The Innermost Secrets Of The TI 99/4A

by Randy Holcomb

A Computer Shopper Book - From Patch Publishing Inc.

INNERMOST SECRETS OF THE TI 99/4A
TABLE OF CONTENTS

Innermost Secrets Of T.I. 99/4A

by Randy Holcomb

Patch Publishing Co.
Titusville, FL U.S.A.

Printed In The United States of America.

PREFACE

The T.I. 99/4 Home Computer first came out in the summer of 1978. It represented Texas Instrument's entry into the personal computer market. Their marketing strategy dictated that this was to be a "Home Computer", rather than a "Hobbyist", or "Business Computer" and so it was named. The characteristics of the machine reflected this application. It had small "chicklet" keys and the RAM memory available to the user was small. The intent was to program the powerful 16-bit 9900 CPU with cartridges containing programs in Read-Only-Memory. T.I. and their selected software partners would supply the cartridges. Small software houses such as were supplying the Apple II with mountains of software, would be kept out.

The first versions of the TI 99/4 came with a color CRT Monitor and sold for $1,100. They did not prove to be a very good seller. People liked the color and the graphics, but there was little software and the price too high to attract home users. The keyboard and complete lack of expansion units or disks kept away the serious computerists.

Meanwhile Apple and Atari were selling briskly in competition. T.I. withdrew the T.I. 99/4 and came out with the enhanced. T.I. 99/4A which had a regular keyboard and expansion capabilities. Prices were greatly reduced and an advertising campaign was started.

Soon the T.I. 99/4A was selling in stores all over the country. Each time they reduced the price, thousands of new users became T.I. fans. They started to demand Expansion Units, Disk Drives, Modems, Printers and lots and lots of software. T.I. never caught up with the demand and these peripherals were always in short supply. Instead of consolidating their position by supplying the T.I. 99/4A fans with the peripherals they wanted to buy, T.I. kept producing computers and cutting prices in a price war with Commodore. In time these losses came home to roost and the giant Texas Instruments Corp. was in trouble because of it.

Their answer was to dump the T.I. 99/4A and get out of the "Home Computer" business. The sell-off of the inventory at prices as low as $59 was the big Christmas sale of 1983. The T.I. 99/4A became the biggest selling computer in history-while they lasted.

All this time, a loyal group of TI-99s had developed and was bound not to give up the computers that had become their hobby. T.I. had never been liberal with details of how the computers worked and the hobbyists needed all the information they could get, now that they were on their own. Computer Shopper in support of these dedicated T.I.'ers has published a series called, The Innermost Secrets of The T.I. 99/4A, by Randy Holcomb. Randy is one of the Sysops of the T.I. SIG on Compuserve (GO PCS-27). He is also one of the most knowledgeable experts on this system outside Texas Instruments.

We have had so many requests for missing sections of the series and reprints of the entire article that we have now reprinted the series in booklet form. We hope you enjoy it.

Stan Veit
Editor-in Chief
Computer Shopper

INTRODUCTION

Have you ever wondered what really goes on inside your 99/4A? Have you ever wanted to add your own peripherals to do things that no one else have ever dreamed of? Well, here we will be looking at the 99/4A IN DEPTH -- hardware, software, interfacing techniques, you name it.

First off, to really get an appreciation of what we are going to talk about, you should have either the Mini-Memory module or the Editor/Assembler manual for the 99/4A, as these contain valuable information as we tour along inside the various areas.

MEMORY MAP

Let's start with Memory Map:

We will briefly cover each major block separately, beginning with the Console ROM.

The Console ROM contains the start-up code that initializes the 99/4A environment. It includes the GPL (Graphics Programming Language) interpreter, which is where most of the time is spent by BASIC and most of TI's applications ROMs. GPL uses the GROMS (Graphics Read Only Memory) to fetch instructions and execute. The GROM is a proprietary device that in simple terms is a self-incrementing ROM; i.e. once you load a base address into the GROM upon subsequent accesses to the GROM it will fetch the byte at the next address inside the GROM automatically. By having this arrangement it allows for reasonably fast access time of sequentially organized data (such as downloading of character set, which we'll get into later) in a very small package at a very low cost.

In addition to the GPL interpreter, the Console ROM contains the necessary routines to interface the 99/4A environment to device support routines (DSRs) to the outside world. Upon Power-up, the console cycles thru all the peripherals attached on the system and performs initializations code that is contained in each peripheral. Thru this design, no "System"-like operations are necessary.

(How a DSR works will be covered later.)

The floating-point routines for Basic are also contained in the Console ROM, as well as providing the special transcendental functions (SIN, COS, etc.). These routines can be accessed in assembly-language programs.

Finally, code is provided to handle interrupts (either peripheral-generated or by use of the XOP instruction in a program) for use by the various peripherals. VDP RAM and the sound chip make use of system interrupts for various purposes (as will be seen later).

The low half of the Expansion memory card is 8 kilobytes in length; and in most cases contains the "LOADER" used in the Editor/Assembler and Extended BASIC cartridges. When a assembly-language program is to be run, the Editor/Assembler card downloads the LOADER (which is stored in the GROM) into this memory segment and turns control over to it; whereupon it prompts you for the file name(s) of the program you wish to execute. The LOADER opens the files, and while loading the program into memory, checks its symbol table for duplicate label definitions and unresolved references (labels that were declared in the source using REF or DEF statements) and when complete, asks for another file to load. If there are no more files, the LOADER ask you for the program name to execute (which must have been declared in a DEF statement). It then turns control over to your program at that point.

The Extended BASIC loader, although similar in nature to the Editor/Assembler loader, is used exclusively in the Extended BASIC environment and only recognizes this section of memory for assembly-language programs that are linked to the Extended BASIC program.

The section of memory from ↑4000 to ↑5FFF is the area used for the Device Support Routines (DSR's) that are in each peripheral. The DSR contains the machine instructions that interface the peripheral to the TI computing environment. By means of a Peripheral Access Block (PAB) all the input-output calls are standardized and in most cases interchangeable; you can not only LOAD and SAVE programs from cassettes and disks, but you can also do so from the RS232 port, for example. The PAB's for any file that is in use is maintained in the Video Display processor RAM along with any buffer area associated with the file. When the DSR is called on for performing an operation, it is passed the address in VDP RAM of the PAB. Each DSR is responsible for performing any buffering that may be necessary for the device before performing any physical IO operation, and unlike most other micros, the DSR uses the concept of record IO in its program interface versus a "byte at a time" IO scheme.

DSR's are activated by calls to a routine in the console ROM called DSRLNK which scans the system for the proper DSR by cycling thru a series of CRU addresses. Each particular CRU address triggers the peripheral and the DSRLNK routine

Addr	Contents
>0000	- ROM in console. Contains GPL interpreter, DSR interface code, floating point routines, and
>1FFF	- system initialization
>2000	- Low half of Expansion
>3FFF	- Memory card. (8 kbytes)
>4000	- Device Support Routine interface code. (Bank selected based on
>5FFF	- peripheral requested.
>6000	- Application code in front cartridge port. Not normally user accessible. (Used by Extended Basic
>7FFF	- for various routines.)
>8000	- Memory-mapped devices (VDP RAM, GROM, Speech, Sound Synthesizer.)
>9FFF	- And Scratpad RAM.
>A000	- High half of Expansion
>FFFF	- Memory card (24 kbytes)

memory map

interrogates the accessed peripheral on addresses to check; in which case an error code is returned. Check to see if it is the proper peripheral; if it isn't, it continues until it finds the peripheral or runs out of CRU. (There is ONE exception to calling DSRLNK for peripherals -- cassette I/O uses GPLLNK to perform the I/O for cassette; but still uses a PAB like every other peripheral.)

Applications code stored in ROM cartridges is accessed in the areas between †6000 and †7FFF. Extended BASIC, the Mini-Memory cartridge, the Terminal Emulator II cartridge, and games like Parsec all have ROM at this location. In the case of Extended BASIC, judicious logic designes allowed TI to have more than 8 kbytes of code in this address space for such things as sprite and speech support. By having this location available it allows good sized machine code programs to execute without having to purchase Expansion memory; in fact, the Mini-Memory has 4 kbytes of CMOS RAM using a lithium battery for backup to store and execute small programs in either BASIC or machine code.

From †8000 TO †9FFF live all the memory-mapped devices. In addition to these devices, 256 bytes of RAM (normally starting at †8300) is used for GPL and Basic for its stack and scratchpad purposes. The scratchpad is also used to pass parameters to many of the support routines inside the system ROM (such as the floating point accumulator for the floating point routines.) This area is well-defined and can cause you trouble if certain scratchpad contents are violated; so read your manuals carefully.

The rest of the memory (†A000 thru †FFFF) is the high section of the Expansion memory card, and is used by Extended BASIC for program and data storage, as well as by many TI packages such as the Editor/Assembler, the UCSD p-system, TI-Writer and Multiplan. All of this memory is available for your use with the exception of †FFFC thru †FFFF, which contain the jump vectors for XOP 1. As long as you do not make use of XOP 1, this area is free for your use.

Now that you have a basic idea on how things are laid out internally, we can proceed in getting a little more deeper into the inner workings of the 99/4A. The first item we will examine in depth is the heart of the 99/4A -- The TMS9918A Video Display processor. We will examine the programming of the VDP and how it interfaces with the rest of the 99/4A.

THE TMS 9918A VIDEO PROCESSOR

Here is the "heart" of the 99/4A - the TMS9918A Video Display processor (VDP) which is not only used in the 99/4A but in the ColecoVision system (including the ADAM) and in the new MSX Standard machines. Of course, we will primarily concentrate on the 99/4A implementation, but this knowledge can be applied to the other machines as well.

The TMS9918A is a 40-pin DIP package that provides a microprocessor with a video interface with a versatile display interface. The 9918A interfaces with the system in a memory mapped format using 4 memory locations. In the 99/4A the VDP Write Address location is at >8C02; the VDP Read Data location is at >8800; the VDP Write Data location is at >8C00, and the VDP Status location is at >8802. What these locations do is explained below:

VDP Write Address (>8C00). This location is used to determine the address of the memory location of VDP RAM you wish to access. (Note that in the VDP-based system, the RAM is not directly addressed by the host processor; all VDP memory accesses is done thru the VDP). To set the address, you write the least significant address byte of the location you wish to read or write into the VDP Write Address register. This operation takes a few microseconds for the VDP to complete, so a delay should be inserted between writing the least significant byte and the most significant byte; inserting a NOP works fine. Now that the LSB has been written, you are ready to write the most significant byte; but before you do you MUST set the high-order bits of the address to a 01 IF you are planning to write data into the VDP.

Here is an example of setting the VDP address using the pre-defined symbol VDPWA.

```
REF VDPWA - :VDP Write Addr
LI R1,>4100 - ;addr>0100 in ;VDP
    ram + 01
SWPB - ;SETS LSB
MOVB R1,@VDPWA - ;WRITE
    LSB
SWPB - ;KILLS TIME AND
    ;SETS UP MSB
MOVB R1,@VDPWA - ;WRITES
    MSB.
```

Once this address has been set-up, it automatically increments the address on successive reads and writes; which makes programming a lot easier as the VDP automatically increments the address; the programmer doesn't have to worry about incrementing it. Of course, if you have to change to a completely different location, then you must reprogram the address; but this is one of the nice features of this chip. This address is also used to program the control registers; more on those later.

VDP Read Data (>8800). This location reads data pointed to by the VDP Address register. After the data is read; the VDP address register increments. The label for this address is VDPRO. Example:

```
REF VDPRO - ;defines VDPRO
.
.
.
MOVB@VDPRO,R1 - ;places data
    in ;MSB of Rgstr 1
```

VDP Write data (>8C00). Works opposite of VDP Read data. Writes Data pointed to by VDP Address register, which had its highest 2 bits of the address set to 01. Also autoincrements the VDP address register. The Label for this location is VDPWD. Example:

```
REF VDPWD - ;defines VDPWD
.
.
.
MOVB R1,@VDPWA - ;writes MSB
    of ;R1 into VDP.
```

VDP Status Register (>8802). This location gives the VDP status as follows:

```
---------------------------------------------------
      : 0 : 1 : 2 : 3 : 4 : 5 : 6 : 7 :
---------------------------------------------------
      :int:5sf:col: fifth sprite nbr:
---------------------------------------------------
```

int – VDP interrupt flag; set when VDP fires an interrupt to the system. Cleared by resetting VDP and by reading the status register.

5sf – Five Sprites Flag. Set when 5 or more sprites (defined later) on the same screen line. Cleared by resetting VDP and by reading the status register.

col – Sprite collision Flag (or coincidence flag). Set when 2 or more sprites overlap (either on or off-display). Cleared by resetting VDP or reading status register.

fifth sprite nbr – binary value of the fifth sprite on the display line when the collision flag is set. Cleared by resetting VDP or reading status register.

Programming the VDP. The 9918 has 8 write-only registers used to set up the display environment. These registers are defined below:

Register 0 - Mode register 1

: 0 : 1 : 2 : 3 : 4 : 5 : 6 : 7 :
: (reserved – must be 0) : m3:xvi:

m3 – Mode bit 3. When set, places VDP in graphics 2 (bit-map) mode.

xvi – external video input. When set, allows additional video input into the 9918 via external video input pin.

Register 1 - Mode Register 2

: 0 : 1 : 2 : 3 : 4 : 5 : 6 : 7 :
:416:ben:int: ml: m2: 0 :sss:sms:

416 – 4/16k DRAM Selection. When set to 1, the VDP will access a full 16K of VDP RAM. When reset, will access only 4K of VDP RAM.

ben – Blank Enable/Disable. When set, allows for display on the Screen. When reset, the VDP only displays the border color.

int - Interrupt Enable. When set, causes the VDP to interrupt the processor every 60th of a second; and is required to use automatic sprite motion. When reset, no interrupts are generated by the 9918.

ml – Mode 1 Bit. When set, the VDP is in text mode.

m2 – Mode 2 bit. When set, the VDP is in multicolor mode.

sss – Sprite size selection. When set, any sprite used is defined to be 16x16 pixels high. When reset, sprites are 8x8 pixels.

sms – Sprite magnification selection. When set, sprites are magnified. When reset, sprites are normal size.

VDP Register 2 - Screen Image Table sets the address of the screen image table; address is defined as the value of this register times >400.

VDP Register 3 - Color Table Address. Sets the address of the color table (not used in text mode.) Address defined as register contents times >40.

VDP Register 4 - Pattern Descriptor Table Address. Address defined as register contents times >800.

VDP Register 5 - Sprite Attribute List Address (not used in text mode). Defined as register contents times >80.

VDP Register 6 – Sprite Descriptor Table Address (not used in text mode). Defined as register contents times >800.

VDP Register 7 – Screen color. High-order nibble defines the foreground color in text mode; the low-order nibble defines the background color in all modes.

The VDP modes. Now that the registers and their locations have been defined, we can now start to take a look at the modes that the 9918A can run in. The 9918A runs in 4 modes: Graphics 1, Graphics 2, Text, and Multicolor.

In the graphics 1 mode (which is the standard mode that TI BASIC and TI Extended BASIC run in) the screen is divided into 768 blocks, 32 blocks long and 24 blocks high. Each block can contain a value from 0 to 255. This section of memory is called the screen image table; it is responsible to point to another location in memory which contains the actual character pattern that is to be displayed on the screen.

In addition to pointing to the pattern definition, the screen image table points to another table which defines the color that the display will take on, on the screen at that location. An example follows to help clarify this process:

(1) The VDP Screen image table is defined at >0000.

(2) The VDP Pattern Descriptor table starts at >0800.

(3) The VDP Color table starts at >0300.

(4) The contents of the 1st character of the screen image table is >00. Now we use this to index ourselves into each of the other tables. Now for argument's sake, the 1st entry in the color table is >17 and the first eight bytes in the pattern descriptor table is >3C, >7E, >FF, >FF, >FF, >FF, >7E, and >3C. (For those.)

(5) Since the 1st byte of the screen image table is pointing to the 1st character pattern stored in the pattern descriptor table, the VDP gets the pattern definition from 8 consecutive bytes with each byte determining whether the pixel is on or off. In our example, at screen position 1, the bytes stored in the pattern descriptor table assemble to form the image as shown:

x = pixel is on
pixel is off

84218421	
---	
xxxx	>3C
xxxxxx	>7E
xxxxxxxx	>FF
xxxxxxxx	>FF
xxxxxxxx	>FF
xxxxxxxx	>FF
xxxxxx	>7E
xxxx	>3C

So now we have a ball-shaped object being displayed at position 1 of our video display. But we aren't quite thru yet. The 1st byte is also indexing into the color table which provides us the colors for this element. (Each entry in the color table determines the color of eight successive characters in the pattern descriptor table; so that means that characters >0 thru >7 all have the same color, characters >8 thru >F are defined by the next entry in the color table; and so on.) Since it has a >17 in the 1st position, the element has a black background (those pixels that are "on") on a cyan (pixels that are "off") background. The color table is described below.

