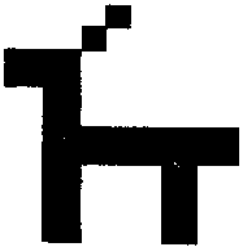
Questdata
Volume 2 Issue # 6

MERRY CHRISTMAS

ELF WELCOME



by
Hugh Dagg

This program can take any message, story, letter, or figure you care to create and move it across a TV screen from right to left. It can be run on a Cosmac computer with at least five pages of memory. With five pages of memory, up to 256 characters can be displayed and be repeated endlessly. For every page of memory beyond five, 256 more characters can be displayed.

The basic setup of the pages is shown in Diagram 1. To describe the main program itself, I have used a flow chart, Diagram 2.

Using the program is relatively easy. Once the message you want is formed, encode it using the Table below and load the numbers starting at 0400. If you would like the message to repeat, load an FF after the message.

Along with messages, pictures can be displayed. An example of this is my brother's boat. To see it load 2C 2D 2E 2F 1A FF starting at 0400 and start the program running again. Since it is near Christmas, my mother made up some Christmas characters portrayed in the following encoded Christmas message: 31 1A 30 1A 0C 04 11 11 18 1A 02 07 11 08 12 13 0C 00 12 1A FF.

To create your own objects or scenes on the screen, first draw a likeness on paper 8 lines high and less than 64 bytes long. Second, divide the scene into byte-wide columns and load it as in Diagram 3. Note loading can occur anywhere in either page 1 or 2 as long as the first of the eight bytes for a character starts at a n0 or n8 where n is any hex digit.

At this point one must be aware that there is a capability of 64 characters and each character has a hex number from 00 to 3F inclusive. 00 is the number of the character represented by 0100 to 0107 up to 3F which is the number of the character represented by 02F8 to 02FF. So wherever you put your character, remember the hex character number for it.

Added Features and Notes.

1. Near the beginning of the program at memory location 0031 to 003F the display page is cleared. If you would like to clear just the eight lines where the message appears and have a display of your own in the rest of the page, load 47 in (M0032) and 3F in (M0035) and load in your display.

2. The position of the shifted message can occur anywhere on the screen by changing (M0080) which is the right-most byte of the top line. Be careful to have that hex number greater than 07 and less than C7 and of the form n0 or nF. If the number is not in the above range the register that works with the display page could increment or decrement to an adjacent page.

3. The rate at which the characters cross the screen is determined by (M00AD). I have found that 05 creates a good speed.

4. For an interesting effect after a character string is loaded, change location (M00A2) to 0F. What is really happening here?

Diagram 1

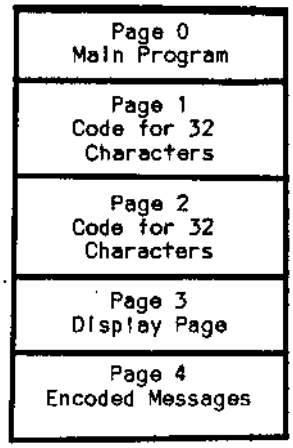
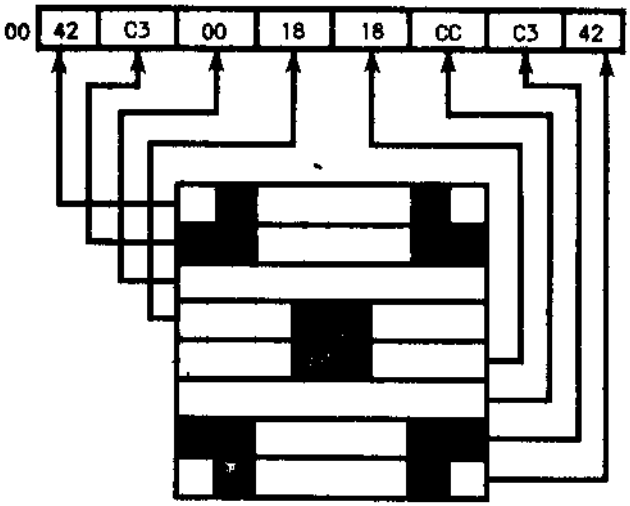


