Introduction to Programming the TAPR/AMSAT **DSP-93**

Ron Parsons, WSRKN (w5rkn@ amsat.org)
Don Haselwood, K4JPJ
Bob Stricklin, N5BRG @brg@tapr.org)

Tucson Amateur Packet Radio, Corp.
8987-309 E. Tanque Verde Rd #337
Tucson, AZ 85749-9399
tapr@tapr.org • http://www.tapr.org/tapr

Abstract

The purpose of this paper is to give a brief overview to assist potential programmers, new to the TAPR/AMSAT DSP-93 environment, insight into the tools and techniques available when developing for the DSP-93. The full developers/programming guide is available from TAPR [I].

Introduction

Any introduction regarding DSP-93 programming must begin with discussing how to locate and secure reference materials. The first step is to locate your local Texas Instruments distributor and call them. Local distributors have been known to give free access to their literature room. Books that you should be looking for include:

- **TMS320C2x User's Guide By Texas Instruments; Document # SPRUOl4C**
  This book covers the TMS320C25 DSP chip used in the DSP-93. It covers the chip's electrical properties, memory models, interrupt processing, and, of course, the instruction set.

- **Linear Circuits; Data Conversion, DSP Analog Interface, and Video Interface; Data Book Volume 2 By Texas Instruments; Document # SLYDOO4A.**
  This book covers the TLC32044CN AI0 chip used in the DSP-93. It covers the chip's electrical properties, configuration, etc.

- **Digital Signal Processing Applications with the TMS320 Family; Theory, Algorithms, and Implementations**
  Volume 1 Document # SPRAO12A
  Volume 2 Document # SPRA016
  Volume 3 Document # SPRA017
  These books cover various DSP algorithms which may or may not be useful to you.

- **Digital signal processing with the TMS320C25** Chassaing, Rulph, published 1990 by Wiley, New York 464 pg
  ISBN 0471510661

- **Digital Signal Processing: A laboratory approach using PC-DSP.** Alkin, Oktay, published 1994 by Prentice-Hall, Inc.
  ISBN O-13-328139-6
  An introduction to Digital Signal Processing techniques. Comes with a DOS program which can compute FIR coefficients, among other things..

The next basic step is to get a good feel for the basic architecture of the DSP-93. Spend some time examing the schematics and get a good look at the interconnect lines. The basic system includes a DSP engine board and a radio/computer interface board. The DSP Engine, contains the TMS32OC25 DSP, 32K by 16 bits of program and data memory - upgradable to 64K, the clock circuitry (40MHz) and some programmable array logic for system I/O. The Radio/Computer Interface Board, top board, contains two eight pin female mini-DIN connectors for radio interfacing. Incoming radio signals pass through a voltage divider to establish the initial levels, then through an eight channel multiplex chip. The multiplex chip then feeds the single A/D input with either of the radio inputs or one of the six auxiliary inputs. The Texas Instruments TLC32044 Analog I/O chip is used to sample and update the input signal at a rate of up to 45K operations per second and includes aliasing filters. This board also communicates with your computer at speeds up to 19.2K baud using a serial connection and, with special programming, this can be increased to the maximum rate attainable by a 16C550 and your computer. [2,3]

The next step is to begin to get comfortable with either of the development systems (Mac [5] or Windows [4]) and play with assembling available source. A low cost shareware assembler, TASM TMS320-25 Assembler, is available for code development and both the Mac and Windows interfaces provide near seamless use of the assembler. Code development and testing can become a quick task.

What follows is a basic overview of various areas needed, when developing for the DSP-93.

Memory Map

The following memory map exists after you have entered a ‘G’ command from the Monitor. The ‘G’ command is executed after a program download using DSPLOAD.EXE or the windows program D93WE. The unit is operating entirely in static RAM and in the high speed mode at this point.

```
      PROGRAM                 DATA

 0000  Interrupts        0000 TMS320C25
      and                      MEMORY MAPPED
      Reserved                 REGISTERS
 O O l F  -               0 0 0 5  - -
                               Reserved
                          0 0 6 0  -
                               On DSP Block B2
                          0 0 7 F  -
      Reserved
      For  Monitor        O l F F  -
                          0 2 0 0  -
                               On DSP Block BO
                          0 2 F F  -
                          0 3 0 0  - -
                               On DSP Block Bl
                          0 3 F F  -
                          0 4 0 0  -
                               Page 8 Data
                                      Memory
 O F F F  -
 l-000 TINT
 1002 RINT
 1004 XINT e
 1006 TRAP
 1 0 0 8  -

      User Program             User Data
      Memory Space             Memory Space

 F F F F  -               F F F F  -
```

