INTERFACING TO BASIC09

The object code generated by the Microware C Compiler can be made callable from the BASIC09 "RUN" statement. Certain portions of a BASIC09 program written in C can have a dramatic effect on execution speed. To effectively utilize this feature, one must be familiar with both C and BASIC09 internal data representation and procedure calling protocol.

C type "int" and BASIC09 type "INTEGER" are identical; both are two byte two's complement integers. C type "char" and BASIC09 type "BYTE" and "BOOLEAN" are also identical. Keep in mind that C will sign-extend characters for comparisons yielding the range -128 to 127.

BASIC09 strings are terminated by 0xff (255). C strings are terminated by 0x00 (0). If the BASIC09 string is of maximum length, the terminator is not present. Therefore, string length as well as terminator checks must be performed on BASIC09 strings when processing them with C functions.

The floating point format used by C and BASIC09 are not directly compatible. Since both use a binary floating point format it is possible to convert BASIC09 reals to C doubles and vice-versa.

Multi-dimensional arrays are stored by BASIC09 in a different manner than C. Multi-dimensional arrays are stored by BASIC09 in a column-wise manner; C stores them row-wise. Consider the following example:

BASIC09 matrix: DIM array(5,3):INTEGER.
The elements in consecutive memory locations (read left to right, line by line) are stored as:
(1,1),(2,1),(3,1),(4,1),(5,1)
(1,2),(2,2),(3,2),(4,2),(5,2)
(1,3),(2,3),(3,3),(4,3),(5,3)

C matrix: int array[5][3];
The elements in consecutive memory locations (read left to right, line by line) are stored as:
(1,1),(1,2),(1,3)
(2,1),(2,2),(2,3)
(3,1),(3,2),(3,3)
(4,1),(4,2),(4,3)
(5,1),(5,2),(5,3)

Therefore to access BASIC09 matrix elements in C, the subscripts must be transposed. To access element array(4,2) in BASIC09 use array[2][4] in C.

The details on interfacing BASIC09 to C are best described by example. The remainder of this appendix is a mini tutorial demonstrating the process starting with simple examples and working up to more complex ones.

EXAMPLE 1 - SIMPLE INTEGER ARITHMETIC CASE

This first example illustrates a simple case. Write a C function to add an integer value to three integer variables.

```
build bt1.c
? addints(cnt,value,s1,arg1,s2,arg2,s3,arg3,s4)
? int *value,*arg1,*arg2,*arg3;
? {
?     *arg1 += *value;
?     *arg2 += *value;
?     *arg3 += *value;
? }
?
```

That's the C function. The name of the function is "addints". The name is information for C and c.link; BASIC09 will not know anything about the name. Page 9-13 of the BASIC09 Reference manual describes how BASIC09 passes parameters to machine language modules. Since BASIC09 and C pass parameters in a similar fashion, it is easy to access BASIC09 values. The first parameter on the BASIC09 stack is a two byte count of the number of following parameter pairs. Each pair consists of an address and size of value. For most C functions, the parameter count and pair size is not used. The address, however, is the useful piece of information. The address is declared in the C function to always be a "pointer to..." type. BASIC09 always passes addresses to procedures, even for constant values. The arguments cnt, s1, s2, s3, and s4 are just place holders to indicate the presence of the parameter count and argument sizes on the stack. These can be used to check validity of the passed arguments if desired.

The line "int *value,*arg1,*arg2,*arg3" declares the parameters (in this case all "pointers to int"), so the compiler will generate the correct code to access the BASIC09 values. The remaining lines increment each arg by the passed value. Notice that a simple arithmetic operation is performed here (addition), so C will not have to call a library function to do the operation.

To compile this function, the following C compiler command line is used:

```
cc2 bt1.c -rs
```

Cc2 uses the Level-Two compiler. Replace cc2 with cc1 if you are using the Level-One compiler. The -r option causes the compiler to leave bt1.r as output, ready to be linked. The -s option suppresses the call to the stack checking function. Since we will be making a module for BASIC09, cstart.r will not be used. Therefore, no initialized data, static data, or stack checking is allowed. More on this later.

