A6 base address of OT, receive offsets globals

A5 Address of HaveCabert Context

A4 IP (address of next byte code)

A3 CSP (address of top of context stack)

A2

A1

A0

D7 ~~Stack pointer (word offset from A5)~~ ?

D6 Restart ~~offset~~ address

D5 receive address oop?

D4 Method Base Addr (literal access)

D3

D2

D1

D0

Global Variables

Active Context : oop

Active Context Base : address

Method : oop

Method Base : address

Byte code Dispatch

Table of 32-bit addresses (1k bytes)

```
CycleHead: ; process next byte code
4+1    MOVEQ    #Ø, DØ
8+2    MOVE.B   (IP)+, DØ         ; fetch next byte code
4+1    ADD.W    DØ, DØ  }  LSL.W #2, DØ ; 10+1
4+1    ADD.W    DØ, DØ  }  ; convert byte code to longword offset
18+4   MOVE.L   Table(PC, DØ.W), AØ
8+2    JMP      AØ
46+11  or  48+10 if shift
```

Table of 16 bit offsets (.5k bytes)

```
CycleHead:
4+1    MOVE Q   #, DØ
8+2    MOVE.B   (IP)+, DØ
4+1    ADD.W    DØ, DØ
14+3   MOVE.W   TABLE(PC, DØ.W), DØ
14+3   JMP      Ø(PC, DØ)
44+10
```

---

table of BRA's (.5k bytes)

```
4+1    MOVE Q   #Ø, DØ
8+2    MOVE.B   (IP)+, DØ
4+1    ADD.W    DØ, DØ
4+1    ADD.W    DØ, DØ
14+3   JMP      TABLE(PC, DØ.W)

1Ø +2
entry  BRA.L    n
44+10  or  46+9
```

No table
16 bytes / byte code

~~16~~ 32 bytes / byte code

```
4+1    MOVEQ    #0, D0
8+2    MOVE.B   (IP)+, D0
16+1   LSL.W    #5, D0
14+3   JMP      bytecodes(SP, D0.W)
42+7
```

possible byte code sequences

```
bytes
4      MOVE.W   n(Home), D0
4      CNTUP    D0
       ~~ADDQ   #2, CSP~~
4      CNTDOWN  stack top
2      MOVE.W   D0, (CSP)
4      JMP      0(A6, D0)
20
```

Table of 16 bit (15) absolute addresses

```
4+1    MOVEQ    #0, D0
8+2    MOVE.B   (IP)+, D0
4+1    AND.W    D0, D0
14+3   MOVE.W   TABLE(PC, D0.W), A0
8+2    JMP      (A0)
38+9

8      MOVE.W   #<Table div 2>, D0
8      MOVE.B   (IP)+, D0
4      ADD.W    D0, D0
4      MOVEA.W  D0, A0
8      MOVE.W   (A0), A0
       JMP      (A0)
40
```

Next interpreter cycle alternatives:

Addr Top of loop on stack
RTS 16 + 4
Restore TOS: PFA 16 + 2 + 2
change TOS to indicate
asychronous event
32 + 6 + 2

Addr Top of loop in Register
JMP (Areg) 8 + 2
don't need to restore
change register to
indicate event
8 + 2

Addr of routine in global
MOVE.L ; JMP () 16 + 4
don't need to restore 8 + 2
change global for event
24 + 6

BRA to Loop, test boolean
TST.B 10 + 2
Bcc 12 + 3
8 + 1
30 + 6

Data Register contains Displacement of
Loop from OF Base
JMP Φ(A6, Dx) 14 + 3

---

self mod. code doesn't need register
BRA To loop
Loop head is noop or BRA To interrupt
10 + 2
4 + 1

```
pushInstVar(n)
define(pushInstVar, [
ldinst$1:
        trace1or2('Push Receive Instance Variable $1')
    trace_fetchpember_ofObject__(receiverOop, $1)
        MOVEA.L     receiveReg, A0
        MOVE.W      $1*2, D0
    increaseReferences (A0)
    Addq  #2, CSP
    decrease References ((CSP))
     move.w         D0, (CSP)
      Next Cycle
    ])
```

```
; push InstVar (n)

!dinst $1:
    trace1or2( Push Receiver Instance Variable !$ )
     MOVE A,L    receiverReg, A0
     move.w      $1*2+4(A0), D0 ; get instance var
     trace_fetchpointer (receiverOop, $1)
     moveq       #0, D1                BTST #0, D0
      move.w     D0, D1                BEQ.S !$
      ADD.W      D1, D1
      ADDQ.B     #upCount, 0(OTbase, D1.L)
      BVC.S      !$
      MOVE.B     #hiCount, 0(OTbase, D1.L)
!$:    ADDQ      #2, CSP
       MOVE.W    (CSP), D1
       BTST      #0, D1
       BEQ.S     2$
        ADDQW    D1, D1
        SUB.Q.B  #downCount, 0(OTbase, D1.L)
        BVC.S    2$
        MOVE.B   #hiCount, 0(otbase, D1.L)
        MOVE.W   D0, (CSP)
      trace_Push (D0)
       Next Cycle
```

; The main loop

executeLoop:

```
        MOVEQ     #0, D0
        MOVE.B    (IP)+, D0   ; get next bytecode
        ADD.W     D0, D0      ; convert to word offset
        MOVE.W    ByteCodeDispatch(PC, D0.W), A0
        JMP       (A0)
```

ByteCodeDispatchTable

```
        ByteCode(0, ldinst0)
        ByteCode(1, ldinst1)
        ByteCode(2, ldinst2)
          :
        ByteCode(15, ldinst15)
        ByteCode(16, ldtemp0)
          :
        ByteCode(31, ldtemp15)
        ByteCode(32, ldlit0
          :
        ByteCode(63, ldlit31)
        ByteCode(64, ldliti0)
          :
        ByteCode(95, ldliti31
```

```
* 1dinst0 : 4+4 move A.1        receiver Addr, A0
*          12+3 Move.w          4(A0), D0; inst var value

*          8+2 Btst             #0, D0
*          10+2 BEQ.S           1$
*          4+1 MOVEq            #0, D1
*          4+1 MOVE.w           D0, D1
*          4+1 ADD.w            D1, D1
*          18+2+1 ADDQ.B        #upCount, 0(OTbase, D1.L)
*          10+2 BVC.S           1$
           [18]+3+1 MOVE.B      #hiCount, 0(OTbase, D1.L)
1$ *       8+1 Addq             #2, CSP
*          8+2 MOVE.W           (CSP), D1
*          8+2 BTst             #0, D1
*          10+2 BEQ.S           2$
*          4+1 ADD.w            D1, D1
*          18+2+1 SUBQ.B        #downCount, 0(otbase, D1.L)
*          10+2 BVC.S           2$
           [18]+3+1 MOVE.B      #hiCount, 0(otbase, D1.L)
2$ *       8+1+1 MOVE.W         D0, (CSP)
                 Next Cycle
```

1$ 32