The Harvard architecture used in the TMS320 is quite different from the typical microprocessor. There are two memories utilized during one instruction-data and program. Until experience is gained working with this architecture, it is easy to forget this basic principle. There are two memories active at the same time. By having two memories, a single instruction can load a word out of program memory and do something with a word out of data memory, all within the same cycle. It facilitates implementing filters and other digital signal processing algorithms. For example, stepping down a table of constants, such as filter coefficients and doing a multiply and add to accumulator with a data array of signal values can be accomplished at full machine cycle speed with no overhead for instruction fetches, nor double accessing of a single memory.

Program and data memory can change, and keeping track of “what is which” is needed. How this is done is largely a function of how the DSP-93 design uses the TMS320. Therefore, the TMS literature will not give the whole story necessary to understand the DSP-93.

The DSP-93 has a number of physical memories. Some memory is internal to the TMS32OC25 chip and other is external. The tables below outline the physical memories, as well as the conditions which determine whether they are used by the TMS320C25 as program or data.

Internal memory (TMS320C25)

Label	Can be configured as	Size
BO	Data or program *	OlOOh
Bl	Data only	Olooh
B2	Data only	OOlOh

* BO is in data mode after a hardware reset, Monitor reset, or CNFD instruction. It is program memory after a Monitor ‘G’ command, or CNFP instruction.

External memory

How the external memories are configured depends on the state of the XF bit. Hardware reset turns the XF bit ON, as will the SXF instruction. RXF turns it OFF. SRAM U104, U105, UllO, and Ul 11 come with the basic kit and provide two 32K memories (one program and one data). Four more SRAM IC’s will raise the amount to 64K for both program and data memory.

(1) - XF line/bit is ON after a hardware reset, after a Monitor reset command (‘R’), or after a SXF instruction. RXF turns the bit off, and it is also turned off after a Monitor ‘G’ command.

(2) - Data addresses below 0400h access internal memory in the TMS320 and therefore are not available for use in the external memory.

(3) - Addresses FFOOh - FFFFh access internal memory, BO, when BO has been configured as program memory (normally after a Monitor ‘G’ command).

Note that all memory is organized as 16 bit words, and not 8 bit bytes. Along the same lines remember that the accumulator is 32 bits long so be careful about unintended sign extension.

Internal memory, BO, is configured to the data mode after a hard reset. Also, the Monitor in EPROM configures BO to data when a ‘R’ (reset) command is executed, and to program when a ‘G’ command is executed. Switching BO modes is done with the instructions CNFP and CNFD, (set program, and set data, respectively), or the hardware reset that switches it to data.

External memory is switched via the XF line out of the TMS320. This line is controlled by the XF bit. A hardware reset sets this bit high. The bit can be set/reset by the instructions SXF/RXF, respectively.

IC number	XF = ON(l) "Slow speed"		XF = OFF(O) "Fast speed"	
SRAM				
u104, u105	Data	0000 - 7FFF(2)	Prog	0000 - 7FFF(2)
U106, U107	Data	8000 - FFFF	Prog	8000 - FFFF(3)
U108, U109	N.A.	....-....	Data	0000 - 7FFF(2)
UllO, Ulll	N.A.	....-....	Data	8000 - FFFF
EPROM				
u102, u103	Prog	0000 - 7FFF(2)	N.A.	....-....

When the XF bit is high, such as after power-up reset, the EPROM, which contains the Monitor programs, is active as program memory. This makes it possible to get the machine running with something intelligent. XF also selects a low speed mode, so that the processor is slow enough to accommodate the EPROM. XF also makes external memories U104,5,6,7 switch to data mode and U108,9,10,11 not accessible. This arrangement allows the EPROM Monitor to load program into data memory (U104,5,6,7 and BO). This switching is necessary since the TMS320 does not execute instructions which store anything into program memory; the downloaded program is placed into the DSP-93 as data.

