SPHERE NEWSLETTER

VOLUME V NUMBERS 2 & 3 February 1981

EDITORS: Roger J. Spott
Jeffrey Brownstein

SOFTWARE

HARDWARE

EDITOR'S MAILBAG

PLEASE SEND MATERIAL FOR THE NEXT ISSUE TO: DR. JEFF BROWNSTEIN
2 TOR ROAD
WAPPINGERS, N.Y.
12590

Fast string-comparison routine serves the M6800

Thomas A Pickering
Pickering Radio Co Inc, Portsmouth, RI

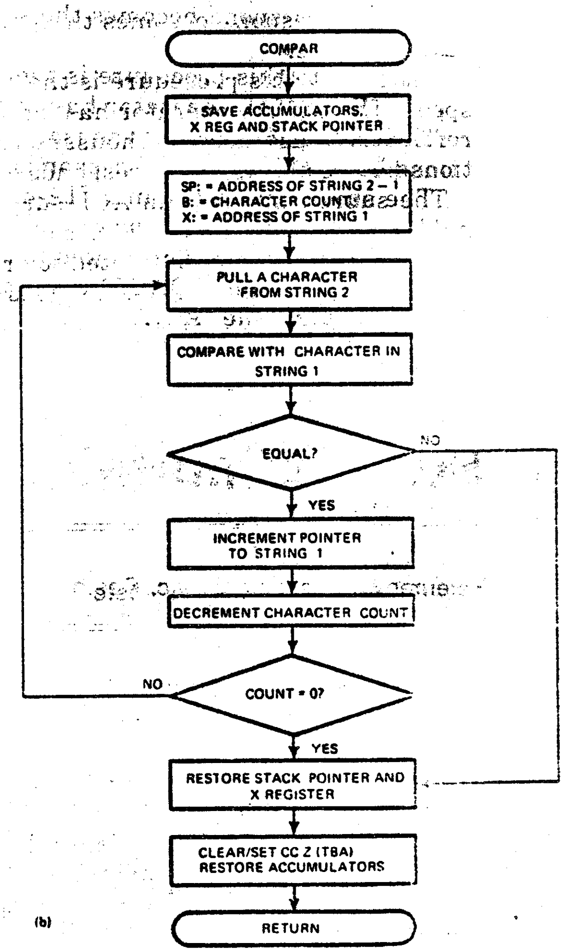
The character-string subroutine in EDN Software Note #1 (Jan 5, 1978, pg 31) certainly proves useful. In our consulting work with the M6800, however, we use the subroutine shown here, which runs almost four times faster. This speed does exact a penalty: Because the stack pointer acts as a second index register, no interrupts are permitted.

The key to the new COMPAR is the TBA (transfer B to A) instruction. Because the B register behaves like a counter, it will contain zero in the event of a successful string compare; otherwise it will be nonzero. The TBA sets the Z flag either up (good compare) or down so the calling program can branch accordingly.

Execution time for this subroutine is 23 machine cycles/character, compared with the 88 cycles/byte required by the original COMPAR subroutine. When you're testing a full 256-byte string, this difference can prove important. □

```
0000 0000     N         NAM     COMPAR
                   *
                   *       STRING COMPARE SUBROUTINE
                   *
                   * ENTER WITH (X) = A PARA LIST OF 5 BYTES:
                   *                   A (STRING1)
                   *                   A (STRING2)
                   *                   CHAR COUNT
                   *
                   * ON RETURN IF CC Z IS SET THERE IS A MATCH
                   *
                   * NOTE: DISABLE INTERRUPTS WHILE THIS ROUTINE RUNS
                   *
0000 36                   PSH A           SAVE ACCUMULATORS
0001 37                   PSH B
0002 BF 0022 R            STS SPSAV       SAVE STACK POINTER
0005 FF 0024 R            STX XSAV        SAVE X-REGISTER
0008 AE 02                LDS 2,X         SP := A(STRING2)
000A 34                   DES             ADJUST SP
000B E6 04                LDA B 4,X       LOAD CHAR COUNT
000D EE 00                LDX 0,X         X := A(STRING1)
                   *
000F 32           CMP1    PUL A           GET A BYTE FROM STRING2
0010 A1 00                CMP A 0,X       COMPARE WITH BYTE IN STRING1
0012 26 04                BNE NOMCH       NO MATCH
                   *
0014 08                   INX             POINT TO NEXT CHAR IN STRING1
0015 5A                   DEC B           DECREMENT COUNT
0016 26 F7                BNE CMP1        DO ANOTHER COMPARISON
                   *
0018 BE 0022 R    NOMCH   LDS SPSAV       RESTORE SP
001B FE 0024 R            LDX XSAV        RESTORE X-REG
001E 17                   TBA             SET/CLEAR CC Z
001F 33                   PUL B           RESTORE ACCUMULATORS
0020 32                   PUL A
0021 39                   RTS             ALL DONE
                   *
0022 0002         SPSAV   RMB 2           STACK POINTER TEMP SAVE
0024 0002         XSAV    RMB 2           X-REGISTER TEMP SAVE
                   *
                          END
```

(a)



(b)

Comparison subroutine (a) uses the stack pointer as a second index register to speed execution. A TBA instruction sets/clears condition code Z to allow conditional branching after the subroutine returns to the calling program **(b)**.

Subroutine returns a random byte

Robert D Grappel
Hemenway Associates Inc, Boston, MA

Dozens of algorithms exist that produce pseudorandom numbers for games or more serious applications. These algorithms, however, furnish a finite numerical sequence, and the sequence's repetition might be objectionable. The algorithms also require a "seed" value, which must be varied if differing sequences must be generated.

In contrast, the subroutine presented here uses the reaction time of a human operator at a terminal as the source of randomness: The operator is prompted to press a key; the time required to respond becomes the source of the random value.

The key to this procedure is the computer's speed: Even if the operator has lightning-fast reflexes, the μP executes thousands of instructions before the operator responds.

The subroutine uses an ACIA-driven terminal on a Motorola 6800 processor system. On entry, the operator is prompted to press a key; a monitor routine (PDATA) outputs the character string. The ACIA is then reset and reinitialized.

```
0000 0000     N         NAM RANDOM
                    *
                    * SUBROUTINE USES REACTION-TIME TO GENERATE RANDOM BYTE
                    * USE ACIA-DRIVEN TERMINAL   RETURNS IN "B" ACCUMULATOR
                    * USES MONITOR ROUTINE "PDATA" TO PRINT PROMPT STRING
                    *
0000 8008           ACIA    EQU $8008          ACIA ADDRESS OF TERMINAL DEVICE
0000 E07E           PDATA   EQU $E07E          PROMPT-OUTPUT ROUTINE ADDRESS
                    *
0000 CE 001A R      GETRND  LDX #PROMPT        ISSUE USER PROMPT
0003 BD E07E                JSR PDATA
0006 CE 8008                LDX #ACIA          POINT TO ACIA
0009 86 03                  LDA A #$3
000B A7 00                  STA A 0,X          RESET ACIA
000D 86 15                  LDA A #$15
000F A7 00                  STA A 0,X          INITIALIZE PIA
0011 A6 00          LOOP    LDA A 0,X          CHECK PIA STATUS ON INPUT
0013 5C                     INC B              INCREMENT COUNTER
0014 47                     ASR A
0015 24 FA                  BCC LOOP           LOOP UNTIL KEY PRESSED
                    *
0017 A6 01                  LDA A 1,X          CLEAR CHARACTER
0019 39                     RTS
                    *
001A 48             PROMPT  FCC 'HIT ANY KEY'
0025 0D0A                   FDB $0D0A
0027 0000                   FDB $0000
0029 04                     FCB 4
                    *
                            END
```

The computer then loops while waiting for the key depression. When the ACIA status indicates an input, the character is read (clearing the ACIA), and the program returns the number of times it looped while waiting for the key depression. This number is effectively random and can assume any value between 0 and 256_{10}. **EDN**

Subroutine return performs a computed jump

Robert D Grappel
Hemenway Associates Inc, Boston, MA

One problem that arises frequently in programming the 6800 μP involves making a jump to an address which results from a computation. This jump is especially difficult if the index register is used to pass information. You can circumvent this problem with a self-modifying indexed-jump instruction, but this programming trickery makes understanding and debugging the code very difficult. A much cleaner and more efficient method uses a subroutine-return (RTS) instruction. If you initialize the stack properly, such an RTS becomes an excellent computed jump.

```
LDA     A     JADDR       ; load desired jump address
LDB     B     JADDR+1     ;
PSH     B                 ; push low byte of address
PSH     A                 ; push high byte of address
RTS                       ; "jump" to address
```

These 6800 instructions *let you jump to a computed address.*

The RTS is essentially a jump to an address which lies on the top of the system stack. Normally, a JSR (jump to subroutine) or BSR (branch to subroutine) instruction places a return address on the stack. However, you can stack addresses directly using the PSH (push) instruction.

Suppose that the A and B accumulators contain the address of the jump destination: A holds the high byte and B the low byte. (This address can result from calculations or other manipulations.) If the accumulators are pushed onto the stack, B followed by A, the effect is the same as that of a subroutine call. When an RTS is executed, the program unstacks the just-pushed address and jumps there. Thus, the program jumps to the address contained in A and B. **EDN**

4

Implement a FIFO stack with software

Robert Grappel
Hemenway Associates, Boston, MA

A first-in, first-out stack finds application wherever asynchronous processes must pass data. Hardware FIFO's are available, but if high speed is not required, you can readily program a FIFO stack. The technique is termed circular buffering because it uses an area of, RAM in a circular fashion. The system stores data in the buffer and retrieves it sequentially until it reaches the end of the RAM area. The next element used is the first location in the buffer. Two pointers mark the data in the buffer: The head-pointer indicates the first available location, while the tail-pointer indicates the oldest data entry. As the system stores data, the head-pointer moves forward and loops around to the top of the buffer; the buffer is full when this pointer wraps around to meet the tail-pointer. Similarly, when the system extracts data from the buffer, the tail-pointer moves; the buffer is empty when it meets the head-pointer. **Fig 1** shows the initialization, **Fig 2** the pointer move, **Fig 3** the push-data and **Fig 4** the pull-data subroutines, coded for the M6800. ▭

```
0000 0000     N           NAM FIFO
                   *
0000 000A          BUFFER RMB 10           ARBITRARY LENGTH CIRCULAR BUFFER
000A 000A     R    BUFEND EQU *            END OF BUFFER AREA
000A 0002          HDPNT  RMB 2            HEAD-POINTER
000C 0002          TLPNT  RMB 2            TAIL-POINTER
                   *
                   *
000E CE 0000  R    INIT   LDX #BUFFER      BEGIN INITIALIZATION
0011 FF 000A  R           STX HDPNT        INIT HEAD-POINTER
0014 FF 000C  R           STX TLPNT        INIT TAIL-POINTER
                          .
                          .
                          .
                          .
```

Fig 1—Set up the buffer, head-pointer and tail-pointer before using the FIFO.

```
0017 08            MVEPTR INX              MOVE POINTER IN X-REG
0018 8C 000A R            CPX #BUFEND      BEYOND BUFFER?
001B 26 03                BNE DONE         NO, RETURN
                   *
001D CE 0000 R            LDX #BUFFER      YES, RESET POINTER
                   *
0020 39            DONE   RTS
```

Fig 2—This subroutine moves a pointer to the next location in the buffer. The system increments the pointer unless such a move would bring the pointer beyond the buffer. In that case it sets the pointer to the top of the buffer.

```
0021 FE 000A R PUSH    LDX HDPNT       GET HEAD-POINTER
0024 A7 00             STA A 0,X       STORE BYTE
0026 BD 0017 R         JSR MVEPTR      MOVE POINTER
0029 BC 000C R         CPX TLPNT       POINTERS MEET?
002C 26 02             BNE NOCAR       NO
               *
002E 0D                SEC             YES, SET CARRY BIT
002F 39                RTS
               *
0030 FF 000A R NOCAR   STX HDPNT       NEW HEAD-POINTER
0033 0C                CLC             CLEAR CARRY
0034 39                RTS
```

Fig 3—Use this coding to place the contents of the A register in the buffer. If there is no room for the store, the system sets the carry bit before the return; otherwise it clears the carry bit.

```
0035 FE 000C R PULL    LDX TLPNT       GET TAIL-POINTER
0038 BC 000A R         CPX HDPNT       BUFFER EMPTY?
003B 27 0A             BEQ EMPTY       YES
               *
003D A6 00             LDA A 0,X       NO, GET BYTE
003F BD 0017 R         JSR MVEPTR      MOVE POINTER
0042 FF 000C R         STX TLPNT       NEW TAIL-POINTER
0045 0C                CLC             CLEAR CARRY BIT
0046 39                RTS
               *
0047 0D        EMPTY   SEC             SET CARRY
0048 39                RTS
```

Fig 4—This subroutine gets a byte from the buffer and returns it to the A register. If the buffer is empty, the subroutine sets the carry bit before the return; otherwise it clears the carry bit.

Share your experiences

Have you developed hardware that expands development-system capabilities? How about software (even games you've programmed for a specific system)? Hobby applications such as ham radio or robot-construction projects make interesting reading, too, and we solicit your inputs.

5

Hash coding

You can search a hash-coded table at speeds virtually independent of the number of items in that table. Here's how.

Jack Hemenway and **Edward Teja,**
Associate Editors

Along with the convenience of storing data in a computer's memory comes the problem of retrieving that data when you need it. The actual process of reading data from memory locations is no problem; rather, the trouble lies in organizing the storage process so that symbols are:

- Stored in unique locations
- Stored only once
- Stored quickly
- Retrieved quickly.

Meeting these needs requires making some design decisions. You could store data sequentially, for example, but retrieving a particular item in that case requires knowing the item's relative position in the series; if you know only the stored symbol itself, you must search for it linearly through the list—a time-consuming process. Alternatively, you could store the items in a table and index them.

What type of index would make sense in that case? Consider for a moment what an entry in an index is—a reference to another data item. Rather than generate this reference in a manner unrelated to the symbol itself, why not perform some operation on the stored symbol to produce its index entry? Better yet, why not make this operation actually produce the address of the symbol in the table? This process is termed hashing.

Finding the key

In hashing (or hash coding), the *key* is the portion of the stored symbol used to generate the needed address. It should be as small as possible (for convenience and simplicity), yet large enough to ensure that each symbol, when hashed, produces as unique an address as is practicable. (If two symbols produce the same address, the result is termed a *collision.)*

Every symbol consists of characters. To store the ith symbol in a collection, you must create a record (R_i). This record must be long enough to store the entire symbol; therefore, R_i can be divided into R_{i1}, R_{i2}, R_{i3} and so on, corresponding to the characters in the symbol: S_{i1}, S_{i2}, S_{i3}... The usual procedure for storing a symbol collection provides enough table-entry space to accommodate the longest symbol; shorter symbol entries are then padded, typically with blanks or nulls.

The algorithm that provides a symbol's *home address* in this process is termed the hashing function. If the first two characters of each symbol (S_{i1} and S_{i2}) constitute the key, for example, the hashing algorithm (H) is some function of those elements. The home address of S_i is thus defined by $H(S_{i1}, S_{i2})$.

Many types of algorithms can effectively hash-code symbols. Standard versions employ one of the following methods:

- Division—Divides the key by an integer n and uses the remainder to produce the home address.
- Random—Uses a pseudorandom number generator with the symbol's key as its seed. The first random number produced becomes the symbol's home address, after it's normalized.
- Midsquare—Multiplies the key by itself and masks out all but a middle k-bit field in the product; this field becomes the home address.
- Radix—Treats the key as a string of octal digits and converts them to base 10. A middle k-bit field of this number becomes the home address.
- Algebraic coding—Treats the key as a polynomial, divides it by a constant polynomial and uses the remainder to form the home address.
- Folding—Picks several k-bit fields from the n-bit key and adds them to produce the home address.
- Digit analysis—Performs a skewness test on each digit of the key, selecting the k best (least skewed) digits and deleting them, leaving the home address.

Building an assembler's symbol table

EDN faced the problem of generating a symbol table when writing a cross assembler (designated XA6809) to generate code for the 6809 µP on a 6800. In this case, the symbol table must contain the labels used in a program being assembled. The assembler creates a record (R_i) corresponding to each label. Each time it encounters a label, it must find the corresponding record; if no such record exists, it must create one.

