain. Specifically, you may use, modify, and redistribute it without limitation, but it comes with ABSOLUTELY NO WARRANTY.

Currently, it detects the following systems, but he had not tested **iForth** or **VFX Forth** and I have only tested **gforth** and **ciforth**:

bigForth http://www.jwdt.com/~paysan/bigforth.html

28d ⟨*ForBigForth* 28d⟩≡ (32)
```
\ bigforth compat.f -e bye
S" BIGFORTH" ENVIRONMENT? env-str?
    uses CREATE ForBigForth
```

ciforth http://home.hccnet.nl/a.w.m.van.der.horst/ciforth.html

28e ⟨*ForCiForth* 28e⟩≡ (32)
```
\ lina -s compat.f
S" NAME" ENVIRONMENT? S" ciforth" env-str=
    uses CREATE ForCiForth
```

FICL http://ficl.sourceforge.net

28f ⟨*ForFicl* 28f⟩≡ (32)
```
\ ficl compat.f
\ Grrr.  FICL has no way to quit automatically after running a script.
S" ficl-version" ENVIRONMENT? env-str?
    uses CREATE ForFicl
```

gforth http://www.jwdt.com/~paysan/gforth.html

29a ⟨*ForGForth* 29a⟩≡ (32)
```
\ gforth compat.f -e bye
S" gforth" ENVIRONMENT? env-str?
    uses CREATE ForGForth
```

iForth http://home.iae.nl/users/mhx/i4faq.html

29b ⟨*ForIForth* 29b⟩≡ (32)
```
\ ???
S" IFORTH" ENVIRONMENT? env-flag?
     uses CREATE ForIForth
```

kForth http://ccreweb.org/software/kforth/kforth.html

29c ⟨*ForKForth* 29c⟩≡ (32)
```
\ kforth compat.f -e bye
?? NONDEFERRED
    uses CREATE ForKForth
```

PFE http://pfe.sourceforge.net

29d ⟨*ForPfe* 29d⟩≡ (32)
```
\ pfe -q -y compat.f
S" FORTH-NAME" ENVIRONMENT? S" pfe" env-str=
    uses CREATE ForPfe
```

pForth http://www.softsynth.com/pforth

29e ⟨*ForPForth* 29e⟩≡ (32)
```
\ pforth compat.f
?? ::::loadp4th.fth
    uses CREATE ForPForth
```

SP-Forth http://spf.sourceforge.net

29f ⟨*ForSpf4* 29f⟩≡ (32)
```
\ spf4 compat.f BYE
S" FORTH-SYS" ENVIRONMENT? S" SP-FORTH" env-str=
    uses CREATE ForSpf4
        \ REQUIRE CASE-INS lib/ext/caseins.f
```

VFX Forth http://www.mpeforth.com

29g ⟨*ForVfx* 29g⟩≡ (32)
```
\ ??? vfx include compat.f bye ???
?? VFXFORTH
    uses CREATE ForVfx
```

Win32Forth `http://win32forth.sourceforge.net`

30a ⟨*ForWin32Forth* 30a⟩≡ (32)
```
\ win32forth include compat.f bye
S" WIN32FORTH" ENVIRONMENT? env-str?
    uses CREATE ForWin32Forth
```

SwiftForth `http://forth.com/swiftforth`

30b ⟨*ForSwiftForth* 30b⟩≡ (32)
```
\ sf include compat.f bye
?? VERSION DUP uses S" SwiftForth" version over compare 0= AND
    uses CREATE ForSwiftForth
```

Other systems to consider, but most are not standard:

4p `http://maschenwerk.de`

dsForth `http://www.delosoft.com`

FINA `http://code.google.com/p/fina-forth`

hForth `http://www.taygeta.com/hforth.html`

IsForth `http://isforth.com`

NTF/LXF `http://falth.homelinux.net/readme2.html`

MinForth `http://falth.homelinux.net/readme2.html`

RetroForth `http://retroforth.org`

3.3.1 Rationale

This eventually came to be fairly simple, but it took him several iterations to get it "right", so he published it in case it may save someone else some work.

A good test must:

- Reliably detect the current system: some free systems don't provide a way to test for this, so we have to get creative.
- Run out of the box on all other systems. Many useful words aren't in the CORE wordset and hence aren't provided by even all standard systems. Also, most so-called standard systems have words which are implemented in a sub-standard fashion.

3.3.2 Helper Words

Process the rest of line only if **`flag`** is true. **`[IF] .. [THEN]`** don't exist everywhere, so we use this conditional comment word instead.

30c ⟨*uses* 30c⟩≡ (32)
```
: uses ( flag -- )  0= IF POSTPONE \ THEN ;
```

[DEFINED] doesn't exist everywhere, and on some systems the interpreter uses **WORD**, so **BL WORD FIND** doesn't work in the interpreter.

31a ⟨*??* 31a⟩≡ (32)
```
: ?? ( "name" -- flag )  BL WORD FIND SWAP DROP 0= 0= ;
```

We need a string comparison word, but on ciforth, **COMPARE** is not loaded by default.

31b ⟨*STR=* 31b⟩≡ (32)
```
?? STR= 0= uses : MATCH? >R COUNT ROT COUNT ROT = R> AND ;
?? STR= 0= uses : (STR=) BEGIN DUP 0 > WHILE MATCH? 1- REPEAT 0= ;
?? STR= 0= uses : STR= ROT 2DUP = >R MIN (STR=) >R 2DROP R> R> AND ;
```

Dummy **ENVIRONMENT?** (since kForth doesn't have it)

31c ⟨*ENVIRONMENT?* 31c⟩≡ (32)
```
?? ENVIRONMENT? 0= uses : ENVIRONMENT? 2DROP 0 ;
```

The return from **ENVIRONMENT?** is a bit of a nuisance, so we have words to deal with various possibilities and return a single flag.

31d ⟨*env-words* 31d⟩≡ (32)
```
: env-flag?  DUP IF DROP THEN ;              \ flag true?
: env-str?  DUP IF >R 2DROP R> THEN ;        \ string present?
: env-str=  ROT IF STR= ELSE 2DROP 0 THEN ;  \ string matches?
```

3.3.3 System Tests

Originally Josh defined a word to check for the presence of each system. But Win32Forth doesn't like it if you compile **ENVIRONMENT?** into a word (apparently it isn't available in turnkey applications). He could have gotten around that by precomputing the values and defining constants. But then he realized that there was no sense in cluttering up the dictionary with all those words. So this version just uses interpreted tests. He recommend that your system-specific preludes define a no-op word named after the Forth system. Then you can later do something like this:

```
[defined] ciforth [if] ... [then]
```

One reason *for* defining a test word for each system was that ANS CORE doesn't require **S"** to work in interpretation state. But the **FILE** wordset does, and since this is in a file... At any rate, all systems which I have tested have an interpreted **S"**. If you add a system which doesn't, you could replace **S"** with something like:

```
: PAD" ( -- c-addr u ) [CHAR] " WORD COUNT PAD PLACE PAD COUNT ;
```

Each system has a comment describing how to invoke the system for testing purposes, the test itself, and the '**uses**' line which gives an example of how to include another source file (some systems are case-sensitive, some only define **INCLUDED**, and so on).

32 ⟨*compat.f* 32⟩≡ (22)

```
⟨uses 30c⟩
⟨?? 31a⟩
⟨STR= 31b⟩
⟨ENVIRONMENT? 31c⟩
⟨env-words 31d⟩

⟨ForBigForth 28d⟩
⟨ForCiForth 28e⟩
⟨ForFicl 28f⟩
⟨ForGForth 29a⟩
⟨ForIForth 29b⟩
⟨ForKForth 29c⟩
⟨ForPfe 29d⟩
⟨ForPForth 29e⟩
⟨ForSpf4 29f⟩
⟨ForVfx 29g⟩
⟨ForWin32Forth 30a⟩
⟨ForSwiftForth 30b⟩
```

3.4 Tools

Those auxiliary thingies that are not appropriate elsewhere.

33 ⟨*tools.frt* 33⟩≡ (22)

```
( $Id: tools.frt,v 1.2 2005/01/04 23:23:48 albert Exp $ )
( Copyright{2000}: Albert van der Horst, HCC FIG Holland by GNU Public License)
( Uses Richard Stallmans convention. Uppercased word are parameters.    )

[DEFINED] ForGForth [IF]  WARNINGS OFF  [THEN]

⟨display 34a⟩

[DEFINED] ForSwiftForth
[DEFINED] ForGForth OR [IF]
     : NOT ( flag -- flag' )   0= ;

     ⟨string-primitives 35a⟩
     ⟨H. 35b⟩
     ⟨num-out 34c⟩

      : CORA swap over compare ;
      : ETYPE TYPE ;
      : TOGGLE ( a b -- ) over @ xor swap ! ;
      variable exit-code
      0 value _  ' _ to _

     ⟨BAGS 36a⟩
     ⟨BIN-SEARCH 38a⟩
     ⟨EXCHANGE 40d⟩
     ⟨QSORT 38c⟩
[ELSE]
    REQUIRE H.    REQUIRE RESTORED
     ⟨.NFA 40e⟩
[THEN]

⟨debug 40a⟩
⟨QSORT-SAFE 38b⟩
⟨INVENTED-NAMES 39e⟩
⟨=<>= 40b⟩
⟨?ABORT 40c⟩

[DEFINED] ForCiForth [IF]
    REQUIRE $=    REQUIRE ."$"
[THEN]
```

3.4.1 Vectored display

To enable output to be turned off, or sent to a file.

34a ⟨*display* 34a⟩≡ (33)

```
CREATE 'TYPE   ' TYPE DUP , ,

: TYPE ( a n -- )   'TYPE @ EXECUTE ;

: RESTORED ( a -- )   DUP CELL+ @  SWAP ! ;

⟨SHUTUP 34b⟩

[DEFINED] ForSwiftForth [IF]  true constant ForDOS  [THEN]
[DEFINED] ForGForth [IF] \ Add missing GForth definitions
s" PWD" getenv drop c@ char \ = [IF]    \ using DOS path separators
     true constant ForDOS [THEN] [THEN]

[DEFINED] ForDOS [IF]
     CREATE <EOL>   2 C, 13 C, 10 C, \ We are assuming DOS line terminators
[ELSE]
     CREATE <EOL>   1 C, 10 C,       \ We are assuming UNIX line terminators
[THEN]

CREATE CHAR-BUF   0 C,

: CR ( -- )         'TYPE 2@ = IF  CR  ELSE  <EOL> COUNT TYPE  THEN ;
: EMIT ( c -- )     'TYPE 2@ = IF  EMIT  ELSE
         CHAR-BUF C!  CHAR-BUF 1 TYPE  THEN ;
: SPACE ( -- )      'TYPE 2@ = IF  SPACE  ELSE  BL EMIT  THEN ;
: SPACES ( n -- )   'TYPE 2@ = IF  SPACES  ELSE  0 ?DO  SPACE  LOOP  THEN ;
```

Make the output disappear till the end of the calling word.

34b ⟨*SHUTUP* 34b⟩≡ (34a)

```
: 2DROP'  2DROP ;      \ Need a high level word here.
: (SHUTUP) ( xt -- )   ['] 2DROP' 'TYPE !  EXECUTE  'TYPE RESTORED ;
: SHUTUP ( -name- )   ' POSTPONE LITERAL  POSTPONE (SHUTUP) ; IMMEDIATE
```

ANS Forth specific

34c ⟨*num-out* 34c⟩≡ (33)

```
: C.   S>D <# # #S #> TYPE SPACE ; ( print byte )
: (.)   S>D TUCK DABS <# #S ROT SIGN #> ;
: .   (.) TYPE SPACE ;
: .R   >R (.) R> OVER - 0 MAX SPACES TYPE ;
```

3.4.2 String handling

ANS Forth specific

35a ⟨*string-primitives* 35a⟩≡ (33)

```
: cappend ( char to -- )   DUP >R COUNT + C! R@ C@ 1+ R> C! ;
: append ( from len to -- )   2DUP >R >R  COUNT + SWAP MOVE
    R> R@ C@ + R> C! ;
: place ( from len to -- )   0 OVER C! SWAP 255 MIN SWAP APPEND ;

: @+ ( a -- a' n )   DUP >R CELL+ R> @ ;
: $= ( a1 n1 a2 n2 -- f )   COMPARE 0= ;
: $@ ( a -- a' n )   COUNT ;
: $, ( a n -- a' )   HERE >R  DUP C,  0 ?DO  $@ C,  LOOP  DROP R> ;
: $! ( a n a' -- )   PLACE ;
: $+! ( a n a' -- )   APPEND ;
: $C+ ( c a -- )   CAPPEND ;
```

3.4.3 Hex output

ANS Forth specific Print SINGLE in hex.

35b ⟨*H.* 35b⟩≡ (33)

```
VARIABLE BASE'
: <HEX   BASE @ BASE' ! HEX ;       ( 0/1  SWITCH TO HEX)
: HEX>   BASE' @ BASE !     ;       ( 1/0  AND BACK)

[DEFINED] ForCiForth [IF]
⟨4? 35c⟩
    : (DH.)   <HEX  <#  1- 0 ?DO  #  I 4?  LOOP  #  #>  HEX> ;
[ELSE]
    : (DH.)   <HEX  <#  1- 0 ?DO  #  LOOP  #  #>  HEX> ;
[THEN]
: (H.)   S>D  2 CELLS (DH.) ;
: H.   (H.) TYPE ;
```

ciForth specific

In lina, Albert adds a , after 4 digits.

35c ⟨*4?* 35c⟩≡ (35b)

```
: 4? ( n -- )   1+ 4 MOD 0= IF  [CHAR] , HOLD THEN ;
```

3.4.4 Bags of stacks

36a ⟨*BAGS* 36a⟩≡ (33)
⟨*BUILD-BAG* 36b⟩
⟨*BAG* 36c⟩
⟨*!BAG* 36d⟩
⟨*BAG?* 36e⟩
⟨*BAG+!* 36f⟩
⟨*BAG@-* 36g⟩
⟨*BAG-REMOVE* 37a⟩
⟨*BAG-HOLE* 37b⟩
⟨*BAG-INSERT* 37c⟩
⟨*|BAG|* 37d⟩
⟨*BAG-LOOP* 37e⟩
⟨*.BAG* 37f⟩
⟨*BAG-WHERE* 37g⟩
⟨*IN-BAG?* 37h⟩
⟨*BAG-* 37i⟩
⟨*SET* 37j⟩

Build a bag (i.e. stack) with X items.

36b ⟨*BUILD-BAG* 36b⟩≡ (36a)
```
: BUILD-BAG ( n -- )   HERE CELL+ , CELLS ALLOT ;
```

Create a bag "x" with X items.

36c ⟨*BAG* 36c⟩≡ (36a)
```
: BAG ( n -- )   CREATE HERE CELL+ , CELLS ALLOT DOES> ;
```

Make the BAG empty.

36d ⟨*!BAG* 36d⟩≡ (36a)
```
: !BAG ( bag -- )   DUP CELL+ SWAP ! ;
```

For the BAG : it IS non-empty.

36e ⟨*BAG?* 36e⟩≡ (36a)
```
: BAG? ( bag -- )   @+ = 0= ;
```

Push ITEM to the BAG

36f ⟨*BAG+!* 36f⟩≡ (36a)
```
: BAG+! ( x bag -- )   DUP >R @ ! 0 CELL+ R> +! ;
```

From BAG: pop ITEM

36g ⟨*BAG@-* 36g⟩≡ (36a)
```
: BAG@- ( bag -- x )   0 CELL+ NEGATE OVER +! @ @ ;
```

Remove entry at ADDRESS from BAG.

37a ⟨*BAG-REMOVE* 37a⟩≡ (36a)
```
: BAG-REMOVE ( a bag -- )
    >R  DUP CELL+ SWAP  OVER R@ @ SWAP - MOVE  -1 CELLS R> +! ;
```

Make hole at ADDRESS in BAG.

37b ⟨*BAG-HOLE* 37b⟩≡ (36a)
```
: BAG-HOLE ( a bag -- )
    >R  DUP CELL+   OVER R@ @ SWAP - MOVE   0 CELL+ R> +! ;
```

Insert VALUE at ADDRESS in BAG.

37c ⟨*BAG-INSERT* 37c⟩≡ (36a)
```
: BAG-INSERT ( x a bag -- )   OVER SWAP BAG-HOLE ! ;
```

For BAG : NUMBER of items.

37d ⟨|*BAG*| 37d⟩≡ (36a)
```
: |BAG| ( bag -- n )   @+ SWAP - 0 CELL+ / ;
```

Loop over a bag, see '**.BAG**' for an example.

37e ⟨*BAG-LOOP* 37e⟩≡ (36a)
```
: DO-BAG  POSTPONE @+ POSTPONE SWAP POSTPONE ?DO ; IMMEDIATE
: LOOP-BAG 0 CELL+ POSTPONE LITERAL POSTPONE +LOOP ; IMMEDIATE
```

Print BAG.

37f ⟨*.BAG* 37f⟩≡ (36a)
```
: .BAG ( bag -- )   DO-BAG  I ?  LOOP-BAG ;
```

For VALUE and BAG : ADDRESS of value in bag/nill.

37g ⟨*BAG-WHERE* 37g⟩≡ (36a)
```
: BAG-WHERE ( x bag -- a )   DO-BAG  DUP I @ = IF
        DROP I UNLOOP EXIT  THEN
    LOOP-BAG  DROP 0 ;
```

For VALUE and BAG : value IS present in bag.

37h ⟨*IN-BAG?* 37h⟩≡ (36a)
```
: IN-BAG? ( x bag -- )   BAG-WHERE 0= 0= ;
```

Remove VALUE from BAG.

37i ⟨*BAG-* 37i⟩≡ (36a)
```
: BAG- ( x bag -- )   DUP >R   BAG-WHERE   R> BAG-REMOVE ;
```

Add/remove VALUE to bag, used as a SET, i.e. no duplicates.

37j ⟨*SET* 37j⟩≡ (36a)
```
: SET+ ( x bag -- )   2DUP IN-BAG? IF 2DROP ELSE BAG+! THEN ;
: SET- ( x bag -- )   2DUP IN-BAG? IF BAG- ELSE 2DROP THEN ;
```

3.4.5 Bin search

Uses a comparison routine with execution token 'COMP'. 'COMP' must have the stack diagram (IT -- flag) , where flag typically means that IT compares lower or equal to some fixed value. It may be TRUE , FALSE or undefined for 'IMIN' , but it must be monotonic down in the range [IMIN,IMAX), i.e. if IMIN<=IX<=IY<IMAX then if IX COMP gives false, IY COMP cannot give true.

BIN-SEARCH finds the first index 'IT' between 'IMIN' and 'IMAX' (exclusive) for which 'COMP' returns false or else 'IMAX'. An empty range is possible, (e.g. 'IMIN' and 'IMAX' are equal).

38a ⟨*BIN-SEARCH* 38a⟩≡ (33)

```
: BIN-SEARCH ( n IMIN, n IMAX, xt COMP -- n IRES )   >R
    BEGIN       \ Loop variant IMAX - IMIN
        2DUP  <> WHILE
            2DUP + 2/  ( -- ihalf )
            DUP R@ EXECUTE IF
                1+  SWAP ROT DROP \ Replace IMIN
            ELSE
                SWAP DROP \ Replace IMAX
            THEN
    REPEAT
    R> 2DROP ;
```

3.4.6 Quick sort

Make QSORT safe by allowing an empty range. Not tested and maybe not necessary.

38b ⟨*QSORT-SAFE* 38b⟩≡ (33)

```
\ : QSORT-SAFE 2>R 2DUP < IF 2R> QSORT ELSE 2DROP 2R> 2DROP THEN ;
```

ANS Forth specific

The **QSORT** facility is part of **lina**, so it must be added to ANS Forths.

Sort the range FIRST to LAST (inclusive) of item compared by the xt COMPARING and exchanged by the xt EXHANGING. All indices in this range must be proper to pass to the xt's. The xt's are filled in into ***<*** and ***<-->*** and must observe the interface. After the call we have that : For FIRST<=I<J<=LAST I J *<--> EXECUTE leaves TRUE.

38c ⟨*QSORT* 38c⟩≡ (33)

```
( QSORT ) \ AvdH A2apr22

\ Compare item N1 and N2. Return 'N1' IS lower and not equal.
VARIABLE *<
\ Exchange item N1 and N2.
VARIABLE *<-->
⟨PARTITION 39a⟩
⟨(QSORT) 39b⟩

: QSORT ( xt1 xt2 -- )   *<--> !  *< !  (QSORT) ;
```

Partition inclusive range LO HI leaving LO_1 HI_1 LO_2 HI_2.

39a ⟨*PARTITION* 39a⟩≡ (38c)
```
: PARTITION   2DUP + 2/   >R  ( R: median)
    2DUP BEGIN      ( lo_1 hi_2 lo_2 hi_1)
        SWAP BEGIN  DUP R@ *< @ EXECUTE WHILE  1+  REPEAT
        SWAP BEGIN  R@ OVER *< @ EXECUTE WHILE  1-  REPEAT
        2DUP > 0= IF
            \ Do we have a new position for our pivot?
            OVER R@ = IF R> DROP DUP >R ELSE
                DUP  R@ = IF R> DROP OVER >R THEN THEN
            2DUP *<--> @ EXECUTE
            >R 1+ R> 1-
        THEN
    2DUP > UNTIL    ( lo_1 hi_2 lo_2 hi_1)
    R> DROP                          ( R: )
    SWAP ROT ;      ( lo_1 hi_1 lo_2 hi_2)
```

Sort the range LOW to HIGH inclusive observing 'LOW' and 'HIGH' must be indices compatible with the current values of ***<** and ***<-->**.

39b ⟨*(QSORT)* 39b⟩≡ (38c)
```
: (QSORT)             ( lo hi -- )
    PARTITION         ( lo_1 hi_1 lo_2 hi_2)
    2DUP < IF  RECURSE  ELSE  2DROP  THEN
    2DUP < IF  RECURSE  ELSE  2DROP  THEN ;
```

3.4.7 Invented names

Make ADDRESS return some label NAME, static memory so use immediately.

39c ⟨*INVENT-NAME* 39c⟩≡ (39e)
```
: INVENT-NAME   s" L" NAME-BUF $!   0 8 (DH.) NAME-BUF $+! NAME-BUF $@ ;
```

For ADDRESS and NAME: "that name WAS invented".

39d ⟨*INVENTED-NAME?* 39d⟩≡ (39e)
```
: INVENTED-NAME?  9 <> IF 2DROP 0 ELSE SWAP INVENT-NAME CORA 0= THEN ;
```

39e ⟨*INVENTED-NAMES* 39e⟩≡ (33)
```
CREATE NAME-BUF   256 ALLOT

⟨INVENT-NAME 39c⟩
⟨INVENTED-NAME? 39d⟩

\D HEX S" EXPECT: L00000042 " TYPE 42 INVENT-NAME TYPE 1 <?> CR
\D HEX S" EXPECT: 0 " TYPE 42 s" L00000043" INVENTED-NAME? . 2 <?> CR
\D HEX S" EXPECT: -1 " TYPE 42 s" L00000042" INVENTED-NAME? . 3 <?> CR
\D DECIMAL
```

3.4.8 Miscellaneous

Debug comments.

40a ⟨*debug* 40a⟩≡ (33)
```
: \D POSTPONE \ ;
\ : \D ;
```

Missing conditionals.

40b ⟨=<>= 40b⟩≡ (33)
```
: >= < 0= ;
: <= > 0= ;
```

If FLAG is not zero, output STRING on the error channel and exit with an error code of 2.

40c ⟨*?ABORT* 40c⟩≡ (33)
```
: ?ABORT ROT IF ETYPE 2 EXIT-CODE ! BYE ELSE 2DROP THEN ;
```

ANS Forth specific

Exchange the content at ADDR1 and ADDR2 over a fixed LENGTH.

40d ⟨*EXCHANGE* 40d⟩≡ (33)
```
: EXCHANGE ( a1 a2 n -- )   0 ?DO  OVER I +  OVER I +  OVER C@  OVER C@
        >R SWAP C!  R> SWAP C!  LOOP  2DROP ;
```

ciForth specific

Print name of following definition.

40e ⟨*.NFA* 40e⟩≡ (33)
```
: .^   .S R@ @ >NFA @ $@ TYPE ;
```

3.5 Reverse Engineering Assembler

This file '`asgen.frt`' contains generic tools and has been used to make assemblers for 8080 8086 80386 Alpha 6809 and should be usable for Pentium 68000 6502 8051. It should run on ISO Forth's provided some ciforth facilities are present or emulated. The assemblers -- with some care -- have the property that the disassembled code can be assembled to the exact same code.

Most instruction sets follow this basic idea that they contain three distinct parts:

1. the opcode that identifies the operation
2. modifiers such as the register working on
3. data, as a bit field in the instruction.
4. data, including addresses or offsets.

This assembler goes through three stages for each instruction:

1. postit: assembles the opcode with holes for the modifiers. This has a fixed length. Also posts requirements for commaers.
2. fixup: fill up the holes, either from the beginning or the end of the post. These can also post required commaers.
3. fixup's with data. It has user supplied data in addition to opcode bits. Both together fill up bits left by a postit.
4. The commaers. Any user supplied data in addition to opcode, that can be added as separate bytes. Each has a separate command, where checks are built in.

Keeping track of this is done by bit arrays, similar to the a.i. blackboard concept. This is ONLY to notify the user of mistakes, they are NOT needed for the assembler proper. This setup allows a complete check of validity of code and complete control over what code is generated. Even so all checks can be defeated if need be.

The generic tools include:

- the defining words:
  for 1 2 3 4 byte postits,
  for fixups from front and behind
  for comma-ers.
- showing a list of possible instructions, for all opcodes or for a single one.
- disassembly of a single instruction or a range.
- hooks for more tools, e.g. print the opcode map as postscript.
- hooks for prefix instructions
- hooks for classes of instructions, to be turned off as a whole.

To write an assembler, make the tables, generate the complete list of instructions, assemble it and disassemble it again. If equal, you have a starting point for confidence in your work.

This code was at some time big-endian dependant and assumed a 32 bit machine! It is not sure that all traces of this have vanished. You cannot use this program as a cross-assembler if there are instructions that don't fit in a hosts cell (i.e. its postit) IT USES THE VOCABULARY AS A LINKED LIST OF STRUCTS (ciforth)! IT USED KNOWLEDGE OF THE INTERPRETER AND THE HEADERS! Now if you think that this makes this code non-portable, think again. You have to change about 8 lines to adapt. Now if you only have to adapt 8 lines in a 40k lines c-program with the same functionality, it would smack portable. Wouldn't it?