With the Monitor program executing out of EPROM, the usual step is to download a program from a general purpose computer. The program being downloaded is stored as data, in data memory, which is U104,5,6,7 at this time. Upon completion of the downloading, the general purpose computer issues a ‘G’ command which causes the DSP-93 Monitor to turn off the XF bit. This switches U104,5,6,7 from data memory to program memory. The Monitor also jumps to location 1008h (of program memory) to start the program that was just loaded (into what was data memory).

With XF low, program normally runs out of external SRAM, U104,5,6,7, using U108,9,10,11 for data storage. BO, B 1, and B2 can also be used. BO as stated before can be configured either as data or program, though remember that the Monitor sets it to program after the loading process completes and a ‘G’ command is given. Operating out of internal memory is somewhat faster than external memory and may be needed for time critical operations. The TMS320C2x User’s Guide shows timings for instructions according to the memory combination being used.

Since BO can be switched between data and program, it can be loaded with a program which can be executed. During the normal program downloading process, BO is configured as data, so it can be loaded with program no differently than U104,5,6,7. When the loading process completes, the Monitor does a ‘G’ command and jumps to 1008h. The program at 1008h can then jump to BO and execute the code which was loaded. When BO is configured as program, it is no longer at locations 0200h - 02FFh, but occupies FFOO - FFFFh. Therefore, the code assembled at 0200h - 02FFh must be capable of executing properly when moved to locations FFOOh - FFFFh. The jump from the program in U104,5,6,7 will be to FFxyh, if the beginning of the code loaded into BO is 02xyh.

External memory is not accessed for data addresses below 04OOh, as these are reserved for the TMS320. Also, FFOOh - FFFFh of program memory is not accessed when BO is configured as program.

Note that some addresses are really registers within the TMS320, such as locations 0 and 1 which are used to load/receive the serial shift register data to/from the AI0 chip. The TMS manual covers these in detail.

External memory, (prog, fast), 0400h - lOOOh holds the Monitor which is used when in the fast mode. If these locations are blasted, such as with a Monitor Fill command, the Monitor is lost and a hard reset is required.

The TMS320 cannot load and store (i.e. move) data from program memory to program memory. Data-to-data memory can be accomplished, as well as program-to-data and data-to-program. Therefore, it is not likely that a runaway program will blast the Monitor stored in 0400h - lOOOh.

The Monitor uses BO during the transition from EPROM to the SRAM or from low speed operation to high speed. Therefore, it is not possible to load the entire 256 words with program, or Fill it via the Monitor. If a reset command is executed by the Monitor, locations 0200h - 0275h are overwritten with “stuff’ from the Monitor, wrecking what might have been downloaded into BO. As long as a Monitor reset does not occur between the loading of BO and utilization of BO (either as data or program), then the full page can be used. Otherwise, only those locations not used by the Monitor can be used (0276h - 02FFh).

DSP-93 Firmware - Monitor Operation

When the DSP-93 is powered up, the firmware Monitor takes control of the unit. The Monitor conditions the TMS320C25 and then begins polling the serial data link looking for single character instructions to execute. The action taken by the Monitor during each operation will be explained in more detail here.

After initialization is complete, the Monitor enters a polling loop checking the serial port for an input character. The Monitor is not case sensitive and an entry of '?' will bring a listing of the Monitor version and the commands. The commands available in the monitor are as follows:

A-AR REGISTERS
This command displays the value contained in the eight 16 bit registers on the TMS32OC25. The first register is used by the Monitor and so the value in register zero will normally be 0400 hex. This is what you should see when you enter 'A' after power up.

D-DUMP MEMORY
The dump memory command displays the specified block of program or data memory. You must specify whether you want to use program or data memory. Then, enter the hex starting address and the ending address of the memory block. The memory contents will then be presented in block form with one 16 bit data address and eight 16 bit data values on each line.

F-FILL MEMORY
The Fill memory command works like the data dump command except it is filling memory with a 16 bit hex value. If you write over any program areas, you will kill the Monitor.