The location in which a label's record is stored is determined by the hashing function—sometimes also termed a mapping function or randomizing technique. Because it is hash coded, the symbol table must have a defined size; in XA6809, the assembler itself calculates

```
                         * TABLE MANIPULATION ROUTINES FOR TABLES
                         * SYMTAB AND MNTAB
                         *
                         * STORAGE LOCATIONS USED BY THE ROUTINES
                         *
 0AC3 0002               PSTNG1 RMB   2          ADDRESS OF MNEMONIC
 0AC5 0002               PSTNG2 RMB   2          ADDRESS IN THE TABLE
 0AC7 0001               PCOUNT RMB   1          LENGTH OF MNEMONIC
 0AC8 0002               TBADD  RMB   2          TABLE POINTER
 0ACA 0006               HSMBL  RMB   6          SYMBOL TEMP LOC
 0AD0 0002               HKEYA  RMB   2          HASHED CODE
 0AD2 0002               HKEYB  RMB   2          TEMP LOC FOR HASHED CODE
 0AD4 0002               HSAV1  RMB   2          TEMP LOC FOR PTR
 0AD6 0002               HSAV2  RMB   2          TEMP LOC FOR PTR
                         *
                         *
                         * STORE A SYMBOL IN SYMTAB
                         * ON ENTRY DESCRA CONTAINS ADDRESS OF
                         * THE SYMBOL,AND DESCRC CONTAINS THE LENGTH
                         * A STANDARD HASH CODED METHOD IS USED
                         *
                         *
 0AD8 BD 0BAA R STOSYM JSR   HASH       GET HASHED KEY
 0ADB FF 048A R        STX   SYMPTR     SAVE
 0ADE BD 2092 R        JSR   XREF
 0AE1 FE 048A R        LDX,  SYMPTR
                         *
                         * SEE IF LOC(HKEYA) IS EMPTY)
                         *
 0AE4 A6 00     SYMA   LDA A 0,X        GET FIRST CHAR
 0AE6 81 20            CMP A #$20       BLANK ?
 0AE8 26 2F            BNE   SYMB       NO
                         *
                         * STORE SYMBOL IN SYMTAB
                         *
 0AEA FF 0AD6 R        STX   HSAV2      SAVE TABLE PTR
 0AED CE 0ACA R        LDX   #HSMBL     POINT TO HSMBL
 0AF0 FF 0AD4 R        STX   HSAV1      SAVE
 0AF3 C6 06            LDA B #6         LOAD SYMBOL LENGTH
                         *
                         * DO TRANSFER
                         *
 0AF5 FE 0AD4 R SYM1   LDX   HSAV1      POINT TO HSYMBL
 0AF8 A6 00            LDA A 0,X        GET CHAR
 0AFA 08               INX
 0AFB FF 0AD4 R        STX   HSAV1      POINT TO NEXT CHAR
 0AFE FE 0AD6 R        LDX   HSAV2      POINT TO TABLE ENTRY
 0B01 A7 00            STA A 0,X        STORE CHAR IN SYMTAB
 0B03 08               INX
 0B04 FF 0AD6 R        STX   HSAV2      POINT TO NEXT POSITION
 0B07 5A               DEC B            ALL DONE ?
 0B08 26 EB            BNE   SYM1       NO
                         *
                         * STORE LC, AND SET INFO BYTE
                         *
 0B0A B6 04AB R        LDA A LC         GET LC
 0B0D A7 00            STA A 0,X        STORE
 0B0F B6 04AC R        LDA A LC+1       GET LS BYTE OF LC
 0B12 A7 01            STA A 1,X        STORE
 0B14 86 40            LDA A #$40       INFO BYTE =RELOC,DEFINED
 0B16 A7 02            STA A 2,X
 0B18 39               RTS              RETURN
                         *
                         * COMPARE HSMBL WITH ENTRY IN SYMTAB
                         *
 0B19 BD 0B6F R SYMB   JSR   SYMCMP     COMPARE
 0B1C 26 10            BNE   SYMC       NO MATCH
                         *
                         * ERROR, SYMBOL ALREADY IN TABLE
                         *
 0B1E FE 04BA R        LDX   SYMPTR     GET ADDRESS OF ENTRY
 0B21 86 80            LDA A #$80
 0B23 AA 08            ORA A 8,X        SET REDEFINED BIT
 0B25 A7 08            STA A 8,X
 0B27 CE 0206          LDX   #$0206     LOAD ERROR#
 0B2A BD 10DE R SYMB1  JSR   PRINTE     PRINT IT
 0B2D 39               RTS              RETURN
                         *
                         * FIND ANOTHER SLOT IN SYMTAB FOR SYMBOL
                         *
 0B2E BD 0B84 R SYMC   JSR   SYMMOD     GET A[NEXT SLOT]
 0B31 BC 0AD0 R        CPX   HKEYA      CHECKED ALREADY ?
 0B34 27 02            BEQ   *+4        YES,TABLE IS FULL
                         *
 0B36 20 AC            BRA   SYMA       TRY AGAIN
                         *
 0B38 CE 0221          LDX   #$0221     LOAD ERROR#
 0B3B 20 ED            BRA   SYMB1      PRINT IT & RETURN
                         * ROUTINE TO COMPARE SYMBOL WITH ENTRY
                         *
 0B6F FF 0AC3 R SYMCMP STX   PSTNG1     SAVE PTR TO ENTRY
 0B72 86 06            LDA A #6
 0B74 B7 0AC7 R        STA A PCOUNT     PCOUNT:=L(SYMBOL)
 0B77 CE 0ACA R        LDX   #HSMBL
 0B7A FF 0AC5 R        STX   PSTNG2     POINT TO HSMBL
 0B7D CE 0AC3 R        LDX   #PSTNG1    POINT TO PARMS
 0B80 BD 095E R        JSR   COMPAR     COMPARE
 0B83 39               RTS
                         *
                         *
                         * FIND NEXT SLOT IN SYMTAB
                         * SYMPTR:=SYMPTR+9 (MODULO NSYM)
                         *
 0B84 FF 04BA R SYMMOD LDX   SYMPTR     GET A[CURRENT SLOT]
 0B87 08               INX              SYMPTR:=SYMPTR+9
 0B88 08               INX
 0B89 08               INX
 0B8A 08               INX
 0B8B 08               INX
 0B8C 08               INX
 0B8D 08               INX
 0B8E 08               INX
 0B8F 08               INX
                         *
                         * BEYOND SYMTAB ?
                         *
 0B90 BC 04A4 R        CPX   SYMEND
 0B93 26 03            BNE   *+5        NO
 0B95 FE 04A0 R        LDX   SYMTAB     POINT TO FIRST ENTRY
 0B98 FF 04BA R        STX   SYMPTR     SAVE PTR TO ENTRY
 0B9B 39               RTS
                         * DELETE LAST SYMBOL ENTERED
                         *
 0B9C FE 04BA R DELSYM LDX   SYMPTR
 0B9F 86 20            LDA A #$20       LOAD BLANK
 0BA1 C6 09            LDA B #9         LOAD ENTRY LENGTH
                         *
 0BA3 A7 00     DEL1   STA A 0,X        BLANK BYTE
 0BA5 08               INX              POINT TO NEXT BYTE
 0BA6 5A               DEC B            ALL DONE ?
 0BA7 26 FA            BNE   DEL1       NO
                         *
 0BA9 39               RTS              YES, RETURN
                         * HASH SYMBOL TO PRODUCE A KEY
                         *
                         *
 0BAA CE 2020   HASH   LDX   #$2020     BLANK HSMBL
 0BAD FF 0ACA R        STX   HSMBL
 0BB0 FF 0ACC R        STX   HSMBL+2
 0BB3 FF 0ACE R        STX   HSMBL+4
                         *
                         * MOVE SYMBOL TO HSMBL
                         *
 0BB6 CE 0ACA R        LDX   #HSMBL     POINT TO HSMBL
 0BB9 FF 0AD6 R        STX   HSAV2      SAVE
 0BBC FE 04B3 R        LDX   DESCRA     POINT TO SYMBOL
 0BBF FF 0AD4 R        STX   HSAV1      SAVE
 0BC2 F6 04B5 R        LDA B DESCRC     GET L(SYMBOL)
                         *
 0BC5 FE 0AD4 R HASH1  LDX   HSAV1      POINT TO SYMBOL
 0BC8 A6 00            LDA A 0,X        GET CHAR
 0BCA 08               INX
 0BCB FF 0AD4 R        STX   HSAV1      POINT TO NEXT CHAR
 0BCE FE 0AD6 R        LDX   HSAV2      POINT TO HSYMBL
 0BD1 A7 00            STA A 0,X        STORE CHAR
 0BD3 08               INX
 0BD4 FF 0AD6 R        STX   HSAV2      POINT TO NEXT CHAR
 0BD7 5A               DEC B            ALL DONE?
 0BD8 26 EB            BNE   HASH1      NO
                         *
                         * FOLD OVER HSMBL CREATING KEYA
                         *
 0BDA FE 0ACA R        LDX   HSMBL      HKEYA:=HSMBL(2)
 0BDD FF 0AD0 R        STX   HKEYA
 0BE0 FE 0ACC R        LDX   HSMBL+2
 0BE3 FF 0AD2 R        STX   HKEYB
 0BE6 CE 0AD0 R        LDX   #HKEYA
 0BE9 BD 0EF5 R        JSR   ADD16          +HSMBL+2(2)
 0BEC FE 0ACE R        LDX   HSMBL+4
 0BEF FF 0AD2 R        STX   HKEYB
 0BF2 CE 0AD0 R        LDX   #HKEYA
 0BF5 BD 0EF5 R        JSR   ADD16          +HSMBL+4(2)
                         *
                         * HKEYA:=REMAINDER OF HKEYA/NSYM
                         *
 0BF8 B6 0AD0 R        LDA A HKEYA      LOAD VALUES:
 0BFB F6 0AD1 R        LDA B HKEYA+1
 0BFE FE 04A2 R        LDX   NSYM
 0C01 FF 0AD2 R        STX   HKEYB
 0C04 CE 0AD2 R        LDX   #HKEYB     POINT TO NSYM
 0C07 BD 0EAA R        JSR   DIV16
 0C0A FF 0AD0 R        STX   HKEYA      SAVE REMAINDER
                         *
                         * HKEYA:=HKEYA*9
                         *
 0C0D 4F               CLR A
 0C0E C6 09            LDA B #9
 0C10 CE 0AD0 R        LDX   #HKEYA
 0C13 BD 0E86 R        JSR   MPY16
 0C16 B7 0AD0 R        STA A HKEYA
 0C19 F7 0AD1 R        STA B HKEYA+1
                         *
                         * ADD IN BASE ADDRESS OF SYMTAB
                         *
 0C1C FE 04A0 R        LDX   SYMTAB
 0C1F FF 0AD2 R        STX   HKEYB
 0C22 CE 0AD0 R        LDX   #HKEYA
 0C25 BD 0EF5 R        JSR   ADD16
 0C28 FE 0AD0 R        LDX   HKEYA
 0C2B 39               RTS
```

Fig 1—Hashing a symbol *provides a memory address for storing that symbol—so long as there isn't another symbol already stored at that address.*

this size during its first pass over the source code—the table occupies all available memory after the requisite macro tables have been set up.

The entire space assigned to the symbol table (SYMTAB) is filled with blanks (20_H) during initialization. Each table entry occupies nine bytes of table space:

SSSSSS	First six characters of the symbol, padded with blanks as necessary
XX	2-byte address of the symbol
F	One byte of bit flags, formatted as: Bit 7—Redefined symbol Bit 6—Relocatable symbol Bit 5—Macro name Bit 4—Common symbol Bit 3—External symbol

The hashing function produces a symbol's home address

Bit 2—Entry-point name
Bit 1—Unused
Bit 0—Unused.

Putting symbols in memory

Fig 1 shows the software used in the storing operation. The assembler uses a subroutine named STOSYM (store symbol) to put a symbol into the table. A pointer indicates the beginning of the string to be stored, and a variable named DESCRC contains that string's length. The subroutine then calls the hash routine (HASH) to produce an address from the string.

HASH moves the symbol's first six characters into a variable location—these characters form the key. (Because the assembler's syntax rules limit label lengths to a maximum of six characters, the entire symbol is the key, affording the greatest probability of each symbol's hashing to a unique address and thus avoiding collision.) If the symbol doesn't have six characters, HASH pads it with blanks to fill out the entry. The routine then uses the folding method to hash the key (Fig 2): It adds the symbol's first two bytes to its second two and adds that sum to the third byte pair, ignoring overflows. It divides this value by the number of entries possible in the table; the remainder of this division is then multiplied by nine (the size of each entry). The result is added to the symbol table's base address to produce the home address; this home address is then returned to STOSYM in the index register.

STOSYM checks the home address to ensure that it contains only blanks; if it does, STOSYM stores the symbol there, along with its value and flags. If the location is occupied, however, the symbol in question might have already been stored there. To test for this condition, a compare routine (COMPAR) compares the stored symbol with the one to be stored, indicating the result in the processor's condition codes.

If the stored symbol is not the same as the one to be stored, the two symbols are in collision. To handle this collision, another routine (SYMMOD) provides a new address—specifically, it moves the home address of the symbol to be stored nine bytes further through the table. If this new location is empty, the entry goes there; if it is occupied, COMPAR again compares the symbol stored in the location with the symbol to be stored, and if they are not the same, SYMMOD moves the home address nine more bytes through the table.

If SYMMOD reaches the end of the table before finding an unoccupied location, the home address wraps around to the starting location and an error message results. If the compare routine ever indicates that the stored symbol matches the one to be stored, a different error message is produced—any symbol can only be stored once, because program labels must be unique.

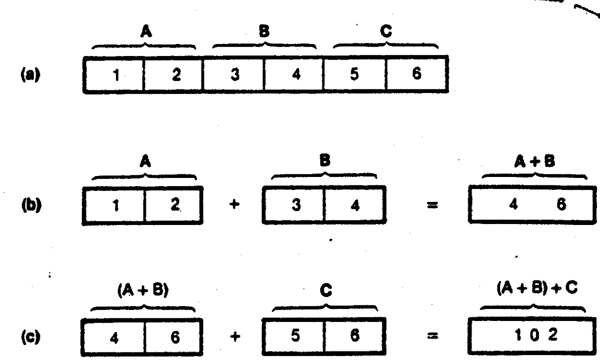


Fig 2—The folding method *of generating a symbol's home address treats the symbol's 2-character key as three 16-bit binary numbers* **(a)**, *hashed by adding the first two* **(b)** *and then adding the resulting sum to the third* **(c)**.

```
                    * LOOK UP SYMBOL  IN SYMTAB
                    * ON ENTRY DESCRA CONTAINS ADDRESS OF SYMBOL
                    * AND DESCRC CONTAINS THE LENGTH OF THE
                    * SYMBOL
                    * ON RETURN:
                    *  B=VALUE OF INFO BYTE
                    *  B=FF SYMBOL NOT FOUND
                    *  X=VALUE OF SYMBOL
                    *
                    *
0B3D BD 0BAA R LKPSYM JSR   HASH      GET KEY
0B40 FF 04BA R        STX   SYMPTR    SAVE
0B43 7D 04AD R        TST   PASS      PASS1?
0B46 27 06            BEQ   LKPSM1    YES
                    *
0B48 BD 2092 R        JSR   XREF
0B4B FE 04BA R        LDX   SYMPTR
                    *
0B4E A6 00     LKPSM1 LDA A 0,X
0B50 81 20            CMP A #$20      BLANK?
0B52 26 03            BNE   LKPSM3    NO
                    *
                    * ENTRY NOT IN SYMTAB
                    *
0B54 C6 FF     LKPSM2 LDA B #$FF      LOAD RC
0B56 39               RTS             RETURN
                    *
                    * COMPARE SYMBOL WITH ENTRY IN SYMTAB
                    *
0B57 BD 0B6F R LKPSM3 JSR   SYMCMP    COMPARE
0B5A 26 08            BNE   LKPSM4    NO MATCH
                    *
                    * FOUND, EXTRACT INFO, AND VALUE
                    *
0B5C FE 04BA R        LDX   SYMPTR    POINT TO ENTRY
0B5F E6 08            LDA B 8,X       GET INFO BYTE
0B61 EE 06            LDX   6,X       GET VALUE
0B63 39               RTS
                    *
                    * PROBE AGAIN FOR SYMBOL IN SYMTAB
                    *
0B64 BD 0B84 R LKPSM4 JSR   SYMMOD    GET NEXT KEY
0B67 BC 0AD0 R        CPX   HKEYA     ALREADY CHECKED?
0B6A 26 E2            BNE   LKPSM1    NO, TRY AGAIN
0B6C C6 FF            LDA B #$FF.     SET RC
0B6E 39               RTS
```

Fig 3—This routine looks up a symbol *in much the same way that the routine in Fig 1 stores one. Here, though, a match between the sought symbol and the stored one doesn't produce an error message.*

What goes in must come out

To reverse the hash-coding process, a routine named LKPSYM (Fig 3) looks up a symbol in the table. This routine is called with the searched-for string's address in the pointer DESCRA and its length in DESCRC. LKPSYM operates like STOSYM, except that it expects COMPAR to find a match; it indicates such a match with a status code in accumulator B. When the symbol is found, the routine puts its address in SYMPTR (symbol pointer), and the routine XREF forms a cross-reference output block for the symbol.

EDN

Multitask μP executive routine uses only six instructions

Here's a simple six-instruction subroutine that lets your 6800 microprocessor control several external processes simultaneously. To use it, organize your software as follows:

1. Set up a process-control block as shown for each process to be controlled.
2. Write a program for each process as if no other programs are running in the same microprocessor.
3. Insert JSR SPND instructions into each program at convenient points to allow other programs to run.

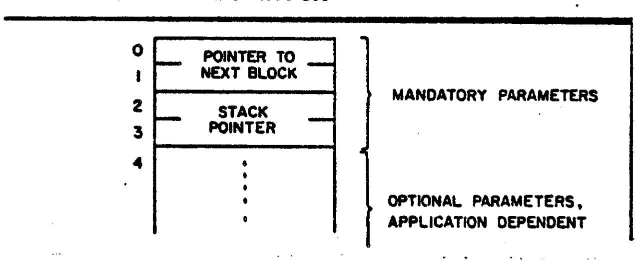
Whenever a process suspends itself by executing a subroutine jump to SPND, the SPND routine swaps the process-control block pointer (PCB) and the stack pointer to set up the next process. Then a simple return instruction causes the program for that process to start running again where it left off. Each process-control block contains at least two parameters: a pointer to the next control block and a stack pointer the current control block. The control blocks are arranged in a circular list so that SPND will automatically return to the first block after executing the last.

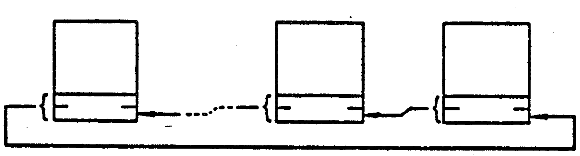
A sample program illustrates how process control

A six-instruction executive routine

```
SPND LDX PCB   SET INDEX REGISTER TO CURRENT
               CONTROL BLOCK
     STS 2,X   SAVE CURRENT STACK POINTER
     LDX X     SET INDEX REGISTER TO NEXT CON-
               TROL BLOCK
     STX PCB   SAVE CONTROL BLOCK POINTER
     LDS 2,X   GET NEW STACK POINTER
     RTS       RETURN TO PROCESS
```

Process-control block





Each block points to the next in a circular list.

Start-up routine

```
* SET INDEX REG-
* ISTER TO FIRST
* CONTROL BLOCK,
* GET CORRESPONDING
* STACK POINTER,
* AND BEGIN EXECUTION:
*
START LDX PCB
      LDS 2,X
      RTS
```

A sample program

```
* APPLICATION DEPENDENT
* PROCESS CONTROL BLOCK
* PARAMETERS:
*
MODE     EQU 4
STAT     EQU 5
BUFIN    EQU 6
BUFOUT   EQU 8
*
* FETCH A BUFFER. IF
* NO BUFFERS ARE AVAIL-
* ABLE, SUSPEND AND TRY
* AGAIN:
*
IDLE     JSR BUFGET
         BNE READY
         JSR SPND
         BRA IDLE
*
* PREPARE TO RECEIVE:
*
READY    LDX PCB
         CLR STAT,X
         LDAA #1
         STAA MODE,X
         JSR RCV
*
* SUSPEND. THEN, IF
* AN INPUT MESSAGE HAS
* BEGUN, GO TO INPUT.
* IF AN OUTPUT MESSAGE
* IS WAITING, GO TO
* OUTPUT. OTHERWISE,
* REPEAT:
*
LOOP     JSR SPND
         LDAA STAT,X
         BNE INPUT
         LDX BUFOUT,X
         BNE OUTPUT
         BRA LOOP
```

blocks and SPND instructions work together. Only the idle loop is shown. The complete program supervises an interrupt routine, which handles message flow to and from a teletypewriter. The program sets the teletypewriter port to receive, then waits for the interrupt routine to receive the first character. If an output message arrives first, the port is switched to output, and the message is printed on the teletypewriter. Notice that the program always suspends itself while waiting for something to happen.

All communication between background and interrupt levels occurs via the process-control block. MODE, for example, tells the interrupt routine whether to send or receive. STAT tells the background program that the interrupt routine has started or completed the message. BUFIN and BUFOUT are pointers to tell the interrupt routine where to store an input message or find an output message in microprocessor memory.

In a typical communications application, there might be several I/O ports, each having its own process-control block. Each control block may have a separate background program, or a single program may be shared by all control blocks. The sample program can be shared by multiple control blocks, because

- All data references are either to or through the control block.
- Each control block has its own return address stack.

Be careful with this multitask operation, however. Remember:

1. Processes aren't suspended by a "time-slicing" interrupt, but must suspend themselves often enough to let other programs run.
2. Processes should suspend themselves only at points where it is safe to lose the register contents, since SPND doesn't restore any registers except the stack pointer. (If this is a problem, register-save-and-restore instructions can easily be added to the SPND routine.)
3. Interrupts may remain enabled continuously, but every control block's return-address stack should be large enough to accommodate every interrupt routine's worst-case requirements.
4. All control blocks and stacks as well as the PCB pointer must contain proper initial values before starting the system. A brief start-up routine initiates normal operation.

David W. Johnson, formerly Senior Engineer with Control Data Corp., Santa Ana, CA 92704 now with NCR Corp., 3325 Platt Springs Rd., West Columbia, SC 29169.