The blackboard consist of three bit arrays. At the start of an instruction they are all zero. '**TALLY-BI**' '**TALLY-BY**' '**TALLY-BA**' keep track of instruction bits, instruction byte and bad things respectively.

An instructions generally has a single postit that defines the opcode. It assembles the opcode, advancing '**HERE**' leaving zero bits that needs still filling in by fixups. It sets these bits in '**TALLY-BI**'. It may also post that commaers are required by setting a bit in '**TALLY-BY**'.

Then comes the fixups. They fill up the holes left in the instruction -- before '**HERE**' -- by or-ing and maintain '**TALLY-BI**', resetting bits. They end in '**|**' where the other assembly actions end in '**,**'. They may require more commaers, posting to '**TALLY-BY**'. The commaers advance '**HERE**' by a whole number of bytes assembling user supplied information and reset the corresponding bits in '**TALLY-BY**'.

All parts of an instruction can add bits to '**TALLY-BA**'. If any two consecutive bits are up this is bad. Its bits can never be reset but '**TALLY-BA**' is reset as a whole at the start of an instruction.

An example: load an index register with a 16 bit value, 8080.

TALLY-BI	TALLY-BY	TALLY-BA	HERE	8A43 4 DUMP	
0000	0000	0801	8A43	`.. .. .. ..`	**LXI,**
0030	0002	0002	8A44	**01** `.. .. ..`	**SP|**
0000	0002	0002	8A44	**31** `.. .. ..`	**SP0 @ X,**
0000	0000	0002	8A46	**31 00 FE** `..`	**HLT,**
0000	0000	0000	8A47	**31 00 FE 76**	**...**

The bit in '**TALLY-BA**' means a 16 bit operation. Now if '**TALLY-BA**' contains 3 it would mean that it is at the same time an 8 bit and 16 bit operation. Bad!)

The following problems can be detected:

- postit when '**TALLY-BI**' or '**TALLY-BY**' contains bits up
- setting or resetting bits for the second time in '**TALLY-BI**' or '**TALLY-BY**'
- commaing when '**TALLY-BI**' still contains bits up
- setting '**TALLY-BA**' bad

A prefix PostIt has its prefix field filled in with an execution token. This token represents the action performed on the **TALLY-BA** flags, that is used instead of resetting it. This can be used for example for the OS -- operand size -- prefix in the Pentium. Instead of putting the information that we are in a 16 bit operand segment in **TALLY-BA**, it transforms that information to 32 bit.

42 ⟨*asgen.frt* 42⟩≡ (22)

```
( $Id: asgen.frt,v 4.31 2005/03/07 11:54:58 albert Exp $ )
( Copyright{2000}: Albert van der Horst, HCC FIG Holland by GNU Public License)
```

```
( Uses Richard Stallmans convention. Uppercased word are parameters.    )
```

⟨*prelude* 44a⟩
```
( ################ PART I ASSEMBLER ###################################### )
```
⟨*assembler-utilities* 44b⟩
⟨*dependant-utilities* 46⟩
⟨*independant-utilities* 48c⟩
⟨*assembler-bookkeeping* 49d⟩
⟨*assembler-defining-words* 52e⟩
⟨*obsolescent* 54e⟩
⟨*preferred* 55e⟩
⟨*end-preferred* 56e⟩
⟨*super-defining-words* 56i⟩
```
( ################ PART II DISASSEMBLER ################################## )
```
⟨*structures* 57f⟩
⟨*tryers* 58d⟩
⟨*disassemblers* 61c⟩

43 ⟨*defining-words* 43⟩≡ 66d ▷
⟨*conveniences* 67⟩
```
PREVIOUS
```

3.5.1 Prelude

Wrapper for asgen, when we want to test without label mechanisms. These are hot patched for reverse engineering.

44a ⟨*prelude* 44a⟩≡ (42)

```
[DEFINED] ForSwiftForth
[DEFINED] ForGForth OR [IF]
    : (?ERROR) ( f n -- )   swap if throw else drop then ;
    CREATE '?ERROR   ' (?ERROR) DUP , ,   : ?ERROR '?ERROR @ EXECUTE ;
    : ?CSP ( -- ) ; \ check stack
    : !CSP ( -- ) ; \ set stack check
    : ?EXEC ( -- ) ;
    : asm-create ( -- )   create ;
[THEN]
[DEFINED] ForCiForth [IF]
    : asm-create ( -- )   (WORD) (CREATE) ;
    REQUIRE ALIAS
    REQUIRE @+ ( Fetch from ADDRES. Leave incremented ADDRESS and DATA )
    REQUIRE BAG
    REQUIRE POSTFIX
[THEN]

\ Vectors that are hot-patched in aswrap.frt
CREATE 'AS-ALLOT   ' ALLOT DUP , ,   : AS-ALLOT 'AS-ALLOT @ EXECUTE ;
CREATE 'AS-HERE    ' HERE DUP , ,    : AS-HERE 'AS-HERE @ EXECUTE ;
CREATE 'AS-C,      ' C, DUP , ,      : AS-C, 'AS-C, @ EXECUTE ;
CREATE '_AP_       ' HERE DUP , ,    : _AP_ '_AP_ @ EXECUTE ;

: (-ADORN-ADDRESS) DROP CR ;   ( Action between two disassembled instr.    )

CREATE 'ADORN-ADDRESS   ' (-ADORN-ADDRESS) DUP , ,
: ADORN-ADDRESS ( a -- )   'ADORN-ADDRESS @ EXECUTE ;
```

3.5.2 Maybe not present utilities

44b ⟨*assembler-utilities* 44b⟩≡ (42)

```
⟨!+ 45a⟩
⟨@- 45b⟩
⟨CTRL 45c⟩
⟨TABLE1 45d⟩
⟨ROTLEFT 45e⟩

[DEFINED] ForCiForth [IF]
    'TABLE1 HIDDEN
[THEN]
```

Store DATA to ADDRES. Leave incremented ADDRESS

45a ⟨*!+* 45a⟩≡ (44b)

```
: !+ ( x a -- a' )   >R R@ ! R> CELL+ ;
```

Fetch from decremented ADDRES. Leave DATA and ADDRESS

45b ⟨*@-* 45b⟩≡ (44b)

```
: @- ( a -- x a' )   0 CELL+ - >R R@ @ R>  ;
```

45c ⟨*CTRL* 45c⟩≡ (44b)

```
: CTRL CHAR 31 AND ;
: [CTRL] CTRL POSTPONE LITERAL ; IMMEDIATE
```

`x TABLE1 +` `@` yields $100^[-x mod 4]

45d ⟨*TABLE1* 45d⟩≡ (44b)

```
CREATE TABLE1   1 , 1 ,
```

Rotate X by I bytes left leaving X' Left i.e. such as it appears in memory! Not as printed on a big endian machine! aqa "**`8 * LSHIFT`**" on bigendian.

45e ⟨*ROTLEFT* 45e⟩≡ (44b)

```
: ROTLEFT ( x n -- x' )   TABLE1 + @ UM* OR ;
```

3.5.3 System dependant utilities

Common fields in the defining words for posits fixups and commaers. All leave a single ADDRESS. The first data field for a postit/fixup contains instruction bits, for a commaer it contains the xt of the comma action for a data fixup it contains the position of the bits.

46 ⟨*dependant-utilities* 46⟩≡ (42)

```
: %>BODY ; ( From DEA to the DATA field of a created word, now the same )
: %BODY> ; ( Reverse of above)

: DEA-FIELD ( u size -- u' )   CREATE  OVER , +
    DOES> ( dea -- a )   @ SWAP %>BODY + ;

0
  1 CELLS DEA-FIELD >LFA    ( link to previous word for compatibility   )
  1 CELLS DEA-FIELD >NFA    ( variable length name field is at the end  )
  1 CELLS DEA-FIELD >DATA

( Work on TALLY-BI etc.        Effects  for posits fixups and commaers. )
(                                          |||    |||       |||       )
  1 CELLS DEA-FIELD >BI                  ( OR!    AND!      --        )
  1 CELLS DEA-FIELD >BY                  ( OR!    OR!       AND!      )
  1 CELLS DEA-FIELD >BA                  ( OR!U   OR!U      OR!U      )
  1 CELLS DEA-FIELD >CNT    ( 'HERE' advances with count )
  0 CELLS DEA-FIELD >DIS    ( disassembler only for COMMA , 0 -> default)
  1 CELLS DEA-FIELD >PRF    ( prefix flag, only for PI ,    0 -> default)
  1 CELLS DEA-FIELD >DFA    ( type of word, replacing the DOES> check   )
CONSTANT |DEA| ( the name is tacked onto the end when it is created     )

⟨ID. 47a⟩

VOCABULARY ASSEMBLER IMMEDIATE   ALSO ASSEMBLER DEFINITIONS HEX

[DEFINED] ForCiForth [IF]
    ⟨'alias 48b⟩
[THEN]

⟨DOES 47b⟩
⟨IGNORE? 47c⟩
⟨VOCEND? 47d⟩
⟨NEXT 47e⟩
⟨STARTVOC 47f⟩
⟨IS-A 47g⟩
⟨MEMBER 47h⟩
⟨?ERROR- 47i⟩
⟨CREATE-- 48a⟩
```

Print a definition's name from its DEA.

47a ⟨*ID.* 47a⟩≡ (46)
```
: %ID. >NFA @ $@ TYPE SPACE ;
```

From DEA to the **DOES>** pointer for a '**DOES>**' word

47b ⟨*DOES* 47b⟩≡ (46)
```
: %>DOES ( dea -- x ) >DFA ;
```

Leave for DEA : it is to be ignored in disassemblies. This is used for supressing the bare bones of the sib mechanism in i586.

47c ⟨*IGNORE?* 47c⟩≡ (46)
```
: IGNORE? >NFA @ CHAR+ C@ [CHAR] ~ = ;
```

Given a DEA, return the next DEA. For a DEA as returned from **(>NEXT%)** : it is the end, not a real DEA.

47d ⟨*VOCEND?* 47d⟩≡ (46)
```
: VOCEND? ( dea -- f )   >LFA @ 0= ;
```

As **(>NEXT%)** but skip holes, i.e. words with names starting in '**-**'.

47e ⟨*NEXT* 47e⟩≡ (46)
```
: (>NEXT%) ( dea -- dea )   >LFA @ ;
: >NEXT% ( dea -- dea' )   BEGIN  (>NEXT%) DUP >NFA @ CHAR+ C@
    [CHAR] - - UNTIL ;
```

Leave the first DEA of the assembler vocabulary.

47f ⟨*STARTVOC* 47f⟩≡ (46)
```
0 VALUE STARTVOC ( -- dea )
```

Build: allocate place to remember a **DOES>** address of a '**CREATE**'d word. Leave that ADDRESS to be filled in by '**REMEMBER**' Execution: Leave for DEA : it is of same type as the remembered **DOES>**.

47g ⟨*IS-A* 47g⟩≡ (46)
```
VARIABLE 'IS-A   1 'IS-A !
: IS-A ( -- )   CREATE  'IS-A @ ,  DOES> ( dea -- f ) @ SWAP %>DOES @ = ;
```

Patch up the data field of a preceeding word defined by '**IS-A**'. To be called when sitting at the **DOES>** address. The **!CSP** / **?CSP** detects stack changes. Now split it into 2 checks.

47h ⟨*MEMBER* 47h⟩≡ (46)
```
: MEMBER ( n -- )   STARTVOC >DFA ! ;
: REMEMBER ( -- )   ?CSP  'IS-A @ POSTPONE LITERAL  POSTPONE MEMBER
    1 'IS-A +!  !CSP ; IMMEDIATE
```

Also needed : ?ERROR that defeats all checks.

47i ⟨*?ERROR-* 47i⟩≡ (46)
```
\ : ?ERROR DROP DROP ;
```

Behaves as 'CREATE' except, if the word to be created has name "--" it is ignored, by making the header unfindable. Not strictly needed.

48a ⟨*CREATE--* 48a⟩≡ (46)
```
: CREATE--   SAVE-INPUT  CREATE  HERE DUP >R  |DEA| DUP ALLOT  ERASE
    RESTORE-INPUT THROW  BL WORD $@ $, R@ >NFA !
    STARTVOC R@ >LFA !  R> TO STARTVOC ;
```

ciForth specific

Make an alias for "'" in the minimum search order called "%".

48b ⟨*'alias* 48b⟩≡ (46)
```
'ONLY >WID CURRENT !    \ Making ONLY the CONTEXT is dangerous! This will do.
"'" 'ONLY >WID (FIND)   ALIAS %         ( "'" ) 2DROP
CONTEXT @ CURRENT !     \ Restore current.
```

3.5.4 System independant utilities

Note that the assembler works with multi-character bigendian numbers.

48c ⟨*independant-utilities* 48c⟩≡ (42)
⟨*CONTAINED-IN* 48d⟩
⟨*lsbyte,* 48e⟩
⟨*lsbyte-at* 48f⟩
⟨*lsbytes* 48g⟩
⟨*MCat* 49a⟩
⟨*MC<0* 49b⟩
⟨*MC@-S* 49c⟩

The FIRST bitset is contained in the SECOND one, leaving it IS.

48d ⟨*CONTAINED-IN* 48d⟩≡ (48c)
```
: CONTAINED-IN OVER AND = ;
```

Compile the ls 8 bits of X at here, leaving the REMAINING bits.

48e ⟨*lsbyte,* 48e⟩≡ (48c)
```
: lsbyte, DUP AS-C, 0008 RSHIFT ;
```

For X and ADDRESS , add the byte below address to x at l.s. place. Leave X and decremented ADDRESS.

48f ⟨*lsbyte-at* 48f⟩≡ (48c)
```
: lsbyte@ 1- SWAP 8 LSHIFT OVER C@ OR SWAP ;
```

For X ADDRESS LENGTH , return the NUMBER that at address (bigendian). x provides a filler, -1 results in sign extension.

48g ⟨*lsbytes* 48g⟩≡ (48c)
```
: lsbytes  >R R@ + BEGIN R> DUP WHILE 1- >R  lsbyte@ REPEAT 2DROP ;
```

For ADDRESS LENGTH , return the NUMBER that is there (bigendian). "Multiple byte fetch".

49a ⟨*MCat* 49a⟩≡ (48c)
```
: MC@ 0 ROT ROT lsbytes ;
```

For ADDRESS LENGTH , return the "number there IS negative".

49b ⟨*MC<0* 49b⟩≡ (48c)
```
: MC<0 + 1- C@ 80 AND 80 = ;
```

For ADDRESS LENGTH , return the NUMBER that is there. bigendian and signextended. "Multiple byte fetch, signed".

49c ⟨*MC@-S* 49c⟩≡ (48c)
```
: MC@-S 2DUP MC<0 ROT ROT lsbytes ;
```

3.5.5 Assembler bookkeeping

The bookkeeping is needed for error detection and disassembly.

49d ⟨*assembler-bookkeeping* 49d⟩≡ (42)
```
⟨TALLY-BI 50a⟩
⟨TALLY-BY 50b⟩
⟨TALLY-BA 50c⟩
⟨BA-DEFAULT 50d⟩
⟨OLDCOMMA 50e⟩
⟨ISS 50f⟩
⟨ISL 50g⟩
⟨BA-XT 50h⟩
⟨RESET-BAD 50i⟩
⟨!TALLY 50j⟩
⟨AT-REST? 50k⟩
⟨BADPAIRS? 51a⟩
⟨CONSISTENT? 51b⟩
DECIMAL
⟨CHECK26 51c⟩
⟨CHECK32 51d⟩
⟨CHECK31 51e⟩
⟨CHECK31A 51f⟩
⟨CHECK32B 51g⟩
⟨CHECK33 51h⟩
⟨CHECK28 51i⟩
⟨CHECK29 52a⟩
⟨CHECK30 52b⟩
HEX
⟨OR! 52c⟩
⟨AND! 52d⟩
```

Bits that need to be fixed up.

50a ⟨*TALLY-BI* 50a⟩≡ (49d)
```
VARIABLE TALLY-BI
```

Bits represent a commaer that is to be supplied.

50b ⟨*TALLY-BY* 50b⟩≡ (49d)
```
VARIABLE TALLY-BY
```

State bits, bad if two consequitive bits are up.

50c ⟨*TALLY-BA* 50c⟩≡ (49d)
```
VARIABLE TALLY-BA
```

Bits set in the default can be used to exclude certain classes of instructions, e.g. because they are not implemented.

50d ⟨*BA-DEFAULT* 50d⟩≡ (49d)
```
VARIABLE BA-DEFAULT    0 BA-DEFAULT !
```

Previous comma, or zero.

50e ⟨*OLDCOMMA* 50e⟩≡ (49d)
```
VARIABLE OLDCOMMA
```

Start of current instruction.

50f ⟨*ISS* 50f⟩≡ (49d)
```
VARIABLE ISS
```

Length of current instruction

50g ⟨*ISL* 50g⟩≡ (49d)
```
VARIABLE ISL
```

To be executed instead of reset BA between prefix and instruction.

50h ⟨*BA-XT* 50h⟩≡ (49d)
```
VARIABLE BA-XT
```

Reset '**BA**' to default for begin instruction, unless prefix.

50i ⟨*RESET-BAD* 50i⟩≡ (49d)
```
: RESET-BAD ( -- )   BA-XT @ DUP IF EXECUTE ELSE
        DROP  BA-DEFAULT @ TALLY-BA ! THEN ;
```

Initialise '**TALLY**'.

50j ⟨*!TALLY* 50j⟩≡ (49d)
```
: !TALLY ( -- )   0 TALLY-BI !   0 TALLY-BY !   RESET-BAD   0 OLDCOMMA ! ;
    0 BA-XT !   !TALLY
```

Return: instruction IS complete, or not started.

50k ⟨*AT-REST?* 50k⟩≡ (49d)
```
: AT-REST? TALLY-BI @ 0=   TALLY-BY @ 0=  AND ;
```

For N : it CONTAINS bad pairs.

51a ⟨*BADPAIRS?* 51a⟩≡ (49d)
```
: BADPAIRS? DUP 1 LSHIFT AND AAAAAAAAAAAAAAAA AND ;
: BAD? TALLY-BA @ BADPAIRS? ;
```

The state of assembling is inconsistent. If STATUS were added to '**TALLY-BA**' would that create a bad situation?

51b ⟨*CONSISTENT?* 51b⟩≡ (49d)
```
: CONSISTENT? TALLY-BA @ OR BADPAIRS? 0= ;
```

Generate errors. None have net stack effects, such that they may be replaced by NULL definitions. Error at postit time.

51c ⟨*CHECK26* 51c⟩≡ (49d)
```
: CHECK26 AT-REST? 0= ABORT" PREVIOUS INSTRUCTION INCOMPLETE" ;
```

Always an error.

51d ⟨*CHECK32* 51d⟩≡ (49d)
```
: CHECK32 BAD? ABORT" PREVIOUS OPCODE PLUS FIXUPS INCONSISTENT" ;
```

Generate error for fixup, if for the BI, some of the BITS would stick out it. Leave MASK and BITS . Programming error!

51e ⟨*CHECK31* 51e⟩≡ (49d)
```
: CHECK31 2DUP SWAP CONTAINED-IN 0=
    ABORT" DESIGN ERROR, INCOMPATIBLE MASK" ;
```

Generate error for "FIXUP-DATA" , if the BI and the LEN are not compatible. Leave BI and LEN . Programming error!

51f ⟨*CHECK31A* 51f⟩≡ (49d)
```
: CHECK31A 2DUP OVER >R RSHIFT 1 OR OVER LSHIFT R> <>
    ABORT" DESIGN ERROR, INCOMPATIBLE MASK" ;
```

The part of BITS outside of BITFIELD must be either all ones or zeros. This checks for a shifted signed field.

51g ⟨*CHECK32B* 51g⟩≡ (49d)
```
: CHECK32B  2DUP OR INVERT 0= ( all ones) >R
    INVERT AND 0= ( all zero's ) R> OR ( okay)
    0= ABORT" PREVIOUS OPCODE PLUS FIXUPS INCONSISTENT" ;
```

Generate error for postit, if for the inverted BI , some of the the BITS would stick out it. Leave MASK and BITS . Programming error!

51h ⟨*CHECK33* 51h⟩≡ (49d)
```
: CHECK33 2DUP SWAP INVERT CONTAINED-IN 0=
    ABORT" DESIGN ERROR, INCOMPATIBLE MASK" ;
```

BITS would stick out it. Leave MASK and BITS . Programming error! Generate error on data for postit/fixup, if some BITS to fill in are already in the MASK. Leave BITS and MASK.

51i ⟨*CHECK28* 51i⟩≡ (49d)
```
: CHECK28 2DUP AND ABORT" UNEXPECTED FIXUP/COMMAER" ;
```

Generate error on data for commaer, if the BITS to reset are not present in the MASK. Leave BITS and MASK.

52a ⟨*CHECK29* 52a⟩≡ (49d)
```
: CHECK29 2DUP OR -1 - ABORT" DUPLICATE FIXUP/UNEXPECTED COMMAER" ;
```

Generate error if COMMAMASK is not in ascending order. Leave IT.

52b ⟨*CHECK30* 52b⟩≡ (49d)
```
: CHECK30 DUP OLDCOMMA @ < ABORT" COMMAERS IN WRONG ORDER"
    DUP OLDCOMMA ! ;
```

Or DATA into ADDRESS. If bits were already up its wrong.

52c ⟨*OR!* 52c⟩≡ (49d)
```
: OR! >R R@ @  CHECK28 OR R> ! ;
: OR!U >R R@ @  OR R> ! ;
```

Or DATA into ADDRESS. Unchecked. Reset bits of DATA into ADDRESS. If bits were already down it's wrong.

52d ⟨*AND!* 52d⟩≡ (49d)
```
: AND! >R INVERT R@ @ CHECK29 AND R> ! ;
```

3.5.6 Assembler defining words

52e ⟨*assembler-defining-words* 52e⟩≡ (42)
- ⟨*assemble,* 52f⟩
- ⟨*!POSTIT* 52g⟩
- ⟨*TALLY:,* 53a⟩
- ⟨*POSTIT* 53b⟩
- ⟨*BUILD-IP* 53c⟩
- ⟨*PIS* 53d⟩
- ⟨*IS-PI* 53e⟩
- ⟨*TALLY:|* 53f⟩
- ⟨*FIXUP* 53g⟩
- ⟨*xFI* 53h⟩
- ⟨*TRIM-SIGNED* 53i⟩
- ⟨*FIXUP-DATA* 54a⟩
- ⟨*FIXUP-SIGNED* 54b⟩
- ⟨*DFI* 54c⟩
- ⟨*DFIs* 54d⟩

Assemble INSTRUCTION for "ISL" bytes. ls byte first.

52f ⟨*assemble,* 52f⟩≡ (52e)
```
: assemble, ( x -- )   ISL @ 0 DO  lsbyte,  LOOP  DROP ;
```

Initialise in behalf of postit.

52g ⟨*!POSTIT* 52g⟩≡ (52e)
```
: !POSTIT ( -- )   AS-HERE ISS !  0 OLDCOMMA ! ;
```

Bookkeeping for a postit using a pointer to the BIBYBA information, can fake a postit in disassembling too.

53a ⟨*TALLY:*, 53a⟩≡ (52e)
```
: TALLY:, ( a -- )   DUP >BI @ TALLY-BI !  DUP >BY @ TALLY-BY !
    DUP >BA @ TALLY-BA OR!U  DUP >CNT @ ISL !  >DIS @ BA-XT ! ;
```

Post the instruction using DATA.

53b ⟨*POSTIT* 53b⟩≡ (52e)
```
: POSTIT ( a -- )   CHECK26   !TALLY   !POSTIT
    DUP >DATA @ >R   TALLY:,   R> assemble, ;
```

Build an instruction given by BA BY BI the OPCODE and COUNT.

53c ⟨*BUILD-IP* 53c⟩≡ (52e)
```
: BUILD-IP ( ba by bi opc cnt -- )   STARTVOC >CNT !  STARTVOC >DATA !
    STARTVOC >BI !  STARTVOC >BY !  STARTVOC >BA !
    0 ( prefix) STARTVOC >PRF ! ;
```

Define an instruction by BA BY BI and the OPCODE. For 1 2 3 and 4 byte opcodes.

53d ⟨*PIS* 53d⟩≡ (52e)
```
IS-A IS-1PI : 1PI  CHECK33 CREATE-- REMEMBER  1 BUILD-IP DOES>  POSTIT ;
IS-A IS-2PI : 2PI  CHECK33 CREATE-- REMEMBER  2 BUILD-IP DOES>  POSTIT ;
IS-A IS-3PI : 3PI  CHECK33 CREATE-- REMEMBER  3 BUILD-IP DOES>  POSTIT ;
IS-A IS-4PI : 4PI  CHECK33 CREATE-- REMEMBER  4 BUILD-IP DOES>  POSTIT ;
```

For DEA : it REPRESENTS some kind of opcode.

53e ⟨*IS-PI* 53e⟩≡ (52e)
```
: IS-PI  >R 0
    R@ IS-1PI OR  R@ IS-2PI OR  R@ IS-3PI OR   R@ IS-4PI OR R> DROP ;
```

Bookkeeping for a fixup using a pointer to the BIBYBA information, can fake a fixup in disassembling too.

53f ⟨*TALLY:|* 53f⟩≡ (52e)
```
: TALLY:| ( a -- )   DUP >BI @ TALLY-BI AND!  DUP >BY @ TALLY-BY OR!
    >BA @ TALLY-BA OR!U ;
```

Fix up the instruction using a pointer to DATA.

53g ⟨*FIXUP* 53g⟩≡ (52e)
```
: FIXUP>   DUP >DATA @ ISS @ OR!   TALLY:|   CHECK32 ;
```

Define a fixup by BA BY BI and the FIXUP bits. One size fits all, because of the or character of the operations.

53h ⟨*xFI* 53h⟩≡ (52e)
```
IS-A IS-xFI : xFI  CHECK31 CREATE-- REMEMBER STARTVOC >DATA !
    STARTVOC >BI !  STARTVOC >BY !  STARTVOC >BA ! DOES>  FIXUP> ;
```

For a signed DATA item a LENGTH and a BITFIELD. Shift the data item into the bit field and leave IT. Check if it doesn't fit.

