Inter-Office Memorandum

To	BCPL Users	Date	June 18, 1973
From	Gene McDaniel	Location	Palo Alto Coyote Hill
Subject	BCPL Machine Language Interface	Organization	PARC

XEROX

The following document purports to describe BCPL calling conventions for assembly language programmers. The following description only applies to pre-structure versions of BCPL. The term procedure will refer to both routines and procedures.

All BCPL procedure calls use static cells which point to procedure code body and a few machine language routines which are always loaded by the BCPL loader. A procedure static cell points to the first executable statement in a BCPL PROCEDURE. The code in a procedure body references these static cells through local parameters on the stack or through local pointers in the code body. At the time of a procedure call, the stack pointer is in AC2 and the value of the first two parameters (if there are any) are in ACØ and AC1.

> NOTE: In the following discussion Procedure A is the calling procedure and Procedure B is the called procedure. i.e., some place "inside A" there occurs the piece of code, "B(...)."

A calls B by doing a JSR indirect through the static cell pointing to B. At this point, all four accumulators may have valuable data. B's first action is to store AC3 (return address) in a page zero location and then to call "GETFRAME" through a page zero pointer (static). When control returns from GETFRAME, sundry bookkeeping information has been stored in the stack, all parameters, if any, have been stored on the stack, AC2 has the new stackpointer and ACØ, AC1, and AC3 have been clobbered since the initial entry.

The following details about GETFRAME are of principle interest to the curious or to people who wish to call BCPL routines from assembly language programs. If you aren't interested, skip to the section on "How To Use It".

GETFRAME stores the return address in B in a page zero cell (GETFMRETURN). Then it stores ACØ and AC1 as the first two parameters on the new stack frame, along with the old (A's) stack pointer. In traditional NOVA custom, the word following B's JSR to GETFRAME is a parameter - the size of stack frame which B wishes allocated. BCPL will compute the new stack pointer and it will abort if the stack frame is too big.

GETFRAME also places B's return address in procedure A, the value of cell 20B, and the number of arguments passed by A into the stack. The value of GETFRMRETURN plus one is assumed to be GETFRAME's return link into B. That value, whose other alias is the "CODEOWNER" (i.e., of the current stack frame) is stored in the stack. The word immediately following A's original JSR to B is also treated as a parameter - the number of arguments which A is passing to B. If that number is two or fewer, GETFRAME immediately returns to B via the "CODEOWNER" entry on the new stack (with the new stack pointer in AC2).

Procedure A places any remaining parameters in the stack in the following way: GETFRAME places the address minus one of where the next parameter should fall in the stack in cell 20B (a page zero auto incrementing cell). It moves the new stack pointer into AC1, and leaves the old stack pointer in AC2. GETFRAME now "calls" procedure A via the return address in the stack. Procedure A then computes the values of the remaining parameters and stores them onto the new stack frame. When this process is complete, Procedure A calls the routine, "RECALL" through a page zero pointer. RECALL stores AC3 as B's correct return address into Procedure A onto the stack. It also assumes AC1 still contains the value of the new stack frame pointer which it moves into AC2. RECALL then "returns" to Procedure B through the "CODEOWNER" entry on the stack.

When procedure B wishes to return to Procedure A, it places its "RESULTIS" value into AC0 and calls the BCPL routine "RETURN" through a page zero pointer.

RETURN performs the following operations:

1. It returns the old value of cell 20 which was stored in the stack.

2. It stores B's stackpointer into "SNEXT".

3. It restores the old stack pointer and, finally, it returns control to the original caller.

The BCPL Stack

NOTE: The stack pointer is offset by 200B to take better advantage of NOVA indexing hardware.

AC2-2ØØB+Ø	Old AC2
AC2-2ØØB+1	Return address to caller
AC2-2ØØB+2	Address of code owner
AC2-2ØØB+3	Stack size = 6+ num args expected + num locals
AC2-2ØØB+4	Old value, cell 2ØB
AC2-2ØØB+5	num args passed by caller
AC2-2ØØB+6	parameter 1
AC2-2ØØB+7	parameter 2

How to Use It, or, Folklore Made Simple

I. In your assembly language routines

1. Leave a pointer to the first executable statement of each assembly language routine in some known location. This will be your analogue of the BCPL procedure static call.

2. Use the code shown in the example to call setframe.

3. When done, place the result, if any, in ACØ and return to the BCPL calling routine as illustrated in the example.

4. ASM your program(s) as you normally would.

5. RLDR your programs as you normally would. Note: Do not put a starting address on your assembly language routines; this would confuse BCPL's loader.

II. Using BCPL

1. Compile your programs as always.

2. See the example for how to call the assembly language procedure.

3. Use the BCPL loader, BLDR to make the final BCPL program:
BLDR/T/R "your assembly language save file" "/I your BCPL routines"
If you use the DOS debugger with your assembly language routines, don't use the "/D" switch on BLDR as the loader (RLDR) has already placed the debugger in the assembly language save file.