COMPARE TWO MEMORY AREAS

```
 LABEL OPCODE OPERAND(S)
 1       8      3

* THIS PROGRAM COMPARES TWO AREAS OF MEMORY BYTE FOR BYTE. *
* IT IS INTENDED TO TEST PROGRAM LOADING AT 200H AGAINST   *
* THE SAME PROGRAM LOADED AT 2880H.  IT THEREFORE SERVES   *
* AS A MEMORY TEST OR CAN BE USED TO COMPARE TWO PROGRAMS  *
* AGAINST EACH OTHER.  WHEN AN UNEOUAL CONDITION IS FOUND  *
* LOCATION AND MEMORY CONTENTS ARE DISPLAYED AND PROGRAM   *
* EXITS TO MONITOR.  TO CONTINUE, CORRECT THE BYTE IN ERROR*
* AND RESTART THIS PROGRAM FROM LOAD ADDRESS.  IT CAN BE   *
* RELOCATED ANYWHERE IN MEMORY, AND THE TWO COMPARE AREAS  *
* CAN BE RELOCATED ALSO.                                   *
************************************************************
********************** H. G. FRIEDMAN *********************
******************** FEBRUARY 12, 1981 ********************
*
*EQUATES
*
PRGRM   EQU   $0200
COPY    EQU   $2880
DBUG    EQU   $FE64
PUTHEX  EQU   $FF02
PUTCHR  EQU   $FCBC
BYTI    EQU   $C0
PTRI    EQU   $C2
PTRI+I  EQU   $C3
PTR2    EQU   $C4
PTR2+I  EQU   $C5

        ORG   $4F00      RELOCATE AS REQUIRED
        LDX   #PRGRM     LD X REG W/FIRST ADDRESS
        LDAA  0,X        LD FIRST DATA BYTE
        STAA  BYTI       SAVE IT
        INX              ADVANCE X POINTER
        STX   PTRI       SAVE IT
        LDX   #COPY      LD X REG W/COMPARE ADDRESS
        LDAA  0,X        LD COMPARE DATA BYTE
TEST    CMPA  BYTI       CIF =
        BNE   OUT        NO, GO EXIT
        INX              ADVANCE X POINTER
        STX   PTR2       SAVE IT
        LDX   PTRI       BEGIN NEXT COMPARE
        LDAA  0,X
        STAA  BYTI
        INX
        STX   PTRI       SAVE NEXT X POINTER
        LDX   PTR2       RESTORE NEXT COMPARE ADDRESS
        LDAA  0,X
        BRA   TEST       LOOP TO DO AGAIN
OUT     LDAA  PTRI       BEGIN EXIT ROUTINE
        JSR   PUTHEX     PRINT HI ORDER ADDRESS
        LDAA  PTRI+I     LD LO ORDER ADDRESS
        SUBA  #01        ADJUST POINTER
        JSR   PUTHEX     PRINT LO ORDER ADDRESS
        LDAA  #$20       LD A SPACE
        JSR   PUTCHR     PRINT IT
        LDAA  PTR2       PRINT COMPARE ADDRESS
        JSR   PUTHEX     HI ORDER
        LDAA  PTR2+I
        JSR   PUTHEX     LO ORDER
        LDAA  #$60       CARRIAGE RETURN, LINE
        JSR   PUTCHR                       FEED
        LDAA  #$20       LD A SPACE
        XSR   RMXEMR     RRXMX XX
        LDAB  #$0B       LD COUNT
LOOP    JSR   PUTCHR     PRINT SPACE
        DECB
        BNE   LOOP       PRINT II SPACES
        LDX   PTRI
        DEX              ADJUST X POINTER
        LDAA  0,X
        JSR   PUTHEX     PRINT MEMORY CONTENTS
        LDAA  #$20       LD A SPACE
        LDAB  #$03       LD COUNT
LOOPI   JSR   PUTCHR     PRINT SPACE
        DECB
        BNE   LOOPI      PRINT 3 SPACES
        LDX   PTR2
        LDAA  0,X
        JSR   PUTHEX     PRINT MEMORY CONTENTS
        JSR   DBUG       EXIT TO MONITOR
        END
```

PROGRAM GETCHR   ASSEMBLY LISTING

```
ADDR  OBJECT      SPHERE          MOTOROLA

                                         NAM    GETCHR
                  =      800             ORG    $800
                                  * FOLLOWING ARE LOW MEMORY LOCATIONS
                  A=     1A       ENDMEM EQU    $1A       TEMP FOR REPEATED KBD CHARS
                  B=     1B       ENDME1 EQU    $1B       ENDMEM+1
                  C=     1C       CSRPTR EQU    $1C       SCREEN CURSOR LOCATION
                                  * FOLLOWING IS A PDS-V3A AND PDS-V3N LOCATION
                  %=     FC64     KBDPIA EQU    $FC64     CONTAINS KBD PIA ADDRESS
0800  86 16       VX 86 D16       INITKB LDA A  #$16      INIT KEYBOARD, NO INTERRUPTS
0802  FE FC64        FE E@%              LDX    KBDPIA    GET KEYBOARD PIA ADDRESS
0805  A7 01          A7 D01              STA A  1,X
0807  7F 001A        7F E@A              CLR    ENDMEM    SET UP FOR NO REPEATED KEY
080A  7F 001B        7F E@B              CLR    ENDME1    UPPER CASE NORMAL FLAG
080D  39             39                  RTS
080E  96 1B       VP 96 D@B       GETCHR LDA A  ENDME1    GET UPPER/LOWER CASE FLAG
0810  36             36                  PSH A            SAVE CASE FLAG
0811  96 1A          96 D@A              LDA A  ENDMEM    TEST FOR KEY REPEATED
0813  36             36                  PSH A            SAVE ENDMEM VALUE
0814  26 33          26 R@VT             BNE    GET6      USE SAVED KEY VALUE IF REPEATED
0816  DE 1C       VQ DE D@C       GETCHZ LDX    CSRPTR    GET SCREEN WRITE ADDRESS
0818  63 00          63 D00              COM    0,X       SET BLINK BLOCK ON OR OFF
081A  CE 1400        CE E1400            LDX    #5120     SET BLINK COUNT
081D  DF 1A          DF D@A              STX    ENDMEM    SAVE X REGISTER
081F  DE 1A       VR DE D@A       GET1   LDX    ENDMEM
0821  09             09                  DEX
0822  27 F2          27 R@VQ             BEQ    GETCHZ
0824  86 40          86 D40              LDA A  #$40      TEST FOR KBD CHAR AVAILABLE
0826  DF 1A          DF D@A              STX    ENDMEM    SAVE X REGISTER
0828  FE FC64        FE E@%              LDX    KBDPIA    GET KBD PIA ADDRESS
082B  A5 01          A5 D01              BIT A  1,X       TEST FOR KBD CHAR AVAILABLE
082D  27 F0          27 R@VR             BEQ    GET1
082F  DE 1C          DE D@C              LDX    CSRPTR    MAKE SURE ORIGINAL SCREEN
0831  A6 00          A6 D00              LDA A  0,X       CHAR IS PRESENT
0833  2A 02          2A R@VS             BPL    GET2
0835  63 00          63 D00              COM    0,X
0837  FE FC64     VS FE E@%       GET2   LDX    KBDPIA    GET KBD PIA ADDRESS
083A  A6 00          A6 D00              LDA A  0,X       GET KBD CHAR
083C  81 01          81 D01              CMP A  #$01      CHECK FOR CONTROL-A
083E  26 09          26 R@VT             BNE    GET6      BRANCH IF REGULAR CHAR
0840  30             30                  TSX              GET UPPER/LOWER CASE FLAG
0841  A6 02          A6 D02              LDA A  1,X
0843  88 80          88 D80              EOR A  #$80      FLIP UPPER/LOWER CASE FLAG
0845  A7 02          A7 D02              STA A  1,X       STORE UPPER/LOWER CASE FLAG
```

```
COLUMN    1         2         3         4         5         6         7
123456789012345678901234567890123456789012345678901234567890123456789012
```

PROGRAM GETCHR   ASSEMBLY LISTING

```
ADDR OBJECT      SPHERE              MOTOROLA

0847 20 D6         20 R@VR                BRA    GET1      NEXT CHARACTER
0849 36         VT 36          GET6    PSH A              SAVE KEY VALUE
084A 30            30                  TSX                GET ENDMEM VALUE
084B A6 01         A6 D01              LDA A  1,X
084D CE 1000       CE E1000            LDX    #4096      SET LOOP FOR QUARTER SECOND
0850 4D            4D                  TST A              IF REPEAT, USE SIXTEENTH SECOND
0851 27 03         27 R@VU             BEQ    GET7
0853 CE 0400       CE E0400            LDX    #1024
0856 DF 1A      VU DF D@A      GET7    STX    ENDMEM     SAVE LOOP COUNT
0858 32            32                  PUL A              RESTORE KEY VALUE
0859 36            36                  PSH A
085A FE FC64       FE E@%              LDX    KBDPIA     GET KBD PIA ADDRESS
085D A1 00         A1 D00              CMP A  0,X        IS KEY STILL DOWN
085F 27 03         27 R@VV             BEQ    GET8       LOOP IF KEY STILL DOWN
0861 4F            4F                  CLR A              FLAG TO INDICATE KEY UP
0862 20 05         20 R@VW             BRA    GET10
0864 DE 1A      VV DE D@A      GET8    LDX    ENDMEM     RESTORE LOOP COUNT
0866 09            09                  DEX
0867 26 ED         26 R@VU             BNE    GET7       LOOP WHILE KEY IS DOWN
0869 97 1A      VW 97 D@A      GET10   STA A  ENDMEM     SAVE KEY VALUE OR KEY-UP FLAG
086B 30            30                  TSX                GET UPPER/LOWER CASE FLAG
086C A6 02         A6 D02              LDA A  2,X
086E 97 1B         97 D@B              STA A  ENDME1     SAVE CASE FLAG
0870 32            32                  PUL A              RESTORE KBD CHARACTER
0871 31            31                  INS                CLEAR OUT ENDMEM VALUE
0872 31            31                  INS                CLEAR OUT ENDME1 VALUE
0873 2A 15         2A R@VC             BPL    GET14
0875 81 41         81 D41              CMP A  #$41       CHECK FOR UPPER CASE A TO Z
0877 2D 11         2D R@VC             BLT    GET14
0879 81 5A         81 D5A              CMP A  #$5A
087B 2E 03         2E R@VB             BGT    GET13
087D 8B 20         8B D20              ADD A  #$20       SHIFT UPPER CASE CHAR TO LOWER
087F 39            39                  RTS
0880 81 61      VB 81 D61      GET13   CMP A  #$61       CHECK FOR LOWER CASE A TO Z
0882 2D 06         2D R@VC             BLT    GET14
0884 81 7A         81 D7A              CMP A  #$7A
0886 2E 02         2E R@VC             BGT    GET14
0888 80 20         80 D20              SUB A  #$20       SHIFT LOWER CASE CHAR TO UPPER
088A 39         VC 39          GET14   RTS

                END                    END

                COLUMN    1         2         3         4         5         6         7
                123456789012345678901234567890123456789012345678901234567890123456789012
```

```
                                         NAM    DUMBTER
                  =    4800              ORG    $4800
                                  TSMODE EQU    $09      TERMINAL MODE--/16, 7 BIT, EVEN
                  $=   1C         CSRPTR EQU    $1C      SCREEN CURSOR LOCATION
                  C=   28         TEMP2C EQU    $28      TWO-BYTE TEMP
                  *=   38         ACIANO EQU    $38      CASSETTE ACIA ADDRESS
                  ==   FC3D       CLEAR  EQU    $FC3D    CLEAR SCREEN
                  I=   FC64       KBDPIA EQU    $FC64    CONTAINS KBD PIA ADDRESS
                  >=   FC8F       INSERT EQU    $FC8F    EDIT A CHARACTER ON SCREEN
4800  DE 38       GG DE D@*       INTLZX LDX    ACIANO   GET TIMESHARING ACIA ADDRESS
4802  86 13          86 D13              LDA A  #$13     MASTER RESET ACIA
4804  A7 00          A7 D00              STA A  0,X
4806  86 09          86 D09              LDA A  #TSMODE  SET ACIA--/16 7 BIT EVEN PAR.
4808  A7 00          A7 D00              STA A  0,X
480A  BD FC3D        BD E@=              JSR    CLEAR    CLEAR SCREEN
480D  DE 1C       GH DE D@$       BLINK  LDX    CSRPTR   BLINK SCREEN CURSOR
480F  63 00          63 D00              COM    0,X      SHIFT TO/FROM CURSOR BLOCK
4811  CE 1800        CE E1800            LDX    #$1800   LOAD BLINK COUNT
4814  DF 28          DF D@C              STX    TEMP2C   SAVE BLINK COUNT
4816  DE 28       GI DE D@C       BLINK1 LDX    TEMP2C   RESTORE BLINK COUNT
4818  09             09                  DEX
4819  27 F2          27 R@GH             BEQ    BLINK    RESTART BLINK COUNTER
481B  DF 28       GY DF D@C       ACIAIN STX    TEMP2C   SAVE X REGISTER
481D  86 01          86 D01              LDA A  #$01
481F  DE 38          DE D@*              LDX    ACIANO
4821  A5 00          A5 D00              BIT A  0,X      TEST RECEIVE REGISTER FULL
4823  27 05          27 R@GZ             BEQ    KBDIN    BRANCH IF NO ACIA INPUT
4825  A6 01          A6 D01              LDA A  1,X      GET INPUT CHAR
4827  BD FC8F        BD E@>              JSR    INSERT   DISPLAY ACIA INPUT CHAR
482A  FE FC64     GZ FE E@I       KBDIN  LDX    KBDPIA   GET KEYBOARD PIA ADDR
482D  86 40          86 D40              LDA A  #$40     GET TEST MASK
482F  A5 01          A5 D01              BIT A  1,X      TEST FOR KEYBOARD CHAR
4831  27 E3          27 R@GI             BEQ    BLINK1   LOOP IF NO KBD INPUT
4833  DE 1C          DE D@$              LDX    CSRPTR   RESET BLINK CHAR
4835  A6 00          A6 D00              LDA A  0,X
4837  2A 02          2A R@HE             BPL    KBDIN1
4839  63 00          63 D00              COM    0,X
483B  FE FC64     HE FE E@I       KBDIN1 LDX    KBDPIA   GET KEYBOARD PIA ADDR
483E  A6 00          A6 D00              LDA A  0,X      GET KBD CHAR
4840  27 0F          27 R@HH             BEQ    EXIT     END IF 00 HEX
4842  C6 02       HF C6 D02       ACOUT  LDA B  #$02
4844  DE 38          DE D@*              LDX    ACIANO
4846  E5 00       HG E5 D00       ACOT10 BIT B  0,X      TEST SEND REGISTER FULL
4848  27 FC          27 R@HG             BEQ    ACOT10   LOOP UNTIL EMPTY
484A  A7 01          A7 D01              STA A  1,X      SEND KBD CHAR
484C  BD FC8F        BD E@>              JSR    INSERT   DISPLAY CHAR IN ASCII
484F  20 BC          20 R@GH             BRA    BLINK    CONTINUE MAIN LOOP
4851  39          HH 39           EXIT   RTS
                  END                    END
```

PROGRAM CIRCLE ASSEMBLY LISTING

```
LINE  ADDR  OBJECT      SPHERE           MOTOROLA

                                          NAM    CIRCLE
                                   *
                                   * CIRCLE--JANUARY 12, 1978 VERSION
                                   *
                                   * PROGRAMMED BY
                                   * JIM RAEHL, 943 BEGONIA, ESCONDIDO, CA  92027
                                   * (714) 746-3562  (714) 485-2580 (WORK)
                                   *
                                   * THIS PROGRAM IS TAKEN FROM AN ARTICLE ENTITLED
                                   * "SERENDIPITOUS CIRCLES" IN THE AUGUST 1977 ISSUE OF
                                   * BYTE MAGAZINE (P. 70). THE ORIGINAL PROGRAM USED A
                                   * DAC TO DISPLAY THE OUTPUT ON AN OSCILLOSCOPE. THIS
                                   * VERSION IS ADAPTED FOR THE SPHERE CRT BOARD.
                                   * INSTRUCTIONS ARE GIVEN IN BOXES IN THE PROGRAM TO
                                   * ADAPT IT TO A CRT8 OR GRAPHICS BOARD. THE ORIGINAL
                                   * PROGRAM WAS FOUND TO BE TOO DULL  I LIVENED IT UP
                                   * BY ALTERING THE 16-BIT X AND Y COORDINATE VALUES BY
                                   * A DETERMINED ALGORITHM--INCREMENT THE COORDINATE BY
                                   * A CONSTANT, AND THEN INCREMENT THE CONSTANT FOR MORE
                                   * VARIETY. I ALSO FOUND THAT PREVIOUS POINTS HAD TO BE
                                   * ERASED, OR THE SCREEN WOULD EVENTUALLY FILL UP.
                                   * THUS, ONLY THE 64 MOST RECENT POINTS ARE RETAINED.
                                   * IT WAS ALSO FOUND TO BE MORE INTERESTING IF THE RATE
                                   * OF DISPLAY WAS SLOWED DOWN.
                                   *

                                   ***********************************************
                                   *
                                   * CIRCLE DRAWING PROGRAM
                                   * WRITTEN BY WILLIAM F. GALWAY
                                   * 30 DEC 76
                                   * THE ALGORITHM USED IS AS FOLLOWS
                                   *
                                   * LOOP
                                   *      X <- X - Y/2
                                   *      Y <- Y + X/2
                                   *      GOTO LOOP
                                   *
                                   ***********************************************
                                   *
                                   * SIXTEEN BIT ARITHMETIC IS USED ALTHOUGH ONLY
                                   * 5 BITS OF X AND 4 BITS OF Y ARE DISPLAYED.
                                   ***********************************************
                                   *
                                   * SOME GOOD STARTING VALUES FOR (X,Y) ARE
                                   *   (7F00,0000)
                                   *   (7F00,8100)
                                   *   (7D00,7D00)
                                   *   (7CF3,7CF3)
                                   *

COLUMN     1         2         3         4         5         6
123456789012345678901234567890123456789012345678901234567890123456789012
```

PROGRAM CIRCLE   ASSEMBLY LISTING

```
ADDR  OBJECT      SPHERE              MOTOROLA

                                  ************************************************
                                  *
                                  * THE PATTERN GENERATION CAN BE VARIED BY CHANGING THE
                                  * OPERAND FIELDS OF CERTAIN INSTRUCTIONS. THESE ARE
                                  * MARKED WITH **N IN THE COMMENT FIELD, WHERE N IS A
                                  * NUMBER. THE INSTRUCTION CAN BE CHANGED TO NOP (HEX 01)
                                  * IN ALL ITS BYTES TO DISABLE THE FEATURE. FEATURE
                                  * EXPLANATIONS
                                  * 1. STARTING VALUES OF X AND Y (SEE ABOVE). SMALL UPPER
                                  *    BYTE VALUES GENERATE SMALL CIRCLES; BIGGER ONES
                                  *    GENERATE BIGGER CIRCLES. IF THESE INSTRUCTIONS ARE
                                  *    SET TO NOP, THE INITIAL VALUES DEPEND ON WHAT IS IN
                                  *    WK1 AND WK2.
                                  * 2. CENTER OF CIRCLE. IF SET TO LOAD 0, THE CENTER IS
                                  *    IN THE CORNERS (THE ONLY OTHER INTERESTING PLACE).
                                  * 3. X AND Y BIAS TO AVOID FIXED PATTERNS. IF SET TO
                                  *    LOAD 0'S, THE POINT GENERATION GOES INTO A FIXED
                                  *    LOOP.
                                  * 4. DELAY COUNT TO SLOW PATTERN GENERATION, SO IT'S
                                  *    NOT A BLINDING BLUR.
                                  * 5. ERASE BUFFER END, TO ERASE 64TH PREVIOUS POINT.
                                  ************************************************

                   =     400             ORG    $400

                                  * DEFINE TEMP STORAGE LOCATIONS

                  A=     50       WK1    EQU    $50       X VALUE
                  B=     52       WK2    EQU    $52       Y VALUE
                  G=     54       WK3    EQU    $54       DELAY COUNT
                  C=     56       WK4    EQU    $56       ARITHMETIC TEMP
                  O=     57       WK41   EQU    $57
                  D=     58       WK5    EQU    $58       SAVE FOR X REGISTER
                  I=     5A       WK6    EQU    $5A       SAVE FOR HIGH 5 BITS OF X
                  J=     5C       WK7    EQU    $5C       X BIAS TO AVOID FIXED PATTERN
                  M=     5D       WK71   EQU    $5D
                  K=     5E       WK8    EQU    $5E       Y BIAS TO AVOID FIXED PATTERN
                  N=     5F       WK81   EQU    $5F
                  P=     60       WK9    EQU    $60       PTR TO CURRENT ERASE BUFF POS
                  Q=     62       WK10   EQU    $62       Y SCREEN CENTER BIAS
                  R=     64       WK11   EQU    $64       X SCREEN CENTER BIAS
                  S=     66       WK12   EQU    $66       ERASE BUFFER END LOCATION

                                  * SET INITIAL VALUES

0400  CE 7CF3     E  CE E7CF3     START  LDX    #$7CF3    **1--SET X INITIAL VALUE
0403  DF 50          DF D@A              STX    WK1
0405  CE 7CF3        CE E7CF3            LDX    #$7CF3    **1--SET Y INITIAL VALUE
0408  DF 52          DF D@B              STX    WK2
040A  86 01          86 D01              LDA A  #$01      **2--HEX 8 TO ADD TO TOP Y 4 BITS

                  COLUMN  1         2         3         4         5         6         7
                  1234567890123456789012345678901234567890123456789012345678901234567890123456789012
```

