INTCODE - An Interpretive Machine

Code for BCPL

by

M. Richards

ABSTRACT

INTCODE is a very simple machine code with an equally simple assembly language. An assembler and interpreter for it is easy to write and may be used for the initial step of bootstrapping BCPL onto a new machine.

December 1972 (revised August 1975)

The Computer Laboratory
Corn Exchange Street
Cambridge CB2 3QG

INTCODE - An Interpretive Machine Code for BCPL

INTCODE is an interpretive machine code which was designed to ease the initial bootstrapping of BCPL °1, 2, 3¢ onto a new machine. The main advantage of INTCODE is that it is compact and its assembler and interpreter are both easy to implement. The assembler and interpreter are together about 4 to 6 times smaller than a typical BCPL codegenerator from OCODE to assembly language and can be implemented in machine code in about 2 days. The INTCODE form of the entire BCPL compiler from BCPL to INTCODE takes about 1000 lines of 72 characters composed as follows:

Syntax analyser	BCPL to AE TREE	400 lines
Translation phase	AE TREE to OCODE	400 lines
Code Generator	OCODE to INTCODE	200 lines

Even though with INTCODE one loses a factor of about 10 to one in execution speed, it is still a useful tool for the initial implementation of BCPL on a new machine since it allows the first production codegenerator to be written directly in BCPL.

The INTCODE Machine

The INTCODE machine has a store consisting of equal sized locations addressed by consecutive integers. The word size is implementation dependent but should normally be at least 24 bits in order to hold all the fields of an instruction. On a 16 bit machine, one can use 16 bit and 32 bit instructions; the choice between short and long instructions being made by the INTCODE assembler.

The central registers of the machine are as follows:

A,B:	The Accumulator and Auxiliary Accumulator.
C:	The Control Register giving the location of next instruction to be executed.
D:	The Address register used to hold the effective address of an instruction.
P:	A pointer used to address the local work area and function arguments.

G: A pointer used to address the global vector.

The format of an instruction is composed of five fields
as follows:

Function Part:	This is a three bit field specifying one of the eight possible machine functions described below.
Address field:	This is a field specifying a positive integer which is the initial value of D. The address field should contain at least 14 or 15 bits.
P bit:	A single bit to specify whether P is to be added into D at the second stage of address evaluation.
G bit:	A single bit to specify whether G is to be added into D at the third stage of address evaluation.
I bit:	This is the indirection bit. If it is a one then D is replaced by the contents of the location addressed by D at the last stage of address evaluation.

The effective address is evaluated in the same way for every
instruction independent of the particular machine function specified.

The eight machine functions are as follows:

0) Load Mnemonic L

Load the effective address into the accumulator A saving
its previous value in B.

B := A; A := D

1) Store Mnemonic S

Store the accumulator A into the location addressed by D.

Location (D) := A

2) Add Mnemonic S

Add the effective address D into the accumulator A.

A := A + D

3) Jump Mnemonic J

Cause a transfer of control by setting the control register C to the effective address D.

```
C := D
```

4) Jump if True   Mnemonic T

This is a conditional transfer which sets the control register C to D if the Accumulator A is non-zero.

```
IF A^= O DO C := D
```

5) Jump if False   Mnemonic F

This is a conditional transfer which sets the control register C to D if the accumulator A is zero.

```
IF A=0 DO C := D
```

6) Call a function   Mnemonic K

Cause a recursive function call to take place. The current stack frame size is specified by D and the function entry point is given in A. The first two cells of the new stack frame are set to hold the return link information.

```
D := P + D

Location(D), Location(D+1) := P, C

P,C := D,A
```