The bt1.r file must now be converted to a loadable module that BASIC09 can link to by using a special linking technique as follows:

```
c.link bt1.r -b=addints -o=addints
```

This command tells the linker to read bt1.r as input. The option "-b=addints" tells the linker to make the output file a module that BASIC09 can link to and that the function "addints" is to be the entrypoint in the module. You may give many input files to c.link in this mode. It resolves references in the normal fashion. The name given to the "-b=" option indicates which of the functions is to be entered directly by the BASIC09 RUN command. The option "-o=addints" says what the name of the output file is to be, in this case "addints". This name should be the name used in the BASIC09 RUN command to call the C procedure. The name given in the "-o=" option is the name of the procedure to RUN. The "-b=" option is merely information to the linker so it can fill in the correct module entrypoint offset.

Enter the following BASIC09 program:

```
PROCEDURE btest
DIM i,j,k:INTEGER
i=1
j=132
k=-1033
RUN addints(4,i,j,k)
PRINT i,j,k
END
```

When this procedure is RUN, it should print:

```
5       136     -1029
```

indicating that our C function worked!

EXAMPLE 2 - MORE COMPLEX INTEGER ARITHMETIC CASE

The next example shows how static memory area can be used. Take the C function from the previous example and modify it to add the number of times it has been entered to the increment:

```
build bt2.c
? static int entcnt;
?
? addints(cnt,cmem,cmemsiz,value,s1,arg1,s2,arg2,s3,arg3,s4)
? char *cmem;
? int *value,*arg1,*arg2,*arg3;
? {
? #asm
?  ldy 6,s base of static area
? #endasm
?      int j = *value + entcnt++;
?
?      *arg1 += j;
?      *arg2 += j;
?      *arg3 += j;
? }
?
```

This example differs from the first in a number of ways. The line "static int entcnt" defines an integer value named entcnt global to bt2.c. The parameter cmem and the line "char *cmem" indicate a character array. The array will be used in the C function for global/static storage. C accesses non-auto and non-register variables indexed off the Y register. Cstart.r normally takes care of setting this up. Since cstart.r will not be used for this BASIC09 callable function, we have to take measures to make sure the Y register points to a valid and sufficiently large area of memory. The line "ldy 6,s" is assembly language code embedded in C source that loads the Y register with the first parameter passed by BASIC09. If the first parameter in the BASIC09 RUN statement is an array, and the "ldy 6,s" is placed IMMEDIATELY after the "{" opening the function body, the offset will always be "6,s". Note the line beginning "int j = ...". This line uses an initializer which, in this case, is allowed because j is of class "auto". No classes but "auto" and "register" can be initialized in BASIC09 callable C functions.

To compile this function, the following C compiler command line is used:

```
cc  bt2.c -rs
Where cc is cc1 or cc2.
```

Again, the -r option leaves bt2.r as output and the -s option suppresses stack checking.

Normally, the linker considers it to be an error if the "-b=" option appears and the final linked module requires a data memory

allocation. In our case here, we require a data memory allocation and we will provide the code to make sure everything is set up correctly. The "-t" linker option causes the linker to print the total data memory requirement so we can allow for it rather than complaining about it. Our linker command line is:

```
c.link bt2.r -o=addints -b=addints -t
```

The linker will respond with "BASIC09 static data size is 2 bytes". We must make sure cmem points to at least 2 bytes of memory. The memory should be zeroed to conform to C specifications.