The end result is a black ball on a cyan background in the first character position on the screen. For TI BASIC users: you will have seen similar things like this in the routines

COLOR HEX	COLOR	TMS9918A		TMS9928A/9929A		
		LUMINANCE (DC) VALUE	CHROMINANCE (AC VALUE)	Y	COLOR DIFFERENCE R-Y	B-Y
0	TRANSPARENT	0.00	-	-	-	-
1	BLACK	0.00	-	0.00	.47	.47
2	MEDIUM GREEN	.53	.53	.53	.07	.20
3	LIGHT GREEN	.67	.40	.67	.17	.27
4	DARK BLUE	.40	.60	.40	.4	1.00
5	LIGHT BLUE	.53	.53	.53	.43	.93
6	DARK RED	.47	.47	.47	.83	.30
7	CYAN	.67	.60	.73	0.00	.70
8	MEDIUM RED	.53	.60	.53	.93	.27
9	LIGHT RED	.67	.60	.67	.93	.27
A	DARK YELLOW	.73	.47	.73	.57	.07
B	LIGHT YELLOW	.80	.33	.80	.57	.17
C	DARK GREEN	.46	.47	.47	.13	.23
D	MAGENTA	.53	.40	.53	.73	.67
E	GRAY	.80	-	.80	.47	.47
F	WHITE	1.00	-	1.00	.47	.47
-	BLACK LEVEL	0.00	-	0.00	.47	.47
-	COLOR BURST	0.00	.40	0.00	47(28A) 73(29A)	.1(28A) .2(29A)
-	SYNC LEVEL	-0.40	-	-.46	.47	.47
-	EXTERNAL VIDEO	-	-	0.00	.47	.47
-	LEVEL	-	-	0.00	-.46	-.46

Figure 0 - Color Table

CALL COLOR and CALL CHAR.

Text mode works in the same way, except that sprites (which we will discuss in another section) are not available and the screen is divided into a 960 block grid with each block being 8 pixels high by 6 pixels wide; allowing for 40 blocks per line and 24 rows per screen. In addition, color information is placed in the color register of the VDP as there is no color table in text mode.

As we mentioned previously, the VDP depends on the programming of various control registers. However, I made a MAJOR omission in not describing how to program the registers! To correct this glaring error, we'll take care of it right now. To program a VDP register, you must select the particular VDP register to program and writing this value into the VDP Register (values 0 thru 7) followed by writing the actual register value. For example, to set register 4 with the value of >20, the following TMS 9900 code will perform the function:

```
REF     VWTR
LI      R0,>0420
BLWP    @VWTR ;write reg.#
SWPB    R0    ;set up value
BLWP    @VWTR ;and set it.
```

Getting back to the summary, Figures 1 and 4 show how Graphics I mode is implemented, showing the layouts of the various tables.

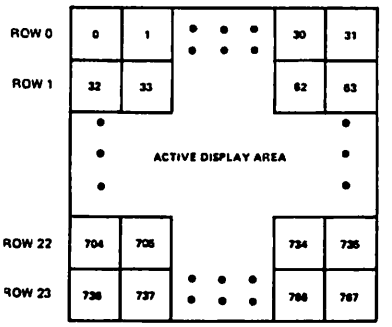


Figure 1 - Pattern Graphics Name Table Mapping.

The next mode we discussed was the Text mode, with figures 5 and 6 showing the structure of this mode.

The next mode we will cover is the Graphics II, or bit-map mode. This mode expands the graphics capabilities significantly by splitting the screen up into 3 equal regions with each region having its own color pattern table and pattern descriptor table. In addition, the pattern color table is expanded to map one-for-one with the pattern descriptor table in each region; so now each pixel row in the pattern has its own "on-color" and "off-color" (see Figure 7). The easiest way to use this mode is to sequentially initialize the values in each screen image table from >00 to >FF and then alter the pixel patterns in the pattern descriptor table. In Figure 8, note how the mapping is achieved for P1, P2, and P3.

The last mode available in the 9918A is the multicolor mode. In the multicolor mode the screen is divided up into a 64x48 matrix, with each cell being 4 pixels high by 4 pixels wide. Each pixel cell can be any of the colors available in the 9918A's palette. The key with multicolor mode is that instead of the pattern descriptor table containing patterns it now contains colors, but instead of using all eight bytes of the entry only two bytes are used with each

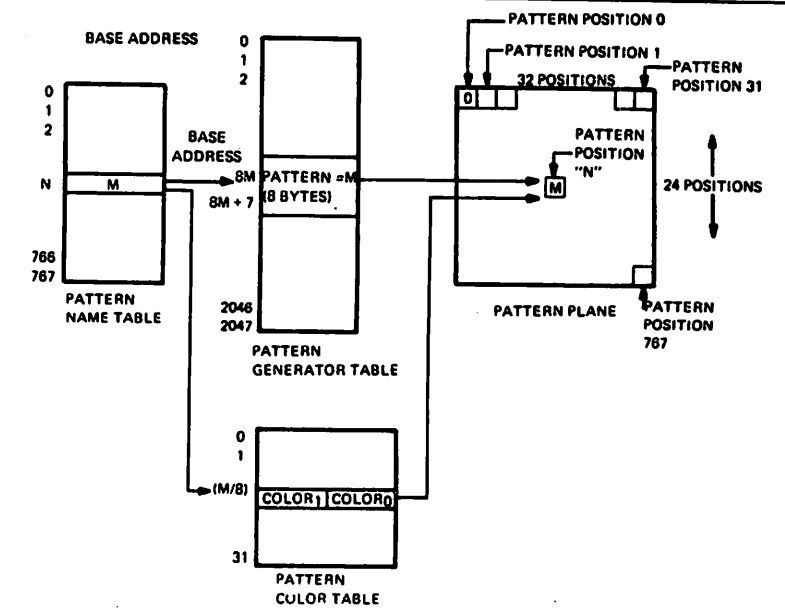


Figure 2 - Graphics I Mode Mapping

ROW/BYTE	COLUMN (PATTERN)						BIT (PATTERN DEFINITION)							
	0	1	2	3	4	5	0	1	2	3	4	5	6	7
0		C	C	C	C	C	0	1	1	1	1	1	0	0
1						C	0	0	0	0	0	1	0	0
2						C	0	0	0	0	0	1	0	0
3			C	C	C	C	0	0	1	1	1	1	0	0
4						C	0	0	0	0	0	1	0	0
5						C	0	0	0	0	0	1	0	0
6		C	C	C	C	C	0	1	1	1	1	1	0	0
7							0	0	0	0	0	0	0	0

Notes: VDP register 7 entry: 71

16.

Color code 7 is cyan (signified above by 'C').
Color code 1 is black (signified above by a space).
Bit 0 is the most significant bit of each data byte

Figure 3 - Pattern Display Mapping.

Byte No	Pattern No.	Byte No.	Pattern No.
0	0..7	16	128..135
1	8..15	17	136..143
2	16..23	18	144..151
3	24..31	19	152..159
4	32..39	20	160..167
5	40..47	21	168..175
6	48..55	22	176..183
7	56..63	23	184..191
8	64..71	24	192..199
9	72..79	25	200..207
10	80..87	26	208..215
11	88..95	27	216..223
12	96..103	28	224..231
13	104..111	29	232..239
14	112..119	30	240..247
15	120..127	31	248..255

Figure 4 - Graphics I Mode Color Table.

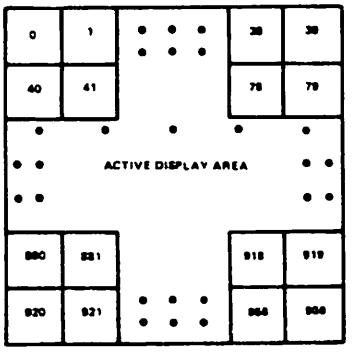


Figure 5 - Text Mode Name Table Pattern Positions.

nibble describing the color desired. The color selection is dependent on the position of the screen position where the name is mapped. It uses this to index into the descriptor table to get the color sequence to use. A good example of the layout of the multicolor mode is given in the Editor/Assembler Manual on page 332.

Sprites. Perhaps the best-known feature of the 9918A is the implementation of sprites. Sprites are basically a pattern which is placed on a plane (the plane is transparent) that allows for easy movement of fixed-size patterns (from 8x8 pixels to 32x32 pixels) across a screen. When a number of planes are layered one on top of another, some exciting displays can be generated. In addition, logic inside the 9918A can detect when two or more sprites "collide", i.e. when a pattern overlaps another pattern on a different plane. In addition, each sprite has "priority" in which when sprite 1 collides with sprite 2 the portion of sprite 1 that collides with sprite 2 covers that portion of the intersection. which can create some useful and striking three-dimensional effects. Figure 9 show how sprites can be used to implement simple animation.

To set up sprites, you must allocate a separate Sprite Pattern table (which is just like the standard pattern descriptor table) and a sprite attribute entry. The Sprite Attribute table entry format is shown in Figure 10. There is a maximum of 32 sprites that can be displayed on the screen, in order from sprite 0 (which has the highest display priority) to sprite 31 (the lowest priority). When defining the pattern of a sprite you must take into account what you programmed into VDP Register 1 for Sprite size and magnification. The table below shows the effects of the programming:

Size	Mag	Area	Resolution	bytes /pin
0	0	8x8	1 pixel	8
1	0	16x16	1 pixel	32
0	1	16x16	4 pixels +	8
1	1	32x32	4 pixels +	32

(+ 2x2 pixel block = 4 pixels)

The first 2 bytes determine the

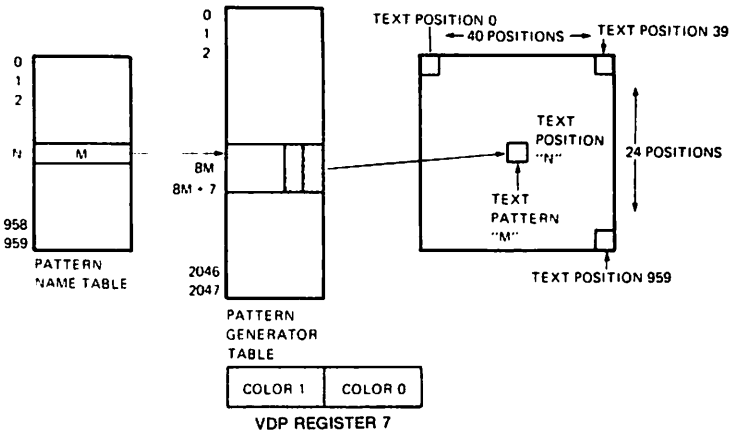


Figure 6 - Mapping of VRAM Into The Pattern Plane In Text Mode.

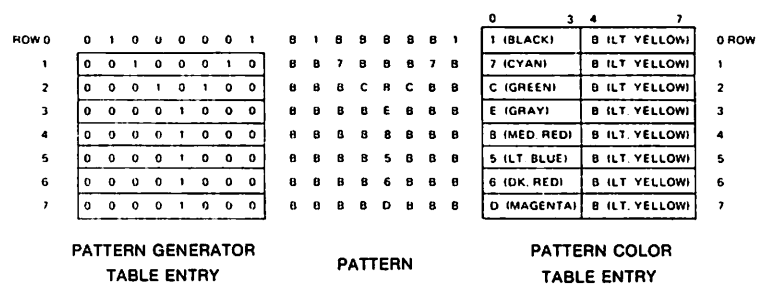


Figure 7 - Pattern Display Mapping

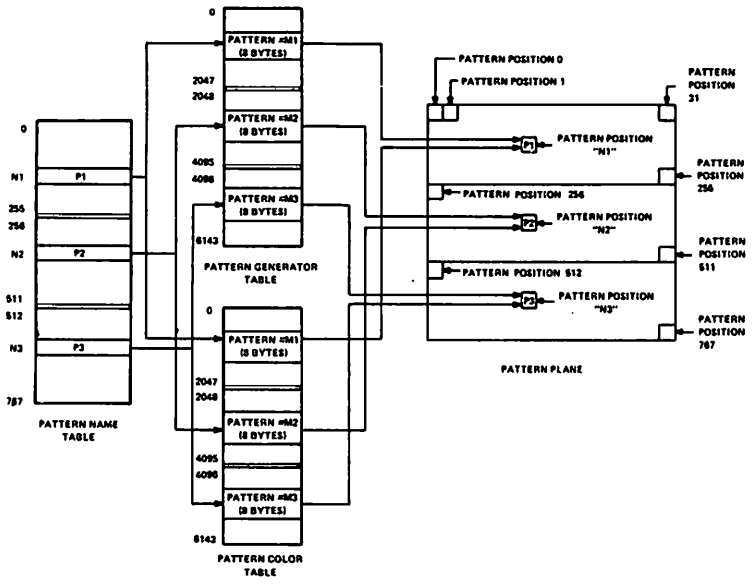


Figure 8 - Graphics II Mode Mapping

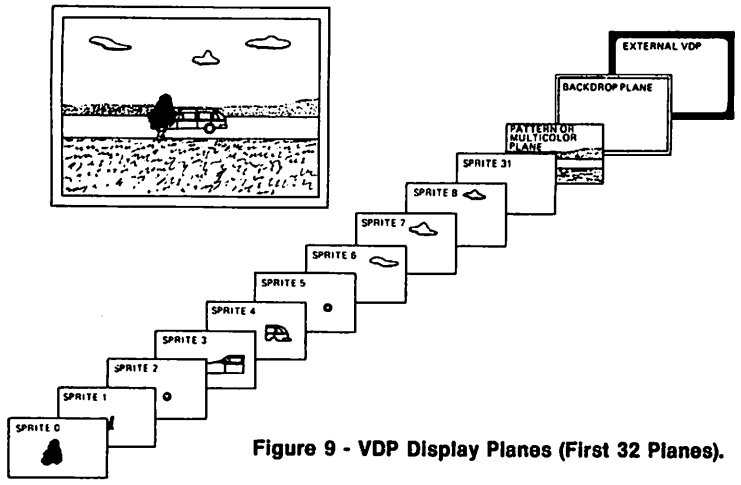


Figure 9 - VDP Display Planes (First 32 Planes).

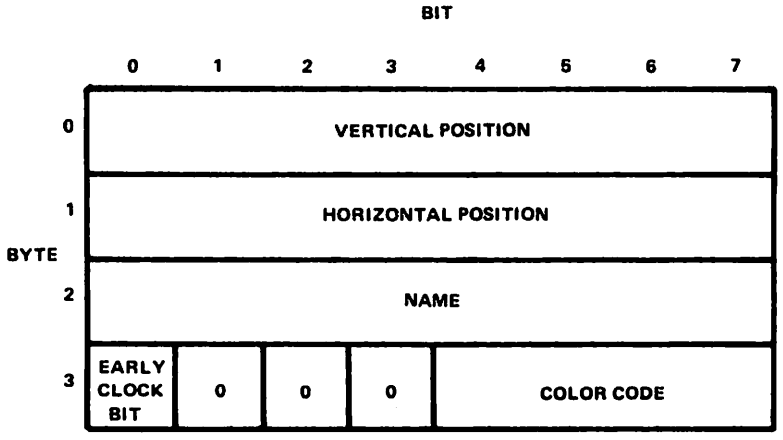


Figure 10 - Sprite Attribute Table Entry.

sprites position on the screen with the upper left hand corner being defined as 0,0. The upper-left hand corner of the sprite being defines its 0,0 position for calculation purposes in determining whether two or more sprites collide. When the value of the vertical position is between >E1 and >00 in hex the sprite comes in from the top of the screen. To get a sprite to come into the screen from the right a value of >FF is placed in the horizontal position location in the sprite attribute table. To get a sprite to come in from the left portion of the screen, the horizontal register must be set to >00 AND the early clock bit must be set to a 1. The early clock bit causes the sprite to move to the left 32 pixels so that a pixel can come in from the left of the screen properly; remember that all pixel measurements start from the left. If you programmed a >00 in the horizontal position the sprite would appear because it would be hinged with column 0 on the display. If a value of >D0 is placed in the vertical position of any Sprite Attribute, Sprite processing stops with that entry; any sprites defined subsequent to this one will be ignored. The Sprite Color code and the Sprite name are in common with the other modes, with the exception that the sprite name points to the sprite pattern table, and that a sprite pattern can be up to 32 bytes long when the size bit is set.