Diagram 2



Dump Listing of Characters

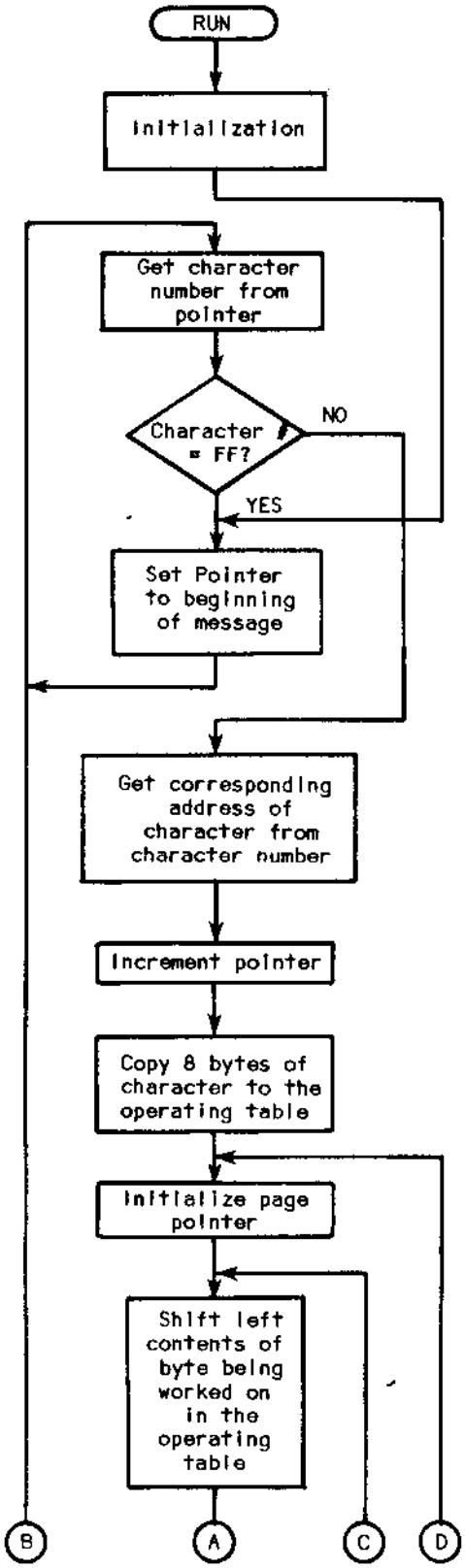
NO.	CHAR.	ADDRESS	CODE							
00	A	0100	7E	42	42	7E	42	42	42	42
01	B	0108	7C	42	42	7C	7C	42	42	7C
02	C	0110	1E	60	40	40	40	40	60	1E
03	D	0118	78	44	42	42	42	42	44	78
04	E	0120	7E	40	40	7C	7C	40	40	7E
05	F	0128	7E	40	40	7C	40	40	40	40
06	G	0130	7E	40	40	40	4F	42	42	7E
07	H	0138	42	42	42	7E	7E	42	42	42
08	I	0140	18	18	18	18	18	18	18	18
09	J	0148	02	02	02	02	02	42	42	7E
0A	K	0150	42	44	48	70	70	48	44	42
0B	L	0158	40	40	40	40	40	40	7E	7E
0C	M	0160	42	66	5A	42	42	42	42	42
0D	N	0168	62	62	52	52	4A	4A	46	46
0E	O	0170	3C	42	42	42	42	42	42	3C
0F	P	0178	7E	42	42	7E	42	40	40	40
10	Q	0180	3C	42	42	42	52	4A	46	3E
11	R	0188	7E	42	42	7E	50	48	44	42
12	S	0140	7E	42	40	7E	02	02	42	7E
13	T	0148	7E	18	18	18	18	18	18	18
14	U	01A0	42	42	42	42	42	42	42	7E
15	V	01A8	42	42	42	24	24	24	18	18
16	W	01B0	42	42	42	42	42	5A	66	42
17	X	01B8	42	24	24	18	18	24	24	42
18	Y	01C0	42	24	18	18	18	18	18	18
19	Z	01C8	7E	02	04	08	10	20	40	7E
1A	(blank)	01D0	00	00	00	00	00	00	00	00
1B	"	01D8	28	28	00	00	00	00	00	00
1C	.	01E0	00	00	00	00	00	00	40	40
1D	,	01E8	00	00	00	00	60	60	20	40
1E	!	01F0	18	18	18	18	18	18	00	18
1F	?	01F8	7E	02	02	02	1E	18	00	18
20	0	0200	7E	42	42	42	42	42	42	7E
21	1	0208	18	28	18	18	18	18	18	7F
22	2	0210	7E	02	02	02	02	04	18	7E
23	3	0218	7E	02	02	3E	02	02	02	7E
24	4	0220	42	42	42	42	7E	02	02	02
25	5	0228	7E	40	40	40	7E	02	02	7E
26	6	0230	7E	40	40	40	7E	42	42	7E
27	7	0238	7E	02	02	02	02	02	02	02
28	8	0240	7E	42	42	7E	42	42	42	7E
29	9	0248	7E	42	42	7E	02	02	02	02

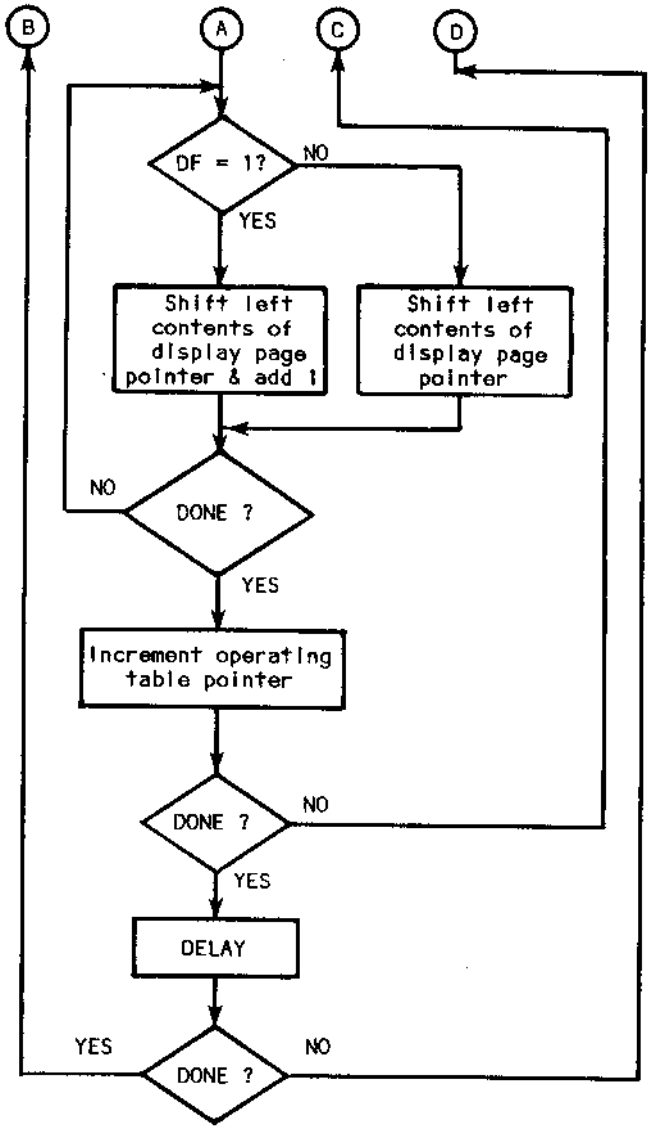
NO.	CHAR.	ADDRESS	CODE							
2A	tank	0250	00	00	1F	00	0F	0F	03	01
2B	""	0258	20	78	F8	78	FF	FF	56	FC
2C	boat	0260	00	00	1F	7F	04	7F	3F	1F
2D	""	0268	07	02	3F	28	28	FF	FF	FF
2E	""	0270	00	00	F8	28	28	FF	FF	FF
2F	""	0278	00	00	1E	1E	10	F8	FC	F8
30	reindeer	0280	08	10	60	20	3F	3E	22	22
31	tree	0288	08	08	1C	1C	3E	3E	7F	08

.
.
. Extra Space for your own characters
.
3F

Registers Used:

P=7 or 1
X=5 or 2
0=DMA Pointer
1=Interrupt
2=Stack Pointer
3=Work Space Pointer
4=Character Pointer
5=Display Pointer
6=Message Pointer
7=Main PC
8=Line Counter
9=Byte Counter
A=Unused
B=Unused
C=Unused
D=Unused
E=Unused
F=Unused






```
ADDR CODE
0000 90
0001 B1
0002 B2
0003 B3
0004 B7
0005 F8
0006 14
0007 A1
0008 F8
0009 FD
000A A2
000B F8
000C 31
000D A7
000E F8
000F 03
0010 B5
0011 D7
0012 72
0013 70
0014 22
```

```
COMMENT
R1=0014
Video interrupt
Entry point

R2=00FD
X for interrupt


R7=0031
Main PC

R5=03
Display register
Works with
display page


Take a trip
Restore X,D & P
```

```
ADDR CODE       COMMENT
0015 78         X & P --> Stack
0016 22
0017 52         Store D
0018 C4         For
0019 C4         timing
001A C4
001B F8         R0=0300
001C 03
001D B0         Page 3 is
001E F8         Display page
001F 00
0020 A0
0021 80
0022 E2
0023 E2
0024 20         Delay
0025 A0         between
0026 E2         DMA's
0027 20
0028 A0
0029 E2
002A 20
002B A0
002C 3C         Branch to
002D 21         0021 if not finished DMA
002E 2B         Decrement RB by 1
002F 30         Branch to
0030 12         0012
0031 F8
0032 FF         (or 47)<--Only if you want
                the features
0033 A5
0034 F8
0035 FF         (or 3F)<--Only if you want
                the features
0036 A8
0037 E5
0038 90
0039 73         Clear
003A 28         display
003B 88         Page 3
003C 3A
003D 38
003E 90
003F 55
0040 F8
0041 05         Set delay
0042 AB
0043 E2         Sex
0044 69         TV on
0045 30         Branch to
0046 4C         004C
0047 06         Character # out
0048 FC         Add 1
0049 01
004A 3A         If 00 start
004B 53         message over
004C 93
004D A6
004E F8         R6=0400
004F 04
0050 B6
0051 30         Branch to
0052 47         0047
0053 06         Char. # out
0054 FE
0055 FE         Shift 3 times
0056 FE         (Multiplication by 8)
0057 33         If D>1F branch
0058 61         To 00 61
0059 FC         If not add
005A 07         7
005B A4         Store R4 low
005C F8
005B 01         R4=01
```

```
ADDR CODE       COMMENT
005E B4
005F 30         Branch to
0060 6D         006D
0061 06         Char. # out
0062 FC         Subtract 20
0063 E0         and
0064 FE         Shift
0065 FE         3 times
0066 FE         (Multiplication by 8)
0067 FC         Add 7
0068 07
0069 A4         Store R4.0
006A F8
006B 02         R4=02
006C B4
006D 16         Increment R6
006E F8
006F 08
0070 A9         Move 8
0074 F8         bytes of
0072 F7         character
0073 A3         into
0074 04         00F0-00F8
0075 53         Using
0076 24         R3 & R4
0077 23
0078 29
0079 89
007A 3A
007B 74
007C F8         For 8 shifts
007D 08         For whole
007E AA         letter
007F F8
0080 0F         R5=03 0F
0081 A5
0082 F8         For shifting
0083 08         All 8 lines
0084 A8
0085 F8
0086 F0         R3=00 F0
0087 A3
0088 F8         For shifting
0089 08         8 bytes in
008A A9         a line
008B 03         R3 contents out
008C FE         shift
008D 53         back in
008E 33         Test DF
008F 95
0090 05         If DF=0
0091 FE         Shift R5
0092 55
0093 30         Branch to
0094 9B         00 9B
0095 05
0096 FE         If DF=0
0097 AF         Shift R5 &
0098 1F         Add 1
0099 8F
009A 55
009B 25
009C 29         Till 8 bytes
009D 89         in line are
009E 3A         done
009F 8E
00A0 85         Add 10 to
00A1 FC         R5 to get
00A2 10         to next line
00A3 A5
00A4 13         Increment R3 by 1
00A5 28
00A6 88         Do till all
00A7 3A         8 lines are
00A8 88         done
```

```
ADDR CODE          COMMENT
00A9 8B
00AA 3A            Delay
00AB A9
00AC F8            Reset
00AD 05            Delay
00AE AB
00AF 2A
00B0 8A            Do until all 8
00B1 3A            bits of
00B2 7F            character shifted
00B3 30            Branch to
00B4 47            0047
```

```
0000 90B1 B2B3 B7F8 14A1 F8FD A2F8 31A7 F803
0010 B5D7 7270 2278 2252 C4C4 C4F8 03B0 F800
0020 A080 E2E2 20A0 E220 A0E2 20A0 3C21 2B30
0030 12F8 FFA5 F8FF A8E5 9073 2888 3A38 9055
0040 F805 ABE2 6930 4C06 FC01 3A53 93A6 F804
0050 B630 4706 FEFE FE33 61FC 07A4 F801 B430
0060 6D06 FCE0 FEFE FEFC 07A4 F802 B416 F808
0070 A9F8 F7A3 0453 2423 2989 3A74 F808 AAF8
0080 0FA5 F808 A8F8 F0A3 F808 A903 FE53 3395
0090 05FE 5530 9805 FEAF 1F8F 5525 2989 3A8E
00A0 85FC 10A5 1328 883A 888B 3AA9 F805 AB2A
00B0 8A3A 7F30 4700 0000 0000 0000 0000 0000
00C0 0000 0000 0000 0000 0000 0000 0000 0000
00D0 0000 0000 0000 0000 0000 0000 0000 0000
00E0 0000 0000 0000 0000 0000 0000 0000 0000
00F0 0808 0808 0808 08F8 0000 0005 27FF 0000
0100 7E42 427E 4242 4242 7C42 427C 7C42 427C
0110 1E60 4040 4040 601E 7844 4242 4242 4478
0120 7E40 407C 7C40 407E 7E40 407C 4040 4040
0130 7E40 4040 4F42 427E 4242 427E 7E42 4242
0140 1818 1818 1818 1818 0202 0202 0242 427E
0150 4244 4870 7048 4442 4040 4040 4040 7E7E
0160 4266 5A42 4242 4242 6262 5252 4A4A 4646
0170 3C42 4242 4242 423C 7E42 427E 4040 4040
0180 3C42 4242 524A 463E 7E42 427E 5048 4442
0190 7E42 407E 0202 427E 7E18 1818 1818 1818
01A0 4242 4242 4242 427E 4242 4224 2424 1818
01B0 4242 4242 425A 6642 4224 2418 1824 2442
01C0 4224 1818 1818 1818 7E02 0408 1020 407E
01D0 0000 0000 0000 0000 2828 0000 0000 0000
01E0 0000 0000 0000 4040 0000 0000 6060 2040
01F0 1818 1818 1818 0018 7E02 0202 1E18 0018
0200 7E42 4242 4242 427E 1828 1818 1818 187E
0210 7E02 0202 0204 187E 7E02 023E 0202 027E
0220 4242 4242 7E02 0202 7E40 4040 7E02 027E
0230 7E40 4040 7E42 427E 7E02 0202 0202 0202
0240 7E42 427E 4242 427E 7E42 427E 0202 0202
0250 0000 1F00 0F0F 0301 2078 F878 FFFF 56FC
0260 0000 1F7F 047F 3F1F 0702 3F28 28FF FFFF
0270 0000 F828 28FF FFFF 0000 1E1E 10F8 FCF8
0280 0810 6020 3F3E 2222 0808 1C1C 3E3E 7F08
```