7) Execute operation   Mnemonic X

This instruction allows auxiliary operations to be executed. The operation is specified by the value of D which should be a small integer. These operations are mainly of an arithmetic or logical nature working on the accumulators A and B, and are specified as follows:

```
X1: A := Location (A)

X2: A := -A

X3: A := NOT A

X4: This causes a return from the current function or routine;
    by convention the result fo a function is left in A.

    C := Location(P + 1)

    P := Location(P)

X5: A := B * A
```

```
X6: A := B / A

X7: A := B REM A

X8: A := B + A

X9: A := B - A

X10: A := B = A

X11: A := B ^= A

X12: A := B < A

X13: A := B >= A

X14: A := B > A

X15: A := B <= A

X16: A := B LSHIFT A     // vacated positions

X17: A := B RSHIFT A     // are filled with zeroes

X18: A := B LOGAND A

X19: A := B LOGOR A

X20: A := B NEQV A

X21: A := B EQV A

X22: FINISH
X23: Switch on the value of A using data in the location
     addressed by C, C+1 etc.

     B, D := Location(C), Location(C + 1)

     UNTIL B = 0 DO

          $( B, C := B - 1, C + 2

               IF A=Location(C) DO

                    $( D := Location(C + 1)

                         BREAK $)       $)

     C := D

X24:These are implementation dependent instructions
X25: for input/output operations and other special
     functions.  See the listing  of the INTCODE interpreter
```

in the appendix.

INTCODE Assembly Language

The assembly language for INTCODE has been designed to be compact and simple to assemble, but care has also been taken so that it can be read and modified with reasonable ease by a programmer. The text of the assembly language is composed of letters, digits, spaces, newlines and the characters '/' and dollar '$'.

Slash is used as a continuation symbol; it is skipped and the remaining characters of the line up to and including the next newline character are ignored. Its main purpose is to simplify the efficient use of cards as a medium for transferring INTCODE programs.

Dollar marks the entry point of a function or routine and is otherwise ignorable. Its sole purpose is to help the implementer find his way around compiled code.

The assembly form of an instruction consists of the mnemonic letter for the machine function, optionally followed by 'I' if indirection is specified, optionally followed by 'P' or 'G' if P or G modifications are specified, followed by the address which is either a decimal integer or an assembly parameter which appears as 'L' followed by a decimal integer. Assembly parameters are numbered in the range 1 to 500 and are used to label points in the program. A number not preceded by a letter is interpreted as a label and causes the specified assembly parameter to be set to the address of the next location to be loaded.

The mnemonics for the machine functions are L, S, A, J, T, F, K and X as described in the previous section.

Data may be assembled by a statement consisting of 'D' followed by a signed decimal integer for constant values or 'DL' followed by an assembly parameter number for pointers. Characters may be packed and assembled using character statements of the form 'C' followed by the integer code for the character. The character size and number of characters per word are machine dependent and it is left to the assembler to pack character strings appropriately. A label, instruction or data statement will cause the latest character string to be padded with zeros so that the loading pointer points to the start of a full word.

It is possible to initialise global variables during assembly, using a directive of the form 'G' followed by a global number, followed by 'L' and an assembly parameter number. Thus G36L73 will cause global 36 of the INTCODE machine to be set to the value of assembly parameter number 73.

'Z' is used to mark the end of each segment of code. Its effect is to unset all the numerical parameters.

Conclusion

The effectiveness of INTCODE lies mainly in its simplicity making it easy to understand and implement; however, it is also compact and even with a simple non-optimising code generator the compiled code is smaller than straightforward machine code for most machines by a factor of nearly two to one.

Example

The following BCPL program:

```
GLOBAL $( START:1; WRITEF:76 $)

LET START () BE $(1

LET F(N) = N=0 -> 1, N*F(N-1)

FOR I = 1 TO 10 DO WRITEF("F(%N), = %N*N", I, F(I))

FINISH $)1
```

compiles into the following INTCODE:

```
$ 1 JL4
$ 2 LO LIP2 X10 FL6 L1 SP3 JL5 6 LIP2 L1 X9 SP5 LIL3 K3 LIP2 X5 SP3 5 L/
IP3 X4 4 L1 SP2 J17 8 LI499 SP5 LIP2 SP6 LIP2 SP9 LIL3 K7 SP7 LIG76 K3 /
LIP2 A1 SP2 7 LIP2 L10 X15 TL8 X22 X22
3 DL2 499 C11 C70 C40 C37 C78 C41 C32 C61 C32 C37 C78 C10
GIL1
Z
```

References

°1¢ Richards, M. BCPL - A tool for Compiler Writing and Sysyems Programming. SJCC 1969.