PROGRAM CIRCLE   ASSEMBLY LISTING

```
LINE ADDR OBJECT      SPHERE          MOTOROLA

     040C 97 62        97 D@Q             STA A  WK10
     040E 86 10        86 D10             LDA A  #$10     **2--HEX 10 TO ADD TO TOP X 4 BIT
     0410 97 64        97 D@R             STA A  WK11
     0412 86 03        86 D03             LDA A  #3       **3--X BIAS TO AVOID FIXED PATT
     0414 97 5C        97 D@J             STA A  WK7
     0416 86 01        86 D01             LDA A  #1       **3--BIAS INCREMENT
     0418 97 5D        97 D@M             STA A  WK71
     041A 86 03        86 D03             LDA A  #3       **3--Y BIAS TO AVOID FIXED PATT
     041C 97 5E        97 D@K             STA A  WK8
     041E 86 01        86 D01             LDA A  #1       **3--BIAS INCREMENT
     0420 97 5F        97 D@N             STA A  WK81
     0422 CE 0800      CE E0800           LDX    #$0800   **4--DELAY COUNT FOR POINT GEN
     0425 DF 54        DF D@G             STX    WK3
     0427 CE 00B0      CE E@U             LDX    #PTEND   **5--END OF ERASE BUFFER
     042A DF 66        DF D@S             STX    WK12

                                   * INITIALIZE ERASE BUFFER TO HARMLESS ERASE LOCS

     042C CE 0070      CE E@T             LDX    #PTBUF   GET POINT BUFFER LOC
     042F DF 60        DF D@P             STX    WK9      INIT CURRENT BUFFER LOC POINTER
     0431 86 D2        86 DD2             LDA A  #$D2     HARMLESS LOC VALUE
     0433 A7 00      V A7 D00      ELOOP  STA A  0,X
     0435 08           08                 INX             NEXT LOC TO INIT
     0436 9C 66        9C D@S             CPX    WK12     LAST LOC TO INIT?
     0438 26 F9        26 R@V             BNE    ELOOP    LOOP UNTIL DONE

                                   * CLEAR SCREEN

     043A FE FC40      FE EFC40           LDX    $FC40    GET FIRST SCREEN LOC TO CLEAR
     043D C6 20        C6 D20             LDA B  #$20     SET SCREEN TO SPACES
     043F 09         Y 09          DLOOP  DEX             NEXT LOC TO CLEAR
     0440 E7 00        E7 D00             STA B  0,X
     0442 BC FC38      BC EFC38           CPX    $FC38    LAST LOC TO CLEAR?
     0445 26 F8        26 R@Y             BNE    DLOOP    LOOP UNTIL DONE

                                   *************************************************
                                   *
                                   * IF THE CRT USED IS A CRT8, THE 7 INSTRUCTIONS
                                   * FOLLOWING THIS BLOCK SHOULD BE REPLACED WITH THE
                                   * CODE IN THIS BLOCK. THE CODE IN THIS BLOCK EXTRACTS
                                   * THE TOP BITS OF THE X VALUE SUCH THAT THE RESULT
                                   * IS A VALUE FROM 0 TO 79. THIS IS DONE BY EXTRACTING
                                   * THE TOP 7 BITS (VALUE 128), MULTIPLYING BY 5, AND
                                   * DIVIDING BY 8. IN GENERAL, ANY SCREEN WIDTH CAN BE
                                   * ACCOMMODATED IN THIS WAY. THE NEXT HIGHER MULTIPLE
                                   * OF 2 ABOVE THE SCREEN WIDTH IS FOUND. THE
                                   * CORRESPONDING NUMBER OF TOP BITS IS EXTRACTED AND
                                   * SHIFTED TO THE RIGHT BORDER OF A BYTE. IF THE
                                   * SCREEN LENGTH IS A MULTIPLE OF 2, WE ARE DONE. IF
                                   * NOT, FIND A FRACTION WHICH CAN BE USED TO MULTIPLY

                 COLUMN    1         2         3         4         5         6
                 123456789012345678901234567890123456789012345678901234567890123456789012
```

PROGRAM CIRCLE   ASSEMBLY LISTING

```
ADDR  OBJECT      SPHERE            MOTOROLA

                                    * THE NEXT HIGHER POWER OF 2 TO GET THE SCREEN WIDTH.
                                    * MULTIPLY THE EXTRACTED RESULT BY THE NUMERATOR OF
                                    * THIS FRACTION. THE MULTIPLY IS DONE BY SHIFTING THE
                                    * EXTRACTED RESULT TO GET MULTIPLIES BY 2, AND ADDING
                                    * THE DESIRED PARTIAL PRODUCTS. THE RESULT OF THE
                                    * MULTIPLIES IS DIVIDED BY THE DENOMINATOR, WHICH MUST
                                    * BE A POWER OF 8. THE DIVIDE IS DONE BY RIGHT SHIFTING.
                                    *
                                    * DISPLAY ALGORITHM FOR CRT8
                                    *   POSITION = $E000 +
                                    *              ((TOP Y 5 BITS * 3 / 4 + 12) MOD 24)*80 +
                                    *              ((TOP X 7 BITS * 5 / 8 + 40) MOD 80)
                                    *
                                    *LOOP  LDA A  WK1      GET HI ORDER BYTE OF X
                                    *      ASR A           USE HIGH 7 BITS
                                    *      AND A  #$7F
                                    *      STA A  WK6      SAVE ORIGINAL EXTRACTED VALUE
                                    *      ASL A           MULTIPLY BY 4
                                    *      ROL A
                                    *      ADC B  #0       PUT OVERFLOW IN B
                                    *      ADD A  WK6      ADD ORIGINAL VALUE
                                    *      ADC B  #0       NOW FULLY MULTIPLIED BY 5
                                    *      ASR A           DIVIDE LOWER PART OF RESULT BY 8
                                    *      ASR A
                                    *      ASR A
                                    *      AND A  #$1F     GET RID OF SIGN PROPAGATION
                                    *      ASL B           SHIFT UPPER PART OF RESULT
                                    *      ASL B           TO ADD TO A
                                    *      ASL B
                                    *      ASL B
                                    *      ASL B
                                    *      ABA             ADD TOGETHER LOWER & UPPER PARTS
                                    *      ADD A  #40      **2--SHIFT ORIG TO 40TH LINE CHAR
                                    *      CMP A  #80      CHECK FOR OVERFLOW
                                    *      BLT    GLOOP    BRANCH IF NO OVERFLOW
                                    *      SUB A  #80      SHIFT OVERFLOW TO LINE FIRST HALF
                                    *
                                    ***********************************************

                                    * DISPLAY ALGORITHM FOR CRT1
                                    *   POSITION = $E000 +
                                    *              ((TOP Y 4 BITS + HEX 8) MOD HEX 10)*32 +
                                    *              ((TOP X 5 BITS + HEX 10) MOD HEX 20)

0447  96 50     F  96 D@A           LOOP   LDA A  WK1      GET HI ORDER OF X
0449  47           47                      ASR A           USE HIGH 5 BITS
044A  47           47                      ASR A
044B  47           47                      ASR A
044C  84 1F        84 D1F                  AND A  #$1F
044E  9B 64        9B D@R                  ADD A  WK11     ADD HEX 10 TO TOP X 5 BITS

                COLUMN   1         2         3         4         5         6         7
                1234567890123456789012345678901234567890123456789012345678901234567890123456789012
```

PROGRAM CIRCLE   ASSEMBLY LISTING

```
ADDR  OBJECT       SPHERE            MOTOROLA

0450  84 1F        84 D1F                 AND A  #$1F     MOD HEX 20
0452  97 5A      Z 97 D@I          GLOOP  STA A  WK6      SAVE HIGH 5 BITS OF X
                                   ***********************************************
                                   *
                                   * IF A CRT8 IS USED, THE 6 INSTRUCTIONS FOLLOWING
                                   * THIS BLOCK SHOULD BE REPLACED WITH THE CODE IN THIS
                                   * BLOCK. THE CODE IN THIS BLOCK EXTRACTS THE TOP BITS OF
                                   * THE Y VALUE SUCH THAT THE RESULT IS A VALUE FROM 0 TO
                                   * 24. THIS IS DONE BY EXTRACTING THE TOP 5 BITS (VALUE
                                   * 32), MULTIPLYING BY 3, AND DIVIDING BY 4. THE RESULT
                                   * IS SHIFTED TO THE 12TH SCREEN LINE, AND MULTIPLIED BY
                                   * 80.
                                   *
                                   *      LDA A  WK2      GET HI ORDER BITS OF Y
                                   *      ASR A
                                   *      ASR A
                                   *      AND A  #$3E     USE HIGH 5 BITS
                                   *      TAB             PUT HI 5 BITS*2 IN B
                                   *      ASR A
                                   *      ABA             BITS*2 + BITS = BITS*3
                                   *      ASR A           DIVIDE (RESULT*3) BY 4
                                   *      ASR A
                                   *      ADD A  #12      **2--SHIFT ORIGIN TO 12TH LINE
                                   *      CMP A  #24      GET (DIVIDE RESULT) MOD 24
                                   *      BLT    YLOOP
                                   *      SUB A  #24
                                   *YLOOP CLR B
                                   *      ASL A           MULTIPLY (MOD RESULT) BY 16
                                   *      ASL A
                                   *      ASL A
                                   *      ROL A
                                   *      ADC B  #0
                                   *      STA B  WK4      SAVE MULTIPLIED RESULT
                                   *      STA A  WK41
                                   *      ASL B           MULTIPLY (MOD RESULT) BY 64
                                   *      ROL A
                                   *      ADC B  #0
                                   *      ASL B
                                   *      ROL A
                                   *      ADC B  #0
                                   *      ADD A  WK41     ADD (MULT BY 64) AND (MULT BY 16)
                                   *      ADC B  WK4
                                   *
                                   ***********************************************
0454  96 52        96 D@B                 LDA A  WK2      GET HI ORDER OF Y
0456  5F           5F                     CLR B
0457  49           49                     ROL A           USE HI ORDER 4 BITS OF Y * 32

                   COLUMN   1         2         3         4         5         6         7
                   123456789012345678901234567890123456789012345678901234567890123456789012
```

PROGRAM CIRCLE ASSEMBLY LISTING

```
ADDR  OBJECT       SPHERE              MOTOROLA

0458  D9 62          D9 D@Q                  ADC B  WK10     ADD HEX 8 TO TOP Y 4 BITS
045A  C4 01          C4 D01                  AND B  #$01     MOD HEX 10
045C  84 E0          84 DE0                  AND A  #$E0
045E  20 02          20 R@2                  BRA    ALOOP

0460  20 E5      1   20 R@F          BLOOP   BRA    LOOP

0462  FB FC38    2   FB EFC38        ALOOP   ADD B  $FC38    ADD SCREEN BASE ADDRESS
0465  9B 5A          9B D@I                  ADD A  WK6      ADD X VALUE
0467  C9 00          C9 D00                  ADC B  #0
0469  DE 60          DE D@P                  LDX    WK9      GET 64TH PREVIOUS PT LOC
046B  EE 00          EE D00                  LDX    0,X
046D  37             37                      PSH B           SAVE B
046E  C6 20          C6 D20                  LDA B  #$20     SPACE TO ERASE 64TH PREV LOC
0470  E7 00          E7 D00                  STA B  0,X      **5--CLEAR 64TH PREVIOUS POINT
0472  33             33                      PUL B           RESTORE B
0473  DE 60          DE D@P                  LDX    WK9
0475  E7 00          E7 D00                  STA B  0,X      SAVE CURRENT POINT LOCATION
0477  A7 01          A7 D01                  STA A  1,X
0479  EE 00          EE D00                  LDX    0,X      PUT CURRENT LOC INTO X
047B  86 6F          86 D6F                  LDA A  #$6F     DISPLAY SMALL LETTER O
047D  A7 00          A7 D00                  STA A  0,X      DISPLAY THE POINT
047F  DE 60          DE D@P                  LDX    WK9
0481  08             08                      INX             NEXT PREVIOUS POINT TO CLEAR
0482  08             08                      INX
0483  9C 66          9C D@S                  CPX    WK12     WRAP AROUND TO BUFFER START
0485  26 03          26 R@W                  BNE    FLOOP    IF AT END
0487  CE 0070        CE E@T                  LDX    #PTBUF   BUFFER START LOC
048A  DF 60      W   DF D@P          FLOOP   STX    WK9      SAVE PTR TO BUFFER LOC
048C  CE 0050        CE E@A                  LDX    #WK1     SET X TO POINT TO WORK AREA

                                     * FIGURE NEXT POINT LOCATION

048F  E6 03          E6 D03                  LDA B  3,X      B GETS LO Y
0491  A6 02          A6 D02                  LDA A  2,X      A GETS HI Y
0493  47             47                      ASR A           GET Y/2
0494  56             56                      ROR B
0495  40             40                      NEG A           DO A 16 BIT NEGATE
0496  50             50                      NEG B           NEGATE LO ORDER
0497  82 00          82 D00                  SBC A  #0       PROPAGATE CARRY
0499  EB 01          EB D01                  ADD B  1,X      GET X+(-Y/2) LO ORDER
049B  A9 00          A9 D00                  ADC A  0,X      THEN HI ORDER
049D  E7 01          E7 D01                  STA B  1,X      STORE LO X
049F  A7 00          A7 D00                  STA A  0,X      THEN HI X
04A1  47             47                      ASR A           GET X/2 HI ORDER
04A2  56             56                      ROR B           LO ORDER
04A3  EB 03          EB D03                  ADD B  3,X      ADD LO Y
04A5  A9 02          A9 D02                  ADC A  2,X      ADD HI Y WITH CARRY
04A7  E7 03          E7 D03                  STA B  3,X      SAVE LO Y
04A9  D6 5E          D6 D@K                  LDA B  WK8      ADD Y BIAS TO AVOID FIXED PATT
```

```
COLUMN    1         2         3         4         5         6         7
1234567890123456789012345678901234567890123456789012345678901234567890012
```

PROGRAM CIRCLE   ASSEMBLY LISTING

```
LINE  ADDR  OBJECT      SPHERE              MOTOROLA

      04AB  1B             1B                    ABA
      04AC  DB 5F          DB D@N                ADD B  WK81     CHANGE Y BIAS
      04AE  D7 5E          D7 D@K                STA B  WK8
      04B0  A7 02          A7 D02                STA A  2,X      SAVE HI Y
      04B2  A6 00          A6 D00                LDA A  0,X      GET X HI BYTE
      04B4  D6 5C          D6 D@J                LDA B  WK7      ADD X BIAS TO AVOID FIXED PATT
      04B6  1B             1B                    ABA
      04B7  DB 5D          DB D@M                ADD B  WK71     CHANGE X BIAS
      04B9  D7 5C          D7 D@J                STA B  WK7

                                    * DELAY LOOP TO SLOW PATTERN GENERATION

      04BB  DE 54          DE D@G                LDX    WK3      DELAY LOOP TO SLOW PATTERNS
      04BD  09          L  09           TLOOP    DEX
      04BE  26 FD          26 R@L                BNE    TLOOP    LOOP UNTIL TIMED-OUT

                                    * TEST IF KEYBOARD KEY IS TYPED. EXIT IF SO.

      04C0  86 40          86 D40                LDA A  #$40     GET MASK FOR KEYBOARD
      04C2  FE FC64        FE EFC64              LDX    $FC64    GET KEYBOARD ADDRESS
      04C5  A5 01          A5 D01                BIT A  1,X      TEST FOR KEY TYPED
      04C7  27 97          27 R@1                BEQ    BLOOP    LOOP TO DISPLAY SOME MORE
      04C9  8D 03          8D R@X                BSR    CLEAR    CLEAR SCREEN
      04CB  DF 1C          DF D1C                STX    $1C      SET CURSOR POINTER
      04CD  39             39                    RTS             EXIT TO MONITOR

                                    * CLEAR SCREEN

      04CE  FE FC40     X  FE EFC40     CLEAR    LDX    $FC40    GET FIRST SCREEN LOC TO CLEAR
      04D1  C6 20          C6 D20                LDA B  #$20     SET SCREEN TO SPACES
      04D3  09          H  09           CLOOP    DEX             NEXT LOC TO CLEAR
      04D4  E7 00          E7 D00                STA B  0,X
      04D6  BC FC38        BC EFC38              CPX    $FC38    LAST LOC TO CLEAR?
      04D9  26 F8          26 R@H                BNE    CLOOP    LOOP UNTIL DONE
      04DB  39             39                    RTS

                        T=      070     PTBUF    EQU    $070     ERASE BUFFER START LOCATION
                        U=      0B0     PTEND    EQU    $0B0     ERASE BUFFER END LOCATION

                        END                      END

                        COLUMN  1         2         3         4         5         6         7
                        1234567890123456789012345678901234567890123456789012345678901234567890 12
```

```
                        PROGRAM WORM      ASSEMBLY LISTING
                                              MOTOROLA
ADDR   OBJECT            SPHERE
                                              NAM    WORM
                         =     E200           ORG    $E200

                         N=    02       TEMP2  EQU   $2      TEMP FOR LOCATION FINDING
                         1=    03       TEMP2A EQU   $3
                         O=    3C       BFRPTR EQU   $3C     START LOCATION TO CRAWL
                         P=    3E       BFRSZE EQU   $3E     TEMP FOR MOVING WORM TO START
                         Q=    38       ACIANO EQU   $38     TEMP FOR MOVING WORM TO START
                         R=    E78B     ADDSET EQU   $E78B   PARAMETER CRACK ROUTINE
                         Y=    FC38     SCBEG  EQU   $FC38   SCREEN START ADDRESS

                                        *****************************************
                                        * INTERFACE TO RAEHL EXTENDED MONITOR
                                        *****************************************

E200   81 4B             S  81 D4B      WORM   CMP A  #$4B     K FOR WORM FUNCTION
E202   26 31                26 R@D             BNE    WORME1
E204   BD E78B              BD E@R             JSR    ADDSET   GET LOCATION TO START FROM

                                        *****************************************
                                        * SET MEMORY TO SWI (3F), MOVE WORM TO START
                                        *****************************************

                                        * SET MEMORY TO SWI INSTRUCTIONS

E207   86 3F                86 D3F             LDA A  #$3F     SWI OP CODE
E209   DE 3C                DE D@O             LDX    BFRPTR   GET START ADDR TO CLEAR
E20B   DF 3E                DF D@P             STX    BFRSZE
E20D   A7 00             T  A7 D00      WORM1  STA A  0,X      SET LOCATION TO SWI
E20F   08                   08                 INX             NEXT LOCATION TO SET
E210   26 FB                26 R@T             BNE    WORM1    LOOP UNTIL CLEARED TO $FFFF

                                        * FIND THIS ROUTINE'S LOCATION (ASSUMES 100 HEX MULT)

E212   20 23                20 R@Z             BRA    WORM4A
E214   30                A  30          WORM0  TSX             STACK REGISTER TO X
E215   EE 00                EE D00             LDX    0,X      GET WORM PROGRAM START ADDR
E217   DF 02                DF D@N             STX    TEMP2
E219   86 8C                86 D8C             LDA A  #$8C     SECOND BYTE OF WORM END ADDR
E21B   97 03                97 D@1             STA A  TEMP2A   SET UP WORM END ADDR

                                        * MOVE WORM TO START LOCATION

E21D   9C 02             U  9C D@N      WORM2  CPX    TEMP2    CHECK ALL OF PROGRAM MOVED
E21F   27 10                27 R@V             BEQ    WORM3
E221   A6 00                A6 D00             LDA A  0,X      GET BYTE TO MOVE
E223   08                   08                 INX
E224   DF 38                DF D@Q             STX    ACIANO   SAVE SOURCE BYTE ADDRESS
E226   DE 3E                DE D@P             LDX    BFRSZE   GET DESTINATION ADDRESS
E228   A7 00                A7 D00             STA A  0,X      STORE BYTE AT DESTINATION

            COLUMN   1         2         3         4         5         6         7
            1234567890123456789012345678901234567890123456789012345678901234567890123456789012
```

PROGRAM WORM   ASSEMBLY LISTING

```
ADDR  OBJECT       SPHERE          MOTOROLA

E22A  08             08                    INX
E22B  DF 3E          DF D@P                STX    BFRSZE
E22D  DE 38          DE D@Q                LDX    ACIANO    GET SOURCE ADDRESS
E22F  20 EC          20 R@U                BRA    WORM2
E231  DE 3C        V DE D@O        WORM3   LDX    BFRPTR    GET ADDRESS TO START AT
E233  6E 00          6E D00                JMP    0,X       START WORM CRAWLING
E235  20 55        D 20 R@X        WORME1  BRA    WORME
E237  8D DB        Z 8D R@A        WORM4A  BSR    WORM0     PUT WORM START ADDR IN STACK
                                   ****************************************
                                   * WORM PROGRAM (INDEPENDENT FROM ABOVE CODE)
                                   ****************************************
E239  8E 0060      W 8E E0060      WORM4   LDS    #$0060    INITIAL STACK LOCATION
E23C  20 4C          20 R@I                BRA    TOP       GET NEW TOP OF WORM ADDRESS
                                   * RELOCATE STACK TO TOP OF WORM
E23E  30           B 30            BEGIN   TSX              STACK REGISTER TO X
E23F  EE 00          EE D00                LDX    0,X       TOP OF WORM TO X
E241  35             35                    TXS              TOP OF WORM TO STACK REGISTER
                                   * START OF INNER LOOP
E242  C6 3A          C6 D3A                LDA B  #58       UPPER PART LENGTH
E244  34           C 34            INNERL  DES              BACKTRACK STACK POINTER
E245  32             32                    PUL A            GET NEXT BYTE TO MOVE
E246  31             31                    INS              STACK TO LOC TO MOVE TO
E247  36             36                    PSH A            MOVE BYTE UP ONE
E248  34             34                    DES              STACK POINTER DOWN ONE BYTE
E249  A1 00          A1 D00                CMP A  0,X       CHECK MOVED BYTE
E24B  26 FE        E 26 R@E        HANGI   BNE    HANGI     HANG IF BAD BYTE
E24D  09             09                    DEX
E24E  5A             5A                    DEC B            LOOP UNTIL UPPER PART MOVED
E24F  26 F3          26 R@C                BNE    INNERL
E251  86             86                    FCB    $86       DUMMY 2-BYTE INSTRUCTION
                                   * START OF OUTER LOOP
E252  C6 19          C6 D19                LDA B  #25       LOWER PART LENGTH
E254  34           F 34            OUTERL  DES              BACKTRACK STACK POINTER
E255  32             32                    PUL A            GET NEXT BYTE TO MOVE
E256  31             31                    INS              STACK TO LOC TO MOVE TO
E257  36             36                    PSH A            MOVE BYTE UP ONE
E258  34             34                    DES              STACK POINTER DOWN ONE BYTE
E259  A1 00          A1 D00                CMP A  0,X       CHECK MOVED BYTE
E25B  26 FE        H 26 R@H        HANGO   BNE    HANGO     HANG IF BAD BYTE
```

```
COLUMN    1         2         3         4         5         6         7
123456789012345678901234567890123456789012345678901234567890123456789012
```