53i ⟨*TRIM-SIGNED* 53i⟩≡ (52e)
```
: TRIM-SIGNED >R   2DUP R@ SWAP RSHIFT CHECK32B   LSHIFT R> AND ;
```

Fix up the instruction using DATA and a pointer to the bit POSITION.

54a ⟨*FIXUP-DATA* 54a⟩≡ (52e)
```
: FIXUP-DATA ( a -- )   DUP >DATA @  SWAP >R  LSHIFT ISS @ OR!
    R> TALLY:|  CHECK32 ;
```

Fix up the instruction using DATA and a pointer to the bit POSITION.

54b ⟨*FIXUP-SIGNED* 54b⟩≡ (52e)
```
: FIXUP-SIGNED ( a -- )   DUP >DATA @  SWAP >R
    R@ >BI @ TRIM-SIGNED ISS @ OR!
    R> TALLY:| CHECK32 ;
```

Define a data fixup by BA BY BI, and LEN the bit position. At assembly time: expect DATA that is shifted before use. One size fits all, because of the or character of the operations.

54c ⟨*DFI* 54c⟩≡ (52e)
```
IS-A IS-DFI : DFI  CHECK31A CREATE-- REMEMBER STARTVOC >DATA !
    STARTVOC >BI !  STARTVOC >BY !  STARTVOC >BA !  DOES>  FIXUP-DATA ;
```

Same, but for signed data.

54d ⟨*DFIs* 54d⟩≡ (52e)
```
IS-A IS-DFIs : DFIs  CHECK31A CREATE-- REMEMBER STARTVOC >DATA !
    STARTVOC >BI !  STARTVOC >BY !  STARTVOC >BA !  DOES>  FIXUP-SIGNED ;
```

3.5.7 Obsolescent

54e ⟨*obsolescent* 54e⟩≡ (42)
⟨*REVERSE-BYTES* 54f⟩
⟨*CORRECT-R* 54g⟩
⟨*TALLY:|R* 55a⟩
⟨*FIXUP<* 55b⟩
⟨*FIR* 55c⟩
⟨*DFIR* 55d⟩

Reverses bytes in a WORD. Return IT.

54f ⟨*REVERSE-BYTES* 54f⟩≡ (54e)
```
: REVERSE-BYTES
    1 CELLS 0 DO DUP  FF AND SWAP 8 RSHIFT   LOOP
    8 CELLS 0 DO SWAP I LSHIFT OR   8 +LOOP ;
```

Rotate the MASK etc from a fixup-from-reverse into a NEW mask fit for using from the start of the instruction. We know the length!

54g ⟨*CORRECT-R* 54g⟩≡ (54e)
```
: CORRECT-R 0 CELL+ ISL @ - ROTLEFT ;
```

Bookkeeping for a fixup-from-reverse using a pointer to the BIBYBA information, can fake a fixup in disassembling too.

55a ⟨*TALLY:|R* 55a⟩≡ (54e)
```
: TALLY:|R ( a -- )   DUP >BI @ CORRECT-R TALLY-BI AND!
    DUP >BY @ TALLY-BY OR!  >BA @ TALLY-BA OR!U ;
```

Fix up the instruction from reverse with DATA.

55b ⟨*FIXUP<* 55b⟩≡ (54e)
```
: FIXUP<   CORRECT-R ISS @ OR! ;
```

Define a fixup-from-reverse by BA BY BI and the FIXUP bits. One size fits all, because of the character of the or-operations. BI and fixup are specified that last byte is lsb, such as you read it.

55c ⟨*FIR* 55c⟩≡ (54e)
```
IS-A IS-FIR : FIR  CHECK31 CREATE-- REMEMBER REVERSE-BYTES STARTVOC >DATA !
    REVERSE-BYTES STARTVOC >BI !  STARTVOC >BY !  STARTVOC >BA !
    DOES>  DUP >DATA @  FIXUP< TALLY:|R  CHECK32 ;
```

Define a fixup-from-reverse by BA BY BI and LEN to shift. One size fits all, because of the character of the or-operations. BI and fixup are specified that last byte is lsb, such as you read it.

55d ⟨*DFIR* 55d⟩≡ (54e)
```
IS-A IS-DFIR : DFIR  CHECK31 CREATE-- REMEMBER STARTVOC >DATA !
    REVERSE-BYTES STARTVOC >BI !  STARTVOC >BY !  STARTVOC >BA !
    DOES>  DUP >DATA @  SWAP >R  LSHIFT REVERSE-BYTES FIXUP<
        R> TALLY:|R  CHECK32 ;
```

3.5.8 Preferred

Not yet used???

55e ⟨*preferred* 55e⟩≡ (42)
⟨*(AND!BYTE)* 55f⟩
⟨*AND!BYTE* 55g⟩
⟨*(OR!BYTE)* 56a⟩
⟨*OR!BYTE* 56b⟩
⟨*TALLY:|R'* 56c⟩
⟨*FIXUP<'* 56d⟩

If bits were already down it is wrong, for next two words. Reset bits of DATA into ADDRESS bytewise.

55f ⟨*(AND!BYTE)* 55f⟩≡ (55e)
```
: (AND!BYTE) >R 0FF AND INVERT R@ C@ CHECK29 AND R> C! ;
```

Reset bits of DATA byte by byte into ADDRESS.

55g ⟨*AND!BYTE* 55g⟩≡ (55e)
```
: AND!BYTE BEGIN 2DUP (AND!BYTE) SWAP 8 RSHIFT DUP WHILE SWAP 1+ REPEAT 2DROP ;
```

If bits were already up its wrong, for next two words. Or DATA into ADDRESS bytewise.

56a ⟨*(OR!BYTE)* 56a⟩≡ (55e)
```
: (OR!BYTE) >R R@ C@  CHECK28 OR R> C! ;
```

Or DATA byte by byte from behind into ADDRESS.

56b ⟨*OR!BYTE* 56b⟩≡ (55e)
```
: OR!BYTE BEGIN 1- 2DUP (OR!BYTE) SWAP 8 RSHIFT DUP WHILE SWAP REPEAT 2DROP ;
```

Bookkeeping for a fixup-from-reverse using a pointer to the BIBYBA information, can fake a fixup in disassembling too.

56c ⟨*TALLY:|R'* 56c⟩≡ (55e)
```
: TALLY:|R'  DUP >BI @ TALLY-BI AND!BYTE  DUP >BY @ TALLY-BY OR!
    >BA @ TALLY-BA OR!U ;
```

Fix up the instruction from reverse using a pointer to DATA.

56d ⟨*FIXUP<'* 56d⟩≡ (55e)
```
: FIXUP<'   DUP >DATA @ ISS @ ISL @ + OR!BYTE  TALLY:|R'  CHECK32 ;
```

3.5.9 End preferred

56e ⟨*end-preferred* 56e⟩≡ (42)
⟨*TALLY:„* 56f⟩
⟨*COMMA* 56g⟩
⟨*COMMAER* 56h⟩

Bookkeeping for a commaer using a pointer to the BIBYBA information. Not used by the disassembler.

56f ⟨*TALLY:„* 56f⟩≡ (56e)
```
: TALLY:„ ( a -- )   DUP >BY @ CHECK30 TALLY-BY AND!  >BA @ TALLY-BA OR!U ;
```

56g ⟨*COMMA* 56g⟩≡ (56e)
```
: COMMA ( a -- )   DUP >DATA @ >R  TALLY:„  CHECK32  R> EXECUTE ;
```

Build with an disassembly ROUTINE, with the LENGTH to comma, the BA BY information and the ADDRESS that is executing the commaer. A disassembly routine gets the "DEA" of the commaer on stack.

56h ⟨*COMMAER* 56h⟩≡ (56e)
```
IS-A IS-COMMA : COMMAER CREATE-- REMEMBER STARTVOC >DATA !  0 STARTVOC >BI !
    STARTVOC >BY !  STARTVOC >BA !  STARTVOC >CNT !  STARTVOC >DIS !
    DOES>  COMMA ;
```

3.5.10 Assembler super defining words

56i ⟨*super-defining-words* 56i⟩≡ (42)
⟨*PRO-TALLY* 57a⟩
⟨*T!* 57b⟩
⟨*T!R* 57c⟩
⟨*Tat* 57d⟩
⟨*FAMILIES* 57e⟩

Prototype for TALLY-BI BY BA.

57a ⟨*PRO-TALLY* 57a⟩≡ (56i)
```
CREATE PRO-TALLY 3 CELLS ALLOT
```

Fill in the tally prototype with BA BY BI.

57b ⟨*T!* 57b⟩≡ (56i)
```
: T! PRO-TALLY !+ !+ !+ DROP ;
```

Reversed BI information.

57c ⟨*T!R* 57c⟩≡ (56i)
```
: T!R   REVERSE-BYTES T! ;
```

Get the data from the tally prototype back BA BY BI.

57d ⟨*Tat* 57d⟩≡ (56i)
```
: T@ PRO-TALLY 3 CELLS +  @- @- @- DROP ;
```

Add INCREMENT to the OPCODE a NUMBER of times, and generate as much instructions, all with the same BI-BA-BY from '**PRO-TALLY**'. For each assembler defining word there is a corresponding family word. Words named "--" are mere placeholders.

57e ⟨*FAMILIES* 57e⟩≡ (56i)
```
: 1FAMILY,    0 DO   DUP >R T@ R> 1PI   OVER + LOOP DROP DROP ;
: 2FAMILY,    0 DO   DUP >R T@ R> 2PI   OVER + LOOP DROP DROP ;
: 3FAMILY,    0 DO   DUP >R T@ R> 3PI   OVER + LOOP DROP DROP ;
: 4FAMILY,    0 DO   DUP >R T@ R> 4PI   OVER + LOOP DROP DROP ;
: xFAMILY|    0 DO   DUP >R T@ R> xFI   OVER + LOOP DROP DROP ;
: FAMILY|R    0 DO   DUP >R T@ REVERSE-BYTES  R> FIR   OVER + LOOP DROP DROP ;
: xFAMILY|F   0 DO   DUP >R T@ R> DFI   OVER + LOOP DROP DROP ;
```

3.5.11 Disassembler data structures

57f ⟨*structures* 57f⟩≡ (42)
```
⟨DISS 58a⟩
: !DISS DISS !BAG ;
: .DISS-AUX DISS @+ SWAP DO
        I @ DUP IS-COMMA OVER IS-DFI OR OVER IS-DFIs OR IF I DISS - . THEN
        [DEFINED] ForSwiftForth [IF]  .'  [THEN]
        [DEFINED] ForGForth [IF]  ID.  [THEN]
    0 CELL+ +LOOP  CR ;
⟨DISS-VECTOR 58b⟩
: +DISS DISS BAG+! ;
: DISS? DISS BAG? ;
⟨DISS- 58c⟩
```

A row of dea's representing a disassembly.

58a ⟨*DISS* 58a⟩≡ (57f)
```
12 BAG DISS
```

DISS-VECTOR can be redefined to generate testsets.

58b ⟨*DISS-VECTOR* 58b⟩≡ (57f)
```
VARIABLE DISS-VECTOR    ' .DISS-AUX DISS-VECTOR !
```

Discard last item of '**DISS**'

58c ⟨*DISS-* 58c⟩≡ (57f)
```
: DISS- 0 CELL+ NEGATE DISS +! ;
```

3.5.12 Tryers

Tryers try to construct an instruction from current bookkeeping. They can backtrack to show all possibilities.

58d ⟨*tryers* 58d⟩≡ (42)
```
⟨TRY-PI 58e⟩
⟨TRY-xFI 59a⟩
⟨TRY-DFI 59b⟩
⟨TRY-FIR 59c⟩
⟨TRY-COMMA 59d⟩
⟨REBUILD 59e⟩
⟨BACKTRACK 60a⟩
⟨RESULT? 60b⟩
⟨.RESULT 60c⟩
\     % RESULT +DISS Spurious? Remove after next total test.
⟨SHOW-STEP 60d⟩
⟨SHOW-ALL 60e⟩
⟨SHOW-OPCODES 61a⟩
⟨SHOW: 61b⟩
```

These tryers are quite similar: if the DEA on the stack is of the right type and if the precondition is fullfilled it does the reassuring actions toward the tally as with assembling and add the fixup/posti/commaer to the disassembly struct. as if this instruction were assembled. Leave the DEA.

58e ⟨*TRY-PI* 58e⟩≡ (58d)
```
: TRY-PI
    DUP IS-PI IF
    AT-REST? IF
        DUP TALLY:,
        DUP +DISS
    THEN
    THEN ;
```

59a ⟨*TRY-xFI* 59a⟩≡ (58d)
```
: TRY-xFI
    DUP IS-xFI IF
    DUP >BI @ TALLY-BI @ CONTAINED-IN IF
        DUP TALLY:|
        DUP +DISS
    THEN
    THEN ;
```

59b ⟨*TRY-DFI* 59b⟩≡ (58d)
```
: TRY-DFI
    DUP IS-DFI OVER IS-DFIs OR IF
    DUP >BI @ TALLY-BI @ CONTAINED-IN IF
        DUP TALLY:|
        DUP +DISS
    THEN
    THEN ;
```

59c ⟨*TRY-FIR* 59c⟩≡ (58d)
```
: TRY-FIR
    DUP IS-FIR IF
    DUP >BI @ CORRECT-R TALLY-BI @ CONTAINED-IN IF
        DUP TALLY:|R
        DUP +DISS
    THEN
    THEN ;
```

59d ⟨*TRY-COMMA* 59d⟩≡ (58d)
```
: TRY-COMMA
    DUP IS-COMMA IF
    DUP >BY @ TALLY-BY @ CONTAINED-IN IF
        DUP TALLY:„
        DUP +DISS
    THEN
    THEN ;
```

Generate bookkeeping such as to correspond with '**DISS**'.

59e ⟨*REBUILD* 59e⟩≡ (58d)
```
: REBUILD
    !TALLY
    DISS? IF
        DISS @+ SWAP !DISS DO  ( Get bounds before clearing)
            I @ TRY-PI TRY-xFI TRY-DFI TRY-FIR TRY-COMMA DROP
        0 CELL+ +LOOP
    THEN ;
```

Discard the last item of the disassembly -- it is either used up or incorrect --. Replace DEA with the proper DEA to inspect from here.

60a ⟨*BACKTRACK* 60a⟩≡ (58d)

```
: BACKTRACK
(   S" BACKTRACKING" TYPE                                    )
    DROP DISS @ @- DISS !
(   DROP DISS @ 0 CELL+ - @                                  )
(   S" Failed at :" TYPE DUP ID. CR                          )
    >NEXT%
(   DISS-                                                    )
    REBUILD ;
```

If the disassembly contains something: '**AT-REST?**' means we have gone full cycle rest->postits->fixups->commaers. Return: the disassembly contains a result.

60b ⟨*RESULT?* 60b⟩≡ (58d)

```
: RESULT? AT-REST? DISS? AND   BAD? 0= AND ;
```

If present, print a result and continue searching for a new last item.

60c ⟨*.RESULT* 60c⟩≡ (58d)

```
: .RESULT
    RESULT? IF
        DISS-VECTOR @ EXECUTE
        DISS-
        REBUILD
    THEN ;
```

Try to expand the current instruction in '**DISS**' by looking whether DEA fits. Leave the next DEA.

60d ⟨*SHOW-STEP* 60d⟩≡ (58d)

```
: SHOW-STEP
    TRY-PI TRY-DFI TRY-xFI TRY-FIR TRY-COMMA
    .RESULT
    >NEXT%
(       DUP ID.                                              )
    BAD? IF BACKTRACK THEN
    BEGIN DUP VOCEND? DISS? AND WHILE BACKTRACK REPEAT ;
```

Show all the instructions present in the assembler vocabulary.

60e ⟨*SHOW-ALL* 60e⟩≡ (58d)

```
: SHOW-ALL ( -- )
    !DISS   !TALLY
    STARTVOC BEGIN
        SHOW-STEP
    DUP VOCEND? UNTIL DROP ;
```

Show all the opcodes present in the assembler vocabulary.

61a ⟨*SHOW-OPCODES* 61a⟩≡ (58d)
```
: SHOW-OPCODES ( -- )
    !DISS   !TALLY
    STARTVOC BEGIN
        DUP IS-PI IF DUP %ID. THEN >NEXT%
    DUP VOCEND? UNTIL DROP ;
```

Show at least all instructions valid for the "OPCODE" given.

61b ⟨*SHOW:* 61b⟩≡ (58d)
```
: SHOW:
    !DISS   !TALLY
    ' DUP BEGIN
        SHOW-STEP
    OVER DISS CELL+ @ - OVER VOCEND? OR UNTIL DROP DROP ;
```

3.5.13 Disassemblers

Disassemblers try to reconstruct an instruction from current bookkeeping. They are similar to tryers but disassemblers take one more aspect into account, a piece of actual code. They do not backtrack but fail.

61c ⟨*disassemblers* 61c⟩≡ (42)
- ⟨*AS-POINTER* 62a⟩
- ⟨*INSTRUCTION* 62b⟩
- ⟨*LATEST-INSTRUCTION* 62c⟩
- ⟨*DIS-PI* 62d⟩
- ⟨*DIS-xFI* 62e⟩
- ⟨*DIS-DFI* 63a⟩
- ⟨*DIS-DFIR* 63b⟩
- ⟨*DIS-FIR* 63c⟩
- ⟨*DIS-COMMA* 63d⟩
- ⟨*.DFI* 64a⟩
- ⟨*.DFIR* 64b⟩
- ⟨*.COMMA-STANDARD* 64c⟩
- ⟨*.COMMA-SIGNED* 64d⟩
- ⟨*.COMMA* 64e⟩
- ⟨ *ID.* 64f⟩
- ⟨*.DISS* 65a⟩
- ⟨*SHOW-MEMORY* 65b⟩
- ⟨*((DISASSEMBLE))* 65c⟩
- ⟨*DIS* 66a⟩
- ⟨*FORCED-DISASSEMBLY* 66b⟩
- ⟨*DISASSEMBLE-RANGE* 66c⟩

Contains the position that is being disassembled.

62a ⟨*AS-POINTER* 62a⟩≡ (61c)
```
VARIABLE AS-POINTER       HERE AS-POINTER !
```

Get the valid part of the INSTRUCTION under examination.

62b ⟨*INSTRUCTION* 62b⟩≡ (61c)
```
: INSTRUCTION  ISS @   ISL @   MC@ ;
```

This is kept up to date during disassembly. It is useful for intelligent disassemblers.

62c ⟨*LATEST-INSTRUCTION* 62c⟩≡ (61c)
```
VARIABLE LATEST-INSTRUCTION
```

These disassemblers are quite similar: if the DEA on the stack is of the right type and if the precondition is fullfilled and if the dissassembly fits, it does the reassuring actions toward the tally as with assembling and add the fixup/posti/commaer to the disassembly struct. Leave the DEA.

62d ⟨*DIS-PI* 62d⟩≡ (61c)
```
: DIS-PI ( dea -- dea )
    DUP IS-PI IF
    AT-REST? IF
    DUP >BI OVER >CNT @  MC@ INVERT
    >R AS-POINTER @ OVER >CNT @  MC@ R>   AND
    OVER >DATA @ = IF
        DUP TALLY:,
        DUP +DISS
        DUP LATEST-INSTRUCTION !
        AS-POINTER @ ISS !
        DUP >CNT @ AS-POINTER +!
    THEN
    THEN
    THEN ;
```

62e ⟨*DIS-xFI* 62e⟩≡ (61c)
```
: DIS-xFI ( dea -- dea )
    DUP IS-xFI IF
    DUP >BI @ TALLY-BI @ CONTAINED-IN IF
    DUP >BI @ INSTRUCTION AND   OVER >DATA @ = IF
    DUP >BA @  CONSISTENT? IF
        DUP TALLY:|
        DUP +DISS
    THEN
    THEN
    THEN
    THEN ;
```

63a ⟨*DIS-DFI* 63a⟩≡ (61c)

```
: DIS-DFI ( dea -- dea )
    DUP IS-DFI OVER IS-DFIs OR IF
    DUP >BI @ TALLY-BI @ CONTAINED-IN IF
    DUP >BA @  CONSISTENT? IF
        DUP TALLY:|
        DUP +DISS
    THEN
    THEN
    THEN ;
```

63b ⟨*DIS-DFIR* 63b⟩≡ (61c)

```
: DIS-DFIR ( dea -- dea )
    DUP IS-DFIR IF
    DUP >BI @ CORRECT-R   TALLY-BI @ CONTAINED-IN IF
    DUP >BA @  CONSISTENT? IF
        DUP TALLY:|R
        DUP +DISS
    THEN
    THEN
    THEN ;
```

63c ⟨*DIS-FIR* 63c⟩≡ (61c)

```
: DIS-FIR ( dea -- dea )
    DUP IS-FIR IF
    DUP >BI @ CORRECT-R   TALLY-BI @ CONTAINED-IN IF
    DUP >BI @ CORRECT-R   INSTRUCTION AND   OVER >DATA @ CORRECT-R = IF
    DUP >BA @  CONSISTENT? IF
        DUP TALLY:|R
        DUP +DISS
    THEN
    THEN
    THEN
    THEN ;
```

63d ⟨*DIS-COMMA* 63d⟩≡ (61c)

```
: DIS-COMMA ( dea -- dea )
    DUP IS-COMMA IF
    DUP >BY @ TALLY-BY @ CONTAINED-IN IF
    DUP >BA @  CONSISTENT? IF
        DUP TALLY:„
        DUP +DISS
    THEN
    THEN
    THEN ;
```

Print a disassembly for the data-fixup DEA.

64a ⟨*.DFI* 64a⟩≡ (61c)

```
: .DFI
    INSTRUCTION   OVER >BI @ AND   OVER >DATA @ RSHIFT   U.
    %ID. ;
```

Print a disassembly for the data-fixup from reverse DEA.

64b ⟨*.DFIR* 64b⟩≡ (61c)

```
: .DFIR
    INSTRUCTION   OVER >BI @ CORRECT-R AND   OVER >DATA @ RSHIFT
    REVERSE-BYTES CORRECT-R U.
    %ID. ;
```

Print a standard disassembly for the commaer DEA.

64c ⟨*.COMMA-STANDARD* 64c⟩≡ (61c)

```
: .COMMA-STANDARD
    AS-POINTER @ OVER >CNT @ MC@ U.
    DUP >CNT @ AS-POINTER +!
    %ID. ;
```

Print a signed disassembly for the commaer DEA.

64d ⟨*.COMMA-SIGNED* 64d⟩≡ (61c)

```
: .COMMA-SIGNED
    AS-POINTER @ OVER >CNT @ MC@ .
    DUP >CNT @ AS-POINTER +!
    %ID. ;
```

Print the disassembly for the commaer DEA, advancing '**AS-POINTER**' past the comma-content.

64e ⟨*.COMMA* 64e⟩≡ (61c)

```
: .COMMA   DUP >DIS @ IF   DUP >DIS @ EXECUTE   ELSE
        .COMMA-STANDARD   THEN ;
```

Print the DEA but with suppression, i.e. ignore those starting in '~'.

64f ⟨ *ID.* 64f⟩≡ (61c)

```
: %~ID. DUP IGNORE? IF DROP ELSE %ID. THEN  ;
```

Print the disassembly '**DISS**'.

65a ⟨*.DISS* 65a⟩≡ (61c)

```
: .DISS   DISS @+ SWAP DO
        I @
        DUP IS-COMMA IF
            .COMMA
        ELSE DUP IS-DFI IF
            .DFI
        ELSE DUP IS-DFIs IF
            .DFI            \ For the moment.
        ELSE DUP IS-DFIR IF
            .DFIR
        ELSE
            %~ID.
        THEN THEN THEN THEN
    0 CELL+ +LOOP ;
```

From **AS-POINTER** show memory because the code there can't be disassembled. Leave incremented **AS-POINTER**.

65b ⟨*SHOW-MEMORY* 65b⟩≡ (61c)

```
VARIABLE I-ALIGNMENT    1 I-ALIGNMENT !   ( Instruction alignment )

: SHOW-MEMORY ( a -- a' )   BEGIN  COUNT C. S"  C, " TYPE
        DUP I-ALIGNMENT @ MOD WHILE  REPEAT ;
```

Dissassemble one instruction from **AS-POINTER** starting at DEA. Based on what is currently left in '**TALLY**'! Leave a **AS-POINTER** pointing after that instruction.

65c ⟨*((DISASSEMBLE))* 65c⟩≡ (61c)

```
: ((DISASSEMBLE)) ( a dea -- a' )
    SWAP
    DUP AS-POINTER !  >R
    3 SPACES
    ( startdea -- ) BEGIN
        DIS-PI DIS-xFI DIS-DFI DIS-DFIR DIS-FIR DIS-COMMA
        >NEXT%
(       DUP ID. S" : " TYPE  DISS-VECTOR @ EXECUTE                          )
    DUP VOCEND? RESULT? OR UNTIL DROP
    RESULT? IF
        .DISS     \ Advances pointer past commaers
        LATEST-INSTRUCTION @ >PRF @ BA-XT !
        R> DROP AS-POINTER @
    ELSE
        R> SHOW-MEMORY
    THEN ;
```

Dissassemble one instruction from ADDRESS using the whole instruction set and starting with a clean slate. Leave an ADDRESS pointing after that instruction.

66a ⟨*DIS* 66a⟩≡ (61c)
```
: (DISASSEMBLE) ( a -- a' )   !DISS !TALLY STARTVOC ((DISASSEMBLE)) ;
: DIS ( x -- )   PAD !  PAD (DISASSEMBLE) ;
```

Forced dissassembly of one instruction from '**AS-POINTER**'. Force interpretation as DEA instruction. This is useful for instructions that are known or hidden by another instruction that is found first.

66b ⟨*FORCED-DISASSEMBLY* 66b⟩≡ (61c)
```
: FORCED-DISASSEMBLY ( dea -- )
    !DISS  !TALLY  AS-POINTER @ SWAP ((DISASSEMBLE)) DROP ;
```

Dissassemble one instruction from address ONE to address TWO.

66c ⟨*DISASSEMBLE-RANGE* 66c⟩≡ (61c)
```
: DISASSEMBLE-RANGE ( first last -- )
    SWAP  BEGIN DUP ADORN-ADDRESS  (DISASSEMBLE) 2DUP > 0= UNTIL  2DROP ;
```

3.5.14 Defining words framework

Close an assembly definition: restore and check.