When 2 or more sprites collide, the coincidence flag in the VDP Status register is set. If 5 or more sprites are on the same horizontal line whether they are coinciding or not, the fifth sprite flag is set and the number of the sprite that violated this is placed in the status register. The result on the display is the location of the fifth and subsequent sprites that are on the horizontal line are not displayed.

Automatic motion of sprites. The 9918A does NOT support automatic motion of sprites, but Extended BASIC, assembly and the UCSD Pascal SPRITE unit implement automatic movement of sprites thru the use of a console interrupt routine and a dedicated location of memory called the Sprite Motion table at location >0780 in VDP RAM. This

is described quite nicely on page 340 and 341 of the Editor /Assembler reference manual.

Architecture of TMS9900

In this section we will discuss assembly language programming of the TMS9900. We will give a cursory overview of the architecture of the 9900 and give you helpful hints on how to maximize programming efficiency, along with a few tricks here and there. After careful thought, this will not be so much a tutorial but serve as a springboard to point you in the right direction for you to learn assembly language.

Recently, Steve Davis Publishing released a new book for the beginning Language Programmer called Assembly Language Programming For the TI Home Computer, written by Ralph Moelsworth, which is available at better stores or direct from Steve Davis Publishing.

For those of you who assembly language programming is new, some helpful words of advice:

1. Start out small. Take a small BASIC program you wrote (or portion of a program) and try coding it in assembler to get the feel of things. Remember, you are literally telling the computer what function it is to perform. Unlike BASIC, assembly language creates object code that the computer can directly act upon, rather than being intrepreted the way BASIC is. If you need to modify the assembly language program you have to change the "source" code (code which can be read by a person) than you must use a program called an assembler which translates the source into "object" code (which the computer can understand.)
2. Take a college course in assembly language programming. Taking a course gives you the necessary instruction in fundamentals of computer architecture, plus the fact that most courses take you thru assembly language programming at a reasonable pace. Although the assembler course you take probably won't be dealing with the architecture of the 9900, getting the concepts will allow you to migrate to the 9900 with little problem.
3. Study a listing. With the Editor/Assembler package comes source code for the debugger and in most cases, the game **Tombstone City**. Assemble both of these programs and get listings. Then take the E/A manual and look at how these programs flow. Both programs are commented quite nicely, and with a little bit of time on your part you can see what it is the program is doing.
4. Work at it. Assembly language is not the kind of thing that comes easy for most people. It takes an awful amount of work to do some things that were a breeze to do in any other language. But for the aggravation involved the results can be very rewarding. Good assembly code not only is smaller than code generated by compilers but also tends to be much faster as well. On the larger TI minicomputers such as the 990/10, TI Pascal has a program that takes the output of the Pascal compiler and performs what is called a reverse assembly: it shows you the object code the compiler generated to the particular source line. With this information the programmer can further refine the speed of the program by eliminating or rewriting portions of code that may be redundant or unnecessary.

Although machine language is fast, the 99/4A takes some finagling and some knowledge of how this particular 9900-based system is configured to get the speed up there. First, the memory map. As you remember from the first section, the base 99/4A console has 8K of System ROM and a 256-byte scratchpad memory internally. This memory is on the 16-bit wide data bus. Why is this significant? Because the rest of the 99/4A's memory is either memory-mapped (VDP, sound, speech, and GROM) or is on the 8-bit bus (where you expand your system off of). Machine language code executes the fastest inside the ROM and in scratchpad RAM (if you put code there) because the bus multiplexing logic used by the peripherals is defeated, allowing such code to run full tilt at the system rate of 3 MHz. However, when external devices are accessed (such as disk, a ROM Cartridge or GROM) the Bus multiplexing logic comes into play. This logic slows the machine down by inserting 3 WAIT states to the 9900. Way back when, memory was slow, and for slow memory to be used with a computer the interface circuitry had to have a way of telling the computer to wait a specific period of time for the memory to present the data for the CPU. In the /4A case, wait states are used to place the data on the TI's 16-bit data bus by latching the most significant byte of the 16-bit word inside the console. The multiplexing circuit generates another 3 wait states to latch the least significant byte. After fetching the least significant byte, the word latched and fed to the processor.

So, you are saying to yourself, "So that's why BASIC is so slow!!" Not quite. Remember the GROM? Well, as it turns out most of BASIC is written in TI's proprietary GPL (Graphics Programming Language). As it turns out reading from a GROM is even slower than from a peripheral IF you are not accessing GROM sequentially (which was the way it was designed -- the memory speed of a GROM from address presentation to availability of data is 2 microseconds). Now comes the kicker-GPL is an interpretive language, just like BASIC. So instead of dealing with one interpreter (BASIC, normally written in machine code) you have BASIC, an interpreter being interpreted by GPL, itself another interpreter!

Well, things aren't quite that bad, some of the more time -- critical routines are stashed in ROM (such as floating point routines) and there are a slew of support routines that are at your disposal when you write in assembly language. Most of these routines are described in detail in the Editor/Assembler reference manual.

Once you have become proficient writing assembly-language code, you may want to speed things up a bit. Here are some suggestions on that line:

1. Keep the registers full of often-used data. This is where the TI excels. Because register operations create more compact code (moving a register to a register is just one word; moving a symbolic item to another symbolic item takes 3 words) keeping the registers full of the data

you are using the most (like counters, pointers and such) will cut code size down and also increase execution speed as the additional fetches needed to acquire addresses are not needed. Remember to be careful with register 0 (can't be used for indexing), register 11 (the return address from the BL is kept here), register 12 (the base address for CRU instructions) and registers 13, 14 and 15 (the previous workspace, program counter and status registers respectively; stored by BLWP, interrupt ad XOP instructions).

2. Keep workspace registers in the scratchpad RAM in the console. Be careful however of not overwriting areas that are used by some of the system routines; especially at >83CO, which is the GPL workspace registers and other areas which may be used by TI BASIC. Pages 404 thru 406 of the Editor /Assembler manual describe the addresses and locations of the scratchpad memory locations in detail.

3. Make use of subroutines. If you have code that is constantly used in a good number of sections of your code, make it a subroutine callable by a BL or BLWP instruction. If you are really creative you might even want to implement a "new" instruction of your own using the XOP command. Back when the 990/10 minicomputer was floating about the XOP was (and still is) used to issue what is termed a SUPERVISOR CALL -- requesting that a system service be performed. In the /4A environment most (but not all) consoles allow you to do the same thing by inserting the workspace pointer address at >FFD8 and the program counter at >FFF8 for XOP 1, and >83A0 and >8300 for XOP 2, respectively. (XOP 1 was used by Texas Instruments for software development, and is used by the debugger software for inserting soft breakpoints.)

In some instances, the XOP was used to turn control over to a specialized piece of hardware that took control of the bus, performed its function and returned control back to the computer.

This gives you a rough idea of the world of assembly language of the TI. I omitted giving out code examples for the reason that there are sufficient number of texts out that give better examples than I probably could have given. What I hope to have done is to give you the impetus to start really digging in and make you WANT to learn assembly language.

UCSD P-SYSTEM FOR 99/4A

In the next section we will go and cover the UCSD p-system for the TI and explain what the p-system is all about and the TI 99/4A implementation.

For those of you who are interested in the P-System, you will want to acquire the book **Beginner's Guide For The UCSD Pascal System**, by Kenneth L. Bowles (Byte Books). This book gives you a very good introduction to the concepts and facilities available in the P-System.

The P-System is based on a hypothetical processor (called a p-machine, or pseudo-machine) that executes a well-defined set of instructions defined by the creators of the p-machine. In reality, the UCSD P-system uses what is called a p-interpreter which implements the p-machine environment on a variety of processor families: the PDP-11, the 808x family, the 68xx/68xxx family, the Z80, the 65xx family, the TMS9900/99000 family, and other machines such as the General Automation GA16, the AM-100, and the NCR ALP-2. In addition, there are processors which implement the p-machine and execute the code directly; the Western Digital Microengine is an example.

The p-system was designed for ease of portability of applications; taking code from one machine to another without having to make major modifications. As a matter of fact; the portability of the p-system extends not only to source code but to object code as well. Pascal programs written on the TI 99/4A can be moved over to other p-system machines and will execute on those machines; provided that no machine-dependent features of the original machine were used. Although Pascal is the standard language, other languages are available for use in the p-system (most noticeably FORTRAN 77 and BASIC). The p-system provides for an easy-to-use environment suited for development of software in a single-user environment; one of the most striking examples of this is the tie-in of the compiler to the editor. Let's say you are compiling a program and the program has an error in it. The compiler places on the screen the line in error, the line number and either an error code or error text (depending on the presence of the file SYSTEM.SYNTAX) and gives you the option of allowing you to continue with the compile; aborting the compile or calling in the Editor. If you select the Editor, the compiler quits, calls in the Editor from the system disk, reads in the source file and positions you to the line that was in error for editing. This is the kind of feature that very few of the big mainframe computers can offer; and here it exists on a micro!

Of course, there is a penalty for all that the p-system offers: Since the p-system is based on the p-machine, the p-machine object has to be interpreted much like a BASIC program, and typically execution speeds of p-systems programs can be as slow by as much as a factor of 7 over native machine code of the host processor. If you need assembly-language speed in the p-system you can use the p-system assembler and link routines to your programs. Also there is a package called the native object code for the processor that is running the p-system.

In the TI implementation of Version IV.O of the UCSD P-system; the heart of the p-system is implemented as a peripheral card with a Device Support Routine (DSR) at CRU address >1F00. A switch on the back of the card enables the p-system environment; when the switch is thrown and the console powered up, the DSR takes over and becomes the p-interpreter. GROMs in the p-code peripheral contains the main portion of the operating system (SYSTEM. PASCAL, SYSTEM.STARTUP, and SYSTEM.CHARAC, the character set used by the p-system which can be altered; more on that later). To use the p-system you also need expansion memory and at least 1

(preferable 2) disk drives. The RS232 card, although not required, is highly recommended.

To fully utilize the capabilities of the p-system; you will need the 4 system diskettes: The Editor/Filer diskette, the Pascal Compiler diskette, the Assembler/Linker diskette, and the Utilities diskette. If you are just running applications programs and do not intend to program; you will need the Editor/Filer and the Utilities diskettes to at least maintain the files that you may create and to change certain system characteristics.

Each I/O device in the p-system has a device number associated with itself. There are subroutines that exist which allow you to act on the I/O devices directly. The Devices in the 99/4A version of the p-system are described below:

Nbr	Name	Description
===	====	===========
#1	CONSOLE:	Keyboard/Display
#2	SYSTERM:	Keyboard/Display (no echo)
#4	(disk name):	1st disk drive
#5	(disk name):	2nd disk drive
#6	PRINTER:	RS232/2.BA=9600.PA=0
#7	REMIN:	RS232.BA=300.PA=E.EC
#8	REMOUT:	(same as REMIN:)
#9	(disk name):	3rd disk drive
#14	OS:	P-code Peripheral
#31	TAPE:	Cassette tape
#32	TP:	Thermal Printer

PRINTER:, REMIN: and REMOUT: can be changed to use another device by use of the MODRS232 program on the Utilities diskette. The first disk drive is known as the root volume; if a file is on this disk that is also on the p-code card (OS:) it will use the file on the root volume instead of the p-code version. With this you can change the character set that is loaded by placing your own SYSTEM.CHARAC file on the first drive.

In terms of differences between the TI implementation versus other implementations: the TI version is not as fast as other versions; mainly this is because of the slower TI processor speed and that p-codes are stored in both VDP RAM and in GROM. (Because p-codes are stored in VDP RAM, the high-resolution graphics mode cannot be used without crashing the system. Also, certain large programs (such as Volition Systems Advanced Systems Editor) may not be able to run on the TI due to the p-system's memory map.

A number of UNITs (a group of pre-compiled subroutines and functions stored in a single file or a library) have been written for the TI that allow the Pascal programmer to take advantage of many of the unique features of the TI hardware: sprites, speech, sound, and other miscellaneous support routines for string handling, joystick and keyboard handling.

A p-system user's group, called USUS, has a number of programs available and recently transferred their library to include the TI 99/4A disks. Their membership dues are $25.00 a year. To get a membership form, their address is the UCSD P-system User's Society, P.O. Box 1148, La Jolla, CA 92308.

If you want to learn Pascal and have a TI, or if you are into developing software and want to hit as many machines as you can, no one system covers this as well as the p-system. Although it may be difficult to find the software these days, it is a well designed and reasonably priced system for the quality of the software you get.

In this section we will explore the workings of a Device Support Routine and how to create your own DSR for a peripheral. In order to do this properly it will be a 2-part section; it will also be the last in the Innermost Secrets of the 99/4A Series. But to make it worth everyone's while, we will be using the TI RS232 DSR as an example as how a DSR is constructed; to complement this a COMPLETE disassembly listing can be found in the appendix.

DSR FUNDAMENTALS

Whenever a file is to be processed in the 99/4A environment a special block of memory is created in VDP memory known as the Peripheral Access Block (or PAB). The PAB is the key to linking the I/O environment of the 99/4A with the peripherals. All file devices (disk, RS232, PIO, etc.) use the PAB to pass information back and forth to the program and provide information to the DSR so the DSR can carry out the desired function. When you use the BASIC OPEN (#filenumber) you are creating a PAB for the file you have opened which describes what the organization of the file is, the access mode of the file, and the file size.

FILE ORGANIZATION

To understand file processing concepts we have to define the common denominator of file processing, and that entity is known as a RECORD. A record contains one or more pieces of data that are organized into one unit that is accessed all at the same time. For example, a payroll record would contain an employee's name, his social security number, gross wages, deductions, and net wages. In BASIC we would READ or WRITE that record with a statement that looks like:

```
100 READ #1:EMPNAMES$,
    SOC-SEC-NBR,GROSS-
    WAGES,WAGES,
    DEDUCTIONS, NET-
    WAGES
```

The READ #1 indicates that file #1 (which hopefully was preceded by an OPEN statement, otherwise we'll get a nasty error message) is to store in the fields EMPNAMES$, SOC-SEC-NBR, GROSS--WAGES,DEDUCTIONS and NET-WAGES the record that is stored on the file. In the TI 99/4A environment there are 2 major file organization structures and 2 storage attributes. The details of the structures and organizations can be found in your User's Reference Manual. The above BASIC example shows how a group of fields is collected together to form a LOGICAL Record, which is the normal method that an application program (and the programmer) sees the file. Taking this one step further we introduce the concept of a PHYSICAL record which consists of one or more LOGICAL records which are handled by special routines inside the

DSR. These sit in between the program and the device. The memory area that is used for holding logical records is known as a BUFFER. The buffers correlate to physical records in the case where more than one logical record resides in a physical record. The buffer is most commonly used for disk applications, since the other I/O peripherals operate with the record area as the buffer. The physical device is what is called an unblocked device, i.e., having no buffers.

Getting back to our disk example, the disk physical record size is 256 bytes. If your logical record size in your BASIC program is 64 bytes, the disk buffer will hold 4 records when the particular record is read. When you rewrite the record, the updated record image is placed in the BUFFER and not rewritten back out to disk until a new physical record is accessed. At that point the updated buffer is flushed out and rewritten to disk. Also note that if you read an entire file, you will only be performing (in the case of relative and fixed-length files) N*(s/256) I/O operations, where N is the number of records in the file and s is the size of the record in bytes. Since you can have variable length records in a sequential file, the above formula doesn't hold. When more than one record sits inside a disk buffer, getting the next record is just a matter of extracting the record from the buffer and passing it to the application program. The disk DSR does ALL this for you invisibly, and shows one of the characteristics of how a file management system operates.

PAB CONSTRUCTION

As stated previously, all PABs are located in VDP RAM, along with the record area and buffers. Once created and the file is opened, they are to be maintained and the memory area used is not to be released until the file is CLOSED. Note that this interface is uniform for every peripheral defined in the 99/4A environment (sans keyboard, VDP and joystick port). Of course, the DSR will be using these fields to determine the I/O operation that has to take place, and is responsible for getting the necessary flag bits to indicate whether or not an error condition occurred during the I/O. Following is the PAB Format, with descriptions of the field and allowable contents:

Start Of The PAB:

Byte 0: I/O Opcode

This defines the operation to take place on the file. The values are:

0-Open File

This command must be issued prior to performing any I/O operation (except LOAD and SAVE, which is described later).

