

# Expanding the HPC Address Space

National Semiconductor  
Application Note 497  
Joe Cocovich  
August 1988



## INTRODUCTION

The maximum address range of the HPC family of 16-bit High Performance microControllers is 64k bytes using the external address/data bus to interface with external memory. This application note describes a method to increase the amount of memory in a system to 544k bytes utilizing bank switching techniques. Block diagrams are presented to aid in circuit design. Software examples are given for memory and bank management.

## HPC ADDRESSING

Program memory addressing is accomplished by the 16-bit Program Counter on a byte basis (instructions are always fetched a byte at a time). Memory can be addressed as words or bytes directly by instructions or indirectly through the B, X and SP registers. Words are always addressed on even-byte boundaries. The HPC uses memory-mapped organization to support registers, I/O and on-chip peripheral functions.

The external address/data bus of the HPC is 16 bits wide. This means the maximum address that the bus can hold is FFFF for a maximum address range of 64K bytes (65,536). Keep in mind, this uses the external address/data bus (A0:A15 for Address/Data and B10, 11, 12, 15) for Control.

## BANK SWITCHING

If more than 64k of addressing is needed in the HPC system, the following method of increasing memory space can be used. Divide the total address range into two halves (32k bytes each). One half of this address range will be the MAIN memory address space. The MAIN memory address space will contain logical addresses (those addresses which the Program Counter can generate) in the range 8000 to FFFF and is accessed when A15 is a '1'. This includes the Interrupt vectors and the Reset vector memory locations. The other half of the address range will be the BANK memory address space. The BANK memory address space will contain logical addresses in the range 0000 to 7FFF and is accessed when A15 is a '0'. This includes the on-chip I/O, registers, and RAM at locations 0000 to 01FF.

Now, four additional address lines are created using Port B pins (B8, B9, B13, B14). This prevents the use of the four timer synchronous outputs TS0-TS3 which are the alternate functions for these pins. The BANK memory is now addressed using A0:A14, B8, B9, B13, B14 and is accessed when A15 is a '0'. The BANK memory address space is now expanded to 512k bytes broken down into 16 individually selectable banks of 32k bytes each selected by these four bits of Port B.

A look at Table 1 and Figure 1 quickly tells you that only one bank in the BANK memory space can share the logical address range 0000:7FFF at any one time. Therefore, programs running in the BANK memory address space can only directly access data and programs in the MAIN memory address space or in its own bank (selected by B8, B9, B13, B14). On chip resources, which include RAM, I/O, and registers are mapped into logical addresses 0000 to 01FF. These logical addresses are in the BANK memory address space, but, since these addresses are considered to be al-

ways on-chip by the HPC, it never looks at the external address/data bus and will not read external memory in this range. Therefore, the first 256 bytes in each bank of memory in the BANK memory space will not be accessible by the HPC, but this address range (on chip resources) is directly accessible by any bank of memory in the BANK memory address space. This is why Figure 1 shows a total available memory of 536.5k.

The interrupt vectors are mapped into logical addresses FFF0 to FFFF which are in the MAIN memory address space. Interrupts are handled properly if they occur while executing a program out of one of the banks of memory in BANK memory space, since the interrupt vector locations have A15 set to '1' which will allow access to the MAIN memory space. However, these interrupt vectors must either point to a routine in the MAIN memory address space which performs the interrupt service or point to code that selects the appropriate bank of memory in the BANK memory space and go there if the interrupt service routine is located there.

The stack must be located so that it can be directly accessible from anywhere in memory. It can be placed in the MAIN memory space or in the on-chip RAM. Programs and data storage that must be shared and directly accessed by all memory banks in the BANK memory space should also reside in the MAIN memory space.