66d ⟨*defining-words* 43⟩+≡ ◁43
```
: END-CODE
    ?CSP ?EXEC CHECK26 CHECK32 PREVIOUS ; IMMEDIATE

( FIXME : we must get rid of this one )
: ;C POSTPONE END-CODE S" WARNING: get rid of C;" TYPE CR ; IMMEDIATE

\ The following two definitions must *NOT* be in the assembler wordlist.
PREVIOUS DEFINITIONS DECIMAL

ALSO ASSEMBLER
⟨CODE 66e⟩
⟨;CODE 66f⟩
```

Define "word" using assembly instructions up till **END-CODE**. One could put a '**SMUDGE**' in both.

66e ⟨*CODE* 66e⟩≡ (66d)
```
: CODE
    ?EXEC ASM-CREATE POSTPONE ASSEMBLER !TALLY !CSP ; IMMEDIATE
```

Like '**DOES>**' but assembly code follows, closed by **END-CODE**.

66f ⟨*;CODE* 66f⟩≡ (66d)
```
: ;CODE
    ?CSP   POSTPONE (;CODE)   POSTPONE [   POSTPONE ASSEMBLER ; IMMEDIATE
```

3.5.15 Conveniences

Abbreviations for interactive use. In the current dictionary.

67 ⟨*conveniences* 67⟩≡ (43)

```
: DDD ( a -- a' )   (DISASSEMBLE) ;
: D-R ( first last -- )   DISASSEMBLE-RANGE ;
```

3.5.16 Notes

1. We use the abstraction of a DEA "dictionary entry address". aqa "xt". Return the DEA from "word". A DEA is an address that allows to get at header data like flags and names. In ciforth an xt will do.

3.6 Assembler wrapper

This file hot patches some words in the prelude of asgen.frt. It must be loaded after asgen.frt.

68a ⟨*aswrap.frt* 68a⟩≡ (22)

```
( $Id: aswrap.frt,v 1.21 2009/03/26 19:40:39 albert Exp $ )
( Copyright{2000}: Albert van der Horst, HCC FIG Holland by GNU Public License)
( Uses Richard Stallmans convention. Uppercased word are parameters.    )

⟨HOT-PATCH 68b⟩
⟨CODE-LENGTH 69a⟩
⟨SECTION-REGISTRY 69b⟩
⟨CURRENT-SECTION 69c⟩
⟨SECTION 69d⟩
⟨DEFAULT-SECTION 69e⟩
⟨'AS-HERE 70a⟩
⟨TARGET-END 70b⟩
⟨PLAUSIBLE-LABEL? 70c⟩
⟨HOST-END 70d⟩
⟨ORG 70e⟩
⟨HOST>TARGET 70f⟩
⟨TARGET>HOST 70g⟩
⟨FILE>TARGET 70h⟩
⟨TARGET>FILE 70i⟩

\ Abbreviation.
[DEFINED] ForSwiftForth NOT [IF]
    ' TARGET>HOST ALIAS th
[THEN]

⟨SUB-SECTION 70j⟩
⟨'AP 70k⟩
⟨SWAP-AS 71a⟩
⟨'AS-ALLOT 71b⟩
⟨'AS-C, 71c⟩
⟨RESB 71d⟩
⟨RES-TIL 71e⟩
⟨AS-ALIGN 71f⟩
```

Copy the behaviour of the latest definition into "name" affecting all words already using that word.

68b ⟨*HOT-PATCH* 68b⟩≡ (68a)

```
: HOT-PATCH ( xt -- ) ' >BODY ! ;
```

Length of the code buffer.

69a ⟨*CODE-LENGTH* 69a⟩≡ (68a)
```
VARIABLE CODE-LENGTH 2000000 CODE-LENGTH !
```

A bag with the DEA's of all segments.

69b ⟨*SECTION-REGISTRY* 69b⟩≡ (68a)
```
100 BAG SECTION-REGISTRY
```

Current segment pointer. Create section pointers with **FILE-OFFSET TARGET-START** and **CODE-SPACE**

69c ⟨*CURRENT-SECTION* 69c⟩≡ (68a)
```
0 VALUE CURRENT-SECTION

: SECTION-FIELD ( u size -- u' ) CREATE OVER , +
    DOES> ( -- a ) @ CURRENT-SECTION + ;

0
    2 CELLS SECTION-FIELD SECTION-RESERVED
    1 CELLS SECTION-FIELD CP \ The local dictionary pointer ("code pointer")
    1 CELLS SECTION-FIELD 'CODE-SPACE \ Start of the code space
    1 CELLS SECTION-FIELD 'TARGET-START \ Return corresponding target address.
    1 CELLS SECTION-FIELD 'FILE-OFFSET \ Return corresponding files address.
CONSTANT |SECTION|

: CODE-SPACE ( -- a ) 'CODE-SPACE @ ;
: -ORG- ( a -- ) 'TARGET-START ! ;
: TARGET-START ( -- a ) 'TARGET-START @ ;
: FILE-OFFSET ( -- a ) 'FILE-OFFSET @ ;
```

Create section with **FILE-OFFSET TARGET-START**. Assign ample code space. Leave it current.

69d ⟨*SECTION* 69d⟩≡ (68a)
```
: ((SECTION)) ( file target code -name- )
    SAVE-INPUT CREATE HERE DUP >R DUP SECTION-REGISTRY BAG+!
    |SECTION| DUP ALLOT ERASE RESTORE-INPUT THROW BL WORD $@ $, R@ >NFA !
    CURRENT-SECTION R@ >LFA ! R> TO CURRENT-SECTION DUP CP !
    'CODE-SPACE ! 'TARGET-START ! 'FILE-OFFSET !
    DOES> TO CURRENT-SECTION ;
: (SECTION) ( file target -- ) CODE-LENGTH @ ALLOCATE THROW ((SECTION)) ;
CREATE 'SECTION ' (SECTION) DUP , , : SECTION 'SECTION @ EXECUTE ;
```

Define at least one segment lest the user forgets.

69e ⟨*DEFAULT-SECTION* 69e⟩≡ (68a)
```
: DEFAULT-SECTION ( -- )
    0 \ File start address
    0 \ Target start address
    s" SECTION the-default-section" EVALUATE ;
DEFAULT-SECTION
```

'**HERE**' such as used in assembly.

70a ⟨*'AS-HERE* 70a⟩≡ (68a)
```
:NONAME ( -- a ) CP @ ; HOT-PATCH 'AS-HERE
```

Use only while disassembling. Return the END of the file as a target address (non-inclusive).

70b ⟨*TARGET-END* 70b⟩≡ (68a)
```
: TARGET-END ( -- a ) TARGET-START CP @ CODE-SPACE - + ;
```

For ADDR return "it is a pointing into the target space"

70c ⟨*PLAUSIBLE-LABEL?* 70c⟩≡ (68a)
```
: PLAUSIBLE-LABEL? ( a -- f ) TARGET-START TARGET-END WITHIN ;
```

Use only while disassembling. The end of the code area while disassembling: a host address.

70d ⟨*HOST-END* 70d⟩≡ (68a)
```
: HOST-END ( -- a ) CP @ ;
```

Associate target ADDRESS with start of '**CODE-BUFFER**'. The valid range from the code buffer goes to '**CP @**' and is not affected.

Associate ADDRESS with the start of '**CODE-SPACE**'.

70e ⟨*ORG* 70e⟩≡ (68a)
```
: ORG ( a -- ) -ORG- CODE-SPACE CP ! ;
```

Convert host memory ADDRESS. Leave target memory ADDRESS.

70f ⟨*HOST>TARGET* 70f⟩≡ (68a)
```
: HOST>TARGET ( a -- a' ) CODE-SPACE - TARGET-START + ;
```

Convert target memory ADDRESS. Leave host memory ADDRESS.

70g ⟨*TARGET>HOST* 70g⟩≡ (68a)
```
: TARGET>HOST ( a -- a' ) TARGET-START - CODE-SPACE + ;
```

Convert file memory ADDRESS. Leave target memory ADDRESS.

70h ⟨*FILE>TARGET* 70h⟩≡ (68a)
```
: FILE>TARGET ( a -- a' ) FILE-OFFSET - TARGET-START + ;
```

Convert target memory ADDRESS. Leave file memory ADDRESS.

70i ⟨*TARGET>FILE* 70i⟩≡ (68a)
```
: TARGET>FILE ( a -- a' ) TARGET-START - FILE-OFFSET + ;
```

70j ⟨*SUB-SECTION* 70j⟩≡ (68a)
```
: SUB-SECTION ( a -name- ) DUP TARGET>FILE
    OVER ROT TARGET>HOST CP @ >R ((SECTION)) R> CP ! ;
```

Instruction pointer in assembly. View used in branches etc.

70k ⟨*'AP* 70k⟩≡ (68a)
```
:NONAME ( -- a ) CP @ HOST>TARGET ; HOT-PATCH '_AP_
```

Swap dictionary pointer back and forth to assembler area.

71a ⟨*SWAP-AS* 71a⟩≡ (68a)

```
[DEFINED] ForCiForth
[DEFINED] ForGForth OR [IF]
    : SWAP-AS ( -- ) CP @ DP @ CP ! DP ! ;
[THEN]
[DEFINED] ForSwiftForth [IF]
    : SWAP-AS ( -- ) CP @ H @ CP ! H ! ;
[THEN]
```

Wrapper for '**ALLOT**' such as used in assembly.

71b ⟨*'AS-ALLOT* 71b⟩≡ (68a)

```
:NONAME ( n -- ) CP +! ; HOT-PATCH 'AS-ALLOT
```

Only Needed. Maybe '**CP C! 1 CP +!**'. Wrapper for '**C,**' such as used in assembly.

71c ⟨*'AS-C,* 71c⟩≡ (68a)

```
:NONAME ( c -- ) CP @ 1 AS-ALLOT C! ; HOT-PATCH 'AS-C,
```

Reserve X bytes, without specifying a content.

71d ⟨*RESB* 71d⟩≡ (68a)

```
: RESB ( u -- ) AS-HERE OVER AS-ALLOT SWAP ERASE ;
```

Reserve bytes till target ADDRES. (Compare '**ORG**'.)

71e ⟨*RES-TIL* 71e⟩≡ (68a)

```
: RES-TIL ( a -- ) _AP_ - AS-ALLOT ;
```

Align to a target address, that is multiple of N.

71f ⟨*AS-ALIGN* 71f⟩≡ (68a)

```
: AS-ALIGN ( n -- ) _AP_ BEGIN 2DUP SWAP MOD WHILE 1+ REPEAT RES-TIL DROP ;
```

3.7 80386 Assembler

72 ⟨*asi386.frt* 72⟩≡ (22)

```
( $Id: asi386.frt,v 4.23 2005/05/09 01:00:40 albert Exp $ )
( Copyright{2000}: Albert van der Horst, HCC FIG Holland by GNU Public License)

ALSO ASSEMBLER DEFINITIONS HEX
```

⟨*additions* 74a⟩

```
( ############### 80386 ASSEMBLER PROPER ################################# )
( The decreasing BY means that a decompiler hits them in the right      )
( order to reassemble.                                                  )
( Fields: a disassembly XT,      LENGTH to comma, the BA BY information )
( and the XT that puts data in the dictionary.                          )
( Where there is a placeholder ‘‘_” the execution token is filled in   )
( later. )
(   CNT                                                                 )
( XT   BA   BY     XT-AS          NAME                                  )
  0 2  0000 00100 ’ (W,) COMMAER OW,    ( obligatory word     )
  0 4  8000 00080 ’ (L,) COMMAER (RL,)  ( cell relative to IP )
  0 2  4000 00080 ’ (W,) COMMAER (RW,)  ( cell relative to IP )
  0 1  0000 00040 ’ AS-C, COMMAER (RB,) ( byte relative to IP )
  0 2  0000 00020 ’ (W,) COMMAER SG,    (  Segment: WORD      )
  0 1  0000 00010 ’ AS-C, COMMAER P,    ( port number ; byte     )
  0 1  0000 00008 ’ AS-C, COMMAER IS,   ( Single -obl-  byte )
  0 4 20002 00004 ’ (L,) COMMAER IL,    ( immediate data : cell)
  0 2 10002 00004 ’ (W,) COMMAER IW,    ( immediate data : cell)
  0 1  0001 00004 ’ AS-C, COMMAER IB,   ( immediate byte data)
  0 4  8008 00002 ’ (L,) COMMAER L,     ( immediate data : address/offset )
  0 2  4008 00002 ’ (W,) COMMAER W,     ( immediate data : address/offset )
  0 1  0004 00002 ’ AS-C, COMMAER B,    ( immediate byte : address/offset )
  _ 1  0000 00001 _    COMMAER SIB„ ( An instruction with in an instruction )

( Meaning of the bits in TALLY-BA :                                     )
( Inconsistent:  0001 OPERAND IS BYTE     0002 OPERAND IS CELL  W/L     )
(                0004 OFFSET  IS BYTE     0008 OFFSET  IS CELL  W/L     )
( By setting 0020 an opcode can force a memory reference, e.g. CALLFARO )
(               0010 Register op         0020 Memory op                 )
(               0040 ZO|                 0080 [BP]% {16} [BP] [BP  {32} )
(  sib:         0100 no ..             0200 [AX +8*| DI]                )
(  logical      0400 no ..             0800 Y| Y’| Z| Z’|               )
(  segment      1000 no ..             2000 ES| ..                      )
(  AS:          4000 16 bit Addr       8000 32 bit Address              )
(  OS:         10000 16 bit Op        20000 32 bit Operand              )
(  Use debug   40000 no ..            80000 CR0 ..DB0                   )
```

```
 (  FP:        100000 FP-specific     200000 Not FP                       )

 ( Names *ending* in percent BP|% -- not BP'| the prime registers -- are )
 ( only valid for 16 bits mode, or with an address overwite. Use W, L,   )
 ( appropriately.                                                        )

8200 0 38 T!R
  08 00 8 FAMILY|R AX] CX] DX] BX] 0] BP] SI] DI]
 8200 0 0C0 T!R
  40 00 4 FAMILY|R  +1* +2* +4* +8*
 8200 0 07000001 T!
  01 00 8 FAMILY|R [AX [CX [DX [BX [SP -- [SI [DI
 8280 00 01000007 05 FIR [BP   ( Fits in the hole, but disallow ZO| )
 8248 02 01000007 05 FIR [MEM  ( Fits in the hole, but requires ZO| )

4120 0 07 T!R
   01 00 8
     FAMILY|R [BX+SI]% [BX+DI]% [BP+SI]% [BP+DI]% [SI]% [DI]% -- [BX]%
 40A0 0000 07 06 FIR [BP]%  ( Fits in the hole, safe inconsistency check)
 8120 0 07 T!R
  01 00 4 FAMILY|R [AX] [CX] [DX] [BX]
 8120 01 07 04 FIR ~SIB|   ( Fits in the hole, but requires ~SIB, )
 81A0 00 07 05 FIR [BP]   ( Fits in the hole, but disallow ZO| )
 8120 0 07 T!R  01 06 2 FAMILY|R [SI] [DI]

 200111 0 07 T!R  01 00 8 FAMILY|R AL| CL| DL| BL| AH| CH| DH| BH|
 200112 0 07 T!R
  01 00 8 FAMILY|R AX| CX| DX| BX| SP| BP| SI| DI|
 0160 00 0C0 00 FIR      ZO|
 0124 02 0C0 40 FIR      BO| 0128 02 0C0 80 FIR      XO|
 200110 00 0C0 0C0 FIR      R|
 204048 02 0C7 06 FIR      MEM|% ( Overrules ZO| [BP]% )
 208108 02 0C7 05 FIR      MEM| ( Overrules ZO| [BP] )

 241101 0000 38 T!R
  08 00 8 FAMILY|R AL'| CL'| DL'| BL'| AH'| CH'| DH'| BH'|
 241102 0000 38 T!R  08 00 8 FAMILY|R AX'| CX'| DX'| BX'| SP'| BP'| SI'| DI'|
 242100 0000  38 T!R   08 00 6 FAMILY|R ES| CS| SS| DS| FS| GS|
 280002 0000 38010000 T!   ( 3)
  08 00 5 FAMILY|R CR0| -- CR2| CR3| CR4|                 ( 3)
  0008 0100 8 FAMILY|R DR0| DR1| DR2| DR3| DR4| DR5| DR6| DR7| ( 3)

 200000 0000 0200 T!R  0200 00 2 FAMILY|R F| T|
 240401 0000 0100 0000 FIR B|
 240402 0000 0100 0100 FIR X|
```

⟨ *SIB,* 78b⟩
⟨*two-fixup-operands* 75⟩
⟨*one-fixup-operands* 76a⟩
⟨*no-fixup-operands* 76b⟩
⟨*special-fixups* 77⟩
⟨*no-fixups* 78a⟩
⟨*SIB-bytes* 78c⟩
⟨*AS:,OS:,* 74b⟩

```
( ############## 80386 ASSEMBLER PROPER END ############################ )
```
⟨*Rx,alt* 74c⟩
⟨*BITS* 74d⟩

```
BITS-32
PREVIOUS DEFINITIONS DECIMAL
( ############## 8086 ASSEMBLER POST ################################### )
```

These definitions are such that they work regardless of the endianness of the host. Lay down word (16 bits) and long (32 bits) constants.

74a ⟨*additions* 74a⟩≡ (72)
```
   : (W,) lsbyte, lsbyte, DROP ;
   : (L,) lsbyte, lsbyte, lsbyte, lsbyte, DROP ;
```

Fill in the transformation to TALLY-BA for **`AS:,`** and **`OS:,`** This flags them as prefixes. The toggle inverts the 16 and 32 bits at the same time.

74b ⟨*AS:,OS:,* 74b⟩≡ (72)
```
  :NONAME   TALLY-BA  C000 TOGGLE ;  ' AS:, >BODY >PRF !
  :NONAME   TALLY-BA 30000 TOGGLE ;  ' OS:, >BODY >PRF !
```

You may want to use these always instead of **`(Rx,)`**

74c ⟨*Rx,alt* 74c⟩≡ (72)
```
  : RB, _AP_ 1 + - (RB,) ;    ' .COMMA-SIGNED   ' (RB,) >BODY >DIS !
  : RW, _AP_ 2 + - (RW,) ;    ' .COMMA-SIGNED   ' (RW,) >BODY >DIS !
  : RL, _AP_ 4 + - (RL,) ;    ' .COMMA-SIGNED   ' (RL,) >BODY >DIS !
```

Require instructions as per a 32 resp. 16 bits segment.

74d ⟨*BITS* 74d⟩≡ (72)
```
  : BITS-32   28000 BA-DEFAULT ! ;
  : BITS-16   14000 BA-DEFAULT ! ;
```

3.7.1 Two fixup operands

75 ⟨*two-fixup-operands* 75⟩≡ (72)

```
041000 0000 FF03 T!
 0008 0000 8 2FAMILY, ADD, OR, ADC, SBB, AND, SUB, XOR, CMP,
041000 0000 FF01 T!
 0002 0084 2 2FAMILY, TEST, XCHG,
041000 0000 FF03 0088 2PI MOV,
1022 0 FF00 008D 2PI LEA,
1022 0 FF00 T!   0001 00C4 2 2FAMILY, LES, LDS,
1022 0 FF00 0062 2PI BOUND,  ( 3)
1002 0 FF00 0063 2PI ARPL,   ( 3)
1002 04 FF00 0069 2PI IMULI, ( 3)
1002 08 FF00 006B 2PI IMULSI, ( 3)
1002 0 FF0000 T! 0100 00020F 2 3FAMILY, LAR, LSL, ( 3)
1002 0 FF0000 T! 0800 00A30F 4 3FAMILY, BT, BTS, BTR, BTC, ( 3)
1002 0 FF0000 T! 0800 00A50F 2 3FAMILY, SHLD|C, SHRD|C,    ( 3)
1002 0 FF0000 T! 0100 00BC0F 2 3FAMILY, BSF, BSR,          ( 3)
1002 08 FF0000 T! 0800 00A40F 2 3FAMILY, SHLDI, SHRDI,    ( 3)
1022 0 FF0000 T! 0100 00B20F 4 3FAMILY, LSS, -- LFS, LGS, ( 3)
1501 0 FF0000 T! 0800 00B60F 2 3FAMILY, MOVZX|B, MOVSX|B,  ( 3)
1502 0 FF0000 T! 0800 00B70F 2 3FAMILY, MOVZX|W, MOVSX|W,  ( 3)
1002 0 FF0000 00AF0F 3PI IMUL,                      ( 3)
```

3.7.2 One fixup operands

It is dubious but fairly intractible whether the logical operation with sign extended bytes belong in the 386 instruction set. They are certainly there in the Pentium.

76a ⟨*one-fixup-operands* 76a⟩≡ (72)

```
0 04 C701 00C6 2PI MOVI,
0012 0 0007 T!   0008 40 4 1FAMILY, INC|X, DEC|X, PUSH|X, POP|X,
0012 0 0007 90 1PI XCHG|AX,
0011 04 0007 B0 1PI MOVI|B,
0012 04 0007 B8 1PI MOVI|X,
0 04 C701 T!
 0800 0080 8 2FAMILY, ADDI, ORI, ADCI, SBBI, ANDI, SUBI, XORI, CMPI,
0002 08 C700 T!
 0800 0083 8 2FAMILY, ADDSI, ORSI, ADCSI, SBBSI, ANDSI, SUBSI, XORSI, CMPSI,
0000 0 C701 T!
 0800 10F6 6 2FAMILY, NOT, NEG, MUL|AD, IMUL|AD, DIV|AD, IDIV|AD,
 0800 00FE 2 2FAMILY, INC, DEC,
0 04 C701 00F6 2PI TESTI,
0002 0 C700 008F 2PI POP,
0002 0 C700 30FF 2PI PUSH,
0002 0 C700 T!  1000 10FF 2 2FAMILY, CALLO, JMPO,
0022 0 C700 T!  1000 18FF 2 2FAMILY, CALLFARO, JMPFARO,
0002 08 C70000 T!  080000 20BA0F 4 3FAMILY, BTI, BTSI, BTRI, BTCI, ( 3)
0002 0 C70000 T! ( It says X but in fact W : descriptor mostly - ) ( 3)
  080000 00000F 6 3FAMILY, SLDT, STR, LLDT, LTR, VERR, VERW,  ( 3)
  100000 20010F 2 3FAMILY, SMSW, LMSW,       ( 3)
0022 0 C70000 T! ( It says X but in fact memory of different sizes) ( 3)
  080000 00010F 4 3FAMILY, SGDT, SIDT, LGDT, LIDT, ( 3)
```

3.7.3 No fixup operands

76b ⟨*no-fixup-operands* 76b⟩≡ (72)

```
0001 0 02000001 00 FIR B'|
0002 0 02000001 01 FIR X'|
0008 02 0201 T!    0002 A0 2 1FAMILY, MOV|TA, MOV|FA,
0 04 0201 T!
 0008 04 8 1FAMILY, ADDI|A, ORI|A, ADCI|A, SBBI|A, ANDI|A, SUBI|A, XORI|A, CMPI|A,
0000 04 0201 00A8 1PI TESTI|A,
0000 0 0201 T!  0002 A4 6 1FAMILY, MOVS, CMPS, -- STOS, LODS, SCAS,
0 10 0201 T!   0002 E4 2 1FAMILY, IN|P, OUT|P,
0 00 0201 T!   0002 EC 2 1FAMILY, IN|D, OUT|D,
0 00 0201 T!   0002 6C 2 1FAMILY, INS, OUTS,     ( 3)
```

3.7.4 Special fixups

77 ⟨*special-fixups* 77⟩≡ (72)

```
0800     0000 01000001 T!R     01 00 2 FAMILY|R Y| N|
0800     0000 0400000E T!R     02 00 8 FAMILY|R O| C| Z| CZ| S| P| L| LE|
0800 40 050F 0070 1PI J,

2102 0 FF02 008C 2PI MOV|SG,

0000 0 02000200 0000 FIR 1|   0000 0 02000200 0200 FIR V|          ( 3)
0100 0 2C703 T! ( 20000 is a lockin for 1| V|)                   ( 3)
 0800 00D0 8 2FAMILY, ROL, ROR, RCL, RCR, SHL, SHR, -- SAR,  ( 3)
0000 8 C701 T!  0800 00C0 8 2FAMILY, ROLI, RORI, RCLI, RCRI, SHLI, SHRI, -- SARI,  ( 3)
80012 0000 3F0300 C0200F 3PI  MOV|CD,  ( 3)

0800 80 50F00 800F 2PI J|X,                                        ( 3)
0800 0 0100 T!R  0100 0000 2 FAMILY|R Y'| N'|                       ( 3)
0800 0 0E00 T!R  0200 0000 8 FAMILY|R O'| C'| Z'| CZ'| S'| P'| L'| LE'| ( 3)
0901 0 C70F00 00900F 3PI SET,  ( 3)
```

3.7.5 No fixups

78a ⟨*no-fixups* 78a⟩≡ (72)

```
2000 0000 0 T!  0008 06 4 1FAMILY, PUSH|ES, PUSH|CS, PUSH|SS, PUSH|DS,
2000 0000 0 T!  0008 07 4 1FAMILY, POP|ES, -- POP|SS, POP|DS,

0001 04 0000 T!    0001 D4 2 1FAMILY, AAM, AAD, 0001 04 0000 0CD 1PI INT,
0008 22 0000 09A 1PI CALLFAR,
0008 22 0000 0EA 1PI JMPFAR,
0 0100 0000 T!   0008 C2 2 1FAMILY, RET+, RETFAR+,
0004 80 0000 T!   0001 E8 2 1FAMILY, CALL, JMP,
0 40 0000 EB 1PI JMPS,
0 40 0000 T!   0001 E0 4 1FAMILY, LOOPNZ, LOOPZ, LOOP, JCXZ,
0000 0 0000 T!
   0008   0026 4 1FAMILY, ES:, CS:, SS:, DS:,
   0008   0027 4 1FAMILY, DAA, DAS, AAA, AAS,
   0001   0098 8 1FAMILY, CBW, CWD, -- WAIT, PUSHF, POPF, SAHF, LAHF,
   0008   00C3 2 1FAMILY, RET,  RETFAR,
   0001   00CC 4 1FAMILY, INT3, -- INTO, IRET,
   0001   00F0 6 1FAMILY, LOCK, -- REPNZ, REPZ, HLT, CMC,
   0001   00F8 6 1FAMILY, CLC, STC, CLI, STI, CLD, STD,
   0001   0060 2 1FAMILY, PUSH|ALL, POP|ALL, ( 3)
   0001   0064 4 1FAMILY, FS:, GS:, OS:, AS:, ( 3)
 0100 A00F 3 2FAMILY, PUSH|FS, POP|FS, CPUID,
 0100 A80F 2 2FAMILY, PUSH|GS, POP|GS, ( RSM,)
  0002 04 0000   0068 1PI PUSHI|X,  ( 3)
  0001 04 0000   006A 1PI PUSHI|B,  ( 3)
  0001 0104 0000   00C8 1PI ENTER, ( 3)
      0000 0 00   00C9 1PI LEAVE, ( 3)
      0000 0 00   00D7 1PI XLAT,  ( 3)
      0000 0 00 060F 2PI CLTS,  ( 3)
```

3.7.6 Handling the SIB byte

These must be found last

78b ⟨ *SIB,* 78b⟩≡ (72)

```
0600 0 01FF 0000 1PI ~SIB,
```

Handle a 'sib' bytes as an instruction-within-an-instruction. This is really straightforward, we say the sib commaer is a sib instruction. as per -- error checking omitted -- "**`10000 ' ~SIB, >CFA COMMAER SIB,„`**". All the rest is to nest the state in this recursive situation: Leaving BY would flag commaers to be done after the sib byte as errors.