1-Close File

This command closes the file presently opened and allows the PAB area to be used for other purposes. If a file was open in either OUTPUT or APPEND mode, an end-of-file marker (EOF) is written to the device before the file is released.

2-Read Record

This command reads a record from the specified device and stores the record in the buffer (pointed to in bytes 23).

3-Write File

This command writes the record pointed-to by the VDP buffer address to the device.

4-Restore/Rewind

This command, usable for disk files only, repositions the file to the beginning (if a sequential file) or to a specified record (relative file) with the next I/O operation uses that frame of reference. Restore should only be used in INPUT or UPDATE Open modes.

5-Load, and 6-Save

These commands transfer direct memory images from VDP RAM to and from a device. These commands do not require an OPEN or CLOSE; they act independent of other DSR commands. The PAB for these operations shares the same structure with these changes: bytes 2 and 3 contain the starting address of the memory image area to be saved or loaded, and bytes 6 and 7 indicate the number of bytes to transfer. (These are the commands used in OLD and SAVE commands in BASIC and with the RUN PROGRAM FILE option of the Editor/Assembler package.)

Other I/O Opcodes

The remainder of the opcodes provide some useful functions:

7-Delete File

This opcode deletes the file from the device. Normally this command is used in the disk DSR as disk is the only device where a delete makes sense.

8-Scratch Record

This command is "supposed" to remove a record from a relative record file, where bytes 6 and 7 point to the record number to remove. I say "supposed" to because the command was never implemented!

9-STATUS

This command can be used at any time. Normally used when a file is open, it returns some useful information in byte 8 of the PAB described below:

Bit 0: EOF. If set, the file is at a logical end of file.

Bit 1: Physical EOF. If set, the file is FULL as there is no more room to write any more records (disk only). The following bits have meaning ONLY if the file has NOT been OPENED:

Bit 2: Record Type. If set, the file being interrogated has the variable attribute; if reset, the file is of FIXED organization.

Bit 3: File Type: If set, the file is a PROGRAM in a file; if reset, it is a standard data file.

Bit 4: Data Type. If set, the data stored in the file is in INTERNAL format. If reset, the file is either a PROGRAM file or a DISPLAY FILE (ASCII).

Bit 5: Not used; not implemented on current peripherals.

Bit 6: PROTECT Flag. If set, the file protect flag is invoked to prevent modification to the data file; if reset, no file protection is enabled.

Bit 7: File Presence. Only valid for disk; when set, means the disk file requested does exist on the specified drive; when reset, indicates the file is not present on the disk. Not valid for unit-record peripherals as ANY device can exist.

GETTING BACK TO THE PAB

Byte 1: Flags/Status. This byte is set during an open to identify the file types, open modes, record types, and

returns an error code for completion of the I/O Operation. Once again, here's the bit assignments:

Bit 0: File Type: If set, indicates a relative record file; if reset, indicates a sequential file.

Bits 1 and 2: Open mode. Mode described below:

00 = UPDATE
01 = OUTPUT
10 = INPUT
11 = APPEND

Bit 3: Data Type. If set, indicates the file is a DISPLAY type; if reset, indicates an INTERNAL type.

Bit 4: Record Type. If set, indicates a fixed length file; if reset indicates a variable length file.

Bits 5 thru 7: Error Flags. These flags are set after an I/O operation:

000 - Device not in system.
001 - Device is write-protected.
010 - Bad open attribute.
011 - Illegal operation.
100 - Out of table or buffer space.
101 - Attempt to read beyond EOF.
110 - Device error.
111 - File error.

Bytes 2 and 3: Data Buffer Address. Defines the location of the buffer to which data is to be read into or written from during an I/O operation.

Byte 4: Logical Record Length. Depending on the open mode, this byte sets the logical record length. If during an OPEN command no length is given, the DSR will automatically assign a default file length. Along with byte 4 comes

Byte 5: Character Count. During a WRITE this byte is set to indicate the number of characters to be written to the device; during a READ Operation this byte is set to the number of characters actually read.

Bytes 6 and 7: Record Number. Used for RELATIVE files, this field contains the relative record number to be accessed with the high-order bit ignored.

Byte 8: Screen Offset. Normally used by the Cassette DSR, this byte is used to offset the screen characters in their value to their normal storage (ASCII) value.

Byte 9: Name Length. This byte contains the Length of the file to be accessed. The address of this byte is placed in the console RAM at location >8356 for DSRLNK to use in its searching routine.

Bytes 10-thru??: Peripheral Name. This series of locations contains the actual device name of the file to be processed. During DSR processing, the DSRLNK routine looks only at the characters up thru the first period character to determine if the DSR exists; if the DSR does not exist, bit 2 of the GPL STATUS byte (>837C) is set indicating DSRLNK couldn't find the device requested.

Why all the above info? This will be necessary to do what we want to do in our final section: to understand HOW a DSR works with a real-world example. You will find in this section part of the RS232 DSR disassembled; the rest will appear in the next section.

DEVICE SUPPORT ROUTINES (DSR)

In the previous section we examined the format of the Peripheral Access Block (PAB) structure that is used for the Device Support routines. In this, the final section, we will examine the workings of a live DSR (The RS232 card) and explain the structure and the requirements necessary to implement a DSR for a peripheral.

Each DSR is stored in ROM at address >4000, and is activated by setting the appropriate CRU address of the device on. In most cases, the DSR is activated by setting the zero bit on at the DSR address block. In the case of the RS232 card, the DSR address is at >1300; so having register 12 loaded with >1300 and doing a SBO 0 instruction activates the DSR ROM. Once activated, NO OTHER DSR can be active until the current active DSR is shut off. To shut off a DSR, the DSR base address must be loaded into register 12, then an SBZ 0 is issued. As a rule, this is of no concern to the DSR programmer as the DSRLNK routine handles DSR linkages.

The format of how a DSR is structured is well-defined in order to allow for minimal overhead of system calls to DSRs and allow the DSR to have enormous flexibility in handling device operations. The major elements that constitute a DSR include: The Symbol Definition Block, which serves to define the memory interface requirements for the DSR with the console; the Header/Linkage blocks which the DSR is required to define to allow DSRLNK to perform its linking function, and the main code section which actually performs the operation. The main section is responsible for determining the I/O operation to be performed, setting any special device switches, buffering the data, and perform any error handling that is needed. Two other functions that the DSR may be required to perform is to pre-initialize the device by means of a power-up routine and for devices that run in an interrupt mode, a special interrupt routine is defined to process interrupt requests from the device.

The Symbol Definition Block consists of defining EQUs that the DSR programmer can use to make programming easier. Normally, the Symbol Definition Block is defined as per the TI Standard Definition:

```
PAD     EQU   ->E0
FAC     EQU   PAD+>4A
ROLO    EQU   PAD+>E1
RILO    EQU   PAD+>E3
OPCODE  EQU   FAC
FLAGS   EQU   FAC+1
BUFADR  EQU   FAC+2
LRECL   EQU   FAC+4
CHARCT  EQU   FAC+5
RECNBR  EQU   FAC+6
SCROFF  EQU   FAC+8
OPTLEN  EQU   FAC+9
DEVLEN  EQU   FAC+10
PABVDP  EQU   FAC+12
```

Where PAD is the start of the CPU RAM in the console, FAC is the start of the scratchpad area available for use by the DSR, and R0LO and R1LO are Registers 0 and 1 of the WP that the DSR is given when the DSR is called. During the scan by DSRLNK it looks at the header and linkage blocks; once DSRLNK finds the proper device in the table it loads the address into register 9 and then DSRLNK issues a BL *R9 and turns control over to the DSR. At this point you can store the current workspace pointer into any register (R4 is preferred) and via displacement addressing any location in CPU RAM may be addressed by the DSR.

The header/linkage block defini-

tion can be best examined by looking at the RS232 disassembly. Starting at location >4000 you notice the value of >AA01. The first byte >AA is required by the DSR support; this indicates a valid DSR is present. The next byte can have any value; it is normally used to indicate a version number. The next word is not used by the DSR; its value is zero. At location >4004 is the value >4010, which points to a two word sequence >0000 and >40F4. This is the vectors for the power-up routine which we will examine later. The next 2 words (with the first word contains >0000 again) is the pointer to the device names/linkages. For now let's look at the first location that it is pointing to, >4016. Looking at the data at >4016 we see that it is pointing to >4020. Following this word is >416E. The >4020 is the next device name/linkage to process should this device name fail the device name matching of DSRLNK. The >416E is the location that the system is to proceed to, should the device name match. Following the entry address of >416E is the length of the device name (1 byte) and the actual device name. So, ROM >4016 thru >401F is the pointer to the next device name, the entry point and the device name. The first word in the device/linkage chain will have a zero at the end of all the devices. In this case you can see the end of the chain at >4060; try following the device/linkage entries and identify the entry addresses, the next entry chains and the lengths and the device names.

At locations >400A and >400C are the vectors that point to the interrupt routine in the DSR; in the case of the RS232 DSR these addresses point to the start of the interrupt routine at >40D2 (remember, the first word of the entry vectors is >0000). Like the power-up sequence, if the entry words are all zeroes, this indicates that the particular feature is not implemented. Here we're lucky as we have both a power-up routine and an interrupt routine to hack at.

First, the power-up sequence. On power-up, the console operating system will check each and every device to see if the device needs to be pre-initialized before being used in the system. Let's examine what happens in the RS232's power-up sequence. First, we go to the power-up vector and fetch the location to start, which points us to >40F4. Control is turned over to >40F4 and we start executing code. First thing that is done is we save the previous R12 value into a temporary register. We need to do this to prevent the DSRLNK routine from trying to shut the DSR off with a bad R12 value; if the original R12 value is NOT restored when we leave the DSR, the system will likely fail. Once saved, we then do an SBO 7, then an SBO 2, followed by an SBZ 1. The SBO 7 turns on the DSR LED on the peripheral, indicating that it is in use. The SBO 2 and the SBZ 1 are used by the PIO circuitry to tell the PIO to reset it. (In the CorComp RS232 DSR the PIO is implemented as a TMS 9901 PSI chip; this code does not hold true for this card.)

The next 2 sets of instructions are used to reset the TMS 9902 Asynchronous Communication Controller chips used by the RS232 card for the RS232 interface. Each chip has extensive logic and is capable of doing a whole slew of things: it has its own built-in baud rate generator and an interval timer, all packaged in a 18-pin package. Unlike most other devices that are memory-mapped, the TMS 9902 uses the CRU logic for its interfacing making it easy to access and program the device. However, the TMS 9902 is an article in itself, but we will mention particulars of the device as we examine the code. If you are really interested in the device, contact your TI Semiconductor representative and ask for the data sheets on the TMS 9902 (NOT the Consumer Products division!) Doing the AI R12,>40 sets up the base addresses for all CRU activity for the first 9902 device. The SBO 31 instruction resets all the internal registers of the 9902. The next instruction AI R12,>40 sets up for the next SBO 31, which resets the second 9902. Finally, R12 is restored, Bit 7 is turned off and we exit the power-up routine via a B *R11. You must exit the DSR with a B *R11.

The interrupt routine is somewhat similar in that when an interrupt is fired, the console must determine who fired the interrupt. To determine this the console ROM will interrogate all peripherals and check for the presence of an interrupt vector; if one is found, then control is turned over to the interrupt routine for processing. In this DSR an interrupt occurs upon receipt of a character, so in this case control is turned over to >40D2. First it stores the WP in R4 (though the disassembler doesn't show it; this is a problem due to an instruction that couldn't be translated at >40CA). The DSR light is turned on, then both R11 and R12 are backed up. The first 9902 is tested for receipt of a character (TB 16) and if true, the processor jumps to >410E where it BL to the routine at >4874. If it fails the receive interrupt test, it checks to see if a data set change, a timer interrupt or a transmitter interrupt has occurred; if one has, then it drops through to the power-up routine; this indicates that the DSR ran into an interrupt condition it shouldn't have and resets the ACC in question. The same procedure also applies to the second 9902 with the exception that if the device fails the tests, it sets the receive interrupt enable on the second 9902 and then branches out of the interrupt routine via *R5 - note that this is OK as the contents of R11 were moved to R5.

Now comes the rough part and that's the main body of the code. To try to explain what all this 2 + K of code does in this article would run on forever. It's not that I'm skirting the issue; it's just that this is a LOT of material to digest and unfortunately, it is difficult to do in this kind of media. However, I will pass along some information that you should find useful in exploring the innards of this code; and if you find something real novel in this code, let people know. Here are some hints to help you along.

(1) DSRLNK places the PAB in the CPU RAM memory for operations before the operation and places the PAB back out in VDP RAM when the operation completes. This appears to be the case as you may notice that many of the operands in the disassembly point to a negative displacement off of R4 which makes

it point to CPU RAM, most notably the PAB definitions and some scratchpad workspace used by the DSR.

(2) There is a switch table starting at >4076 and running thru >4098 which contains the two byte character representation of the switch value and an address that immediately follows the switch literal; also from >40A6 thru >40B2 is a table with the binary values for the baud rate which is used by the DSR to determine the value to program into the shift register to set the baud rate of the 9902. The initialization of the control register is done between locations >482C thru >4840.

(3) Useful TB instructions to look for are TB 27 (data set ready test bit), TB 21 (receive buffer loaded), TB 9 (receive error) and TB 22 (wait for transmitter shift register to empty). Some SBZ instructions to look for are SBZ 18 which resets the receive buffer register and SBZ 16 which shuts the transmitter off. SBO 16 turns on the transmitter on the 9902. Take a look at the code between >47DE and >4806 for some of the logic used by these instructions as well as >4650 and >4668.

I hope this has helped you to start to understand some of the innermost workings that go on inside your 99/4A; and even though the article series was intended to give you the deep dark secrets, we have just barely scratched the surface of the capabilities of this machine. If there are some issues that we left out, it wasn't intentional. It was either that no one asked for them or we just simply don't know enough about them to make inroads. I hope that as time goes on there will be other individuals who will tear into the machine the way we have and share their insights; for this is the ONLY way we will be able to continue to get the most out of our systems. This isn't the end of the line by any means...just keep reading and DOING! ●

ABOUT THE AUTHOR

Randy Holcomb, is a Programmer/Analyst at First Federal Of Michigan by profession and a dedicated TI 99/4A user by avocation. As a member of the South Eastern Michigan Computer Organization he was active in the TI Special Interest Group and was recommended to me as a knowledgeable person about TI matters. I was looking for a person to serve as TI Sysop on CEMSIG, the SIG I ran on CompuServe.

Randy soon became known throughout the country as a person who had special knowledge of the TI 99/4A and who could teach it to others. In a short time most of the messages on the board were addressed to him, reflecting the huge amount of interest in this computer and the dearth of information available about it. When I moved my editorial desk from Computers & Electronics magazine to Computer Shopper, Randy was the first person I thought of to write my TI 99/4A column. He agreed to do it, only after we ran a whole series of articles devoted to educating people about the 99/4A. Shortly after the series began, Texas Instruments discontinued the 99/4A. With this one event, the interest exploded in this machine. Thousands of people had bought the machine at close-out prices and wanted to learn more about it.

Our stock of back issues with the first articles in the series was depleted in no time. Still the letters came asking for them. That is the reason for publishing this book. We are watching the response to it with great interest in order to learn if the public will support a computer long after the manufacturer discontinues it.

Oh yes! Randy can be reached for any questions you have concerning the TI 99/4A at the TI Forum on CompuServe. Just, GO PCS27 at the prompt. If you are a 99er join the SIG, there is no extra charge from CompuServe. Leave a message to SYSOP RANDY and you are sure to get a return answer.

The Computer Shopper electronic addresses are TCS575 on The Source, and 70275,1023 on CompuServe. My personal IDs are CPA013 on The Source, 70210,300 on CompuServe and SVEIT on MCI Mail. We would like to hear from you about this book.

