

Questdata

Volume 3 Issue # 4

©

LUNAR LANDER

by
Gilbert Hamner

This game, using three pages of memory, provides a way to add two-dimensional movement to your display. This is done by using a control "bit" to point to the location to begin drawing on the screen. I got the idea for this method after reading the "Doodle Program" (Questdata #8) and thought it would be a simple way to point to a location on the screen. The control bit points to the upper left corner of a figure to be drawn. For this program it is a one byte wide by five byte high spacecraft. Two bytes per line are actually used for the display since the figure is shifted right to the location of the "bit" within the control byte.

A lunar lander, with you controlling its motion, attempts to make a landing. It moves downward from the pull of gravity with upward or sideways motion controlled by firing "rockets". This motion is simulated by incrementing movement counts each time through the program so that each successive time the spacecraft is drawn it is moved a greater distance.

How the Program Works

The program uses input from the keyboard to control the movement of the spaceship. A "0" turns the rockets off. A "1" moves the ship left and a "2" moves it right. Any other input will move the spaceship up. Only one rocket can be fired at a time. The input/movement subroutine takes the input and increments the appropriate movement count. The up movement is subtracted from the down movement and the difference is used to move the control byte either up or down. The left movement is subtracted from the right movement with the difference used to shift the control "bit" either left or right. Once the new location for the spaceship has been determined the program goes to the display subroutine. The old location of the spaceship is first cleared. The first byte of the new location, set by the control byte, is checked to see if it is already occupied by the surface line. If it is, the temporary memory address is set to zero and the

subroutine returns to the main program. If the location is not occupied, the data byte is drawn into the location and shifted right to the location of the control "bit". The display location is moved down one line and the check for occupancy and drawing is repeated. This continues until the spaceship is completely drawn. The main program is returned to and first checks if the temporary memory address is zero (the surface is hit) and branches to draw the crash if it is. This is done by first moving the display address up six lines, since it is at the surface, and pointing to the crash data and then going to the draw routine. If the spaceship was completely drawn the next line is checked (the display address is pointing there) to see if it is the surface. If it is blank, the program branches to a delay and then goes back to get another input. A check is made of the movement and if there is no left/right movement and the downward movement is 2 or less, a man is drawn and the program resets, otherwise it draws a crash and resets.

An added feature to the input/movement subroutine is a fuel count which is decremented each time through the program if any of the rocket inputs have been pressed. The count is shown on the hex display. When zero is reached, the "Q" comes on and the keyboard inputs are ignored with the spaceship falling to the surface.

Playing the Game

Load the program and C0 04 00 into the start of page 0 since the program begins on page 4. Push Run and the screen will show the display page. Pushing Input will clear the screen and draw the landing surface and the spaceship in the upper left corner. Push an initial rocket control, either the 0, 1, or 2 key, since an "up" key will roll the spaceship through the top of the screen and into the bottom resulting in a crash being drawn. Push the Input again to start the spaceship motion. Pushing the various movement

keys will control the spaceship. You must stop the sideways motion and slow the downward motion for a landing. If you take too long and run out of fuel, the Q light comes on and the spaceship will fall to the surface. The spaceship can move through the sides of the screen but a crash will be drawn if it moves out the top. Whether you make it or crash, pushing the Input will reset the game for another try. To change the difficulty of the landing, the fuel count at location 0448 or the down movement at location 047F can be changed. Although a score counter is not in the program, you could try and see the minimum amount of fuel you use to land the spaceship.

Going Further

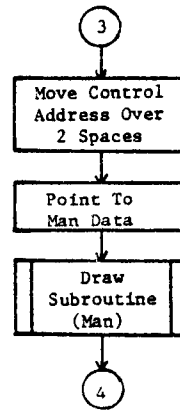
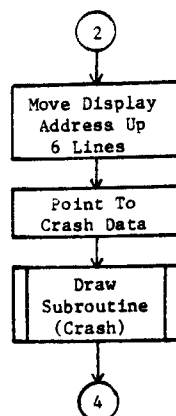
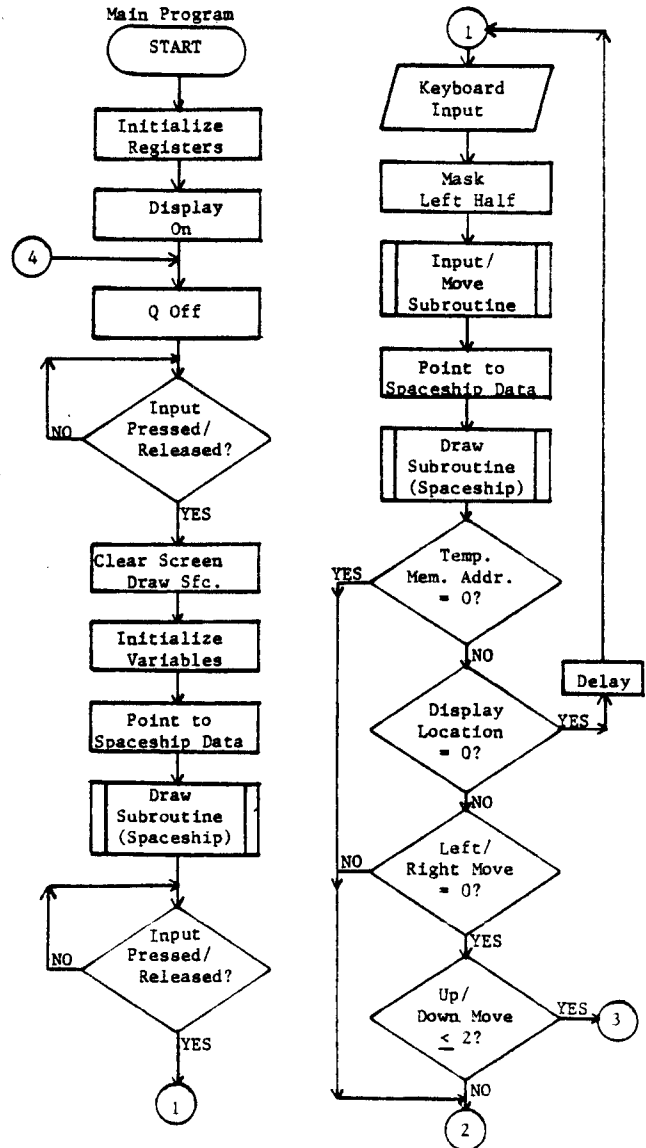
This program can be moved to any other pages of memory by simply changing the high order bytes at initialization. I used the input/movement subroutine for ease of understanding the program. If an additional register is needed, the subroutine can be easily incorporated into the main program. You can draw your own spacecraft of one to five bytes or adjust the display count and display location resets for a spaceship of any size. There should be sufficient room remaining in the two program pages to accommodate various modifications.

This program adds another dimension to video animation. I would like to hear from anyone who has ideas on modifying this program or using the above technique to animate other programs.