Enter the following BASIC09 program:

```
PROCEDURE btest
DIM i,j,k,n:INTEGER
DIM cmem(10):INTEGER
FOR i=1 TO 10
     cmem(i)=0
NEXT i
FOR n=1 TO 5
     i=1
     j=132
     k=-1033
     RUN addints(cmem,4,i,j,k)
     PRINT i,j,k
NEXT n
END
```

This program is similar to the previous example. Our area for data memory is a 10 integer array (20 bytes) which is way more than the 2 bytes for this example. It is better to err on the generous side. Cmem is an integer array for convenience in initializing it to zero (per C data memory specifications). When the program is run, it calls addints 5 times with the same data values. Because addints adds the number of times it was called to the value, the i,j,k values should be 4+number of times called. When run, the program prints:

```
5       136     -1029
6       137     -1028
7       138     -1027
8       139     -1026
9       140     -1025
```

Works again!

EXAMPLE 3 - SIMPLE STRING MANIPULATION

This example shows how to access BASIC09 strings through C functions. For this example, write the C version of SUBSTR:

```
build bt3.c
? /* Find substring from BASIC09 string:
?         RUN findstr(A$,B$,fndpos)
?    returns in fndpos the position in A$ that B$ was found or
?    0 if not found.  A$ and B$ must be strings, fndpos must be
?    INTEGER.
? */
? findstr(cnt,string,strcnt,srchstr,srchcnt,result)
? char *string,*srchstr;
? int strcnt,srchcnt,*result;
? {
?     *result = finder(string,strcnt,srchstr,srchcnt);
? }
?
? static finder(str,strlen,pat,patlen)
? char *str,*pat;
? int strlen,patlen;
? {
?     int i;
?     for(i=1;strlen-- > 0 && *str!=0xff; ++i)
?         if(smatch(str++,pat,patlen))
?             return i;
? }
?
? static smatch(str,pat,patlen)
? register char *str,*pat;
? int patlen;
? {
?     while(patlen-- > 0 && *pat != 0xff)
?         if(*str++ != *pat++)
?             return 0;
?     return 1;
? }
?
```

Compile the program:

```
cc bt3.c -rs
Where cc is cc1 or cc2
```

And link it:

```
c.link bt3.r -o=findstr -b=findstr
```

The BASIC09 test program is:

```
PROCEDURE btest
DIM a,b:STRING[20]
DIM matchpos:INTEGER
LOOP
INPUT "String ",a
INPUT "Match  ",b
RUN findstr(a,b,matchpos)
PRINT "Matched at position ",matchpos
ENDLOOP
```

When the program is run, it should print the position where the matched string was found in the source string.

EXAMPLE 4 - QUICKSORT

The next example programs demonstrate how one might implement a quicksort written in C to sort some BASIC09 data.

C integer quicksort program:

```
#define swap(a,b) { int t; t=a; a=b; b=t; }

/* qsort to be called by BASIC09:
     dim d(100):INTEGER  any size INTEGER array
     run cqsort(d,100)   calling qsort.
*/

qsort(argcnt,iarray,iasize,icount,icsiz)
int  argcnt,      /* BASIC09 argument count */
     iarray[],    /* Pointer to BASIC09 integer array */
     iasize,      /*  and it's size */
     *icount,     /* Pointer to BASIC09 (sort count) */
     icsiz;       /* Size of integer */
{
     sort(iarray,0,*icount);  /* initial qsort partition */
}

/* standard quicksort algorithm from Horowitz-Sahni */
static sort(a,m,n)
register int *a,m,n;
{
     register i,j,x;

     if(m < n) {
          i = m;
          j = n + 1;
          x = a[m];
          for(;;) {
               do i += 1; while(a[i] < x);  /* left partition */
               do j -= 1; while(a[j] > x);  /* right partition */
               if(i < j)
                    swap(a[i],a[j])          /* swap */
               else break;
          }
          swap(a[m],a[j]);
          sort(a,m,j-1);                     /* sort left */
          sort(a,j+1,n);                     /* sort right */
     }
}
```