PROGRAM WORM   ASSEMBLY LISTING

```
ADDR  OBJECT        SPHERE               MOTOROLA

E25D  09               09                      DEX
E25E  5A               5A                      DEC B              LOOP UNTIL LOWER PART MOVED
E25F  26 F3            26 R@F                  BNE    OUTERL

                                    * PRINT BOTTOM ADDRESS OF WORM

E261  8D 02            8D R@J                  BSR    PRINT       PRINT ADDRESS ON SCREEN
E263  20 25            20 R@I                  BRA    TOP         RELOCATE STACK

E265  DF 00         J  DF D00       PRINT      STX    $00         SAVE ADDRESS TO PRINT
E267  FE FC38          FE E@Y                  LDX    SCBEG       GET SCREEN ADDRESS
E26A  96 00            96 D00                  LDA A  $00         PRINT FIRST BYTE OF ADDRESS
E26C  8D 02            8D R@K                  BSR    CNVT
E26E  96 01            96 D01                  LDA A  $01         PRINT SECOND BYTE OF ADDRESS
E270  16            K  16           CNVT       TAB                SAVE BYTE TO PRINT
E271  47               47                      ASR A              MOVE LEFT DIGIT RIGHT
E272  47               47                      ASR A
E273  47               47                      ASR A
E274  47               47                      ASR A
E275  8D 01            8D R@L                  BSR    PNT         CONVERT LEFT DIGIT TO CHAR
E277  17               17                      TBA                CONVERT RIGHT DIGIT TO CHAR
E278  84 0F         L  84 D0F       PNT        AND A  #$0F        ISOLATE DIGIT TO PRINT
E27A  8B 30            8B D30                  ADD A  #$30        ADD CHAR BIAS
E27C  81 3A            81 D3A                  CMP A  #$3A        CHECK FOR A TO F
E27E  2D 02            2D R@M                  BLT    PNT1
E280  8B 07            8B D07                  ADD A  #$07        ADD A TO F EXTRA BIAS
E282  A7 00         M  A7 D00       PNT1       STA A  0,X         PRINT THE DIGIT
E284  08               08                      INX                NEXT PRINT POSITION

                                    * COVER TRACKS WITH SWI INSTRUCTIONS

E285  86 3F            86 D3F                  LDA A  #$3F        SOFTWARE INTERRUPT CODE
E287  36               36                      PSH A
E288  31               31                      INS                RESTORE STACK POINTER
E289  39               39                      RTS

                                    * PUT NEW TOP OF WORM ADDRESS IN STACK

E28A  8D B2         I  8D R@B       TOP        BSR    BEGIN       NEW ADDRESS IN STACK

E28C  39            X  39           WORME      RTS                RETURN TO EXT MONITOR IF NOT WORM

                    END                        END

                    COLUMN    1         2         3         4         5         6         7
                    123456789012345678901234567890123456789012345678901234567890123456789012
```

```
                        PROGRAM CRTOUT   ASSEMBLY LISTING
LINE   ADDR  OBJECT       SPHERE             MOTOROLA

                                                NAM     CRTOUT

                          =      200            ORG     $200

                          C=     1C      CSRPTR EQU     $1C        SCREEN CURSOR LOCATION
                          D=     1D      CSRPT1 EQU     $1D
                          J=     58      CRTBEG EQU     $58        START ADDR OF CRT MEMORY
                          K=     5A      CRTEND EQU     $5A        LAST ADDR OF CRT MEMORY
                          L=     5C      LINSIZ EQU     $5C        LENGTH OF CRT LINE
                          M=     5E      SCRLFG EQU     $5E        SCROLL FLAG FOR CRT
                          N=     60      LNCNTR EQU     $60        POSITION ON CRT LINE
                          Q=     66      TEMPA  EQU     $66        TEMP FOR A REGISTER IN PSHX
                          ^=     70      SCRBEG EQU     $70        VIDEO SCREEN START LOCATION
                                         TEMPX  EQU     $84        TEMP FOR X REGISTER IN PSHX
                                         TEMPX1 EQU     $85
                                         TEMPX2 EQU     $86        TEMP FOR RETURN ADDR IN PSHX
                                         TEMPX3 EQU     $87

                          (=     FC38    SCBEG  EQU     $FC38      SCREEN START LOC FROM SPHERE
                          &=     FD47    LASTLN EQU     $FD47      LAST SCREEN LINE POSITION

                                         ***************************************************
                                         *
                                         * CRT DRIVER INITIALIZATION
                                         *
                                         ***************************************************

       0200  4F           RA 4F          SCINIT CLR A              SET LINE POSITION
       0201  97 60           97 D@N             STA A   LNCNTR
       0203  C6 20           C6 D20             LDA B   #$20       SET SCROLL FLAG
       0205  D7 5E           D7 D@M             STA B   SCRLFG
       0207  FE FC38         FE E@(             LDX     SCBEG      SET SCREEN START ADDRESS
       020A  DF 70           DF D@^             STX     SCRBEG
       020C  FE FD47         FE E@&             LDX     LASTLN     SET SCREEN END ADDRESS
       020F  DF 5A           DF D@K             STX     CRTEND
       0211  09           RD 09          SCLOOP DEX                NEXT SCREEN LOC TO BLANK
       0212  E7 00           E7 D00             STA B   0,X
       0214  9C 70           9C D@^             CPX     SCRBEG     TEST ENTIRE SCREEN BLANKED
       0216  26 F9           26 R@RD            BNE     SCLOOP
       0218  B6 FD63         B6 EFD63           LDA A   $FD63      SET LINE LENGTH
       021B  97 5C           97 D@L             STA A   LINSIZ
       021D  39              39                 RTS


                                         ***************************************************
                                         *
                                         * INDEPENDENT CRT DRIVER
                                         *
                                         ***************************************************


                           COLUMN    1         2         3         4         5         6         7
                           12345678901234567890123456789012345678901234567890123456789012345678 2
```

PROGRAM CRTOUT   ASSEMBLY LISTING

```
ADDR OBJECT     SPHERE        MOTOROLA

021E 36         SA 36         CRTOUT PSH A             SAVE REGISTERS
021F 37            37                PSH B
0220 8D 6B         8D R@RB           BSR    PSHX
0222 DE 1C         DE D@C            LDX    CSRPTR     GET CURSOR ADDRESS
0224 8D 07         8D R@SB           BSR    CRT000     PRINT THE CHARACTER
0226 DF 1C         DF D@C            STX    CSRPTR     SAVE NEW CURSOR ADDRESS
0228 8D 77         8D R@RC           BSR    PULX       RESTORE REGISTERS
022A 33            33                PUL B
022B 32            32                PUL A
022C 39            39                RTS
                              *
                              * (HT) HORIZONTAL TAB 8 SPACES
                              *
022D 81 09      SB 81 D09     CRT000 CMP A  #$09       CHECK FOR HORIZONTAL TAB
022F 26 3E         26 R@TC           BNE    CRT080
0231 96 1D         96 D@D            LDA A  CSRPT1     INCREMENT LINE POS TO MULT OF 8
0233 8A 07         8A D07            ORA A  #7
0235 97 1D         97 D@D            STA A  CSRPT1
0237 96 60         96 D@N            LDA A  LNCNTR     INCREMENT CURSOR POS TO MULT OF 8
0239 8A 07         8A D07            ORA A  #7
023B 97 60         97 D@N            STA A  LNCNTR
023D DE 1C         DE D@C            LDX    CSRPTR     GET CURRENT CURSOR POSITION
023F D6 5C         D6 D@L            LDA B  LINSIZ     GET LINE LENGTH
0241 5A            5A                DEC B             LINE LENGTH - 1
0242 11            11                CBA               CHECK FOR END OF LINE
0243 20 0E         20 R@SD           BRA    CRT011     INCREMENT CURSOR POSITION


                              *
                              * NORMAL ASCII CHARACTER
                              *
0245 85 E0      SC 85 DE0     CRT010 BIT A  #$E0       TEST FOR CONTROL CHARACTER
0247 27 19         27 R@TH           BEQ    CRT110
0249 A7 00         A7 D00            STA A  0,X        STORE THE CHARACTER
024B 08            08                INX               NEXT CHARACTER POSITION
024C 7C 0060       7C E@N            INC    LNCNTR     INCREMENT POSITION IN LINE
024F 96 5C         96 D@L            LDA A  LINSIZ     CHECK FOR END OF LINE
0251 91 60         91 D@N            CMP A  LNCNTR
0253 26 0C      SD 26 R@SE    CRT011 BNE    CRT012
0255 DF 1C         DF D@C            STX    CSRPTR     CURSOR IN SYNC W.TH LNCNTR
0257 86 0D      TB 86 D0D     CRT015 LDA A  #$0D       PRINT CARRIAGE RETURN
0259 8D C3         8D R@SA           BSR    CRTOUT
025B 86 0A         86 D0A            LDA A  #$0A       PRINT LINE FEED
025D 8D BF         8D R@SA           BSR    CRTOUT
025F DE 1C         DE D@C            LDX    CSRPTR     GET NEW CURSOR ADDRESS
0261 39         SE 39         CRT012 RTS
                              *
                              * OTHER CONTROL CHARACTERS
                              *
0262 8A 40      TH 8A D40     CRT110 ORA A  #$40       ADD LETTERS BIAS TO CONTROL

                COLUMN    1         2         3         4         5         6         7
                123456789012345678901234567890123456789012345678901234567890123456789012
```

PROGRAM CRTOUT . ASSEMBLY LISTING

```
ADDR  OBJECT      SPHERE          MOTOROLA

0264  36             36                  PSH A               SAVE CONVERTED CONTROL CHAR
0265  86 5E          86 D5E              LDA A   #$5E        PRINT UP ARROW
0267  8D B5          8D R@SA             BSR     CRTOUT
0269  32             32                  PUL A               PRINT CONVERTED CONTROL CHAR
026A  8D B2          8D R@SA             BSR     CRTOUT
026C  DE 1C          DE D@C              LDX     CSRPTR      GET CURSOR LOCATION
026E  39             39                  RTS

                                  *
                                  * (VT) VERTICAL TAB
                                  *
026F  81 0B       TC 81 D0B       CRT080 CMP A   #$0B        CHECK FOR VERTICAL TAB
0271  26 10          26 R@SQ             BNE     CRT070
0273  86 13          86 D13              LDA A   #$13        GET DOWN ARROW CHARACTER
0275  C6 04          C6 D04              LDA B   #4          4 LINE VERTICAL TAB
0277  8D A5       TD 8D R@SA      CRT081 BSR     CRTOUT      PRINT VERTICAL TAB
0279  5A             5A                  DEC B
027A  26 FB          26 R@TD             BNE     CRT081      LOOP UNTIL DONE
027C  86 0D          86 D0D              LDA A   #$0D        PRINT CARRIAGE RETURN
027E  8D 9E          8D R@SA             BSR     CRTOUT
0280  DE 1C          DE D@C              LDX     CSRPTR      GET NEW CURSOR ADDRESS
0282  39             39                  RTS
                                  *
                                  * (ESC) PRINT INT. CR, CR, LF
                                  *
0283  81 1B       SQ 81 D1B       CRT070 CMP A   #$1B        CHECK FOR ESC CHAR
0285  26 2E          26 R@SI             BNE     CRT030
0287  86 60          86 D60              LDA A   #$60        PRINT INTERNAL CR
0289  8D 93          8D R@SA             BSR     CRTOUT
028B  20 CA          20 R@TB             BRA     CRT015      PRINT CR, LF

028D  97 66       RB 97 D@Q       PSHX   STA A   TEMPA       SAVE A REGISTER
028F  DF 84          DF D84              STX     TEMPX       STORE X FOR STACKING
0291  32             32                  PUL A               GET HIGH RETURN ADDR BYTE
0292  97 86          97 D86              STA A   TEMPX2
0294  32             32                  PUL A               GET LOW RETURN ADDR BYTE
0295  97 87          97 D87              STA A   TEMPX3
0297  96 85          96 D85              LDA A   TEMPX1      GET LOW X REGISTER BYTE
0299  36             36                  PSH A
029A  96 84          96 D84              LDA A   TEMPX       GET HIGH X REGISTER BYTE
029C  36             36                  PSH A
029D  96 87          96 D87              LDA A   TEMPX3      GET LOW RETURN ADDR BYTE
029F  20 0B          20 R@UJ             BRA     PULX1

02A1  97 66       RC 97 D@Q       PULX   STA A   TEMPA       SAVE A REGISTER
02A3  32             32                  PUL A               GET HIGH RETURN ADDR BYTE
02A4  97 86          97 D86              STA A   TEMPX2
02A6  32             32                  PUL A               GET LOW RETURN ADDR BYTE
02A7  30             30                  TSX                 GET STACK POINTER IN X
02A8  EE 00          EE D00              LDX     0 X         GET STACKED X REGISTER VALUE
```

```
COLUMN   1         2         3         4         5         6         7
123456789012345678901234567890123456789012345678901234567890123456789012
```

PROGRAM CRTOUT   ASSEMBLY LISTING

```
LINE   ADDR  OBJECT      SPHERE           MOTOROLA

       02AA  31             31                 INS              CLEAR X REG VALUE FROM STACK
       02AB  31             31                 INS
       02AC  36          UJ 36          PULX1  PSH A            STACK LOW RETURN ADDR BYTE
       02AD  96 86          96 D86             LDA A  TEMPX2    STACK HIGH RETURN ADDR BYTE
       02AF  36             36                 PSH A
       02B0  96 66          96 D@Q             LDA A  TEMPA     RESTORE A REGISTER
       02B2  39             39                 RTS

       02B3  20 90       UH 20 R@SC     CRT016 BRA    CRT010

                                        *
                                        * (LF) LINE FEED
                                        *
       02B5  81 0A       SI 81 D0A      CRT030 CMP A  #$0A      CHECK FOR LINE FEED CHAR
       02B7  26 4C          26 R@SF            BNE    CRT020
       02B9  D6 60          D6 D@N             LDA B  LNCNTR    GET POSITION IN LINE
       02BB  37             37                 PSH B            SAVE POSITION IN LINE
       02BC  50             50                 NEG B
       02BD  DB 5C          DB D@L             ADD B  LINSIZ    SUBT IT FROM LINE LENGTH
       02BF  08          SJ 08          CRT031 INX              INCREMENT TO END OF LINE
       02C0  5A             5A                 DEC B
       02C1  26 FC          26 R@SJ            BNE    CRT031
       02C3  9C 5A          9C D@K             CPX    CRTEND    CHECK FOR SCREEN END
       02C5  26 25          26 R@SL            BNE    CRT033
       02C7  DE 58          DE D@J             LDX    CRTBEG
       02C9  7D 005E        7D E@M             TST    SCRLFG    BRANCH IF WRAP AROUND
       02CC  27 1E          27 R@SL            BEQ    CRT033
                                        *
                                        *
       02CE  DF 84       TR DF D84      CRT32C STX    TEMPX     SAVE CURRENT MOVE TO LOC
       02D0  D6 5C          D6 D@L             LDA B  LINSIZ    GET LINE LENGTH
       02D2  4F             4F                 CLR A            CLEAR UPPER LINE LENGTH BYTE
       02D3  DB 85          DB D85             ADD B  TEMPX1    ADD LINE LENGTH TO MOVE LOC
       02D5  99 84          99 D84             ADC A  TEMPX
       02D7  97 86          97 D86             STA A  TEMPX2    SAVE MOVE FROM LOC
       02D9  D7 87          D7 D87             STA B  TEMPX3
       02DB  DE 86          DE D86             LDX    TEMPX2    GET MOVE FROM LOC
       02DD  9C 5A          9C D@K             CPX    CRTEND    CHECK FOR LAST MOVE LOC
       02DF  27 09          27 R@SP            BEQ    CRT32D
       02E1  A6 00          A6 D00             LDA A  0,X       GET BYTE TO MOVE
       02E3  DE 84          DE D84             LDX    TEMPX     RESTORE MOVE TO LOC
       02E5  A7 00          A7 D00             STA A  0,X       MOVE CHAR ONE LINE BACK
       02E7  08             08                 INX              NEXT LOCATION TO MOVE
       02E8  20 E4          20 R@TR            BRA    CRT32C
       02EA  DE 84       SP DE D84      CRT32D LDX    TEMPX     GET MOVE TO LOC
                                        *
       02EC  DF 84       SL DF D84      CRT033 STX    TEMPX     SAVE CURSOR LOCATION
       02EE  D6 5C          D6 D@L             LDA B  LINSIZ    GET LINE LENGTH
       02F0  86 20          86 D20             LDA A  #$20      GET SPACE FOR BLANKING
       02F2  A7 00       SM A7 D00      CRT034 STA A  0,X       BLANK CURRENT SCREEN POSITION

                         COLUMN    1         2         3         4         5         6         7
                         1234567890123456789012345678901234567890123456789012345678901234567890012
```

PROGRAM CRTOUT ASSEMBLY LISTING

```
ADDR  OBJECT       SPHERE           MOTOROLA

02F4  08               08                   INX               NEXT SCREEN POSITION
02F5  5A               5A                   DEC B
02F6  26 FA            26 R@SM              BNE    CRT034     LOOP UNTIL LINE BLANKED
02F8  DE 84            DE D84               LDX    TEMPX      RESTORE CURSOR LOCATION
                                     *
02FA  33               33                   PUL B             RESTORE ORIGINAL POS IN LINE
02FB  5D               5D                   TST B             QUIT IF AT START OF LINE
02FC  27 04            27 R@SO              BEQ    CRT036
02FE  08            SN 08            CRT035 INX               NEXT POSITION IN LINE
02FF  5A               5A                   DEC B
0300  26 FC            26 R@SN              BNE    CRT035     LOOP UNTIL ORIGINAL POS IN LINE
0302  39            SO 39            CRT036 RTS

0303  20 AE         UD 20 R@UH       CRT014 BRA    CRT016
                                     *
                                     * (CR) CARRIAGE RETURN
                                     *
0305  81 0D         SF 81 D0D        CRT020 CMP A  #$0D       CHECK FOR CARRIAGE RETURN
0307  26 0C            26 R@SR              BNE    CRT037
0309  7D 0060          7D E@N               TST    LNCNTR     CHECK ALREADY AT LINE START
030C  27 06            27 R@SH              BEQ    CRT022
030E  09            SG 09            CRT021 DEX               BACKSPACE ONE LINE CHAR
030F  7A 0060          7A E@N               DEC    LNCNTR     DECREMENT POSITION ON LINE
0312  26 FA            26 R@SG              BNE    CRT021     LOOP UNTIL START OF LINE
0314  39            SH 39            CRT022 RTS
                                     *
                                     * (DC3) DOWN ONE LINE
                                     *
0315  81 13         SR 81 D13        CRT037 CMP A  #$13       CHECK FOR DC3
0317  26 0F            26 R@SY              BNE    CRT050
0319  D6 5C            D6 D@L               LDA B  LINSIZ     GET LINE LENGTH
031B  08            SS 08            CRT038 INX               NEXT LINE POSITION
031C  9C 5A            9C D@K               CPX    CRTEND     CHECK FOR SCREEN BOTTOM
031E  26 02            26 R@ST              BNE    CRT039
0320  DE 58            DE D@J               LDX    CRTBEG     CYCLE TO TOP, IF BOTTOM
0322  5A            ST 5A            CRT039 DEC B             LOOP FOR LINE LENGTH
0323  26 F6            26 R@SS              BNE    CRT038
0325  39               39                   RTS

0326  20 DB         UC 20 R@UD       CRT013 BRA    CRT014
                                     *
                                     * (CLEAR) CLEAR SCREEN
                                     *
0328  81 18         SY 81 D18        CRT050 CMP A  #$18       CHECK FOR CLEAR CHAR
032A  26 0C            26 R@TA              BNE    CRT060
032C  86 20            86 D20               LDA A  #$20       SPACE FOR CLEARING SCREEN
032E  A7 00         SZ A7 D00        CRT051 STA A  0 X        STORE SPACE AT SCREEN POSITION
0330  08               08                   INX               NEXT POSITION
0331  9C 5A            9C D@K               CPX    CRTEND     CHECK FOR END OF SCREEN
0333  26 F9            26 R@SZ              BNE    CRT051
```

```
COLUMN    1         2         3         4         5         6         7
1234567890123456789012345678901234567890123456789012345678901234567890I2
```