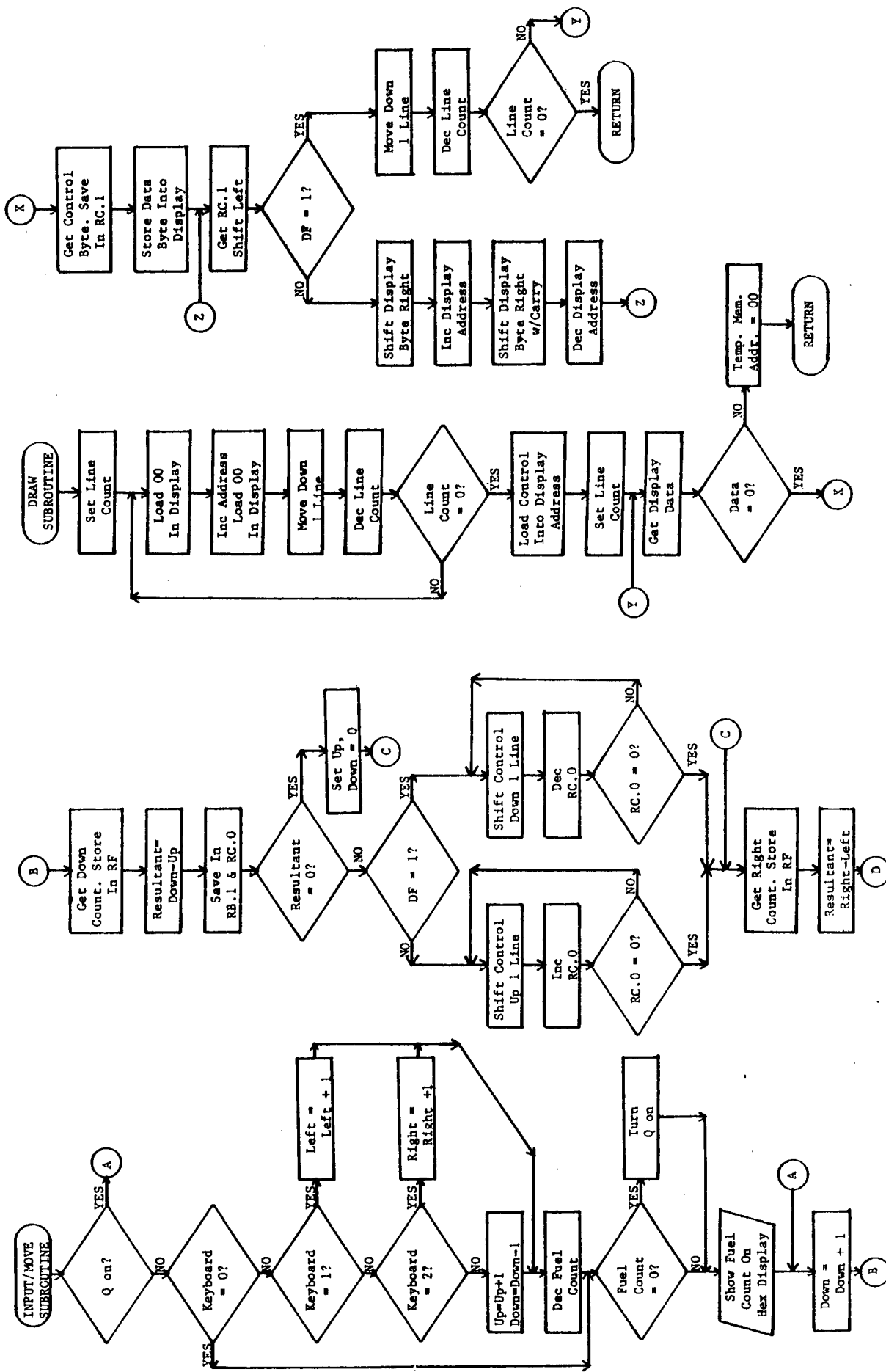
Gilbert Hemmer
236 Sacramento St. #8
Auburn, Ca. 95603

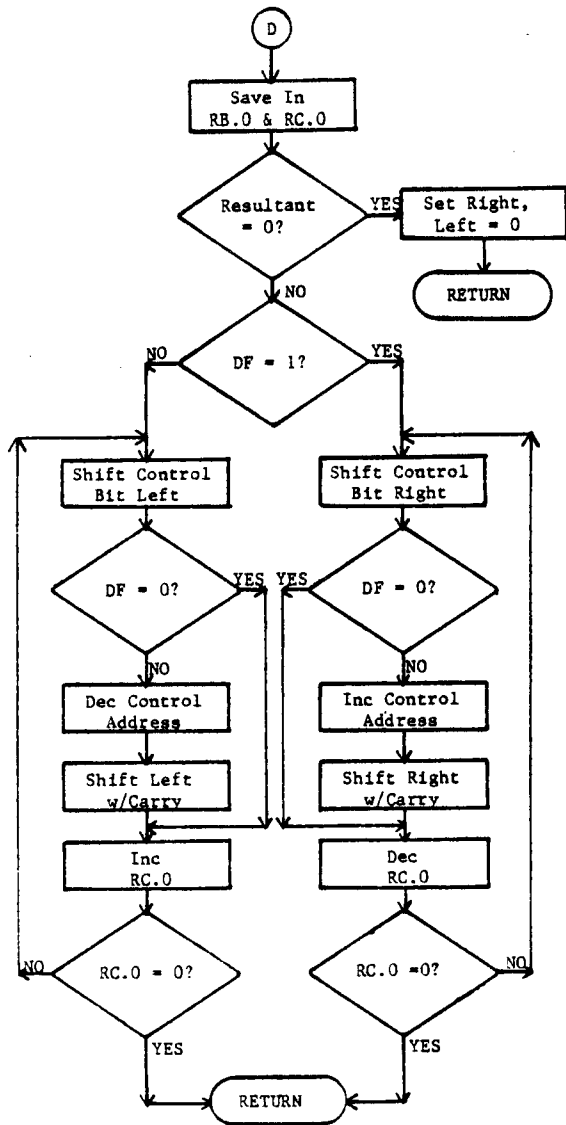
Register Assignments

- | | |
|---------------------------|-------------------------|
| R0= DMA | R9= Draw Subroutine |
| R1= Interrupt | RA.0=Control Bit |
| R2= Stack | RA.1=Keyboard Input |
| R3= Main | RB.0=Left/right Count |
| R4= Display Page | RB.1=Up/down Count |
| R5= Control Location | RC= Variable Counter |
| R6= Fuel Count | RD= Left/right Movement |
| R7= Unused | RE= Up/down Movement |
| R8= Input/move Subroutine | RF= Temporary Storage |



Quest Electronics Documentation and Software by Roger Pflin is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License.





Main Program

Loc.	Code	Mnem.	Comments
0400	F8 04	LDI	SET UPPER
0402	B1 B2	PHI	REGISTERS
0404	B3	PHI	
0405	F8 05	LDI	
0407	B8 B9	PHI	
0409	BF	PHI	
040A	F8 06	LDI	SET DISPLAY
040C	B4 B5	PHI	PAGE
040E	F8 D0	LDI	SET INTERRUPT
0410	A1	PLO	COUNTER
0411	F8 F0	LDI	SET INTERRUPT
0413	A2	PLO	STACK
0414	F8 01	LDI	SET INPUT/MOVE
0416	A8	PLO	SUBROUTINE
0417	F8 81	LDI	SET DRAW
0419	A9	PLO	SUBROUTINE
041A	F8 1E	LDI	SET MAIN
041B	A3	PLO	PROGRAM
041D	D3	SEP	SET P
041E	E2	SEX	SET X
041F	69	INP	DISPLAY ON
0420	7A	REQ	RESET Q
0421	3F 21	BN4	WAIT FOR INPUT PRESSED
0423	37 23	B4	WAIT FOR INP. RELEASED
0425	E4	SEX	SET X
0426	F8 FF	LDI	CLEAR DISPLAY
0428	A4	PLO	PAGE