78c ⟨*SIB-bytes* 78c⟩≡ (72)

⟨*(SIB),„* 79a⟩
⟨*DIS-SIB* 79b⟩
⟨*SIBfixups* 79c⟩

Handle bad bits by hand, prevent resetting of '**TALLY-BA**' which could switch 16/32 bits modes. 0900 are the bad bits conflicting with **~SIB,**. Fill in deferred data creation action.

79a ⟨*(SIB)„* 79a⟩≡ (78c)
```
: (SIB)„   TALLY-BY @   0 TALLY-BY !
    CHECK32 TALLY-BA @ 0900 INVERT AND TALLY-BA !   ['] NOOP BA-XT !
    ~SIB,   TALLY-BY !   ;
' (SIB)„  ' SIB„ >BODY >DATA !
```

Disassemble the sib byte where the disassembler sits now. '**FORCED-DISASSEMBLY**' takes care itself of incrementing the disassembly pointer. Fill in deferred disassembler action.

79b ⟨*DIS-SIB* 79b⟩≡ (78c)
```
: DIS-SIB DROP
    LATEST-INSTRUCTION @        \ We don't want sib visible.
    [ ' ~SIB, >BODY ] LITERAL FORCED-DISASSEMBLY
    LATEST-INSTRUCTION ! ;
' DIS-SIB    ' SIB„ >BODY >DIS !
```

Redefine some fixups, such that the user may say "**[AX**" instead of "**~SIB| SIB„ [AX**". Note that the disassembly is made to look like the same. The **~SIB|** and the **~SIB,** inside the **SIB„** are print-suppressed.

79c ⟨*SIBfixups* 79c⟩≡ (78c)
```
: [AX   ~SIB| SIB„ [AX ;
: [SP   ~SIB| SIB„ [SP ;
: [CX   ~SIB| SIB„ [CX ;
: [BP   ~SIB| SIB„ [BP ;
: [DX   ~SIB| SIB„ [DX ;
: [SI   ~SIB| SIB„ [SI ;
: [BX   ~SIB| SIB„ [BX ;
: [DI   ~SIB| SIB„ [DI ;
: [MEM  ~SIB| SIB„ [MEM ;
```

3.8 Access words

Contains access words and other little utilities that complement ciasdis. In particular regarding unexpected missing words.

79d ⟨*access.frt* 79d⟩≡ (22)
```
( $Id: access.frt,v 1.6 2005/01/04 19:20:35 albert Exp $ )
( Copyright{2000}: Albert van der Horst, HCC FIG Holland by GNU Public License)
( Uses Richard Stallmans convention. Uppercased word are parameters.    )

ALSO ASSEMBLER
: B@ TARGET>HOST C@ ;
: W@ TARGET>HOST 2 MC@ ;
: L@ TARGET>HOST 4 MC@ ;
PREVIOUS
```

3.9 Handle labels

80a ⟨*labelas.frt* 80a⟩≡ (22)

```
( $Id: labelas.frt,v 1.17 2009/03/26 19:40:39 albert Exp $ )
( Copyright{2000}: Albert van der Horst, HCC FIG Holland by GNU Public License)
( Uses Richard Stallmans convention. Uppercased word are parameters.    )

[DEFINED] ForCiForth [IF]
    REQUIRE BAG             \ Simple bag facility
    REQUIRE DO-BAG          \ More advanced bag facility
    REQUIRE POSTFIX
[THEN]

⟨FIX-DEA 80b⟩
⟨BACKSPACE-IN 81a⟩
⟨FIX-NMB 81b⟩
⟨ERROR10 81c⟩
⟨ERROR12 81d⟩

[DEFINED] ForCiForth [IF]
    REQUIRE OLD:
[THEN]

⟨?ERROR-FIXING 81e⟩
⟨RESET-SECTION 81f⟩
⟨PASSES 82a⟩
⟨'LABELS 82b⟩
⟨IS-A-LABEL? 82c⟩
⟨KNOWN-LABEL? 82d⟩

[DEFINED] ForCiForth [IF]
    ⟨:PREFIX 82e⟩
[THEN]

⟨constant-data 83a⟩
```

Make sure undefined labels don't fool up the first pass of the assembly. Replace not found FLAG, by a valid DEA with the stack effect of a label. The result is that unknown words are compiled as a _, i.e. it generates a don't care value. Supposedly these are labels that have not been defined yet. Go on compiling. Loading the same code another time will give correct code.

80b ⟨*FIX-DEA* 80b⟩≡ (80a)

```
: FIX-DEA DROP ['] _ ;
```

Backspace a character, but not if we are at the end of input. We find out by trial reading another character, then back two up.

81a ⟨*BACKSPACE-IN* 81a⟩≡ (80a)
```
[DEFINED] ForSwiftForth
[DEFINED] ForGForth OR [IF]
     : BACKSPACE-IN   -2 >IN +! 0 ;
[THEN]
[DEFINED] ForCiForth [IF]
    : BACKSPACE-IN   IN[] IF -2 IN +! THEN DROP ;
[THEN]
```

Make sure undefined labels that looks like numbers, don't fool up the first pass of the assembly. Not that we endorse the idea to name labels like **250HUP**. All of a word may have been scanned, so before using **(WORD)**, we backspace one char. Afterwards we backspace again, such that the number routine we return to concludes it is ready. We leave some random number, which is okay, but it must be single precision!

81b ⟨*FIX-NMB* 81b⟩≡ (80a)
```
[DEFINED] ForSwiftForth
[DEFINED] ForGForth OR [IF]
    : FIX-NMB   -1 >IN +!  BL WORD DROP  BACKSPACE-IN ;
[THEN]
[DEFINED] ForCiForth [IF]
    : FIX-NMB   -1 IN +!  (WORD) 2DROP  BACKSPACE-IN   0 DPL ! ;
[THEN]
```

If FLAG we have a misspelled number, skip its remainder.

81c ⟨*ERROR10* 81c⟩≡ (80a)
```
: ERROR10 ( f n -- )   DROP IF  FIX-NMB  THEN ;
```

If FLAG we have an unknown word, treat it as a label.

81d ⟨*ERROR12* 81d⟩≡ (80a)
```
: ERROR12 ( f n -- )   DROP IF  FIX-DEA  THEN ;
```

Replacement for **?ERROR**, if FLAG, give error NUMBER. Fix up errors, see **FIX-NMB** and **FIX-DEA**.

81e ⟨*?ERROR-FIXING* 81e⟩≡ (80a)
```
: ?ERROR-FIXING ( f n -- )
    DUP 10 = IF  ERROR10  ELSE
    DUP 12 = IF  ERROR12  ELSE
        (?ERROR)
    THEN THEN ;
```

Ignore **FILE-OFFSET** and **TARGET** address. Make section "name" current, and reset its allocation pointer. Like '**SECTION**' but this behaviour is appropriate for the second pass.

81f ⟨*RESET-SECTION* 81f⟩≡ (80a)
```
: RESET-SECTION ( file target -- )   2DROP  BL WORD COUNT EVALUATE  CODE-SPACE CP ! ;
```

Ignore undefined labels during first pass ... Define section in the first pass ... but just start section, and have normal errors in the second pass.

82a ⟨*PASSES* 82a⟩≡ (80a)
```
: FIRSTPASS ( -- )   S" FIRSTPASS " TYPE
    ['] ?ERROR-FIXING '?ERROR !
    'SECTION RESTORED ;

: SECONDPASS ( -- )   S" SECONDPASS " TYPE
    ['] RESET-SECTION 'SECTION !
    '?ERROR RESTORED ;
```

All labels link back to here.

82b ⟨*'LABELS* 82b⟩≡ (80a)
```
CREATE 'LABELS  HERE ,
```

For NAME: "name REPRESENTS a label."

82c ⟨*IS-A-LABEL?* 82c⟩≡ (80a)
```
: IS-A-LABEL? ( a n -- f )   GET-CURRENT SEARCH-WORDLIST DUP IF
        SWAP >BODY  BEGIN  >LFA ['] @ CATCH IF  DROP 0= EXIT
            THEN  ?DUP WHILE  DUP 'LABELS = IF  DROP EXIT
                THEN  REPEAT  0=  THEN ;
```

For NAME: NAME and "it is a KNOWN label." We don't need to define it if there is already a label of that name. If it has not the value of the program counter we must report a phase error.

82d ⟨*KNOWN-LABEL?* 82d⟩≡ (80a)
```
: KNOWN-LABEL? ( a n -- a n f )   2DUP IS-A-LABEL? >R
    R@ IF  2DUP GET-CURRENT SEARCH-WORDLIST IF
            EXECUTE _AP_ <> IF  S" ERROR: phase error defining label "
                TYPE  2DUP TYPE  CR
    THEN THEN THEN  R> ;
```

ciForth specific

Make a denotation for labels. They look like '**:LABEL**'. Put '**:**' in the **ONLY** wordlist, such that it doesn't interfere with the normal semicolon. A word starting with a '**:**' is a label definition denotation. The part after the '**:**' may be defined already, but if it is a label it must have the value of the current program counter. So it is possible to redefine words as labels (heed the warnings). This is very tricky, but the assembler programmer must not be restricted by what words are in Forth. Note: this is actually an abuse of the denotation mechanism.

82e ⟨*:PREFIX* 82e⟩≡ (80a)
```
'ONLY >WID CURRENT !  \ Making ONLY the CONTEXT is dangerous! This will do.
: : (WORD)
    KNOWN-LABEL? IF 2DROP ELSE 2>R _AP_ 2R> POSTFIX CONSTANT THEN ;
PREFIX IMMEDIATE DEFINITIONS
```

3.9.1 Handle constant data

Handle constant data in assembler.

83a ⟨*constant-data* 83a⟩≡ (80a)
```
⟨DX-SET 83b⟩
⟨GET-DX-SET 83c⟩
⟨C,-DX-SET 83d⟩
⟨db 83e⟩
ALSO ASSEMBLER
⟨W,-DX-SET 83f⟩
⟨dw 84a⟩
⟨L,-DX-SET 84b⟩
⟨dl 84c⟩
⟨(,) 84d⟩
⟨,-DX-SET 84e⟩
⟨d 84f⟩
PREVIOUS
```

Contains the data on the remainder of the line in reverse order.

83b ⟨*DX-SET* 83b⟩≡ (83a)
```
100 BAG DX-SET    : !DX-SET DX-SET !BAG ;
```

Fill '**DX-SET**' from the remainder of the line in reverse order.

83c ⟨*GET-DX-SET* 83c⟩≡ (83a)
```
: GET-DX-SET ( -- )   DEPTH >R
    BEGIN  BL WORD DUP C@  WHILE  FIND IF  EXECUTE  ELSE
           COUNT 0 0 2SWAP >NUMBER 2DROP DROP
        THEN  REPEAT  DROP
    DEPTH R> ?DO  DX-SET BAG+!  LOOP ;
```

Output '**DX-SET**' as bytes.

83d ⟨*C,-DX-SET* 83d⟩≡ (83a)
```
: C,-DX-SET ( -- )   BEGIN  DX-SET BAG@- AS-C,  DX-SET BAG? 0=  UNTIL ;
```

Add remainder of line to codespace, as bytes.

83e ⟨*db* 83e⟩≡ (83a)
```
: db ( -- )   !DX-SET  GET-DX-SET  C,-DX-SET ;
```

NOTE: The following assumes **(W,)** and **(L,)** are defined in the specific assembler. These must not be commaers, just lay down 16 or 32 bits entities in the right endian format.

Output '**DX-SET**' as words (16-bits).

83f ⟨*W,-DX-SET* 83f⟩≡ (83a)
```
: W,-DX-SET ( -- )   BEGIN  DX-SET BAG@- (W,)  DX-SET BAG? 0=  UNTIL ;
```

Add remainder of line to codespace, as words.

84a ⟨*dw* 84a⟩≡ (83a)
```
: dw ( -- )   !DX-SET  GET-DX-SET  W,-DX-SET ;
```

Output '**DX-SET**' as longs (32-bits)

84b ⟨*L,-DX-SET* 84b⟩≡ (83a)
```
: L,-DX-SET ( -- )   BEGIN  DX-SET BAG@- (L,)  DX-SET BAG? 0=  UNTIL ;
```

Add remainder of line to codespace, as longs (or, mostly, cells).

84c ⟨*dl* 84c⟩≡ (83a)
```
: dl ( -- )   !DX-SET  GET-DX-SET  L,-DX-SET ;
```

Lay down a STRING in assembler memory.

84d ⟨*(,)* 84d⟩≡ (83a)
```
: ($,) ( a n -- )   AS-HERE SWAP DUP AS-ALLOT MOVE ;
```

Output '**DX-SET**' as longs (32-bits)

84e ⟨*,-DX-SET* 84e⟩≡ (83a)
```
: $,-DX-SET ( -- )
    BEGIN  DX-SET BAG@- DUP 255 > IF
        DX-SET BAG@- ($,)  ELSE  AS-C,  THEN
    DX-SET BAG? 0=  UNTIL ;
```

Add remainder of line to codespace, as strings.

84f ⟨*d* 84f⟩≡ (83a)
```
: d$ ( -- )   !DX-SET  GET-DX-SET  $,-DX-SET ;
: "   [CHAR] " PARSE ;
```

3.10 Handle labels in disassembly

Handle labels as far as disassembly is concerned. There is a separate one for the assembler.

85 ⟨*labeldis.frt* 85⟩≡ (22)

```
( $Id: labeldis.frt,v 1.86 2010/06/03 23:27:28 albert Exp $ )
( Copyright{2004}: Albert van der Horst, HCC FIG Holland by GNU Public License)
( Uses Richard Stallmans convention. Uppercased word are parameters.    )

[DEFINED] ForCiForth [IF]
    REQUIRE ALIAS    REQUIRE @+    REQUIRE QSORT    REQUIRE EXCHANGE
    REQUIRE BIN-SEARCH    REQUIRE POSTFIX    REQUIRE H. \ In behalf of (DH.)
    REQUIRE 2>R REQUIRE BAG \ Simple bag facility
[THEN]

1000 CONSTANT MAX-LABEL

\ : \D ;

\ -------------------- INTRODUCTION --------------------------------

⟨generic-labels 88b⟩
⟨names-of-labels 93⟩
⟨comment-remainder 96b⟩
⟨multi-line-comment 98a⟩
⟨printing-strings 100a⟩
⟨start-of-line 102a⟩
⟨disassembly-ranges 103⟩
⟨disassemble 105d⟩
⟨unspecified 106d⟩
⟨bytes 107a⟩
⟨words 107g⟩
⟨longs 108f⟩
⟨strings 109c⟩
⟨IGNORE 86a⟩
⟨.HOW-FIT 86b⟩
⟨HOW-FIT 86c⟩
⟨HOW-FIT-END 86d⟩
⟨DISASSEMBLE-ALL 86e⟩

\ ------------------- Generic again -------------------

⟨(ADORN-ADDRESS) 87a⟩
⟨INIT-ALL 87b⟩
⟨SORT-ALL 87c⟩
⟨DECOMP-ONE 87d⟩
⟨DECOMP-ALL 87e⟩
```

⟨*MAKE-CUL* 87f⟩
⟨*SHOW-REGISTER* 87g⟩
⟨*DISASSEMBLE-TARGET* 87h⟩
⟨*DISASSEMBLE-TARGET* 87h⟩
⟨*DEFAULT-DISASSEMBLY* 88a⟩

```
ALSO ASSEMBLER
```

⟨*ranges* 110b⟩

```
DEFINITIONS
```

⟨*instructions* 111a⟩

```
PREVIOUS DEFINITIONS
```

Ignore addresses in this range. Define range to ignore.

86a ⟨*IGNORE* 86a⟩≡ (85)

```
' 2DROP DIS: IGNORE ( a1 a2 -- )
: -ignore ( a n -- )   2>R ['] IGNORE 2R> RANGE ;
' -ignore RANGE: -ignore: ( -name- )
: -ignore- ( -- )   NONAME$ -ignore ;
```

Print a remark about whether START and END fit.

86b ⟨*.HOW-FIT* 86b⟩≡ (85)

```
: .HOW-FIT ( a1 a2 -- )   2DUP = IF  2DROP  ELSE  > IF
            S" \ WARNING: This range overlaps with the previous one."  ELSE
            S" \ WARNING: There is hole between this range and the previous one"
    THEN  CR TYPE CR  THEN ;
```

Print a remark about whether start of the current range fits to the END of the previous range. Leave END of current range for the next check.

86c ⟨*HOW-FIT* 86c⟩≡ (85)

```
: HOW-FIT ( a -- a' )   RANGE-START .HOW-FIT  RANGE-END ;
```

Print a remark about whether the END of the previous range is really the end of the input file.

86d ⟨*HOW-FIT-END* 86d⟩≡ (85)

```
: HOW-FIT-END ( a -- )   TARGET-END .HOW-FIT ;
```

Disassemble all those ranges with their own disassemblers. No range will print their end labels, which is no problem if everything fits, except for the last range. Do that expressly.

86e ⟨*DISASSEMBLE-ALL* 86e⟩≡ (85)

```
: DISASSEMBLE-ALL ( -- )   TARGET-START  RANGE-LABELS DO-LAB
        I CELL+ @ RANGE-SECTION  HOW-FIT RANGE-DECODE
    LOOP-LAB  HOW-FIT-END   HOST-END CR-ADORNED ;
```

During assembly there is no decision needed whether to have a new line. Just do new line at ADDRESS, and get the eol-comment, if any. Revector '**ADORN-ADDRESS**' used in "**asgen.frt**".

87a ⟨*(ADORN-ADDRESS)* 87a⟩≡ (85)
```
: (ADORN-ADDRESS)   DUP CR-ADORNED   REMEMBER-COMMENT: ;
' (ADORN-ADDRESS) 'ADORN-ADDRESS !
```

Initialise all registered label classes.

87b ⟨*INIT-ALL* 87b⟩≡ (85)
```
: INIT-ALL   THE-REGISTER DO-BAG  I @ TO CURRENT-LABELSTRUCT  LABELS !BAG
    LOOP-BAG  INIT-COMMENT: ;
```

Sort all registered label classes.

87c ⟨*SORT-ALL* 87c⟩≡ (85)
```
: SORT-ALL   THE-REGISTER DO-BAG  I @ TO CURRENT-LABELSTRUCT  SORT-LABELS
    LOOP-BAG ;
```

Decompile all labels of current label class.

87d ⟨*DECOMP-ONE* 87d⟩≡ (85)
```
: DECOMP-ONE  LAB-UPB 1+ 1 ?DO  I DECOMP  LOOP ;
```

Generate the source of all label classs.

87e ⟨*DECOMP-ALL* 87e⟩≡ (85)
```
: DECOMP-ALL   THE-REGISTER DO-BAG  I @ TO CURRENT-LABELSTRUCT  DECOMP-ONE
    LOOP-BAG ;
```

Make a full blown cul file from the internal data.

87f ⟨*MAKE-CUL* 87f⟩≡ (85)
```
: MAKE-CUL  TARGET-START H. S"  -ORG-" TYPE CR DECOMP-ALL ;
```

Show what type of labels there are.

87g ⟨*SHOW-REGISTER* 87g⟩≡ (85)
```
: SHOW-REGISTER   THE-REGISTER DO-BAG  I @ %ID.  LOOP-BAG ;
```

Disassemble the current program as stored in the '**CODE-BUFFER**'. Using what is known about it.

87h ⟨*DISASSEMBLE-TARGET* 87h⟩≡ (85) 87i▷
```
: DISASSEMBLE-TARGET ( -- )   TARGET-START . S"  ORG" TYPE CR
    DISASSEMBLE-ALL ;
```

i386 dependant, should somehow be separated out.

87i ⟨*DISASSEMBLE-TARGET* 87h⟩+≡ (85) ◁87h
```
: DISASSEMBLE-TARGET ( -- )   S" [ASM HEX BITS-32" TYPE CR  DISASSEMBLE-TARGET CR ;
```

This is used to plug holes, where the user doesn't specify how to disassemble.

88a ⟨*DEFAULT-DISASSEMBLY* 88a⟩≡ (85)

```
VARIABLE DEFAULT-DISASSEMBLY
' -d$- DEFAULT-DISASSEMBLY !
: -ddef- DEFAULT-DISASSEMBLY @ EXECUTE ;
```

3.10.1 Generic definition of labels

Labels are bags of two cell structs, a target address and a pointer to the payload (mostly a string). They are sorted on target address for convenience.

88b ⟨*generic-labels* 88b⟩≡ (85)

⟨*THE-REGISTER* 89a⟩
⟨*REALLOC* 89b⟩
⟨*REALLOC-POINTER* 89c⟩
⟨*LABEL-FIELD* 89d⟩

⟨*LABELS[]* 90e⟩
⟨*REMOVE-LABEL* 90f⟩
⟨*DO-LAB* 90g⟩
⟨*.PAY.* 90h⟩
⟨*.PAY* 90i⟩
⟨*.PAY-DEA* 91a⟩
⟨*LABEL-NAME* 91b⟩
⟨*.LABELS* 91c⟩
⟨*LAB-BOUNDS* 91d⟩
⟨*LAB<* 91e⟩
⟨*LAB<-* 91f⟩
⟨*SORT-LABELS* 91g⟩
⟨*CONT* 91h⟩
⟨*L<* 91i⟩
⟨*WHERE-LABEL* 91j⟩

```
VARIABLE LABEL-CACHE    \ Index of next label.
```

⟨*FIND-LABEL* 91k⟩
⟨*>LABEL* 92a⟩

```
VARIABLE MAX-DEV-P   -8 MAX-DEV-P !    \ Max deviation acceptable with previous
VARIABLE MAX-DEV-N    8 MAX-DEV-N !    \ Max deviation acceptable with next
```

⟨*(LABEL)* 92b⟩
⟨*IMPROVE-LABEL* 92c⟩
⟨ *LABEL* 92d⟩
⟨*ROLL-LABEL* 92e⟩
⟨*NEXT-LABEL* 92f⟩

A bag with the dea's of all labelclasses.

89a ⟨*THE-REGISTER* 89a⟩≡ (88b)
```
100 BAG THE-REGISTER
```

Realloc ADDRESS to new LENGTH, return new ADDRESS. Ignoring old length, we may copy garbage, too bad.

89b ⟨*REALLOC* 89b⟩≡ (88b)
```
: REALLOC ( u -- a )   HERE >R  DUP ALLOT  R@ SWAP MOVE   R> ;
```

Realloc POINTER to old buffer to new LENGTH. Afterwards pointer points to new buffer.

89c ⟨*REALLOC-POINTER* 89c⟩≡ (88b)
```
: REALLOC-POINTER ( a n -- )   >R  DUP @ R> REALLOC  SWAP ! ;
```

Define a class for label-like things of length N. A label-like thing is two cells: address and a payload.

89d ⟨*LABEL-FIELD* 89d⟩≡ (88b)
```
0 VALUE CURRENT-LABELSTRUCT \ current label pointer

: LABEL-FIELD ( u size -- u' )   CREATE  OVER , +
    DOES> ( -- a )   @ CURRENT-LABELSTRUCT + ;

0
    2 CELLS LABEL-FIELD LABEL-RESERVED
    1 CELLS LABEL-FIELD 'CURRENT-LABEL
    1 CELLS LABEL-FIELD 'DECOMP          \ (Re)generate source for INDEX.
    1 CELLS LABEL-FIELD '.PAY            \ Print payload
    1 CELLS LABEL-FIELD 'MAX-LAB
    0 LABEL-FIELD LABELS           \ Return ADDRESS
CONSTANT |LABELSTRUCT|

: CURRENT-LABEL ( -- a )   'CURRENT-LABEL @ ;
: DECOMP ( -- )   'DECOMP @ EXECUTE ;
: .PAY ( -- )   '.PAY @ EXECUTE ;

⟨DOUBLE-SIZE 90a⟩
⟨MAX-LAB 90b⟩
⟨LAB-UPB 90c⟩
⟨RELOCATABLE 90d⟩

: LABELSTRUCT ( n print gen -- )   SAVE-INPUT  CREATE  HERE DUP >R
    DUP THE-REGISTER BAG+!  |LABELSTRUCT| DUP ALLOT  ERASE
    RESTORE-INPUT THROW  BL WORD $@ $, R@ >NFA !
    CURRENT-LABELSTRUCT R@ >LFA !  R> TO CURRENT-LABELSTRUCT
    DUP 'CURRENT-LABEL !  'DECOMP !  '.PAY !  DUP 'MAX-LAB ! 2* BUILD-BAG
    DOES>  TO CURRENT-LABELSTRUCT ;
```

Remember that from now on two times as much labels are allowed.

90a ⟨*DOUBLE-SIZE* 90a⟩≡ (89d)
```
: DOUBLE-SIZE ( -- )   'MAX-LAB DUP @ 2* SWAP ! ;
```

Return a VARIABLE containing the max labels allowed.

90b ⟨*MAX-LAB* 90b⟩≡ (89d)
```
: MAX-LAB ( -- n )   'MAX-LAB @ ;
```

Return largest INDEX of labels present.