The BASIC09 program is:

```
PROCEDURE sorter
DIM i,n,d(1000):INTEGER
n=1000
i=RND(-(PI))
FOR i=1 TO n
d(i):=INT(RND(1000))
NEXT i
PRINT "Before:"
RUN prin(1,n,d)
RUN qsortb(d,n)
PRINT "After:"
RUN prin(1,n,d)
END

PROCEDURE prin
PARAM n,m,d(1000):INTEGER
DIM i:INTEGER
FOR i=n TO m
PRINT d(i); " ";
NEXT i
PRINT
END
```

C string quicksort program:

```
/* qsort to be called by BASIC09:
     dim cmemory:STRING[10] This should be at least as large as
                            the linker says the data size should
                            be.
     dim d(100):INTEGER     Any size INTEGER array.

     run cqsort(cmemory,d,100) calling qsort. Note that the pro-
                            cedure name run is the linked OS-9
                            subroutine module.  The module name
                            need not be the  name of the C func-
                            tion.
*/

int maxstr;     /* string maximum length */

static strbcmp(str1,str2)            /* basic09 string compare */
register char *str1,*str2;
{
     int maxlen;

     for (maxlen = maxstr; *str1 == *str2 ;++str1)
          if (maxlen-- > 0 || *str2++ == 0xff)
               return 0;
     return (*str1 - *str2);
}
```

```
cssort(argcnt,stor,storsiz,iarray,iasize,elemlen,elsiz,
            icount,icsiz)
int  argcnt;          /* BASIC09 argument count */
char *stor;          /* Pointer to string (C data storage) */
char iarray[];       /* Pointer to BASIC09 integer array */
int  iasize,          /*  and it's size */
     *elemlen,        /* Pointer integer value (string length) */
     elsiz;           /* Size of integer */
     *icount,         /* Pointer to integer (sort count) */
     icsiz;           /* Size of integer */
{
/* The following assembly code loads Y with the first
   arg provided by BASIC09.  This code MUST be the first code
   in the function after the declarations.  This code assumes the
   address of the data area is the first parameter in the BASIC09
   RUN command. */

#asm
 ldy 6,s get addr for C data storage
#endasm

/* Use the C library qsort function to do the sort.  Our
   own  BASIC09 string compare function will compare the strings.
*/

     qsort(iarray,*icount,maxstr=*elemlen,strbcmp);
}

/* define stuff cstart.r normally defines */
#asm
_stkcheck:
 rts dummy stack check function

 vsect
errno: rmb 2 C function system error number
_flacc: rmb 8 C library float/long accumulator
 endsect
#endasm
```

The BASIC09 calling program: (words file contains strings to sort)

```
PROCEDURE ssorter
DIM a(200):STRING[20]
DIM cmemory:STRING[20]
DIM i,n:INTEGER
DIM path:INTEGER
OPEN #path,"words":READ

n=100
FOR i=1 TO n
INPUT #path,a(i)
NEXT i
CLOSE #path
RUN prin(a,n)
RUN cssort(cmemory,a,20,n)
```

```
RUN prin(a,n)
END

PROCEDURE prin
PARAM a(100):STRING[20]; n:INTEGER
DIM i:INTEGER
FOR i=1 TO n
PRINT i; " "; a(i)
NEXT i
PRINT
END
```