## HPC OPERATING MODES

The HPC must be configured to run in one of its Expanded modes of operation by setting the EA bit in the PSW to be able to address the BANK memory range of 0000 to 7FFF. This memory expansion addressing scheme will work if the HPC is configured in either the Normal Expanded mode (EXM pin tied low) or ROMless Expanded mode (EXM pin tied high). The Normal mode differs from the ROMless mode only by the fact that the HPC will access the on-chip ROM for addresses in the range of E000 to FFFF (in the case of the HPC16083) and will access the external MAIN memory for addresses in the range of 8000 to DFFF.

The external data bus size is determined once, at reset, by sampling the state of HBE (B12). If HBE is high when sampled, the HPC enters 8-bit mode. In 8-bit mode, only pins A0-A7 are used to transfer data and pins A8-A15 continue to hold the most-significant eight bits of the address. So, only the lower eight bits of the address need to be latched externally (Figure 2). If HBE is low when sampled, the HPC enters 16-bit mode. In 16-bit mode, all 16 pins of Port A are used to transfer data as well as addresses. Two octal latches are then required externally to hold each address as it is issued by the HPC. The signal ALE from the HPC clocks the latches (Figure 3).

Keep in mind that if the external memory is configured as 8-bit memory, then the program stack must be in internal on-chip RAM because it has to be accessible as 16-bit words. If the external memory is configured as 16-bit memory then the stack can be in external RAM but must be in the MAIN memory address space to be directly accessible by all banks.

## PROGRAMMING CONVENTIONS

A convention must be followed for maintaining linkages between the programs and data running in the MAIN memory space and the programs and data running in the BANK memory space. For the following discussion, the MAIN memory space will be referred to as just another bank of memory.

### MAIN bank reserved portion

A portion of the MAIN memory bank should be reserved for Jump instructions to subroutines in the MAIN memory bank that need to be called by programs running in any selected bank in the BANK memory space. These Jump instructions serve as entry points for programs and subroutines. Typically, common functions that are required by programs running in several banks would be put in the MAIN memory bank. These could include: interrupt service routines, I/O drivers, and data handling and conversion routines. This portion also contains address pointers to tables of data in the MAIN memory bank that also are required by programs running in any selected bank in the BANK memory space. See Listing 1 for an example.

### BANK memory reserved portion

A portion of each bank in the BANK memory space should be reserved for Jump instructions to subroutines in that bank that need to be called by programs running in the MAIN memory bank. These Jump instructions serve as entry points for programs and subroutines. For example, each bank in the BANK memory space could contain routines that perform unique but related functions. One bank could be reserved for math routines; another bank could perform message handling; and yet another could contain diagnostic routines. All of these functions could be scheduled and executed from some sort of Supervisor running in the MAIN memory bank performing the linkages to all these routines thru the entry points. This reserved portion of each bank also contains address pointers to tables of data in that bank that also are required by programs running in the MAIN memory bank. In the case of a bank running message handling routines, address pointers could be inserted to point to buffers that programs running in MAIN memory need to access. See Listing 2 for an example.

### Linkage areas

These reserved portions of each memory bank (MAIN space or BANK space) must be fixed and known to each other memory bank that requires access to programs and data in that bank. Therefore, one other requirement in each bank is a set of labels that are assigned the values of the pointer locations to subroutines and tables in the bank of interest (see Listings 3 and 4).

One last requirement in the MAIN memory bank, if it is to perform bank to bank moves and for general housekeeping, is to reserve two byte locations to be used to keep track of the bank currently selected (high byte value on Port B) being used in the transfer of data (see Listing 5).

From the MAIN memory bank, the user can access all memory in the system. He can call subroutines in any bank in the BANK memory space and read/write data to the entire memory. From any bank in the BANK memory space, the user can call subroutines in the MAIN memory bank and read/write data to the MAIN memory bank in addition to his own local bank.