°2¢ -------- BCPL Programming Manual. Computer Laboratory, Cambridge 1973.

°3¢ -------- The Portability of the BCPL Compiler. Software Practice and Experience, Vol. 1, No. 2 (1971).

Appendix - The INTCODE Assembler and Interpreter

```
// This program is an ASCII INTCODE assembler and interpreter
// for a 16 bit EBCDIC machine, hence the need for the ASCII and
// EBCDIC tables near the end.  It has been tested on the IBM 370
// (a 32 bit EBCDIC machine).

GET "LIBHDR"

GLOBAL $(
SYSPRINT:100; SOURCE:101
ETOA:102; ATOE:103
$)

MANIFEST $(
FSHIFT=13
IBIT=#10000; PBIT=#4000; GBIT=#2000; DBIT=#1000
ABITS=#777
WORDSIZE=16; BYTESIZE=8
LIG1=#012001
K2  =#140002
X22 =#160026
$)

GLOBAL $(
G:110; P:111; CH:112; CYCLECOUNT:113
LABV:120; CP:121; A:122; B:123; C:124; D:125; W:126 $)


LET ASSEMBLE() BE
$(1   LET V = VEC 500
      LET F = 0
      LABV := V

CLEAR:FOR I = 0 to 500 DO LABV:I := 0
      CP := 0

NEXT: RCH()
SW:   SWITCHON CH INTO

$(S   DEFAULT: IF CH=ENDSTREAMCH RETURN
               WRITEF("*NBAD CH %C AT P = %N*N", CH, P)
               GOTO NEXT

      CASE '0' : CASE '1' : CASE '2' : CASE '3' : CASE '4' :
      CASE '5' : CASE '6' : CASE '7' : CASE '8' : CASE '9' :
               SETLAB(RDN())
               CP := 0
               GOTO SW
      CASE '$' : CASE '*S' : CASE '*N' : GOTO NEXT
```

```
       CASE 'L' : F := 0; ENDCASE
       CASE 'S' : F := 1; ENDCASE
       CASE 'A' : F := 2; ENDCASE
       CASE 'J' : F := 3; ENDCASE
       CASE 'T' : F := 4; ENDCASE
       CASE 'F' : F := 5; ENDCASE
       CASE 'K' : F := 6; ENDCASE
       CASE 'X' : F := 7; ENDCASE

       CASE 'C' : RCH(); STC(RDN()); GOTO SW

       CASE 'D' : RCH()
                  TEST CH='L'
                    THEN $( RCH()
                            STW(0)
                            LABREF(RDN(), P-1)   $)
                    OR STW(RDN())
                  GOTO SW

       CASE 'G' : RCH()
                  A := RDN() + G
                  TEST CH='L' THEN RCH()
                        OR WRITEF("*NBAD CODE AT P = %N*N", P)
                  ]A := 0
                  LABREF(RDN(), A)
                  GOTO SW

       CASE 'Z' : FOR I = 0 TO 500 DO
                      IF LABV]I>0 DO WRITEF("L%N UNSET*N", I)
                  GOTO CLEAR $)S

       W := F<<FSHIFT
       RCH()
       IF CH='I' DO $( W := W+IBIT; RCH() $)
       IF CH='P' DO $( W := W+PBIT; RCH() $)
       IF CH='G' DO $( W := W+GBIT; RCH() $)

       TEST CH='L'

         THEN $( RCH()
                 STW(W+DBIT)
                 STW(0)
                 LABREF(RDN(), P-1) $)

         OR   $( LET A = RDN()
                 TEST (A&ABITS)=A
                   THEN STW(W+A)
                    OR $( STW(W+DBIT); STW(A) $) $)