90c ⟨*LAB-UPB* 90c⟩≡ (89d)
```
: LAB-UPB ( -- n )   LABELS |BAG| 2/ ;
```

Add to '**LABELS**'. Reallocate if the class is full. 6 cells : does> pointer, 4 fields and upperbound of bag. Make label relocatable and back. Don't use in between!

90d ⟨*RELOCATABLE* 90d⟩≡ (89d)
```
: >RELOCATABLE ( -- )   LABELS DUP @  OVER - SWAP ! ;
: RELOCATABLE> ( -- )   LABELS DUP +! ;
: ?REALLOC? ( -- )
    MAX-LAB LAB-UPB = IF  DOUBLE-SIZE
        >RELOCATABLE  CURRENT-LABEL MAX-LAB 2* 6 + CELLS REALLOC-POINTER
        CURRENT-LABEL EXECUTE   RELOCATABLE>  THEN ;

: LAB+! ( a -- )   LABELS BAG+!  ?REALLOC? ;
```

For I return the ith LABEL . 1 returns the first label. All indices are compatible with this.

90e ⟨*LABELS[]* 90e⟩≡ (88b)
```
: LABELS[] ( n -- a )   1- 2* CELLS LABELS CELL+ + ;
```

Remove label I.

90f ⟨*REMOVE-LABEL* 90f⟩≡ (88b)
```
: REMOVE-LABEL ( -- )   LABELS[] LABELS  2DUP BAG-REMOVE BAG-REMOVE ;
```

Loop through all '**LABELS**', similar to '**DO-BAG .. DO-LOOP**' but with a stride of 2 cells and the bag built-in.

90g ⟨*DO-LAB* 90g⟩≡ (88b)
```
: DO-LAB   POSTPONE LABELS POSTPONE DO-BAG ; IMMEDIATE
: LOOP-LAB   2 CELLS POSTPONE LITERAL POSTPONE +LOOP ; IMMEDIATE
```

A simple printout of the payload.

90h ⟨*.PAY.* 90h⟩≡ (88b)
```
: .PAY. ( a -- )   CELL+ ? ;
```

Print the payload of the label at ADDRESS , provided it is a string.

90i ⟨*.PAY* 90i⟩≡ (88b)
```
: .PAY$ ( a -- )   CELL+ @ $@ TYPE  3 SPACES ;
```

Print the name of the label at ADDRESS , provided it is a dea. This applies to plain labels that are in fact constants.

91a ⟨*.PAY-DEA* 91a⟩≡ (88b)
```
: .PAY-DEA ( a -- )   CELL+ @ %ID. ;
```

For label INDEX return the label NAME, provided it is a dea.

91b ⟨*LABEL-NAME* 91b⟩≡ (88b)
```
: LABEL-NAME ( n -- a n )   LABELS[] CELL+ @ >BODY >NFA @ $@ ;
```

Print the addresses and payloads of the labels.

91c ⟨*.LABELS* 91c⟩≡ (88b)
```
: .LABELS ( -- )   DO-LAB  I @ .  I .PAY  CR  LOOP-LAB ;
```

Return LOWER and UPPER indices of the labels, inclusive. The lower index is 1 and the upper index is corresponding.

91d ⟨*LAB-BOUNDS* 91d⟩≡ (88b)
```
: LAB-BOUNDS ( -- l u )   1 LAB-UPB ;
```

In behalf of qsort. For INDEX1 and INDEX2: "the value of the first label is less than that of the second"

91e ⟨*LAB<* 91e⟩≡ (88b)
```
: LAB< ( i1 i2 -- f )   LABELS[] @  SWAP LABELS[] @  SWAP < ;
```

In behalf of qsort. Exchange the labels with INDEX1 and INDEX2 .

91f ⟨*LAB<-* 91f⟩≡ (88b)
```
: LAB<-> ( i1 i2 -- )   LABELS[] SWAP LABELS[]  2 CELLS EXCHANGE ;
```

Sort the labels of '**LABELS**' in ascending order.

91g ⟨*SORT-LABELS* 91g⟩≡ (88b)
```
: SORT-LABELS ( -- )   LAB-BOUNDS  ['] LAB<  ['] LAB<->  QSORT ;
```

In behalf of bin-search. Comparant.

91h ⟨*CONT* 91h⟩≡ (88b)
```
VARIABLE CONT
```

In behalf of bin-search. For INDEX1 : "the value of the label is less than '**CONT**'"

91i ⟨*L<* 91i⟩≡ (88b)
```
: L< ( n -- f )   LABELS[] @  CONT @ < ;
```

Find where ADDRESS belongs in a sorted array. Return the INDEX. If address is already present, its index is returned. This may be outside, if it is larger than any.

91j ⟨*WHERE-LABEL* 91j⟩≡ (88b)
```
: WHERE-LABEL ( a -- )   CONT !  LAB-BOUNDS 1+  ['] L<  BIN-SEARCH ;
```

Find the first label that is equal to (or greater than) VALUE. Return INDEX or zero if not found. Put it in the label-cache too. Note '**BIN-SEARCH**' returns the non-inclusive upper bound if not found.

91k ⟨*FIND-LABEL* 91k⟩≡ (88b)
```
: FIND-LABEL ( n -- )   WHERE-LABEL  DUP LAB-UPB 1+ <> AND  DUP LABEL-CACHE ! ;
```

Find ADDRESS in the label table. Return table ADDRESS of an exact matching label or zero if not found.

92a ⟨*>LABEL* 92a⟩≡ (88b)
```
: >LABEL ( a -- )   DUP >R  FIND-LABEL DUP IF  LABELS[] DUP @  R@ <> IF
    DROP 0  THEN THEN
    R> DROP ;
```

Find ADDRESS in the label table. Return INDEX of an approximately matching label or zero if not found.

92b ⟨*(LABEL)* 92b⟩≡ (88b)
```
: (~LABEL)   DUP MAX-DEV-P @ + FIND-LABEL
    DUP 0= IF 2DROP 0 ELSE
        OVER MAX-DEV-N @ + OVER LABELS[] @  < IF 2DROP 0 ELSE
    SWAP DROP THEN THEN ;
```

For ADDRESS and INDEX return ADDRESS and INDEX where the deviation is minimal.

92c ⟨*IMPROVE-LABEL* 92c⟩≡ (88b)
```
: IMPROVE-LABEL
    BEGIN  DUP LAB-UPB <> IF  2DUP 1+ LABELS[] @ < 0=  ELSE  0  THEN
    WHILE  1+  REPEAT ;
```

Find target ADDRESS in the label table. Return table ADDRESS of an approximately matching label, or zero if not found, plus the OFFSET.

92d ⟨ *LABEL* 92d⟩≡ (88b)
```
: ~LABEL   DUP (~LABEL) DUP 0= IF  2DROP 0 0
    ELSE  IMPROVE-LABEL  LABELS[] SWAP OVER @ -  THEN ;
```

Roll the last label to place INDEX. A label occupies two consecutive places!

92e ⟨*ROLL-LABEL* 92e⟩≡ (88b)
```
: ROLL-LABEL   DUP   LABELS[]  DUP LABELS BAG-HOLE   LABELS BAG-HOLE
    LAB-BOUNDS SWAP DROP   LAB<->   -2 CELLS LABELS  +! ;
```

FIXME: The following is dead code. (As is **LABEL-CACHE**). Return the next label or 0.

92f ⟨*NEXT-LABEL* 92f⟩≡ (88b)
```
: NEXT-LABEL   LABEL-CACHE @   DUP IF
        1+ DUP LAB-BOUNDS + = IF DROP 0 THEN
    DUP LABEL-CACHE ! THEN ;
```

3.10.2 Names of labels

93 ⟨*names-of-labels* 93⟩≡ (85)

⟨*.EQU* 94a⟩
⟨*EQU-LABELS* 94b⟩
⟨*LABELED* 94c⟩
⟨*LABEL* 94d⟩
⟨*=EQU-LABEL* 94e⟩
⟨*.EQU-ALL* 94f⟩
⟨*ADORN-WITH-LABEL* 94g⟩

```
HEX FFFF0000 CONSTANT LARGE-NUMBER-MASK
```

⟨*.0?* 94h⟩
⟨*SMART.* 95a⟩
⟨*. LABEL* 95b⟩

```
DECIMAL
VARIABLE SMALL-LABEL-LIMIT   100 SMALL-LABEL-LIMIT !
```

⟨*.LABEL/.* 95c⟩

```
\D 0 SMALL-LABEL-LIMIT !
\D 12 LABEL AAP
\D 5 LABEL NOOT
\D 2 LABEL MIES
\D 123 LABEL POPI

\D .LABELS CR
\D SORT-LABELS
\D .LABELS CR

\D 200 FIND-LABEL . CR
\D 12 FIND-LABEL  LABELS[] .PAY CR
\D 12 1- FIND-LABEL  LABELS[] .PAY CR
\D 12 >LABEL .PAY CR
\D 12 1- >LABEL H. CR
\D 12 ADORN-WITH-LABEL  4 <?> CR  \ Should give zero, not found!
\D 12 0 HOST>TARGET - ADORN-WITH-LABEL  CR
```

⟨*@LABEL* 95d⟩
⟨*LABEL=* 95e⟩
⟨*REMOVE-TRIVIAL* 95f⟩
⟨*CLEAN-LABELS* 96a⟩

Decompile label INDEX.

94a ⟨*.EQU* 94a⟩≡ (93)
```
: .EQU    LABELS[] DUP @ H. S"  EQU " TYPE  CELL+ @ %ID. CR ;
```

Contains equ labels, i.e. classes as associate with '**LABEL**'.

94b ⟨*EQU-LABELS* 94b⟩≡ (93)
```
MAX-LABEL ' .PAY-DEA ' .EQU LABELSTRUCT EQU-LABELS  LABELS !BAG
```

Generate a equ label at (target) ADDRESS with NAME, this can be any symbolic constant in fact. The payload is the dea of a constant leaving that address.

94c ⟨*LABELED* 94c⟩≡ (93)
```
: LABELED ( x a n -- )   S" CREATE " PAD $!  PAD COUNT + OVER
    2SWAP PAD $+!  PAD $@ EVALUATE  HERE >R  'LABELS @ , ( >LFA )
    R@ 'LABELS !  HERE 2 CELLS + , ( >NFA )  ROT DUP , ( >DATA )
    -ROT $, DROP  EQU-LABELS  LAB+!  R> LAB+!
    DOES> ( -- x )   >DATA @ ;
```

Generate a equ label at (target) ADDRESS with "NAME". Like '**LABEL**'.

94d ⟨*LABEL* 94d⟩≡ (93)
```
: LABEL   BL WORD COUNT LABELED ;
: EQU LABEL ;
```

For host ADDRESS return an associated equ LABEL or 0. CAVEAT: if there are more than one label for the same addres, just the first one is returned.

94e ⟨*=EQU-LABEL* 94e⟩≡ (93)
```
: =EQU-LABEL   HOST>TARGET  EQU-LABELS >LABEL ;
```

For host ADDRESS print all labels at that addres, return the NUMBER of labels printed.

94f ⟨*.EQU-ALL* 94f⟩≡ (93)
```
: .EQU-ALL   HOST>TARGET  EQU-LABELS   0 ( no labels printed) SWAP
    LAB-UPB 1+ OVER WHERE-LABEL ?DO
        DUP I LABELS[] @ <> IF LEAVE THEN
        SWAP 1+ SWAP
        [CHAR] : EMIT I LABELS[] .PAY
    LOOP DROP ;
```

Adorn the ADDRESS we are currently disassembling with a named label if any.

94g ⟨*ADORN-WITH-LABEL* 94g⟩≡ (93)
```
: ADORN-WITH-LABEL   .EQU-ALL    0= IF 12 SPACES THEN ;
```

Prevent leading hex letter for NUMBER by printing a zero.

94h ⟨*.0?* 94h⟩≡ (93)
```
: .0?   DUP 0A0 100 WITHIN SWAP 0A 10 WITHIN OR IF [CHAR] 0 EMIT THEN ;
```

Print a NUMBER in hex in a smart way.

95a ⟨*SMART.* 95a⟩≡ (93)
```
: SMART.   DUP ABS 100 < IF DUP .0? . ELSE
    LARGE-NUMBER-MASK OVER AND IF H. ELSE 0 4 (DH.) TYPE THEN SPACE THEN ;
```

For label INDEX and OFFSET print the label with offset.

95b ⟨*. LABEL* 95b⟩≡ (93)
```
: .~LABEL   SWAP .PAY   ?DUP IF
    DUP 0< IF NEGATE . S" - " TYPE ELSE . S" + " TYPE THEN THEN ;
```

Print X as a symbolic label if possible, else as a number.

95c ⟨*.LABEL/.* 95c⟩≡ (93)
```
: .LABEL/.   EQU-LABELS
    DUP ABS SMALL-LABEL-LIMIT @ < IF SMART.
    ELSE DUP ~LABEL OVER IF .~LABEL DROP
    ELSE 2DROP SMART.
    THEN THEN ;
```

For label INDEX , return its XT.

95d ⟨*@LABEL* 95d⟩≡ (93)
```
: @LABEL    LABELS[] CELL+ @ ;

\D 5 DUP INVENT-NAME LABELED
\D SORT-LABELS .LABELS 5 <?> CR
\D .LABELS 6 <?> CR
\D S" EXPECT 5: " TYPE 5 FIND-LABEL @LABEL >DATA @ . 7 <?> CR
```

For labels INDEX1 and INDEX2 return "they are equal".

95e ⟨*LABEL=* 95e⟩≡ (93)
```
: LABEL=   @LABEL >DATA @   SWAP @LABEL >DATA @   = ;

\D S" EXPECT -1:" TYPE 5 FIND-LABEL DUP 1+ LABEL= . CR
```

Get rid of a label with INDEX if trivial . Return next INDEX to try.

95f ⟨*REMOVE-TRIVIAL* 95f⟩≡ (93)
```
: REMOVE-TRIVIAL   DUP @LABEL DUP >DATA @ SWAP >NFA @ $@ INVENTED-NAME? IF
        DUP REMOVE-LABEL ELSE 1+ THEN ;

\D 5 FIND-LABEL  DUP REMOVE-TRIVIAL .  REMOVE-TRIVIAL .
\D .LABELS 8 <?> CR
```

Get rid of superfluous equ labels

96a ⟨*CLEAN-LABELS* 96a⟩≡ (93)

```
: CLEAN-LABELS   EQU-LABELS
    2 BEGIN DUP LAB-UPB < WHILE DUP DUP 1- LABEL= >R DUP DUP 1+ LABEL= R> OR IF
            REMOVE-TRIVIAL ELSE 1+ THEN REPEAT DROP ;

\D 5 DUP INVENT-NAME LABELED  SORT-LABELS
\D .LABELS 9 <?> CR
\D CLEAN-LABELS
\D .LABELS 10 <?> CR
```

3.10.3 Comment remainder of line

Comment till remainder of line.

96b ⟨*comment-remainder* 96b⟩≡ (85)

⟨*.COMMENT:* 96c⟩
⟨*COMMENT:-LABELS* 96d⟩
⟨*COMMENT:* 97a⟩
⟨*COMMENT:-TO-BE* 97b⟩
⟨*INIT-COMMENT:* 97c⟩
⟨*PRINT-OLD-COMMENT:* 97d⟩
⟨*REMEMBER-COMMENT:* 97e⟩

```
\D .LABELS CR
\D SORT-LABELS
\D .LABELS CR

\D 200 FIND-LABEL . CR
\D 12 FIND-LABEL  LABELS[] .PAY CR
\D 12 1- FIND-LABEL  LABELS[] .PAY CR
\D 12 >LABEL .PAY CR
\D 12 1- >LABEL H. CR
```

Decompile comment: label INDEX.

96c ⟨*.COMMENT:* 96c⟩≡ (96b)

```
: .COMMENT:   LABELS[] DUP @ H. S"  COMMENT: " TYPE  CELL+ @ $@ TYPE CR ;
```

Contains comment labels, i.e. classes as associate with '**COMMENT:**'

96d ⟨*COMMENT:-LABELS* 96d⟩≡ (96b)

```
MAX-LABEL ' .PAY$ ' .COMMENT: LABELSTRUCT COMMENT:-LABELS  LABELS !BAG
```

Generate a comment label at ADDRESS. A pointer to the remainder of the line is the payload.

97a ⟨*COMMENT:* 97a⟩≡ (96b)

```
: COMMENT:   COMMENT:-LABELS   LAB+!  [CTRL] J PARSE $, LAB+! ;

\D 12 COMMENT: AAP
\D 115 COMMENT: NOOTJE
\D 2 COMMENT: MIES
\D 123 COMMENT: POPI
```

Remember the comment at the end of this instruction. Zero means no comment.

97b ⟨*COMMENT:-TO-BE* 97b⟩≡ (96b)

```
VARIABLE COMMENT:-TO-BE
```

Initialise to no comment.

97c ⟨*INIT-COMMENT:* 97c⟩≡ (96b)

```
: INIT-COMMENT:   0 COMMENT:-TO-BE ! ;
  INIT-COMMENT:
```

Print comment at the end of previous instruction.

97d ⟨*PRINT-OLD-COMMENT:* 97d⟩≡ (96b)

```
: PRINT-OLD-COMMENT:   COMMENT:-TO-BE @ DUP IF
    S" \ " TYPE   $@ TYPE _ THEN DROP
    INIT-COMMENT: ;
```

Remember what comment to put after the disassembly of ADDRESS.

97e ⟨*REMEMBER-COMMENT:* 97e⟩≡ (96b)

```
: REMEMBER-COMMENT:   COMMENT:-LABELS   HOST>TARGET >LABEL
    DUP IF CELL+ @ COMMENT:-TO-BE ! _ THEN DROP  ;

\D 12 REMEMBER-COMMENT: PRINT-OLD-COMMENT: CR  \ Should give nothing, not found!
\D 12 0 HOST>TARGET - REMEMBER-COMMENT: PRINT-OLD-COMMENT: CR
```

3.10.4 Multi-line comments

Multiple line comment/command in front.

98a ⟨*multi-line-comment* 98a⟩≡ (85)
⟨*.MDIRECTIVE* 98b⟩
⟨*MCOMMENT-LABELS* 98c⟩
⟨*NEW-DIRECTIVE* 98d⟩
⟨*OLD-DIRECTIVE* 98e⟩
⟨*DIRECTIVE* 99a⟩
⟨*COMMENT* 99b⟩
⟨*PRINT-DIRECTIVE* 99c⟩

```
\D .LABELS CR
\D SORT-LABELS
\D .LABELS CR

\D 200 FIND-LABEL . CR
\D 12 FIND-LABEL  LABELS[] .PAY CR
\D 12 1- FIND-LABEL  LABELS[] .PAY CR
\D 12 >LABEL .PAY CR
\D 12 1- >LABEL H. CR
```

Decompile mcomment label INDEX. Duplicate the line heading on each line of output.

98b ⟨*.MDIRECTIVE* 98b⟩≡ (98a)

```
: .MDIRECTIVE   LABELS[] DUP @ DUP >R H. S"  :COMMENT " TYPE
    CELL+ @ $@  BEGIN  2DUP 10 SCAN ?DUP
    WHILE  1 /STRING 2SWAP 2OVER NIP - 1- TYPE CR
        R@ H. S"  :COMMENT " TYPE
    REPEAT  R> 2DROP TYPE CR ;
```

Contains multiple line comment labels, i.e. classes associate with '**COMMENT**'

98c ⟨*MCOMMENT-LABELS* 98c⟩≡ (98a)

```
MAX-LABEL ' .PAY$ ' .MDIRECTIVE LABELSTRUCT MCOMMENT-LABELS  LABELS !BAG
```

New directive STRING at ADDRESS. (See '**DIRECTIVE**'). Primitive, doesn't keep it sorted.

98d ⟨*NEW-DIRECTIVE* 98d⟩≡ (98a)

```
: NEW-DIRECTIVE ( a n x -- )   LAB+! $, LAB+! ;
```

Directive STRING to old ADDRESS. Append. (See '**DIRECTIVE**').

98e ⟨*OLD-DIRECTIVE* 98e⟩≡ (98a)

```
: OLD-DIRECTIVE ( a n -- )   >LABEL CELL+ DUP >R
    @ $@ PAD $!  [CTRL] J PAD $C+  PAD $+!  PAD $@ $, R> ! ;
```

Make STRING the command in front of label at ADDRESS. A pointer to this string the payload. If there is already a directive there, this one is appended. Keep things sorted.

99a ⟨*DIRECTIVE* 99a⟩≡ (98a)

```
: DIRECTIVE ( a n x -- )   MCOMMENT-LABELS >R
    R@ >LABEL IF   R@ OLD-DIRECTIVE   ELSE
    R@ NEW-DIRECTIVE   R@ WHERE-LABEL ROLL-LABEL THEN R> DROP ;
```

Make STRING the comment in front of label at ADDRESS. A pointer to this string the payload.

99b ⟨*COMMENT* 99b⟩≡ (98a)

```
: COMMENT ( a n x -- )   >R  S" \ " $,  DUP >R  OVER ALLOT  $+!
    R> $@  R> DIRECTIVE ;
: :COMMENT ( x -line- )   [CTRL] J PARSE $, $@ ROT COMMENT ;

\D S" AAP" 12 COMMENT
\D S" NOOT" 5 COMMENT
\D S" MIES" 2 COMMENT
\D S" POPI
\ \D JOPI"
\D 123 COMMENT
```

Print comment for instruction at ADDRESS , if any.

99c ⟨*PRINT-DIRECTIVE* 99c⟩≡ (98a)

```
: PRINT-DIRECTIVE MCOMMENT-LABELS  HOST>TARGET  >LABEL DUP IF
    CR   .PAY _ THEN DROP ;

\D 12 PRINT-DIRECTIVE CR  \ Should give nothing, not found!
\D 12 0 HOST>TARGET - PRINT-DIRECTIVE CR
```

3.10.5 Printing strings

The special printing of strings.

100a ⟨*printing-strings* 100a⟩≡ (85)

```
[DEFINED] ForCiForth [IF]
    REQUIRE NEW-IF
[THEN]

⟨TABLE2 100b⟩
⟨IS-CTRL 100c⟩
⟨IS-PRINT 101a⟩
⟨ACCU 101b⟩
⟨.ACCU 101c⟩
⟨.C 101d⟩

\ FIXME: to be renamd in WHERE-FLUSH
VARIABLE NEXT-CUT       \ Host address where to separate db etc. in chunks.
VARIABLE CUT-SIZE    16 CUT-SIZE !   \ Chunks for data-disassembly.

⟨ACCU-C+ 101e⟩
```

Contains a printing indicator:

	description	syntax	example
0	as hex	8A	
1	control character	^J	
2	a blank	BL	"xxx xxx"
3	normal printable	&Z	"xxxZxxx"

100b ⟨*TABLE2* 100b⟩≡ (100a)

```
CREATE TABLE2 256 ALLOT      TABLE2 256 ERASE
: /TABLE2 [CHAR] ~ 1 + BL 1 + DO 3 TABLE2 I + C! LOOP ;
/TABLE2
2 BL TABLE2 + C!
1 CTRL I TABLE2 + C!
1 CTRL J TABLE2 + C!
1 CTRL M TABLE2 + C!
1 CTRL L TABLE2 + C!
```

For CHAR: "it is control"

100c ⟨*IS-CTRL* 100c⟩≡ (100a)

```
: IS-CTRL   TABLE2 + C@ 1 = ;

\D S" EXPECT 0 -1 :" TYPE CHAR A IS-CTRL .   CTRL J IS-CTRL . CR 11 <?>
```

For CHAR: "it is printable".

101a ⟨*IS-PRINT* 101a⟩≡ (100a)
```
: IS-PRINT   TABLE2 + C@ 1 > ;

\D S" EXPECT 0 -1 -1 :" TYPE CTRL A IS-PRINT .
\D CHAR A IS-PRINT . BL IS-PRINT . CR 12 <?>
```

Accumulates characters that may form a string.

101b ⟨*ACCU* 101b⟩≡ (100a)
```
CREATE ACCU 100 ALLOT           ACCU 100 ERASE

\ \D S" Expect " TYPE  """ AA""""AA """ TYPE CHAR : EMIT " AA""AA " S" $" TYPE CR 13 <?>
```

Print the accumulated chars, if any.

101c ⟨*.ACCU* 101c⟩≡ (100a)
```
: .ACCU   ACCU $@
    OVER C@ BL = OVER 1 = AND IF  2DROP  S"  BL" TYPE  ELSE
        DUP 1 > IF  SPACE [CHAR] " EMIT SPACE TYPE [CHAR] " EMIT  ELSE
            IF  SPACE [CHAR] " EMIT SPACE C@ EMIT [CHAR] " EMIT  ELSE  DROP
    THEN THEN THEN  0 0 ACCU $! ;

\D S" EXPECT " TYPE  S" XY :" TYPE  S" XY" ACCU $! .ACCU CR 14 <?>
\D S" EXPECT BL :" TYPE   S"  " ACCU $!   .ACCU CR 15 <?>
\D S" EXPECT CHAR Y :" TYPE   S" Y" ACCU $!   .ACCU CR 16 <?>
```

Display a BYTE in clean hex. Display the non-printable character.

101d ⟨*.C* 101d⟩≡ (100a)
```
: .B-CLEAN   DUP .0? 0 <# BL HOLD #S #> TYPE ;
: .C   .ACCU SPACE DUP IS-CTRL IF  S" CTRL " TYPE  [CHAR] @ + EMIT
    ELSE  .B-CLEAN  THEN  ;

\D S" EXPECT CTRL J :" TYPE  CTRL J .C CR 17 <?>
\D S" EXPECT 0: " TYPE  0 .C CR 18 <?>
\D S" EXPECT 9A: " TYPE  HEX 9A .C CR 19 <?> DECIMAL
\D S" EXPECT 0FA: " TYPE  HEX FA .C CR 20 <?> DECIMAL
```

For ADDR of a (printable) char, add it to the accumulated range. Force an immediate flush, if the range is full. Otherwise postpone the flush at least one char. If the character following is a string ender, this is a desirable place to break. (String enders like ^J and 0 are not printable.)

101e ⟨*ACCU-C+* 101e⟩≡ (100a)
```
: ACCU-$C+   DUP C@ ACCU $C+   ACCU @ 64 = IF 1+ ELSE 2 + THEN NEXT-CUT ! ;
```

3.10.6 Start of line

Things to print at the start of a line.