Loc.	Code	Mnem.	Comments
0429	F8 00	LDI	
042B	73	STXD	
042C	84	GLO	
042D	3A 29	BNZ	
042F	F8 FF	LDI	DRAW LINE
0431	A4	PLO	ALONG
0432	F8 FF	LDI	BOTTOM
0434	73	STXD	
0435	84	GLO	
0436	FF F8	SMI	
0438	33 32	BDF	
043A	F8 00	LDI	SET DISPLAY/CONTROL
043C	A4 A5	PLO	ADDRESSES
043E	AE AD	PLO	SET UP/LEFT/DOWN
0440	BE	PHI	=0
0441	F8 04	LDI	SET RIGHT MOVEMENT
0443	BD	PHI	=4
0444	F8 80	LDI	SET CONTROL
0446	AA	PLO	BIT
0447	F8 20	LDI	SET FUEL
0449	A6	PLO	COUNT
044A	F8 F0	LDI	POINT TO SPACESHIP
044C	AF	PLO	DATA
044D	D9	SEP	GO TO DRAW SUBROUTINE
044E	3F 4E	BN4	WAIT FOR INPUT PRESSED
0450	37 50	B4	WAIT FOR INPUT RELEASE
0452	85	GLO	PUT CONTROL ADDRESS
0453	A4	PLO	INTO DISPLAY ADDRESS
0454	E2	SEX	SET X
0455	6C	INP4	READ KEYBOARD
0456	FA 0F	ANI	MASK LEFT HALF
0458	BA	PHI	OF BYTE
0459	D8	SEP	GO TO INPUT SUBROUTINE
045A	30 70	BR	

0470	F8 F0	LDI	POINT TO SPACESHIP
0472	AF	PLO	DATA
0473	D9	SEP	GO TO DRAW SUBROUTINE
0474	8F	GLO	GET ADDRESS VALUE
0475	32 84	BZ	BRANCH IF ZERO
0477	04	LDN	GET DISPLAY BYTE
0478	32 96	BZ	BR. NOT OCCUPIED
047A	8B	GLO	GET LEFT/RIGHT MOVE.
047B	3A 84	BNZ	BRANCH NOT ZERO
047D	9B	GHI	GET UP/DOWN MOVE.
047E	FF 02	SMI	SUBTRACT 2
0480	32 8F	BZ	BRANCH IF ZERO
0482	3B 8F	BNF	BR. IF NEGATIVE
0484	F8 FA	LDI	POINT TO CRASH
0486	AF	PLO	DATA
0487	84	GLO	GET DISPLAY ADDRESS
0488	FF 28	SMI	MOVE UP 6 LINES
048A	A4 A5	PLO	SAVE LOCATION
048C	D9	SEP	DRAW CRASH
048D	30 20	BR	BRANCH TO RESET
048F	15 15	INC	MOVE CONTROL ADDR.
0491	85	GLO	GET CONTROL ADDR.
0492	A4	PLO	PUT INTO DISPLAY ADDR.
0493	D9	SEP	DRAW MAN
0494	30 20	BR	BRANCH TO RESET
0496	F8 30	LDI	DELAY
0498	BC	PHI	
0499	2C	DEC	
049A	9C	GHI	
049B	3A 99	BNZ	
049D	30 52	BR	BR. TO CONTINUE

Input/Move Subroutine

Loc.	Code	Mnem.	Comments
0500	D3	SEP	RETURN
0501	EF	SEX	SET X
0502	F8 FF	LDI	POINT TO TEMP.
0504	AF	PLO	MEM. STORAGE
0505	31 29	BQ	BRANCH Q ON
0507	9A	GHI	GET KEYBOARD VALUE
0508	32 22	BZ	BRANCH IF ZERO
050A	FB 01	XRI	CHECK IF = 1
050C	32 1A	BZ	IF SO, BRANCH
050E	9A	GHI	GET KEYBOARD VALUE
050F	FB 02	XRI	CHECK IF = 2
0511	32 1D	BZ	IF SO, BRANCH
0513	1E	INC	INC UP MOVE COUNT
0514	9E	GHI	GET DOWN MOVE COUNT