       GOTO SW $)1

AND STW(W) BE $( ]P := W
                 P, CP := P+1, 0 $)
```

```
AND STC(C) BE $( IF CP=0 DO $( STW(0); CP := WORDSIZE $)
                 CP := CP - BYTESIZE
                 ](P-1) := ](P-1) + (C<<CP) $)

AND RCH() BE $(1 CH := RDCH()
                 UNLESS CH='/' RETURN
                 UNTIL CH='*N' DO CH := RDCH() $)1 REPEAT

AND RDN() = VALOF
    $( LET A, B = 0, FALSE
       IF CH='-' DO $( B := TRUE; RCH() $)
       WHILE '0'<=CH<='9' DO $( A := 10*A + CH - '0'; RCH() $)
       IF B DO A := -A
       RESULTIS A $)

AND SETLAB(N) BE
     $( LET K = LABV]N
        IF K<0 DO WRITEF("L%N ALREADY SET TO %N AT P = %N*N",N,-K,P)
        WHILE K>0 DO $( LET N = ]K
                        ]K := P
                        K := N $)
        LABV]N := -P $)

AND LABREF(N, A) BE
    $( LET K = LABV]N
       TEST K<0 THEN K := -K OR LABV]N := A
       ]A := ]A + K $)

AND INTERPRET() = VALOF
$(1

FETCH: CYCLECOUNT := CYCLECOUNT + 1
       W := ]C
       C := C + 1

       TEST (W&DBIT)=0
         THEN D := W&ABITS
         OR $( D := ]C; C := C + 1 $)

       IF (W & PBIT) NE 0 DO D := D + P
       IF (W & GBIT) NE 0 DO D:= D + G
       IF (W & IBIT) NE 0 DO D := ]D

       SWITCHON W>>FSHIFT INTO

   $(  ERROR:
       DEFAULT: SELECTOUTPUT(SYSPRINT)
                WRITEF("*NINTCODE ERROR AT C = %N*N", C-1)
                RESULTIS -1

       CASE 0: B := A; A := D; GOTO FETCH

       CASE 1: ]D := A; GOTO FETCH
```

```
        CASE 2: A := A + D; GOTO FETCH

        CASE 3: C := D; GOTO FETCH

        CASE 4: A := NOT A

        CASE 5: UNLESS A DO C := D; GOTO FETCH

        CASE 6: D := P + D
                D]0, D]1 := P, C
                P, C := D, A
                GOTO FETCH

        CASE 7: SWITCHON D INTO

        $(  DEFAULT: GOTO ERROR

            CASE 1:  A := ]A; GOTO FETCH
            CASE 2:  A := -A; GOTO FETCH
            CASE 3:  A := NOT A; GOTO FETCH
            CASE 4:  C := P]1
                     P := P]0
                     GOTO FETCH
            CASE 5:  A := B * A; GOTO FETCH
            CASE 6:  A := B / A; GOTO FETCH
            CASE 7:  A := B REM A; GOTO FETCH
            CASE 8:  A := B + A; GOTO FETCH
            CASE 9:  A := B - A; GOTO FETCH
            CASE 10: A := B = A; GOTO FETCH
            CASE 11: A := B NE A; GOTO FETCH
            CASE 12: A := B < A; GOTO FETCH
            CASE 13: A := B >= A; GOTO FETCH
            CASE 14: A := B > A; GOTO FETCH
            CASE 15: A := B <= A; GOTO FETCH
            CASE 16: A := B << A; GOTO FETCH
            CASE 17: A := B >> A; GOTO FETCH
            CASE 18: A := B & A; GOTO FETCH
            CASE 19: A := B LOGOR A; GOTO FETCH
            CASE 20: A := B NEQV A; GOTO FETCH