102a ⟨*start-of-line* 102a⟩≡ (85)
```
⟨.TARGET-ADDRESS 102b⟩
⟨CR-ADORNED 102c⟩
⟨NEXT-CUT? 102d⟩
⟨CR+GENERIC 102e⟩
⟨CR+dx 102f⟩
```

Print the ADDRES as target address in hex.

102b ⟨.*TARGET-ADDRESS* 102b⟩≡ (102a)
```
: .TARGET-ADDRESS S" ( " TYPE DUP HOST>TARGET H. S"  )   " TYPE ;
```

Start a new line, with printing the decompiled ADDRESS as seen.

102c ⟨*CR-ADORNED* 102c⟩≡ (102a)
```
: CR-ADORNED
    PRINT-OLD-COMMENT:
    DUP PRINT-DIRECTIVE
    CR 'ADORN-ADDRESS 2@ - IF  .TARGET-ADDRESS  THEN
    ADORN-WITH-LABEL ;
```

For ADDRESS: "it is at next cut." If so, advance.

102d ⟨*NEXT-CUT?* 102d⟩≡ (102a)
```
: NEXT-CUT?   NEXT-CUT @ =  DUP IF CUT-SIZE @ NEXT-CUT +! THEN ;
```

For ADDRESS and assembler directive STRING (such "**db**"), interrupt the laying down of memory classes by a new line and possibly a label, when appropriate.

102e ⟨*CR+GENERIC* 102e⟩≡ (102a)
```
: CR+GENERIC   2>R DUP =EQU-LABEL >R DUP NEXT-CUT?   R> OR IF
    DUP CR-ADORNED  2R@ TYPE THEN REMEMBER-COMMENT: 2R> 2DROP ;

: CR+$         2>R DUP =EQU-LABEL >R DUP NEXT-CUT?   R> OR IF .ACCU
    DUP CR-ADORNED  2R@ TYPE THEN REMEMBER-COMMENT: 2R> 2DROP ;
```

For ADDRESS : interupt byte display.

102f ⟨*CR+dx* 102f⟩≡ (102a)
```
: CR+dn   S"   " CR+GENERIC ;
: CR+db   S"   db " CR+GENERIC ;
: CR+dw   S"   dw " CR+GENERIC ;
: CR+dl   S"   dl " CR+GENERIC ;
: CR+d$   S"   d$ " CR+$ ;
```

3.10.7 Disassembly ranges

A 'range' is defined here as a range of addresses that is kept together, even during relocation and such, and contains data for the same type. A 'section' is defined here as a range of addresses that is kept together, even during relocation and such, where data need not be of the same type. 'Disassembly' is to be understood as interpreting the content of a range, not necessarily as executable code. Range ADDRESS1 .. ADDRESS2 always refers to a target range, where address2 is exclusive.

103 ⟨*disassembly-ranges* 103⟩≡ (85)

```
0 VALUE CURRENT-RANGE \ current range pointer
```

⟨*RANGE-FIELD* 104a⟩

```
0
    2 CELLS RANGE-FIELD RANGE-RESERVED
    1 CELLS RANGE-FIELD 'RANGE-START      \ Start of range
    1 CELLS RANGE-FIELD 'RANGE-END        \ End of range
    1 CELLS RANGE-FIELD 'RANGE-STRIDE     \ For the moment = 1 FIXME!
    1 CELLS RANGE-FIELD 'RANGE-XT
    1 CELLS RANGE-FIELD 'RANGE-SECTION
CONSTANT |RANGE|

: RANGE-START ( -- a )   'RANGE-START @ ;
: RANGE-END! ( a -- )   'RANGE-END ! ;
: RANGE-END ( -- a )   'RANGE-END @ ;
: RANGE-STRIDE ( -- n )   'RANGE-STRIDE @ ;
: RANGE-XT ( -- xt )   'RANGE-XT @ ;
: RANGE-DECODE ( -- )   'RANGE-XT @ >R RANGE-START RANGE-END R> EXECUTE ;
: RANGE-SECTION ( a -- )   TO CURRENT-RANGE
    'RANGE-SECTION @ TO CURRENT-SECTION ;
```

⟨*.PAY-RANGE* 104b⟩

```
20 BAG RANGE-TYPES  \ Contains dea of dumper, creator, alternating.
```

⟨*ARE-COUPLED* 104c⟩
⟨*CREATOR-XT* 104d⟩
⟨*MAKE-CURRENT* 104e⟩
⟨*DECOMP-RANGE* 104f⟩
⟨*RANGE-LABELS* 104g⟩
⟨*RANGE* 105a⟩

```
0 VALUE DISASSEMBLERS
0 VALUE RANGE-RANGES

0
    2 CELLS DEA-FIELD RANGE-RESERVED
    1 CELLS DEA-FIELD >DIS:
CONSTANT |DISASSEMBLER|
```

⟨*DIS:* 105b⟩
⟨*RANGE:* 105c⟩

Define a range.

104a ⟨*RANGE-FIELD* 104a⟩≡ (103)
```
: RANGE-FIELD ( u size -- u' )   CREATE  OVER , +
    DOES> ( -- a )   @ CURRENT-RANGE + ;
```

Print the range LAB as a matter of testing.

104b ⟨*.PAY-RANGE* 104b⟩≡ (103)
```
: .PAY-RANGE   CELL+ @ DUP RANGE-SECTION
    RANGE-START H.  SPACE RANGE-END H.  S"  BY " TYPE
    RANGE-XT >BODY %ID.  %ID. ;
```

DEA of dump belongs to DEA of creator. Add to '**RANGE-TYPES**'.

104c ⟨*ARE-COUPLED* 104c⟩≡ (103)
```
: ARE-COUPLED   >BODY SWAP >BODY  RANGE-TYPES BAG+! RANGE-TYPES BAG+! ;
```

For current range, return the XT of a proper defining word.

104d ⟨*CREATOR-XT* 104d⟩≡ (103)
```
: CREATOR-XT   RANGE-XT >BODY RANGE-TYPES BAG-WHERE CELL+ @ ;
```

Make range I current, provided the payload is a dea.

104e ⟨*MAKE-CURRENT* 104e⟩≡ (103)
```
: MAKE-CURRENT ( n -- )   LABELS[] CELL+ @ RANGE-SECTION ;
```

Display range INDEX in a reconsumable form. Shortens the **NONAME** ranges with a "**-**" replacing the "**:**" at the end of the type name.

104f ⟨*DECOMP-RANGE* 104f⟩≡ (103)
```
: DECOMP-RANGE   MAKE-CURRENT RANGE-START H. SPACE RANGE-END H. SPACE
    CURRENT-RANGE >NFA @ $@  CREATOR-XT >NFA @ $@
    2OVER S" NONAME" COMPARE 0= IF
        1- TYPE ." - "  2DROP
    ELSE  TYPE SPACE  TYPE
    THEN  CR ;
```

Contains range specification, limits plus type.

104g ⟨*RANGE-LABELS* 104g⟩≡ (103)
```
MAX-LABEL ' .PAY-RANGE ' DECOMP-RANGE LABELSTRUCT RANGE-LABELS  LABELS !BAG
```

Create a disassembly range from AD1 to AD2 using dis-assembler DEA1 with or without a name. Register it as a labeled range.

105a ⟨*RANGE* 105a⟩≡ (103)
```
: RANGE ( ad1 ad2 dea1 a n -- )   S" CREATE " PAD $!  PAD COUNT + OVER
    2SWAP PAD $+!  PAD $@ EVALUATE  HERE DUP >R  |RANGE| DUP ALLOT  ERASE
    $, R@ >NFA !  CURRENT-RANGE R@ >LFA !  R> TO CURRENT-RANGE
    'RANGE-XT !  1 'RANGE-STRIDE !  'RANGE-END !  'RANGE-START !
    RANGE-LABELS  RANGE-START LAB+!  CURRENT-RANGE LAB+!
    CURRENT-SECTION 'RANGE-SECTION !
    DOES>  RANGE-SECTION ;

: ANON-RANGE ( ad1 ad2 dea1 -- )   NONAME$ RANGE ;
```

105b ⟨*DIS:* 105b⟩≡ (103)
```
: DIS: ( xt -- )   SAVE-INPUT  CREATE  HERE DUP >R  |DISASSEMBLER| DUP
    ALLOT  ERASE  RESTORE-INPUT THROW  BL WORD $@ $, R@ >NFA !
    DISASSEMBLERS R@ >LFA !  R@ >DIS: !  R> TO DISASSEMBLERS
    DOES> ( a1 a2 -- )   >R TARGET>HOST SWAP TARGET>HOST
        DUP NEXT-CUT !  R> >DIS: @ EXECUTE ;
```

105c ⟨*RANGE:* 105c⟩≡ (103)
```
: RANGE: ( xt -- )   SAVE-INPUT  CREATE  HERE DUP >R  |DISASSEMBLER| DUP
    ALLOT  ERASE  RESTORE-INPUT THROW  BL WORD $@ $, R@ >NFA !
    RANGE-RANGES R@ >LFA !  R@ >DIS: !  R> TO RANGE-RANGES
    DOES> ( a1 a2 -- )   >R BL WORD COUNT  R> >DIS: @ EXECUTE ;
```

3.10.8 Disassemble

105d ⟨*disassemble* 105d⟩≡ (85)
```
⟨(D-R-T) 105e⟩
⟨D-R-T 105f⟩
⟨-dc 106a⟩
⟨-dc: 106b⟩
⟨-dc- 106c⟩
' D-R-T  ' -dc:  ARE-COUPLED
```

Disassemble to ADDRESS2 from ADDRESS1.

105e ⟨*(D-R-T)* 105e⟩≡ (105d)
```
: (D-R-T) ( a2 a1 -- )   SWAP DISASSEMBLE-RANGE ;
```

Disassemble from target ADDRESS1 to ADDRESS2.

105f ⟨*D-R-T* 105f⟩≡ (105d)
```
' (D-R-T) DIS: D-R-T ( a1 a2 -- )
```

Range ADDRESS1 .. ADDRESS2 is code with name NAME.

106a ⟨*-dc* 106a⟩≡ (105d)
```
: -dc ( a n -- )   2>R ['] D-R-T 2R> RANGE ;
```

Range ADDRESS1 .. ADDRESS2 is code with name "name".

106b ⟨*-dc:* 106b⟩≡ (105d)
```
' -dc RANGE: -dc: ( -name- )
```

Range ADDRESS1 .. ADDRESS2 is an anonymous code range.

106c ⟨*-dc-* 106c⟩≡ (105d)
```
: -dc- ( -- )   NONAME$ -dc ;
```

3.10.9 Dump unspecified content

106d ⟨*unspecified* 106d⟩≡ (85)
```
⟨(DUMP-N) 106e⟩
⟨DUMP-N 106f⟩
⟨-dn 106g⟩
⟨-dn: 106h⟩
⟨-dn- 106i⟩
' DUMP-N  ' -dn:  ARE-COUPLED
```

Dump storage with unspecified content to ADDRESS2 from ADDRESS1.

106e ⟨*(DUMP-N)* 106e⟩≡ (106d)
```
: (DUMP-N) ( a2 a1 -- )   DUP CR+dn - .LABEL/. S"  RESB" TYPE CR   ;
```

Dump such from target ADDRESS1 to ADDRESS2 where only address1 may be adorned with with a label.

106f ⟨*DUMP-N* 106f⟩≡ (106d)
```
' (DUMP-N) DIS: DUMP-N ( a1 a2 -- )
```

Range ADDRESS1 .. ADDRESS2 are such with name NAME.

106g ⟨*-dn* 106g⟩≡ (106d)
```
: -dn ( a n -- )   2>R ['] DUMP-N 2R> RANGE ;
```

Range ADDRESS1 .. ADDRESS2 are such with name "name".

106h ⟨*-dn:* 106h⟩≡ (106d)
```
' -dn RANGE: -dn: ( -name- )
```

Range ADDRESS1 .. ADDRESS2 is an anonymous such range.

106i ⟨*-dn-* 106i⟩≡ (106d)
```
: -dn- ( -- )   NONAME$ -dn ;
```

3.10.10 Dump bytes

107a ⟨*bytes* 107a⟩≡ (85)
⟨*(DUMP-B)* 107b⟩
⟨*DUMP-B* 107c⟩
⟨*-db* 107d⟩
⟨*-db:* 107e⟩
⟨*-db-* 107f⟩
```
' DUMP-B  ' -db:  ARE-COUPLED \ Register the decompiler.
```

Dump bytes from target ADDRESS1 to ADDRESS2 plain.

107b ⟨*(DUMP-B)* 107b⟩≡ (107a)
```
: (DUMP-B) ( a2 a1 -- )   DO  I DUP CR+db C@ .B-CLEAN  LOOP     PRINT-OLD-COMMENT: CR ;
```

Dump bytes from target ADDRESS1 to ADDRESS2 adorned with labels.

107c ⟨*DUMP-B* 107c⟩≡ (107a)
```
' (DUMP-B) DIS: DUMP-B ( a1 a2 -- )
```

Range ADDRESS1 .. ADDRESS2 are bytes with name NAME.

107d ⟨*-db* 107d⟩≡ (107a)
```
: -db ( a n -- )   2>R ['] DUMP-B 2R> RANGE ;
```

Range ADDRESS1 .. ADDRESS2 are bytes with name "name".

107e ⟨*-db:* 107e⟩≡ (107a)
```
' -db RANGE: -db: ( -name- )
```

Range ADDRESS1 .. ADDRESS2 is an anonymous byte range.

107f ⟨*-db-* 107f⟩≡ (107a)
```
: -db- ( -- )   NONAME$ -db ;
```

3.10.11 Dump words

107g ⟨*words* 107g⟩≡ (85)
⟨*W.* 107h⟩
⟨*(DUMP-W)* 108a⟩
⟨*DUMP-W* 108b⟩
⟨*-dw* 108c⟩
⟨*-dw:* 108d⟩
⟨*-dw-* 108e⟩
```
' DUMP-W  ' -dw:  ARE-COUPLED
```

Print X as a word (4 hex digits).

107h ⟨*W.* 107h⟩≡ (107g)
```
: W. 0 4 (DH.) TYPE SPACE ;
```

Dump words to ADDRESS1 from ADDRESS2, plain.

108a ⟨*(DUMP-W)* 108a⟩≡ (107g)
```
: (DUMP-W) ( a2 a1 -- )   DO  I DUP CR+dw @ W.  2 +LOOP
    PRINT-OLD-COMMENT: CR ;
```

Dump words from target ADDRESS1 to ADDRESS2 adorned with labels.

108b ⟨*DUMP-W* 108b⟩≡ (107g)
```
' (DUMP-W) DIS: DUMP-W ( a1 a2 -- )
```

Range ADDRESS1 .. ADDRESS2 are words with name NAME.

108c ⟨*-dw* 108c⟩≡ (107g)
```
: -dw ( a n -- )   2>R ['] DUMP-W 2R> RANGE ;
```

Range ADDRESS1 .. ADDRESS2 are words with name "name".

108d ⟨*-dw:* 108d⟩≡ (107g)
```
' -dw RANGE: -dw: ( -name- )
```

Range ADDRESS1 .. ADDRESS2 is an anonymous word range.

108e ⟨*-dw-* 108e⟩≡ (107g)
```
: -dw- ( -- )   NONAME$ -dw ;
```

3.10.12 Dump longs

108f ⟨*longs* 108f⟩≡ (85)
⟨*(DUMP-L)* 108g⟩
⟨*DUMP-L* 108h⟩
⟨*-dl* 108i⟩
⟨*-dl:* 109a⟩
⟨*-dl-* 109b⟩
```
' DUMP-L  ' -dl:  ARE-COUPLED
```

Dump longs to ADDRESS1 from ADDRESS2, plain.

108g ⟨*(DUMP-L)* 108g⟩≡ (108f)
```
: (DUMP-L) ( a2 a1 -- )   DO  I DUP CR+dl @ .LABEL/.  4 +LOOP
    PRINT-OLD-COMMENT: CR ;
```

Dump words from target ADDRESS1 to ADDRESS2 adorned with labels.

108h ⟨*DUMP-L* 108h⟩≡ (108f)
```
' (DUMP-L) DIS: DUMP-L ( a1 a2 -- )
```

Range ADDRESS1 .. ADDRESS2 are longs with name NAME.

108i ⟨*-dl* 108i⟩≡ (108f)
```
: -dl ( a n -- )   2>R ['] DUMP-L 2R> RANGE ;
```

Range ADDRESS1 .. ADDRESS2 are longs with name "name".

109a ⟨*-dl:* 109a⟩≡ (108f)
```
' -dl RANGE: -dl: ( -name- )
```

Range ADDRESS1 .. ADDRESS2 is an anonymous long range.

109b ⟨*-dl-* 109b⟩≡ (108f)
```
: -dl- ( -- )   NONAME$ -dl ;
```

3.10.13 Dump strings

109c ⟨*strings* 109c⟩≡ (85)
```
⟨(DUMP-) 109d⟩
⟨DUMP- 109e⟩
⟨-d 109f⟩
⟨-d: 109g⟩
⟨-d- 110a⟩
' DUMP-$  ' -d$:  ARE-COUPLED
```

Print all chars to ADDR1 from ADDR2 appropriately. Try to combine, playing with the next flush.

109d ⟨*(DUMP-)* 109d⟩≡ (109c)
```
: (DUMP-$) ( a2 a1 -- )
    DO  I CR+d$
        I C@ IS-PRINT   IF I ACCU-$C+ ELSE I C@ .C THEN
    LOOP  .ACCU   PRINT-OLD-COMMENT: CR ;

\D 'ADORN-ADDRESS @  'ADORN-ADDRESS RESTORED  HERE
\D S" AAP" $, DROP CTRL J C, CTRL M C, CHAR A C, CHAR A C, BL C, CHAR P C,
\D 0 C, 1 C, BL C, 2 C, 3 C,
\D HERE
\D S" EXPECT  ''3 0 0 0 'AAP' XX ^J ^M 'AA P' 0 1 BL 2 3 ":" TYPE
\D SWAP (DUMP-$) CR  'ADORN-ADDRESS !  22 <?>
```

Dump words from target ADDRESS1 to ADDRESS2 adorned with labels.

109e ⟨*DUMP-* 109e⟩≡ (109c)
```
' (DUMP-$) DIS: DUMP-$ ( a1 a2 -- )
```

Range ADDRESS1 .. ADDRESS2 are longs with name NAME.

109f ⟨*-d* 109f⟩≡ (109c)
```
: -d$ ( a n -- )   2>R ['] DUMP-$ 2R> RANGE ;
```

Range ADDRESS1 .. ADDRESS2 are longs with name "name".

109g ⟨*-d:* 109g⟩≡ (109c)
```
' -d$ RANGE: -d$: ( -name- )
```

Range ADDRESS1 .. ADDRESS2 is an anonymous long range.

110a ⟨*-d-* 110a⟩≡ (109c)
```
: -d$- ( -- )   NONAME$ -d$ ;
```

3.10.14 Ranges

asi386 dependant part, does it belong here?

Not yet definitions, these thingies must be visible in the disassembler.

110b ⟨*ranges* 110b⟩≡ (85)
```
⟨(D-R-T-16) 110c⟩
⟨D-R-T-16 110d⟩
⟨-dc16 110e⟩
⟨-dc16: 110f⟩
⟨-dc16- 110g⟩

' D-R-T-16  ' -dc16:  ARE-COUPLED
```

Disassemble from target ADDRESS1 to ADDRESS2 as 16 bit.

110c ⟨*(D-R-T-16)* 110c⟩≡ (110b)
```
: (D-R-T-16) ( a2 a1 -- )   BITS-16 CR S" BITS-16" TYPE  SWAP DISASSEMBLE-RANGE
    BITS-32 CR S" BITS-32" TYPE ;
```

Disassemble from target ADDRESS1 to ADDRESS2.

110d ⟨*D-R-T-16* 110d⟩≡ (110b)
```
' (D-R-T-16) DIS: D-R-T-16 ( a1 a2 -- )
```

Range ADDRESS1 .. ADDRESS2 is 16-bit code with name NAME.

110e ⟨*-dc16* 110e⟩≡ (110b)
```
: -dc16 ( a n -- )   2>R ['] D-R-T-16   2R> RANGE ;
```

Range ADDRESS1 .. ADDRESS2 is 16-bit code with name "name".

110f ⟨*-dc16:* 110f⟩≡ (110b)
```
' -dc16 RANGE: -dc16: ( -name- )
```

Range ADDRESS1 .. ADDRESS2 is an anonymous 16-bit code-range.

110g ⟨*-dc16-* 110g⟩≡ (110b)
```
: -dc16- ( -- )   NONAME$ -dc16 ;
```

3.10.15 Instructions

111a ⟨*instructions* 111a⟩≡ (85)
⟨*AS-@+* 111b⟩
⟨*AS-S-@+* 111c⟩
⟨*LATEST-OFFSET* 111d⟩
⟨*.COMMA-LABEL* 111e⟩
⟨*ID.-NO()* 111f⟩
⟨*NEXT-INSTRUCTION* 112a⟩
⟨*GET-OFFSET* 112b⟩
⟨*GOAL-RB* 112c⟩
⟨*.BRANCH/.* 112d⟩
⟨*.COMMA-REL* 112e⟩
⟨*UNCONDITIONAL-TRANSFERS* 113a⟩
⟨*JUMPS* 113b⟩

DEA is a commaer. Fetch proper DATA from autoincremented '**AS-POINTER**'

111b ⟨*AS-@+* 111b⟩≡ (111a)
```
: AS-@+ ( xt -- )   >CNT @ >R  AS-POINTER @ R@ MC@  R> AS-POINTER +! ;

\D HEX
\D S" EXPECT 34 12 " TYPE 1234 PAD ! PAD AS-POINTER !
\D ' IB, >BODY DUP AS-@+ . AS-@+ . CR 23 <?>
\D DECIMAL
```

DEA is a commaer. Fetch proper signed DATA from autoincremented '**AS-POINTER**'

111c ⟨*AS-S-@+* 111c⟩≡ (111a)
```
: AS-S-@+ ( xt -- )   >CNT @ >R  AS-POINTER @ R@ MC@-S  R> AS-POINTER +! ;

\D HEX
\D S" EXPECT -1 12 " TYPE 12FF PAD ! PAD AS-POINTER !
\D ' IB, >BODY DUP AS-S-@+ . AS-S-@+ . CR
\D DECIMAL
```

This is kept up to date during disassembly. It is useful for the code crawler.

111d ⟨*LATEST-OFFSET* 111d⟩≡ (111a)
```
VARIABLE LATEST-OFFSET
```

Print a disassembly, for a commaer DEA , taking into account labels, (suitable for e.g. the commaer '**IX,**').

111e ⟨*.COMMA-LABEL* 111e⟩≡ (111a)
```
: .COMMA-LABEL   DUP AS-@+ .LABEL/. %ID. ;
```

For DEA print the name without the surrounding brackets.

111f ⟨*ID.-NO()* 111f⟩≡ (111a)
```
: ID.-NO() ( xt -- )   >NFA @ $@  2 - SWAP 1 + SWAP TYPE SPACE ;
```

Assuming the disassembly sits at the offset of a relative branch assembled by commaer DEA , return the host space ADDRESS of the next instruction.

112a ⟨*NEXT-INSTRUCTION* 112a⟩≡ (111a)
```
: NEXT-INSTRUCTION  >CNT @ AS-POINTER @ + ;
```

Assuming the disassembly sits at the offset of a relative branch assembled by commaer DEA , return that OFFSET.

112b ⟨*GET-OFFSET* 112b⟩≡ (111a)
```
: GET-OFFSET   AS-POINTER @ SWAP >CNT @ MC@-S DUP LATEST-OFFSET ! ;
```

For the commaer DEA return ADDRESS in host space that is the target of the current relative jump.

112c ⟨*GOAL-RB* 112c⟩≡ (111a)
```
: GOAL-RB   DUP GET-OFFSET SWAP NEXT-INSTRUCTION + ;
```

For the relative branch commaer DEA print the target ADDRESS as a symbolic label if possible else print the branch offset, followed by an appropriate commaer for each case.

112d ⟨*.BRANCH/.* 112d⟩≡ (111a)
```
: .BRANCH/.  EQU-LABELS   ~LABEL OVER IF .~LABEL ID.-NO() ELSE
    2DROP DUP GET-OFFSET . %ID. THEN  ;
```

Print a disassembly for a relative branch DEA. This relies on the convention that the commaer that consumes an absolute address has the name of that with a relative address surrounded with brackets.

112e ⟨*.COMMA-REL* 112e⟩≡ (111a)
```
: .COMMA-REL
    DUP  DUP GOAL-RB HOST>TARGET  .BRANCH/.
    >CNT @ AS-POINTER +! ;

\D 5 .LABEL/. CR
\D 5 .LABEL/. CR
\D ' (RB,) >BODY ID.-NO() CR

' .COMMA-LABEL  ' OW,   >BODY >DIS !  ( obligatory word     )
' .COMMA-REL    ' (RL,) >BODY >DIS !  ( cell relative to IP )
' .COMMA-REL    ' (RW,) >BODY >DIS !  ( cell relative to IP )
' .COMMA-REL    ' (RB,) >BODY >DIS !  ( byte relative to IP )
' .COMMA-LABEL  ' SG,   >BODY >DIS !  (  Segment: WORD      )
' .COMMA-LABEL  ' P,    >BODY >DIS !  ( port number ; byte     )
' .COMMA-LABEL  ' IS,   >BODY >DIS !  ( Single -obl-  byte )
' .COMMA-LABEL  ' IL,   >BODY >DIS !  ( immediate data : cell)
' .COMMA-LABEL  ' IW,   >BODY >DIS !  ( immediate data : cell)
' .COMMA-LABEL  ' IB,   >BODY >DIS !  ( immediate byte data)
' .COMMA-LABEL  ' L,    >BODY >DIS !  ( immediate data : address/offset )
' .COMMA-LABEL  ' W,    >BODY >DIS !  ( immediate data : address/offset )
' .COMMA-LABEL  ' B,    >BODY >DIS !  ( immediate byte : address/offset )
```

Contains all instruction that represent an unconditional transfer of control. It may be followed by data instead of code.