Loc.	Code	Mnem.	Comments	Loc.	Code	Mnem.	Comments				
0515	FF 01	SMI	SUBTRACT 1	Draw Subroutine							
0517	BE	PHI	AND SAVE	0580	D3	SEP	RETURN				
0518	30 21	BR	BRANCH	0581	F8 05	LDI	SET LINE				
051A	1D	INC	INC LEFT MOVE COUNT	0583	AC	PLO	COUNT				
051B	30 21	BR	BRANCH	0584	F8 00	LDI	CLEAR OLD				
051D	9D	GHI	GET RT. MOVE COUNT	0586	54	STR	SPACESHIP				
051E	FC 01	ADI	ADD 1	0587	14	INC					
0520	BD	PHI	AND SAVE	0588	54	STR					
0521	26	DEC	DEC FUEL COUNT	0589	84	GLO					
0522	86	GLO	GET FUEL COUNT	058A	FC 07	ADI					
0523	3A 26	BNZ	BRANCH NOT ZERO	058C	A4	PLO					
0525	7B	SEQ	TURN Q ON	058D	2C	DEC					
0526	5F	STR	STORE FUEL COUNT	058E	8C	GLO					
0527	64	OUT4	DISPLAY FUEL COUNT	058F	3A 84	BNZ					
0528	2F	DEC	DEC TEMP. MEM. LOC.	0591	85	GLO	GET CONTROL ADDRESS				
0529	9E	GHI	GET DOWN MOVEMENT	0592	A4	PLO	PUT INTO DISPLAY ADDR.				
052A	FC 01	ADI	ADD 1 FOR GRAV.	0593	F8 05	LDI	SET LINE				
052C	BE	PHI	AND SAVE	0595	AC	PLO	COUNT				
052D	9E	GHI	GET DOWN MOVEMENT	0596	04	LDN	GET DISPLAY BYTE				
052E	5F	STR	STORE IN TEMP. MEM.	0597	3A B6	BNZ	BRANCH IF OCCUPIED				
052F	8E	GLO	GET UP MOVEMENT	0599	8A	GLO	GET CONTROL BIT				
0530	F5	SD	SUBTRACT FROM DOWN	059A	BC	PHI	STORE IN TEMP. COUNTER				
0531	AC	PLO	SAVE IN COUNTER	059B	4F	LDA	GET DATA BYTE				
0532	BB	PHI	AND RESULTANT	059C	54	STR	STORE IN DISPLAY				
0533	3A 39	BNZ	BRANCH NOT ZERO	059D	9C	GHI	GET CONTROL BIT				
0535	AE	PLO	OR SET UP/DOWN	059E	FE	SHL	SHIFT LEFT				
0536	BE	PHI	TO ZERO	059F	BC	PHI	SAVE				
0537	30 4D	BR	BRANCH	05A0	33 AC	BDF	BRANCH IF DF = 1				
0539	33 45	BDF	BRANCH IF POSITIVE	05A2	04	LDN	GET DISPLAY BYTE				
053B	85	GLO	GET CONTROL ADDRESS	05A3	F6	SHR	SHIFT RIGHT				
053C	FF 08	SMI	MOVE UP ONE LINE	05A4	54	STR	STORE				
053E	A5	PLO	AND SAVE	05A5	14	INC	INC DISPLAY LOCATION				
053F	1C	INC	INC UP COUNT	05A6	04	LDN	GET DISPLAY BYTE				
0540	8C	GLO	GET UP COUNT	05A7	76	SHRC	SHIFT RIGHT W/CARRY				
0541	3A 3B	BNZ	BR. BACK IF NOT ZERO	05A8	54	STR	STORE				
0543	30 4D	BR	BRANCH	05A9	24	DEC	RESET DISPLAY LOC.				
0545	85	GLO	GET CONTROL ADDRESS	05AA	30 9D	BR	BRANCH				
0546	FC 08	ADI	MOVE DOWN ONE LINE	05AC	84	GLO	GET DISPLAY ADDRESS				
0548	A5	PLO	AND SAVE	05AD	FC 08	ADI	MOVE DOWN ONE				
0549	2C	DEC	DEC DOWN COUNT	05AF	A4	PLO	LINE				
054A	8C	GLO	GET DOWN COUNT	05B0	2C	DEC	DEC LINE COUNT				
054B	3A 45	BNZ	BR. BACK IF NOT ZERO	05B1	8C	GLO	GET LINE COUNT				
054D	9D	GHI	GET RIGHT MOVEMENT	05B2	3A 96	BNZ	BRANCH NOT ZERO				
054E	5F	STR	STORE IN TEMP. MEM.	05B4	30 80	BR	BRANCH TO RETURN				
054F	8D	GLO	GET LEFT MOVEMENT	05B6	F8 00	LDI	SET TEMP. MEM. ADDR.				
0550	F5	SD	SUBTRACT FROM RIGHT	05B8	AF	PLO	TO ZERO				
0551	AC	PLO	SAVE IN COUNTER	05B9	30 80	BR	BRANCH TO RETURN				
0552	AB	PLO	AND RESULTANT	Display Subroutine							
0553	3A 59	BNZ	BRANCH NOT ZERO	Loc.	Code	Mnem.	Comments	Data			
0555	AD	PLO	OR SET LEFT/RIGHT	04CE	72	LDXA	INTERRUPT	05F0 18	SPACESHIP		
0556	BD	PHI	TO ZERO	04CF	70	RET	RETURN	05F1 3C			
0557	30 00	BR	BRANCH TO RETURN	04D0	22	DEC		05F2 FF			
0559	33 68	BDF	BRANCH IF POSITIVE	04D1	78	SAV		05F3 24			
055B	8A	GLO	GET CONTROL BYTE	04D2	22	DEC		05F4 42			
055C	FE	SHL	SHIFT LEFT	04D3	52	STR		05F5 04	MAN		
055D	3B 61	BNF	BRANCH DF=0	04D4	C4 C4	NOP	NOPs FOR	05F6 1F			
055F	25	DEC	DEC CONTROL ADDRESS	04D6	C4	NOP	SYNC.	05F7 04			
0560	7E	SHLC	SHIFT LEFT W/CARRY	04D7	F8 06	LDI	SET DMA	05F8 0A			
0561	AA	PLO	SAVE CONTROL BYTE	04D9	B0	PHI	POINTER	05F9 11			
0562	1C	INC	INC COUNTER	04DA	F8 00	LDI		05FA 24	CRASH		
0563	8C	GLO	GET COUNTER	04DC	A0	PLO		05FB 81			
0564	3A 5B	BNZ	BRANCH NOT ZERO	04DD	80	GLO	INTERRUPT	05FC 24			
0566	30 00	BR	BRANCH TO RETURN	04DE	E2	SEX	ROUTINE	05FD 42			
0568	8A	GLO	GET CONTROL BYTE	04DF	E2	SEX		05FE 18			
0569	F6	SHR	SHIFT RIGHT	04E0	20	DEC		05FF xx	TEMP. STORAGE		
056A	3B 6E	BNF	BRANCH DF = 0	04E1	A0	PLO					
056C	15	INC	INC CONTROL ADDRESS	04E2	E2	SEX					
056D	76	SHRC	SHIFT RT. W/CARRY	04E3	20	DEC					
056E	AA	PLO	SAVE CONTROL BYTE	04E4	A0	PLO					
056F	2C	DEC	DEC COUNTER	04E5	E2	SEX					
0570	8C	GLO	GET COUNTER	04E6	20	DEC					
0571	3A 68	BNZ	BRANCH COUNT NOT ZERO	04E7	A0	PLO					
0573	30 00	BR	BRANCH TO RETURN	04E8	3C DD	BN1	BR. TO INTERRUPT				
				04EA	30 CE	BR	BRANCH TO RETURN				
				04EC	00 00		STACK AREA				
				04EE	00 00						
				04F0	00						

Q*BUG

In a previous column, we added some new commands and shortened some other command names. Four of the shortened commands require a letter designator after the command word to establish the physical means of data input or output. These commands are:

LOAD (X) X = Required letter designator
 SAVE (X) C = Cassette I/O
 D/L (X) F = Floppy Disk I/O
 D/S (X) S = Stringy Floppy I/O

If you are not fortunate enough to have a Stringy Floppy and/or Floppy Disk I/O device interfaced to your computer, how would you like to eliminate the typing of the letter designator? If you use only a cassette for data I/O, the use of a letter designator is redundant and a simple fix will do away with it.

First, a little background on what Super does when you type commands such as "LOAD C". In processing an inputted command, Super puts the command, byte by byte, into the line buffer area on work page 3500. The buffer area is location 3500 thru 355F. Thus, "LOAD C", in the direct execution mode, would appear in the line buffer as:

Location	Code
3500	4C (L)
3501	4F (O)
3502	41 (A)
3503	44 (D)
3504	20 (Space)
3505	43 (C)
3506	0D (c/r)

Super will then "read" the command word in the line buffer and search the command table for a matching word. If Super finds a match, it will "stuff" the token for the command into the first position in the intermediate line buffer area at location 35D0 on work page 3500. Super then examines the byte following the command word in the line buffer. In our illustration, this is a space (20) and Super will ignore this space. (This is one way OUR Basic saves space in memory. The TRS 80 and some other Basics write ALL spaces into their program memory.)

Super examines the next byte in the line buffer (43) and determines its acceptability as a valid ASCII letter. Since "43" is the ASCII letter "C", Super assigns a prefix designator of "D1" to the "43" and stuffs both bytes into the next positions in the intermediate line buffer.

"LOAD C" will now appear in the intermediate line buffer as:

Location	Code
35D0	98 (Token for LOAD)
35D1	D1 (Letter prefix)
35D2	43 (C)
35D3	0D (c/r)

After Super has loaded the command to the intermediate line buffer, it then examines the intermediate line buffer to determine what it should do. The first byte (token 98) "points" Super to the execution table location 0730. At this location, Super picks up the address of the actual "LOAD" machine language routine, which is 0E00, and goes to that location to perform the LOAD routine.

In all of the LOAD, SAVE, D/L, or D/S, routines, Super uses a subroutine at location 14FE to test for the presence of the "D1" after the token in the intermediate line buffer. If Super finds the "D1", it examines the the next byte. Depending on the letter found, Super will stuff either "32 06" (for C) or "F8 06" (for F or S) into locations 35A0 and 35A1 on work page 3500. These locations are the statement call address to which Super will branch later on in the LOAD or SAVE routines.

If Super does not find a D1 after the token in the intermediate line buffer, it will return an error message (#60). When LOAD is typed without the letter designator, the intermediate line buffer will look like this:

35D0	98
35D1	0D

To keep Super from returning an error message if we do not use a letter designator, we must bypass the test for the D1 in the subroutine at location 14FE.