G-FLIP & RUN PROGRAM @ 1008h
The 'G' command is used after you have loaded a program and you are ready to run it. The Flip referred to here means that the Monitor is flipping from the EPROM over into the static RAM.

H-INTEL LOADER HIGH BITS
This is an Intel style hex loader for placing bits in data memory. To execute properly, the DSP-93 must be in the LOW speed state. The data is expected in the following sequence; count (8 bits), byte address (16 bits) (the byte address is twice the word address), Intel hex code command (8 bits), some quantity (count) of data (8 bits) and finally an 8 bit checksum. If no checksum errors are found, the data is placed into memory in the address locations specified. If an error occurs, a checksum error message is transmitted via the serial port. When the loader is finished, control is returned to the Monitor.

J-JUMP TO XXXX & RUN
JUMP and run executes in the same manner as the 'G' command except execution begins at the specified address. The interrupt vectors must still be in place if you are going to allow interrupts to occur.

L-INTEL LOADER LOW BITS
The 'L' command works exactly like the 'H' command except it deals with the lower 8 bits of the 16 bit words.

M-MODIFY WORD
This command is used to change the contents of a 16 bit memory location in program or data memory. The command can be used to force the D/A output of the TLC3204X to a particular value. This can be done by modifying location 0001. D/A changes will only occur if the AI0 chip is active.

P-PROGRAM
This command is used to launch one of the firmware programs located in the DSP-93 EPROM. '?' will list all programs available.

R-RESET
Entering an 'R' will cause the DSP-93 to go through a soft reset. The results of this should be equivalent to a hardware reset. If you have entered the 'G' command and you are working in static RAM, the DSP-93 will bounce back to the EPROMs just as if you hit the reset button. Some of the areas in data RAM are initialized during a reset cycle.

S-SHOW WORD
This command is used to display the contents of a particular memory location.

T-MEMORY TEST
This command tests the current data memory for errors. The memory is tested by writing OOOOh, 5555h, and OFFFFh into every location and reading it back. The test will loop through all locations and give a '*' prompt if no errors occur. If an error occurs, the location of the error will be reported along with the value written and the value returned. Both RAM banks can be tested by issuing the 'T' command in slow mode (i.e., after a reset) and again in fast mode (i.e., after a 'G' command).

Sine/Cosine table

In versions of the Monitor prior to Version 2.17, the sine/cosine table located in EPROM was copied to RAM memory when the 'G' command was issued. Beginning with Version 2.17, although the table is still in EPROM, it is no longer copied to RAM. This 1) frees up some Monitor code area 2) eliminates the possibility the sin table will write over code the user loads, and 3) allows user to locate the sin table in data memory which is the place he will really need it.

The table in EPROM is a 540 degree, 0.25 degree step, sine/cosine table is located in EPROM from 767Fh to 7EEFh and is scaled by 2*15. Sine begins at 767Fh; cosine begins at 77E7h.

The Parts of a DSP-93 Program

One of the best ways to learn DSP-93 programming is to read, study, and understand existing programs. A lot of source code is provided on the system diskettes. Use this valuable resource. Not every program will use all these parts, and not all that do use them will use them exactly as shown. If they did, there would be only one program!

Header (Instructions, Copyrights, Disclaimers)

It's a good idea to have a header that tells what the program is, how to assemble and execute it, who wrote it and a Copyright and Disclaimer statement. See the sample program for more examples.

The Include Files

There are currently five include files that define most of the constants that you will use when programming the DSP-93. It is *strongly* encouraged that all programs include these files and use the constants therein. This will make your program easier to understand and will reduce programming errors. These files are:

MACROS.INC
Purpose: This macro defines the origin in the code segment that the initialized data is to begin. This macro defines storage in the code segment while also creating a symbol representing the location the data will reside in data memory after an initialization block move.

MONITOR.INC
Purpose: This file defines the DSP-93 Monitor functions and addresses. Symbolic names for all of the entry points have been assigned. This file should be used instead of the absolute addresses since a linker does not exist. The file may also contain any macros defined in the future that can help in using the Monitor functions.