VB1B HEX DUMP

by
Phillip B. Lieschesk i III

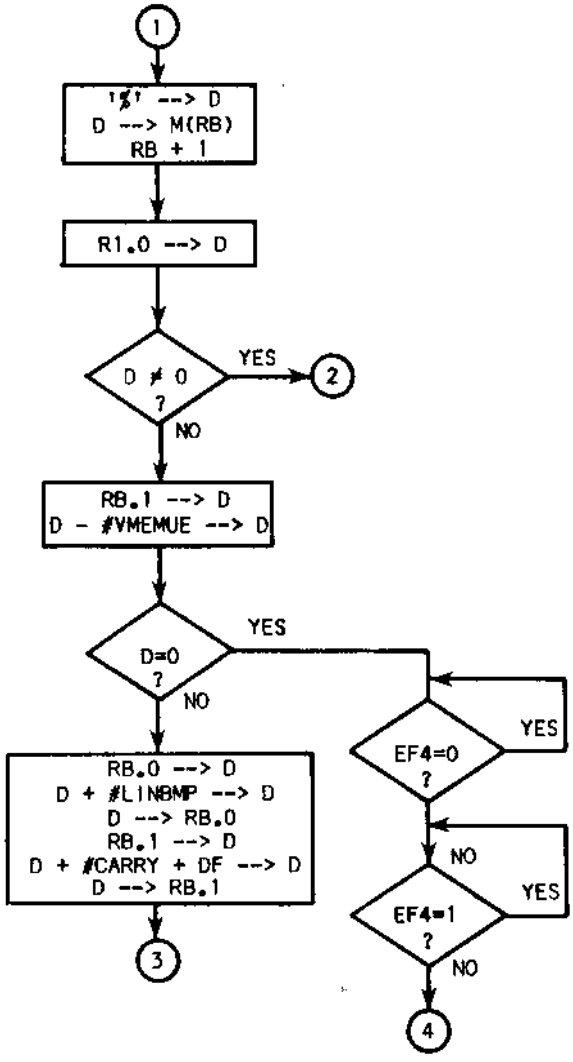
VB1B Hex Dump is a small program which displays the contents of a certain portion of memory on a video monitor via the VB1B video board. This program was more or less created out of necessity. It is quite useful in inspecting the code of a large program after manual loading or during debugging. It makes memory inspection quick and less tedious.

Hex Dump is based on the VB1B video board. It assumes that the VB1B is operating in the 32 character/line mode, and its video memory begins and at F000. The program uses R2 as stack pointer, RA as memory inspect pointer, RB as video pointer and R4 as a subroutine call register. It assumes that R3 is the main program counter. Basically, it begins by inspecting memory at 0000. First the memory address which is contained in RA is displayed at the beginning of the line. A colon with a blank follows the memory address and then eight bytes of memory are displayed with a blank inserted in between each displayed byte. After this a new line is displayed, until twelve lines have been put on the screen. In order to view the next portion of memory, one merely presses the Input key on the hex keypad. Since Hex Dump contains no screen clearing routine, portions of the video memory which are not written upon by the Hex Dump will remain as is, so video garbage may be seen around the edges of the display.

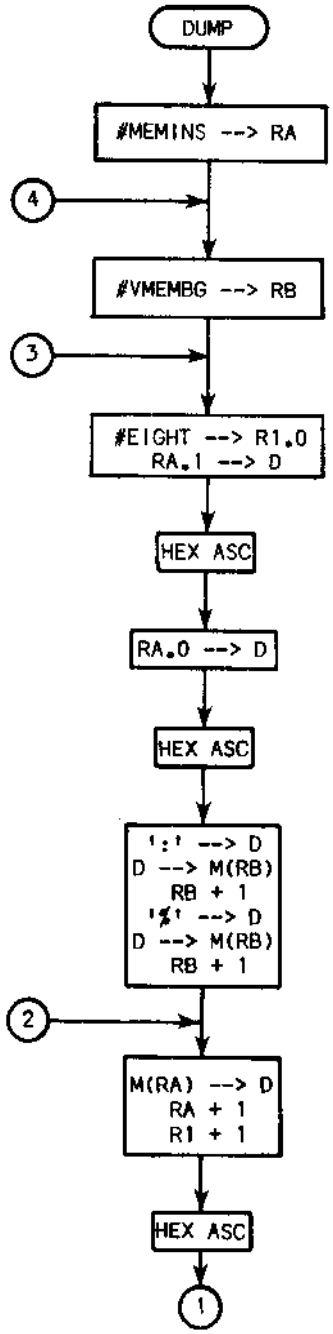
Hex Dump contains one subroutine known as HEXASC. It is called by using a SEP instruction and R4 is its program counter. It basically accepts a byte contained in accumulator D, converts it to ASCII code and displays it on the VB1B video board. First, it operates upon the upper nibble of D and then on the lower nibble of the D. This subroutine is called upon in order to display the memory addresses and the bytes in memory.