This is the present routine at location 14FE with my homemade comments:

Location	Code	Comment
14FE	4B	Get byte AFTER token byte at 35D0
14FF	FB D1	Test for "D1" if yes, D will = 0
1501	32 07	If D = 0, branch to 1507
1503	D4 09 FF	Error routine - do if D <> 0
1506	3C	Garbage byte - leftover assem-

bler label?

```

1507 BA Put D (00) in register BA.1
1508 4B Get next byte at location 35D2
1509 FB 43 Test for "C" if yes, D will = 0
150B 3A 15 If not "C" branch to 1515
150D F8 E1 Put E1 in D
150F AA Put D in register BA.0 - BA =
00E1
1510 0A Get byte at location 00E1 -
should be "32"
1511 DD 20 Put "32" in location 35A0 (3580
+ 20)
1513 30 2B Branch to 152B
1515 FB 05 Test for "F" (43 XOR 5 = 46(F))
1517 3A 21 If not "F" branch to 1521
1519 F8 E2 Put E2 in D
151B AA Put D in register BA.0 - BA now
= 00E2
151C 0A Get byte at location 00E2 (F8)
151D DD 20 Put "F8" in location 35A0
151F 30 2B Branch to 152B
1521 FB 15 Test for "S" (46 XOR 15 = 53(S))
1523 3A 03 If not S branch back to 1503
(error)
1525 F8 E3 Put E3 in D
1527 AA Put D in register BA.0 - BA now
= 00E3
1528 0A Get byte at location 00E3 (F8)
1529 DD 20 Put in location 35A0
152B F8 06 Put 06 in D
152D DD 21 Put 06 in location 35A1 (If
letter designator was "C", lo-
cations 359F thru 35A2 will now
read "D4 32 06 D5". If "F" or
"S", it would read "D4 F8 06
D5". Location 359F thru 35A2 is
the call to the actual LOAD or
SAVE routines. Location 3206 is
the cassette I/O routines.)
152F AF Also put 06 in register RF.0
1530 0A Get byte at location 00E1 for
"C" or location 00E2 for "F" or
location 00E3 for "S"
1531 BF Put byte in register BF.1 - 32
for C or F8 for F or S
1532 0F Get byte at location pointed to
by register BF - 3206 for C,
F806 for F or S
1533 FB C0 Test for C0 - location 3206
DOES contain C0 - location F806
may or may not contain C0
1535 CA 0C C2 If the byte is not C0, an error
message "N/A" will be printed
1538 D5 Return to calling location

```

To bypass the part of this routine that checks for the D1 and letter is simply a matter of changing the first five byte at location 14FE to:

```
14FE F8 00 BA 30 0D
```

Now, Super will immediately stuff "32 06" into the call location and go right to work LOADING or SAVEing.

One word of caution, however, is in order. If you are running a program that was written WITH a letter designator, Super will treat this as an error and abort the program AFTER the D/L or D/S. It will perform the Data Save or Data Load correctly.

My advice is to review any programs that have DSAVE, D/S, DLOAD, or D/L, and remove the letter designator. One final change! Let's put a shorthand command for LOAD or SAVE in the command table. Make the following changes:

Location	Code
06EE	63
06EF	4C
06F0	CC
06F1	98
06F2	63
06F3	53
06F4	D3
06F5	97

Now, LOAD or LL will load a cassette, SAVE or SS will dump to a cassette.

Finally, make a new master Super program tape.

Keep a permanent record of the original routine at location 14FE. Should you ever wish to interface a Stringy Floppy or Floppy Disk, you will probably have to restore the routine to its original state.

Now, for ELF II owners, let's shrink the serial version of Super to a promable length of 12 1/4 K.

In a previous column, we eliminated the need for part of the initialization routines at location 1800 thru 183F and 3493 thru 34CC. We will now use these areas for the extra patch routines required by the ELF II Video Board. (Page 71 in the Super manual)

The first patch shown on page 71 is:

```
3300: C0 34 93
```

This is the terminal timing routine which we previously eliminated by putting "C0 31 48" at location 3300. This change will remain as is but will go in a different location.

The second patch on page 71 is:

```
3303: C0 34 F3
```

This is the Break routine at location 34F3. We will put this routine on page 3300 but at a different location, also.

The next patch is:

```
3306: C0 33 1C
```

This is a branch to an ELF II required output patch routine at location 331C as detailed on page 71 of the manual. This routine is performed BEFORE the output routine at location 3406. We will put this patch routine into the free area at location 1800 as follows:

```
1800: BF 12 12 12 02 FB 0F 3A
1808: 2B 72 A6 02 B6 9F D5 22
1810: 22 22 9F 73 DD 9B FF 41
1818: 3B 3D D4 2F F9 F8 01 DD
1820: 1B 12 02 D4 34 06 D5
```

Change the branches at locations 1807 and 1818 and the subroutine call at location 1823 as follows:

```
1807: 3A 0F
1818: 3B 21
1823: D4 33 16 D5
```

At location 3306, change the long branch to:

```
3306: C0 18 00
```

The last patch on page 71 is:

```
3309: D4 34 09
330C: D4 2E 3B (etc)
```

This is an input patch routine. The routine from location 330C thru 331B is performed AFTER the input routine at location 3409.

We will start this patch routine by putting a long branch to the remaining free area at location 1827 as follows:

```
3309: C0 18 27
```

Starting at location 1827, insert the following:

```
1827: D4 33 19 D4 2E 3B 73 DD 9B
1830: FF 40 3B 37 D4 2F F9 12
1838: 02 D5 C4 C4 C4 C4 C4 C4
```

This is the patch originally shown on page 71 for location 3309 thru 331B. I have already corrected the branches for you. The line length is set for 64 characters on the CRT screen so if you use a different length, make the appropriate changes to the bytes at locations 1817 and 1831.

Now, change location 3300 thru 3305 to:

```
3300: C0 33 0C C0 33 10
```

Put the CLS branch at 330C:

```
330C: D4 31 48 D5
```

Put the Break routine at location 3310:

```
3310: 3F 14 FF 00 D5
```

This finishes the relocation of the ELF II patches. Now, we must move the actual input, output, and delay routines, from page 3400 to page 3300. First, block move the input/output routine from location 3406 thru 3492 to location 3316 thru 33A2. When this is done, make the following branch changes:

```
3317: 63
331A: 35
3339: 38
333E: 40
3347: 4D
334C: 53
3350: 52
3357: 44
335F: 35
3362: 98
3369: 6D
3380: 83
338B: 7F
3390: 95
339A: 72
```

Next, block move the delay routine from location 34CD thru 34F2 to location 33A3 thru 33C8. Make the following branch changes:

```
33A9: A6
33AF: AC
33B4: C2
33BC: B9
33C1: B3
33C6: A3
33C8: AA
```

You might find, as I did, that this shifting of routines somehow messes up the terminal timing and you get a double echo of the typed character on your CRT. In my case, I simply changed the time constants on work page 0000 at locations 00E7 and 00E8. They changed from "80 55" to "00 55". This cleared up my problem but you may have to experiment with different values.

Now, if everything is working to your satisfaction, make a new master Super program tape.

LINE by LINE

by

Werner Cirsovius

ASSEMBLER

This simple Assembler supports the development of machine written programs by translating mnemonic code into hexadecimal machine code. The length of this program is about 4.75k bytes, including string and array areas. The assembler is software protected, so any try to write out of the allocated Ram space will result in an error message and break.

NOTE: The available Ram for machine code is above the Basic program. To use the hole created by the 'DEFUS' statement, a short program for moving data into the hole must be written by the user.

This assembler supports:

- all standard RCA mnemonics
- two Standard Call and Return Technique mnemonics
- three pseudo opcodes
- decimal, hexadecimal and character operands (but no labels)
- multiple mnemonics per line separated by a blank

OPERANDS

Operands may be decimal, hexadecimal or characters. If the pseudo opcode DC' (Define constant) is used, string operands are legal, too. While decimal operands need no prefix, the hexadecimal operands use the hash sign (#) and the characters/strings the dollar (\$) sign.