            CASE 21: A := B EQV A; GOTO FETCH
            CASE 22: RESULTIS 0  // FINISH
            CASE 23: B, D := C]0, C]1   // SWITCHON
                     UNTIL B=0 DO
                     $( B, C := B-1, C+2
                        IF A=C]0 DO
                        $( D := C]1
                           BREAK $) $)
                     C := D
                     GOTO FETCH

// CASES 24 UPWARDS ARE ONLY CALLED FROM THE FOLLOWING
// HAND WRITTEN INTCODE LIBRARY - ICLIB:

//    11 LIP2 X24 X4 G11L11 /SELECTINPUT
//    12 LIP2 X25 X4 G12L12 /SELECTOUTPUT
//    13 X26 X4      G13L13 /RDCH
//    14 LIP2 X27 X4 G14L14 /WRCH
//    42 LIP2 X28 X4 G42L42 /FINDINPUT
//    41 LIP2 X29 X4 G41L41 /FINDOUTPUT
//    30 LIP2 X30 X4 G30L30 /STOP
//    31 X31 X4 G31L31 /LEVEL
//    32 LIP3 LIP2 X32 G32L32 /LONGJUMP
//    46 X33 X4 G46L46 /ENDREAD
//    47 X34 X4 G47L47 /ENDWRITE
//    40 LIP3 LIP2 X35 G40L40 /APTOVEC
//    85 LIP3 LIP2 X36 X4 G85L85 /GETBYTE
//    86 LIP3 LIP2 X37 X4 G86L86 /PUTBYTE
//    Z


          CASE 24: SELECTINPUT(A); GOTO FETCH
          CASE 25: SELECTOUTPUT(A); GOTO FETCH
          CASE 26: A := ETOA]RDCH(); GOTO FETCH
          CASE 27: WRCH(ATOE]A); GOTO FETCH
          CASE 28: A := FINDINPUT(STRING370(A)); GOTO FETCH
          CASE 29: A := FINDOUTPUT(STRING370(A)); GOTO FETCH
          CASE 30: RESULTIS A;  // STOP(A)
          CASE 31: A := P]0; GOTO FETCH  // USED IN LEVEL()
          CASE 32: P, C := A, B;         // USED IN LONGJUMP(P,L)
                   GOTO FETCH
          CASE 33: ENDREAD(); GOTO FETCH
          CASE 34: ENDWRITE(); GOTO FETCH
          CASE 35: D := P+B+1            // USED IN APTOVEC(F, N)
                   D]0, D]1, D]2, D]3 := P]0, P]1, P, B
                   P, C := D, A
                   GOTO FETCH
          CASE 36: A := ICGETBYTE(A, B)  // GETBYTE(S, I)
                   GOTO FETCH
          CASE 37: ICPUTBYTE(A, B, P]4)  // PUTBYTE(S, I, CH)
                   GOTO FETCH
      $)  $)  $)1

AND STRINGTO370(S) = VALOF
     $( LET T = TABLE 0,0,0,0,0,0,0,0

        PUTBYTE(T, 0, ICGETBYTE(S, 0))
        FOR I = 1 TO ICGETBYTE(S, 0) DO
                  PUTBYTE(T,I,ATOE]ICGETBYTE(S,I))

        RESULTIS T  $)

AND ICGETBYTE(S, I) = VALOF
     $( LET W = S](I/2)
        IF (I&1)=0 DO W := W>>8
        RESULTIS W&255  $)

AND ICPUTBYTE(S, I, CH) BE
     $( LET P= @S](I/2)
        LET W = ]P
        TEST (I&1)=0 THEN ]P := W&#X00FF LOGOR CH<<8
                     OR   ]P := W&#XFF00 LOGOR CH    $)

LET START(PARM) BE
$(1

LET PROGVEC = VEC 20000
LET GLOBVEC = VEC 400

G, P := GLOBVEC, PROGVEC

SYSPRINT := FINDOUTPUT("SYSPRINT")
SELECTOUTPUT(SYSPRINT)

WRITES("INTCODE SYSTEM ENTERED*N")

SOURCE := FINDINPUT("INTIN")
SELECTINPUT(SOURCE)