PORTS.INC
Purpose: This file defines the DSP-93 I/O ports and I/O bits. The file assigns symbolic names for each of the ports and bits. The user should make use of the TASM bitwise AND (8~) and OR (I) operators to manipulate the bits. The programmer is discouraged from using hex values in programs, as other users may have to read the listings, and the symbolic names assign greater meaning to the code.

SERIAL.INC
Purpose: Define symbols used in configuring the serial port for the DSP-93. The symbols in this file describe the 16550 UART. Programmers should use this file instead of using magic numbers in their programs.

REGS.INC
Purpose: This file defines the register replacements for the TASM assembler. The register file AR0 through AR7 are defined as well as the memory mapped registers defined, by the processor.

The following lines of code should be in your source after the header:

```
          .NOLIST
#include  "MACROS.INC"
#include  "REGS.INC"
#include  "PORTS.INC"
#include  "MONITOR.INC"
#include  "SERIAL.INC"
          .LIST
```

Memory Location Equates

Storage variable locations in internal and/or external memory must be defined in your program. These variables are defined using an .EQU directive to assign values to labels. For example:

```
·  global  variables
b.JFI   .EQU   060h   ;A10 input buffer
BUFO    .EQU   061h   ;A10 output buffer
DO      .EQU   062h   ;data output buffer
```

Program Constant Equates

Constants used in your program sho.uld be assigned a label and that label be given a value using an .EQU directive. It is *strongly* encouraged that all programs use labels for constants rather than using constants in the body of the program.

Program Origin

The TASM assembler does not, by default, set the DSP-93 starting address of 10OOh. So be sure to include the directive: `.ORG 1 0 0 0h`

Interrupt Vectors

The first four instructions of the program handle the various interrupt vectors and must branch to the appropriate labels. The Timer and Trap interrupts will not occur in the DSP-93 programs, so they branch to the program starting label GO. Program execution begins at 1008h, just following the TRAP interrupt vector.

```
              ;Define Vectors
    B  GO     ;branch to program
              ;  start (TINT) Timer
    B  RINT   ;A10 receive interrupt
              ;  service routine
    B  XINT   ;A10 transmit interrupt
              ;  service routine
    B  GO     ;branch to program
              ; start (TRAP)
GO  DINT      ;program starts here
```

Initialization of the Math, Serial and Memory Model

There are various parameters for specifying how mathematical operations, the DSP chip serial IO, and memory models will be handled. Your program should set these values at the beginning of the program.

Initialization of the AI0

The initialization of the AI0 chip is probably the most confusing aspect of DSP-93 programming. However, using the recipes written by the development group should enable you to start the chip sampling at the rate you desire without problem.

The AI0 chip is reset and enabled by manipulating the IO data lines D 15 and D 14. In doing so, the variable CFG in which the value to be output to the RADIO_GAIN port must be in external memory, i.e. page 8 or greater.

Immediately after resetting and enabling the AI0 chip, it must be configured to specify the AI0 gain, sync, filters and the values of RA/TA and RB/TB which set the sample conversion frequency. See “AI0 Port Programming, Setting the AI0 conversion frequency” [11.

Initialization of the DSP

The initialization of the DSP consists of initializing of any program values your program may use. For example:

```
DSP-IN1   ZAC          ;   zero A

          SACL  DO  ;   zero buffer
          SACL  SO  ;   zero buffer
```

Handling Interrupts

The transmit and receive interrupts, generated by the AI0 chip, must be handled by your code. This requires two functions RINT and XINT, pointed to by the interrupt vectors at the beginning of your program. Data received from the AI0 are stored in the variable BUFI and processed, in this example, by the function DSP. Data to be sent to the AI0 is stored in the variable BUFO and will be processed when a transmit interrupt occurs. TINT and TRAP interrupts can be handled in the same way.

Serial Port IO and Exiting to the Monitor
If your program is to read the DSP-93's Serial Port while executing, code is available that provides that capability. The function should be called from someplace in your code that is executed repeatedly. The character read is returned in the variable CHARREAD. If no character were available, ACC will be zero upon return. In any case, to be a user-friendly DSP-93 program, include this function always. If an upper- or lower-case R is sent to the DSP-93 Serial Port, the program will exit to the Monitor.