PROGRAM CRTOUT   ASSEMBLY LISTING

```
ADDR OBJECT      SPHERE        MOTOROLA

0335 DE 1C        DE D@C              LDX    CSRPTR   GET CURSOR LOCATION
0337 39           39                  RTS
                               *
                               * (HOME) HOME CURSOR
                               *
0338 81 0C     TA 81 D0C       CRT061 CMP A  #$0C     CHECK FOR HOME CHAR
033A 26 06        26 R@TE             BNE    CRT100
033C DE 58        DE D@J              LDX    CRTBEG   GET SCREEN START LOCATION
033E 7F 0060      7F E@N              CLR    LNCNTR   POSITION COUNTER AT LINE START
0341 39           39                  RTS
                               *
                               * (DC1) UP ONE LINE
                               *
0342 81 11     TE 81 D11       CRT100 CMP A  #$11     CHECK FOR DC1 CHAR
0344 26 0D        26 R@TI             BNE    CRT120
0346 D6 5C        D6 D@L              LDA B  LINSIZ   GET LINE LENGTH
0348 9C 58     TF 9C D@J       CRT101 CPX    CRTBEG   CHECK FOR SCREEN TOP
034A 26 02        26 R@TG             BNE    CRT102
034C DE 5A        DE D@K              LDX    CRTEND   CYCLE TO SCREEN BOTTOM, IF TOP
034E 09        TG 09           CRT102 DEX             BACK ONE CHAR POSITION
034F 5A           5A                  DEC B           DECREMENT POSITION COUNTER
0350 26 F6        26 R@TF             BNE    CRT101
0352 39           39                  RTS
                               *
                               * (RS) SET OPTION TO SCROLL
                               *
0353 81 1E     TI 81 D1E       CRT120 CMP A  #$1E     CHECK FOR RS CHARACTER
0355 26 03        26 R@TJ             BNE    CRT130
0357 97 5E        97 D@M              STA A  SCRLFG
0359 39           39                  RTS
                               *
                               * (VS) SET OPTION TO WRAP AROUND
                               *
035A 81 1F     TJ 81 D1F       CRT130 CMP A  #$1F     CHECK FOR VS CHARACTER
035C 26 04        26 R@TK             BNE    CRT140
035E 7F 005E      7F E@M              CLR    SCRLFG   CLEAR SCROLL FLAG
0361 39           39                  RTS
                               *
                               * (DC2) RIGHT ONE CHARACTER
                               *
0362 81 12     TK 81 D12       CRT140 CMP A  #$12     CHECK FOR DC2 CHARACTER
0364 26 14        26 R@TN             BNE    CRT045
0366 08           08                  INX             INCREMENT CURSOR POSITION
0367 7C 0060      7C E@N              INC    LNCNTR   INCREMENT LINE POSITION
036A 96 5C        96 D@L              LDA A  LINSIZ   GET LINE LENGTH
036C 91 60        91 D@N              CMP A  LNCNTR   CHECK FOR LINE END
036E 26 09        26 R@TL             BNE    CRT141
0370 7F 0060      7F E@N              CLR    LNCNTR   SET TO LINE POSITION 0 (START)
0373 9C 5A        9C D@K              CPX    CRTEND   CHECK FOR END OF SCREEN
0375 26 02        26 R@TL             BNE    CRT141

COLUMN    1         2         3         4         5         6         7
1234567890123456789012345678901234567890123456789012345678901234567890 12
```

PROGRAM CRTOUT   ASSEMBLY LISTING

```
ADDR OBJECT      SPHERE              MOTOROLA

0377 DE 58          DE D@J                LDX    CRTBEG   CYCLE TO START, IF AT END
0379 39          TL 39            CRT141 RTS
                                  *
                                  * (DEL) BACKSPACE AND DELETE (NORMALLY USES CRT040)
                                  * (BS) BACKSPACE AND DELETE
                                  *
037A 81 7F       TN 81 D7F        CRT045 CMP A  #$7F     TEST FOR DEL
                                  * USE FOLLOWING INSTRUCTION TO TREAT DEL AS REGULAR CHAR
037C 27 A8          27 R@UC              BEQ    CRT013
                                  * USE FOLLOWING INSTRUCTION TO TREAT DEL AS BACKSPACE
                                  *      BEQ    CRT040
037E 81 08          81 D08               CMP A  #$08     TEST FOR BACKSPACE
0380 26 07          26 R@SV              BNE    CRT041
0382 8D 09       SU 8D R@TM       CRT040 BSR    CRT044   BACK UP ONE CHARACTER
0384 86 20          86 D20               LDA A  #$20     BLANK OUT CURRENT CHARACTER
0386 A7 00          A7 D00               STA A  0,X
0388 39             39                   RTS
                                  *
                                  * (DC4) BACK UP ONE CHARACTER
                                  *
0389 81 14       SV 81 D14        CRT041 CMP A  #$14     CHECK FOR BACK ONE CHARACTER
038B 26 99          26 R@UC              BNE    CRT013
038D 9C 58       TM 9C D@J        CRT044 CPX    CRTBEG   CHECK FOR START OF SCREEN
038F 26 02          26 R@SW              BNE    CRT042
0391 DE 5A          DE D@K               LDX    CRTEND   USE END OF SCREEN IF AT START
0393 96 60       SW 96 D@N        CRT042 LDA A  LNCNTR   CHECK FOR START OF LINE
0395 26 04          26 R@SX              BNE    CRT043
0397 96 5C          96 D@L               LDA A  LINSIZ   RESTART AT END OF LINE
0399 97 60          97 D@N               STA A  LNCNTR
039B 7A 0060     SX 7A E@N        CRT043 DEC    LNCNTR   DECREMENT POSITION ON LINE
039E 09             09                   DEX
039F 39          PA 39            CRTEX  RTS

                 END                     END

                 COLUMN    1         2         3         4         5         6         7
                 1234567890123456789012345678901234567890123456789012345678901234567890l2
```

SUBSET OF ASCII to PACKED BASE 40 and Back

Two complementary programs which follow are based on an algorithm which was written up in INTERFACE AGE magazine OCT. 1980, PP. 80-84 by David Veldof. The idea is to select a subset of Ascii code, in this case including capital letters, numbers zero thru nine, and four other characters (space, $, period and minus sign). The Ascii, three bytes at a time, is packed into 16 bit words (see figure 1) giving a true 50% savings in storage space and possibly shortening I/O time especially if a slow peripheral device is used. I am currently using these routines to store 192 bytes of data in each disk sector which usually carries 128 bytes.

The most involved part of the programs is the handling of the Ascii in the event that the number of bytes is not exactly divisible by three. Also, the routines have been written to search for an end character (* which is Hex 2A) so as to allow any lenth Ascii file to be handled.

When less than three bytes remain to be converted, the one or two are just converted to base 40 but passed thru unpacked. Also, the first byte, if there are two left over, is complemented. These changes of format allow the unpacking program to know that the last few bytes before the end character must be unpacked differently. The complement was done so that no combination of two base 40 bytes could exist with the same value as some packed 16 bit combination.

There are many possible changes which can customize these routines. One could easily substitute characters for the four miscellaneous ones. The dollar sign can usually be part of the application program instead of being stored in the data file. The comma might thus be substituted for the $. Of course, the choice of end character is up to the user; just let the program take this from ram instead of loading it immediate. Each application could specify the end character in that way. Fig. 4 shows an alternate scheme of allocating the base 40 character set but this "CAN CODE" only gets 39 characters instead of 40 because it uses zero differently. My version, by using zero for a space, makes some use of the 40th position.

The Base 40 to Ascii program calls the DIVIDE routine which sits in the V3N and most other proms which our users have. If it is no longer available to you in prom, just tack it at the end of the 16 bits to Ascii module.

Now to try the programs out:

Set up any areas for buffer 1 and 2 when you type in the programs.

```
A- place into buffer 2  00   temp
                        53   S
                        50   P
                        48   H
  + Load                45   E
    Ascii To            52   R
    BASE 40 at 200      45   E
                        2A   end of file
```

B-do 0200 control J
C-find the packed code in buffer 1 : 79 48 22 15 2A

D- leave the packed code (or save it to tape or disk for later retrieval)

E- load the packed base 40 to Ascii program

F- do 200 control J   NOTE THAT EITHER PROGRAM MAY BE LOADED ANYWHERE IN RAM OR ROM BECAUSE RELATIVE ADDRESSING IS USED.

G_ You should find the original Ascii code back in buffer 2.

Special note: It should be possible to pass a larger number of characters than the 40 if this is needed. Only 40 different ones will be packed but some infrequently used but very necessary characters could be encoded and passed thru imbedded in the packed code. The unpacking program would have to be similarly modified to test for these characters. It should be possible, but probably not really necessary, to have an extended version of these routines which would handle the entire Ascii character set but only pack the forty most commonly used characters. This would slightly reduce the density of the packed code while removing the limitations.

Please let me know if you find these routines useful.

Jeff

P... ALGORITHM

The following section summarizes the steps necessary to pack Base 40 characters three to a word.

1. Take the Base 40 character H and shift it three places to the left (multiply it by 8) and save this partial result.
2. Shift H two more places to the left (multiply it by 32) and add the saved partial result (equivalent to H*40). Save this as the sum.
3. Take M and add it to the sum (from step 2). Shift this quantity over three places to the left (multiply by 8) and save the partial result. Shift the quantity two more places to the left (multiply by 32) and add the partial result to it (M*40). Save this in sum.
4. Take the sum and add L to it. This gives you three packed Base 40 characters in one word.
5. Repeat the whole procedure from step 1 for the next group of three Base 40 characters.

Fig. 1

BASE 40 GETS 40

Base 40 Character Set

Ascii Character	Base 40 Equivalent
Space	0
A - Z	1 - 26
$	27
[illegible]	28
-	29
0 - 9	30 - 39

Figure 1. Characters normally represented in Base 40.

Fig. 3

U... ALGORITHM

To summarize the steps necessary to unpack Base 40 characters:

1. Divide the packed characters (HML) by 40^2. The quotient will be the unpacked high order.
2. Divide the remainder (from step 1) by 40. The quotient will be the unpacked middle character and the remainder will be the unpacked lower-order character.

We have now shown how to pack and unpack Ascii characters, assuming they are already expressed in their Base 40 equivalents. But how do we get them in their Base 40 equivalents?

Fig. 2

can code conversion

HEX	CAN	HEX	CAN	HEX	CAN
0		0		0	
640	A	28	A	1	A
C80	B	50	B	2	B
12C0	C	78	C	3	C
1900	D	A0	D	4	D
1F40	E	C8	E	5	E
2580	F	F0	F	6	F
~~[illegible]~~ 2BC0	G	118	G	7	G
3200	H	140	H	8	H
3840	I	168	I	9	I
3E80	J	190	J	A	J
44C0	K	1B8	K	B	K
4B00	L	1E0	L	C	L
5140	M	208	M	D	M
5780	N	230	N	E	N
5DC0	O	258	O	F	O
6400	P	280	P	10	P
6A40	Q	2A8	Q	11	Q
7080	R	2D0	R	12	R
76C0	S	2F8	S	13	S
7D00	T	320	T	14	T
8340	U	348	U	15	U
8980	V	370	V	16	V
8FC0	W	398	W	17	W
9600	X	3C0	X	18	X
9C40	Y	3E8	Y	19	Y
A280	Z	410	Z	1A	Z
A8C0	0	438	0	1B	0
AF00	1	460	1	1C	1
B540	2	488	2	1D	2
BB80	3	4B0	3	1E	3
C1C0	4	4D8	4	1F	4
C800	5	500	5	20	5
CE40	6	528	6	21	6
D480	7	550	7	22	7
DAC0	8	578	8	23	8
E100	9	5A0	9	24	9
E740	:	5C8	:	25	:
ED80	.	5F0	.	26	.
F3C0	$	618	$	27	$

Note that can code version gets only 34 usable characters

Fig. 4

16 Bits PACKED BASE 40 TO ASCII

```
     Main program
0200 CE LDX #3000     Buffers
0203 DF STX  52
0205 CE LDX #2000
0208 DF STX  50
020A DE LDX  50
020C C6 LDAB#2A    * EOF
020E A6 LDAAX00
0210 11 CBA
0211 26 BNE  05           0218
0213 DE LDX  52
0215 E7 STABX00
0217 39 RTS
0218 A6 LDAAX01
021A 11 CBA
021B 26 BNE  0B           0228
021D A6 LDAAX00
021F 8D BSR  60           0281
0221 DE LDX  52
0223 A7 STAAX00
0225 E7 STABX01
0227 39 RTS
0228 A6 LDAAX02
022A 11 CBA
022B 26 BNE  1C           0249
022D A6 LDAAX00
022F 81 CMPA#FA
0231 23 BLS  16           0249
0233 A6 LDAAX00
0235 43 COMA
0236 8D BSR  49           0281
0238 DE LDX  52
023A A7 STAAX00
023C DE LDX  50
023E A6 LDAAX01
0240 8D BSR  3F           0281
0242 DE LDX  52
0244 A7 STAAX01
0246 E7 STABX02
0248 39 RTS
0249 86 LDAA#06
024B 97 STAA 06
024D 86 LDAA#40   @
024F 97 STAA 07
0251 E6 LDABX00
0253 A6 LDAAX01
0255 08 INX
0256 08 INX
0257 DF STX  50
0259 8D BSR  59           02B4
025B 8D BSR  24           0281
025D DE LDX  52
025F A7 STAAX00
0261 96 LDAA 07
0263 C6 LDAB#28   (
0265 D7 STAB 07
0267 D6 LDAB 06
0269 7F CLR  0006
026C 8D BSR  46           02B4
026E 8D BSR  11           0281
0270 DE LDX  52
0272 A7 STAAX01
0274 96 LDAA 07
0276 8D BSR  09           0281
0278 A7 STAAX02
027A 08 INX
027B 08 INX
027C 08 INX
027D DF STX  52
027F 20 BRA  89           020A Loop
```

Set up Buffers

Handle EOF at bytes 1st, 2nd and 3rd ... EOF

```
     Base 40 to ASCII
0281 81 CMPA#27
0283 2E BGT  2B           02B0
0285 4D TSTA
0286 26 BNE  04           028C
0288 86 LDAA#20
028A 20 BRA  27           02B3
028C 81 CMPA#1B
028E 26 BNE  04           0294
0290 86 LDAA#24   $
0292 20 BRA  1F           02B3
0294 81 CMPA#1C
0296 26 BNE  04           029C
0298 86 LDAA#2E   .
029A 20 BRA  17           02B3
029C 81 CMPA#1D
029E 26 BNE  04           02A4
02A0 86 LDAA#2D   -
02A2 20 BRA  0F           02B3
02A4 81 CMPA#1D
02A6 2E BGT  04           02AC
02A8 8B ADDA#40   @
02AA 20 BRA  07           02B3
02AC 8B ADDA#12
02AE 20 BRA  03           02B3
02B0 7E JMP  0B04   Report illegal character
02B3 39 RTS
02B4 7E JMP  FFAF
```

Access Divide routine in PROM

	Packed		EOF
2000	73 - 81	2215	2A

	ASCII Restored				EOF
3000	53	50	48	4? 3?	2A

3 BYTES ASCII TO 16 BITS BASE 40


```
0200  CE LDX  #2000
0203  DF STX   50
0205  CE LDX  #2FFF
0208  DF STX   52
020A  A6 LDAAX01
020C  81 CMPA#2A    *           0215
020E  26 BNE   05
0210  DE LDX   50
0212  A7 STAAX00
0214  39 RTS
0215  8D BSR   58               026F
0217  A7 STAAX01
0219  A6 LDAAX02
021B  81 CMPA#2A    *
021D  26 BNE   09               0228
021F  E6 LDABX01
0221  DE LDX   50
0223  E7 STABX00
0225  A7 STAAX01
0227  39 RTS
0228  8D BSR   45               026F
022A  A7 STAAX02
022C  A6 LDAAX03
022E  81 CMPA#2A    *
0230  26 BNE   10               0242
0232  E6 LDABX01
0234  A6 LDAAX02
0236  DE LDX   50
0238  53 COMB
0239  E7 STABX00
023B  A7 STAAX01
023D  86 LDAA#2A    *
023F  A7 STAAX02
0241  39 RTS
0242  8D BSR   2B               026F
0244  A7 STAAX03
0246  8D BSR   5A               02A2
0248  A6 LDAAX01
024A  5F CLRB
024B  AB ADDAX02
024D  E9 ADCBX00
024F  E7 STABX00
0251  A7 STAAX01
0253  8D BSR   4D               02A2
0255  A6 LDAAX01
0257  5F CLRB
0258  AB ADDAX03
025A  E9 ADCBX00
025C  DE LDX   50
025E  E7 STABX00
0260  A7 STAAX01
0262  08 INX
0263  08 INX
0264  DF STX   50
0266  DE LDX   52
0268  6F CLR  X03
026A  08 INX
026B  08 INX
026C  08 INX
026D  20 BRA   99               0208
```

```
026F  81 CMPA#20
0271  26 BNE   01               0274
0273  4F CLRA
0274  81 CMPA#24    $
0276  26 BNE   02               027A
0278  86 LDAA#1B
027A  81 CMPA#2E    .
027C  26 BNE   02               0280
027E  86 LDAA#1C
0280  81 CMPA#2D    -
0282  26 BNE   02               0286
0284  86 LDAA#1D
0286  81 CMPA#40    @
0288  2F BLE   08               0292
028A  81 CMPA#5A    Z
028C  2E BGT   04               0292
028E  80 SUBA#40    @
0290  20 BRA   0F               02A1
0292  81 CMPA#2F    /
0294  2F BLE   08               029E
0296  81 CMPA#39    9
0298  2E BGT   04               029E
029A  80 SUBA#12
029C  20 BRA   03               02A1
029E  7E JMP   FE64     TO DEBUG IF illegal character
02A1  39 RTS
```

```
02A2  8D BSR   1D               02C1
02A4  8D BSR   1B               02C1
02A6  8D BSR   19               02C1
02A8  A6 LDAAX00
02AA  97 STAA  06
02AC  A6 LDAAX01
02AE  97 STAA  07
02B0  8D BSR   0F               02C1
02B2  8D BSR   0D               02C1
02B4  A6 LDAAX01
02B6  E6 LDABX00
02B8  9B ADDA  07
02BA  D9 ADCB  06
02BC  A7 STAAX01
02BE  E7 STABX00
02C0  39 RTS
02C1  68 ASL  X01
02C3  A6 LDAAX00
02C5  A9 ADCAX00
02C7  A7 STAAX00
02C9  39 *** RTS
```

DD	53	5?	43	45	52	45	2A

2000	7143	2215	2A

Simple circuit and software replace PROM programmer for 6800-based systems

In the development of software for microprocessor-based systems, debugged software must often be transferred to PROM for later use in the system. It is preferable to write in the system's PROM directly from the system's RAM, instead of copying down the contents of the RAM and then writing them into an external PROM with a PROM programmer. The advantages of direct transfer are realized easily with just two transistors, a flip-flop and a few resistors, which connect a 2708/2704 PROM to the peripheral interface adapter (PIA) used in systems based on the 6800. The required programming can reside in the system monitor or in any other areas designated by the user.