The basic procedure used to call a program in the BANK memory space from the MAIN memory bank is merely to set the proper value on the Port B select lines and execute a Jump to SubRoutine through a pointer in the selected bank:

## Interrupts

Regardless of where the interrupt service routine actually resides, an image of the bank selected must be retained by the service routine to allow it to return to the appropriate bank when complete. If the interrupt service routine is in the MAIN memory bank, the linkage is handled in the normal fashion where the interrupt vector points to the service routine. The interrupt service can reside in the BANK memory space and takes a little extra overhead for the linkage.

To call a program in the MAIN memory bank from the BANK memory space, merely execute a Jump to SubRoutine through a pointer in the MAIN memory bank:

```
JSRL CMPBLNK ;see Listing 1 and 4
```

## EXAMPLE SOFTWARE

Now that a convention has been established for communicating between the MAIN memory space and the BANK memory space, let's take a look at some sample code that can be used to move data between these memory spaces. In order to make the selection of bank memory efficient, it is important to keep in mind that the four bits of the high byte of Port B that are used to select a bank of memory in the BANK memory space can be written to directly since the other 4 bits of this byte of Port B are used for memory control outputs (the external control bus) and are not affected by a write to the high byte of Port B.

### Bank to Bank data transfer by MAIN

Listing 6 shows the setup required to initialize the linkage area in order to perform a transfer of data from one bank to another bank in the BANK memory space by a program running in the MAIN memory space. This involves setting up the RAM locations that are used to 'select' the source bank and the destination bank, select the source bank to determine the starting address of the area to move, select the destination bank to determine the starting address of the area to move data into, then finally calling the subroutine in MAIN memory that performs the move. After the setup portion, the subroutine that performs the transfer is presented. This code assumes that the external memory is configured in 16-bit mode.

### Bank to MAIN data transfer by Bank

Listing 7 presents a similar example for moving blocks of data from a bank in BANK memory to MAIN memory by a program running in that bank. This code also assumes that the external memory is configured in 16-bit mode.

### External 8-bit mode

If the external memory is configured in 8-bit mode, the setup portion changes because the initialization of the RAM address pointers SSTART, DSTART and DEND requires building word address pointers from word pointers in the external reserved areas of each bank. In 8-bit mode, this requires two 8-bit transfers compared to one 16-bit transfer in 16-bit mode (see Listing 8). Once these address pointers have been built, however, the subroutine that actually performs the move does not have to change because 1) word transfers are allowed between On-chip RAM and registers regardless of the mode and 2) the subroutine performs byte moves. To improve speed in the 16-bit mode, this subroutine can be modified to perform 16-bit moves. However, keep in mind that this will impose the restriction on the address pointers in the linkage areas of requiring that addresses be on word boundaries. Listing 9 presents a similar example for moving blocks of data from a bank in BANK memory to MAIN memory by a program running in that bank.

## PROGRAM DEVELOPMENT

The MOLE monitor software can support the development of HPC programs in multiple banks of memory. It provides the means of qualifying a trigger condition, as set in Trace or Breakpoint functions, with the memory bank number. The BANK command will allow a trigger only when executing in the memory bank of interest. The MOLE supports a total of 16 memory banks which are normally selected by 4 bits of Port B as described earlier. See the HPC Personality Board User's Manual for further detail on this command.

## CONCLUSION

What has been presented is a method to expand the memory space of the HPC to 544k. Although this method utilized four bits of Port B to accomplish the extra addressing, theoretically, the remaining 8 bits could have been used if not required for other purposes. This could mean a maximum addressability for the HPC of greater than 128 Megabytes. However, the MOLE will only support the fixed definition of four extra address lines. Clever utilization of existing resources can enable you to get the most out of hardware and software limited only by one's imagination.