Wait Functions
There are two "wait" routines in the Monitor (See MONITOR.INC). There are also three "wait" functions that are commonly included in DSP-93 programs. These have delays of:

WAIT4	104 msec
WAIT2	52 msec
WAIT1	26 msec

Defining Data Tables
Tables of data such as filter coefficients, strings, etc. may be defined within your program.

Using Pre-defined Data Tables
There are two tables of waveforms provided with the DSP-93 source code. One table is the Sin/Cos table:

```
0..511 is Sin(theta), 512 words = 2 PI
64..639 is Cos(theta), 512 words = 2 PI

512 words long for each table, total
640 words long.  The table is scaled by
2Y1
```

the other is the general waveform table:

```
Table 0 = Sine(theta) scaled by 2Y5
Table 1 =  Triangle(theta)
Table 2 =  Square(theta)
Table 3 =  Sawtooth(theta)
```

The End
Don't forget the following directive at the end of your program.

```
          .END
; end of program
```

AI0 Port Programming

Setting the AI0 conversion frequency
The AI0 conversion frequency (sampling frequency) is set during the AI0 configuration. There are two values that determine the conversion frequency, TA and TB. The conversion frequency (in Hertz) is:

$$\frac{10,000,000}{2 \cdot TA \cdot TB}$$

The values are most easily set using the macros CMDAJAL and CMDB-VAL.

See "Linear Circuits; Data Conversion, DSP Analog Interface, and Video Interface; Data Book Volume 2" for more information on setting the conversion frequency. The AI0 chip has a specified upper limit on the conversion frequency of 19.2 kHz, but the chip will operate considerably in excess of this. For example, the 9600 bps FSK modems use a conversion frequency 41666 Hz. A table of conversion frequencies is provided in the programming guide [11.

IO Port Programming

Which Port does What
The TMS32OC25 has 16 IO ports, many of which are implemented in the DSP-93. Data is written to an I/O port with the OUT instruction and read with the IN instruction. For example:

```
        OUT     CHARREAD,UART-CTRL
        IN      CHARREAD,UART-READ
```

Debugging DSP-93 Programs

Debug functions in the Monitor
The Monitor ROM includes a debug routine developed by Tom McDermott, NSEG. This routine can be called as a development aid when you are generating new DSP code or if you are just studying existing code.

Debugging usin LOC.ASM
The code in L8C.ASM was developed for finding problems. It may be used to determine when program execution stops. An author of code which seems to be stopping mysteriously should

integrate this code into his program for testing. Then one of the users with a DSP-93 which hangs may run the modified program and report back with the results. Results can be reported by dumping memory in the DSP-93 using the Monitor. The memory dump can be captured and posted for evaluation by the author of the code. If we can determine when the problem is occurring the solution may also appear.

Monitor IO Routines

A collection of Monitor routines which should help speed code development is available for use.

Refer to the file MONITOR.INC included with the release disks for the exact location of these routines.

GET4HEX and GETVAL16
These two routines are the same. They collect a 16 bit hex value from the serial port. The value should be presented as four ASCII characters (0 through F digits). The four hex digits are converted to a sixteen bit HEX word and stored in a single memory location for later use. Look at MONITOR.INC for the storage location and a suggested label. This memory storage location is for the current page of memory established by the LDPK instruction.. The LDPK is normally set to page 8 with the instruction LDPK 8.

GET2HEX
This routine collects an 8 bit hex value from the serial port. The value should be presented as two ASCII characters (0 through F digits). The two hex digits are converted to an eight bit HEX word and stored in a single memory location for later use. Look at MONITOR.INC for the storage location and a suggested label. This memory storage location is for the current page of memory established by the LDPK instruction.. The LDPK is normally set to page 8 with the instruction LDPK 8.

GETCHAR
This routine collects a 4 bit hex value from the serial port. The value should be presented as one ASCII character (0 through F digits). The hex digit is converted to a four bit HEX word and stored in a single memory location for later use. Look at MONITOR.INC for the storage location and a suggested label. This memory storage location is for the current page of memory established by the LDPK instruction.. The LDPK is normally set to page 8 with the instruction LDPK 8.

HEXOUT
This routine assumes a single HEX digit is stored in the accumulator. This digit is converted to ASCII and sent to the serial port.

