Symbolic Data Memory Allocation for XP-32 Memory

Introduction

The simple CPFORTRAN function subprogram described in this article permits the automatic symbolic allocation of XP-32 Local Main Data to variables and arrays. Most examples of CPFORTRAN programming in applications articles and FPS documentation show memory space calculated and pre-allocated by the system designer or programmer.

Adding a simple function to the CPFORTRAN system library can greatly reduce this effort. The function shown below assigns memory segments to user-supplied symbols, or variables, while hiding hardware-dependent constraints such as variable memory configurations, and Even start addresses for arrays. Another significant advantage is gained when the size of arrays is modified at a future date: memory need not be re-allocated manually after each such change.

The function takes as input the variable ISIZE, indicating the size, in words, of the area requested. If it is successful, the function returns an integer pointer to the base address of the area, or −1 if none is available. The physical size of the memory is a constant assigned in a DATA statement in the function.

```
      INTEGER FUNCTION ALLOCLMD (ISIZE)
C= = = Allocate a segment of Local Main Data
      COMMON /LMDPTR/ LMDNEXT, LMDMAX
      DATA LMDNEXT /0/
      DATA LMDMAX /65535/   !or 16383 for 16K version
      IF(ISIZE .EQ. IOR(ISIZE,1)) THEN   !check for Even request
         IS=ISIZE + 1
      ELSE
         IS=ISIZE
      ENDIF
      IF(LMDNEXT + IS .GT. LMDMAX) THEN
         ALLOCLMD = −1
      ELSE
         ALLOCLMD = LMDNEXT
         LMDNEXT  = LMDNEXT + IS
      ENDIF
      RETURN
      END
      SUBROUTINE FREELMD
C= = = Free all allocation of Local Main Data by resetting pointer
      COMMON /LMDPTR/ LMDNEXT, LMDMAX
      LMDNEXT = 0
      RETURN
      END
```

Figure 1. CPFORTRAN version to allocate and deallocate Local Main Data

The simplicity of this method becomes apparent when it is used in an example application. The following code fragment is for a small XP32 application written in CPFORTRAN, using the allocation function for the memory mapping.

```
                    PROGRAM XPTEST
C = = = XPTEST routine, in CPFORTRAN
                    PARAMETER (ISZA=100), (ISZB=100), (ISZC=100)
                    PARAMETER (ISZD=100), (ISZE=100), (ISZF=100)
                    COMMON /DATA/ A, B, C, D, E, F
                    REAL A(ISZA), B(ISZB), C(ISZC), D(ISZD), E(ISZE), F(ISZF)
                    INTEGER IA, IB, IC, ID, IE, IF, J, IXPNUM
C
C                       allocate LMD space for 6 vectors
C
                    IA = ALLOCLMD (ISZA)
                    IB = ALLOCLMD (ISZB)
                    IC = ALLOCLMD (ISZC)
                    ID = ALLOCLMD (ISZD)
                    IE = ALLOCLMD (ISZE)
                    IF = ALLOCLMD (ISZF)
C
                    CALL XPDMAR  (20, IA, 1, A, 1, 2, 1)      !load input array x
                    CALL XPDMAR  (20, IB, 1, B, 1, 2, 1)      !load input array y
                    CALL XPDMAR  (20, IC, 1, C, 1, 2, 1)      !load input array z
                    CALL XPISNC
                    CALL ZVNRM3  (IA, IB, IC, ID, IE, IF, 20) !call xpal routine
                    CALL XPISNC
                    CALL XPDMAR  (20, ID, 1, D, 1, 2, -1)     !unload output array x
                    CALL XPDMAR  (20, IE, 1, E, 1, 2, -1)     !unload output array y
                    CALL XPDMAR  (20, IF, 1, F, 1, 2, -1)     !unload output array z
                    CALL XPISNC
C
                    IXPNUM = 1
                    J = XPSEL (IXPNUM)          !select an XP to assign
                    CALL XPRUN                  !dispatch the channel list
                    J = XPWAIT (IXPNUM)         !wait for the XP to finish
                    CALL FREELMD                !release memory allocation
                    STOP
                    END
```

Figure 2. Example CPFORTRAN application program

The code shown in Figure 1 assumes that all XPs will have the same map; among many possible features to add would be separate pointers for each XP. Extensions to this method are quite straightforward. For instance, a bit map of the allocations can be included, so that selective areas may be freed and re-allocated later. Collecting previously used buffers, or setting aside pools to be used by requests of different length, can help avoid fragmentation. However, in most signal processing applications, space is allocated at the beginning and remains static for the duration of the run. It is also possible to write a MAXL version of this routine, using a more complicated calling list, which allocates System Common Memory in a similar manner.

Conclusion

The overhead for the function described here is minimal. The advantages in ease of use for development and maintenance of programs outweigh this overhead, which can usually be ignored.

Ian Curington is a Signal Image Processing Specialist with Floating Point Systems UK, Ltd., where his responsibilities include performance analysis, graphics, and image processing for northern Europe. He formerly held positions in product marketing and performance analysis at Floating Point Systems World Headquarters in Beaverton, Oregon. Before joining FPS in 1979, he worked in real-time control and micro-computer systems design. Mr. Curington has a BS in Mathematics from Lewis and Clark College. His professional interests include computer graphics, image synthesis, and motion dynamics. He is a member of ACM-SIGGRAPH, EUROGRAPHICS, and the Computer Arts Society. *For additional information, contact Doug Atterbury, Senior Systems Support Engineer, (800) 547-1445, extension 1680.*