TABLE I. Logical Addresses vs Physical Memory Locations

Logical Address	Bank #	Hi Byte Port B	Physical Address
0000:7FFF	0	00	00000:07FFF
0000:7FFF	1	01	08000:0FFFF
0000:7FFF	2	02	10000:17FFF
0000:7FFF	3	03	18000:1FFFF
0000:7FFF	4	20	20000:27FFF
0000:7FFF	5	21	28000:2FFFF
0000:7FFF	6	22	30000:37FFF
0000:7FFF	7	23	38000:3FFFF
0000:7FFF	8	40	40000:47FFF
0000:7FFF	9	41	48000:4FFFF
0000:7FFF	A	42	50000:57FFF
0000:7FFF	B	43	58000:5FFFF
0000:7FFF	C	60	60000:67FFF
0000:7FFF	D	61	68000:6FFFF
0000:7FFF	E	62	70000:77FFF
0000:7FFF	F	63	78000:7FFFF
8000:FFFF	—	—	08000:0FFFF (MAIN)

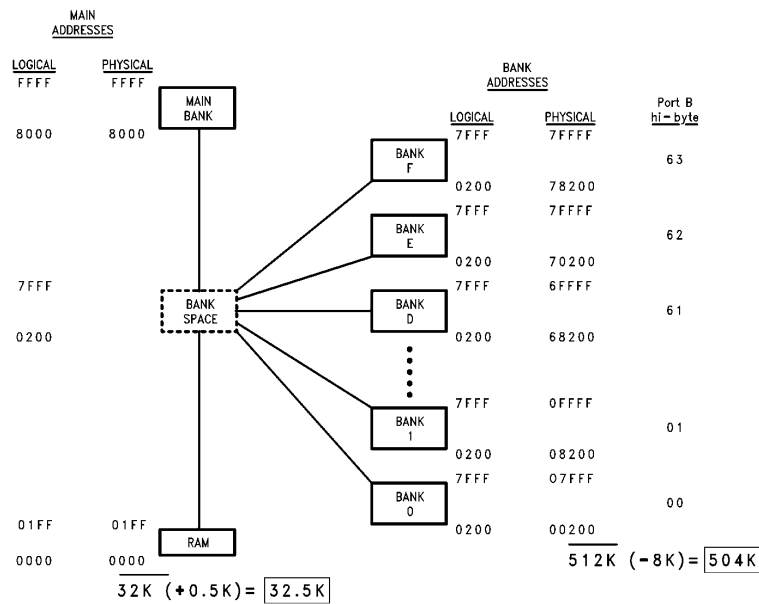


FIGURE 1. How BANK Memory is Mapped into the HPC Address Space

TL/DD/9342-1

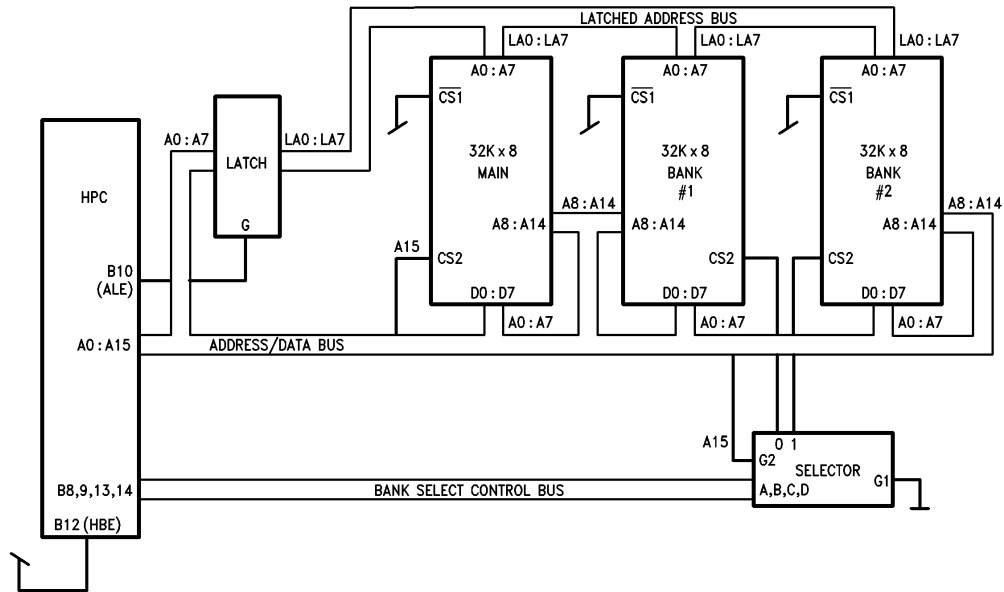


FIGURE 2. HPC in 8-Bit Mode

TL/DD/9342-2

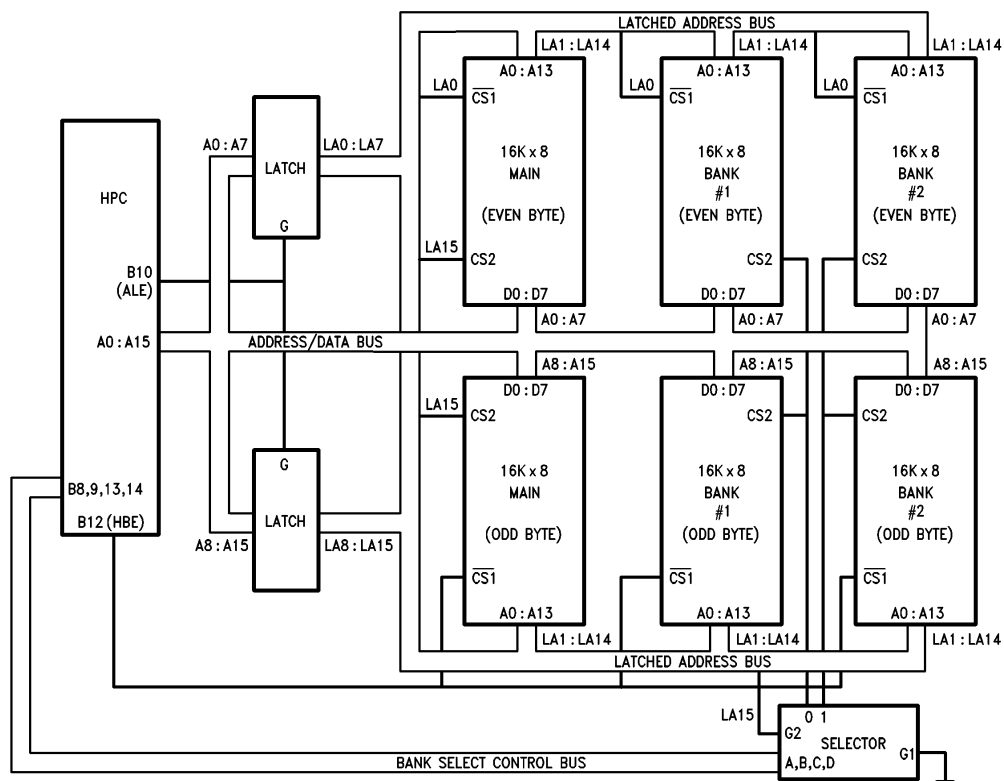


FIGURE 3. HPC in 16-Bit Mode

TL/DD/9342-3

```

        . = 08000      ;set PC counter to 8000
;This code resides in the MAIN memory bank
;
;   The following address pointers are inserted to allow
;   programs running in BANK memory to find these
;   locations. They represent the starting and ending
;   location for code in MAIN memory.
;
        .WORD INIT    ;addr pointer to first location in bank
        .WORD PROGEND  ;addr pointer to last location in bank
;
;   The following Jump instructions are inserted to allow
;   programs running in BANK memory to call these
;   subroutines. They represent subroutines that compare
;   blocks of memory in MAIN memory space with blocks of
;   memory in BANK memory space or compare blocks of memory
;   in BANK memory for zeros.
;
        JMPL CMFPM      ;entry for compare blocks (MAIN-BANK)
        JMPL CMPBFB     ;entry for compare BANK cleared