Examples: 10 Decimal 10
 #10 Hex 10 (Decimal 16)
 \$A Character 'A' (Hex 41, decimal 65)

MNEMONICS

All available mnemonics are stored in a data field (Lines 2000-2990). Each mnemonic uses three data statements:

- Statement 1 : Mnemonic string
- Statement 2 : Line for executing the type of mnemonic
- Statement 3 : Hexadecimal Opcode

The last statement in the data list is the asterisk (*), indicating the end of the list. If the asterisk is found, the assembler prints:

'UNKNOWN MNEMONIC'

Description of the various modes.

MODE 1: Opcode with no operand (Line 300)

Format : /Mnemonic/

Examples: IDL NOP

Code generated: 00 C4

MODE 2: Opcode with Register operand (Line 200)

Format : /Mnemonic// (Prefix) Operand/ With operand in range 0..15 (decimal) or 0..F (hex)

Examples: SEX 2 SEP #A

Code generated: E2 DA

MODE 3: Opcode with immediate operand (Line 400)

Format : /Mnemonic/ / (Prefix) Operand/ With operand in range 0...255

Examples: LDI 16 ADI #FE (RI \$0

Code generated: F8 10 FC FE F9 30

MODE 4 : Opcode with 8 bit pointer (Branches) (Line 600)

Format : /Mnemonic/ /Operand/ The operand is always in hex notation and needs no prefix

Examples: BR 10 BN4 D0 BGE 0

Code generated: 30 10 3F D0 33 00

MODE 5 : Opcode with 16 bit address (Long Branches) (Line 700)

Format : /Mnemonic/ /Operand/ The operand is always in hex notation as in mode 4

Examples: LBR 13 LBNF F00

Code generated: C0 00 13 CB F0 00

MODE 6 : Opcode with I/O operand (Line 500)

Format : /Mnemonic/(Prefix) Operand/
With operand in range 1..7

Examples: OUT 4 INP #7

Code generated: 64 6F

MODE 7 : Pseudo opcode 'ORG', set origin (Line 800)

Format : ORG /Operand/
With operand always in hex notation

Examples: ORG 100 ORG F000

Sets PC to : 0100 F000

MODE 8 : Pseudo opcode 'END', end of assembler (Line 900)

Format : END, prints the message 'END ON XXXX'
and enters Basic

MODE 9 : Pseudo opcode 'DC', define constant (Line 1900)

Format : DC /(Prefic) Operand/

If the operand is decimal (no prefix) and hexadecimal (prefix #), the numbers must be in the range 0..65535 (0000..FFFF). If the operand is less than 256, only 8 bits will be set up as constant, otherwise 16 bits are used. If the operand is a string (prefix \$) up to 32 characters in that string are legal.

Examp's: DC 200 DC #10 DC 256 DC #1000 DC \$ ABC

Constant
generated: C8 10 0100 100020 41 42 43

ERROR MESSAGES

Whenever an error is detected, the assembler prints an error message followed by the mnemonic line with a question mark behind the error string portion.

'UNKNOWN MNEMONIC ERROR'	If a mnemonic is not found in the list
'MISSING OPERAND ERROR'	If the assembler finds no operand
'OPCODE ERROR'	If the mnemonic 'LDN 0' is found (Generating code 00 which is used for mnemonic 'IDL')
'OPERAND ERROR'	If the register or I/O operands are out of range

'STRING ERROR'

If there are more than 32 characters in the string of a 'DC' opcode

'ERROR'

If an error in the operand field is detected. That is
-if a (hexa)decimal number is out of range
-if a digit is not 0..9 or A..F

'ADDRESS VIOLATION'

If the memory address is within the protected area. The assembler responds with 'ORG ' and enters the 'ORG' sequence

'MEMORY FULL'

If no more memory is available. This message results in an end of the assembler

MEMORY SPACE

As found in the Super Basic manuals, the following pages are stored in the workspace (Note that in the listing this is page 01, see lines 20 and 50. It will be page 2F in V3.0 and page 35 in V5.0).

XX99 Page for end of string variables

XXBB Stack page

These pages will be loaded and referred to as start-and-end-address.

Assume end of string variables is page '31' and the stack page '4F'.

The assembler prints at the beginning:

```
'FIRST ADDRESS AVAILABLE 3300'
'LAST ADDRESS AVAILABLE 4DFF'
```

To be sure no dynamic assignment of memory areas will crash the program, there are some free pages.

TABLE OF VARIABLES

A	Holds first available memory address
B	Length of hex string (2 for byte, 4 for 16 bits)
C	End flag of substring (1 if no end, 0 if end of string)
D	Length of decimal string (3 for byte, 5 for 16 bits)
E	Holds last available memory address
F	Range flag (0 if character 0..9,A..F, 1 otherwise)
H	Work variable

I,J For .. Next loop variables
 L Length of substring or executing mode line number
 M Current memory address
 N Current string pointer for line string
 O Opcode and operand counter
 T Work variable holds divisor for hex printing (Line 1200)
 U String pointer for string in 'DC' opcode
 V Nibble counter (2 if 8 bits, 4 if 16 bits)
 X Operand
 Y,Z Work variables

C(\$) Substring, holding mnemonic or operand
 H(\$) Mnemonic from data list
 O(\$) Constant text for error routine
 T(\$) Mnemonic and operand input string
 X(\$) Token string (Holds prefix # or \$)

O(32) Array for opcode/operand/constant

SAMPLEPROGRAM

In some applications it's useful to work with the Gray-code. To work with Gray-coded information, a converter must be used.

The conversion of Binary to Gray is easy:

-Load Accu with binary word

-Shift accu right one bit

-Exor accu with original binary word

-Store content of accu as Gray code

The conversion of Gray code to Binary is similar, but uses the above routine M-times, where M depends on the number of bits (N) in the following way:

$$N \leq 2^X \quad m = 2^X - 1$$

The assembler program consists of two parts: (Note the start address of hex '1600', this may differ from normal Super Basic versions)

Part 1 : 1600-1613 Calculate the number of loops (M) depending on the number of bits (N) in Lo register 8. The calculation starts with $x=1, 2^x=2$. If the comparison $N-2^x$ is less or equal 0, the result is $2^x - 1$

Part 2 : 1620-1626 Perform the conversion (One loop)

Registers used : Reg 2 as stack pointer
 Reg 8 as parameter register

The end of both parts is a SEP 5 (RTS) instruction.

SAMPLE RUN OF ASSEMBLER

(SUBROUTINES FOR GRAY CODE CONVERSION)

FIRST ADDRESS AVAILABLE @1600
 LAST ADDRESS AVAILABLE @2DFF
 ENTER MNEMONIC

```
1600:LDI 2 STR 2 GLO 8 SM
1600 FB 02
1602 52
1603 8B
1604 F7
1605:BL OE BZ OE LDN 2 SHL STR 2 BR 03
1605 3B OE
1607 32 OE
1609 02
160A FE
160B 52
160C 30 03
160E:LDN 2 SMI 1 PLO 8 RTS
160E 02
160F FF 01
1611 AB
1612 D5
1613:ORG 1620 GLO 8 STR 2 SHR XDR PLO 8 RTS
1620 8B
1621 52
1622 F6
1623 F3
1624 AB
1625 D5
1626:END
```

SAMPLE PROGRAM IN BASIC

This short program is the main program for the conversion of Gray code to Binary code and vice versa.