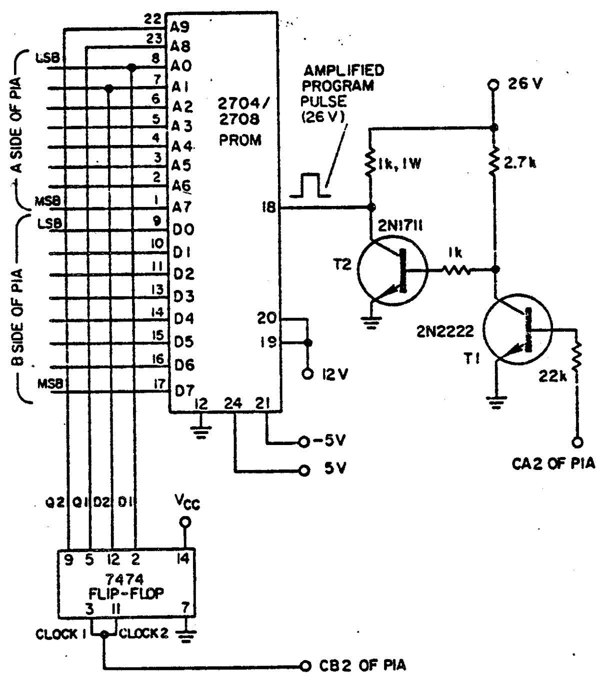
The resident software (see listing) presents the address, data bits and the program pulse to the PROM through a 6820 PIA. Programming waveforms are created exactly as recommended by the PROM manufacturer. After the address and data are set up, one program pulse per address is applied to the PROM's program input (pin 18). One scan through all addresses constitutes a program loop. One hundred program loops are required as per the manufacturer's data sheet.

Before executing the program, the user must supply the starting address of the data source (RAM) and the starting and ending address where data are to be written (PROM), and must account for the data invert/normal option. The program terminates with an RTS instruction, which returns control to the monitor.

Transistors T_1 and T_2 (see schematic) amplify the program pulse transmitted by the 6800 to a 26-V level for the 2708/2704 PROM. The 7474 flip-flop expands the word length of port A and provides ten address bits simultaneously to the PROM. Data bits are presented to the PROM through Port B of the PIA. CA_2 and CB_2 of the PIA are defined as outputs: CA_2 supplies the program pulse, and CB_2 provides the flip-flop clock pulse.

The cost for all the additional hardware is about $3—a negligible amount when compared to the cost of a microprocessor development system. The circuit and software have been tested only with a 6800 microprocessor system and a 2708/2704 PROM. However, any PROM should be programmable using this technique, if appropriate changes are made in the hardware and software.

Victor Mathew and James Thomas, R&D Engineers, OEN India Ltd., Vyttila, P.B. No. 2, Cochin-682 019 Kerala, India.



With only a few components, the 2704/2708 PROM connects to a 6800-system's peripheral interface adapter (PIA) for direct software transfer. The "A side" of the PIA (and the flip-flop) outputs address bits; the "B side" outputs data.

```
                * PIA INITIALISATION *

        1000    4F                      CLR A           :DEFINE A AND B SIDE OF PIA AS OUTPUT
        1001    97 C0                   STA A            CONTROL PORT A SIDE=00C0
        1003    86 FF                   LDA A            CONTROL PORT B SIDE=00C1
        1005    97 80                   STA A            OUTPUT PORT  A SIDE=0080
        1007    86 04                   LDA A            OUTPUT PORT  B SIDE=0081
        1009    97 C0                   STA A
        100B    4F                      CLR A
        100C    97 C1                   STA A
        100E    86 FF                   LDA A
        1010    97 81                   STA A
        1012    86 04                   LDA A
        1014    97 C1                   STA A

                * MAIN *
        1016    5F                      CLR B           :LOOP COUNTER INITIALISE
        1017    DE 00       LOOP2       LDX             :RAM START ADDR.AT 0000/0001
        1019    DF 07                   STX             :PRESENT RAM ADDR.STORED
        101B    DE 02                   LDX             :PROM START ADDR.AT 0002/0003
        101D    DF 09                   STX             :PROM PRESENT ADDR.STORED
        101F    96 09       LOOP1       LDA A           :OUTPUT HIGH BYTE ADDR.OF PROM
        1021    97 80                   STA A
        1023    86 3C                   LDA A
        1025    97 C1                   STA A           :CB2 PULSE GOES HIGH
        1027    8D 35                   BSR             :250 MICROSECOND DELAY SUBROUTINE
        1029    86 34                   LDA A           :CB2 PULSE GOES LOW
        102B    97 C1                   STA A
        102D    96 0A                   LDA A           :OUTPUT LOW BYTE ADDR.OF PROM
        102F    97 80                   STA A
        1031    DE 07                   LDX             :RAM START ADDR.
        1033    A6 00                   LDA A
        1035    7D 0006                 TST             :TEST ADDR.0006 FOR DATA-NORMAL/INVERT
        1038    27 01                   BEQ     NORM    :DATA NORMAL
        103A    43                      COM A           :DATA INVERT
        103B    97 81       NORM        STA A
        103D    8D 1F                   BSR             :250 MICROSECOND DELAY SUBROUTINE
        103F    86 3C                   LDA A           :CA2 PULSE GOES HIGH
        1041    97 C0                   STA A
        1043    8D 26                   BSR             :1MILLISECOND DELAY SUBROUTINE
        1045    86 34                   LDA A           :CA2 PULSE GOES LOW
        1047    97 C0                   STA A
        1049    8D 13                   BSR             :250 MICROSECOND DELAY SUBROUTINE
        104B    08                      INX             :INCREMENT RAM ADDR.
        104C    DF 07                   STX
        104E    DE 09                   LDX
        1050    08                      INX             :INCREMENT ROM ADDR.
        1051    DF 09                   STX
        1053    09                      DEX
        1054    9C 04                   CPX             :CHECK FOR PROM ADDR.AT 0004/0005
        1056    26 C7                   BNE     LOOP1   :CHECK FOR LOOP END
        1058    5C                      INC B           :INCREMENT LOOP COUNTER
        1059    C1 64                   CMP B           :CHECK FOR 100 LOOPS
        105B    26 BA                   BNE     LOOP2
        105D    39                      RTS

                *250 MICROSECOND DELAY SUBROUTINE*
        105E    7F 000B                 CLR             :INITIALISE COUNTER FOR DELAY
        1061    7C 000B     LOOP3       INC
        1064    96 0B                   LDA A
        1066    81 0F                   CMP A
        1068    26 F7                   BNE     LOOP3
        106A    39                      RTS

                *1 MILLISECOND DELAY SUBROUTINE*
        106B    7F 000B                 CLR             :INITIALISECOUNTER FOR DELAY
        106E    7C 000B     LOOP4       INC
        1071    96 0B                   LDA A
        1073    81 40                   CMP A
        1075    26 F7                   BNE     LOOP4
        1077    39                      RTS
```

The programming required for direct transfer of debugged software from RAM to PROM can reside in the system monitor or in any other areas designated by the user.

Adapting the M6800 processor for automatic telephone dialing

by Moshe Bram
Allied Chemical Corp., Automotive Products Division, Mount Clemens, Mich.

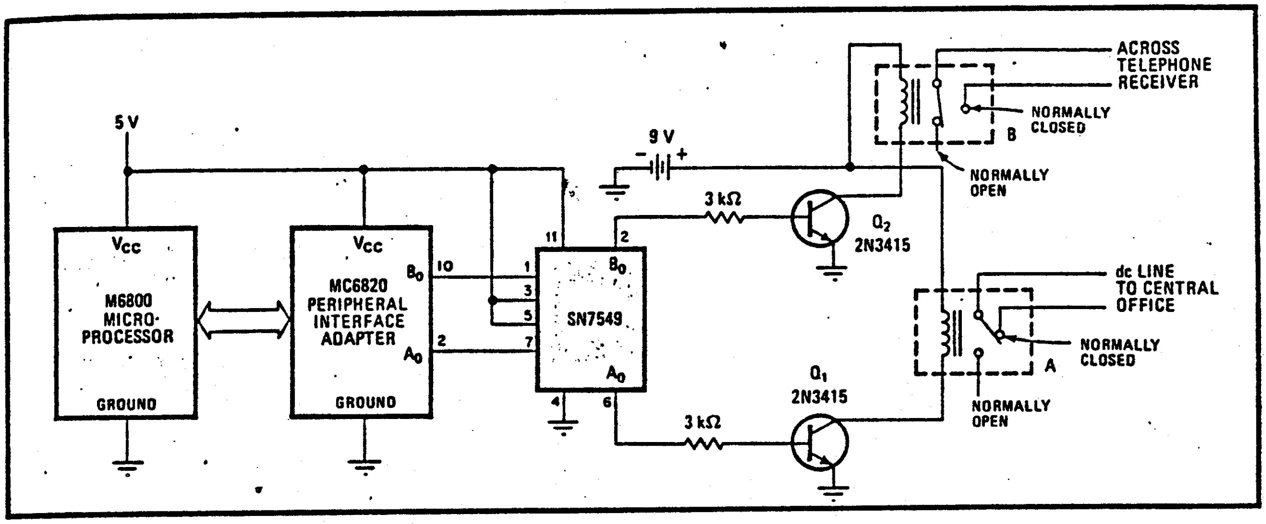
A short program and a simple interface for a rotary-dial telephone enables the well-known M6800 microprocessor to dial stored telephone numbers on command. The versatile microprocessor thus becomes a viable alternative to commercial automatic dialers, which essentially perform only one function and are expensive—chiefly because they are in great demand.

The program for the M6800 is divided into two sections, as shown in the table. The phone numbers are stored in the lower locations of memory, the dialing subroutine in the upper sections. The only limit to the number of phone numbers that can be stored is the amount of memory available to the system. The operation of the entire program is explained by the comments column of the table.

To access any number, the user (or a subroutine) simply initiates the program at, or routes the program to,

M6800 PROGRAM FOR AUTOMATIC DIALING

Location	Object code	Source statement		Comments
0000	C6 X1	LDA B	X1	Load accumulator B with the first digit of phone number (X_1)
0002	BD 01C8	JSR	01C8	Jump to dial subroutine located at address 01C8
0005	C6 X2	LDA B	X2	Load accumulator B with the second digit X_2
0007	BD 01C8	JSR	01C8	Jump to dial subroutine located at address 01C8
000A	C6 X3	LDA B	X3	Load accumulator B with the third digit X_3
000C	BD 01C8	JSR	01C8	Jump to dial subroutine located at address 01C8
000F	C6 X4	LDA B	X4	Load accumulator B with the fourth digit X_4
0011	BD 01C8	JSR	01C8	Jump to dial subroutine located at address 01C8
0014	C6 X5	LDA B	X5	Load accumulator B with the fifth digit X_5
0016	BD 01C8	JSR	01C8	Jump to dial subroutine located at address 01C8
0019	C6 X6	LDA B	X6	Load accumulator B with the sixth digit X_6
001B	BD 01C8	JSR	01C8	Jump to dial subroutine located at address 01C8
001E	C6 X7	LDA B	X7	Load accumulator B with the seventh digit X_7
0020	BD 01C8	JSR	01C8	Jump to dial subroutine located at address 01C8
0023	3F	SWI		End of dialed number
.	.	.		
.	.	.		} Other numbers stored as required
.	.	.		
01C8	86 FF	LDA A	FF	Initial clearing of data direction and control
01CA	B7 8004	STA A	8004	registers of the PIA.
01CD	B7 8005	STA A	8005	
01D0	B7 8006	STA A	8006	
01D3	B7 8007	STA A	8007	
01D6	86 01	LDA A	01	A "HIGH" (1) is loaded into A_0 & B_0 of PIA
01D8	B7 8006	STA A	8006	
01DB	B7 8004	STA A	8004	
01DE	CE 18FF	LDX	09FF	A counter is set allowing A_0 & B_0 to be high (1)
01E1	09	DEX		during the count down.
01E2	26 FD	BNE	FD	
01E4	4F	CLR A		Data line A_0 goes low (0).
01E5	B7 8004	STA A	8004	
01E8	CE 18FF	LDX	09FF	A counter is set allowing A_0 to be low (0)
01EA	09	DEX		during the count down.
01EC	26 FD	BNE	FD	
01EE	5A	DEC B		One cycle of pulse generation has been completed.
01EF	26 E5	BNE	E5	A branch instruction to generate the next pulse
01F1	B7 8006	STA A	8006	cycle is executed. Line B_0 goes low (0) at the end.
01F4	C6 02	LDA B	02	
01F6	CE FFFF	LDX	FFFF	A counter is set allowing a time interval between
01F9	09	DEX		dialed digits.
01FA	26 FD	BNE	FD	
01FC	5A	DEC B		
01FD	26 F7	BNE	F7	
01FF	39	RTS		A return from subroutine instruction is executed
				to load the next digit for dialing.



Command performance. Small program (see table) and simple interface adapt M6800 for automatic number dialing. PIA's output pulses are sent via relay A to central office, while telephone receiver is disabled by relay B to minimize annoying clicking sound in headset.

the location corresponding to the first digit of the number desired. For the sake of efficiency, program command time should be negligible with respect to the actual dialing time. Consequently, the command procedure should keep user interaction to a minimum—that is, it should be unnecessary to depress more than one key of an m·n matrix for each number desired.

Emanating from the 6820 peripheral interface adapter are seven groups of pulses corresponding to the number dialed (this may be extended to 10 groups or more if dialing into other area codes is contemplated). The program ensures that there is a suitable gap between each group of pulses so that the central office can differentiate between pulses belonging to separate digits.

The dial interface is a simple circuit connected between the PIA lines and the central office. It is designed to open the normally closed dc line relay A for each output pulse from the PIA, thus transmitting the digit data to the office. The 7549 serves as a latch and buffer to transfer the PIA's dial-pulse information to switching transistor Q_1, which has actual control of the relay. The output of port B_0 of the 7549 is high during the time the pulses are sent, and relay B is therefore closed in order to disable the headset receiver, minimizing the annoying clicking sounds that are heard in the receiver during dial-out.

The user is cautioned against connecting active-element relays, such as transistors, directly to the line. More likely than not, such an arrangement will require a small amount of power from the dc line to energize the active device, and even a load of only 2 milliamperes will be sufficient to cause trouble at the central office. □

Routine reverses data-storage order

Rao N Bhaskara
Inelec Boumerdes, Algiers, Algeria

Computations such as digital convolutions, FFTs and DFTs require reversal of the order of the stored data when performed off line. Typically, you write the data out into a second memory block, but this approach requires enough extra memory to store the data twice—not a desirable situation in a µC system.

Another approach to the problem involves exchanging the data bytes between the first and last memory locations, then between the second and next-to-last, and so on, as in the accompanying code. Thus, if there are k+1 data bytes in the memory block, complete reversal requires k/2 (if k+1 is odd) or (k+1)/2 (if k+1 is even) exchanges.

Let the first memory location be m and the last m+k. Note that there are k+1 memory locations in the block, containing k+1 data bytes. **EDN**

ENTER:	LXI##m	Index Register loaded with first address
	LSP##m+k−1	SP loaded with last-but-one address
LOADA:	LDAA X	Data loaded into Acc A from the address pointed to by X
	PULB	SP incremented by 1; B loaded from the memory location pointed to by this value of SP
	STAB X	Exchange of data bytes takes place, and SP decremented by 1
	PSHA	
	INX	To get the next address from the top
	DES	To get the next address from the bottom
	CPX##n	Compare X with n, where $n = m + \frac{k+1}{2}$ if k + 1 is even, and $n = m + k/2$ if k + 1 is odd
	BNE LOADA	Repeat the exchange operations until all data bytes used, then
	RTS	Return to main program

Save memory *while reversing blocks of stored data by using this M6800 subroutine.*

Multiplexed-memory technique doubles μP's addressing capacity

Memory-map switching extends an 8-bit μP's storage access to 128k—increasing application possibilities without sacrificing processing efficiency or capabilities.

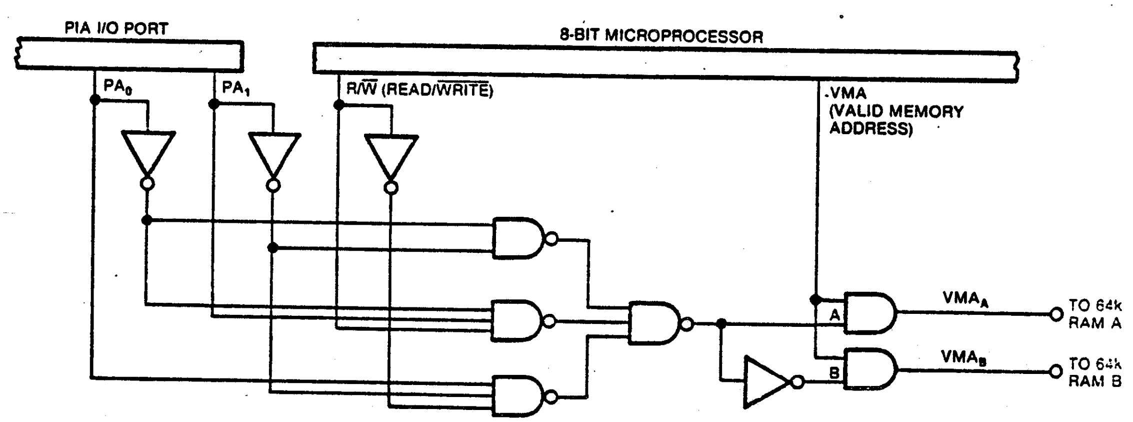
Stephen Strom, Motorola Semiconductor Products Inc

When an 8-bit μP's 64k-byte memory-addressing capability begins to cramp your programming prospects, don't assume that you must necessarily upgrade to a 16-bit device. The software-directed dual memory-mapping approach described here allows you to expand an 8-bit μP's addressing range to 128k bytes by multiplexing two 64k RAMs. This unified hardware/software approach lets you incorporate more processing functions and support more complex and efficient functions in an 8-bit system.

The dual-RAM mapping approach suits both long programs and short ones that need excessive buffer space during execution—two conditions that often occur in applications such as word processors, text editors and RAM/ROM testers. With enlarged memory, you can also overcome the size constraints of a combined operating-system, I/O and control program. In fact, you can assign an entire 64k map to the operating system and control program and reserve the second map for the program buffer. This broad storage allocation permits you to add numerous program features without infringing on buffer space.

Simple concept demands design care

To implement the dual-memory-map approach, merely multiplex an 8-bit μP to two identical 64k RAMs. This seemingly elementary method requires careful consideration of several hardware/software design factors, however. At any instant, for example, the μP can access only one RAM; the entire workspace thus splits into two independent 64k blocks (memory maps). For proper hardware operation, the μP must retain each memory map's identity and correctly transfer control and data between maps. Software must



I/O PORT PA_0	I/O PORT PA_1	READ/$\overline{WRITE}$ ($R/\overline{W}$)	A (ENABLE)	B (ENABLE)	ADDRESS MODE
0	0	X	1	0	1
0	1	1	1	0	2
0	1	0	0	1	2
1	0	1	0	1	3
1	0	0	1	0	3
1	1	X	0	1	4

Fig 1—Decode logic *connects a μP and a PIA to two 64k RAMs and transfers data and instructions under software control. Various logic-state combinations determine the source and destination of code-storing, loading and executing operations.*

Dual mapping overcomes 8-bit μP's memory limit

govern these map transfers and allow the μP to access memory without loss of processing efficiency.

To satisfy these hardware/software design constraints, memory-map multiplexing calls for software-driven decode logic connected between the μP's I/O port and two 64k RAMs (Fig 1). Although this particular hardware configuration relies on a 6800 μP and a 6820 peripheral interface adapter (see box, "6800-μP definitions"), the memory-map switching principles involved apply to most 8-bit μPs with little modification.

In Fig 1, note that the μP's Valid Memory Address (VMA) output, in conjunction with the decode logic, produces RAM-selector signals VMA_A and VMA_B. When VMA_A goes HIGH, the decode logic switches RAM A to the μP; similarly, when VMA_B goes HIGH, RAM B comes under μP control.

By storing data via a properly designated I/O port, the μP selects one of four addressing modes:

- Mode 1—Load, execute and store code in RAM A
- Mode 2—Load and execute code from RAM A and store in RAM B
- Mode 3—Load and execute code from RAM B and store in RAM A
- Mode 4—Load, execute and store code in RAM B.

While Modes 1 and 4 concentrate on an individual RAM, Modes 2 and 3 direct the μP to load programs from one memory map to the other as well as pass

6800-μP definitions

Address bus (A_0 to A_{15})—Accesses memory and peripheral devices for μP; a 16-bit, 3-state bus.

Data bus (D_0 to D_7)—Allows data to pass between memories and μP's programmable registers; an 8-bit, 3-state, bidirectional bus.