```

**LISTING 1. MAIN Bank Reserved Portion**

```

        . = 0200      ;set PC counter to 200
;This code resides in any bank in BANK memory
;
;   The following address pointers are inserted to allow
;   programs running in MAIN memory to find these
;   locations. They represent the ending location for code
;   in this bank of BANK memory.
;
        .WORD PROGEND  ;addr pointer to last loc in this bank
;
;   The following Jump instructions are inserted to allow
;   programs running in MAIN memory to call these
;   subroutines. They represent subroutines that compare
;   blocks of memory in MAIN memory space with blocks of
;   memory in this bank, diagnostic routines, and interrupt service routine.
;
        JMPL CMPMB     ;entry for comp blocks (MAIN-this bank)
        JMPL BTEST     ;entry for this bank's diag routines
        JMPL BINTS     ;entry for this bank's interrupt service routine

```

**LISTING 2. Typical Bank Reserved Portion**

```

;This code resides in the MAIN memory bank
;
;   linkages to Bank 0
;
BOSTART = 0200      ;addr of pointer to first avail loc
;
CMPMB0 = 0202      ;addr of JMPL to routine that compares
;                  move results
BOTEST = 0205      ;addr of JMPL to test routines
;
;   linkages to Bank 1
;
B1START = 0200      ;addr of pointer to first avail loc
;
CMPMB1 = 0202      ;addr of JMPL to routine that compares
;                  move results
B1TEST = 0205      ;addr of JMPL to test routines
;
;   linkages to Bank 2
;
B2START = 0200      ;addr of pointer to first avail loc
;
CMPMB2 = 0202      ;addr of JMPL to routine that compares
;                  move results
;      = 0205      ;addr of JMPL to test routines
;
B2INTS = 0208      ;addr of JMPL to interrupt service routine

```

**LISTING 3. MAIN Memory Bank Linkage Area**

```

;This code resides in any bank in BANK memory
;
;   linkages to MAIN memory
;
MSTART = 08000      ;addr of pointer to first avail loc
MEND = 08002        ;addr of pointer to last avail loc
;
CMPM = 08004        ;addr of JMPL to routine that compares
;                  move results
CMPBLNK = 08007     ;addr of JMPL to routine that compares
;                  if a block in selected BANK is zero

```

**LISTING 4. Typical Bank Linkage Area**

```

;This code resides in the MAIN memory bank
;
; The following locations are used for bank to bank moves
; and compares
    BANKS = 01C0      ;source bank byte value
    BANKD = 01C1      ;destination bank byte value
;
    BANK0 = 0          ;Port B high byte value to select bank 0
    BANK1 = 1          ;
    BANK2 = 2          ;
    BANK3 = 3          ;
    BANK4 = 020        ;
    BANK5 = 021        ;
    BANK6 = 022        ;
    BANK7 = 023        ;
    BANK8 = 040        ;
    BANK9 = 041        ;
    BANKA = 042        ;
    BANKB = 043        ;
    BANKC = 060        ;
    BANKD = 061        ;
    BANKE = 062        ;
    BANKF = 063        ;
;
; Main Memory Bank is logical and physical address range
; 8000:FFFF. Switched Memory Banks are logical addresses
; in the range 0000:7FFF combined with the
; Port B(14,13,9,8) bits to create physical addresses in
; the range 00000:7FFFF
;

```

#### LISTING 5. BANK Memory Management

```

    LD M(0E3),BANK1;set bank select lines to select bank 1
    JSRL B1TEST    ;see Listing 2 and 3
    .
    .
    .
INT35:
LD    BANKS,M(0E3) ;save bank interrupted from
LD    M(0E3),BANK2 ;set bank select lines to select bank 2
JSRL  B2INTS      ;see listing 2 and 3
    .
    .
    .
LD    M(0E3),BANKS ;restore bank interrupted from
RETI
    .
    .
    .
.IPT  2,INT35     ;set interrupt vector