ASSEMBLE()
SOURCE := FINDINPUT("SYSIN")
UNLESS SOURCE=0 DO SELECTINPUT(SOURCE)

WRITEF("*NPROGRAM SIZE = %N*N", P-PROGVEC)


OTOE := 1+TABLE -1,
          0,  0,  0,  0,  0,  0,  0,  0,  // ASCII TO EBCDIC
          0,  5, 21,  0, 12,  0,  0,  0,  // '*T' '*N' '*P'
          0,  0,  0,  0,  0,  0,  0,  0,
          0,  0,  0,  0,  0,  0,  0,  0,

         64, 90,127,123, 91,108, 80,125,  // '*S' ] " # $ % & '
         77, 93, 92, 78,107, 96, 75, 97,  //   (  ) * + , - . /
        240,241,242,243,244,245,246,247,  //   0  1 2 3 4 5 6 7
        248,249,122, 94, 76,126,110,111,  //   8  9 : ; < = > ?
        124,193,194,195,196,197,198,199,  //   @  A B C D E F G
        200,201,209,210,211,212,213,214,  //   H  I J K L M N O
        215,216,217,226,227,228,229,230,  //   P  Q R S T U V W
        231,232,233, 66, 98, 67,101,102,  //   X  Y Z ° ≤ ¢ µ ∂
         64,129,130,131,132,133,134,135,  //      a b c d e f g
        136,137,145,146,147,148,149,150,  //   h  i j k l m n o
        151,152,153,162,163,164,165,166,  //   p  q r s t u v w
        167,168,169, 64, 79, 64, 95,255  //    x  y z   !   ^


ETOA := 1+TABLE -1,
      0,   0,   0,   0,   0, #11,   0,   0,
      0,   0,   0, #13, #14, #15,   0,   0,
      0,   0,   0,   0,   0, #12,   0,   0,
      0,   0,   0,   0,   0,   0,   0,   0,
      0,   0,   0,   0,   0, #12,   0,   0,
      0,   0,   0,   0,   0,   0,   0,   0,
      0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,
        #40,   0,#133,#135,   0,   0,   0,   0,
          0,   0,   0, #56, #74, #50, #53,#174,
        #46,   0,   0,   0,   0,   0,   0,   0,
          0,   0, #41, #44, #52, #51, #73,#176,
        #55, #57,#134,   0,   0,#136,#137,   0,
          0,   0,   0, #54, #45,#140, #76, #77,
          0,   0,   0,   0,   0,   0,   0,   0,
          0,   0, #72, #43,#100, #47, #75, #42,
          0,#141,#142,#143,#144,#145,#146,#147,
       #150,#151,   0,   0,   0,   0,   0,   0,
          0,#152,#153,#154,#155,#156,#157,#160,
       #161,#162,   0,   0,   0,   0,   0,   0,
          0,   0,#163,#164,#165,#166,#167,#170,
       #171,#172,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,
          0,#101,#102,#103,#104,#105,#106,#107,
       #110,#111,   0,   0,   0,   0,   0,   0,
          0,#112,#113,#114,#115,#116,#117,#120,
       #121,#122,   0,   0,   0,   0,   0,   0,
          0,   0,#123,#124,#125,#126,#127,#130,
       #131,#132,   0,   0,   0,   0,   0,   0,
        #60, #61, #62, #63, #64, #65, #66, #67,
        #70, #71,   0,   0,   0,   0,   0,   0


C := TABLE L1G1, K2, X22

CYCLEOUT := 0
A := INTERPRET()

SELECTOUTPUT(SYSPRINT)
WRITEF("*N*NEXECUTION CYCLES = %N, CODE = %N*N", CYCLEOUT, A)
IF A<0 DO MAPSTORE()
FINISH  $)1
```