Stan Veit
Editor-in-Chief
Computer Shopper

APPENDIX DISASSEMBLY LISTING OF DSR

ADDR	DATA	TEXT	DECIMAL	CODE	SOURCE	DEST
4000	AA01	??	-22015	A	R1	@0000(R8)
4002	0000	??	0			
4004	4010	@?	16400	SZC	*R0	R0
4006	0000	??	0	?		
4008	4016	@?	16406	SZC	*R6	R0
400A	0000	??	0	?		
400C	406C	@l	16492	SZC	@0000(R12)	R1
400E	0000	??	0			
4010	0000	??	0	?		
4012	40F4	@?	16628	SZC	*R4+	R3
4014	0000	??	0	?		
4016	4020	@	16416	SZC	@416E	R0
4018	416E	An	16750			
401A	0552	?R	1362	INV	*R2	
401C	5332	S2	21298	SZCB	*R2+	R12
401E	3332	32	13106	LDCR	*R2+	12
4020	402C	@,	16428	SZC	@416E(R12)	R0
4022	416E	An	16750			
4024	0752	?R	1874	ABS	*R2	
4026	5332	S2	21298	SZCB	*R2+	R12
4028	3332	32	13106	LDCR	*R2+	12
402A	2F31	/1	12081	?		
402C	4038	@8	16440	SZC	*R8+	R0
402E	4174	At	16756	SZC	*R4+	R5
4030	0752	?R	1874	ABS	*R2	
4032	5332	S2	21298	SZCB	*R2+	R12
4034	3332	32	13106	LDCR	*R2+	12
4036	2F32	/2	12082	?		
4038	4040	@@	16448	SZC	R0	R1
403A	415E	A^	16734	SZC	*R14	R5
403C	0350	?P	848	IDLE		
403E	494F	IO	18767	SZC	R15	@404A(R5)
4040	404A	@J	16458			
4042	415E	A^	16734	SZC	*R14	R5
4044	0550	?P	1360	INV	*R0	
4046	494F	IO	18767	SZC	R15	@2F31(R5)
4048	2F31	/1	12081			
404A	4054	@T	16468	SZC	*R4	R1
404C	4164	Ad	16740	SZC	@0550(R4)	R5
404E	0550	?P	1360			
4050	494F	IO	18767	SZC	R15	@2F32(R5)
4052	2F32	/2	12082			
4054	4060	@`	16480	SZC	@4180	R1
4056	4180	A?	16768			
4058	0752	?R	1874	ABS	*R2	
405A	5332	S2	21298	SZCB	*R2+	R12
405C	3332	32	13106	LDCR	*R2+	12
405E	2F33	/3	12083	?		
4060	0000	??	0	?		
4062	417A	Az	16762	SZC	*R10+	R5
4064	0752	?R	1874	ABS	*R2	
4066	5332	S2	21298	SZCB	*R2+	R12
4068	3332	32	13106	LDCR	*R2+	12
406A	2F34	/4	12084	?		
406C	0000	??	0	?		
406E	40D2	@?	16594	SZC	*R2	R3
4070	0000	??	0	?		
4072	0800	??	2048	SRA	16	R0
4074	0303	??	771	LIMI	4543	
4076	4543	EC	17731			

ADDR	DATA	TEXT	DECIMAL	CODE	SOURCE	DEST
4078	4512	E?	17682	SZC	*R2	*R4
407A	4352	CR	17234	SZC	*R2	R13
407C	4518	E?	17688	SZC	*R8	*R4
407E	4C46	LF	19526	SZC	R6	*R1+
4080	451E	E?	17694	SZC	*R14	*R4
4082	4E55	NU	20053	SZC	*R5	*R9+
4084	4524	E$	17700	SZC	@4441(R4)	*R4
4086	4441	DA	17473			
4088	4570	Ep	17776	SZC	*R0+	*R5
408A	4241	BA	16961	SZC	R1	R9
408C	4536	E6	17718	SZC	*R6+	*R4
408E	5041	PA	20545	SZCB	R1	R1
4090	4540	E@	17728	SZC	R0	*R5
4092	5457	TW	21591	SZCB	*R7	*R1
4094	4596	E?	17814	SZC	*R6	*R6
4096	4348	CH	17224	SZC	R8	R13
4098	452A	E*	17706	SZC	@0000(R10)	*R4
409A	0000	??	0			
409C	0028	?(	40	?		
409E	40B6	@?	16566	SZC	*R6+	R2
40A0	0030	?0	48	?		
40A2	40C4	@?	16580	SZC	R4	R3
40A4	0000	??	0	?		
40A6	006E	?n	110	?		
40A8	012C	?,	300	?		
40AA	0258	?X	600	ANDI	04B0	R8
40AC	04B0	??	1200			
40AE	0960	?`	2400	SRL	6	R0
40B0	12C0	??	4800	JLE	-64	4032
40B2	2580	%?	9600	CZC	R0	R6
40B4	0000	??	0	?		
40B6	8563	?c	-31389	C	@8482(R3)	*R5
40B8	8482	??	-31614			
40BA	8209	??	-32247	C	R9	R8
40BC	015B	?[	347	?		
40BE	8082	??	-32638	C	R2	R2
40C0	8041	?A	-32703	C	R1	R1
40C2	002B	?+	43	?		
40C4	85AA	??	-31318	C	@849C(R10)	*R6
40C6	849C	??	-31588			
40C8	8271	?q	-32143	C	*R1+	R9
40CA	01A1	??	417	?		
40CC	809C	??	-32612	C	*R12	R2
40CE	804E	?N	-32690	C	R14	R1
40D0	8027	?'	-32729	C	@02A4(R7)	R0
40D2	02A4	??	676			
40D4	1D07	??	7431	SBO	7	
40D6	C14B	?K	-16053	MOV	R11	R5
40D8	C18C	??	-15988	MOV	R12	R6
40DA	022C	?,	556	AI	0040	R12
40DC	0040	?@	64			
40DE	1F10	??	7952	TB	16	
40E0	1316	??	4886	JEQ	+22	410E
40E2	1F1F	??	7967	TB	31	
40E4	1306	??	4870	JEQ	+6	40F2
40E6	022C	?,	556	AI	0040	R12
40E8	0040	?@	64			
40EA	1F10	??	7952	TB	16	
40EC	1310	??	4880	JEQ	+16	410E

ADDR	DATA	TEXT	DECIMAL	CODE	SOURCE	DEST
40EE	1F1F	??	7967	TB	31	
40F0	1632	?2	5682	JNE	+50	4156
40F2	C306	??	-15610	MOV	R6	R12
40F4	C18C	??	-15988	MOV	R12	R6
40F6	1D07	??	7431	SBO	7	
40F8	1D02	??	7426	SBO	2	
40FA	1E01	??	7681	SBZ	1	
40FC	022C	?,	556	AI	0040	R12
40FE	0040	?@	64			
4100	1D1F	??	7455	SBO	31	
4102	022C	?,	556	AI	0040	R12
4104	0040	?@	64			
4106	1D1F	??	7455	SBO	31	
4108	C306	??	-15610	MOV	R6	R12
410A	1E07	??	7687	SBZ	7	
410C	045B	?[	1115	B	*R11	
410E	06A0	??	1696	BL	@4874	
4110	4874	Ht	18548			
4112	1621	?!	5665	JNE	+33	4156
4114	D064	?d	-12188	MOVB	@FF24(R4)	R1
4116	FF24	?$	-220			
4118	B060	?`	-20384	AB	@45F9	R1
411A	45F9	E?	17913			
411C	9901	??	-26367	CB	R1	@FF22(R4)
411E	FF22	?"	-222			
4120	1201	??	4609	JLE	+1	4124
4122	04C1	??	1217	CLR	R1	
4124	9901	??	-26367	CB	R1	@FF23(R4)
4126	FF23	?#	-221			
4128	1306	??	4870	JEQ	+6	4136
412A	3607	6?	13831	STCR	R7	8
412C	1F09	??	7945	TB	9	
412E	1607	??	5639	JNE	+7	413E
4130	0207	??	519	LI	FF00	R7
4132	FF00	??	-256			
4134	1004	??	4100	JMP	+4	413E
4136	0207	??	519	LI	FE00	R7
4138	FE00	??	-512			
413A	D064	?d	-12188	MOVB	@FF24(R4)	R1
413C	FF24	?$	-220			
413E	D901	??	-9983	MOVB	R1	@FF24(R4)
4140	FF24	?$	-220			
4142	0981	??	2433	SRL	8	R1
4144	A064	?d	-24476	A	@FF20(R4)	R1
4146	FF20	?	-224			
4148	0241	?A	577	ANDI	3FFF	R1
414A	3FFF	??	16383			
414C	06A0	??	1696	BL	@484E	
414E	484E	HN	18510			
4150	4000	@?	16384	SZC	R0	R0
4152	DBC7	??	-9273	MOVB	R7	@FFFE(R15)
4154	FFFE	??	-2			
4156	1D12	??	7442	SBO	18	
4158	C306	??	-15610	MOV	R6	R12
415A	1E07	??	7687	SBZ	7	
415C	0455	?U	1109	B	*R5	
415E	0206	??	518	LI	0001	R6
4160	0001	??	1			
4162	1002	??	4098	JMP	+2	4168

ADDR	DATA	TEXT	DECIMAL	CODE	SOURCE	DEST
4164	0206	??	518	LI	0002	R6
4166	0002	??	2			
4168	0703	??	1795	SETO	R3	
416A	04C2	??	1218	CLR	R2	
416C	1011	??	4113	JMP	+17	4190
416E	0206	??	518	LI	0001	R6
4170	0001	??	1			
4172	1008	??	4104	JMP	+8	4184
4174	0206	??	518	LI	0001	R6
4176	0001	??	1			
4178	1008	??	4104	JMP	+8	418A
417A	0206	??	518	LI	0002	R6
417C	0002	??	2			
417E	1005	??	4101	JMP	+5	418A
4180	0206	??	518	LI	0002	R6
4182	0002	??	2			
4184	0202	??	514	LI	0040	R2
4186	0040	?@	64			
4188	1002	??	4098	JMP	+2	418E
418A	0202	??	514	LI	0080	R2
418C	0080	??	128			
418E	04C3	??	1219	CLR	R3	
4190	02A4	??	676	STWP	R4	
4192	C90B	??	-14069	MOV	R11	@FF84(R4)
4194	FF84	??	-124			
4196	8181	??	-32383	C	R1	R6
4198	1302	??	4866	JEQ	+2	419E
419A	0460	?`	1120	B	@4480	
419C	4480	D?	17536			
419E	C184	??	-15996	MOV	R4	R6
41A0	0226	?&	550	AI	FF78	R6
41A2	FF78	?x	-136			
41A4	0205	??	517	LI	0006	R5
41A6	0006	??	6			
41A8	04F6	??	1270	CLR	*R6+	
41AA	0605	??	1541	DEC	R5	
41AC	16FD	??	5885	JNE	-3	41A8
41AE	1D07	??	7431	SBO	7	
41B0	A302	??	-23806	A	R2	R12
41B2	06A0	??	1696	BL	@4842	
41B4	4842	HB	18498			
41B6	0000	??	0	?		
41B8	0205	??	517	LI	000A	R5
41BA	000A	??	10			
41BC	C184	??	-15996	MOV	R4	R6
41BE	0226	?&	550	AI	FF6A	R6
41C0	FF6A	?j	-150			
41C2	DDAF	??	-8785	MOVB	@FBFE(R15)	*R6+
41C4	FBFE	??	-1026			
41C6	0605	??	1541	DEC	R5	
41C8	16FC	??	5884	JNE	-4	41C2
41CA	5920	Y	22816	SZCB	@460B	@FF6B(R4)
41CC	460B	F?	17931			
41CE	FF6B	?k	-149			
41D0	9920	?	-26336	CB	@40B3	@FF6A(R4)
41D2	40B3	@?	16563			
41D4	FF6A	?j	-150			
41D6	1606	??	5638	JNE	+6	41E4
41D8	F920	?	-1760	SOCB	@4132	@FF7D(R4)
41DA	4132	A2	16690			
41DC	FF7D	?)	-131			

ADDR	DATA	TEXT	DECIMAL	CODE	SOURCE	DEST
41DE	5920	Y	22816	SZCB	@40B3	@FF6A(R4)
41E0	40B3	@?	16563			
41E2	FF6A	?j	-150			
41E4	9824	?$	-26588	CB	@FF6A(R4)	@41A7
41E6	FF6A	?j	-150			
41E8	41A7	A?	16807			
41EA	1202	??	4610	JLE	+2	41F0
41EC	0460	?`	1120	B	@4450	
41EE	4450	DP	17488			
41F0	06A0	??	1696	BL	@4490	
41F2	4490	D?	17552			
41F4	D164	?d	-11932	MOVB	@FF6A(R4)	R5
41F6	FF6A	?j	-150			
41F8	0985	??	2437	SRL	8	R5
41FA	0A15	??	2581	SLA	1	R5
41FC	C165	?e	-16027	MOV	@4202(R5)	R5
41FE	4202	B?	16898			
4200	0455	?U	1109	B	*R5	
4202	4210	B?	16912	SZC	*R0	R8
4204	4464	Dd	17508	SZC	@4236(R4)	*R1
4206	4236	B6	16950			
4208	42FA	B?	17146	SZC	*R10+	R11
420A	4450	DP	17488	SZC	*R0	*R1
420C	4338	C8	17208	SZC	*R8+	R12
420E	43D2	C?	17362	SZC	*R2	R15
4210	D0A4	??	-12124	MOVB	@FF6E(R4)	R2
4212	FF6E	?n	-146			
4214	1609	??	5641	JNE	+9	4228
4216	06A0	??	1696	BL	@4842	
4218	4842	HB	18498			
421A	4004	@?	16388	SZC	R4	R0
421C	0202	??	514	LI	5000	R2
421E	5000	P?	20480			
4220	D902	??	-9982	MOVB	R2	@FF6E(R4)
4222	FF6E	?n	-146			
4224	DBC2	??	-9278	MOVB	R2	@FFFE(R15)
4226	FFFE	??	-2			
4228	D064	?d	-12188	MOVB	@FF6B(R4)	R1
422A	FF6B	?k	-149			
422C	2060	`	8288	COC	@43CA	R1
422E	43CA	C?	17354			
4230	1663	?c	5731	JNE	+99	42F8
4232	0460	?`	1120	B	@444A	
4234	444A	DJ	17482			
4236	0743	?C	1859	ABS	R3	
4238	5920	Y	22816	SZCB	@4132	@FF6F(R4)
423A	4132	A2	16690			
423C	FF6F	?o	-145			
423E	D1E4	??	-11804	MOVB	@FF6E(R4)	R7
4240	FF6E	?n	-146			
4242	C264	?d	-15772	MOV	@FF6C(R4)	R9
4244	FF6C	?l	-148			
4246	06A0	??	1696	BL	@4740	
4248	4740	G@	18240			
424A	1607	??	5639	JNE	+7	425A
424C	06A0	??	1696	BL	@463A	
424E	463A	F:	17978			
4250	9187	??	-28281	CB	R7	R6
4252	1402	??	5122	JHE	+2	4258
4254	0460	?`	1120	B	@4456	
4256	4456	DV	17494			
4258	C1C6	??	-15930	MOV	R6	R7
425A	0987	??	2439	SRL	8	R7
425C	1348	?H	4936	JEQ	+72	42EE
425E	06A0	??	1696	BL	@463A	
4260	463A	F:	17978			
4262	06A0	??	1696	BL	@4740	
4264	4740	G@	18240			
4266	133A	?:	4922	JEQ	+58	42DC
4268	D064	?d	-12188	MOVB	@FF78(R4)	R1
426A	FF78	?x	-136			
426C	1307	??	4871	JEQ	+7	427C
426E	06A0	??	1696	BL	@474A	
4270	474A	GJ	18250			
4272	1334	?4	4916	JEQ	+52	42DC
4274	0286	??	646	CI	R6	0D00
4276	0D00	??	3328			
4278	1631	?1	5681	JNE	+49	42DC
427A	1039	?9	4153	JMP	+57	42EE
427C	0286	??	646	CI	R6	0D00
427E	0D00	??	3328			
4280	1325	?%	4901	JEQ	+37	42CC
4282	0286	??	646	CI	R6	7F00
4284	7F00	??	32512			
4286	1312	??	4882	JEQ	+18	42AC
4288	0286	??	646	CI	R6	1200
428A	1200	??	4608			
428C	1625	?%	5669	JNE	+37	42D8
428E	C064	?d	-16284	MOV	@FF6C(R4)	R1
4290	FF6C	?l	-148			
4292	06A0	??	1696	BL	@4850	
4294	4850	HP	18512			
4296	06A0	??	1696	BL	@46EE	
4298	46EE	F?	18158			
429A	C089	??	-16247	MOV	R9	R2
429C	60A4	`?	24740	S	@FF6C(R4)	R2
429E	FF6C	?l	-148			
42A0	1003	??	4099	JMP	+3	42A8
42A2	06A0	??	1696	BL	@47DE	
42A4	47DE	G?	18398			
42A6	0602	??	1538	DEC	R2	
42A8	16FC	??	5884	JNE	-4	42A2
42AA	10D9	??	4313	JMP	-39	425E
42AC	8264	?d	-32156	C	@FF6C(R4)	R9
42AE	FF6C	?l	-148			
42B0	13D6	??	5078	JEQ	-42	425E
42B2	0587	??	1415	INC	R7	
42B4	0609	??	1545	DEC	R9	
42B6	C049	?I	-16311	MOV	R9	R1
42B8	06A0	??	1696	BL	@4850	
42BA	4850	HP	18512			
42BC	06A0	??	1696	BL	@47DE	
42BE	47DE	G?	18398			
42C0	0286	??	646	CI	R6	0D00
42C2	0D00	??	3328			
42C4	16CC	??	5836	JNE	-52	425E
42C6	06A0	??	1696	BL	@4700	
42C8	4700	G?	18176			