```

```

;This code resides in the MAIN memory bank
;
    LD M(BANKS),BANK0      ;prepare to move data from Bank 0
    LD M(BANKD),BANK1      ;to Bank 1
    LD M(0E3),BANK0        ;select Bank 0
    LD W(SSTART),W(BOSTART) ;set starting address in source bank
    LD M(0E3),BANK1        ;select Bank 1
    LD W(DSTART),W(B1START) ;set starting address in destination bank
    LD W(DEND),W(B1START)   ;set ending address in destination bank
    ADD W(DEND),1023        ;to 1K greater than starting address
    JSRL MOVBB             ;do it
    .
    .
    .
    .
    .
;
; This subroutine moves data from bank memory to bank memory
; where the source bank is defined by the contents of the byte
; at RAM location BANKS and the destination bank is defined by
; the contents of the byte at RAM location BANKD. In addition,
; the following locations must be set up before calling:
;
; SSTART → RAM location containing source bank start address
; DSTART → RAM location containing destination bank start address
; DEND → RAM location containing destination bank end address
;
MOVBB:
    LD B,W(DSTART)         ;B ← starting address (destination)
    LD K,W(DEND)           ;K ← ending address (destination)
    LD X,W(SSTART)         ;X ← starting address (source)
LOOPBB:
    LD M(0E3),M(BANKS)     ;select source BANK
    LD A,M(X+)             ;byte at source into A
                           ;increment source pointer
    LD M(0E3),M(BANKD)     ;select destination BANK
    XS A,M(B+)             ;A into byte at destination, bump pntr
    JP LOOPBB             ;back for more if B less than K
    RET

```

**LISTING 6. Move Data by MAIN from BANK to BANK (16-Bit Mode)**



```

;This code resides in any bank in BANK memory
;
    LD W(SSTART),TABLE1    ;starting address of table in this memory
    LD W(DSTART),W(MSTART) ;starting address in main memory
    LD W(DEEND),TABLE1+1023 ;ending address in main memory
    JSRL MOVE              ;do it
        .
        .
        .
        .
        .
;
; This subroutine moves data from this bank to main memory
;
; SSTART → RAM location containing source memory start address
; DSTART → RAM location containing destination memory start addr
; DEEND → RAM location containing destination memory end address
;
MOVE:
    LD B,W(DSTART)        ;B ← starting address (destination)
    LD K,W(DEEND)         ;K ← ending address (destination)
    LD X,W(SSTART)        ;X ← starting address (source)
LOOPBM:
    LD A,M(X+)            ;byte at source into A
                          ;increment source pointer
    XS A,M(B+)            ;A into byte at destination, bump pntr
    JP LOOPBM             ;back for more if B less than K
    RET

```

**LISTING 7. Move Data by BANK from BANK to MAIN (16-Bit Mode)**

```

;This code resides in the MAIN memory bank
;
LD M(BANKS),BANK0      ;prepare to move data from Bank 0
LD M(BANKD),BANK1      ;to Bank 1
LD M(0E3),BANK0        ;select Bank 0
LD M(SSTART),M(BOSTART) ;set starting address in source bank
LD M(SSTART+1),M(BOSTART+1)
LD M(0E3),BANK1        ;select Bank 1
LD M(DSTART),M(B1START) ;set starting address in destination bank
LD M(DSTART+1),M(B1START+1)
LD M(DEND),M(B1START)   ;set ending address in destination bank
LD M(DEND+1),M(B1START+1)
ADD M(DEND),L(1023)      ;to 1K greater than starting address
ADC M(DEND+1),H(1023)
JSRL MOVBB              ;do it
    .
    .
    .
    .
    .
;
; This subroutine moves data from bank memory to bank memory
; where the source bank is defined by the contents of the byte
; at RAM location BANKS and the destination bank is defined by
; the contents of the byte at RAM location BANKD. In addition,
; the following locations must be set up before calling:
;
; SSTART → RAM location containing source bank start address
; DSTART → RAM location containing destination bank start address
; DEND → RAM location containing destination bank end address
;
MOVBB:
LD B,W(DSTART)          ;B ← starting address (destination)
LD K,W(DEND)            ;K ← ending address (destination)
LD X,W(SSTART)          ;X ← starting address (source)
LOOPBB:
LD M(0E3),M(BANKS)      ;select source BANK
LD A,M(X+)              ;byte at source into A
                        ;increment source pointer
LD M(0E3),M(BANKD)      ;select destination BANK
XS A,M(B+)              ;A into byte at destination, bump pntr
JP LOOPBB               ;back for more if B less than K
RET