After initialization, the program asks for the mode. The user may select one of the four available modes with the following keys:

B Define number of bits to be converted. Basic calls the routine part 1 (Line 30) and prints:

'N BITS NEED M RUNS'

G Convert Gray to Binary

D Convert Binary to Gray

Both conversion modes print after execution:
 'CONVERTED FROM X TO Y'

E End of program

```
10 DEFINT Z
20 INPUT "BITS TO BE CONVERTED" B
30 M=USR(@1600,B,0)
40 PRINT B;" BITS NEED " ;M;" RUNS"
50 INPUT "MODE" M#
60 IF MID$(M#,1,1)="G" GOTO 110
70 IF MID$(M#,1,1)="D" GOTO 120
80 IF MID$(M#,1,1)="B" GOTO 20
90 IF MID$(M#,1,1)="E" GOTO 200
100 GOTO 50
110 L=M: GOTO 130
120 L=1
130 INPUT "WORD" W:H=W
140 FOR I=1 TO L:W=USR(@1620,W,0): NEXT
150 PRINT "CONVERTED FROM " ;H;" TO " ;W
160 GOTO 50
200 DEFINT
```

```

BITS TO BE CONVERTED?B
8 BITS NEED 7 RUNS
MODE?B
BITS TO BE CONVERTED?4
4 BITS NEED 3 RUNS
MODE?G
WORD?6
CONVERTED FROM 6 TO 4
MODE?D
WORD?4
CONVERTED FROM 4 TO 6
MODE?B
BITS TO BE CONVERTED?8
8 BITS NEED 7 RUNS
MODE?G
WORD?100
CONVERTED FROM 100 TO 71
MODE?D
WORD?71
CONVERTED FROM 71 TO 100
MODE?E
    
```

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

Publisher.....Quest Electronics
Editor.....Paul Messinger
QBUG Editor.....Fred Hannan
Proof Reading.....Judy Pitkin
Production.....John Larimer

The contents of this publication are copyright and shall not be reproduced without permission of QUESTDATA. Permission is granted to quote short sections of articles when used in reviews of this publication. QUESTDATA welcomes contributions from its readers. Manuscripts will be returned only when accompanied by a self addressed stamped envelope. Articles or programs submitted will appear with the authors name unless the contributor wishes otherwise. Payment is at the rate of \$15 per published page. QUESTDATA exists for the purpose of exchanging information about the RCA 1802 microcomputer.

```

5 REM LINE BY LINE ASSEMBLER
6 REM FOR THE COSMAC CDP 1802
7 REM BY W.CIRSOVIUS, JUNE 81
10 DEFINT Z: DIM O(32): CLS: PRINT : PRINT
15 O$="OPERAND"
19 H=2
20 A=(PEEK(@O199)+H)*256:M=A
30 C$=" ADDRESS AVAILABLE @"
40 PRINT "FIRST";C$: GOSUB 1200: PRINT
50 E=(PEEK(@O188)-1)*256-1:M=E
60 PRINT " LAST";C$: GOSUB 1200: PRINT
70 PRINT "ENTER MNEMONIC": PRINT :M=A
75 REM **MAIN LOOP**
80 GOSUB 1200: INPUT " :";T$:N=1
90 GOSUB 1100: RESTORE
100 READ H$: IF H$="" PRINT "UNKNOWN MNEMONIC": GOTO 1000
110 READ L,O(1): IF H$=C$ GOTO L
130 GOTO 100
200 REM **REGISTER OPERAND**
210 IF C=0 GOTO 999
220 GOSUB 1300: IF X>15 PRINT O$: GOTO 1000
240 O(1)=O(1)+X: IF O(1)=0 PRINT "OPCODE": GOTO 1000
300 REM **NO OPERAND**
305 O=1
320 IF M<A PRINT "ADDRESS VIOLATION": INPUT "ORG @":T$:N=1: GOTO 820
330 IF M>E PRINT "MEMORY FULL": GOTO 920
335 IF M+O>E+1 GOTO 330
336 GOSUB 1200: PRINT " ";
340 FOR I=1 TO O: POKE(M,O(I)):H=M+1
350 M=O(I): GOSUB 1200: PRINT " ";
360 M=H: NEXT I: PRINT
380 IF C<>0 GOTO 90
390 GOTO 80
400 REM **IMMEDIATE OPERAND**
410 IF C=0 GOTO 999
420 GOSUB 1300
430 O(2)=X:O=2: GOTO 320
500 REM **I/O CODE**
510 IF C=0 GOTO 999
520 GOSUB 1300: IF X=0 GOTO 550
540 IF X<8 GOTO 240
550 PRINT O$: GOTO 1000
600 REM **BRANCH**
610 B=2: GOTO 720
700 REM **LONG BRANCH**
710 B=4
720 IF C=0 GOTO 999
730 GOSUB 1100: GOSUB 1420: IF B=2 GOTO 430
750 O(2)=X/256:O(3)=X-O(2)*256:O=3: GOTO 320
800 REM **ORIGIN**
810 IF C=0 GOTO 999
820 B=4: GOSUB 1100: GOSUB 1420:M=X: GOTO 380
900 REM **END**
910 PRINT : PRINT : PRINT "END ON @": GOSUB 1200: PRINT
920 DEFINT
930 END
998 REM **ERROR HANDLER**
999 PRINT "MISSING OPERAND":
1000 PRINT " ERROR": PRINT
1020 FOR I=1 TO N-1: PRINT MID$(T$,I,1): NEXT
1030 PRINT "?": IF N=LEN(T$) GOTO 1050
1040 FOR I=N TO LEN(T$): PRINT MID$(T$,I,1): NEXT
1050 PRINT : GOTO 80
    
```