Read/$\overline{\text{Write}}$ (R/$\overline{\text{W}}$)—3-state output-control signal. When HIGH, it indicates that the CPU is reading PIA data from the data bus. When LOW, it indicates that the CPU is writing data onto the data bus for delivery to the PIA. Normal standby state is HIGH.

Valid Memory Address (VMA)—CPU output-control signal; goes HIGH whenever a valid address appears on the address bus. When either A or B enable or decode logic also goes HIGH, RAM A or RAM B switches into operation under μP control.

6820 peripheral interface adapter (PIA)—Provides 16 pins configured as two 8-bit I/O ports (PA_0 to PA_7 and PB_0 to PB_7). Each I/O-port line operates as either input or output but does not support bidirectional data transfers. The PIA's 3-state, bidirectional data bus (D_0 to D_7) carries all transactions to and from the 6800 CPU.

Software mnemonics—

JMP—Jump to designated address
JSR—Jump to subroutine
ORG—Originate starting program location
LDAA—Load accumulator A
LDX—Load index register
RTS—Return from subroutine
STAA—Store accumulator A.

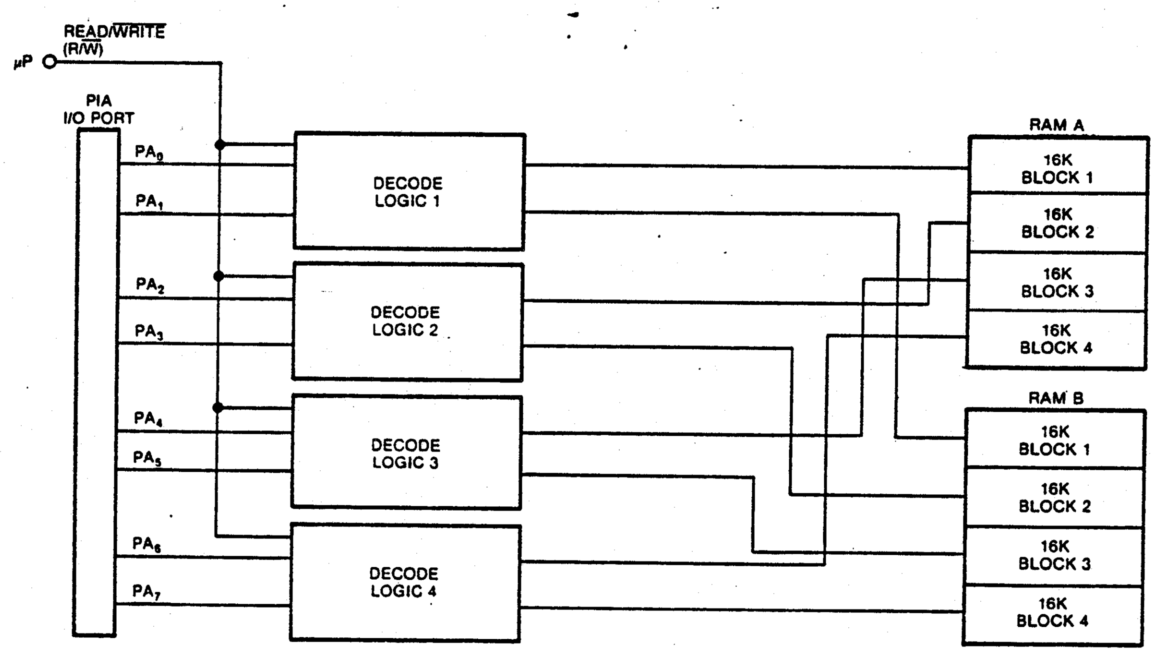


Fig 2—For tighter memory control, *four decode-logic units access 128k of total memory as eight 16k memory maps, divided equally between two 64k RAMs. The resulting software flexibility simplifies memory-map transfers, program loading and common buffering.*

6800 software transfer routines

The following routines prove useful if you implement the memory-mapped approach described in this article on a 6800-µP-based system.

```
ORG $F000   (RAM A or RAM B)
```

Location $F000 lies within the user's program area; $0000 represents the initial transfer-routine location. To change control from one map to another, the program jumps to the map-transfer routines (see example routines below).

```
LDX    #ADDR
JSR    TRNSFR
```

A TRNSFER routine transfers µP control from one memory map to another. It then proceeds to branch to the address stored in the index register. By branching, the subroutine call stores the last address on the program stack.

```
JMP    RETURN
```

A RETURN routine also transfers µP control from one memory map to another. It then proceeds to branch to the address stored in the program stack by executing an RTS statement.

Map-transfer routines:

```
       ORG  $0000   (RAM A)
TRNSFR LDAA #03
       STAA PIA0
       ORG  #0000   (RAM B)
TRNSFR LDAA #00
       STAA PIA0
```

After one map's data accumulates in the PIA, control automatically transfers it to the other map. Operand PIA0 is dedicated to map transfers.

```
       JMP  0,X
RETURN LDAA #03
       STAA PIA0
       RTS
       JMP  0,X
RETURN LDAA #00
       STAA PIA0
       RTS
```

parameters between them. The Read/$\overline{\text{Write}}$ (R/$\overline{\text{W}}$) line activates these latter two modes as follows: When the line goes HIGH, the µP executes a read cycle and transfers code from one map into its CPU; when the line goes LOW, the µP executes a write cycle and stores code in the other memory map (see **box,** "6800 software transfer routines").

The software aspect of dual-map switching yields several advantages: The µP performs all relocations automatically; program parameters and control pass easily between maps; memory maps exchange at any time during the program's execution no matter which map or memory location resides in the CPU; and processing efficiency does not degrade.

Smaller maps offer program versatility

For even tighter memory control, you can subdivide the 128k of total memory into eight 16k memory maps, distributed as four maps in each 64k RAM (Fig 2). This subdivision mandates a fourfold increase in decode-logic hardware, but the advantages of increased software flexibility greatly outweigh the extra expense.

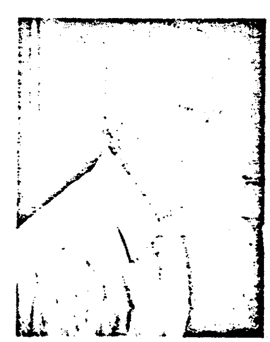
One application of this memory arrangement, for example, places a µP's operating system, program stack and transfer routines in a common memory. In this manner, you eliminate most of the map-transfer software complexities.

Another application employs two operating systems with a common program buffer. In this example, you load a disc operating system into one 16k block of RAM A and a BASIC program into the corresponding block of RAM B. You can then readily transfer control from the low-level language to the high-level one, and vice versa. This loading technique also permits you to program a variety of complex operations within the same buffer space. A variation on this approach provides a choice of program modes by allowing map or mode transfers by means of software or a set of hardware switches.

You can extend a µP's addressing range even further by multiplexing address lines to switch several 64k memory blocks. With only a slight modification of the decode logic, you can thus structure the µP to address 128k, 192k or 256k bytes. For such multiple-map switching, adapt the same hardware/software considerations utilized for the dual-map configuration.

In each case, keep track of the program stack in some common memory set aside for this purpose, because the stack-pointer register within the CPU does not change during map-transfer operations. Locating the stack in common memory permits access of the entire stack by subroutines in all memory maps. **EDN**

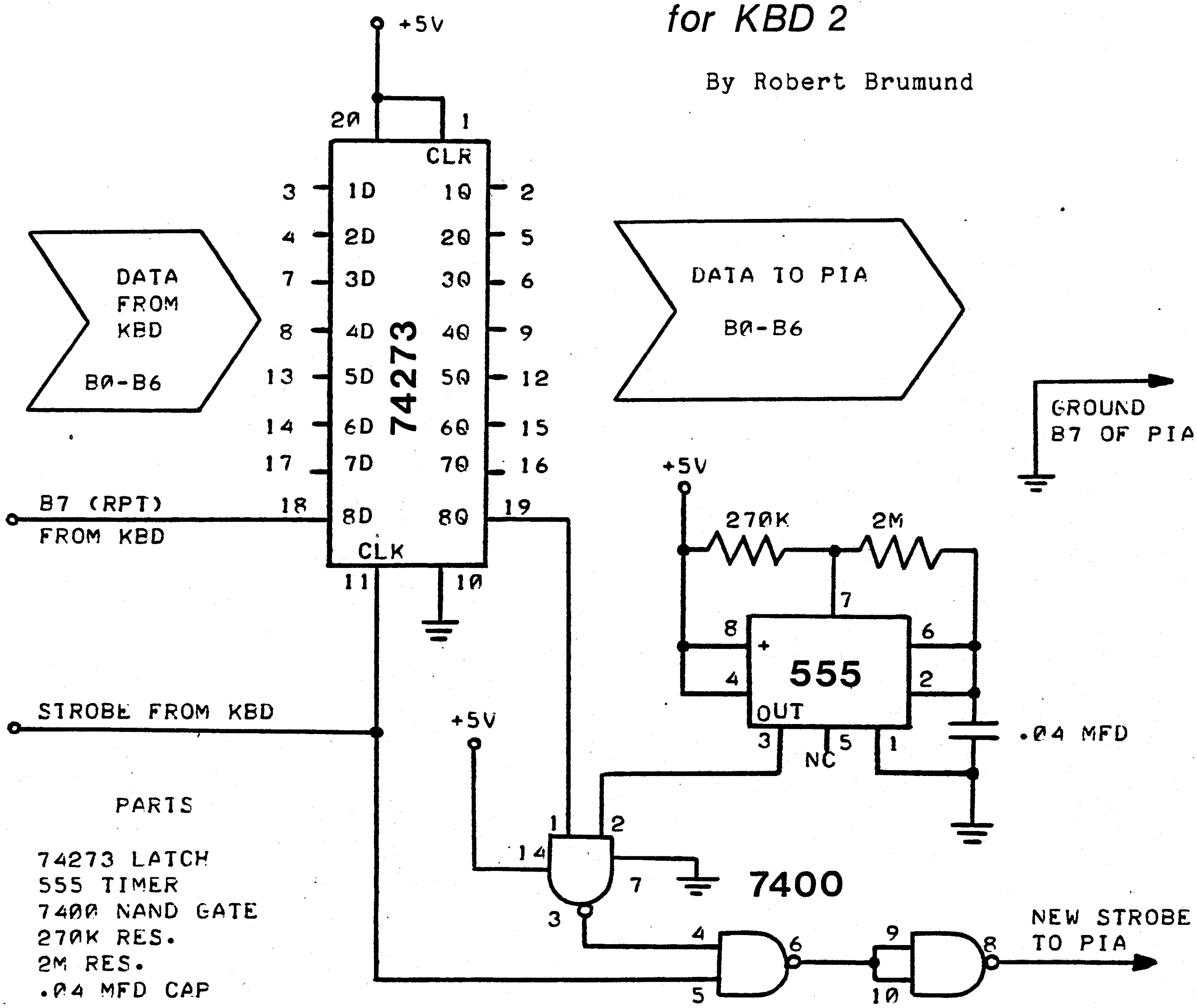
Author's biography



Stephen Strom, a design engineer engaged in microcomputer system development at Motorola Semiconductor Products Inc, Mesa, AZ, previously worked at Harris Corp and earned a BSEE at Carnegie-Mellon University. Stephen's hobbies include tennis, raquetball and carpentry.

Repeat Key / Data Latch for KBD 2

By Robert Brumund



PARTS

74273 LATCH
555 TIMER
7400 NAND GATE
270K RES.
2M RES.
.04 MFD CAP

THE CIRCUIT MAY BE PLACED ANYWHERE BETWEEN THE KEYBOARD AND THE PIA. OPERATION: WITH THE REPEAT KEY DOWN, PRESS ANY CHARACTER KEY. THIS CHARACTER WILL THEN BE OUTPUTTED TO THE PIA AT THE RATE OF 6 PER SEC FOR AS LONG AS THE CHARACTER KEY IS KEPT DOWN.

Subject: Increased System Speed

The Sphere 300 Systems were designed with a basic clock cycle of 1.5 milli-seconds. This results in a 666KHz. clock, which is well below the specified maximum of the MC6800 CPU.

The major reason for the reduced clock rate is that 1702A proms were used. These have an access time of approximately 1 milli-second and therefore Ø2 must be at least this long to allow operation.

One solution which Sphere chose to implement in the 500 systems was to slow down the Ø2 only when the proms are accessed. There are several CPU board changes needed but an upgrade is feasable.

The second obvious solution was to replace the proms with a faster memory. A 2708 adapter was formerly available from Programma and it worked well.

Once the proms have been changed, the system will still not run above 1.2 milliseconds. The reason for this is in the memory write timing. The secret is in the 74123 which (both on CPU and MeM boards) disables the $\overline{WE}$ signal while the data buss does not contain valid data. The present design has a delay of 355 Nano-seconds, which, when coupled with propagation delays, yields only 60 nano-seconds to write the data to the memory with a 500 Nano-second Ø2. This can be modified by changing R1 on each MEM board from a 33K resistor to 12K. This will change the leading edge of the $\overline{WE}$ signal to coincide with the worst case leading edge of the valid data (200 Nano-seconds after the Ø2 leading edge). There is a corresponding resistor on CPU boards which may be changed for memory banks on a CPU board.

Charles Matteson
450 Silas Deane Hwy.
Wethersfield, Connecticut

Editor's note: Most Sphere boards and replacement parts are still available at reasonable cost from Charles.

LISTING OF 1980-1981 ACTIVE SPHERE USERS

1	1	80	ASI MARKET RESEARCH	7655 SUNSET BLVD.	HOLLYWOOD CA.90046	
2	110	80	JOHN BAYLIS	P.O. BOX 137	NSL UTAH 84054	
3	6	80	BITS AN PIECEZ	P.O. BOX 23	WATERLOO 2017 AUSTRALIA	
4	7	80	JEAN-FRANCOIS BOIVIN	1405 RUE IBERVILLE	MONTREAL CANADA H2K 3B2	
5	9	80	SAM BRONSTEIN	9026 AUTOVILLE DR.	COLLEGE PARK MD. 20740	
6	66	80	JEFFREY BROWNSTEIN D.D.S	2 TOR ROAD	WAPPINGERS NY. 12590	914 297-3950
7	116	80	ROBERT BRUMUND	2136 BEL AIR AVE.	SAN JOSE CA. 95128	
8	96	80	DOUG CALLEY	RT 1 BOX 51 LEUPP BANK	FLAGSTAFF AZ. 86001	
9	99	80	JIM CHIN	33-34 70TH ST.	JACKSON HEIGHTS NY. 11372	
10	18	80	SCOTT D'AMRON	2123 SUFFIELD DR.	WINTER PARK FL.32789	
11	22	80	LEROY DANNER	3900(3900-45TH STS.)#23	KENOSHA WI. 53140	
12	21	80	OWEN DAVIS	710 WAIKIKI DR.	DES PLAINES IL.60016	
13	24	80	JOSEPH DAWES	2510 BROADWAY	BIG SPRING TX. 79720	
14	25	80	JOHN DEPPE	1201 2ND ST.	DELANCO NJ. 08075	
15	28	80	R.S. DOWNS	ROUTE 7 BOX 211-A	RALEIGH NC. 27614	
16	27	80	CARLYLE EASTMAN	6016 N.ARLINGTON	SAN PABLO CA. 94806	
17	29	80	ROBERT ENNIS	9322 LAUREL	FONTANA CA. 92335	
18	124	80	DAVID FRANKEL	2012 WEST ST. LOUIS DR.	KOKOMO IN. 46901	
19	31	80	HARRY FRIEDMAN	945 DUDLEY DR.	SHREVEPORT LA. 71104	
20	38	80	CARL GECHNAUER	HIGHWAY 219 S. BOX 26	LAKE PARK IA. 51347	
21	34	80	DAVID GHERSON	1745 RAVIZZA AVE.	SANTA CLARA CA. 95051	
22	35	80	JOHN GIBBON	3 PUDDINGSTONE RD.	NORRIS PLAINS NJ. 07950	
23	37	80	R.M. GRAINGER	RR NO.1 PRESCOTT	ONTARIO CANADA KOE 1TO	
24	40	80	G.K. HALE-LONG ENGINEERING	400 FLYNT VALLEY RD.	WINSTON-SALEM NC. 27104	
25	41	80	WILLIAM HARTWEG	228 ST.MARKS PLACE	STATEN ISLAND NY. 10301	
26	117	80	ROBERT HATFIELD	821 HUNT ST.	ASHLAND KY. 41101	
27	46	80	JOHN HEACOCK	4 STANFORD DR. #3A	BRIDGEWATER NJ. 08807	
28	106	80	DR.GEORGE HORNER	80 DELAMERE AVE.	STRATFORD ONT.CANADA N5A 425	
29	42	80	JOHN IRSIK	1017 MICHIGAN	BEAUMONT CA. 92223	
30	53	80	MICHAEL KOVIS	3810 MAIN STREET	STRATFORD CT. 06497	
31	54	80	DAVID LAKE	243 W.SIRIUS	ANAHEIM CA. 92802	
32	52	80	G.H. LATTA	RT. 3 SMYRNA RD.	SEARCY AR. 72143	
33	58	80	DICK MASON	1037 PARK HILL LANE	ESCONDIDO CA. 92025	
34	61	80	CHARLES MATTESON	450 SILAS DEANE HGWY.	WETHERSFIELD CT. 06109	
35	60	80	TOM MEIER	401 N. LEVITT ST.	ROME N.Y. 13440	
36	64	80	E.HUGH MELTON JR.	8314 UNIVERSITY DRIVE	RICHMOND VA. 23229	
37	119	80	DAVID PAUL MOORE-DIRECTOR	COMPUTER CTR. QUINCY COLLEGE	QUINCY IL. 62301	
38	95	80	DAVE PERRY	57 FOREST HILL ROAD	WEST ORANGE NJ. 07052	
39	72	80	J.C. PIRTLE	P.O. BOX 537	AZLE TX. 76020	
40	76	80	JIM RAEHL	336 N. 750 EAST	OREM UTAH 84057	
41	79	80	WARREN REDDEN	RR 1 BOX 22	GYPSUM KS. 67448	
42	78	80	JOHN RIBLE	51 DAVENPORT ST.	CAMBRIDGE MA. 02141	
43	80	80	W.J. RUTLEDGE	1201 PIERCE ST APT#305	ARLINGTON VA. 22209	
44	82	80	LAWRENCE SAMBUCO	22 FREDRICK DR.	POUGHKEEPSIE NY. 12603	
45	81	80	MIKE SCHWARTZ	719 PATTERSON ST.WEST	LONG BEACH CA. 90806	
46	92	80	DR. HARRY SPAIN	382 BLUE RIDGE RD. HOLIDAY PARK	PITTSBURGH PA. 15239	
47	83	80	DR. ROGER J. SPOTT	13975 CONNECTICUT AVE.	WHEATON MD. 20906	301 299 6030
48	123	80	P.M. TALAJIC	10 KEEFER ST.	OTTAWA ONTARIO K1M2G2 CANADA	
49	86	80	RICHARD THERIAULT	12120 'ARCHVEQUE	MONTREAL CANADA H1H 3C1	
50	87	80	GENE WALLIS	1954R OLD MIDDLEFIELD WAY	MOUNTAIN VIEW CA. 94043	
51	91	80	WARREN WEIMER	23025 KINARD AVENUE	CARSON CA. 90745	213 835-9417
52	89	80	CHAN WAI YUNG	P.O. BOX K-2296	KOWLOON HONG KONG	

N E X T   I S S U E

The editors find it hard to believe there will ever be another SPHERE newsletter if there is nothing to print in it. Those of us left must do our parts in keeping the information flowing.

Send in anything you have....we would very much like to hear from those users who have not ever sent in material. After five years we know you must have been running something!

From now on we will produce a newsletter whenever there is enough material. The regular issue spacing of the past will not be possible with the current scarcity of contributions.

Our most generous Thank Yous go to the Editors of EDN magazine, who allowed us to reprint some of the material in this issue.

****

The enclosed sequence provides for excellent user prompt for input statements in BASIC as it begins on the first dot and only accepts the desired number of characters. You must use two dots more than the number of characters requested.

```
100 PRINT "PATIENT'S FIRST NAME"
105 PRINT "...........";
110 PAT 7EFCFD
120 INPUT A$
130 IF LEN(A$)>10 THEN GOTO 100
140 PRINT A$
```

****

CSS BASIC accepts tapes of Peter Stark's 6800 Assembler (written in Basic). It should be easy to modify (add macros) but it runs slowly. Available from STAR KITS on cassette.