If you use the DOS debugger with your assembly language routines, don't use the "/D" switch on BLDR as the loader (RLDR) has already placed the debugger in the assembly language save file.

Examples: in BCPL

Let Prog = #1Ø1ØØØ// this creates a pointer to the "Assembly Language // Static Cell" which is presumably located location 1ØØØB.

```
FOO = PROG (A,B,C,...)//call the routine
```

(asm. lang. examples to be provided.)

GMcD:tn

NOTE: Lines typed by user are underlined

ASM/L ORIGINAL.WS↵

Programs relocatable

```
        .TITL  DISK
```

RLDR ORIGINAL↵

Disk

No starting address for load module

```
NMAX  ØØ1Ø36
ZMAX  ØØØØ5Ø
CSZE
 EST
 SST
```

R
BLDR/T/R ORIGINAL/I BCPLPROG.1↵
R

Suppose your BCPL program is on the file, "BCPLPROG1". In that program, the following sequences of code will call the assembly language program described above.

```
.
.
.
LET  ASMPROG = #1Ø1ØØØ  //LOCATION 1ØØØB BETTER HAVE A POINTER TO THE SUBROUTINE
.
.
.
ASMPROG( BUFFER, DISKADDR, DISKCHANNEL, READ)  //CALL THE ASSEMBLY LANGUAGE
//PROGRAM
.
.
.
(Wasn't that easy!)
```

```
  0001  DISK
                        .TITL   DISK
                    ;
        177606          FIRST = -172    ;DEFINE INDEX OF THE  PARAMETERS ON
THE STACK
        177607          SECND = -171    ;REMEMBER THAT  "EARLIER" ENTRIES AR
E FOR BCPL'S
        177610          THIRD = -170    ;INTERNAL USE.
        177611          FORTH = -167
        177612          FIFTH = -166
        177613          SIXTH = -165
        054337          .DUSR   SAVER = STA 3,337       ;DEFINE SOME MACROS
TO CALL THE GUTS OF BCPL
        006352          .DUSR   FIXUP = JSR @352        ;"SAVER" SAVES THE R
ETURN, "FIXUP" CALLS
        006351          .DUSR   RTURN = JSR @351        ;"GETFRAME" AND "RTU
RN" DOES BCPL RETURNS
                    ;
                        .NREL
                    ;
                    ;
                    ;   PAGING(CORE, DISK, CHANNEL, R/W FLAG)
                    ;
   00000'000001'DPTR:   .+1     ;BECAUSE THIS IS THE ONLY ASSEMBLY LANGUAGE
PROGRAM, WE KNOW
   00001'054337 DSKIO:  SAVER   ;KNOW THAT "DPTR" WILL BE LOCATED AT LOC 100
0B (ABSOLUTE)
                    ;   THE EXAMPLE BCPL PROGRAM USES THIS FACT
                    ;
   00002'006352         FIXUP           ;CALL "GETFRAME". ADD 6 TO EXPECTED
NUMBER OF PARAMETERS
   00003'000012         12              ;4 PARAMETERS
   00004'050426         STA     2,STACK         ;SAVE STACK POINTER
   00005'020424         LDA     0,C8            ;LOOP COUNTER
   00006'040425         STA     0,TEMP
   00007'021206         LDA     0,FIRST,2       ;CORE ADDRESS
   00010'025207         LDA     1,SECND,2       ;DISK ADDRESS
   00011'035211         LDA     3,FORTH,2       ;R/W FLAG
   00012'031210         LDA     2,THIRD,2       ;CHANNEL NUMBER
   00013'175004         MOV     3,3,SZR         ;READ IS 0
   00014'000410         JMP     DSKWR           ;WRITE IS 1
   00015'034412         LDA     3,WREAD
   00016'054402         STA     3,FNC
   00017'006002 DISK:   .SYSTM
   00020'076077 FNC:    .RDR    CPU
   00021'000413         JMP     DERRO           ;ERROR RETURN
   00022'030410         LDA     2,STACK         ;RESTORE STACK POINTER
   00023'006351         RTURN           ;DO A BCPL RETURN. CALLING PROGRAM D
OESN'T EXPECT
                    ;       A RETURN VALUE, SO DON'T LOAD AC0.
                    ;
   00024'034404 DSKWR:  LDA     3,WRITE
   00025'054773         STA     3,FNC
   00026'000771         JMP     DISK
                    ;
   00027'076077 WREAD:  .RDR    CPU
   00030'077477 WRITE:  .WRR    CPU
   00031'000010 C8:     10
   00032'000000 STACK:  0
   00033'000000 TEMP:   0
                    ;
   00034'030776 DERRO:  LDA     2,STACK
```



```
   00035'006351         RTURN                   ;RETURN AFTER DEBUGGING
                        .END
```