This program has been very handy in inspecting code and as stated earlier it was created out of necessity. Hopefully, others who have not already written similar programs can use this program in their work. Finally, it is believed that with little modification, Hex Dump can be used to operate the Gremlin video board, but this has not yet been tried.

```
ADDR CODE     COMMENT
0000 E2       Set R2 as Stack Pointer
0001 93       Determine which page it is operating in
0002 B2       and push it into R2.1 and R4.1
0003 B4
0004 F8 00    Set Memory Inspect Pointer (RA)
0006 AA BA
0008 F8 FF    Finish setting up Stack Pointer
000A A2
000B F8 43    Finish setting up HEXASC Subroutine
              Pointer (R4)
000D A4
000E F8 40    Initialize Video Pointer (RB)
0010 AB
0011 F8 F0
0013 BB
0014 F8 08    Initialize 8 Counter (R1)
0016 A1
0017 9A       Get High Memory Address
0018 D4       Display it on Video
0019 8A       Get low Memory Address
001A D4       Display it on Video
001B F8 3A    Place a colon after Address
001D 5B
001E 1B       Bump Video Pointer
001F F8 20    Then a Blank
0021 5B
0022 1B       Bump Video Pointer
0023 4A       Get byte from memory and bump pointer
0024 D4       Display it on Video
0025 F8 20    Place a blank after it
0027 5B
```




```
ADDR CODE     COMMENT
0028 1B       Bump Video Pointer
0029 21       Bump 8 counter
002A 81       Check if finished with a line
002B 3A 23    If not, display another byte
002D 9B       Finished with a line
002E FF F3    Check if out of Video Memory
0030 32 3C    If so, go wait for an I key depression.
0032 8B
0033 FC 22    Set video Pointer to point to the
              beginning of
0035 AB       the next line
0036 9B
0037 7C 00
0039 BB
003A 30 14    Try it again
003C 3F 3C    Wait for I key depression
003E 37 3E
0040 30 0E    Go display next portion of memory
```

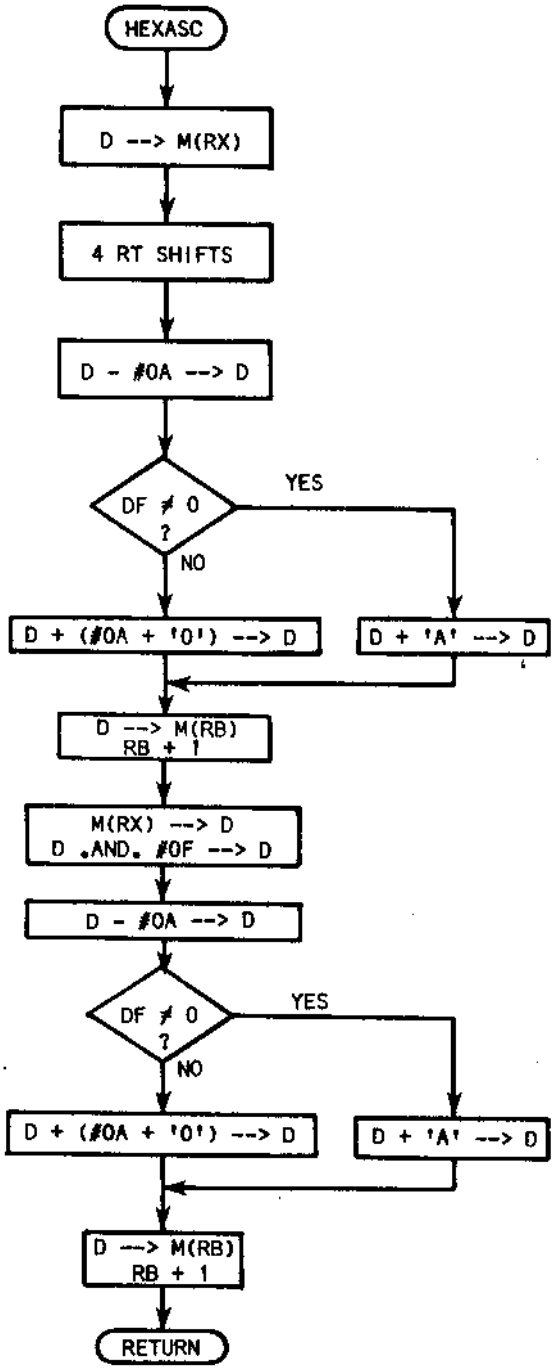


```
ADDR CODE    COMMENT
0042 D3      Return
0043 52      Entry point; Save D on Stack
0044 F6 F6   Inspect upper nibble of D
0046 F6 F6
0048 FF 0A   Convert it to ASCII Code
004A 33 4F
004C FC 3A
004E C8
004F FC 41
0051 5B      Store it in Video Memory
0052 1B      Bump Video Pointer
0053 F0      Retrieve value from Stack
0054 FA 0F   Inspect lower nibble of D
0056 FF 0A   Convert it to ASCII Code
0058 33 5D
005A FC 3A   If .LT. 0A, add 0A+'0'
005C C8
005D FC 41   If .GE. 0A, add 'A'.
005F 5B      Store it in Video Memory
0060 1B      Bump Video Pointer
0061 30 42   Branch to Return
*
* XX- This program can operate in any full page of
* memory.
*
```

Registers Used:

R1= 8 Counter
R2= Stack Pointer
R3= Main Program Counter
R4= HEXASC Program Counter
RA= Memory Inspect Pointer
RB= Video Pointer

```
0000 E293 B2B4 F800 AABA F8FF A2F8 43A4 F840
0010 ABF8 F0BB F808 A19A D48A D4F8 3A5B 1BF8
0020 205B 1B4A D4F8 205B 1B21 813A 239B FFF3
0030 323C 8BFC 22AB 9B7C 00BB 3014 3F3C 373E
0040 300E D352 F6F6 F6F6 FF0A 334F FC3A C8FC
0050 415B 1BF0 FA0F FF0A 335D FC3A C8FC 415B
0060 1B30 42
```



Flowchart Constant Table:

```
MEMINS = 0000
VMEMBG = 0F00
EIGHT  = 08
VMEMUE = FC
LINBMP = 21
CARRY  = 00
```