Quest Electronics Documentation and Software by Request Policy is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License.

```

1100 REM **GET SUBSTRING**
1110 FOR I=N TO LEN(T$)
1120 IF MID$(T$,I,1)=" " EXIT 1150
1130 NEXT :C=0
1140 C$=MID$(T$,N,I-N-C):N=I: RETURN
1150 C=1:I=I+1: GOTO 1140
1200 REM **PRINT HEX**
1205 V=4: GOTO 1210
1206 V=2
1210 X=M:T=16^(V-1): FOR J=1 TO V
1220 Y=X/T:Z=Y: IF Z>9Z=Z+7
1230 PRINT CHR$(Z+48);:X=X-Y*T:T=T/16: NEXT : RETURN
1300 REM **GET OPERAND**
1305 D=3:B=2
1310 GOSUB 1100:X$=MID$(C$,1,1):L=LEN(C$)
1320 IF X$="*" GOTO 1410
1330 IF X$="%" GOTO 1490
1335 REM **DECIMAL**
1340 IF L>D EXIT 1000
1350 X=0: FOR I=1 TO L
1360 D=ASC(MID$(C$,I,1)): GOSUB 1630
1370 IF F=1 EXIT 1395
1380 X=X*10+D-48: NEXT : RETURN
1395 EXIT 1000
1400 REM **HEX**
1410 C$=MID$(C$,2,LEN(C$))
1420 L=LEN(C$): IF L>B EXIT 1000
1430 X=0: FOR I=1 TO L
1440 D=ASC(MID$(C$,I,1)): GOSUB 1600
1450 IF F=1 EXIT 1395
1460 IF D>57D=D-7
1470 X=X*16+D-48: NEXT : RETURN
1490 REM **CHARACTER**
1500 X=ASC(MID$(C$,2,1)): RETURN
1600 REM **RANGE TEST**
1605 F=0: IF D>70F=1
1610 IF D<65 IF D>57F=1
1620 GOTO 1650
1630 F=0: IF D>57F=1
1650 IF D<48F=1
1660 RETURN
1900 REM **CONSTANT**
1910 IF C=0 GOTO 999
1915 O=LEN(T$)-N:U=N: IF O=0 GOTO 1925
1920 GOSUB 1100:X$=MID$(C$,1,1):L=LEN(C$): IF L>O GOTO 1930
1925 PRINT "CONSTANT": GOTO 1000
1930 IF X$="*"B=4: GOSUB 1410: GOTO 1960
1940 IF X$="%" GOTO 1980
1950 D=5: GOSUB 1340
1960 IF X<2560=1:O(1)=X: GOTO 320
1970 O=2:O(1)=X/256:O(2)=X-O(1)*256: GOTO 320
1980 IF O>32 PRINT "STRING": GOTO 1000
1990 FOR I=1 TO O:O(I)=ASC(MID$(T$,U+I,1)): NEXT :C=0: GOTO 320
2000 REM **MNEMONIC DATA**
2010 DATA "IDL",300,#00,"NOP",300,#C4,"SEP",200,#D0,"SEX",200,#E0
2020 DATA "SEQ",300,#7B,"REQ",300,#7A,"SAV",300,#7B,"MARK",300,#79
2030 DATA "RET",300,#70,"DIS",300,#71,"LDN",200,#00,"LDA",200,#40
2040 DATA "ORG",800,0,"END",900,0,"DC",1900,0
2050 DATA "LDXA",300,#72,"LDX",300,#F0,"LDI",400,#FB,"STR",200,#50
2060 DATA "STXD",300,#73,"INC",200,#10,"DEC",200,#20,"IRX",300,#60
2070 DATA "GLO",200,#80,"PLO",200,#A0,"GHI",200,#90,"PHI",200,#B0
2080 DATA "ORI",400,#F9,"OR",300,#F1,"XOR",300,#F3,"XRI",400,#FB
2090 DATA "AND",300,#F2,"ANI",400,#FA,"SHRC",300,#76,"SHR",300,#F6
2100 DATA "SHLC",300,#7E,"SHL",300,#FE,"ADD",300,#F4,"ADI",400,#FC
2110 DATA "ADCI",400,#7C,"ADC",300,#74,"SDBI",400,#7D,"SDB",300,#75
2120 DATA "SDI",400,#FD,"SD",300,#F5,"SMBI",400,#7F,"SMB",300,#77
2130 DATA "SMI",400,#FF,"SM",300,#F7,"BR",600,#30,"NBR",300,#38
2140 DATA "BZ",600,#32,"BNZ",600,#3A,"BDF",600,#33,"BNF",600,#3B
2150 DATA "BPZ",600,#33,"BGE",600,#33,"BM",600,#3B,"BL",600,#3B
2160 DATA "BQ",600,#31,"BNQ",600,#39,"B1",600,#34,"BN1",600,#3C
2170 DATA "B2",600,#35,"BN2",600,#3D,"B3",600,#36,"BN3",600,#3E
2180 DATA "B4",600,#37,"BN4",600,#3F,"LBR",700,#C0,"NLBR",300,#CB
2190 DATA "LBZ",700,#C2,"LBNZ",700,#CA,"LBDF",700,#C3,"LENF",700,#CB
2200 DATA "LBQ",700,#C1,"LBNQ",700,#C9,"SKP",300,#3B,"LSKP",300,#CB
2210 DATA "LSZ",300,#CE,"LSNZ",300,#C6,"LSDF",300,#CF,"LSNF",300,#C7
2220 DATA "LSQ",300,#CD,"LSNQ",300,#C5,"LSIE",300,#CC,"OUT",500,#60
2230 DATA "INP",500,#6B
2240 DATA "CALL",700,#D4,"RTS",300,#D5
2990 DATA "*"

```

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

*A 12 issue subscription to QUESTDATA, the publication devoted entirely to the COSMAC 1802 is \$12.
(Add \$6.00 for airmail postage to all foreign countries except Canada and Mexico add \$2.00)
Your comments are always welcome and appreciated. We want to be your 1802's best friend.*

Payment.

- Check or Money Order Enclosed
- Made payable to Quest Electronics
- Master Charge No. _____
- Visa Card No. _____
- Expiration Date: _____

NAME _____

ADDRESS _____

Signature _____

710 381200 CITY _____

STATE _____

ZIP _____

- Renewal
- New Subscription

MEMORY TEST

by
Gary Gehlhoff

Occasionally I have felt that my ELF System was dropping a bit in memory somewhere along the line. To provide an initial check the following short program was written.

The "Memory Test" program puts the hexadecimal value FF into each memory location with each bit then being shifted into the DF register. DF is then tested for zero. If a zero occurs in DF, control is transferred to an error display (See Below). When all the bits of each memory location have been tested via DF, each memory location is then tested for zero. If all locations function properly, the Q L.E.D. will come on and the beginning address of the test will be displayed on the hex display.

If an error is found during testing, the Q L.E.D. will remain off while the high address of the memory location will be displayed on the hex display. Depressing and releasing the Input Button will show the low address of the memory location on the hex display.

Slight modifications to OP Codes in locations 0015, 002D, and 0032 will allow any section of memory to be tested.

It's interesting to note that there are 256 to the 256 power (which is beyond the range of any standard scientific calculator) possible bit combinations in one page of memory. Processing those combinations at the rate of 4,000,000 sec. (Elf clock frequency) would take 9.2 EE 62 years for your Elf to complete the task.

ADDR CODE	COMMENT
0010 90	R(0) 1 --> D
0011 B1 B2 B3	D --> R(1,2,3).1
0014 F8 50 A1	50 --> R(1).0 .0050 Start Test
0017 F8 4F A2	4F --> R(2).0
001A F8 08 A3	08 --> R(3).0
001D E1	X = 1
001E F8 FF	FF --> D
0020 51	D --> M(R(1))
0021 F0	M(R(X)) --> D
0022 F6	Shift Right --> DF
0023 51	D --> (M(R(1)))
0024 3B 42	If DF=0 Go To 42 Error Display
0026 23	R(3)-1
0027 83	R(3).0 --> D
0028 3A 21	If D ≠ 0
002A 11	R(1)+1
002B 91	R(1).1 --> D
002C FF 10	D - 10 --> D High Ending Byte
002E 3A 1A	If D=0 Goto 1A
0030 81	R(1).0 --> D
0031 FF 00	D - 00 --> D Low Ending Byte
0033 3A 1A	If D=0 Goto 1A
0035 21	R(1)-1
0036 F0	M(R(X)) --> D
0037 3A 42	If D ≠ 0 Goto 42 Error Display
0039 91	R(1).1 --> D
003A 3A 35	If D ≠ 0 Goto 35
003C 81	R(1).0 --> D
003D FF 50	D - 50 --> D
003F 3A 35	If D ≠ 0 Goto 35
0041 7B	Q on
0042 91	R(1).1 --> D
0043 E2	X = 2
0044 52	D --> M(R(2))
0045 64	M(R(X)) --> Hex Display R(X)+1
0046 22	R(X)-1
0047 3F 47	Wait for "I" Depressed
0049 37 49	Wait for "I" Released
004B 81	R(1).0 --> D
004C 52	D --> M(R(2))
004D 64	M(R(X)) --> Hex R(X)+1
004E 00	Stop.
004F	Stack Location

COSMAC CLUB COSMAC CLUB COSMAC CLUB COSMAC CLUB COSMAC CLUB COSMAC CLUB COSMAC

28

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