113a ⟨*UNCONDITIONAL-TRANSFERS* 113a⟩≡ (111a)

```
0 BAG UNCONDITIONAL-TRANSFERS
    ' CALL, , ' CALLFAR, , ' CALLFARO, , ' CALLO, , ' INT, , ' INT3, , ' INTO, ,
    ' IRET, , ' JMP, , ' JMPFAR, , ' JMPFARO, , ' JMPO, , ' JMPS, , ' RET+, ,
    ' RET, , ' RETFAR+, , ' RETFAR, ,
HERE UNCONDITIONAL-TRANSFERS ! 100 CELLS ALLOT        \ Allow to put more here
```

Contains all instructions that represent intra-segment jumps.

113b ⟨*JUMPS* 113b⟩≡ (111a)

```
0 BAG JUMPS
    ' CALL, , ' J, , ' JCXZ, , ' JMP, , ' JMPS, , ' J|X, ,
    ' LOOP, , ' LOOPNZ, , ' LOOPZ, ,
HERE JUMPS ! 100 CELLS ALLOT        \ Allow to put more here
```

3.11 Crawler

Crawling is the process of following jumps to determine code space.

114 ⟨*crawl.frt* 114⟩≡ (22)

```
( $Id: crawl.frt,v 1.36 2009/03/26 09:07:17 albert Exp $ )
( Copyright{2000}: Albert van der Horst, HCC FIG Holland by GNU Public License)
( Uses Richard Stallmans convention. Uppercased word are parameters.    )

[DEFINED] ForCiForth [IF]
    REQUIRE H.
    REQUIRE BAG
[THEN]

\ : \D ;
```

⟨*INSERT-EQU* 115a⟩
⟨*INSERT-EQU-INVENT* 115b⟩
⟨*?INSERT-EQU?* 116a⟩

```
\D HEX
\D RANGE-LABELS       LABELS !BAG
\D 4FE 510 -dc-
\D 520 530 -dc: oops
\D 530 570 -dc-
\D 560 590 -db: bytes
\D S" EXPECT: 4 ranges :" TYPE CR .LABELS CR 25 <?>
\D DECIMAL
```

⟨*COMPATIBLE?* 116b⟩
⟨*RANGE-NAME* 116c⟩

```
[DEFINED] ForCiForth [IF]
    \D REQUIRE H.
[THEN]
```

⟨*NEW-RANGE-START* 116d⟩
⟨*NEW-RANGE-END* 116e⟩
⟨*REPLACE* 116f⟩
⟨*SAME-ALIGN* 117a⟩
⟨*END+START* 117b⟩
⟨*OVERLAP?* 117c⟩
⟨*OVERLAP-OR-BORDER?* 118a⟩
⟨*GAP?* 118b⟩
⟨*IS-NAMED* 118c⟩
⟨*COLLAPSE* 118d⟩
⟨*TRIM-RANGE* 118e⟩

⟨*COMBINE* 119⟩
⟨*KILL-OVERLAP* 120a⟩
⟨*FILL-GAP* 120b⟩
⟨*CLEANUP-RANGES* 121a⟩
⟨*PLUG-FIRST* 121b⟩
⟨*PLUG-LAST* 121c⟩
⟨*PLUG-SPECIAL* 121d⟩
⟨*PLUG-HOLES* 121e⟩

```
ALSO ASSEMBLER
```

⟨*STARTERS* 121f⟩
⟨*REQUIRED-XT* 121g⟩
⟨*NORMAL-DISASSEMBLY* 122a⟩
⟨*IN-CURRENT-CODE?* 122b⟩
⟨*IN-CODE-N?* 122c⟩
⟨*IN-CODE?* 122d⟩
⟨*KNOWN-CODE?* 122e⟩
⟨*IN-CODE-SPACE?* 122f⟩
⟨*STARTER?* 122g⟩
⟨*JUMP-TARGET* 122h⟩
⟨*ANALYSE-INSTRUCTION* 122i⟩
⟨*COLLAPSE(I1)* 123a⟩
⟨*INSERT-RANGE* 123b⟩
⟨*CRAWL-ONE* 123c⟩
⟨*?CRAWL-ONE?* 123d⟩
⟨*(CRAWL)* 123e⟩
⟨*CRAWL* 123f⟩
⟨*dl-range* 124a⟩
⟨*CRAWL16* 123g⟩

```
PREVIOUS
: \D ;
```

Insert the equ-label ADDRESS1 with an NAME. If equ labels was sorted, it remains so.

115a ⟨*INSERT-EQU* 115a⟩≡ (114)

```
: INSERT-EQU 2>R DUP EQU-LABELS WHERE-LABEL SWAP 2R> LABELED
    ROLL-LABEL ;
```

Insert the equ-label ADDRESS1 with an invented name. If equ labels was sorted, it remains so.

115b ⟨*INSERT-EQU-INVENT* 115b⟩≡ (114)

```
: INSERT-EQU-INVENT DUP INVENT-NAME INSERT-EQU ;
```

Add target ADDRESS to the equ labels if it is not there. Invent a name.

116a ⟨*?INSERT-EQU?* 116a⟩≡ (114)
```
: ?INSERT-EQU?    EQU-LABELS DUP >LABEL IF DROP ELSE INSERT-EQU-INVENT THEN ;

\D EQU-LABELS LABELS !BAG
\D S" EXPECT: empty " TYPE EQU-LABELS .LABELS  CR
\D 42 ?INSERT-EQU?
\D S" EXPECT: L00000042 added " TYPE EQU-LABELS .LABELS     CR
\D 42 ?INSERT-EQU?
\D S" EXPECT: L00000042 NOT added again " TYPE EQU-LABELS .LABELS     CR 24 <?>
```

For range INDEX : "it is of the same type as the previous one".

116b ⟨*COMPATIBLE?* 116b⟩≡ (114)
```
: COMPATIBLE?   DUP MAKE-CURRENT RANGE-XT   SWAP 1- MAKE-CURRENT RANGE-XT  = ;

\D S" EXPECT: -1 :" TYPE 2 COMPATIBLE? . CR
\D S" EXPECT: -1 :" TYPE 3 COMPATIBLE? . CR
\D S" EXPECT: 0 :" TYPE 4 COMPATIBLE? . CR
```

Get the name of range INDEX.

116c ⟨*RANGE-NAME* 116c⟩≡ (114)
```
: RANGE-NAME LABELS[] CELL+ @ >NFA @ $@ ;

\D S" EXPECT: NONAME :" TYPE 1 RANGE-NAME TYPE CR
\D S" EXPECT: oops :" TYPE 2 RANGE-NAME TYPE CR 26 <?>
```

For a collapsible pair of range with INDEX1 and INDEX2 return INDEX1 and INDEX2 plus a new START for the combined range.

116d ⟨*NEW-RANGE-START* 116d⟩≡ (114)
```
: NEW-RANGE-START OVER MAKE-CURRENT RANGE-START  OVER MAKE-CURRENT RANGE-START
    MIN ;

\D  S" EXPECT 520 : " TYPE 2 3 NEW-RANGE-START H. 2DROP CR
```

For a collapsible pair of range with INDEX1 and INDEX2 return INDEX1 and INDEX2 plus a new END for the combined range.

116e ⟨*NEW-RANGE-END* 116e⟩≡ (114)
```
: NEW-RANGE-END OVER MAKE-CURRENT RANGE-END  OVER MAKE-CURRENT RANGE-END
    MAX ;

\D  S" EXPECT 590 : " TYPE 3 4 NEW-RANGE-END  H. 2DROP CR 27 <?>
```

Replace the two ranges INDEX1 and INDEX2 with the last range. Place it at index1 (which has the correct start address).

116f ⟨*REPLACE* 116f⟩≡ (114)
```
: REPLACE  OVER >R REMOVE-LABEL REMOVE-LABEL R> ROLL-LABEL ;

\D  S" EXPECT 1 LESS : " TYPE 2 3 REPLACE CR .LABELS CR 28 <?>
```

This looks like a proper design.

- sort on the start address, type (disassembler) and end address.
- start with the last range and work down until the second.
- if it overlaps with or borders at the previous one and has the same type and alignment, and the second one is not named collapse the ranges.
- if it overlaps with the previous one and has the same type and alignment, and the second one is named, trim the first range.
- if it overlaps with the previous and has different type, issue warning.
- if it has a gap, introduce a character range. This may lead to an extra range, one less range, or no change in the number of ranges, but only at or after the current range.
- As a last action, introduce extra ranges at the beginning and end.

This leads to words: **SAME-TYPE SAME-ALIGN OVERLAP BORDER GAP IS-NAMED**.

For range INDEX: "It has the same type and alignment as the previous one."

117a ⟨*SAME-ALIGN* 117a⟩≡ (114)

```
: SAME-ALIGN    DUP MAKE-CURRENT  RANGE-START SWAP
    1- MAKE-CURRENT   RANGE-START - RANGE-STRIDE MOD 0= ;

\D INIT-ALL RANGE-LABELS  HEX
\D 12 34 -dc-
\D 34 65 -db: AAP
\D 38 80 -dl-
\D 82 90 -dl-
\D 88 94 -dl-

\D S" EXPECT: -1 :" TYPE 2 SAME-ALIGN . CR
\D S" EXPECT: -1 :" TYPE 3 SAME-ALIGN . CR \ Must become 0
\D S" EXPECT: -1 :" TYPE 4 SAME-ALIGN . CR
```

For range INDEX return END of previous, START of this one,

117b ⟨*END+START* 117b⟩≡ (114)

```
: END+START DUP MAKE-CURRENT RANGE-START SWAP 1- MAKE-CURRENT RANGE-END SWAP ;

\D S" EXPECT: 34 34 :" TYPE 2 END+START SWAP . . CR
\D S" EXPECT: 65 38 :" TYPE 3 END+START SWAP . . CR
```

Range INDEX overlaps with previous one.

117c ⟨*OVERLAP?* 117c⟩≡ (114)

```
: OVERLAP? END+START > ;

\D S" EXPECT: 0 :" TYPE 2 OVERLAP? . CR
\D S" EXPECT: -1 :" TYPE 3 OVERLAP? . CR
\D S" EXPECT: 0 :" TYPE 4 OVERLAP? . CR
```

Range INDEX overlaps or borders with the previous one.

118a ⟨*OVERLAP-OR-BORDER?* 118a⟩≡ (114)
```
: OVERLAP-OR-BORDER? END+START >= ;

\D S" EXPECT: -1 :" TYPE 2 OVERLAP-OR-BORDER? . CR
\D S" EXPECT: -1 :" TYPE 3 OVERLAP-OR-BORDER? . CR
\D S" EXPECT: 0 :" TYPE 4 OVERLAP-OR-BORDER? . CR
```

Range INDEX has a gap with the previous one.

118b ⟨*GAP?* 118b⟩≡ (114)
```
: GAP? END+START < ;

\D S" EXPECT: 0 :" TYPE 2 GAP? . CR
\D S" EXPECT: 0 :" TYPE 3 GAP? . CR
\D S" EXPECT: -1 :" TYPE 4 GAP? . CR
```

For range INDEX: "It has a name"

118c ⟨*IS-NAMED* 118c⟩≡ (114)
```
: IS-NAMED ( n -- )   RANGE-NAME NONAME$ $= 0= ;

\D S" EXPECT: -1 :" TYPE 2 IS-NAMED . CR
\D S" EXPECT: 0 :" TYPE 3 IS-NAMED . CR
```

Collapse range I into the previous range, that determines the properties.

118d ⟨*COLLAPSE* 118d⟩≡ (114)
```
: COLLAPSE ( i -- )   DUP MAKE-CURRENT  RANGE-END OVER 1- MAKE-CURRENT
    RANGE-END MAX RANGE-END!  REMOVE-LABEL ;

\D S" EXPECT: 5 82 94 4 :" TYPE
\D LAB-UPB . 5 COLLAPSE 4 MAKE-CURRENT RANGE-START . RANGE-END .  LAB-UPB . CR
```

Trim the range previous to INDEX, such that it borders to range index.

118e ⟨*TRIM-RANGE* 118e⟩≡ (114)
```
: TRIM-RANGE ( i -- )   DUP MAKE-CURRENT  RANGE-START SWAP 1- MAKE-CURRENT
    RANGE-END! ;

\D 90 1000 -dl-
\D S" EXPECT: 82 90 :" TYPE 5 TRIM-RANGE 4 MAKE-CURRENT
\D RANGE-START . RANGE-END .  CR
```

Combine range INDEX with the previous one.

119 ⟨*COMBINE* 119⟩≡ (114)

```
: COMBINE ( n -- )
    DUP OVERLAP-OR-BORDER? OVER IS-NAMED 0= AND IF DUP COLLAPSE THEN
    DUP OVERLAP? OVER IS-NAMED AND IF DUP TRIM-RANGE THEN  DROP ;

\D INIT-ALL
\D 10  30 -dl-
\D 20  40 -dl-
\D 30 50  -dl: aap
\D 60 80  -dl-
\D 90 100 -dl: noot
\D S" EXPECT: 5 5 :" TYPE LAB-UPB . 5 COMBINE LAB-UPB . CR
\D S" EXPECT: 5 5 :" TYPE LAB-UPB . 4 COMBINE LAB-UPB . CR
\D S" EXPECT: 5 5 20 30 :" TYPE LAB-UPB . 3 COMBINE LAB-UPB . 2 MAKE-CURRENT
\D RANGE-START . RANGE-END . CR
\D S" EXPECT: 5 4 10 30 :" TYPE LAB-UPB . 2 COMBINE LAB-UPB . 1 MAKE-CURRENT
\D RANGE-START . RANGE-END . CR
```

Combine range INDEX with a previous overlapping or bordering range.

120a ⟨*KILL-OVERLAP* 120a⟩≡ (114)

```
: KILL-OVERLAP ( i -- )   DUP SAME-ALIGN  OVER COMPATIBLE? AND IF
        DUP COMBINE  THEN  DROP ;

\D INIT-ALL
\D 10  30 -dl-
\D 20  40 -dl-
\D 30 50  -dl: aap
\D 60 80  -dl-
\D 90 100 -dl: noot
\D S" EXPECT: 5 5 :" TYPE LAB-UPB . 5 KILL-OVERLAP LAB-UPB . CR
\D S" EXPECT: 5 5 :" TYPE LAB-UPB . 4 KILL-OVERLAP LAB-UPB . CR
\D S" EXPECT: 5 5 20 30 :" TYPE LAB-UPB . 3 KILL-OVERLAP LAB-UPB .
\D 2 MAKE-CURRENT RANGE-START . RANGE-END . CR
\D S" EXPECT: 5 4 10 30 :" TYPE LAB-UPB . 2 KILL-OVERLAP LAB-UPB .
\D 1 MAKE-CURRENT RANGE-START . RANGE-END . CR
\D INIT-ALL
\D 10  30 -dl-
\D 20  28 -db-
\D 30 70  -dl: aap
\D 60 80  -dl-
\D 7F 10F -dl-
\ The following is actually wrong because the aligning is not tested yet.
\D S" EXPECT: 5 4 60 10F :" TYPE LAB-UPB . 5 KILL-OVERLAP LAB-UPB .
\D 4 MAKE-CURRENT RANGE-START . RANGE-END . CR
\D S" EXPECT: 4 3 30 10F :" TYPE LAB-UPB . 4 KILL-OVERLAP LAB-UPB .
\D 3 MAKE-CURRENT RANGE-START . RANGE-END . CR
\D S" EXPECT: 3 3 20 28 :" TYPE LAB-UPB . 3 KILL-OVERLAP LAB-UPB .
\D 2 MAKE-CURRENT RANGE-START . RANGE-END . CR
\D S" EXPECT: 3 3 10 30 :" TYPE LAB-UPB . 2 KILL-OVERLAP LAB-UPB .
\D 1 MAKE-CURRENT RANGE-START . RANGE-END . CR
```

Introduce char range to fill the gap at INDEX. Note that the result is unordered.

120b ⟨*FILL-GAP* 120b⟩≡ (114)

```
: FILL-GAP ( i -- )   DUP GAP? IF   DUP END+START -ddef-
        DUP 1+ LAB-UPB MAX KILL-OVERLAP
        DUP KILL-OVERLAP  THEN  DROP ;

\D S" EXPECT: 3 4 28 30 :" TYPE LAB-UPB . 3 FILL-GAP LAB-UPB .
\D 4 MAKE-CURRENT RANGE-START . RANGE-END . CR
\D S" EXPECT: 4 4 20 28 :" TYPE LAB-UPB . 2 FILL-GAP LAB-UPB .
\D 2 MAKE-CURRENT RANGE-START . RANGE-END . CR
```

Clean up the range labels, from behind. Although the bounds may not be valid after a clean up, this works because a clean up of a range only concerns higher ranges, no longer considered. So a range can comfortably be removed using the regular removal mechanism for bags. A newly introduced range automatically falls into place, because of the conditions regarding the start addresses.

121a ⟨*CLEANUP-RANGES* 121a⟩≡ (114)
```
: CLEANUP-RANGES ( -- )   RANGE-LABELS
    2 LAB-UPB 2DUP <= IF  DO  I KILL-OVERLAP  -1 +LOOP  THEN ;
```

Plug a hole at the first range.

121b ⟨*PLUG-FIRST* 121b⟩≡ (114)
```
: PLUG-FIRST ( -- )   1 MAKE-CURRENT
    TARGET-START RANGE-START 2DUP <> IF
        -ddef- _ _  THEN  2DROP ;
```

Plug a hole at the last range.

121c ⟨*PLUG-LAST* 121c⟩≡ (114)
```
: PLUG-LAST ( -- )   LAB-UPB MAKE-CURRENT
    RANGE-END TARGET-END 2DUP <> IF
        -ddef- _ _  THEN  2DROP ;
```

If there are no ranges at all, make the buffer into a default range. Else check last and first ranges. Note that plugging results in a change of the number of ranges, interfering with other plugging.

121d ⟨*PLUG-SPECIAL* 121d⟩≡ (114)
```
: PLUG-SPECIAL ( -- )   LAB-UPB IF  PLUG-LAST PLUG-FIRST  ELSE
    TARGET-START TARGET-END -ddef-  THEN ;
```

Fill any holes with character ranges.

121e ⟨*PLUG-HOLES* 121e⟩≡ (114)
```
: PLUG-HOLES ( -- )   CURRENT-SECTION  RANGE-LABELS LAB-UPB 1+ 2
    2DUP > IF  DO  I FILL-GAP  LOOP  ELSE  2DROP  THEN
    SORT-LABELS  PLUG-SPECIAL  SORT-LABELS
    TO CURRENT-SECTION ;
```

Jump targets that are starting points for further crawling. Adding and removing from this bag ressembles a recursive action. Recursion will not do here! This is because ranges are not added until the end is detected.

121f ⟨*STARTERS* 121f⟩≡ (114)
```
1000 BAG STARTERS
```

Required xt. Return the XT that is required for the current disassembly.

121g ⟨*REQUIRED-XT* 121g⟩≡ (114)
```
VARIABLE (R-XT)
: REQUIRED-XT (R-XT) @ ;
```

Specify normal disassembly.

122a ⟨*NORMAL-DISASSEMBLY* 122a⟩≡ (114)
```
: NORMAL-DISASSEMBLY ['] D-R-T (R-XT) ! BITS-32 ;
  NORMAL-DISASSEMBLY
```

The following are auxiliary words for '**KNOWN-CODE?**' mainly. For all those range labels must be current and sorted. Prepend '**RANGE-LABELS**' if you want to use the auxiliary words.

For ADDRESS : "it is in a current code range"

122b ⟨*IN-CURRENT-CODE?* 122b⟩≡ (114)
```
: IN-CURRENT-CODE? ( -- f )   RANGE-START RANGE-END WITHIN
    RANGE-XT REQUIRED-XT =  AND ;
```

For ADDRESS and range number N: "address SITS in code range n"

122c ⟨*IN-CODE-N?* 122c⟩≡ (114)
```
: IN-CODE-N? ( i -- f ) MAKE-CURRENT IN-CURRENT-CODE? ;
```

For ADDRESS and range I : "It is code and address is part of it, or same holds for previous range."

122d ⟨*IN-CODE?* 122d⟩≡ (114)
```
: IN-CODE?  DUP 0 = IF 2DROP 0 ELSE   \ Not present.
        2DUP IN-CODE-N? IF 2DROP -1 ELSE
            DUP 1 = IF 2DROP 0 ELSE   \ Previous not present.
                1- IN-CODE-N? THEN THEN THEN ;
```

For ADDRESS: "It is known code, according to '**RANGE-LABELS**'".

122e ⟨*KNOWN-CODE?* 122e⟩≡ (114)
```
: KNOWN-CODE?   RANGE-LABELS DUP WHERE-LABEL LAB-UPB MIN IN-CODE? ;
```

For ADDRESS : "it falls within the binary image"

122f ⟨*IN-CODE-SPACE?* 122f⟩≡ (114)
```
: IN-CODE-SPACE?   TARGET-START TARGET-END WITHIN ;
```

For ADDRESS: "It is usable as a new starter".

122g ⟨*STARTER?* 122g⟩≡ (114)
```
: STARTER?   DUP KNOWN-CODE? 0=  SWAP IN-CODE-SPACE? AND ;
```

Return the target ADDRESS of the current instruction. It must be a jump of course.

122h ⟨*JUMP-TARGET* 122h⟩≡ (114)
```
: JUMP-TARGET   AS-POINTER @   LATEST-OFFSET @  + HOST>TARGET ;
```

Analyse current instruction after disassembly. **DISS LATEST-INSTRUCTION ISS ISL** are all valid.

122i ⟨*ANALYSE-INSTRUCTION* 122i⟩≡ (114)
```
: ANALYSE-INSTRUCTION   LATEST-INSTRUCTION @ JUMPS IN-BAG? IF
    JUMP-TARGET DUP ?INSERT-EQU?
    STARTER? IF JUMP-TARGET STARTERS SET+ THEN THEN ;
```

Collapse the label at INDEX with the next and or previous labels.

123a ⟨*COLLAPSE(I1)* 123a⟩≡ (114)
```
: COLLAPSE(I1) RANGE-LABELS
    DUP LAB-UPB < IF DUP 1+ KILL-OVERLAP THEN
    DUP 1 > IF DUP KILL-OVERLAP THEN
    DROP ;

\D LABELS !BAG
\D 4FE 520 -dc-
\D 520 530 -dc: oops
\D 52A 570 -dc-
\D 560 590 -db: bytes \D .LABELS
\D S" EXPECT 1 LESS : " TYPE 2 COLLAPSE(I1) CR .LABELS CR 29 <?>
```

Add the information that ADDRESS1 to ADDRESS2 is a code range. If range labels was sorted, it remains so.

123b ⟨*INSERT-RANGE* 123b⟩≡ (114)
```
: INSERT-RANGE   OVER RANGE-LABELS WHERE-LABEL >R
    REQUIRED-XT ANON-RANGE   R@ ROLL-LABEL   R> COLLAPSE(I1) ;
```

Analyse the code range from ADDRESS up to an unconditional transfer. Add information about jumps to '**STARTERS**' and new ranges to '**LABELS**'.

123c ⟨*CRAWL-ONE* 123c⟩≡ (114)
```
: CRAWL-ONE  DUP >R TARGET>HOST BEGIN
        (DISASSEMBLE) ANALYSE-INSTRUCTION DUP HOST-END >=
        LATEST-INSTRUCTION @ UNCONDITIONAL-TRANSFERS IN-BAG? OR
    UNTIL R> SWAP HOST>TARGET INSERT-RANGE ;
```

Analyse code from ADDRESS , unless already known.

123d ⟨*?CRAWL-ONE?* 123d⟩≡ (114)
```
: ?CRAWL-ONE? DUP STARTER? IF CRAWL-ONE _ THEN DROP ;
```

Crawl through code from all points in '**STARTERS**'.

123e ⟨*(CRAWL)* 123e⟩≡ (114)
```
: (CRAWL)   BEGIN STARTERS BAG? WHILE STARTERS BAG@- ?CRAWL-ONE? REPEAT ;
```

ADDRESS points into code. Crawl through code from there, i.e. add all information about code ranges that can be derived from that.

123f ⟨*CRAWL* 123f⟩≡ (114)
```
: CRAWL   DUP ?INSERT-EQU?   RANGE-LABELS SORT-LABELS
    STARTERS DUP !BAG BAG+!   SHUTUP (CRAWL) ;
```

INTEL 80386 specific. There is a need to specify the disassembly xt. Crawl with normal disassembly (observing '**TALLY-BA**') resp. Crawl through 16 / 32 bits code. The other owns change it all the time.

123g ⟨*CRAWL16* 123g⟩≡ (114)
```
: CRAWL16  ['] D-R-T-16 (R-XT) ! BITS-16 CRAWL NORMAL-DISASSEMBLY ;
```

3.11.1 DL range

124a ⟨*dl-range* 124a⟩≡ (114)
⟨*NEW-LABEL?* 124b⟩
⟨*ADD-L-LABELS* 124c⟩
⟨*ALL-L-LABELS* 124d⟩

For ADDR create a label if it points in the target space.

124b ⟨*NEW-LABEL?* 124b⟩≡ (124a)
```
: NEW-LABEL? ( a -- )   DUP PLAUSIBLE-LABEL? IF  ?INSERT-EQU? _  THEN  DROP ;
```

For dl-range from ADDR1 to ADDR2 add all plausible labels found in data.

124c ⟨*ADD-L-LABELS* 124c⟩≡ (124a)
```
: ADD-L-LABELS ( l h -- )   SWAP DO  I L@ NEW-LABEL?  0 CELL+ +LOOP ;
```

For all dl-ranges add all plausible labels.

124d ⟨*ALL-L-LABELS* 124d⟩≡ (124a)
```
: ALL-L-LABELS ( -- )   CURRENT-SECTION  RANGE-LABELS DO-LAB
        I CELL+ @ RANGE-SECTION  RANGE-XT ['] DUMP-L = IF
            RANGE-START RANGE-END ADD-L-LABELS  THEN
    LOOP-LAB  TO CURRENT-SECTION ;
```

4 Extracting code

A script for converting this document to PDF form follows:

124e ⟨*final* 124e⟩≡
```
lyx -e pdf cfasdis.lyx
```

124f ⟨* 124f⟩≡
```
echo "Extract script $1 from cfasdis.lyx..."
rm -f cfasdis.nw
lyx -e literate cfasdis.lyx
notangle -R$1 cfasdis.nw > $1
chmod a+x $1
```

Each of these scripts can be pulled out manually given the default `*` script defined above.