QUESTDATA
P.O. Box 4430
Santa Clara, CA 95054

Publisher .....Quest Electronics
Editor .....Paul Messinger
Assistant to Editor ...Jeanette Johnson
Associate Editor .....Allan Armstrong
Contributing Editors Ron Cenker
Van Baker
Art and Graphics .....Holly Olson
Proof Reading .....Judy Pitkin
Production .....John Larimer
Circulation .....Sue Orr

CHRISTMAS MUSIC

by
Jan Beckman

Below is a Music Algorithm which is a modified version of the Algorithm written by Ed McCormick which was published in Popular Electronics Magazine Feb. 1978 issue, and published earlier in Questdata Volume I, Issue 3. Along with the Algorithm is some "Christmas Music for our Elf".

```
ADDR CODE
0000 E5
0001 F8 47 A5
0004 F8 00 B5
0007 F0
0008 3A 0B
000A 00
000B A8
000C 15 64 25
000F F0
0010 A7
0011 F8 10 A9
0014 87 FC B4
0017 33 21
0019 31 1E
001B 7B
001C 30 21
001E 7A
001F 30 21
0021 87
0022 FF 01
0024 3A 22
```

```
ADDR CODE
0026 89
0027 FF 01
0029 A9
002A 3A 34
002C 88
002D FF 01
002F A8
0030 3A 11
0032 30 3C
0034 C4 C4
0036 30 38
0038 30 3A
003A 30 14
003C 7A
003D 15
003E F8 0E BE
0041 2E 9E
0043 3A 41
0045 30 07
0047  MUSIC PROGRAM
```

Joy to the World

by
Gerald VanHorn

```
ADDR CODE
0047 25 12
0049 1A 14
004B 08 17
004D 2A 1B
004F 0C 1F
0051 19 22
0053 13 27
0055 1B 2D
0057 0E 1B
0059 2F 17
005B 0F 17
005D 34 14
005F 11 14
0061 37 12
0063 12 12
0065 12 12
0067 11 14
0069 0F 17
006B 0E 1B
006D 16 1B
006F 06 1F
0071 0C 22
0073 12 12
0075 12 12
0077 11 14
0079 0F 17
007B 0E 1B
007D 16 1B
007F 06 1F
```

```
ADDR CODE
0081 0C 22
0083 0C 22
0085 0C 22
0087 0C 22
0089 0C 22
008B 06 22
008D 06 1F
008F 2A 1B
0091 06 1F
0093 06 22
0095 0A 27
0097 0A 27
0099 0A 27
009B 05 27
009D 06 22
009F 24 1F
00A1 06 22
00A3 05 27
00A5 09 2D
00A7 25 12
00A9 0F 17
00AB 16 1B
00AD 05 1F
00AF 0C 22
00B1 0C 1F
00B3 17 22
00B5 14 27
00B7 26 2D
00B9 00
```

Silent Night Music

by
Jan Beckman

```
ADDR CODE
0047 21 24
0049 0C 1F
004B 16 24
004D 36 2D
004F 21 24
0051 0C 1F
0053 16 24
0055 36 2D
0057 41 15
0059 21 15
005B 52 1B
005D 3A 19
005F 1D 19
0061 41 24
0063 31 1F
0065 18 1F
0067 29 19
0069 0F 1B
006B 18 1F
006D 21 24
006F 0C 1F
0071 16 24
0073 24 2D
0075 0B 4C
```

```
ADDR CODE
0077 31 1F
0079 18 1F
007B 29 19
007D 0E 1B
007F 18 1F
0081 21 24
0083 0C 1F
0085 16 24
0087 24 2D
0089 0B 4C
008B 41 15
008D 21 15
008F 39 11
0091 10 15
0093 1B 1B
0095 4E 19
0097 4A 12
0099 0B 4C
009B 29 19
009D 0B 24
009F 12 2D
00A1 27 24
00A3 0A 2A
00A5 10 33
00A7 3A 3B
00A9 00
```

God Rest Ye Merry Gentlemen

by
Gerald M. VanHorn

```
ADDR CODE
0047 12 2D
0049 12 2D
004B 1C 1B
004D 1B 1B
004F 18 1F
0051 16 24
0053 14 27
0055 12 2D
0057 10 33
0059 12 2D
0058 14 27
005D 16 24
005F 18 1F
0061 51 1B
0063 12 2D
0065 12 2D
0067 1C 1B
0069 1B 1B
006B 18 1F
006D 16 24
006F 14 27
0071 12 2D
0073 10 33
0075 12 2D
0077 14 27
0079 16 24
007B 18 1F
007D 37 1B
```

```
ADDR CODE
007F 0A 4C
0081 1C 1B
0083 1D 19
0085 18 1F
0087 1C 1B
0089 1D 19
008B 21 15
008D 25 12
008F 1C 1B
0091 18 1F
0093 16 24
0095 12 2D
0097 14 27
0099 16 24
009B 31 1F
009D 16 24
009F 18 1F
00A1 37 1B
00A3 1D 19
00A5 1C 1B
00A7 1C 1B
00A9 18 1F
00AB 16 24
00AD 14 27
00AF 24 2D
00B1 0B 24
00B3 0A 27
00B5 12 2D
```

```
ADDR CODE
00B9 31 1F
00B9 15 24
00BB 18 1F
00BD 1C 1B
00BF 1D 19
00C1 21 15
00C3 25 12
00C5 1C 1B
00C7 18 1F
00C9 16 24
00CB 14 27
00CD 36 2D
00CF 00
```

MORE CHRISTMAS MUSIC

by
Paul Thompson

Here are some more Christmas tunes to be played with the Beckman Music Algorithm.