The next example shows how to access BASIC09 reals from C functions:

```
flmult(cnt,cmemory,cmemsiz,realarg,realsize)
int cnt;            /* number of arguments */
char *cmemory;      /* pointer to some memory for C use */
double *realarg;    /* pointer a real */
{
#asm
 ldy 6,s get static memory address
#endasm

     double number;

     getbreal(&number,realarg);     /* get the BASIC09 real */
     number *= 2.;                  /* number times two*/
     putbreal(realarg,&number);     /* give back to BASIC09 */

}

/* getbreal(creal,breal)
     get a 5-byte real from BASIC09 format to C format */

getbreal(creal,breal)
double *creal,*breal;
{
     register char *cr,*br;   /* setup some char pointers */

     cr = creal;
     br = breal;
#asm
*  At this point U reg contains address of C double
*                  0,s contains address of BASIC09 real
 ldx 0,s get address of B real

 clra clear the C double
 clrb
 std 0,u
 std 2,u
 std 4,u
 stb 6,u
 ldd 0,x
```

```
 beq g3 BASIC09 real is zero

 ldd 1,x get hi B mantissa
 anda #$7f clear place for sign
 std 0,u put hi C mantissa
 ldd 3,x get lo B mantissa
 andb #$fe mask off sign
 std 2,u put lo C mantissa
 lda 4,x get B sign byte
 lsra shift out sign
 bcc g1
 lda 0,u get C sign byte
 ora #$80 turn on sign
 sta 0,u put C sign byte
g1 lda 0,x get B exponent
 suba #128 excess 128
 sta 7,u put C exponent
g3 clra clear carry
#endasm

}

/* putbreal(breal,creal)
     put C format double into a 5-byte real from BASIC09 */

putbreal(breal,creal)
double *breal,*creal;
{
     register char *cr,*br;   /* setup some char pointers */

     cr = creal;
     br = breal;
#asm
*  At this point U reg contains address of C double
*                  0,s contains address of BASIC09 real
 ldx 0,s get address of B real

 lda 7,u get C exponent
 bne p0 not zero?
 clra clear the BASIC09
 clrb real
 std 0,x
 std 2,x
 sta 4,x
 bra p3 and exit

p0 ldd 0,u get hi C mantissa
 ora #$80 this bit always on for normalized real
 std 1,x put hi B mantissa
 ldd 2,u get lo C mantissa
 std 3,x put lo B mantissa
 incb round mantissa
 bne p1
 inc 3,x
 bne p1
```

```
        inc 2,x
        bne p1
        inc 1,x
       p1 andb #$fe turn off sign
        stb 4,x put B sign byte
        lda 0,u get C sign byte
        lsla shift out sign
        bcc p2 bra if positive
        orb #$01 turn on sign
        stb 4,x put B sign byte
       p2 lda 7,u get C exponent
        adda #128 less 128
        sta 0,x put B exponent
       p3 clra clear carry
       #endasm
       }

       /* replace cstart.r definitions for BASIC09 */
       #asm
       _stkcheck:
       _stkchec:
        rts

        vsect
       _flacc: rmb 8
       errno: rmb 2
        endsect
       #endasm
```

BASIC09 calling program:

```
       PROCEDURE btest
       DIM a:REAL
       DIM i:INTEGER
       DIM cmemory:STRING[32]
       a=1.
       FOR i=1 TO 10
         RUN flmult(cmemory,a)
         PRINT a
       NEXT i
       END
```

The last program is an example of accessing BASIC09 matrix elements.
The C program:

```
matmult(cnt,cmemory,cmemsiz,matxaddr,matxsize,scalar,scalsize)
char *cmemory;      /* pointer to some memory for C use */
int matxaddr[5][3]; /* pointer a double dim integer array */
int *scalar;        /* pointer to integer */
{
#asm
 ldy 6,s get static memory address
#endasm

     int i,j;

     for(i=0; i<5; ++i)
          for(j=1; j<3; ++j)
               matxaddr[j][i]  *=  *scalar;  /* multiply by value
*/
}
#asm
_stkcheck:
_stkchec:
 rts

 vsect
_flacc: rmb 8
errno: rmb 2
 endsect
#endasm
```

BASIC09 calling program:

```
PROCEDURE btest
DIM im(5,3):INTEGER
DIM i,j:INTEGER
DIM cmem:STRING[32]
FOR i=1 TO 5
    FOR j=1 TO 3
        READ im(i,j)
    NEXT j
NEXT i
DATA 11,13,7,3,4,0,5,7,2,8,15,0,0,14,4
FOR i=1 TO 5
    PRINT im(i,1),im(i,2),im(i,3)
NEXT i
PRINT
RUN matmult(cmem,im,64)
FOR i=1 TO 5
    PRINT im(i,1),im(i,2),im(i,3)
NEXT i
END
```