ADDR	DATA	TEXT	DECIMAL	CODE	SOURCE	DEST
42CA	10C9	??	4297	JMP	-55	425E
42CC	06A0	??	1696	BL	@474A	
42CE	474A	GJ	18250			
42D0	1303	??	4867	JEQ	+3	42D8
42D2	06A0	??	1696	BL	@46EE	
42D4	46EE	F?	18158			
42D6	100B	??	4107	JMP	+11	42EE
42D8	06A0	??	1696	BL	@47E6	
42DA	47E6	G?	18406			
42DC	C049	?I	-16311	MOV	R9	R1
42DE	06A0	??	1696	BL	@484E	
42E0	484E	HN	18510			
42E2	4000	@?	16384	SZC	R0	R0
42E4	DBC6	??	-9274	MOVB	R6	@FFFE(R15)
42E6	FFFE	??	-2			
42E8	0589	??	1417	INC	R9	
42EA	0607	??	1543	DEC	R7	
42EC	16B8	??	5816	JNE	-72	425E
42EE	6264	bd	25188	S	@FF6C(R4)	R9
42F0	FF6C	?l	-148			
42F2	0A89	??	2697	SLA	8	R9
42F4	D909	??	-9975	MOVB	R9	@FF6F(R4)
42F6	FF6F	?o	-145			
42F8	101D	??	4125	JMP	+29	4334
42FA	C0C3	??	-16189	MOV	R3	R3
42FC	1301	??	4865	JEQ	+1	4300
42FE	0703	??	1795	SETO	R3	
4300	C064	?d	-16284	MOV	@FF6C(R4)	R1
4302	FF6C	?l	-148			
4304	06A0	??	1696	BL	@4850	
4306	4850	HP	18512			
4308	D1E4	??	-11804	MOVB	@FF6F(R4)	R7
430A	FF6F	?o	-145			
430C	06A0	??	1696	BL	@4740	
430E	4740	G@	18240			
4310	1603	??	5635	JNE	+3	4318
4312	C187	??	-15993	MOV	R7	R6
4314	06A0	??	1696	BL	@47E6	
4316	47E6	G?	18406			
4318	0987	??	2439	SRL	8	R7
431A	1304	??	4868	JEQ	+4	4324
431C	06A0	??	1696	BL	@47DE	
431E	47DE	G?	18398			
4320	0607	??	1543	DEC	R7	
4322	16FC	??	5884	JNE	-4	431C
4324	06A0	??	1696	BL	@4740	
4326	4740	G@	18240			
4328	1305	??	4869	JEQ	+5	4334
432A	06A0	??	1696	BL	@474A	
432C	474A	GJ	18250			
432E	1302	??	4866	JEQ	+2	4334
4330	06A0	??	1696	BL	@46EE	
4332	46EE	F?	18158			
4334	0460	?`	1120	B	@4464	
4336	4464	Dd	17508			
4338	C024	?$	-16348	MOV	@FF70(R4)	R0
433A	FF70	?p	-144			
433C	06A0	??	1696	BL	@47E4	
433E	47E4	G?	18404			

ADDR	DATA	TEXT	DECIMAL	CODE	SOURCE	DEST
4340	1600	??	5632	JNE	0	4342
4342	0205	??	517	LI	0007	R5
4344	0007	??	7			
4346	0201	??	513	LI	C01C	R1
4348	C01C	??	-16356			
434A	06A0	??	1696	BL	@4870	
434C	4870	Hp	18544			
434E	1307	??	4871	JEQ	+7	435E
4350	0601	??	1537	DEC	R1	
4352	16FB	??	5883	JNE	-5	434A
4354	06A0	??	1696	BL	@4880	
4356	4880	H?	18560			
4358	0605	??	1541	DEC	R5	
435A	16F5	??	5877	JNE	-11	4346
435C	10EF	??	4335	JMP	-17	433C
435E	0709	??	1801	SETO	R9	
4360	06A0	??	1696	BL	@45C6	
4362	45C6	E?	17862			
4364	C1C6	??	-15930	MOV	R6	R7
4366	06A0	??	1696	BL	@45C6	
4368	45C6	E?	17862			
436A	0986	??	2438	SRL	8	R6
436C	E1C6	??	-7738	SOC	R6	R7
436E	06A0	??	1696	BL	@45A0	
4370	45A0	E?	17824			
4372	06A0	??	1696	BL	@46B4	
4374	46B4	F?	18100			
4376	8248	?H	-32184	C	R8	R9
4378	1304	??	4868	JEQ	+4	4382
437A	06A0	??	1696	BL	@47E4	
437C	47E4	G?	18404			
437E	1500	??	5376	JGT	0	4380
4380	10EE	??	4334	JMP	-18	435E
4382	81C0	??	-32320	C	R0	R7
4384	1A68	?h	6760	JL	+104	4456
4386	06A0	??	1696	BL	@47E4	
4388	47E4	G?	18404			
438A	0600	??	1536	DEC	R0	
438C	06A0	??	1696	BL	@4686	
438E	4686	F?	18054			
4390	0709	??	1801	SETO	R9	
4392	C04A	?J	-16310	MOV	R10	R1
4394	06A0	??	1696	BL	@484E	
4396	484E	HN	18510			
4398	4000	@?	16384	SZC	R0	R0
439A	06A0	??	1696	BL	@45C6	
439C	45C6	E?	17862			
439E	DBC6	??	-9274	MOVB	R6	@FFFE(R15)
43A0	FFFE	??	-2			
43A2	0607	??	1543	DEC	R7	
43A4	16FA	??	5882	JNE	-6	439A
43A6	06A0	??	1696	BL	@45A0	
43A8	45A0	E?	17824			
43AA	C0C3	??	-16189	MOV	R3	R3
43AC	1302	??	4866	JEQ	+2	43B2
43AE	06A0	??	1696	BL	@48A2	
43B0	48A2	H?	18594			
43B2	8209	??	-32247	C	R9	R8
43B4	1306	??	4870	JEQ	+6	43C2

ADDR	DATA	TEXT	DECIMAL	CODE	SOURCE	DEST
43B6	C1E4	??	-15900	MOV	@FF80(R4)	R7
43B8	FF80	??	-128			
43BA	06A0	??	1696	BL	@47E4	
43BC	47E4	G?	18404			
43BE	1500	??	5376	JGT	0	43C0
43C0	10E7	??	4327	JMP	-25	4390
43C2	06A0	??	1696	BL	@47E4	
43C4	47E4	G?	18404			
43C6	0600	??	1536	DEC	R0	
43C8	022A	?*	554	AI	0100	R10
43CA	0100	??	256			
43CC	C1E4	??	-15900	MOV	@FF7E(R4)	R7
43CE	FF7E	?~	-130			
43D0	10DD	??	4317	JMP	-35	438C
43D2	C04A	?J	-16310	MOV	R10	R1
43D4	06A0	??	1696	BL	@4850	
43D6	4850	HP	18512			
43D8	06A0	??	1696	BL	@463A	
43DA	463A	F:	17978			
43DC	0286	??	646	CI	R6	1600
43DE	1600	??	5632			
43E0	16FB	??	5883	JNE	-5	43D8
43E2	0709	??	1801	SETO	R9	
43E4	C0C3	??	-16189	MOV	R3	R3
43E6	1302	??	4866	JEQ	+2	43EC
43E8	06A0	??	1696	BL	@48A2	
43EA	48A2	H?	18594			
43EC	C1A4	??	-15964	MOV	@FF70(R4)	R6
43EE	FF70	?p	-144			
43F0	06A0	??	1696	BL	@45D0	
43F2	45D0	E?	17872			
43F4	06C6	??	1734	SWPB	R6	
43F6	06A0	??	1696	BL	@45D0	
43F8	45D0	E?	17872			
43FA	06A0	??	1696	BL	@45B4	
43FC	45B4	E?	17844			
43FE	06A0	??	1696	BL	@463A	
4400	463A	F:	17978			
4402	0286	??	646	CI	R6	0600
4404	0600	??	1536			
4406	16ED	??	5869	JNE	-19	43E2
4408	C1E4	??	-15900	MOV	@FF70(R4)	R7
440A	FF70	?p	-144			
440C	06A0	??	1696	BL	@4686	
440E	4686	F?	18054			
4410	0709	??	1801	SETO	R9	
4412	C04A	?J	-16310	MOV	R10	R1
4414	06A0	??	1696	BL	@4850	
4416	4850	HP	18512			
4418	D1AF	??	-11857	MOVB	@FBFE(R15)	R6
441A	FBFE	??	-1026			
441C	06A0	??	1696	BL	@45D0	
441E	45D0	E?	17872			
4420	0607	??	1543	DEC	R7	
4422	16FA	??	5882	JNE	-6	4418
4424	06A0	??	1696	BL	@45B4	
4426	45B4	E?	17844			
4428	06A0	??	1696	BL	@463A	
442A	463A	F:	17978			

ADDR	DATA	TEXT	DECIMAL	CODE	SOURCE	DEST
442C	0286	??	646	CI	R6	0600
442E	0600	??	1536			
4430	1307	??	4871	JEQ	+7	4440
4432	C0C3	??	-16189	MOV	R3	R3
4434	1302	??	4866	JEQ	+2	443A
4436	06A0	??	1696	BL	@48A2	
4438	48A2	H?	18594			
443A	C1E4	??	-15900	MOV	@FF80(R4)	R7
443C	FF80	??	-128			
443E	10E8	??	4328	JMP	-24	4410
4440	022A	?*	554	AI	0100	R10
4442	0100	??	256			
4444	C1E4	??	-15900	MOV	@FF7E(R4)	R7
4446	FF7E	?~	-130			
4448	10E1	??	4321	JMP	-31	440C
444A	0201	??	513	LI	4000	R1
444C	4000	@?	16384			
444E	1008	??	4104	JMP	+8	4460
4450	0201	??	513	LI	6000	R1
4452	6000	`?	24576			
4454	1005	??	4101	JMP	+5	4460
4456	0201	??	513	LI	8000	R1
4458	8000	??	-32768			
445A	1002	??	4098	JMP	+2	4460
445C	0201	??	513	LI	C000	R1
445E	C000	??	-16384			
4460	F901	??	-1791	SOCB	R1	@FF6B(R4)
4462	FF6B	?k	-149			
4464	06A0	??	1696	BL	@4842	
4466	4842	HB	18498			
4468	4001	@?	16385	SZC	R1	R0
446A	DBE4	??	-9244	MOVB	@FF6B(R4)	@FFFE(R15)
446C	FF6B	?k	-149			
446E	FFFE	??	-2			
4470	06A0	??	1696	BL	@4842	
4472	4842	HB	18498			
4474	4005	@?	16389	SZC	R5	R0
4476	DBE4	??	-9244	MOVB	@FF6F(R4)	@FFFE(R15)
4478	FF6F	?o	-145			
447A	FFFE	??	-2			
447C	05E4	??	1508	INCT	@FF84(R4)	
447E	FF84	??	-124			
4480	024C	?L	588	ANDI	FF00	R12
4482	FF00	??	-256			
4484	C2E4	??	-15644	MOV	@FF84(R4)	R11
4486	FF84	??	-124			
4488	1D02	??	7426	SBO	2	
448A	1E01	??	7681	SBZ	1	
448C	1E07	??	7687	SBZ	7	
448E	045B	?[	1115	B	*R11	
4490	C90B	??	-14069	MOV	R11	@FF86(R4)
4492	FF86	??	-122			
4494	06A0	??	1696	BL	@4730	
4496	4730	G0	18224			
4498	1305	??	4869	JEQ	+5	44A4
449A	0208	??	520	LI	4076	R8
449C	4076	@v	16502			
449E	0201	??	513	LI	B200	R1
44A0	B200	??	-19968			

ADDR	DATA	TEXT	DECIMAL	CODE	SOURCE	DEST
44A2	1004	??	4100	JMP	+4	44AC
44A4	0208	??	520	LI	408A	R8
44A6	408A	@?	16522			
44A8	0201	??	513	LI	8300	R1
44AA	8300	??	-32000			
44AC	0205	??	517	LI	012C	R5
44AE	012C	?,	300			
44B0	C244	?D	-15804	MOV	R4	R9
44B2	0229	?)	553	AI	FFFA	R9
44B4	FFFA	??	-6			
44B6	D641	?A	-10687	MOVB	R1	*R9
44B8	06A0	??	1696	BL	@45F4	
44BA	45F4	E?	17908			
44BC	D024	?$	-12252	MOVB	@FF73(R4)	R0
44BE	FF73	?s	-141			
44C0	0980	??	2432	SRL	8	R0
44C2	6024	`$	24612	S	@FF74(R4)	R0
44C4	FF74	?t	-140			
44C6	1217	??	4631	JLE	+23	44F6
44C8	C064	?d	-16284	MOV	@FF76(R4)	R1
44CA	FF76	?v	-138			
44CC	06A0	??	1696	BL	@4850	
44CE	4850	HP	18512			
44D0	0706	??	1798	SETO	R6	
44D2	C000	??	-16384	MOV	R0	R0
44D4	1310	??	4880	JEQ	+16	44F6
44D6	06A0	??	1696	BL	@4798	
44D8	4798	G?	18328			
44DA	2E00	.?	11776	?		
44DC	130C	??	4876	JEQ	+12	44F6
44DE	C1C8	??	-15928	MOV	R8	R7
44E0	0986	??	2438	SRL	8	R6
44E2	D1AF	??	-11857	MOVB	@FBFE(R15)	R6
44E4	FBFE	??	-1026			
44E6	0600	??	1536	DEC	R0	
44E8	06C6	??	1734	SWPB	R6	
44EA	C077	?w	-16265	MOV	*R7+	R1
44EC	1311	??	4881	JEQ	+17	4510
44EE	C0B7	??	-16201	MOV	*R7+	R2
44F0	8181	??	-32383	C	R1	R6
44F2	16FB	??	5883	JNE	-5	44EA
44F4	0452	?R	1106	B	*R2	
44F6	D064	?d	-12188	MOVB	@FF6A(R4)	R1
44F8	FF6A	?j	-150			
44FA	1307	??	4871	JEQ	+7	450A
44FC	06A0	??	1696	BL	@4730	
44FE	4730	G0	18224			
4500	1606	??	5638	JNE	+6	450E
4502	06A0	??	1696	BL	@46B2	
4504	46B2	F?	18098			
4506	C2A4	??	-15708	MOV	@FF6C(R4)	R10
4508	FF6C	?l	-148			
450A	06A0	??	1696	BL	@4822	
450C	4822	H"	18466			
450E	1066	?f	4198	JMP	+102	45DC
4510	109C	??	4252	JMP	-100	444A
4512	0201	??	513	LI	FF78	R1
4514	FF78	?x	-136			
4516	100B	??	4107	JMP	+11	452E