Jolly Old Saint Nicholas

```
ADDR CODE
0047 15 27
0049 15 27
004B 15 27
004D 15 27
004F 12 2D
0051 12 2D
0053 25 2D
0055 10 33
0057 10 33
0059 10 33
005B 10 33
005D 52 27
005F 0E 3F
0061 0E 3F
0063 0E 3F
0065 0E 3F
0067 0C 47
0069 0C 47
006B 21 33
006D 10 37
006F 10 33
0071 12 2D
0073 15 27
0075 49 2D
0077 15 27
0079 15 27
007B 15 27
007D 15 27
007F 12 2D
0081 12 2D
0083 25 2D
0085 10 33
0087 10 33
0089 10 33
008B 10 33
008D 52 27
008F 0E 3F
0091 0E 3F
0093 0E 3F
0095 0E 3F
0097 0C 47
0099 0C 47
009B 21 33
009D 12 2D
009F 10 33
00A1 12 2D
00A3 15 27
00A5 41 33
00A7 44 4C
00A9 00
```

Good King Wenceslaus

```
ADDR CODE
0047 10 33
0049 10 33
004B 10 33
004D 12 2D
004F 10 33
0051 10 33
0053 19 47
0055 0E 3F
0057 0C 47
0059 0E 3F
005B 10 37
005D 21 33
005F 21 33
0061 10 33
0063 10 33
0065 10 33
0067 12 2D
0069 10 33
006B 10 33
006D 19 47
006F 0E 3F
0071 0C 47
0073 0E 3F
0075 10 37
0077 21 33
0079 21 33
007B 19 1F
007D 16 24
007F 15 27
0081 12 2D
0083 15 27
0085 12 2D
0087 21 33
0089 0E 3F
008B 0C 47
008D 0E 3F
008F 10 37
0091 21 33
0093 21 33
0095 0C 47
0097 0C 47
0099 0E 3F
009B 10 37
009D 10 33
009F 10 33
00A1 25 2D
00A3 19 1F
00A5 16 24
00A7 15 27
00A9 12 2D
00AB 21 33
00AD 2C 24
00AF 21 33
00B1 44 4C
00B3 00
```

Auld Lang Syne

```
ADDR CODE
0047 19 47
0049 31 33
004B 10 37
004D 21 33
004F 29 27
0051 37 2D
0053 10 33
0055 25 2D
0057 29 27
0059 31 33
005B 10 33
005D 29 27
005F 31 1F
0061 A5 1B
0063 37 1B
0065 4A 1F
0067 15 27
0069 29 27
006B 21 33
006D 37 2D
006F 10 33
0071 25 2D
0073 29 27
0075 31 33
0077 0E 3F
0079 1C 3F
007B 19 47
007D 62 33
007F 37 1B
0081 4A 1F
0083 15 27
0085 29 27
0087 21 33
0089 37 2D
008B 10 33
008D 25 2D
008F 37 1B
0091 4A 1F
0093 15 27
0095 29 27
0097 31 1F
0099 A5 1B
009B 37 1B
009D 4A 1F
009F 15 27
00A1 29 27
00A3 21 33
00A5 37 2D
00A7 10 33
00A9 25 2D
00AB 29 27
00AD 31 33
00AF 0E 3F
00B1 1C 3F
00B3 19 47
00B5 62 33
00B7 44 4C
00B9 00
```

O Christmas Tree

```
ADDR CODE
0047 0C 47
0049 08 33
004B 08 33
004D 10 33
00BF 12 2D
0051 0A 27
0053 0A 27
0055 15 27
0057 15 27
0059 09 2D
005B 0A 27
005D 16 24
005F 10 37
0061 12 2D
0063 10 33
0065 0C 47
0067 08 33
0069 08 33
006B 10 33
006D 12 2D
006F 0A 27
0071 0A 27
0073 15 27
0075 15 27
0077 09 2D
0079 0A 27
007B 16 24
007D 10 37
007F 12 2D
0081 10 33
0083 19 1F
0085 0C 1F
0087 0A 27
0089 1C 1B
008B 19 1F
008D 0C 1F
008F 0B 24
0091 16 24
0093 16 24
0095 0B 24
0097 09 2D
0099 19 1F
009B 16 24
009D 0B 24
009F 0A 27
00A1 15 27
00A3 0C 47
00A5 08 33
00A7 08 33
00A9 10 33
00AB 12 2D
00AD 0A 27
00AF 0A 27
00B1 15 27
00B3 15 27
00B5 09 2D
00B7 0A 27
00B9 16 24
00BB 10 37
00BD 12 2D
00BF 10 33
00C1 44 4C
00C3 00
```

USING SUPER MONITOR I/O

by
Van C. Baker

Super Monitor versions 2.0 and 2.1 contain very useful video display output and keyboard input routines which you can use to avoid having to write your own. Version 2.0 is for use with either a Solid State Music Board or a Polymorphics Video Board, both utilizing a 64-character-per-line format. Version 2.1 is for use with the Gremlin board, which utilizes a 32-character-per-line format with 16 lines per screen. Both versions support ASCII keyboard input on Input Port 5, using EF2 as a Data Available flag.

INPUT: The operation of both Super Monitor versions on input is the same. The input routine resides at location 8300 and is called via a Standard Call procedure. Before calling it, however, be sure that R2 (Which becomes the Stack Pointer if not already so designated) is pointing to a free location. The parity bit (Most significant bit, D7) is automatically stripped, ie., set to 0, and the input byte with D7 zero is saved in RF.1 (High half of register F). Immediately after receiving an input byte, the input routine branches to the display routine so that the input character is automatically displayed. Your calling program retrieves the input byte by accessing RF.1.

When using Version 2.0 of the Super Monitor, you must electrically pull the MSB (Bit D7) on the keyboard input high if the Polymorphic board is used or pull it low if the Solid State Music board is being used. This is done so that the output routine can determine which board you are using. Consequently, you should not use the parity bit generated by your keyboard.

OUTPUT: Version 2.0 and 2.1 output routines function in slightly different ways because the display formats are different; however, they are both called in exactly the same way. Register F.1 must first be pre-loaded with the character to be output and R2 must point to a location having at least 7 free bytes below it; ie., the stack being pointed to by R2 must be at least 7 deep. When RF.1 is loaded with the output character, the most significant bit (D7) must be zero, ie., the character must be 7-bit ASCII for the character to "print". Otherwise, if the MSB is 1, the character will not be displayed. (Version 2.1 users please see the Gremlin Users' Manual for a description of the ASCII format used for display using the Gremlin Color Video). A so-called "CRTXY" value is saved in location 98BC in the 1/4 K RAM area to hold the cursor position, so this location MUST NOT be altered at any time by your calling program.

The output routine begins at 8303. Before using the output routine you must FIRST initialize the display by outputting a FORM FEED (CLEAR SCREEN) character. This will move the cursor to the top left-hand corner of the screen and remove any random patterns from the display.