```

**LISTING 8. Move Data by MAIN from BANK to BANK (8-Bit Mode)**

```

;This code resides in any bank in BANK memory
;
LD M(SSTART),L(TABLE1) ;starting address of table in this memory
LD M(SSTART+1),H(TABLE1)
LD M(DSTART),M(MSTART) ;starting address in main memory
LD M(DSTART+1),M(MSTART+1)
LD M(DEND),M(MSTART) ;set ending address in main memory
LD M(DEND+1),M(MSTART+1)
ADD M(DEND),L(1023) ;to 1K greater than starting address
ADC M(DEND+1),H(1023)
JSRL MOVE ;do it
.
.
.
.
.
;
; This subroutine moves data from this bank to main memory
;
; SSTART → RAM location containing source memory start address
; DSTART → RAM location containing destination memory start addr
; DDEND → RAM location containing destination memory end address
;
MOVE:
LD B,W(DSTART) ;B ← starting address (destination)
LD K,W(DEND) ;K ← ending address (destination)
LD X,W(SSTART) ;X ← starting address (source)
LOOPBM:
LD A,M(X+) ;byte at source into A
;increment source pointer
XS A,M(B+) ;A into byte at destination, bump pntr
JP LOOPBM ;back for more if B less than K
RET

```

**LISTING 9. Move Data by BANK from BANK to MAIN (8-Bit Mode)**

The code listed in the App Note is available on Dial-A-Helper.

Dial-A-Helper is a service provided by the Microcontroller Applications Group. The Dial-A-Helper system provides access to an automated information storage and retrieval system that may be accessed over standard dial-up telephone lines 24 hours a day. The system capabilities include a MESSAGE SECTION (electronic mail) for communicating to and from the Microcontroller Applications Group and a FILE SECTION mode that can be used to search out and retrieve application data about NSC Microcontrollers. The minimum system requirement is a dumb terminal, 300 or 1200 baud modem, and a telephone.

With a communications package and a PC, the code detailed in this App Note can be downloaded from the FILE SECTION to disk for later use. The Dial-A-Helper telephone lines are:


Modem (408) 739-1162  
Voice (408) 721-5582

**For Additional Information, Please Contact Factory**

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

 <b>National Semiconductor Corporation</b> 1111 West Bardin Road Arlington, TX 76017 Tel: 1(800) 272-9959 Fax: 1(800) 737-7018	<b>National Semiconductor Europe</b> Fax: (+49) 0-180-530 85 86 Email: cnjwge@tevm2.nsc.com Deutsch Tel: (+49) 0-180-530 85 85 English Tel: (+49) 0-180-532 78 32 Français Tel: (+49) 0-180-532 93 58 Italiano Tel: (+49) 0-180-534 16 80	<b>National Semiconductor Hong Kong Ltd.</b> 19th Floor, Straight Block, Ocean Centre, 5 Canton Rd. Tsimshatsui, Kowloon Hong Kong Tel: (852) 2737-1600 Fax: (852) 2736-9960	<b>National Semiconductor Japan Ltd.</b> Tel: 81-043-299-2309 Fax: 81-043-299-2408
---	---	--	--

National does not assume any responsibility for use of any circuitry described, no circuit patent licenses are implied and National reserves the right at any time without notice to change said circuitry and specifications.