ADDR	DATA	TEXT	DECIMAL	CODE	SOURCE	DEST
4518	0201	??	513	LI	FF79	R1
451A	FF79	?y	-135			
451C	1008	??	4104	JMP	+8	452E
451E	0201	??	513	LI	FF7A	R1
4520	FF7A	?z	-134			
4522	1005	??	4101	JMP	+5	452E
4524	0201	??	513	LI	FF7C	R1
4526	FF7C	?|	-132			
4528	1002	??	4098	JMP	+2	452E
452A	0201	??	513	LI	FF7B	R1
452C	FF7B	?{	-133			
452E	A044	?D	-24508	A	R4	R1
4530	F460	?`	-2976	SOCB	@4132	*R1
4532	4132	A2	16690			
4534	1034	?4	4148	JMP	+52	459E
4536	C0C3	??	-16189	MOV	R3	R3
4538	1632	?2	5682	JNE	+50	459E
453A	06A0	??	1696	BL	@45E2	
453C	45E2	E?	17890			
453E	102F	?/	4143	JMP	+47	459E
4540	C0C3	??	-16189	MOV	R3	R3
4542	162D	?-	5677	JNE	+45	459E
4544	06A0	??	1696	BL	@4798	
4546	4798	G?	18328			
4548	3D00	=?	15616	DIV	R0	R4
454A	13E2	??	5090	JEQ	-30	4510
454C	5660	V`	22112	SZCB	@40A1	*R9
454E	40A1	@?	16545			
4550	0986	??	2438	SRL	8	R6
4552	0286	??	646	CI	R6	004E
4554	004E	?N	78			
4556	1323	?#	4899	JEQ	+35	459E
4558	0286	??	646	CI	R6	0045
455A	0045	?E	69			
455C	1306	??	4870	JEQ	+6	456A
455E	0286	??	646	CI	R6	004F
4560	004F	?O	79			
4562	16D6	??	5846	JNE	-42	4510
4564	F660	?`	-2464	SOCB	@40A1	*R9
4566	40A1	@?	16545			
4568	101A	??	4122	JMP	+26	459E
456A	F660	?`	-2464	SOCB	@422C	*R9
456C	422C	B,	16940			
456E	1017	??	4119	JMP	+23	459E
4570	C0C3	??	-16189	MOV	R3	R3
4572	1615	??	5653	JNE	+21	459E
4574	06A0	??	1696	BL	@4798	
4576	4798	G?	18328			
4578	3D00	=?	15616	DIV	R0	R4
457A	13CA	??	5066	JEQ	-54	4510
457C	06A0	??	1696	BL	@4754	
457E	4754	GT	18260			
4580	F660	?`	-2464	SOCB	@4074	*R9
4582	4074	@t	16500			
4584	0225	?%	549	AI	FFF9	R5
4586	FFF9	??	-7			
4588	1303	??	4867	JEQ	+3	4590
458A	0605	??	1541	DEC	R5	
458C	16C1	??	5825	JNE	-63	4510

ADDR	DATA	TEXT	DECIMAL	CODE	SOURCE	DEST
458E	1002	??	4098	JMP	+2	4594
4590	5660	V`	22112	SZCB	@45F9	*R9
4592	45F9	E?	17913			
4594	1004	??	4100	JMP	+4	459E
4596	5660	V`	22112	SZCB	@42DC	*R9
4598	42DC	B?	17116			
459A	F660	?`	-2464	SOCB	@4004	*R9
459C	4004	@?	16388			
459E	1099	??	4249	JMP	-103	44D2
45A0	C90B	??	-14069	MOV	R11	@FF86(R4)
45A2	FF86	??	-122			
45A4	06A0	??	1696	BL	@463A	
45A6	463A	F:	17978			
45A8	C206	??	-15866	MOV	R6	R8
45AA	06A0	??	1696	BL	@463A	
45AC	463A	F:	17978			
45AE	06C6	??	1734	SWPB	R6	
45B0	E206	??	-7674	SOC	R6	R8
45B2	1014	??	4116	JMP	+20	45DC
45B4	C90B	??	-14069	MOV	R11	@FF86(R4)
45B6	FF86	??	-122			
45B8	C189	??	-15991	MOV	R9	R6
45BA	06A0	??	1696	BL	@47E6	
45BC	47E6	G?	18406			
45BE	06C6	??	1734	SWPB	R6	
45C0	06A0	??	1696	BL	@47E6	
45C2	47E6	G?	18406			
45C4	100B	??	4107	JMP	+11	45DC
45C6	C90B	??	-14069	MOV	R11	@FF86(R4)
45C8	FF86	??	-122			
45CA	06A0	??	1696	BL	@463A	
45CC	463A	F:	17978			
45CE	1004	??	4100	JMP	+4	45D8
45D0	C90B	??	-14069	MOV	R11	@FF86(R4)
45D2	FF86	??	-122			
45D4	06A0	??	1696	BL	@47E6	
45D6	47E6	G?	18406			
45D8	06A0	??	1696	BL	@47C0	
45DA	47C0	G?	18368			
45DC	C2E4	??	-15644	MOV	@FF86(R4)	R11
45DE	FF86	??	-122			
45E0	045B	?[	1115	B	*R11	
45E2	C90B	??	-14069	MOV	R11	@FF88(R4)
45E4	FF88	??	-120			
45E6	06A0	??	1696	BL	@4798	
45E8	4798	G?	18328			
45EA	3D00	=?	15616	DIV	R0	R4
45EC	1391	??	5009	JEQ	-111	4510
45EE	06A0	??	1696	BL	@4754	
45F0	4754	GT	18260			
45F2	1002	??	4098	JMP	+2	45F8
45F4	C90B	??	-14069	MOV	R11	@FF88(R4)
45F6	FF88	??	-120			
45F8	0201	??	513	LI	40A6	R1
45FA	40A6	@?	16550			
45FC	04C2	??	1218	CLR	R2	
45FE	C2F1	??	-15631	MOV	*R1+	R11
4600	1387	??	4999	JEQ	-121	4510
4602	82C5	??	-32059	C	R5	R11
4604	1302	??	4866	JEQ	+2	460A
4606	05C2	??	1474	INCT	R2	
4608	10FA	??	4346	JMP	-6	45FE
460A	D2E0	??	-11552	MOVB	@000C	R11
460C	000C	??	12			
460E	098B	??	2443	SRL	8	R11
4610	0201	??	513	LI	409C	R1
4612	409C	@?	16540			
4614	C171	?q	-16015	MOV	*R1+	R5
4616	1327	?'	4903	JEQ	+39	4666
4618	82C5	??	-32059	C	R5	R11
461A	1302	??	4866	JEQ	+2	4620
461C	05C1	??	1473	INCT	R1	
461E	10FA	??	4346	JMP	-6	4614
4620	A091	??	-24431	A	*R1	R2
4622	C052	?R	-16302	MOV	*R2	R1
4624	1505	??	5381	JGT	+5	4630
4626	F660	?`	-2464	SOCB	@4072	*R9
4628	4072	@r	16498			
462A	0241	?A	577	ANDI	7FFF	R1
462C	7FFF	??	32767			
462E	1002	??	4098	JMP	+2	4634
4630	5660	V`	22112	SZCB	@4072	*R9
4632	4072	@r	16498			
4634	C901	??	-14079	MOV	R1	@FFFE(R4)
4636	FFFE	??	-2			
4638	1023	?#	4131	JMP	+35	4680
463A	C90B	??	-14069	MOV	R11	@FF88(R4)
463C	FF88	??	-120			
463E	06A0	??	1696	BL	@4870	
4640	4870	Hp	18544			
4642	1303	??	4867	JEQ	+3	464A
4644	06A0	??	1696	BL	@4880	
4646	4880	H?	18560			
4648	10FA	??	4346	JMP	-6	463E
464A	C0C3	??	-16189	MOV	R3	R3
464C	160E	??	5646	JNE	+14	466A
464E	04C6	??	1222	CLR	R6	
4650	3606	6?	13830	STCR	R6	8
4652	1E12	??	7698	SBZ	18	
4654	1F0B	??	7947	TB	11	
4656	1307	??	4871	JEQ	+7	4666
4658	1F0C	??	7948	TB	12	
465A	1305	??	4869	JEQ	+5	4666
465C	D2E4	??	-11548	MOVB	@FF7B(R4)	R11
465E	FF7B	?{	-133			
4660	130F	??	4879	JEQ	+15	4680
4662	1F0A	??	7946	TB	10	
4664	160D	??	5645	JNE	+13	4680
4666	0460	?`	1120	B	@445C	
4668	445C	D\	17500			
466A	1D01	??	7425	SBO	1	
466C	1E02	??	7682	SBZ	2	
466E	1F02	??	7938	TB	2	
4670	1603	??	5635	JNE	+3	4678
4672	06A0	??	1696	BL	@4880	
4674	4880	H?	18560			
4676	10FB	??	4347	JMP	-5	466E
4678	04C6	??	1222	CLR	R6	

ADDR	DATA	TEXT	DECIMAL	CODE	SOURCE	DEST
467A	D1A0	??	-11872	MOVB	@5000	R6
467C	5000	P?	20480			
467E	1D02	??	7426	SBO	2	
4680	C2E4	??	-15644	MOV	@FF88(R4)	R11
4682	FF88	??	-120			
4684	045B	?[	1115	B	*R11	
4686	C90B	??	-14069	MOV	R11	@FF88(R4)
4688	FF88	??	-120			
468A	D1C7	??	-11833	MOVB	R7	R7
468C	1309	??	4873	JEQ	+9	46A0
468E	06A0	??	1696	BL	@46B4	
4690	46B4	F?	18100			
4692	0227	?'	551	AI	FF00	R7
4694	FF00	??	-256			
4696	C907	??	-14073	MOV	R7	@FF7E(R4)
4698	FF7E	?~	-130			
469A	0207	??	519	LI	0100	R7
469C	0100	??	256			
469E	1006	??	4102	JMP	+6	46AC
46A0	C1C7	??	-15929	MOV	R7	R7
46A2	1602	??	5634	JNE	+2	46A8
46A4	0460	?`	1120	B	@4464	
46A6	4464	Dd	17508			
46A8	04E4	??	1252	CLR	@FF7E(R4)	
46AA	FF7E	?~	-130			
46AC	C907	??	-14073	MOV	R7	@FF80(R4)
46AE	FF80	??	-128			
46B0	10E7	??	4327	JMP	-25	4680
46B2	0707	??	1799	SETO	R7	
46B4	C90B	??	-14069	MOV	R11	@FF8A(R4)
46B6	FF8A	??	-118			
46B8	04C1	??	1217	CLR	R1	
46BA	06A0	??	1696	BL	@484E	
46BC	484E	HN	18510			
46BE	4000	@?	16384	SZC	R0	R0
46C0	06A0	??	1696	BL	@485A	
46C2	485A	HZ	18522			
46C4	C087	??	-16249	MOV	R7	R2
46C6	0982	??	2434	SRL	8	R2
46C8	0206	??	518	LI	0064	R6
46CA	0064	?d	100			
46CC	04C1	??	1217	CLR	R1	
46CE	3C46	<F	15430	DIV	R6	R1
46D0	0221	?!	545	AI	0030	R1
46D2	0030	?0	48			
46D4	0A81	??	2689	SLA	8	R1
46D6	B064	?d	-20380	AB	@FF72(R4)	R1
46D8	FF72	?r	-142			
46DA	DBC1	??	-9279	MOVB	R1	@FFFE(R15)
46DC	FFFE	??	-2			
46DE	04C5	??	1221	CLR	R5	
46E0	3D60	=`	15712	DIV	@4796	R5
46E2	4796	G?	18326			
46E4	C185	??	-15995	MOV	R5	R6
46E6	16F2	??	5874	JNE	-14	46CC
46E8	06A0	??	1696	BL	@485A	
46EA	485A	HZ	18522			
46EC	101E	??	4126	JMP	+30	472A
46EE	C90B	??	-14069	MOV	R11	@FF8A(R4)
46F0	FF8A	??	-118			

ADDR	DATA	TEXT	DECIMAL	CODE	SOURCE	DEST
46F2	D2E4	??	-11548	MOVB	@FF79(R4)	R11
46F4	FF79	?y	-135			
46F6	1619	??	5657	JNE	+25	472A
46F8	06A0	??	1696	BL	@47E4	
46FA	47E4	G?	18404			
46FC	0D00	??	3328	?		
46FE	1002	??	4098	JMP	+2	4704
4700	C90B	??	-14069	MOV	R11	@FF8A(R4)
4702	FF8A	??	-118			
4704	D064	?d	-12188	MOVB	@FF7C(R4)	R1
4706	FF7C	?|	-132			
4708	1307	??	4871	JEQ	+7	4718
470A	0205	??	517	LI	0006	R5
470C	0006	??	6			
470E	06A0	??	1696	BL	@47E4	
4710	47E4	G?	18404			
4712	0000	??	0	?		
4714	0605	??	1541	DEC	R5	
4716	16FB	??	5883	JNE	-5	470E
4718	D064	?d	-12188	MOVB	@FF79(R4)	R1
471A	FF79	?y	-135			
471C	1606	??	5638	JNE	+6	472A
471E	D064	?d	-12188	MOVB	@FF7A(R4)	R1
4720	FF7A	?z	-134			
4722	1603	??	5635	JNE	+3	472A
4724	06A0	??	1696	BL	@47E4	
4726	47E4	G?	18404			
4728	0A00	??	2560	SLA	16	R0
472A	C2E4	??	-15644	MOV	@FF8A(R4)	R11
472C	FF8A	??	-118			
472E	045B	?[	1115	B	*R11	
4730	D064	?d	-12188	MOVB	@FF6A(R4)	R1
4732	FF6A	?j	-150			
4734	0981	??	2433	SRL	8	R1
4736	0221	?!	545	AI	FFFB	R1
4738	FFFB	??	-5			
473A	1301	??	4865	JEQ	+1	473E
473C	0601	??	1537	DEC	R1	
473E	045B	?[	1115	B	*R11	
4740	D064	?d	-12188	MOVB	@FF6B(R4)	R1
4742	FF6B	?k	-149			
4744	2060	`	8288	COC	@4072	R1
4746	4072	@r	16498			
4748	045B	?[	1115	B	*R11	
474A	D064	?d	-12188	MOVB	@FF6B(R4)	R1
474C	FF6B	?k	-149			
474E	0241	?A	577	ANDI	1000	R1
4750	1000	??	4096			
4752	045B	?[	1115	B	*R11	
4754	C90B	??	-14069	MOV	R11	@FF8A(R4)
4756	FF8A	??	-118			
4758	04C1	??	1217	CLR	R1	
475A	04CB	??	1227	CLR	R11	
475C	1003	??	4099	JMP	+3	4764
475E	D1AF	??	-11857	MOVB	@FBFE(R15)	R6
4760	FBFE	??	-1026			
4762	0600	??	1536	DEC	R0	
4764	C1C6	??	-15930	MOV	R6	R7
4766	0987	??	2439	SRL	8	R7

ADDR	DATA	TEXT	DECIMAL	CODE	SOURCE	DEST
4768	0227	?'	551	AI	FFD0	R7
476A	FFD0	??	-48			
476C	110C	??	4364	JLT	+12	4786
476E	0287	??	647	CI	R7	0009
4770	0009	??	9			
4772	1B09	??	6921	JH	+9	4786
4774	058B	??	1419	INC	R11	
4776	3860	8`	14432	MPY	@4796	R1
4778	4796	G?	18326			
477A	C041	?A	-16319	MOV	R1	R1
477C	1606	??	5638	JNE	+6	478A
477E	A087	??	-24441	A	R7	R2
4780	C042	?B	-16318	MOV	R2	R1
4782	C000	??	-16384	MOV	R0	R0
4784	16EC	??	5868	JNE	-20	475E
4786	C2CB	??	-15669	MOV	R11	R11
4788	1602	??	5634	JNE	+2	478E
478A	0460	?`	1120	B	@444A	
478C	444A	DJ	17482			
478E	C141	?A	-16063	MOV	R1	R5
4790	C2E4	??	-15644	MOV	@FF8A(R4)	R11
4792	FF8A	??	-118			
4794	045B	?[	1115	B	*R11	
4796	000A	??	10	?		
4798	C17B	?{	-16005	MOV	*R11+	R5
479A	9185	??	-28283	CB	R5	R6
479C	1307	??	4871	JEQ	+7	47AC
479E	D1AF	??	-11857	MOVB	@FBFE(R15)	R6
47A0	FBFE	??	-1026			
47A2	0600	??	1536	DEC	R0	
47A4	9185	??	-28283	CB	R5	R6
47A6	1302	??	4866	JEQ	+2	47AC
47A8	C000	??	-16384	MOV	R0	R0
47AA	16F9	??	5881	JNE	-7	479E
47AC	C000	??	-16384	MOV	R0	R0
47AE	1307	??	4871	JEQ	+7	47BE
47B0	04C6	??	1222	CLR	R6	
47B2	D1AF	??	-11857	MOVB	@FBFE(R15)	R6
47B4	FBFE	??	-1026			
47B6	0600	??	1536	DEC	R0	
47B8	0286	??	646	CI	R6	2000
47BA	2000	?	8192			
47BC	13F7	??	5111	JEQ	-9	47AC
47BE	045B	?[	1115	B	*R11	
47C0	C046	?F	-16314	MOV	R6	R1
47C2	0241	?A	577	ANDI	FF00	R1
47C4	FF00	??	-256			
47C6	2A41	*A	10817	XOR	R1	R9
47C8	C049	?I	-16311	MOV	R9	R1
47CA	0941	?A	2369	SRL	4	R1
47CC	2849	(I	10313	XOR	R9	R1
47CE	0241	?A	577	ANDI	FF00	R1
47D0	FF00	??	-256			
47D2	0941	?A	2369	SRL	4	R1
47D4	2A41	*A	10817	XOR	R1	R9
47D6	0B71	?q	2929	SRC	7	R1
47D8	2A41	*A	10817	XOR	R1	R9
47DA	06C9	??	1737	SWPB	R9	
47DC	045B	?[	1115	B	*R11	