The output drivers will accept the following special control characters for moving the cursor:

CTRL H (08)	Backspace cursor
CTRL I (09)	Horizontal Tab
CTRL J (0A)	Linefeed
CTRL K (0B)	Vertical Tab (Home)
CTRL L (0C)	Form feed (Clear Screen)
CTRL X (18)	Line Cancel

Other characters such as <CR> (carriage return) and SPACEBAR function in the conventional fashion.

NOTE: The Gremlin board uses EF1 to send to the 1802 the status of the Video Generator to prevent static appearing on the TV screen. If your Version 2.1 monitor appears to be "hung up" at 836E-836F, check your Super Elf and Expansion Board Modifications to be sure they have been correctly done, especially those affecting EF1.

BREAK ROUTINE

Also included with both I/O drivers is a Break routine at 8306 which can be used in conjunction with your program to sense the input of a special "abort" character. This routine scans input port 5 for the presence of either a SPACEBAR or CTRL @ typed.

When a SPACEBAR is typed, the monitor break routine enters a loop, waiting for a non-space character to be typed. This feature is useful to temporarily stop the output display for inspection. Typing a non-space character will cause output to resume.

When a CTRL @ is typed, the DF flag is set to 1. Your main program can then check for DF set and take the appropriate action to suspend the output if so. Thus, your calling program must make sure that DF is cleared (eg., via an ADI #00 instruction) prior to calling the break routine in the Monitor. Note that typing a SPACEBAR will not affect DF.

Monitor Version 2.0 also has a built-in pause routine which will cause the display to pause whenever the "I" key on the Hex Keypad is pressed. Releasing the key will cause output to resume. This feature is part of the Version 2.0 output driver and is automatic; it requires no special calling sequence.

2-PART CHRISTMAS CAROL

by
Lester Hands

Since the Q output of the 1802 is only one line, only one-part melodies can be synthesized and harmony is impossible, right? Wrong! Or, at least, partly wrong.

This program thinks it is a banjo. Pairs of notes are played in rapid succession so that the overall effect is of a melody plus a harmony part. The structure is so simple that it could be very easily altered to play three-part harmony.

1/4K memory is required, but if you have more, the program will run as written.

The tempo can be changed by altering 000C, and the "plucking speed" of the banjo is stored in location 0009.

```
 DR CODE          COMMENT
0000 90 B1 B2     Set high addresses of R(1), R(2)
0003 F8 1C A1     Address of Tone subroutine
0006 F8 31 A2     Address of Note Table
0009 F8 04 B3     Length of each note
000C F8 04 A3     Number of times each note pair
                  repeated
000F D1           Call tone subroutine
0010 12           Point to next note in table
0011 D1           Call tone subroutine
0012 22           Point to previous note in table
0013 23 83 3A 0F  Has tone pair been repeated
                  enough?
0017 12 12        Point to first of next tone pair
0019 30 0C        Loop back
001B D0.          Return
001C 93 B4        Store length of note in R(4)
001E 02 32 30     Load note, continue if not zero
0021 A5           Store note in R(5)
0022 24 25        Decrement
0024 94 32 1B     Is note length done?
0027 85 3A 22     Loop back if note delay not
                  finished
002A C5 7A 38 7B  Change Q
002E 30 1E        Loop back
0030 00           Note value was zero: halt
0031 2D 2D
0033 2D 2D
0035 23 2D
0037 23 2F
0039 23 35
003B 23 35
003D 28 2F
003F 28 3C
0041 2D 35
0043 2D 43
0045 21 35
0047 21 2D
0049 21 2F
004B 21 3C
004D 23 2D
004F 23 2D
0051 28 2D
0053 28 2F
0055 23 2D
0057 23 28
0059 1E 2D
005B 1E 35
005D 1E 2F
005F 1E 3C
```

```
ADDR CODE        ADDR CODE
0061 20 35       008B 2F 3C
0063 20 50       008D 23 3C
0065 1E 3C       008F 23 35
0067 1E 3C       0091 28 2F
0069 1E 3C       0093 28 3C
006B 1E 3C       0095 2D 35
006D 23 2D       0097 2D 43
006F 23 2D       0099 2D 3C
0071 1A 43       009B 2D 35
0073 1A 3C       009D 2F 3C
0075 1A 35       009F 2F 43
0077 1E 35       00A1 2D 47
0079 21 2F       00A3 2D 50
007B 21 28       00A5 2D 5A
007D 23 2D       00A7 2D 5A
007F 23 2D       00A9 2D 5A
0081 28 2F       00AB 2D 5A
0083 28 3C       00AD 00
0085 2D 35
0087 2D 35
0089 2F 3C
```

MEMORY DUMP

by
Al A. Williams WD5GNR

The following program is what I feel is an improved version of the "Memory Recaller" presented in Questdata Vol. 1, Issue #3. It runs in less memory and faster than the previous program. The program dumps each byte of memory (starting with 00) to the hex display. The memory is in no way changed by running any of the program. I am including several modifications to make it more useful. You may change the pause between bytes by changing the number in location 05.

Alternatively, in a one page system you may remove address 01 to save space. Then, of course, all addresses after 00 must be decremented. If you are placing this in another memory block use an LDI 00 and renumber accordingly. You can easily change the starting address prior to execution by inserting an LDI between the B1 and A1 at location 01-02. Other modifications are apparent and trivial (i.e. single stepping under push button control, change of starting address by the keypad when running, stop bytes or addresses, etc., etc.).

```
ADDR CODE    OPCODE  COMMENT
0000 90      GHI R0  ;put 0 in D- use only in first
                      page
0001 A1      PHI R1  ;put D (0) in R1.1
0002 B1      PLO R1  ;put D (0) in R1.0
0003 E1      SEX R1  ;let X=1
0004 64      OUT 4   ;output byte - R1+1
0005 F8 60   LDI 60  ;load delay
0007 B2      PHI R2  ;put in R2
0008 22      DCR R2  ;decrease R2 by 1
0009 92      GHI R2  ;get delay count
000A 3A 08   BNZ 08  ;if delay does not = 0 do it
                      again
000C 30 04   BR 04   ;get next byte
```


18 QUESTDATA
P.O. Box 4430
Santa Clara, CA 95054

ADDRESS CORRECTION REQUESTED

BULK RATE
U.S. Postage Paid
QUEST
Electronics
Permit No. 549
Santa Clara, CA

NE

48063