INBIT
This routinue retrieves incoming data from the serial port. Data is placed in the indirect address location pointed at by AR(ARP).

OUTBIT
This sends the lower 8 bits of AR(ARP) to the serial port. This routine uses memory location 0060h of the current page for temporary storage. Location 0060h is used to store the UART register information while the routine is sending the data. The register information from the UART is used to determine if CTS is set or cleared and if the transmit data buffer is full.

OUTBIT
Sends the lower 8 bits of location 0062h of the current page to the serial port. This routine uses memory location 0060h of the current page for temporary storage. Location 0060h is used to store the UART register information while the routine is sending the data. The register information from the UART is used to determine if CTS is set or cleared and if the transmit data buffer is full.

PRVAL08
The 8 bits in a defined memory location of the current page are converted to ASCII and sent to the serial port. MSB is sent first. See MONITORING for the proper memory location and label to use for this routine. This routine uses HEXOUT.

PRVAL16
The 16 bits in a defined memory location of the current page are converted to ASCII and sent to the serial port. MSB is sent first. See MONITOR.INC for the proper memory location and label to use for this routine. This routine uses HEXOUT.

RESET
Jump to this location to restart the Monitor.

SD CRLF
Sends a single carriage return and line feed to the serial port.

SD STR
Calling this routine with a 16 bit address stored in the current AR(ARP) register will cause a string to be sent to the serial port. The C-style string should begin at the AR(ARP) memory pointer and end with 00. Only the lower 8 bits of the memory words will be sent. The string data must be in data memory space.

SPl through SPlO
These routines will send the indicated number of spaces (20 Hex) to the serial port.

MWAITl and MWAIT A
The WAIT routines are shown here. You can enter the routine at MWAITl or MWAIT A with your own value set for AR2. With a 40.MHz clock, the wait from MWAITl will be about 32 milliseconds while the wait from MWAIT A will be about N mS where N is the value in AR2. Use MONITOR.INC for the entry locations for MWAITl or MWAIT-A.

MP BLK MV
Thi&outiie will move a block of data memory to program memory. You must provide pointers to the beginning and end of the block in program memory space and the beginning of the block in data memory space. This routine would be useful if you are dynamically creating program code in data memory and need to move it to program space.

PM BLK MV
This routine will move a block of program memory to data memory. You must provide pointers to the beginning and end of the block in program memory space and the beginning of the new block in data memory space. This routine would be useful if you want to include a sin table in your program. Your program would be loaded into the DSP-93 in what will become program memory. After starting your program you would use this routine to move the table to data memory. The routine may also be used to move string data from program memory to data memory for later use. The following example shows how the routine would be called.

Assembling a DSP-93 Program

How to assemble a file using TASM under DOS

If you are using DOS, the command line to assemble a TASM source file, say XXX.ASM, is: `TASM -3225 -la1 -go XXX.ASM`
This will result in a listing file XXX.LST being generated and an object file XXX.OBJ. For further options when using TASM, see the TASM documentation.

Windows and Macintosh Assembly

See the Help files for information on using D93WE or DSP-93Control as a code development environment.

References:

1. Parsons, Ron, W5RKN. (1995). Programming Guide for the DSP-93. , 1995. TAPR: Tucson, AZ.

2. Stricklin, Bob. (1994). TAPR/AMSAT Joint DSP Project: DSP-93. Proceedings of the TAPR 1994 Annual Meeting. Tucson Amateur Packet Radio Corp.

3. DSP-93: The TAPRIAMSAT Joint DSP Program. (1994). Issue #55, Summer 1994, Packet Status Register. pp. l-4. Tucson Amateur Packet Radio Corp.

4. McDermott, Tom, NSEG. (1995). D93WE Windows Development Environment for the TAPR/AMSAT DSP-93. Proceedin@ of the 1995 TAPR Annual Meeting. St Louis, MO. March, 1995. TAPR: Tucson, AZ. p. 31.

5. Parsons, Ron, WSRKN. (1995). DSP-93 Control Macintosh Development Environment for the TAPR/AMSAT DSP-93. Proceedings of the 1995 TAPR Annual Meeting. St Louis, MO. March, 1995. TAPR: Tucson, AZ. p. 38.