ADDR	DATA	TEXT	DECIMAL	CODE	SOURCE	DEST
47DE	D1AF	??	-11857	MOVB	@FBFE(R15)	R6
47E0	FBFE	??	-1026			
47E2	1001	??	4097	JMP	+1	47E6
47E4	C1BB	??	-15941	MOV	*R11+	R6
47E6	C90B	??	-14069	MOV	R11	@FF8C(R4)
47E8	FF8C	??	-116			
47EA	C0C3	??	-16189	MOV	R3	R3
47EC	160D	??	5645	JNE	+13	4808
47EE	1D10	??	7440	SBO	16	
47F0	1F1B	??	7963	TB	27	
47F2	1602	??	5634	JNE	+2	47F8
47F4	1F16	??	7958	TB	22	
47F6	1303	??	4867	JEQ	+3	47FE
47F8	06A0	??	1696	BL	@4880	
47FA	4880	H?	18560			
47FC	10F6	??	4342	JMP	-10	47EA
47FE	3206	2?	12806	LDCR	R6	8
4800	1E10	??	7696	SBZ	16	
4802	C2E4	??	-15644	MOV	@FF8C(R4)	R11
4804	FF8C	??	-116			
4806	045B	?[	1115	B	*R11	
4808	1E01	??	7681	SBZ	1	
480A	1F02	??	7938	TB	2	
480C	13F5	??	5109	JEQ	-11	47F8
480E	D806	??	-10234	MOVB	R6	@5000
4810	5000	P?	20480			
4812	1E02	??	7682	SBZ	2	
4814	1F02	??	7938	TB	2	
4816	1303	??	4867	JEQ	+3	481E
4818	06A0	??	1696	BL	@4880	
481A	4880	H?	18560			
481C	10FB	??	4347	JMP	-5	4814
481E	1D02	??	7426	SBO	2	
4820	10F0	??	4336	JMP	-16	4802
4822	C0C3	??	-16189	MOV	R3	R3
4824	1303	??	4867	JEQ	+3	482C
4826	1D02	??	7426	SBO	2	
4828	1E01	??	7681	SBZ	1	
482A	045B	?[	1115	B	*R11	
482C	1D1F	??	7455	SBO	31	
482E	3224	2$	12836	LDCR	@FFFA(R4)	8
4830	FFFA	??	-6			
4832	1E0D	??	7693	SBZ	13	
4834	3324	3$	13092	LDCR	@FFFE(R4)	12
4836	FFFE	??	-2			
4838	D064	?d	-12188	MOVB	@FF7D(R4)	R1
483A	FF7D	?}	-131			
483C	1301	??	4865	JEQ	+1	4840
483E	1D12	??	7442	SBO	18	
4840	045B	?[	1115	B	*R11	
4842	C064	?d	-16284	MOV	@FF76(R4)	R1
4844	FF76	?v	-138			
4846	6064	`d	24676	S	@FF74(R4)	R1
4848	FF74	?t	-140			
484A	0221	?!	545	AI	FFF6	R1
484C	FFF6	??	-10			
484E	A07B	?{	-24453	A	*R11+	R1
4850	D7E4	??	-10268	MOVB	@0003(R4)	*R15
4852	0003	??	3			

ADDR	DATA	TEXT	DECIMAL	CODE	SOURCE	DEST
4854	1000	??	4096	JMP	0	4856
4856	D7C1	??	-10303	MOVB	R1	*R15
4858	045B	?[	1115	B	*R11	
485A	0201	??	513	LI	2020	R1
485C	2020		8224			
485E	B064	?d	-20380	AB	@FF72(R4)	R1
4860	FF72	?r	-142			
4862	0202	??	514	LI	000E	R2
4864	000E	??	14			
4866	DBC1	??	-9279	MOVB	R1	@FFFE(R15)
4868	FFFE	??	-2			
486A	0602	??	1538	DEC	R2	
486C	16FC	??	5884	JNE	-4	4866
486E	045B	?[	1115	B	*R11	
4870	C0C3	??	-16189	MOV	R3	R3
4872	1604	??	5636	JNE	+4	487C
4874	1F1B	??	7963	TB	27	
4876	1601	??	5633	JNE	+1	487A
4878	1F15	??	7957	TB	21	
487A	045B	?[	1115	B	*R11	
487C	1F02	??	7938	TB	2	
487E	045B	?[	1115	B	*R11	
4880	C04C	?L	-16308	MOV	R12	R1
4882	020C	??	524	LI	0024	R12
4884	0024	?$	36			
4886	30E0	0?	12512	LDCR	@4073	3
4888	4073	@s	16499			
488A	1FF5	??	8181	TB	-11	
488C	1304	??	4868	JEQ	+4	4896
488E	30E0	0?	12512	LDCR	@4074	3
4890	4074	@t	16500			
4892	1FF5	??	8181	TB	-11	
4894	1602	??	5634	JNE	+2	489A
4896	C301	??	-15615	MOV	R1	R12
4898	045B	?[	1115	B	*R11	
489A	C301	??	-15615	MOV	R1	R12
489C	0460	?`	1120	B	@445C	
489E	445C	D\	17500			
48A0	ABCD	??	-21555	A	R13	@0B80(R15)
48A2	0B80	??	2944			
48A4	0B80	??	2944	SRC	8	R0
48A6	0B80	??	2944	SRC	8	R0
48A8	0B80	??	2944	SRC	8	R0
48AA	0B80	??	2944	SRC	8	R0
48AC	0B80	??	2944	SRC	8	R0
48AE	045B	?[	1115	B	*R11	
48B0	FFFF	??	-1	SOCB	*R15+	*R15+
48B2	FFFF	??	-1	SOCB	*R15+	*R15+
48B4	FFFF	??	-1	SOCB	*R15+	*R15+
48B6	FFFF	??	-1	SOCB	*R15+	*R15+
48B8	FFFF	??	-1	SOCB	*R15+	*R15+
48BA	FFFF	??	-1	SOCB	*R15+	*R15+
48BC	FFFF	??	-1	SOCB	*R15+	*R15+
48BE	FFFF	??	-1	SOCB	*R15+	*R15+
48C0	FFFF	??	-1	SOCB	*R15+	*R15+
48C2	FFFF	??	-1	SOCB	*R15+	*R15+
48C4	FFFF	??	-1	SOCB	*R15+	*R15+
48C6	FFFF	??	-1	SOCB	*R15+	*R15+
48C8	FFFF	??	-1	SOCB	*R15+	*R15+

ADDR	DATA	TEXT	DECIMAL	CODE	SOURCE	DEST
48CA	FFFF	??	-1	SOCB	*R15+	*R15+
48CC	FFFF	??	-1	SOCB	*R15+	*R15+
48CE	FFFF	??	-1	SOCB	*R15+	*R15+
48D0	FFFF	??	-1	SOCB	*R15+	*R15+
48D2	FFFF	??	-1	SOCB	*R15+	*R15+
48D4	FFFF	??	-1	SOCB	*R15+	*R15+
48D6	FFFF	??	-1	SOCB	*R15+	*R15+
48D8	FFFF	??	-1	SOCB	*R15+	*R15+
48DA	FFFF	??	-1	SOCB	*R15+	*R15+
48DC	FFFF	??	-1	SOCB	*R15+	*R15+
48DE	FFFF	??	-1	SOCB	*R15+	*R15+
48E0	FFFF	??	-1	SOCB	*R15+	*R15+
48E2	FFFF	??	-1	SOCB	*R15+	*R15+
48E4	FFFF	??	-1	SOCB	*R15+	*R15+
48E6	FFFF	??	-1	SOCB	*R15+	*R15+
48E8	FFFF	??	-1	SOCB	*R15+	*R15+
48EA	FFFF	??	-1	SOCB	*R15+	*R15+
48EC	FFFF	??	-1	SOCB	*R15+	*R15+
48EE	FFFF	??	-1	SOCB	*R15+	*R15+
48F0	FFFF	??	-1	SOCB	*R15+	*R15+
48F2	FFFF	??	-1	SOCB	*R15+	*R15+
48F4	FFFF	??	-1	SOCB	*R15+	*R15+
48F6	FFFF	??	-1	SOCB	*R15+	*R15+
48F8	FFFF	??	-1	SOCB	*R15+	*R15+
48FA	FFFF	??	-1	SOCB	*R15+	*R15+
48FC	FFFF	??	-1	SOCB	*R15+	*R15+
48FE	FFFF	??	-1	SOCB	*R15+	*R15+

IF YOU OWN A TEXAS INSTRUMENTS TI 99/4A HOME COMPUTER, YOU NEED TO KNOW:

WHAT IS UNISOURCE?

- The leading mail order supplier of TI 99/4A equipment and software
- Has availablility of over a thousand different items for the TI 99/4A
- Provides a complete Encyclopedia/Catalog at a nominal charge (or free with an order)
- Offers discount prices
- Provides a toll free order line for your convenience
- Accepts MasterCard or Visa charge cards at no additional charge
- Normally ships in-stock merchandise to you within 48 hours

Coming Soon!!

The TI 99/4A UNISOURCE Encyclopedia/Catalog

You will find this catalog represents the most complete collection of software, peripherals and accessories available today for your TI 99/4A Home Computer. It includes all software and peripherals, and hundreds of new & exciting software packages from both TI and third-party developers. There's also a collection of peripherals, accessories and supplies available for your 99/4A.

It will be available to you for ONLY $3.00 refundable with your first order. And, you'll automatically get future updates at no extra cost. For our current customers who already own our catalog, a new edition will be sent to you free.

Our new discount program will allow you to receive additional savings on 3rd party software and books ... 10% OFF on orders of $50 or more ... 20% OFF on orders of $100 or more (not including shipping).

Call today. Give us your name, address, Visa or MasterCard number, and we'll charge the $3.00 plus $1.50 shipping to your account. Or, send check or money order for $4.50 to us at the address below. Remember, the $3.00 is refundable with your first order!

PERIPHERALS & SOFTWARE

From Double-Side Disk Drives to dust covers ... everything you might need for your TI Home Computer:

TI Software • TI Hardware
CorComp Peripherals • Atarisoft
Percom Data Disk System
Signalman Modems • Gemini Printers
Imagic • Sega • TI COUNT Business Software
Scott Adams Adventures
AND MANY, MANY MORE!

TE-1200

A 1200 Baud Terminal Emulator Package

The TE-1200 Terminal Emulator Disk Program will support 1200 baud asynchronous modems. It is functionally compatible with the normal TTY, and file transmit functions of the TI TE-II Cartridge. The use of 1200 baud should significantly reduce the connect times on online services such as Texnet, Compuserve or Dowjones since it will receive data from them 4 times faster. Requires Disk System, 32K Memory and Editor/Assembler..... **$49.95**

Ordering Information

ORDER HOTLINE - TOLL FREE

1-800-858-4580

IN TEXAS CALL 1-806-745-8834

UNISOURCE
ELECTRONICS, INC.

Just give us your name, shipping address and Visa or Master Card number and we'll charge the order to your account. Please include $2.00 shipping and handling per order for software ($5 for peripherals).** Texas residents also add 5% sales tax. For mail-in orders, send to **P.O. Box 64240, Lubbock, Texas 79464.** Allow two weeks clearance for personal checks.

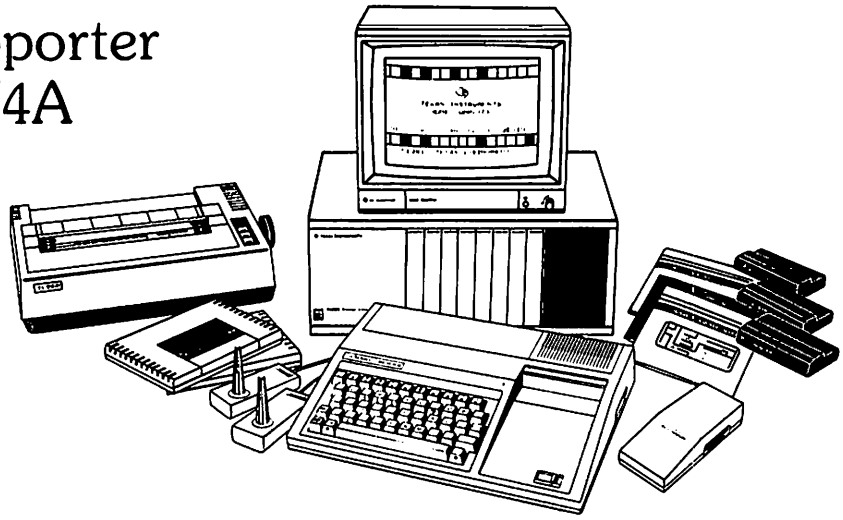
Normal orders of in-stock items will be shipped within a few days of order placement. Allow two weeks for delivery on any phone orders.

**Continental U.S. only. All others include actual airmail and insurance. A separate shipping charge of $10 for the PE Box will apply.

(prices and terms may change without prior notice)

MULTICOM, INC.

Leading Supporter
of the TI-99/4A



Has the technology, products, and service to supply your TI-99/4A expansion needs.

Our product line includes:

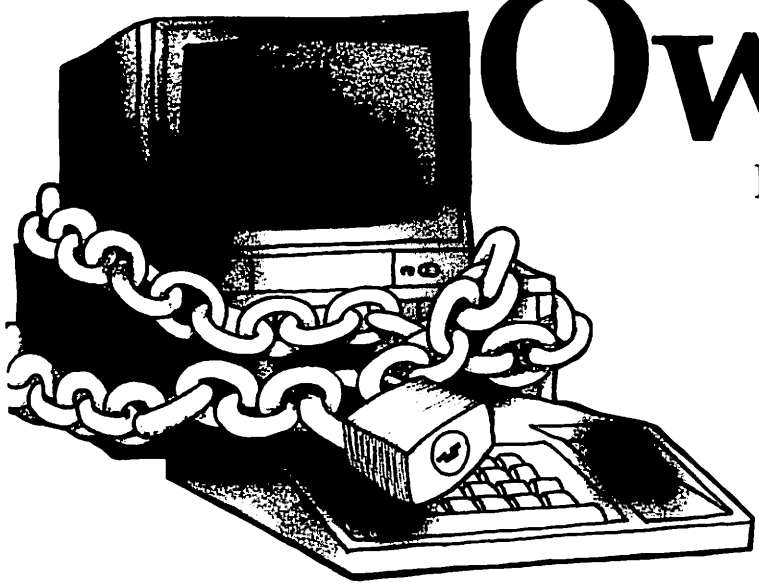
- 32K Memory Expansion
- RS232 Ports
- Parallel Ports
- Cables
- ABC Switches
- Software
- Printers

For our current catalog.

Call **(801) 572-6272**

or write **MULTICOM, INC.**
P.O. Box 1693
Sandy, Utah 84091

Attention TI99 4/A Owners!



Introducing two powerful tools to **UNLOCK** inaccessible files, **PROTECT** valuable originals, **DUPLICATE** diskettes and **SAVE** you money!

1. DISK FIXER PROGRAM: Unlocks the Secrets of the Disk.

DISK FIXER PROGRAM allows you to probe the inner mysteries of the 99/4 disk format and learn many things.

DISK FIXER is a must for the serious 99/4 hobbyist who wants to do more than shove command modules into the GROM port and also for anyone who has a "sick disk" which suffers from a damaged directory. Disk Fixer lets you recover unscathed information as if you were sorting out the broken eggs from the carton that had been dropped. This is done by reading the disk by sector number rather than file name.

DISK FIXER lets you search your disk by sector rather than by file name and display/print the actual binary contents with a single command. You can change any byte on any sector, even move data from one sector to another.

2. Safeguard Your Masters • Fast • With SUPER-DUPER

Use the high-speed SUPER- DUPER PROGRAM CARTRIDGE to duplicate your disks, single and double sided,then lock away your originals. For single-drive systems, a special data-compression routine stores most of your information in expanded memory to reduce disk swapping. Most disks are copied in one or two passes — and SUPER-DUPER works even faster with multiple-drive systems.

The program automatically formats blank diskettes before copying and allows you to verify your backups byte for byte against the originals.

NAVARONE INDUSTRIES, Inc.
510 Lawrence Expressway, #800
Sunnyvale, CA 94086

NAVARONE

(408) 985-2932